AI Intro Basics #4

MLOps – Machine Learning Operations

Roland Potthast, Marek Jacob, Daniele Nerini

EUMETNET

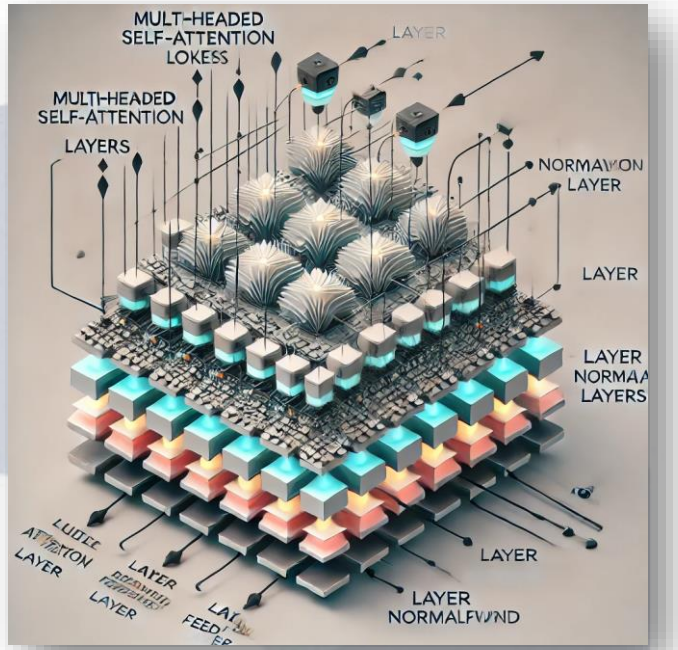
E-AI Basic Tutorials

Tutorial E-AI Basics 4: January Wednesday 22, 2025, 11-12 CET
"MLOps" - Machine Learning Operations
4.1 Overview (20') [RP]
4.2 MLOps in relation to traditional Weather forecasting (20') [MJ]
4.3 Road to MLOps (20') [DN]

4.1

Tutorial E-AI Basics 5: February Wednesday 19, 2025, 11-12 CET
MLflow - an open-source platform for managing the machine learning lifecycle
5.1 Overview - User perspective (20') [TG]
5.2 Logging to MLflow as a ML software developer (20') [HT]
5.3 Running MLflow server as a user and as a service (20') [MJ]

Tutorial E-AI Basics 6: February Wednesday 26, 2025, 11-12 CET
CI/CD - Continuous Integration and Continuous Deployment of ML codes
6.1 Overview – What can CI/CD do for you? (20') [MJ]
6.2 Basic tests with Pytest (20') [JD]
6.3 Setting up a runner (20') [FP]



AI generated Images,
© Roland Potthast 2024

What is DevOps?

DevOps is an approach that fosters collaboration between development (Dev) and operations (Ops) within an organization. The goal is to reduce the time from development to operation by enhancing communication, automation, and quality assurance.

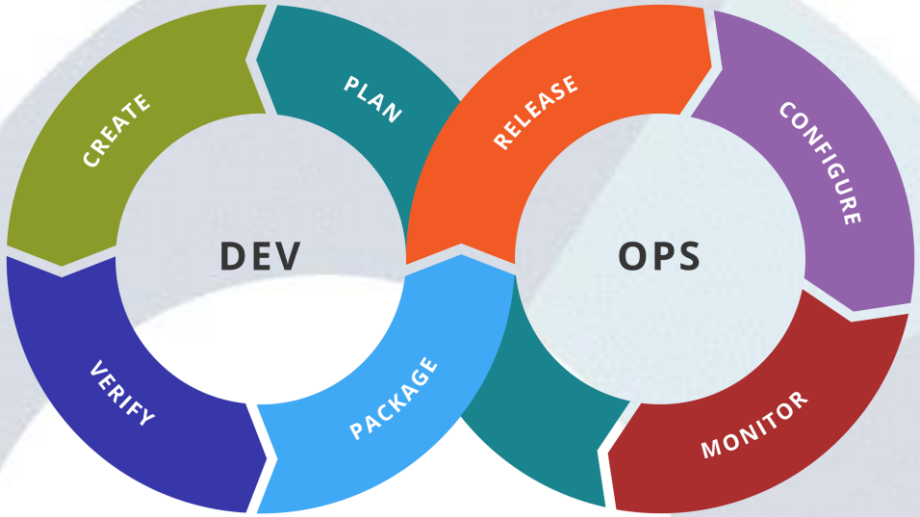
DevOps in Weather and Climate Applications and Services

In the context of weather and climate applications and services, DevOps enables close collaboration between ML model developers and service operators. This ensures that models and systems are:

Accurately implemented for operational use.

Supported by automated processes, such as:

- **Data integration**
- **Model training**
- **Validation**
- **Deployment**



What is ML Ops?

ML Ops is a specialized approach focused on managing machine learning models—from training and validation to deployment and monitoring. The goal is to improve model quality, reliability, and scalability throughout its lifecycle.

ML Ops in Weather and Climate Applications and Services

Operators ensure that ML models are:

Correctly trained and validated.

Monitored for performance, with error identification and corrective actions.

Scalable and robust to handle operational demands.

Key Practices

- ✓ **Model Versioning**: Managing different versions and tracking performance.
- ✓ **Automated Testing**: Ensuring model quality through continuous testing.
- ✓ **Continuous Integration & Deployment**: Seamless updates to models.
- ✓ **Data Security**: Safeguarding data used for training and operation.

From Research to Operations – Building Blocks I

1. Version Control and Source Code Management

Use Case: Manages code changes and enables team collaboration (e.g., Git).

Considerations: Branching strategies (e.g., GitFlow), conflict resolution, versioning infrastructure code.

2. CI/CD (Continuous Integration/Continuous Deployment)

Use Case: Automates the build, test, and deployment processes.

Considerations: Test coverage, automation tools (e.g., Jenkins, GitHub Actions), deployment strategies (e.g., Blue-Green Deployments).

3. Infrastructure as Code (IaC)

Use Case: Defines and manages infrastructure programmatically (e.g., Terraform, Ansible).

Considerations: Repeatability, modularity, handling sensitive credentials securely.

4. Monitoring and Logging

Use Case: Monitors systems and applications (e.g., Prometheus, Grafana, ELK Stack).

Considerations: Define KPIs, avoid logging sensitive data, and set up alerts for critical issues.

5. Automation

Use Case: Reduces manual tasks (e.g., testing, deployments, updates).

Considerations: Write effective scripts, integrate workflows, and choose suitable automation tools.

From Research to Operations – Building Blocks II

6. Collaboration and Communication

Use Case: Improves transparency and teamwork between development and operations.

Considerations: Use tools like Slack or Rocket Chat, establish feedback loops, and define clear responsibilities.

7. Security (DevSecOps)

Use Case: Integrates security into development from the start.

Considerations: Automate security checks, manage vulnerabilities, and enforce access controls.

8. Containerization and Orchestration

Use Case: Standardizes applications and deployment processes (e.g., Docker, Kubernetes).

Considerations: Resource allocation, scalability, and managing network communication.

9. Feedback and Continuous Improvement

Use Case: Drives improvement through feedback loops and analytics.

Considerations: Gather data, hold retrospectives, and define metrics to measure success.

10. Scalability, Reliability and Portability

Use Case: Ensures systems are flexible and robust.

Considerations: Redundancy, load testing, and implementing auto-scaling strategies.

Versioning of Code - Example Git Playground 1

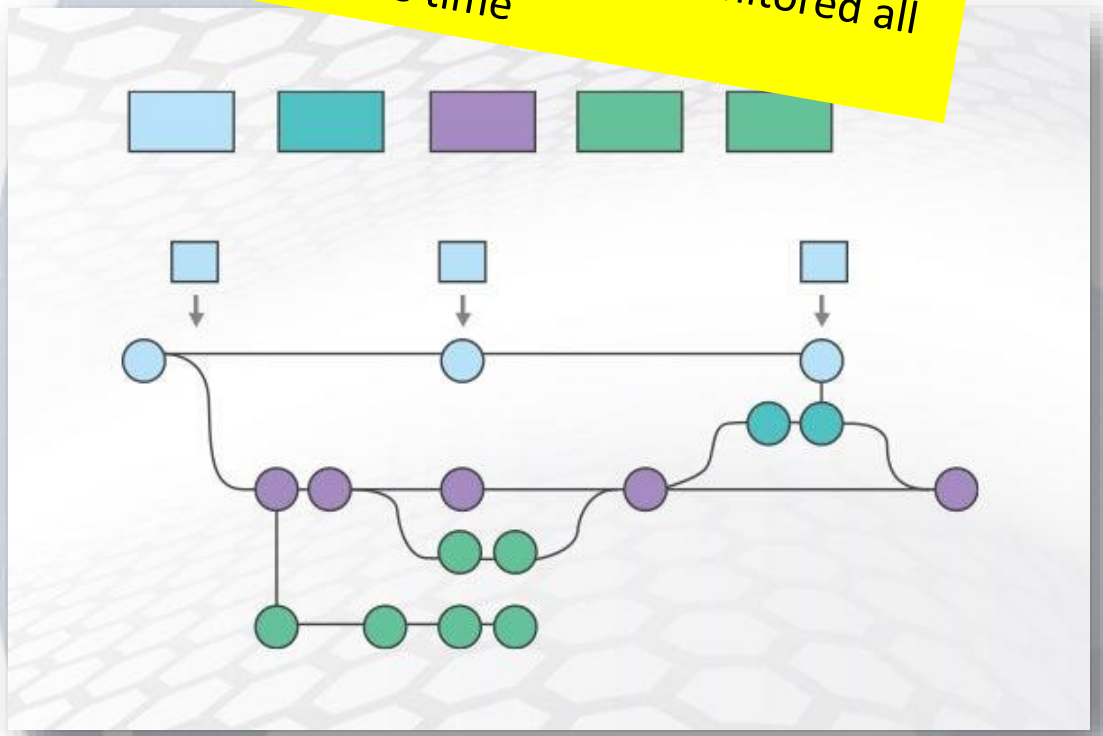
- Install **git bash** on your computer (e.g. windows), open a bash
- Let us generate a **bare git repository**

```
mkdir git_demo_store
cd git_demo_store/
git init --bare
```

- **Clone your repo** to get a working repo

```
git clone git_demo_store git_demo_work
Cloning into 'git_demo_work'...
warning: You appear to have cloned an empty repository.
done.
```

USE Git repositories to make sure
a) Code is always saved on another computer
b) All changes are monitored all the time



Example Git Playground 2: bare repo and cloned repo

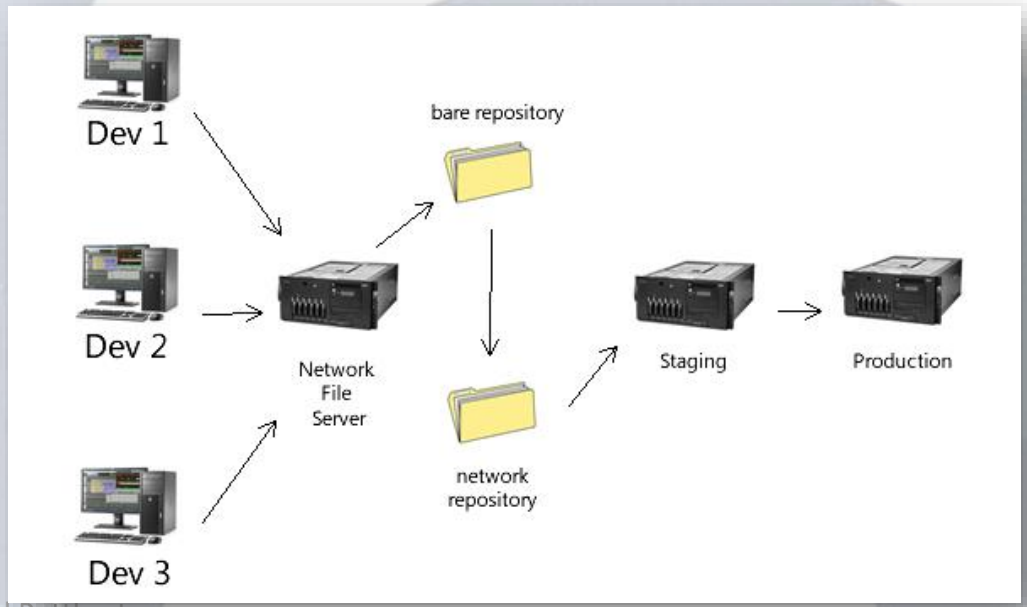
- We now have a **bare repo** (which is a full repo) and a **cloned repo** for work. This can also be done via ssh, and you can use the base repo as your reference and work with various work repos on other computers.

```
rpotthas@oofnbbe532 MINGW64 /d/e-ai_tutorials/tutorial4 (main)
$ ll -h
total 3.4M
-rw-r--r-- 1 rpotthas 1049089 3.4M Jan 20 09:06 E-AI_Talks_Basics_04_MLOps.pptx
drwxr-xr-x 1 rpotthas 1049089    0 Jan 20 08:55 git_demo_store
drwxr-xr-x 1 rpotthas 1049089    0 Jan 20 09:03 git_demo_work
```

- Add a demo file, commit and push

```
echo "Hello, World!" > demo.txt

git add demo.txt
git commit -m "Initial commit"
git push
```

Play around with git add, checkout, push and also explore how to restore previous versions, how to monitor commits etc …



CI-CD trigger: Example Git Playground 3: post-receive

- In hooks/ directory of the base repo

  `git_demo_store`

generate a file

  `post-receive`

with the content on the right.

