Name :· Subodh Ghankute Exp No: 1

Roll no:- 26

div:- TE-A(IT)

subject :- DevOps

Aim :- To understand DevOps: Principles, Practices, and DevOps Engineer Role and Responsibilities

DevOps principles

Culture

DevOps is initially the culture and mindset forging strong collaborative bonds between software development and infrastructure operations teams. This culture is built upon the following pillars.

Constant collaboration and communication. These have been the building blocks of DevOps since its dawn. Your team should work cohesively with the understanding of the needs and expectations of all members.

Gradual changes. The implementation of gradual rollouts allows delivery teams to release a product to users while having an opportunity to make updates and roll back if something goes wrong.

Shared end-to-end responsibility. When every member of a team moves towards one goal and is equally responsible for a project from beginning to end, they work cohesively and look for ways of facilitating other members' tasks

Early problem-solving. DevOps requires that tasks be performed as early in the project lifecycle as possible. So, in case of any issues, they will be addressed more quickly.

Automation of processes

Automating as many development, testing, configuration, and deployment procedures as possible is the golden rule of DevOps. It allows specialists to get rid of time-consuming repetitive work and focus on other important activities that can't be automated by their nature.

Measurement of KPIs (Key Performance Indicators)

Decision-making should be powered by factual information in the first place. To get optimal performance, it is necessary to keep track of the progress of activities

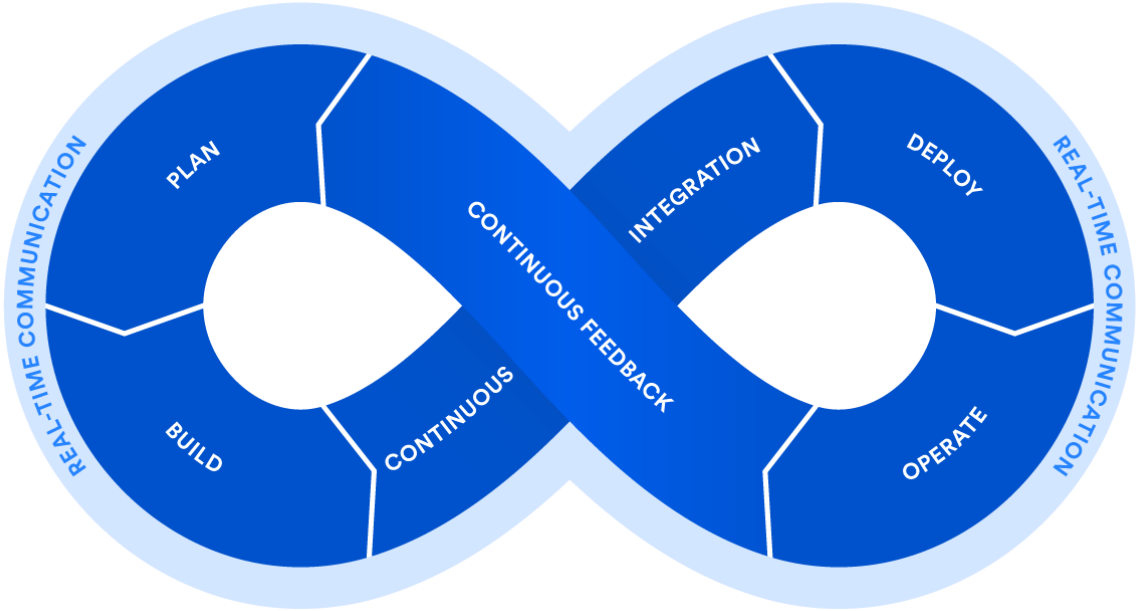
composing the DevOps flow. Measuring various metrics of a system allows for understanding what works well and what can be improved.

Sharing

Sharing is caring. This phrase explains the DevOps philosophy better than anything else as it highlights the importance of collaboration. It is crucial to share feedback, best practices, and knowledge among teams since this promotes transparency, creates collective intelligence and eliminates constraints. You don't want to put the whole development process on pause just because the only person who knows how to handle certain tasks went on a vacation or quitted.

DevOps practices

DevOps requires a delivery cycle that comprises planning, development, testing, deployment, release, and monitoring with active cooperation between different members of a team.



