**Day 2 Assignment: Git & Jenkins**

*Assignment on Git Merge Conflict*

#Creating new directory to demonstrate merge conflict

**$ mkdir git-demo**

**$cd git-demo**

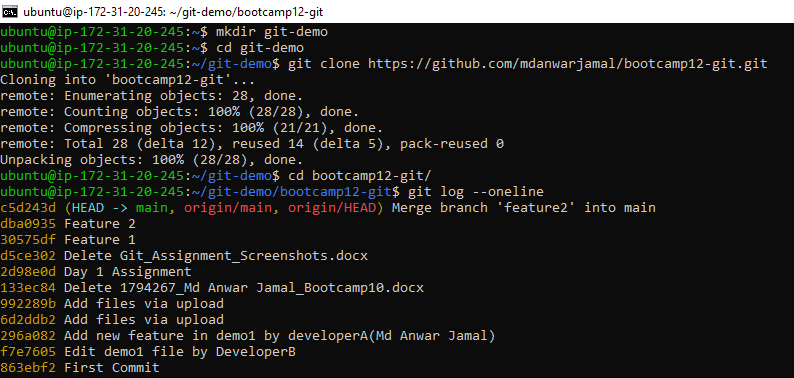
#Cloning online github repo

**$git clone** [**https://github.com/mdanwarjamal/bootcamp12-git.git**](https://github.com/mdanwarjamal/bootcamp12-git.git)

**$cd bootcamp12-git**

#Display commit history –oneline (to display oneline for each commit (id,msg))

**$ git log --oneline**



#Check all local branches available

**$ git branch**

#Creating two new branches feature1 and feature2

**$ git branch feature1**

**$ git branch feature2**

**$ git branch**

# Switch to feature1 branch

**$ git checkout feature1**

# Making changes in demo2 file

**$ vi demo2**

# Show changes made

**$ git diff**

#Check status of files

**$ git status**

#Adding file to staging area

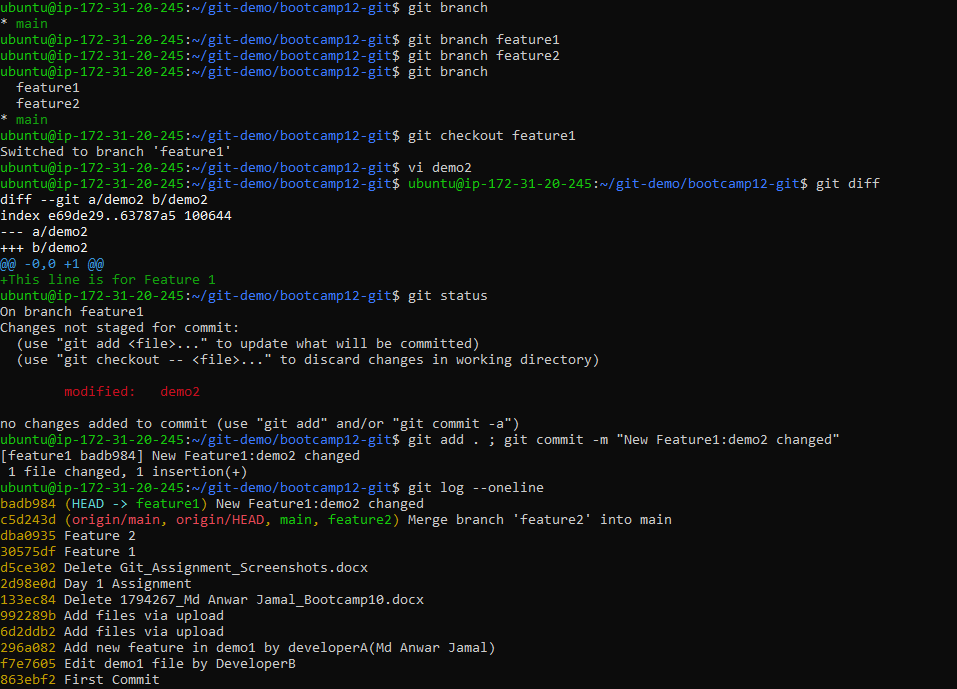
**$ git add .**

# Committing the changes to local repo (for feature1 branch)

**$ git add . ; git commit –m “New Feature1:demo2 changed”**

#Display commit history –oneline (to display oneline for each commit (id,msg))

**$ git log --oneline**



# Switch to feature2 branch

**$ git checkout feature2**

# Making changes in demo2 file

**$ vi demo2**

# Show changes made

**$ git diff**

#Check status of files

**$ git status**

#Adding file to staging area

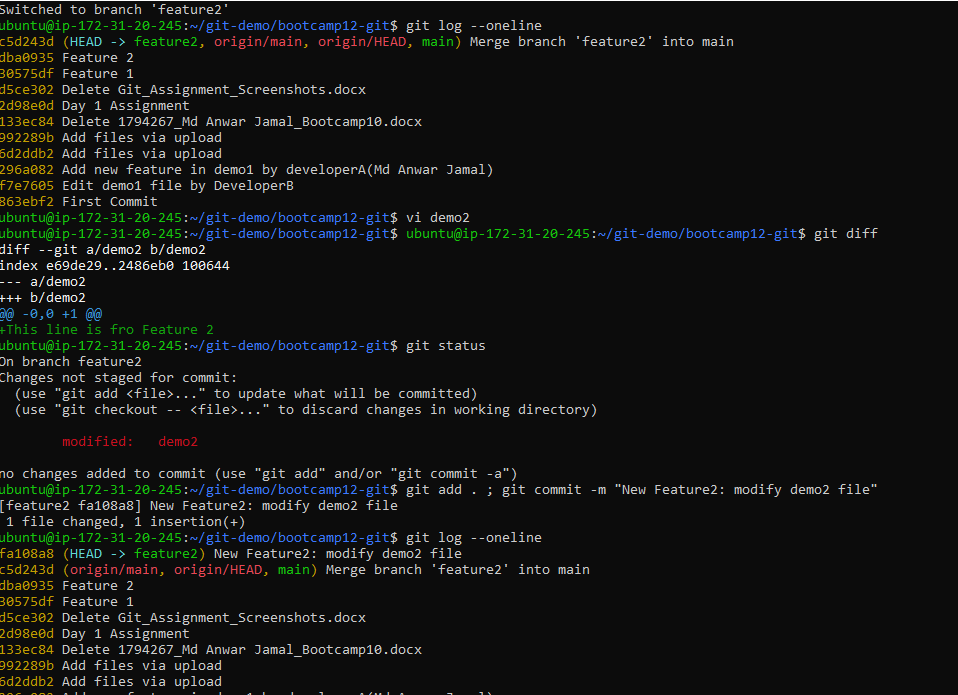
**$ git add .**

# Committing the changes to local repo (for feature2 branch)

**$ git add . ; git commit –m “New Feature2: modify demo2 file”**

#Display commit history –oneline (to display oneline for each commit (id,msg))

**$ git log --oneline**



# Switch to main branch

**$ git checkout main**

# Set vimdiff as mergetool in git

**$ git config --global merge.tool vimdiff**

**$ git config --global merge.conflictstyle diff3**

**$ git config --global mergetool.prompt false**

# Merge changes from feature1 branch into main branch

**$ git merge feature1**

#Display commit history –oneline (to display oneline for each commit (id,msg))

**$ git log –oneline**

# Merge changes from feature2 branch into main branch

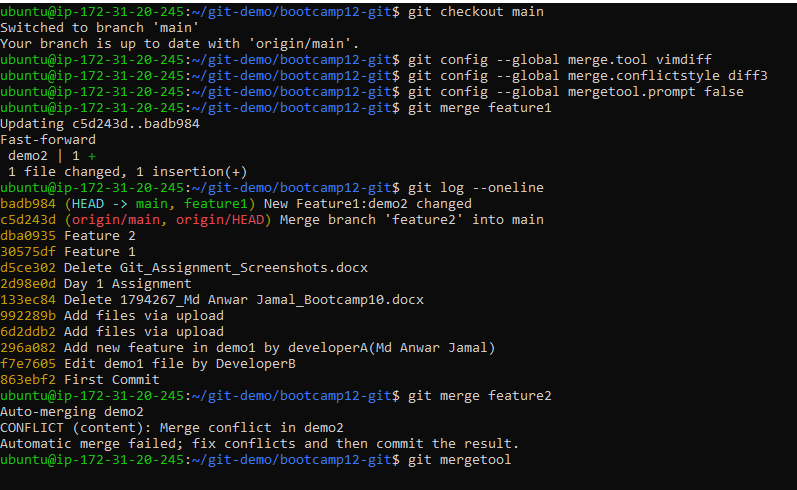
**$ git merge feature1**

We will get merge conflict because for same file for same line git detect two diverging changes one from feature1 and feature2

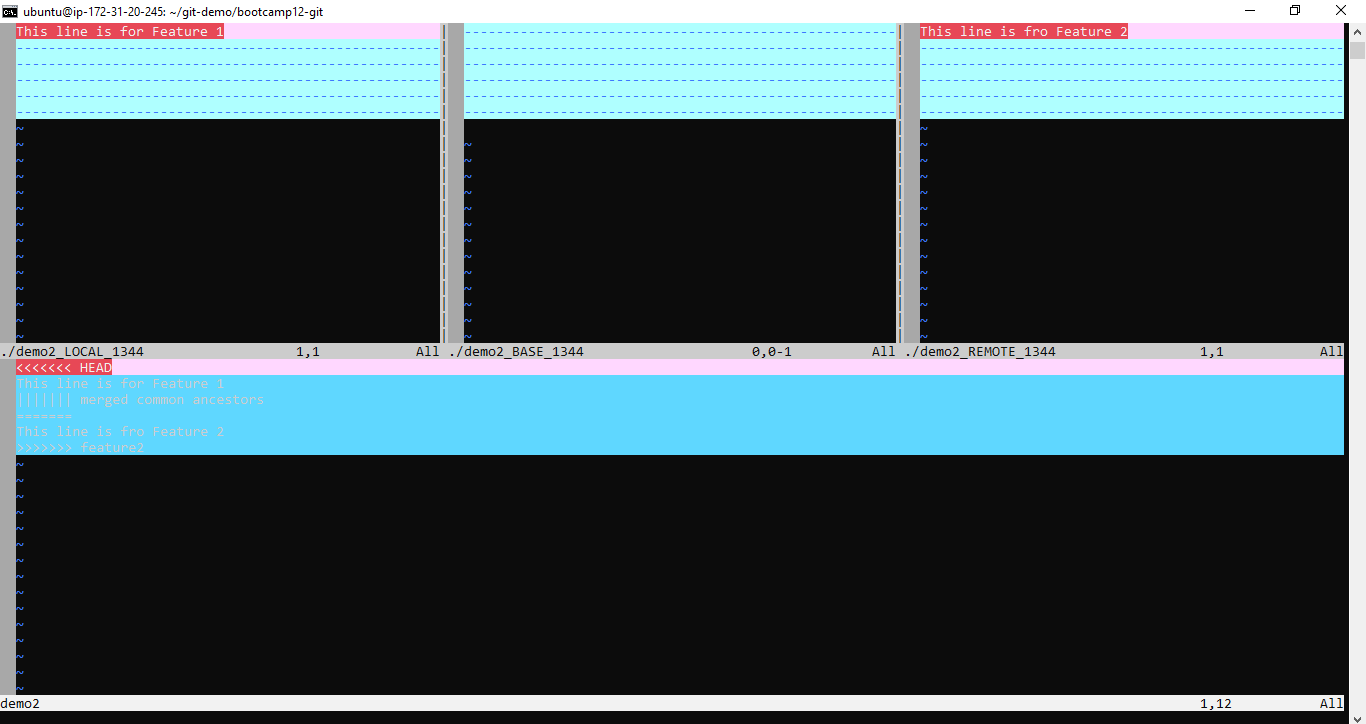
To resolve this merge conflict we will make use of mergetool (vimdiff)

# To launch mergetool to resolve conflict

**$ git mergetool**



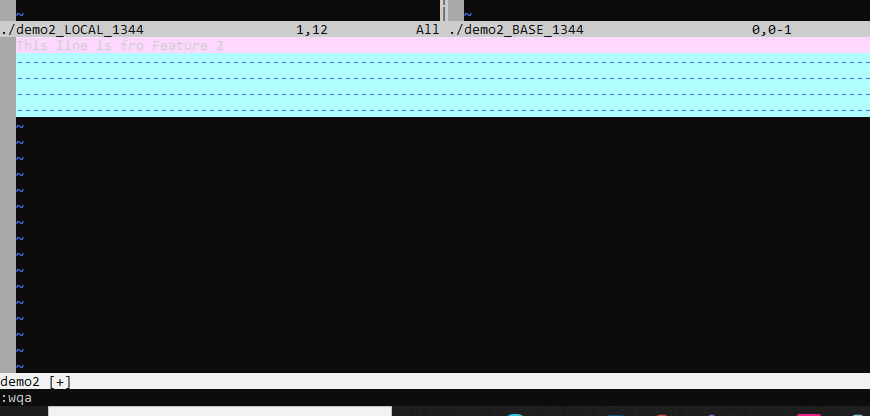
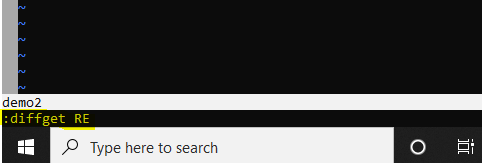
Vimdiff tool where all varying version of files are displayed to select from



To select changes in REMOTE, we will use

**<ESC> :diffget RE**

**<ESC> :wqa**



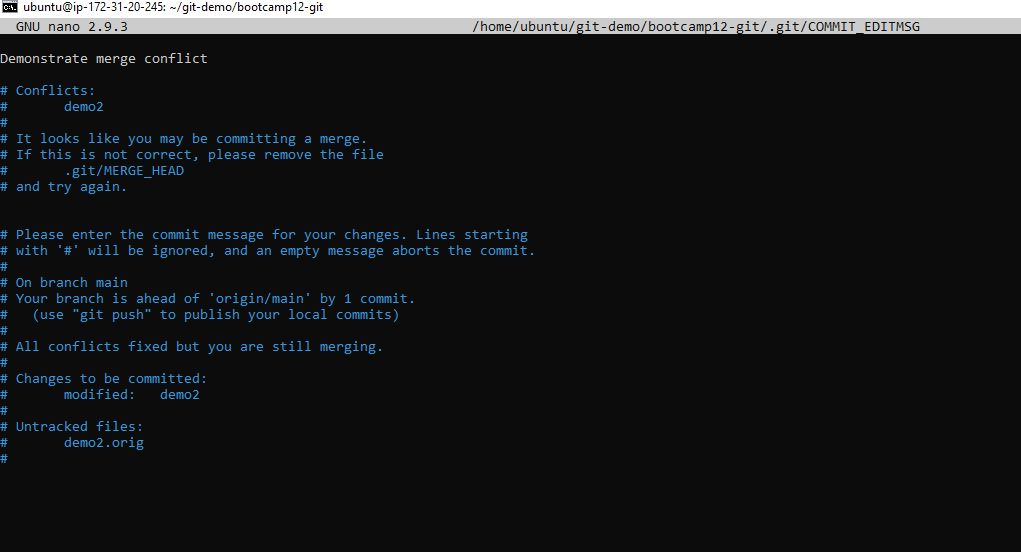
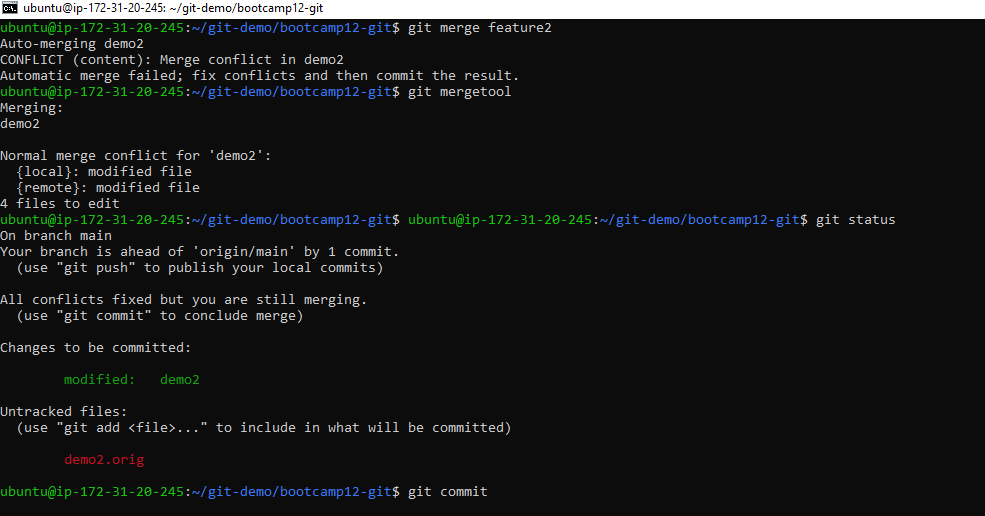
# Notice resolved changes are now staged

