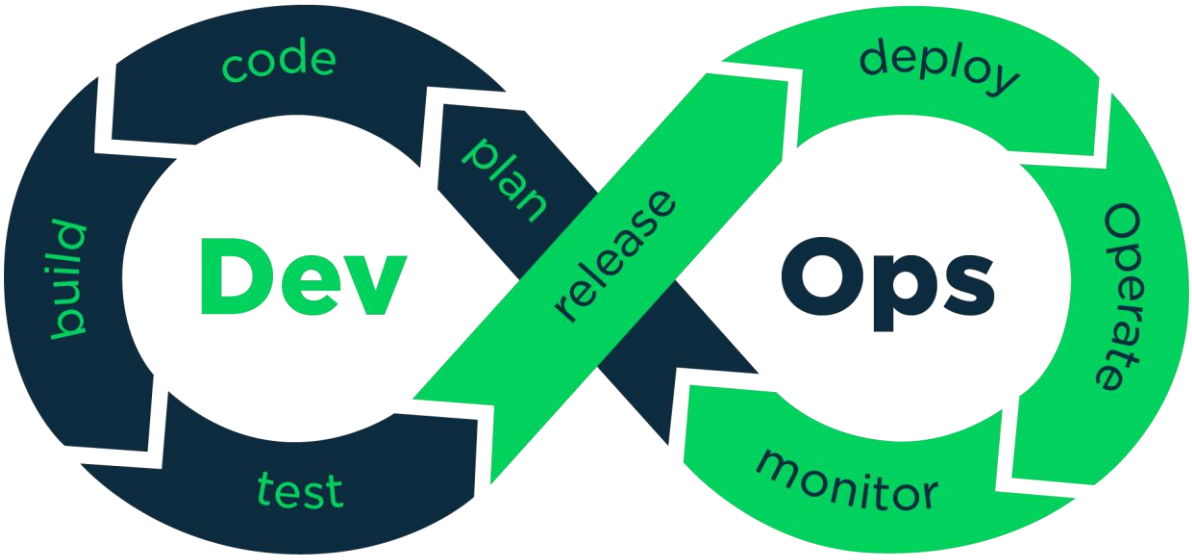

code
plan
build
Dev
test
release
deploy
Operate
Ops
monitor

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

Software Development Process

In software engineering, a software development process is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle. The methodology may include the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application.

Software process (also known as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

Most modern development processes can be vaguely described as agile. Other methodologies include waterfall, prototyping, iterative and incremental development, spiral development, rapid application development, and extreme programming.

Some people consider a life-cycle "model" a more general term for a category of methodologies and a software development "process" a more specific term to refer to a specific process chosen by a specific organization. For example, there are many specific software development processes that fit the spiral life-cycle model. The field is often considered a subset of the systems development life cycle.

Any software process must include the following four activities:

- **Software specification (or requirements engineering):** Define the main functionalities of the software and the constrains around them.
- **Software design and implementation**: The software is to be designed and programmed.
- **Software verification and validation:** The software must conform to its specification and meets the customer needs.
- **Software evolution (software maintenance):** The software is being modified to meet customer and market requirements changes.

In practice, they include sub-activities such as requirements validation, architectural design, unit testing, etc. There are also supporting activities such as configuration and change management, quality assurance, project management, user experience. Along with other activities aim to improve the above activities by introducing new techniques, tools, following the best practice, process standardization (so the diversity of software processes is reduced) etc.

When we talk about a process, we usually talk about the activities in it. However, a process also includes the process description, which includes:

1. **Products**: It is the outcome of an activity. For example, the outcome of architectural designs maybe a model for the software architecture.
2. **Roles**: The responsibilities of the people involved in the process. For example, the project manager, programmer, etc.
3. **Pre and post conditions**: The conditions that must be true before and after an activity. For example, the pre-condition of the architectural design is the requirements have been approved by the customer, while the post condition is the diagrams describing the architectural have been reviewed.

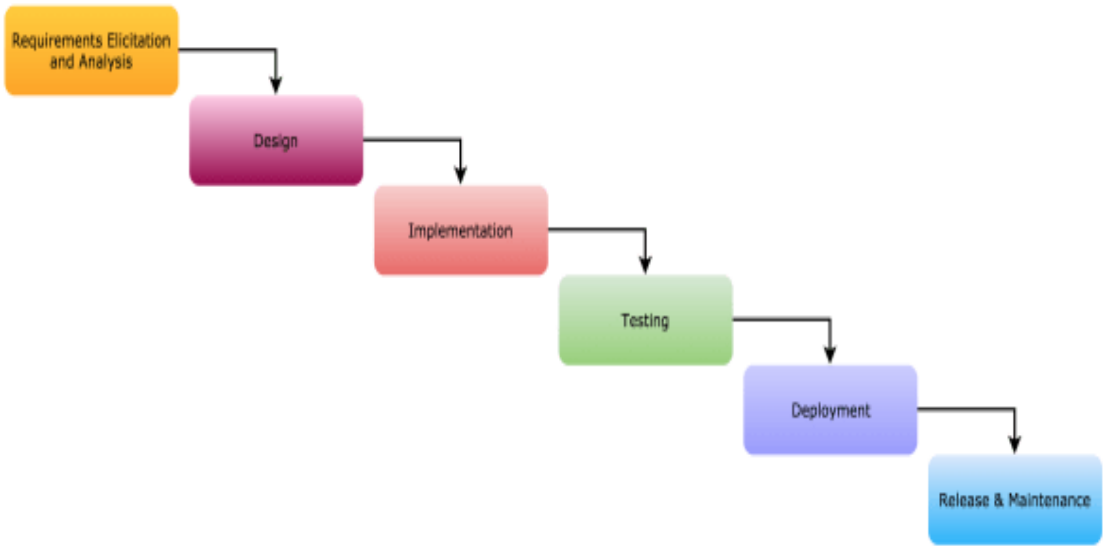
Software process is complex, it relies on making decisions. There's no ideal process and most organizations have developed their own software process. For example, an organization works on critical systems has a very structured process, while with business systems, with rapidly changing requirements, a less formal, flexible process is likely to be more effective.

Waterfall model

Before devops organization were using this particular development model methodology and this is first documented in 1970 and this is first public documented lifecycle model. It describes the development method that is linear and sequential. It has distinct goals for each phase of development. Imagine a waterfall on the cliff of a steep mountain. Once the water has flowed over the edge of the cliff, it cannot turn back. The same is the case of waterfall development strategy as well. An application goes to the next stage only when previous stage is complete.

Traditional Waterfall Model

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.



The sequential phases in Waterfall model are −

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Disadvantages of waterfall model

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

AGILE Methodology

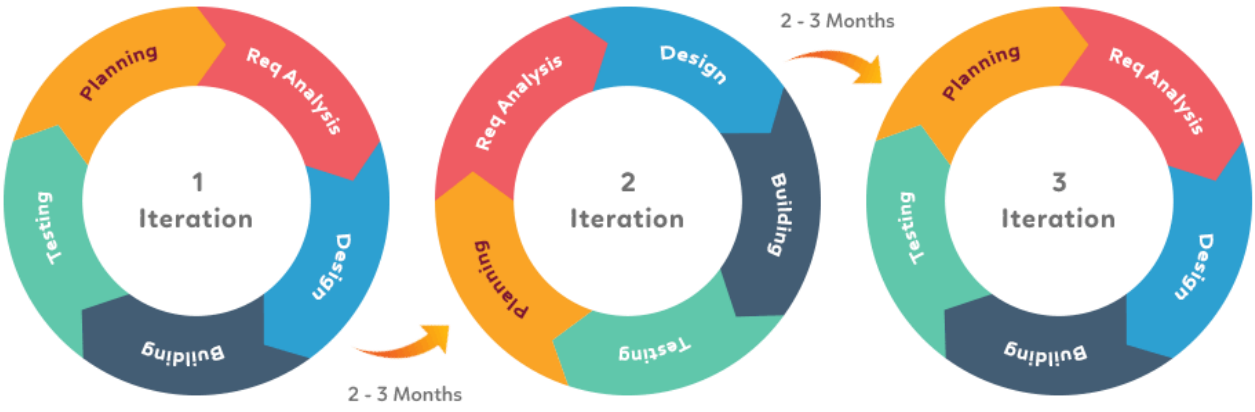
Agile software development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s).It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change. The term agile (sometimes written Agile) was popularized, in this context, by the Manifesto for Agile Software Development. The values and principles espoused in this manifesto were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban.

There is significant anecdotal evidence that adopting agile practices and values improves the agility of software professionals, teams and organizations; however, some empirical studies have found no scientific evidence. AGILE methodology is a practice that promotes continuous iteration of development and testing throughout the software development lifecycle of the project. Both development and testing activities are concurrent unlike the Waterfall model

The agile software development emphasizes on four core values.

1. Individual and team interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan.

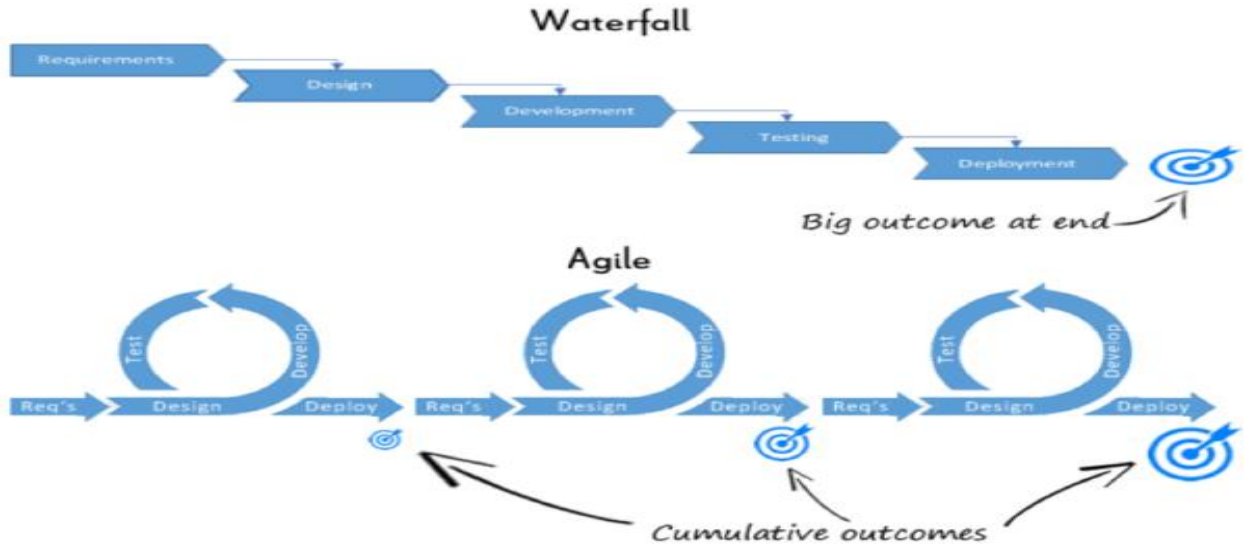
Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.



Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

Agile Vs Traditional SDLC Models



Limitation of Agile

Development and testing is happening continuously and when we talk about deployment it is not continuous, a lots of conflicts will come because of the different production environment.

Collaboration

DevOps attempts to fill these gaps by developing a partnership between the development and operations teams. The DevOps movement emphasizes communication, collaboration, and integration between software developers and IT operations. DevOps promotes collaboration, and collaboration is facilitated by automation and orchestration in order 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 and processes leveraging the benefits of cloud solutions. Agile development and testing methodologies help us meet the goals of continuously integrating, developing, building, deploying, testing, and releasing applications. It provides a mechanism for constant feedback from different teams and stakeholders. It also provides transparency in the form of a platform

for collaboration across teams, such as business analysts, developers, and testers. In short, agile and DevOps are compatible and increase each other's value.

What is DevOps?

DevOps is a culture which promotes collaboration between Development and Operations Team to deploy code to production faster in an automated & repeatable way. The word 'DevOps' is a combination of two words 'development' and 'operations.'

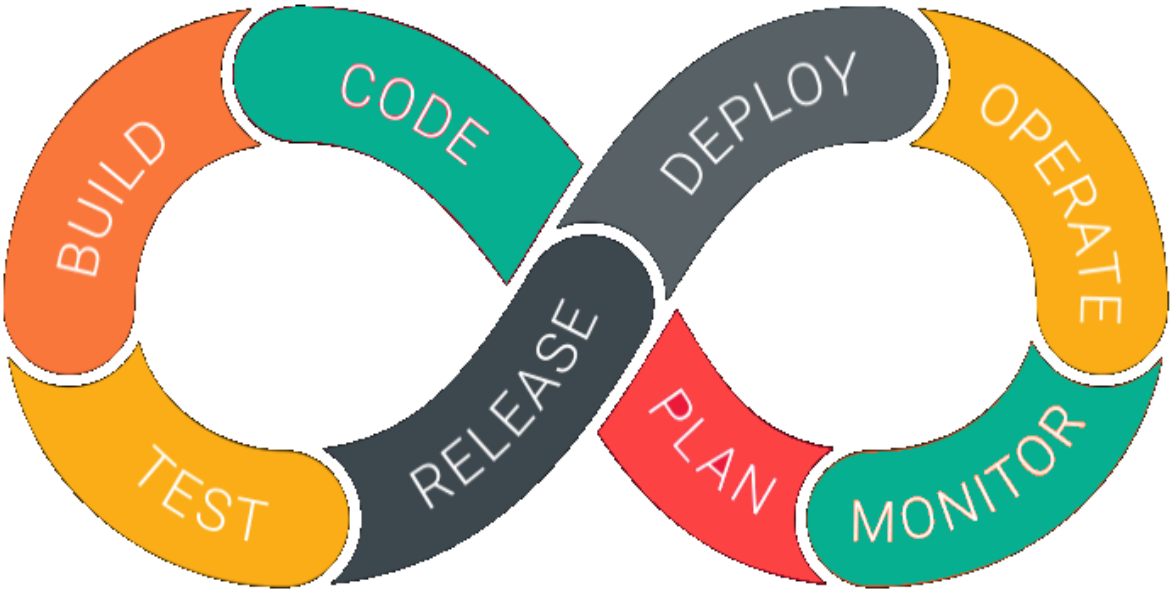
DevOps helps to increases an organization's speed to deliver applications and services. It allows organizations to serve their customers better and compete more strongly in the market.

In simple words, DevOps can be defined as an alignment of development and IT operations with better communication and collaboration.

Why DevOps is Needed?

- Before DevOps, the development and operation team worked in complete isolation.
- Testing and Deployment were isolated activities done after design-build. Hence they consumed more time than actual build cycles.
- Without using DevOps, team members are spending a large amount of their time in testing, deploying, and designing instead of building the project.
- Manual code deployment leads to human errors in production
- Coding & operation teams have their separate timelines and are not in synch causing further delays.

DevOps Lifecycle



DevOps is deep integration between development and operations. Understanding DevOps is not possible without knowing DevOps lifecycle.

Here is brief information about the Continuous DevOps life-cycle:

1. Development

In this DevOps stage the development of software takes place constantly. In this phase, the entire development process is separated into small development cycles. This benefits DevOps team to speed up software development and delivery process.

2. Testing

QA team use tools like Selenium to identify and fix bugs in the new piece of code.

3. Integration

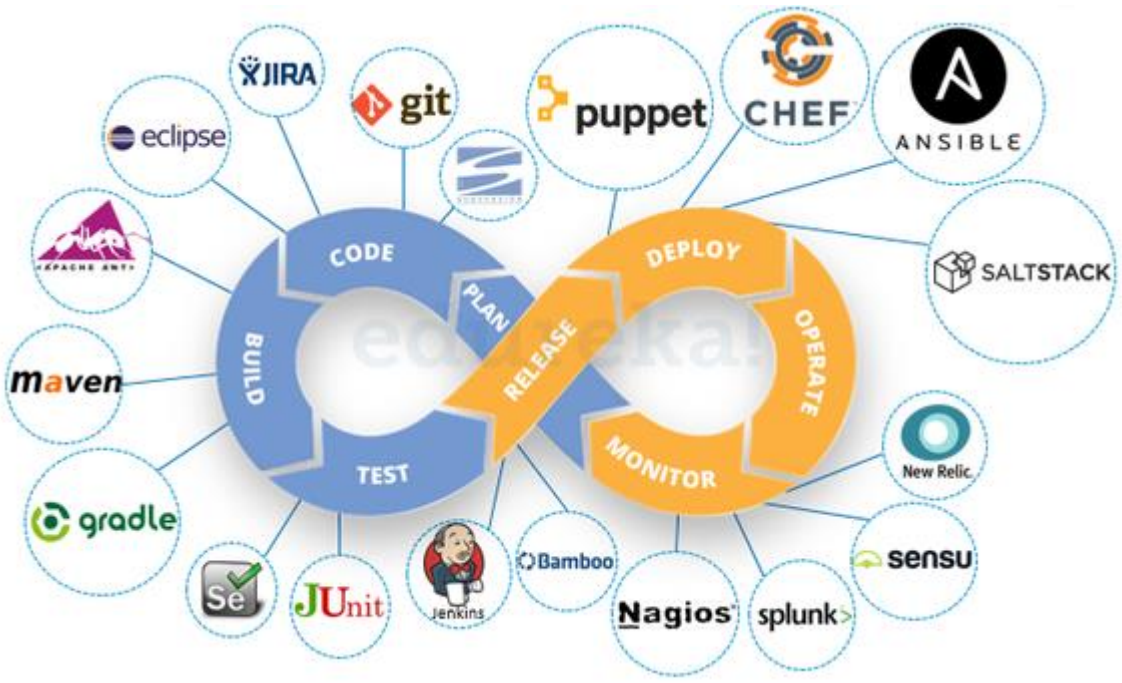
In this stage, new functionality is integrated with the prevailing code, and testing takes place. Continuous development is only possible due to continuous integration and testing.

4. Deployment

In this phase, the deployment process takes place continuously. It is performed in such a manner that any changes made any time in the code, should not affect the functioning of high traffic website.

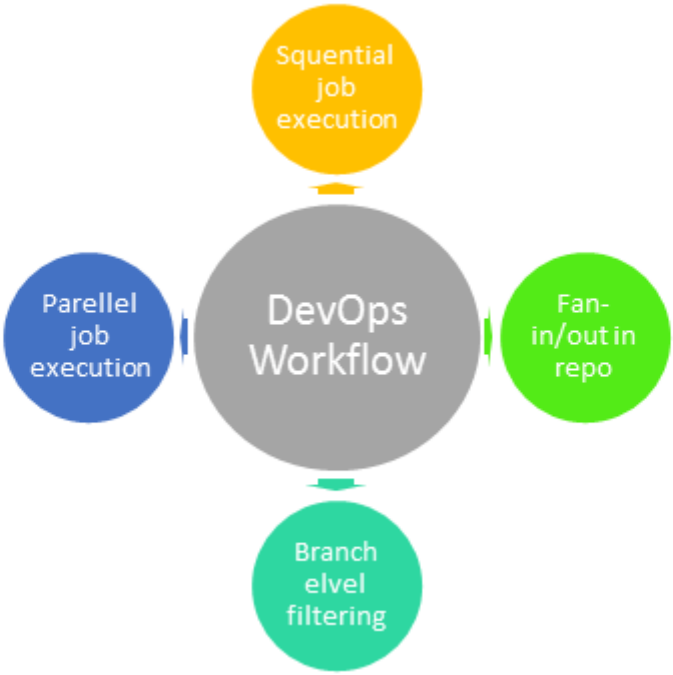
5. Monitoring

In this phase, operation team will take care of the inappropriate system behavior or bugs which are found in production.



DevOps Work Flow

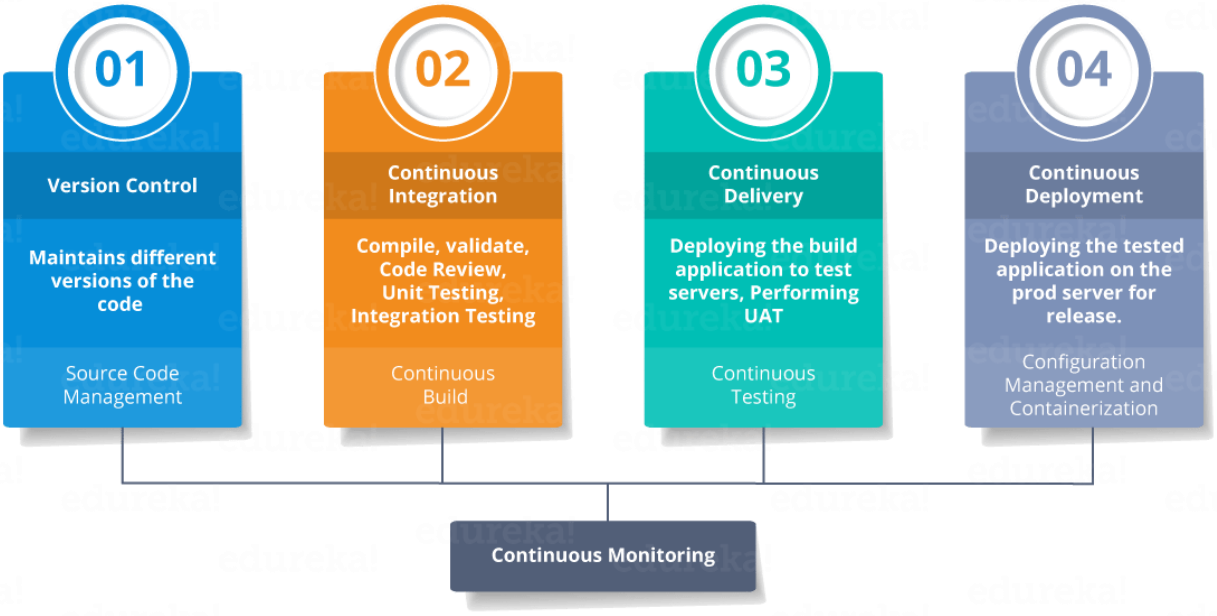
Workflows provide a visual overview of the sequence in which input is provided. It also tells about actions are performed, and output is generated for an operations process.



Workflow allows the ability to separate and arrange jobs which are top-requested by the users. It also gives the ability to mirror their ideal process in the configuration jobs.

Devops Staging

A staging environment (stage) is a nearly exact replica of a production environment for software testing. Staging environments are made to test codes, builds, and updates to ensure quality under a production-like environment before application deployment.



DevOps Automation Tools

It is vital to automate all the testing processes and configure them to achieve speed and agility. This process is known as DevOps automation. The difficulty faced in large DevOps Team that maintain large huge IT infrastructure can be classified briefly into six different categories.

1. Infrastructure Automation
2. Configuration Management
3. Deployment Automation
4. Performance Management
5. Log Management
6. Monitoring.

Let's see a few tools in each of these categories and how they solve the pain points–

Infrastructure Automation

Amazon Web Services (AWS): Being cloud service you do not need to be physically present in the data center. Also, they are easy to scale on-demand. There are no up-front hardware costs. It can be configured to provision more servers based on traffic automatically.

Configuration Management

Chef: It is a useful DevOps tool for achieving speed, scale, and consistency. It can be used to ease out complex tasks and perform configuration management. With this tool, DevOps team can avoid making changes across ten thousand servers. Instead, they need to make changes in one place which is automatically reflected in other servers.

Deployment Automation

Jenkins: This tool facilitates continuous integration and testing. It helps to integrate project changes more easily by quickly finding issues as soon as a built is deployed.

Log Management

Splunk: This is a tool solves the issues like aggregating, storing, and analyzing all logs in one place.

Performance Management

App Dynamic: It is DevOps tool which offers real-time performance monitoring. The data collected by this tool helps developers to debug when issues occur.

Monitoring

Nagios: It is also important to make sure people are notified when infrastructure and related services go down. Nagios is one such tool for this purpose which helps DevOps teams to find and correct problems.

Introduction into version control systems

What is a version control system?

A version control system (VCS) allows you to track the history of a collection of files. It supports creating different versions of this collection. Each version captures a snapshot of the files at a certain point in time and the VCS allow you to switch between these versions. These versions are stored in a specific place, typically called a repository.

You may, for example, revert the collection of files to a state from 2 days ago. Or you may switch between versions of your files for experimental features. The process of creating different versions (snapshots) in the repository is depicted in the following graphic. Please note that this picture fits primarily to Git. Other version control systems like Concurrent Versions System (CVS) don't create snapshots of the files but store file deltas.

VCS are typically used to track changes in text files. These text files can for example be source code for a programming language, HTML or configuration files. Of course, version control systems are not limited to text files; they can also handle other types of files. For example, you may use a VCS to track the different versions of a png file.

Localized and centralized version control systems

A localized version control system keeps local copies of the files. This approach can be as simple as creating a manual copy of the relevant files.

A centralized version control system provides a server software component which stores and manages the different versions of the files. A developer can copy (checkout) a certain version from the central sever onto their individual computer.

Both approaches have the drawback that they have one single point of failure. In a localized version control systems it is the individual computer and in a centralized version control systems it is the server machine. Both systems make it also harder to work in parallel on different features.

Distributed version control systems

In a distributed version control system each user has a complete local copy of a repository on his individual computer. The user can copy an existing repository. This copying process is typically called cloning and the resulting repository can be referred to as a clone.

Every clone contains the full history of the collection of files and a cloned repository has the same functionality as the original repository.

Every repository can exchange versions of the files with other repositories by transporting these changes. This is typically done via a repository running on a server which is, unlike the local machine of a developer, always online. Typically, there is a central server for keeping a repository but each cloned repository is a full copy of this repository. The decision which of the copies is considered to be the central server repository is pure convention.

Introduction into Git

What is Git?

Git is currently the most popular implementation of a distributed version control system. Git originates from the Linux kernel development and was founded in 2005 by Linus Torvalds. Nowadays it is used by many popular open source projects, e.g., the Android or the Eclipse developer teams, as well as many commercial organizations.

The core of Git was originally written in the programming language C, but Git has also been re-implemented in other languages, e.g., Java, Ruby and Python.

Git repositories

A Git repository contains the history of a collection of files starting from a certain directory. The process of copying an existing Git repository via the Git tooling is called cloning. After cloning a repository the user has the complete repository with its history on his local machine. Of course, Git also supports the creation of new repositories.

If you want to delete a Git repository, you can simply delete the folder which contains the repository.

If you clone a Git repository, by default, Git assumes that you want to work in this repository as a user. Git also supports the creation of repositories targeting the usage on a server.

- Bare repositories are supposed to be used on a server for sharing changes coming from different developers. Such repositories do not allow the user to modify locally files and to create new versions for the repository based on these modifications.
- Non-bare repositories target the user. They allow you to create new changes through modification of files and to create new versions in the repository. This is the default type which is created if you do not specify any parameter during the clone operation.

A local non-bare Git repository is typically called local repository.

Working tree

A local repository provides at least one collection of files which originate from a certain version of the repository. This collection of files is called the working tree. It corresponds to a checkout of one version of the repository with potential changes done by the user.

The user can change the files in the working tree by modifying existing files and by creating and removing files.

A file in the working tree of a Git repository can have different states. These states are the following:

- **Untracked:** the file is not tracked by the Git repository. This means that the file never staged nor committed.
- **Tracked:** committed and not staged
- **Staged:** staged to be included in the next commit
- **Dirty / modified:** the file has changed but the change is not staged

After doing changes in the working tree, the user can add these changes to the Git repository or revert these changes.

Adding to a Git repository via staging and committing

After modifying your working tree you need to perform the following two steps to persist these changes in your local repository:

- Add the selected changes to the staging area (also known as index) via the `git add` command
- Commit the staged changes into the Git repository via the `Git commit` command.

This process is depicted in the following graphic.

The `Git add` command stores a snapshot of the specified files in the staging area. It allows you to incrementally modify files, stage them, modify and stage them again until you are satisfied with your changes. Some tools and Git user prefer the usage of the index instead of staging area. Both terms mean the same thing.

After adding the selected files to the staging area, you can commit these files to add them permanently to the Git repository. Committing creates a new persistent snapshot (called commit or commit object) of the staging area in the Git repository. A commit object, like all objects in Git, is immutable.

The staging area keeps track of the snapshots of the files until the staged changes are committed. For committing the staged changes you use the `Git commit` command. If you commit changes to your Git repository, you create a new commit object in the Git repository.

Synchronizing with other Git repositories (remote repositories)

Git allows the user to synchronize the local repository with other (remote) repositories.

Users with sufficient authorization can send new version in their local repository to remote repositories via the push operation. They can also integrate changes from other repositories into their local repository via the fetch and pull operation.

The concept of branches

Git supports branching which means that you can work on different versions of your collection of files. A branch allows the user to switch between these versions so that he can work on different changes independently from each other.

For example, if you want to develop a new feature, you can create a branch and make the changes in this branch. This does not affect the state of your files in other branches. For example, you can work independently on a branch called production for bug fixes and on another branch called `feature_123` for implementing a new feature.

Branches in Git are local to the repository. A branch created in a local repository does not need to have a counterpart in a remote repository. Local branches can be compared with other local branches and with remote-tracking branches. A remote-tracking branch proxies the state of a branch in another remote repository.

Git supports the combination of changes from different branches. The developer can use Git commands to combine the changes at a later point in time.

Reference: - https://www.vogella.com/tutorials/Git/article.html

Introduction into GitLab

What is GitLab?

Before we dive into definition for GitLab, first we need to understand few terminologies. We often come across these terms like Git, Gitlab, GitHub, and Bit bucket. Let's see definition of all these as below −

Git - It is a source code versioning system that lets you locally track changes and push or pull changes from remote resources.

GitLab, GitHub, and Bit bucket - Are services that provides remote access to Git repositories. In addition to hosting your code, the services provide additional features designed to help manage the software development lifecycle. These additional features include managing the sharing of code between different people, bug tracking, wiki space and other tools for 'social coding'.

- **GitHub** is a publicly available, free service which requires all code (unless you have a paid account) be made open. Anyone can see code you push to GitHub and offer suggestions for improvement. GitHub currently hosts the source code for tens of thousands of open source projects.
- **GitLab** is a GitHub like service that organizations can use to provide internal management of Git repositories. It is a self-hosted Git-repository management system that keeps the user code private and can easily deploy the changes of the code.

History

GitLab was found by Dmitriy Zaporozhets and Valery Sizov in October 2011. It was distributed under MIT license and the stable version of GitLab is 10.4 released in January 22, 2018.

Why to use GitLab?

GitLab is great way to manage git repositories on centralized server. GitLab gives you complete control over your repositories or projects and allows you to decide whether they are public or private for free.

Features

- GitLab hosts your (private) software projects for free.
- GitLab is a platform for managing Git repositories.
- GitLab offers free public and private repositories, issue-tracking and wikis.
- GitLab is a user friendly web interface layer on top of Git, which increases the speed of working with Git.
- GitLab provides its own Continuous Integration (CI) system for managing the projects and provides user interface along with other features of GitLab.

Advantages

- GitLab provides GitLab Community Edition version for users to locate, on which servers their code is present.
- GitLab provides unlimited number of private and public repositories for free.
- The Snippet section can share small amount of code from a project, instead of sharing whole project.

Disadvantages

- While pushing and pulling repositories, it is not as fast as GitHub.
- GitLab interface will take time while switching from one to another page.

You can install the GitLab runner on different operating systems, by installing Git versioning system and creating user account in the GitLab site.

Git is a version control system used for −

- Handling the source code history of projects
- Tracking changes made to files
- Handling small and large projects with speed and efficiency
- To collaborate with other developers on different projects.

GitLab is a Git-based platform provides remote access to Git repositories and helpful for software development cycle by creating private and public repositories for managing the code.

GitLab supports different types of operating systems such as Windows, Ubuntu, Debian, CentOS, open SUSE and Raspberry Pi 2. In this chapter, we will discuss about how to install GitLab on Windows and Ubuntu operating systems −

Installation of GitLab on Windows:

Step 1 − First create a folder called 'GitLab-Runner' in your system. For instance, you can create in C drive as C:\GitLab-Runner.

Step 2 − Now download the binary for x86 or amd64 and copy it in the folder created by you. Rename the downloaded binary to gitlab-runner.exe.

Step 3 − Open the command prompt and navigate to your created folder. Now type the below command and press enter.

```
C:\GitLab-Runner>gitlab-runner.exe register
```

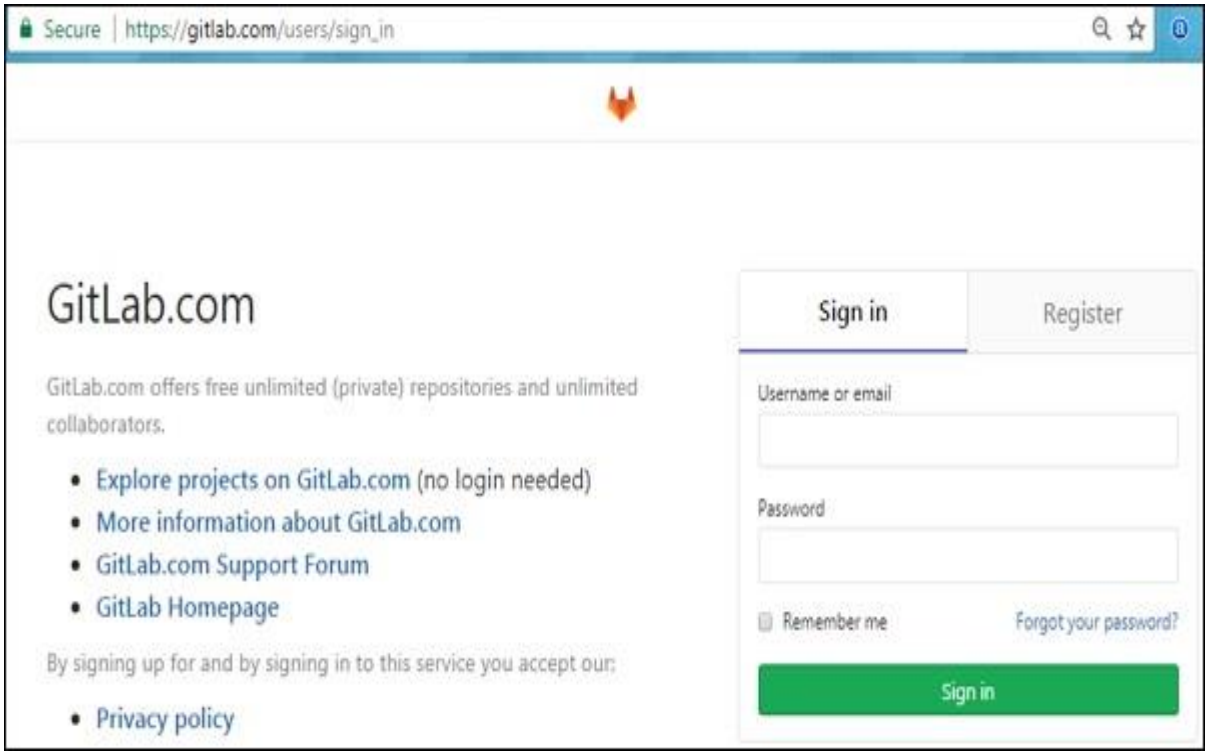
Step 4 − After running the above command, it will ask to enter the GitLab-ci coordinator URL.

```
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://gitlab.com
```

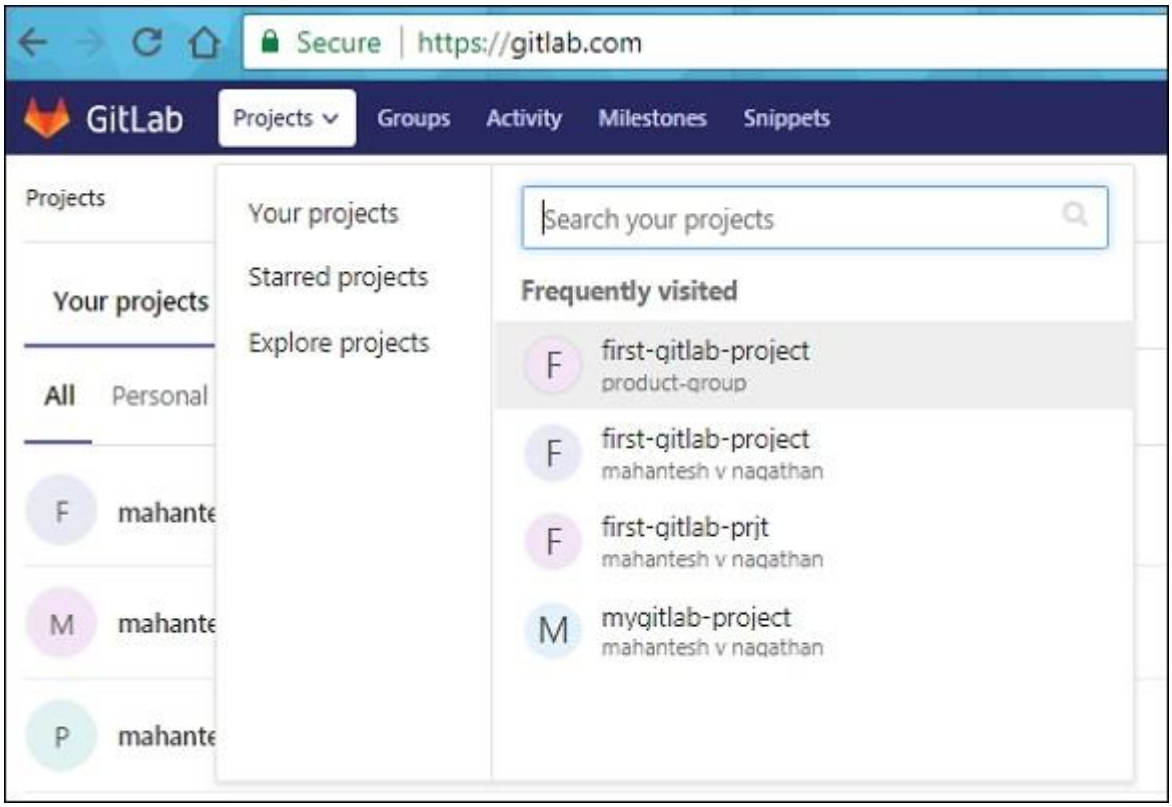
Step 5 − Enter the gitlab-ci token for the runner.

```
Please enter the gitlab-ci token for this runner:
xxxxx
```

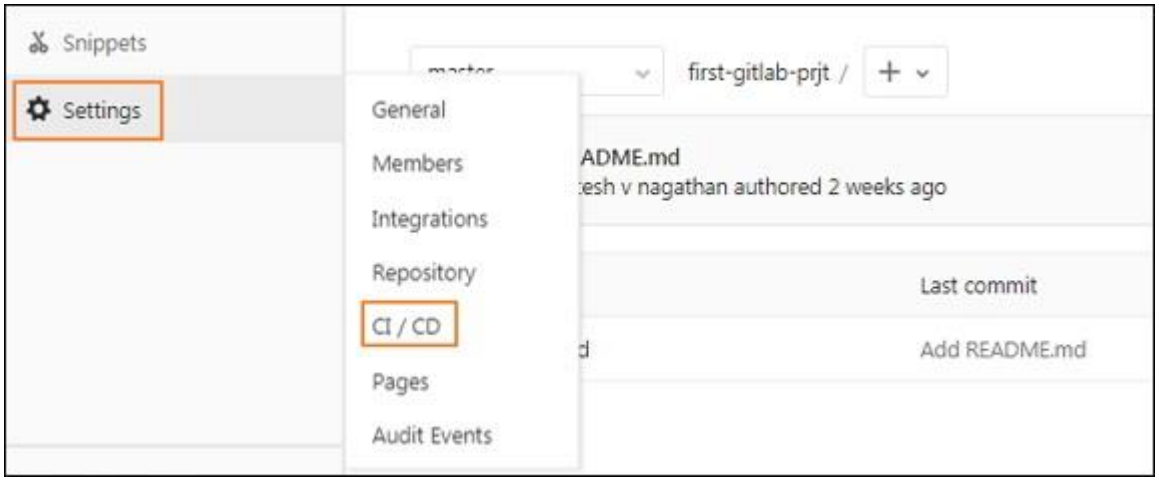
- To get the token, login to your GitLab account −



- Now go to your project –



- Click on the CI/CD option under Settings tab and expand the Runners Settings option.

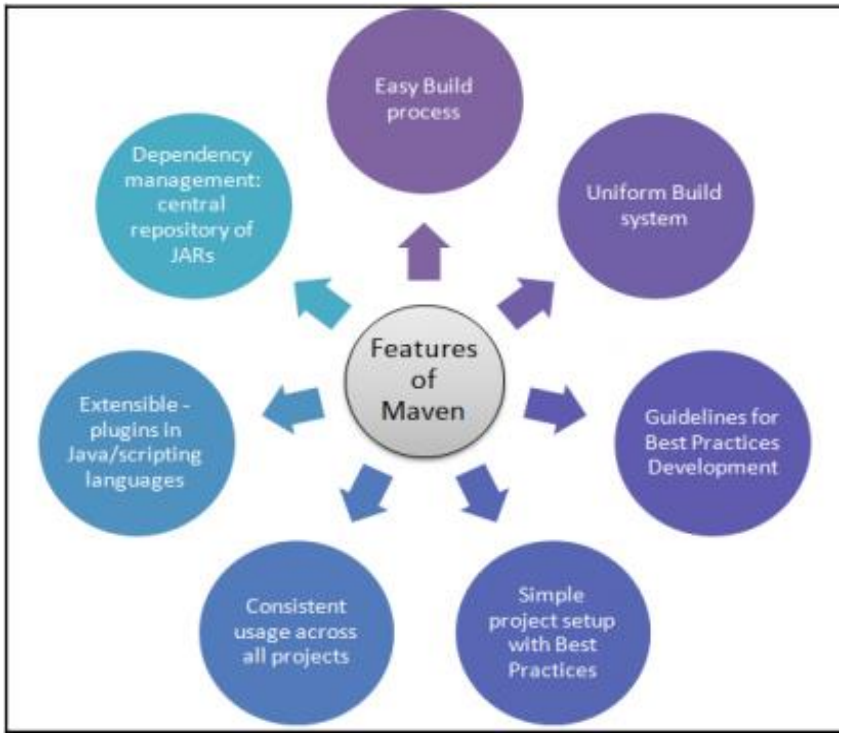


Introduction into MAVEN

What is Maven?

Apache Maven is a build tool with the Apache 2.0 license. It is used for Java projects and can be used in a cross-platform environment. It can be also 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 about the name of the application, owner information, how the application distribution file can be created, and how dependencies can be managed.

Build Lifecycle Basics

Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined.

For the person building a project, this means that it is only necessary to learn a small set of commands to build any Maven project, and the POM will ensure they get the results they desired. There are three built-in build lifecycles: default, clean and site. The default lifecycle handles your project deployment; the clean lifecycle handles project cleaning, while the site lifecycle handles the creation of your project's site documentation.

A Build Lifecycle is made up of Phases. Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the Lifecycle Reference):

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

These lifecycle phases (plus the other lifecycle phases not shown here) are executed sequentially to complete the default lifecycle. Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.

Introduction into JENKINS

Jenkins is an open source Continuous Integration server capable of orchestrating a chain of actions that help to achieve the Continuous Integration process (and not only) in an automated fashion. Jenkins is entirely written in Java. Jenkins is a widely used application around the world that has around 300k installations and growing day by day.

It is a server-based application and requires a web server like Apache Tomcat. The reason Jenkins became so popular is that of its monitoring of repeated tasks which arise during the development of a project. For example, if your team is developing a project, Jenkins will continuously test your project builds and show you the errors in early stages of your development.

By using Jenkins, software companies can accelerate their software development process, as Jenkins can automate build and test at a rapid rate. Jenkins supports the complete development lifecycle of software from building, testing, documenting the software, deploying and other stages of a software development lifecycle.

Why Continuous Integration with Jenkins?

Some people might think that the old-fashioned way of developing the software is the better way. Let's understand the advantages of CI with Jenkins with the following example

Let us imagine that there are around 10 developers who are working on a shared repository. Some developer completes their task in 25 days while others take 30 days to complete.

- Once all Developers had completed their assigned coding tasks, they used to commit their code all at same time. Later, Build is tested and deployed but after Jenkins the code is built and tests as soon as Developer commits code. Jenkin will build and test code many times during the day
- Code commit built, and test cycle was very infrequent, and a single build was done after many days but after the Jenkins If the build is successful, then Jenkins will deploy the source into the test server and notifies the deployment team.

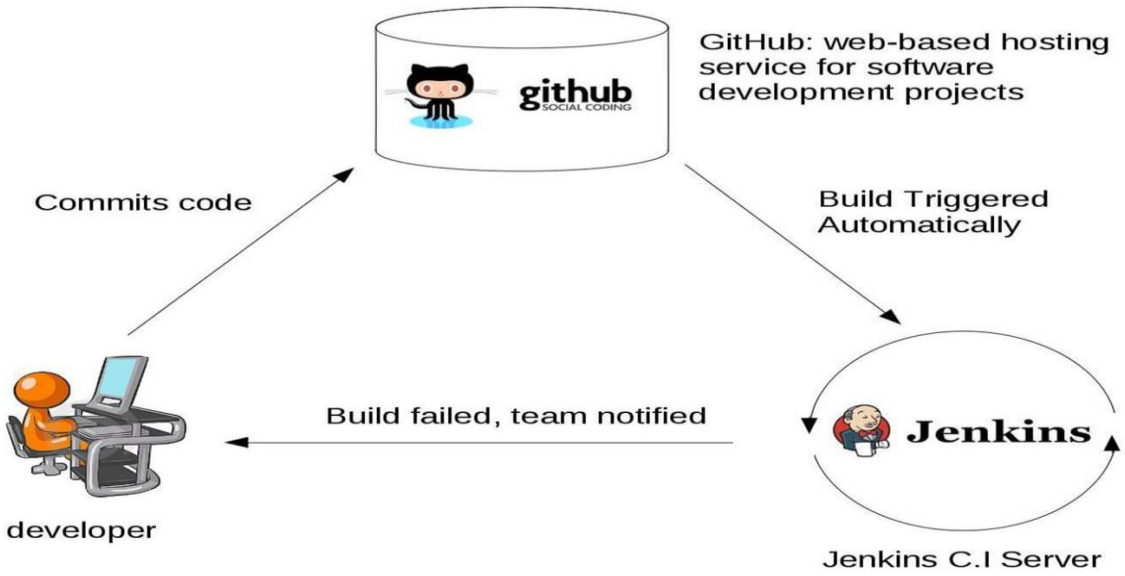
  If the build fails, then Jenkins will notify the errors to the developer team.
- Since the code was built all at once, some developers would need to wait until other developers finish coding to check their build and after Jenkins The code is built immediately after any of the Developer commits.

- It is not an easy task to isolate, detect, and fix errors for multiple commits and after it Since the code is built after each commit of a single developer, it's easy to detect whose code caused the built to fail
- Development Cycle is slow but now the development cycle is fast. New features are more readily available to users. Increases profits.
- The code is deployed once all the errors are fixed and tested and now The code is deployed after every successful build and test.

Real-world case study of Continuous Integration

I am sure all of you aware of old phone Nokia. Nokia used to implement a procedure called nightly build. After multiple commits from diverse developers during the day, the software built every night. Since the software was built only once in a day, it's a huge pain to isolate, identify, and fix the errors in a large code base.

Later, they adopted Continuous Integration approach. The software was built and tested as soon as a developer committed code. If any error is detected, the respective developer can quickly fix the defect.



Advantages of using Jenkins

- Jenkins is being managed by the community which is very open. Every month, they hold public meetings and take inputs from the public for the development of Jenkins project.
- So far around 280 tickets are closed, and the project publishes stable release every three months.
- As technology grows so does Jenkins. So far Jenkins has around 320 plugins published in its plugins database. With plugins, Jenkins becomes even more powerful and feature rich.
- Jenkins also supports cloud-based architecture so that you can deploy Jenkins in cloud-based platforms.
- The reason why Jenkins became popular is that it was created by a developer for developers.

Prerequisites before installing Jenkins Server

How to Install Java on CentOS and Fedora

Introduction

This tutorial will show you how to install Java on CentOS 7 (also 6 and 6.5), modern Fedora releases, and RHEL. Java is a popular software platform that allows you to run Java applications and applets.

The installation of the following versions of Java is covered:

- OpenJDK 8
- OpenJDK 7

Feel free to skip to your desired section using the **Contents** button on the sidebar!

Prerequisites

Before you begin this guide, you should have a regular, non-root user with sudo privileges configured on both of your servers--this is the user that you should log in to your servers as. You can learn how to configure a regular user account by following the steps in our initial server setup guide for Centos 7.

Variations of Java

There are three different editions of the Java Platform: Standard Edition (SE), Enterprise Edition (EE), and Micro Edition (ME). This tutorial is focused on Java SE (Java Platform, Standard Edition).

There are two different Java SE packages that can be installed: the Java Runtime Environment (JRE) and the Java Development Kit (JDK). JRE is an implementation of the Java Virtual Machine (JVM), which allows you to run compiled Java applications and applets. JDK includes JRE and other software that is required for writing, developing, and compiling Java applications and applets.

There are also two different implementations of Java: OpenJDK and Oracle Java. Both implementations are based largely on the same code but OpenJDK, the reference implementation of Java, is fully open source while Oracle Java contains some proprietary code. Most Java applications will work fine with either but you should use whichever implementation your software calls for.

You may install various versions and releases of Java on a single system, but most people only need one installation. With that in mind, try to only install the version of Java that you need to run or develop your application(s).

Install OpenJDK 8

This section will show you how to install the prebuilt OpenJDK 8 JRE and JDK packages using the yum package manager, which is similar to apt-get for Ubuntu/Debian. OpenJDK 8 is the latest version of OpenJDK.

Install OpenJDK 8 JRE

To install OpenJDK 8 **JRE** using yum, run this command:

- **sudo yum install java-1.8.0-openjdk**

At the confirmation prompt, enter y then RETURN to continue with the installation.

Install OpenJDK 8 JDK

To install OpenJDK 8 **JDK** using yum, run this command:

- **sudo yum install java-1.8.0-openjdk-devel**

At the confirmation prompt, enter y then RETURN to continue with the installation.

Install OpenJDK 7

This section will show you how to install the prebuilt OpenJDK 7 JRE and JDK packages using the yum package manager.

Install OpenJDK 7 JRE

To install OpenJDK 7 **JRE** using yum, run this command:

- **sudo yum install java-1.7.0-openjdk**

At the confirmation prompt, enter y then RETURN to continue with the installation.

Install OpenJDK 7 JDK

To install OpenJDK 7 **JDK** using yum, run this command:

- **sudo yum install java-1.7.0-openjdk-devel**

At the confirmation prompt, enter y then RETURN to continue with the installation.

Set Default Java

If you installed multiple versions of Java, you may want to set one as your default (i.e. the one that will run when a user runs the java command). Additionally, some applications require certain environment variables to be set to locate which installation of Java to use. This section will show you how to do this.

By the way, to check the version of your default Java, run this command:

- **java –version**

Using Alternatives

The alternatives command, which manages default commands through symbolic links, can be used to select the default Java command.

To print the programs that provide the java command that are managed by alternatives, use this command:

- **sudo alternatives --config java**

Here is an example of the output:

Output
There are 5 programs which provide 'java'.

```
Selection    Command
  1          ----------------------------------------------
  2          java-1.7.0-openjdk.x86_64 (/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.161-
             2.6.12.0.el7_4.x86_64/jre/bin/java)
  3     *+ 2          java-1.8.0-openjdk.x86_64 (/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-
             5.b12.el7_4.x86_64/jre/bin/java)
  4          /usr/lib/jvm/jre-1.6.0-openjdk.x86_64/bin/java
```

Enter to keep the current selection **[+],** or type selection number:

Simply enter the selection number to choose which java executable should be used by default.

Using Environment Variables

Many Java applications use the JAVA_HOME or JRE_HOME environment variables to determine which java executable to use.

For example, if you installed Java to /usr/java/jdk1.8.0_161/jre/bin (i.e. java executable is located at /usr/java/jdk1.8.0_161/jre/bin/java),

You could set your JAVA_HOME environment variable in a bash shell or script like so:

For example, if you installed Java to /usr/java/jdk1.8.0_161/jre/bin (i.e. java executable is located at /usr/java/jdk1.8.0_161/jre/bin/java), you could set your JAVA_HOME environment variable in a bash shell or script like so:

- **export JAVA_HOME=/usr/java/jdk1.8.0_161/jre**

If you want JAVA_HOME to be set for every user on the system by default, add the previous line to the /etc/environment file. An easy way to append it to the file is to run this command:

- **sudo sh -c "echo export JAVA_HOME=/usr/java/jdk1.8.0_161/jre >> /etc/environment"**

How to Install Apache Maven on CentOS 7?

Prerequisites

- A newly deployed or existing CentOS 7 server instance.
- Java Development Kit (JDK) – Maven 3.3+ require JDK 1.7 or above to execute.

Install Apache Maven in CentOS 7

Next, go to the official Apache Maven download page and grab the latest version or use the following wget command to download it under the maven home directory '/usr/local/src'.

- **cd /usr/local/src**
- **wget http://www-us.apache.org/dist/maven/maven-3/3.5.4/binaries/apache-maven-3.5.4-bin.tar.gz**

Extract the downloaded archive file, and rename it using following commands.

- **tar -xf apache-maven-3.5.4-bin.tar.gz**
- **mv apache-maven-3.5.4/ apache-maven/**

Configure Apache Maven Environment

Now we need to configure the environments variables to pre-compiled Apache Maven files on our system by creating a configuration file 'maven.sh' in the '/etc/profile.d' directory.

- **cd /etc/profile.d/**

- **vim maven.sh**

Add the following configuration in 'maven.sh' configuration file.
Apache Maven Environment Variables:-

MAVEN_HOME for Maven 1 - M2_HOME for Maven 2

- **export M2_HOME=/usr/local/src/apache-maven**
- **export PATH=${M2_HOME}/bin:${PATH}**

Now make the 'maven.sh' configuration file executable and then load the configuration by running the 'source' command.

- **chmod +x maven.sh**
- **source /etc/profile.d/maven.sh**

Check Apache Maven Version

To verify Apache Maven installation, run the following **maven** command.

- **mvn --version**

And you should get an output similar to the following:

```
Apache Maven 3.5.4 (1edded0938998edf8bf061f1ceb3cfdeccf443fe; 2018-06-17T19:33:14+01:00)
Maven home: /usr/local/src/apache-maven
Java version: 9.0.4, vendor: Oracle Corporation, runtime: /opt/java/jdk-9.0.4
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.17.6-1.el7.elrepo.x86_64", arch: "amd64", family: "unix"
```

Create Demo Web Server (Optional)

This is an optional step. If you want to test your node.js install. Let's create a web server with "Welcome Node.js" text. Create a file **demo_server.js**

- **vim demo_server.js**

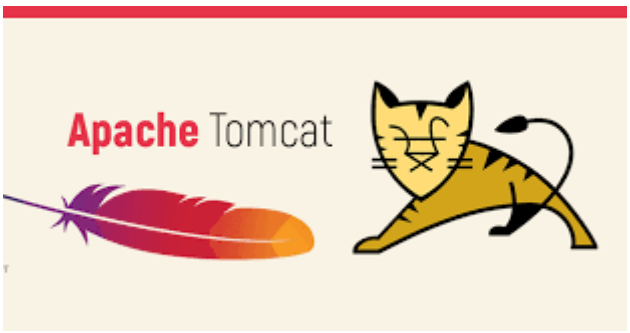
And add the following content

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Welcome Node.js');
}).listen(3001, "127.0.0.1");
console.log('Server running at http://127.0.0.1:3001/');
```

Now start the web server using the command.

```
node --debug demo_server.js

debugger listening on port 5858
Server running at http://127.0.0.1:3001/
```

Web server has been started on port 3001. Now access **http://127.0.0.1:3001/** url in browser.



Introduction into Tomcat

Introduction

Apache Tomcat is a web server and servlet container that is used to serve Java applications. Tomcat is an open source implementation of the Java Servlet and JavaServer Pages technologies, released by the Apache Software Foundation. This tutorial covers the basic installation and some configuration of the latest release of Tomcat 8 on your CentOS 7 server.

Prerequisites

Before you begin with this guide, you should have a separate, non-root user account set up on your server. You can learn how to do this by completing steps 1-3 in the initial server setup for CentOS 7. We will be using the demo user created here for the rest of this tutorial.

How to Install Apache Tomcat 8 on CentOS 7

Install Tomcat

The easiest way to install Tomcat 8 at this time is to download the latest binary release then configure it manually.

Download Tomcat Binary

Find the latest version of Tomcat 8 at the Tomcat 8 Downloads page. At the time of writing, the latest version is **8.5.37**. Under the **Binary Distributions** section, then under the **Core** list, copy the link to the "tar.gz".

Let's download the latest binary distribution to our home directory using wget.

First, install wget using the yum package manager:

- **sudo yum install wget**

Then, change to your home directory:

- **cd /usr/local**

Now, use wget and paste in the link to download the Tomcat 8 archive, like this (your mirror link will probably differ from the example):

- **wget http://mirrors.estointernet.in/apache/tomcat/tomcat-8/v8.5.40/bin/apache-tomcat-8.5.40.zip**

We're going to install Tomcat to the /opt/tomcat directory. Create the directory, and then extract the archive to it with these commands:

- **unzip apache-tomcat-8.5.40.zip**
- **mv apache-tomcat-8.5.40 tomcat8**

Before starting the Tomcat Service, configure **CATALINA_HOME** environment variable in your system using following command.

- **echo "export CATALINA_HOME="/usr/local/tomcat8"" >> ~/.bashrc**
- **source ~/.bashrc**

Now we all set to start the tomcat web server using the scripts provided by the tomcat package.

- **cd /usr/local/tomcat8/bin**

Update Permissions

The tomcat user that we set up needs to have the proper access to the Tomcat installation. We'll set that up now.

- EDIT ---- To run Tomcat from the terminal

- Make sure all .sh scripts are executable (**chmod a+x *.sh**)
- run startup.sh: ./startup.sh or bin/startup.sh (depending on whether you are inside TOMCAT_HOME or inside TOMCAT_HOME/bin)

Go to /tomcat8/bin and run

- **./startup.sh**

Tomcat is not completely set up yet, but you can access the default splash page by going to your domain or IP address followed by: 8080 in a web browser:

Open in web browser:

- **http://server_IP_address:8080**

You will see the default Tomcat splash page, in addition to other information. Now we will go deeper into the installation of Tomcat.

Configure Tomcat Web Management Interface

In order to use the manager webapp that comes with Tomcat, we must add a login to our Tomcat server. We will do this by editing the tomcat-users.xml file:

- **sudo vi /usr/local/tomcat/conf/tomcat-users.xml**

This file is filled with comments which describe how to configure the file. You may want to delete all the comments between the following two lines, or you may leave them if you want to reference the examples:

```
<tomcat-users>
...
</tomcat-users>
```

You will want to add a user who can access the manager-gui and admin-gui (webapps that come with Tomcat). You can do so by defining a user similar to the example below. Be sure to change the username and password to something secure:

```
<role rolename="manage-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-status"/>
<role rolename="manager-jmx"/>
<user username="tomcat" password="tomcat" roles="manage-gui,manager-script,manager-status,manager-jmx"/>
```

Save and quit the tomcat-users.xml file.

By default, newer versions of Tomcat restrict access to the Manager and Host Manager apps to connections coming from the server itself. Since we are installing on a remote machine, you will probably want to remove or alter this restriction. To change the IP address restrictions on these, open the appropriate context.xml files.

For the Manager app, type:

- **sudo vi /usr/local/tomcat8/webapps/manager/META-INF/context.xml**

For the Host Manager app, type:

- **sudo vi /opt/tomcat/webapps/host-manager/META-INF/context.xml**

Inside, comment out the IP address restriction to allow connections from anywhere. Alternatively, if you would like to allow access only to connections coming from your own IP address, you can add your public IP address to the list:

context.xml files for Tomcat webapps

```
<Context antiResourceLocking="false" privileged="true" >
  <!--<Valve className="org.apache.catalina.valves.RemoteAddrValve"
        allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->
</Context>
```

Save and close the files when you are finished.

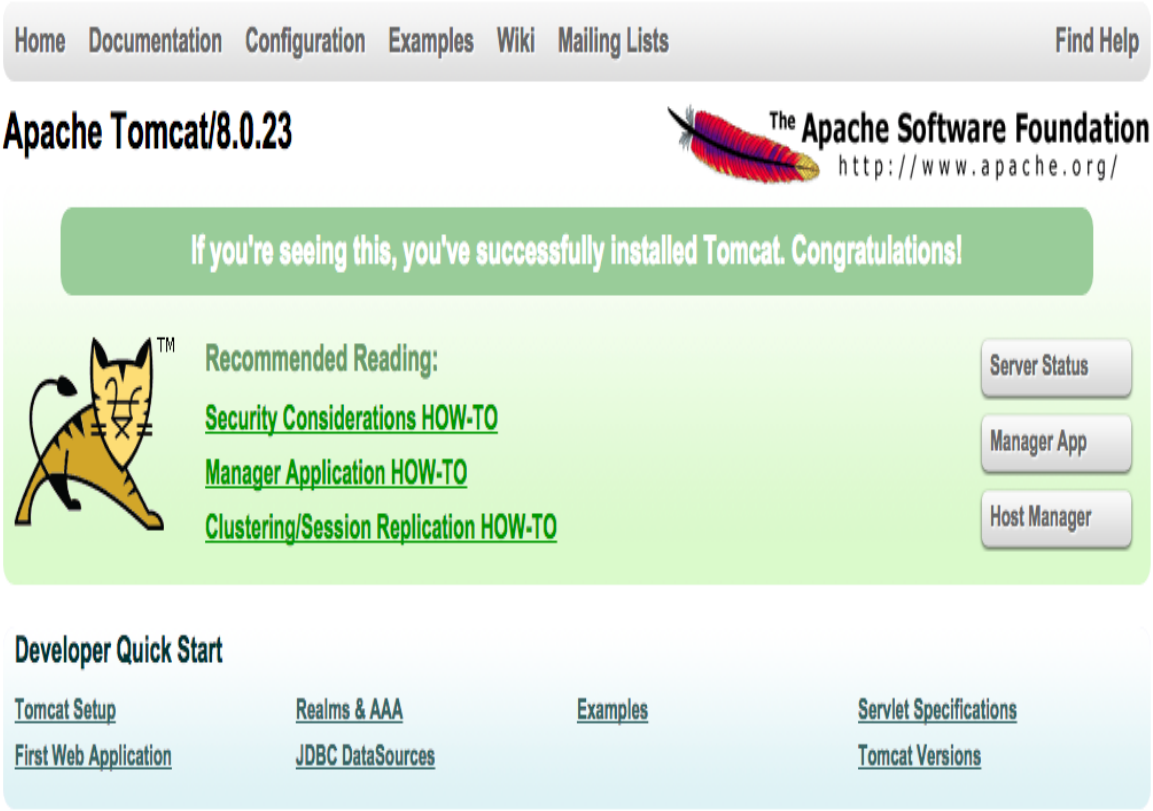
To put our changes into effect, restart the Tomcat service:

- **/usr/local/tomcat/bin/startup.sh**

Access the Web Interface

Now that Tomcat is up and running, let's access the web management interface in a web browser. You can do this by accessing the public IP address of the server, on port 8080:

Open in web browser:
http://server_IP_address:8080
You will see something like the following image:



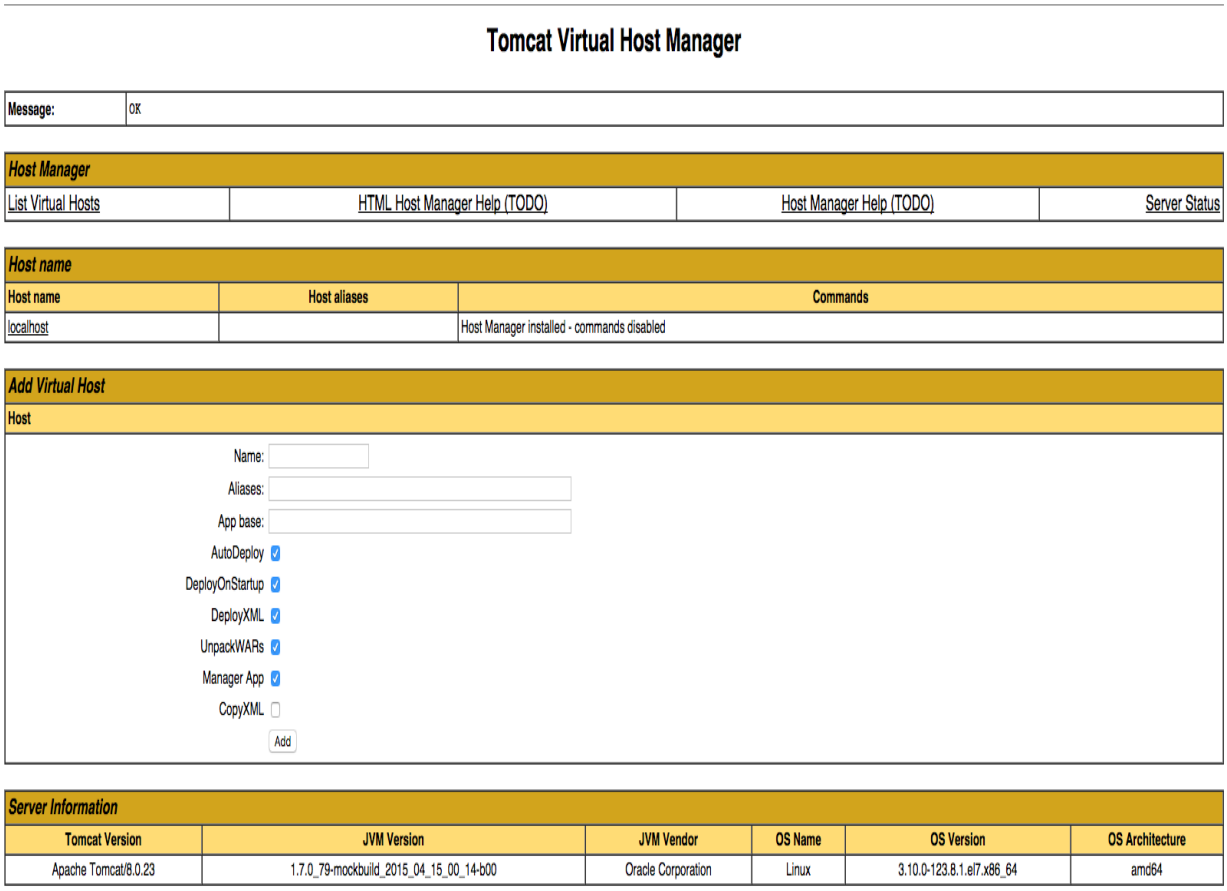
As you can see, there are links to the admin webapps that we configured an admin user for.

Let's take a look at the Manager App, accessible via the link
or **http://server_IP_address:8080/manager/html**:

The Web Application Manager is used to manage your Java applications. You can Start, Stop, Reload, Deploy, and Un-deploy here. You can also run some diagnostics on your apps (i.e. find memory leaks). Lastly, information about your server is available at the very bottom of this page.

Now let's take a look at the Host Manager, accessible via the link
or **http://server_IP_address:8080/host-manager/html/:**

From the Virtual Host Manager page, you can add virtual hosts to serve your applications from.



Tomcat Virtual Host Manager

Message:	OK

Host Manager			
List Virtual Hosts	HTML Host Manager Help (TODO)	Host Manager Help (TODO)	Server Status

Host name		
Host name	Host aliases	Commands
localhost		Host Manager installed - commands disabled

Add Virtual Host

Host

Name:
Aliases:
App base:
AutoDeploy
DeployOnStartup
DeployXML
UnpackWARs
Manager App
CopyXML
Add

Server Information					
Tomcat Version	JVM Version	JVM Vendor	OS Name	OS Version	OS Architecture
Apache Tomcat/8.0.23	1.7.0_79-mockbuild_2015_04_15_00_14-b00	Oracle Corporation	Linux	3.10.0-123.8.1.el7.x86_64	amd64

Jenkins Installation

Step 1 — Installing Jenkins

There are two basic ways to install Jenkins on CentOS: through a repository, or repo, and via the WAR file. Installing from a repo is the preferred method, and it's what we'll outline first.

You'll need Java to run Jenkins (either method), so if your server doesn't yet have Java, install it with:

- **sudo yum -y install java**

In general, if you need a service or tool but you're not sure what package provides it, you can always check by running:

- **yum whatprovides service**

Where service is the name of the service or tool you require.

Installing from the Repo

Now, run the following to download Jenkins from the Red Hat repo:

- **sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo**

The wget tool downloads files into the filename specified after the "O" flag (that's a capital 'O', not a zero).

Then, import the verification key using the package manager RPM:

- **sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key**

Finally, install Jenkins by running:

- **sudo yum install jenkins**

That's it! You should now be able to start Jenkins as a service:

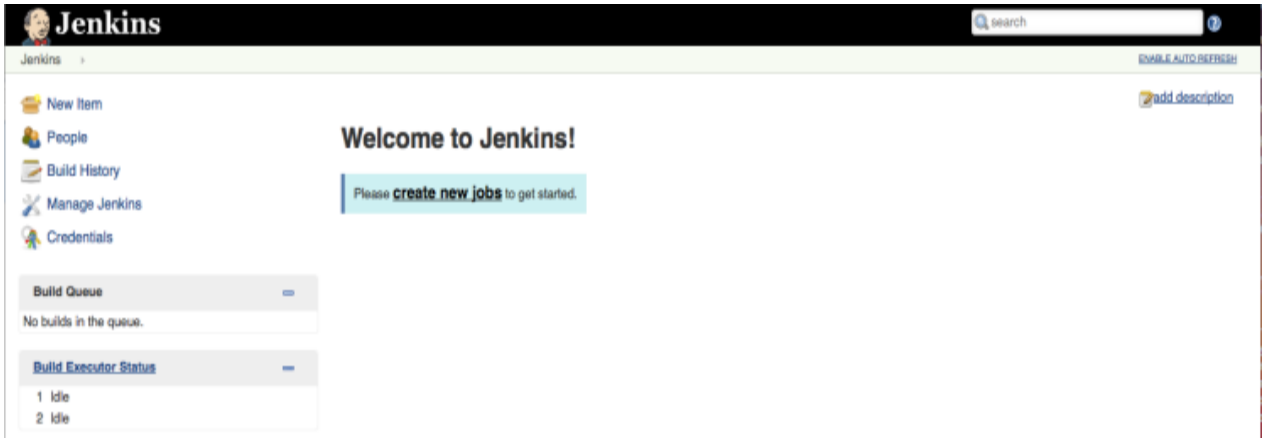
- **sudo systemctl start jenkins.service**

Once the service has started, you can check its status:

- **sudo systemctl status jenkins.service**

This will give you a fairly lengthy readout with a lot of information on how the process started up and what it's doing, but if everything went well, you should see two lines similar to the following:

Loaded: loaded (/etc/systemd/system/jenkins.service; disabled)
Active: active (running) since Tue 2015-12-29 00:00:16 EST; 17s ago
This means that the Jenkins services completed its startup and is running. You can confirm this by visiting the web interface as before, at http://ip-of-your-machine:8080.



Likewise, you can stop the service:

- **sudo systemctl stop jenkins.service**

or restart it:

- **sudo systemctl restart jenkins.service**

More information on managing services with systemctl can be found in the How To Use Systemctl to Manage Systemd Services and Units article.

Installing from the WAR File

If you choose not to install Jenkins via the repo for whatever reason, you can accomplish the same results using the WAR file, though this requires somewhat more work.

Let's first download the Jenkins WAR file to the server and run it without frills to make sure the basics work correctly with no hassle.

The most recent version of Jenkins at any given time is available on the Jenkin's mirror. You can use any tool you like to download this file. The following method employs a command line tool called wget:

- **wget http://mirrors.jenkins-ci.org/war/latest/jenkins.war**

When you're ready, start Jenkins via Java:

- **java -jar jenkins.war**

You should see output in the console indicating that Jenkins has started running:

```
...
INFO: Jenkins is fully up and running
...
You can now access the interface via the browser (http://ip-of-your-machine:8080).
```

Having confirmed that Jenkins runs successfully, end the process so that you can make changes to it to allow it to run as a service, as discussed in the next step. To end a running foreground process, press CTRL-C.

Step 2 — Running Jenkins as a Service

If in the previous section you opted not to install Jenkins via the repo, and instead used the WAR file, you won't yet be able to use Jenkins like a standard service. If you did use the repo, skip this step.

When we configure Jenkins the following way, it will still run through Java, but you'll be able to treat it like a service, starting and stopping it and letting it run in the background with ease. The service will essentially be working as a wrapper.

First, make sure the WAR file you've downloaded is sitting in a location convenient for long-term storage and use:

- **sudo cp jenkins.war /usr/local/bin/jenkins.war**

Then, go to your /etc/systemd/system/ directory, and create a new file called jenkins.service. The following demonstration uses the nano editor, but naturally you can use whatever editing tool you wish.

- **cd /etc/systemd/system/**
- **sudo nano jenkins.service**

Now, add the following lines to the new jenkins.service file. In a moment, we'll go over exactly what these lines accomplish.

```
/usr/local/bin/jenkins.war

[Unit]
Description=Jenkins Service
After=network.target

[Service]
Type=simple
User=root
ExecStart=/usr/bin/java -jar /usr/local/bin/jenkins.war
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

- If you've seen configuration files before (INI files or similar), you'll recognize the structure being used here. The bracketed text denotes a section heading. This means that, for instance, [Service] declares a section called "Service," and all the assignments below it contain relevant information that the system will then know how to find and relate to the section header.
- A configuration file, this one included, is typically a text file – meaning it has no intrinsic meaning to the computer. Rather, the text file will be parsed by some process, and that process will use the headings and other information to find its way around. For this reason, it's technically irrelevant how a given configuration file is laid out – as long as the program which reads it can understand what everything means.
- The first section, Unit, contains only two configuration directives. The first is simply a name. It can be whatever name you'd like, but ideally it should be one that uniquely identifies the new process. The second directive states what service, if any, is necessary for the current service to start.
- In the next section, the Type directive allows you to select what type of startup this service will use. The value simple indicates that the process noted in the later directive ExecStart will be the primary process of the service being created. Really, type is unnecessary, as simple is assumed when type is unspecified, but we are leaving it in for clarity.
- User specifies which user has control over this process, and Restart is used to indicate that, in this case, if the process terminates but the exit code implies error, the service will be restarted. This is useful in maintaining the continuity of the service in case of unexpected crashes.
- As mentioned, ExecStart is the directive where we indicate what process is to become the main action of the service. This directive represents the main wrapper for Jenkins – the service will run the WAR through Java rather than treating it a foreground process.
- Finally, in the Install section, multi-user.target indicates a target, called a runlevel prior to CentOS 7. It provides for the system a sense of what resources to provide this service and what amount of intensity will be required by the user.

Once your file is created and saved, you should be able to start up your new Jenkins service!

When you are ready, run:

- **sudo systemctl daemon-reload**

This applies the changes you have made to this unit (actually, it applies the changes to any and all units that have been altered).

You should now be able to start Jenkins as a service:

- **sudo systemctl start jenkins.service**

Once the service has started, you can check its status:

- **sudo systemctl status jenkins.service**

This will give you a fairly lengthy readout with a lot of information on how the process started up and what it's doing, but if everything went well, you should see two lines similar to the following:

Loaded: loaded (/etc/systemd/system/jenkins.service; disabled)
Active: active (running) since Tue 2015-12-29 00:00:16 EST; 17s ago
This means that the Jenkins services completed its startup and is running. You can confirm this by visiting the web interface as before, at http://ip-of-your-machine:8080.

Likewise, you can stop the service:

- **sudo systemctl stop jenkins.service**

or restart it:

- **sudo systemctl restart jenkins.service**

More information on managing services with systemctl can be found in the How To Use Systemctl to Manage Systemd Services and Units article.

Step 3 — Creating Users

Once Jenkins is running smoothly, establishing good security is the next step. From here on out, your exact actions will largely depend on your purposes for Jenkins. However, the following are general guidelines of how Jenkins can best be set up and used, along with some examples to pave the way.

Jenkins provides settings for security and role management, useful for controlling access and defining user actions. We'll visit that briefly to introduce those concepts. To get to those settings, return to the Jenkins interface via your browser once your service is running (http://ip-of-your-machine:8080). You will see a menu on the left – choose **Manage Jenkins** from within that. This will take you to a page containing a number of options for customization. You may also notice an an alert at the top: **Unsecured Jenkins allows anyone on the network to launch processes on your behalf. Consider at least enabling authentication to discourage misuse.** This is Jenkins' directive to get you to introduce some element of security to your system.

The first step to take here is to go to **Configure Global Security**, near top of the list of links on the manage Jenkins page. Check the option box for **Enable security** to bring up a group of options for this purpose. There is any number of ways to configure security on Jenkins – you can read the in-depth explanation in the Standard Security Setup section of the **Use Jenkins** documentation.

The most straightforward of these options, and the one we will lay out today, has Jenkins use its own database to store user configurations. Under the **Access Control** section that appeared when we flagged the checkbox, select **Jenkins' own user database**. Briefly, the other options are to link Jenkins to existing Unix users and groups, to use an organization-wide login (LDAP option), or to allow a Java servlet to manage access. Other options can be added through plugins (we'll discuss plugins in a bit).

Whether you should allow new users to sign up largely depends on your own needs. In general, however, it pays to restrict access, and allowing users to sign up as they wish can allow a level of openness that can potentially be dangerous. To restrict this, deselect the checkbox marked **Allow users to sign up**. Once this setting has been turned off, only administrators can create new accounts. In a moment, you'll supply administrative privileges for a user you'll create, and we'll go into detail on adding new users, as well.

Under **Authorization**, select the **Matrix-based security** option. This allows some fine-tuning of the controls without resorting to complex setups. You'll see a user named **Anonymous** is already present. An anonymous user is anybody from anywhere, even when they're not logged in, which is why by default the anonymous user has no abilities. Since this is the initial setup of the Jenkins instance, you must give these user full permissions: there are no users other than anonymous right now, and you're not logged in, so turning off anonymous permissions would effectively cut you off from accessing Jenkins at all.

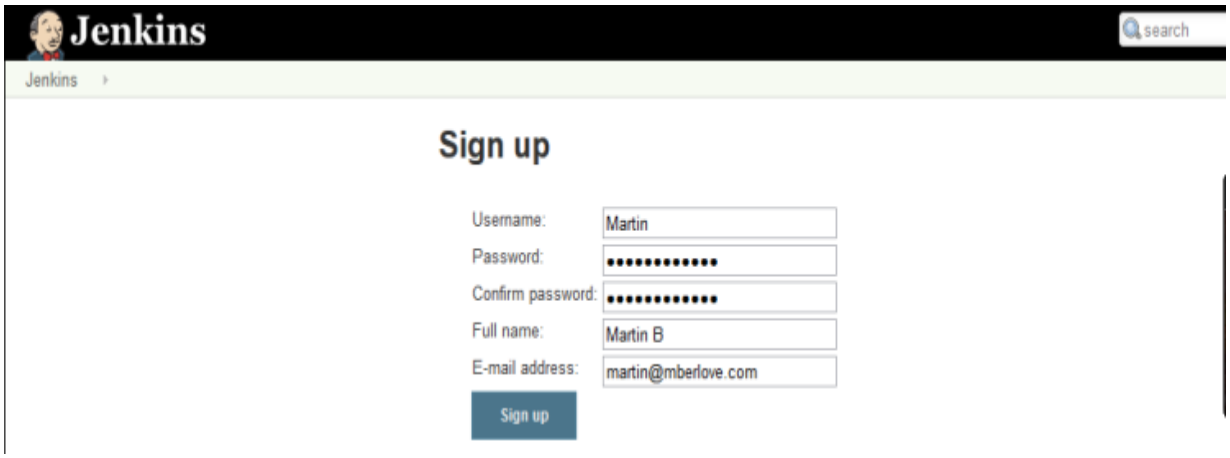
Use the small button to the right of the **Anonymous** row to select all permissions. Next, use the **User/group to add** input field to specify a new user for which to add permissions. Note that this does not actually create a user, but rather specifies permissions for the user you will create shortly.

