WG 3 - MLOps best practices

Gabriela Aznar Siguan, Frank Guibert, Marek Jakob

E. Briola, L. Trani, N. Tarin Burriel, I. Schicker, H. Theissen, B. Butler, J. Castro, M. Cattaneo, H. de Laroussilhe, H. Kivril, B. Hargitai, O. Hinojo, K. Krus, S. Montazeri, D. Nerini, Y. Radev, M. Stoicescu, F. Teichert, H. Vandenbroucke-Menu, M. van Ginderachter, R. Derks, M. Zacharuk, S. Olah, ...
and all WG 3 participants

E-AI summer workshop 10.07.2025

What is MLOps?

- MLOps combines machine learning, software engineering, and DevOps principles to streamline and operationalize ML systems.
- The concept first emerged in 2015 in the paper "*Hidden Technical Debt in Machine Learning Systems, D. Scully et al".*

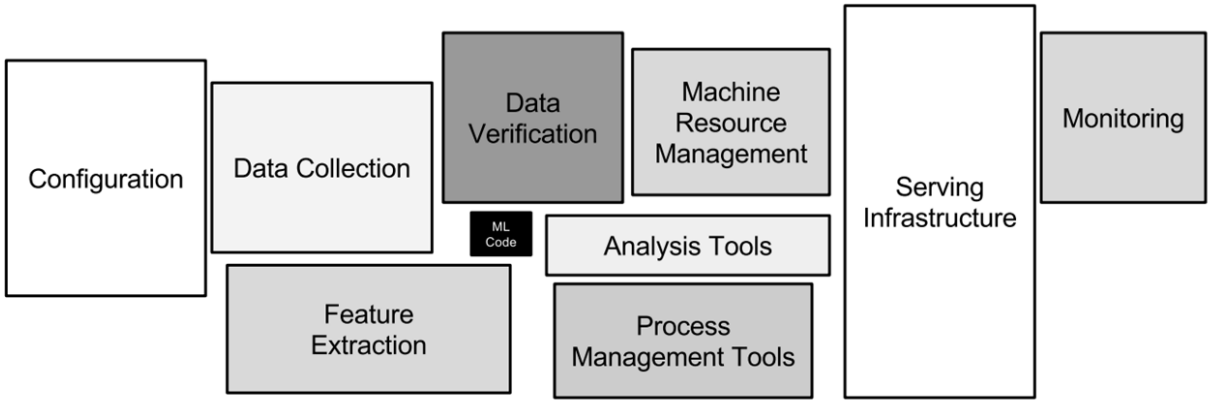
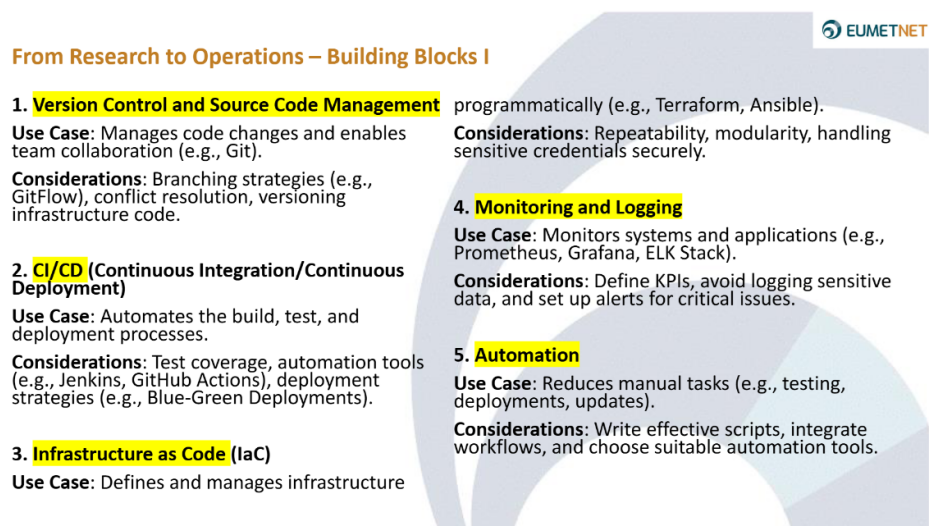
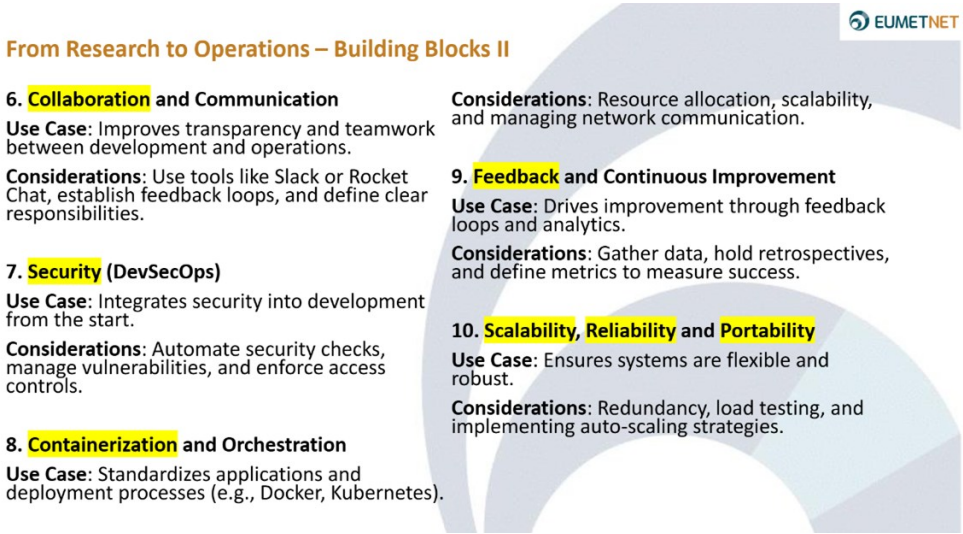


