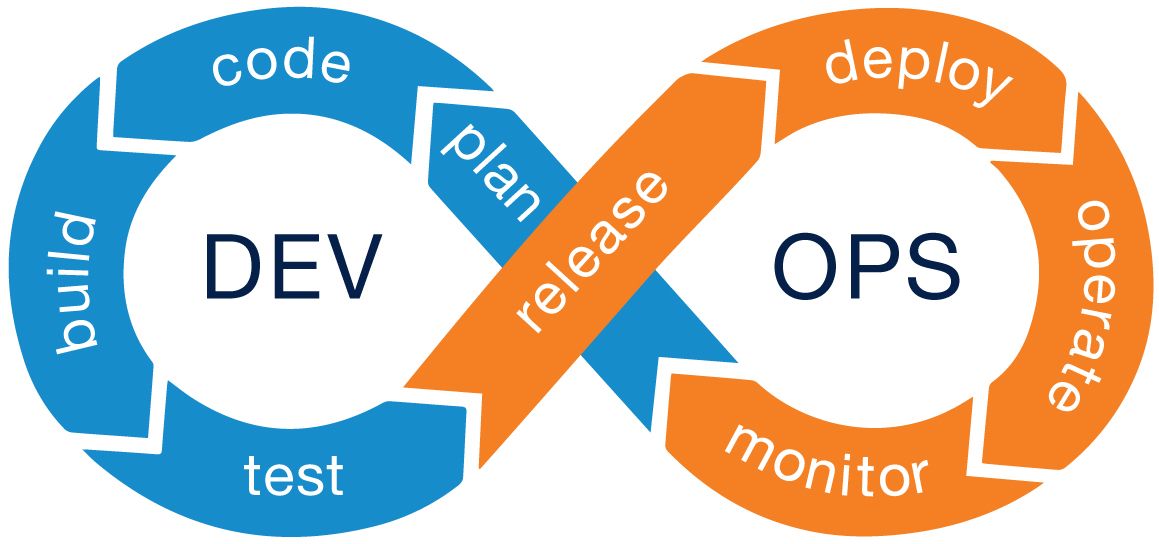
**What is DevOps?**



**DevOps is a methodology that helps engineering teams build better products by continuously integrating user feedback.**

**DevOps at the organization level**

If you search images for "DevOps", you'll likely see an image like the one below.

Photo credits: University of California at San Fransisco

DevOps was originally envisioned as a way to structure engineering organizations to efficiently integrate user feedback:

1. A plan is made for what users might need, often in scrum meetings between product managers and developers.
2. Developers write code based on what was planned.
3. The code is built into artifacts (containers, apps, executables, etc.)
4. Those artifacts are tested automatically and/or by QA teams.
5. Once the artifacts are confirmed to be error free by testing and QA, they are combined into a release (version 2.0.0, for example)
6. That release is distributed (the file is hosted, the containers are deployed, the apps are published, etc)
7. After the release is distributed, the application need to be monitored for code errors and scaling problems.
8. Monitoring keeps track of key metrics: Does the product feel slow, is it down?

Finally, usage metrics and feedback are incorporated into the planning phase (often in the form of Agile and "user stories").

**The three pillars of DevOps Engineering**

Another common definition for DevOps is as an umbrella term for "infrastructure for building and deploying code." Many developers have titles like "DevOps Engineer" or "Platform Engineer." Their jobs have little do to with planning or coding, and more to do with three key pillars:

* Pull request automation
* Deployment automation
* Application Performance Management (APM)

The goal of DevOps engineering is to release high quality software quickly, and make sure it continues being delightful and bug-free for end users.

**Pillar #1: Pull request automation**

Developers commonly share work with each other by proposing new sets of changes. The likeliest tools used are GitHub, BitBucket, and GitLab. Developers use git and push sets of changes together and open a "pull request" (or "merge request") to get those sets of changes integrated into the primary codebase.

Commonly, each set of changes is reviewed by another developer on the same team, but the review process can include many stakeholders in larger teams. The likeliest people to conduct reviews are:

* The developers that know the areas of code being changed (often called "code owners")
* An engineering manager or product manager in charge of the functionality being proposed.
* For visual changes: The designers that created the original specification.
* Translators, accessibility stakeholders, security reviewers, and other non-technical parties.