Normally, you would create a new user first and then specify permissions for them in this part of the form. Since no user exists yet, you'll set up permissions and then create the user.

Enter a username and press **Add**. Due to a known bug, it is recommended that you keep the usernames lowercase. Give the new user all permissions the same way you did for the anonymous user. This essentially sets up a new administrator.

When you're done, press **Apply** and then **Save**.

You will be taken automatically to a signup page, from which you can create a new account. The username of the account you create should correspond to the one for which you specified permissions earlier:



When you finish, you should find yourself automatically logged in.

Return to the security page (**Manage Jenkins** -> **Configure Global Security**) and scroll down to the security matrix. Now that you've created an administrative user, you can restrict the permissions for the anonymous user. Deselect all the permissions in the anonymous row, and then click **Apply** and **Save**. Your new user will now be the only user with access to Jenkins.

If you turned off the automatic sign up earlier, you might need to manually create additional new users. Here's how:

Return to the **Manage Jenkins** page, scroll down to near the bottom and click on **Manage Users**. On the left you'll see a sidebar with links; click on **Create User**. Enter the information for the new user the same way as you created the first user, and click **Sign up**. You'll be redirected to the list of users, which will now include the new user. This user will have no permissions, so you will need to repeat the permissions process, going to **Configure Global Security**, using the **User/group to add** field to add a row to the matrix, specifying permissions, and clicking **Apply** and **Save**. For simplicity's sake, if you have multiple users to create, create them all before moving on to adding permissions.

When creating new users, keep in mind that restrictiveness can be a major security asset. You can learn more about the specific ins and outs of matrix-based security in the Matrix-based Security section of the **Use Jenkins** documentation.

Typically, the next step is to assign roles to your users, controlling their exact abilities. We won’t go into details in this article, but this is a good article on the subject. Be sure to save your changes after you assign roles.

Step 4 — Installing Plugins

Once Jenkins is installed, minimally configured, and reasonably secured, it's time to make it fit your needs. As found when it is first installed, Jenkins has relatively few abilities. In fact, Jenkins typifies a credo of many software developers: do one thing, and do it well. Jenkins "does one thing" by acting as a middleman for your software projects and “does it well” by providing plugins.

Plugins are add-ons that allow Jenkins to interact with a variety of outside software or otherwise extend its innate abilities. As with many areas of the Jenkins setup, the exact plugins you install will be significantly dependent on your projects.

From the main left hand side menu in Jenkins, click **Manage Jenkins** -> **Manage Plugins.** The page you land on shows plugins that are already installed but need updating – you can perform this easily by selecting the plugins you want to update and clicking the button at the bottom.

If you click on **Available** from this page, you will be taken to a colossal list of available plugins. Obviously, you don't want to install all possible plugins, so the next question is how to select those you will need.

As mentioned, your choice in this matter will depend on your needs and goals. Fortunately, the Jenkins wiki provides a nice rundown of plugins by topic.

This list is definitely worth perusing, but no matter your project, there are a few plugins which you almost certainly should include. Here are a few — some generic, some specific:

1. **Source control**
   Git, SVN, and Team Foundation Server are some of the more common source control systems. All three of these have plugins in the Jenkins list, and others exist for less common systems as well. If you don't know what source control is, you should really learn about it and start incorporating it in your projects. Be sure to install the plugin for your source control system, so Jenkins can run builds through it and control tests.
2. **Copy Artifact**
   This plugin allows you to copy components between projects, easing the pain of setting up similar projects if you lack a true dependency manager.
3. **Throttle Concurrent Builds**
   If you have multiple builds running which might introduce a conflict (due to shared resources, etc.), this will easily allow you to alleviate this concern.
4. **Dependency Graph Viewer**
   A nifty plugin providing a graphic representation of your project dependencies.
5. **Jenkins Disk Usage**
   Jenkins may be fairly lightweight, but the same can't always be said for the projects with which it integrates. This plugin lets you identify how much of your computing resources any of your jobs are consuming.
6. **Build tools**
   If your project is large, you probably use a build manager, such as Maven or Ant. Jenkins provides plugins for many of these, both to link in their basic functionality and to add control for individual build steps, projection configuration, and many other aspects of your builds.
7. **Reporting**
   While Jenkins provides its own reports, you can extend this functionality to many reporting tools.
8. **Additional Authentication**
   If the default Jenkins abilities for security don't suite you, there are plenty of plugins to extend this – from Google logins, to active directory, to simple modifications of the existing security.

In general, if your project requires a certain tool, search the plugin list page on the wiki for the name of it or for a keyword regarding its function – chances are such a plugin exists, and this is an efficient way to find it.

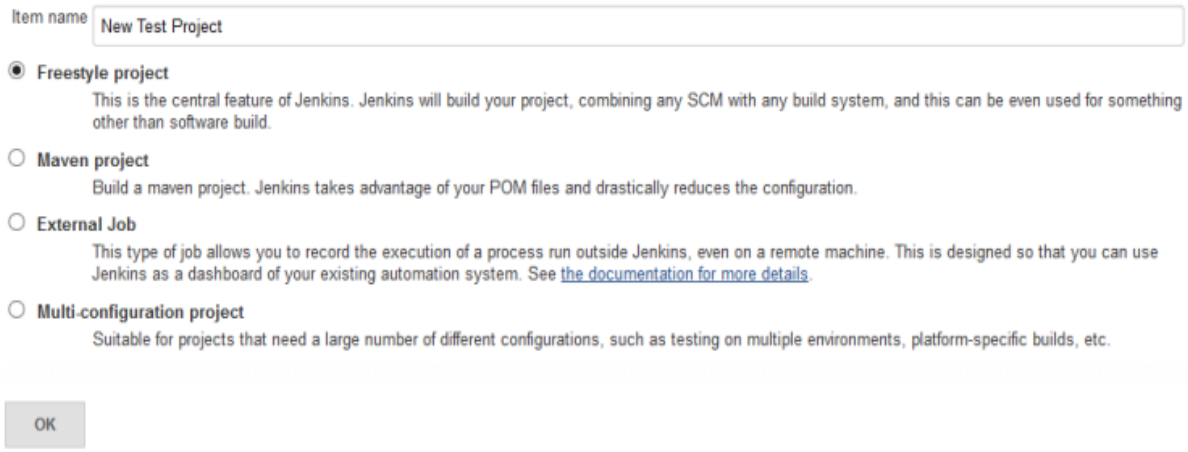
Once you have selected those plugins you want to install on the **Available** tab, click the button marked **Download now and install after restart**.

Now that Jenkins is up and running the way you want it, you can start using it to power your project integration. Jenkins' capabilities are nearly endless within its domain, but the following example should serve to demonstrate both the extent of what Jenkins can do and the beginnings of how to get a Jenkins job started.

Step 5 — Creating a Simple Project

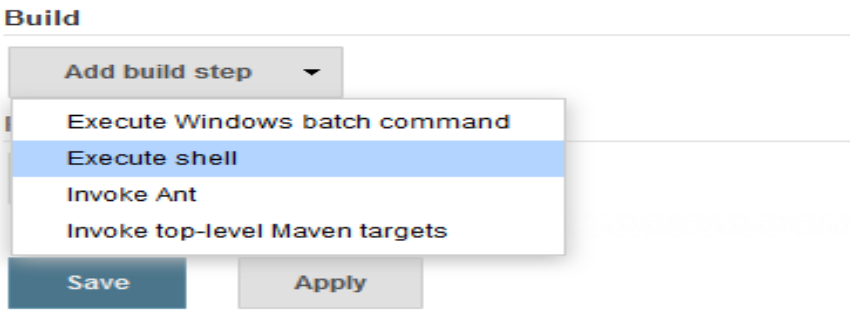
There are a lot of interesting uses you can get out of Jenkins, and even playing around with the settings can be informative. To get started, though, it helps to understand how to set up a basic task. Follow the example in this section to learn how to establish and run a straightforward job.

From the Jenkins interface home, select **New Item**. Enter a name and select **Freestyle project**.

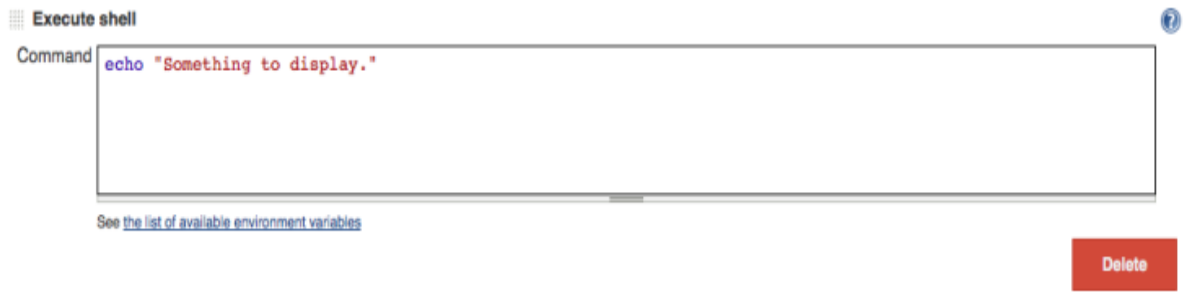


This next page is where you specify the job configuration. As you'll quickly observe, there are a number of settings available when you create a new project. Generally, one of the more important controls is to connect to a source repo. For purposes of this introductory example, we'll skip that step.

On this configuration page you also have the option to add build steps to perform extra actions like running scripts.



This will provide you with a text box in which you can add whatever commands you need. Use this to run various tasks like server maintenance, version control, reading system settings, etc.



We'll use this section to run a script. Again, for demonstration purposes, we'll keep it extremely simple.

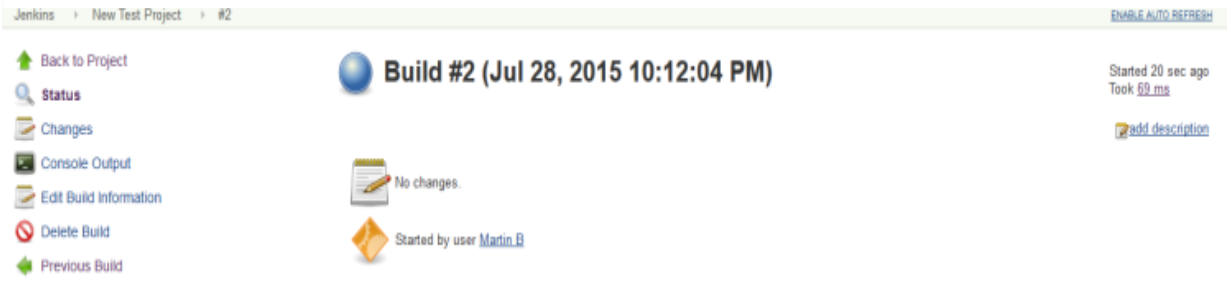
If you want, you can add subsequent build steps as well. Keep in mind that if any segment or individual script fails, the entire build will fail.

You can also select post-build actions to run, such as emailing the results to yourself.

Save the project, and you'll be taken to its project overview page. Here you can see information about the project, including its built history, though there won't be any of that at the moment since this is a brand-new project.

Click **Build Now** on the left-hand side to start the build. You will momentarily see the build history change to indicate it is working. When done, the status icon will change again to show you the results in a concise form.

To see more information, click on that build in the build history area, whereupon you'll be taken to a page with an overview of the build information:



The **Console Output** link on this page is especially useful for examining the results of the job in detail — it provides information about the actions taken during the build and displays all the console output. Especially after a failed build, this can be a useful place to look.

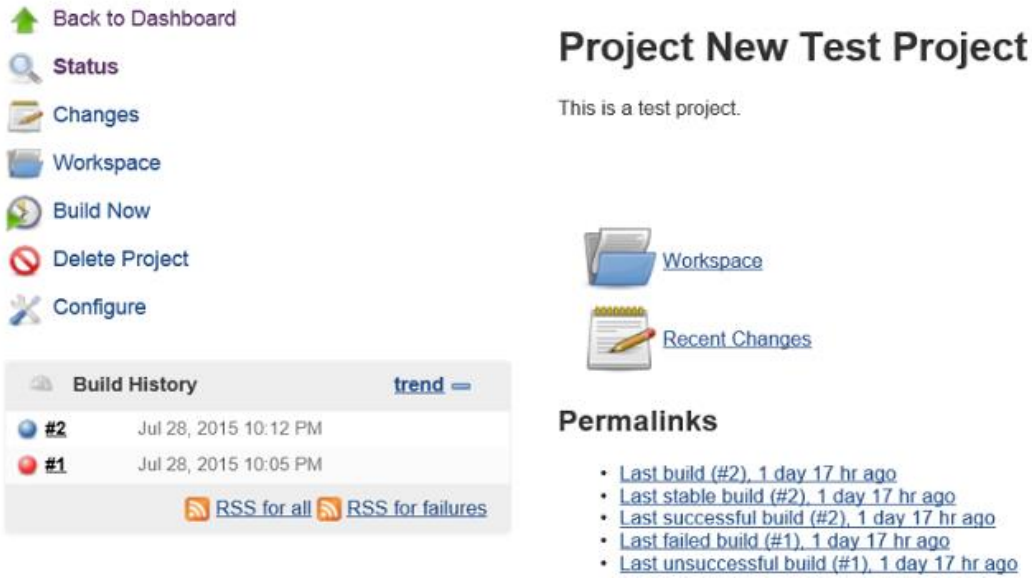
If you go back to Jenkins home, you'll see an overview of all projects and their information, including status (in this case there's only the one):

Status is indicated two ways, by a weather icon (on the home page dashboard, seen above) and by a colored ball (on the individual project page, seen below). The weather icon is particularly helpful as it shows you a record of multiple builds in one image.

In the image above, you see clouds, indicating that some recent builds succeeded and some failed. If all of them had succeeded, you'd see an image of a sun. If all builds had recently failed, there would be a poor weather icon.

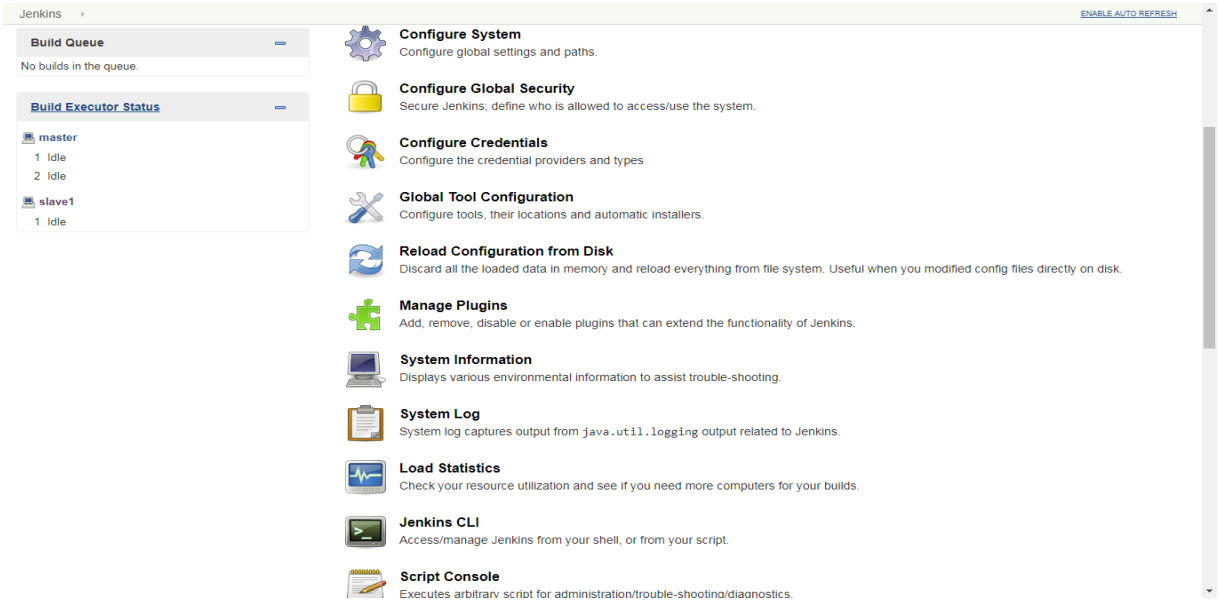
These statuses have corresponding tooltips with explanations on hover and, coupled with the other information in the chart, cover most of what you need in an overview.

You can also rebuild the project from here by clicking (**Build Now**).



Of course, implementing a full-scale project setup will involve a few more steps and some fine-tuning, but it’s clear that without much effort, you can set up some very useful, very pragmatic monitors and controls for your projects. Explore Jenkins, and you’ll quickly find it to be an invaluable tool.

The Configuration Dashboard—The Manage Jenkins Screen



This screen lets you configure different aspects of your Jenkins server. Each link on this page takes you to a dedicated configuration screen, where you can manage different parts of the Jenkins server. Some of the more interesting options are discussed here:

Configure System

This is where you manage paths to the various tools you use in your builds, such as JDKs, and versions of Ant and Maven, as well as security options, email servers, and other system-wide configuration details. Many of the plugins that you install will also need to be configured here—Jenkins will add the fields dynamically when you install the plugins.

Reload Configuration from Disk

Jenkins stores all its system and build job configuration details as XML files stored in the Jenkins home directory. It also stores all of the build history in the same directory. If you are migrating build jobs from one Jenkins instance to another, or archiving old build jobs, you will need to add or remove the corresponding build job directories to Jenkins's builds directory. You don't need to take Jenkins offline to do this—you can simply use the "Reload Configuration from Disk" option to reload the Jenkins system and build job configurations directly. This process can be a little slow if there is a lot of build history, as Jenkins loads not only the build configurations but also all of the historical data as well.

Manage Plugins

One of the best features of Jenkins is its extensible architecture. There is a large ecosystem of third-party open source plugins available, enabling you to add extra features to your build server, from support for different SCM tools such as Git, Mercurial or ClearCase, to code quality and code coverage metrics reporting. Plugins can be installed, updated and removed through the Manage Plugins screen. Note that removing plugins needs to be done with some care, as it can sometimes affect the stability of your Jenkins instance.

System Information

This screen displays a list of all the current Java system properties and system environment variables. Here, you can check exactly what version of Java Jenkins is running in, what user it is running under, and so forth. You can also check that Jenkins is using the correct environment variable settings. Its main use is for troubleshooting, so that you can make sure that your server is running with the system properties and variables you think it is.

System Log

The System Log screen is a convenient way to view the Jenkins log files in real time. Again, the main use of this screen is for troubleshooting. You can also subscribe to RSS feeds for various levels of log messages. For example, as a Jenkins administrator, you might want to subscribe to all the ERROR and WARNING log messages.

Load Statistics

Jenkins keeps track of how busy your server is in terms of the number of concurrent builds and the length of the build queue (which gives an idea of how long your builds need to wait before being executed). These statistics can give you an idea of whether you need to add extra capacity or extra build nodes to your infrastructure.

Script Console

This screen lets you run Groovy scripts on the server. It is useful for advanced troubleshooting: since it requires a strong knowledge of the internal Jenkins architecture, it is mainly useful for plugin developers and the like.

Manage Nodes

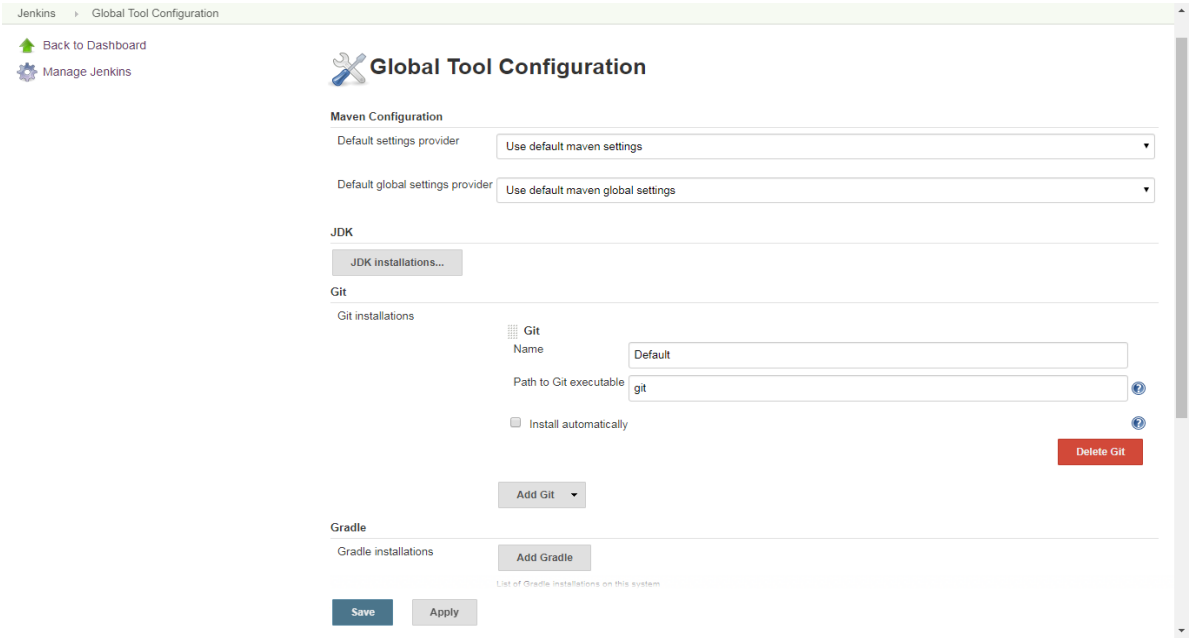
Jenkins handles parallel and distributed builds well. In this screen, you can configure how many builds you want. Jenkins runs simultaneously, and, if you are using distributed builds, set up build nodes. A build node is another machine that Jenkins can use to execute its builds.

Prepare for Shutdown

If you need to shut down Jenkins, or the server Jenkins is running on, it is best not to do so when a build is being executed. To shut down Jenkins cleanly, you can use the Prepare for Shutdown link, which prevents any new builds from being started. Eventually, when all of the current builds have finished, you will be able to shut down Jenkins cleanly.

Configuring the System Environment

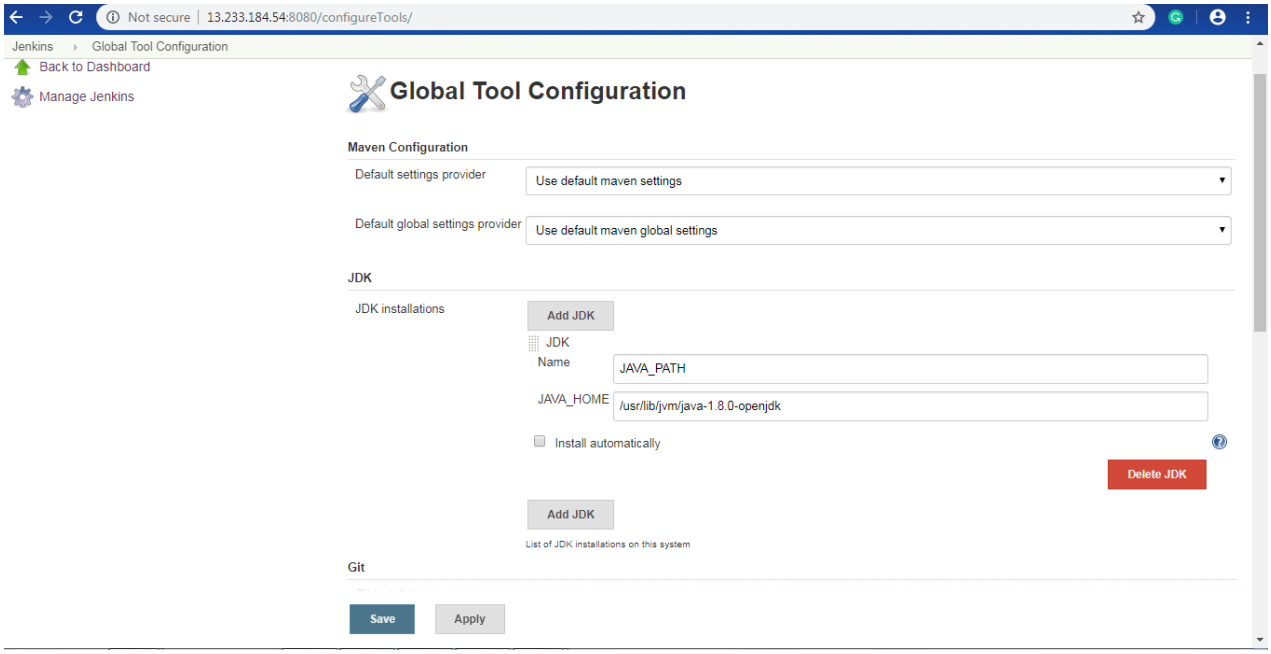
The most important Jenkins administration page is the Configure System screen. Here, you set up most of the fundamental tools that Jenkins needs to do its daily work. The default screen contains a number of sections, each relating to a different configuration area or external tool. In addition, when you install plugins, their system wide configuration is also often done in this screen.



Configuring Your JDKs

One of the most common uses of Jenkins has been to build Java applications. So Jenkins naturally provides excellent built-in support for Java. By default, Jenkins will build Java applications using whatever version of Java it finds on the system path, which is usually the version that Jenkins itself is running under. However, for a production build server, you will probably want more control than this. For example, you may be running your Jenkins server under Java 6, for performance reasons. However, your production server might be running under Java 5 or even Java 1.4. Large organizations are often cautious when it comes to upgrading Java versions in their production environments, and some of the more heavyweight application servers on the market are notoriously slow to be certified with the latest JDKs.

Jenkins provides good support for working with multiple JVMs. Indeed, Jenkins makes it very easy to configure and use as many versions of Java as you want. Like most system level configuration, we do this in the Configure System screen. Here, you will find a section called JDK which allows you to manage the JDK installations you need Jenkins to work with.



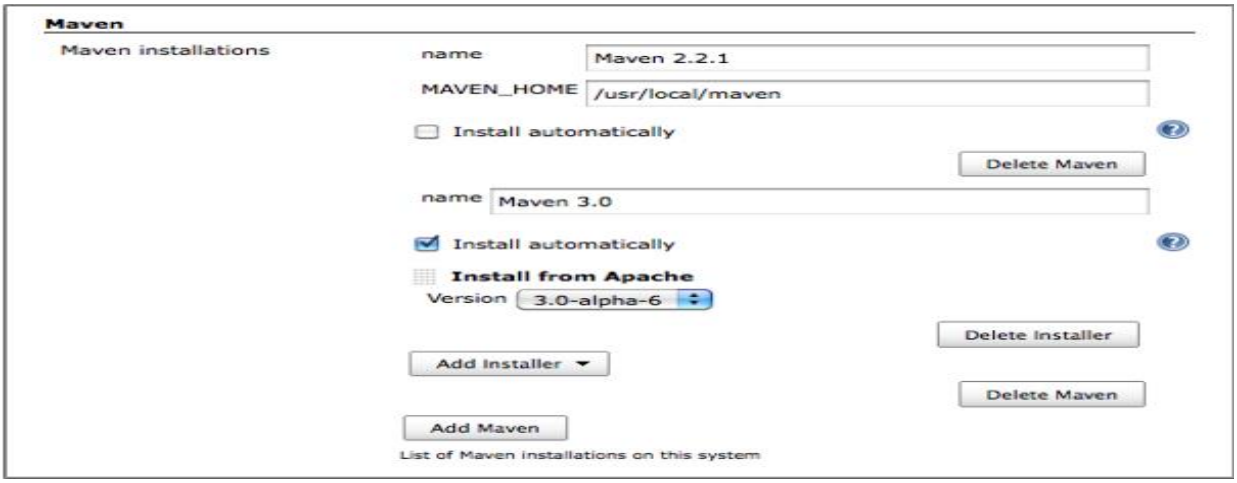
Configuring Your Build Tools

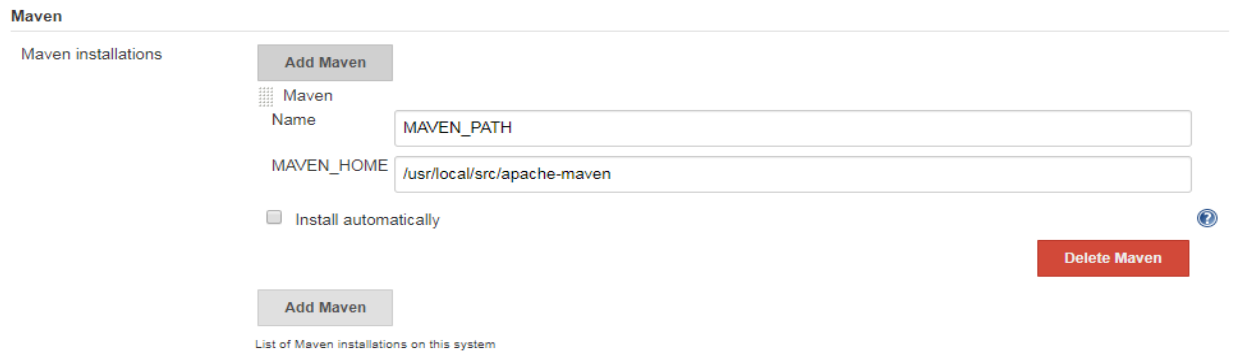
Jenkins supports three principal build tools: Ant, Maven, and the basic shell-script (or Batch script in Windows). Using Jenkins plugins, you can also add support for other build tools and other languages, such as Gant, Grails, MSBuild, and many more.

Maven

Maven is a high-level build scripting framework for Java that uses notions such as a standard directory structure and standard life cycles, Convention over Configuration, and Declarative Dependency Management to simplify a lot of the low-level scripting that you find in a typical Ant build script. In Maven, your project uses a standard, well-defined build life cycle—compile, test, package, deploy, and so forth. Each life cycle phase is associated with a Maven plugin. The various Maven plugins use the standard directory structure to carry out these tasks with a minimum of intervention on your part. You can also extend Maven by overriding the default plugin configurations or by invoking additional plugins.

Jenkins provides excellent support for Maven, and has a good understanding of Maven project structures and dependencies. You can either get Jenkins to install a specific version of Maven automatically.





Ant

Ant is a widely-used and very well-known build scripting language for Java. It is a flexible, extensible, relatively low-level scripting language, used in a large number of open source projects. An Ant build script (typically called build.xml) is made up of a number of targets. Each target performs a particular job in the build process, such as compiling your code or running your unit tests. It does so by executing tasks, which carry out a specific part of the build job, such as invoking javac to compile your code, or creating a new directory. Targets also have dependencies, indicating the order in which your build tasks need to be executed. For example, you need to compile your code before you can run your unit tests. Jenkins provides excellent build-in support for Ant—you can invoke Ant targets from your build job, providing properties to customize the process as required.

Ant is available on the system path, Jenkins will find it. However, if you want to know precisely what version of Ant you are using, or if you need to be able to use several different versions of Ant on different build jobs, you can configure as many installations of Ant as required. Just provide a name and installation directory for each version of Ant in the Ant section of the Configure System screen. You will then be able to choose what version of Ant you want to use for each project.

Shell-Scripting Language

If you are running your build server on Unix or Linux, Jenkins lets you insert shell scripts into your build jobs. This is handy for performing low-level, OS-related tasks that you don't want to do in Ant or Maven. In the Shell section, you define the default shell that will be used when executing these shell scripts. By default, this is /bin/sh, but there are times you may want to modify this to another command interpreter such as bash or Perl. In Windows, the Shell section does not apply—you use Windows batch scripting instead. So, on a Windows build server, you should leave this field blank.

Working with GitHub

Contributed by Matthew McCullough Git is a popular distributed version control system that is a logical successor to Subversion and a mind-share competitor to Mercurial. Git support in Jenkins is both mature and full-featured. There are a number of plugins that can contribute to the overall story of Git in Jenkins.

Installing the plugin

The Git plugin is available in the Jenkins Plugin Manager and is documented on its own wiki page. The plugin assumes that Git (version 1.3.3 or later) has already been installed on your build server, so you will need to make sure that this is the case. You can do this by running the following command on your build server:

- **git –version**

```
git version 1.7.1
```

Next, go back to Jenkins, check the corresponding check box in the Jenkins Plugin Manager page and click the Install button.

System-wide configuration of the plugin

After installing the Git plugin, a small new set of configuration options will be available on the Manage **Jenkins→Configure** System page. In particular, you need to provide the path to your Git executable. If Git is already installed on the system path, just put "**git**" here.

SSH key setup

If the Git repository you are accessing uses SSH passphrase-less authentication—for example, if the access address is similar to **git@github.com:mat thewmccullough/some-repo.git**—you'll need to provide the private half of the key as file ~/.ssh/id_rsa where ~ is the home directory of the user account under which Jenkins is running.

Task:- Build Java-maven project and deploy on Tomcat server automatically

Repository: - https://github.com/topgun93/spring-project.git

Install Plugin: - Maven Integration

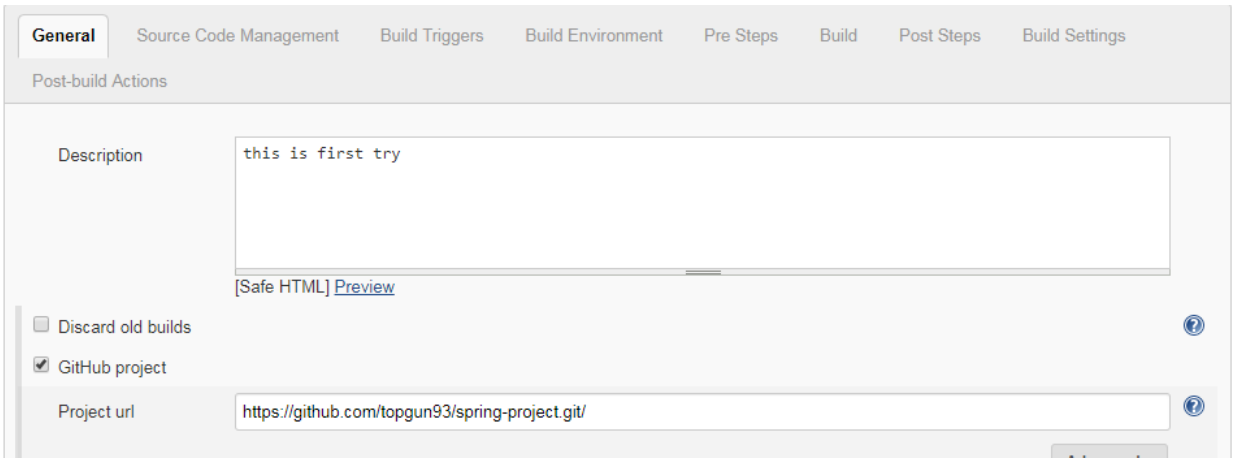
Step 1:-

Go to Jenkins Dashboard →Manage Jenkins →Manage plugins →Available →Maven Integration →Install

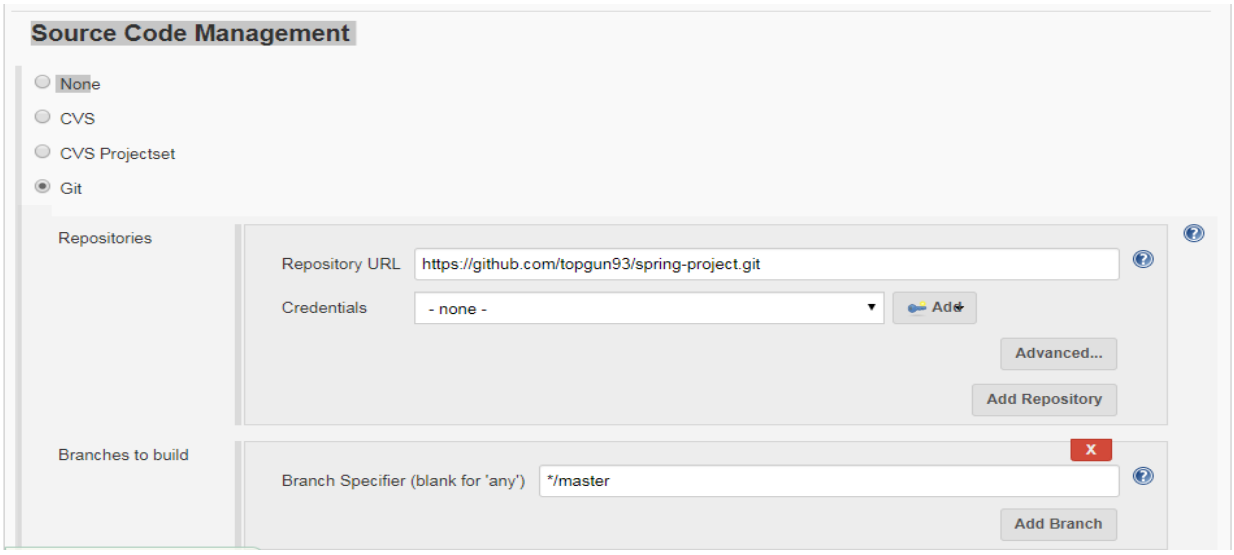
Step 2:-

Go to Jenkins Dashboard -> New Item -> Maven Project option will be available

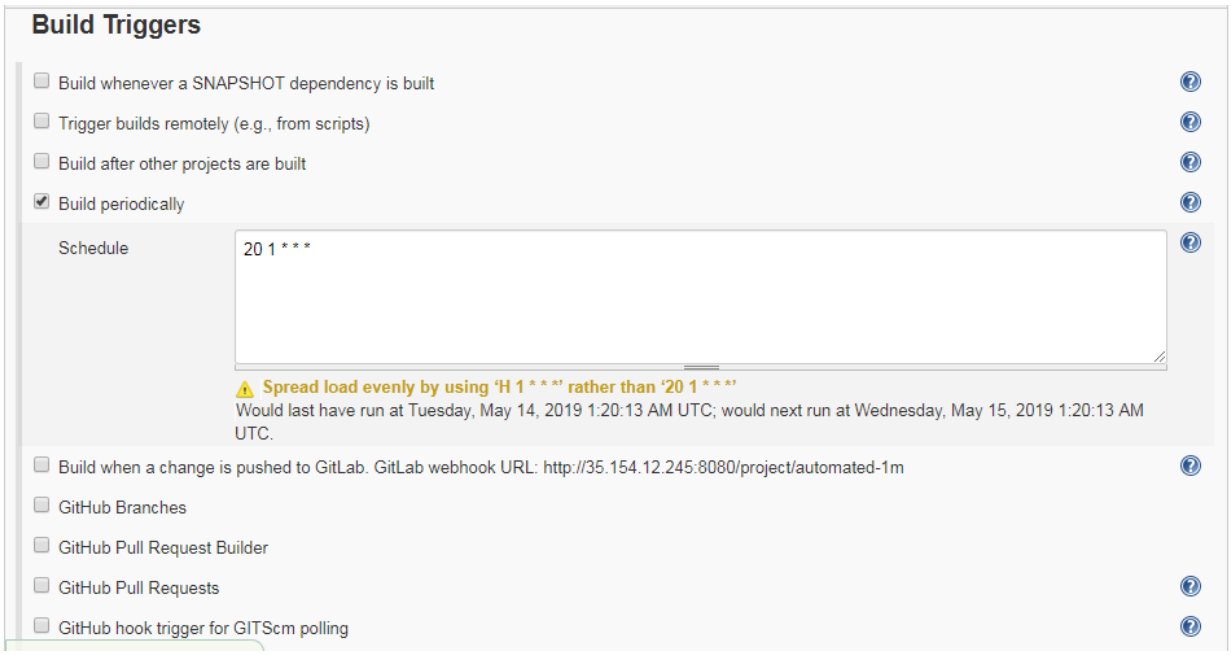
Step 3:- In general section put Url of Maven project which is in GitHub.



Step 4:- Go to source code management and add repo Url and branch from which you want to take project



Step6:- Set trigger according to your requirement

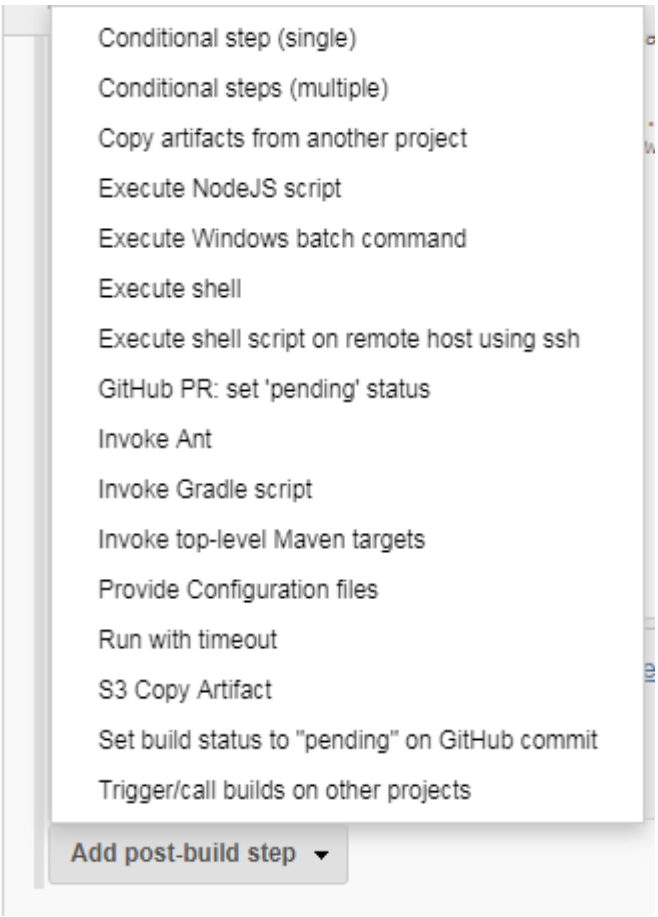


Step 7:- If you want to run any command so you can choose Pre steps build

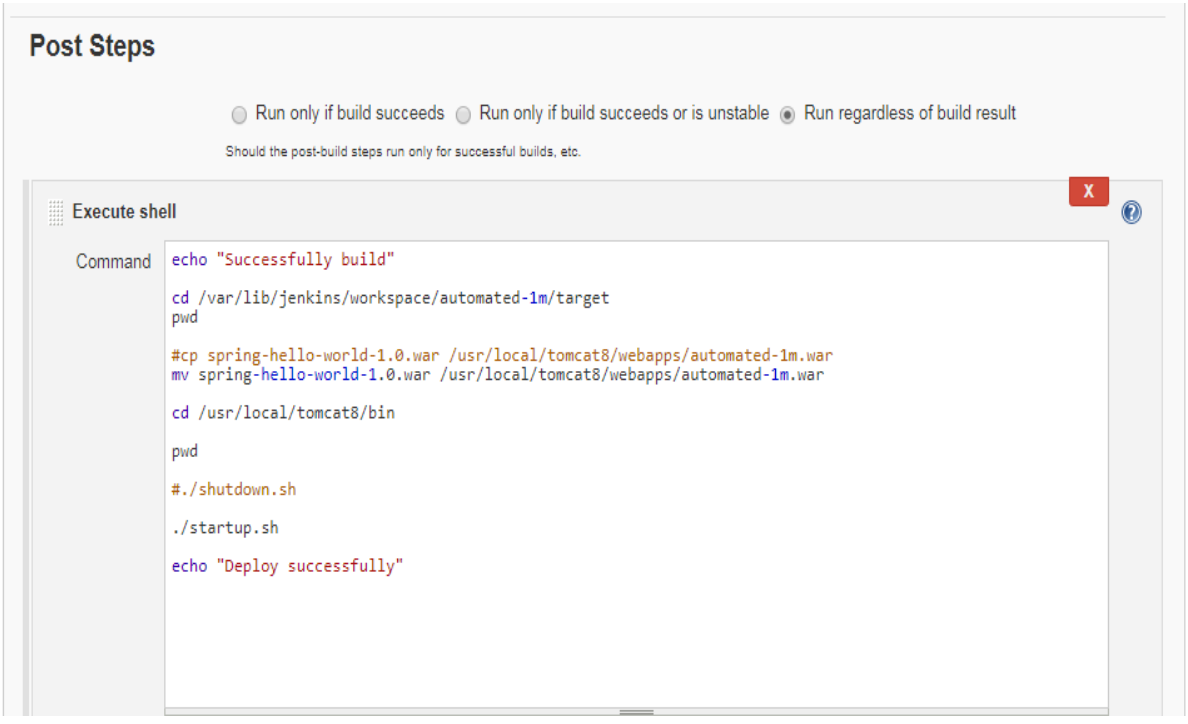
In build section set goals and **options clean package**

Step 8:- We use post step section for deployment on tomcat.

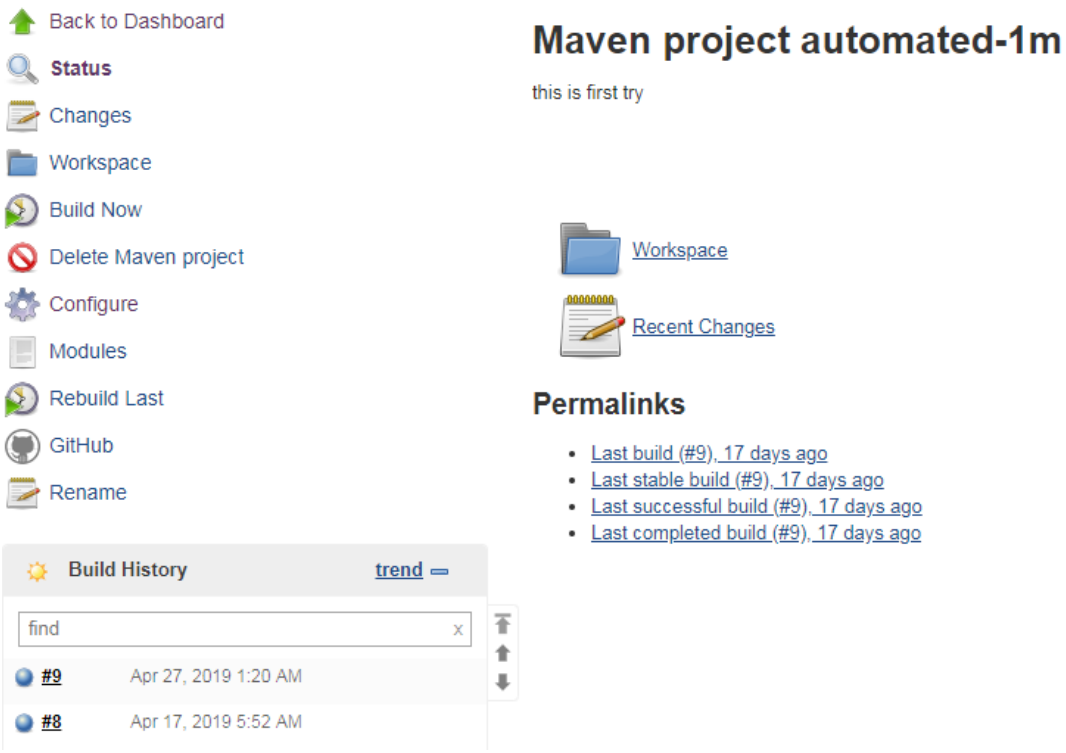
Click on add post-build step→ click execute shell → write the script to deploy project
echo "Successfully build" cd /var/lib/jenkins/workspace/automated-1m/target pwd #cp spring-hello-world-1.0.war /usr/local/tomcat8/webapps/automated-1m.war mv spring-hello-world-1.0.war /usr/local/tomcat8/webapps/automated-1m.war cd /usr/local/tomcat8/bin #./shutdown.sh ./startup.sh echo "Deploy successfully"



This script is copying the build artifact from Jenkins and put in tomcat server



Step 9:- click on save and build the project.



Troubleshooting:-

We have to install tomcat in Jenkins and set the permission for Jenkins otherwise Jenkins user is not able to make any changes in tomcat and we haven't deployed artifact.

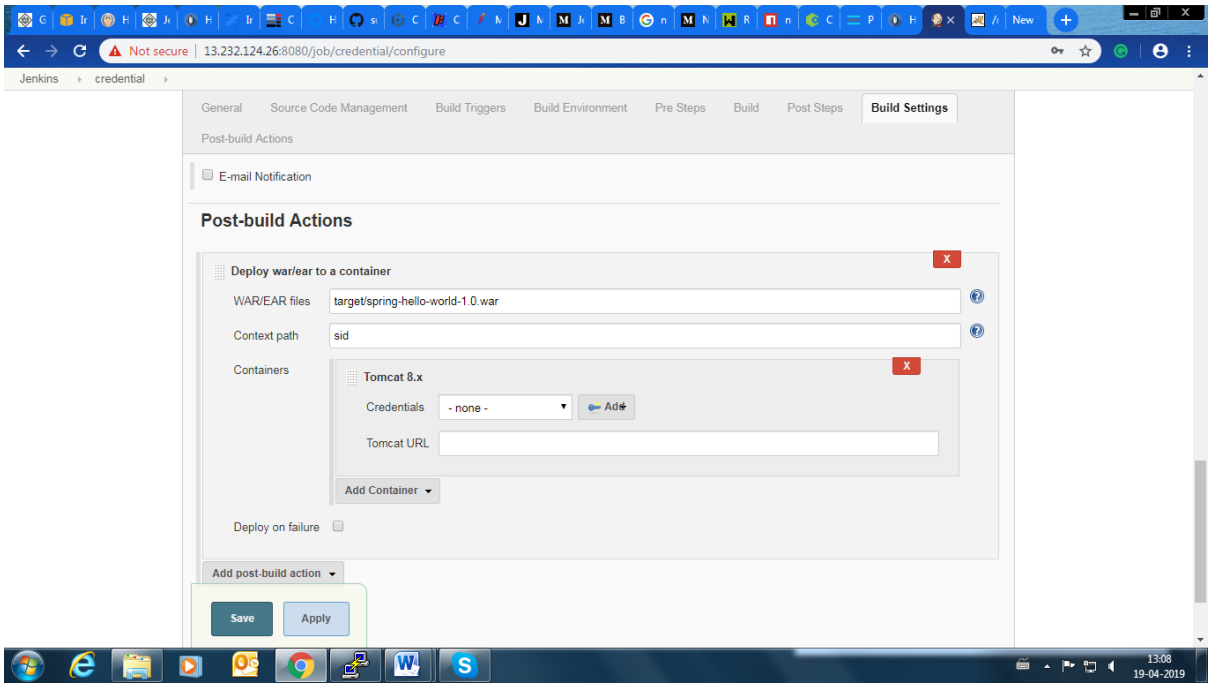
Set all the permission of tomcat in Jenkins and 755

```
echo "Successfully build"

cd /var/lib/jenkins/workspace/manual_automate/target
pwd

#cp spring-hello-world-1.0.war /usr/local/tomcat8/webapps/automated-1m.war
mv spring-hello-world-1.0.war /var/lib/jenkins/tomcat8/webapps/manual_automate.war

cd /var/lib/jenkins/tomcat8/bin

pwd

#./shutdown.sh

./startup.sh
echo "Deploy successfully"
```

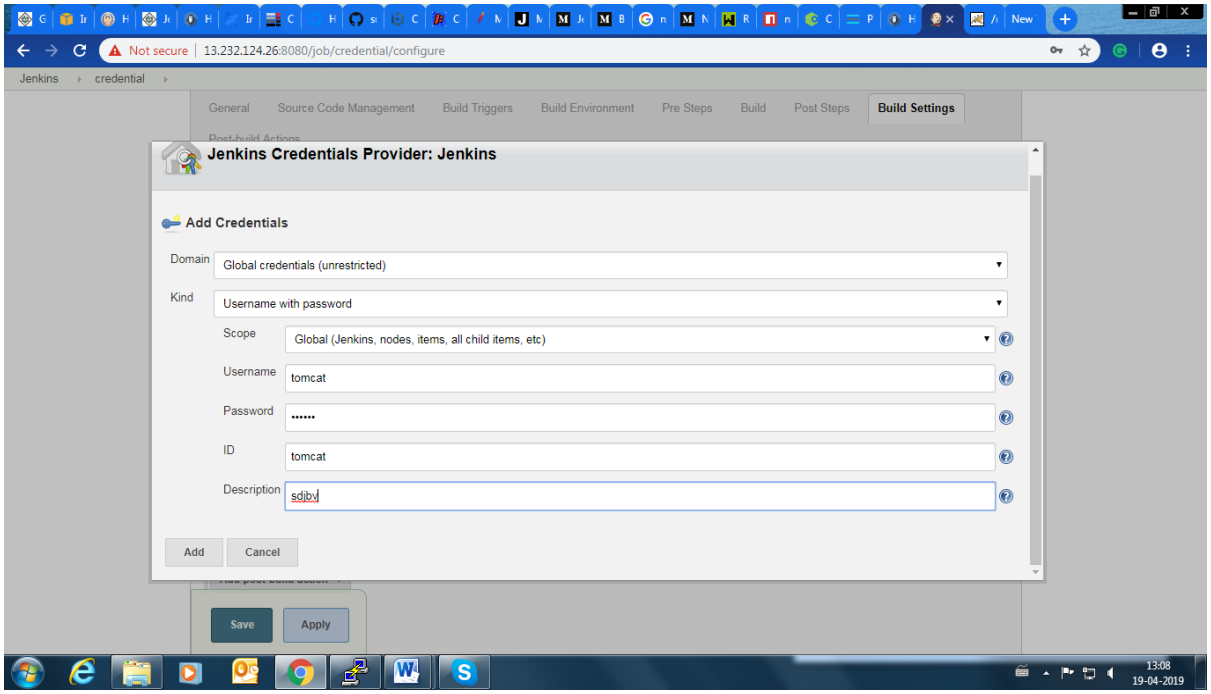
We can deploy this project in another way

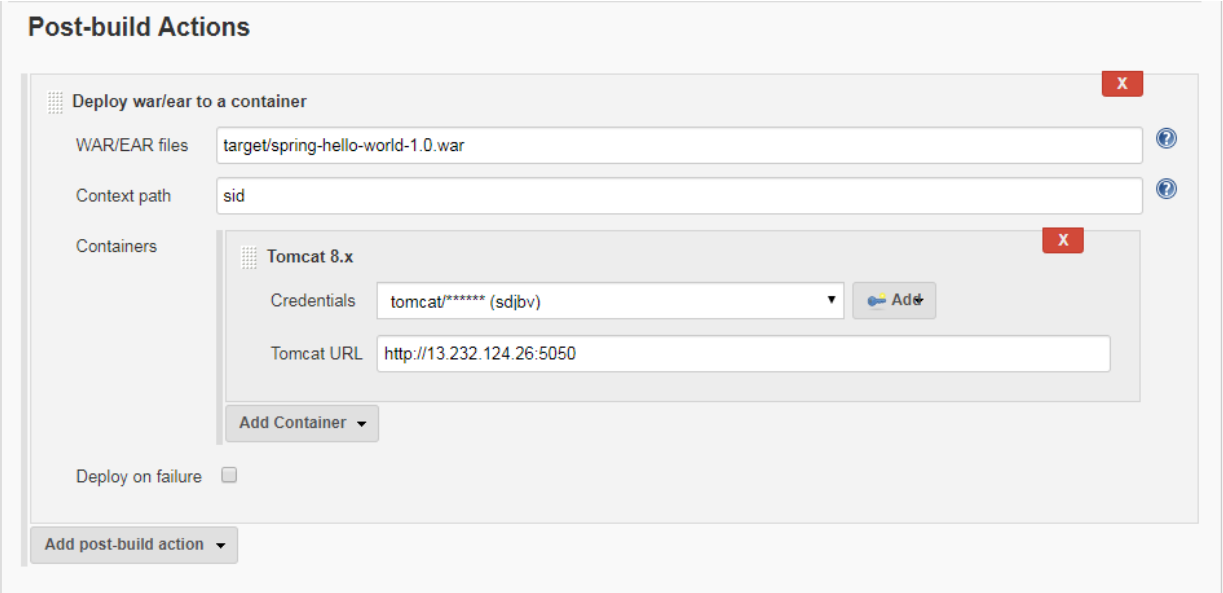
Install Plugin: - Deploy to container plugin

Step 1:- In Post-build actions we add Deploy war/ear to a container option



Step 2:- Add credentials of tomcat





No other changes is required

Step 3:- click on save and build the project.

Task: - Installation of Node Package Manager and build NodeJS project and deploy on Tomcat

Install Latest Nodejs on CentOS/RHEL 7/6

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Latest version node.js yum repository is maintaining by its official website. Use this tutorial to add yum repository and install Latest Nodejs to CentOS/RHEL 7/6 systems with the simple commands.
To install specific nodejs version, Visit our tutorial **Install Specific Nodejs Version with NVM**.

Step 1: - Add Node.js Yum Repository

First of all, you need to enable node.js yum repository in your system provided by the Node.js official website. You also need development tools to build native add-ons to be installed on your system.

For Latest Release:-

- **yum install -y gcc-c++ make**
- **curl -sL https://rpm.nodesource.com/setup_11.x | sudo -E bash –**

For Stable Release:-

- **yum install -y gcc-c++ make**
- **curl -sL https://rpm.nodesource.com/setup_10.x | sudo -E bash**

Step 2: - Install Node.js on CentOS

After adding yum repository in your system lets install Node.js package. NPM will also be installed with node.js. This command will also install many other dependent packages on your system.

- **sudo yum install nodejs**

Don't Miss => Yarn Installation (A Node Modules Manager)

Step 3 : - Check Node.js and NPM Version

After installing node.js verify and check the installed version. You can find more details about current version on node.js official website.

- **node -v**

  v11.12.0

Also, check the version of npm.

- **npm -v**

  6.7.0

Now configure Jenkins for NPM build

Repository: - https://github.com/contentful/the-example-app.nodejs.git

Go to the clone location
- **cd the-example-app.nodejs/**

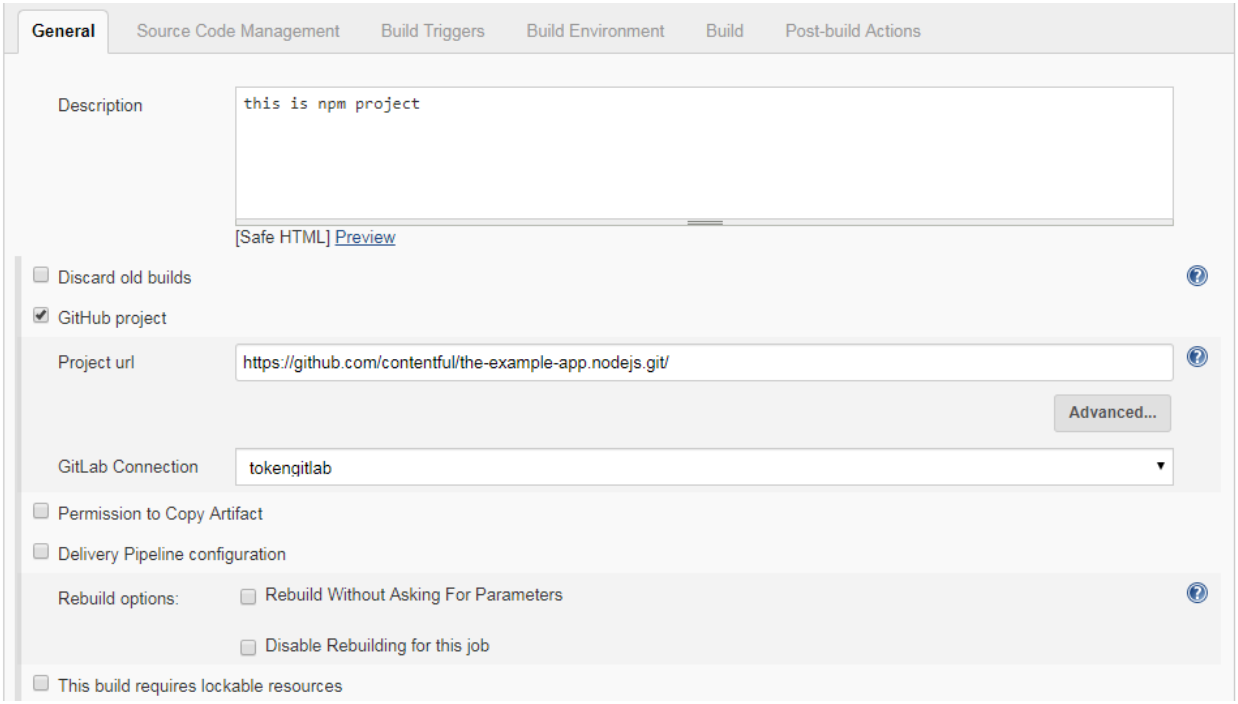
Fetch all the dependencies
- **npm install**

In node_module you can get all the module and important configuration
- **cd node_modules**

File which contains all the package configuration
- **cat package-json/package.json**

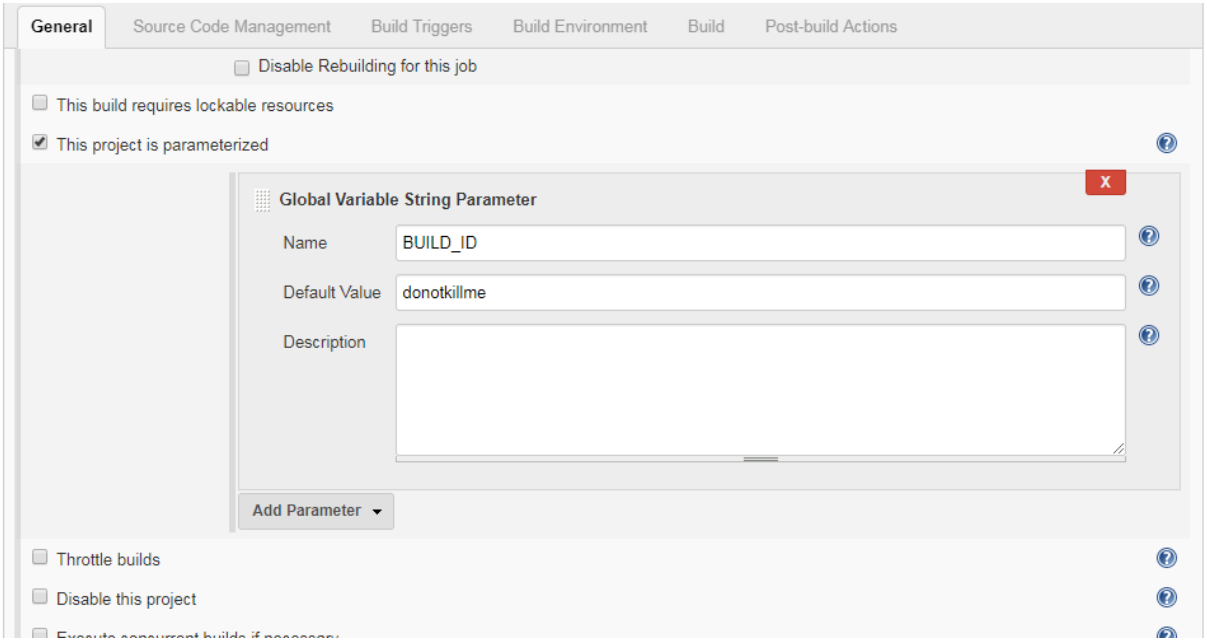
To start npm
- **npm run start:dev (it will start on http://ip:3000)**

Step 1: - Enter url of Git repository in General section

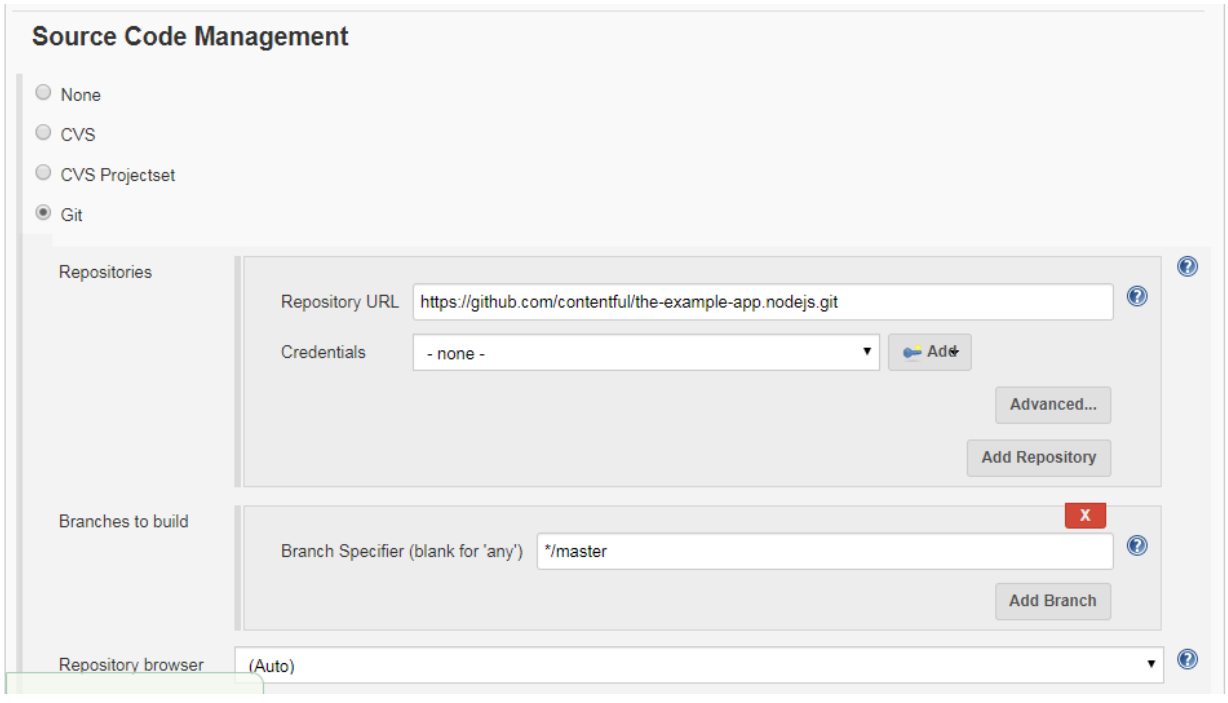


Step 2: - In general section, we have to give variable string parameter for run in daemon.

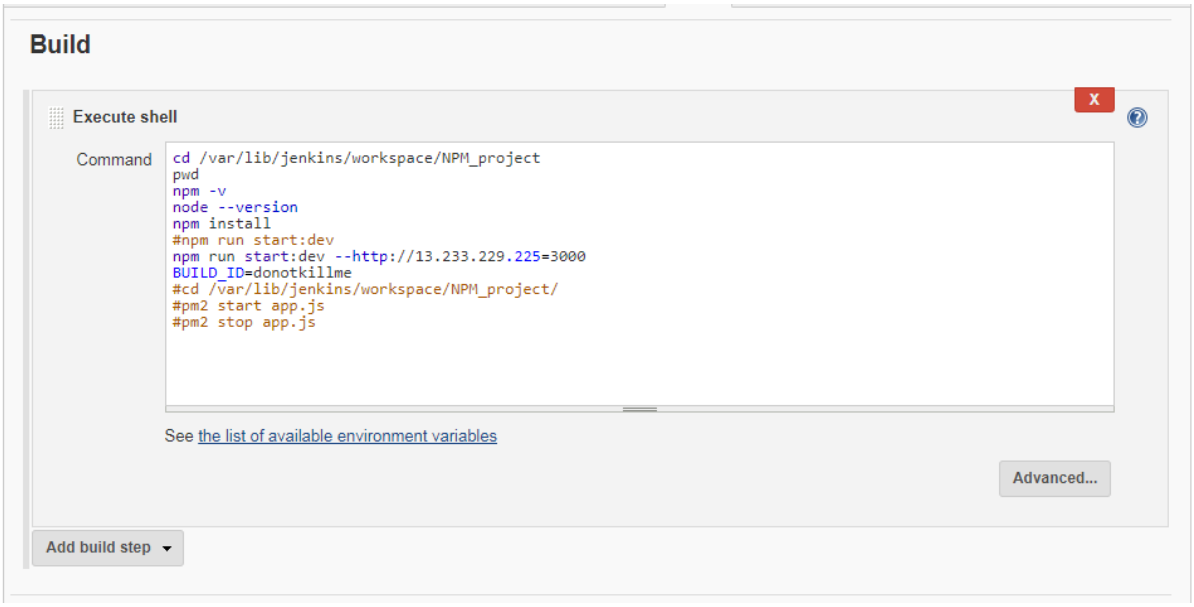
Install Plugin: - Global variable string parameter



Step 3: - Give Git repository in source code management and branch to build

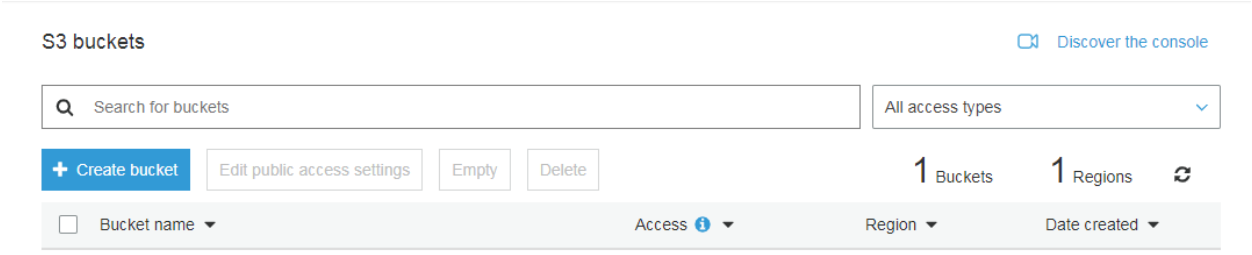


Step 4: - In build section, click on add build step and select **Execute shell**.

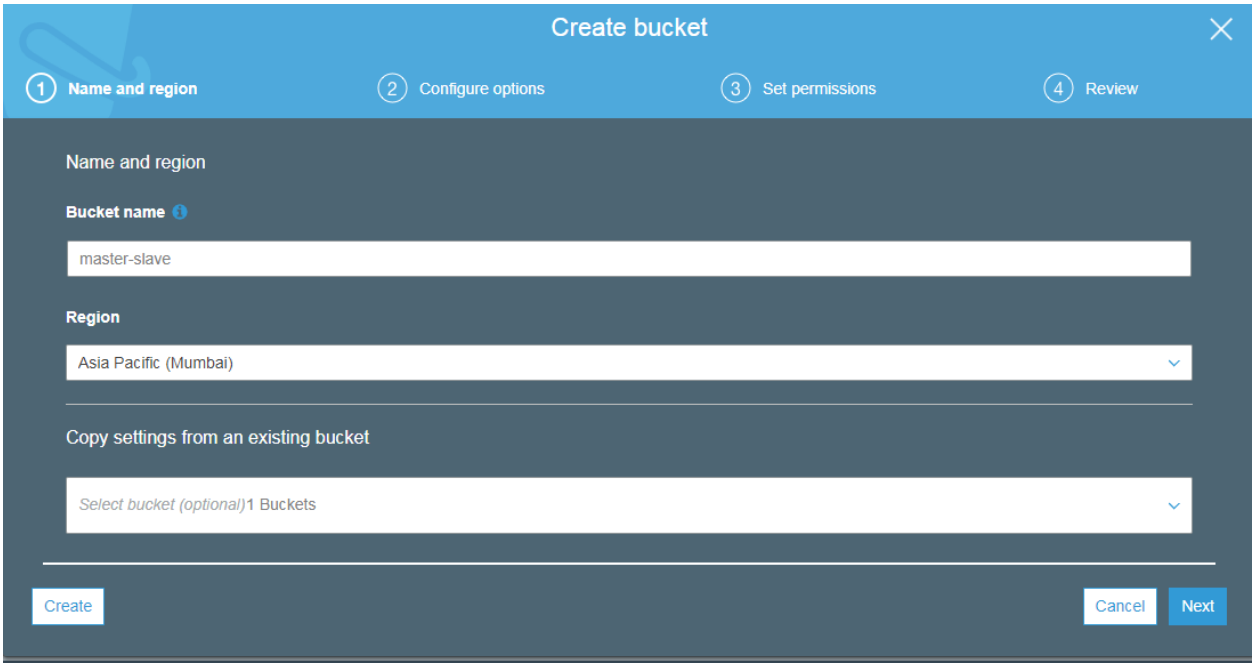


Task: - Configuration of S3 buckets in aws and apply IAM to ec2 instance.

Step 1: - Login in aws account and **click on s3 →Create a bucket**



Step 2: - Give name to bucket → select **region** → click on **create**

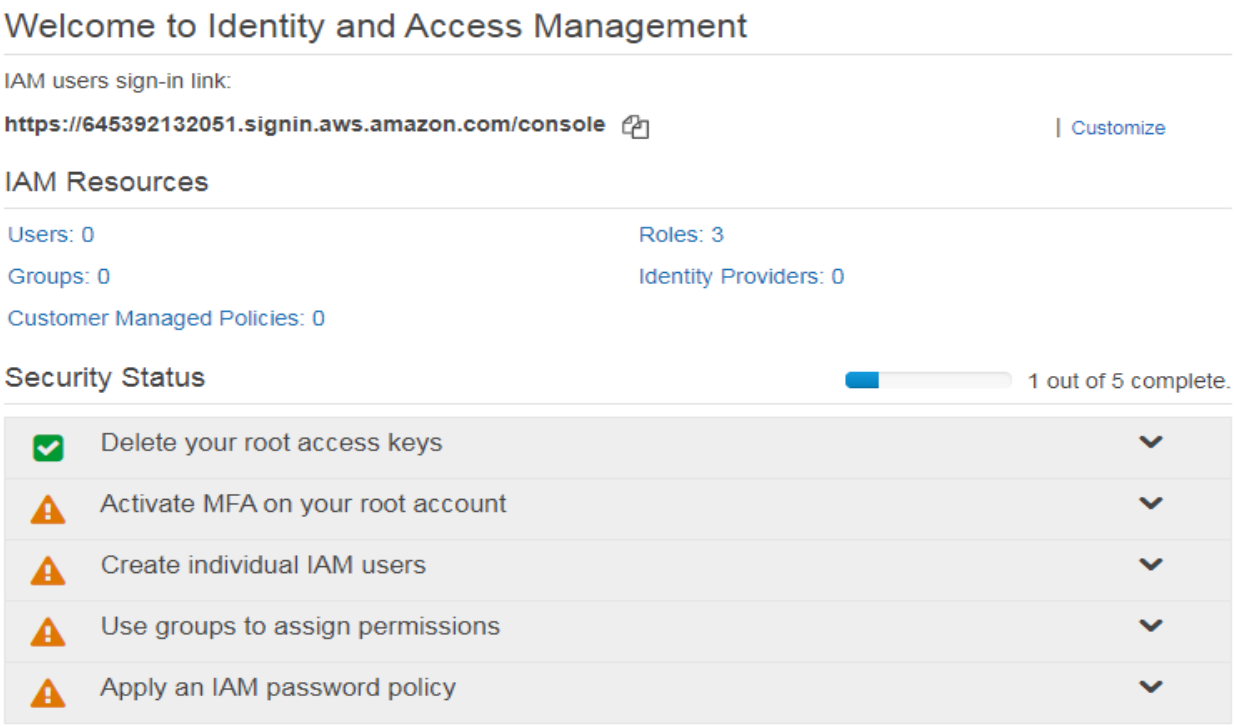


You have created bucket successfully.

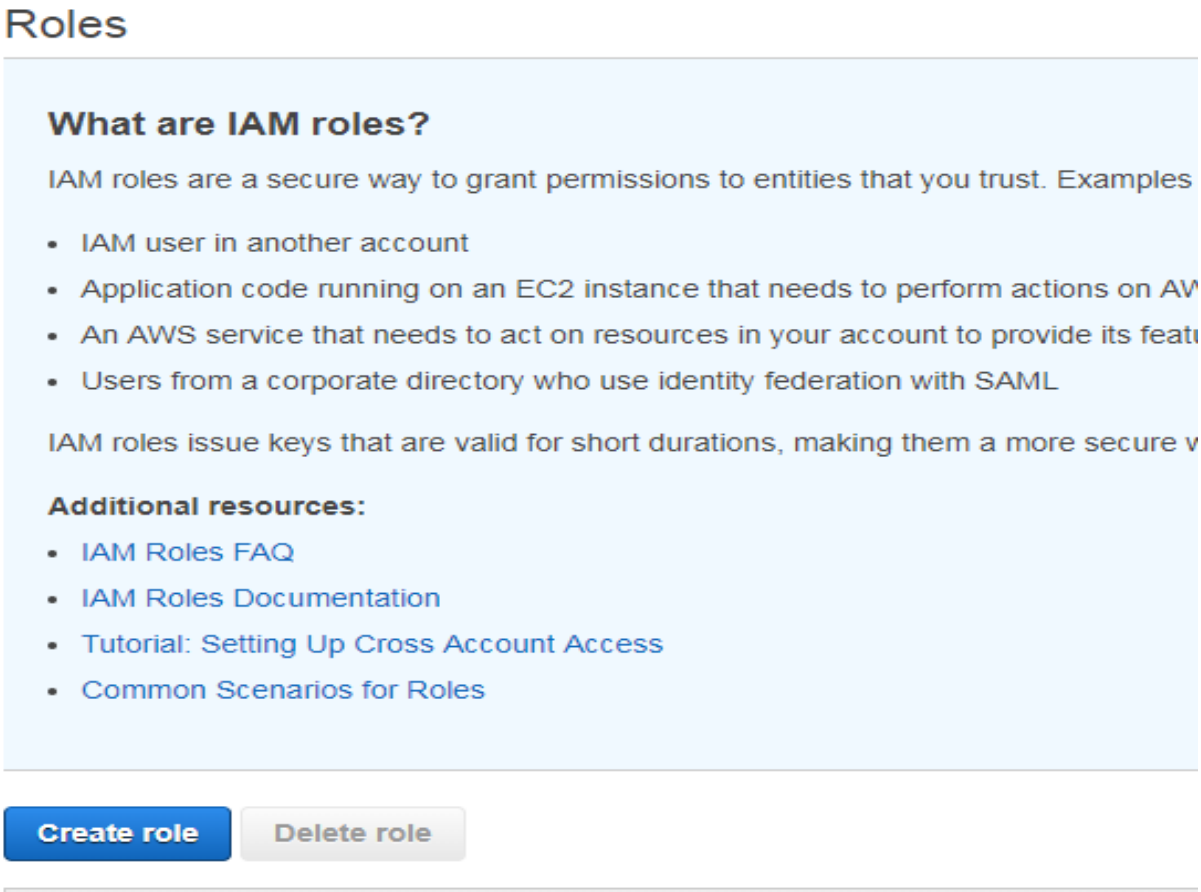
Step 3: - Create new IAM role with S3 full access and assign it to Jenkins server. Roles is used for one aws service with another.

