Allure System Health Lab Setup

Catching bugs early using automated testing!

Frinze Erin Lapuz

Table of contents

1. Allure Setup

Allure is a test reporting server. More information here

1.1 Allure Docker Service

The Allure Docker service is a service that incorporates Allure test reporting into an interface and API that can be used test reporting from CI/CD while presenting it in such a way that enables administrators for some extra functionalities as well as to showcase the test results on viewers.

1.2 System Health Lab Setup

This is a setup customised for the System Health Lab, and serves as a source control for the setup. For a more general setup, please see the original documentation for Allure Docker Service linked at the top.

1.3 More

Read more information here or in raw format.

2. Setup Overview

To be able to use the Allure Reporting Server in the System Health Lab. The following instructions will need to be executed below.

> **Examples**
>
> There are examples on this repository that outlines step 2-4 in a language dependent. With the examples, you only need to focus on 2-3 since step 4 can just be copy pasted (with tiny setup change). Essentially with the examples, you can copy it to your own repository and change the tests and the source files that you are testing, .

2.1 1. Setup a Allure Project

This is done by an administrator that have access to the Allure Reporting Server.

1. Login as an administrator using credential
2. At the top-right corner, press the button that signifies addition and create a project

> **Credentials**
>
> The credentials that are entered here will need to be known for the Upload step

2.2 2. Create your own tests (in your own repository)

This will need to be done by the developer/engineer that will be using this test reporting server. See examples in this repository.

2.3 3. Setup the Testing Folder Structure (in your own repository)

This is important for code readability, and that is by splitting the tests from the source files. See examples in this repository how this is done.

2.4 4. Setup the Continuous Integration Pipeline (in your own repository)

This is dependent on how your CI pipeline will work, but usually the steps are:

1. Install dependencies
2. Run the Test
3. Upload the test

2.4.1 Uploading tests

This is done using `upload_scripts` in this repository. If you take a good look at the `.github/workflows/test.yml`, this is an example of file that you will just copy, and no configuration that you need to do aside from changing the `PROJECT_NAME`.

Environment Variables

If your repository does not belong in the `uwasystemhealthlab` organisation, you will have to put the environment variables:

```
ALLURE_USER=<CHANGE THIS>
ALLURE_PASSWORD=<CHANGE THIS>
ALLURE_API_SERVER=<CHANGE THIS>
```

Otherwise, no extra step that needs to be done.

3. Example

3.1 Python

Python is a very common language used in the System Health Lab. This documentation is to explain how the example works in this repository.

3.1.1 Installation of Packages

You will want to install packages to run automated-testing, but as well as to run all the tests in your local machine.

Create Environment

Create your own environment to isolate the packages. Refer to more information here.

> **Conda Create new Environment**
>
> In my local environment, I have done the following:
>
> ```
> conda create --name test
> ```
>
> where `test` is the name of my environment

Install Packages

Just in case you are unfamiliar of this, just type:

```
pip install -r requirements.txt
```

3.1.2 Running your tests locally

Just type

```
py.test
```

or

```
pytest
```

This will run all your test files that have prefix "test_"

> **What is Pytest?**
>
> Pytest is justs python test runner. There are lot of them, but I chose this for this example.

3.1.3 Most common usecase

1. Copy paste the example to your own repo
2. Replace `/src` files with your own source files and create your own tests and put them in `/tests` with prefix `test_`
3. Go to the `.github/workflows/test.yml` and change the `PROJECT_NAME` to your project name (make sure to ask do step 1 of `Setup Overview` first)

4. Commit

After that setup, everytime you do a git commit in the repository, the CI pipeline runs and tests your code and emails you (through your github email) when problems arises. You also get your reports at https://allure.systemhealthlab.com/.

3.1.4 Files and Folders Explanation

Source Files `/src`

These are files that you use as part of your software/reports that you want to keep all together. In my example, I have the following files:

```
__init__.py
addition.py
```

??? note "What is `__init__.py?` This file enables python to search for multi-directory files. This is helpful if you want to move your scripts in different places. Note that by doing this, the imports will all be relative to where you are running:

```
python script.py
```

Test Files `/tests`

This contains all your tests. It is **very important** that it has a prefix "test_" so that it could be detected as a test script. You could separate it to multiple files and multiple functions within each file as you please.

> **Import Path**
>
> Note that the syntax
>
> ```
> from src import addition
> ```
>
> will only apply with respect to the execution path. For example when I run my test file in the root directory:
>
> ```
> python tests/test_addition.py
> ```

Testing Configuration `pytest.ini`

This is just a configuration file on how to run the test. Customise it as you please, but most cases you don't need to. Refer to more information here.

Upload Script `send_and_generate.py`

This is just a script that sends the `test_results` folder upon running tests

Git Ignore `.gitignore`

This is just a file that specifies all files/folders git will ignore. The test runner usually creates caches and results that are not usually uploaded in source controls, but uploaded somewhere such as the Allure Reporting Server.

CI Pipeline `.github/workflows/test.yml`

This is a github action file that executes codes on a specific event. Change this as you please, but I have set it up for the most common use case in the System Health Lab (pushing on master). If you look at the documentation for `Setup Overview` then you would be familiar with the high-level step of this.

Requirements `requirements.txt`

This is just a file that contains dependencies in the test runner setup.