Updates the git_demo_work repo when pushing from git_demo_work2

```
#!/bin/bash

# Simple post-receive hook demo
WORK_TREE="/d/e-
ai_tutorials/tutorial4/git_demo_work"
GIT_DIR="$(pwd)"  # Automatically set to
the path of the base repository

echo "Post-receive hook triggered.
Updating work tree..."
git --work-tree="$WORK_TREE" --git-
dir="$GIT_DIR" checkout -f
echo "Work tree updated successfully."
```

You might need to set file rights to executable:

chmod +x /d/e-ai_tutorials/tutorial4/git_demo_store/hooks/post-receive

Example Git Playground 4: post-receive trigger

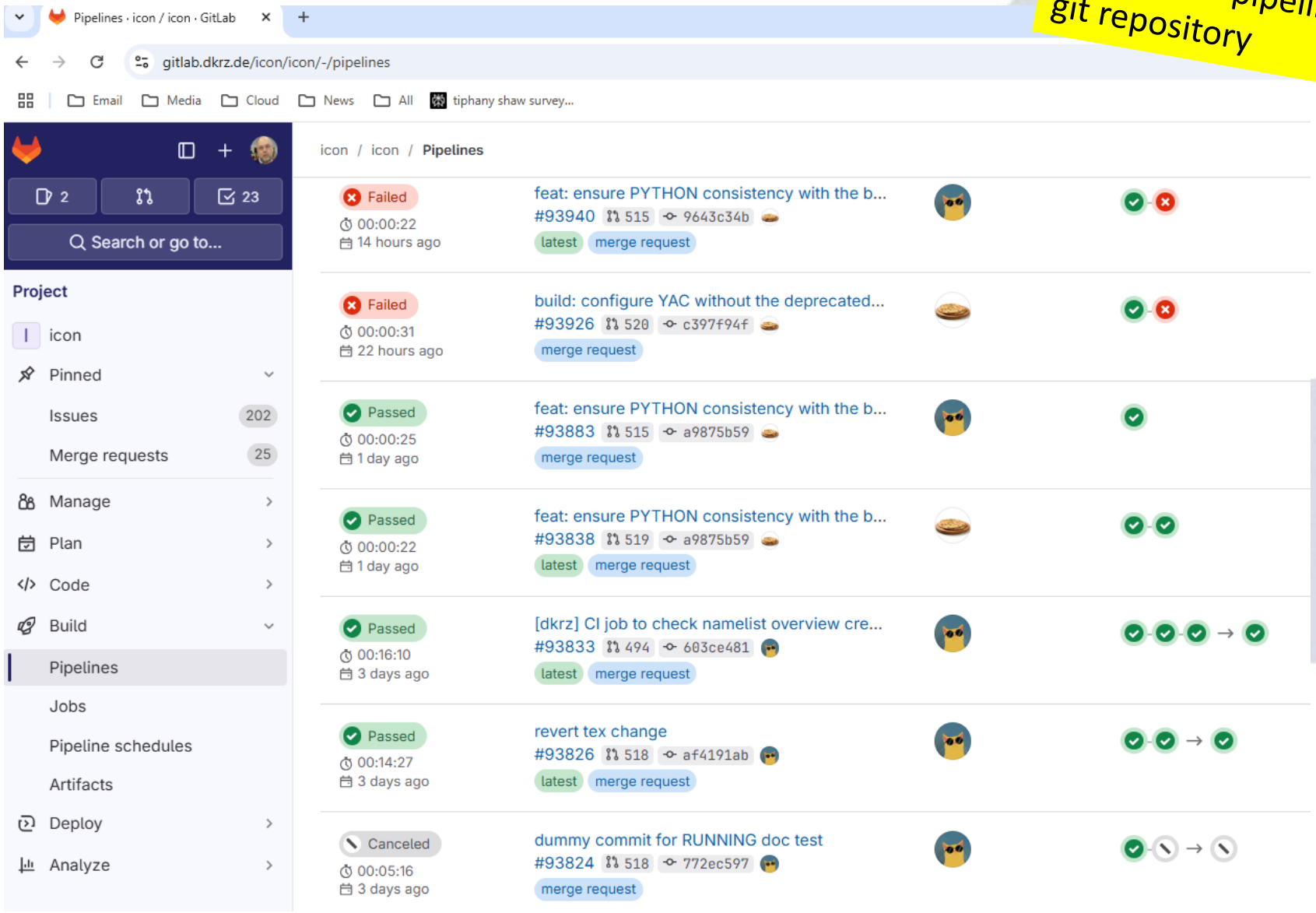
- Now clone the repo another time into **git_demo_work2**
- Change the demo.txt in the second git clone, add, commit and push.
- You get an output such as:

```
rpotthas@oofnbbe532 MINGW64 /d/e-ai_tutorials/tutorial4/git_demo_work2
(master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 297 bytes | 297.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Post-receive hook triggered. Updating work tree...
remote: Work tree updated successfully.
To D:/e-ai_tutorials/tutorial4/git_demo_store
   c9be361..2b2fd50  master -> master
```

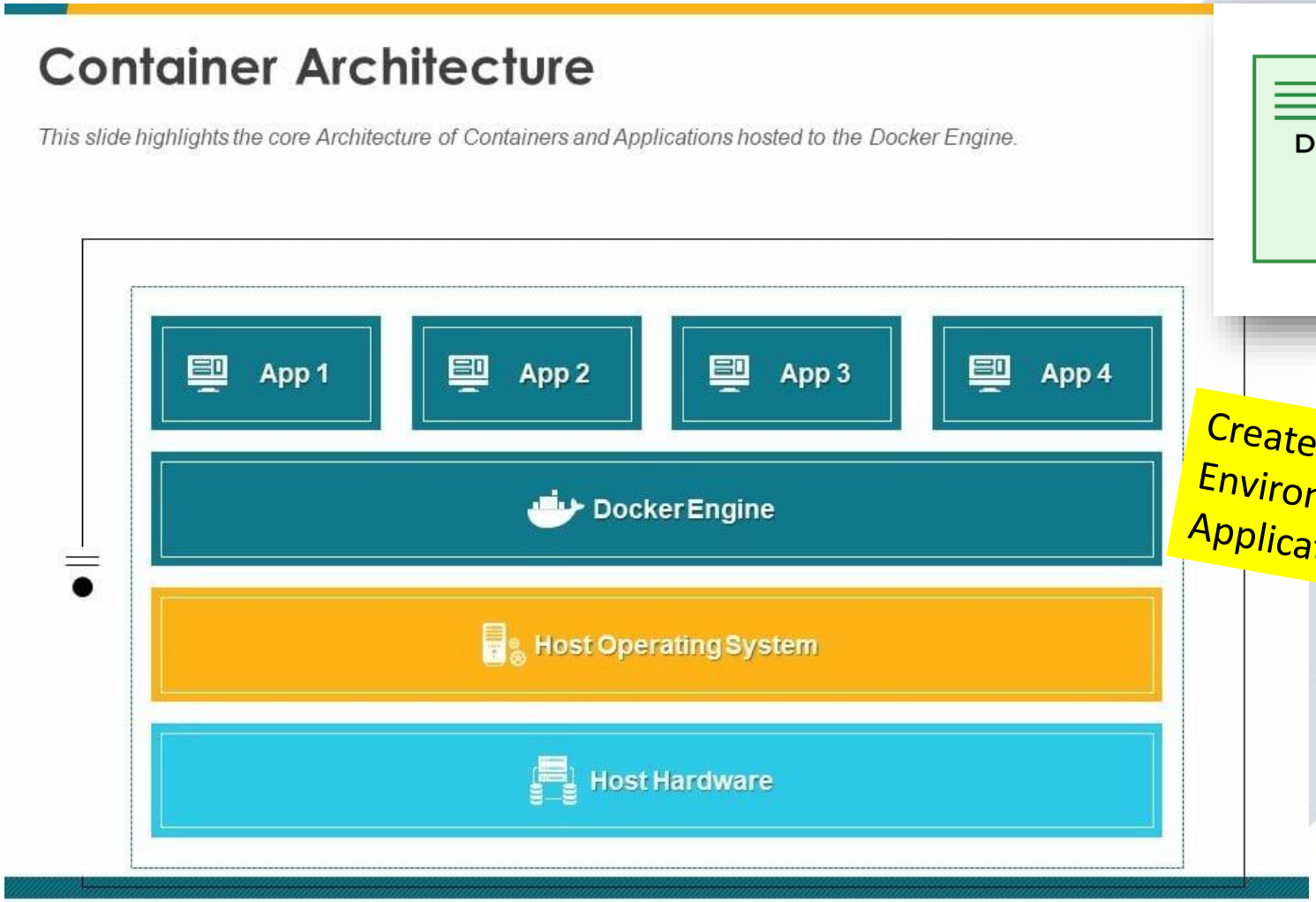
Updates the git_demo_work repo when pushing from git_demo_work2

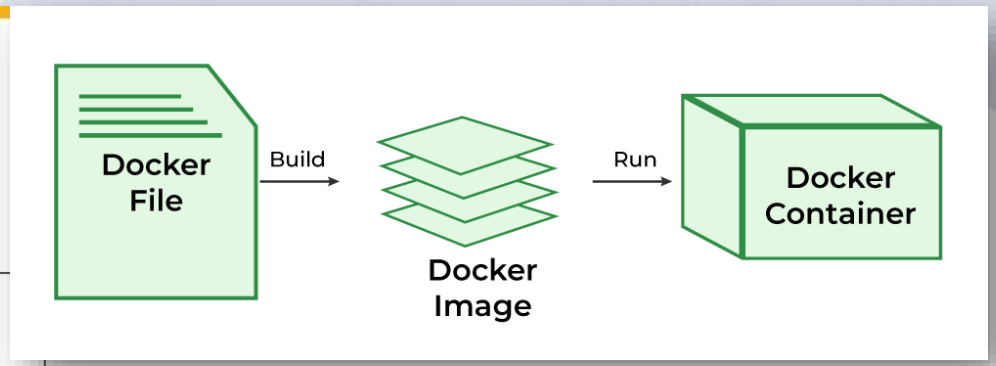
Gitlab CI-CD Pipeline

Integrate continuous integration pipeline into git repository



Infrastructure as Code (IaC) / Container Example: Dockerfile





Create a controllable Environment for your Application

Infrastructure as Code (IaC) / Container Example: Dockerfile

Make sure docker is installed on your machine.

