**Jenkins**

Jenkins is a powerful application that allows **continuous integration** and **continuous delivery** of projects written in java.

Jenkins is nothing but a middle man between your code repo and your build server. It checks for changes on your server every few minutes. If it found them, it gathers them and sends them to your build server. That's what Jenkins is.

You can integrate Jenkins with a number of testing and deployment technologies.

**Simple work flow of Jenkins:**



**Why Jenkins and advantages of Jenkins:**

1. It is open source and it is user-friendly, easy to install and does not require additional installations or components.
2. It is free of cost.
3. Easily Configurable. Jenkins can be easily modified and extended. It deploys code instantly, generates test reports. Jenkins can be configured according to the requirements for continuous integrations and continuous delivery.
4. Platform Independent. Jenkins is available for all platforms and different operating systems, whether OS X, Windows or Linux.
5. Rich Plugin ecosystem. The extensive pool of plugins makes Jenkins flexible and allows building, deploying and automating across various platforms.
6. Easy support. Because it is open source and widely used, there is no shortage of support from large online communities of agile teams.
7. Developers write the tests to detect the errors of their code as soon as possible. So the developers don’t waste time on large-scale error-ridden integrations.
8. Issues are detected and resolved almost right away which keeps the software in a state where it can be released at any time safely.
9. Most of the integration work is automated. Hence fewer integration issues. This saves both time and money over the lifespan of a project.

(Simply to say, open source tool->easy to install->has more than 1000 plug-in to make the work easier->easy to create new Jenkins plugin if one is not available->a tool which is written in Java, hence it can be portable to almost all major platforms)

**What is Continuous Integration?**

Continuous Integration is a development practice that requires developers to integrate code into a shared repository at regular intervals.

This concept was meant to remove the problem of finding later occurrence of issues in the build lifecycle.

Continuous integration requires the developers to have frequent builds. The common practice is that whenever a code commit occurs, a build should be triggered.



**Jenkins Installation:**

Download Jenkins from the official website

https://jenkins.io/

Click the link “Older but stable version” to download the Jenkins war file.

## Starting Jenkins

Open the command prompt. From the command prompt, browse to the directory where the jenkins.war file is present. Run the following command

**E:\>java –jar jenkins.war**

After the command is run, various tasks will run, one of which is the extraction of the war file which is done by an embedded webserver called winstone.

**E:\>java –jar jenkins.war**

Running from: C:\jenkins.war

Webroot: $user.home/ .jenkins

Sep 29, 2015 4:10:46 PM winstone.Logger logInternal

INFO: Beginning extraction from war file

Once the processing is complete without major errors, the following line will come in the output of the command prompt.

INFO: Jenkins is fully up and running

## Accessing Jenkins:

## Once Jenkins is up and running it can be accessed from the link,

−> **http://localhost:8080**

This link will bring up the Jenkins dashboard.

**User name**: provide user name (admin)

**Pwd:** will be available in C:\Users\mdodda\.jenkins\secrets\initialAdminPassword (09b10f8570144e729c8fe64d8beaeec2)



By default, Jenkins is accessed on port 8080. Using the below command, port can be changed if any conflicts occur.

**E:\>java –jar jenkins.war –httpPort=8088**

**Scenario-1:** -> ***Jenkins - Git Setup***

For setting up Git in Jenkins,

1. Go to Jenkins dashboard
2. Click manage Jenkins (on left side)
3. Click manage plugins
4. Click on Available tab -> show list of plugins -> which are available for downloading -> Search for Git Plugin in Filter tab
5. Check the Git Plugin Option and click on the button “Install without restart”
6. Installation will be done in Jenkins
7. Once done with installation restart Jenkins with the link [**http://localhost:8080/jenkins/restart**](http://localhost:8080/jenkins/restart)
8. To verify, click on New Item in the menu options, then enter a name for a job, like Demo. Select Freestyle project as item type and click ok button.
9. If you browse to the Source code management section, you will now see “Git” as an option.
10. If once “Git” option is seen Git setup is done successfully in Jenkins.

Check the following screenshots,

Click the ‘Manage Plugins’ option. 

Click the Available tab. This tab will give a list of plugins which are available for downloading. In the ‘Filter’ tab type ‘Git plugin’

The list will then be filtered. Check the Git Plugin option and click on the button ‘Install without restart’



The installation will then begin and the screen will be refreshed to show the status of the download as shown in the below screenshot.



Once all installations are complete, restart Jenkins by issuing the following command in the browser -> **http://localhost:8080/jenkins/restart**

After Jenkins is restarted, Git will be available as an option whilst configuring jobs. To verify, click on New Item in the menu options for Jenkins. Then enter a name for a job, in the following case, the name entered is ‘Demo’. Select ‘Freestyle project’ as the item type. Click the Ok button.



If you browse to the Source code Management section, you will now see ‘Git’ as an option.

**Scenario-2:** -> ***Integrating Jenkins & GitHub***

( i.e., Compiling and running a java application in jenkins on Windows by cloning code from git repository)

**Jenkins** is a continuous integration server and this means it needs to check out source code from a source code repository and build code.

Jenkins has excellent support for various source code management systems like CVS, Subversion, etc.

**Git** is fast becoming one of the most popular source code management systems. Jenkins works with Git through the Git plugin. In above scenario, already we have seen how to set up a git plugin.

**Accessing Git Repository:**

Created an account in github -> https://github.com/

Created my repository by clicking "+" which is present on top left -> My Repository -> Repo\_name (myRepo)

**Uploading fles:**

To upload files into repo-> go to particular repository ( myRepo ) -> Upload files -> choose your files -> add comment/or description -> commit (to save the changes done)

To check the files present in the repo have to click on the particular repo name.

**Modifying files:**

We can modify files which are already uploaded in the repository -> select a file to be changed -> select edit on top left -> make necessary changes -> commit

**Deleting files:**

To delete files from the repository -> select file to be deleted -> select delete icon on top left -> commit

I have pushed java code into the git repository and committed the changes.

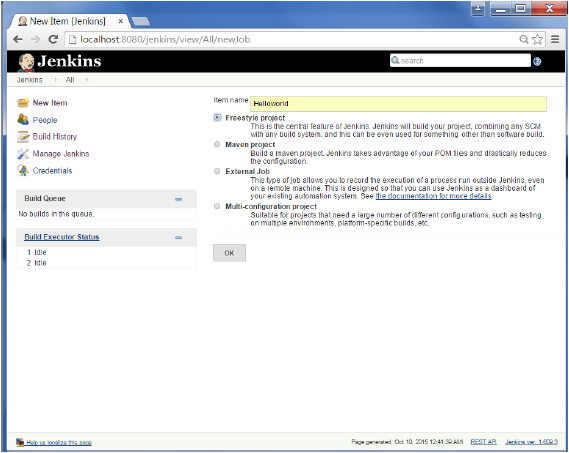
Let’s start the scenario, first create a job in Jenkins which picks up a simple HelloWorld application, builds and runs the java program.

**Step-1:** Create a new job in Jenkins dashboard by clicking on NewItem in the leftside of dashboard.

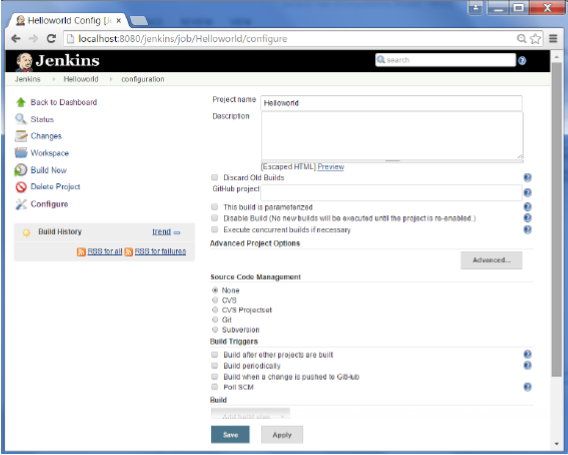
Build jobs are at the heart of the Jenkins build process. Simply put, you can think of a Jenkins build job as a particular task or step in your build process. This may involve simply compiling your source code and running your unit tests. Or you might want a build job to do other related tasks, such as running your integration tests, measuring code coverage or code quality metrics, generating technical documentation, or even deploying your application to a web server. A real project usually requires many separate but related build jobs.



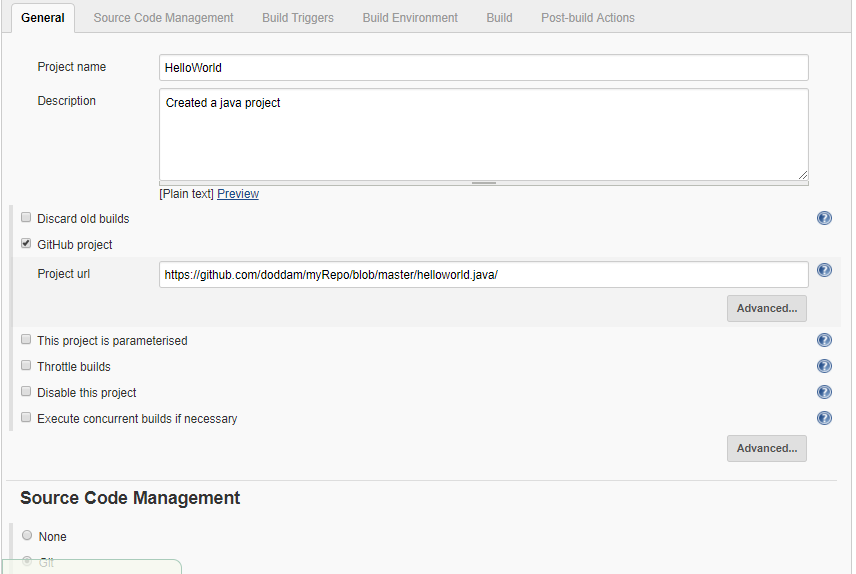
**Step-2:** Enter the Item name, in this case I have named it as “Helloworld”. Choose the ‘Freestyle project option’



**Step-3:**  The following screen will come up in which you can specify the details of the job.



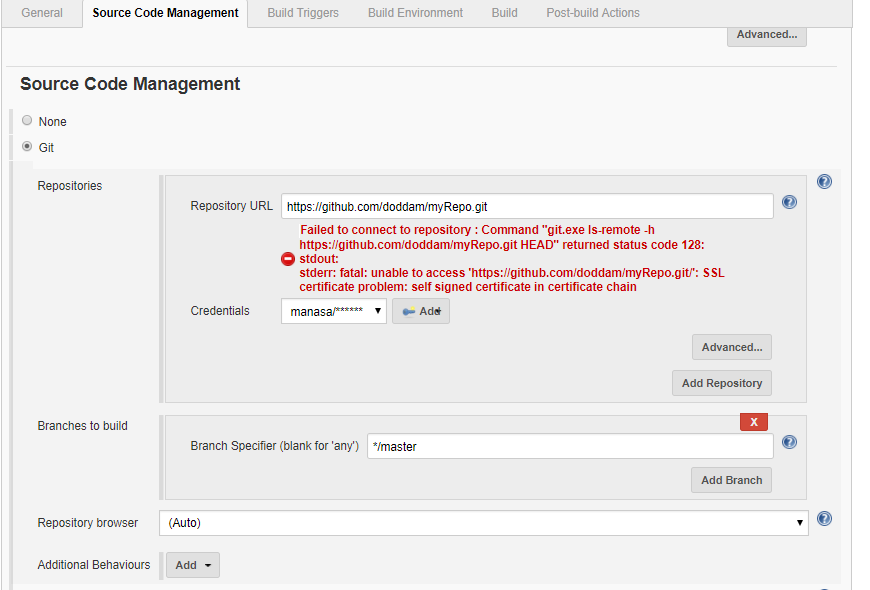
**Step-4:**

As my repository is hosted on Github, I entered the url of that repository here.****

**Step-5:**

We need to specify the location of files which need to be built. In this example, we will assume, a git repository has been setup which contains a ‘HelloWorld.java’ file. Hence scroll down and click on the Git option and enter the URL of the local git repository.

In addition to this, you would need to click on the Add button for the credentials to add a user name and password to the github repository so that the code can be picked up from the remote repository.



