UNIT – II
DevOps Tools and Technologies

Code Repositories : Git, Differences between SVN and Git, Build tools – Maven, Continuous integration tools – Jenkins, Container Technology – Docker, Monitoring Tools – Zenoss, Continuous integration with Jenkins 2, Creating built-in delivery pipelines, Creating Scripts, Creating a pipeline for compiling and executing test units, Using the Build Pipeline plugin, Integrating the deployment operation, Getting started with Chef, Overview of hosted Chef, Installing and configuring a Chef workstation.

What is Git?

Git is a distributed version control system, which means that a local clone of the project is a complete version control repository. These fully functional local repositories make it easy to work offline or remotely. Developers commit their work locally, and then sync their copy of the repository with the copy on the server. This paradigm differs from centralized version control where clients must synchronize code with a server before creating new versions of code.

Git's flexibility and popularity make it a great choice for any team. Many developers and college graduates already know how to use Git. Git's user community has created resources to train developers and Git's popularity make it easy to get help when needed. Nearly every development environment has Git support and Git command line tools implemented on every major operating system.

Git basics

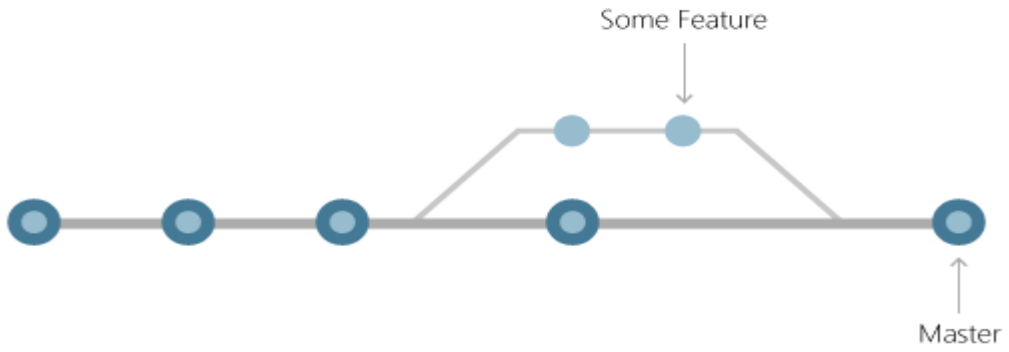
Every time work is saved, Git creates a commit. A commit is a snapshot of all files at a point in time. If a file hasn't changed from one commit to the next, Git uses the previously stored file. This design differs from other systems that store an initial version of a file and keep a record of deltas over time.

Commits create links to other commits, forming a graph of the development history. It's possible to revert code to a previous commit, inspect how files changed from one commit to the next, and review information such as where and when changes were made. Commits are identified in Git by a unique cryptographic hash of the contents of the commit. Because everything is hashed, it's impossible to make changes, lose information, or corrupt files without Git detecting it.

Branches

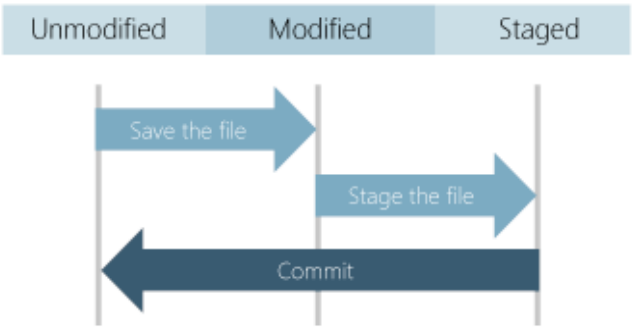
Each developer saves changes to their own local code repository. As a result, there can be many different changes based off the same commit. Git provides tools for isolating changes and later merging them back together. Branches, which are lightweight pointers to work in progress,

manage this separation. Once work created in a branch is finished, it can be merged back into the team's main (or trunk) branch.



Files and commits

Files in Git are in one of three states: modified, staged, or committed. When a file is first modified, the changes exist only in the working directory. They aren't yet part of a commit or the development history. The developer must *stage* the changed files to be included in the commit. The staging area contains all changes to include in the next commit. Once the developer is happy with the staged files, the files are packaged as a *commit* with a message describing what changed. This commit becomes part of the development history.



Staging lets developers pick which file changes to save in a commit in order to break down large changes into a series of smaller commits. By reducing the scope of commits, it's easier to review the commit history to find specific file changes.

Benefits of Git

The benefits of Git are many.

Simultaneous development

Everyone has their own local copy of code and can work simultaneously on their own branches. Git works offline since almost every operation is local.

Faster releases

Branches allow for flexible and simultaneous development. The main branch contains stable, high-quality code from which you release. Feature branches contain work in progress, which are merged into the main branch upon completion. By separating the release branch from development in progress, it's easier to manage stable code and ship updates more quickly.

Built-in integration

Due to its popularity, Git integrates into most tools and products. Every major IDE has built-in Git support, and many tools support continuous integration, continuous deployment, automated testing, work item tracking, metrics, and reporting feature integration with Git. This integration simplifies the day-to-day workflow.

Strong community support

Git is open-source and has become the de facto standard for version control. There is no shortage of tools and resources available for teams to leverage. The volume of community support for Git compared to other version control systems makes it easy to get help when needed.

Git works with any team

Using Git with a source code management tool increases a team's productivity by encouraging collaboration, enforcing policies, automating processes, and improving visibility and traceability of work. The team can settle on individual tools for version control, work item tracking, and continuous integration and deployment. Or, they can choose a solution like GitHub or Azure DevOps that supports all of these tasks in one place.

Pull requests

Use pull requests to discuss code changes with the team before merging them into the main branch. The discussions in pull requests are invaluable to ensuring code quality and increase knowledge across your team. Platforms like GitHub and Azure DevOps offer a rich pull request experience where developers can browse file changes, leave comments, inspect commits, view builds, and vote to approve the code

Difference Between GIT and SVN

In version control systems (VCS), Git and SVN (Subversion) are two of the most widely used tools. Both systems help developers manage code changes, collaborate with team members, and maintain the history of their projects. However, there are fundamental differences in their design, workflows, and features. This article will explore these differences, helping you understand which system might be the best fit for your needs.

GIT

Git is an open-source distributed version control system developed by Linus Torvalds in 2005. Its emphasis on speed and data integrity in which there is no centralized connectivity is needed. It is powerful and cheap branching with easy merge in which each developer has his repository and has a local copy in which they can change history. It supports non-linear development branches and applications with a large number of code files.

Here are some .git directory structures used in GIT:

- **HEAD/:** A pointer structure used in git.
- **Config/:** Contains all configuration preferences.
- **description/:** Description of your project.
- **index/:** It is used as a staging area between the working directory.
- **object/:** All the data are stored here.
- **logs/:** Keeps record of changes that are made.

Features of GIT

1. **Distributed Version Control**: Each developer has a full copy of the repository, including its entire history. This allows for offline work and decentralized collaboration.

2. **Efficient Branching and Merging**: Git makes it easy and fast to create, merge, and delete branches. This supports workflows like feature branching and pull requests.
3. **Speed and Performance**: Local operations are very fast since they don't require network access. Commands like commit, diff, and log execute quickly.
4. **Staging Area (Index)**: Git has a staging area that allows you to prepare changes before committing. This provides granular control over what changes are included in a commit.
5. **Strong Community and Ecosystem**: Git is widely supported with numerous tools, integrations, and platforms like GitHub, GitLab, and Bitbucket, enhancing its functionality and usability.

SVN

Apache Subversion is an open-source software version and revision control system under the Apache license. It managed files and folders that are present in the repository. It can operate across the network, which allows it and used by people on different computer .we can say that a repository is like an ordinary file server which allows it to be used by people on a different computer.

Features of SVN

1. **Centralized Version Control**: All files and their historical versions are stored in a central repository. This ensures all team members are always working with the latest version.
2. **Atomic Commits**: SVN ensures that commits are atomic, meaning either all changes in a commit are applied, or none are. This prevents partial changes from causing inconsistencies.
3. **Directory Versioning**: SVN can track changes to directories, renames, and file metadata, not just file content. This provides a more comprehensive versioning system.
4. **Efficient Handling of Binary Files**: SVN handles binary files efficiently, which can be beneficial for projects that include non-text files like images or compiled libraries.
5. **Access Control**: SVN offers robust access control mechanisms at the repository, directory, and file levels, allowing for fine-grained permissions management.

Difference between GIT and SVN:

GIT	SVN
Git is open source distributed vice control system developed by Linus Torvalds in 2005. It emphasis on speed and data integrity	Apache Subversion is an open source software version and revision control system under Apache license.
Git has a Distributed Model.	SVN has a Centralized Model.
In git every user has their own copy of code on their local like their own branch.	In SVN there is central repository has working copy that also make changes and committed in central repository.
In git we do not required any Network to perform git operation.	In SVN we required Network for runs the SVN operation.
Git is more difficult to learn. It has more concepts and commands.	SVN is much easier to learn as compared to git.
Git deals with large number of files like binary files that change quickly that why it become slow.	SVN control the large number of binary files easily.

GIT	SVN
In git we create only .git directory.	In SVN also we create only one .svn directory. From one version onwards, SVN only create one .svn directory, too.
It does not have good UI as compared to SVN.	SVN has simple and better user interface

MAVEN

Build Management :

- Build Management refers to the process of converting raw source code into a distributable package after being tested and validated.
- Maven is one of Build Management tools
- Technically, Build refers to syntax check in source code.

Advantages of Build tool :

- Purpose – to build the code.
- Builds are done in SDLC to identify bugs at early stage of life cycle
- Eliminate human errors

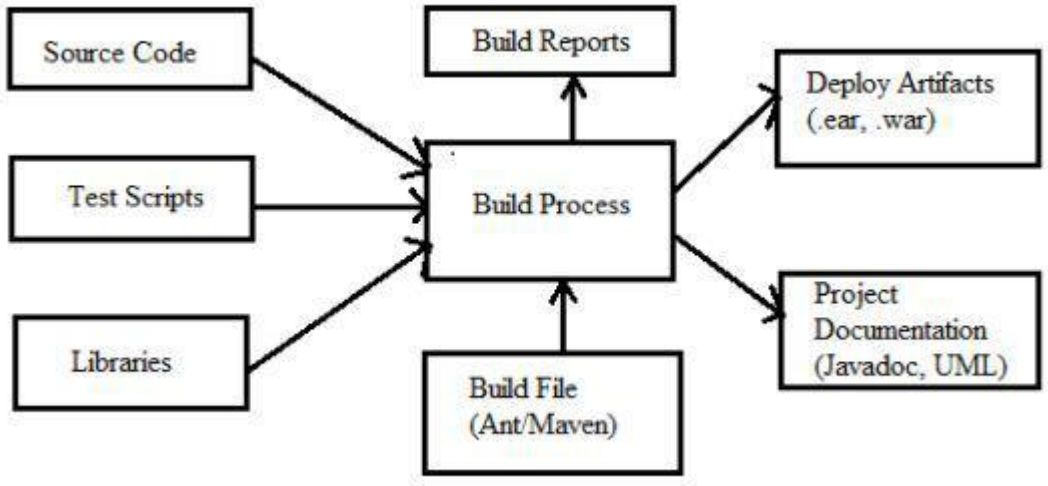
Maven build tool :

- Apache maven is an advanced project management tool for java software projects which is based on POM (project object model).
- It uses POM (project object model) file to manage project's build, dependency and documentation.
- The most powerful features of maven are to download the project dependency libraries automatically and to create the right project structure.
- Introduced in 2002 by Apache
- Open source
- Can build any java framework
- Important configuration file is : pom.xml

