What Is DevOps? A Comprehensive Introduction

April 16, 2021
7 minute read
Shanika Wickramasinghe

DevOps is a trending topic that's popular for increasing company productivity, no matter your industry. Every day more companies work to bring this disruptive model to their organizations.

DevOps has one primary goal to achieve: continuous integration to continuous delivery.

In doing so, the development and operations processes become faster and more resource friendly. Companies can save money while producing more high-quality software products for customer consumption or internal use.

In this introductory article, we'll explain the basic concepts of DevOps, including how we got here, best practices, key topologies, and the most common benefits of a DevOps environment.

(This article is part of our DevOps Guide. Use the right-hand menu to go deeper into individual practices and concepts.)

What is DevOps?

Traditionally in the IT industry, employees belonged to the development or 'dev' team. A separate team known as IT operations or 'Ops' worked to support IT.

Dev and Ops teams had separate leadership, responsibilities, and objectives to achieve. In many companies, the teams worked on separate floors and rarely communicated. This 'silos' culture led to poor communication and collaboration among these teams—a rigid barrier between them that they'd bridge only when absolutely required.

The major organizational culture shift of DevOps successfully broke this silos mentality, bridging the gap between the Dev and Ops teams by making them:

- Work together
- Share responsibilities
- Take full ownership of the software they deliver throughout its software lifecycle

Nowadays, DevOps is widely embraced by the IT industry, including leading tech companies such as Amazon, Facebook, Google, Netflix, and BMC Software. More recently, companies are looking at how to spread the principles of DevOps across the entire organization, beyond IT.

bmc
Traditional Dev
Traditional Ops
DevOps

How DevOps developed

Before the year 2000, most IT industries adopted the classical waterfall model, a linear approach for software development.

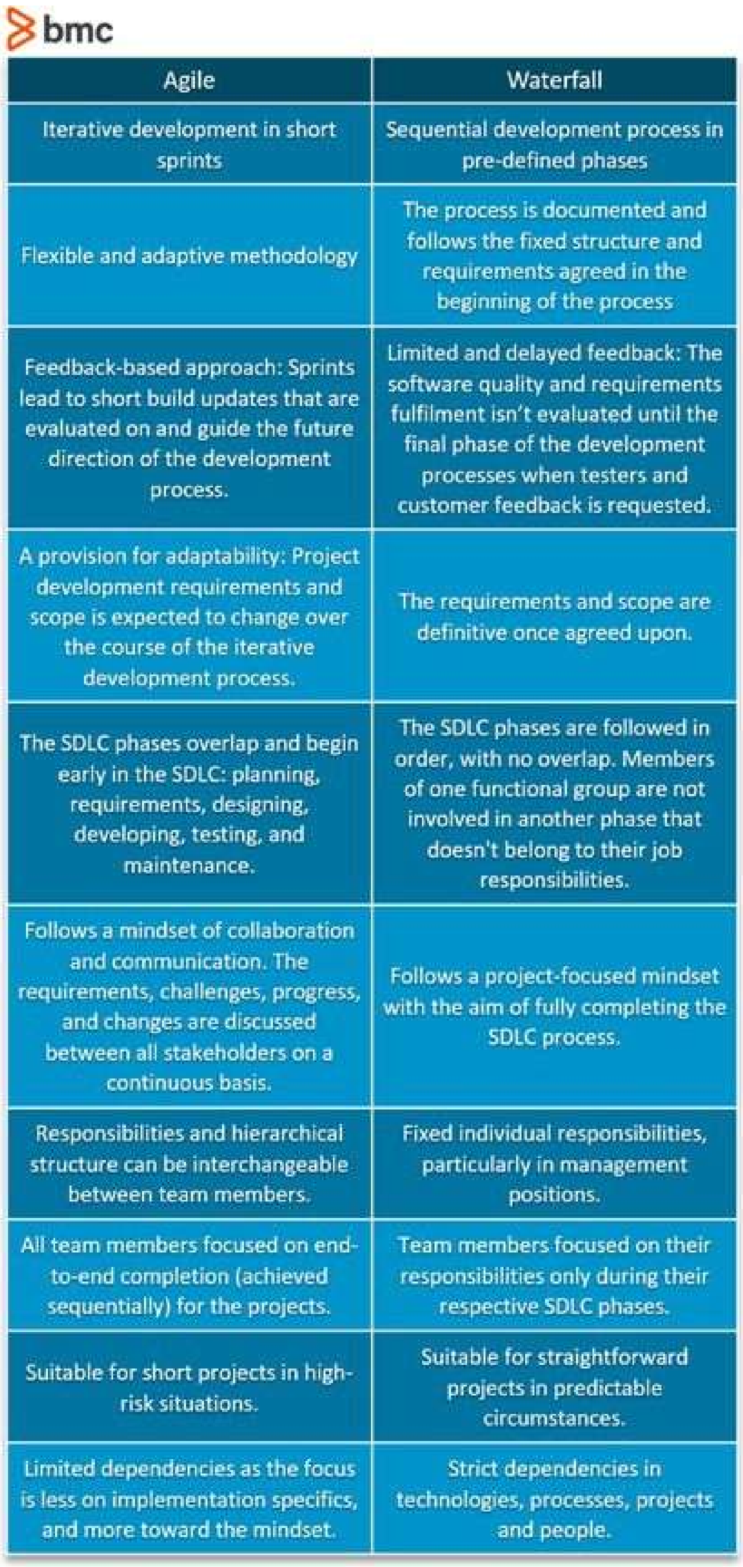
- Developers had to spend a lot of time developing and integrating heavy pieces of code.
- QA engineers and operations teams, who worked in silos, spent more time testing the code.

