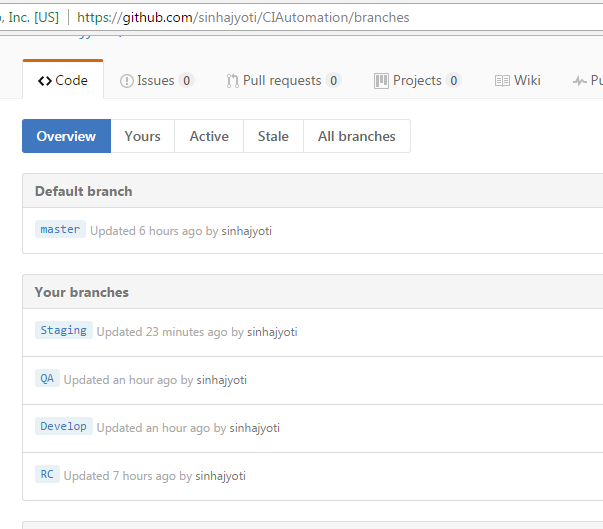
Setting up Jenkins CI

## Download and Installation:

1. Go to following URL:<https://jenkins.io/index.html> , download the TLS version of download available for windows.
2. Run the downloaded .msi installer and follow the default instructions
3. Once the installation is ‘finished’, it automatically tries to open the browser at following url (unless you change the port in Jenkins.xml): <http://localhost:8080>
4. Provide the Admin password provided in the file in Jenkins installation folder
5. Install the standard plug-ins(that should include Github integration, github API plugin)
6. Once, the step 5 is completed, it asks for setting up the ‘Admin’ user
7. Provide the information required for creating the Admin user. Click on ‘Save and Continue’.

## Repository configuration (skip this section if you already have a repo configured):

1. Login to your Github account
2. Create a new repository, for example, *CIAutomation*
3. Ensure you have configured your sample code branches for example: Develop, QA, Staging and RC as in picture below:

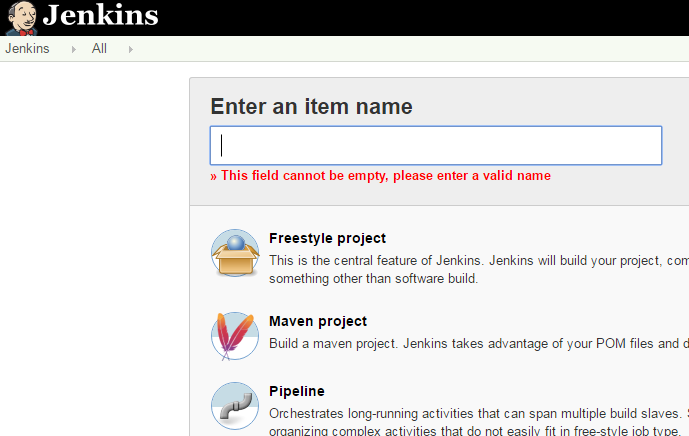


## Jenkins Service Configuration:

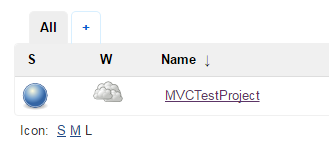
1. Please go to *Control Panel->System and Security->Administrative Tools-> Services*
2. Locate the Service *Jenkins* -> right click -> *Stop*
3. Right click on *Jenkins, -> Properties -> tab “Log on”-> This Account*
4. Enter the logged in user credential and hit OK
5. Locate the Service *Jenkins* -> right click -> *Start*

## Jenkins Project Configuration:

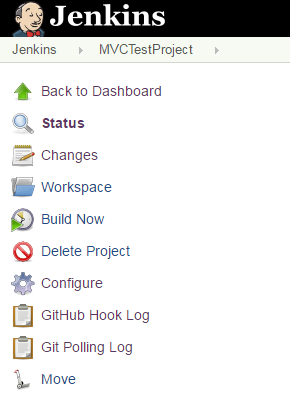
1. Enter the Jenkins URL(it should be <http://localhost:8080> unless you have changed it to different port) -> *Create Job*
2. Enter a Project name -> select “Freestyle Project” -> hit “OK”



