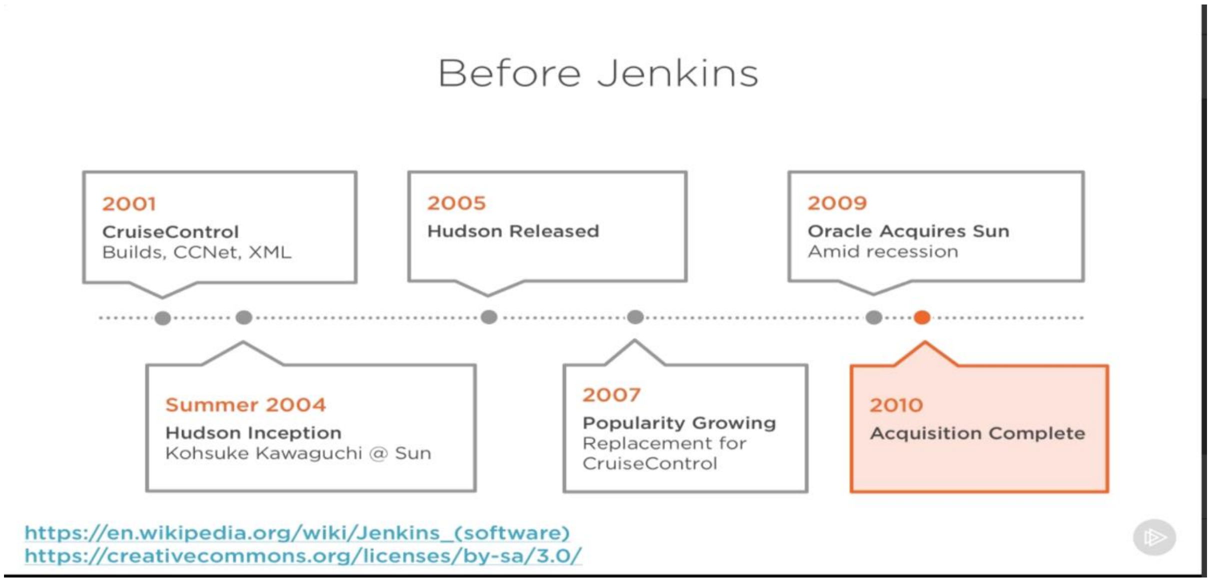
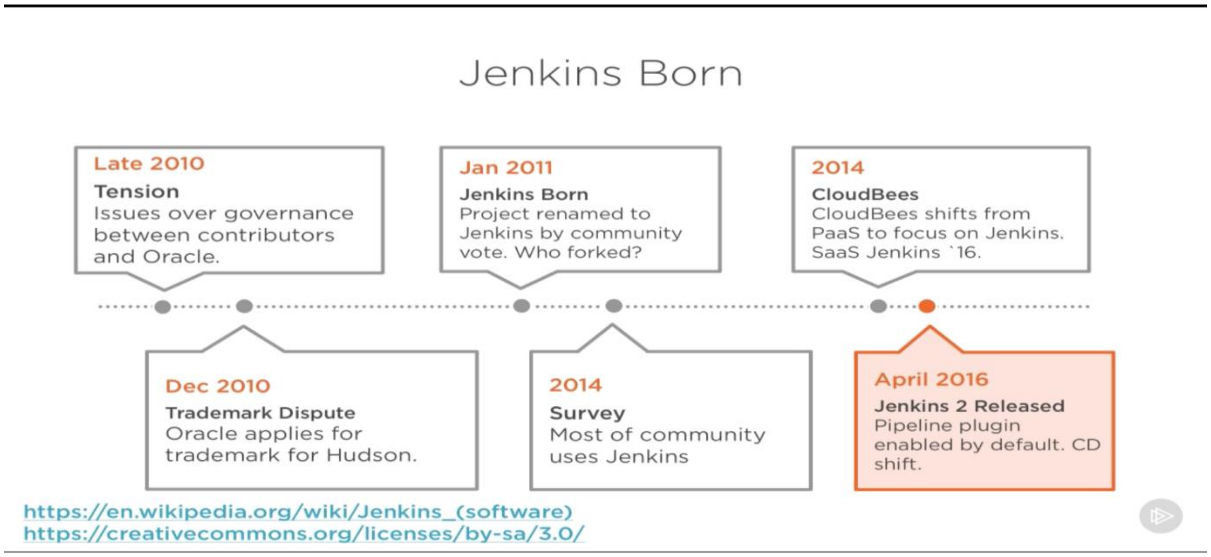
Jenkins Simple Notes



Before Jenkins

https://en.wikipedia.org/wiki/Jenkins_(software)
https://creativecommons.org/licenses/by-sa/3.0/

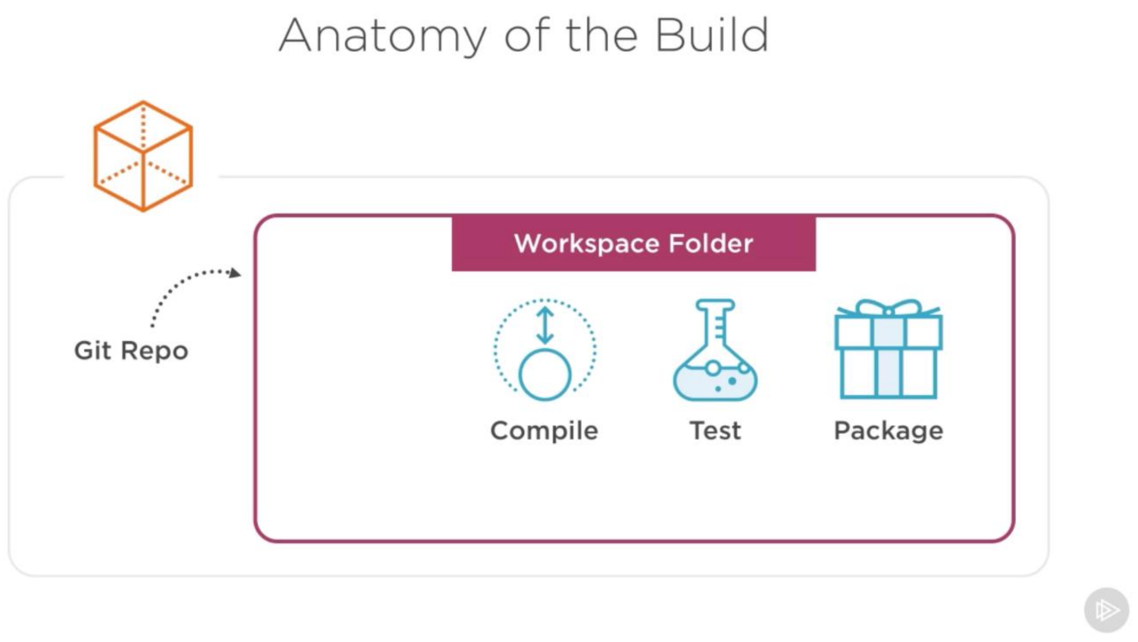


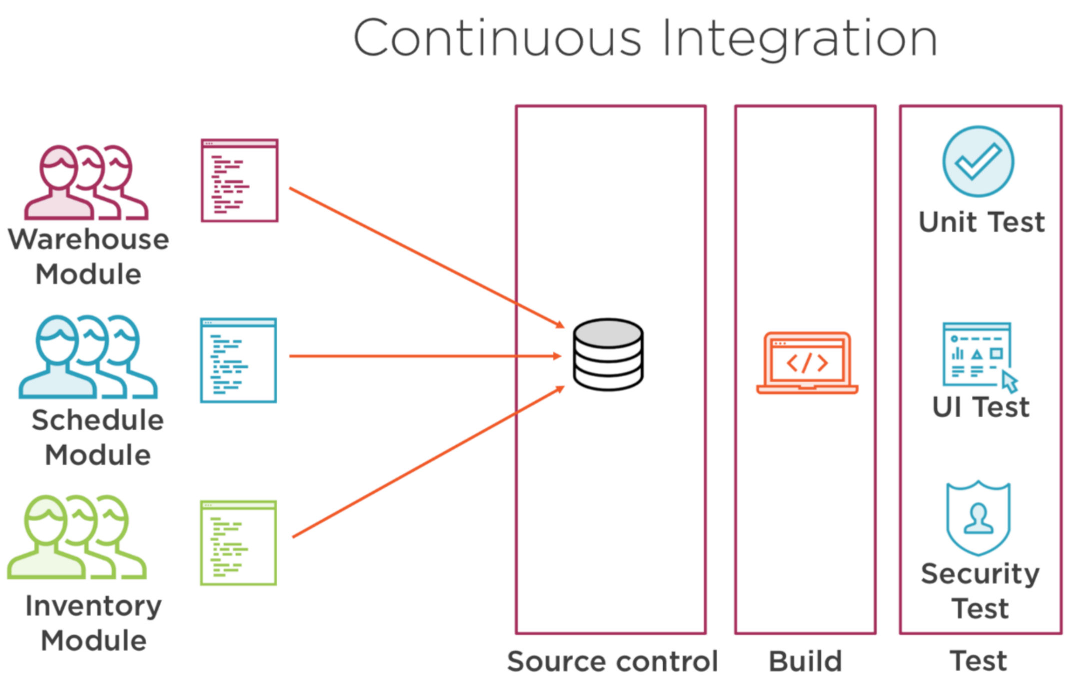
Jenkins Born

Late 2010
Tension
Issues over governance between contributors and Oracle.
Dec 2010
Trademark Dispute
Oracle applies for trademark for Hudson.
Jan 2011
Jenkins Born
Project renamed to Jenkins by community vote. Who forked?
2014
Survey
Most of community uses Jenkins
2014
CloudBees
CloudBees shifts from PaaS to focus on Jenkins. SaaS Jenkins `16.
April 2016
Jenkins 2 Released
Pipeline plugin enabled by default. CD shift.

https://en.wikipedia.org/wiki/Jenkins_(software)
https://creativecommons.org/licenses/by-sa/3.0/

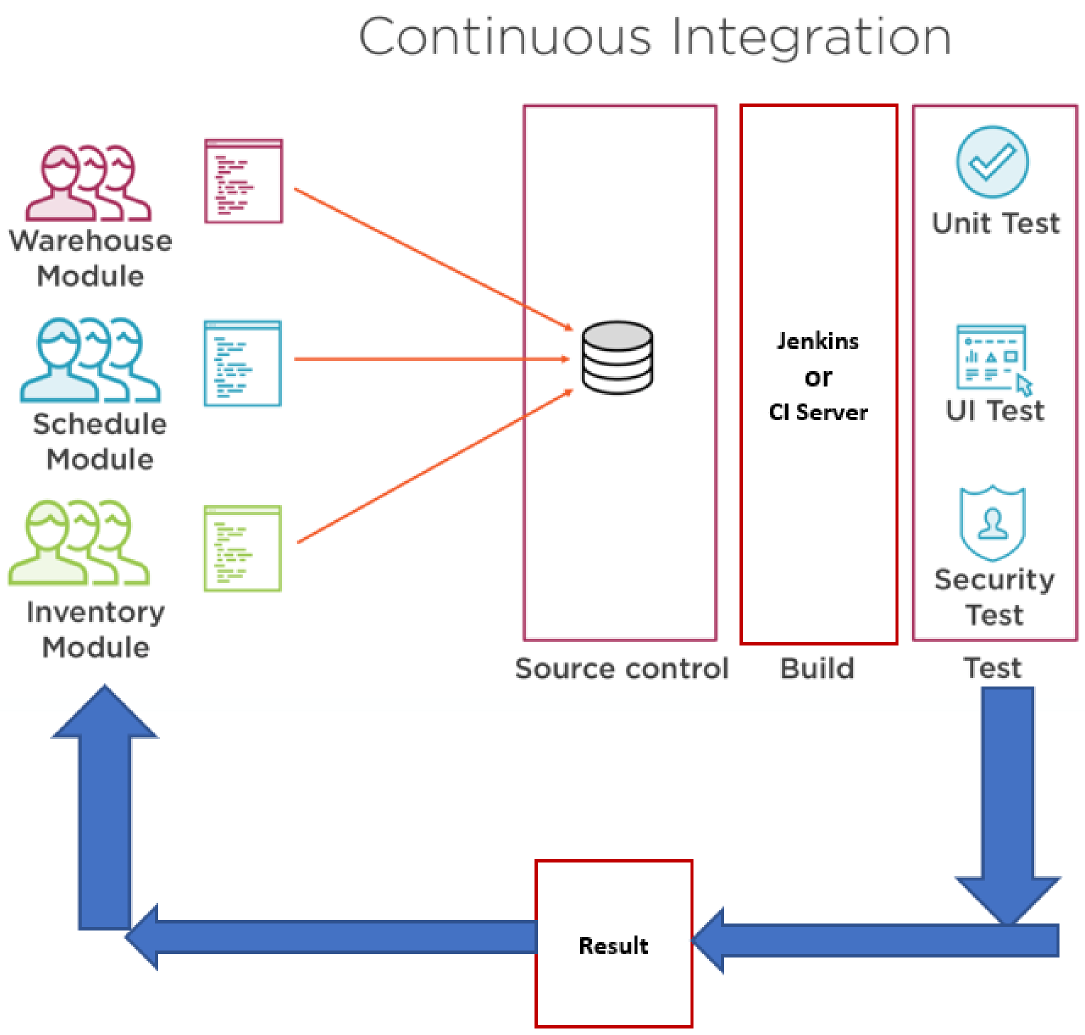
Anatomy of Build (Build process before Jenkins)



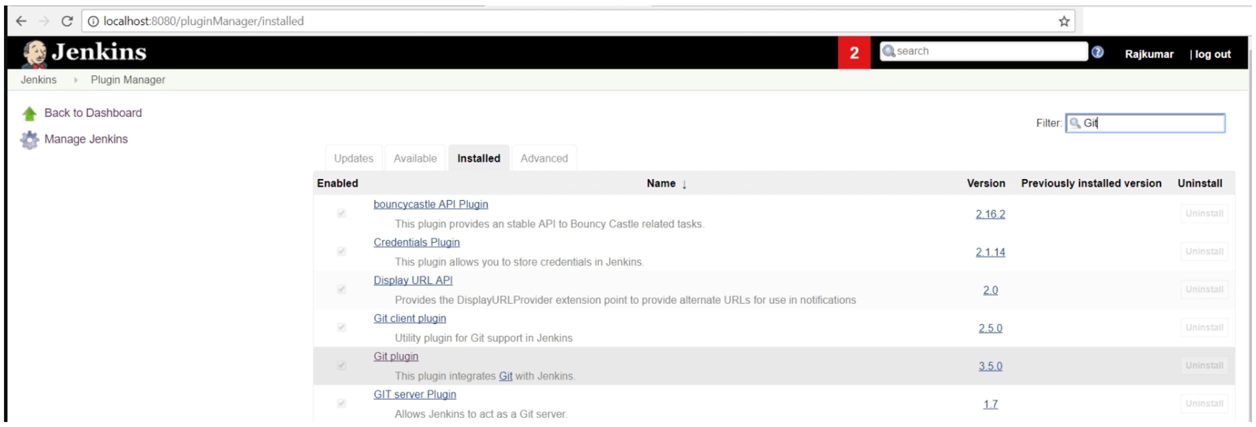


Before Jenkins Once all Developers had completed their assigned coding tasks, they used to commit their code all at same time. Later, Build is tested and deployed.
Code commits built, and test cycle was very infrequent, and a single build was done after many days. but using Jenkins we can automate everything.

