

# What is Cucumber?

Cucumber is a **Behavior Driven Development** tool used to develop test cases for the behavior of software's functionality. It plays a supporting role in automated testing.

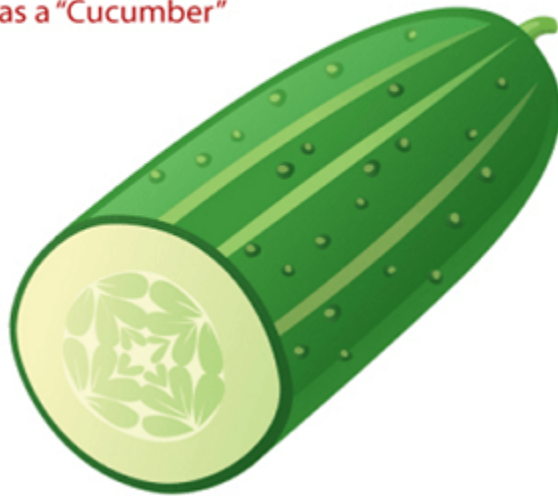
In other words,

"Cucumber is a software tool used by the testers to develop test cases for the testing of behavior of the software."

Cucumber tool plays a vital role in the development of acceptance test cases for automation testing. It is mainly used to write acceptance tests for web applications as per the behavior of their functionalities.

It follows a **BDD** (Behavior Driven Development) framework to observe the behavior of the software's functionalities.

Cucumber tool Supports  
Behavior Driven Development  
and is as Cool as a "Cucumber"



In the Cucumber testing, the test cases are written in a simple English text, which anybody can understand without any technical knowledge. This simple English text is called the **Gherkin language**.

It allows business analysts, developers, testers, etc. to automate functional verification and validation in an easily readable and understandable format (e.g., plain English).

We can use Cucumber along with Watir, Selenium, and Capybara, etc. It supports many other languages like **PHP, Net, Python, Perl**, etc.

## What is BDD?

BDD (Behavioral Driven Development) is a software development approach that was developed from **Test Driven Development (TDD)**.

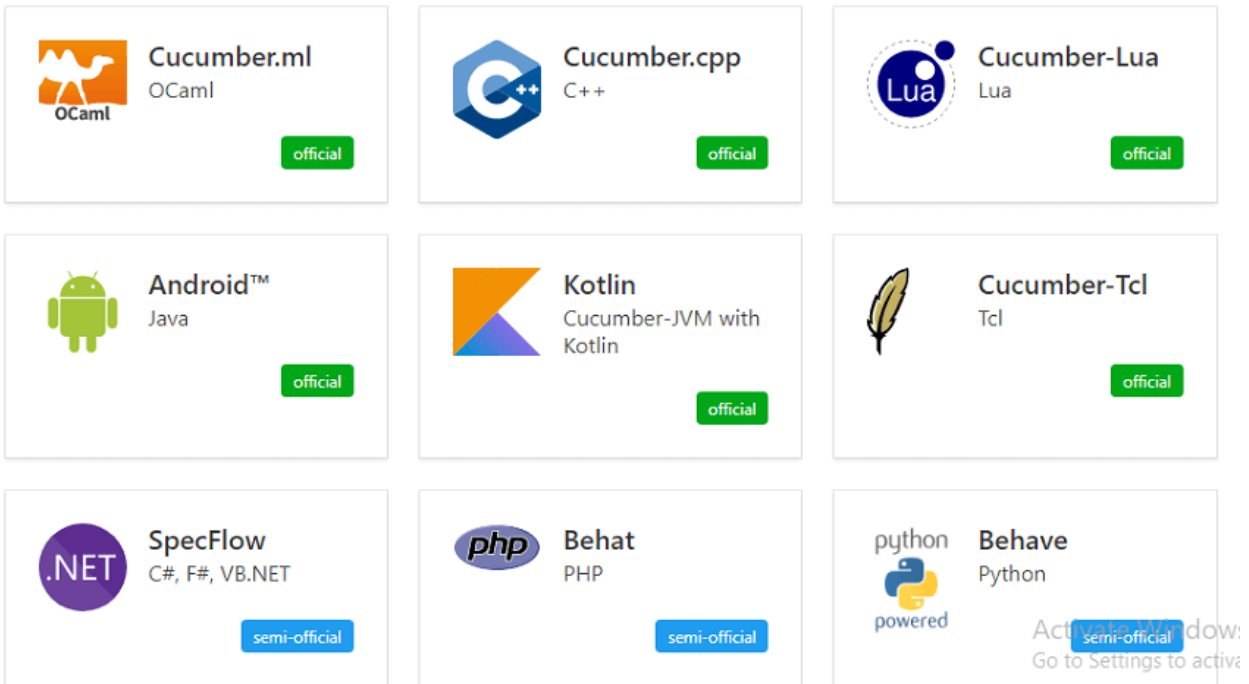
BDD includes test case development in the form of simple English statements inside a **feature file**, which is human-generated. Test case statements are based on the system's behavior and more user-focused.

BDD is written in simple English language statements rather than a typical programming language, which improves the communication between technical and non-technical teams and stakeholders.

## Which language is used in cucumber?

**Cucumber** tool was originally written in the "**Ruby**" programming language. It was exclusively used only for testing of Ruby as a complement to the **RSpec** BDD framework.

But now, Cucumber supports a variety of different programming languages including Java, JavaScript, PHP, Net, Python, Perl, etc. with various implementations. In Java, it supports **native JUnit**.



## Basic Terms of Cucumber

- Feature File
- Features
- Tags
- Scenario
- Gherkin Language
- Step Definition

## How does Cucumber Testing Works?

Cucumber test cases are written parallel with the code development of software. These test cases are called step in a Gherkin Language.

- Firstly, Cucumber tool reads the step written in a Gherkin or plain English text inside the feature file.

- Now, it searches for the exact match of each step in the step definition file. When it finds its match, then executes the test case and provides the result as pass or fail.
- The code of developed software must correspond with the BDD defined test scripts. If it does not, then code refactoring will be required. The code gets freeze only after successful execution of defined test scripts.

## Software tools supported by Cucumber

The piece of code to be executed for testing may belong to different software tools like **Selenium**, **Ruby on Rails**, etc. But cucumber supports almost all popular software platforms, and this is the reason behind Cucumber's popularity over other frameworks such as **JDave**, **Easyb**, **JBehave**, etc. Some Cucumber supported tools are given below:

- Ruby on Rails
- Selenium
- PicoContainer
- Spring Framework
- Watir

## Advantages of Cucumber Tool

- The main focus of the Cucumber Testing is on the end-user experience, as the success of the software ultimately depends on the end-user experience.
- The test case writing is very easy and understandable.
- It provides an end-to-end testing framework, unlike other tools.
- It supports almost all popular different languages like Java.net, JavaScript Ruby, PHP, etc.

- It works as a bridge between business and technical language, and this bridge is sustainable because of test cases written in a plain English text.
- The testing environment set up and execution are both very quick and easy.
- It is a well efficient tool for testing.

## How to create Selenium Maven Project in Eclipse?

Before creating Selenium Maven testing Project for cucumber testing in Eclipse, first we must have the following dependencies in our computer system:

- **Java**
- **Eclipse**
- **Cucumber Eclipse Plugin**
- **Web driver Java Client**
- **Selenium**

Here we are creating a Maven project to test the registration feature through the data table:

### Step 1

In order to create a Maven project first, Go to **File** → **New** → **Others** → **Maven** → **Maven Project** → **Next**.

After clicking the **Next**, a window will prompt. In this window, we need to provide **group Id** (group Id identifies the project uniquely across all projects). We have provided group Id as "**com.test**." You can give any name on your own choice.

Also, provide **artifact Id** (artifact Id represents the name of the project). We have provided artifact Id as "**CucumberTesting**." You can give any name on your own choice.

Click on **Finish**.

## Step 2

Open pom.xml by using the following steps:

- Go to package explorer on the left side of the Eclipse window.
- Expand the project which is created for data table testing in cucumber then select its pom.xml file.
- Now open pom.xml and add the following dependencies.

**Add dependency inside pom.xml for Selenium:** This will indicate to Maven, which Selenium jar files will be downloaded from the central repository to the local repository.

- In pom.xml file, create a dependencies tag (**<dependencies></dependencies>**), inside the project tag.
- Now, inside the dependencies tag, create a dependency tag (**<dependency></dependency>**), and provide the following information within it.
  1. **<dependencies>**
  2. **<dependency>**
  3. **<groupId> org.seleniumhq.selenium </groupId>**
  4. **<artifactId> selenium-java </artifactId>**
  5. **<version> 2.47.1 </version>**
  6. **</dependency>**
  7. **</dependencies>**

## Step 3

**Add dependency inside pom.xml for Cucumber-Java:** It will indicate to Maven; which Cucumber files will be downloaded from the central repository to the local repository.

- Now, inside the dependencies tag (**<dependencies></dependencies>**), create a dependency tag (**<dependency></dependency>**), and provide the following information within it.

1. `<dependencies>`
2. `<dependency>`
3. `<groupId> info.cukes </groupId>`
4. `<artifactId> cucumber-java </artifactId>`
5. `<version> 1.0.2 </version>`
6. `<scope> test </scope>`
7. `</dependency>`
8. `</dependencies>`

#### Step 4

**Add dependency for Cucumber-Junit:** It will indicate to Maven, which Cucumber JUnit files will be downloaded from the central repository to the local repository.

- Now, inside the dependencies tag (**`<dependencies></dependencies>`**), create a dependency tag (**`<dependency></dependency>`**), and provide the following information within it.

1. `<dependencies>`
2. `<dependency>`
3. `<groupId> info.cukes </groupId>`
4. `<artifactId> cucumber-junit </artifactId>`
5. `<version> 1.0.2 </version>`
6. `<scope> test </scope>`
7. `</dependency>`
8. `</dependencies>`

#### Step 5

**Add dependency for Junit:** It will indicate to Maven, which JUnit files will be downloaded from the central repository to the local repository.

- Now, inside the dependencies tag (**<dependencies></dependencies>**), create a dependency tag (**<dependency></dependency>**), and provide the following information within it.
1. **<dependencies>**
  2. **<dependency>**
  3. **<groupId> junit </groupId>**
  4. **<artifactId> junit </artifactId>**
  5. **<version> 4.10 </version>**
  6. **<scope> test </scope> </dependency>**
  7. **</dependencies>**

After completing all dependencies, verify binaries.

- Once pom.xml is completed successfully, then save it.
- Go to your Project → Clean - It can take a few minutes.

Now, create a package named **dataTable** under **src/test/java** folder of your project.

## Step 6

Create a Feature file:

- Inside the package dataTable, create a feature file, named **dataTable.feature**.
- Inside the feature file, write the following text.

### Feature - Data table

Verify that the new user registration is successful after passing correct inputs.

#### Scenario:

**Given** the user on the user registration page.

**When** user enter invalid data on the page

Fields	Values	
First Name	Preeti	
Last Name	Sharma	
Email Address	someone@gmail.com	



Re-enter Email Address	someone@gmail.com
Password	PASSWORD
Birthdate	02

**Then** the user registration should be successful.

- Save this file.

## Step 7

Creation of the step definition file:

- Create the step definition file inside the package dataTable with extension ".java" and named as 'dataTable.java.'
- Inside the step definition file, write the following code.
  1. **package** dataTable;
  2. **import** java.util.List;
  3. **import** org.openqa.selenium.By;
  4. **import** org.openqa.selenium.WebDriver;
  5. **import** org.openqa.selenium.WebElement;
  6. **import** org.openqa.selenium.firefox.FirefoxDriver;
  7. **import** org.openqa.selenium.support.ui.Select;
  8. **import** cucumber.annotation.en.Given;
  9. **import** cucumber.annotation.en.Then;
  10. **import** cucumber.annotation.en.When;
  11. **import** cucumber.table.DataTable;
  12. **public class** StepDefinition {
  13.   WebDriver driver = **null**;
  14.   @Given("^I am on user registration page\$")
  15.   **public void** goToFacebook() {
  16.     //Intiate web browser instance. driver = new FirefoxDriver();

```
17. driver.navigate().to("https://www.google.com/");
18. }
19.
20. @When("^I enter valid data on the page$")
21. public void enterData(DataTable table){
22.     //Initialize data table
23.     List<list> data = table.raw();
24.     System.out.println(data.get(1).get(1));
25.
26.     //Enter data
27.     driver.findElement(By.name("firstname")).sendKeys(data.get(1).get(1));
28.     driver.findElement(By.name("lastname")).sendKeys(data.get(2).get(1));
29.
30.     driver.findElement(By.name("registered_email_")).sendKeys(data.get(3).get(1));
31.     driver.findElement(By.name("registered_email_confirmation_")).
32.         sendKeys(data.get(4).get(1));
33.
34.     driver.findElement(By.name("registered_passwd_")).sendKeys(data.get(5).get(1)
35.         );
36.
37.     Select dropdownB = new Select(driver.findElement(By.name("birth_day")));
38.     dropdownB.selectByValue("12");
39.
40.     Select dropdownM = new Select(driver.findElement(By.name("birth_month")));
41.     dropdownM.selectByValue("7");
42.
43.     Select dropdownY = new Select(driver.findElement(By.name("birth_year")));
44.     dropdownY.selectByValue("1992");
45. }
```

```

43. driver.findElement(By.className("_59mt")).click();
44. // Click submit button driver.findElement(By.name("websubmit")).click();
45. }
46.
47. @Then("^User registration should be successful$")
48. public void User_registration_should_be_successful() {
49.     if(driver.getCurrentUrl().equalsIgnoreCase("https://www.google.com/")){
50.         System.out.println("Test Pass");
51.     } else {
52.         System.out.println("Test Failed");
53.     }
54.     driver.close();
55. }
56.}

```

## Step 8

After creating the step definition file now, we need to create a runner class file.

- Create a runner class inside the package dataTable with extension ".java" and named as RunTest.java.
- Inside runner class **RunTest.java**, write the following code.
  1. **package** dataTable;
  2. **import** org.junit.runner.RunWith;
  3. **import** cucumber.junit.Cucumber;
  4. @RunWith(Cucumber.class)
  5. @Cucumber.Options(format = {"pretty", "html:target/cucumber"})
  6. **public class** RunTest { }

Save this file, and run the test by using the following options:

- Select the runner class i.e., RunTest.java file inside your package.
- Right-click on it, and select the option, **Run as** → **JUnit**.

If your execution is successful, you will observe the following things:

- **Google** website gets loaded.
- We will see the home page or the page provided by the respective website.
- Data can be entered on the registration page.
- Submit button will be clicked.

## Maven

### What is Maven?

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

### What is 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 the following sequence of phases.

Phase	Handles	Description
prepare-resources	resource copying	Resource copying can be customized in this phase.
validate	Validating the information	Validates if the project is correct and if all necessary information is available.
compile	compilation	Source code compilation is done in this phase.
Test	Testing	Tests the compiled source code suitable for testing framework.
package	packaging	This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml.

install	installation	This phase installs the package in local/remote maven repository.
Deploy	Deploying	Copies the final package to the remote repository.

There are always pre and post phases to register goals, which must run prior to, or after a particular phase.

When Maven starts building a project, it steps through a defined sequence of phases and executes goals, which are registered with each phase.

