DEVOPS

Training Material

DEVOPS TOOLS AND AUTOMATION

TABLE OF CONTENTS

Chapter 1: Overview of DevOps

1.1 Overview of DevOps

- **Definition of DevOps**
  - DevOps is a set of practices that combines software development (Dev) and IT operations (Ops) to shorten the development lifecycle while delivering features, fixes, and updates frequently in close alignment with business objectives.
- **Key Principles of DevOps**
  - **Collaboration**: Breaking down silos between development and operations teams.
  - **Automation**: Automating repetitive tasks, including testing, integration, deployment, and infrastructure management.
  - **Continuous Improvement**: Regular, incremental updates and performance monitoring to optimize systems.
  - **Customer-centric Action**: Delivering value to the customer with faster and reliable releases.
- **DevOps Goals**
  - **Speed**: Faster software delivery.
  - **Improved Quality**: Enhanced testing and error detection.
  - **Reliability**: Consistent system performance.
  - **Scalability**: Easily scaling infrastructure to meet demand.
- **DevOps vs. Other Methodologies**
  - Difference from Agile, Waterfall, and Lean methodologies, with a focus on end-to-end automation and cross-team collaboration.

1.2 DevOps History

- **Origins of DevOps**
  - The term "DevOps" emerged in 2009 as a movement to overcome the challenges associated with the separation between development and operations teams. The idea was introduced by Patrick Debois, and the movement grew rapidly at DevOpsDays, a series of conferences that focused on collaboration between development and operations.
- **Early Software Development Challenges**
  - Traditionally, development and operations were separate functions. Developers would write code, and then operations teams were responsible for deploying and maintaining it in production, leading to bottlenecks, miscommunication, and delays.

- **Evolution of DevOps Practices**
    - **Pre-DevOps Era**: Development followed a linear, waterfall model with limited cross-team interaction.
    - **Introduction of Agile**: Agile introduced iterative development but didn't fully address the gap between development and operations.
    - **DevOps Revolution**: With the rise of cloud computing, the need for rapid and reliable deployment became critical. DevOps emerged to align development and operations, focusing on continuous integration, delivery, and automated workflows.
- **Notable Milestones**
    - **2009**: The first DevOpsDays event in Belgium.
    - **2011**: The concept of Continuous Integration (CI) and Continuous Delivery (CD) started gaining momentum.
    - **2015-Present**: DevOps became mainstream, with tools like Jenkins, Docker, and Kubernetes becoming foundational in DevOps practices.

1.3 How DevOps is Different from Traditional IT

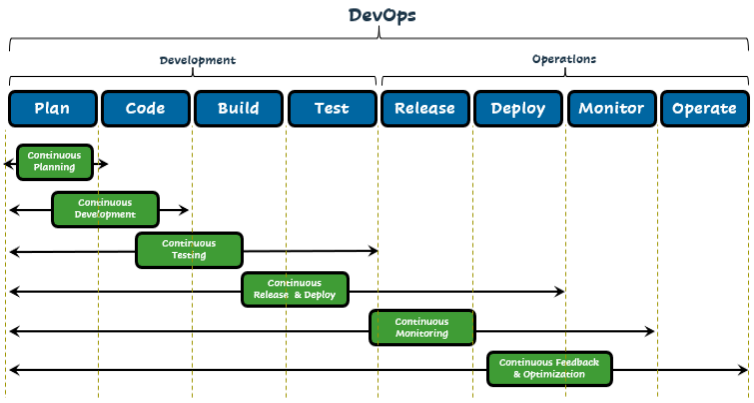
- **Traditional IT Practices**
    - **Development vs. Operations Divide**: In traditional IT environments, the development and operations teams work in silos with distinct roles and responsibilities.
    - **Waterfall Model**: Follows a sequential development process, often leading to long development cycles and delayed releases.
    - **Manual Processes**: Operations are managed manually, leading to slower deployment times and increased risk of human errors.
- **DevOps Practices**
    - **Integrated Teams**: Developers, operations engineers, and quality assurance (QA) engineers work closely together throughout the software development lifecycle.
    - **Automation**: Tools and scripts automate testing, integration, and deployment tasks, resulting in faster and more reliable releases.
    - **Continuous Integration/Continuous Delivery (CI/CD)**: Ensures that new code is integrated and delivered quickly and efficiently, reducing the time it takes to release updates or fixes.
- **Key Differences**
    - **Collaboration**: DevOps encourages a culture of collaboration, where teams share responsibility for the success of the product.
    - **Agility**: DevOps practices are iterative and agile, leading to quicker releases and better responsiveness to changes.

- **Automation**: Automated deployment, infrastructure provisioning, and testing reduce manual efforts and errors, leading to more reliable systems.

1.4 DevOps Organization Structure

- **Cross-functional Teams**
  - In a DevOps organization, teams are structured in a way that encourages collaboration between different roles. The traditional boundaries between developers, operations engineers, and QA are minimized or eliminated.
- **Roles in DevOps Teams**
  - **DevOps Engineers**: Work across both development and operations to create efficient workflows.
  - **Software Developers**: Collaborate with operations to build code that can be deployed quickly and reliably.
  - **Site Reliability Engineers (SREs)**: Focus on ensuring system reliability and scalability through monitoring, automation, and performance improvements.
  - **QA Engineers**: Automate testing and ensure quality at every stage of the pipeline.
  - **Security Engineers (DevSecOps)**: Integrate security into the development and operations processes to ensure that security is baked into every part of the application lifecycle.
- **Team Structures**
  - **Centralized DevOps Team**: A specialized team responsible for overseeing DevOps processes, tools, and culture across the organization.
  - **Embedded DevOps Team**: DevOps roles are embedded within development and operations teams for closer collaboration.
  - **DevOps as a Culture**: In some organizations, DevOps is treated as a cultural shift rather than a distinct team, where everyone is responsible for the product's success.

DevOps Lifecycle



DevOps – Continuous Development

- Continuous Development is the ***first and foremost phase*** of the DevOps lifecycle.
- In this phase, ***planning and coding*** of the software take place.
- The version of the project is also decided during the planning.
- At the same time, software developers write the code for applications.
- The planning does not require any primary tools, but there are certain tools for maintaining the code.

