**Continuous Integration/Deployment – APIGEE**

**API M Practice**

Miracle Software Systems, Inc.

Contents

[1. Introduction 2](#_Toc475648821)

[2. Why Jenkins: 3](#_Toc475648822)

[3. Continuous Integration: 3](#_Toc475648823)

[4. Purpose: 3](#_Toc475648824)

[5. Architectural Diagram: 4](#_Toc475648825)

[6. Requirements: 4](#_Toc475648826)

[7. Jenkins Installation Steps: 4](#_Toc475648827)

[8. Creating user in Jenkins: 5](#_Toc475648828)

[9. Adding required Plugins in Jenkins: 6](#_Toc475648829)

[GitHub Plugin: 6](#_Toc475648830)

[Maven Installation Plugin: 6](#_Toc475648831)

[Credentials Plugin: 7](#_Toc475648832)

[10. Creating Job: 8](#_Toc475648833)

[11. Ubuntu Directory Locations: 17](#_Toc475648834)

[Jenkins Log file 19](#_Toc475648835)

[POM.XML 21](#_Toc475648836)

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.

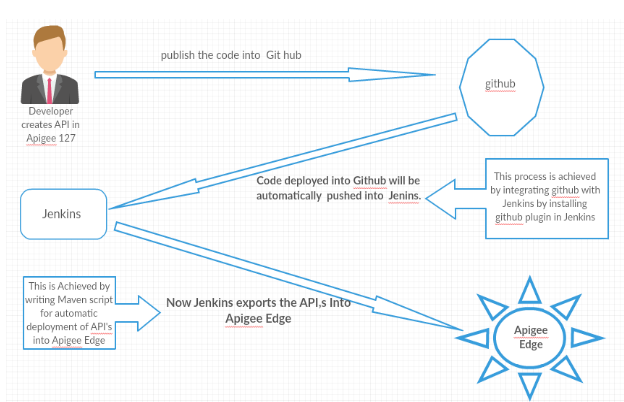
# 3. Continuous Integration:

* 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. Purpose:

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

# 5. Architectural Diagram:



**Fig 1. Jenkins Architectural Diagram**

# 6. Requirements:

* Git Hub
* Jenkins

# 7. Jenkins Installation Steps:

* For Installing Jenkins in Ubuntu we have to follow the below steps.
* Open your terminal. In that go to root by clinking “sudo -s”.
* and then follow the below commands.

wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt-get update

sudo apt-get install Jenkins

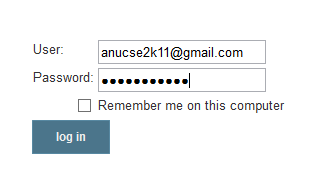
* Now Jenkins is installed in your machine, to update it to a latest version run the following command.

sudo apt-get update

sudo apt-get install Jenkins

# 8. Creating user in Jenkins:

* So now Jenkins is installed, updated and it is in running state.
* Go to browser and give “[http://localhost:8080](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**

# 9. Adding required Plugins in Jenkins:

* For Integrating Git Hub with Jenkins using SSH key install all the required plugins in Jenkins like
* github plugin
* git plugin
* Credentials plugin