Maven has the following three standard lifecycles –

- clean
- default(or build)
- site

A goal represents a specific task which contributes to the building and managing of a project. It may be bound to zero or more build phases. A goal not bound to any build phase could be executed outside of the build lifecycle by direct invocation.

The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked. For example, consider the command below. The clean and package arguments are build phases while the dependency:copy-dependencies is a goal.

```
mvn clean dependency:copy-dependencies package
```

Here the *clean* phase will be executed first, followed by the *dependency:copy-dependencies* goal, and finally *package* phase will be executed.

## Clean Lifecycle

When we execute *mvn post-clean* command, Maven invokes the clean lifecycle consisting of the following phases.

- pre-clean
- clean
- post-clean

Maven clean goal (*clean:clean*) is bound to the *clean* phase in the clean lifecycle. Its *clean:clean* goal deletes the output of a build by deleting the build directory. Thus, when *mvn clean* command executes, Maven deletes the build directory.

We can customize this behavior by mentioning goals in any of the above phases of clean life cycle.

In the following example, We'll attach *maven-antrun-plugin:run* goal to the pre-clean, clean, and post-clean phases. This will allow us to echo text messages displaying the phases of the clean lifecycle.

## MSBuild

Microsoft Build Engine, better known as MSBuild, is a free and open-source build tool set for managed code as well as native C++ code and was part of .NET Framework. Visual Studio depends on MSBuild, but not the vice versa

MSBuild files generated from Visual Studio are really like the generated ANT scripts you get from Eclipse that build your projects, remember your includes and define your dependencies. You can modify them directly for fun and profit. I like MSBuild, it fixes some of the stuff I find annoying about ANT.

## NUnit

NUnit is an open-source unit testing framework for the .NET Framework and Mono. It serves the same purpose as JUnit does in the Java world, and is one of many programs in the xUnit family.

## SpecFlow

SpecFlow is another tool that can be added on top of Selenium that helps separate out the specific tests from the code into a more human friendly format readable by anyone. It is **the . NET version of Cucumber** and uses the Gherkin language (given/when/then) to format tests.

## Jenkins

Jenkins is an open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat.

### What is Jenkins and why we use it?

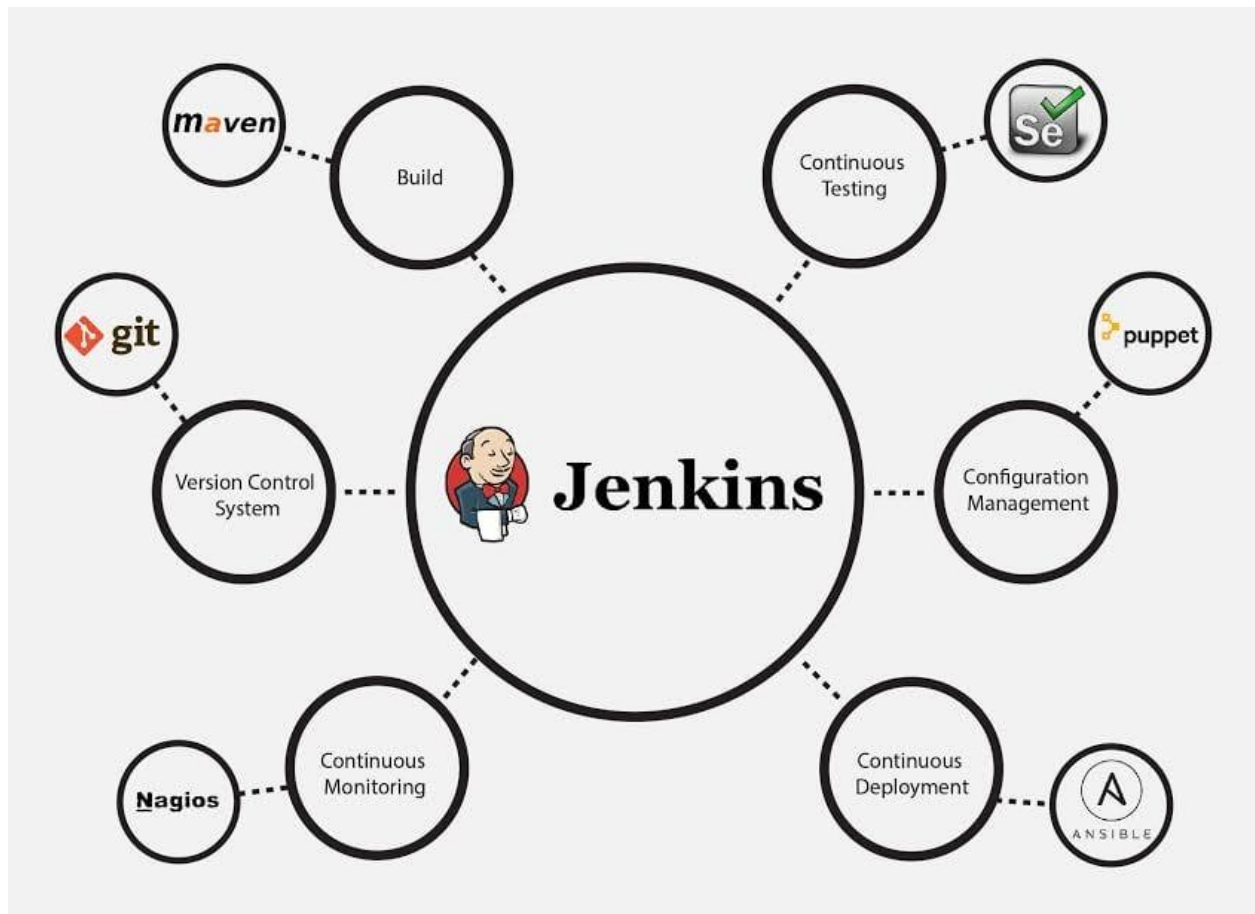
Jenkins is an open-source automation tool written in Java with plugins built for Continuous Integration purposes. Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh 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 allow 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 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 it with the community.
- It is free of cost.
- It is built with Java and hence, it is portable to all the major platforms.

There are certain things about Jenkins that separates it from other the Continuous Integration tool. Let us take a look on those points.

## Jenkins Features

The 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.

It is evident from the above points that Jenkins has a very high demand globally. Before we dive into Jenkins it is important to know what is Continuous Integration and why it was introduced.

## What is Continuous Integration?

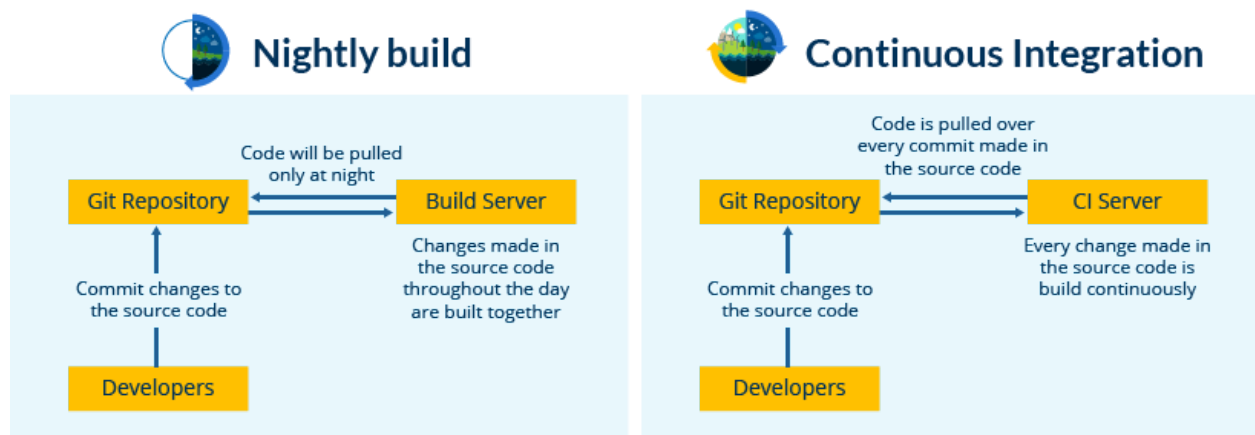
Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently. Every commit made in the repository is then built. This allows the teams to detect the problems early. Apart from this, depending on the Continuous Integration tool, there are several other functions like deploying the build application on the test server, providing the concerned teams with the build and test results, etc.

Let us understand its importance with a use-case.

## Continuous Integration Example: Nokia

I am pretty sure you all have used Nokia phones at some point in your life. In a software product development project at Nokia, there was a process called Nightly builds. Nightly builds can be thought of as a predecessor to Continuous Integration. It means that

every night an automated system pulls the code added to the shared repository throughout the day and builds that code. The idea is quite similar to Continuous Integration, but since the code that was built at night was quite large, locating and fixing of bugs was a real pain. Due to this, Nokia adopted Continuous Integration (CI). As a result, every commit made to the source code in the repository was built. If the build result shows that there is a bug in the code, then the developers only need to check that particular commit. This significantly reduced the time required to release new software.



Now is the correct time to understand how Jenkins achieves Continuous Integration.

## Continuous Integration With Jenkins

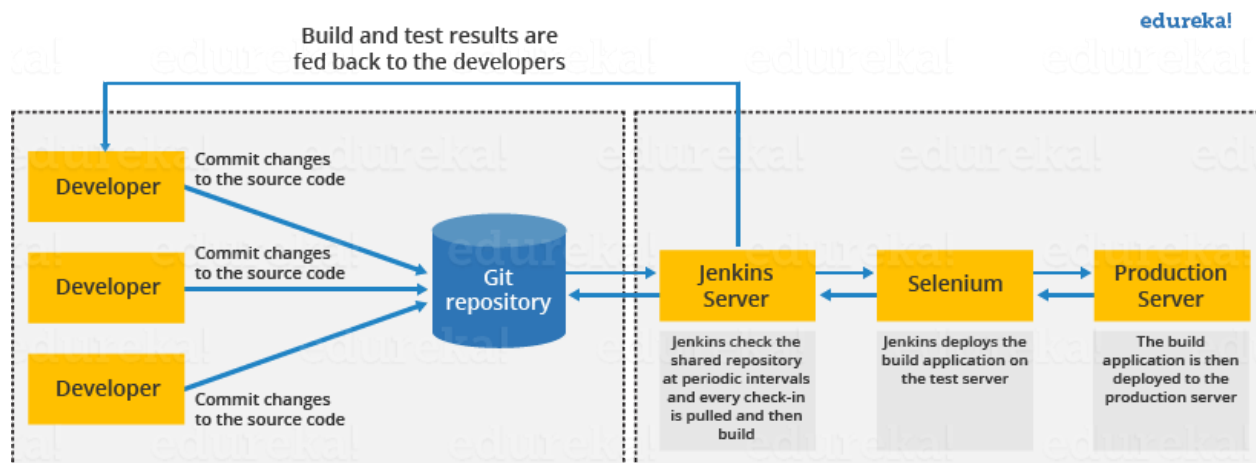
Let us imagine a scenario where the complete source code of the application was built and then deployed on test server for testing. It sounds like a perfect way to develop software, but, this process has many flaws. I will try to explain them one by one:

- Developers have to wait until 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 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.

I will first explain to you a generic flow diagram of Continuous Integration with Jenkins so that it becomes self-explanatory, how Jenkins overcomes the above shortcomings. This will help you in understanding how does Jenkins work.



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.

You now know how Jenkins overcomes the traditional SDLC shortcomings. The table below shows the comparison between “Before and After Jenkins”.

## Before and After Jenkins

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.

## Docker

### Docker Tutorial for Beginners - With Java and Spring Boot

- What is Docker?
- Why Is Docker Popular?
- Understand Docker Architecture
- How to install Docker
- What is Docker Registry?
- How to create Docker Hello World Rest API Image and Pull from Registry
- Deploy Hello World Spring Boot Application on Docker
- Docker Commands

## What is Docker?

Docker is an **open-source centralized platform designed** to create, deploy, and run applications. Docker uses **container** on the host's operating system to run applications. It allows applications to use the same **Linux kernel** as a system on the host computer, rather than creating a whole virtual operating system. Containers ensure that our application works in any environment like development, test, or production.

Docker includes components such as **Docker client, Docker server, Docker machine, Docker hub, Docker composes**, etc.

Let's understand the Docker containers and virtual machine.

## Docker Containers

Docker containers are the **lightweight** alternatives of the virtual machine. It allows developers to package up the application with all its libraries and dependencies, and ship it as a single package. The advantage of using a docker container is that you don't need to allocate any RAM and disk space for the applications. It automatically generates storage and space according to the application requirement.

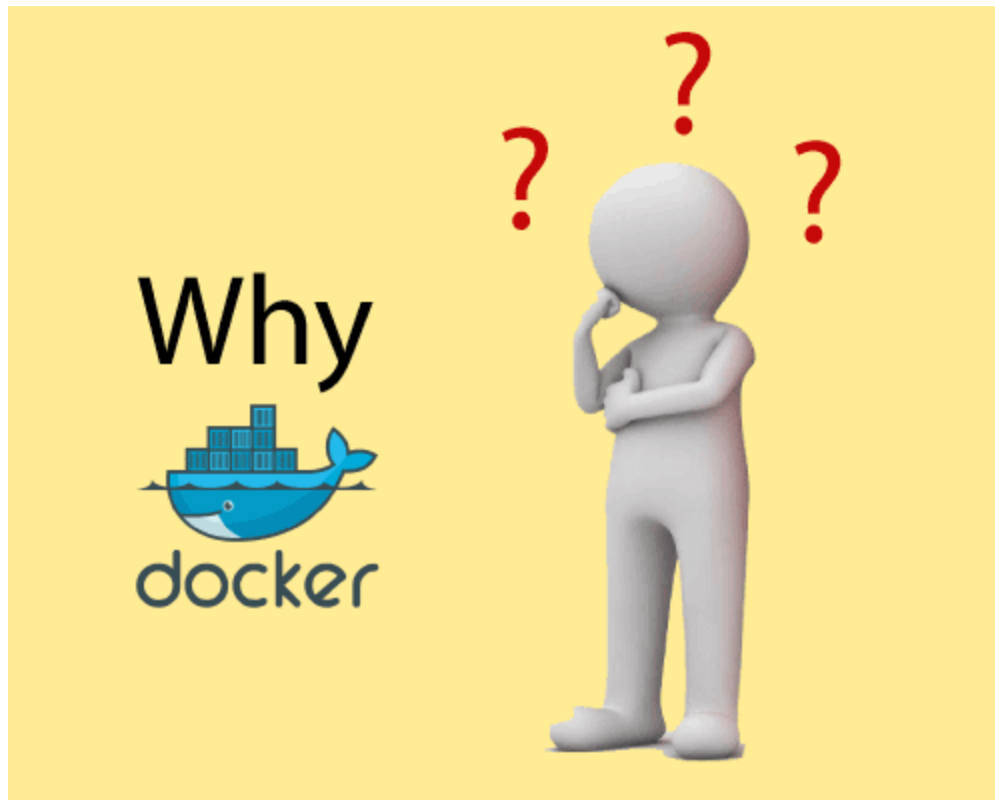
## Virtual Machine

A virtual machine is a software that allows us to install and use other operating systems (Windows, Linux, and Debian) simultaneously on our machine. The operating system in which virtual machine runs are called virtualized operating systems. These virtualized operating systems can run programs and performs tasks that we perform in a real operating system.

