3. Understanding the common problem without Maven

There are many problems that we face during the project development. They are discussed below:

- **Adding set of Jars and dependencies in each project:** In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.
- **Creating and maintaining the right project structure:** We must create the right project structure in servlet, struts etc, otherwise it will not be executed.
- **Building and Deploying the project:** We must have to build and deploy the project so that it may work.

4. What it does?

Maven simplifies and give solution to the above-mentioned problems. It performs mainly following tasks.

- It makes a project easy to build
- It provides uniform build process (maven project can be shared by all the maven projects)
- It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
- It is easy to migrate for new features of Maven

5. What is Build Tool

A build tool takes care of everything for building a process. It does following:

- Generates source code (if auto-generated code is used)
- Generates documentation from source code
- Compiles source code
- Packages compiled code into JAR of ZIP file
- Installs the packaged code in local repository, server repository, or central repository

6. Uses of Apache Maven

- Use as Build Tool
- Use as to manage Project structure
- Building, publishing and deploying
- Documentation
- Reporting
- Releases
- Distribution

7. Setup and Installation for Maven

You can download and install maven on windows, Linux and MAC OS platforms.

Note: Maven is Java based tool, so the very first requirement is to have JDK installed on your machine.

Download Maven archive

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

OS	Archive name
Windows	apache-maven-3.5.0-bin.zip
Linux	apache-maven-3.5.0-bin.tar.gz **or sudo apt-get install maven**
Mac	apache-maven-3.5.0-bin.tar.gz

Extract the Maven archive

Extract the archive, to the directory you wish to install Maven 3.5.0. The subdirectory apache-maven-3.5.0 will be created from the archive.

OS	Location (can be different based on your installation)
Windows	C:\Program Files\Apache Software Foundation\apache-maven-3.5.0
Linux	/usr/local/apache-maven
Mac	/usr/local/apache-maven

Set Maven environment variables

Add M2_HOME, M2, MAVEN_OPTS to environment variables.

OS	Output

Windows	Set the environment variables using system properties. M2_HOME=C:\Program Files\Apache Software Foundation\apache-maven-3.5.0 M2=%M2_HOME%\bin MAVEN_OPTS=-Xms256m -Xmx512m
Linux	Open command terminal and set environment variables. export M2_HOME=/usr/local/apache-maven/apache-maven-3.5.0 export M2=$M2_HOME/bin export MAVEN_OPTS='-Xms256m -Xmx512m'
Mac	Open command terminal and set environment variables. export M2_HOME=/usr/local/apache-maven/apache-maven-3.5.0 export M2=$M2_HOME/bin export MAVEN_OPTS=-Xms256m -Xmx512m

Add Maven bin directory location to system path

Now append M2 variable to System Path

OS	Output
Windows	Append the string ;%M2% to the end of the system variable, Path.
Linux	export PATH=$M2:$PATH
Mac	export PATH=$M2:$PATH

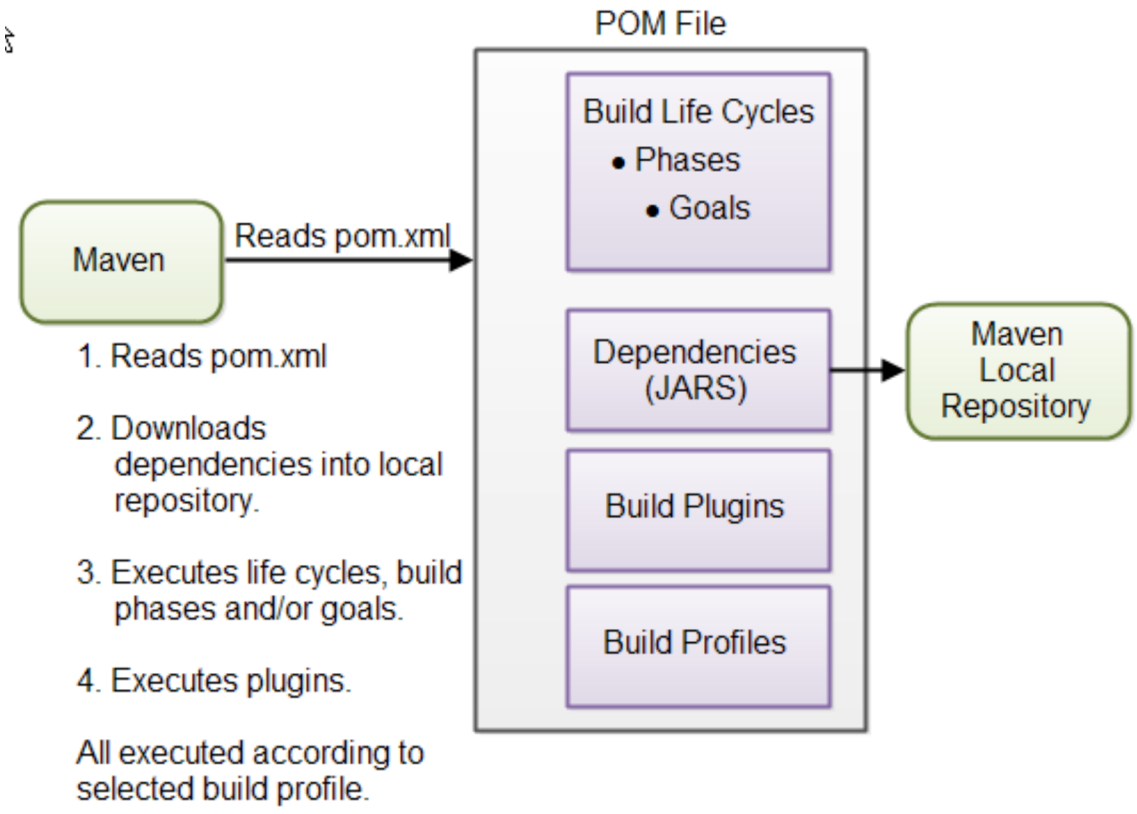
Verify Maven installation

Now open console, execute the following mvn command.

