**Cristian Pana – technical test for MindGeek**

**1. Overview**

The solution consists in offloading the processing of each image to a taks queue (using RabbitMQ).

The setup uses Docker(docker-compose) to create an environment with the following services:

- php – running PHP-FPM for the interface

- nginx – web server

- db - database server with MariaDB

- rabbitmq - RabbitMQ service

- php-consume - PHP CLI app that will consume the tasks from RabbitMQ queue and actually download the images

There is just one project using Symfony Framework for both front and taks consumers.

**2. Deployement**

Change directory to where the project archive is decompressed and use docker-compose to create and start containers:

$ docker-compose up -d

Creating network "project\_symfony" with the default driver

Creating project\_nginx\_1 ... done

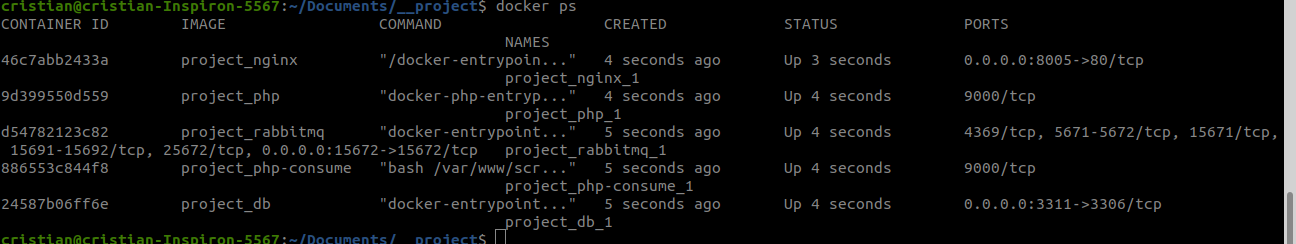
Creating project\_db\_1 ... done

Creating project\_rabbitmq\_1 ... done

Creating project\_php-consume\_1 ... done

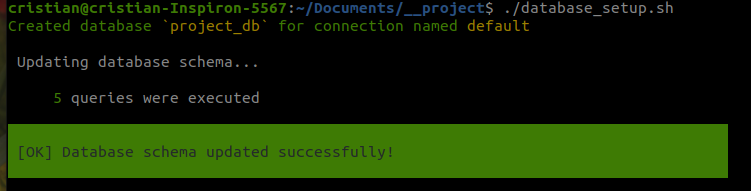
Creating project\_php\_1 ... done

Checking if all services are started should display something similar to:



Create database and tables using command:

$ ./database\_setup.sh



You can choose to have more than one consumer service by running:

$ docker-compose up -d --scale php-consume=2

project\_db\_1 is up-to-date

Starting project\_php-consume\_1 ...

Starting project\_php-consume\_1 ... done

project\_php\_1 is up-to-date

project\_nginx\_1 is up-to-date

Creating project\_php-consume\_2 ... done

**3. How to use**

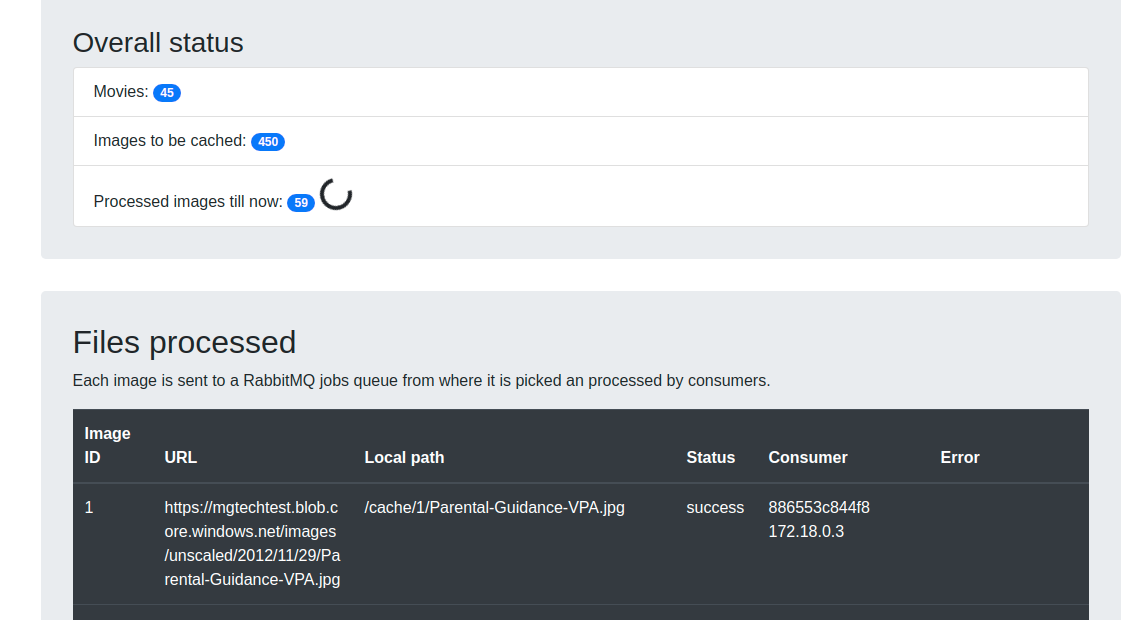
Access in your browser http://localhost:8005/