The result? A large, sometimes years-long gap between software releases, with frequent bug fixes and software patches deployed between each release.

With the establishment of the Agile software methodology, IT industries moved on to developing software iteratively and frequently released them into production. Continuous Integration (CI) and Continuous Delivery (CD) are among the major techniques adapted in this model for the rapid delivery of software. (We'll look at these concepts shortly.)

DevOps consequently promoted the smooth collaboration between development and operations teams at each step of the cycle. So, we can safely say that DevOps has its roots in the Agile methodology.

(Learn more about the history of DevOps.)



Agile	Waterfall
Iterative development in short sprints	Sequential development process in pre-defined phases
Flexible and adaptive methodology	The process is documented and follows the fixed structure and requirements agreed in the beginning of the process
Feedback-based approach: Sprints lead to short build updates that are evaluated on and guide the future direction of the development process.	Limited and delayed feedback: The software quality and requirements fulfilment isn't evaluated until the final phase of the development processes when testers and customer feedback is requested.
A provision for adaptability: Project development requirements and scope is expected to change over the course of the iterative development process.	The requirements and scope are definitive once agreed upon.
The SDLC phases overlap and begin early in the SDLC: planning, requirements, designing, developing, testing, and maintenance.	The SDLC phases are followed in order, with no overlap. Members of one functional group are not involved in another phase that doesn't belong to their job responsibilities.
Follows a mindset of collaboration and communication. The requirements, challenges, progress, and changes are discussed between all stakeholders on a continuous basis.	Follows a project-focused mindset with the aim of fully completing the SDLC process.
Responsibilities and hierarchical structure can be interchangeable between team members.	Fixed individual responsibilities, particularly in management positions.
All team members focused on end-to-end completion (achieved sequentially) for the projects.	Team members focused on their responsibilities only during their respective SDLC phases.
Suitable for short projects in high-risk situations.	Suitable for straightforward projects in predictable circumstances.
Limited dependencies as the focus is less on implementation specifics, and more toward the mindset.	Strict dependencies in technologies, processes, projects and people.