While developing a source code begins by choosing a development language. Python, C, C++, Ruby, and JavaScript are some of the DevOps most popularly used languages

DEVOPS – Continuous Integration

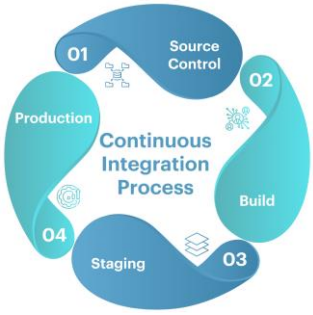
- Continuous Integration, generally known as CI, is one of the most important phases of DevOps Lifecycle.
- Continuous integration is a software development practice where developers need to commit ***frequent changes*** to the source code.
- The changes occur daily or at a gap of a few days .
- The code is compiled and goes through ***unit testing, integration testing, packaging, and code review.***

DevOps – Continuous Integration Process

Generally, the continuous integration process consists of 4 significant steps. These steps are:

1. Source Control (Commit Change)
2. Build (Run Build And Unit Tests)
3. Staging (Deploy To Test Environment And Run Tests)
4. Production (Deploy To Production Environment)

DevOps-Continuous Integration Process



DevOps – Benefits of CI

- CI can find bugs and errors quicker through frequent testing.
- Continuous Integration delivers updates at a more frequent rate to the customers.
- Reduce last-minute risk of the project.
- CI helps organizations to scale better and quicker.
- It improves the feedback loop by providing faster feedback for business decisions.
- CI improves overall accountability and communication between the teams.

DevOps –Continuous Testing

- In the continuous testing phase, the developed software is ***continuously tested for bugs.***
- A test environment is created with the help of ***Docker and Containers.***
- Automated tests are run, and reports are generated for the test evaluation process.
- At last, a ***UAT (User Acceptance Testing)*** is done, and the software is bug-free and ready to move to the next phase of DevOps lifecycle.

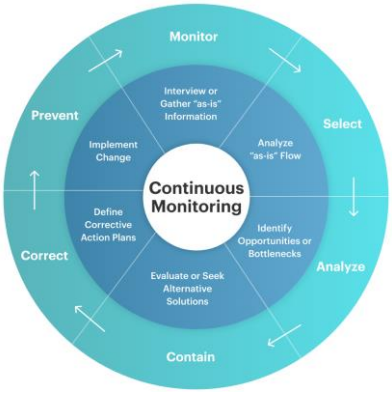
DevOps – Continuous Testing

DevOps –Continuous Testing Benefits

- ❑ Faster, continuous, and reliable feedback
- ❑ Minor code changes
- ❑ Faster MTTR (Mean Time To Resolution)
- ❑ Faster software release
- ❑ Early bug and error detection
- ❑ Less time needed for code review
- ❑ Resource-saving while implementing CI
- ❑ Mitigate risk at early stages of development

DevOps –Continuous Monitoring

- ❑ In this phase of DevOps life cycle, we ***monitor the performance of application.***
- ❑ This information is then used to organize the functionalities of the entire application.
- ❑ Errors such as '***server not reachable***' or '***low memory***' are resolved in this phase.
- ❑ Also, the root cause of the issues are found and fixed. Other ***network-related*** problems are also taken care of in this phase of DevOps lifecycle.



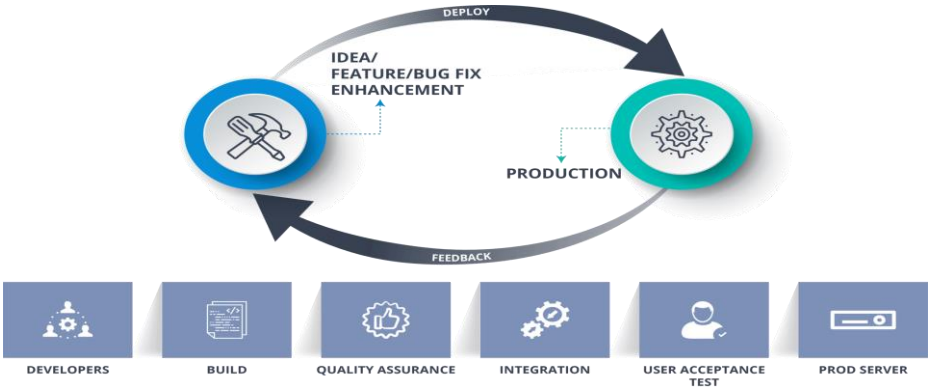
DevOps –Continuous Feedback

- ❑ Continuous feedback is critical to ***optimize DevOps processes***.
- ❑ While tools can enable ***feedback loops***, DevOps teams still must reduce unnecessary noise and continually measure success.
- ❑ Once all the changes are implemented, the DevOps team moves forward to release the new version of the software.

DevOps –Continuous Deployment

- Continuous Deployment in DevOps is a software release process in which any code that passes the automated testing phase is released into the production environment. These changes are directly reflected to the end software users.
- It is essential to ensure that the code is correctly used on all the servers.
- The key aspect of the continuous deployment phase is ***configuration management***.
- ***Vagrant*** and ***Docker*** are popular tools that are used for this purpose.

DevOps –Continuous Deployment



DevOps –Continuous Operation

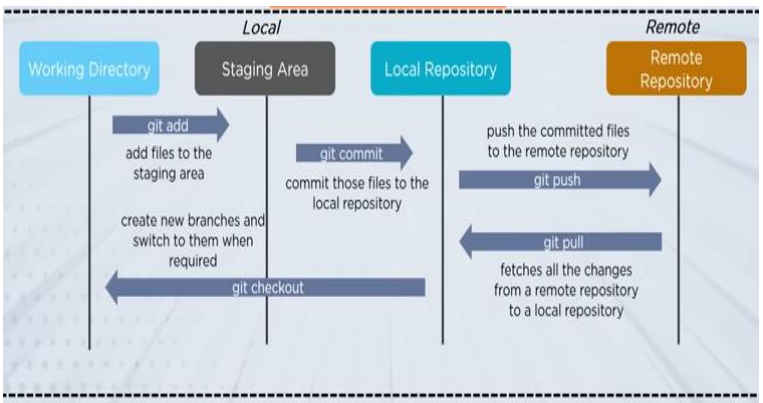
- Continuous Operations is the last and the shortest phase of DevOps lifecycle.
- The main aim of continuous operations is to automate the release of software and its updates.
- The development cycle in this phase of DevOps is short, allowing engineers a faster time to market the product

