4.0 - MLOps: Automation

Introduction

Now that we have configured our GitHub Actions to work with Kubernetes and Azure, we can tie everything together in this final assignment. This focusses more on the MLOps aspect of chaining the GitHub Actions.

Objective

Setting up the GitHub Actions to connect with Azure will allow us to implement the MLOps best practices in configuring reusable pipelines that can plug in where needed. We will focus on using Azure Machine Learning Pipelines and the components that Azure manages.

Knowledge

- Azure
- Azure Machine Learning
- Neural Networks

Skills

- Creating an MLOps pipeline in GitHub Actions
- Deploying AI models automatically into a Kubernetes cluster

Necessities

- The Python scripts used in
- The animals data in
- The FastAPI from
- The local runner from
- The Azure setup from

GitHub Actions Pt. 1 — Prepare and Train

As GitHub Actions is already configured to use the Azure CLI in your username as setup in we can proceed to setup everything regarding the MLOps pipeline now.

In our pipeline, we want to automate a few steps, that we can always repeat.

1. Create a compute machine
2. Create our environments
3. Apply our Components
4. Start the training job

This will just be the first part of the full pipeline which we can automate. Many of these steps have been done using the Azure CLI in which is where we will

base everything on.

That assignment ended with a full pipeline which was setup in one YAML file.

The full YAML file can be found here:

▼ Full Pipeline

```
$schema: https://azuremlschemas.azureedge.net/latest/pipelineJob.schema.json

type: pipeline
name: animals-classification-cli
display_name: Animals Classification
experiment_name: classification

inputs:
  train_test_split_percentage: 20
  epochs: 5

outputs:
  model:
    mode: upload
  registration_details:
    mode: upload

settings:
  # default_compute: serverless
  default_compute: azureml:cli-created-machine

jobs:
  data_prep_pandas:
    type: command
    # component: azureml:dataprep:0.1.0
    component: ./components/dataprep/dataprep.yaml
    inputs:
      data:
        type: uri_folder
        path: azureml:pandas:1
```

```
  outputs:
    output_data:
      mode: rw_mount

data_prep_cats:
  type: command
  # component: azureml:dataprep:0.1.0
  component: ./components/dataprep/dataprep.yaml
  inputs:
    data:
      type: uri_folder
      path: azureml:cats:1

  outputs:
    output_data:
      mode: rw_mount

data_prep_dogs:
  type: command
  # component: azureml:dataprep:0.1.0
  component: ./components/dataprep/dataprep.yaml
  inputs:
    data:
      type: uri_folder
      path: azureml:dogs:1

  outputs:
    output_data:
      mode: rw_mount

data_split:
  type: command
  component: ./components/dataprep/data_split.yaml
  inputs:
    animal_1: ${{parent.jobs.data_prep_pandas.outputs.output_data}}
    animal_2: ${{parent.jobs.data_prep_cats.outputs.output_data}}
    animal_3: ${{parent.jobs.data_prep_dogs.outputs.output_data}}
    train_test_split_percentage: ${{parent.inputs.train_test_split_percent
```

```
age}}
    outputs:
      testing_data:
        mode: rw_mount
      training_data:
        mode: rw_mount

  training:
    type: command
    component: ./components/training/training.yaml
    inputs:
      training_folder: ${{parent.jobs.data_split.outputs.training_data}}
      testing_folder: ${{parent.jobs.data_split.outputs.testing_data}}
      epochs: ${{parent.inputs.epochs}}
    outputs:
      output_folder: ${{parent.outputs.model}}

  register:
    type: command
    component: azureml://registries/azureml/components/register_mode
l/versions/0.0.21
    inputs:
      model_name: animal-classification
      model_type: custom_model
      model_path: ${{parent.jobs.training.outputs.output_folder}}
    outputs:
      registration_details_folder: ${{ parent.outputs.registration_details }}
```

We note that the file contains some “hard-coded” settings such as the name of the compute machine. We don’t necessarily need to override this here, but we could do so if we want to automate it fully. This is some extra parts of the assignments.

1. Compute — Creation

Every time we start our pipeline, we want to create our compute machine, to make sure we have a fresh copy in case it had been deleted in the past.

