

Continuous Integration

And

Continuous Deployment

February 15, 2017

**Document Control Information**

**Document Information**

|  |  |
| --- | --- |
| **Name** | CI CD Document |
| **Program Name** | Enterprise Integration |
| **Author** | Ankit Prakash |
| **Version** | 1.0 |
| **Status** | Draft |

**Document Edit History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | Date | Description | Author |
| 1.0 | 02/14/2017 | Initial | P M Arjun |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Support / SME Resources**

|  |  |  |  |
| --- | --- | --- | --- |
| **Team** | Name | Role | Contact (Email / Phone) |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[1. Prerequisites 4](#_Toc474856501)

[2. Alterations To The Parent-Pom And Maven 5](#_Toc474856502)

[3. CI CD Process 8](#_Toc474856503)

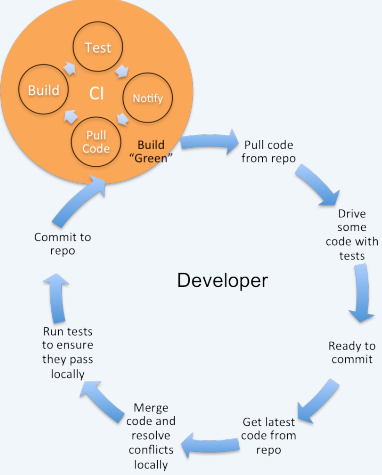
1. Overview

This document explains about the Continuous Integration and Continuous Deployment process and the setup that is required to achieve it

CI is the process of executing the software Build-Deploy-Test (BDT) cycle frequently with minimal manual intervention with the underlying principle as one of constant communication and feedback among team members

**Delivery** builds upon the concept of providing fast, automated feedback on the correctness and production readiness of your application every time there is a change to code, infrastructure, or configuration.

In this document the CI CD process starts from the time a developer commits his source code into a repository and a CI software such as Jenkins takes the project from the repository for each change made to project by the developer for further build-tests and deployment.



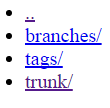
1. Prerequisites
2. You need to have java as well as maven installed, follow the below document for developer machine setup:



1. An instance of Jenkins running on a server, with Jenkins credentials in order to log in into Jenkins.
2. The developer must mavenize the project using parent-pom.xml and pom.xml. For more details on parent-pom refer below document:



1. As the artifact that the build created from the Jenkins server would be stored in a Nexus repository, you must have repository created in nexus with credentials to log in.
2. A secured repository where the developer can commit his code. In this case we are using a SVN repository with authentication required to access the repository.
3. When the developer commits his code into his repository, it is really important that a folder structure is maintained in the repository. In this case we are using SVN version control tool so below given folder structure must be maintained:

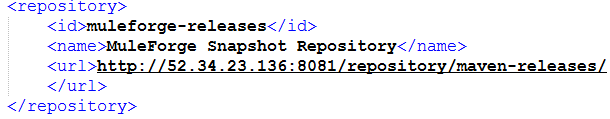


And the developer must commit the code that he is working on currently in the trunk folder.

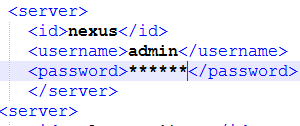
1. The Jenkins must also have all the plugins that are required to take the code from SVN and build the Mavenized code. Plugins such SVN plugin, all the maven plugins, plugins for showing the test results that would be run, email notification plugin In order to notify the developer and team members on the result of the build status.
2. Alterations To The Parent-Pom And Maven

So there are some points and changes that you need be aware of with respect to parent-pom and Maven configurations:

1. Now in case we have created some custom jars and have uploaded it into a repository say into nexus repository and we wish to download the jar from the repository during the build then below configuration has to be added to parent-pom:



So we have uploaded a jar into the repository location given above and specifying the above configuration in parent pom under repository section and specifying the credential details in maven settings.xml, we would be able to download the jar during the build:



Specify the above under servers section in setting.xml in maven (for password check with nexus admin)

1. Now in case we need to make deployment once the build is successful, then we need to provide the deployment details in the parent-pom. Details would include whether the deployment is on-premise or On cloud, the environment- test, dev, prod, the credentials etc. :



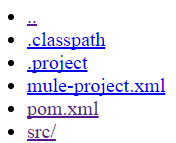
These would be details for On Cloud Deployment. Now all the values are soft coded in parent pom, these details such as worker size, username, password, worker type etc. are provided as Global properties in Jenkins software and also these details can pass from the command prompt

The Global properties section is inside the Jenkins dashboard:

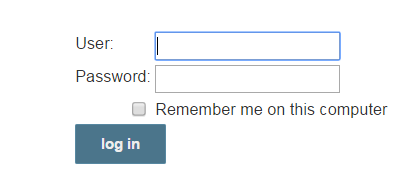
Manage Jenkins-> Configure System-> Global properties.

