

DevOps & Deployment

Reading time: ~10 min · Prereqs: Lessons 01–07 · *Snapshot 2026 — the culture is stable; CI/CD and orchestrator brand-names move every couple of years.*

Why this matters to you

When you open the Spotify app and a fresh playlist appears, when your bank's mobile app gets a new "Send to a friend" button, when Notion ships a new slash command — that change started as a few lines of code on an engineer's laptop and arrived on your phone, probably the same day. Twenty years ago this took weeks of manual handoffs between developers, QA, and operations teams. Today the whole journey is mostly automated. The cluster of tools and habits that automate it is called **DevOps**. Knowing the moving parts makes you a better product manager, recruiter, executive, or curious bystander — you can read a deployment diagram, ask the right questions in a planning meeting, and stop being mystified when someone says "we'll ship it after the pipeline goes green."

1. From "throw it over the wall" to "you build it, you run it"

Old-school IT split labor: developers wrote code, then handed a zipped folder to a separate **operations** team to install on a server. The two teams used different tools, had different incentives ("ship fast" vs "don't get paged at 3 AM"), and blamed each other when things broke. **DevOps**, the cultural movement that emerged in the late 2000s, dissolved that wall: the same people who write the feature also write the deployment scripts, get the pager when it breaks, and own the customer outcome. The tooling chapter of DevOps — CI, containers, IaC, monitoring — is what grew around that culture to make ownership practical at scale. **SRE** (Site Reliability Engineering, a Google-coined sibling discipline) overlaps almost entirely; treat the two as the same job family for the purposes of this lesson. The shorthand "you build it, you run it" was popularized by Werner Vogels at Amazon and is now industry default *as of 2026*.

2. CI — Continuous Integration

Every time a developer pushes a change to a shared git repo, a **CI pipeline** automatically runs: it pulls a fresh copy of the code, installs dependencies, compiles, runs the test suite, runs linters, and builds artifacts. If anything fails, the change is blocked from merging. As of 2026 the popular CI hosts are **GitHub Actions** (now dominant, especially in open source), **GitLab CI**, **CircleCI**, **Buildkite**, and **Jenkins** (the open-source granddaddy, still entrenched in many large enterprises). All work roughly the same way: you commit a small YAML file to your repo that declares "on every push, run these steps in this order on this kind of machine". The result you see is a green check or a red X next to every pull request — the modern code reviewer's first signal. CI also pays a quieter dividend: every test that exists in your repo is guaranteed to run on every change, which catches the regressions humans forget about.

3. CD — Continuous Deployment / Continuous Delivery

Once tests pass, the natural next step is to **deploy** the new code. **Continuous Delivery** means every successful build is *ready* to deploy (a human still clicks the button). **Continuous Deployment** means every successful build *is* deployed, automatically, no human in the loop. The most mature teams (Amazon, GitHub, Shopify) deploy thousands of times a day. The trade-off: full continuous deployment requires excellent tests and excellent monitoring — because there's no human gate, the system itself has to catch regressions. Most teams as of 2026 sit on a spectrum somewhere between manual once-a-week deploys and full continuous deployment, often using **feature flags** (LaunchDarkly, Unleash, or homegrown) so the deploy and the launch can be decoupled.

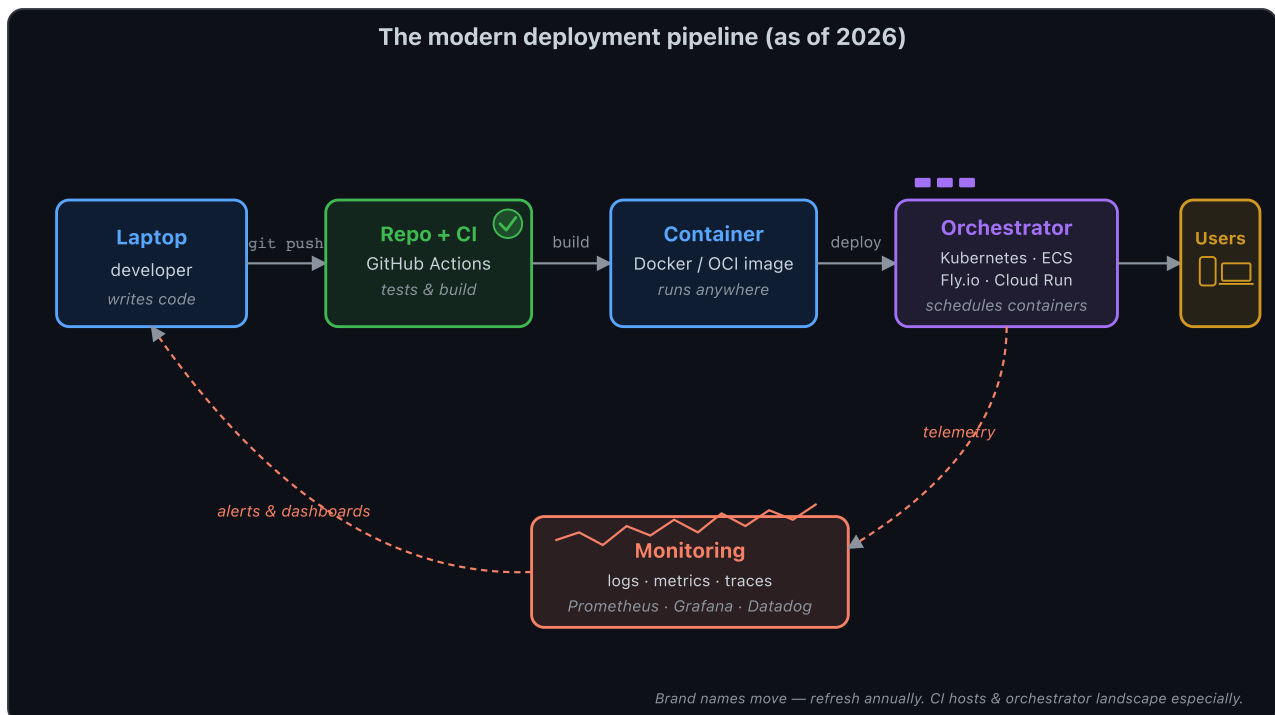
4. Containers — the "works on my machine" cure

