
Community Experience Distilled

# DevOps for Web Development

## Achieve continuous development and deployment of your web applications with ease

Mitesh Soni

[PACKT] PUBLISHING open source* community experience distilled

# Table of Contents

# 1
# Getting Started - DevOps Concepts, Tools, and Technology

*The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency – Bill Gates*

DevOps is not a tool or technology; it is an approach or culture that makes things better. This chapter describes in detail on how DevOps solves different problems of traditional application delivery cycle. It also describes how it can be used to make Development and Operations teams efficient and effective to make time to market faster by improving culture. It also explains key concepts essential for evolving DevOps culture.

Readers will learn about DevOps culture, its lifecycle and its key concepts, tools, technologies and platforms used for automating different aspects of application lifecycle management.

In this chapter, we will cover the following topics:

- Understanding DevOps movement
- DevOps Lifecycle-All about “Continuous”
- Continuous Integration
- Configuration Management
- Continuous Delivery / Continuous Deployment
- Continuous Monitoring
- Continuous Feedback

- Tools and Technologies
- Overview of Sample JEE Application

# Understanding DevOps movement

Let's try to understand what DevOps is. Is it a real technical word? Answer is No and the reason for this is DevOps is not about only technical stuff. It is also not a technology nor an innovation. In simple terms, DevOps is a blend of complex terminologies. It can be considered as a concept, culture, development and operational philosophy or a movement.

To understand DevOps, let's revisit the old days of any IT organization. Consider there are multiple environments where application is deployed. Following sequence of events takes place when any new feature is implemented or bug is fixed:

1. The development team writes a code to implement a new feature or fixes a bug. New code is deployed in to development environment and generally tested by the development team.
2. New code is deployed in the QA environment where it is verified by the testing team.
3. New code is provided to the operations team for deploying it into production environment.
4. Operation team is responsible for managing and maintaining the code

Let's list out the possible issues in the above mentioned approach:

- Transition of current application build from development environment to production environment lasts over weeks or months
- Priorities of Development Team, QA Team and IT Operations Team are different in an organization and effective and efficient co-ordination becomes necessity for smooth operations
- Development team is focused on latest development release while Ops team cares about Stability of an Production environment
- Development and Operations team are not aware about each other's work and work culture
- Both teams work in different type of environments; there is a possibility where development team has resource constraints and hence they manage different kind of configuration. It may work on localhost or in Dev environment.
- Operations team work on production resources and thus there will be a

huge gap in configuration and deployment environment. It may not work where it needs to run – in production environment.

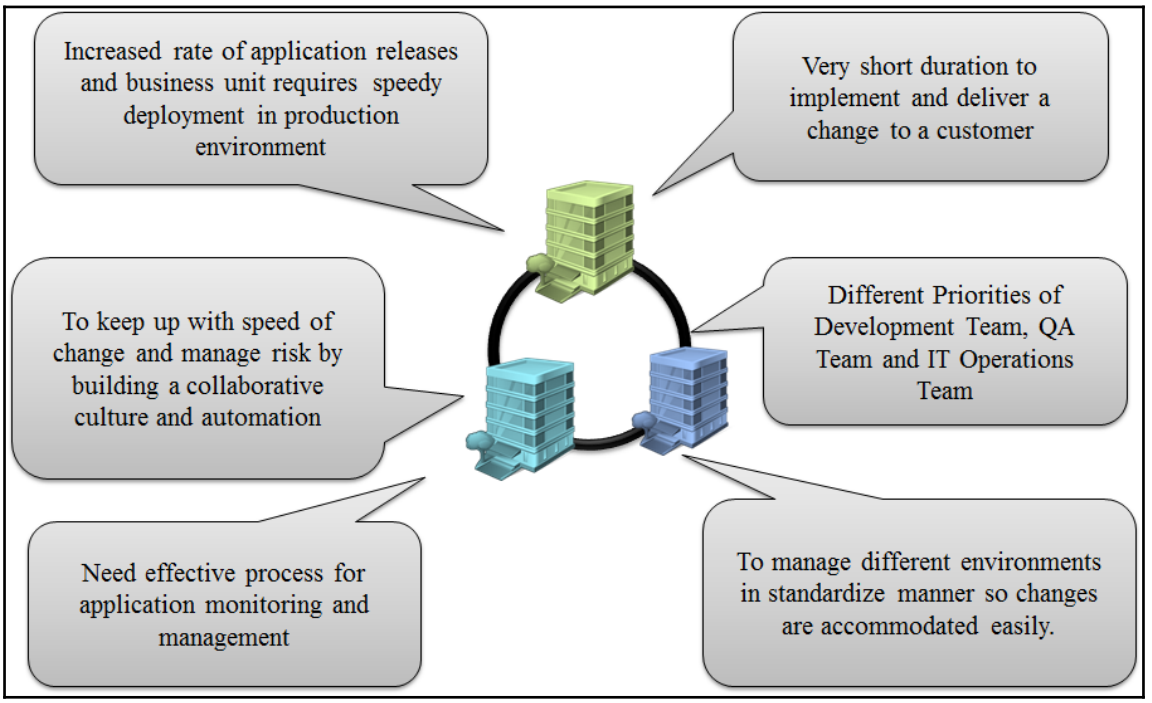
- Assumptions are key in such scenario and it is not possible that both team will work under same set of assumptions
- There is a manual work involved in the setting up runtime environment, configuration, and deployment activities. The biggest issue with manual application deployment process is its non-repeatability and manual process is error-prone.
- The development team has the installable files, configuration files, database scripts, and deployment documentation. They provide it to operations team. All these artifacts are verified in development environment and not in production or staging.
- Each team may take different approach for setting up runtime environment, configuration, and deployment activities considering resource constraints and resource availability.
- In addition, deployment process needs to be documented for future usage. Now, maintaining the documentation is a time-consuming task that requires collaboration between different stakeholders.
- Both teams work separately and hence there can be situation where both use different automation techniques
- Both teams are not aware about challenges faced by each other and hence they may not be able to visualize or understand an ideal scenario where application works
- While operations team is busy in deployment activities, development team may get another request for feature implementation or bug fix; in such case, if operations team faces any issues in deployment then they may try to consult development team who is already occupied in new implementation. It results in communication gaps and required collaboration may not happen.
- There is hardly any collaboration between the development team and the operations team. The poor collaboration that causes many issues in the application deployment to different environments that results into back and forth communication via mail, chat, calls, meetings, and so on and it often ends up with quick fixes.
- Challenges for Developers Team
    - Competitive Market creates pressure of on time delivery
    - Production ready Code Management and New feature Implementation
    - Release cycle is often long and hence development team

has to make assumptions before the application deployment finally takes place. In such scenario it takes more time to fix the issues occurred while deployment in staging or production environment.

- Challenges for Operations Team
  - Resource contention – Difficult to handle increasing demands of resources
  - Redesigning or tweaking is needed to run the application in Production environment
  - To diagnose and rectify the issues after application deployment in isolation

# DevOps with changing times

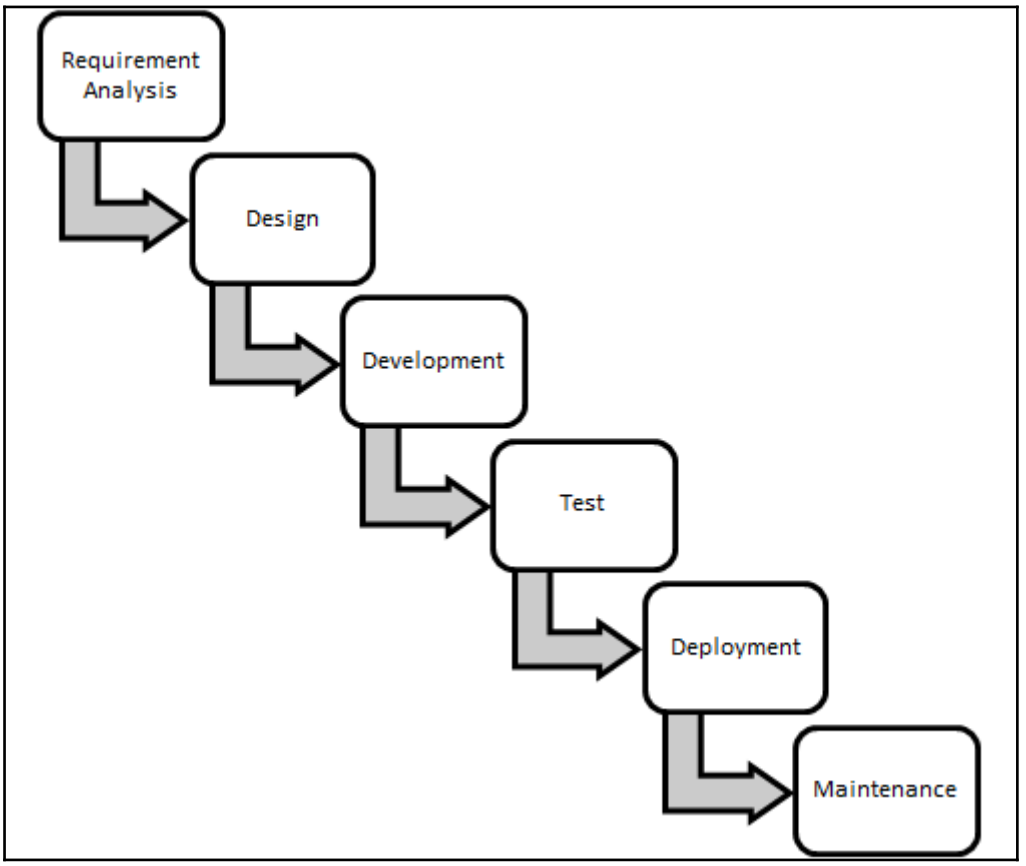
Time changes everything. In modern era, customers expect and demand extremely quick response, and we need to deliver new features continuously to stay in the business. Users and customers today have rapidly changing needs, they expect 24/7 connectivity and reliability, and access services over smart-phones, tablets and PCs. As software product vendors – irrespective of whether in development and / or operations – organizations need to push updates frequently to satisfy customers' needs to stay relevant. In short, organizations are facing following challenges:




Change in the behavior of customer or market demand affected the way development process takes place.

## The waterfall model

Since long, Waterfall Model is used for software development.




It has its own advantages as follows:

- Easy to understand
- Easy to manage – Input and Output of each phase is defined
- Sequential process – Order is maintained
- Better control

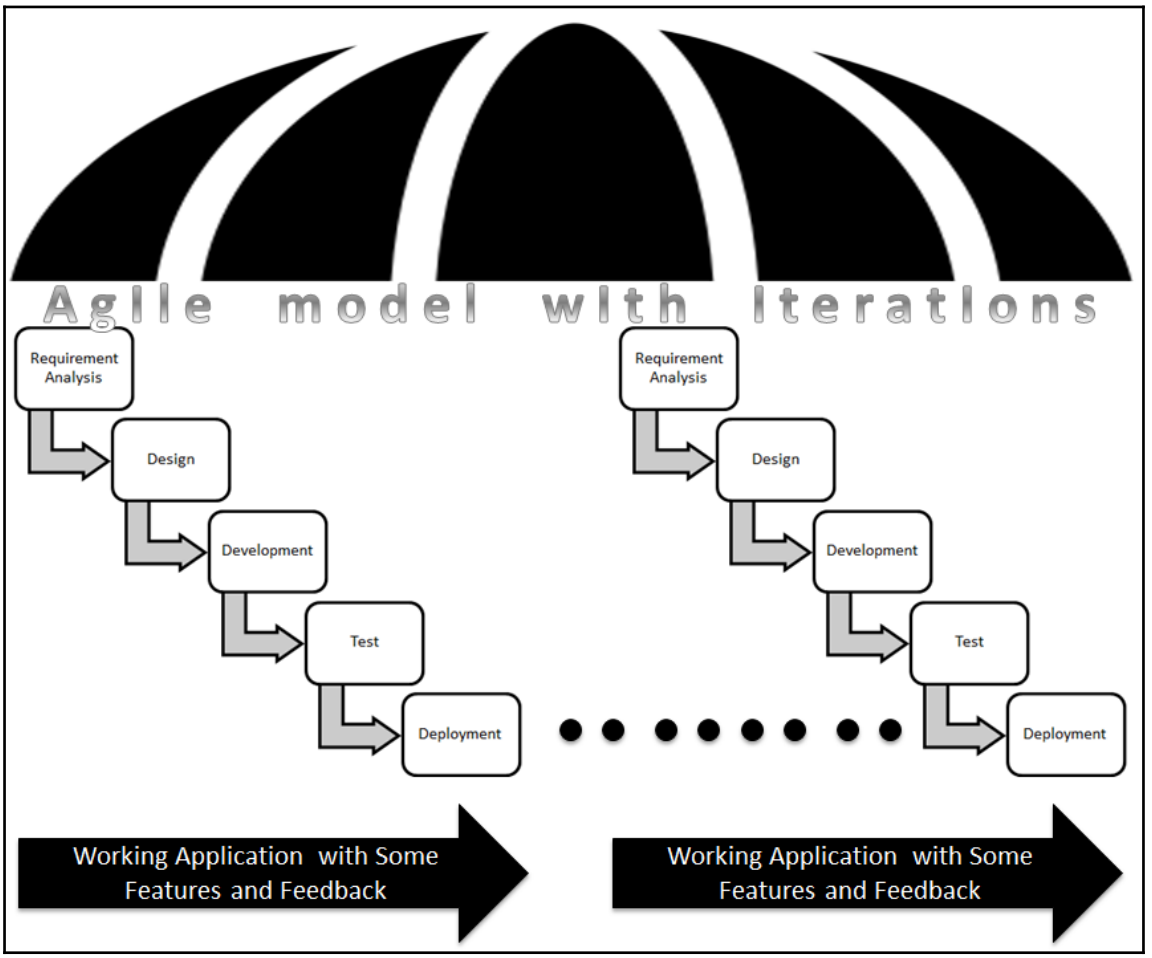
However, it was useful in the scenarios where requirements are predefined and fixed. As it is a rigid model with sequential process, we can't go back to any phase and change things. It has its own share of disadvantages as follows:

- No Revision
- No outcome or application package until all phases are completed

- Not possible to integrate feedback until all phases are completed
- Not suitable for changing requirements
- Not suitable for long term and complex projects

## Agile Model

In Waterfall model, inefficient estimation, long time to market, and other issues led to a change. It is known as Agile Model. Agile development or agile methodology is a method of building an application by empowering individuals and encouraging interactions, by giving importance to working software, by customer collaboration – using feedback for improvement in next steps, and responding change in efficient manner. It emphasizes on customer satisfaction through continuous delivery in small interactions for specific features in short timelines or sprints.




One of the most attractive benefits of agile development is Continuous Delivery in short time frames or in agile terms – Sprints. Now, it is not a one-time deployment but it is the case of multiple deployments. Why? After each sprint, application with some feature is ready for showcase. It needs to be deployed in the specific environments for demo and thus deployment is no longer a one-time activity.

It is extremely essential from organization's perspective to meet changing demands of customers. To make it more efficient, communication and collaboration between all cross functional teams is essential. Many organizations have adopted agile methodology.

In such case, traditional manual deployment processes work as a speed breakers for

incremental deployments. Hence, it is necessary to change other processes also along with the change in application development methodology. One key can't be used for all the locks; similarly waterfall is not suitable in all projects. We need to understand that Agile is customer focused and feedback is vital. Based on customer feedback, changes happen and release cycles may increase. Just imagine a scenario where input is high but input processing is slow. Consider an example of a shoe company where one department prepares shoes and another department is working on final touches and packaging. What will happen if packaging process is slow and inefficient? It will be a shoe pile up in the packaging department. Now let's add a twist in this situation. What if shoe making department brings new machines and improve process of making shoe. It makes shoe making process 2 to 3 times faster. Now imagine a situation of packaging department. Similarly, Cloud computing and DevOps has gained momentum that increases speed of delivery and improve quality of end product. Thus, agile approach of application development, improvement in technology, and disruptive innovations and approaches has created a gap between development and operations team.

# Collaboration

DevOps attempts to fill the gaps by developing a partnership between Development and Operations team. DevOps movement emphasizes communication, collaboration and integration between software developers and IT operations. DevOps promotes collaboration and collaboration is facilitated by automation and orchestration to improve processes. In other words, DevOps essentially extends the continuous development goals of the agile movement to continuous integration and release. DevOps is a combination of agile practices, processes leveraging the benefits of cloud solutions. Agile development and testing methodology help us to meet the goals of continuous integrate, develop, build, deploy, test, and release application. It provides mechanism for constant feedback from different teams and stakeholders. It also provides transparency, platform for collaboration across teams such as business analysts, developers and testers. In short, Agile and DevOps are compatible and increases value of each other.

One of the most popular saying is practice makes a man perfect. What if that saying is applied in production like environment? It will be much easier to repeat the entire process as there is no last minute surprises and most of issues in the deployment are already experienced and dealt with. The development team supports operational requirements such as deploy scripts, diagnostics, and load and performance testing from the beginning of the application delivery life cycle; and the operations team provides knowledgeable support and feedback before, during, and after deployment. The remedy is to integrate the testing, deployment, and release activities into the development process. By performing all activities multiple times and ongoing part of development so that by the time you are ready to release your system into production there is little to no risk, because deployment process

is already rehearsed it on many different environments in a progressively more production-like environments.

## Cloud Computing: The Disruptive Innovation

One of the major challenges is to manage infrastructure for all environments. Virtualization and Cloud environment can help to get started with this. Cloud helps us to overcome this hurdle by providing flexible on demand resources and environments. It provides distributed access across the globe and helps in effective utilization of resources. Cloud provides repository of software, tools which can be used on-demand basis. We can clone environments, reproduce required versions as and when required. The entire development, test, and production environments can be monitored and managed using the facilities provided by the cloud providers. With the advent of Cloud computing, it is easy to re-create every piece of infrastructure used by application with the use of automation. That means operating systems, OS configuration, runtime environments, its configuration, infrastructure configuration, and so forth can all be managed. In this way, it is easy to recreate production environment exactly in an automated fashion. Thus DevOps on Cloud brings in the best of breed from both agile development as well as cloud solutions. It helps in providing Distributed Agile in Cloud, leading to Continuous Accelerated Delivery.

## Why DevOps?

DevOps is effective because of new methodology, automation tools, agile resources by cloud service providers, and other disruptive innovations, practices, and technologies. However, it is not only about tools and technology. DevOps is more about culture than tools or technology alone.

> *Technology is just a tool. In terms of getting the kids working together and motivating them, the teacher is the most important-Bill Gates*

There is an urgent need of huge change the way development and operations team collaborates and communicates. Organizations need to have chnage in culture and have long term business goals that include DevOps in vision. It is important to establish pain points and obstacles experienced by different teams or business units and use that knowledge for refining business strategy and fix goals.

> *People always fear change. People feared electricity when it was invented, didn't they? People feared coal; they feared gas-powered engines… There will always be ignorance, and ignorance leads to fear. But with time, people will come to accept their silicon masters-Bill Gates*

If we identify common issues faced by different section of organization and change strategy to bring more value then it makes sense. It can be a stepping stone in the direction of DevOps. With same old values and objectives, it is difficult to adopt any new path. It is very important to align people with new process first. For example, team has to understand value of agile methodology else they will resist using it. They might resist it because they are comfortable with old process. Hence, it is important to make them realize the benefit and empower them also to bring the change.

> *Change is hard because people over estimate the value of what they have-and under estimate the value of what they may gain by giving that up-James Belasco and Ralph Stayer*

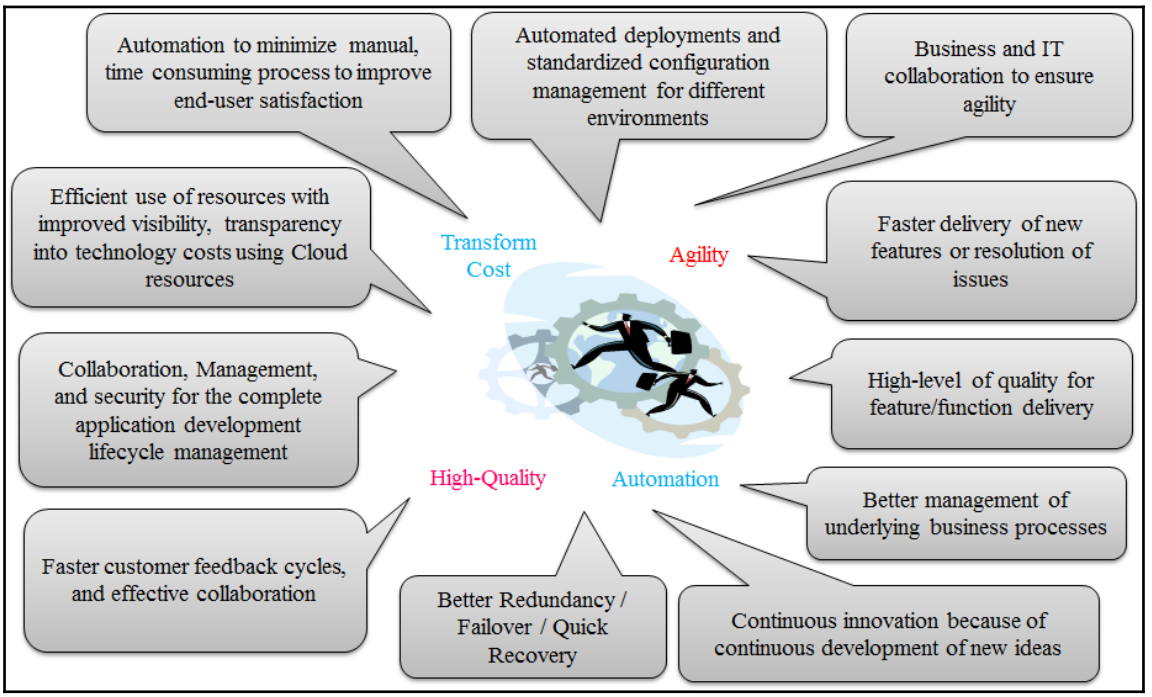
Self-dependent teams bring best out of them when they are empowered. We also need to understand that power comes with accountability and responsibility. Cross functional teams work together and enhance the quality by giving their expertise in the development process; however it is not isolated function. Communication and collaboration across teams makes quality way higher.

The end objective of DevOps culture is Continuous Improvement. We learn from mistakes and it becomes experience. Experience helps us to identify robust design patterns and minimize errors in the processes. This leads to enhancement of productivity and hence we achieve new heights with continuous innovations.

> *Software innovation, like almost every other kind of innovation, requires the ability to collaborate and share ideas with other people, and to sit down and talk with customers and get their feedback and understand their needs-Bill Gates*

# Benefits of DevOps

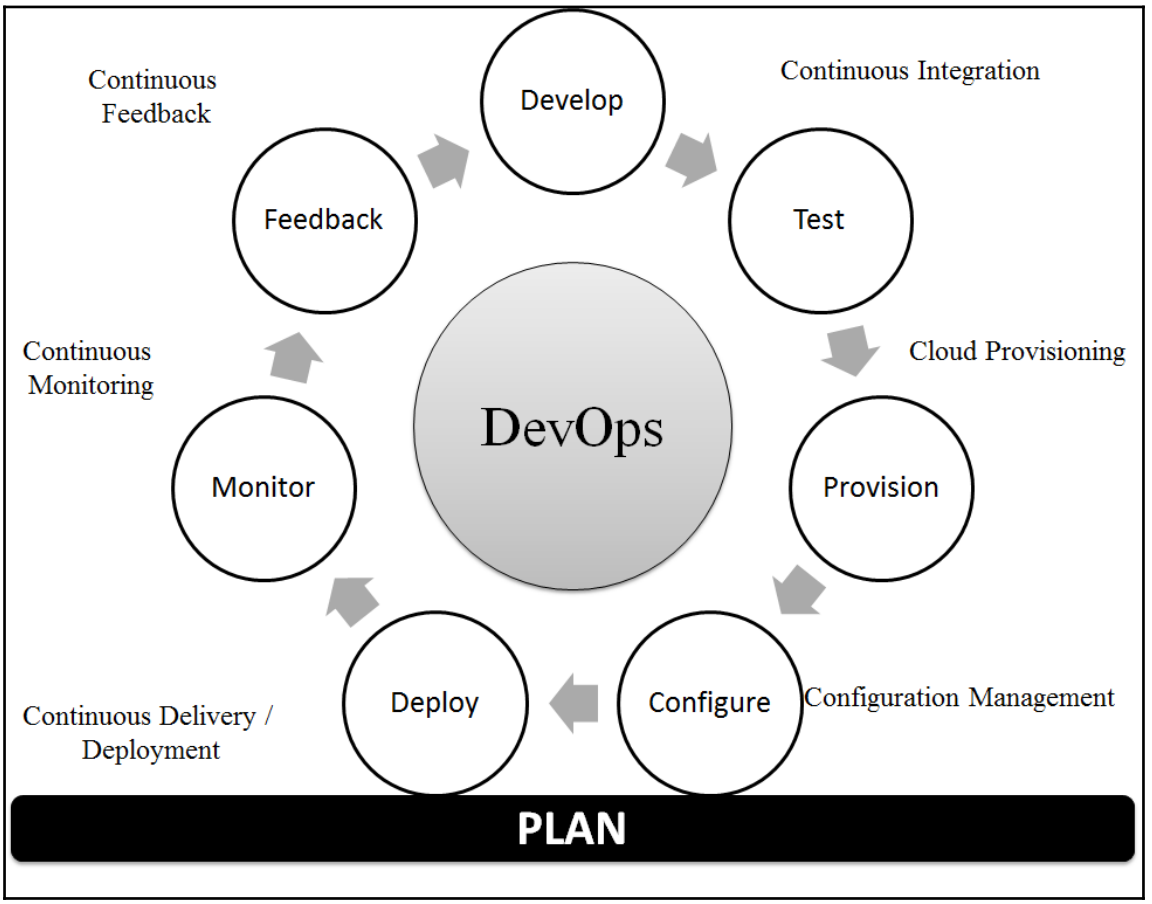
We will be covering all the benefits of DevOps in the following image:




Collaboration across different stakeholders brings many business and technical benefits that helps organizations to achieve their business goals.

# DevOps lifecycle – all about "Continuous"

Continuous integration (CI), Continuous Testing (CT), and continuous delivery (CD) are significant part of DevOps culture. CI includes automation of build, unit test and package process while CD includes application delivery pipeline across different environments. CI and CD accelerates the application development process through automation across different phases such as build, test, code analysis and so on; and enables users to achieve end to end automation for application delivery lifecycle.

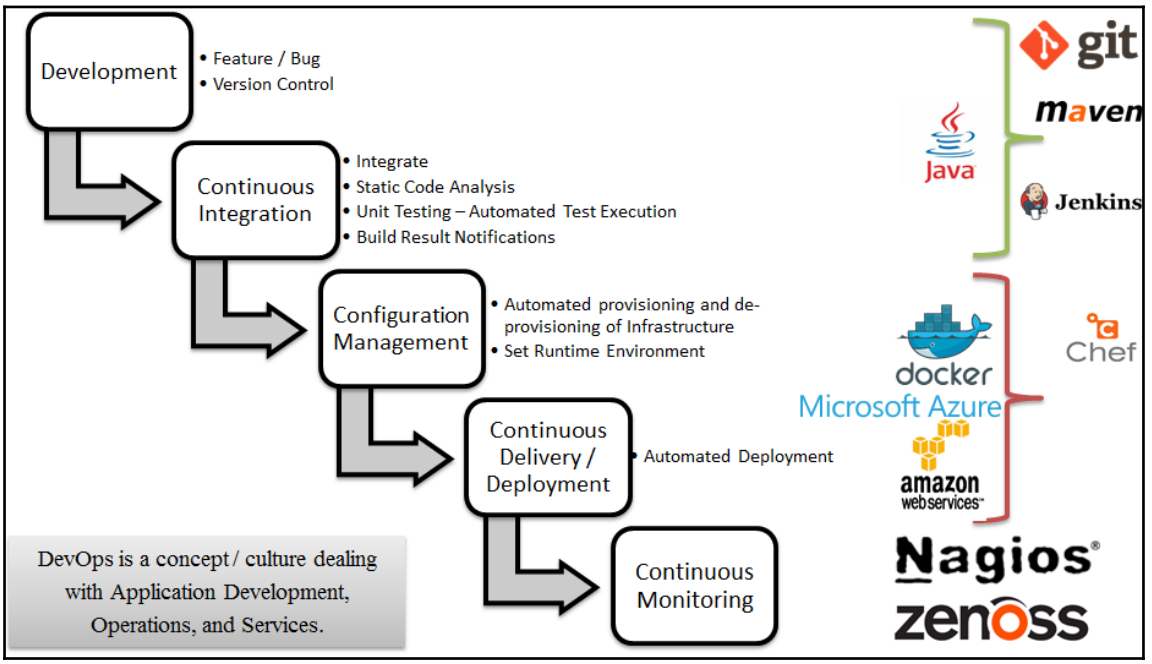



Continuous Integration and Continuous Delivery or Deployment is well supported by Cloud provisioning and Configuration Management. Continuous Monitoring helps to identify issues or bottlenecks in the end to end pipeline and helps to make pipeline effective.

Continuous Feedback is integral part of this pipeline which directs the stakeholders whether are near to the required outcome or going in the different direction.

> *Continuous effort — not strength or intelligence — is the key to unlocking our potential- Winston Churchill*

Following diagram shows mapping of different parts of Application delivery pipeline with toolset for Java Web application.




We will use Sample spring application throughout this book for demonstration purpose and hence toolset is related to Java technology.

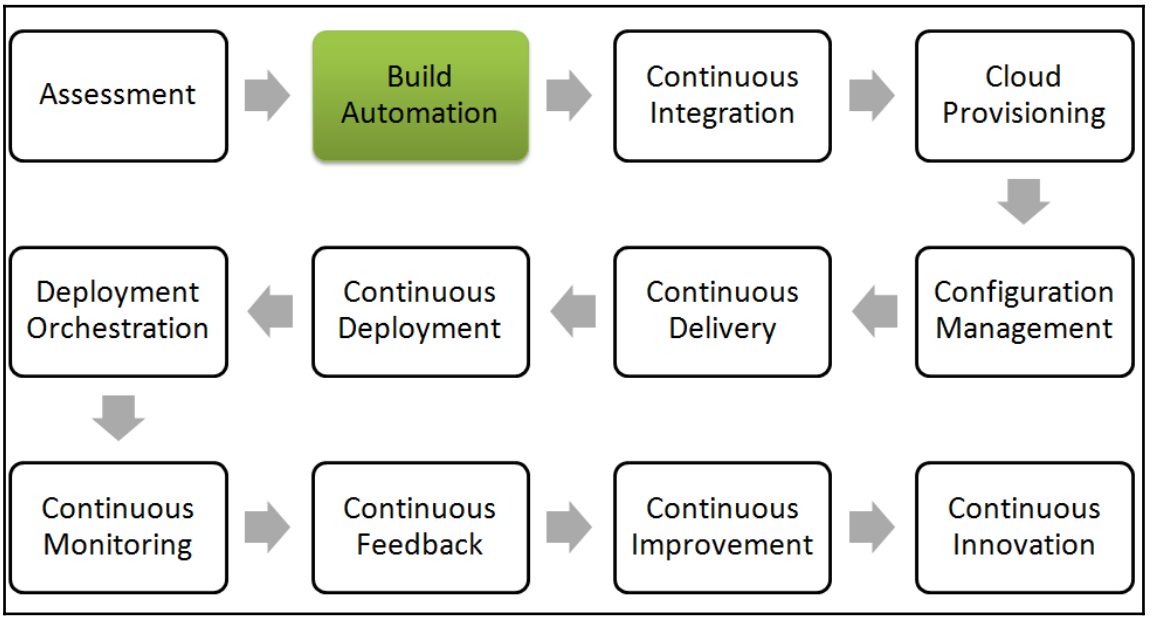
# Build automation

Automated build helps to create application build using build automation tools such as Apache Ant, Apache Maven, and so on. Automated build process include following activities:

- Compile Source Code into Class files or Binary Files
- To provide reference to the third party library files
- To provide path of configuration files
- Packaging Class files or Binary Files into WAR files in case of Java
- To execute automated test cases
- To deploy WAR file into local or remote machine
- To reduce manual effort in creating WAR file

Apache Maven and Apache Ant automate build process and it makes build process simple,

repeatable, less error prone as it is a Create once Run Multiple times concept. Build automation is base of any automation in Application Delivery Pipeline.

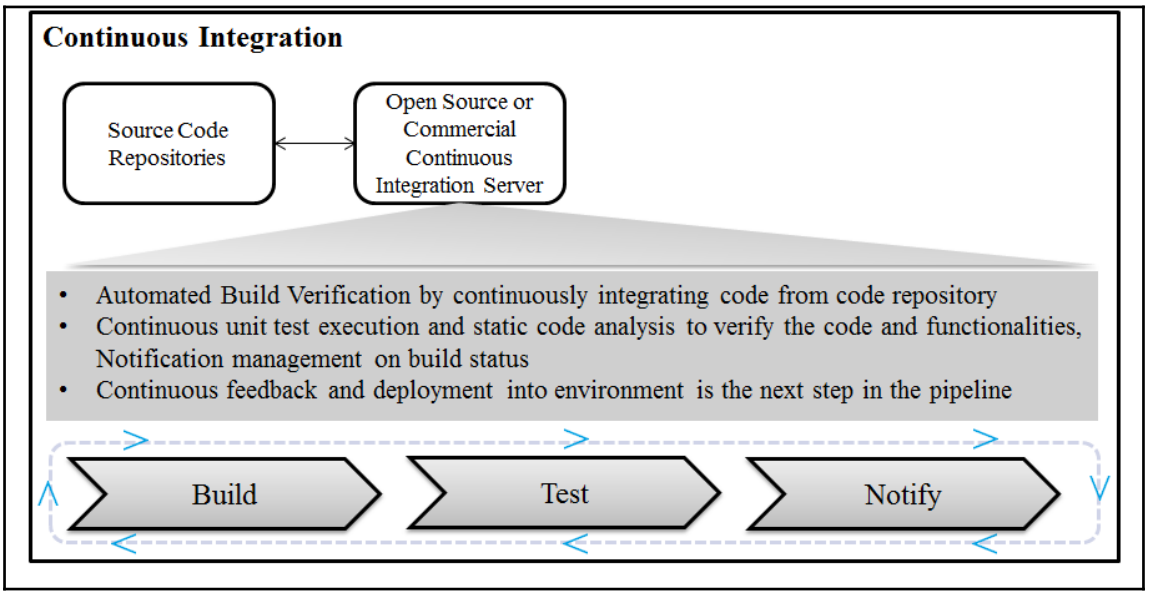



Build automation is essential for Continuous Integration and rest of the automation is effective only if build process is automated. All CI servers such as Jenkins, Atlassian Bamboo and so on are using build files for continuous integration and creating application delivery pipeline.

# Continuous integration

What is Continuous Integration? In simple words, Continuous Integration (CI) is a software engineering practice where each check-in by a developer is verified by

- Pull mechanism: executing automated build at a scheduled time or
- Push mechanism: executing automated build when changes are saved in repository and
- Executing unit test against latest changes available in source code repository.




The main benefit of continuous integration is quick feedback based on the result of build execution. If it is successful then all is well else fix the responsibility on the developer whose commit has broken the build, notify all stakeholders and fix the issue.

Continuous Integration
http://martinfowler.com/articles/continuousIntegration.html

Why CI is needed or in other words, what is the requirement of it? Answer is, it makes things simple and identify bugs or errors in the code at very early stage of development and it is relatively easy to fix them. Just imagine if same scenario takes place after a long duration and there are too many dependencies and complexities we need to manage. In early stages it is far easier to cure and fix issues; consider health issues as an example and things will be more clear in that context.

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

CI is a significant part and in fact a base of release management strategy of any organization that wants to develop DevOps culture.

Following are instant benefits of CI:

- Automated integration with Pull or Push mechanism
- Repeatable Process without any manual intervention
- Automated test case execution
- Coding standard verification
- Execution of scripts based on the requirement
- Quick feedback – Build status notification to stakeholders via mail
- Teams are focused on their work and not in managing processes

Jenkins, Apache Continuum, Buildbot, GitLabCI, and so on are some of the examples of open source CI Tools. AnthillPro, Atlassian Bamboo, TeamCity, Team Foundation Server, and so on are some of the examples of commercial CI Tools.

## Best practices

We will now be looking at the best practices that can be useful while considering Continuous Integration implementation:

- Maintain a code repository such as Git or SVN
- Check-in third-party jars, build scripts, other artifacts and so on into Code repository
- It is advisable to execute builds fully from Code repository – Use clean build
- Automate the build using Maven or Ant for Java
- Make the build self-testing: Create unit tests
- Commit all changes at least once a day per feature
- Every commit should be built to verify the integrity of changes
- Authenticate users and enforce access control (Authentication and Authorization)
- Use alphanumeric characters for build names and avoid symbols
- Keep different build jobs to maintain granularity and managing operations in a better way. Single job for all task is difficult when we try to troubleshoot. It also helps to assign build execution to slave instances if that concept is supported by CI server
- Backup Home directory of CI server regularly as it contains archived builds and other artifacts too which may be useful in troubleshooting
- Make sure CI server has enough free disk space available as it store lot of details related to builds
- Better not to schedule multiple jobs to start at the same time or use master slave

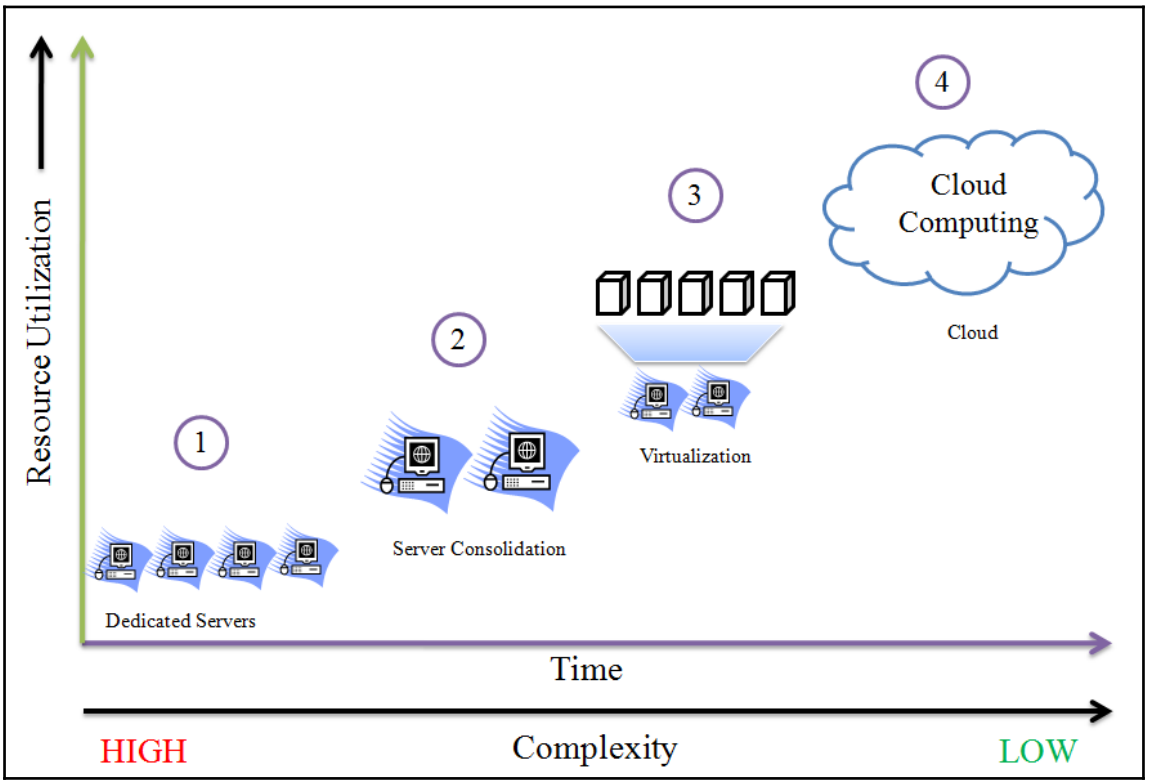
concept where specific jobs are assigned to slave instances so multiple build jobs can be executed at same time

- Set up Email, SMS or twitter notification to specific stakeholders of a project or an application. It is advisable to use customized mail to specific stakeholders
- It is advisable to use community plugins

# Cloud computing

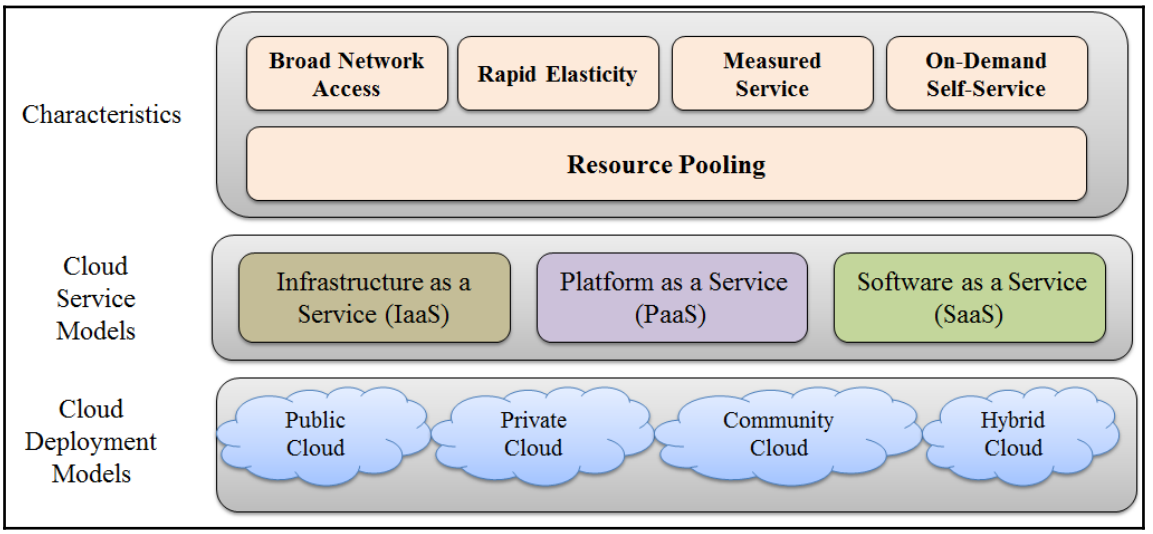
Cloud Computing is regarded as a ground breaking innovation in the recent years. It is reshaping the technology landscape. With breakthroughs made in appropriate service and business models, cloud computing has expanded its role as a backbone for IT services. Based on the experience, organizations improved from dedicated servers to consolidation, to virtualization to Cloud computing.

Cloud Computing provides elastic and unlimited resources which can be efficiently utilized in the time of peak load and normal load with pay per use pricing model. Pay as you go feature is a boon for development team which had faced resources scarcity since years. It is possible to automate provisioning resources and configuring resources based on requirements and that has reduced a lot of manual effort.




NIST SP 800-145, The NIST Definition of Cloud computing
http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

It has opened various opportunities in terms of availability of application deployment environments considering three service models and four deployment models.




There are four Cloud Deployment Models with that address specific requirements.

- Public Cloud: Cloud Infrastructure is available to general public
- Private Cloud: Cloud Infrastructure is operated for single organization
- Community Cloud: Cloud Infrastructure is shared by specific community that has shared concerns
- Hybrid Cloud: Cloud Infrastructure is composition of two or more cloud models

Cloud computing is pivotal components if we want to achieve our goals of automation to empower DevOps culture in any organization. Infrastructure can be considered as a code can be treated similar to code while creating resources, configuring them and managing resources with use of configuration management tools. Cloud resources play essential role in to successful adoption of DevOps culture. Elastic, scalable and pay as you go resource consumption allows organization to use same type of cloud resources in all different environments. The major problems in all the environments are inconsistency and limited capacity. Cloud computing solves this problem and that to with economic benefits.

# Configuration management

Configuration Management (CM) manages changes in the system or to be more specific, in the server runtime environment. Let's consider an example where we need to manage multiple servers with same kind of configuration. For an example, we need to install tomcat

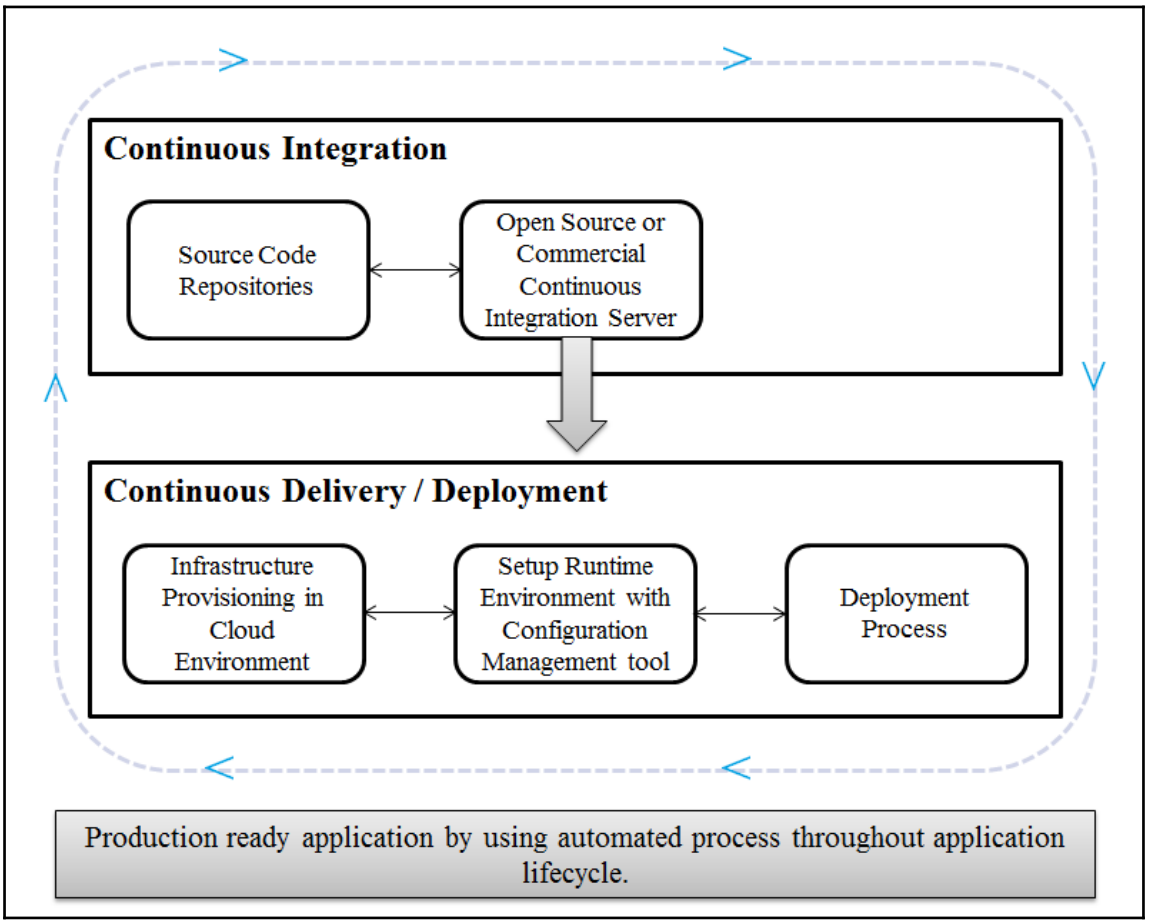
in each server. What if we need to change port in all servers or update some packages or provides rights to some users? Any kind of modifications in this scenario is a manual and error prone process if it is manual. As it is same configuration for all the servers, automation can be a key here. Automation of installation and modification in server runtime environment or permissions brings servers in desired state in effective manner.

CM is also about keeping track or versions of details related to state of specific nodes or servers. It is a far better situation when we need to change in many servers and we can push those changes to servers or all those server nodes can pull those changes and bring themselves into compliance of new policy. Centralized change can trigger this or nodes can communicate with CM server whether they need to update themselves or they are in a desired state already. CM tools makes process efficient when only changed behaviour is updated and not all installation and modification are applied again to server nodes.

There are many popular configuration management tools are available in the market such as Chef, Puppet, Ansible, Salt, and so on. Each tool are different in the way they work but characteristics and end goal is same – to bring standardized behaviour in state change of specific nodes without any errors.