**$ git status**

# To make merge commit

**$ git commit**

A popup window will come up where we need to give commit message for Merge Commit



# See merge commit is now showing up in commit history

**$ git log --oneline**

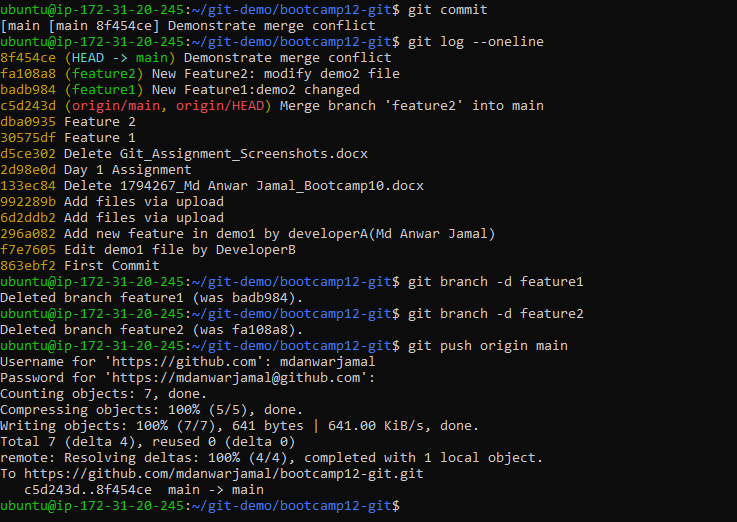
# Deleting both merged branches

**$ git branch –d feature1**

**$ git branch –d feature2**

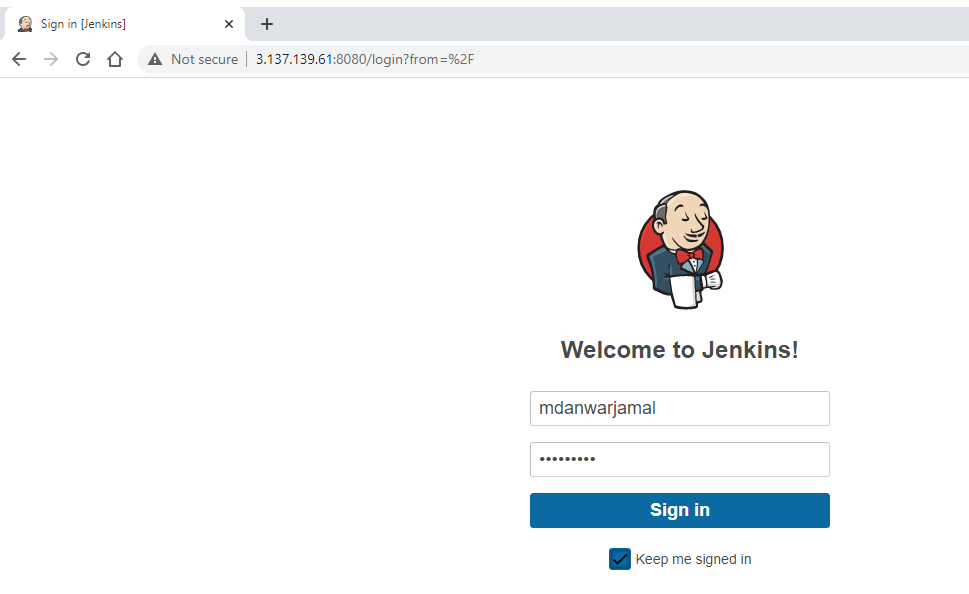
# Pushing changes to Remote Repo

**$ git push origin main**

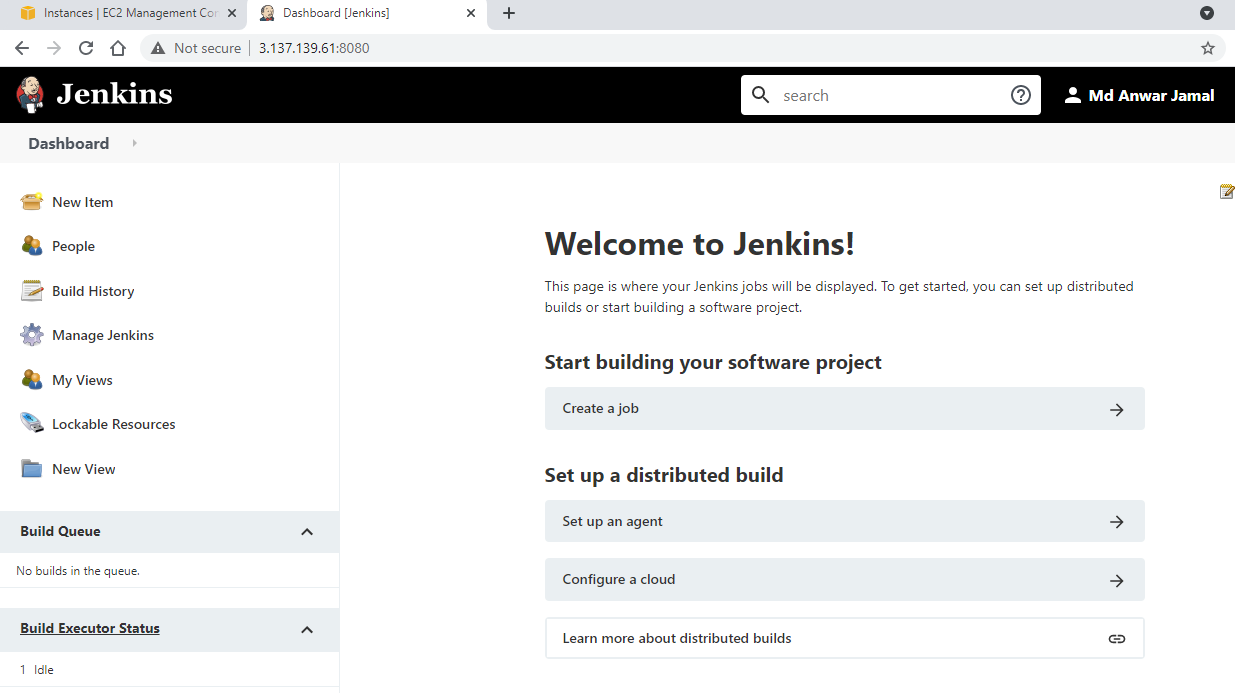


*Jenkins Setup*

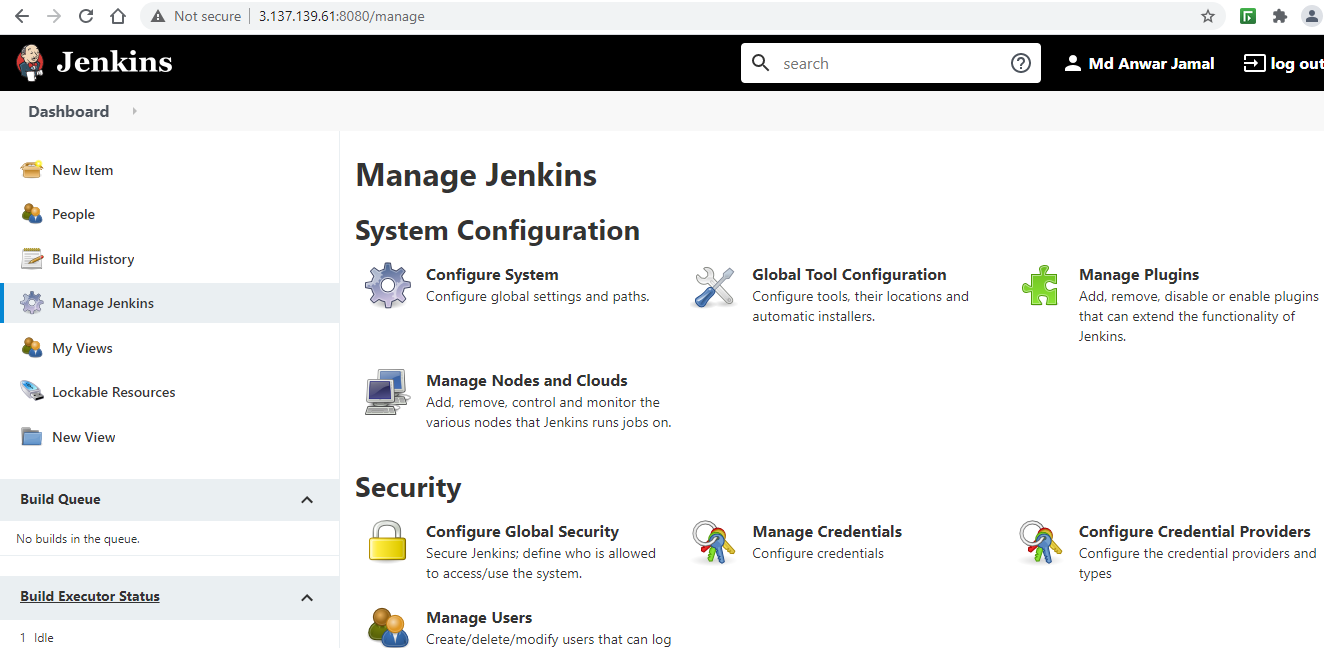
**Login as Admin**

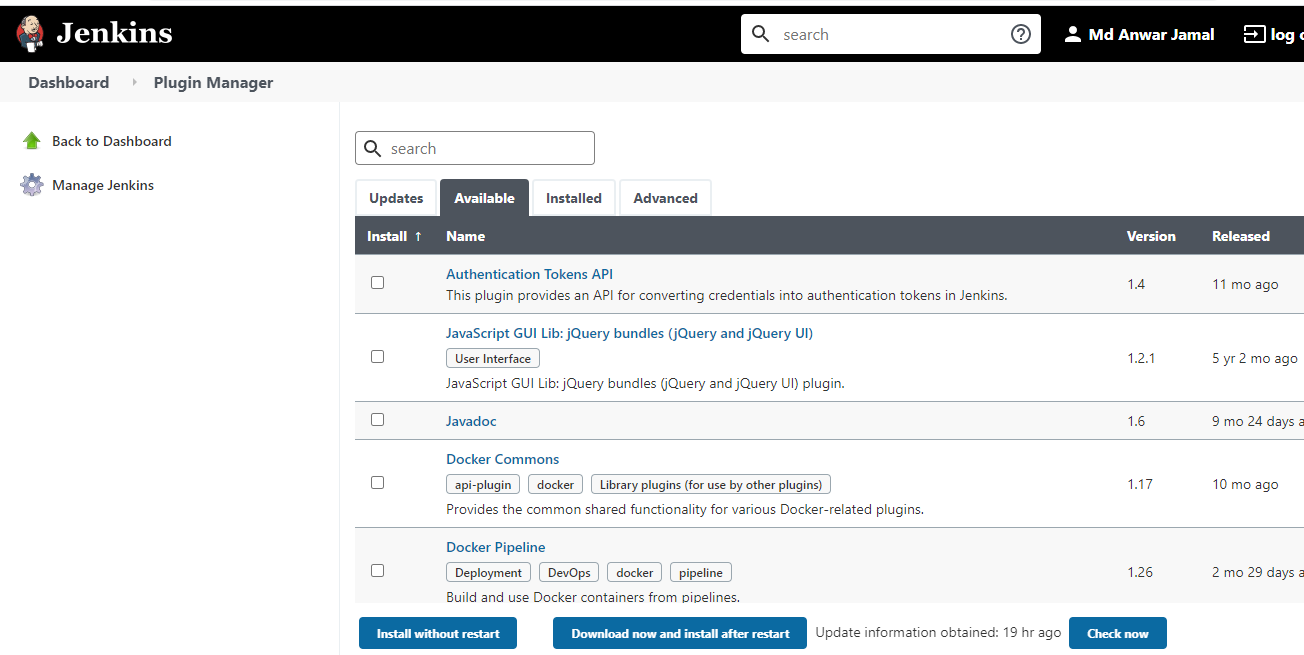


