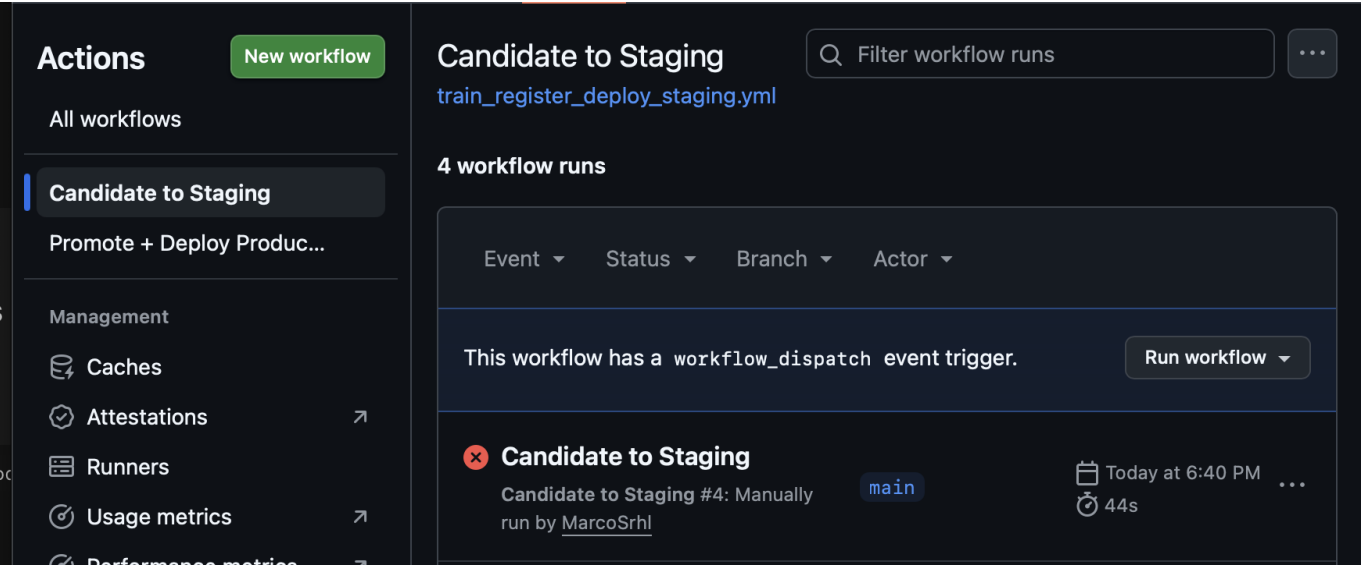


Exercise: Model Promotion Pipeline (Staging -> Production)

Github link: <https://github.com/MarcoSrhI/MLopsExercise> Marco-Naji Serhal

Task 1: Run Candidate -> Staging workflow



Trigger Candidate to Staging workflow (workflow_dispatch)



After Running the workflow we can see that it failed.

train_register
failed 1 hour ago in 40s


Search logs




>  Set up job 0s

>  Run actions/checkout@v4 1s

>  Set up Python 0s

>  Install ML deps 27s

  Train + register in MLflow (capture JSON only) 9s

1 ▶ Run set -e

27 Registered model 'churn-model' already exists. Creating a new version of this model...

28 2026/01/28 17:41:23 INFO
mlflow.store.model_registry.abstract_store: Waiting up to 300 seconds for model version to finish creation. Model name: churn-model, version 6

29 Created version '6' of model 'churn-model'.

30 {"run_id": "2f5656f854a845bfa7f5178a2d3cd936", "accuracy": 0.8225, "model_version": "6"}

  Evaluate gate 0s

1 ▶ Run printf '%s' '{"run_id": "2f5656f854a845bfa7f5178a2d3cd936", "accuracy": 0.8225, "model_version": "6"}' | python ml/evaluate.py