# Continuous delivery / continuous deployment

Continuous Delivery and Continuous Deployment are used interchangbly more often then not. However, there is a small difference between them. Continuous Delivery is a process to deploy application in any environment in automated fashion and continuous feedback to improve the quality of an application. Continuous Deployment on other hand is all about deploying application with latest changes to the production environment. In other words we can say that Continuous Deployment implies Continuous Delivery while Continuous Delivery doesn't imply Continuous Deployment.




Continuous Delivery is significant because of the incremental releases after short span of implementation or sprint in agile terms. To deploy feature ready application from development to testing may include multiple iterations in a sprint due to change in requirements or change in interpretation. However, at the end of sprint, final feature ready application is deployed into the production environment. As we discussed about multiple deployments in testing environment even in short span of time, it is advisable to have automated approach for it. Scripts to create infrastructure and runtime environment for all environments are useful. It is easier to provision resources in such environment.

For example, to deploy an application in Microsoft Azure environment we need following resources:

- Azure web app configured with specific types of resources
- Storage account to store BACPAC file to create database
- To create SQL Server to host database
- To import BACPAC file from Storage account to create a new database
- Deploy a web application into Microsoft Azure environment

In above scenario, we may consider to use configuration file for each environment with respect to naming conventions and paths. However, we need similar types of resources in each environment. It is possible that configuration of resources change according to environment but that can be managed in configuration file for each environment. Automation scripts can use configuration files based on the environment and create resources and deploy an application into it. Hence repetative steps can be easily managed by automated approach and it is helpful in Continuous Delivery and Continuous Deployment both.

# Best practices for continuous delivery:

Following are some common practices we should follow to implement continuous delivery:

- Plan to automate everything in a application delivery pipeline: Consider a situation where a single commit only is required to deploy an application in the target environment. It should include compilation, unit test execution, code verification, notification, instance provisioning, setting up runtime environment, deployment of an application
    - Automate repetitive tasks
    - Automate difficult tasks
    - Automate manual tasks
        - Develop and Test newly implemented feature of bug fixing in a production like environment; it is possible now with pay per use resources provided by Cloud computing
            - Deploy frequently in Development and Test environment to gain experience and consistency

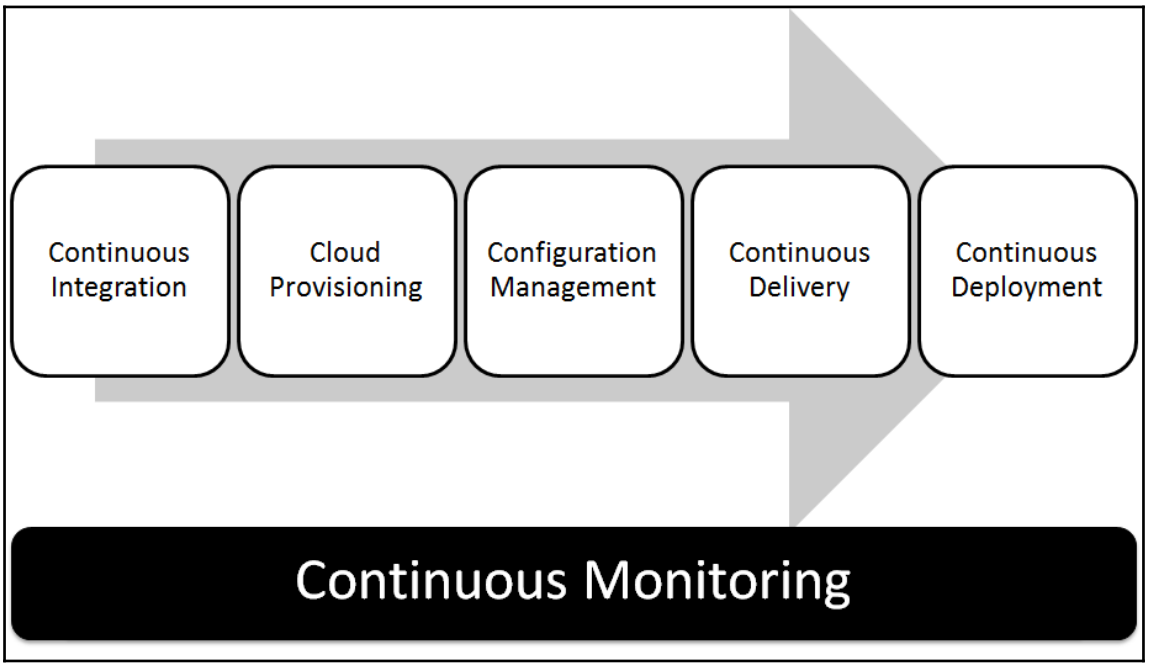
Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation
http://martinfowler.com/books/continuousDelivery.html

Continuous Delivery vs Continuous Deployment
http://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/

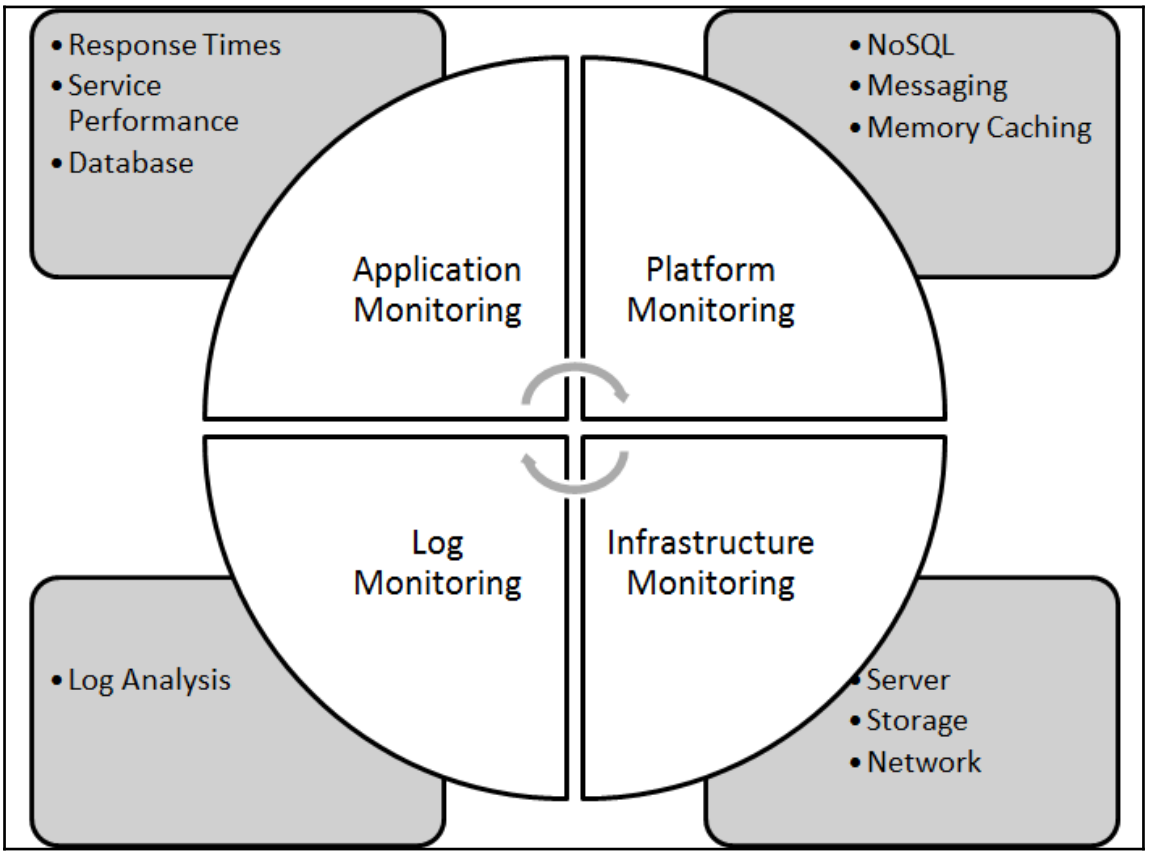
Continuous Delivery versus Continuous Deploy
http://devops.com/2015/10/30/continuous-delivery-versus-continuous-deploy/

# Continuous monitoring

Continuous Monitoring is a backbone of end to end delivery pipeline and open source monitoring tools are like toppings on an ice cream scoop. It is desirable to have monitoring at almost every stage to have transparency about all the process as shown in below image. It also helps in the troubleshooting within quick time. Monitoring should be a well thought out implementation of plan that starts in the beginning itself.




There is a likely scenario where end to end deployment is implemented in automated fashion but issues arise due to coding problems, query related problems, infrastructure related issues and so on. We can consider different types of monitoring as shown in the figure.

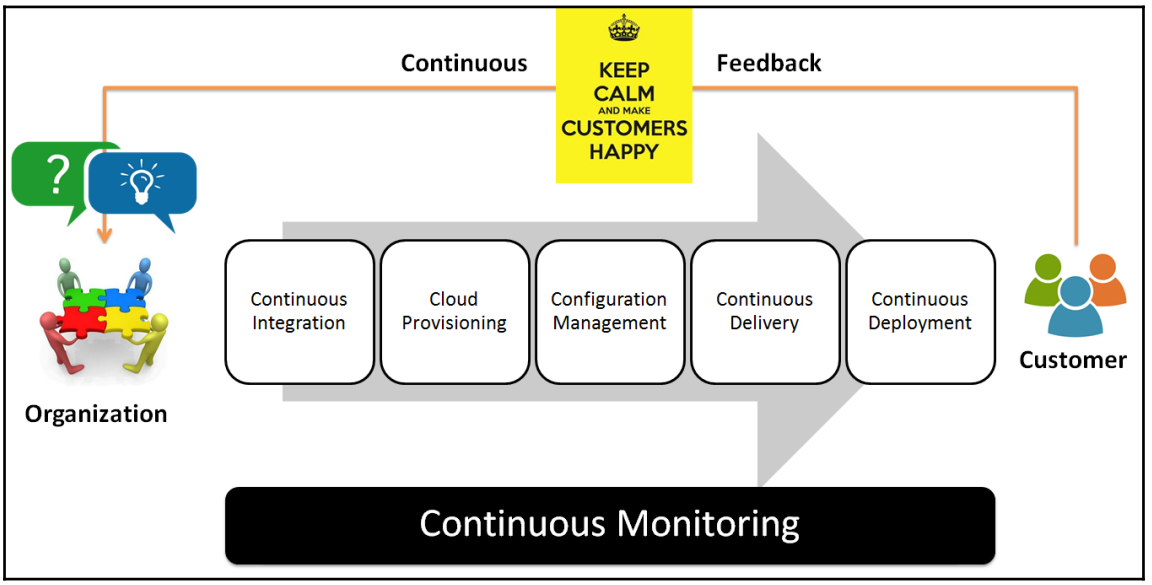



However, there is normal tendency to monitor only infrastructure resources. The question one must ask is whether it is enough or we must focus on other types of monitoring as well? To answer this question, we must have monitoring strategy in place in the planning stage only. It is always better to identify stakeholders, monitoring aspects, and so on based on culture and experience of an organization.

Continuous monitoring demystified
http://searchsecurity.techtarget.com/feature/Continuous-monitoring-demystified

# Continuous feedback

Continuous feedback is the last important component in the DevOps culture that provides way for improvement and innovation. Feedback always provides way of improvement if it comes from the stakeholders who know what they need and how the outcome should be. Feedback from the customer after deployment activities can server to developers as inputs for improvement as shown in below figure and its correct integration make customer happy.




Here, we are considering situation where feature implementation is provided to the stakeholders and they provide their feedback. In waterfall model, feedback cycle is very long and hence developers may not be aware about whether the end product is what customer asked for or interpretation of what needs to be delivered is changed somewhere. In Agile or DevOps culture, shorter feedback cycle is major difference as stakeholders can actually see the end result of small implementation phase and hence outcome is verified multiple times. If customer is not satisfied then feedback is available at a stage where it is not much painful to change things. In waterfall model it was a disaster as feedback used to come very late. With time and dependencies, complexities increases and changes in such situation takes long time. In addition to it, none remembers what they wrote 2 months back. Hence, faster feedback cycle improves overall process and also connects end points as well as finding patterns in mistakes, learning lessons, and using improved patterns. However, continuous feedback not only improves technical aspects of implementation but it also

provides way to assess current features and whether they fits into overall scenario or still there is a room of improvement. It is important to realize that Continuous feedback plays significant role in making customers happy by providing improved experience.

# Tools and technologies

Tools and technologies play important role in the DevOps culture however it is not the only part that needs attention. For all parts of application delivery pipeline, different tools, disruptive innovations, open source initiatives, community plugins and so on are required to keep the entire pipeline running to produce effective outcomes.

# Code repositories – Git

Subversion is a version control system that is used to track all the changes made to files and folders. By this a track can be kept on the applications which are being built. The features added months ago can also be tracked using the version code. It is all about tracking the code. Whenever any new features are added or any new code is made, it is first tested and then it is committed by the developer. Now the code is sent to the repository to track the changes and a new version is given to it. A comment can also be made by the developer so that other developer can easily understand the changes that are made. Other developers only have to update their checkout to see the changes made.

## Advantages

Following are some advantages of using source code repositories:

- No. of developers can work simultaneously on the same code
- If a computer crashes still the code can be recovered as it was committed in the server
- If a bug occurs the new code can be easily reverted back to the previous version

Git is an open source distributed version control system which is designed to handle from small to very large projects with speed and efficiency. It is easy to learn and has a good performance. It comprises of full-fledged repository and version control tracking capabilities independent of central server or a network access. It was developed and designed by Linux Torvalds in 2005.

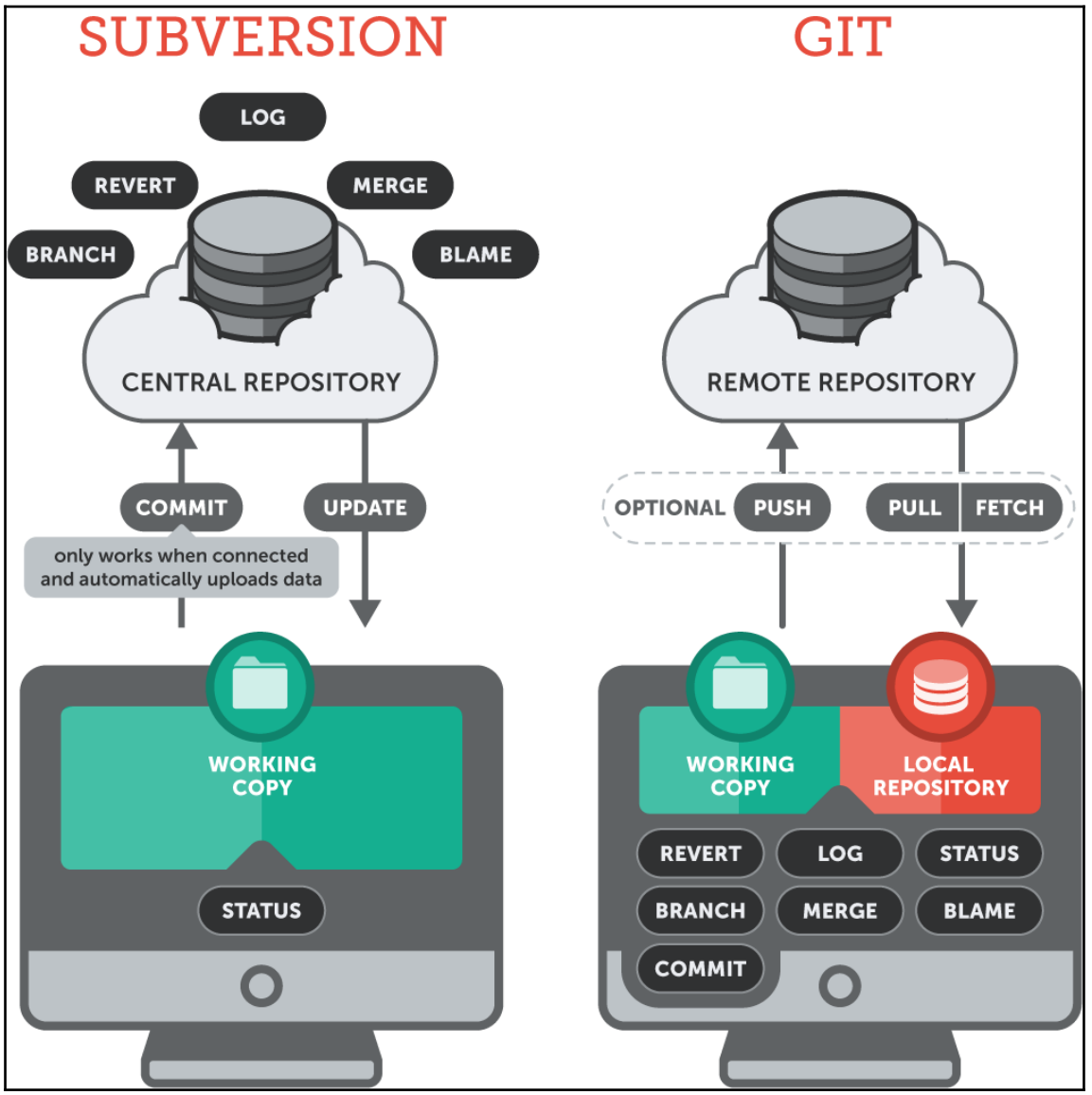
## Characteristics

Following are some significant characteristics of Git:

- It provides support for non-linear development
- It is compatible with existing systems or protocols
- It ensures cryptographic authentication of history
- It has well designed pluggable merge strategies
- It consists of tool-kit based designs
- It supports various merging techniques such as Resolve, Octopus, and Recursive

## Differences between SVN and Git

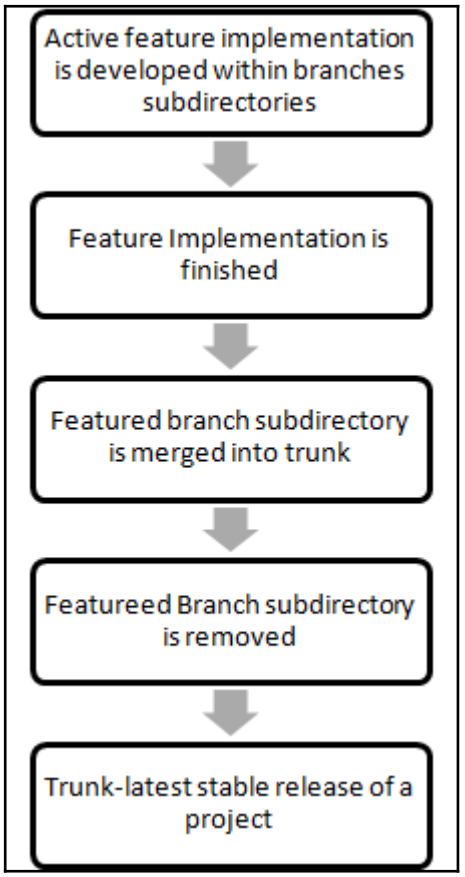
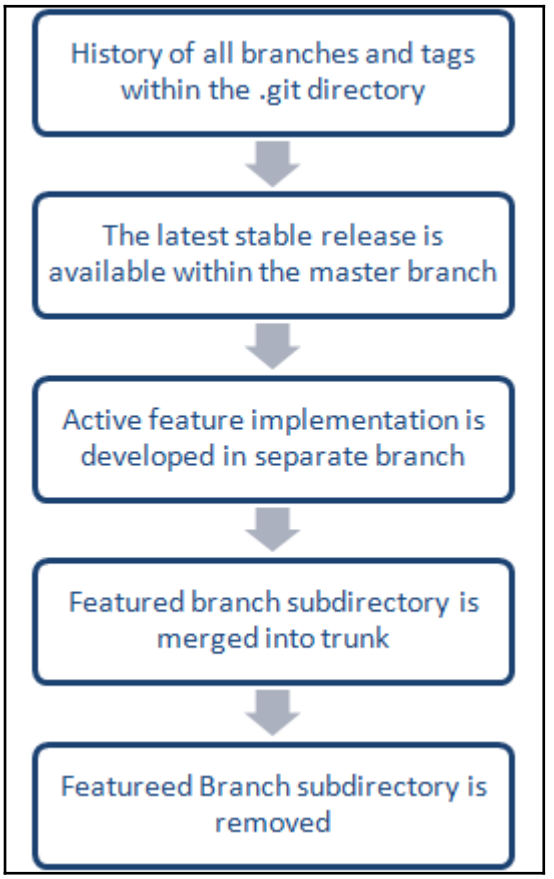
SVN and Git both are very popular source code repositories, however Git is getting more popular in recent times. Let's see the major differences between them both. The following figure shows visual difference between SVN and Git.



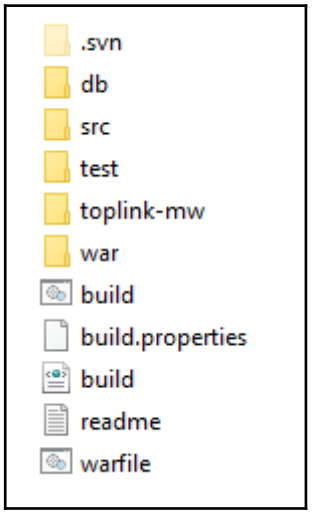

(Reference: https://www.git-tower.com/learn/git/ebook/en/mac/appendix/from-subversion-to-git)

| **Subversion** | **Git** |
|---|---|

| Centralized Version Control System | Distributed Version Control System |
|---|---|
| Snapshot of a specific version of the project is available on developer's machine. | Complete clone of a full-fledged Repository is available on developer's machine. |
| Perform operations such as commit, merge, blame, revert and so on; verify branch and log from a central repository. | Perform operations such as commit, merge, blame, and so on; verify branch and log from a local repository. Pull and Push operation to remote repository if developer needs to share work with others. |
| URLs are used to trunk, branches, or tags<br>Example of Repository URL: https://<URL/IP Address>/svn/trunk/AntExample1/ | .git is the root of project and commands are used to address branches and not URLs<br>Example of Repository URL: git@github.com:mitesh51/game-of-life.git |

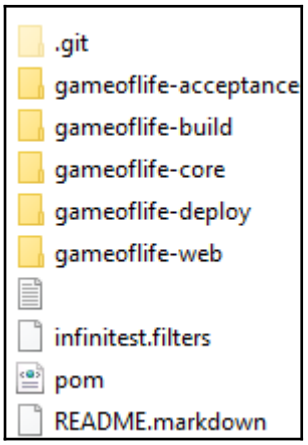
| A SVN Workflow: | A Git Workflow: |
|---|---|
|  |  |
| B05561_01_16 | B05561_01_17 |
| File changes are included in the next commit. | File changes has to be marked explicitly and then only they are included in the next commit. |
| Committed work is directly transferred to central repository and hence direct connection to repository must be available. | Committed work is directly transferred to remote repository. It is committed to local repository. To share it with other developers, we need to push it to remote repository and in this case we need a connection to remote repository. |
| Each commit gets ascending revision number | Each commit gets commit hashes rather than ascending revision number |

Application Directory:



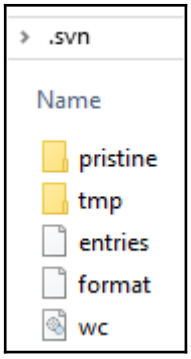

B05561_01_18

Application Directory:



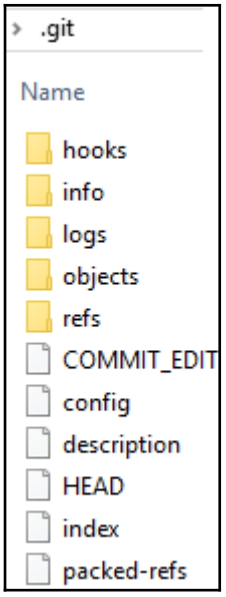

B05561_01_19

.svn directory structure:




B05561_01_20

.git directory structure:



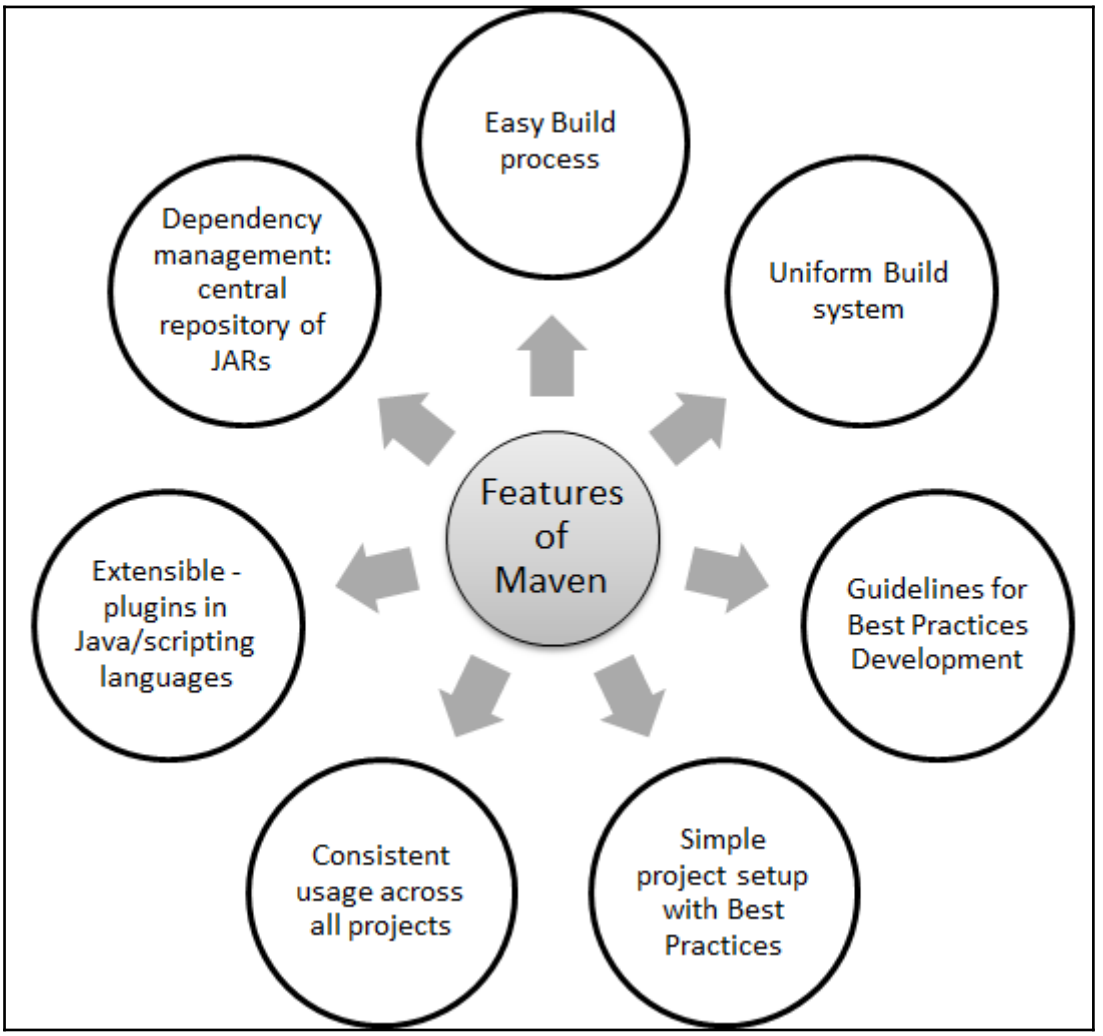

B05561_01_21

| Short Learning Curve | Long Learning Curve |
|---|---|

# Build tools – Maven

Apache Maven is build tool having Apache License 2.0 license. It is used for Java projects and it can be used in cross platform environment. However it can be used for Ruby, Scala, C#, and other languages.

The following are the important features of Maven:




A Project Object Model (POM)- XML file, contains information on name of the application, owner information, how application distribution file can be created, how dependencies can be managed.

## Example of POM.XML

POM.XML has pre-defined targets such as validate, generate-sources, process-sources, generate-resources, process-resources, compile, process-test-sources, process-test-resources,

test-compile, test, package, install, and deploy.

Following is an example of sample pom.xml file that is used in Maven:x

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.springframework.samples</groupId>
    <artifactId>spring-petclinic</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <name>petclinic</name>
    <packaging>war</packaging>
    <properties>

    <dependencies>
    <build>
        <defaultGoal>install</defaultGoal>
        <testResources>
          <testResource>
        <!-- Spring config files, next to corresponding JUnit test class  -->
            <directory>${project.basedir}/src/test/java</directory>
          </testResource>
          <testResource>
            <directory>${project.basedir}/src/test/resources</directory>
          </testResource>
        </testResources>
        <plugins>
    </build>
        <reporting>