**Jenkins Dashboard**

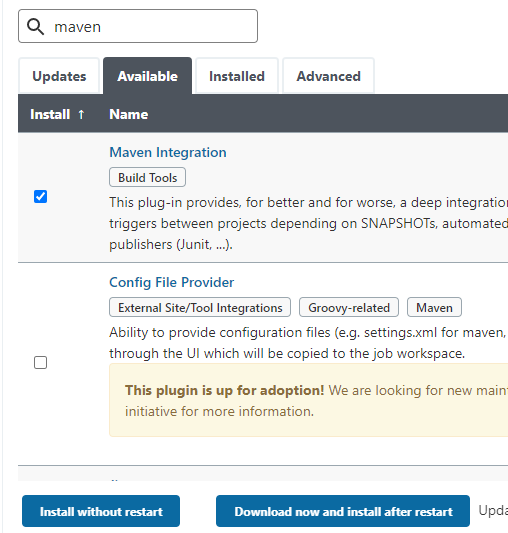


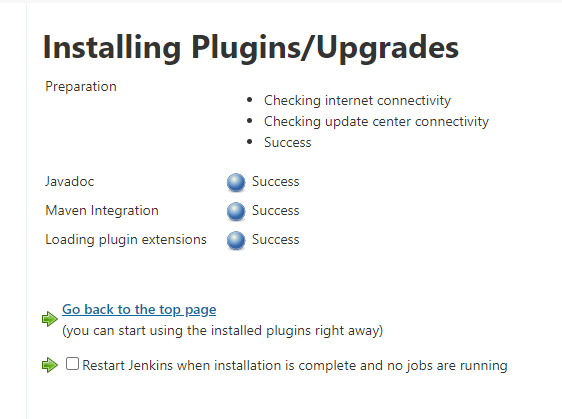
**Manage Jenkins-> Manage Plugins**



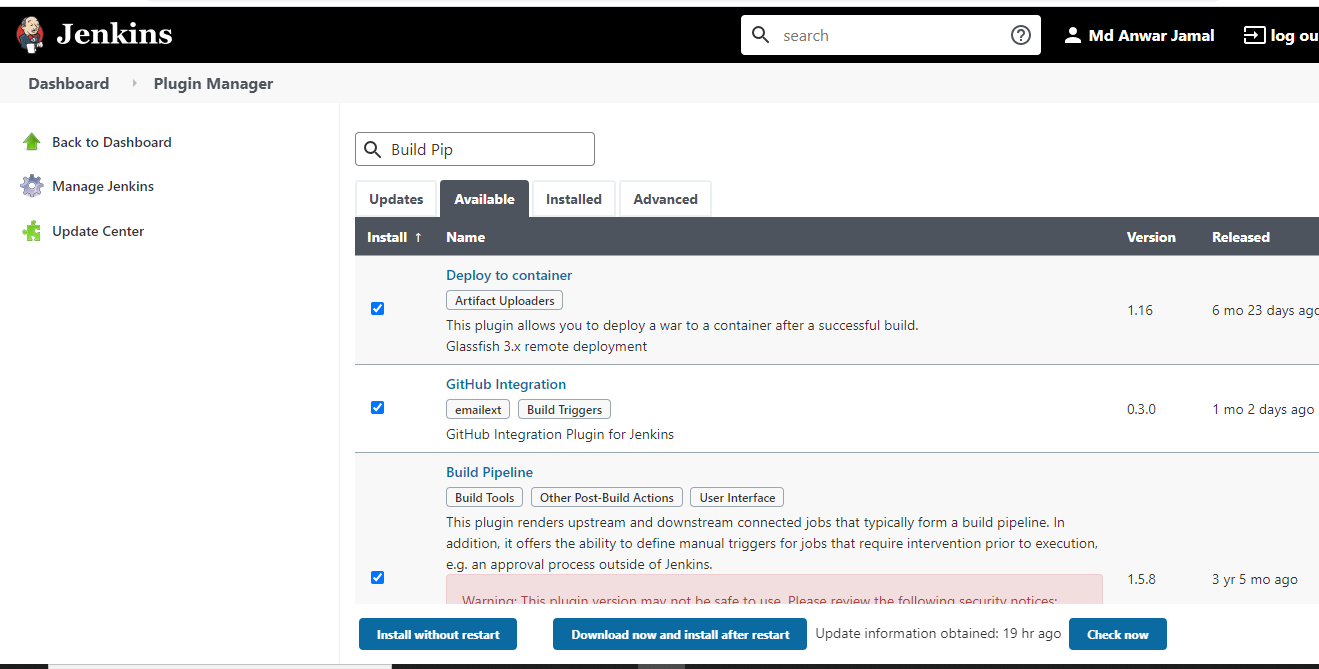


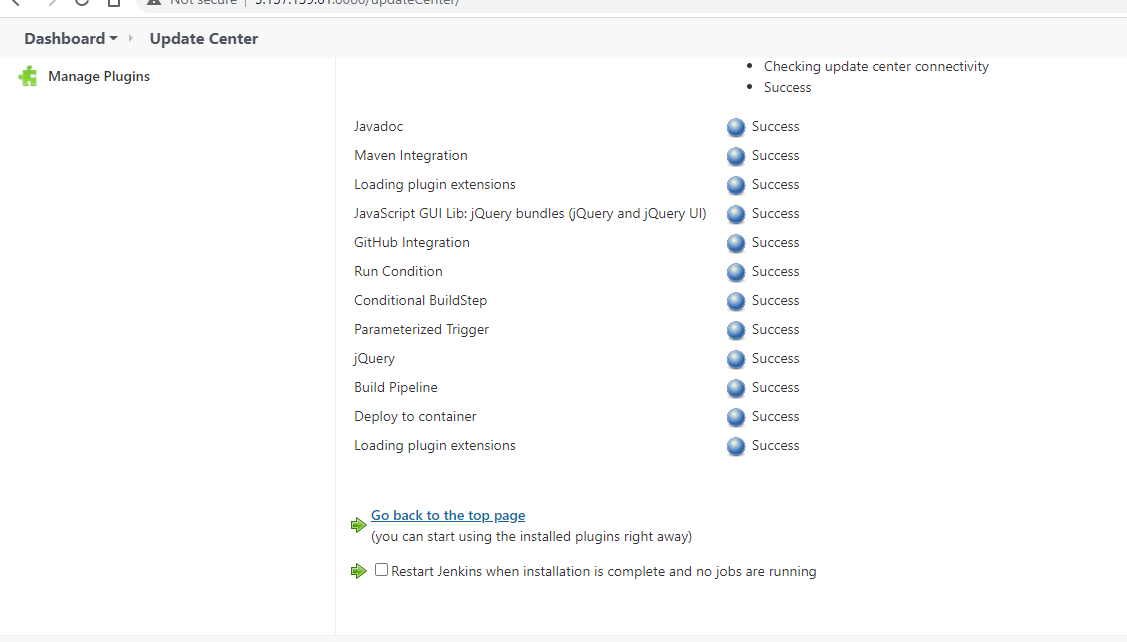
**Downloading *Maven Integration* plugin**



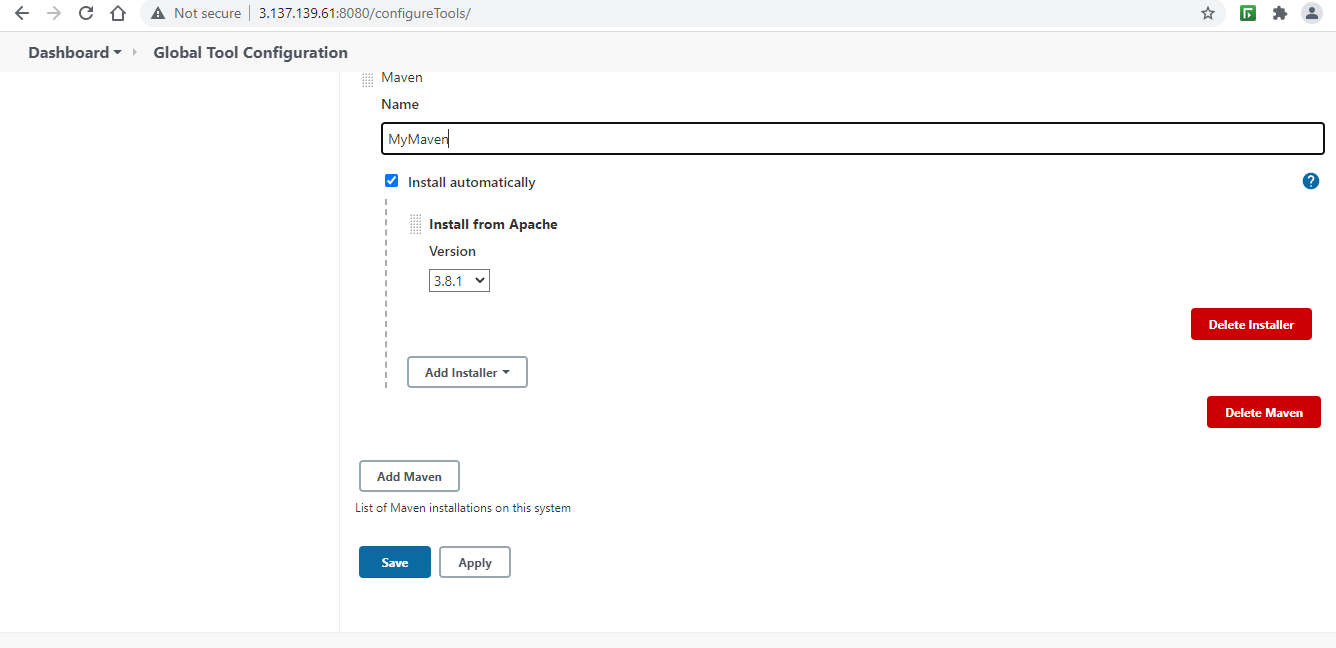


**Installing *Deploy to container, GitHub Integration and Build Pipeline* plugins**





**Installing Maven from Manage Jenkins > Manage Global Tool Configuration**

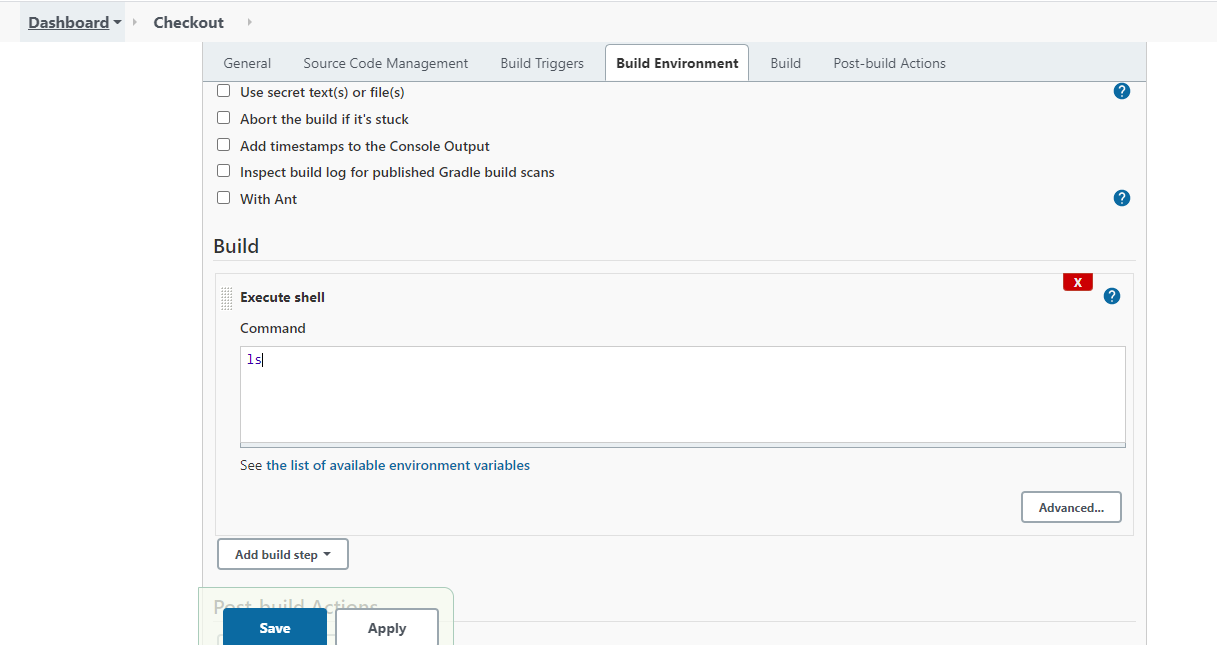
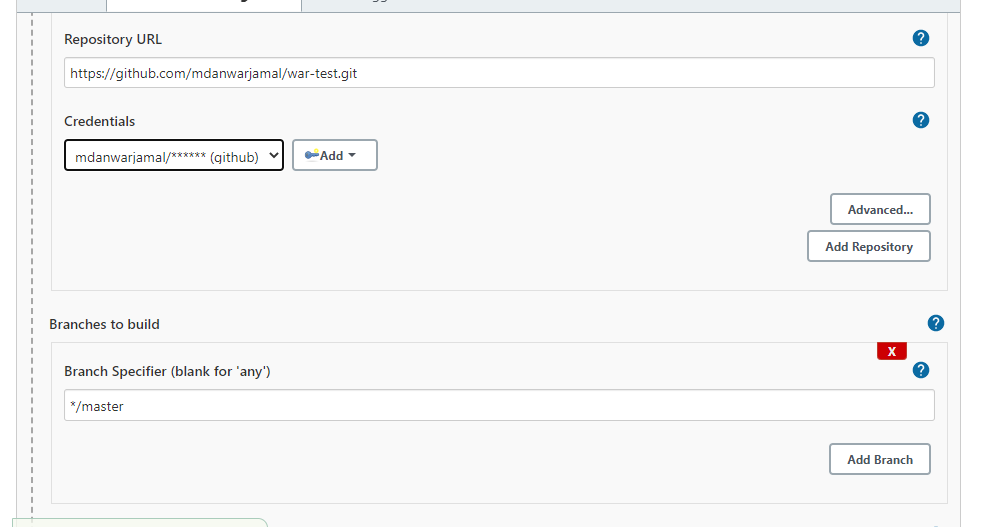
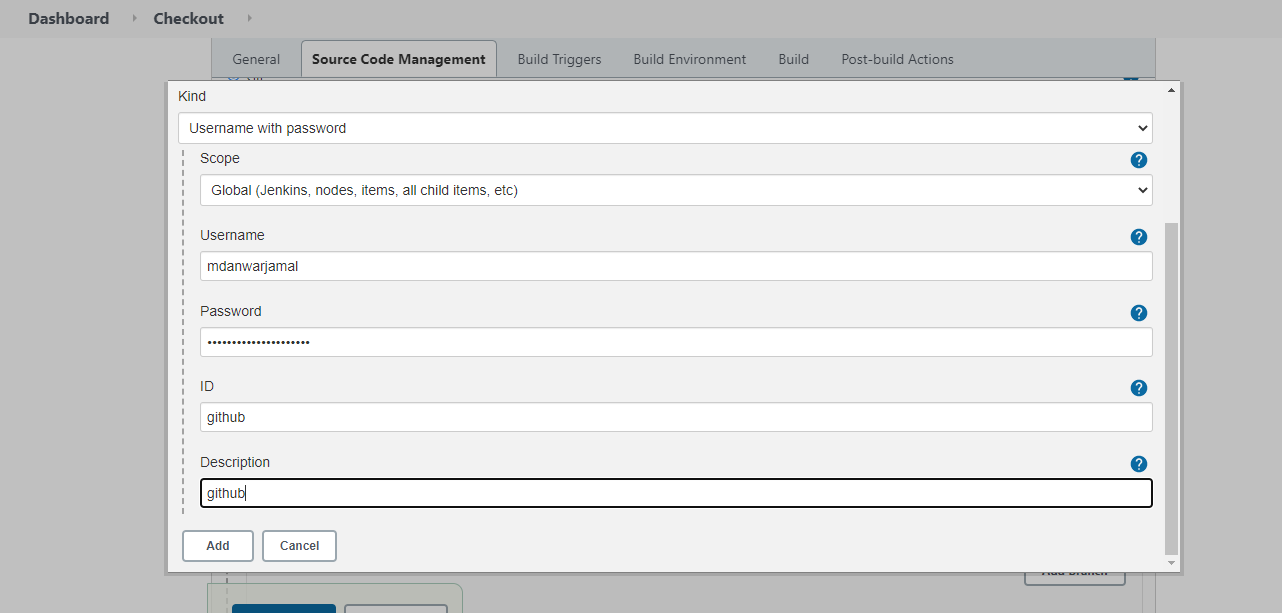
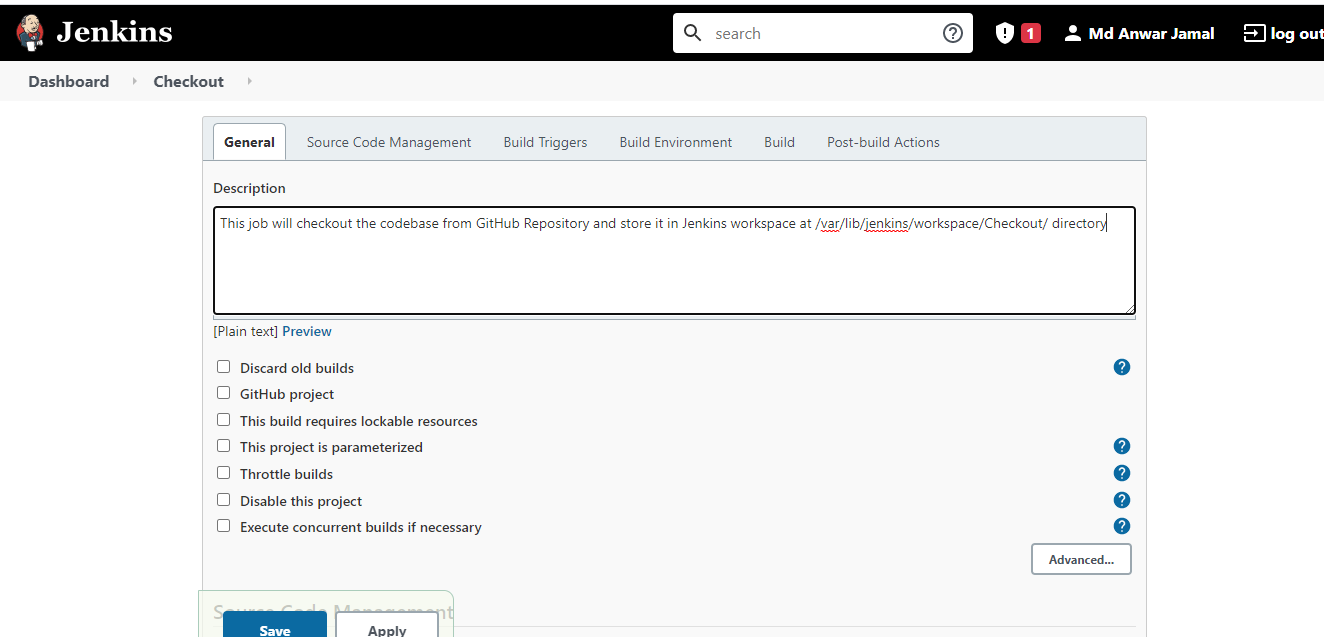
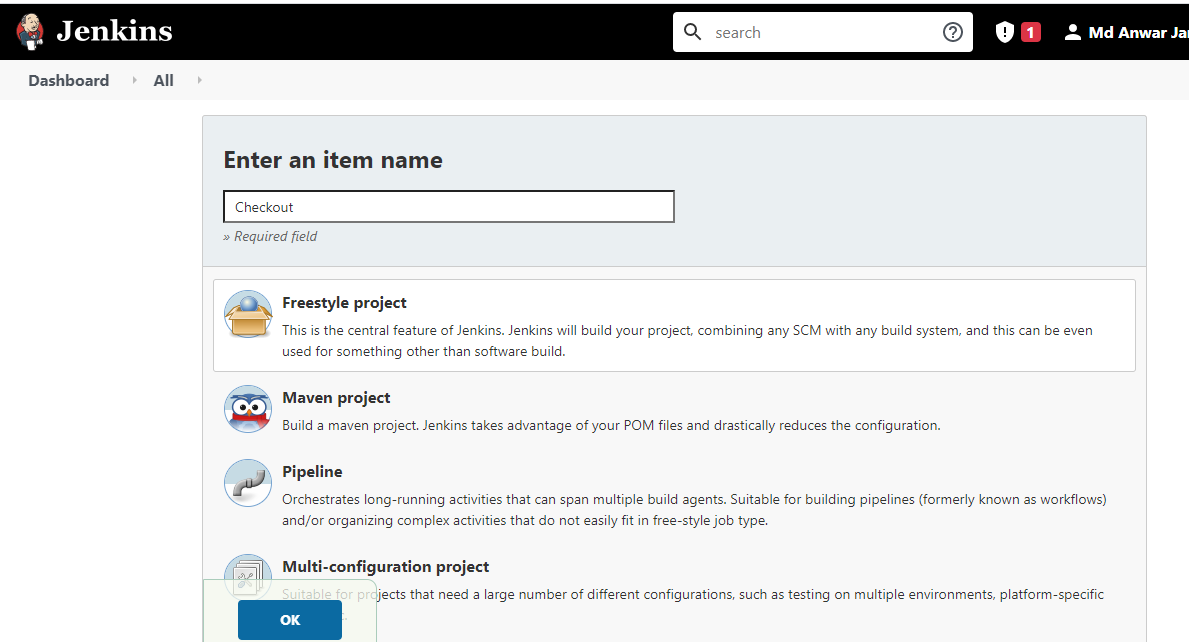