VARIOUS TOOLS IN DEVOPS

GIT

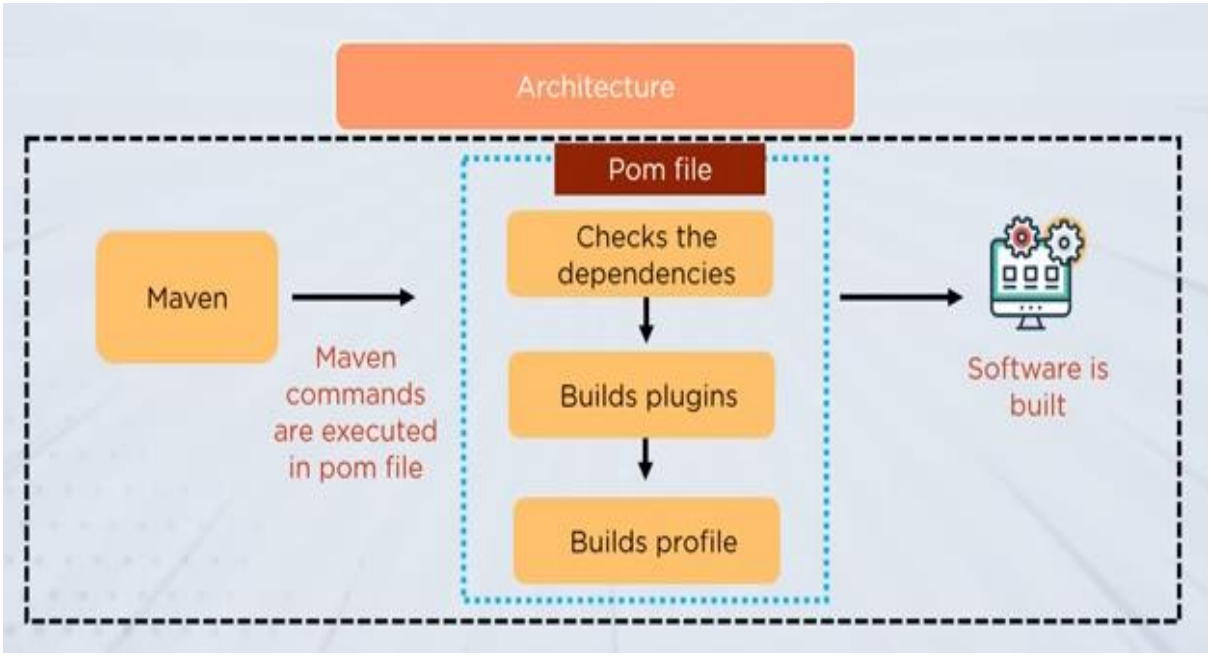
Git is a distributed version control tool which is used to manage different versions of the source code.

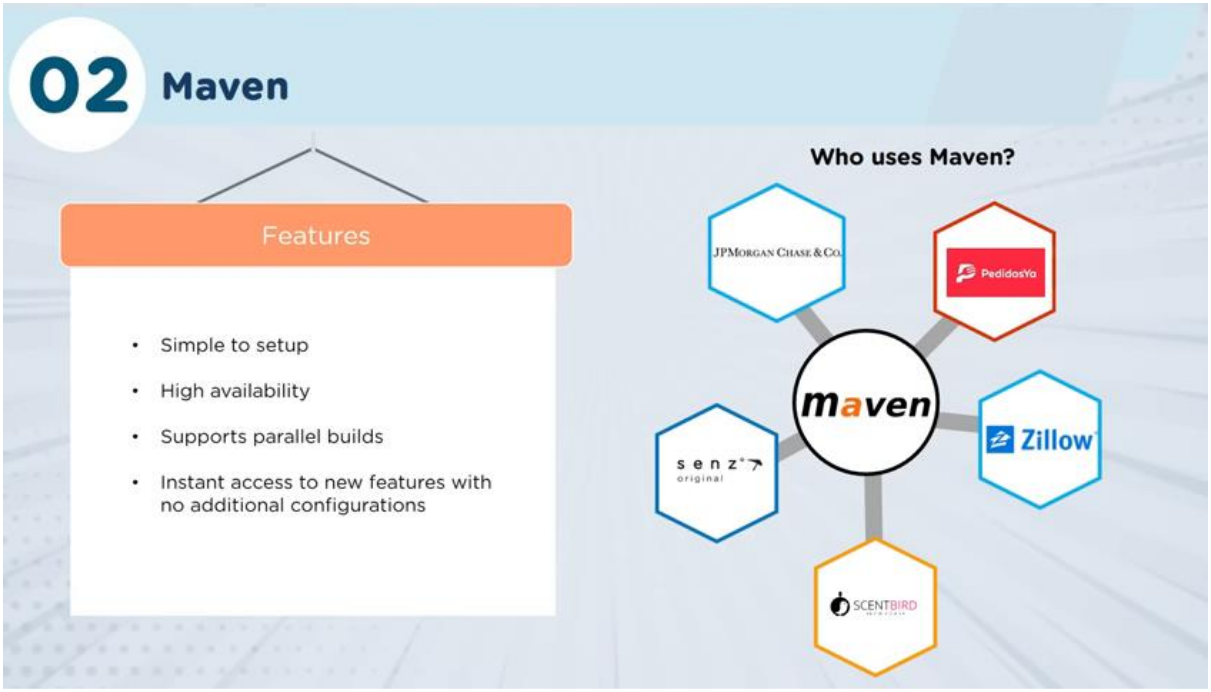
ARCHITECTURE:



2.MAVEN

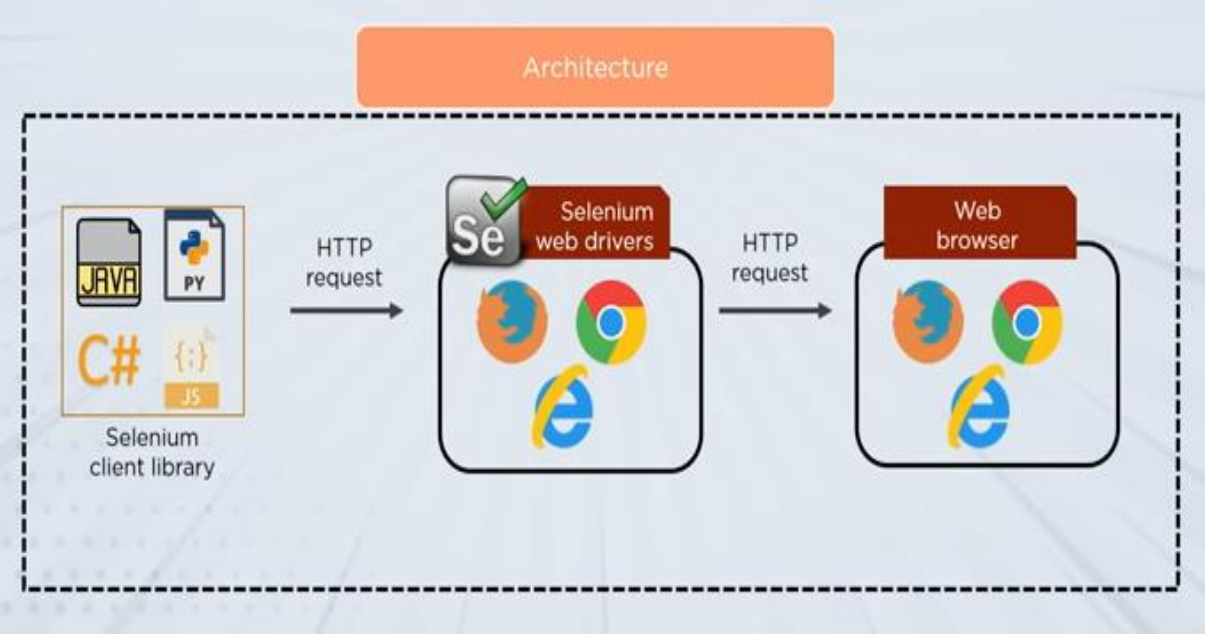
Maven is an automation tool which helps to build and manage software projects within a less period of time.