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

What do you need to implement MLOps?

MLOps Building blocks

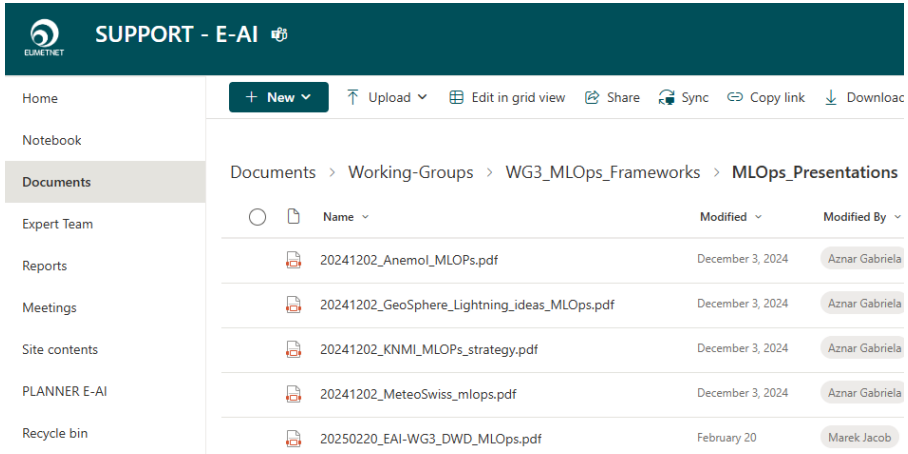


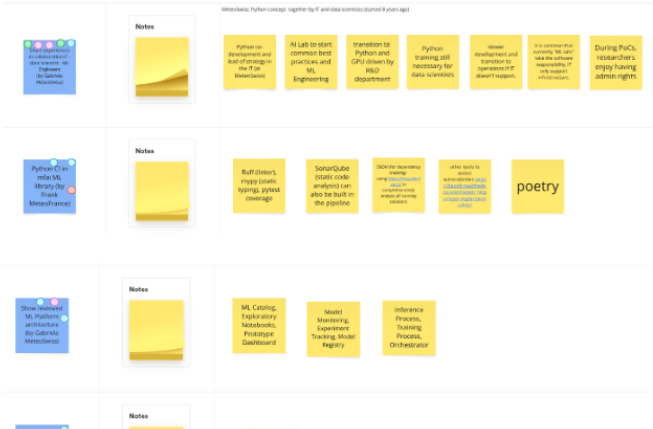


[E-AI Tutorial on MLOps]

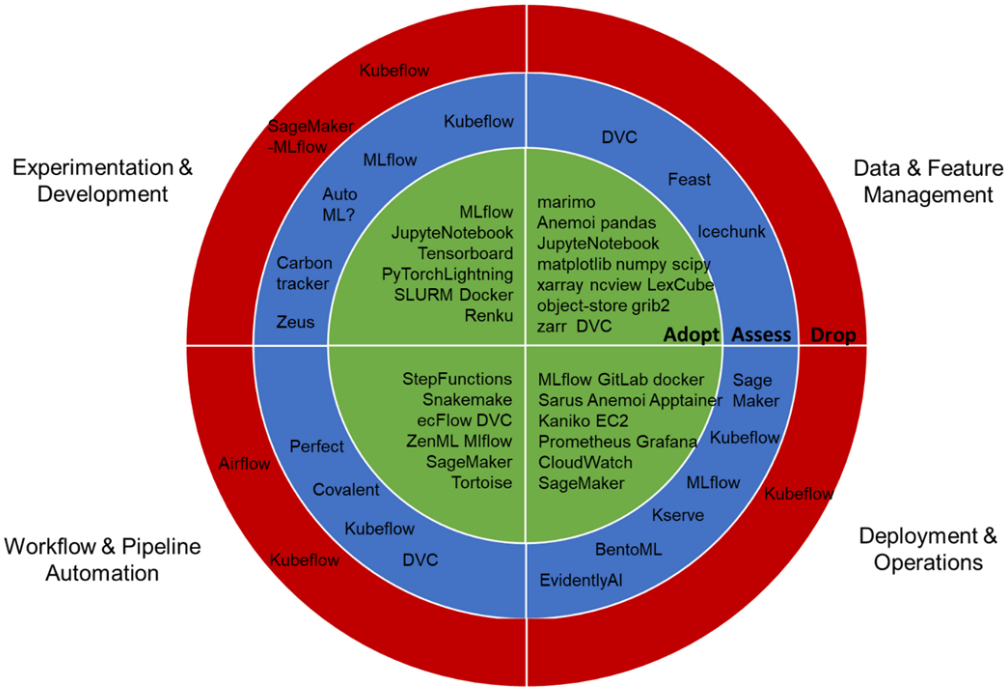
What have we been doing in WG3 wrt MLOps?