    <url>demopetclinic</url>
</project>
```

# Continuous integration tools – Jenkins

Jenkins is originally open source continuous integration software written in Java having MIT License. However, Jenkins 2 an open source automation server that focuses on any automation including continuous integration and continuous delivery.

Jenkins can be used across different platforms such as Windows Ubuntu/Debian, Red Hat/Fedora, Mac OS X, openSUSE, and FreeBSD. Jenkins enables user to utilize continuous

integration services for software development in agile environment. It can be used to build free style software project based on Apache Ant and Maven 2/ Maven 3 Project. It can also execute Windows batch commands and shell scripts.

It can be easily customized with the use of plug-ins. There are different kinds of plug-ins available to customize Jenkins based on specific needs. Categories of plug-ins include Source code management (that is, Git Plugin, CVS Plugin, Bazaar Plugin), build triggers (i.e. Accelerated Build Now Plugin, Build Flow Plug-in), Build reports (that is, CodeScanner Plug-in, Disk Usage Plug-in), Authentication and user management (i.e. Active Directory plug-in, Github OAuth Plug-in), Cluster management and distributed build (that is, Amazon EC2 Plugin, Azure Slave Plugin), and so on.

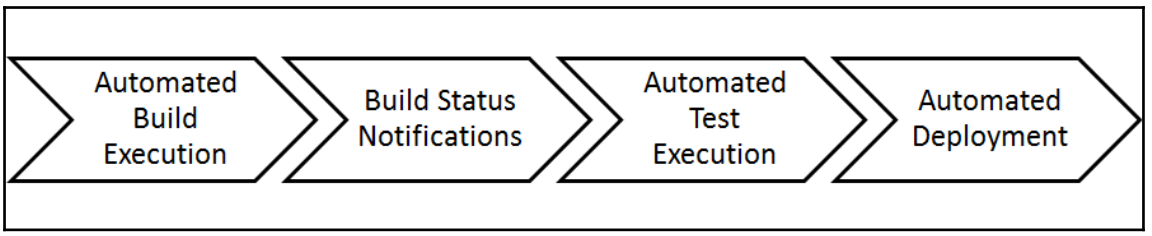
To know more about all plugins, visit
https://wiki.jenkins-ci.org/display/JENKINS/Plugins

To explore on how to create a new plugin, visit
https://wiki.jenkins-ci.org/display/JENKINS/Plugin+tutorial

To download different versions of plugins, visit
https://updates.jenkins-ci.org/download/plugins/

Continuous Integration Server: Jenkins http://jenkins.io/

Jenkins accelerates the software development process through automation



# Key Features and Benefits

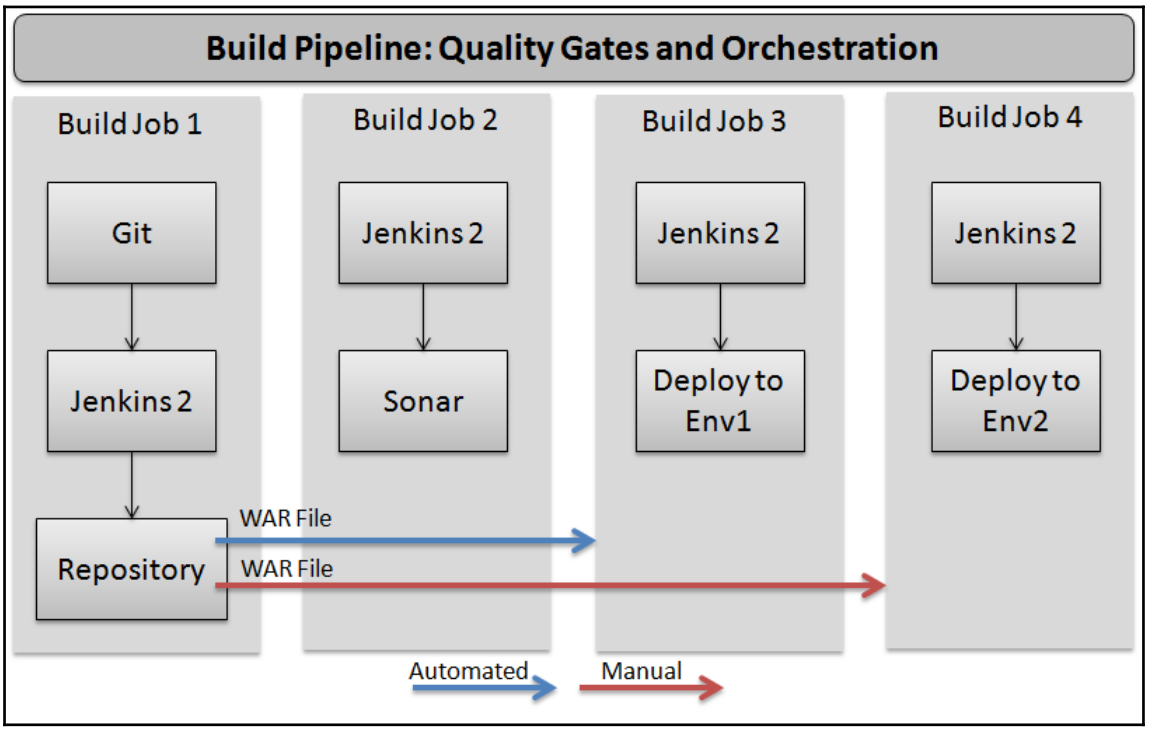
Following are some striking benefits of Jenkins:

- Easy install, easy upgrade, easy configuration
- Supported Platforms: Windows, Ubuntu/Debian, Red Hat/Fedora/CentOS, Mac OS X, openSUSE, FreeBSD, OpenBSD, Solaris, Gentoo
- Manages and controls development lifecycle processes
- Non java projects supported by Jenkins: .Net, Ruby, PHP, Drupal, Perl, C++, Node.js, Python, Android, Scala
- A development methodology of daily integrations verified by automated builds
- Every commit can trigger a build
- Jenkins is a fully featured technology platform that enables users to implement Continuous Integration (CI) and Continuous Delivery (CD)
- Use of Jenkins is not limited from continuous integration (CI) to continuous delivery (CD). It is possible to include model and orchestrate entire pipeline with the use of Jenkins as it supports shell and windows batch commands execution. Jenkins 2.0 supports delivery pipeline that uses a domain-specific language (DSL) for modeling entire deployment or delivery pipeline
- Pipeline as code provides a common language-DSL to help development and operations teams to collaborate in effective manner
- Jenkins 2 brings new GUI with stage view to observe the progress across delivery pipeline
- Jenkins 2.0 is fully backward compatible to Jenkins 1.x series of releases
- Jenkins 2 now requires Servlet 3.1 to run
- Use embedded Winstone-Jetty or use container that supports Servlet 3.1 (for example, Tomcat 8)
- GitHub, Collabnet, SVN, TFS code repositories, and so on are supported by Jenkins for Collaborative Development
- Continuous Integration: Automate build, test – automated testing (Continuous Testing), package, and static code analysis
- Supports common test frameworks such as HP ALM Tools, Junit, Selenium, MSTest, and so on
- For Continuous Testing, Jenkins has plugins for both; Jenkins slaves can execute the test suites on different platforms
- Jenkins supports static code analysis tools such as Code Verification support by Checkstyle and Findbug. It also integrates with Sonar
- Continuous Delivery and Continuous Deployment: automates the application

deployment pipeline; Integration with popular configuration management tools, and automated environment provisioning

- To achieve continuous delivery and deployment, Jenkins Supports Automatic Deployment; it provides plug-in for direct integration with IBM uDeploy
- Highly configurable tool-plugins-based architecture that provides support to many technology, repositories, build tools, and test tools; an Open-source CI server and provides 400+ plugins to achieve extensibility
- Supports Distributed builds: Jenkins supports the "master/slave" mode, where the workload of building projects are delegated to multiple "slave" nodes
- Machine-consumable remote access API to retrieve information from Jenkins for programmatic consumption, to trigger a new build, and so on
- Deliver better application faster by automating the application development lifecycle allowing faster delivery

Jenkins – Build Pipeline (Quality Gates) provides Build Pipeline View of upstream and downstream connected jobs: Chain of jobs each one subjecting build to quality assurance steps. It has the ability to define manual triggers for jobs that require intervention prior to execution such as an approval process outside of Jenkins.




Jenkins can be used with following tools in different categories.

| **Language** | **Java** | **.Net** |
|---|---|---|
| Code Repositories | Subversion, Git, CVS, StarTeam | |
| Build Tools | Ant, Maven | NAnt, MS Build |
| Code Analysis Tools | Sonar, CheckStyle, Findbugs, Ncover, Visual Studio Code Metrics PowerTool | |
| Continuous Integration | Jenkins | |

| | |
|---|---|
| Continuous Testing | Jenkins Plugins (HP Quality Center 10.00, with the QuickTest Professional Add-in, HP Unified Functional Testing 11.5x and 12.0x, HP Service Test 11.20 and 11.50, HP LoadRunner 11.52 and 12.0x, HP Performance Center 12.xx, HP QuickTest Professional 11.00, HP Application Lifecycle Management 11.00, 11.52, and 12.xx, HP ALM Lab Management 11.50, 11.52, and 12.xx), Junit, MSTest, VsTest) |
| Infrastructure Provisioning | Configuration Management Tool – Chef |
| Virtualization / Cloud Service Provider | VMware, AWS – Amazon Web Services, Microsoft Azure (IaaS), Traditional Environment |
| Continuous Delivery / Deployment | Chef / Deployment Plugin / Shell Scripting / Powershell Scripts / Windows Batch Commands |

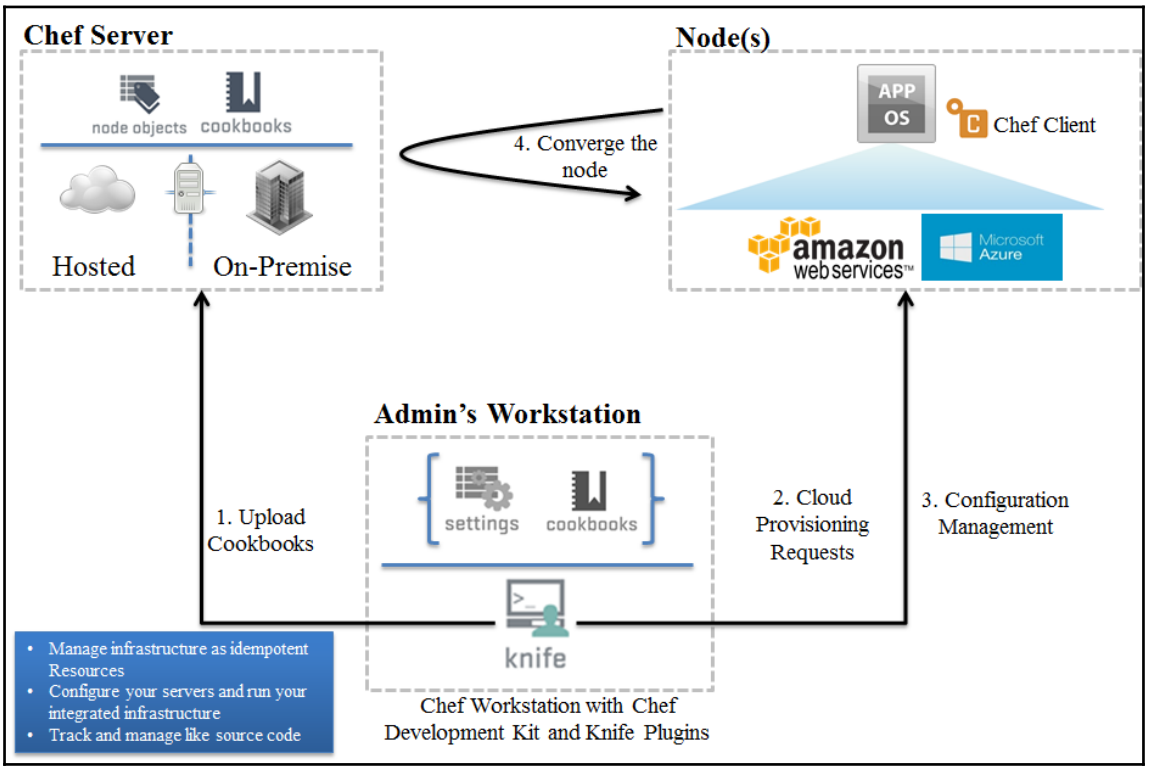
# Configuration management tools – Chef

Software Configuration Management (SCM) is a software engineering discipline comprising of tools and techniques that an organizations used to manage the changes in software components. It includes technical aspects of the project, communication, and control of modifications changes to the projects during development phase. It also called as Software Control Management. It constitutes of practices for all software projects ranging from development, rapid prototyping, or ongoing maintenance. It enriches the reliability and quality of software.

Chef is a configuration management tool which is used to transform the infrastructure into code. It automates building, deploying, and managing of the infrastructure. By using chef, infrastructure can be considered as a code. Concept behind chef is of reusability. It uses recipes to automate the infrastructure. Recipes are instructions required for configuring databases, web servers, and load balances. It describes every part of the infrastructure and how it should be configured, deployed, and managed. It uses building blocks known as resources. A resource describes parts of infrastructure such as template, package, and files to be installed.

This recipes and configuration data are stored in Chef Servers. Chef Client is installed on each node of the network. A node can be physical or virtual server.

The Chef client periodically checks the Chef server for the latest recipes and to see if the node is in compliance with the policy defined by the recipes. If it is out of date, the Chef client runs them on the node to bring it up to date.




# Features

Following are some important features of Chef Configuration Management Tool:

- Chef Server
  - It manages a huge amount of the nodes
  - It maintains a blueprint of the infrastructure
    - Chef Client
      - It manages various operating systems such as Linux, Windows, Mac OS, Solaris, and FreeBSD
      - It provides integration with cloud providers.
        - It is easy to manage the

containers in a version-able, testable, and repeatable way

- Chef provides automation Platform to continuously define, build, and manage Cloud infrastructure that is used for deployment
- It enables resource provisioning and configuration of resources programmatically and it will help in the deployment pipeline to automate provisioning and configuration

Three basic concepts of Chef will enable organizations to quickly manage any infrastructure with Chef:

- Achieving desired state
- Centralized modeling of IT infrastructure
- Resource primitives that serve as building blocks

Chef Configuration Management tool https://www.chef.io/

Chef Management Console https://manage.chef.io/login

# Cloud service providers

AWS and Microsoft Azure are popular public cloud providers in recent times. They provide cloud services in different areas and both have they strong areas. Based on the organization culture and past partnership anyone can be considered after detailed assessment based on requirements.

Followings are side by side comparison in terms of services:

| | **AWS** | **Microsoft Azure** |
|---|---|---|
| **Virtual Machines** | Amazon EC2 | Virtual Machine |
| **PaaS** | Elastic Beanstalk | Azure Web Apps |
| **Container Services** | Amazon EC2 Container Services | Azure Container Services |
| **RDBMS** | Amazon RDS | Azure SQL Database |
| **NoSQL** | DynamoDB | DocumentDB |
| **BIG Data** | Amazon EMR | HD Insight |
| **Networking** | Amazon VPC | Virtual Network |
| **Cache** | Amazon Elasticache | Azure RadisCache |
| **Import/Export** | Amazon Import/Export | Azure Import/Export |
| **Search** | Amazon CloudSearch | Azure Search |
| **CDN** | CloudFront | Azure CDN |
| **Identity and Access Management** | AWS IAM and Directory Services | Azure Active Directory |
| **Automation** | AWS OpsWorks | Azure Automation |

Amazon Web Services http://aws.amazon.com/

Microsoft Azure https://azure.microsoft.com

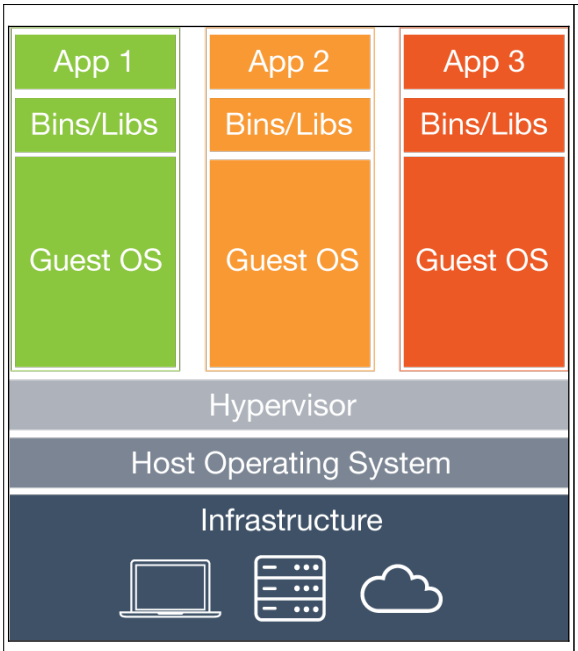
# Container technology

Containers use OS level virtualization where kernel is shared between isolated user spaces. Docker and OpenVZ are popular open source example of operating system-level virtualization technology.

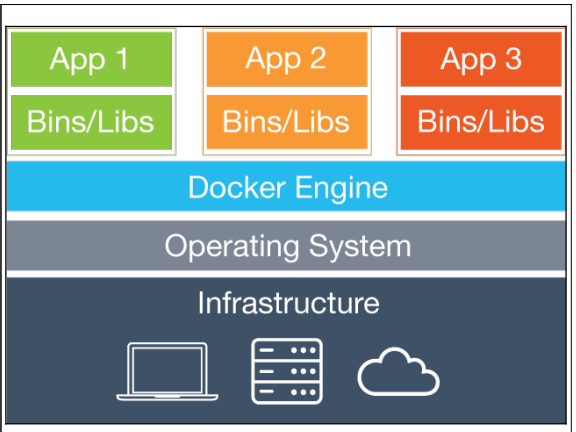
## Docker

Docker is an open source initiative to wrap code, runtime environment, system tools, and libraries. Docker containers share the kernel they are running on and hence they start instantly and lightweight in nature. Docker containers runs on Microsoft operating systems and Linux distributions. It is important to understand how containers and virtual machines are different. Below is the comparison table of Virtual machines and Containers.

| Virtual Machine | Docker Container |
|---|---|




B05561_01_27 (www.docker.com)




B05561_01_28 (www.docker.com)

| | |
|---|---|
| Virtual machines depend on the traditional virtualization. It can be considered as Hardware level Virtualization. | Container depends on containerization technique at a kernel level. It can be considered as Operating system level Virtualization. |
| Each virtual machine contains Guest operating system, Binaries, and library files and application itself. | Each container include application, Binaries, and library files but the major difference compared to virtual machine is the shared kernal; each container runs as isolated process in user space on Host OS. |
| Size of the each virtual machine is in GBs as each one runs on its own. | As each container runs as isolated process in user space on Host OS and separate operating system is not required for each container, size of each container is much less. |
| Each virtual machine has its own set of resources – better isolation and less sharing of resources. | Each container share kernel and hence more scope of sharing resources. |
| Virtual Machine can not run on Container. | Container can run on Virtual machine. |

Docker – the open-source application container engine
https://github.com/docker/docker

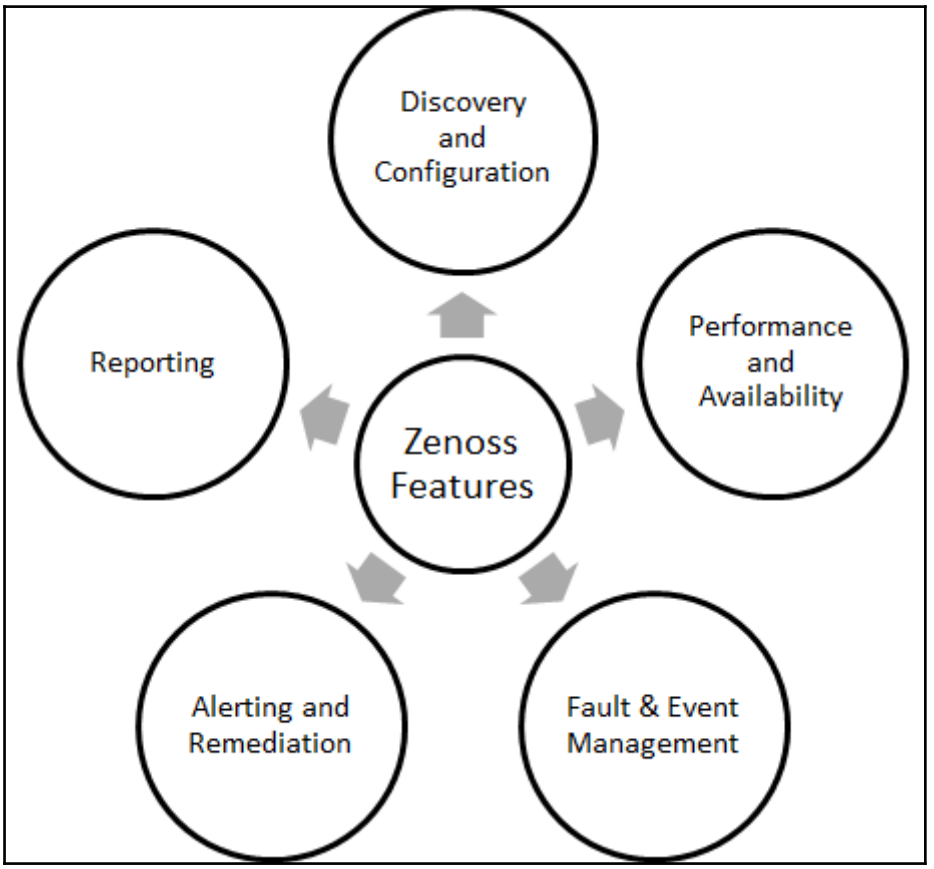
# Monitoring tools

There are many open source tools are available for monitoring resources. Zenoss and Nagios are one of the most popular open source tools adopted by many organizations.

## Zenoss

Zenoss is an agent less and open source management platform for application, server, and

network released under the GNU General Public License (GPL) version 2 based on the Zope application server. Zenoss Core consists of extensible programming language Python, object-oriented web server Zope Application server, Monitoring protocol Net, Graph and log time series data by RRDtool, MySQL, and event-driven networking engine Twisted. It provides easy to use web portal to monitor alerts, performance, configuration, and inventory.




Zenoss Core 5 http://www.zenoss.org/

## Nagios

Nagios is a cross platform and open source monitoring tool for infrastructure and network. It monitors network services such as FTP, HTTP, SSH, and SMTP. It monitors resources, detects the problems, and alerts the stakeholders. Nagios can empower organizations and service providers to identify and resolve issues in a way that outages have minimal impact on IT infrastructure and processes hence highest compliance to SLAs. Nagios can monitor cloud resources such as compute, storage, and network.

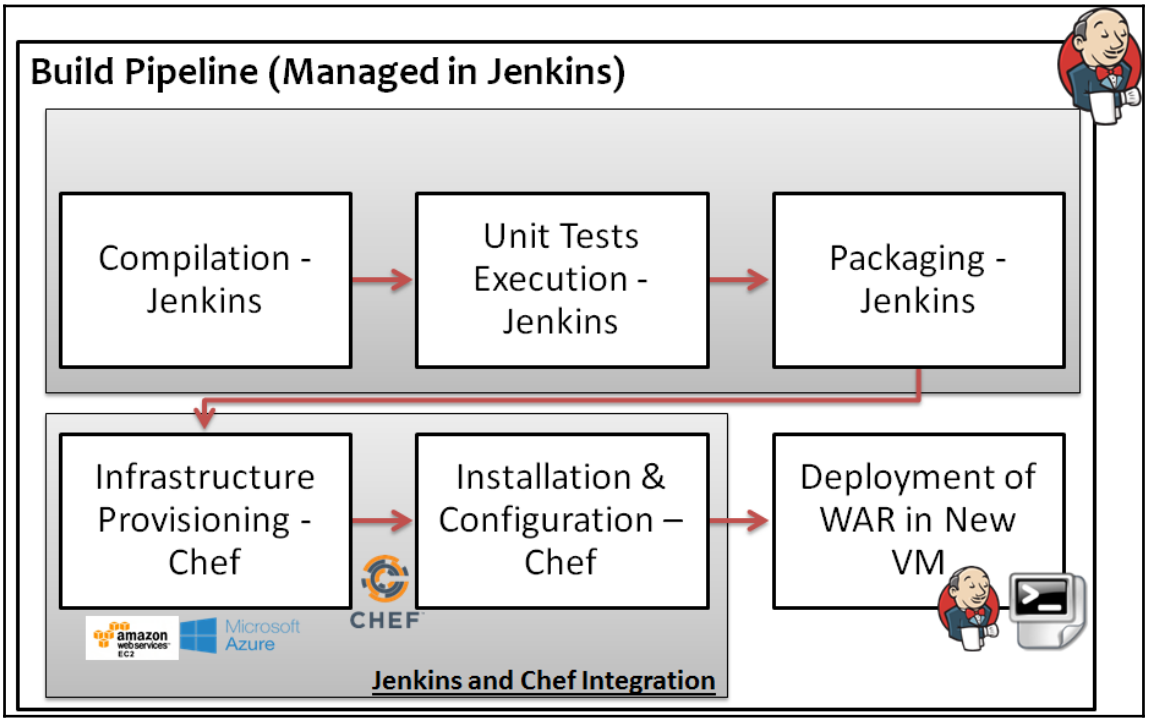
The Industry Standard In IT Infrastructure Monitoring
https://www.nagios.org/

# Deployment Orchestration / Continuous Delivery – Jenkins

Build pipeline or Deployment Pipeline or Application Delivery pipeline, can be used to achieve end to end automation for all operations. Starting from Continuous Integration, Cloud Provisioning, Configuration Management, Continuous Delivery, Continuous Deployment, and Notifications. Jenkins plugins can be used for overall orchestration of all activities involved in end to end automation.

- Continuous Integration: Jenkins
- Configuration Management: Chef
- Cloud Service Providers: AWS, Microsoft Azure
- Container Technology: Docker
- Continuous Delivery / Deployment: ssh
- End to End Orchestration: Jenkins Plugins

Following is a sample representation of end to end automation using different tools:




Jenkins can be used to manage unit testing, code verification; Chef can be used for setting up runtime environment; knife plugins can be used for creating a virtual machine in AWS or in Microsoft Azure; Build pipeline or Deployment pipeline plugin in Jenkins can be used for managing deployment orchestration.

From a single pipeline dashboard, we can view status of all builds which are configured in pipeline. Each build in the pipeline is a kind of quality gate. If one build fails then execution won't go further. Another dimensions can be added such as notification based on compilation failures, unit test failure or for unsuccessful deployment. Final deployment can be based on some sort of permission from a specific stakeholder. We can consider a scenario for parameterized build or promoted build concept. What we will do? Wait for upcoming chapters and all secretes will be revealed.

# DevOps Dashboard

One of the most desired components to get into DevOps culture is Dashboard functionality or GUI that provides combined status of all end to end activities. For automation tools, easy

to use web GUI is handful for management of resources. For end to end automation in application deployment activity, multiple open source or commercial tools are used. There is a high possibility where single product may not be used for all activities. For example, Git or SVN as repository, Jenkins as CI server, IBM Urbancode Deploy as deployment orchestration tool. In such scenario, it is easier if there is single pane of glass view where we can track multiple tools for a specific application.

Hygieia is an open source DevOps dashboard that provides way to track status of deployment pipeline. It basically tracks 6 different areas as of now including features (Jira, VersionOne), code repo (GitHub, Subversion), build (Jenkins, Hudson), quality (Sonar, Cucumber/Selenium), monitoring, and deployment (IBM UrbanCode Deploy)

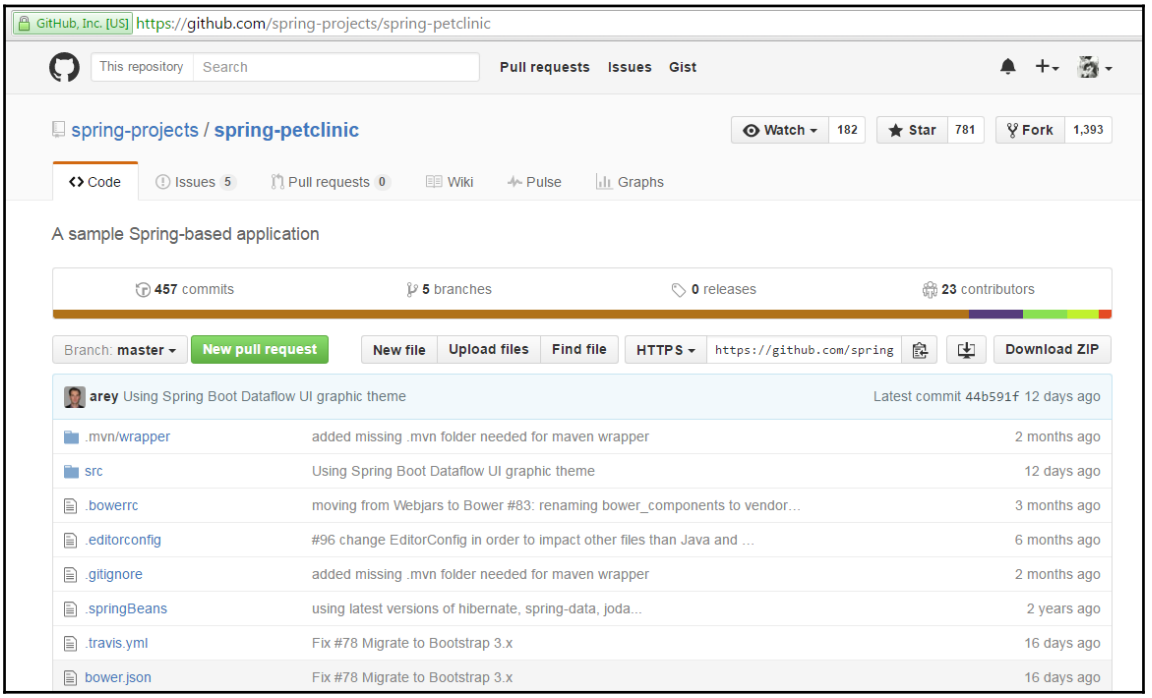
CapitalOne DevOps Dashboard https://github.com/capitalone/Hygieia

# Overview of Sample JEE Application

We are going to use PetClinic Application available on GitHub.

A sample Spring-based application
https://github.com/spring-projects/spring-petclinic

The PetClinic sample application can be used to build simple and robust database-oriented applications to demonstrate the use of Spring's core functionality. It is accessible via web browser.




A few use cases:

- Add a new pet owner, a new pet, and information pertaining to a visit to the pet's visitation history to the system
- Update the information pertaining to a pet and a pet owner
- View a list of veterinarians and their specialties, a pet owner, a pet, and pet's visitation history

Once WAR file is created, we can deploy it in Tomcat or another web server and to verify it in localhost, visit http://localhost:8080/petclinic. We will see something like:


# List of Tasks

Following are the tasks we will try to complete in rest of the chapters:

- Jenkins Installation, Configuration, UI Personalization
- Java configuration (JAVA_HOME) in Jenkins
- Maven or Ant configuration in Jenkins
- Plugins installation and configuration in Jenkins
- Security (Access Control, Authorization, Project based Security) in Jenkins
- Jenkins build configuration and Execution
- Email Notification Configuration
- Deployment of WAR file to Web Application Server
- Eclipse Integration with source code repository
- Create and configure a Build / Deployment Pipeline

- Install and Configure Chef – Configuration Management tool
- Install and Configure Docker
- Create and configure Virtual machine in AWS and Microsoft Azure and containers
- Deploy WAR file into Virtual Machine and Container
- Configure Infrastructure monitoring
- Orchestrate Application delivery pipeline using Jenkins plugins

# Self-Test Questions

1. Which of the following statement is not related to Development team in Traditional Environment?

1. Competitive Market creates pressure of on time delivery of feature or bug fixing

2. Production ready Code Management and New feature Implementation

3. Release cycle is often long and hence development team has to make assumptions before the application deployment finally takes place

4. Redesigning or tweaking is needed to run the application in Production environment

1. Which of the following are benefits of DevOps?

5. Collaboration, Management, and security for the complete application development lifecycle management

6. Continuous innovation because of continuous development of new ideas

7. Faster delivery of new features or resolution of issues

8. Automated deployments and standardized configuration management for different environments

9. All of the above

1. Which of the following are parts of DevOps culture or Application Delivery Pipeline?

10. Continuous Integration

11. Cloud Provisioning
12. Configuration Management
13. Continuous Delivery / Deployment
14. Continuous Monitoring
15. Continuous Feedback

    1. Which of the following are by product of DevOps culture or Application Delivery Pipeline?

16. Continuous Integration
17. Continuous Delivery / Deployment
18. Continuous Monitoring
19. Continuous Feedback
20. Continuous Improvement
21. Continuous Innovation

    1. State whether following statement is True or False: Jenkins and Atlassian Bamboo are a build automation tool.

22. True
23. False

    1. State whether following statement is True or False: Apache Ant and Apache Maven are Continuous Integration Tools

24. True
25. False

    1. State whether following statement is True or False: Chef is a configuration management tool.

26. True
27. False

1. State whether following statement is True or False: Build automation is essential for Continuous Integration and rest of the automation is effective only if build process is automated.

28. True

29. False

1. State whether following statement is True or False: Subversion is a Distributed Version Control System.

30. True

31. False

1. State whether following statement is True or False: Git is a Centralized Version Control System.

32. True

33. False

1. Which of the followings are Cloud Deployment Models according to NIST's definition of Cloud Computing?

34. Public Cloud

35. Private Cloud

36. Community Cloud

37. Hybrid Cloud

38. All of the above

1. Which of the followings are Cloud Service Models according to NIST's definition of Cloud Computing?

39. Software as a Service

40. Platform as a Service

41. Infrastructure as a Service

42. All of the above

1. State whether following statement is True or False: AWS and Microsoft Azure are Public Cloud Service Providers

43. True

44. False

1. Which of the following are main component of Chef Installation?

45. Chef Server / Hosted Chef

46. Chef Workstation

47. Nodes

48. All of the Above

# Summary

In this chapter, we have learnt about difficulties faced by development and operations team in traditional environment and how Agile helps in such scenario. What has changed after arrival of agile development methodology and what challenges it brought with its arrival? We have covered important aspects of DevOps culture including Continuous Integration and Continuous Delivery. We also covered details regarding Cloud computing and Configuration Management that enhances the processes and helps to adopt DevOps culture.

In terms of tools and technologies, we have covered brief overview of SVN, Git, Apache Maven, Jenkins, AWS, Microsoft Azure, Chef, Nagios, Zenoss, and Hygieia-DevOps Dashboard.

In the next chapter, we will see how to install and configure Jenkins and how to implement Continuous Integration best practices by using sample spring application available on Github.

It is a right time to quote Charles Darwin as it is relevant in the context of DevOps culture:

It is not the most intellectual or the strongest species that survives, but the species that survives is the one that is able to adapt to or adjust best to the changing environment in which it finds itself.

# 2
# Continuous Integration with Jenkins 2

*"The way to get started is to quit talking and begin doing."* – *Walt Disney*

Jenkins 2 has already arrived. Jenkins 2 comes with Built-in support for delivery pipelines, Improved usability – a new setup experience and total backwards compatibility with existing Jenkins installations. For the last time, we are using Jenkins 2 in this book.

This chapter describes in detail how Jenkins plays an important role in Continuous Integration. It covers how to prepare runtime environment for application lifecycle management and configure it with Jenkins. It manages all aspects of running a build to create a distribution file or war file for deployment by integrating source code repository such as SVN / Git for Sample JEE application. Jenkins 2 is recently made available for general usage and we have used Jenkins 2 in this book.

Readers will learn how to install and configure Jenkins and they will be able to get end to end experience from build job creation, configuration of build job, static code analysis, notifications, Jenkins plugins etc. and details on what exactly the sample application is all about.

In this chapter, we will cover the following topics:

- Jenkins – Introduction
- Jenkins installation with plugins
- Java, Maven/Ant, configuration in Jenkins
- Create and configure build job for Java application with Maven
- Dashboard view plugin – overview and usage
- Email notifications based on build status

- Jenkins and Sonar integration

# Introduction

We all know what **Continuous Integration (CI)** is, right? It is the first step in our journey.

> *"The journey of a thousand miles begins with one step." Lao Tzu father of Taoism*

In simple words, CI is a software engineering practice where each check-in by a developer is verified by

- Pull mechanism: executing automated build at a scheduled time or
- Push mechanism: executing automated build when changes are saved in repository and
- Executing unit test against latest changes available in source code repository.

Jenkins doesn't need an introduction; still it is an open source and one the most popular Continuous Integrations tools available in the market. It helps in automating repetitive task of continuous integration. Jenkins makes the process effective and transparent.

> *"We are what we repeatedly do. Excellence, then, is not an act, but a habit." – Aristotle*

Next question you may ask is what makes Jenkins so popular? I have already given one reason; can you recollect?

Yes, because it is open source; however, open source tools come with predefined notions but Jenkins community is different and Jenkins as a tool is quite different.

So, what are the other reasons for popularity of Jenkins? Let's have a look:

- Written in Java
- Extensibility: 400+ plugins for different integrations are available
    - Source code management
    - Build triggers
    - Build reports
    - Artifact uploaders
    - External site/tool integrations
    - UI plugins
    - Authentication and user management
    - Cluster management and distributed build

- Others
  - Supports Java, .NET, Ruby, Groovy, Grails, PHP, Android, iOS Applications
  - Easy to use
    - Easy Installation
    - Easy Configuration
      - Simple learning curve
      - User Interface was already simple, now improved after Jenkins 2 is available for General Availability

# Installing Jenkins

Jenkins provides multiple ways to install it for all types of users. We can install Jenkins on following operating systems of platforms:

- Ubuntu/Debian
- Windows
- Mac OS X
- OpenBSD
- FreeBSD
- OpenSUSE
- Gentoo
- CentOS/Fedora/Red Hat

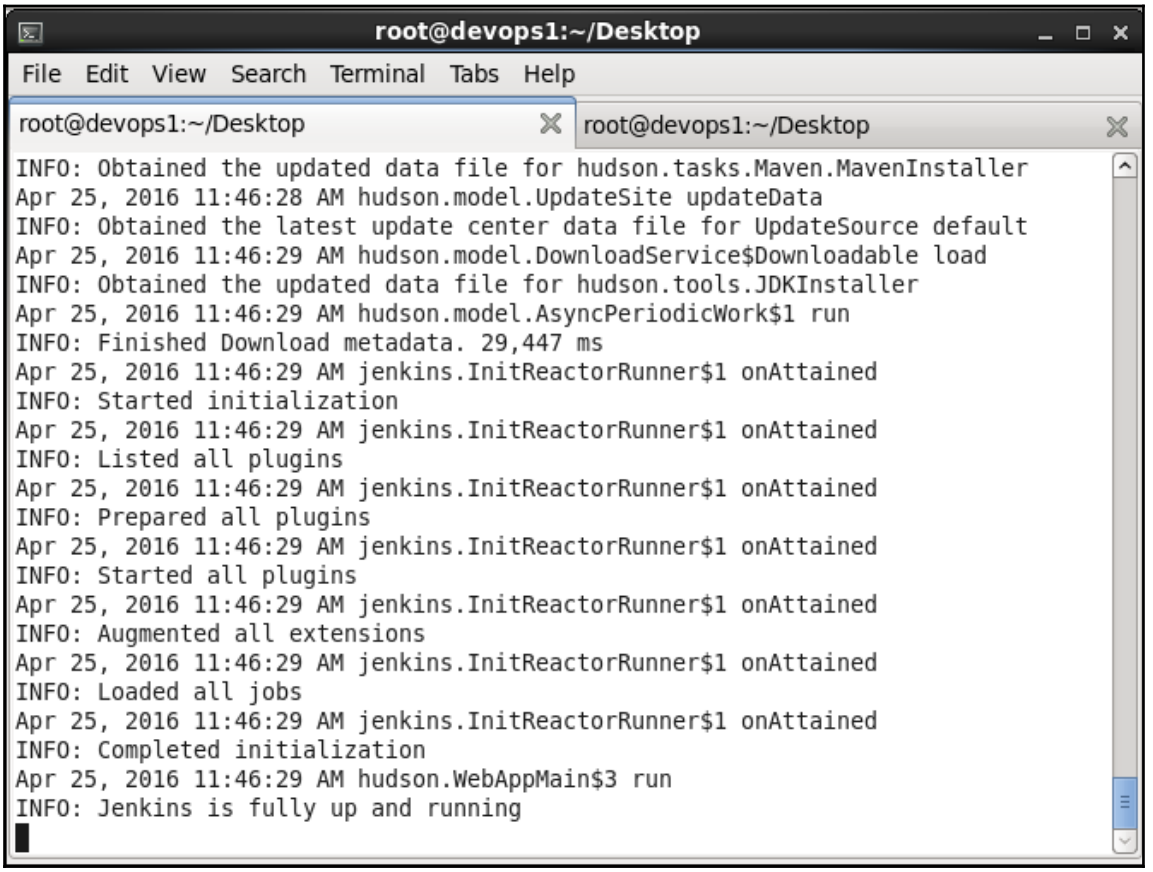
One of the easiest options I recommend is to use WAR file. WAR file can be used with container or Web application Server or without it. Java installation is must before we intend to use WAR file for Jenkins.

1. Download jenkins.war file from https://jenkins.io/
2. Open command prompt in Windows or Terminal in CentOS; go to the directory where Jenkins.war file is stored and execute following command in command

prompt or terminal:

```
java - jar jenkins.war
```

1. Once, Jenkins is fully up and running as shown in the screenshot below, explore Jenkins in the web browser by visiting http://localhost:8080.



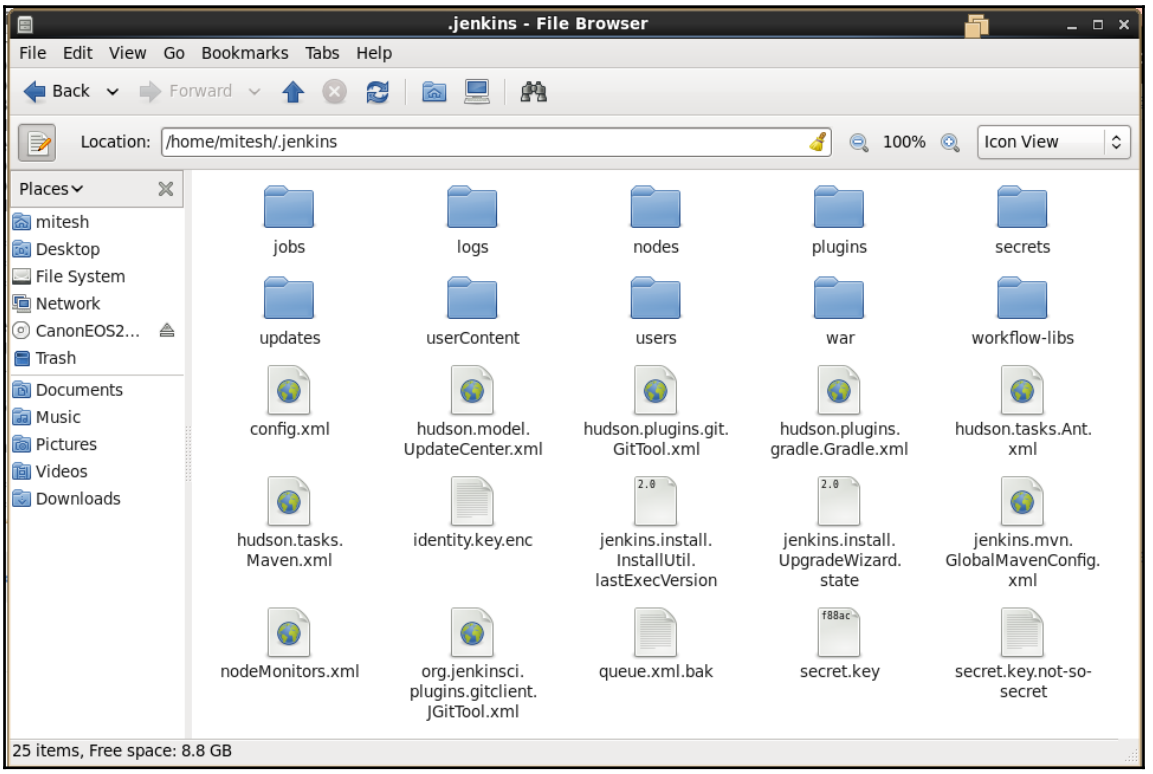

1. By default, Jenkins works on port 8080. Execute the following command at command prompt in Windows or Terminal in Linux:

```
java -jar jenkins.war --httpPort=9999
```

1. For https, use the following command at command prompt in Windows or Terminal in Linux:

```
java -jar jenkins.war --httpsPort=8888
```

1. Once Jenkins is running, visit Jenkins home directory. In our case we have installed Jenkins 2 on CentOS 6.7 virtual machine.
2. Go to /home/<username>/.jenkins directory as shown in the screenshot below. If .jenkins directory is not available then make sure that hidden files are visible. In CentOS, press ctrl+h to make hidden files visible.

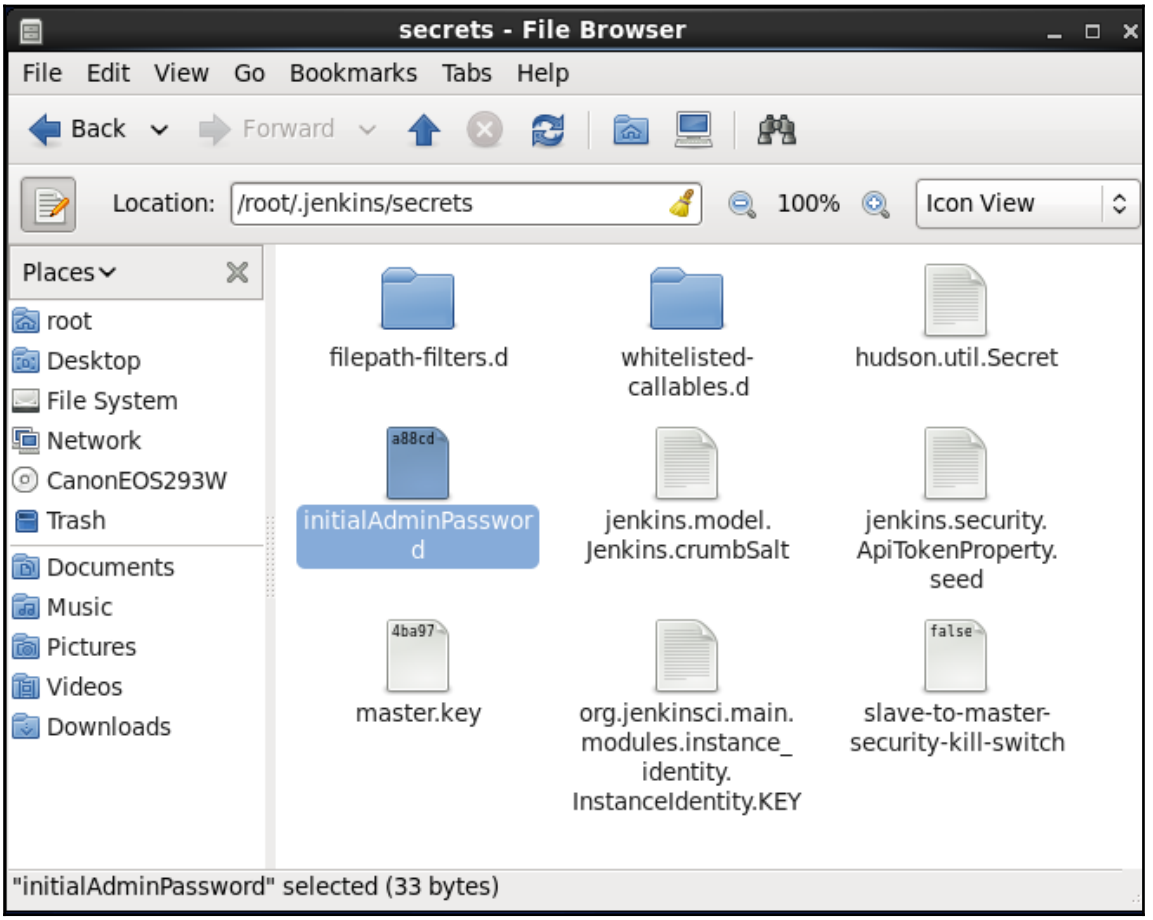



# Setting up Jenkins

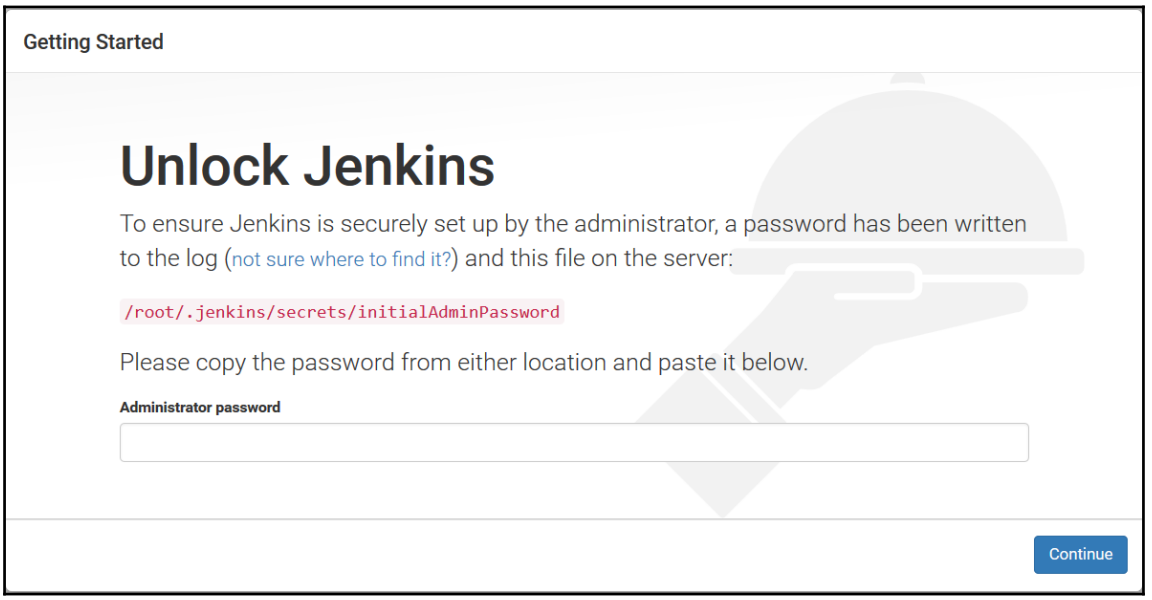
Now that we have installed Jenkins, let's verify whether Jenkins is running or not. Open any browser installed in your system and navigate to http://localhost:8080 or http://<IP_ADDRESS>:8080. If you have already used Jenkins earlier and recently downloaded Jenkins 2 WAR file, then it will ask for security setup.

To unlock Jenkins, we will follow these steps:

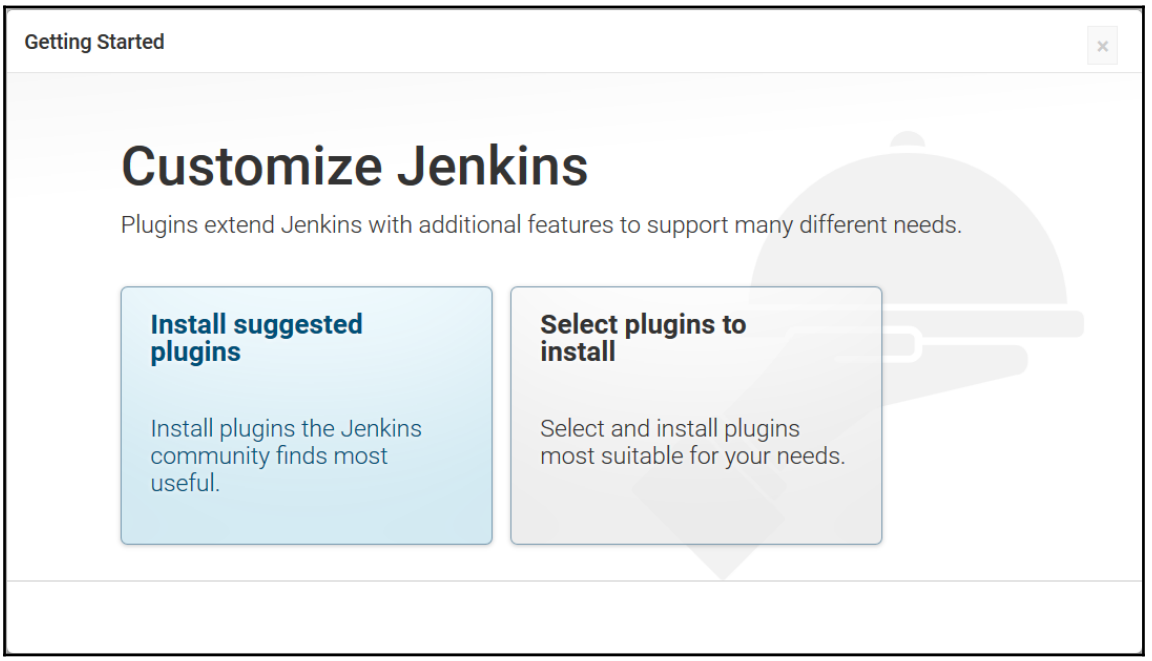
1. Go to .jenkins directory and open initialAdminPassword file from secrets subdirectory as shown below:



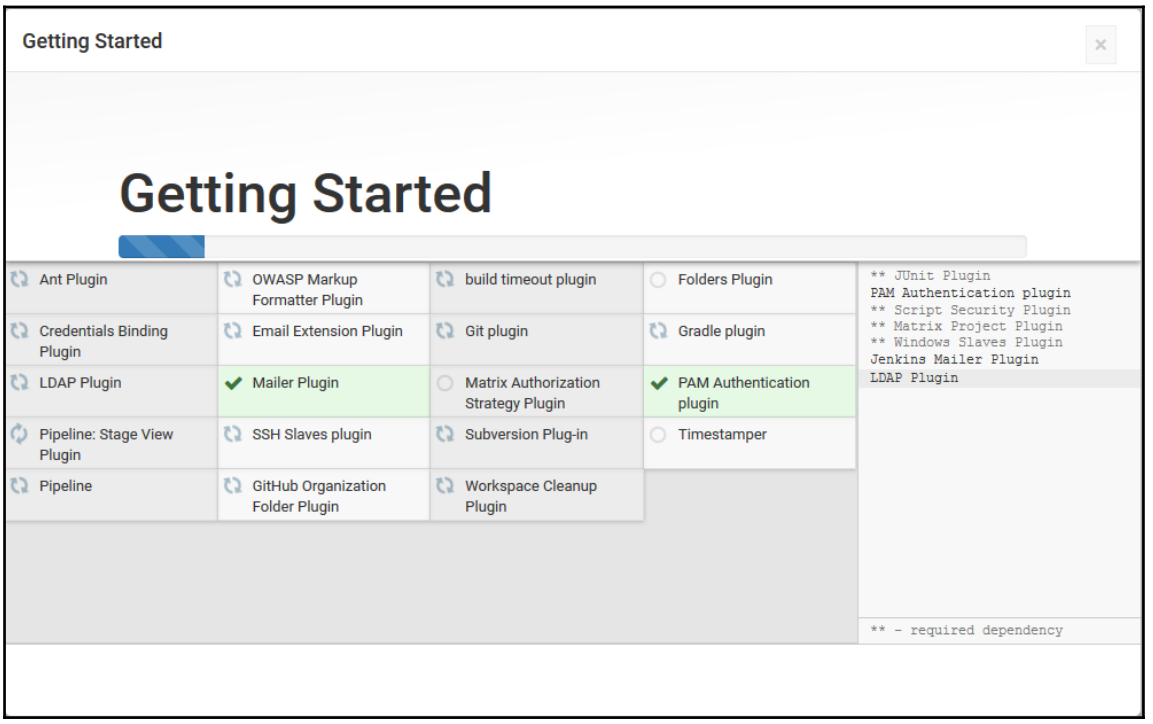

**2.** Copy password available in that file and paste it in **Administrator password** box and click on **Continue** as shown below:



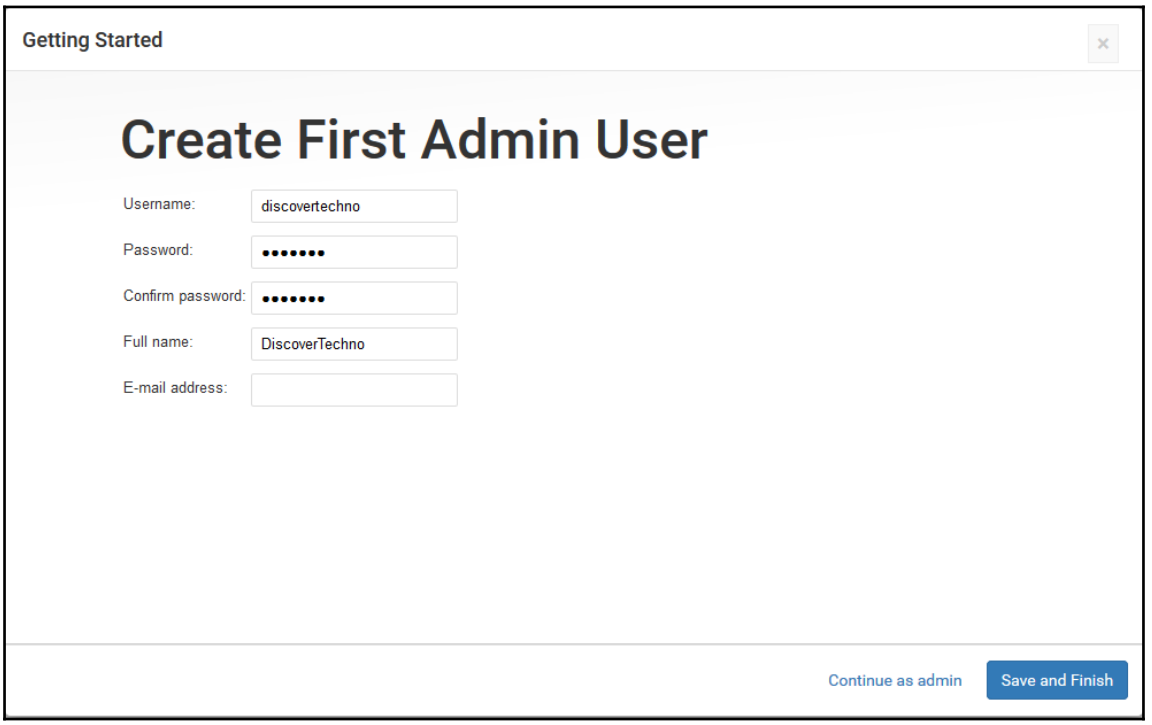

3. Clicking **Continue** will redirect you to **Customize Jenkins** page as shown below. Click on **Install suggested plugins**.



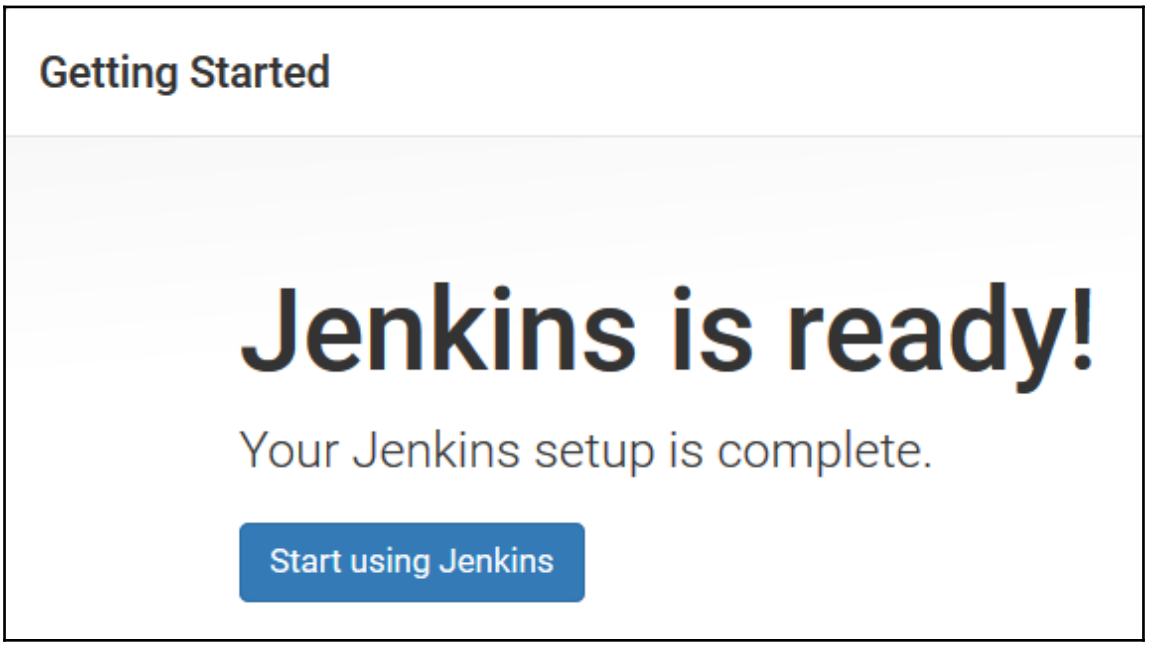

4. Installation of required plugins will start. Make sure that you have internet connection working.




5. Once all required plugins are installed; you will see **Create First Admin User** page. Provide required details and click on **Save and Finish**.




**6. Jenkins is Ready!** Our Jenkins setup is completed. Click on **Start using Jenkins**.



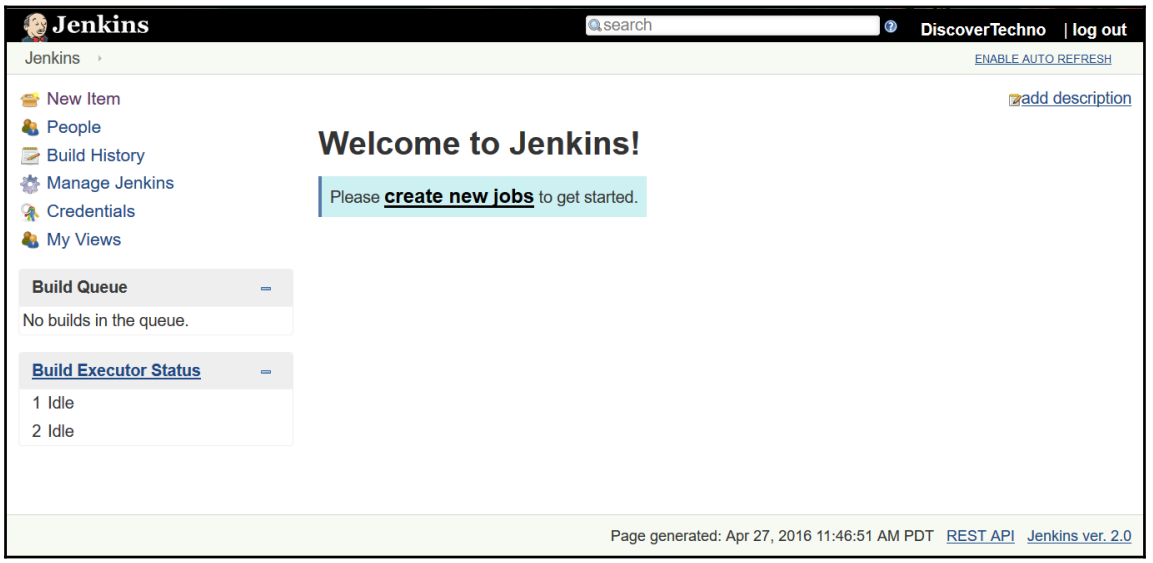

Get Jenkins plugins at:
https://wiki.jenkins-ci.org/display/JENKINS/Plugins

# Jenkins dashboard

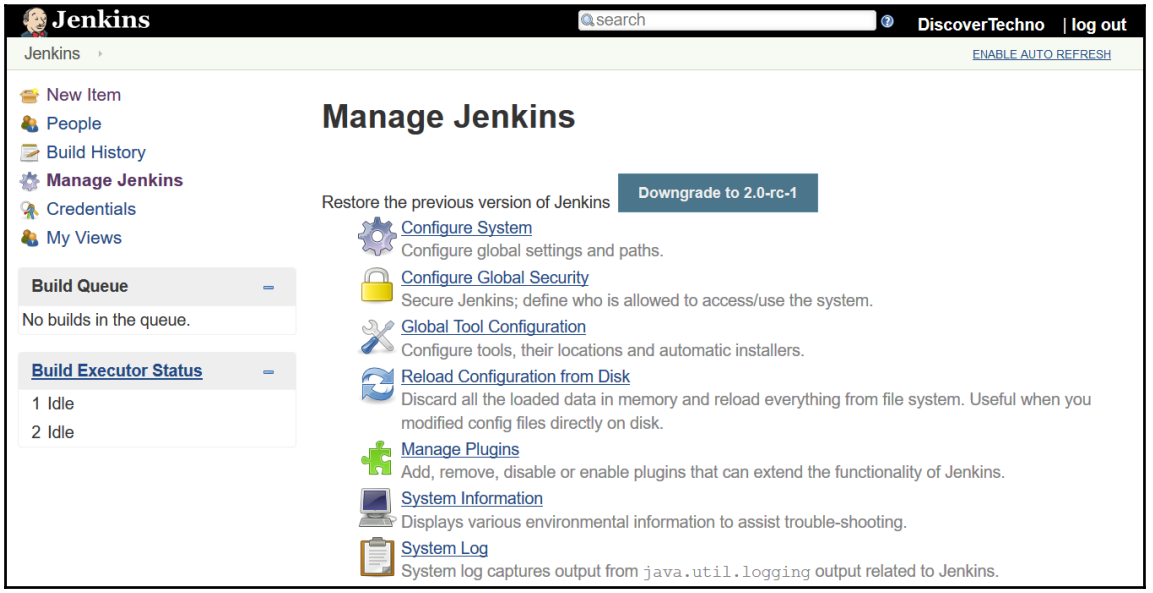
Jenkins dashboard is a simple and powerful place where we can manage all builds and hence we can manage application delivery pipeline too. Open http://<localhost or IP address>:8080 from browser. Log in with the user credentials which we have created earlier. It will direct us to dashboard of Jenkins.

Let's understand the dashboard parameters:

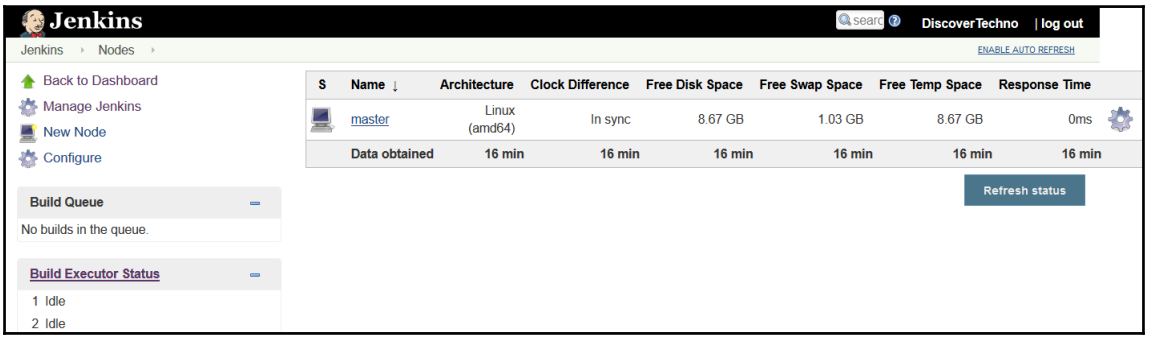
- **New Item:** To create new build job, pipeline or build flow in Jenkins 2.




- **Manage Jenkins:** Allows Jenkins 2 administrator to manage plugins, users, security, nodes, credentials, global tools configuration and so on.



- To know about existing nodes that are used for build execution, click on **Manage Nodes**. **master** node entry will be available. It is the node where Jenkins is installed. We can add multiple slave node to distribute the load the we will learn later in this chapter.




Once we have installed Jenkins and as we become familiar with the Jenkins dashboard, the next is to configure different tools that are used for build execution and create a base for Continuous Integration.

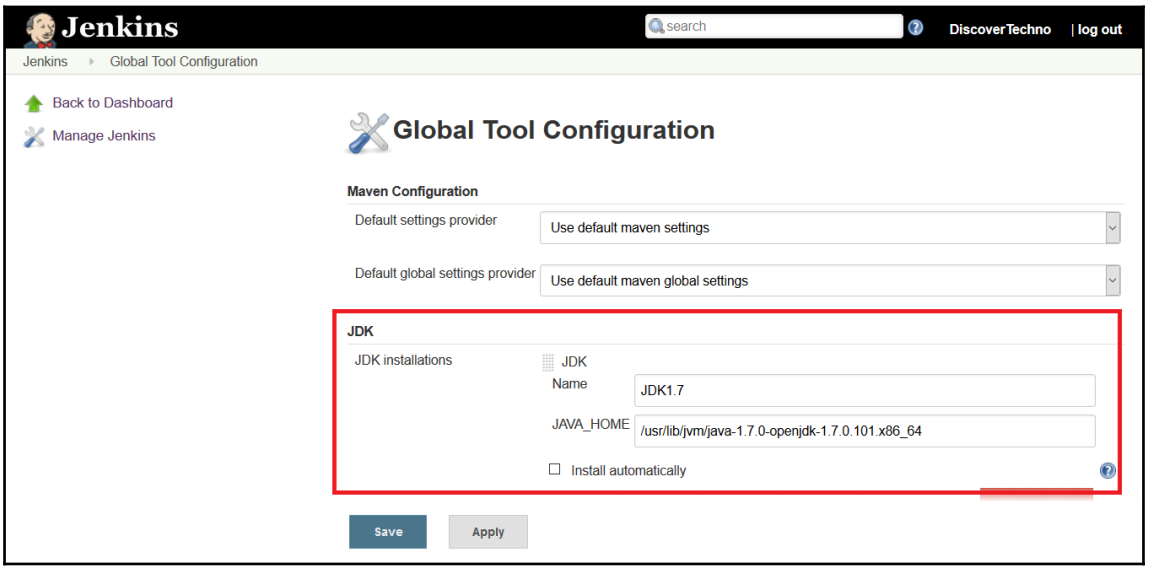
In next section, we will install or configure Java, Maven, Ant and so on.

# Configuration Java, Maven/Ant in Jenkins

In Jenkins 2, **Global Tool Configuration** section is introduced and that is a good move. All major configurations related to external tools, their locations and automatic installers tools can be done in this section. Earlier, these configurations were part on **Configure System** which used to make that page bit cluttered.

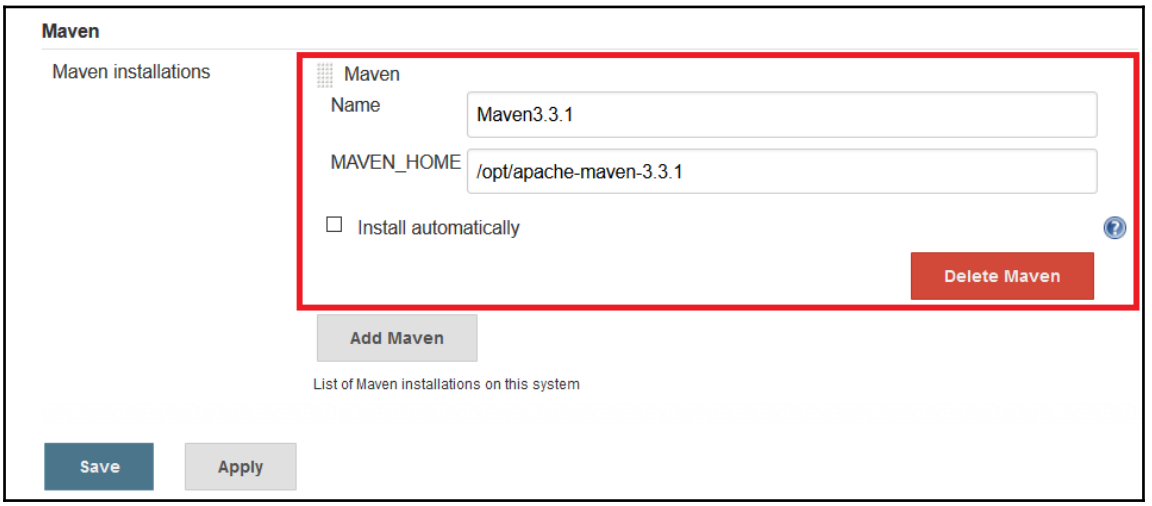
# Configuring Java

To configureJava, provide **Name** and **JAVA_HOME** path or click on **Install automatically**.




# Configuring Maven

To configure Maven, download installable files of Maven, extract it and keep in some directory of your Jenkins virtual machine. In **Global Tool Configuration** section, provide **Name** and **JAVA_HOME** path or click on **Install automatically**.



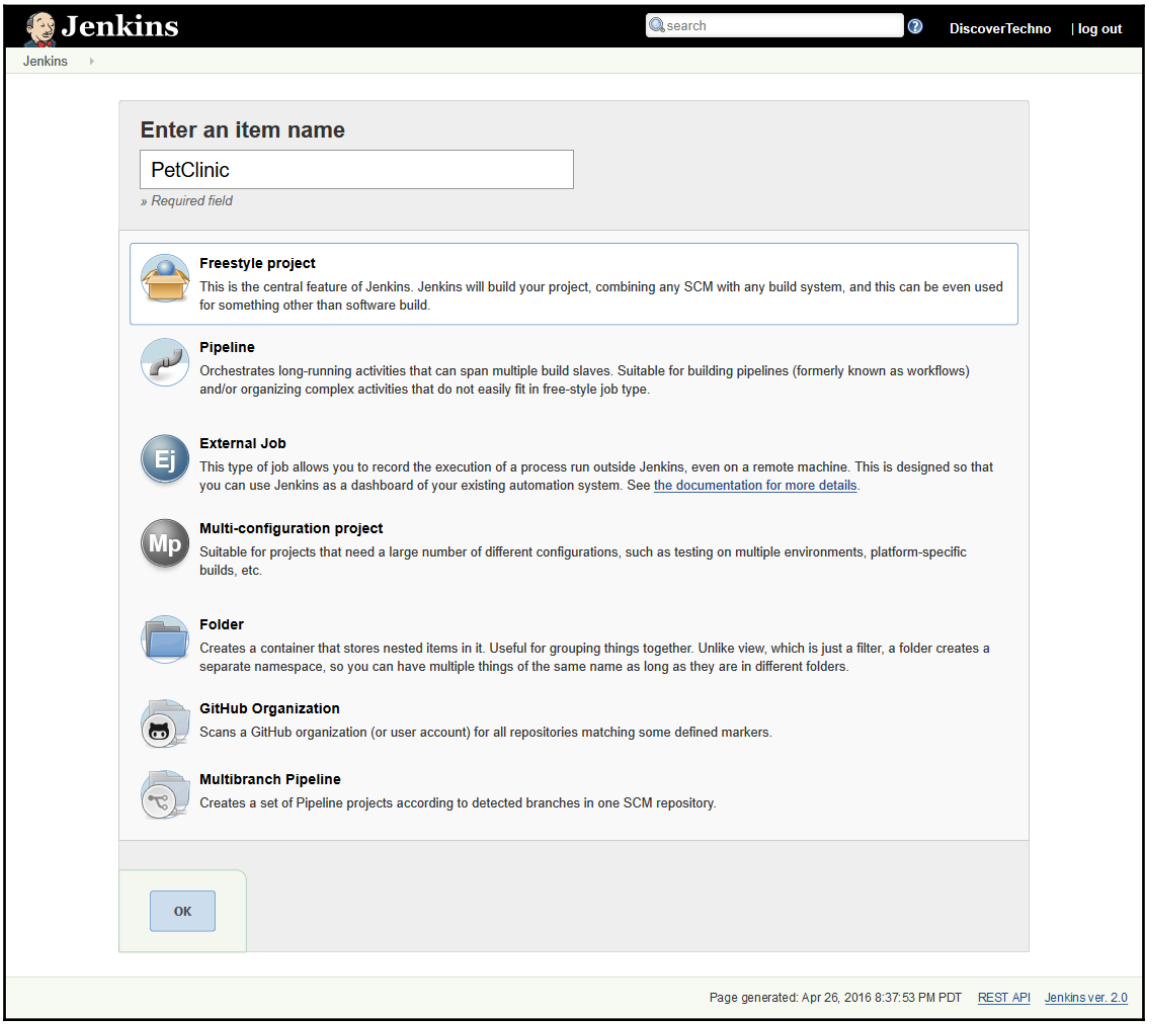

That's it! Our major configurations for running a simple build is done. Now let's go to the Home page of Jenkins dashboard to create and configure build job.

# Creating and Configuring build job for Java application with Maven

Now, let's perform steps to create and configure a new build job. Go to Jenkins Dashboard and click on **New Item**.

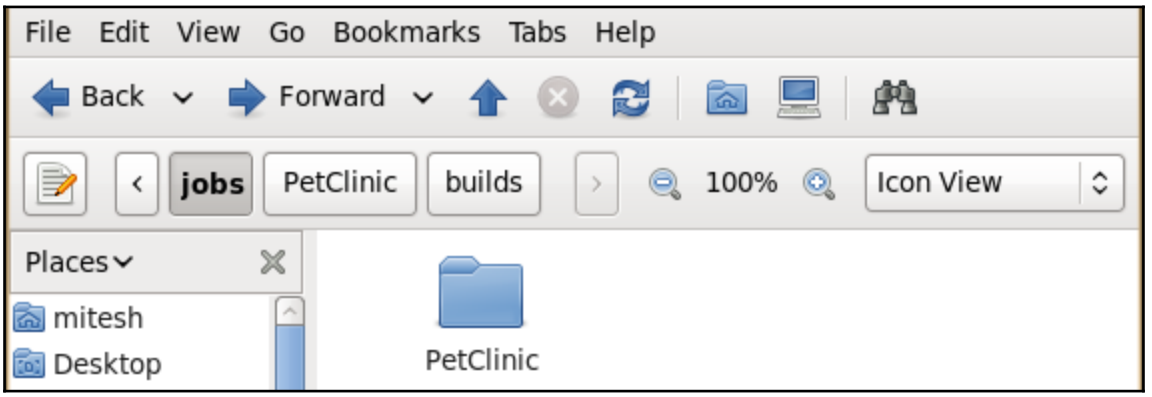
Go through all the options available of type of jobs we can create. In our case let's create a freestyle project for a demo purpose:

1. Enter an item name such as PetClinic.
2. Select **Freestyle project**.
3. Click on **OK**.




1. Let's verify what this operation does! Go to Jenkins home directory and navigate to **jobs** directory.

- We can see directory is created for newly created job with same name as shown below in the screenshot.




Next step is to configure source code repository with build job. We will use open source spring application that is hosted on GitHub as information is provided in Chapter 1,*Getting Started-DevOps Concepts, Tools, and Technology*.

1. Create a GitHub account and fork https://github.com/spring-projects/spring-petclinic.

2. After that we will get URL similar to: https://github.com/mitesh51/spring-petclinic.

- Install Git on virtual machine by using instruction available on Git documentation.

- Getting Started – Installing Git at: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

- Download for Windows at: https://git-scm.com/

3. Let's generate a new SSH key to use for authentication. Open terminal in CentOS virtual machine and make sure Git is installed in it.

4. Execute `ssh-keygen -t rsa -b 4096 -C "your_email@example.com"` command by substituting in GitHub email address.

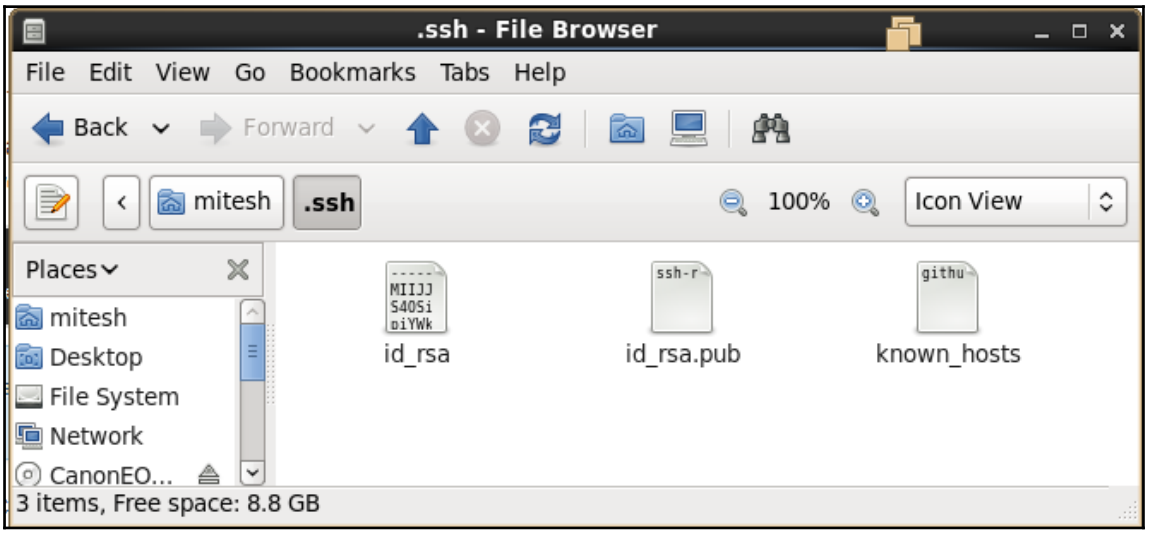
5. Press Enter when you are prompted for `Enter file in which to save the key`.