This also allows the so-called **cold-start** where the pipeline is running for the very first time. It's very important to think about that too, because you will often restart in a new Azure environment if needed.

To create the machine, we executed this command.

`az ml compute create --file ./environment/compute.yaml`

Please keep in mind that the files should be stored under that exact location in order for this to work.

The full step for this job would be:

```
    - name: Azure -- Create compute
      uses: Azure/CLI@v2.1.0
      with:
        azcliversion: 2.64.0
        inlineScript: |
          az extension add --name ml
          az configure --defaults group=$GROUP workspace=$WORKSPACE
location=$LOCATION
          az ml compute create --file ./environment/compute.yaml
```

❓ **QUESTION 1 - Check latest versions**
In the assignment you checked which versions were the latest, make sure to also update the versions in your pipeline here.
And in all the next steps as well!

 **ANSWER 1 - Check latest versions**

- `azcli` — Current version: 2.64.0
- `Azure/CLI` — Current version: v2.1.0

2. Compute — Start

Starting the compute from the Azure CLI will give **an error** when the machine **is already on.** But to be compliant with our **cold-start** we need to execute the command anyway.

To fix this, we can ignore the error for this specific command, as a quick fix. A better and more complex solution would be to work around it and perform more checks before executing the command.

By using the `continue-on-error` we simply ignore the error, and still continue the rest of the pipeline by going to the next step in this job.

```
    - name: Azure -- Start Compute
      uses: azure/CLI@v2.1.0
      with:
        azcliversion: 2.64.0
        inlineScript: |
          az extension add --name ml -y
          az configure --defaults group=$GROUP workspace=$WORKSPACE
location=$LOCATION
          az ml compute start --name cli-created-machine
      continue-on-error: true # Ignore any errors, also the crucial ones.
```

ANSWER 2 - Robust Pipelines: Compute - *Extra*

```
    - name: Azure -- Start Compute
      uses: Azure/CLI@v2.1.0
      env:
        AZURE_CONFIG_DIR: ${{ env.AZURE_CONFIG_DIR }}
      with:
        azcliversion: 2.64.0
        inlineScript: |
          export AZURE_CONFIG_DIR="$AZURE_CONFIG_DIR"
          CREDS='${{ secrets.AZURE_CREDENTIALS }}'
          CLIENT_ID=$(python3 -c "import sys, json; print(json.load
(sys.stdin)['clientId'])" <<< "$CREDS")
          CLIENT_SECRET=$(python3 -c "import sys, json; print(jso
n.load(sys.stdin)['clientSecret'])" <<< "$CREDS")
          TENANT_ID=$(python3 -c "import sys, json; print(json.loa
d(sys.stdin)['tenantId'])" <<< "$CREDS")
          SUB_ID=$(python3 -c "import sys, json; print(json.load(sy
s.stdin)['subscriptionId'])" <<< "$CREDS")
          az login --service-principal -u "$CLIENT_ID" -p "$CLIENT_
SECRET" --tenant "$TENANT_ID" --output none
          az account set --subscription "$SUB_ID"
          az extension add --name ml
          az configure --defaults group=$GROUP workspace=$WO
RKSPACE location=$LOCATION

          # Check if compute exists before trying to start it
          if az ml compute show --name warre-cluster &> /dev/null;
then
            STATE=$(az ml compute show --name warre-cluster --q
uery "provisioning_state" -o tsv)
            echo "Compute exists with state: $STATE"

            if [ "$STATE" != "Succeeded" ]; then
              echo "Compute is not in Succeeded state. Current state:
$STATE"
              echo "Waiting for compute to be ready..."
              sleep 30
```

```
            fi

            # Check if compute is already running
            CURRENT_STATE=$(az ml compute show --name warre-
cluster --query "provisioning_state" -o tsv)
            if [ "$CURRENT_STATE" == "Succeeded" ]; then
              echo "Starting compute cluster..."
              az ml compute start --name warre-cluster || echo "Com
pute may already be starting or running"
            fi
          else
            echo "Compute cluster does not exist. It will be created i
n the next step."
          fi
        continue-on-error: true
```

3. Environments & Components

For your environments `./environment/pillow.yaml` and `./environment/tensorflow.yaml` and the different components from `./components/*.yaml` we can combine them in a few steps.