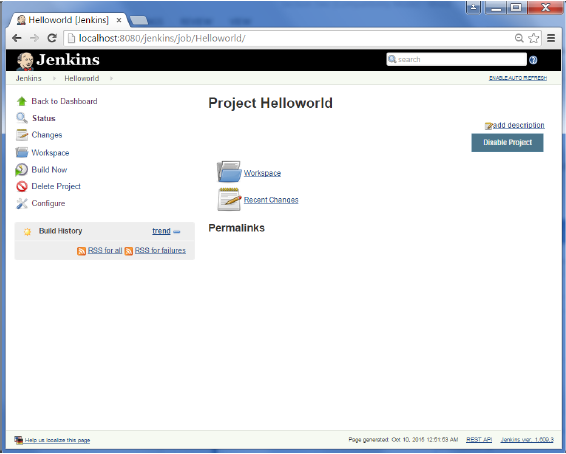
**Step-6:** Now go to the Build section and click on Add build step → Execute Windows batch command

**Step-7:** In the command window, enter the following commands and then click on the Save button.

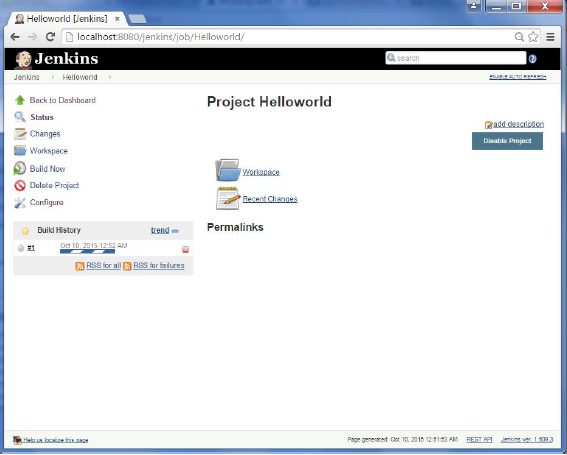
javac HelloWorld.java

java HelloWorld

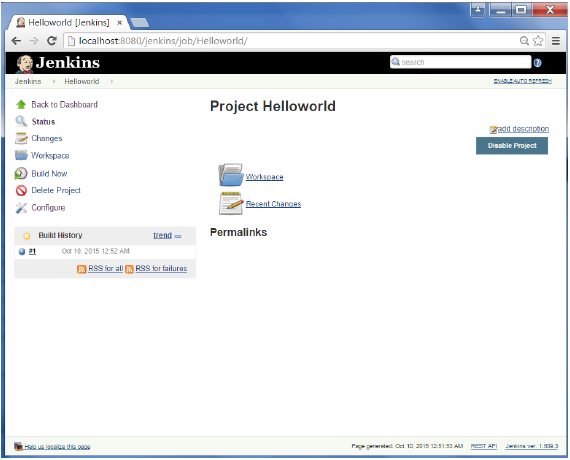
**Step-8:** Once saved, you can click on the Build Now option to see if you have successfully defined the job.



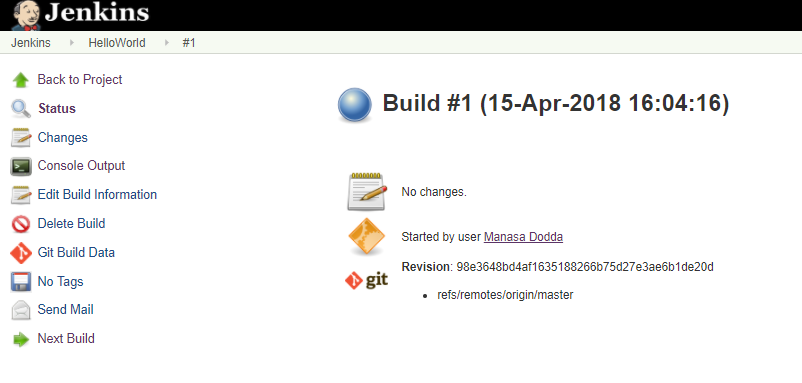
**Step-9:** Once the build is scheduled, it will run. The following Build history section shows that a build is in progress.



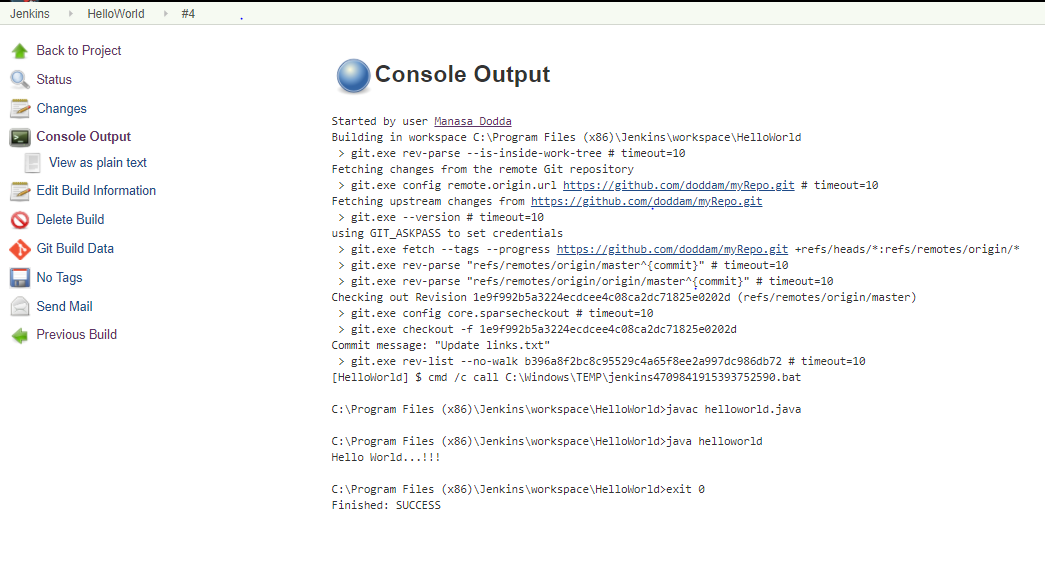
**Step-10:** Once the build is completed, a status of the build will show if the build was successful or not. In our case, the following build has been executed successfully. Click on the #1 in the Build history to bring up the details of the build.



**Step-11:** Click on the Console Output link to see the details of the build.



**Step-12:** You can see output by clicking Console output on the left side of dashboard.



**Scenario-3:** -> ***Jenkins - Tomcat Setup***

Why?

Jenkins – standalone server (owns a servlet container - Jetty/winstone)

Start all web applications on a single server tomcat

Deploy Jenkins in tomcat servlet container

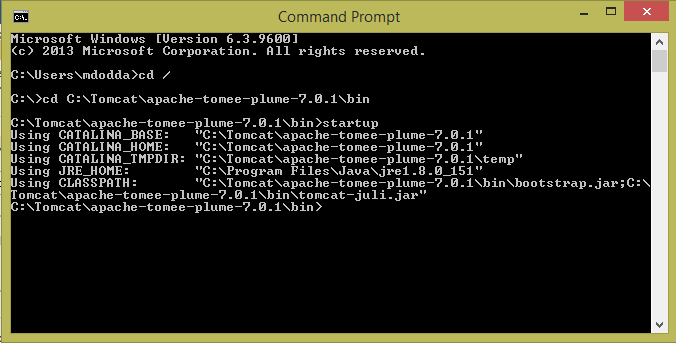
Need java 7 or above in system, tomcat 5 or above

**Step-1:** Download and Install tomcat

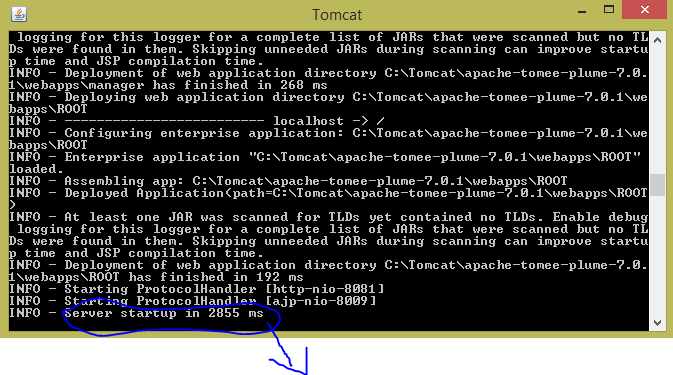
1. Go to [http://tomcat.apache.org](http://tomcat.apache.org/) ⇒ Under "Tomcat 9.0.{xx} Released" (where {*xx*} is the latest upgrade number) ⇒ Click "Download" ⇒ Under "9.0.{xx}" ⇒ Binary Distributions ⇒ Core ⇒ "**ZIP**" package (e.g., "apache-tomcat-9.0.{*xx*}.**zip**", about 9.8 MB).
2. UNZIP the downloaded file into your project directory "C:\Tomcat". Tomcat will be unzipped into directory "C:\Tomcat\apache-tomcat-9.0.7".
3. For ease of use, I shall shorten and rename this directory to "C:\Tomcat\tomcat" or leave it.
4. I have already tomcat in my system, so I will use that “C:\Tomcat\apache-tomee-plume-7.0.1”

**Step-2:** Start tomcat

Open windows command prompt -> navigate to your bin directory where tomcat is present C:\Tomcat\apache-tomee-plume-7.0.1\bin -> type the following command



When you type the above command, a separate window will open and a series of messages will appear, followed by the message indicating the server is started



Indicates tomcat server has been started and the exact number of milliseconds will vary based on the number of web applications that are deployed, among other factors

**Step-3:** How to know that tomcat is running….!?

The default port for Tomcat is 8080. After starting Tomcat on your local machine, you can validate if Tomcat is running the URL:

Type the URL http://localhost:8080 from a Web browser

**Note:**

Choosing a Different Port (Optional) by default, Tomcat runs on part 8080. You can change it to a different port. To do so, open C:\Tomcat\apache-tomee-plume-7.0.1\conf\server.xml using a text editor such as NotePad. Search for 8080 and change it to a desired port number such as 8081 in the following context.

<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"

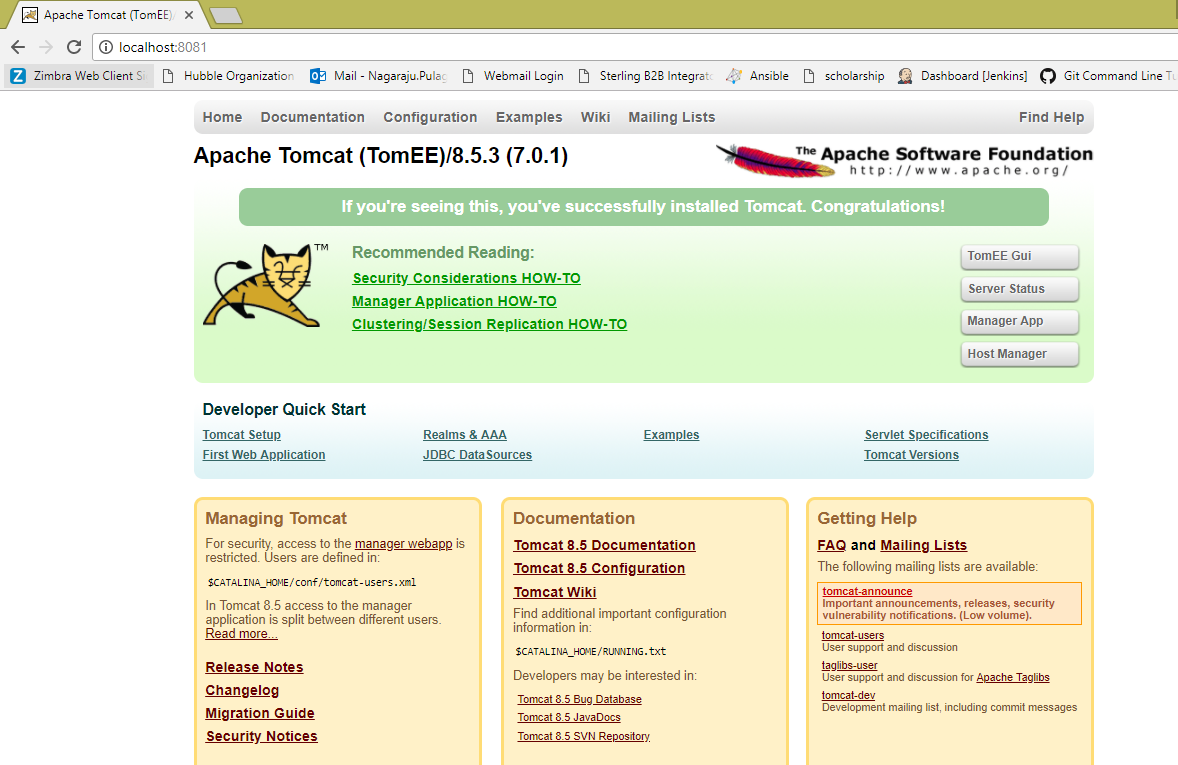
port="8080" minProcessors="5" maxProcessors="75"

enableLookups="true" redirectPort="8443"

acceptCount="100" debug="0" connectionTimeout="20000"

