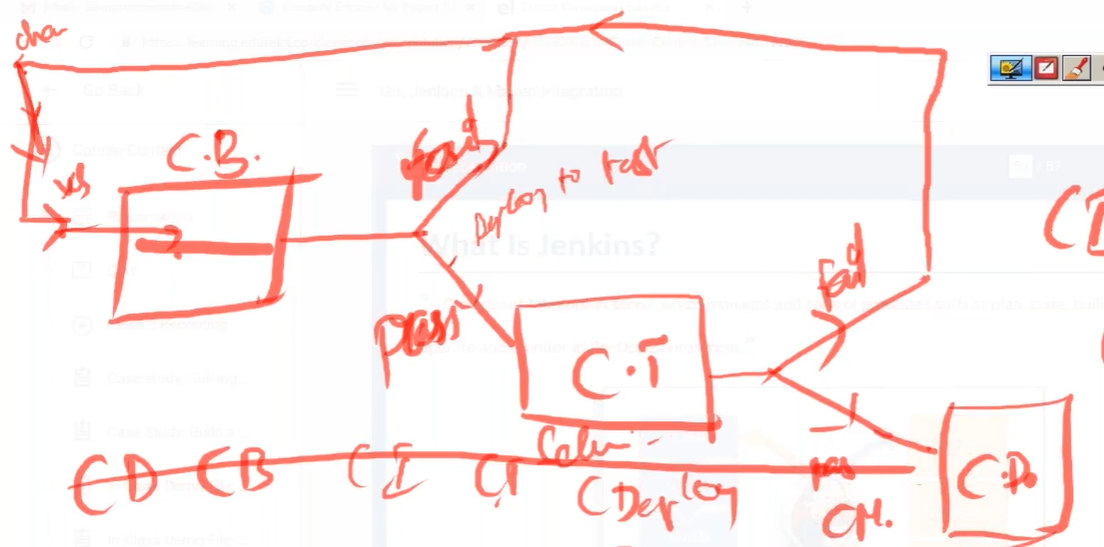
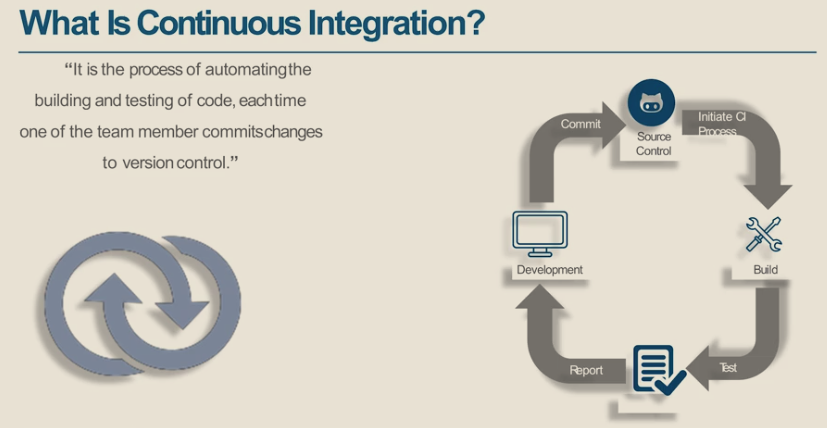
* **How we are solving waterfall model’s problem?**
* In water fall methodology, complete software is delivered at once after so long time when the client doesn’t need the product at all.
* By using agile we are solving waterfall model’s problem.



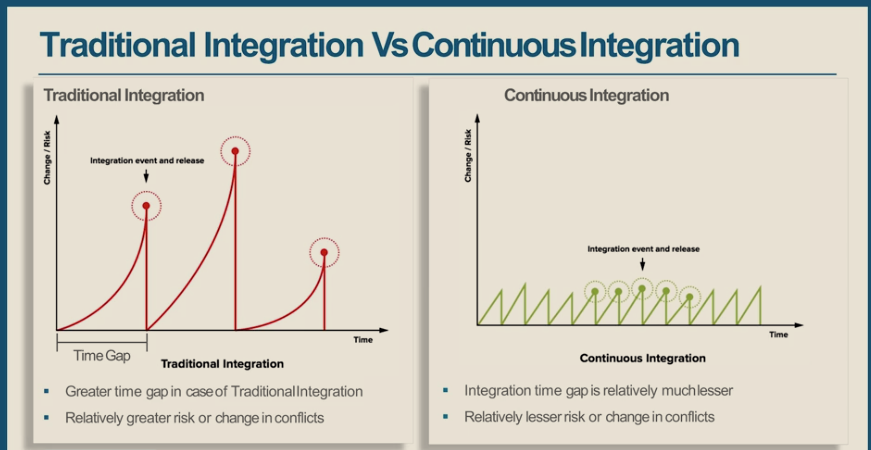
* **Devops consist of following pipeline**
* Continuous Development (CD)🡪Continuous Build (CB)🡪Continuous Integration (CI)🡪Continuous Testing (CT)🡪Continuous Deployment (CD)🡪Continuous Monitoring (CM)
* CB🡪CT🡪CD This is called as **Continuous Integration** and it is achieved by using continuous integration server called as **Jenkins.**
* Agile consist of continuous development (CD), continuous Build(CB), continuous Testing(CT) but there is no point of continuous deployment.



* **What is continuous Integration**



* **Differences between traditional and continuos integration.**



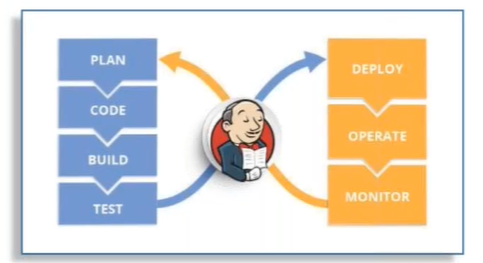
* In traditional integration time gap between the integration or releases are `more, whereas in the continuous integration the time gap is very less, and the integration is frequent.
* By using Continuous integration identifying the bug become faster, quality to product can obtain.
* **Importance of continuous integration**



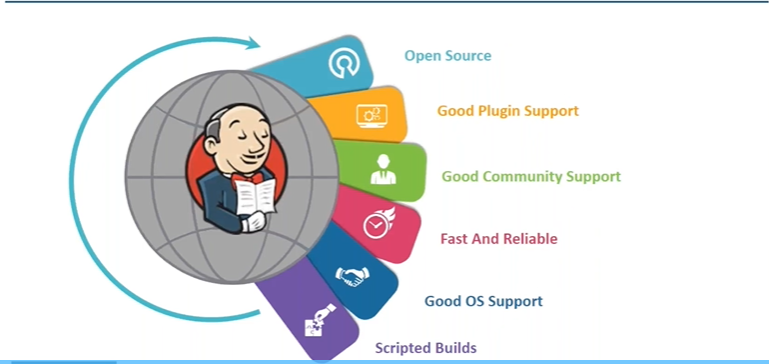
* **Popular Continuous Integration tools**



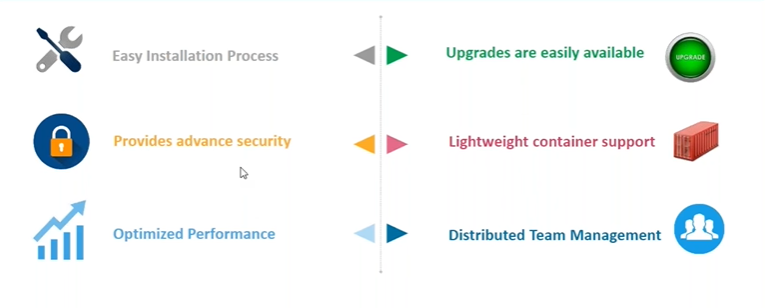
* **What is Jenkins?**
* Jenkins is developed using Java, its java application, which is deployed on some application server, but it won’t restrict to work with any technology.
* Jenkins is a continuous integration server which manages and control processes such as plan, build, test, deploy, operate and monitor in Devops Environment.
* Jenkins is open source, plugin-based tool (for any functionality plugin is there e.g git, docker, log, notification, maven etc.)
* Jenkins is called as the heart of CI/CD pipeline. It is also called as orchestrator.
* Jenkins act as a manager which manages the different jobs. (like start job, scheduling jobs, managing job etc.)
* It just uses some tools to perform the jobs like in CI/CD pipeline for Continuous Build (it uses Maven) for continuous testing (it uses selenium) for another job it uses respective tools.
* Jenkins, we use to automate the process like how we do from server (mvn compile) in local manually but the same we can automate by using Jenkins by creating a compile job.



* **Why Jenkins in demand ?**



* **Features of Jenkins**



* **Role of Jenkins in Devops**

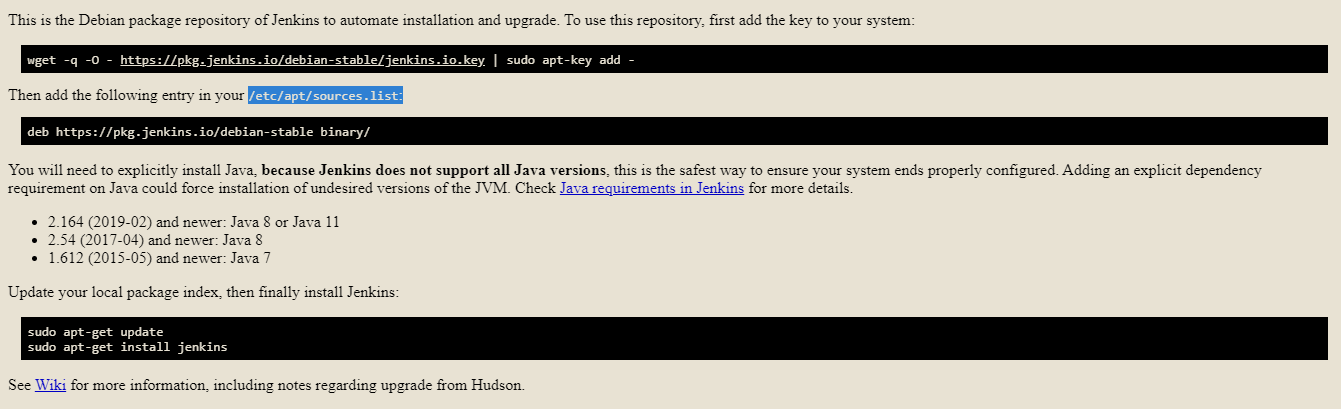


* **Jenkins Installation steps**

1. **Update the repository : sudo apt-get update**
2. **Install java jdk by using** : **sudo apt-get install openjdk-8-jdk**
3. **Validate:**  **Java -version**
4. **Check java Home Path: which java output: usr/bin/java**
5. **Now set the java home variable: vim /etc/profile**



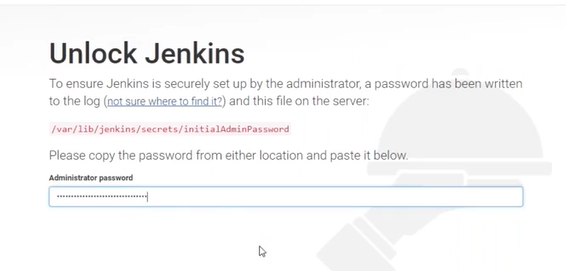
1. **To execute the profile file: source /etc/profile**
2. **Install the wget package if not already installed: apt-get install wget**
3. **Now go to the Jenkins documentation website and get the command to get the Jenkins installed** : <https://pkg.jenkins.io/debian-stable/>
4. **We will get command something like this based on the os you are working with.**



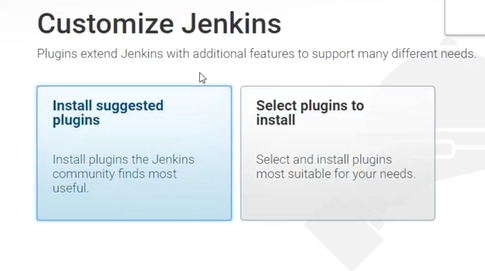
1. **By adding the “**[**https://pkg.jenkins.io/debian-stable binary/” in /etc/apt/sources.list**](https://pkg.jenkins.io/debian-stable%20binary/)

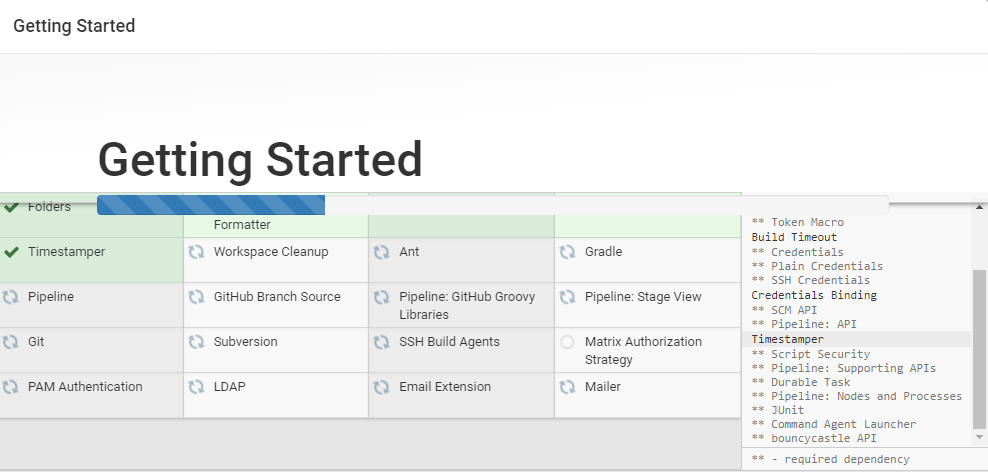
**It will download the Jenkins packages by referring this sources.list file, and when we do “apt-get update” that file executes and update all the latest url.Now the Jenkins is installed.**

1. **To start the Jenkins service: “systemctl start Jenkins”( this command will start the Jenkins service)**
2. **Jenkins by default runs on port 8080**
3. **Get the ip of the machine: ip addr something like(192.168.X.X)**
4. **Go to browser and launch Jenkins by using machine ip and port number like this 192.168.X.X:8080**
5. **Jenkins application will be launched with locked mode which need to be unlock by getting the password stored at /ver/libs/Jenkins/secrets/initialAdminPassword location**



1. **Now get the password and login into the Jenkins application.**
2. **Click on the install suggested plugins for the initial setup.**





1. **Once plugin download is done click on the save and continue. You will get below screen**

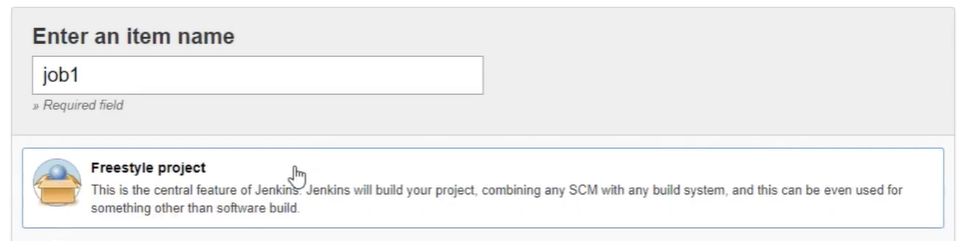
* **Jenkins first screen**

A screenshot of a cell phone

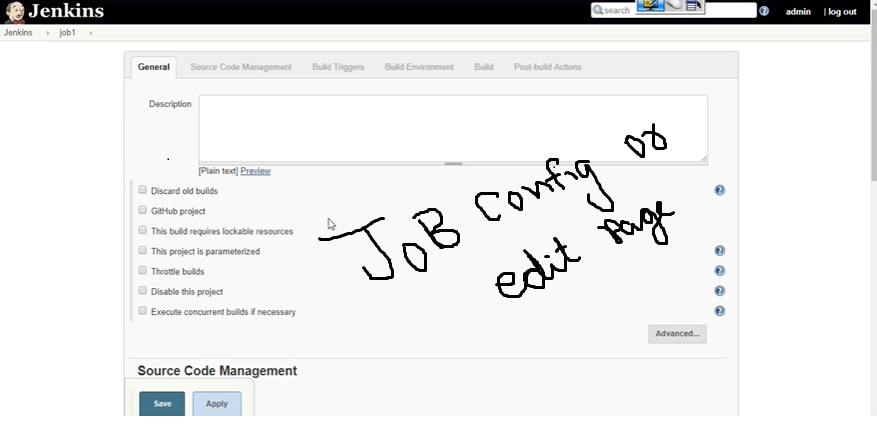
Description automatically generated

* While you login very first time on Jenkins you will see the Jenkins dashboard as showed in above image.
* On the dashboard there is an option for creating new jobs, this is nothing, but the jobs/project/new Item creation terms are interchangeable.
* **Creating Job in Jenkins**

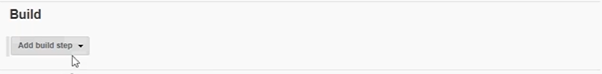
1. Click on the **New Item** on the left side panel.
2. Enter the **Job name** and select job/project type for example select **freestyle project** and click on OK.



