MIRACLE
SOFTWARE SYSTEMS

JENKINS RELEASE MANAGEMENT

A.AnuManasa
Apigee Developer
Miracle Software Systems, Inc.

CH.Naga Laxmi
Apigee Developer
Miracle Software Systems, Inc.

1. Introduction:

- Jenkins is a powerful application that allows “continuous integration” and “continuous delivery” of projects.

  Jenkins is a Software that allows continuous integration and continuous delivery of projects, regardless of the platform.

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

2. Why Jenkins:

- Jenkins is a software that allows **continuous integration**. Jenkins will be installed on a server where the central build will take place.

3. Continuous Integration:

- Continuous Integration is a development practice that requires developers to integrate code into a shared repository at several times per day. Continuous Integration improves Software Quality and Reduces the Risk.
- Committing code frequently.
- Categorizing developer tests.
- Using a dedicated integration build machine.

- Using continuous feedback mechanisms.
- Staging builds.

4. Release Management:

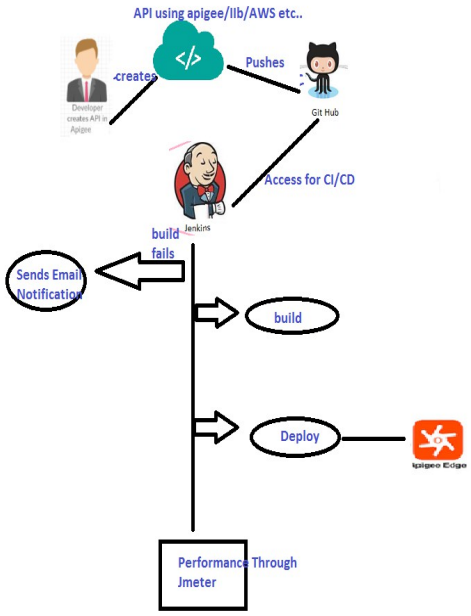
- **Release management** is the process of **managing**, planning, scheduling and controlling a software build through different stages and environments; including testing and deploying software releases.

5. Purpose:

- It automates the process of building an API from Git Hub to Apigee Edge using Jenkins.

6. Architectural Diagram:

Fig 1. Jenkins Architectural Diagram



7. Requirements:

- Git Hub(public Repository)
- Jenkins
- Apigee Edge
- GitBash

8. Jenkins Installation Steps:

- For Installing Jenkins in Windows we have to follow the below steps.
- Download the war file from the Jenkins website(Let the war file name be “jenkins.war”)
- Now, go to the location where we have the war file and run it using the followinf command.

```
java -jar jenkins.war
```

9. Creating user in Jenkins:

- So now Jenkins is installed, updated and it is in running state.
- Go to browser and give "http://localhost:8080".
- Give your details as shown in the following screen shot.
- And click save and finish.
- Now you can see the Jenkins screen as shown below.
- Click "start using Jenkins.

Fig 2. Login to Jenkins

10. Adding required Plugin s in Jenkins:

- For Integrating Git Hub with Jenkins install all the required plugins in Jenkins like
  - github plugin

- git plugin

GitHub Plugin:

- This plugins is mainly used for integrates Jenkins with Git Hub project.

Maven Installation Plugin:

- Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM).
- When a Maven project is created, Maven creates default project structure.

- For installing the required plugins go to
  - "Manage Jenkins (in the left navigation menu) --> Manage plugins".

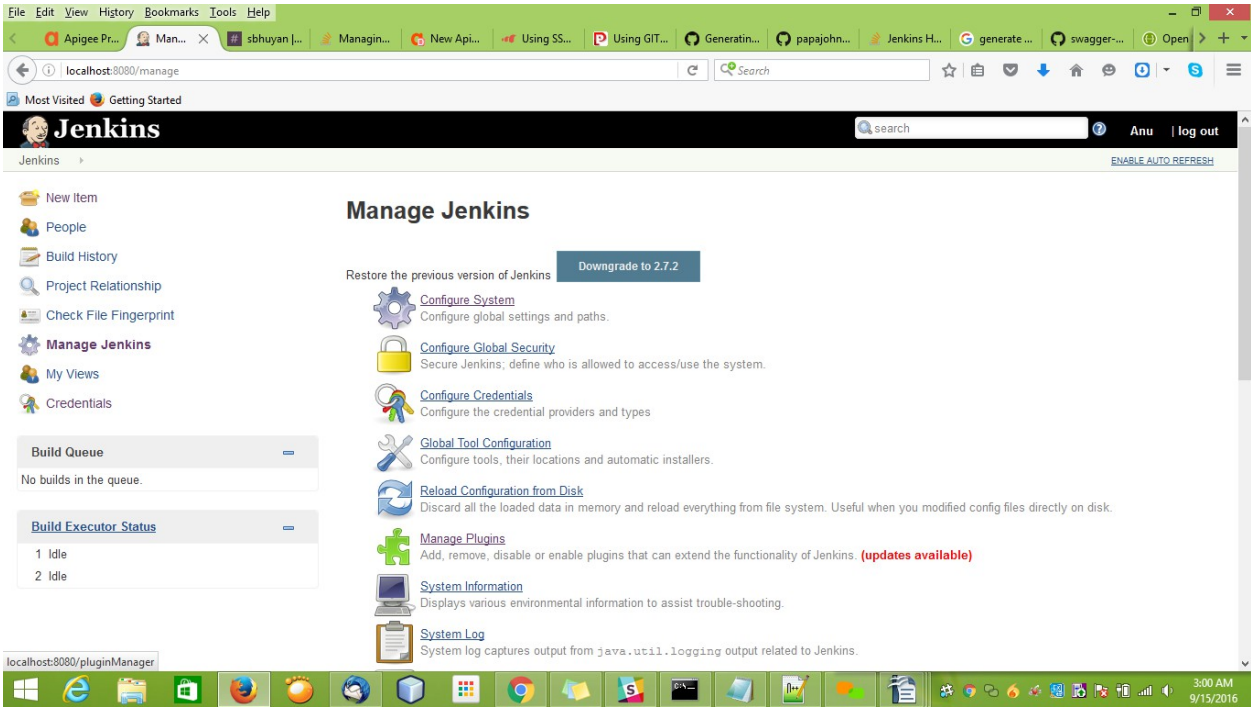


Fig 3. Adding Plugins

- Search for the required plugin to be installed in Available tab. If the plugin is already installed it was in Installed tab.

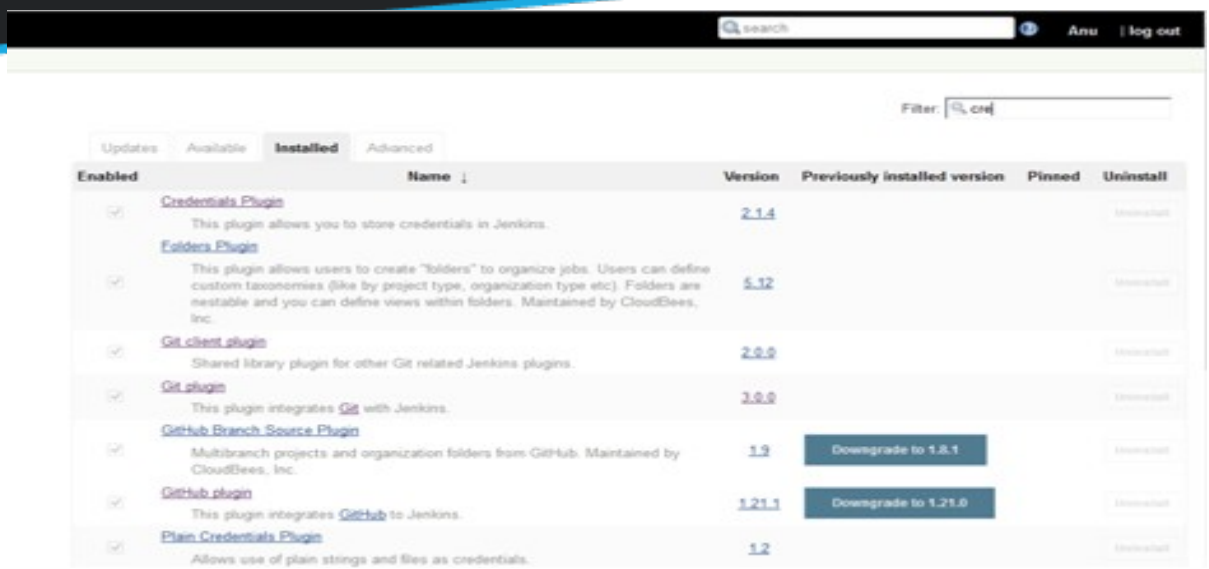


Fig 4. Installed Plugins

11. Creating Job:

- Click on new item then select a Maven project.

