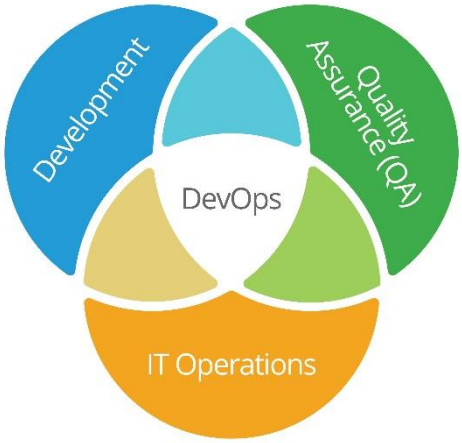
DevOps Methodoly and Tools

What is DevOps?

DevOps is a software development methodology that combines software development (**Dev**) with information technology operations (**Ops**) participating together in the entire service lifecycle, from design through the development process to production support.

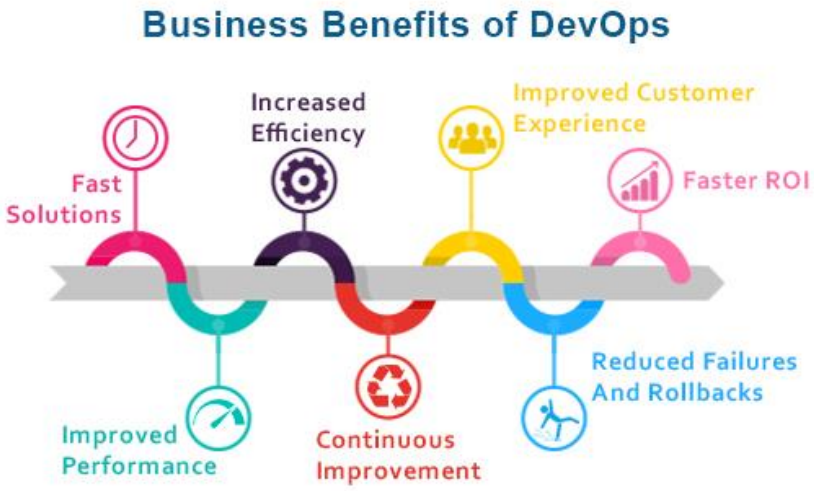


The goals of DevOps

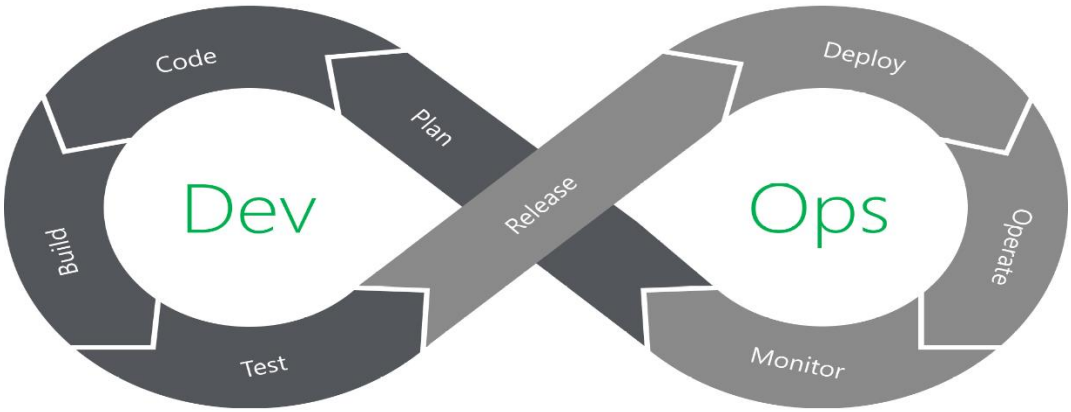
- Fast Development Methodologies
- Fast Quality Assurance Methodologies
- Fast Deployment Methodologies
- Faster time to market
- Iteration & Continuous Feedback (strong and continuous communication between stakeholders — the end users and customers, product owners, development, quality assurance, and production engineers)

The benefits of DevOps

- ***Environment Stabilization***
  - *Enforces consistency, increase up-time*
- ***Shorter Development Cycle***
  - *Manage requirements and code-repository*
- ***Increased Release Velocity***
  - *Continuous build, push-button deployments*
- ***Reduced Defects***
  - *Regiment processes, automated testing*
- ***Process Metrics***
  - *Track both time at each stage, and the errors and exceptions*



Phases of a DevOps



Plan

The Plan stage covers everything that happens before the developers start writing code, and it's where a Product Manager or Project Manager earns their keep. Requirements and feedback are gathered from stakeholders and customers and used to build a product roadmap to guide future development. The product roadmap can be recorded and tracked using a ticket management system such as Jira, Azure DevOps or Asana which provide a variety of tools that help track project progress, issues and milestones.