OS	Task	Command
Windows	Open Command Console	c:\> mvn --version
Linux	Open Command Terminal	$ mvn --version
Mac	Open Terminal	machine:~ joseph$ mvn --version

Maven help in project structure ,dependencies management,the reason behind is that maven repositories,maven talk to repositories and form the project structure.

Note: Maven get all its knowledge from maven repositories,how will be the project structure,what is the project type etc.



Maven uses the concept of the repository to hold the jar files. There are two types of repository namely local and remote repository.

Every maven project has a pom.xml file. If u run Your application with maven then, Now it checks your local repository for the jar files. If the jar files are not found it goes to the remote repository and download the jar. After downloading jars it will Execute life cycle, build phases and/or goals.

8. First Sample Application:

(Here ,I use Linux System)

1.Open Command Prompt ctrl+Alt+T

Command : $mvn archetype:generate

(when you run this for 1st time it download lots of maven plugins)

2,choose a number (What selection number is ?)

' There are number of archetype, the number is the specific number of a archetype that is available to use.Then the project structure will form accordingly.

For example:

1718: remote -> org.springframework.boot:spring-boot-sample-jetty-archetype (Spring Boot Jetty Sample)
1719: remote -> org.springframework.boot:spring-boot-sample-profile-archetype (Spring Boot Profile Sample)

But here we will go with default which is 981:
(Choose org.apache.maven.archetypes:maven-archetype-quickstart version:)

9. Maven pom.xml file

POM is an acronym for **Project Object Model.** The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

Maven reads the pom.xml file, then executes the goal.

→ In Our case the generated project has this pom.xml file.

```
<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>java.org.wahid</groupId>
  <artifactId>MavenTestApp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>MavenTestApp</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

The simple pom.xml file, contains following elements:

Element	Description
project	It is the root element of pom.xml file.
modelVersion	It is the sub element of project. It specifies the modelVersion. Which is set to 1.0.0.
groupId	It is the sub element of project. It specifies the id for the project group.
artifactId	It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs.
version	It is the sub element of project. It specifies the version of the artifact under given group.

packaging	defines packaging type such as jar, war etc.

name	defines name of the maven project.
url	defines url of the project.
dependencies	defines dependencies for this project.
dependency	defines a dependency. It is used inside dependencies.
scope	defines scope for this maven project. It can be compile, provided, runtime, test and system.

3. Compile Maven Project:

To compile maven project go to the directory where pom.xml file is there,then execute the command

$ mvn compile

Then it will download all the dependencies and compile the project.after compile project you will a target directory is created which having all the compile classes

4. Package Maven Project:

For Packaging the project just run following command

$ mvn package

So that it will package the project into mentioned format jar, war etc.

Note: When we package the project, it will automatically run test Junit Cases and give the Build result.

After package you see target folder contains: classes, maven-archiver , surefire , surefire-reports test-classes, TestMavenApp-1.0-SNAPSHOT.jar

```
Building jar: /home/vave/workspace/Maven-Project/TestAppMaven/target/TestAppMaven-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

5.To run The Project :

$ java -cp target/TestMavenApp-1.0-SNAPSHOT.jar com.eqv.wahid.App

```
Hello World!
```

6.To run Test Cases :

$ mvn test

After the executing this command it will find the test cases and execute it and give the result