```
[mitesh@devops1 git]$ ssh-keygen -t rsa -b 4096 -C "[redacted]@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/mitesh/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mitesh/.ssh/id_rsa.
Your public key has been saved in /home/mitesh/.ssh/id_rsa.pub.
The key fingerprint is:
d5:48:73:9f:94:d8:02:32:75:5d:c8:08:da:33:2b:5d mitesh.soni83@gmail.com
The key's randomart image is:
+--[ RSA 4096]----+
|        o.*oo*.+.|
|         * *+o*. |
|        . * E.o  |
|         o =     |
|        S o      |
|         .       |
|                 |
|                 |
|                 |
+-----------------+
[mitesh@devops1 git]$
```

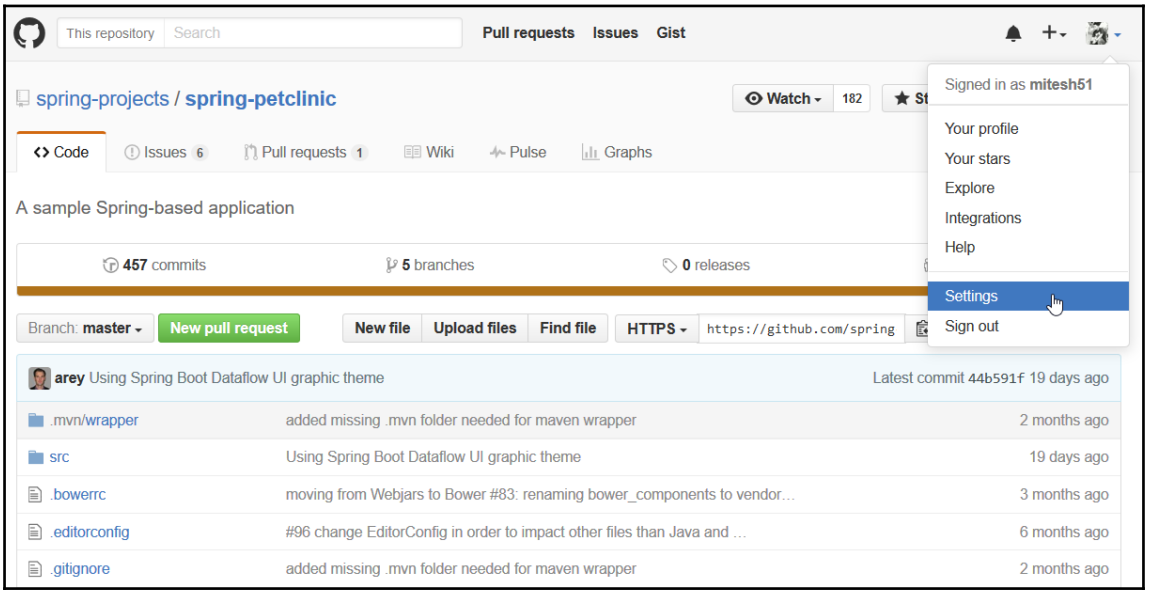
6. Add your SSH key to the ssh-agent.