Press I**mport data** button in order to:

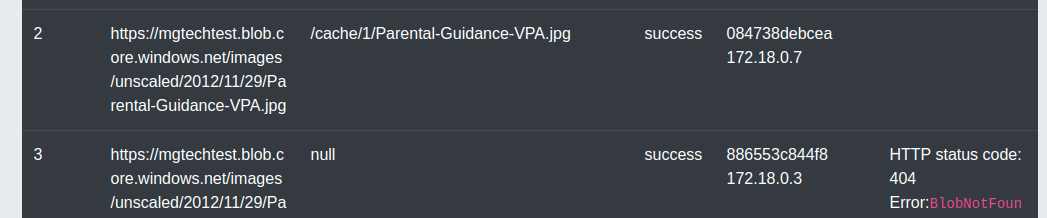
1. Synchronously read the JSON found at https://mgtechtest.blob.core.windows.net/files/showcase.json

and insert data into two database tables: record and image resulting 45 records and 450 images(cardImage and keyArtImage). After saving to DB each image a message is sent to RabbitMQ “messeges” queue. You can access RabbitMQ instance in your browser at [http://localhost:15672](http://localhost:15672/) with user **guest** and password **guest**

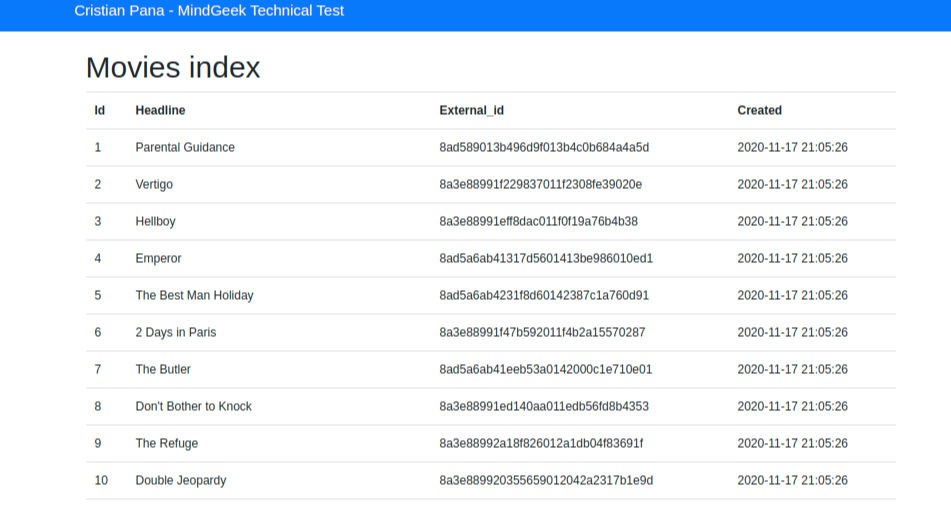
2. By using a log table (caching\_log) and AJAX requests the interface will be updated with the progress regarding caching each image:



Each time the consumer finishes processing an image the info in **image** table is updated and also a new row is inserted in **caching\_log** table in order to keep track of the progress.

In the Consumer column can be seen that different consumers are working if you scaled the number of consumers to more than one.

After all 450 images are processed please press **View data** button:



Click on any of the rows in order to see detailed view:



For demonstration purpose I’ve chose to display the HTTP code/error message for each image which was not processed successfully.

**4. Automated testing**

I’ve wrote unit tests for parts of the code that seem more in danger of having issues:

- reading and parsing the input JSON file

- downloading and saving images

Run tests with these command:

$ docker-compose exec php /bin/bash -c './vendor/bin/phpunit ./tests/'

…

Time: 00:00.013, Memory: 6.00 MB

OK (7 tests, 9 assertions)