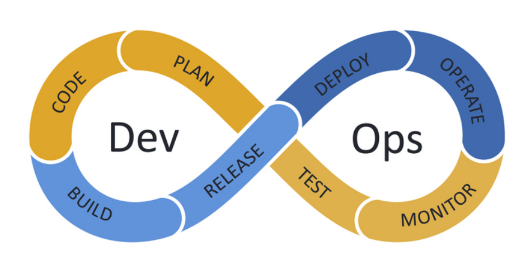
**DevOps Lifecycle**

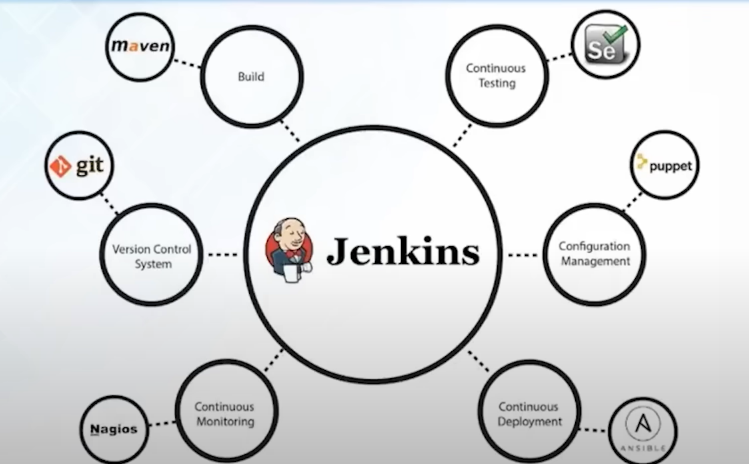
* alternate methodology for software development that increases speed, efficiency, and output.
* minimises the number of steps in the software lifecycle by integrating development and operations

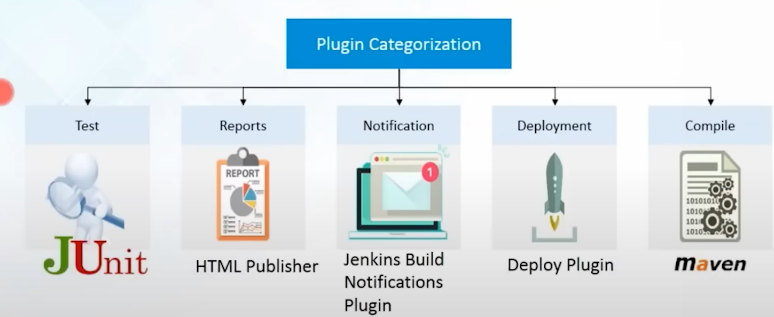


* DevOps lifecycle is a continuous software development process that employs DevOps best practices to plan, build, integrate, deploy, monitor, operate, and offer continuous feedback throughout the software’s lifecycle.

**What is Jenkins?**

* Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.
* Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.







* Easy installation and upgrade on various OSs
* Simple and user-friendly interface
* Extensible with huge community-contributed plugin resource
* Easy environment configuration in the user interface
* Supports distributed builds with master-slave architecture
* Build schedules based on expressions
* Supports shells and Windows command execution in pre-build steps
* Supports notification on the build status

**GIT BRANCHING STRATEGIES**

* A Git branching strategy is a strategy adopted by the software development team when creating, merging, and deploying code. It represents a set of rules developers can follow to determine how everyone interacts with a shared codebase.
* helps organize Git repositories and prevents application errors and merge conflicts.

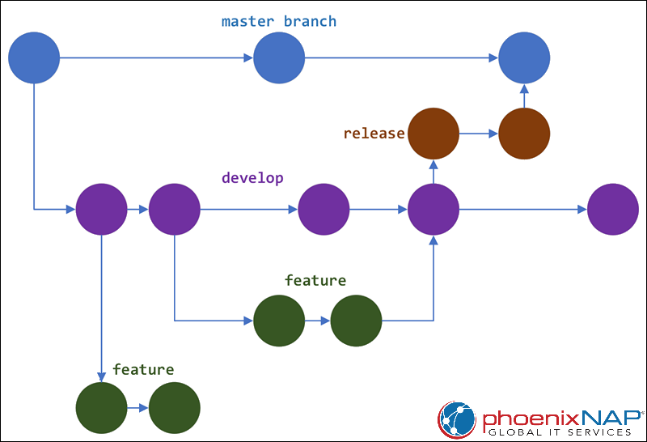
**Why Should You Use Git Branching Strategies?**

Using branching strategies facilitates collaboration on a project by preventing developers from interfering with each other's work. Therefore, without a branching strategy, the workflow slows down, which hinders an efficient DevOps process and obstructs speedy code releases.