```
[mitesh@devops1 git]$ ssh-add ~/.ssh/id_rsa
Identity added: /home/mitesh/.ssh/id_rsa (/home/mitesh/.ssh/id_rsa)
[mitesh@devops1 git]$
```

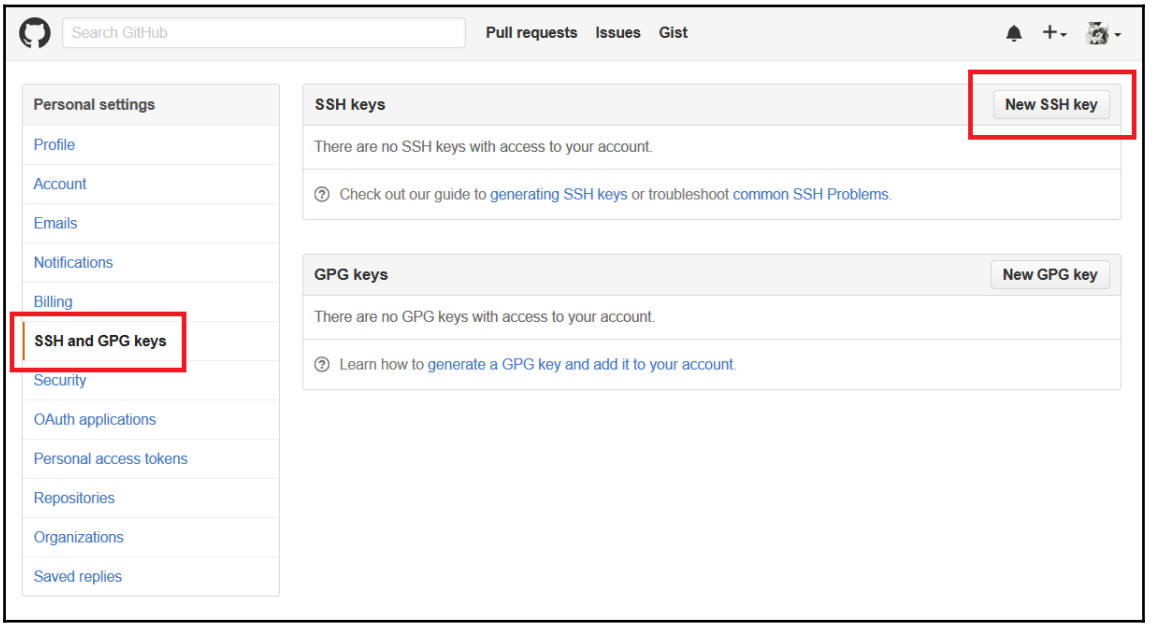
7. Verify newly generated keys in .ssh folder.



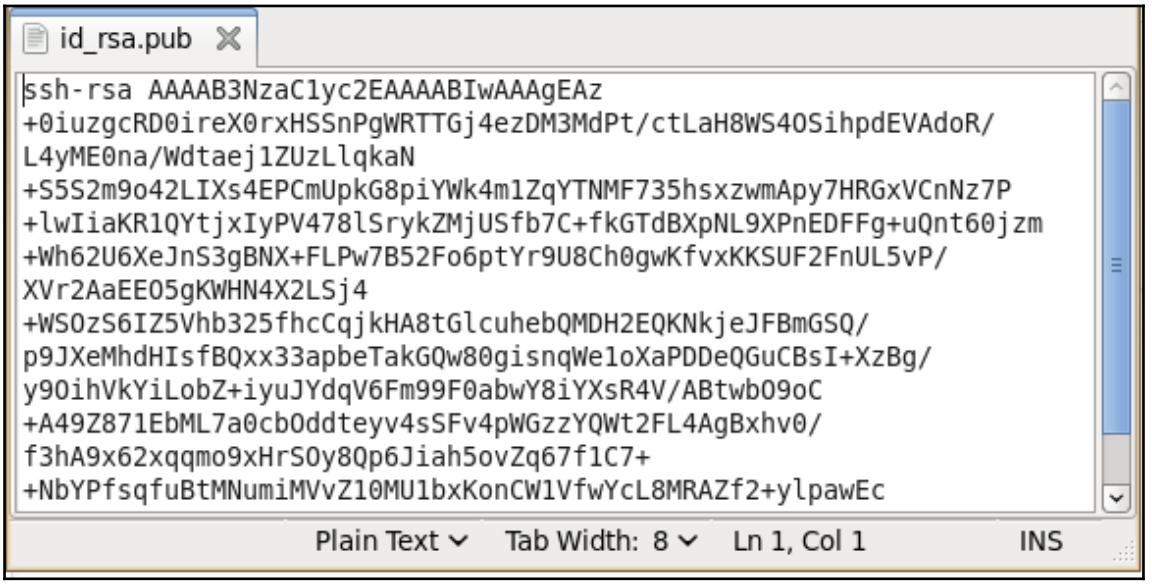

8. To configure GitHub account to use new SSH key, add it to your GitHub account. Go to https://github.com/mitesh51 and click on **Settings**.



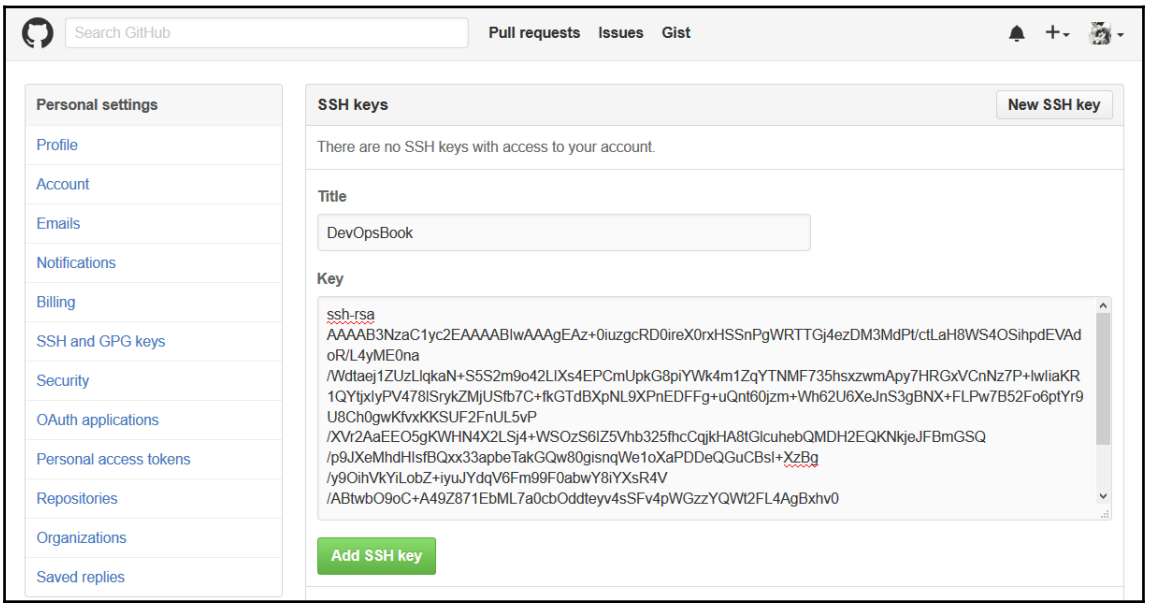

9. In the **Personal settings** sidebar, click **SSH and GPG keys**. Click **New SSH key.**



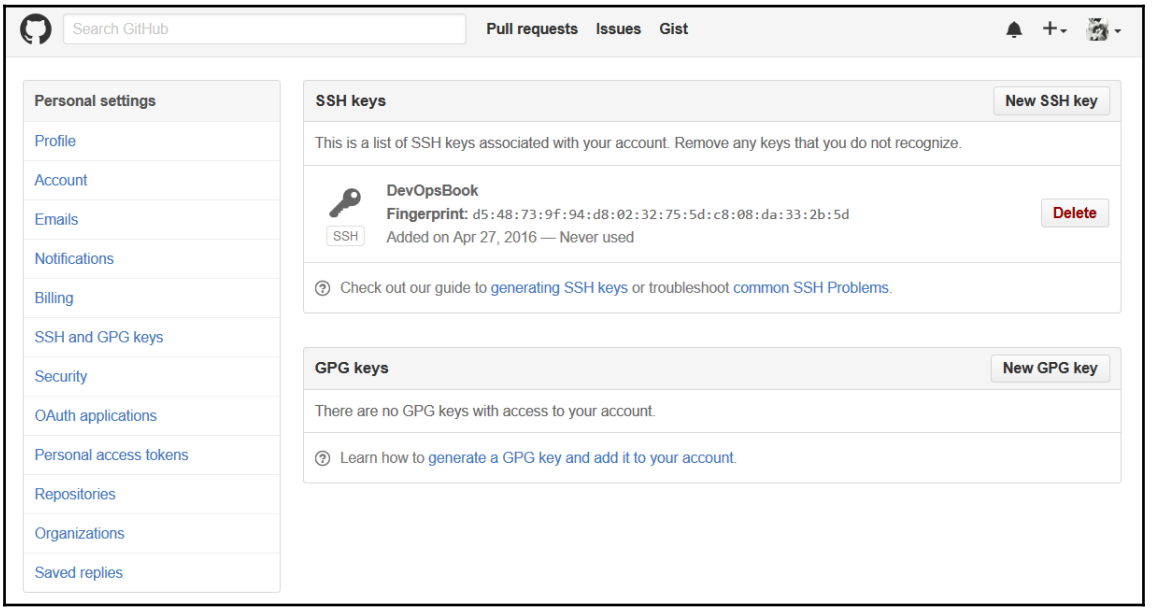

10. Open /.ssh/id_rsa.pub file in editor from CentOS virtual machine and copy the content.




11. In the **Title** field, add a descriptive label for the new key and paste the copied key content. Click on the **Add SSH key** as shown below:



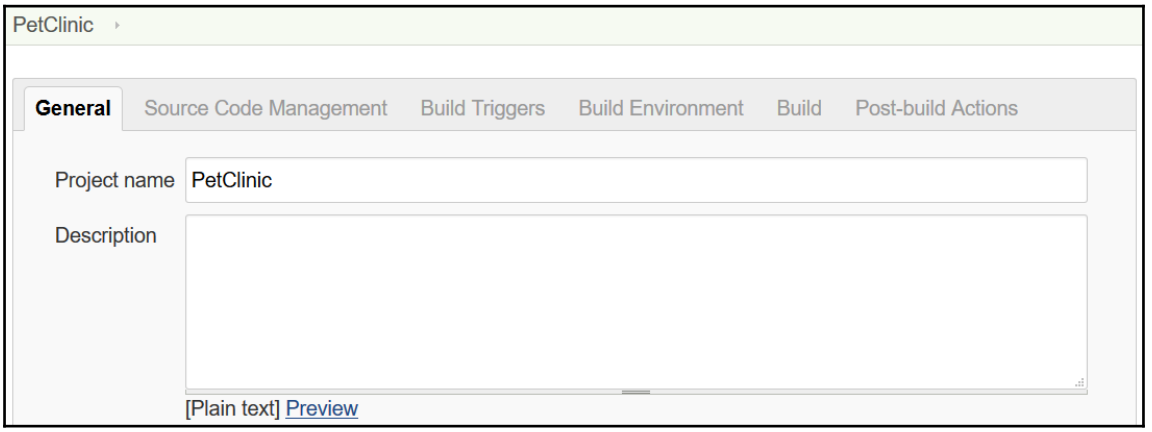

12. Verify the added SSH Key.



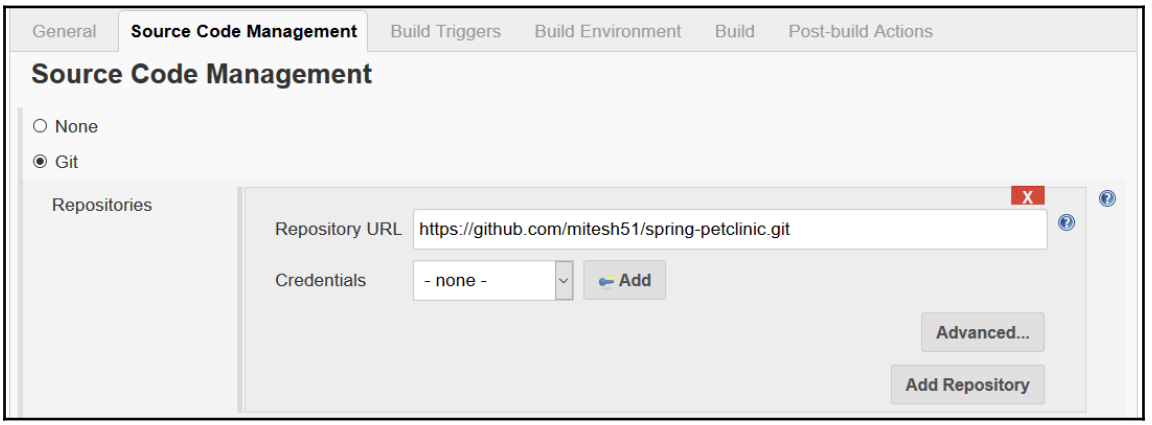
13. Now, let's verify authentication.


Once Git authentication is done, let's configure PetClinic build job.

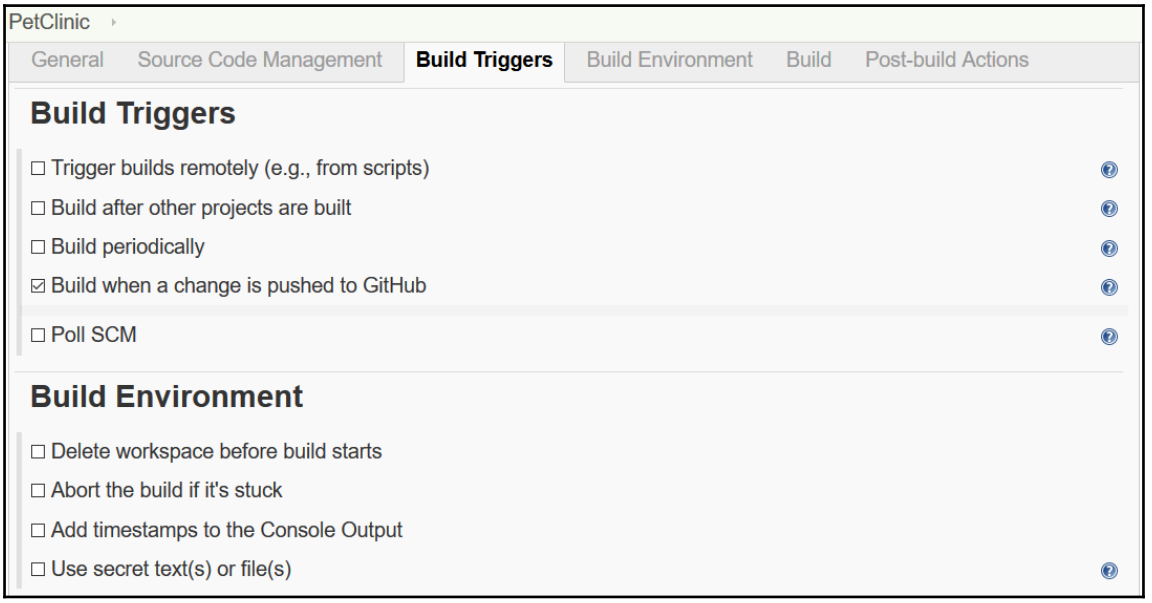
1. Click on the PetClinic build job in Jenkins dashboard. Click on **Configure** link of a PetClinic build job.



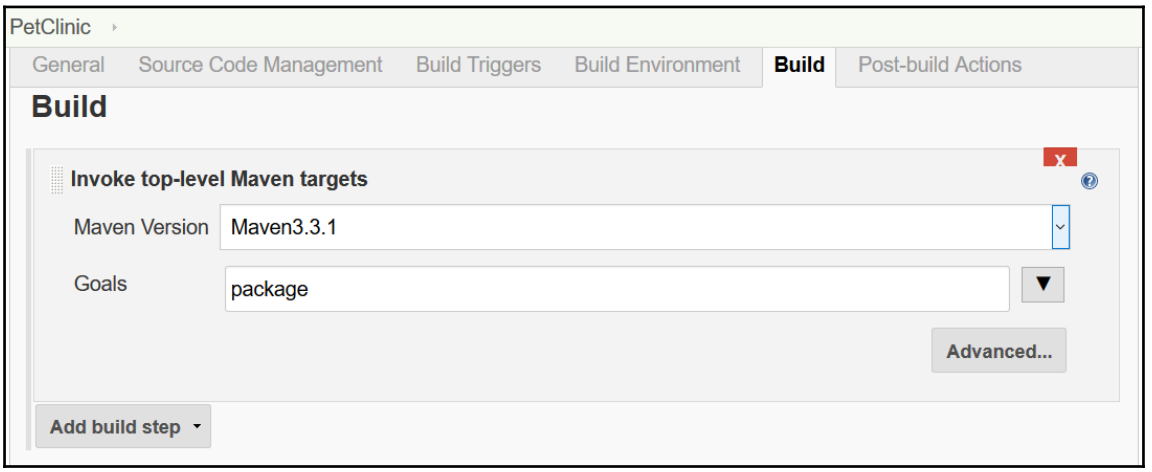

2. In **Source Code Management** provide Git URL for the Sample spring project we forked as shown below:



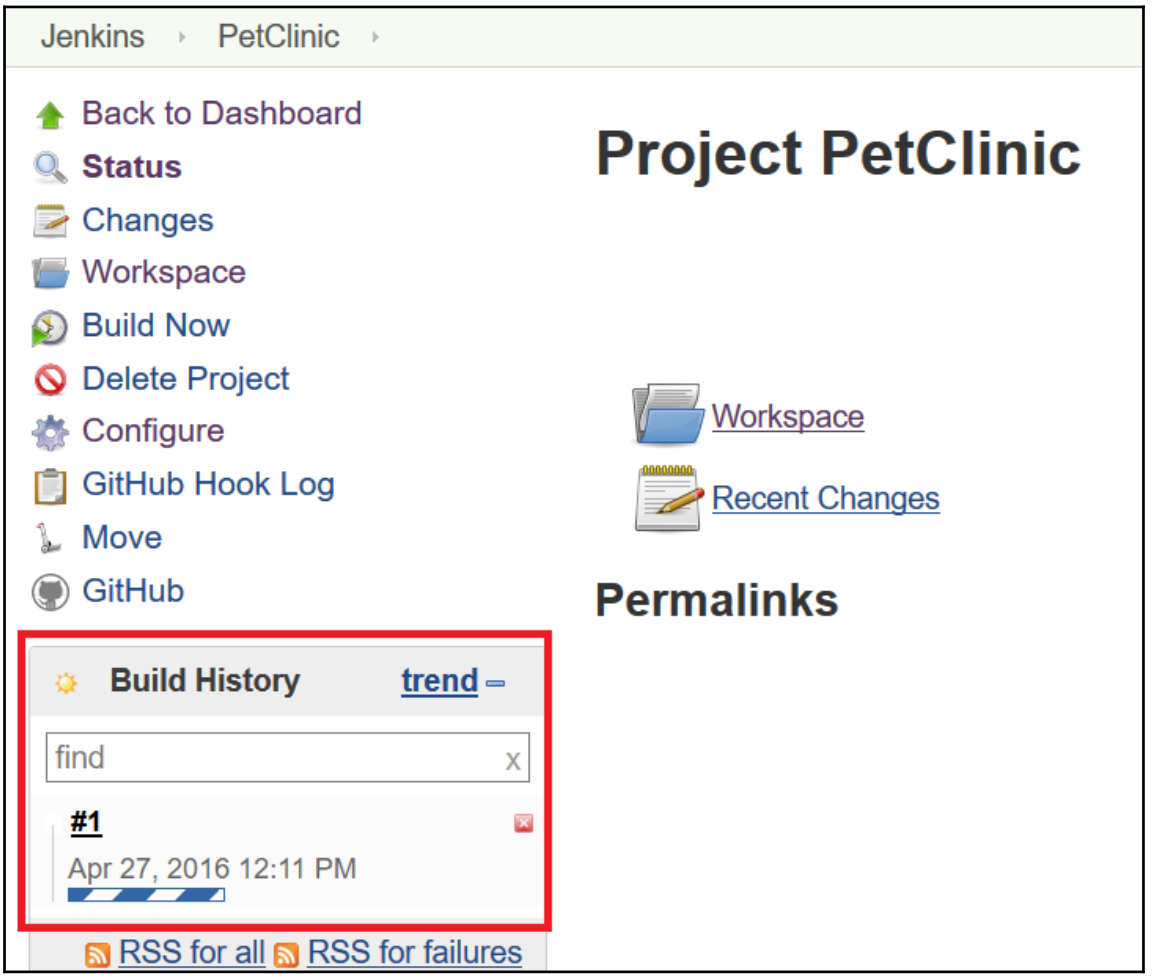

3. We will Configure **Build Triggers** and **Build Environment** as shown:



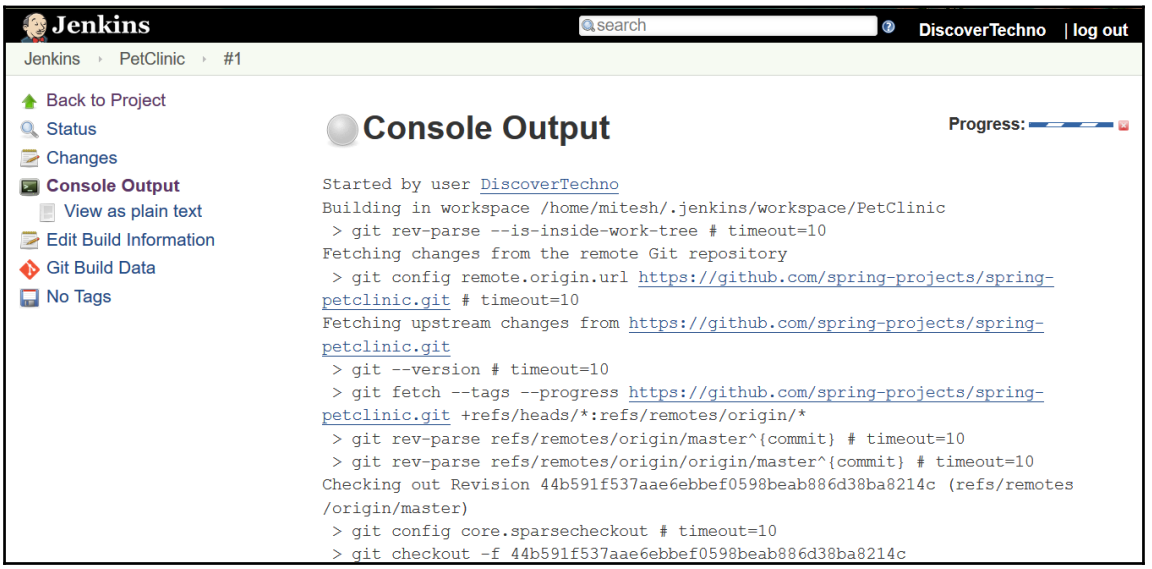

4. Click on **Add build step** and select **Invoke top-level Maven targets**. Select Maven version we configured in **Global Tools Configuration**. Enter Maven target and Click on **Save**.




5. Let's manually trigger the build by clicking on **Build Now**.



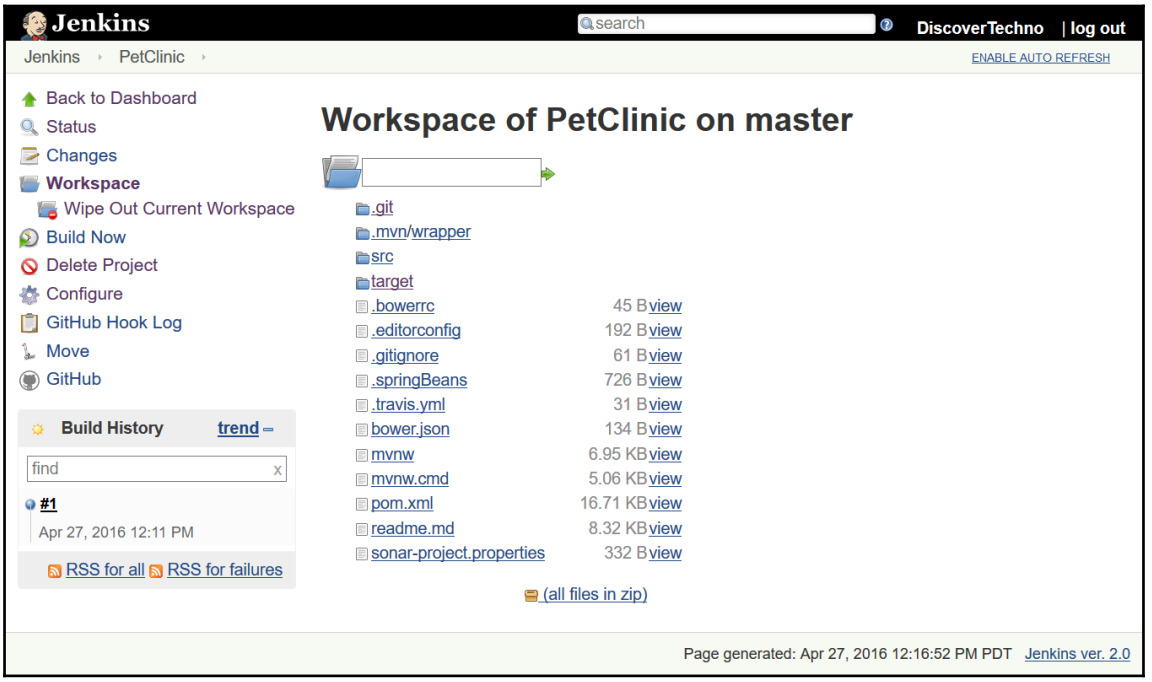

6. Click on the Build number with # sign. Open **Console Output**. Verify the Git operations executing before Maven target execution.




7. Once source code is available in the build job's workspace, Maven target will be executed and war file will be created. Verify the build status.

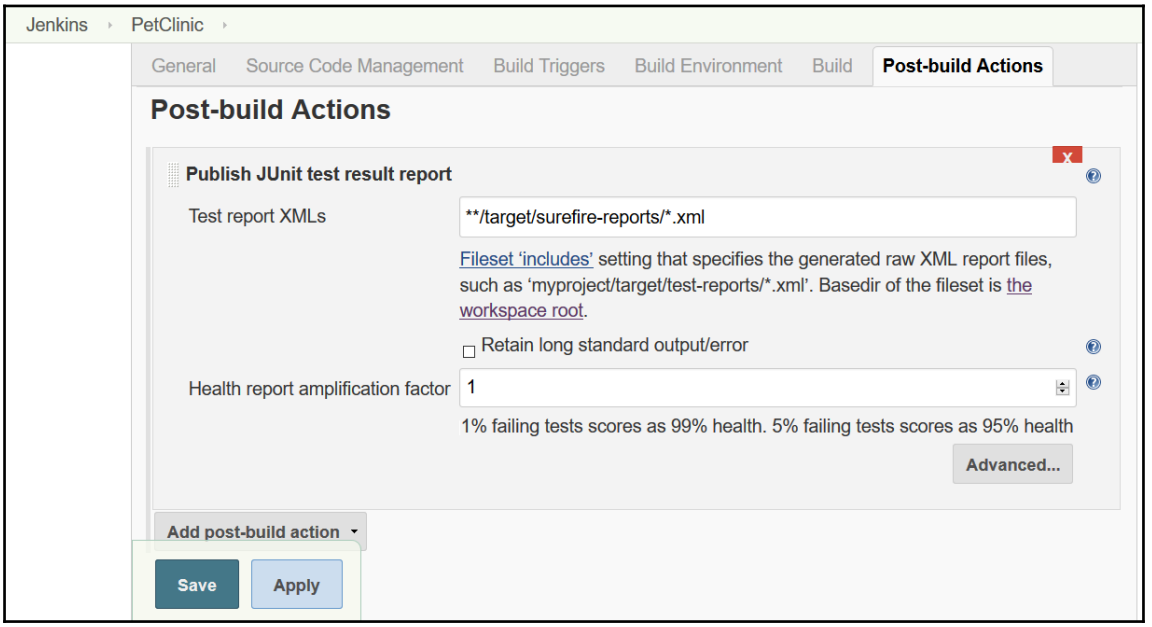

8. To verify the workspace of a build Job, click on the **Workspace** link. Verify all the

files available in the workspace. We can find these files in .jenkins folder under specific build job.



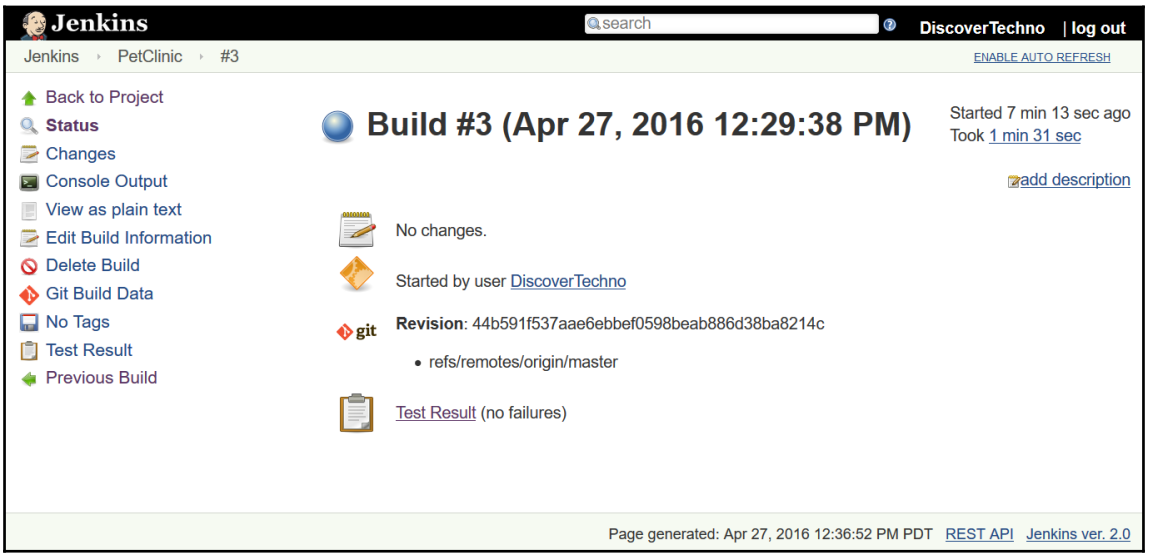

Our sample application has JUnit test cases and to execute them, we need to configure JUnit related settings in build job configuration.

1. In **Post-build Actions**, select **Publish JUnit test result report**.
2. Provide path for **Test Report XMLs** based on the workspace.
3. Click on **Apply** and then click on **Save**.

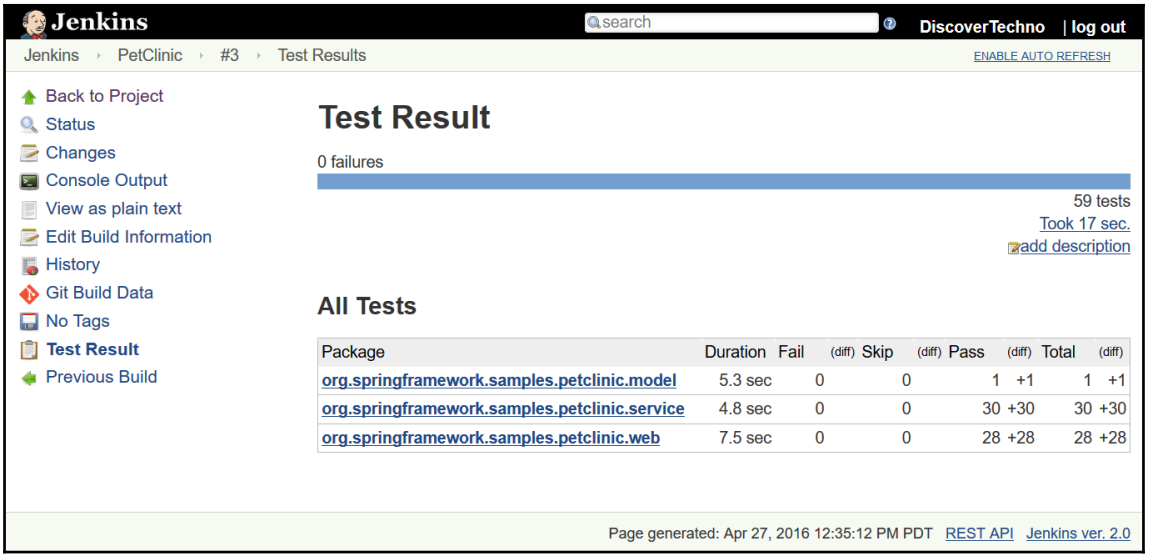



4. Once we configure JUnit settings in a build, wait for the build execution based on scheduling or click on the **Build Now** link.

5. Verify the build status on Jenkins dashboard and you will see **Test Result** link with small summary. Click on the **Test Result** link.




6. Verify all test execution status package wise. It also provides information related to duration, failed test cases.



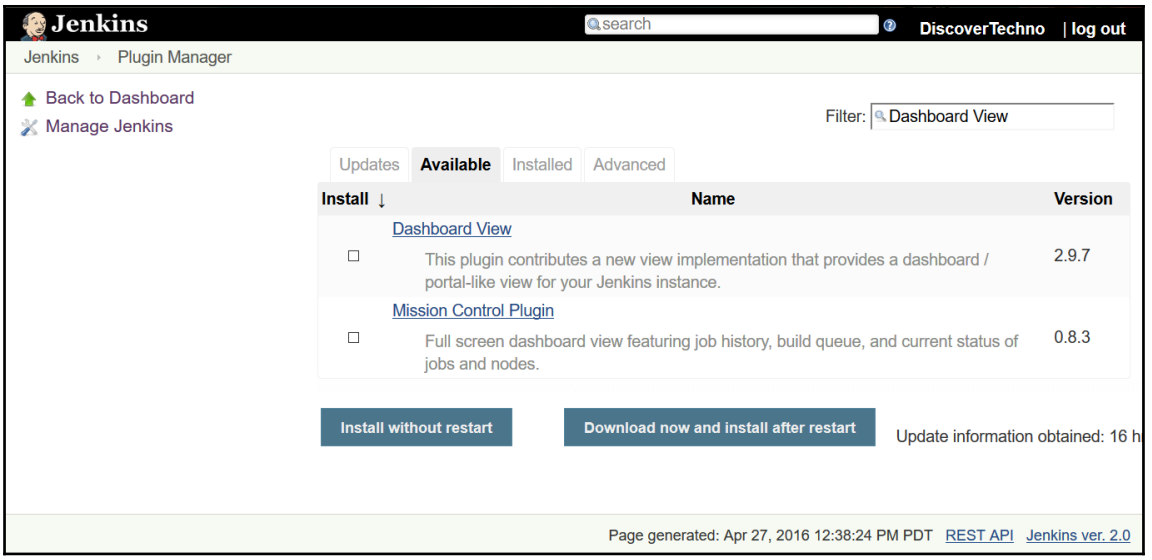

In the next section we will cover Dashboard View plugin to customize view for build jobs.

# Dashboard view plugin – overview and usage

**Dashboard View** plugin provides different view implementation considering portal kind of layout. We can select different build jobs to be included in new view and configure different portlets for view.

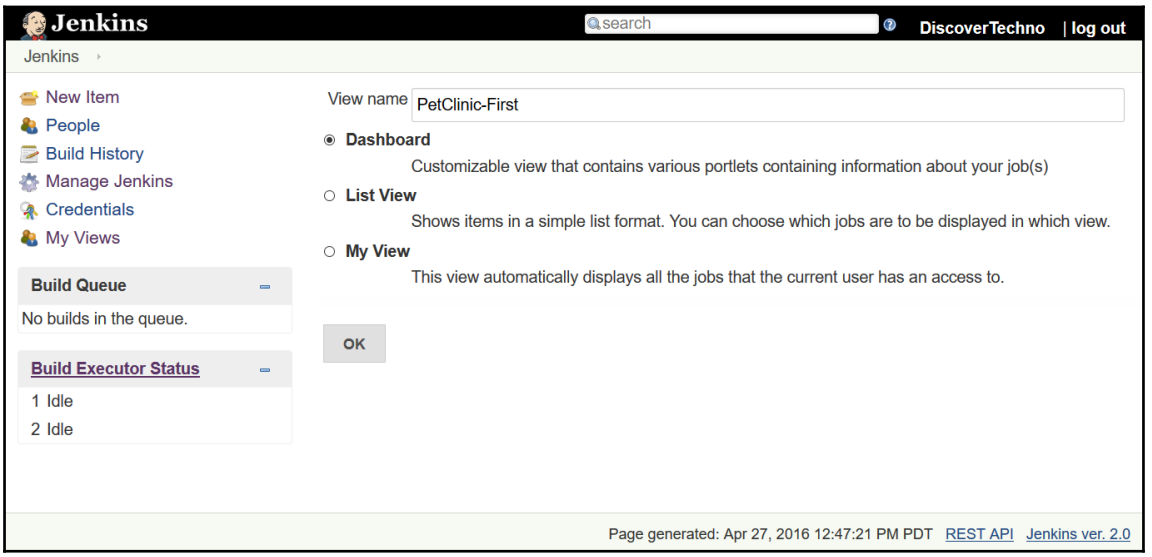
To configure:

1. Go to **Plugin Manager** from **Manage Jenkins**, and click on the **Available** tab. Search for Dashboard View plugin and click **Install without restart**.




2. Once installation of Dashboard View plugin is completed successfully, we can create a new view by clicking on the + sign on Jenkins dashboard.

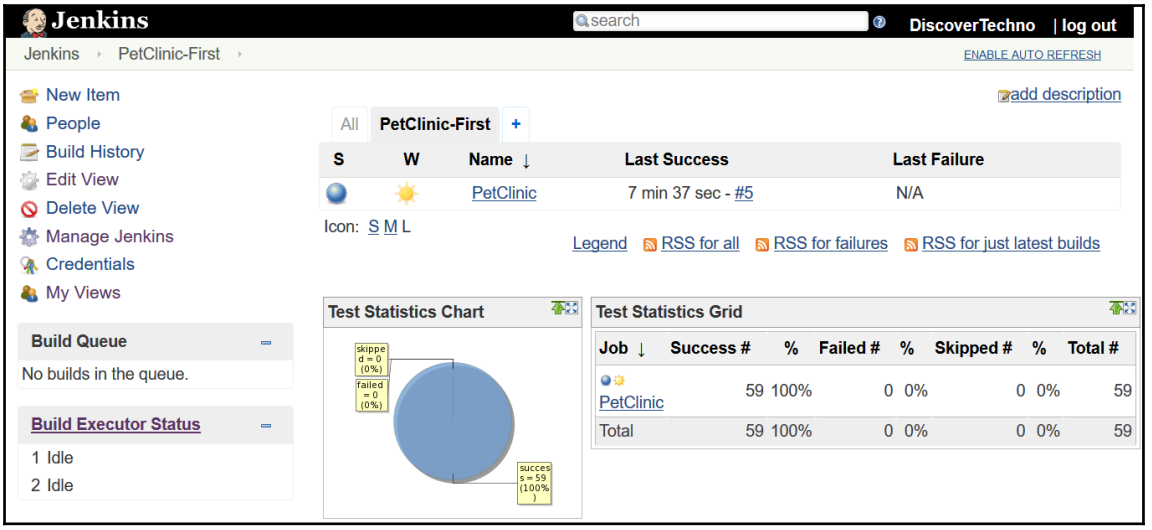
3. Enter **View name**, select view type and click on **OK**.



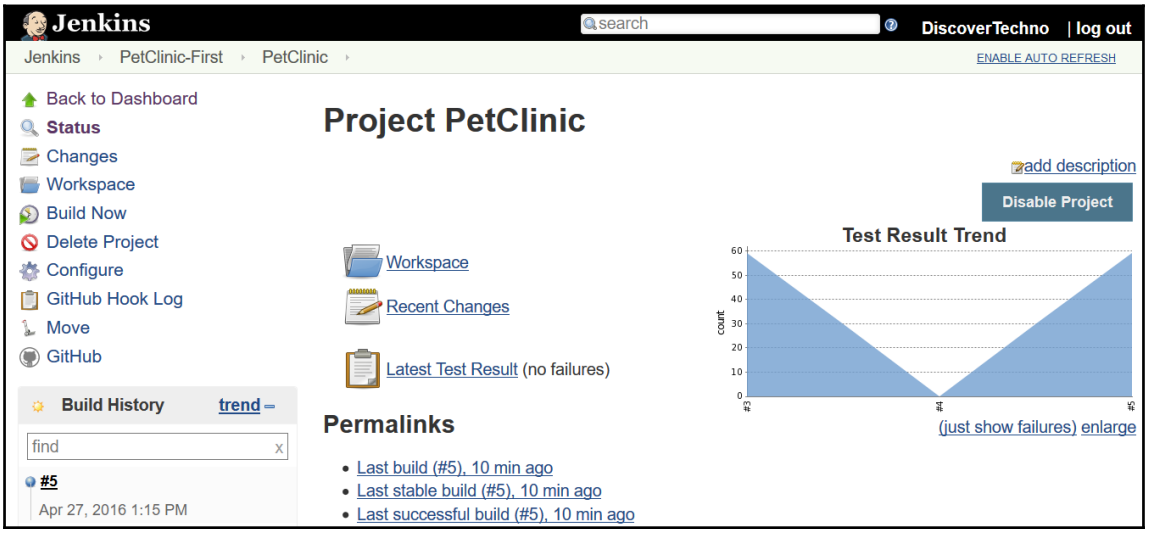

4. Click on **Edit** and configure **Dashboard Portlets** for top view, left column, right column, and bottom view. We can use different portlets such as **Test Statistics Chart**, **Trends**, and so on.


5. Add different portlets based on needs into the view and save it. Sample view is

given in the below screenshot:




6. Once we run the build job, we can find test result chart on the build job's dashboard as well.




In the next section, one of the most popular feature of Jenkins and that is distributed builds.

Consider a scenario where you want different Java applications that need different kind of JDK version to compile source files?

How to manage such situation in an effective manner? We will see answers in next section.

# Managing Nodes

Jenkins provides Master-Slave concept to manage above mentioned scenarios. We can assign different build jobs to different slaves in build configuration and Master-Slave manage its overall lifecycle. Master node itself can execute the build if slave node is not configured explicitly in the build job configuration.

There are quite a few reasons why we should use this feature of Jenkins:

- Build job execution requires resources and they compete for resource availability.
- Different runtime environment for different build jobs.
- To distribute the load across slave nodes.

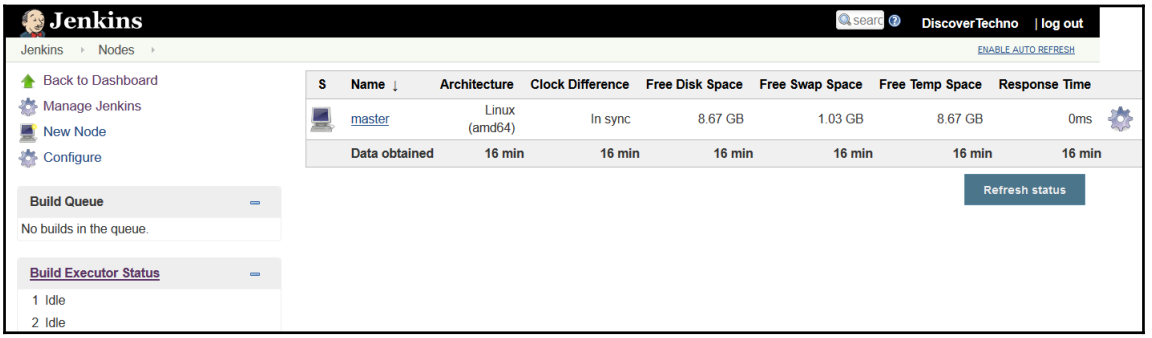
To make things more clear, we need not to install Jenkins in the slave nodes. We only need to configure slave node properly which we will demonstrate in this section.

The only requirements are:

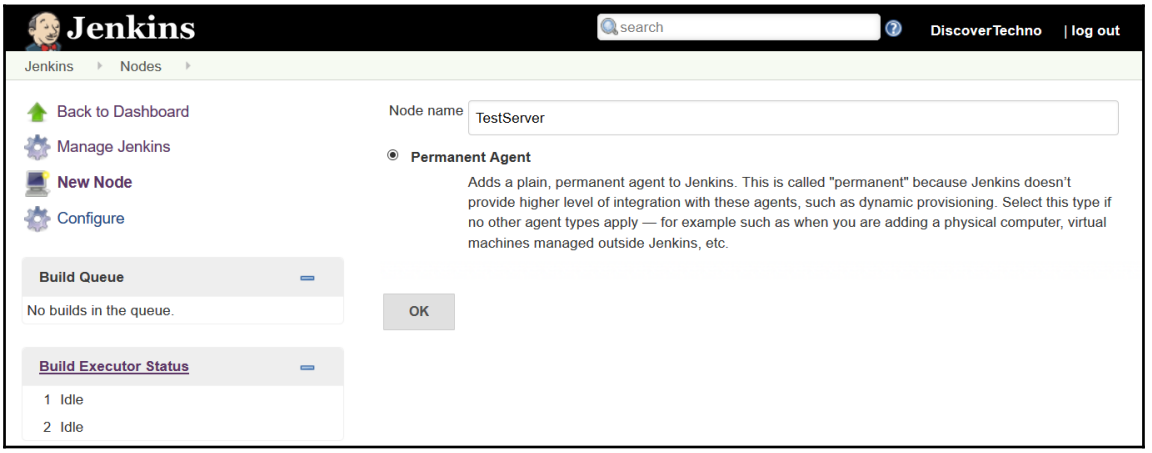
- Configurations and Runtime Environment has to be available on the slave node.
- Path needs to be configured correctly on Master node for Runtime. Environments or tools used by slave node for execution.

To create a slave node in Jenkins 2:

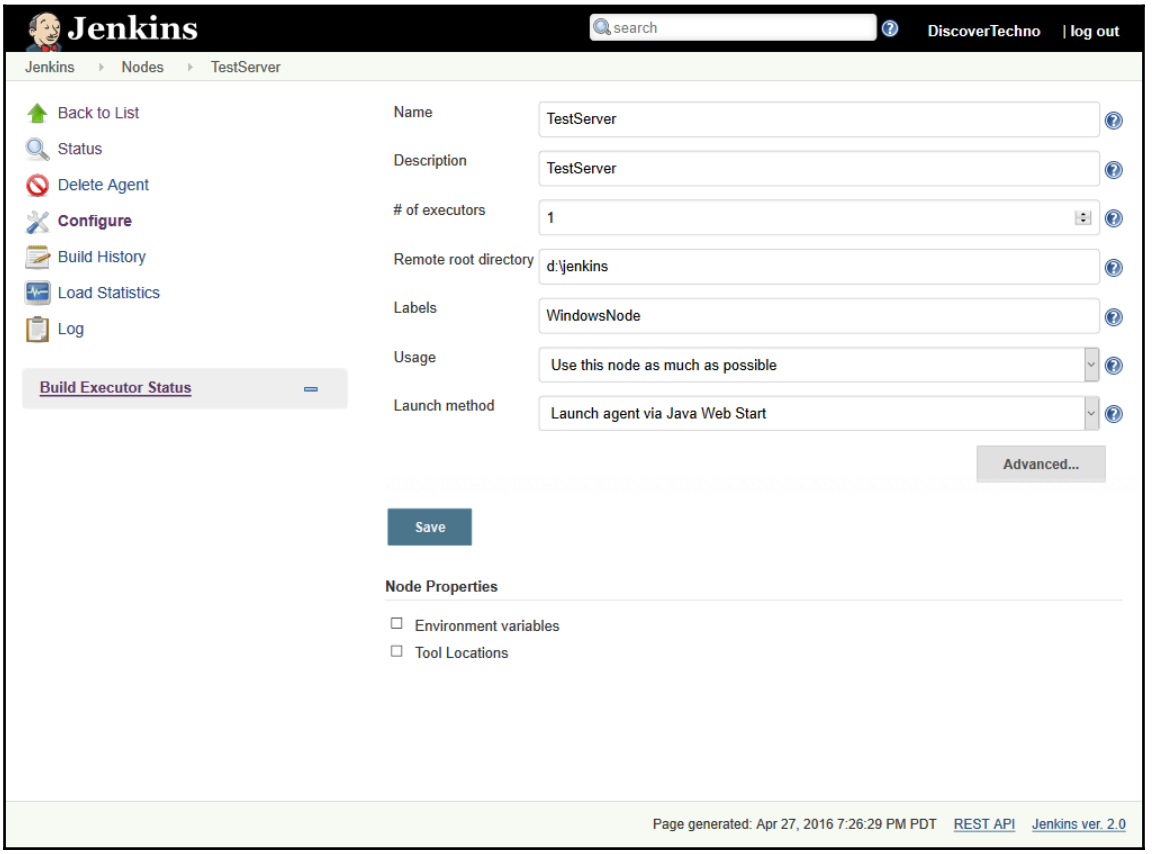
1. Click on **Manage Jenkins** link on Jenkins dashboard.



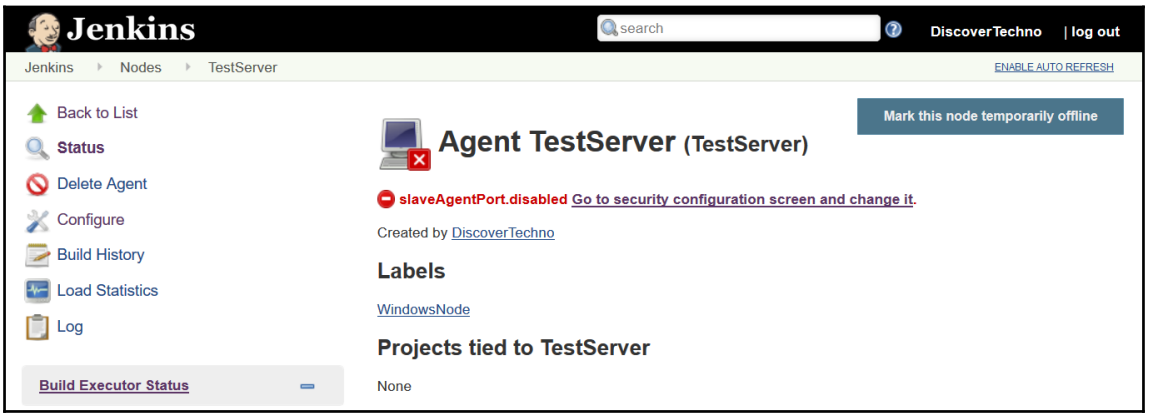

2. Verify that only Master node's entry is available. To add a new node, click on **New Node** in left sidebar. Enter node name in **Node name** field and click on **OK**.




3. Next step is to configure the newly created node. Enter **Remote root directory** that will store details related to build jobs on slave node. Give **Labels** to this node. Labels can be used to assign different build jobs to specific slave machine.

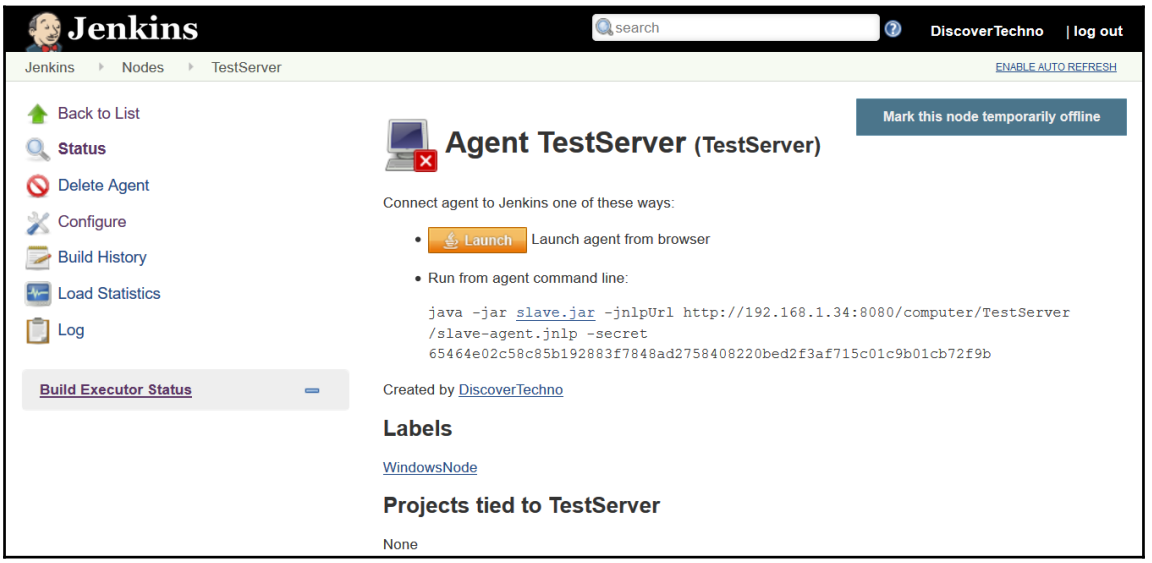



4. In Jenkins 2, after creating slave node and configuring it, if there is an error **slaveAgentPort.disabled** as shown in figure below, then we need to first solve it and then perform the further steps.

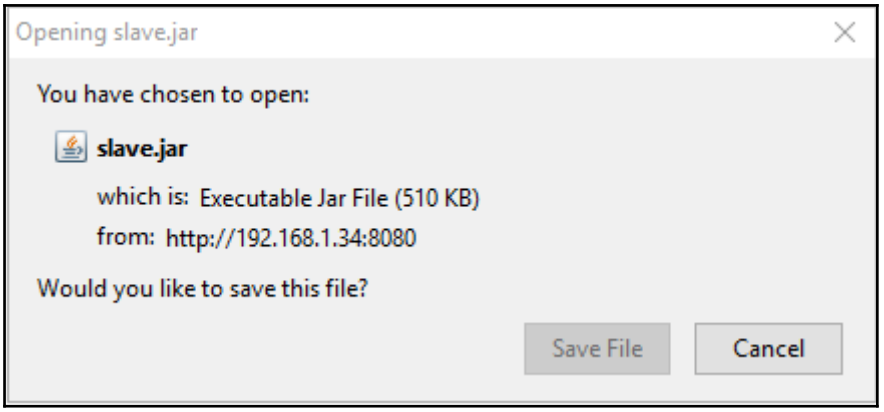



5. Go to **Manage Jenkins** page, and click on **Configure Global Security** link. Select **Enable security** and select **Fixed** or **RandomTCP port for JNLP agents** and save the configuration.


6. Next step is to connect Jenkins slave with Jenkins Master. We will connect agent to Jenkins by using command line.

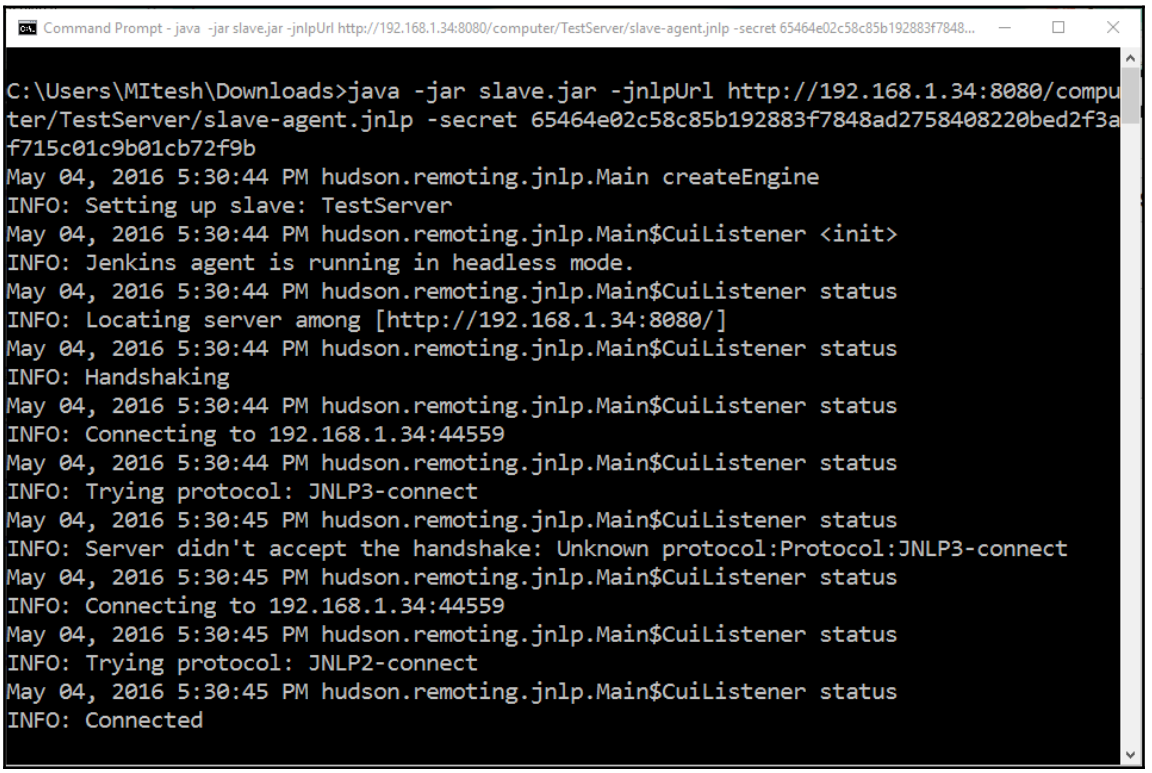



7. Download the slave.jar file and put it on slave node.



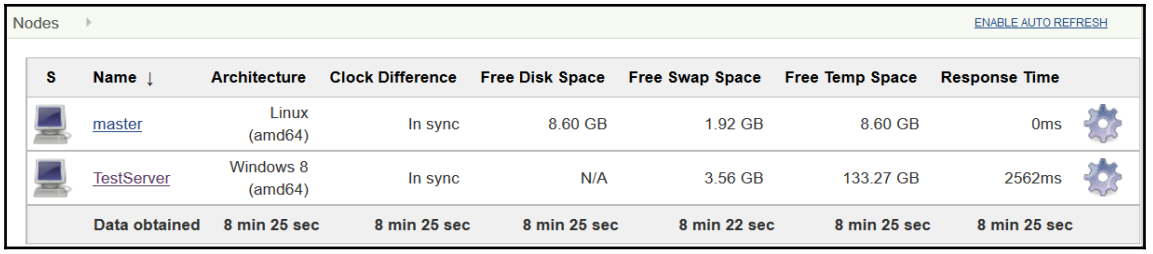

8. Execute following code in the terminal or command prompt based on the operating systems on the slave node:

**java -jar slave.jar -jnlpUrl http://192.168.1.34:8080/computer/TestServer/slave-agent.jnlp -secret 65464e02c58c85b192883f7848ad2758408220bed2f3af715c01c9b01cb72f9b**



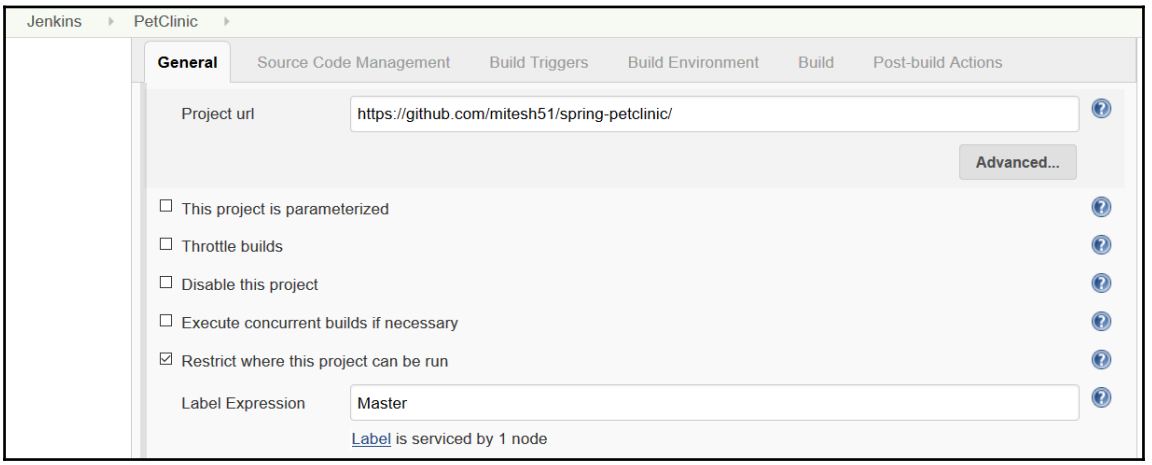

9. Verify the status of slave node in the Jenkins dashboard.


10. Now, we can see two nodes in Jenkins dashboard.

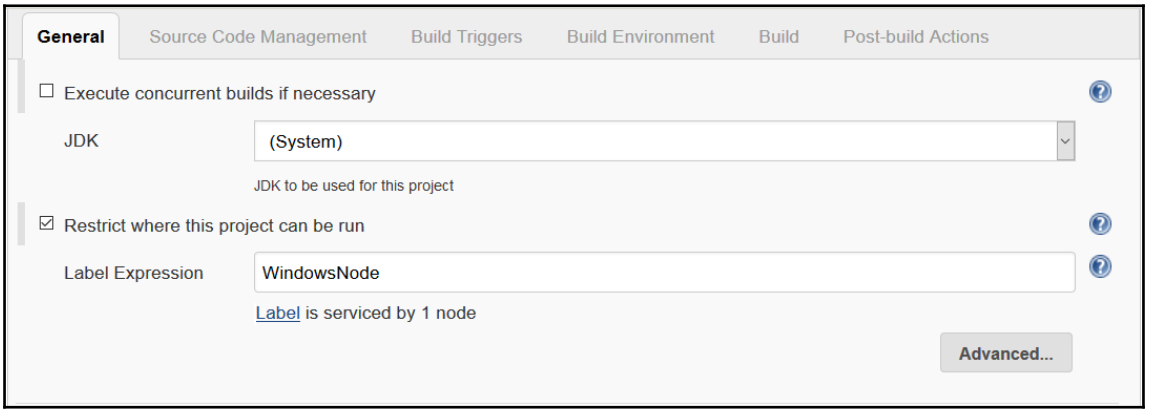



11. To configure build job to run on master, open build configuration and in **General** section select **Restrict where this project can be run**.

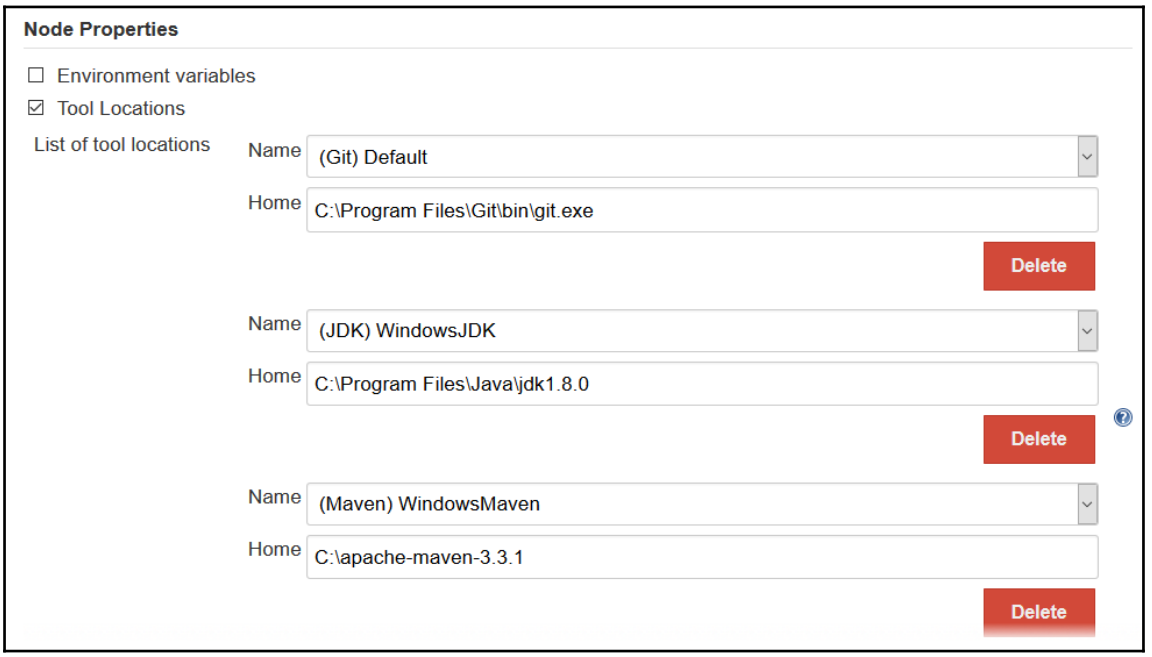
12. In **Label Expression**, enter label of the master node.




13. To configure build job to run on slave node, enter label of slave node in **Label Expression**. We can also configure **JDK** or other required path for build execution.




14. To configure tools specific to slave node, click on **Configure** in **Manage Nodes** section. In **Node Properties**, configure **Tool Locations** for slave node as shown in below image:




In the next section, we will see how to configure email notifications.

# Email notifications based on build status

> *"Failure is simply the opportunity to begin again, this time more intelligently."—Henry Ford*

However, it is extremely vital to be aware about failure or at least to know when things fail to fix it and remove issues.

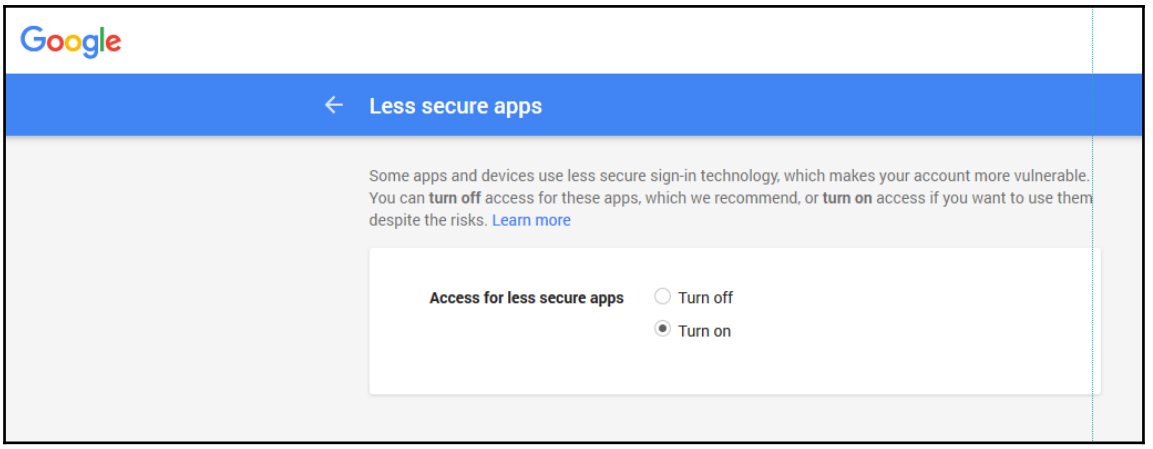
Notifications are always helpful in case of failures. Consider a scenario where build failure or test case failure has to be notified to specific set of stakeholders. In such situation it is desirable to have email notifications.

We will use Gmail configuration for setting up email notifications.

To make things work,

1. Go to: https://www.google.com/settings/u/1/security/lesssecureapps and

**Turn onAccess for less secure apps** as shown below to send email notifications from Jenkins 2.




2. In Jenkins dashboard:

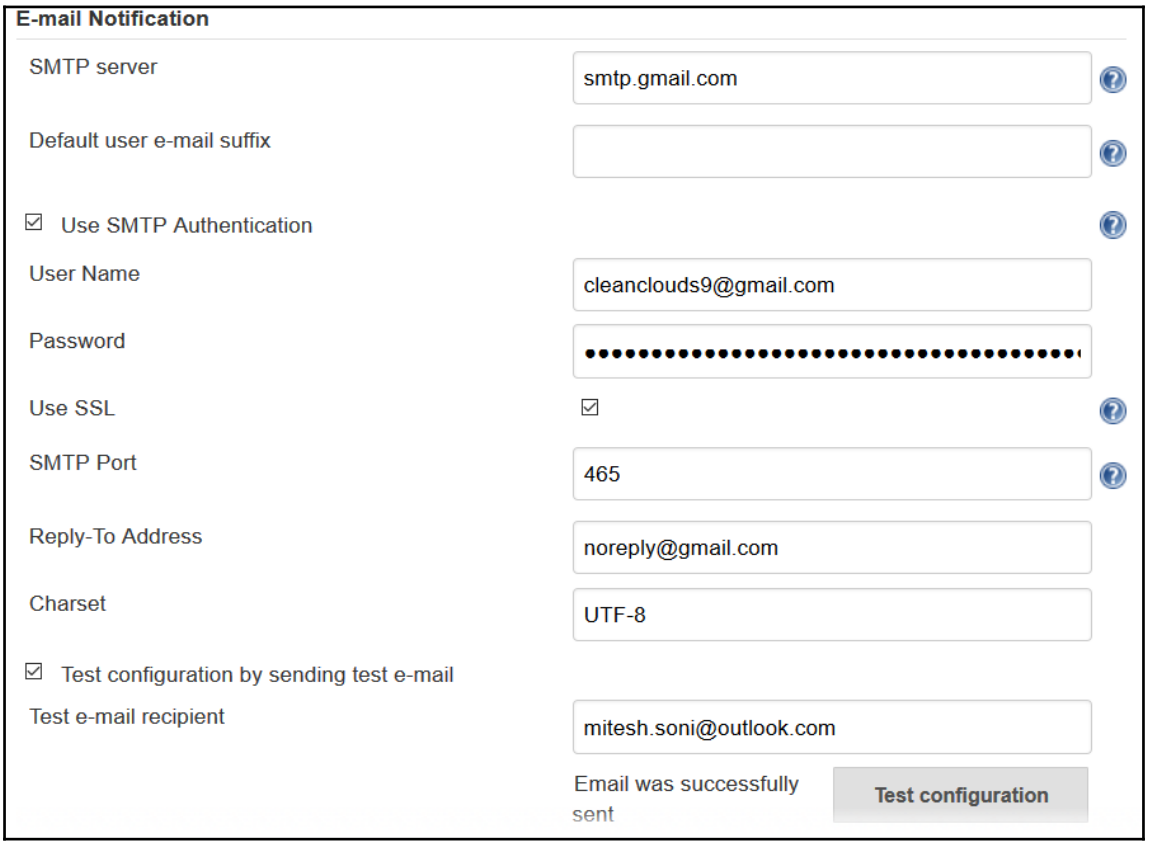
    1. Click on **Manage Jenkins** and go to **Configure System** section.

    2. Go to **E-mail Notification** sub section and enter values for **SMTP Server** and **Default user e-mail suffix**.

    3. Select **Use SMTP Authentication** checkbox, enter **User Name** and **Password**.

    4. Select **Use SSL** checkbox, enter **SMTP Port**, **Reply-To Address**.

    5. Finally select **Test configuration by sending test e-mail**. If Email configurations are correct then you will find a message **Email was successfully sent**.



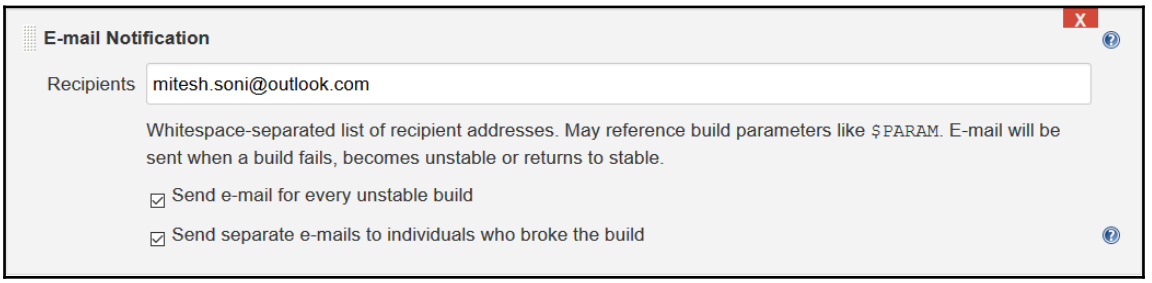

6. To verify Email notifications, simulate failure in one of the build job. Open any build job and click on **Configure**.

7. In **Post-build** Actions, click on **Add post-build** action

3. Select **E-mail Notification**.

4. Enter list of **Recipients**.

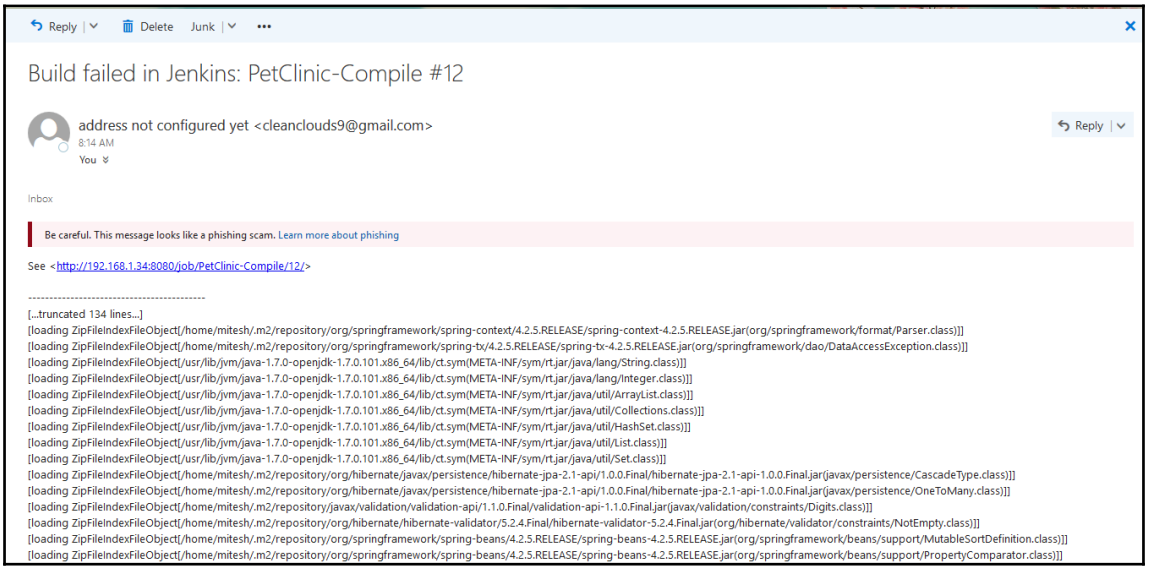
5. Select **Send e-mail for every unstable build** and **Send separate e-mails to individuals who broke the build**.




In our case, we execute compile goal against Maven build and we wanted to publish JUnit Test result to simulate failure. We can see that compilation of files are successful but Post-build action fails and it triggers Email notification based on the configuration.

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 21.332 s
[INFO] Finished at: 2016-04-28T19:44:26-07:00
[INFO] Final Memory: 25M/134M
[INFO] ------------------------------------------------------------------------
Recording test results
ERROR: Step 'Publish JUnit test result report' failed: No test report files were
found. Configuration error?
Sending e-mails to: mitesh.soni@outlook.com
Finished: FAILURE
```

Following is the Email received from Jenkins build job failure. It contains stack trace of the execution.




Let's consider a scenario where we want to send customized content in the mail. How to achieve that?

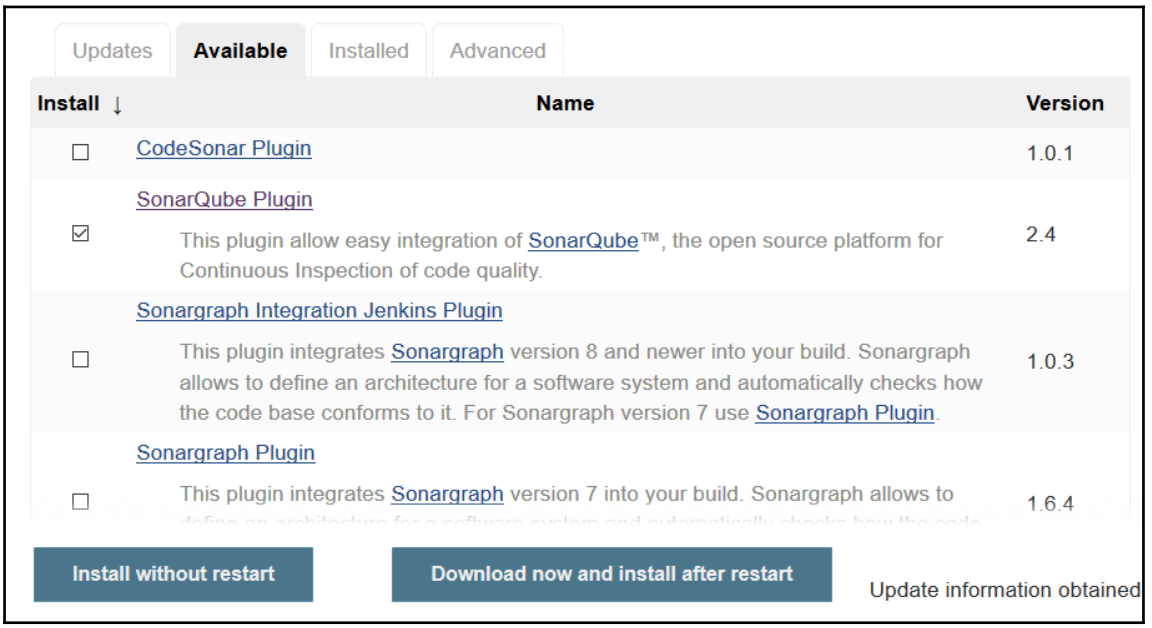
Hint: Configure Extended E-mail Notification. Try it as an exercise.

# Jenkins and Sonar integration

SonarQube is an open source tool to manage code quality of an application. It manages seven axes of code quality such as Architecture & Design, Duplications, Unit Tests, Potential Bugs, Complexities, Coding Rules, and Comments. It covers programming languages such as ABAP, C/C++, C#, COBOL, CSS, Erlang, Flex / ActionScript, Groovy, Java, Java Properties, JavaScript, JSON, Objective-C, PHP, PL/I, PL/SQL, Puppet, Python, RPG, Swift, VB.NET, Visual Basic 6, Web, and XML. One of the striking feature is its extensibility. It is easy to cover new languages and adding rules engines using extension mechanism called plugins.

To install, SonarQube plugin,

1. Go to **Manage Jenkins**, click on **Manage Plugins**. Click on **Available** tab. Search SonarQube plugin and install it by clicking on **Install without restart**.