Features of Maven :

- Open source
- Generate document
- Generate reports
- Project management tool
- Follows POM Model
- Does builds and runs test cases

Architecture of maven :



Maven – Installation, Configuration, verification :

1. visit official site of Maven, web site : http://maven.apache.org

2. go to downloads, download binary zip - apache-maven-3.6.3-bin.zip

3. Unzip it to c:\programfiles\

4. Need to create Configure environment variables

- go to system properties, go to advanced tab,

- Under system variables. Please create below -

1. MAVEN_HOME C:\Program Files\apache-maven-3.6.3\bin

2. M2_HOME C:\Program Files\apache-maven-3.6.3\bin

6. Then open command prompt, and verify using below command

Follow below steps to configure Maven in Jenkins

1. Open Jenkins

2. open Manage Jenkins --> open " Global Tool configuration"

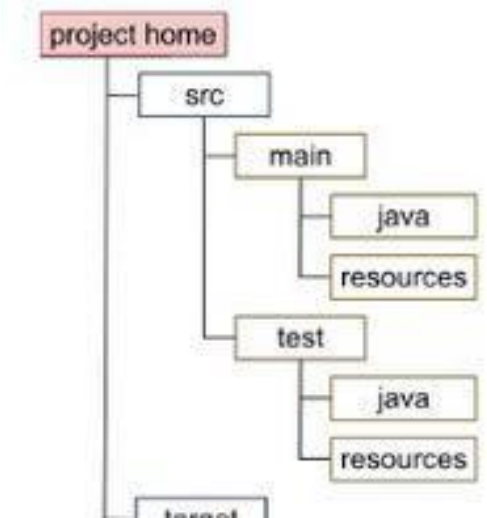
- need to configure maven

MAVEN_HOME

Maven Life-Cycle Phases :

Phase	Description
validate	Validate the project is correct and all necessary information is available.
compile	It compiles the source code of the project.
Test	Tests the compiled source code using a suitable testing framework.
package	This phase take the compiled code and creates the JAR/WAR package as mentioned in the packaging in POM.xml.
install	This phase installs the package in local maven repository.
Deploy	This phase copies the final package to the remote repository.

Maven Project Structure :



Maven repositories:

- Maven repositories are directories of packaged JAR files with extra meta-data.
- The meta-data is represented by POM files.
- A repository contains all the project jars, library jar, plugins and any other project specific artifacts.

Types of maven repository:

1. **Local Repository**
2. **Central Repository**
3. **Remote Repository**

1. Maven Local Repository:

- Maven local repository is a directory on the developer's machine.
- It gets created when we run any maven command for the first time.
- t contains all the dependencies (downloaded by maven) like library jars, plugin jars etc.
- Default location of maven local repository is user-home/.m2 directory.
- We can change the default location of maven local repository by changing the settings.xml file. It is located in MAVEN_HOME/conf/settings.xml.

```
<settings>

   <localRepository>

      //Set desired location

   </localRepository>

</settings>
```

2. Maven Central Repository:

- Maven central repository is created by the apache maven community itself.
- It contains a lot of commonly used libraries.

- By default Maven looks in this central repository for any dependencies needed but not found in your local repository.

Maven central repository path: http://repo1.maven.org/maven2/.

3. Maven Remote Repository:

- Maven remote repository is a repository on a web server.
- A remote repository can be located anywhere on the internet or inside a local network.
- We can configure a remote repository in the POM file.
- We have to put the following XML elements right after the element:

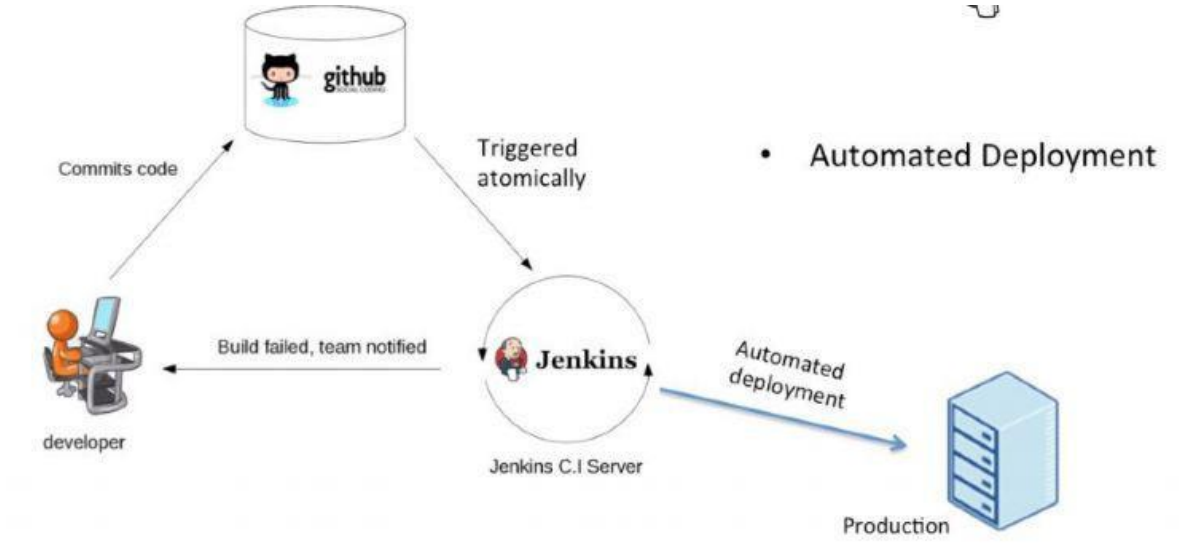
```
<repositories>

  <repository>

    <id>codesjava.code</id>

    <url>https:
//maven.codesjava.com/maven2/lib</url>
  </repository>

</repositories>
```

JENKINS

Continuous Integration(CI):

2. Continuous integration (CI) is a software engineering practice where, the fresh changes are immediately tested, validated and reported , then they are integrated to main branch
3. In traditional software development process integration is generally done at the end of day when every developer has completed their work as a result of this integration took lots of time, however CI is done as soon as code is developed.

Jenkins workflow:



Ways of CI:

- Maintain a code repository.
- Automate your build.
- Daily commits to the baseline by everyone on the team.
- Every commit (to the baseline) should be build.

Benefits of CI:

3. Faster product release
4. Reduce repetitive manual processes
5. Enable better project visibility
6. Reduction in bugs count
7. Good quality code

Why only jenkins:

4. Easy to use
5. Great extensibility
6. Support different version control systems
7. Build notifiers
8. UI customization

Java installation and configuration:

Step 1: Download JDK

- Goto Java SE download site @ http://www.oracle.com/technetwork/java/javase/downloads/index. html .
- Download the JDK 1.8 Version

Step 2 : Install JDK on Computer

Step 3: set Path, Include JDK's "bin" Directory in the PATH

You need to include JDK's "bin" in the PATH to run the JDK programs.

To edit the PATH environment variable in Windows :

1. Launch "Control Panel" ⇒ (Optional) "System and Security" ⇒ "System" ⇒ Click " dvanced system settings" on the left pane.
2. Switch to "Advanced" tab ⇒ Click "Environment Variables" button.
3. Under "System Variables"
4. Add PATH and JAVA_HOME information

Step 4: Verify the JDK Installation

Launch a C D via one of the following means:

1. Click "Search" button ⇒ Type "cmd" ⇒ Choose "Command Prompt", or
2. Right-click the "Start" button ⇒ run... ⇒ enter "cmd", or
3. Click "Start" button ⇒ Windows System ⇒ Command Prompt
4. Use java --version to check java version

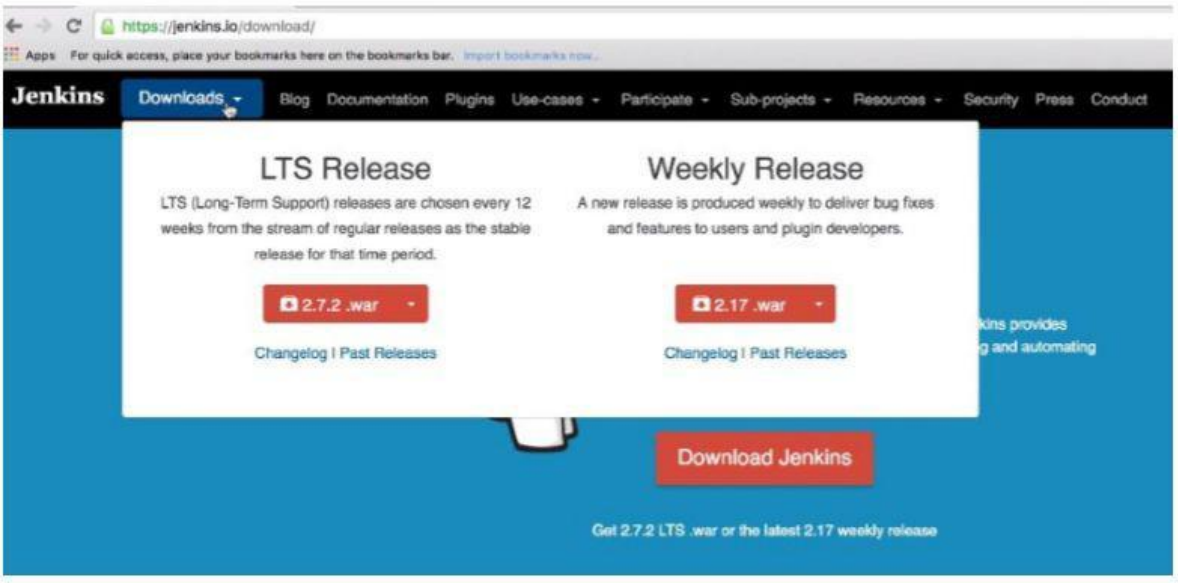
Jenkins installation and configuration:

Step 1 - Open Official Page of Jenkins, https://jenkins.io

Step 2 - Click on downloads

Step 3 - Make sure to download, Long-Term Support version of Jenkins

● Download the LTS version



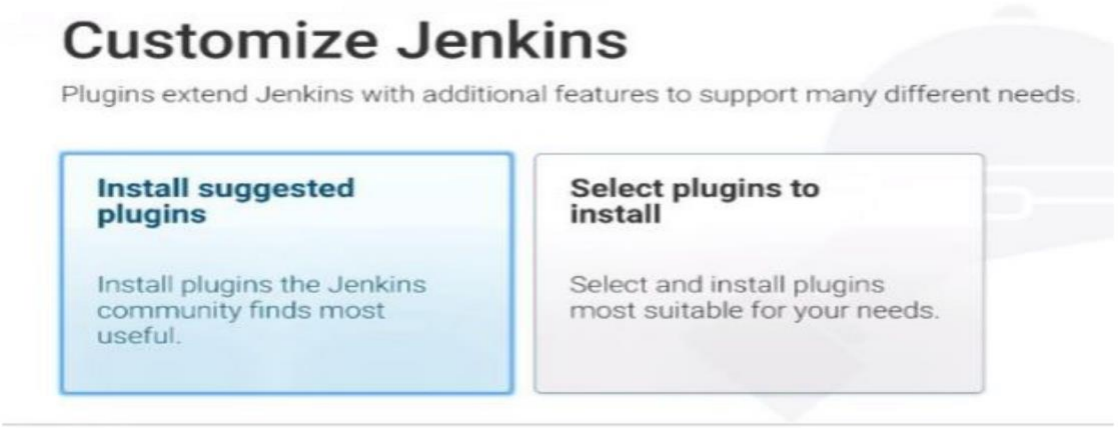
Step 4 - Preferable to download " Generic Java package (.war)"

