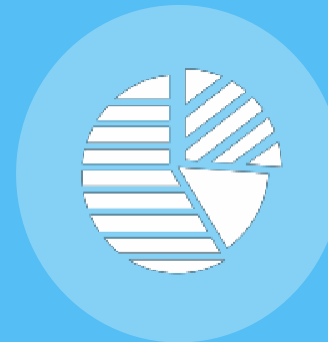
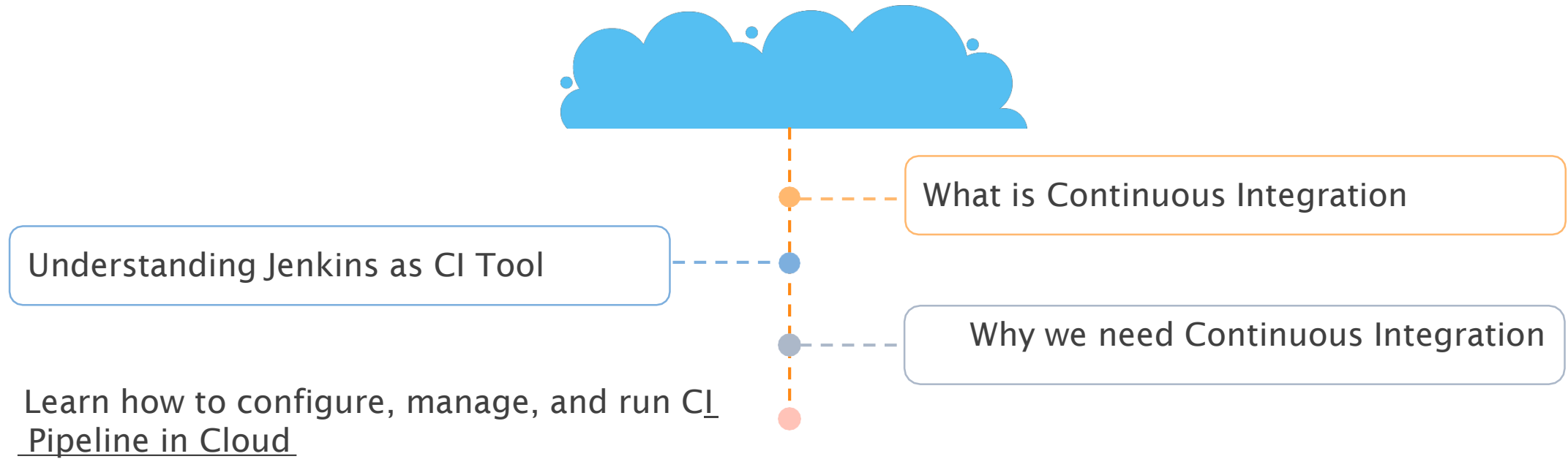


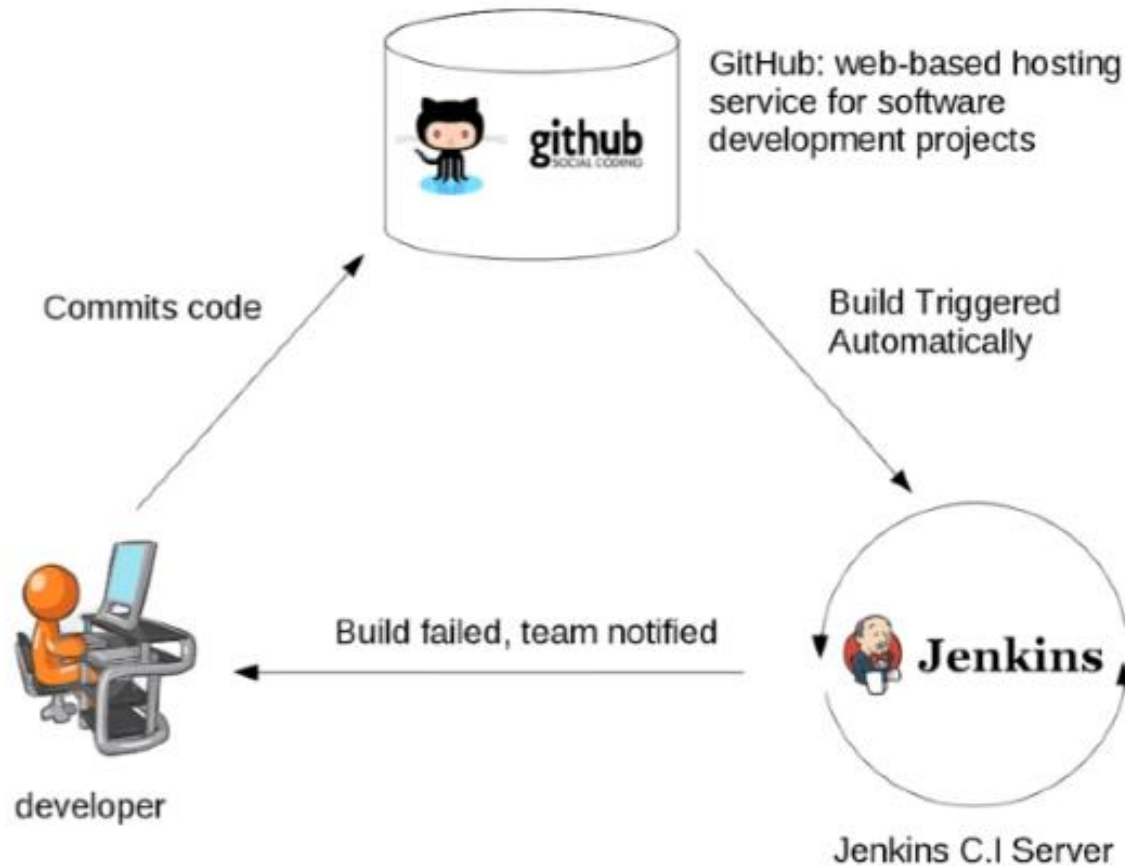
DevOps

Continuous Integration and Continuous Delivery using Jenkins
Prakash Kumar : DevOps Trainer