1. Once you will click on **OK,** then we will get another page/screen where you must configure the job. (These options is available based on plugin we have added)

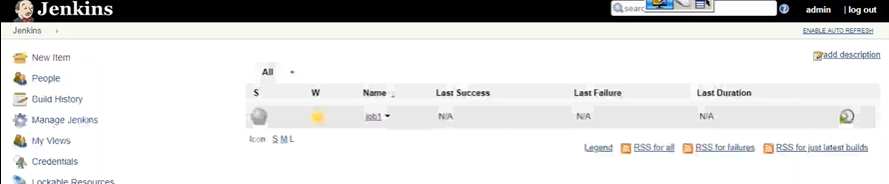


1. Here we are running a shell script in that job, for that we need to click on the **Build** tab and then click on **Add build step** a drop-down list would be open there select **Execute Shell.**



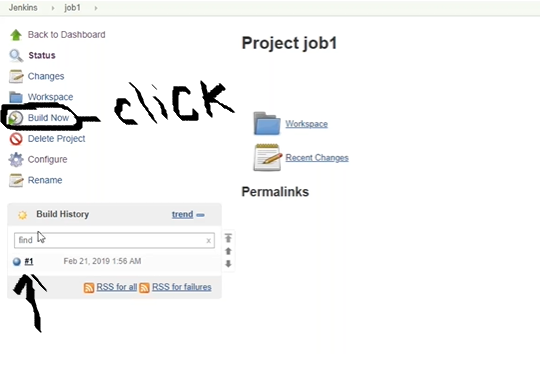


1. Once you will click on Save a job would be created and the same can be seen on the dashboard.

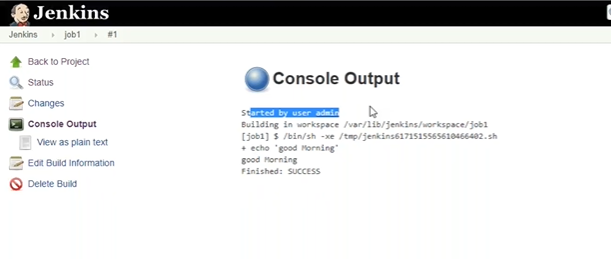


* **Execute/Start the job**

1. Select the job from the dashboard.
2. Then click on the **Build Now** option from the left side panel. This will start or run the job.



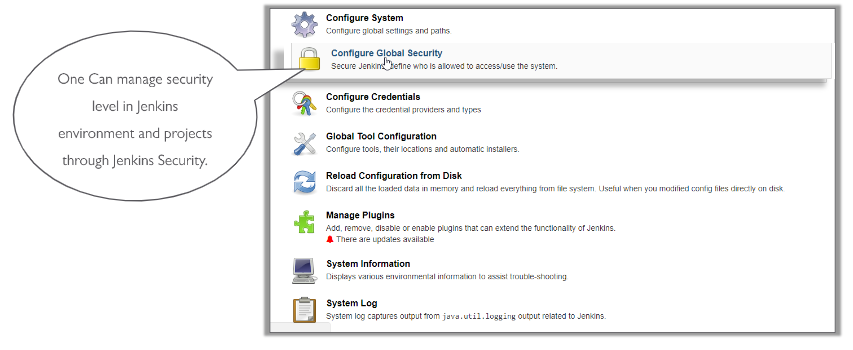
1. Once job is completed, we can see the job id (like #1) under **Build History**.
2. Now to see the build log click on the job id, a new screen will open where we must click on the **Console Output.**  It will show the job execution output details.



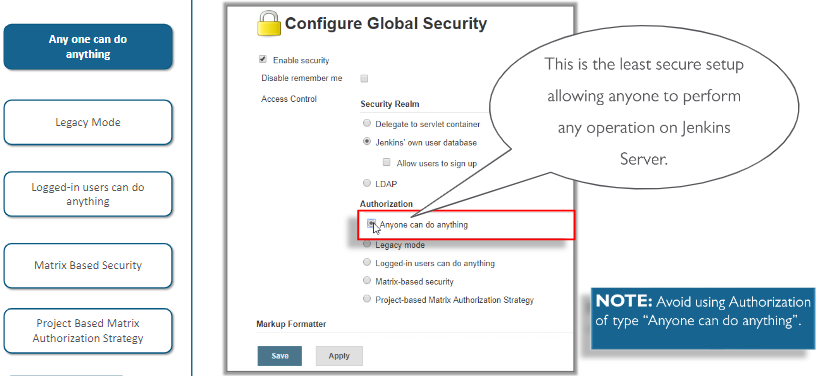
* **Who execute this shell script, Is it Jenkins?**
* No, this shell script is executed by the shell program on the server. Jenkins is just starting the job, triggering the job or managing the job.
* For example, Compilation of code will not be done by the Jenkins, it’s done by some tools like Maven. Jenkins will just trigger that job and will manage in the pipeline.
* Actual job is done by the program installed on the server Jenkins just start or manage the job.
* **Q. Does all the job run on the same server?**
* No, all the job does not run on the same server, we can run the job on the remote server as well. There is the concept of **Jenkins Master Slave concept**.
* **Jenkins Security Management**
* We need security because its distributed across the team.
* We need to configure, who is allowed to access/use the system.

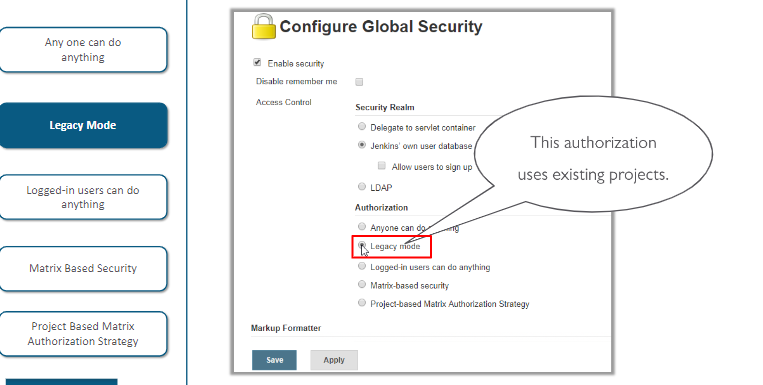
Steps to configure Security

1. Go to Jenkins’ dashboard. Click on **Manage Jenkins** on left side panel. You will get the following screen.



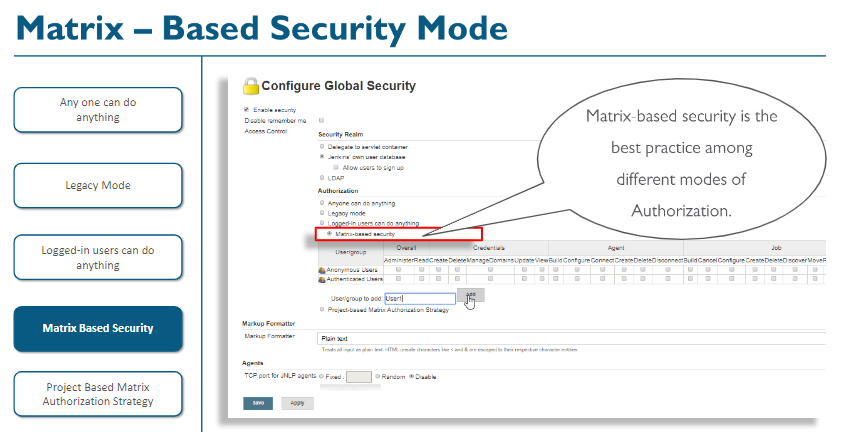
* Anyone who has the ip address and port number details can do anything which is not good practice.





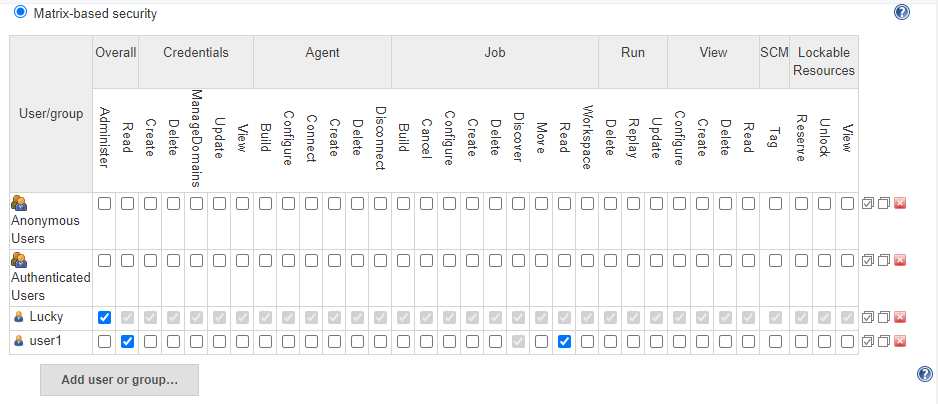


* **Matrix Based Security**
* When we have a team there might be different role of every individual like developer, tester, admin they need different permission on different jobs then we need to go for the Matric based security mode.

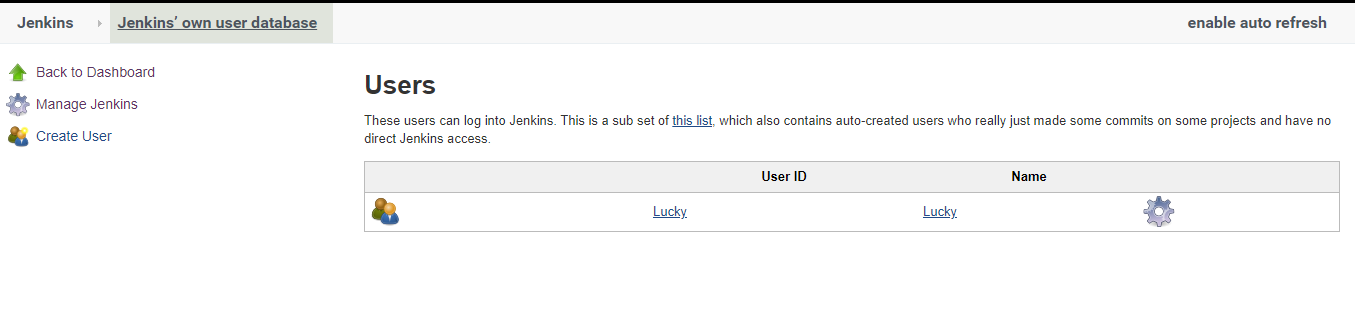


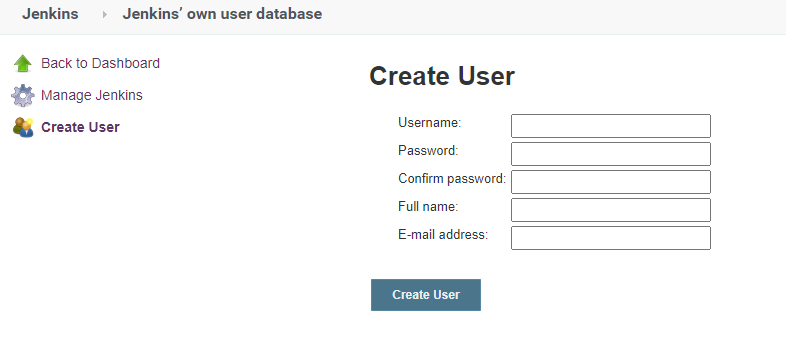
* Add the user or user group and tick mark the permission need to give the user. Can seen in below image, has added (Lucky )as a user and given all the user.
* For example, in the team one member(user1) is monitoring the job so he/she needs read only access.

|  |  |  |  |
| --- | --- | --- | --- |
| **Job Name** | **User1** | **User2** | **User3** |
| **Job1** | Read | Read | All |
| **Job2** | Read | All | All |



* **How to create user?**
* On dashboard click on manage Jenkins and select Manage users.
* To create new user, click on Create User on the left side panel and fill all the details.

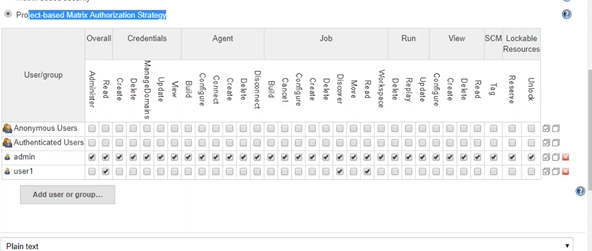




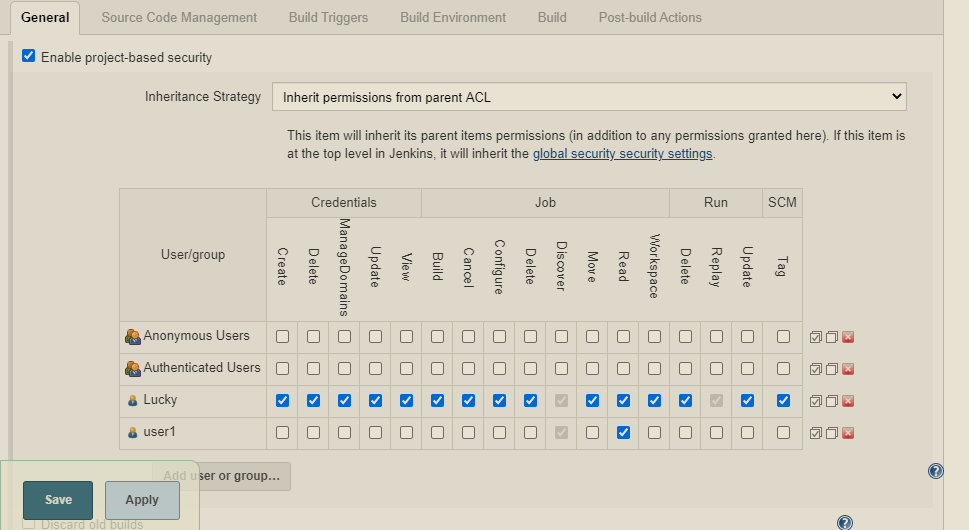
* **Project Based Matrix Authorization Strategy**
* Let’s take an example, we have team consist of different profile like tester, developer and admin and we have 3 project(jobs)
* Same user having different permission across different job. This is possible using Project/Job Based Matrix strategy.

|  |  |  |  |
| --- | --- | --- | --- |
| **Profile** | **Job1(development job)** | **Job2(test job)** | **Job3(Deployment job)** |
| **Developer** | All | Read | Read |
| **Tester** | Read | All | Read |
| **Admin** | Read | Read | All |

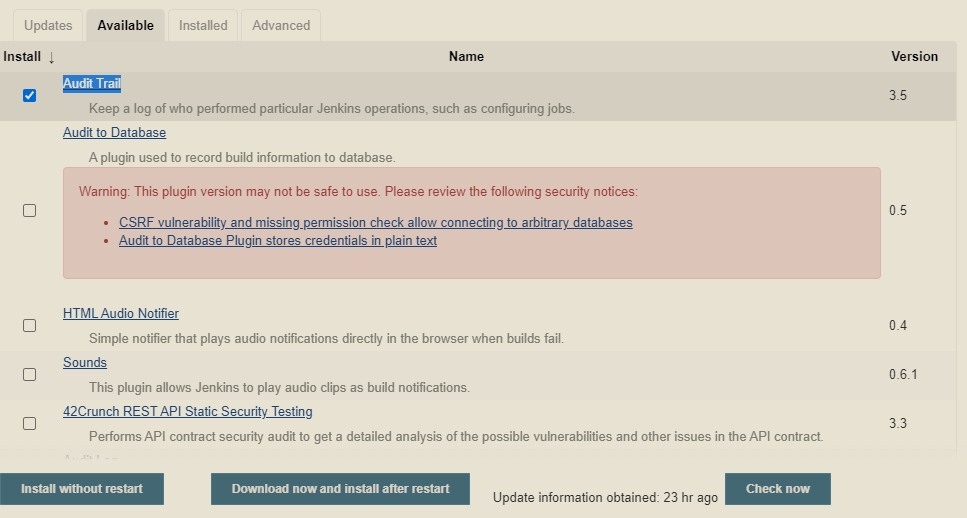




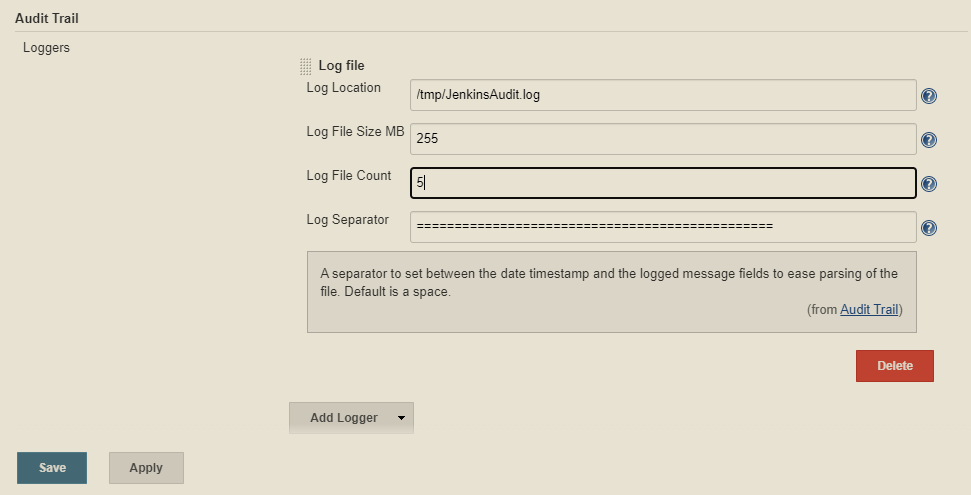
* Once above step is done then we can add access to particular project.
* **Note:** Project based access is given under each job/project. For that click on the particular project/job and click on configure then click on general tab and tick mark on **Enable project-based security**.



