**ThousandEyes Challenge**

1. Introduction**:**

I have created a reverse-proxy service for San Fransisco’s public transportation using the NextBus feed. This application is containerized using Docker. Please go through the following description for the details. I have created one public git repository and pushed all my code and scripts there. Here is the git repo: <https://github.com/Apoooorv/thousandEyesChallenge>

1. About Reverse-proxy service:
   1. Per the description given, I have created a reverse-procxy service for San Fransisco's public transportation using the NextBus feed.
   2. It is a **Django** application which is a reverse proxy service. The API endpoints are slightly changed; but all of the URLs are similar to the original APIs.
   3. All the APIs send the same response status code which comes from NextBus APIs.
   4. I have deployed that Django application with **gunicorn** server.
2. Statistics (API: /api/v1/stats):
   1. **Slow Requests**: I calculate time taken for each request, and if any request is slower than the threshold value, I keep track of that particular request. All those slow requests are sent as a response with the time taken by that request. If one API request takes more time than threshold value more than once, then the /stats/ API sends the maximum time took by that API.
   2. **Queries**: I calculate the number of times each API gets called and this count is sent along with the API name.
   3. Here I do not keep track the parameters sent with any API endpoint. For e.g. */agencylist/* and */agencylist?a=sf-muni* are considered as a same for stats.
3. Caching:
   1. I have added a layer of caching with a configurable timeout. If some API requires more time than the set timeout, then response is sent from cache. However, if an API response is sent from cache, then it is not considered for /stats/.
   2. I have used **pylru** library.
   3. One thing to note here is, initially, a cache is empty, so if there is some problem occurs with the request, and there is nothing in cache for that API, then I am sending a ‘Timeout’ as a response.
4. Containerization:

I have containerized the application using **Docker**. In order to make configuration of variables like timeout, threshold and database credentials, I am using environment variables. Because of this, we can make configurable timeout / thresholds per container. Using these environment variables, one can connect to any remote / local database and start using it.

1. Common Database:

Per the description given, I have created one **MySQL** database using AWS RDS to which I am connecting in my test scritps.

1. Automating scripts:

I have created following files for automation.

* 1. run.sh: Before using docker-compose, my running script (run.sh.old) was creating the docker image, create a container and running it. You could provide the database credentials, timeout period and threshold for slow queries period. I was passing these variables as **environmental variables** inside docker container.

Now, everything is built and taken care of automatically due to **docker-compose**. Therefore, there’s just one instruction in run.sh which is for docker-compose. I am keeping all the environmental variables in a file named **env-variables.env.**

* 1. test.sh: In this script, I just called some of the APIs which can be provided by the service.

1. About service’s infrastructure:
   1. Light weight: I used Django application for developing this infrastructure. One thing I could do in order to make it lighter is, I could use Flask or a simple python script. However, I did not use any of them because Django provides a layer of middlewares and in future, if I need to upgrade my service (or make it more complex) and develop more flexible APIs, I think Django is better option to use. Therefore, I opt Django for any other applications.
   2. Scalability: I am making it horizontally scalable using docker-compose. Moreover, I am using Django, because of which it will be easier for me to create future more complex APIs.
2. Remaining Tasks:

I can make this service less vulnerable, less error-prone and more secure in future.

1. Conclusion:

I tried to complete all the requirements which are asked. I was pleasure working on this mini-project. Thanks a lot for giving me an opportunity to work on these technologies which I want to work on in future.