To break down the process even more, let's have a look at the core practices that constitute the DevOps:

Agile planning

In contrast to traditional approaches of project management, Agile planning organizes work in short iterations (e.g. sprints) to increase the number of releases. This means that the team has only high-level objectives outlined, while making detailed planning for two iterations in advance. This allows for flexibility and

pivots once the ideas are tested on an early product increment. Check our Agile infographics to learn more about different methods applied.

Continuous development

The concept of continuous “everything” embraces continuous or iterative software development, meaning that all the development work is divided into small portions for better and faster production. Engineers commit code in small chunks multiple times a day for it to be easily tested.

Continuous automated testing

A quality assurance team sets committed code testing using automation tools like Selenium, Ranorex, UFT, etc. If bugs and vulnerabilities are revealed, they are sent back to the engineering team. This stage also entails version control to detect integration problems in advance. A Version Control System (VCS) allows developers to record changes in the files and share them with other members of the team, regardless of their location.

Continuous integration and continuous delivery (CI/CD)

The code that passes automated tests is integrated in a single, shared repository on a server. Frequent code submissions prevent a so-called “integration hell” when the differences between individual code branches and the mainline code become so drastic over time that integration takes more than actual coding.

Continuous delivery, detailed in our dedicated article, is an approach that merges development, testing, and deployment operations into a streamlined process as it heavily relies on automation. This stage enables the automatic delivery of code updates into a production environment.

Continuous deployment

At this stage, the code is deployed to run in production on a public server. Code must be deployed in a way that doesn’t affect already functioning features and can be available for a large number of users. Frequent deployment allows for a “fail fast” approach, meaning that the new features are tested and verified early. There are various automated tools that help engineers deploy a product increment. The most popular are Chef, Puppet, Azure Resource Manager, and Google Cloud Deployment Manager.

Continuous monitoring

The final stage of the DevOps lifecycle is oriented to the assessment of the whole cycle. The goal of monitoring is detecting the problematic areas of a process and

analyzing the feedback from the team and users to report existing inaccuracies and improve the product's functioning.

Infrastructure as a code

Infrastructure as a code (IaC) is an infrastructure management approach that makes continuous delivery and DevOps possible. It entails using scripts to automatically set the deployment environment (networks, virtual machines, etc.) to the needed configuration regardless of its initial state.

Without IaC, engineers would have to treat each target environment individually, which becomes a tedious task as you may have many different environments for development, testing, and production use.

Having the environment configured as code, you

1. Can test it the way you test the source code itself and
2. Use a virtual machine that behaves like a production environment to test early.

Once the need to scale arises, the script can automatically set the needed number of environments to be consistent with each other.

Containerization

Virtual machines emulate hardware behavior to share computing resources of a physical machine, which enables running multiple application environments or operating systems (Linux and Windows Server) on a single physical server or distributing an application across multiple physical machines.

Containers, on the other hand, are more lightweight and packaged with all runtime components (files, libraries, etc.) but they don't include whole operating systems, only the minimum required resources. Containers are used within DevOps to instantly deploy applications across various environments and are well combined with the IaC approach described above. A container can be tested as a unit before deployment. Currently, Docker provides the most popular container toolset.

Microservices

The microservice architectural approach entails building one application as a set of independent services that communicate with each other, but are configured individually. Building an application this way, you can isolate any arising problems ensuring that a failure in one service doesn't break the rest of the application functions. With the high rate of deployment, microservices allow for keeping the whole system stable, while fixing the problems in isolation. Learn

more about microservices and modernizing legacy monolithic architectures in our article.

Cloud infrastructure

Today most organizations use hybrid clouds, a combination of public and private ones. But the shift towards fully public clouds (i.e. managed by an external provider such as AWS or Microsoft Azure) continues. While cloud infrastructure isn't a must for DevOps adoption, it provides flexibility, toolsets, and scalability to applications. With the recent introduction of serverless architectures on clouds, DevOps-driven teams can dramatically reduce their effort by basically eliminating server-management operations.

An important part of these processes are automation tools that facilitate the workflow. Below we explain why and how it is done.

A DevOps Engineer: role and responsibilities

In the book Effective DevOps by Ryn Daniels and Jennifer Davis, the existence of a specific DevOps person is questioned: "It doesn't usually make much sense to have a director of DevOps or some other position that puts one person in charge of DevOps. DevOps is at its core a cultural movement, and its ideas and principles need to be used throughout entire organizations in order to be effective."

Some other DevOps experts partly disagree with this statement. They also believe that a team is a key to effectiveness. But in this interpretation, a team – including developers, a quality assurance leader, a code release manager, and an automation architect – work under the supervision of a DevOps engineer.