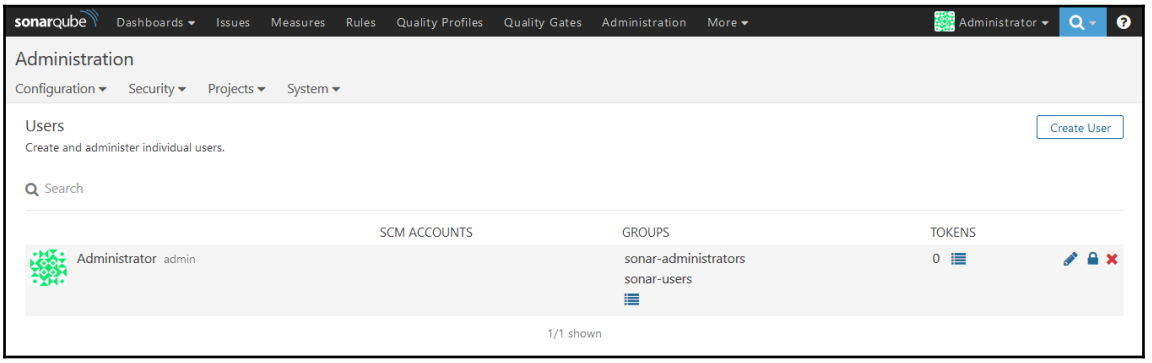

2. Download sonar from http://www.sonarqube.org/downloads/.

3. Extract installable directory from the zip file and go to bin sub-directory.

4. Based on the operating system, select the installable directory and run the StartSona.* file as shown in below image.

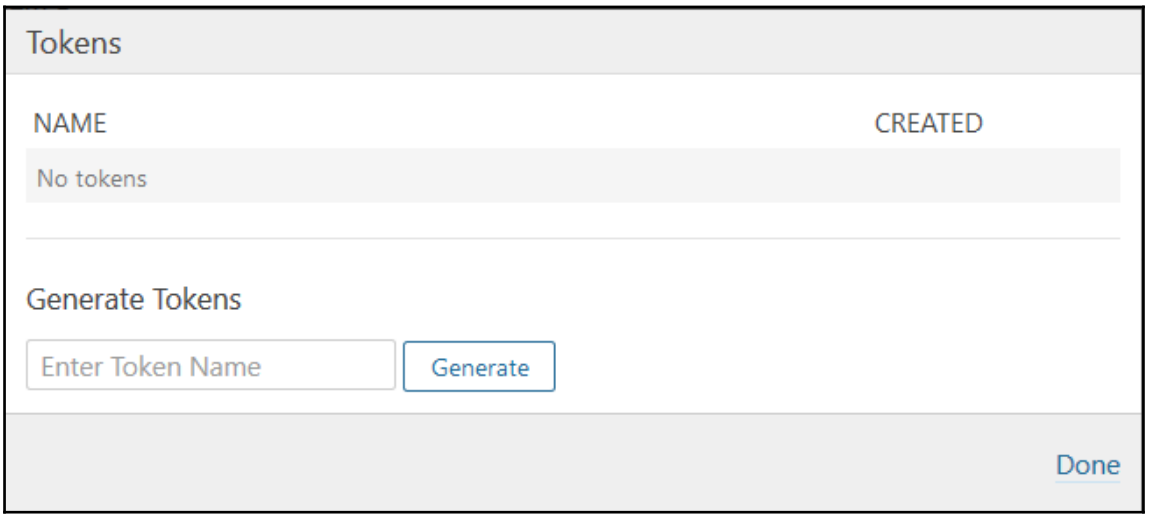
```
D:\##DevOps Book\Installables\sonarqube-5.4\bin\windows-x86-64>StartSonar.bat
wrapper  | --> Wrapper Started as Console
wrapper  | Launching a JVM...
jvm 1    | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org
jvm 1    |   Copyright 1999-2006 Tanuki Software, Inc.  All Rights Reserved.
jvm 1    |
jvm 1    | 2016.04.29 23:57:37 INFO  app[o.s.a.AppFileSystem] Cleaning or creating temp directory D:\##DevOps Book\Install
ables\sonarqube-5.4\temp
jvm 1    | 2016.04.29 23:57:38 INFO  app[o.s.p.m.JavaProcessLauncher] Launch process[search]: C:\Program Files\Java\jre1.8
.0_45\bin\java -Djava.awt.headless=true -Xmx1G -Xms256m -Xss256k -Djava.net.preferIPv4Stack=true -XX:+UseParNewGC -XX:+Use
ConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -XX:+UseCMSInitiatingOccupancyOnly -XX:+HeapDumpOnOutOfMemoryError -
Djava.io.tmpdir=D:\##DevOps Book\Installables\sonarqube-5.4\temp -cp ./lib/common/*;./lib/search/* org.sonar.search.Search
Server C:\Users\MItesh\AppData\Local\Temp\sq-process7000722619322287622properties
jvm 1    | 2016.04.29 23:57:49 INFO  app[o.s.p.m.Monitor] Process[search] is up
jvm 1    | 2016.04.29 23:57:49 INFO  app[o.s.p.m.JavaProcessLauncher] Launch process[web]: C:\Program Files\Java\jre1.8.0_
45\bin\java -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djruby.management.enabled=false -Djruby.compile.invokedynamic=
false -Xmx768m -Xms256m -XX:MaxPermSize=160m -XX:+HeapDumpOnOutOfMemoryError -Djava.net.preferIPv4Stack=true -Djava.io.tmp
dir=D:\##DevOps Book\Installables\sonarqube-5.4\temp -cp ./lib/common/*;./lib/server/*;D:\##DevOps Book\Installables\sonar
qube-5.4\lib\jdbc\h2\h2-1.3.176.jar org.sonar.server.app.WebServer C:\Users\MItesh\AppData\Local\Temp\sq-process3019138822
364693273properties
jvm 1    | 2016.04.29 23:59:07 INFO  app[o.s.p.m.Monitor] Process[web] is up
```

5. Once Sonar is up and running, Open browser and visit http://localhost:9000/ or http://<IP_Address>:9000/. We will get the Sonar dashboard.

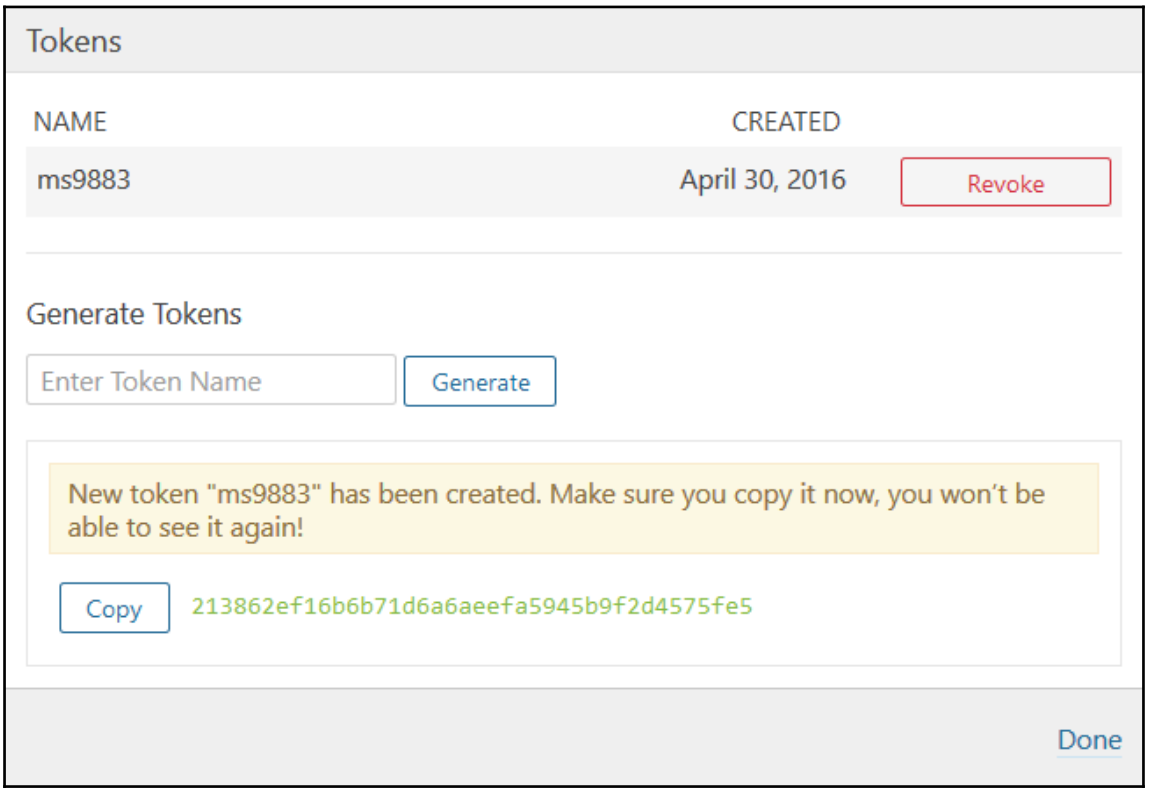



One of the important step for Jenkins 2 and Sonar integration is security token.

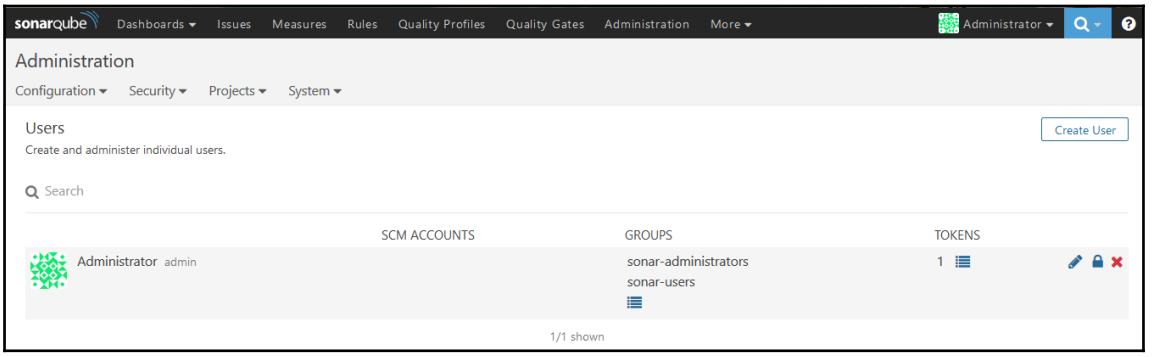
1. Go to **My Account** link on top right corner.
2. Click on **security** tab and **Generate Tokens**.




3. Enter Token name and click on **Generate**. Copy the token value and click on **Done**.

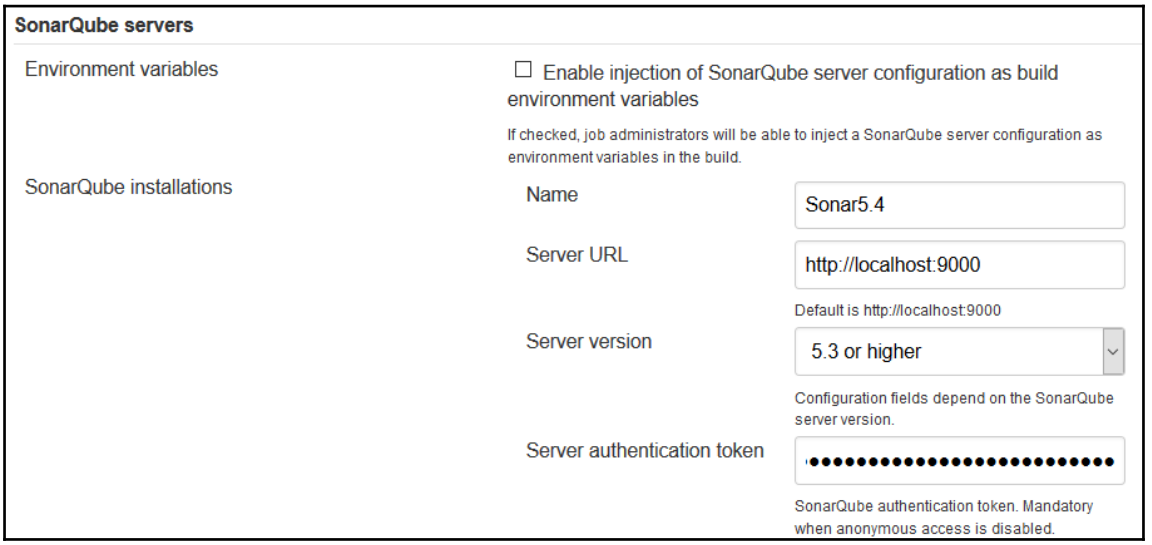



4. Verify the Tokens column in the Sonar dashboard.

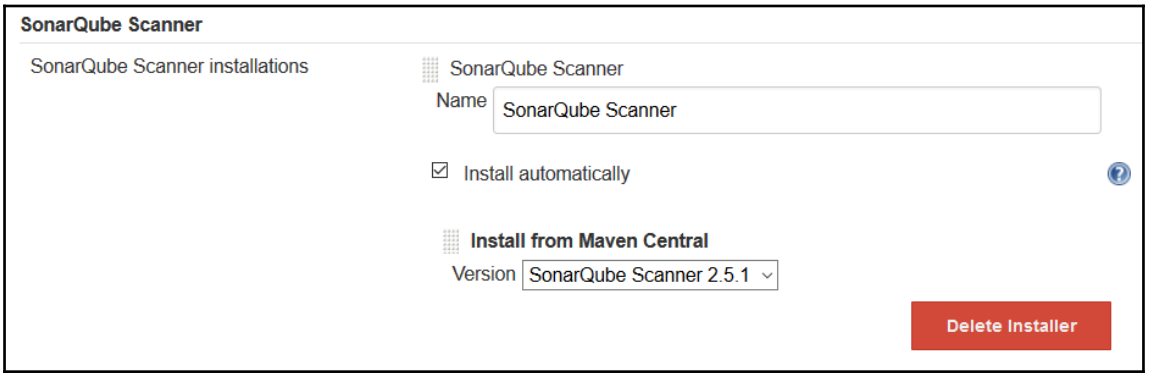



Once we have a security token ready, next step is to integrate Jenkins and Sonar.

1. In **Manage Jenkins** section, click on **Configure System** and add **SonarQube servers**. Here provide **Server URL** and security token and save the settings.




2. In **Global Tool Configuration**, configure **SonarQube Scanner** installations also.




Once all Sonar related installations and configurations are completed, we need to add Build step to execute SonarQube Scanner. Run the build Job.

1. We need sonar-project.properties for Sonar configuration with specific application. In our sample application, sonar-project.properties file is already available.

```
# Required metadata
sonar.projectKey=java-sonar-runner-simple
sonar.projectName=Simple Java project analyzed with the SonarQube Runner
sonar.projectVersion=1.0

# Comma-separated paths to directories with sources (required)
sonar.sources=src

# Language
sonar.language=java

# Encoding of the source files
sonar.sourceEncoding=UTF-8
```

2. Verify the console output of a build job for Sonar execution.

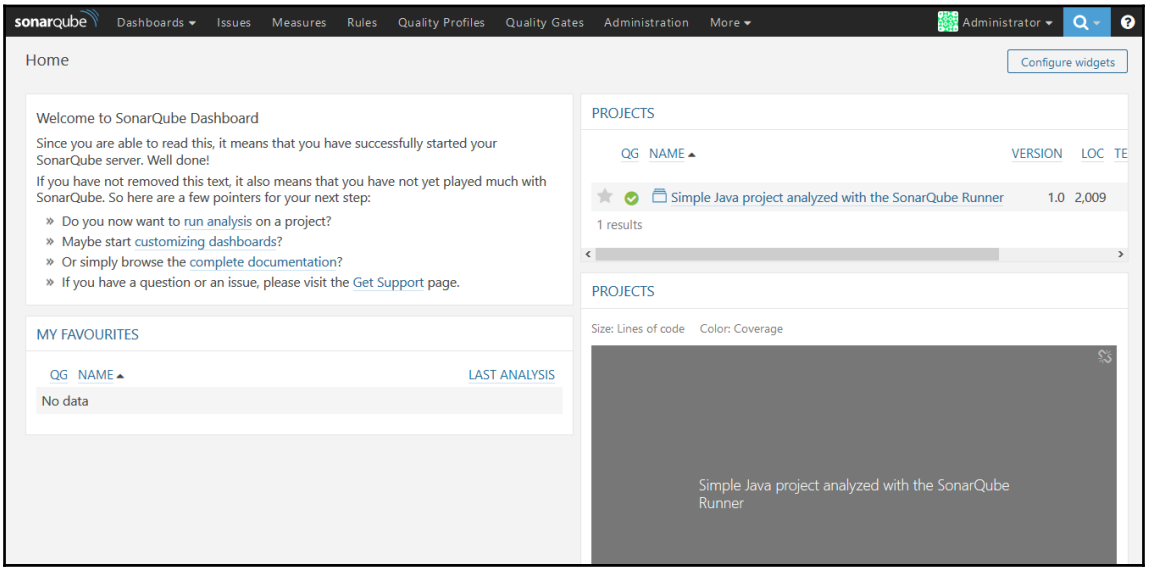
```
  D:\##DevOps Book\Installables\sonar-scanner-2.6
  INFO: Scanner configuration file: D:\##DevOps Book\Installables\sonar-
scanner-2.6\conf\sonar-scanner.properties
    INFO: Project root configuration file: d:\jenkins\workspace\PetClinic-Test\sonar-
project.properties
  INFO: SonarQube Scanner 2.6
  INFO: Java 1.8.0-ea Oracle Corporation (64-bit)
  INFO: Windows 8.1 6.3 amd64
  INFO: Error stacktraces are turned on.
  INFO: User cache: C:\Users\MItesh\.sonar\cache
  INFO: Load global repositories
  INFO: Load global repositories (done) | time=1131ms
  INFO: User cache: C:\Users\MItesh\.sonar\cache
  INFO: Load plugins index
  INFO: Load plugins index (done) | time=16ms
    INFO: Download sonar-csharp-plugin-4.4.jar
    INFO: Download sonar-java-plugin-3.10.jar
    INFO: Download sonar-scm-git-plugin-1.0.jar
    INFO: Download sonar-scm-svn-plugin-1.2.jar
    INFO: Download sonar-javascript-plugin-2.10.jar
    INFO: SonarQube server 5.4
  INFO: Default locale: "en_US", source code encoding: "UTF-8"
  INFO: Process project properties
  INFO: Load project repositories
  INFO: Load project repositories (done) | time=133ms
  INFO: Apply project exclusions
  INFO: Load quality profiles
  INFO: Load quality profiles (done) | time=927ms
  INFO: Load active rules
  INFO: Load active rules (done) | time=4068ms
  INFO: Publish mode
    INFO: ------------  Scan Simple Java project analyzed with the SonarQube Runner
```

```
    INFO: Language is forced to java
    INFO: Load server rules
    INFO: Load server rules (done) | time=656ms
    INFO: Base dir: d:\jenkins\workspace\PetClinic-Test
      INFO: Working dir: d:\jenkins\workspace\PetClinic-Test\.sonar
      INFO: Source paths: src
      INFO: Source encoding: UTF-8, default locale: en_US
      INFO: Index files
      INFO: 56 files indexed
    INFO: Quality profile for java: Sonar way
    INFO: JaCoCoSensor: JaCoCo report not found : d:\jenkins\workspace\PetClinic-
Test\target\jacoco.exec
    INFO: JaCoCoItSensor: JaCoCo IT report not found: d:\jenkins\workspace\PetClinic-
Test\target\jacoco-it.exec
    INFO: Sensor JavaSquidSensor
    INFO: Configured Java source version (sonar.java.source): none
    INFO: JavaClasspath initialization...
    INFO: Bytecode of dependencies was not provided for analysis of source files, you might
end up with less precise results. Bytecode can be provided using sonar.java.libraries property
    INFO: JavaClasspath initialization done: 1 ms
    INFO: JavaTestClasspath initialization...
    INFO: Bytecode of dependencies was not provided for analysis of test files, you might end
up with less precise results. Bytecode can be provided using sonar.java.test.libraries property
    INFO: JavaTestClasspath initialization done: 1 ms
      INFO: Java Main Files AST scan...
      INFO: 56 source files to be analyzed
    INFO: 46/56 files analyzed, current file: d:\jenkins\workspace\PetClinic-
Test\src\test\java\org\springframework\samples\petclinic\service\AbstractClinicServiceTests.ja
va
    INFO: Java Main Files AST scan done: 12107 ms
    INFO: Java bytecode has not been made available to the analyzer. The
org.sonar.java.bytecode.visitor.DependenciesVisitor@4f1150f5,
org.sonar.java.checks.unused.UnusedPrivateMethodCheck@3fba233d are disabled.
    INFO: Java Test Files AST scan...
    INFO: 0 source files to be analyzed
    INFO: Java Test Files AST scan done: 1 ms
    INFO: Sensor JavaSquidSensor (done) | time=15295ms
    INFO: Sensor Lines Sensor
      INFO: 56/56 source files have been analyzed
    INFO: 0/0 source files have been analyzed
    INFO: Sensor Lines Sensor (done) | time=28ms
    INFO: Sensor QProfileSensor
    INFO: Sensor QProfileSensor (done) | time=29ms
    INFO: Sensor SurefireSensor
    INFO: parsing d:\jenkins\workspace\PetClinic-Test\target\surefire-reports
    INFO: Sensor SurefireSensor (done) | time=531ms
    INFO: Sensor SCM Sensor
    INFO: SCM provider for this project is: git
```

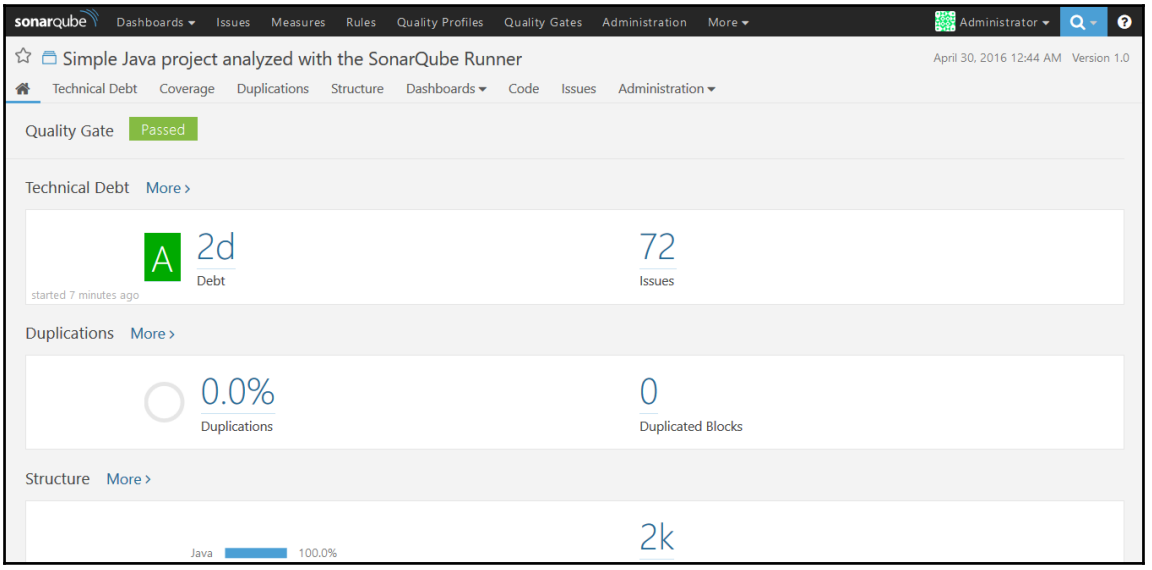
```
INFO: 56 files to be analyzed
INFO: 56/56 files analyzed
INFO: Sensor SCM Sensor (done) | time=3754ms
INFO: Sensor Code Colorizer Sensor
INFO: Sensor Code Colorizer Sensor (done) | time=9ms
INFO: Sensor CPD Sensor
INFO: JavaCpdIndexer is used for java
INFO: Sensor CPD Sensor (done) | time=303ms
INFO: Analysis report generated in 1055ms, dir size=294 KB
  INFO: Analysis reports compressed in 629ms, zip size=191 KB
  INFO: Analysis report uploaded in 524ms
  INFO: ANALYSIS SUCCESSFUL, you can browse
    http://localhost:9000/dashboard/index/java-sonar-runner-simple
  INFO: Note that you will be able to access the updated dashboard once the server has
processed the submitted analysis report
  INFO: More about the report processing at
    http://localhost:9000/api/ce/task?id=AVRjchhfszI1jSgY1AZe
INFO: ------------------------------------------------------------------------
  INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 57.737s
INFO: Final Memory: 52M/514M
INFO: ------------------------------------------------------------------------
  Recording test results
  Finished: SUCCESS
```

3. Let's verify the Sonar UI at
http://localhost:9000/dashboard/index/java-sonar-runner-simple

4. In the **Projects** section, we can find project details available now. Click on the project name.

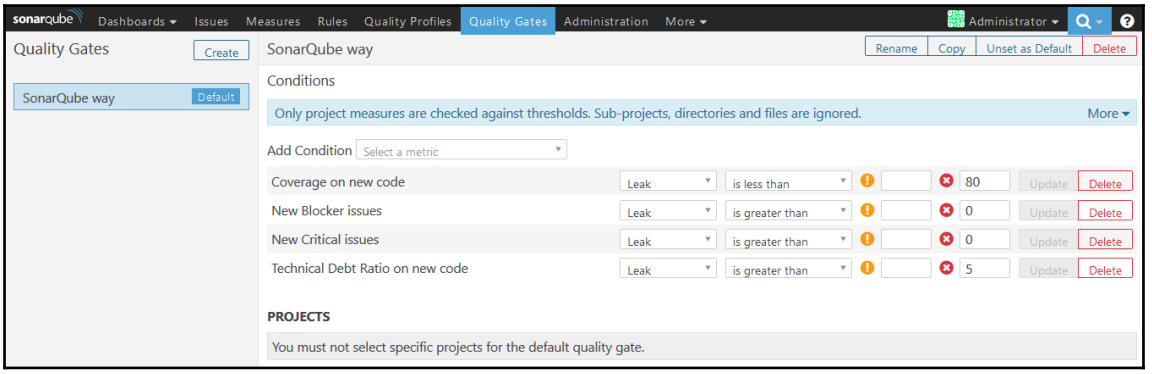



5. We can see the result of analysis here. Quality Gate is passed. It provides details about **Technical Debt**, **Duplications**, **and Structure** too.

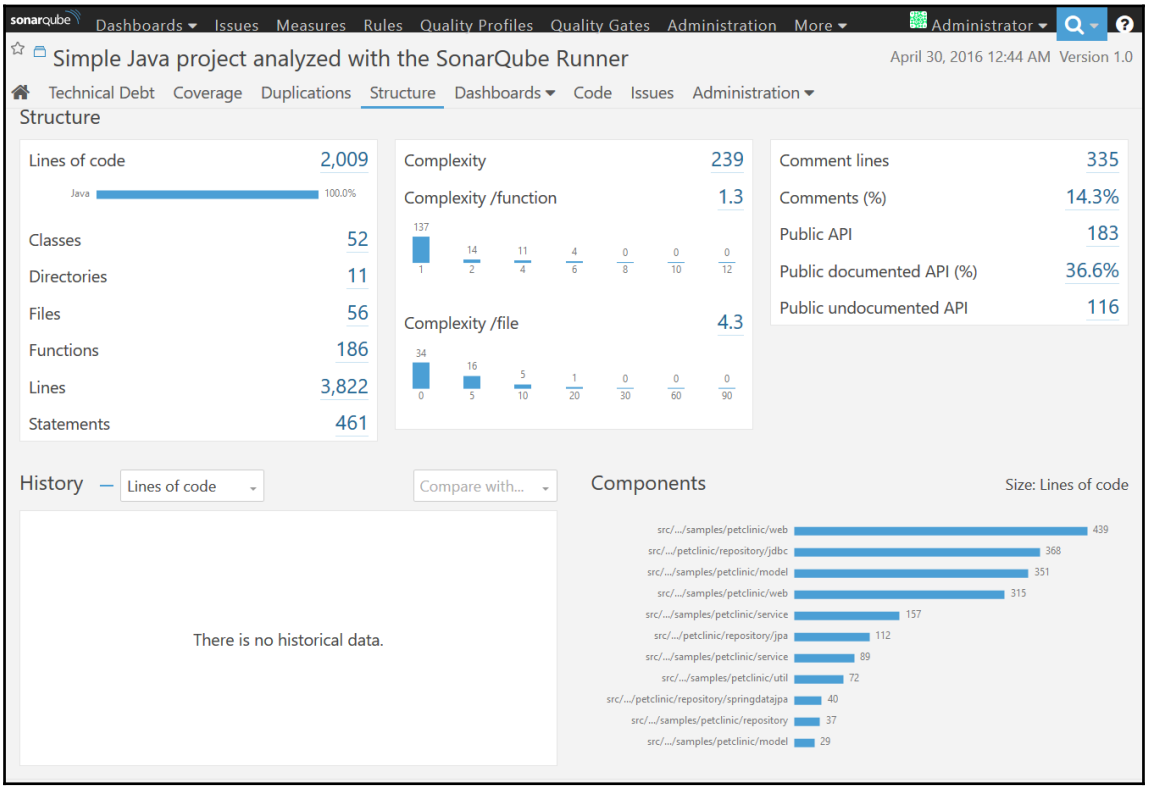



6. **Quality Gates** can be defined in the Sonar dashboard. We have used default

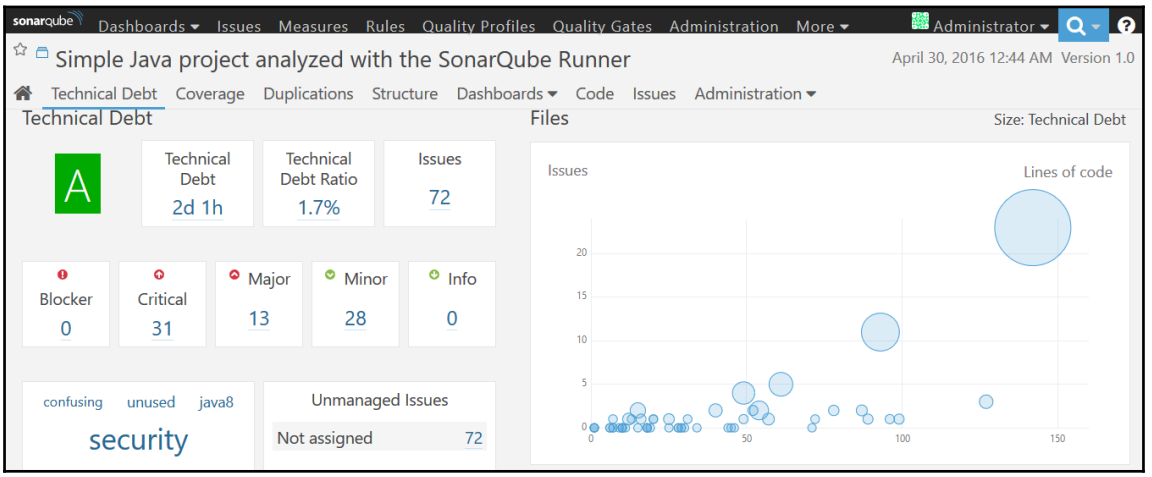
quality gate here.




7. To verify **Lines of code**, **Complexity**, and **Comment lines** click on **Structure** tab in Sonar dashboard.




8. To get more insights into issues in specific files, click on **Technical Debt** tab and click on Bubbles available in the chart.




- Sonar stores historical data in 24-hour slices.

# Self-Test Questions

1. State whether following statement is True or False: Jenkins is written in Java.

    1. True
    2. False

2. On which of the following operating systems Jenkins can be installed?

    1. Ubuntu/Debian
    2. Windows
    3. Mac OS X
    4. CentOS/Fedora/Red Hat
    5. All of the above

3. Which of the following command can be used to change the default port on which Jenkins is running?

1. java -jar jenkins.war –httpPort=9999
2. java -jar jenkins.war –http=9999
3. java -jar jenkins.war –https=9999
4. java -jar jenkins.war –httpsPort=9999

4. State whether following statement is True or False: Sonar stores historical data in 22-hour slices

1. True
2. False

# Summary

In this chapter, we have learnt about some new features in Jenkins 2, why Jenkins is so popular, how to install Jenkins, what are improvements with respect to security and plugin installations while setup, how to configure Java and Maven, what happens in the background when we create a new job in Jenkins, how to authenticate with Git, how to configure Git in Jenkins, unit test execution in sample spring application, how to configure dashboard view plugin with different portlets for customized view, how to manage master and slave node for load distribution and managing different environment as per need, how to configure E-mail notifications for build status, and how to integrate sonar and Jenkins.

In the next chapter, we will see one of the most important aspect in terms of orchestration of end to end pipeline of application delivery. We will discuss Pipeline concept of Jenkins 2 and Build Pipeline Plugin.

It is a right time to quote Ralph Waldo Emerson as it is relevant in the context of failures while build execution in the process of Continuous Integration:

> *"Our greatest glory is not in never failing, but in rising up every time we fail."*

# 3
# Building the Code and Configuring Build Pipeline

*"Start wide, expand further, and never look back."*

*– Arnold Schwarzenegger*

It is always better to start early and visualize the things which we want to achieve. That is the objective of this chapter. It is easy to visualize the end or realize the importance of this chapter when we will be ending the last line of last chapter of this book. One of the Highlights of Jenkins 2 release is Built-in support for delivery pipelines. We know that Jenkins is a Continuous Integration server but what if we want to use it for Continuous Delivery or Continuous Deployment too? Automation and Orchestration both are equally important while dealing with application delivery pipeline.

This chapter describes in detail how to create pipeline of different jobs for a sample JEE application. It will also cover deployment of an application to local web/application server and configuration of Build pipeline for lifecycle of continuous integration. This way Jenkins users can model application delivery pipelines as code. Once we can make it as a code then we can store in code repository and it can be managed in a better way. One important benefit is collaboration. As it can be stored in version control, different teams can reuse it for different operations based on the environment.

Readers will learn how to manage lifecycle of continuous integration including pulling code from code repository, building the code, unit test execution, and static code analysis using different jobs.

In this chapter, we will cover the following topics:

- Built-in Delivery Pipelines of Jenkins 2

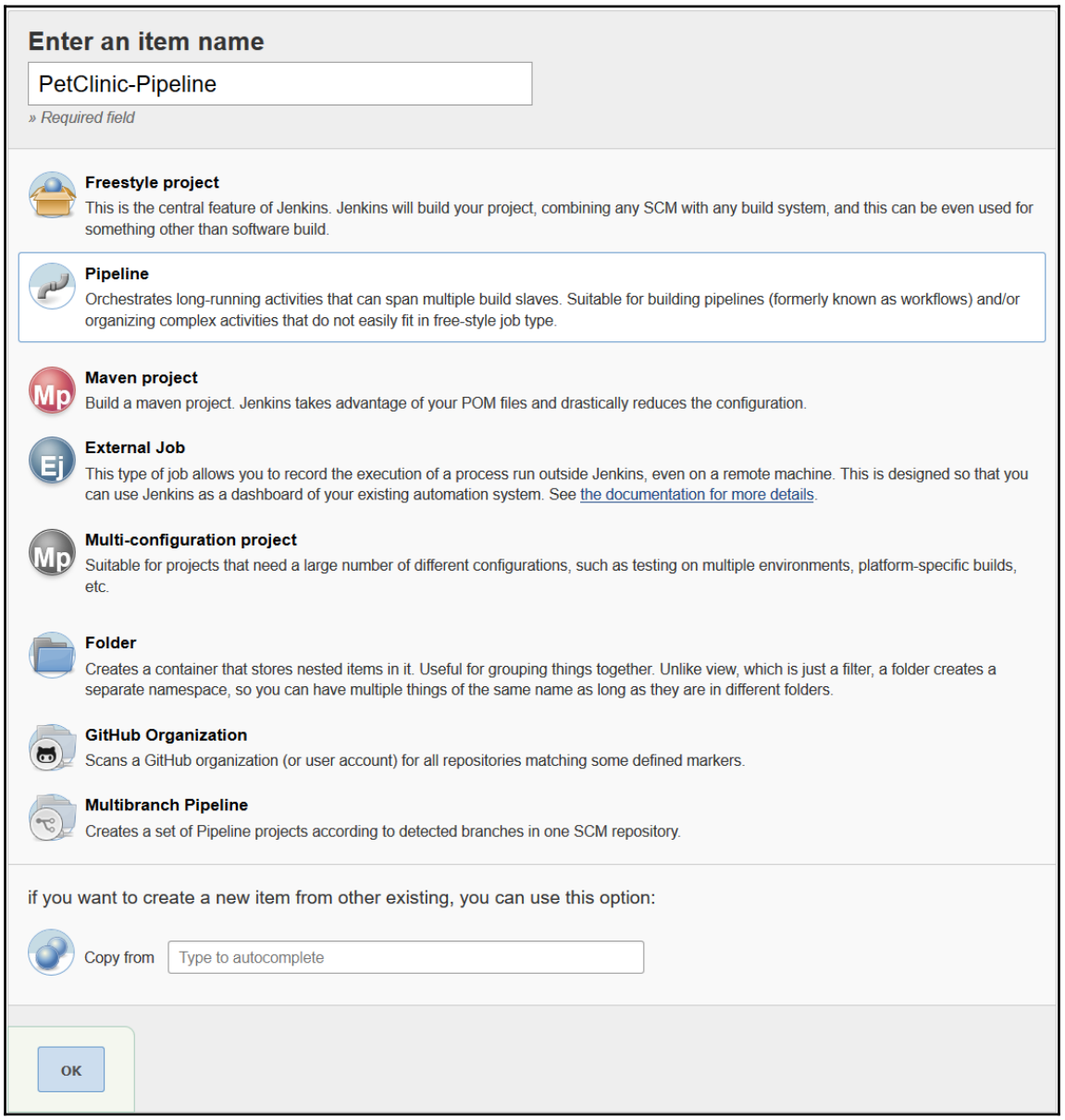
- Build Pipeline Configuration for End to End Automation to manage lifecycle of continuous integration
- Deploying a WAR file from Jenkins to Local Tomcat Server

# Creating Built-in Delivery Pipelines

Jenkins 2 provides a way to create delivery pipelines using a Domain-Specific Language (DSL).

Steps for creating Built-in Delivery Pipeline are as follows:

1. Go to Jenkins dashboard and click on **New Item**.
2. Enter an item name and select **Pipeline** as shown in below image.
3. Click on **OK**.

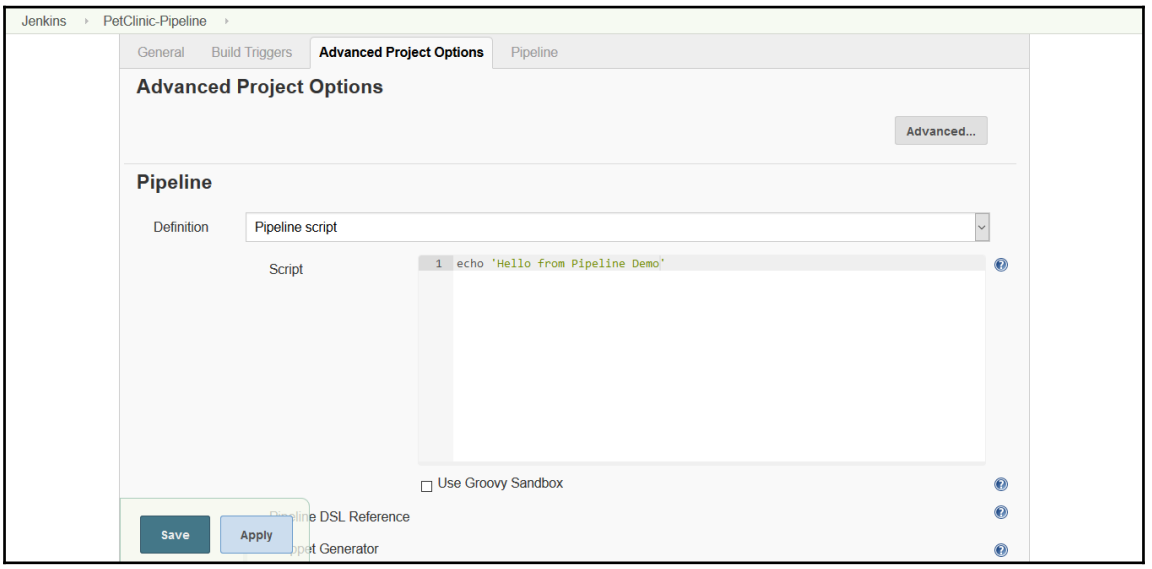



4. In case you have existing Pipeline available then you can create new pipeline by copying from it.

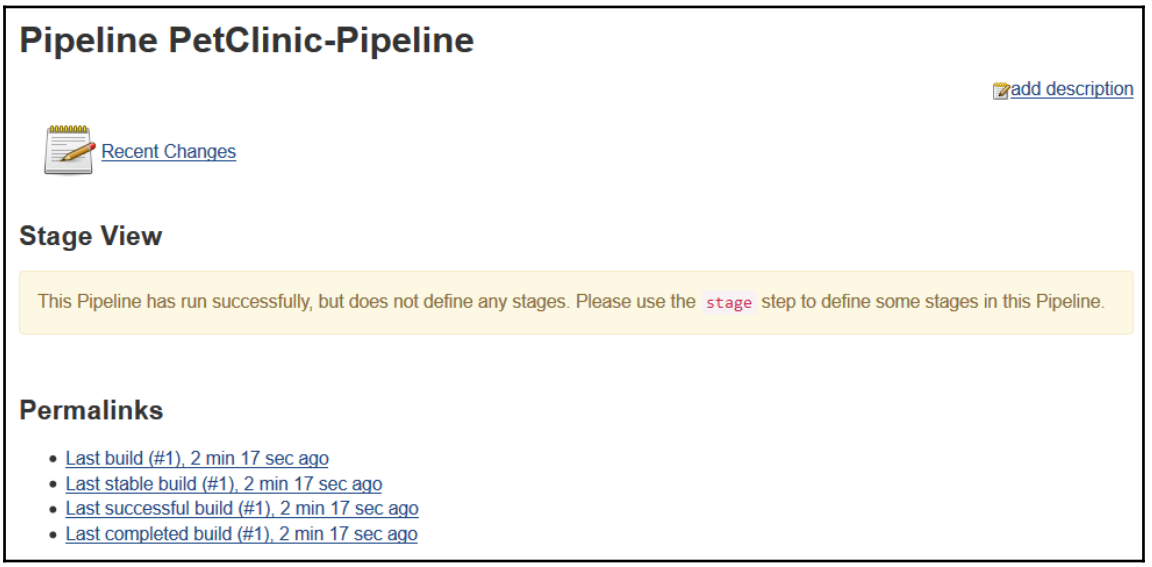
5. Go to **Advanced Project Options**. For the learning purpose, input echo

'Hello from Pipeline Demo' in the Script box.

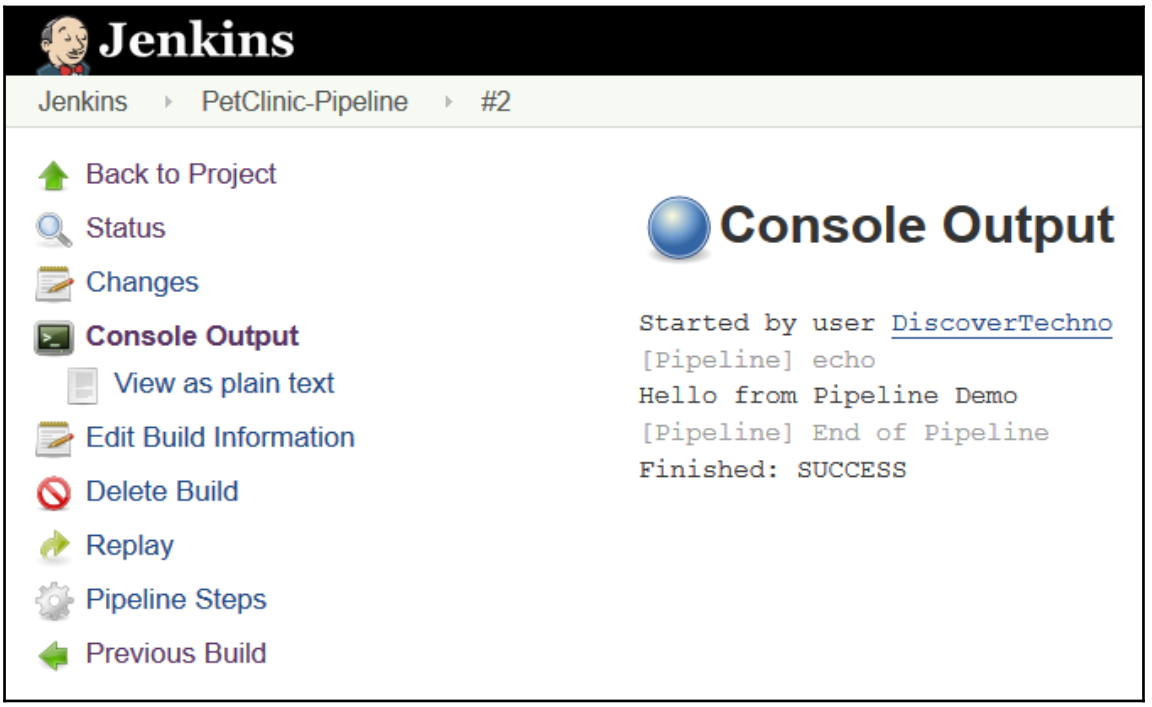
6. Click on **Save** to save the configuration.




7. As we have not created any stage, we will get warning as shown in the below image. However, we can execute the pipeline for demo purpose.



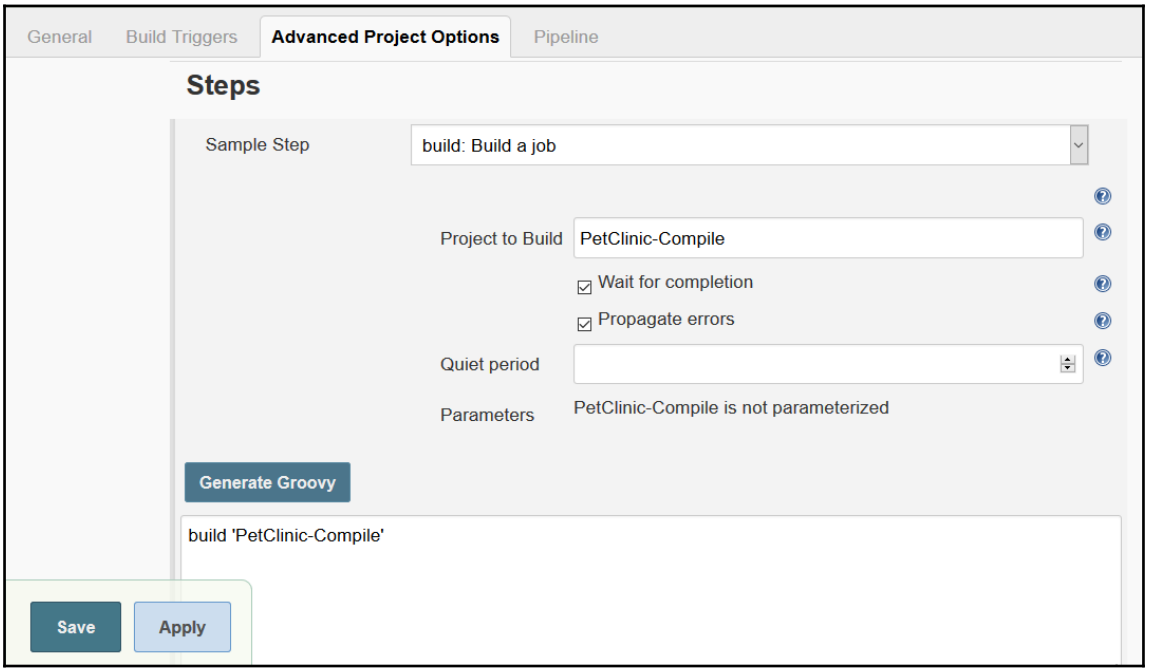

8. Click on the **Build Now**. Verify the **Console Output**. We can see the successful completion of script execution.




Let's go step by step and learn how we can create script. To make things easier refer Pipeline DSL Reference or use Snippet Generator. Select the checkbox and then select a **Sample Step**. Provide specific parameters required by the step and click on **Generate Groovy**.