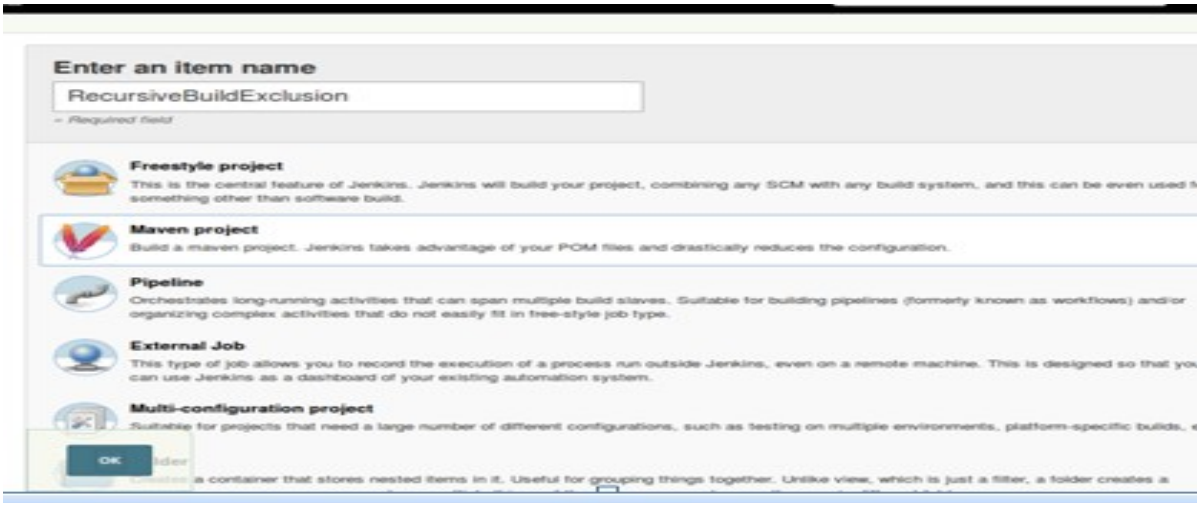


Fig 5.Creation of Maven Project

- Now configure the job with the following details

- In the "**Source Code Management** " section click on "Git " radio button and give the following github url.https://github.com/AnuManasaA/ReleaseManagement.git

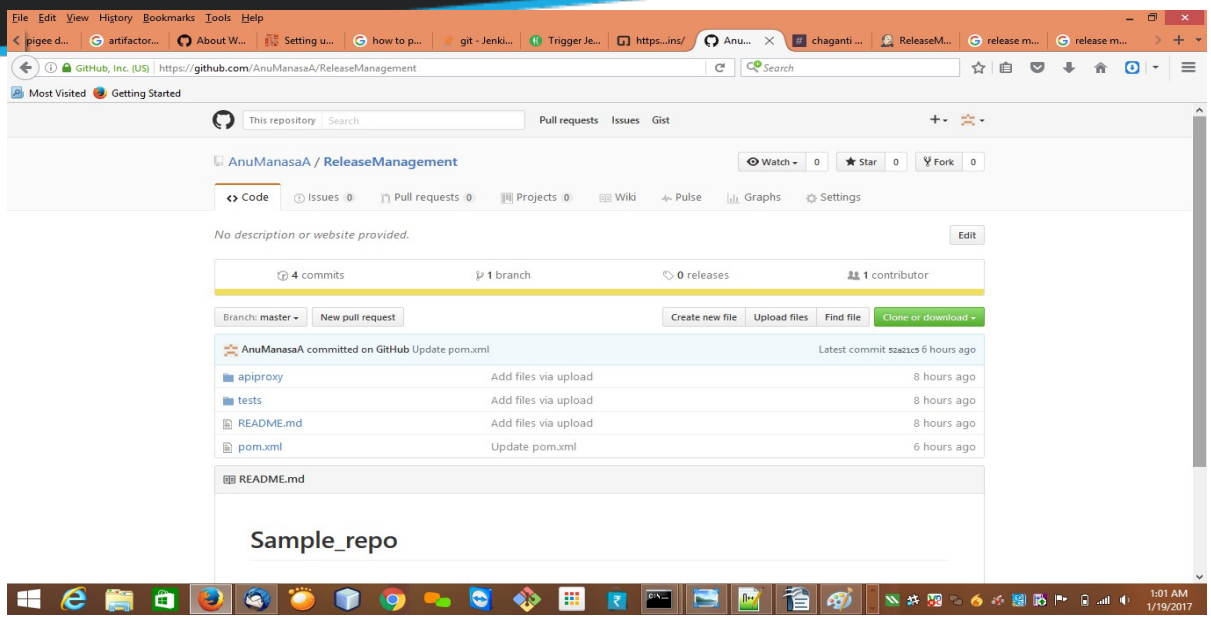


Fig 6. Git Hub Repository Details

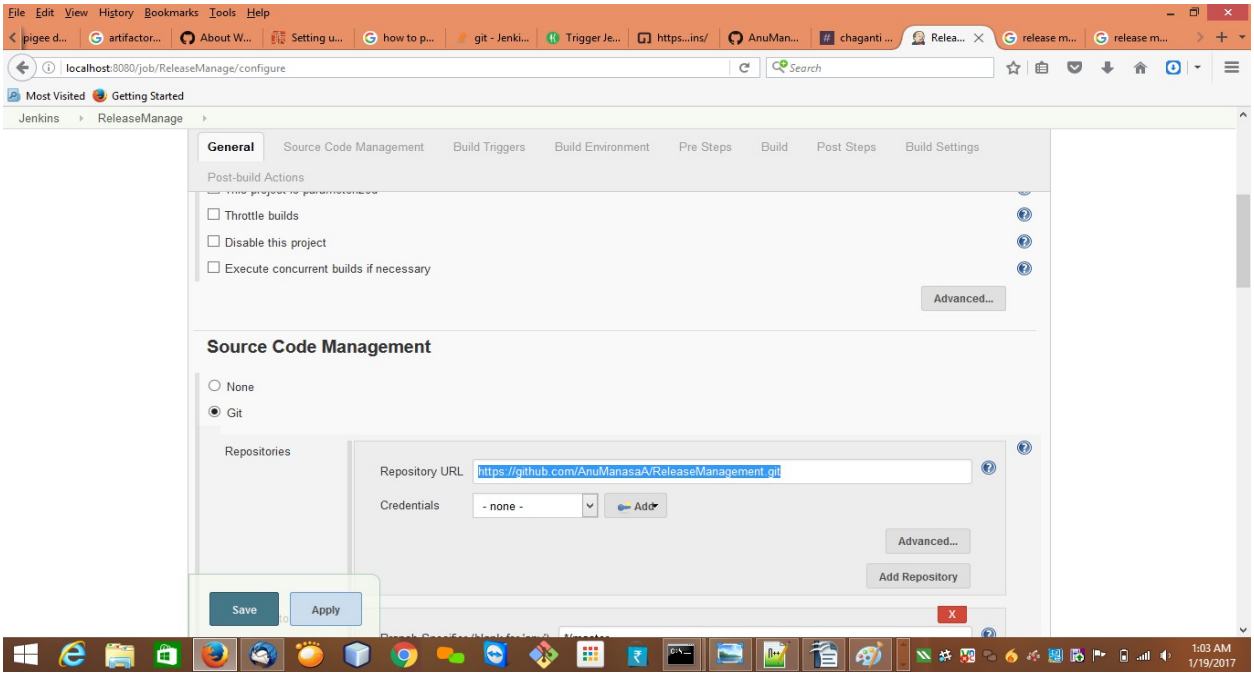


Fig 7. Configuring Jenkins with Git Hub Details

- and now give click on the "credentials" drop down and select the username which you have given for your "SSH Public key" credentials.

Fig 8. Configuring pom.xml

- In Build tab we have to mention the path of pom.xml. Here pom.xml is a build file. It defines the goals we have to achieve.
- In Goals & Actions we have to give the maven command for integrating with Apigee Edge.

The following is the command:

install -P dev1 -D username=$ae_username -D password=$ae_password -Dorg=$ae_org

- Next we need to specify the parameters used in Maven command.
- For that click on Manage Jenkins and go to the Configure system.