Anatomy of Build (Build process after Jenkins)



Install GIT Plugin and integrate with Jenkins


localhost:8080/pluginManager/installed

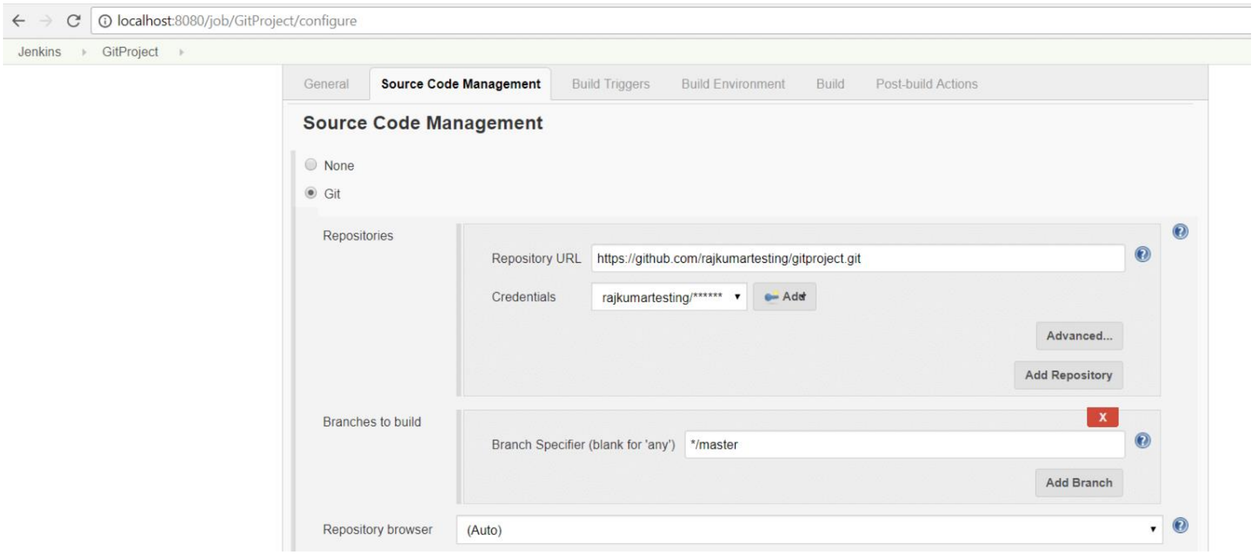
Jenkins › Plugin Manager

Back to Dashboard
Manage Jenkins

Filter: Git

Updates | Available | **Installed** | Advanced

Enabled	Name ↓	Version	Previously installed version	Uninstall
✓	bouncycastle API Plugin This plugin provides an stable API to Bouncy Castle related tasks.	2.16.2		Uninstall
✓	Credentials Plugin This plugin allows you to store credentials in Jenkins.	2.1.14		Uninstall
✓	Display URL API Provides the DisplayURLProvider extension point to provide alternate URLs for use in notifications	2.0		Uninstall
✓	Git client plugin Utility plugin for Git support in Jenkins	2.5.0		Uninstall
✓	Git plugin This plugin integrates Git with Jenkins.	3.5.0		Uninstall
✓	GIT server Plugin Allows Jenkins to act as a Git server.	1.7		Uninstall


localhost:8080/job/GitProject/configure

Jenkins › GitProject ›

General | **Source Code Management** | Build Triggers | Build Environment | Build | Post-build Actions

Source Code Management

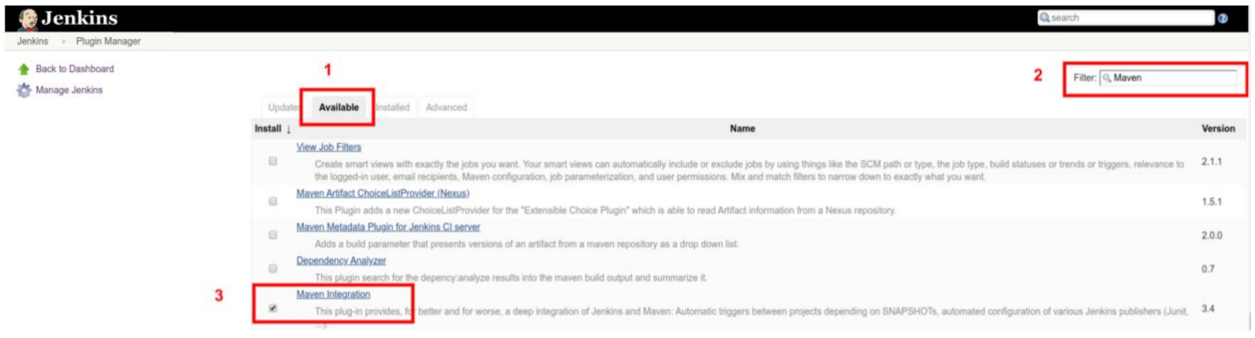
None [✓ Git]

Repositories
Repository URL: https://github.com/rajkumartesting/gitproject.git
Credentials: rajkumartesting/****** | Add
Advanced... | Add Repository

Branches to build
Branch Specifier (blank for 'any'): */master
Add Branch

Repository browser: (Auto)

Install Maven Plugin and integrate with Jenkins

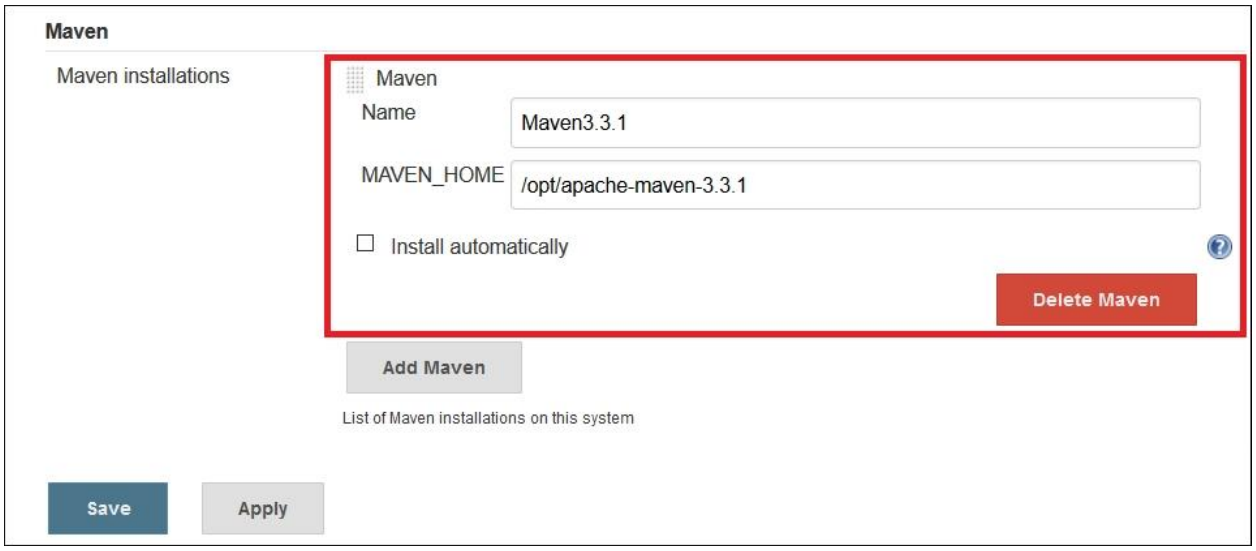

Jenkins › Plugin Manager

Back to Dashboard
Manage Jenkins

Filter: Maven

Updates | **Available** | Installed | Advanced

Install ↓	Name	Version
☐	View Job Filters Create smart views with exactly the jobs you want. Your smart views can automatically include or exclude jobs by using things like the SCM path or type, the job type, build statuses or trends or triggers, relevance to the logged-in user, email recipients, Maven configuration, job parameterization, and user permissions. Mix and match filters to narrow down to exactly what you want.	2.1.1
☐	Maven Artifact ChoiceListProvider (Nexus) This Plugin adds a new ChoiceListProvider for the "Extensible Choice Plugin" which is able to read Artifact information from a Nexus repository.	1.5.1
☐	Maven Metadata Plugin for Jenkins CI server Adds a build parameter that presents versions of an artifact from a maven repository as a drop down list.	2.0.0
☐	Dependency Analyzer This plugin search for the depency:analyze results into the maven build output and summarize it.	0.7
✓	Maven Integration This plug-in provides, for better and for worse, a deep integration of Jenkins and Maven: Automatic triggers between projects depending on SNAPSHOTs, automated configuration of various Jenkins publishers (Junit,	3.4



Maven

Maven installations

Maven

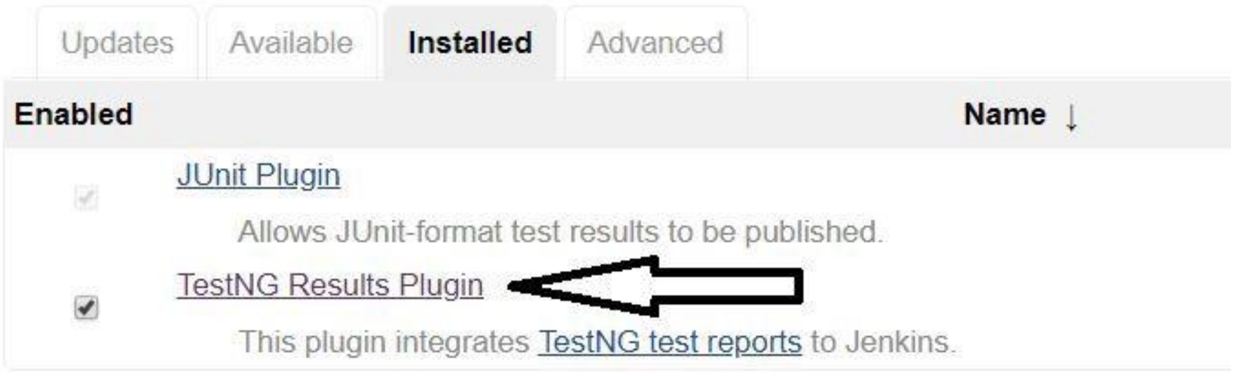
Name: Maven3.3.1

MAVEN_HOME: /opt/apache-maven-3.3.1

☐ Install automatically

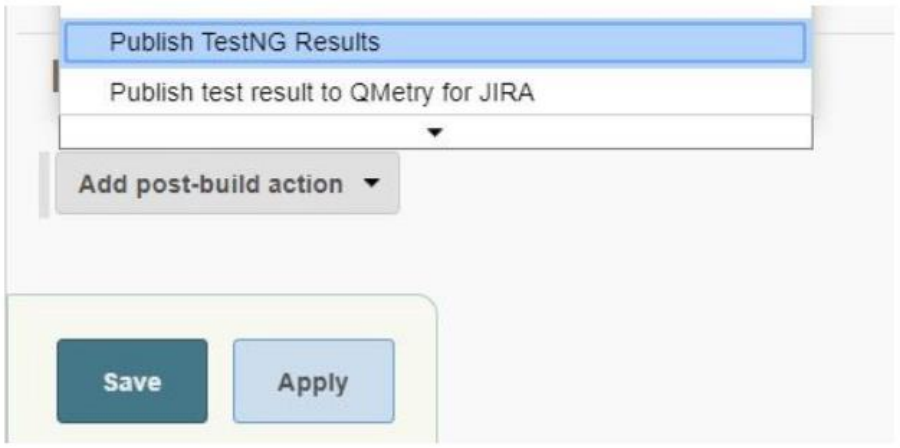
List of Maven installations on this system

Install TestNG Plugin and integrate with Jenkins

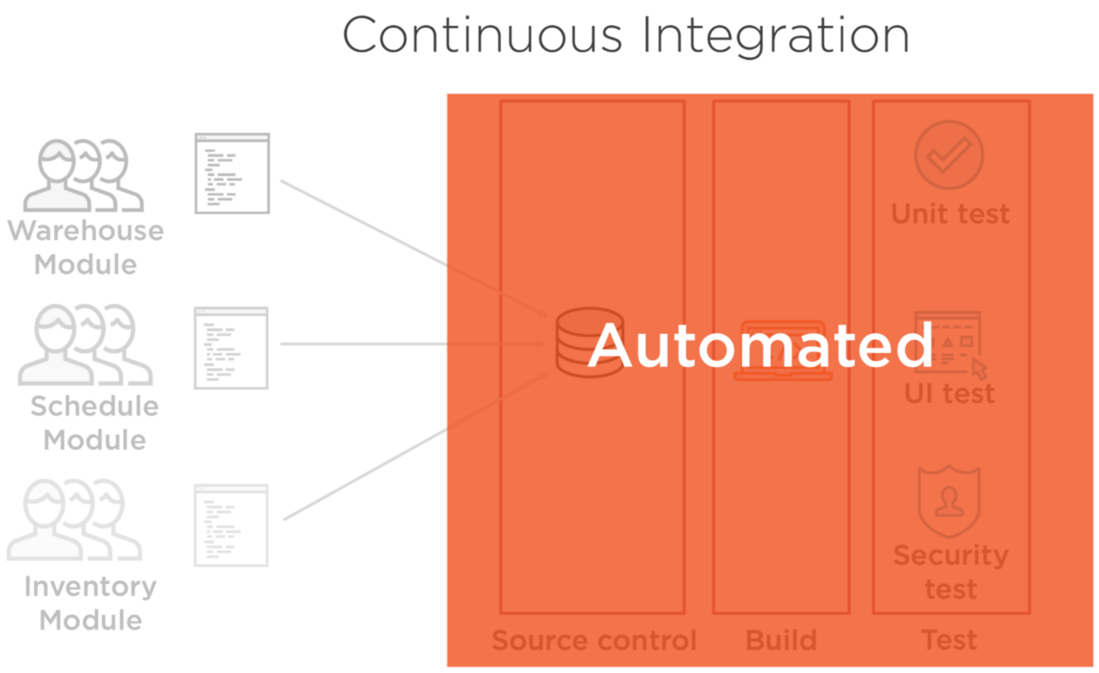


Enabled	Name ↓
✓	JUnit Plugin Allows JUnit-format test results to be published.
✓	TestNG Results Plugin This plugin integrates TestNG test reports to Jenkins.

