



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. Žacharuk, S. Olah,
...
and all WG 3 participants

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*”.

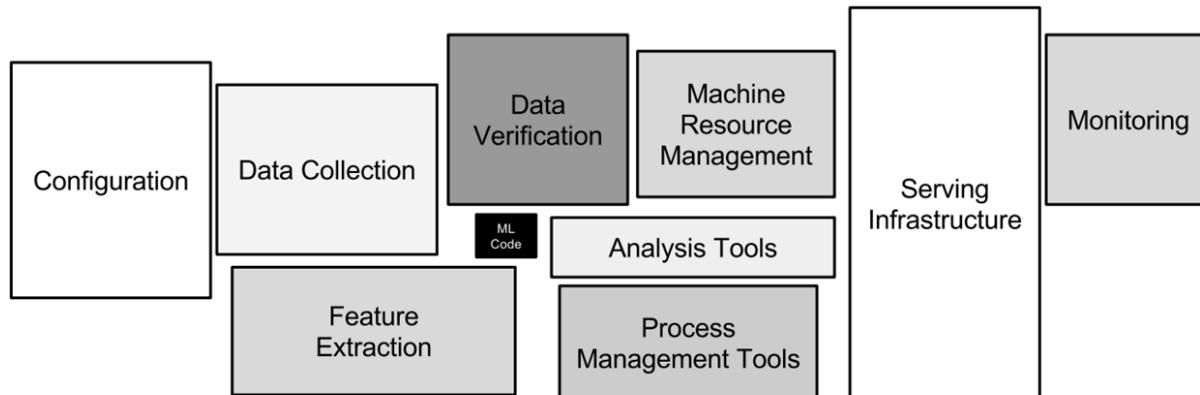


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

What do you need to implement MLOps?



MLOps Building blocks



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.

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).

 **SUPPORT - E-AI** 

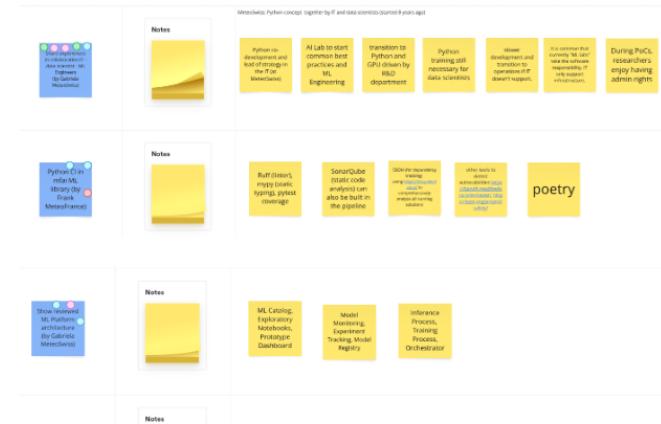
Home + New Upload Edit in grid view Share Sync Copy link Download