A **container** is your application plus *everything it needs to run* (the OS libraries, the language runtime, the dependencies, the config) packaged into a sealed image. The image runs identically on your laptop, on the CI runner, on a colleague's machine, and on the production server. **Docker** is the brand that mainstreamed this approach in 2013 and is still where most developers first learn it. The image format itself is now an open standard (**OCI**). Containers are *not* virtual machines — they share the host's kernel and

start in well under a second, where a VM takes a minute. The shift to containers (~2014–2020) was the single biggest change in how software gets shipped this century — and it's what made the next tool possible.

5. The modern deployment pipeline, one diagram

Here is the full journey of a change from a developer's laptop to a user's phone, with a monitoring loop back to the developer. When you read a deployment diagram in the wild, you can map every box on it onto one of these stages.



6. Orchestrators — running many containers

One container is easy; a hundred containers across fifty machines is not. An **orchestrator** schedules containers onto machines, restarts them when they crash, routes incoming traffic to healthy ones, scales them up or down with demand, and replaces them during deploys without dropping requests. **Kubernetes** (also known as **k8s**, originally from Google) is the heavyweight industry standard — powerful and famously complex. Most teams *as of 2026* do NOT run their own Kubernetes; they pay a cloud to run it for them: **EKS** (AWS), **GKE** (Google), **AKS** (Azure). Lighter, opinionated alternatives have grown popular for smaller teams: **AWS ECS / Fargate**, **HashiCorp Nomad**, **Fly.io**, **Render**, **Railway**, and **Google Cloud Run**. (This space is on [refresh-checklist.md](#) as medium-churn — the vendor list will shift between annual refreshes.)

7. Infrastructure as Code

IaC means you declare your servers, networks, databases, DNS records, and other cloud resources as text files — versioned in git alongside your application code — rather than clicking around a cloud console. **Terraform** (HashiCorp; with the **OpenTofu** open-source fork after the 2023 license change) is the multi-cloud standard *as of 2026*.

Pulumi lets you write the same kind of declarations in Python or TypeScript. **AWS CloudFormation** is the AWS-native version. **Ansible** is more about *configuring* existing servers than provisioning new ones. The upshot: your infrastructure becomes reviewable, reproducible, and disaster-recoverable from a git repo — the same workflow you already use for code.

8. Monitoring & observability

Once your code is running, you need to know if it's healthy. Three telemetry families: **logs** (lines of text — Datadog Logs, AWS CloudWatch, Splunk, Loki); **metrics** (numbers over time — Prometheus + Grafana, Datadog Metrics); **traces** (a single user request's path across services — Jaeger, Honeycomb, Datadog APM). The whole field has rebranded as **observability**; the unifying open standard is **OpenTelemetry**. If a tree falls in production and nobody's monitoring, your users tell you on Twitter — and the post-mortem starts with "why didn't our alerts fire first?"

What people argue about

- **Kubernetes everywhere vs Kubernetes nowhere.** Large enterprises love it; many small-to-mid teams now say "we don't need it" and run on Fly.io, Render, or Cloud Run instead. The pendulum is currently swinging away from default-k8s for teams under ~50 engineers *as of 2026*.
- **Self-hosted CI (Jenkins) vs SaaS CI (GitHub Actions).** SaaS won mindshare, but self-hosted is making a comeback in regulated industries and at very large scale — cost, sovereignty, and custom hardware are the usual drivers.
- **Multi-cloud as a goal.** Every CTO claims to want it; almost nobody achieves it. The practical reality for most teams is "pick one cloud, get really good at it"

— multi-cloud only earns its weight at very large scale or under explicit regulatory pressure.

Reading on

- **The Phoenix Project** by Gene Kim — the classic novelization of how DevOps culture arose. Easier than it sounds; a good first read for non-engineers.
- **Accelerate** by Forsgren, Humble, and Kim — the data behind why fast, automated deployment correlates with business performance. Short and quantitative.
- **The DORA report** — dora.dev. Annual, free; the standard industry benchmark on deployment frequency, lead time, change-failure rate, and recovery time.
- **Lesson 03 of this course** — returns to the cloud platforms where most of this machinery actually runs.

LAST-REVIEWED: 2026-05-25