Below is the example for the environments.

```
    - name: Azure -- Environment Setup
      uses: Azure/CLI@v2.1.0
      with:
        azcliversion: 2.64.0
        inlineScript: |
          az extension add --name ml
          az configure --defaults group=$GROUP workspace=$WORKSPACE
location=$LOCATION
          az ml environment create --file ./environment/pillow.yaml
          az ml environment create --file ./environment/tensorflow.yaml
```

Repeat this for the components

```
    - name: Azure -- Component Setup
      uses: Azure/CLI@v2.1.0
      with:
        azcliversion: 2.64.0
        inlineScript: |
          az extension add --name ml
          az configure --defaults group=$GROUP workspace=$WORKSPACE
location=$LOCATION
          az ml components create --file ./components/dataprep/dataprep.ya
ml
          az ml components create --file ./components/dataprep/data_split.ya
ml
          az ml components create --file ./components/training/training.yaml
```

❓ **QUESTION 3 - Loop over components**

‼️ Answer this question when you've gone through the complete pipeline

Find a way to loop over the components and environments in a nice pattern so we don't have to repeat ourselves.

ANSWER 3 - Loop over components

```
    - name: Azure -- Environment Setup
      uses: Azure/CLI@v2.1.0
      env:
        AZURE_CONFIG_DIR: ${{ env.AZURE_CONFIG_DIR }}
      with:
        azcliversion: 2.64.0
        inlineScript: |
          export AZURE_CONFIG_DIR="$AZURE_CONFIG_DIR"
          CREDS='${{ secrets.AZURE_CREDENTIALS }}'
          CLIENT_ID=$(python3 -c "import sys, json; print(json.load
(sys.stdin)['clientId'])" <<< "$CREDS")
          CLIENT_SECRET=$(python3 -c "import sys, json; print(jso
n.load(sys.stdin)['clientSecret'])" <<< "$CREDS")
          TENANT_ID=$(python3 -c "import sys, json; print(json.loa
d(sys.stdin)['tenantId'])" <<< "$CREDS")
          SUB_ID=$(python3 -c "import sys, json; print(json.load(sy
s.stdin)['subscriptionId'])" <<< "$CREDS")
          az login --service-principal -u "$CLIENT_ID" -p "$CLIENT_
SECRET" --tenant "$TENANT_ID" --output none
          az account set --subscription "$SUB_ID"
          az extension add --name ml
          az configure --defaults group=$GROUP workspace=$WO
RKSPACE location=$LOCATION

          # Loop through all environment YAML files
          for file in ./assignment/environment/*.yaml; do
            if [[ -f "$file" ]] && grep -q "environment.schema.json"
"$file"; then
              echo "Creating environment from $file"
              az ml environment create --file "$file" || echo "Environm
ent may already exist"
            else
              echo "Skipping $file (not an environment definition)"
            fi
          done
```

```
    - name: Azure -- Component Setup
      uses: Azure/CLI@v2.1.0
      env:
        AZURE_CONFIG_DIR: ${{ env.AZURE_CONFIG_DIR }}
      with:
        azcliversion: 2.64.0
        inlineScript: |
          export AZURE_CONFIG_DIR="$AZURE_CONFIG_DIR"
          CREDS='${{ secrets.AZURE_CREDENTIALS }}'
          CLIENT_ID=$(python3 -c "import sys, json; print(json.load
(sys.stdin)['clientId'])" <<< "$CREDS")
          CLIENT_SECRET=$(python3 -c "import sys, json; print(jso
n.load(sys.stdin)['clientSecret'])" <<< "$CREDS")
          TENANT_ID=$(python3 -c "import sys, json; print(json.loa
d(sys.stdin)['tenantId'])" <<< "$CREDS")
          SUB_ID=$(python3 -c "import sys, json; print(json.load(sy
s.stdin)['subscriptionId'])" <<< "$CREDS")
          az login --service-principal -u "$CLIENT_ID" -p "$CLIENT_
SECRET" --tenant "$TENANT_ID" --output none
          az account set --subscription "$SUB_ID"
          az extension add --name ml
          az configure --defaults group=$GROUP workspace=$WO
RKSPACE location=$LOCATION
          COMPONENT_VERSION="${COMPONENT_VERSION:-0.
1.${GITHUB_RUN_NUMBER}}"
          echo "Publishing components with version: $COMPONEN
T_VERSION"

          # Loop through components (only actual component YAM
L files, not conda.yaml or environment.yaml)
          for dir in ./assignment/components/*/; do
            for file in "$dir"*.yaml; do
              # Skip conda.yaml and environment.yaml files - only pro
cess component definition files
              basename_file=$(basename "$file")
              if [[ "$basename_file" != "conda.yaml" ]] && [[ "$basena
me_file" != "environment.yaml" ]] && [[ -f "$file" ]]; then
                echo "Creating component from $file (version $COMP
```

```
ONENT_VERSION)"
              az ml component create --file "$file" --set version
="$COMPONENT_VERSION"
            else
              echo "Skipping $file (not a component definition)"
            fi
          done
        done
```