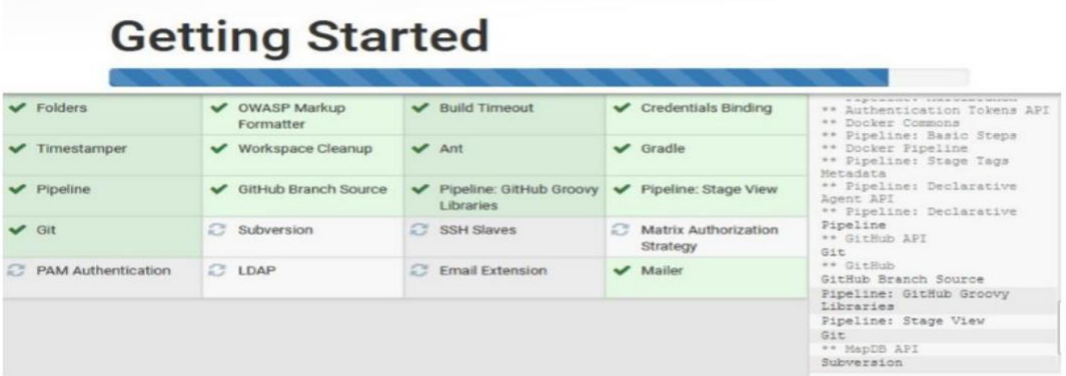
Note : WAR file:

The Web application ARchive (WA) file version of Jenkins can be installed on any operating system or platform that supports Java. To download and run the WAR file version of Jenkins:

1. Download the latest stable Jenkins WAR file to an appropriate directory on your machine.
2. Open up a terminal/command prompt window to the download directory.
3. Run the command java -jar jenkins.war.

4. Browse to http://localhost:8080 and wait until the Unlock Jenkins page appears.
5. Continue on with the Post-installation setup wizard below.





Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:

Instance Configuration

Jenkins URL: http://192.168.1.23:8080

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Getting Started

Jenkins is ready!

Your Jenkins setup is complete.

Start using Jenkins

Free style project:

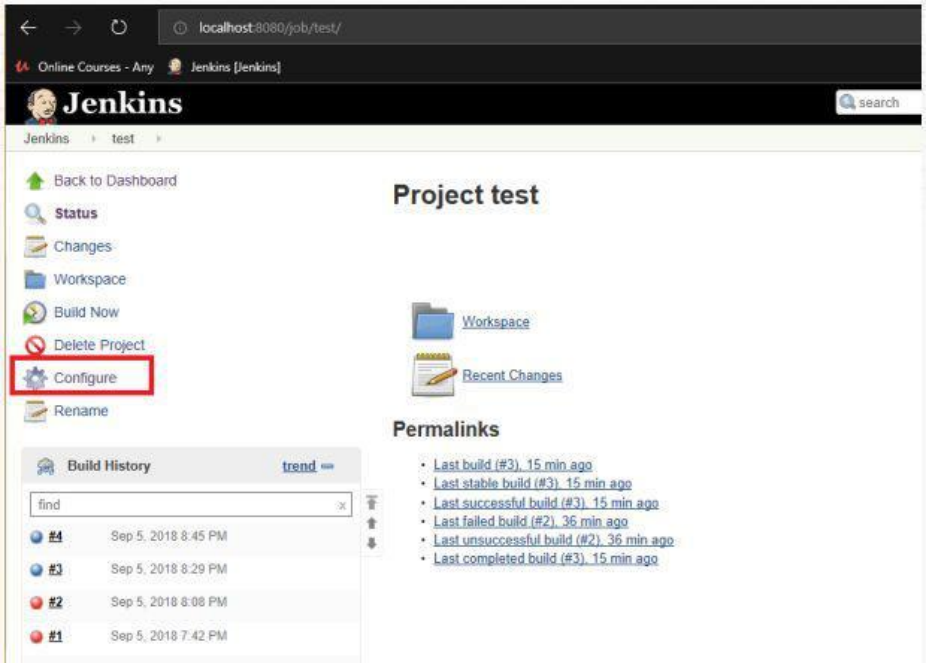
- A freestyle project in Jenkins is a project that spans multiple operations. It can be a build, a script run, or even a pipeline.
- A freestyle project is a typical build job or task. This could be as simple as running tests, building or packaging an application, sending a report, or even running some commands.

Jenkins plugins:

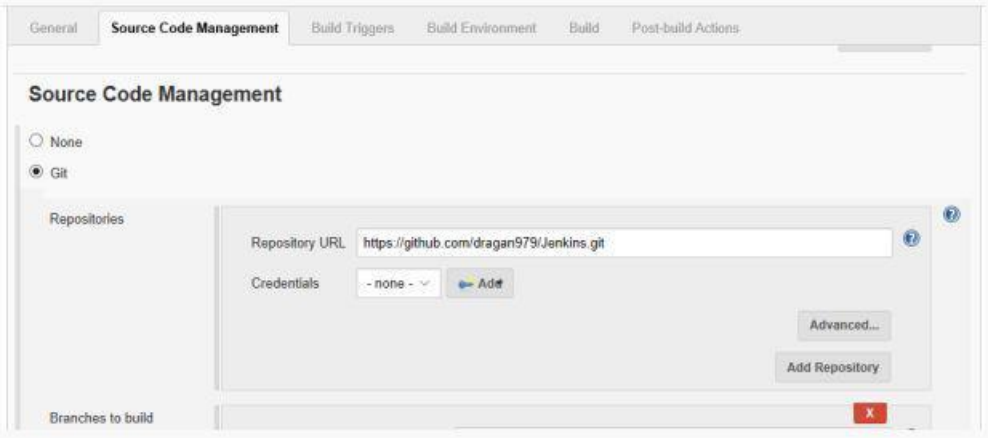
- Plugins are the primary means of enhancing the functionality of a Jenkins environment to suit organization- or user-specific needs.
- The 1,400 plugins encompass source code management, administration, platforms, UI/UX, building management, and much more.
- There are over a thousand different pluginswhich can be installed on a Jenkins master and to integrate various build tools, cloud providers, analysis tools, and much more.
- Plugins can be automatically downloaded, with their dependencies, from the Update Center.

Source code polling(GIT):

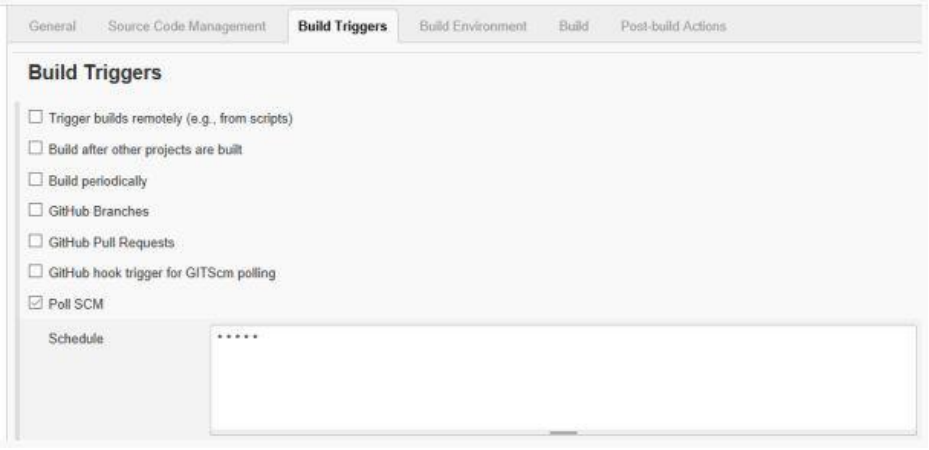
- Under project click Configure



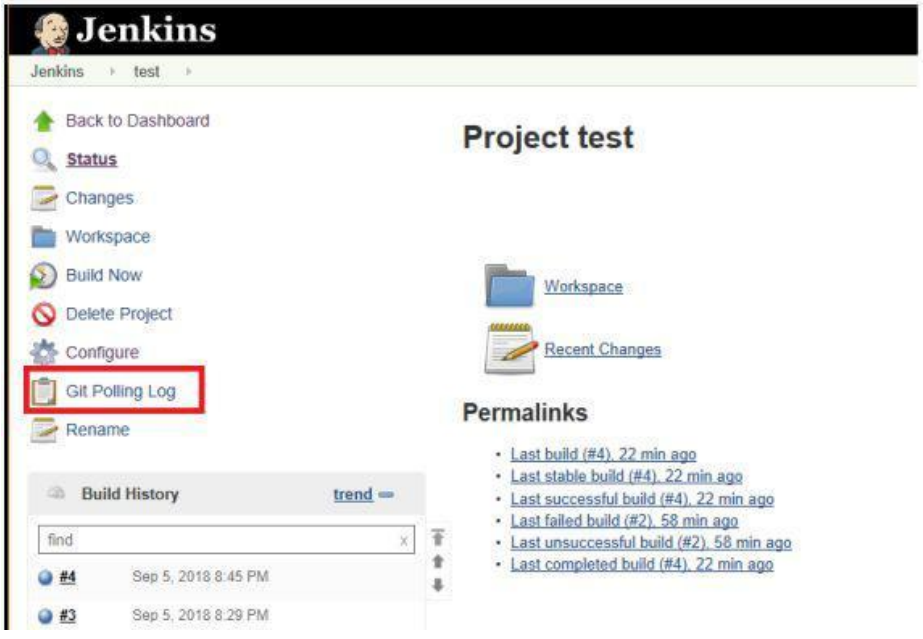
- Under Source Code Management click Git-paste Git Repo URL



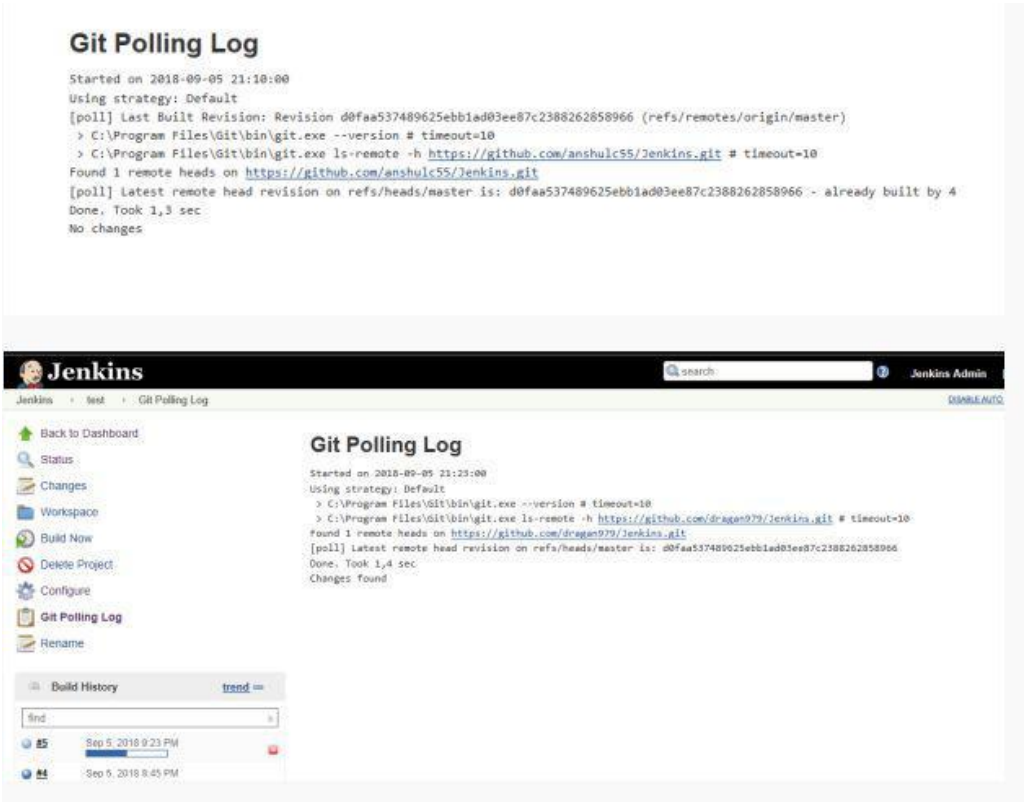
- Under Build Triggers click Poll SCM and specify time interval for checking GitHub repository for changes



