# DevOps

DevOps is a common buzzword in software development. But whether you're new to the industry or have been in it for years, you should know what DevOps is and how it may impact the way you work. In this lesson, you'll explore DevOps and examine its culture and common components.

## What is DevOps?

The definition of DevOps depends on who you ask! But for most experts, DevOps boils down to the following:

> ***DevOps*** *is an approach to software development in which development (Dev) and operations (Ops) teams collaborate to deliver rapid, high-quality software updates.*

Let's break down key components of this definition.

### Development

The "Dev" in DevOps refers to the **development** team. This team writes, edits, and documents the software's code and maintains it in a code base or repository.

### IT operations

The "Ops" in DevOps refers to the **information technology (IT) operations** team. This team sets up, manages, and optimizes the infrastructure on which development depends, including the hardware, software, network, and cloud storage. The team also deploys software updates to users and otherwise supports the code.

### Speed

One goal of DevOps is to release deliverables—products or services that are part of a project—quickly and frequently through rapid deployment. With DevOps, organizations release small but continuous updates instead of large, infrequent changes.

### Quality

Deliverables must be high quality, meaning they meet stakeholders' expectations for performance, functionality, stability, reliability, availability, and security. These stakeholders include development and operations teams but also security teams, line of business teams, and users, among other parties.

## Culture

DevOps only works if the organization embraces a DevOps culture, and the cornerstone of that culture is **collaboration.**

Traditionally, development and operations teams have worked separately in their own silos. But with DevOps, they work together and maintain open communication throughout the **software development lifecycle (SDLC)**. They share information freely and transparently and work together to ensure that developers receive **quick feedback** about any software issues.

The teams also **share responsibility** for product quality.

- Developers not only build the code but test it and help maintain it over time; they don't leave these tasks to the operations team.
- And operations personnel participate in planning discussions for software updates. By doing so, they can ensure that developers build in monitoring, logging, security testing, and other features that the system requires.

By collaborating throughout the SDLC, the teams better understand each other's needs. In turn, they can adjust their work practices to address those needs, improving team rapport and eliminating obstacles that delay releases.

## Components

Let's discuss other commonly recognized components of DevOps.

### Agility

DevOps typically involves an agile approach to software development. To explain the agile approach, let's first discuss its predecessor, the waterfall approach.

The **waterfall approach** is linear. The team follows a long, detailed plan with distinct sequential phases such as planning, design, development, testing, and support. This plan specifies deadlines for each phase and responsibilities for each team member, and the goal is to execute the plan as precisely as possible.

But this approach presents several problems for software development.

- As the project moves forward, developers may encounter obstacles or think of new, better ideas, and clients may change their requirements. Yet the rigid plan makes adapting difficult and sometimes costly.
- What's worse, clients and users typically don't see the product until the testing phase, so developers don't receive feedback until then. Addressing bugs and client concerns

that late in the process may be more costly and time-intensive than earlier in development.

In contrast, **agile development** is iterative. Instead of completing the project in longer, distinct phases, teams work in short iterations, known as sprints, usually several weeks long each. The goal of each sprint is to produce a small deliverable on which the client can provide feedback quickly. This way, developers can learn the client's concerns early on and account for them moving forward. Plus, testing occurs throughout each sprint instead of late in the project's timeline, so teams identify and fix bugs sooner rather than later.

Like a waterfall project, an agile project still has an overall goal. However, the plan is less detailed and more flexible to accommodate new ideas and requirements.

In short, the agile approach is ideal for DevOps.

- Focusing on brief sprints, small deliverables, and continuous testing meshes well with DevOps's goal of frequent, high-quality software updates.
- Developers and operations personnel can quickly adapt to changing project demands and quality standards.

## Microservices

To speed up deployment, DevOps requires an application design focused on microservices. A **microservice** is a small, independently functioning component of an application that can work with other components to comprise the complete application.

In traditional development, developers build all parts of an application into one large, monolithic piece. When adding a feature or fixing a bug, developers must redeploy the entire application.

But when using microservices, developers only need to edit the relevant component, not the entire application. These smaller-scale changes make updates quicker and reduce the risk of breaking the application.

## Automation

DevOps involves using tools to **automate** code's movement through the SDLC.

Take testing, for example. Code must undergo different levels of testing throughout its lifecycle, and various tools can automatically run the appropriate tests for each stage. Tools can also prepare and configure infrastructure components, monitor performance and security issues, identify incompatible components or assets, and deploy code to production.

Automation reduces the time to deployment and the chance for human error. It also frees developers and operations personnel to focus on coding, problem-solving, and other crucial tasks to improve the product.

## Continuous integration and continuous delivery

Continuous integration and continuous delivery (CI/CD) are two automated processes essential for DevOps.

- **Continuous integration (CI)** is a process in which developers regularly integrate small pieces of newly written code into their main code base. The code undergoes automatic testing, and the code base manager reviews and approves it before integration.
- **Continuous delivery (CD)** comes next. It’s a process that sends this new code to a preproduction environment to verify that it’s ready for quick and safe deployment to production. Here, the code undergoes additional testing and review, and then moves to production.

CI/CD is invaluable to DevOps.

- It automates testing, saving time otherwise spent manually testing code and providing instant feedback to developers about bugs.
- It automates cybersecurity checks like application and vulnerability scanning, which are standard parts of **DevSecOps** (DevOps with security built in).
- Frequent testing, scanning, and review also mean higher quality code.
- CI/CD automates deployment to the preproduction environment.

With CI/CD, updates get to production quicker and with less effort.