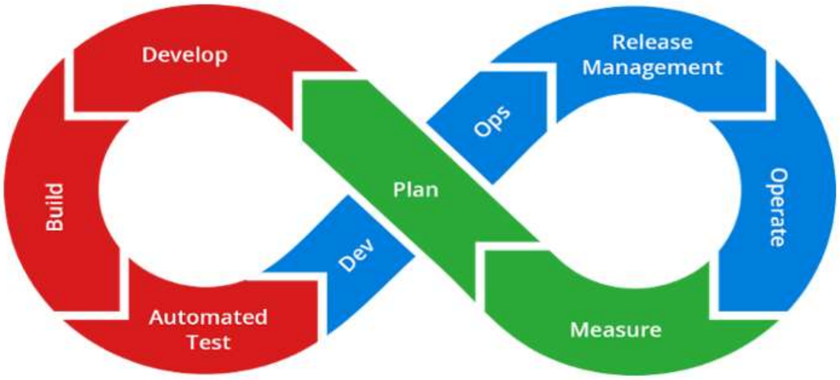
Jenkins Overview

Jenkins is an open-source automation server that is widely used for continuous integration and continuous delivery (CI/CD) of software projects. It was originally developed as a fork of the Hudson project in 2011 and has since become a popular tool in the software development community.

The primary purpose of Jenkins is to automate various aspects of the software development process, especially the integration and delivery stages. It works by coordinating the building, testing, and deployment of software, ensuring that changes made by developers are regularly integrated into the shared repository and thoroughly tested. This helps catch integration issues early on, leading to more reliable and stable software releases.



Key features of Jenkins include:

1. Continuous Integration (CI): Jenkins can automatically trigger builds whenever new code is pushed to the version control system, ensuring that the latest changes are integrated into the project's codebase and checked for build errors.

2. Continuous Delivery/Deployment (CD): Jenkins supports automating the deployment process, allowing teams to release their software more frequently and reliably.

3. Plugin Architecture: Jenkins has a vast ecosystem of plugins that extend its functionality. There are plugins for integrating with various version control systems, build tools, testing frameworks, deployment targets, and more.

4. Distributed Builds: Jenkins can distribute build and test tasks across multiple machines, which helps accelerate the process, especially for large projects.

5. Extensibility: Jenkins can be customized to suit the specific needs of a development team or organization through custom scripts, plugins, and configurations.

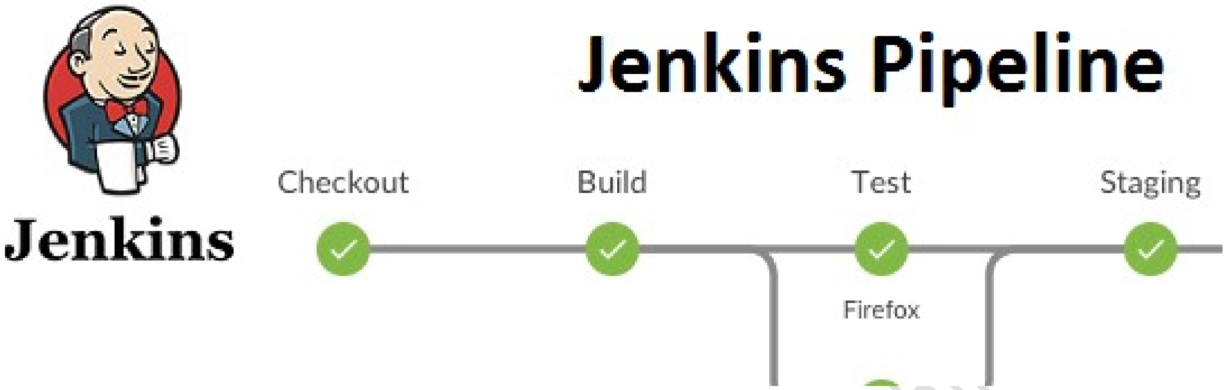
6. **Monitoring and Notifications:** Jenkins provides real-time monitoring of builds and can send notifications via email, chat applications, or other communication channels to keep the team informed about the status of their projects.

Jenkins is widely used in the software development industry because of its flexibility, ease of use, and active community support. It has become an essential tool for automating and streamlining the software development and delivery pipeline.