## GitHub Plugin:

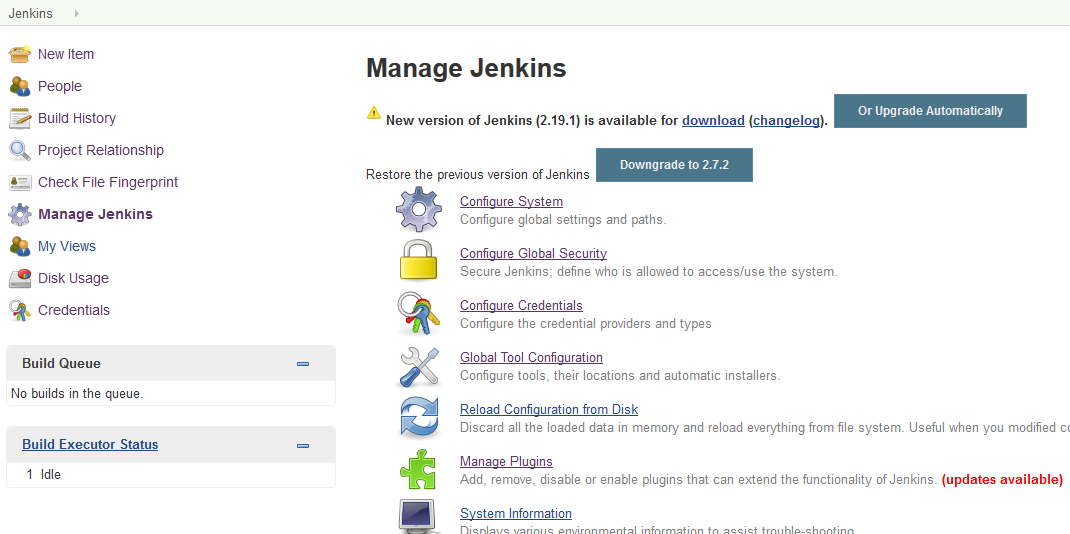
* This plugins is mainly used for integrates Jenkins with Git Hub project.

## 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.

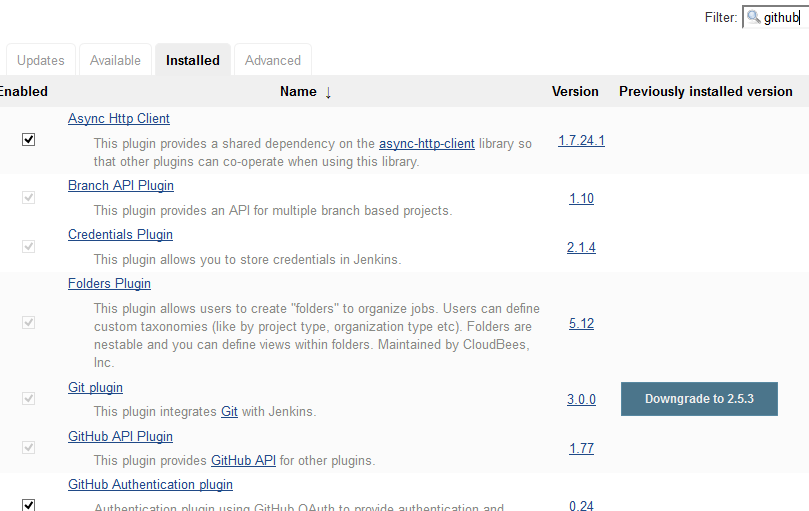
## Credentials Plugin:

* This plugin allows you to store credentials in Jenkins.
* The credentials plugin provides a standardized API for other plugins to store and retrieve different types of credentials.
* For installing the required plugins go to
* “Manage Jenkins (in the left navigation menu) --> Manage plugins”.



**Fig 3. Adding Plugins**

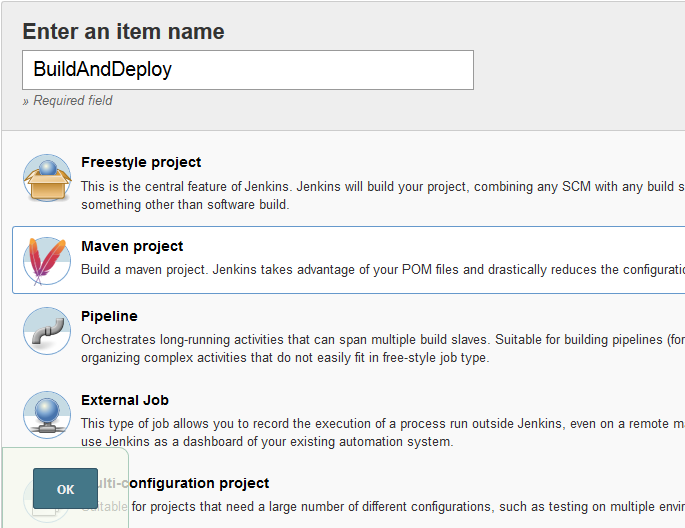
* Search for the required plugin to be installed in Available tab. If the plugin is already installed it was in Installed tab.