Open IAM → click on roles → create role → ec2→ next permission → s3fullaccess→ tags

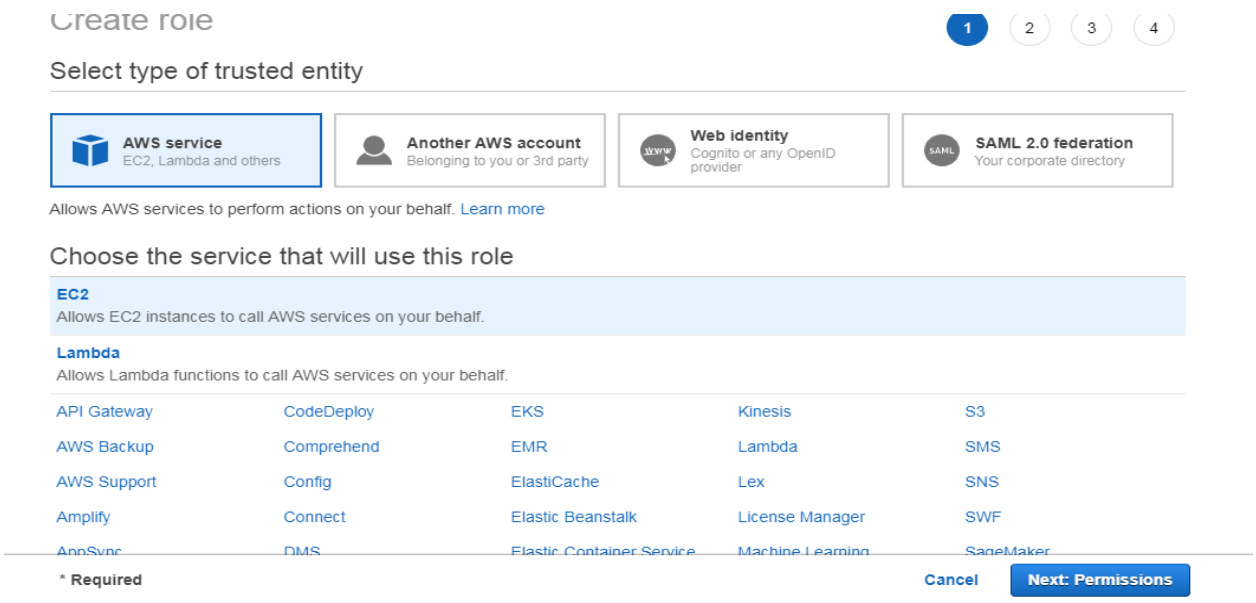
Step 3.1:- Open **IAM** and click on **Roles**



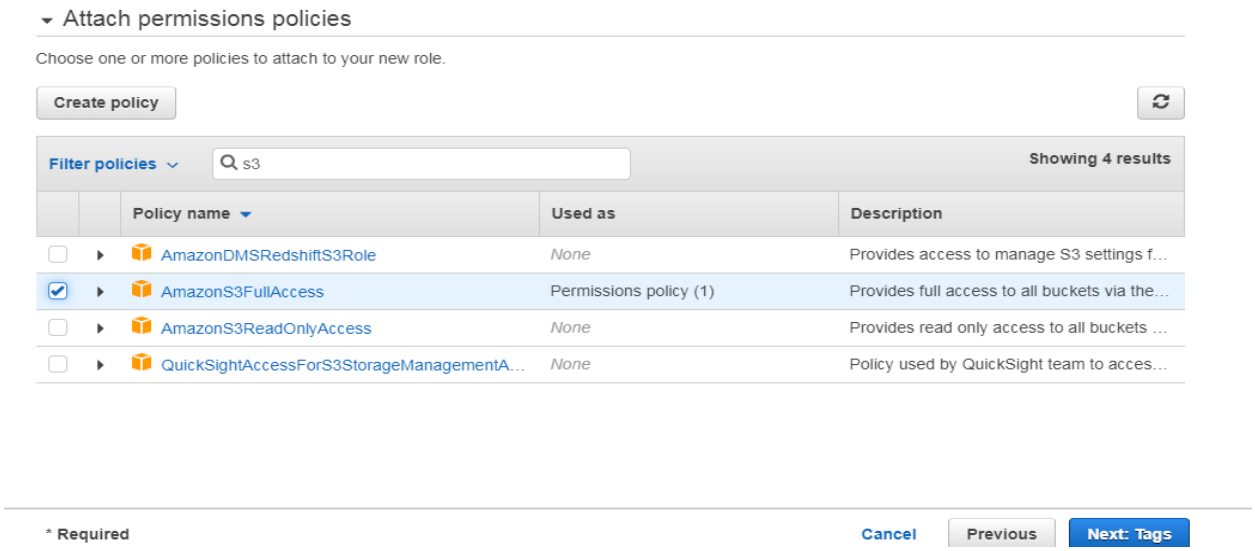
Step 3.2:- Now click on **create role**



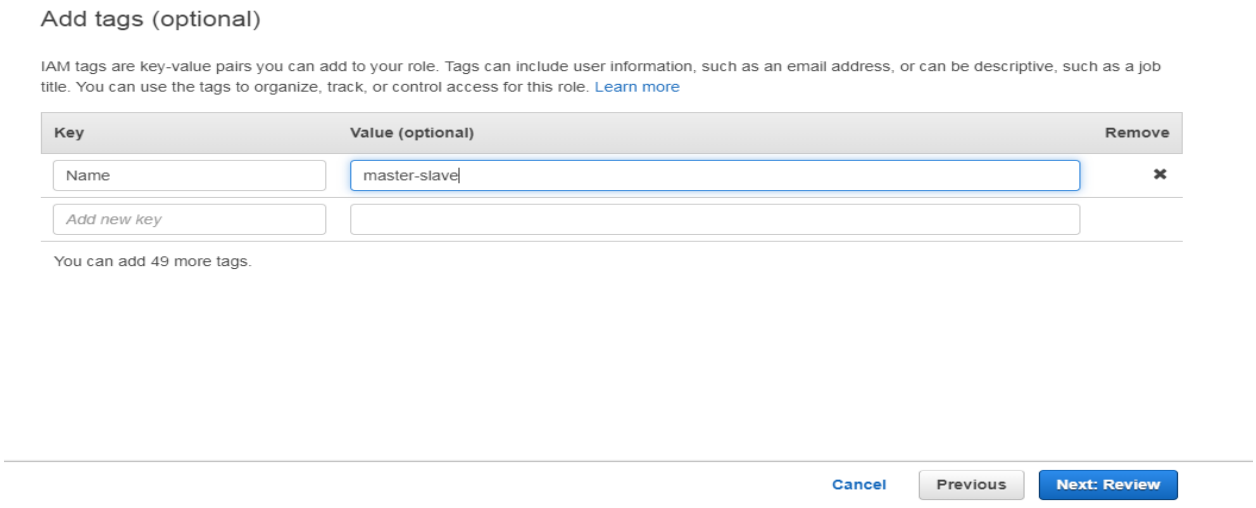
Step 3.3:- Now select **ec2** and click on **next permission**.



Step 3.3:- Now select **AmazonS3 FullAccess** and click on **Next Tag**.

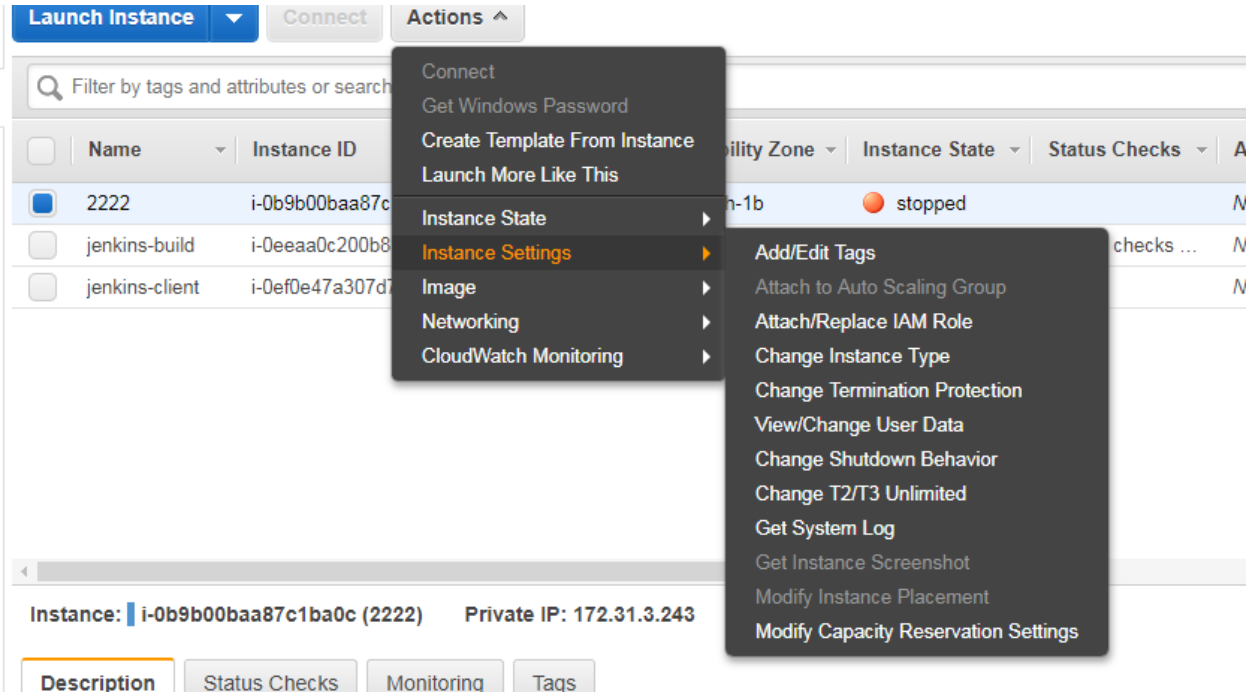


Step 3.4:- Now give **key and value** and click on **Next Review**.

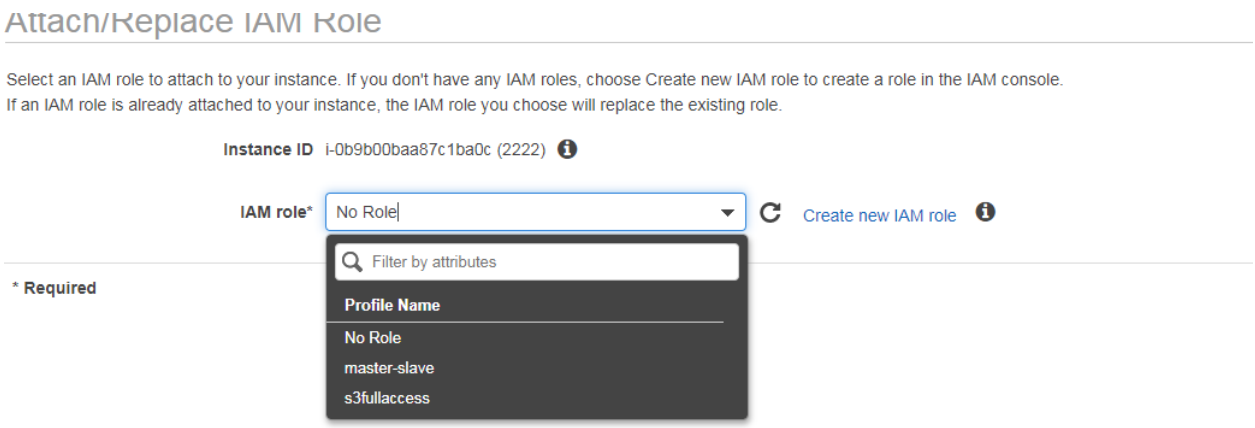


Step 4:- Now attach IAM role to EC2 and put IAM role

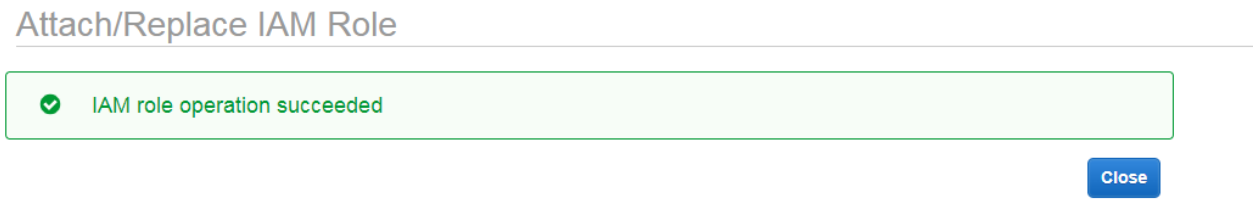
Step 4.1:- Go to **instance settings** and click on **Attach/Replace IAM role.**



Step 4.2:- Select Your **IAM role**.



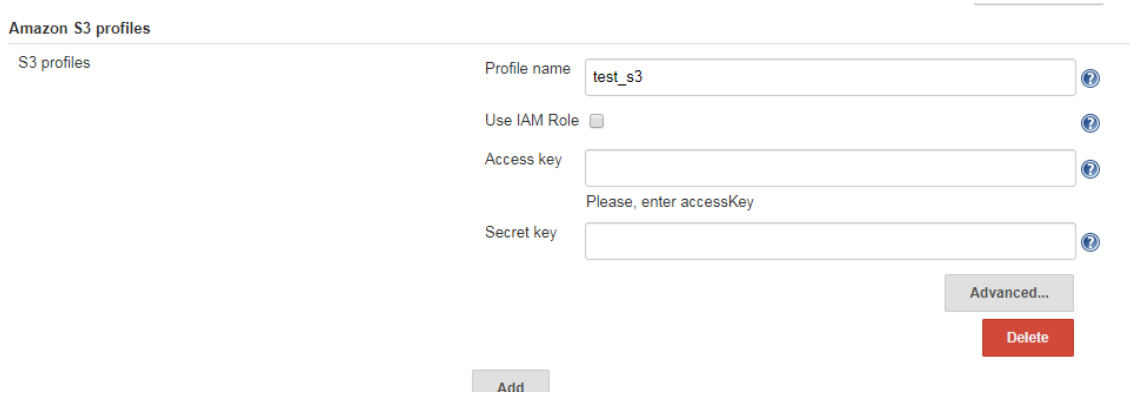
Your IAM role has successfully give to your instance.



Task: - Publish Artifacts to S3 bucket

Step: - 1 Generate Access Keys for accessing s3.

Step: - 2 Then go to manage Jenkins → configure system → then Amazon S3 profile.

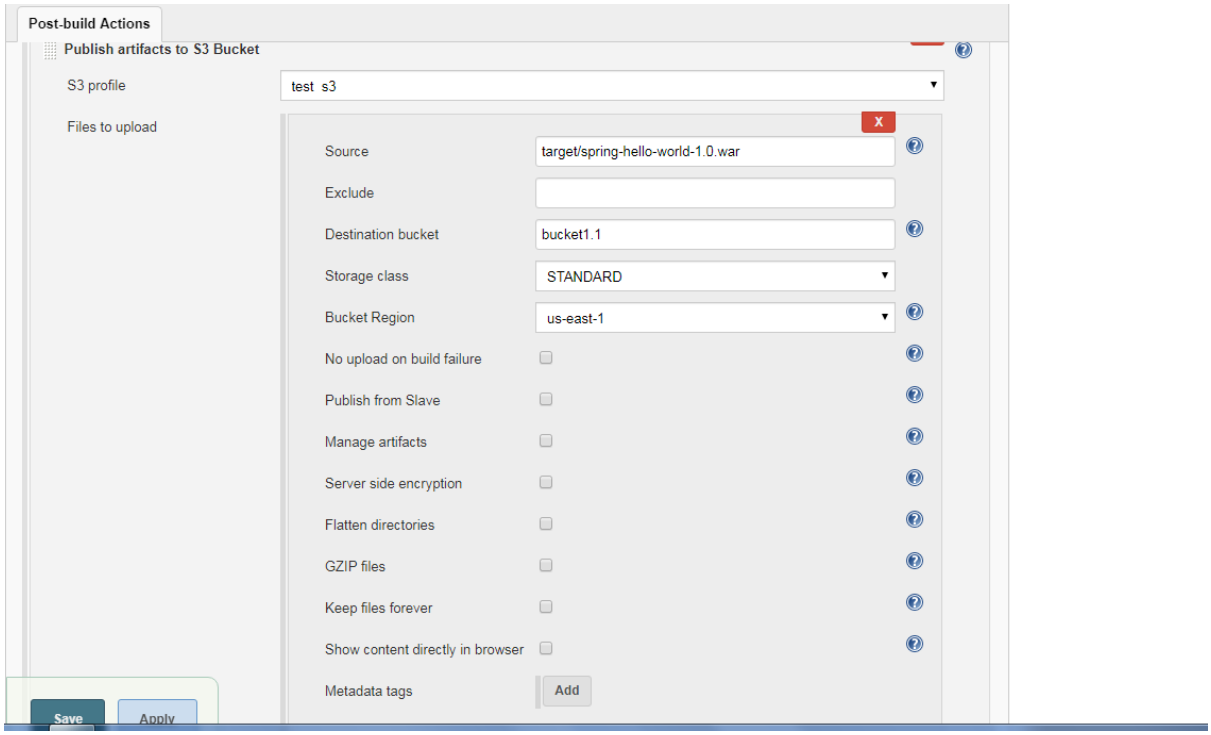


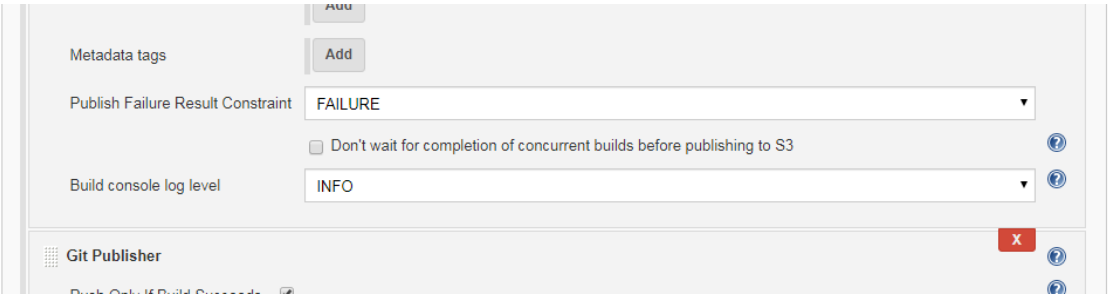
Step: - 3 enter the Access key and secret key and profile name. Then click Apply and Save.

Step: - 4 Select maven project.

Step: - 5 configure the project with necessary options.

Step: - 6 In "Post-build Actions" select Publish artifact to S3 bucket.





Step: - 7 Select the S3 profile which you have added in the configure system.
Step: - 8 Enter the source where the .war file will be generated
Step: - 9 Enter Destination Bucket (name of the bucket)
Step: - 10 Select the Bucket region and click on apply and save.

Shell script to build maven project in Jenkins

- **cp target/spring-hello-world-1.0.war /var/lib/jenkins/apache-tomcat-8.5.40/webapps/helloworld.war**
- **BUILD_ID=dontKillMe**
- **sh /var/lib/jenkins/apache-tomcat-8.5.40/bin/startup.sh**

Configure npm in Jenkins

- **Shell script to run npm in background in jenins**
- **npm install**
- **BUILD_ID=dontKillMe**
- **npm run start:dev &**

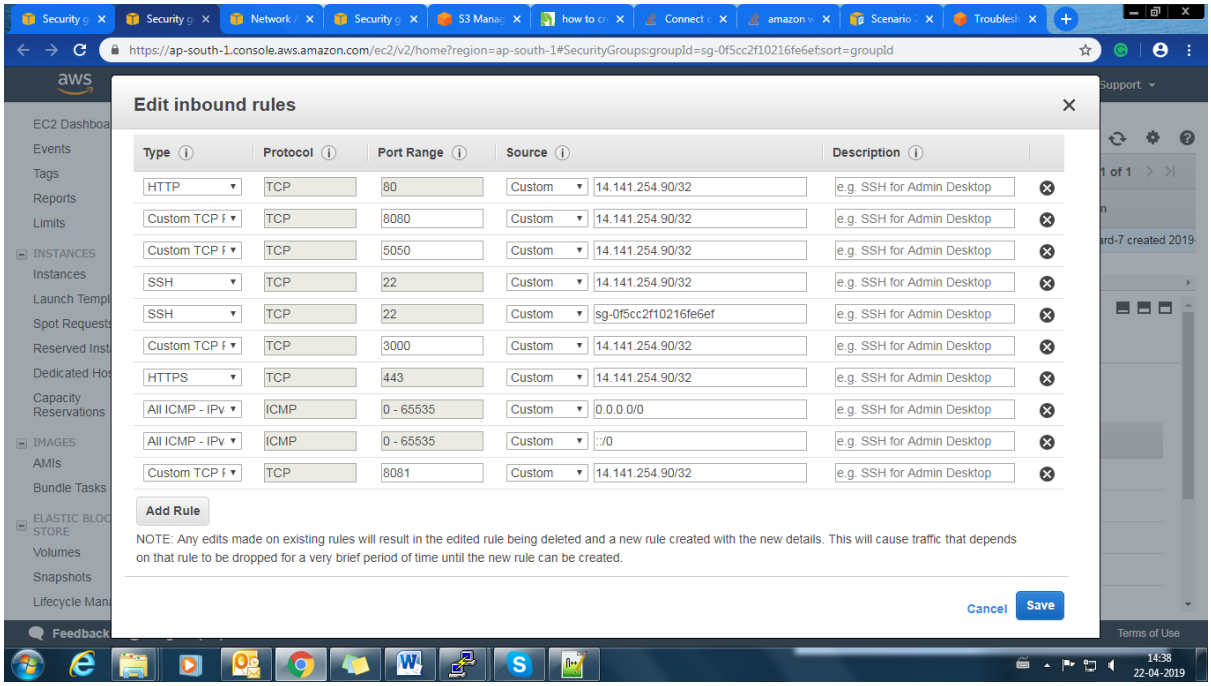
Task: - Server-client configuration in Jenkins (Using master-slave configuration build job and store artifact in S3 bucket and deploy it on client)

Install Plugin: - ssh-plugin

Requirement

- we have two machine one is server and another is client we have to do ssh and deploy through client.
- Check and add in the same VPC

Step 1: - Add inbound rules for instance.

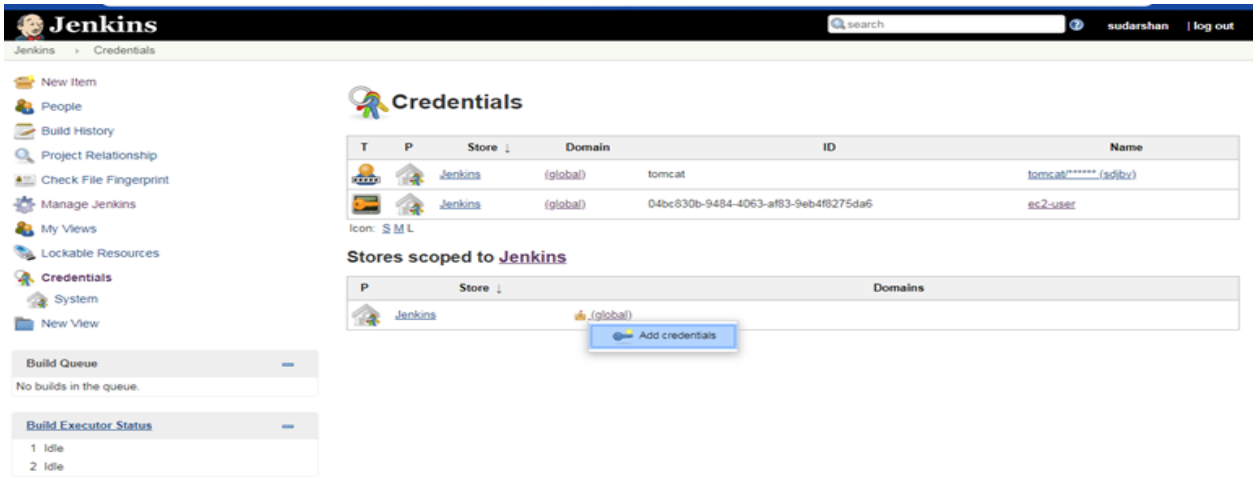


Step 2: - check you are able to do ssh from one instance to another.

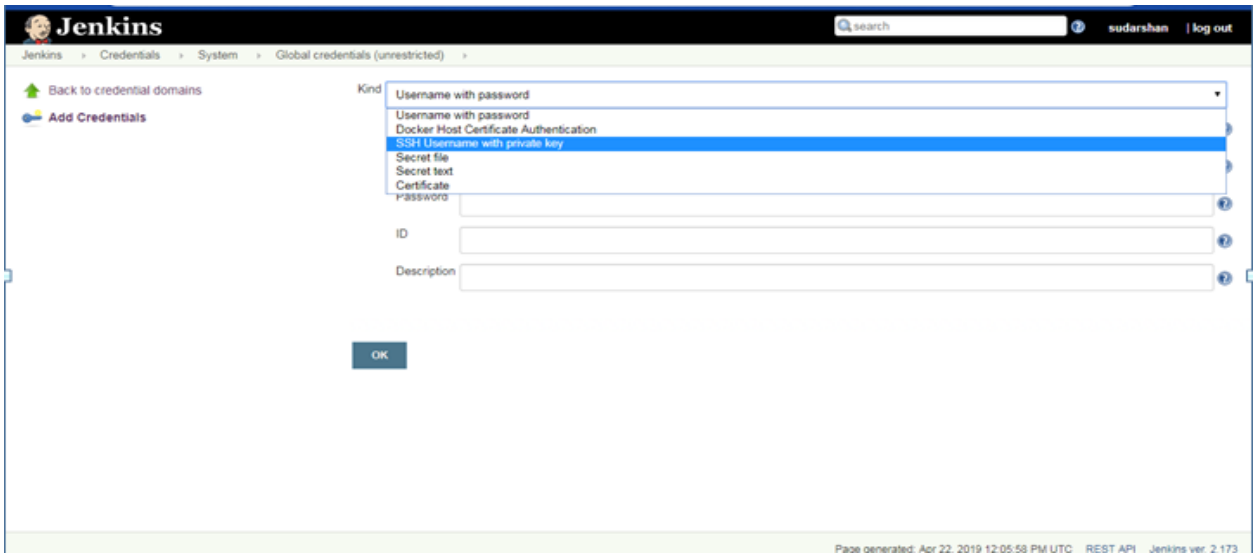
- **ssh -i AMI-jenkins.pem ec2-user@ec2-35-154-111-25.ap-south-1.compute.amazonaws.com**
- **ec2-user@ private ip for connectivity**
- **ec2-user@public ip for internet**

Troubleshooting:- we add https because server is not able to ping because security group was changed.

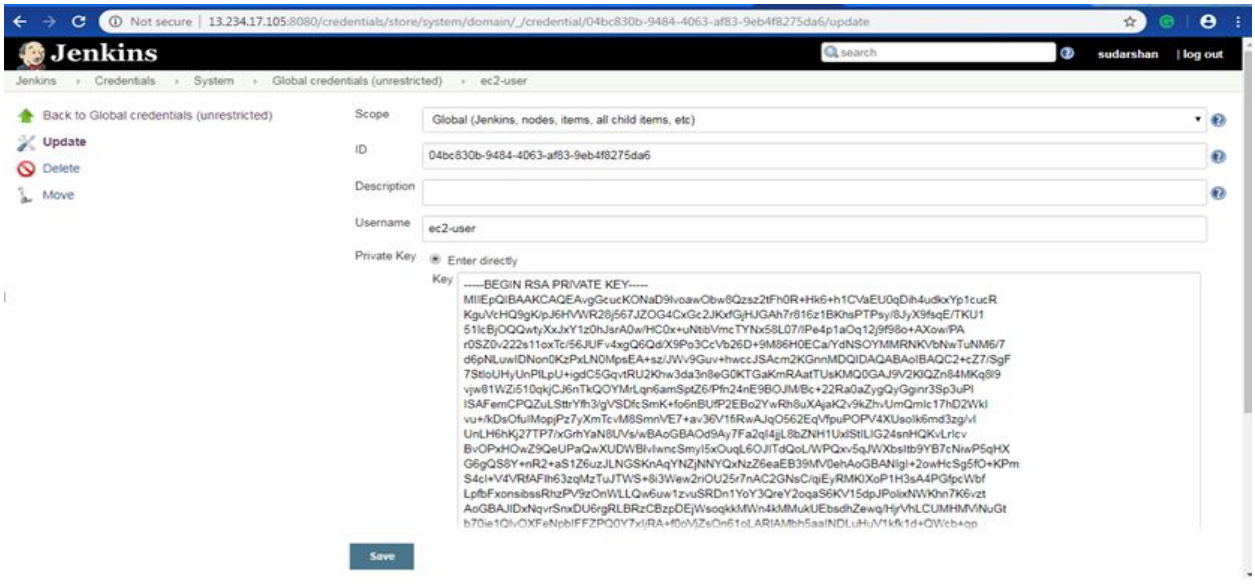
Step 3: - Add credentials



Step 3.1: - Select ssh username with private key.

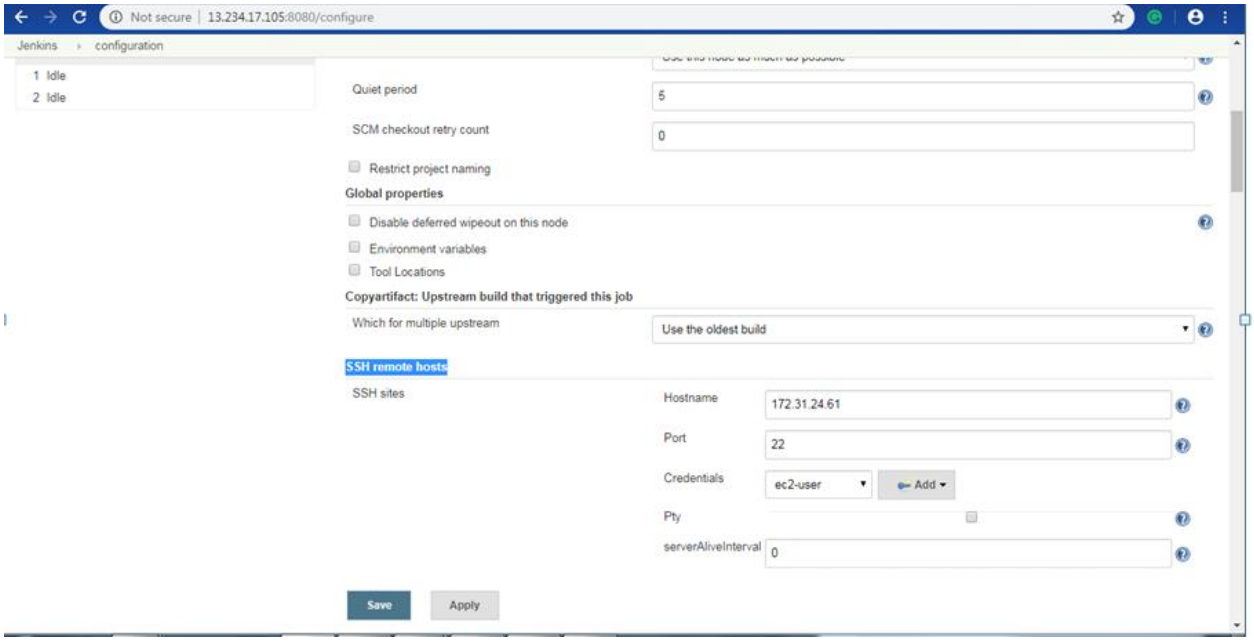


Step 3.2: - Client username and private key .pem



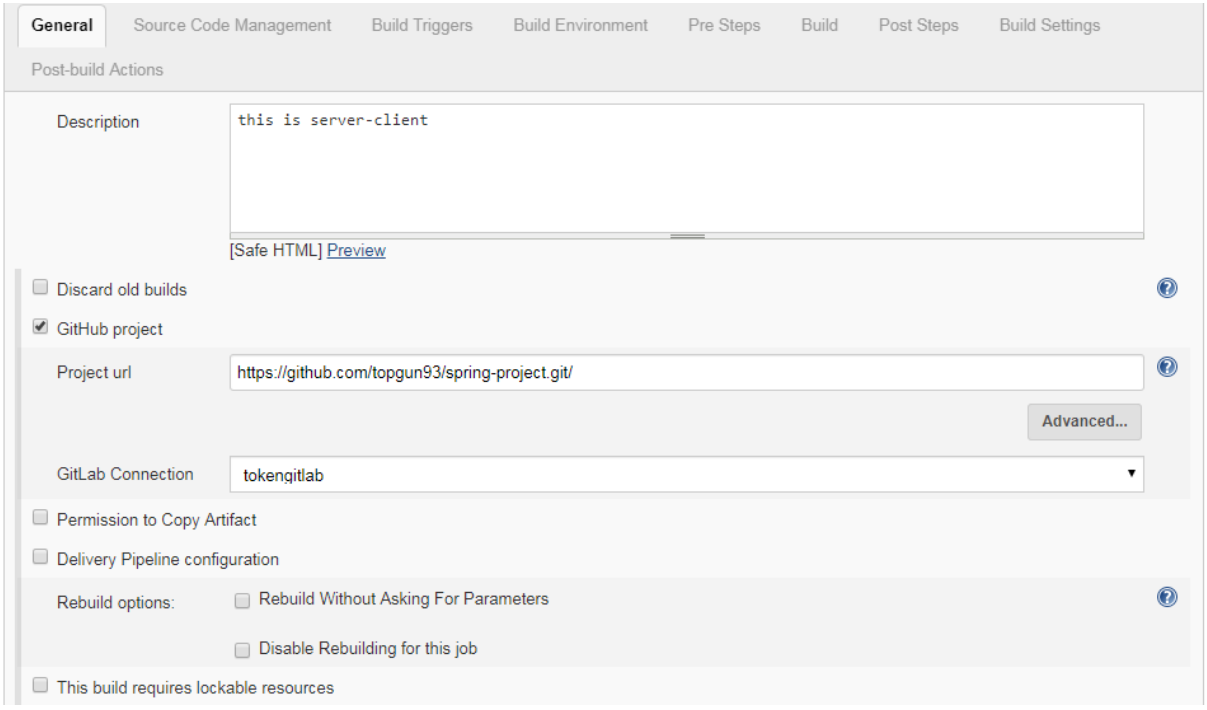
Step 4: - Now go to **manage Jenkins → configure system → select ssh remote hosts**

Add private ip address, port no. and credential then save it.

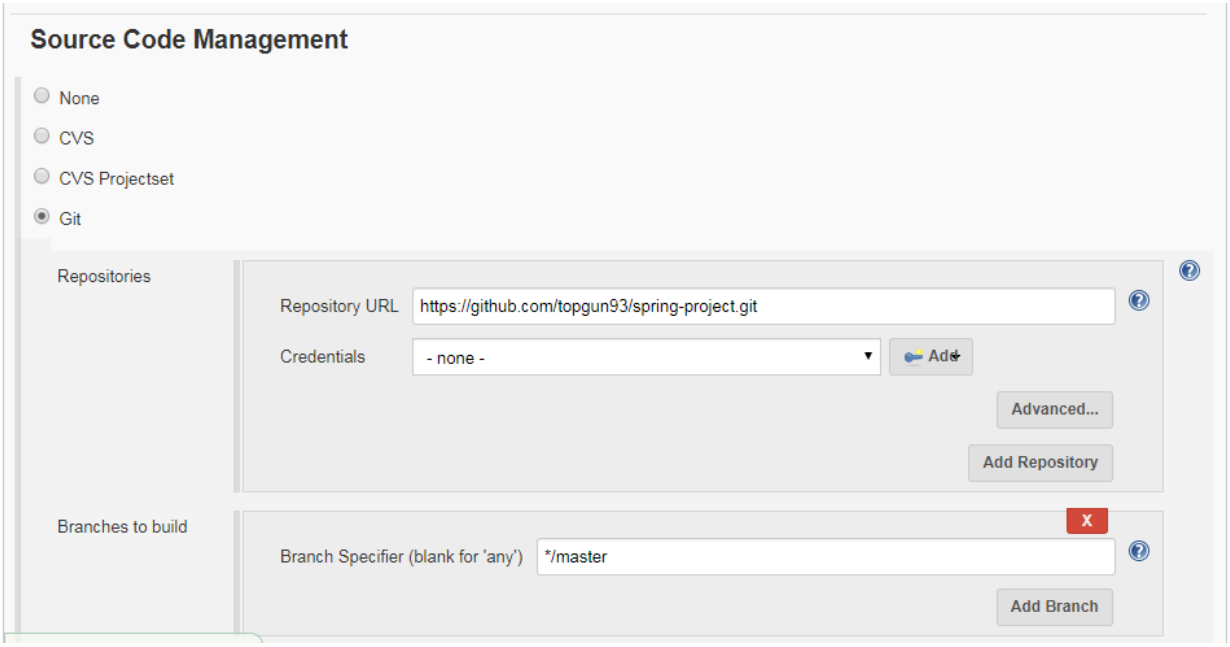


Step 5: - In client we need java 1.8 and tomcat

Step 6: - Give URL of repo.



Step 7: - Repo url



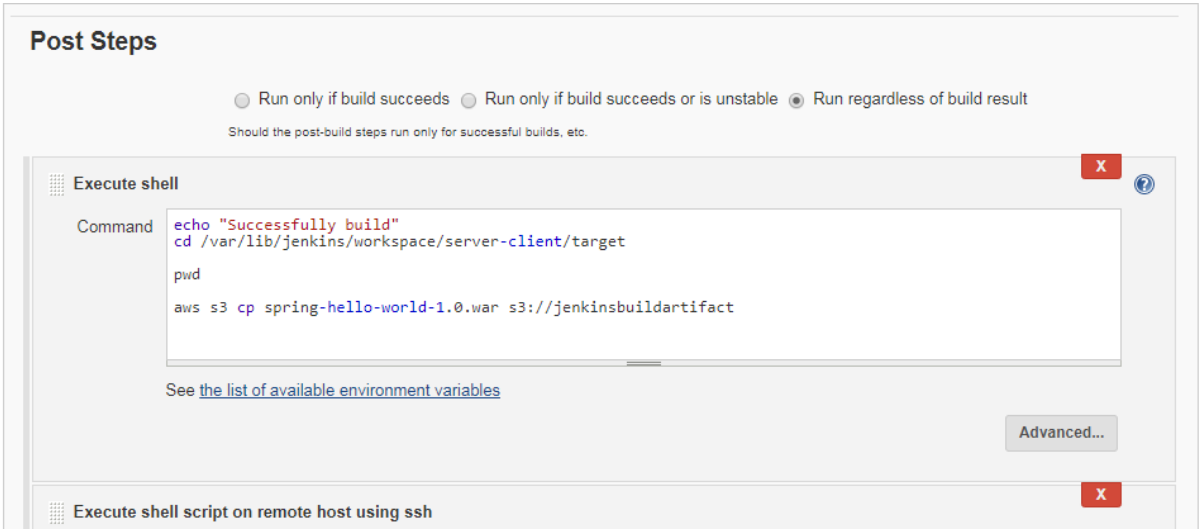
Step 8: - Give goals

Step 9: - In post step, we add execute shell

- **cd /var/lib/jenkins/workspace/artifact_project/target**

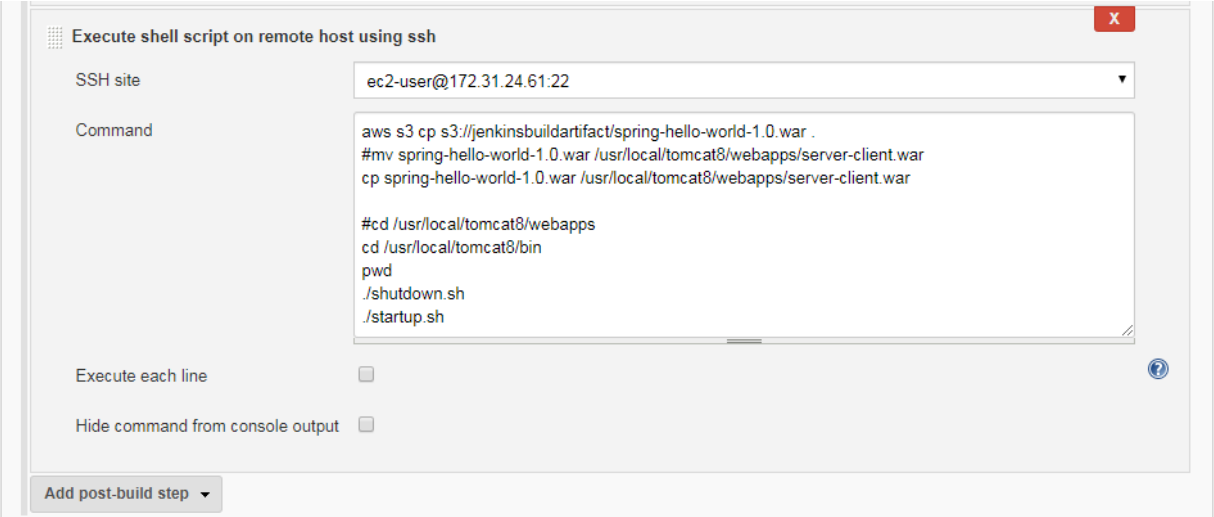
Now copy war file in aws bucket

- **aws s3 cp spring-hello-world-1.0.war s3://<<BUCKET NAME>>**



Step 10: - Now if you want to download then go any location and run the below command

- **aws s3 cp s3://jenkinsbuildartifact/spring-hello-world-1.0.war .**

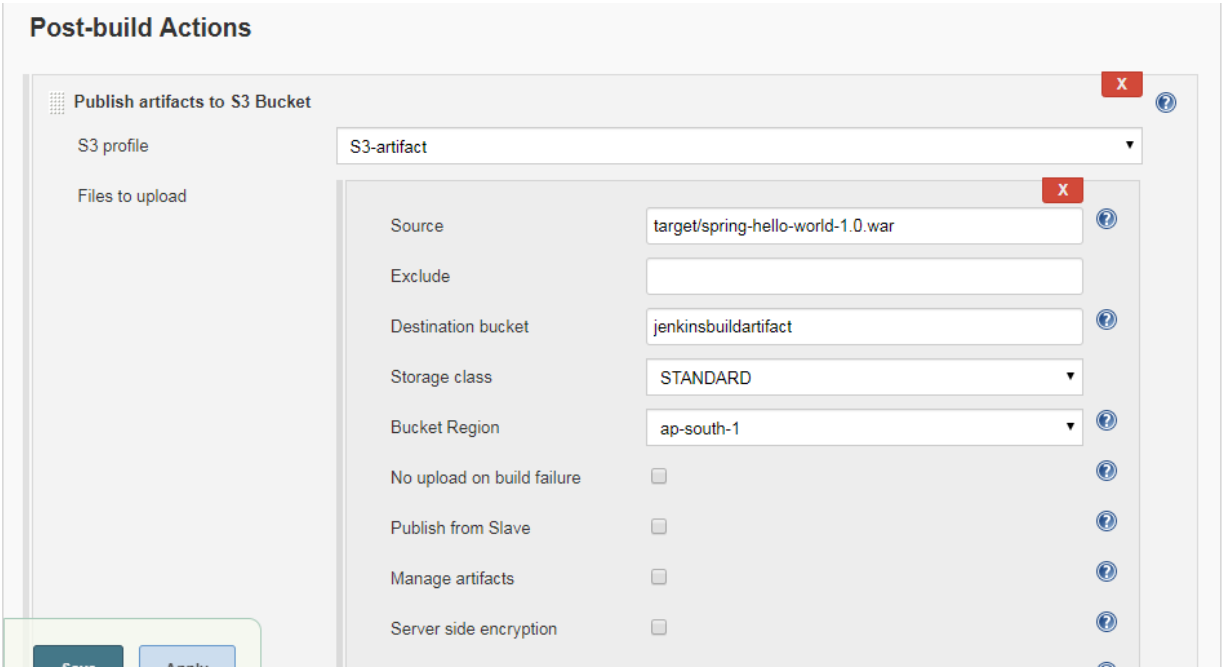


Step 11: - click on save and build it.

Another way to store artifact in S3

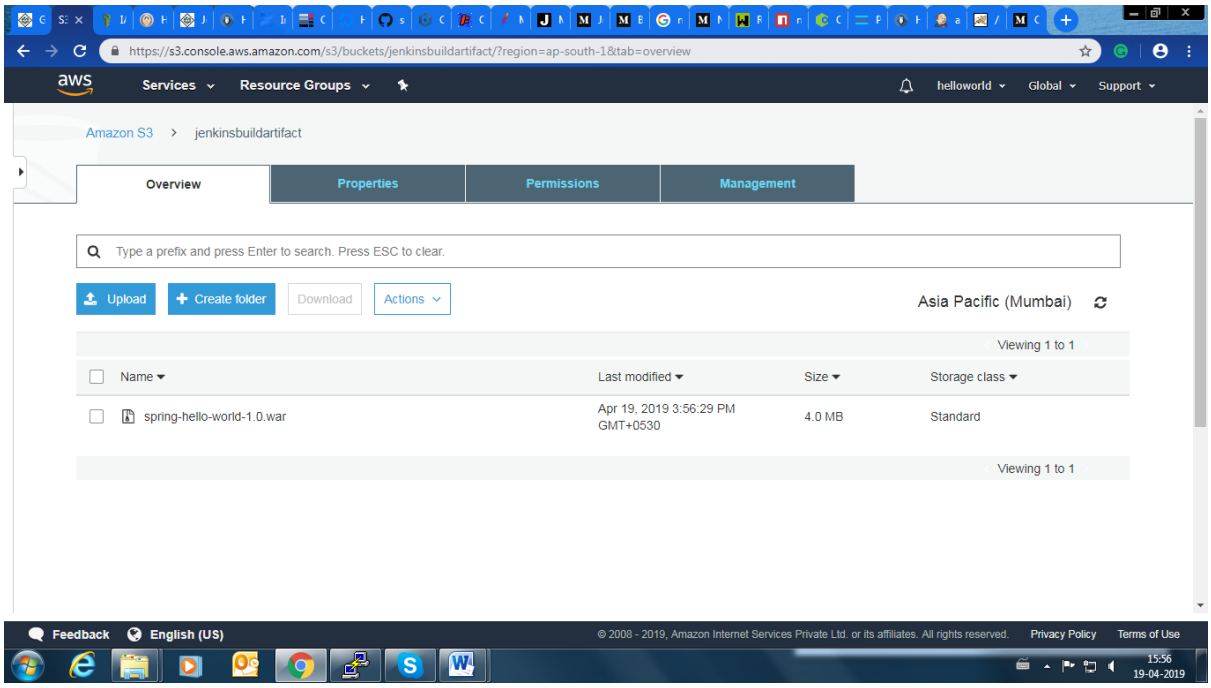
Install Plugin: - S3 publisher plugin

Step 1: - In post-build actions, add Publish artifacts to s3 bucket and add source and region of bucket



Step 2: - Click on save and start build.

Step 3: - you can check by refreshing the s3 bucket



Step 4: - Check the deployment on http://ip:port/manager

Task: - **Installation of dependency packages of python and deploy on server.**

Note: - For deployment of python project we need flask. Flask is a web application framework for python.

Repository: - https://github.com/sudarshan1995/Flaskex.git

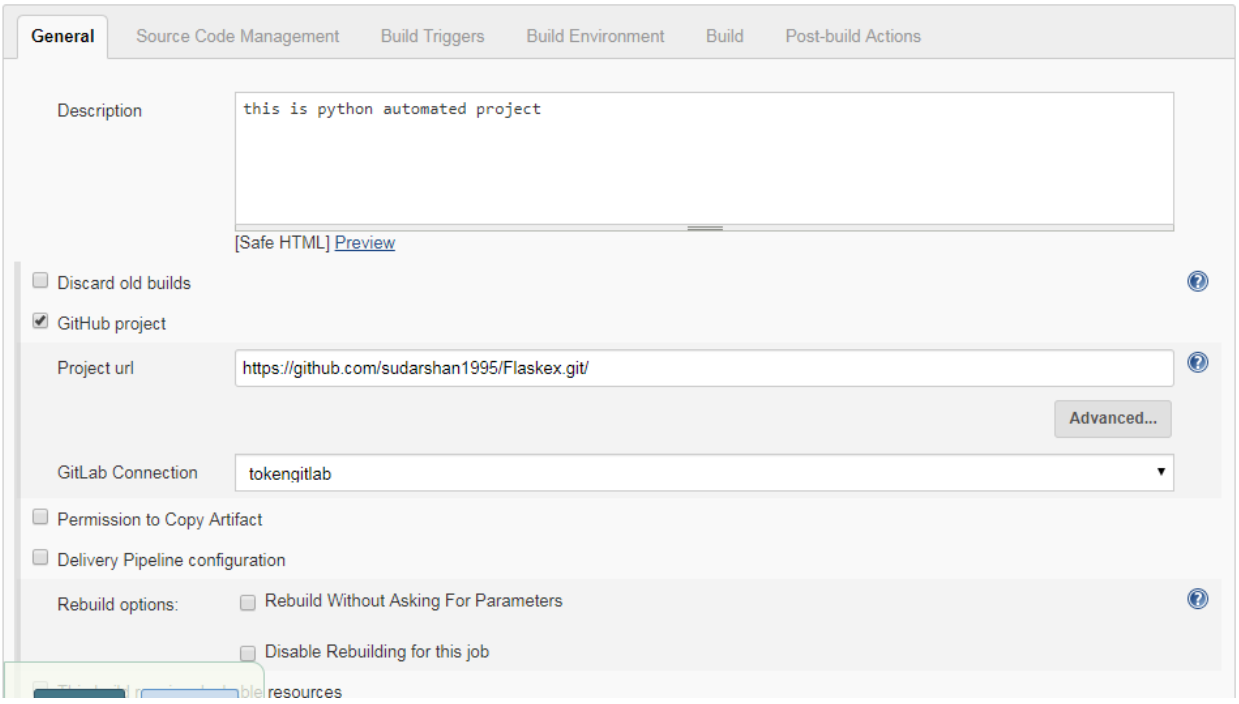
From console

- **git clone https://github.com/anfederico/Flaskex**
- **cd Flaskex**
- **pip install -r requirements.txt**
- **python app.py**

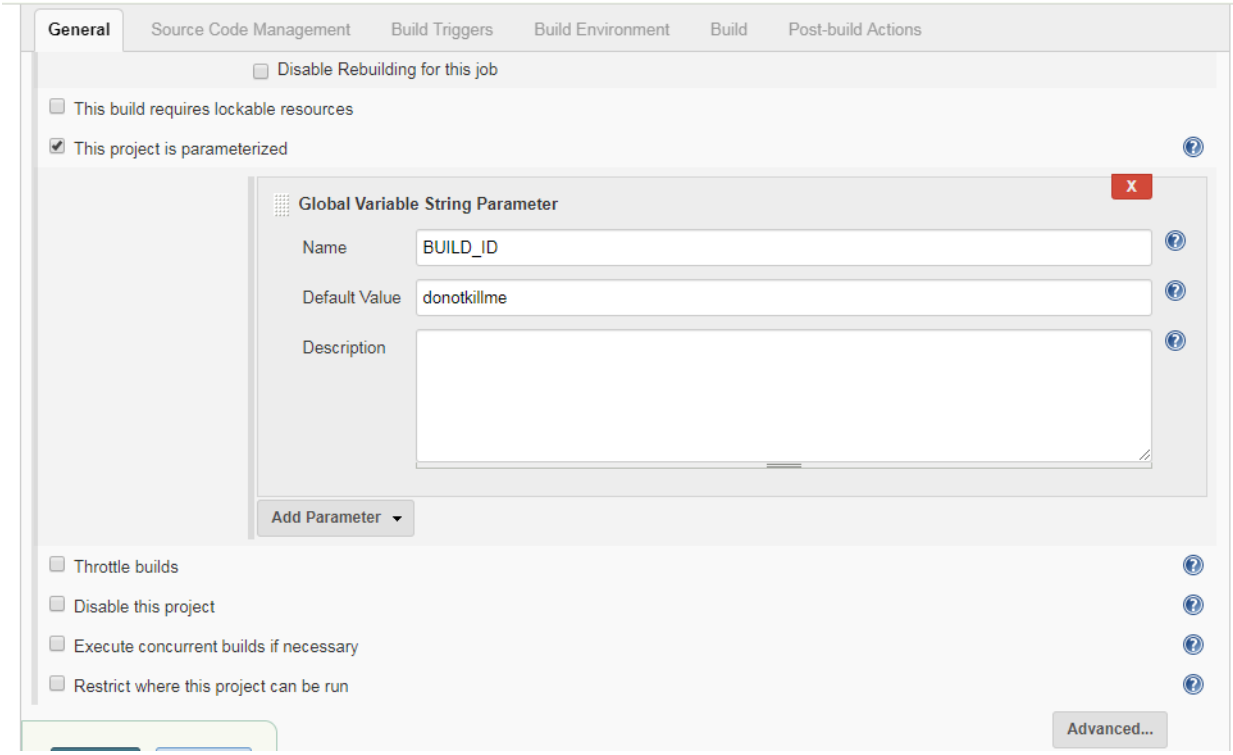
Jenkins configuration

Step 1: - New item → give project name → select **freestyle project**

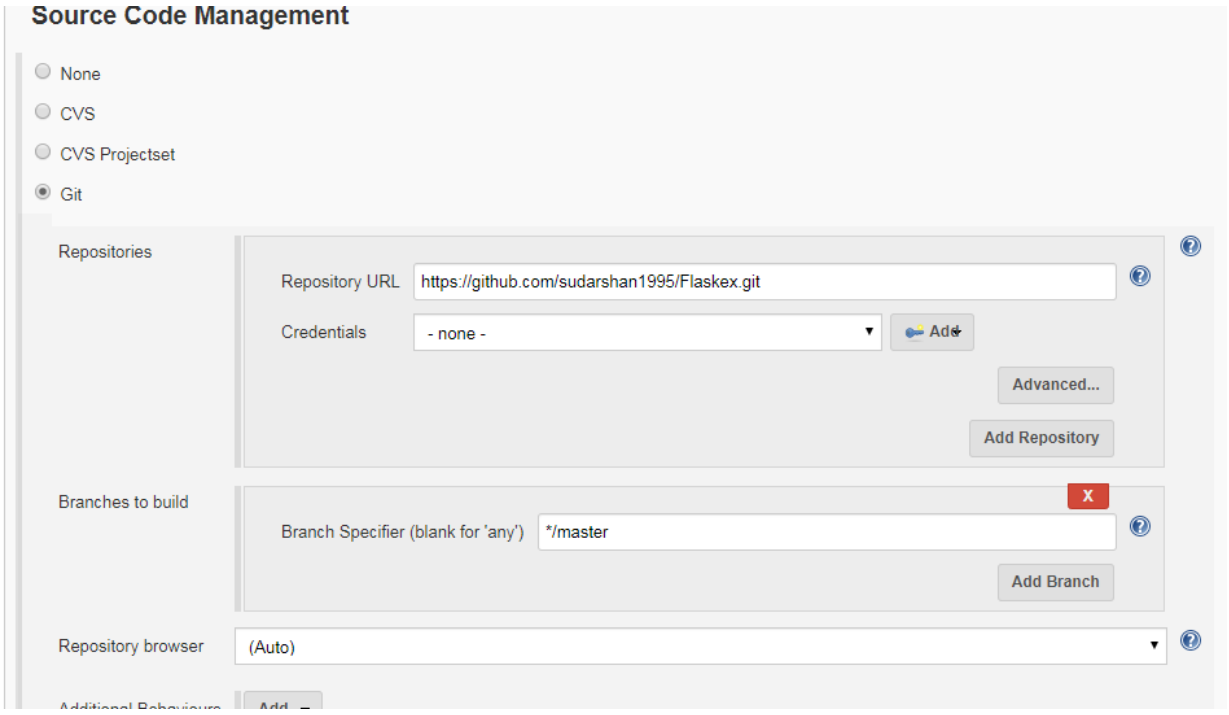
Step 2: - In general section, give project url



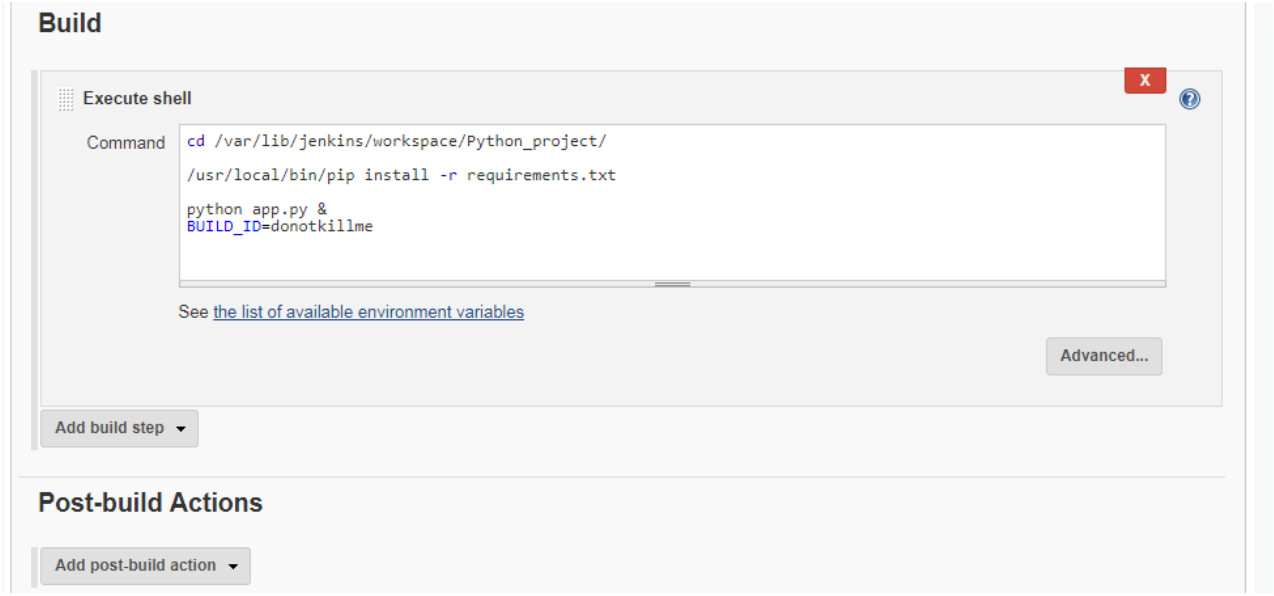
Step 3: - set parameter



Step 4: - In source code management, Enter repository URL and branch



Step 5: - In build section, add build step execute shell



Step 6: - save and build

Troubleshooting: - I face the post issue it was hitting on local address

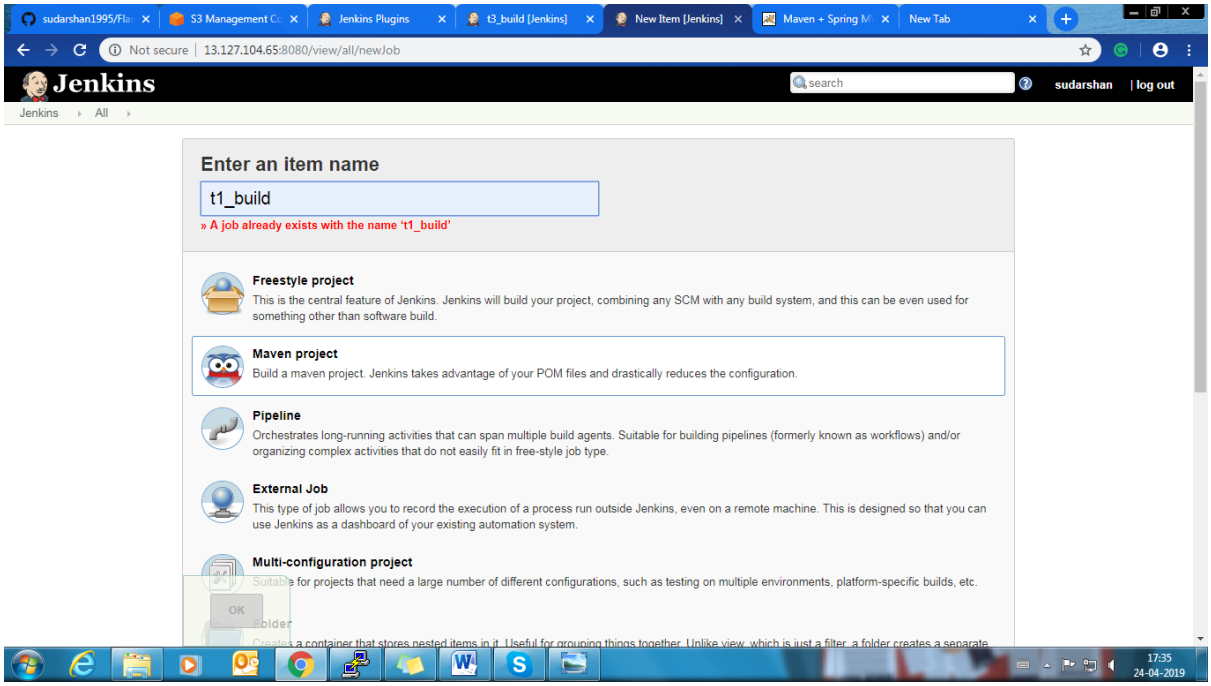
So we go in **vi app.py** and add this there in last line: - app.run(host="0.0.0.0",debug=True

Task: - Configure three jobs in Jenkins for build; upload artifact and deployment on server (upstream and downstream).

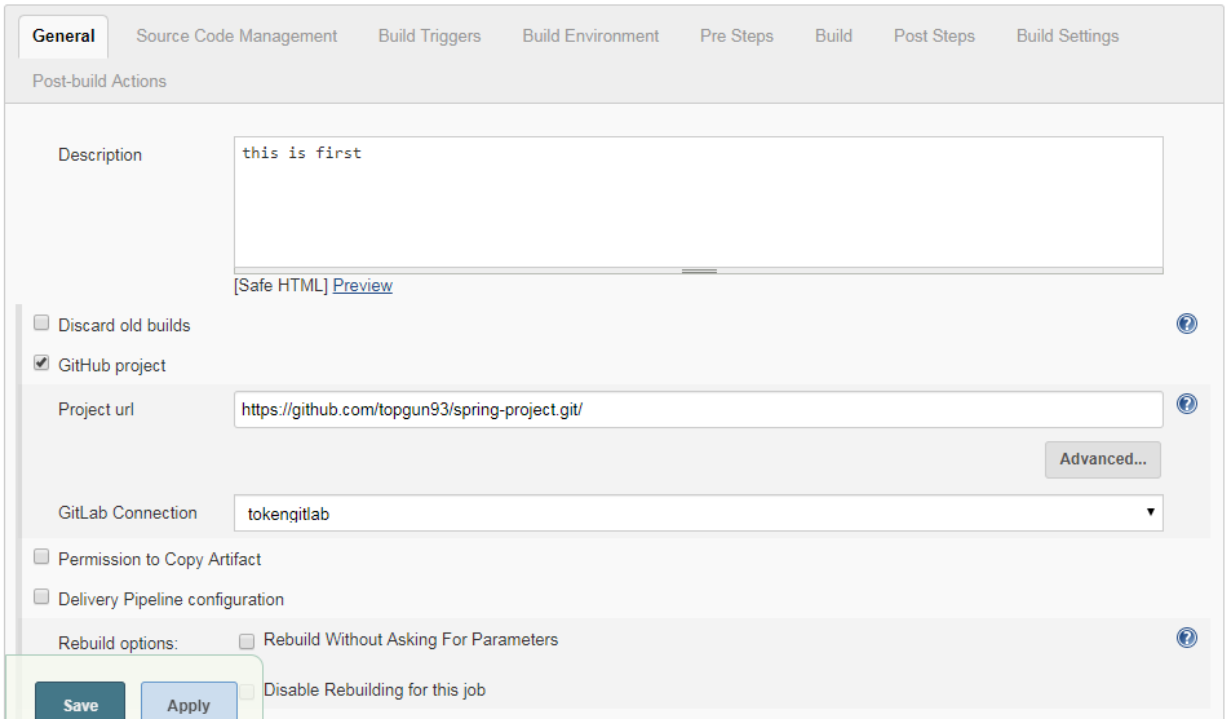
Jenkins configuration

Install Plugin: - Copy data to workspace

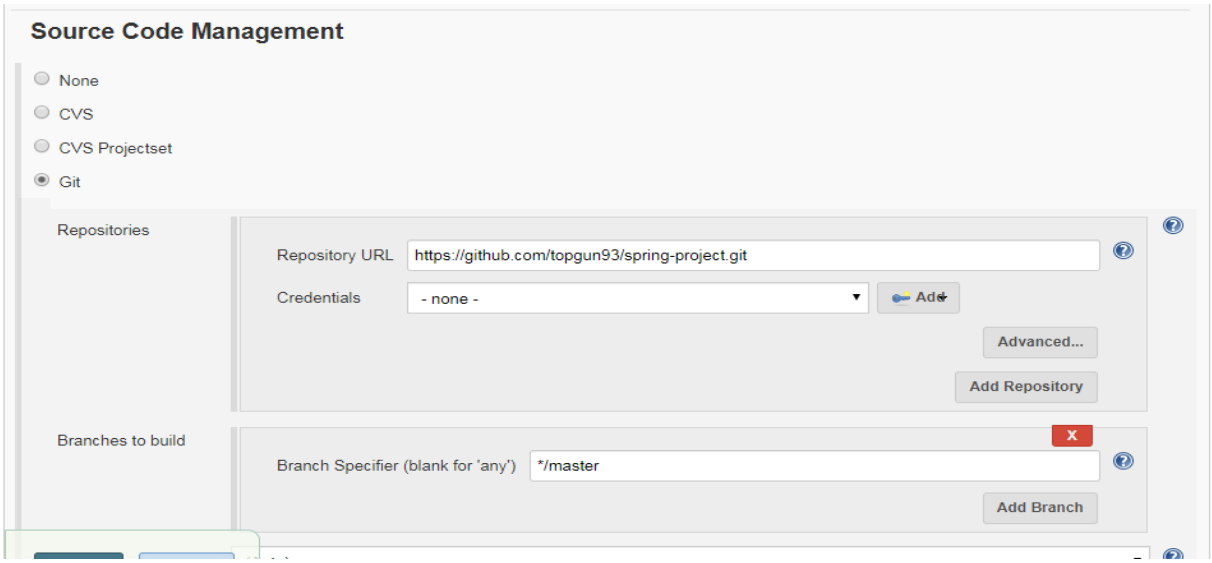
Step 1: - New item → give project name → select **maven project. (t1_build)**



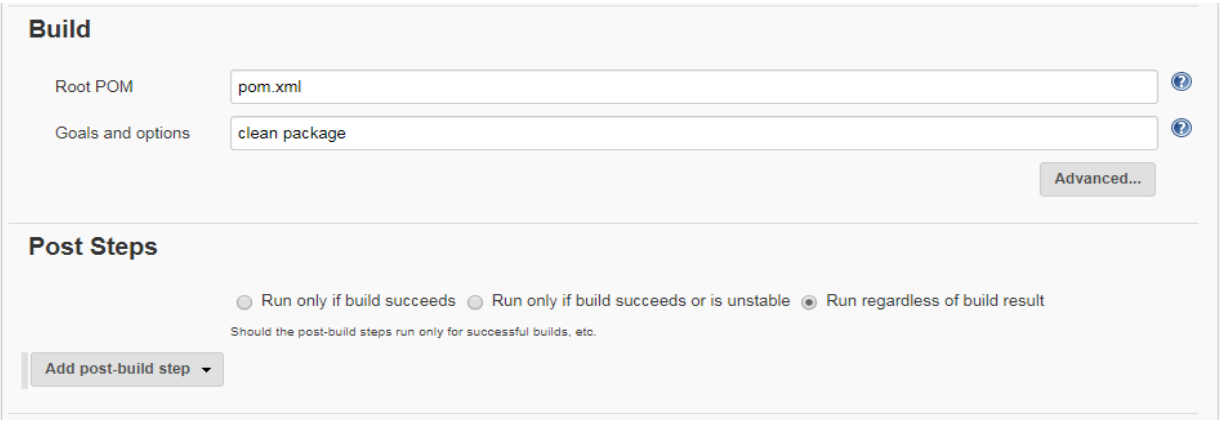
Step 2: - - In general section, give project url



Step3:- In source code management, Enter repository URL and branch



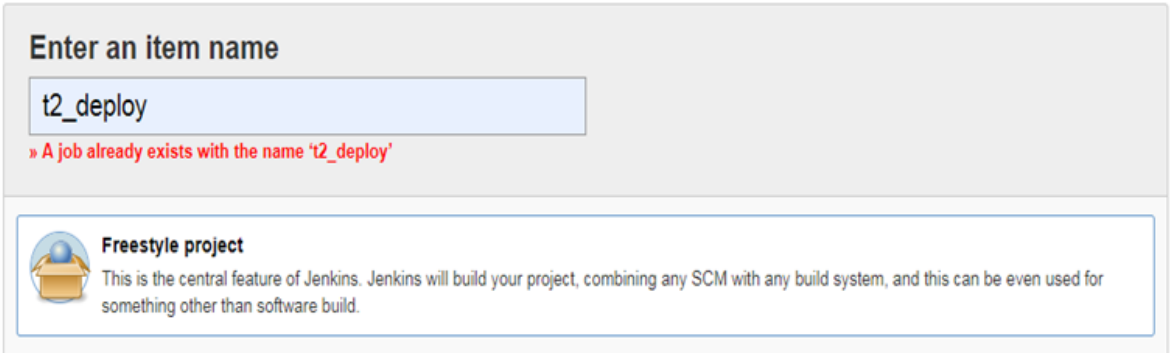
Step 4:- In build section, set **goals and options**



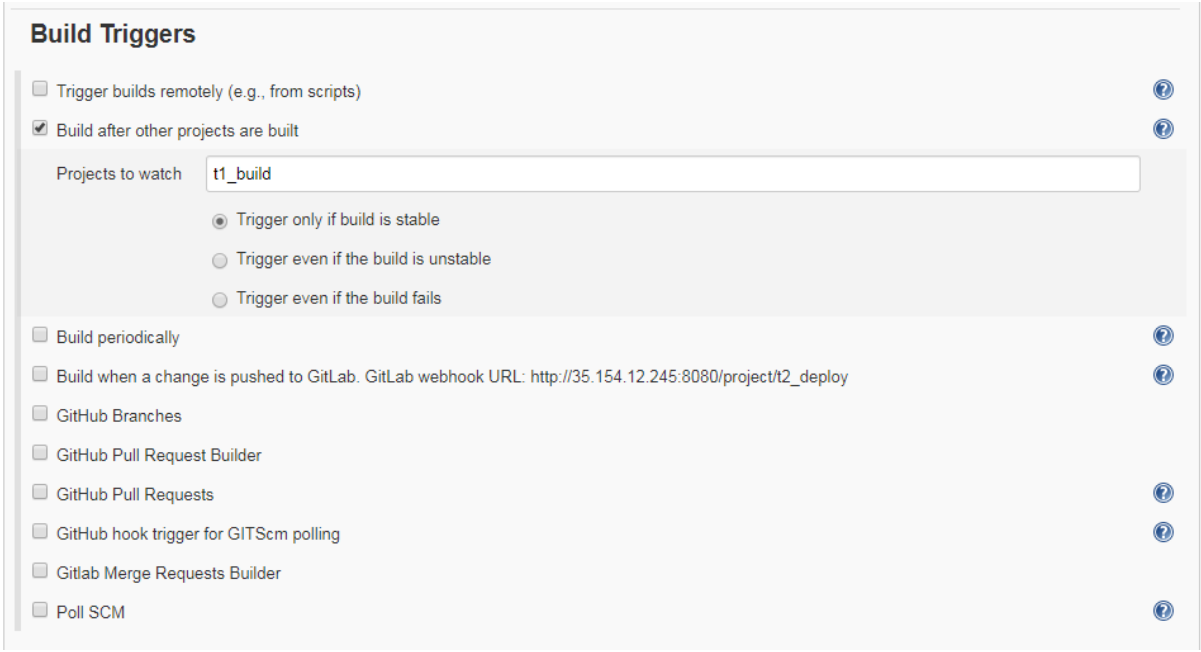
Step 5: - After this **apply and save.**

Configure another project t2_deploy.

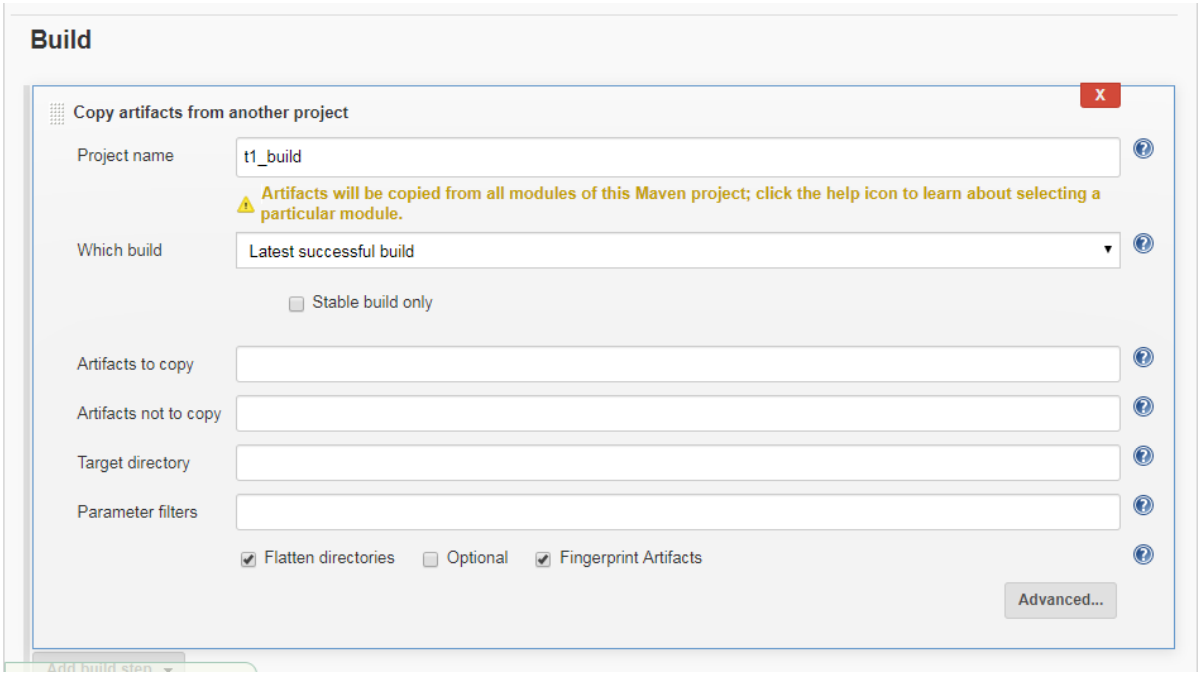
Step 6: - New item → give project name → select **Freestyle project. (t1_build)**



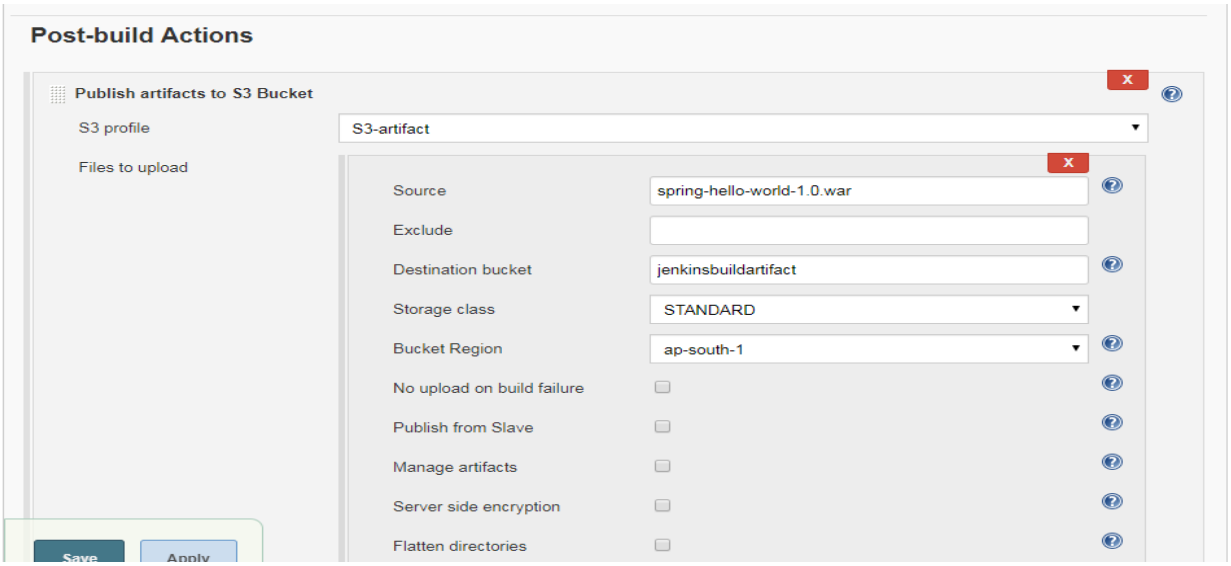
Step 7: - we set build trigger when t1_build is completed then it will start.



Step 8: - Copy build artifact from t1_build project.



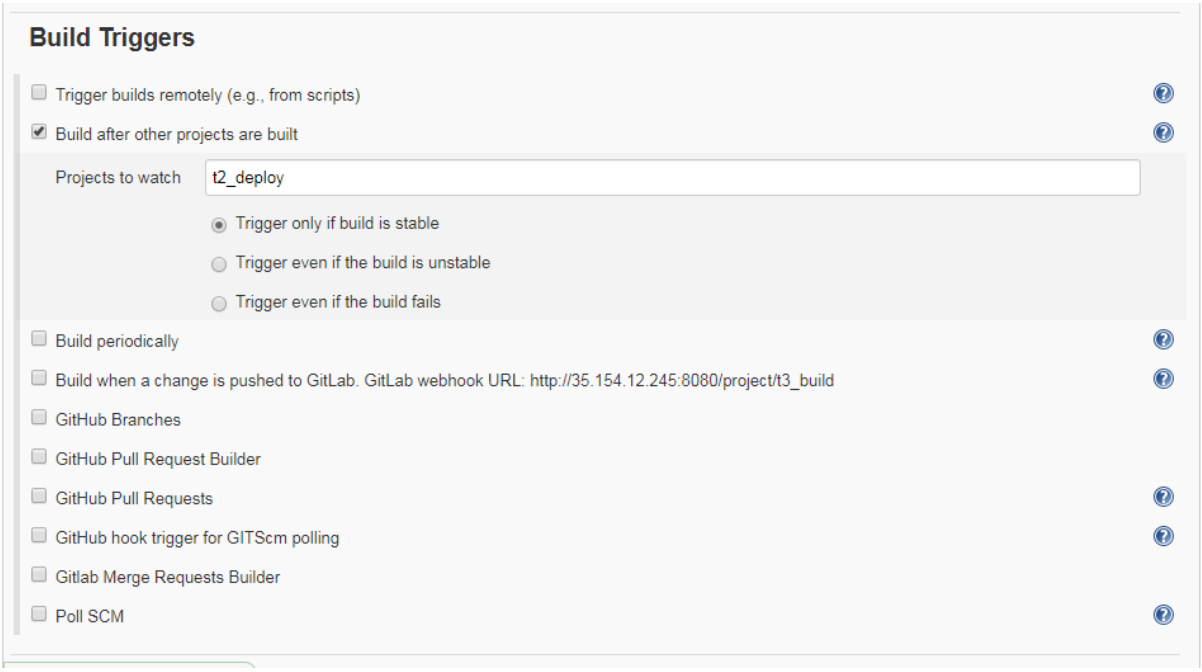
Step 9: - In post build actions, select publish artifact in s3 bucket to upload in S3 bucket.



Step 10: - click on **save and build** it. The artifact will store in s3 after successful build.

Configure another **project t3.**

Step 11: - we set build trigger when t2_deploy build is completed then it will start.