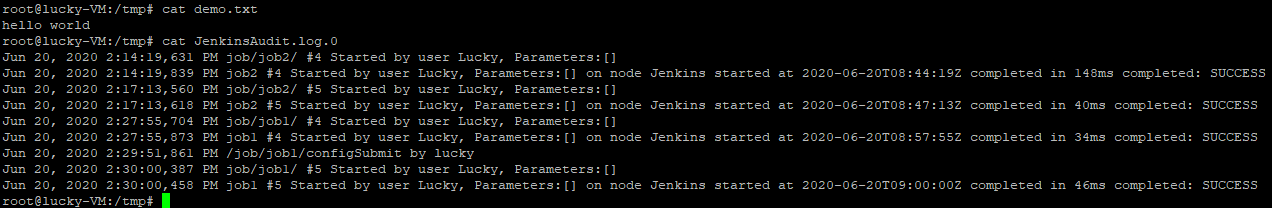
* **Plugin Manager**
* Plugin manager is used to include the functionality.
* **Auditing record (**[**Audit Trail**](https://plugins.jenkins.io/audit-trail) **Plugin)**
* For example for auditing purpose we can install “[**Audit Trail**](https://plugins.jenkins.io/audit-trail)” which can help to achieve to keep track of the user activity on Jenkins system.



* Click on Install and once installation is done then go to the dashboard and click on manage Jenkins and then select Configure system where we have to mention the log file location on the Jenkins server. We have set location under /tmp folder



* We can check the audit log by going to the mentioned location under the system configuration for audit trail. (like /tmp)



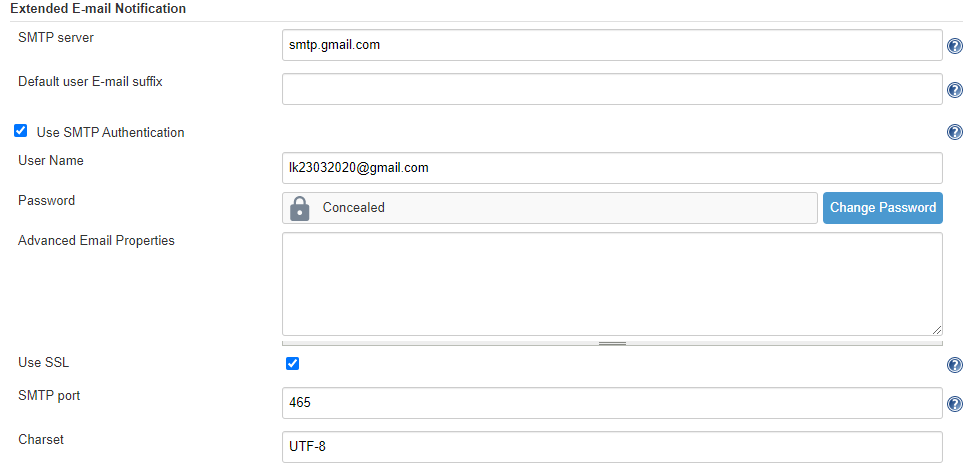
* **Notification plugin**
* When we are creating a pipeline in CI/CD each job is executed upon the successful of the previous job.
* Let’s assume some job get failed in the pipeline then the CI/CD pipeline would be terminated so to notify the concern team via email we need notification plugin to be configured on each job.
* To send notification Jenkins use SMTP (Simple mail transfer protocol) server. Every organization has their own smtp server.

**How to configure**

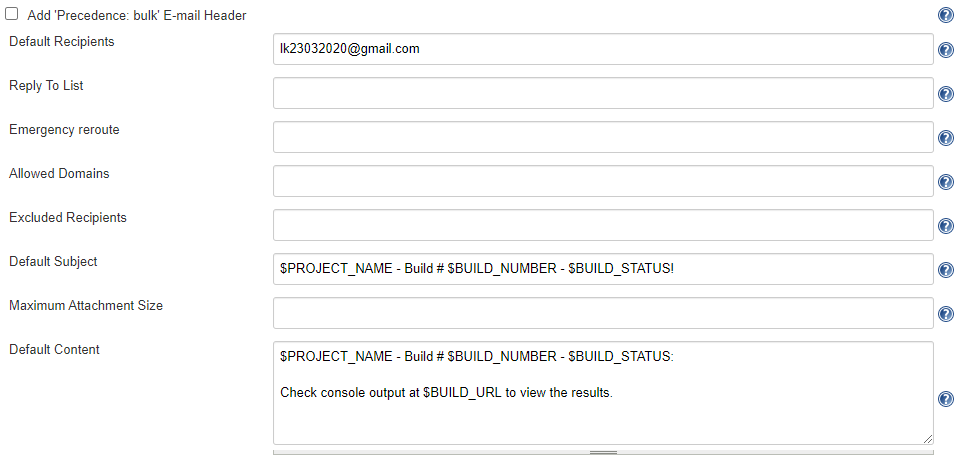
1. Go to dashboard and click on the Manage Jenkins, inside that click on configure system.
2. Go to the **Extended Email Notification** setting and enter the details.



1. Mention the SMTP server address (here we are using Gmail smtp server so mentioned as smtp.gmail.com port:465)
2. Then click on Advance button and set the required field as below.

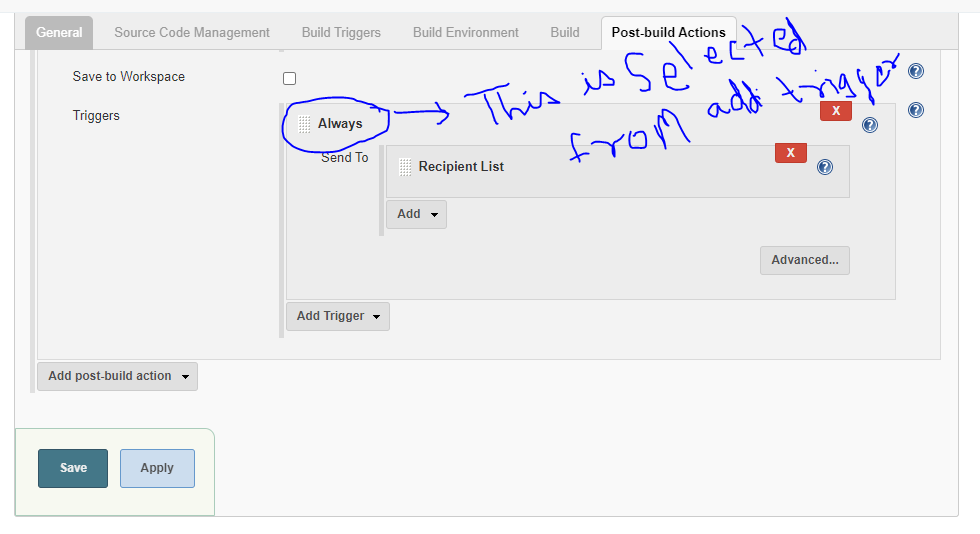


1. Then add the default recipient email address and save it.



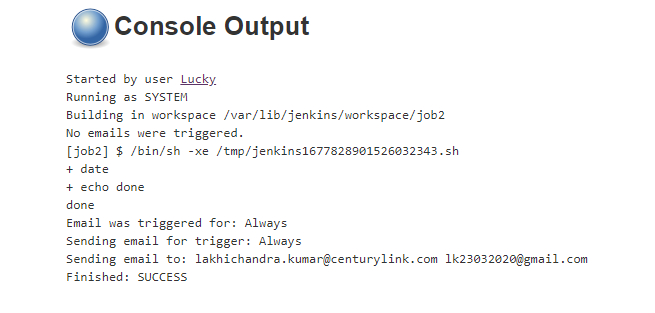
**Note:** $variable\_name is used as the global variable in Jenkins.

1. Then go to the respective job where you want to integrate notification post build. Click on the **Post Build Action** tab and enter the recipient list and then select when to trigger from the **Triggers** drop down.



Blue encircled **Always** is selected from the **Add Trigger** drop down.

1. We can verify the email sent or not by looking the build log output.



* **Configuring Microsoft Team for Notification**

**Note:** Firstly create a group on the Microsoft Team application and then right click on the general tab inside the group name as showed in below image.

Then select the **Connectors** option and follow the steps mentioned there.

# 

Top of Form

**Name**Enter a name for your Jenkins connection.



**Webhook URL**Copy the following URL to save it to the Clipboard. You'll need this URL when you go to the Jenkins website.

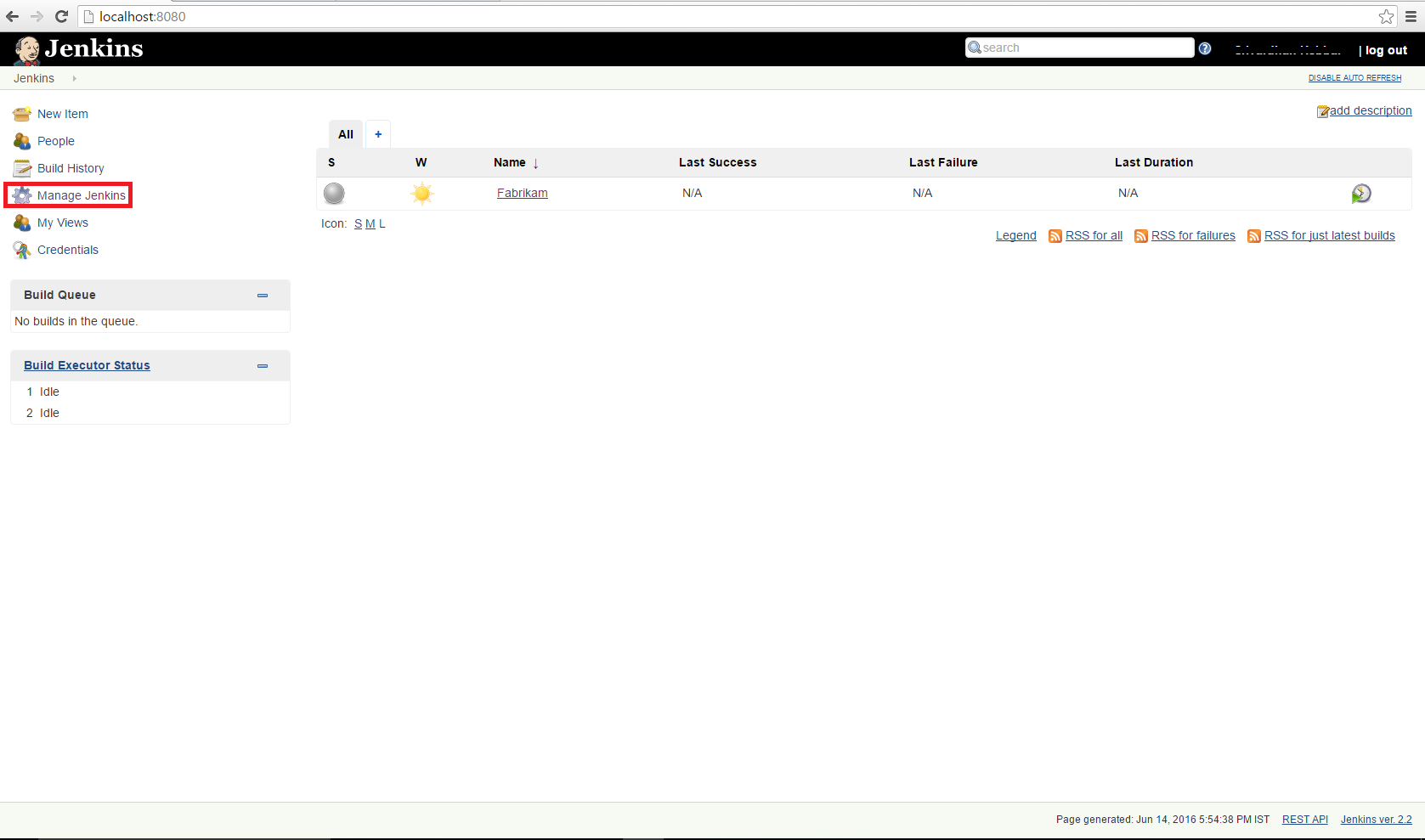
Notifications will be sent about the following events in Jenkins:

* Whenever activity occurs in Jenkins.

Follow these steps to create your Jenkins connector. 

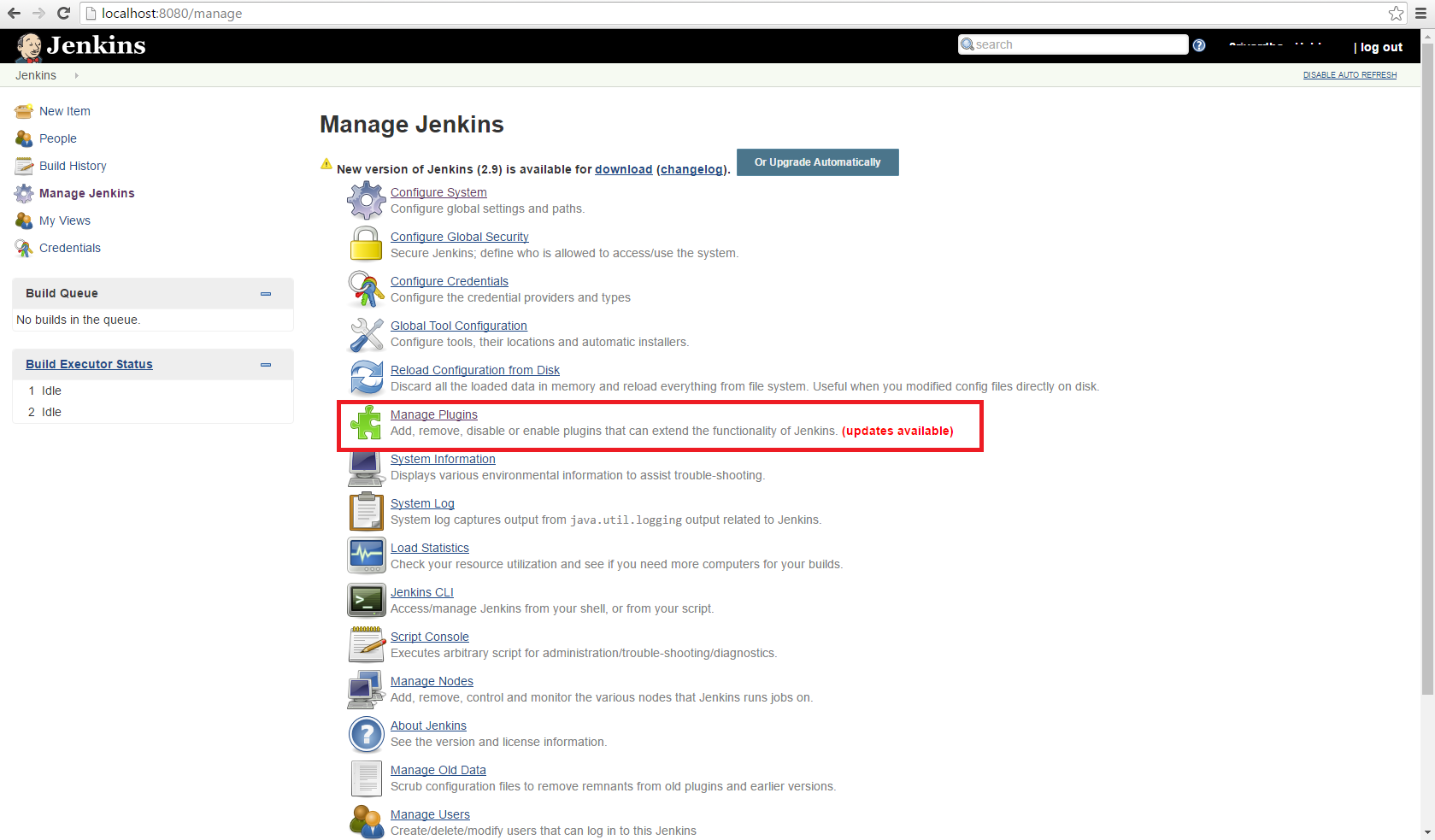
**Step 1**

Log into Jenkins and in the Jenkins dashboard (Home screen), click **Manage Jenkins** from the left-hand side menu.