What's in It for Me

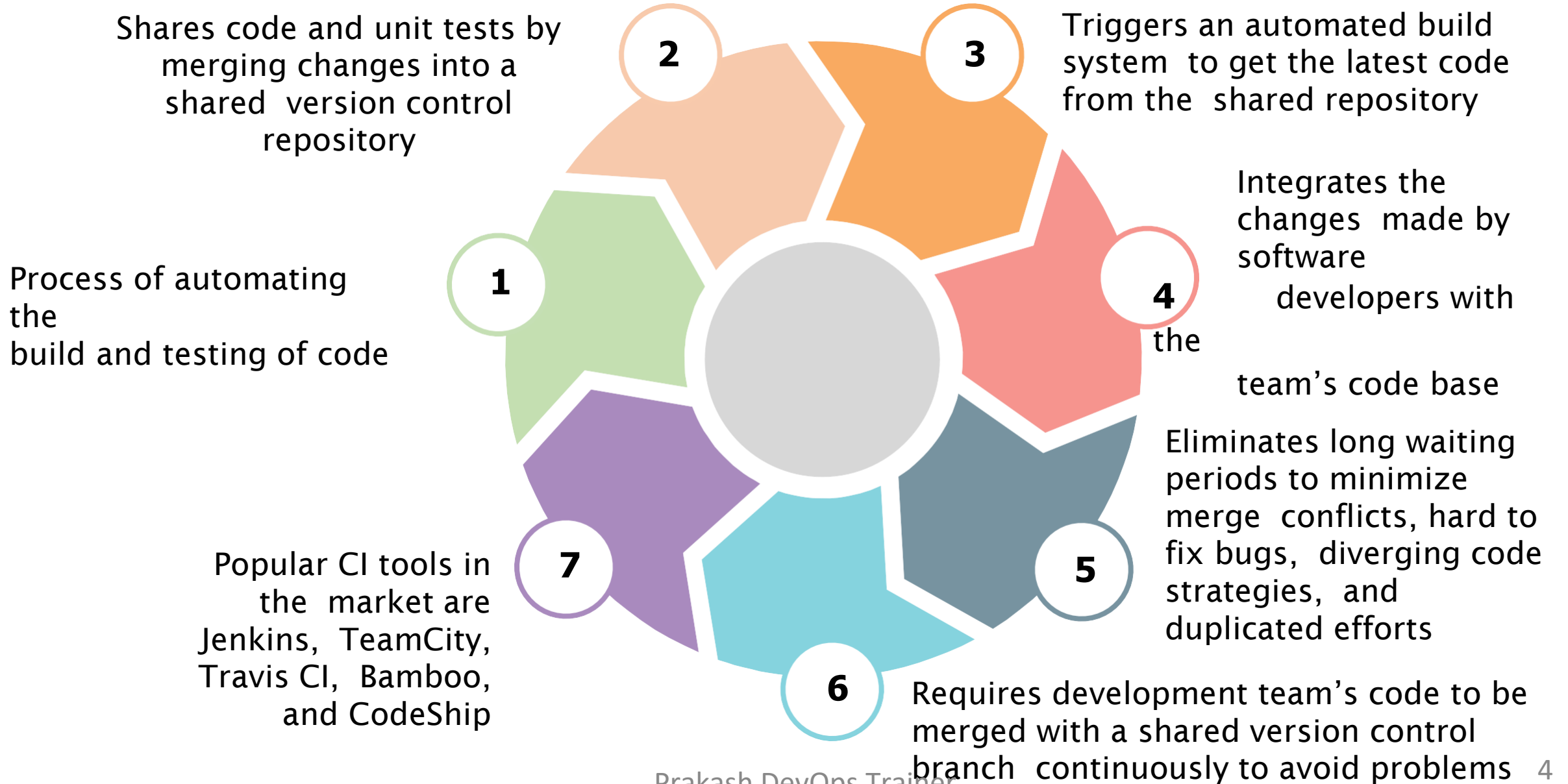




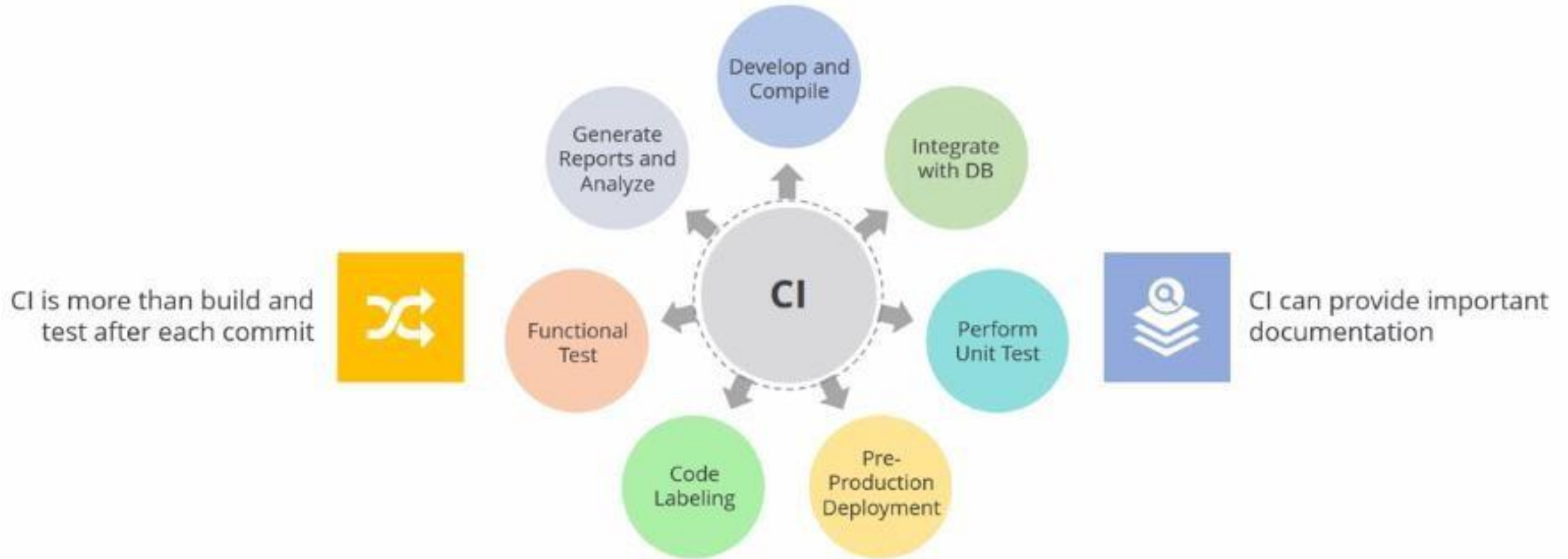
What is Continuous Integration?

- Developers commit code to a shared repository on a regular basis.
- Version control system is being monitored. When a commit is detected, a build will be triggered automatically.
- If the build is not green, developers will be notified immediately.

Continuous Integration

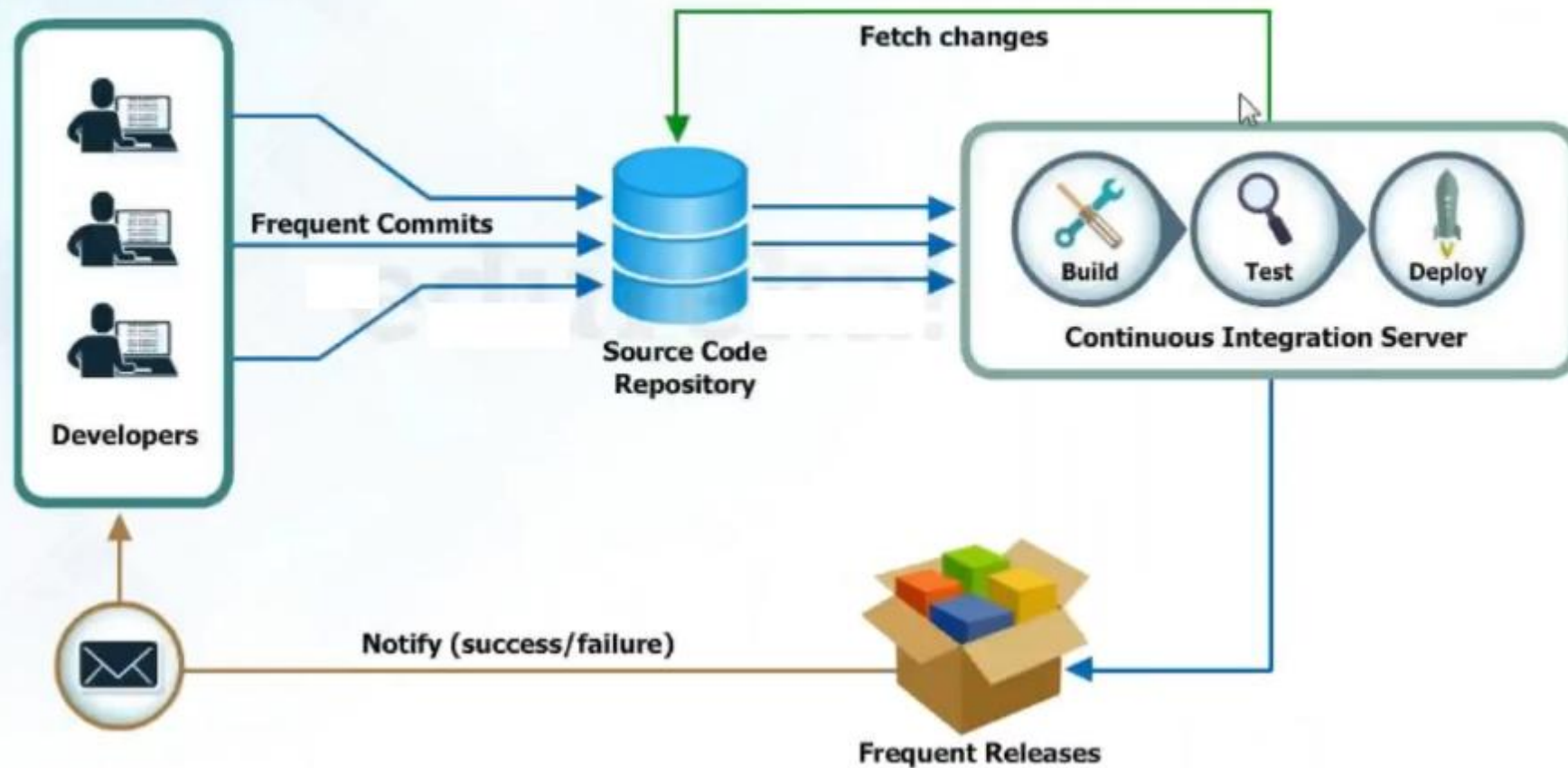


Continuous Integration (Contd.)



