Quote Generator

QA-SFIA Project 2

This is the outcome of the first personal SFIA-2 project for the QA Academy - Week 9, as specified on:

https://portal.qa-community.co.uk/~/devops/projects/practical--devops

Project resources:

GitHub : https://github.com/Zhal73/SFIA_PROJECT_2_Quote_generator.git

Jira board : https://domenico-gagliano.atlassian.net/jira/software/projects/QGS/boards/8

DockerHub : https://hub.docker.com/u/zhal73

Index

Project Brief

Create a multi-services application consisting in a front-end service (service 1) that communicates with with three other services which in form of API will provide information that will be presented by the front-end service.

Requirements

The basic requirement for the application are specified on the assessment text as follow:

- Service-oriented architecture with at least 4 services that work together
- Project tracking
- Use of Ansible playbook
- Use of a reverse proxy
- Code changes managed via Webhook
- Project deployed using containerisation and orchestration tools

- Produce Documentation
- Produce Risk assessment
- Test Implementation
- Version Control
- CI server

General Approach

This web application is designed to generate a random quote from a famous person. It consists in four different service each with a different function:

- Service 1 : Front-end Service. It requests an author to service 2 and a quote to service 3, the it will pass these information to service 4 to check the genuinity of the quote. Then it displays everything in an appropriate page and stores the information in a database.
- Service 2 : Back-end service. It returns to service 1 the name of a famous person that will be used as author of the quote.
- Service 3 : Back-end service. It returns to service one a sentence representing a famous quote.
- Service 4 : Back-end service. It takes the information from service 1, it checks if the author matches the quote, using a Python dictionary and returns a sentence that judge the quality or the quote, in particular if the quote belongs to the author, it returns *"Awesome!!! This is a pearl of wisdom"*, otherwise it returns *"Not quite! But is still a good one!!!"*

Extended Requirements

Once I decided the general structure of the application I extended its requirements, from a user perspective, so I added the following requirements:

- Read the quotes previously generate
- Add a new author to the list of authors
- Add a new quote to the list of quotes
- Add the user own quotes
- Delete an author from the list of authors
- Delete a quote from the list of quotes
- Good looking visual aspect

This brought me to skim those requirement so that the minimum viable product (MVP) could be delivered by the deadline. To do so I performed a MoSCoW analysis as follow:

Must have:

- Service-oriented architecture with at least 4 services that work together
- Project tracking
- Use of Ansible playbook
- Use of a reverse proxy
- Code changes managed via Webhook
- Project deployed using containerisation and orchestration tools
- Produce Documentation
- Produce Risk assessment
- Test Implementation
- Version Control
- CI server

Should have:

- Read the quotes previously generate

Could have:

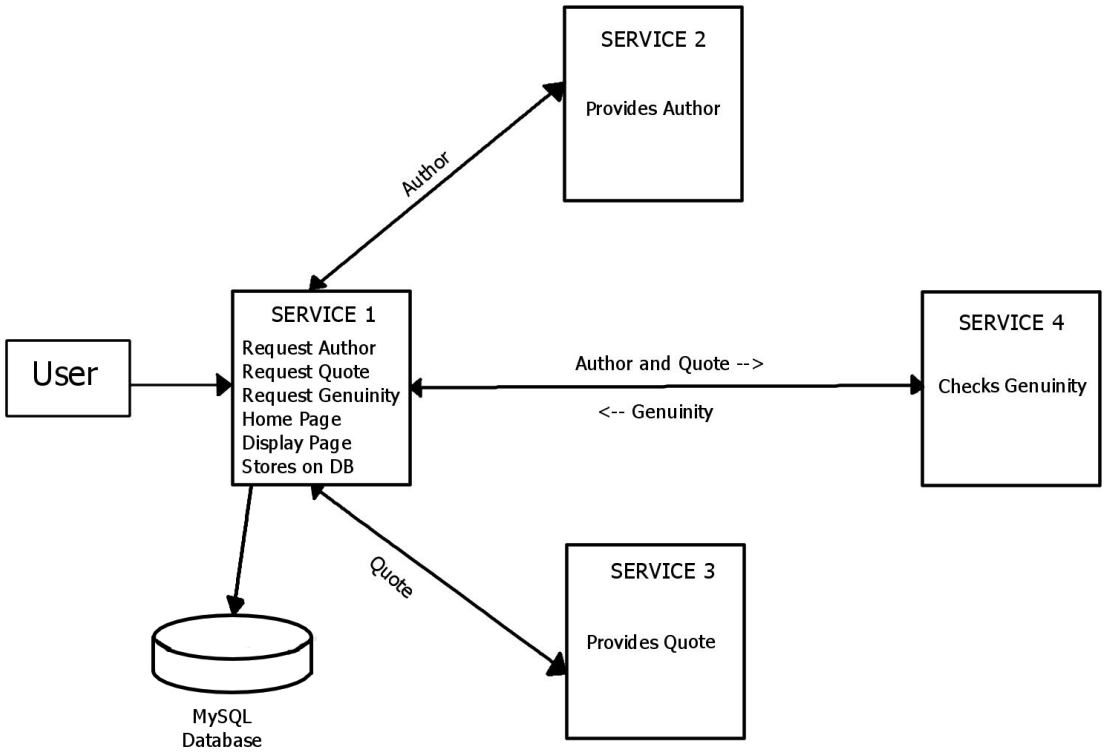
- Add a new author to the list of authors
- Add a new quote to the list of quotes
- Add the user own quotes
- Delete an author from the list of authors
- Delete a quote from the list of quotes