Similarly we can provide details in parent pom in case of on-premise deployment as well.

1. CI CD Process
2. So let us understand the process for CI CD using a sample project that we have created ‘**StateOfGeorgia’**.
3. So let consider that a Mavenized mule project has been committed by the developer in the SVN repository under the trunk folder, with parent-pom included.



1. Next we need to navigate to our CI software tool Jenkins that is running on our server. Enter your credentials and log in into Jenkins.



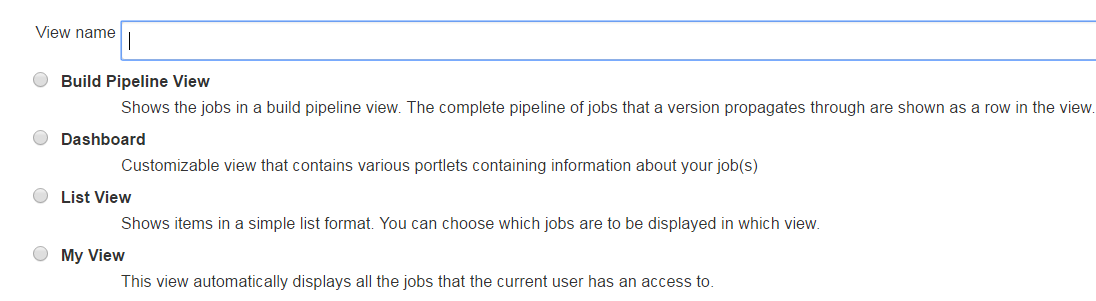
1. Once all the plugins and everything is set up from Jenkins end we can start building our project. In Jenkins in order for a build could happen, we must first create a job for the project. And each project would have three jobs –

Build- 1: To verify whether a regular build is successful for the project, all the dependencies are met and there is a successful build.

Build-2: Once the test cases are added and committed by the developer, we need to check whether the test cases are successful. This build would be to check the test cases.

Build-3: Once test cases are successfully built, then we could deploy the application into on-cloud or on-premise server. This build would be for deploying the application.

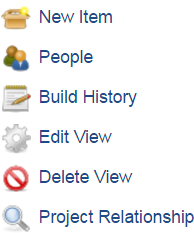
1. As each project would have three builds, we could create our own view for each project. In the Jenkins Dashboard you could click on ‘+’ sign to create your own view. Clicking on it would navigate you to a page where we can provide the name for the view and how and where that view must be displayed.



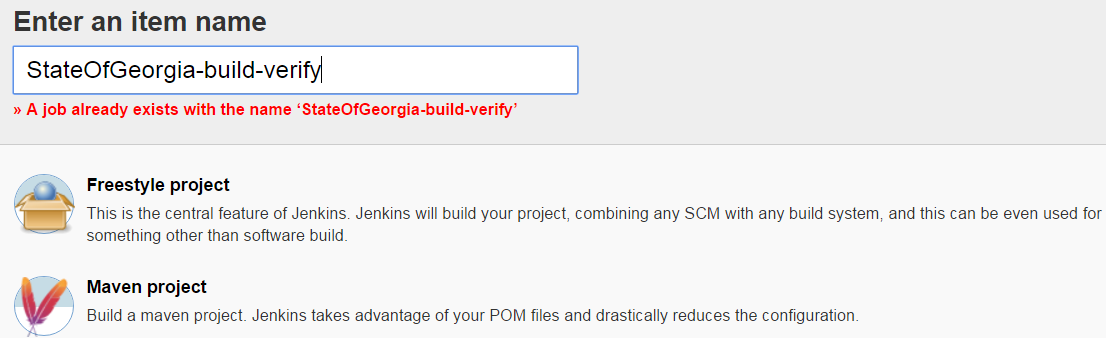
1. So for our ‘**StateOfGeorgia’** we have created a view, the view will be displayed on the Jenkins dashboard.



1. Let us create the first job to build the project for the first time. Enter your view and click on ‘New Item’ from the panel that is on the left hand side of the dashboard.