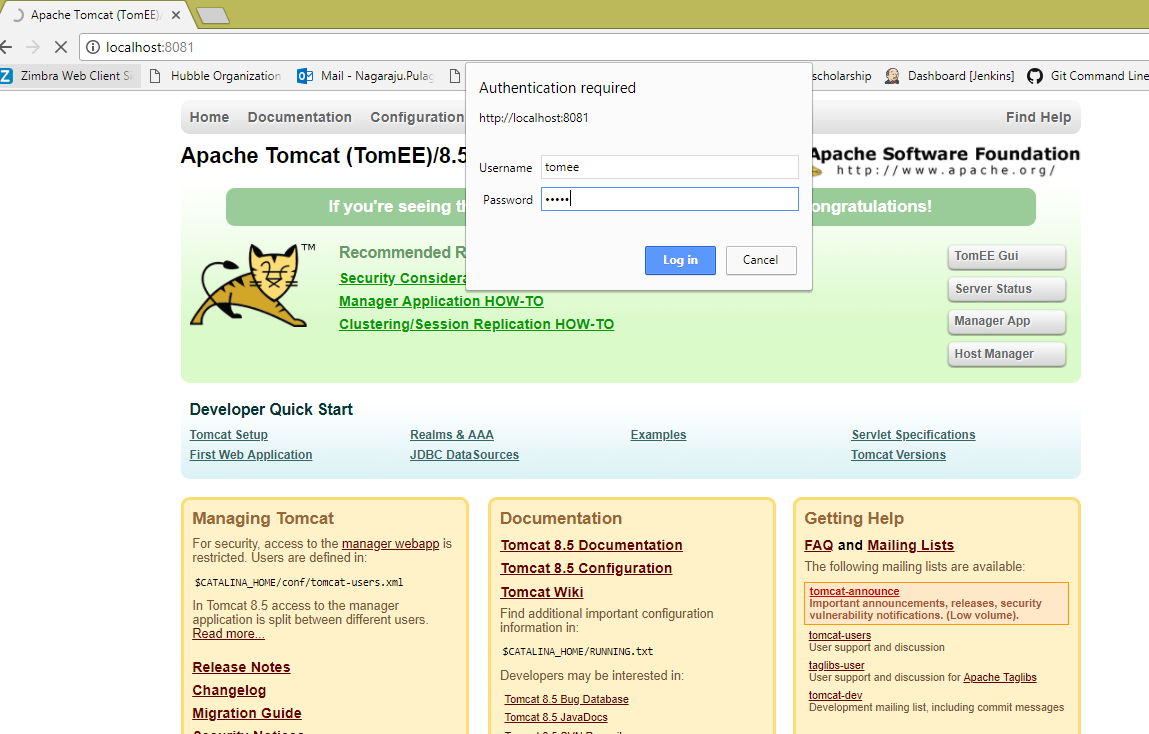
useURIValidationHack="false" disableUploadTimeout="true" />

Now, I run tomcat by the url URL http://localhost:8081 from a Web browser as shown below

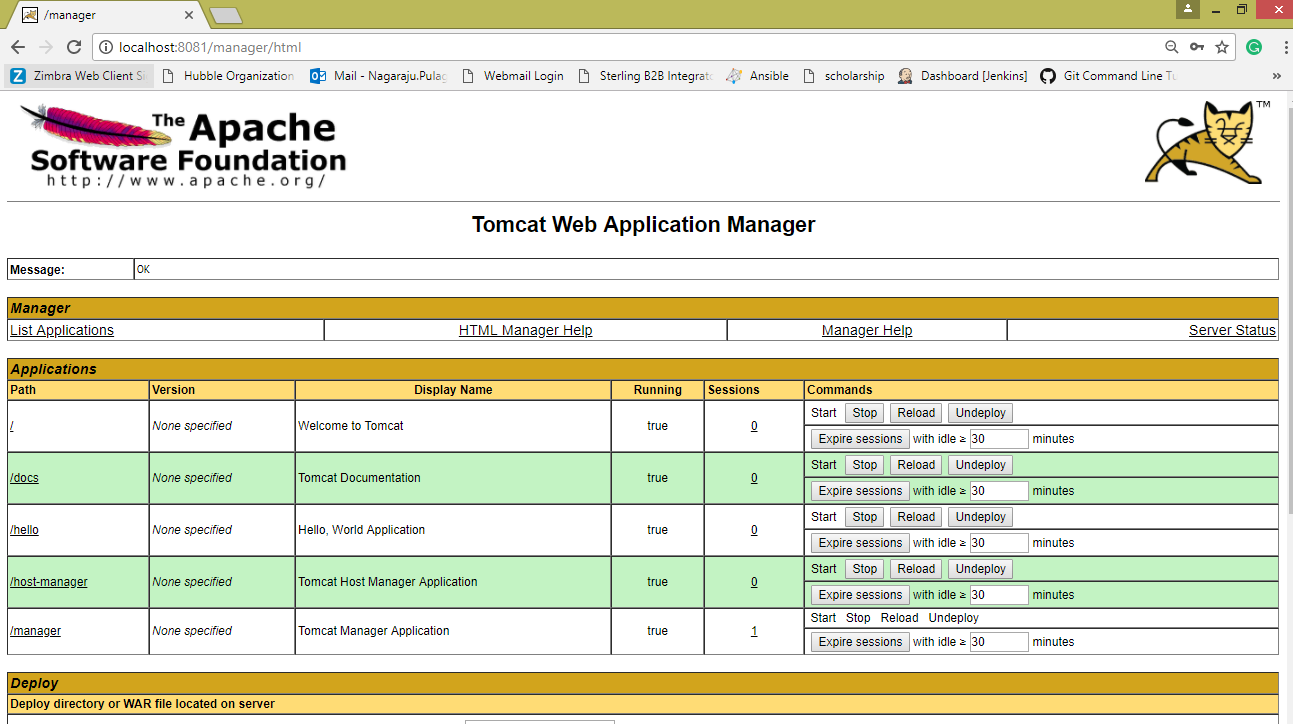


**Step-4:**

Now in the browser open manager app in gui, give user name and password > these will be in your conf/tomcat-users.xml > click login

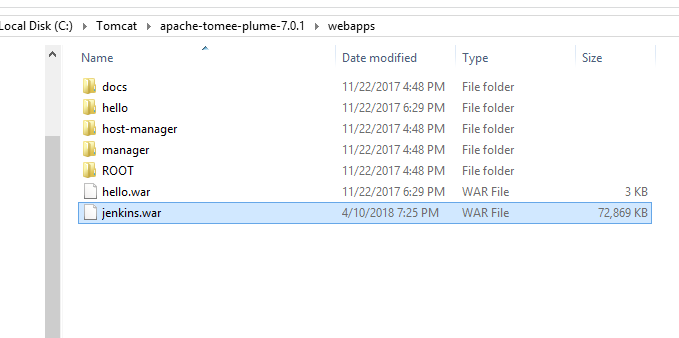


Check in the below screenshot, tomcat web application manager is opened which contains all the applications.

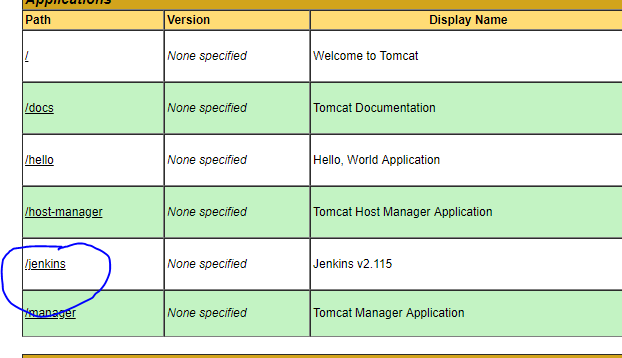


Stop the tomcat server from the command prompt by typing shutdown

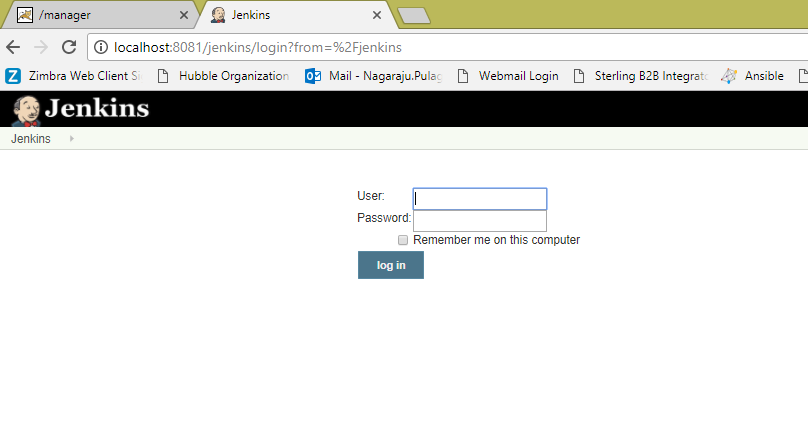
Now navigate to the path C:\Tomcat\apache-tomee-plume-7.0.1\webapps -> copy the jenkins.war file



Start tomcat server -> open browser -> run tomcat -> you can see Jenkins application



You can access Jenkins application by giving the url > http://localhost:8081/jenkins



**Scenario - 4:** -> ***Jenkins accessing an application from local system (on windows)***

Simply say as a basic scenario, Jenkins picks up a java application from local system which is placed in the Jenkins workplace.

[i.e., compile and run a simple java file in jenkins on Windows]

Hello.java

public class Hello {

public static void main(String args[]){n

System.out.println("I'm dancing");

}

}

**Steps:**

1. Create a new job/project in jenkins dashboard -> say “JavaProject”
2. Go to configuration page of your job/project
3. Select "Execute Windows batch command" from "Add build step"
4. Type the following commands:
   1. **javac Hello.java**
   2. **java Hello**
5. Save configuration.
6. Now place the file **Hello.java** in the path -> /Jenkins/workspace/ jobname/

Eg: /Jenkins/workspace/ JavaProject/

1. Build the project/job by clicking "Build Now" link and see the Console Output

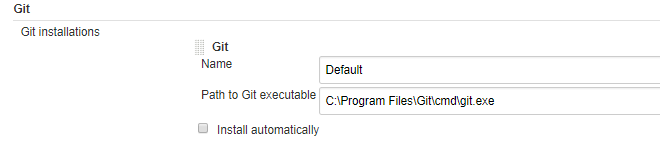
**Troublshooting:** *“*[***ERROR: Error cloning remote repo 'origin'***](https://stackoverflow.com/questions/37155321/error-error-cloning-remote-repo-origin) *”*

After building the project (integrating Jenkins and Git), I received the above error on the console output section.

Apparently the Jenkins Git Plugin executes \*\* before \*\* the environment is inherited.