Notebook

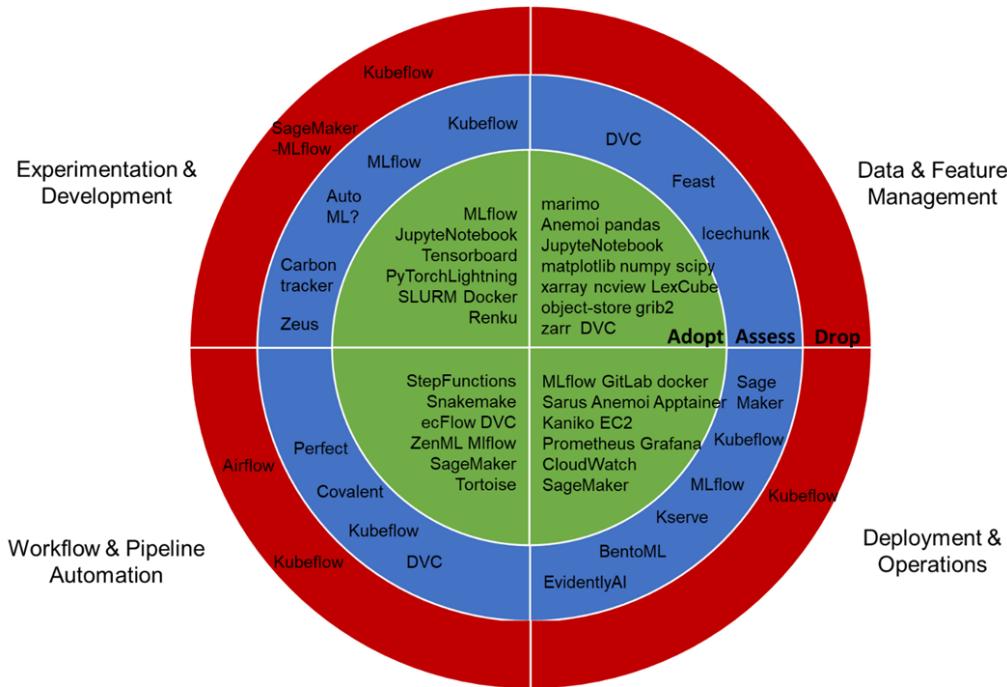
Documents

Documents > Working-Groups > WG3_MLOps_Frameworks > MLOps_Presentations

	Name	Modified	Modified By
	20241202_Anemol_MLOPs.pdf	December 3, 2024	Aznar Gabriela
	20241202_GeoSphere_Lightning_ideas_MLOps.pdf	December 3, 2024	Aznar Gabriela
	20241202_KNMI_MLOPs_strategy.pdf	December 3, 2024	Aznar Gabriela
	20241202_MeteoSwiss_mllops.pdf	December 3, 2024	Aznar Gabriela
	20250220_EAI-WG3_DWD_MLOps.pdf	February 20	Marek Jacob



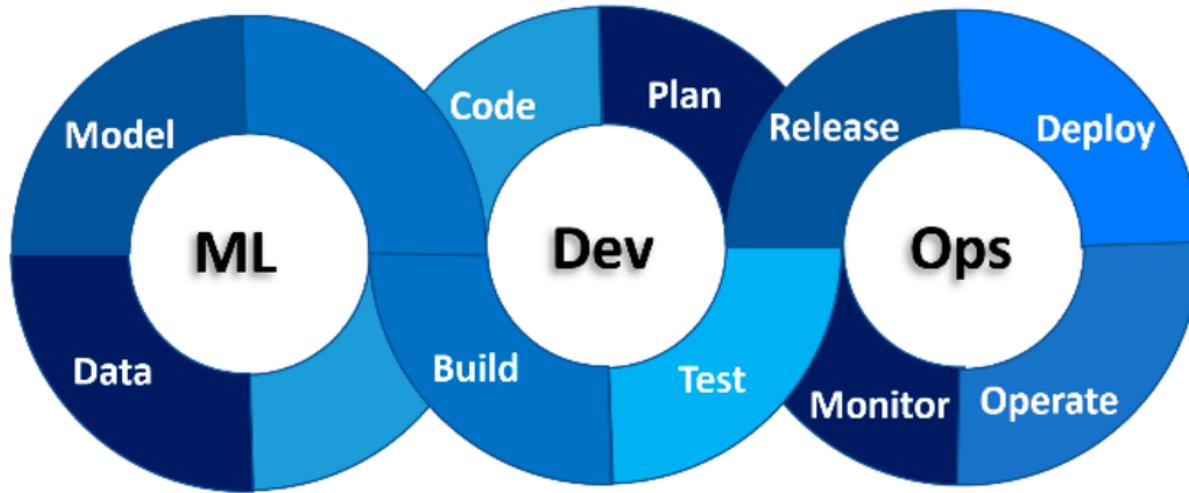
State of Technology Trials and Adoption



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).