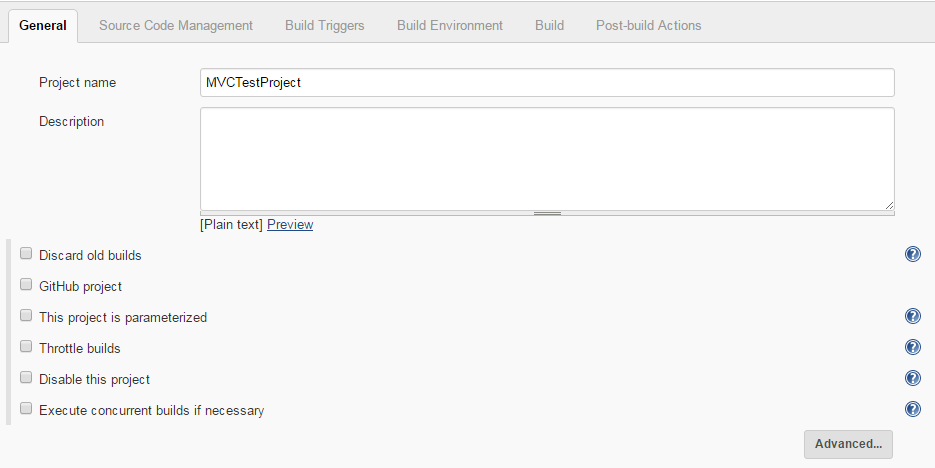
1. Project shall be listed in Dashboard



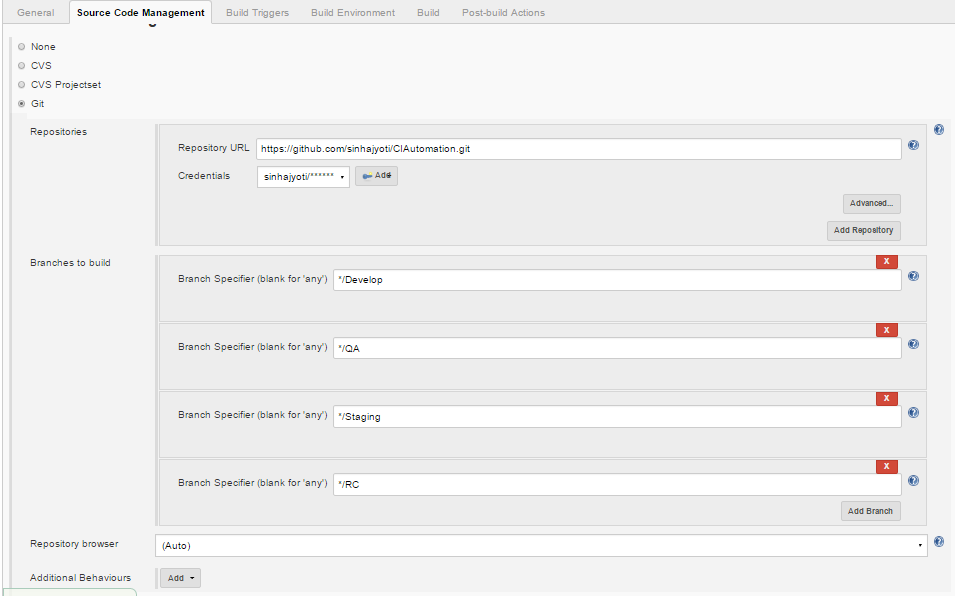
1. Open the Project, hit side menu item ‘Configure’



