**Devops**

**What is Devops?**

DevOps, short for Development and Operations, is a set of practices and cultural philosophies that aim to improve and streamline the collaboration between software development (Dev) and IT operations (Ops) teams. DevOps is not just a set of tools or a specific job role; it's a cultural and organizational movement that promotes communication, collaboration, and automation across the software development lifecycle.

**(or)**

Devops is a process of improving the application delivery by ensuring there is proper automation in place, code quality maintain, continuous monitoring, continuous testing

**Why Devops?**

The word DevOps is a combination of two words Development and Operations. Before getting into what DevOps is, let us get an idea about the two teams involved in software development. The **development team** is responsible for developing, designing, and building the application. The **operation team** deals with the deployment and testing of the application. If there are problems with the application, the operation team also provides feedback to the development team. Now let us get to the history of DevOps.

**Software Development Life Cycle?**

The Software Development Life Cycle (SDLC) is a structured framework for planning,designing, developing, testing, and maintaining software applications. It provides a systematic and well-defined approach to software development, ensuring that projects are completed efficiently, with high quality, and in a predictable manner.

**Waterfall model and Agile model**

The Waterfall model and Agile model are two different software development methodologies that have been widely used in the IT industry. Each has its own approach to managing the software development process, and they cater to different project needs and organizational structures. Let's explore each model:

**1. Waterfall Model:**

**Sequential Process:**

The Waterfall model follows a linear and sequential approach to software development.

Each phase must be completed before moving on to the next.

**Phases:**

Requirements, Design, Implementation, Testing, Deployment, Maintenance.

**Advantages:**

Clear documentation at each stage.

Well-defined milestones.

Easy to manage and understand.

**Challenges:**

Lack of flexibility for changes after the project has started.

Limited customer involvement until the end.

The Waterfall model is often used in projects where requirements are well-defined, and changes are expected to be minimal throughout the development process.

**2. Agile Model:**

**Iterative and Incremental:**

The Agile model is iterative and incremental, allowing for flexibility and adaptation to changes.

Development is done in small, iterative cycles (sprints).

**Phases:**

Planning, Requirements Analysis, Design, Coding, Unit Testing, Acceptance Testing.

**Advantages:**

Customer involvement throughout the development process.

Quick response to changes and evolving requirements.

Early and regular delivery of working software.

Challenges:

Requires active and consistent customer involvement.

Initial setup and adaptability may be challenging for some teams.

The Agile model is particularly suitable for projects where requirements are expected to evolve or change, and customer collaboration is crucial.

**DevOps:**

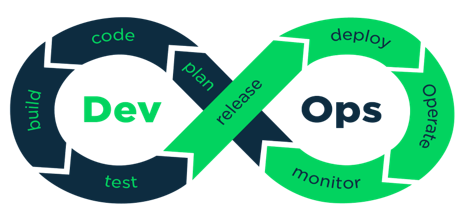
As we know about the problems faced in traditional models like in the waterfall model, there is a problem of a one-way stream of work, due to which if there is any mistake, the whole process repeats, and there is no interaction with customers. Now, this is solved in agile by splitting the whole development plan into several iterations for better production efficiency.  The agile model also includes customer interaction with the company to rectify mistakes. But there is another problem faced in Agile too.

Here, the problem arises when the development team continuously changes the code for better performance and sends the code to the operations team for testing. But there may be a delay in the operations team feedback in situations like if the developers sent code for review at night but due to the unavailability of the operations team, there will be a delay in the project feedback.

So, DevOps is the solution to this problem. DevOps is a practice or a methodology in which the development and operations teams work together by including automation at the initial stages. So they can work on rapidly changing systems, fix bugs, and help to deliver a good quality of software in time.

**DevOps Architecture:**

Let us now discuss different phases of DevOps architecture:



**Plan**– In DevOps planning plays an important role. In this stage, all the requirements of the project and everything regarding the project like time for each stage, cost. etc are discussed. This will help everyone in teams to get a brief idea about the project.