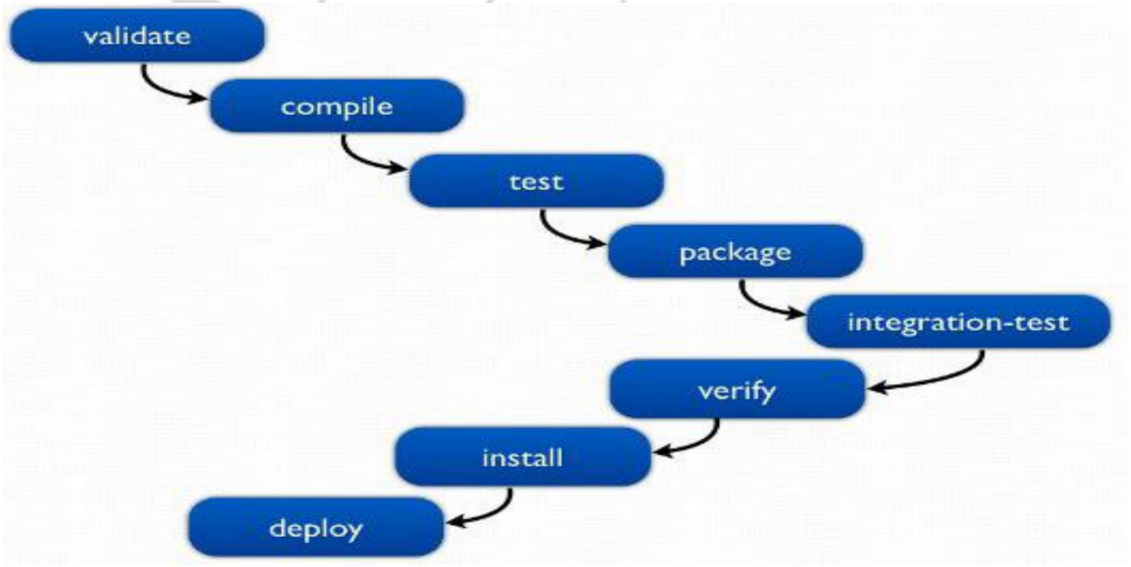
```
--- maven-surefire-plugin:2.10:test (default-test) @ TestMavenApp ---
[INFO] Surefire report directory: /home/vave/workspace/Maven-
Project/TestMavenApp/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.eqv.wahid.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.059 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

10. Maven - Build LifeCycle

A Build Lifecycle is a well-defined sequence of phases which define the order in which the goals are to be executed. Here phase represents a stage in life cycle.
As an example, a typical Maven Build Lifecycle consists of following sequence of phases

A Build Lifecycle is Made Up of Phases:

Each of these build lifecycles is defined by a different list of build phases, where in a build phase represents a stage in the lifecycle.



For example, the default lifecycle comprises of the following phases :

- **Validate** - validate the project is correct and all necessary information is available
- **Compile** - compile the source code of the project
- **test** - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **Package** - take the compiled code and package it in its distributable format, such as a JAR.
- **Verify** - run any checks on results of integration tests to ensure quality criteria are met
- **Install** - install the package into the local repository, for use as a dependency in other projects locally
- **Deploy** - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar,wa), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.

Summary:

- ✓ Maven is a software build management and comprehension tool.
- ✓ Maven is based on the concept of a project object model, Maven helps in code build, dependency management, documentation creation, site publication, and distribution publication are all controlled from the declarative file.
- ✓ Maven can be extended by plugins to utilise a number of other development tools for reporting or the build process.
- ✓ Maven Archetype is a set of tools to deal with archetypes, i.e. an abstract representation of a kind of project that can be instantiated into a concrete customized Maven project. An archetype knows which files will be part of the instantiated project and which properties to fill to properly customize the project.

XI. Continuous Integration with Jenkins

In the previous chapter we have seen how development process is followed by developers for creating softwares and even maintain that software. So as the developers write code to create softwares, they also merge all that code into a centralised repository or Version Control System like Github.