4. Pipeline in Azure

After all the components, environments and even the compute machine has been set, we can start applying our job.

- Create a new step in the GitHub Action Jobs (appending to the same file)
- Make sure it executes `az ml job create --file ./pipelines/animals-classification.yaml`
- We will add two extra parameters for this command.
  1. `--stream` will allow us to view the results of this command into our terminal, so we can also wait until it's finished. Otherwise the command will quickly give us a result that it started.
  2. `--set name=animals-classification-${{ github.sha }}-${{ github.run_id }}` This parameter changes the name that can now be dynamically created from this GitHub Action Run. It uses 2 properties from the GitHub Action
     - `github.sha` is the commit hash, which is unique for every commit. This is useful for traceability to which code has been executed, which we will explain more about soon.
     - `github.run_id` is the unique number that increments every time this pipeline is running. That way, the settings that the users can fill in is also traced.

When this pipeline is done, we can stop our compute machine as a Cleanup step

```
az ml compute stop --name cli-created-machine
```

Also add this line, so it can continue if the machine is already stopped.
`continue-on-error: true`

This will be the end of this **job**, and the next **job** can then continue.

GitHub Actions Pt. 2 — Download trained model

For this part of the pipeline, we create an extra **job**.

This means our structure will now be like this (schematically)

```
1. azure-pipeline:
   ----------------
   - Check out repository
   - Azure Login
   - Azure -- Create Compute
   - Azure -- Start Compute
   - Azure -- Environment Setup
   - Azure -- Component Setup
   - Azure -- Pipeline Run
   - Cleanup Compute

2. download:
   -----------
```

Check this, and focus on the **first line** and the **last step.**

```
download:
    needs: azure-pipeline # New!!
    runs-on: ubuntu-24.04
    steps:

    - name: Check out repository
      uses: actions/checkout@v4

    - name: Azure Login
      uses: azure/login@v2
```

```
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

    - name: Azure -- Download Model
      uses: azure/CLI@v2.1.0
      with:
        azcliversion: 2.64.0
        inlineScript: |
          az extension add --name ml -y
          az configure --defaults group=$GROUP workspace=$WORKSPACE l
ocation=$LOCATION
          VERSION=$(az ml model list -n animal-classification --query "[0].ver
sion" -o tsv)
          <Replace this with the command you used in Question 3>

    # New!!
    - name: Docker -- Upload API code from Inference
      uses: actions/upload-artifact@v4.3.3
      with:
        name: docker-config
        path: inference
```

❓ **QUESTION 3 - Download AI Model Pipeline**

1. Download the AI model using the `az ml` command like you did in
2. What is the purpose of the **`needs: azure-pipeline`** ?
3. What's the point of the `actions/upload-artifact@v4.3.3` ?
   https://github.com/marketplace/actions/upload-a-build-artifact

ANSWER 3 - Download AI Model Pipeline

1. Command:

```
VERSION=$(az ml model list -n animal-classification --query "[0].
version" -o tsv)
az ml model download --name animal-classification --version $V
ERSION --download-path ./assignment/inference/model
```