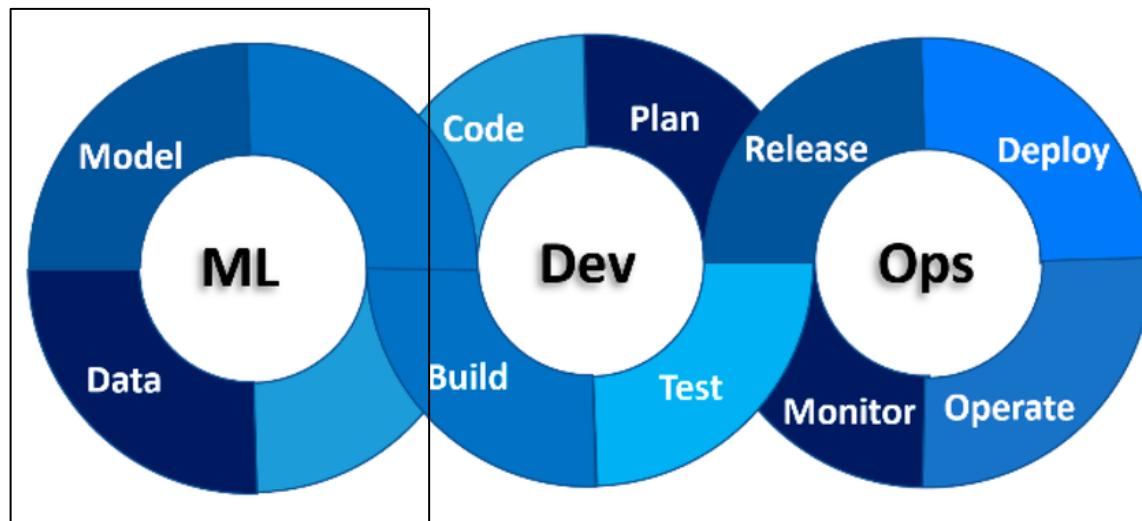
How the Technologies Are Being Applied



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)



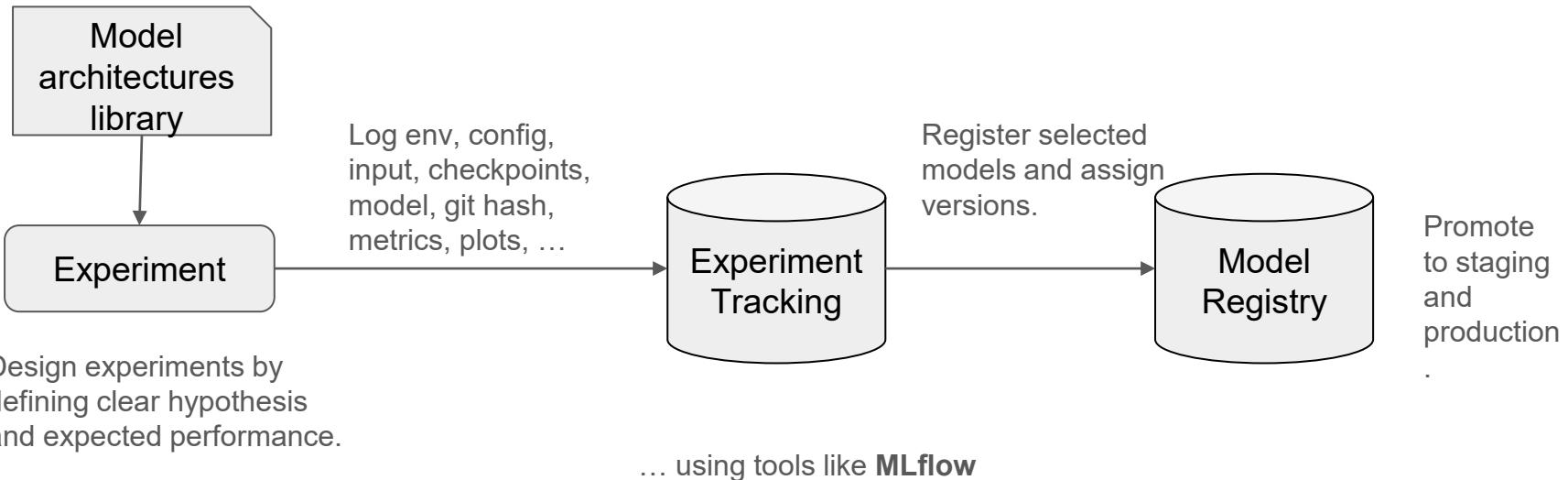
Preserve original data
in its **native format**.