This code is pushed into repositories several times a day and over the period of time all of the code gets merged. Traditional software development methods don't dictate how frequently or regularly you integrate all of the source on a project. Programmers can work separately for hours, days, or even weeks on the same source without realizing how many conflicts (and perhaps bugs) they are generating.

1. Integration is painful

Agile teams produce workable and robust code in each iteration. All that code if built and evaluated returns lot of conflicts, bugs and errors. Developers needs to solve those conflicts and issues before moving to next iteration. The more programmers are sharing the code, the more problematic this is.

For these reasons, agile teams often therefore choose to use Continuous Integration.

2. Some Terminologies before we begin.

Source Code

All the code that developers writes to create the software is called as Source Code.

The build process

It is process by which source code is converted into a stand-alone form that can be run on a computer. For example, a source code written to develop a windows software when built will create a .exe or .msi file. Another example if a Java Source code is built it may create a .jar, .war or .ear file. This deployable piece of software is called as Artefact.

The source code can be built, packaged and deployed manually. But there are some build tools that make developers life easy when it comes to building artefact or even deploying it. These are called as build automation tools.

Some build tools

- Ant
- Maven
- Gradle
- Msbuild

- Nant

Unit Testing

Unit testing simply verifies the individual unit of code(mostly functions) works as expected. Developer along with writing the code will write the test cases that can be executed at the build time. Some test cases can be automatically generated.

The objective of unit testing is to isolate a section of code(unit) and verify its correctness.

3. What is Continuous Integration

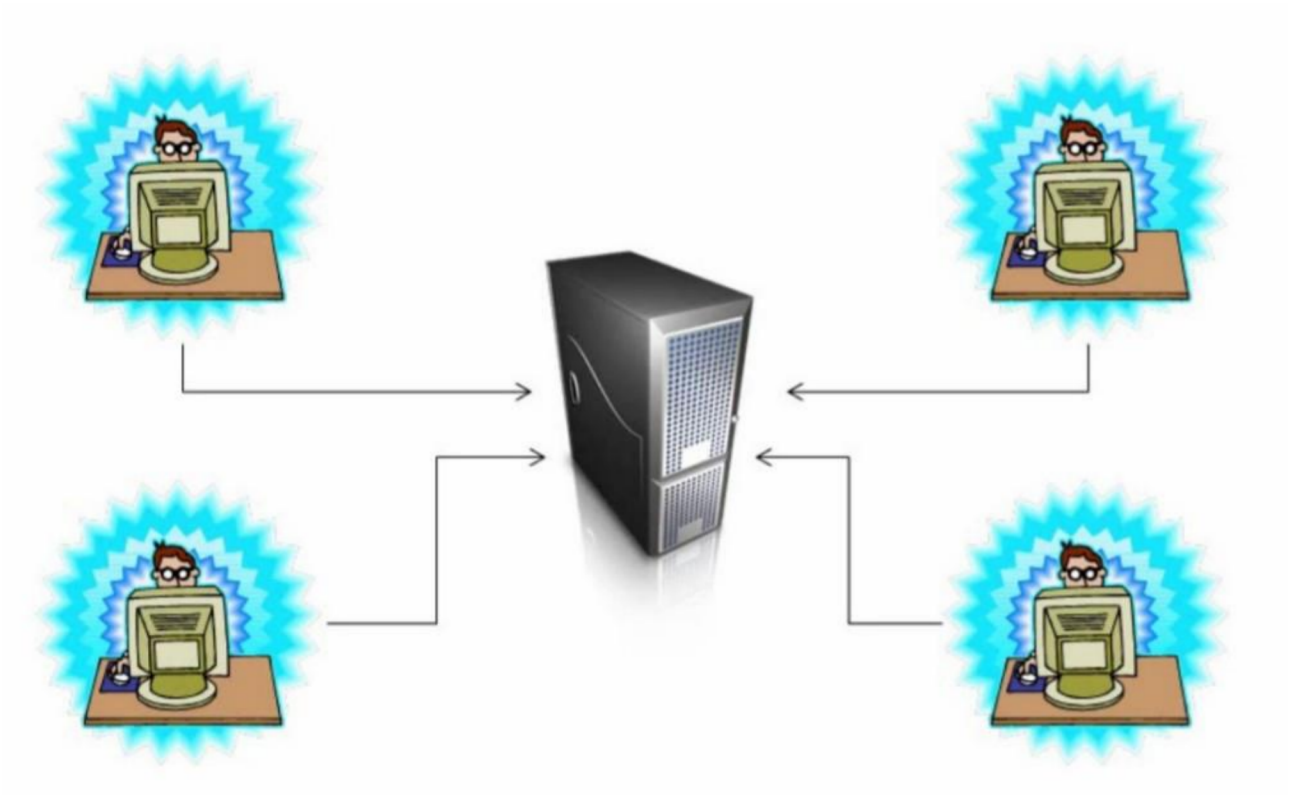
Continuous Integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control. CI encourages developers to share their code and unit tests by merging their changes into a shared version control repository after every small task completion. Committing code triggers an automated build system to grab the latest code from the shared repository and to build, test, and validate the full master branch (also known as the trunk or main).

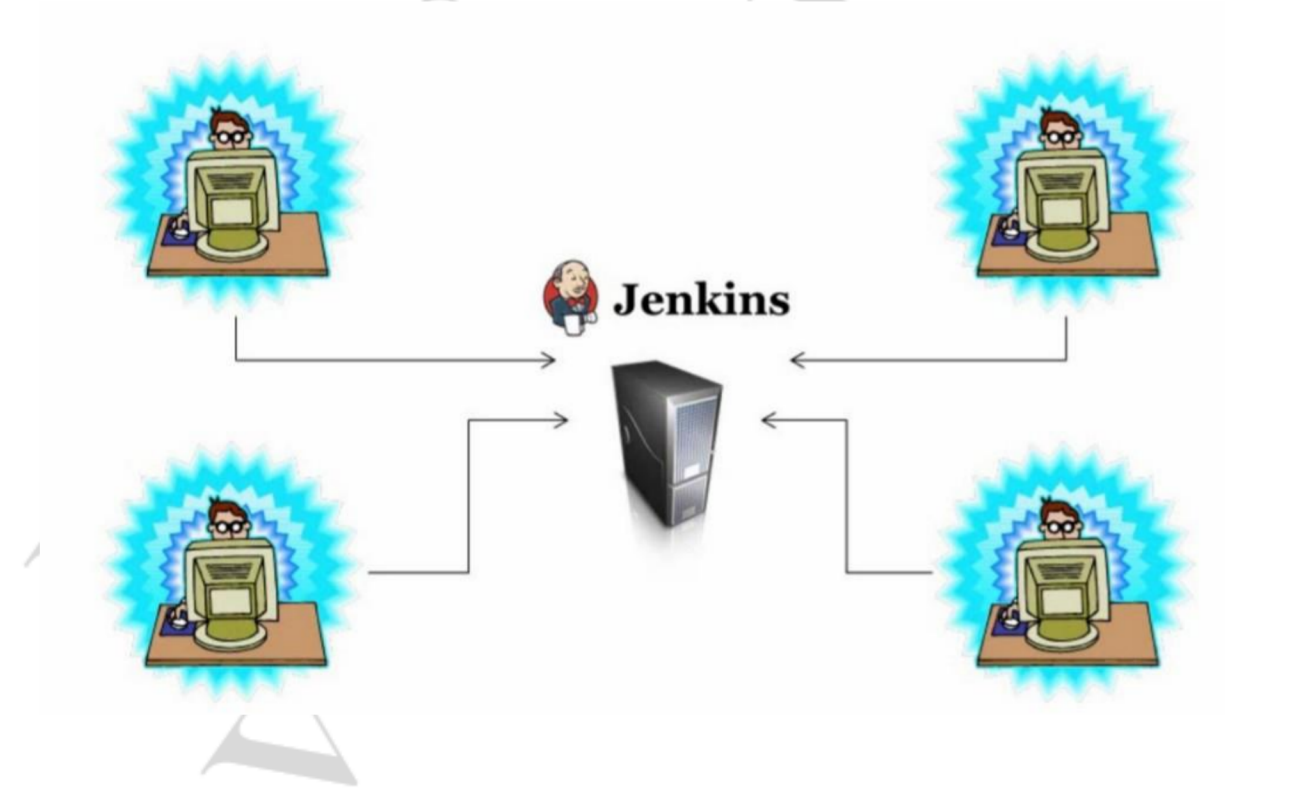
The Problem

Developers will write the code and BUILD it in their local system. Once developers test the code and verify locally they push it to the centralised repository like github. Similarly, all the developers would be pushing their code to VCS several times a day. Developers would be working in their own silos or caves and keep writing the code until they finish a particular task or the project. Now all the code which developers have pushed into the VCS, if built and tested will return lots & lots of conflicts, error due to which build will fail.