- Click on Git Polling log to see changes



- In case i changed something in code, project build will start/

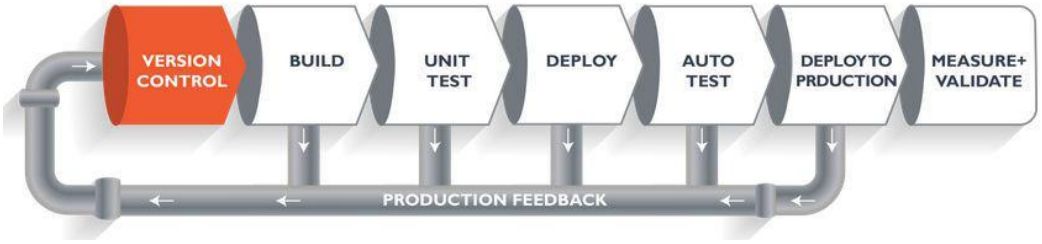


Upstream and Downstream projects:

- An upstream job is a configured project that triggers a project as of its execution.
- A downstream job is a configured project that is triggered as part of a execution of a pipeline.
- We can configure the actual job not to be triggered if the upstream job is failed.
- In the same way, we can configure the downstream job not to be triggered if the actual job is failed.
- Upstream and downstream jobs help you to configure the sequence of execution for different operations and hence you can orchestrate the flow of execution.
- We can configure one or more projects as downstream jobs in Jenkins.

CI-CD pipeline:

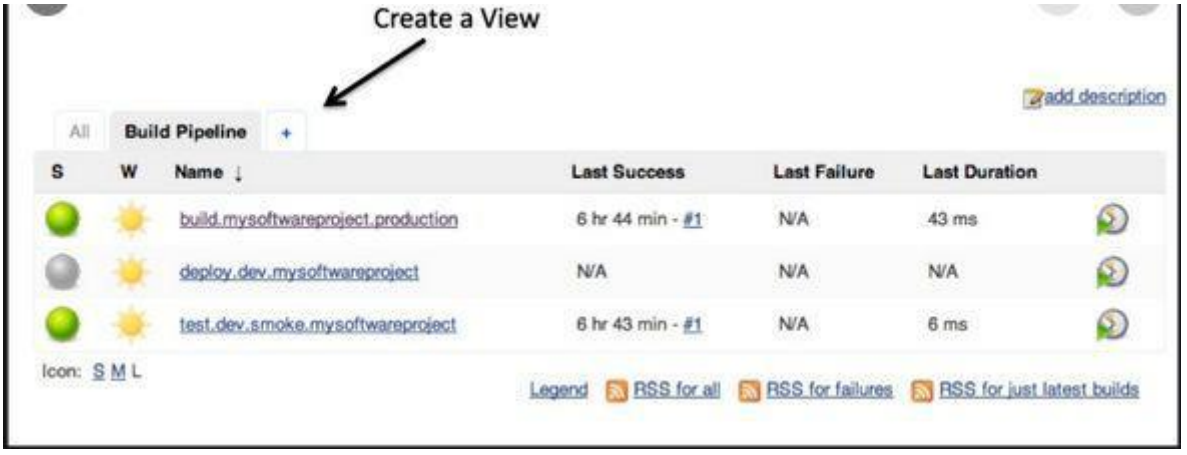
- It bridges the gap between development and operations teams by automating the building, testing, and deployment of applications.



- The above pipeline is a logical demonstration of how software will move along the various stages in this lifecycle before it is delivered to the customer or before it is live in production.
- A CI/CD pipeline helps you automate steps in your software delivery process, such as initiating code builds, running automated tests, and deploying to a staging or production environment.
- Automated pipelines remove manual errors, provide standardized development feedback loops and enable fast product iterations.

Jenkins views:

- Views in Jenkins allow us to organize jobs and content into tabbed categories, which are displayed on the main dashboard.



- Clicking on the + tab, jenkins will navigate us to the basic view configuration page. From this page we will need to specify a name and view type for our new view.
- s a Jenkins instance expands, it is logical to create associated views for appropriate groups and categories.
- The default view type available in Jenkins is the List view.

Types of Jenkins plugins and its dependencies

Jenkins plugins are packages of software that extend the functionality of the Jenkins automation server. Plugins allow you to integrate Jenkins with various tools, technologies, and workflows, and can be easily installed and configured through the Jenkins web interface.

Some popular Jenkins plugins include:

Git Plugin: This plugin integrates Jenkins with Git version control system, allowing you to pull code changes, build and test them, and deploy the code to production.

Maven Plugin: This plugin integrates Jenkins with Apache Maven, a build automation tool commonly used in Java projects.

Amazon Web Services (AWS) Plugin: This plugin allows you to integrate Jenkins with Amazon Web Services (AWS), making it easier to run builds, tests, and deployments on AWS infrastructure.

Slack Plugin: This plugin integrates Jenkins with Slack, allowing you to receive notifications about build status, failures, and other important events in your Slack channels.

Blue Ocean Plugin: This plugin provides a new and modern user interface for Jenkins, making it easier to use and navigate.

Pipeline Plugin: This plugin provides a simple way to define and manage complex CI/CD pipelines in Jenkins.

Jenkins plugins are easy to install and can be managed through the Jenkins web interface. There are hundreds of plugins available, covering a wide range of tools, technologies, and use cases, so you can easily find the plugins that best meet your needs.

By using plugins, you can greatly improve the efficiency and automation of your software development process, and make it easier to integrate Jenkins with the tools and workflows you use.

Git Plugin

The Git Plugin is a popular plugin for Jenkins that integrates the Jenkins automation server with the Git version control system. This plugin allows you to pull code changes from a Git repository, build and test the code, and deploy it to production.

With the Git Plugin, you can configure Jenkins to automatically build and test your code whenever changes are pushed to the Git repository. You can also configure it to build and test code on a schedule, such as once a day or once a week.

The Git Plugin provides a number of features for managing code changes, including:

Branch and Tag builds: You can configure Jenkins to build specific branches or tags from your Git repository.

Pull Requests: You can configure Jenkins to build and test pull requests from your Git repository, allowing you to validate code changes before merging them into the main branch.

Build Triggers: You can configure Jenkins to build and test code changes whenever changes are pushed to the Git repository or on a schedule.

Code Quality Metrics: The Git Plugin integrates with tools such as SonarQube to provide code quality metrics, allowing you to track and improve the quality of your code over time.

Notification and Reporting: The Git Plugin provides notifications and reports on build status, failures, and other important events. You can configure Jenkins to send notifications via email, Slack, or other communication channels.

By using the Git Plugin, you can streamline your software development process and make it easier to manage code changes and collaborate with other developers on your team.

file system layout

In DevOps, the file system layout refers to the organization and structure of files and directories on the systems and servers used for software development and deployment. A well-designed file system layout is critical for efficient and reliable operations in a DevOps environment.

Here are some common elements of a file system layout in DevOps

Code Repository: A central code repository, such as Git, is used to store and manage source code, configuration files, and other artifacts.

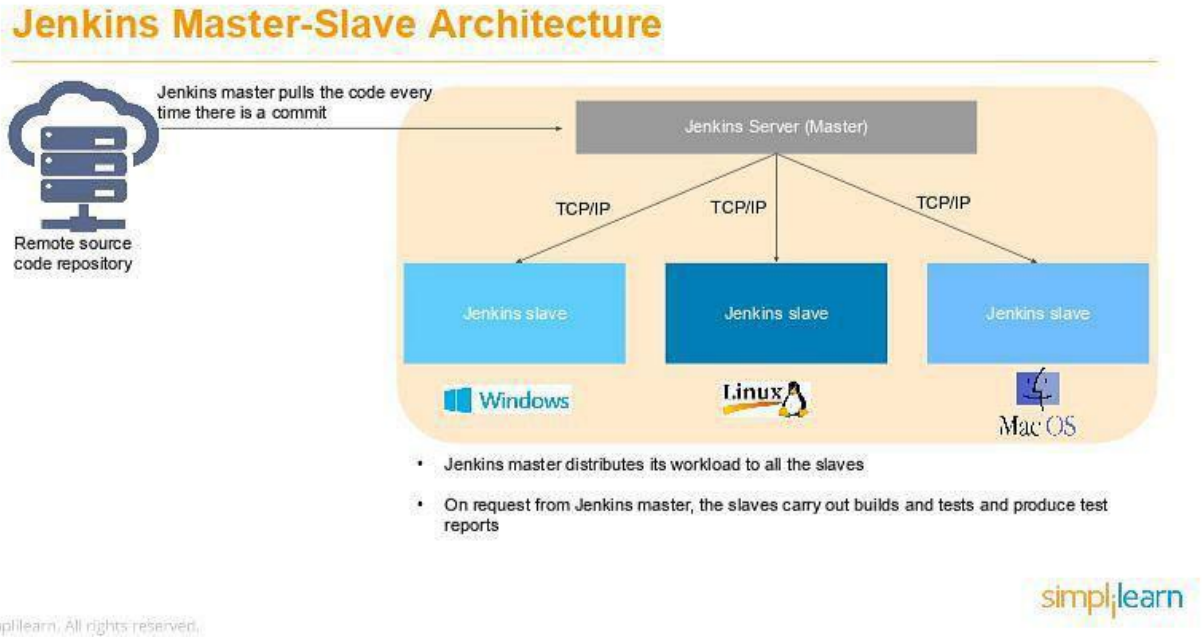
Build Artifacts: Build artifacts, such as compiled code, are stored in a designated directory for easy access and management.

Dependencies: Directories for storing dependencies, such as libraries and tools, are designated for easy management and version control.

Configuration Files: Configuration files, such as YAML or JSON files, are stored in a designated directory for easy access and management.

Log Files: Log files generated by applications, builds, and deployments are stored in a designated directory for easy access and management.

Jenkins Master-Slave Architecture



As you can see in the diagram provided above, on the left is the Remote source code repository. The Jenkins server accesses the master environment on the left side and the master environment can push down to multiple other Jenkins Slave environments to distribute the workload.

That lets you run multiple builds, tests, and product environment across the entire architecture. Jenkins Slaves can be running different build versions of the code for different operating systems and the server Master controls how each of the builds operates.

Supported on a master-slave architecture, Jenkins comprises many slaves working for a master. This architecture - the Jenkins Distributed Build - can run identical test cases in different environments. Results are collected and combined on the master node for monitoring.

The standard Jenkins installation includes Jenkins master, and in this setup, the master will be managing all our build system's tasks. If we're working on a number of projects, we can run numerous jobs on each one. Some projects require the use of specific nodes, which necessitates the use of slave nodes.

The Jenkins master is in charge of scheduling jobs, assigning slave nodes, and sending builds to slave nodes for execution. It will also keep track of the slave node state (offline or online), retrieve build results from slave nodes, and display them on the terminal output. In most installations, multiple slave nodes will be assigned to the task of building jobs.