**Code** – In this Stage the code is written over here according to the client’s requirements. Here the code is divided into small codes called Units. This is done to get a clear picture of the code. For example, if the team is doing a project on an online -Ekart application then the login part is divided as one unit, after login the page which shows all the categories is divided as another unit, user profile as another unit, etc.

Some of the examples of the tools used are Git, JIRA

**Build** – In this stage Building of the units is done. Some of the examples of the tools used are maven, Gradle.

**Test** – Testing of all units is done in this stage. So we will get to know where exactly the code is having bugs and if there are mistakes found it is returned. Some of the examples of the tools used are Selenium, PYtest

**Integrate** – In this stage, all the units of the codes are integrated. That means in this step we will be creating a connection between the development team and the operation team to implement Continuous Integration and Continuous Deployment. An example of the tool used is Jenkins.

**Deploy** – In this stage, the code is deployed on the client’s environment. Some of the examples of the tools used are AWS, Docker.

**Operate** – Operations are performed on the code if required. Some of the examples of the tools used are Kubernetes, open shift.

**Monitor** – In this stage monitoring of the application is done over here in the client’s environment. Some of the examples of the tools used are Nagios, elastic stack.

**What are day to day activities for devops engineer?**

The day-to-day tasks of an AWS DevOps Engineer involve managing AWS infrastructure, deploying applications, automating processes, monitoring system performance, troubleshooting issues, implementing security measures, and collaborating with development and operations teams.

**Continuous Integration/Continuous Deployment (CI/CD):**

Setting up and maintaining CI/CD pipelines to automate the building, testing, and deployment of software.

Monitoring and managing the pipeline for efficiency and reliability.

**Infrastructure as Code (IaC):**

Writing and maintaining infrastructure code (using tools like Terraform or CloudFormation) to provision and manage servers, networks, and cloud resources.

Automating infrastructure deployments and updates.

**Configuration Management:**

Using tools like Ansible, Puppet, or Chef to automate server configuration and ensure consistency across environments.

**Containerization and Orchestration:**

Working with Docker containers and container orchestration platforms like Kubernetes to manage and scale applications.

**Monitoring and Alerting:**

Implementing monitoring solutions (e.g., Prometheus, Grafana, ELK stack) to track application and infrastructure performance.

Setting up alerts to respond to incidents proactively.

**Security and Compliance:**

Integrating security practices into the DevOps pipeline (DevSecOps).

Ensuring compliance with industry regulations and company policies.

**Collaboration and Communication:**

Collaborating with development, operations, and other teams to ensure smooth software delivery.

Communicating updates and issues effectively.

**Scripting and Automation:**

Writing scripts (e.g., Bash, Python) to automate routine tasks and troubleshoot issues.

**Version Control:**

Managing code repositories using version control systems (e.g., Git) and ensuring proper branching and merging strategies.

**Performance Optimization:**

Identifying and addressing performance bottlenecks in applications and infrastructure.

**Incident Response and Troubleshooting:**

Participating in incident response and troubleshooting activities to minimize downtime and resolve issues quickly.

**Capacity Planning:**

Analyzing resource usage trends and planning for capacity scaling as needed.

**Documentation:**

Maintaining documentation for configurations, processes, and procedures.

**Backup and Recovery:**

Implementing backup and disaster recovery strategies to ensure data integrity and availability.

**Tooling and Technology Evaluation:**

Staying updated with new DevOps tools and technologies and evaluating their potential benefits for the organization.

**Continuous Learning:**

Keeping up-to-date with industry trends and best practices to continually improve DevOps processes.

It's important to note that the exact responsibilities can vary from one organization to another and may evolve as the organization's needs change. DevOps engineers often work in dynamic environments that require adaptability and a willingness to learn and embrace new technologies.