Jenkins offers a variety of use-cases that make it a valuable tool for software development and project management:

1. **Automation of GUI from Command Prompt:** Jenkins can convert command prompt code into GUI button clicks by wrapping the script as a Jenkins job. This saves developers from writing hundreds of lines of code, making the process more efficient.
2. **Integration of Individual Jobs:** Jenkins allows combining small, purpose-specific jobs into pipelines, enabling sequential or parallel execution, similar to the concept of Linux pipelines.
3. **Slack Integration:** Jenkins can be synchronized with communication platforms like Slack, enabling teams to share information about triggered activities, their results, and other relevant details.
4. **Effortless Auditing and Troubleshooting:** Jenkins captures console output from job runs, aiding in troubleshooting. Plugins like Time stamper help measure job run times and identify slow steps for performance tuning.
5. **Greater Data Support for Project Management:** Each project activity can be represented as a Jenkins job, allowing identification of success or failure and measuring job completion time. Jenkins supports REST API or SDK for measuring success, failure, or timing.
6. **Manual Tests Option:** Continuous Integration in Jenkins allows testing code against the current state of the codebase in a production-like environment, helping detect issues that may not surface in local environments.
7. **Increased Code Coverage:** Jenkins can check code for test coverage, promoting transparency and accountability within the team. Test results are displayed on the build pipeline, ensuring adherence to guidelines.
8. **Code Deployment to Production:** Jenkins facilitates automated deployment of code to staging or production environments when all relevant tests for a specific feature or release branch pass.
9. **Avoiding Broken Code during Shipping:** Continuous Integration in Jenkins ensures code coverage and thorough testing, preventing broken code from being shipped to production. Developers are alerted if the master build is broken.
10. **Decrease Code Review Time:** Jenkins can communicate with Version Control Systems like Git, notifying users when a merge request is ready for review and meets all testing requirements. This speeds up the code review process.

Jenkins Pipeline



Jenkins Pipeline is a powerful and extensible way to define the continuous delivery (CD) process for your software projects. It allows you to define the steps and stages involved in building, testing, and deploying your application in a structured and scriptable manner. Jenkins Pipeline is a key feature of Jenkins and has become the preferred way to create and manage complex build and deployment workflows.

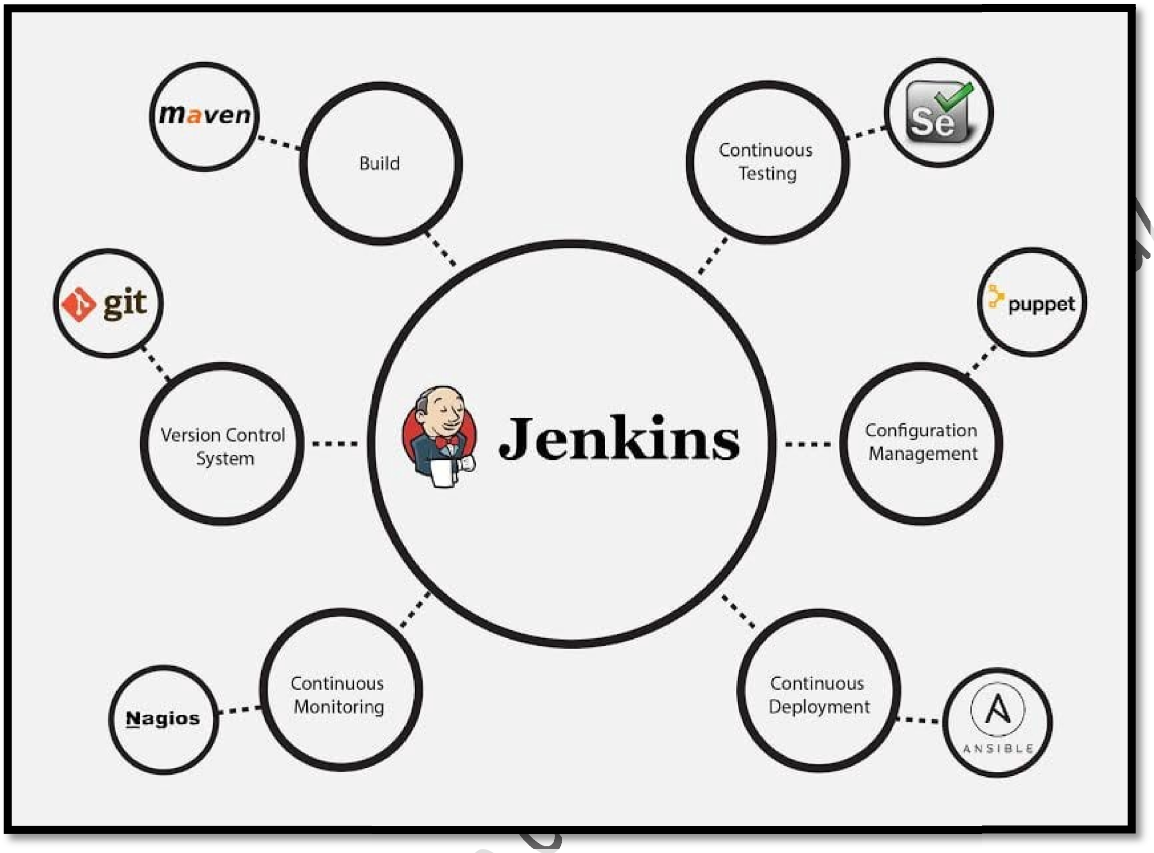
In a Jenkins Pipeline, you can define the following elements:

1. **Stages:** A stage represents a logical division within the Pipeline, such as "Build," "Test," or "Deploy." Each stage can contain one or more steps.
2. **Steps:** Steps are individual tasks that Jenkins executes during a stage. Examples of steps include compiling code, running tests, deploying artifacts, etc.
3. **Post Actions:** Post actions allow you to define tasks that should be executed after the Pipeline completes, regardless of the Pipeline's success or failure. For example, you might want to send notifications or clean up resources.
4. **Environment:** The Jenkins environment provides various variables and information about the Pipeline run that you can use in your scripts.

With Jenkins Pipeline, you can version control your entire build and deployment process, making it easier to manage and reproduce different versions of your Pipeline. Additionally, you can reuse and share Pipeline code across multiple projects, promoting consistency and best practices.

Jenkins Pipeline also integrates well with version control systems like Git, allowing you to trigger Pipeline runs automatically whenever changes are pushed to the repository. This tight integration with CI/CD practices enables teams to maintain a fast and reliable software delivery pipeline.

CONTINUOUS INTEGRATION



Continuous Integration (CI) in Jenkins refers to the process of automating the integration and testing of code changes in a continuous and automated manner. Jenkins, being a popular automation server, is widely used to implement CI workflows for software development projects.

In Jenkins, the CI process typically involves the following steps:

1. **Code Repository:** Developers work on their code changes in isolated branches within a version control system like Git.
2. **Automated Build:** Whenever a developer pushes code changes to the central repository, Jenkins is configured to automatically detect these changes. Jenkins then triggers a build process, where it retrieves the latest code from the repository and compiles it, generating executable artifacts.
3. **Automated Testing:** After the build, Jenkins runs a suite of automated tests against the newly built code. These tests can include unit tests, integration tests, functional tests, and any other relevant test cases to verify the correctness and functionality of the code.
4. **Feedback and Reporting:** Jenkins provides rapid feedback to the development team regarding the build and test results. If any issues are detected during the build or testing process, Jenkins notifies the team, indicating which tests failed and the nature of the failures.

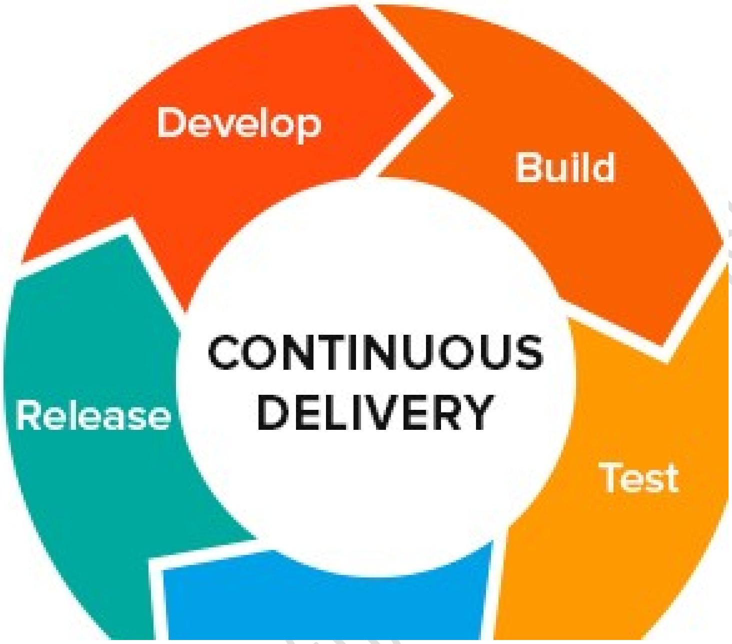
5. ***Continuous Deployment (optional):*** Depending on the project's requirements, Jenkins can be configured for Continuous Deployment (CD). If all tests pass successfully, Jenkins can automatically deploy the code to a staging environment or production server.

Continuous Integration in Jenkins ensures that code changes are continuously integrated and tested, promoting collaboration, early bug detection, and frequent releases. It helps teams deliver software more rapidly while maintaining a high level of code quality and stability.

Jenkins provides a flexible and configurable environment for setting up CI pipelines. Pipelines can be defined using Jenkins' Scripted Pipeline or Declarative Pipeline syntax, allowing developers to specify the stages, steps, and testing procedures required for their specific project.

By automating the CI process with Jenkins, development teams can streamline their software development workflows, improve code quality, and respond quickly to any issues that arise during development, thereby delivering more reliable and efficient software products.