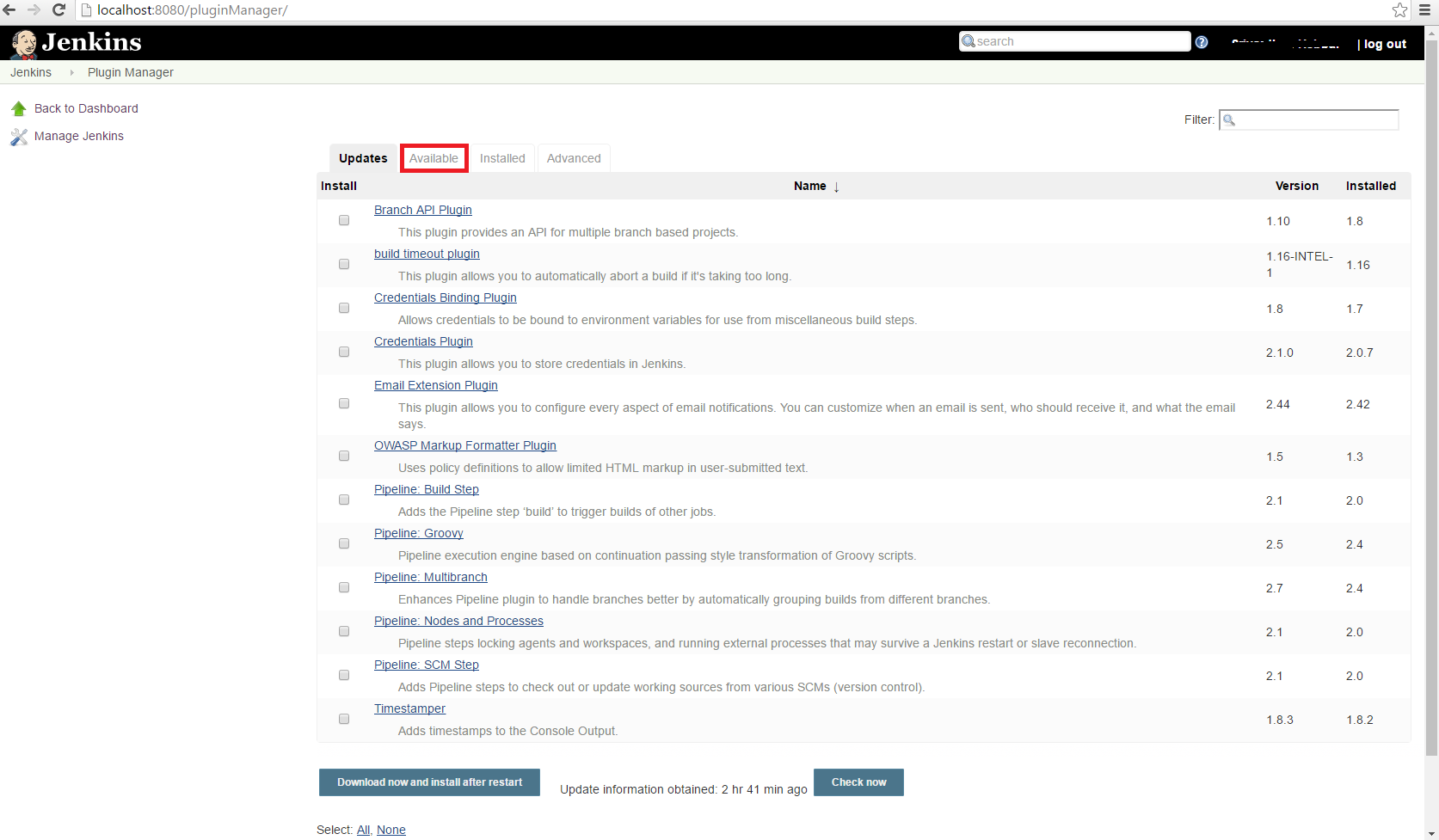
**Step 2**

Click on **Manage Plugins** from the right hand side.



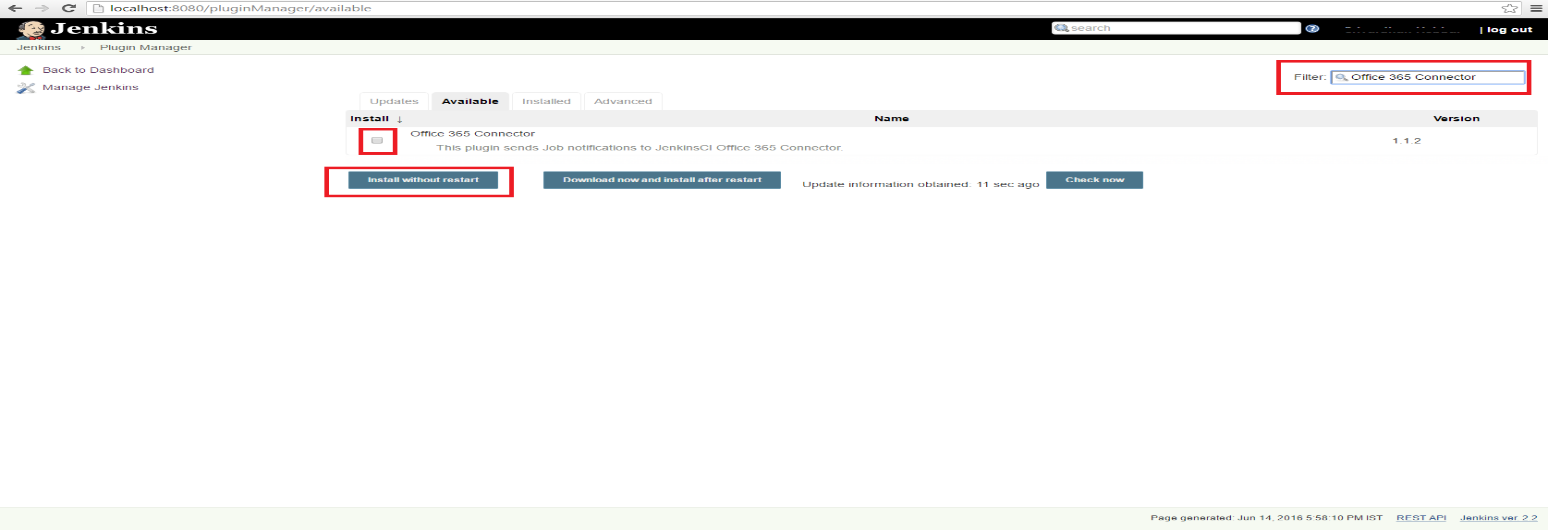
**Step 3**

Click on the **Available** Tab.



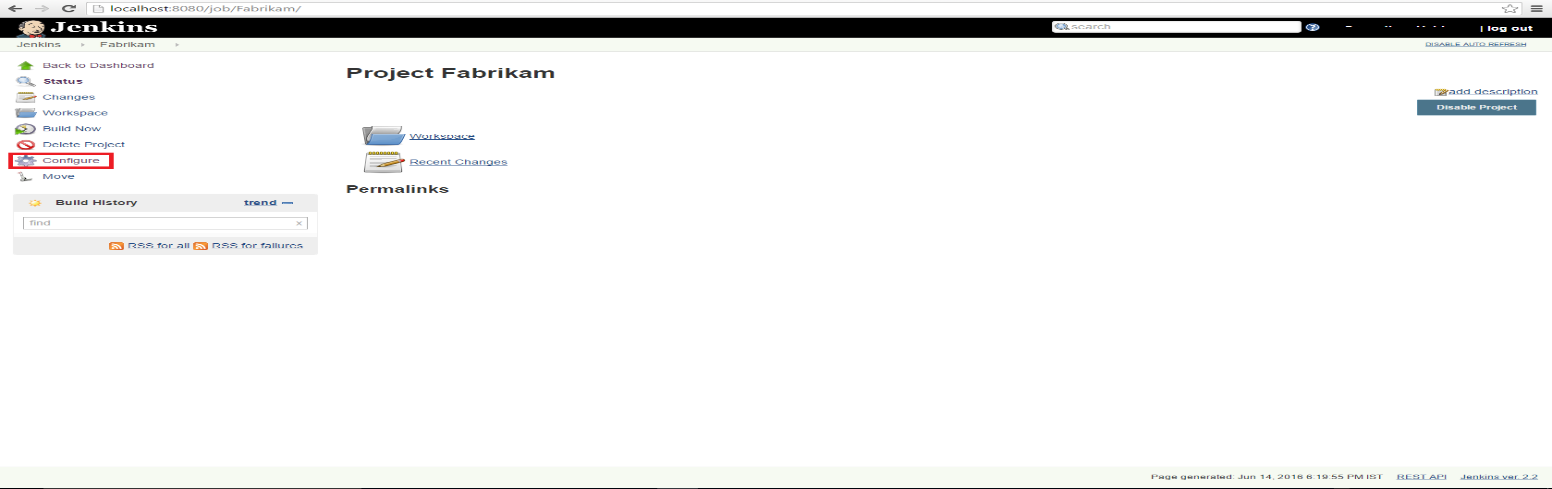
**Step 4**

Search for **Office 365 Connector** and then check the checkbox and click the **Install without restart** button.



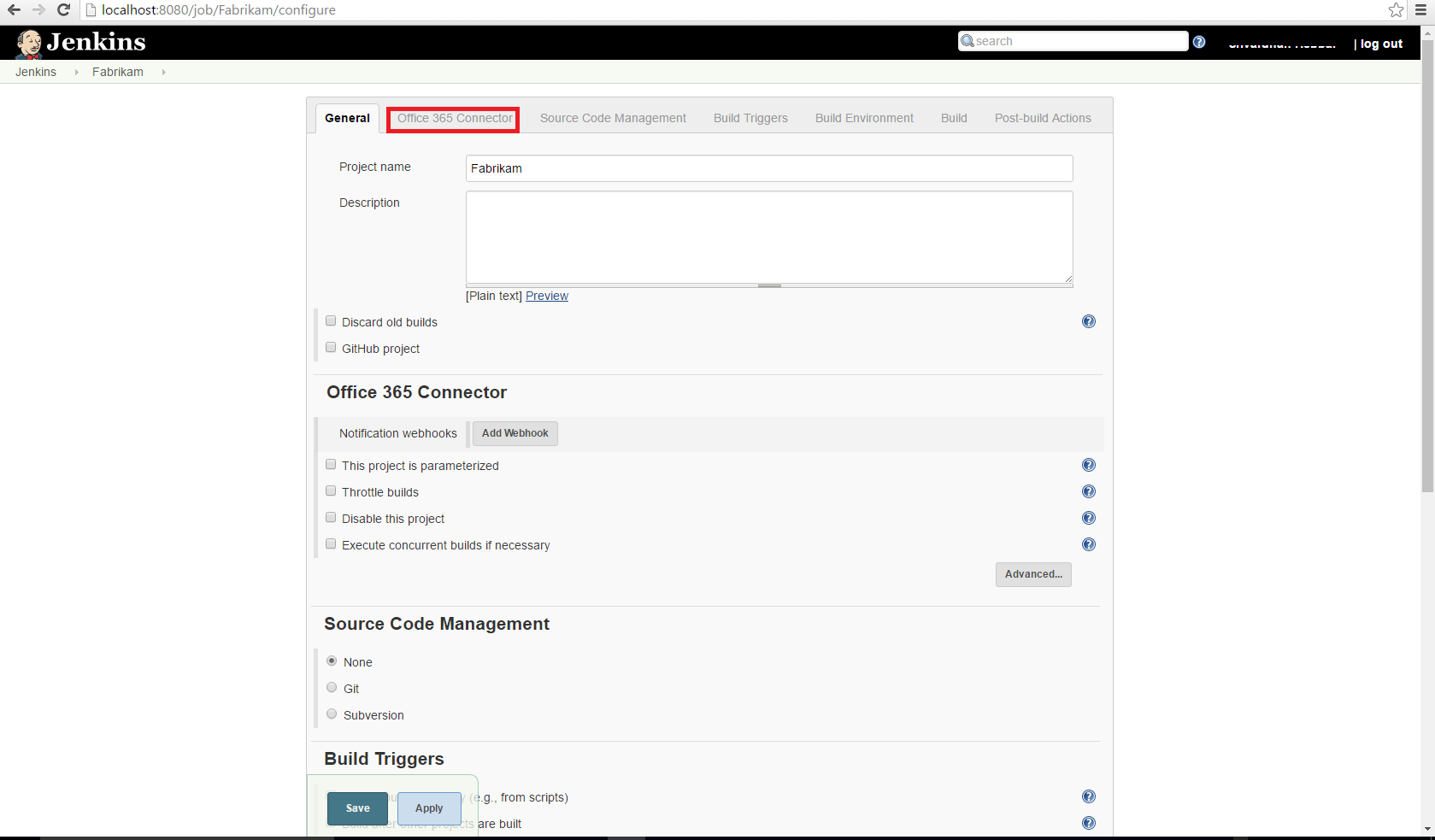
**Step 5**

Go to your project and click on the **Configure** button.



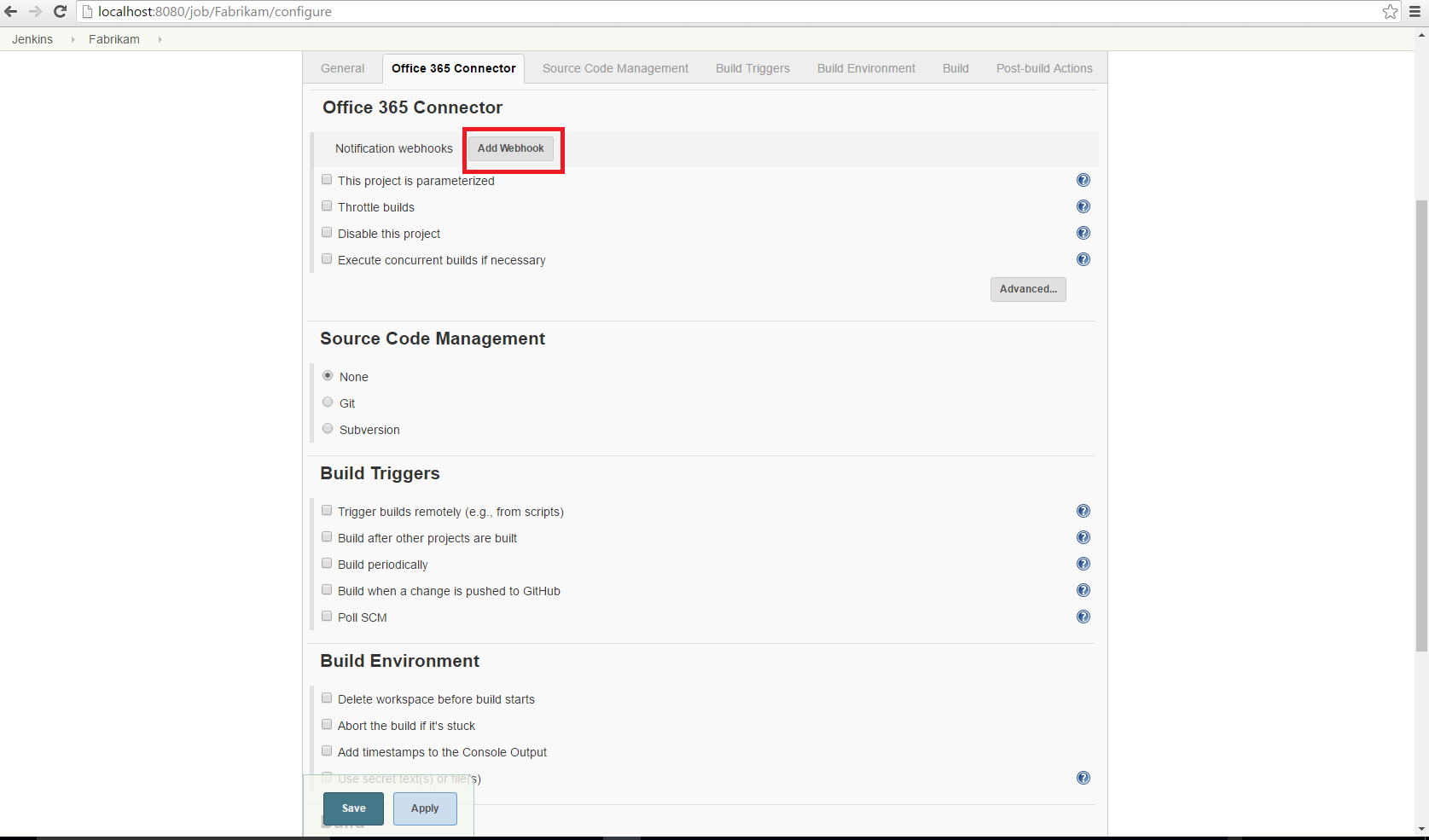
**Step 6**

Click on the **Office 365 Connector** tab.