- **Presentations** of MLOps experiences and strategies of various organisations.
- Open **discussions** following the Lean Coffee approach.
- MLflow **workshop** with ECMWF ML Pilot.
- Consolidation: living **documents** MLflow best practices (currently in E-AI SharePoint).





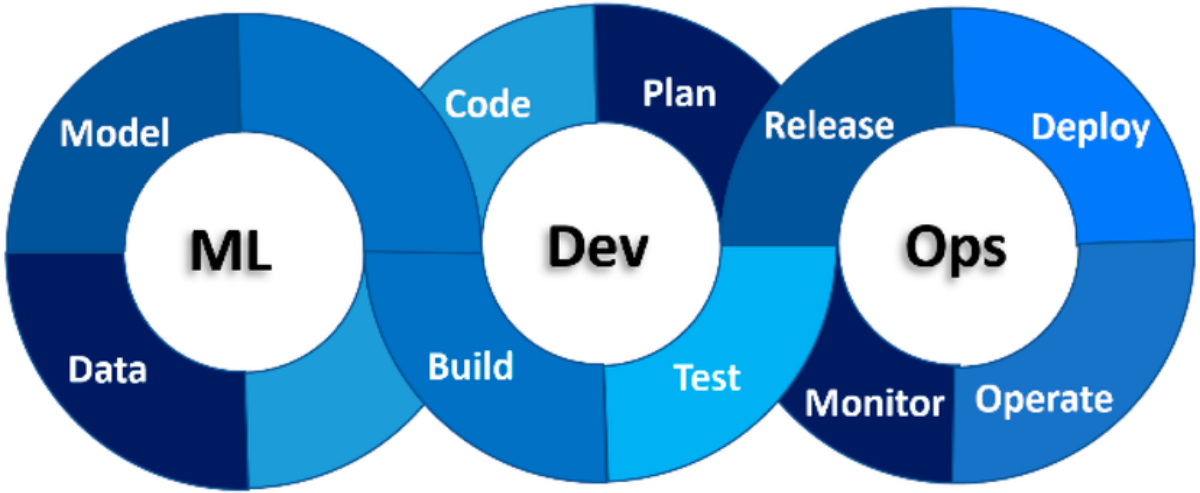
State of Technology Trials and Adoption



Conclusion:

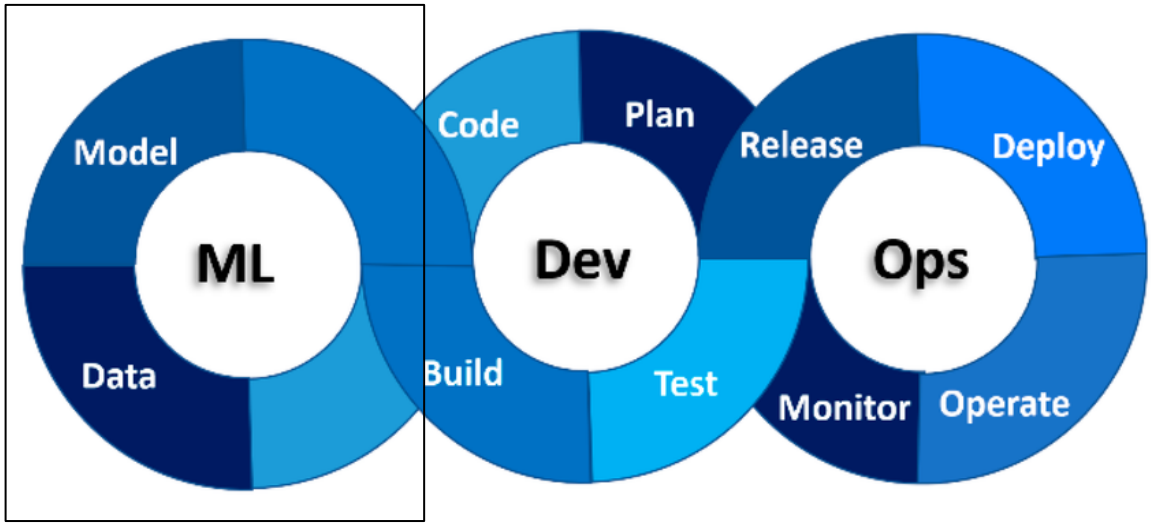
- Exchange, we have a lot of experience with a wide range of technologies.
- There's no magic formula – what works for some doesn't work for others (e.g. Kubeflow, SageMaker).
- Some solutions are starting to be used by the whole community (e.g. MLflow).