1. `needs`: The `needs` keyword creates a dependency chain between jobs in GitHub Actions. It ensures that the `download` job only starts after the `azure-pipeline` job has successfully completed.
2. `upload-artifact`:

```
This action persists files between jobs in a GitHub Actions workfl
ow. Specifically:
  - Each job runs in a fresh, isolated environment
  - Files created in one job are lost when that job completes
  - `upload-artifact` saves files (the inference code + model) to Gi
tHub's artifact storage
  - The `deploy` job (running on a different/self-hosted runner) ca
n then `download-artifact` to get these files
  - Without this, the deploy job wouldn't have access to the model
or inference code
  - It acts as a "handoff" mechanism between jobs running on diff
erent machines
```

GitHub Actions Pt. 3 — Deployment to Kubernetes

The following GitHub Actions job comes right under the `download` pipeline in a similar structure

Here's a little bit more explanation about the different steps

1. **Gather Docker Meta Information**

This job step prepares the tags for the Docker Build.

In this case, we are planning to name the image **ghcr.io/nathansegers/mlops-animals-api.** We chose `ghcr.io` (GitHub Container Registry) because this allows us for many more **private** Docker Images than the Docker Hub Registry.
It's important to change `nathansegers` to your name, so you have the correct access rights to push. The name `mlops-animals-api` is free to choose.

The tags we will add to this image will be based on some GitHub Attributes. We can use the `type=ref,event=branch` to make sure we are adding an image `mlops-animals-api:main` when this pipeline is running in the `main` branch.

In another tag, we are using the GItHub Sha, like we already explained earlier. This is already traced back to the right GitHub Commit.

Later, we will use this for the `build_and_push` step a little further down.

2. We need to log in to the GitHub Container Registry or the Docker Hub.

   For Docker Hub, we need an extra token, as the secret `DOCKER_HUB_PASSWORD` (or Personal Access Token from Docker Hub)

   For GitHub Container Registry, we can use the same Token we already have, **but we need to set some extra rights for the pipeline as explained here** (GitHub Container Registry)

3. We download the inference code

```
deploy:
  needs: download
  runs-on: self-hosted
  steps:
  - name: Docker -- Gather Tags
    id: docker-meta-data
    uses: docker/metadata-action@v5.5.1
    with:
      # list of Docker images to use as base name for tags
      # Use this for Docker Hub
      # nathansegers/mlops-animals-api
      images: |
        ghcr.io/nathansegers/mlops-animals-api
      # generate Docker tags based on the following events/attributes:
```

```
        # The GitHub Branch
        # The GitHub SHA
        # More info: https://github.com/docker/build-push-action/blob/master/
docs/advanced/tags-labels.md
        tags: |
          type=ref,event=branch
          type=sha

    # Enter your GITHUB Token here!
    - name: Docker -- Login to GHCR
      uses: docker/login-action@v3.2.0
      with:
        registry: ghcr.io
        username: ${{ github.repository_owner }}
        password: ${{ secrets.GITHUB_TOKEN }}

    # # IF YOU USE DOCKER HUB, UNCOMMENT THIS
    # # Enter your DOCKER HUB Token in the DOCKER_HUB_PASSWORD se
cret
    # - name: Docker -- Login to Docker Hub
    #   uses: docker/login-action@v3.2.0
    #   with:
    #     username: ${{ github.repository_owner }}
    #     password: ${{ secrets.DOCKER_HUB_PASSWORD }}

    # Download artifact from previous step
    - name: Docker -- Download API Code for Inference
      uses: actions/download-artifact@v4.1.7
      with:
        name: docker-config
        path: inference

    - name: Docker Build and push
      id: docker_build
      uses: docker/build-push-action@v5.3.0
      with:
        context: ./inference
```

```
      push: true
      tags: ${{ steps.docker-meta-data.outputs.tags }}
```

Kubernetes deployment

Now that the Docker image has been pushed, we can proceed to deploying it locally, through Kubernetes, or to an Azure environment if you wish.

Try to add a new pipeline job to deploy to Kubernetes using the `kubectl` command line and your GitHub Actions Runner. This will be similar to and the way that was set up.

Use your creativity in either creating a new pipeline job, a new pipeline step, or even a new pipeline workflow. It's all your choice.