**Step 7**

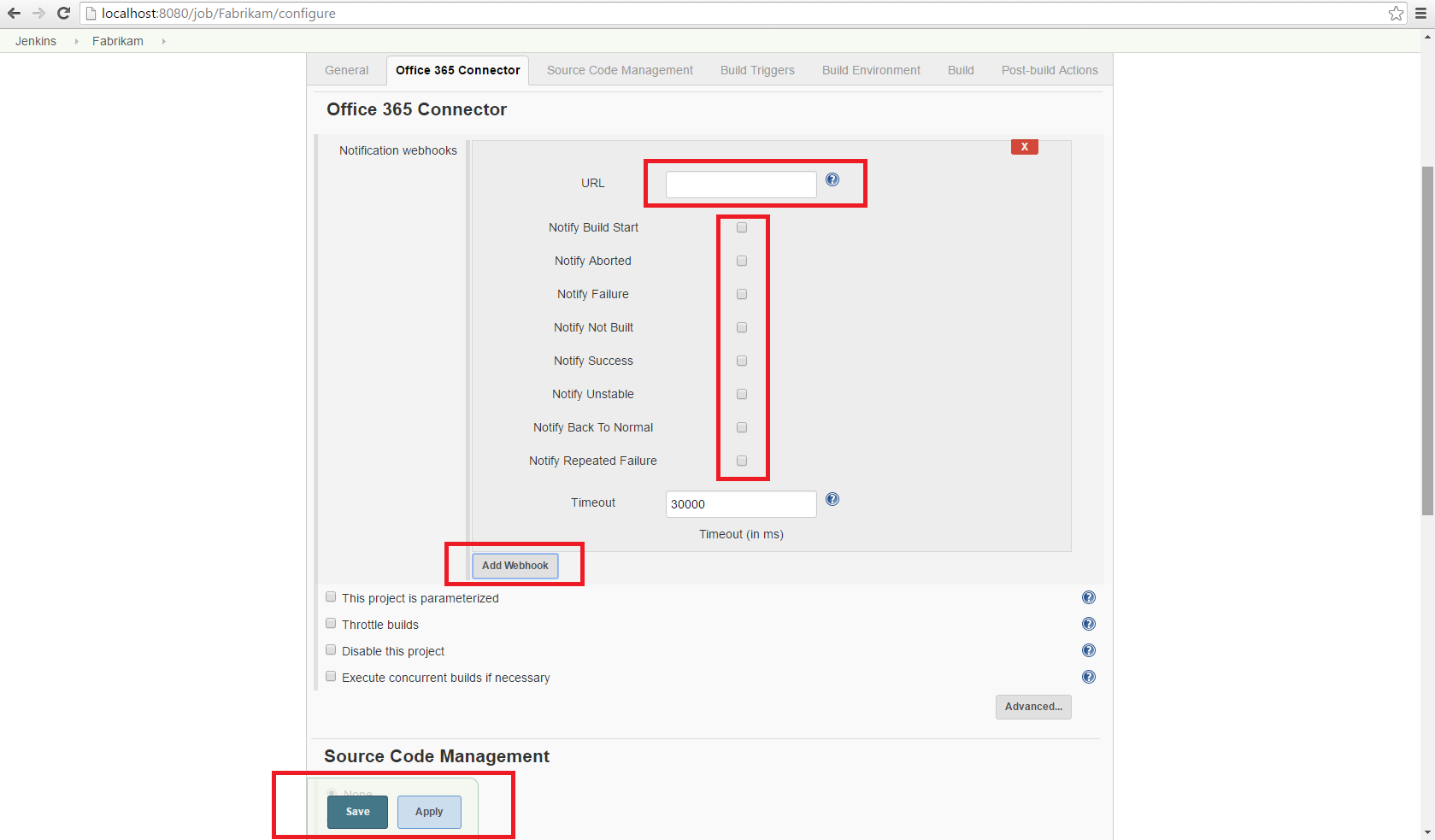
Click on the **Add Webhook** button.



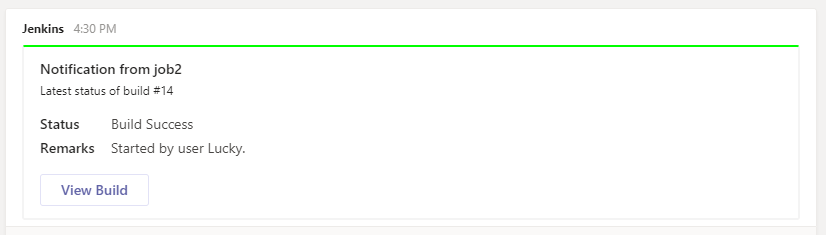
**Step 8**

Paste the webhook url in the **URL** box and check for all those boxes for which you want to receive events and then click the **Save** button.



* You will get notification something like as below.

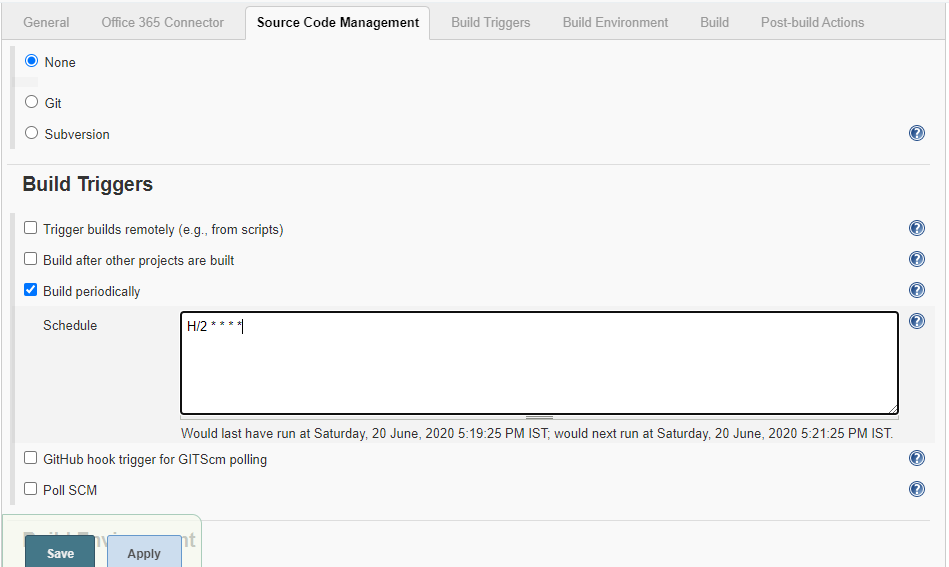


* **Job Scheduling**
* So far, we are running the job manually by select the job and click on the build now.
* We can automate this task by using the concept of **Job Schedule.**

**There are different ways to schedule the job like**

1. **Timer (Cron)**
2. **Poll** **SCM** (detect when any commit happens from the GitHub)
3. **Pipeline** (In CI/CD pipeline one job starts another job)
4. **Timer**

* Go to the job and click on **configure.**
* Click on the **Build Trigger** Tab and select **Build Periodically.** It’s basically a **cron** tab where we can configure the **cron** timer.
* Based on the timer configuration, it will trigger the job automatically.



**Note::**

This field follows the syntax of cron (with minor differences). Specifically, each line consists of 5 fields separated by TAB or whitespace:

MINUTE HOUR DOM MONTH DOW

|  |  |
| --- | --- |
| MINUTE | Minutes within the hour (0–59) |
| HOUR | The hour of the day (0–23) |
| DOM | The day of the month (1–31) |
| MONTH | The month (1–12) |
| DOW | The day of the week (0–7) where 0 and 7 are Sunday. |
|  |  |

Examples:

# every fifteen minutes (perhaps at :07, :22, :37, :52)

H/15 \* \* \* \*

# every ten minutes in the first half of every hour (three times, perhaps at :04, :14, :24)

H(0-29)/10 \* \* \* \*

# once every two hours at 45 minutes past the hour starting at 9:45 AM and finishing at 3:45 PM every weekday.

45 9-16/2 \* \* 1-5

# once in every two hours slot between 9 AM and 5 PM every weekday (perhaps at 10:38 AM, 12:38 PM, 2:38 PM, 4:38 PM)

H H(9-16)/2 \* \* 1-5

# once a day on the 1st and 15th of every month except December

H H 1,15 1-11 \*

* We can see the job is started by **timer**



1. **Poll SCM**

* Click on the configure job and select the **Source Code Management** (**SCM**) Tab.
* Select the Git radio button and provide the GitHub repository URL.

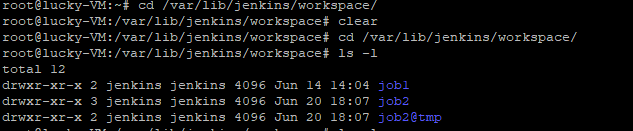
**Note: for working with git ensure the git is installed on the Jenkins server.**

(apt-get install git)

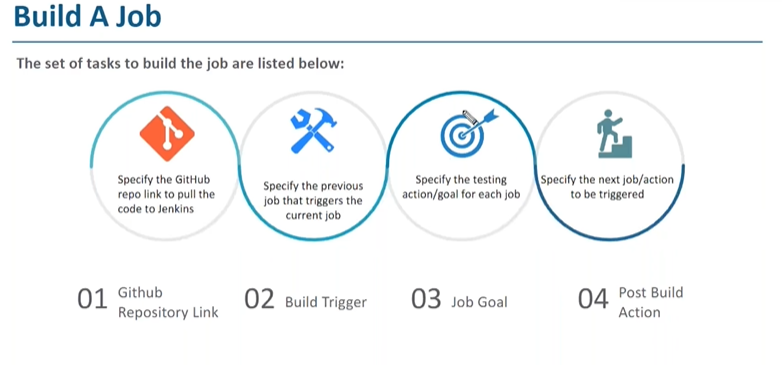
Verify by Linux terminal, type command git, it will return something related to git.

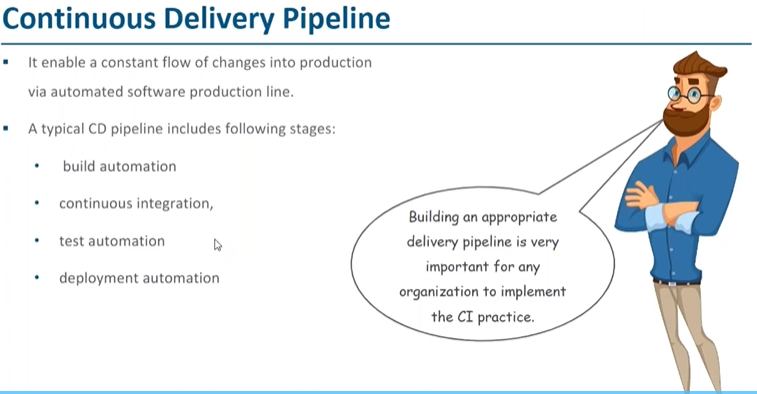
if it’s not install it will say invalid command git.

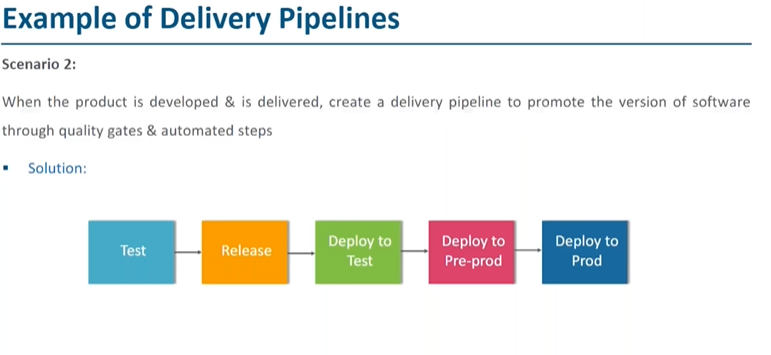
* **Workspace in Jenkins**
* The default location on the Jenkins server for the job is as /var/lib/Jenkins/workspace.
* Workspace is the directory which is used by the job.



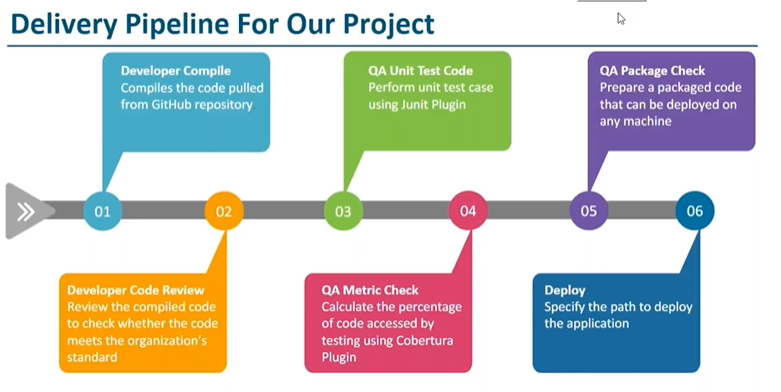
* **/var/lib/Jenkins/workspace/job\_name** works as the **local repository** on Jenkins.







* **Here each step is job. Which represent different phases.**



* **Edureka GitHub repository**

[**https://github.com/edureka-git/DevOpsClassCodes**](https://github.com/edureka-git/DevOpsClassCodes)

* **Maven (Build Automation tool)**
* Like we do from command line manually by going to the project directory and executing command **“mvn compile”.** Here compile is called as goal or target.
* Goal is executed by plugin.
* All the maven project consists of **pom.xml** which is like heart of maven, by referring that file only maven get compile and build instruction.
* Target or goal is present in the pom file.

**e.g.**

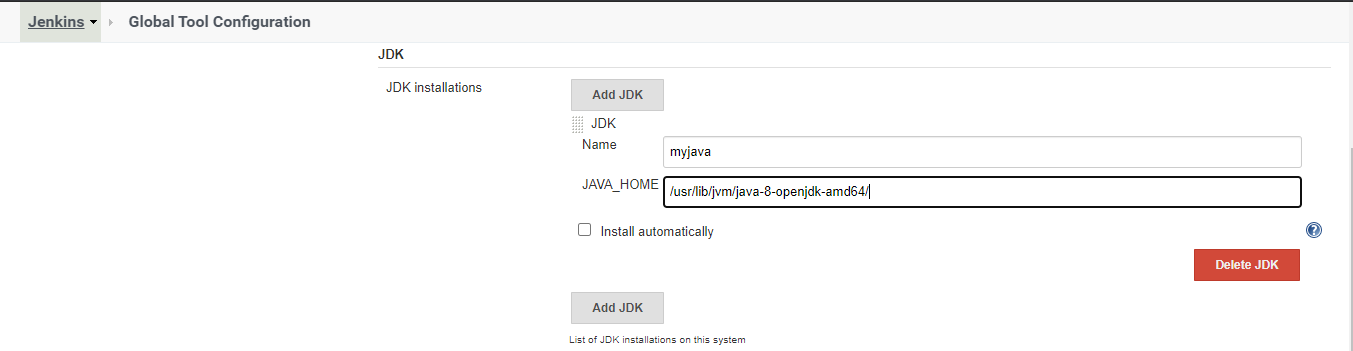
1. **Compile**- mvn compile (here compile is the goal or target) [ we need to install maven plugin]
2. **Code review –** mvn -P matrics pmd:pmd **(** PMD **(Program Mistake Detector))**
3. **Unit testing –** mvn test (**Junit plugin**)
4. **Code coverage –** mvn cobertura:cobertura -Dcobertura.report.format=xml **(Cobertura)**
5. **Packaging-** mvn package

**Note:** for compile job make sure java,git and meven is installed in jenkins server machine.

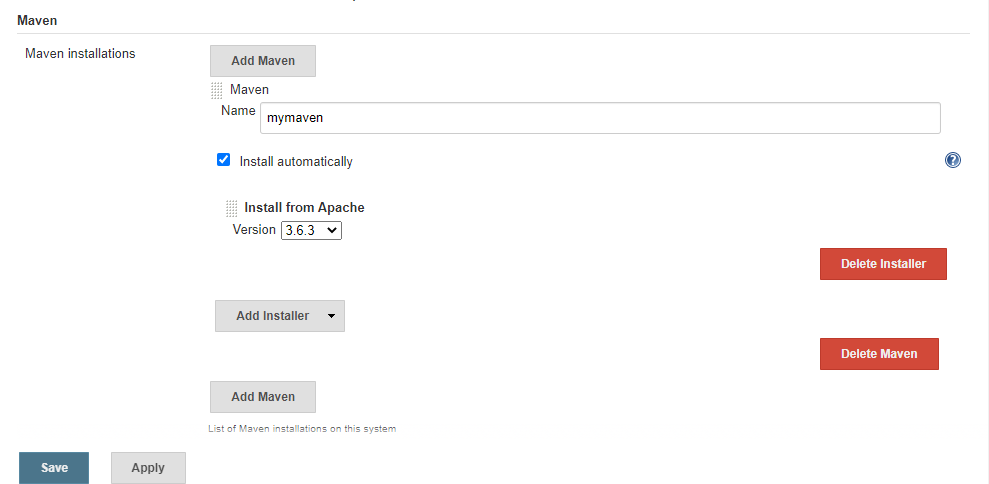
(**apt install maven** – to install the maven

**which maven**- to know the location of the installation directory)

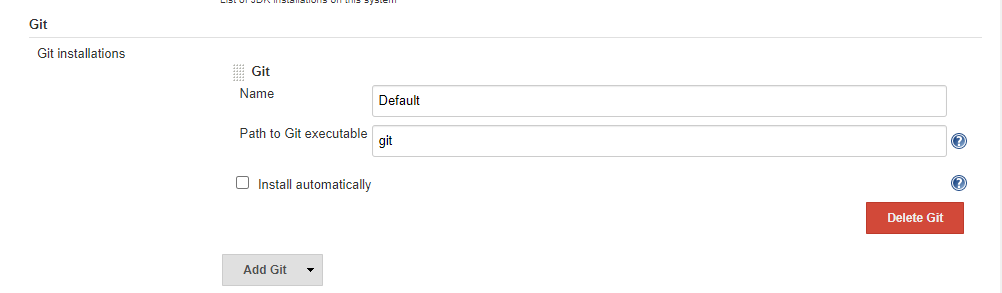
* We can install java either on the jenkins server manually by command line or by using the jenkins GUI.
* For setting the environment variable for JDK go to the **Mangae Jenkins** and select **Global Tool Configuration**.
* JAVA\_Home location is **/usr/lib/jvm/java-8-openjdk-amd64**