How the Technologies Are Being Applied



Subramanya, et al. , 2022

How the Technologies Are Being Applied
- ML -



ML - Data
Ensure reproducibility and lineage of datasets

Transform:
- Apply using reproducible, version-controlled code
- Preserve and propagate **metadata** (up to inference)

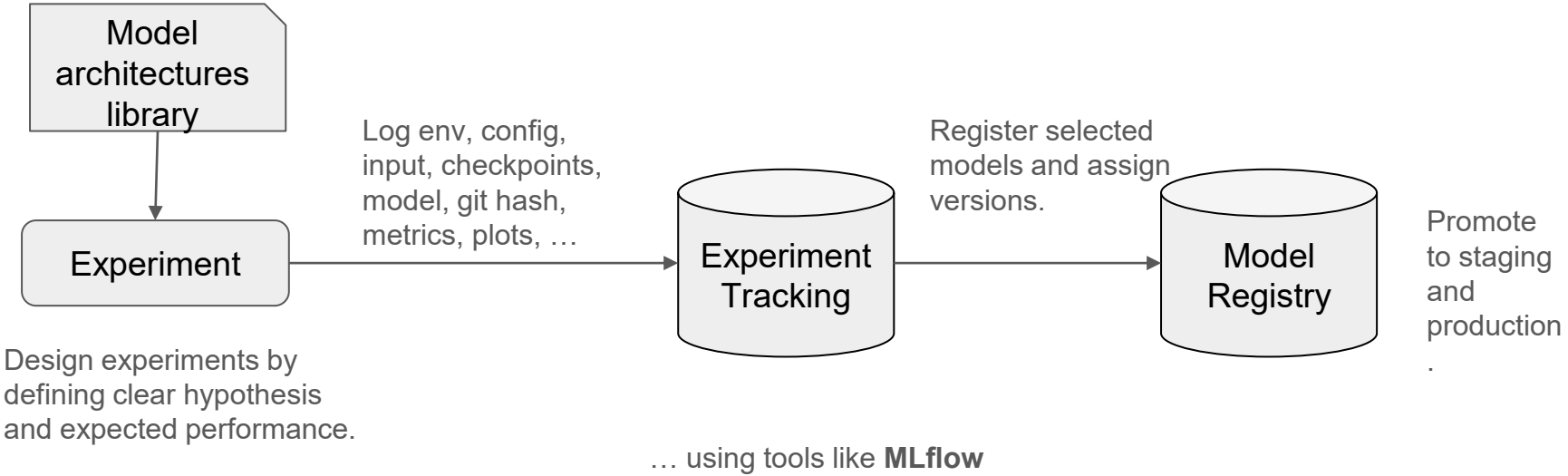
Preserve original data
in its **native format.**

- **Zarr** format for large datasets. Optimize dataset for specific ML task needs (chunking, variables).

- Versioning using tools like **Data Version Control (DVC)**.

Resources in E-AI: E-AI Data Inventory (WG 1), Anemoi Datasets Catalogue

ML - Model
Train, evaluate and register ML models



Resources in E-AI: Neural Network Architectures in mfai and Anemoi, Anemoi's Experiment tracking and Model catalogue.

Data Version Control and MLflow

- DVC lets you capture the **versions** of your data in Git commits, while storing them on-premises or in cloud storage.

```
dvc add data/train.csv
git add data/train.csv.dv
git commit -m "Track trai
dvc remote add -d myremote
dvc push
```

- MLflow provides a lightweight framework to track **experiments** by logging parameters, metrics, artifacts, and models.

```
# Example: sklearn
```