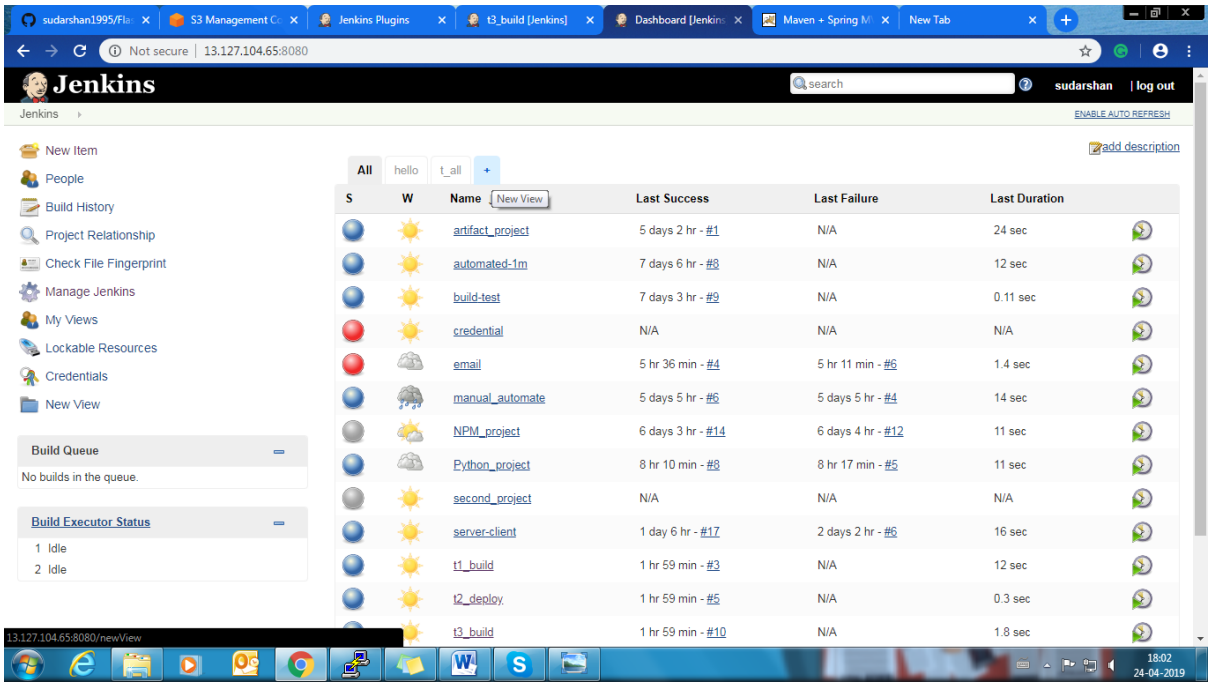
Step 12: - In Build section, select Execute shell and put artifact in tomcat server from s3 bucket. **(check the permission and give it once more in the last** Jenkins:Jenkins **)**

Step 13: - click on save.

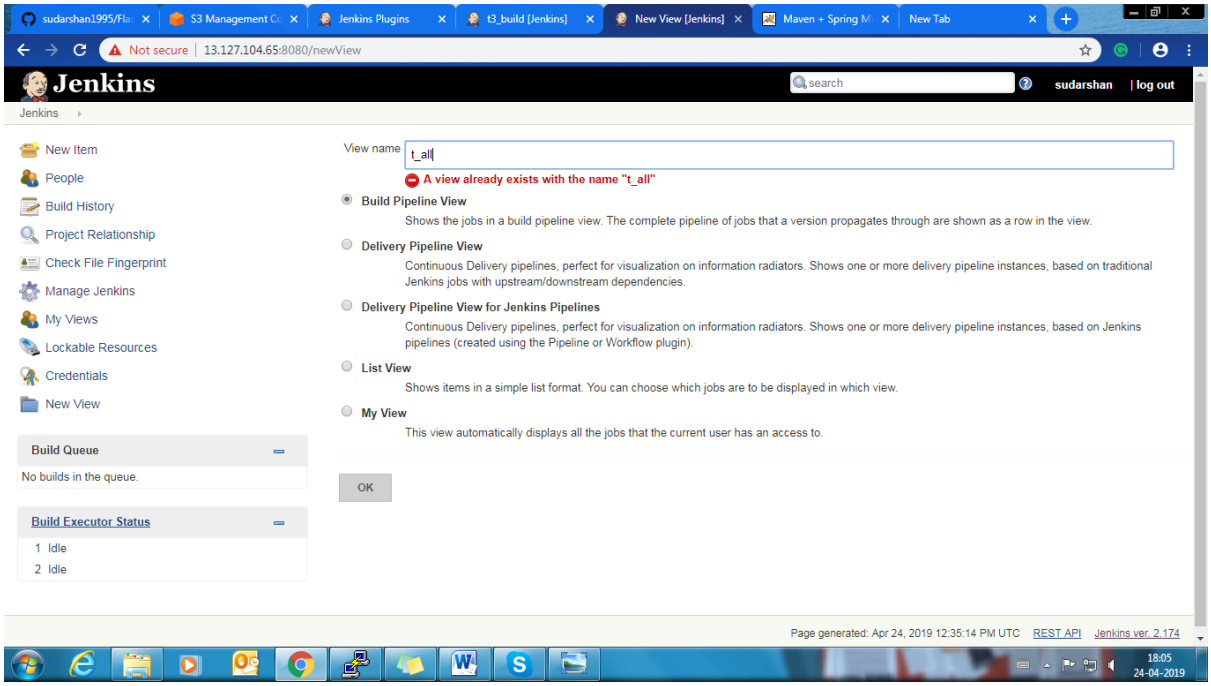
- We can check by build each project separately.
- After the successfully building of all we can build first t1_build so that other two project t2 and t3 will build automatically.
- We need two plugins for monitoring and management of this pipeline.

Install Plugin: - Build pipeline and Delivery pipeline

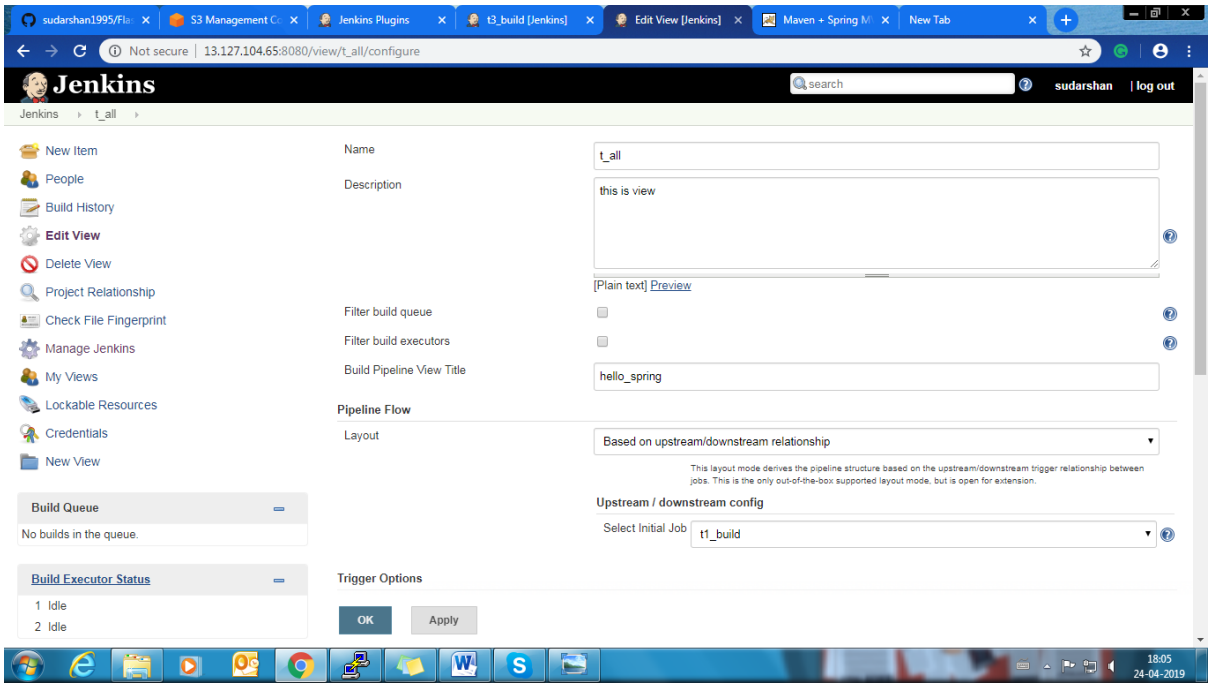
Step 13: - Now click on + sign to create build pipeline and delivery pipeline.



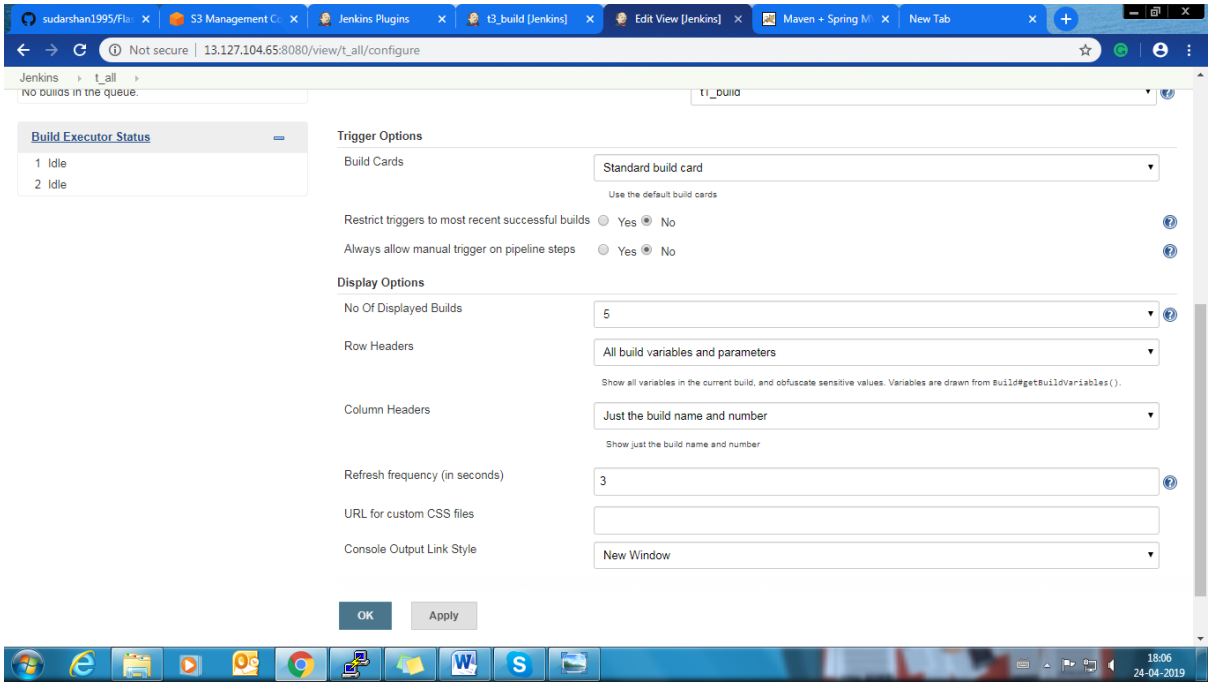
Step 14: - Click on **Build Pipeline view**



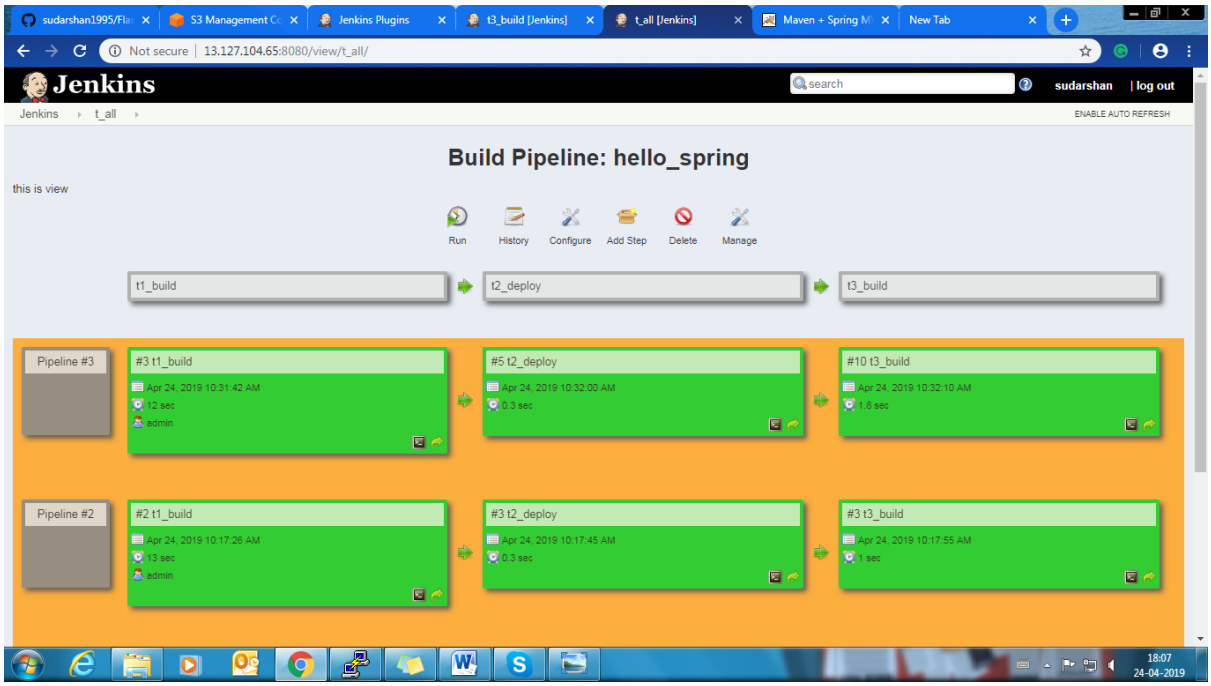
Step 14: - Give parameter



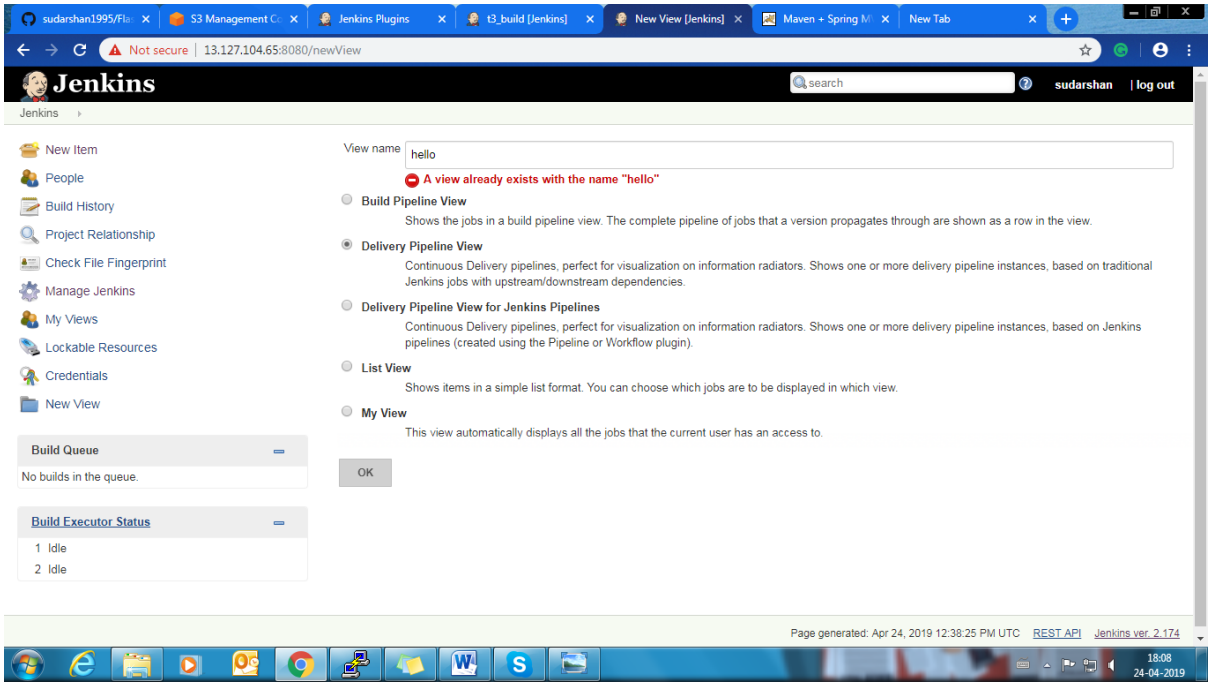
Step 14: - Click on apply and ok and click on build.



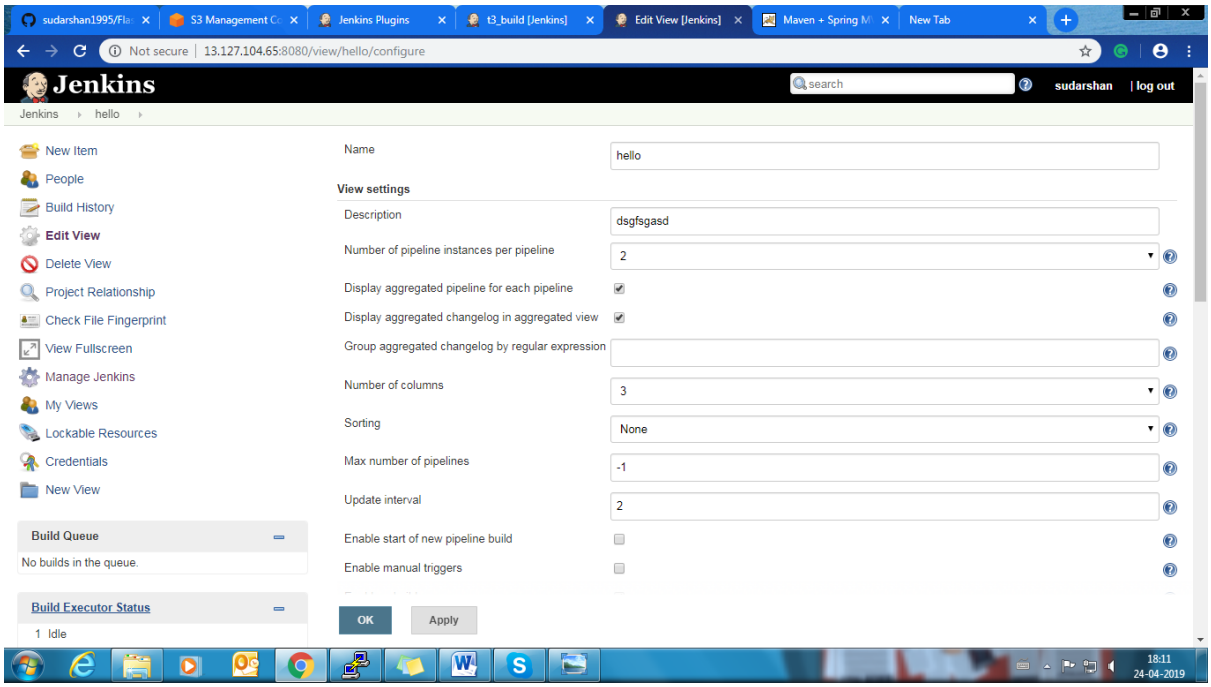
Step 15: - we can build directly and monitor from here.



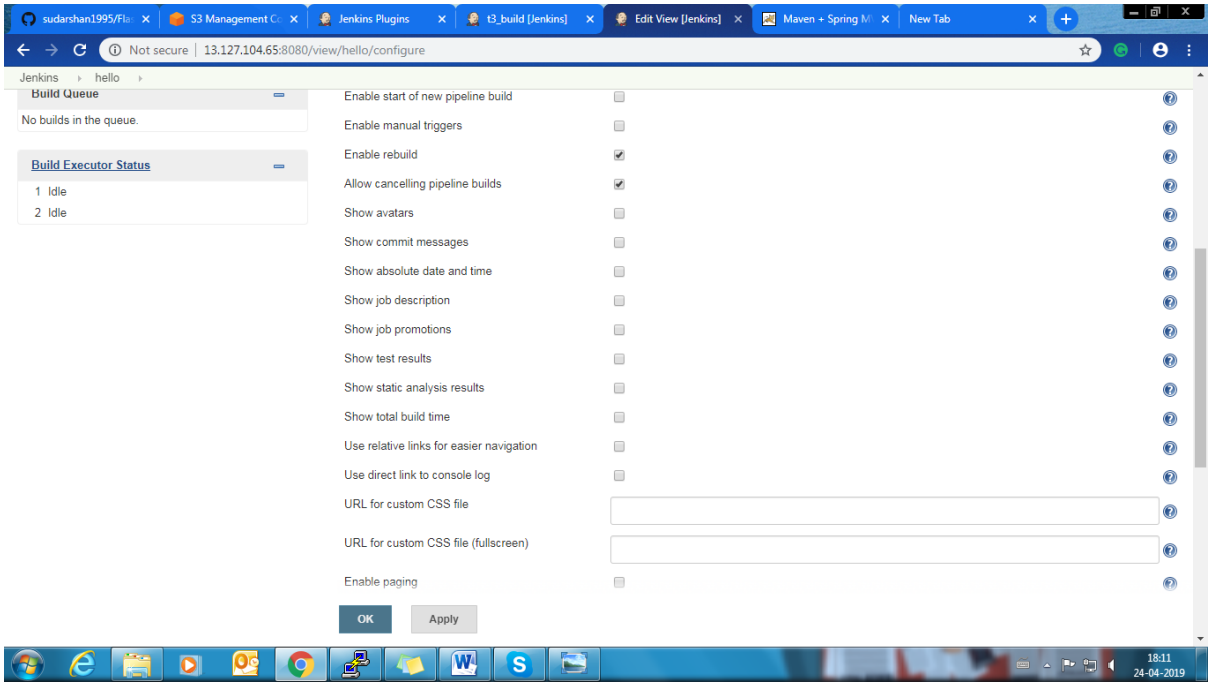
Step 16: - Now create delivery pipeline



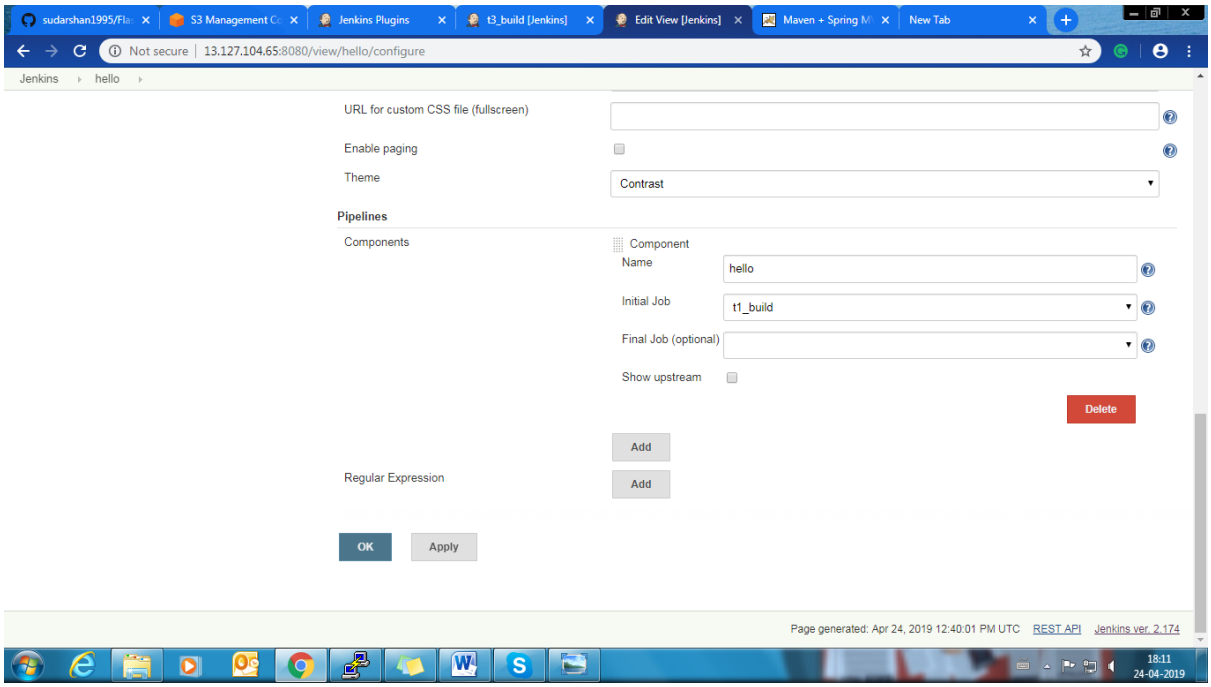
Step 17: - Give parameter



Step 18: - Select option according to your requirement



Step 19: - click ok you can add multiple option by using **edit view option**



If you want to edit then click on left side **edit view option**

You can add multiple options for abort rebuild and aggregate and many more

Task: - Configuration of email notification for builds in Jenkins.

Email is Jenkins's more fundamental notification technique—when a build fails, it will send an email message to the developer who committed the changes, and optionally to other team members as well. So Jenkins needs to know about your email server

For email notification we have to Plugin

- Mailer Plugin
- Email Extension Plugin

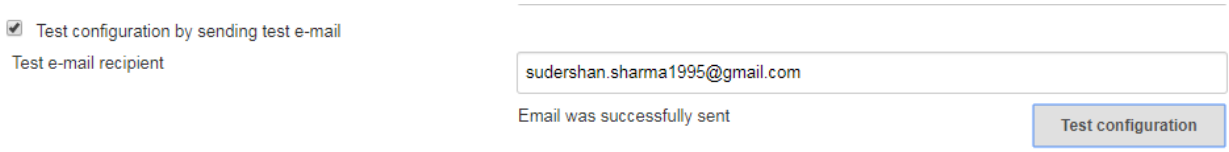
Go to **manage Jenkins** → configuration system → E-mail notification

The System Admin email address is the address from which the notification messages are sent. You can also use this field to check the email setup—if you click on the Test configuration button, Jenkins will send a test email to this address.

Jenkins also provides for more sophisticated email configuration, using more advanced features such as SMTP authentication and SSL. If this is your case, click on the advanced button to configure these options.

For example, many organizations use Google Apps for their email services. You can configure Jenkins to work with the Gmail service. All you need to do in this case is to use the Gmail SMTP server, and provide your Gmail username and password in the SMTP Authentication (you also need to use SSL and the nonstandard port of 465).

Test configuration



Troubleshooting: -

Allow less secure app to use Gmail: - https://www.google.com/settings/security/lesssecureapps

Give permission for access:-https://accounts.google.com/b/0/DisplayUnlockCaptcha

If two step verification is on then less secure app option is not there

Now we use email notification in building job

Configuration of mail is done successfully and after failed build it is generated mail.

Some basics of GitLab which are useful in project

Import your project from GitHub to GitLab

Using the importer, you can import your GitHub repositories to GitLab.com or to your self-hosted GitLab instance.

References to pull requests and issues are preserved (GitLab.com & 8.7+), and each imported repository maintains visibility level unless that visibility level is restricted, in which case it defaults to the default project visibility.

How it works

When issues and pull requests are being imported, the importer attempts to find their GitHub authors and assignees in the database of the GitLab instance (note that pull requests are called "merge requests" in GitLab).

For this association to succeed, prior to the import, each GitHub author and assignee in the repository must have either previously logged in to a GitLab account using the GitHub icon or have a GitHub account with a public email address that matches their GitLab account's email address.

If a user referenced in the project is not found in GitLab's database, the project creator (typically the user that initiated the import process) is set as the author/assignee, but a note on the issue mentioning the original GitHub author is added.

The importer creates any new namespaces (groups) if they do not exist, or, if the namespace is taken, the repository is imported under the namespace of the user who initiated the import process. The namespace/repository name can also be edited, with the proper permissions.

The importer will also import branches on forks of projects related to open pull requests. These branches will be imported with a naming scheme similar **to GH-SHA-username/pull-request-number/fork-name/branch**. This may lead to a discrepancy in branches compared to those of the GitHub repository.

For additional technical details, you can refer to the GitHub Importer developer documentation.

Import your GitHub repository into GitLab

Using the GitHub integration

Before you begin, ensure that any GitHub users who you want to map to GitLab users have either:

- A GitLab account that has logged in using the GitHub icon - or -
- A GitLab account with an email address that matches the public email address of the GitHub user

User-matching attempts occur in that order, and if a user is not identified either way, the activity is associated with the user account that is performing the import.

Note: If you are using a self-hosted GitLab instance, this process requires that you have configured the GitHub integration.

1. From the top navigation bar, click **+** and select **New project**.
2. Select the **Import project** tab and then select **GitHub**.
3. Select the first button to **List your GitHub repositories**. You are redirected to a page on github.com to authorize the GitLab application.
4. Click **Authorize gitlabhq**. You are redirected back to GitLab's Import page and all of your GitHub repositories are listed.
5. Continue on to selecting which repositories to import.

Using a GitHub token

Note: For a proper author/assignee mapping for issues and pull requests, the GitHub integration method (above) should be used instead of the personal access token. If you are using GitLab.com or a self-hosted GitLab instance with the GitHub integration enabled, that should be the preferred method to import your repositories. Read more in the How it works section.

If you are not using the GitHub integration, you can still perform an authorization with GitHub to grant GitLab access your repositories:

1. Go to https://github.com/settings/tokens/new
2. Enter a token description.
3. Select the repo scope.
4. Click **Generate token**.
5. Copy the token hash.
6. Go back to GitLab and provide the token to the GitHub importer.
7. Hit the **List Your GitHub Repositories** button and wait while GitLab reads your repositories' information. Once done, you'll be taken to the importer page to select the repositories to import.

Selecting which repositories to import

After you have authorized access to your GitHub repositories, you are redirected to the GitHub importer page and your GitHub repositories are listed.

1. By default, the proposed repository namespaces match the names as they exist in GitHub, but based on your permissions, you can choose to edit these names before you proceed to import any of them.
2. Select the **Import** button next to any number of repositories, or select **Import all repositories**.
3. The **Status** column shows the import status of each repository. You can choose to leave the page open and it will update in real time or you can return to it later.
4. Once a repository has been imported, click its GitLab path to open its GitLab URL.

Mirroring and pipeline status sharing

Depending your GitLab tier, project mirroring can be set up to keep your imported project in sync with its GitHub copy.

Additionally, you can configure GitLab to send pipeline status updates back GitHub with the GitHub Project Integration.

If you import your project using CI/CD for external repo, then both of the above are automatically configured.

Improving the speed of imports on self-hosted instances

Note: Admin access to the GitLab server is required.

For large projects it may take a while to import all data. To reduce the time necessary, you can increase the number of Sidekiq workers that process the following queues:

- github_importer
- github_importer_advance_stage

For an optimal experience, it's recommended having at least 4 Sidekiq processes (each running a number of threads equal to the number of CPU cores) that only process these queues. It's also recommended that these processes run on separate servers. For 4 servers with 8 cores this means you can import up to 32 objects (e.g., issues) in parallel.

Reducing the time spent in cloning a repository can be done by increasing network throughput, CPU capacity, and disk performance (e.g., by using high performance SSDs) of the disks that store the Git repositories (for your GitLab instance). Increasing the number of Sidekiq workers will not reduce the time spent cloning repositories.

How to remove project

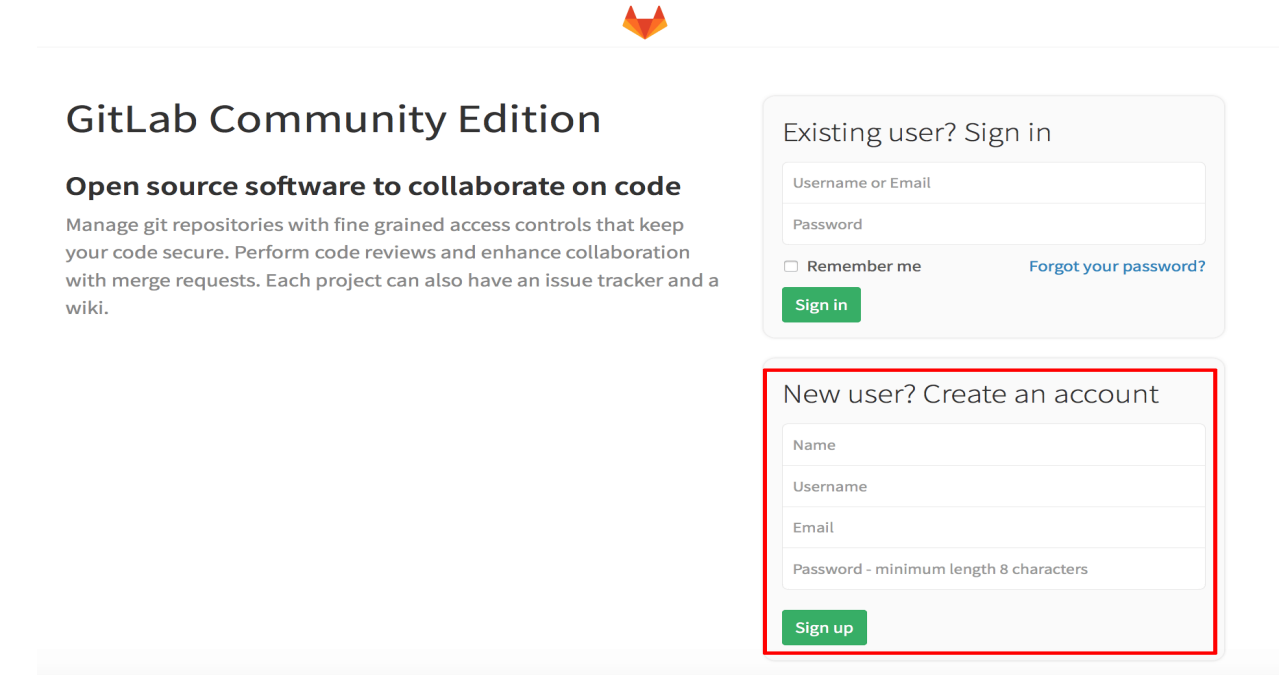
1. Go to the **project page**
2. Select "**Settings**"
3. If you have enough rights at the bottom of the page then will be a button for "**Advanced settings**" (i.e. project settings that may result in data loss) or "**Remove project**" (in newer GitLab versions)
4. Push this button and follow the instructions

Task: - **Create a GitLab project and Configure Jenkins with GitLab.**

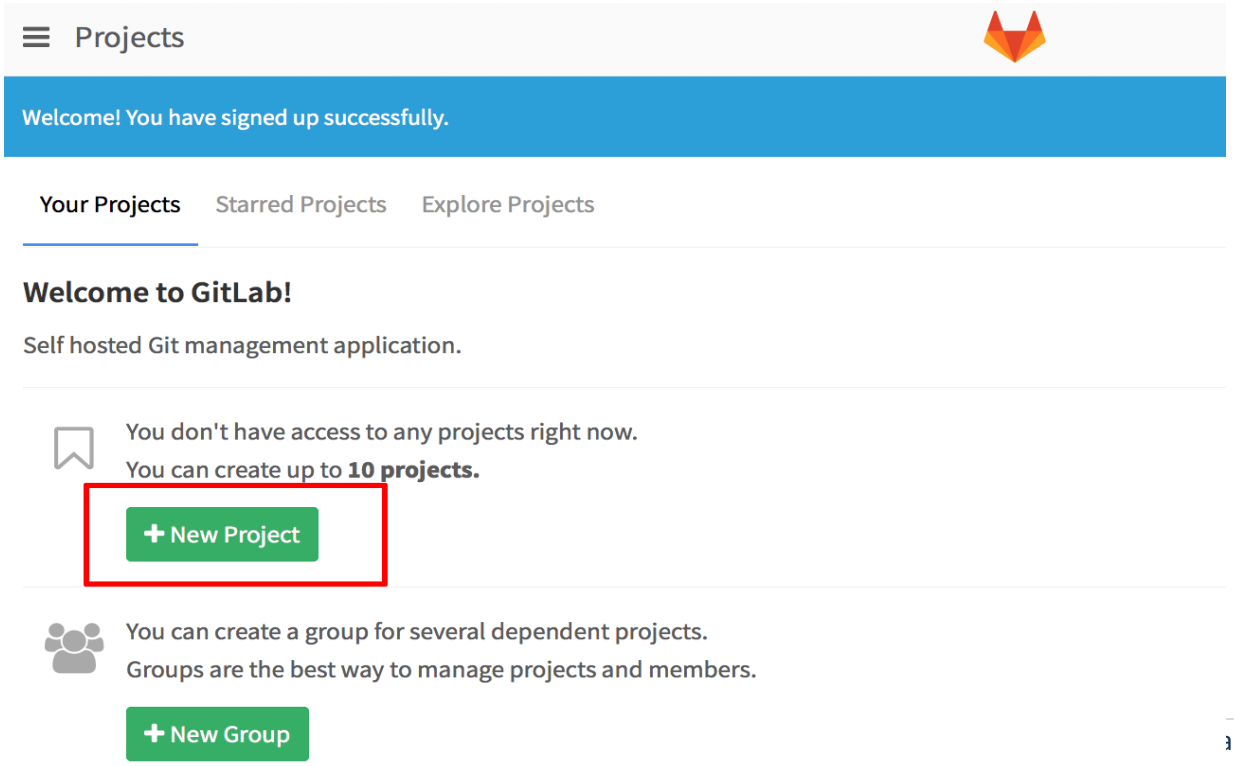
- ## Create a GitLab Project

The first step is to create a Git repository for your project source code using GitLab. This is where you and your team will push code changes, and this will also be the source repository for Jenkins' automated builds. Follow these steps:

Step 1: - Create a new user account on GitLab and log in.

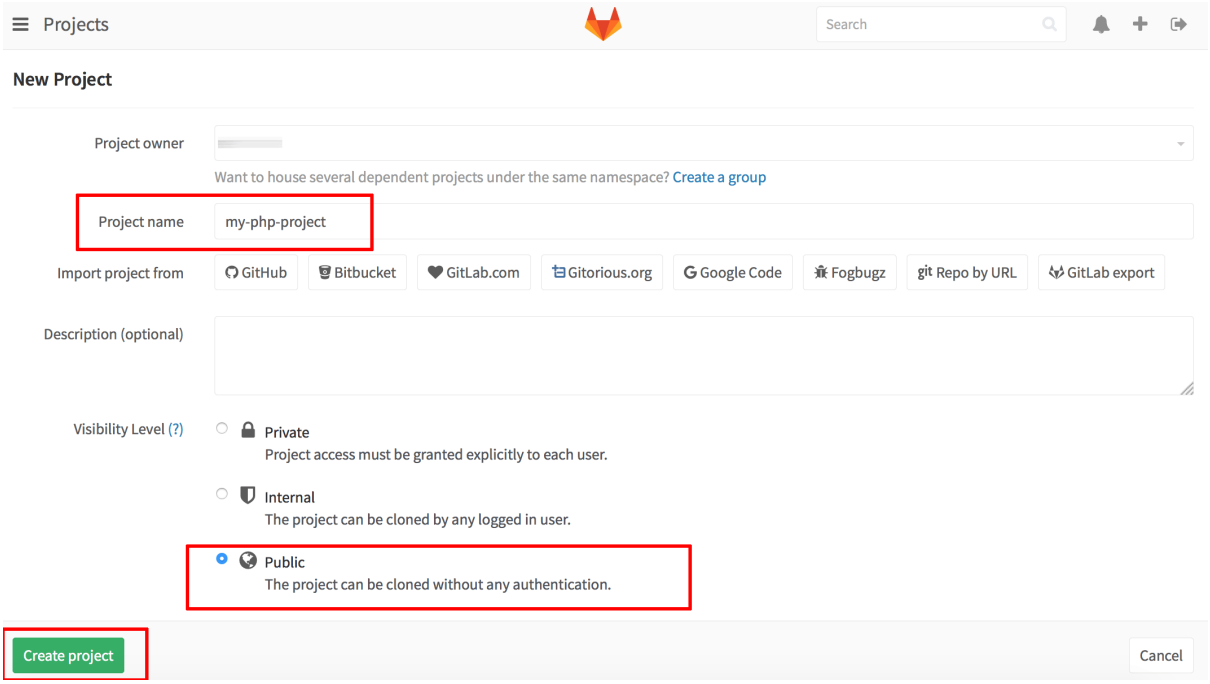


Step 2: - From the "Your Projects" tab, click the "New Project" button.



Step 3: - On the “New Project” page, enter a name for the project and set the visibility level. The project in this example is set to “Public”, but you can also choose to make your project “Private” or “Internal”. Click “Create Project” once done.

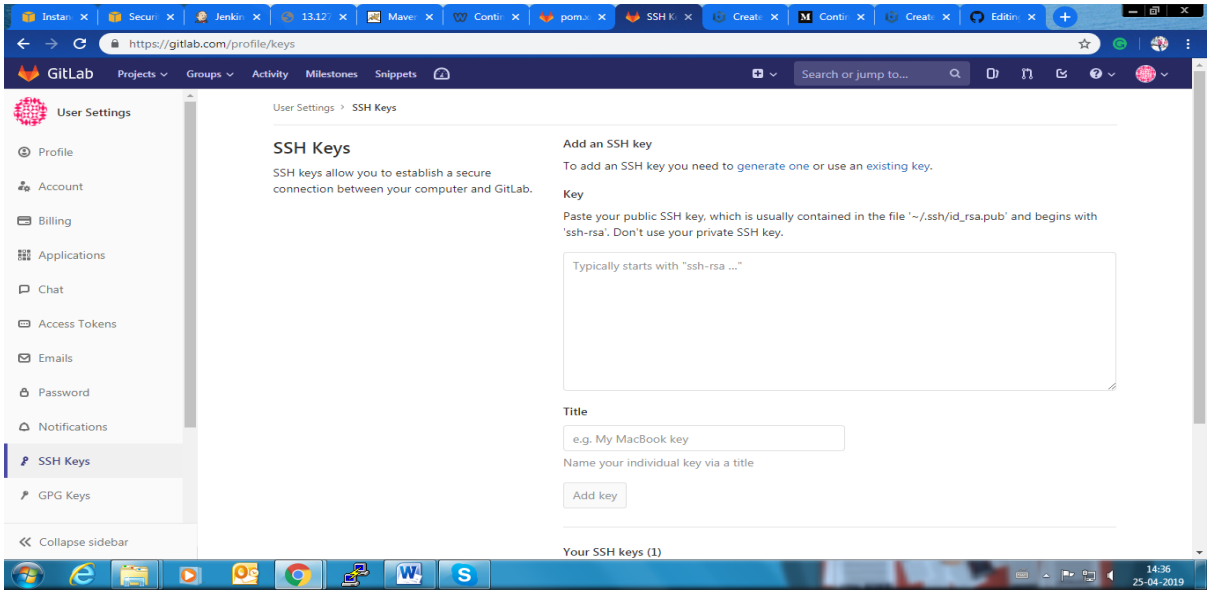
GitLab will initialize an empty Git repository for the project, as shown below. Note the SSH URL to the repository, as you will need it in the next step.



- ## Configure SSH Access to the GitLab Project

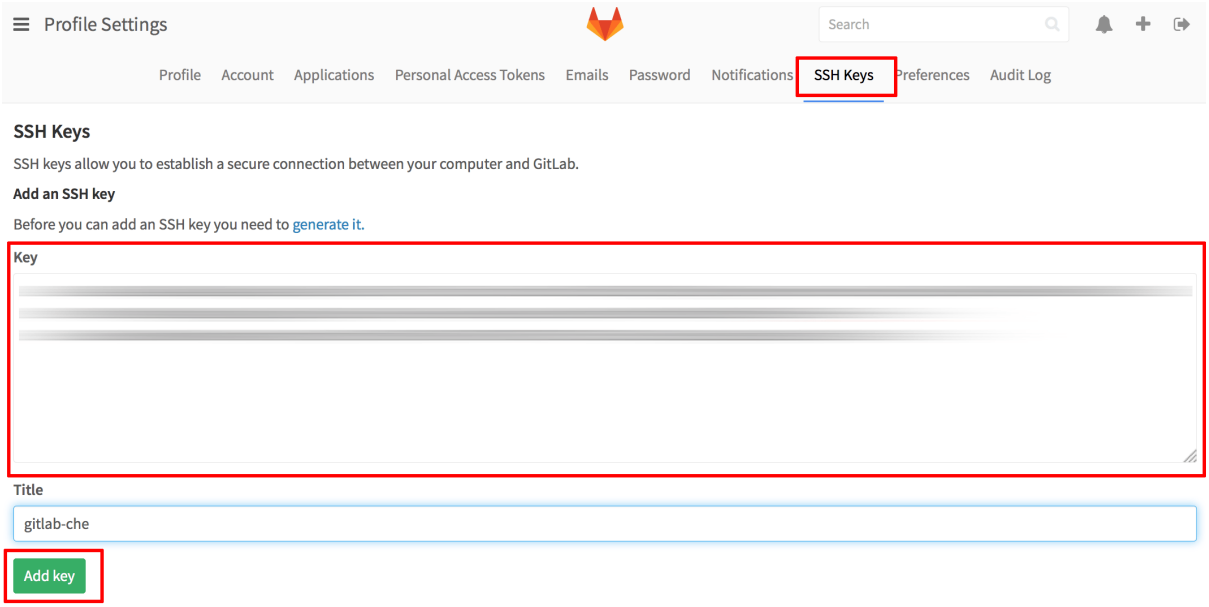
Both Jenkins and your Git client will need access to the project repository, so this is a good time to configure that access. Follow this step:

Step 1: - From the GitLab main menu (accessed via the hamburger icon in the top left corner), select the “Profile Settings” option.

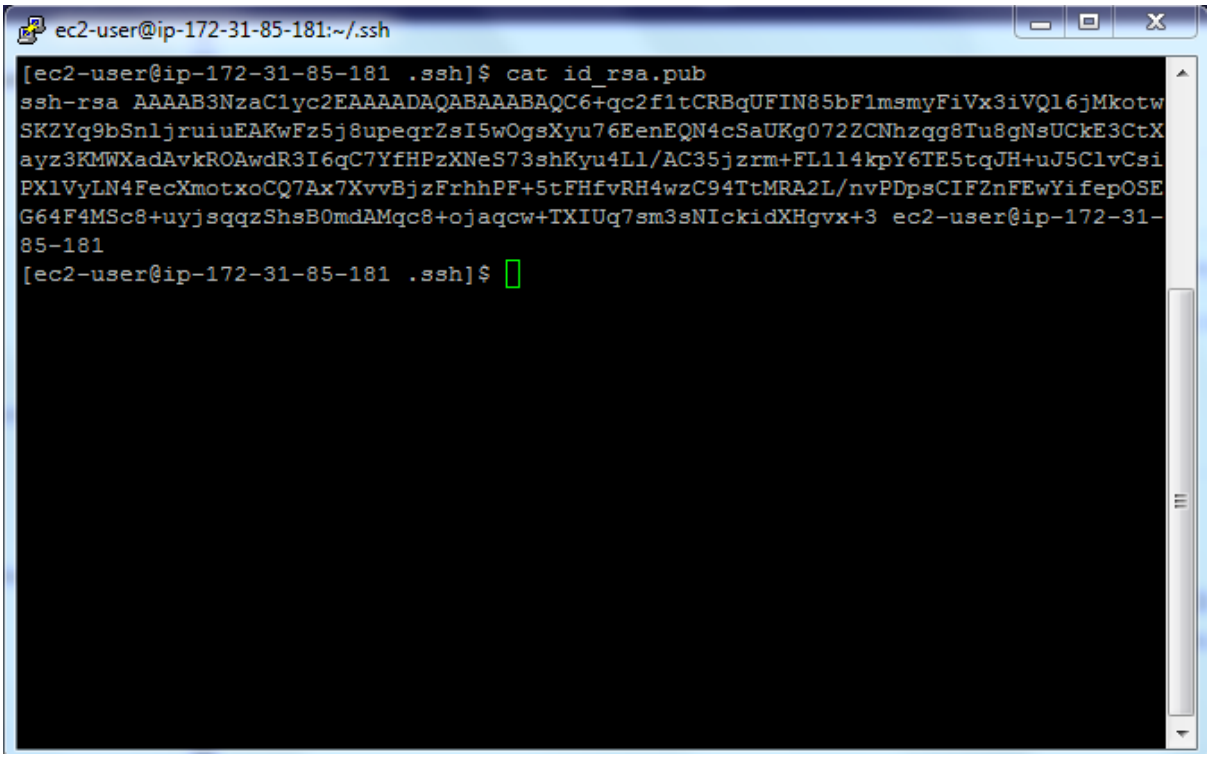


Select the “SSH Keys” tab and enter the public key content (as a single line) for use between GitLab and your Git client. Add a descriptive title and click the “Add key” button to add it to your profile.

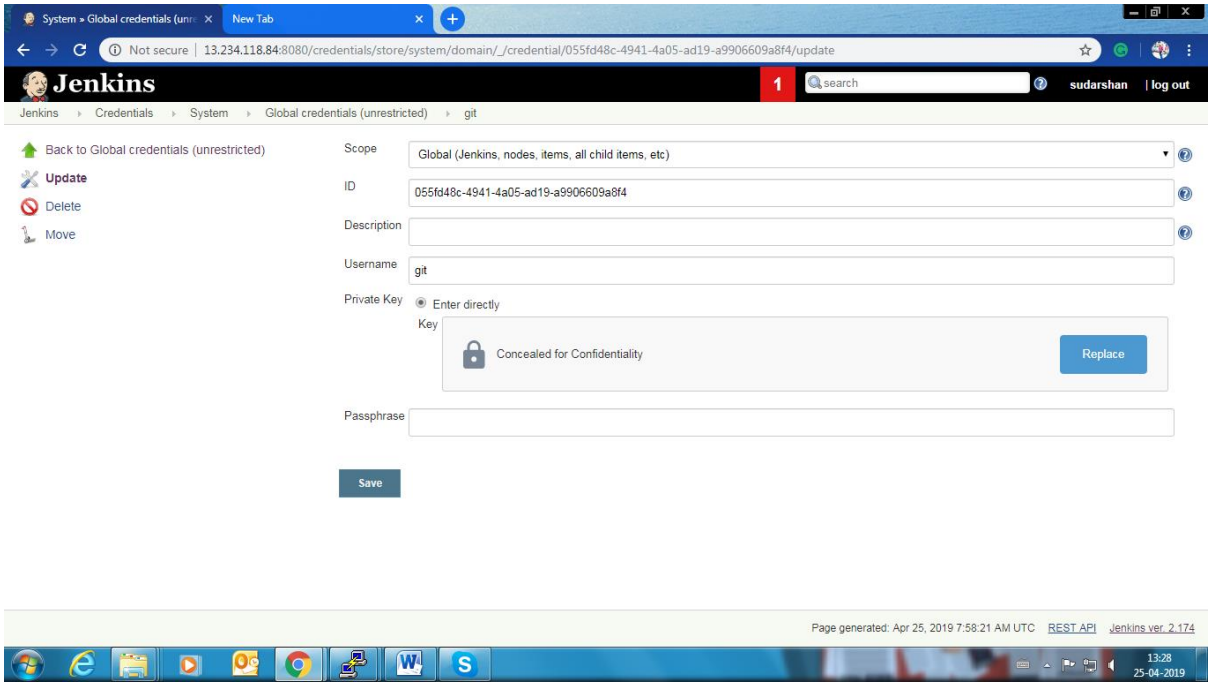
Generate private key with ssh-keygen on instance and put it here.



TIP: Public key files typically have a .pub suffix.



Step 2: - Go to manager **jenkins→ global credentials→** and add credential **and private key**

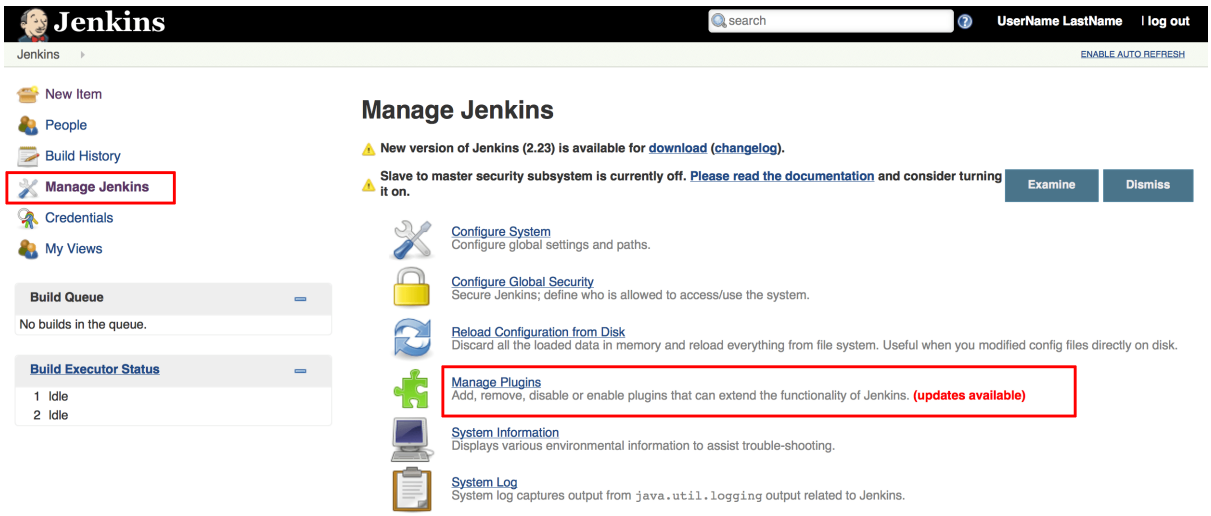


- **Configure Jenkins**

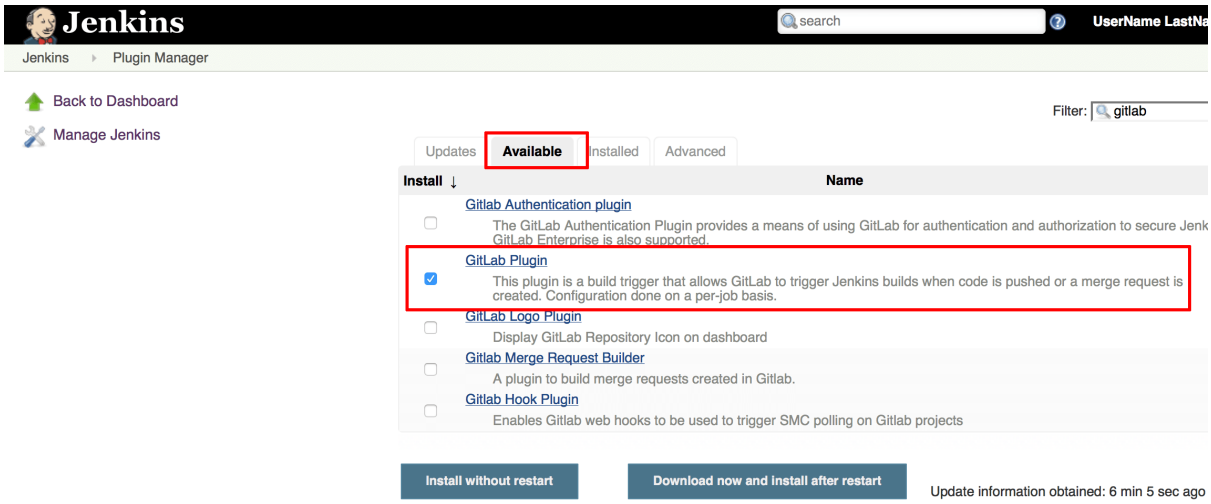
The next step is to connect Jenkins with GitLab, such that a new push to the GitLab repository automatically triggers a Jenkins build. This requires you to add the GitLab plugin to Jenkins and create SSH credentials so that Jenkins can access the GitLab project.

Step 1: - Log in to Jenkins and click the “Manage Jenkins” link.

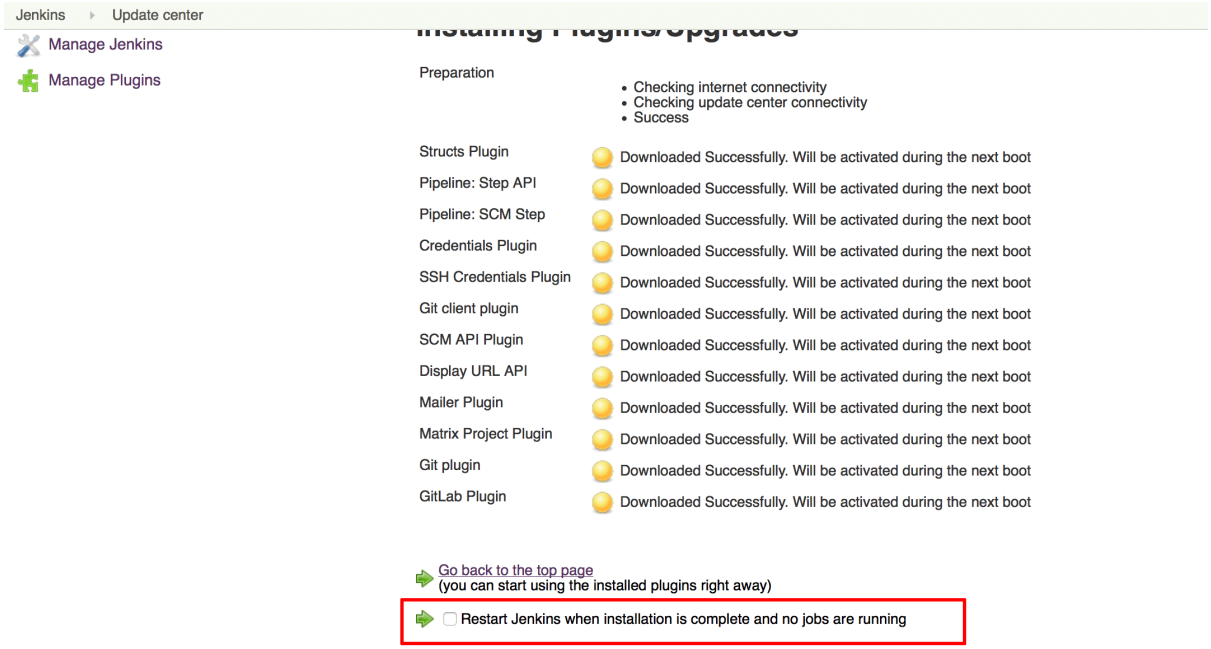
Step 2: - Select the “Manage Plugins” option.



Step 3: - From the “Available” tab, find and select the GitLab plugin, GitLab hook. Click the “Download now and install after restart” button to download it.

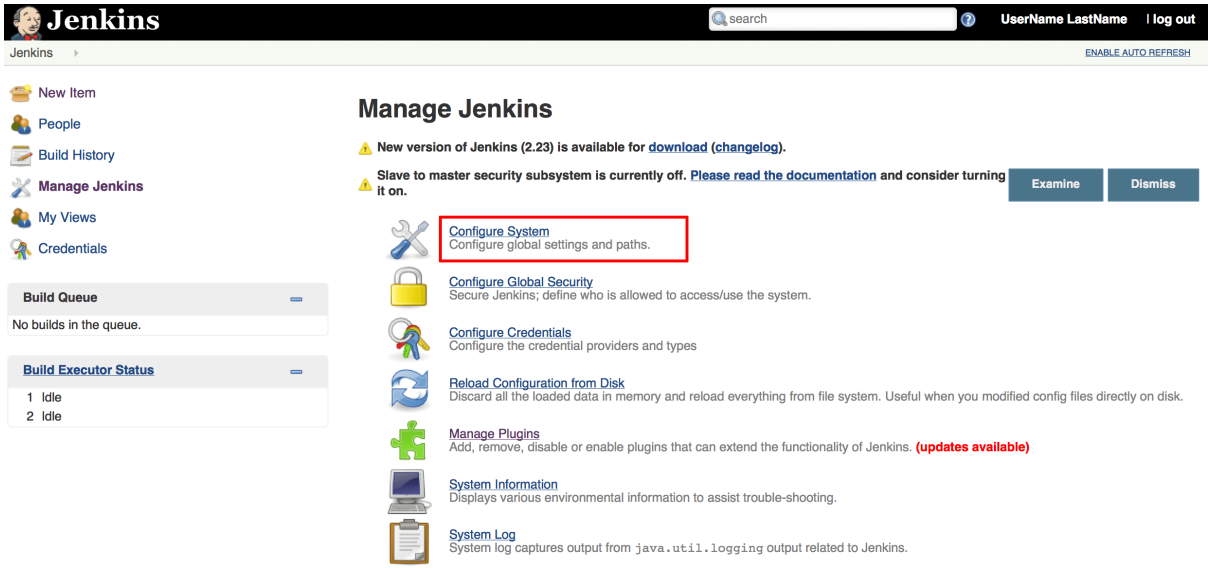


Step 4: - Once the plugin has been downloaded, click the “Restart Jenkins” checkbox and wait for Jenkins to restart.



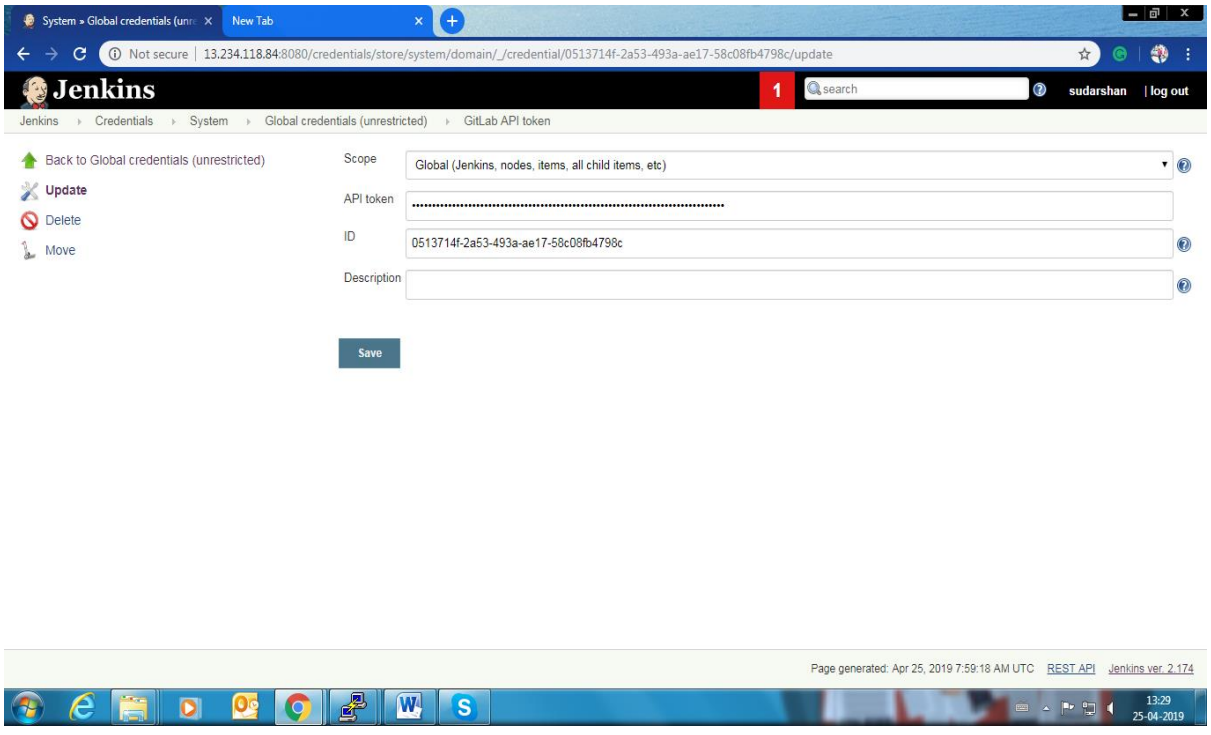
Step 5: - If you want Jenkins to merge successful builds back to the GitLab repository, you also need to configure a Git user name and email address in Jenkins. To do this:

- Click the “Manage Jenkins” link.
- Select the “Configure System” option.

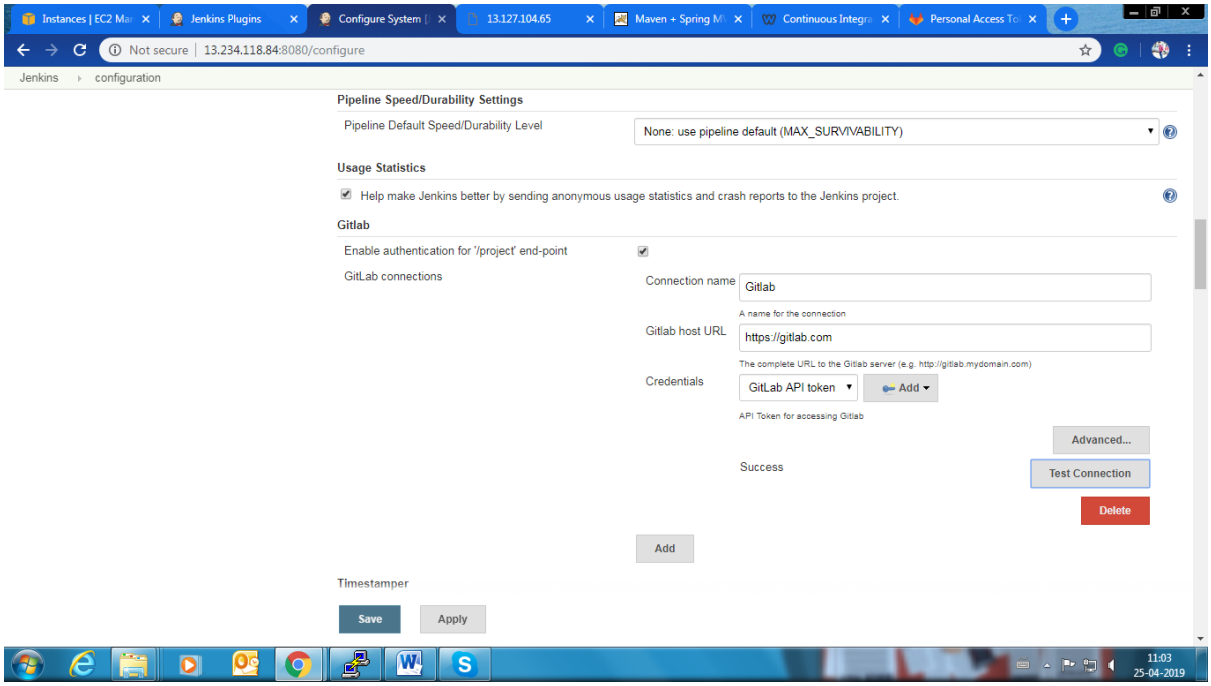


Step 6: - In the “Git plugin” section, enter a user name and email address.

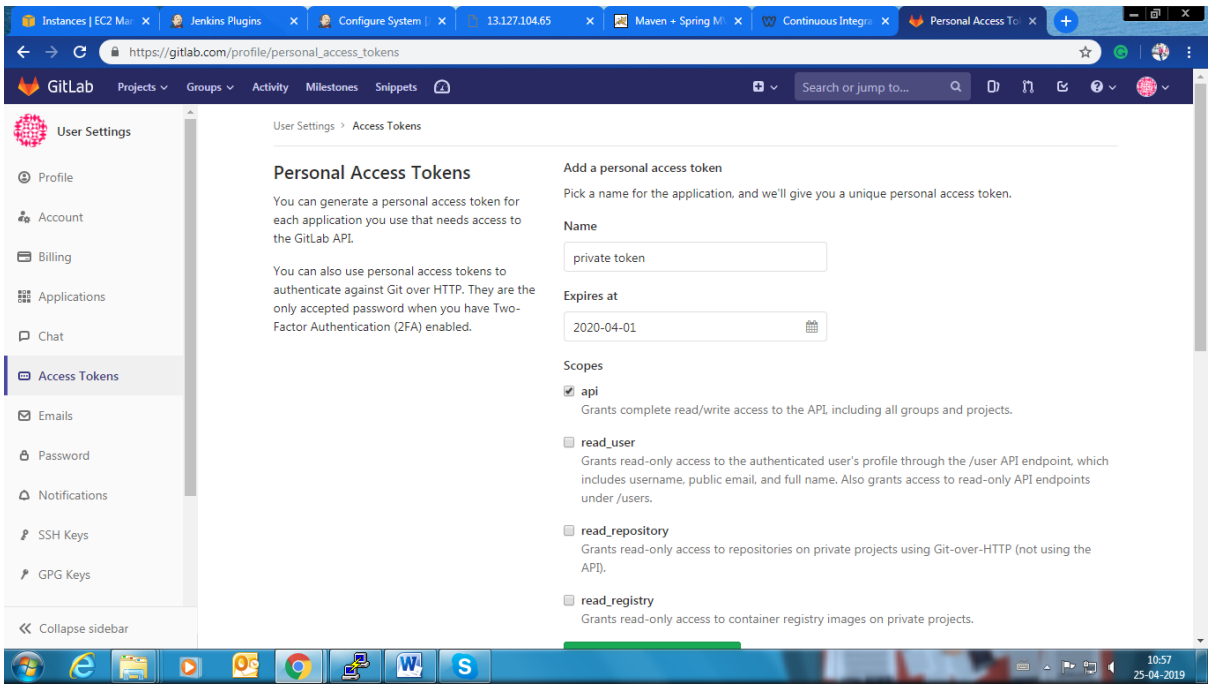
Add credential for git lab



Step 7: - Test connection with GitLab



Step 8: - Need to create a private token



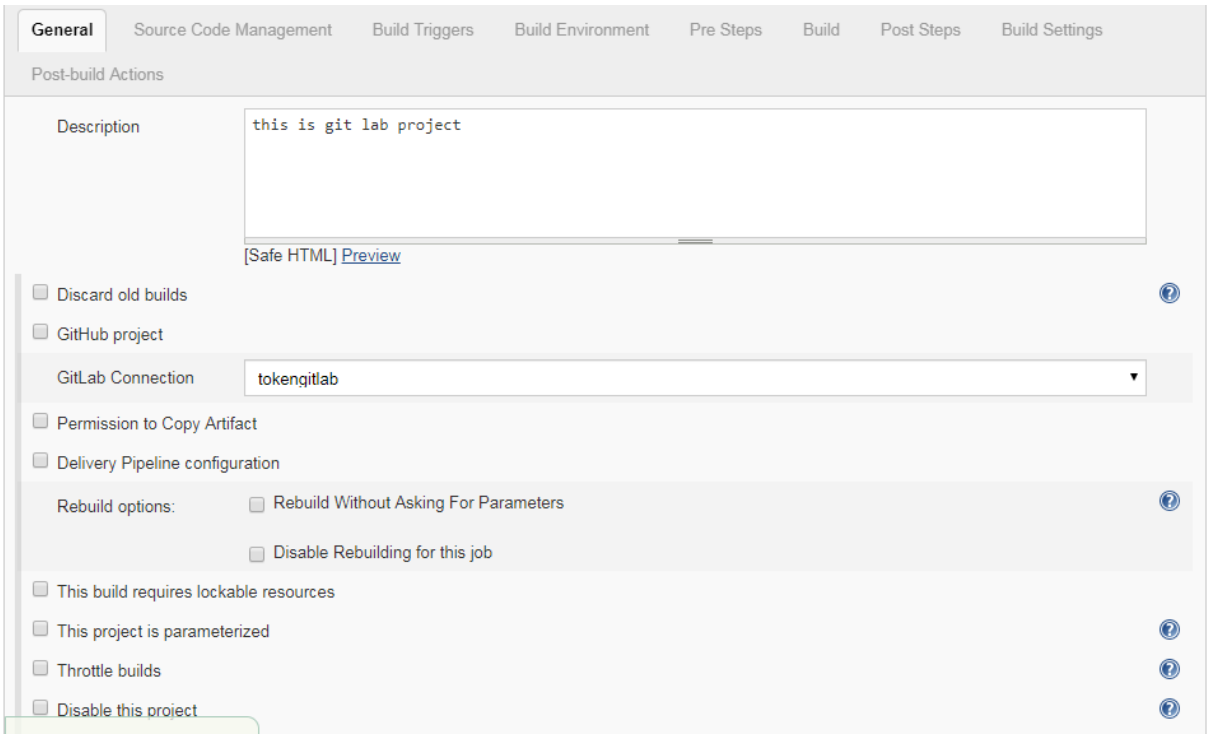
Now you are able to communicate with GitLab.

- ## Create a Jenkins Project

You're now ready to create a Jenkins project and connect it to GitLab.

Step 1: - Log in to Jenkins and click the "New Item" link to create a new job.

Step 2: - Enter a name for the job and set it to be a "Freestyle project".

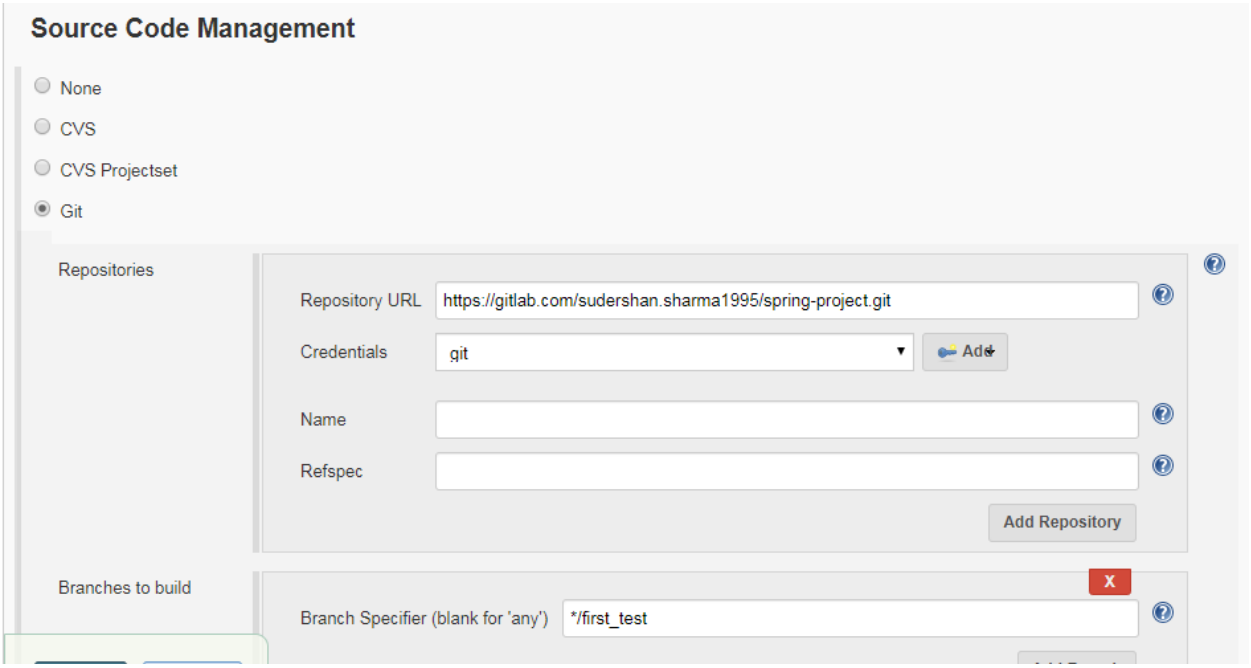


Step 3: - On the job configuration page, find the "Source Code Management" section and select "Git". Configure the GitLab connection as follows:

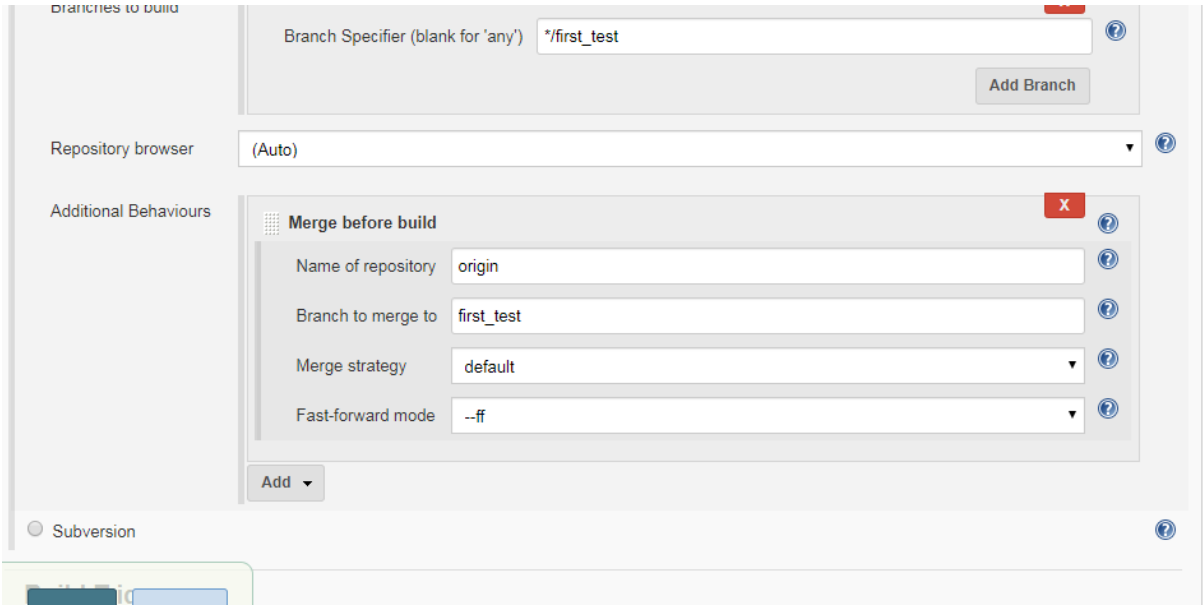
• Enter the SSH URL to the GitLab repository.

• Click the "Add" button next to the "Credentials" field and select the "Jenkins" option.

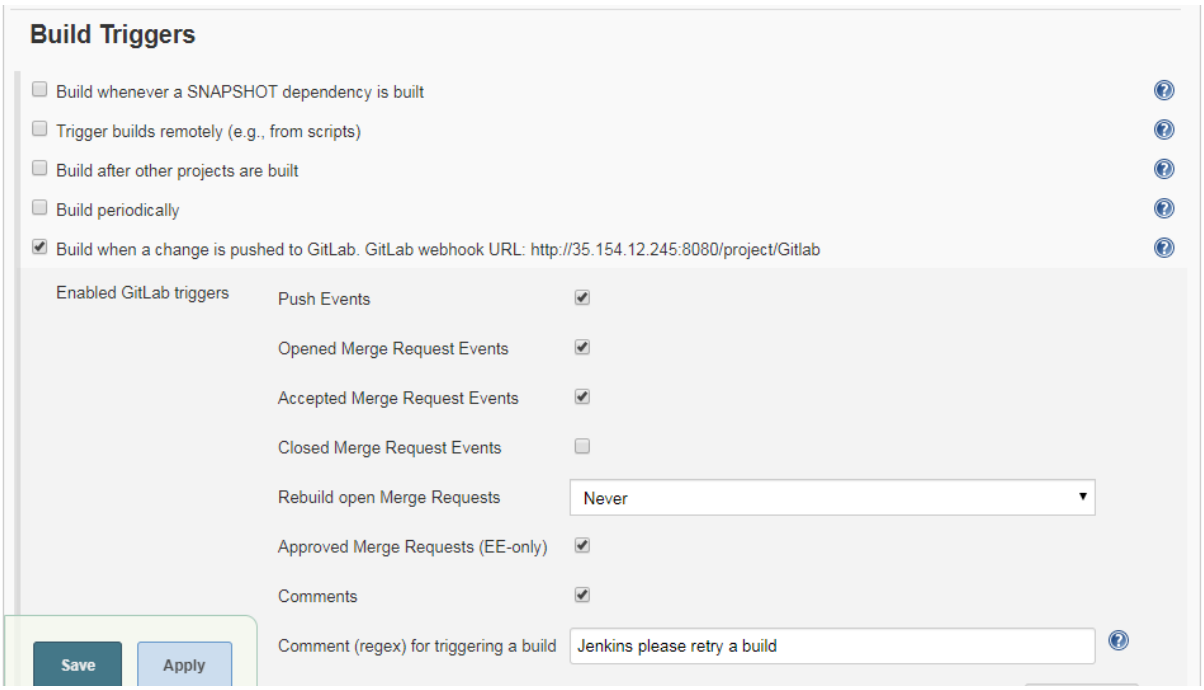
In the resulting dialog, select "SSH Username with private key" as the credential type, set the "Username" to git, and enter the content of the private key selected for use between GitLab and Jenkins (remember that you already attached the corresponding public key to your GitLab profile)

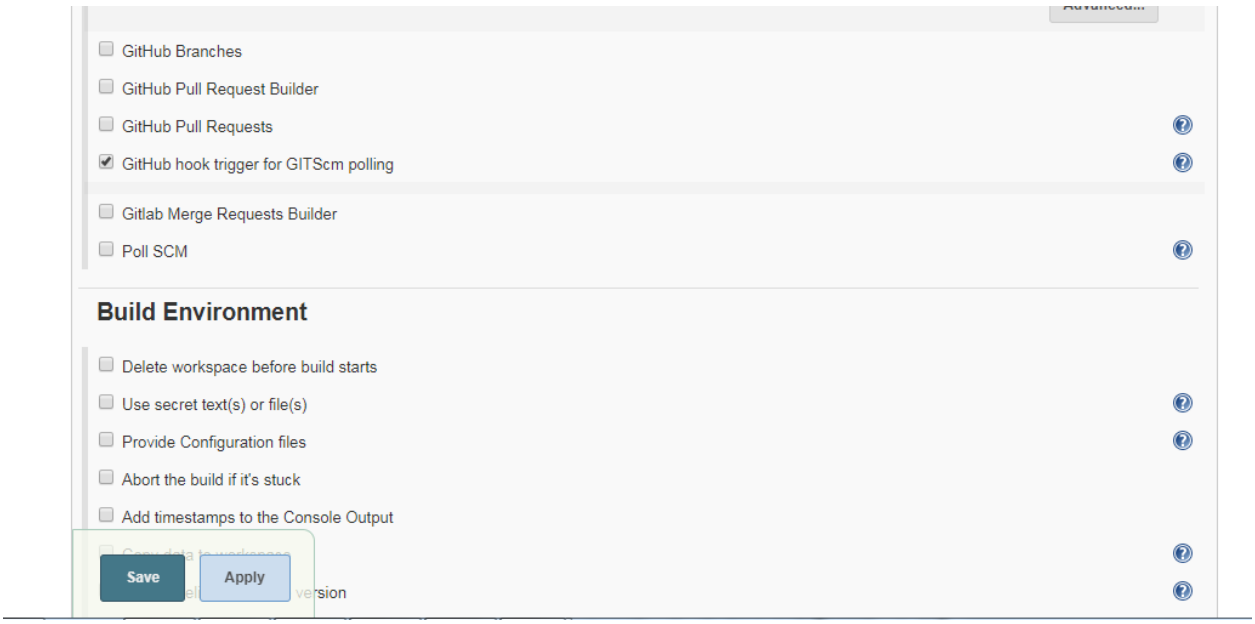


Step 4: - It is common to have Jenkins merge the new changes into the production branch before building. On the same configuration page, find the “Additional Behaviours” section and check the option to “Merge before build”. Specify the name of the repository as origin and the branch to merge to as first_test.