## Containers Vs. Virtual Machine

Containers	Virtual Machine
Integration in a container is faster and cheap.	Integration in virtual is slow and costly.
No wastage of memory.	Wastage of memory.
It uses the same kernel, but different distribution.	It uses multiple independent operating systems.

## Why Docker?



Docker is designed to benefit both the Developer and System Administrator. There are the following reasons to use Docker -

- Docker allows us to easily install and run software without worrying about setup or dependencies.
- Developers use Docker to eliminate machine problems, i.e. **"but code is worked on my laptop."** when working on code together with co-workers.
- Operators use Docker to run and manage apps in isolated containers for better compute density.
- Enterprises use Docker to securely built agile software delivery pipelines to ship new application features faster and more securely.
- Since docker is not only used for the deployment, but it is also a great platform for development, that's why we can efficiently increase our customer's satisfaction.



# Advantages of Docker

There are the following advantages of Docker -

- It runs the container in seconds instead of minutes.
- It uses less memory.
- It provides lightweight virtualization.
- It does not require full operating system to run applications.
- It uses application dependencies to reduce the risk.
- Docker allows you to use a remote repository to share your container with others.
- It provides continuous deployment and testing environment.

# Disadvantages of Docker

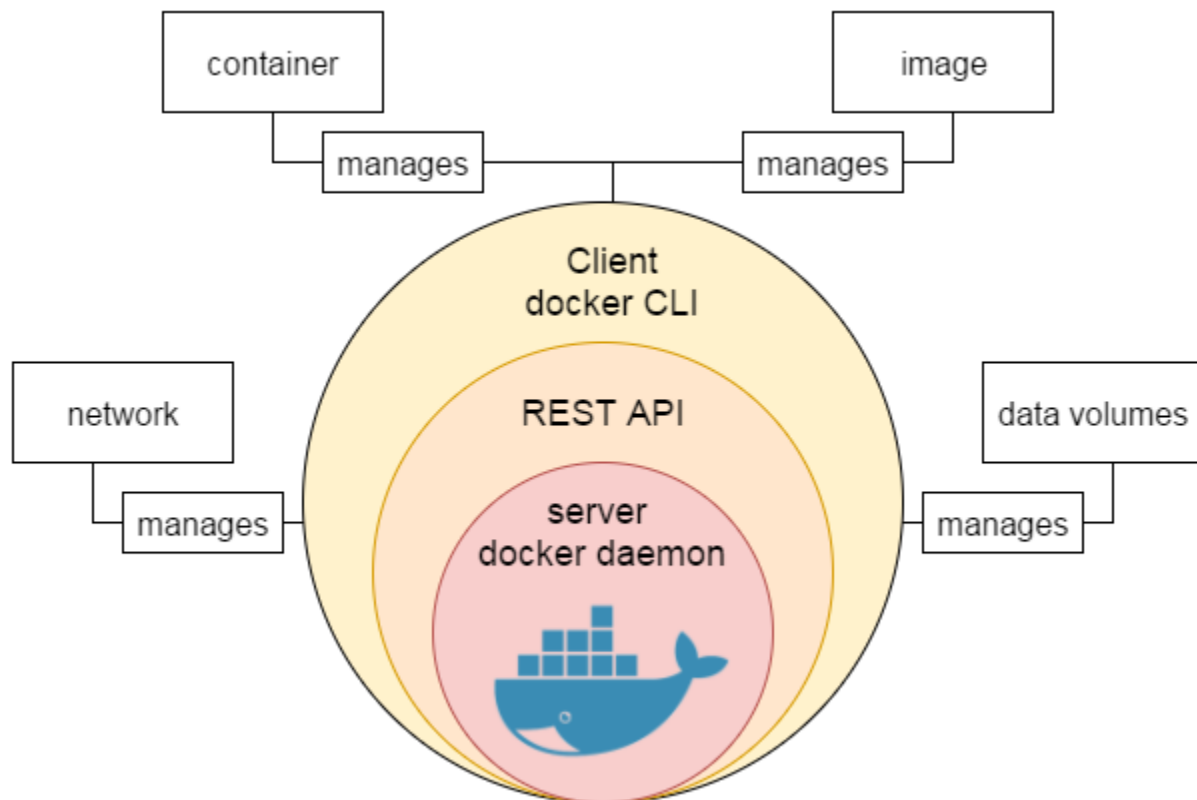
There are the following disadvantages of Docker -

- It increases complexity due to an additional layer.
- In Docker, it is difficult to manage large amount of containers.
- Some features such as container self-registration, containers self-inspects, copying files from host to the container, and more are missing in the Docker.
- Docker is not a good solution for applications that require rich graphical interface.
- Docker provides cross-platform compatibility means if an application is designed to run in a Docker container on Windows, then it can't run on Linux or vice versa.

# Docker Engine

It is a client server application that contains the following major components.

- A server which is a type of long-running program called a daemon process.
- The REST API is used to specify interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface client.



## What Does Docker Do?

- Docker provides a common approach and format for building self-contained packages for your microservices. Docker calls this package an image.
- Docker provides a runtime for running your image.

## What Does This Enable?

- You can create Docker images for a wide variety of applications irrespective of their language, runtime or operating system.
- You can run Docker images (as containers) where ever Docker runtime is available.

Are you ready to jump into the world of Docker?

## Why is Docker Popular?

Docker is popular because of the possibilities it presents for software delivery and deployment. Many common problems and inefficiencies are resolved with containers.

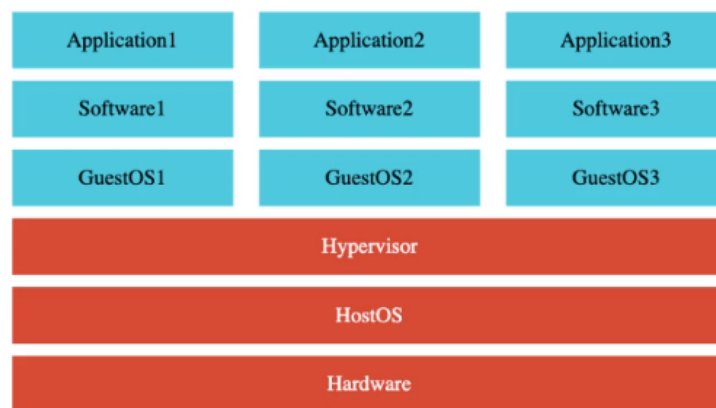
The awesome thing about Docker is, you can easily install Docker on the cloud. Most cloud providers provide container-based services

They provide services where you just need to provide a Docker image and it would automatically run on the cloud.

## The Virtual Machine Legacy

Before Docker, virtual machines were very popular.

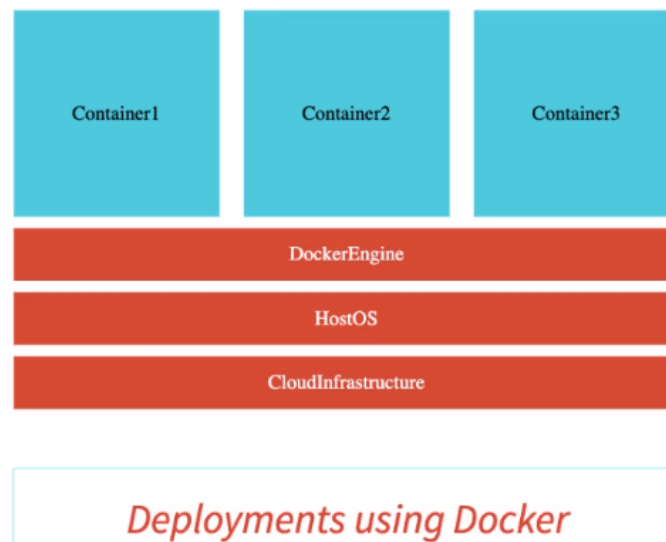
The image below shows the typical architecture with a Virtual Machine



*Deployments using Virtual Machines*

As you can see there are two operating systems present one is Guest OS and another is Host OS.

Due to this, virtual machine architecture becomes heavyweight. Hence, you will not be able to benefit from the entire power of your hardware because of virtualization.



With Docker, all that you need to run containers is a Docker Engine. Other main reasons why Docker is popular is:

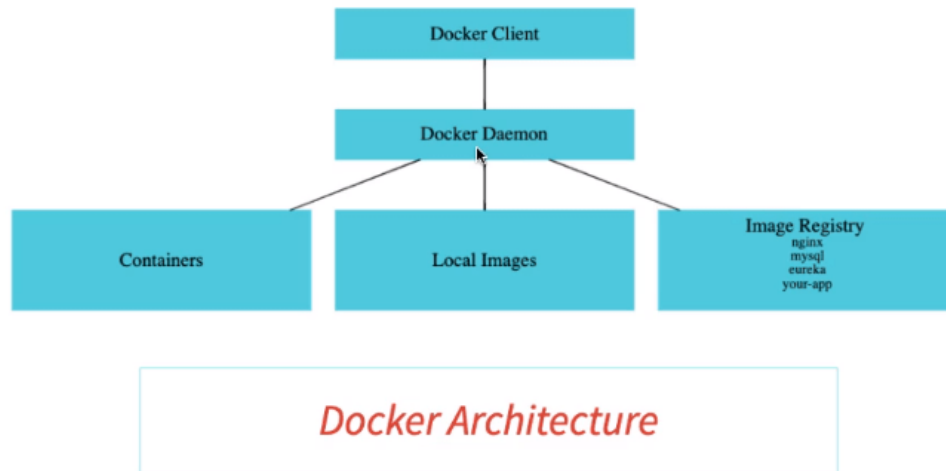
- Docker images are self-sufficient: The Docker image contains all that is needed to run a container: the libraries, the software are directly part of these containers.
- Docker is lightweight: Because there is just one OS, the host OS, Docker is relatively lightweight and therefore, is very efficient.
- Docker is convenient for the cloud: All the cloud providers provide a number of container orchestration services around Docker.
  - Azure provides a service called Azure Container Service.
  - AWS, Amazon Web Services, provides a service called Elastic Container Service and also AWS Fargate.

Kubernetes is a popular container orchestration platform. All cloud providers provide services around Kubernetes.

The rise of microservice architecture also one of the reasons for Docker's popularity.

## Docker Architecture

Docker uses a client-server architecture. The Docker client talks to the Docker **daemon**, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.



## Docker Daemon

The Docker daemon (`dockerd`) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

It is responsible for:

- Managing containers
- Managing local images
- Pulling images from the image registry if you need it

- Pushing a locally created image to an image registry

## Docker Client

The Docker client(`docker`) is the primary way that many Docker users interact with Docker. When you use commands such as `docker run`, the client sends these commands to `dockerd`, which carries them out. The `docker` command uses the Docker API. The Docker client can communicate with more than one daemon.

## Docker Registry

A Docker `registry` stores Docker images. Docker Hub is a public registry that anyone can use, and Docker looks for images on Docker Hub by default.

When you use the `docker pull` or `docker run` commands, the required images are pulled from your configured registry. When you use the `docker push` command, your image is pushed to your configured registry.

## How to Install Docker

Docker makes the life of a developers and operations team simple and installing Docker today on most of the operating systems is really a cakewalk!

We recommend installing Docker Desktop on Mac and Windows.

- [Mac](#)
- [Windows](#)

On Linux, you can find instructions for your specific distribution [here](#).

To verify whether it has been installed or not, open the command prompt and type following command:

```
$ docker --version Docker version 19.03.5, build 633a0ea
```

## What is Docker Registry?

A Docker registry contains a lot of repositories, different versions of various applications. Since Docker Hub is a public registry, anybody can access it.

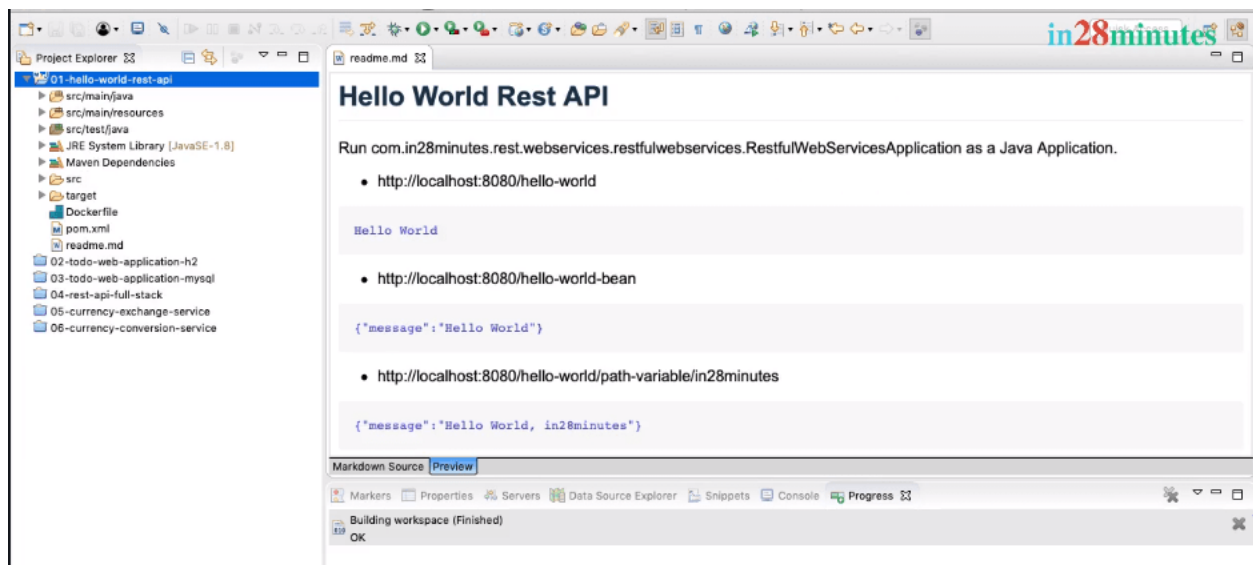
Typically, when we work in an enterprise, we use private repositories, so that our images can only be accessed by somebody who has the right credentials.

## Docker Container for Spring Boot Hello World Application

The complete project is located [here](#).

You can clone or download the entire GitHub repository and import this project as a Maven project into eclipse.

Here is the screenshot of the project setup in Eclipse.



Let's now build a Docker image for the Hello World Rest API. Let's start with following a manual approach to create the image.

### Step 1: Create a JAR File

The first thing we need to do is build a JAR for the application. This particular application is packaged as a JAR file.

Building a JAR file should be really easy. All that you need is a clean Maven build, by using `mvn clean package`.