- DVC introduces a buil
execute and track dat
of ML models through

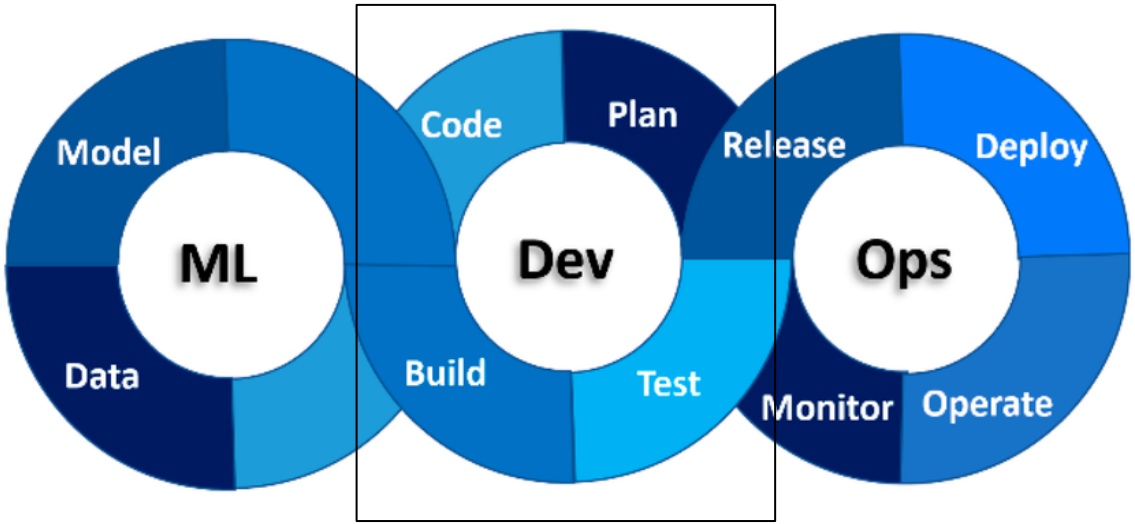
```
dvc stage add -n train_mod
  -d data/train.csv -d src
  -o models/model.pkl \
  python src/train.py
git add dvc.yaml dvc.lock s
git commit -m "Add ML pipeline and MLflow tracking"
dvc repro  # run the pipeline
```

```
client = MlflowClient()
client.create_registered_model(model_name)  # Optional: only first time
client.create_model_version(name=model_name, source=model_uri, run_id=run_id)
```

DVC + MLflow = ♥

Component	Tool	Purpose
Data	DVC	Versioned storage + remote sync
Experiments	MLflow	Param/metric logging, comparisons
Models	MLflow	Model tracking
Pipelines	DVC	Reproducible steps and dependencies

How the Technologies Are Being Applied
- DEV -

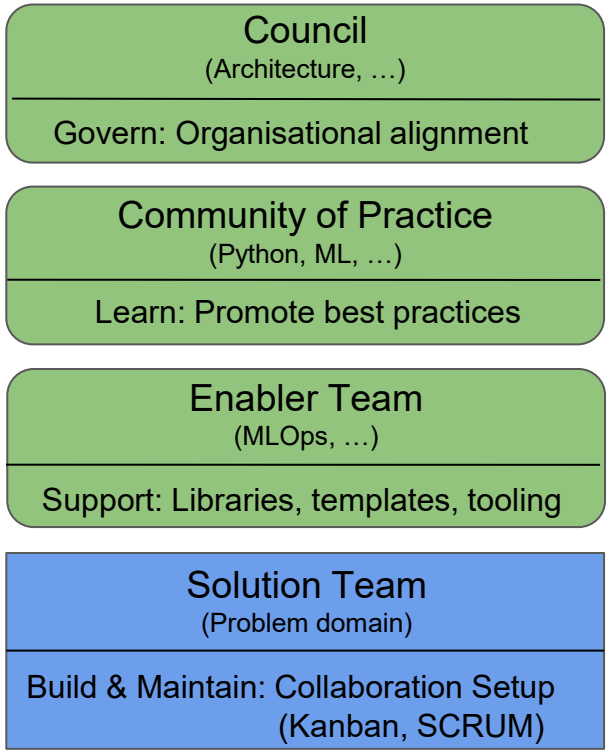


DEV - Plan

Define objectives, data strategy, evaluation protocols and collaboration frameworks

Key considerations:

- Objectives
- Requirements
- Evaluation
- Data Strategy
- Traceability
- Risks
- Lifecycle
- Development
- Organization



DEV - Code

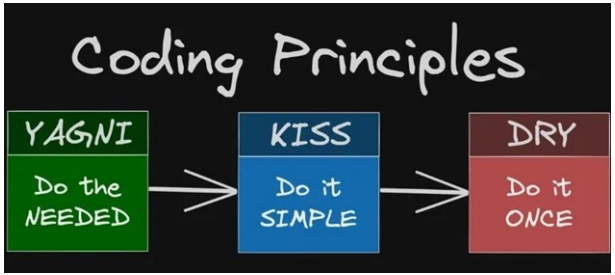
Develop modular, testable and version-controlled ML logic and pipelines

Key considerations:

- Principles
- Standards
- Configuration
- Secrets
- Version
- Documentation
- Visualization
- Collaboration
- Pipelines

- The Zen of Python
- TDD (Test-Driven Development)

- YAGNI (You Aren't Gonna Need It)
- DRY (Don't Repeat Yourself)
- KISS (Keep It Simple, Stupid)