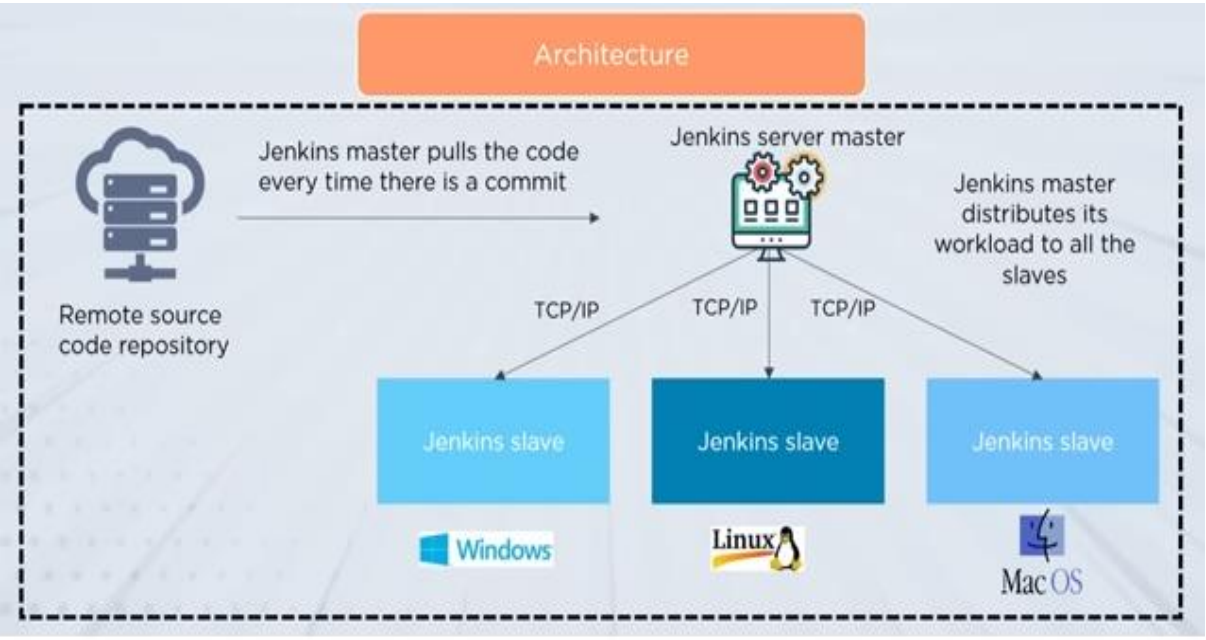
3.SELENIUM

Selenium is an open source automation tool which is used for testing web application.



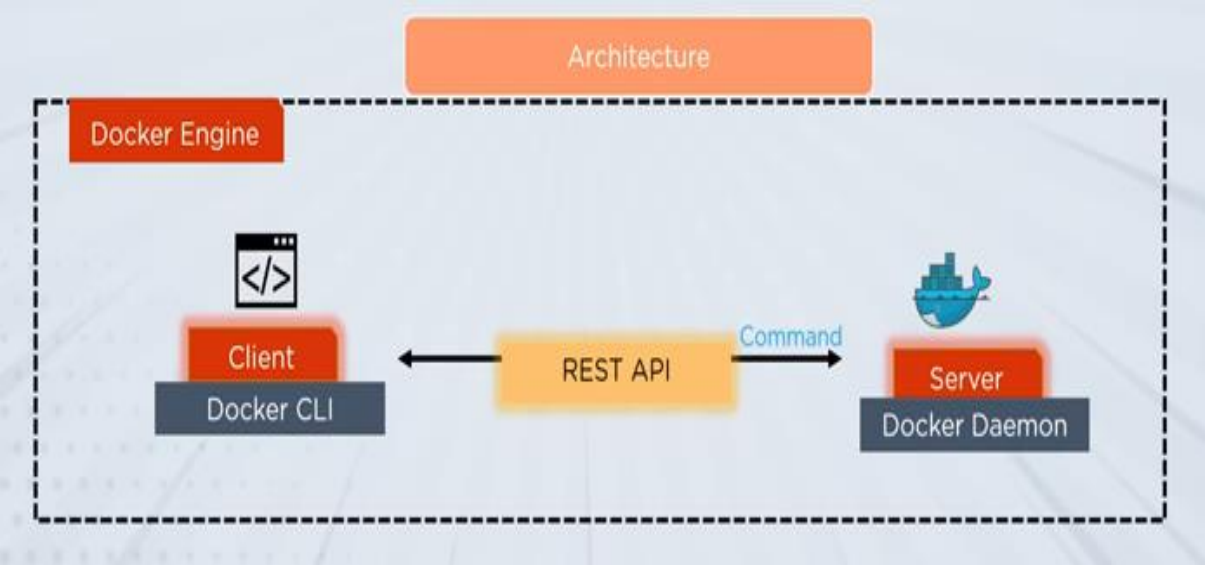
4.JENKINS

- Jenkins is a continuous integration tool which is used to building, testing and deploying software application



5.DOCKER

- Docker is tool which is used to automate the deployment of application in lightweight containers so that application can work efficiently in different environments.



1.GIT and GIT HUB

GIT:

- Git is a version control system widely used in software development as part of DevOps practices.
- It allows developers to track changes in their codebase over time and collaborate with others on the same codebase.

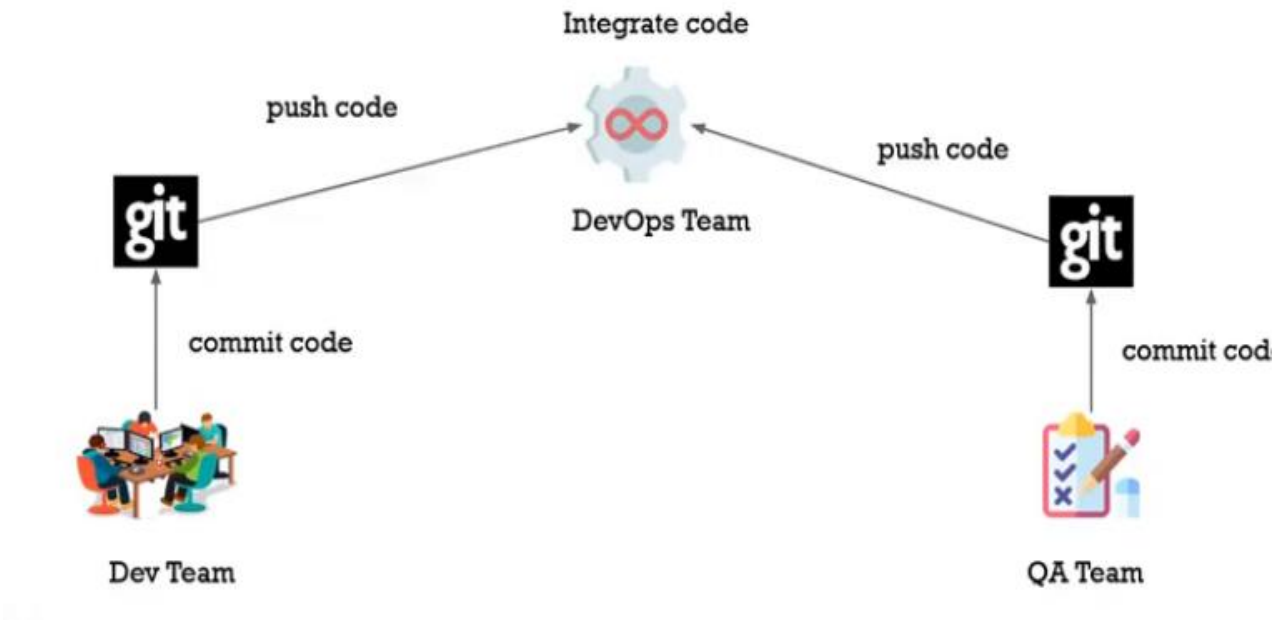
GIT HUB:

- GitHub is a web-based platform for hosting Git repositories.
- It provides a graphical user interface for managing Git repositories and offers features such as issue tracking, code review, and project management tools.
- Developers can use GitHub to collaborate on projects and contribute to open-source projects.

GitHub provides a social platform for developers to share their code and collaborate with other developers. It allows developers to fork repositories, which creates a copy of the repository that they can modify and use as a starting point for their own projects. Developers can also submit pull requests to contribute changes to the original repository, and other developers can review and approve these changes before they are merged into the main repository.

GIT vs GIT HUB

Git is a version control system, and GitHub is a platform for hosting and collaborating on Git repositories.



Working with Git and GitHub

Steps to Install and Use Git on Windows

Git

Git is a widely used open-source software tracking application used to track projects across different teams and revision levels.

Prerequisites

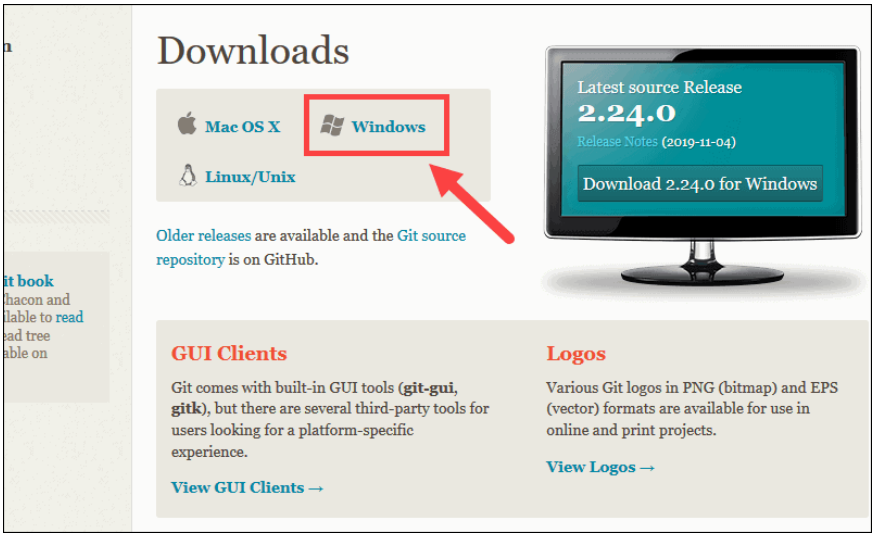
- Administrator privileges
- Access to a command-line
- Your favourite coding text editor
- Username and password for the Git hub website (optional)

Steps for Installing Git for Windows

Installing Git prompts you to select a text editor. If you don't have one, we strongly advise you to install prior to installing Git.

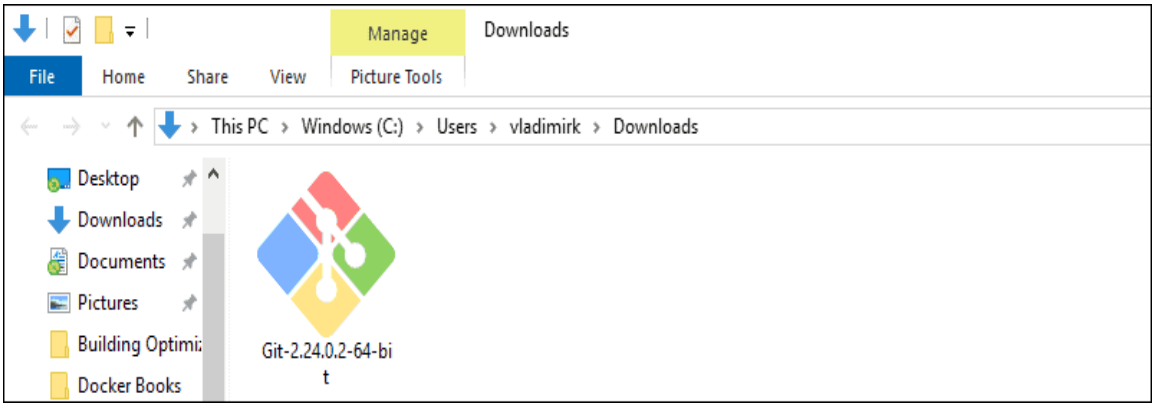
Download Git for Windows

1. Browse to the official Git website: https://git-scm.com/downloads
2. Click the download link for Windows and allow the download to complete.

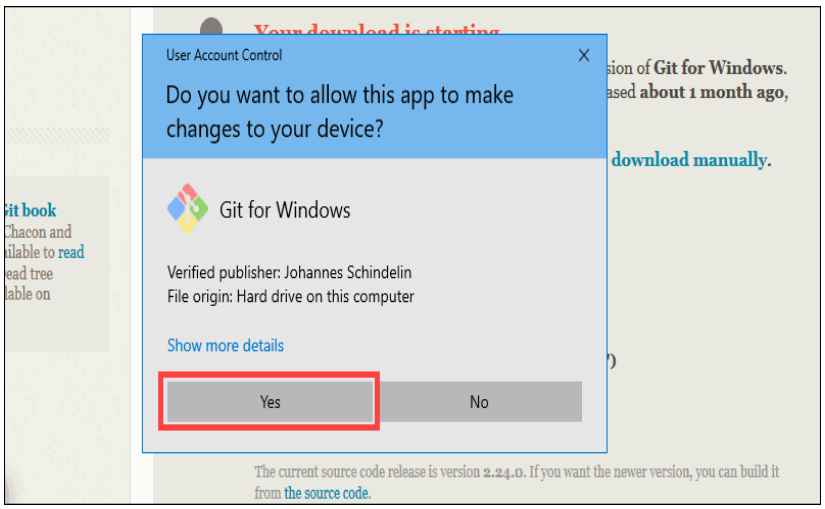


Extract and Launch Git Installer

3. Browse to the download location (or use the download shortcut in your browser). Double-click the file to extract and launch the installer.



