automation$_{s}niper Documentation$

Release latest

Feb 21, 2023

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 What is Automation-Sniper?

Automation-sniper is a library used to generate a performance framework from Open API Specs.

Nowadays API is commonly used in every service. So folk follows Open API specs to define the APIs.

Writing a performance script for each API is a really painful task.

That's why Automation Sniper came into the picture. You can generate a performance framework from Open API specs.

To start using Automation-Sniper, go to *Installation*

1.1.1 How Can Automation Sniper Tool Help You

Swagger Specs

If your team owns swaggers spec of version V2 or V3 then you can generate performance(python+locust) framework using automation-sniper Tool. The process is very simple. You just need the automation-sniper tool setup. Once this is done either you can pass the swagger doc link or swagger JSON/YML file to the tool. The tool will generate the framework for you within a few ms.

Postman Collection

```
`Inprogress -- In development`
```

1.1.2 Features

- **Convert swagger specs into locust framework**

  You can get a performance framework that supports python and locust from your swagger specs. So you don't need to write any code for your API. Looks Cool Right!!!

- **Support specs deprecation handling in the framework**

  It supports swagger specs deprecation handling in your framework. Deprecated API will have the deprecated annotation in your framework.

- **Support postman collection to locust framework**

  It supports the postman collection too. That means you can convert your all postman APIs into a performance framework.

- **Support regeneration of framework if specs are deprecated**

  Now the best feature you will like surely. It supports the regeneration of the framework. That means if tomorrow your specs change then you just need to upload the old framework and all-new specs will be overwritten into your framework.

** This is the Beauty of *No Code Project* **

1.1.3 Name & background

Automation-Sniper was born out of frustration. When I was writing a performance script for APIs, it was a really time taking process. Writing the wrapper of each API from the swagger spec is a pathetic task. Also tomorrow any specs change then also maintaining that is again another big task. As you know Automation is everywhere. So here Idea came to automate this pathetic process. Hence *No Code Project* Automation-Sniper came to the market as a warrior.

history

1.1.4 Authors

- Pankaj Kumar Nayak

1.1.5 Contributor

- Vibhore Jain

1.1.6 License

Open-source licensed under the Apache License 2.0 (see LICENSE file for details).

1.2 Installation

Install Python 3.6 or later, if you don't already have it.

Install automation-sniper:

```
$ pip3 install automation-sniper
```

Validate your installation:

Now it is time to *create your first framework from API Specs!*

Note: If you have any issues installing, check the stackoverflow for possible solutions.

1.2.1 Pre-release builds

If you need the latest and greatest version of automation-sniper and cannot wait for the next proper release, you can install a dev build like this:

```
$ pip3 install automation-sniper==dev
```

Pre-release builds are published every time a branch/PR is merged into master.

1.3 Getting started

Once you installed and verified the version of Automation-sniper then the next step is to familiar yourself with some basic commands of the tool.

```
$ automation-sniper -h
```

Output:

```
usage: automation-sniper [-h] [--version] -f FRAMEWORK_NAME -p PATH [-r RESULTS_PATH] [-
↪sp SCRIPT_PATH]
                         [-o {get,post,put,patch,delete,head,options,trace} [{get,post,put,
↪patch,delete,head,options,trace} ...]] [-v] [-ba BLACKLIST_API [BLACKLIST_API ...]]
                         [-wa WHITELIST_API [WHITELIST_API ...]]

This tool helps to convert API Specs into automation Framework

options:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -f FRAMEWORK_NAME, --framework_name FRAMEWORK_NAME
                        Provide the name of the framework
  -p PATH, --path PATH  Provide the path to the swagger file/postman file or provide␣
↪swagger api-docs url/postman collection url
  -r RESULTS_PATH, --results-path RESULTS_PATH
                        path to store generated automation framework default: result
  -sp SCRIPT_PATH, --script-path SCRIPT_PATH
                        provide script folder path as zip format
  -o {get,post,put,patch,delete,head,options,trace} [{get,post,put,patch,delete,head,
↪options,trace} ...], --operations {get,post,put,patch,delete,head,options,trace} [{get,
↪post,put,pat
ch,delete,head,options,trace} ...]
                        operations to use in API testing
  -v, --verbose         verbose
  -ba BLACKLIST_API [BLACKLIST_API ...], --blacklist-api BLACKLIST_API [BLACKLIST_API ...
↪]
                        tags to use in api testing
  -wa WHITELIST_API [WHITELIST_API ...], --whitelist-api WHITELIST_API [WHITELIST_API ...
↪]
                        tags to use in API testing

And that's how you'd like to use this tool. To know more please read the documentation␣
↪https://automation-sniper.readthedocs.io/
```

1.3.1 Basic Usages

To use automation-sniper you need either the Swagger docs link or swagger JSON.