Wont have:

- Good looking visual aspect

Architecture

The four services are small Flask application written in Python. The following diagram represent the architecture of the application with the interaction between the services:



The application is implemented in a number of virtual machines hosted by Google Cloud Platform. In all of them has been installed the containerisation tool Docker, so that the four services can run in different containers. Furthermore the machines forms a Docker Swarm network, so that it is possible to deploy different replicas of the various services across the different VM. By doing so, is possible to improve the reliability of the application, because if some of the service's replica experience a problem, the Swarm orchestrator sent the traffic to the others available. The use of an orchestration tolls such as Docker Swarm, helps by balancing the traffic amongst the various available replica, so that no one particular machine should be overloaded.

The application also contains and Nginx web server that act as reverse proxy, this allows the user to reach the application through the standard http port 80, instead of the Flask port 5000.

The database is also containerised with docker and deployed with multiple replicas as for the services forming the main application. The database used consist in a single table as here represented:

all_quotes	
PK	ID
	Author Quote Genuinity

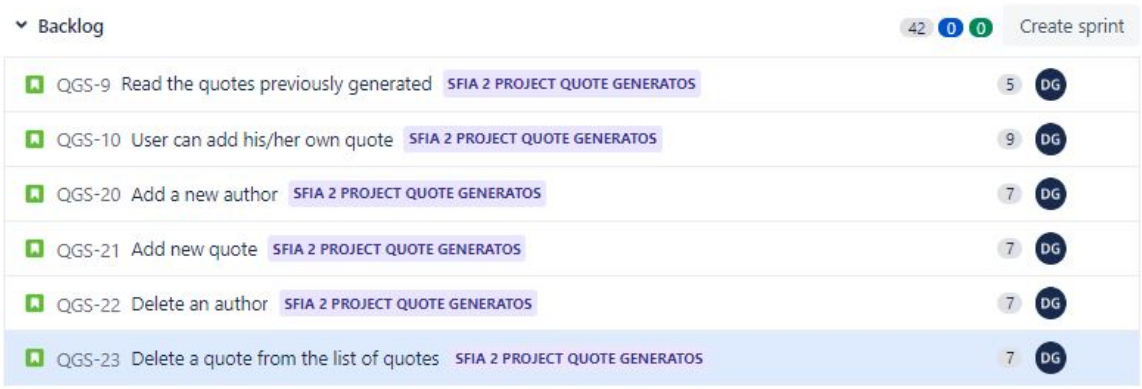
Project Tracking

To track the development cycle of this web application I used a Jira board which could is available at:

Jira board

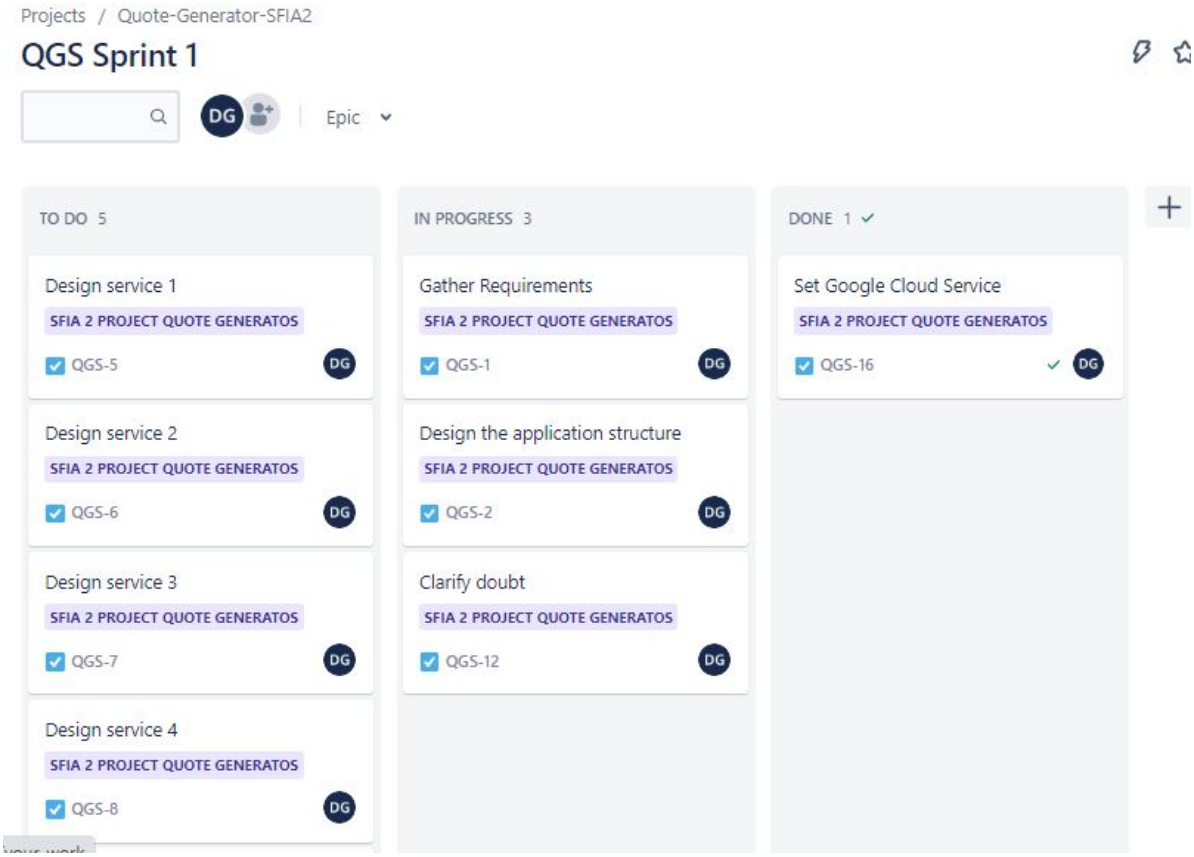
From the requirements I created a number of user's stories, then I broke up the stories into tasks and I estimated the amount of work needed to complete each story. This allowed me to assign to each story and amount of story points. I gave a total of 5 point to the story with less amount of work required, then I assign an higher score to the more complex ones.

The following images illustrates the stories I created and their score:



I development of the application follower the QA-Academy timeline in terms of topics, so I developed different aspect of the application once I learned the relevant topic. So I had to adapt my Sprints to cover these aspect, and to fit within the time window I had for the project, which was three weeks. So I covered the project development with a total of two Sprints.

the following images illustrates a Sprint one:



Risk Assessment

As any project, problem may happen that can compromise the completion of the project. As I started to work at the project I wrote an initial Risk assessment.

To evaluate the various aspect of the risks I produced a grade scale for the Likelihood , the Impact and the Tolerance, the various grade are as follow:

- **Likelihood**
    - High : the risk is likely to append because the source is technically capable, or the user is likely to make mistake in using the application.
    - Medium: the risk could happen, but controls are in place to mitigate it.
    - Low: the risk is unlikely to happened
- **Impact**
    - High : the risk occurrence can seriously lead to site malfunction or inaccessibility.
    - Medium: the risk occurrence doesn't lead to major malfunctions or loss of data.
    - Low: consequences of the risk occurrence can be recovered because effective procedure are in place
- **Risk Tolerance**
    - Tolerate: No further action taken, because the risk's consequences are manageable.
    - Treat: Action taken to minimise the risk's consequences.

The **initial risk assessment** can be found here, and is represented below:

Threat	Description	Likelihood Level	Impact Level	Responsibility	Current Mitigation	Proposed Mitigation	Response	Tolerance
SQL database Inaccessible	The MySQL database hosted by Google Cloud Platform cold become inaccessible, then the web application would become unusable	Low	Low	Google Cloud Platform	NONE	Set up backup procedures for the database, either by using dedicated VM or in local computer. So the DB could be clone into another instance or moved to another provider	Try to restart the database, if GCP is accessible otherwise contact GCP	tolerate
The VM hasting the web server become inaccessible	As the database, the webserver for the application is running in a Virtual machine hosted by GCP. If the VM become inaccessible, the website will not function	Low	Low	Google Cloud Platform	NONE	Set a backup VM so it can be run as soon as possible, either with GCP or another provider	Try to restart the database, if GCP is accessible otherwise contact GCP	tolerate
Poor time management	I might not be able to manage the tight time window for the project, because I may need to revise concept I studied in the academy and I could not be able to complete the procject effectivelly	Medium	High	Domenico Gagliano	Clarify any doubt during the lecture, and ask for help when I recognise I am stuck with something	keep an eye on the project tracking tool, make sure at the end of each week everything is clear, otherwise ask the trainer and/or other member of the cohort	Follow the time allocated on the project tracking tool	Treat
Covid-19	The infection is still around, so it would be possible for me to catch the infection, so I won't be able to finish the project (at least before the due date)	Low	High	Domenico Gagliano	Use face cover and respect social distance when in contact with people	Follow government directives as they regularly change	None	Treat

Once the project move forward in its development I revised it and I produced a final risk assessment. As for the initial risk assessment I used a grade scale for the Likelihood , the Impact and the Tolerance with the same grades previously defined.

The final risk assessment can be found here, and is represented below:

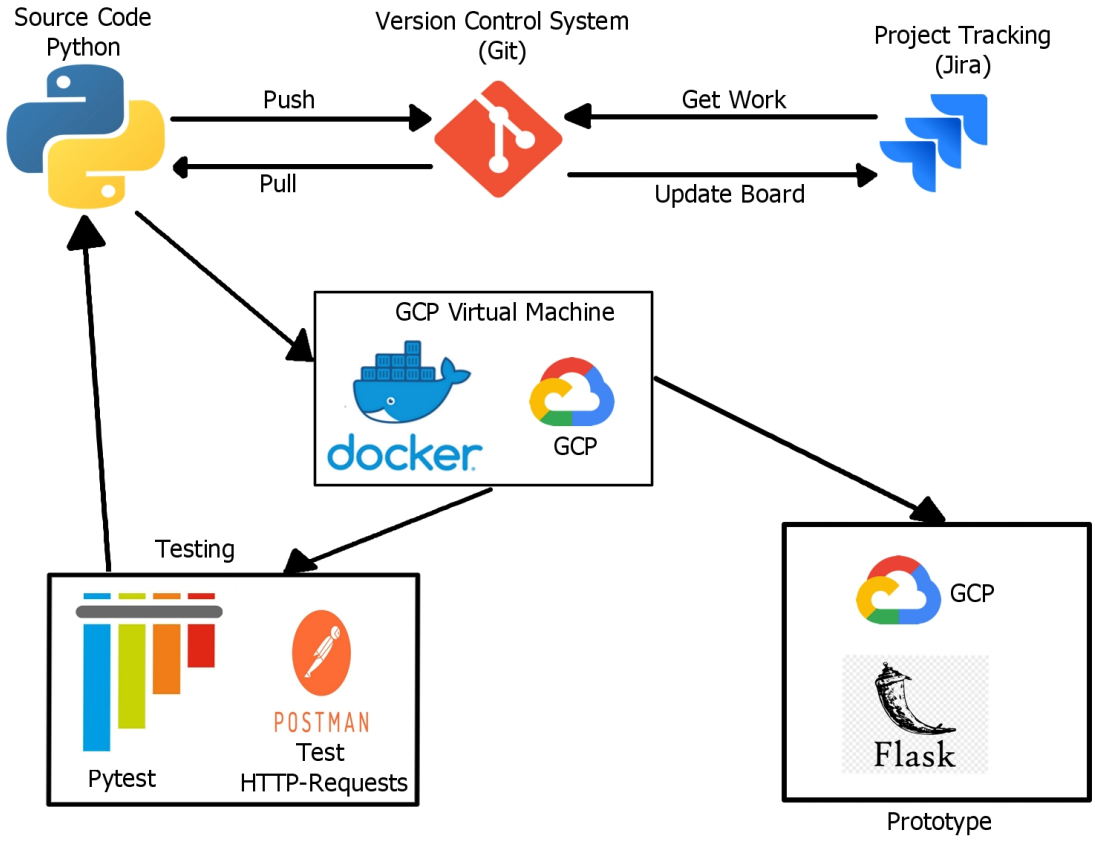
Threat	Description	Likelihood Level	Impact Level	Responsibility	Current Mitigation	Proposed Mitigation	Response	Tolerance
The VMs hosting the web server become inaccessible	As the database, the webserver for the application is running in a Virtual machine hosted by GCP. If the VM become inaccessible, the website will not function	Low	Low	Google Cloud Platform	NONE	Set a backup VM so it can be run as soon as possible, either with GCP or another provider	Try to restart the database, if GCP is accessible otherwise contact GCP	tolerate
Jenkis Pipeline Fails	Jenkis pipeline fails any of its stages, or webhoos used to track code changes don't work properly.	Low	High	Domenico Gagliano	Check if the VM hosting the jenkins server changes ip address and update the webhook appropriately. Check the scripts included in the Pipeline	Continue to monitor IP address changes and scripts efficiency	Update Webhoock	Treat
Docker Hub repository Down	As the buit images are pushed to the Docker-Hub site, if the site goes down the the docker swarm will not be able to store and/or retrieve those images.	Low	High	Docker Hub	None	Implement a local registry for the images	None	tolerate
DDOS attack	Distributed denial-of-service attack is an attempt to disrupt normal traffic to the server or services targeted by flooding it with internet traffic	Low	High	Domenico Gagliano	None	Set Firewall, VPN and install mitigation tools against DDOS attack	None	tolerate
HTTP used instead of HTTPS	The connection to and from the website use HTTP protocol, so data is visible by any potential attacker.	High	Low	Domenico Gagliano	NONE	Implement HTTPS connection which uses encryption to move information.	Potentially we are unaware if malicious entity have monitored our website traffic. If we found any change or inconsistency in the web application contents we can use backups to restore consistency	tolerate