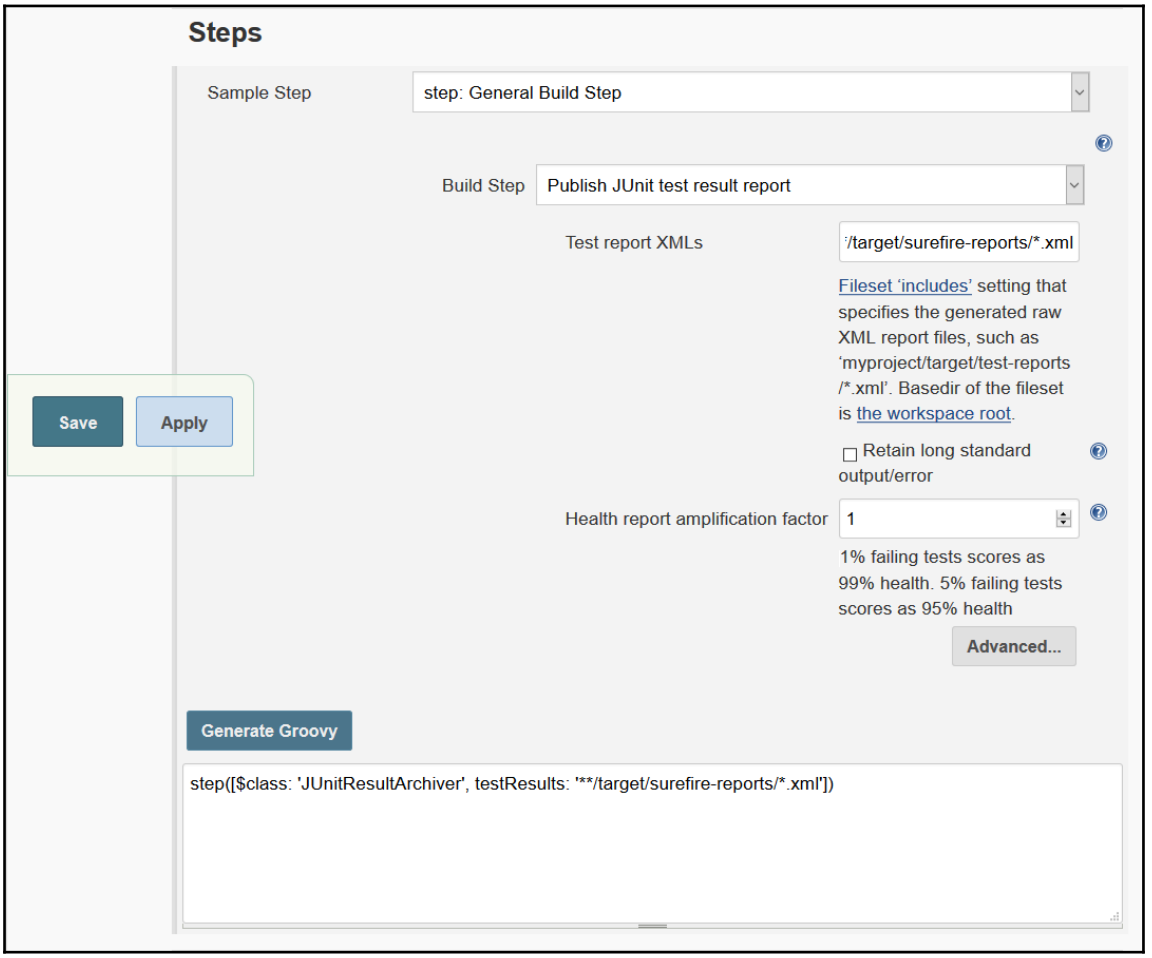
Example 1: Groovy script to build a job. It triggers a new downstream job to build.

| **Sample Step** | **build: Build a Job** |
|---|---|
| Parameters | **Project to build**: PetClinic-Compile<br>Parameters: None<br>Other configurations: Default |



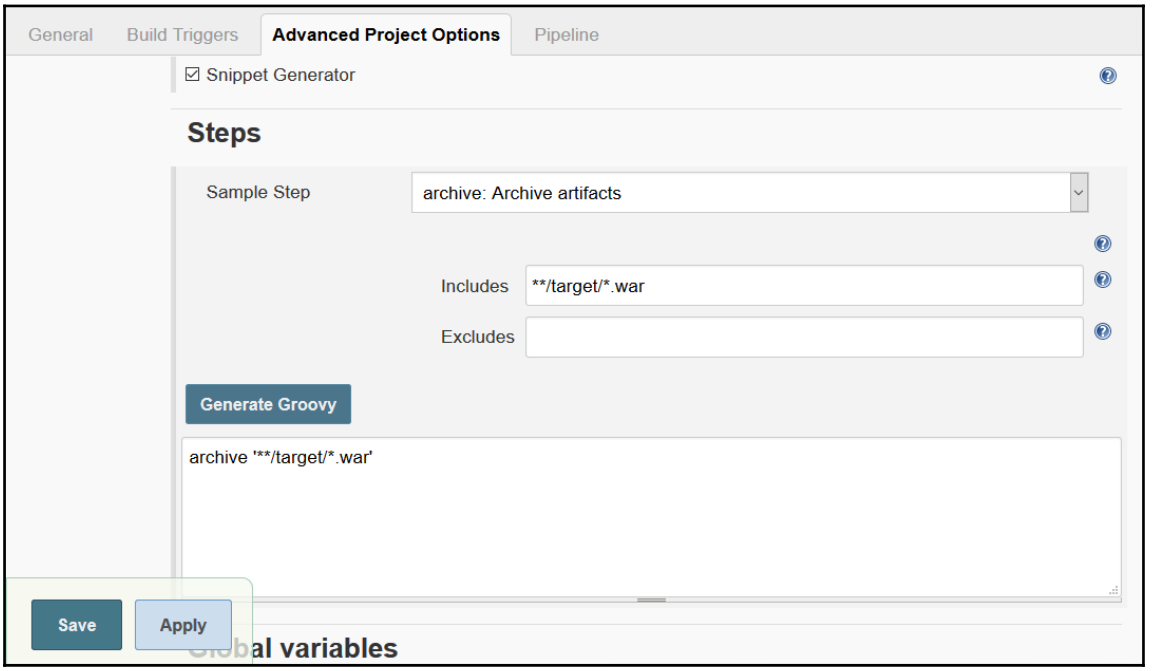

Example 2: Create a step – Generate a build step. It is used to configure post build actions or in general build step that are Pipeline-compatible based on the dropdown list.

| **Sample Step** | **step: General Build Setup** |
|---|---|
| Parameters | **Build Step**: **Publish JUnit test result report**<br>**Test Report XMLs**: **/target/surefire-reports/TEST-*.xml<br>Other configurations: Default |



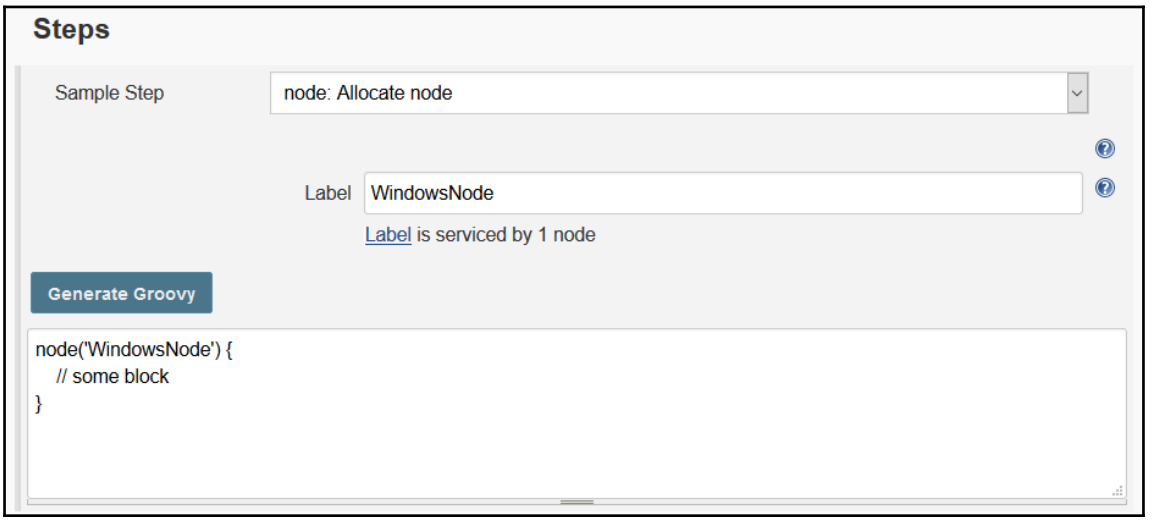

Example 3: To archive build job artifacts.

| **Sample Step** | **archive: Archive artifacts** |
|---|---|
| Parameters | **Includes**: It includes artifacts using comma separated list matching Ant style pattern for archiving artifacts.<br>**Excludes**: It excludes artifacts using comma separated list matching Ant-style pattern for not archiving artifacts. |



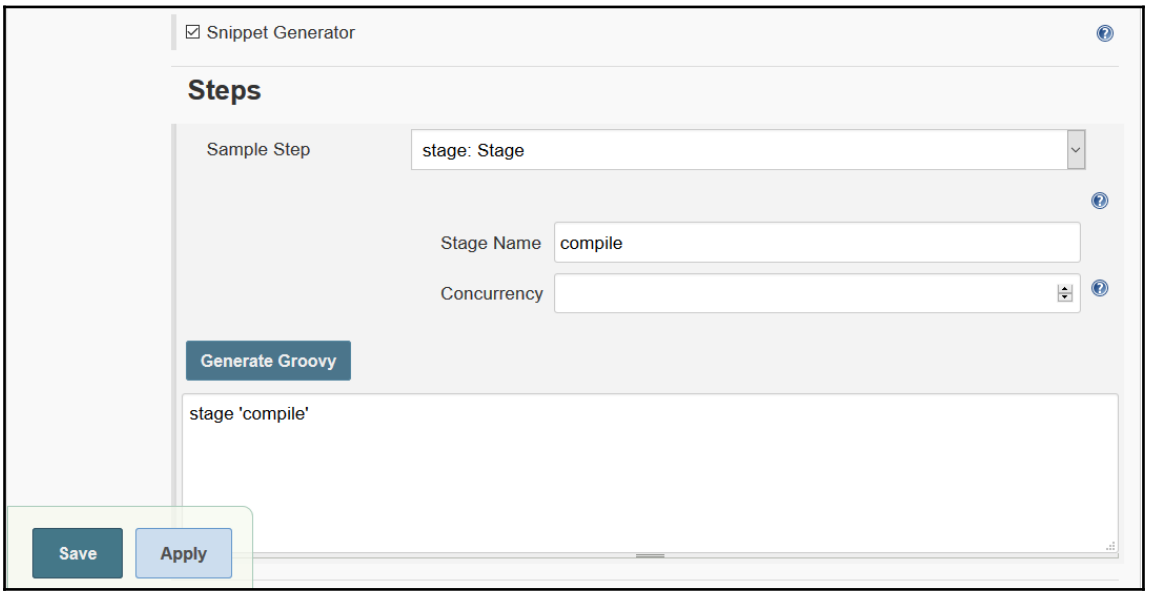

Example 4: For example, to run build step on a specific node, we need to write a script. Use Snippet Generator and select sample step node and select the slave node label. Click on **Generate Groovy**.

| **Sample Step** | **node: Allocate node** |
|---|---|
| Parameters | **Label**: Label associated with slave node. Refer to Chapter 2, *Continuous Integration with Jenkins.* for more details on Master Slave nodes in Jenkins 2. |




Example 5: Groovy script to mark definite sections of a build as being controlled by limited concurrency.

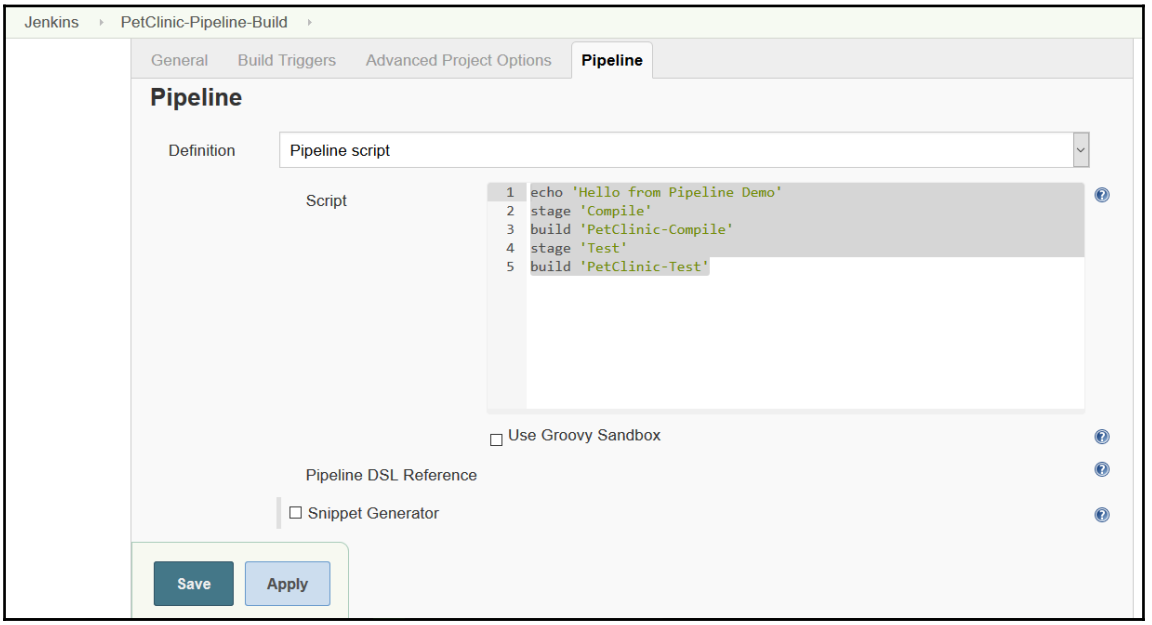
| **Sample Step** | **stage: Stage** |
|---|---|
| Parameters | Stage Name: Compile / Test / Deploy<br>Other configurations: Default |



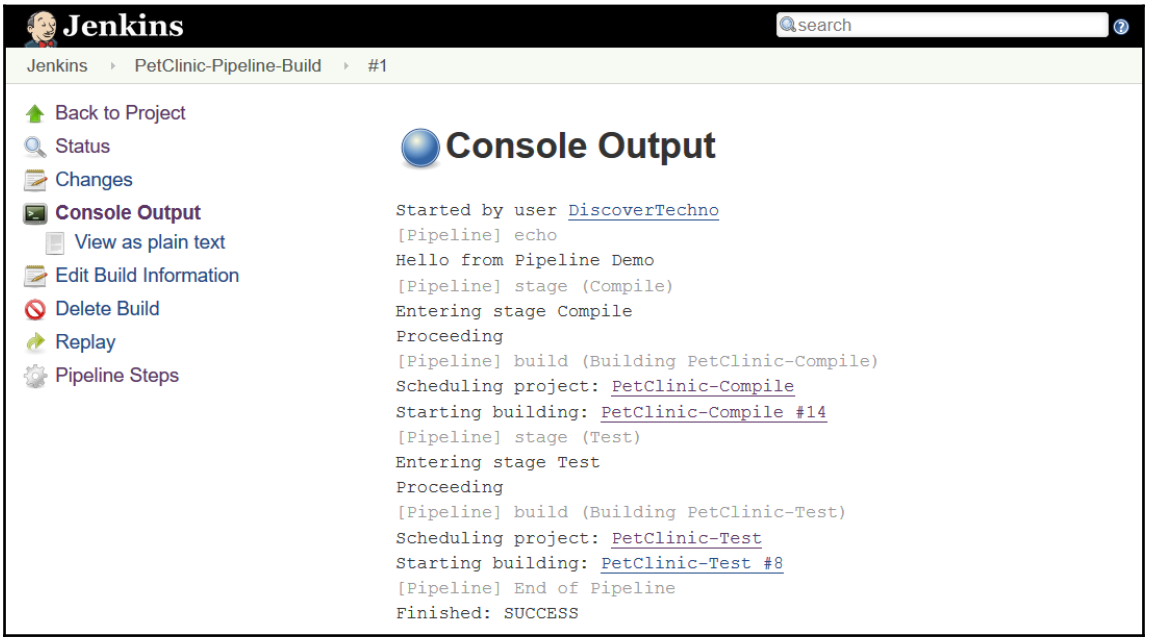

For test purpose let's try a simple scenario to create pipeline for compiling source files and executing unit test cases.

1. Let's write below script in the Script Box.

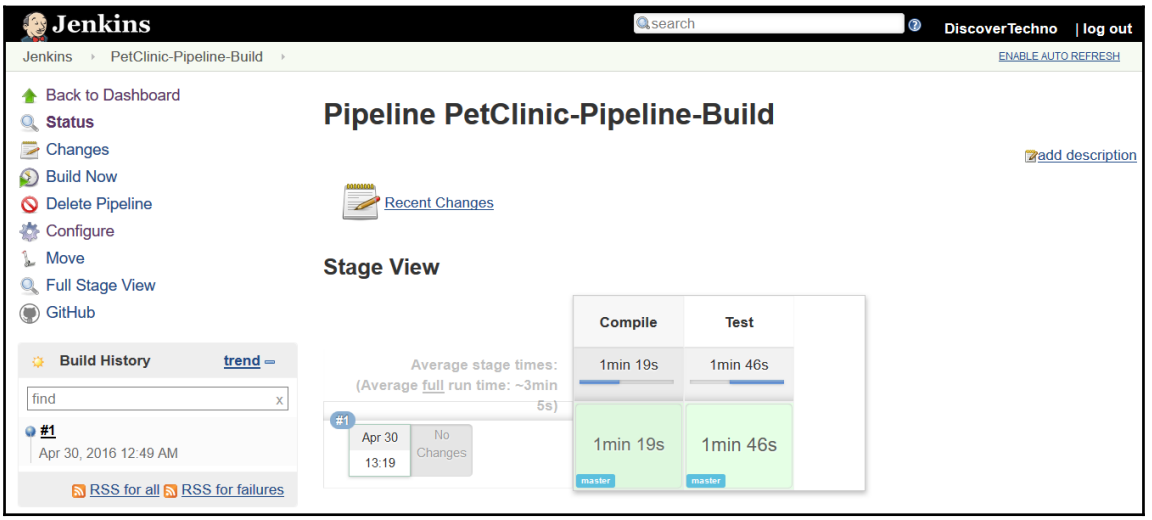
```
echo 'Hello from Pipeline Demo'
stage 'Compile'
build 'PetClinic-Compile'
stage 'Test'
build 'PetClinic-Test'
```




2. Click on the **Build Now** and go to **Console Output** to verify the execution process.

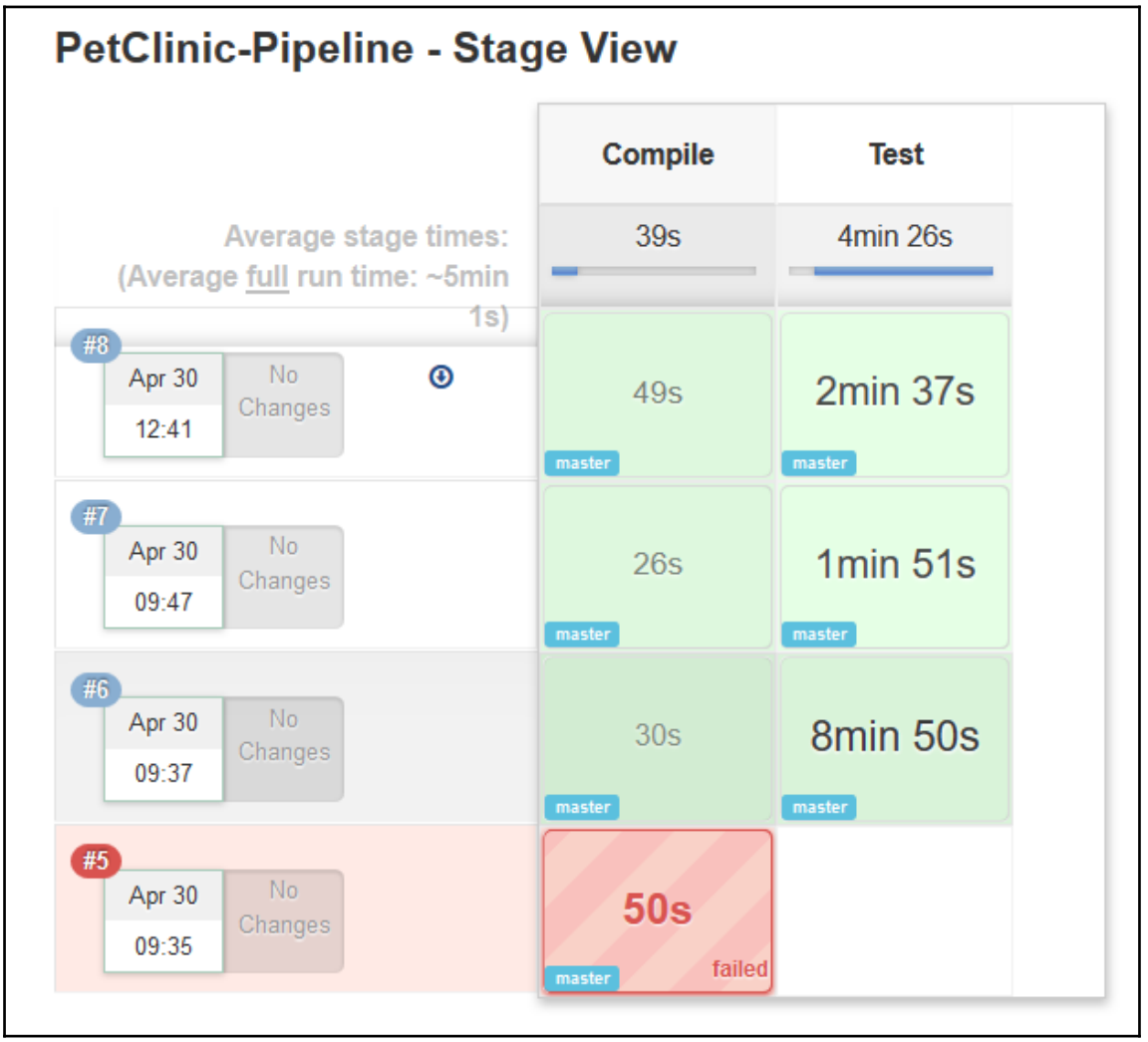



3. Go to Build Job's main page. We can see **Stage view** here. Remember, we have created two stages, one is compile and another is test. Stage view provides instant visualization. It provides details such as build completion time, on which node build has been executed, build has been failed or executed successfully.

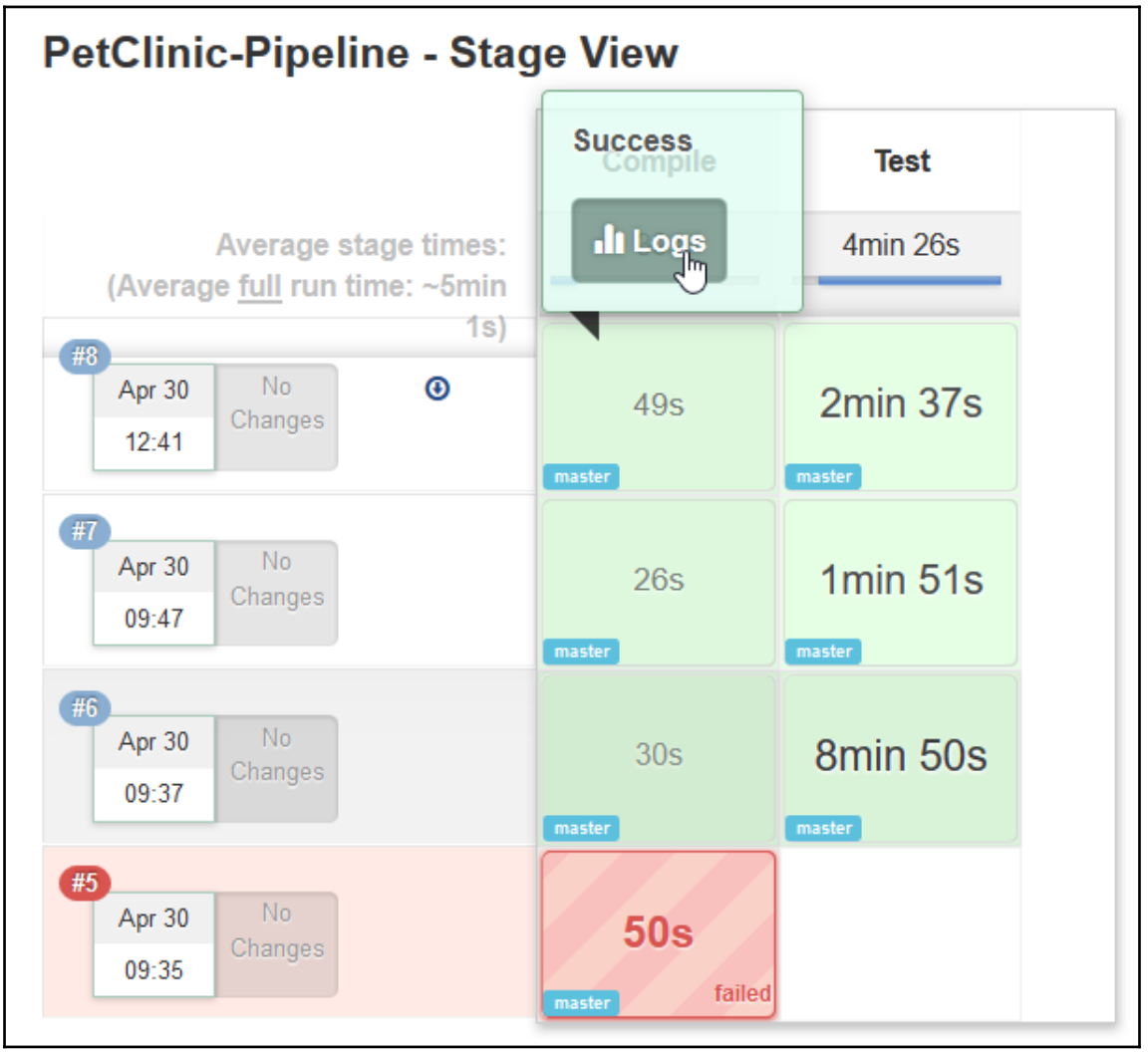



4. In the specific build execution, we can verify Pipeline Steps also.


5. Click on **Full stage view** to get full screen view as shown in below image:




6. To get details specific to stage, mouse over specific stage and it will show us status of that stage execution as well as **Logs** link.

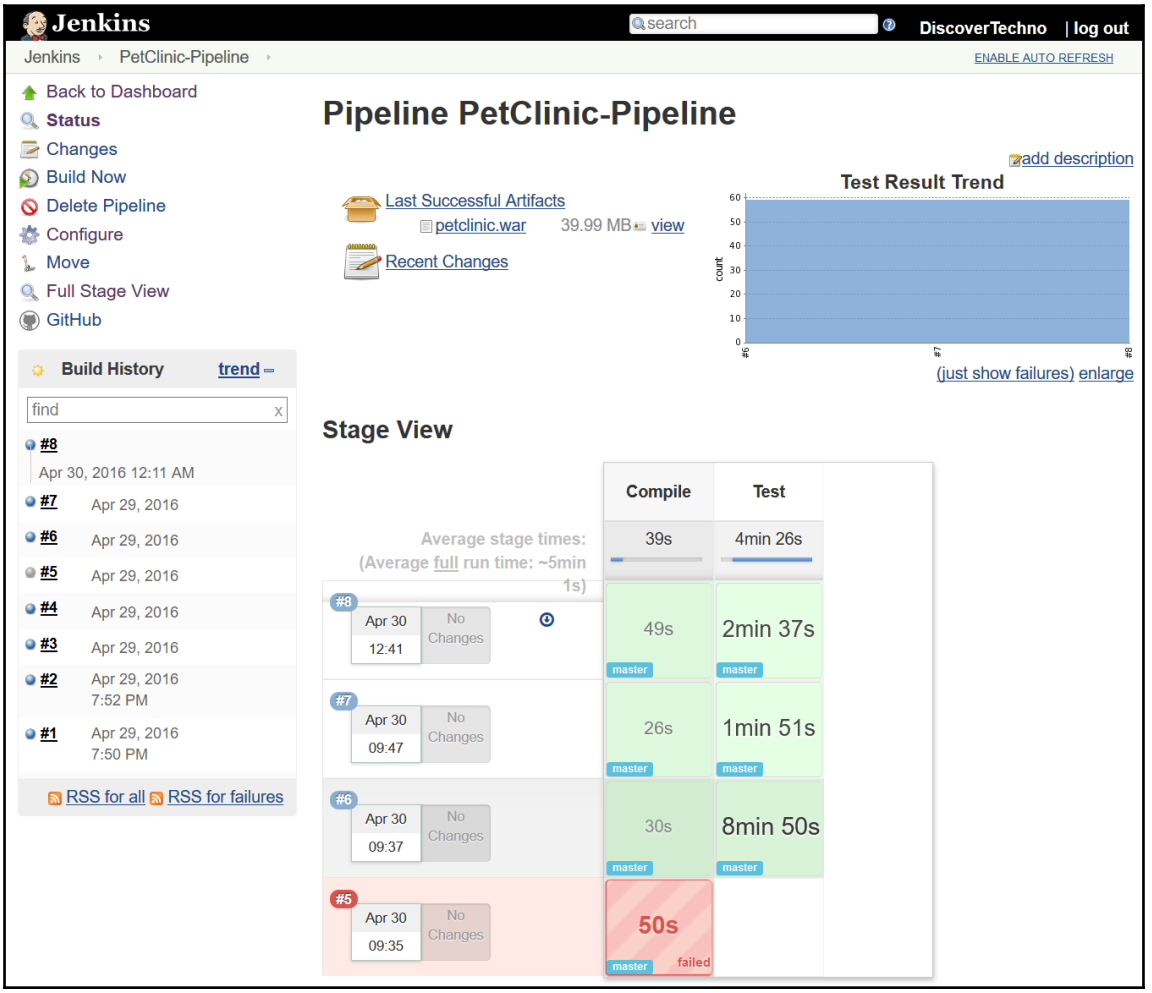



7. Click on the **Stage Logs** link and it will provide log details respective to stage. Click on dropdown to get more details about logs.

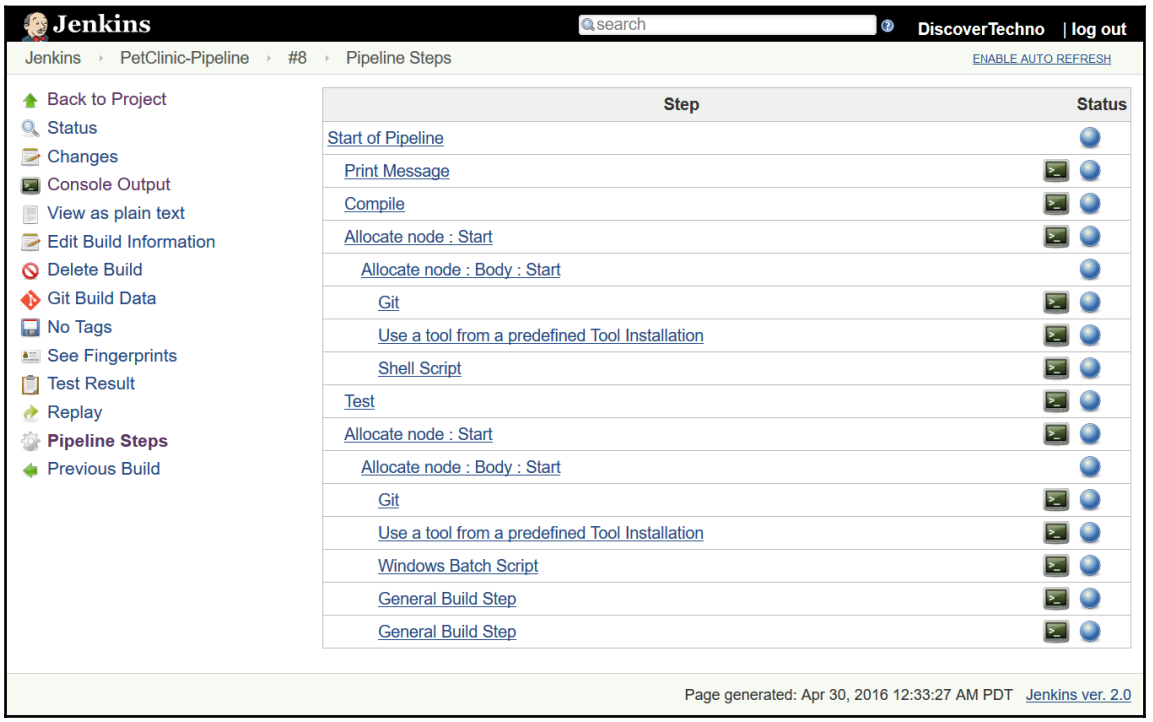

8. Let's consider a scenario where we want to execute different stages on different nodes.

```
echo 'Hello from Pipeline Demo'
stage 'Compile'
node {
  git url: 'https://github.com/mitesh51/spring-petclinic.git'
  def mvnHome = tool 'Maven3.3.1'
  sh "${mvnHome}/bin/mvn -B compile"
}
stage 'Test'
node('WindowsNode') {
  git url: 'https://github.com/mitesh51/spring-petclinic.git'
  def mvnHome = tool 'WindowsMaven'
  bat "${mvnHome}\\bin\\mvn -B verify"
  step([$class: 'ArtifactArchiver', artifacts: '**/target/*.war', fingerprint: true])
  step([$class: 'JUnitResultArchiver', testResults: '**/target/surefire-reports/TEST-*.xml'])
}
```

9. Click on **Build Now** and verify the **Stage View**.



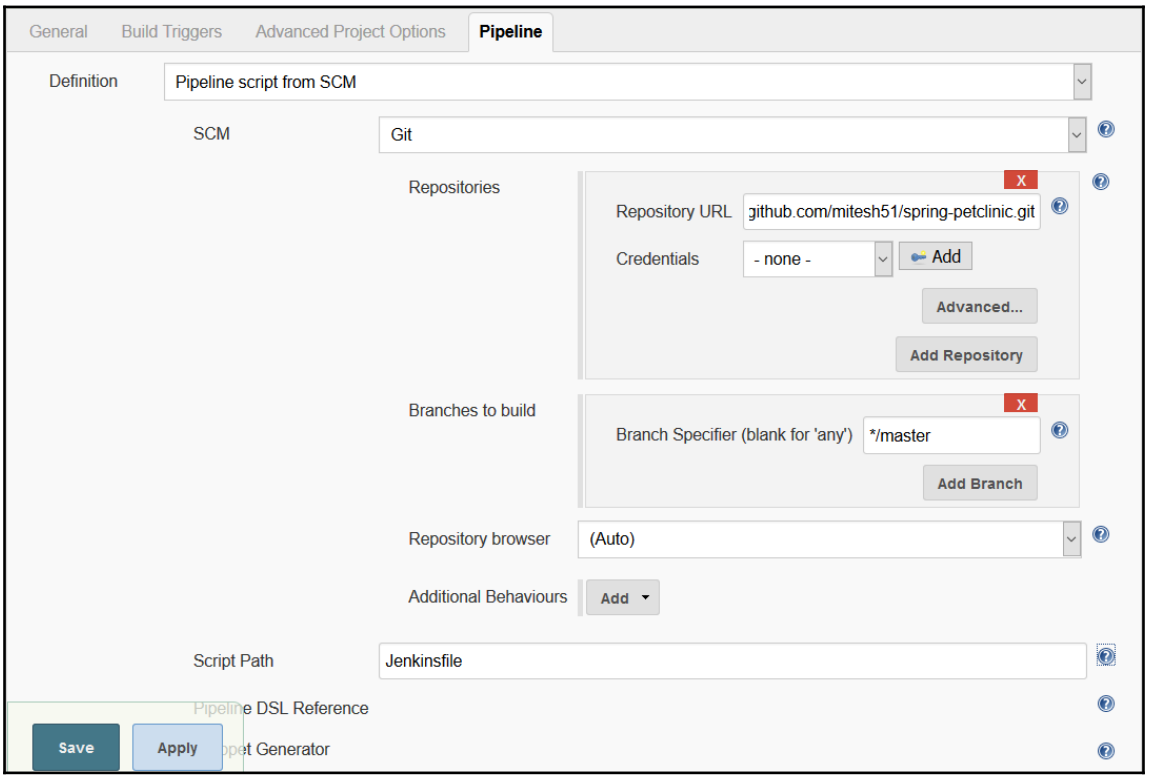

10. Pipeline steps describes drill down details of execution as shown below:




11. Let's verify stage logs for Git Operation. Mouse over the compile stage and click on **logs**. Expand **Git** dropdown as shown in the below image to get more details.


Can you guess what can be the potential issue with Groovy script for creating pipeline?

Yes, again it is a code. It becomes difficult to manage it over the time and hence it is always better to store them in repository. In Pipeline definition, there is an option available to load Pipeline script from SCM. We can select SCM from Git or Subversion and then we need to provide repository details and script file details.



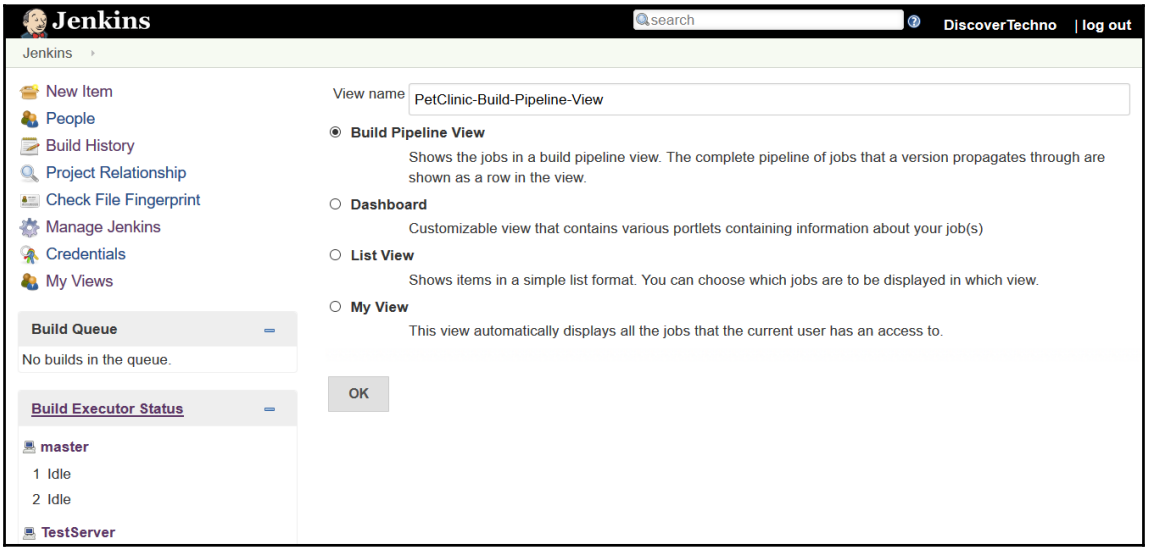

Getting Started with Pipeline at https://jenkins.io/doc/pipeline/

# Building Pipeline plugin

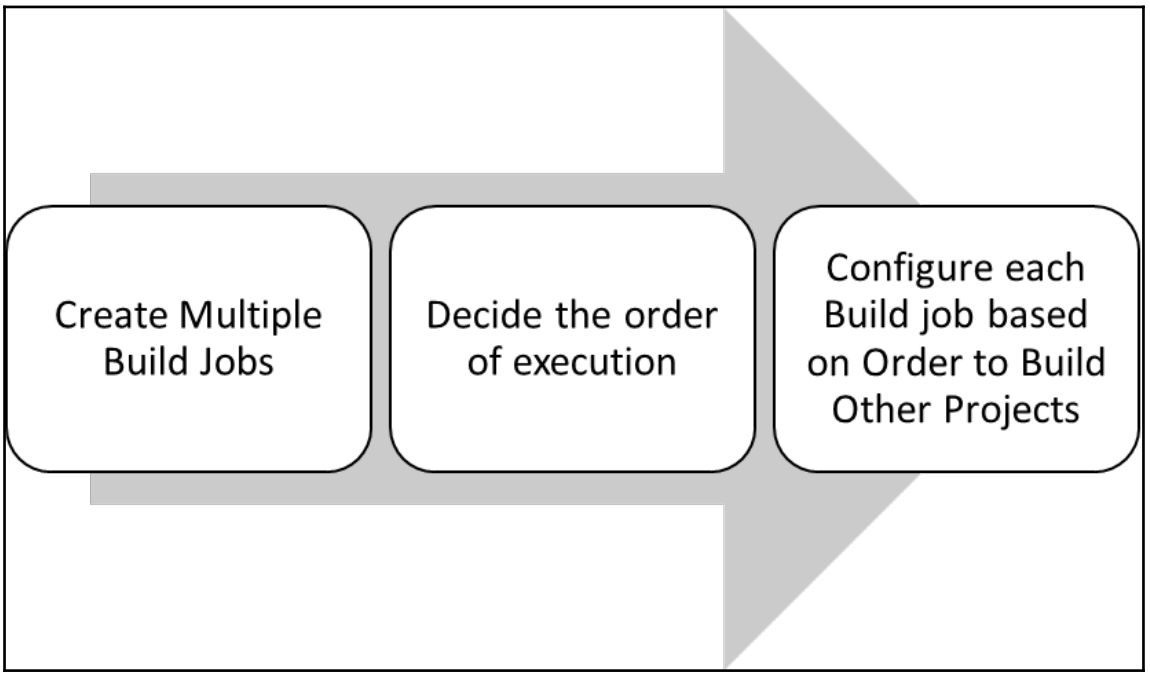
We have seen built-in pipeline concept of Jenkins 2. It is a very flexible and powerful concept but for that we need to write a groovy script. One another way that has easy learning curve is to use build pipeline plugin. It provides simple visualization upstream and downstream build jobs. It also allows manual triggers for a situation where we need approvals for executing specific build. We can create chain of jobs for end to end automation. Here we assume that reader is aware about concept of upstream and downstream build jobs.

To create a Build pipeline:

1. Install Build Pipeline plugin.
2. On Jenkins dashboard, click on plus sign that will open a page to create **Build Pipeline View**. Provide name for the build pipeline and click on **OK**.



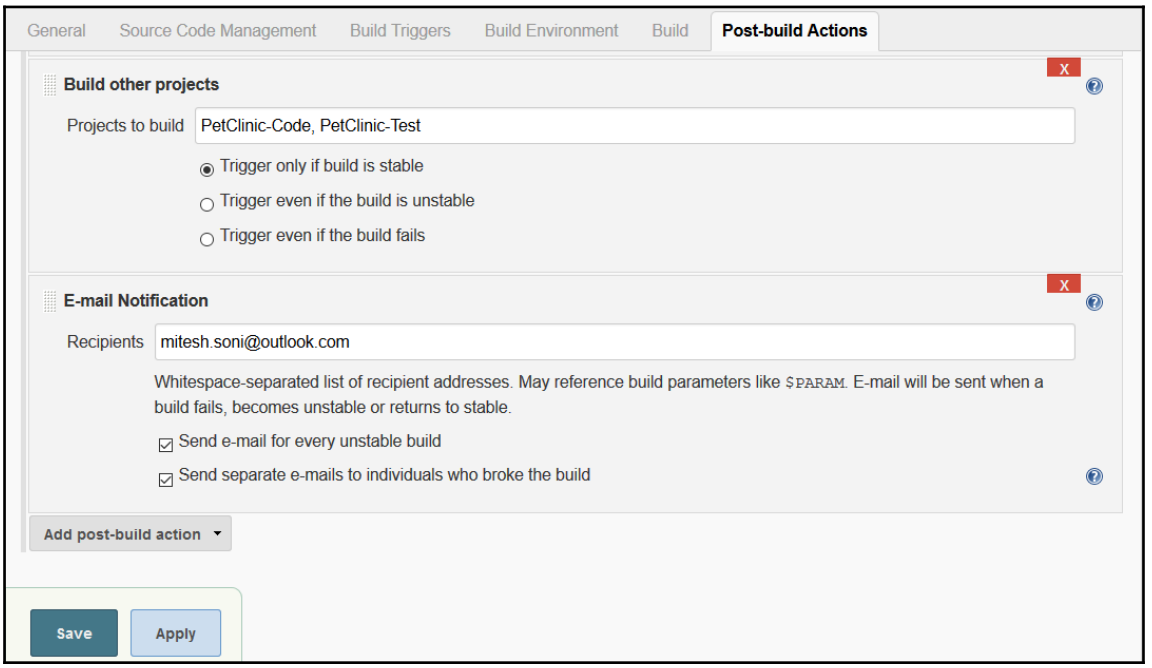

It is important to configure upstream and downstream build jobs.



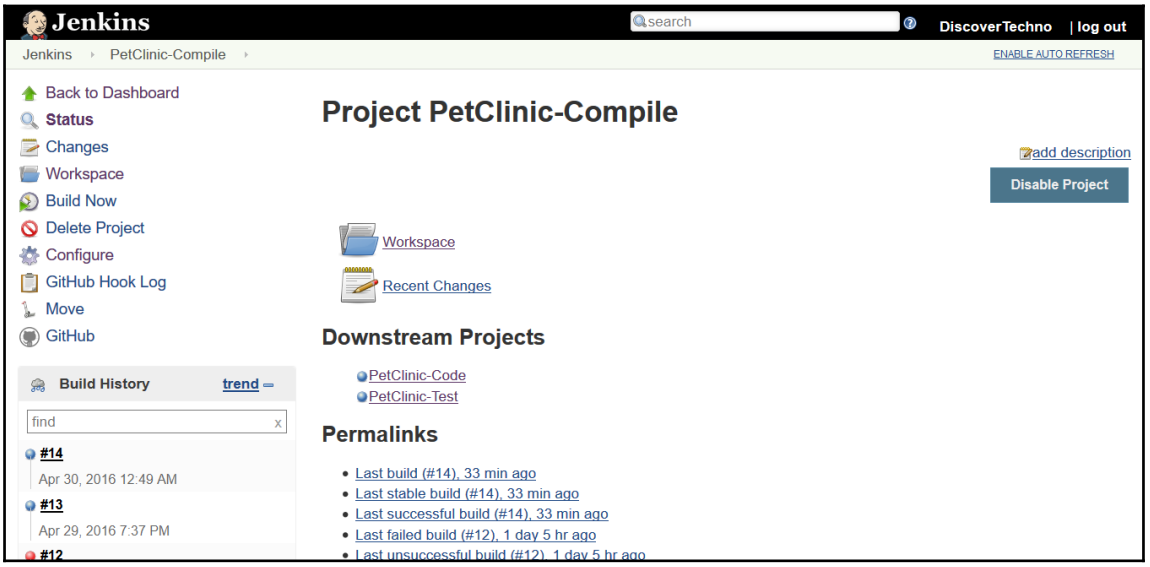

We have created multiple build jobs to compile the source code, to verify source code using Sonar, and to execute JUnit test cases.

We have defined the order also, if compilation is successful and then rest of the two build jobs will be executed. In our case it is PetClinic-Code and PetClinic-Test.

1. Go to configuration page of PetClinic-Compile build job.
2. Go to **Post-build Actions** section.
3. Enter name of the Build jobs in the **Project to build** box. We can provide a comma separated list here.
4. Click on **SAVE** to save the configuration.




