



CI, CD and CT for ML: What Changes

CI/CD for ML 01 | What differs from software CI/CD: data and model as versioned artefacts, the trigger map (push, tag, schedule, data event, drift alert), the life of a change, pipeline-as-code maturity, delivery metrics for model releases

#CICD #ContinuousTraining #MLOps #DORA

What changes compared with software

CI/CD/CT for ML

In one sentence

●○○

Software CI/CD ships code; ML must also ship data and models, which change on their own schedule and are judged by statistics, not by pass/fail asserts alone.

Practitioner

●●○

Each artefact gets an immutable id, its own trigger and its own gate; CI tests code with a smoke train, CT retrains on new data, CD deploys a pinned version.

- code → git SHA, image → digest
- data → snapshot hash
- model → registry version

Expert

●●●

The deployment is a tuple (image digest, model version, feature definitions, config); every tuple is reproducible, gated, signed and reversible by changing one pin.

- rollback = previous tuple
- skew tests compare training and serving features

Analogy

A restaurant chain: the recipe (code), the ingredients delivered each morning (data) and the dish served (model) are all inspected, not only the recipe.

CI/CD for ML at three depths

Before CI/CD/CT for ML, can you explain...

	Not yet	Partly	Yes
MUST KNOW Explain CI, CD and CT in one sentence each Learn: mlops 03	☐	☐	☐
MUST KNOW Version data with DVC or object versions Learn: mlops 02, workflows 10	☐	☐	☐
MUST KNOW Read a GitHub Actions workflow Learn: workflows 10	☐	☐	☐
HELPFUL Name the parts of a container image Learn: tools-and-platforms (Docker)	☐	☐	☐
HELPFUL Know what drift is Learn: mlops 05	☐	☐	☐

Before this sheet: can you already do these?

Software CI/CD	ML CI/CD/CT
Input that changes: code	Inputs that change: code, data, features, base model, prompts
Tests are deterministic asserts	Gates are thresholds on noisy metrics with a baseline and a tolerance
Build output: binary or image (MB)	Outputs: image plus weights (MB to hundreds of GB) and eval reports
Triggered by commits	Triggered by commits, schedules, new data and monitoring alerts
Same code → same behaviour	Same code on new data → a different model
Rollback = previous image	Rollback = previous image + model + feature definitions together
Runs on CPU runners	Some jobs need GPUs, large disks and data access

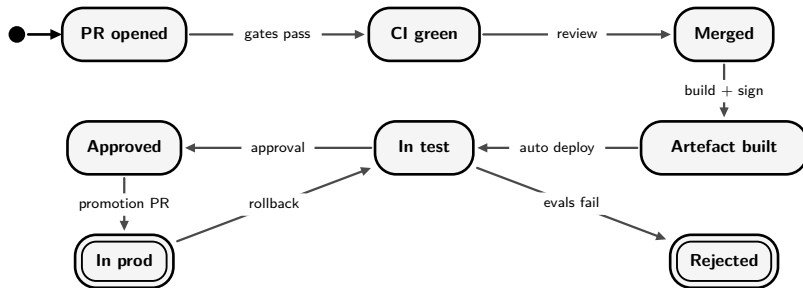
Data and model as artefacts

Artefact	Immutable id	Stored in	Produced by
Source and configs	git commit SHA	GitHub	developers
Container image	digest sha256:...	Amazon ECR (immutable tags on)	CI
Dataset snapshot	DVC md5 / S3 version id / Iceberg snapshot	S3	data pipeline
Eval set	content hash, frozen per release	S3 + git pointer	evaluation owners
Model version	registry name + version	MLflow or SageMaker Model Registry	CT / training job
Prompt or LLM config	prompt version or git SHA	git or prompt registry	app team
Infrastructure	Terraform state serial + plan file	S3 state bucket	IaC pipeline
Deployment manifest	config-repo commit SHA	GitHub config repo	promotion PRs

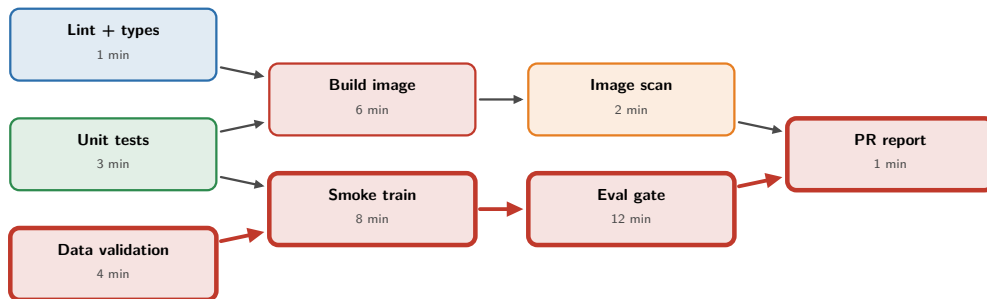
Every artefact, its immutable id and where it lives in the reference stack (AWS)

