Engineer

DevOps

∞ practices that join software development and operations

What it is

DevOps is a way of organising software work in which the people who write a system also own how it is built, released and kept running, and the steps between writing code and running it are automated rather than handed from one team to another. The name is a blend of development and operations, the two groups it was coined to bring together.

NIST, the US National Institute of Standards and Technology, defines it as practices for automating the processes between development and operations teams “so that they can build, test, and release software faster and more reliably”. That definition is useful because it names the part that matters: the processes in between. A company can have excellent developers and excellent operators and still lose days at the point where one hands a release to the other.

In practice, DevOps shows up as a handful of habits that reinforce each other:

- **Shared ownership of production.** Whoever changes the code sees what the change does once it runs, through monitoring, logs and alerts, and answers for it.
- **An automated path to release.** Build, test and deployment run as a pipeline, the same way every time, usually on top of CI.
- **Infrastructure written as files.** Servers, networks and databases are described in templates or playbooks kept beside the code, so an environment can be rebuilt instead of remembered.
- **One build, many environments.** The artefact that passed its tests in staging is the same one that reaches production, with only its settings changed.
- **Small, frequent releases.** Each one carries few changes, so a fault is easy to find and easy to roll back.

DevOps is not a product, and it is only loosely a job title. Vendors sell tools under the name and companies hire “DevOps engineers”, but neither is the thing itself. A DevOps engineer usually builds and looks after the pipelines and infrastructure that let other developers ship on their own, which is a role inside the practice rather than a substitute for it.

Use the concept in a decision

Put “What is DevOps?” to work in a real decision. We can help relate the concept to the system you are building or buying. Tested, released and reversible, so changes reach customers on a Tuesday afternoon rather than late on a Friday night.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Automated GIS SaaS deployment with GitHub Actions — deployment, data and reporting, all off the hands

Automated GIS SaaS deployment with GitHub Actions Get in touch →

Why it matters

Every hand-off between teams loses something. The developer knows why a change was made and what it depends on; the operator knows what the servers look like and what broke last month. When a release crosses from one to the other as a ticket and a package, both halves of that knowledge arrive incomplete, and the gaps surface during deployment, which is the most expensive time to find them.

The Twelve-Factor App, a methodology for how software-as-a-service apps are built, names three of those gaps precisely. There is a gap in time, when code written today reaches production weeks later. There is a gap in people, when one group writes the code and another deploys it. And there is a gap in tools, when a laptop runs one database and production runs another. DevOps is largely a project to close all three: deploy hours after writing, let the authors deploy, and keep every environment as alike as possible.

Release size is the second argument. Google’s book on site reliability engineering puts it plainly: frequent releases mean fewer changes between versions. A release with three changes in it is easy to reason about when it goes wrong. A release with three hundred is a search.

The third is toil. The same book defines toil as work tied to running a service that is manual, repetitive, automatable and without lasting value, and that grows in step with the service. A team that restarts a process by hand every Tuesday,

renews certificates from a calendar reminder and resizes disks when an alert fires is spending its time on work that a script would do more reliably. Automating that work is how a small team keeps up with a growing system.

Security guidance has settled on the same shape. NIST's guidance on DevSecOps for cloud-native applications starts from a pipeline with stages for building, testing, packaging, deployment and operations, and counts infrastructure, runtime policy and monitoring among the kinds of code that pipeline describes.

What it looks like in practice

The pages on this site show DevOps less as a title than as a scope of responsibility: the same engineer designing a system, getting it to production and keeping it there.

For a mapping product, the founder's work spanned infrastructure, data processing and the customer-facing map application for two years. The work covered architecture, implementation and support across those layers: designing, building and maintaining the operating environment, data-processing workflows and map application. That connects development and operations in one engagement. The record establishes the scope and duration of that engineering and maintenance, rather than uptime or response-time measurements.

On a database estate of 10 PostgreSQL and MS SQL Server instances on Ubuntu Linux VPS, the operational side was automated piece by piece: backups through `pg_dump` and SQL Server Agent jobs, monitoring in Prometheus and Grafana, regular patching, UFW firewalls, role-based access control and documented disaster recovery. The page reports 99.9% uptime across all ten servers and manual maintenance cut by 50% through automation, which is the toil argument measured on one estate.

On an Azure platform with development, staging and production environments, deployment was owned end to end along a fixed route. Owning it meant owning every link in that path: CI/CD builds the images, Bicep templates keep each stage on matching infrastructure, and the settings particular to each stage are held separately from secrets. One built artefact is therefore promoted from stage to stage instead of rebuilt for each, and changes reach every environment predictably, without a single ad-hoc manual deploy.

Where it goes wrong

DevOps fails quietly, usually by keeping the vocabulary and losing the habits.

The title arrives and the wall stays. An operations team is renamed, or a "DevOps team" is hired to sit between the developers and production. Releases still cross a hand-off; it just has a new name on the other side. The test is simple: ask who can ship a change and who gets woken when it breaks. If those are different people with different information, the practice has not arrived.

Deployment is still a person following a checklist. The pipeline builds and tests, and then someone logs in to a server and runs the last five steps from memory, slightly differently each time. Put the last steps in the pipeline too, give releases one route through the environments, and treat a manual push to production as an incident to explain.

Each environment gets its own build. Rebuilding for staging and again for production means the thing that was tested is not the thing that shipped. A dependency that resolved differently on the second build turns up as a fault nobody can reproduce. Build once, tag the artefact with the commit it came from, and promote that artefact, with settings coming from the environment:

```
tag="$(git rev-parse --short HEAD)"
docker build -t "registry.example.com/app:$tag" .
docker push "registry.example.com/app:$tag"
# staging and production run this same tag;
# only the settings they start it with differ
docker run --env-file staging.env "registry.example.com/app:$tag"
```

Staging drifts away from production. A setting changed by hand on one server, a database upgraded in production first, a library version pinned in one place and floating in another: each makes staging a little less useful as a rehearsal. Describe the infrastructure in templates, apply the same templates everywhere, and run the same backing services in development that production runs.

Toil grows with the system. Manual jobs that took ten minutes a week for one server take hours for twenty. Without a deliberate budget for automating them, the team ends up spending its days keeping the lights on. Count the recurring manual work, and cap the share of time it is allowed to take.

Further reading

- DevOps — the NIST definition, with the special publications it is drawn from.
- Build, release, run — why a build is made once and combined with configuration to form a release.
- Dev/prod parity — the gaps in time, people and tools between development and production, and how to close them.
- Config — keeping everything that varies between deploys out of the code and in the environment.
- Eliminating Toil — Google's definition of toil and the case for capping it.
- Release Engineering — hermetic builds, self-service releases and policies enforced by the release tooling.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is DevOps?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/devops/