**What can be automated**

The business goal for a DevOps Engineer working on pull request automation is to speed up the review process. Some examples of functionality that can be built out include:

* Automated test running with a CI provider ([what is CI?](https://webapp.io/blog/what-is-ci/)), which gives developers confidence that the change does not break existing functionality
* Per-change ephemeral environments, which helps interested parties actually interact with the proposed change to ensure it solves all required business goals.
* Automated security scanning, which helps ensure that proposed changes do not introduce new vulnerabilities into the product.
* Notifications to reviewers automatically, so that the correct reviewers can quickly request changes to a pull request.

Ideally, changes should be reviewed and merged within 24 hours of when they are made. The most common reason that software developers are slowed down is a long review cycle.

**Pillar #2: Deployment automation**

In a [famous post from the year 2000](https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/), Joel Spolsky (founder of Stack Overflow) places "Can you make a build in one step?" as the second most important question for the health of a code base.

The efficiency of a build process isn't the only goal of Deployment automation, other goals include:

* Deploying a feature to a certain set of users as a final test before rolling it out more publicly (feature flagging and canary deployments)
* Starting new versions of services without causing downtime (blue/green deployments and rolling deployments)
* Rolling back to the prior version in case something does go wrong.

It's easy to overcomplicate deployments. Many companies have complex internal platforms for building and distributing releases.

Broadly, success in deployment automation is finding the appropriate deployment tools to fulfill business goals, and configuring them. In an ideal world, there should be little-to-no custom code for deploying.

**Pillar #3: Application performance management**

Even the best code can be hamstrung by operational errors. [User generated content brought down Stack Overflow when a user submitted a popular post with a lot of leading spaces, for example.](https://stackstatus.net/post/147710624694/outage-postmortem-july-20-2016)

The core goal of this pillar is to ensure that your service continues to perform well in production. It has four core targets for automation:

1. **Metrics**: Numerical measurements of key numbers in production. Usually in the form of finite resources (disk space, memory usage) and times (average response time, job processing time, etc)
2. **Logging:** Text descriptions of what is happening during processing. Logs often come with metadata about their source, time, and related metrics.
3. **Monitoring:** Take the metrics and logs and convert them into health metrics: Does the product feel slow, is it down, are all of the features working without errors?
4. **Alerting:** If monitoring detects a problem, the correct problem-solver should be notified automatically: In the case of an outage, an on-call engineer should be notified. Performance issues and errors with features often automatically log tickets.

**Example DevOps Engineering timelines**

A new product should not dive into complete automation for all of the pillars of DevOps engineering. Developers would add automation as the situation required, especially in response to user churn.

1. **A new startup with no users building a webapp**: Pillars #2 and #3 are essentially useless - outages will not be noticed by anyone. Such a product should only ensure developers can collaborate quickly. Stack: [Netlify](https://www.netlify.com/)/[Vercel](https://vercel.com/)/webapp.io
2. **A team building an app for 10 enterprise users:**Enterprise users are more sensitive to downtime. Test coverage and business-hours alerting should be priorities. Stack: [Sentry](https://sentry.io/), [PagerDuty](https://www.pagerduty.com/), [CodeCov](https://about.codecov.io/), [Bitrise](https://www.bitrise.io/)/[CircleCI](https://circleci.com/)
3. **A social media app like Reddit:**Reddit's users are in many timezones, and very sensitive to downtime. All three pillars are of vital importance. Stack: [Sentry](https://sentry.io/), [Elasticsearch/Logstash/Kibana](https://www.elastic.co/what-is/elk-stack), [Pingdom](https://www.pingdom.com/), [LaunchDarkly](https://launchdarkly.com/), [Terraform](https://www.terraform.io/)

**Conclusion**

DevOps engineering is vital for developer teams. Without being cognizant of its three pillars, customers will have a confusing and disappointing experience.

New products do not need to automate very much, but as a product matures and gets more users, it's important to dedicate resources to DevOps engineering.