Docker Container are build based on descriptive code, the Dockerfile

```
# Use Ubuntu as the base image
FROM ubuntu:20.04
# Ubuntu already includes Bash, but we'll ensure updates
RUN apt-get update && apt-get install -y bash && apt-get clean
# Set the default command to start Bash
CMD ["bash"]
```

Dockerfile

Build Container

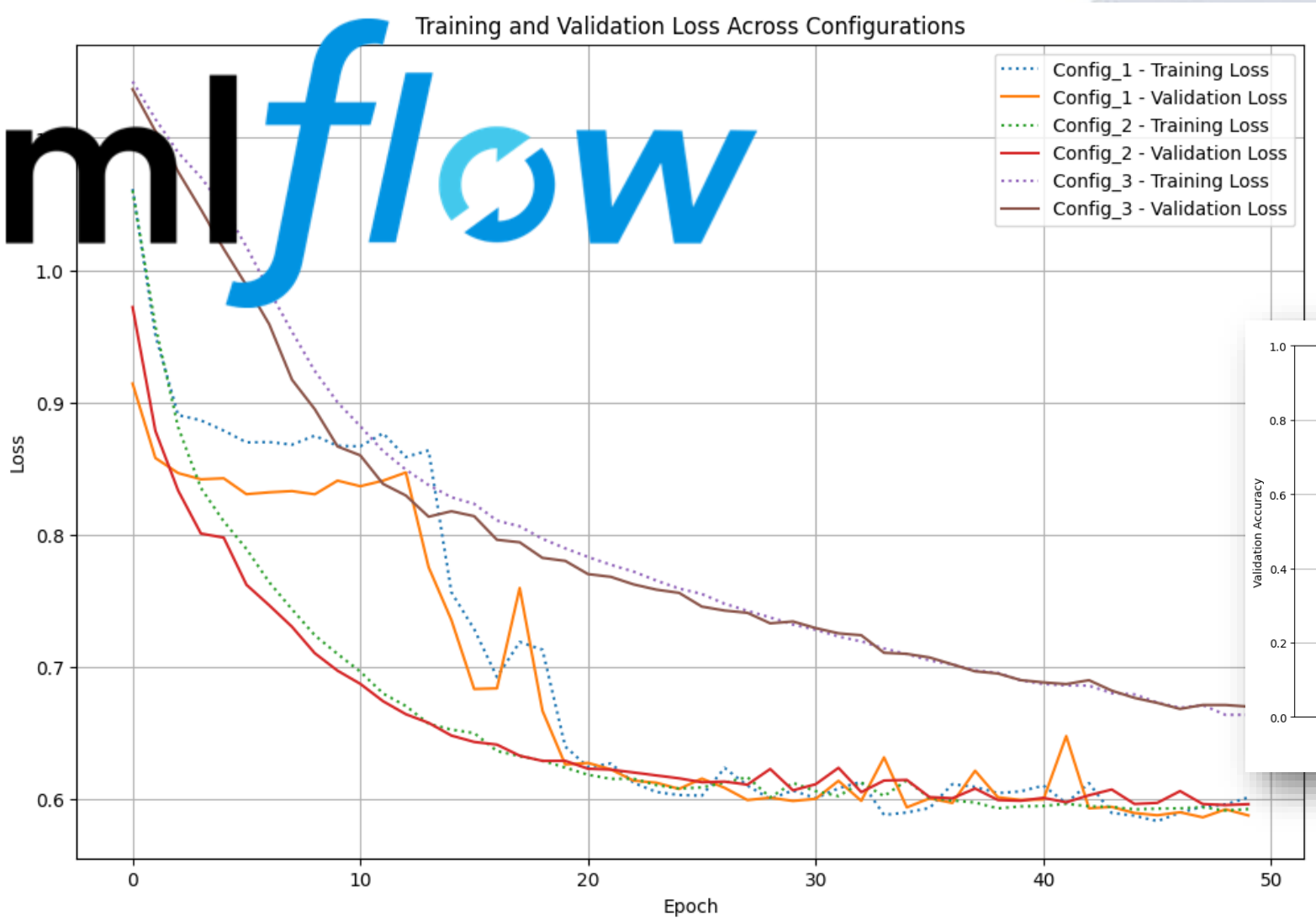
Run Container

```
docker build -t my-ubuntu-bash-container .

docker run -it my-ubuntu-bash-container
```

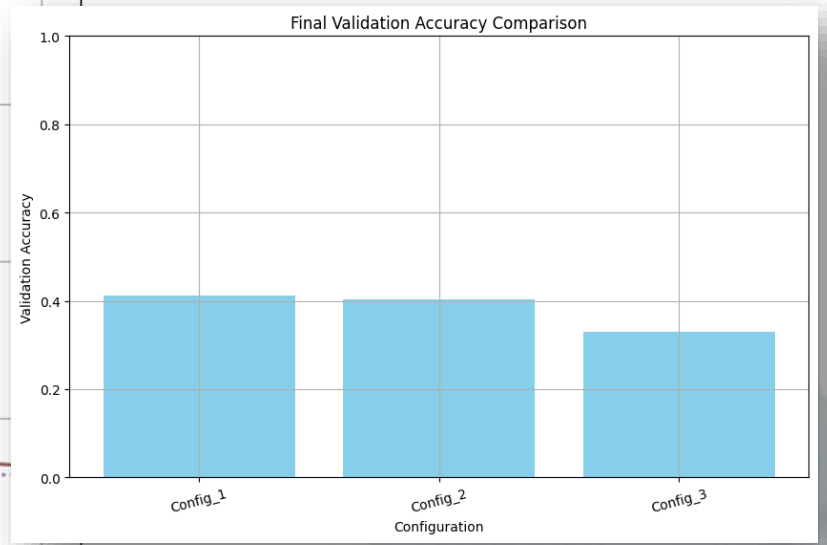
Docker File for Machine Learning Applications

```
# Use an official PyTorch base image
FROM pytorch/pytorch:2.0.1-cuda11.8-cudnn8-runtime

# Install Jupyter and other dependencies
RUN apt-get update && apt-get install -y python3-pip && \
    pip install --no-cache-dir notebook && \
    apt-get clean

# Expose the port for Jupyter Notebook
EXPOSE 8888

# Set the default command to start Jupyter Notebook
CMD ["jupyter", "notebook", "--ip=0.0.0.0", "--port=8888", "--
no-browser", "--allow-root"]
```

Monitoring ML Training and Validation and Deployment



Framework for logging and displaying Training and Validation



Store or communicate losses during training

Configurations

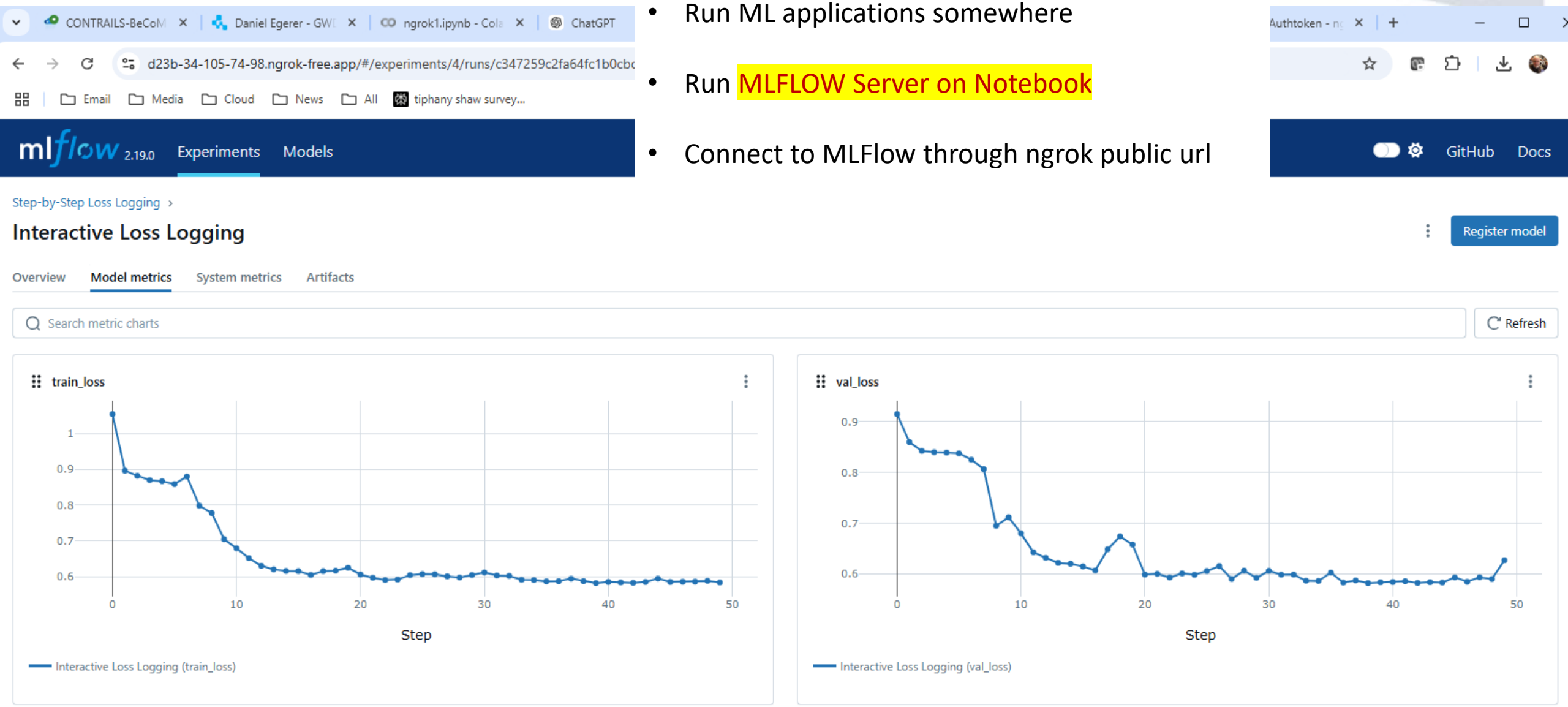
```
# Define configurations to compare
configurations = [
    {"hidden_dim": 64, "learning_rate": 0.01, "batch_size": 16, "epochs": 50, "name": "Config_1"},
    {"hidden_dim": 128, "learning_rate": 0.005, "batch_size": 32, "epochs": 50, "name": "Config_2"},
    {"hidden_dim": 32, "learning_rate": 0.001, "batch_size": 8, "epochs": 50, "name": "Config_3"},
]

# Train and log each configuration
for config in configurations:
    with mlflow.start_run(run_name=config["name"]):
        # Log metrics for this epoch
            mlflow.log_metric("train_loss", train_loss, step=epoch)
            mlflow.log_metric("val_loss", val_loss, step=epoch)

        # Log final metrics and parameters
        mlflow.log_param("hidden_dim", config["hidden_dim"])
        mlflow.log_param("learning_rate", config["learning_rate"])
        mlflow.log_param("batch_size", config["batch_size"])
        mlflow.log_param("epochs", config["epochs"])

        # Save metrics to JSON for visualization
        results = {"train_loss": train_losses, "val_loss": val_losses}
        with open(f"{output_dir}/{config['name']}_metrics.json", "w") as f:
            json.dump(results, f)
        print(f"Metrics saved for {config['name']}.")
```

Log Parameters

Log Losses

Easy Setup of ML Flow e.g. on Colab using ngrok

- Run ML applications somewhere
- Run MLFLOW Server on Notebook
- Connect to MLFlow through ngrok public url



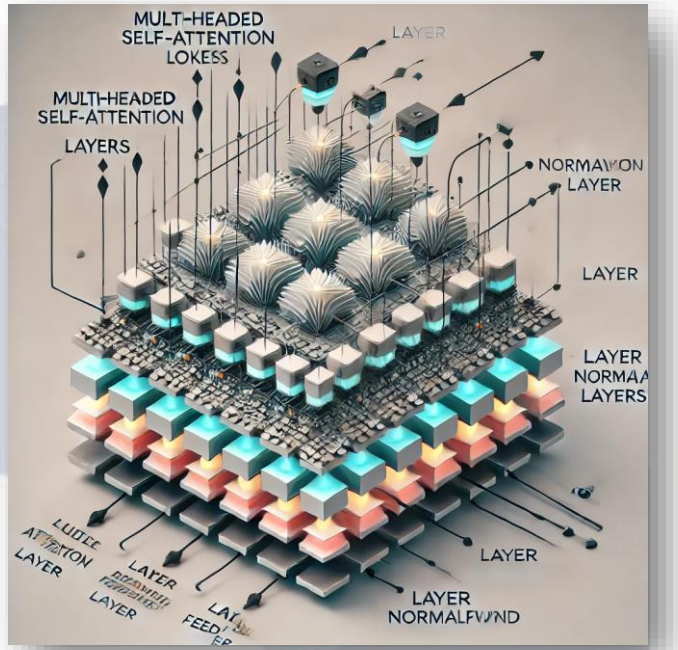
E-AI Basic Tutorials

Tutorial E-AI Basics 4: January Wednesday 22, 2025, 11-12 CET
"MLOps" - Machine Learning Operations
4.1 Overview (20') [RP]
4.2 MLOps in relation to traditional Weather forecasting (20') [MJ]
4.3 Road to MLOps (20') [DN]