The product roadmap can be broken down into Epics, Features and User Stories, creating a backlog of tasks that lead directly to the customers' requirements. The tasks on the backlog can then be used to plan sprints and allocate tasks to the team to begin development.

Code

Once the team had grabbed their coffees and had the morning stand-up, the developments can get to work. In addition to the standard toolkit of a software developer, the team has a standard set of plugins installed in their development environments to aid the development process, help enforce consistent code-styling and avoid common security flaws and code anti-patterns.

This helps to teach developers good coding practice while aiding collaboration by providing some consistency to the codebase. These tools also help resolve issues that may fail tests later in the pipeline, resulting in fewer failed builds.

Build

The Build phase is where DevOps really kicks in. Once a developer has finished a task, they commit their code to a shared code repository. There are many ways this can be done, but typically the developer submits a pull request — a request to merge their new code with the shared codebase. Another developer then reviews the changes they've made, and once they're happy there are no issues, they approve the pull-request. This manual review is supposed to be quick and lightweight, but it's effective at identifying issues early.

Simultaneously, the pull request triggers an automated process which builds the codebase and runs a series of end-to-end, integration and unit tests to identify any regressions. If the build fails, or any of the tests fail, the pull-

request fails and the developer is notified to resolve the issue. By continuously checking code changes into a shared repository and running builds and tests, we can minimise integration issues that arise when working on a shared codebase, and highlight breaking bugs early in the development lifecycle.

Test

Once a build succeeds, it is automatically deployed to a staging environment for deeper, out-of-band testing. The staging environment may be an existing hosting service, or it could be a new environment provisioned as part of the deployment process. This practice of automatically provisioning a new environment at the time of deployment is referred to as Infrastructure-as-Code (IaC) and is a core part of many DevOps pipelines. More on that in a later article.

Once the application is deployed to the test environment, a series of manual and automated tests are performed. Manual testing can be traditional User Acceptance Testing (UAT) where people use the application as the customer would to highlight any issues or refinements that should be addressed before deploying into production.

At the same time, automated tests might run security scanning against the application, check for changes to the infrastructure and compliance with hardening best-practices, test the performance of the application or run load testing. The testing that is performed during this phase is up to the organisation and what is relevant to the application, but this stage can be

considered a test-bed that lets you plug in new testing without interrupting the flow of developers or impacting the production environment.

Release

The Release phase is a milestone in a DevOps pipeline — it's the point at which we say a build is ready for deployment into the production environment. By this stage, each code change has passed a series of manual and automated tests, and the operations team can be confident that breaking issues and regressions are unlikely.

Depending on the DevOps maturity of an organisation, they may choose to automatically deploy any build that makes it to this stage of the pipeline. Developers can use feature flags to turn off new features so they can't be seen by the customers until they are ready for action. This model is considered the nirvana of DevOps and is how organisations manage to deploy multiple releases of their products every day.

Alternatively, an organisation may want to have control over when builds are released to production. They may want to have a regular release schedule or only release new features once a milestone is met. You can add a manual approval process at the release stage which only allows certain people within an organisation to authorise a release into production.

The tooling lets you customise this, it's up to you how you want to go about things.

Deploy

Finally, a build is ready for the big time and it is released into production. There are several tools and processes that can automate the release process to make releases reliable with no outage window.

The same Infrastructure-as-Code that built the test environment can be configured to build the production environment. We already know that the test environment was built successfully, so we can rest assured that the production release will go off without a hitch.