1. Go to Manage Jenkins > Global Tool Configurations
2. Under Git configuration > Git installations
3. Enter complete path to git executable including git.exe > Save > Again build the project

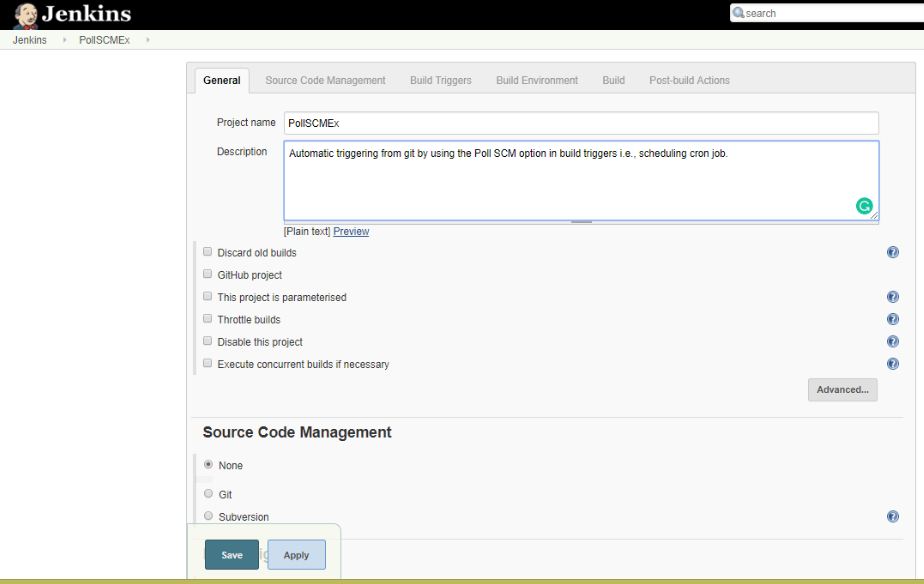
Check the below screenshot:



1. Sometimes this error occurs when disk space is less. So, do check the space of your machine where Jenkins is hosted.

**Scenario - 6:** -> ***Configuring & Building a job using the feature Poll SCM in build triggers*** (created a job that pulls source code from SCM repository such as GitHub or Bit bucket and performs a build)

**Step-1:** Created a job namely PollSCMEx

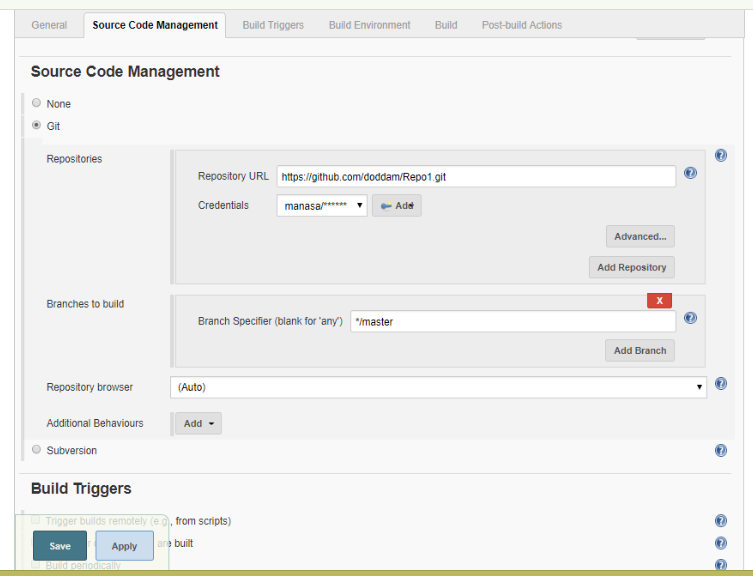


**Step-2:**

The job’s been configured to pull source code from the master branch of the given git repository.

 Type the URL of the SCM repository in the *Repository URL* text field.

We also need to provide user credentials so that Jenkins can access the repository (give Jenkins credentials).



**Step-3:**

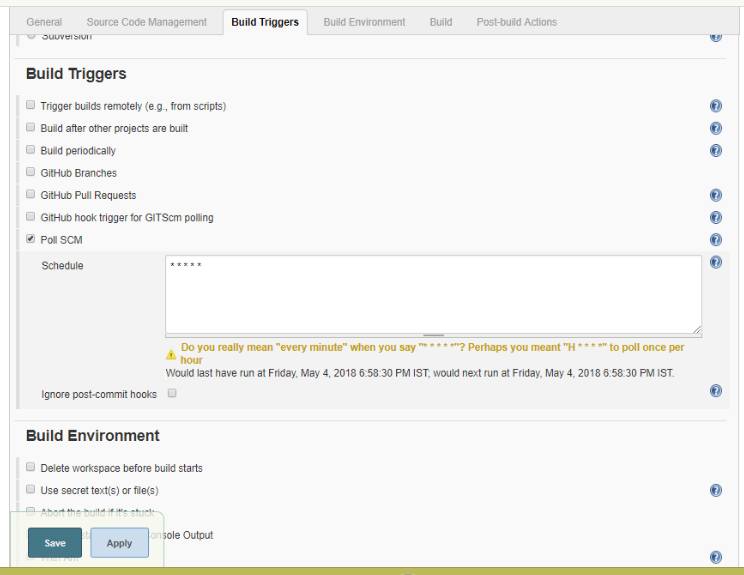
Check the below screenshot in Build Triggers tab, have selected Poll SCM option.

Here comes the actual scenario, how the cron job is scheduled or set -> in the same way the job is built by polling SCM.

I have set cron job as below,

\* \* \* \* \*

So, for every minute – SCM is polled and then git repository is checked for any changes, if so then a new build is made in Jenkins automatically after pulling source code from the git repo.



**Step-4:**

* I have created a Java application in a notepad and save as **Prog.java**

public class Prog

{

public static void main(String[] args)

{

for(int i=1;i<=5;i++)

{

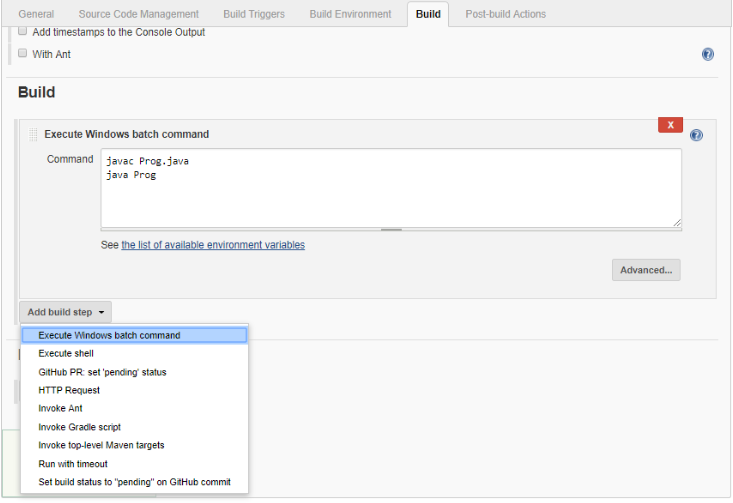
System.out.println("Hiiii ... "+i);

}

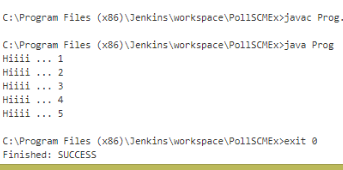
}

}

* Select execute windows batch command as we are working on windows
* Give the commands that are required to run an application.
* Save the configuration.

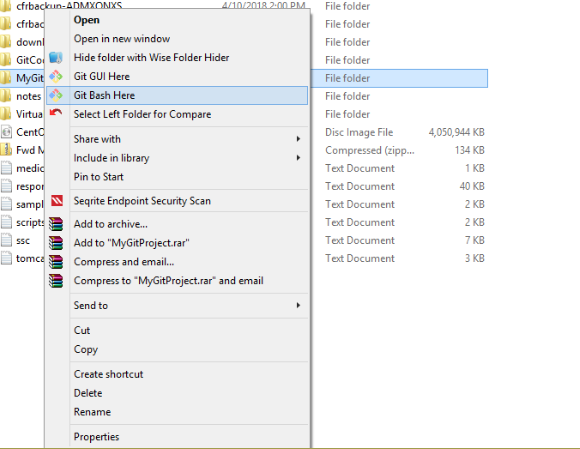


Now check the output:

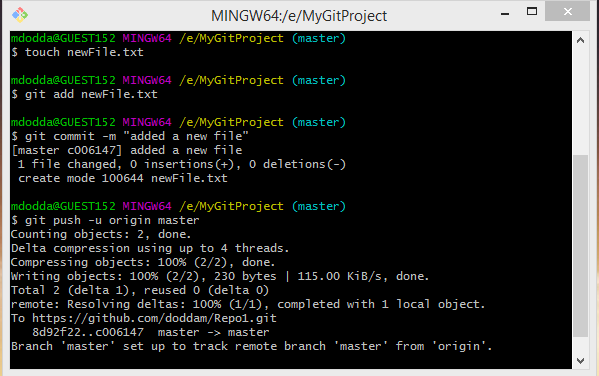


**Step-5:**

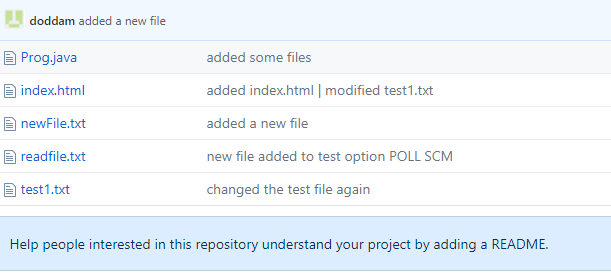
Go to the path where you have added git remote repository to the local system> right click > select Git Bash here



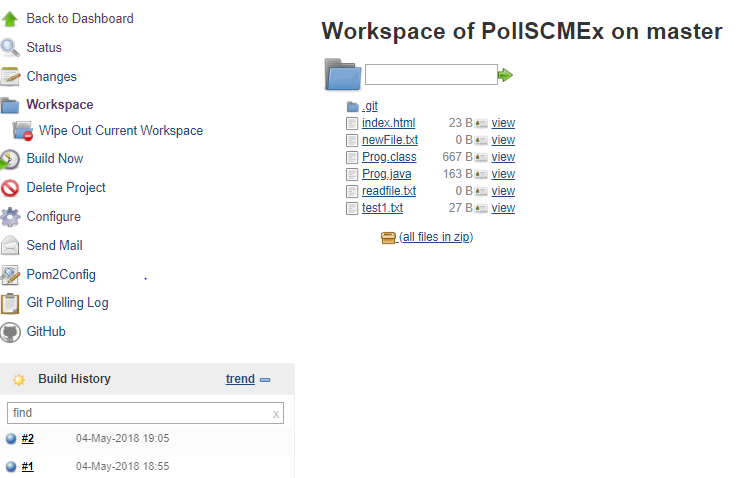
**Step-6:** Here a new file is added and pushed to git repo through git bash.



**Step-7:**  Now check in the git repository “newFile” is added.



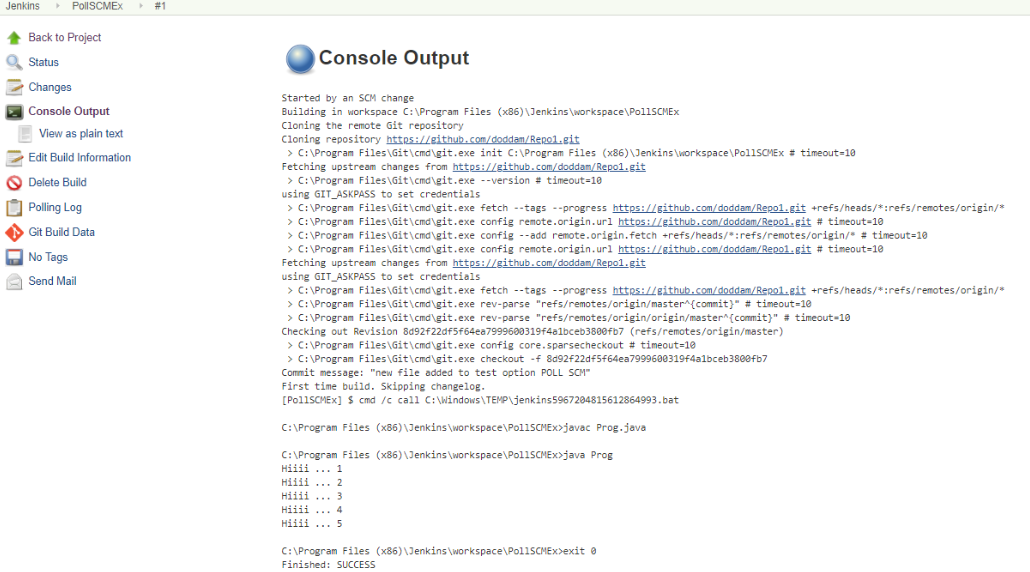
**Step-8:** Check the below screenshot, the file is added in the jenkins project dashboard.