4.2

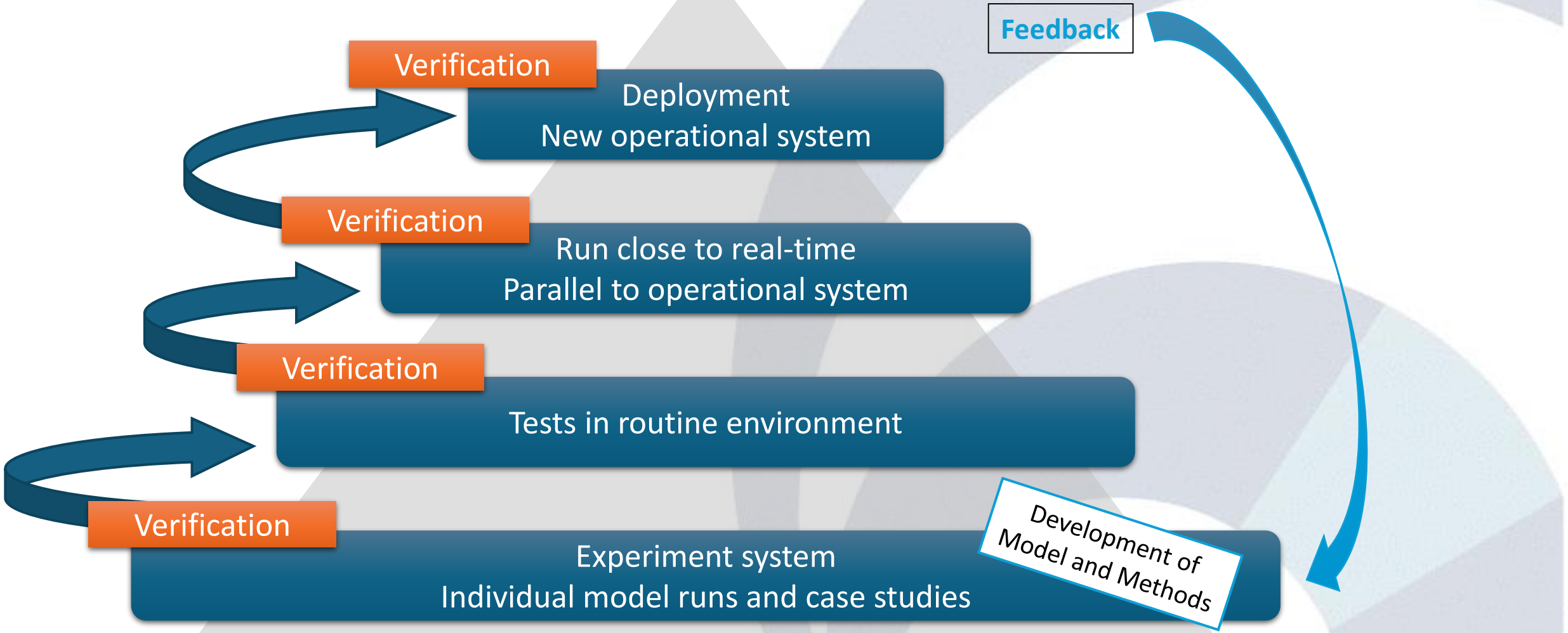
Tutorial E-AI Basics 5: February Wednesday 19, 2025, 11-12 CET
MLflow - an open-source platform for managing the machine learning lifecycle
5.1 Overview - User perspective (20') [TG]
5.2 Logging to MLflow as a ML software developer (20') [HT]
5.3 Running MLflow server as a user and as a service (20') [MJ]

Tutorial E-AI Basics 6: February Wednesday 26, 2025, 11-12 CET
CI/CD - Continuous Integration and Continuous Deployment of ML codes
6.1 Overview – What can CI/CD do for you? (20') [MJ]
6.2 Basic tests with Pytest (20') [JD]
6.3 Setting up a runner (20') [FP]

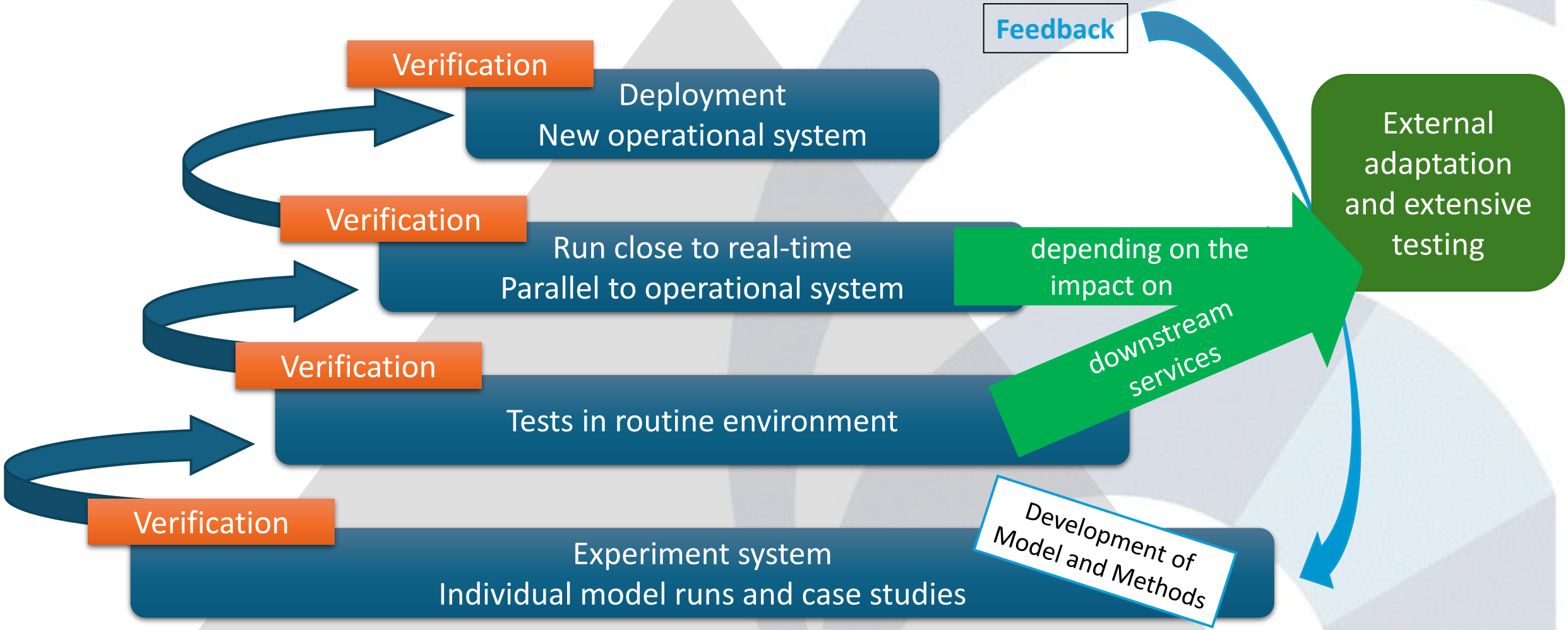


AI generated Images,
© Roland Potthast 2024

Weather Forecasting Innovation Process



Weather Forecasting Innovation Process



Weather-Forecasting-System Update Cycle

- Frequency/Agility of update process depends on

Strategic Goals

Versioning system
- Code
- Changelog

User Communication

Involved Departments
- Responsibilities

Dependencies
Identification of
- Interfaces
- Incremental Updates
- Breaking Updates

Degree of Automation
- Technical test
- Forecast cycling
- Score verification
- Communication

From Research to Operations in traditional NWP – Building Blocks I

1. Version Control and Source Code Management

Use Case: Manages code changes and enables team collaboration (e.g., Git).

NWP Code is controlled in one or the other way (git, SVN, VCS, enumerated directories...)

2. CI/CD (Continuous Integration/Continuous Deployment)

Use Case: Automates the build, test, and deployment processes.

Depends on the NWP Model or Component.

- Automation build into version control system
- Automated via additional tools like Jenkins

Test suite

- Centralized set of manual experiments
- Decentralized set of manual experiments

3. Infrastructure as Code (IaC)

Use Case: Defines and manages infrastructure programmatically (e.g., Terraform, Ansible).

Classically software is pre-installed on the HPC
Some flexibility with module system
Slowly changing requirements

4. Monitoring and Logging

Use Case: Monitors systems and applications (e.g., Prometheus, Grafana, ELK Stack).

System monitoring by system department or vendor

5. Automation

Execution of NWP model, i.e. forecast and related tasks are usually automated (e.g. ecflow)

Deployment, Updates,
... are usually manual or semi-manual

From Research to Operations in traditional NWP – Building Blocks II

6. Collaboration and Communication

Use Case: Improves transparency and teamwork between development and operations.

Considerations: Use tools like Slack or Rocket chat, establish feedback loops, and define clear responsibilities.

7. Security (DevSecOps)

- Code is self-developed or from trustworthy partners
- NWP code has fewer dependencies than a modern Python environment
- Access control through POSIX filesystem and/or separate operational systems

8. Containerization and Orchestration

Not extensively used in HPC NWP
Challenge: MPI in container would depend on the hardware/vendor

Considerations: Resource allocation, scalability, and managing network communication.

9. Feedback and Continuous Improvement

Use Case: Drives improvement through feedback loops and analytics.

Continuous feedback by forecasters and research community

10. Scalability, Reliability and Portability

Use Case: Ensures systems are flexible and robust.

NWP system often tightly bound to HPC setup

Key novelties in AI/ML – Model Reproducibility

- ML model consists of model structure (code) and **weights**
  - Whole models can be serialised into a single binary file
  - ➢ Generic runtime becomes inference application

➢ Key factors for Model-Versioning and Reproducibility

- **Data Lineage**
  - Which data was used in the training?
  - Which data at which stage?
  - Pre-processing?
- **Implementation**
- **Training Settings**
  - Trained from scratch or using pre-trained model?
- **Randomness**
  - Does learning process involve randomness?
  - Can a seed be used to control it?

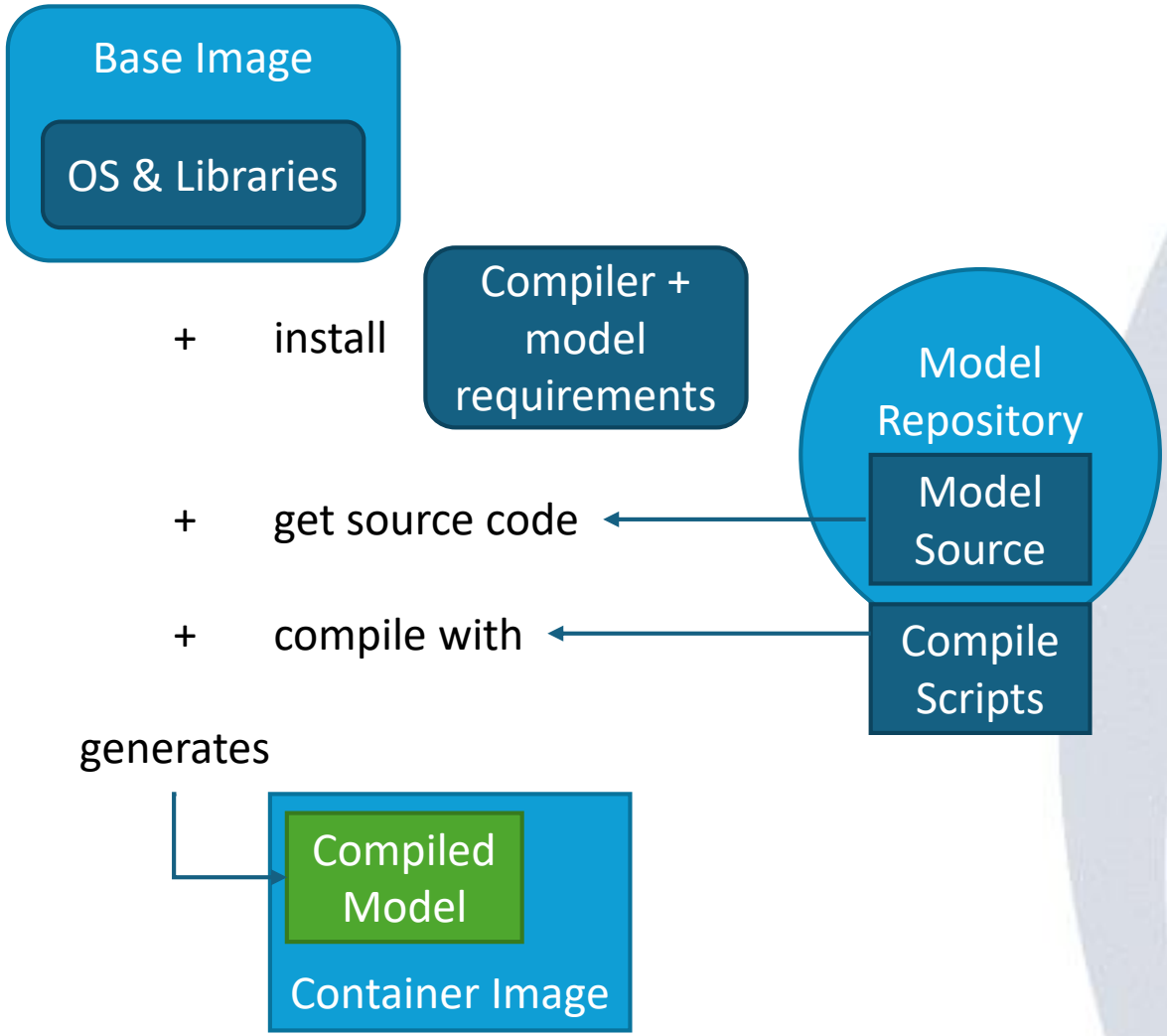
- Good documentation is key for continuous improvement
- Many aspects can be tracked with tools like MLflow