Threat	Description	Likelihood Level	Impact Level	Responsibility	Current Mitigation	Proposed Mitigation	Response	Tolerance
Covid-19	The infection is still around, so it would be possible for me to catch the infection, so I won't be able to finish the project (at least before the due date)	Low	High	Domenico Gagliano	Use face cover and respect social distance when in contact with people	Follow government directives as they regularly change	None	Treat

CI Pipeline

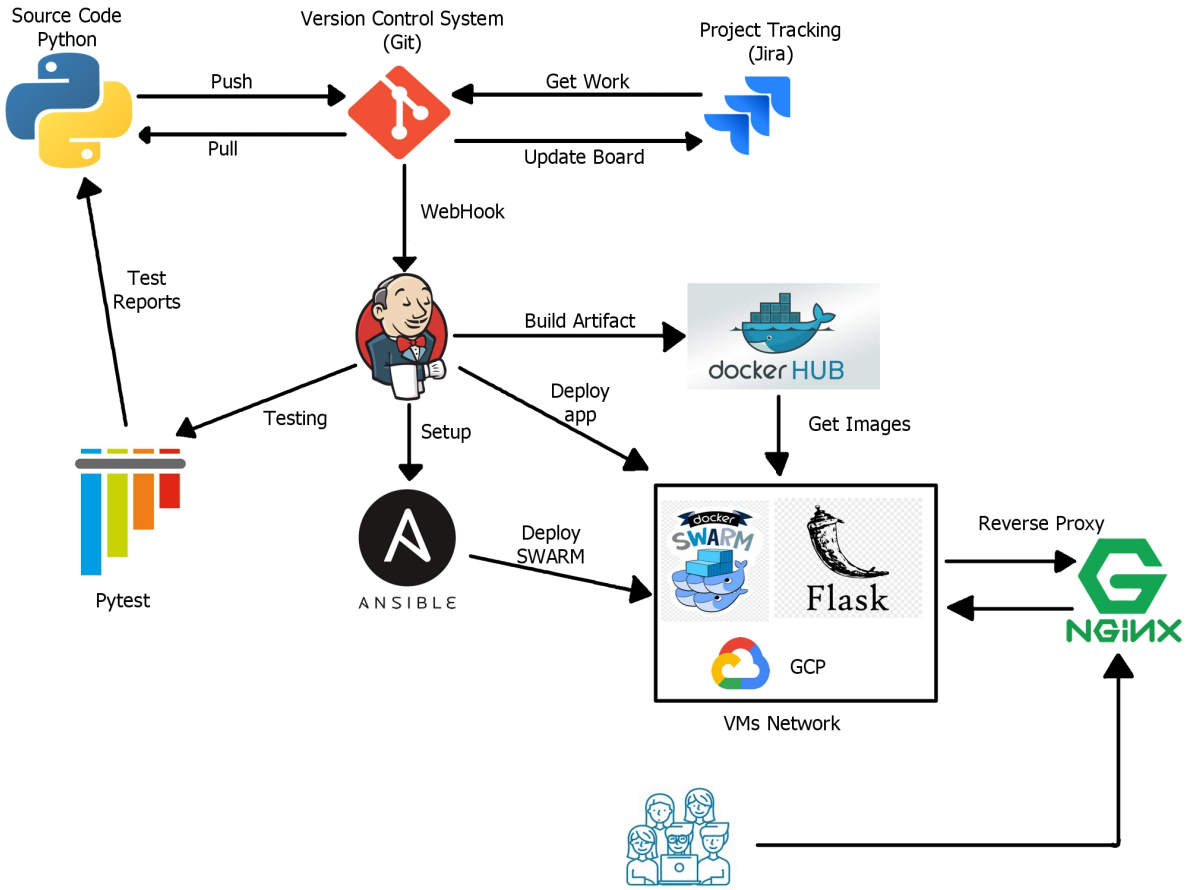
As mentioned before, the project development followed the path given by the QA academy regarding the relevant topics studied each week. Because of that the CI pipeline evolved and more tools were introduced gradually.