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

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

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

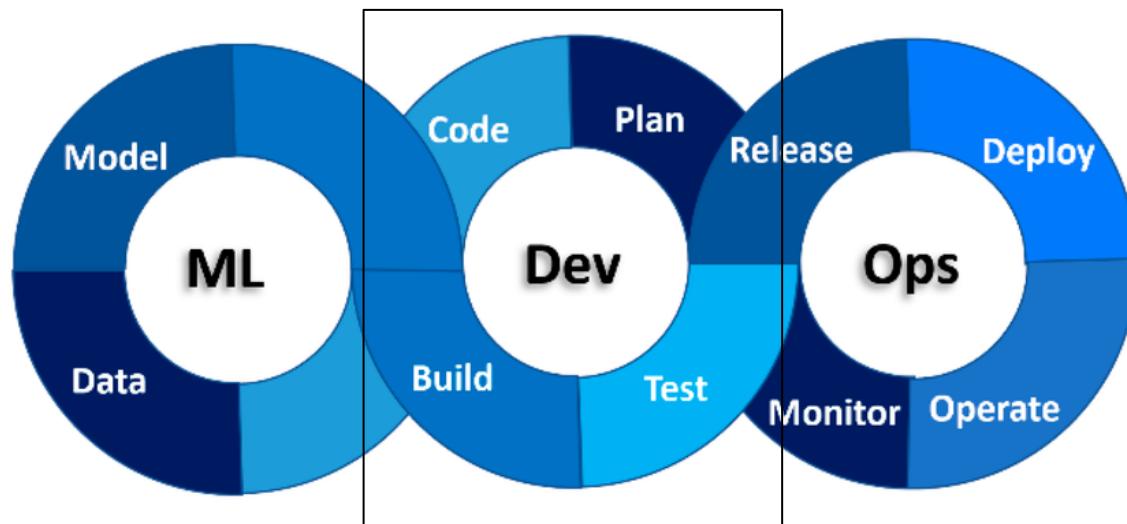
- DVC introduces a build step to execute and track data pipelines.

```
dvc stage add -n train_model
  -d data/train.csv -d src
  -o models/model.pkl \
  python src/train.py
git add dvc.yaml dvc.lock
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)
```

How the Technologies Are Being Applied

- DEV -

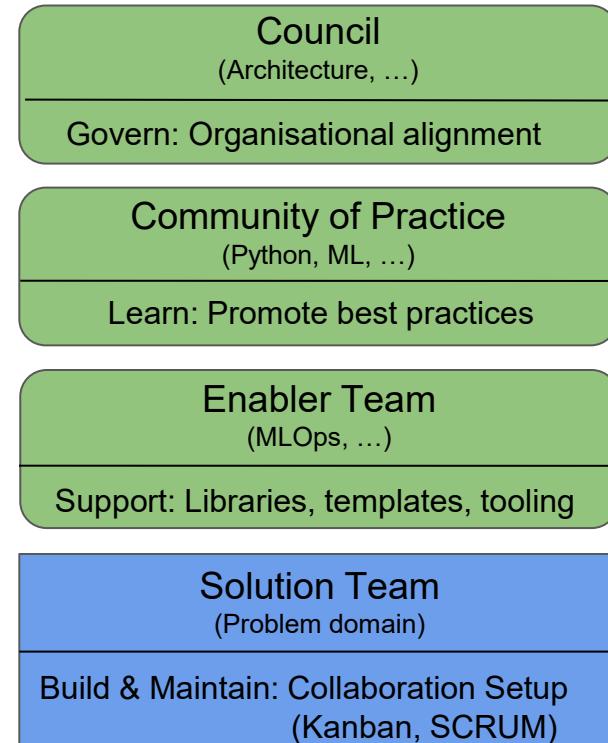


DEV - Plan

Define objectives, data strategy, evaluation protocols and collaboration frameworks

Key considerations:

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



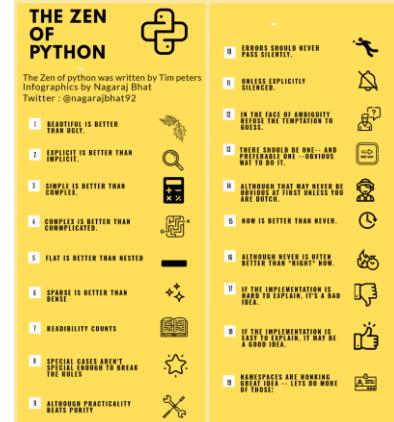
DEV - Code

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

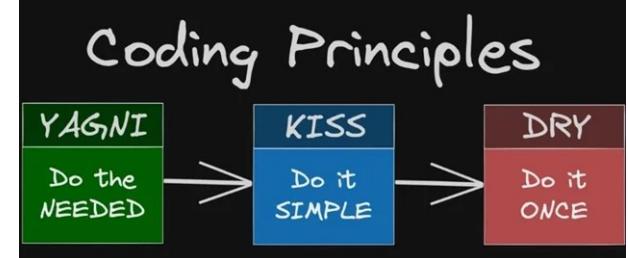
Key considerations:

- 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.

```
Microsoft Windows [Version 10.0.14951]
(c) 2016 Microsoft Corporation. All rights reserved.
```

```
C:\Users\uanwer>cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
Cloning into 'cookiecutter-pypackage'...
remote: Counting objects: 1905, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 1905 (delta 0), reused 0 (delta 0), pack-reused 1899
Receiving objects: 100% (1905/1905), 306.18 KiB | 224.00 KiB/s, done.
Resolving deltas: 100% (1102/1102), done.
Checking connectivity... done.
full_name [Audrey Roy Greenfeld]: Usman Anwer
email [aroy@alum.mit.edu]: uanwer@microsoft.com
github_username [audreyr]: eclectir
project_name [Python Boilerplate]: testproj
project_slug [testproj]: oopsusedthatalready
project_short_description [Python Boilerplate contains all the boilerplate you need to create a Python package.]: test project
pypi_username [eclectir]:
version [0.1.0]:
use_pytest [n]: y
use_pypi_deployment_with_travis [y]: n
```

Command to clone template from Github



User provided context that will be inserted into the project files generated from the template