**Step-9:**

After pulling the source code, the job will execute the script containing a provided windows execute command and

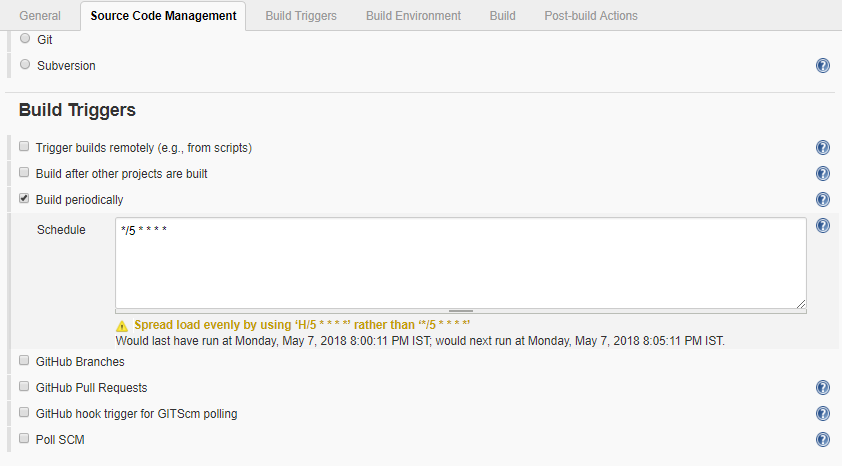
job is built automatically as some changes are made to the git repository and can check in console output “Started by an SCM change”.



At the end of the output, we should see *BUILD SUCCESS* message.

**Scenario - 5:** -> ***Scheduling a job in jenkins***

* In the job configuration page, scroll down straight to the *Build Triggers* section.
* Our intention is to create a straightforward job, select the checkbox marked *Build periodically*. As soon as we select this checkbox, a Text Box is displayed with the *Schedule* label.
* We have to provide value in a[cron-compliant format](http://www.baeldung.com/cron-expressions). There’s extensive information available on the page if we click the question mark next to the box.
* Let’s type *\*/2 \* \* \* \**  here, which represents an interval of two minutes(scheduling cron job)
* Save the above configuration.

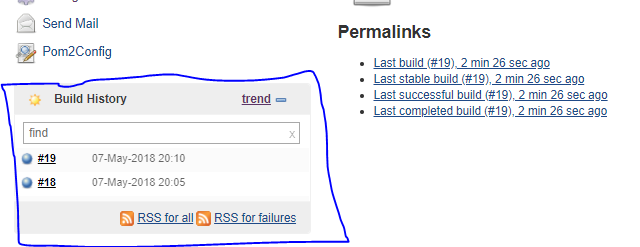


Upon tabbing out of the text box, we can see information right beneath the box. It tells us about when the job will run next.

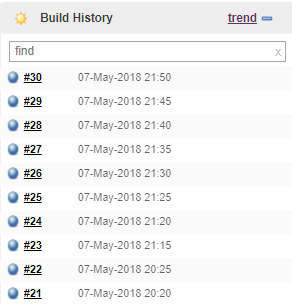
Let’s save the job – in about 5 minutes, we should see the status of the first execution of the job.

I scheduled job at 8:00, so at 8:05, 8:10 …. Job is built periodically.

Check the build history in the below screenshot,



We can observe in the below screenshot that the job is built for every 5 minutes periodically.



**Scenario-7:** -> ***How to setup delivery pipeline in Jenkins***

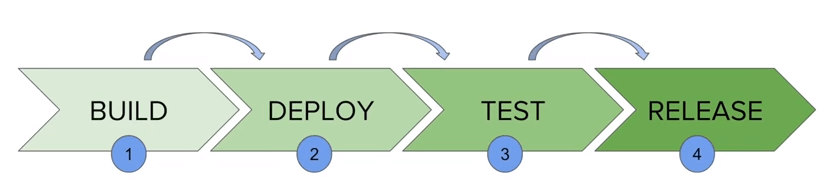
**Jenkins Pipeline:**

In Jenkins, a pipeline is a group of events or jobs which are interlinked with one another in a sequence.

In simple words, Jenkins Pipeline is a combination of plugins that support the integration and implementation of continuous delivery pipelines using Jenkins. A pipeline has an extensible automation server for creating simple or complex delivery pipelines "as code," via pipeline DSL (Domain-specific Language).

**What is Continuous Delivery Pipelines? How it Works?**

In a Jenkins pipeline, every job or event has some sort of dependency on at least one or more events.



The picture above represents a continuous delivery pipeline in Jenkins. It contains a group of states called build, deploy, test and release. These events are interlinked with each other. Every state has its events, which work in a sequence called a continuous delivery pipeline.

A continuous delivery pipeline is an automated expression to display your process for getting software for version control. Thus, every change made in your software goes through a number of complex processes on its way to being released. It also involves developing the software in a reliable and repeatable manner, and progression of the built software through multiple stages of testing and deployment.

**Why Use Jenkin's Pipeline?**

Jenkins is an open continuous integration server which has the ability to support the automation of software development processes. You can create multiple automation jobs with the help of use cases, and run them as a Jenkins pipeline.

Here are the reasons why you use should use Jenkins pipeline:

* Jenkins pipeline is implemented as a code which allows multiple users to edit and execute the pipeline process.
* Pipelines are robust. So if your server undergoes an unforeseen restart, the pipeline will be automatically resumed.
* You can pause the pipeline process and make it wait to resume until there is an input from the user.
* Jenkins Pipelines support big projects. You can run multiple jobs, and even use pipelines in a loop.

**Jenkins Pipeline Concepts**

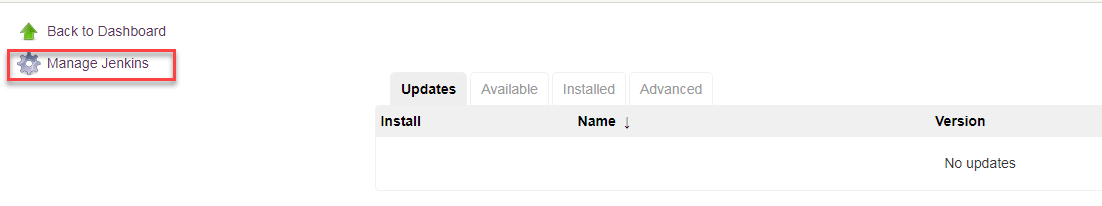
|  |  |
| --- | --- |
| **Term** | **Description** |
| Pipeline | The pipeline is a set of instructions given in the form of code for continuous delivery and consists of instructions needed for the entire build process. With pipeline, you can build, test, and deliver the application. |
| Node | The machine on which Jenkins runs is called a node. A node block is mainly used in scripted pipeline syntax. |
| Stage | A stage block contains a series of steps in a pipeline. That is, the build, test, and deploy processes all come together in a stage. Generally, a stage block is used to visualize the Jenkins pipeline process. |
| Step | A step is nothing but a single task that executes a specific process at a defined time. A pipeline involves a series of steps. |

## Install Delivery Pipeline Plugin in Jenkins

## This plugin visualize Delivery Pipelines (Jobs with upstream/downstream dependencies)

## Here is how you can install the build pipeline plugin in your Jenkins:

**Step-1:** The settings for the plugin can be found under **Manage Jenkins > Manage Plugins.**



If you have already installed the plugin, it is shown under the installed tab.



**Step-2:** If you do not have the plugin previously installed, it shows up under the Available tab.

You can Select the plugin and install it.

Once you have successfully installed the **delivery pipeline** plugin in your Jenkins, follow these steps.

**Delivery Pipeline Plugin:**

Jenkins Pipeline (or simply "Pipeline") is a suite of plugins which supports implementing and integrating *continuous delivery pipelines* into Jenkins.

A *continuous delivery pipeline* is an automated expression of your process for getting software from version control right through to your users and customers.

Visualization of Continuous Delivery pipelines. Renders pipelines based on upstream/downstream jobs or Jenkins pipelines. Provides a full screen view for information radiators.  
In Continuous Delivery feedback and visualisation of the delivery process is one of the most important areas. When using Jenkins as a build server it is with the Delivery Pipeline plugin possible to visualize one or more delivery pipelines in the same view, even in full screen.

**To setup a delivery pipeline:**

**Step - 1:** Chain the required jobs in sequence

For every job, add a dependent upstream or downstream job.

**Upstream job:** A job to be executed before current job

**Downstream job:** A job to be executed after current job

Creating sample jobs for demo

To chain the jobs first we need to create the sample jobs.

Created three jobs of type FreeStyle project

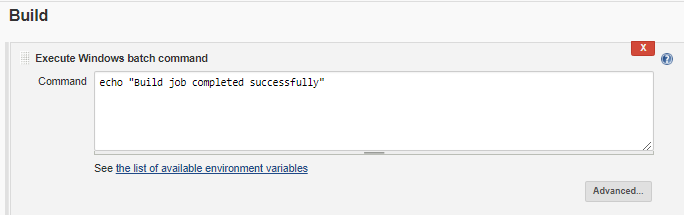


Apply and save the jobs. Once created, the jobs can be seen in the dashboard.

**Step - 2:** Configured each job by adding build step in **Build Environment** tab > Execute windows batch command

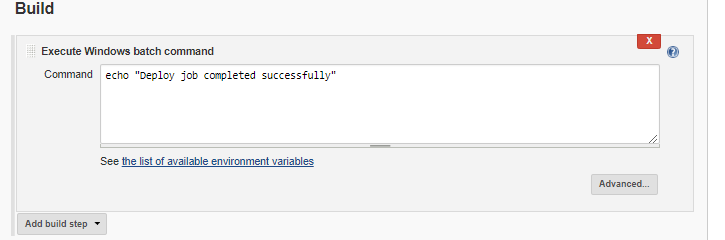
Provide the sample commands to be executed and save it.

For SampleBuildJob,



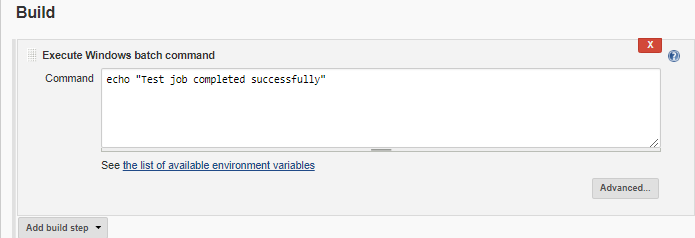
Click **Save**.

For SampleDeployJob,



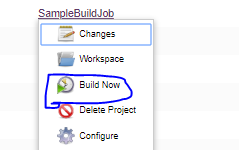
Click **Save.**

For SampleTestJob,

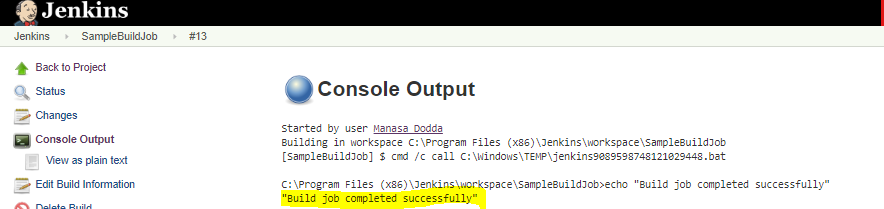


Click **Save.**

**Step - 3:** Run the jobs to test them working fine or not.



Click on the arrow beside the job name, the list above is shown > click on **Build Now** project.



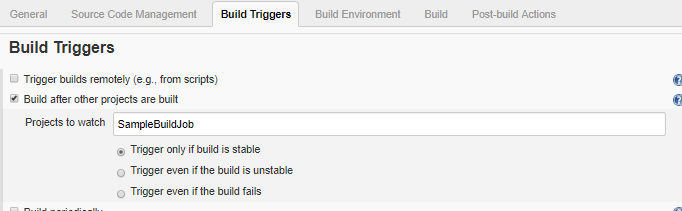
You can check the output in the console output.