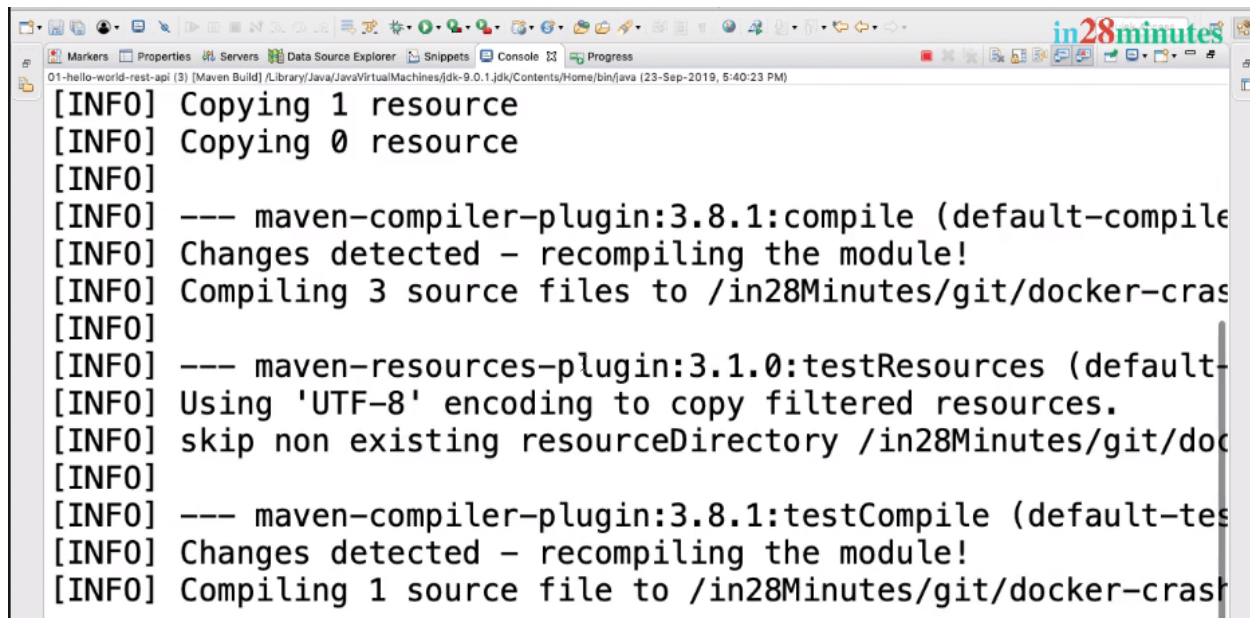
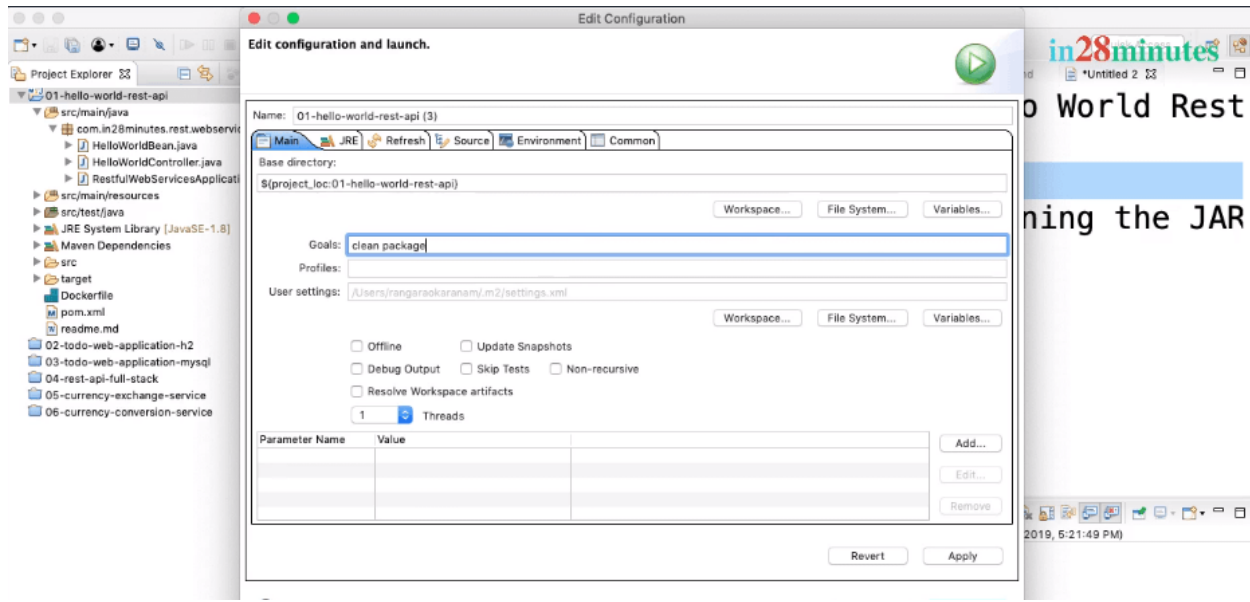
### **pom.xml:**

XML

```
1      <groupId>com.test.rest.webservices</groupId>
2
3      <artifactId>01-hello-world-rest-api</artifactId>
4
5      <version>0.0.1-SNAPSHOT</version>
6
7      <packaging>jar</packaging>
8
9      <description>Demo project for Spring Boot</description>
10
11     <name>hello-world-rest-api</name>
12
13     <parent>
14
15         <groupId>org.springframework.boot</groupId>
16
17         <artifactId>spring-boot-starter-parent</artifactId>
18
19         <version>2.1.7.RELEASE</version>
20
21         <relativePath /> <!-- lookup parent from repository -->
22
23     </parent>
```

This would compile the code, run the unit tests, and finally, a JAR would be packaged.





The name of the JAR is `hello-world-rest-api.jar`, and it is under the `target` folder.

The name of the JAR is coming in from `pom.xml`. If you search for something called `FinalName` in the build, we are configuring a name for this specific JAR, `hello-world-rest-api.jar`.

## pom.xml:

XML

```
<build>
    <finalName>hello-world-rest-api</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

## Step 2: Decide the Right Base Image

The second thing would be to set up the prerequisites for running the JAR file.

We need Java installed, and we'll use a **OpenJDK** image, such as **openjdk:8-jdk-alpine**. This is the most popular image used to run Java 8.

Alpine images is based on AlpineLinux to keep the size of the image down.

Let's now run the alpine jdk image.

```
docker run -dit openjdk:8-jdk-alpine.
```

**-dit** > **-d** is actually something which allows us to run the container in a detached mode. The container is running but we want to be able to execute the commands in using an interactive

shell. With `-it`, we are attaching a interactive shell to the running container. This allows us execute a command against the running container.

`openjdk:8-jdk-alpine` image will be downloaded from Docker Hub. It would take a little while as it is a large file — a 100 MB download.

You can run `docker images` to see the downloaded image.

```
rangaraokaranam$ cd /in28Minutes/git/docker-crash-course/01-hello-world-rest-api
rangaraokaranam$ docker run -dit openjdk:8-jdk-alpine
Unable to find image 'openjdk:8-jdk-alpine' locally
8-jdk-alpine: Pulling from library/openjdk
e7c96db7181b: Pull complete
f910a506b6cb: Pull complete
c2274a1a0e27: Pull complete
Digest: sha256:94792824df2df33402f201713f932b58cb9de94a0cd524164a0f2283343547b3
Status: Downloaded newer image for openjdk:8-jdk-alpine
8a0d79c6b1546759e778cf3b30eb6278e7585bb406a73cf489a88601b706a45e
rangaraokaranam$
```

Run `docker container ls` to see the running container - `naughty_knuth`.

## Step 3: Copy the JAR File into Docker Image

The third step is to copy the JAR into a specific image. Then, we would want to be able to run the JAR as part of the container. Let's get started with all that right now.

### Copying The JAR File

Here's the command to copy a jar file into the container:

```
docker container cp target/hello-world-rest-api.jar naughty_knuth:/tmp
```

```

rangaraokaranam$ cd /in28Minutes/git/docker-crash-course/01-hello-world-rest-api
rangaraokaranam$ docker run -dit openjdk:8-jdk-alpine
Unable to find image 'openjdk:8-jdk-alpine' locally
8-jdk-alpine: Pulling from library/openjdk
e7c96db7181b: Pull complete
f910a506b6cb: Pull complete
c2274a1a0e27: Pull complete
Digest: sha256:94792824df2df33402f201713f932b58cb9de94a0cd524164a0f2283343547b3
Status: Downloaded newer image for openjdk:8-jdk-alpine
8a0d79c6b1546759e778cf3b30eb6278e7585bb406a73cf489a88601b706a45e
rangaraokaranam$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
openjdk              8-jdk-alpine       a3562aa0b991       4 months ago       105MB
rangaraokaranam$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
ATUS                PORTS              NAMES
8a0d79c6b154        openjdk:8-jdk-alpine  "/bin/sh"          46 seconds ago     Up
44 seconds         naughty_knuth
rangaraokaranam$

```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
openjdk	8-jdk-alpine	a3562aa0b991	4 months ago	105MB

```

rangaraokaranam$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
ATUS                PORTS              NAMES
8a0d79c6b154        openjdk:8-jdk-alpine  "/bin/sh"          46 seconds ago     Up
44 seconds         naughty_knuth
rangaraokaranam$ docker container exec naughty_knuth ls /tmp
rangaraokaranam$ docker container cp target/hello-world-rest-api.jar naughty_knuth:/tmp
rangaraokaranam$ docker container exec naughty_knuth ls /tmp
hello-world-rest-api.jar
rangaraokaranam$ docker container commit naughty_knuth in28min/hello-world-rest-api:manual1
sha256:d0db83c1a9f9a580cb5458f762900d63f9d8ecb29bb7d7c353c74c9dd5556ca6
rangaraokaranam$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
in28min/hello-world-rest-api  manual1            d0db83c1a9f9       About a minute ago  122MB
openjdk              8-jdk-alpine       a3562aa0b991       4 months ago

```

```

in28min/hello-world-rest-api manual1 d0db83c1a9f9 About a minute
ago 122MB
openjdk 8-jdk-alpine a3562aa0b991 4 months ago
105MB
rangaraokaranam$ docker run in28min/hello-world-rest-api:manual1
rangaraokaranam$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
ATUS                PORTS              NAMES
8a0d79c6b154       openjdk:8-jdk-alpine  "/bin/sh"          7 minutes ago       Up
7 minutes          naughty_knuth
rangaraokaranam$ docker container commit --change='CMD ["java","-jar","/tmp/hello-world-rest-api.jar"]' naughty_knuth in28min/hello-world-rest-api:manual2
sha256:8c0e332bb5fb62416ef2408beb0bc5886e6d7bd2eb142e556e0d8542f58af8b6
rangaraokaranam$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
in28min/hello-world-rest-api manual2             8c0e332bb5fb       4 seconds ago
in28min/hello-world-rest-api manual1             d0db83c1a9f9       5 minutes ago
openjdk              8-jdk-alpine       a3562aa0b991       4 months ago

```

You can also use the ID of the container.

PowerShell

```

docker container cp target/hello-world-rest-api.jar
54cf414254e48d5f68c4d468b2dd4cbdd95d17f9e2074fdb9df7f64987697f2b:/tmp

```

The command successfully executed, which means our JAR file is now copied into the running container.

Let's see what is inside the container:

Shell

```

docker container exec naughty_knuth ls /tmp

```

You will see the jar file `hello-world-rest-api.jar`

## Step 4: Running The JAR Within The Container

Now, we would want to run this JAR. Before that, we need to do two simple steps.

## Saving The Container Image

The first step is actually to save the container which we have created as an image.

Here's the command:

Shell

```
docker container commit naughty_knuth reponame/hello-world-rest-api:manual1
```

1

We are giving a repository name of `reponame/hello-world-rest-api` and a tag of `manual1`.

If you run `docker images` you would see a new image.

## Running The Image

Let's see if we can run this image.

Shell

```
docker run reponame/hello-world-rest-api:manual1
```

1

Let's see if there is anything running at this point in time.

Shell

```
docker container ls
```

1

You will see that the container which we launched just is not really running at all!

How can we get it to run the application?

## Attaching The JAR To Startup

The reason it is not running is we did not attach anything to run at the launch of the container.

We would want to launch a Java application jar at the startup of the container. We copied the JAR file into this image, but did not specify that the JAR file has to be launched at startup.

How can we do that?

Shell

```
docker container commit --change='CMD
["java","-jar","/tmp/hello-world-rest-api.jar"]' naughty_knuth
reponame:hello-world-rest-api:manual2
```

1

2

We are adding a startup command by using an option `--change`.

CONTAINER ID	IMAGE	COMMAND	CREATED	ST
ATUS	PORTS	NAMES		
8a0d79c6b154	openjdk:8-jdk-alpine	"/bin/sh"	7 minutes ago	Up
7 minutes		naughty_knuth		

```
rangaraokaranam$ docker container commit --change='CMD ["java","-jar","/tmp/hello-world-rest-api.jar"]' naughty_knuth in28min/hello-world-rest-api:manual2
sha256:8c0e332bb5fb62416ef2408beb0bc5886e6d7bd2eb142e556e0d8542f58af8b6
rangaraokaranam$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
in28min/hello-world-rest-api	manual2	8c0e332bb5fb	4 seconds ago
in28min/hello-world-rest-api	manual1	d0db83c1a9f9	5 minutes ago
openjdk	8-jdk-alpine	a3562aa0b991	4 months ago

```
105MB
rangaraokaranam$ docker run -p 8080:8080 in28min/hello-world-rest-api:manual2
docker: Error response from daemon: driver failed programming external connectivity on endpoint elegant_tu (b13e154cbd7a42adc1e4f918a6c77b14ec92cc65ddaec9305918673fb5d226d8): Error starting userland proxy: listen tcp 0.0.0.0:8080: bind: address already in use.
```

## Running The Container Again

Let's do `docker images`, and a new image is seen to be created with tag as `manual2`. Now, we can run it saying `docker run reponame/hello-world-rest-api:manual2`. We would also want to publish the port, as in `8080:8080`.

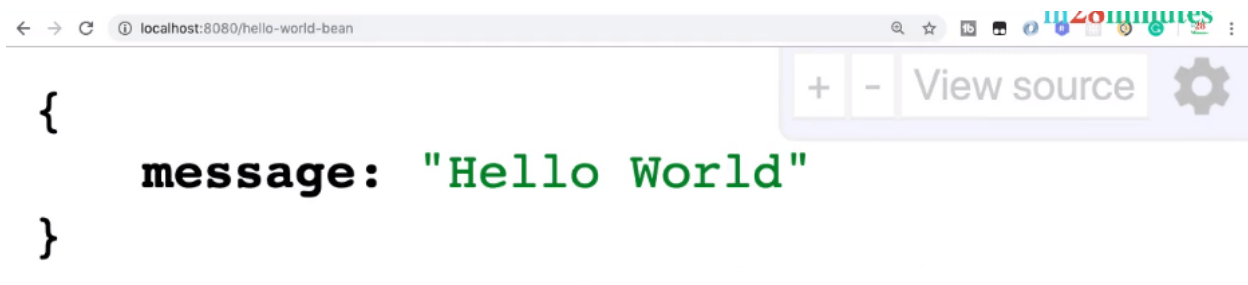


```
docker run -p 8080:8080 reponame/hello-world-rest-api:manual2
```

If you have any container running on 8080, you can stop it.

```
ctory      : Autowiring by type from bean name 'org.springframework.boot.autoconfigure
.web.servlet.error.ErrorMvcAutoConfiguration$DefaultErrorViewResolverConfiguration' v
ia constructor to bean named 'org.springframework.boot.web.servlet.context.Annotation
ConfigServletWebServerApplicationContext@490d6c15'
2019-09-23 12:27:59.920 DEBUG 1 --- [          main] o.s.b.f.s.DefaultListableBeanFa
ctory      : Autowiring by type from bean name 'org.springframework.boot.autoconfigure
.web.servlet.error.ErrorMvcAutoConfiguration$DefaultErrorViewResolverConfiguration' v
ia constructor to bean named 'spring.resources-org.springframework.boot.autoconfigure
.web.ResourceProperties'
2019-09-23 12:28:00.110 DEBUG 1 --- [          main] .s.b.w.e.t.TomcatServletWebServ
erFactory : Code archive: /tmp/hello-world-rest-api.jar
2019-09-23 12:28:00.112 DEBUG 1 --- [          main] .s.b.w.e.t.TomcatServletWebServ
erFactory : Code archive: /tmp/hello-world-rest-api.jar
2019-09-23 12:28:00.114 DEBUG 1 --- [          main] .s.b.w.e.t.TomcatServletWebServ
erFactory : None of the document roots [src/main/webapp, public, static] point to a d
irectory and will be ignored.
2019-09-23 12:28:00.186 INFO 1 --- [          main] o.s.b.w.embedded.tomcat.TomcatW
ebServer  : Tomcat initialized with port(s): 8080 (http)
2019-09-23 12:28:00.269 INFO 1 --- [          main] o.apache.catalina.core.Standard
Service   : Starting service [Tomcat]
2019-09-23 12:28:00.271 INFO 1 --- [          main] org.apache.catalina.core.Standa
```

If we go to the browser and refresh, the URL continues to work.



