Devops LIfeCycle

What is devops?

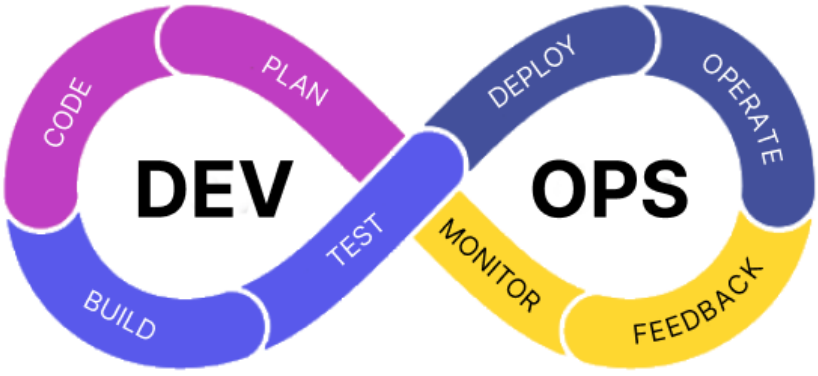
The DevOps is a combination of two words, one is software Development, and second is Operations. This allows a single team to handle the entire application lifecycle, from development to testing, deployment, and operations. DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.

DevOps promotes collaboration between Development and Operations team to deploy code to production faster in an automated & repeatable way.

What Is the DevOps Lifecycle?

The DevOps lifecycle is the continuous and iterative set of practices and processes that facilitate the integration and collaboration between software development (Dev) and IT operations (Ops) teams. It aims to streamline the development and delivery process, ensuring faster and more reliable releases while still maintaining a high level of quality.

Below is the DevOps Lifecycle diagram — a closed and continuous infinity cycle, representing the iterative flow of stages involved.



Stages of the DevOps Lifecycle

1. Plan

Exactly as its name suggests, this stage involves planning the entire project lifecycle. During this stage, the three essential stakeholders hold meetings to discuss the project requirements (collected from customers, internal/external experts/managers) and perform necessary analysis to come up with a strategic approach to building the application.

During this stage, details on resource allocation, project timeline, team structures, tech stack, tools, and risk mitigation approaches are also decided.

2. Code

Before the coding stage, there can be a preliminary Design stage where software engineers analyze and identify the best solutions to create the software. For example, they can decide on an appropriate architectural pattern that best fits the project's needs, then break down the system architecture into smaller, more manageable components.

At this point developers start to work on the code. Version control systems, like Git, are used to manage code changes and enable collaboration among team members.

3. Build

With the new code ready, it is now time for developers to integrate code changes into a shared repository, ideally multiple times a day. Continuous integration (CI) plays a crucial role here, which ensures that any integration issues are detected and resolved early on, so that the codebase is always in a consistent and working state.

Once the developers commit code and the lead developer approves, the CI process is triggered, initiating the build process. The CI server will compile the source code into low-level machine code that can be executed. CI tools also identify the dependencies, such as external libraries, frameworks, and modules required for the code to run properly. The output of this build process is a “build artifact,” an executable and deployable form ready for testing and distribution to end users.

4. Test

This is a critical phase to ensure the software’s quality and reliability. A wide variety of tests are performed to validate all aspects of the build, including functionality, performance, aesthetics, security, and more.

Automated Testing is key to DevOps as it enables faster and more reliable verification of code changes, reduces manual effort, and facilitates continuous integration (CI) and continuous delivery (CD). Testers need to perform automated testing at all levels, including unit, integration, and system testing.

- Unit testing: to verify the functionality of individual units of code
- Integration testing: to verify the flow of data between different code units or components, ensuring that they **communicate effectively with each other**
- System testing: to test user interaction in an environment as closely related to the production environment as possible

5. Deploy

During this phase, the code that has been thoroughly tested is released and deployed to different environments, including staging, UAT, and production. The software/application is now available for users to access and interact with, but the level of access differs for each:

- The **staging environment** (also known as a preproduction environment) is a near exact replica of a production environment for software testing. These environments bridge the development environment with the production environment, providing a preview of how the software works and ensuring it aligns with stakeholders' expectations.
- The **UAT environment** is where the code is deployed after software testing is completed on staging. It is also a near exact replica of the production environment, but customers or product owners perform testing here instead of the QA team.
- The **production environment** is where the final deployment is done. Also called the "live" environment, all customers can interact with it in real time. During this stage, bugs are not expected to be found. If a bug is found, the QA will immediately be alerted.

Automated deployments are extremely valuable to create a seamless flow, simplifying the deployment process and reducing human errors. It is also the key to achieve continuous deployment, where software changes are automatically pushed to production environments as soon as they pass automated tests.

6. Operate

Once the software is deployed, it enters the operational phase, where it is fully available for use by end users. A dedicated operations team will take over the management of the software in the production environment, performing many tasks to keep it running smoothly, ensure its availability, and make optimization decisions to bring better experience to the users.