Continuous Integration (Contd.)

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to deploy code quickly to production.



Why do we need Continuous Integration?

- Detect problems or bugs, as early as possible, in the development life cycle.
- Since the entire code base is integrated, built and tested constantly , the potential bugs and errors are caught earlier in the life cycle which results in better quality software.

CI - Defined

.□ “Continuous Integration is a software development practice where members of a development team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including Code compilation, Integration test, Unit test etc.) to detect integration errors as quickly as possible” – Prakash Kumar

Jenkins

Using Jenkins as Continuous Integration Tool

Continuous Integration is the most important part of DevOps that is used to integrate various DevOps stages. Jenkins is the most famous Continuous Integration tool,

What is Jenkins?

- Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose.
- Jenkins is used to build and test your software projects 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 you to continuously deliver your software by integrating with a large number of testing and deployment technologies.
- With Jenkins, organizations can accelerate the software development process through automation. Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis and much more.
- Jenkins achieves Continuous Integration with the help of plugins. Plugins allows the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example: Git, Maven 2 project, Amazon EC2, HTML publisher etc.

The History of Jenkins

Hudson



Jenkins

- Hudson was started in 2004 at Sun by Kohsuke Kawaguchi as a hobby project.
- First release in 2005.
- Kohsuke worked on Hudson full time in early 2008.
- Became the leading Continuous Integration solution with a market share of over 70% in 2010.
- Renamed to Jenkins in 2011.

Jenkins

Java based

Open source
automation server

Software
development

Cross-platform
tool

Jenkins is written in Java. It was forked from Hudson when Oracle bought Sun Microsystems

It provides hundreds of plugins to support build creation, deployment, and automation of any software project

It helps automate non-human part of software development process with CI and continuous deployment (CD)

It is a cross-platform tool, and it offers configuration both through GUI interface and console commands

Jenkins

CI server

Distribution

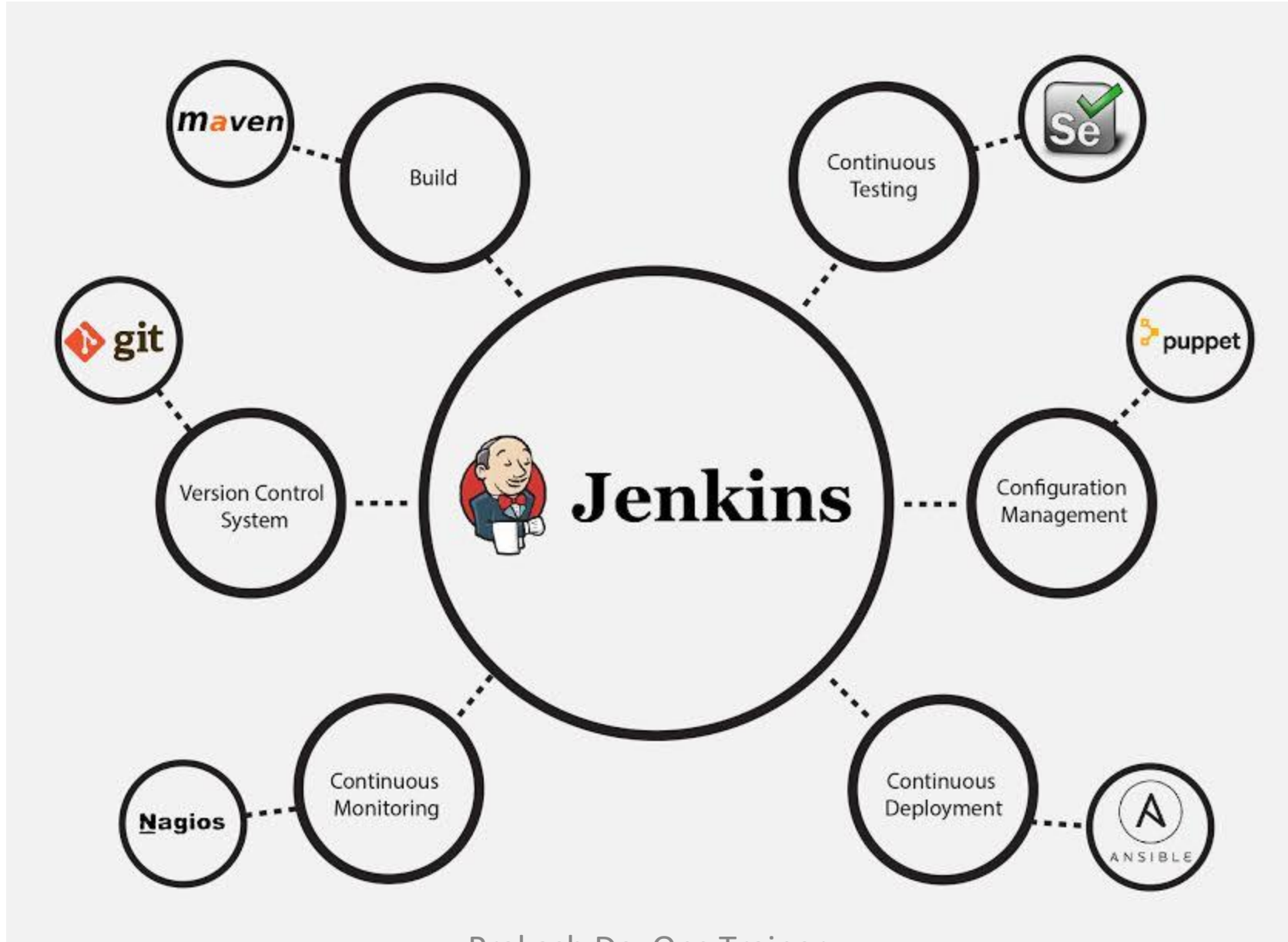
Cross-platform

It can be used as a CI server or as a continuous delivery hub for a project

It can easily distribute work across different machines and help trigger builds, tests, and deployments to multiple machines and platforms faster

It works on iOS, .Net, Android Development, Ruby, and Java

The image below depicts that Jenkins is integrating various DevOps stages:



Advantages of Jenkins include:

- It is an open source tool with great community support.
- It is easy to install.
- It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share with the community.
- It is free of cost.
- It is built with Java and hence, it is portable to all the major platforms.

Jenkins Key Metrics:

Following are some facts about Jenkins that makes it better than other Continuous Integration tools:

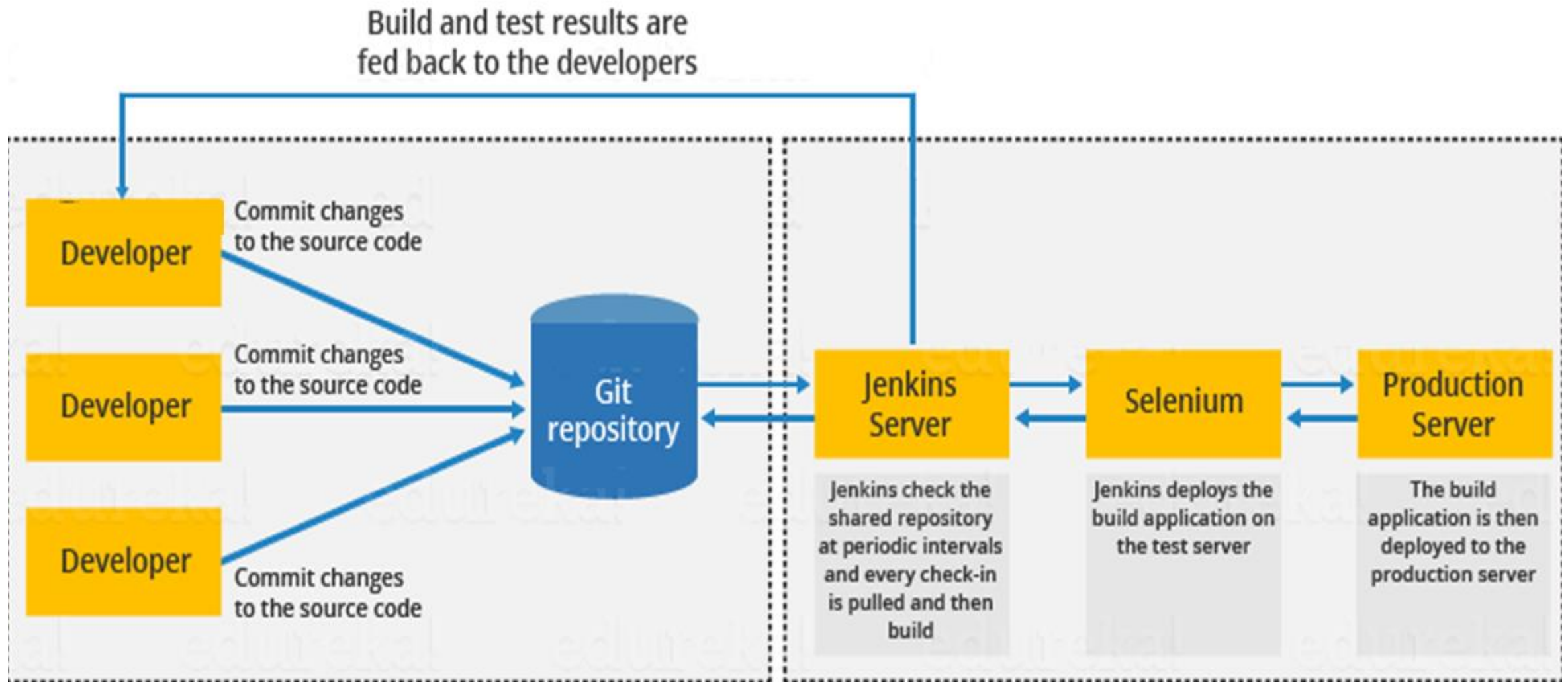
- **Adoption:** Jenkins is widespread, with more than 147,000 active installations and over 1 million users around the world.
- **Plugins:** Jenkins is interconnected with well over 1,000 plugins that allow it to integrate with most of the development, testing and deployment tools.