❓ **QUESTION 5 - Deploy to Kubernetes**

1. Paste the YAML file for your Kubernetes deployments
2. Paste the GitHub Actions pipeline step with the `kubectl` commands

ANSWER 5 - Deploy to Kubernetes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: animal-classification-deployment
  labels:
    app: animal-classification
spec:
  replicas: 1
  selector:
    matchLabels:
      app: animal-classification
  template:
    metadata:
      labels:
        app: animal-classification
    spec:
      containers:
        - name: animal-classification
          image: ghcr.io/snaetwarre/mlops-animals-api:master
          ports:
            - containerPort: 8000
          env:
            - name: MODEL_PATH
              value: /app/model/animal-classification/INPUT_model_pa
th/animal-cnn/model.keras
---
apiVersion: v1
kind: Service
metadata:
  name: animal-classification-service
spec:
  type: LoadBalancer
  selector:
    app: animal-classification
  ports:
```

```
  - port: 80
    targetPort: 8000
```

```
deploy:
  needs: download
  runs-on: self-hosted
  steps:
    - name: Check out repository
      uses: actions/checkout@v4

    - name: Docker -- Gather Tags
      id: docker-meta-data
      uses: docker/metadata-action@v5.5.1
      with:
        images: |
          ghcr.io/${{ github.repository_owner }}/mlops-animals-api
        tags: |
          type=ref,event=branch
          type=sha

    - name: Docker -- Login to GHCR
      uses: docker/login-action@v3.2.0
      with:
        registry: ghcr.io
        username: ${{ github.repository_owner }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Docker -- Download API Code for Inference
      uses: actions/download-artifact@v4.1.7
      with:
        name: docker-config
        path: assignment/inference

    - name: Docker Build and push
      id: docker_build
      uses: docker/build-push-action@v5.3.0
      with:
        context: ./assignment/inference
```

```
          push: true
          tags: ${{ steps.docker-meta-data.outputs.tags }}

      - name: Prepare Kubernetes Config
        run: |
          mkdir -p $HOME/.kube
          printf '%s\n' "${{ secrets.KUBE_CONFIG }}" > $HOME/.kub
e/config
          chmod 600 $HOME/.kube/config

      - name: Kubernetes -- Deploy
        run: |
          kubectl apply -f ./assignment/kubernetes/deployment.yaml
```

Alternative: Deploy to HuggingFace

If you want to use HuggingFace to deploy your application instead, that's also a possibility!

Feel free to work on that as an extra part. This will allow your application to be hosted into a public environment with auto-scaling implemented by design.

This also allows you to use HuggingFace to store and download your model.

❓ **QUESTION 5b - Deploy to HuggingFace**

1. Paste the GitHub Actions pipeline step with the HuggingFace adaptations.

ANSWER 5b - Deploy to HuggingFace

```
      - name: Push Model to Hugging Face Hub
        env:
          HF_TOKEN: ${{ secrets.HF_TOKEN }}
        run: |
          pip install huggingface_hub

          # Login to Hugging Face
          huggingface-cli login --token $HF_TOKEN

          # Upload the model
          huggingface-cli upload ${{ github.repository_owner }}/anim
al-classification \
            ./assignment/inference/model \
            --repo-type model

      - name: Create Hugging Face Space for Inference
        env:
          HF_TOKEN: ${{ secrets.HF_TOKEN }}
        run: |
          # Create a Spaces repo for the API
          huggingface-cli repo create animal-classification-api \
            --type space \
            --space_sdk gradio