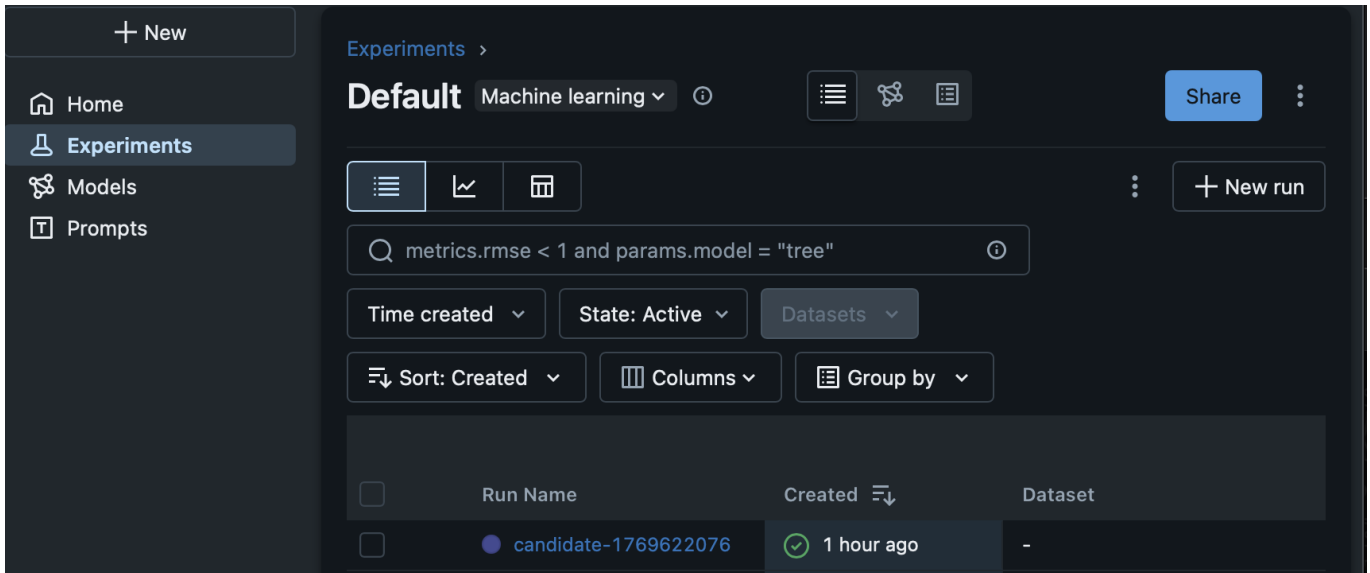
12 FAIL: accuracy=0.8225 < 0.9

13 **Error:** Process completed with exit code 1.

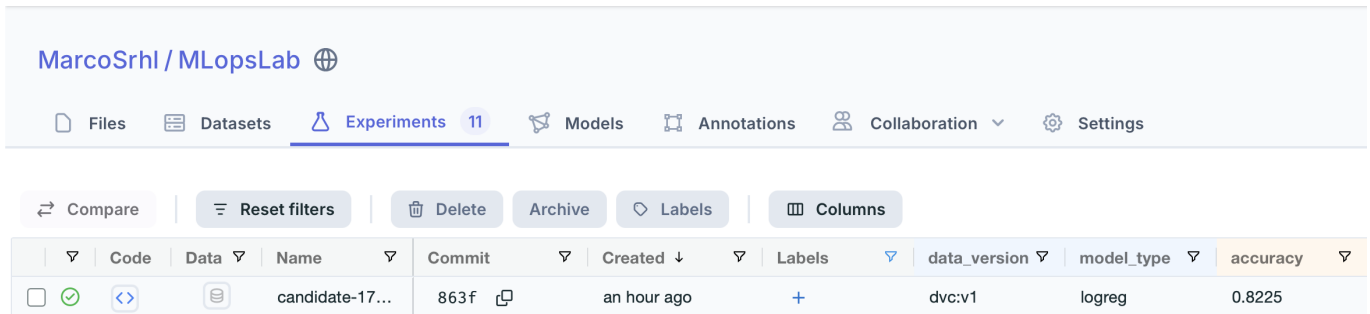
As we can see, the pipeline correctly trained the Logistic Regression model, computed the accuracy, logged the metrics in MLFlow and also registered a new model version.

