# **Continuous Delivery Pipeline Using Jenkins:**

We will be using Jenkins to create a Continuous Delivery Pipeline, which will include the following tasks:

## **The following are the steps involved :**

* Fetching the code from GitHub
* Compiling the source code
* Unit testing and generating the JUnit test reports
* Packaging the application into a WAR file and deploying it on the Tomcat server



Pre-requisites:

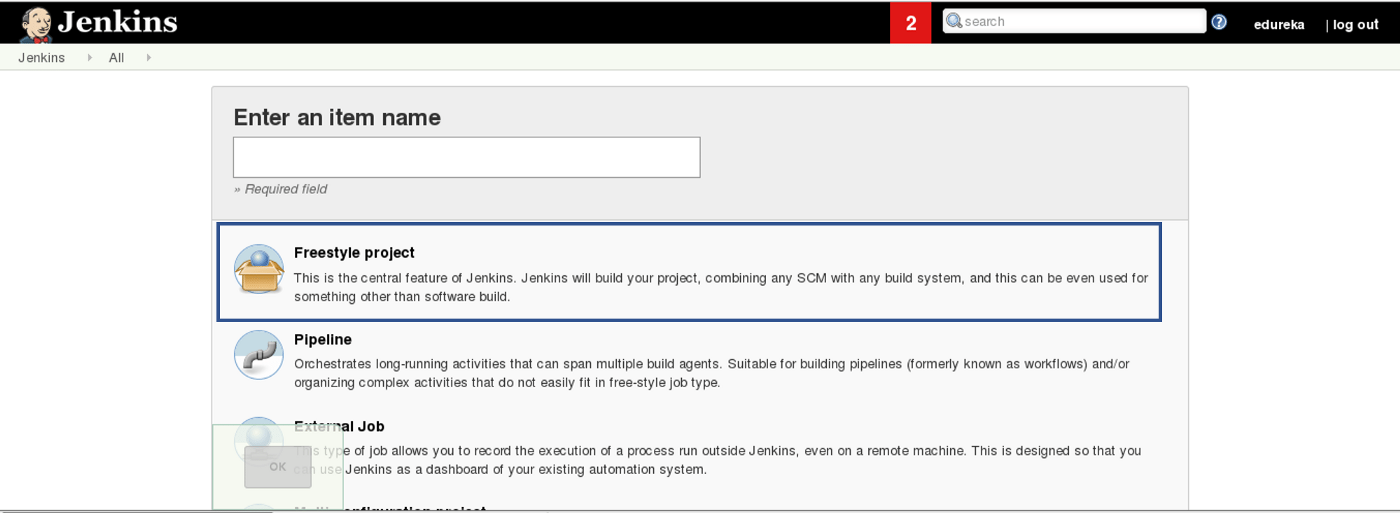
* CentOS 7 Machine
* Jenkins 2.121.1
* Docker
* Tomcat 7

## **Step — 1 Compiling The Source Code:**

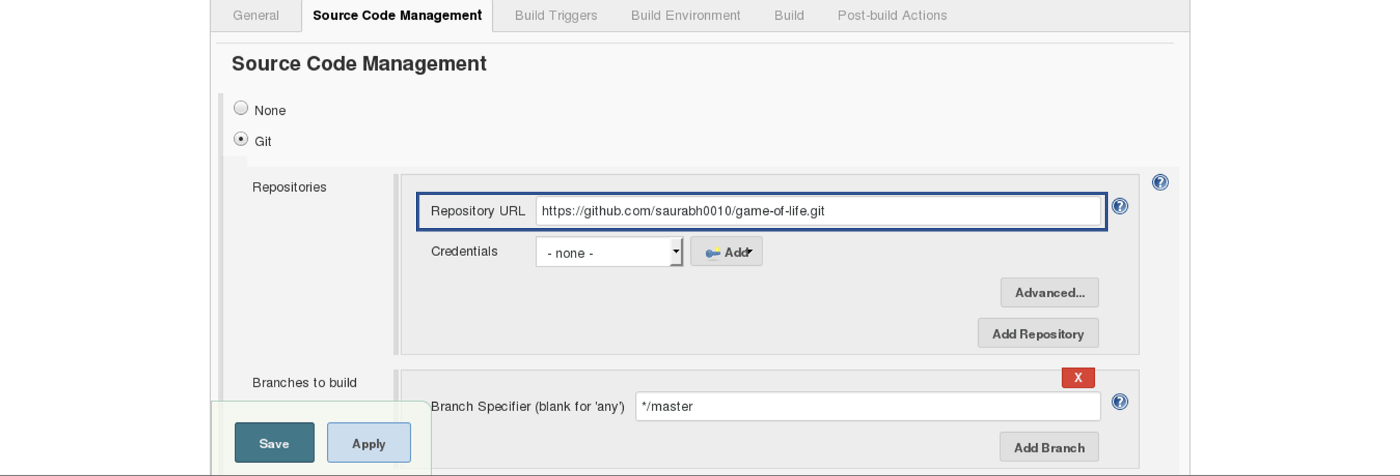
Let’s begin by first creating a Freestyle project in Jenkins. Consider the below screenshot:



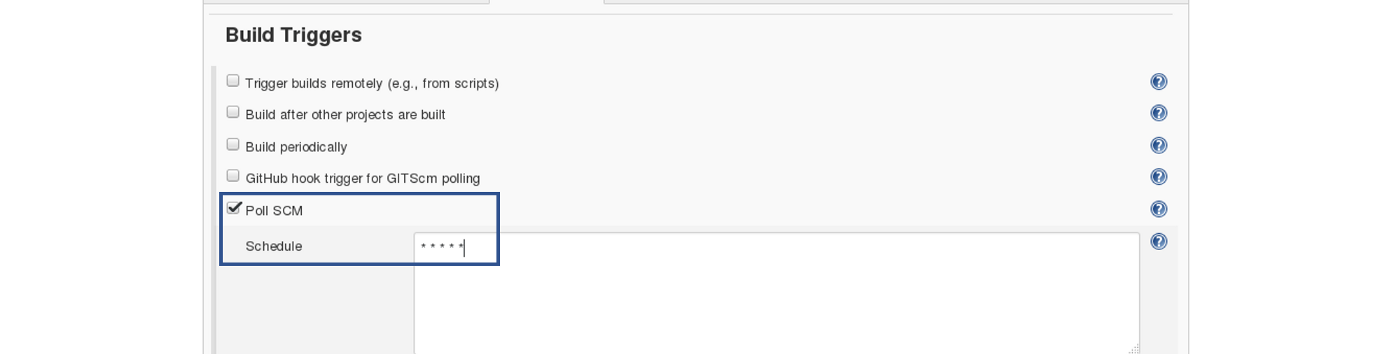
Give a name to your project and select Freestyle Project:



When you scroll down you will find an option to add source code repository, select git and add the repository URL, in that repository there is a pom.xml fine which we will use to build our project. Consider the below screenshot:



Now we will add a Build Trigger. Pick the poll SCM option, basically, we will configure Jenkins to poll the GitHub repository after every 5 minutes for changes in the code. Consider the below screenshot:



**Maven Build Cycle:**

Each of the build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

Following is the list of build phases:

* 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 built environment, copies the final package to the remote repository for sharing with other developers and projects.

Run the below command, for compiling the source code, unit testing, and even packaging the application in a war file:

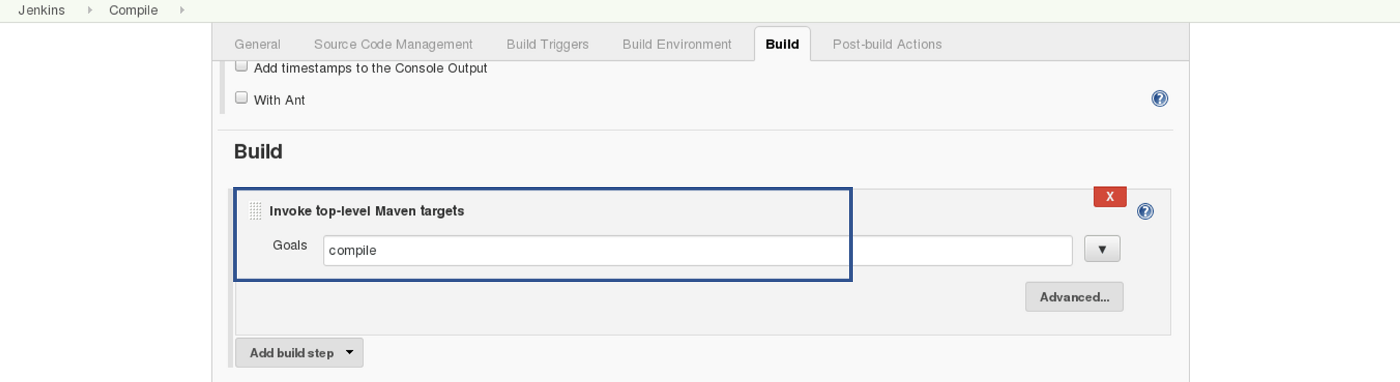
mvn clean package

You can also break down your build job into a number of build steps. This makes it easier to organize builds in clean, separate stages.

So we will begin by compiling the source code. In the build tab, click on invoke top-level maven targets and type the below command:

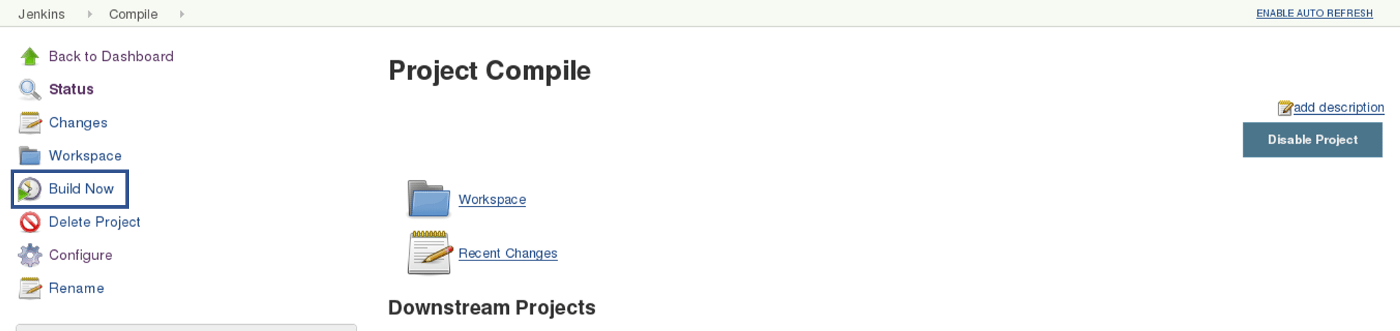
compile

Consider the below screenshot:

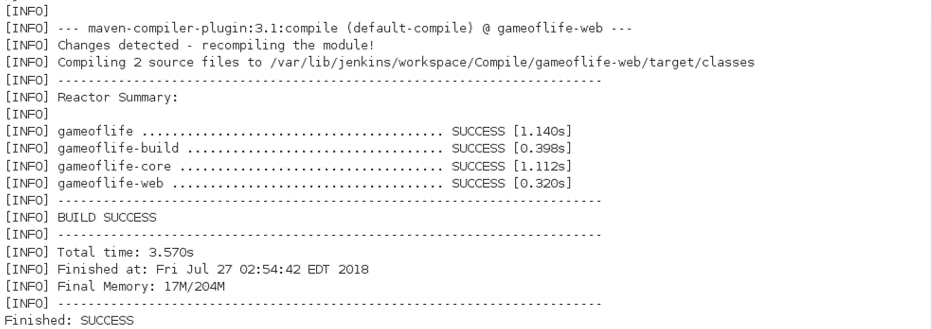


This will pull the source code from the GitHub repository and will also compile it (Maven Compile Phase).

Click on Save and run the project.



Now, click on the console output to see the result.



Now we will create one more Freestyle Project for unit testing.

## **Step — 2 Unit Testing:**

Add the same repository URL in the source code management tab, as done in the previous job.

Now, in the “Buid Trigger” tab click on the “build after other projects are built”. There type the name of the previous project where we are compiling the source code, and you can select any of the below options:

* Trigger only if the build is stable
* Trigger even if the build is unstable
* Trigger even if the build fails

the above options are pretty self-explanatory so, select anyone. Consider the below screenshot:



test

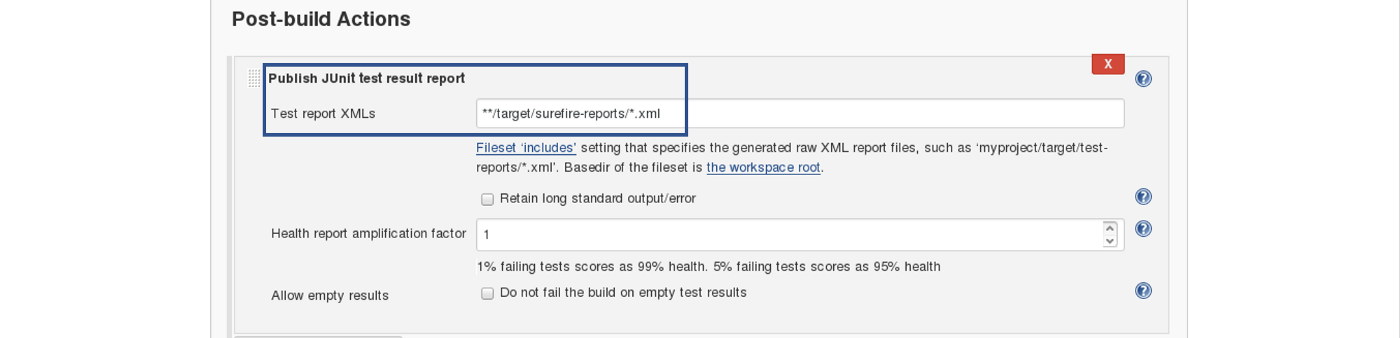
Jenkins also does a great job of helping you display your test results and test result trends.

The de facto standard for test reporting in the Java world is an XML format used by JUnit. This format is also used by many other Java testing tools, such as TestNG, Spock, and Easyb. Jenkins understands this format, so if your build produces JUnit XML test results, Jenkins can generate nice graphical test reports and statistics on test results over time, and also let you view the details of any test failures. Jenkins also keeps track of how long your tests take to run, both globally, and per test-this can come in handy if you need to track down performance issues.