Add post-build action → Publish TestNG Results

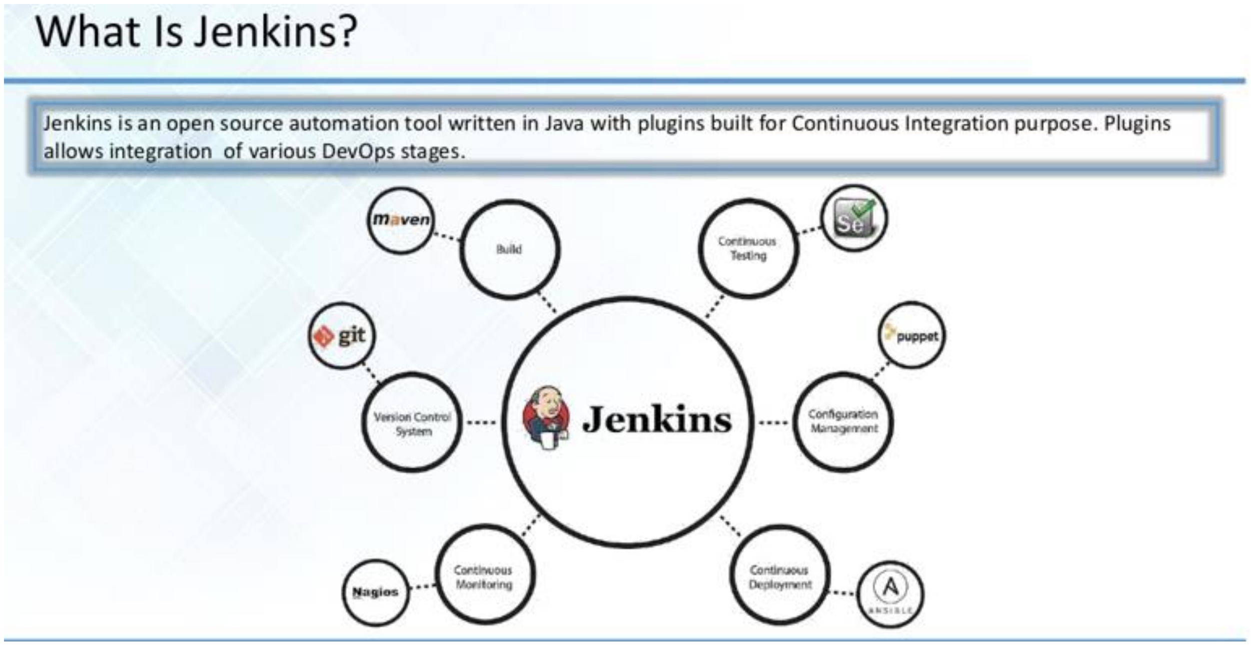


Anatomy of Build (Build process after Jenkins)



Before Jenkins	**After Jenkins**

Once all Developers had completed their assigned coding tasks, they used to commit their code all at same time. Later, Build is tested and deployed. Code commit built, and test cycle was very infrequent, and a single build was done after many days.	The code is built and test as soon as Developer commits code. Jenkin will build and test code many times during the day If the build is successful, then Jenkins will deploy the source into the test server and notifies the deployment team. If the build fails, then Jenkins will notify the errors to the developer team.
Since the code was built all at once, some developers would need to wait until other developers finish coding to check their build	The code is built immediately after any of the Developer commits.
It is not an easy task to isolate, detect, and fix errors for multiple commits.	Since the code is built after each commit of a single developer, it's easy to detect whose code caused the built to fail
Code build and test process are entirely manual, so there are a lot of chances for failure.	Automated build and test process saving timing and reducing defects.
The code is deployed once all the errors are fixed and tested.	The code is deployed after every successful build and test.
Development Cycle is slow	The development cycle is fast. New features are more readily available to users. Increases profits

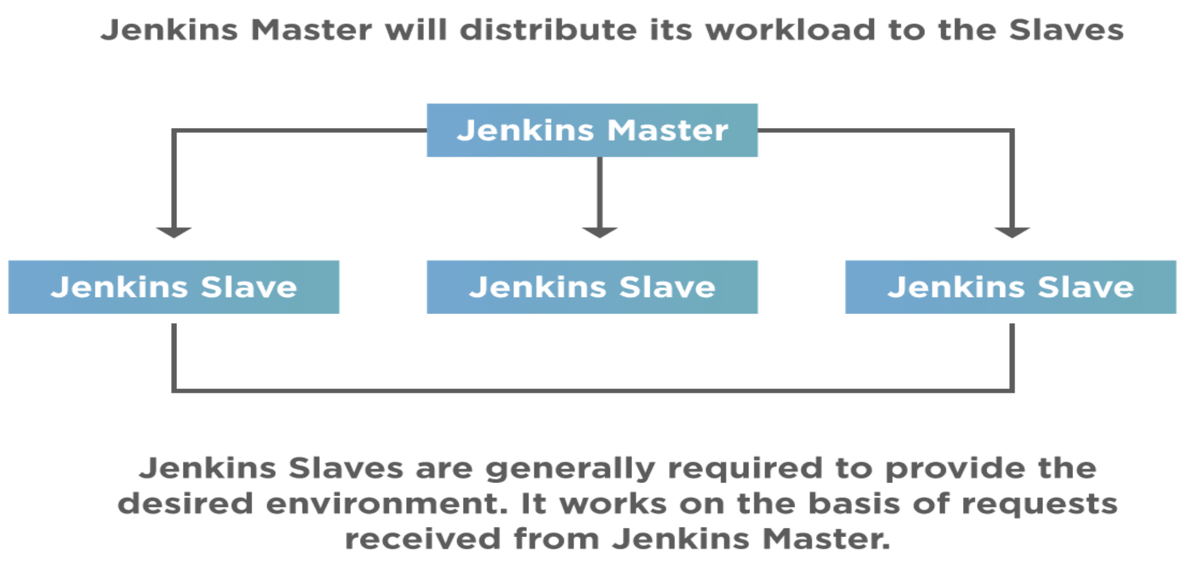


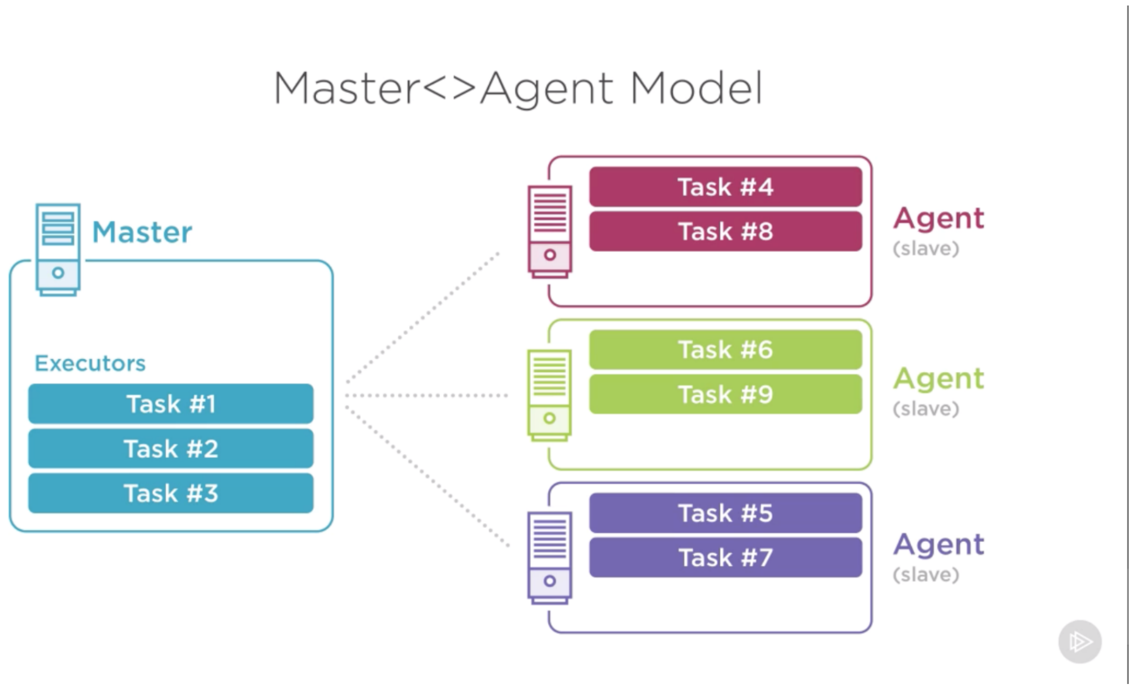
What Is Jenkins?

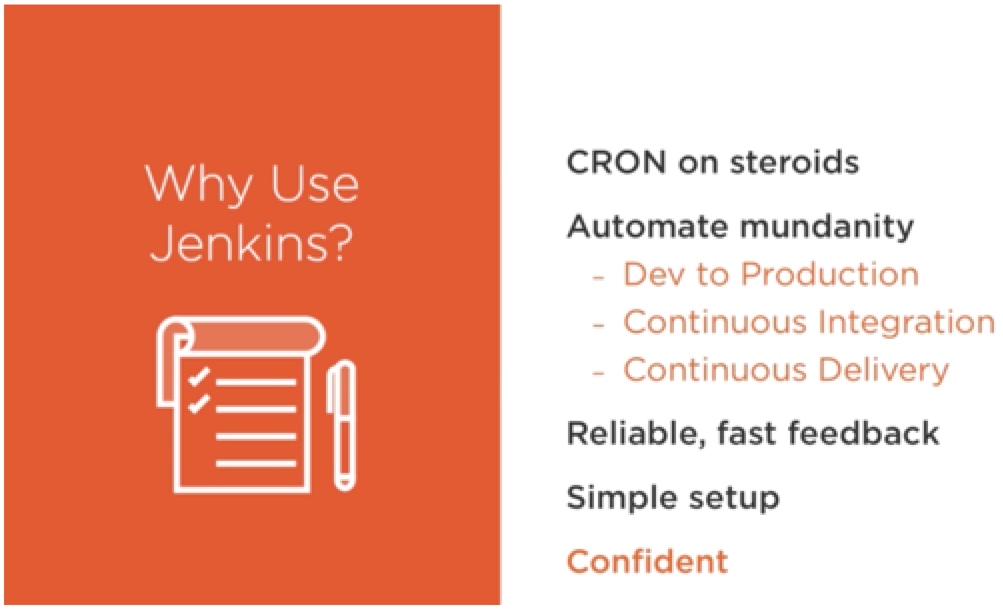
Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose. Plugins allows integration of various DevOps stages.

Jenkins Distributed Architecture

The below image shows a basic setup for the Jenkins distributed architecture:


Jenkins Master will distribute its workload to the Slaves
Jenkins Master
Jenkins Slave
Jenkins Slave
Jenkins Slave
Jenkins Slaves are generally required to provide the desired environment. It works on the basis of requests received from Jenkins Master.





Why Use Jenkins?

CRON on steroids

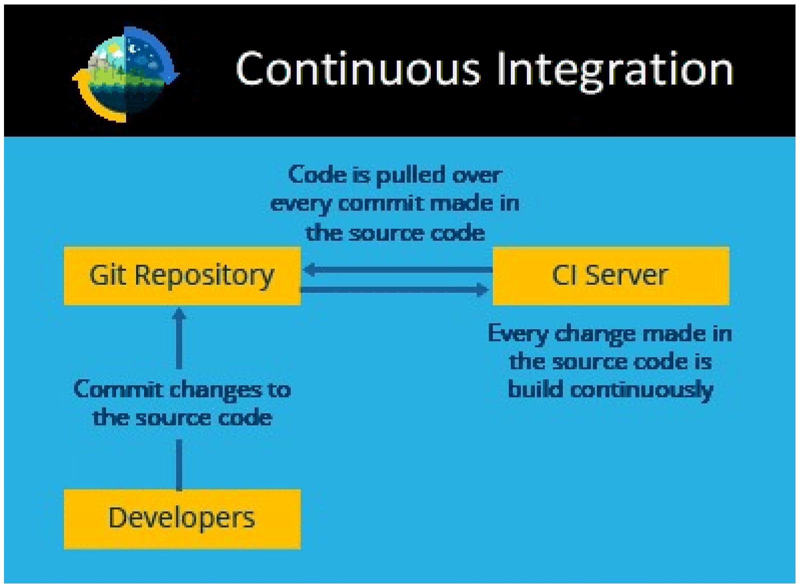
Automate mundanity
- Dev to Production
- Continuous Integration
- Continuous Delivery