Step 5: - On the same configuration page, find the “Build Triggers” section and check the option to “Build when a change is pushed to GitLab”.

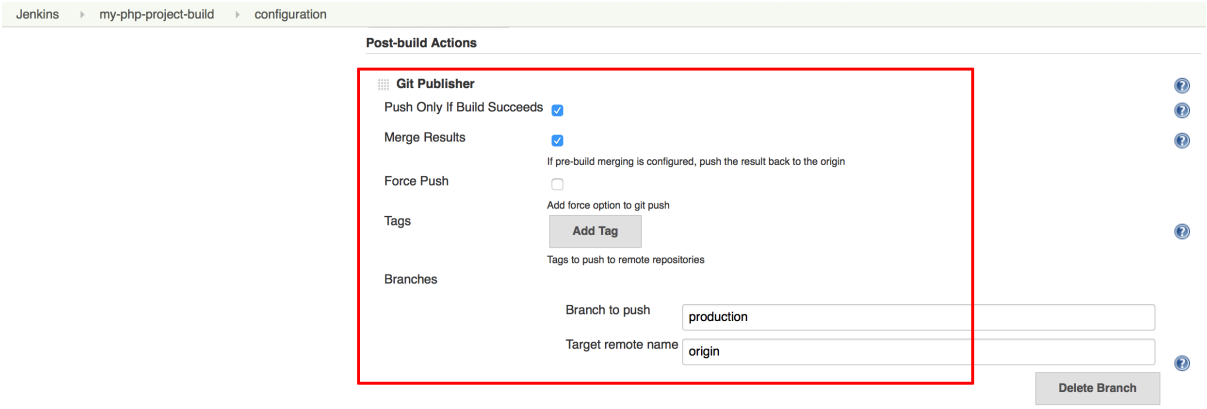




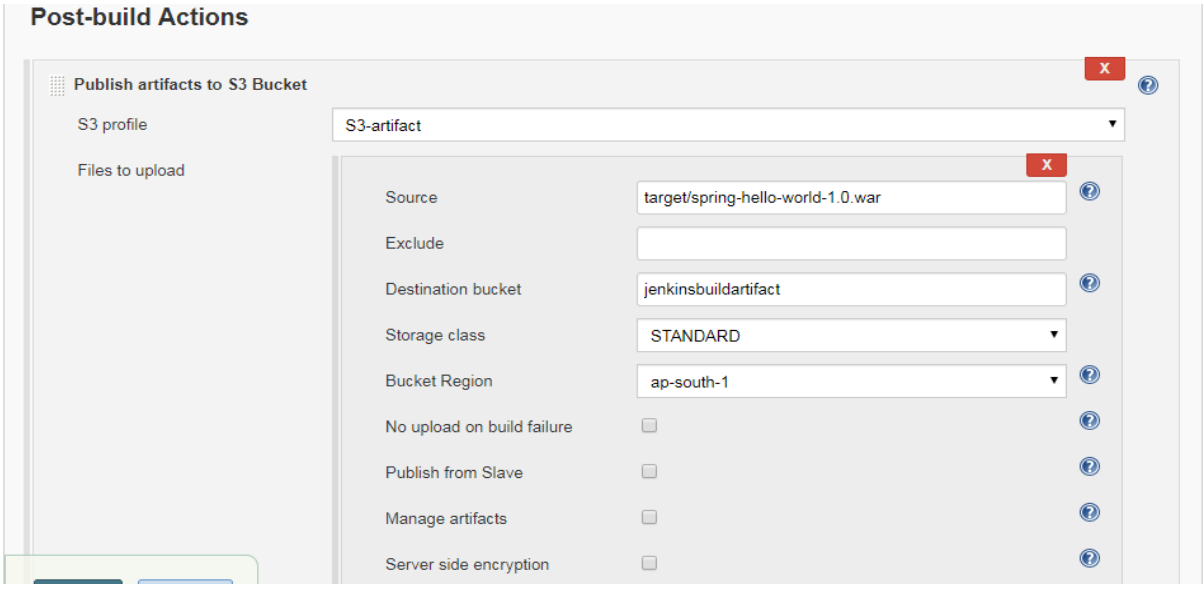
Step 6: - Set goals and option

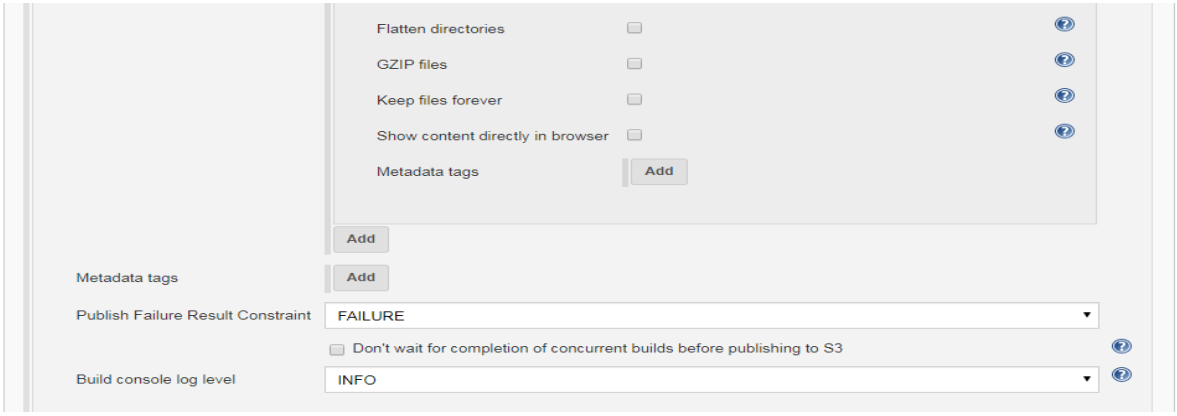
Step 7: - As part of the continuous integration process, you will also usually want Jenkins to automatically merge successful builds back into the production branch. To do this:

- Click the “Add post-build action” button.
- Select the “Git Publisher” option
- Check the “Push only if build succeeds” and “Merge Results” options.
- Specify the remote name and branch to push as origin and production respectively. You can also add custom tags if you wish.

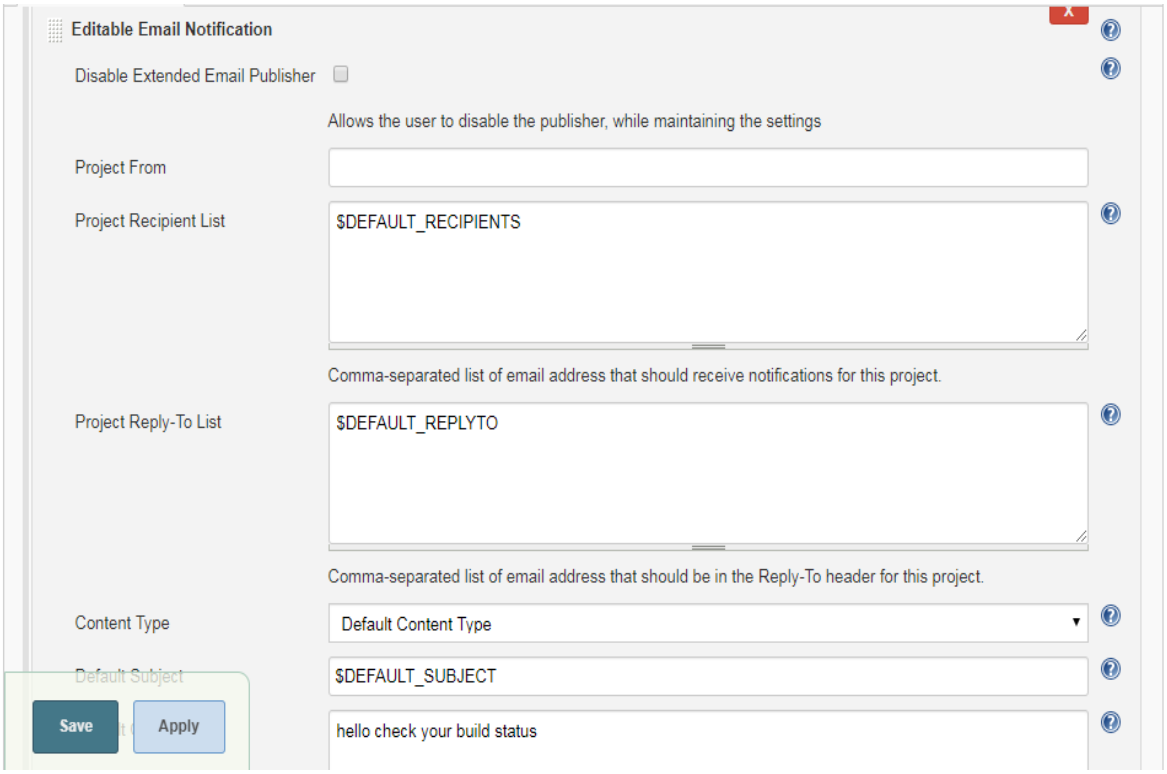


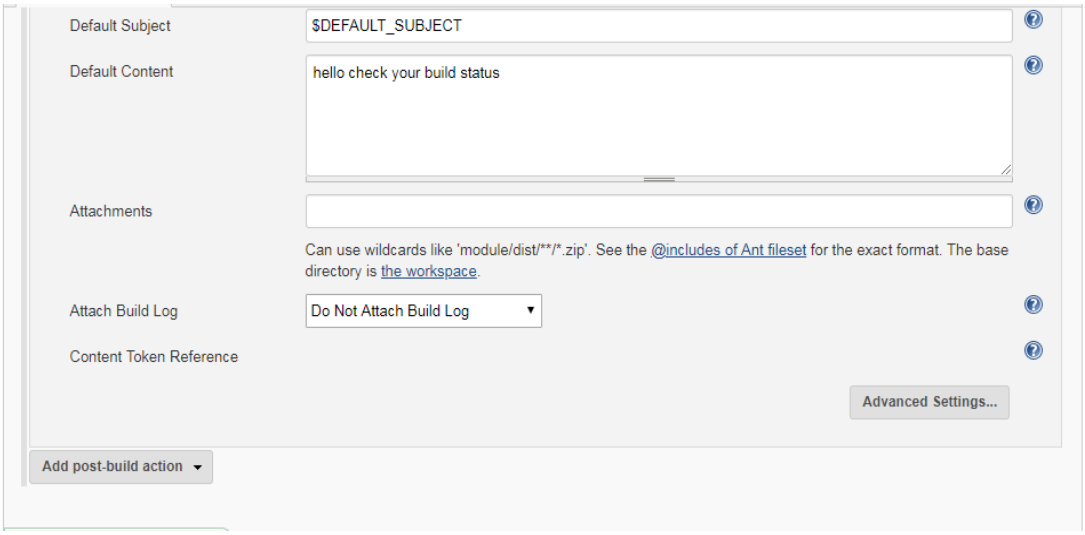
Step 8: - Select publish artifacts to s3 bucket



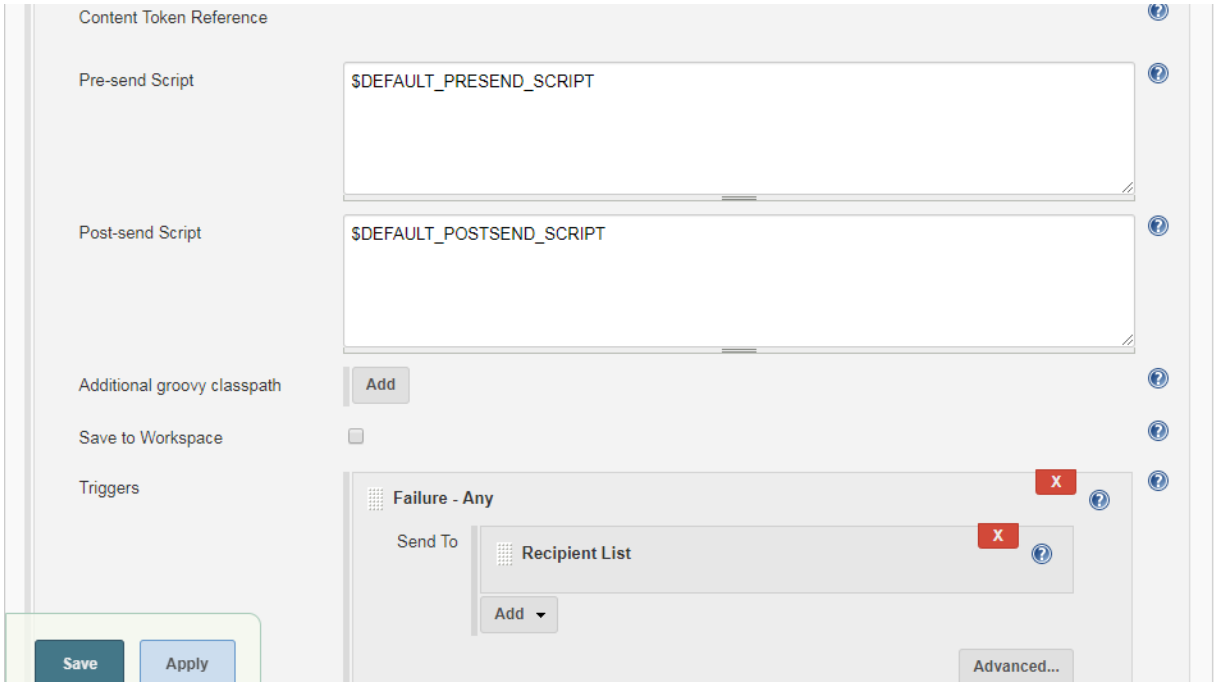


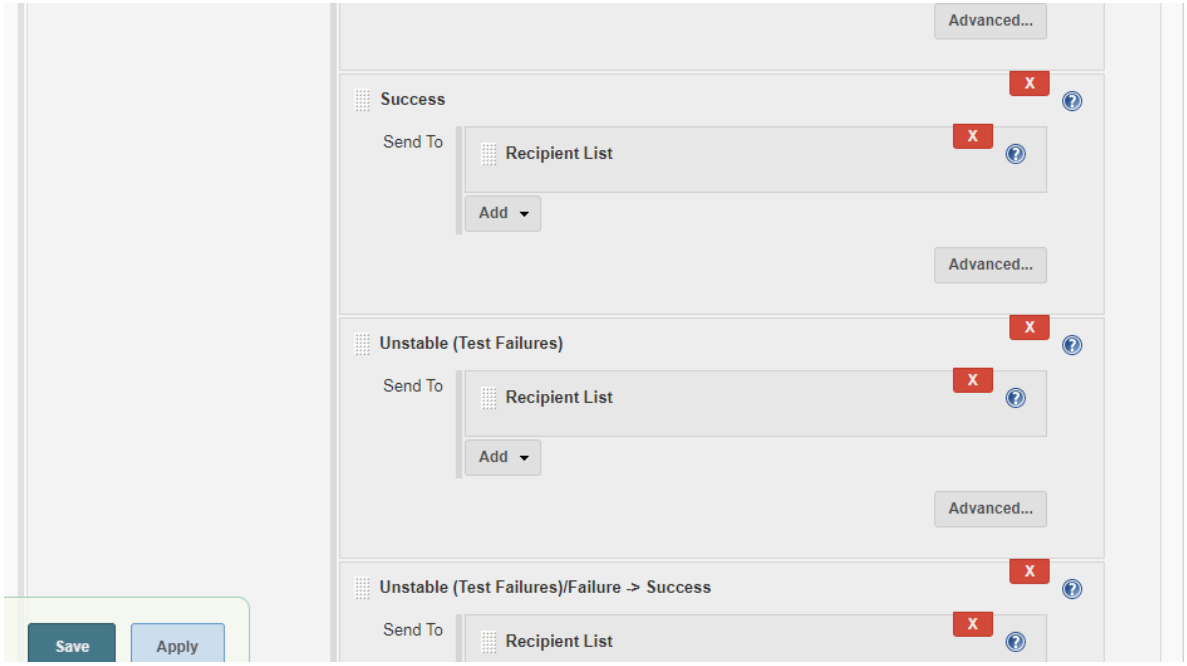
Step 9: - Configuration of email





Step 10: - Go to advance settings and set trigger for email notification





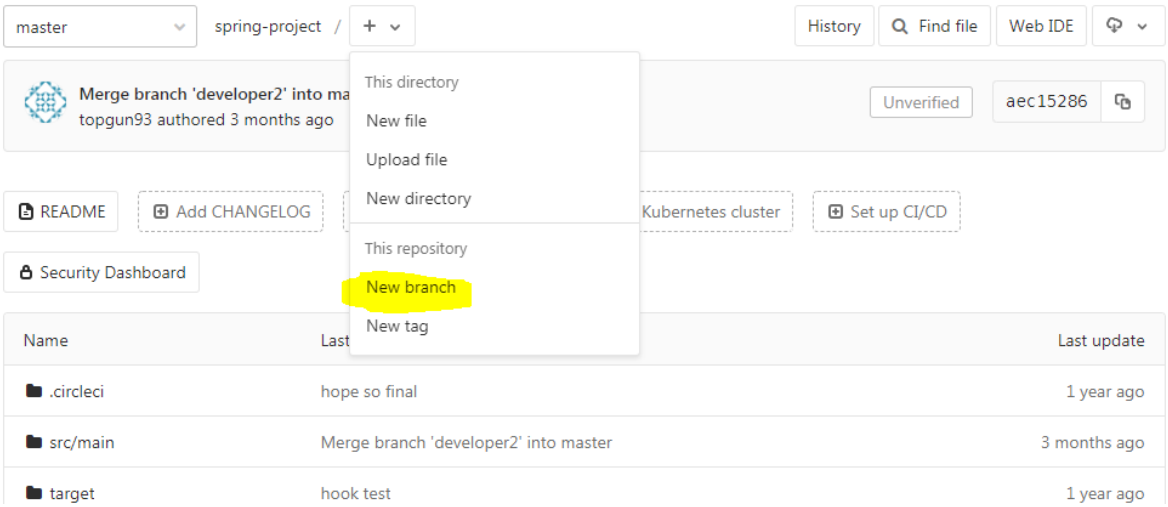
Save and build.

- **Create a GitLab Branch for Jenkins**

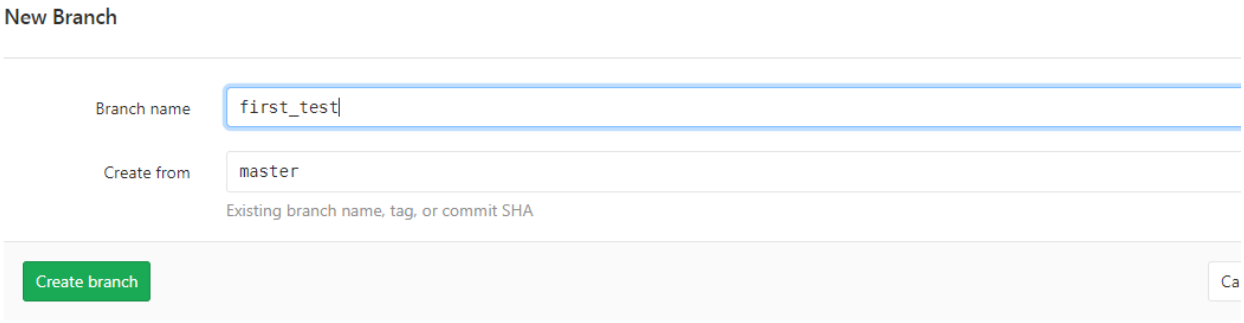
The Jenkins configuration in the previous step will merge successful builds to a first_branch in the GitLab repository. This branch doesn't exist yet so go ahead and create it, as follows:

Step 1: - Log in to GitLab and browse to the GitLab page for the project.

Step 2: - Select the "New branch" (accessed via the plus symbol) option.



Step 3: - Create a new branch named first_test derived from the existing master branch.



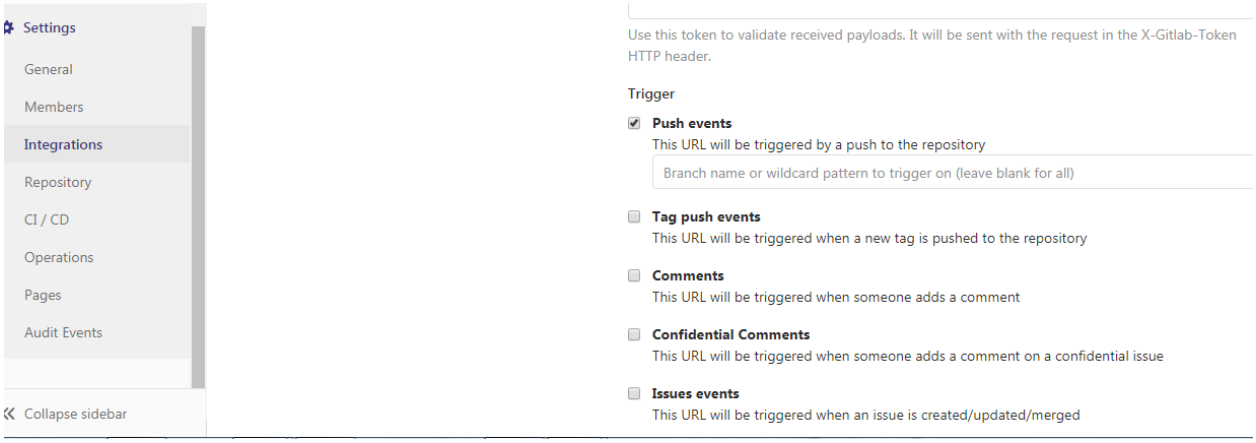
Step 4: - Click "Create branch" to create the new branch.

- **Connect GitLab And Jenkins**

The final step is to connect Jenkins and GitLab, such that Jenkins is automatically notified when there's a new commit to the GitLab repository and can commit successful builds back to it. To do this:

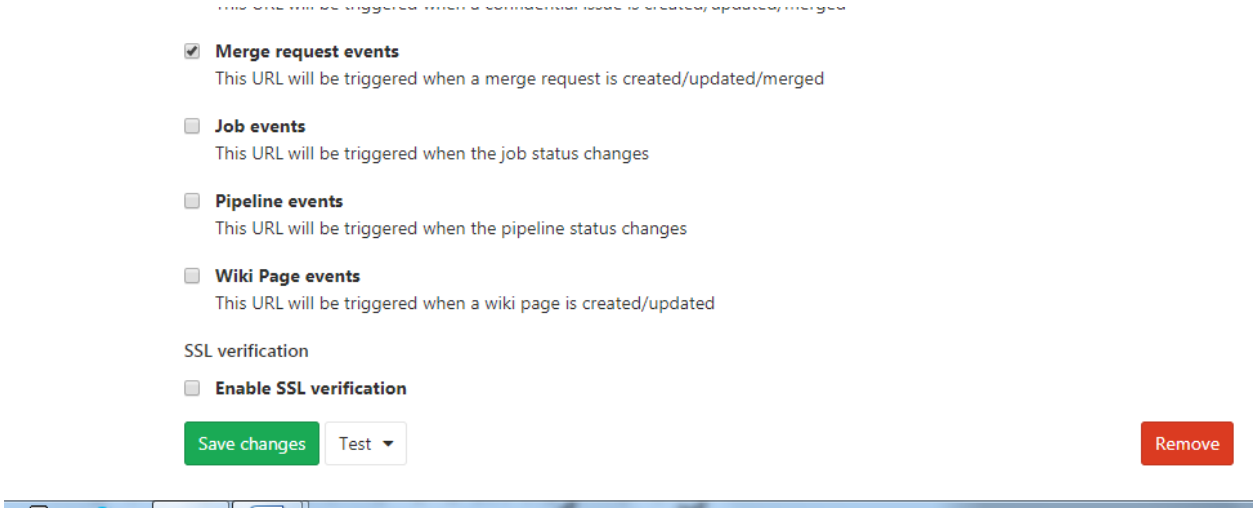
Step 1: - Log in to GitLab and browse to the GitLab page for the project.

Step 2: - Open the project setting page (accessed via the cogwheel icon in the top right corner) and select the "Webhooks" option.



Configure a new Jenkins webhook as follows:

- In the "URL" field, enter the Jenkins service URL
- In the list of triggers, ensure that the "Push events" trigger is checked. Optionally, select other events that should trigger a new Jenkins build, such as new tags or new comments.
- If you haven't configured SSL, disable SSL by unchecking the "Enable SSL verification" flag.



Step 3: - Click “Add Webhook” to save the new settings.

You’re all set! To verify that it all works, head back to your project, make a change to your project source code and commit the changes. Then, push the changes to the GitLab repository

The push should automatically trigger a new build in Jenkins, with the build results visible on the Jenkins overview page for the project.

If the build is successful, Jenkins will also merge the build back to the first_test branch in the GitLab repository.

Your Continuous Integration (CI) pipeline is now ready for use!

Reference:- https://medium.com/@teeks99/continuous-integration-with-jenkins-and-gitlab-fa770c62e88a

Task: - Configure GitHub and Jenkins for GitHub pull request Builder

First create a separate repo on git by using git bash because when we do fork it fetches account of that person who have repository.

- Create separate repository
- Create branch apart from master

```
Already up to date.

sudarshan.sharma@YI1009261DT MINGW64 ~/Desktop/maven_proj (firstbranch)
$ history
    1  git init
    2  git remote add origin "https://github.com/sudarshan1995/maven_proj.git"
    3  ll
    4  git pull origin master
    5  ll
    6  ll
    7  git  branch
    8  ll
    9  git clone https://github.com/topgun93/spring-project.git
   10  ll
   11  cd spring-project/
   12  ll
   13  mv * ../
   14  cd ..
   15  ll
   16  rm -rvf spring-project/
   17  ll
   18  git status
   19  git add -A
   20  git status
   21  git branch firstbranch
   22  git checkout firstbranch
   23  ll
   24  git statuds
   25  git status
   26  touch sample.txt
   27  vim sample.txt
   28  git add -A
   29  git commit -a -m "added sample file"
   30  git checkout master
   31  git commit -a -m "version 1"
   32  ssh -T git@github.com
   33  ll
   34  git status
   35  git checkout firstbranch
   36  git push origin firstbranch
   37  git checkout master
   38  git push origin master
   39  git checkout firstbranch
   40  git merge master
   41  git checkout master
   42  ll
   43  git merge master
   44  ll
   45  git branch
   46  git branch firstbranch
```

```
   43  git merge master
   44  ll
   45  git branch
   46  git branch firstbranch
   47  ll
   48  git branch
   49  git checkout firstbranch
   50  git merge origin master
   51  git merge master
   52  history
```

Now configure Jenkins

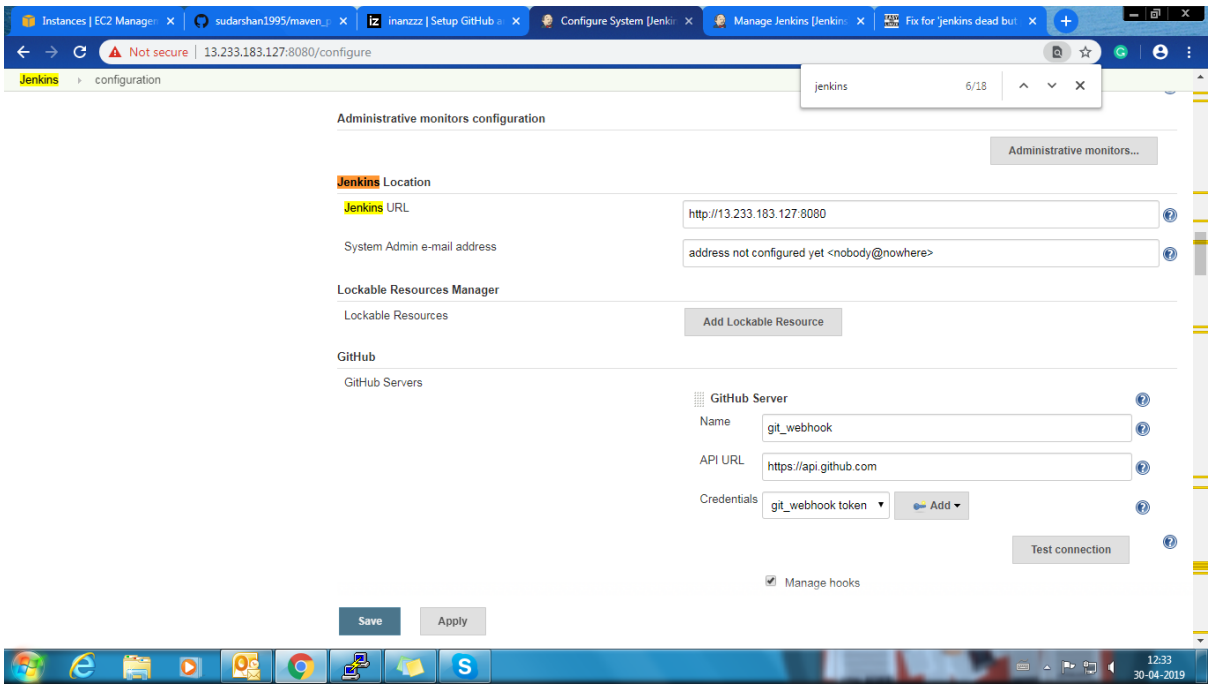
Install Plugin: - Jenkins GitHub plugin

Step 1: - Installation of plugin

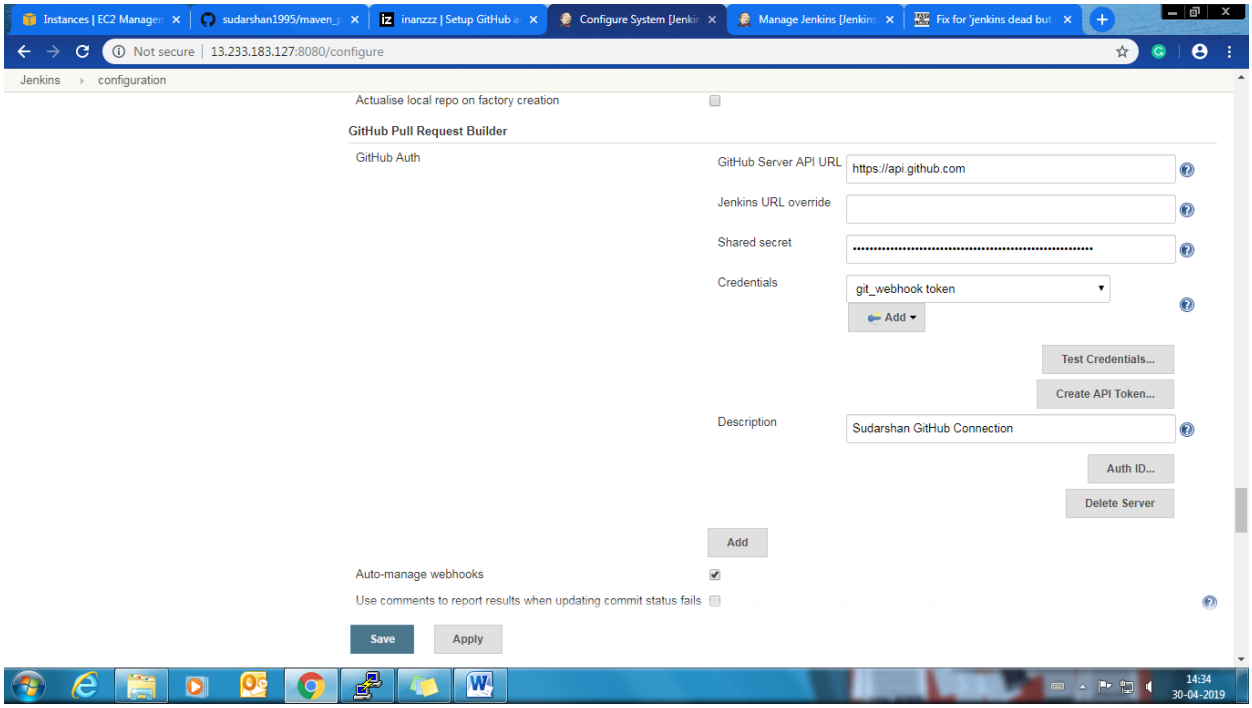
- Go to http://localhost:8080.
- Manage Jenkins → Manage Plugins
- If not installed by default, find and tick "Git plugin", "GitHub plugin", "GitHub Pull Request Builder" and "Rebuilder" plugins.
- Click "Install without restart" button.
- Restart Jenkins with $ sudo service Jenkins restart.

Step 2: - Go to configure system.

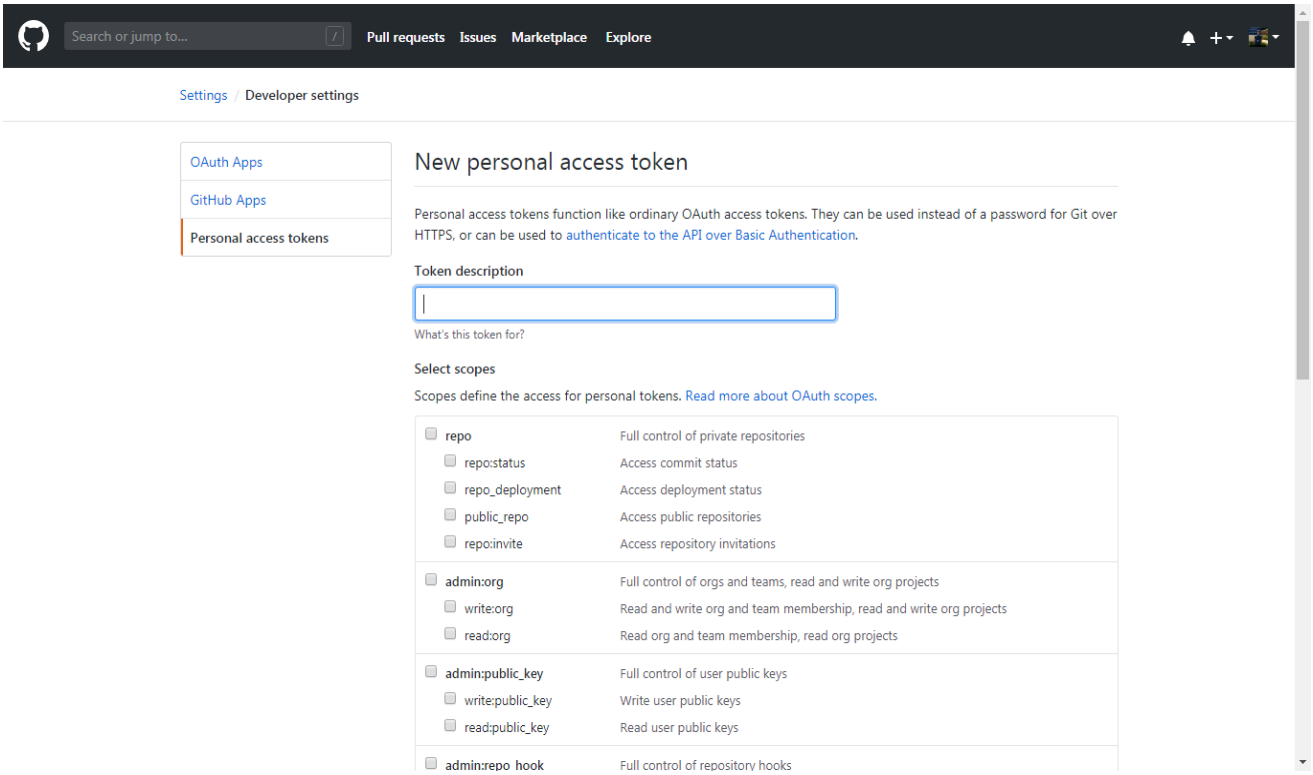
- Manage Jenkins → Configure System
- Under "**GitHub Pull Request Builder**" section, do the following.



Step 3: - select credential in **GitHub Pull request builder**



Step 4: - First, in GitHub, go to account settings -> Developer Settings -> Personal access tokens



Step 5: - Select the following Permissions

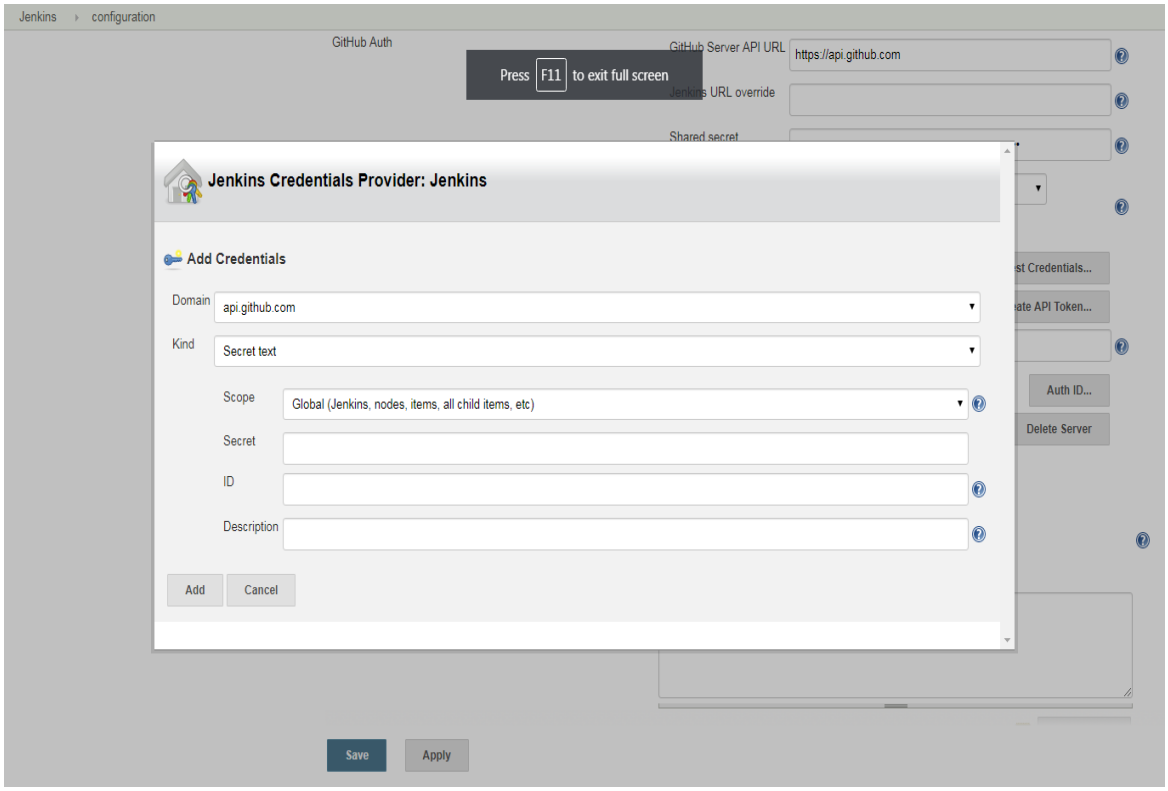
-repo
-admin:org
-admin:public_key
-admin_org_hook
-user
-admin:gpg_key

Step 5: - Then click “Generate token”. There you will get a token on the screen. Copy the token in the Jenkins Credential Provider as shown above, and paste it in Secret, with some description. After adding credentials, select the added credential for GitHub.

Now go to jenkins and the credential there

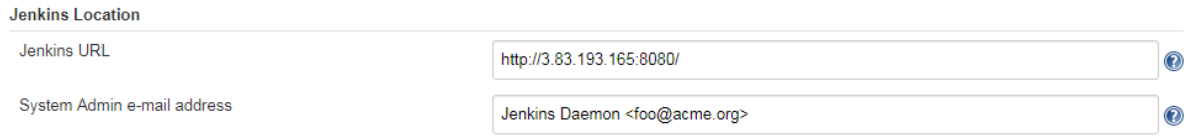
- One with username and password
- With secret text

Step 6: - Select the credentials. If you have not entered any credentials, the you will have to add credentials and click on add.

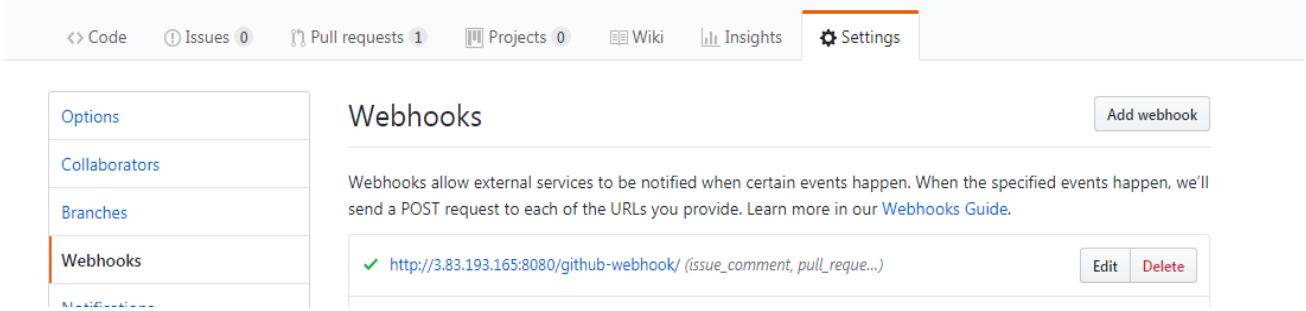


- **Configure GitHub web hooks**

Step 1: - Under Jenkins Location Enter the Jenkins IP as shown below.



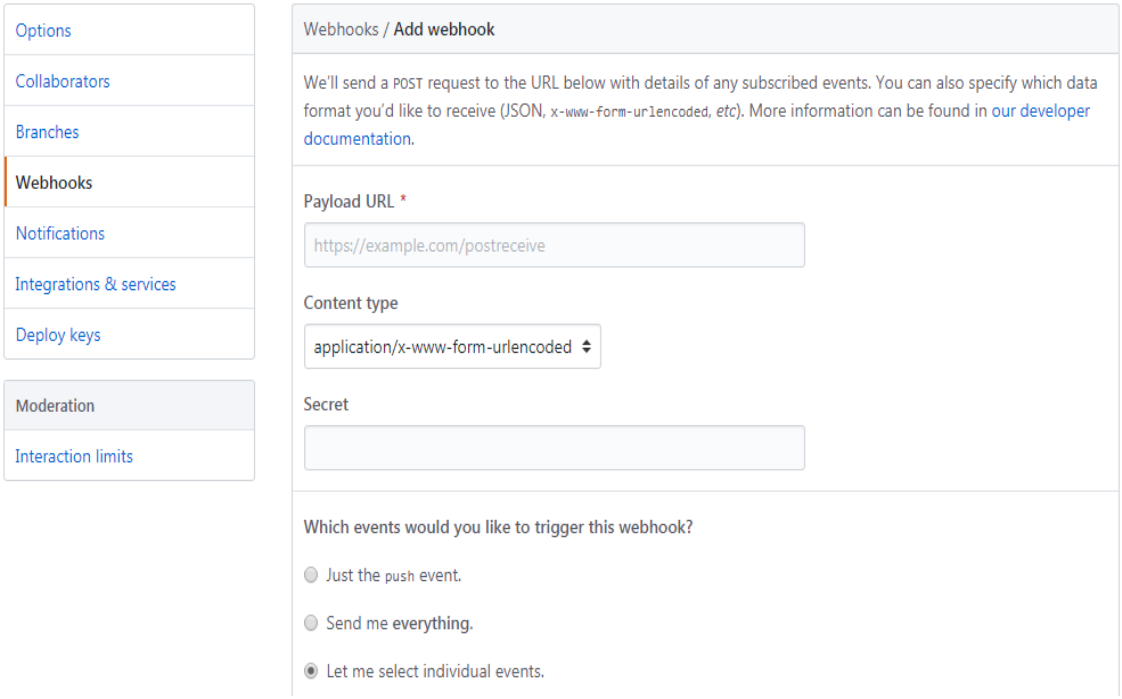
Step 2: - Go to your project setting → webhooks → click on Add webhook → and enter the payloadURL http://your-jenkins-domain.com:8080/github-webhook/



Step 3: - Then Select "Let me select individual events". Then select following options

- ✓ **Issue Comments**
- ✓ **Pull Requests**
- ✓ **Pushes**

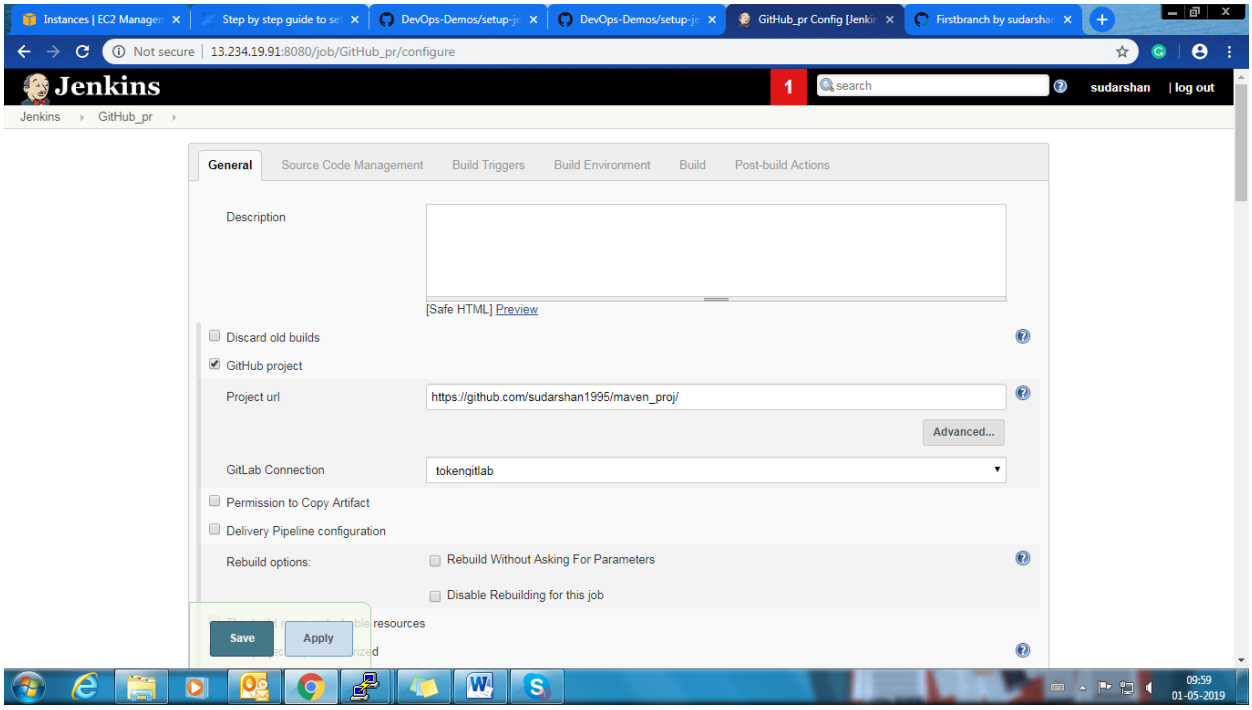
Step 4: - Then click "**Add Webhook**"



- [Create a Jenkins Project]

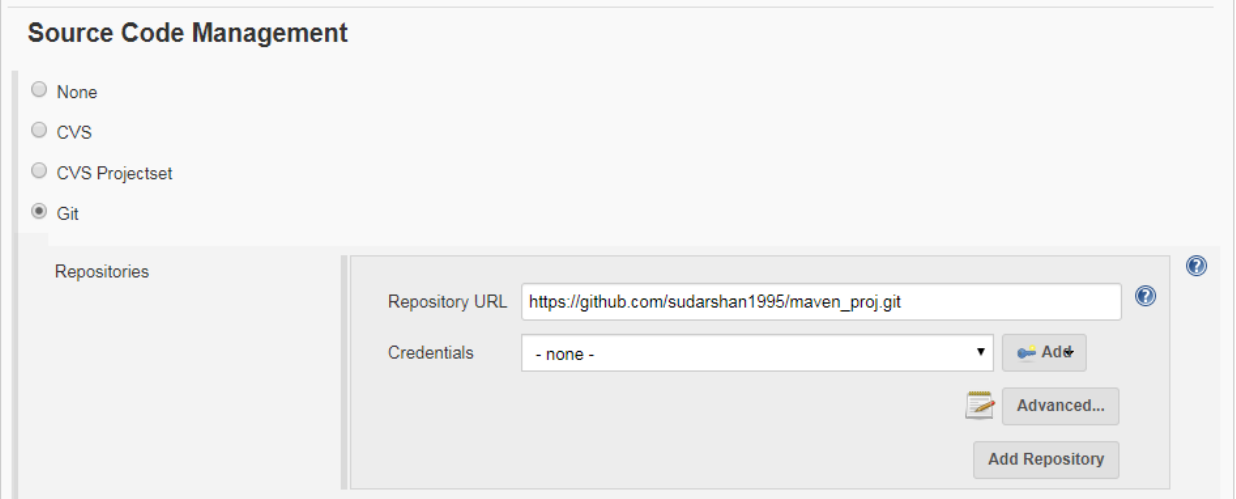
Step 1: - To create new job go to New Items → Select the project which you want to select → click ok.

Step 2: -In General, Select GitHub Project → Enter the repo URL

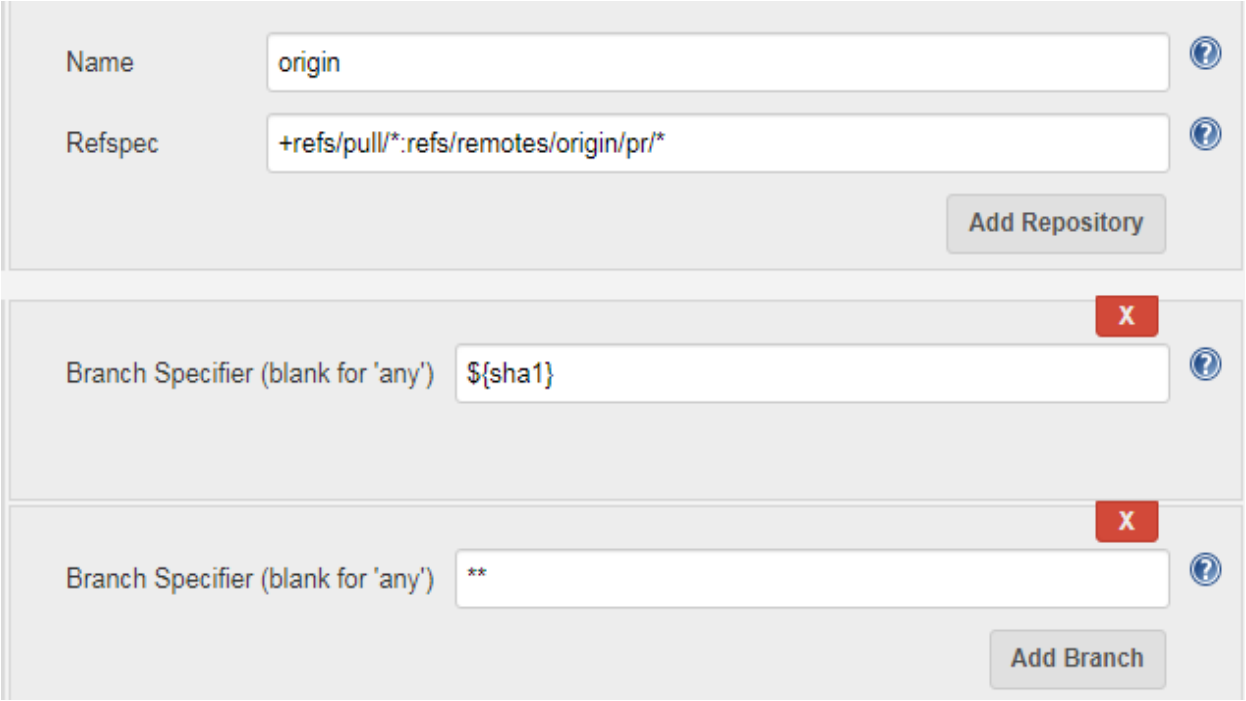


Don't use .git in url of git

Step 3: - In Source Control Management, select Git → Enter the .git repo URL
If the repo is private, then select the credentials related to that repo account else if its public then credentials are not required.



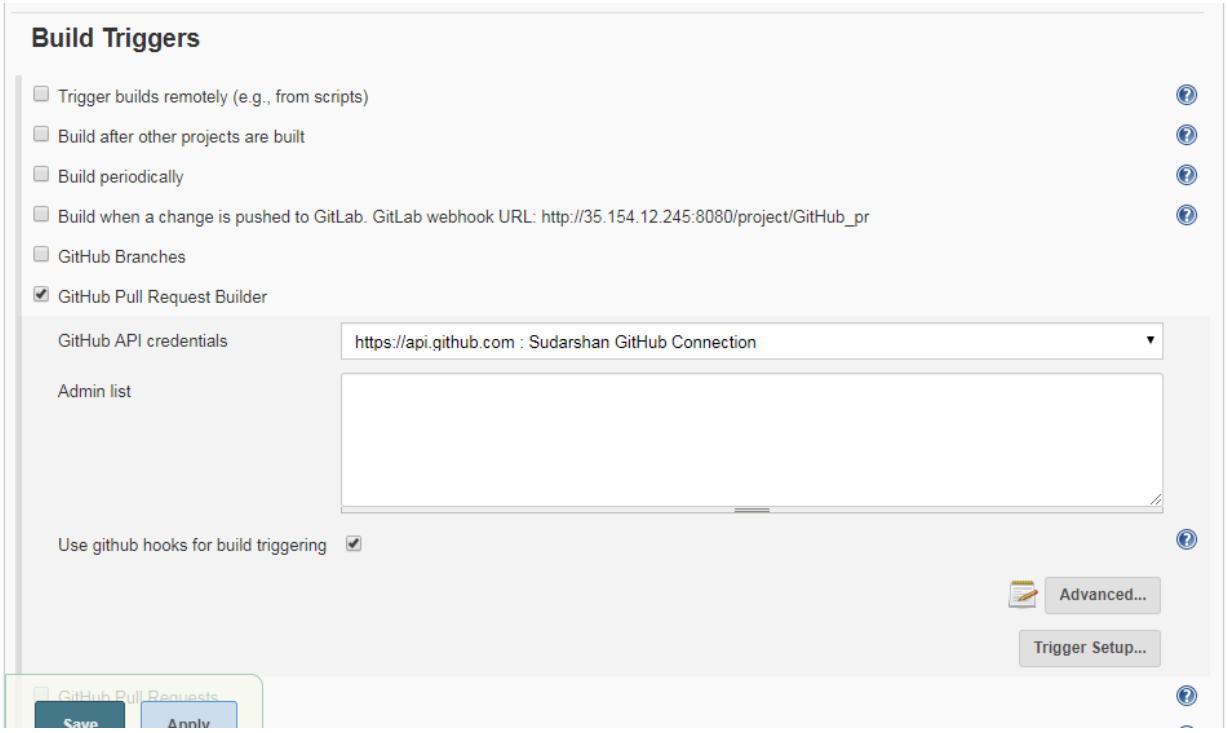
Step 4: - Click "Advanced" button then type origin in "Name" and refs/pull/*:refs/remotes/origin/pr/* in "Refspec" field.



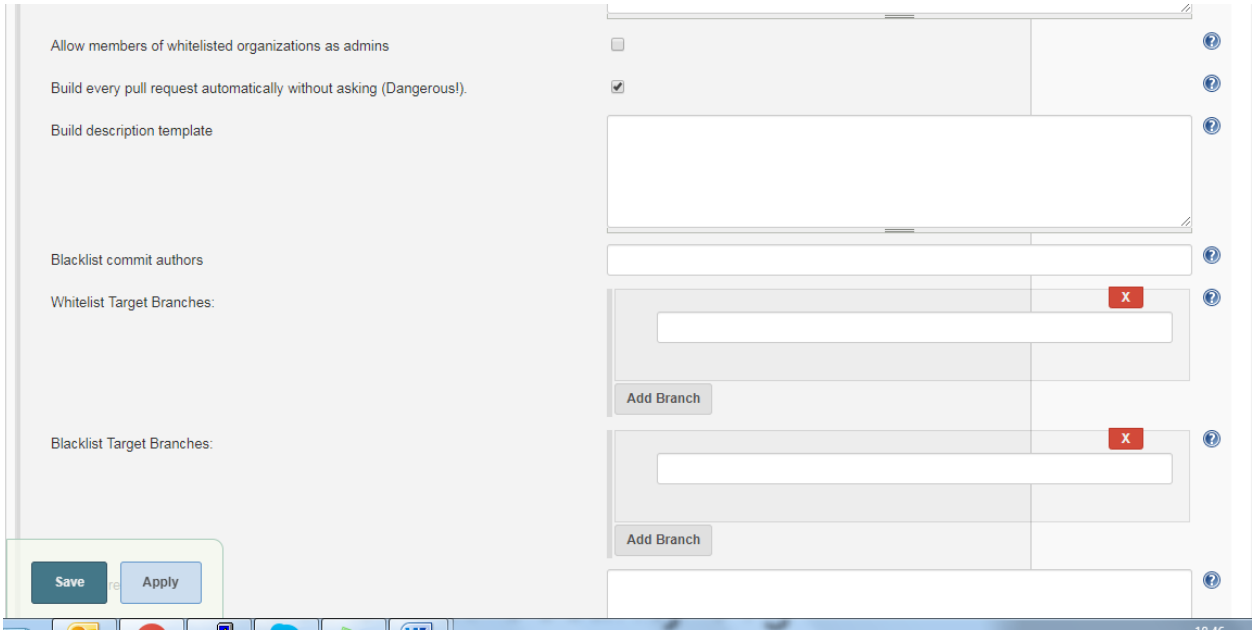
Step 4: - Type `${sha1}` in "Branch Specifier" field Then click “Add Branch” and type “**”.

Step 5: - In Build Triggers, Select GitHub Pull Request Builder →Select Use GitHub hooks for build triggering.

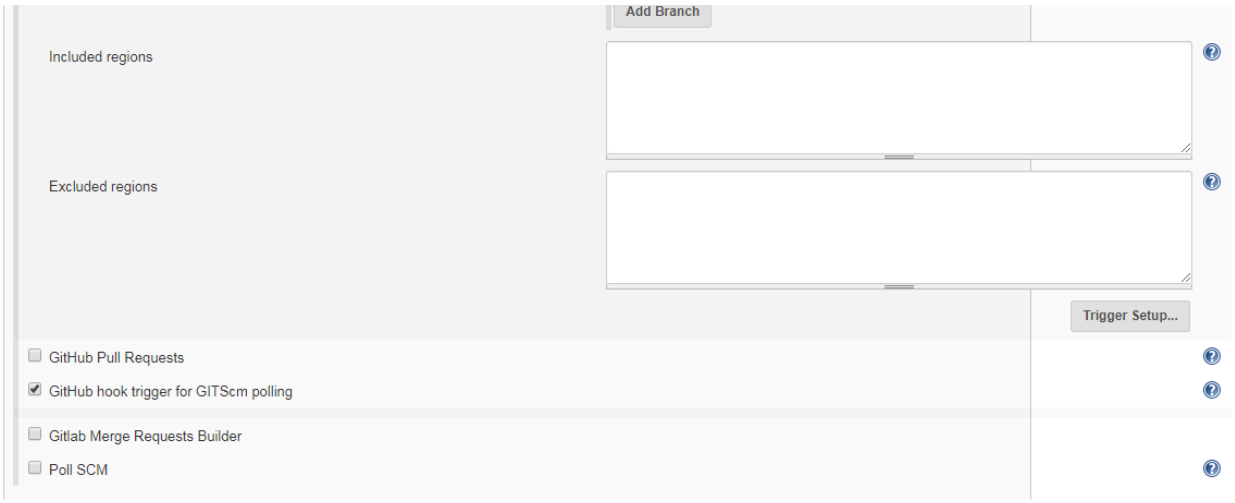
- Tick github pull request builder
- And click on **advance** button
- Tick on **github hooks for build triggering**



Step 5: - Mark on **build every pull request automatically without asking (Dangerous!).**

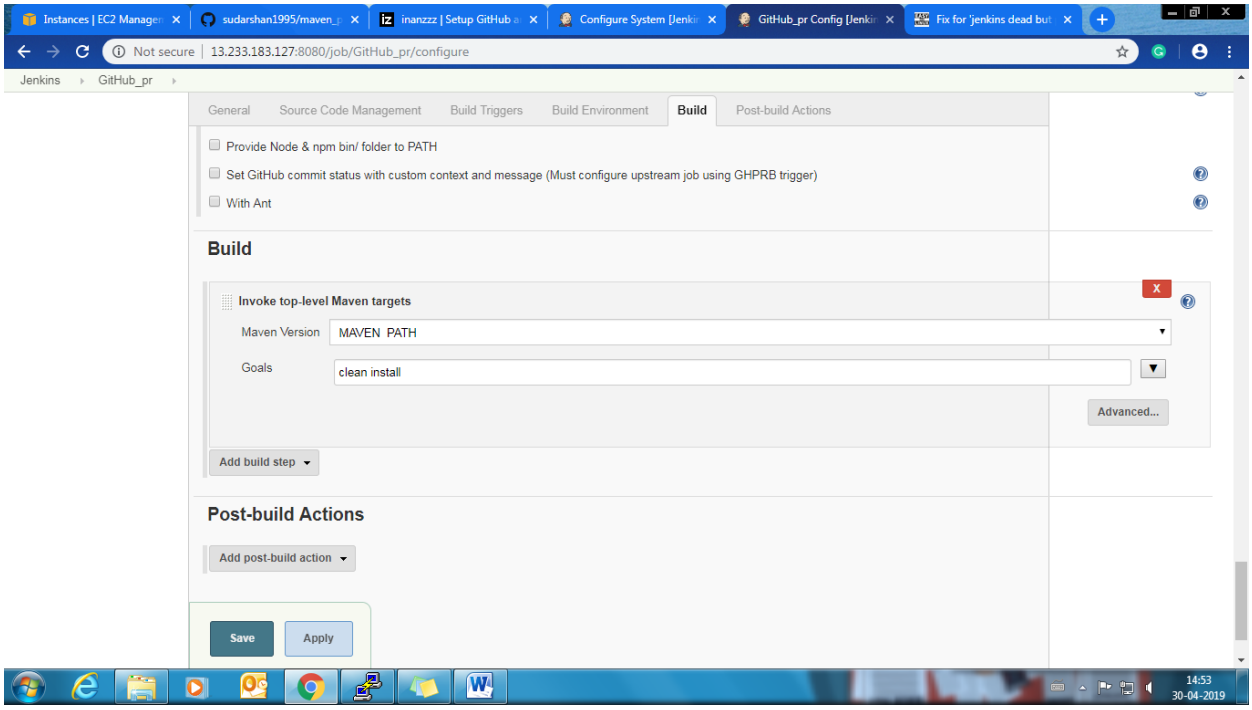


Step 5: - Select GitHub hook trigger for GITScm polling



Step 6: - In Build Environment, no changes and configuration is required.

Step 7: - In Build (in case of Maven Project) select Invoke top-level Maven targets → select Maven Version based on global toll configuration → Then enter goals based on Maven lifecycle. In POM, enter the exact path where pom file is present in repo.

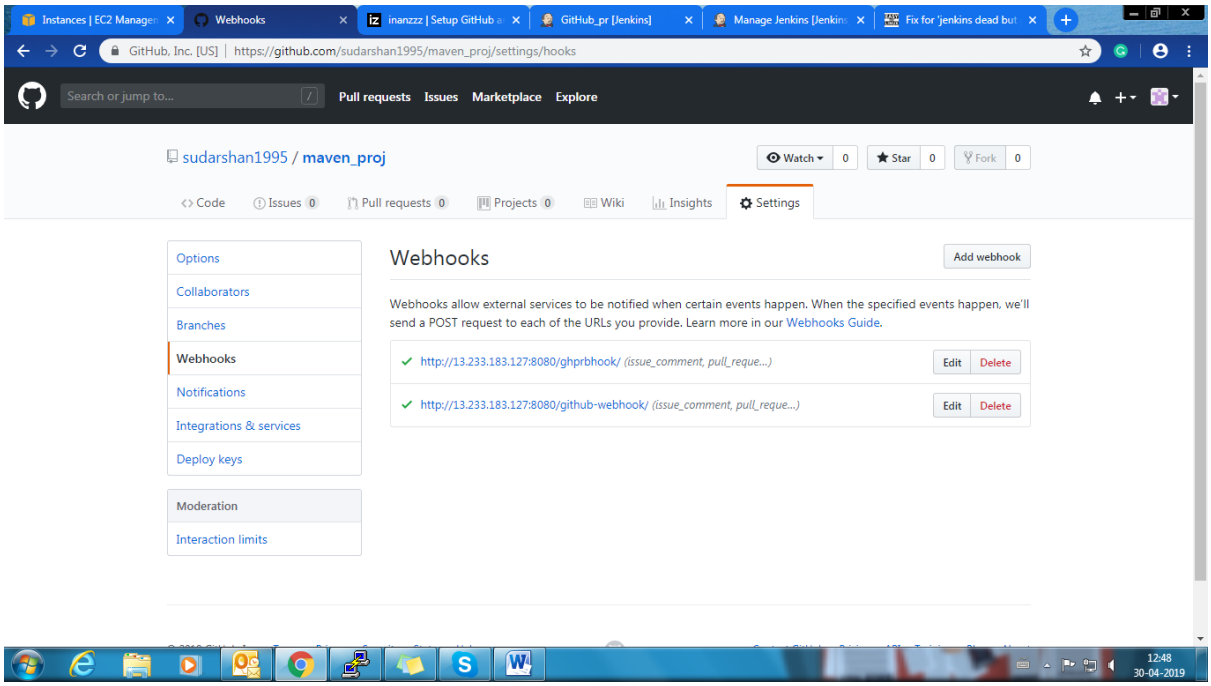


Step 8: - In Post-build Actions, nothing is required and clicks on Apply and Save.

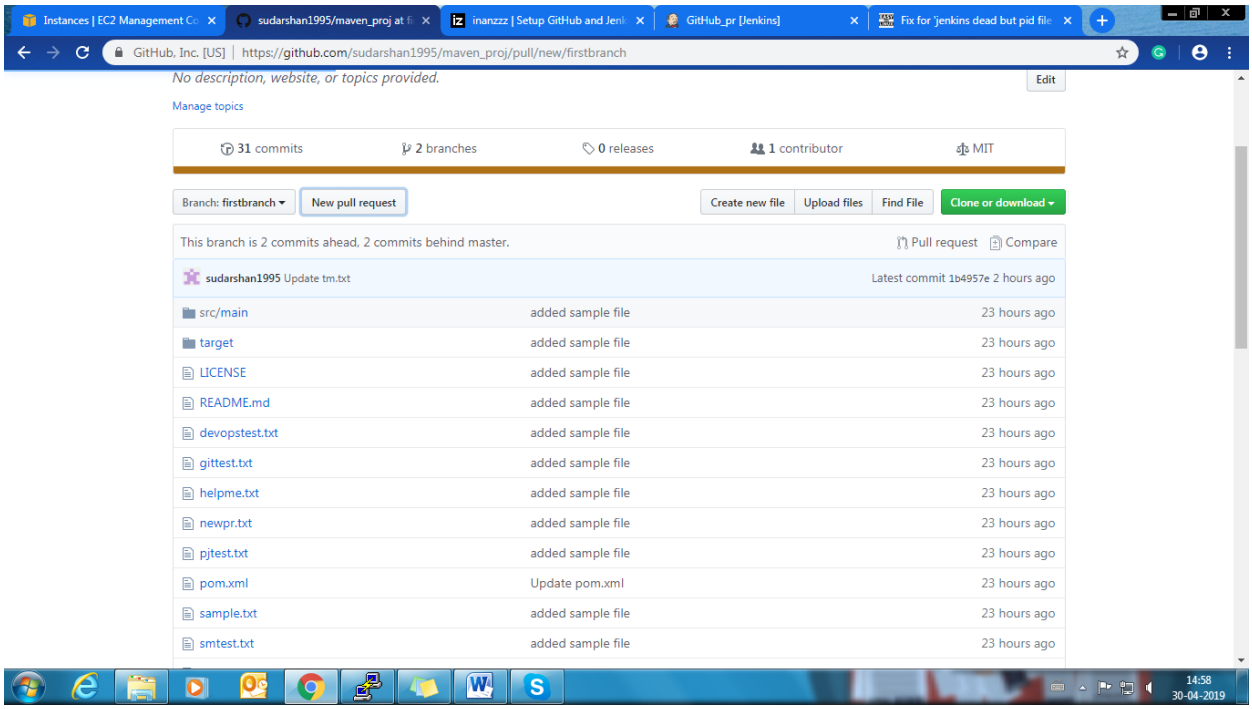
- After Saving the job, in Web Hooks in GitHub you will be able to see another URL
- To check whether the job is working or not, commit some changes in any file of the branch of a repo and make a New Pull request.

Step 9: - Now go in GitHub project and create webhooks

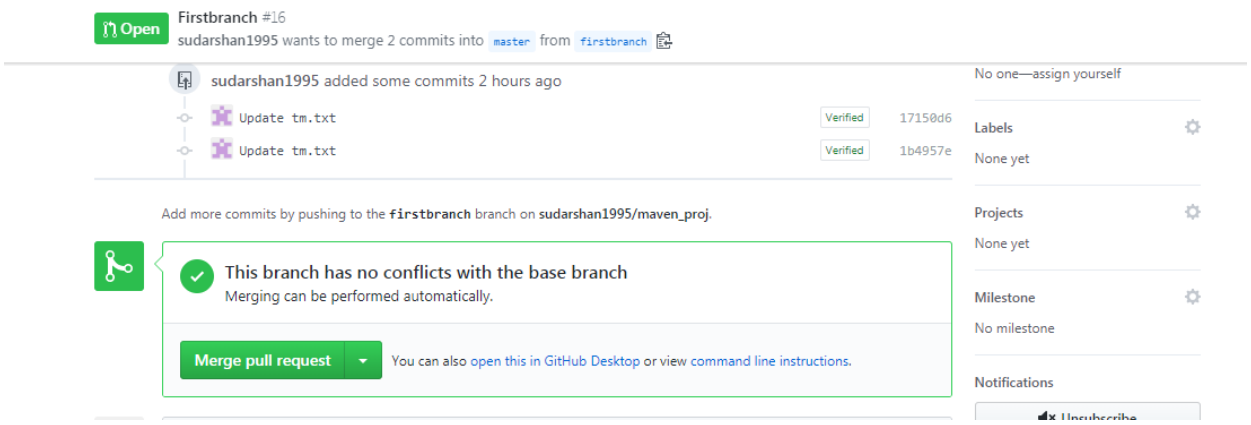
http://13.233.183.127:8080/ghprbhook/ (issue_comment, pull_request, and push)
http://13.233.183.127:8080/github-webhook/ (issue_comment, pull_request, and push)



Step 10: - Now make any changes in **firstbranch** file and click on **New Pull request**

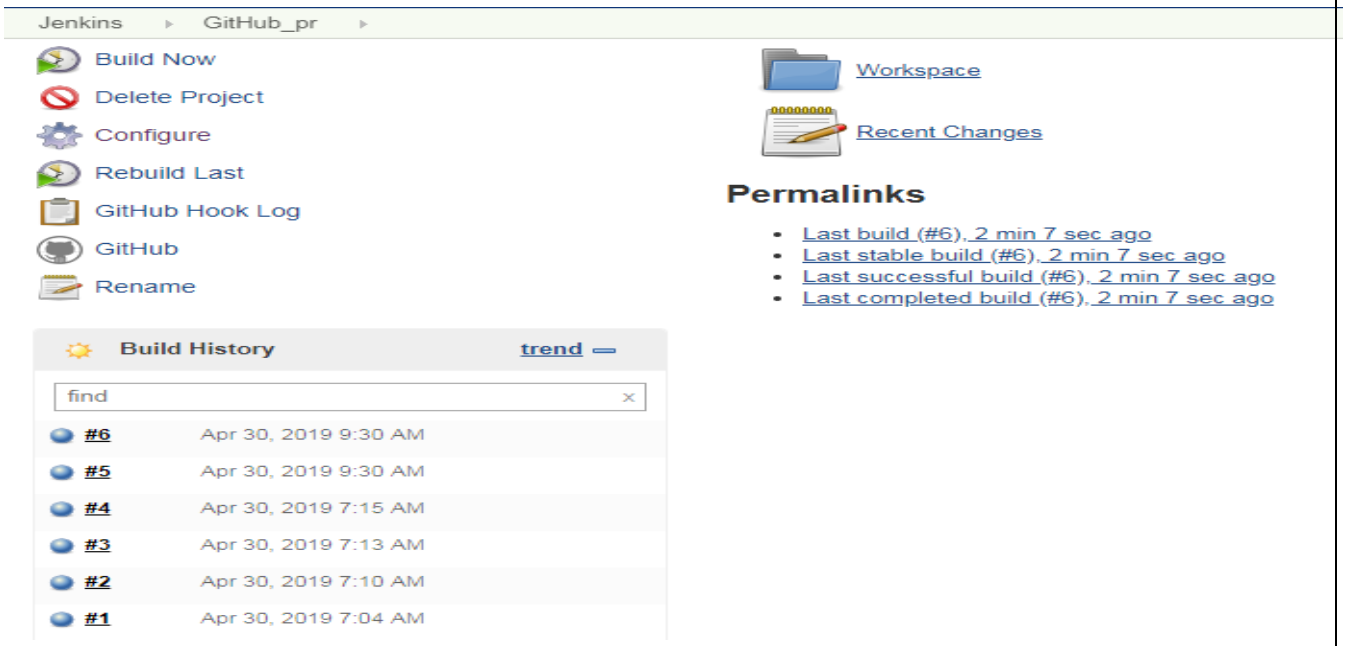


Step 11: - Now merge pull request.



Step 12: - With this you have successfully created a job. You can close your pull request if you don't want to merge in master branch.

Step 13: - In the Jenkins building of project had started



Reference: - http://www.inanzzz.com/index.php/post/ljgv/setup-github-and-jenkins-integration-for-pull-request-builder-and-merger

Troubleshooting: - Fix for 'Jenkins dead but pid file exists' error

Here is the Fix for error Jenkins dead but pid file exists You see this error when you check Jenkins service status. One important thing to note that you won't find this error while starting the Jenkins service.

```
[root@localhost vagrant]# /etc/init.d/jenkins start
Starting Jenkins                                   [  OK  ]
[root@localhost vagrant]# /etc/init.d/jenkins status
jenkins dead but pid file exists
```

Fix1: This issue can happen if Jenkins's user does not have enough permission on **/var/log/Jenkins and /var/cache/Jenkins**

Permission issue usually happens when you change Jenkins user from default.

First find your Jenkins user:

CentOS/RedHat:

```
[root@localhost vagrant]# cat /etc/sysconfig/jenkins | grep JENKINS_USER
JENKINS_USER="jenkins"
```

Debian/Ubuntu:

```
[root@localhost vagrant]# cat /etc/default/jenkins | grep JENKINS_USER
JENKINS_USER="jenkins"
```

Now make all files and directories under /var/log/Jenkins and /var/cache/Jenkins owned by the Jenkins user.

chown -R jenkins. {**/var/log/jenkins/**,**/var/cache/jenkins/**}

Then give proper permission for all files and directories.

```
find {/var/cache/jenkins,/var/log/jenkins} -type f -exec chmod 644 {} \;
find {/var/cache/jenkins,/var/log/jenkins} -type d -exec chmod 755 {} \;
```

Jenkins should be fine now.

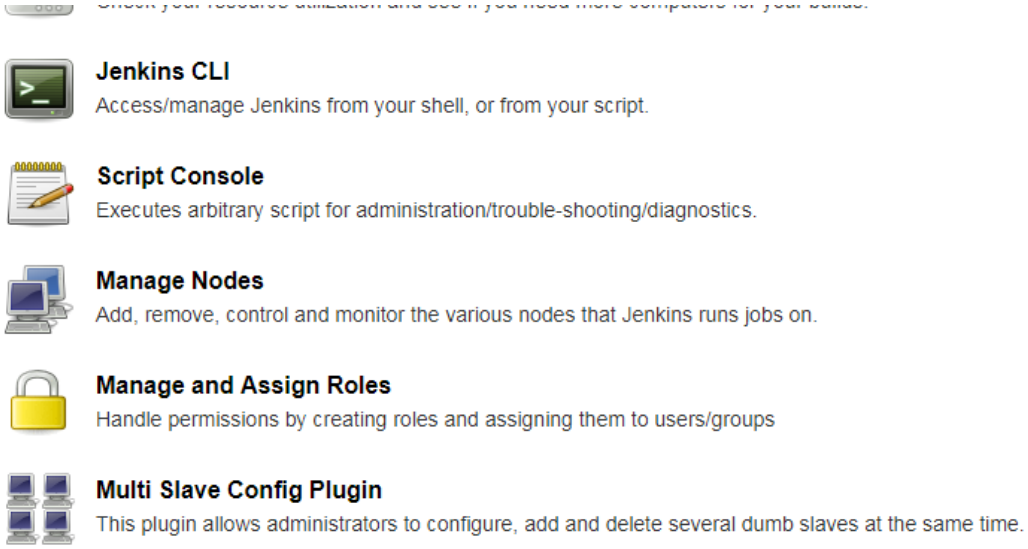
```
[root@localhost vagrant]# /etc/init.d/jenkins start
Starting Jenkins                                   [  OK  ]
[root@localhost vagrant]# /etc/init.d/jenkins status
jenkins (pid  9049) is running...
```

Task: - Configuration of Jenkins Master-Agent

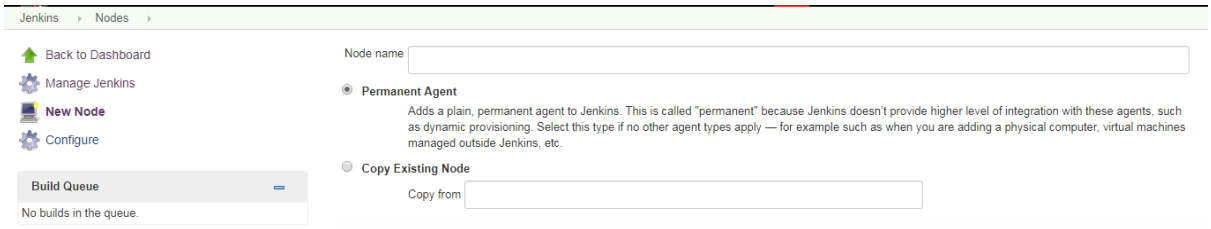
Configuration via ssh and connecting agent to the master.