The Solution

To get around this very problem whenever the developer push the code to the VCS it should be fetched, built & tested by a build server at the same time.




Jenkins

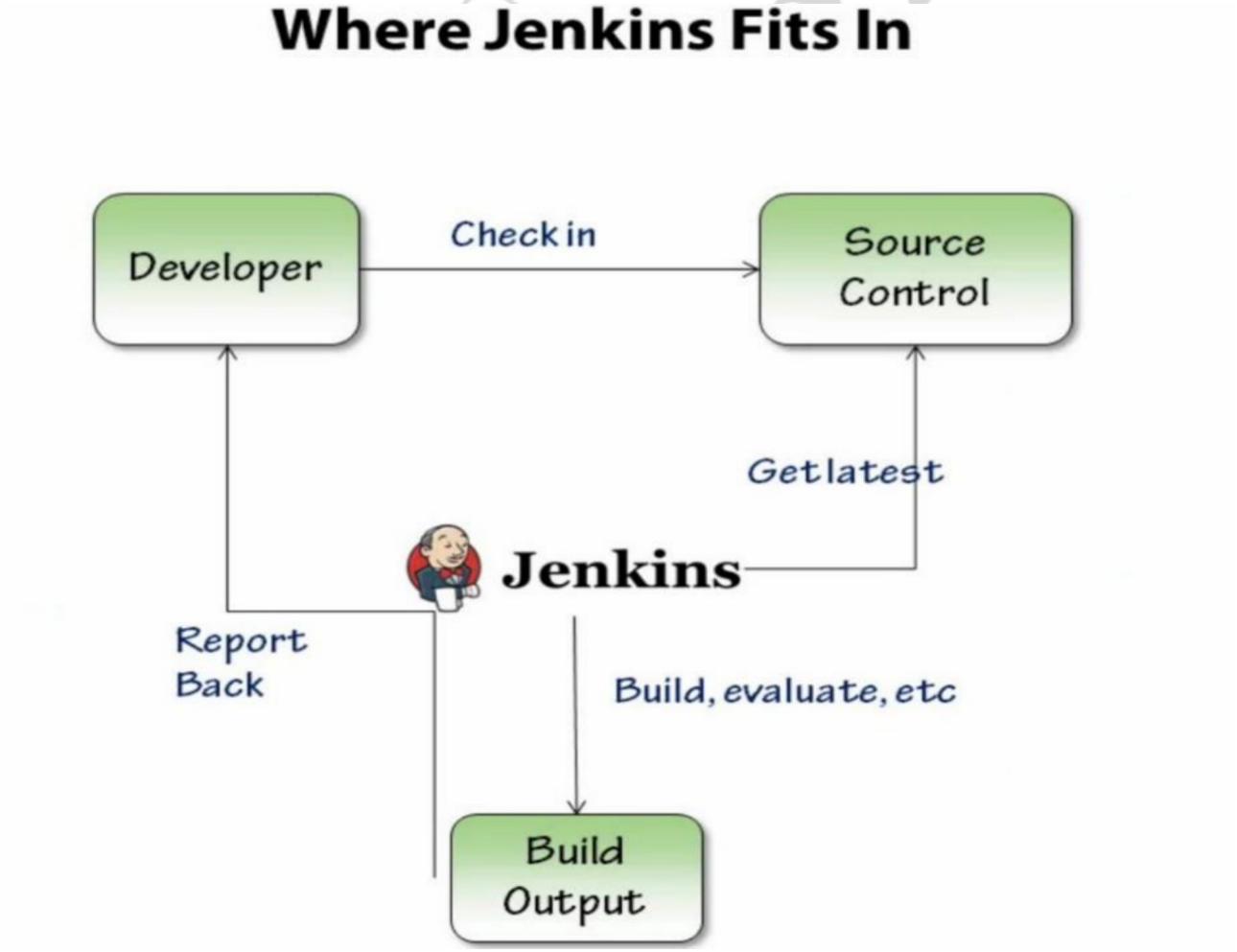
This process repeated several times in a week or day or daily once is called as continuous integration. Developers code is continuously getting integrated so any point in time we have a workable software, if their is any issue in the build process the developers will get notified through email and they will fix the problem.

4. What is Jenkins

Jenkins is a continuous integration server which can fetch the latest code from VCS, build it, test it and notify it to the developers. Jenkins can do many more things apart from just being a CI server. It was originally known a Hudson, Oracle inc owns Hudson now. Jenkins is an open source project written by Kohsuke Kawaguchi.

Jenkins is a java based web application server. As a prerequisite, we need to setup first Java on the machine to run Jenkins server.

Where Jenkins fits in



5. Features of Jenkins

OpenSorce

As jenkins is opensource there is lot contribution all around the world to the jenkins software. It has all the latest and greatest feature that developers integrating into it regularly.

Extensible

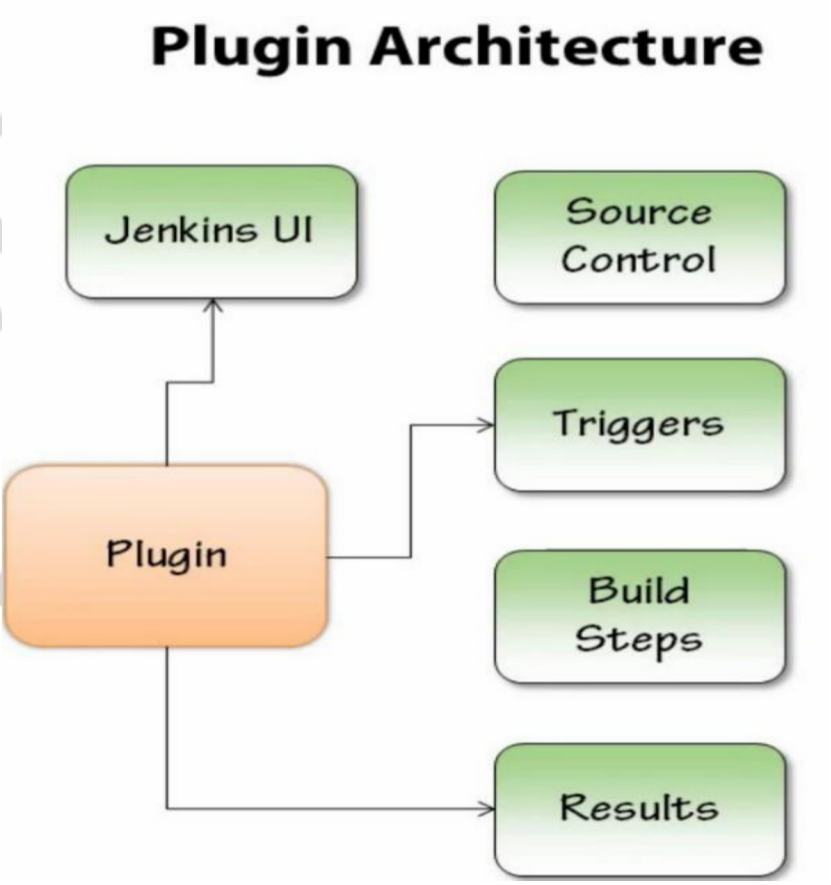
Jenkins comes with lot of goodies but its just not limited by that, Jenkins main power is its extensibility that can be achieved by installing plugins into it.

Jenkins opensource community has written tons of plugins, these plugins can do variety of tasks, like integration with external tools or servers.

- VCS plugins – git, svn, subversion etc
- Build plugins – Maven, ANT, Msbuild etc
- Notification plugins – Email, chat, sms etc
- Cloud plugins – Create cloud instances, deploy code to cloud services etc
- Testing plugins – Code analysis, Unit test case, Static code analysis etc

The list of plugins is very long, whenever we want Jenkins to do some tasks just search for that plugin and most of the time you will find something.

For example, if you want jenkins to deploy java artefact to tomcat server, search for the plugin named “deploy to container”.



6. Jenkins Setup

Jenkins can be installed on windows, Linux or Mac OS. Jenkins just needs java software to run.

In this tutorial, we will install jenkins on a ubuntu server. You can setup a vm or a cloud instance.

Prereqs

Java runtime environment/ JRE can be installed on the system but we will install JDK as we will setup maven moving along and build some java code. To Build the java code we will need JDK.