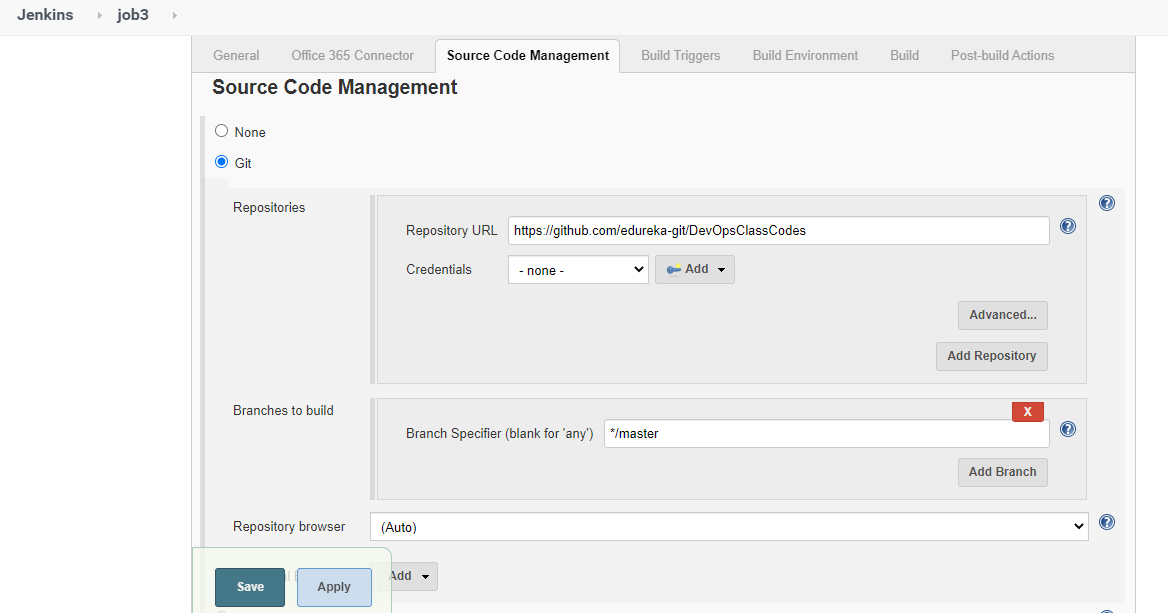
* For setting the Maven\_Home variable



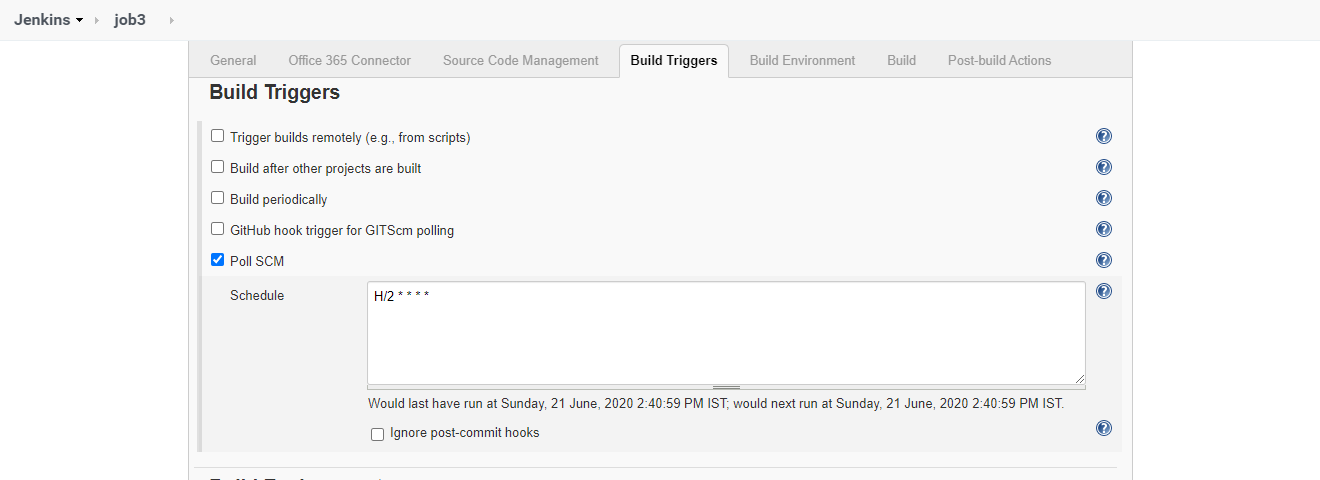
* Git\_Home Variable setting



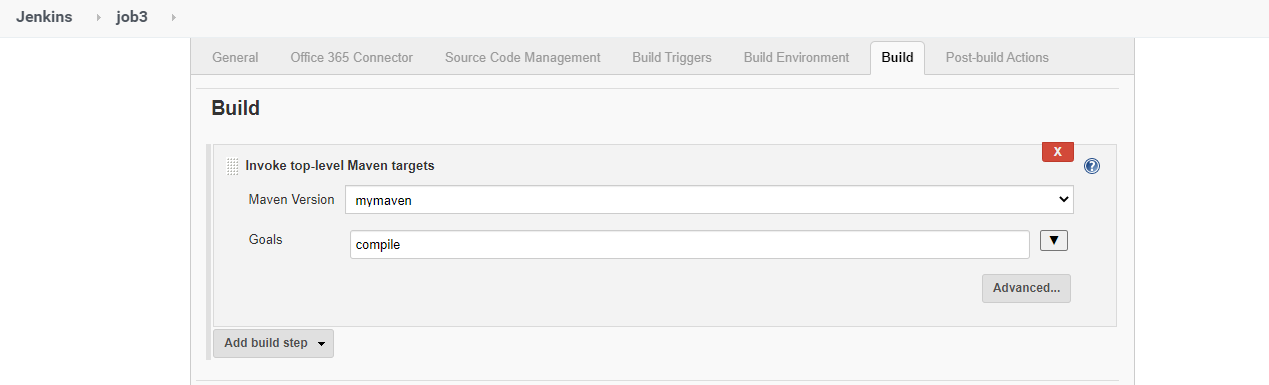
* Once above setup is done(git,jdk,maven).we are ready to compile our java project.
* **Configure the compile job**
* Step1 (SCM Configuration)



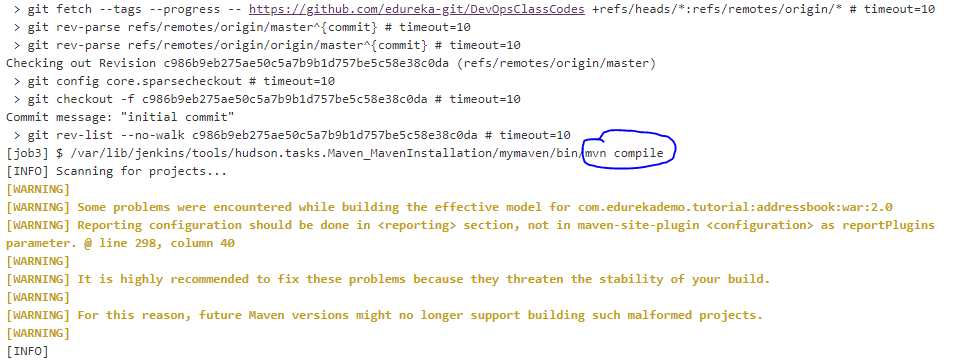
* This configuration will pull the code from the GitHub repository and will place in workspace directory on Jenkins server.
* Step2 (Build Triggers)



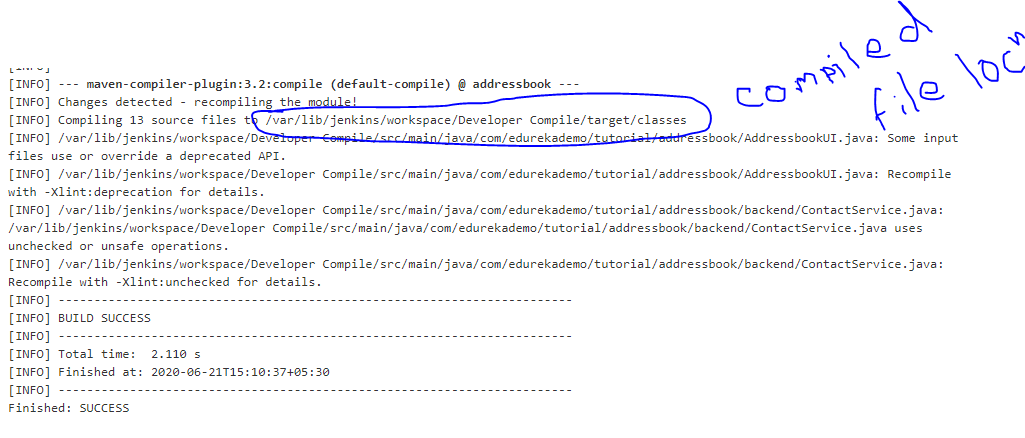
* Here I have scheduled cron to Poll(watch) SCM repository on the interval of 2 minutes. So, when any commits will happen in the GitHub repository by any developer it will pull the changes and will build the code.
* Step 3 (Build)

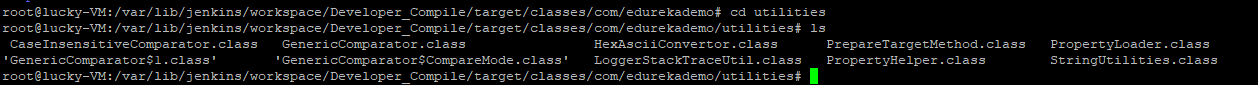


* Here I have configured the build step using maven like (mvn compile). Goals we have mentioned the target as **compile**
* We can see the maven command captured from job log.



* Compile .class file location on jenkins server has encircled in below snapshot





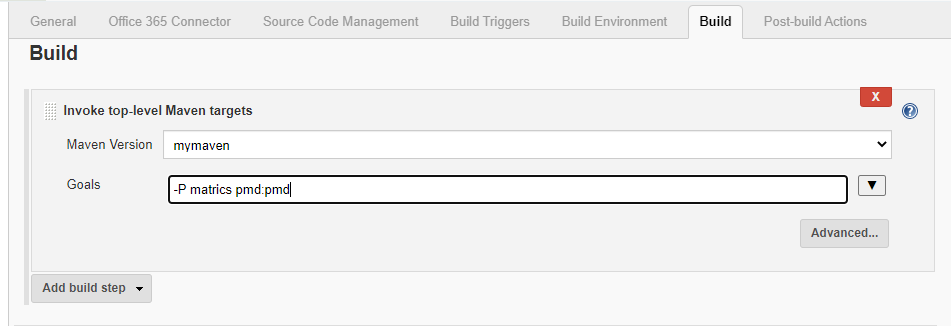
* **Configure the Code\_Review Job**

**Q.** What is PMD?

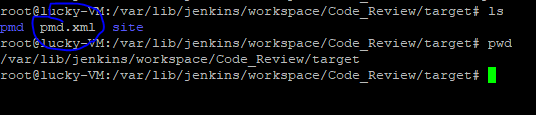
* **PMD** is a static source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It’s mainly concerned with **Java and Apex**, but **supports six other languages**.
* PMD features many **built-in checks** (in PMD lingo, *rules*), which are documented for each language in our [Rule references](https://pmd.github.io/latest/index.html#shuffle-panel-rule-references).
* We also support an extensive API to [**write your own rules**](https://pmd.github.io/latest/index.html#shuffle-panel-writing-rules), which you can do either in Java or as a self-contained XPath query.
* PMD is most useful when **integrated into your build process**. It can then be used as a quality gate, to enforce a coding

standard for your codebase. Among other things, PMD can be run:

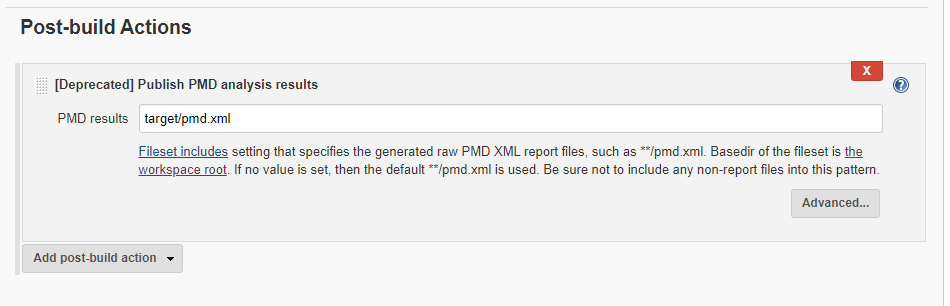
* As a [Maven goal](https://pmd.github.io/latest/pmd_userdocs_tools_maven.html)
* As an [Ant task](https://pmd.github.io/latest/pmd_userdocs_tools_ant.html)
* As a [Gradle task](https://docs.gradle.org/current/userguide/pmd_plugin.html)
* From [command-line](https://pmd.github.io/latest/pmd_userdocs_installation.html#running-pmd-via-command-line)
* **CPD**, the **copy-paste detector**, is also distributed with PMD.
* Step 1 (Create job)
* Create a job named as Code\_Review.
* Step 2 (Configure SCM)
* Configure the git repository url as mentioned in previous job.
* Step 3 (Configure Build)
* Configure the goal as **-P matrics pmd:pmd** [PMD (Program Mistake Detector)]



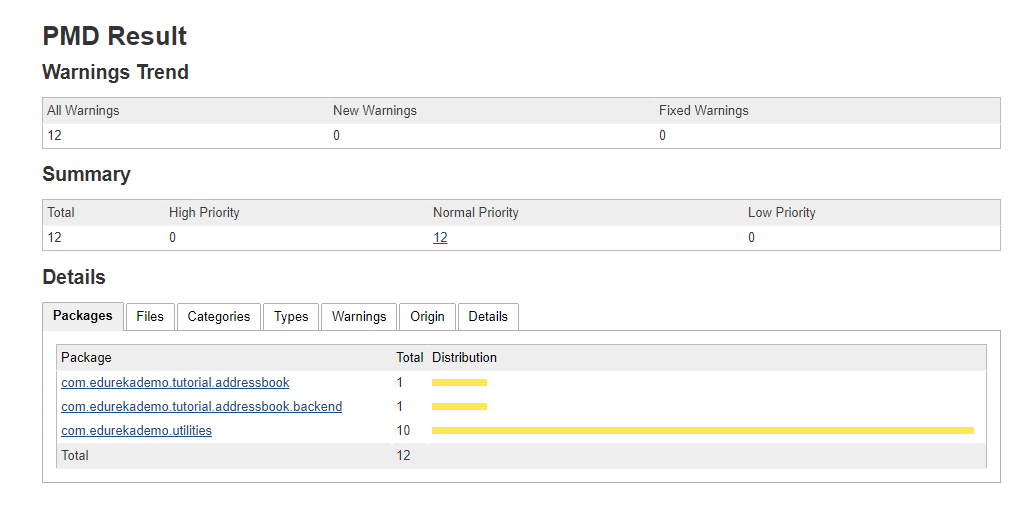
* Once job is started it will generate the pmd.xml file at /var/lib/jenkins/workspace/Code\_Review/target location.



* This pmd.xml is generated by the maven as a goal.
* Now to get the friendly UI for pmd data, we need to install a plugin on Jenkins named as PMD. Which will embed the data generated by pmd.xml in a UI look.
* Step 4 (Post Build Action Configuration)
* Once plugin is installed, we can configure the PMD as below.
* Select the **Publish PMD analysis results** from **add post build action** dropdown.
* Here we have to provide the relative path of pmd.xml file (target/pmd.xml).

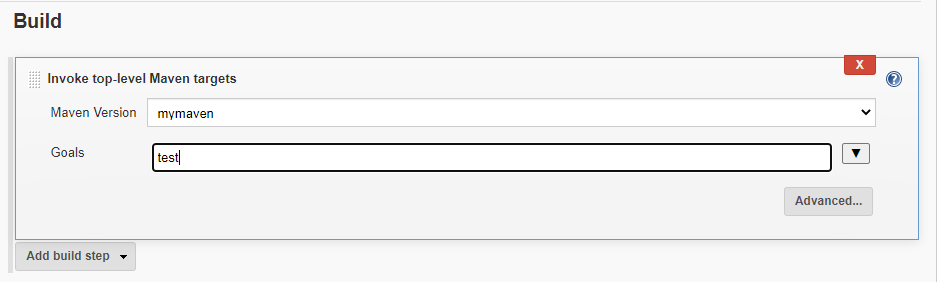


* Once post build action configuration is done, run the job and it will show the PMD result in user friendly UI.

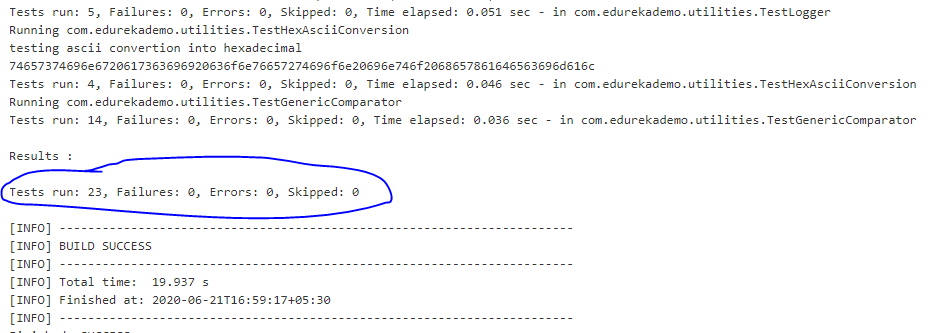


**Note: When ever we are using any plugin make sure to make an entry in the pom.xml file. Like instead of PMD we can use some other code review tool. So, for that specific plugin we have to add in pom.xml.**

* **Configure the Unit Test Job**
* This job will execute the junit test class written by developer.
* Step 1 (Create job)
* Create a job named as Test\_Job.
* Step 2 (Configure SCM)
* Configure the git repository url as mentioned in previous job.
* Step 3 (Configure Build)
* Configure the goal as **test (mvn test)**



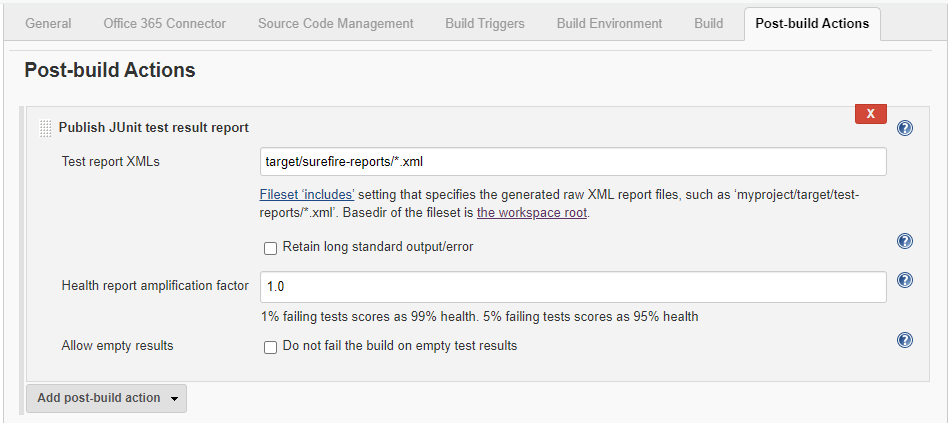
* Once above configuration is done we can start the job and can get the job output log something like this.