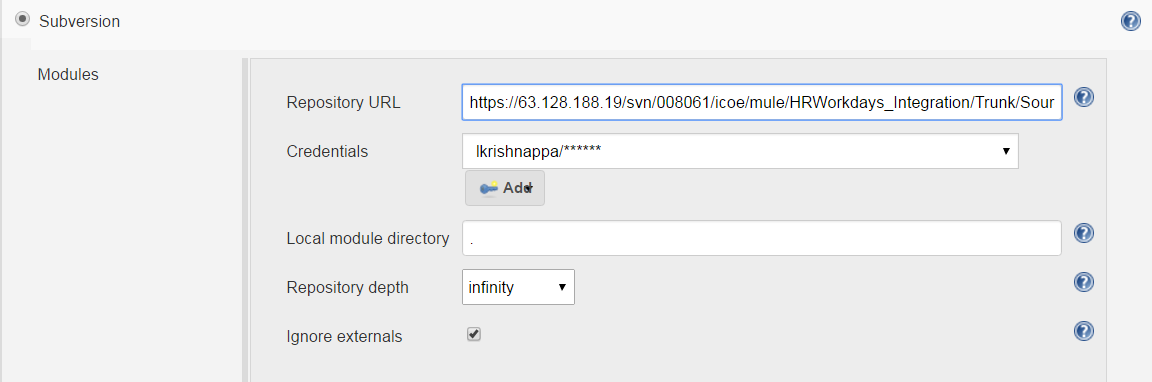
1. You would be navigated to a page where we need to specify the name of the project and the type of project.



We need to provide the name and select ‘Maven Project’ as the project we are going to build is a maven project and click on ‘OK’.

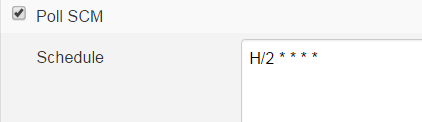
1. Then we would be navigated to the page where we would be able to configure our build.

So the first configuration that we need to make is to provide the SVN path for our project and provide credentials for SVN under ‘Source Code Management’ section.



So the path would point till the trunk folder and using ‘Add’ we can add the credentials.

1. Next in ‘Build Trigger’ section check the checkbox ‘POLL SCM’.

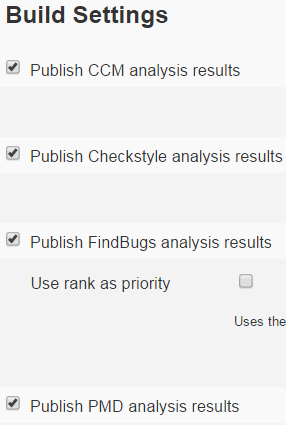


This would check repository for any changes every 2 mins.

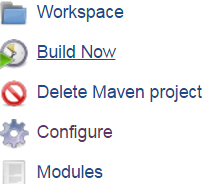
1. Next we move on to the ‘Build Section’. Here in Goals and options we can provide various arguments such as encryption keys, environment where application has to be deployed, whether to skip unit testing, whether a zip file needs to be created on build etc.

clean verify pmd:pmd dependency:analyze-report -Ptestwithcoverage - DApplicationCoverageRequired=90 -Dkey=\*\*\*\*\*\* dependency:tree -DskipTests

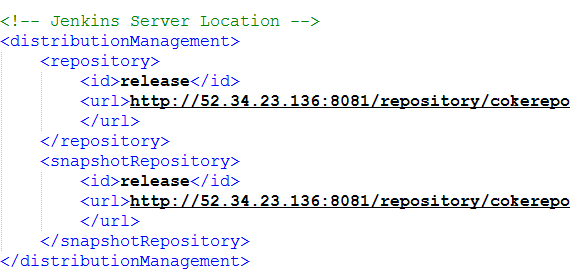
1. In the ‘Build Settings’ section we need to check the checkboxes as shown below in order to get various analysis reports.



1. And at the end we need to check the check box Email-notification, using which we would be able to send email notifications to developers regarding the status of the build. You can click on ‘Save’ and click on ‘Build Now’ from the panel on the left hand side.

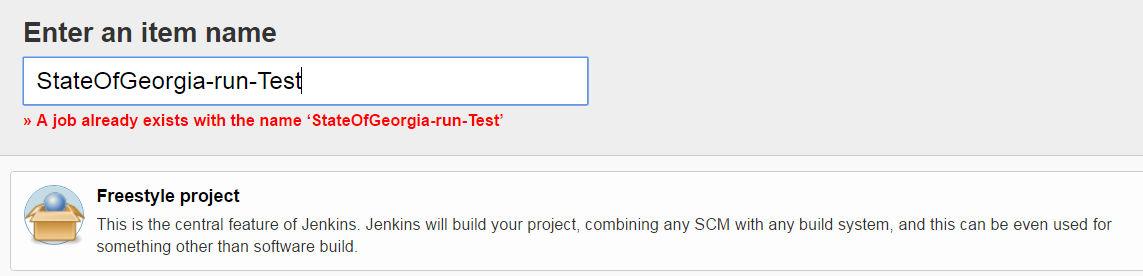


1. Now if the build is successful then it would send the artifact that is created during the build into a repository in nexus. This is achieved by specifying the following details in the parent pom:

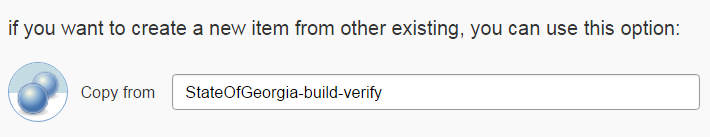


The developer will also be notified with the email notification with success of the build.