This image we can share with anyone in the world and they can run in all environment they would wish to.

## Creating the Image using Dockerfile

The command is shown below:



```
docker build -t reponame/hello-world-rest-api:dockerfile1
```

Couple of important things to note:

- `-t` option is to specify the repository name and the tag
- `.` is used to specify build context

```
rangaraokaranam$ open .
rangaraokaranam$ docker build -t in28min/hello-world-rest-api:dockerfile1 .
Sending build context to Docker daemon 16.9MB
Step 1/3 : FROM openjdk:8-jdk-alpine
----> a3562aa0b991
Step 2/3 : ADD target/hello-world-rest-api.jar hello-world-rest-api.jar
----> 21cc134d0446
Step 3/3 : ENTRYPOINT ["sh", "-c", "java -jar /hello-world-rest-api.jar"]
----> Running in 4db897817321
Removing intermediate container 4db897817321
----> 981aa844d405
Successfully built 981aa844d405
Successfully tagged in28min/hello-world-rest-api:dockerfile1
rangaraokaranam$
```

The build context (current folder) is sent to Docker Daemon and after that, different steps are executed. Finally, we have a image built, and the image is also tagged with this specific tag name that we gave it.

## Running The Container

Let's run the command:

```
docker run -p 8080:8080 reponame/hello-world-rest-api:dockerfile1
```

The application is up and running. Let's see what we have in the browser at the URL `localhost:8080/hello-world`.



# Hello World

Cool! We have the application up and running in the browser as well.

We have now created a **Dockerfile**, and used that **Dockerfile** to build a image.

## 2. Using the Jib Plugin To Create Docker Images

The Dockerfile Maven plugin is from Spotify, and the great thing about it is that it provides a clear separation between what the **Dockerfile** does, and what the Spotify Dockerfile Maven plugin does.

### Building a Docker Image with Spotify Maven Plugin

Here are the important steps:

- Instructions are specified in the **Dockerfile**
- Spotify Dockerfile Maven plugin is responsible for integrating the building of image with the Maven build process

### Jib - Alternative To **Dockerfile**

One of the popular alternatives to Spotify Dockerfile Maven Plugin is **Jib**.

Jib is a Maven plugin for building Docker and OCI images for your Java applications.

### Understanding The Open Container Initiative (OCI)

**OCI is Open Container Initiative.** Just like you have interfaces and you have implementations, OCI is like an interface, and Docker image is an implementation of OCI.

OCI is a specification, and any container image creator can choose to adhere to the OCI standards. Docker has taken active part in creating the OCI specification, and also adheres to the OCI specification.

### Getting Started With Jib

The most important point you need to remember about Jib is, you don't need `Dockerfile` at all.

Here's a project ready to use with JIB Plugin -

<https://github.com/reponameutes/docker-crash-course/blob/master/01-hello-world-rest-api/code-backup/02-jib.md>

Important configuration is shown below:

A few important things to note:

- In the configuration, we are using `USE_CURRENT_TIMESTAMP`. By default, JIB adds a constant timestamp to ensure reproducibility of images. We are overriding it to use the current timestamp.
- `package` - We are configuring the JIB plugin to run during the package phase

If we have some source code and generate a image today, and then use the same source code to generate another image one year later, we should get the same image. This is called reproducibility of images. The thing is, if you have different creation timestamp on these images, the final hash for these images will be different. By default, Jib puts a creation date of `1-1-1970`. In computer terminology, that's what is called the "**Epoch**", the start of time for computers.

You can now run `mvn clean package`, and see the magic unfold.

```

rangaraokaranam$ mvn clean package -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building hello-world-rest-api 0.0.1-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ 01-hello-world-rest-api -
--
[INFO] Deleting /in28Minutes/git/docker-crash-course/01-hello-world-rest-api/target

```

```

PSHOT. If you'd like to specify a different tag, you can set the <to><image> paramete
r in your pom.xml, or use the -Dimage=<MY IMAGE> commandline flag.
[WARNING] Setting image creation time to current time; your image may not be reproduc
ible.
[INFO]
[INFO] Containerizing application to Docker daemon as 01-hello-world-rest-api:0.0.1-S
NAPSHOT...
[INFO]
[INFO] Container entrypoint set to [java, -cp, /app/resources:/app/classes:/app/libs/
*, com.in28minutes.rest.webservices.restfulwebservices.RestfulWebServicesApplication]
[INFO]
[INFO] Built image to Docker daemon as 01-hello-world-rest-api:0.0.1-SNAPSHOT
[INFO] Executing tasks:
[INFO] [=====] 100.0% complete
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:47 min

```

The first time with Jib, the build will be a little slow because it is downloading a lot of stuff.

The artifact id is being used as the repository name, and the version is being used as the tag.

Jib does not need Dockerfile

One interesting thing you'd note in here is, Jib is able to automatically detect that `RestfulWebServicesApplication`, is the class that we would want to run.

Remember, that we don't have a Dockerfile configured at all. So, Jib is automatically able to detect the type of project and identify the class to run as ENTRYPOINT.

Examining The Jib generated Image

Now, before we go any further, let's see actually what's inside that image. If we do a **docker history** and use the image name created by Jib, there are multiple images present.

```
...se/01-hello-world-rest-api — AWS, Azure, Microservices and Full Stack Courses with Java and Spring Boot ...1-hello-world-rest-api — AWS, Azure, Microservices and Full Stack Courses with Java and Spring Boot
rangaraokaranam$ docker history 01-hello-world-rest-api:0.0.1-SNAPSHOT
IMAGE          CREATED          CREATED BY          SIZE
COMMENT
dde42907edbc   59 seconds ago  jib-maven-plugin:1.6.1  3.32kB
classes
<missing>      59 seconds ago  jib-maven-plugin:1.6.1  42B
resources
<missing>      59 seconds ago  jib-maven-plugin:1.6.1  16.9MB
dependencies
<missing>      49 years ago    bazel build ...       106MB
<missing>      49 years ago    bazel build ...       1.93MB
<missing>      49 years ago    bazel build ...       15.1MB
<missing>      49 years ago    bazel build ...       1.82MB
rangaraokaranam$
```

The base image which is viewed by Jib is something called distroless Java. We can change the base image as well, a little later. For now, the important thing to remember is Jib builds the images in multiple layers.

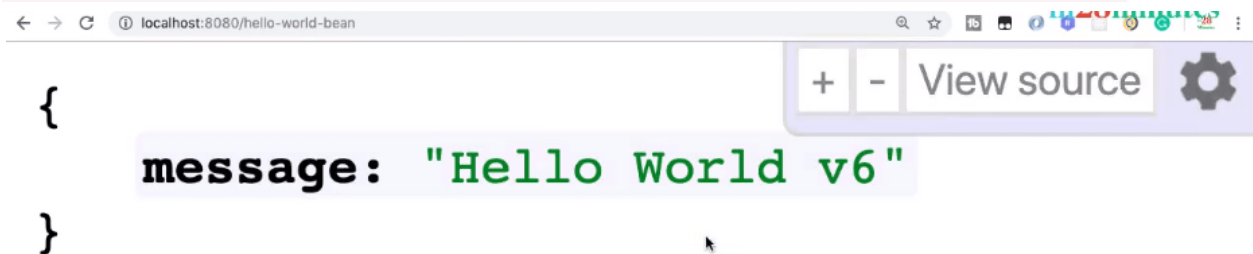
Instead of copying a fat JAR in, JIB explodes the fat JAR, copies the dependencies first, and the classes later:

- The dependencies are being copied in first - **16.9 MB**
- then the resources - which are the property files
- and after that, the actual classes

## Running The Jib Generated Image

You can also run this image.

```
docker run -p 8080:8080 webservices/01-hello-world-rest-api
```



We have seen that Jib makes it very easy to generate your Docker images. All that we needed to do was, add a simple plugin in and that's it. No **Dockerfile** is required.

### Configurations Supported By Jib

Jib also supports a lot of configuration.

The default base image for Jib is distroless Java. To specify that you want to use **openjdk:alpine**, you can configure the image name as well. You can configure the tags as well.

It is possible to configure JVM options, and the main class. If the default main class picked up by Jib is not the right one, you can configure the main class as well. You can also configure the container port for the image.

If your needs are very very simple, Jib is an awesome tool. It adheres to all the best practices of generating container images.

## Jib Is Java Specific

However, Jib is Java-specific. If you want to build container images for projects outside Java, Jib is not the right option. We would recommend you explore the FAQ around Jib for more information on the Jib plugin.

## 3. Using Fabric8 Docker Maven Plugin To Create Docker Images

With Spotify Dockerfile Maven plugin, we needed a Dockerfile, and the plugin would integrate the creation of the image with the Maven build process.

With Jib, we don't really need a Dockerfile at all. The plugin would automatically create the container image based on the best practices.

## Introducing The Fabric8 Docker Maven Plugin

The next plugin which we will quickly look at is something called Fabric8 Docker Maven Plugin. The home page is [dmp.fabric8.io](http://dmp.fabric8.io).

One interesting fact is that this was one of the first Maven plugins integrated well with Docker.

The original intention for the Docker Maven plugin was as a replacement of **Dockerfile**.

Today, almost everybody uses **Dockerfile** to specify the instructions to create the image. When this plugin came in in 2015, **Dockerfile** was not as popular as it is today.

## The Options With The Fabric8 Plugin

Here's a simple configuration of the plugin to use an existing Dockerfile:

The complete project is present here:

<https://github.com/reponameutes/docker-crash-course/blob/master/01-hello-world-rest-api/code-backup/03-fabric8-docker-plugin-using-dockerfile.md>

You can add the plugin and specify that in the package phase, we want to execute the build goal. It then uses the **Dockerfile** present in the root of the project. When you execute **mvn clean package**, it would build the image using that **Dockerfile**.

In that sense, it's similar to Spotify Docker Maven plugin.

You can also configure the name of the final jar file as a maven property.

## Configuring The `pom.xml` - Options

The other possibility with the Fabric8 Docker Maven plugin is to specify the `Dockerfile` instructions directly inside your plugin configuration.

This was a popular way of doing it before `Dockerfile` really became famous, and is not common today.

Here's an example:

## Why So Many Plugins?

One question does arise: why do we need these many plugins? The most important thing is, you don't really need to be an expert in all these plugins.

The documentation of these plugins is quite good. If you face a problem, you should be able to easily fix it by looking at the documentation. It's very important, at a high-level, to be aware of the important things behind each plugin.

## The essential Differences Among Plugins

The Dockerfile Maven plugin provides you a clear separation. You specify all the build instructions in the `Dockerfile`, and the plugin integrates the build of the Docker image into your Maven's lifecycle.

Jib, on the other hand, takes complete control. It doesn't need the Dockerfile at all, and creates the image for you.

The Fabric8 Docker Maven plugin, on the other hand, provides two options. It says, either you can use the Dockerfile, or you can specify XML configuration telling what needs to be done.

If you look at the industry today, most of the people use external `Dockerfile` configuration.



Using an external **Dockerfile** gives us complete control over the Docker image that is being created.

The other important thing is, this is language-independent. We can use this Dockerfile whether we are trying to build a Java project, a Python project, or a frontend project.

## Docker Commands

Docker CommandDescriptionUsage with an Example

**docker version**

Show the docker CLI version

```
$ docker --version Docker version 19.03.5, build 633a0ea
```

Run a command in a new container

```
$ docker run -p 8080:8080 reponame/hello-world-rest-api:manual OR  
$ docker run -p 8080:8080 webservices/01-hello-world-rest-api
```

Remove one or more containers by specifying **container Id** separating by **(,)**

Remove one or more images by specifying **image id** separating by **(,)**

## What Is Ansible?

This is the first blog of my Ansible tutorial series on “What Is Ansible”. I hope you will enjoy reading it.

Ansible is an open source IT Configuration Management, Deployment & Orchestration tool. It aims to provide large productivity gains to a wide variety of automation challenges. This tool is very simple to use yet powerful enough to automate complex multi-tier IT application environments.

On this blog you will be learning:

- What is Ansible?
- [Why do we need Ansible?](#)
- [Advantages of using Ansible](#)
- [What Ansible can do?](#)
- [Ansible Architecture](#)
- [Ansible in DevOps](#)
- [Real-Life usage of Ansible by NASA](#)
- Some [Ansible terms](#), to help you understand Ansible better.

## Why Do We Need Ansible?

Well before I tell you what is Ansible, it is of utmost importance to understand the problems that were faced before Ansible.

Let us take a little flashback to the beginning of networked computing when deploying and managing servers reliably and efficiently has been a challenge. Previously, system administrators managed servers by hand, installing software, changing configurations, and administering services on individual servers.

As data centers grew, and hosted applications became more complex, administrators realized they couldn't scale their manual systems management as fast as the applications they were enabling. It also hampered the velocity of the work of the developers since the development team was agile and releasing software frequently, but IT operations were spending more time configuring the systems. That's why server provisioning and configuration management tools came to flourish.

Consider the tedious routine of administering a server fleet. We always need to keep updating, pushing changes, copying files on them etc. These tasks make things very complicated and time consuming.

But let me tell you that there is a solution to the above stated problem. The solution is – *Ansible*.

But before I go ahead to explain you all about Ansible, let me get you familiarized with few Ansible terminologies:

## Ansible Terms:

- **Controller Machine:** The machine where Ansible is installed, responsible for running the provisioning on the servers you are managing.
- **Inventory:** An initialization file that contains information about the servers you are managing.
- **Playbook:** The entry point for Ansible provisioning, where the automation is defined through tasks using YAML format.
- **Task:** A block that defines a single procedure to be executed, e.g. Install a package.
- **Module:** A module typically abstracts a system task, like dealing with packages or creating and changing files. Ansible has a multitude of built-in modules, but you can also create custom ones.
- **Role:** A pre-defined way for organizing playbooks and other files in order to facilitate sharing and reusing portions of a provisioning.
- **Play:** A provisioning executed from start to finish is called a play. In simple words, execution of a playbook is called a play.
- **Facts:** Global variables containing information about the system, like network interfaces or operating system.
- **Handlers:** Used to trigger service status changes, like restarting or stopping a service.