CLI command to convert a swagger specs into NFR framework.

```
$ automation-sniper -f "TestFramework" -p "https://petstore.swagger.io/v2/swagger.json"
```

Output:

```
2022-03-03 19:33:03,711 - TRANSFORMATION_TOOL_LOG INFO     Got request for framework_
↪name TestFramework and upload_path \result
2022-03-03 19:33:05,483 - TRANSFORMATION_TOOL_LOG INFO     Framework is generated in␣
↪this path : \result\TestFramework
```

The framework is created under the result folder.

CHAPTER

TWO

UNDERSTANDING AUTOMATION-SNIPER

2.1 CLI Usages

This section explains about all CLI parameters.

2.1.1 Command Arguments Description

```
FRAMEWORK_NAME [-f]: The team name is the main/root folder for the framework. This␣
↪cannot be empty. eg. if your swagger name is Petstore
then you can give the framework name as "petstore".

$ automation-sniper -f "petstore"
```

```
PATH [-p]: This accepts either API specs link or specs file path as JSON/YAML. If your␣
↪swagger link is in private VPC then you can give
           JSON path so that you can generate the framework

$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json
```

```
RESULTS_PATH [-r]: This is the path where the generated framework will be present. If␣
↪you leave it empty then it will create a "result"
folder in your current directory and an output framework will be generated here.

$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json -r /
↪output
```

```
SCRIPT_PATH [-sp]: This is required when you need to regenerate the framework if any API␣
↪specs changed. It only accepts only zip
folders. So you need to zip your old framework and provide that.

$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json -sp /
↪result/petstore.zip
```

```
OPERATIONS [-o]: This is basically an operation filter. If you need to generate an NFR␣
↪framework only for "get" requests then you
need to pass "-o get" in the command. The multi request is also supported. e.g. if you␣
↪need both get and post request then you need to
pass "-o get post".
```

(continues on next page)

(continued from previous page)

```
Command: automation-sniper -f "<Team Name/ framework Name>" -p <API Spec Path> -r
↪<output path>

$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json -o get
$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json -o get␣
↪post
```

```
BLACKLIST_API [-ba]: This is also one type of blacklist filter. This is used when you␣
↪wanted to generate NFR framework for all API
specs except some APIs. In this case you need to pass that API path with this filter. So␣
↪that NFR framework will be generated with all
API except blacklisted API. You need to pass the command like this "-ba /pet/
↪findPetsByStatus".

$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json -ba /
↪pet/findPetsByStatus
$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json -ba /
↪pet/findPetsByStatus /pet/findPetsByStatus/name
```

