MINISTRY OF EDUCATION OF REPUBLIC OF MOLDOVA
Technical University of Moldova
Faculty of Computers, Informatics and Microelectronics
Department of Software Engineering and Automatics

Report

on Artificial Intelligence Fundamentals
Laboratory Work nr. 4

Performed by: **Vasile Boaghi, Ion Dodon**, FAF-172
Verified by: **Mihail Gavriliță**, asist. univ.

Chișinău, 2021

Contents

Model Deployment

Software deployment is all of the activities that make a software system available for use.

The general deployment process consists of several interrelated activities with possible transitions between them. These activities can occur at the producer side or at the consumer side or both. Because every software system is unique, the precise processes or procedures within each activity can hardly be defined. Therefore, "deployment" should be interpreted as a general process that has to be customized according to specific requirements or characteristics.

In software engineering, CI/CD or CICD generally refers to the combined practices of continuous integration and either continuous delivery or continuous deployment.

CI/CD bridges the gaps between development and operation activities and teams by enforcing automation in building, testing and deployment of applications. Modern day DevOps practices involve continuous development, continuous testing, continuous integration, continuous deployment and continuous monitoring of software applications throughout its development life cycle. The CI/CD practice or CI/CD pipeline forms the backbone of modern day DevOps operations.

The Task

The task is to make the linear regression model from laboratory work number 3 available for anyone and from anywhere. Also, is it needed to setup CI/CD workflow for the project. In GitHub action should be prepared a workflow that is triggered each time there is a push on the main branch and this workflow should test and build the application. A deployment workflow should be prepared as well and it should be triggered each time a new release has been made.

Solution Description

In laboratory work number 3 we had the model as a simple python file. That project only show that there is a model that does something. But, now we have to deliver the model model to the client to be used.

First of all we have to decide how the client will communicate with our model. One of the simplest solution is to use the HTTP protocol. This approach is probably not the best, because there can be a big amount of data that comes to the model to process, for example when a client what to make more predictions at once. But, we will use this protocol for simplicity.

If we want to communicate over HTTP, we have to prepare a HTTP server. Since our model is made and runs on python, the HTTP will also be made with python. Since in the case of this application, the most important part is the model and the server is used just to open an endpoint for communication, we will use Flask, because it is lightweight. An endpoint *GET /predict* will be available and it will take *application/json* content in the request's body.

Now, what is that data sanded by the client to the server/model? It is the data needed to make predictions. And we let it to be able to take more data sets in order to make more

predictions. The following is an example when it takes just one set of data and the model will make one prediction:

Listing 1: Input data example

```
{
    "1":{
        "1":-117.130000,
        "2":-118.040000
    },
    "2":{
        "1":32.700000,
        "2":33.760000
    },
    "complexAge":{
        "1":42.000000,
        "2":25.000000
    },
    "totalRooms":{
        "1":1210.000000,
        "2":4061.000000
    },
    "totalBedrooms":{
        "1":292.000000,
        "2":545.000000
    },
    "complexInhabitants":{
        "1":945.000000,
        "2":1623.000000
    },
    "apartmentsNr":{
        "1":258.000000,
        "2":527.000000
    },
    "8":{
        "1":0.899100,
        "2":7.157200
    },
    "medianCompexValue":{
        "1":78900.000000,
        "2":294900.000000
    }
}
```

The output for this input will be and array with the prediction for each data set from the input

Listing 2: Output data example

```
[
  114116.93713370664,
  332237.6029283451
]
```

We made a simple service that will wrap the model and tested this service with two tests - a positive test and a negative test. We decided that if the client send non-valid data the the client will receives a message in the form {*"errMessage": "Error. Check test data format!"*}. For simplicity, this message will be returned for any type of error in the server.

After having the server prepared and tested locally, we made a GitHub workflow that is triggered each time a push or pull request on the main branch has been made. This workflow runs the tests and if there are any non-passed tests the developer will be notified that the new code does not assure the correctness of the application. Then we create a workflow that build the application, creates a docker image, publishes the image on Docker Hub, spins a DigitalOcean[4] droplet and on that droplet run a container with the new image.

Code and Mentions

The server is a simple Flask application with just one endpoint.