**How to introduce yourself as a devops engineer?**

Introducing yourself as a DevOps engineer should provide a brief overview of your professional background, skills, and enthusiasm for the DevOps field. Here's a template you can use as a starting point:

"Hello, I'm [Your Name]. I'm a DevOps engineer with [X years] of experience in the field. I specialize in [mention your key areas of expertise, e.g., CI/CD, cloud infrastructure, automation, etc.]. My passion lies in optimizing development and operations processes to enhance software delivery and reliability.

In my previous roles, I have successfully [mention specific accomplishments or projects that highlight your skills]. I'm well-versed in tools like [mention relevant DevOps tools, e.g., Jenkins, Docker, Kubernetes], and I believe in the power of automation to streamline workflows and improve efficiency.

I'm excited about the constant evolution of DevOps practices and the impact they have on delivering high-quality software. I'm eager to collaborate with cross-functional teams and tackle complex challenges to drive innovation and ensure smooth and reliable software delivery.

Outside of work, I enjoy [mention some personal interests or hobbies] and am always looking for opportunities to expand my knowledge and stay current with industry trends. I'm thrilled to be part of the DevOps community and contribute to the success of [current or prospective] projects and teams."

Remember to adapt this introduction to your specific experiences and interests to make it more personal and relevant.

**Operating system**

An operating system (OS) is a software program that manages computer hardware and provides services for computer programs. It acts as an intermediary between the hardware and software, ensuring that various software applications can interact with the hardware components of a computer system without needing to understand the intricate details of the hardware.

Key functions of an operating system include:

**Process Management:** It manages processes (programs in execution) by allocating CPU time, memory, and other resources.

**Memory Management:** It controls and allocates memory resources to different processes and ensures efficient memory usage.

**File System Management:** It manages files and directories, including storage, retrieval, and organization of data on storage devices.

**Device Management:** It controls and coordinates input and output devices such as keyboards, mice, printers, and storage devices.

**User Interface:** It provides a user-friendly interface for interacting with the computer, including graphical user interfaces (GUIs) and command-line interfaces (CLIs).

**Security and Access Control:** It enforces security policies, controls user access to system resources, and protects data from unauthorized access.

**Networking:** It facilitates network communication and manages network connections and protocols.

There are several types of operating systems, each designed for specific purposes and platforms. **Here are some common types:**

**Desktop Operating Systems:** These are designed for personal computers and workstations. Examples include Microsoft Windows, macOS (formerly OS X), and various Linux distributions like Ubuntu.

**Server Operating Systems:** These are optimized for server hardware and are used to manage and control server resources. Examples include Windows Server, Linux server distributions (e.g., CentOS, Debian), and various Unix-based server OSes.

**Mobile Operating Systems:** These are designed for smartphones and tablets. Examples include Android, iOS, and HarmonyOS.

**Embedded Operating Systems:** These are lightweight OSes used in embedded systems and devices like IoT devices, ATMs, and industrial equipment. Examples include Embedded Linux, FreeRTOS, and VxWorks.

**LINUX TUTORILAS**

**LINUX** stands for **Lovable Intellect Not Using XP**. Linux was developed by Linus Torvalds and named after him.

Linux is an open-source, Unix-like operating system kernel. It serves as the core component of many different operating systems, commonly referred to as "Linux distributions" or "Linux distros."

**Open Source:** Linux is released under an open-source license, which means its source code is freely available for anyone to view, modify, and distribute. This open nature has led to a large and active developer community, contributing to its rapid evolution and improvement.

**Kernel:** Linux is essentially the kernel of the operating system, responsible for managing hardware resources, scheduling tasks, and providing a foundation for software applications to run.

**Unix-Like:** Linux is Unix-like, meaning it shares many design principles and features with the Unix operating system, which is known for its stability, security, and scalability.

**Command-Line Interface (CLI):** Linux provides a powerful command-line interface (CLI) in addition to graphical user interfaces (GUIs). The CLI allows users to interact with the system using text commands, making it highly flexible and scriptable.