✗ Deploying :latest

A tag can be moved, so two pods may run different code and a rollback to :latest restores nothing. Pin images by digest (image@sha256:...) and models by registry version; enable tag immutability on the ECR repository.



The life of one change: every arrow is automated except the two human decisions (review, approval)



critical path: Data validation → Smoke train → Eval gate → PR report = 25 min

A PR pipeline as a DAG; durations in minutes. The critical path, not the sum, sets the wall-clock time

⚡ The trigger map

Trigger

	CI (test, build)	CT (retrain)	CD (deploy)	Eval only
PR on code	5			3
Merge to main	5	1	5	1
Tag v*	1		5	
Cron schedule		5		3
New data in S3		5		1
Drift alert		3		5
Base model update	3	3		5

Pipeline →

Which trigger starts which pipeline: 5 = always, 3 = usually, 1 = sometimes, 0 = never

One workflow, five triggers, routed by path (GitHub Actions)

on:

```

pull_request:
  paths: ["src/**", "tests/**", "pyproject.toml", "uv.lock"]
push:
  branches: [main]
  tags: ["v*"] # main -> test, v* tag -> prod
schedule:
  - cron: "0 3 * * 1" # weekly retrain, Monday 03:00 UTC
repository_dispatch:
  types: [new-data, drift-alert] # sent by EventBridge / monitor
workflow_dispatch:
  inputs: {data_snapshot: {required: true}}
  
```

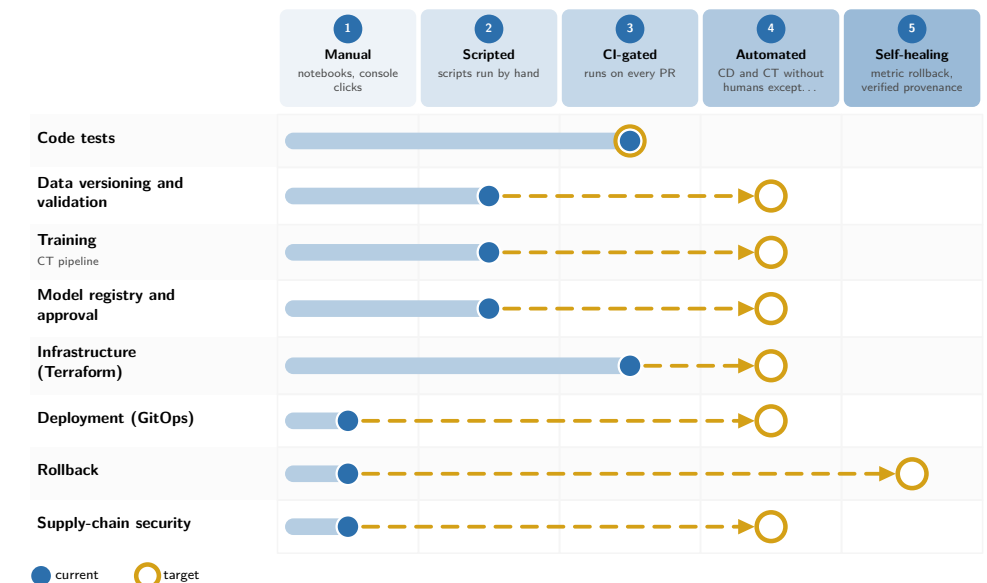
💡 TIP

Route by path: changes to `src/**` run CI, to `dvc.lock` or `data/*.dvc` run CT, to `deploy/overlays/**` run nothing in CI because Argo CD deploys them (sheet 06). Schedules run on the default branch only and use UTC.

📊 Pipeline-as-code maturity and delivery metrics

📖 INFO

Recall (mlops 03): Google's levels 0-2 go from manual, to an automated training pipeline (CT), to CI/CD of that pipeline. Below, the same journey is assessed on what is actually written as code.