DevOps culture

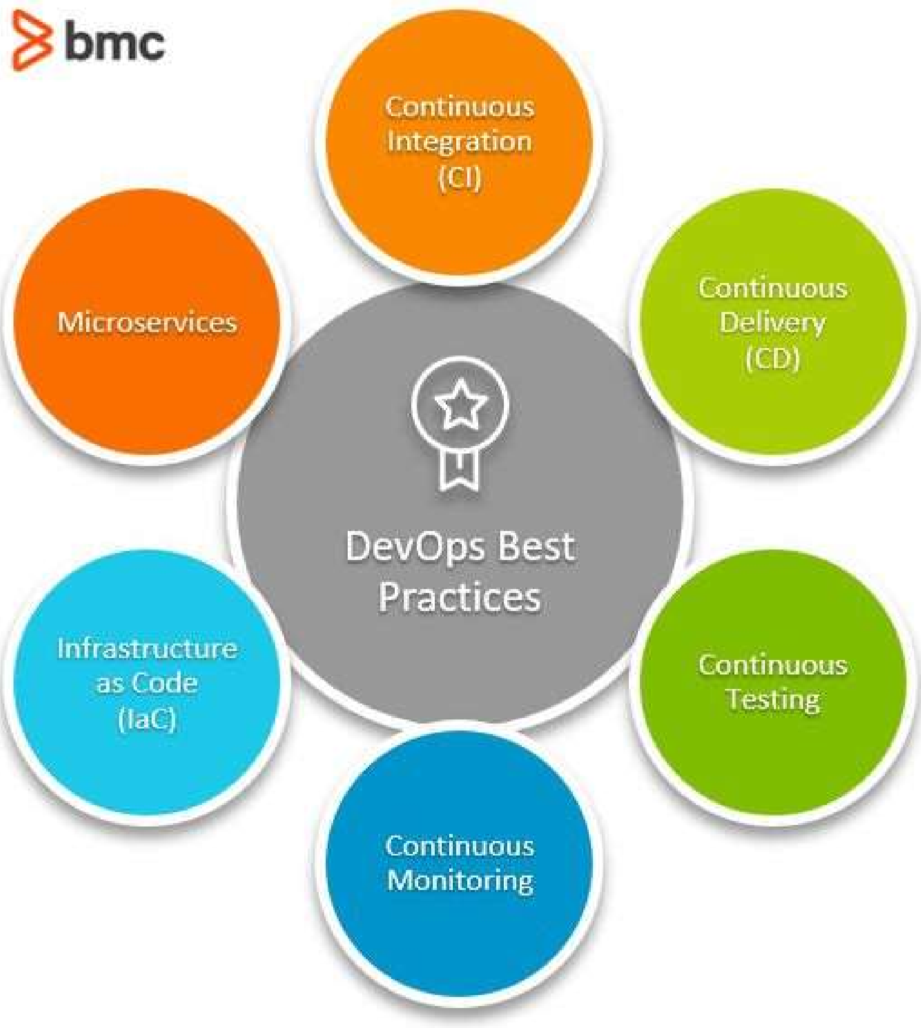
When transitioning to a DevOps culture, organizations must change two important components:

- Their traditional methods of operating
- Their mindset

Generally, in a DevOps culture, both Dev and Ops teams work collaboratively through frequent communication.

In some organizations, DevOps engineers may do both development and operations work. They do not limit their responsibilities to the defined scope in their job role. Rather, they always share the responsibilities with other teams, considering the entire development lifecycle as a part of their responsibilities.

These engineers aim to increase the productivity and the quality of service to deliver the maximum value to their customers.



DevOps practices

Of course, DevOps isn't only about communication and collaboration. DevOps best practices are aimed to help you release software frequently and with the best quality. Prioritizing high efficiency, DevOps encourages the widespread use of tools to automate manual tasks.

Let's take a look at common practices and related tools of any DevOps culture.

Continuous Integration

Traditionally, developers manually updated their code and then manually tested it.

Continuous Integration (CI) is a DevOps practice in which developers frequently merge their code changes to the central repository. Then, the code is built automatically and automated tests are run against the built code.

The main objective of this is to quickly identify bugs to improve the quality of the software.

Continuous Delivery

In Continuous Delivery (CD), merged code changes are automatically built and deployed to a testing environment. Then, automated tests are executed against the deployed code to identify any bugs and allow the developers to fix them in advance.

Typically, the code is deployed to different testing environments progressively, where the code achieves a higher level of quality through a standard automated testing procedure before deploying to the production. Continuous Delivery ensures that the team always has a development-ready code.

CI/CD tools such as Jenkins, Bamboo, Travis, TeamCity, and many others help automates these tasks.