Ansible is a helpful tool that allows you to create groups of machines, describe how these machines should be configured or what actions should be taken on them. Ansible issues all commands from a central location to perform these tasks.

No other client software is installed on the node machines. It uses SSH to connect to the nodes. Ansible only needs to be installed on the control machine (the machine from which you will be running commands) which can even be your laptop. It is a simple solution to a complicated problem.

I am not boasting off when I say that Ansible has filled up all the holes in Configuration Management and IT Orchestration world. You will know it too, when you take a look at the benefits of Ansible mentioned below:

## Advantages Of Using Ansible



**Simple:** Ansible uses a simple syntax written in YAML called *playbooks*. YAML is a human-readable data serialization language. It is extraordinarily simple. So, no special coding skills are required and even people in your IT organization, who do not know what is Ansible can likely read a playbook and understand what is happening. Ansible always executes tasks in order. It is simple to install too . Altogether the simplicity ensures that you can get started quickly.



**Agentless:** Finally, Ansible is completely agentless. There are no agents/software or additional firewall ports that you need to install on the client systems or hosts which you want to automate. You do not have to separately set up a management infrastructure which includes managing your entire systems, network and storage. Ansible further reduces the effort required for your team to start automating right away.



**Powerful & Flexible:** Ansible has powerful features that can enable you to model even the most complex IT workflows. In this aspect, Ansible's *batteries included approach* (This philosophy means that something is self-sufficient, comes out-of-the-box ready to use, with everything that is needed) can manage the infrastructure, networks, operating systems and services that you are already using, as Ansible provides you with hundreds of modules to manage them. Together Ansible's capabilities allow you to orchestrate the entire application environment regardless of where it is deployed.



**Efficient:** No extra software on your servers means more resources for your applications. Also, since Ansible modules work via JSON, Ansible is extensible with modules written in a programming language you already know. Ansible introduces modules as basic building blocks for your software. So, you can even customize it as per your needs. For e.g. If you have an existing message sending module which sends messages in plain-text, and you want to send images too, you can add image sending features on top of it.

## What Ansible Can Do?

Ansible is usually grouped along with other Configuration Management tools like Puppet, Chef, SaltStack etc. Well, let me tell you, Ansible is not just limited to

Configuration Management. It can be used in many different ways too. I have mentioned some of them below:



Provisioning: Your apps have to live somewhere. If you're PXE (Preboot eXecution Environment) booting and kick starting bare-metal servers or Virtual Machines, or creating virtual or cloud instances from templates, Ansible & Ansible Tower helps to streamline this process. For example, if I want to test the debug version of an application that is built with Visual C++, I ought to meet some prerequisite requirements like having Visual C++ library DLLs (msvcr100d.dll). I will also need Visual Studio installed in your computer. This is when Ansible makes sure that the required packages are downloaded and installed in order to provision my application.



Configuration Management: It establishes and maintains consistency of the product performance by recording and updating detailed information which describes an enterprise's hardware and software. Such information typically includes the versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices. For e.g. If you want to install the new version of Tomcat on all of the machines present in your enterprise, it is not feasible for you to manually go and update each and every machine. You can install Tomcat in one

go on all of your machines with Ansible playbooks and inventory written in the most simple way. All you have to do is list out the IP addresses of your nodes in the inventory and write a playbook to install Tomcat. Run the playbook from your control machine & it will be installed on all your nodes.



## Explore Curriculum



Application Deployment: When you define your application with Ansible, and manage the deployment with Ansible Tower, teams are able to effectively manage the entire application life cycle from development to production. For example, let's say I want to deploy the Default Servlet Engine. There are a number of steps that needs to be undergone to deploy the engine.

- Move a .war application from dropins directory to apps directory
- Add server.xml file
- Navigate to the webpage to see your application.

But why worry about performing these steps one by one when we have a tool like Ansible. All you need to do is list these tasks in your Ansible playbook and sit back watching Ansible executing these tasks in order.



**Security and Compliance:** When you define your security policy in Ansible, scanning and remediation of site-wide security policy can be integrated into other automated processes. And it'll be integral in everything that is deployed. It means that, you need to configure your security details once in your control machine and it will be embedded in all other nodes automatically. Moreover, all the credentials (admin users id's & passwords) that are stored within Ansible are not retrievable in plain-text by any user.



**Orchestration:** Configurations alone don't define your environment. You need to define how multiple configurations interact and ensure the disparate pieces can be managed as a whole. Out of complexity and chaos, Ansible brings order. Ansible provides Orchestration in the sense of aligning the business request with the applications, data, and infrastructure. It defines the policies and service levels through automated workflows, provisioning, and change management. This creates an application-aligned infrastructure that can be scaled up or down based on the needs of each application.



For example, Consider the situation where I want to deploy a new website in place of my existing one. For that, we will remove the existing website, and deploy our new website, and restart the load balancer or the web cluster if needed. Now, if we just did something like this, users would notice downtime because we have not removed live traffic going to these machines via the load balancer. So, we need some type of pre-task, where we tell the load balancer to put this web server into maintenance mode, so that we can temporarily disable traffic from going to it, as it gets upgraded. Let's say, I added a block up here, that says a pre-task will be to disable web node in the load balancer.

So, this is our pre-task, where we disable traffic, then down here, we upgrade the node using these various tasks. Finally, we need some type of post-task, which will enable traffic to this web node again, by taking it out of maintenance mode. These tasks can be written in Ansible playbooks and hence it helps to orchestrate the environment.

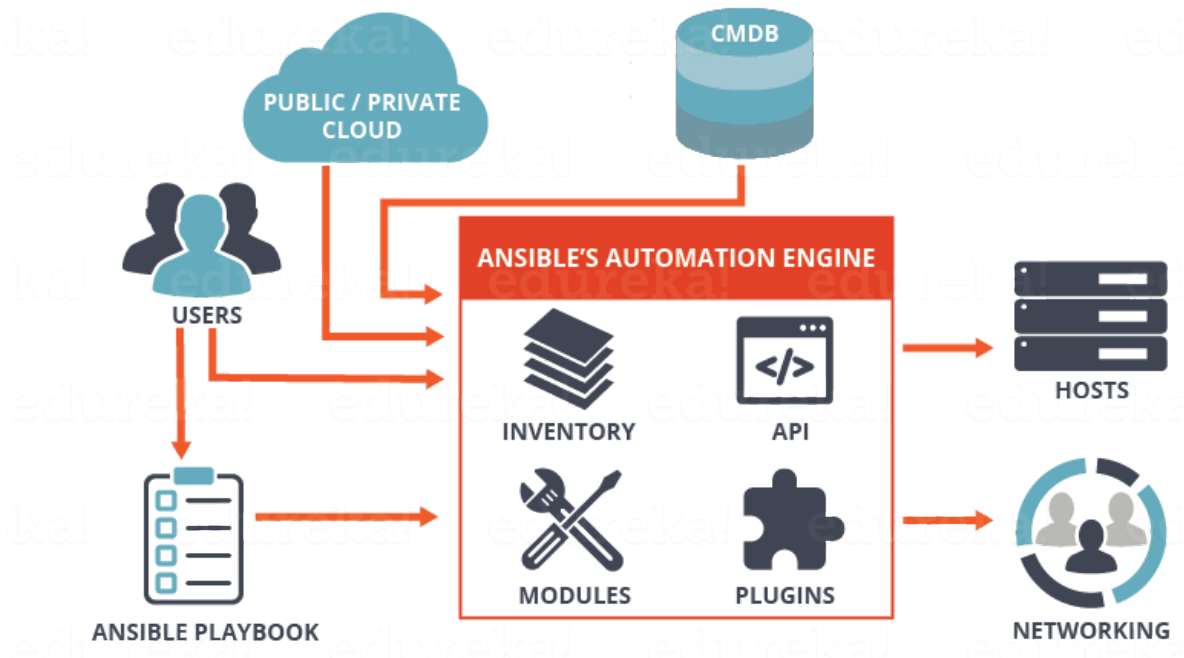
You will understand the working of Ansible better when you get a clear picture of its (Ansible's) architecture.

## What is Ansible & its Architecture?

Ansible architecture is fairly straightforward. Refer to the diagram below to understand the Ansible architecture:

## ANSIBLE ARCHITECTURE

edu:



As you can see, in the diagram above, the Ansible automation engine has a direct interaction with the users who write playbooks to execute the Ansible Automation engine. It also interacts with cloud services and Configuration Management Database (CMDB).

The Ansible Automation engine consists of:

- Inventories: Ansible inventories are lists of hosts (nodes) along with their IP addresses, servers, databases etc. which needs to be managed. Ansible then takes action via a transport – SSH for UNIX, Linux or Networking devices and WinRM for Windows system.
- APIs: APIs in Ansible are used as transport for Cloud services, public or private.
- Modules: Modules are executed directly on remote hosts through playbooks. The modules can control system resources, like services, packages, or files (anything really), or execute system commands. Modules do it by acting on system files, installing packages or making API calls to the service network. There are over 450 Ansible-provided modules that automate nearly every part of your environment. For e.g.
  - Cloud Modules like *cloudformation* which creates or deletes an AWS cloud formation stack;

- Database modules like *mssql\_db* which removes MYSQL databases from remote hosts.
- Plugins: Plugins allows to execute Ansible tasks as a job build step. Plugins are pieces of code that augment Ansible's core functionality. Ansible ships with a number of handy plugins, and you can easily write your own. For example,
  - *Action* plugins are front ends to modules and can execute tasks on the controller before calling the modules themselves.
  - *Cache* plugins are used to keep a cache of 'facts' to avoid costly fact-gathering operations.
  - *Callback* plugins enable you to hook into Ansible events for display or logging purposes.

There are a few more components in Ansible Architecture which are explained below:

**Networking:** Ansible can also be used to automate different networks. Ansible uses the same simple, powerful, and the agentless automation framework IT operations and development are already using. It uses a data model (a playbook or role) that is separate from the Ansible automation engine that easily spans different network hardware.

**Hosts:** The hosts in the Ansible architecture are just node systems which are getting automated by Ansible. It can be any kind of machine – Windows, Linux, RedHat etc.

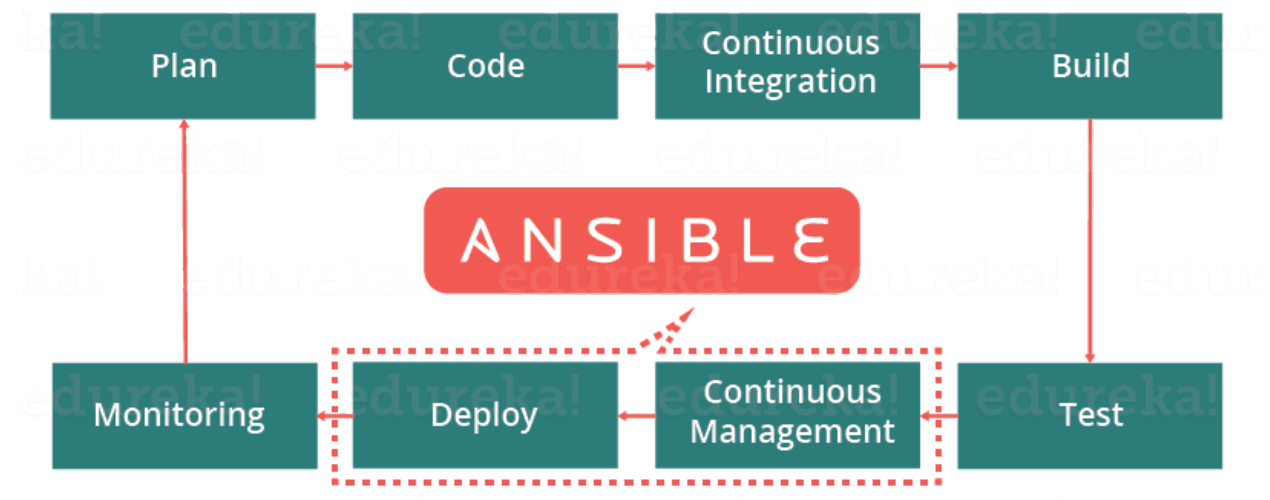
**Playbooks:** Playbooks are simple files written in YAML format which describes the tasks to be executed by Ansible. Playbooks can declare configurations, but they can also orchestrate the steps of any manual ordered process, even if it contains jump statements. They can launch tasks synchronously or asynchronously.

**CMDB :** It is a repository that acts as a data warehouse for IT installations. It holds data relating to a collection of IT assets (commonly referred to as configuration items (CI)), as well as to describe relationships between such assets.

Cloud: It is a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server. You can launch your resources and instances on cloud and connect to your servers.

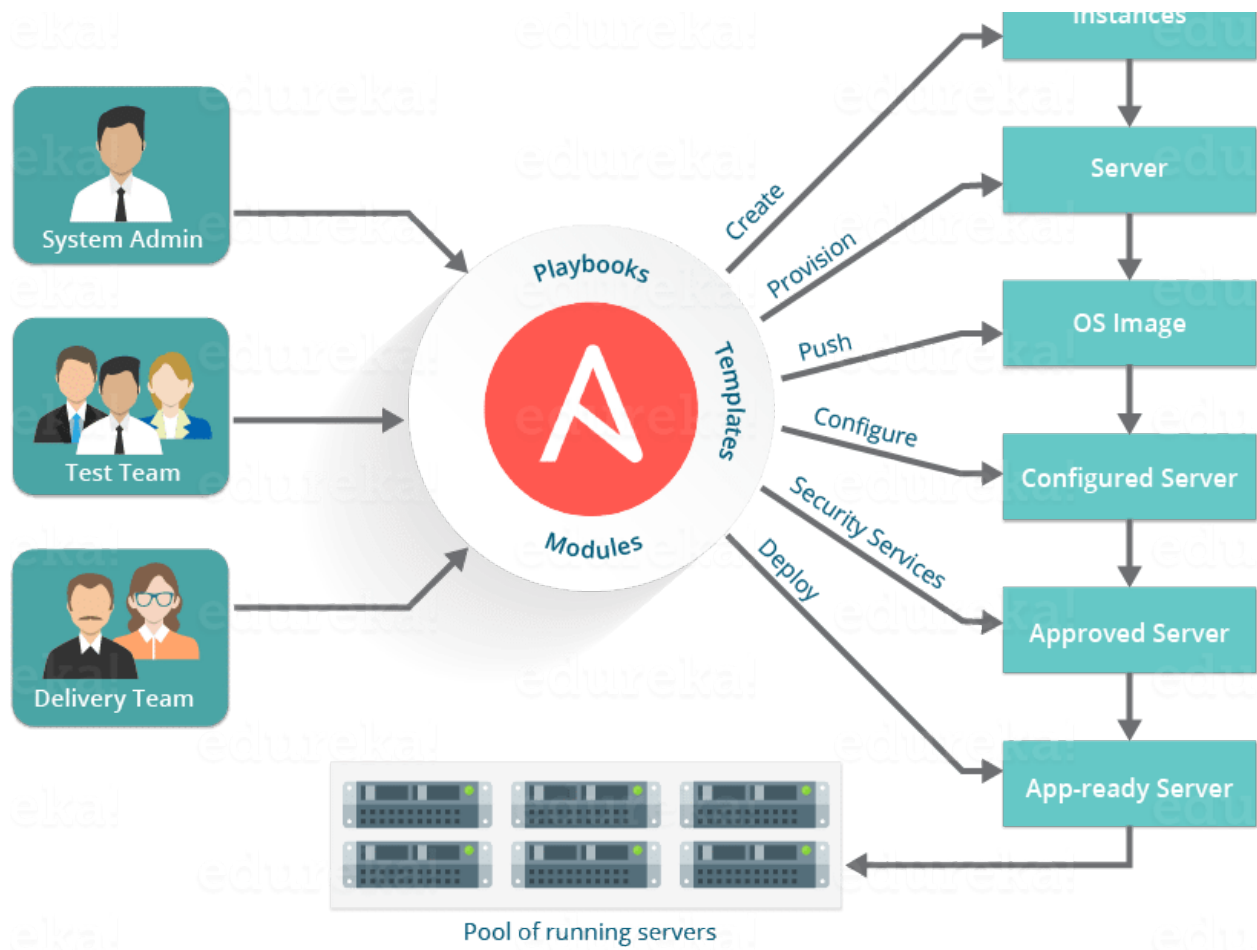
## What is Ansible in DevOps?

