Content

1. Introduction
2. Short description about external API
3. Reasons why I chosen MongoDB
4. Few words about Docker
5. Description about structure project
6. Steps to reproduce deploy application
7. **Introduction**

This article describes the user guide to the project.

The project is a Flask REST application has a MongoDB database. This application implements CRUD operations, which allow various manipulations with user data.

Data generation is performed using an external service that simplifies the task of loading test data into the database.

Deploy flask application have to be on OS Ubuntu 18.04 LTS.

1. **Short description about external API**

In the test task was said to use the service [" https://randomuser.me/](RandomGenerator)" for generating random data (users). With the help of which it is possible to simplify the task of generating test data and uploading them to the database.

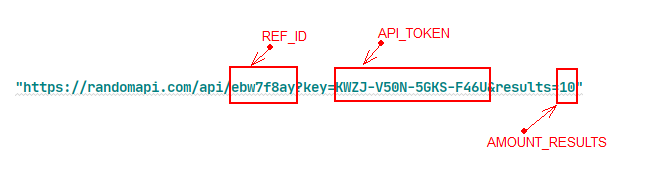
Application provided by the service should be written in javascript programming language.

Communication with the application is made by several parameters:

REF\_ID - unique number of the application written by you

API\_TOKEN - application key

Example presents below:



AMOUNT\_RESULTS – max amount of data from one response external api equal 10.

1. **Reasons why I chosen MongoDB**
2. The desire to get acquainted with noSql database closer.
3. Judging by the data with which the flask application works, the database does not have any dependence, it’s just grows in depth.
4. The task says that the Python script should generate 100 users. I can make a suggestion that a lot of data should be written to the database, so I chose MongoDB.
5. **Few words about Docker**

*Docker* is software to automate the deployment and management of applications in *containerized environments*, is used to manage individual containers (services) of which the application is composed

*Containerized environment* (Docker container) is a virtual operating system with only the essentials and the application code that is created using the *Docker image*

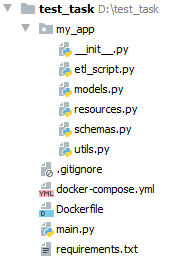
*Docker image* is used to create containers. Each Docker image corresponds to a file called a *Dockerfile*

*Dockerfile*is a set of instructions on how to create Docker image.

*Docker compose*is used to manage multiple containers that are part of an application simultaneously

**5. Description about structure project**

Structure project you can see on figure below:



Structure test project

Overview project structure:

*my\_app/etl\_script.py* – implementing connect to external API, pulling data from API, and upload to database;

*my\_app/models.py* ­– display information about the user data you are working with. They contain fields and the behaviour of data;

*my\_app/schemas.py* – is a data serializer for user data;

*my\_app/resources.py* – CRUD operations for manipulate data (made by REST);

*my\_app/utils.py* ­­– file for auxiliary functions;

*.gitignore* – auxiliary file for ignore files to upload on remote repository (Github)

*Dockerfile* – file to manage the individual containers (services) of which the application consists;

*docker-compose.yml* – file is used to control multiple containers that are part of an application at the same time;

*main.py* – file for run flask application;

*requirements.txt* – a list of all dependencies (packages) in python.

**6. Steps to reproduce deploy application**

1. First, you should use OS Ubuntu 18.04 LTS and install all dependencies need to success deploying application.

*sudo apt install git*

*sudo apt install docker*

1. After installation dependencies you need to clone project from remote repository on Github.

To clone project you can use next command: *git clone* *<attached\_link>*

Link presented below:

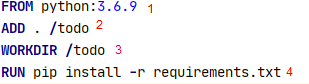
<https://github.com/Wroll/test_task123.git>

1. Next step is go to cloned project by you. For this, you need to use a next command:

*cd test\_task123*

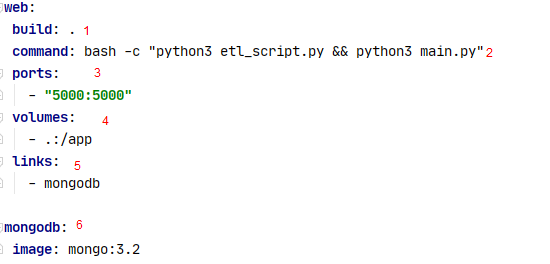
1. And now, you have all for comfortable deployment api. But before start to deploy api need to introduce with files *docker-compose.yml* and *Dockerfile*

Overview *Dockerfile*



1. base image based on python
2. copies files and folders to the container
3. create work directory
4. performs a command, which install python packages into container

Overview *docker-compose.yml*



1. The keyword "build" allows you to specify the path to the Dockerfile file that you want to use for creating the image
2. Commands need to run before flask image run
3. Connection to speak with flask application
4. Path to the folder on the host
5. Command to get additional information to speak with containers (optional)
6. Pulling from DockerHub existing image
7. Introduced a bit with Docker and Docker files you can deploy application
8. Next command building you app to images:

*sudo docker-compose build*

1. And final step is run containers which created on images

*sudo docker-compose up*