

CI/CD: Automating Integration and Deployment

17-313 Fall 2026

Foundations of Software Engineering

<https://cmu-17313q.github.io>

Eduardo Feo Flushing

Administrivia

- P3 is out. P3A deadline next Monday, P3B the following Monday.
- Mid-semester grades: overall, everyone is doing great. Lots of opportunities to improve.

Midterm: General Feedback

- Great performance overall: the median was about 80%.
- Common mistakes:
 - Unclear answers. Clarity in written communication is a fundamental skill.
 - Bringing in concepts unrelated to the question.
 - Answers not grounded in the specific scenario given.
 - Overly generic responses, e.g. "random inputs to trigger failures," without specifics.
- Subjective questions? See next slide.

Why Some Questions Are Open-Ended

- In software engineering, the answer is often "it depends."
- Good decisions depend on the system, users, risks, and constraints.
- Knowing a concept is not enough. You must be able to:
 - Connect it to a scenario.
 - Compare alternatives and tradeoffs.
 - Justify your decision.
- Subjective does not mean arbitrary: answers must be consistent with the scenario and supported by course concepts.
- In practice, you will rarely be asked to define a term. You will be asked to make and defend a decision.

The goal is to assess applied reasoning, not memorization.

We have checks. When do they run?

We can test and analyze our software. But someone still has to:

- Remember to run the checks.
- Run them on the version the team will use.
- Notice failures and respond.
- Deploy the intended version to the intended environment.

Checks we already know

Testing & Fuzzing

Property-Based Testing

Static Analysis

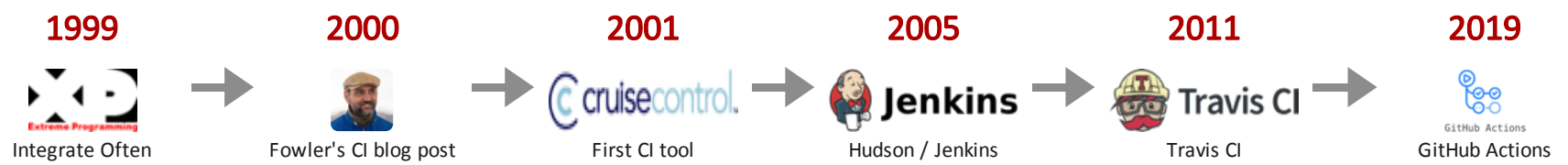
Dynamic Analysis

Knowing how to check software does not ensure that the team consistently checks and deploys it.

NOTES

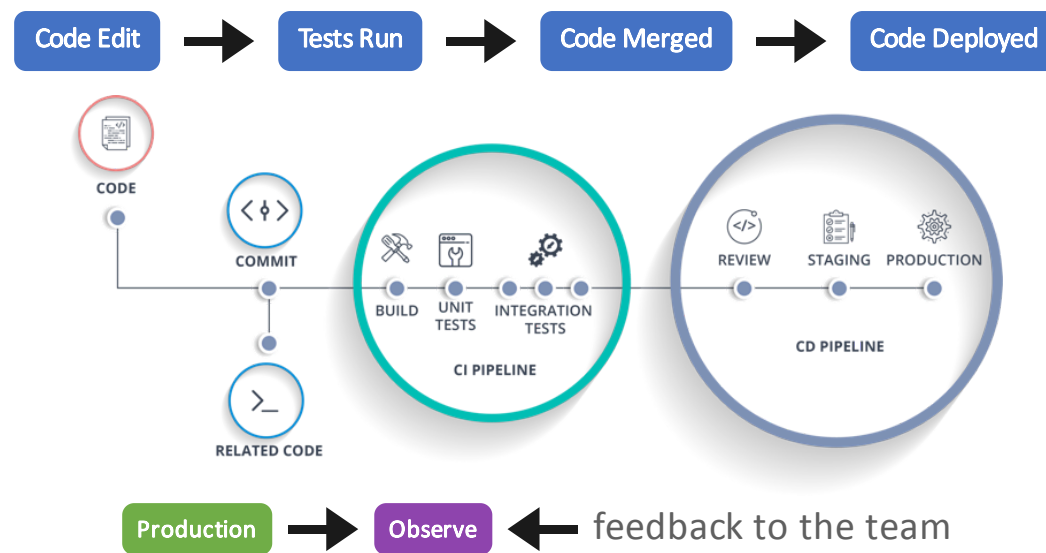
Automation makes checks part of the workflow instead of relying on someone remembering to run them. The team still needs to choose useful checks and decide how to respond to failures.