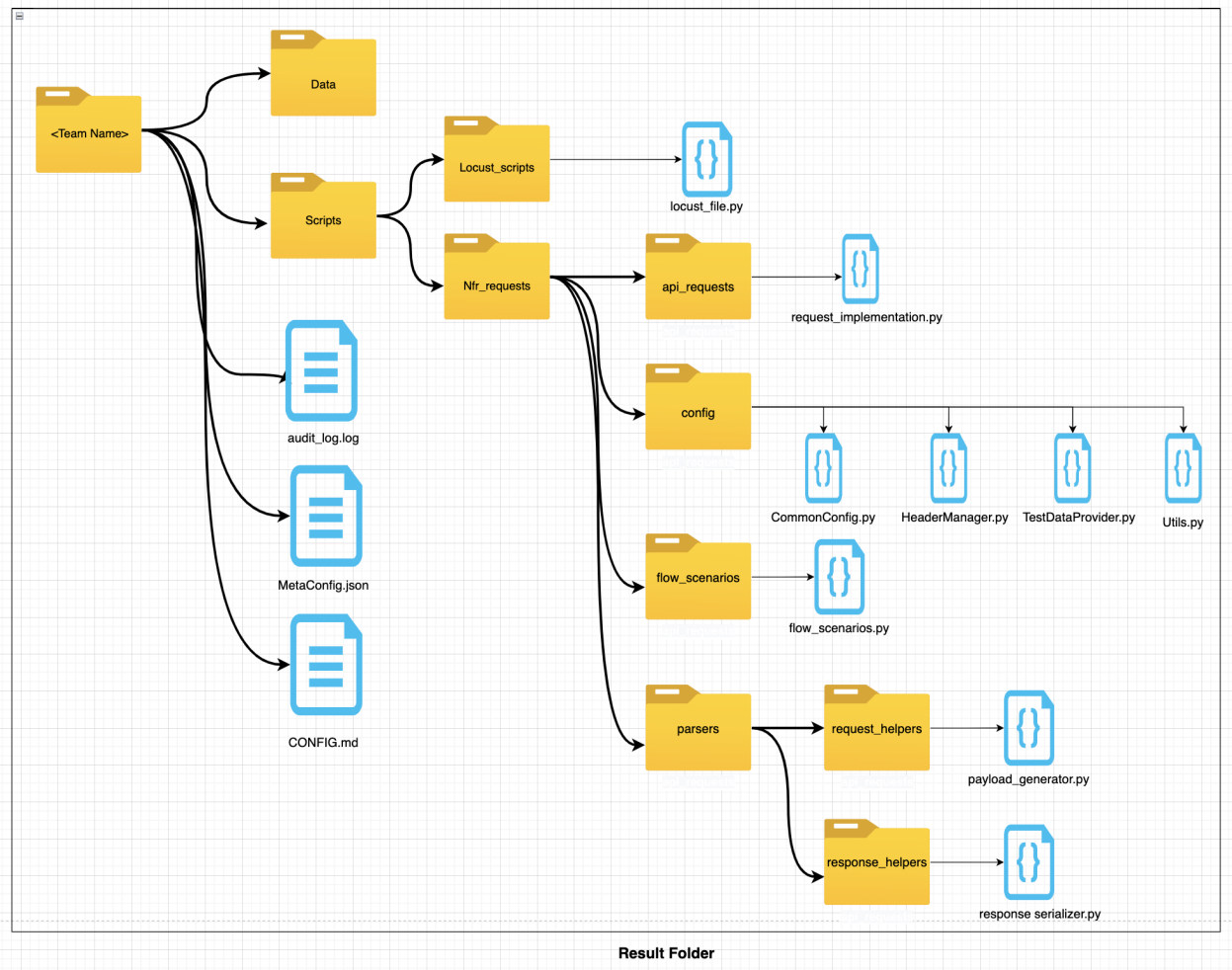
```
WHITELIST_API [-wa]: This is also one type of whitelist filter. This is used when you␣
↪wanted to generate NFR framework for only few
APIs from the specs. In this case, you need to pass that API path with this filter. So␣
↪that NFR framework will be generated for specific
API. You need to pass the command like this "-wa /pet/findPetsByStatus".

$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json -wa /
↪pet/findPetsByStatus
$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json -wa /
↪pet/findPetsByStatus /pet/findPetsByStatus/name
```

```
VERBOSE [-v]: This is used to print debug logs. If you are facing any issue then you␣
↪need to append this argument to see the debug log.

$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json -v
```

2.2 Framework Structure



2.2.1 Framework Folders and Files

`Data folder - This folder is used to hold all test data files used by the NFR script run. Use can create a test data file and put inside this folder`

`Scripts folder - This folder is used to hold nfr request folder and locust scripts folder.`

`Locust_script Folder - This folder is used to hold all locust files. All flow scenario is imported into the locust file.`

`Nfr_requests Folder - This folder is used to hold all supporting library folders used in the NFR framework.`

`api_requests Folder - This folder is used to hold all API implementation scripts. Which is further imported in the flow scenario script.`

`flow_scenarios Folder - This folder is used to hold all Flow scenarios of the API implementation. Which is further imported in the locust script.`

`config Folder - This folder is used to hold all config scripts like headermanager, commonconfig, etc.`

`CommonConfig.py - This file contains common static config details like host, etc used across the framework.`

`HeaderManager.py - This file contains the Header manager class which generate/manipulate headers used in API request.`

`TestDataProvider.py - This file contains all functions to read data files belonging to the data folder.`

`Utils.py - This file contains all utility functions used in the framework.`

`parser Folder - This folder is used to hold all scripts for request payload serialization and response serialization.`

`request_helpers Folder - This folder is used to hold all request payload generator scripts. Where a user has to update payload data that is going to be used in the framework run.`

`response_helpers Folder - This folder is used to hold all response sterilization scripts.`

`audit_log.log File - This file holds all audit logs of the script generation.`

`MetaConfig.json File - This file holds all data-related swagger specs and function names with file path information. This is used to generate a framework if a new spec is released.`

`CONFIG.md File - This file is used by Flash. It contains python version details only.`

2.3 Custom Data Support

2.3.1 How To Add Custom Header

2.3.2 Custom Header Support

Once the framework is generated then there will be a file HeaderManager.py present inside the config folder.