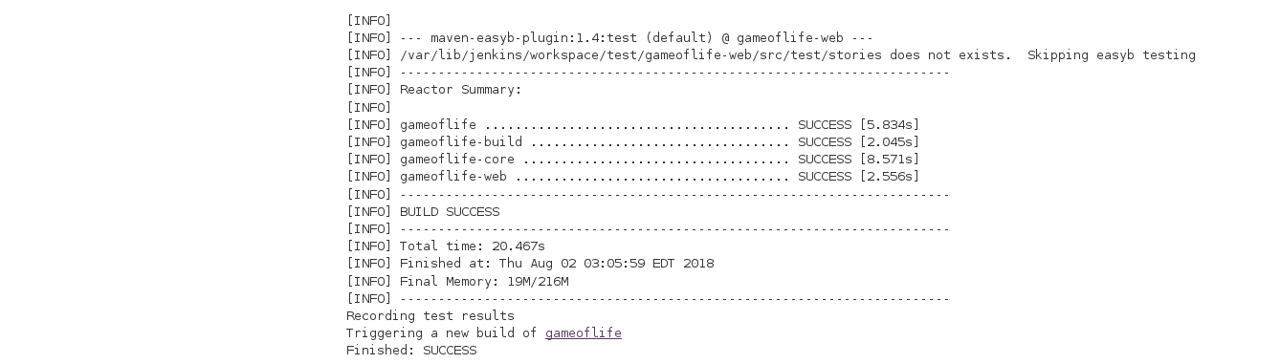
So the next thing we need to do is to get Jenkins to keep tabs on our unit tests.

Go to the **Post-build Actions** section and tick **“Publish JUnit test result report” checkbox.** When Maven runs unit tests in a project, it automatically generates the XML test reports in a directory called surefire-reports . So enter **“\*\*/target/surefire-reports/\*.xml”** in the **“Test report XMLs”** field. The two asterisks at the start of the path **(“\*\*”)** are a best practice to make the configuration a bit more robust: they allow Jenkins to find the target directory no matter how we have configured Jenkins to check out the source code.

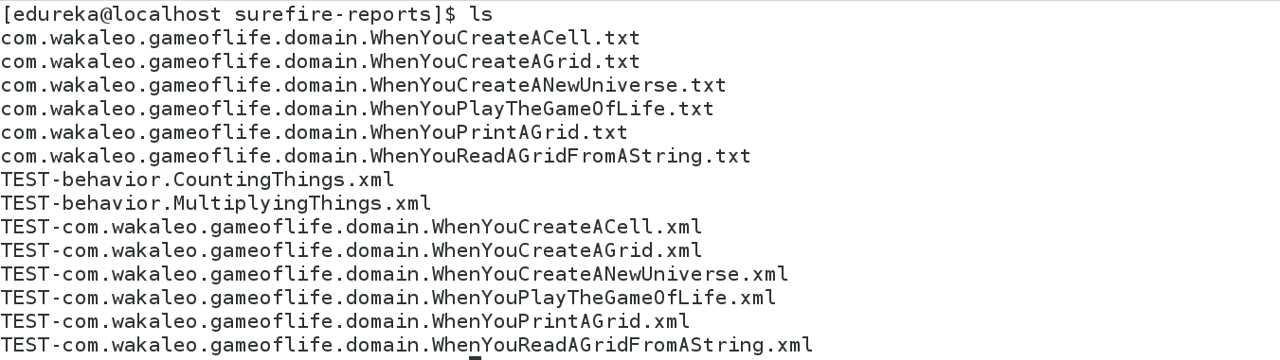
\*\*/target/surefire-reports/\*.xml



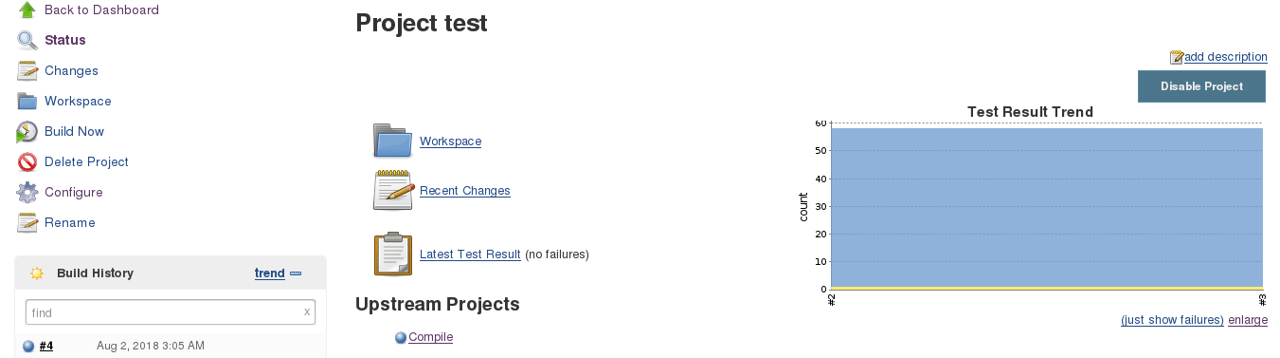
Again save it and click on Build Now.



Now, the JUnit report is written to /var/lib/jenkins/workspace/test/gameoflife-core/target/surefire-reports/TEST-behavior.