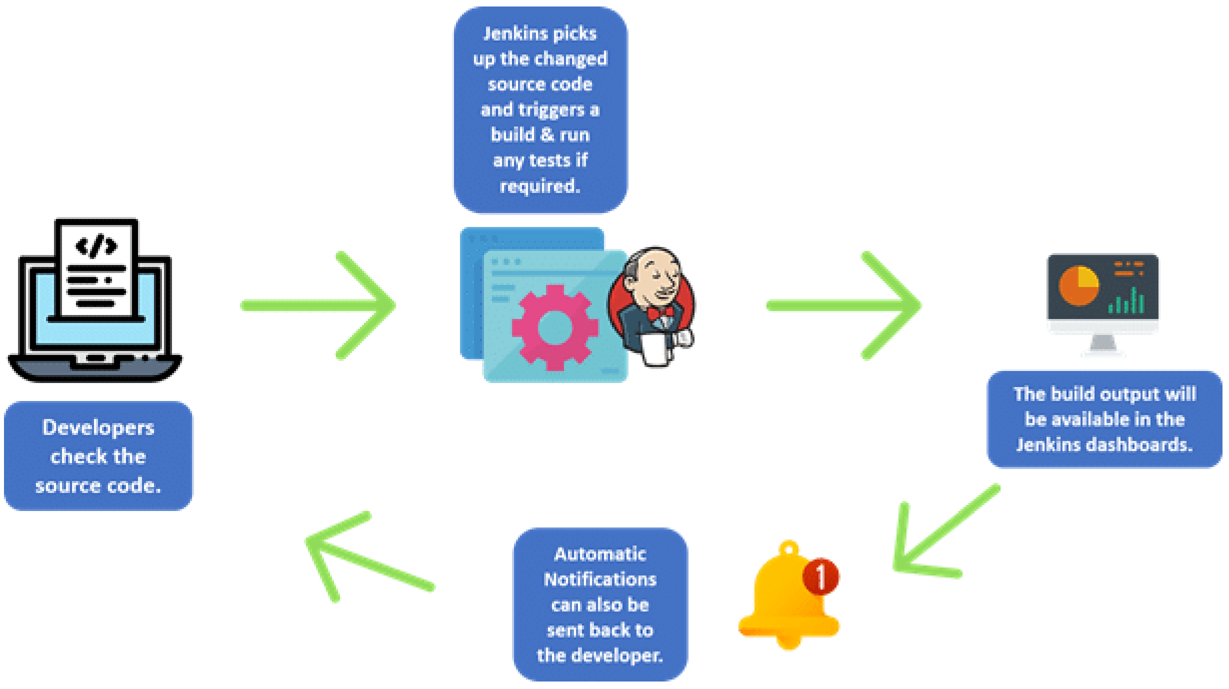
Reliable, fast feedback

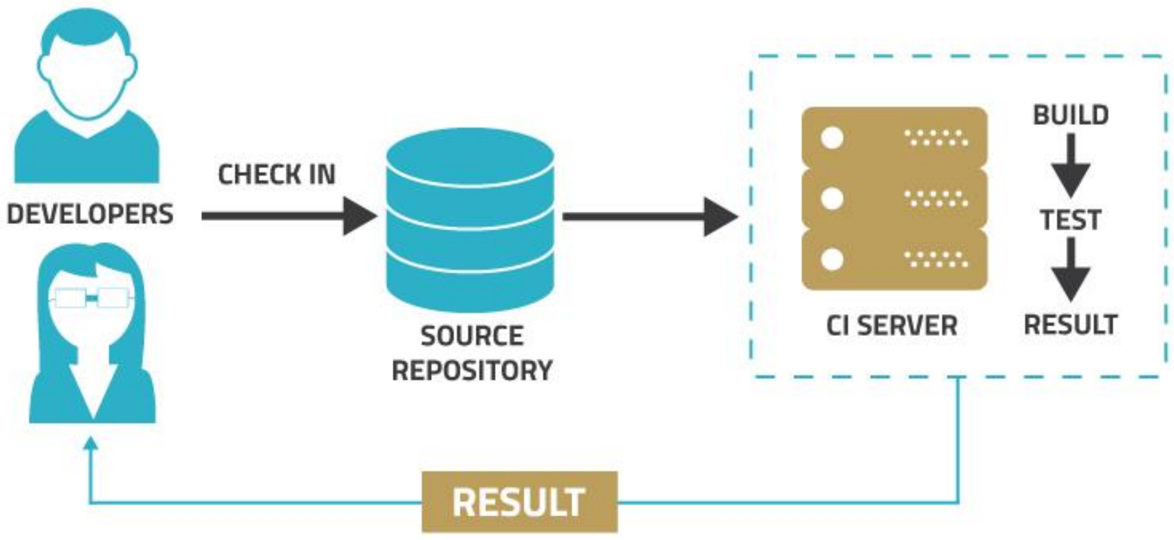
Simple setup

Confident

Continuous Integration


Continuous Integration
Code is pulled over every commit made in the source code
Git Repository
CI Server
Every change made in the source code is build continuously
Commit changes to the source code
Developers


Jenkins picks up the changed source code and triggers a build & run any tests if required.
Developers check the source code.
The build output will be available in the Jenkins dashboards.
Automatic Notifications can also be sent back to the developer.
1



Continuous integration is a software development practice in which developers are required to frequently commit changes to the source code in a shared repository. Each commit is then continuously pulled & built. Jenkins is an open-source, Continuous Integration (CI) tool, written in **Java**. It continuously pulls, builds and tests any code commits made by a developer with the help of plugins.

CI principles

Have a single place where all the code lives

Everyone commits to the mainline every day

Automate the build process
- Fix broken build immediately
- Make and keep the build fast

Every commit triggers a build

Automate the testing process

Everyone has access to the latest result

Everyone can see everything

CI benefits

Integration takes less effort

Issues will come up more early

Automation means less issues

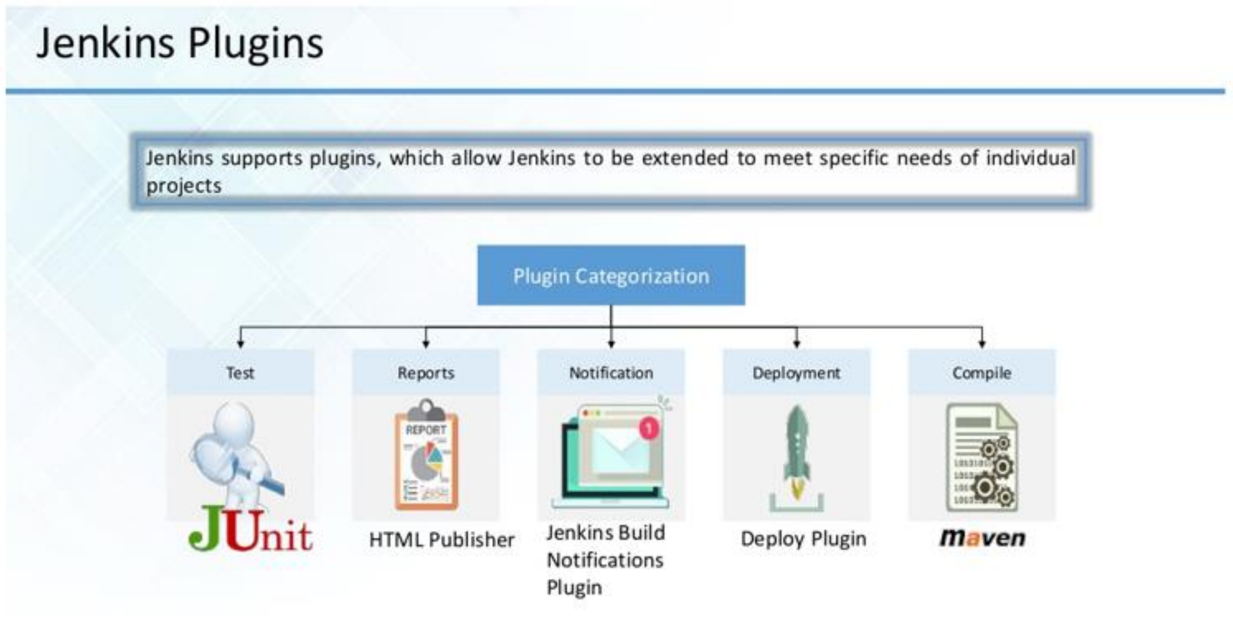
The process is more visible

Improved team communication

Short integration iterations means more flexibility

The code is **ready to be delivered** more often

Jenkins Plugins


Jenkins Plugins
Jenkins supports plugins, which allow Jenkins to be extended to meet specific needs of individual projects
Plugin Categorization
Test
Reports
Notification
Deployment
Compile
JUnit
HTML Publisher
Jenkins Build Notifications Plugin
Deploy Plugin
maven

Jenkins comes with over 2000 plugins and each plugin has unique functionality.

Step to install the plugins

Jenkins Dashboard -> Manage Jenkins -> Manage Plugins -> Available

In the filter text field enter the name of the plugin you want to install.

How To Install Manually Jenkins Plugin

Step 1: First download plugin from Jenkins plugin directory. https://updates.jenkins-ci.org/download/plugins/

Step 2: Here you find your desired plugin and clicked on plugin name, now .hpi file will downloaded.

Step 3: Now open Jenkins and go to Manage Jenkins > Manage Plugins > Advance configuration(tab)

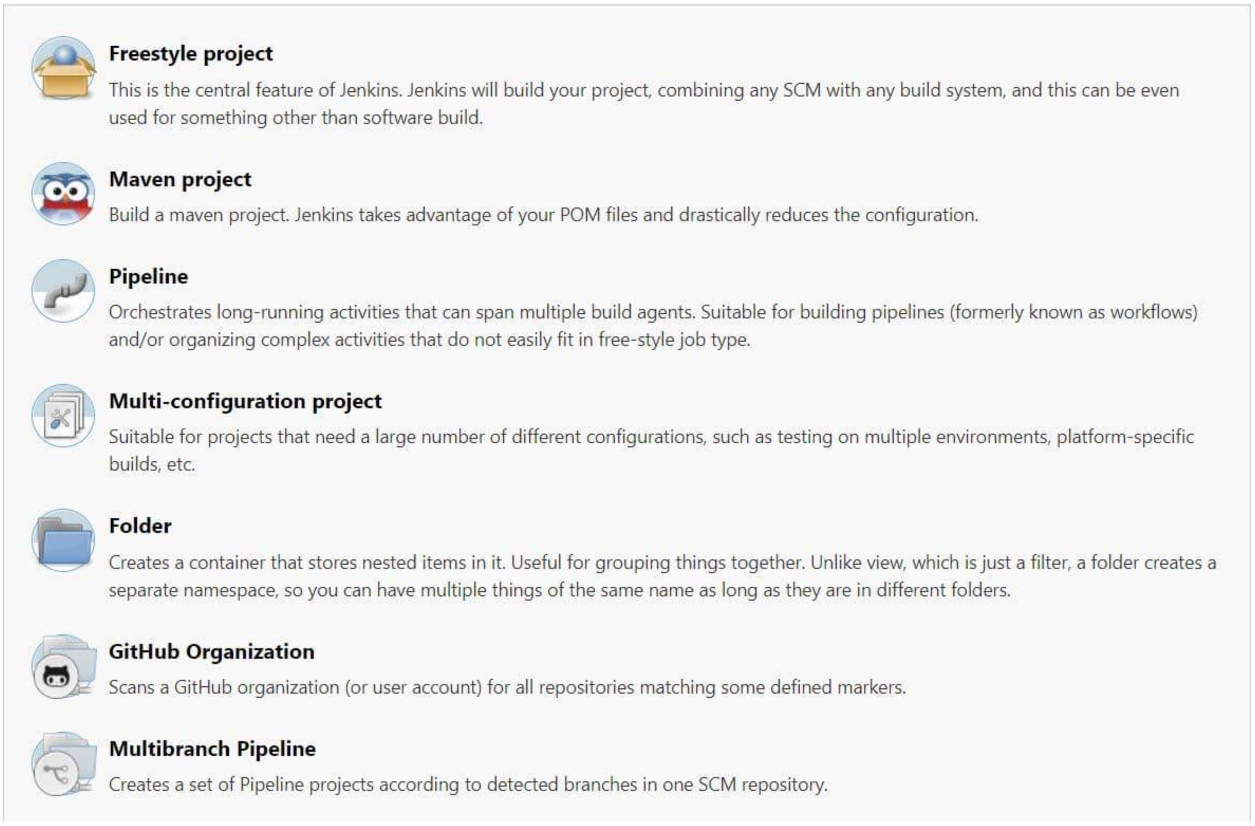
Step 4: Upload your-plugin.hpi file and save.

Step 5: Restart Jenkins.

Different Types of Jenkins Jobs

Jenkins provides the option of choosing from different types of jobs to build your project.

Below are the types of jobs you can choose from:


Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

- **Freestyle**

  Freestyle build jobs are general-purpose build jobs, which provides maximum flexibility. It can be used for any type of project.
- **Pipeline**

  This project runs the entire software development workflow as code. Instead of creating several jobs for each stage of software development, you can now run the entire workflow as one code.
- **Multiconfiguration**

  The multiconfiguration project allows you to run the same build job on different environments. It is used for testing an application in different environments.
- **Folder**

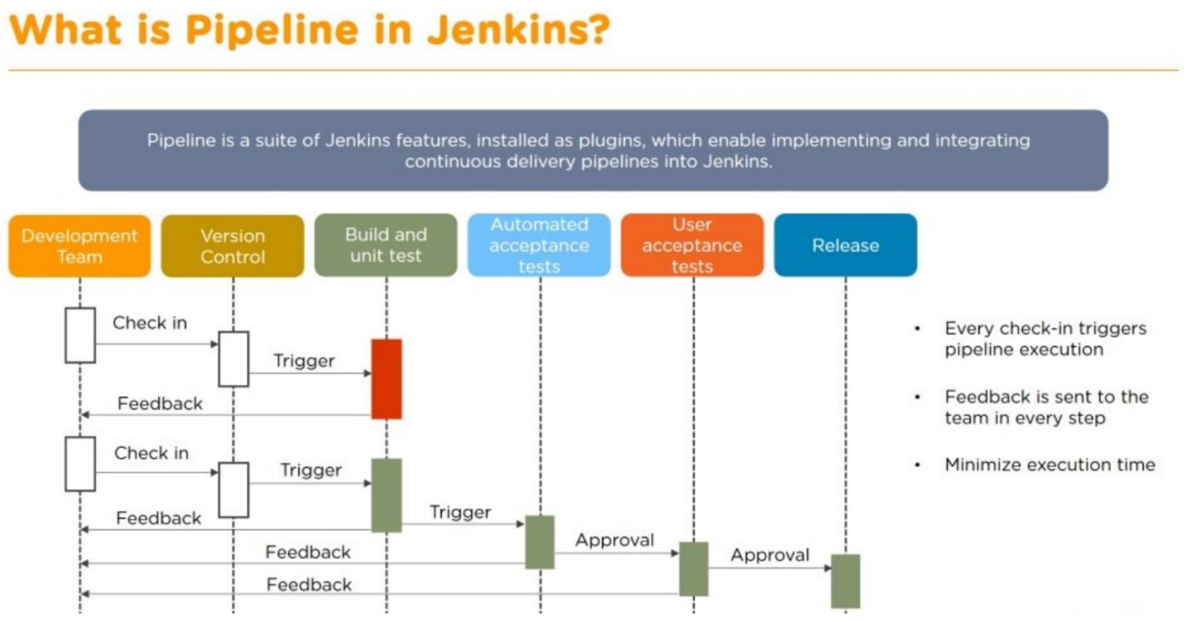
  This project allows users to create folders to organize and categorize similar jobs in one folder or subfolder.