The first stage of the development, even if it is not a proper CI pipeline , could be represented as follow:



Once more tools were introduced I was able to change the development pipeline into a proper CI pipeline by using Ansible (Configuration tools), Docker SWARM (Orchestration Tool) and Nginx (web server as reverse proxy) and ,of course Jenkins (CI/CD server).

So the final CI pipeline became:



Testing

The application tests have been done by using the Python modules pytest.

The initial situation for the four services, before unit testing was:

service 1

```
----------- coverage: platform linux, python 3.6.9-final-0 -----------
Name                        Stmts   Miss  Cover   Missing
---------------------------------------------------------
application/__init__.py         9      0   100%
application/models.py           6      0   100%
application/routes.py          16      9    44%   13, 17-26
---------------------------------------------------------
TOTAL                          31      9    71%
```

service 2

```
-- Docs: https://docs.pytest.org/en/stable/warnings.html

----------- coverage: platform linux, python 3.6.9-final-0 -----------
Name                        Stmts   Miss  Cover   Missing
---------------------------------------------------------
application/__init__.py         4      0   100%
application/routes.py           8      3    62%   13-15
---------------------------------------------------------
TOTAL                          12      3    75%
Coverage HTML written to dir htmlcov
```

service 3

```
-- Docs: https://docs.pytest.org/en/stable/warnings.html

---------- coverage: platform linux, python 3.6.9-final-0 -----
Name                        Stmts   Miss  Cover   Missing
---------------------------------------------------------
application/__init__.py         4      0   100%
application/routes.py           8      3    62%   13-28
---------------------------------------------------------
TOTAL                          12      3    75%
```

service 4

```
-- Docs: https://docs.pytest.org/en/stable/warnings.html

---------- coverage: platform linux, python 3.6.9-final-0 ----
Name                        Stmts   Miss  Cover   Missing
---------------------------------------------------------
application/__init__.py         4      0   100%
application/routes.py          15     10    33%   13-44
---------------------------------------------------------
TOTAL                          19     10    47%
```

My approach was to inspect the lines highlighted on the test report and write specific tests for each block of codes that needed to be tested. Because the services communicate between them, and during the unit test they are tested independently, I needed to mock the http requests present in the different services.

In particular service 1, before rendering the page that contains the quote, requests and author, a quote and the genuinity to the other services, by using HHTTP requests. To mock these requests I used the Mocker method from the requests_mock library, this allows me to pretend that the other service returned their response, so that service 1 could produce the generate quote page.

This particular test is illustrated below:

```
def test_generate_quote(self):
        with self.client:
            with requests_mock.Mocker() as m:
                m.get('http://service2:5001/get_author', text='Albert Einstein')
                m.get('http://service3:5002/get_quote', text='In the middle of
difficulty lies opportunity.')
                m.post('http://service4:5003/get_genuinity', text='Awesome!!!
This is a pearl of wisdom')
                resp = self.client.get(url_for('generate_quote'))
                self.assertEqual(resp.status_code, 200)
```

Here is possible to see that the Mocker method simulates two get request to service2 and service 3, a post request to service4. It also simulates the returned text from the three services.

In addition, because the database used for the application is built in a docker container, for test purpose I had to set up and external MySQL test database hosted by Google Cloud Platform.