Same process done with the other two jobs also.

**Step - 4:** Add upstream/downstream jobs.

Added an upstream job for **SampleDeployJob** to trigger and execute immediately after **SampleBuildJob** is executed or built successfully. To add the upstream job,

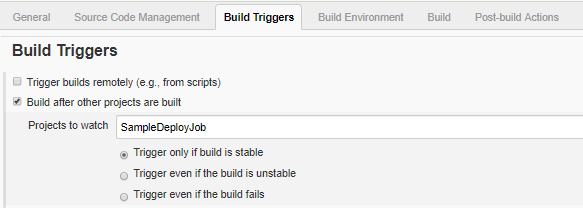
Click on the project SampleDeployJob > Configure > Build triggers tab (provide details) > Save



Made **SampleDeployJob** as a dependent job.

In the same process, making **SampleTestJob** job as dependent by adding upstream as **SampleDeployJob.**

Click on the project SampleTestJob > Configure > Build triggers tab (provide details) > Save



**(**Build triggers option is not applicable for the first job by default**)**

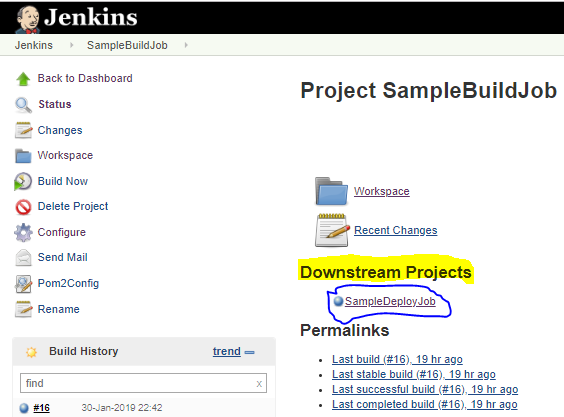
**Sequence created by chaining the below jobs**:

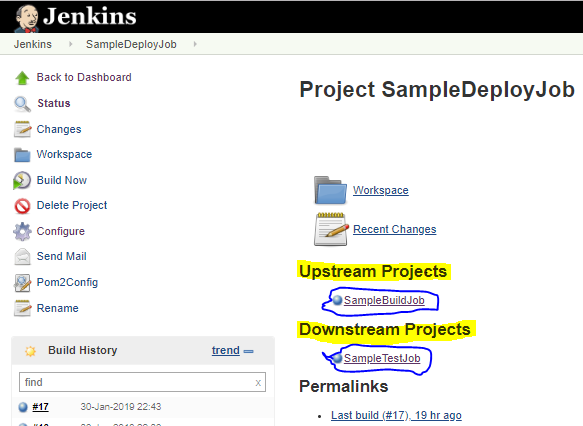
**SampleBuildJob SampleDeployJob SampleTestJob**

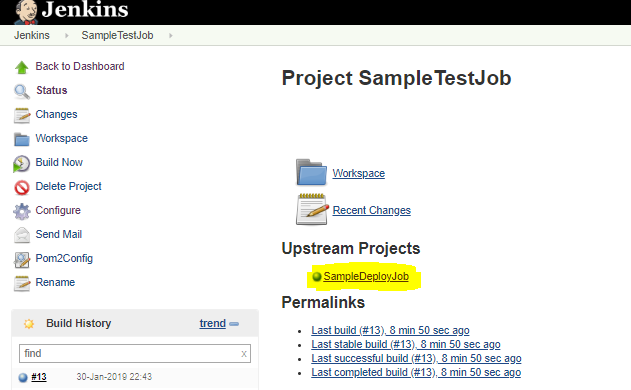
**Step - 5:** Running the sequence of jobs

Once above configuration is done,

Now, start building the job (**SampleBuildJob)** that is first chained and then other two jobs are triggered one after other if the job is stable.



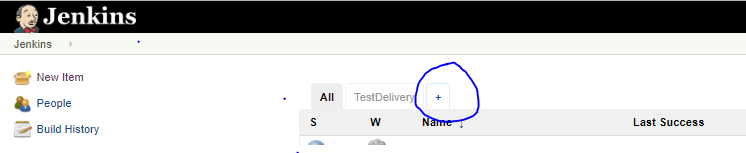




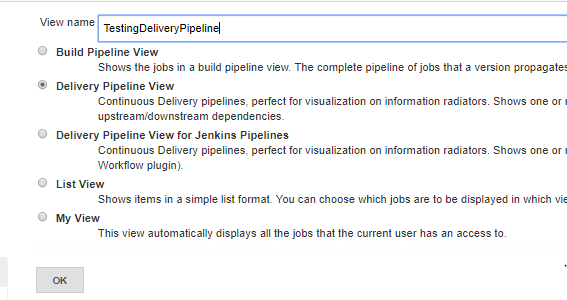
In the above screenshots, we can clearly observe the mentioned upstream and downstream jobs for a particular job.

**Step - 6:** Add delivery pipeline view in Jenkins

Creating a view in jenkins dashboard by clicking plus ‘+’ icon as shown in the below screenshot.



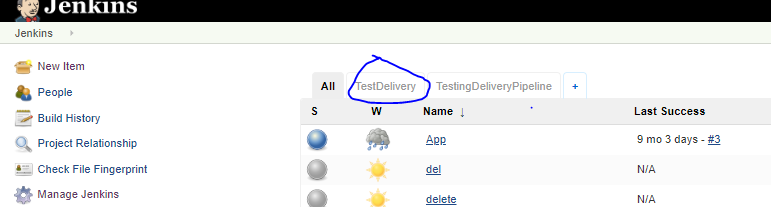
Provide a view name and click **OK** by selecting the respective view type



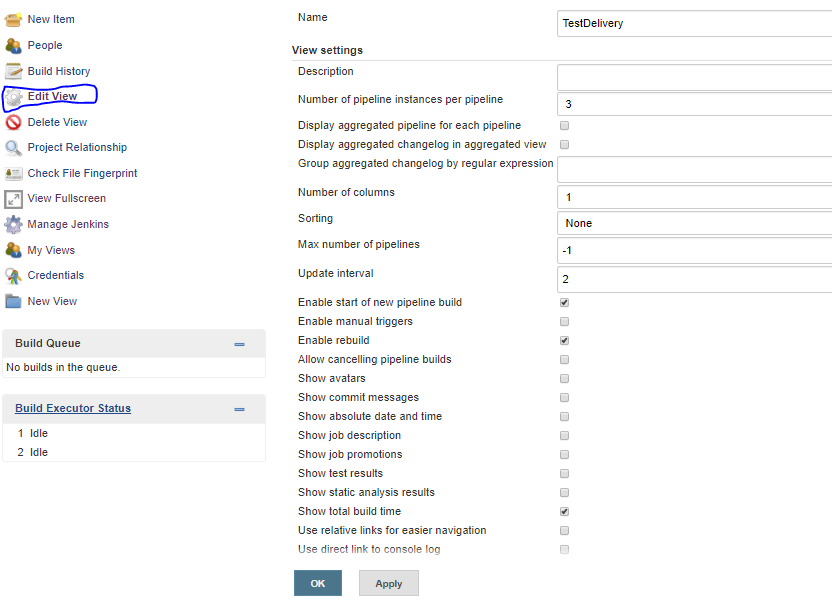
Before configuring the new view, let us verify the view (TestDelivery) which I have already created and configured.

To open the view and to verify/modify/update the settings,

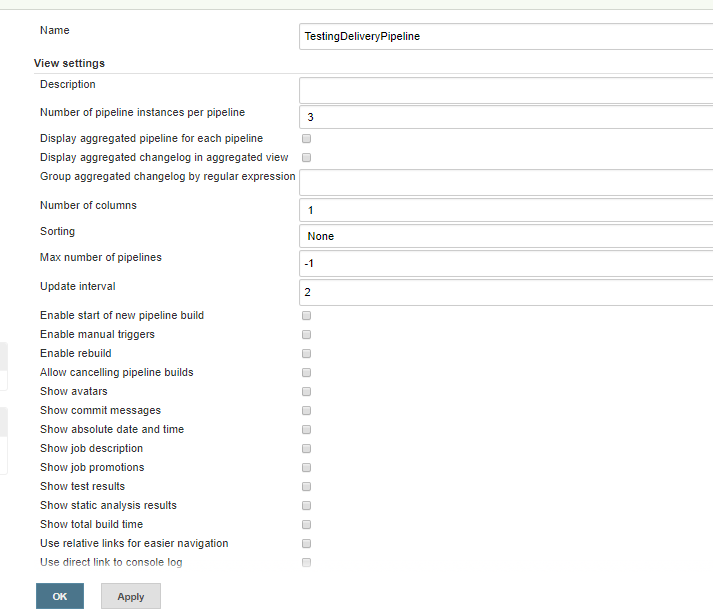
Open the view **TestDelivery** > Click on the **Edit View** present on the left side of Jenkins dashboard.



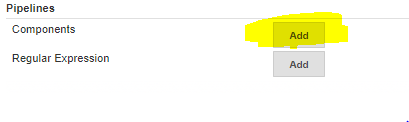
Check the below settings made for the TestDelivery view,



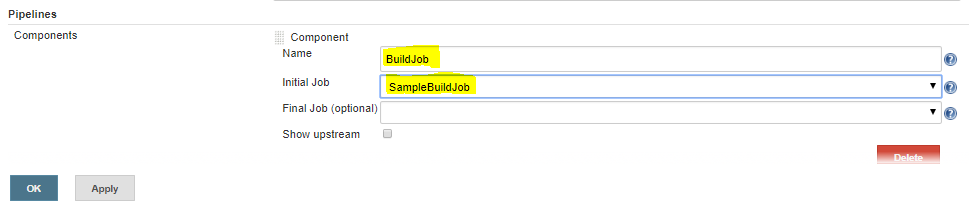
Open the newly created view, find the default settings below



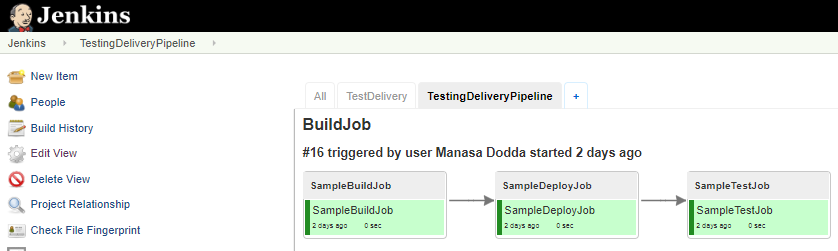
By keeping all the values as default, click **add button** to add the component name.



Now, provide the component name and initial job name > Apply > OK as shown below

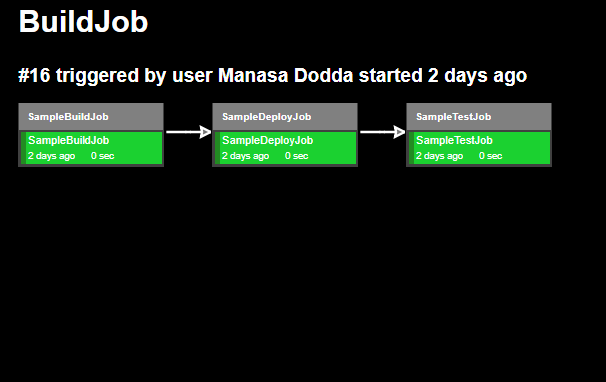


The below screen is opened when you click **OK** and it will show all the details of past runs.