(Explore CI/CD and whether continuous deployment fits.)

Continuous Testing

The Continuous Testing practice helps to identify potential risks as early as possible in all stages of the development lifecycle to minimize the impact on end-users.

For instance, when the code is deployed to build servers, automated unit tests will run to identify any bugs in the code. If the unit tests are failed, the build will be rejected and the feedback is sent to the developer to revise the code. Thus, the code will be deployed to the QA environment for functional testing only if the build passes the unit tests.

Selenium, Travis, and Appium are some popular continuous testing tools in the IT industry.

Continuous Monitoring

The application, infrastructure, middleware, and networks will be monitored continuously for their performance, any defects, or security and compliance. To identify issues, most companies monitor metrics such as:

- CPU and memory usage
- Disk space
- Customer activities
- Security policies

By putting continuous monitoring into practice, you'll always be alerted about any issues in environments from testing to production, helping you ensure high availability.

Popular Continuous Monitoring tools include Nagios, Sensu, Prometheus, and many more.

Infrastructure as Code

Infrastructure as Code (IaC) is a practice where infrastructure—virtual machines, load balancers, networks, etc.—are configured and managed using code rather than manually setting up and managing those resources. This has become an essential DevOps practice in organizations who specifically have moved to cloud platforms.

For example, Amazon Web services (AWS) provides APIs to programmatically interact with their cloud infrastructure. The use of code to define configuration helps to standardize the process allowing them to quickly deploy resources to the cloud.

Microservices

Unlike in traditional monolithic architectures, in microservices architecture, a single application is built as a set of small services or components. These separate services have their own functionality and their communication happens through a lightweight interface or an API.

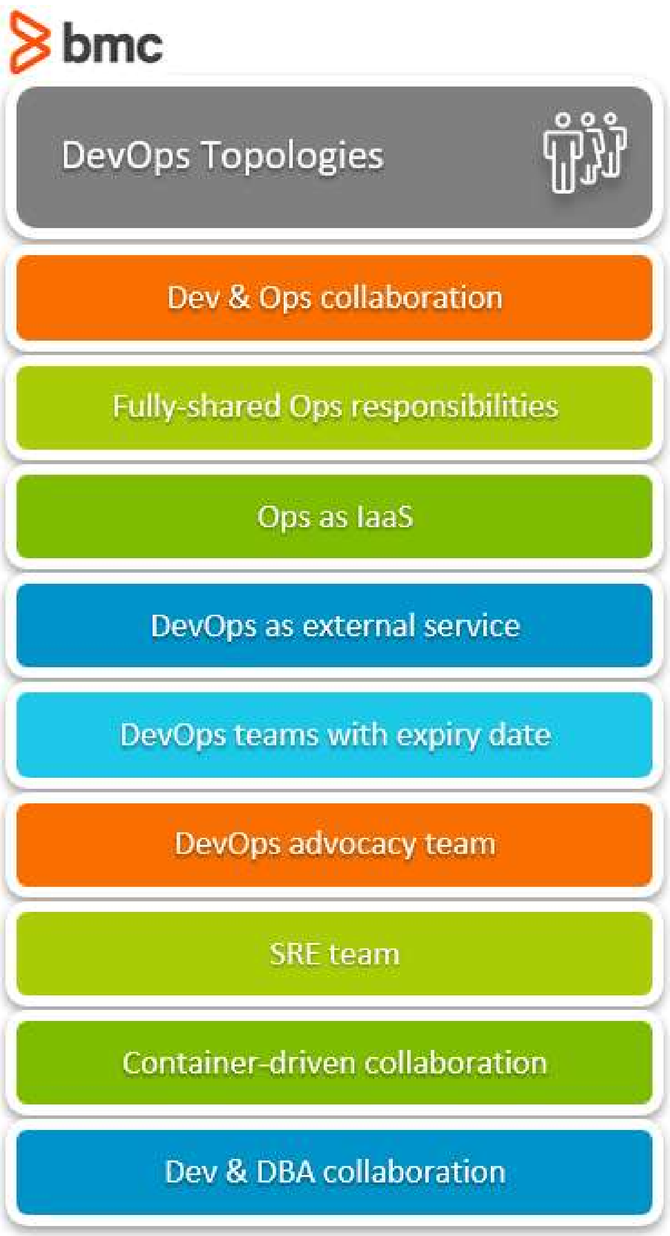
As a widely adapted architecture in DevOps culture, microservices:

- Promote independent resource management within different components.
- Enhance the system availability by preventing a single point of failure, as there is no impact to other components when one component is crashed.
- Allow DevOps teams to add additional components with different functionalities without impacting other components.