4. Allow the app to make changes to your device by clicking yes on the User Account Control dialog that opens.



5. Review the GNU General Public License, and when you're ready to install, click next.

6. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click next.

GIT COMMANDS TO PUSH FILE IN GIT HUB REPOSITORY:

To check the version of git

git --version

To Configure Git

The git config user. name and git config user.email commands are used to set the username and email address for Git commits. These values are associated with the commits you make in a Git repository and help identify who made each commit.

git config –-global user.name "---"

git config –-global user.email ------

"Print Working Directory". It is used to display the current directory or path of the user in the shell.

pwd

- The git init command is used to initialize the git into the existing project. By using this command we can convert the normal project folder into a git repository.
- Initializes a new Git repository in the current directory.

git init

Git add: The git add command is used to place the modified version of the working directory in the staging area.

touch alpha.txt

Git to use Notepad as the default text editor notepad alpha.txt

[edit the file beta gamma]

To check status

git status

git add <file>: Adds a file to the staging area, which prepares it to be committed.

git add .

git status: Displays the status of the current branch, including any untracked, modified, or staged files.

git status

git commit -m "<commit message>": Commits the changes in the staging area to the local repository with a message describing the changes.

git commit -m “beta gamma”

git log: Shows the commit history of the current branch.

git log

The git remote add origin <path> command is used to add a remote repository to your local Git repository. The origin is typically the default name given to the remote repository.

The <path> argument specifies the location of the remote repository.

git remote add origin _____path____

The git remote -v command is used to list the remote repositories that are currently associated with your local Git repository. The -v option shows the verbose output, which includes the URL of the remote repository next to the name of the remote repository.

git remote - v

The git push -u origin master command is used to push the changes in your local "master" branch to the remote "origin" repository. The -u option is used to set the upstream branch for the "master" branch to the remote "origin/master" branch, which means that future pushes and pulls will be automatically synchronized with the remote "origin" repository.

Here's a breakdown of the command:

git push: This command is used to push changes from the local repository to a remote repository.

-u: This option is used to set the upstream branch for the current branch to the specified remote branch.

origin: This is the name of the remote repository that you want to push your changes to. "Origin" is the default name given to the remote repository, but you can use any name you like.

master: This is the name of the local branch that you want to push to the remote repository. You can replace "master" with the name of any other branch that you want to push.

When you run this command, Git will upload the changes in your local "master" branch to the remote

git push -u origin master

----------COMPLETED----------

1. JENKINS TOOL:

- Jenkins is an open-source automation server that is commonly used in DevOps for ***Continuous Integration (CI) and Continuous Deployment (CD).***

- Jenkins allows developers to ***build, test, and deploy their applications automatically***, thereby reducing the time and effort needed for these tasks.
- Jenkins is highly extensible, and there are thousands of plugins available that can be used to customize the build and deployment process.
- Jenkins also supports integration with various tools used in the DevOps toolchain, such as ***Git, GitHub, Jira, Docker, Kubernetes***, and many others.
- Jenkins works by executing a ***series of jobs*** that are defined in the Jenkins file.
- The jobs can be configured to perform tasks such as compiling code, running unit tests, packaging the application, deploying it to a server, and sending notifications.

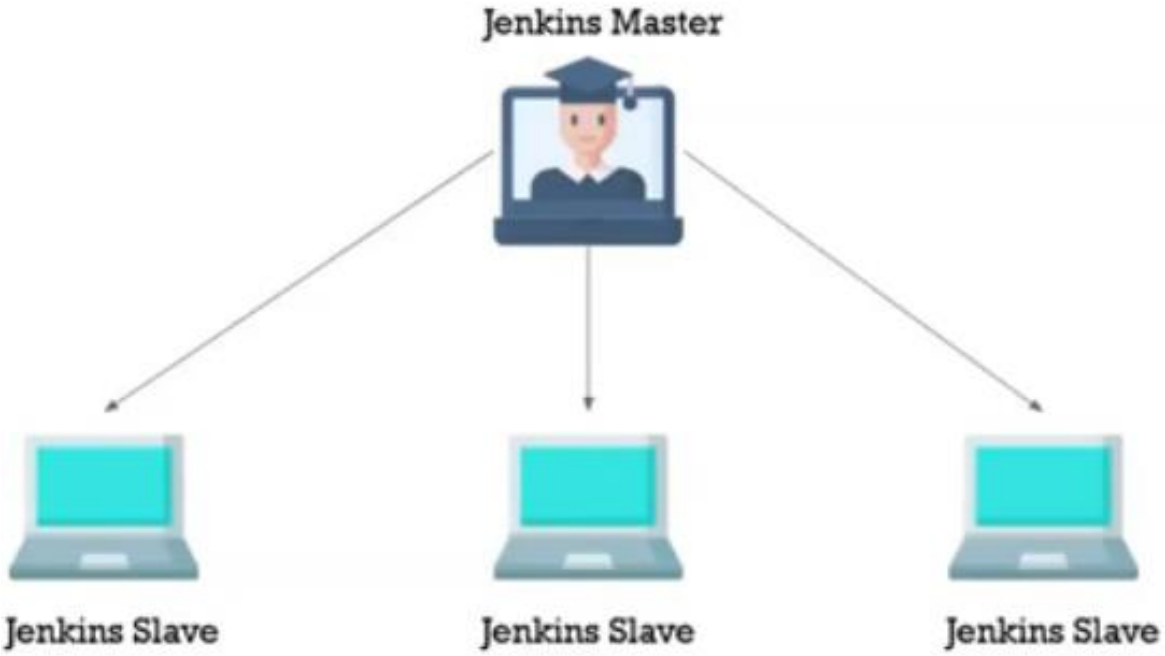
WORKING IN JENKINS:

1. **Install and set up Jenkins:**

Once installed, you can configure Jenkins by creating a new job, setting up source code management, and defining build triggers.

2. **Create a new job:**
    - In Jenkins, a ***job represents a task*** or a step in the build process.
    - To create a new job, go to the Jenkins dashboard, click on "***New Item,"*** enter a name for the job, and select the type of job you want to create.
3. **Configure the job:**
    - Once you create a job, you can configure it by defining the ***build steps, build triggers***, and other parameters.
    - For example, you can define the source code repository, build tool, and target platform for your job.
4. **Run the job:**
    - After configuring the job, you can run it manually or set up ***automatic triggers*** to run it based on ***certain conditions***.
    - Jenkins will execute the build steps defined in the job and report the results.
5. **Monitor the job**:
    - Jenkins provides ***real-time feedback*** on the ***status*** of the build process.
    - We can monitor the job by checking the ***console output,*** build history, and build artifacts.