- **GitHub Organization**

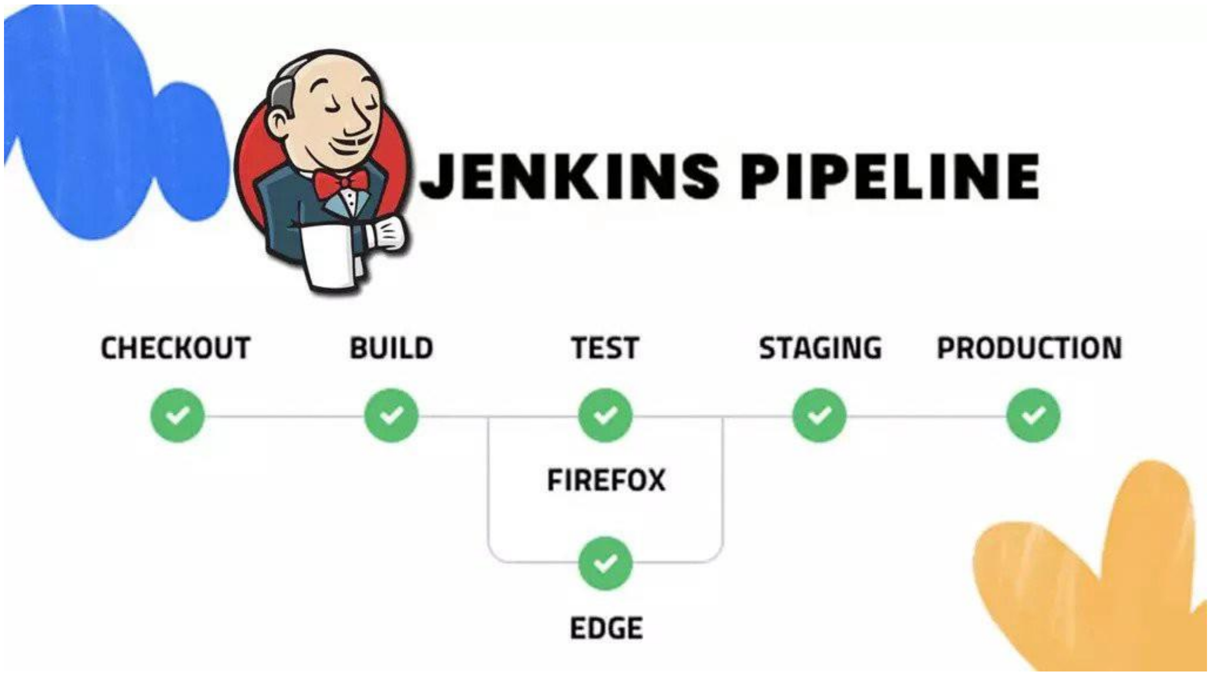
  This project scans your entire GitHub organization and creates Pipeline jobs for each repository containing a Jenkinsfile
- **Multibranch Pipeline**

  This project type lets you implement different Jenkinsfiles for different branches of the same project.

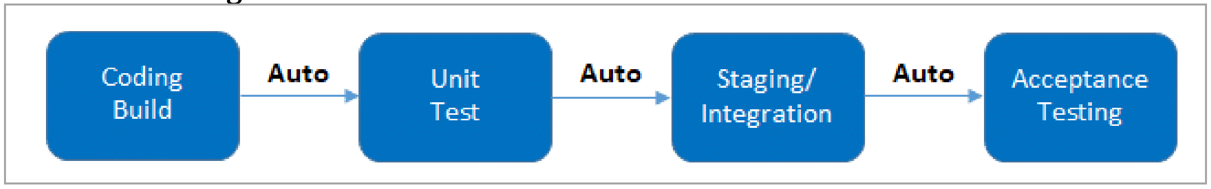
Jenkins Pipeline


What is Pipeline in Jenkins?
Pipeline is a suite of Jenkins features, installed as plugins, which enable implementing and integrating continuous delivery pipelines into Jenkins.
Development Team
Version Control
Build and unit test
Automated acceptance tests
User acceptance tests
Release
Check in
Trigger
Feedback
Check in
Trigger
Feedback
Trigger
Feedback
Approval
Feedback
Approval
Every check-in triggers pipeline execution
Feedback is sent to the team in every step
Minimize execution time

Jenkins pipeline is a single platform that runs the entire *pipeline as code.* Instead of building several jobs for each phase, you can now code the entire workflow and put it in a Jenkinsfile.

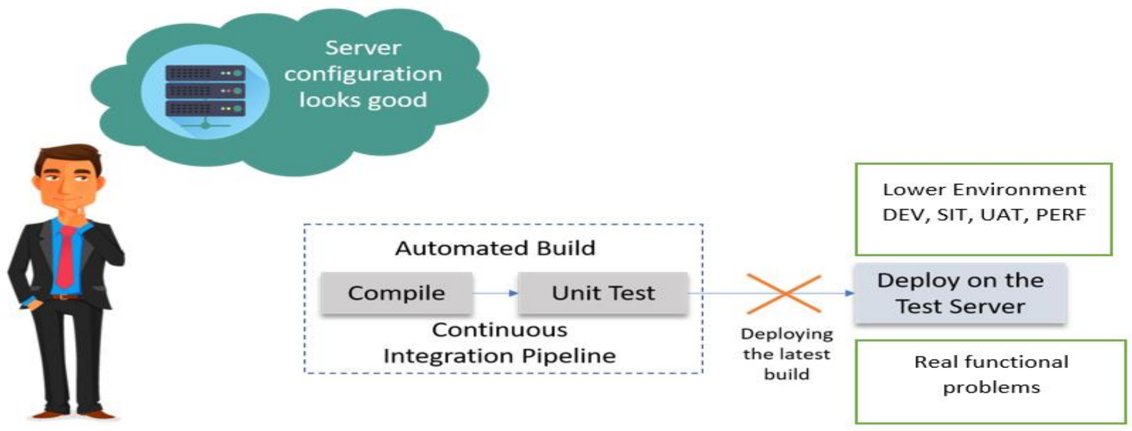


Continuous Integration:



(Involves keeping the latest copy of the source code at a commonly shared hub where all the developers can check to fetch out the latest change in order to avoid conflict**)**

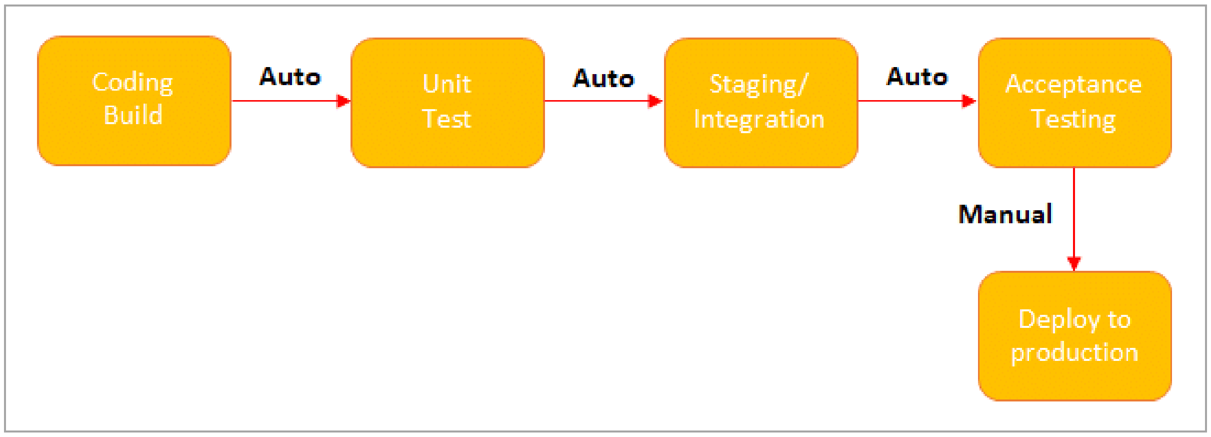
Why We Need Continuous Delivery when we have continuous integration?


Server configuration looks good
Automated Build
Compile
Unit Test
Continuous Integration Pipeline
Deploying the latest build
Lower Environment DEV, SIT, UAT, PERF
Deploy on the Test Server
Real functional problems

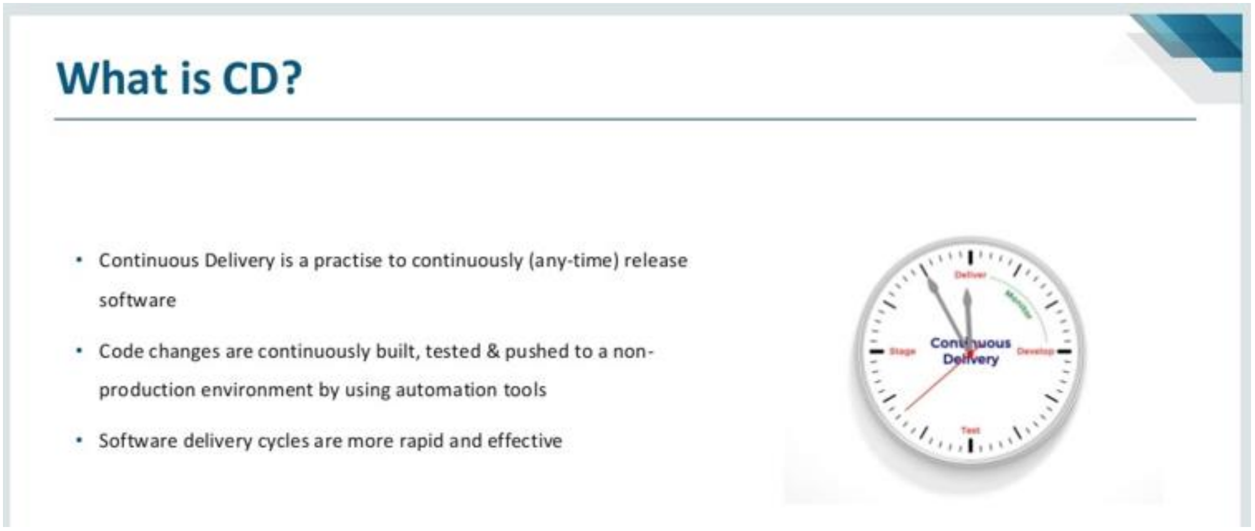
- Continuous Integration pipelines gives automated builds it includes Unit Testing as well.

- identify the real functional problems.
- Deploying the application on the test environment is a complex, manually intensive process that was quite prone to error. This meant that every attempt at deployment was a new experiment — a manual, error-prone process.
- built a Continuous Delivery pipeline, in order to make sure that the application is seamlessly deployed on the production environment, by making sure that the application works fine when deployed on the test server (Lower environment) which is a replica of the production server.

Continuous Delivery:



(Manual Deployment to Production. Does not involve every change to be deployed.**)**



What is CD?

- Continuous Delivery is a practise to continuously (any-time) release software
- Code changes are continuously built, tested & pushed to a non-production environment by using automation tools
- Software delivery cycles are more rapid and effective

Continuous Delivery

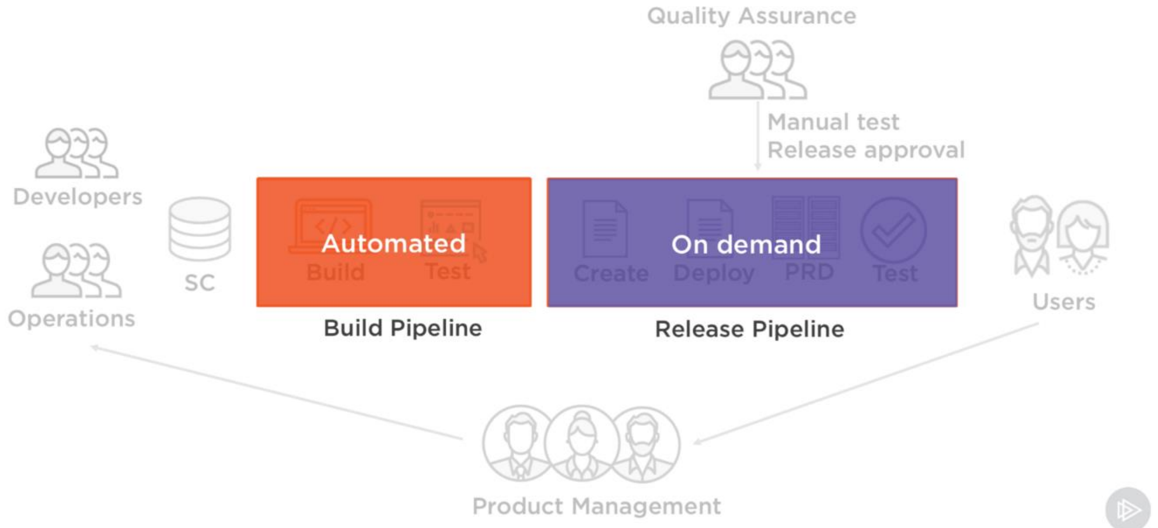
Continuous Delivery

Release Often

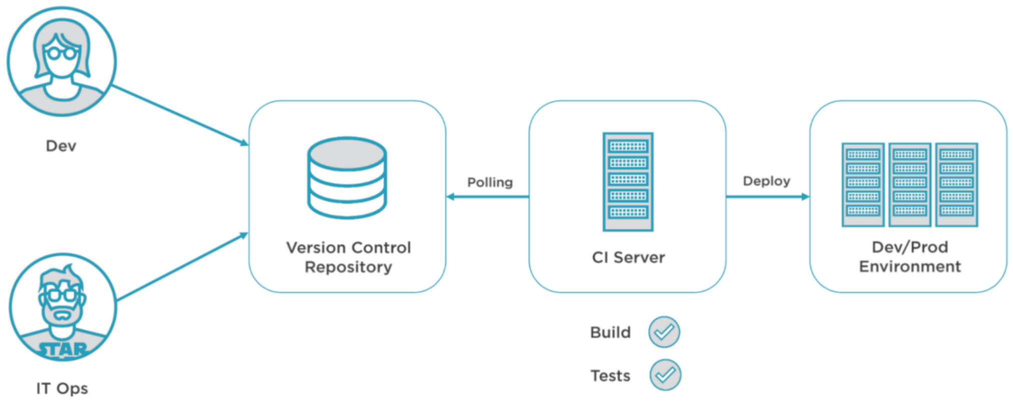
Release Faster

Greater Reliability

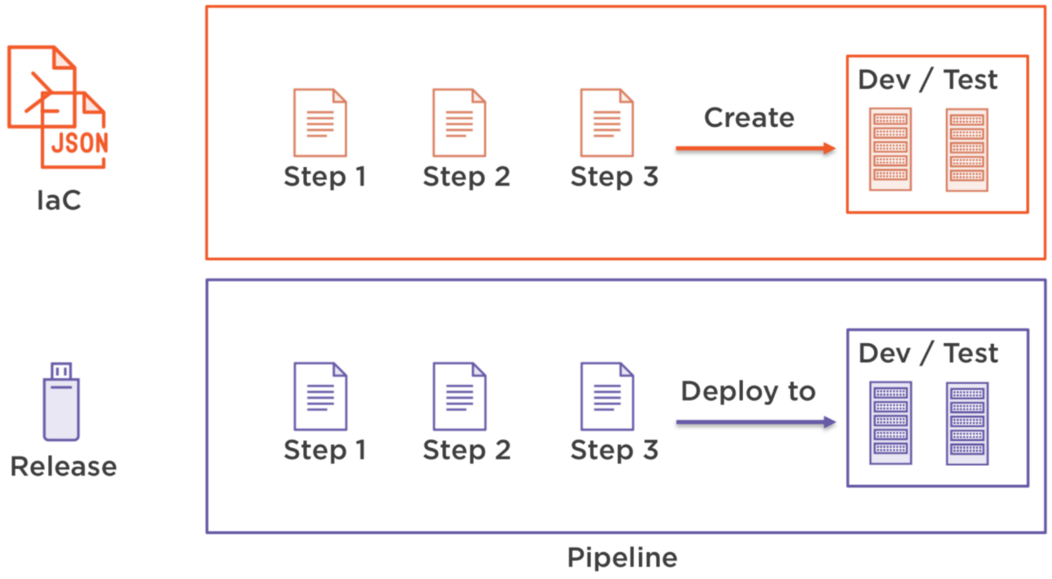
Continuous Delivery



Continuous Delivery (CD)