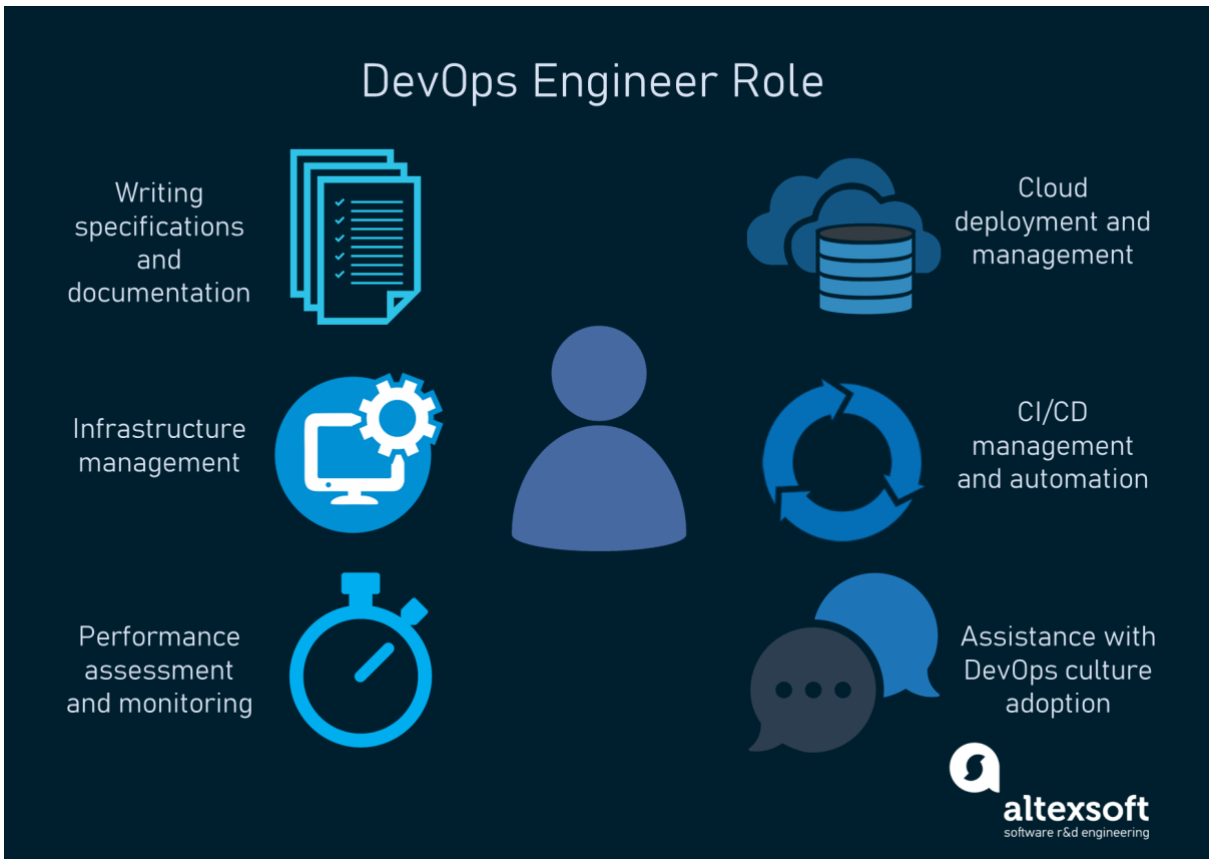
So, the title of a DevOps Engineer is an arguable one. Nonetheless, DevOps engineers are still in demand on the IT labor market. Some consider this person to be either a system administrator who knows how to code or a developer with a system administrator's skills.

DevOps engineer responsibilities

In a way, both definitions are fair. The main function of a DevOps engineer is to introduce the continuous delivery and continuous integration workflow, which requires the understanding of the mentioned tools and the knowledge of several programming languages.

Depending on the organization, job descriptions differ. Smaller businesses look for engineers with broader skillsets and responsibilities. For example, the job description may require product building along with the developers. Larger

companies may look for an engineer for a specific stage of the DevOps lifecycle that will work with a certain automation tool.