Step 1: - To configure master-agent

Go to manage Jenkins → click on Manage Nodes

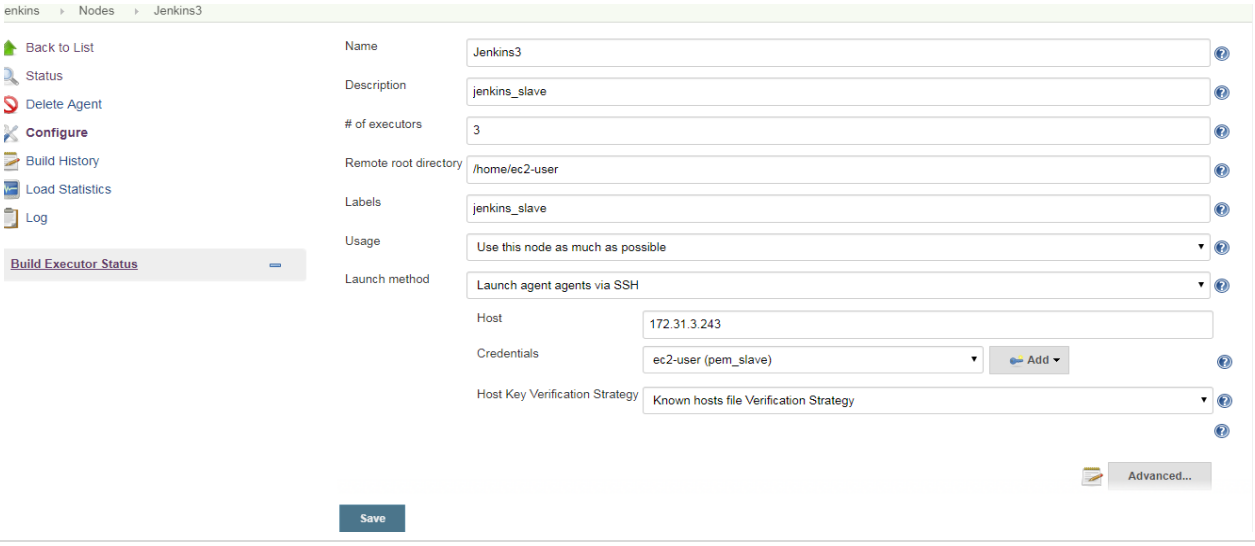


Step 2: - Create new node. Click on "New Node"



Enter "Node name", then select "Permanent Agent".

Step 2: - Configure new node

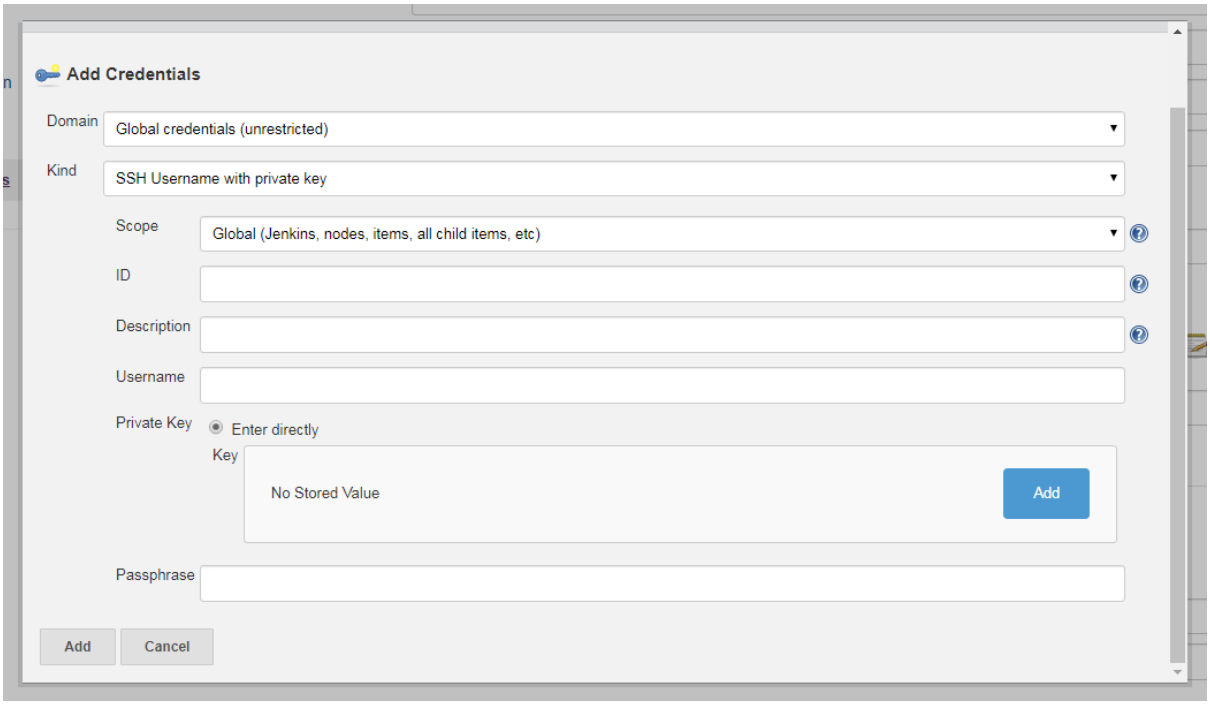


- Enter the “# of executors” as per your job requirement, and then enter the “Remote root directory”, where you want the job to be executed in the agent node.
- Enter the “Label”, so that while running job if you enter the label for nodes, then if any node is busy while running the job, then another node will be given to the running job.
- Select the launch method, if you want to connect via SSH or via connecting it to master

Launch agent via SSH

Step 1: - Select “Launch agent via SSH” from dropdown menu.

Step 2: - Enter the host “Private IP Address” → select the Credentials, if not added, then click add.

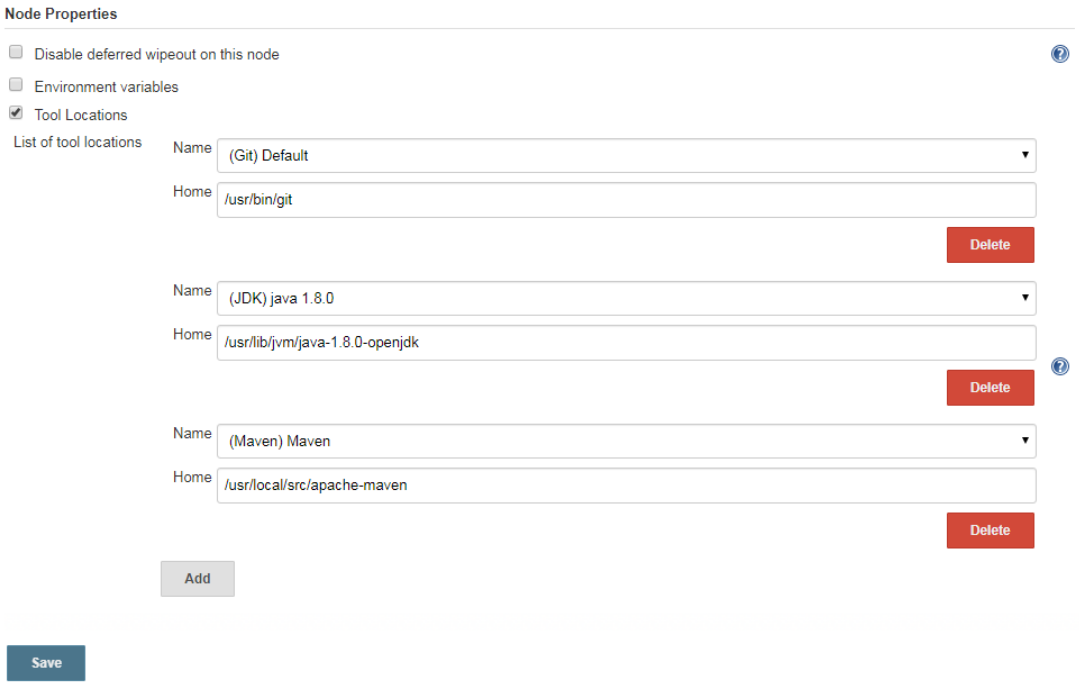


Here select Kind as “SSH Username with private key”, then enter the Username of the agent’s machine → select “Private Key” → enter master’s private SSH key.

Enter some description to identify the correct key while selecting, click “Add”.

Step 4: - Select the recently added credential for host from dropdown menu. Select “Host Key Verification Strategy” as “Known hosts file Verification Strategy”.

Step 5: - In “Node Properties” select “Tool properties” and from the list of tools select the tools and enter their respective path from the agent node.

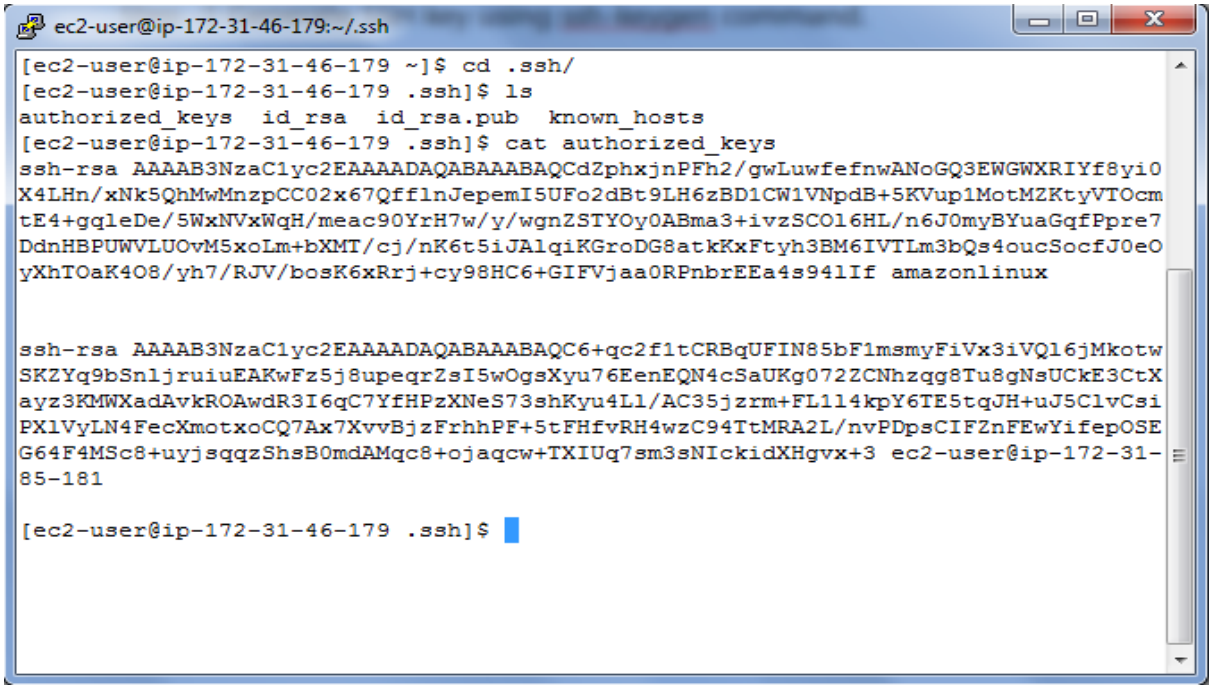


In Agent Node

Step 1: - Generate SSH key using ssh-keygen command.

Step 2: - Go to .ssh folder in the respective user home directory, then in authorized_keys file paste the public key of master.

For ssh: - ssh -i AMI-jenkins.pem ec2-user@private_ip_of_slave



Step 3: - Install latest java openjdk and devel as given above in java installation.

Step 4: - Install maven, as steps are given above in Maven installation.

Step 5: - Install Git.

In master machine

Step 1: - Generate the ssh key using ssh-keygen command.

Step 2: - Try to ssh between the systems.

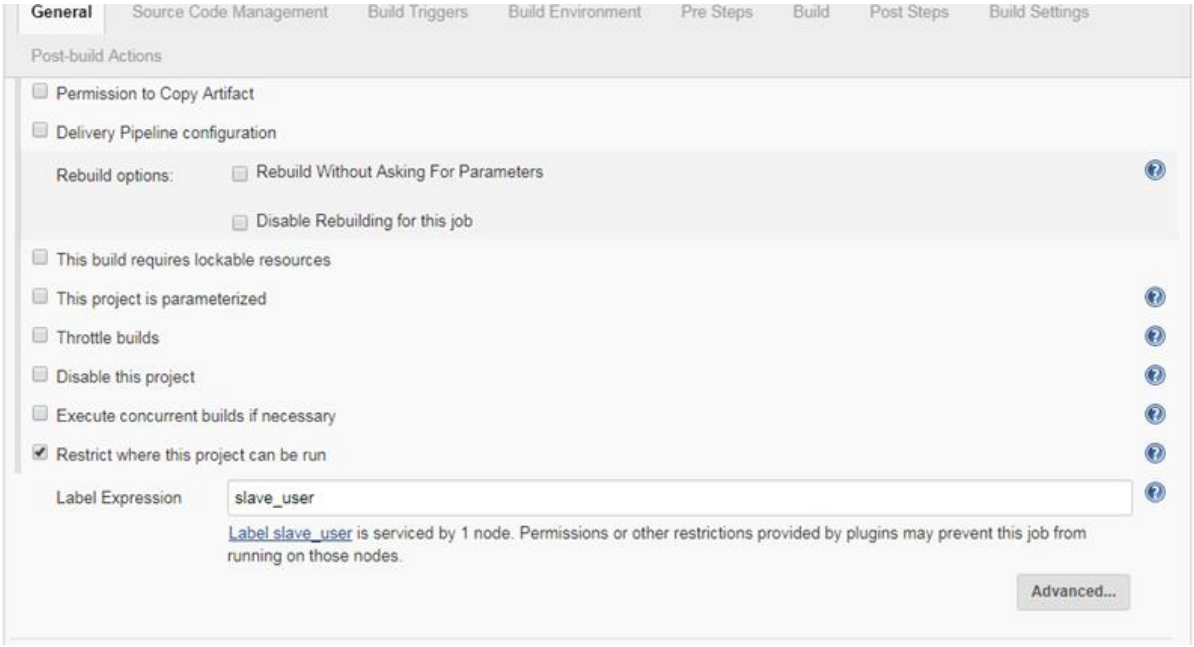
Step 3: - Then create .ssh folder in jerkins location (/var/lib/Jenkins) and create an empty known_hosts file with permission of 700 as sudoers.

Step 4: - Run the command ssh-keyscan –H <ip of agent> >> known_hosts inside /var/lib/Jenkins/.ssh directory.

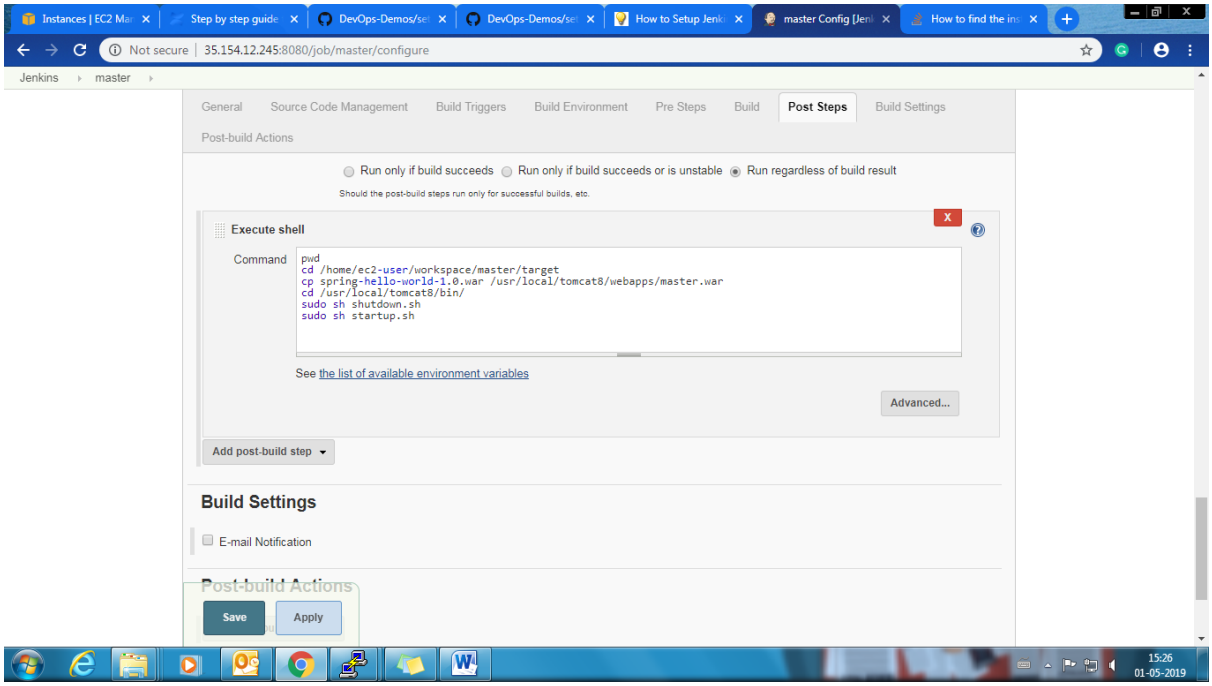
Configure jerkins new project

Step 1: - Create new job in “new items”.

Step 2: - Select “Restrict where the project can be run”, enter either the node name or the label name, based on the number of execution requirements.

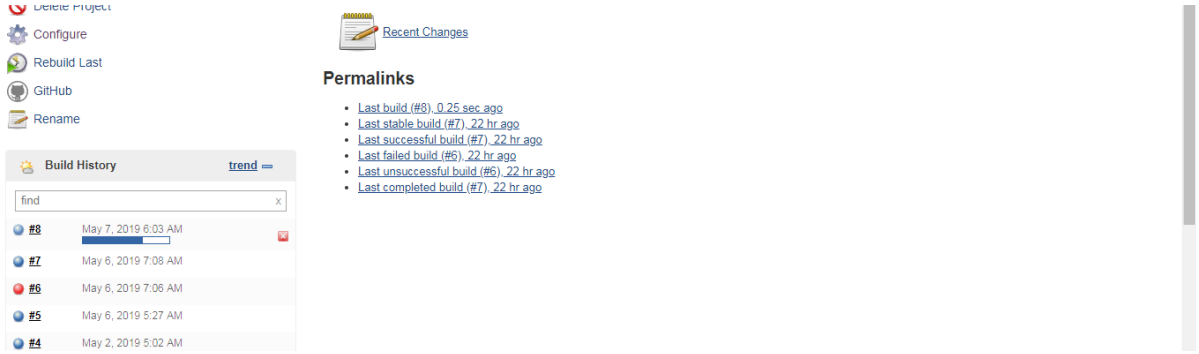


Step 3: - Select execute shell



Step 4: - Rest configures other part of the project the click “Apply and save”.

Step 5: - Run the test build. You should be able to see like this shown below.

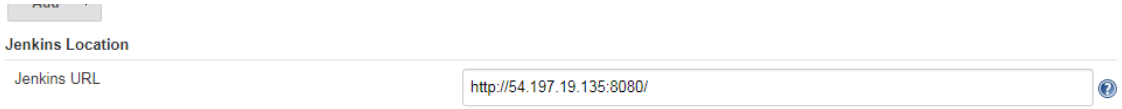


With this your job has been build successfully.

Launch agent by connecting it to master

Step 1: - sometimes this option is not enabled while adding node, then to enable this option go to Manage Jenkins -> configure global security -> Then in "Agents" select TCP port for inbound agent as Random.

Step 2: - Enter the correct Jenkins URL in configure system.



Step 3: - Create new node then follow same steps as in ssh configuration, in Launch method select "Launch agent by connecting it to master". Rest the configuration is same.

Step 4: - After saving the configuration, you will be able to see the following as showing below.

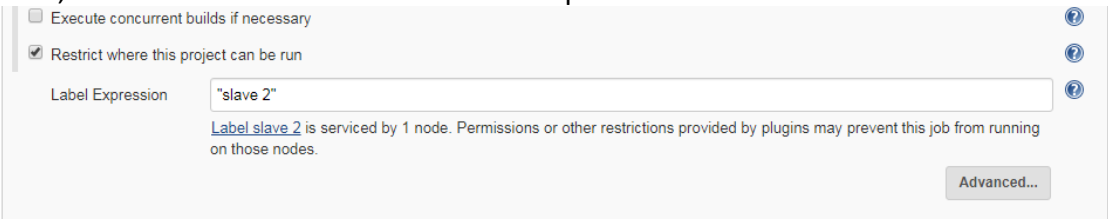
Step 5: - Just copy the following command in agent node, it should show like this

```
[ec2-user@ip-172-31-46-179 .ssh]$ cd
[ec2-user@ip-172-31-46-179 ~]$ java -jar agent.jar -jnlpUrl http://54.197.19.135
:8080/computer/slave%202/slave-agent.jnlp -secret 45bb438c7c674402403d434b24f16d
49d84bd15c9fd1dc3ba9fcb93bce0f52d4
May 07, 2019 6:18:55 AM hudson.remoting.jnlp.Main createEngine
INFO: Setting up agent: slave 2
May 07, 2019 6:18:55 AM hudson.remoting.jnlp.Main$CuiListener <init>
INFO: Jenkins agent is running in headless mode.
May 07, 2019 6:18:55 AM hudson.remoting.Engine startEngine
INFO: Using Remoting version: 3.29
May 07, 2019 6:18:55 AM hudson.remoting.Engine startEngine
WARNING: No Working Directory. Using the legacy JAR Cache location: /home/ec2-us
er/.jenkins/cache/jars
May 07, 2019 6:18:55 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Locating server among [http://54.197.19.135:8080/]
May 07, 2019 6:18:55 AM org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver
resolve
INFO: Remoting server accepts the following protocols: [JNLP4-connect, Ping]
May 07, 2019 6:18:55 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Agent discovery successful
  Agent address: 54.197.19.135
  Agent port:    36319
  Identity:      92:79:54:59:3c:e2:31:c0:08:36:22:83:33:f8:8e:06
May 07, 2019 6:18:55 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Handshaking
May 07, 2019 6:18:55 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connecting to 54.197.19.135:36319
May 07, 2019 6:18:55 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Trying protocol: JNLP4-connect
May 07, 2019 6:18:55 AM hudson.remoting.jnlp.Main$CuiListener status
```

Running job in agent node

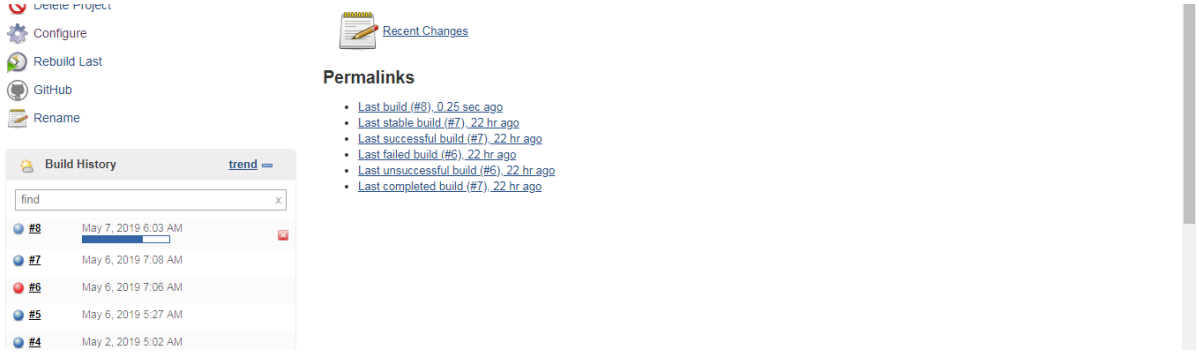
Step 1: - Create new job in “new items”.

Step 2: - Select “Restrict where the project can be run”, enter either the node name or the label name, based on the number of execution requirements.

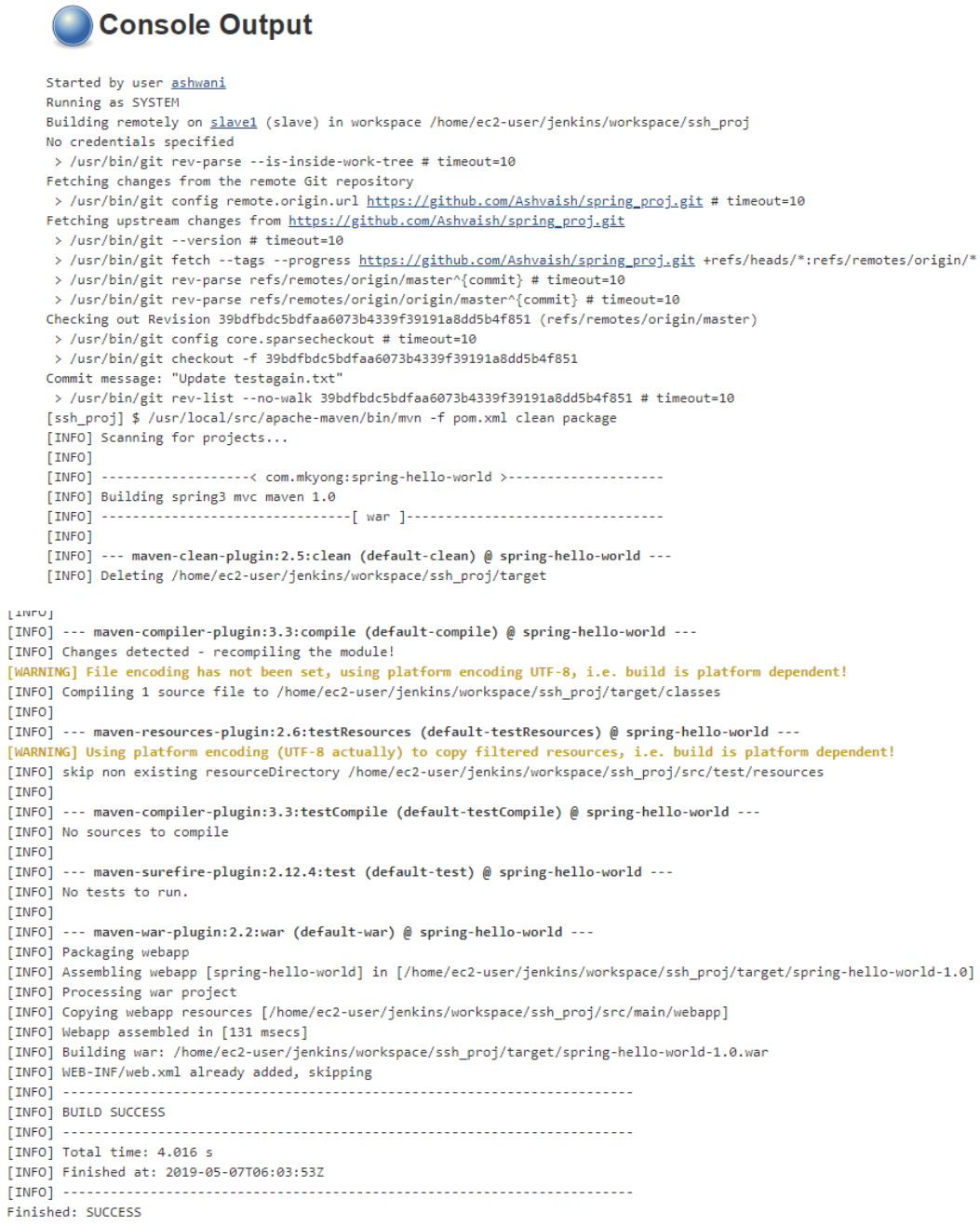


Step 3: - Rest configures other part of the project the click “Apply and save”.

Step 4: - Run the test build. You should be able to see like this shown below.



Step 5: - Output



Console Output

```
Started by user ashwani
Running as SYSTEM
Building remotely on slave1 (slave) in workspace /home/ec2-user/jenkins/workspace/ssh_proj
No credentials specified
 > /usr/bin/git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > /usr/bin/git config remote.origin.url https://github.com/Ashvaish/spring_proj.git # timeout=10
Fetching upstream changes from https://github.com/Ashvaish/spring_proj.git
 > /usr/bin/git --version # timeout=10
 > /usr/bin/git fetch --tags --progress https://github.com/Ashvaish/spring_proj.git +refs/heads/*:refs/remotes/origin/*
 > /usr/bin/git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > /usr/bin/git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 39bdfbdc5bdfaa6073b4339f39191a8dd5b4f851 (refs/remotes/origin/master)
 > /usr/bin/git config core.sparsecheckout # timeout=10
 > /usr/bin/git checkout -f 39bdfbdc5bdfaa6073b4339f39191a8dd5b4f851
Commit message: "Update testagain.txt"
 > /usr/bin/git rev-list --no-walk 39bdfbdc5bdfaa6073b4339f39191a8dd5b4f851 # timeout=10
[ssh_proj] $ /usr/local/src/apache-maven/bin/mvn -f pom.xml clean package
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< com.mkyong:spring-hello-world >--------------------
[INFO] Building spring3 mvc maven 1.0
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ spring-hello-world ---
[INFO] Deleting /home/ec2-user/jenkins/workspace/ssh_proj/target
[INFO]
[INFO] --- maven-compiler-plugin:3.3:compile (default-compile) @ spring-hello-world ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file to /home/ec2-user/jenkins/workspace/ssh_proj/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ spring-hello-world ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/ec2-user/jenkins/workspace/ssh_proj/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.3:testCompile (default-testCompile) @ spring-hello-world ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ spring-hello-world ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ spring-hello-world ---
[INFO] Packaging webapp
[INFO] Assembling webapp [spring-hello-world] in [/home/ec2-user/jenkins/workspace/ssh_proj/target/spring-hello-world-1.0]
[INFO] Processing war project
[INFO] Copying webapp resources [/home/ec2-user/jenkins/workspace/ssh_proj/src/main/webapp]
[INFO] Webapp assembled in [131 msecs]
[INFO] Building war: /home/ec2-user/jenkins/workspace/ssh_proj/target/spring-hello-world-1.0.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.016 s
[INFO] Finished at: 2019-05-07T06:03:53Z
[INFO] ------------------------------------------------------------------------
Finished: SUCCESS
```

With this your job has been build successfully.

Reference:- https://www.howtoforge.com/tutorial/ubuntu-jenkins-master-slave/

SonarQube Installation

Important: SonarQube requires at least **2GB of RAM** to run efficiently. Please check the SonarQube official documentation to know the detailed prerequisites.

Step 1: - Update your CentOS system

First, update the system packages by running "yum update" and reboot the server.

- yum update
- shutdown -r now

Step 2: - Install the necessary packages

- yum install wget unzip -y

Step 3: - Install Java

Now, install Java and set the environment variables.

- yum install java-1.8.0-openjdk-devel.x86_64

You can check the java version by running the following command:

- java -version

Next, we need to set up the Java environment variables.

- cp /etc/profile /etc/profile_orig
- echo -e "export JAVA_HOME=/usr/lib/jvm/jre-1.8.0-openjdk" >> /etc/profile
- echo -e "export JRE_HOME=/usr/lib/jvm/jre" >> /etc/profile
- source /etc/profile

Now, verify that the Java environment variables are set up properly by running the following command:

- echo -e "$JAVA_HOME\n$JRE_HOME"

Step 3: - Create a new user for running SonarQube

This is because you cannot run the newer versions of elastic search as the root user.

- useradd sonarqube

Step 4: - Install and configure PostgreSQL

Install PostgreSQL repository by typing:

- sudo rpm -Uvh https://download.postgresql.org/pub/repos/yum/9.6/redhat/rhel-7-x86_64/pgdg-centos96-9.6-3.noarch.rpm

Install PostgreSQL database server by running:

- sudo yum -y install postgresql96-server postgresql96-contrib

Initialize the database:

- sudo /usr/pgsql-9.6/bin/postgresql96-setup initdb

Edit the /var/lib/pgsql/9.6/data/pg_hba.conf to enable MD5-based authentication.

- sudo nano /var/lib/pgsql/9.6/data/pg_hba.conf

Find the following lines and change peer to trust and idnet to md5.

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            ident
# IPv6 local connections:
host    all             all             ::1/128                 ident
```

Once updated, the configuration should look like the one shown below.

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
```

Start PostgreSQL server and enable it to start automatically at boot time by running:

- sudo systemctl start postgresql-9.6

- sudo systemctl enable postgresql-9.6

Change the password for the default PostgreSQL user.

- sudo passwd postgres

Switch to the postgres user.

- su – postgres

Create a new user by typing:

- createuser sonar

Switch to the PostgreSQL shell.

- Psql

Set a password for the newly created user for SonarQube database.

- ALTER USER sonar WITH ENCRYPTED password 'StrongPassword';

Create a new database for PostgreSQL database by running:

- CREATE DATABASE sonar OWNER sonar;

Exit from the psql shell:

- \q

Switch back to the sudo user by running the exit command

Step 5: - Download and install SonarQube

Download the latest version of SonarQube from here.

- wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-6.7.7.zip

Extract it using the following command.

- unzip sonarqube-6.7.5.zip
- mv sonarqube-6.7.5 /opt/sonarqube

Since we are running SonarQube as a separate user, assign proper ownerships to SonarQube files.

- chown sonarqube. /opt/sonarqube -R

Next, open the SonarQube configuration file “conf/sonar.properties” on your favorite text editor.

vi /opt/sonarqube/conf/sonar.properties
Find the following lines.

- #sonar.jdbc.username=
- #sonar.jdbc.password=

Uncomment and provide the PostgreSQL username and password of the database that we have created earlier. It should look like:

- sonar.jdbc.username=sonar
- sonar.jdbc.password=StrongPassword

Next, find:

- sonar.jdbc.url=jdbc:postgresql://localhost/sonar

Uncomment the line, save the file and exit from the editor.

Open the SonarQube startup script and specify the sonarqube user details.

vi /opt/sonarqube/bin/linux-x86-64/sonar.sh

Add the following entry to it.

- RUN_AS_USER=sonarqube

Step 5: - Starting SonarQube and Setting it as a Systemd service

You can now start SonarQube by running the following command.

```
/opt/sonar/bin/linux-x86-64/sonar.sh start
```

If you have done everything correctly, SonarQube will start listening on ports 9000. You can access it using

http://your_ip_address:9000

For setting SonarQube as a Systemd Service, create a new file "sonar.service"

under "/etc/systemd/system/".

vi /etc/systemd/system/sonar.service
Now, copy the below lines into it.

```
[Unit]
Description=SonarQube
After=syslog.target network.target

[Service]
Type=forking

ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start
ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop

User=sonarqube
```

```
Group=sonarqube
Restart=always

[Install]
WantedBy=multi-user.target
```

Now, enable SonarQube service to automatically start at boot time by running the following command.

```
systemctl enable sonar.service
```

Start SonarQube by running

```
systemctl start sonar.service
```

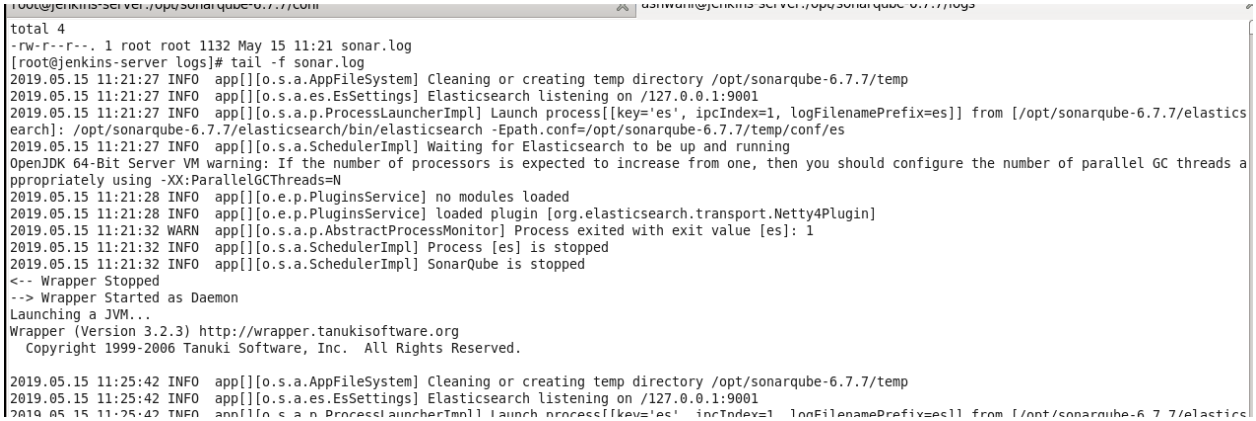
And stop SonarQube by running

```
systemctl stop sonar.service
```

The default username and password of SonarQube is admin and admin.

Troubleshooting: -

Sometimes there will be problem when starting SonarQube, The problem will be memory Heap issue for jvm, you can check it in sonar.log



```
total 4
-rw-r--r--. 1 root root 1132 May 15 11:21 sonar.log
[root@jenkins-server logs]# tail -f sonar.log
2019.05.15 11:21:27 INFO  app[][o.s.a.AppFileSystem] Cleaning or creating temp directory /opt/sonarqube-6.7.7/temp
2019.05.15 11:21:27 INFO  app[][o.s.a.es.EsSettings] Elasticsearch listening on /127.0.0.1:9001
2019.05.15 11:21:27 INFO  app[][o.s.a.p.ProcessLauncherImpl] Launch process[[key='es', ipcIndex=1, logFilenamePrefix=es]] from [/opt/sonarqube-6.7.7/elastics
earch]: /opt/sonarqube-6.7.7/elasticsearch/bin/elasticsearch -Epath.conf=/opt/sonarqube-6.7.7/temp/conf/es
2019.05.15 11:21:27 INFO  app[][o.s.a.SchedulerImpl] Waiting for Elasticsearch to be up and running
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to increase from one, then you should configure the number of parallel GC threads a
ppropriately using -XX:ParallelGCThreads=N
2019.05.15 11:21:28 INFO  app[][o.e.p.PluginsService] no modules loaded
2019.05.15 11:21:28 INFO  app[][o.e.p.PluginsService] loaded plugin [org.elasticsearch.transport.Netty4Plugin]
2019.05.15 11:21:32 WARN  app[][o.s.a.p.AbstractProcessMonitor] Process exited with exit value [es]: 1
2019.05.15 11:21:32 INFO  app[][o.s.a.SchedulerImpl] Process [es] is stopped
2019.05.15 11:21:32 INFO  app[][o.s.a.SchedulerImpl] SonarQube is stopped
<-- Wrapper Stopped
--> Wrapper Started as Daemon
Launching a JVM...
Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org
  Copyright 1999-2006 Tanuki Software, Inc.  All Rights Reserved.

2019.05.15 11:25:42 INFO  app[][o.s.a.AppFileSystem] Cleaning or creating temp directory /opt/sonarqube-6.7.7/temp
2019.05.15 11:25:42 INFO  app[][o.s.a.es.EsSettings] Elasticsearch listening on /127.0.0.1:9001
```

To resolve this error edit the sonar.properties file in your sonar directory inside conf /opt/sonar/conf/sonar.properties And change the memory heap in COMPUTE ENGINE

Sonar.ce.javaopts=Xmx1024m –Xms512m –XX:+HeapDumpOnOutOfMemoryError

```
#--------------------------------------------------------------------------------------------------
# COMPUTE ENGINE
# The Compute Engine is responsible for processing background tasks.
# Compute Engine is executed in a dedicated Java process. Default heap size is 512Mb.
# Use the following property to customize JVM options.
#    Recommendations:
#
#    The HotSpot Server VM is recommended. The property -server should be added if server mode
#    is not enabled by default on your environment:
#    http://docs.oracle.com/javase/8/docs/technotes/guides/vm/server-class.html
#
sonar.ce.javaOpts=-Xmx1024m -Xms512m -XX:+HeapDumpOnOutOfMemoryError

# Same as previous property, but allows to not repeat all other settings like -Xmx
#sonar.ce.javaAdditionalOpts=
```

As well as in WEB SERVER

Sonar.web.javaopts=Xmx1024m –Xms512m –XX:+HeapDumpOnOutOfMemoryError

```
#--------------------------------------------------------------------------------------------------
# WEB SERVER
# Web server is executed in a dedicated Java process. By default heap size is 512Mb.
# Use the following property to customize JVM options.
#    Recommendations:
#
#    The HotSpot Server VM is recommended. The property -server should be added if server mode
#    is not enabled by default on your environment:
#    http://docs.oracle.com/javase/8/docs/technotes/guides/vm/server-class.html
#
#    Startup can be long if entropy source is short of entropy. Adding
#    -Djava.security.egd=file:/dev/./urandom is an option to resolve the problem.
#    See https://wiki.apache.org/tomcat/HowTo/FasterStartUp#Entropy_Source
#
sonar.web.javaOpts=-Xmx1024m -Xms512m -XX:+HeapDumpOnOutOfMemoryError

# Same as previous property, but allows to not repeat all other settings like -Xmx
#sonar.web.javaAdditionalOpts=
```

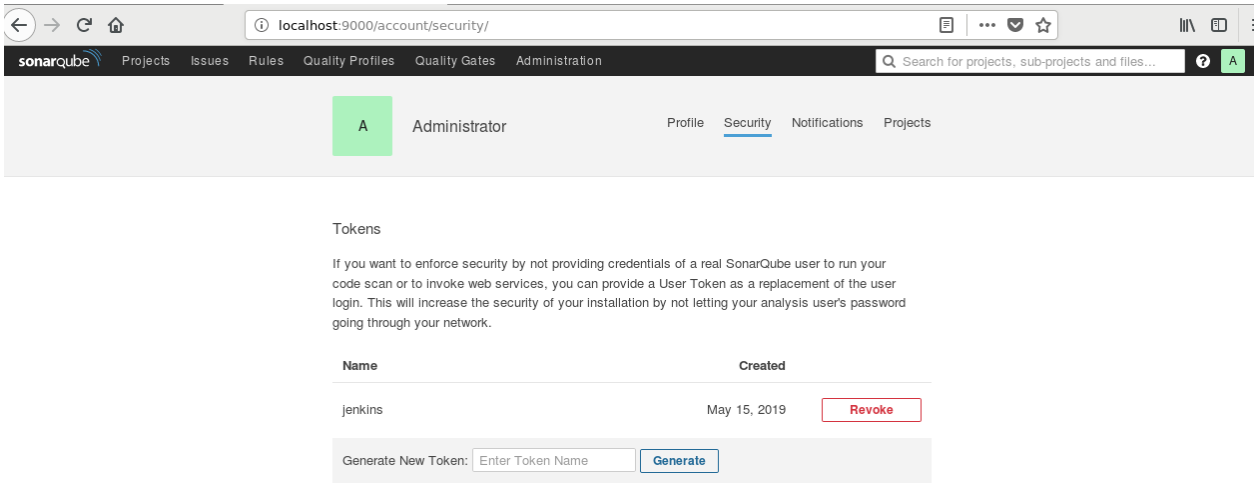
There will be error for permission, cannot be run as root. To resolve this problem, change the owner of the sonar directory to sonar.

Chown –R sonar:sonar /opt/sonar

Task: - Integrate SonarQube with Jenkins

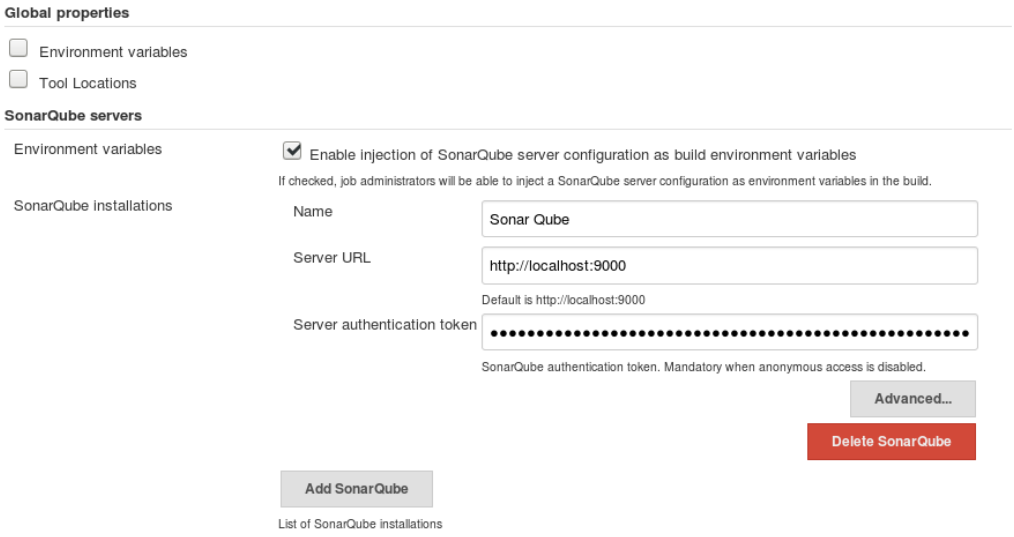
Step 1: - Generate authentication token in SonarQube

Go to SonarQube webpage using http://localhost:9000 then My Account, click on security tab.



Enter the name of token in “Generate New Token” then click Generate. A token will we generated copy and keep it somewhere safe.

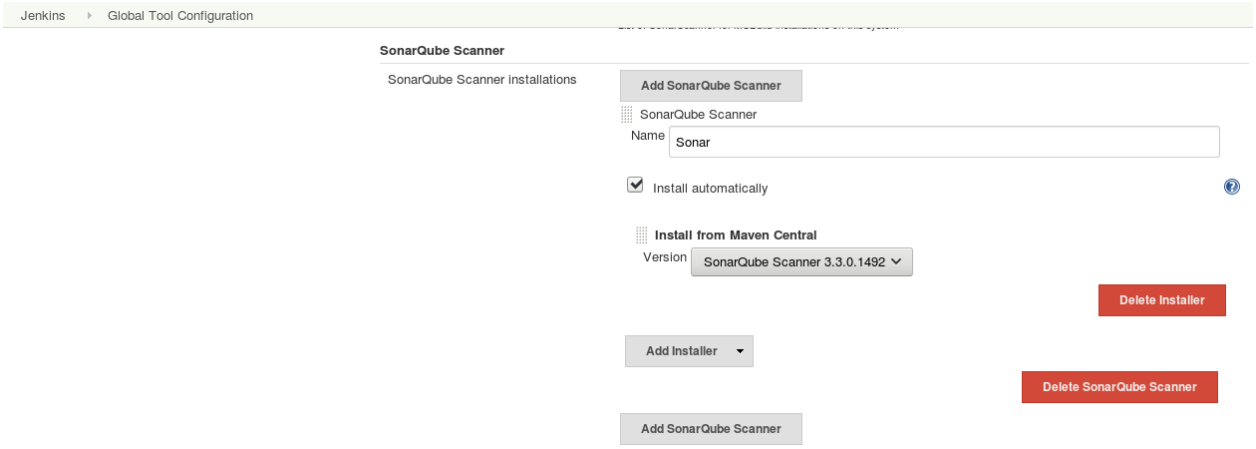
Step 2: - In Jenkins go to manage Jenkins → configure system



Go to SonarQube Servers, then enter the name, ServerURL and the Authentication token of Sonarqube, which you have to generate from SonarQube.

Again goto Global Tool Configuration then,

Go to SonarQube Scanner, enter the name then select the version which you want to install.



Click Apply and Save.

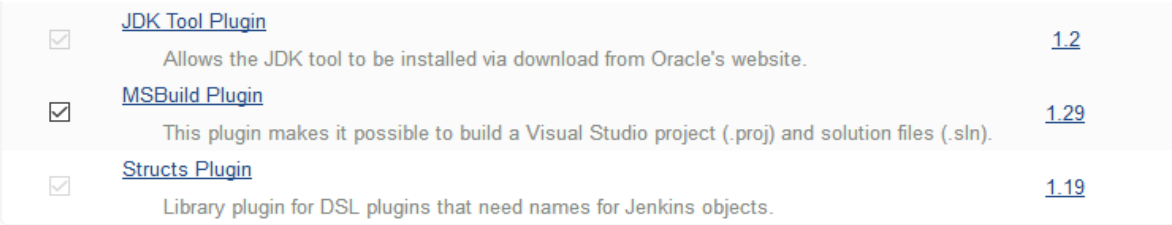
Task: - Configure MS Build project in Jenkins

Pre-requisite:-

- DotNet V4
- MStool
- Nuget Package

These are the software to be installed in the computer.

Step 1: - Install MS Build Plugin in Jenkins.

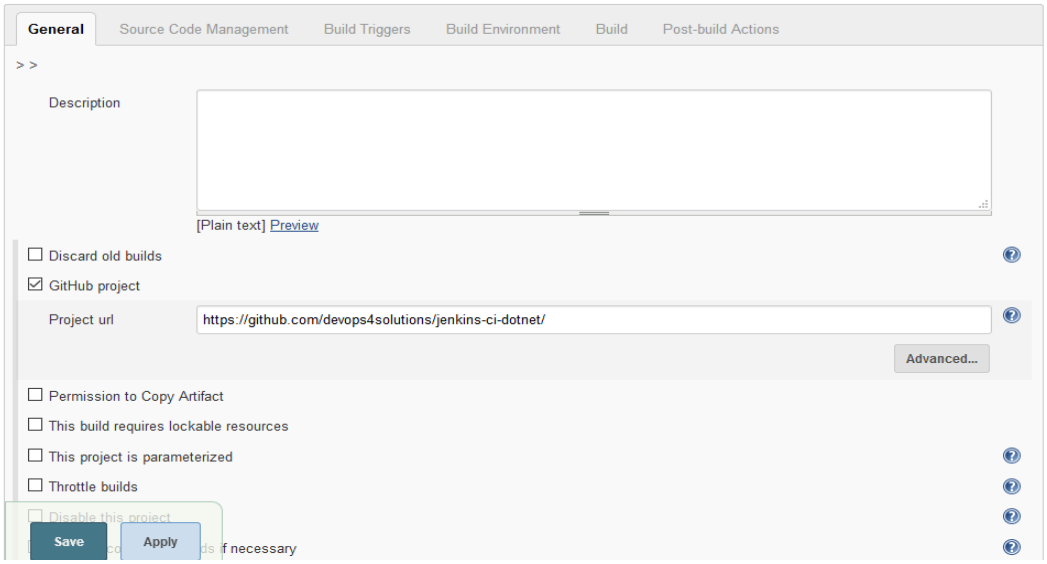


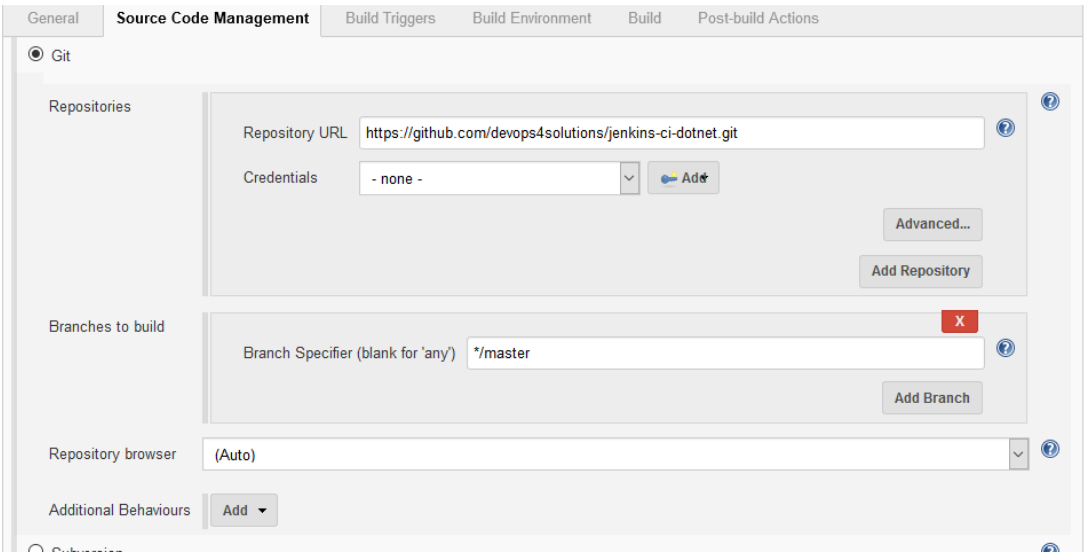
Step 2: - In Global Tools Configuration, click on MS Build.

Here, enter the Name of the plugin, Path to MS Build click Apply and Save.

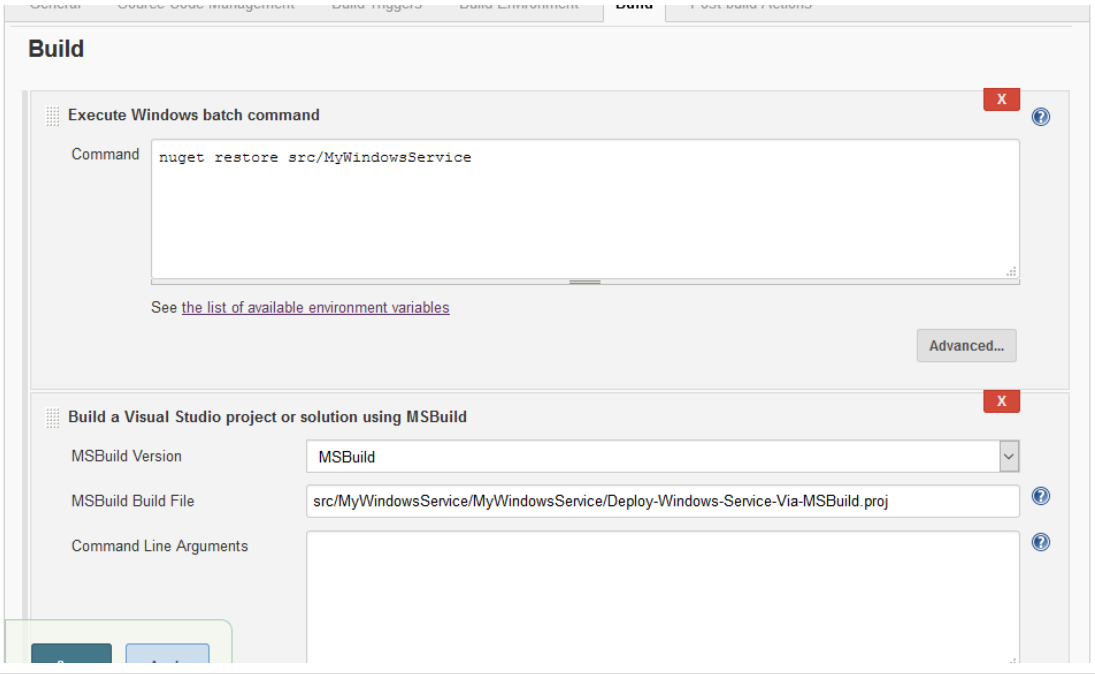
Step 3: - Create a free style project

Step 4: - Enter url of repo.





Step 5: - Execute windows batch command.



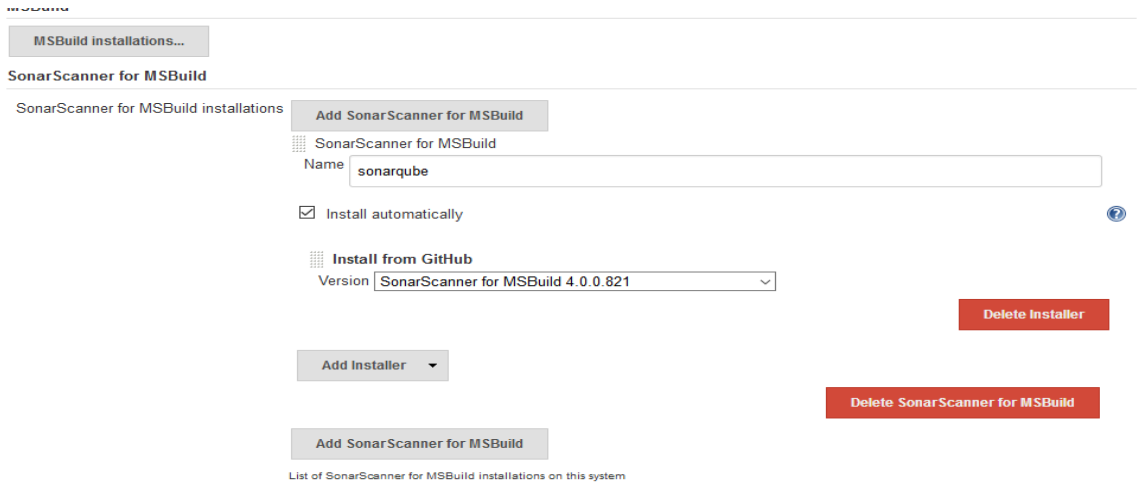
In Build, select Execute Windows batch command, enter the command, then select ‘Build a Visual Studio project or solution’ select the MS Build version, then enter the path for MS Build Build File.

Click Apply and Save.

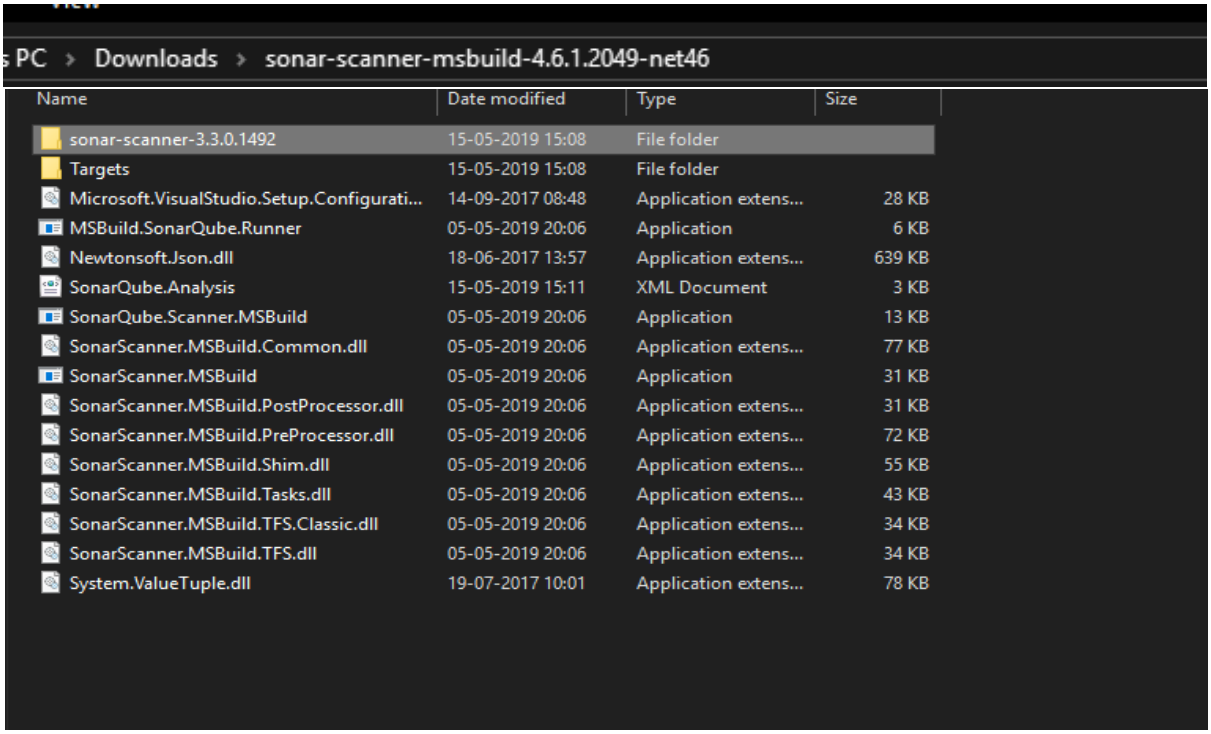
Step 6: - Build the job

- **Integrate SonarQube with MSBuild**

Step 1: - In Jenkins go to Global Tools configuration; go to Sonar Scanner for MS Build installations.



Step 2: - Enter the name, if you have downloaded selective version for sonar scanner for MSBuild then enter the path for that, for ex- In my pc I have downloaded

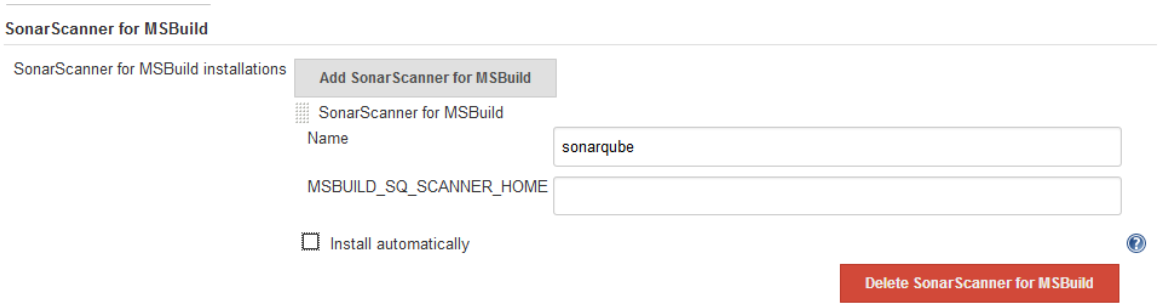


So I will give path for this folder as

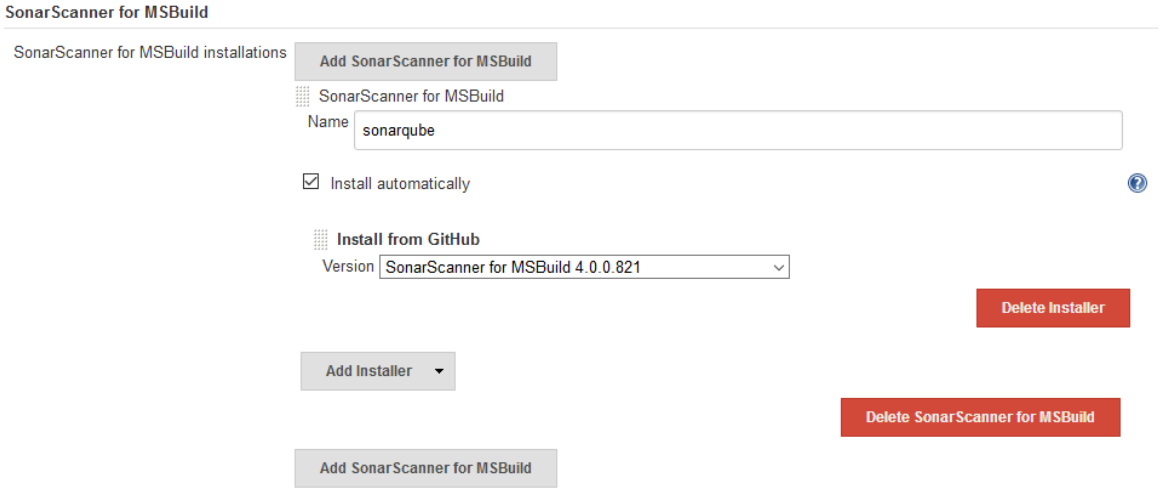
File Edit Format View Help

```
C:\Users\Administrator\Downloads\sonar-scanner-msbuild-4.6.1.2049-net46
```

Here,



Else, if you are not sure from where to download the scanner, then you can select Install automatically, followed by the version which you want to select.

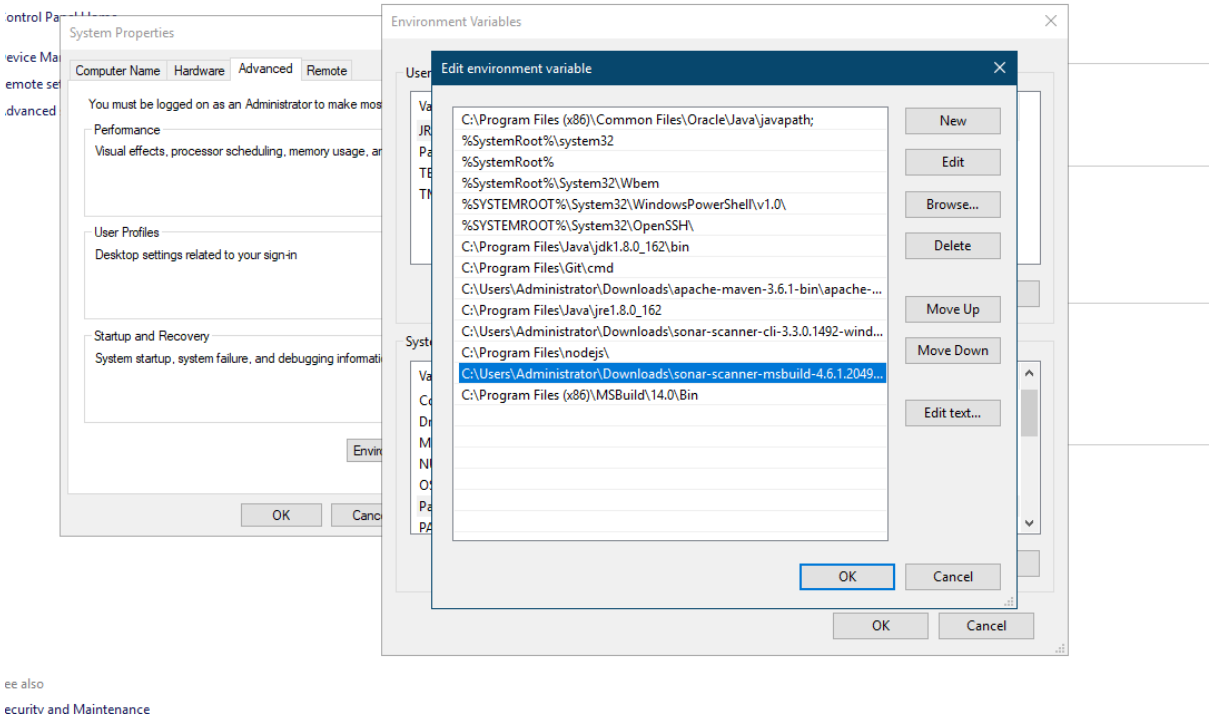


Click Apply and Save.

Step 3: - If you have downloaded the sonar scanner for MSBuild which is shown above, then give the path of the folder in Environment Variables in ThisPC properties,

File Edit Format View Help

```
C:\Users\Administrator\Downloads\sonar-scanner-msbuild-4.6.1.2049-net46
```



To integrate SonarQube with the project via powershell go to the folder of the project then go to the folder where .sln file is present, enter the following command,

```
SonarQube.Scanner.MSBuild.exe begin /k:"<Project name> " /d:sonar.host.url="<sonarQube URL>" /d:sonar.login=<SonarQube Token>"
'C:\Program Files (x86)\MSBuild\14.0\Bin\MSBuild.exe' src\MyWindowsService\MyWindowsService.sln /t:Rebuild
SonarQube.Scanner.MSBuild.exe end /d:sonar.login="SonarQube Token"
```

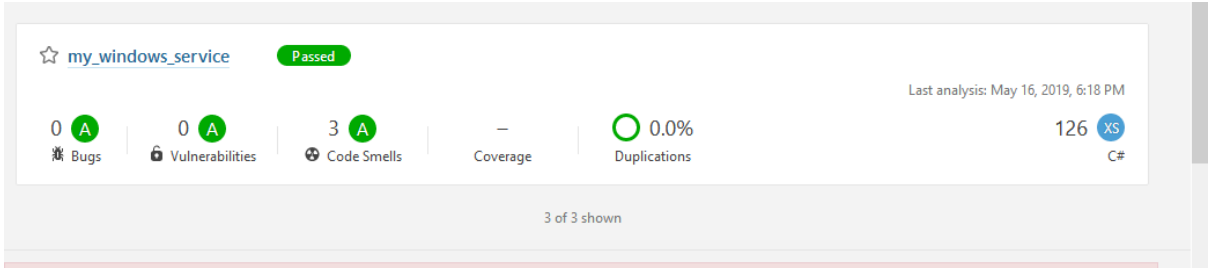
Note: - Generate SonarQube Token which is shown above how to do it.

```
PS C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService> SonarQube.Sc
anner.MSBuild.exe begin /k:"my_windows_service" /d:sonar.host.url="http://localhost:9000" /d:sonar.login="fba1fab4552e7e
43655fa5f564792399bd312176"
WARNING: ------------------------------------------------------------------------
This executable is deprecated and may be removed in next major version of the SonarScanner for MSBuild. Please use 'Sona
rScanner.MSBuild.exe' instead.
------------------------------------------------------------------------
SonarScanner for MSBuild 4.6.1
Using the .NET Framework version of the Scanner for MSBuild
Pre-processing started.
Preparing working directories...
18:13:51.455  Updating build integration targets...
18:13:51.722  Fetching analysis configuration settings...
18:13:57.426  Provisioning analyzer assemblies for cs...
18:13:57.44  Installing required Roslyn analyzers...
18:13:58.675  Pre-processing succeeded.
PS C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService>
```

```
PS C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService> & 'C:\Program Files (x86)\
MSBuild\14.0\Bin\MSBuild.exe' .\MyWindowsService.sln /t:Rebuild
Microsoft (R) Build Engine version 14.0.23107.0
Copyright (C) Microsoft Corporation. All rights reserved.

Building the projects in this solution one at a time. To enable parallel build, please add the "/m" switch.
Build started 16-05-2019 18:15:50.
Project "C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWind
owsService.sln" on node 1 (Rebuild target(s)).
ValidateSolutionConfiguration:
  Building solution configuration "Debug|Any CPU".
The target "RazorCoreCompile" listed in a BeforeTargets attribute at "C:\Users\Administrator\Downloads\jenkins-ci-dotne
t-master\jenkins-ci-dotnet-master\src\MyWindowsService\.sonarqube\bin\targets\SonarQube.Integration.targets (453,49)" d
oes not exist in the project, and will be ignored.
Project "C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWind
owsService.sln" (1) is building "C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src
\MyWindowsService\MyWindowsService\MyWindowsService.csproj" (2) on node 1 (Rebuild target(s)).
CoreClean:
  Deleting file "C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsServic
  e\MyWindowsService\bin\Debug\MyWindowsService.exe.config".
  Deleting file "C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsServic
  e\MyWindowsService\bin\Debug\MyWindowsService.exe".
  Deleting file "C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsServic
  e\MyWindowsService\bin\Debug\MyWindowsService.pdb".
  Deleting file "C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsServic
  e\MyWindowsService\bin\Debug\Common.Logging.Core.dll".
  Deleting file "C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsServic
  e\MyWindowsService\bin\Debug\Common.Logging.dll".
```

```
ce.sln" (Rebuild target) (1) ->
"C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWindowsServi
ce\MyWindowsService.csproj" (Rebuild target) (2) ->
(ResolveAssemblyReferences target) ->
  C:\Program Files (x86)\MSBuild\14.0\bin\Microsoft.Common.CurrentVersion.targets(1819,5): warning MSB3270: There was a
 mismatch between the processor architecture of the project being built "MSIL" and the processor architecture of the re
ference "C:\Windows\Microsoft.NET\Framework\v4.0.30319\\mscorlib.dll", "x86". This mismatch may cause runtime failures.
 Please consider changing the targeted processor architecture of your project through the Configuration Manager so as t
o align the processor architectures between your project and references, or take a dependency on references with a proc
essor architecture that matches the targeted processor architecture of your project. [C:\Users\Administrator\Downloads\
jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWindowsService\MyWindowsService.csproj]
  C:\Program Files (x86)\MSBuild\14.0\bin\Microsoft.Common.CurrentVersion.targets(1819,5): warning MSB3270: There was a
 mismatch between the processor architecture of the project being built "MSIL" and the processor architecture of the re
ference "System.Data", "AMD64". This mismatch may cause runtime failures. Please consider changing the targeted process
or architecture of your project through the Configuration Manager so as to align the processor architectures between yo
ur project and references, or take a dependency on references with a processor architecture that matches the targeted p
rocessor architecture of your project. [C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-mas
ter\src\MyWindowsService\MyWindowsService\MyWindowsService.csproj]


"C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWindowsServi
ce.sln" (Rebuild target) (1) ->
"C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWindowsServi
ce\MyWindowsService.csproj" (Rebuild target) (2) ->
(CoreCompile target) ->
  Service1.cs(34,17): warning S3241: Change return type to 'void'; not a single caller uses the returned value. [C:\Use
rs\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWindowsService\MyWi
ndowsService.csproj]
  Service1.cs(38,37): warning S1075: Refactor your code not to use hardcoded absolute paths or URIs. [C:\Users\Administ
rator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWindowsService\MyWindowsServic
e.csproj]
  Service1.cs(61,59): warning S1075: Refactor your code not to use hardcoded absolute paths or URIs. [C:\Users\Administ
rator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWindowsService\MyWindowsServic
e.csproj]

    6 Warning(s)
    0 Error(s)

Time Elapsed 00:00:36.53
PS C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService>
```

```
PS C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService> SonarQube.Scanner.MSBuild.
exe end /d:sonar.login="fba1fab4552e7e43655fa5f564792399bd312176"
WARNING: ------------------------------------------------------------------------
This executable is deprecated and may be removed in next major version of the SonarScanner for MSBuild. Please use 'SonarScanner.MSBui
ld.exe' instead.
------------------------------------------------------------------------
SonarScanner for MSBuild 4.6.1
Using the .NET Framework version of the Scanner for MSBuild
Post-processing started.
Fixed invalid Code Analysis ErrorLog file. Please check that VS 2015 Update 1 (or later) is installed. Fixed file: C:\Users\Administra
tor\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWindowsService\bin\Debug\MyWindowsService.exe.R
oslynCA_fixed.json
Fixed invalid Code Analysis ErrorLog file. Please check that VS 2015 Update 1 (or later) is installed. Fixed file: C:\Users\Administra
tor\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindowsService\MyWindowsService\bin\Debug\MyWindowsService.exe.R
oslynCA_fixed.json
Calling the SonarQube Scanner...
INFO: Scanner configuration file: C:\Users\Administrator\Downloads\sonar-scanner-msbuild-4.6.1.2049-net46\sonar-scanner-3.3.0.1492\bin
\..\conf\sonar-scanner.properties
INFO: Project root configuration file: C:\Users\Administrator\Downloads\jenkins-ci-dotnet-master\jenkins-ci-dotnet-master\src\MyWindow
sService\.sonarqube\out\sonar-project.properties
INFO: SonarQube Scanner 3.3.0.1492
INFO: Java 1.8.0_162 Oracle Corporation (64-bit)
INFO: Windows Server 2016 10.0 amd64
INFO: User cache: C:\Users\Administrator\.sonar\cache
INFO: SonarQube server 6.7.7
INFO: Default locale: "en_IN", source code encoding: "windows-1252" (analysis is platform dependent)
INFO: Publish mode
INFO: Load global settings
INFO: Load global settings (done) | time=703ms
INFO: Server id: BF41A1F2-AWq1O9wbBYTtm9wIuX_6
INFO: User cache: C:\Users\Administrator\.sonar\cache
INFO: Load plugins index
```



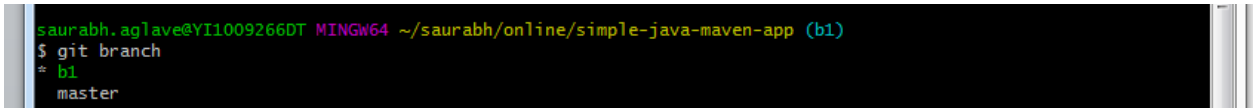
POC- 1: Need to configure 2 pipelines jobs for Linux box maven project

Learn how to create a Continuous Integration (CI) pipeline for a GitHub project using the Jenkins Demo Lab on AWS Cloud Jump Start. Once you've completed the steps in this guide, any changes made to your GitHub project code will be automatically and instantly built and tested by Jenkins, with appropriate success/failure indications as needed.