In the Jenkins dashboard you can also notice the test results:

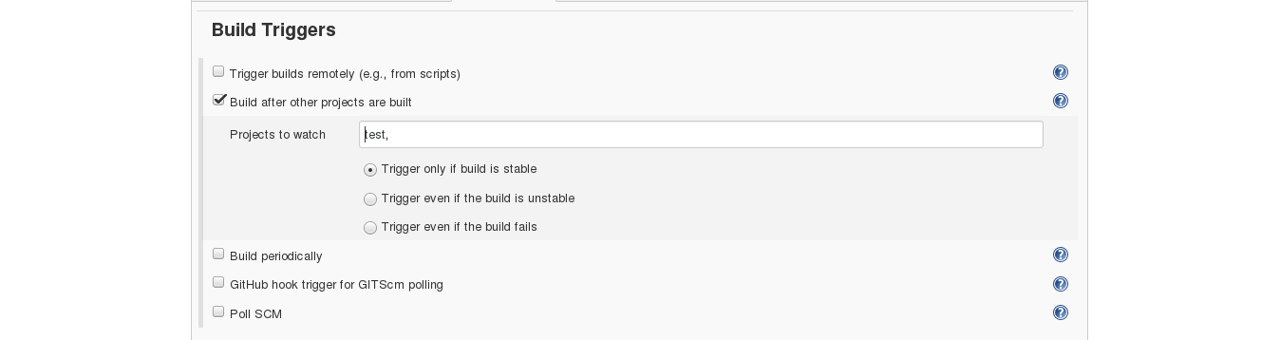


## **Step — 3 Creating A WAR File And Deploying On The Tomcat Server:**

Now, the next step is to package our application in a WAR file and deploy that on the Tomcat server for the User Acceptance test.

Create one more freestyle project and add the source code repository URL.

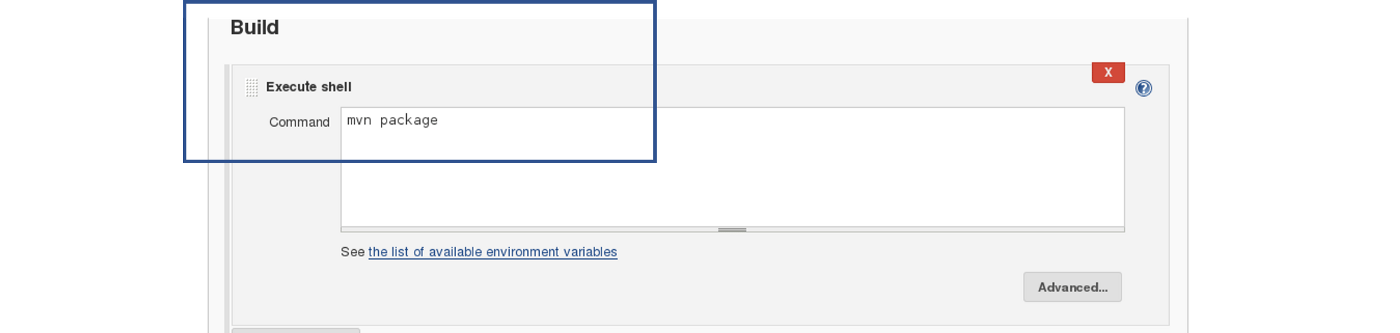
Then in the build trigger tab, select build when other projects are built, consider the below screenshot:



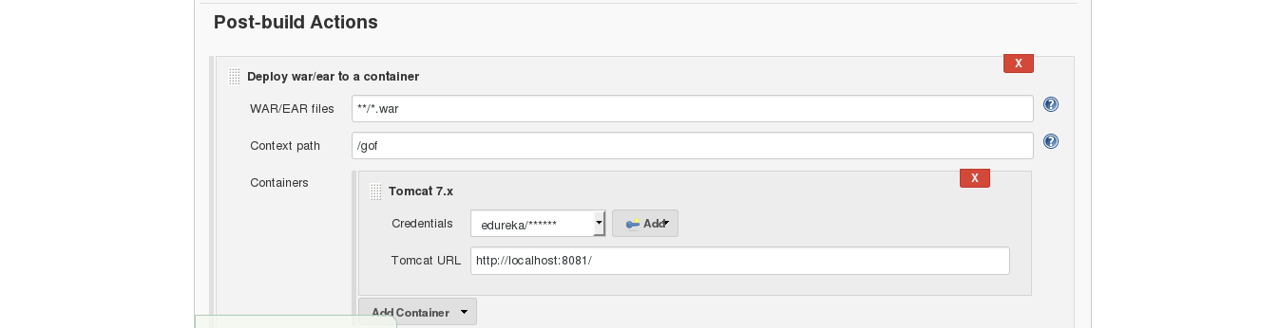
Basically, after the test job, the deployment phase will start automatically.