*Freestyle Jobs Pipeline*

*Checkout*

**This will checkout GitHub repository and keep it in Jenkins's workspace**

**Using Git as SCM**



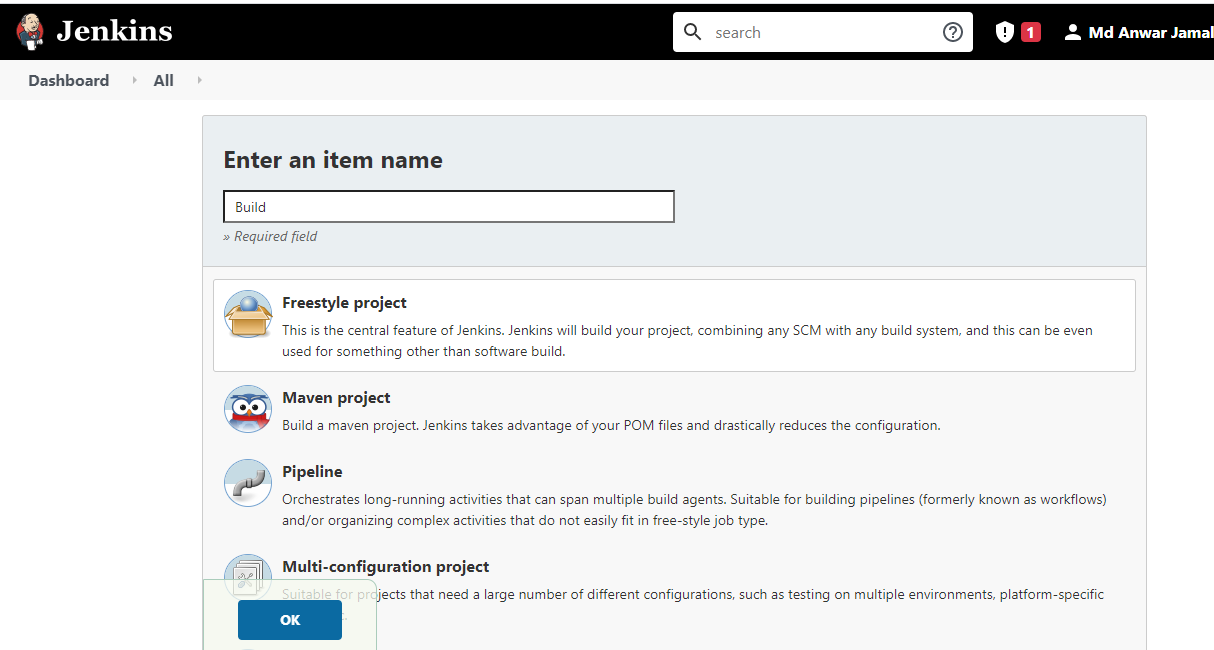
*Build*

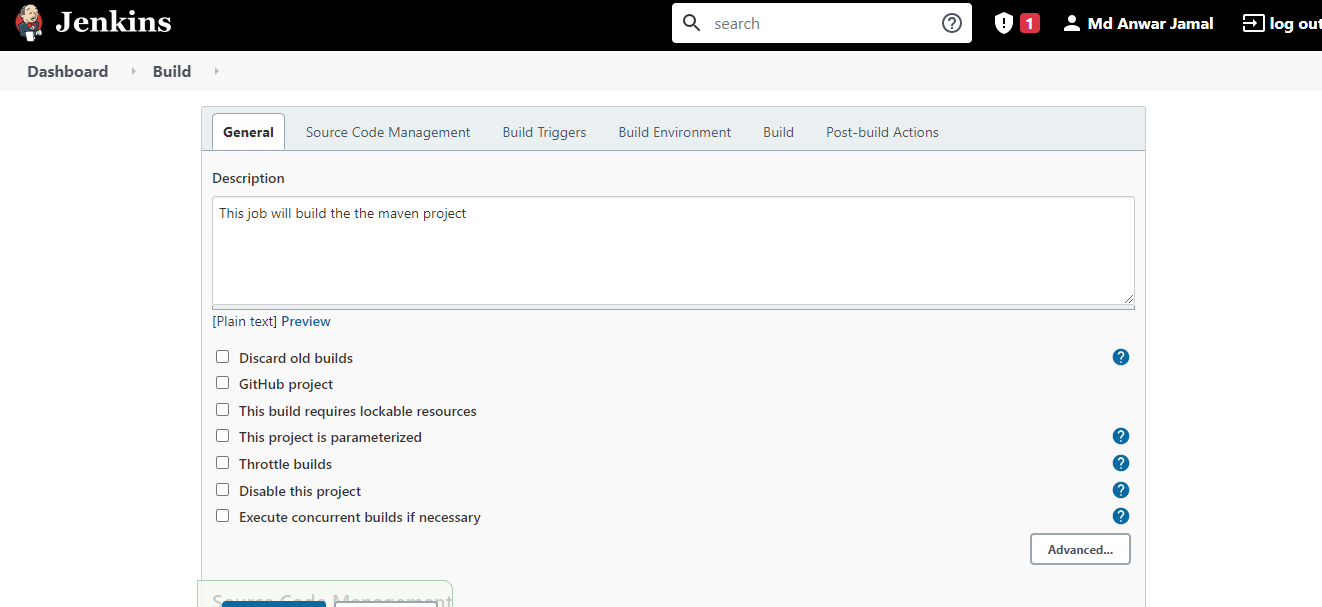
**This will compile the maven project downloaded via Checkout Job**

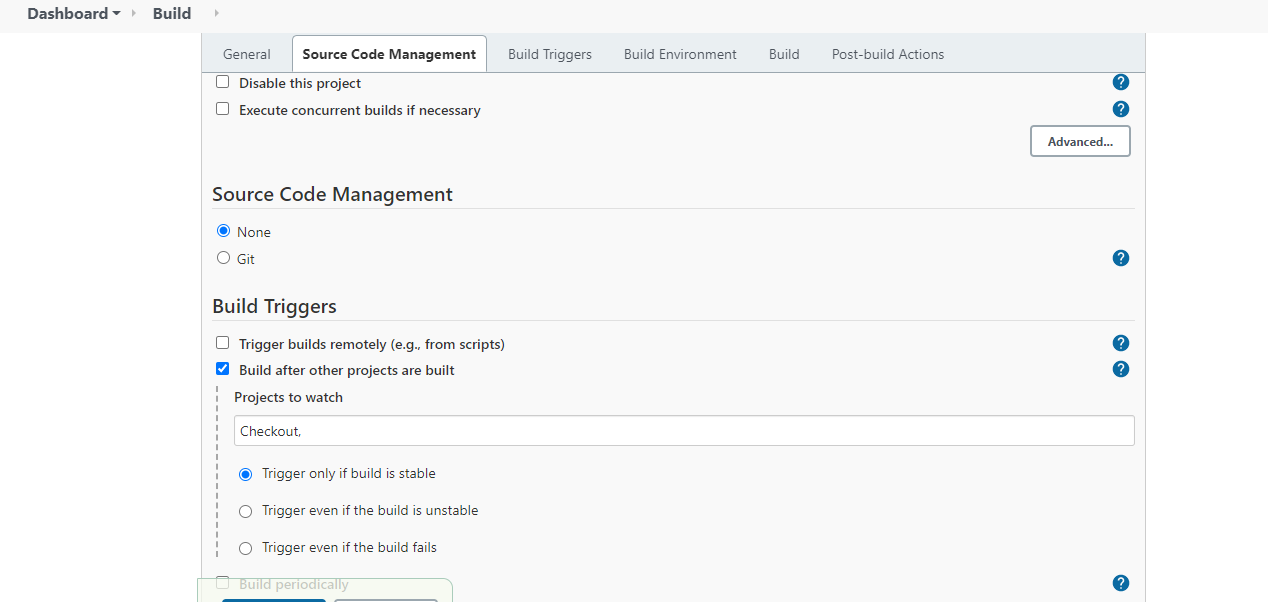
**Triggered after successful run of Checkout Job**

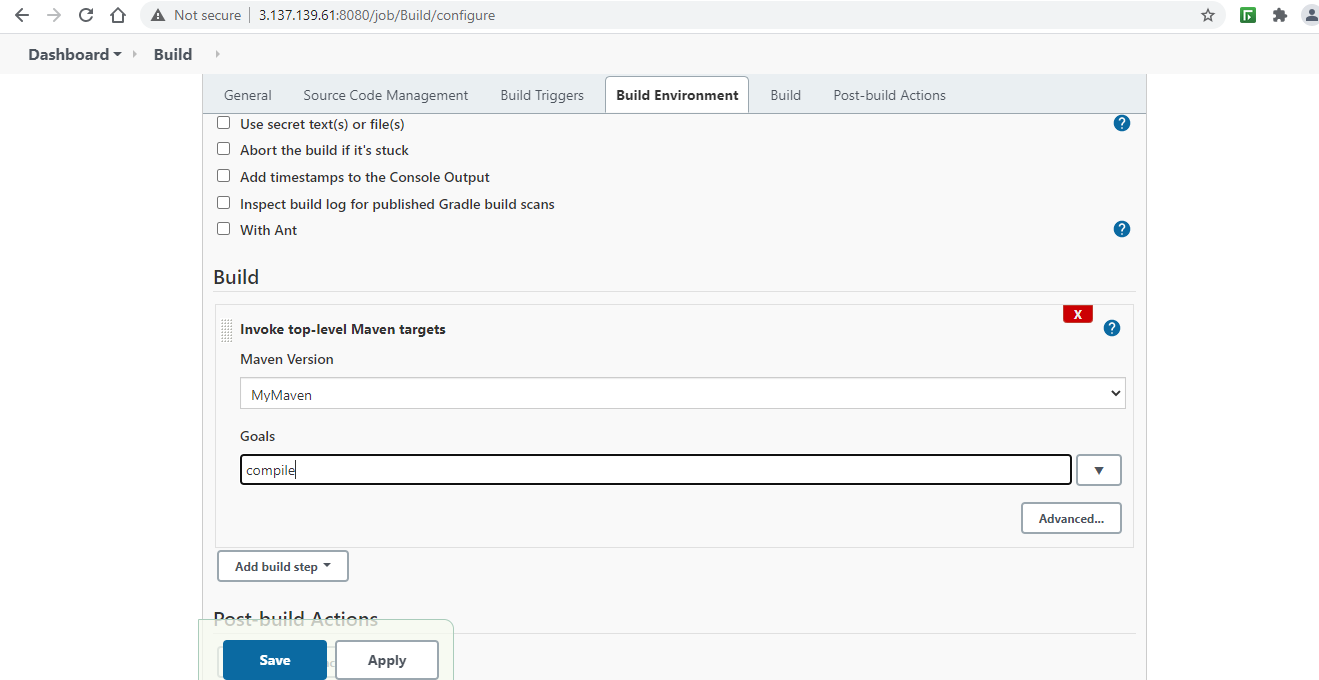
**Using Invoke top-level Maven Goals to execute (Selected our downloaded Maven from Global Tool)**

