API LAYER PROJECT

IMPLEMENTATION OF DEVOPS CONTINUOUS INTEGRATION (CI) AND CONTINUOUS DELIVERY/DEPLOYMENT (CD)

Agenda

- What is DevOps?
- What is DevOps CI/CD?
- DevOps Lifecycle
- Implementation of DevOps CI/CD for API Layer Project
- DEMO

What is DevOps?

DevOps is a set of practices, principles, and culture that aims to improve collaboration, communication, and integration between software development teams (**Dev**) and IT operations teams (**Ops**).

The DevOps approach emphasizes the need for **automation**, continuous delivery, and continuous testing to streamline the software development and deployment process. It also promotes the use of agile and lean methodologies to increase efficiency and reduce waste.

The key benefits of DevOps include faster time to market, increased agility and flexibility, improved quality and reliability of software, and greater collaboration between teams.

What is DevOps? (continued)

DevOps practices can include continuous integration and continuous delivery (CI/CD), infrastructure as code, monitoring and logging, and automation of testing and deployment processes. By adopting these practices, organizations can accelerate the delivery of software, reduce errors and downtime, and improve overall business outcomes.

Overall, DevOps is an important approach to modern software development and is increasingly being adopted by organizations of all sizes and types.

What is DevOps CI/CD?

DevOps CI/CD is an approach to software development that combines DevOps practices with continuous integration (CI) and continuous delivery (CD) methodologies.

Continuous Integration (CI) is the practice of merging code changes into a **shared repository** frequently, which triggers an automated build and test process to catch errors early in the development cycle.

Continuous Delivery/Deployment (CD), on the other hand, is the practice of **automating** the software release process to deliver updates and new features to customers as quickly and frequently as possible.

What is DevOps CI/CD? (continued)

DevOps CI/CD promotes **collaboration** and **communication** between development and operations teams, enabling them to work together to deliver high-quality software at a faster pace.

By automating many of the manual processes involved in building, testing, and deploying software, teams can focus on delivering value to customers more quickly and with greater efficiency.

DevOps Lifecycle

The **DevOps Lifecycle** is the end-to-end process of software **development**, **deployment**, and **management** using the DevOps approach. It typically includes the following stages:

Plan: In this stage, the team identifies the **requirements**, **goals**, and **timelines** for the software project. This includes gathering **feedback** from stakeholders and creating a plan for development and deployment.

Code/Develop: In this stage, developers write code and collaborate with other team members to create a working software product. The code is typically committed to a **shared repository** and tested using automated tools.

DevOps Lifecycle (continued)

Build: In this stage, the code is compiled and built into an executable format. This may include the creation of **libraries**, **packages**, or **containers**.

Test: In this stage, **automated tests** are run to ensure that the code meets the required standards and quality. This includes unit testing, integration testing, and other forms of testing.

Release: In this stage, the software is released to a **staging** or **production** environment. This may include the use of continuous delivery pipelines to automate the deployment process.

DevOps Lifecycle (continued)

Deploy: In this stage, the software is deployed to the target environment, such as a cloud server or customer device. This may include the use of **automated configuration management tools**.

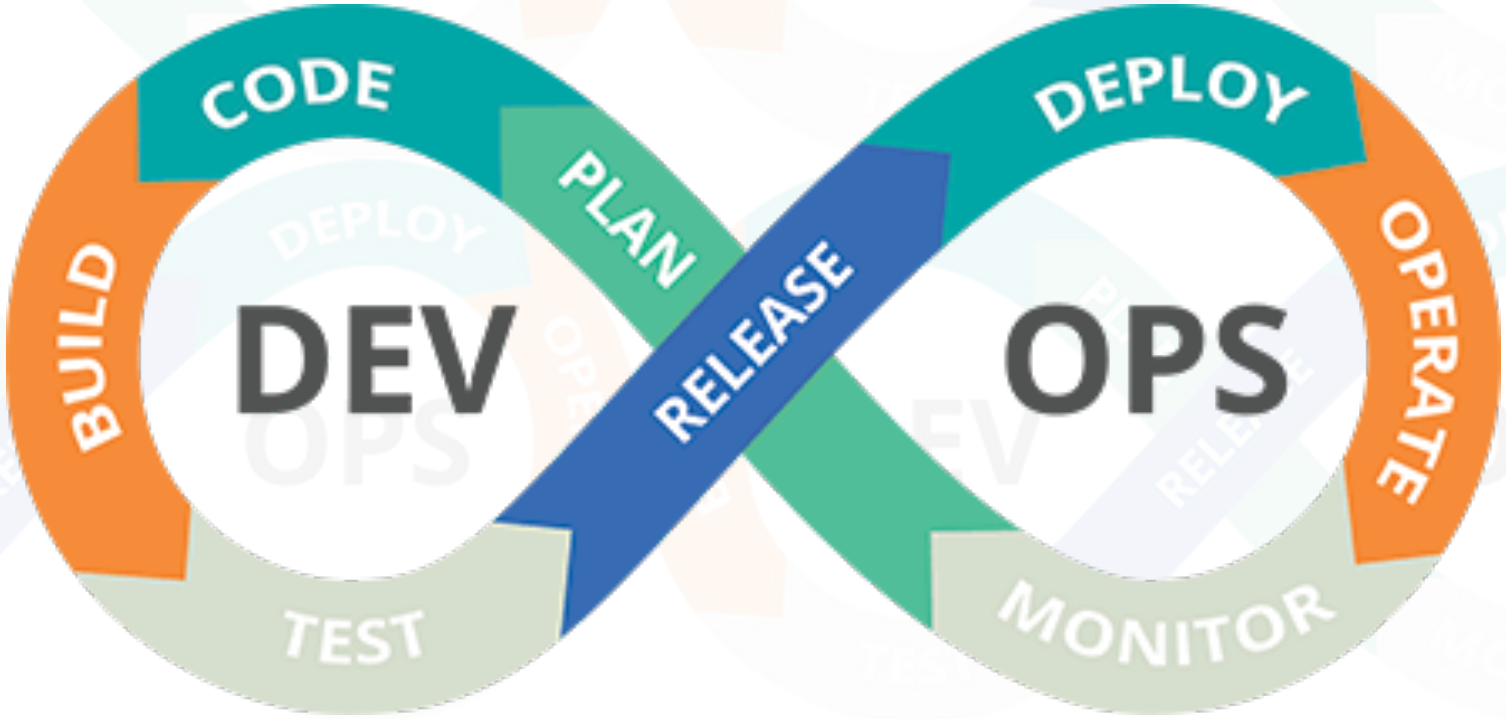
Operate: In this stage, the software is **monitored**, **managed**, and **maintained** in production. This includes the use of tools to monitor performance, troubleshoot issues, and manage updates and patches.