5. Verify list of the **Downstream projects** on Build Job's main page.

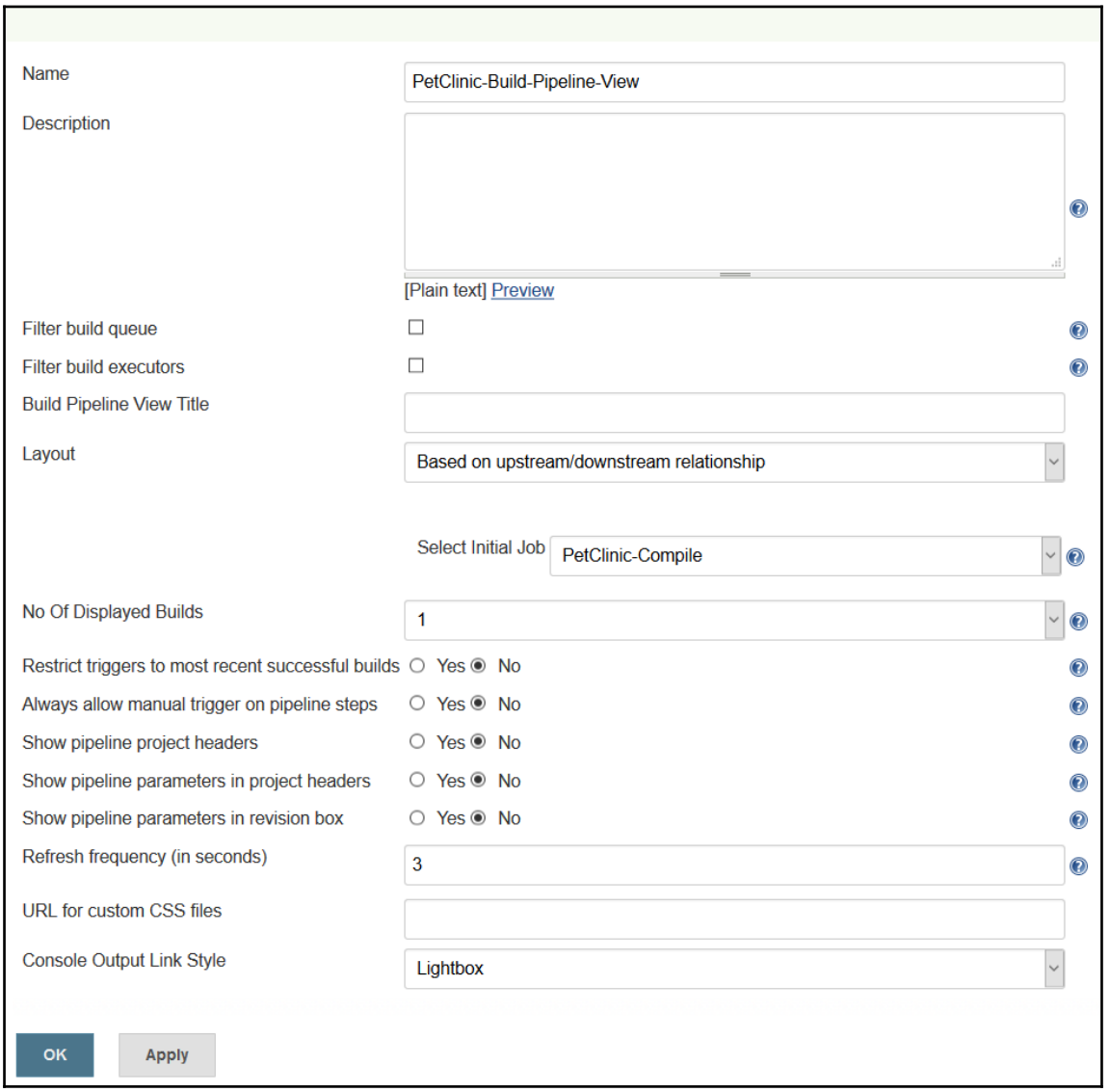



6. Now, the next step is to configure Build Pipeline view that we have created earlier.

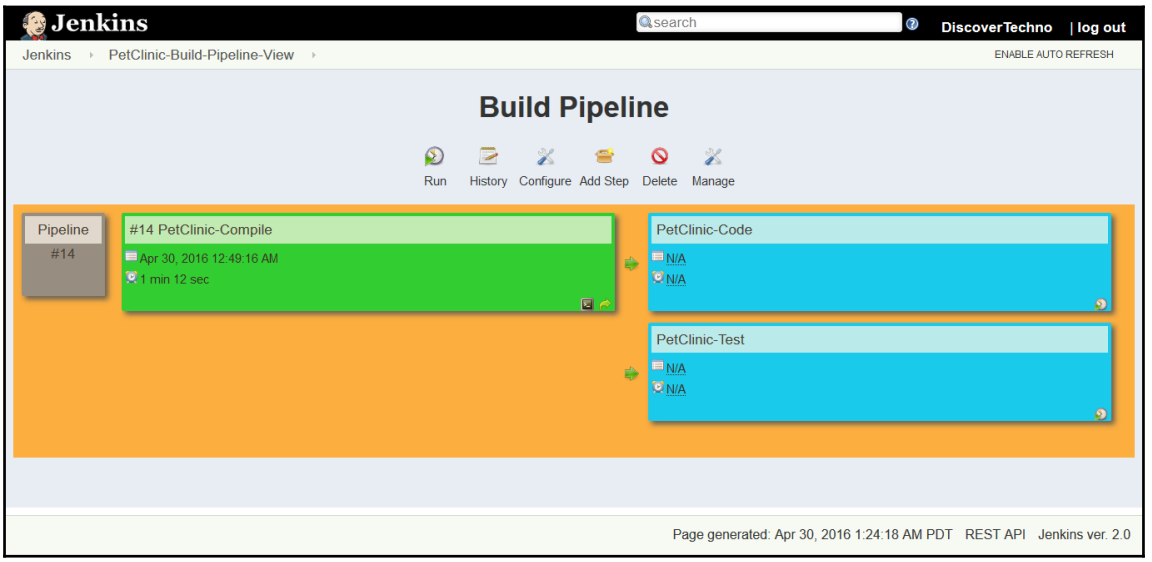
| **Name** | **Name of the Build pipeline** |
|---|---|
| Description | Description is displayed on the Build Pipeline View Page. It can be used to display details such as Pipeline, resources, objective of the pipeline, flow, and so on. |
| Filter build queue | Only jobs in this specific view will be shown in the queue. |
| Filter build executors | To show build executors that could execute the jobs in this view. |
| Build Pipeline View Title | Build Pipeline View Title to display on the Jenkins Dashboard |
| Layout | Based on upstream/downstream relationship: This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs. |
| Select Initial Job | Set the initial or parent Job in the build pipeline view. Rest of the Build Job will be considered based on upstream/downstream relationship. |
| No Of Displayed Builds | Number of build pipelines to display in the view. |
| Restrict triggers to most recent successful builds | To restrict the display of a Trigger button to only the most recent successful build pipelines. |

| | |
|---|---|
| Always allow manual trigger on pipeline steps | To execute again a successful pipeline step using the same parameter values if the build is parameterized. |
| Show pipeline project headers | To show the pipeline definition header in the pipeline view. |
| Show pipeline parameters in project headers | To list the parameters used to run the latest successful job in the pipeline's project headers. |
| Show pipeline parameters in revision box | To list the the parameters used to run the first job in each pipeline's revision box. |
| Refresh frequency (in seconds) | Provide frequency at which the Build Pipeline Plugin updates the build lightbox in seconds |
| URL for custom CSS files | Custom CSS file if any |
| Console Output Link Style | Lightbox, New Window, This Window |

7. We have select PetClinic-Compile build job as Initial Job as shown in the below image:



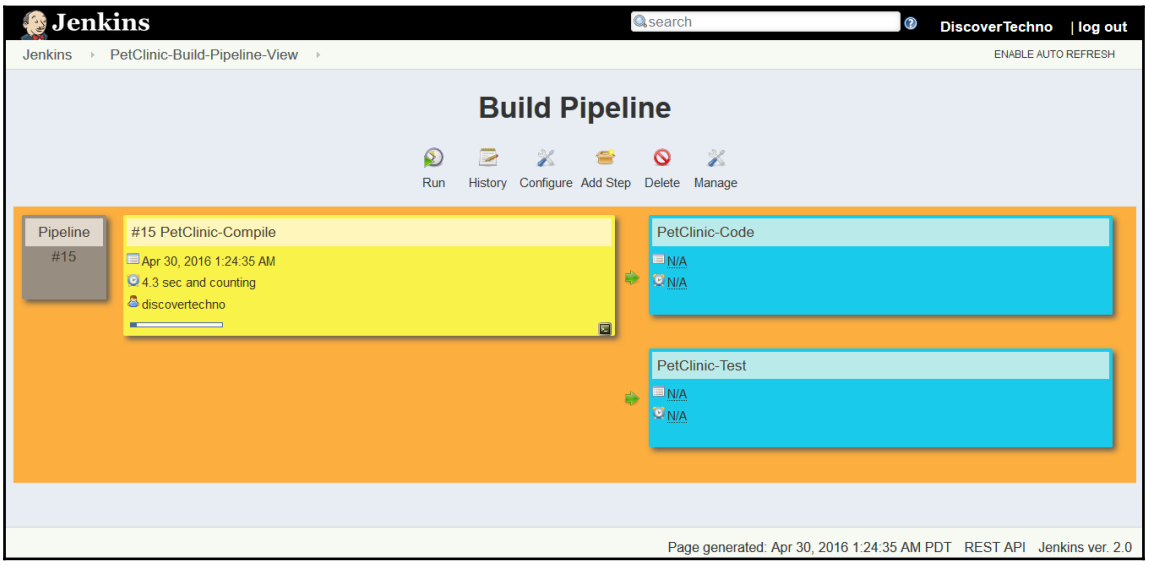

8. On the View page, we can run the build pipeline, view history, configure the pipeline, delete the pipeline, and so on. Click on **Run** to execute Build pipeline for the first time.



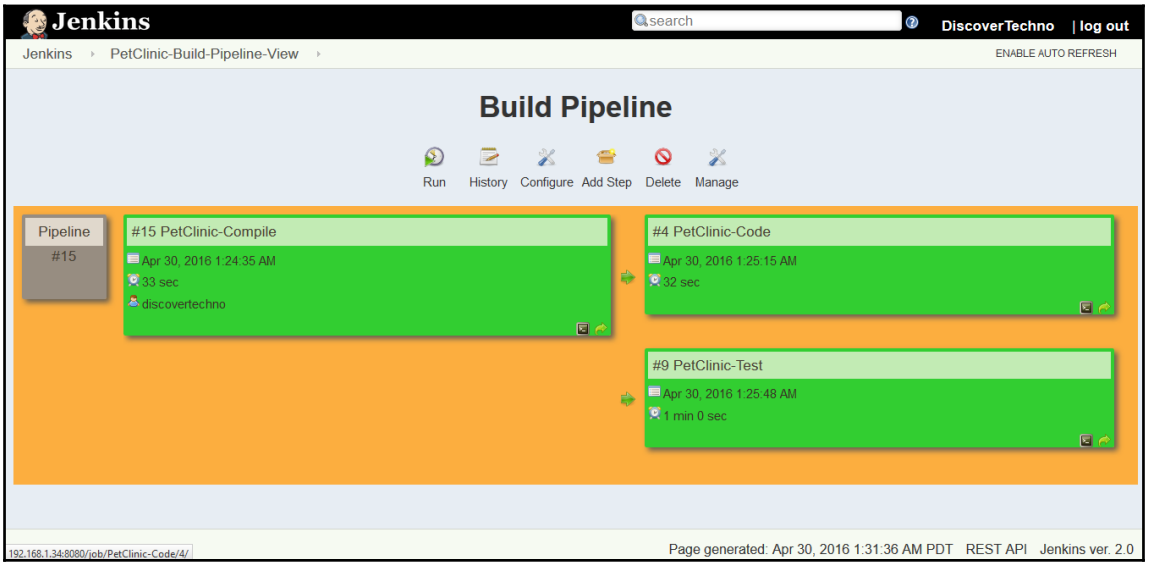

9. Following are color codes by default:

| **Color** | **Description** |
|---|---|
| Red | Indicates Failed execution of Build Job |
| Green | Indicates Successful execution of Build Job |
| Blue | Indicates Build Job that hasn't been executed |
| Yellow | Indicates Running Build Job |

10. Now just observe the execution of build job in this pipeline.

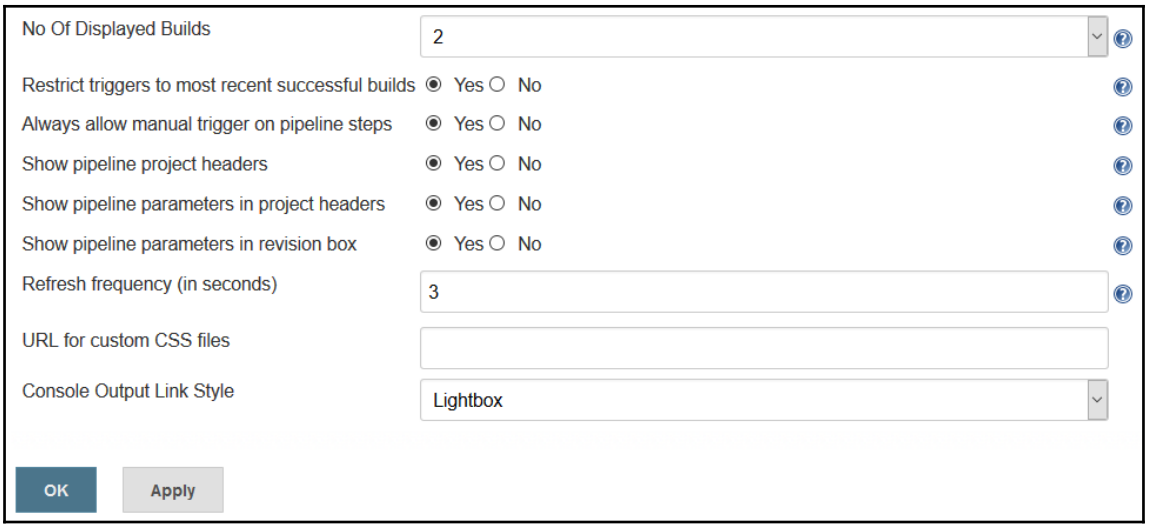



12. We can see all jobs in Green as all the builds have been executed successfully, as shown in the image below:

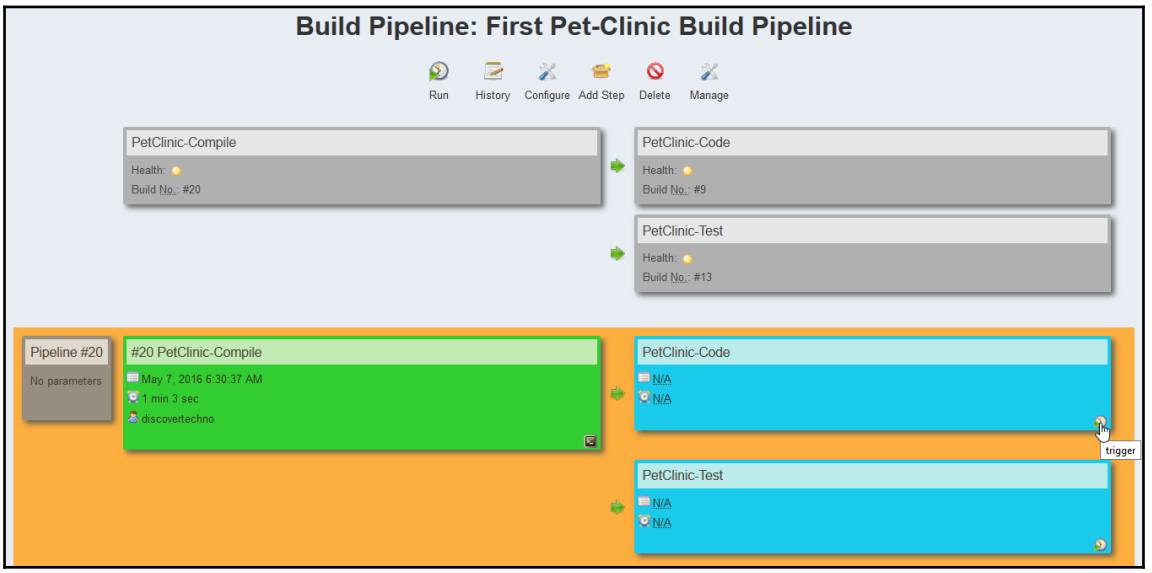



Let's configure Build pipeline using manual trigger:

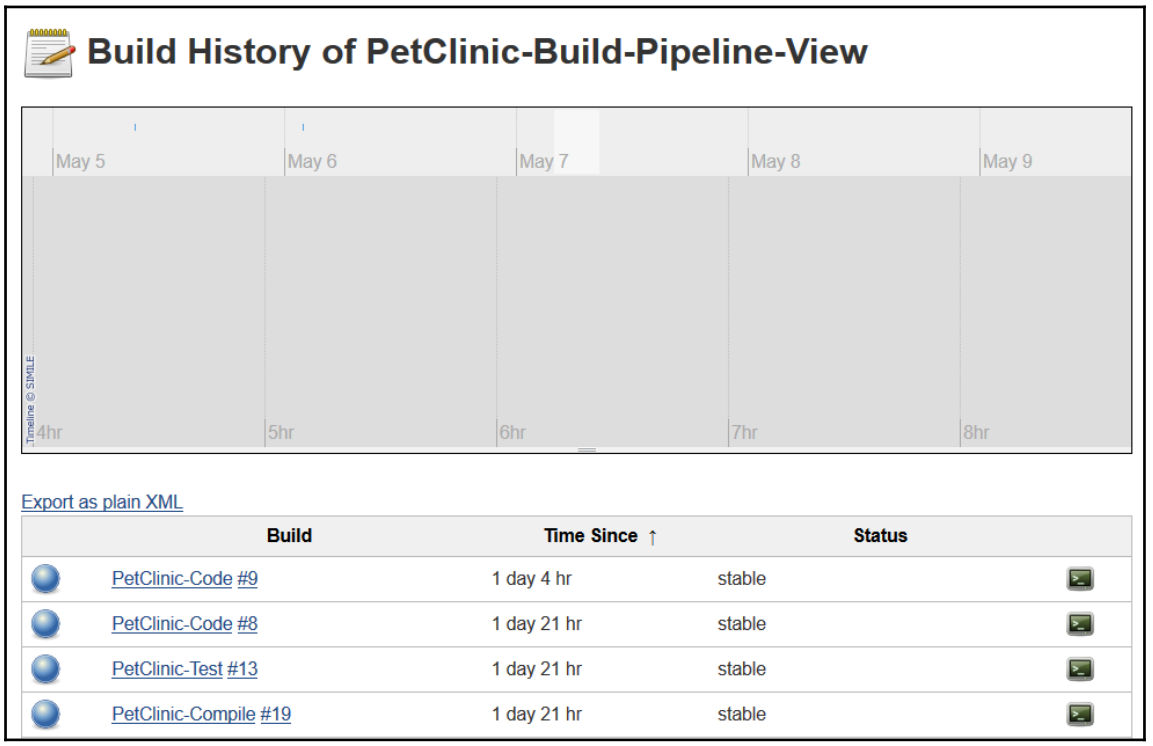
1. Show pipeline project headers, Show pipeline parameters in project headers, Show pipeline parameters in revision box, and so on.




2. Let's save and verify the changes in the Build pipeline view. Verify the manual trigger and Headers with health details of each build job.




3. Verify the History of the Build pipeline as shown in below image.




Download Build Pipeline Plugin at:
https://wiki.jenkins-ci.org/display/JENKINS/Build+Pipeline+Plugin

# Deploying a WAR file

For Maven and Tomcat integration, lets create an admin user. We will use admin user credential to deploy an application into Tomcat server.

1. Open apache-tomcat-7.0.68\conf\tomcat-users.xml and add following statements into it.

- Here we define roles such as manager-gui, manager-script. For this deployment, we will use manager-script role.

- Create a user with name admin and assign password and roles as below:

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="cloud@123" roles="manager-script" />
```

1. Now, we need to add Tomcat's admin user that we created in the Maven setting file.

```
<servers>
<server>
  <id>tomcat-development-server</id>
  <username>admin</username>
  <password>password</password>
</server>
</servers>
```

2. Now let's edit pom.xml file. Find Tomcat Plugin block in Pom.xml and add following details. Make sure that Server Name is same that we provided in settings.xml of Maven as Id.

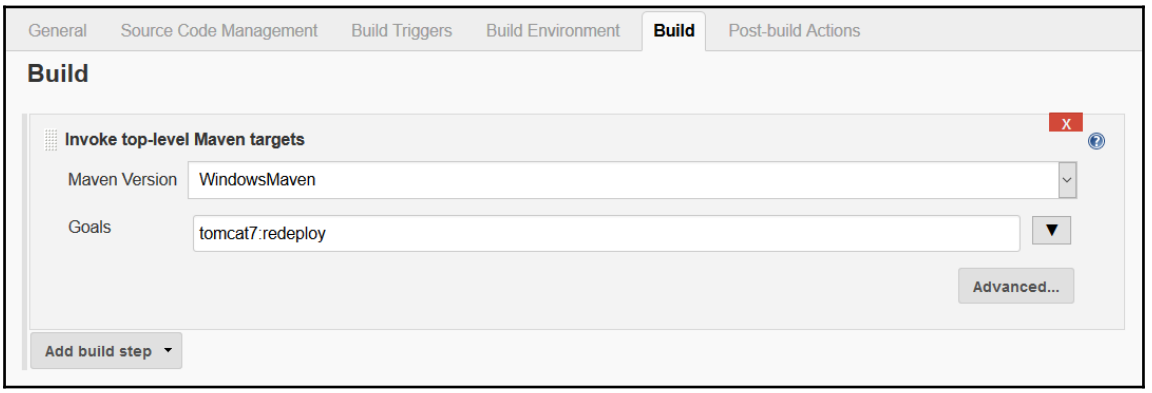
```
<plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.2</version>
            <configuration>
               <server>tomcat-development-server</server>
              <url>http://192.168.1.35:9999/manager/text</url>
              <warFile>target\petclinic.war</warFile>
              <path>/petclinic</path>
             </configuration>
 </plugin>
```

4. We can verify the execution from command line using mvn tomcat7:deploy command. Maven deploy the WAR file to Tomcat 7 using Manager App http://localhost:8080/manager/text, on path /petclinic.

5. In case of any failures because of already existing WAR file in Tomcat webapps folder, use tomcat7:redeploy.

Let's create a build job in Jenkins and add a build step to invoke top-level Maven targets:

1. Use tomcat7:redeploy as goals. Save the configuration.




2. Execute the build by click on **Build Now**. Verify the deployment process in the **Console Output**.

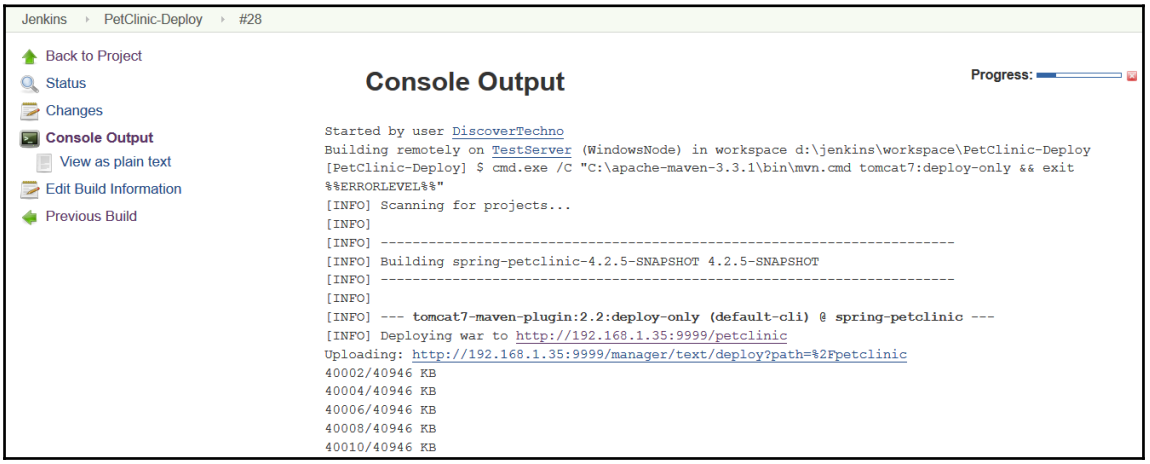
```
[INFO] --- maven-war-plugin:2.3:war (default-war) @ spring-petclinic ---
[INFO] Packaging webapp
[INFO] Assembling webapp [spring-petclinic] in [d:\jenkins\workspace\PetClinic-Deploy\target\spring-
petclinic-4.2.5-SNAPSHOT]
[INFO] Processing war project
[INFO] Copying webapp resources [d:\jenkins\workspace\PetClinic-Deploy\src\main\webapp]
[INFO] Webapp assembled in [969 msecs]
[INFO] Building war: d:\jenkins\workspace\PetClinic-Deploy\target\spring-petclinic-4.2.5-SNAPSHOT.war
[INFO]
[INFO] <<< tomcat7-maven-plugin:2.2:redeploy (default-cli) < package @ spring-petclinic <<<
[INFO]
[INFO] --- tomcat7-maven-plugin:2.2:redeploy (default-cli) @ spring-petclinic ---
[INFO] Deploying war to http://192.168.1.35:9999/petclinic
Uploading: http://192.168.1.35:9999/manager/text/deploy?path=%2Fpetclinic&update=true
40002/40946 KB
40004/40946 KB
40006/40946 KB
40008/40946 KB
40010/40946 KB
40012/40946 KB
```

3. Once WAR file is uploaded successfully, Build Job will be completed successfully.

```
40940/40946 KB
40942/40946 KB
40944/40946 KB
40946/40946 KB
Uploaded: http://192.168.1.35:9999/manager/text/deploy?path=%2Fpetclinic&update=true (40946 KB at
9024.8 KB/sec)

[INFO] tomcatManager status code:200, ReasonPhrase:OK
[INFO] OK - Deployed application at context path /petclinic
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 58.469 s
[INFO] Finished at: 2016-05-07T23:41:13+05:30
[INFO] Final Memory: 38M/263M
[INFO] ------------------------------------------------------------------------
Finished: SUCCESS
```

When we use tomcat7:deploy or tomcat7:redeploy then it includes package lifecycle in the execution. If we want to only deploy the WAR file, then we can use tomcat7:deploy-only as shown in below console output.



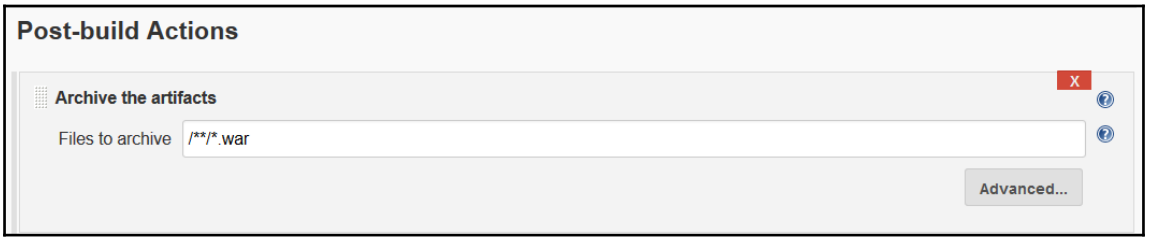

Let's try to integrate deploy operation in the build pipeline.

We need to do following things:

1. Compile source files.
2. Execute JUnit test cases.

3. Archive artifact / WAR file.
4. Copy artifact to deploy build job.

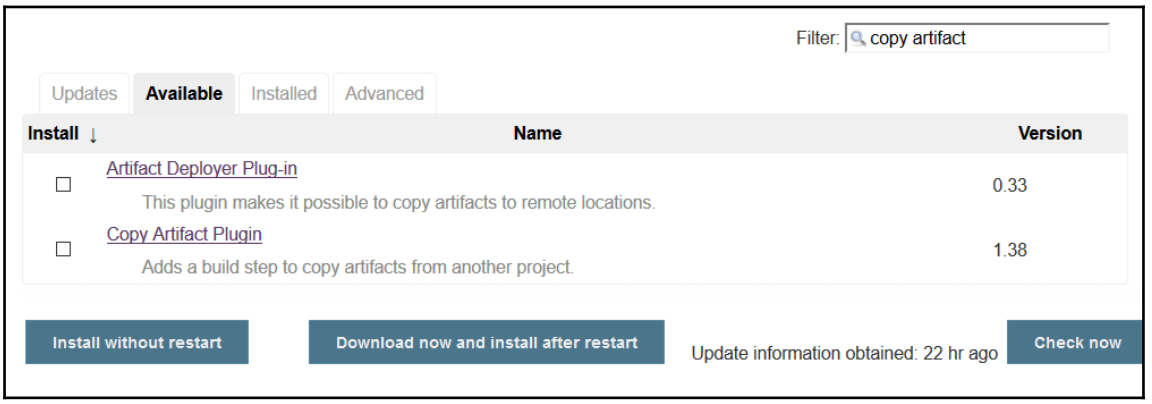
    - It is used to archive the build artifact such as jar files, war files, and zip files so it can be downloaded later. Add post build action in to PetClinic-Test file to archive artifact.



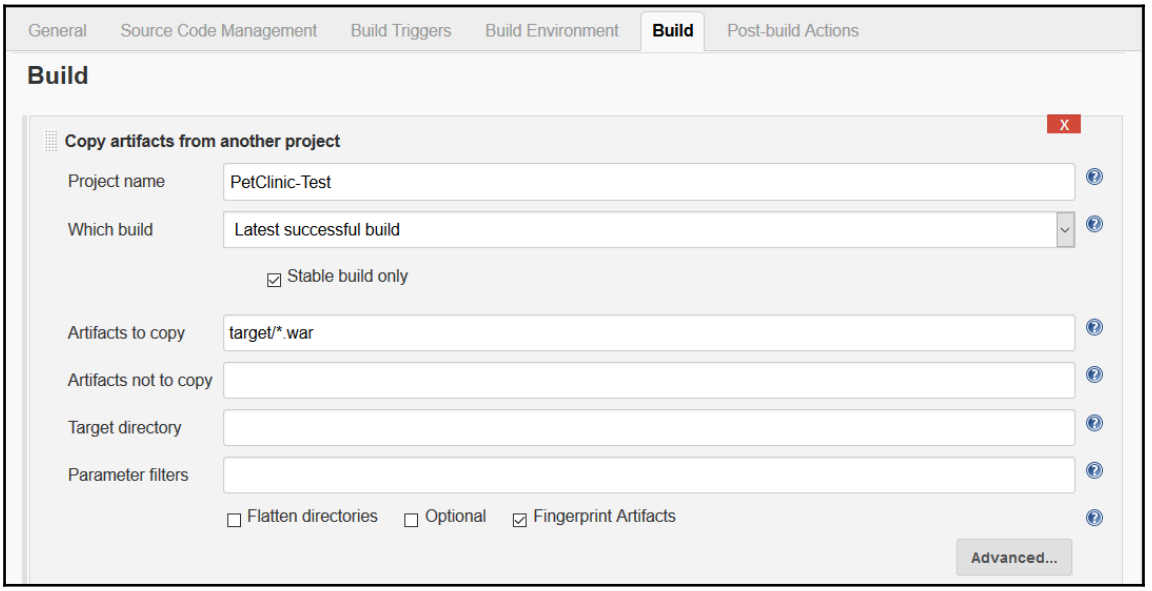

5. Execute the build job as shown below and verify whether it is successfully archived or not.

```
[INFO] --- maven-war-plugin:2.3:war (default-war) @ spring-petclinic ---
[INFO] Packaging webapp
[INFO] Assembling webapp [spring-petclinic] in [d:\jenkins\workspace\PetClinic-Test\target\spring-
petclinic-4.2.5-SNAPSHOT]
[INFO] Processing war project
[INFO] Copying webapp resources [d:\jenkins\workspace\PetClinic-Test\src\main\webapp]
[INFO] Webapp assembled in [2371 msecs]
[INFO] Building war: d:\jenkins\workspace\PetClinic-Test\target\spring-petclinic-4.2.5-SNAPSHOT.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 35.535 s
[INFO] Finished at: 2016-05-08T00:53:30+05:30
[INFO] Final Memory: 29M/271M
[INFO] ------------------------------------------------------------------------
Archiving artifacts
Recording test results
Warning: you have no plugins providing access control for builds, so falling back to legacy behavior
of permitting any downstream builds to be triggered
Triggering a new build of PetClinic-Deploy
Finished: SUCCESS
```

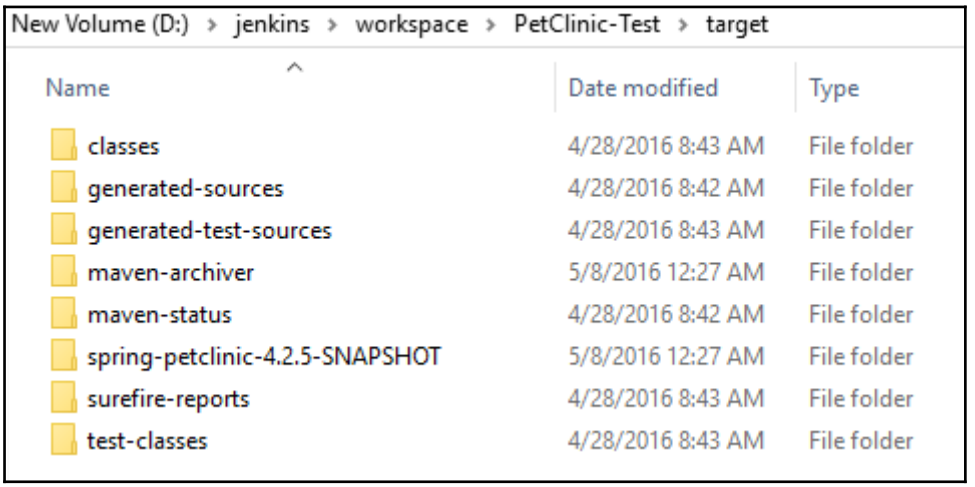
6. We need to add a build step to copy artifacts from PetClinic-Test. Install Copy Artifact Plugin.



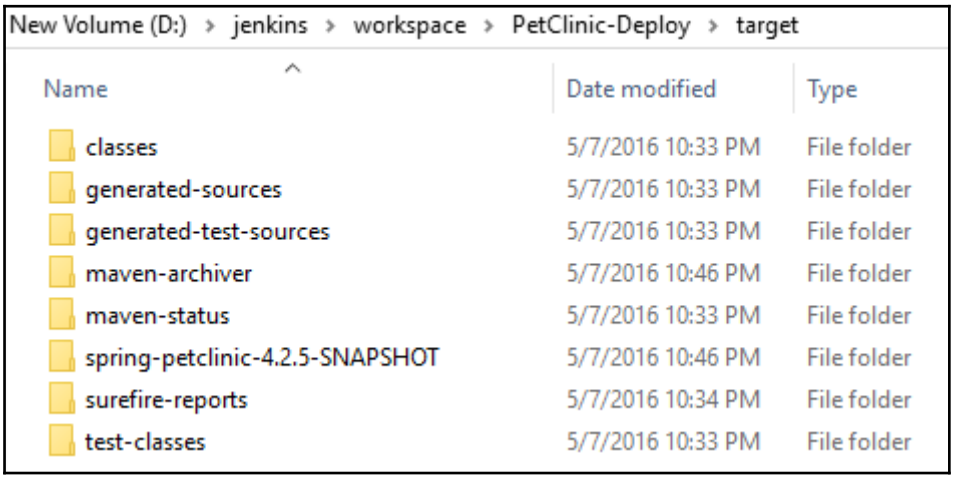

7. Configure Copy Artifact plugin in the PetClinic-Deploy Build job as shown in the below image.



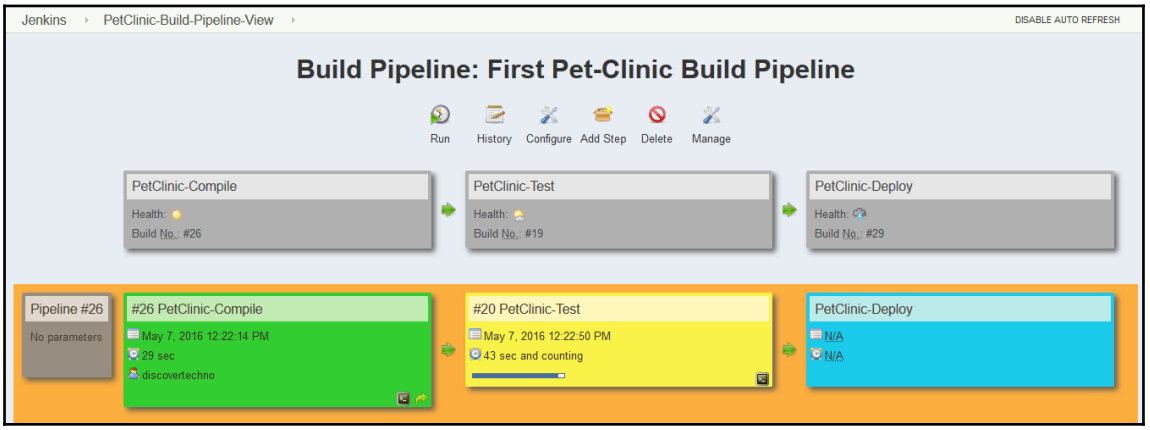

8. Verify the workspace directory. Go to PetClinic-Test's target directory. If war file is there from past build, then remove it.



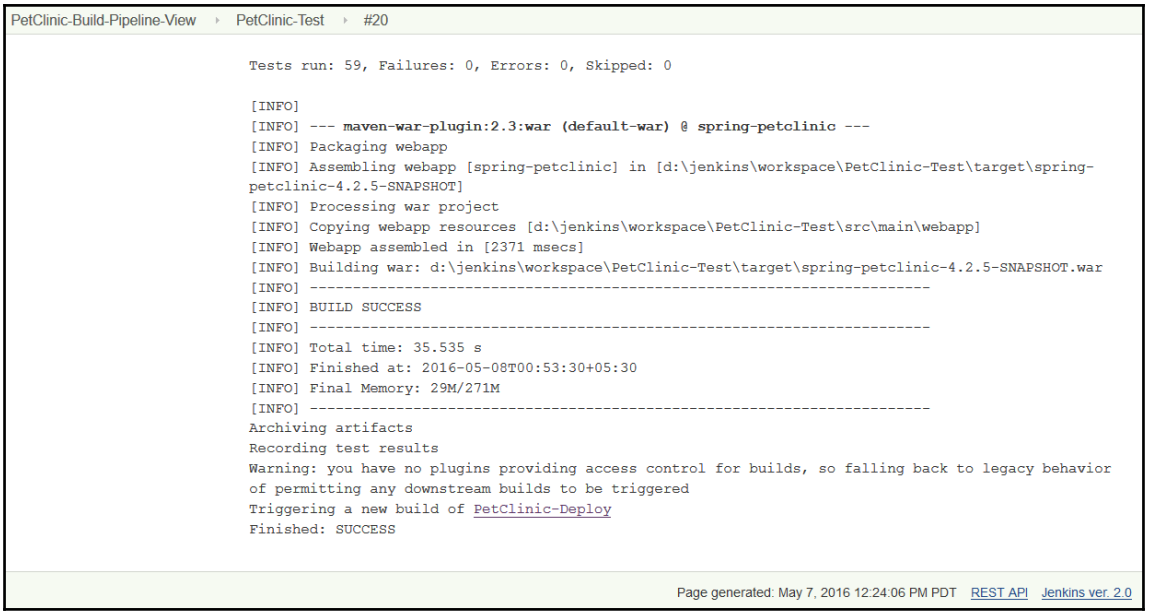

9. Verify the target directory of PetClinic-Deploy folder. No WAR file is available.




10. Add PetClinic-Deploy as Downstream project in the PetClinic-Test. Run the Build Pipeline.



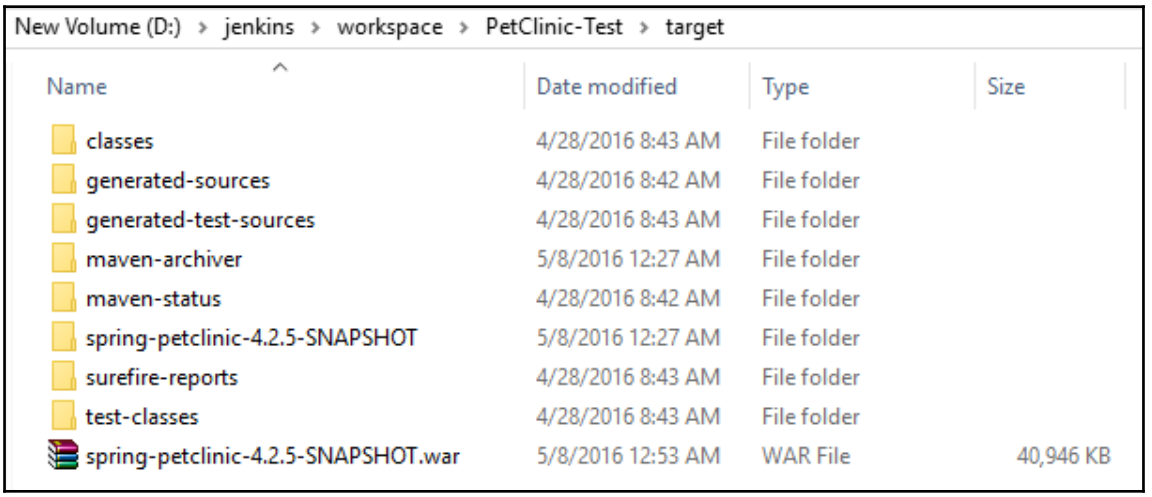

11. Verify the execution of Build Pipeline. Click on the Light box of any build job available in the Build Pipeline. Verify the PetClinic-Test console output.



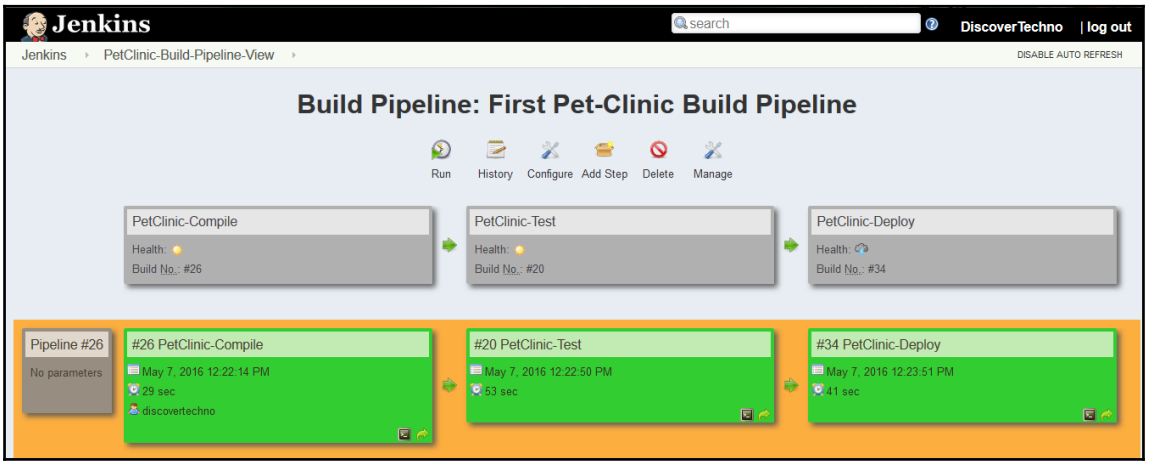

Once PetClinic-Test build job is execution is completed then:

- Verify the target folder in workspace.
- We will see WAR file in the target directory as shown in the below

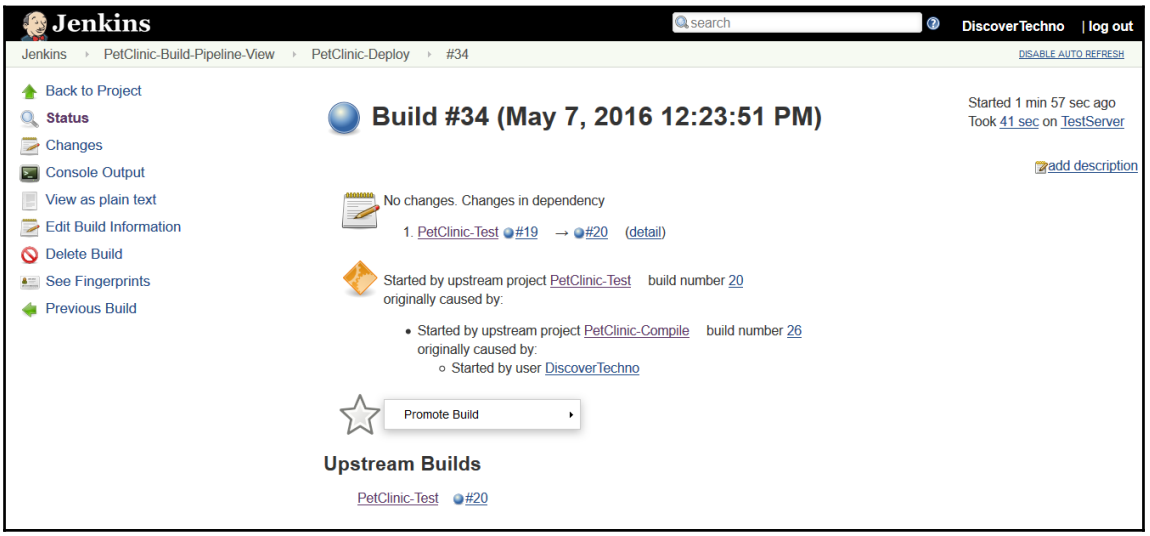
image.




1. Verify the execution of PetClinic-Deploy build job.

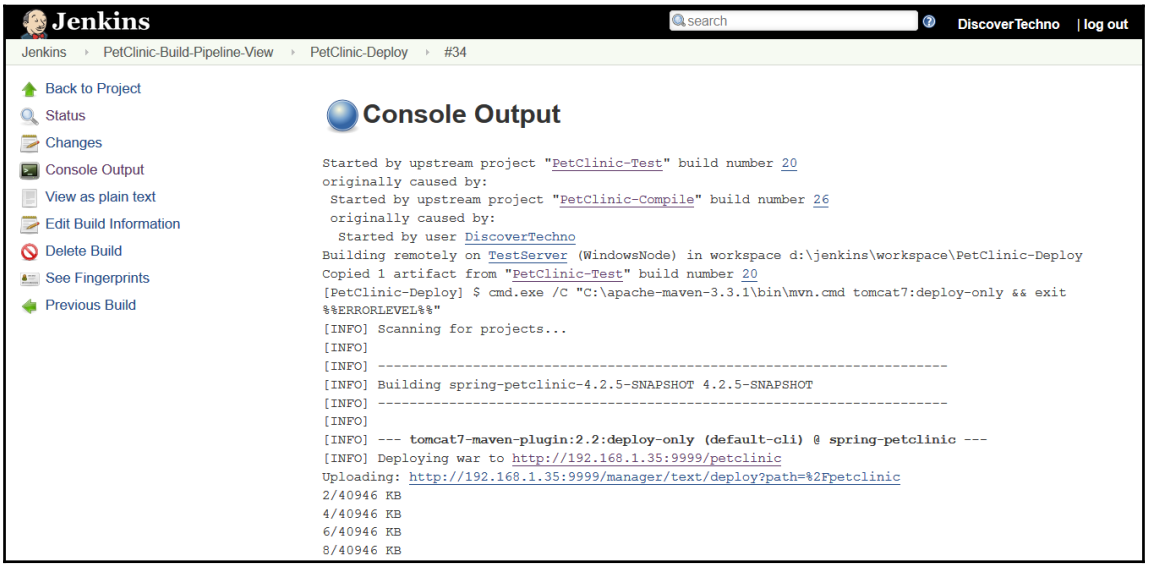



2. Verify the build job's status in the Jenkins dashboard.

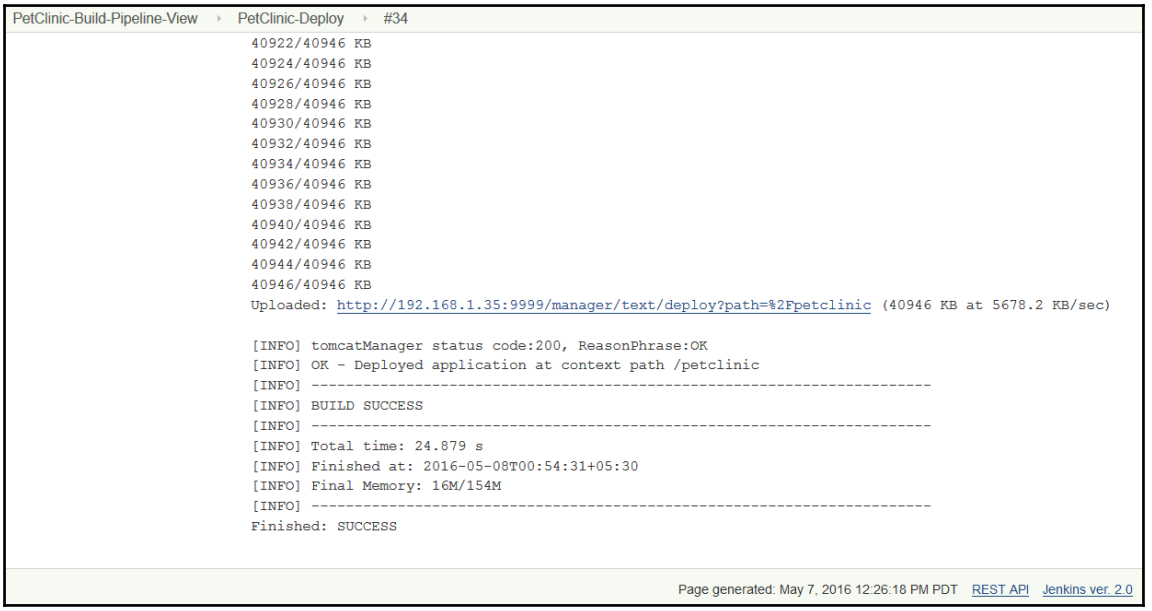



3. Click on the light box of Build pipeline view, it will direct us to console output of specific build job. Click on PetCLinic-Deploy lighbox.

4. Verify the Console output.



5. Verify the successfully uploaded file as per the configuration.




For the self exercise, try to use build flow plugin.

# Self-Test Questions

1. Which feature is one of the Highlights of Jenkins 2 release?

a. Built-in support for continuous integration

b. Built-in support for JUnit

c. Built-in support for delivery pipelines

d. Built-in support for Apache Maven

2. Which language is used to create delivery pipelines ?

a. Java

b. C++

c. C#

d. Domain-specific language

3. In Build Pipeline plugin, what is the significance of Blue color?

a. Indicates Failed execution of Build Job

b. Indicates Successful execution of Build Job

c. Indicates Build Job that hasn't been executed

d. Indicates Running Build Job

# Summary

In this chapter, we have covered latest feature of Jenkins 2 that is one of the Highlights of Jenkins 2 release – Built-in support for delivery pipelines. We have described in details how to use it. It has covered simple groovy script to build a job, to generate a build step, to archive build job artifacts, to run build step on a specific node, to mark definite sections of a build as being controlled by limited concurrency and so on. We have provided a scenario where we want to execute different stages on different nodes. The other similar type of plugin is installed and configured with example – Build Pipeline plugin.

In the next chapter, we will discuss about one of the important pillar of DevOps culture and that is Configuration Management using Chef. First we will see how to install Chef workstation and configure it with Hosted Chef. We will consider installing tomcat using community cookbooks of tomcat installation.