History of CI



The tools changed, but the core idea stayed the same: integrate often, and check automatically.

CI/CD pipeline overview



NOTES

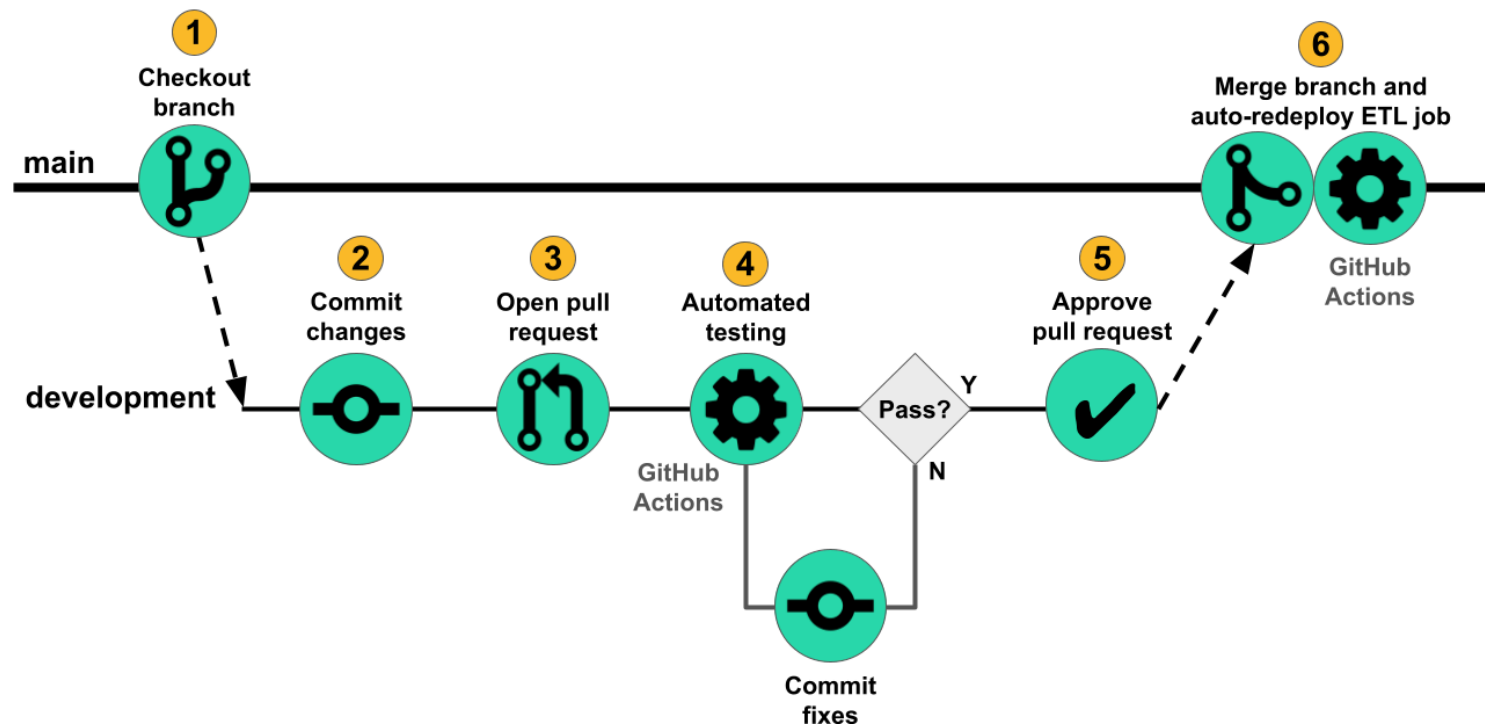
Each pipeline stage provides evidence or an output needed before a change proceeds. Monitoring closes the loop by showing how the deployed software behaves with real users and workloads.

Continuous integration: Integrate often, check automatically

Integrate frequently and get automated feedback on each change.

- CI = frequently integrating small changes into a shared mainline, supported by automated builds and checks.
- A CI service commonly starts after a push or pull-request event. Local checks can happen before a push.
- Having a CI configuration while keeping branches separate for weeks does **not** achieve frequent integration.

Example CI workflow



CI gives the team feedback before accepting a change into the shared codebase.