Before we get started, **let's double-check that we have all of the prerequisites in place for adding a slave node**:

- **Jenkins Server** is up and running and ready to use
- Another server for a slave node configuration
- The Jenkins server and the slave server are both connected to the same network

Jenkins slave :

- if we setup builds to happen in server-client environments, then it is called as Master-Slave setup.
- It helps to load on server is reduced

➢ platform testing

➢ reducing cost

Requirements for setup of Jenkins slave -

- Need to have one Jenkins master
- Need to have a client/node pc, referred as Jenkins slave
- Jenkins master can be windows, Linux, or mac,
- Jenkins slave can be windows, Linux, mac
- Jenkins Master and slave should be on same network
- we do not require any Jenkins installation on Jenkins slave
- On Jenkins slave, need to have a user id and password that has administrative privileges.
- Jenkins Master these are plugin need to be installed

1. if Jenkins slave is windows, install java web start plugin
2. if Jenkins slave is Linux or mac install ssh slave agent plugin

Setup Relationship between Master and Slave (Windows)

● Note -- Make sure that Manage Jenkins -> Manage plugins and install 'SSH Slaves Plugin' or SSH Agent Plugin

● Go to Manage Jenkins -> Manage node

- Select the new node from the left pane, then enter slave machine's IP address and select 'Permanent Agent' and click okay.
- Now, proceed with filling further details of node
- In the 'remote root directory', add the path for a directory dedicated to be used by agent which should be /home/jenkins

example 1: if Slave OS is linux, and if user id is : root , then its home directory will be /root

Example 2 : if Slave OS is linux, and if user id is : india , then its home directory will be /home

- In the launch method, select 'launch slave agents via ssh' and add the slave machine's ip address and credentials
- ** Lanch method --> if linux --> SSH option and if windows--> java webstart

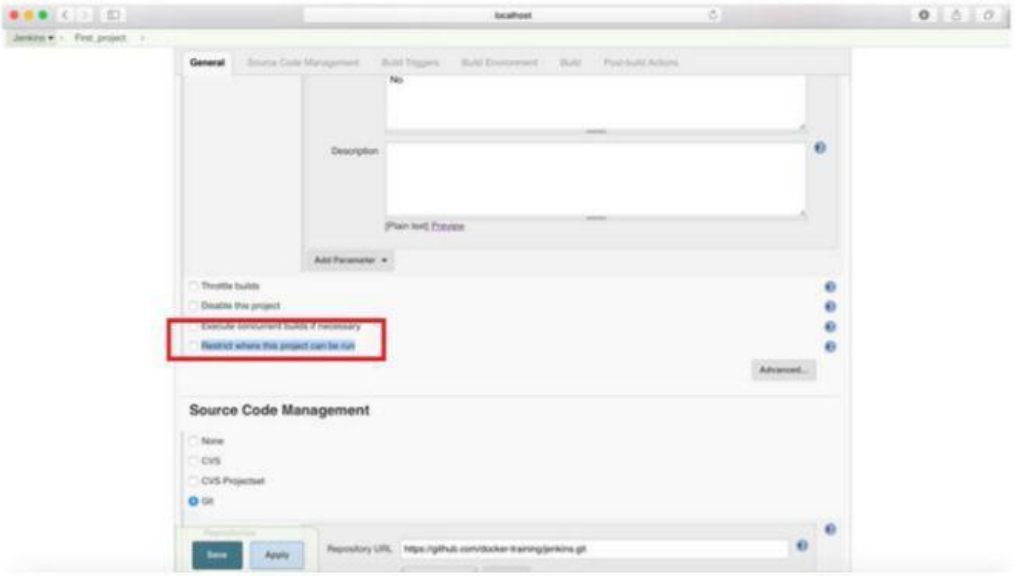
Configure Project to build on slave

Step 1 –

Open the **Project** and click on **Configure**

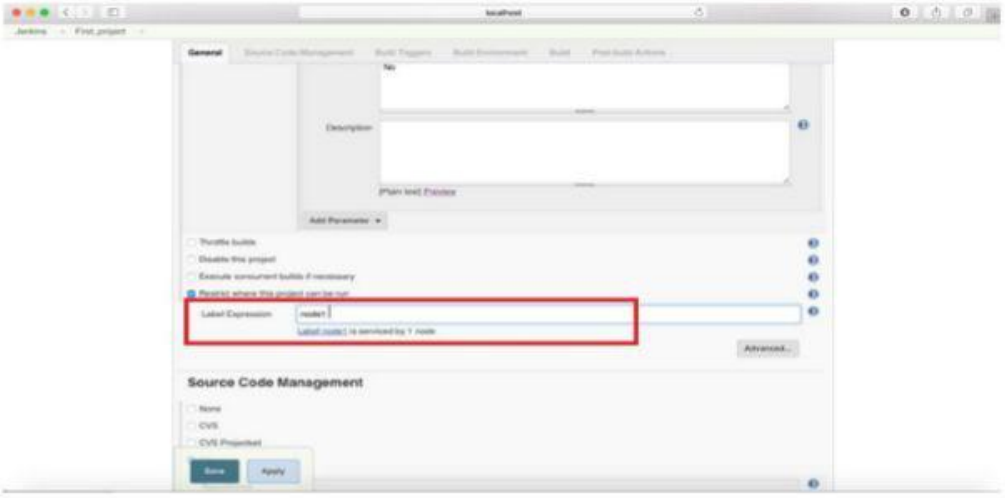
Step 2 -

Find the check-box in General section "**Restrict where this project can be run**" and click on it



Step 3 -

Just type the **Label** name of your slave

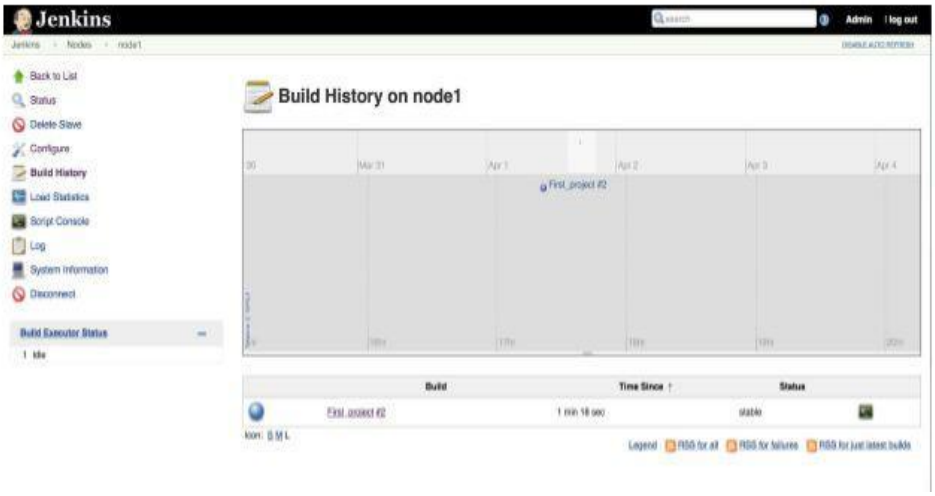


And then **Save** the changes.

- You can also define more than one labels (i.e. more than one node to the project) which will allow you to run the build in parallel and Jenkins will choose the faster slave for your build.

Step 4 - Now, execute the build from Manage Jenkins > Manage Nodes > Select the node > select project to build

- You can also check the project build statistics



Build systems

A build system is a key component in DevOps, and it plays an important role in the software development and delivery process. It automates the process of compiling and packaging source code into a deployable artifact, allowing for efficient and consistent builds.

Here are some of the key functions performed by a build system:

Compilation: The build system compiles the source code into a machine-executable format, such as a binary or an executable jar file.

Dependency Management: The build system ensures that all required dependencies are available and properly integrated into the build artifact. This can include external libraries, components, and other resources needed to run the application.

Testing: The build system runs automated tests to ensure that the code is functioning as intended, and to catch any issues early in the development process.

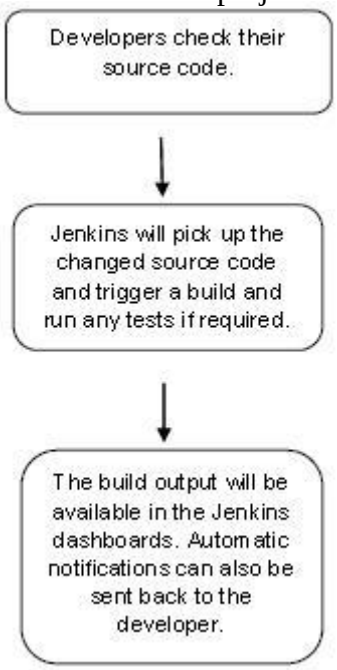
Packaging: The build system packages the compiled code and its dependencies into a single, deployable artifact, such as a Docker image or a tar archive.

Version Control: The build system integrates with version control systems, such as Git, to track changes to the code and manage releases.

Continuous Integration: The build system can be configured to run builds automatically whenever changes are made to the code, allowing for fast feedback and continuous integration of new code into the main branch.

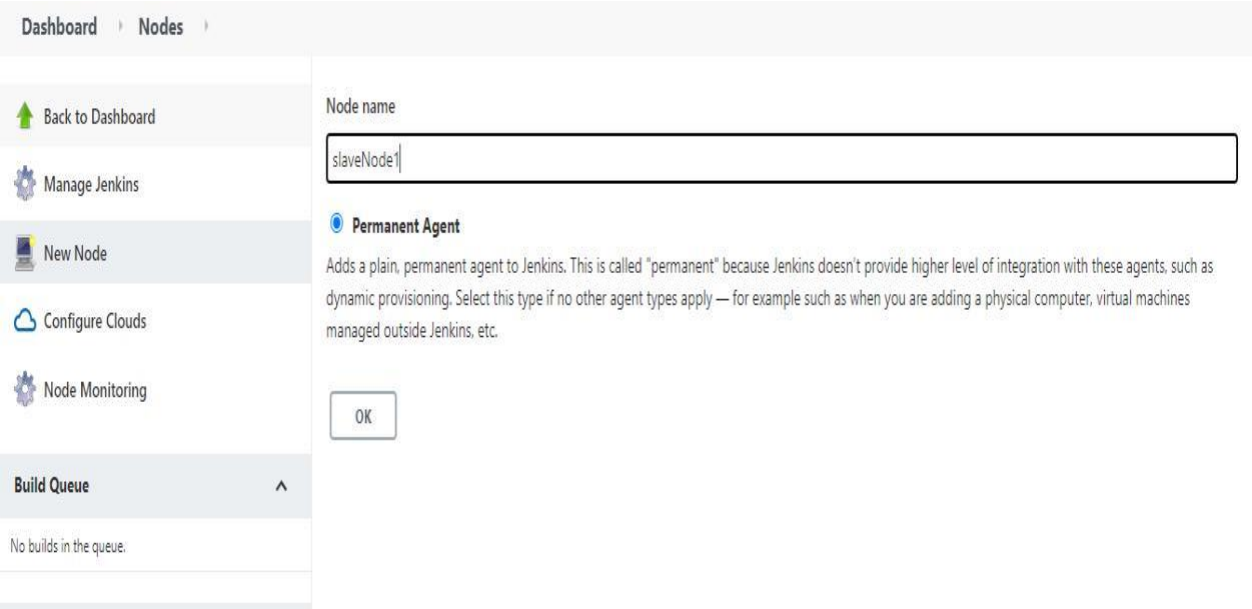
Deployment: The build system can be integrated with deployment tools and processes to automate the deployment of the build artifact to production environments.