**Architecture of Linux**

Linux architecture has the following components:

**Kernel**: Kernel is the core of the Linux based operating system. It virtualizes the common hardware resources of the computer to provide each process with its virtual resources. This makes the process seem as if it is the sole process running on the machine. The kernel is also responsible for preventing and mitigating conflicts between different processes. Different types of the kernel are:

Monolithic Kernel

Hybrid kernels

Exo kernels

Micro kernels

**System Library:** Isthe special types of functions that are used to implement the functionality of the operating system.

**Shell:** It is an interface to the kernel which hides the complexity of the kernel’s functions from the users. It takes commands from the user and executes the kernel’s functions.

**Hardware Layer:** This layer consists all peripheral devices like RAM/ HDD/ CPU etc.

**System Utility:** It provides the functionalities of an operating system to the user.

**What is Kernel?**

The kernel is a computer program that is the core of a computer’s operating system, with complete control over everything in the system. It manages the following resources of the Linux system –

File management

Process management

I/O management

Memory management

Device management etc.

It is often mistaken that Linus Torvalds has developed Linux OS, but actually, he is only responsible for the development of the Linux kernel.

Complete Linux system = Kernel + GNUsystem utilities and libraries + other management scripts + installation scripts.

**What is Shell?**

A shell is a special user program that provides an interface for the user to use operating system services. Shell accepts human-readable commands from users and converts them into something which the kernel can understand. It is a command language interpreter that executes commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or starts the terminal.

Shell is broadly classified into two categories –

Command Line Shell

Graphical shell

**Command Line Shell**

Shell can be accessed by users using a command line interface. A special program called Terminal in Linux/macOS, or Command Prompt in Windows OS is provided to type in the human-readable commands such as “cat”, “ls” etc. and then it is being executed. The result is then displayed on the terminal to the user.

**Graphical Shells**

Graphical shells provide means for manipulating programs based on the graphical user interface (GUI), by allowing for operations such as opening, closing, moving, and resizing windows, as well as switching focus between windows. Window OS or Ubuntu OS can be considered as a good example which provides GUI to the user for interacting with the program. Users do not need to type in commands for every action.

There are several shells are available for Linux systems like –

**BASH (Bourne Again SHell)** – It is the most widely used shell in Linux systems. It is used as default login shell in Linux systems and in macOS. It can also be installed on Windows OS.

**CSH (C SHell)** – The C shell’s syntax and its usage are very similar to the C programming language.

**KSH (Korn SHell)** – The Korn Shell was also the base for the POSIX Shell standard specifications etc.

Each shell does the same job but understands different commands and provides different built-in functions.

**What is a terminal?**

A program which is responsible for providing an interface to a user so that he/she can access the shell. It basically allows users to enter commands and see the output of those commands in a text-based interface. Large scripts that are written to automate and perform complex tasks are executed in the terminal.

To access the terminal, simply search in search box “terminal” and double-click it.

**Shell Scripting**

Usually, shells are interactive, which means they accept commands as input from users and execute them. However, sometimes we want to execute a bunch of commands routinely, so we have to type in all commands each time in the terminal.

As a shell can also take commands as input from file, we can write these commands in a file and can execute them in shell to avoid this repetitive work. These files are called **Shell Scripts**or**Shell Programs**. Shell scripts are similar to the batch file in MS-DOS. Each shell script is saved with**`.sh`** file extension e.g., **myscript.sh.**

A shell script has syntax just like any other programming language. If you have any prior experience with any programming language like Python, C/C++ etc. It would be very easy to get started with it.

A shell script comprises the following elements –

Shell Keywords – if, else, break etc.

Shell commands – cd, ls, echo, pwd, touch etc.

Functions

Control flow – if..then..else, case and shell loops etc.

**Why do we need shell scripts?**

There are many reasons to write shell scripts:

To avoid repetitive work and automation

System admins use shell scripting for routine backups.

System monitoring

Adding new functionality to the shell etc.