JENKINS ARCHITECTURE



COMPONENTS IN JENKINS

- **Master Server:**
  - ✓ Controls Pipeline
  - ✓ Schedules Build
- **Agents and Minions:**
  - ✓ Performs the Build

AGENTS:

- Commit Triggers Pipeline
- Agent selected based on Configured Labels
- Agent runs Build

Types of Agents:

1.Pernament Agents

Dedicated servers for running Jobs.

2.Cloud Agents:

Dynamic Agents spun up on demand

BUILD TYPES:

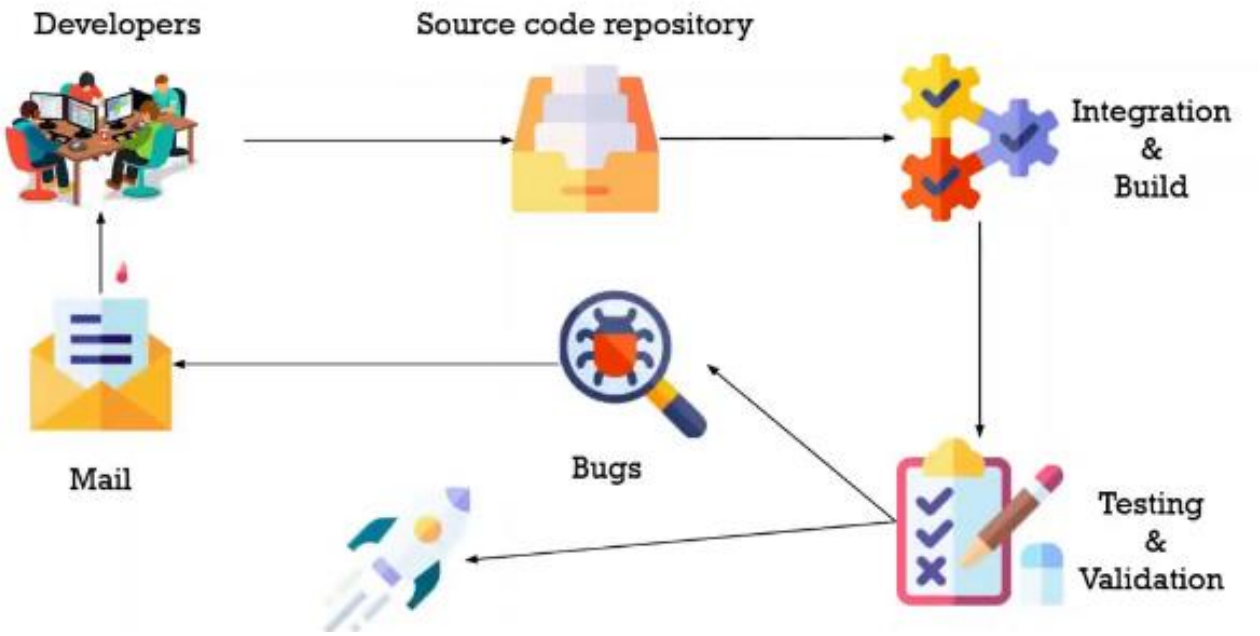
1. Freestyle Build:

- Simplest method to create a build
- "Feels" like Shell Scripting

2.Pipelines:

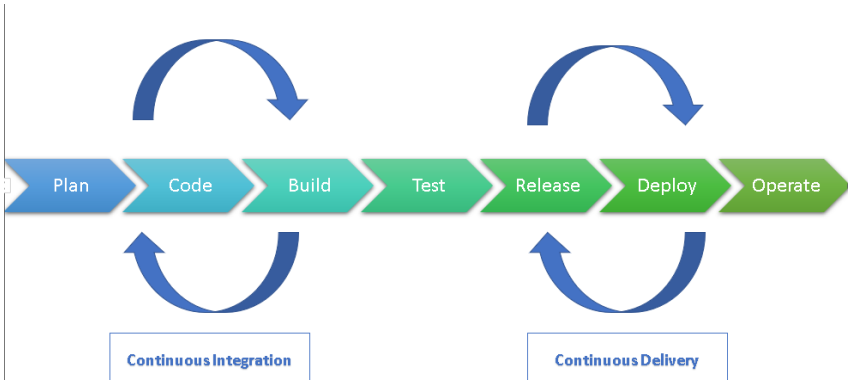
- Use the Groovy Syntax
- Use Stages to break down components

PROCESS BEFORE JENKINS(CI)



PROCESS AFTER JENKINS(CI)

OVERVIEW ABOUT CI/CD Pipeline



CI/CD Pipeline

- A ***Continuous Integration/Continuous Delivery (CI/CD)*** pipeline is a framework that emphasizes iterative, reliable code delivery processes for agile DevOps teams.
- It involves a workflow encompassing continuous integration, testing, delivery, and continuous delivery/deployment practices.
- The pipeline arranges these methods into a unified process for developing high-quality software.
- Test and build automation is key to a CI/CD pipeline, which helps developers identify potential code flaws early in the software development lifecycle (SDLC).
- It is then easier to push code changes to various environments and release the software to production.
- Automated tests can assess crucial aspects ranging from application performance to security.
- In addition to testing and quality control, automation is useful throughout the different phases of a CI/CD pipeline.

It helps produce more reliable software and enables faster, more secure releases.