**Fig 4. Installed Plugins**

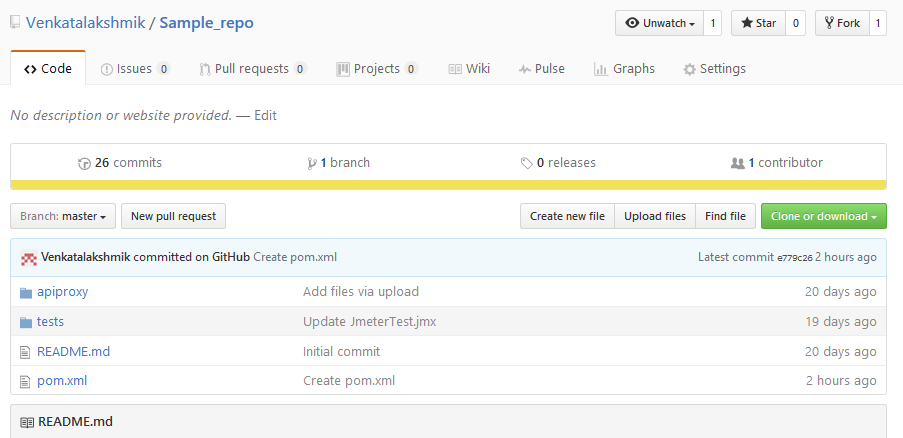
# 10. Creating Job:

* Click on new item then select a Maven project.

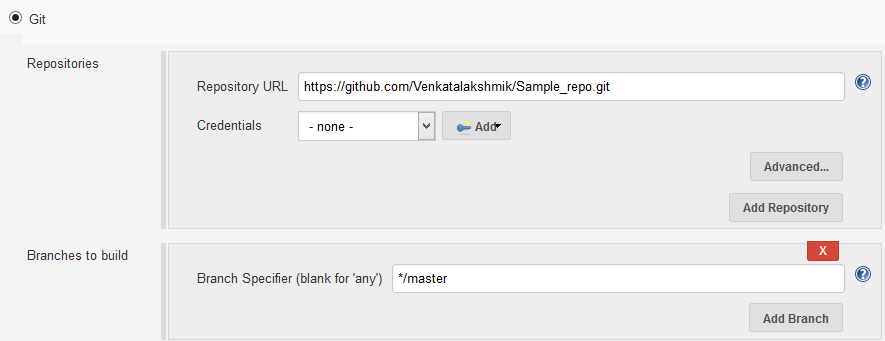


**Fig 8.Creation of Maven Project**

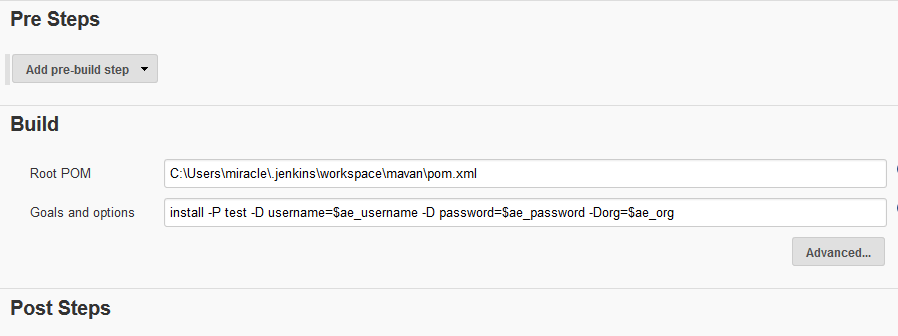
* Now configure the job with the following details
* In the "**Source Code Management** " section click on "Git " radio button and give the following github url.[**https://github.com/papajohns-ds/api-proxy.git**](https://github.com/papajohns-ds/api-proxy.git)