In DevOps, as we know development and operations work is integrated. This integration is very important for modern test-driven application design. Hence, Ansible integrates this by providing a stable environment to both development and operations resulting in smooth orchestration. Refer to the image below to see how Ansible fits into DevOps:



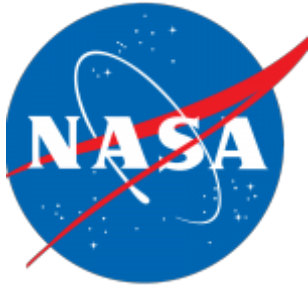
Let us discuss now how Ansible manages the entire DevOps infrastructure. When developers begin to think of infrastructure as part of their application i.e as Infrastructure as code (IaC), stability and performance become normative. Infrastructure as Code is the process of managing and provisioning computing infrastructure (processes, bare-metal servers, virtual servers, etc.) and their configuration through machine-processable definition files, rather than physical hardware configuration or the use of interactive configuration tools. This is where Ansible automation plays a major role and stands out among its peers.

In DevOps, Sysadmins work tightly with developers, development velocity is improved, and more time is spent doing activities like performance tuning, experimenting, and getting things done, and less time is spent fixing problems. Refer to the diagram below to understand how the tasks of sysadmins and other users are simplified by Ansible.



At this point you know how beneficial using Ansible is. So, now let us see a real life example of how NASA has benefited through Ansible.

# Ansible Case Study – A Real Life Usage by NASA



Let us consider the business challenge that was faced by NASA.

NASA needed to move 65 applications from a traditional hardware based data center to a cloud-based environment for better agility and cost savings. The rapid timeline resulted in many applications being migrated 'as it is' to a cloud environment. This created an environment which spanned multiple virtual private clouds (VPCs) and AWS accounts that could not be managed easily. Even simple things, like ensuring every system administrator had access to every server, or simple security patching, were extremely cumbersome.

The solution was to leverage Ansible Tower to manage and schedule the cloud environment.

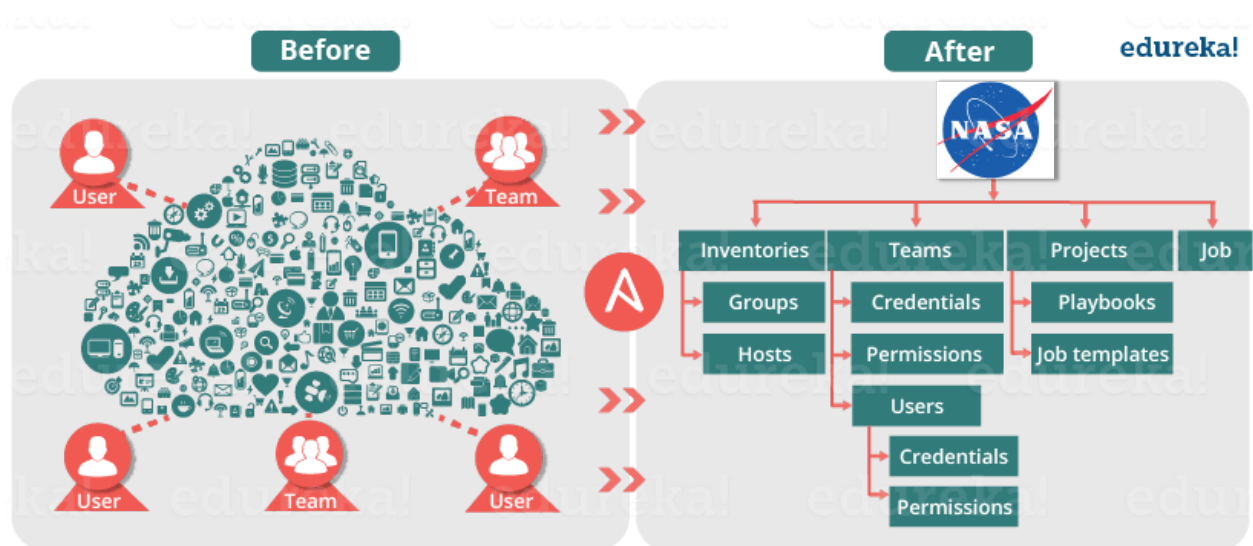
Hence, to solve the problems that NASA had with lack of centralized management and a diverse environment, they evaluated multiple solutions and decided on an implementation of Ansible Tower. NASA is now leveraging Ansible Tower to manage their environment in a very organized and scheduled way.

How NASA is using Ansible:

Ansible Tower provided with a dashboard which provided the status summary of all hosts and jobs which allowed NASA to group all contents and manage access permissions across different departments. It also helped to split up the organization by associating content and control permission for groups as well.

Ansible Tower is a web-based interface for managing Ansible. One of the top items in Ansible users' wishlists was an easy-to-use UI for managing quick deployments and monitoring one's configurations. Ansible management came up with Ansible Tower in response.

Further, Ansible divided the tasks among teams by assigning various roles. It managed the clean up of old job history, activity streams, data marked for deletion and system tracking info. Refer to the diagram below to understand how Ansible has simplified the work of NASA.



As a result, NASA has achieved the following efficiencies:

- NASA web app servers are being patched routinely and automatically through Ansible Tower with a very simple 10-line Ansible playbook.
- Ansible is also being used to re-mediate security issues and was leveraged to re-mediate OpenSSL issues. This not only saved time but allowed to quickly re-mediate a very daunting security issue.

- Every single week, both the full and mobile versions of [www.nasa.gov](http://www.nasa.gov) are updated via Ansible, generally only taking about 5 minutes to do.
- OS level user accounts for mission critical staff are continually checked and created if missing. Now, everyone who needs access has access, even if that means adding or removing a user almost instantly from all servers.
- NASA has also integrated Ansible facts into their CMDB, CloudAware, for better management visibility of entire AWS inventory. As a result, it became possible to organize the inventory of AWS resources in a very granular way that was not possible before.
- Ansible is also used to ensure that the environment is compliant with necessary Federal security standards as outlined by FedRAMP and other regulatory requirements.

Results:

As a result of implementing Ansible, NASA is better equipped to manage its AWS environment. Ansible allowed NASA to provide better operations and security to its clients. It has also increased efficiency as a team.

If we see by the numbers:

- Updating [nasa.gov](http://nasa.gov) went from over 1 hour to under 5 minutes
- Security Patching updates went from a multi-day process to 45 minutes





### [See Batch Details](#)

- Achieving near real-time RAM and disk monitoring (accomplished without agents)
- Provisioning OS Accounts across entire environment in under 10 minutes
- Baselining standard AMIs (Amazon Machine Image) went from 1 hour of manual configuration to becoming an invisible and seamless background process
- Application stacks set up time reduced from 1-2 hours to under 10 minutes per stack.

## Puppet

What is Puppet for?

Puppet is an **open source software configuration management and deployment tool**. It's most commonly used on Linux and Windows to pull the strings on multiple application servers at once. But you can also use Puppet on several platforms, including IBM mainframes, Cisco switches, and Mac OS servers.

## What Is Puppet – Key Metrics

Below are few facts about Puppet:

- Large installed base: Puppet is used by more than 30,000 companies worldwide including Google, Red Hat, Siemens, etc. along with several universities like Stanford and Harvard law school. An average of 22 new organizations per day use Puppet for the first time.
- Large developer base: Puppet is so widely used that lots of people develop for it. Puppet has many contributors to its core source code.
- Long commercial track record: Puppet has been in commercial use since 2005, and has been continually refined and improved. It has been deployed in very large infrastructures (5,000+ machines) and the performance and scalability lessons learned from these projects have contributed in Puppet's development.
- Documentation: Puppet has a large user-maintained wiki with hundreds of pages of documentation and comprehensive references for both the language and its resource types. In addition, it's actively discussed on several mailing lists and has

a very popular IRC channel, so whatever your Puppet problem, it's easy to find the answer.

- Platform support: Puppet Server can run on any platform that supports ruby for ex: CentOS, Microsoft Windows Server, Oracle Enterprise Linux etc. It not only supports the new operating systems but it can also run on relatively old and out-of-date OS and Ruby versions as well.

## Chef

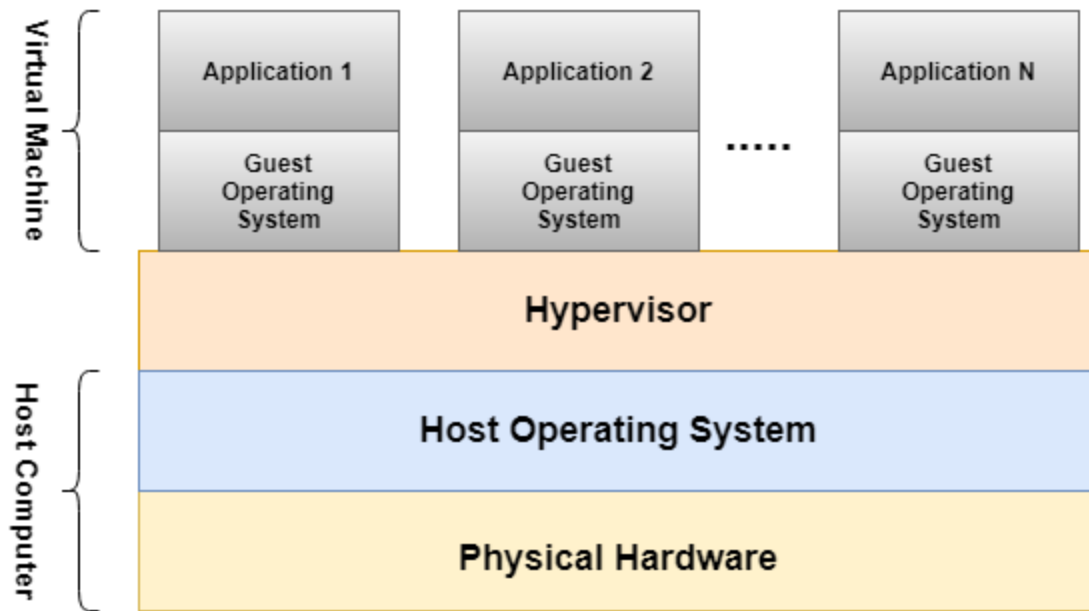
What is chef and how it works?

Chef is **an open source technology developed by Opscode**. ... 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.

## Vagrant

Vagrant is an open-source tool that helps us to automate the creation and management of Virtual Machines. In a nutshell, we can specify the configuration of a virtual machine in a simple configuration file, and Vagrant creates the same Virtual machine using just one simple command. It provides command-line interfaces to automate such tasks.

[Virtual Machine](#) is a machine that does not exist physically but can be used just like a physical computer. Any task that can be done on a physical machine can also be executed in a virtual machine. But Virtual Machine is built on top of a physical system, and multiple virtual machines can be created in a single physical computer. All the virtual machines share the same hardware, but each of them might have a separate operating system. The physical system that hosts all the virtual machines is called the **Host Computer**. The medium that separates the Host Computer hardware and the virtual environments is something called **Hypervisor**, or **Hyper-V**.



*Fig: Structure of Virtual Machine*

Each Virtual Machine should have its own configuration like operating system, CPUs, RAM, Hard Disk Memory, networking, etc. And the creation of such VMs, manually configuring all the properties is really a hectic task. In this scenario, Vagrant comes into the picture.

### **Why Vagrant?**

An application consists of several components which need to be configured properly to run the application. For example, a modern web application might have components like Java, JavaScript, Python, etc. as a language, MySQL, Oracle, MongoDB, etc. as Databases, other components like webserver, load-balancer, API Gateway, Message Queue, etc. based on requirements.

Prior to Vagrant, all these components need to be set up manually. During the setup process, a lot of issues are faced-

- In every machine, the setup needs to be done separately, which takes a lot of time.
- The manual configuration might be erroneous, which needs to debug and fix every time.

- The development, testing, and production environment should be identical. But due to this manual installation and setup of the components, there might be a slight difference which provides us a lot of pain, because, in such a scenario, the application might run in a development environment, but face issues in the production one.

Vagrant is the modern solution to all these problems. Instead of setting up all the components manually, Vagrant provides us the feature of using one configuration file (called Vagrantfile), where all the required software components and their configuration information are specified. So, while this same configuration file is executed in multiple machines, Vagrant creates the Virtual Machine on top of each physical machine, installs and setups all the mentioned components automatically, and provides a ready start-to-work Virtual Machine. And since this provides a Virtual Machine, there is no need to bother about whether one software component would run on Windows OS or Linux OS and what should be their configuration and also developer, QA both work in a separate machine, but both machines should have a completely identical setup.

**Terminologies Of Vagrant:** Before getting into the details of Installation and how to use Vagrant, let's first discuss the basic terminologies related to Vagrant.

**1. Vagrant Box:** The basic unit of Vagrant setup is Vagrant Box. Just like Docker Image, Vagrant Box is a self-contained image of the [Operating System](#). More specifically, it is a packaged Virtual Machine. Instead of installing an operating system and all the software components inside a VM manually,

- The vagrant box is a ready-made base image of a Virtual Machine Environment.
- For example, if there is a need to get some VM with all default setups of Spring Boot application development, one can get the same Vagrant Box. All that is required to do is to download the Vagrant Box and run it. Vagrant creates the VM and development work can be started immediately.

- A lot of Vagrant Box is present in [Vagrant Cloud box catalog](#), which we can use as a base image for our own Virtual Machine.

**2. Vagrant File:** Vagrant maintains one configuration file, called **Vagrantfile**, where all configurations of a VM are mentioned. And Vagrant creates the Virtual Machine with the same configuration mentioned in the file. Even if there is a need to install some software in the VM, one can specify the same in Vagrantfile, and Vagrant downloads and installs the same for us.

Let's look into the steps for provisioning one VM using Vagrant.

