

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

[Introduction](#)

[Objective](#)

Knowledge

Skills

Necessities

GitHub Actions Pt. 1 — Prepare and Train

1. Compute — Creation

2. Compute — Start

3. Environments & Components

4. Pipeline in Azure

GitHub Actions Pt. 2 — Download trained model

GitHub Actions Pt. 3 — Deployment to Kubernetes

Kubernetes deployment

Alternative: Deploy to HuggingFace

Best Practices - Professionalising the pipeline

Pipeline Controlling

Version Controlling

Debugging GitHub Actions

Extra: Adding Responsible AI

Rounding off

What did you learn?

Give three interesting exam questions

Handing in this assignment

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:
- `Azure/CLI` — Current version:

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.
```



QUESTION 2 - Robust Pipelines: Compute - *Extra*

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

Provide a better solution using checks and if-statements.



ANSWER 2 - Robust Pipelines: Compute - *Extra*

```
# Answer here with a YAML snippet
```

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

```
# Answer here with a YAML snippet
```

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 location=$LOCATION
    VERSION=$(az ml model list -n animal-classification --query "[0].version" -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: ...
2. `needs` : ...
3. `upload-artifact` : ...

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

Paste your Kubernetes deployments here

Paste your pipeline steps here

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

```
# Paste your pipeline steps here
```

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. *E.g.: My first suggestions would be to use a separate Secret managing system like Azure Key Vault instead of the GitHub Secrets.*
- 2.

What did you learn?

Fill in something that you learned during this lesson

| ...
| ...

Give three interesting exam questions

1. ...
2. ...
3. ...

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:

- I have duplicated this file
- I have filled in all the answers
- I have added something that I learned
- I have added three interesting exam questions
- 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.