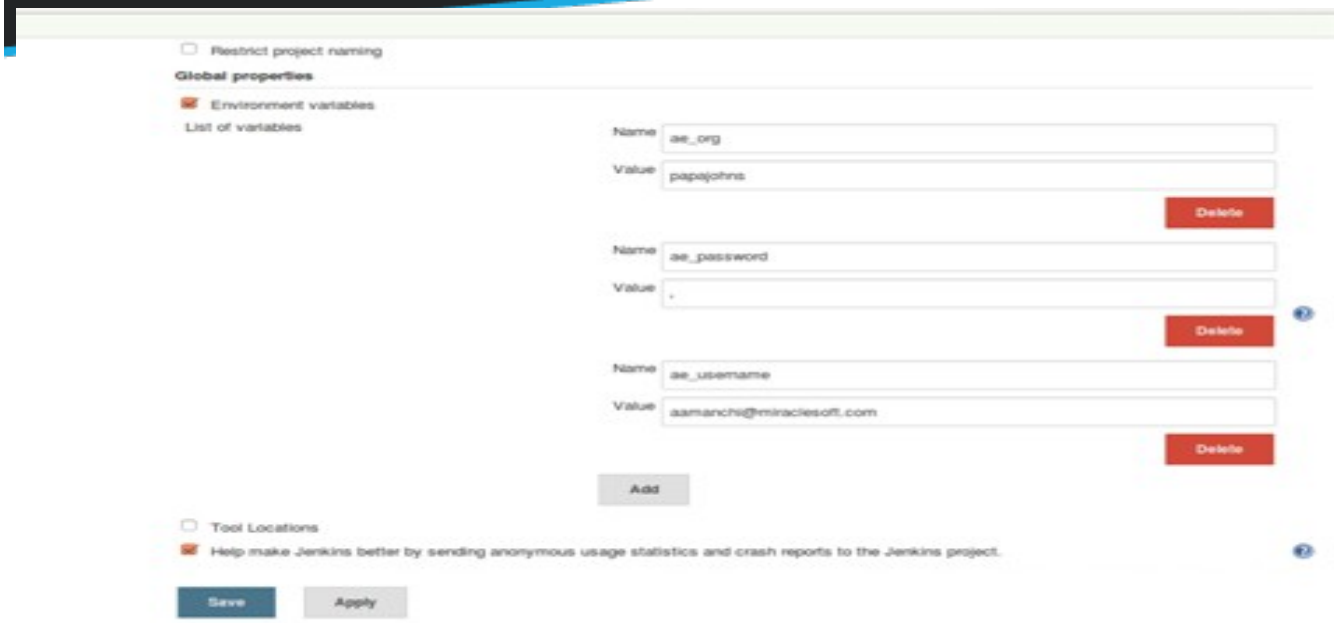


Fig 9. Adding Credentials of Apigee Edge User

- Now we have to give the Global properties for particular Apigee Edge user.

Fig 10. Adding Mask Password Plugin

- For masking the password of Apigge Edge account in the console output we have to add the Mask Password Plugin.

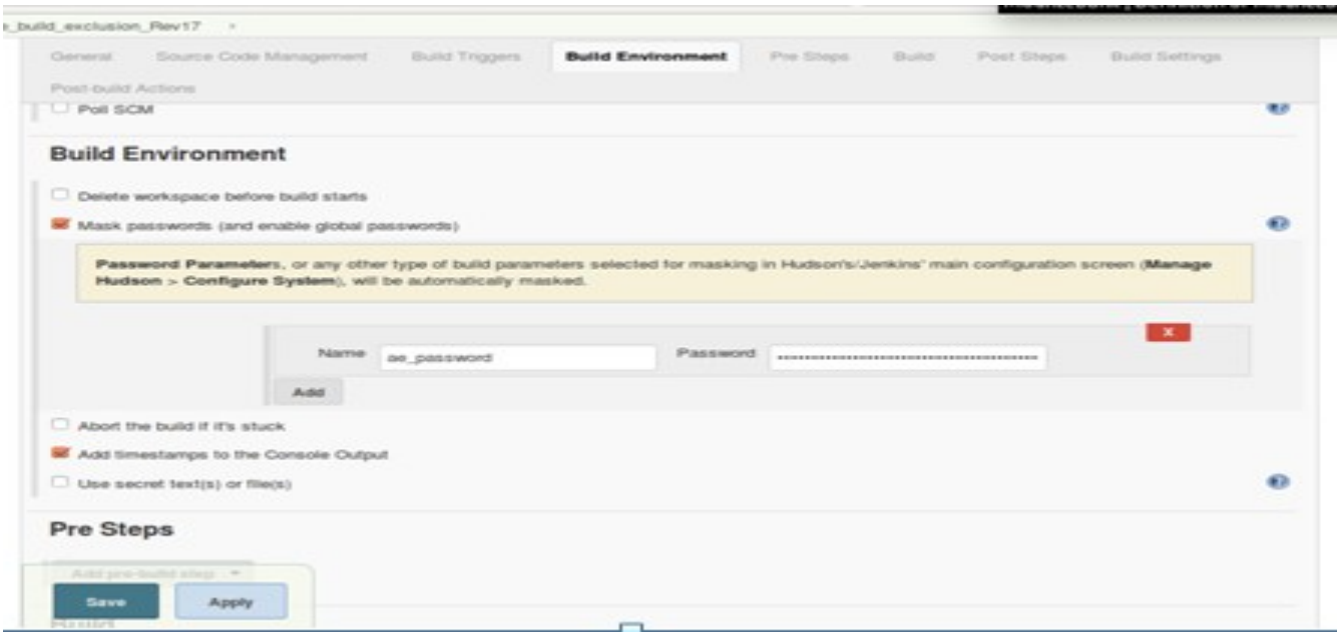


Fig 11.Configuring Password for masking in Jenkins job.

- Select the Mask passwords (and enable global passwords) in the Build Environment for the particular job.
- Mention the Name & password.

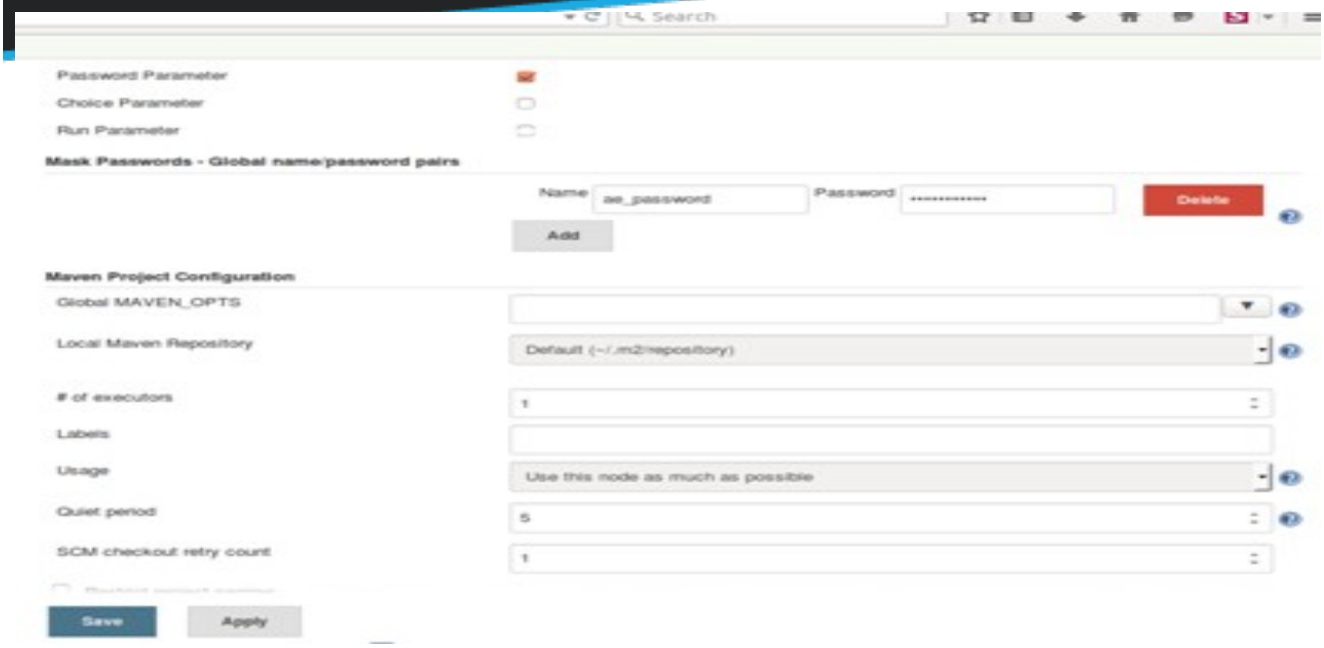


Fig 12. Configuring the name and password for masking.