```
sudo add-apt-repository ppa:openjdk-r/ppa

sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

Installing Jenkins

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt-get update
sudo apt-get install jenkins
```

Install git client and maven in jenkins server.

We will integrate Jenkins with github to download the source code.

We are testing the java source code which will be built by Maven, so we also need to install Maven on Jenkins server. This is not a mandatory requirment to run Jenkins if you are not using git and maven.

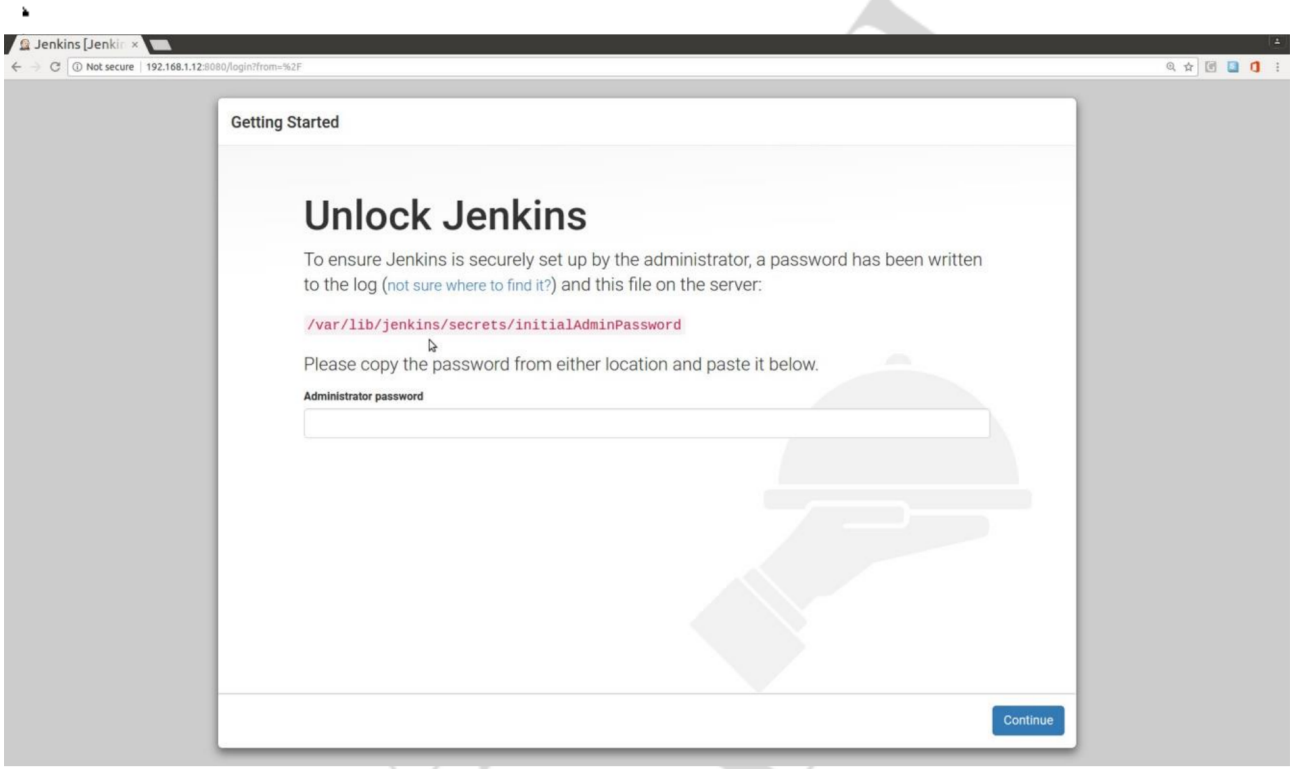
```
sudo apt-get install git
sudo apt-get install maven
```

Accessing Jenkins

Jenkins runs on port 8080 by default

Open up a browser and use below url.