Continuous Delivery



CD principles

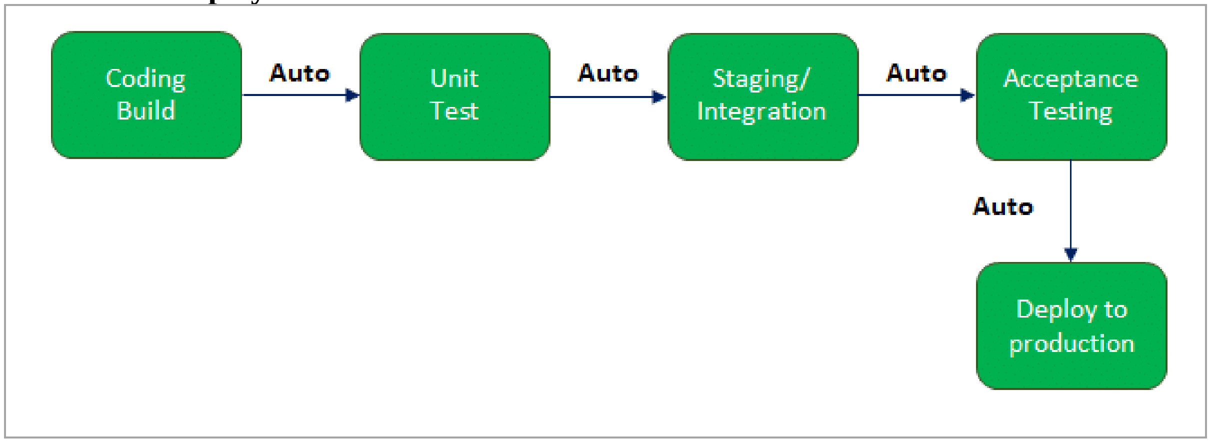
- Have continuous integration in place
- Development and Operations should work well together
- Treat infrastructure as a code artifact
- Automate the environment creation process
- Automate the release process
  - Automate acceptance tests
- Include release to definition of done
- Releasing should be on-demand
- Everyone has access to the latest result
- Everyone can see everything

CD benefits

- Releasing takes less effort
- Releasing is more
  - Reliable
  - Repeatable
- Put control of release in the hands of business
- Release more often
- Get feedback earlier

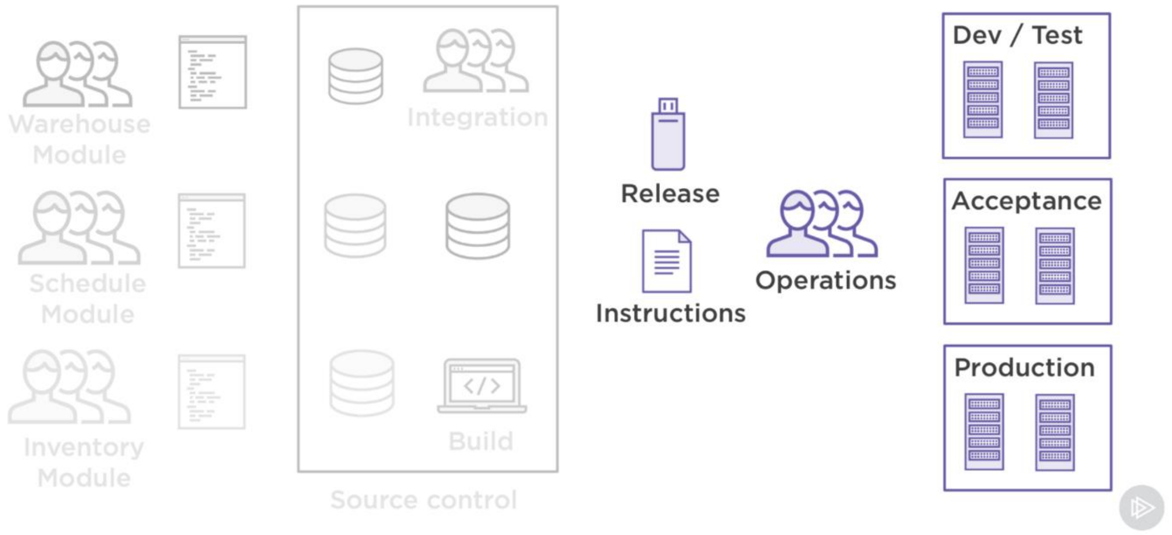
What Can CD Accomplish?
Higher Quality
Faster Delivery
Lower Costs
More Flexibility

Continuous Deployment:



(Automated Deployment to Production. Involves every change to be deployed automatically.**)**

Software Development and Deployment



Continuous Integration & Continuous Deployment Principles

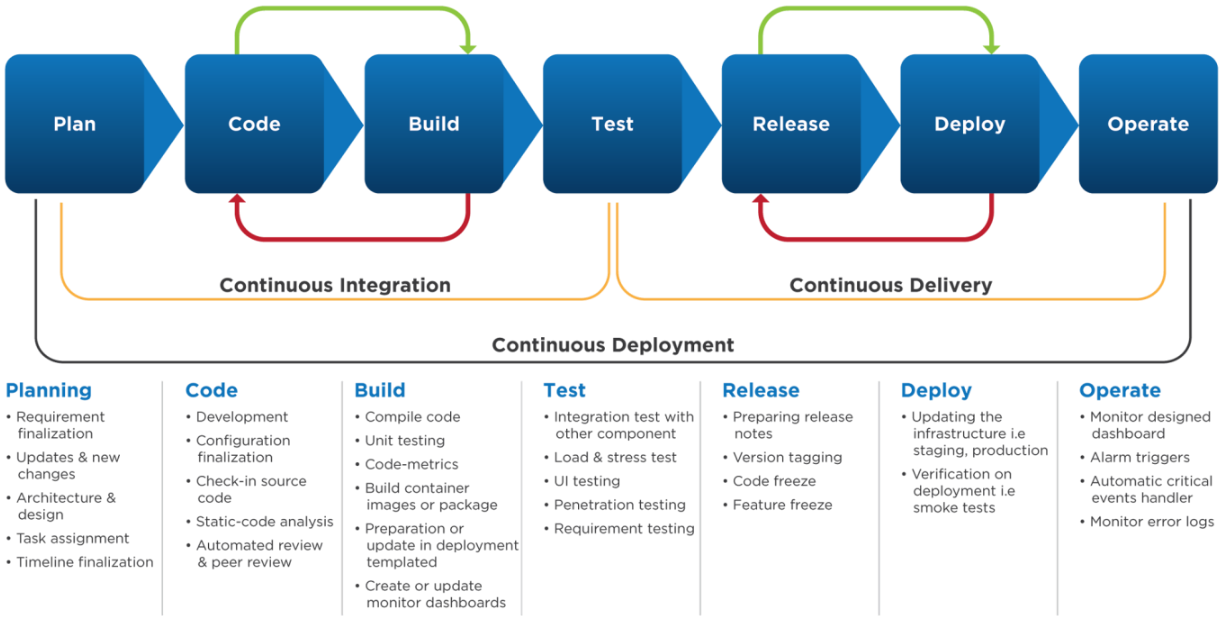
Automate everything

Define infrastructure as code

Store application and infrastructure code in version control

Unify the application and the infrastructure

Perform end-to-end automated testing



Planning
- Requirement finalization
- Updates & new changes
- Architecture & design
- Task assignment
- Timeline finalization

Code
- Development
- Configuration finalization
- Check-in source code
- Static-code analysis
- Automated review & peer review

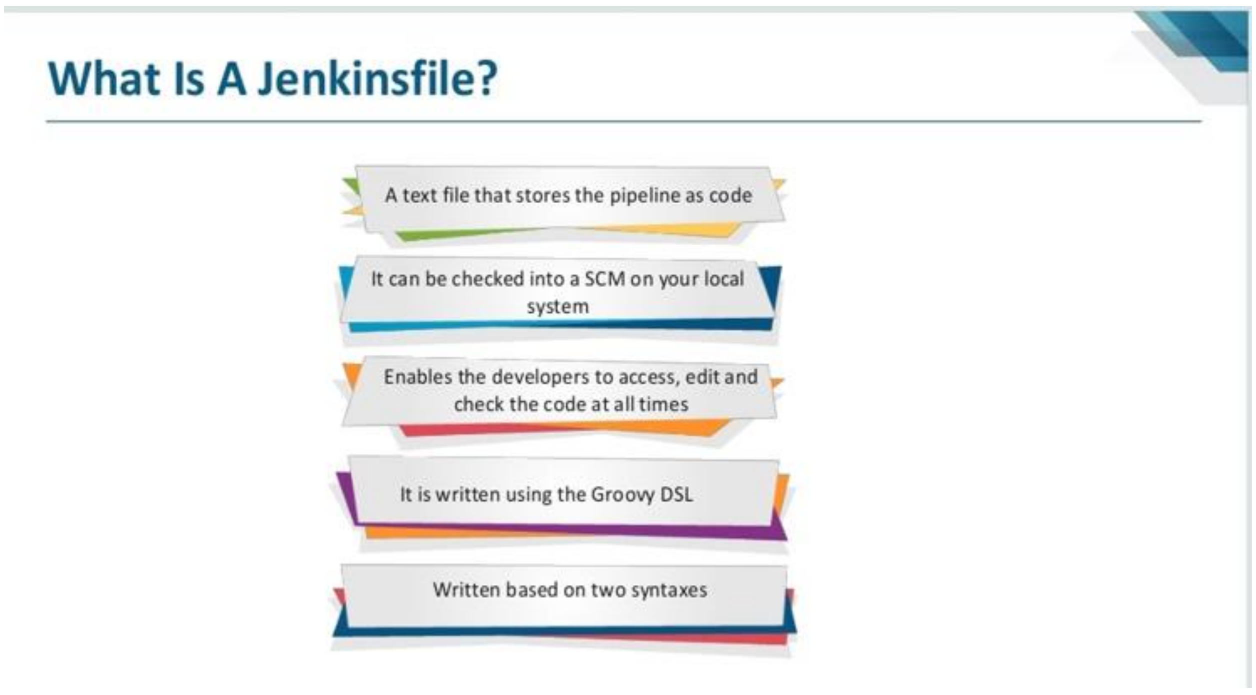
Build
- Compile code
- Unit testing
- Code-metrics
- Build container images or package
- Preparation or update in deployment templated
- Create or update monitor dashboards

Test
- Integration test with other component
- Load & stress test
- UI testing
- Penetration testing
- Requirement testing

Release
- Preparing release notes
- Version tagging
- Code freeze
- Feature freeze

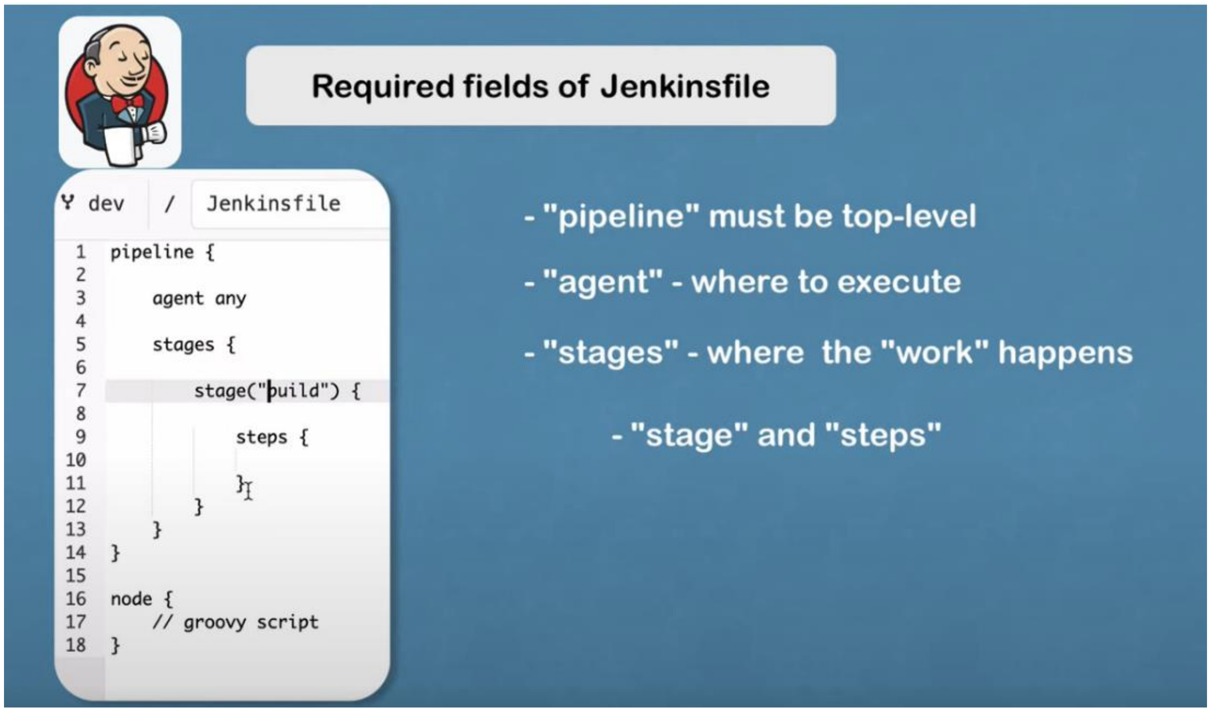
Deploy
- Updating the infrastructure i.e staging, production
- Verification on deployment i.e smoke tests

Operate
- Monitor designed dashboard
- Alarm triggers
- Automatic critical events handler
- Monitor error logs



What Is A Jenkinsfile?