- For hiding the password go to the configure system, Click on Add button in Mask Passwords-Global name/password pairs.
- Here mention Name and password.

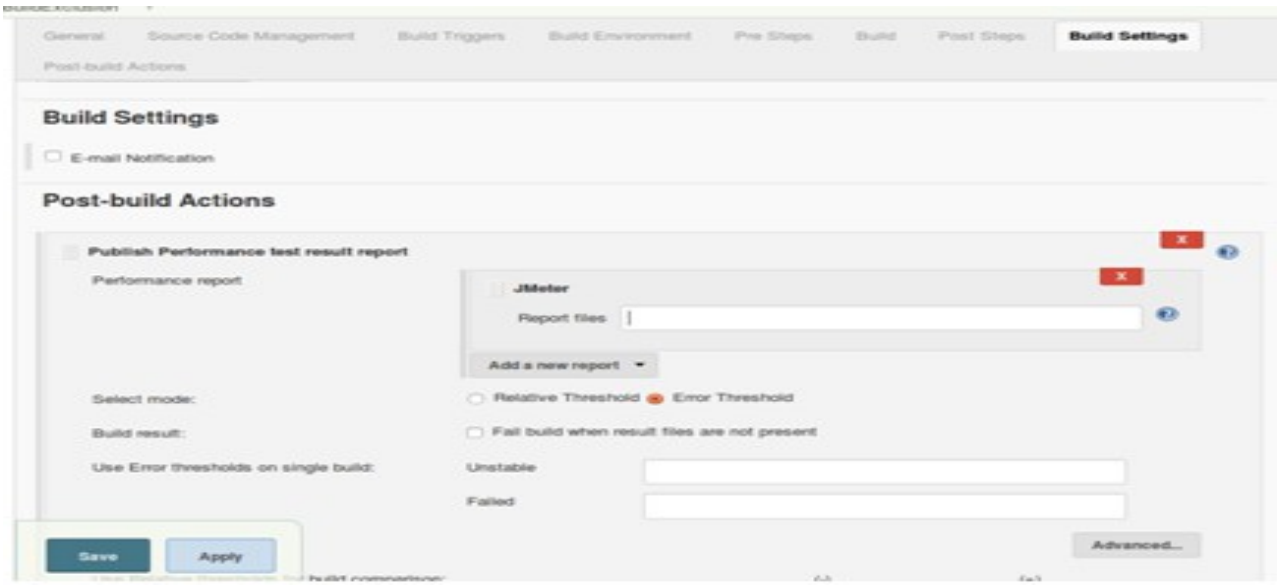


Fig 13. Configuring post-build Actions in Jenkins Job

- In Post-build Actions add publish performance test result report for showing performance of the Jmeter test cases.

- Mention the path for storing the Jmeter test cases result file in the Report files text field .

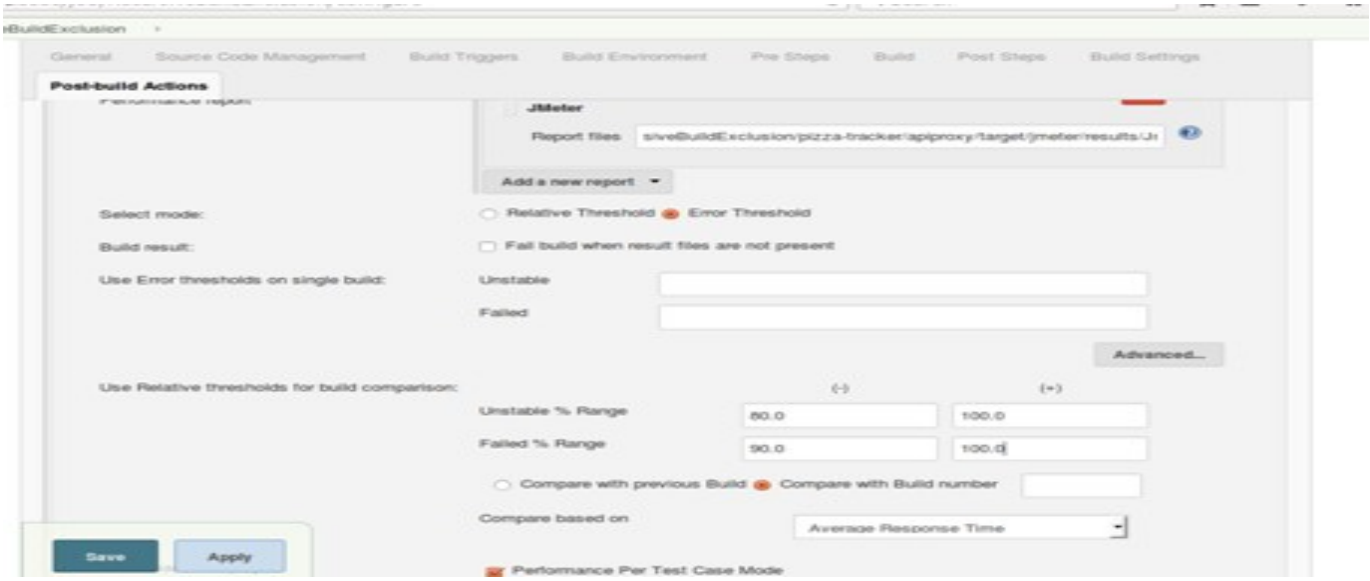


Fig 14. Specify the path for Result Report

12.ReleaseManagement Process:

Whenever we push/make any changes like version changing or adding some extra features to the previous API in GitHub we use this Release Management Process.

- For this, initially clone the repository into our local machine.

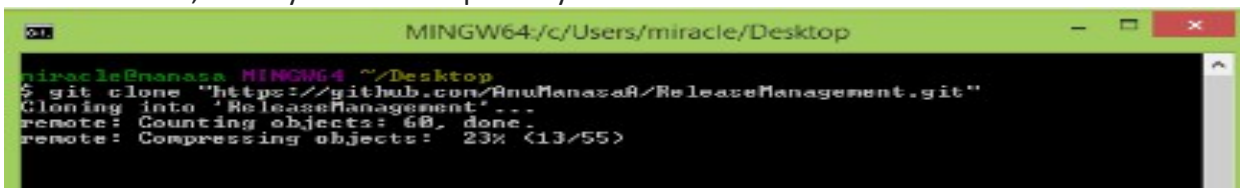


Fig 18. Cloning the Repository

Initially, open the Git Bash

Goto the repository Folder. And check the status of git by using the command "**git status**"

```
MINGW64:/c/Users/miracle/Desktop/ReleaseManagement
miracle@manasa MINGW64 ~/Desktop
$ git clone "https://github.com/AnuManasaA/ReleaseManagement.git"
Cloning into 'ReleaseManagement'...
remote: Counting objects: 60, done.
remote: Compressing objects: 100% (55/55), done.
remote: Total 60 (delta 19), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (60/60), done.
Checking connectivity... done.

miracle@manasa MINGW64 ~/Desktop
$ cd ReleaseManagement

miracle@manasa MINGW64 ~/Desktop/ReleaseManagement (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

Fig 19. Checking the Git Status

- Now , make the changes you want and check the status again.
- Then give the command

  git add .

- This command adds all modified and new files in the current directory and all subdirectories to the staging area

```
miracle@manasa MINGW64 ~/Desktop/ReleaseManagement (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   pom.xml

no changes added to commit (use "git add" and/or "git commit -a")

miracle@manasa MINGW64 ~/Desktop/ReleaseManagement (master)
$ git add .

miracle@manasa MINGW64 ~/Desktop/ReleaseManagement (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   pom.xml
```

Fig 12. Adding Changes

- Now, change the directory to .git/ and then to hooks using commands

  cd .git/

  cd hooks

```
miracle@manasa MINGW64 ~/Desktop/ReleaseManagement (master)
$ cd .git/

miracle@manasa MINGW64 ~/Desktop/ReleaseManagement/.git (GIT_DIR!)
$ cd hooks_
```

Fig 13. Checking Hooks

- Give the command **vi post-commit.** This opens the vi editor given

Fig 14. Opening Vi Editor

- press "i". This gives permission to edit in the vi editor
- Lets see the syntax of the post-commit script