Key novelties in AI/ML – Software Environment

- ML models build on the shoulders of giants
  - Extended use of external software packages
    - ➢ More complex environment
  - Unprecedented pace of development
    - ➢ It's possible to build valuable services quickly
- Need for flexible code execution
  - Huge resource-imbalance between training and inference
  - ➢ Training on external infrastructure

➢ Common solution: Containers
"Infrastructure as Code"

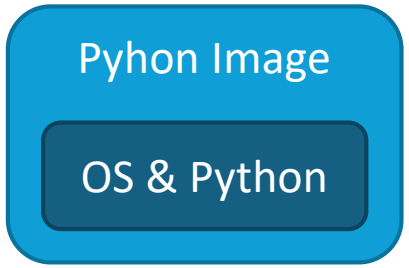
Infrastructure as Code – Numerical Model




```
FROM ubuntu

# install compiler and requirements
RUN apt-get update \
 && apt-get install -y \
    gcc g++ gfortran \
    libc-dev libnetcdff-dev \
 && rm -rf /var/lib/apt/lists/*

RUN git clone "$MODEL_REPO" /tmp/model

# get other prerequisites
COPY scripts/* /tmp/scripts/

RUN cd /tmp/model \
    && /tmp/scripts/build_model.sh
```

Simplified Dockerfile

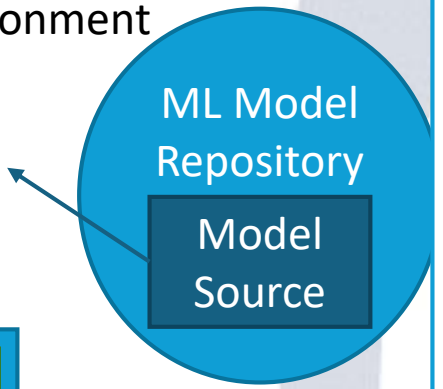
Infrastructure as Code – ML Model



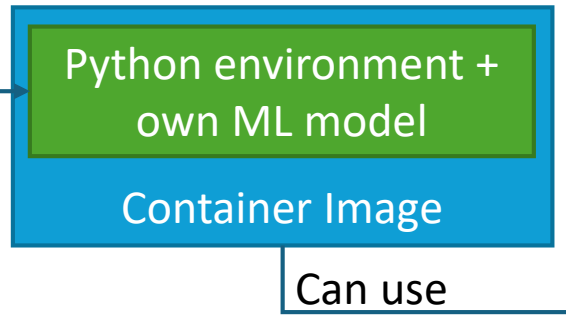
+ compile eccodes (or get it via pip)

Create user

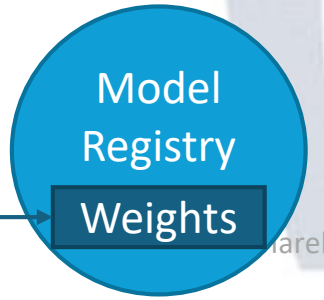
+ create virtual python environment
+ install pytorch
+ clone and install ML Code



generates



Can use




```
FROM python:3.11-slim

# compile eccodes
COPY dockerfiles/provision.eccodes.sh .
RUN bash ./provision. eccodes.sh

RUN useradd -ms /bin/bash user
USER user

WORKDIR /home/user

ENV VIRTUAL_ENV=/home/user/venv
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

# create a new virtual environment
RUN cd /home/user \
 && python3 -m venv --system-site-packages $VIRTUAL_ENV

# install pytorch for CUDA (Nvidia)
RUN pip install --upgrade pip \
 && pip install --no-cache-dir \
        torch==2.1.1+cu118 \
        torchvision==0.16.1+cu118 \
        --extra-index-url \
           https://download.pytorch.org/whl/cu118

RUN git clone "$ML_REPO" /home/user/ml_model \
 && pip install -r /home/user/ml_model/requirements.txt
```

Simplified Dockerfile

provision.eccodes.sh

```
set -e

apt-get update && apt-get install -y \
    wget \
    python3 \
    gcc g++ gfortran \
    libc-dev \
    python3-dev python3-venv \
    git \
    cmake \
    make \
    libaec-dev \
    perl \
 && rm -rf /var/lib/apt/lists/*

wget -q https://confluence.ecmwf.int/download/attachments/45757960/eccodes-2.33.0-Source.tar.gz

tar xzf eccodes-2.33.0-Source.tar.gz
rm eccodes-2.33.0-Source.tar.gz
cd eccodes-2.33.0-Source && mkdir build
cd build && cmake .. -DCMAKE_INSTALL_MESSAGE=NEVER
make -j$(grep processor /proc/cpuinfo | wc -l)
make install VERBOSE=0
cd ../../ && rm -rf eccodes-2.33.0-Source

# clean up packages that were used only for this build process
apt-get remove -y \
    gcc g++ gfortran \
    libc-dev
apt autoremove -y
rm -rf /var/lib/apt/lists/*
```

Infrastructure Comparison

- Numerical Model: Compiled Binary + O(10) libraries (shared objects)
- ML Model: Python environment with O(10 000) modules

```
FROM ubuntu

# install compiler and requirements
RUN apt-get update \
 && apt-get install -y \
    gcc g++ gfortran \
    libc-dev libnetcdff-dev \
 && rm -rf /var/lib/apt/lists/*

RUN git clone "$MODEL_REPO" /tmp/model

# get other prerequisites
COPY scripts/* /tmp/scripts/

RUN cd /tmp/model \
    && /tmp/scripts/build_model.sh
```

```
FROM python:3.11-slim

# compile eccodes
COPY dockerfiles/provision.eccodes.sh .
RUN bash ./provision. eccodes.sh

RUN useradd -ms /bin/bash user
USER user

WORKDIR /home/user

ENV VIRTUAL_ENV=/home/user/venv
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

# create a new virtual environment
RUN cd /home/user \
 && python3 -m venv --system-site-packages $VIRTUAL_ENV

# install pytorch for CUDA (Nvidia)
RUN pip install --upgrade pip \
 && pip install --no-cache-dir \
```

Simplified Dockerfiles

Infrastructure Comparison

- Numerical Model: Compiled Binary +
- ML Model: Python environment with

Note on Containerization:
Operators and developers can separate concerns in different container layers

```
FROM python:3.11-slim

# compile eccodes
COPY dockerfiles/provision.eccodes.sh .
RUN bash ./provision. eccodes.sh

RUN useradd -ms /bin/bash user
USER user

WORKDIR /home/user

ENV VIRTUAL_ENV=/home/user/venv
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

# create a new virtual environment
RUN cd /home/user \
 && python3 -m venv --system-site-packages $VIRTUAL_ENV

# install pytorch for CUDA (Nvidia)
RUN pip install --upgrade pip \
 && pip install --no-cache-dir \
        torch==2.1.1+cu118 \
        torchvision==0.16.1+cu118 \
        --extra-index-url \
           https://download.pytorch.org/whl/cu118

RUN git clone "$ML_REPO" /home/user/ml_model \
 && pip install -r /home/user/ml_model/requirements.txt
```

SYTEM

General Pytorch Environment

Application

Simplified Dockerfiles

ML Ops Implications

- Many ML Ops aspects are already in place
- New aspects:
  - Documentation of training pipeline
    - Data Lineage tracking
    - Settings and Seeds management
    - (Version control of source code)
  - Implementation of containerization for portable and reproducible services
  - Automation in deployment process
  - Novel tools and techniques require collaboration and teamwork from many individuals across research and development to operations
  - ML models benefit from leveraging tools from a broader community

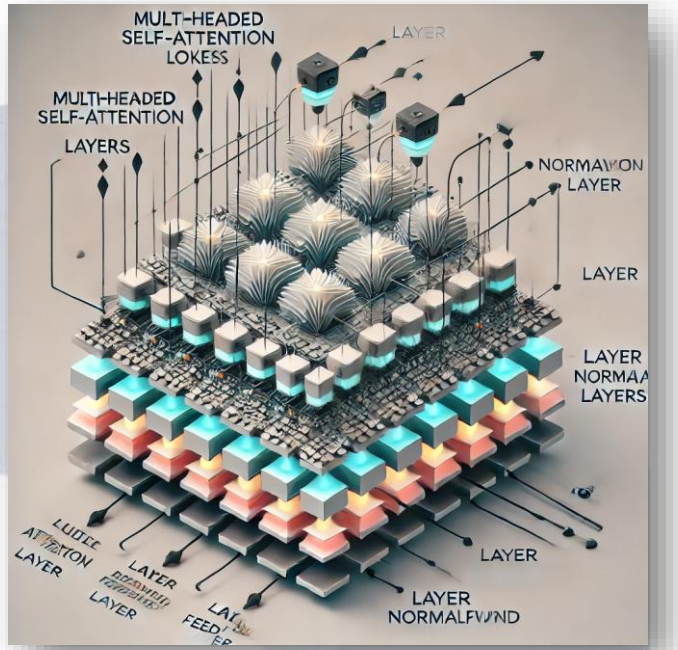
E-AI Basic Tutorials

Tutorial E-AI Basics 4: January Wednesday 22, 2025, 11-12 CET
"MLOps" - Machine Learning Operations
4.1 Overview (20') [RP]
4.2 MLOps in relation to traditional Weather forecasting (20') [MJ]
4.3 Road to MLOps (20') [DN]

4.3

Tutorial E-AI Basics 5: February Wednesday 19, 2025, 11-12 CET
MLflow - an open-source platform for managing the machine learning lifecycle
5.1 Overview - User perspective (20') [TG]
5.2 Logging to MLflow as a ML software developer (20') [HT]
5.3 Running MLflow server as a user and as a service (20') [MJ]

Tutorial E-AI Basics 6: February Wednesday 26, 2025, 11-12 CET
CI/CD - Continuous Integration and Continuous Deployment of ML codes
6.1 Overview – What can CI/CD do for you? (20') [MJ]
6.2 Basic tests with Pytest (20') [JD]
6.3 Setting up a runner (20') [FP]



AI generated Images,
© Roland Potthast 2024

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

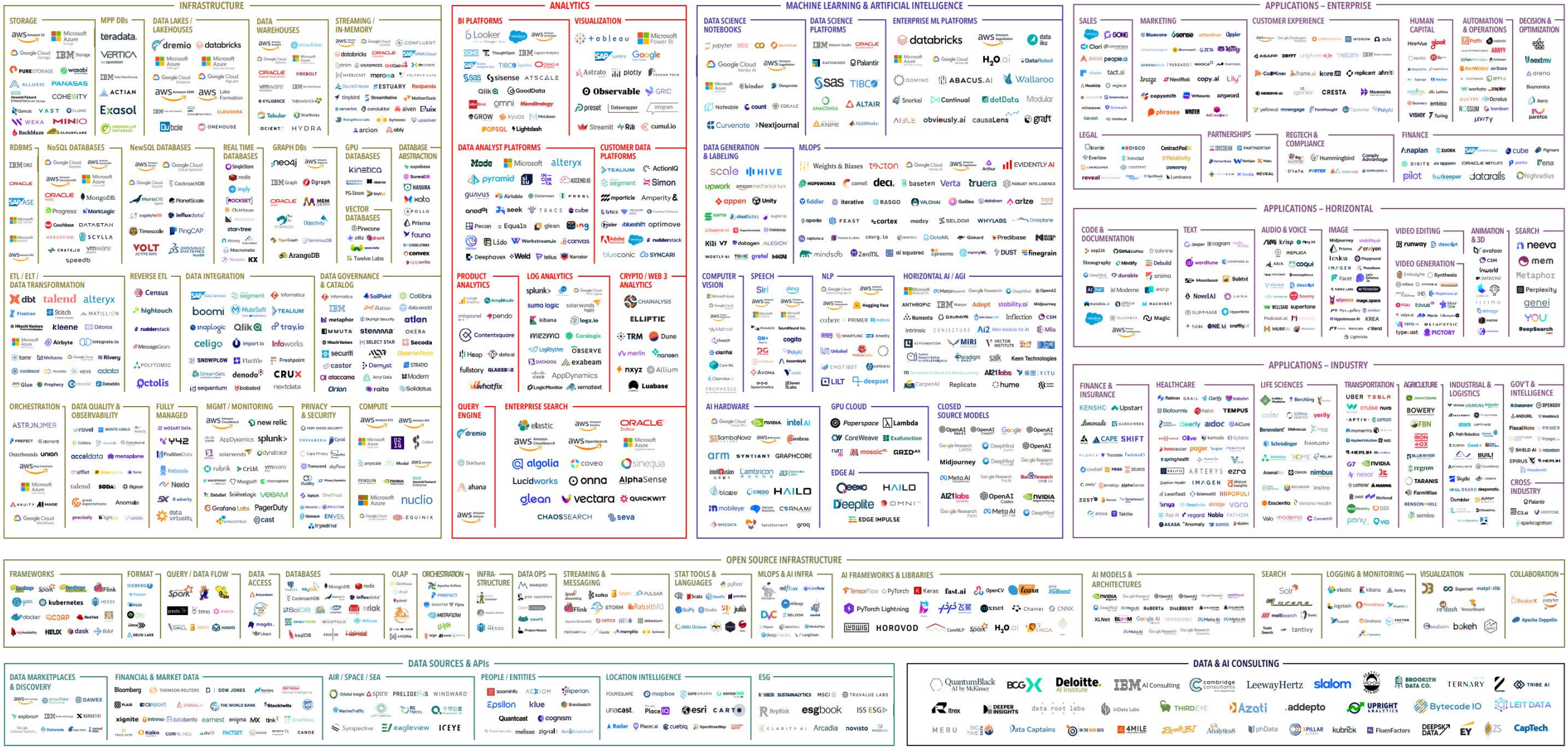
Eidgenössisches Departement des Innern EDI
Bundesamt für Meteorologie und Klimatologie MeteoSchweiz