- Open that file and you can see one code block i.e.

```
CUSTOM_HEADER = {}
```

- You can add your header name: header value inside this dictionary object. e.g. I need to add a custom header “perf-test”: “true”.

You just need to follow the below procedure.

```
CUSTOM_HEADER = {
"perf-test": "true"
}
```

2.3.3 Auth Header Support

- Once the framework is generated then there will be a file HeaderManager.py present inside the config folder.
- For auth header, you need to implement the logic to fetch auth token if your API use Oauth2.

```
import os
import json
"""
Please add custom header in CUSTOM_HEADER dict
"""
CUSTOM_HEADER = {
#"perf-test": "true"
}
AUTH_HEADER = {'apiKey': 'api_key', 'oauth2': 'Bearer'}
class HeaderManager:
    """
    This is a Header manager class which handle global and custom header support.
    """
    def __init__(self):
        """
        Constructor for HeaderManager to initialize empty header
        """
        self.header = {}

    def initialize_header(self):
        """
        This method is responsible to call all header initializer.
        :return: updated header dict
        """
        self.custom_header_initializer()
        self.auth_header_initializer()
        return self.header

    def custom_header_initializer(self):
        """
        This method is responsible to handle custom header given by user.
        :return: updated header dict
        """
        return self.header.update(CUSTOM_HEADER)

    def auth_header_initializer(self):
        """
        This method is responsible to handlde auth header
        :return: updated header dict
        """
        auth_header = {}
        #fixme: Please select one auth and comment remaining
        auth_header = {AUTH_HEADER['apiKey']: self._get_auth_header()}
        auth_header = {'Authorization': "Bearer {}".format(self._get_auth_header())}
        return self.header.update(auth_header)

    @staticmethod
    def _get_auth_header():
```

(continues on next page)

(continued from previous page)

```
        """
        This method handle token generation for auth header handler.
        Note: Please write auth generation logic here.
        :return: token as string
        """
        #fixme: Please add your logic to get token
        token = "API Token"
        return token
```

- You need to implement the auth token fetch logic inside *_get_auth_header()* function

2.3.4 How to Configure Payload In The Framework

If your API uses various data payloads or params then you need to provide that data in the framework. For that, you need to open "parsers/request_helpers/" folder. You can see each API path corresponding payload helper script present inside the folder.

```
def delete_store_store_order_orderid_(self):
    path_headers = {'Content-Type': 'application/json'}
    headers = path_headers.update(self.header)
    path_payload = {'orderId': None}
    request_path = self.helper_object.url_parser(path_payload, "/store/order/{orderId}")
    self.url = self.host+request_path
    response = self.locust_object.client.delete(self.url, headers=headers, name="delete_
↪store_store_order_orderid_")
```

- You need to update "orderId" with your actual value instead of None. You can parameterize this too. So that value will be fetched from sample data present inside the data folder.
- Similarly each payloads/params you need to update.

2.4 Run The Framework

2.4.1 Running It On Local

If you want to run this on local then you need to just type below command inside the *scripts* folder.

```
#TODO:METHOD_IMPLEMENTATION_TAG [DONT DELETE]
```

```
$ pwd
/result/petstore/scripts
$ locust -f locust_scripts/petstore_locust_file.py
```

2.4.2 Don't Do

- Please don't delete "audit_log.log" and "MetaConfig.json" files. Else you cant regenerate the framework if any API spec is changed in the API specs document.
- Please don't delete any identification [#TODO:METHOD_IMPLEMENTATION_TAG [DONT DELETE]] tag present inside any scripts. Else you cant regenerate the framework if any API spec is changed in the API specs document.
- If you have changed any function/method name in the API request folder or anywhere please update that in the "MetaConfig.json" file. Else you cant regenerate the framework if any API spec is changed in the API specs document.