● current ○ target

A team's pipeline-as-code assessment: dot = today, ring = target in two quarters

Metric	Formula	Good direction
Deployment frequency	$DF = \frac{N_{deploy}}{T}$	higher
Lead time for changes	$LT = \text{median}(t_{prod} - t_{merge})$	lower
Change failure rate	$CFR = \frac{N_{failed}}{N_{deploy}}$	lower
Recovery time	$RT = \text{median}(t_{restored} - t_{failure})$	lower

where N_{deploy} = number of production deployments in the period (count); T = length of the period (weeks); t_{merge} , t_{prod} = time a change was merged, time it ran in production (hours); N_{failed} = deployments that needed a rollback, hotfix or incident (count); t_{failure} , t_{restored} = time the failure started, time service was restored (minutes)

DORA delivery metrics, applied to model releases

PROBLEM

A team released models for 13 weeks by hand, then 13 weeks with CI gates, GitOps and Argo Rollouts. Compute DF and CFR for both quarters and the recovery-time gain.

GIVEN

- Before: 6 deployments, 2 failed, median recovery 180 min
- After: 26 deployments, 3 failed, median recovery 9 min
- T = 13 weeks each

1

Deployment frequency

before 6 / 13 = 0.46 per week; after 26 / 13 = 2.0 per week

Why: small, frequent releases are easier to review and roll back

2

Change failure rate

before 2 / 6 = 33 %; after 3 / 26 = 11.5 %

Why: gates catch bad models before prod

3

Recovery time

180 / 9 = 20 times faster

Why: rollback is a metric-triggered abort, not a manual rebuild

ANSWER

After: 4.3 times more deployments per week, CFR down from 33 % to 11.5 %, recovery 20 times faster

CHECK

✔ More deployments with fewer failures: the gates, not luck, explain it if the eval sets were unchanged

Is the new pipeline better? Compare one quarter before and after GitOps with automatic rollback

✓ Recap and self-test

YOU MAY HEAR

"Retraining is just running CI again"

BUSTED

✔ ACTUALLY

Retraining is a separate pipeline with its own trigger

CT runs on schedules or data events, on full data and GPUs; CI runs on PRs in minutes.

YOU MAY HEAR

"Git SHA identifies the model"

BUSTED

✔ ACTUALLY

SHA + data snapshot + params identify it

The same commit trained on another snapshot is another model.

YOU MAY HEAR

"Rollback means retraining the old model"

BUSTED

✔ ACTUALLY

Rollback means re-pinning the previous version

Keep the previous image digest and model version deployable at all times.

YOU MAY HEAR

"A schedule is enough for retraining"

BUSTED

✔ ACTUALLY

Add data-event and drift triggers, with guards

And a budget: retraining that never beats the champion should alert, not loop (sheet 07).

Beliefs that keep ML delivery manual

🔗 How is CI/CD for ML different from CI/CD for a web app?
Three artefacts instead of one (code, data, model), each versioned immutably; statistical gates against a baseline instead of only asserts; extra triggers (schedule, new data, drift) that start continuous training; and rollback restores a tuple of image, model and features.

Tick a box only when you could explain the point to someone else

☐

Every artefact has an immutable id

SHA, digest, snapshot hash, registry version

☐

Triggers are routed by path and event

code → CI, data → CT, manifests → CD

☐

The critical path sets CI time

parallelise around it

☐

Deployment = a pinned tuple

image, model, features, config

☐

Measure delivery

DF, lead time, CFR, recovery time

Tick when you could explain it to a colleague

Q1

Name four triggers ML pipelines use that web CI rarely does.

A

Cron schedule, new data in storage, drift or quality alert, base-model or upstream-model update.

Q2

Why pin an image by digest instead of tag?

A

A digest is the hash of the content and cannot be moved; a tag can point to new bytes later.

Q3

What identifies a model version fully?

A

Code SHA, data snapshot id, parameters, environment (image digest), recorded in the registry.

Q4 What runs on a PR, what runs on new data?

A PR: lint, tests, data validation, smoke train, eval gate. New data: CT, full retrain, champion-challenger evaluation, registration.

Q5 Change failure rate: 3 failed of 26 deployments?

A $3 / 26 = 11.5 \%$.

Q6 Why does adding runners stop speeding up CI?

A Wall-clock time cannot go below the critical path of dependent jobs.

Q7 What is rolled back when a model release fails?

A The whole tuple: previous image digest, model version, feature definitions and config.

Q8 Do GitHub scheduled workflows use local time?

A No, cron schedules are UTC and run on the default branch.

Q9 Which path change should not start CI in a GitOps setup?

A Environment overlays in the config repo: Argo CD deploys them after review.