In DevOps, it's important to have a build system that is fast, reliable, and scalable, and that can integrate with other tools and processes in the software development and delivery pipeline. There are many build systems available, each with its own set of features and capabilities, and choosing the right one will depend on the specific needs of the project and team.



First, we'll go to **"Manage Jenkins -> Manage Nodes -> New Node" to create a new node**:

On the next screen, we **enter the "Node Name" (slaveNode1), select "Permanent Agent", then click "OK"**:

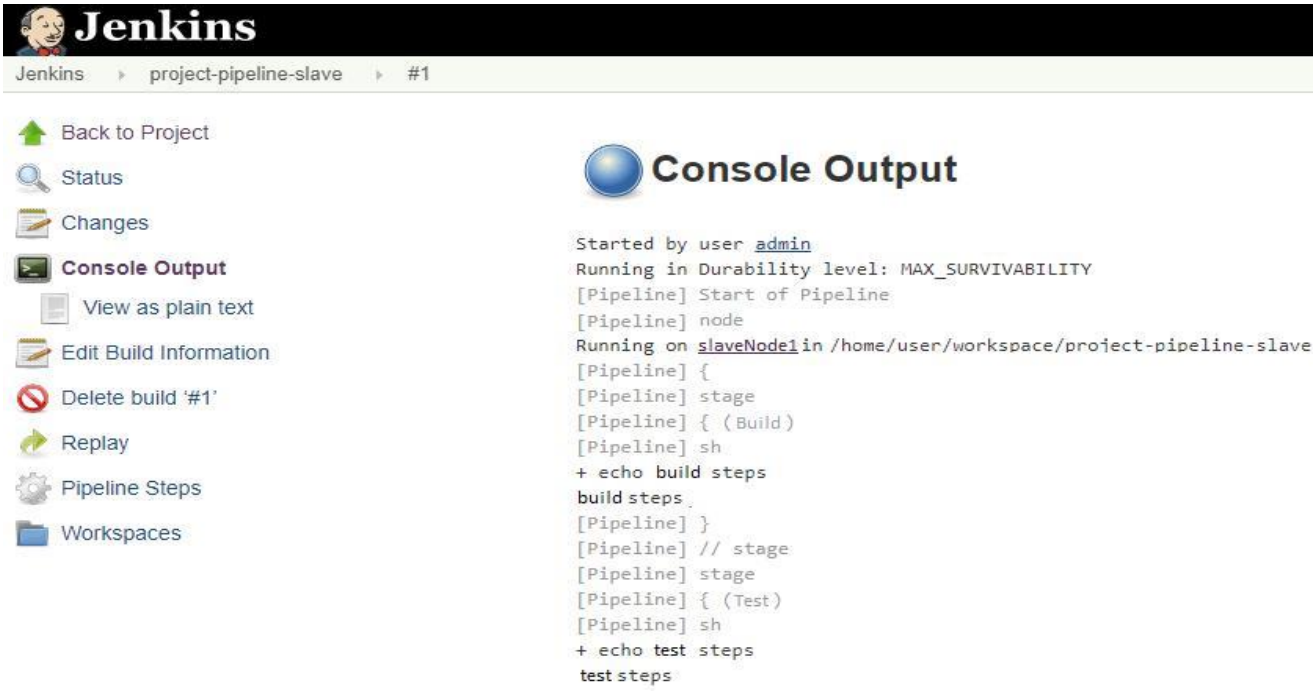


After clicking "OK", **we'll be taken to a screen with a new form where we need to fill out the slave node's information**. We're considering the slave node to be running on Linux operating systems, hence the launch method is set to "Launch agents via ssh".

In the same way, we'll add relevant details, such as the name, description, and a number of executors.

We'll save our work by pressing the "Save" button. The "Labels" with the name "slaveNode1" will help us to set up jobs on this slave node:

On the left pane, we click the “Build Now” button to execute our Pipeline. After Pipeline execution is completed, we'll see the Pipeline view:



We can verify the history of the executed build under the Build History by clicking the build number. As shown above, when we click on the build number and select "Console Output", we can see that the pipeline ran on our *slaveNode1* machine.

Jenkins build server

Jenkins build server

Jenkins is a popular open-source automation server that helps developers automate parts of the software development process. A Jenkins build server is responsible for building, testing, and deploying software projects.

A Jenkins build server is typically set up on a dedicated machine or a virtual machine, and is used to manage the continuous integration and continuous delivery (CI/CD) pipeline for a software project. The build server is configured with all the necessary tools, dependencies, and plugins to build, test, and deploy the project.

The build process in Jenkins typically starts with code being committed to a version control system (such as Git), which triggers a build on the Jenkins server. The Jenkins server then checks out the code, builds it, runs tests on it, and if everything is successful, deploys the code to a staging or production environment.

Jenkins has a large community of developers who have created hundreds of plugins that extend its functionality, so it's easy to find plugins to support specific tools, technologies, and workflows. For example, there are plugins for integrating with cloud infrastructure, running security scans, deploying to various platforms, and more.

Overall, a Jenkins build server can greatly improve the efficiency and reliability of the software development process by automating repetitive tasks, reducing the risk of manual errors, and enabling developers to focus on writing code.

Managing build dependencies

Managing build dependencies is an important aspect of continuous integration and continuous delivery (CI/CD) pipelines. In software development, dependencies refer to external libraries, tools, or resources that a project relies on to build, test, and deploy. Proper management of dependencies can ensure that builds are repeatable and that the build environment is consistent and up-to-date.

Here are some common practices for managing build dependencies in Jenkins:

Dependency Management Tools: Utilize tools such as Maven, Gradle, or npm to manage dependencies and automate the process of downloading and installing required dependencies for a build.

Version Pinning: Specify exact versions of dependencies to ensure builds are consistent and repeatable.

Caching: Cache dependencies locally on the build server to improve build performance and reduce the time it takes to download dependencies.

Continuous Monitoring: Regularly check for updates and security vulnerabilities in dependencies to ensure the build environment is secure and up-to-date.

Automated Testing: Automated testing can catch issues related to dependencies early in the development process.

By following these practices, you can effectively manage build dependencies and maintain the reliability and consistency of your CI/CD pipeline.

Jenkins is an open source automation tool written in Java programming language that allows continuous integration.

Jenkins **builds** and **tests** our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

It also allows us to continuously **deliver** our software by integrating with a large number of testing and deployment technologies.

Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including build, document, test, package, stage, deploy static analysis and much more.

Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

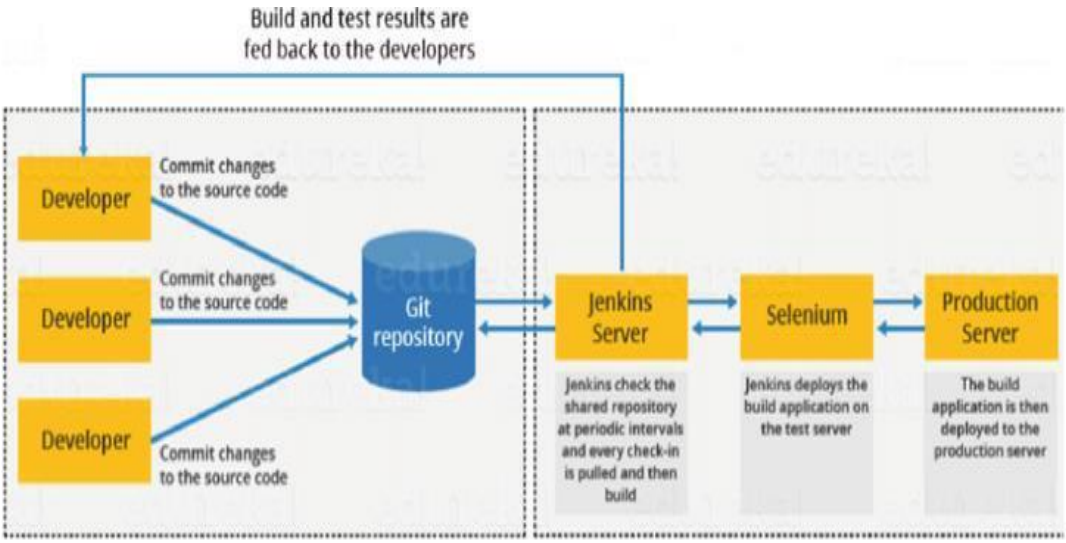
For example: If any organization is developing a project, then **Jenkins** will continuously test your project builds and show you the errors in early stages of your development.

Possible steps executed by Jenkins are for example:

- Perform a software build using a build system like Gradle or Maven
Apache ○ Execute a shell script

○ Archive a build result ○
Running software tests

Jenkin workflow



2. No Broken Code

Jenkins ensures that the code is good and tested well through continuous integration. The final code is merged only when all the tests are successful. This makes sure that no broken code is shipped into production.

What are the Jenkins Features?

Jenkins offers many attractive features for developers:

- **Easy Installation**

  Jenkins is a platform-agnostic, self-contained Java-based program, ready to run with packages for Windows, Mac OS, and Unix-like operating systems.

- **Easy Configuration**

  Jenkins is easily set up and configured using its web interface, featuring error checks and a built-in help function.

- **Available Plugins**

  There are hundreds of plugins available in the Update Center, integrating with every tool in the CI and CD toolchain.

- **Extensible**

  Jenkins can be extended by means of its plugin architecture, providing nearly endless possibilities for what it can do.

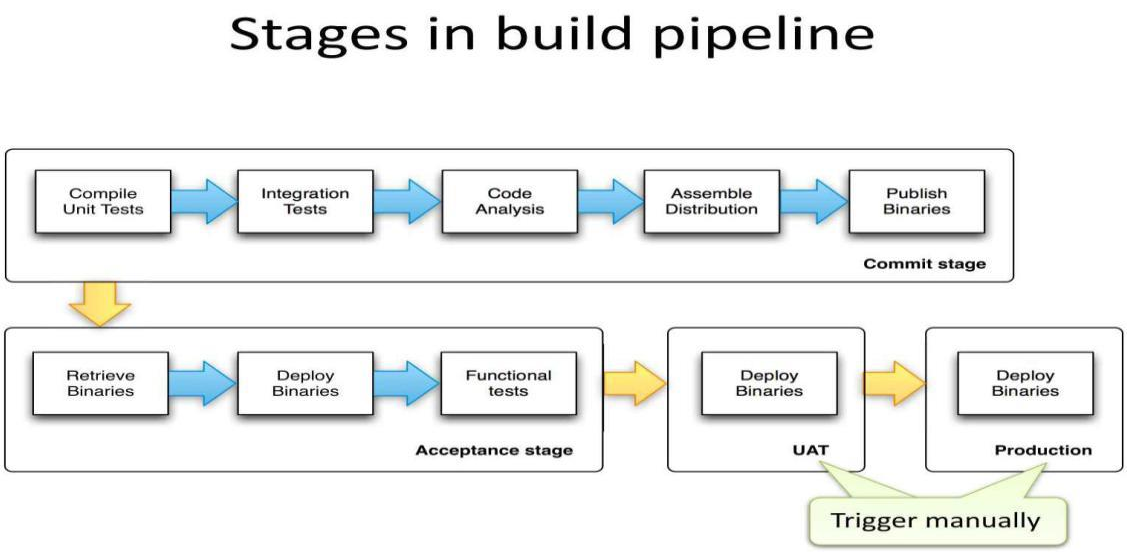
- **Easy Distribution**

  Jenkins can easily distribute work across multiple machines for faster builds, tests, and deployments across multiple platforms.