```
curl --user anucse2k11@gmail.com:00c536958e10c88939fd114cc1100846 http://localhot:8080/job/ReleaseManagement/build?token=gitbuild234
```

- Here in the command give curl –user userid:APIToken url for the build of project? token=token given while triggering
- Go to Configure

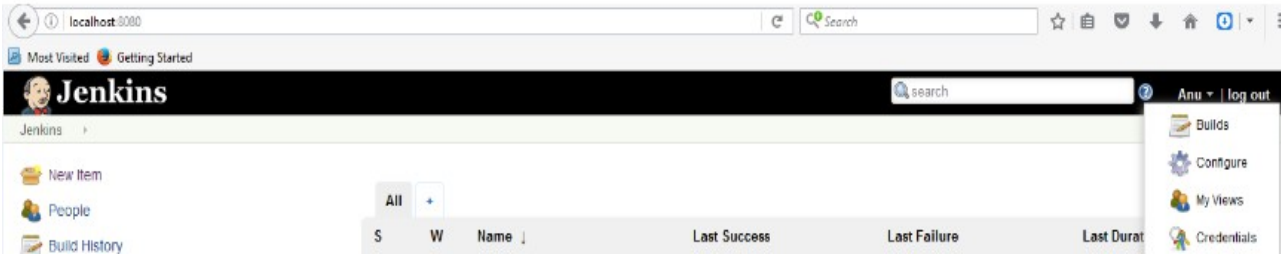


Fig 15. Getting API Token

- Click on **Show API Token**

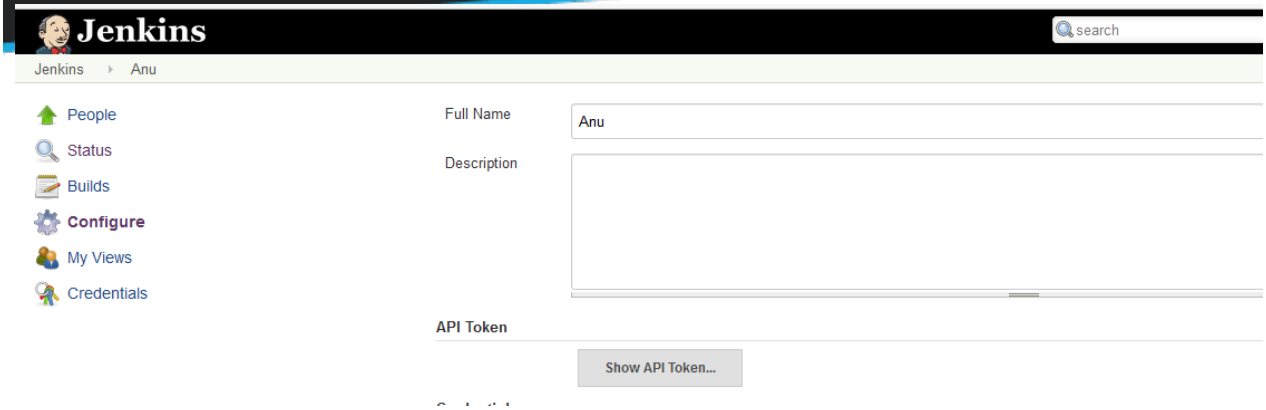


Fig 16. Getting API Token

- You can now note the userid and API token

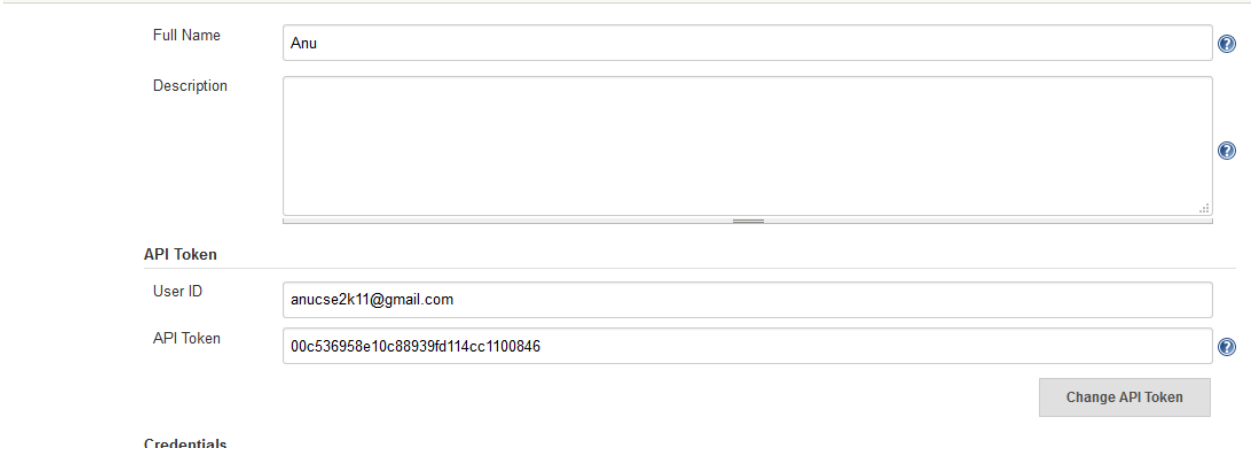


Fig 17. Getting API Token

- Now, go to the project and click on configure

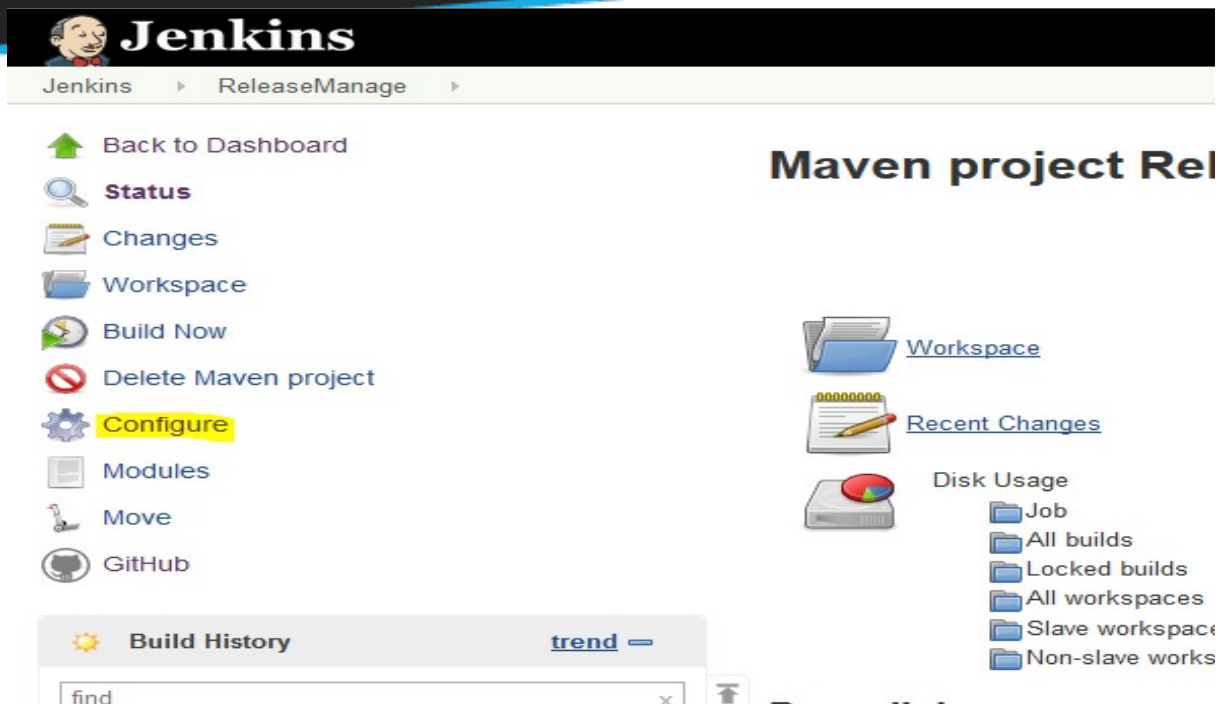


Fig 18. Specify Token

- Then goto **Build Triggers** and select **Trigger builds remotely** and note the token

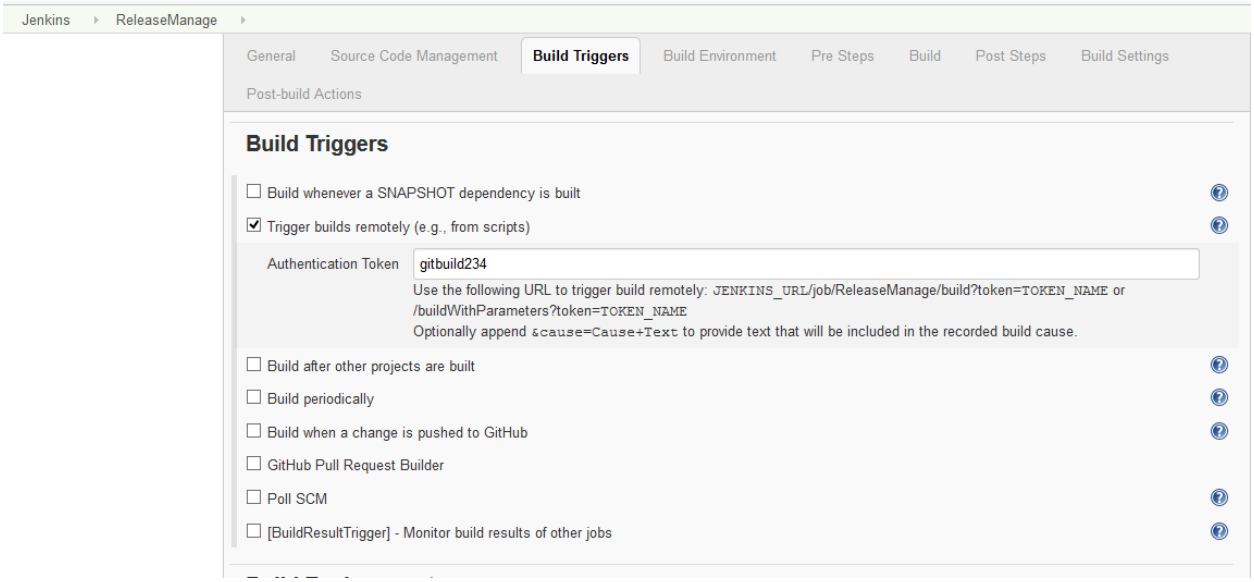


Fig 19. Specifying Authentication Token