In the build tab, select shell script. Type the below command to package the application in a WAR file:

mvn package

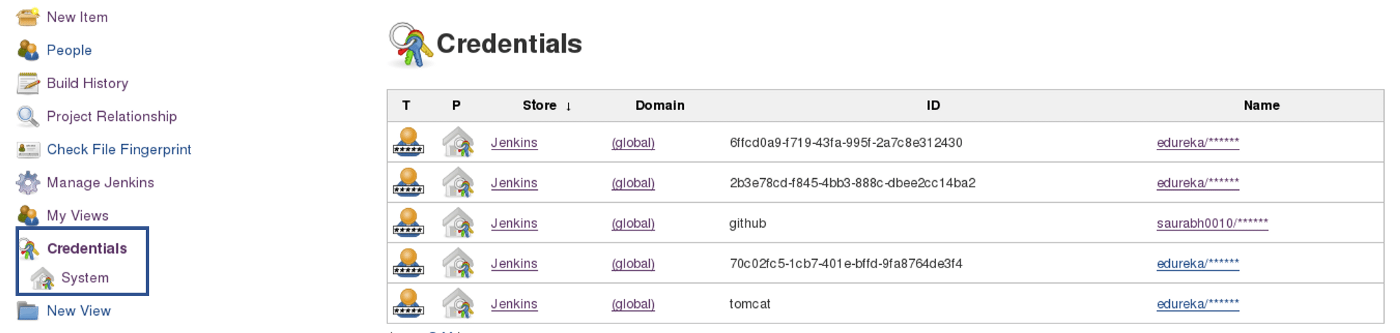


Next step is to deploy this WAR file to the Tomcat server. In the “Post-Build Actions” tab select deploy war/ear to a container. Here, give the path to the war file and give the context path. Consider the below screenshot:

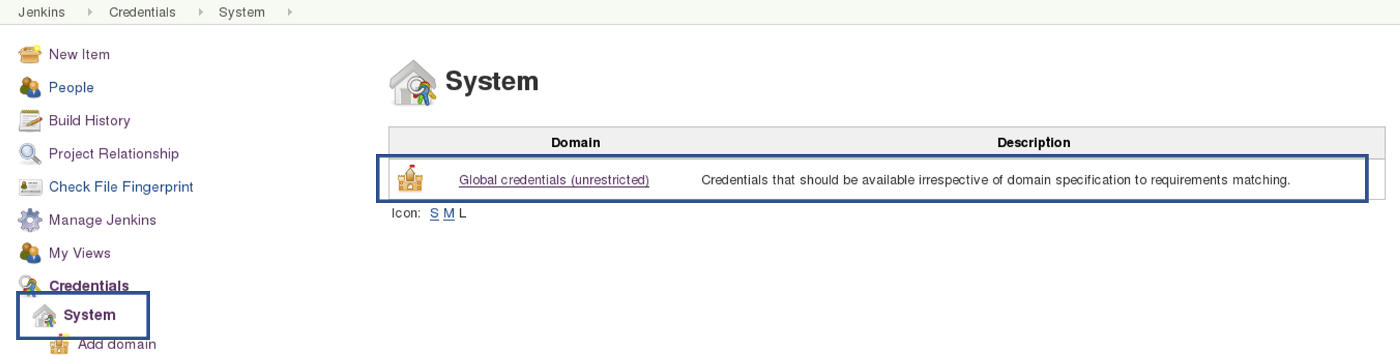


Select the Tomcat credentials and, notice the above screenshot. Also, you need to give the URL of your Tomcat server.

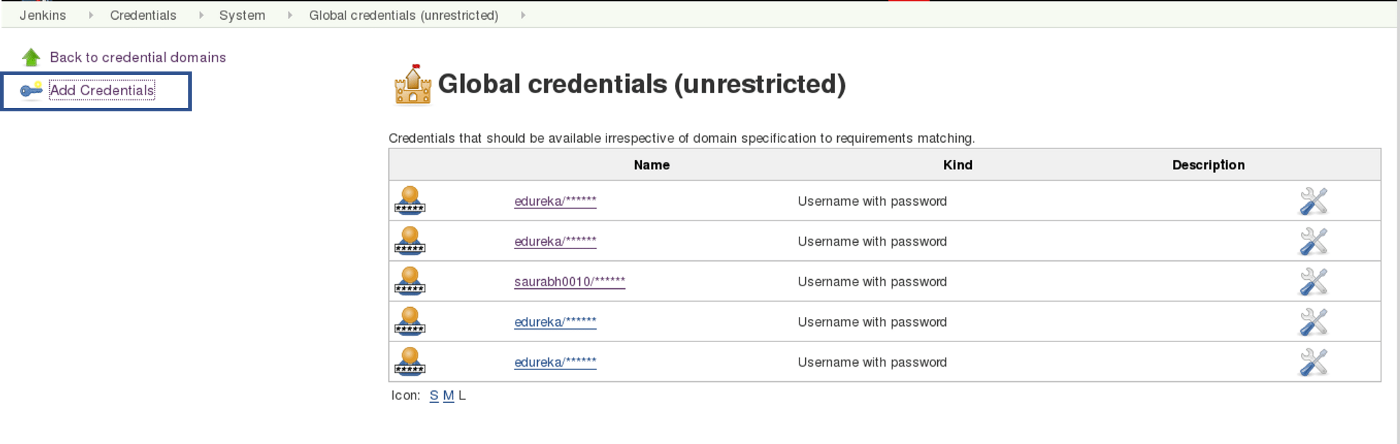
In order to add credentials in Jenkins, click on credentials option on the Jenkins dashboard.



