Experiment No.1
To understand DevOps: Principles, Practices and DevOps Engineer Role and Responsibilities.
Date of Performance:
Date of Submission:

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

Objective: Objective of fostering collaboration, automation, and continuous improvement in software development and IT operations processes.

Theory :

Introduction

DevOps is a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams. It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.

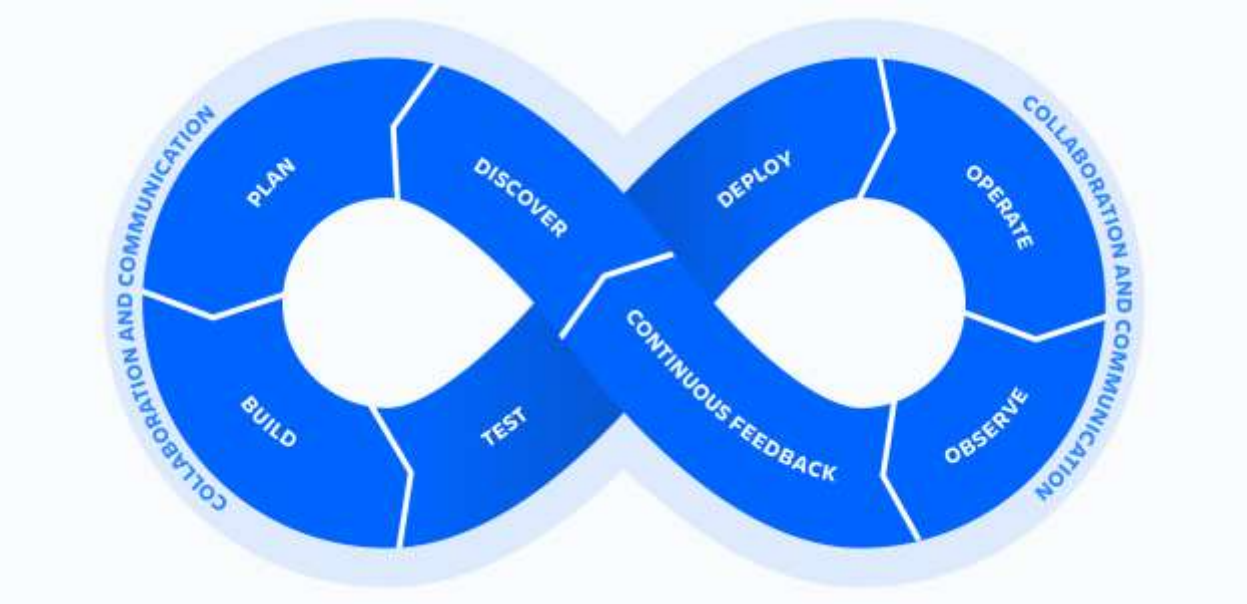

fig. DevOps Lifecycle

The DevOps lifecycle

The DevOps lifecycle consists of eight phases representing the processes, capabilities, and tools needed for development (on the left side of the loop) and operations (on the right side of the loop). Throughout each phase, teams collaborate and communicate to maintain alignment, velocity, and quality.

1. Discover

Building software is a team sport. In preparation for the upcoming sprint, teams must workshop to explore, organize, and prioritize ideas. Ideas must align to strategic goals and deliver customer impact. Agile can help guide DevOps teams.

2. Plan

DevOps teams should adopt agile practices to improve speed and quality. Agile is an iterative approach to project management and software development that helps teams break work into smaller pieces to deliver incremental value.

3. Build

Git is a free and open source version control system. It offers excellent support for branching, merging, and rewriting repository history, which has led to many innovative and powerful workflows and tools for the development build process.

4. Test

Continuous integration (CI) allows multiple developers to contribute to a single shared repository. When code changes are merged, automated tests are run to ensure correctness before integration. Merging and testing code often help development teams gain reassurance in the quality and predictability of code once deployed.

5. Deploy

Continuous deployment (CD) allows teams to release features frequently into production in an automated fashion. Teams also have the option to deploy with feature flags, delivering new code to users steadily and methodically rather than all at once. This approach improves velocity, productivity, and sustainability of software development teams.

6. Operate

Manage the end-to-end delivery of IT services to customers. This includes the practices involved in design, implementation, configuration, deployment, and maintenance of all IT infrastructure that supports an organization's services.

7. Observe

Quickly identify and resolve issues that impact product uptime, speed, and functionality. Automatically notify your team of changes, high-risk actions, or failures, so you can keep services on.

8. Continuous feedback

DevOps teams should evaluate each release and generate reports to improve future releases. By gathering continuous feedback, teams can improve their processes and incorporate customer feedback to improve the next release.

Principles

1. Collaboration

The key premise behind DevOps is collaboration. Development and operations teams come together into a functional team that communicates, shares feedback, and collaborates

throughout the entire development and deployment cycle. Often, this means development and operations teams merge into a single team that works across the entire application lifecycle.

The members of a DevOps team are responsible for ensuring quality deliverables across each facet of the product. This leads to more 'full stack' development, where teams own the complete backend-to-frontend responsibilities of a feature or product. Teams will own a feature or project throughout the complete lifecycle from idea to delivery. This enhanced level of investment and attachment from the team leads to higher quality output.