**Fig 9. Git Hub Repository Details**



**Fig 10. Configuring Jenkins with Git Hub Details**



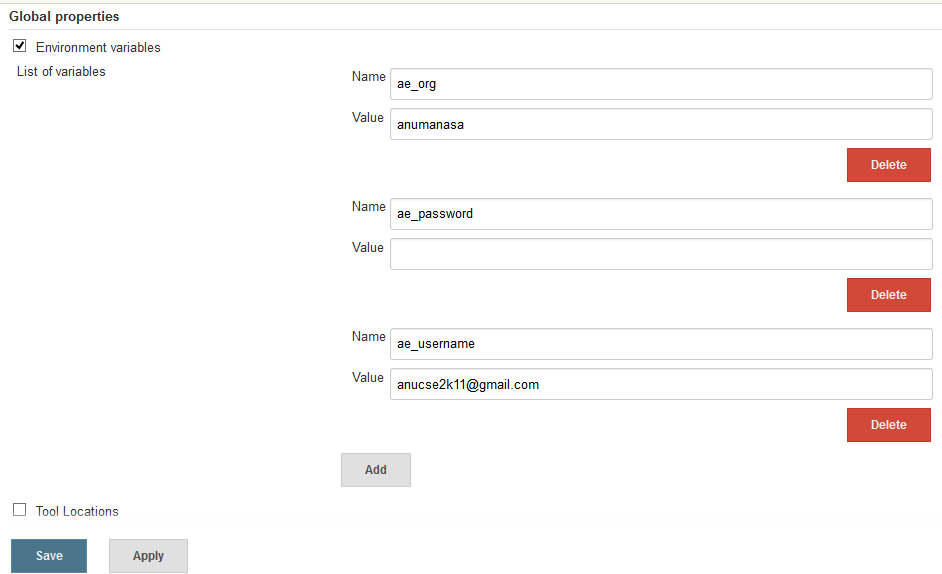
**Fig 11. Configuring pom.xml**

* 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 test -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.



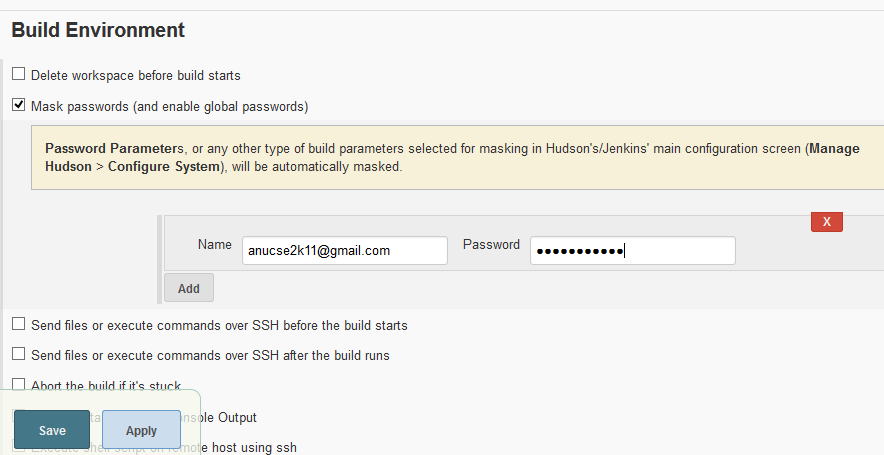
**Fig 12. Adding Credentials of Apigee Edge User**