- **Free Open Source**

Jenkins is an open-source resource backed by heavy community support.

Build pipelines



A build pipeline in DevOps is a set of automated processes that compile, build, and test software, and prepare it for deployment. A build pipeline represents the end-to-end flow of code changes from development to production.

steps involved in a typical build pipeline include:

Code Commit: Developers commit code changes to a version control system such as Git.

Build and Compile: The code is built and compiled, and any necessary dependencies are resolved.

Unit Testing: Automated unit tests are run to validate the code changes.

Integration Testing: Automated integration tests are run to validate that the code integrates correctly with other parts of the system.

Staging: The code is deployed to a staging environment for further testing and validation.

Release: If the code passes all tests, it is deployed to the production environment.

Monitoring: The deployed code is monitored for performance and stability.

A build pipeline can be managed using a continuous integration tool such as Jenkins, TravisCI, or CircleCI. These tools automate the build process, allowing you to quickly and easily make changes to the pipeline, and ensuring that the pipeline is consistent and reliable.

In DevOps, the build pipeline is a critical component of the continuous delivery process, and is used to ensure that code changes are tested, validated, and deployed to production as quickly and efficiently as possible. By automating the build pipeline, you can reduce the time and effort required to deploy code changes, and improve the speed and quality of your software delivery process.

Build servers

When you're developing and deploying software, one of the first things to figure out is how to take your code and deploy your working application to a production environment where people can interact with your software.

Most development teams understand the importance of version control to coordinate code commits, and build servers to compile and package their software, but Continuous Integration (CI) is a big topic.

Why build servers are important

Build servers have 3 main purposes:

- Compiling committed code from your repository many times a day
- Running automatic tests to validate code
- Creating deployable packages and handing off to a deployment tool, like Octopus Deploy

Without a build server you're slowed down by complicated, manual processes and the needless time constraints they introduce. For example, without a build server:

- Your team will likely need to commit code before a daily deadline or during change windows
- After that deadline passes, no one can commit again until someone manually creates and tests a build

- If there are problems with the code, the deadlines and manual processes further delay the fixes

Without a build server, the team battles unnecessary hurdles that automation removes. A build server will repeat these tasks for you throughout the day, and without those human-caused delays.

But CI doesn't just mean less time spent on manual tasks or the death of arbitrary deadlines, either. By automatically taking these steps many times a day, you fix problems sooner and your results become more predictable. Build servers ultimately help you deploy through your pipeline with more confidence.

Building servers in DevOps involves several steps:

Requirements gathering: Determine the requirements for the server, such as hardware specifications, operating system, and software components needed.

Server provisioning: Choose a method for provisioning the server, such as physical installation, virtualization, or cloud computing.

Operating System installation: Install the chosen operating system on the server.

Software configuration: Install and configure the necessary software components, such as web servers, databases, and middleware.

Network configuration: Set up network connectivity, such as IP addresses, hostnames, and firewall rules.

Security configuration: Configure security measures, such as user authentication, access control, and encryption.

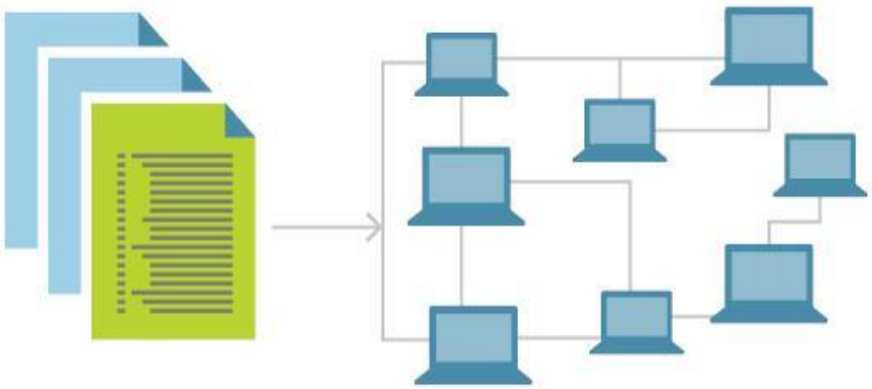
Monitoring and maintenance: Implement monitoring and maintenance processes, such as logging, backup, and disaster recovery.

Deployment: Deploy the application to the server and test it to ensure it is functioning as expected.

Throughout the process, it is important to automate as much as possible using tools such as Ansible, Chef, or Puppet to ensure consistency and efficiency in building servers.

Infrastructure as code

Infrastructure as code (IaC) uses DevOps methodology and versioning with a descriptive model to define and deploy infrastructure, such as networks, virtual machines, load balancers, and connection topologies. Just as the same source code always generates the same binary, an IaC model generates the same environment every time it deploys.



IaC is a key DevOps practice and a component of continuous delivery. With IaC, DevOps teams can work together with a unified set of practices and tools to deliver applications and their supporting infrastructure rapidly and reliably at scale.

IaC evolved to solve the problem of *environment drift* in release pipelines. Without IaC, teams must maintain deployment environment settings individually. Over time, each environment becomes a "snowflake," a unique configuration that can't be reproduced automatically. Inconsistency among environments can cause deployment issues. Infrastructure administration and maintenance involve manual processes that are error prone and hard to track.

IaC avoids manual configuration and enforces consistency by representing desired environment states via well-documented code in formats such as JSON. Infrastructure deployments with IaC are repeatable and prevent runtime issues caused by configuration drift or missing dependencies. Release pipelines execute the environment descriptions and version configuration models to configure target environments. To make changes, the team edits the source, not the target.

Idempotence, the ability of a given operation to always produce the same result, is an important IaC principle. A deployment command always sets the target environment into the same configuration, regardless of the environment's starting state. Idempotency is achieved by either

automatically configuring the existing target, or by discarding the existing target and recreating a fresh environment.

IAC can be achieved by using tools such as Terraform, CloudFormation, or Ansible to define infrastructure components in a file that can be versioned, tested, and deployed in a consistent and automated manner.

Benefits of IAC include:

Speed: IAC enables quick and efficient provisioning and deployment of infrastructure.

Consistency: By using code to define and manage infrastructure, it is easier to ensure consistency across multiple environments.

Repeatability: IAC allows for easy replication of infrastructure components in different environments, such as development, testing, and production.

Scalability: IAC makes it easier to scale infrastructure as needed by simply modifying the code.

Version control: Infrastructure components can be versioned, allowing for rollback to previous versions if necessary.

Overall, IAC is a key component of modern DevOps practices, enabling organizations to manage their infrastructure in a more efficient, reliable, and scalable way.

Building by dependency order

Building by dependency order in DevOps is the process of ensuring that the components of a system are built and deployed in the correct sequence, based on their dependencies. This is necessary to ensure that the system functions as intended, and that components are deployed in the right order so that they can interact correctly with each other.

The steps involved in building by dependency order in DevOps include:

Define dependencies: Identify all the components of the system and the dependencies between them. This can be represented in a diagram or as a list.

Determine the build order: Based on the dependencies, determine the correct order in which components should be built and deployed.

Automate the build process: Use tools such as Jenkins, TravisCI, or CircleCI to automate the build and deployment process. This allows for consistency and repeatability in the build process.

Monitor progress: Monitor the progress of the build and deployment process to ensure that components are deployed in the correct order and that the system is functioning as expected.

Test and validate: Test the system after deployment to ensure that all components are functioning as intended and that dependencies are resolved correctly.

Rollback: If necessary, have a rollback plan in place to revert to a previous version of the system if the build or deployment process fails.

In conclusion, building by dependency order in DevOps is a critical step in ensuring the success of a system deployment, as it ensures that components are deployed in the correct order and that dependencies are resolved correctly. This results in a more stable, reliable, and consistent system.

Build phases

In DevOps, there are several phases in the build process, including:

Planning: Define the project requirements, identify the dependencies, and create a build plan.

Code development: Write the code and implement features, fixing bugs along the way.

Continuous Integration (CI): Automatically build and test the code as it is committed to a version control system.

Continuous Delivery (CD): Automatically deploy code changes to a testing environment, where they can be tested and validated.

Deployment: Deploy the code changes to a production environment, after they have passed testing in a pre-production environment.

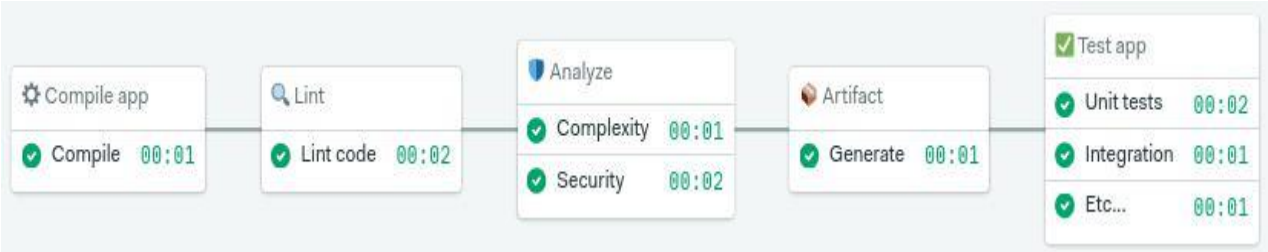
Monitoring: Continuously monitor the system to ensure that it is functioning as expected, and to detect and resolve any issues that may arise.

Maintenance: Continuously maintain and update the system, fixing bugs, adding new features, and ensuring its stability.

These phases help to ensure that the build process is efficient, reliable, and consistent, and that code changes are validated and deployed in a controlled manner. Automation is a key aspect of DevOps, and it helps to make these phases more efficient and less prone to human error.

In continuous integration (CI), this is where we build the application for the first time. The build stage is the first stretch of a CI/CD pipeline, and it automates steps like downloading dependencies, installing tools, and compiling.

Besides building code, build automation includes using tools to check that the code is safe and follows best practices. The build stage usually ends in the artifact generation step, where we create a production-ready package. Once this is done, the testing stage can begin.



The build stage starts from code commit and runs from the beginning up to the test stage

We'll be covering testing in-depth in future articles (subscribe to the newsletter so you don't miss them). Today, we'll focus on build automation.

Build automation verifies that the application, at a given code commit, can qualify for further testing. We can divide it into three parts:

1. **Compilation**: the first step builds the application.
2. **Linting**: checks the code for programmatic and stylistic errors.
3. **Code analysis**: using automated source-checking tools, we control the code's quality.
4. **Artifact generation**: the last step packages the application for release or deployment.

Alternative build servers

There are several alternative build servers in DevOps, including:

Jenkins - an open-source, Java-based automation server that supports various plugins and integrations.

Travis CI - a cloud-based, open-source CI/CD platform that integrates with Github.

CircleCI - a cloud-based, continuous integration and delivery platform that supports multiple languages and integrates with several platforms.

GitLab CI/CD - an integrated CI/CD solution within GitLab that allows for complete project and pipeline management.

Bitbucket Pipelines - a CI/CD solution within Bitbucket that allows for pipeline creation and management within the code repository.

AWS CodeBuild - a fully managed build service that compiles source code, runs tests, and produces software packages that are ready to deploy.

Azure Pipelines - a CI/CD solution within Microsoft Azure that supports multiple platforms and programming languages.

Collating quality measures

In DevOps, collating quality measures is an important part of the continuous improvement process. The following are some common quality measures used in DevOps to evaluate the quality of software systems:

Continuous Integration (CI) metrics - metrics that track the success rate of automated builds and tests, such as build duration and test pass rate.

Continuous Deployment (CD) metrics - metrics that track the success rate of deployments, such as deployment frequency and time to deployment.