(Understand microservice best practices.)

DevOps Topologies

The way DevOps are practiced highly depends on the Organization. According to Matthew Skelton and Manual Pais, Organizations adopt different types of topologies or team structures for DevOps to flourish. They define 9 types of topologies as follows.


bmc
DevOps Topologies
Dev & Ops collaboration
Fully-shared Ops responsibilities
Ops as IaaS
DevOps as external service
DevOps teams with expiry date
DevOps advocacy team
SRE team
Container-driven collaboration
Dev & DBA collaboration

Dev & Ops collaboration

This is the ideal DevOps team structure where Dev and Ops teams smoothly collaborate. For example:

Developers take seriously the tasks from the operations teams, receiving their input when required.

Operations teams get a good understanding of the tasks that dev teams perform routinely.

Fully-shared Ops responsibilities

Implemented at Facebook and Netflix, this topology consists of Dev teams with Operations teams integrated into them. You will see only a small separation between Devs and Ops.

This is most suitable for companies that have single web-based applications.

Ops as infrastructure-as-a-service

This topology is suitable for organizations with different/multiple services which are hosted in the cloud platforms, but still have a traditional IT Operations department that is unlikely to change. Their Ops team is similar to infrastructure as a service (IaaS).

DevOps as external service

Some organizations may not have the expertise or can't afford to have a separate ops team. Here, the dev teams can get support from a service provider who can manage the responsibilities of the software's operational aspects.

DevOps teams with an expiry date

This structure functions as a temporary team that works towards bringing Dev and Ops teams together. That temporary team will become obsolete after achieving the pre-defined goal.

DevOps advocacy team

This topology is most suitable for teams that tend to drift apart. The advocacy team facilitates the dev and ops groups, keeping them engaged and continually making them aware of DevOps practices.

SRE team

Adapted in Google, this topology consists of a Site Reliability Engineering (SRE) Team. Devs need to provide evidence that their software is up to standards—and the SRE team can reject it if proven otherwise.

(Compare SRE & Ops teams.)

Container-driven collaboration

Containerization is the practice of encapsulating an application deployment and run-time requirements into a container. This removes some collaboration dependencies between Dev and Ops teams.

This structure is suitable for an organization that has a robust engineering culture.

(Learn more about managing code and containers.)

Dev & DBA collaboration

Some organizations that have multiple applications connected with large central databases have experimented with this topology. The structure is that database (DBA) specialization roles in both DBA and Dev teams are integrated, helping to bridge the gap between both parties.

Key benefits of DevOps

Organizations that embrace the DevOps culture see improvement in a lot of aspects of their software delivery process, including:

- Improving communication and collaboration between teams
- Enhancing organization efficiency and productivity
- Increasing the velocity of software releases
- Maximizing customer satisfaction (your customers now receive high-quality software)
- Ensuring system availability and reliability thanks to faster and earlier threat and incident detection
- Promoting innovation, thanks to sharing different ideas between teams

In this video, David Rizzo, VP Product Engineering at BMC Software, and David Kennedy, Solutions Architect, share how Compuware evolved to DevOps to increase innovation, reduce escaped defects, and improve MTTR.

11997
total

6882
req. development

5115
backlog waste

How Do We Use It?

How productive is the team; what is our throughput?

Confidential

siness

As I said earlier, DevOps is elopment and operations ity software that satisfies the customer requirements.

DevOps achieves its excellence by integrating defined best practices into each step of the software development life cycle. Organizations are required to identify the team topology that best suit them in DevOps to strive for in the long term.

When embraced correctly, DevOps culture can bring enormous benefits for organizations for successful software releases.