View full screen shows the different view of the same pipeline:

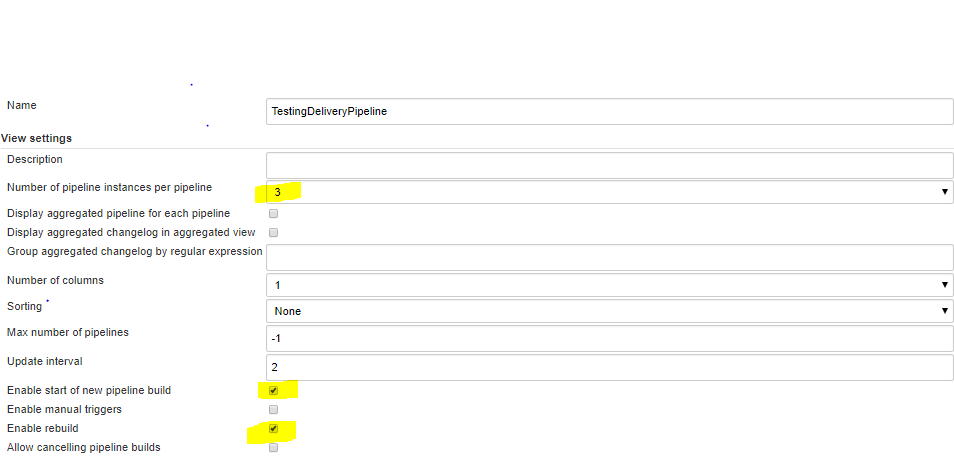
Click on **View Fullscreen** present on the left side of the dashboard



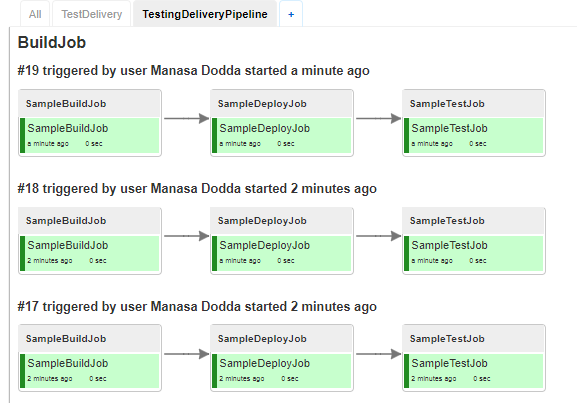
Modified the following settings,

* Number of pipeline instances per pipeline
* Starting new pipeline build
* Allowing rebuild
* Showing total build time
* Changing the theme for visualization
* Added the component name and also mentioned the initial job

The above configurations are explained below,

1. Number of pipeline instances per pipeline

Instance = 3; is the number of execution cycles to be visible on screen (view) as shown below,



1. Starting new pipeline build

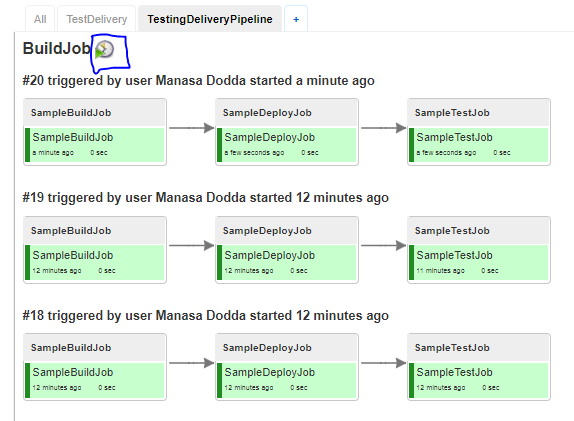
Allow starting a new pipeline run from the delivery pipeline view.



In the Edit View > enable the below > click OK

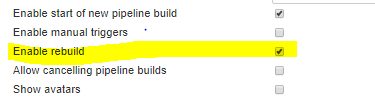
By clicking OK, the below view is opened.

The job is built from the view when we click on the **Build job** icon shown on the left side top of the view.

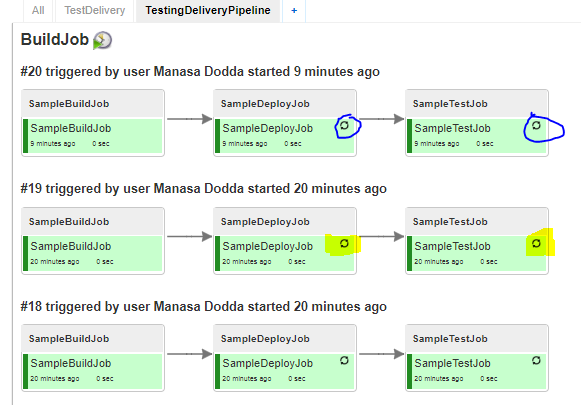


1. Enabling rebuild

Allow rerunning a task from the delivery pipeline view.



Click OK after enabling the rebuild option.



The rebuild symbol is visible on the view.

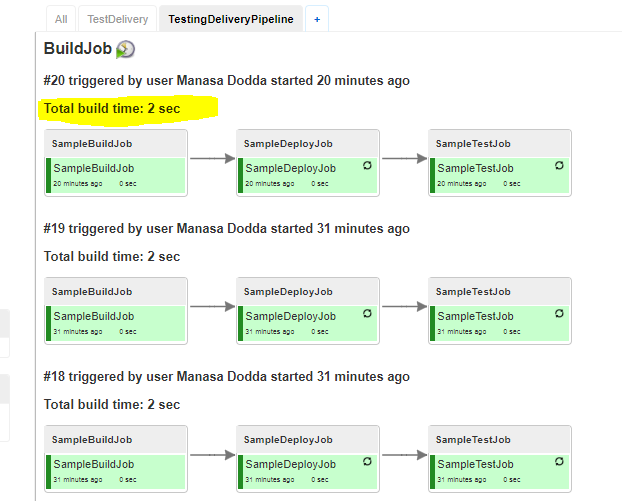
1. Showing the total build time.



Show total build time for a pipeline run.

If there are multiple routes in a pipeline, total build time is calculated as the sum of the build times in the longest route.

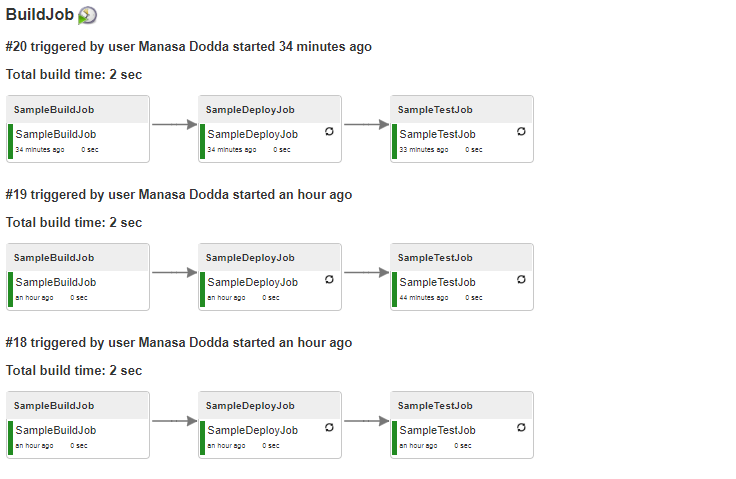
Edit View > Enable the option > Click OK



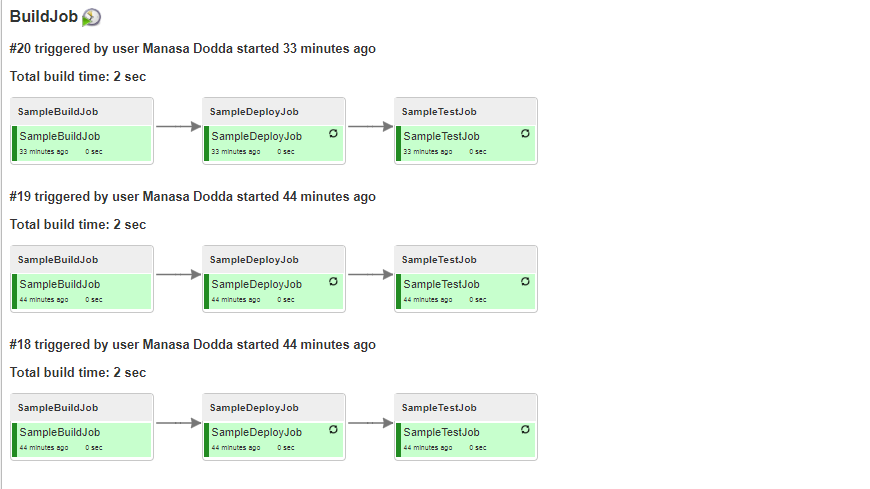
1. Changing the **theme** for visualization

There are three different modes to have a look at the view

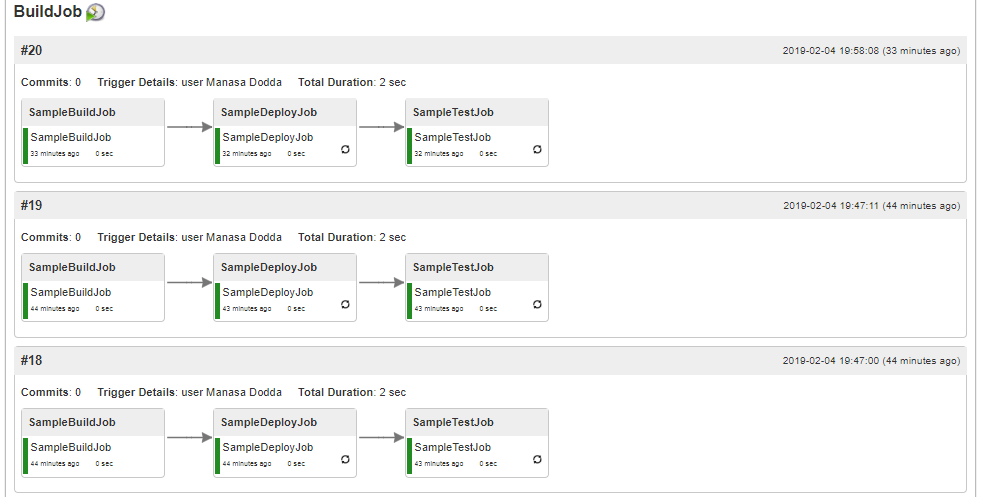
1. Default
2. Contrast
3. Overview
4. By selecting the ***default*** theme, view is as follows,



1. By selecting the ***contrast*** theme, view is as follows,



1. By selecting the ***overview*** theme, view is as follows,



**Scenario-8:** -> ***How to setup build pipeline in Jenkins***

**Build Pipeline Plugin:**

This plugin provides a Build Pipeline View of upstream and downstream connected jobs that typically form a build pipeline.  In addition, it offers the ability to define manual triggers for jobs that require intervention prior to execution, e.g. an approval process outside of Jenkins.