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



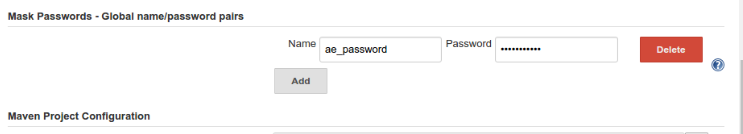
**Fig 13. Adding Mask Password Plugin**

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



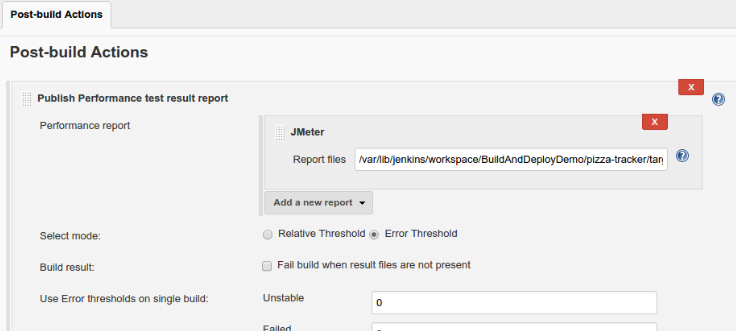
**Fig 14. Configuring Password for masking in Jenkins job.**

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



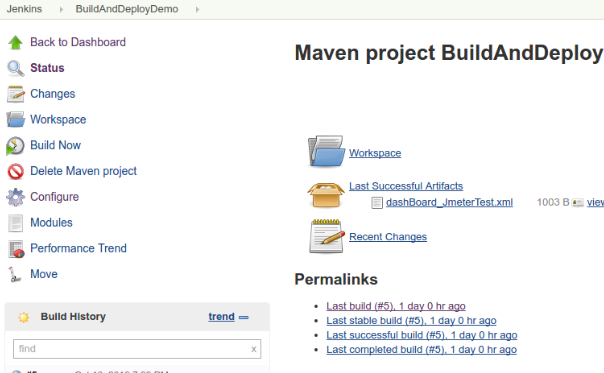
**Fig 15. Configuring the name and password for masking.**

* 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.
* 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 .



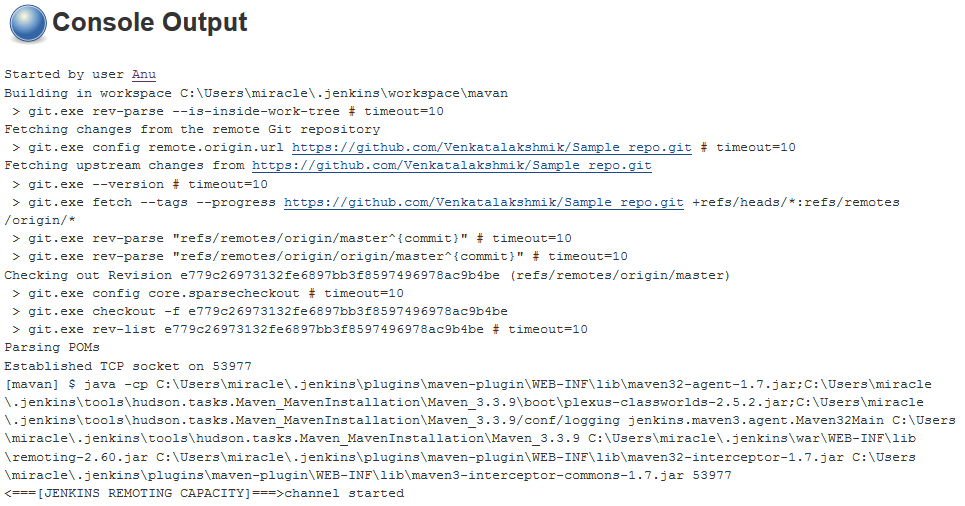
**Fig 16. Specify the path for Result Report**

* Next Build the project and see the console output.