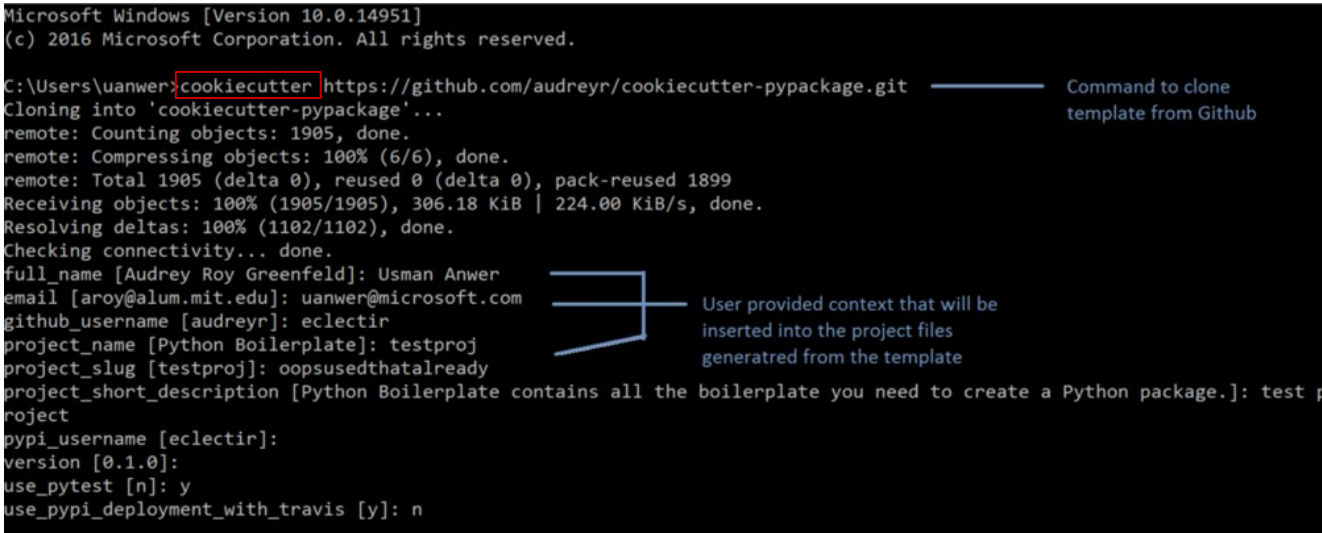


DEV - Code
Develop modular, testable and version-controlled ML logic and pipelines

Key considerations:

- Principles
- Standards
- Configuration
- Secrets
- Version
- Documentation
- Visualization
- Collaboration
- Pipelines

- Adopt and enhance organizational ML blueprints, libraries and coding conventions.
- Use **templating** tools to automate and enforce shared standards across teams.



Open-Source Python Application Template by MeteoSwiss

https://github.com/MeteoSwiss/Open-Source-Template

☑ CI/CD with GitHub Actions

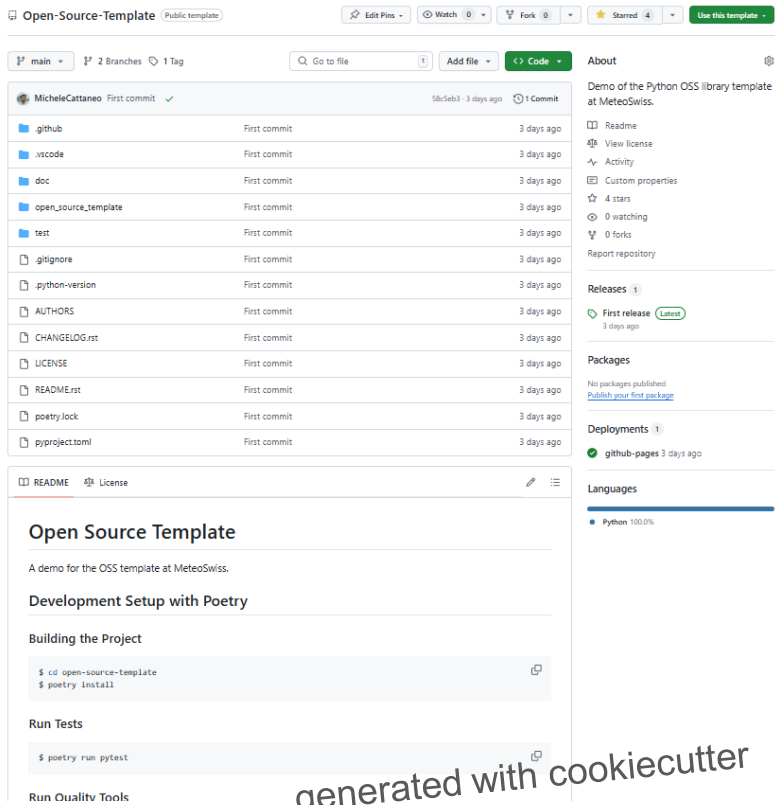
- **CI**: Runs tests on every push and pull request to the main
- **CD**: On version-tagged commits, automatically:
  - Publishes to **TestPyPI**, then **PyPI**
  - Builds and deploys **documentation** to GitHub Pages

Pre-configured Markdown Templates

- Issue templates for bug reports & feature requests
- Pull request templates with guided checklists

