Data Engineering II Final Project

Project Summary

As a part of the curriculum of the Master 2 (M2) course entitled "Data Engineering II", the students will complete a team project work. Each team is composed of 3 members, and the members will take care of dividing the tasks equally between them. The purpose of this project is to combine all the skills collected throughout the entire course, and to provide a solid example of real-life application development in a DevOps environment.

The following sections provide necessary information about the description of the application to be created.

For any further detail, please contact the instructor: **Khodor Hammoud**

1. User Stories

- The application is a toxicity monitor, where the user inputs a piece of text, and the application should be able to infer if the text is toxic or not.
- The text language used must be English
- The application should have a web interface with an input form and a submit button, where users can input their potentially toxic text, and hit submit, then the statistics about the text's toxicity are displayed.
- Every functional part of the application must be tested for proper functionality.
- The application must be able to handle 100 requests per minute.
- The application must be easily deployable.
- The application must be properly monitored after deployment, we want to be able to quickly find any issue that might cause performance problems or down time.

2. Technical Description

2.1 The Toxicity Model

The students will use the hugging-face model: **unitary/toxic-bert**. This model has been trained to detect toxicity in text, and returns analysis about different metrics relating to toxicity analysis. A full description of the model, as well as how to use it, can be found on their huggingface page:

https://huggingface.co/unitary/toxic-bert

2.2 The Web Interface

The students are free to choose whichever technology they know/like to create the web insterface. The end result should be a running application which the end user can access through a web browser, and start using immediately.

2.3 The Application Backend

The students are free to use whichever technology they know/like to create the application backend. The application backend covers the backend web server, as well as the toxicity model. The students have the choise to either implement the entire backend with python (using flask as the web server), or you can use a different technology (like NodeJs) for the server, and python for the toxicity model.

2.3 The Application Package

The final format of the application ready for distribution should be a **set of Docker Images**, which administrators can simply run Containers from. The students should use **Docker Compose** to facilitate the communication between, and the deployment of, Docker containers. Students should provide a description file with their submitted application in which whey describe how to run their image (like providing on which port does the application run by default…).

3. Technical Requirements

The students are to use the following technologies and steps throughout their implementation:

3.1 Task Management

Each team is to use a project management tool of their choice (Trello, Asana, Jira…) to coordinate the tasks between the team members. Divide the user stories into tasks, list these tasks on the project manager, and track the progress of each task as it progresses. Each team is required to present their project management history during their presentation.

3.2 Source Code Management

Each team is required to create a github repository containing their project, and use it as their version control. Each new task should have its own branch on the github repository. At every task completion (from the project manager), the associated team member should merge their task's branch to the master branch. The github repository should contain all your files, including the docker file and any meta data files (like the python requirements.txt if it exists).

Students must use the CD version control branching scheme, where the version control repository will contain a master branch, a develop branch, a feature branch (for every added feaure) and a release branch (for every version release).

3.3 Testing

Each team should provide unit, integration, end-to-end and stress tests to their final application.

- Unit tests are in the form of testing the functionality of each function of your program (when applicable).
- Integration testing will be testing combinations of functions, like clicking a button on the interface should trigger a submit function.
- End-to-end testing would be testing the entire functionality of the system, from frontend to backend. Example: inserting toxic text into the input form and clicking the submit button returns the statistics of the toxicity of the provided text.
- Stress testing will be writing a user simulation to prove that your application can handle 100 requests per minute.

3.4 Automation

The students are to use Jenkins for automating the building, testing, deployment and release (if applicable) of the application. At the end, each team is expected to have a Jenkins pipeline constructed which connects to the different github branches, and applies appropriate respective actions:

- build and run unit tests on feature branches.
- stress test and push to release on the develop branch
- wait for user acceptance on the release branch before pushing to master
- deploy on merging with master

**note: as it might not be possible to have multiple development environments (develop, staging, live), relasing/deploying the code can be substituted with simple print statements.

3.5 Containarization

The final application deliverable should be a set of Docker images, that contains the pre-trained model as well as the application web interface. The huggingface model should also be bundled with the application. Running a container off the delivered image should allow users to view a web interface on their browser and be able to immediately start running queries. The students are to **submit their docker images on Dockerhub**, for ease of reference by others.

3.6 Monitoring

The students are to use Prometheus to monitor:

- Hardware metrics: like CPU usage, memory usage, and disk space usage.
- Software metrics: integrate different software metrics inside your application to monitor information like response time, user requests count, exceptions,

Integrate Counters, Gauges, Histograms and Summaries as you see fit.

Add rules and alerts where you see fit, here are some examples:

- Alert before running out of memory
- Alert when cpu usage is very high
- Alert when your code raises an exception
- Alert when your system is down for more than a specific period of time
- …

Use Grafana as the monitoring dashboard.

One nice example to have is to visualize the different monitored metrics during the stress test.