After I run all the unit tests, I was able to have 100% coverage for all service and illustrated below:

service 1

```
-- Docs: https://docs.pytest.org/en/stable/warnings.html

---------- coverage: platform linux, python 3.6.9-final-0 ----------
Name                      Stmts   Miss  Cover   Missing
-------------------------------------------------------
application/__init__.py       9      0   100%
application/models.py         6      0   100%
application/routes.py        16      0   100%
-------------------------------------------------------
TOTAL                        31      0   100%
```

service 2

```
-- Docs: https://docs.pytest.org/en/stable/warnings.html

---------- coverage: platform linux, python 3.6.9-final-0 ----------
Name                      Stmts   Miss  Cover   Missing
-------------------------------------------------------
application/__init__.py       4      0   100%
application/routes.py         8      0   100%
-------------------------------------------------------
TOTAL                        12      0   100%
```

service 3

```
-- Docs: https://docs.pytest.org/en/stable/warnings.html

---------- coverage: platform linux, python 3.6.9-final-0 -
Name                      Stmts   Miss  Cover   Missing
-------------------------------------------------------
application/__init__.py       4      0   100%
application/routes.py         8      0   100%
-------------------------------------------------------
TOTAL                        12      0   100%
```

service 4

```
-- Docs: https://docs.pytest.org/en/stable/warnings.html

---------- coverage: platform linux, python 3.6.9-final-0 ----------
Name                       Stmts   Miss  Cover   Missing
--------------------------------------------------------
application/__init__.py        4      0   100%
application/routes.py         15      0   100%
--------------------------------------------------------
TOTAL                         19      0   100%
```

Integration Test

I didn't perform Integration test for the application even if the tests themselves are simple to write, because I wasn't sure how to integrate them in the Jenkins Pipeline. However to perform Integration testing all I have to do is to install Selenium and Chomedriver and collect the Xpath of the links present in the application. The Xpath are here listed:

- From the Home page to the Generate quote page:

  ```
  /html/body/div/a
  ```

- From the Generate quote page:
  - Generate another quote:

    ```
    /html/body/div/span/p[1]/a/b
    ```

  - Return to the home page:

    ```
    /html/body/a
    ```

Then a typical test would be:

```
    # check if from the home page a user
    # is redirected to the generate quote page
    # when the correspondent link is clicked
    def test_1_home_to_quotess(self):
        # click the link to the full list of movies
        self.driver.find_element_by_xpath('/html/body/div/a').click()

        # Assert that browser redirects to all_movie page
        assert url_for('generate_quote') in self.driver.current_url
```

Possible Additional Tests

The test suite for the application could be further expanded by adding additional tests such as:

- **Compatibility testing**: This ensures that the application could be used with a variety of systems. This kind of test allow us to check the following:
    - Browser compatibility
    - Operating system compatibility
    - Mobile browsing
    - Printing options
- **Performance Testing**: This allow us to test if the application could sustains an heavy load of traffic from the internet, and understand how the system could be scaled when overloaded. Performance tests include:
    - Web Loading test (check if many users can access the application at the same time)
    - Web Stress test (checks how that application reacts when it is stressed over its limits, such as exceed maximum number of simultaneous connection possible)
- **Security Testing:** This allow us to identify potential vulnerability of the system. Security tests include:
    - Network Scanning
    - Vulnerability Scanning
    - Password Cracking
    - Log Review
    - Integrity Checkers
    - Virus Detection
    - Test unauthorized access to secure pages should not be permitted
    - Restricted files should not be downloadable without appropriate access
    - Check sessions are automatically killed after prolonged user inactivity
    - On use of SSL certificates, website should re-direct to encrypted SSL pages.

However we have not covered these tests in the academy, so I am not familiar with the methodology to implement them.

Deployment

Prerequisites

To deploy the application I used the following technologies:

- 3 GCP Virtual Machines
- 1 GCP MySQL database for unit testing purposes
- Jenkins Pipeline for CI/CD with Github Webhooks
- Pytest to test the application
- Docker to containerise the application
- Docker SWARM to run the application over the network of VM
- Nginx as reverse proxy
- Docker Hub as artefact repository
- GitHub as Version Control System for the application code
    - Feature Branch model used to implement the different aspects of the application.

Installation

Before I deployed the application I had to set up the virtual machines and I had to install the necessary software on them. The steps I followed were:

1. Setup 3 VM's on GCP with Ubuntu 18.04 LTS
2. Setup SSH keys for the 3 VM's and store public key of the virtual machine where the Jenkins server is going to be installed, into all the VM. To do so I used the GCP console, and I stored the ssk-key into the SSH section of the machine configuration, so that at any start of the VM the key is automatically loaded.
3. Install Jenkins on one GCP VM Instance to be used as a Jenkins Test Server. Expose TCP port 8080 in firewall rules
4. Add Jenkins to sudoers group
5. Setup Jenkins pipeline with Webhooks