## Installation

**Step 1:** [Download](#) Vagrant based on your operating system and install it in the system.

**Step 2:** Verify vagrant installation using command ***vagrant -v*** in command prompt. It will show the version of our vagrant installed as below.



```
Command Prompt
C:\>vagrant -v
Vagrant 2.2.16
C:\>
```

**Step 3:** [Download](#) Virtual Box based on your operating system and install it in the system.

## Vagrant Project Setup

**Step 1:** Create a folder where we want to save all the Vagrant-related files.

**Step 2:** Create a file, named **Vagrantfile** for mentioning the configuration of the VM. Since this is the first time Vagrant is being used after the installation, so it is advisable to let Vagrant create the file for us with minimum configuration. And modification can be done later.

**Step 3:** Open PowerShell (for Windows) or Terminal (for Linux) and go to the location of the folder that was created in step 1.

**Step 4:** Run command ***vagrant init bento/ubuntu-16.04***, and let the execution complete.

This command will initialize the directory with the specified Vagrant Box (bento/ubuntu-16.04). We will find Vagrantfile created in this location. If opened, you will see that some sample configuration has already been mentioned with proper examples and all of them are commented. Anyone can check it out to build a Virtual Machine with a more specific configuration. A piece of important information mentioned in the file can be seen.

```
config.vm.box = "bento/ubuntu-16.04"
```

This is the Vagrant Box, that was mentioned during initialization.

### Boot up Virtual Machine Using Vagrant:

```
vagrant up
```

This command will take Vagrantfile and provision one VM with all the configurations mentioned. For now, it boots up a Virtual machine with a Ubuntu-16.04 version of the Operating System.

### SSH into Virtual Machine

**Step 1:** Now, the VM has been created. So, to get into it, run command ***vagrant ssh***. After executing this command you are now inside the newly created VM.

Now, anything can be done inside the VM through this terminal. The following screen will be visible-

```
PS E:\vagrant-project> vagrant ssh
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-209-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento
vagrant@vagrant: $
```

**Step 2:** To log out from Virtual Machine, run the command **logout**. Again the terminal will come back to the host machine.

```
vagrant@vagrant: $ logout
Connection to 127.0.0.1 closed.
PS E:\vagrant-project>
```

### Clean Up Vagrant:

**Step 1:** To shut down the VM using Vagrant, run command **vagrant halt**. This command will switch off the VM and again to power on the VM, run **vagrant up** command.

```
PS E:\vagrant-project> vagrant halt
=> default: Attempting graceful shutdown of W...
PS E:\vagrant-project>
```

**Step 2:** In order to shut down a Virtual Machine, keeping its current state, run ***vagrant suspend***. In this case, when the VM is started again using ***vagrant up***, the system will start from the same position where it left off. All unsaved work can be restored.

But in this case, VM will not release the resources from the Host Machine, even it takes some more disk space to store the current state of its own RAM inside the Host Machine.

```
PS E:\vagrant-project> vagrant suspend
=> default: Saving VM state and suspending execution...
PS E:\vagrant-project>
```



**Step 3:** To delete the VM with all its resources, run **vagrant destroy**. This command will shut down the VM and delete it from the Host System.

```
PS E:\vagrant-project> vagrant destroy
default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Forcing shutdown of VM...
==> default: Destroying VM and associated drives...
PS E:\vagrant-project>
```

### Setting Up Sample Project In VM Using Vagrant:

Let's install a web server inside VM and access the same from your host computer. Follow the steps below-

**Step 1:** Do **vagrant ssh**, to go into the VM created.

**Step 2:** Inside the VM, install a web server, say Nginx, manually. Run below commands-

```
sudo apt update
sudo apt install nginx
```

**Step 3:** Check whether the Nginx service is running or not using the command-

```
systemctl status nginx
```

```
vagrant@vagrant: $ systemctl status nginx
nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-05-16 05:34:23 UTC; 1min 3s ago
     Main PID: 12767 (nginx)
       CGroup: /system.slice/nginx.service
               └─12767 nginx: master process /usr/sbin/nginx -g daemon on; master_process on
                 └─12768 nginx: worker process
                   └─12769 nginx: worker process
```

If the service is not running, start the service using the command-

```
sudo service nginx start
```

**Note-** The above installation can also be done with the Vagrantfile, but for now, let's use **vagrant ssh** functionality.

**Step 4:** Go to **/var/www/html** and create one sample HTML file, say **index1.html** as below-

## HTML

```
<!DOCTYPE html>
<html>
  <body>
    <h2 style = "color: green"> Welcome to GeeksforGeeks </h2>
  <body>
</html>
```

**Step 5:** Now Nginx is running in its default port 80 inside the VM. But to access the VM's Nginx server from the Host Machine, it is required to map port 80 of VM with some port in Host Computer. And this mapping is called **Port Forwarding**.

Port forwarding can be done manually using Virtual Box Manager. But since Vagrant is being used here, let's do it using Vagrantfile.

**Step 6:** Open Vagrantfile and add the line

```
config.vm.network "forwarded_port", guest: 80, host: 85
```

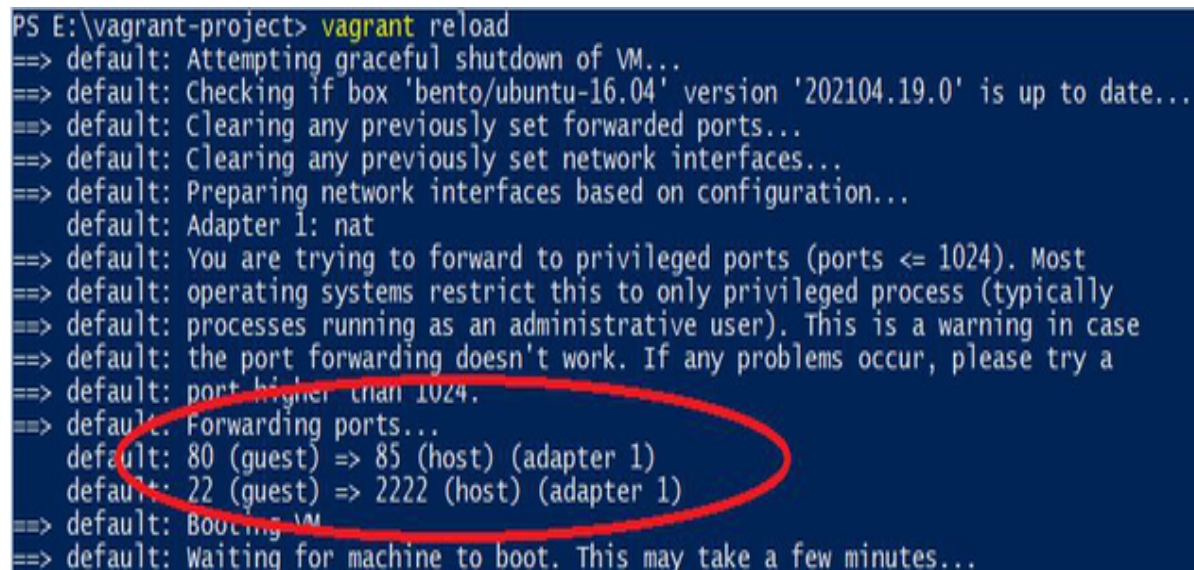
after the line

```
config.vm.box = "bento/ubuntu-16.04"
```

and save the file.

Here basically the 80 port of VM is mapped with 85 port of the host system.

**Step 7:** Exit from the VM in PowerShell or terminal and run the command **vagrant reload** to reload the new setting that we added in Vagrantfile. During reload, one can see the forwarded port as below. In the below snapshot, the first forwarded port is the one that is recently added and the second one is the default for SSH connectivity.



```
PS E:\vagrant-project> vagrant reload
==> default: Attempting graceful shutdown of VM...
==> default: Checking if box 'bento/ubuntu-16.04' version '202104.19.0' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: You are trying to forward to privileged ports (ports <= 1024). Most
==> default: operating systems restrict this to only privileged process (typically
==> default: processes running as an administrative user). This is a warning in case
==> default: the port forwarding doesn't work. If any problems occur, please try a
==> default: port higher than 1024.
==> default: Forwarding ports...
    default: 80 (guest) => 85 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM
==> default: Waiting for machine to boot. This may take a few minutes...
```

**Step 8:** Check whether Nginx is running or not. Now the Nginx server of the VM can be accessed using 85 port from the host system. The HTML file created above can be retrieved, using the URL- **localhost:85/index1.html**



### **Advantages Of Vagrant:**

- Vagrant is free and open-source.
- It offers extremely efficient project scaffolding of a dev/ test environment.
- Vagrant has a great variety of community boxes and plugins.

### **Disadvantages Of Vagrant:**

- Syntax updates have major repercussions for the plugin compatibility
- Community support is good, but it is a lengthy process.
- Major updates have introduced some serious bugs with consequences in the dev projects.

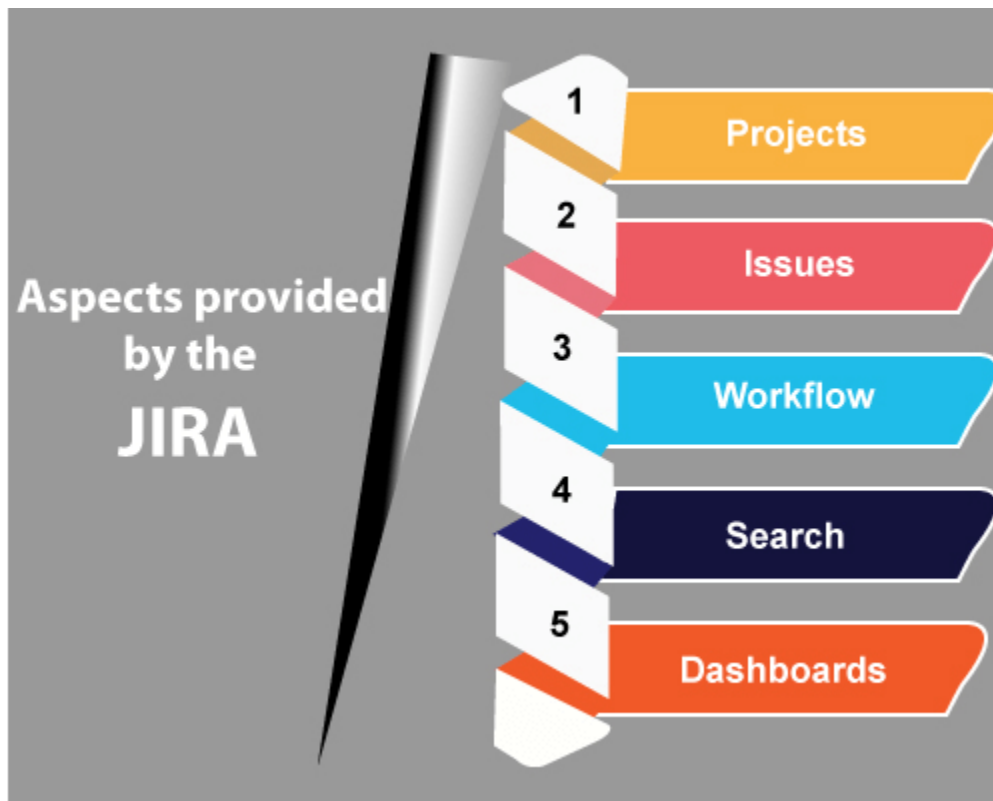
# JIRA

## What is JIRA?

JIRA is a software testing tool developed by the Australian Company Atlassian. It is a bug tracking tool that reports all the issues related to your software or mobile apps. The word JIRA comes from the Japanese word, i.e., "Gojira" which means Godzilla.

JIRA is based on the Agile methodology and the current version of the Jira is 6.

**The following are the useful aspects provided by the Jira:**



- **Projects:** It is used to manage the defects very effectively.
- **Issue:** It is used to track and manage the defects/issues.
- **Workflow:** Processes the Issue/Defect life cycle. Suppose we have a business requirement, we create the technical design and from the technical design, we create the test cases. After creating the test cases, coding is done, and then

testing is performed on the project. This design workflow is possible by using Jira.

- **Search:** Find with ease. Suppose we have done with a project at the beginning of the December and its version is 1.0. Now, we move to version 1.1 and completed at the end of December. What we are doing is that we are adding new versions. Through Jira, we can get to know that what happened in the earlier versions, how many defects occurred in the earlier projects and the learning we achieve from the earlier projects.
- **Dashboards:** Dashboard is a display which you see when you log in to the Jira. You can create multiple dashboards for multiple projects. You can create the personal dashboard and can add the gadgets in a dashboard so that you can keep track of the assignments and issues that you are working on.

## Why JIRA

**JIRA tool is used because of the following reasons:**

- **Plan, Track and Work Faster**

JIRA is a bug-tracking tool mainly used to track, organize, and prioritize the bugs, newly added features, improvements for certain software releases. Projects are subdivided into issues and issues can be of multiple types such as bug, new feature, improvement, and documentation tasks.

When the release date of software comes near, then software developers need to focus on the remaining issues which are to be fixed before the specified date. It also becomes difficult for the QA to maintain the status of the documentation, i.e., sometimes it becomes hard to keep track of everything.

JIRA is a good choice for handling the above issues. It enables software developers to track issues and improvements. It manages the projects as well as maintain the technical documentation.

- **The main source of information**

JIRA is the primary source of information for the next software release. On JIRA, the whole team of the software developers can plan for the new features which are to be added and bugs to be fixed in the next release.

It also helps the QA team in writing the technical documentation. Through JIRA, the QA team can check the status of each feature that is newly added by the software developers, and according to that, they can plan how to document for the new version.

- **Organize the documentation tasks**

JIRA tool is used to organize the documentation tasks. It is useful in grouping the multiple tasks by using the component functionality, and even you can create your own documentation. In this way, you can create a structured way of documentation.

- **Track the progress of our documentation**

It is a very useful tool in tracking the progress of our documentation. JIRA tool provides a very important feature, i.e., pie chart macro. In the pie chart macro, you can view tasks such as Open tasks, Closed tasks, Resolved tasks.

- **Helps to meet the deadlines of a documentation release.**

You can define the specific due date or deadline for the release of documentation, and even you can configure the JIRA tool with the notifications so that you can finish your documentation in time.

- **Measures the time spent on documentation**

JIRA tool does not have the default functionality for measuring the time spent on documentation. JIRA tool is bundled with the Tempo Timesheets, which measures how much time has been spent on the documentation.

- **Provides feedback faster**

JIRA tool provides the Confluence pages where you can connect to the issues in

just a few clicks. If something needs to be updated, then you can create the issues directly from the Confluence page.

## Bamboo

Bamboo is a **continuous integration (CI) server** that can be used to automate the release management for a software application, creating a continuous delivery pipeline.

What is the difference between Jenkins and Bamboo?

Jenkins is an open-source tool, while **Bamboo is a commercial tool**. Jenkins is a project supported by its global community, and Bamboo has its own dedicated team for its development. Bamboo has a more user-friendly approach than Jenkins – as usually, open-source apps are more concerned with other features.

## Nagios

What is Nagios and how it works?

Nagios is an **open source monitoring system for computer systems**. ... Nagios software runs periodic checks on critical parameters of application, network and server resources. For example, Nagios can monitor memory usage, disk usage, microprocessor load, the number of currently running processes and log files.