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

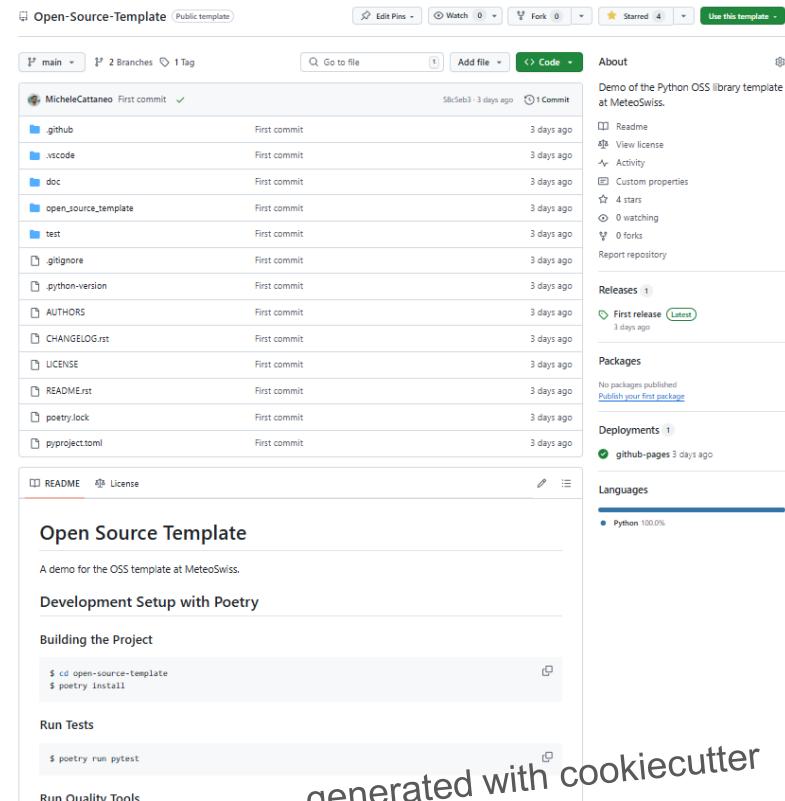
Pre-configured Markdown Templates

- 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**

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



The screenshot shows the GitHub repository page for "Open-Source-Template". The repository has 2 branches and 1 tag. The commit history shows 11 commits from MicheleCattaneo, all made 3 days ago. The repository is public and has 4 stars. It includes sections for Readme, View license, Activity, Custom properties, Report repository, Releases (with a latest release 3 days ago), Packages (no packages published), Deployments (with a deployment 3 days ago), and Languages (Python 100.0%). Configuration sections shown include README, License, Scripts (with examples for building the project and running tests), Development Setup with Poetry, Building the Project, Run Tests, and Run Quality Tools.

generated with cookiecutter

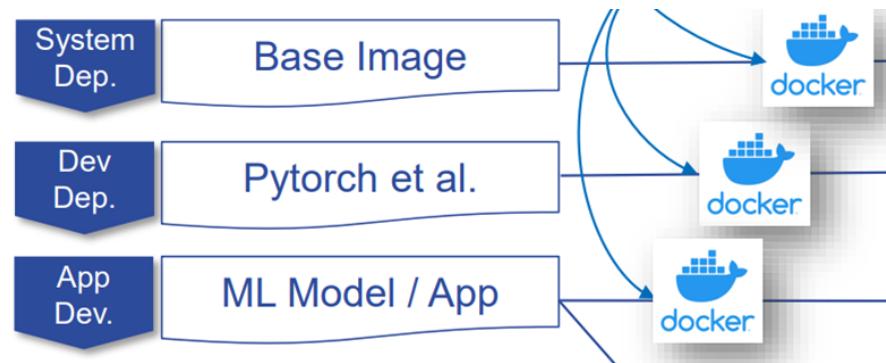
DEV - Build

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.

buildChoice	Build type	Build	Declarative: Checkout SCM	Preflight	Build	Scan	Create Artifacts	Publish Artifacts	Image Security Scan	Deploy	Restart Deployment	Delete Deployment	Declarative: Post Actions
	Build	es: 1s	502ms	2s	58s	27s	20s	6s	0ms	3s	25ms	25ms	2s
	Deploy		436ms	2s	58s	27s	2s						2s
	Release												
	Restart		569ms	3s			39s	6s			3s failed	25ms failed	25ms failed
	Delete												3s
	Security-Scan												

Annotations: The table shows various build steps and their execution times. The 'Build' row has a blue header, indicating it's the active choice. The 'Deploy' row is highlighted in green. The 'Restart' row is highlighted in red. The 'Delete' row is also highlighted in red. The 'Security-Scan' row is greyed out. The 'Create Artifacts' and 'Publish Artifacts' columns are also greyed out.