The principal goals of a branching strategy are to:

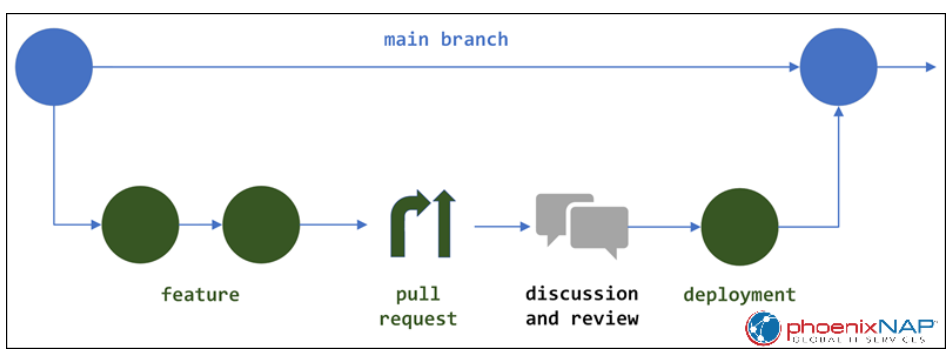
* Enhance productivity through proper coordination among developers.
* Enable simultaneous development.
* Facilitate planned, structured releases.
* Set out a clear path, from making changes to software to production.
* Maintain a bug-free code, allowing developers to quickly implement the fixes into production without disrupting the workflow.

**GitFlow**

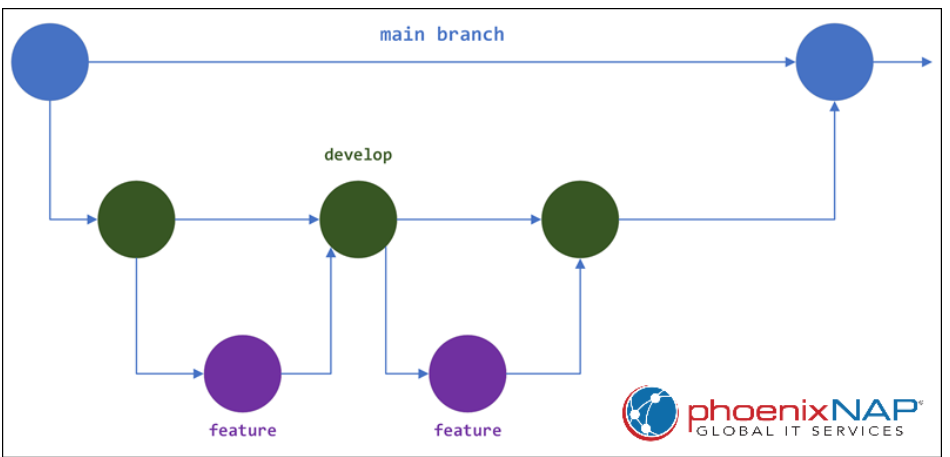


In release it create staging env.

**GitHub Flow**

****

**GITLAB**

****

* GitLab is a suite of tools for managing different aspects of the software development lifecycle. The core product is a web-based Git repository manager with features such as issue tracking, analytics, and a Wiki.
* View, create, and manage codes and project data through branching tools
* Design, develop, and manage codes and project data from a single distributed version control system, enabling rapid iteration and delivery of business values
* Provides a single source of truth and scalability for collaborating on projects and code
* Helps delivery teams fully embrace CI by automating the builds, integration, and verification of source codes
* Provides container scanning, static application security testing (SAST), dynamic application security testing (DAST), and dependency scanning to deliver secure applications along with license compliance
* Helps automate and shorten releases and delivery of applications

**Pros**

* **Robust and scalable**. GitLab Flow is a more robust and scalable Git branching strategy than GitHub Flow. It is suitable for large teams and projects.
* **Clear code separation**. GitLab Flow provides a clear separation of code under development from production-ready code. The separation helps prevent accidental changes to the production code.
* **Independent feature development**. GitLab Flow provides a separate branch for each feature, ensuring their independent development. Later, when merging the features into the main codebase, there are fewer conflicts.
* **Parallel development**. Separate branches allow developers to work simultaneously on different features. This strategy reduces the time it takes to develop new features.