The gate checks allow an accuracy $\geq .90$, but in the observed results we can see 0.8225 so the gate failed and the deployment was aborted.

2 / 11



Although, we still can see that the model has been registered in MLFlow.



the MLFlow experiment was created. We can see from all the above:

- Model version: 6
- Accuracy: 0.8225
- Gate did not pass (failed)

Task 2: Explain what "staging" proves

What staging tests that offline evaluation does not

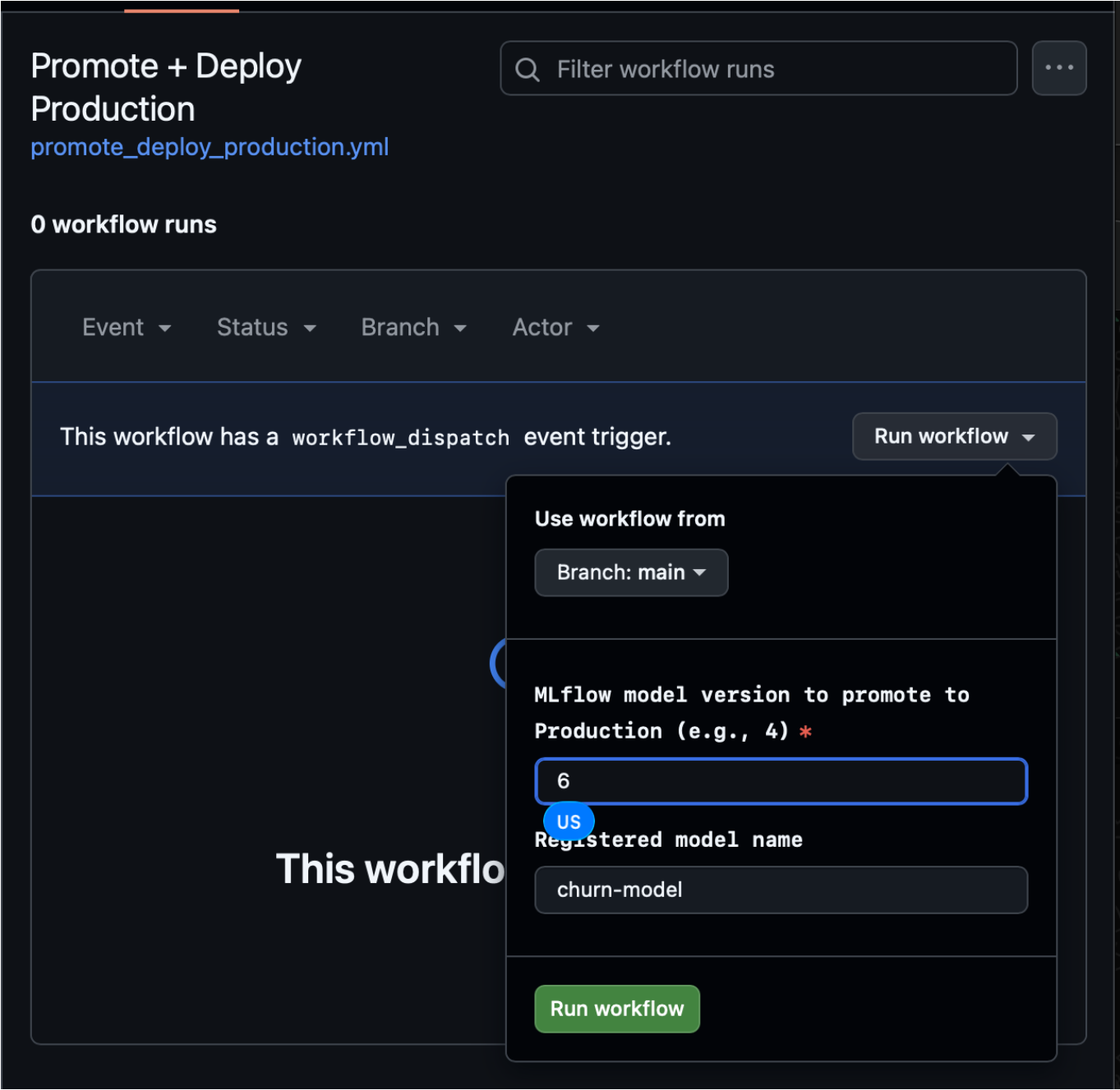
Offline evaluation only shows metrics like accuracy on a dataset. Staging additionally tests things like:

- The model can be loaded from the registry by stage (models:/churn-model/Staging) with the real credentials and artifacts
- The serving code/container starts and runs with the correct dependencies (no missing packages, version mismatches)
- The API server starts correctly
- The API contract works end-to-end (/health, /predict), including input format, preprocessing expectations, and output shape
- Runtime behavior is acceptable (latency, memory), and failures show up as real errors/logs
- Integration with surrounding pieces (Docker, env vars, secrets, network) is correct

In short, staging tests the real deployment, in real production conditions.

Task 3: Promote to production

- Trigger Promote to Production
- Provide model_version from Task 1
- Observe: MLflow stage transition to Production deployment job Deliverable: screenshot of the promotion log line.



we are using the model_version = 6

Promote + Deploy Production #4

Re-run all jobs

...

Summary

All jobs

promote_to_production

deploy_production

Run details

Usage

Workflow file

Manually triggered 4 minutes ago

Status

Total duration

Artifacts

MarcoSrh1 31f5232 main

Success

2m 0s

—

promote_deploy_production.yml

on: workflow_dispatch

promote_to_production

39s

deploy_production

1m 13s

—

+

Promote + Deploy Production

Promote + Deploy Production #4

Re-run all jobs

...

Summary

All jobs

promote_to_production

deploy_production

Run details

Usage

Workflow file

promote_to_production

succeeded 4 minutes ago in 39s

Search logs

> Set up job

1s

> Run actions/checkout@v4

0s

> Set up Python

1s

> Install MLflow client

33s

> Promote model version to Production

2s

1 ▶ Run python - << 'PY'

39 <stdin>:17: FutureWarning:

``mlflow.tracking.client.MlflowClient.transition_model_version_stage`

` is deprecated since 2.9.0. Model registry stages will be removed in

a future major release. To learn more about the deprecation of model

registry stages, see our migration guide here:

[https://mlflow.org/docs/latest/model-registry.html#migrating-from-](https://mlflow.org/docs/latest/model-registry.html#migrating-from-stages)

[stages](https://mlflow.org/docs/latest/model-registry.html#migrating-from-stages)

40 Promoted churn-model v6 -> Production (archived old Production)

> Post Set up Python

0s

> Post Run actions/checkout@v4

0s

> Complete job

0s

As we can see, the workflow updated the MLFlow registry, and this model is now the official production one.

5 / 11

Registered Models > churn-model >

Version 6

Registered At: 01/28/2026 06:41:23 PM

Modified: 01/31/2026 01:09:39 AM

Type: candidate 1769622076

Stage: Production

New model registry UI

> Description

Edit

> Tags

> Schema

Name

Type

[-] Inputs (0)

No schema. See [MLflow docs](#) for how to include input and output schema with your model.

[+] Outputs (0)

Below we can see the deployment job using the production model, confirming the deployment step was executed successfully.