Versioning Consistency Check

- Release tag must match the version in pyproject.toml



generated with cookiecutter

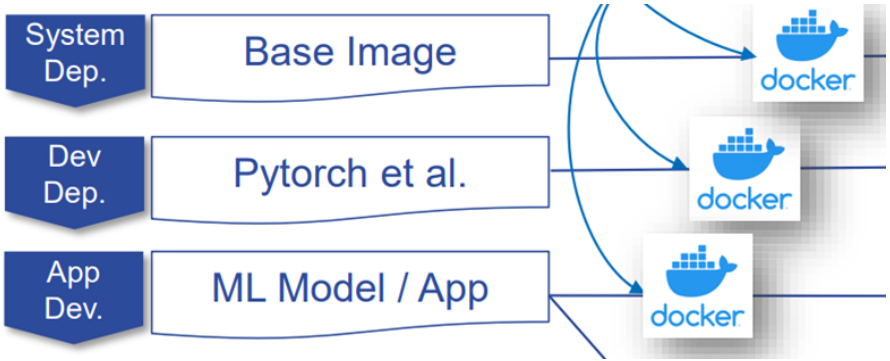
M. Cattaneo, H. de Laroussilhe, L. Knirsch

DEV - Build
Prepare the system for execution and deployment

Key considerations:

- Dependencies
- Containers
- Layering
- Container Registry
- Linting
- Security
- CI/CD
- Caching
- Model Integration

Layered container images with **shared base**



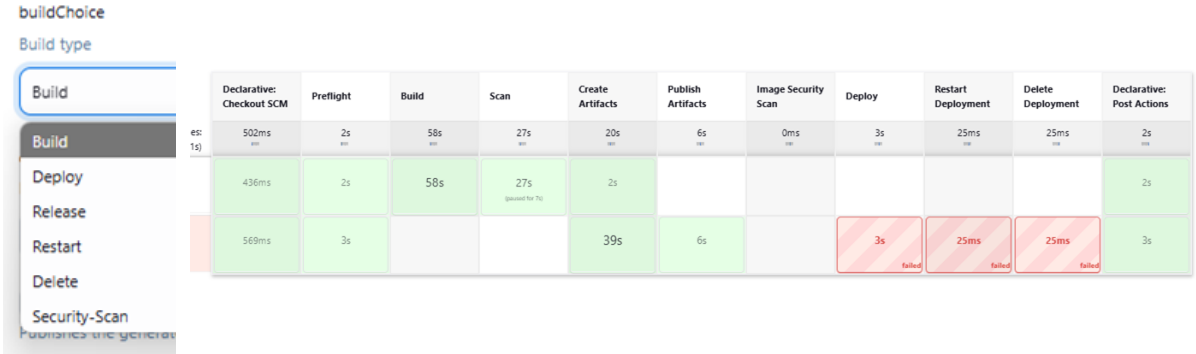
Speeds up builds via caching, improves consistency across projects and simplifies updates and security patching.

DEV - Build
Prepare the system for execution and deployment

Key considerations:

- Dependencies
- Containers
- Layering
- Container Registry
- Linting
- Security
- CI/CD
- Caching
- Model Integration

- **Automate** builds, tests, linting, static analysis, security scans, and deployments using CI/CD tools (GitHub Actions, GitLab CI).
- Contribute to portable, **reusable** scripts that encapsulate the CI/CD workflows and are decoupled from specific projects.



DEV - Test

Ensure your ML code, data and models behave as expected

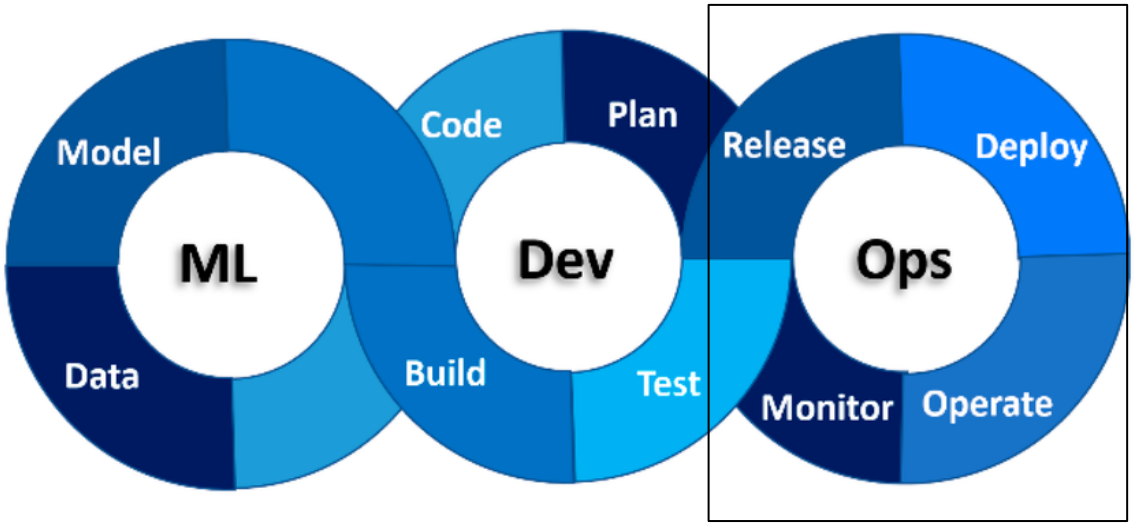
Key considerations:

- Unit & Integration Tests
- Data Validation
- Configuration Reproducibility
- Model Evaluation
- CI Automation

- **Benchmark** models against performance thresholds and baselines across regimes, geographies or forecast lead times.
- Ensure model artifacts **serialize** and reload consistently for deployment.