Repository: - https://github.com/jenkins-docs/simple-java-maven-app

1. Login to Git bash, clone the repo as mentioned and create the new branch then checkout to that branch make some changes add it to the staging area commit the changes.
   Cloning the repository:

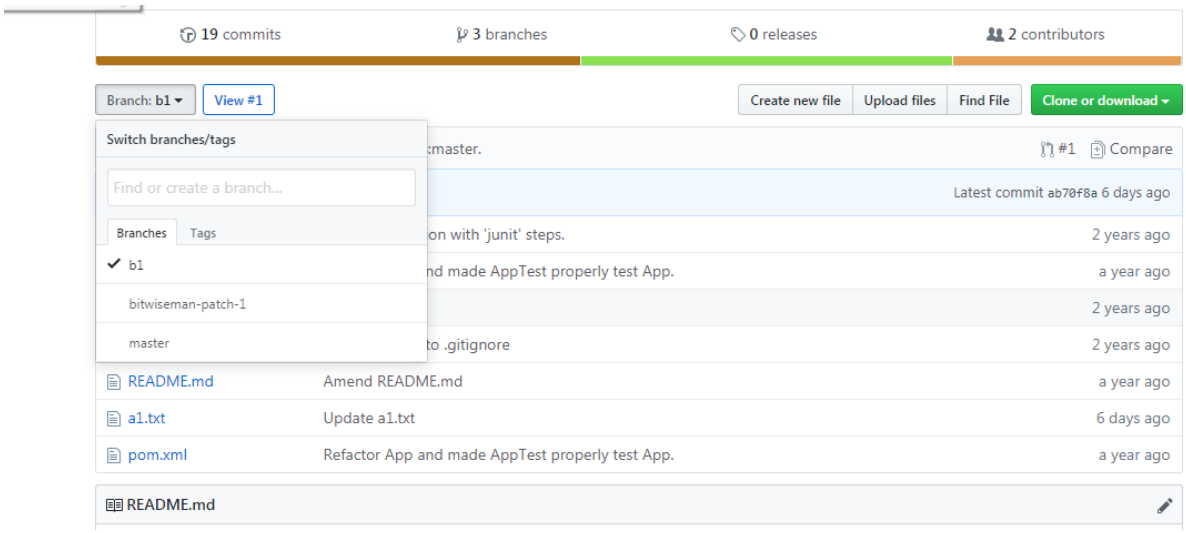
- Creating branch : git branch b1
- To see list of branches : git branch -a



- Checkout to that branch name : git checkout b1
- Create one file and add to to staging area using command git add . do git commit –m " msg " after it.
- And push the branch from public repo to remote repo .. (git push origin "branch name to be pushed ")

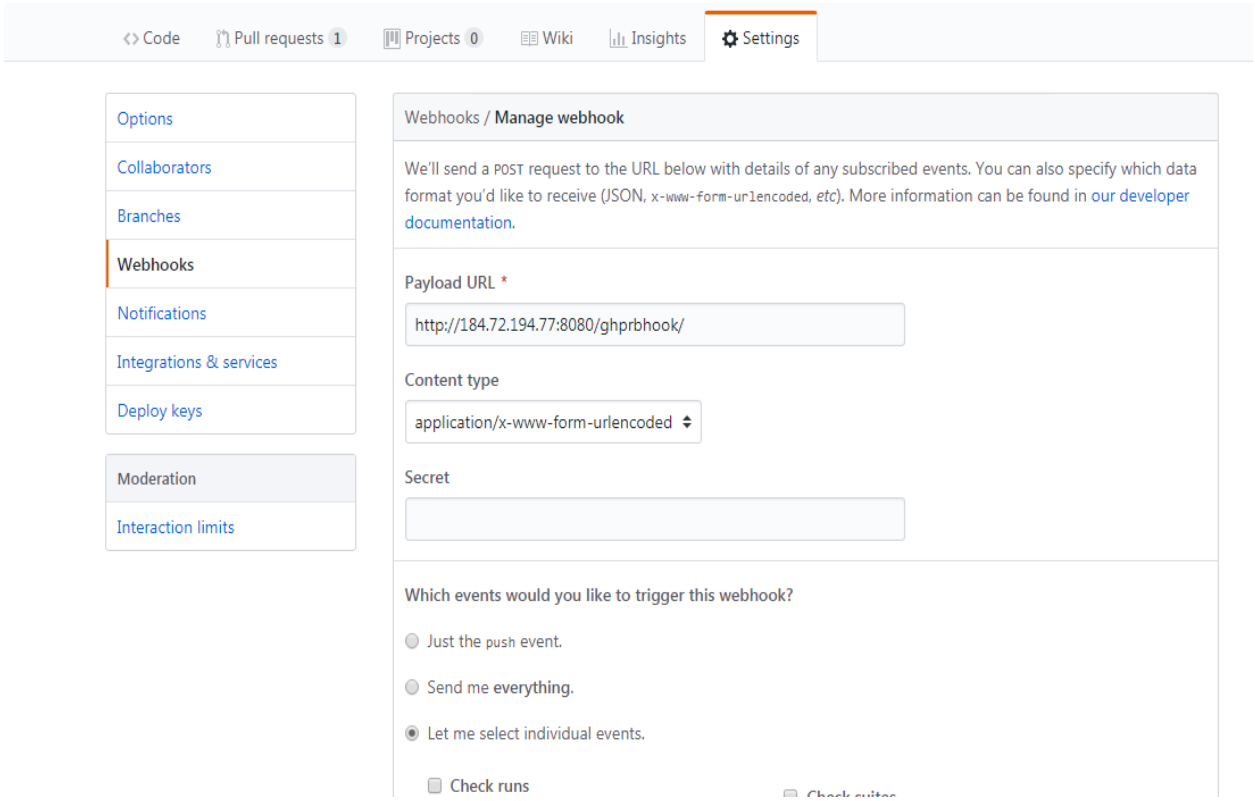
```
389  git clone "https://github.com/yashsaurabh/simple-java-maven-app"
390  ls
391  cd simple-java-maven-app/
392  ls
393  git branch b1
394  git checkout b1
395  touch a1.txt
396  echo "hiii hello" > a1.txt
397  git add .
398  git commit -m "commit"
399  git pull origin master
400  git push origin b1
```

Go to GitHub account and there check whether if new branches are successfully added to remote repository:

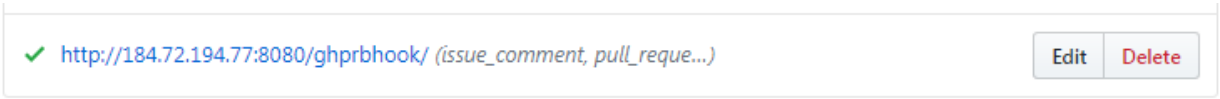


Next step is adding webhook for push and pull events:

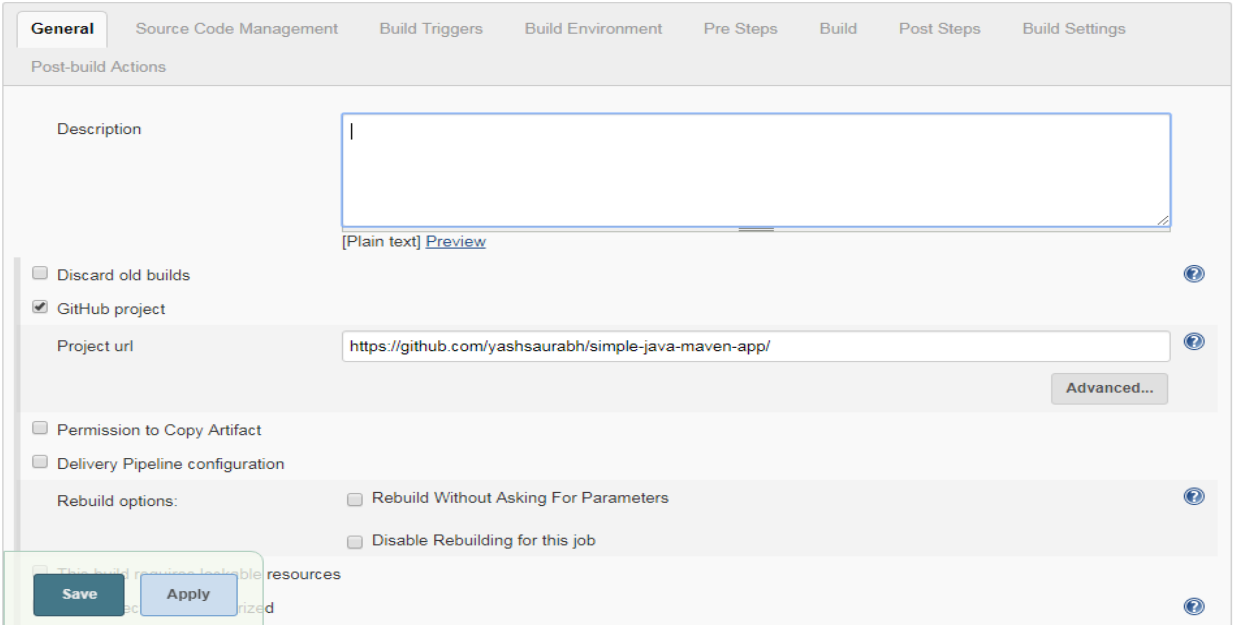
2. Go to project setting → select web hook option:

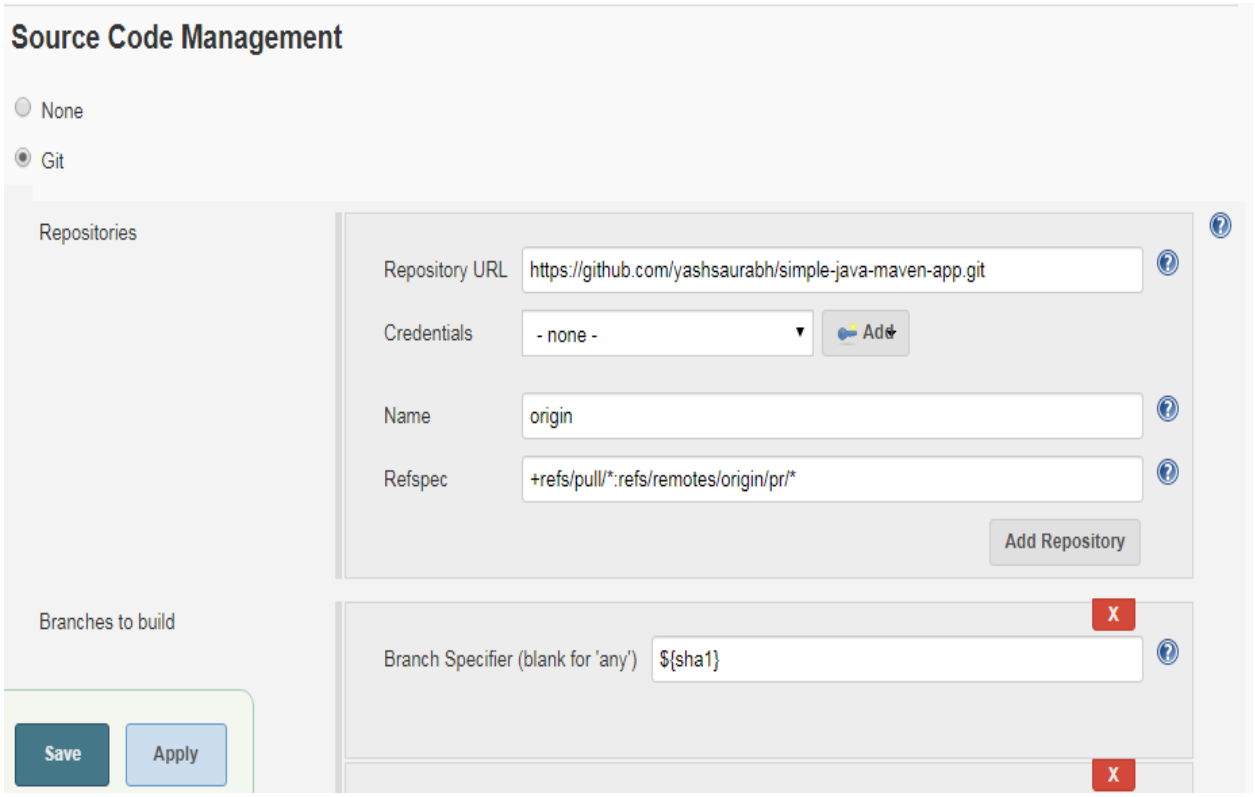


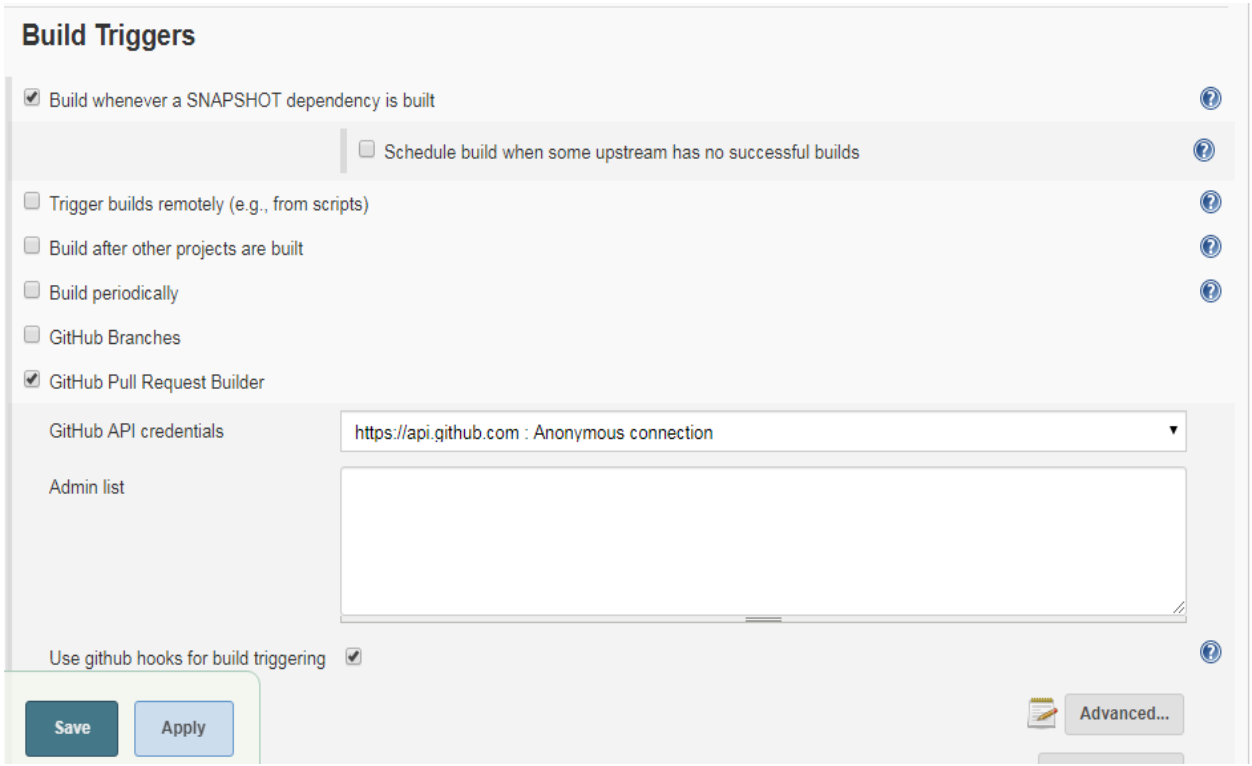
Once webhook is created it will look llike this:

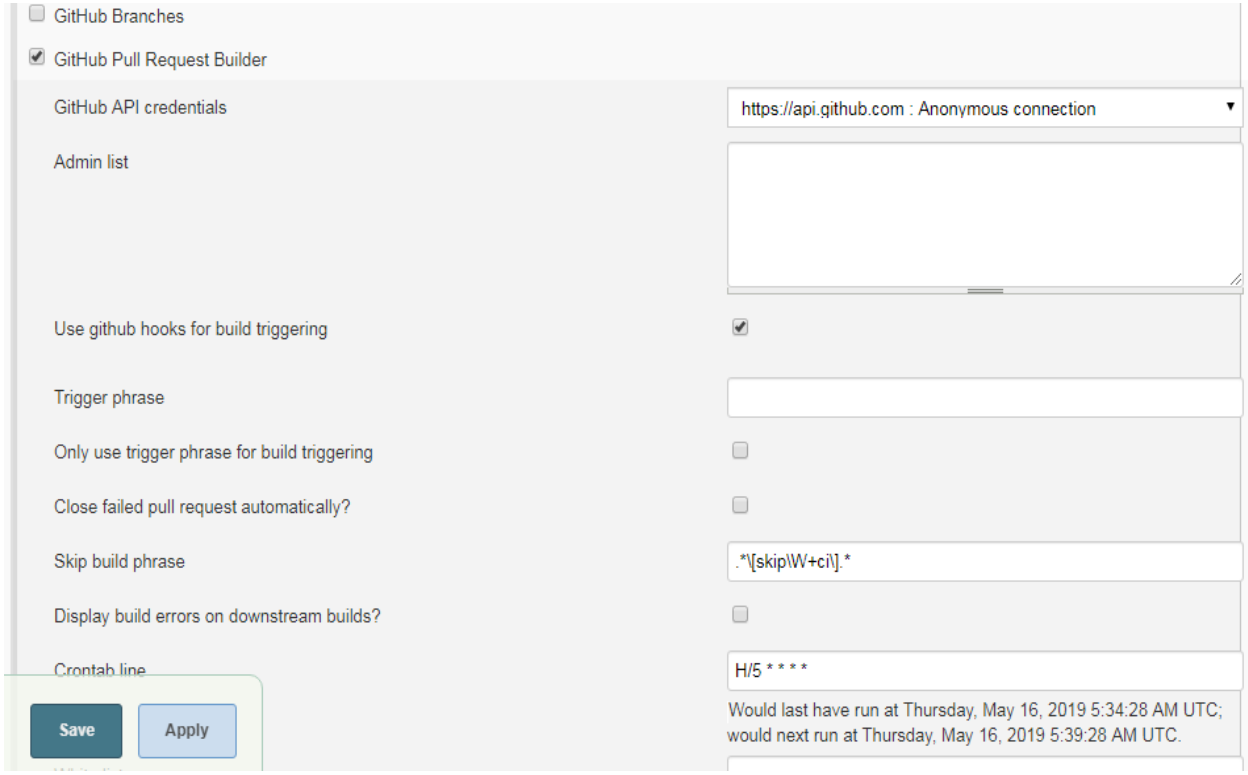


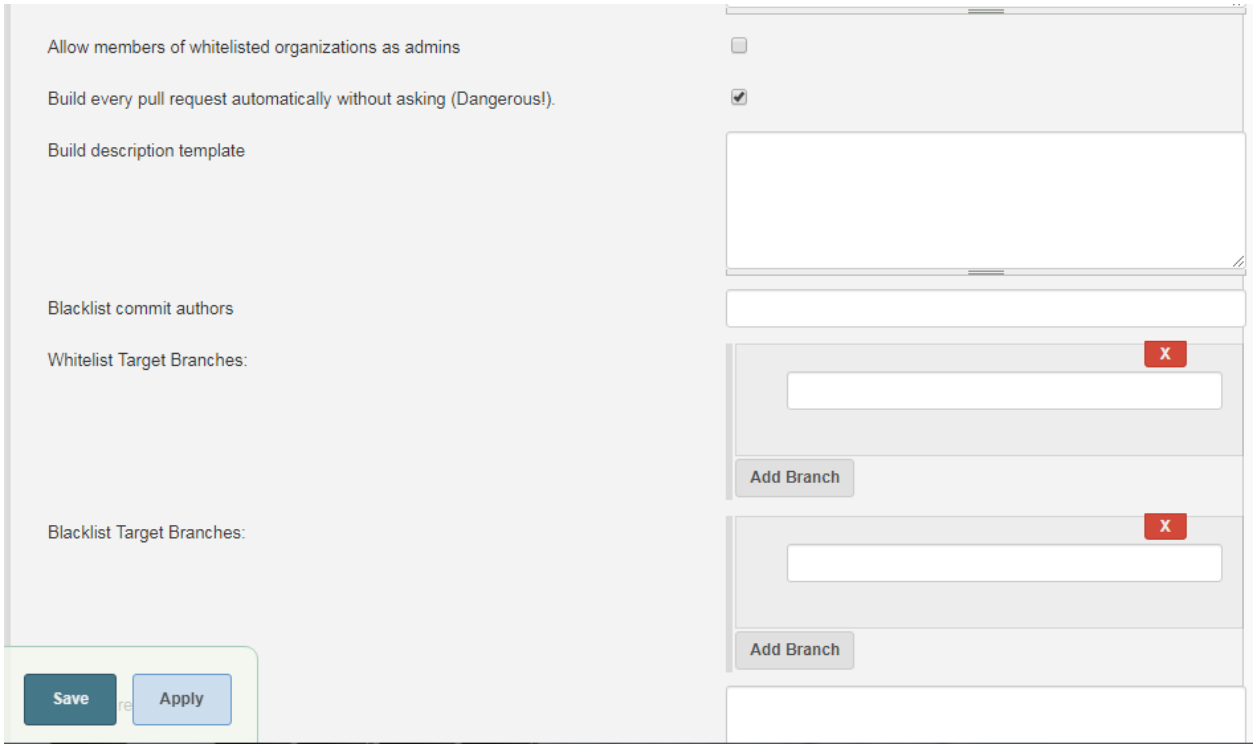
3. Go to Jenkins add new item → select maven projects → configure the project as per given screenshot:

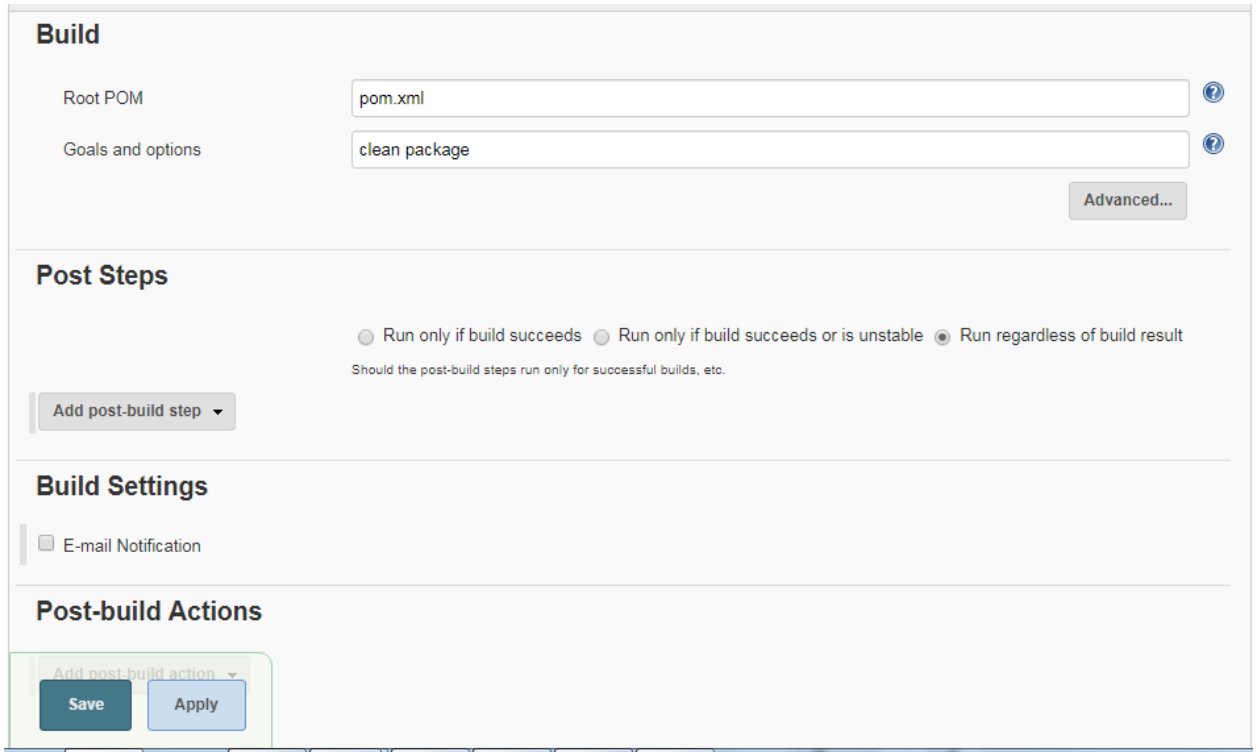



Source Code Management
None
Git
Repositories
Repository URL
https://github.com/yashsaurabh/simple-java-maven-app.git
Credentials
- none -
Add
Name
origin
Refspec
+refs/pull/*:refs/remotes/origin/pr/*
Add Repository
Branches to build
Branch Specifier (blank for 'any')
${sha1}
Save
Apply

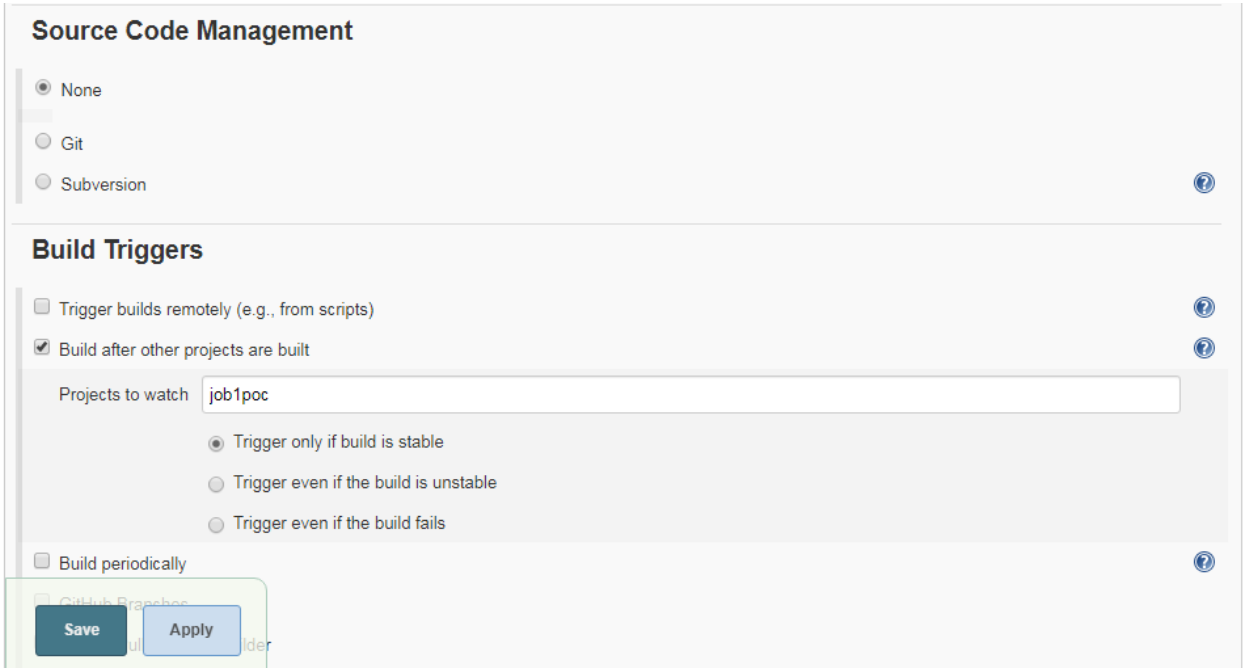

Build Triggers
Build whenever a SNAPSHOT dependency is built
Schedule build when some upstream has no successful builds
Trigger builds remotely (e.g., from scripts)
Build after other projects are built
Build periodically
GitHub Branches
GitHub Pull Request Builder
GitHub API credentials
https://api.github.com : Anonymous connection
Admin list
Use github hooks for build triggering
Save
Apply
Advanced...

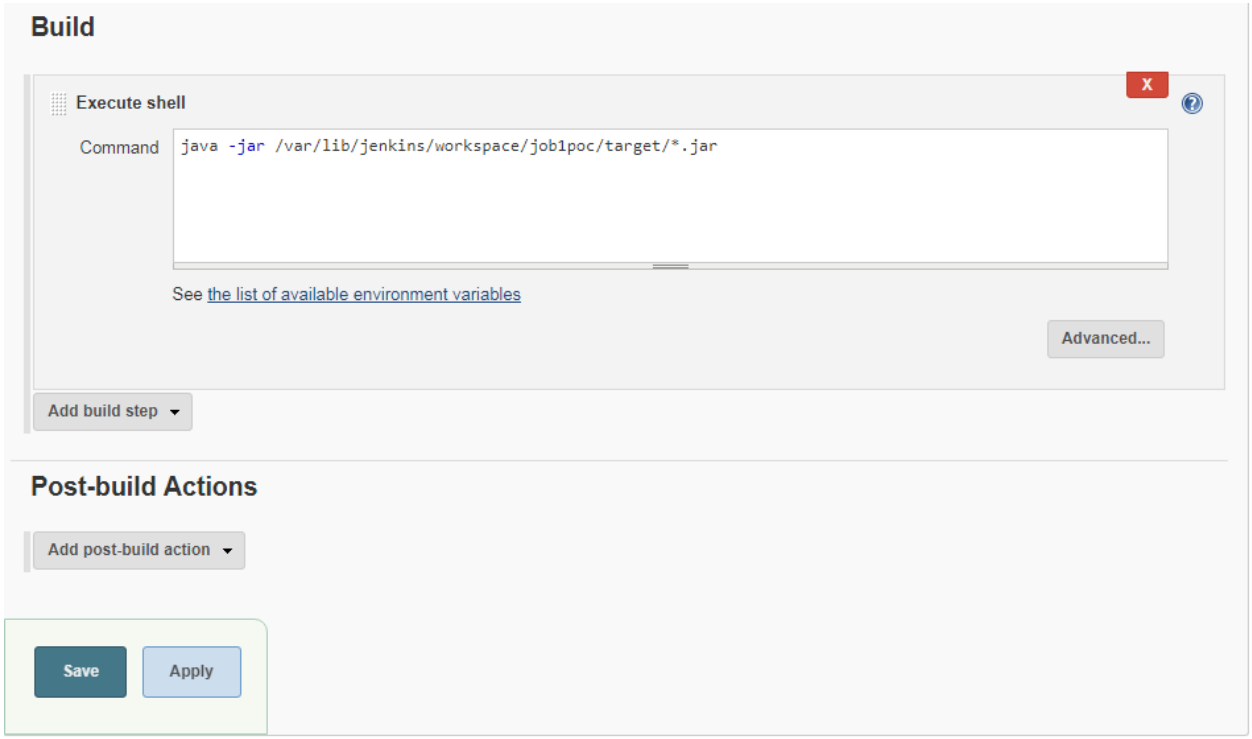

GitHub Branches
GitHub Pull Request Builder
GitHub API credentials
https://api.github.com : Anonymous connection
Admin list
Use github hooks for build triggering
Trigger phrase
Only use trigger phrase for build triggering
Close failed pull request automatically?
Skip build phrase
.*\[skip\W+ci\].*
Display build errors on downstream builds?
Crontab line
H/5 * * * *
Would last have run at Thursday, May 16, 2019 5:34:28 AM UTC; would next run at Thursday, May 16, 2019 5:39:28 AM UTC.
Save
Apply


Allow members of whitelisted organizations as admins
Build every pull request automatically without asking (Dangerous!).
Build description template
Blacklist commit authors
Whitelist Target Branches:
Add Branch
Blacklist Target Branches:
Add Branch
Save
Apply



4. Now configure second job named job2poc take free style project as we want to build pipeline & make the configuration changes:

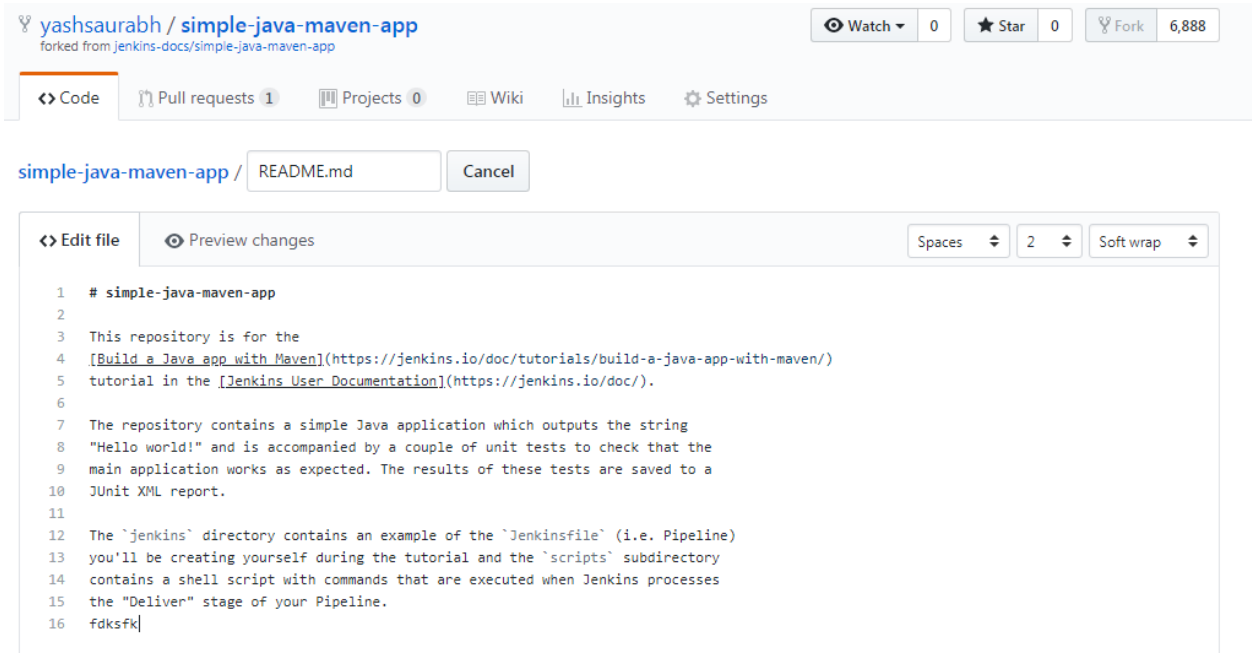


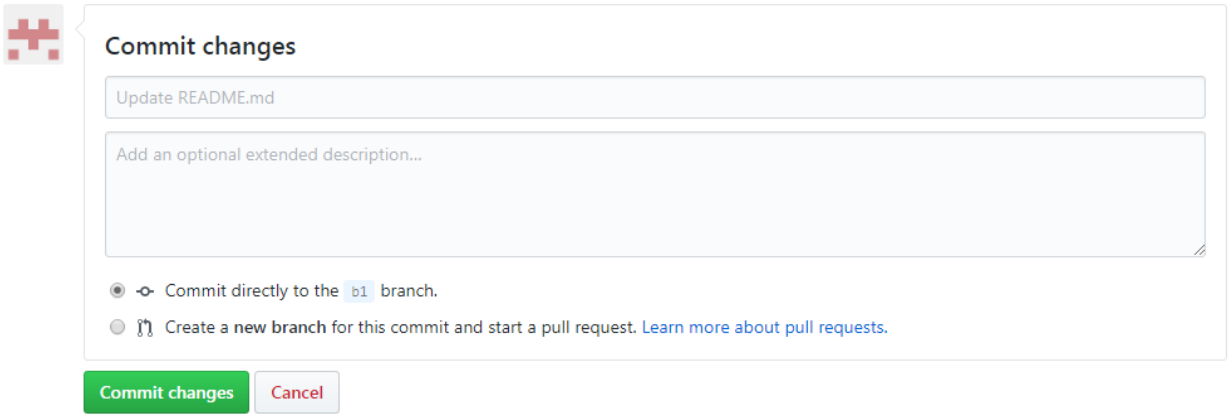


NOTE> jar can't upload on web only war can be uploaded so here in Execute Shell →we write command which will execute and after build project shows output on console.

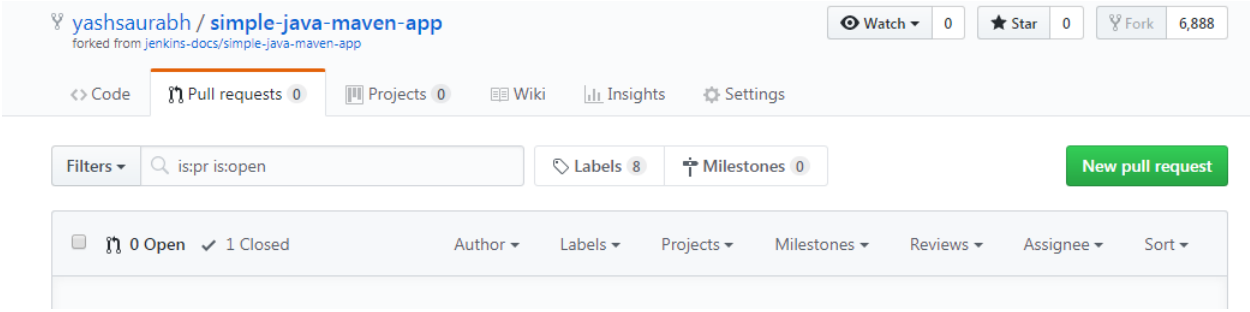
- Save the project now go to github.com → go to your project repo to see create pull request

5. Go to Github Account→Commit some changes in any file any other than master branch:

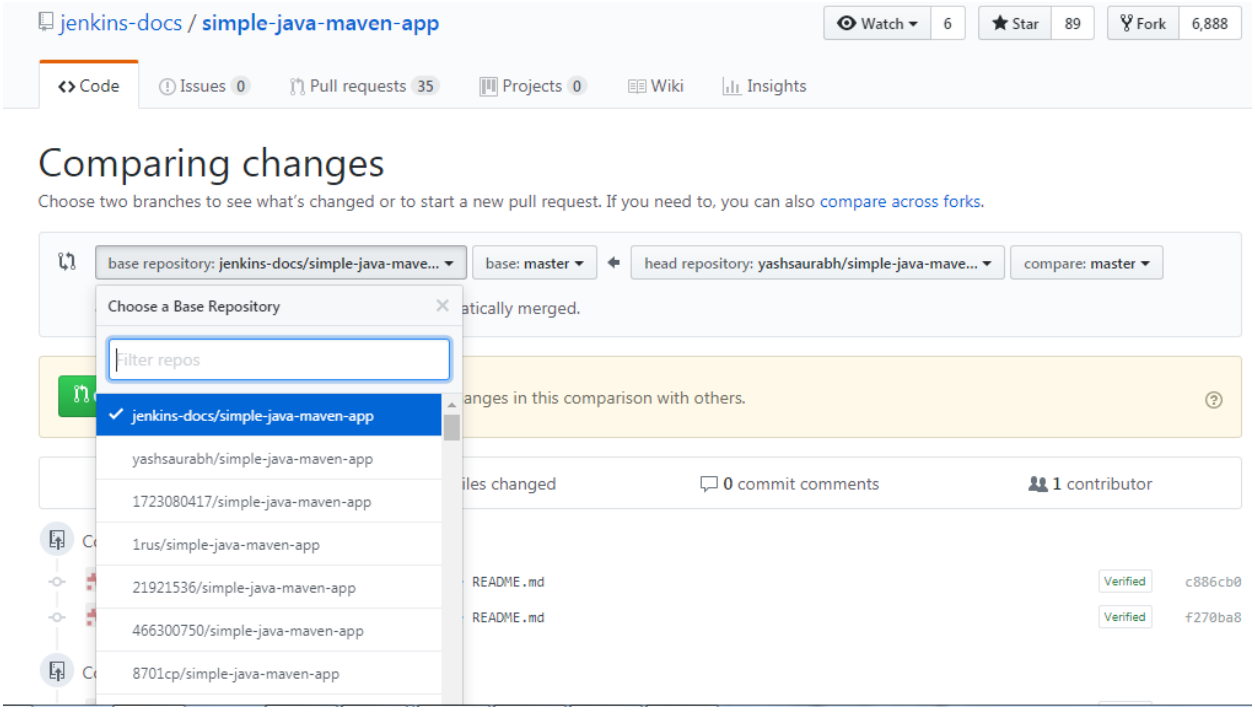




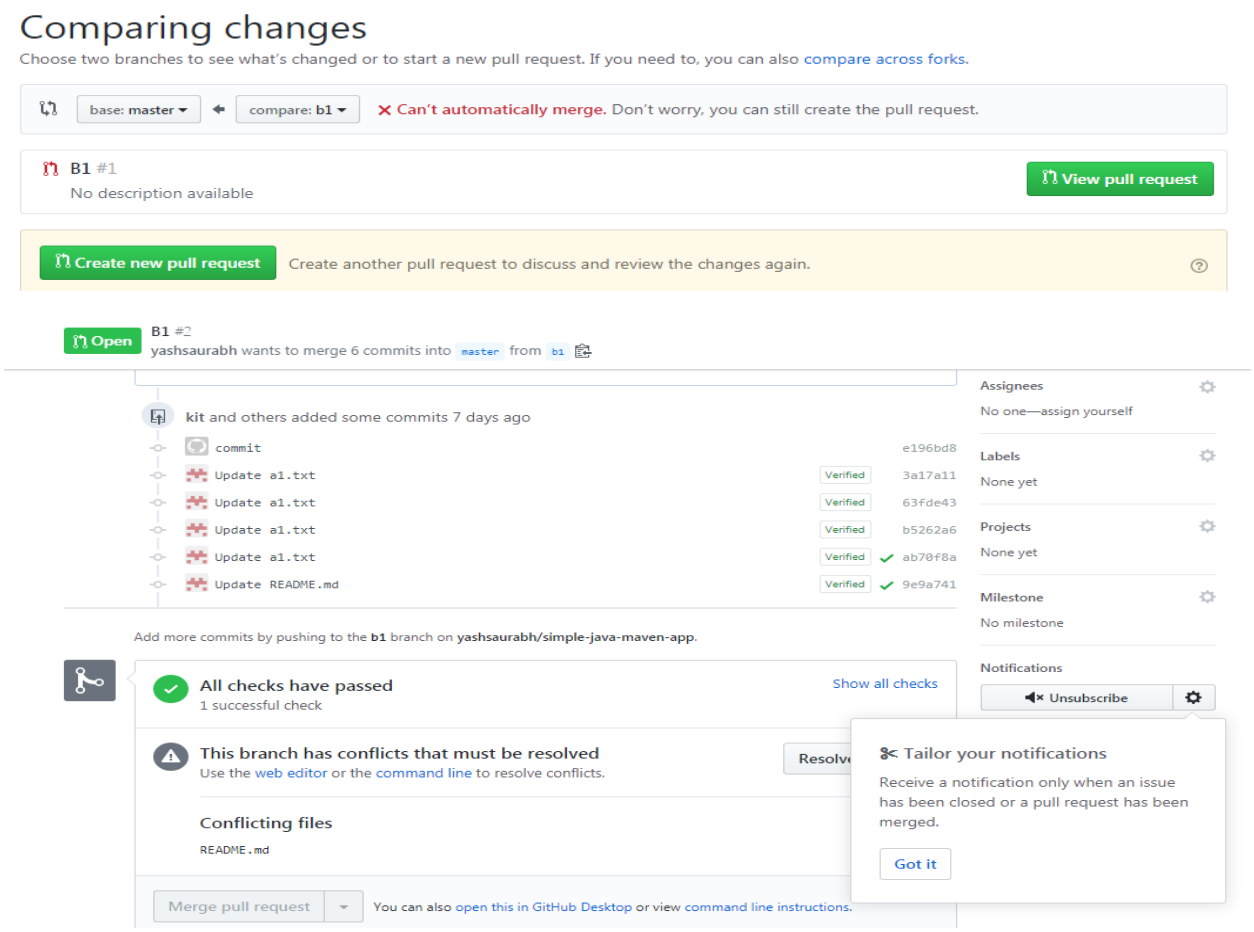
6. Go to pull request tab and check whether there is any pull request is opened if any then close it first.
7. And then got to new pull request tab:



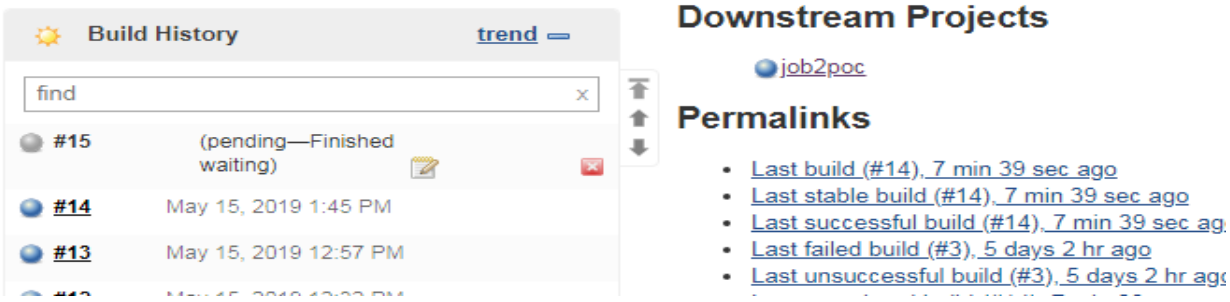
- Select your project repo and select branch inside which you have made the changes:



- Go to view pull request tab:



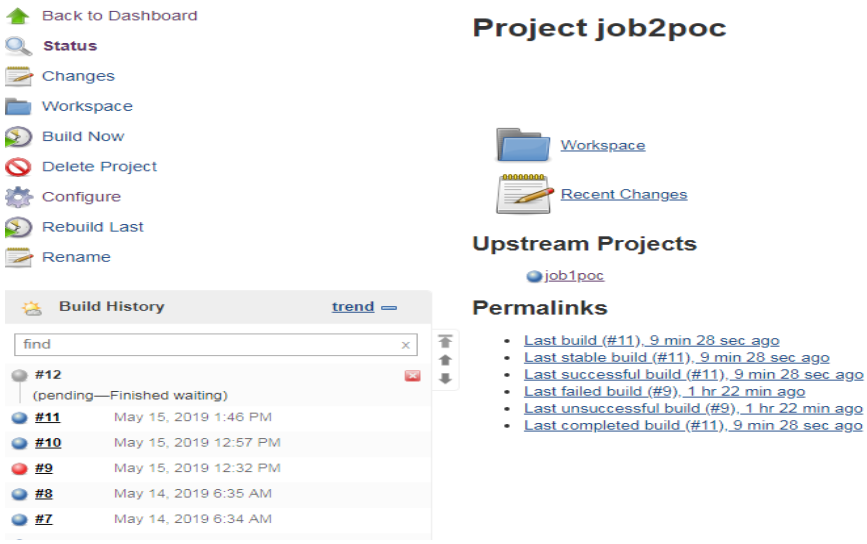
- Go to Jenkins jobs console output of job1poc:
- After build of first job automatically it will create http://IP address of Jenkins: 8080/ghprbhook/ created.
- How to create webhook for PR manually?
- Go to → webhook from setting
- Click on webhook→Add webhook
- Enter IP address on which Jenkins running/ghprhook/ (http:// IP address of Jenkins: 8080 /ghprbhook/)
- Select –Let me select individual event→select issue comment and pull requests
- Update webhook



The output of job1poc will look like as below:

```
 T E S T S
-------------------------------------------------------
Running com.mycompany.app.AppTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.12 sec

Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

[JENKINS] Recording test results
[WARNING] Attempt to (de-)serialize anonymous class hudson.maven.reporters.SurefireArchiver$2; see:
https://jenkins.io/redirect/serialization-of-anonymous-classes/
[WARNING] Attempt to (de-)serialize anonymous class hudson.maven.reporters.BuildInfoRecorder$1; see:
https://jenkins.io/redirect/serialization-of-anonymous-classes/
[INFO]
[INFO] --- maven-jar-plugin:3.0.2:jar (default-jar) @ my-app ---
[INFO] Building jar: /var/lib/jenkins/workspace/job1poc/target/my-app-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 8.619 s
[INFO] Finished at: 2019-05-15T13:55:24Z
[INFO] Final Memory: 17M/93M
[INFO] ------------------------------------------------------------------------
[JENKINS] Archiving /var/lib/jenkins/workspace/job1poc/pom.xml to com.mycompany.app/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.pom
[JENKINS] Archiving /var/lib/jenkins/workspace/job1poc/target/my-app-1.0-SNAPSHOT.jar to com.mycompany.app/my-app/1.0-SNAPSHOT/my-app-
1.0-SNAPSHOT.jar
channel stopped
Setting status of 9e9a741edc441356846d8c37ff0745b79fd8aa19 to SUCCESS with url http://184.72.194.77:8080/job/job1poc/15/ and message:
'Build finished. '
Triggering a new build of job2poc
Finished: SUCCESS
```

- Now go to job2poc to see console output:



Console Output

```
Started by upstream project "job1poc" build number 15
originally caused by:
 GitHub pull request #2 of commit 9e9a741edc441356846d8c37ff0745b79fd8aa19, has merge conflicts.
Running as SYSTEM
Building on master in workspace /var/lib/jenkins/workspace/job2poc
[job2poc] $ /bin/sh -xe /tmp/jenkins6116969897010231820.sh
+ java -jar /var/lib/jenkins/workspace/job1poc/target/my-app-1.0-SNAPSHOT.jar
Hello World!
Finished: SUCCESS
```

- Hence the project is successfully build through pipeline

POC- 2: Installation of SonarQube integrate with Jenkins deploy on Tomcat8x using Build Pipeline

1. **For installation SonarQube require 2GB memory**
2. **Install SonarQube version 6.7.7 on slave machine**
3. **Make sure that java-1.8.0 installed**
4. **Go through root**

- wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-6.7.7.zip
- unzip sonarqube-6.7.7.zip
- useradd sonar
- passwd sonar
- cd /opt/sonarqube/sonarqube-6.7.7
- chown -R sonar:sonar *
- cd conf/
- vim sonar.properties

- Make changes as a change memory 1024 and 512

```
# WEB SERVER
# Web server is executed in a dedicated Java process. By default heap size is 512
Mb.
# Use the following property to customize JVM options.
#    Recommendations:
#
#    The HotSpot Server VM is recommended. The property -server should be added i
f server mode
#    is not enabled by default on your environment:
#    http://docs.oracle.com/javase/8/docs/technotes/guides/vm/server-class.html
#
#    Startup can be long if entropy source is short of entropy. Adding
#    -Djava.security.egd=file:/dev/./urandom is an option to resolve the problem.
#    See https://wiki.apache.org/tomcat/HowTo/FasterStartUp#Entropy_Source
#
#sonar.web.javaOpts=-Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError
sonar.web.javaOpts=-Xmx1024m -Xms512m -XX:+HeapDumpOnOutOfMemoryError
```

```
# COMPUTE ENGINE
# The Compute Engine is responsible for processing background tasks.
# Compute Engine is executed in a dedicated Java process. Default heap size is 51
2Mb.
# Use the following property to customize JVM options.
#    Recommendations:
#
#    The HotSpot Server VM is recommended. The property -server should be added i
f server mode
#    is not enabled by default on your environment:
#    http://docs.oracle.com/javase/8/docs/technotes/guides/vm/server-class.html
#
#sonar.ce.javaOpts=-Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError
sonar.ce.javaOpts=-Xmx1024m -Xms512m -XX:+HeapDumpOnOutOfMemoryError
```

- cd /opt/sonarqube/sonarqube-6.7.7/bin/linux-x86-64/
- vim sonar.sh
- Make changes
- RUN_AS_USER=sonar

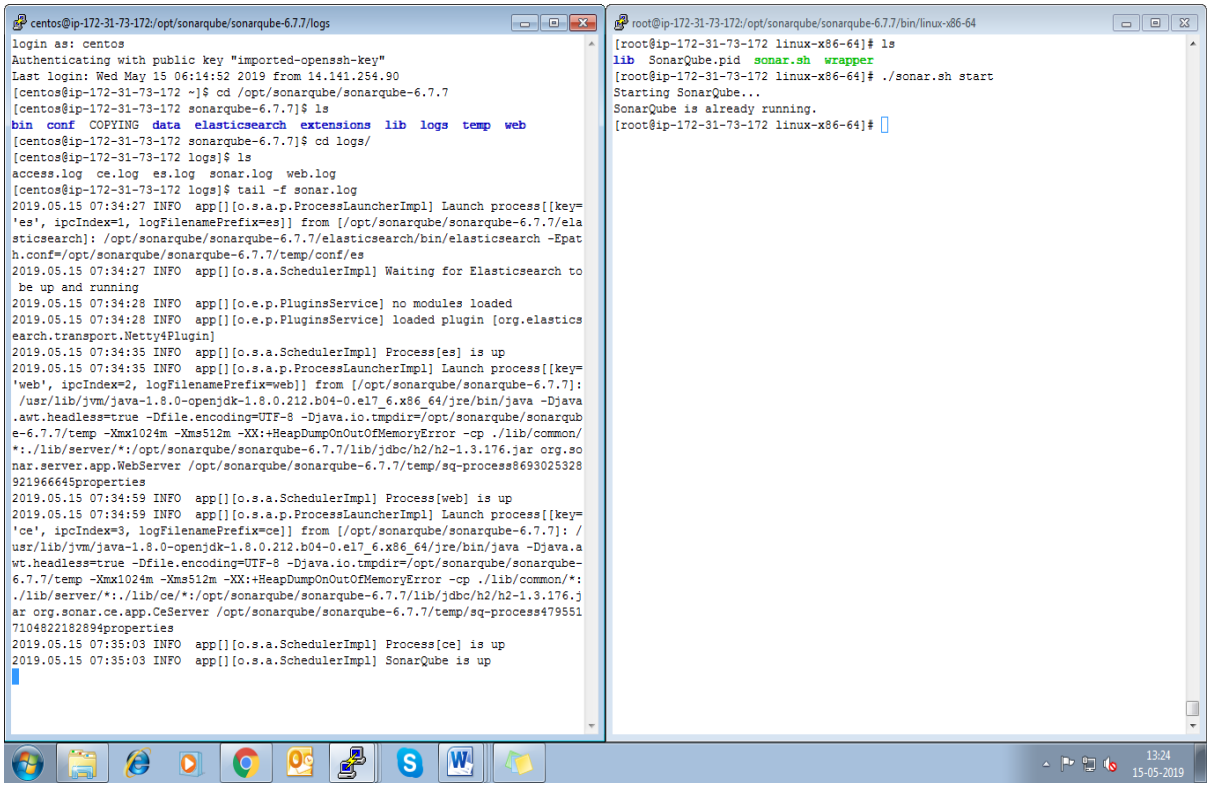
```
#  file will cause the wrapper to exit without any way to write out an erro
#  message.
# NOTE - This will set the user which is used to run the Wrapper as well as
#  the JVM and is not useful in situations where a privileged resource or
#  port needs to be allocated prior to the user being changed.
RUN_AS_USER=sonar
```

- Note—Before starting sonar first check whether sonarqube has given permission as a sonar

- cd /opt/sonarqube/sonarqube-6.7.7/bin/ /linux-x86-64/
- ./sonar.sh start

```
[root@ip-172-31-73-172 linux-x86-64]# ls
lib  SonarQube.pid  sonar.sh  wrapper
[root@ip-172-31-73-172 linux-x86-64]# ./sonar.sh start
Starting SonarQube...
SonarQube is already running.
[root@ip-172-31-73-172 linux-x86-64]#
```

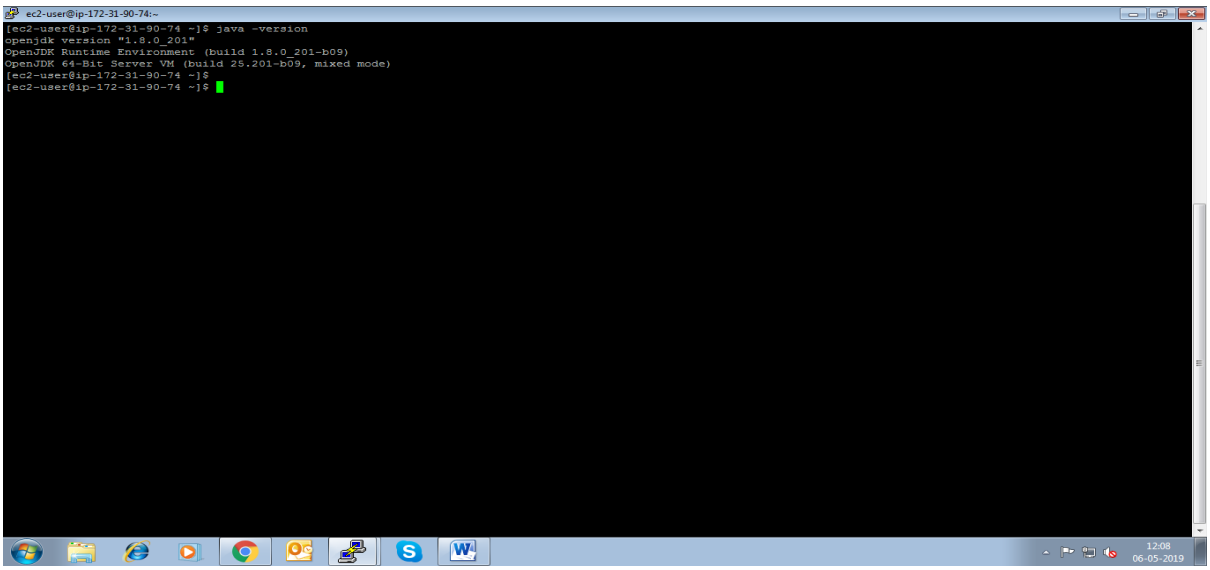
- Go to logs/sonar.logs –It shows sonar is up and running



- **Preparing Your Environment for Jenkins on master machine**

1. Installing Java

- yum install java-1.8.0*
- update-alternatives --config java
- java –version



2. Installing Git

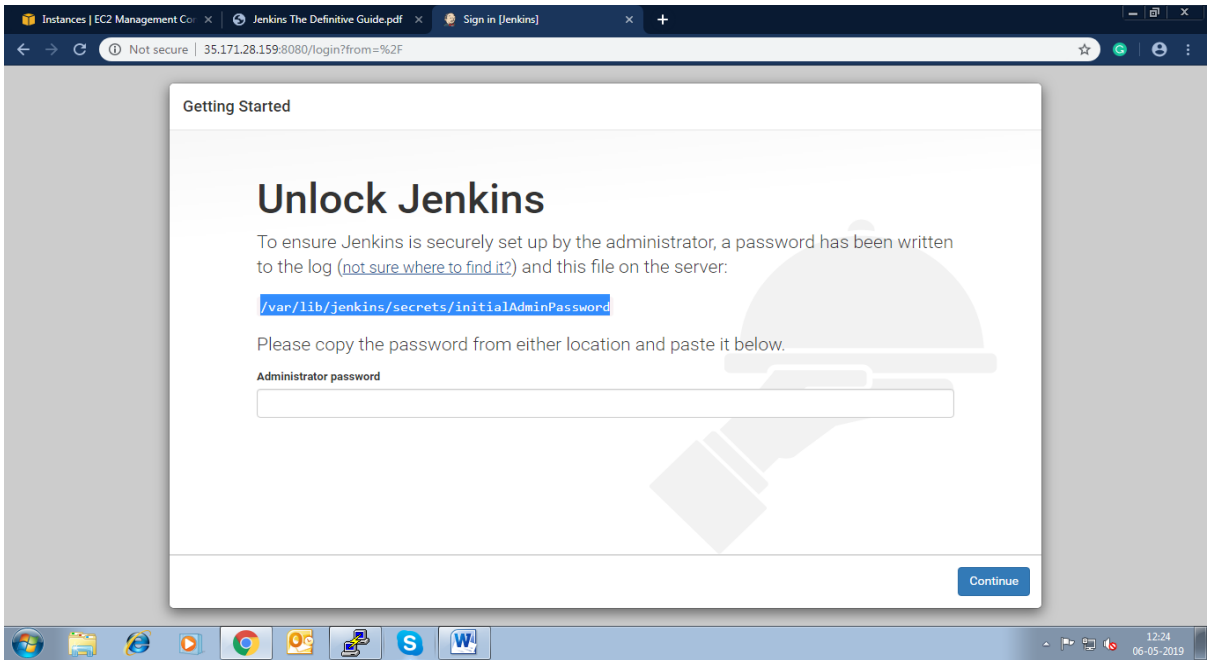
- sudo yum install git

3. Installing Jenkins

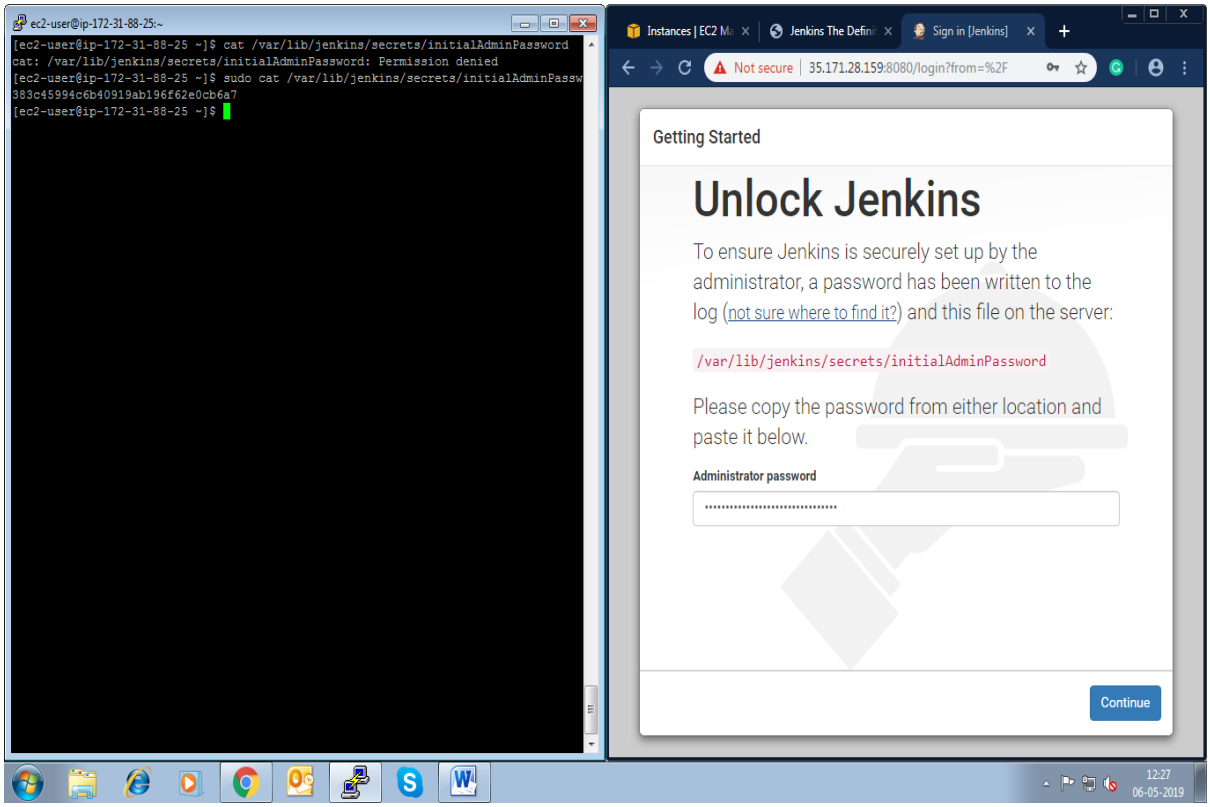
- sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
- sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
- sudo yum install jenkins
- sudo service jenkins start https://www.digitalocean.com/community/tutorials/how-to-set-up-jenkins-for-continuous-development-integration-on-centos-7)
- you can access Jenkins in your web browser on http://localhost: 8080

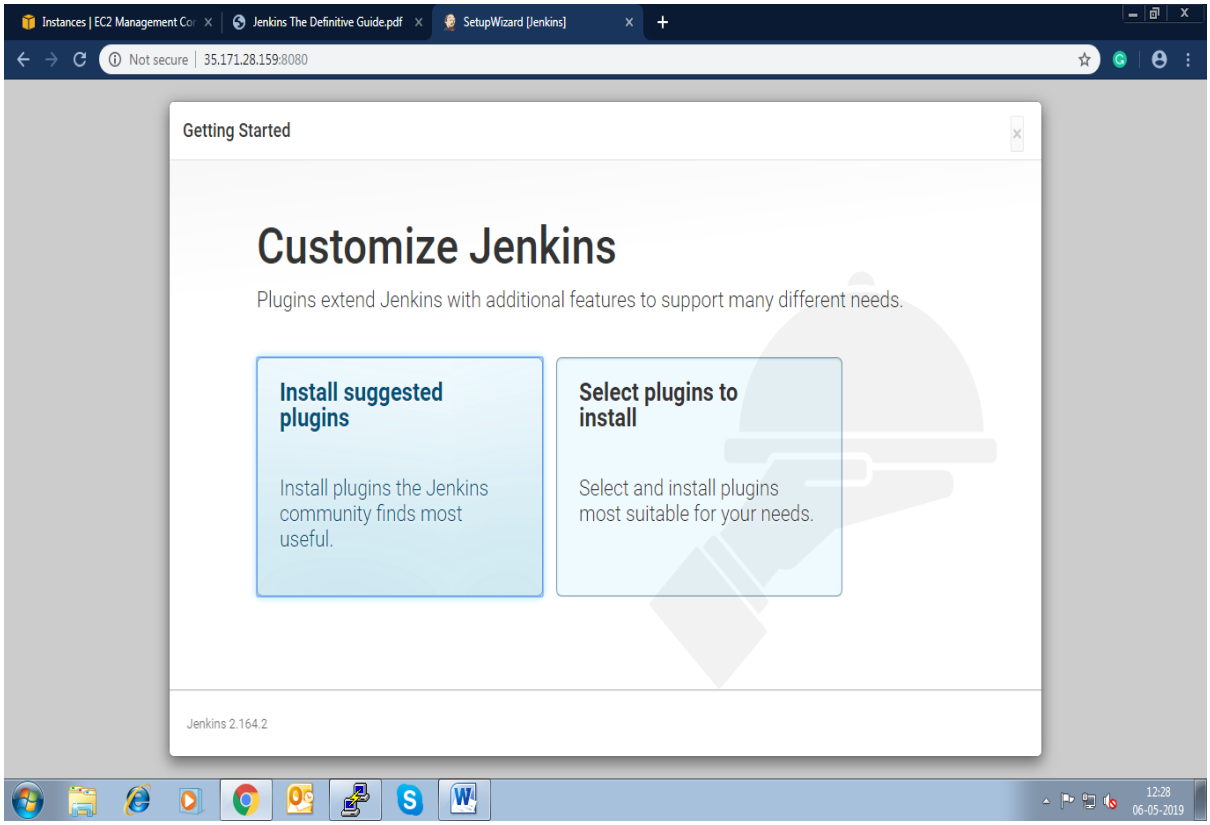
4. Installation of maven

- https://www.tutorialsmaven.com/aws-ec2/install-maven

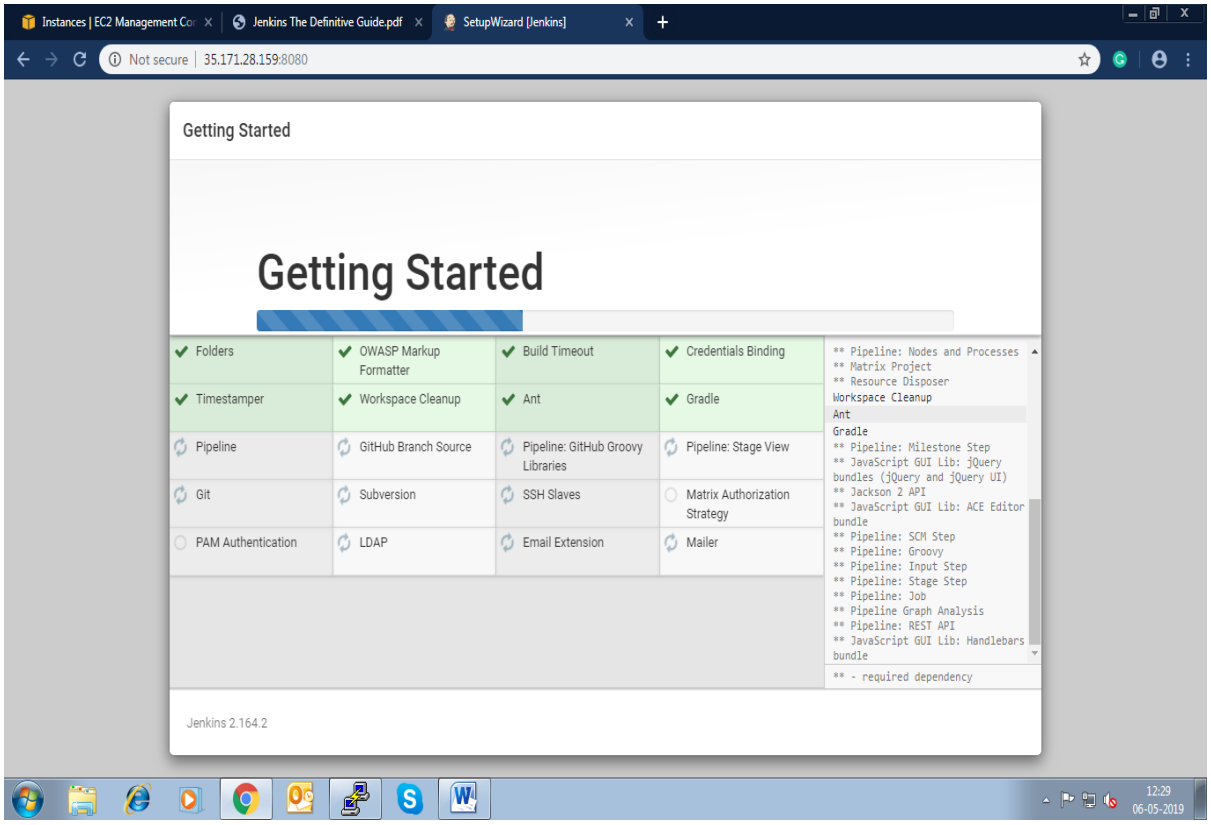


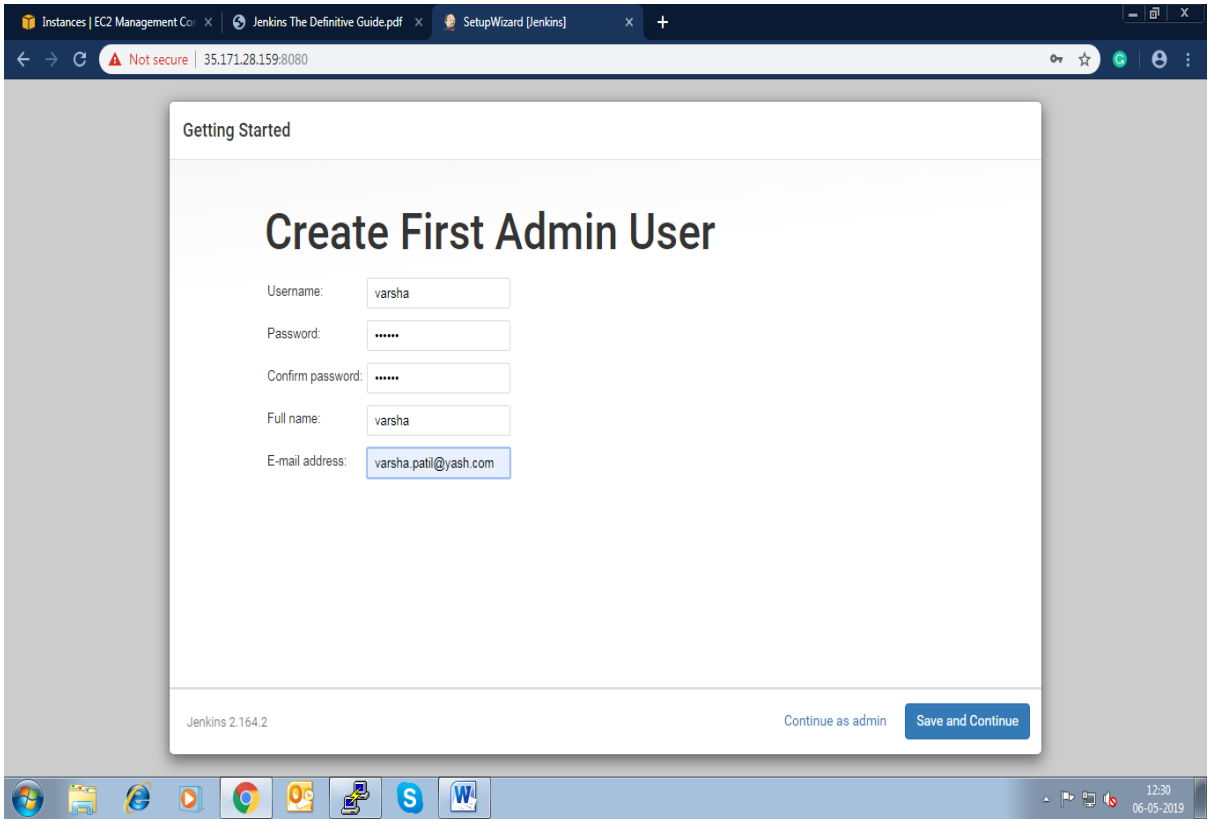
Go to sudo cat /var/lib/jenkins/secrets/initialAdminPasswd and copy passwd

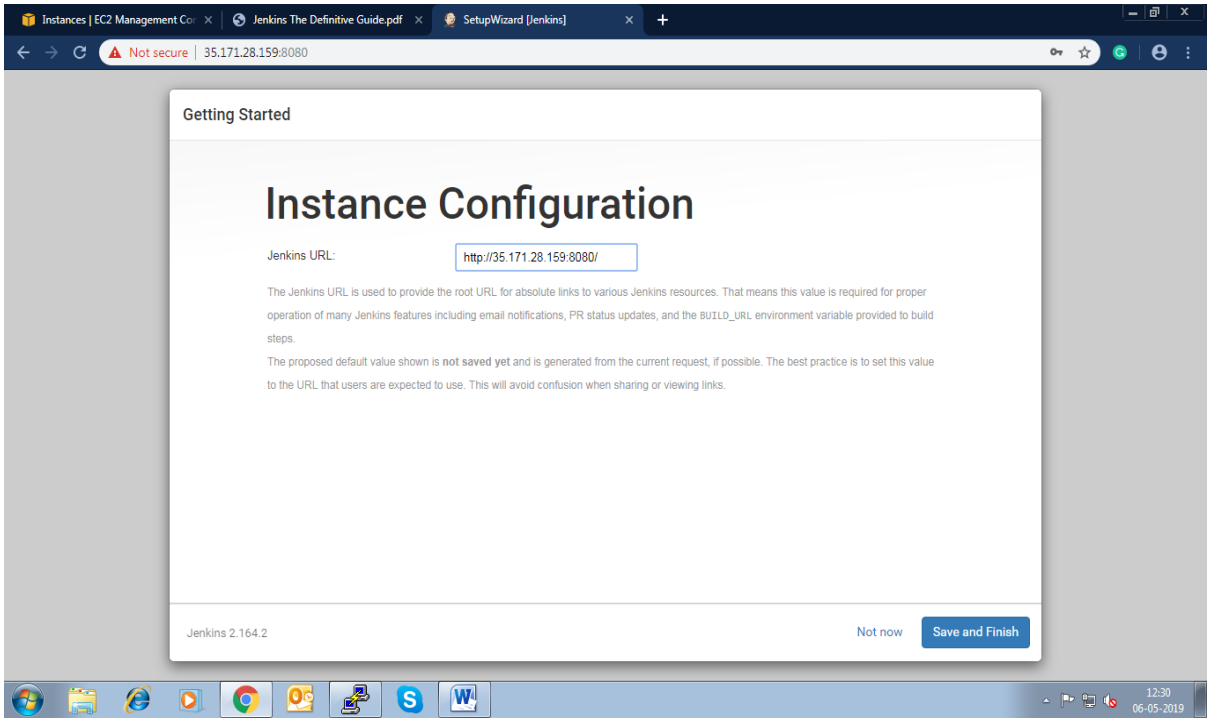




Click on install suggested plugins







5. Installation of Tomcat8

- wget http://mirrors.estointernet.in/apache/tomcat/tomcat-8/v8.5.40/bin/apache-tomcat-8.5.40.zip ---->tomcat8 zip file
- [ec2-user@ip-172-31-72-168 ~]$ ls
- apache-tomcat-8.5.40.zip
- [ec2-user@ip-172-31-72-168 ~]$ mv apache-tomcat-8.5.40.zip /var/lib/Jenkins/
- [ec2-user@ip-172-31-72-168 ~]$ Cd /var/lib/Jenkins/
- [ec2-user@ip-172-31-72-168 jenkins]$unzip apache-tomcat-8.5.40.zip
- [ec2-user@ip-172-31-72-168 jenkins]$sudo chown -R jenkins:jenkins apache-tomcat-8.5.40/sudo chmod 755 –R
- [ec2-user@ip-172-31-72-168 jenkins]$ sudo chmod 755 -R apache-tomcat-8.5.40/bin/*
- [ec2-user@ip-172-31-72-168 jenkins]$ cd apache-tomcat-8.5.40/conf/
- [ec2-user@ip-172-31-72-168 conf]$ sudo vi server.xml →change port numbers because Jenkins uses

8080 port it will cause conflict so change in range of 9090

- cd /var/lib/jenkins/apache-tomcat-8.5.40/bin/
- [ec2-user@ip-172-31-72-168 bin]$ sudo sh startup.sh

Tomcat started

- Go to URL →http://184.72.194.77:9090

Troubleshooting: -

Troubleshooting related Tomcat8

- Check permission owner and user should be Jenkins for apache tomcat
- If permission denied issue coming in check /var/lib/Jenkins/apache-tomcat/logs/catlina.out then clear all logs once again set
- Chown –R Jenkins:Jenkins apache-tomcat/*
- Chmod –R 755 apache-tomcat/bin/*
- Once again stop then start tomcat

Install sonar-scanner in master

- cd /opt/sonar/
- sudo wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-3.2.0.1227-linux.zip
- sudo unzip sonar-scanner-cli-3.2.0.1227-linux.zip
- sudo vim sonar-scanner-3.2.0.1227-linux/conf/sonar-scanner.properties

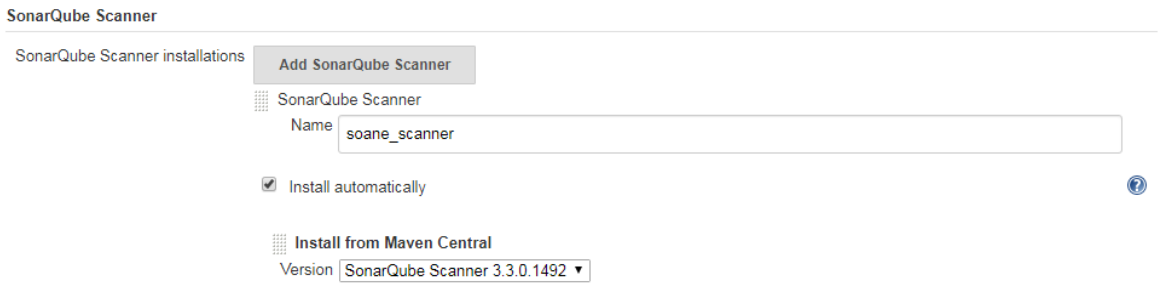
- Make changes
- Here put IP address of machine on which sonarqube is installed with port

```
#Configure here general information about the environment, such as SonarQube serv
er connection details for example
#No information about specific project should appear here

#----- Default SonarQube server
sonar.host.url=http://3.85.103.195:9000/sonar
#----- Default source code encoding
#sonar.sourceEncoding=UTF-8

#----- Global database settings
#sonar.jdbc.username=sonar
#sonar.jdbc.password=sonar
```

Here environment for SonarQube is ready. Installed java-1.8.0*,Jenkins, maven, git, sonar-sanner in master while updated java version and sonarqube-6.7 in slave machine.

Or else there also one option go to **Manage Jenkins-> Global tool Configuration**

Click on SonarQube scanner and check install automatically.