< Promote + Deploy Production

✔ Promote + Deploy Production #4

Re-run all jobs

Summary

All jobs

✔ promote_to_production

✔ **deploy_production**

Run details

Usage

Workflow file

deploy_production

succeeded 8 minutes ago in 1m 13s

Search logs

> ✔ Set up job

0s

> ✔ Run actions/checkout@v4

1s

> ✔ Build + push backend image (production)

1m 9s

> ✔ Deploy to production (placeholder)

0s

> ✔ Post Run actions/checkout@v4

0s

> ✔ Complete job

0s

The docker image also has been created in DockerHub.

Repositories

All repositories within the quantymarco namespace.

Search by repository name

All content

Create a repository

Name	Last Pushed ↑	Contains	Visibility	Scout
quantymarco/churn-backend	8 minutes ago	IMAGE	Public	Inactive

Task 4: Prove production uses registry stage, not "latest code"

Locally: Run staging backend reading "Staging" and prod backend reading "Production" Verify /health returns correct stage

← Candidate to Staging

✓

Candidate to Staging #13

Re-run all jobs

...

Summary

All jobs

✓ train_register

✓ **deploy_staging**

✓ staging_smoke_test

Run details

Usage

Workflow file

deploy_staging

succeeded 1 minute ago in 1m 44s

Search logs

↺ ⚙

> ✓ Set up job1s

> ✓ Run actions/checkout@v41s

> ✓ Set up Python (for MLflow stage transition)0s

> ✓ Install ML deps (same as train job)26s

> ✓ **Promote model version to Staging2s**

1 ▶ Run set -e

48 <stdin>:24: FutureWarning:

``mlflow.tracking.client.MlflowClient.transition_model_version_stage`

` is deprecated since 2.9.0. Model registry stages will be removed in

a future major release. To learn more about the deprecation of model

registry stages, see our migration guide here:

<https://mlflow.org/docs/latest/model-registry.html#migrating-from-stages>

49 Promoting model=churn-model version=15 -> Staging

50 Existing versions: ['1', '2', '3', '4', '5', '6', '7', '8', '9',

'10', '11', '12', '13', '14', '15']

51 Promoted churn-model v15 -> Staging

> ✓ Build + push backend image (staging)1m 10s

> ✓ Deploy to staging (placeholder)0s

> ✓ Post Set up Python (for MLflow stage transition)1s

> ✓ Post Run actions/checkout@v40s

Above, we modified the accuracy threshold (from .90 to .80) to automatically allow one version to pass the gate and be set to staging. We could have also done it manually on MLFlow (I played a lot with it to understand the logic around it).

7 / 11

Registered Models >

churn-model

Created Time: 01/28/2026, 12:12:11 AM

Last Modified: 01/31/2026, 06:31:16 AM

> Description

Edit

> Tags

< Versions

All

Active 2

Compare

New model registry UI

Version

Registered at

Created by

Stage

Description

✓ Version ...

01/31/2026, 06:30:40 AM

Staging

Registered Models > churn-model >

Version 15

Registered At: 01/31/2026, 06:30:40 AM

Modified: 01/31/2026, 06:31:16 AM

Source Run: candidate 1769837434

Stage: Staging

New model registry UI

> Description

Edit

> Tags

< Schema

Name

Type

+ Inputs (0)

+ Outputs (0)

The screenshot shows the MLflow Model Registry interface for a model named 'churn-model'. The top section displays 'Version 6' with a three-dot menu icon. Below this, the registration details are shown: 'Registered At: 01/28/2026 06:41:23 PM', 'Modified: 01/31/2026 01:09:39 AM', 'Run: candidate 1769622076', and 'Stage: Production'. A toggle switch for 'New model registry UI' is on the right. The left sidebar has links for 'Description', 'Tags', and 'Schema'. The main content area shows a table with columns 'Name' and 'Type', containing sections for 'Inputs (0)' and 'Outputs (0)'.