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

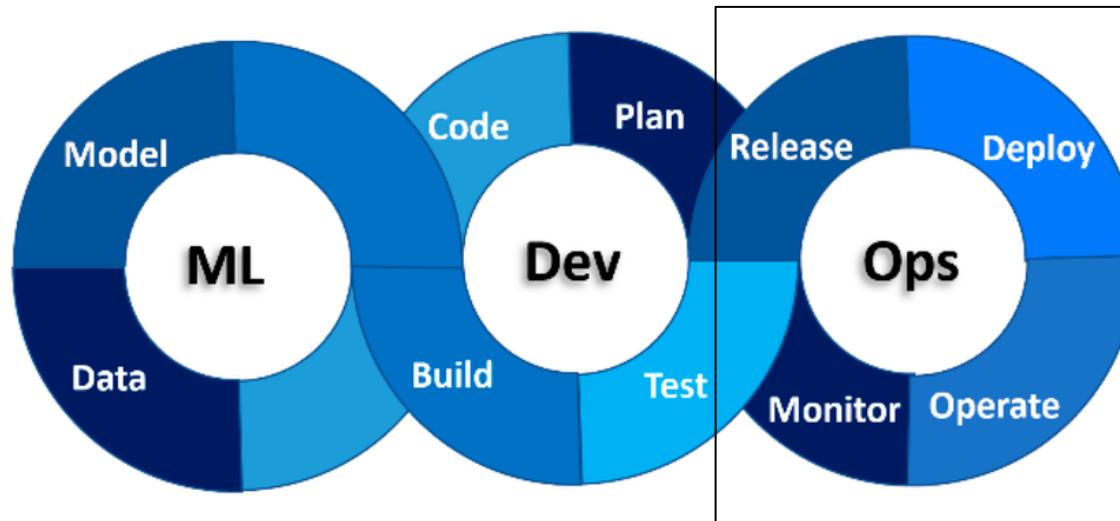
- Benchmark models against performance thresholds and baselines across regimes, geographies or forecast lead times.
- Ensure model artifacts **serialize** and reload consistently for deployment.



- 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 -

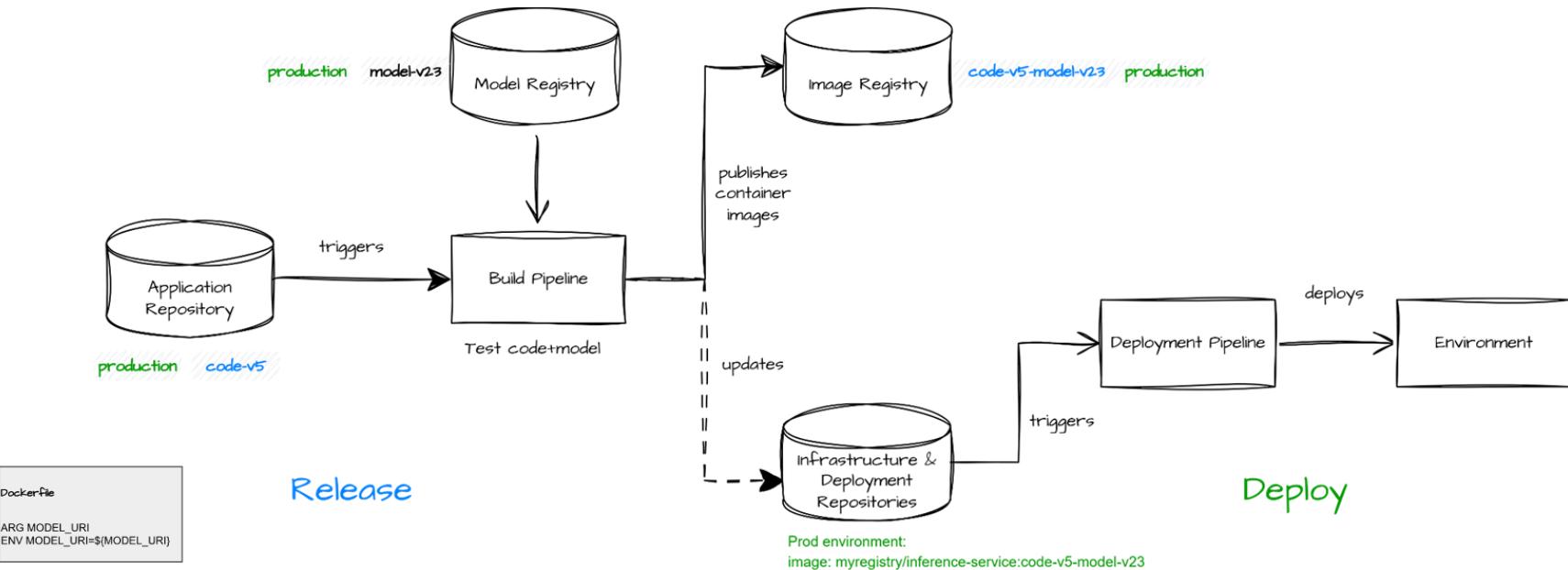


OPS - Release & Deploy

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)