MLOps Engineer Assignment

ZDP-AI Team *
Data & Analytics | ZEISS Digital Partners
Carl Zeiss AG

2024–06–17

*All cases and data in this assignment are entirely fictional.

1 Introduction

The ZEISS Digital Partners unit based in Munich, Budapest, Oberkochen and Jena is realizing cutting-edge customer-centric digital solutions jointly with the ZEISS business units. We are diverse cross-functional growing team of Product Owners, Data Scientists, Machine Learning Engineers, MLOps Engineers, DevOps Engineers and Full Stack Engineers. The team is being built to support strategic business units of ZEISS in their journey towards digitalization. We from ZEISS Digital Partners believe that one can see Data Science as software engineering with a data component. We focus on building our solutions in cloud working with both structured and unstructured data using Azure cloud services, terraform and Python Machine Learning eco system. We strive to have standards for open-source code and to master our tools. The aim of the following test is to assess some of the areas of expertise we think are necessary for a MLOps Engineer at ZEISS Digital Partners.

How to take this test?

This test consists of two parts:

- Part 1: Productionise a simple Data Science case
- Part 2: API build/deployment

Part 1 is a task more related to the role of an Machine Learning Engineer, Part 2 is more related to the role of an MLOps Engineer. Pick either one of the tasks and provide a solution to it, dependent on which position you are applying for. Feel free to also provide solutions for the other task but please do not feel obliged to do so.
Note: for Part 1 we provide a code basis, for Part 2 we do not provide anything and ask you to code from scratch.
Provide all writing, scripts, code, scans of written work you deem to be important for us to evaluate your solution. Please send solution files back by email. We understand that questions are open and time is limited, thus, try to organize your answers and keep in mind that we value quality over quantity.

Note

Here are a few things to consider while answering our questions:

- Machine Learning Engineering:
    1. What is the business case and KPIs to optimize?
    2. What kind of data is available?
    3. What is to be modeled and predicted?
    4. Are labels needed?
    5. What is the machine learning objective?
    6. Code structure?
    7. What would you do with the Jupyter notebooks?
    8. Usage of quickly changing and incompatible external libraries (Pandas, Tensorflow)
    9. What types of open source packages might you take advantage of?
    10. Which code quality standards would you enforce?
    11. Monitoring result quality and managing machine learning models
    12. CI/CD processes/pipelines
    13. Testing. Unit testing.
    14. Throughput performance bottlenecks
    15. Logging in code
- MLOps Engineering:
    1. Operations
    2. Orchestration
    3. Reproducibilty
    4. Throughput performance bottlenecks
    5. Failure Redundancy
    6. Testing. Unit testing.
    7. Logging in code

2 Assignment

Part 1. Simple Data Science case

This task is more related to the role of a Machine Learning Engineer. Thousands of our clients around the world use our hardware devices to conduct fundamental research. Our team needs to reduce the downtime and optimize the periodic maintenance operations for these devices with the help of machine learning algorithms. These devices are collecting log data about their internal state of operation, which we analyse with the help of machine learning.
In this part you are given a simple example Data Science use case: classification based on sensor data. For this assignment we provide data and some code in the attached **data_science_problem/** folder. See more explanation below.
Imagine a Data Scientist came up with a proof of concept in a notebook and her solution seem to solve the business problem at hand. You as an MLOps Engineer are joining the project at that stage to help productionise this use case and create cloud infrastructure.
You can assume that

- The log data are being written periodically to a text file on a computer connected to each device.
- Data are being collected and saved in the Data Lake (e.g. Azure Blob Storage, Amazon S3, Google Cloud Storage) by another team for you. You don't have to think about direct connections and protocols to those devices.
- You have all necessary authorisations and permissions to do what you want
- Data Science code is committed to git (say Azure DevOps) and you are given access to the repository
- The infrastructure should have at least 2 environments for development and production

Note, that the given code is intentionally kept simple and trivial, as we would like you to focus on maintenance, software and Machine Learning Engineering side of things, not the model itself. You are, however, encouraged to make additional and more ambitious assumptions in your answers, such as: more data, more complex Data Science algorithm, etc. ... For this example, we chose sensor data because it is close to the real problems that our team solves at ZEISS.
In **data_science_problem** folder you will find:

- model.ipynb (Data Science code) [exist also as model.html]
- requirements.txt (Python environment)
- data/historical_sensor_data.csv (sample historical data)
- data/real_sensor_data.csv (sample inference data)

How to run this code?

Assuming that you have pip and Python 3.8 on your system or virtual environment:

```
#!/bin/bash
python -m pip install --upgrade pip
pip install -r requirements.txt
# install kernel, that we named ds
python -m ipykernel install --user --name ds --display-name "ds"
# run jupyter server locally and navigate to the notebook
jupyter notebook
```

If you don't want to run the code you can explore model.html in the browser of your choice.