NOTES

If a check fails, investigate the cause, address it, and rerun the checks before merging. Also check your changes combined with the latest version of main, because they may conflict with changes that other developers have merged.

What triggers CI, and where does it run?

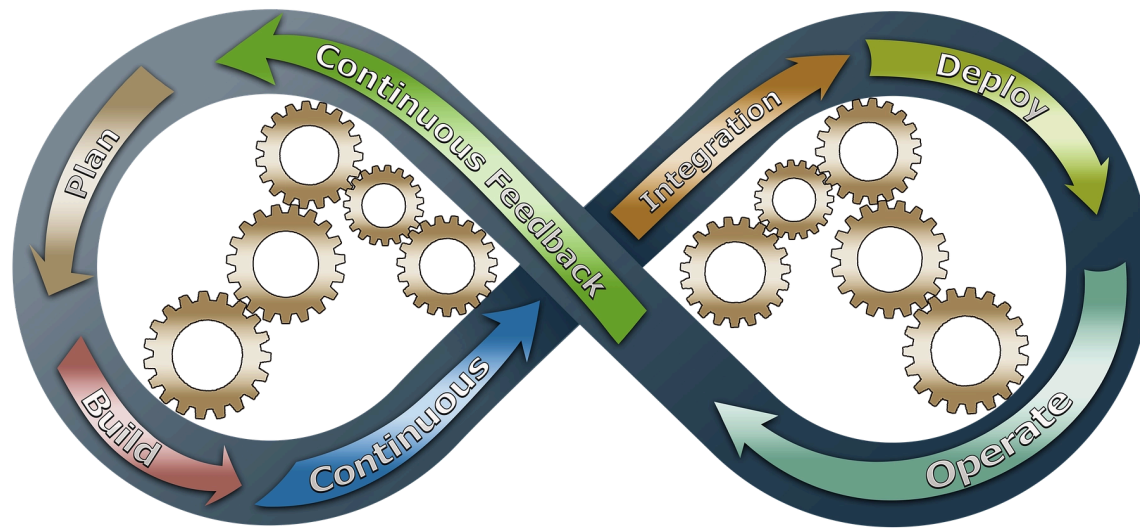


1. A configured repository event occurs (e.g. a commit or pull request).
2. A runner is provisioned with a declared environment.
3. The runner checks out a specific revision.
4. Dependencies are installed.
5. Build/check commands run.
6. A result is reported back.

NOTES

A runner executes the workflow against a specific revision, using declared dependencies and commands. Its logs help distinguish application failures from problems with the checks or execution environment.

Fast feedback makes failures easier to address



Useful feedback must reach the developer while the change is still easy to understand and revise.

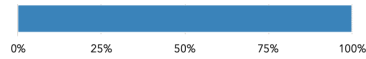
NOTES

Early feedback reaches developers while they still remember the change and have fewer possible causes to investigate. Pipelines therefore often run inexpensive checks first and longer checks later or separately.

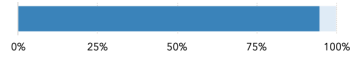
What do developers report about CI?

Hilton et al., "Trade-Offs in Continuous Integration: Assurance, Security, and Flexibility": 523 complete responses from over 30 countries; over 50% of participants had 10+ years of experience.

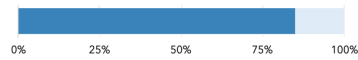
Do developers on projects with CI give more value to automated tests?



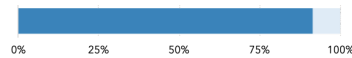
Do projects with CI have higher test quality?



Do projects with CI have higher code quality?



Are developers on projects with CI more productive?



Developers report benefits from CI, but useful automation still depends on how the process is implemented

NOTES

These survey results describe developers' perceptions, rather than proving that CI alone causes better quality or productivity. The benefits depend on the team's checks, workflow, and response to failures.

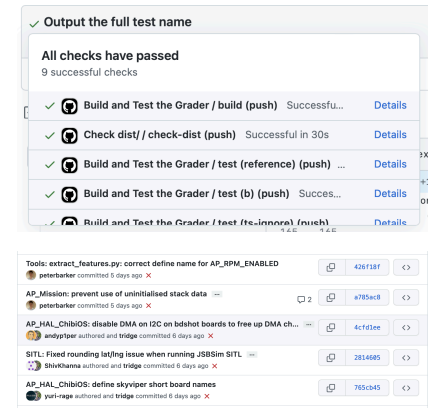
Attributes of effective CI processes

Policies

- Do not allow builds to remain broken for a long time.
- Run checks for every proposed change and relevant mainline update.
- Local checks should not be completely replaced by CI.