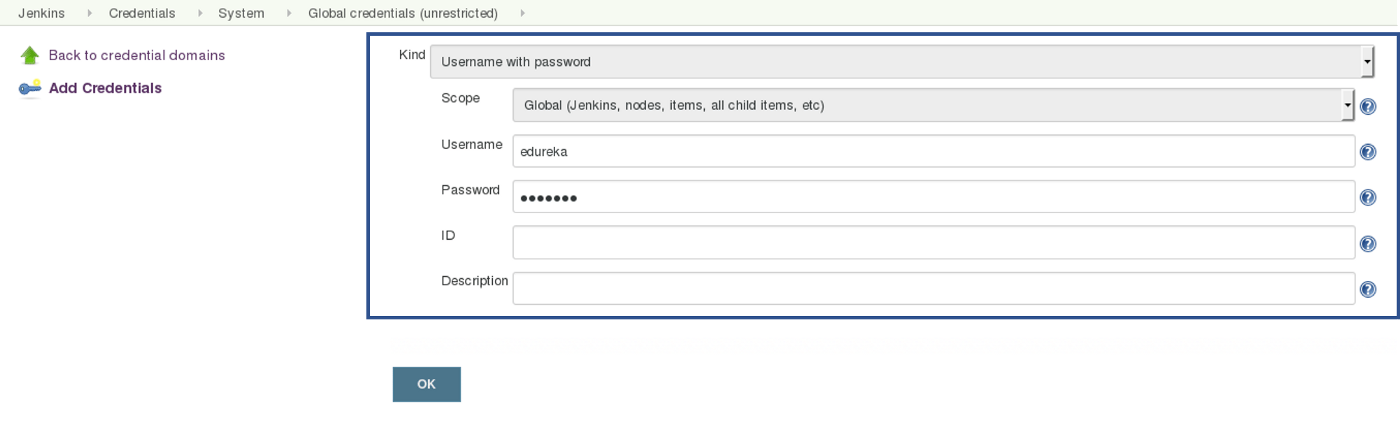
Click on System and select global credentials.



Then you will find an option to add the credentials. Click on it and add credentials.

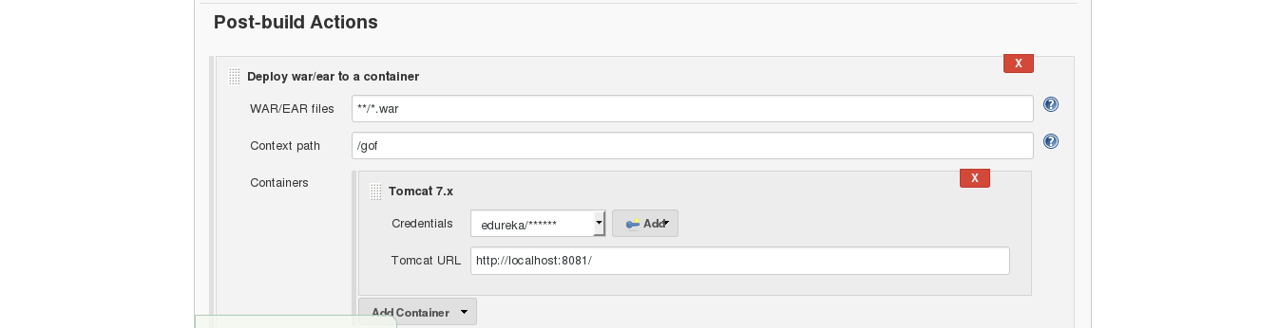


Add the Tomcat credentials, consider the below screenshot.

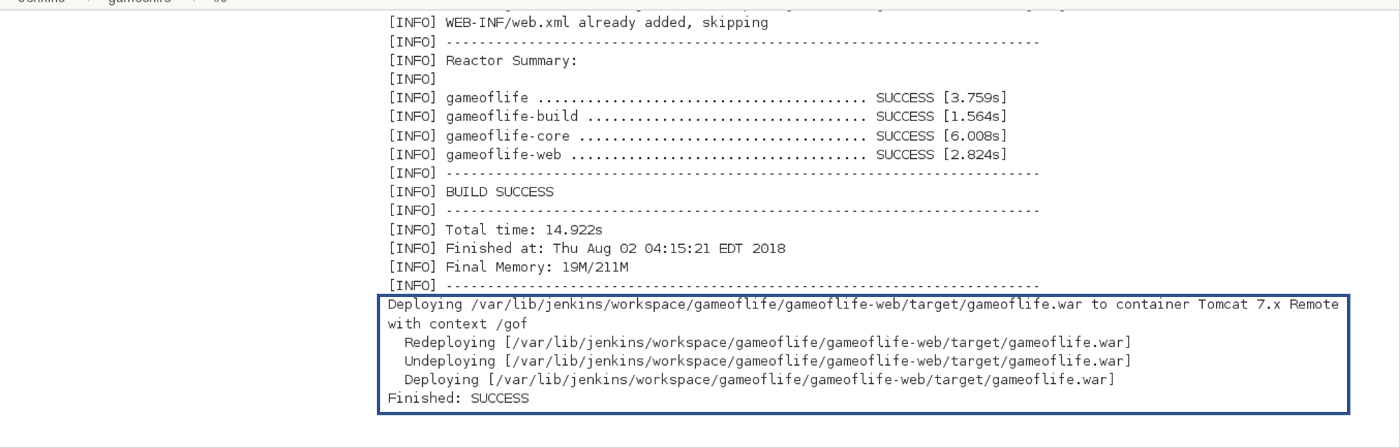


Click on OK.