1. So in this build we are skipping unit testing. But for the next job and next build we need to run the unit tests. Similarly we will create another job:



But now we will create a freestyle project and extract all the configurations and properties from the previous job that we created so that we need not configure all the build configurations again.

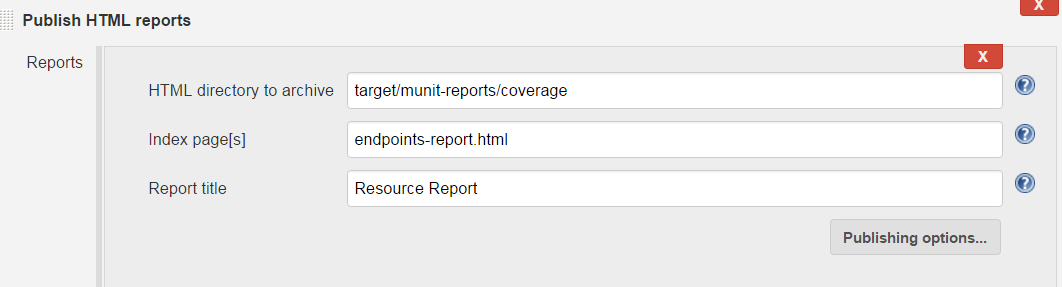


1. Now in this job the only changes that we need to make is with respect to the arguments that we provide:

clean test -fn --fail-never -P testwithcoverage -Denv=dev -Dkey=\*\*\*\*\*\* -DApplicationCoverageRequired=60

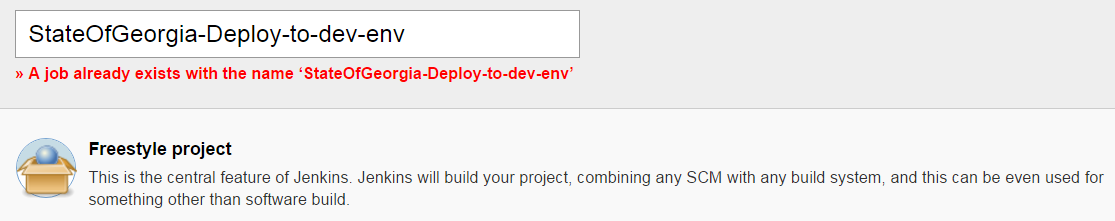
This would capture the code coverage.

1. We also need to publish the test results report this can be done by adding the below content in publish html reports in Post Build Actions section:



So the above build would provide us Munit reports, resource reports, test result graph etc.

1. So once the test result are good and build is successful then we would create the next build to deploy the application into On Cloud or On-premise servers. Similarly we will create a Freestyle project and copy the configuration from previous jobs.

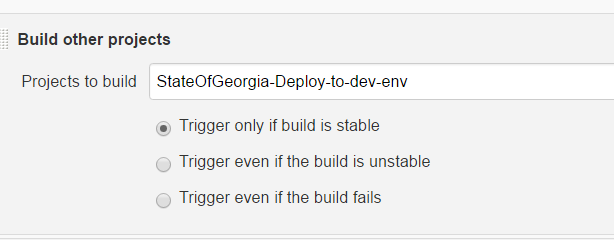


1. Even for this job the only change we need to make is the arguments that we pass:

clean deploy -Pdeployment -DprojectName=stateofgeorgia-dev -DmuleVersion=${muleVersion} -Dusername=${username} -Dpassword=${password} -DworkerType=${workerType} -Dworkers=${workers} -Dregion=${region} -Denvironment=DEV -Denv=dev -Dkey=coke2016wdsitxcsz –DskipTests

So the above arguments would deploy the application if the build is successful.

1. Now we can also make all the three jobs run successively one after the other. So if build verify is successful then it runs the test build and if that is successful then it runs the deploy build and deploy the application on to could or on premise servers. This can be done adding the build other projects option in ‘Post Build Actions’ in the build configurations for each job.



So if the current build is successful then it would build the next job that is specified.