```
miracle@manasa MINGW64 ~/Desktop/ReleaseManagement/.git/hooks (GIT_DIR!)
$ cd ../..

miracle@manasa MINGW64 ~/Desktop/ReleaseManagement (master)
$
```

Fig 20. Getting back to the project Folder

- Whenever we commit the changes by using the command

  "git commit -m "commit the changes""

- This builds the project automatically in Jenkins.

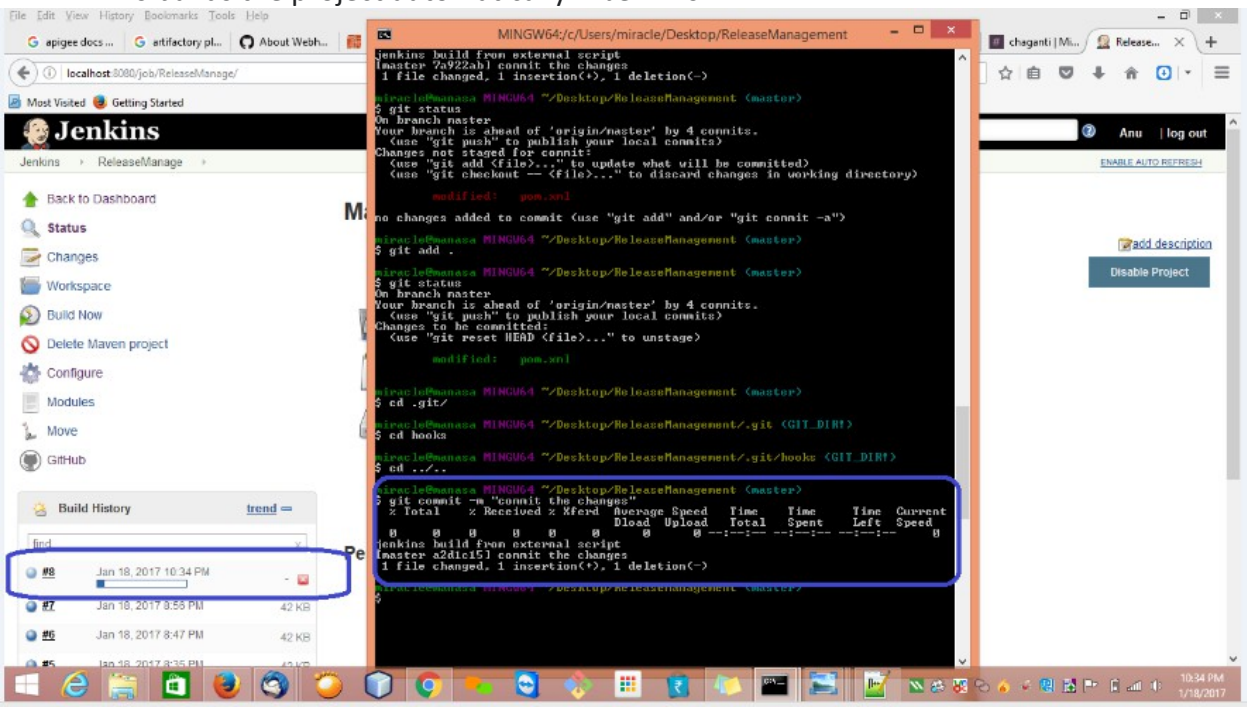


Fig 21. Committing Changes

- In Post-build Actions add publish performance test result report for showing performance of the Jmeter test cases.

- Mention the path for storing the Jmeter test cases result file in the Report files text field .

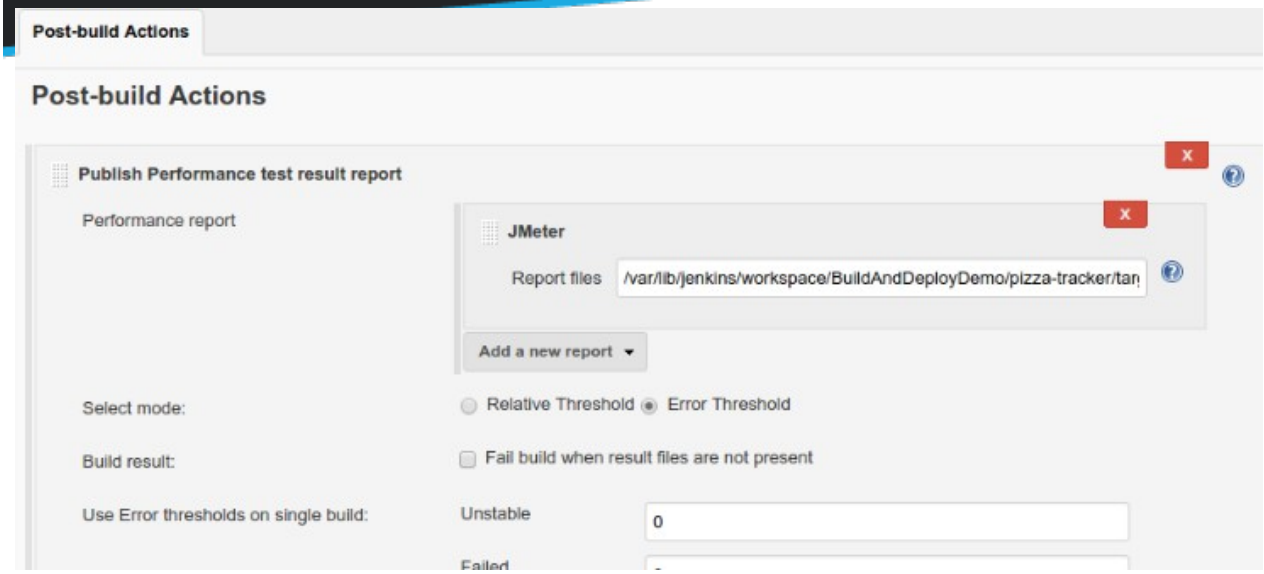


Fig 22. Specify the path for Result Report

- Next Build the project and see the console output.

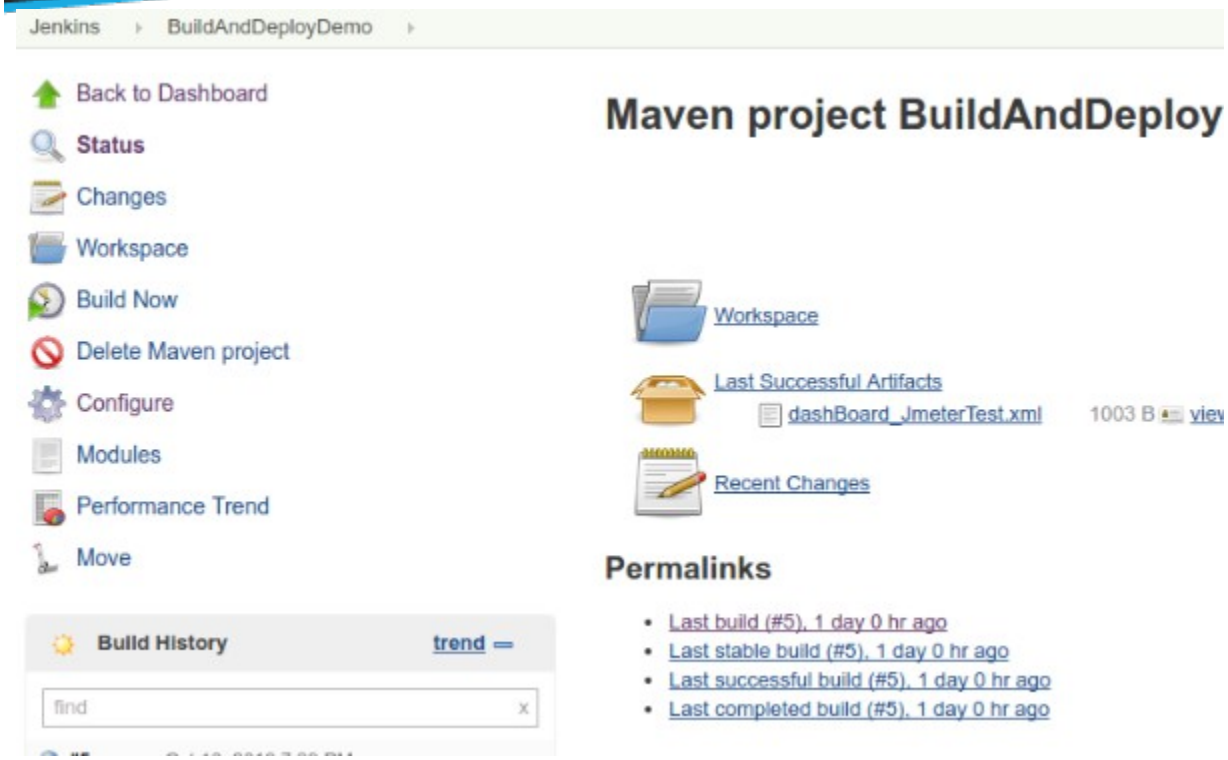


Fig 23. Build the Job

- In the below figure we observe that Jmeter test cases running successfully, and stored into JmeterTest.jtl file.

Console Output

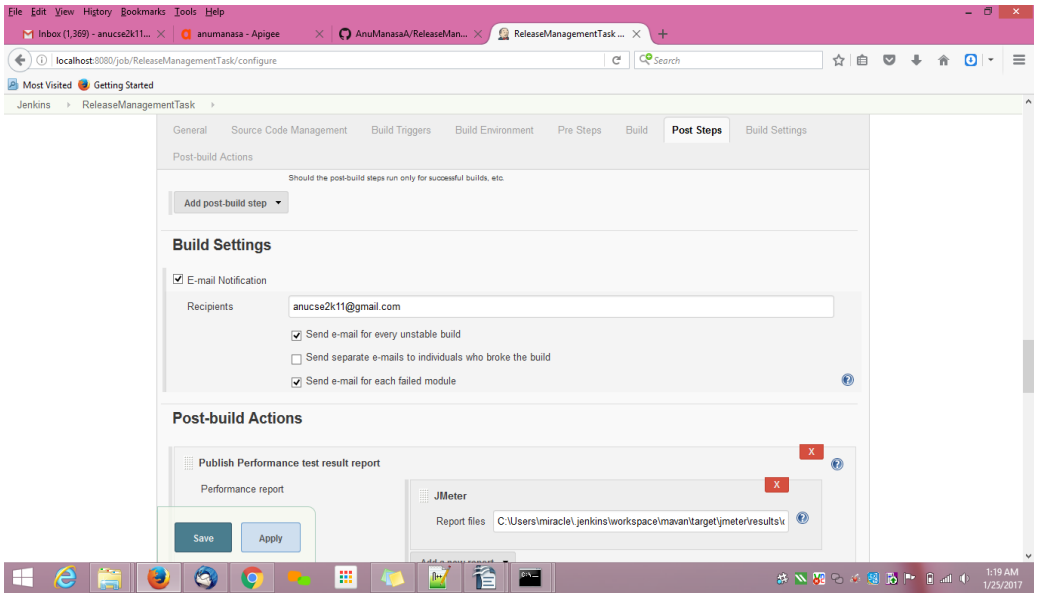
```
Started by user Anu
Building in workspace C:\Users\miracle\.jenkins\workspace\mavan
 > git.exe rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git.exe config remote.origin.url https://github.com/Venkatalakshmik/Sample_repo.git # timeout=10
Fetching upstream changes from https://github.com/Venkatalakshmik/Sample_repo.git
 > git.exe --version # timeout=10
 > git.exe fetch --tags --progress https://github.com/Venkatalakshmik/Sample_repo.git +refs/heads/*:refs/remotes
/origin/*
 > git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
 > git.exe rev-parse "refs/remotes/origin/origin/master^{commit}" # timeout=10
Checking out Revision e779c26973132fe6897bb3f8597496978ac9b4be (refs/remotes/origin/master)
 > git.exe config core.sparsecheckout # timeout=10
 > git.exe checkout -f e779c26973132fe6897bb3f8597496978ac9b4be
 > git.exe rev-list e779c26973132fe6897bb3f8597496978ac9b4be # timeout=10
Parsing POMs
Established TCP socket on 53977
[mavan] $ java -cp C:\Users\miracle\.jenkins\plugins\maven-plugin\WEB-INF\lib\maven32-agent-1.7.jar;C:\Users\miracle
\.jenkins\tools\hudson.tasks.Maven_MavenInstallation\Maven_3.3.9\boot\plexus-classworlds-2.5.2.jar;C:\Users\miracle