- A text file that stores the pipeline as code
- It can be checked into a SCM on your local system
- Enables the developers to access, edit and check the code at all times
- It is written using the Groovy DSL
- Written based on two syntaxes



Required fields of Jenkinsfile

dev / Jenkinsfile

```
pipeline {

    agent any

    stages {

        stage("build") {

            steps {

            }
        }
    }
}

node {
    // groovy script
}
```

- "pipeline" must be top-level
- "agent" - where to execute
- "stages" - where the "work" happens
  - "stage" and "steps"

- **Scripted pipeline**

Code is written on the Jenkins UI instance and is enclosed within the node block

```
node {
    scripted pipeline code
}
```

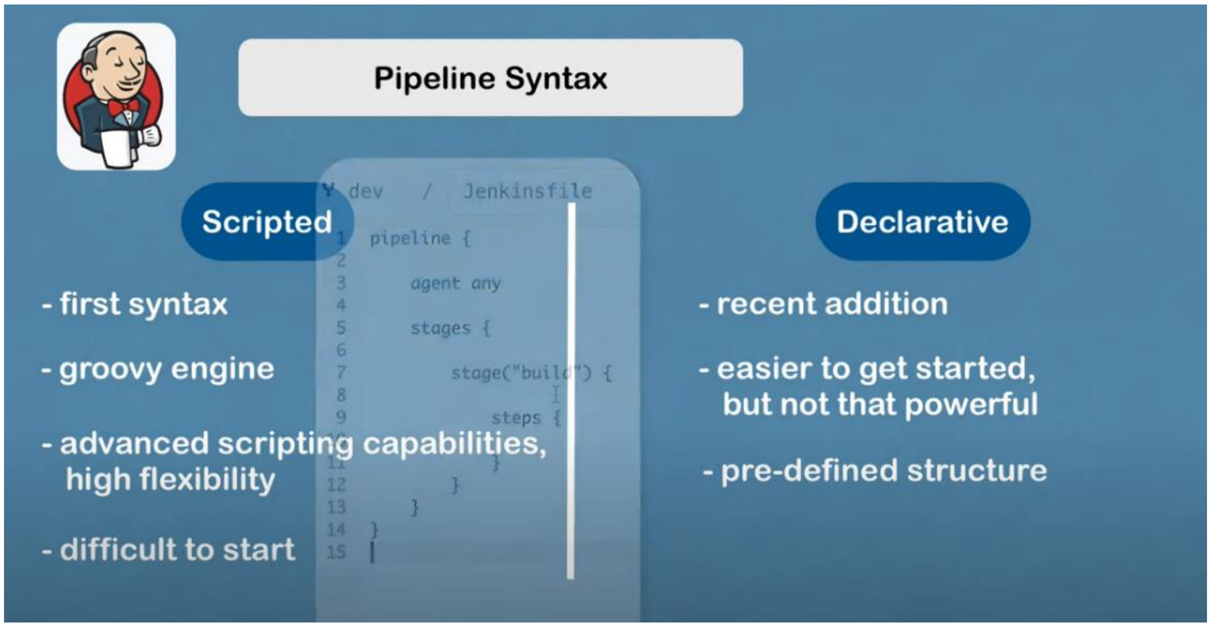
- **Declarative pipeline**

Code is written locally in a file and is checked into a SCM and is enclosed within the pipeline

block

```
pipeline {
    declarative pipeline code
}
```

Scripted Pipelines vs Declarative Pipeline



Pipeline Syntax

Scripted

- first syntax
- groovy engine
- advanced scripting capabilities, high flexibility
- difficult to start

Declarative

- recent addition
- easier to get started, but not that powerful
- pre-defined structure

dev / Jenkinsfile

```
pipeline {

    agent any

    stages {

        stage("build") {

            steps {

            }
        }
    }
}
```

Build Pipeline

Build pipeline can be used to chain several jobs together and run them in a sequence. Let's see how to install Build Pipeline:

Jenkins Dashboard -> Manage Jenkins -> Manage Plugins -> Available

In the filter text field enter the name of the plugin you want to install.

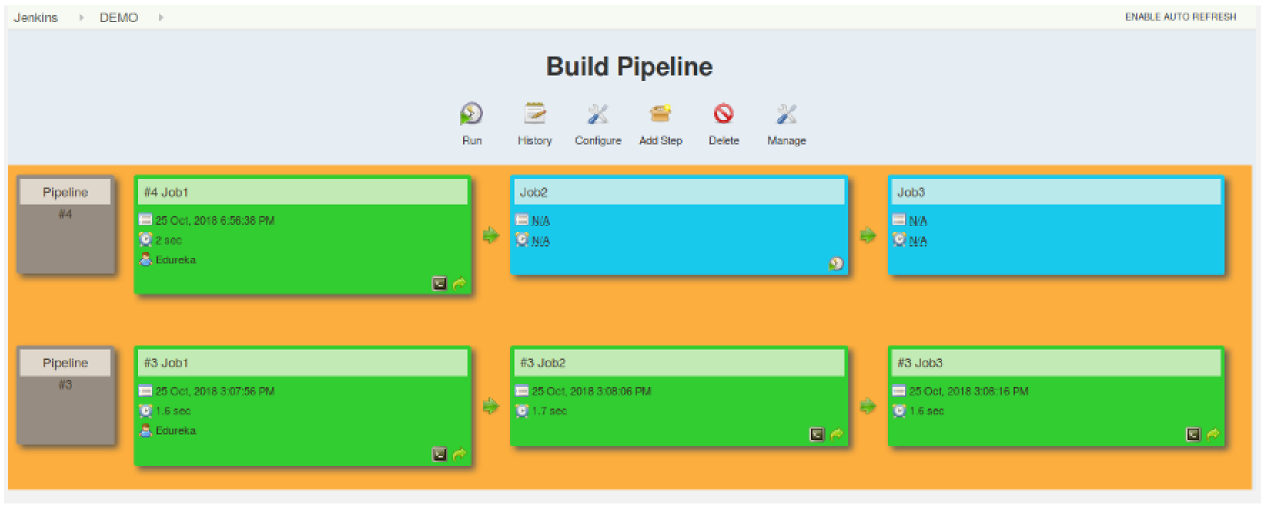
Build Pipeline Example

Step 1: Create 3 freestyle Jobs (Job1, Job2, Job3)

Step 2: Chain the 3 Jobs together
Job1 -> configure -> Post Build -> Build other projects -> Job2
Job2 -> configure -> Post Build -> Build other projects -> Job3

Step 3: Create a build pipeline view
Jenkins Dashboard -> Add view -> Enter a name -> Build pipeline view -> ok -> configure -> Pipeline flow -> Select Initial job -> Job1 -> ok

Step 4: Run the Build Pipeline



Pipeline #4

#4 Job1
25 Oct, 2018 6:56:38 PM
2 sec
Edureka

Job2
N/A
N/A

Job3
N/A
N/A

Pipeline #3

#3 Job1
25 Oct, 2018 3:07:56 PM
1.6 sec
Edureka

#3 Job2
25 Oct, 2018 3:08:06 PM
1.7 sec

#3 Job3
25 Oct, 2018 3:08:16 PM
1.6 sec

Pipeline Concepts

The below fundamentals are common to both, scripted and declarative pipeline:

1. **Pipeline:** A user-defined block which contains all the stages. It is a key part of declarative pipeline syntax.
2. **Node:** A node is a machine that executes an entire workflow. It is a key part of the scripted pipeline syntax.
3. **Agent:** instructs Jenkins to allocate an executor for the builds. It is defined for an entire pipeline or a specific stage.

It has the following parameters:

- *Any*: Runs pipeline/ stage on any available agent
- *None*: applied at the root of the pipeline, it indicates that there is no global agent for the entire pipeline & each stage must specify its own agent

- *Label*: Executes the pipeline/stage on the labelled agent.
- *Docker*: Uses docker container as an execution environment for the pipeline or a specific stage.

1. **Stages:** It contains all the work; each stage performs a specific task.
2. **Steps:** steps are carried out in sequence to execute a stage

Jenkins Pipeline syntax example

```
node {
    stage('SCM checkout') {
        //Checkout from your SCM(Source Control Management)
        //For eg: Git Checkout
    }
    stage('Build') {
        //Compile code
        //Install dependencies
        //Perform Unit Test, Integration Test
    }
    stage('Test') {
        //Resolve test server dependencies
        //Perform UAT
    }
    stage('Deploy') {
        //Deploy code to prod server
        //Solve dependency issues
    }
}
```

Create your first Jenkins Pipeline

After installing Jenkins, building jobs using the Build pipeline and briefly discussing pipeline concepts, let's see how to create a Jenkins pipeline.

Follow the below steps to create both, a scripted pipeline and a declarative pipeline:

Step 1: Log into Jenkins and select 'New Item from the Dashboard'

Step 2: Next, enter a name for your pipeline and select 'Pipeline project'. Click 'ok' to proceed

Step 3: Scroll down to the pipeline and choose if you want a Declarative or Scripted pipeline

Step 4a: If you want a Scripted pipeline, then choose 'pipeline script' and start typing your code

Step 4b: If you want a Declarative Pipeline, select 'Pipeline script from SCM' and choose your SCM and enter your repository URL

Step 5: Within the Script path is the name of the Jenkinsfile that is going to be accessed from your SCM to run. Finally click on 'apply' and 'save'

Jenkins Tips and Tricks

Start, stop and restart Jenkins

Follow the below command to start, stop and restart Jenkins through the CLI.

```
$ sudo service jenkins restart
$ sudo service jenkins stop
$ sudo service jenkins start
```

Deploy a custom build of a core plugin

Step 1: Stop Jenkins.

Step 2: Copy the custom HPI to **$Jenkins_Home/plugins**.

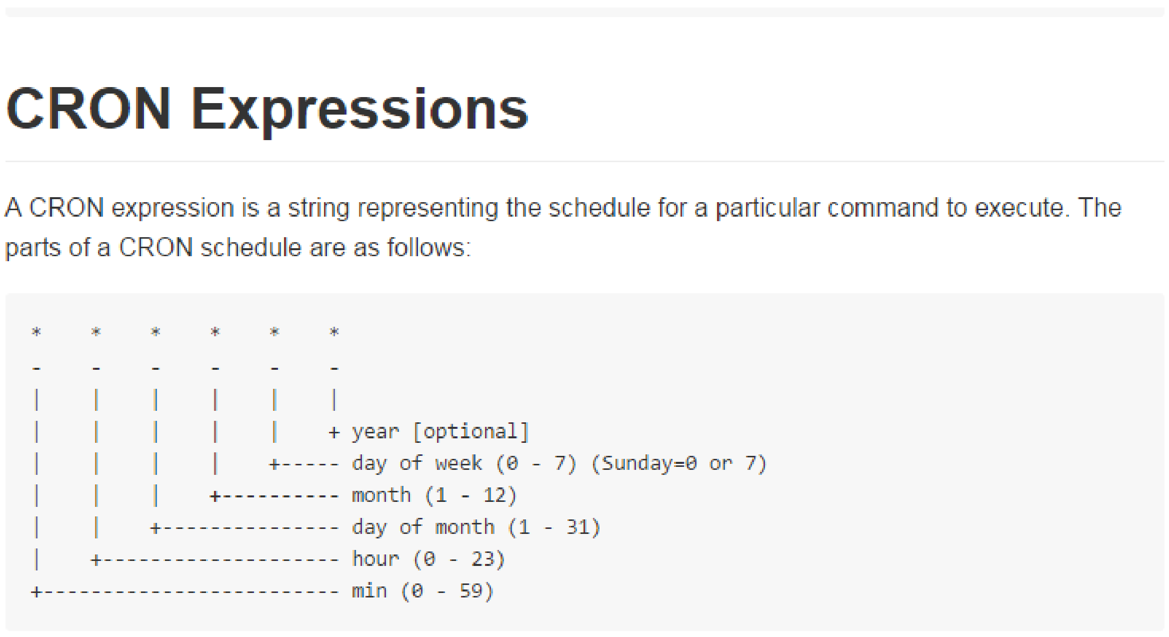
Step 3: Delete the previously expanded plugin directory.

Step 4: Make an empty file called **<plugin>.hpi.pinned**.

Step 5: Start Jenkins.

Schedule a build periodically

Jenkins uses Cron expressions to schedule a job. Each line consists of 5 fields separated by TAB or whitespace:



CRON Expressions

A CRON expression is a string representing the schedule for a particular command to execute. The parts of a CRON schedule are as follows:

```
*    *    *    *    *    *
-    -    -    -    -    -
|    |    |    |    |    |
|    |    |    |    |    + year [optional]
|    |    |    |    +----- day of week (0 - 7) (Sunday=0 or 7)
|    |    |    +---------- month (1 - 12)
|    |    +--------------- day of month (1 - 31)
|    +-------------------- hour (0 - 23)
+------------------------- min (0 - 59)
```

Syntax: (Minute Hour DOM Month DOW)

MINUTE: Minutes in one hour (0-59)

HOURS: Hours in one day (0-23)

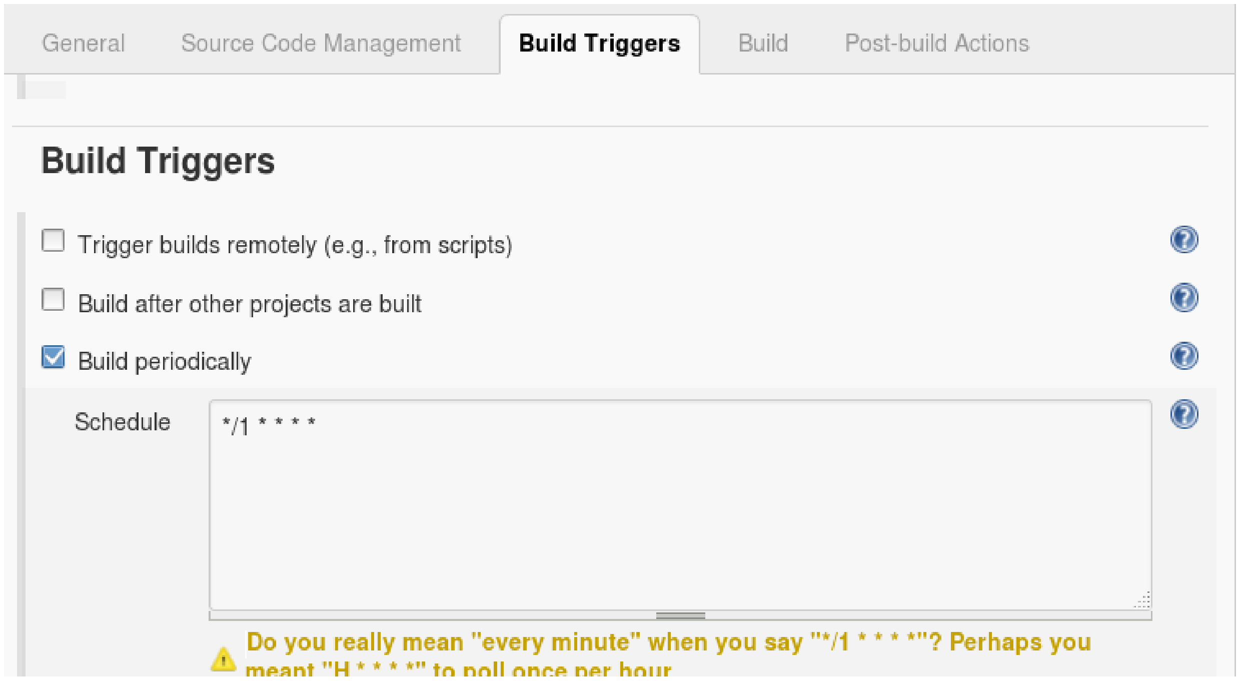
DAYMONTH: Day in a month (1-31)

MONTH: Month in a year (1-12)

DAYWEEK: Day of the week (0-7) where 0 and 7 are sunday

Example: H/2 * * * * (schedule your build for every 2 minutes)

Schedule build for every 2 minutes :
H/2 * * * * (schedules your build for every 2 minutes)