CHAPTER

THREE

OTHER FUNCTIONALITIES

3.1 Debugging

- You can check the warning and debug yourself what's wrong in Specs.

```
"WARNING Empty tag received from payload"

This warning tells that you haven't set up swagger specs properly. Each path of Swagger␣
↪specs contains one tag. If it's not
there it is difficult to generate a framework. e.g. "tags": ["Tag_Name"]. Please follow␣
↪swagger RFC for more details https://swagg
er.io/docs/specification/2-0/grouping-operations-with-tags/. Once you fixed your swagger␣
↪then rerun the tool again.
```

```
"Swagger spec doesn't have a deprecated tag. Please check and fix it"

This warning tells that you haven't set up swagger specs properly. Each path of Swagger␣
↪specs contains one key i.e.
"deprecated". It actually tells the user if that API is alive or deprecated. If it's not␣
↪there we will generate the framework but it's
better to have it. Please follow swagger RFC for more details https://swagger.io/docs/
↪specification/2-0/grouping-operationswith-tags/ on section "Marking as Deprecated".␣
↪Once you fixed your swagger then rerun the tool again.
```

3.2 Regenerate Framework

3.2.1 Why We Need This

- Once you have generated the framework I think your work is done. But API specs are not static ones. From time to time new API will be added and the old API will be deprecated.
- In this case, you need to again implement the new API request method inside the framework. It is again time-consuming.
- To solve this problem you need to regenerate the framework. In this case, you just need to give the old framework path and a new swagger/postman API spec link.
- Now the tool will find out new API details as well as deprecated API details and write a function for that.
- You will have your old framework with an all-new API implementation inside the script. So that you don't need to write code for API implementation.

3.2.2 Procedure For Regeneration

There will be some string Tag Inside all API request scripts i.e.

` #TODO:METHOD_IMPLEMENTATION_TAG [DONT DELETE] `

- Please don't delete this Tag. Else You won't get a new API function.
- You need to zip your old framework.
- You need to provide the old framework root folder name as framework name.

```
$ automation-sniper -f "petstore" -p https://petstore.swagger.io/v2/swagger.json -sp /
↪result/petstore.zip
```

- Once you hit this command you will get your framework with a new API function

3.3 Logging

```
$ automation-sniper -v
```

Output:

You can see the debug data using above command.

3.4 Postman Collection To Framework

` Under Development `

CHAPTER

FOUR

FURTHER READING / KNOWLEDGEBASE

4.1 More About Swagger Details

The (OAS) is a widely adopted, standardized format for describing REST APIs. You can use OpenAPI to detail every part of your API, including endpoints, operation parameters, request responses, and authentication flows. The OpenAPI format is easy for both developers and machines to read and understand. What's harder to immediately decipher is how the versions differ.

At the time of publication, almost everyone will likely want to use OpenAPI v3.0, the latest official major version that was released in 2017.

Some folks are on OpenAPI 2.0 due to being stuck there with tooling that still has not added support for OpenAPI v3.0. Some people might talk about OpenAPI v3.1, but whilst it is still in the "Release Candidate" stage there is essentially no tooling support for it.

However, for those digging into the nuances between versions, it's worth looking into the details.

There are currently two major OpenAPI releases, 2.0 and 3.0 (the latest release is 3.0.3). Version 3.1 was on June 18, 2020, as an initial release candidate. While it takes some important steps forward, it adds to the confusion over which version to use. In this post, we'll look at some of the key differences between OpenAPI 2.0, 3.0, and 3.1.

Differences Between OpenAPI 2.0 and 3.0 The two major versions of OpenAPI have the most significant differences, which come from their history. OpenAPI 2.0 was previously known as Swagger and is intended to replace it with backward compatibility. Once adopted as an open format, the community began working on OpenAPI 3.0, released in 2017. Let's highlight some of the significant changes made to OpenAPI components in version 3.0.

CHAPTER

FIVE

CHANGELOG

Changelog

All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](http://keepachangelog.com/en/1.0.0/) and this project adheres to [Semantic Versioning](http://semver.org/spec/v2.0.0.html).

<!– insertion marker –> ## [0.1.0](https://github.com/pankajnayak1994/automation_sniper/releases/tag/v2.4.20.1.0) - 2023-02-21

<small>[Compare with first commit](https://github.com/pankajnayak1994/automation_sniper/compare/523671ff37e848e55214cf49361a