\.jenkins\tools\hudson.tasks.Maven_MavenInstallation\Maven_3.3.9/conf/logging jenkins.maven3.agent.Maven32Main C:\Users
\miracle\.jenkins\tools\hudson.tasks.Maven_MavenInstallation\Maven_3.3.9 C:\Users\miracle\.jenkins\war\WEB-INF\lib
\remoting-2.60.jar C:\Users\miracle\.jenkins\plugins\maven-plugin\WEB-INF\lib\maven32-interceptor-1.7.jar C:\Users
\miracle\.jenkins\plugins\maven-plugin\WEB-INF\lib\maven3-interceptor-commons-1.7.jar 53977
<===[JENKINS REMOTING CAPACITY]===>channel started
```

Fig 24. Console Output

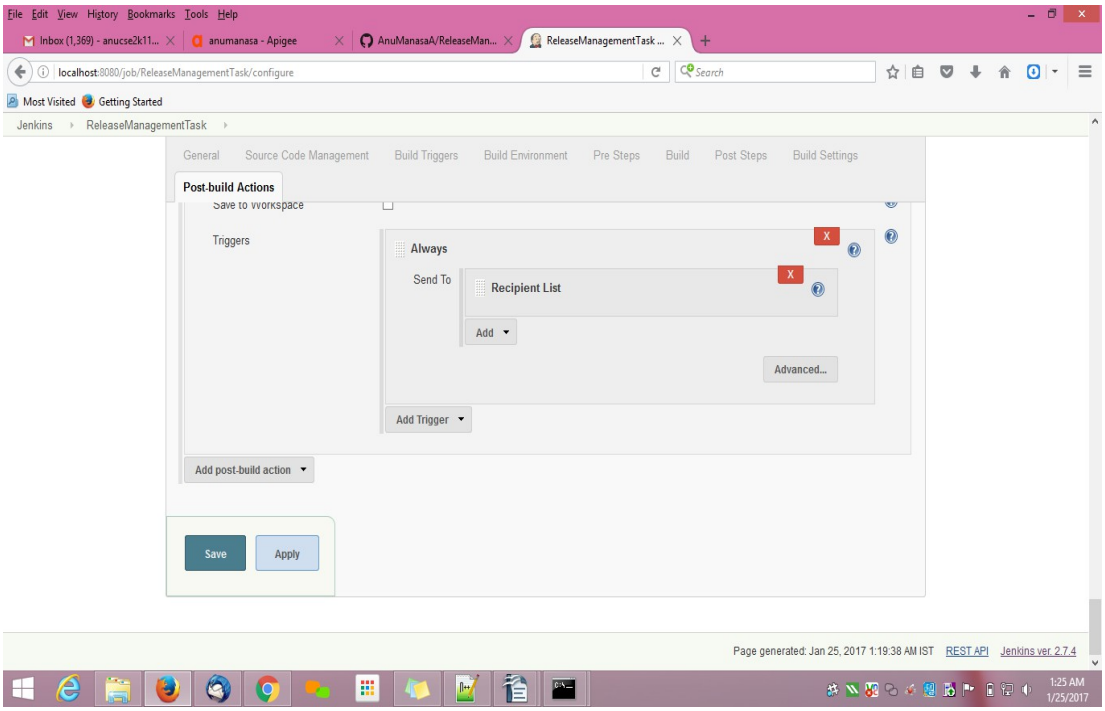
EMAIL Notification:

For sending notification we have to add Email Extension Plug-in

And the job will be configured for failure cases as follows:



Build is failed ,mail will be sent to the recepient as shown:



MIRACLE SOFTWARE SYSTEMS

File Edit View History Bookmarks Tools Help

Inbox (1,369) - anucse2k11... | anumanasa - Apigee | AnuManasaA/ReleaseMan... | Configure System [Jenkins]

localhost:8080/configure

Most Visited | Getting Started

Jenkins › configuration

SMTP server	smtp.gmail.com
Default user e-mail suffix	
☑ Use SMTP Authentication	
User Name	aaamanasa@gmail.com
Password	••
Use SSL	☑
SMTP Port	465
Reply-To Address	anucse2k11@gmail.com
Charset	UTF-8
☑ Test configuration by sending test e-mail	
Test e-mail recipient	anucse2k11@gmail.com
Email was successfully sent	Test configuration

Publish over SSH

Save | Apply

1:26 AM 1/25/2017

Sent Mail - aaamanasa@ | Venkatalakshmik/Calcula | how to increase permge

Secure | https://mail.google.com/mail/#sent

Google | in:sent

Click here to enable desktop notifications for Gmail. Learn more Hide

Gmail | More | 1–30 of 30

COMPOSE

Inbox (80)
Starred
Important
Sent Mail
Drafts
More

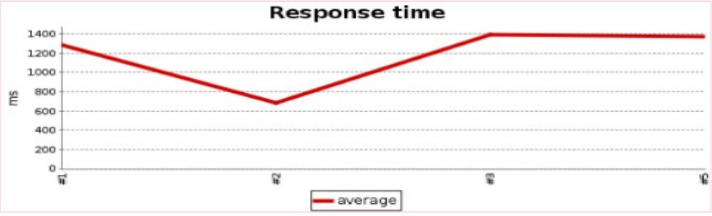
manasa

No recent chats
Start a new one

To: anucse2k11	Jenkins build is back to normal : ReleaseManagementTask #37 - See <http://localhost:8080/job/ReleaseManagementTask/37/>	11:53 pm
To: anucse2k11	Jenkins build is back to normal : ReleaseManagementTask #29 - See <http://localhost:8080/job/ReleaseManagementTask/29/chang	8:56 pm