Questions

Taking the provided use case, prepare a production-ready code example, specifically concentrating on covering the following aspects:

- How would you assess if a code is *ready* for production? What do you think about the given code in model.ipynb?
- Would you change anything in the code? How and why? (Provide code examples)
- How would you rearrange code for production system? (Provide code examples)
- Would you add anything to the code? (Provide code examples)
- Discuss training and inference data provided.
- How would you organize the codebase management. E.g. how do you deliver code to the production environment? (make any assumption about code management system, that you like e.g. GitHub, Bitbucket, GitLab, Azure Repos)

Optional part

- Would you consider any alternatives to the used algorithm and if so, why?
- Discuss pros and cons of your chosen algorithms as well as evaluation metrics.
- Discuss Big O notation of the algorithms, its algorithmic and memory complexity.
- List libraries in various programming languages that you know or have used that implement those algorithms.
- Would your setup change if these were Databricks notebooks with Spark compute clusters behind them, written in PySpark?

Part 2. API build/deployment

For this part, we would like to assess your ability build and deploy an API of your own, and to consume a third party API. While the most important part is that you have fun, we would like to see how you approach the problem and how you solve it. We will evaluate this assignment based on current best practices and code quality, and whether clear instructions are provided on how to run the code and how the solution is documented.
For this part we do not provide any data. We provide one python file with code snippets that is attached in the folder *mlops_assignment*.
Keep it simple. We are not looking for a perfect solution, but rather a solution that is well thought through and well documented.

API with three endpoints: arxiv, queries, results

The API should have three endpoints:

- *arxiv* - scrape the arxiv.org API and store results in a local database (PostgreSQL)
- *queries* - fetch queries from the PostgreSQL database based on *query_start_time* and *query_end_time*
- *results* - provide all query_results of stored query_results in JSON format

In the folder *mlops_assignment* you will find a very simple python script with some help functions that should help you scraping the arxiv.org api and pull the necessary information for the coding challenge.

arxiv endpoint - scrape from arxiv.org API

- access arxiv.org API and query based on information about *author*, *title*, or *journal*. The function, called *search_arxiv()* should take at least one of the three input arguments The function, called *search_arxiv()* should take any combinatino of the three input arguments (*author*, *title*, or *journal*), with at least one of them not an empty string, as well as the maximum amount of query_results to return (*max_query_results*, which should be set to 8 per default).
- The query_results from arxiv.org should be ordered by relevance and in descending order.
- The function should be able to handle the case when no query_results are found.
- The function should be able to handle the case when the arxiv.org API is not available.
- Store the query and the query_results in a PostgresSQL database.

queries endpoint - provide a download option for the *queries* that are stored in the PostgreSQL

- Input should be two timestamps: *query_timestamp_start* (required) and *query_timestamp_end* (optional).
- Give as output all queries that have *query_timestamp* between *query_timestamp_start* and *query_timestamp_end*, both with format: *'yyyy-MM-dd'T'HH:mm:ss'* (24 hrs. format).
- The API call should return a file when invoked with a GET request.
- The API call should return a JSON object when invoked with a POST request.

Output queries	
query	the search string that was used
timestamp	the timestamp at which the query was made
status	the HTTP status code of the response from the arxiv.org API
num_results	the number of query_results found on arxiv.org

The output should have the following format:

```
{
    "query": "ArXiv Query: search_query=au=Ernst Abbe&amp;id_list=&amp;
       start=0&amp;max_results=8",
    "timestamp": "2015-10-12T10:14:14",
    "status": 200,
    "num_results": 42
}
```

***results* endpoint - provide the query_results of already stored query results in JSON format**

- write a GET method with optional inputs *page* and *items_per_page* which allow for pagination of the query_results displayed on *page* (starting with page 0), and *items_per_page* returned items per page.
- ordered increasingly, based on the time-point of storage in the database.

The output should contain the following information:

Output query results	
author	all authors of the result, pasted into one string, separated by ', '
title	the title of the result
journal	the journal in which the result was published

The output should then have the following format:

```
{
    "author": "Ernst Abbe",
    "title": "Ueber einen neuen Beleuchtungsapparat am Mikroskop",
    "journal": "Archiv f. mikrosk. Anatomie"
}
```

Note The content depicted above is not realistic and is not pulled from arxiv.org, it is simply for showcasing the format of the output, not a content we are looking for.

Tech requirements

- Use Python as the programming language
- Use FastAPI to build the API
- Use SQLAlchemy to interact with a PostgreSQL database
- Use docker-compose to run FastAPI and the PostgreSQL database