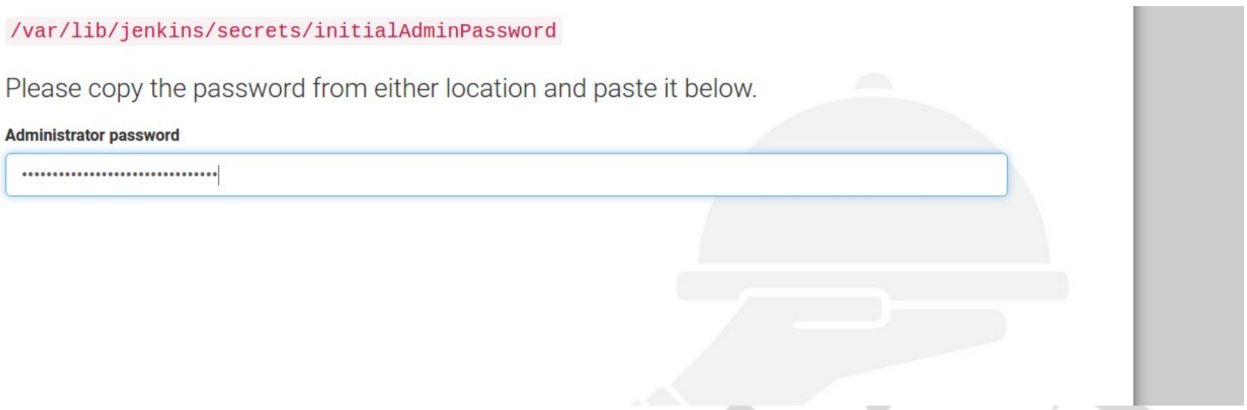
http://jenkinsIP:8080



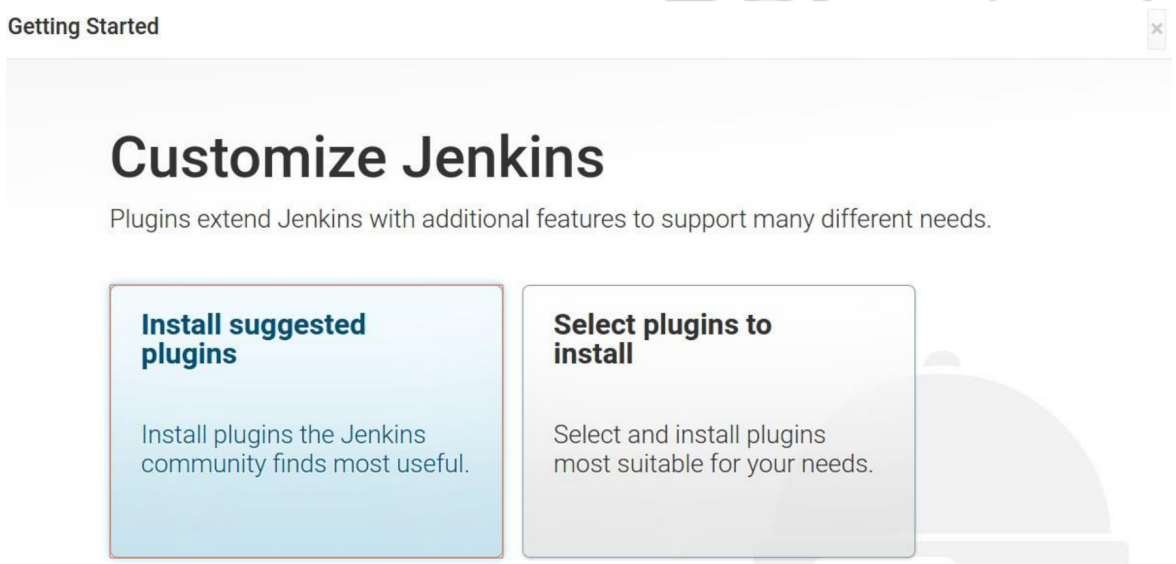
Jenkins will set a random password to unlock jenkins setup.

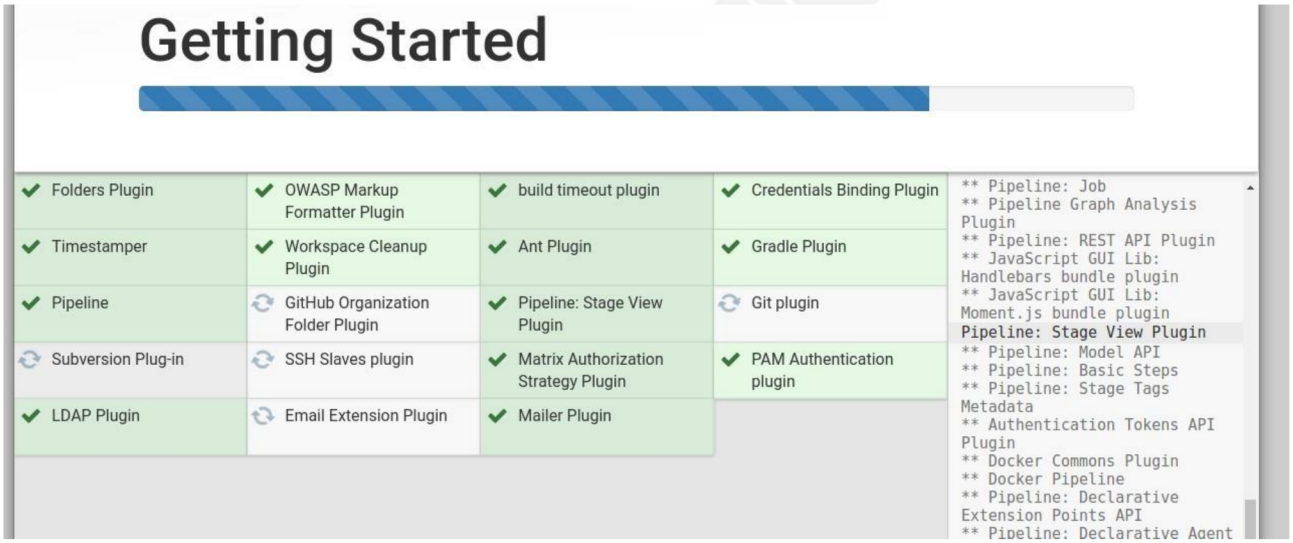
The password would be stored in /var/lib/jenkins/secrets/initialAdminPassword file. Read that file and get the password. Use that password to unlock jenkins.

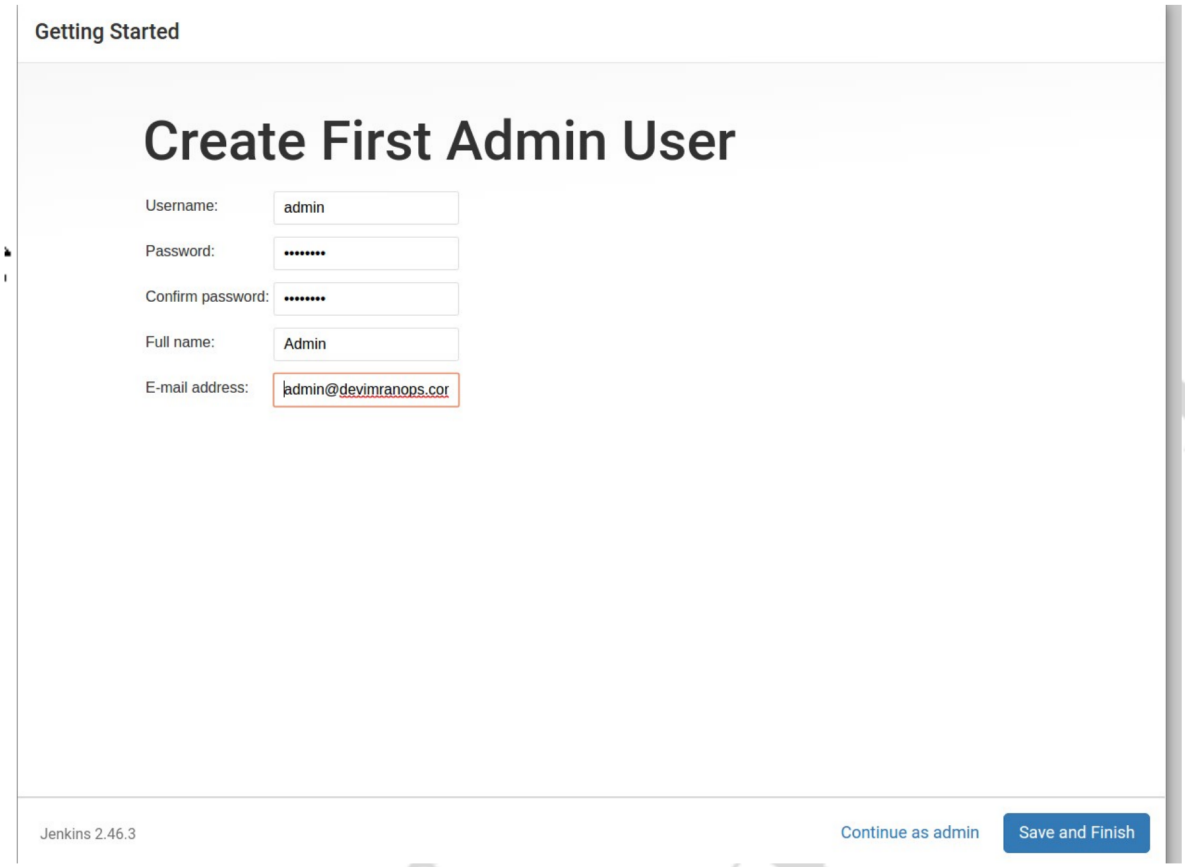
```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
f05498523c3547f89754376def663c6e
vagrant@vagrant-ubuntu-trusty-64:~$
```

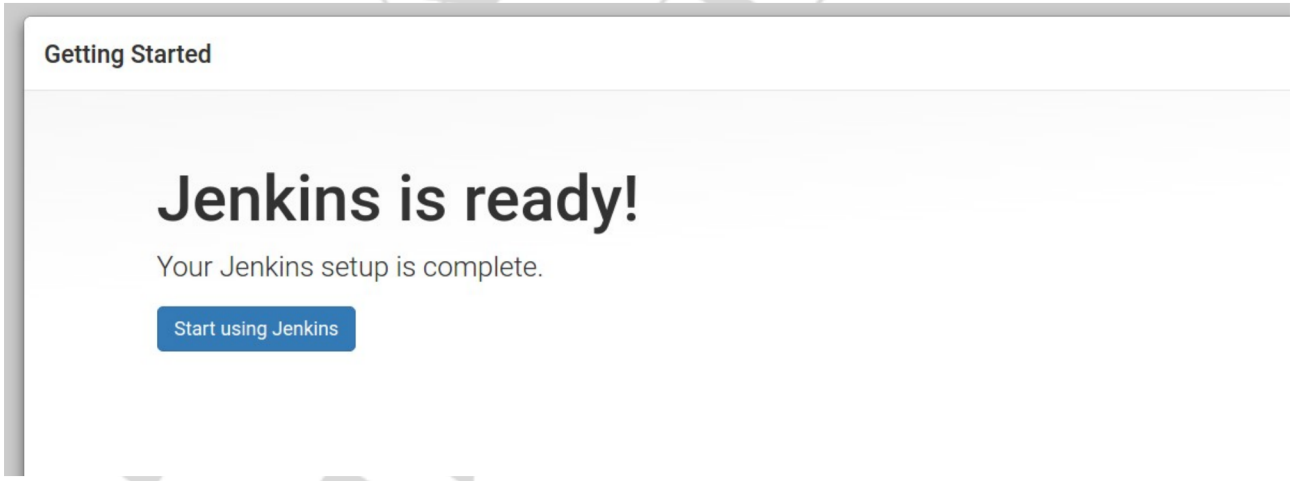


Jenkins gives you an option to install some suggested plugins at the time of setup. You can select individual plugins or install suggested group of plugins. Select suggested plugin for now.