A blue-green deployment lets us switch to the new production environment with no outage. Then the new environment is built, it sits alongside the existing production environment. When the new environment is ready, the hosting service points all new requests to the new environment. If at any point, an issue is found with the new build, you can simply tell the hosting service to point requests back to the old environment while you come up with a fix.

Operate

The new release is now live and being used by the customers. Great work!

The operations team is now hard at work, making sure that everything is running smoothly. Based on the configuration of the hosting service, the environment automatically scales with load to handle peaks and troughs in the number of active users.

The organisation has also built a way for their customers to provide feedback on their service, as well as tooling that helps collect and triage this feedback to help shape the future development of the product. This feedback loop is important — nobody knows what they want more than the customer, and the customer is the world's best testing team, donating many more hours to testing the application than the DevOps pipeline ever could. You need to capture this information, it's worth its weight in gold.

Monitor

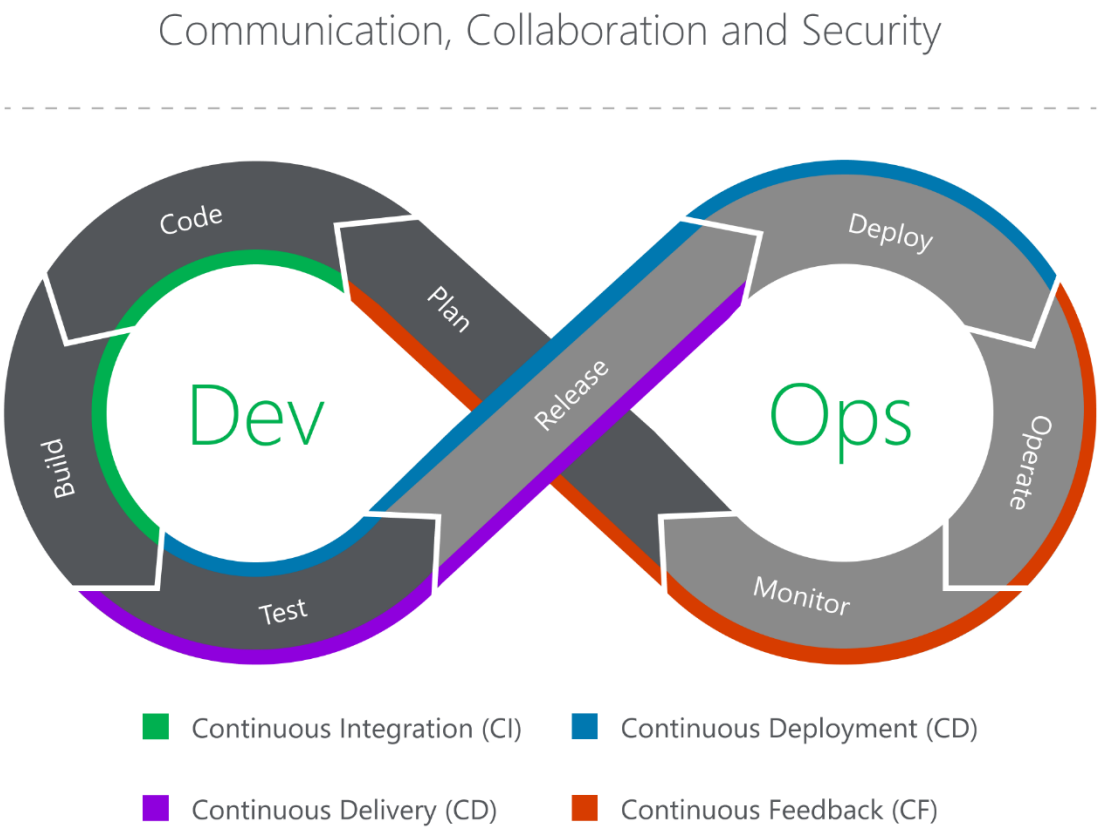
The 'final' phase of the DevOps cycle is to monitor the environment. this builds on the customer feedback provided in the Operate phase by collecting data and providing analytics on customer behaviour, performance, errors and more.

We can also do some introspection and monitor the DevOps pipeline itself, monitoring for potential bottlenecks in the pipeline which are causing frustration or impacting the productivity of the development and operations teams.