- *Open Neural Network Exchange (ONNX) provides a standard way to represent models.*
- *Significantly reduce the image size and startup time, which is especially valuable in cloud, serverless, or edge deployments.*

How the Technologies Are Being Applied
- OPS -

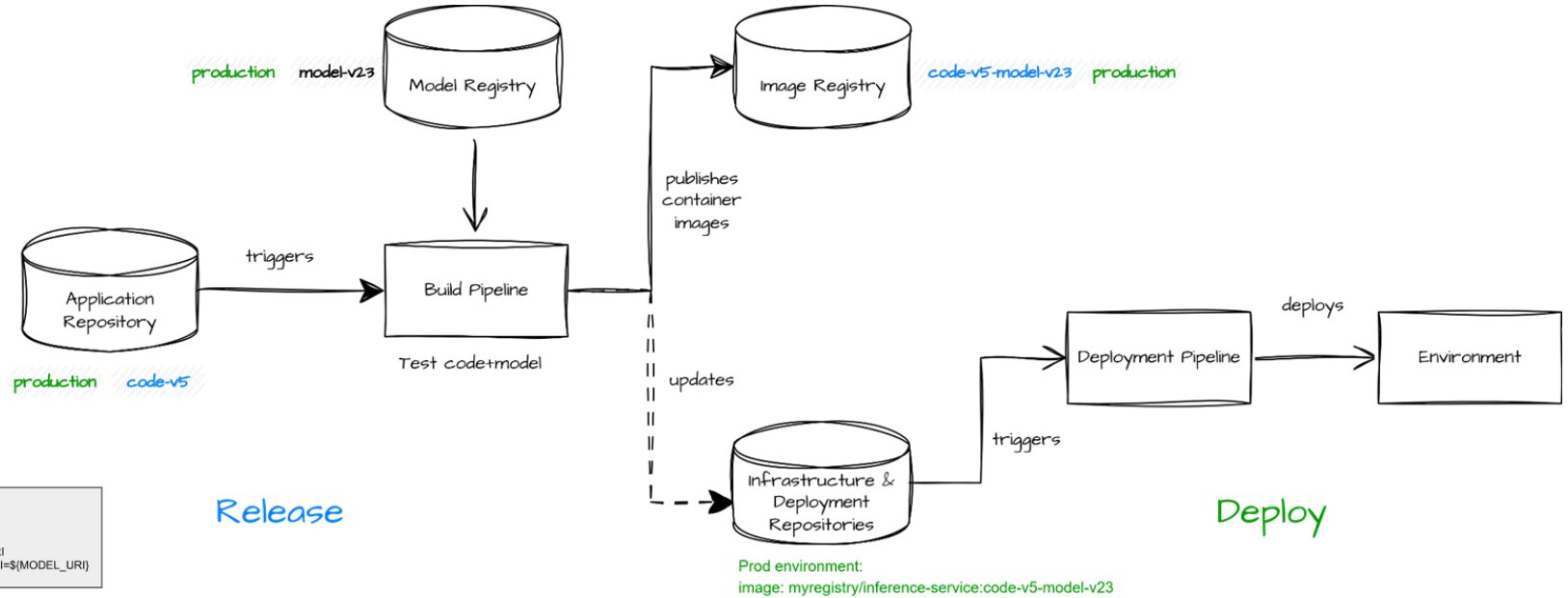


OPS - Release & Deploy

Release: Promote tested models and pipelines to production-readiness

Deploy: Serve models reliably across deployment stages

GitOps: Git as the single source of truth automating deployments



OPS - Operate & Monitor

Operate: Run and manage the production system
Monitor: Track model performance and behavior over time

Monitoring is critical in ML to detect issues like data drift, prediction skew, and performance degradation — but finding the right tool is challenging:

- Need for customizable **visualizations** — especially for geospatial data.
- Must handle high **data** volumes with low latency.
- Preference for low-**maintenance** or easily operable solutions.
- Assessing the trade-offs and benefits of a **centralized** solution across teams or domains.
- The **feedback** loop — i.e., the delay between model predictions and receiving ground truth — complicates real-time performance monitoring and model retraining decisions

Proposed goals for WG-3

- **Interoperability:** agree on interface contracts between Python components and HTTP APIs. Objective: being able to exchange neural network architectures, loss functions, metrics transparently between different projects of different met offices.

- Agree on **development standards** (type checking, linting, complexity, …) and supply **templates** and/or **recipes** to disseminate and enforce them.

- Produce a first version **best practices document** on operating ML-based products (e.g. forecasts, post processing) and services (e.g. chatbots)