Infrastructure

- Fast checks on each change give feedback in minutes; longer scheduled or downstream checks can take longer.
- Results should be reproducible (deterministic).



Automation needs team policies that make its results consequential.

NOTES

Checks need consequences: required checks can block merging, and failures on main need prompt attention. Leaving main broken removes a dependable baseline and makes subsequent failures harder to interpret.

Frequent CI reduces the search for a failure’s cause

prestodb / presto build passing

Current Branches Build History Pull Requests More options

✓ master	This patch bumps Alluxio dependency to 2.3.0-2	→ #52300 passed	🕒 10 hrs 49 min 31 sec
James Sun		→ 36392a2	📅 2 days ago
! master	Handle query level timeouts in Presto on Spark	→ #52287 errored	🕒 11 hrs 6 min 44 sec
Andrii Rosa		→ aa55ea7	📅 2 days ago
! master	Fix flaky test for TestTempStorageSingleStreamSp	→ #52284 errored	🕒 11 hrs 50 min 37 sec
Wenlei Xie		→ 193a4cd	📅 2 days ago
✓ master	Check requirements under try-catch	→ #52283 passed	🕒 11 hrs 3 min 20 sec
Andrii Rosa		→ fff331f	📅 2 days ago
✓ master	Update TestHiveExternalWorkersQueries to creat	→ #52282 passed	🕒 10 hrs 55 min 37 sec
Maria Basmanova		→ 746d7b5	📅 2 days ago
✓ master	Introduce large dictionary mode in SliceDictionar	→ #52277 passed	🕒 10 hrs 43 min 30 sec
Maria Basmanova		→ a99d97a	📅 2 days ago
! master	Add Top N queries to TestHiveExternalWorkersQu	→ #52271 errored	🕒 10 hrs 46 min 36 sec
Maria Basmanova		→ 8b62d43	📅 3 days ago
✗ master	Fix client-info test-name output	→ #52266 failed	🕒 10 hrs 35 min 49 sec
Leiqing Cai		→ 467277a	📅 3 days ago
✓ master	Add Thrift transport support for TaskStatus	→ #52263 passed	🕒 11 hrs 13 min 42 sec
Andrii Rosa		→ fc94719	📅 3 days ago

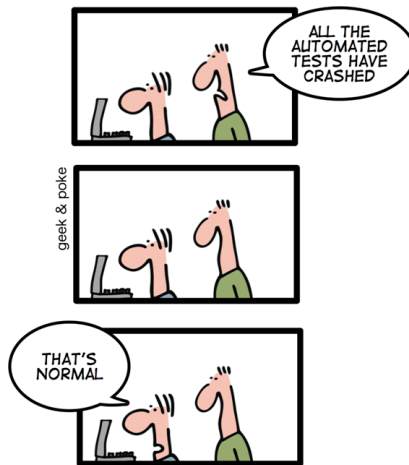
Small, frequent changes narrow the set of likely causes when a previously passing pipeline fails.

Smaller batches reduce investigation effort, even though the pipeline result still needs interpretation.

NOTES

One commit since the last passing run gives the team fewer changes to investigate than a week's worth of commits. A failure can still come from flaky checks or infrastructure, rather than the latest code change.

Flaky failures weaken the feedback loop



Possible causes: timing, shared state, dependencies, insufficient resources.

An unreliable check can train the team to ignore the very feedback CI is meant to provide.

NOTES

A passing rerun does not explain why the check failed earlier. If developers routinely dismiss failures as unreliable tests, they may overlook failures caused by actual bugs in new code.

Activity 1: Repair a CI process

A team uses this process:

1. The author runs tests locally if they remember.
2. A reviewer approves and merges the pull request.
3. On Friday, someone builds and tests main on their laptop.
4. If a check fails, they rerun it. If the next run passes, they merge without investigating.

In pairs, propose three improvements. For each, explain when the check runs, what is automated, and how the team responds to failure.

NOTES

Run builds and tests automatically on pull-request updates in a controlled environment, require passing checks before merging, and check main after merging. Investigate failures rather than rerunning until green, and promptly fix or revert changes that break main.

Continuous delivery: Why?