On master

Go to **Manage Jenkins→Plugin Manager**

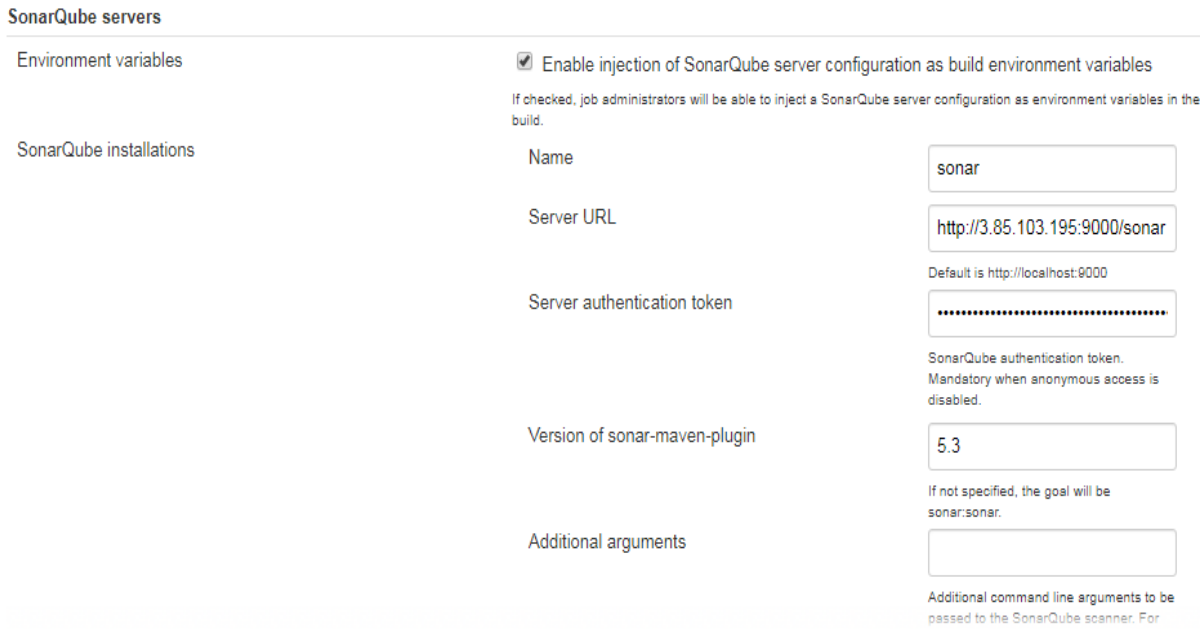
Install plugins

- SonarQube Scanner for Jenkins→Here require it gives detail of path of sonar-scanner, credentials, IP address of machine on which SonarQube installed
- Email Extension Plugin →Used to notify status of job through mail
- Maven Invoker Plugin→ To create maven based job
- JUnit Plugin→Display report
- Build Pipeline Plugin→To create project in Pipeline
- Deploy to container Plugin→Deploy war file on server(Tomcat8X)
- Copt Artifact

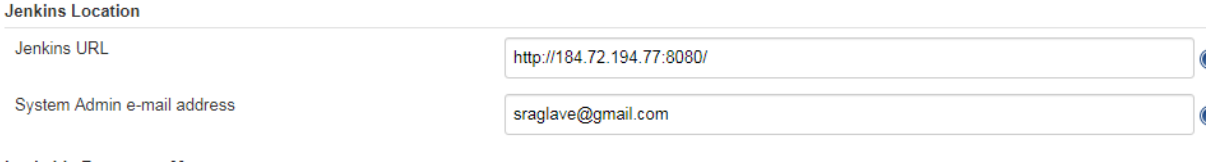
Go to **Manage Jenkins->Configure system**

SonarQube servers

- Name ---Give any
- Sercver Url ---http://IP address of machine on sonarqube/sonar
- Server authentication token---after enter in sonarqubt generate token then paste here

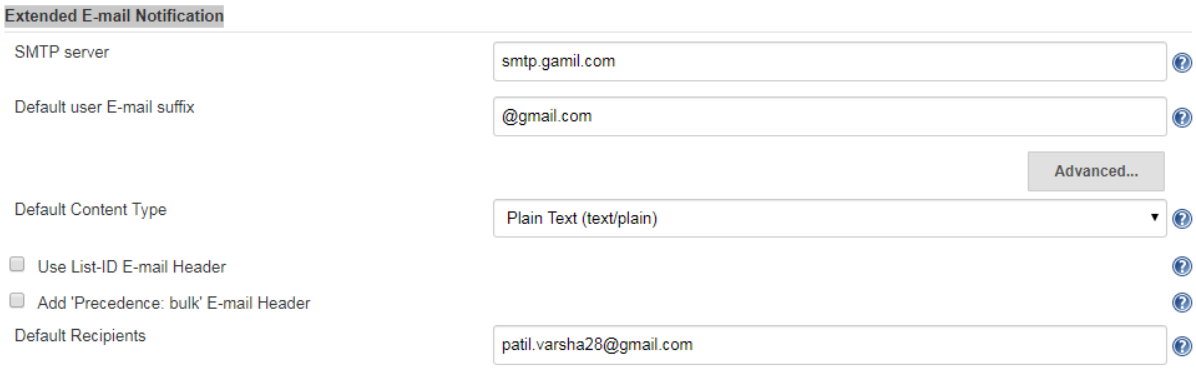


- **Jenkins Location**



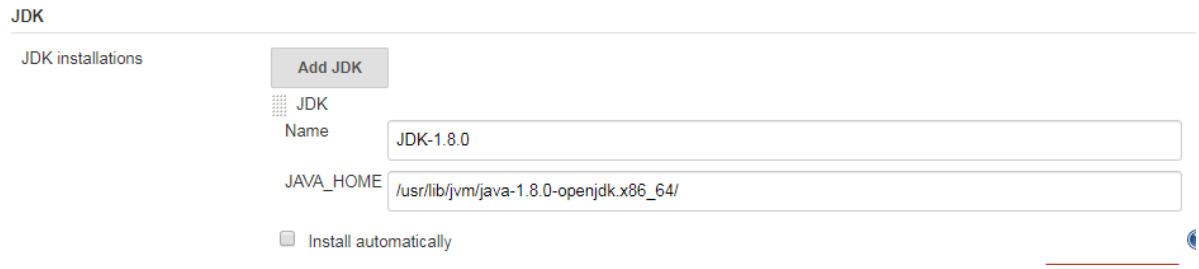
- **Extended E-mail Notification**

- SMTP server ----smtp.gmail.com
- Default user E-mail suffix---@xxx.com
- Default Recipient-email id to whom mail send

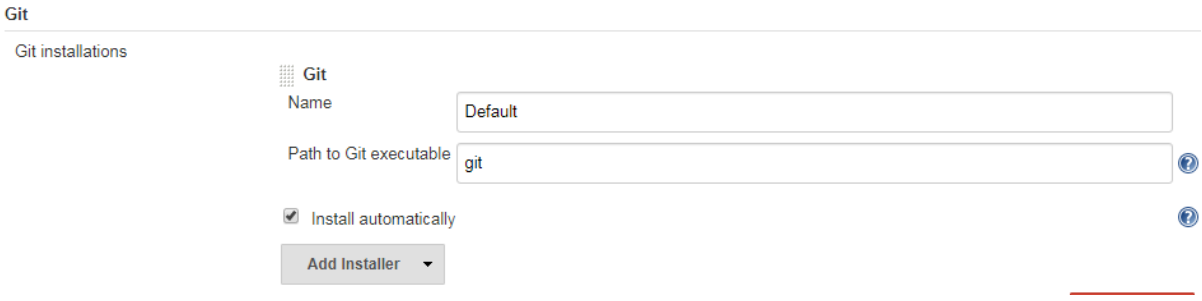


Go to **Manage Jenkins->Global Configuration Tool**

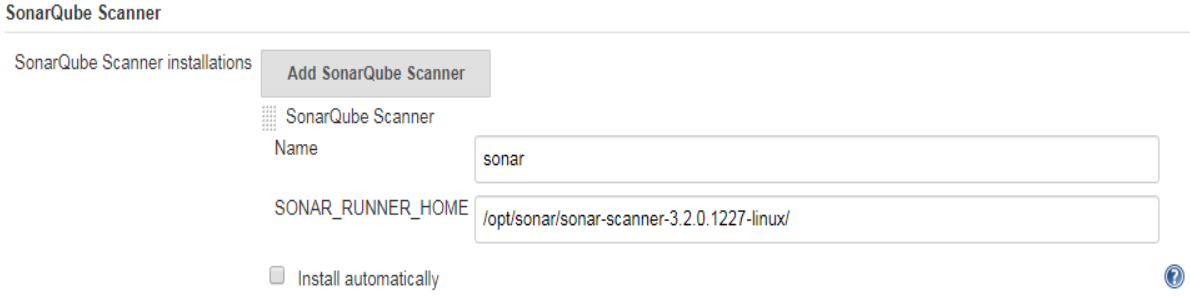
- JDK
- Put path of JDK



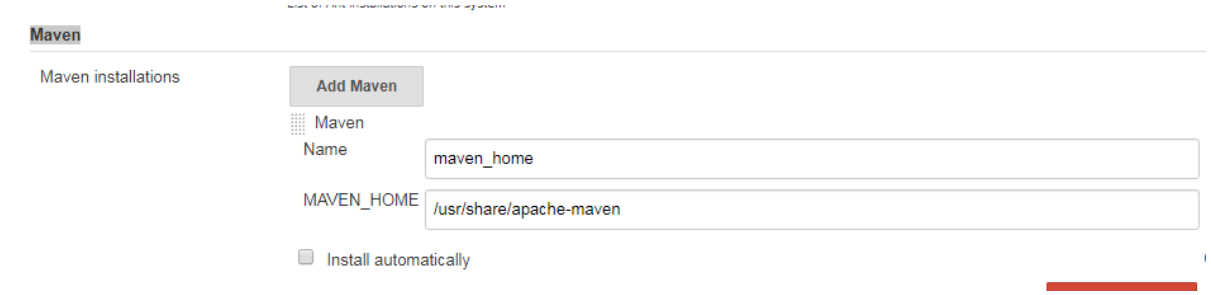
- **Git**



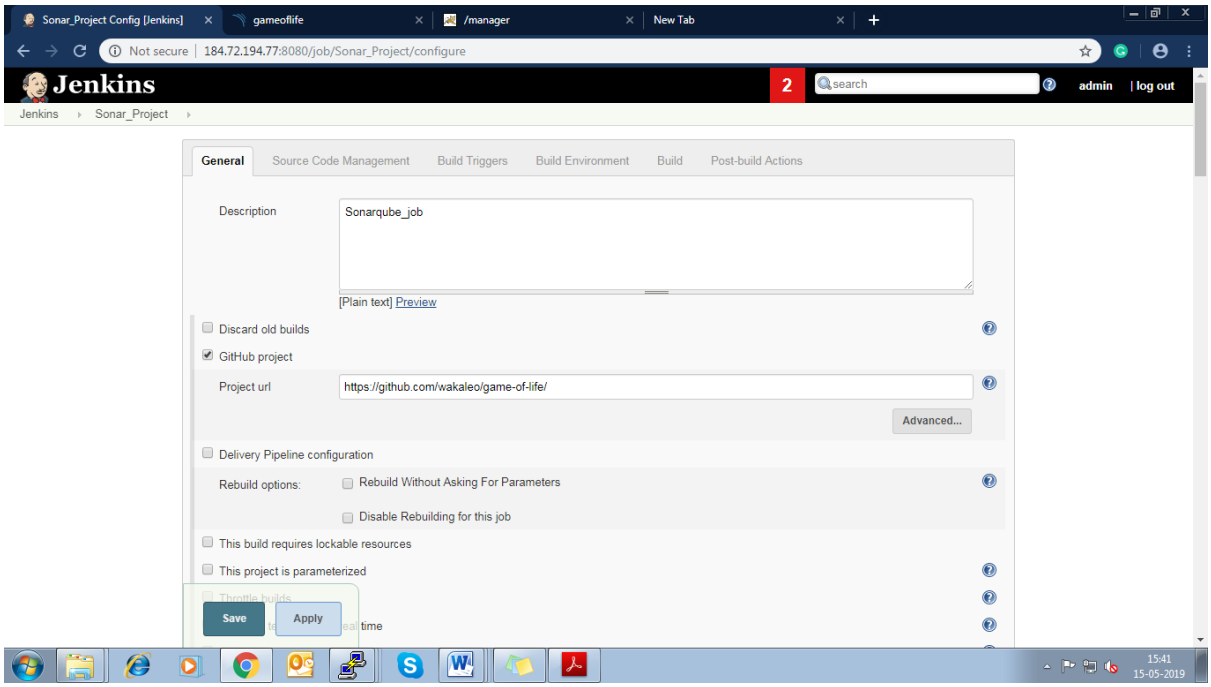
- **SonarQube Scanner**
- Put path of sonar-scanner which installed on master or tick install automatically



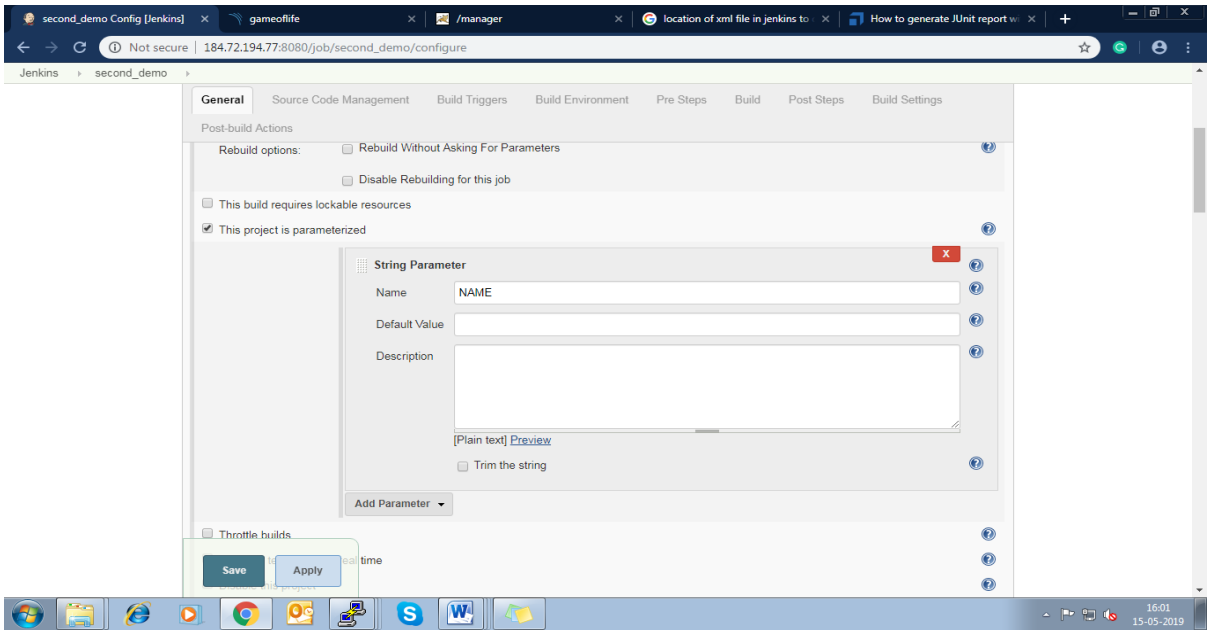
Maven



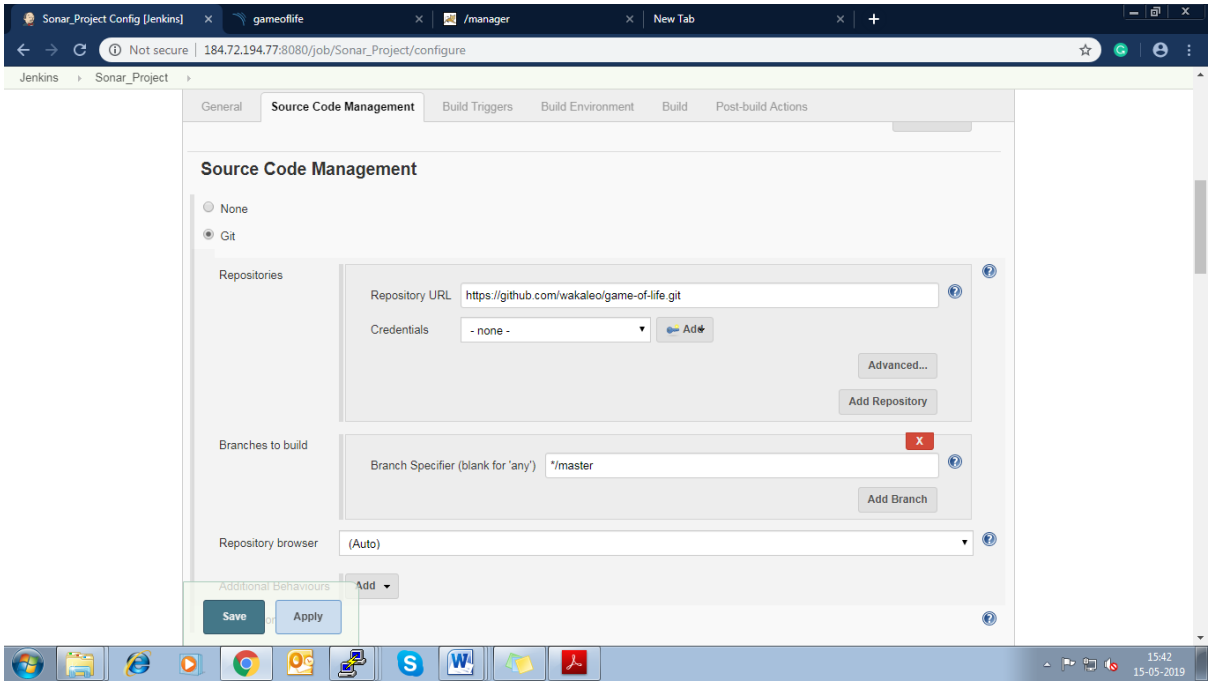
a) Create Maven project



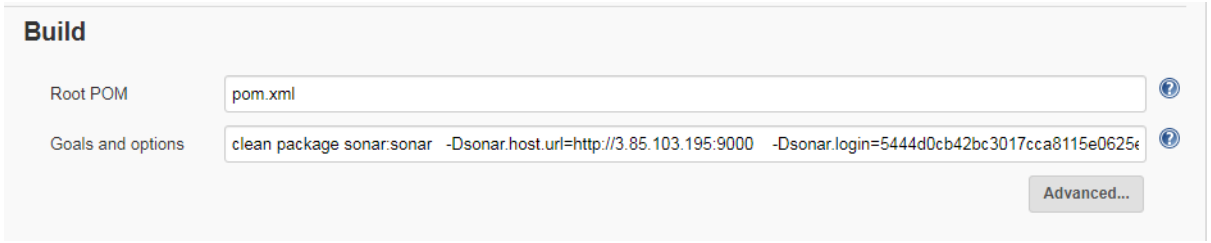
b) To make parameterized project enter parameter as NAME



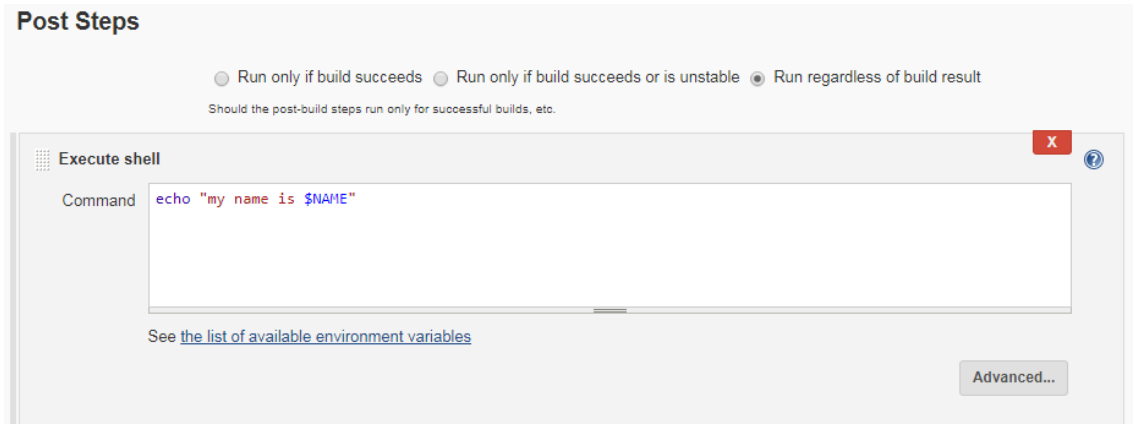
c) Put Project URL- https://github.com/wakaleo/game-of-life/



1. Build-Here after execution whole project SonarQube dashboard display then add name of project generate token .Copy that and add into Manage Jenkins→configure System→SonarQube scanner →credentials
2. After that select language as Maven and machine→Linux it generate code copy that code into Goals and options (remove mvn and / from that copied)



3. Add Execute shell script to pass parameter



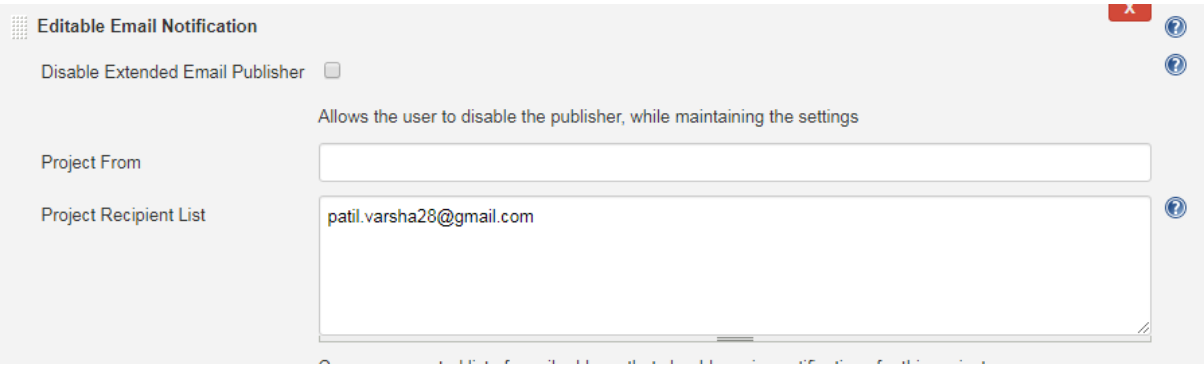
4. For pipeline project we want to copy war file from first job to second job so download copy Artifact plugin then select Archive the artifacts from Post build action.

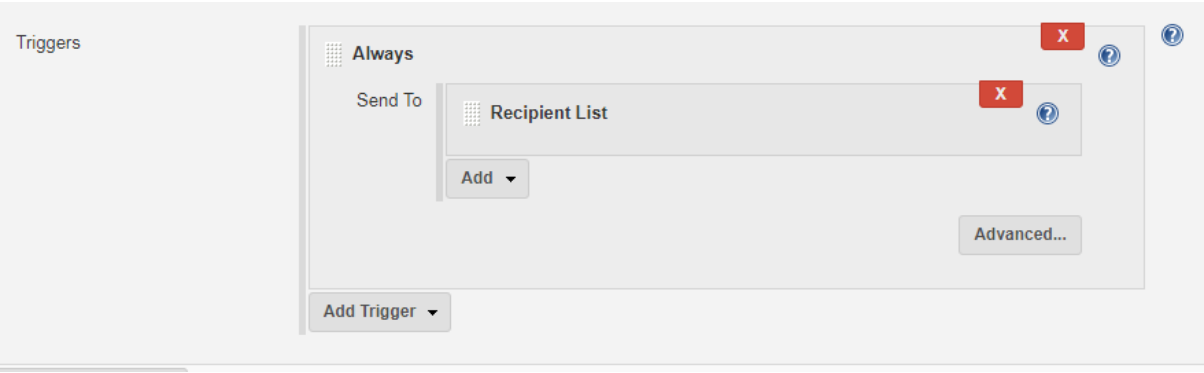
Give the path of war file after build, so that it will archive the artifact.

Create second job Tomcat to deploy war file on Tomcat server for Pipeline. (Freestyle)

a) Editable Email Notification—add email id to whom mail sent

In Advanced select Triggers in that select Recipient List Always

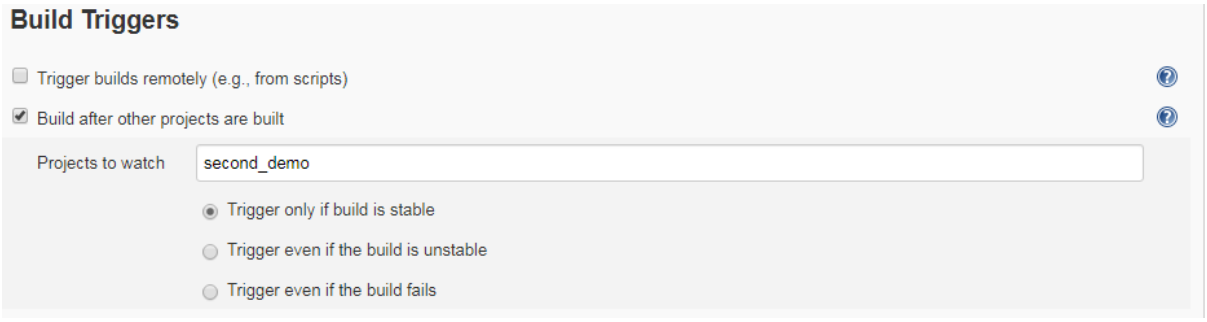




b) Create second project name as Tomcat job(name of project is Tomcat)

NOTE-Do not put Project URL while creating job in pipeline(second job) Specify name for upstream job in Build Triggers

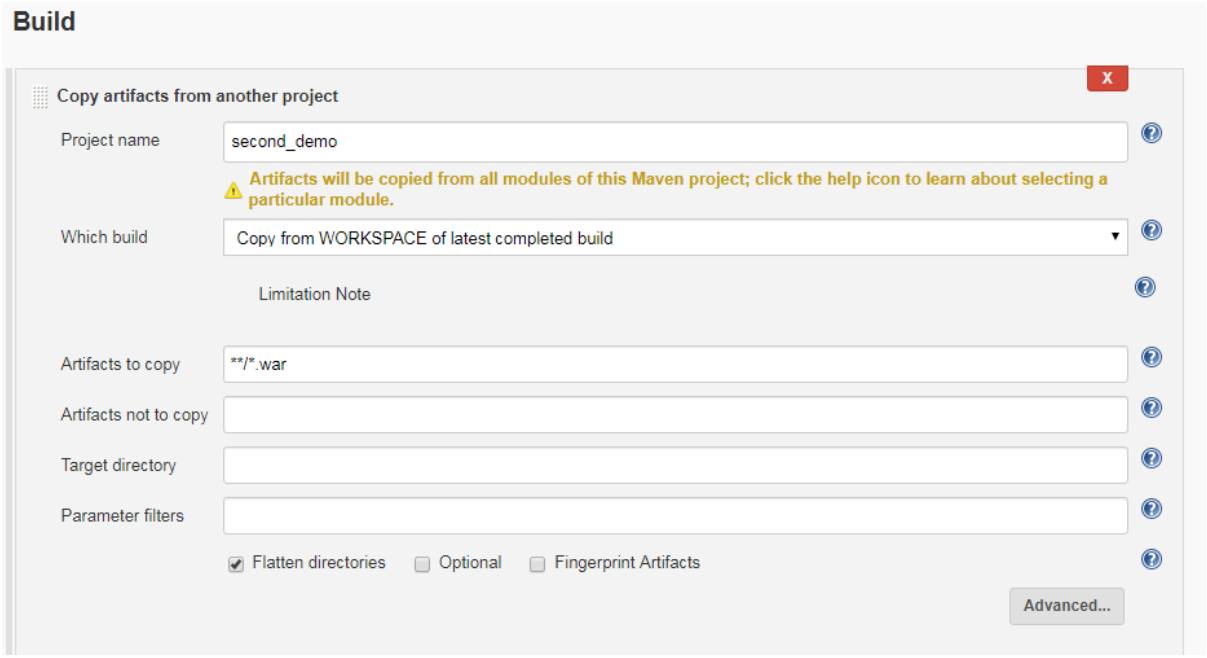
Second project



1. Build →

   Project Name→Name of project from which war will be copied

   Which Build→select latest option or copy from WORKSPACE



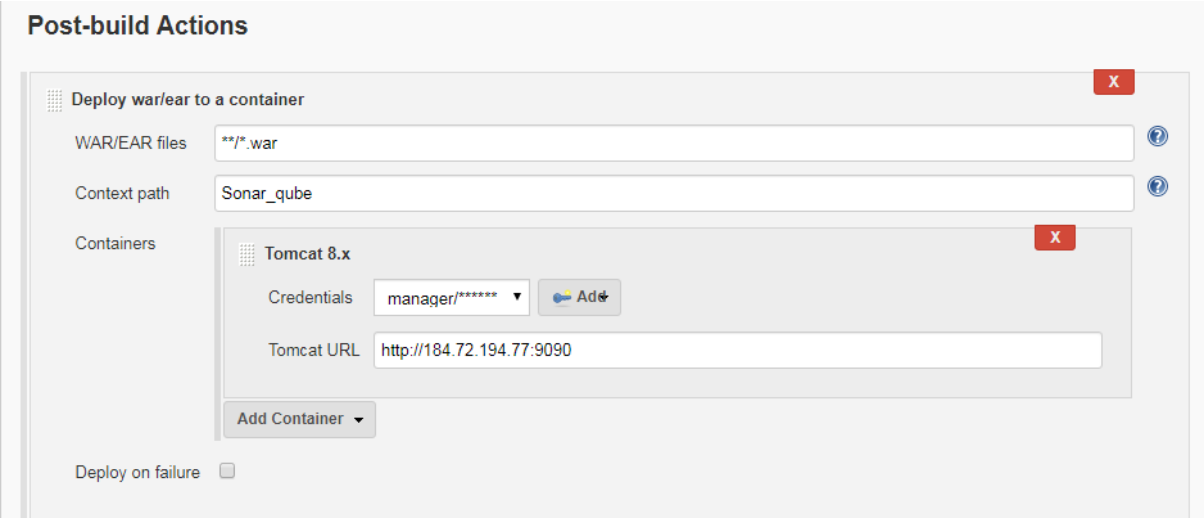
2. To deploy on tomcat select -Deploy war/ear to a container –select war file location

In context path put name of folder in which you want war file on server

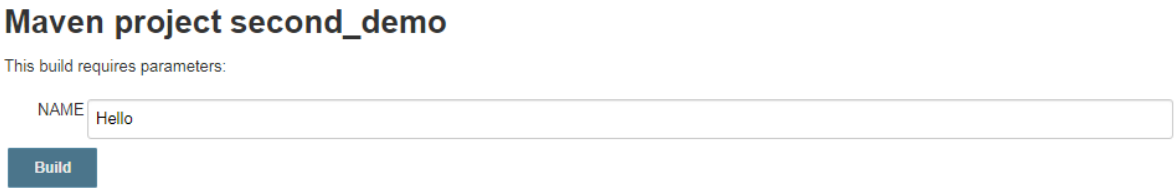
Add credential –Add—Username-manager

Password-manager

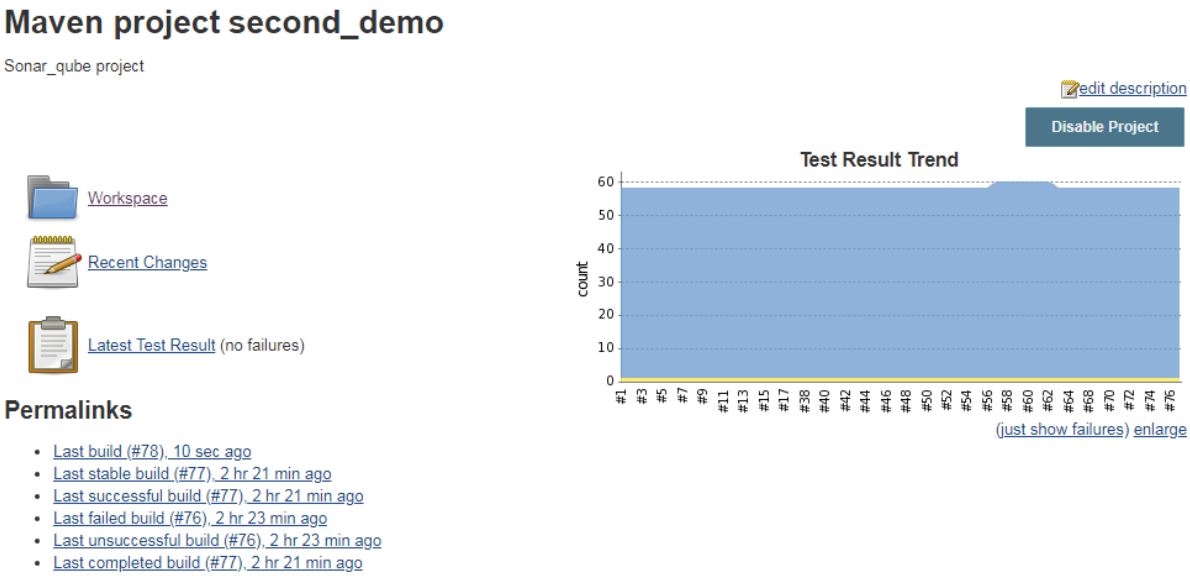
URL---On which tomcat running (master)



- Build Project
  - Click on Build with Parameter



- After Build



- ANLYSIS SUCCESSFUL

http://3.85.103.195:9000/dashboard/index/com.wakaleo.gameoflife:gameoflife
Click on url

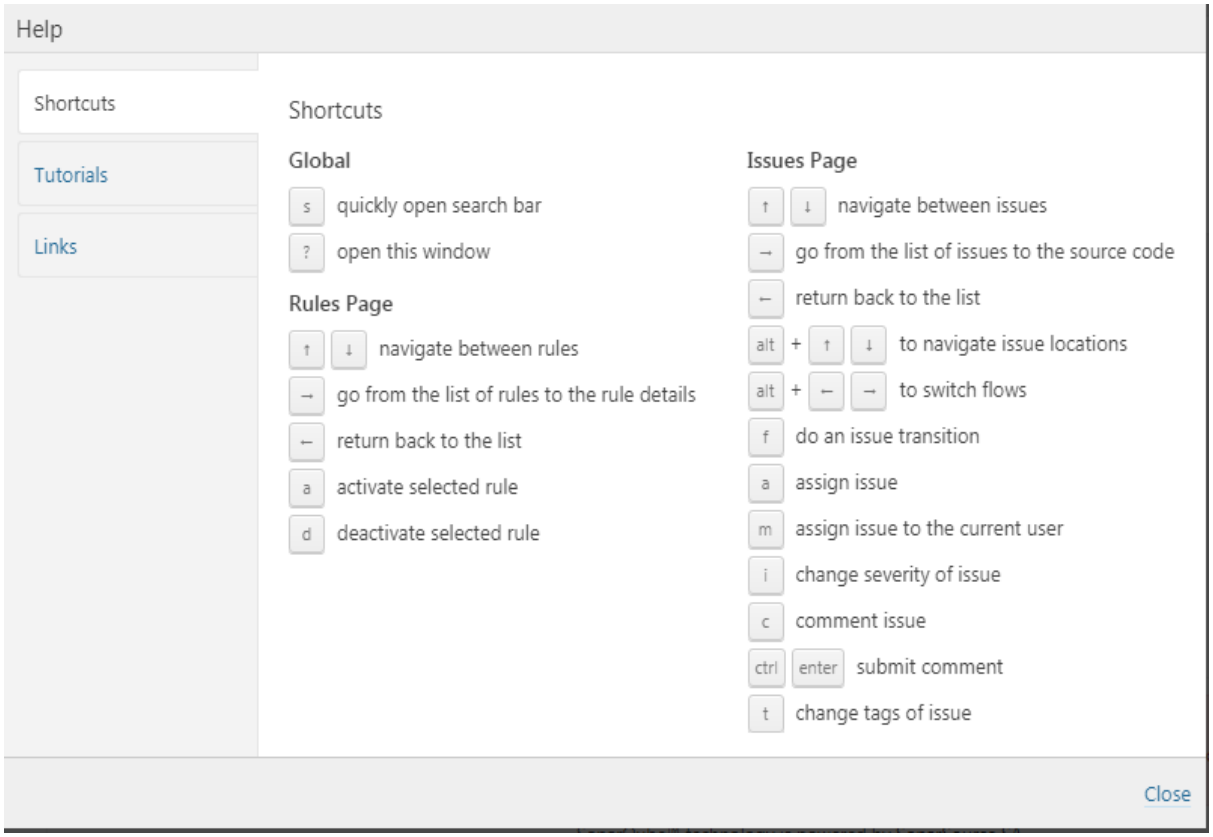
```
[INFO] Analysis reports compressed in 83ms, zip size=80 KB
[INFO] Analysis report uploaded in 66ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://3.85.103.195:9000/dashboard/index/com.wakaleo.gameoflife:gameoflife
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://3.85.103.195:9000/api/ce/task?id=AWq7EaqcdCmLFhw4HDFo
[INFO] Task total time: 13.414 s
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] gameoflife ......................................... SUCCESS [ 15.374 s]
[INFO] gameoflife-build ................................... SUCCESS [  2.194 s]
[INFO] gameoflife-core .................................... SUCCESS [ 11.012 s]
[INFO] gameoflife-web ..................................... SUCCESS [  3.930 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 40.080 s
```

Shows SonarQube dashboard

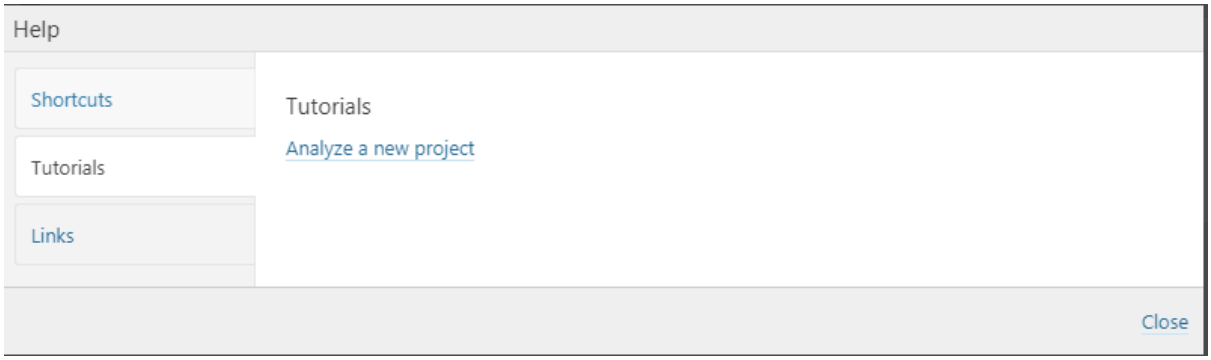
Login as →Username→admin

Password-admin

Click on upper right side '?'(sign)



Go to tutorial →Create new project

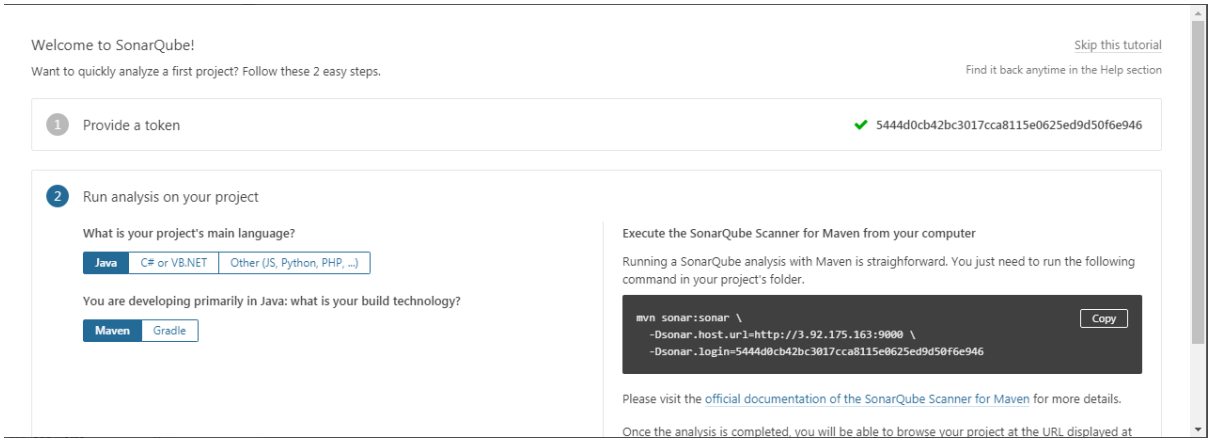


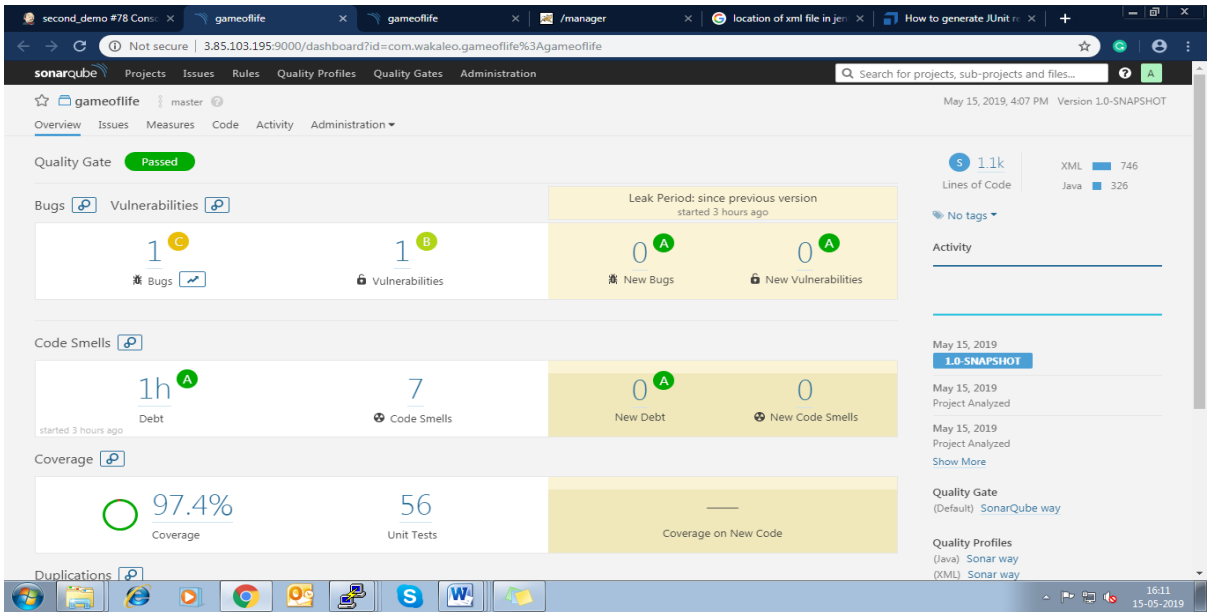
After creating new project it will ask enter previous token or generate new token

Copy that token in Manage Jenkins→Configure system in sonar-scanner

Run analysis on your project → select language (Java)

- Select Build technology(Maven)
- On right side it will generate code
- Copy that code in to goals of first project in BUILDS

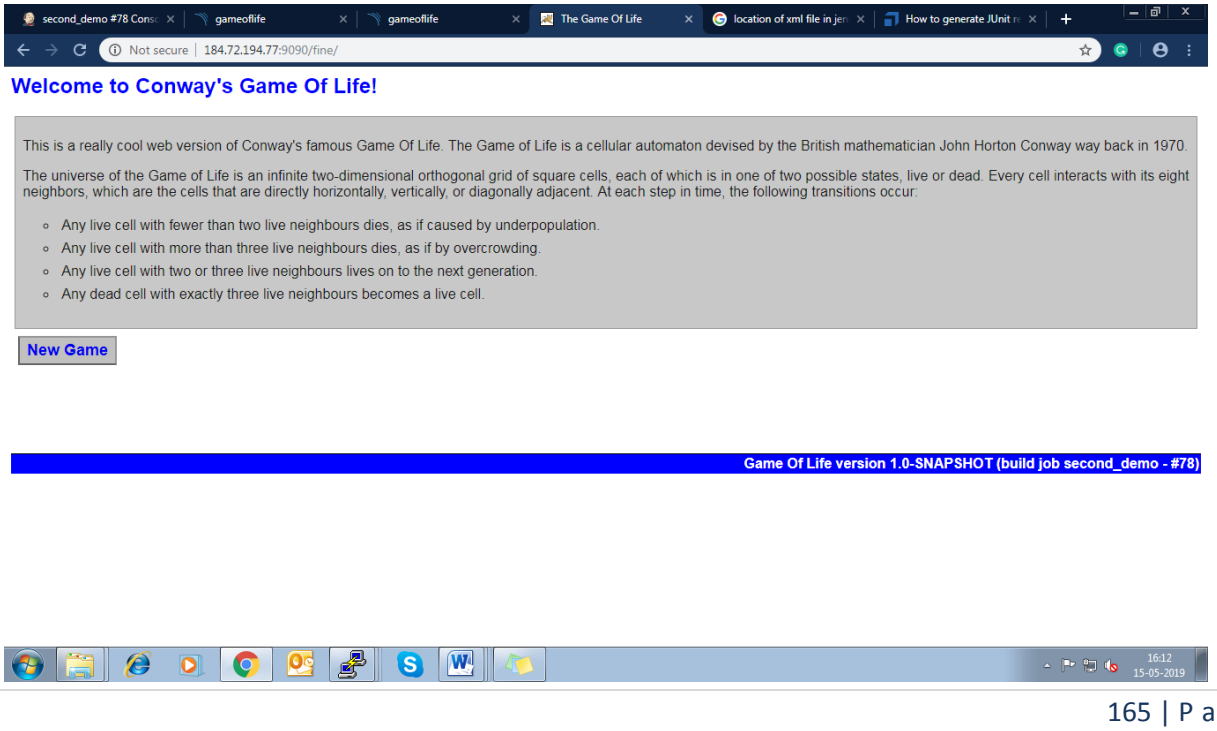




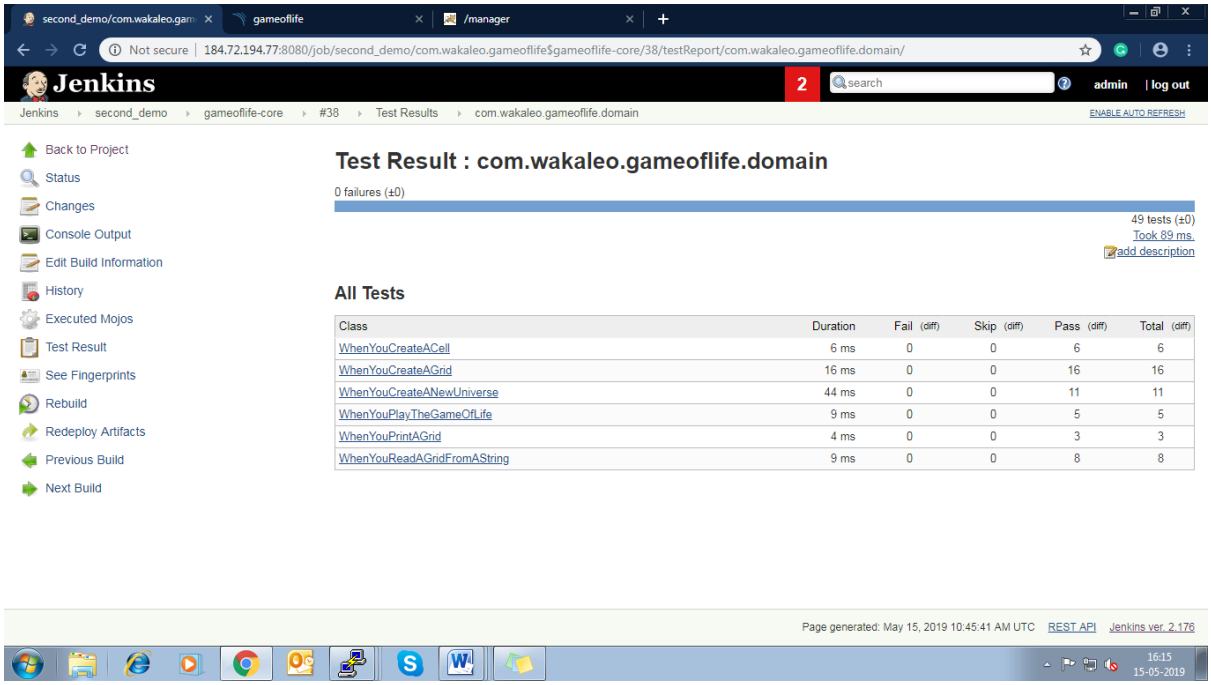
This snap shows whatever parameter passed executed in script, war file deployed on Tomcat also email sent to mentioned email id.

```
[second_demo] $ /bin/sh -xe /tmp/jenkins1520228676423676368.sh
+ echo 'my name is Hello'
my name is Hello
Deploying /var/lib/jenkins/workspace/second_demo/gameoflife-web/target/gameoflife.war to container Tomcat 8.x Remote with context fine
  Redeploying [/var/lib/jenkins/workspace/second_demo/gameoflife-web/target/gameoflife.war]
  Undeploying [/var/lib/jenkins/workspace/second_demo/gameoflife-web/target/gameoflife.war]
  Deploying [/var/lib/jenkins/workspace/second_demo/gameoflife-web/target/gameoflife.war]
Email was triggered for: Always
Sending email for trigger: Always
Sending email to: patil.varsha28@gmail.com
Finished: SUCCESS
```

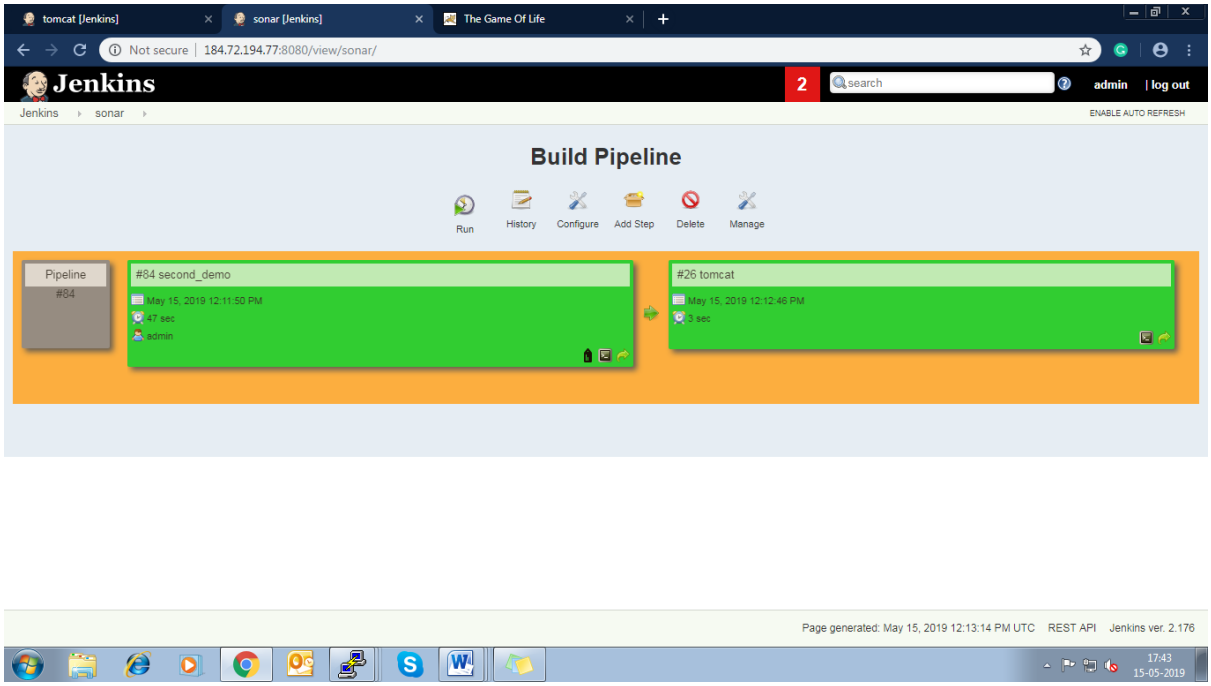
- Clicking on Fine folder (which we have created in second job of pipeline)



- Clicking on Test Results on Jenkins dashboard

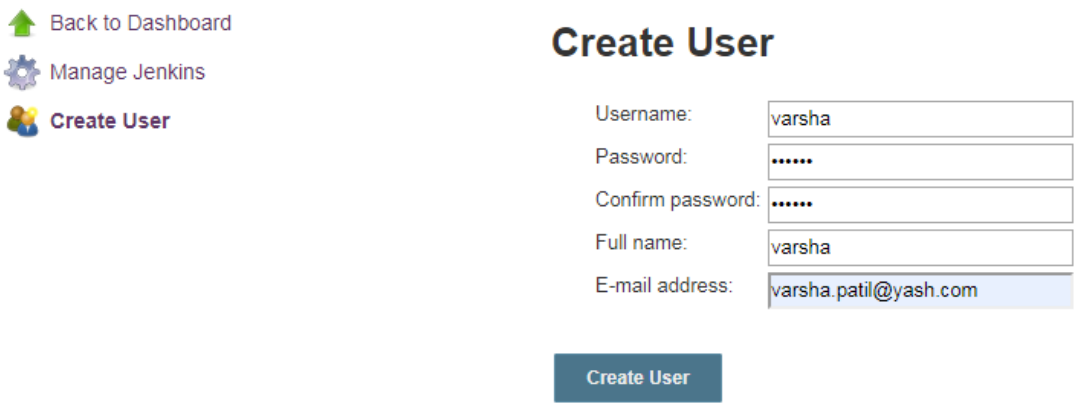


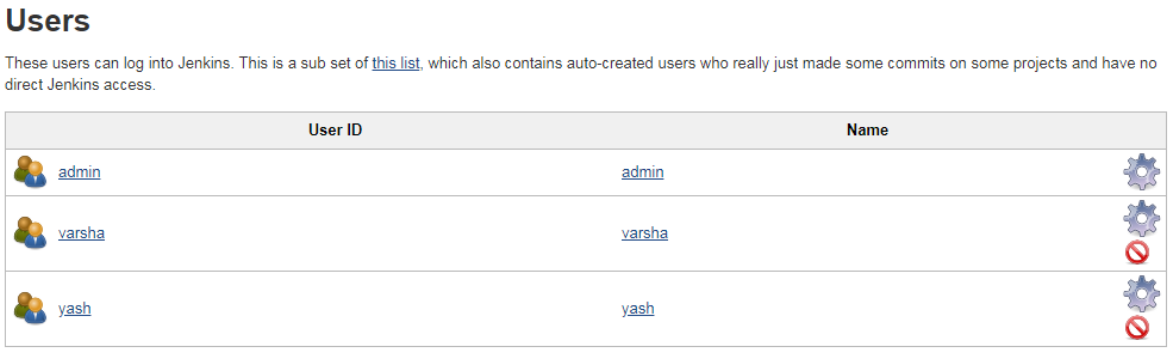
- Project in Build Pipeline after running



Matrix Based Authorization

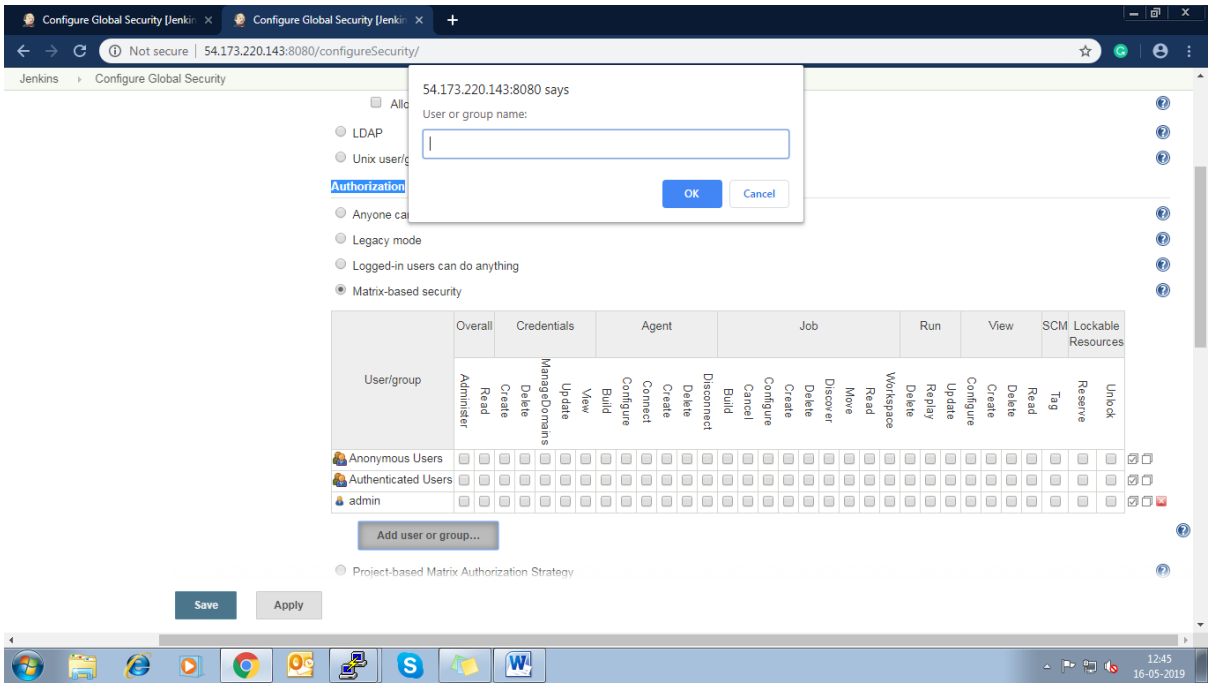
Manage Jenkins→ Manage User

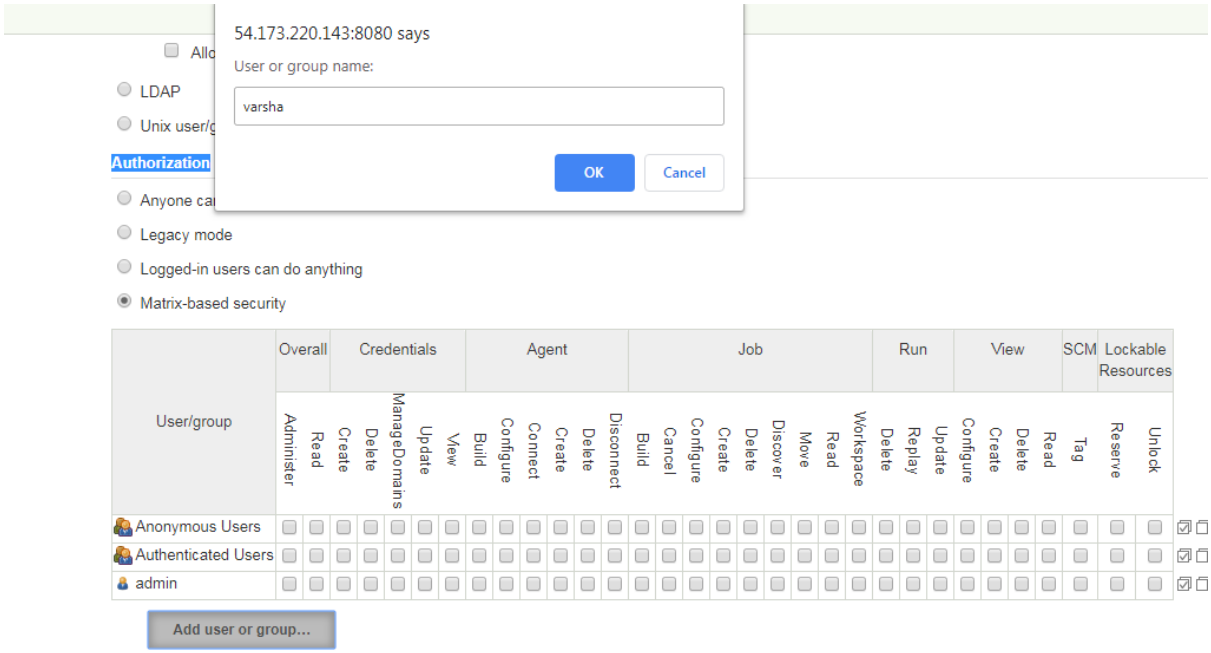




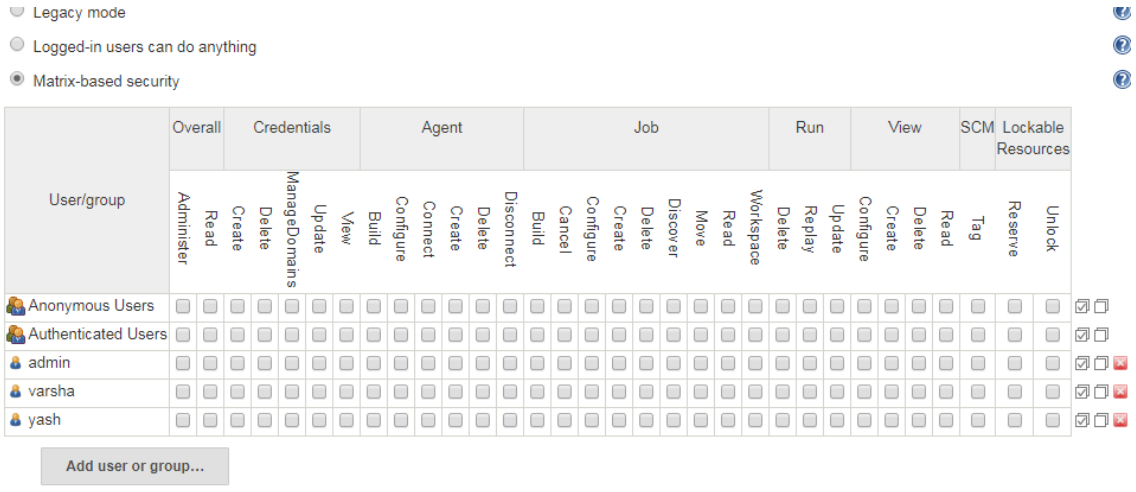
Go to Manage Jenkins→Global Tool Configuration→**Authorization→Matrix based security**

Add user or group



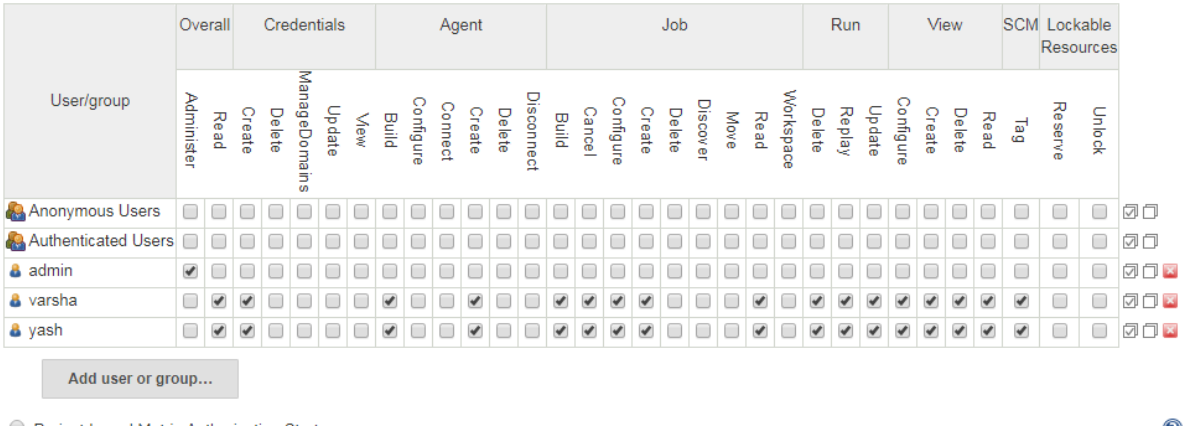


After adding user

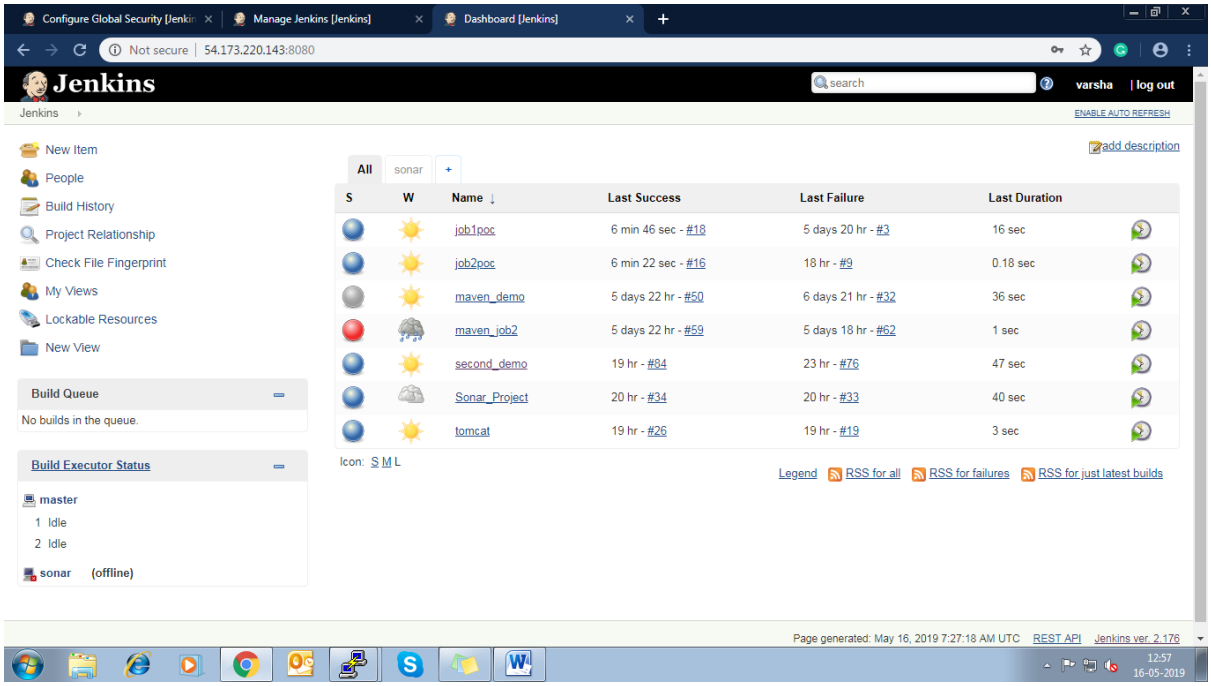


Give permission new created users

Here admin is main user and here two new users created has given selected permissions



Login with new created user



Troubleshooting: -

Error was unable to copy artifacts (war) from previous job to current job

Solution In previous job go to post build action select Archive artifact option specify file name to archived, then go to next job in Build option select copy artifact from another project inside select project name and Artifact to copy i.e.**/*.war

Unable to clone repo origin on console output

At master slave configuration Jenkins wasn't able to recognize path automatically by itself

Solution→changes we done remove path of Git from slave Tool →configuration while launching instance

Remove and install Git properly on slave

Sonarqube-7.6 installation

On Linux machine (another Linux machine where Jenkins is not installed because of memory issue) run following commands.

Prerequisite:

The only prerequisite for running SonarQube is to have Java (Oracle JRE 8 or OpenJDK 8) installed on your machine. If it is not installed then run following commands:

- sudo yum install java-1.8.0-openjdk
- sudo yum install java-1.8.0-openjdk-devel
- update-alternatives --config java
- java -version

Installation of PostgreSQL database:

Refer site: https://www.vultr.com/docs/how-to-install-sonarqube-on-centos-7

- sudo rpm –Uvh https://download.postgresql.org/pub/repos/yum/9.6/redhat/rhel-7-x86 64/pgdg-centos96-9.6-3.noarch.rpm
- sudo yum –y install postgresql96-server postgresql96-contrib
- sudo /usr/pgsql-9.6/bin/postgresql96-setup initdb
- Edit the /var/lib/pgsql/9.6/data/pg|_hba.conf

Find the following lines and change peer to trust and ident to md5

Find the following lines and change `peer` to `trust` and `idnet` to `md5`.

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            ident
# IPv6 local connections:
host    all             all             ::1/128                 ident
```

Once updated, the configuration should look like the one shown below.

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
```

- sudo systemctl start postgresql-9.6
- sudo systemctl enable postgresql-9.6
- sudo passwd postgres
- su – postgres
- createuser sonar
- psql
- ALTER USER sonar WITH ENCRYPTED password ‘sonar’;
- CREATE DATABSE sonar OWNER sonar;
- \q
- exit

Configure SonarQube:

- sudo mkdir /opt/sonarqube
- cd /opt/sonarqube
- sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-7.6.zip
- sudo unzip sonarqube-7.6.zip
- sudo vi /opt/sonarqube/sonarqube-7.6/conf/sonar.properties

Make changes

- At line number 15 (uncomment and give username and password as sonar because in postgresql databse we gave sonar as username and password)

The schema must be created first

sonar.jdbc.username=sonar
sonar.jdbc.password=sonar
At line number 41(uncomment)
(sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube?currentSchema=my_schema)instead of that type following command
sonar.jdbc.url=jdbc:postgresql://localhost/sonar

- At line number 99 (uncomment and increase memory)

sonar.web.javaOpts=-Xmx1024m -Xms512m -XX:+HeapDumpOnOutOfMemoryError

- At line number 107 (uncomment and add ip)

sonar.web.host=172.31.73.172 (give that instance public ip)

- At line number 111

sonar.web.context=/sonar (uncomment)

- At line number 113 (uncomment)

```
sonar.web.port=9000
```

- At line 249 (uncomment and increase memory)

```
sonar.ce.javaOpts=-Xmx1024m -Xms512m -XX:+HeapDumpOnOutOfMemoryError
```

- At line 272(uncomment and increase memory)

JVM options of Elasticsearch process
sonar.ce.javaOpts=-Xmx1024m -Xms512m -XX:+HeapDumpOnOutOfMemoryError

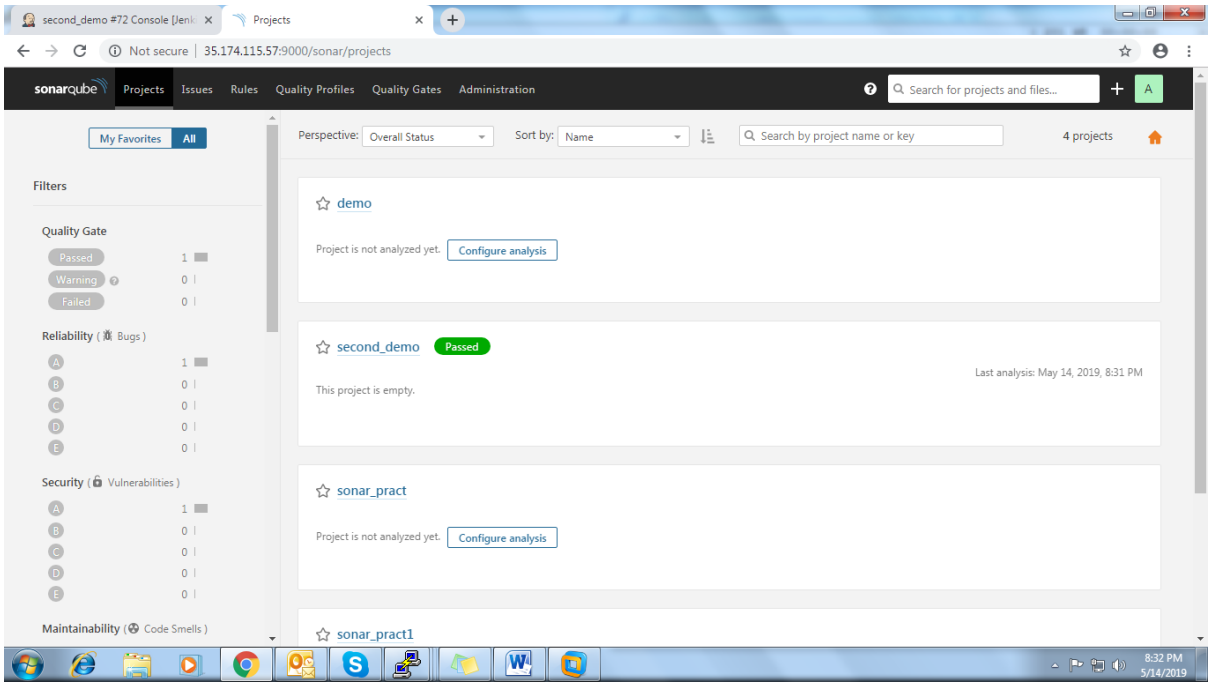
Save it and exit

- Open and make changes

```
cd /opt/sonarqube/sonarqube-7.6/bin/linux-x86-64/sonar.sh
RUN_AS_USER=/sonar
Save it and exit
```

- cd /opt/sonarqube/
- sudo chown –R sonar:sonar sonarqube-7.6/*
- cd /opt/sonarqube/sonarqube-7.6/bin/linux-x86-64/
- sudo ./sonar.sh start

Here SonarQube started dashboard coming but doesn't shows test cases



Here version sonarqubr7.6 not supported so we have chosen SonarQube 6.7.7, for which no need to install database

Troubleshooting: -

- In sonar.properties when we don't remove that line (sonarqube?currentSchema=my_schema) error will get (sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube?currentSchema=my_schema)instead of that type following command

Solution:

For this we checked web.log file which shows error that SonarQube not exits so checked in sonar.properties file in PostgreSQL db. Remove all after localhost and keep only /sonar

Remove that line and add sonar.

sonar.jdbc.url=jdbc:postgresql://localhost/sonar

1. Error at ip in sonar.properties

Solution:

sonar.web.host=172.31.73.172 (give that instance public ip)

2. Error 143 occurred during ./sonar.sh start

Showing in sonar.log

- 2017.10.21 00:26:02 INFO app[][o.s.a.AppFileSystem] Cleaning or creating temp directory /opt/sonarqube/temp
- 2017.10.21 00:26:03 INFO app[][o.s.a.es.EsSettings] Elasticsearch listening on /127.0.0.1:9001
- 2017.10.21 00:26:03 INFO app[][o.s.a.p.ProcessLauncherImpl] Launch process[[key='es', ipcIndex=1, logFilenamePrefix=es]] from [/opt/sonarqube/elasticsearch]: /opt/sonarqube/elasticsearch/bin/elasticsearch -Epath.conf=/opt/sonarqube/temp/conf/es
- 2017.10.21 00:26:03 INFO app[][o.s.a.SchedulerImpl] Waiting for Elasticsearch to be up and running
- 2017.10.21 00:26:03 INFO app[][o.e.p.PluginsService] no modules loaded
- 2017.10.21 00:26:03 INFO app[][o.e.p.PluginsService] loaded plugin [org.elasticsearch.transport.Netty4Plugin]
- 2017.10.21 00:26:04 WARN app[][o.s.a.p.AbstractProcessMonitor] Process exited with exit value [es]: 1
- 2017.10.21 00:26:04 INFO app[][o.s.a.SchedulerImpl] Process [es] is stopped
- 2017.10.21 00:26:04 INFO app[][o.s.a.SchedulerImpl] SonarQube is stopped

bink this is an issue with the docker image.

Solution:

In sonar.properties file edited memory size 512→1024 and 128→512 available different line numbers

sonar.ce.javaOpts=-Xmx1024m -Xms512m -XX:+HeapDumpOnOutOfMemoryError

sonar.ce.javaOpts=-Xmx1024m -Xms512m -XX:+HeapDumpOnOutOfMemoryError

Troubleshooting: -

- Sometimes there will error while running jobs from the agent node. The common error while launching the node will be “no entry found in known hosts file”, to resolve this error copy the known_hosts entry from users .ssh folder to /var/lib/Jenkins/.ssh/known_hosts file. The try to connect.
- There will be an error, the node is unable to find git location, the try to check the correct version and installation of git and also check the given path of the git home directory.
- There will be an error, the node is unable to run maven commands due to error in maven home path directory entry or java version issues. Check for the installation path correctly while configuring nodes and also check for the correct java verson.
- There will be an error while deploying maven project using tomcat in master machine, the error will be permission denied or unable to create catalina.out log in logs folder of tomcat. To resolve this error copy the tomcat folder to the Jenkins installation directory with the Jenkins user permission to that directory, this will be done by using sudo. Then given the executable permission to bin directory of tomcat. Then try to run the startup.sh command using sudo.
- There will be an error that tomcat is configured correctly but, because the users are not created it will show the error, to resolve that error go to /var/lib/jenkins/apache-tomcat-8.5.40/webapps/manager/META-INF edit the context.xml file and comment out the blue entries shown below.

```
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<Context antiResourceLocking="false" privileged="true" >
<!--   <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->
  <Manager sessionAttributeValueClassNameFilter="java\.lang\.(?:Boolean|Integer
Long|Number|String)|org\.apache\.catalina\.filters\.CsrfPreventionFilter\$LruCa
```

Note:- If you want to remove any service, first stop that service, otherwise it block that particular port.

Troubleshooting: -

Problem

```
[root@ip-172-31-80-145 ~]# service jenkins start
JENKINS_HOME directory does not exist: /var/lib/jenkins
```

Solution

```
[root@ip-172-31-80-145 home]# mv /home/jenkins/ /var/lib/jenkins
[root@ip-172-31-80-145 home]# cd /var/lib/jenkins/
[root@ip-172-31-80-145 jenkins]# ls
key.pem  spring3-mvc-maven-xml-hello-world
[root@ip-172-31-80-145 jenkins]# cd
[root@ip-172-31-80-145 ~]# service jenkins start
Starting Jenkins                                           [  OK  ]
```

Tomcat

Troubleshooting related Tomcat8

- Check permission owner and user should be Jenkins for apache tomcat
- If permission denied issue coming in check /var/lib/Jenkins/apache-tomcat/logs/catlina.out then clear all logs once again set
- Chown –R Jenkins:Jenkins apache-tomcat/*
- Chmod –R 755 apache-tomcat/bin/*
- Once again stop then start tomcat

Problem:-

```
        at org.codehaus.cargo.container.tomcat.internal.TomcatManager.list(TomcatManager.java:876)
        at org.codehaus.cargo.container.tomcat.internal.TomcatManager.getStatus(TomcatManager.java:889)
        at org.codehaus.cargo.container.tomcat.internal.AbstractTomcatManagerDeployer.redeploy(AbstractTomcatManagerDeployer.java:173)
        ... 17 more
Caused by: java.io.IOException: Server returned HTTP response code: 403 for URL: http://34.239.106.178:9090/manager/text/list
        at sun.net.www.protocol.http.HttpURLConnection.getInputStream0(HttpURLConnection.java:1894)
        at sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConnection.java:1492)
        at org.codehaus.cargo.container.tomcat.internal.TomcatManager.invoke(TomcatManager.java:571)
        ... 20 more
org.codehaus.cargo.container.tomcat.internal.TomcatManagerException: The username you provided is not allowed to use the text-based
Tomcat Manager (error 403)
        at org.codehaus.cargo.container.tomcat.internal.TomcatManager.invoke(TomcatManager.java:704)
        at org.codehaus.cargo.container.tomcat.internal.TomcatManager.list(TomcatManager.java:876)
        at org.codehaus.cargo.container.tomcat.internal.TomcatManager.getStatus(TomcatManager.java:889)
        at org.codehaus.cargo.container.tomcat.internal.AbstractTomcatManagerDeployer.redeploy(AbstractTomcatManagerDeployer.java:173)
        at hudson.plugins.deploy.CargoContainerAdapter.deploy(CargoContainerAdapter.java:77)
        at hudson.plugins.deploy.CargoContainerAdapter$DeployCallable.invoke(CargoConta
        at hudson.plugins.deploy.CargoContainerAdapter$DeployCallable.invoke(CargoConta
```

How Servlets Work in Java

Solution:

Check the Tomcat credential in tomcat-usr.xml

Note:-

- When instance is launched then check IP address of Jenkins same in Manage Jenkins→ Manage configure→Jenkins
- Check IAM role is applied to instance which created.
- Check port SSH added in inbound rule of instance.
- Also Jenkins's default port is 8080 so make entry of that port in inbound rule of instance.

NEXUS INSTALLATION

Step-1 Create a directory in /opt with the name nexus.

Step-2 Create a user named nexus.

Useradd nexus

Step-3 Download the nexus zip using the following link.

https://www.sonatype.com/download-oss-sonatype

Step-4 Untar the downloaded package. There you will see two directories with name "sonatype-work" and "nexus".

tar -xvf nexus-3.16.1-02-unix.tar.gz

mv nexus-3.16.1-02-unix nexus

Step-5 Change the owner of both the directories by nexus user.

chown -R nexus:nexus /opt/nexus

chown –R nexus:nexus /opt/sonatype-work

Step-6 Go to the nexus directory -> then go to bin,

There open the file nexus.rc

Uncomment the line #run_as_user=""

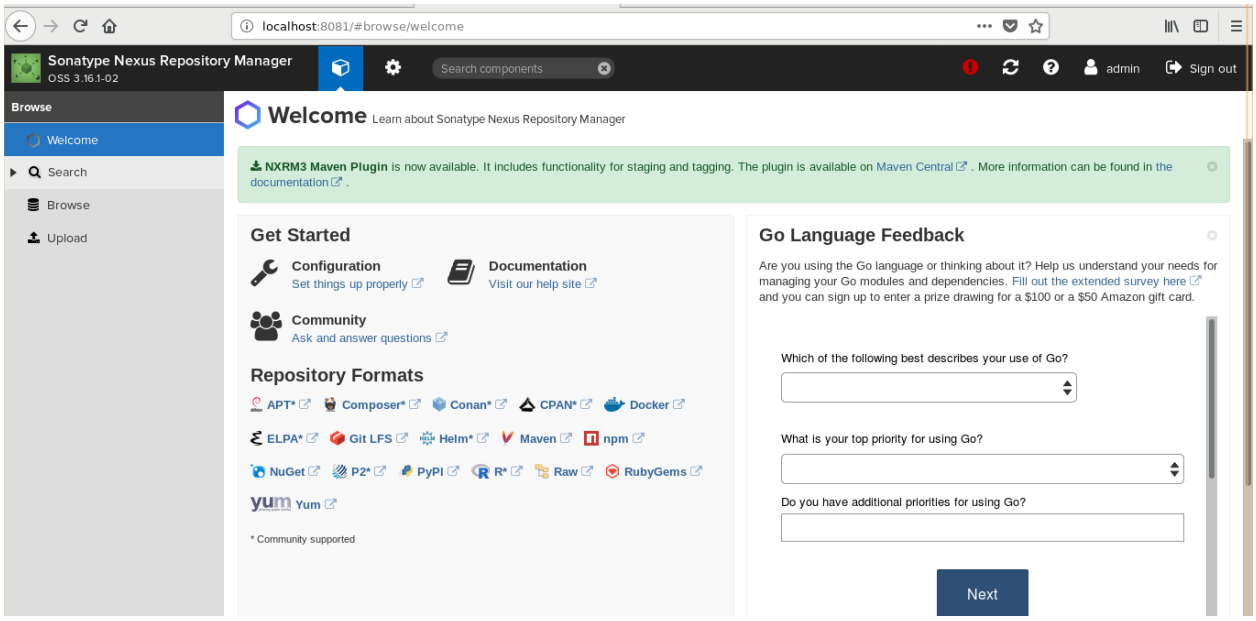
Then enter the user nexus as run_as_user="nexus"

Step-7 Run nexus

Inside bin directory run the command as./nexus run

This will start the nexus, after sometime the nexus will be up and running.

In the browser enter the url as http://localhost:8081



You will be able to see the webpage as above.

NOTE: Make sure your system should have enough memory (2 GB and above) to run nexus as well as latest version of java jdk compatible with it.