***$ mvn compile***









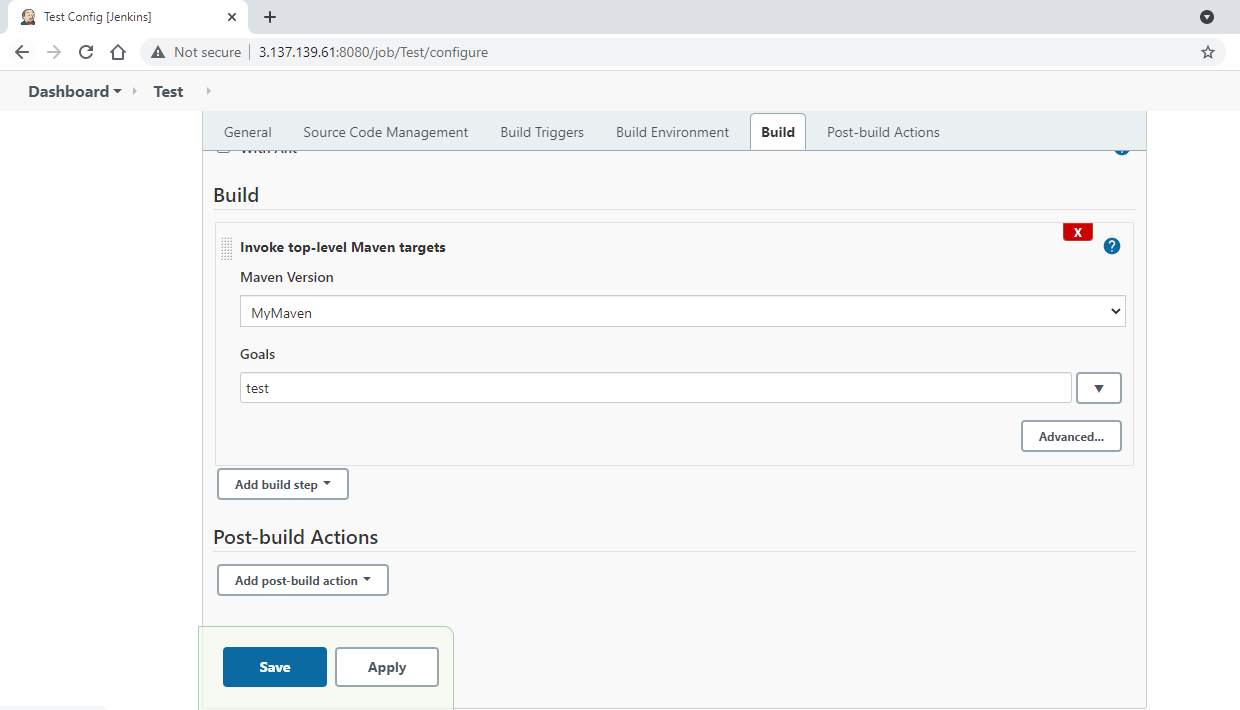
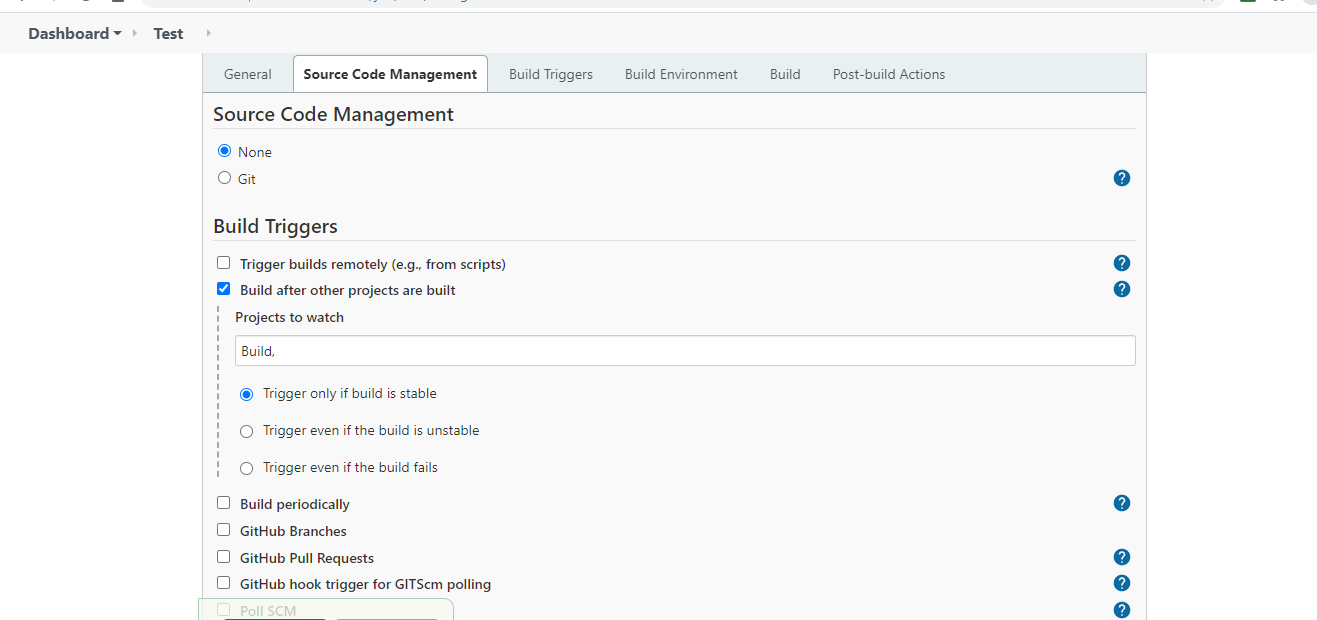
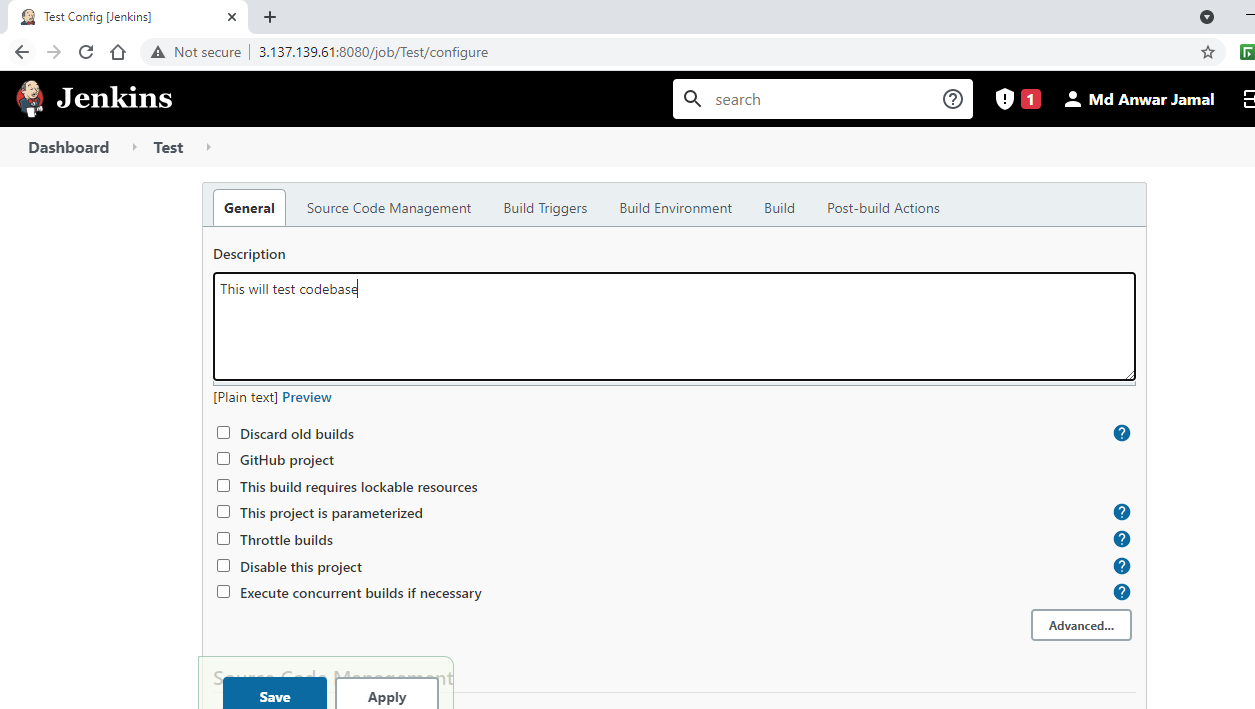
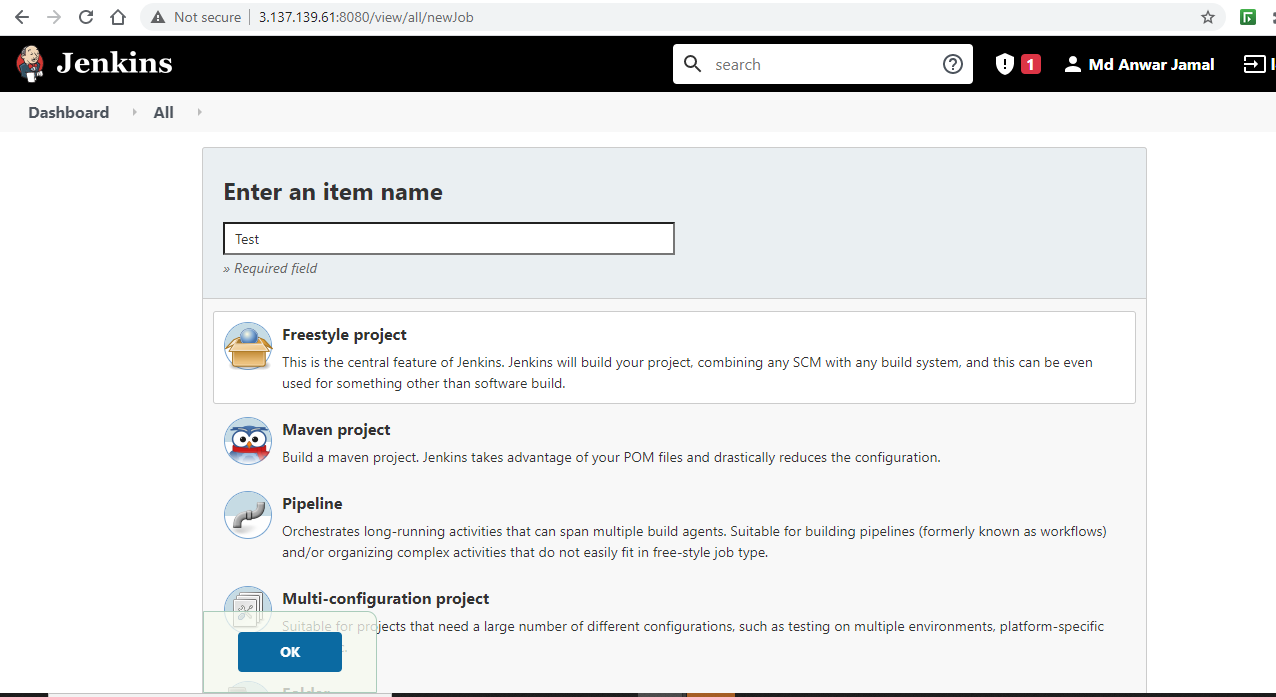
*Test*