## Overview

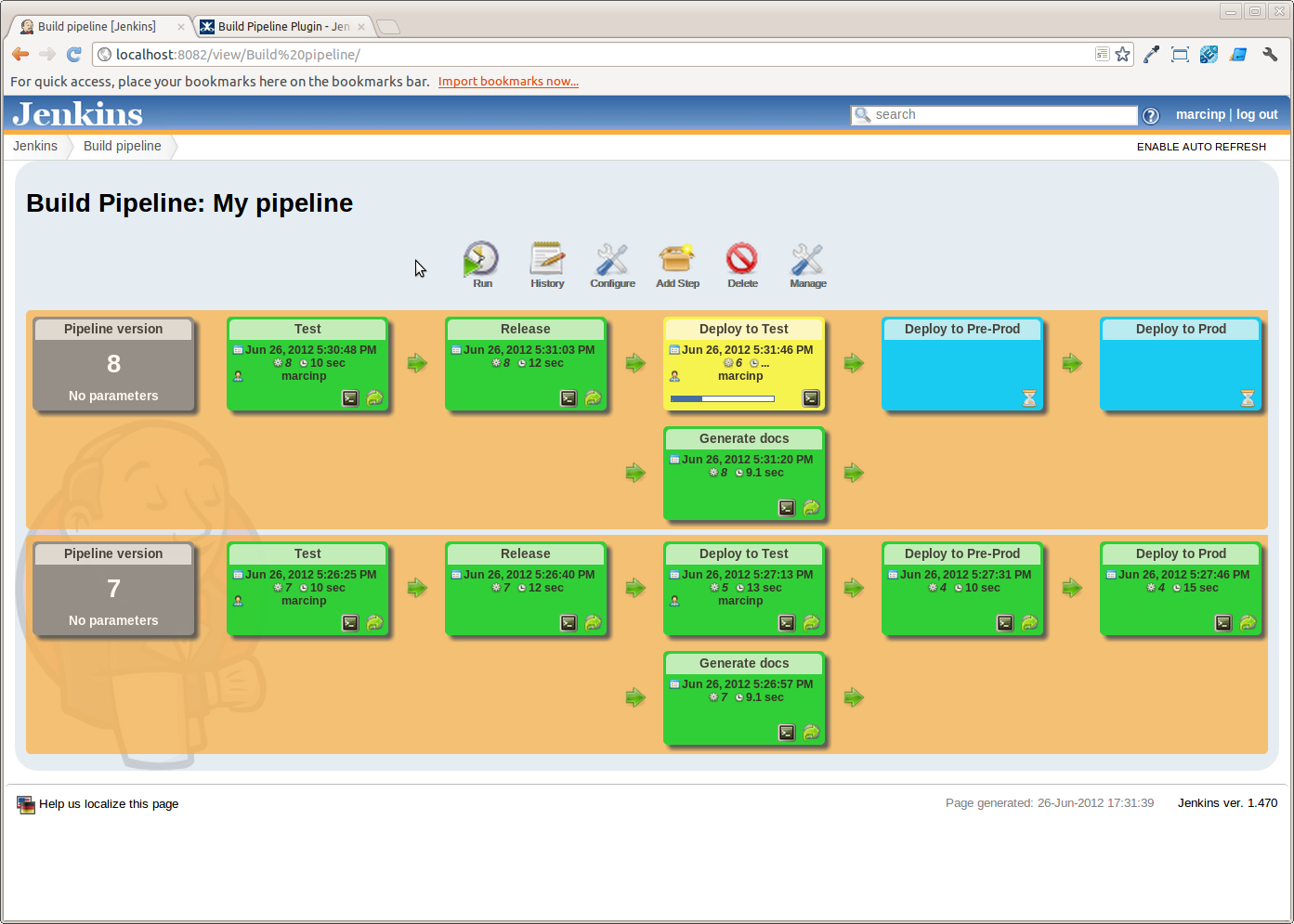
Continuous Integration has become a widely adopted practice in modern software development. Jenkins & Hudson are great tools for supporting Continuous Integration.

**Taking it to the next level:** Continuous integration can become the centerpiece of your [deployment pipeline](http://www.informit.com/articles/article.aspx?p=1621865), orchestrating the promotion of a version of software through quality gates and into production. By extending the concepts of CI you can create a chain of jobs each one subjecting your build to quality assurance steps. These QA steps may be a combination of manual and automated steps. Once a build has passed all these, it can be automatically deployed into production.

In order to better support this process, we have developed the Build Pipeline Plugin. This gives the ability to form a chain of jobs based on their upstream\downstream dependencies. Downstream jobs may, as per the default behaviours, be triggered automatically, or by a suitable authorised user manually triggering it.

You can also see a history of pipelines in a view, the current status and where each version got to in the chain based on its revision number in VCS.

#### The Pipeline View



## Configuration

### **View Configuration**

1. Install the plugin using the Hudson\Jenkins Plugin Manager and restart.
2. Create a view of the new type *Build Pipeline View*.  
   You will then be redirected directly to the configuration page.
3. The table below outlines what each interesting parameter controls:

| **Name** | 1. The name of the Build Pipeline View |
| --- | --- |
| **Description** | This message will be displayed on the view page. Useful for describing what this view is about, or linking to relevant resources. Can contain HTML tags. |
| **Build Pipeline View Title** | Gives a title to the page that displays the view |
| **Select Initial Job** | This is the first job in the build pipeline. It will traverse through the downstream jobs to build up the entire build pipeline.  Select from a drop-down list of jobs. |
| **No of Displayed Builds** | The number of historical builds to be displayed on a page. |
| **Restrict triggers to most recent successful builds** | Select this option to restrict the display of a Trigger button to only the most recent successful build pipelines.  Yes: Only the most recent successful builds displayed on the view will have a manual trigger button for the next build in the pipeline.  No: All successful builds displayed on the view will have a manual trigger button for the next build in the pipeline. |
| **Always allow manual trigger on pipeline steps** | Select this option if you want to manually execute or re-execute any step of the pipeline at any time. |
| **Show pipeline parameters** | Select this option if you want to display the parameters used to run the first job in the pipeline. |

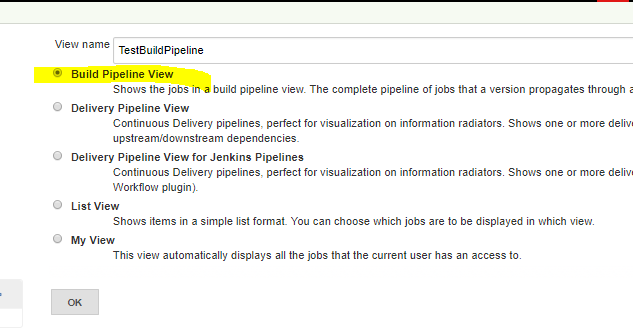
Following steps are created for the demo purpose on the build pipeline:

* Created three sample jobs for the demo purpose to work with the build pipeline.
* Created the sequence of the jobs by chaining the required jobs.
* Configured the jobs by adding the upstream and downstream jobs basing on the requirement.
* Ran and validated the jobs in the dashboard that are chained in the sequence.

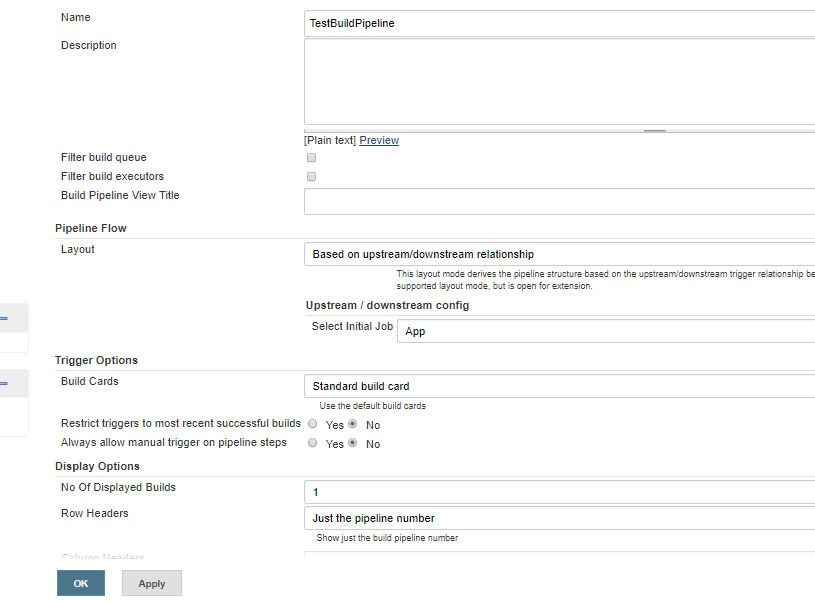
(for the above steps follow scenario number 7)

Follow the below steps to work with the build pipeline view:

**Step - 1:** Created a new view ‘TestBuildPipeline’ of type Build pipeline and click **OK**.

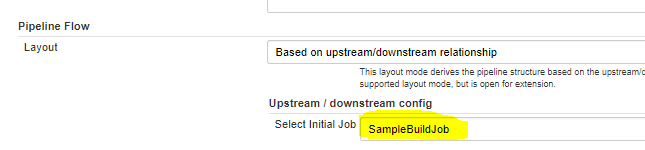


**Step - 2:** Configuration page is opened when the new view is created as shown below.

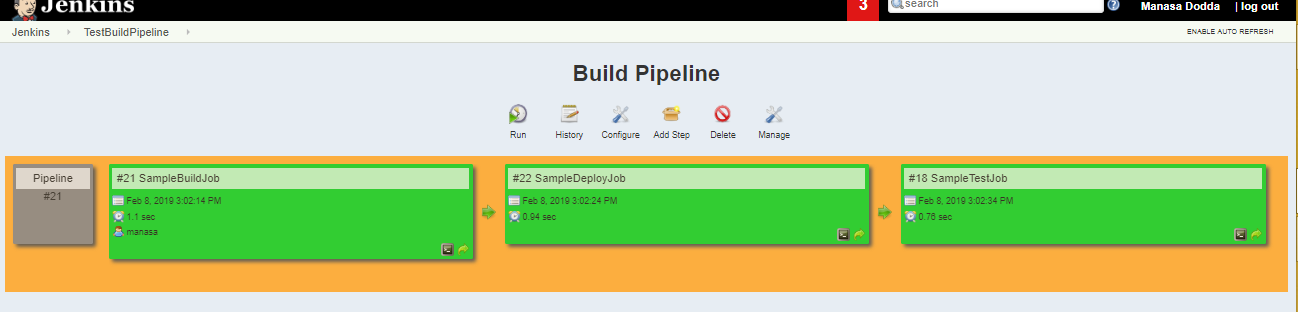


**Step - 3:** Configured the below things,

1. Provided the initial or parent Job in the build pipeline view.
2. The number of build pipelines to display in the view.
3. The console output link style to view the output in a new window.
4. When the initial job name is configured, click OK.

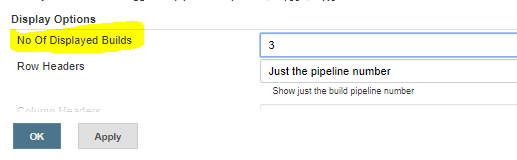


The following screen is displayed,

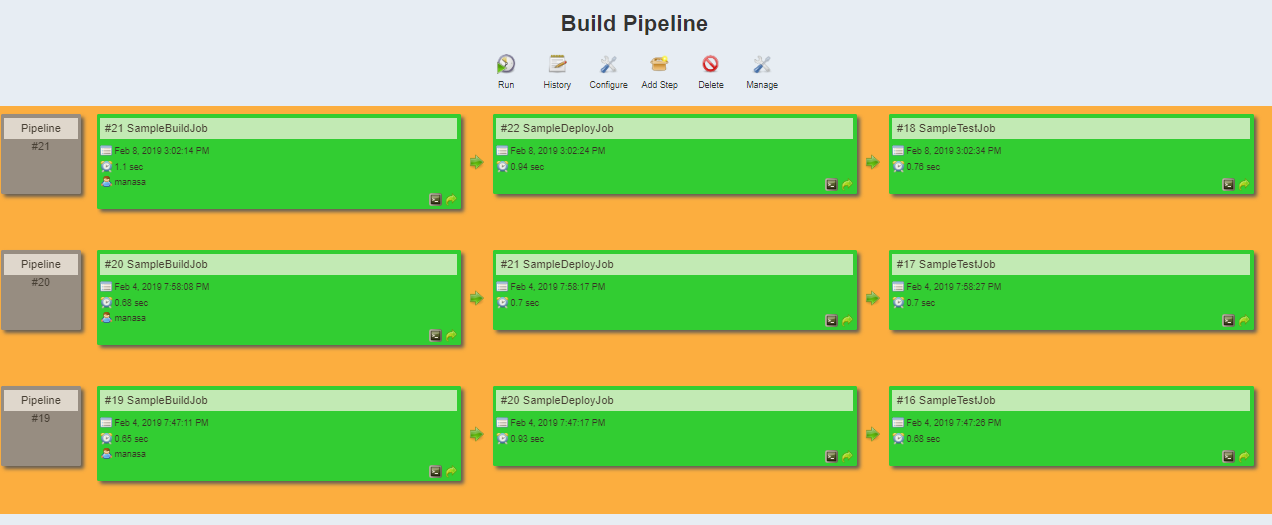


Now click on configure icon present on the top to get back to the configuration page.

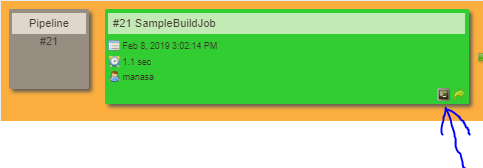
1. Modified the number of build pipelines to be displayed in the view.



Now click OK, and then 3 builds are displayed.



1. Changed the console output link style to view the output in a **new window**.



When we click on the above icon which is marked with an arrow mark, then a new window is opened to display the console output.