We now have both versions of model in the registry and can then run both production and staging workflows and make sure that they are using the correct versions from above.

```
(MLopsExercise) marcoserhal@Marcos-MacBook-Pro MLopsExercise % curl http://localhost:8000/health
{"stage":"Staging","status":"ok"}
(MLopsExercise) marcoserhal@Marcos-MacBook-Pro MLopsExercise % curl http://localhost:8001/health
{"stage":"Production","status":"ok"}
```

Clarification on gate vs stage transitions In the first "Candidate -> Staging" run, the pipeline successfully registered a new MLflow model version (v6) but the quality gate failed (accuracy < threshold as we saw). A gate failure does not prevent the version from existing in the registry—it only prevents the automatic transition to the "Staging" stage and any downstream automated promotion steps. For Task 3, the exercise required demonstrating a 'manual override' promotion to Production, so we promoted a specific version directly to "Production" to show the MLflow stage transition and the production deployment workflow

Task 3: Promote to production

1. Trigger **Promote to Production**

2. Provide **model_version** from Task 1

Later, we lowered the accuracy threshold and re-ran "Candidate -> Staging". This produced a new model version (new run/artifact) that passed the gate and could be transitioned cleanly to "Staging". Even if the training code and data generation were unchanged between runs, a re-run still creates a new MLflow version because it logs a new run and artifact; the only difference here was the gate threshold.

1. Why is it dangerous to deploy "whatever just merged to main" as the model?

It is dangerous because the main reflects code changes, not a validated and reproducible model artifact. It means that we can accidentally ship an untested model (or a model trained on different data), lose traceability (which data/params produced it), and make rollbacks hard because the main would move constantly.

2. What does the registry stage give you that a Git tag does not?

A registry stage points to a specific model artifact or version with its metrics, lineage, and lifecycle state (Staging/Production), not just code. It enables controlled promotion/rollback of models without changing Git history, and multiple model versions can coexist as the Production pointer moves.

3. If staging passes but production fails, what could be the causes?

The causes could be environment or runtime differences (different env vars/secrets, network access, permissions, data), different infrastructure constraints (CPU/memory limits), dependency mismatch, missing model/data access in prod (artifact store auth), config differences (different stage/URI, ports), or production-only traffic/schema edge cases that aren't covered in the staging tests.

4. Where should DVC fit in a serious pipeline:

DVC should be used to version all key datasets to make sure that experiments are comparable and reproducible. Training data snapshot ensures that models can be retrained, the evaluation dataset snapshot guarantees consistent comparisons and the drift reference dataset is used for monitoring, to detect when production data changes over time.

5. What should be added to the gate beyond accuracy?

Accuracy is good to check but not enough to show that we have a good production model. We should add latency checks to make sure the model is fast enough in real-time use. The schema checks are here to prevent crashes from input format change. Fairness constraints is like its name suggests, reduce bias but also unethical behavior. Adversarial or robustness tests ensure that the model remains stable if we use noisy inputs or like its name says adversarial inputs.

Extensions (optional)

'use ngrok to add a public-facing endpoint that you can use to automatically run deployments locally'

We run only our backend locally with 'docker compose -f deploy/docker-compose.staging.yml up --build'. Then we expose it using 'ngrok http 8000'

```
ngrok (Ctrl+C to quit)
One gateway for every AI model. Available in early access *now*: h

Session Status      online
Account             Marco (Plan: Free)
Version             3.35.0
Region              Europe (eu)
Latency             23ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://carmen-tumid-cultivatedly.ngrok

Connections          ttl      opn      rt1      rt5      p50
                   0        0        0.00     0.00     0.00
```

We can see that /health returns same response for both locally and using the ngrok public URL, which confirms that the local deployment is reachable externally, enabling remote triggers/tests against the local environment.

```
(MlopsExercise) marcoserhal@Marcos-MacBook-Pro MlopsExercise % curl https://carm
en-tumid-cultivatedly.ngrok-free.dev/health
{"stage":"Staging","status":"ok"}
(MlopsExercise) marcoserhal@Marcos-MacBook-Pro MlopsExercise %
```