Now in your Project Configuration, add the tomcat credentials which you have inserted in the previous step.



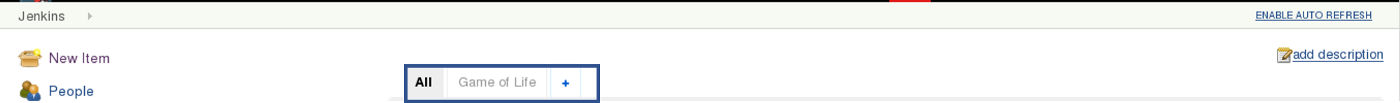
Click on Save and then select Build Now.



Go to your tomcat URL, with the context path, in my case it is [http://localhost:8081.](http://localhost:8081./" \t "https://medium.com/edureka/_blank) Now add the context path in the end, consider the below Screenshot:

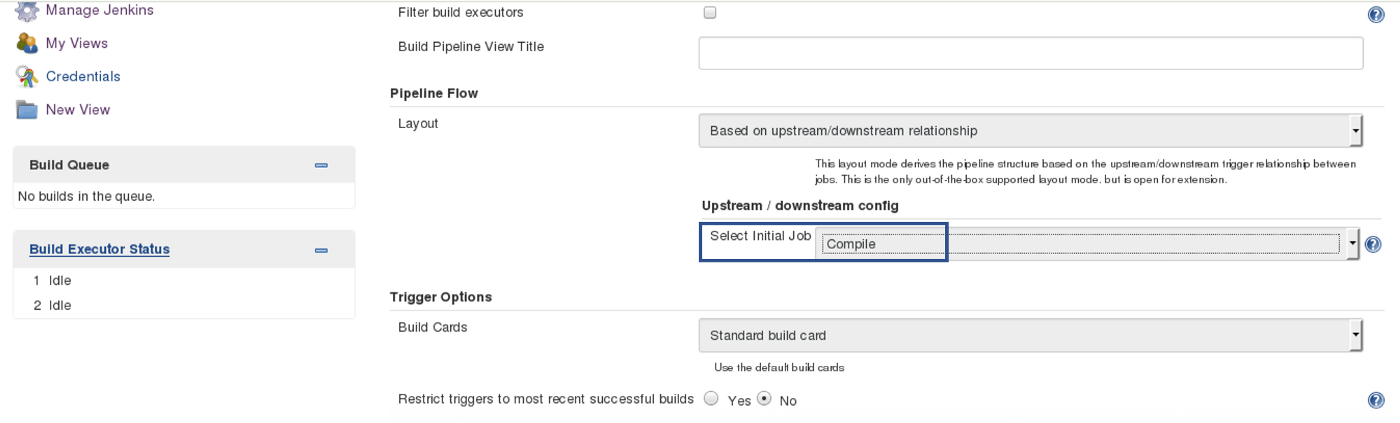
Link - [http://localhost:8081/gof](http://localhost:8081/gof" \t "https://medium.com/edureka/_blank)

Now create a pipeline view, consider the below screenshot:



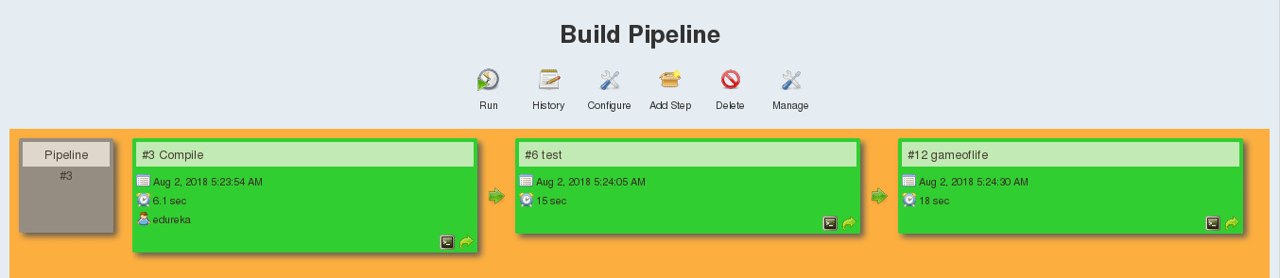
Click on the plus icon, to create a new view.

Configure the pipeline the way you want, consider the below screenshot:



Selecting the initial job. So pipeline will start from compile. Based on the way other jobs have been configured, after compile testing and deployment will happen.

Finally, you can test the pipeline by clicking on RUN. After every five minutes, if there is a change in the source code, the entire pipeline will be executed.



So we are able to continuously deploy our application on the test server for user acceptance test (UAT).