Without - Continuous Integration With Jenkins

Let us imagine a scenario where the complete source code of the application was built and then deployed on test server for testing. It sounds like a perfect way to develop a software, but, this process has many flaws. I will try to explain them one by one:

- Developers have to wait till the complete software is developed for the test results.
- There is a high possibility that the test results might show multiple bugs. It was tough for developers to locate those bugs because they have to check the entire source code of the application.
- It slows the software delivery process.
- Continuous feedback pertaining to things like coding or architectural issues, build failures, test status and file release uploads was missing due to which the quality of software can go down.
- The whole process was manual which increases the risk of frequent failure.

It is evident from the above stated problems that not only the software delivery process became slow but the quality of software also went down. This leads to customer dissatisfaction. So to overcome such a chaos there was a dire need for a system to exist where developers can continuously trigger a build and test for every change made in the source code. This is what CI is all about. Jenkins is the most mature CI tool available so let us see how Continuous Integration with Jenkins overcame the above shortcomings.



The above diagram is depicting the following functions:

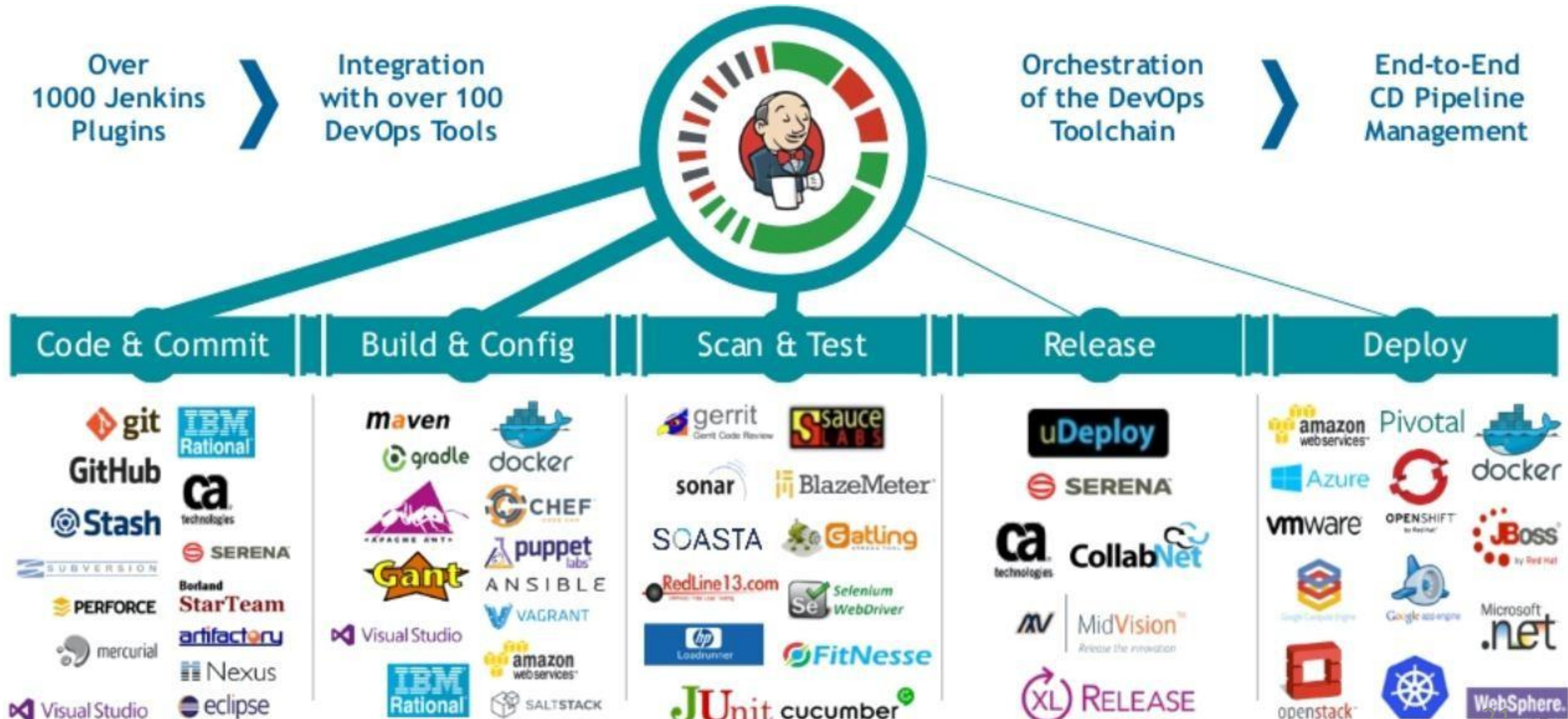
- First, a developer commits the code to the source code repository. Meanwhile, the Jenkins server checks the repository at regular intervals for changes.
- Soon after a commit occurs, the Jenkins server detects the changes that have occurred in the source code repository. Jenkins will pull those changes and will start preparing a new build.
- If the build fails, then the concerned team will be notified.
- If built is successful, then Jenkins deploys the built in the test server.
- After testing, Jenkins generates a feedback and then notifies the developers about the build and test results.
- It will continue to check the source code repository for changes made in the source code and the whole process keeps on repeating.

Before Jenkins	After Jenkins
The entire source code was built and then tested. Locating and fixing bugs in the event of build and test failure was difficult and time consuming, which in turn slows the software delivery process.	Every commit made in the source code is built and tested. So, instead of checking the entire source code developers only need to focus on a particular commit. This leads to frequent new software releases.
Developers have to wait for test results	Developers know the test result of every commit made in the source code on the run.
The whole process is manual	You only need to commit changes to the source code and Jenkins will automate the rest of the process for you.

Continuous Integration Using Jenkins

- Builds can be triggered from a commit in version control system or scheduling a cron job or by other builds in the queue
- Support version control systems like CVS, Subversion, Git, Perforce, Clearcase
- Can be integrated with bug tracking databases Jira, Bugzilla, Sonar Quality Gate
- Integrates with testing tools like Nunit, Junit, TestLink, Celenium Capability Axis, qTest, QMetry for Jira, Sonar
- Integrates with build tools like NAnt, EasyAnt, Ansible, Ant, Maven, Gradle, Visual Studio Code Metrics, SaltStack, Python, Ruby, Shell and Windows commands
- Integrates with config tools like Chef, Puppet, Ansible, Vagrant, IBM Rational, SaltStack
- Has plugins for Puppet Enterprise Pipeline, Ansible, OctopusDeploy, Docker Pipeline, Google Deployment Manager, Amazon Web Services, VMWare, Azure, Microsoft .Net, OpenStack

Continuous Integration Using Jenkins (Contd.)