The Winding Road to MLOps

Tutorial E-AI Basics: MLOps – Machine Learning Operations

Daniele Nerini
Néstor Tarin Burriel
Gabriela Aznar Siguan
Pascal Tay

THE 2023 MAD (MACHINE LEARNING, ARTIFICIAL INTELLIGENCE & DATA) LANDSCAPE



Version 1.0 - Feb 2023

Blog post: mattturck.com/MAD2023

Interactive version: MAD.firstmarkcap.com

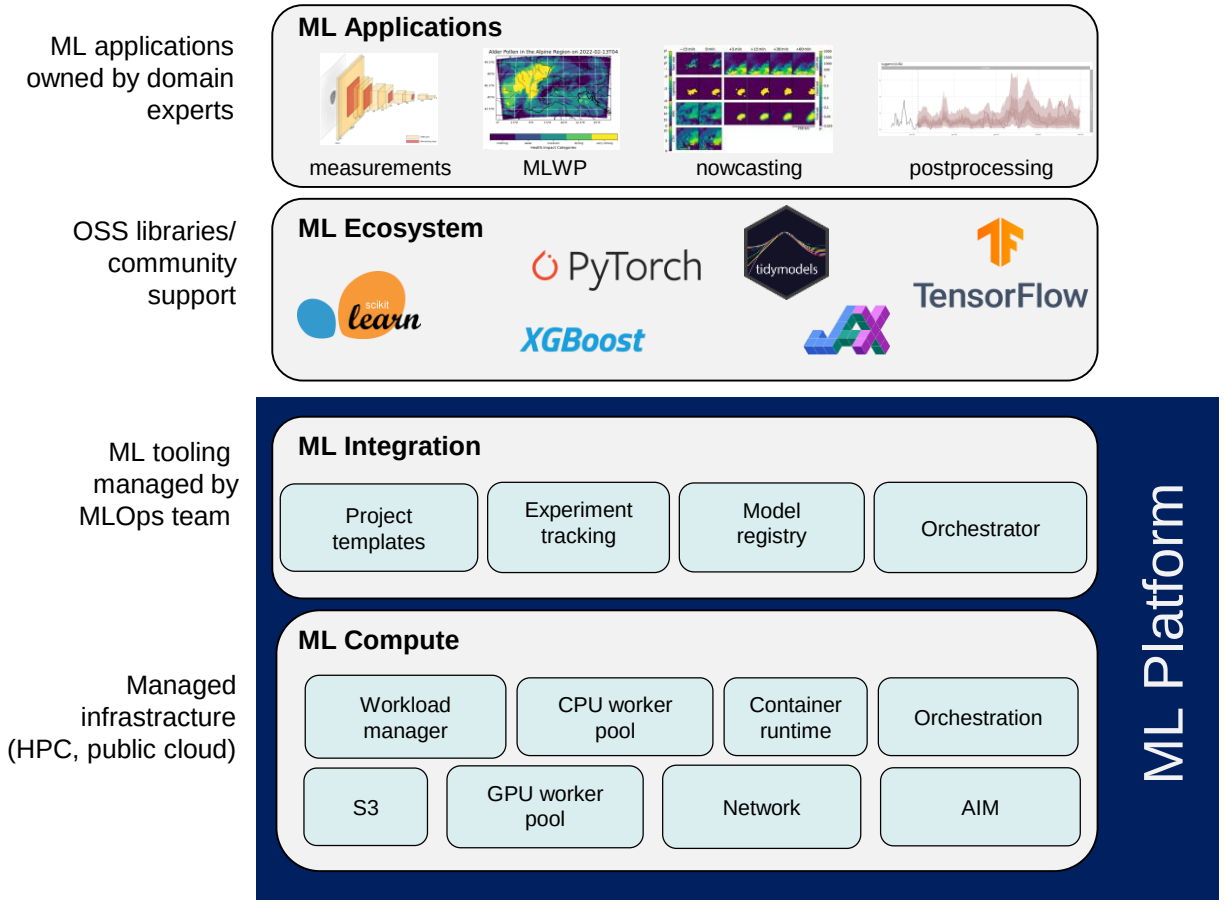
Comments? Email MAD2023@firstmarkcap.com

FIRSTMARK
EARLY STAGE VENTURE CAPITAL

https://mad.firstmark.com/

Vision for Machine Learning Infrastructure

- ML applications are owned by ML scientists at MeteoSwiss.
- ML libraries based on OSS and their communities.
- Integration into production is enforced with blueprints, and well-documented APIs and SDKs, managed by MLOps team.
- Base infrastructure is managed by IT or outsourced to public cloud.



The journey so far …

2022

- PoC1: Kubeflow Hackathon

2023

- PoC2: On-prem ML pipelines

2024

- Design ML Platform architecture
- Plan implementation

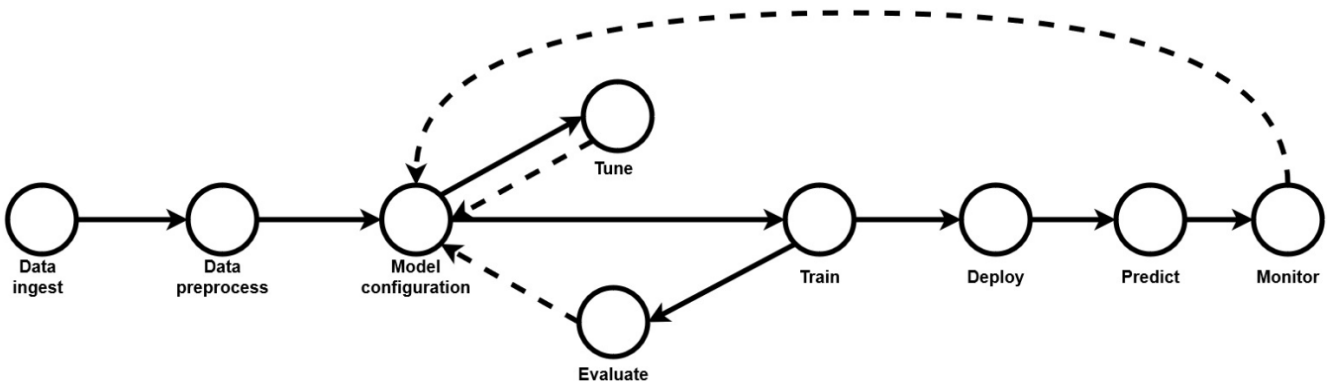
2025

- Go Live!

The road to the St.Gotthard Pass
StudioJS, CC BY-SA 4.0, via Wikimedia Commons

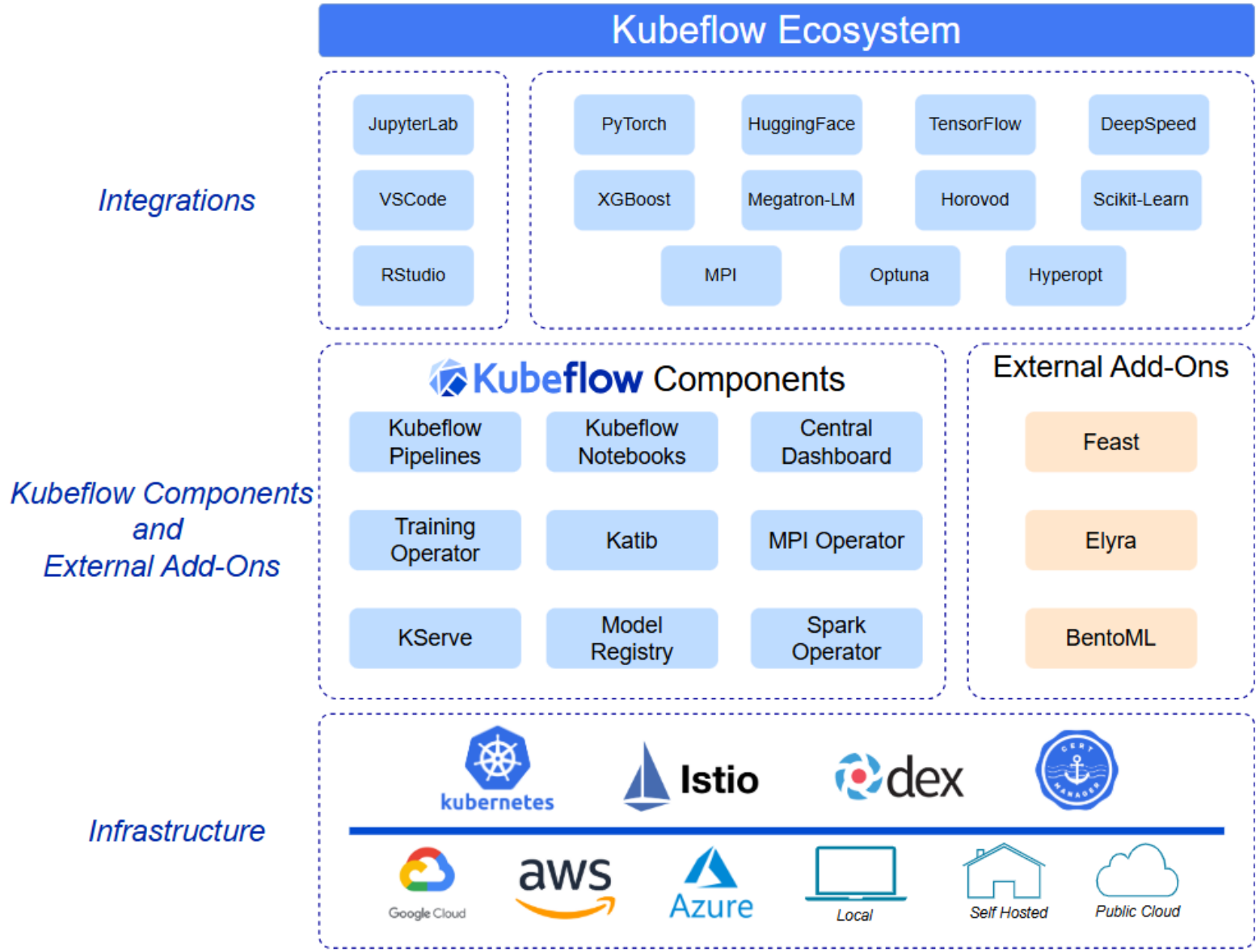
PoC1: Kubeflow Hackathon

- **Goal:** Build an end-to-end ML pipeline on top of Kubernetes.
- **Use case:** Predict horizontal visibility at Zurich Airport (postprocessing)
- **Data:** 1 year of NWP hourly features (forecasts of surface temperature and humidity) and labels (measured visual range in km).