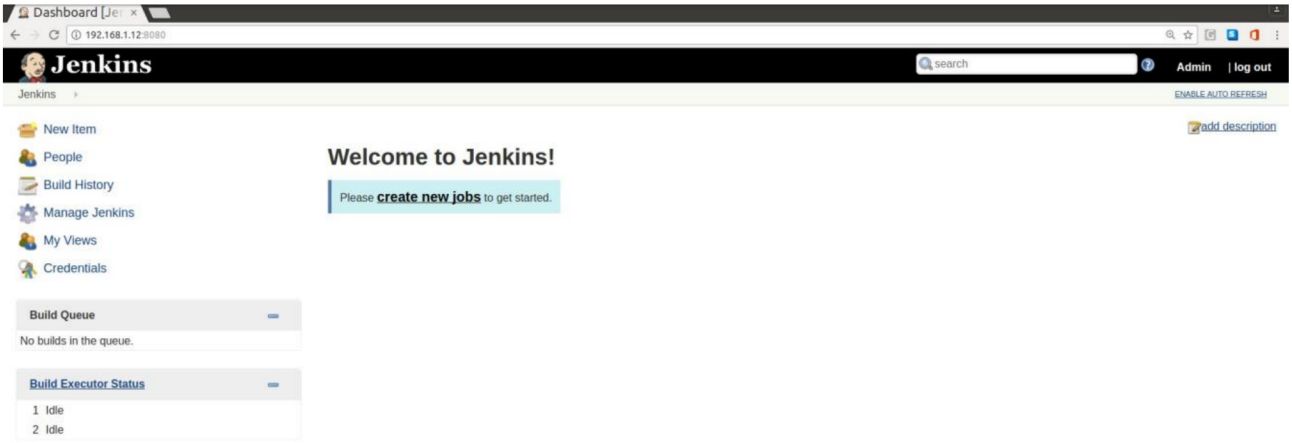





Getting Started
Create First Admin User
Username:
admin
Password:
Confirm password:
Full name:
Admin
E-mail address:
admin@devimranops.cor
Jenkins 2.46.3
Continue as admin
Save and Finish


Getting Started
Jenkins is ready!
Your Jenkins setup is complete.
Start using Jenkins

Main dashboard of jenkins.



7. Creating first jenkins job.

Jenkins manages all the tasks into something called as jobs.
Each job represents some set of activity like build process or software deploy or executing some scripts.

We will create a sample job to understand it better.

➔ Click Create new jobs or New Item => Enter name – “first-jenkins-job” => Select Freestyle project => OK