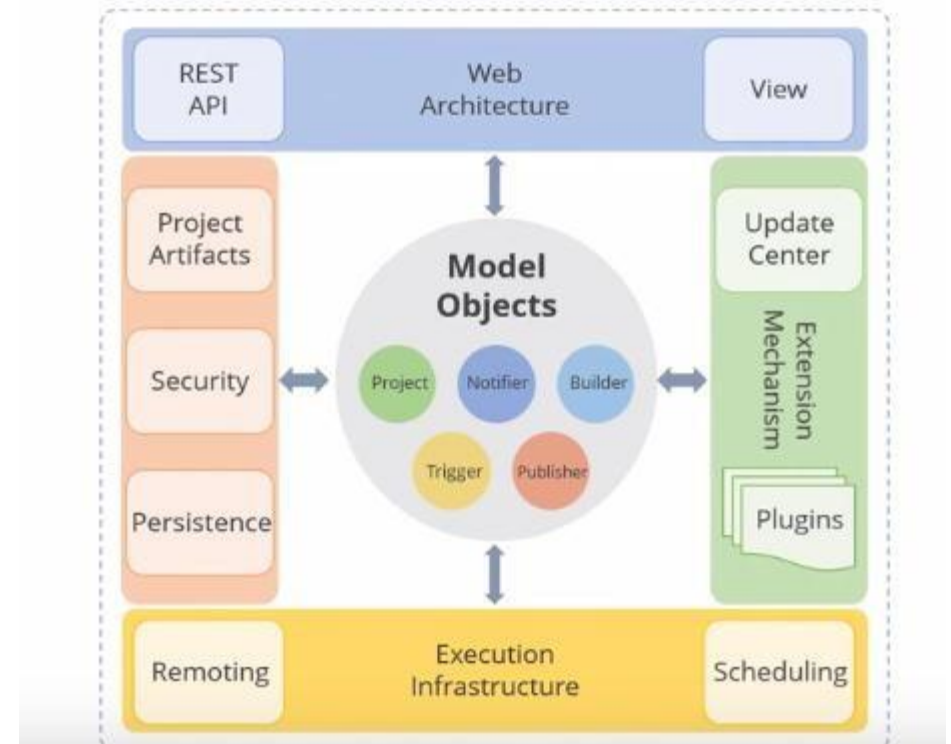


Jenkins Architecture

- Jenkins is a set of Java classes that model the concepts of a build system in a straight-forward fashion
- There are classes like Project, Build, that represents what the name says
- There are interfaces and classes that model code that performs a part of a build, such as SCM for accessing source code control system, Ant for performing an Ant- based build, Mailer for sending out e-mail notifications
- Jenkins classes are bound to URLs by using Stapler. Stapler is a HTTP request handling server.

Jenkins Architecture in Detail

- Views
 - Jenkins' model objects have multiple "views" that are used to render HTML pages about each object.
 - Jenkins uses Jelly as the view technology.
- Persistence
 - Jenkins uses the file system to store its data. Directories are created inside \$JENKINS_HOME in a way that models the object model structure.
- Plugins
 - Jenkins' object model is extensible and it supports the notion of "plugins," which can plug into those extensibility points and extend the capabilities of Jenkins.



Step 1. Launch an instance (**Amazon Linux 2 AMI**) , Access Instance terminal and install and setup java environment

```
sudo yum install -y git aws-cli
sudo yum install fontconfig java-17-openjdk -y
sudo alternatives --config java
```

(In this command point to the latest/or required java version by selection)

Step 2. Install Jenkins : Add the Jenkins repository to the yum repos and install Jenkins from there.

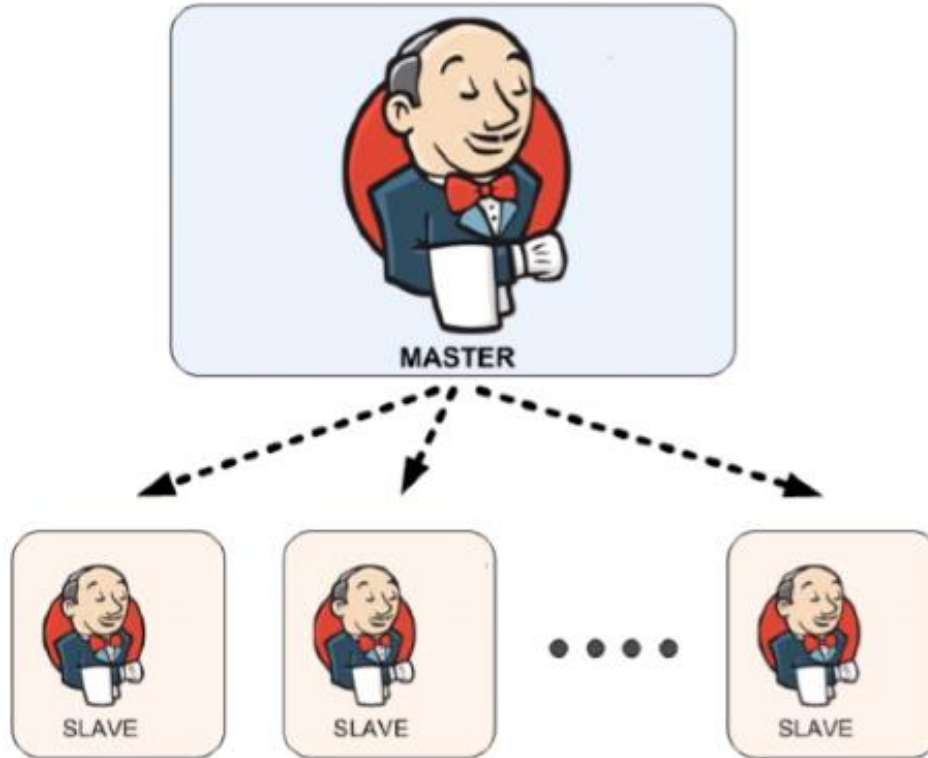
```
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
sudo amazon-linux-extras install epel -y
sudo yum install daemonize -y
sudo yum install -y jenkins
sudo service jenkins start
sudo systemctl enable jenkins
```

Step 3. That's it! Now you can go to browser and access Jenkins URL <http://<instance public ip>:8080>

Copy password sudo **cat /var/lib/jenkins/secrets/initialAdminPassword**

Install all the suggest plugin and done

Jenkins' Master and Slave Architecture



Master:

- Schedule build jobs.
- Dispatch builds to the slaves for the actual job execution.
- Monitor the slaves and record the build results.
- Can also execute build jobs directly.

Slave:







- Execute build jobs dispatched by the master.

Plugin

- A Plugin, like plugins on any other system, is a piece of software that extends the core functionality of the core Jenkins server.

Jenkins

[ENABLE AUTO REFRESH](#)

-  [New Item](#)
-  [People](#)
-  [Build History](#)
-  [Manage Jenkins](#)
-  [My Views](#)
-  [Credentials](#)

 [add description](#)

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

Welcome to Jenkins!

Please **create new jobs** to get started.

Job listing section

Key Takeaways

- Continuous Integration automates the build and testing of code for every commit to version control.
- Tools used for Continuous Integration are Jenkins, TeamCity, Travis CI, Bamboo, and CodeShip.
- Jenkins is a cross-platform tool and offers configuration both through GUI interface and console commands.
- TeamCity is a Java-based build management and continuous integration server.

Different Phases in Maven Build Lifecycle

validate: Validate the project is correct and all necessary information is available.

(what validate phase does: It checks if everything is in order, whether we have pom.xml in place, whether all the configuration is proper, whether we have all the code in a proper way.)

compile: Compile the source code of the project.

test: Test the compiled source code using a suitable unit testing framework.

package: Take the compiled code and package it in its distributable format.

(That we can define in Jenkins job or POM.xml file. the format could be jar/ear/war/zip/nuget)

verify: Run any checks on results of integration tests to ensure quality criteria are met.

install: Install the package into the local repository, to use it as a dependency in other projects locally.