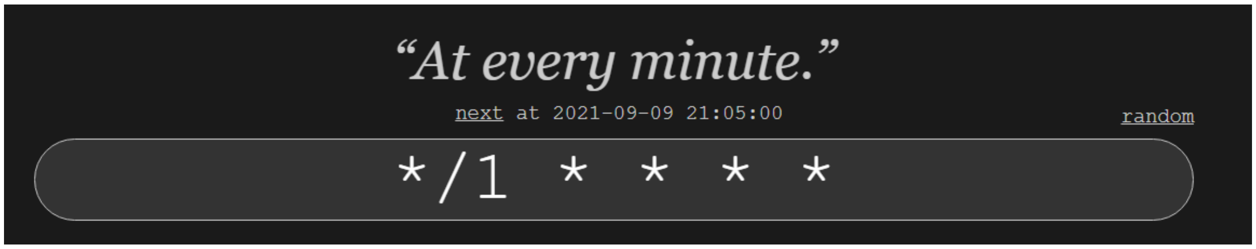


Build Triggers

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [x] Build periodically

Schedule `*/1 * * * *`

Do you really mean "every minute" when you say "*/1 * * * *"? Perhaps you meant "H * * * *" to poll once per hour



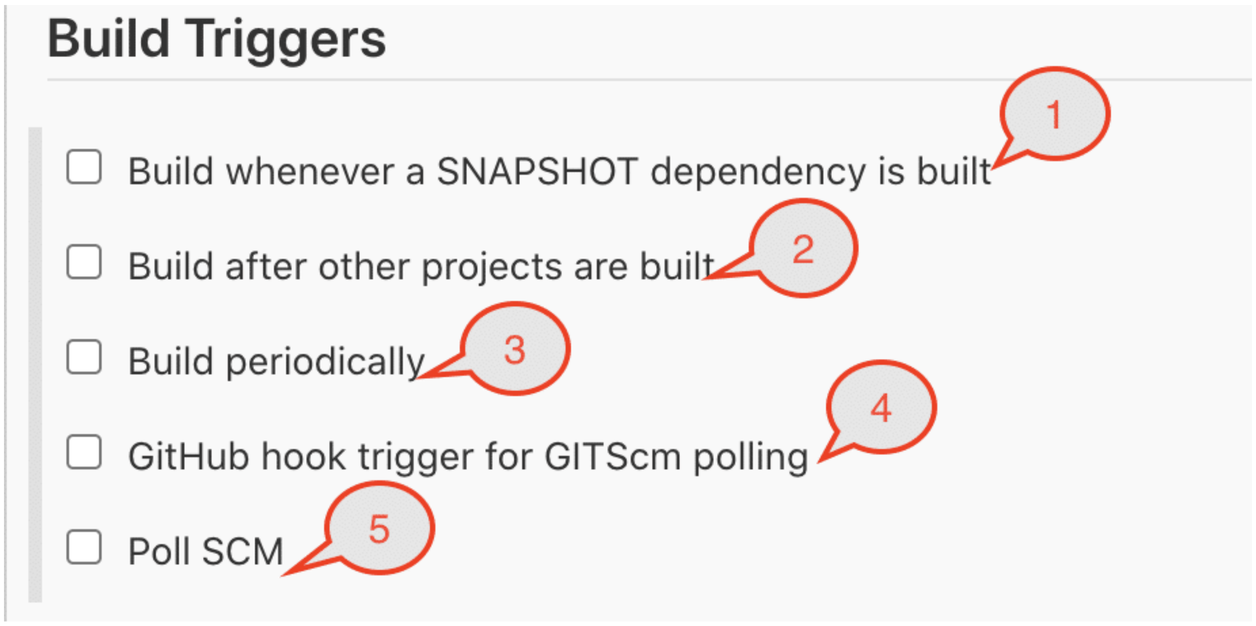
"At every minute."

next at 2021-09-09 21:05:00 random

```
*/1 * * * *
```

Triggers in Jenkins?

Trigger in Jenkins defines the way in which the pipeline should be executed frequently. PollSCM, Cron, etc are the currently available Triggers.



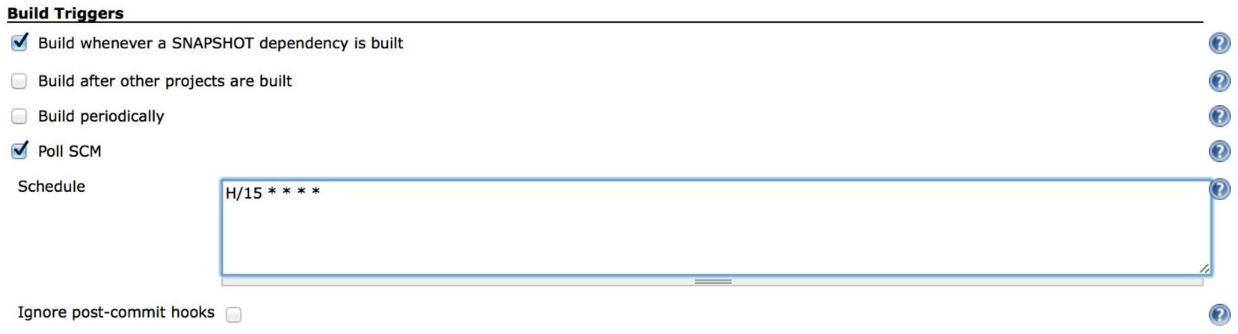
Build Triggers

- Build whenever a SNAPSHOT dependency is built
- Build after other projects are built
- Build periodically
- GitHub hook trigger for GITScm polling
- Poll SCM

You can select one or all of the above-mentioned options to trigger the build automatically. Let's understand under what all conditions these options will trigger the build:

Build Trigger Option	***Behavior***
Build whenever a SNAPSHOT dependency is built	If checked, Jenkins will parse the POMs of this project and check if any of its snapshot dependencies are built on this Jenkins. If so, Jenkins will set up a build dependency relationship so that whenever the dependency job builds, and a new SNAPSHOT jar creates, Jenkins will schedule a build of this project. This is convenient for automatically performing continuous integration. Jenkins will check the snapshot dependencies from the <dependency> element in the POM, as well as <plugin>s and <extension>s used in POMs.
Build after other projects are built	Set up a trigger so that new build schedules for this project when some other projects finish building. This is convenient for running an extensive test after a build is complete, for example. This configuration complements the ***"Build other projects"*** section in the ***"Post-build Actions"*** of an upstream project but is preferable when you want to configure the downstream project.
Build periodically	Build Periodically shall build the project periodically irrespective to whether any changes were made
GitHub hook trigger for GITScm polling	If Jenkins receives/ gets PUSH GitHub hook from repo defined in the Git SCM section, it will trigger Git SCM polling logic. In fact, polling logic belongs to Git SCM.
Poll SCM	Poll SCM polls the SCM periodically for checking if any changes/ new commits were made and shall build the project if any new commits were pushed since the last build

Poll the build at every 15minutes is here:



Build Triggers

- [✓] Build whenever a SNAPSHOT dependency is built
- [ ] Build after other projects are built
- [ ] Build periodically
- [✓] Poll SCM

Schedule: H/15 * * * *

Ignore post-commit hooks [ ]

Snippet Generator

A tool that lets users generate code for individual steps in a scripted pipeline. Let’s look at an example:

Step 1: Create a pipeline job > configure

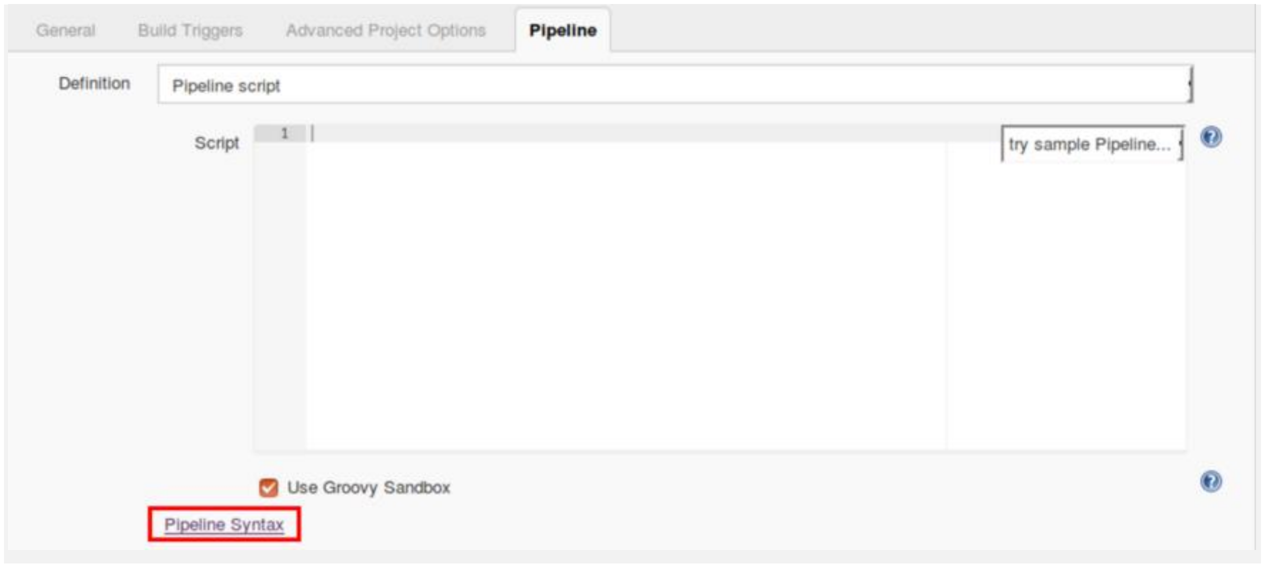
Step 2: Select pipeline script from pipeline definition

Step 3: Click on Pipeline syntax > snippet generator

Step 4: Step > select Git > enter repo URL

Step 5: Scroll down > Generate pipeline script

Step 6: Copy the script into your pipeline script UI

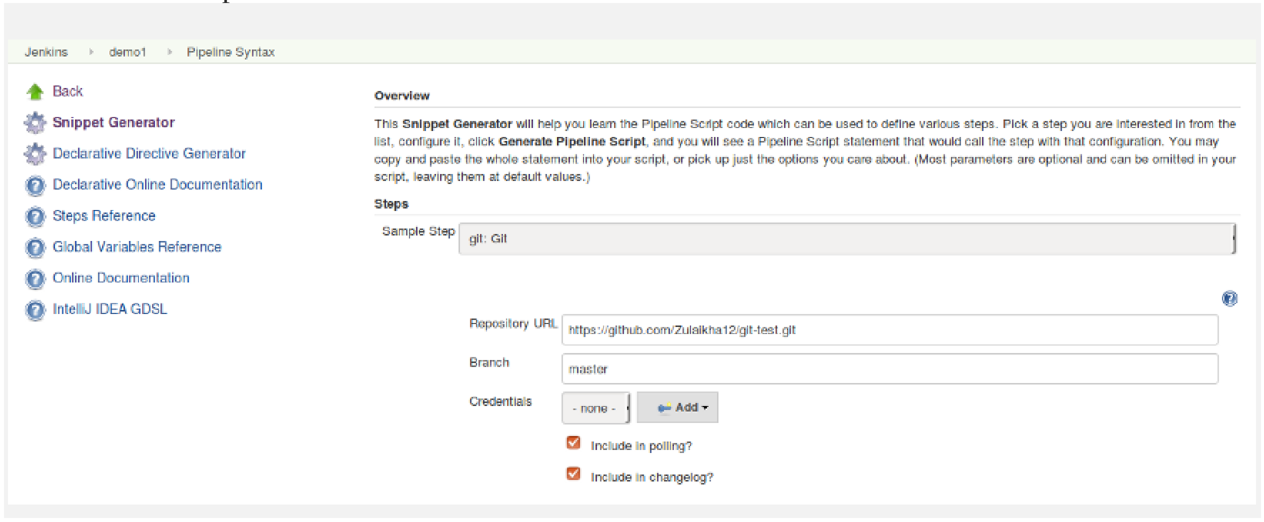


Definition: Pipeline script

Script

Use Groovy Sandbox [✓]

Below is an image of the Snippet Generator. You can select from a variety of steps and generate a code for each step.



Overview

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

Steps

Sample Step: git: Git

Repository URL: https://github.com/Zulaikha12/git-test.git

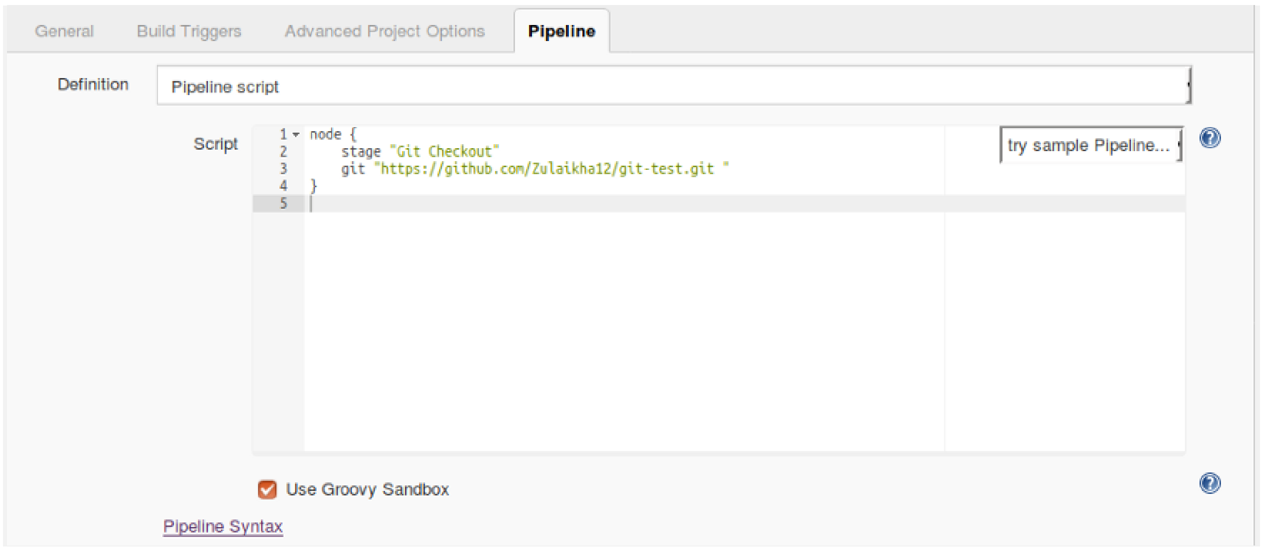
Branch: master

Credentials: - none -

Include in polling? [✓]

Include in changelog? [✓]

Below is an image of the Scripted pipeline UI with the code generated from snippet generator



Definition: Pipeline script

Script

```
node {
    stage "Git Checkout"
    git "https://github.com/Zulaikha12/git-test.git "
}
```

✓ Use Groovy Sandbox

Pipeline Syntax