Listing 3: The endpoint and error handler

```
app = Flask(__name__)


@app.route('/predict')
def predict():
    test_data = request.data.decode('utf-8')
    prediction = predict_apartment_value(test_data)
    return json.dumps(prediction)


@app.errorhandler(werkzeug.exceptions.HTTPException)
def handle_error(e):
    response = e.get_response()
    response.data = json.dumps({"errMessage": "Error. Check test data format!"})
    response.content_type = "application/json"
    return response
```

We created a class the will represent the liner regression model. When the class is instantiated, the model will be trained.

The server is a simple Flask application with just one endpoint.

We cannot check the predicted values if they are correct or not, because the model will return different value each time it is re-stared, because before it being trained, the model will get random weights and finally each time the model will make different predictions. Instead of checking the predicted values, we check is the number of predictions is correct.

Listing 4: Test the number of returned predictions

```
def test_predict_apartment_value_valid_json():
    test_data: str = """{
          "1":{
              "1":-117.130000
           },
           "2":{
              "1":32.700000
           },
           "complexAge":{
              "1":42.000000
           },
           "totalRooms":{
              "1":1210.000000
           },
           "totalBedrooms":{
              "1":292.000000
           },
           "complexInhabitants":{
              "1":945.000000
           },
           "apartmentsNr":{
              "1":258.000000
           },
           "8":{
              "1":0.899100
           },
           "medianCompexValue":{
              "1":78900.000000
           }
    }"""
    prediction = predict_apartment_value(test_data)
    assert len(prediction) == 1
```

The following is the workflow the is triggered on each new release. It will create a Docker image and the run it on a DigitalOcean droplet and so the server will be available for any client.

Listing 5: *build$_d$eploy.yml*

```yaml
name: Build and deploy

on:
  push:
    tags:
      - R*

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout files
        uses: actions/checkout@v2

      - name: Set up Python 3.8.5
        uses: actions/setup-python@v2
        with:
          python-version: 3.8.5

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install flake8 pytest
          if [ -f requirements.txt ]; then pip install -r requirements.txt; fi

      - name: Lint with flake8
        run: |
          # stop the build if there are Python syntax errors or undefined names
          flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
          # exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide
          flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics

      - name: Test with pytest
        run: |
          pytest -v

      - name: Build and push Docker images
        uses: docker/build-push-action@v1
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_PASSWORD }}
          repository: iondodon/lrmodel
```

```
43            tag_with_ref: true
44  #          tag_with_sha: true
45
46
47    deploy:
48      needs: build
49
50
51      runs-on: ubuntu-latest
52
53      steps:
54        - name: Checkout files
55          uses: actions/checkout@v2
56
57        - name: Get tag name
58          uses: olegtarasov/get-tag@v2.1
59
60        - name: Install doctl
61          uses: digitalocean/action-doctl@v2
62          with:
63            token: ${{ secrets.DIGITALOCEAN_ACCESS_TOKEN }}
64
65        - name: Provision Droplet and deploy container
66          run: doctl compute droplet create "$GIT_TAG_NAME" --image docker-18-04
     --size s-1vcpu-1gb --region nyc1 --user-data-file deploy.sh --wait
```

Conclusions

Working on this laboratory work we have learned how to continuously integrate out linear regression model by create a build workflow with GitHub actions and have also learned how to prepare a workflow for continuously delivery. We have understood how much the CI/CD principle help us to easier deliver any service after each new release and also how important CI is to assure that at any time we have a good working system. This was our first time when we prepared pipeline for CI/CD and now we are familiar with GitHub actions. We practiced how to make a docker image and publish it and how to work with Digital Ocean to host out service.

The application can now be tested by making a HTTP GET request to `http://157.245.253.166:5000/predict` and in the body should be a JSON as in the example from above.

References

[1] Vasile Boaghi, Ion Dodon. Source code for the laboratory work. Accessed February 28, 2021. https://github.com/iondodon/FIA_Lab4.

[2] Software Deployment https://en.wikipedia.org/wiki/Software_deployment.

[3] Deploying to DigitalOcean With GitHub Actions https://www.digitalocean.com/community/tech_talks/deploying-to-digitalocean-with-github-actions

[4] DigitalOcean https://www.digitalocean.com/