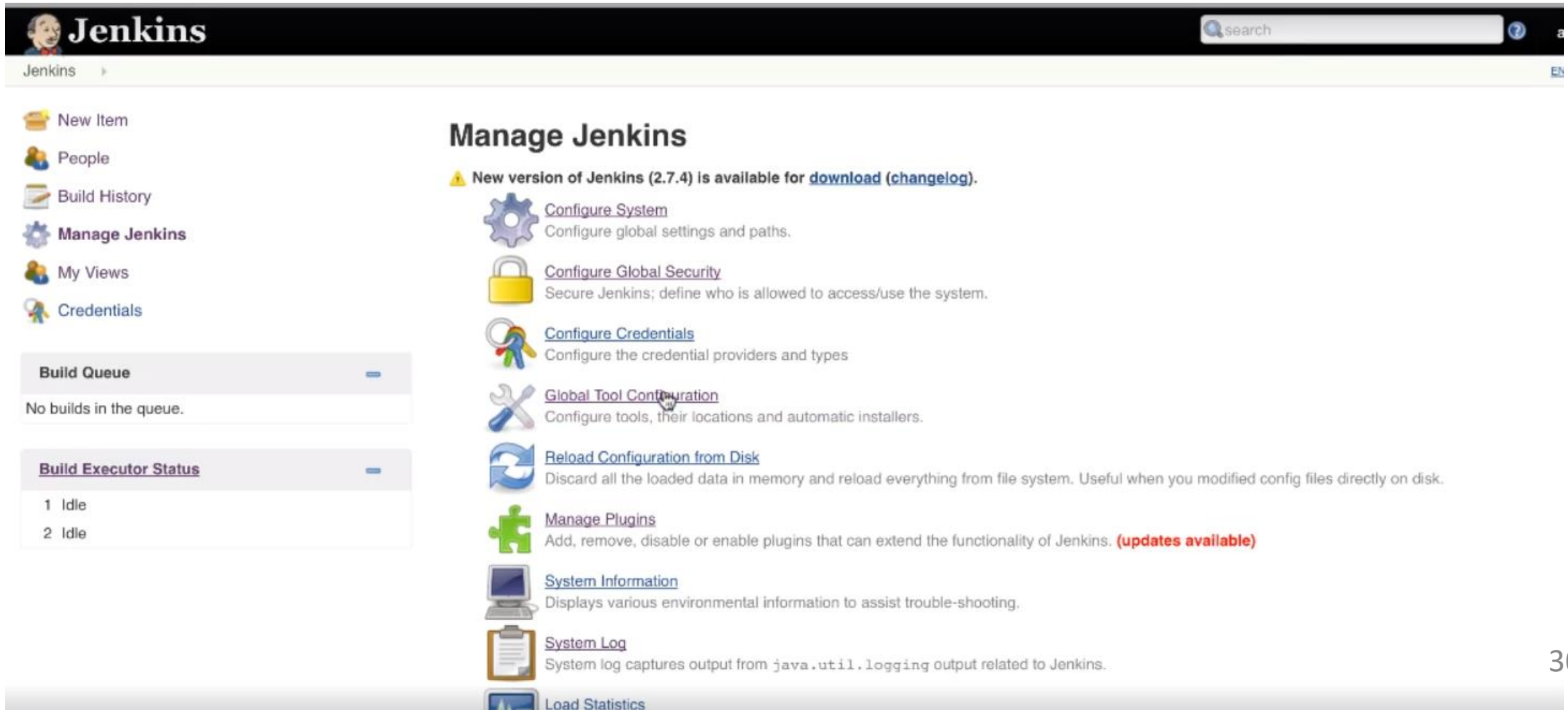
(Install the package in local Maven repository. It doesn't mean install package locally.)

deploy: Copy the final package to the remote repository for sharing with other developers and projects.

(Here remote repository could be Nexus repository or JFrog repository or any artifactory server repository or any drop location)

Configure Jenkins to work Maven, Java and GIT

1. Go to Manage Jenkins then select Global Tool Configuration
2. As you see we need to setup couple of things like JDK, GIT, Maven etc. So Jenkins to know where to locate those tools while build process.



The screenshot shows the Jenkins web interface. At the top is the Jenkins logo and a search bar. Below the logo is a navigation menu with links: New Item, People, Build History, Manage Jenkins (highlighted), My Views, and Credentials. On the left, there are two panels: 'Build Queue' showing 'No builds in the queue.' and 'Build Executor Status' showing two idle executors. The main content area is titled 'Manage Jenkins' and contains a list of configuration options, each with an icon and a description. A notification at the top of this section states: 'New version of Jenkins (2.7.4) is available for [download](#) ([changelog](#)).' The configuration options are: 'Configure System' (gear icon), 'Configure Global Security' (lock icon), 'Configure Credentials' (key icon), 'Global Tool Configuration' (wrench icon, highlighted with a mouse cursor), 'Reload Configuration from Disk' (refresh icon), 'Manage Plugins' (puzzle piece icon), 'System Information' (monitor icon), 'System Log' (clipboard icon), and 'Load Statistics' (bar chart icon).

Jenkins

Search

Jenkins

New Item

People

Build History

Manage Jenkins

My Views

Credentials

Build Queue

No builds in the queue.










Build Executor Status

1 Idle

2 Idle

Manage Jenkins

New version of Jenkins (2.7.4) is available for [download](#) ([changelog](#)).

-  [Configure System](#)
Configure global settings and paths.
-  [Configure Global Security](#)
Secure Jenkins; define who is allowed to access/use the system.
-  [Configure Credentials](#)
Configure the credential providers and types
-  [Global Tool Configuration](#)
Configure tools, their locations and automatic installers.
-  [Reload Configuration from Disk](#)
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.
-  [Manage Plugins](#)
Add, remove, disable or enable plugins that can extend the functionality of Jenkins. **(updates available)**
-  [System Information](#)
Displays various environmental information to assist trouble-shooting.
-  [System Log](#)
System log captures output from `java.util.logging` output related to Jenkins.
-  [Load Statistics](#)

JDK

JDK installations



JDK

Name

LocalJDK

JAVA_HOME

/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.161-0.b14.36.amzn1.x86_64



Install automatically



Delete JDK

Add JDK

List of JDK installations on this system

Git

Git installations



Git

Name

LocalGit

Path to Git executable

git



Maven

Maven installations



Maven

Name

LocalMaven

MAVEN_HOME

/usr/share/apache-maven



Install automatically

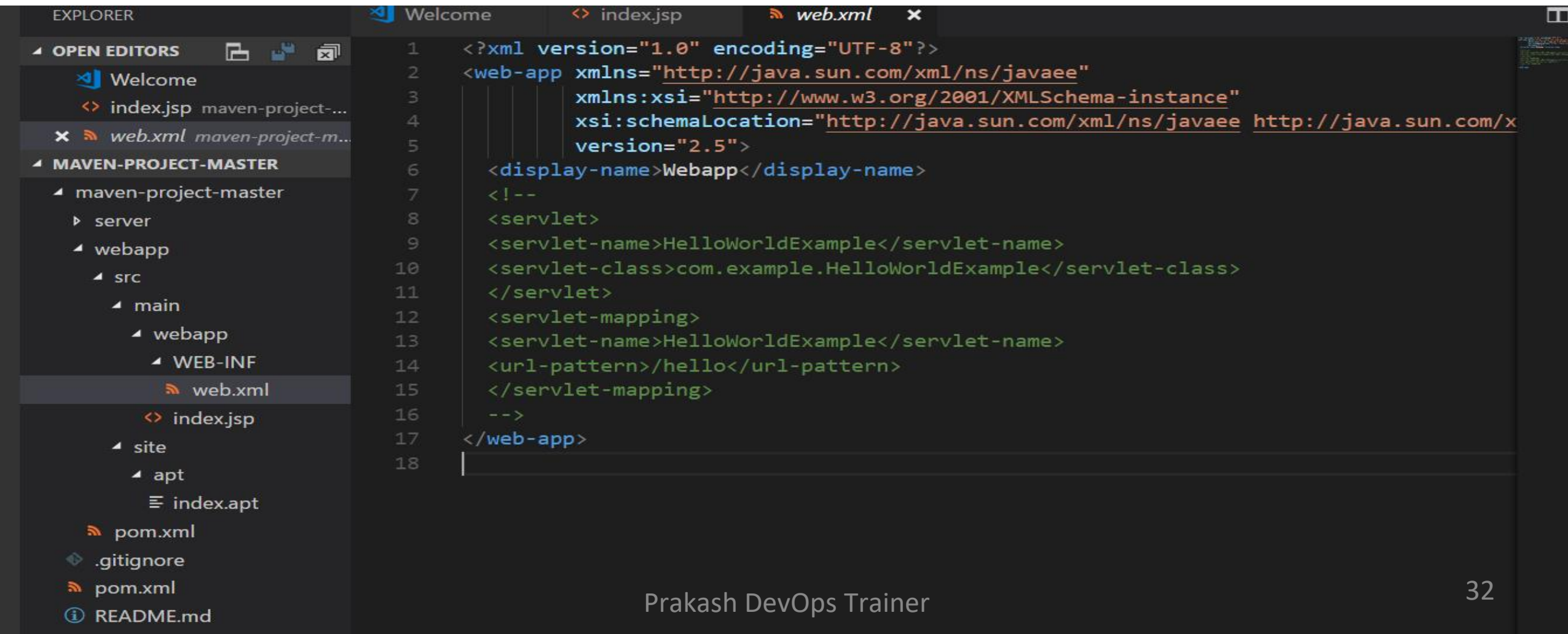
Delete Maven

First Maven Based Project:

Github repo UTR <https://github.com/prakashk0301/maven-project.git>

Web.xml: it tells you how to deploy your application

Index.jsp: Html page information ,which we can see after successful deployment.