2. Automation

An essential practice of DevOps is to automate as much of the software development lifecycle as possible. This gives developers more time to write code and develop new features. Automation is a key element of a CI/CD pipeline and helps to reduce human errors and increase team productivity. With automated processes, teams achieve continuous improvement with short iteration times, which allows them to quickly respond to customer feedback.

3. Continuous Improvement

Continuous improvement was established as a staple of agile practices, as well as lean manufacturing and Improvement Kata. It's the practice of focusing on experimentation, minimizing waste, and optimizing for speed, cost, and ease of delivery. Continuous improvement is also tied to continuous delivery, allowing DevOps teams to continuously push updates that improve the efficiency of software systems. The constant pipeline of new releases means teams consistently push code changes that eliminate waste, improve development efficiency, and bring more customer value.

4. Customer-centric action

DevOps teams use short feedback loops with customers and end users to develop products and services centered around user needs. DevOps practices enable rapid collection and response to user feedback through use of real-time live monitoring and rapid deployment. Teams get immediate visibility into how live users interact with a software system and use that insight to develop further improvements.

5. Create with the end in mind

This principle involves understanding the needs of customers and creating products or services that solve real problems. Teams shouldn't 'build in a bubble', or create software based on assumptions about how consumers will use the software. Rather, DevOps teams should have a holistic understanding of the product, from creation to implementation.

Practices

Continuous Integration

Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous Delivery

Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared for a release to production. It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

Microservices

The microservices architecture is a design approach to build a single application as a set of small services. Each service runs in its own process and communicates with other services through a well-defined interface using a lightweight mechanism, typically an HTTP-based application programming interface (API). Microservices are built around business capabilities; each service is scoped to a single purpose. You can use different frameworks or programming languages to write microservices and deploy them independently, as a single service, or as a group of services.

Infrastructure as Code

Infrastructure as code is a practice in which infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration. The cloud's API-driven model enables developers and system administrators to interact with infrastructure programmatically, and at scale, instead of needing to manually set up and configure resources. Thus, engineers can interface with infrastructure using code-based tools and treat infrastructure in a manner similar to how they treat application code. Because they are defined by code, infrastructure and servers can quickly be deployed using standardized patterns, updated with the latest patches and versions, or duplicated in repeatable ways.

Configuration Management

Developers and system administrators use code to automate operating system and host configuration, operational tasks, and more. The use of code makes configuration changes repeatable and standardized. It frees developers and systems administrators from manually configuring operating systems, system applications, or server software.

Monitoring and Logging

Organizations monitor metrics and logs to see how application and infrastructure performance impacts the experience of their product's end user. By capturing, categorizing, and then analyzing data and logs generated by applications and infrastructure, organizations understand how changes or updates impact users, shedding insights into the root causes of problems or unexpected changes. Active monitoring becomes increasingly important as services must be available 24/7 and as application and infrastructure update frequency increases. Creating alerts or performing real-time analysis of this data also helps organizations more proactively monitor their services.

Communication and Collaboration

Increased communication and collaboration in an organization is one of the key cultural aspects of DevOps. The use of DevOps tooling and automation of the software delivery process establishes collaboration by physically bringing together the workflows and responsibilities of development and operations. Building on top of that, these teams set strong cultural norms around information sharing and facilitating communication through the use of chat applications, issue or project tracking systems, and wikis. This helps speed up communication across developers, operations, and even other teams like marketing or sales, allowing all parts of the organization to align more closely on goals and projects.

DevOps Engineer

A DevOps engineer is an IT generalist who should have a wide-ranging knowledge of both development and operations, including coding, infrastructure management, system administration, and DevOps toolchains. DevOps engineers should also possess interpersonal skills since they work across company silos to create a more collaborative environment.

DevOps engineers need to have a strong understanding of common system architecture, provisioning, and administration, but must also have experience with the traditional developer toolset and practices such as using source control, giving and receiving code reviews, writing unit tests, and familiarity with agile principles.

Role and Responsibilities

The role of a DevOps engineer will vary from one organization to another, but invariably entails some combination of release engineering, infrastructure provisioning and management, system administration, security, and DevOps advocacy.

Release engineering includes the work required to build and deploy application code. The exact tools and processes vary widely depending on many variables, such as what language the code is written in, how much of the pipeline has been automated, and whether the production infrastructure is on-premise or in the cloud. Release engineering might entail selecting, provisioning, and maintaining CI/CD tooling or writing and maintaining bespoke build/deploy scripts.

Infrastructure provisioning and system administration include deploying and maintaining the servers, storage, and networking resources required to host applications. For organizations with on-premise resources this might include managing physical servers, storage devices, switches, and virtualization software in a data center. For a hybrid or entirely cloud-based organization this will usually include provisioning and managing virtual instances of the same components.

DevOps advocacy is often undervalued or overlooked entirely, but is arguably the most important role of a DevOps engineer. The shift to a DevOps culture can be disruptive and confusing to the engineering team members. As the DevOps subject matter expert, it falls to the DevOps engineer to help evangelize and educate the DevOps way across the organization.

Conclusion :

Q1. What is the need for DevOps?
Q2. Explain the role of DevOps Engineer?