"The biggest risk to any software effort is that you end up building something that isn't useful. The earlier and more frequently you get working software in front of real users, the quicker you get feedback to find out how valuable it really is."

[Martin Fowler, Continuous Delivery](#)

NOTES

Passing technical checks does not establish that a feature is useful. A repeatable release process gets working software to users sooner so the team can evaluate whether it achieves the intended outcome.

A deployment mistake with serious consequences

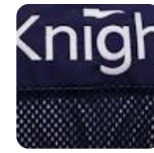
In 2012, a mis-deployed update cost a trading firm \$440 million in 45 minutes.

 Reuters

Knight Capital posts \$389.9 million loss on trading glitch

Knight Capital Group reported a third-quarter net loss due to a glitch on August 1 that forced the electronic trader to take on additional investors to avoid...

Oct 17, 2012



 Bloomberg.com

Knight Shows How to Lose \$440 Million in 30 Minutes

Talk about a bad day. In the mother of all computer glitches, market-making firm Knight Capital Group lost \$440 million in 30 minutes on Aug...

Aug 2, 2012



Deployment failures can have consequences even when the team has written and tested a new version.

Knight Capital: One server did not receive the update

*"In the week before go-live, a Knight engineer **manually deployed** the new RLP code in SMARS to its 8 servers. However, he made a mistake and did not copy the new code to **one of the servers**. Knight did not have a second engineer review the deployment, and neither was there an **automated system to alert** anyone to the discrepancy."*

henricodolfing.com, Project Failure Case Study: Knight Capital



A process that depends on remembering and correctly repeating steps leaves predictable opportunities for error.

NOTES

Missing one server left the deployment targets running different versions, illustrating why checking the code alone is insufficient. A reliable deployment process must also verify that every intended target received the intended version.

What could Knight Capital have done better?

- Use capture/replay (simulated) testing instead of the real market in their tests.
- Avoid including "test" and "dead" code in production deployments.
- **Automate deployments.**
- **Define and monitor risk-based KPIs.**
- **Create checklists for responding to incidents (risk management).**
- **Verify that every intended target is running the intended version.**

Automating execution and verifying the outcome address different parts of the problem.

CI, continuous delivery, and continuous deployment

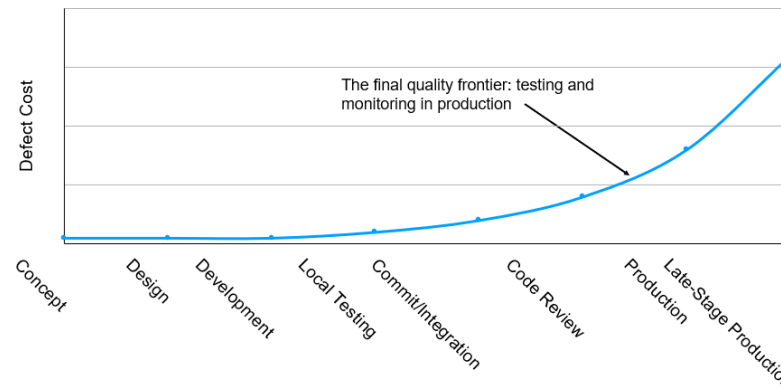
Practice	Commitment	Production decision
Continuous integration	Integrate small changes frequently and run automated builds/checks.	Does not by itself specify production deployment.
Continuous delivery	Keep qualifying changes ready for production through a repeatable, largely automated pipeline.	A person or policy may decide when to deploy.
Continuous deployment	Automatically deploy qualifying changes to production after required pipeline checks.	No routine manual approval for each qualifying change.

The key distinction between delivery and deployment is the production decision, not whether someone types the deployment commands.

NOTES

Continuous delivery can use fully scripted deployments while retaining a manual production approval. Continuous deployment removes that routine approval for qualifying changes; deploying code can still be separate from enabling a feature for users.

Frequent, small releases reduce risk



- Small batches, key performance indicators, phased rollout, evaluating user/business impact.
- "Faster is safer": small, controlled releases can be easier to diagnose and recover from. Speed alone does not make a release safe.

Automation makes small, repeatable releases practical; checks and recovery make them safer.