All of this information is then fed back to the Product Manager and the development team to close the loop on the process. It would be easy to say this is where the loop starts again, but the reality is that this process is continuous. There is no start or end, just the continuous evolution of a product throughout its lifespan, which only ends when people move on or don't need it any more.

Continuous Everything

Alongside the phases of the DevOps pipeline, you'll commonly hear people talking about Continuous Everything — Continuous Integration, Continuous Delivery, Continuous Deployment and more. This is because continuity is at the core of DevOps, and we tech people love our terminology and buzzwords. However, they do serve a purpose. Let's break down each of these terms and how they relate to the phases of the pipeline.



Continuous Integration

One of the biggest difficulties in coordinating a software development team is managing the collaboration of many developers, often in remote locations, on a single codebase. A shared code repository is key to solving this problem, however, there can still be issues in when merging the changes made by multiple people on the same piece of code.

A change made by one developer may impact what somebody else is working on, and the longer that developers wait to integrate their changes back into the shared codebase, the bigger the drift, resulting in more effort and headache in resolving the issues and conflicts.

Continuous integration aligns with the Code and Build phases of the DevOps pipeline. It's the practice of regularly merging a developer's code into the centralised codebase and conducting automated testing to ensure that no regressions have been introduced. By merging smaller changes more regularly, these issues become smaller and easier to manage, improving overall productivity and sanity.

Continuous Delivery

Continuous Delivery is an extension of Continuous Integration which automates the process of deploying a new build into production. The goals of Continuous Delivery is to:

1. Perform automated testing on each new build to verify builds that are ready for release into production, and fail those which are not.
2. Manage the automatic provisioning and configuration of deployment environments, as well as testing of these environments for stability, performance and security compliance.
3. Deploy a new release into production when approved and manually triggered by the organisation.

Continuous Delivery aligns with the Test and Release phases of the pipeline, and allows organisations to manually trigger the release of new builds as regularly as they choose.

Continuous Deployment

Continuous Deployment is a more advanced version of Continous Delivery (which makes the reuse of the 'CD' abbreviation more acceptable). The goals are the same, but the manual step of approving new releases into production is removed. In a Continuous Deployment model, each build which passes all of the checks and balances of the pipeline are automatically deployed into production.

Continuous Feedback

CI and CD tend to get the glory when people talk about DevOps, but an equally important factor is Continuous Feedback. The whole point of DevOps is to release new features and fixes as quickly as possible so that the organisation can get feedback from customers, stakeholders and analytics as quickly as possible to make better decisions when designing the next set of

changes. The whole point is to achieve a strong Continous Feedback loop to develop a better product.

It's Continuous Feedback that ties the ends of the loop together, feeding back data and analytics from the Operate and Monitor phases back into the Plan phase to do it all over again.

DevOps Tools

This consist of testing and building of systems, deployment of application, configuration management, tools monitoring and version control. (CI) Continuous integration, (CD) continuous delivery and (CT) continuous testing demands for different tools that have been mentioned here under.

Now, when you have had a brief knowledge on what DevOps is and its values, let's know about the top DevOps tools closely that can be employed in 2020 for improved results.

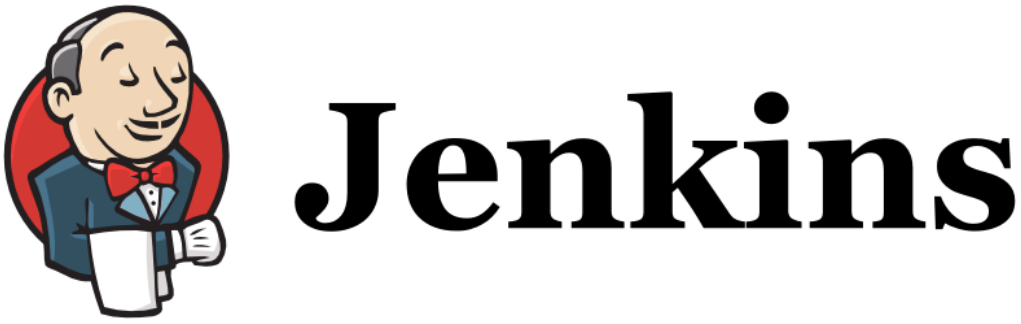
Git

(Netflix , Shopify , Udemy , Reddit , ...)