**This will run test cases for the maven project**

**Triggered after successful run of Build Job**

**Using Invoke top-level Maven Goals to execute (Selected our downloaded Maven from Global Tool)**

***$ mvn test***



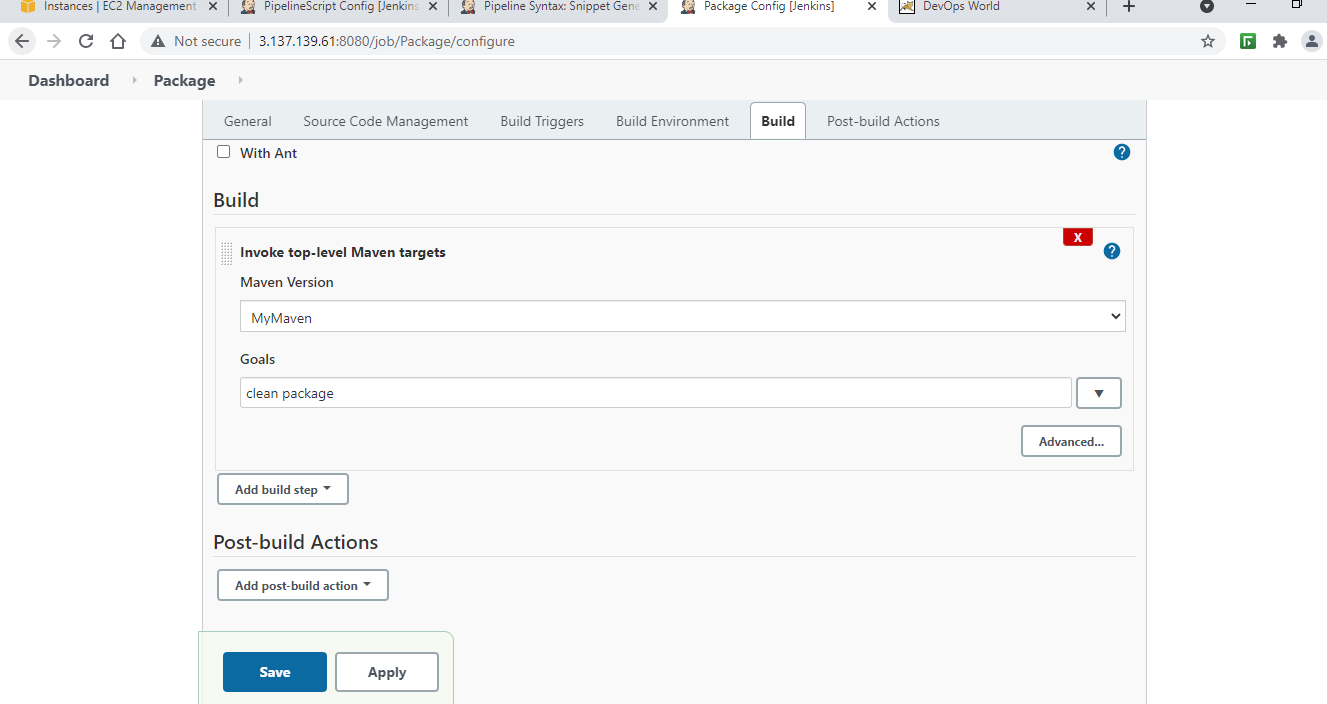
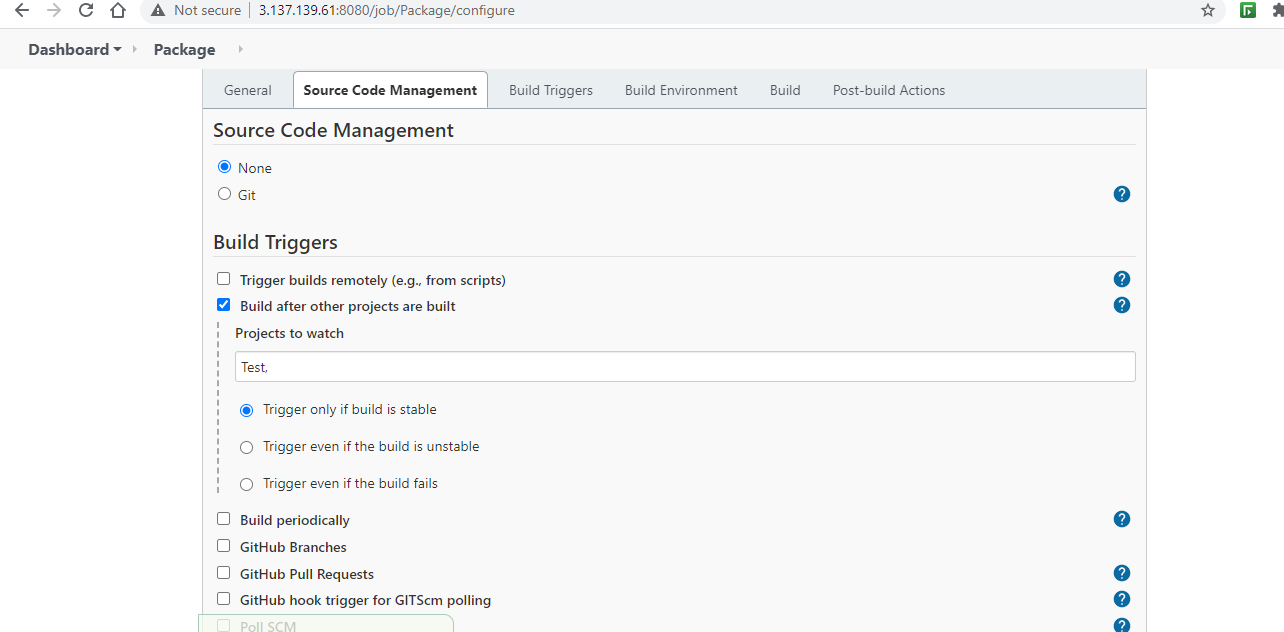
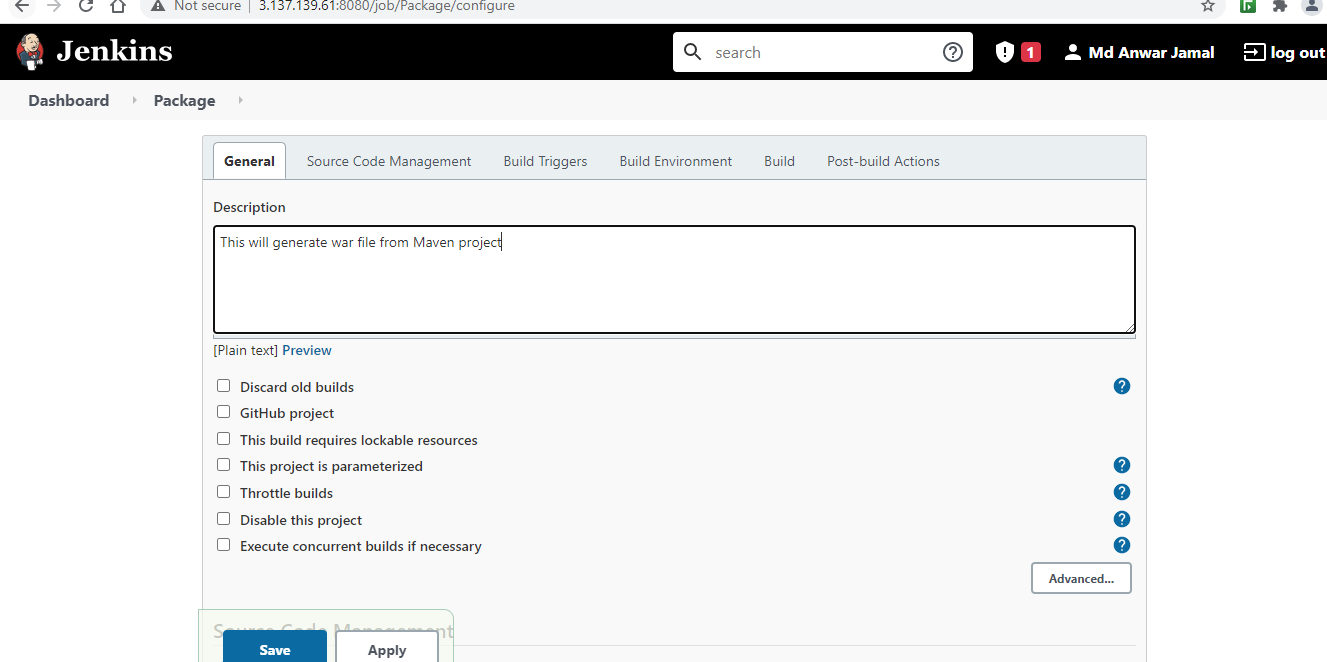
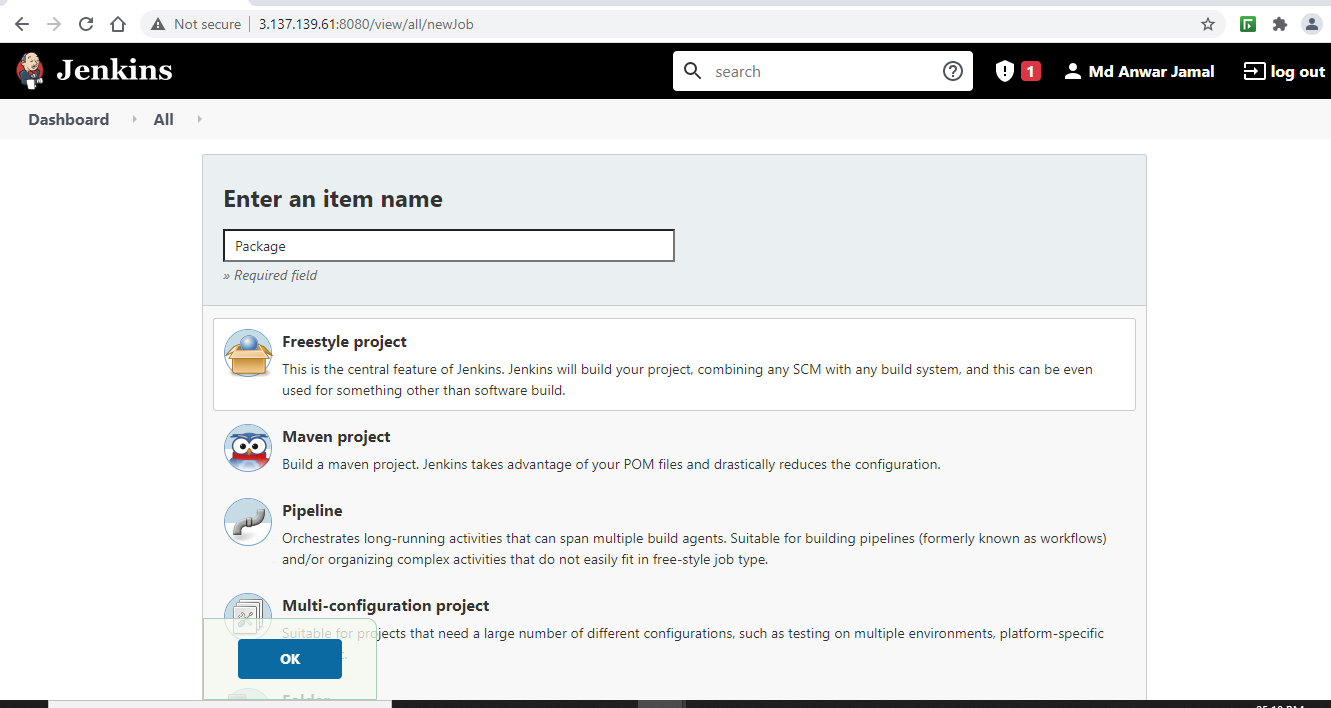
*Package*

**This will create a .war file for the maven project**

**Triggered after successful run of Test Job**

**Using Invoke top-level Maven Goals to execute (Selected our downloaded Maven from Global Tool)**

***$ mvn package***



*Deploy*

**This will deploy .war file to Tomcat Server**

**Triggered after successful run of Package Job**

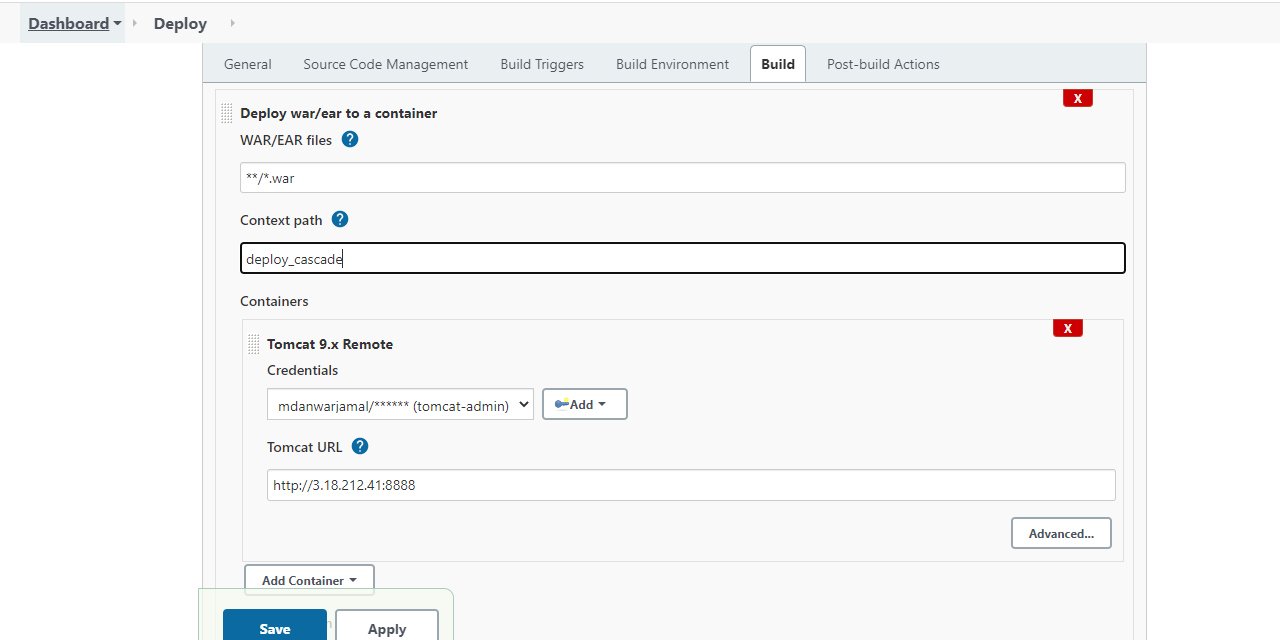
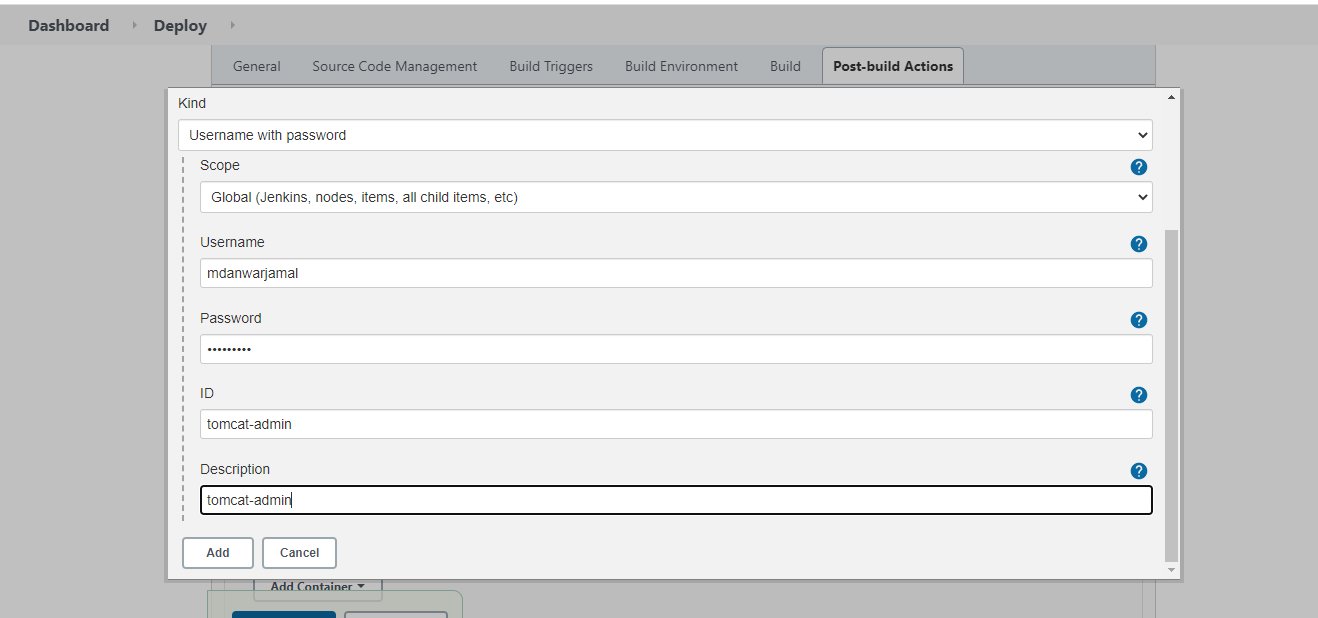
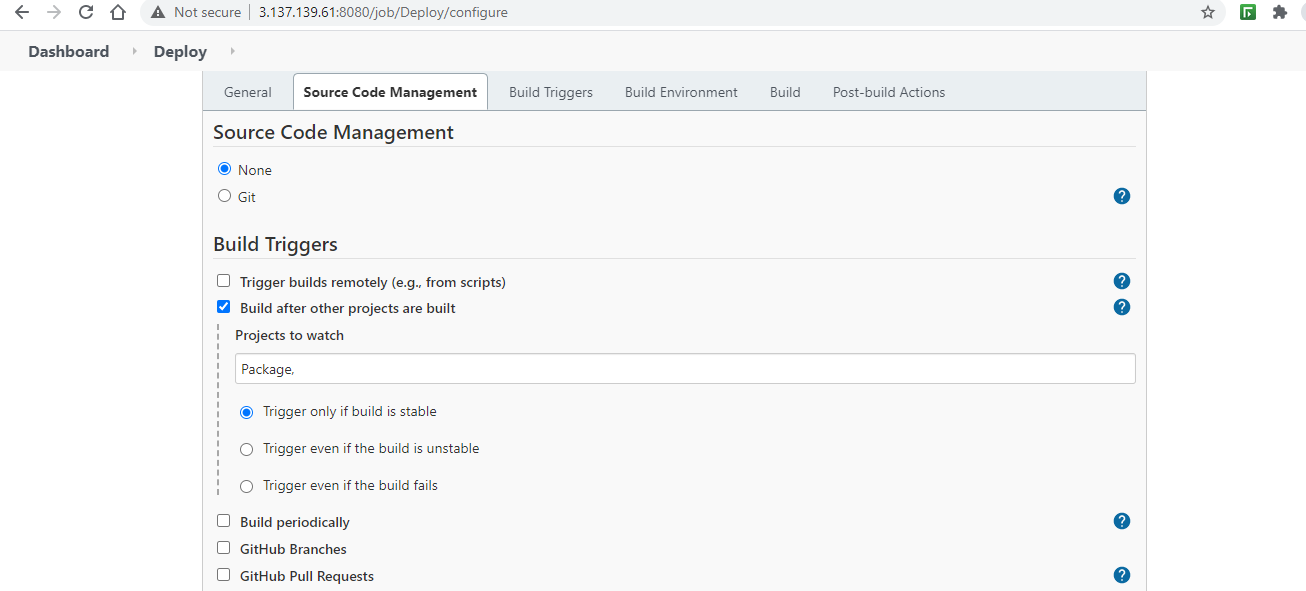
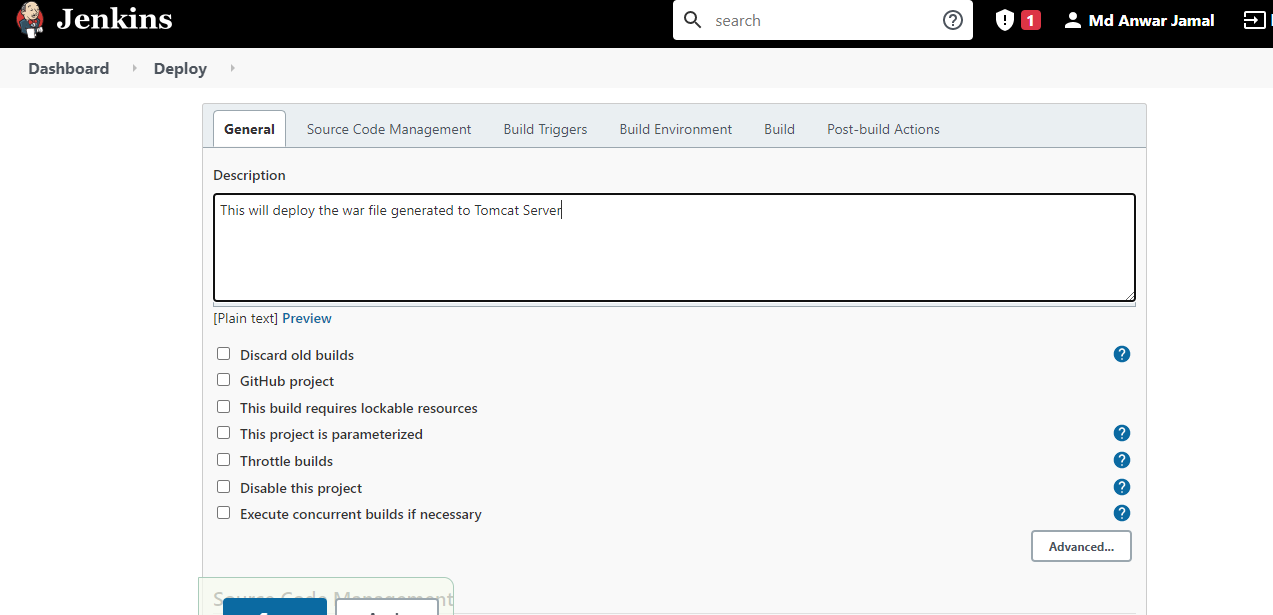
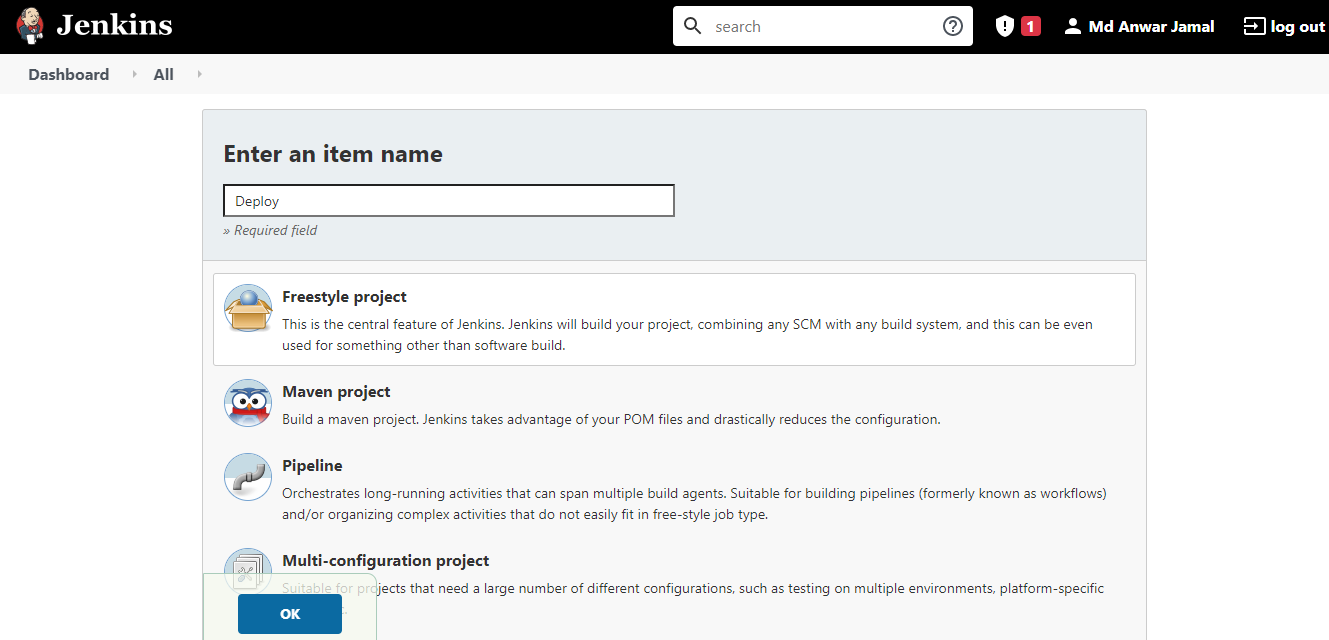
**Used *Deploy to Container plugin***