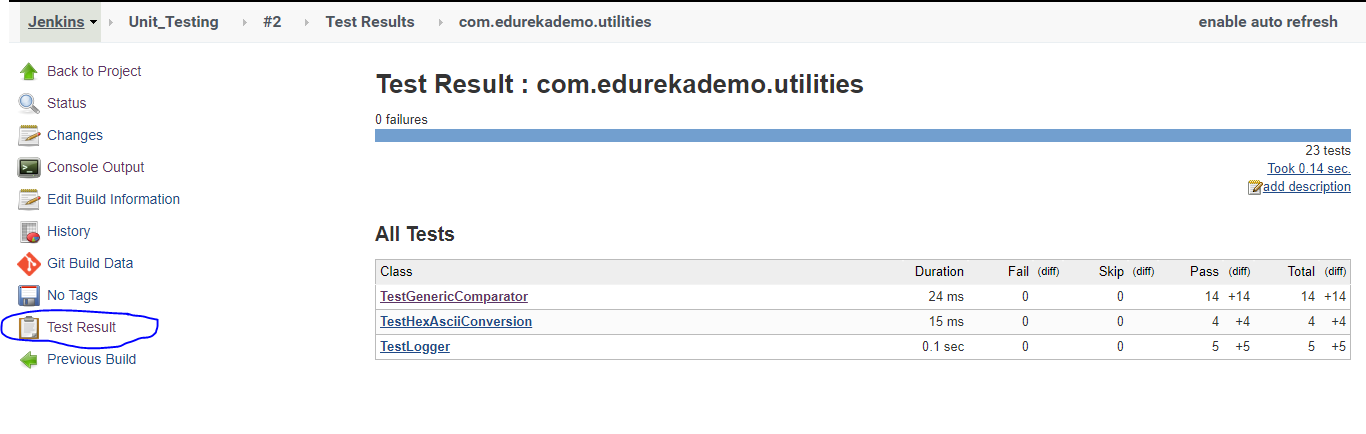
* The test report generated is placed under **/workspace/target/surefire-reports directory.**
* **The generated report is in xml format. So, we need some plugins in Jenkins to view this report in UI.**



* Step 4 (Post Build Action Configuration)
* Add the post build action as “**Publish Junit test result report**” Provide the test report xml file location and build the job.



* We can get the test report as below.

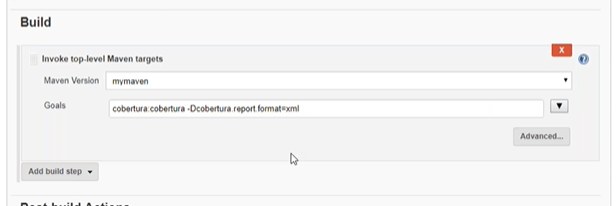


**NOTE: We can run multiple jobs in one job by providing the multiple goals under Build tab.**

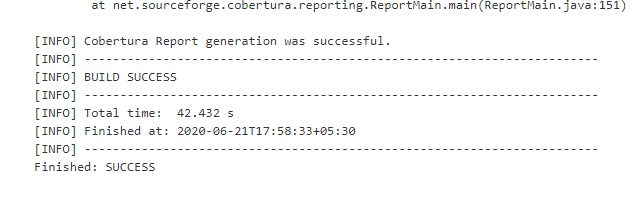
**Also, in real time we run the job in different server.**

* **Configure the MetricCheck Job**
* This job will provide the coverage analysis by using **Cobertura plugin**.
* Cobertura is a free Java tool that calculates the percentage of code accessed by tests. It can be used to identify which parts of your Java program are lacking test coverage. It is based on jcoverage.
* Step 1 (Create job)
* Create a job named as Metric\_Check.
* Step 2 (Configure SCM)
* Configure the git repository url as mentioned in previous job.
* Step 3 (Configure Build)
* Configure the goal as **cobertura:cobertura -Dcobertura.report.format=xml ( This goal will generate the report in xml format)**

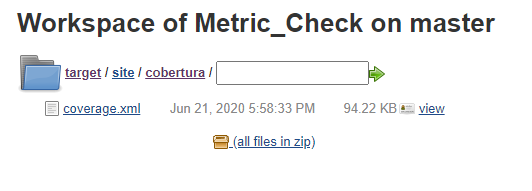
**-D means display**



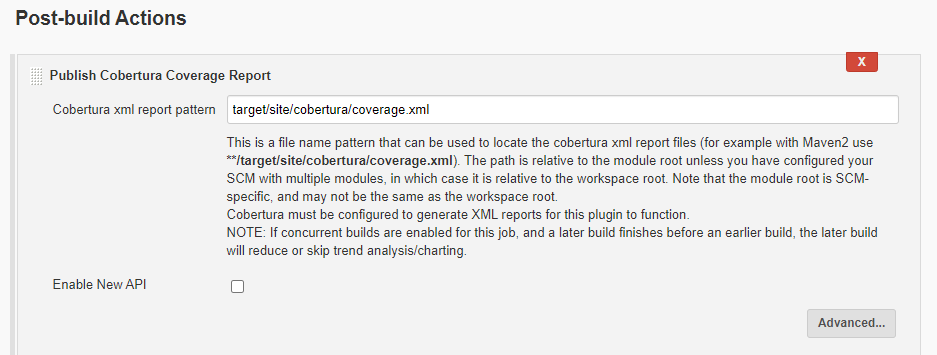
* Once above configuration is done, we can start the job. We can get the log something like as below



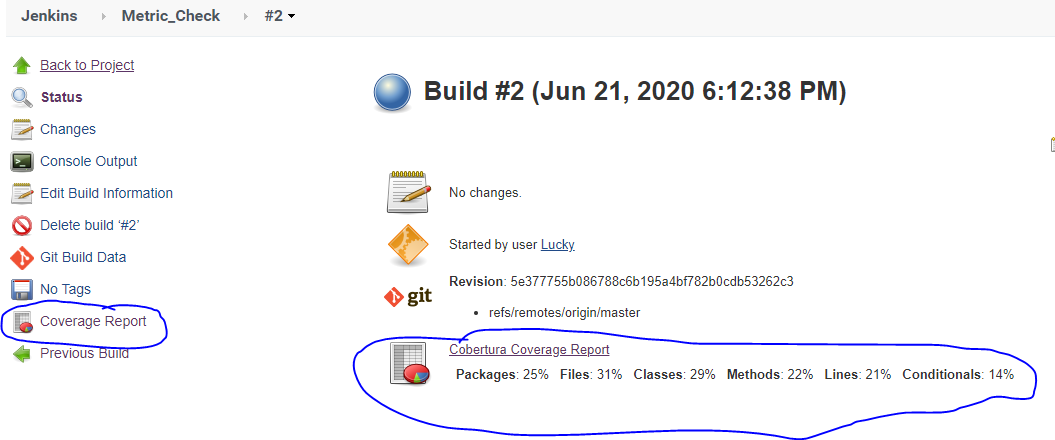
* The generated coverage.xml report is located at /workspace/target/site/cobertura directory.

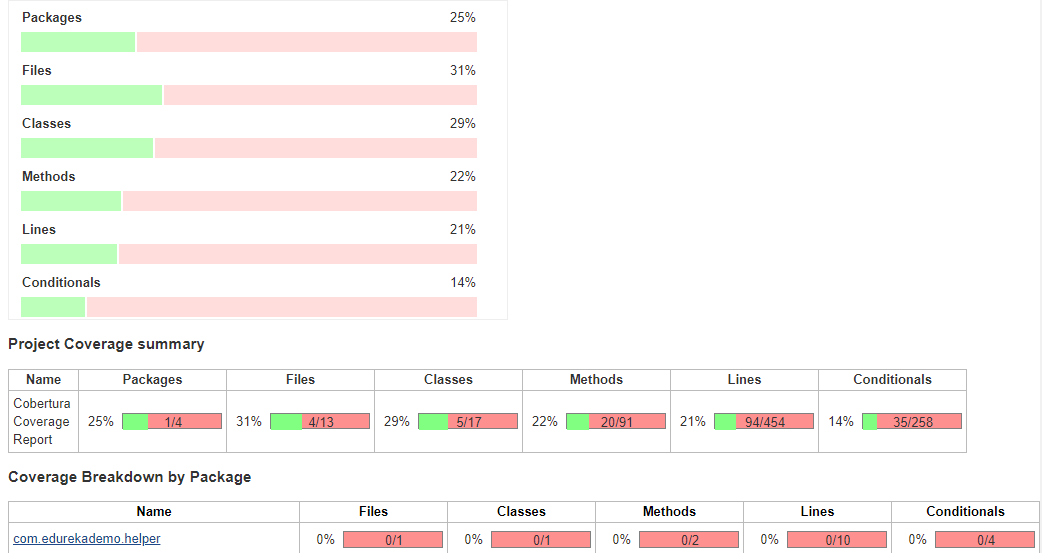


* Now to see the UI view, we can install plugin named as **cobertura.**
* Once plugin is installed, we can configure **Post build action**.
* Step 4 (Post Build Action Configuration)
* Add the post build action as “**Publish Cobertura Coverage report**” Provide the **coverage.xml** file location and build the job.

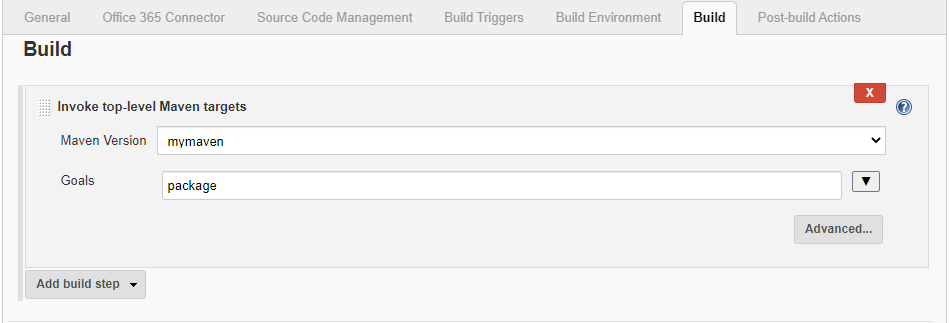


* Once job build is done, we can see the report like below.

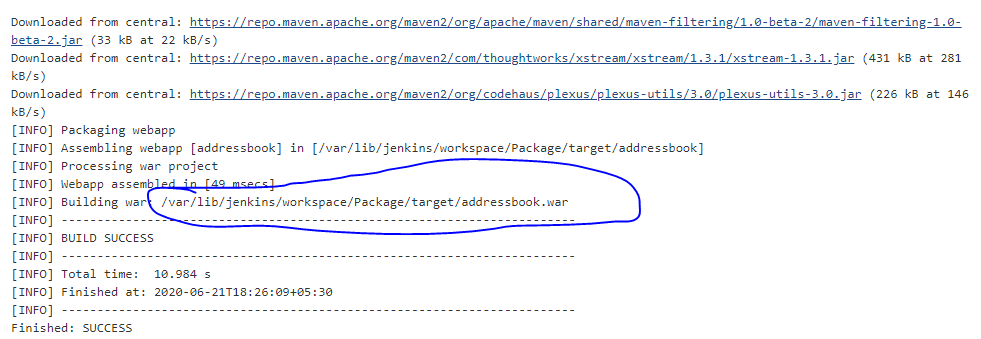




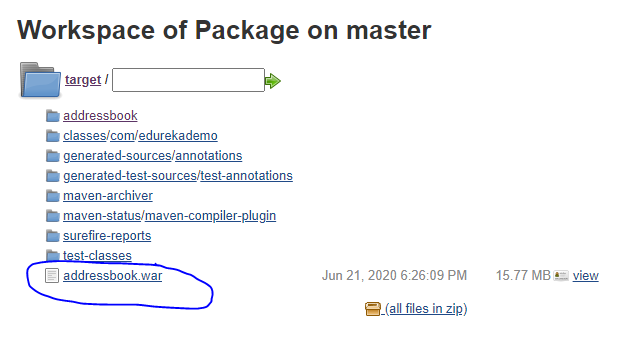
* **Configure the Package Job**
* This job will package the project into a war file ready for deployment.
* Step 1 (Create job)
* Create a job named as Package.
* Step 2 (Configure SCM)
* Configure the git repository url as mentioned in previous job.
* Step 3 (Configure Build)
* Configure the goal as **package (This goal will generate the war packaging file)**



* Once configuration is done, we can build the job and it will show log something like below.



* **War** file will be placed under **/workspace/Package/target/** directory



* **Integrating all jobs**
* We need to integrate all jobs together to create the **CI/CD** pipeline.
* For that we need to configure **Build Trigger** and **Post Build Action.**
* **Preceding job will trigger succeeding job.**
* **J1🡪J2🡪J3🡪j4🡪j5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Job No.** | **Job1** | **Job2** | **Job3** | **Job4** | **Job5** |
| **Build Trigger** | **SCM** | **Job1** | **Job2** | **Job3** | **Job4** |
| **Post Build Action** | **Job2** | **Job3** | **Job4** | **Job5** | **XXXXXXXX** |

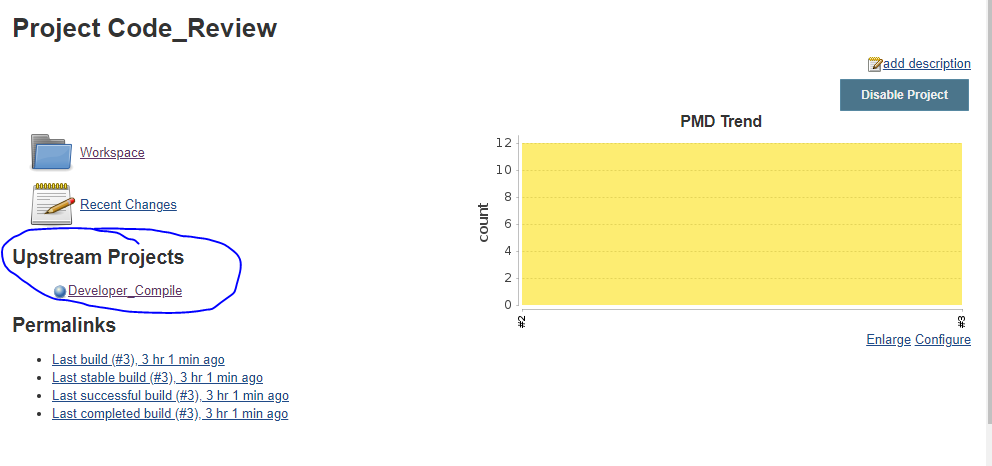
**NOTE:** We can read like **job1** will be triggerd **by SCM** and **Job1** **will trigger job2.**

* **Configuring first(Developer\_compile) job to CI/CD pipeline**
* Firstly, go to the first project (Developer\_compile).
* For that project configure the **Build Trigger** and **Post Build Action**.
* For first job we can configure Build Trigger as the timer or poll SCM or manual trigger.
* For **Post-build Actions**, we can select **Build other projects** from the **Add post-build action** dropdown.
* Then provide the job name we want to trigger **post completion** of the **current job**.

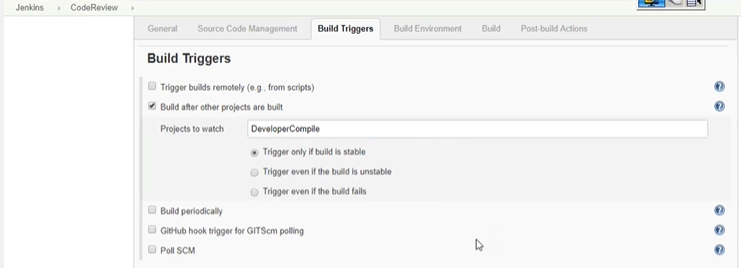


* Once configuration is done. We can see **Downstream Projects** as **Code\_Review** on the **Developer\_Compile** job dashboard.
* Also, if we can see Upstream **Projects** as **Developer\_Compile** on the **Code\_Review** job dashboard.