Continuous Delivery



Continuous Delivery refers to the capability of seamlessly and safely delivering various types of changes, including new features, configuration adjustments, bug fixes, and experiments, into the production environment. This is achieved through a well-structured and efficient process that involves short work cycles. The primary objective of continuous delivery is to establish a predictable deployment mechanism, making it a routine and on-demand activity.

The key to successful continuous delivery lies in maintaining a codebase that is consistently deployable, even in scenarios with numerous developers making frequent changes. This necessitates a continuous integration process that ensures code is always in a reliable and functional state.

Continuous delivery operates on the principle of delivering code changes incessantly, ensuring high-quality results with minimal risks. The ultimate outcome is the creation of one or more artifacts that are ready to be deployed to the production environment. This approach facilitates a smooth and reliable software delivery process, allowing teams to swiftly respond to evolving requirements while maintaining the integrity of the application.

CONTINUOUS DEPLOYMENT (CD)

Continuous Deployment, also known as continuous implementation, represents an advanced stage of continuous delivery, extending the automation process beyond the delivery phase. In this methodology, every change that successfully passes automated testing is automatically deployed to the production environment.

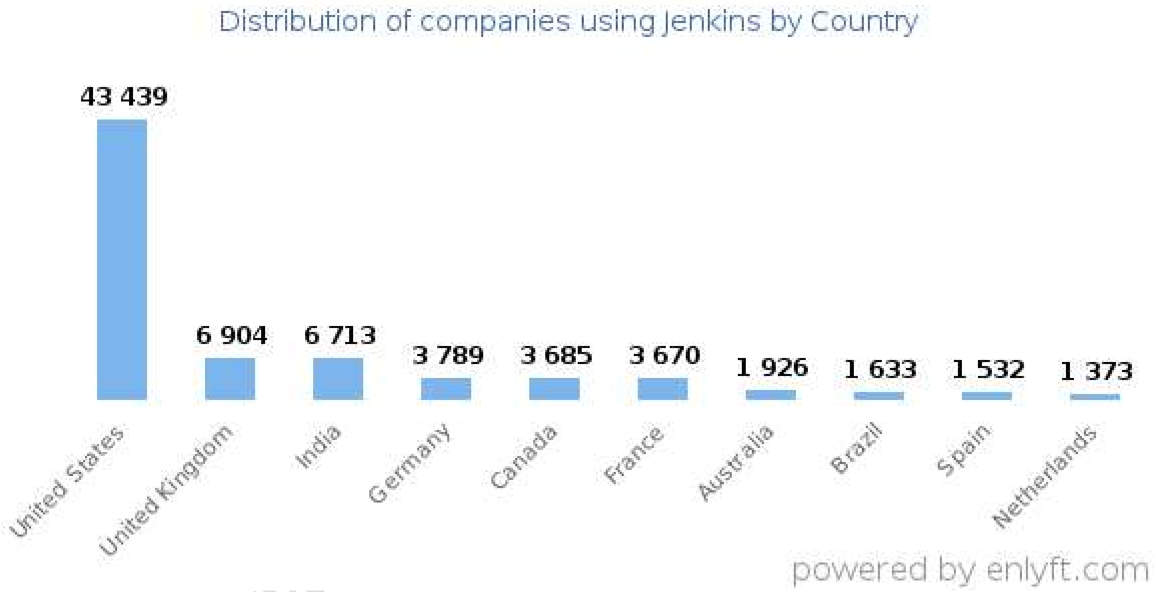
A key principle in continuous deployment is the fail-fast strategy, which emphasizes identifying potential issues as early as possible in the production deployment process. By deploying every change to production, edge cases and unexpected behaviors can be swiftly detected, which might be challenging to identify solely through automated tests.

To fully leverage the benefits of continuous deployment, a robust logging technology becomes crucial in tracking and identifying any increase in errors with newer versions. Additionally, a reliable orchestration technology, such as Kubernetes, allows for a gradual rollout of the new version to users, enabling monitoring for any incidents, and providing the capability to abort the deployment if necessary. This ensures a smooth and controlled deployment process, promoting greater confidence in continuous delivery practices.

Automation

Jenkins, as a job executor, offers automation for repetitive tasks such as database backup/restore, machine management, and service statistics collection. These tasks can be scheduled at desired intervals, such as daily, weekly, or monthly, streamlining operations and reducing manual effort. With Jenkins' flexible scheduling capabilities, it efficiently automates routine activities, enhancing productivity and reliability in software development and system management.

Top Countries that use Jenkins.


Distribution of companies using Jenkins by Country
United States
United Kingdom
India
Germany
Canada
France
Australia
Brazil
Spain
Netherlands
powered by enlyft.com