Most of the tech companies are using this DevOps tool because it is a distributed SCM – source code management tool without which it is not possible for the open source contributors and distant DevOps teams to work. Every project requirement is fulfilled by Git perfectly. Through this tool the developers can have a track on the progress of their development project. Using Git project code's different variations can be saved aptly and when required, the developer's team can hop to the previous version.

Experiments can also be made using Git, as it permits the developers to build different project branches and then merge all of them when they are ready. It can handle efficiently and with speed every type of project may it be large or small.

Jenkins



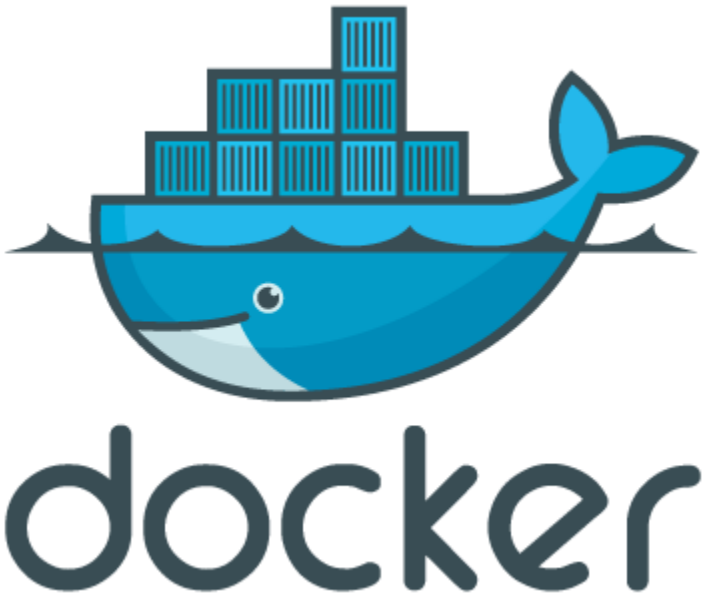
(eBay , Dell , HootSuite , Netflix, …)

Jenkins is a self-contained open-source automation server having several plugins (almost more than thousand) used by maximum developers present

worldwide. Offering around 1000 and more plugins, this tool can be integrated with almost every DevOps tool from Puppet to Docker and the developers can extend the product's functionality. Being a CI/CD server, it permits the software development team to automate different delivery pipeline stages. The developers can plugin as well as customize their CI/CD pipeline as per their terms through this DevOps tool.

Tasks like building, testing and deployment of software can be automated using Jenkins. With it, development team can iterate and install software code speedily that leads to faster delivery of product. Available for Windows, Mac and Linux one can rapidly get started with Jenkins.

Docker



(BBC News , Lyft , ADP , Yelp , ...)

Docker, a container platform, is the best DevOps tool from the day it has been launched i.e. from 2013 and is soaring continuously. Docker makes possible distributed development and automates app's deployment. It also isolates apps into different containers because of which they become more secure and portable. Docker applications are platform independent and also operating system. Docker containers can be used in place of virtual machines like VirtualBox.

This tool integrates with Bamboo and Jenkins too. When combined with any of these two automation servers, the developers can improve their delivery workflow. Moreover, Docker can be used through the cloud.

Puppet

(Accenture , Cisco , AWS , Cognizant , Google , Microsoft , ...)

Puppet is popularly utilized for configuration management. Being an open-source platform, it is having a declarative language that describes the configuration of its system. You can run it on numerous systems as well as on Unix-based systems, macOS Servers, IBM mainframe, Microsift Windows and Cisco switches.

Puppet is used primarily for pulling strings on several application servers immediately. It manages infrastructure just like a code and renders support for DevOps practices such as continuous integration, versioning, continuous delivery and automated testing. With Puppet, application changes can be deployed with confidence as well as developers can solve the errors more quickly.

Nagios

(Uber , Twitch , Dropbox , Fiverr , ...)

Nagios is a great monitoring system that monitors networks, systems, and infrastructure. It enables you to spot and resolve issues related to your IT infrastructure. Nagios keeps an eye on your complete infrastructure for ensuring everything is operating and running perfectly. When it detects any failure, your technical staff is sent an alert instantly and permit them to start recovery process before those issues become massive and affect the business processes critically.