**Cons**

* **Complexity**. GitLab Flow can be a complex branching strategy, especially for teams that are new to Git.
* **Possible merge conflicts**. Merging the **feature** branches into the **develop** branch can lead to merge conflicts. The reason is because the **feature** branches tend to diverge from the **develop** branch over time.
* **Slower development**. Using the GitLab Flow strategy can slow down development because it requires developers to merge their changes into the **develop** branch before release. This can be a problem for teams that need to release new features and bug fixes quickly.

**JENKINS MANAGEMNET**

**Configuration as Code**

* The Jenkins Configuration as Code (JCasC) feature defines Jenkins configuration parameters in a human-readable YAML file that can be stored as source code. This essentially captures the configuration parameters and values that are used when configuring Jenkins from the web UI.

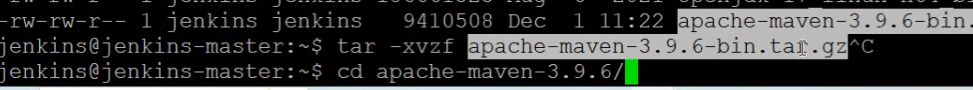
**Managing Plugins**

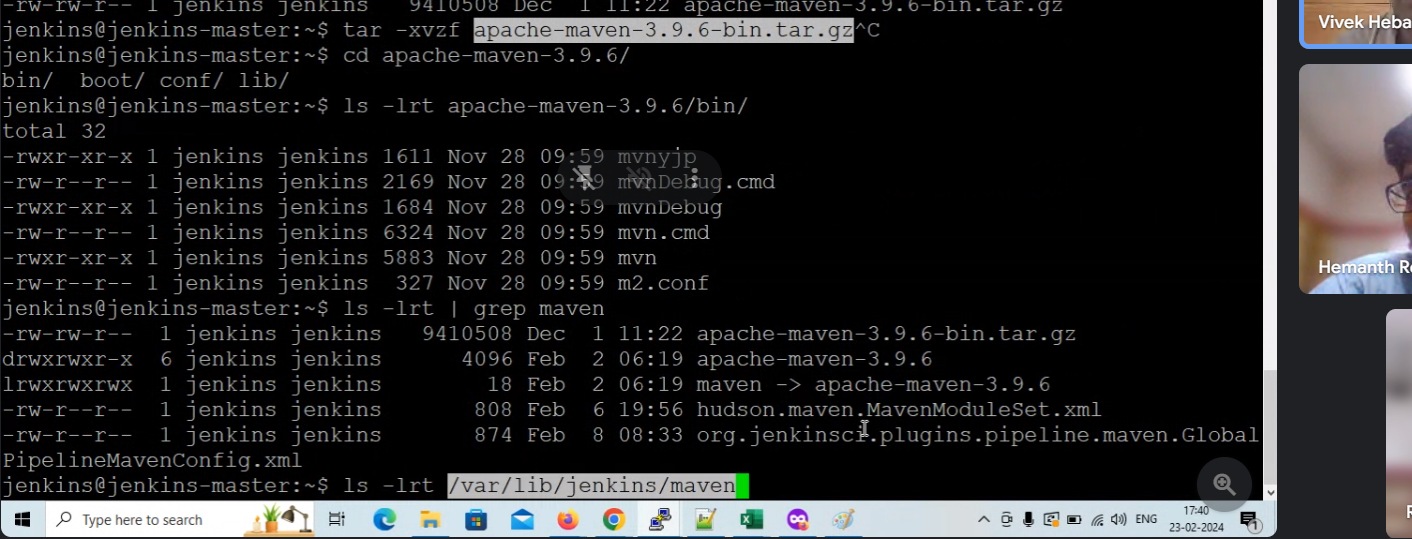
* enhancing the functionality of a Jenkins environment to suit organization- or user-specific needs.
* 1000 different plugins which can be integrated.

**Pipelines**

* a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.

<https://maven.apache.org/download.cgi>



create softlink of the tool

Create and deploy application on ec2 using Jenkins job