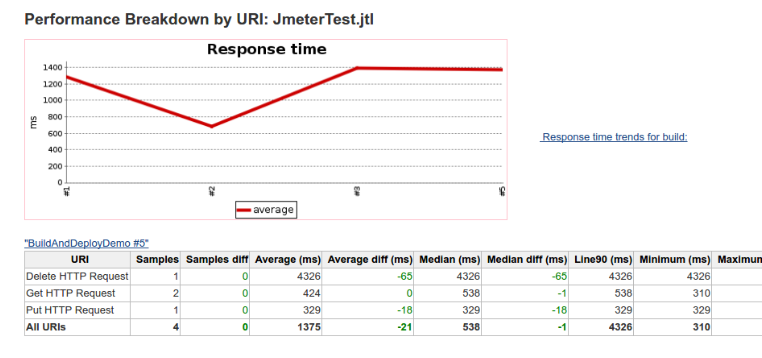


**Fig 18. Build the Job**

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

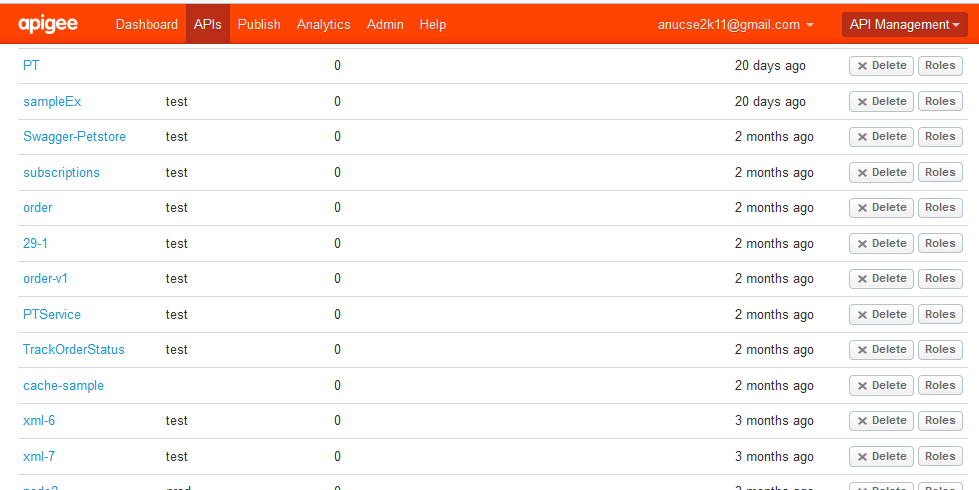


**Fig 19. Console Output**



**Fig 20. Performance Report**

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



**Fig 21. Deployed API Proxy in Apigee Edge**

# 11. Ubuntu Directory Locations:

Following are the files and Directories used for the Jenkins build and deploy process.

/var/lib/jenkins/workspace/BuildandDeploy/pizza-tracker

/var/lib/jenkins/workspace/BuildandDeploy/pizza-tracker/apiproxy

/var/lib/jenkins/workspace/BuildandDeploy/pizza-tracker/apiproxy/policies

/var/lib/jenkins/workspace/BuildandDeploy/pizza-tracker/apiproxy/proxies

/var/lib/jenkins/workspace/BuildandDeploy/pizza-tracker/apiproxy/resources

/var/lib/jenkins/workspace/BuildandDeploy/pizza-tracker/apiproxy/target

/var/lib/jenkins/workspace/BuildandDeploy/pizza-tracker/apiproxy/tests/JmeterTest.jmx

/var/lib/jenkins/workspace/BuildandDeploy/pizza-tracker/apiproxy/tests/order\_test.csv

/var/lib/jenkins/workspace/BuildandDeploy/pizza-tracker/apiproxy/pom.xml

/var/lib/jenkins/workspace/BuildandDeploy/pizza-tracker/target

Jenkins Log file

POM.XML