NOTES

Small releases leave fewer changes to investigate when something goes wrong, and automation makes frequent releases practical. The graph illustrates potential increases in defect costs

Promote through testing, staging, and production



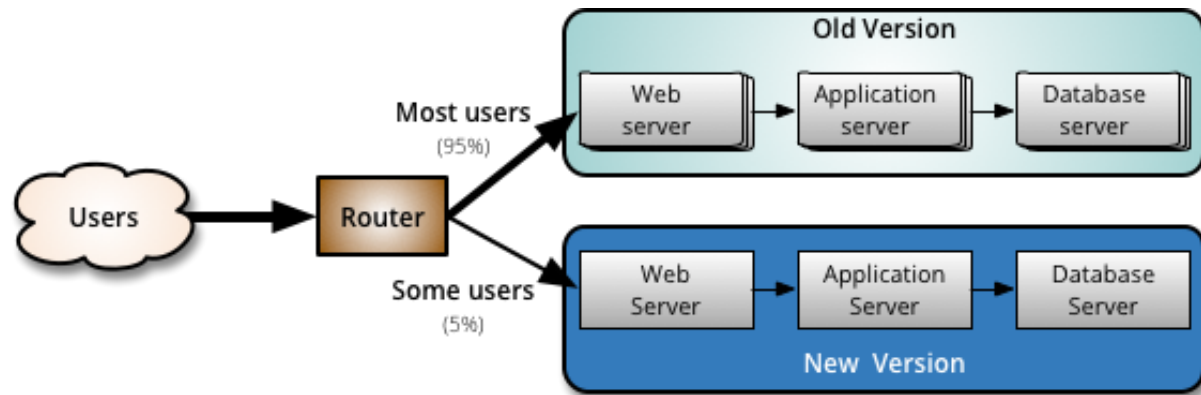
Revisions are promoted toward production; the same artifact identifier carries across environments.
QA takes place at every stage, including production.

Staging provides another opportunity to detect problems before exposing the full production audience.

NOTES

Staging approximates production but cannot reproduce every real workload or dependency. Beta testing involves selected users and dogfooding involves internal users; either may happen in staging or through a limited production release.

Limit the initial production rollout



A canary rollout sends a small share of production traffic to the new version, compares behavior, and expands only if results are acceptable.

A limited production rollout reduces initial exposure while providing evidence that staging cannot supply.

NOTES

A canary rollout gathers evidence from real production use while limiting initial exposure to problems. Expansion requires defined acceptance criteria and enough observation time and traffic; poor results should halt the rollout or return traffic to the previous version.

Continuous delivery needs monitoring

Consider both direct (e.g. business) metrics and indirect (e.g. system) metrics.

Hardware: voltages, temperatures, fan speeds, component health.

OS: memory usage, swap usage, disk space, CPU load.

Middleware: memory, thread/DB connection pools, connections, response time.

Applications: errors, latency, availability, successful user operations, status of third-party components.

Release health includes what users experience, as well as the state of the machines.

NOTES

Normal CPU or memory usage does not establish that users can successfully use the application. Interpret resource measurements alongside errors, latency, successful user operations, and changes in workload.

Monitoring can reveal operational issues



What changed around 13:00 and again around 20:00? What additional context would you need to explain it?

Measurements become more useful when they can be related to a particular deployment.

NOTES

CPU and memory usage drop around 13:00 and rise around 20:00, but the plots alone do not explain why. Traffic levels, deployment records, instance counts, and application metrics help distinguish improved efficiency from reduced activity or a failure.

Monitoring should lead to action



Example: automated rollback at Netflix based on SPS (stream starts per second), Netflix's key business metric for whether a user can actually watch what they want. The experiment (new version, red) falls behind the control (old version, blue) after the change.

Automation should include a response to failure and a way to verify that recovery worked.

NOTES

This is application-level monitoring in practice: Netflix compares a control group running the old version against an experiment group running the new version on SPS (stream starts per second). If the experiment's SPS deviates from the control, Netflix automatically rolls back within seconds, with no manual intervention required. In a real production scenario, we usually want to automate more of the response than just sending a notification.

P3: What is actually happening?



You should be able to explain what your team project automation does, which version it deploys, and how you know it succeeded.

NOTES

Activity: Try CD by yourself

1. Fork this repo: github.com/CMU-17313Q/basic-web-app-f26
2. Follow the instructions in the readme to run and test the development server locally.
3. Once it is running locally, visit `http://localhost:3000` and try different queries, such as "Who was Shakespeare?" and "What is your Andrew ID?"
4. Complete the activity tasks following the instructions at cmu-17313q.github.io/recitations/deployment-workshop-f26
5. Submit a link to the deployed site (Gradescope).