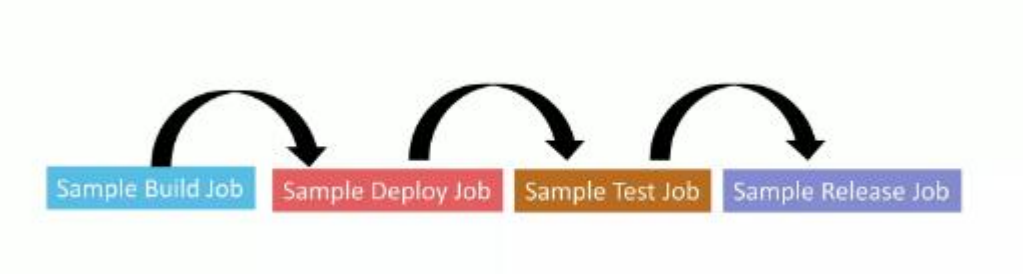
STEPS TO CREATE CI-CD PIPELINE

Step 1: Chain required jobs in sequence Add Upstream/downstream jobs

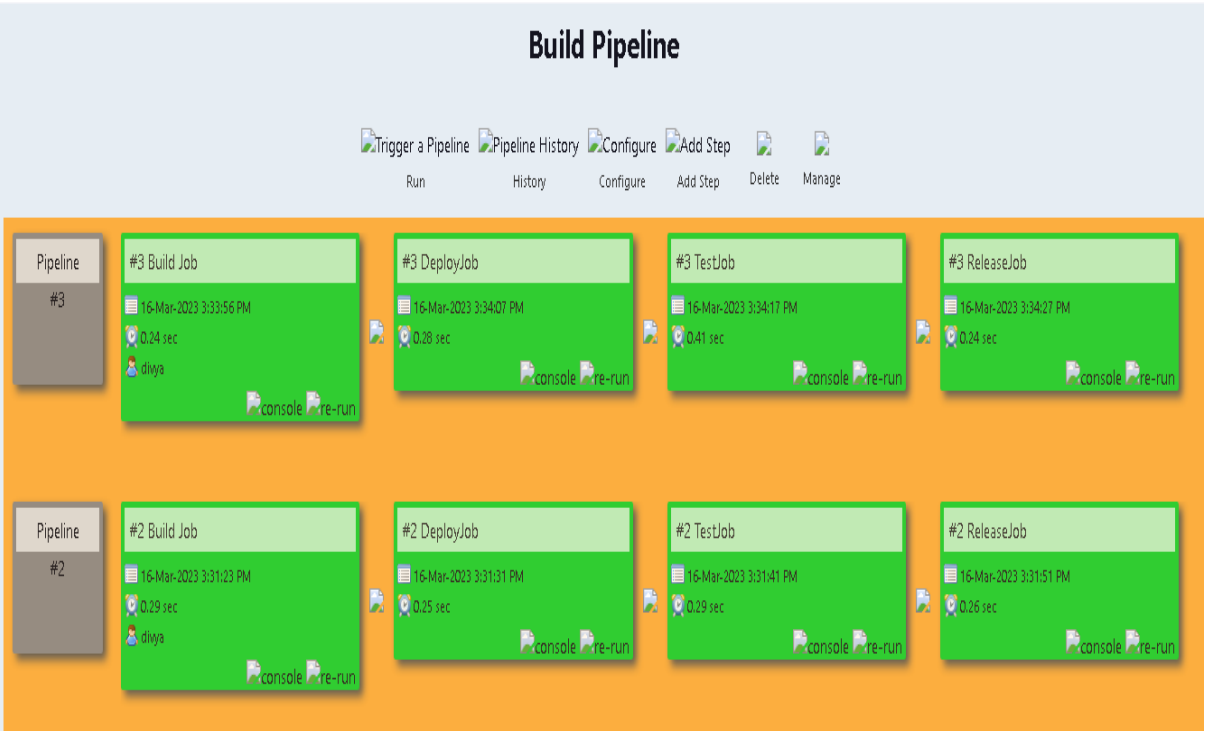
Step 2: Install Build Pipeline and Delivery Pipeline Plugin

Step 3: Add, Build Pipeline View and Configure the view step

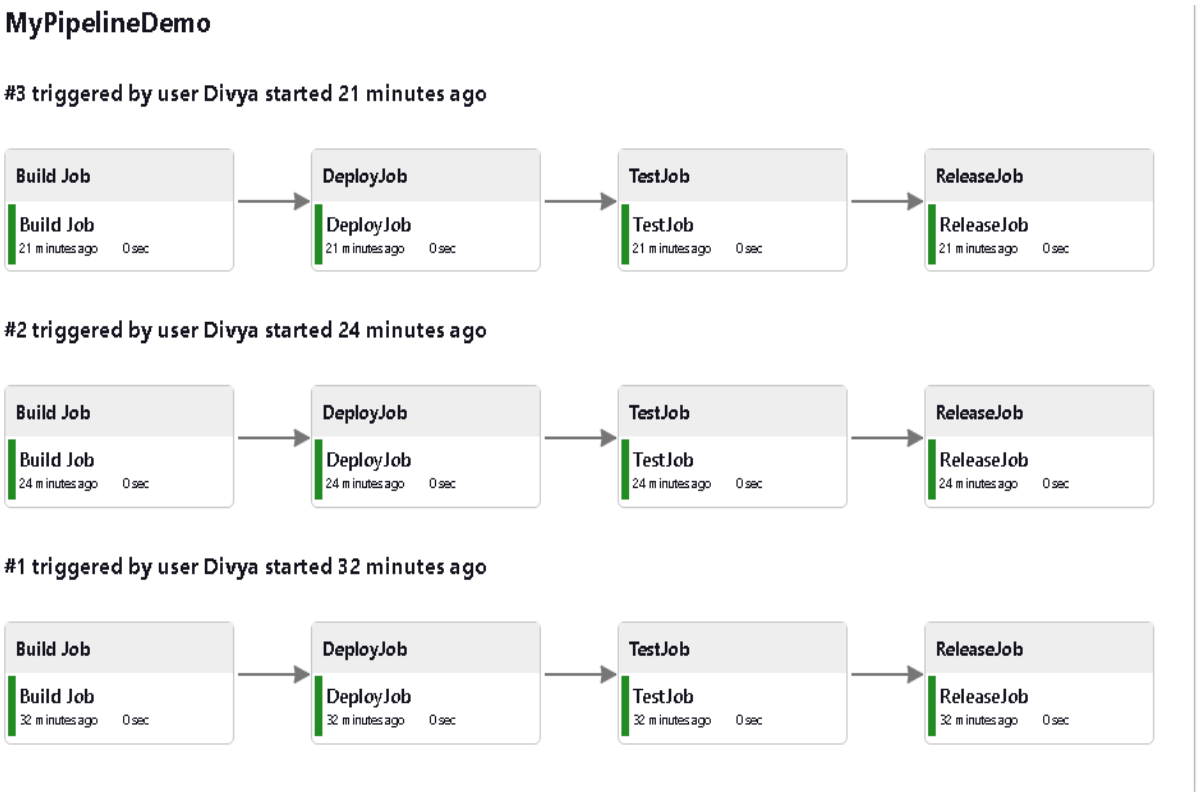
Step 4: Run and Validate



BUILD PIPELINE



DELIVERY PIPELINE



STEP BY STEP PROCEDURE:

CREATING JOBS

- Create a New Item-BuildJob

- Create a Freestyle Project -OK
- Build Steps – Execute shell
- Type the commands
- Example: date
    - **echo** “Build process Successfully done”
- Apply and Save
- Proceed for the remaining three Jobs [Deploy, Test and Release]

CREATE CHAIN

- Open each Job
- Configure -> Build Triggers -> Build after other projects is built →----------
- Apply and Save
- Open the Each job in a separate window
- Run (Build Now) the First Job
- Automatically other three jobs are executed

PLUGIN INSTALLATION:

- Manage Plugin ->Install Plugin ->
- Build Pipeline Plugin
- Delivery Pipeline Plugin
- Install without restart

CREATE PIPELINE

- Dashboard -> All -> +
- Name of the Pipeline
- Select Build Pipeline View
- Pipeline Flow -> Initial job -> Build Job
- Display option -> 5
- OK

CREATE PIPELINE DELIVERY

- All ->+
- Name -> Delivery Pipeline View
- Add Component
- Name and Initial Job