Kubeflow

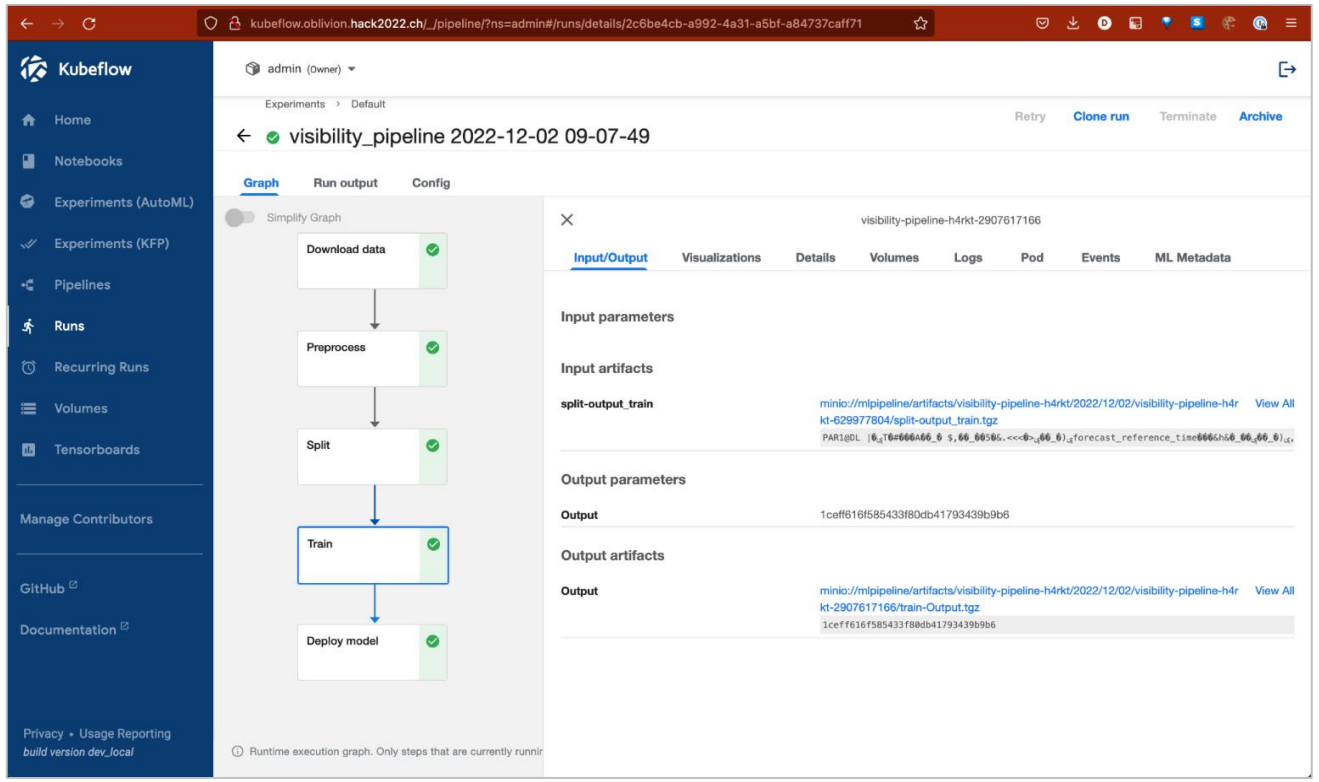
- An ecosystem of open-source projects to address each stage in the ML lifecycle.
- Support for best-in-class open source tools and frameworks.
- Sits on top of Kubernetes.



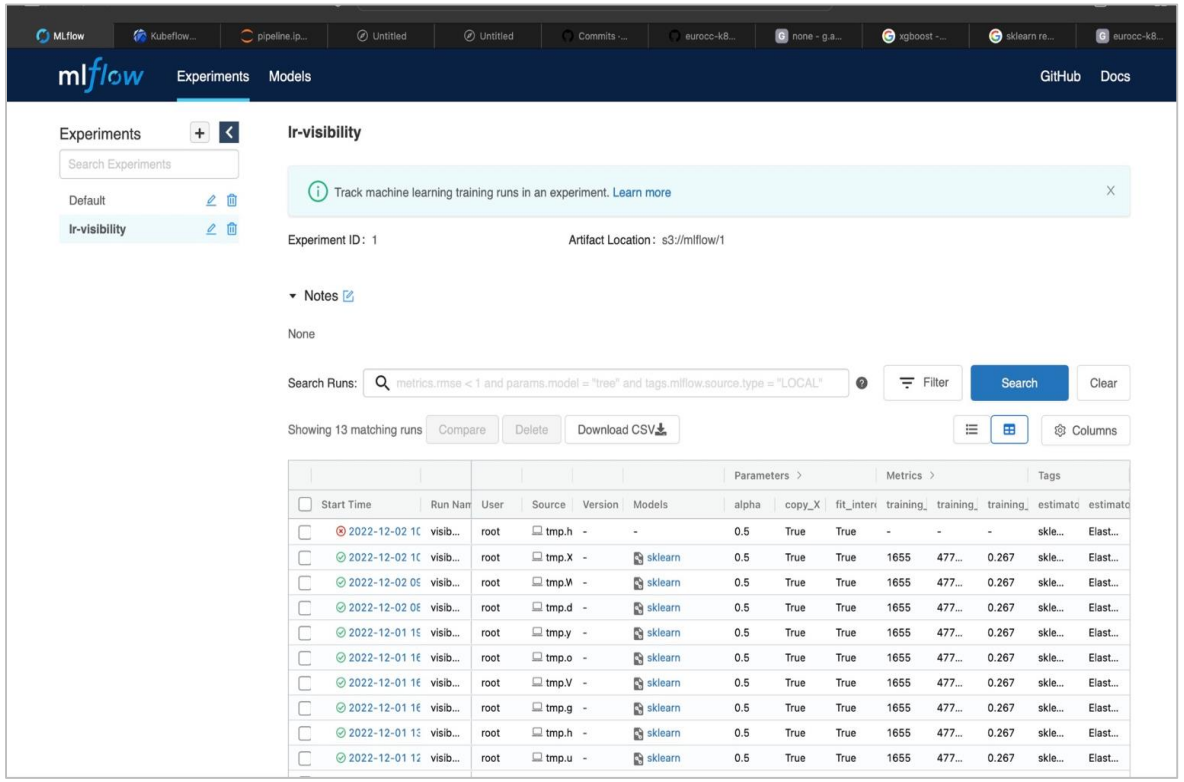
www.kubeflow.org

Results: ML pipeline

Kubeflow UI



MLflow UI



Results: Inference

- The trained model is used to **make predictions** on new data.
- **Online/synchronous predictions** are generated and returned as soon as requested via a REST API.

[30]:
```
# After run successful, predictions work
import numpy as np
import pandas as pd
import requests

host = "http://visibility-api:5000"
data = pd.DataFrame(np.random.uniform(280, 300, (10, 2)))
data = {"data": data.to_json()}
requests.post(host, json=data).json()
```

[30]:
```
{'model_run_id': '756143cb19584877a04760b25f012c14',
 'result': [25228.73570141659,
  16566.433902538003,
  15104.658390621364,
  19934.552981421584,
  25910.4226917302,
  30307.444178520236,
  15664.797000053222,
  31298.610895326594,
  16032.084838253446,
  34509.24563855934]}
```

Conclusions for PoC1

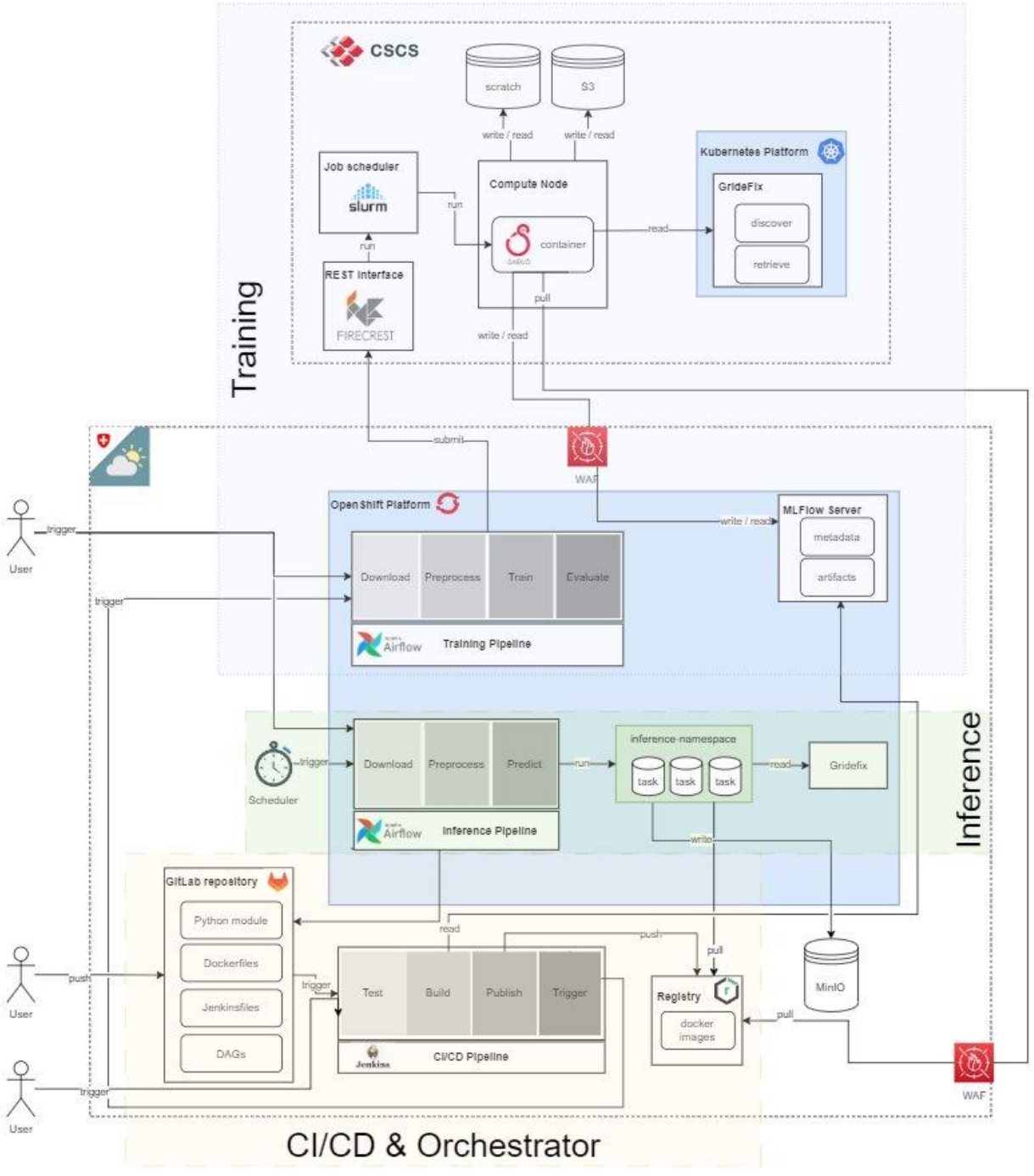
- We managed to build an end-to-end ML pipeline using our internal data infrastructure.
- We only tasted the power of Kubeflow as an MLOps framework, still a lot to explore and learn!
- However, Kubeflow is a complex ecosystem of services, which seems somewhat challenging to maintain.

Next: Test simpler setup in a production-ready environment.