          # Push inference code
          cd ./assignment/inference
          git init
          git remote add hf https://huggingface.co/spaces/${{ github.
repository_owner }}/animal-classification-api
          git add .
          git commit -m "Deploy inference API"
          git push hf main
```

Best Practices - Professionalising the pipeline

📖 Read more on this in the document first, so you're fully up-to-date on Pipeline Controlling and professionalising CI/CD pipelines.

Pipeline Controlling

- Implement the pipeline controlling as defined in the Best Practices, by setting up input parameters in the GitHub Actions pipeline.
- Make sure they are used in the pipeline YAML file, to skip specific jobs if required.

Version Controlling

- Implement version controlling as defined in the Best Practices

Debugging GitHub Actions

As GitHub Actions is just about executing commands in the right order, we can try to debug the steps locally at first.

Here are a few suggestions how you can debug this:

- Clone the repository onto your laptop, or vm
- Create a fresh Virtual Environment for Python.
- Create a `.env` file which will contain all the environment variables that you normally fill in into the GitHub Actions secrets. Copy the `.env.example` to get a starter of the values to fill in.
- Now you can execute all the commands one by one, and fix files if necessary. This is similar to what is already been done
- Don't forget to Commit your changes if you want them to get executed through the GitHub Actions now! In case you have changed some environment variables, make sure to reflect those changes into the `azure-ai.yaml` file too.

Extra: Adding Responsible AI

All the Responsible AI dashboards from Azure can also be implemented in this pipeline if we want to. This could be very interesting for your stakeholders so that the application can be more advanced and robust.

Rounding off

❓ **QUESTION 6 - Final suggestions**

Do you have more suggestions how we can add more functionalities into the GitHub Actions pipeline to create a better MLOps flow?
This is a free question, there are no good or wrong answers. Use your own knowledge or things we learned in the previous sessions / assignments. It might be good to capture your interests to include it in the assignments later on...

 **ANSWER 6 - Final Suggestions**

1. **Automated Testing Pipeline**: Add comprehensive testing sta ges:
   - Unit tests for data preprocessing and model training code
   - Integration tests for the FastAPI endpoints
   - Model performance tests (accuracy threshold checks)
   - Load testing for the deployed API
   - A/B testing framework for comparing model versions

2. **Model Drift Detection**: Implement monitoring to detect whe n model performance degrades:
   - Log predictions and actual outcomes
   - Compare current model metrics against baseline
   - Trigger retraining workflow automatically when drift is detecte d
   - Send alerts to Slack/Teams when issues are found

What did you learn?

Fill in something that you learned during this lesson

> **Orchestrating Complex Multi-Stage Workflows**: I learned how to design and implement a complete MLOps pipeline that chains together multiple jobs across different environments (Azure ML, GitHub Actions, Kubernetes). The use of `needs` dependencies, artifact passing, and coordinating between cloud and self-hosted runners taught me the complexity of real-world CI/CD pipelines.

> **Azure ML Integration with GitHub Actions**: I gained hands-on experience with the Azure ML CLI and how to programmatically manage the entire ML lifecycle - from compute provisioning, environment setup, component registration, pipeline execution, and model registration - all from within GitHub Actions workflows.

> **Debugging Distributed Systems**: I learned practical debugging skills when things go wrong in CI/CD pipelines - from RBAC permission issues, Docker registry authentication, Kubernetes configuration problems, to environment variable management. The importance of proper logging, error handling, and understanding the execution context of each job became very clear.

> **Self-Hosted Runners Architecture**: I understood why and when to use self-hosted runners versus GitHub-hosted runners, particularly for deployment scenarios requiring access to local infrastructure (like k3d Kubernetes clusters) or when you need specific tools/configurations not available in GitHub's runners.

Give three interesting exam questions

1. **Explain the complete data flow in this MLOps pipeline**: Starting from a code push to the repository, describe each stage of the pipeline, what happens in each job, how data is passed between jobs, and where the trained model ultimately ends up deployed. Include the purpose of artifacts and the role of different runner types (GitHub-hosted vs self-hosted).

2. **Azure ML RBAC and Permissions**: Why did the model registration fail initially with an "AuthorizationFailed" error, and what was the root cause? E xplain the difference between the service principal used by GitHub Actions and the compute cluster's managed identity. What permissions are needed for each, and why can't they share the same credentials?

3. **Kubernetes Deployment Architecture**: Explain why the Kubernetes d eployment initially failed with "InvalidImageName" and how template variab le substitution works (or doesn't work) in different contexts. Additionally, d escribe why the MODEL_PATH environment variable was necessary and ho w the containerized application accesses the model file that was download ed during the workflow.

Handing in this assignment

You can hand this in by duplicating this document on Notion, print this document as a `.pdf` and submit that document on Leho.

Also hand in the written Source Code in a `.zip` file please.

Checkboxes:

- [x] ~~I have duplicated this file~~
- [x] ~~I have filled in all the answers~~
- [x] ~~I have added something that I learned~~
- [x] ~~I have added three interesting exam questions~~
- [x] ~~I have zipped my project and uploaded it to Leho~~
- [ ] I turned off all the Azure services I don’t need anymore, to save some costs.