Code review metrics - metrics that track the effectiveness of code reviews, such as review completion time and code review feedback. Performance metrics - measures of system performance in production, such as response time and resource utilization.

User experience metrics - measures of how users interact with the system, such as click-through rate and error rate.

Security metrics - measures of the security of the system, such as the number of security vulnerabilities and the frequency of security updates.

Incident response metrics - metrics that track the effectiveness of incident response, such as mean time to resolution (MTTR) and incident frequency.

By regularly collating these quality measures, DevOps teams can identify areas for improvement, track progress over time, and make informed decisions about the quality of their systems.

ZENOSS –MONITORING TOOL

Zenoss is a software platform for monitoring and managing the performance and availability of IT infrastructure and services. It is commonly used in IT operations and network management to ensure that systems and applications are running smoothly and to identify and address issues proactively. Zenoss provides a unified view of an organization's entire IT environment, allowing IT professionals to monitor servers, networks, storage, virtualization, and cloud resources from a single dashboard.

Key features and capabilities of Zenoss typically include:

1. **Monitoring:** Zenoss can collect and analyze data from various sources, such as servers, switches, routers, and applications. It uses this data to monitor the health and performance of IT assets.
2. **Alerting:** It can send alerts and notifications when it detects issues or anomalies. These alerts can be configured to go to IT administrators or support teams for rapid response.

3. **Event Correlation:** Zenoss can correlate events and alerts to help identify the root cause of problems and reduce alert noise.

4. **Dashboards and Reporting:** Zenoss provides customizable dashboards and reporting capabilities, allowing users to create visual representations of data and generate reports for analysis.

5. **Auto-Discovery:** It can automatically discover and map IT resources in the environment, making it easier to keep track of what's in the infrastructure.

6. **Scalability:** Zenoss is designed to handle large-scale IT environments and can scale to accommodate the needs of organizations with diverse and extensive infrastructure.

7. **Integration:** It often offers integration with other IT management and automation tools, enabling organizations to streamline their IT operations.

Zenoss is typically used in enterprise environments, data centers, and cloud-based infrastructures to ensure the reliability and performance of critical IT services. It helps organizations proactively identify and resolve issues, which can lead to improved service availability and reduced downtime. Please note that the specific features and capabilities of Zenoss may evolve over time, so it's a good idea to check the official Zenoss website or documentation for the most up-to-date information.

As of my last knowledge update in September 2021, Ayehu is a company that specializes in IT automation and orchestration solutions. Ayehu's primary offering is a platform called the "Ayehu NG" (Next Generation) platform, which is designed to help organizations automate their IT and business processes, streamline workflows, and improve operational efficiency. Here are some key aspects of Ayehu and its platform:

1. **Automation and Orchestration:** Ayehu's platform enables the automation of a wide range of tasks and processes across IT operations and beyond. It allows organizations to create workflows that can automate repetitive and manual tasks, reducing the need for human intervention.

2. **Integration:** Ayehu is known for its ability to integrate with various IT systems, applications, and tools. This integration capability allows organizations to orchestrate and automate tasks across their entire technology stack.

3. **Incident Response:** Ayehu's platform is often used for incident response automation. It can automatically detect and respond to IT incidents and security events, helping organizations reduce response times and minimize the impact of incidents

4. **Workflow Creation:** Users can design and customize workflows using a visual, drag-and-drop interface. This makes it accessible to both technical and non-technical users within an organization.

5. **Alerting and Notifications:** The platform can trigger alerts and notifications based on predefined conditions or events. This ensures that relevant personnel are informed promptly when an issue arises.

6. **Compliance and Governance:** Ayehu can assist organizations in enforcing compliance policies and governance standards by automating audit and compliance-related tasks.

7. **Scalability:** The platform is designed to scale and handle automation needs in large and complex IT environments.

8. **Reporting and Analytics:** Ayehu provides reporting and analytics capabilities to monitor the performance of workflows and track automation results.

Chef

Chef is an open source technology developed by Opscode. Adam Jacob, co-founder of Opscode is known as the founder of Chef. This technology uses Ruby encoding to develop basic building blocks like recipe and cookbooks. Chef is used in infrastructure automation and helps in reducing manual and repetitive tasks for infrastructure management.

Chef have got its own convention for different building blocks, which are required to manage and automate infrastructure.

Why Chef?

Chef is a configuration management technology used to automate the infrastructure provisioning. It is developed on the basis of Ruby DSL language. It is used to streamline the task of configuration and managing the company's server. It has the capability to get integrated with any of the cloud technology.

In DevOps, we use Chef to deploy and manage servers and applications in-house and on the cloud.

Features of Chef

Following are the most prominent features of Chef −

- Chef uses popular Ruby language to create a domain-specific language.

- Chef does not make assumptions on the current status of a node. It uses its mechanisms to get the current status of machine.
- Chef is ideal for deploying and managing the cloud server, storage, and software.

Advantages of Chef

Chef offers the following advantages −

- **Lower barrier for entry** − As Chef uses native Ruby language for configuration, a standard configuration language it can be easily picked up by anyone having some development experience.
- **Excellent integration with cloud** − Using the knife utility, it can be easily integrated with any of the cloud technologies. It is the best tool for an organization that wishes to distribute its infrastructure on multi-cloud environment.

Disadvantages of Chef

Some of the major drawbacks of Chef are as follows −

- One of the huge disadvantages of Chef is the way cookbooks are controlled. It needs constant babying so that people who are working should not mess up with others cookbooks.
- Only Chef solo is available.
- In the current situation, it is only a good fit for AWS cloud.
- It is not very easy to learn if the person is not familiar with Ruby.
- Documentation is still lacking.

Key Building Blocks of Chef

Recipe

It can be defined as a collection of attributes which are used to manage the infrastructure. These attributes which are present in the recipe are used to change the existing state or setting a particular infrastructure node. They are loaded during Chef client run and comparted with the existing attribute of the node (machine). It then gets to the status which is defined in the node resource of the recipe. It is the main workhorse of the cookbook.

Cookbook

A cookbook is a collection of recipes. They are the basic building blocks which get uploaded to Chef server. When Chef run takes place, it ensures that the recipes present inside it gets a given infrastructure to the desired state as listed in the recipe.

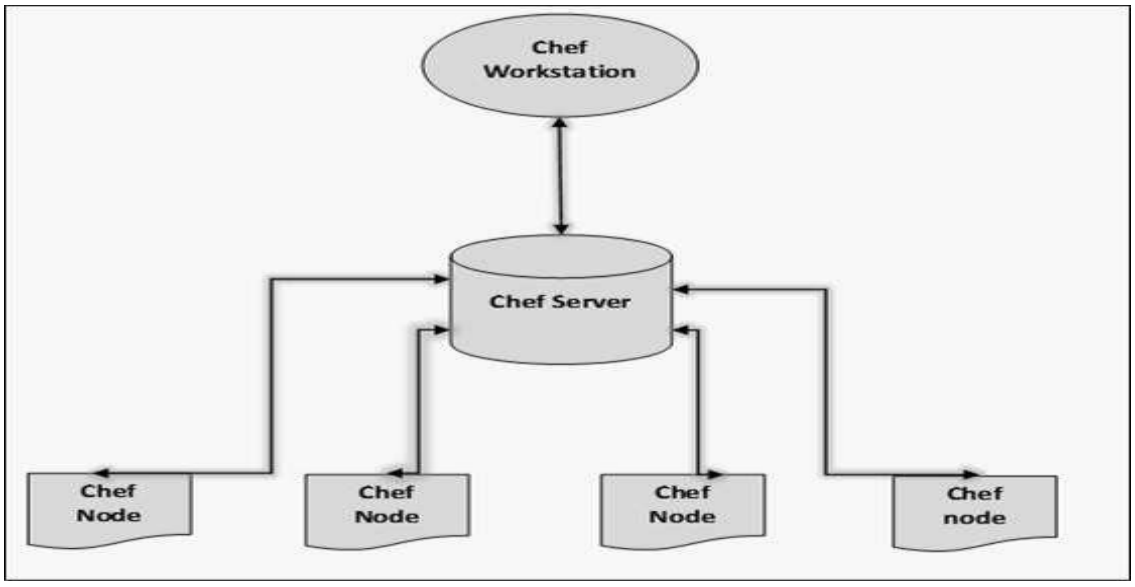
Resource

It is the basic component of a recipe used to manage the infrastructure with different kind of states. There can be multiple resources in a recipe, which will help in configuring and managing the infrastructure. For example −

- **package** − Manages the packages on a node
- **service** − Manages the services on a node
- **user** − Manages the users on the node
- **group** − Manages groups
- **template** − Manages the files with embedded Ruby template
- **cookbook_file** − Transfers the files from the files subdirectory in the cookbook to a location on the node
- **file** − Manages the contents of a file on the node
- **directory** − Manages the directories on the node
- **execute** − Executes a command on the node
- **cron** − Edits an existing cron file on the node

Chef - Architecture

- Chef works on a three-tier client server model wherein the working units such as cookbooks are developed on the Chef workstation. From the command line utilities such as knife, they are uploaded to the Chef server and all the nodes which are present in the architecture are registered with the Chef server.



- In Order to get the working Chef infrastructure in place, we need to set up multiple things in sequence.

- In the above setup, we have the following components.
- Chef Workstation
- This is the location where all the configurations are developed. Chef workstation is installed on the local machine. Detailed configuration structure is discussed in the later chapters of this tutorial.
- Chef Server
- This works as a centralized working unit of Chef setup, where all the configuration files are uploaded post development. There are different kinds of Chef server, some are hosted Chef server whereas some are built-in premise.
- Chef Nodes

They are the actual machines which are going to be managed by the Chef server. All the nodes can have different kinds of setup as per requirement. Chef client is the key component of all the nodes, which helps in setting up the communication between the Chef server and Chef node. The other components of Chef node is Ohai, which helps in getting the current state of any node at a given point of time

Chef workstation installation:-

>> update ip and host address
>> dpkg -i chef-work.....
>> cd /root/chef-repo/.chef
copy both .pem files into .chef folder
>> ls
>> vi knife.rb

```
log_level                :info
log_location             STDOUT
node_name                'student'
client_key               '/root/chef-repo/.chef/student.pem'
validation_client_name   'myorg-validator'
validation_key           '/root/chef-repo/.chef/myorg-validator.pem'
chef_server_url          'https://chefserver.pivotal.com/organizations/myorg
cookbook_path            ['/root/chef-repo/cookbooks']
```

>> knife ssl fetch / knife ssl fetch -s https://chefserver.pivotal.com
>> knife ssl check / knfie ssl check -s https://chefserver.pivotal.com
>> knife bootstrap 192.168.0.221 --ssh-user rishi --sudo --identity-file ~/.ssh/id_rsa --node-name chefnode.pivotal.com
#knife node list

chef node installation:-

>> updat ip & hostadd

>> #dpkg -i chef-client.....
>> mkdir -p /etc/chef
copy both .pem files
>> cd /etc/chef
>> vi client.rb
log_level :info
log_location STDOUT
chef_server_url 'https://chefserver.pivotal.com/organizations/myorg'
validation_client_name 'myorg-validator'
validation_key '/etc/chef/myorg-validator.pem'
client_key '/etc/chef/student.pem'
trusted_certs_dir '/etc/chef/trusted_certs'

>> knife ssl fetch -s https://chefserver.pivotal.com
>> knife ssl check -s https://chefserver.pivotal.com
>> useradd rishi
>> passwd rishi
>> usermod -aG sudo rishi
>> apt-get install ssh
>> ssh-keygen