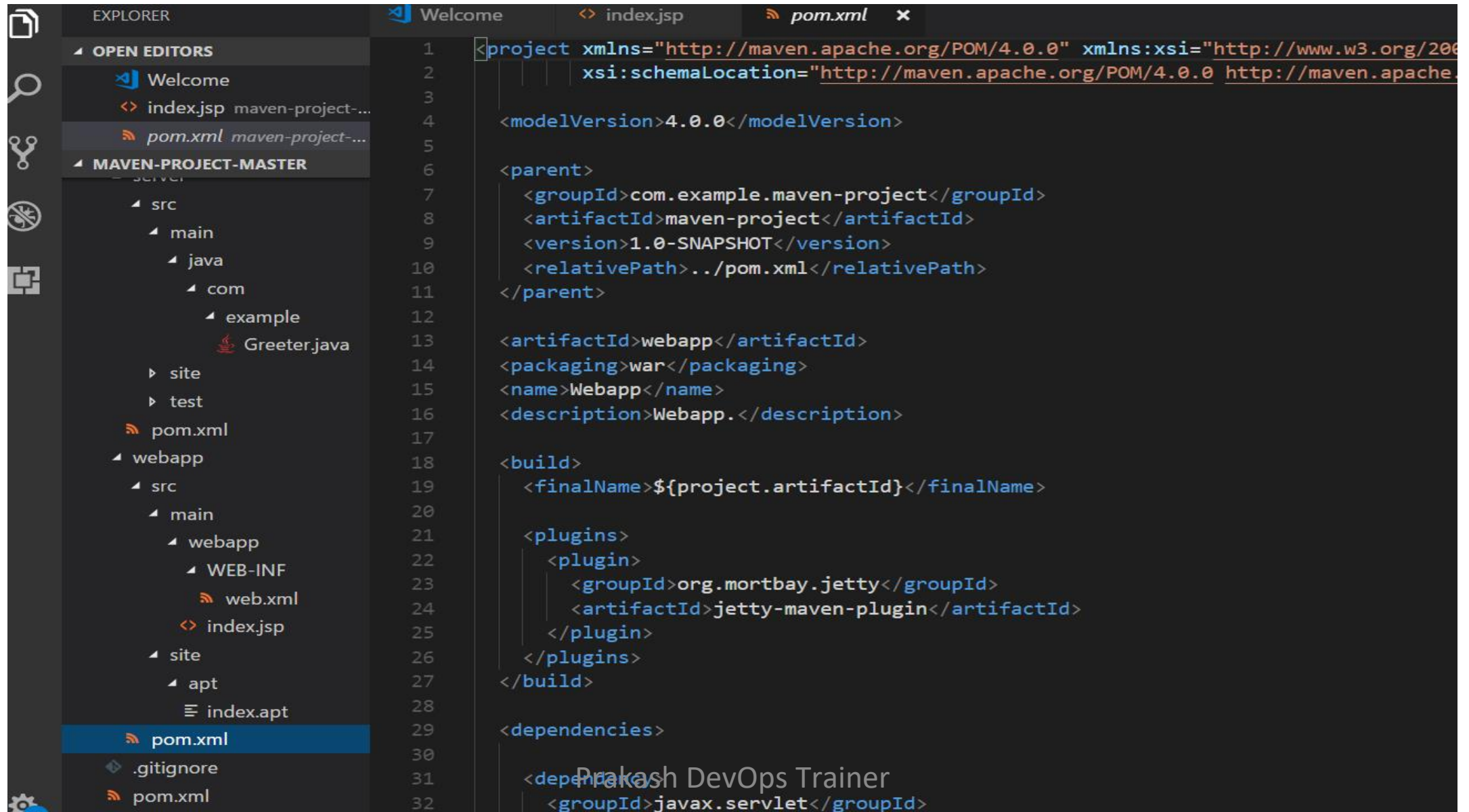
The screenshot shows an IDE with a dark theme. On the left is the Explorer panel showing the project structure of 'MAVEN-PROJECT-MASTER'. The 'webapp' directory is expanded, showing 'src' > 'main' > 'webapp' > 'WEB-INF' > 'web.xml' (selected). The main editor area shows the content of 'web.xml' with line numbers 1 through 18. The code defines a web application with a display name 'Webapp' and a single servlet named 'HelloWorldExample' that maps to the URL pattern '/hello'.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/x
5         version="2.5">
6     <display-name>Webapp</display-name>
7     <!--
8     <servlet>
9         <servlet-name>HelloWorldExample</servlet-name>
10        <servlet-class>com.example.HelloWorldExample</servlet-class>
11    </servlet>
12    <servlet-mapping>
13        <servlet-name>HelloWorldExample</servlet-name>
14        <url-pattern>/hello</url-pattern>
15    </servlet-mapping>
16    -->
17 </web-app>
18
```


Maven pom.xml file

- Describe the software project being built, including
 - The dependencies on other external modules.
 - The directory structures.
 - The required plugins.
 - The predefined targets for performing certain tasks such as compilation and packaging.

In our project we have total 3 pom.xml file



The screenshot displays an IDE interface with a project explorer on the left and a code editor on the right. The project explorer shows a Maven project structure with a 'pom.xml' file highlighted. The code editor shows the content of the 'pom.xml' file, which is a Maven POM (Project Object Model) file. The POM file defines the project's metadata, including its name, version, and dependencies. The POM file is structured as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.example.maven-project</groupId>
    <artifactId>maven-project</artifactId>
    <version>1.0-SNAPSHOT</version>
    <relativePath>../pom.xml</relativePath>
  </parent>
  <artifactId>webapp</artifactId>
  <packaging>war</packaging>
  <name>Webapp</name>
  <description>Webapp.</description>
  <build>
    <finalName>${project.artifactId}</finalName>
    <plugins>
      <plugin>
        <groupId>org.mortbay.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>javax.servlet</groupId>
```

Prakash DevOps Trainer

Build



Invoke top-level Maven targets



Maven Version

localMaven



Goals

clean package



Advanced...

Add build step ▾

Post-build Actions

Add post-build action ▾

Save

Apply

Jenkins has build our project in a workspace under a shared directory on the local box

Console Output

Started by user `admin`

Building in workspace `/Users/Shared/Jenkins/Home/workspace/maven-project`

```
> git rev-parse --is-inside-work-tree # timeout=10
```

Fetching changes from the remote Git repository

```
git rev-parse --is-inside-work-tree # timeout=10
```

Jenkins figured out the right order to build the job, First one is the root directory of our project, You can see job has been started by admin but in real time we need to trigger jobs using own user id such as Prakash Kumar has started the job.