To: anucse2k11	Jenkins build is back to normal : ReleaseManagementTask » PT #29 - See <http://localhost:8080/job/ReleaseManagementTask/apig	8:56 pm
To: anucse2k11	Test email #1 - This is test email #2 sent from Jenkins	8:10 pm
To: anucse2k11	Test email #2 - This is test email #2 sent from Jenkins	8:10 pm
To: anucse2k11	Build failed in Jenkins: ReleaseManagementTask #27 - See <http://localhost:8080/job/ReleaseManagementTask/27/> [...truncated !	7:28 pm
To: anucse2k11	Build failed in Jenkins: ReleaseManagementTask #26 - See <http://localhost:8080/job/ReleaseManagementTask/26/> Started by us	6:42 pm
To: anucse2k11	Build failed in Jenkins: ReleaseManagementTask » PT #26 - See <http://localhost:8080/job/ReleaseManagementTask/apigee$PT/2	6:42 pm
To: anucse2k11	Build failed in Jenkins: ReleaseManagementTask #25 - See <http://localhost:8080/job/ReleaseManagementTask/25/> Started by us	4:14 pm
To: anucse2k11	Build failed in Jenkins: ReleaseManagementTask #24 - See <http://localhost:8080/job/ReleaseManagementTask/24/> Started by us	4:07 pm
To: anucse2k11	Jenkins build is back to normal : ReleaseManagementTask #17 - See <http://localhost:8080/job/ReleaseManagementTask/17/>	Jan 24
To: anucse2k11	Build failed in Jenkins: ReleaseManagementTask #16 - See <http://localhost:8080/job/ReleaseManagementTask/16/> Started by us	Jan 24
To: anucse2k11	Build failed in Jenkins: ReleaseManagementTask #15 - See <http://localhost:8080/job/ReleaseManagementTask/15/> Started by us	Jan 24
To: anucse2k11 (2)	Test email #1 - This is test email #1 sent from Jenkins	Jan 24

1:27 AM 1/25/2017

Performance Breakdown by URI: JmeterTest.jtl



Response time trends for build:

"BuildAndDeployDemo #5"

URI	Samples	Samples diff	Average (ms)	Average diff (ms)	Median (ms)	Median diff (ms)	Line90 (ms)	Minimum (ms)	Maximun
Delete HTTP Request	1	0	4326	-65	4326	-65	4326	4326	
Get HTTP Request	2	0	424	0	538	-1	538	310	
Put HTTP Request	1	0	329	-18	329	-18	329	329	
All URIs	**4**	**0**	**1375**	**-21**	**538**	**-1**	**4326**	**310**	

Fig 25. Performance Report

- If the project is success then the API proxy will be deployed into Apigee Edge as shown below.

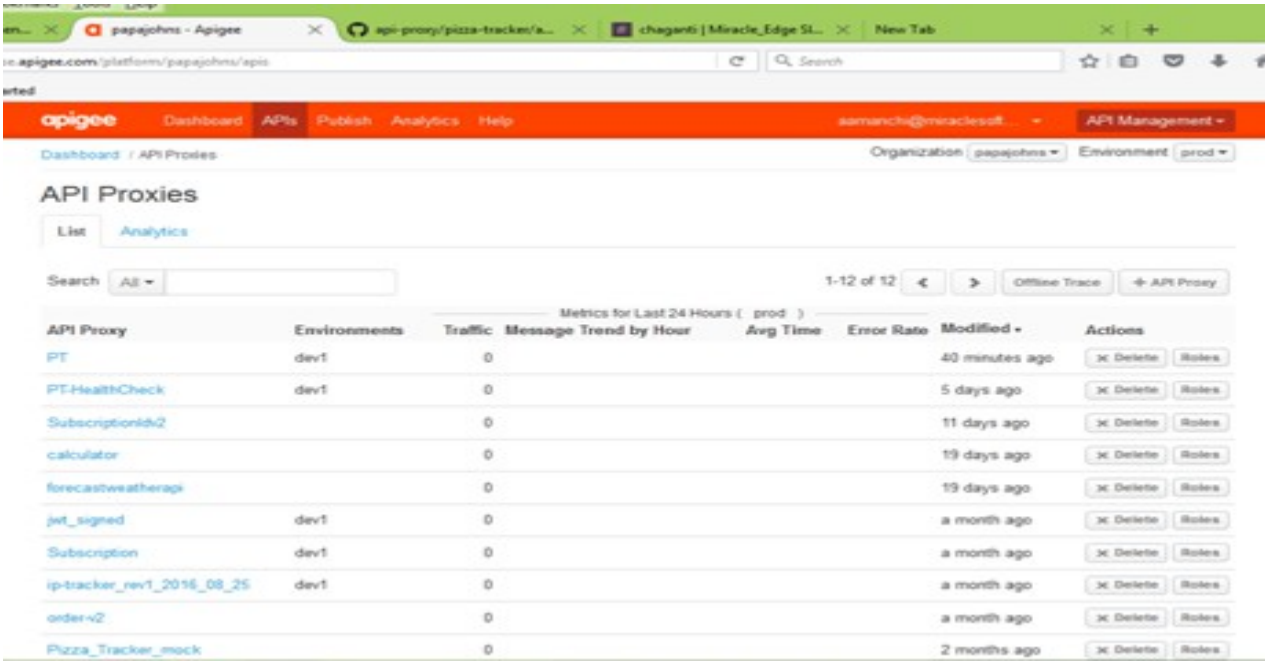


Fig 26. Deployed API Proxy in Apigee Edge