WAR/EAR File: Location of war file to put onto Tomcat Server

Context Path: the path on tomcat server from which it is accessible

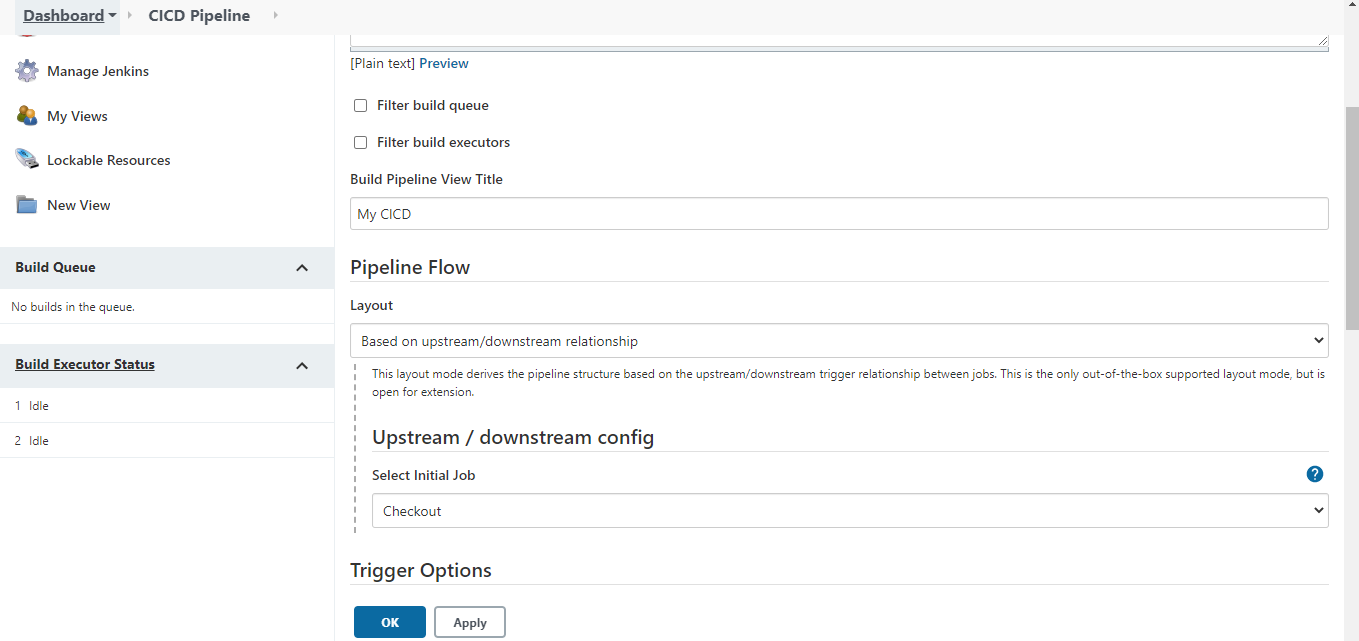
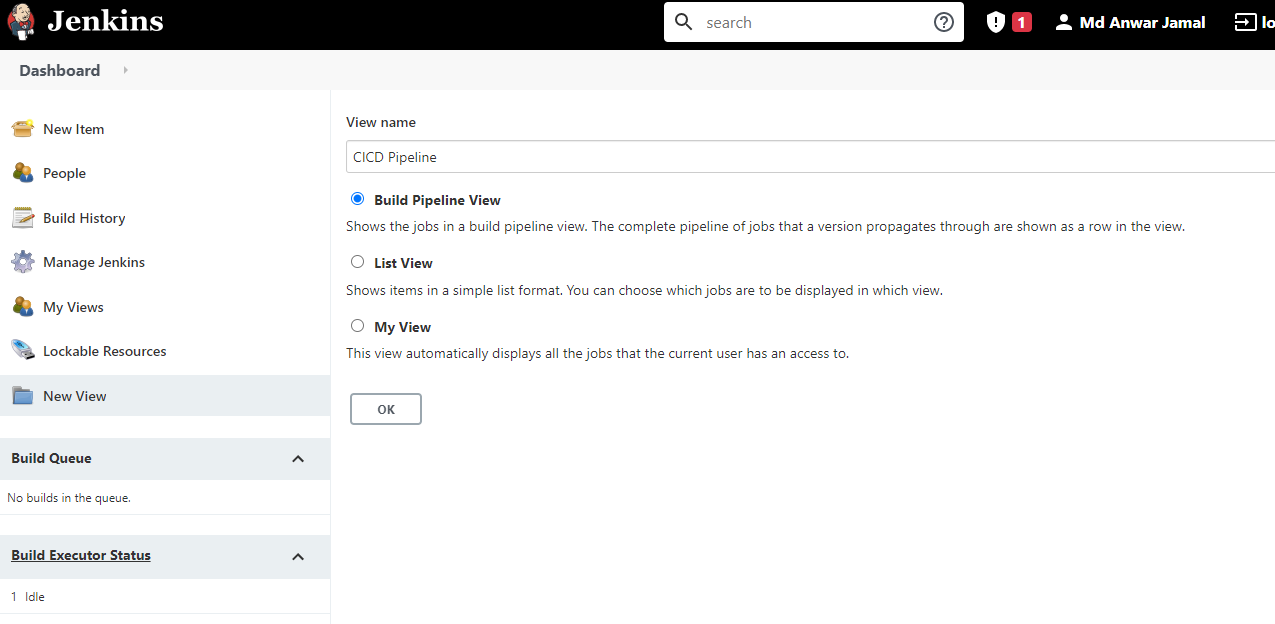
e.g., **index** ===> **<URL:8080>/index**

