PRACHITI PALANDE

121A8038

AIDS TE

SE EXP 1

**DevOps Principles**

In short, the main principles of DevOps are automation, continuous delivery, and fast reaction to feedback. You can find a more detailed explanation of DevOps pillars in the **CAMS**acronym:

**C**ulture represented by human communication, technical processes, and tools

**A**utomation of processes

**M**easurement of KPIs

**S**haring feedback, best practices, and knowledge

Adherence to these principles is achieved through a number of DevOps practices that include continuous delivery, frequent deployments, QA automation, validating ideas as early as possible, and in-team collaboration.

### 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](https://www.altexsoft.com/infographics/agile-project-management-methods/?utm_source=DZone&utm_medium=referral) to learn more about the different methods applied.

### Continuous Delivery and Automation

[Continuous delivery](https://www.altexsoft.com/blog/business/continuous-delivery-and-integration-rapid-updates-by-automating-quality-assurance/?utm_source=DZone&utm_medium=referral), 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.

**Development.**Engineers commit code in small chunks multiple times a day for it to be easily tested.

**Continuous automated testing and integration.**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 its location. 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. The most popular tools for continuous integration are Jenkins, GitLab CI, Bamboo, and TeamCity.

**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 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

The next evolutionary stage of virtual machines is 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](https://www.docker.com/) 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](https://www.altexsoft.com/blog/engineering/using-microservices-for-legacy-system-modernization/?utm_source=DZone&utm_medium=referral) in our article.

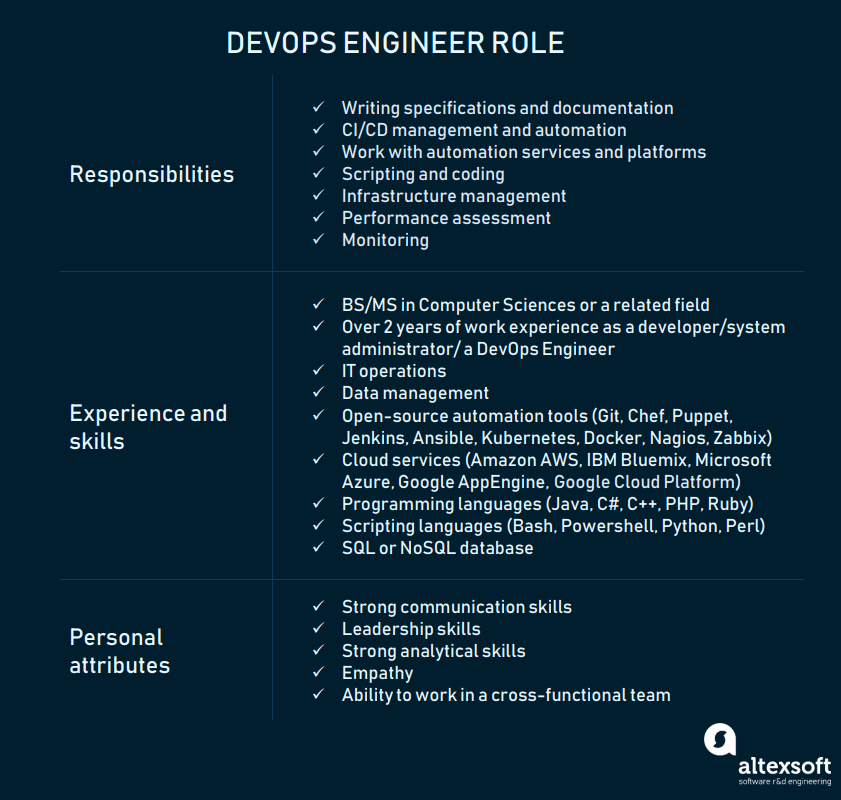
### Cloud Infrastructure

Today most organizations use [hybrid clouds](https://www.altexsoft.com/blog/cloud/preparing-for-hybrid-cloud/?utm_source=DZone&utm_medium=referral), 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](https://www.altexsoft.com/blog/cloud/pros-and-cons-of-serverless-architecture/?utm_source=DZone&utm_medium=referral), 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.

### 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
* Management of continuous deployment and continuous integration (CI/CD)
* CI/CD script writing
* Performance assessment and monitoring

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.