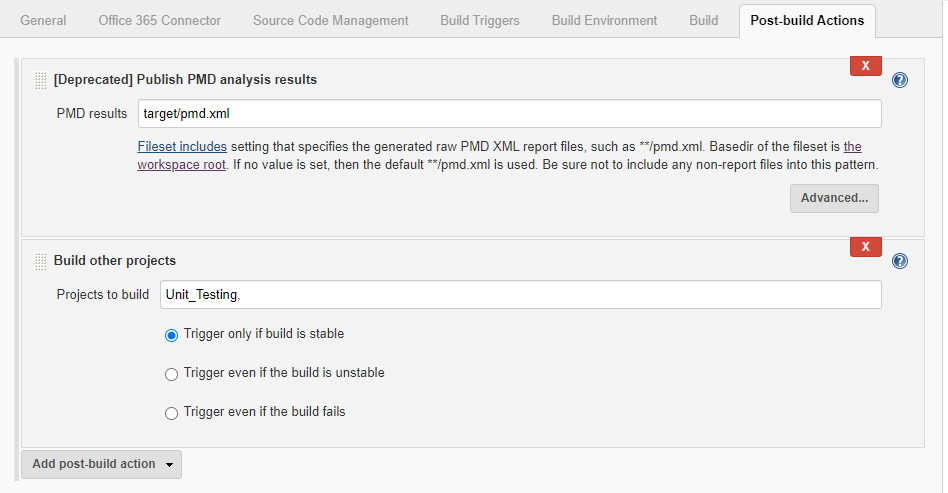




* **Configuring second(Code\_Review) job to CI/CD pipeline**
* Go to the second job (**Code\_Review**).
* For that job configure the **Build Trigger** and **Post Build Action**.
* For **Code\_Review** job we can configure **Build Trigger** as **Build after other projects are built** and provide the dependent job name like **Developer\_Compile.**



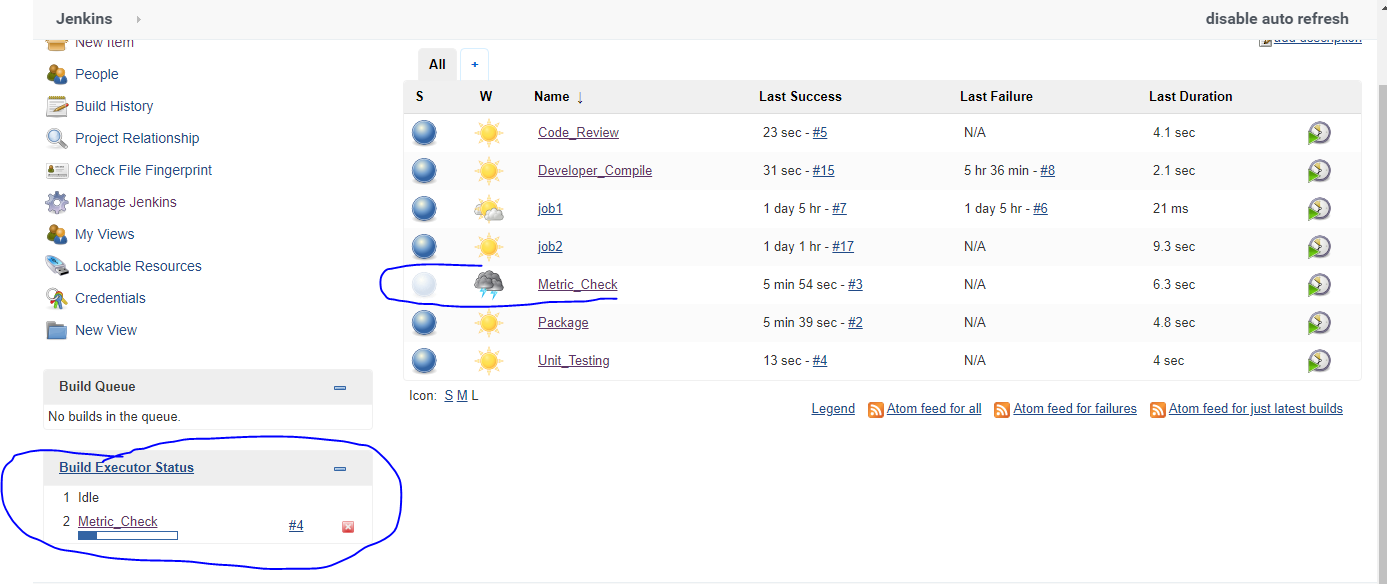
* For **Post-build Actions**, we can select **Build other projects** from the **Add post-build action** dropdown.

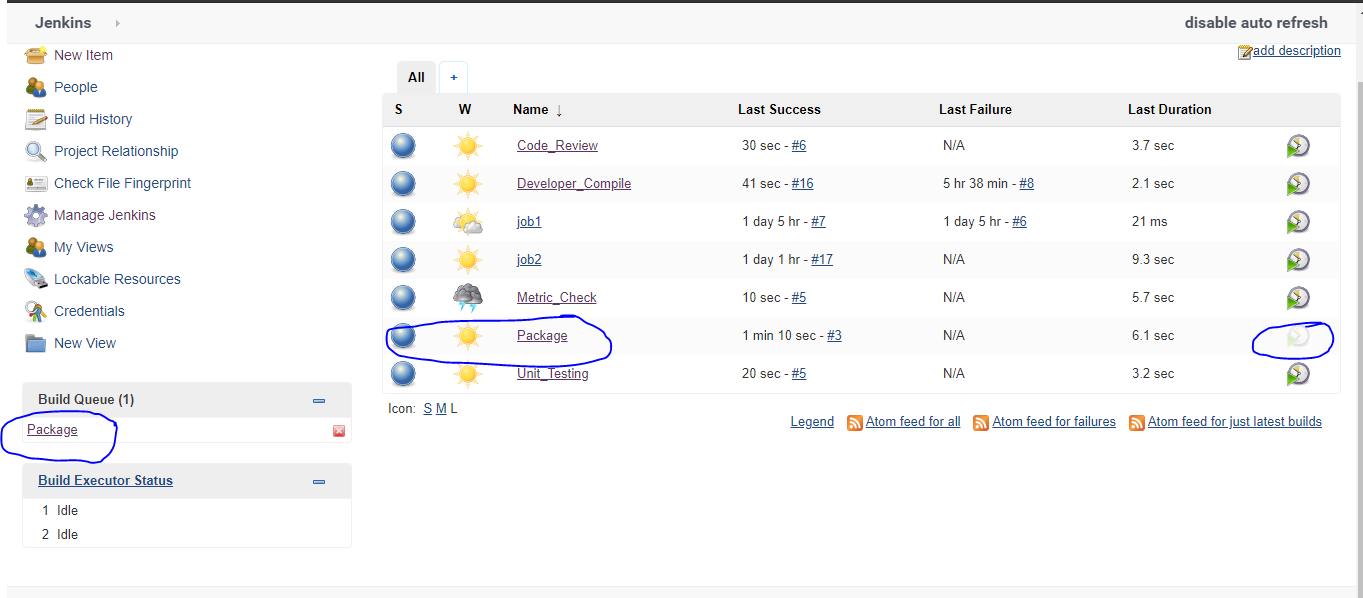


* Then provide the job name we want to trigger **post completion** of the **current job**.
* We need to connect all the job with each other like we did in above job configuration

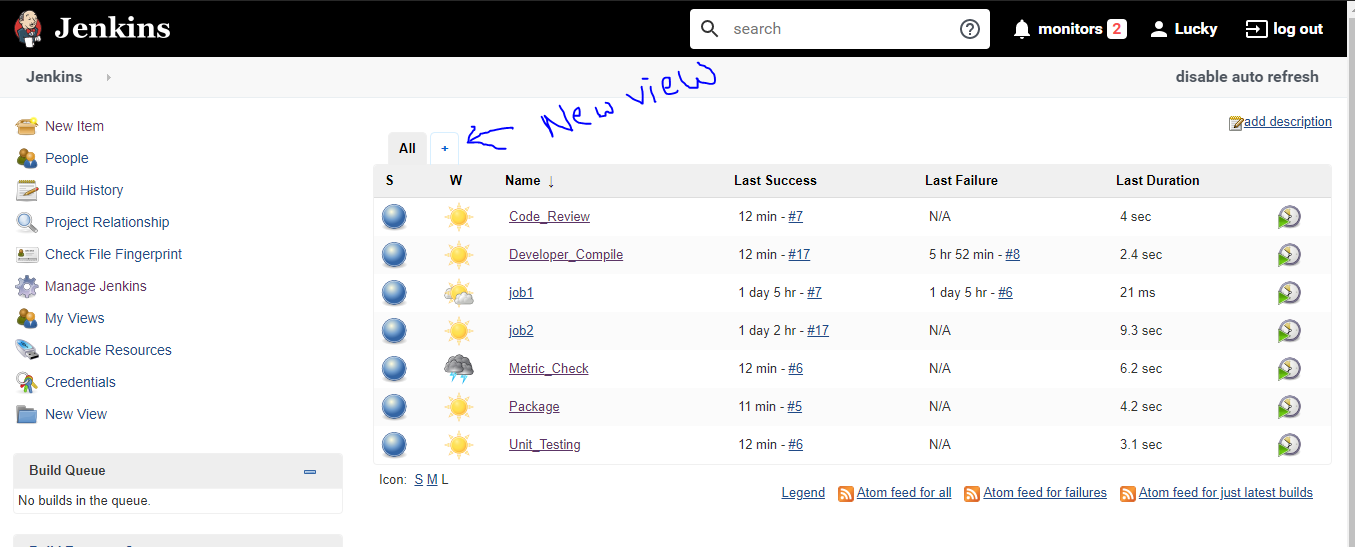
**Note: for the last job no need to configure Post-build Action.**

* **Trigger the CI/CD pipeline**
* **Go to the first job and trigger it by clicking build now from the Jenkins dashboard.**
* **Click on the Enable auto refresh option to see the real time progress.**
* **We can see the current job executing in the CI/CD pipeline.**
* **Then we can see the blink icon on the dashboard.**

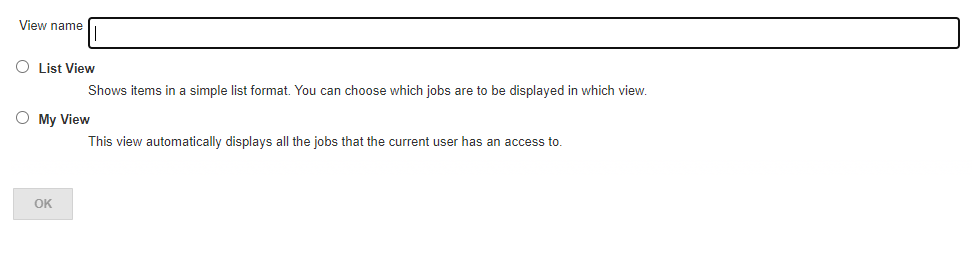




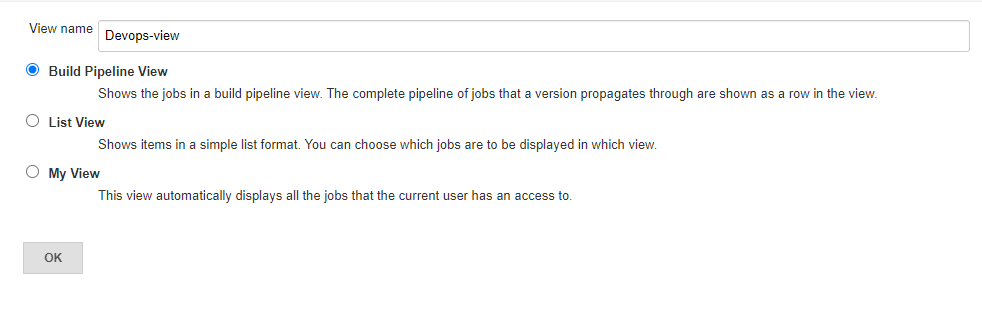
* **To view the better CI/CD pipeline on dashboard**
* Go to the dashboard and click on the(**+**) new view.



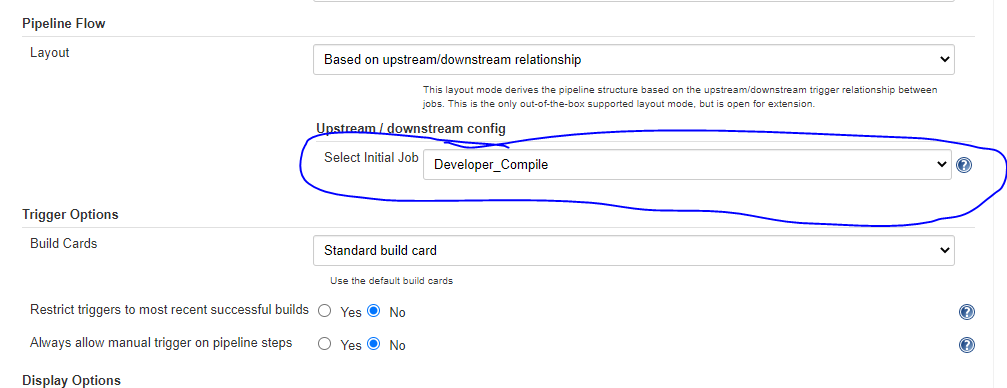
* We can see the List View and My View option



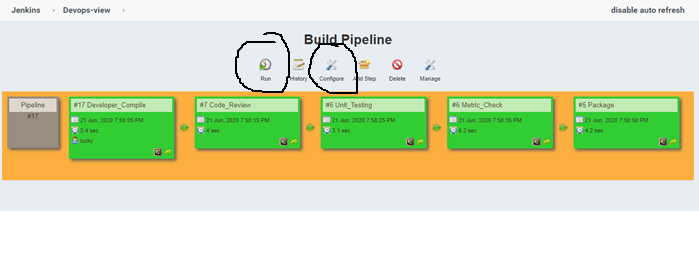
* Now go to the Mange Plugin and install plugin named as **Build Pipeline.**
* Once plugin Is installed, we can see the **Build Pipeline View.** Select the **Build Pipeline View** and give the view name.



* Then click on the OK a new screen will be open where we have to select the initial job name (Developer\_Compile) and click OK.



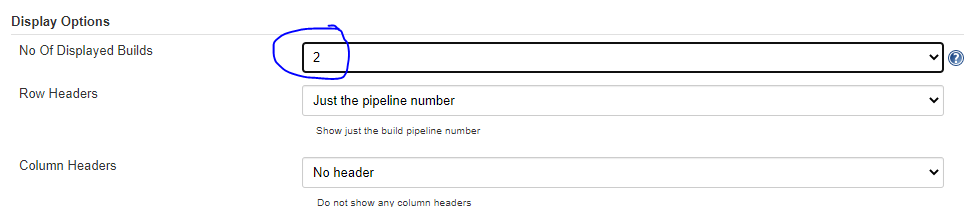
* Once setup is done, we can see the dashboard view like showed in below image.

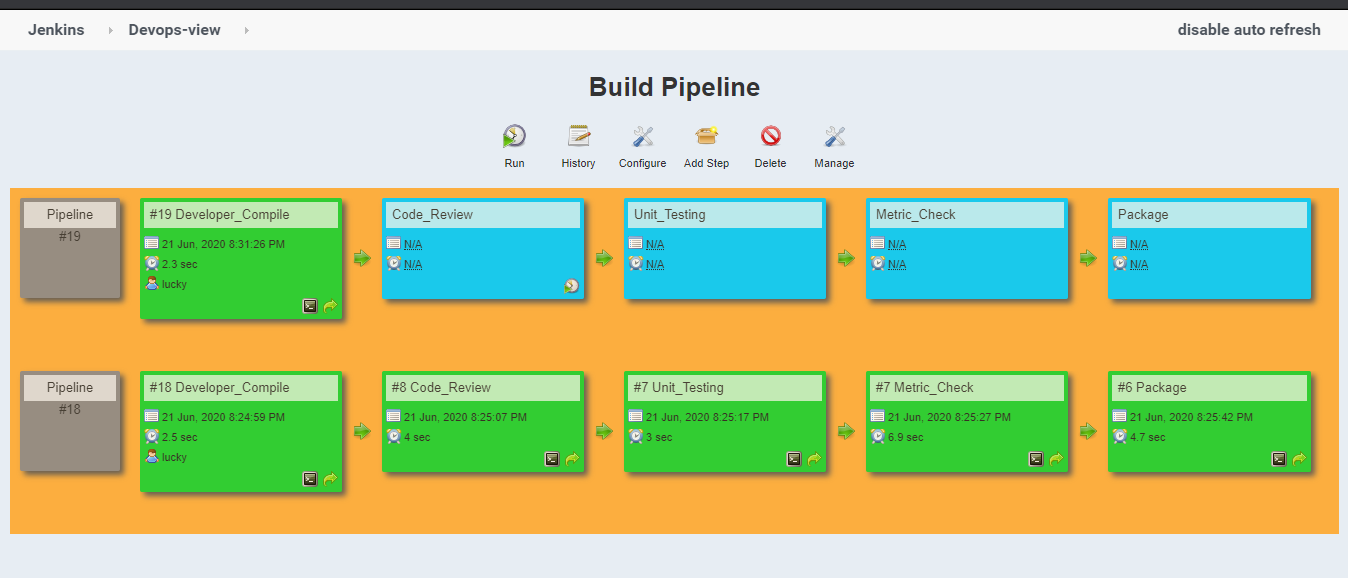


Just click on **Run** button on this dashboard view and we can see the pipeline.

* Yellow: current executing
* Green: executed
* Blue: to be executed
* Red: execution failed

**Note: We can configure two build to show on view. Click configure on the view dashboard.**





* In above dashboard view we can see two build is showing at a time.

Bottom of Form