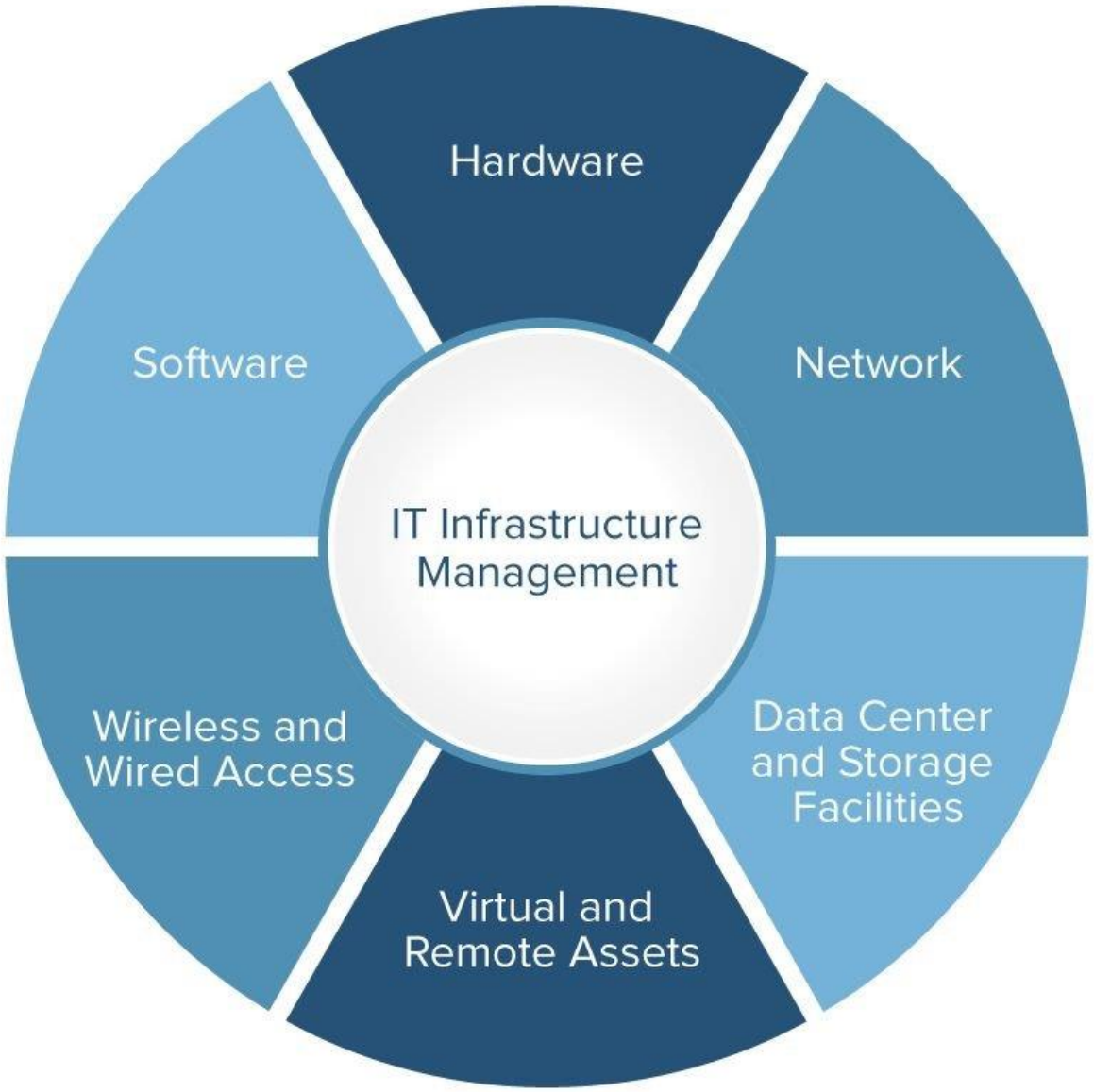


DevOps Engineer Role and Requirements

The basic and widely-accepted responsibilities of a DevOps engineer are:

- Writing specifications and documentation for the server-side features
- Continuous deployment and continuous integration (CI/CD) management
- Performance assessment and monitoring
- Infrastructure management
- Cloud deployment and management
- Assistance with DevOps culture adoption

Additionally, a DevOps engineer can be responsible for IT infrastructure maintenance and management, which comprises hardware, software, network, storages, virtual and remote assets, and control over cloud data storage.



This expert participates in IT infrastructure building, works with automation platforms, and collaborates with the developers, operation managers, and system administrators, facilitating processes they are responsible for.