Deployment process

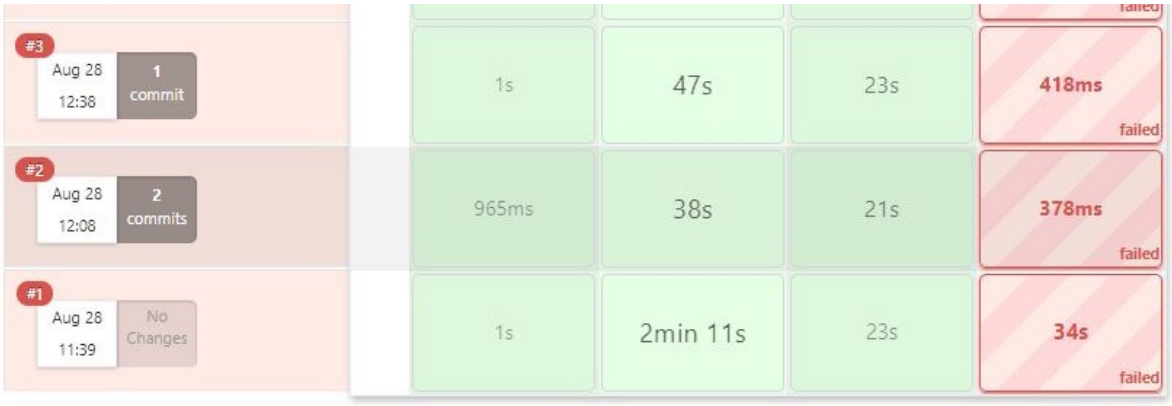
The deployment of this application is fully automated. Once a change of code is pushed on the version control system (Git) the use of WebHooks, allows the CI server Jenkins pull them and start building the application. The Jenkins pipeline consists in three stages, each of which is driver by a specific script.

The stages that Jenkins undertake are:

- Build: Jenkins installs Ansible (configuration tool) , use Ansible to run a playbook that installs docker (containerisation tool), initialise the Docker SWARM (orchestration tool) and assign one machine as SWARM-Manager and the other as SWARM-Worker. The use of a Docker SWARM allow the application to be deployed over a network of machines,and gives the possibility to deploy many replicas of each service (I used four replicas of each services), so that if any of the replicas experiences a problem, or if the system is updated and new version of the services are deployed, the system is always accessible and usable. Finally the building stage builds the docker images for each services and push them into Docker Hub, used as artefact repository.
- Test: Installs the necessary software requirements for each services and performs unit testing.
- Deploy: Deploys the application over a docker stack with the specification written in the docker-compose.yaml file, in particular each service is deployed with four replicas.

The following illustrates the Jenkins stage views for the application:

During the initial attempt of deploy the application I also experiences some problems:



I found the reason by inspecting the build's console log:

```
py2.py3-none-any.whl
Installing collected packages: MarkupSafe, jinja2, PyYAML, enum34, ipaddress, pycparser, cffi, six, cryptography, ansible
Successfully installed MarkupSafe-1.1.1 PyYAML-5.3.1 ansible-2.9.12 cffi-1.14.2 cryptography-3.1 enum34-1.1.10 ipaddress-1.0.23 jinja2-2.11.2
pycparser-2.20 six-1.15.0
./scripts/build.sh: line 12: ansible: command not found
Run ansible Playbook
./scripts/build.sh: line 17: ansible-playbook: command not found
Some services (nginx, service1, service2, service3, service4) use the 'deploy' key, which will be ignored. Compose does not support 'deploy'
configuration - use `docker stack deploy` to deploy to a swarm.
nginx uses an image, skipping
database uses an image, skipping
Building service1
```

In here is possible to se that even if Ansible was successfully installed, Jenkins wasn't able to found the commands to run the playbook. This is due to a possible bug in Jenkins that doesn't propagate the PATH environment variable. To solve this, on the build script instead of just call the ansible command :

```
ansible-playbook -v -i inventory.yaml playbook.yaml
```

I had to specify the full path for the command:

```
~/.local/bin/ansible-playbook -v -i inventory.yaml playbook.yaml
```

Future Improvements

The result of this project represents the minimum viable product that satisfies the minimum requirements. Some of the extended requirements have not been covered, so these could be introduced in future developments of the application. This includes:

- Add a new author to the list of authors
- Add a new quote to the list of quotes
- Add the user own quotes
- Delete an author from the list of authors
- Delete a quote from the list of quotes
- Good looking visual aspect

Acknowledgements

The Trainers at QA-Academy:

- Luke Benson, to which I own at least a drink for pointing me in the right path and taking me out of troubles when I was stuck.
- Harry Volker, for his suggestions and advice during the lectures' activities.

Finally, not least important, all the people at my DevOps July 2020 cohort which are source of suggestion, inspiration and fun!

Author

Domenico Gagliano