[https://github.com/PrakashKumar](#)

```
[INFO] -----  
[INFO] Reactor Build Order:  
[INFO]  
[INFO] Maven Project  
[INFO] Server  
[INFO] Webapp  
[INFO]  
[INFO] -----  
[INFO] Building Maven Project 1.0-SNAPSHOT
```

Cleaning the environment before building the job, Basically it is removing any artifact before building

```
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.jar (25 kB at [INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ maven-project ---
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-api/2.0.6/maven-plugin-api-2.0.6.pom
Progress (1): 1.5 kB

Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-api/2.0.6/maven-plugin-api-2.0.6.pom (1.5 kB at 182 kB/s)
```

Server module: Jenkins is executing the sequence of build command such as compile, test for packaging

```
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ webapp ---
[INFO]
[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ webapp ---
[debug] execute contextualize
[INFO] Using 'utf-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/maven-project/webapp/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ webapp ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-resources-plugin:2.5:testResources (default-testResources) @ webapp ---
[debug] execute contextualize
[INFO] Using 'utf-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/maven-project/webapp/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ webapp ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.11:test (default-test) @ webapp ---
[INFO] No tests to run.
[INFO] Surefire report directory: /var/lib/jenkins/workspace/maven-project/webapp/target/surefire-reports
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit3/2.11/surefire-junit3-2.11.jar
Progress (1): 1.7 kB
```

After server module it prints out the Unit test report

```

T E S T S
-----

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ wehann ---

```

Lastly it package the Server module and producing the jar file under the server directory

[illegible]

Building the webapp module which is pretty similar to the server module

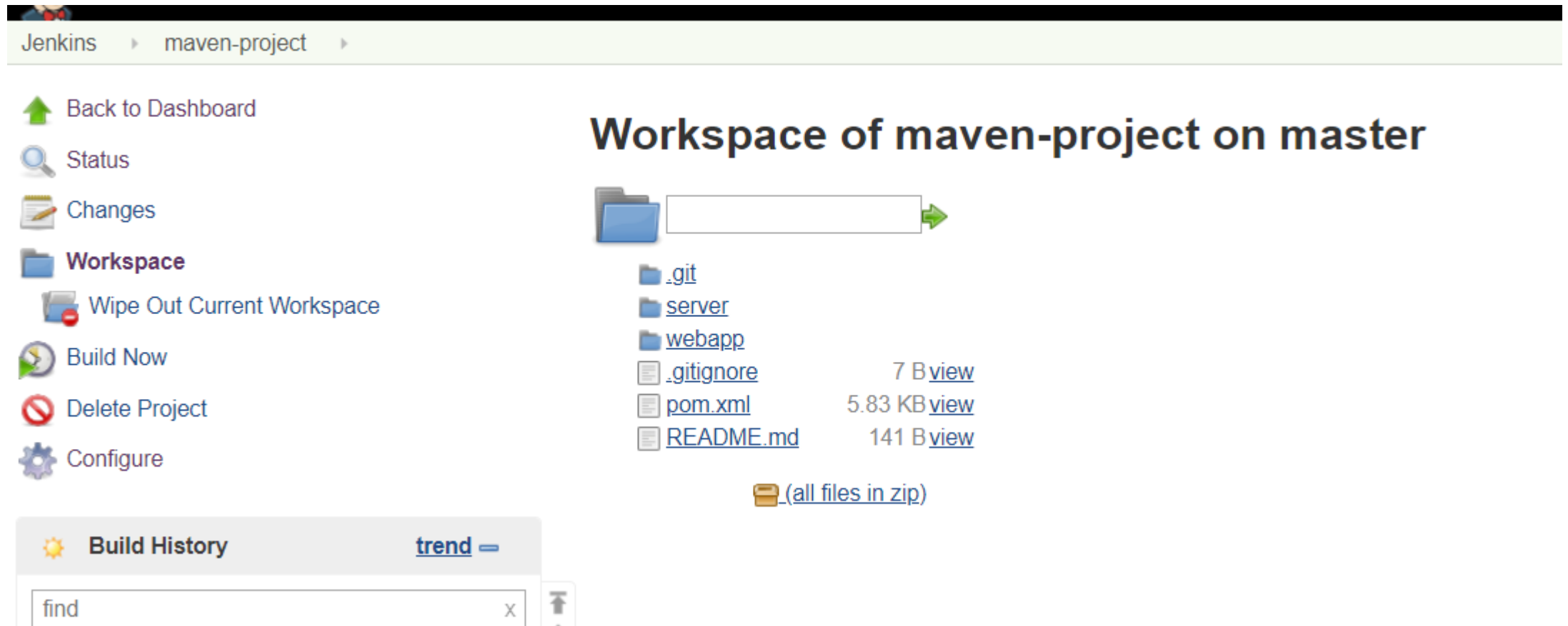
```
[INFO] -----
[INFO] Building Webapp 1.0-SNAPSHOT
[INFO] -----
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-war-plugin/2.2/maven-war-plugin-2.2.pom
Progress (1): 4.1/6.5 kB
Progress (1): 6.5 kB
```

```
^      ^
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ webapp ---
[INFO]
[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ webapp ---
[debug] execute contextualize
[INFO] Using 'utf-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/maven-project/webapp/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ webapp ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-resources-plugin:2.5:testResources (default-testResources) @ webapp ---
[debug] execute contextualize
[INFO] Using 'utf-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/maven-project/webapp/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ webapp ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.11:test (default-test) @ webapp ---
[INFO] No tests to run.
[INFO] Surefire report directory: /var/lib/jenkins/workspace/maven-project/webapp/target/surefire-reports
```


In the end it packages the webapp module and produces the webapp war file which can be further deploy to servlet container like tomcat, apache

```
Downloaded from central: https://repo.maven.apache.org/maven2/com/thoughtworks/xstream/xstream/1.3.1/xstream-1.3.1.jar (431 kB at 7.3 MB/s)
[INFO] Packaging webapp
[INFO] Assembling webapp [webapp] in [/var/lib/jenkins/workspace/maven-project/webapp/target/webapp]
[INFO] Processing war project
[INFO] Copying webapp resources [/var/lib/jenkins/workspace/maven-project/webapp/src/main/webapp]
[INFO] Webapp assembled in [31 msecs]
[INFO] Building war: /var/lib/jenkins/workspace/maven-project/webapp/target/webapp.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] Maven Project ..... SUCCESS [ 1.775 s]
[INFO] Server ..... SUCCESS [ 5.571 s]
[INFO] Webapp ..... SUCCESS [ 0.941 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 8.578 s
[INFO] Finished at: 2018-02-16T23:46:18Z
[INFO] Final Memory: 21M/51M
[INFO] -----
Finished: SUCCESS
```


In workspace: It displays the directory structure of this workspace which Jenkins uses to built the project



The screenshot shows the Jenkins web interface for a project named 'maven-project'. The breadcrumb navigation at the top reads 'Jenkins > maven-project >'. On the left sidebar, there are several action links: 'Back to Dashboard' (with a green arrow icon), 'Status' (with a magnifying glass icon), 'Changes' (with a notepad icon), 'Workspace' (with a folder icon and bold text), 'Wipe Out Current Workspace' (with a trash can icon), 'Build Now' (with a play button icon), 'Delete Project' (with a red prohibition icon), and 'Configure' (with a gear icon). Below these links is a 'Build History' section with a search bar containing the text 'find' and a 'trend' link. The main content area is titled 'Workspace of maven-project on master'. It displays a directory tree with folders '.git', 'server', and 'webapp'. Below the folders are files: '.gitignore' (7 B, view link), 'pom.xml' (5.83 KB, view link), and 'README.md' (141 B, view link). At the bottom of the file list is a download icon and a link '(all files in zip)'.

Webhook Trigger : Set for Jenkins

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. [Webhooks Guide](#).

Which events would you like to trigger this webhook?

Just the push event: Only send data when someone pushed into my repository.

Send me everything: If there is any pull or push event in our repository we will get notified.

Let me select individual events: We can configure for what events we want our data.

- 1.Go to your project repository.
- 2.Go to "settings" in the right corner.
- 3.Click on "webhooks."
- 4.Click "Add webhooks."
- 5.Write the Payload URL as

<http://<Jenkins Instance IP>:8080/github-webhook/>

Prakash DevOps Trainer

The screenshot displays the GitHub repository settings interface. At the top, a navigation bar includes links for Code, Issues (0), Pull requests (4), Actions, Projects (0), Wiki, Security, Insights, and Settings. On the left sidebar, a menu lists Options, Collaborators, Branches, and Webhooks (which is highlighted). The main content area is titled 'Webhooks' and includes an 'Add webhook' button. Below this, a descriptive text explains that webhooks send POST requests to specified URLs. A single webhook is listed with a green checkmark, the URL 'http://35.158.251.73:8080/github-webhook/' followed by '(push)', and 'Edit' and 'Delete' buttons. Below the webhooks section, there is a 'Build Triggers' section with a list of checkboxes: 'Trigger builds remotely (e.g., from scripts)', 'Build after other projects are built', 'Build periodically', 'GitHub hook trigger for GITScm polling' (which is checked), and 'Poll SCM'.