Monitor: In this stage, the team **collects** and **analyzes data** from the software to identify areas for improvement and to ensure that it is meeting the needs of users and stakeholders.

DevOps Lifecycle (continued)

Overall, the DevOps lifecycle is a continuous process of planning, developing, testing, deploying, and managing software. By adopting a DevOps approach, teams can streamline (organize) the entire software lifecycle and improve collaboration, communication, and integration between development and operations teams.

DevOps Lifecycle (continued)



Implementation of DevOps CI/CD for API Layer Project

Plan: API Layer project will be developed in **Node.js** and will be containerized using **Docker**. It will be hosted in **Azure Web App Linux**.

Code: Developers and DevOps agreed to use a **shared-repository** or release branch for the CI/CD pipeline. Branch name will be formatted **release/<yyyymmdd>** wherein <yyyymmdd> represents the **target date** of production deployment. (E.g. **release/20230308**)

Build: **Dockerfile(s)** will be provided by developer. It will be the basis of the containerization, wherein a **docker image** is created which will be used to create a Docker container, and then pushed to **Azure Container Registry** (ACR).

Implementation of DevOps CI/CD for API Layer Project (continued)

Test: **Automated test** (unit testing) will be revisited and implemented soon.* Although, automated tests software of **Sonarqube** and **Mend Bolt (formerly WhiteSource)** are available in **Azure DevOps Marketplace**, but they were not created by Microsoft per se; hence, updates from these software/extensions are not in Microsoft's control (nor ours). These software/extensions have the capabilities to check for any code vulnerabilities during build. They have both free and paid editions.

Action item for developer.

Implementation of DevOps CI/CD for API Layer Project (continued)

Release: After API Layer project is **containerized** (dockerized) and pushed to Azure container registry (ACR), the release pipeline will be created and connect to it.

Deploy: API Layer application will be deployed to its Azure web app *qaapilayer2* (**QA/staging environment**) first. Once QA signed off, the production deployment to *slsaapi* (**production environment**) will be triggered once the **Approval stage** is approved (manual intervention). An email will also be sent to notify the Approver(s). This is the actual **promotion** of a release from QA to Production.

Implementation of DevOps CI/CD for API Layer Project (continued)

Operate: Technically, prior to any deployments of applications, the environments should be set/ready with what is required.

An example would be the following:

- Custom Domains
- SSL/Certificates
- Firewall setup
- DBs
- Etc.

Implementation of DevOps CI/CD for API Layer Project (continued)

Normally, these requirements are communicated during the planning stage of the lifecycle among stakeholders, project owners, development and operations teams.

Operations team will also start monitoring, managing, and maintaining the newly deployed application.

Furthermore, for API Layer application, Azure **scale-out** was enabled to keep it in top performance. It currently has **2 instances** running in **minimum** and capable of scaling out to 3 when 70% of CPU usage started to kick in. The maximum number of instances allowed is 5.

Implementation of DevOps CI/CD for API Layer Project (continued)

Monitor: **Azure Log Stream** provides basic log data collection for analysis. But another logging system for API Layer itself was set up using Azure storage. The data collected from logs will help with the further improvements of the API Layer application.

DEMO