Ideally this team utilizes monitoring tools to collect and analyze real-time data from all components of the software stack, including servers, databases, networks, or application metrics to be constantly updated on its health. If any deviant behavior is detected based on a preconfigured threshold, these tools can send alerts to the team so that they can be proactive in resolution.

This stage also involves capacity management, ensuring that high volumes of users can access the software while not eating up too much of the available resources. Based on historical usage data, the Operations team can predict future resource requirements to handle unexpected demand.

7. Feedback

Finally, the Ops team creates a feedback loop by sending the operational data collected to the Dev team, helping them identify areas for improvement, track the KPIs, and assess if any changes to the software are needed. The DevOps lifecycle begins again, and each of these iterations leads to an enhanced version of the software.

7Cs of the DevOps Lifecycle

You can also view the DevOps lifecycle as a holistic "7C" phase approach:

1. Continuous development
2. Continuous integration
3. Continuous testing
4. Continuous deployment/Continuous delivery
5. Continuous feedback
6. Continuous monitoring
7. Continuous operations

1. Continuous Development

Continuous development fosters a culture of collaboration among developers, testers, and other stakeholders. Instead of incorporating changes in one large batch, developers make small, piece-by-piece changes to the software, such as bug fixes, new features, code improvements, and code refactoring. Development becomes a series of small iterations that gradually add value to the software.

The idea behind continuous development is to make the effort more manageable, minimizing the risks of large-scale failures and allowing organizations to be more responsive to user needs and constantly changing market demands.

For continuous development, dev teams usually leverage Git — the most popular version control system for collaborative code development and easy code branching and merging — while GitHub, GitLab, and Bitbucket are popular web-based platforms for hosting and managing Git repositories.

2. Continuous Integration

After the code has been written, it is committed to the shared repository, where a series of events to seamlessly integrate the code is put into motion. This is also known as the CI pipeline:

- Detect changes in the source code repository (new commits appear)
- Source code quality analysis
- Build
- Execute all unit tests
- Execute all integration tests
- Generate deployable artifacts
- Report status

If one of the steps above unfortunately fails:

- Integration may stop or continue depending on defect severity and configuration
- Results are notified to the team via email or chat system
- The team fixes defects and commits again
- Tasks are performed again

For CI, Jenkins is a widely used open-source CI/CD automation server. It allows developers to automate various aspects of the software development process, including building, testing, and deploying code changes.

3. Continuous Testing

Continuous testing involves the automation of testing activities to ensure that software changes are validated at each step of the pipeline, providing fast and reliable feedback to the development team, without the need for human intervention. The time saved thanks to continuous testing can be better spent on more strategic and planning tasks.

4. Continuous Deployment/Continuous Delivery

Continuous deployment and continuous delivery are two closely related practices in the DevOps world, although they still have distinct differences.

Continuous delivery ensures that software changes are automatically and consistently delivered to production or staging environments after passing through an automated build, test, and validation process. The final decision to deploy the software to production lies with human operators or stakeholders. They can choose when to trigger the deployment.

Continuous deployment takes continuous delivery one step further by fully automating the deployment process to production without any human intervention.

5. Continuous Feedback

In this stage, user feedback is continuously collected to provide insights into how to optimize the codebase. You can set up automated mechanisms to collect feedback, such as automated user feedback surveys, automated error reporting, and automated monitoring systems that collect performance and usage metrics.

Several channels to provide feedback include:

- Feedback forms that pop up in the application when a certain series of events happen
- Support tickets
- Emails
- Community forums

6. Continuous Monitoring

It should be noted that monitoring activities have always been fairly continuous. Monitoring is all about gaining real-time visibility into the performance and behavior of a system, providing continuous and up-to-date insights into various metrics. Continuous monitoring takes this one step further by placing emphasis on three key factors:

- Real-time data collection and analysis
- Proactively and constantly respond to risks
- Ongoing data analysis

In many ways, the idea and goals of continuous monitoring are similar to continuous delivery, which is to help organizations move faster with accuracy and efficiency.

7. Continuous Operations

Continuous operations focuses on the ongoing management and maintenance of software systems in production environments, ensuring that the system remains stable, secure, performant, and available to users throughout its lifecycle. Continuous operations include the following key aspects:

- **Indirect response and management:** detecting and responding to incidents and outages in the production environment. When issues occur, the operations team works to identify the root cause and takes corrective actions to restore normal operations.
- **Change management:** any updates or changes to the software or infrastructure are carefully planned, tested, and rolled out to minimize disruption and risk.

- **Monitoring and alerting:** monitoring tools collect real-time data and metrics to provide insights into the system's behavior. Alerting mechanisms notify the operations team when predefined thresholds or conditions are breached.
- **Backup and recovery:** regular backups are taken to protect data, and disaster recovery plans are prepared to minimize downtime in case of major failures.