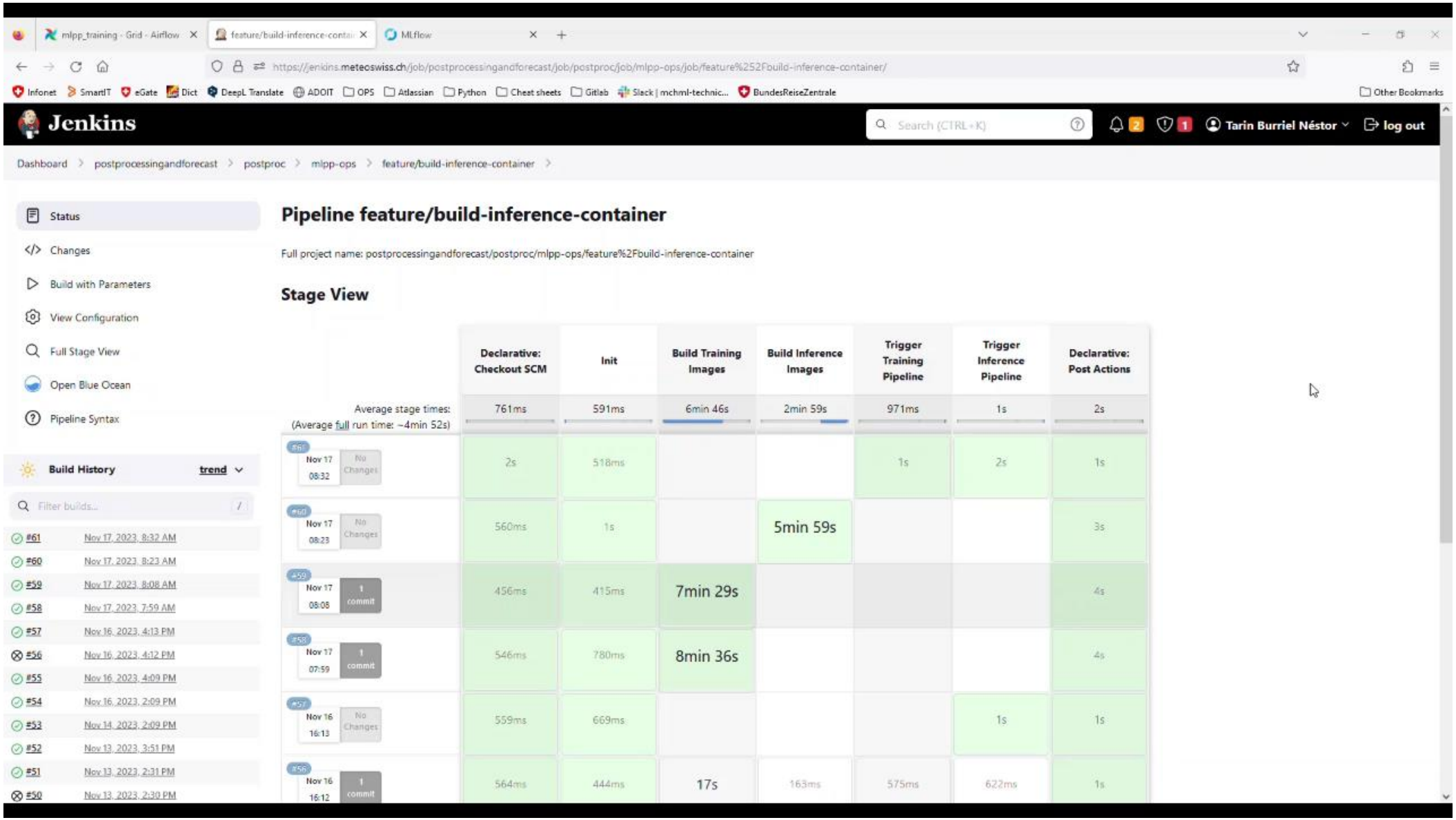
PoC2: On-prem ML pipeline

- Goal: Build an end-to-end ML pipeline using the set of components that are already available in our production environment:
  - GitLab for code versioning
  - Jenkins for CI/CD
  - Airflow for orchestration
  - Use HPC resources for training (access to GPUs)
  - Use Openshift for inference (Openshift)
- Use case: Postprocessing of wind

Setup


CSCS
scratch
S3
write / read
write / read
Job scheduler
slurm
run
Compute Node
container
read
Kubernetes Platform
GridFix
discover
retrieve
run
REST interface
FIRECREST
pull
write / read
Training
submit
WAF
Open Shift Platform
write / read
MLFlow Server
metadata
artifacts
User
trigger
Download
Preprocess
Train
Evaluate
Airflow
Training Pipeline
trigger
Scheduler
trigger
Download
Preprocess
Predict
run
inference-namespace
task
task
task
read
Gridefix
Airflow
Inference Pipeline
write
Inference
GitLab repository
Python module
Dockerfiles
Jenkinsfiles
DAGs
read
push
pull
User
push
trigger
Test
Build
Publish
Trigger
Registry
docker images
MinIO
pull
Jenkins
CI/CD Pipeline
User
trigger
WAF
CI/CD & Orchestrator

Demo PoC2

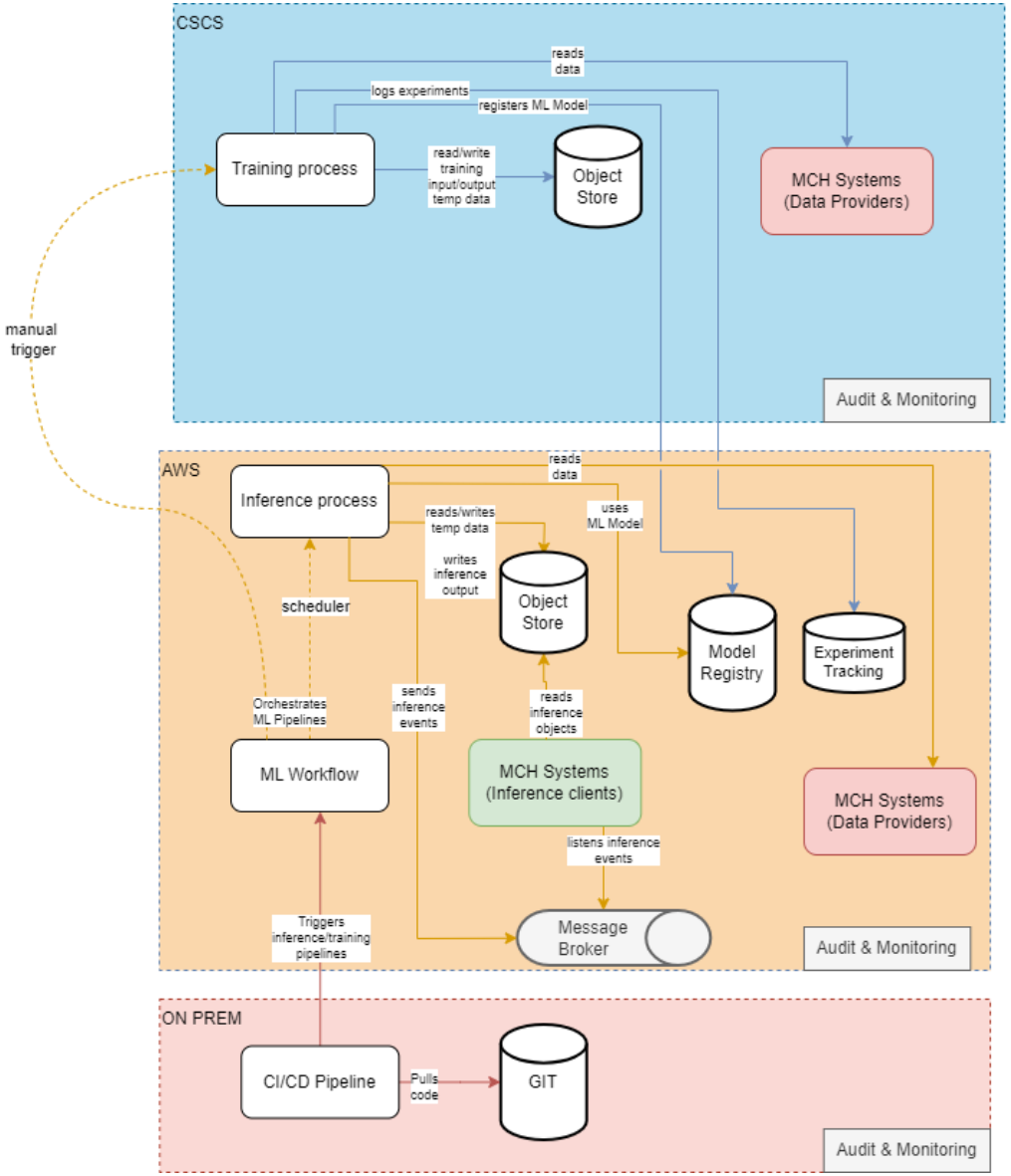

mlpp_training - Grid - Airflow
feature/build-inference-container
MLflow
https://jenkins.meteoswiss.ch/job/postprocessingandforecast/job/postproc/job/mlpp-ops/job/feature%252Fbuild-inference-container/
Jenkins
Search (CTRL+K)
Tarin Burriel Néstor
log out
Dashboard > postprocessingandforecast > postproc > mlpp-ops > feature/build-inference-container
Status
Changes
Build with Parameters
View Configuration
Full Stage View
Open Blue Ocean
Pipeline Syntax
Build History
trend
Pipeline feature/build-inference-container
Full project name: postprocessingandforecast/postproc/mlpp-ops/feature%2Fbuild-inference-container
Stage View
Declarative: Checkout SCM
Init
Build Training Images
Build Inference Images
Trigger Training Pipeline
Trigger Inference Pipeline
Declarative: Post Actions
Average stage times:
(Average full run time: ~4min 52s)
761ms
591ms
6min 46s
2min 59s
971ms
1s
2s
5min 59s
7min 29s
8min 36s
17s

Conclusions for PoC2

- Training at CSCS (HPC).
- Inference on prem (Openshift).
- On-prem job orchestration (Airflow).
- On-prem CICD (Jenkins).
- Track ML experiments and metadata (MLflow).
- Use model registry to bridge training and inference pipelines (MLflow).

Next: Consolidate PoCs into target architecture.

ML Platform: Target architecture



- "Multi-Cloud" architecture
- HPC (CSCS) provides compute and storage infrastructure for heavy workload:
  - **Training data**
  - **Training process**
- Production-critical components are hosted on Public Cloud (AWS):
  - **Orchestration**
  - **Model registry**
  - **Inference process**
- Code version control and CI/CD remain on prem.

Implementation plan

- MeteoSwiss is now evaluating the integration of cloud environments, the ML Platform can profit from this journey.
  - In collaboration with other partners and providers, MeteoSwiss will evaluate the best approach for a multi-cloud ML Platform setup.

- The ML Platform will grow together with different use cases, allowing it to have specific targets for the short-term mission, without forgetting the long-term vision.

- MLOps best practices and standards will be enforced with templates, base services (e.g. MLflow), and quality gates during operationalization.

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

MeteoSchweiz
Operation Center 1
CH-8058 Zürich-Flughafen
T +41 58 460 91 11
www.meteoschweiz.ch

MeteoSvizzera
Via ai Monti 146
CH-6605 Locarno-Monti
T +41 58 460 92 22
www.meteosvizzera.ch

MétéoSuisse
7bis, av. de la Paix
CH-1211 Genève 2
T +41 58 460 98 88
www.meteosuisse.ch

MétéoSuisse
Chemin de l'Aérologie
CH-1530 Payerne
T +41 58 460 94 44
www.meteosuisse.ch

Further Information on E-AI

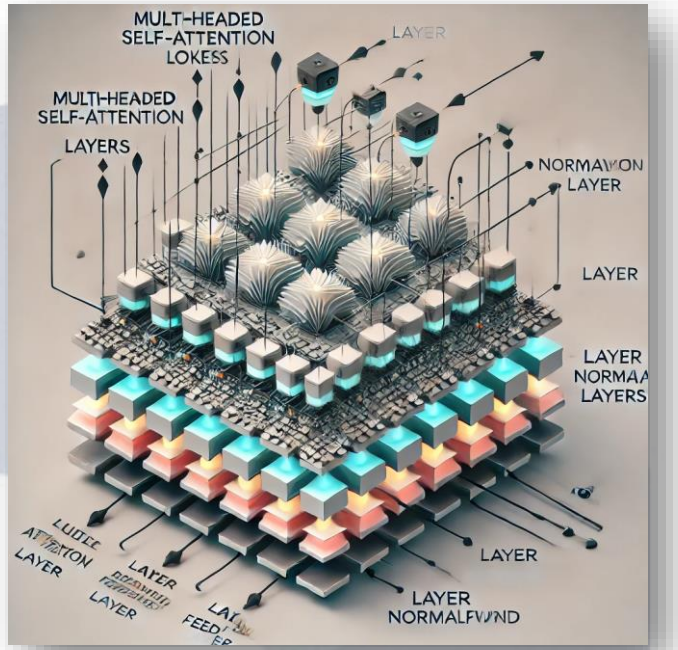
- Slides available at GitHub
  https://github.com/eumetnet-e-ai/tutorials
- Recording will be available at EUMETNET SharePoint
  https://tlnt19059.sharepoint.com/:f:/r/sites/E-AI/Shared%20Documents/Tutorials
- Register for E-AI updates and SharePoint Access:
  marek.jacob@eumetnet.eu
- Contacts:
  - Roland Potthast: Roland.Potthast@dwd.de
  - Marek Jacob: Marek.Jacob@dwd.de
  - Daniele Nerini: Daniele.Nerini@meteoswiss.ch
- E-AI Working Group „WG3 Operations“ for exchange on MLOps
  https://chat.europeanweather.cloud

E-AI Basic Tutorials

Tutorial E-AI Basics 4: January Wednesday 22, 2025, 11-12 CET
"MLOps" - Machine Learning Operations
4.1 Overview (20') [RP]
4.2 MLOps in relation to traditional Weather forecasting (20') [MJ]
4.3 Road to MLOps (20’) [DN]

Tutorial E-AI Basics 5: **February Wednesday 19**, 2025, 11-12 CET
MLflow - an open-source platform for managing the machine learning lifecycle
5.1 Overview - User perspective (20') [TG]
5.2 Logging to MLflow as a ML software developer (20') [HT]
5.3 Running MLflow server as a user and as a service (20') [MJ]

Tutorial E-AI Basics 6: February Wednesday 26, 2025, 11-12 CET
CI/CD - Continuous Integration and Continuous Deployment of ML codes
6.1 Overview – What can CI/CD do for you? (20') [MJ]
6.2 Basic tests with Pytest (20') [JD]
6.3 Setting up a runner (20') [FP]



AI generated Images,
© Roland Potthast 2024