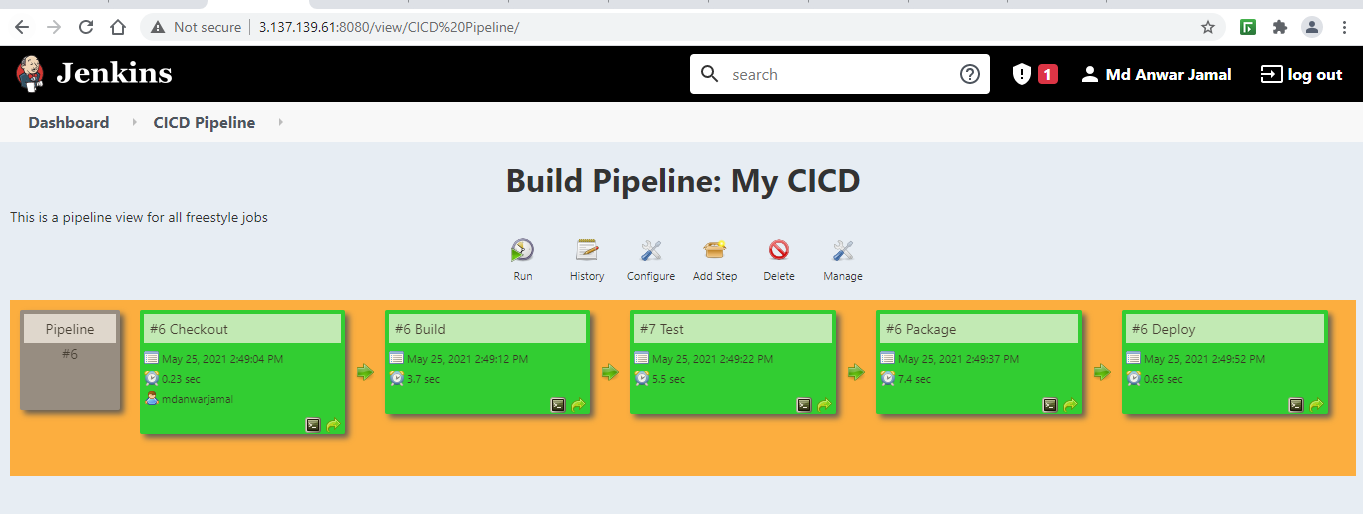
Tomcat Info: URL and User credentials

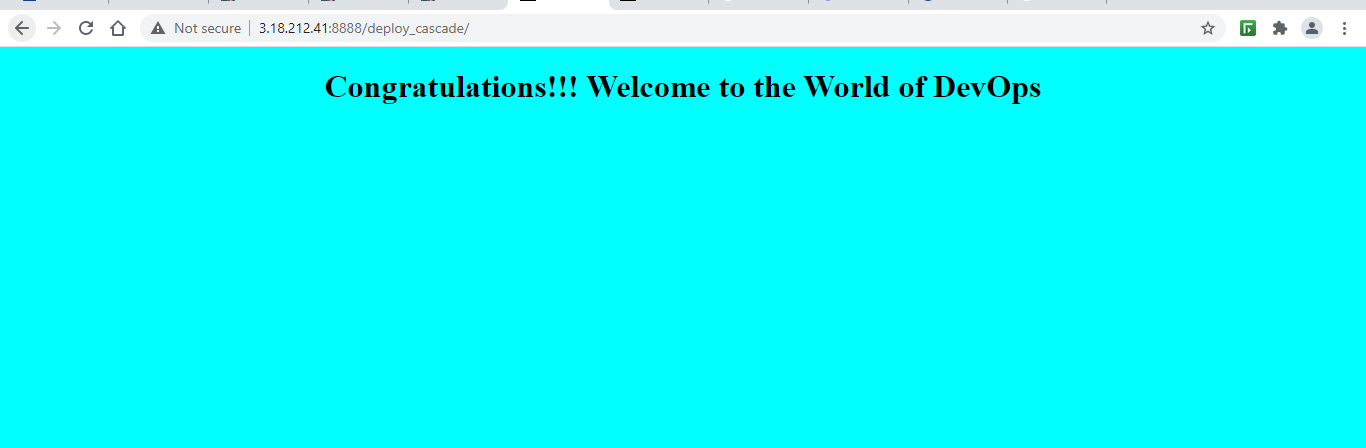
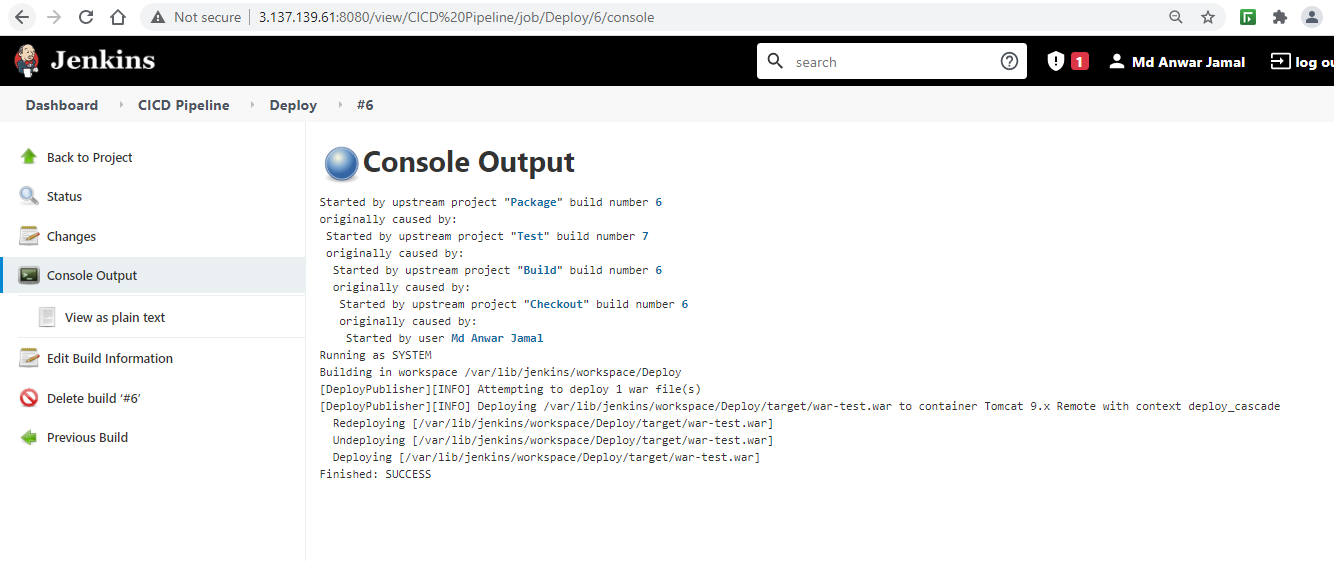
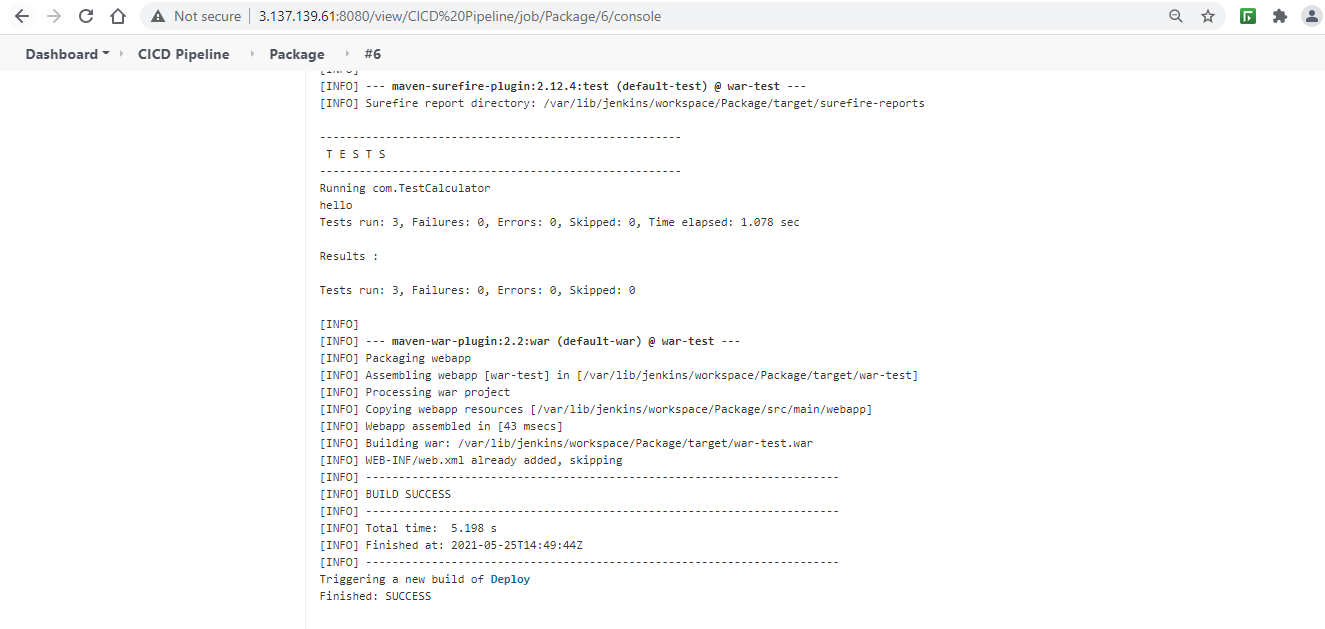
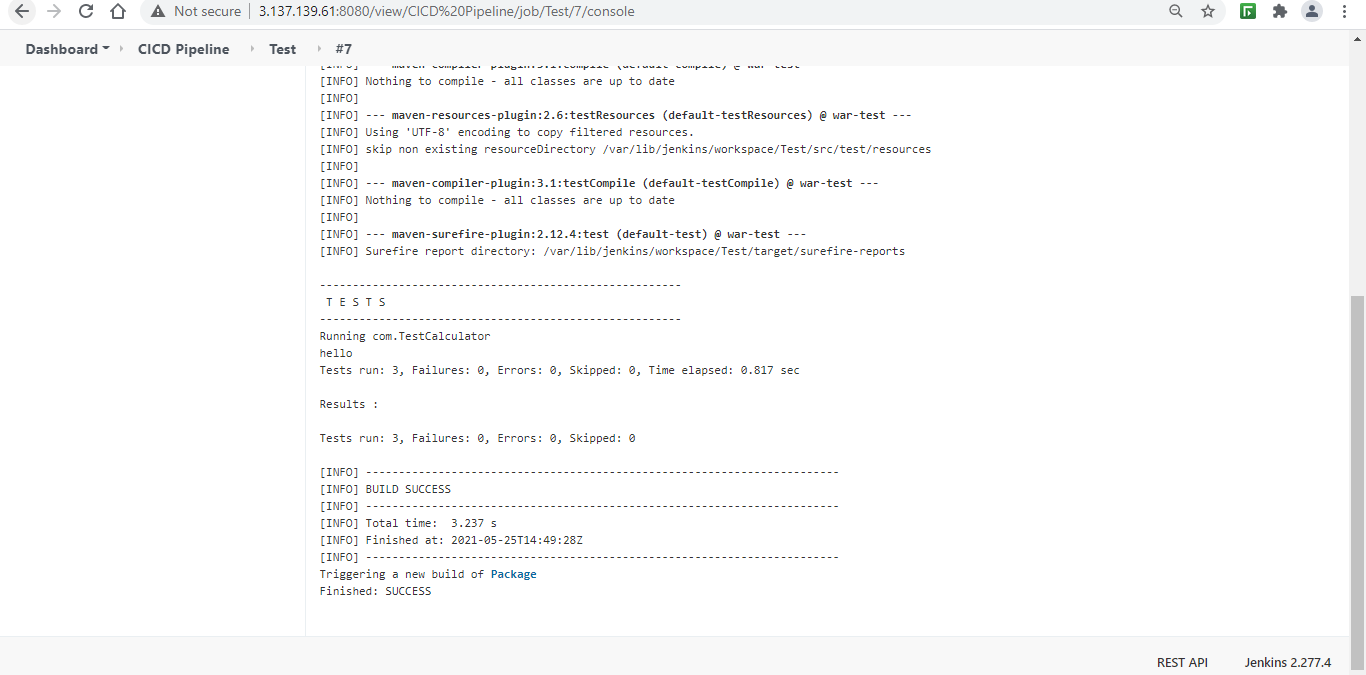
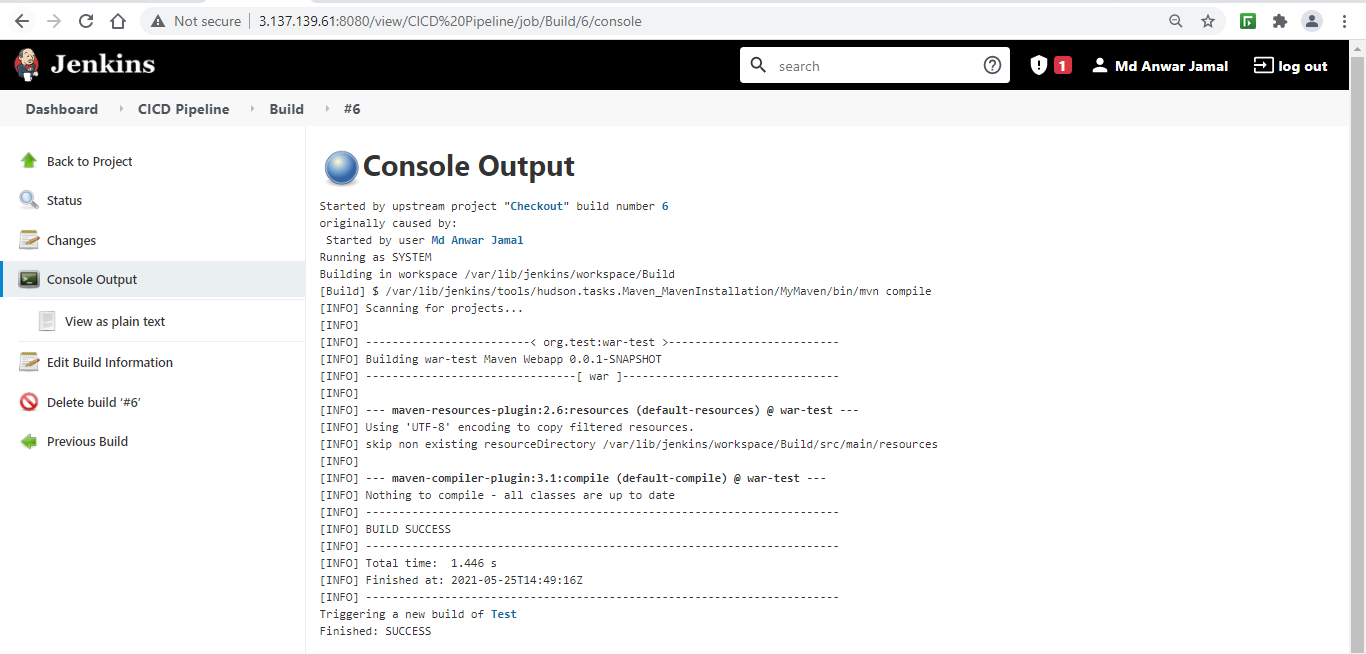
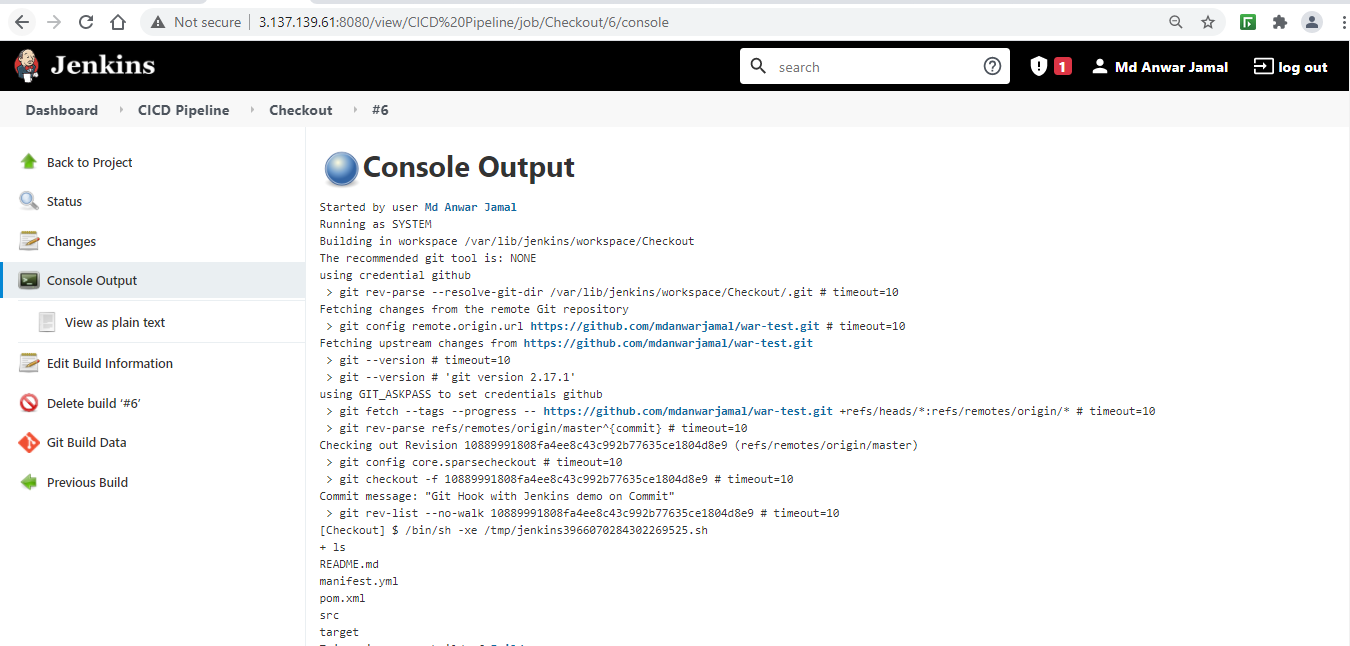


*Pipeline*

**Using Build Pipeline plugin, we can create a pipeline view of Jenkins jobs**







*Using Groovy Script and Git Hook*

**Using Pipeline as Item Type, we can use Groovy Scripts to build a CI/CD Pipeline**

**And With GitHub Hook as build trigger, we can build this pipeline script on some git actions, such as commit, push, etc.**