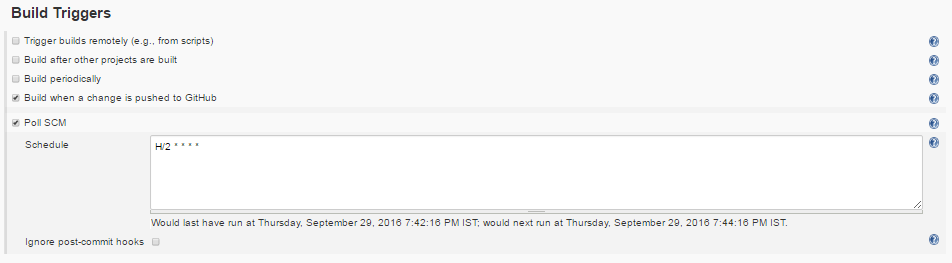
1. General section – no changes



1. Source Code Management section – configure the git repo(or the repo you already have) as following:



1. Build Triggers section – choose the options as following:

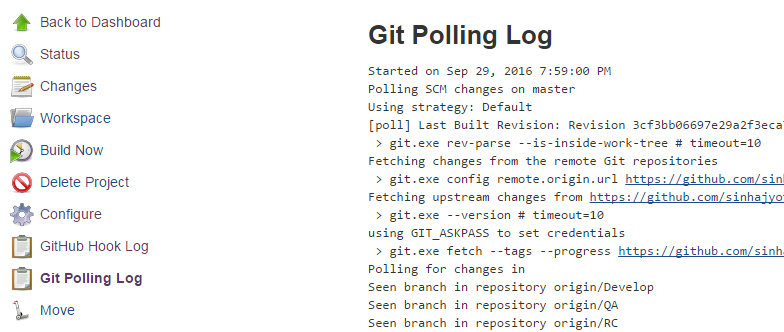


1. Build Section: hit “Add build step” and then select “MSbuild to build ….”
2. Hit “SAVE”

## Verify Git Polling in Jenkins:

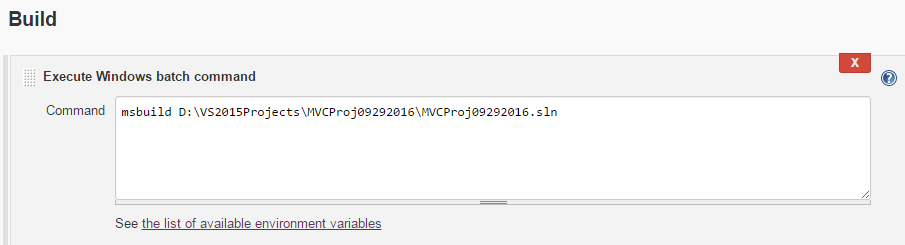
1. Go to Jenkins Dashboard -> click on the Project created
2. In the left hand side menu, hit “Git Polling Log”

If it shows polling happening (per configuration above, polling shall happen every minute), then configuration is all good



## MSBuild Configuration:

1. Ensure you have added the MSBuild.exe path to System PATH environment variable
2. Hit the ‘Build’ section (or tab at top), ‘add a build step’ - > select ‘Execute Windows batch command’
3. In the command area provided, please make following entry(replace the sln name with appropriate solution filename:



This is the bare-minimum command line for enabling build from command-line. You can enhance and add different *targets* and other attributes as required.

***Note: in case, when you are using VS 2013 and few of your projects are failing to compile, that potentially be an issue of using VS2012 template instead of VS2013. In order to fix that, please set the compatibility flag (/p:) to msbuild as following:***

*Msbuild /p:VisualStudioVersion=12.0 D:\VS2015Projects\MVCProj09292016\MVCProj09292016.sln*

## Code Metrics configuration:

1. Make sure you have located metrics.exe on your computer, If metrics.exe is NOT found on the Jenkins server, download the metrics powertool from following location:

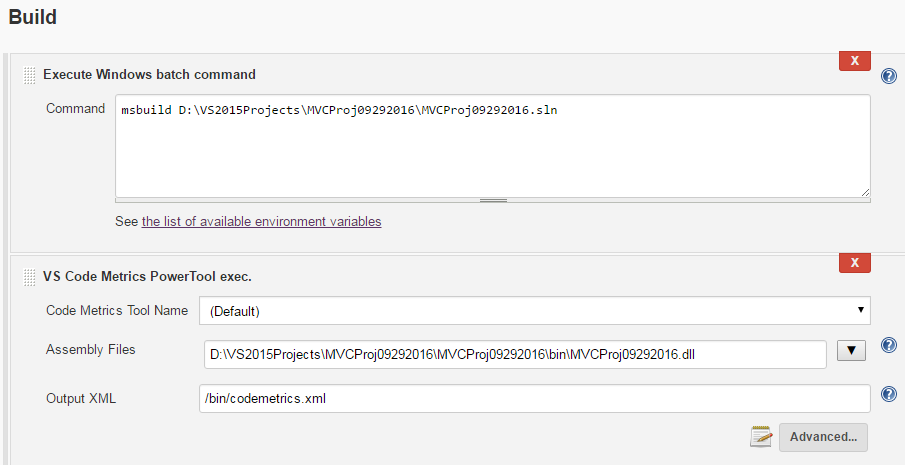
For VS 2015,

<https://www.microsoft.com/en-in/download/confirmation.aspx?id=38196>

For VS 2013,

<https://www.microsoft.com/en-in/download/details.aspx?id=41647>

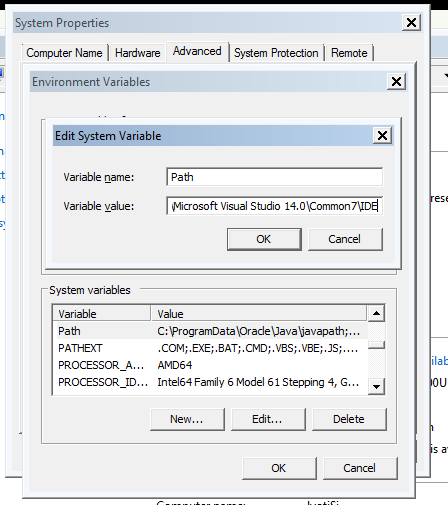
1. Install the powertool and add the metrics.exe to system PATH environment variable
2. In ‘Build’ section, ‘add build steps’ -> ‘VS Code Metrics PowerTool exec.’
3. Make the configurations as following:



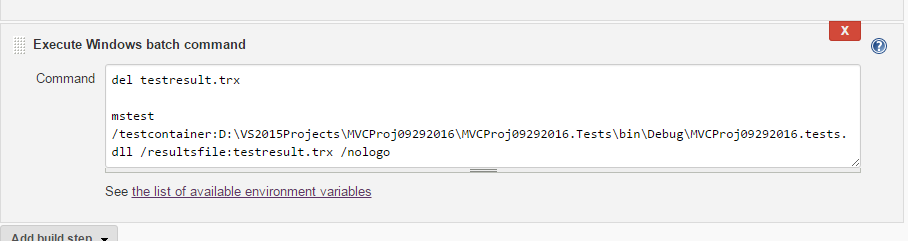
* In the assembly files, list out the names of the assemblies up for evaluation with semicolon delimiter
* In the output XML provide the name of output XML with path relative to Jenkins workspace root location

## MSTest configuration:

1. Ensure that MSTest.exe path is added to PATH in system advanced settings



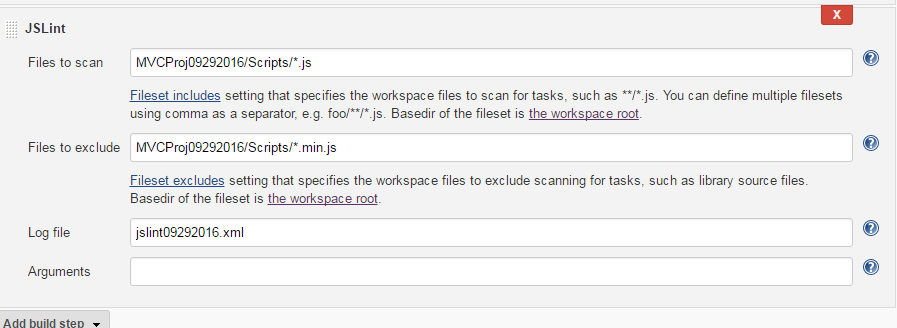
1. In the ‘Build’ section -> ‘Add build step’ – ‘Execute Windows Batch Command’
2. In the command area provide the command-line for MSTest execution as following:



* **del \*.trx**: testresult.trx is the output file of MStest. Since MStest will fail if there is an existing .trx file, make sure you delete the trx file before executing the MStest afresh. As an alternative you can also rename the older file before running the new test
* MStest command-line: provide the MSTest command-line with following:
  + Provide the DLL name from your MStest project. Make sure dll is not the project dll, it should be TEST project compiled <**proj>.tests.dll**
* Path of \*.trx has got to be relative to Jenkins workspace root

**JSLint Configuration:**

1. Install the *JSLint* plug-in in Jenkins – required for JSlint analysis
2. Install the *Checkstyle* plugin in Jenkins – required for publishing the JSLint analysis
3. Go to ‘build’ section -> ‘Add build step’ -> ‘JSLint’
4. Provide the configuration as following:



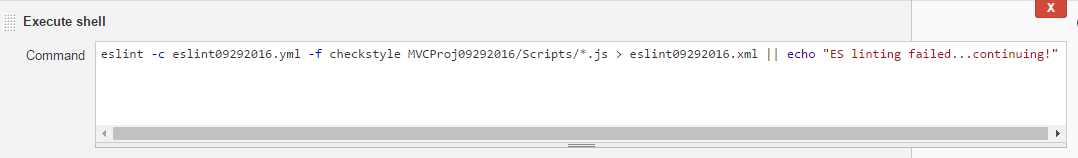
Log file location should be relative to the workspace root

## ESLint Configuration:

1. Install the ESLint using following npm command on command prompt:



1. There is no ESLint plug-in currently available for Jenkins, so it requires to be run through Shell command only.
2. ‘Add build step’ -> ‘Execute Shell’
3. Enter following command line in shell command editor:



That command line breaks down as follows:

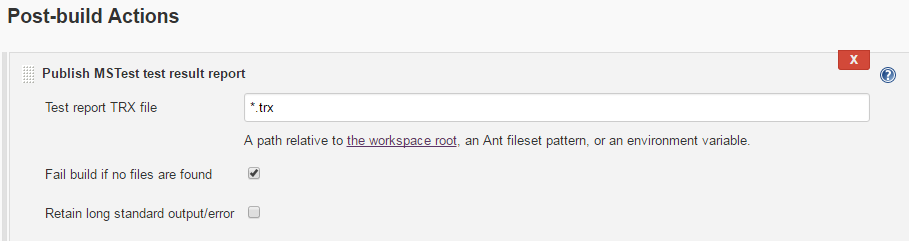
* *eslint* calls the linter
* *-c eslint09292016.yml* points it to our custom configuration file
* *-f checkstyle* outputs the results in checkstyle format. This can be other formats like jslint, junit, or tap, but I found the checkstyle plugin to be to my liking.
* file paths indicate what files should be linted.
* *> eslint09292016.xml* is the command line to pipe the command output into a file.
* *|| echo* is, as with last time, a way to ensure that the build does not fail when linting fails. Again, the reporting plugin will take care of marking the build as unstable when the linting fails. Without this, linting errors will prevent Jenkins from moving on to the next build step

*Make sure config file (eslint09292016.yml) mentioned as –c is copied into workspace root location.*



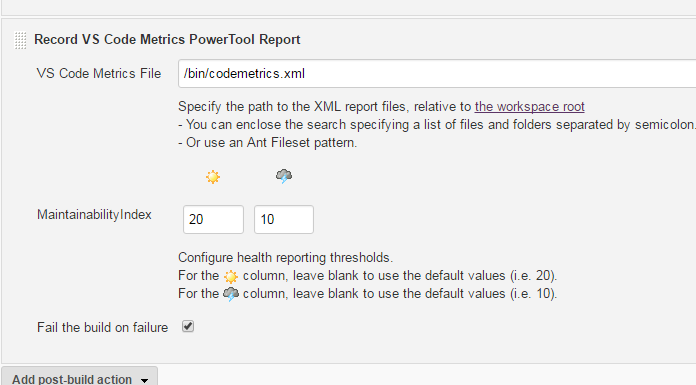
## Configure Post-build events:

1. **MSTest Result Publish**: Hit the ‘Post-build Actions’ – > ‘Add post-build Actions’ – Select ‘Publish MStest test result report’. Provide the config as below:



Make sure .*trx* file path is provided relative to the Jenkins workspace root

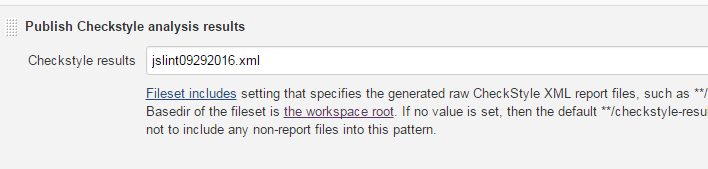
1. **Code Metrics Result publish**: Hit again the button labeled as ‘Add post-build Actions’ -> ‘Record VS code metrics PowerTool Report’. Provide the config as below:



Ensure that VS Code metrics File location is relative to the Jenkins workspace path and is same as the one created in step listed above as ‘*Code metrics configuration*’.

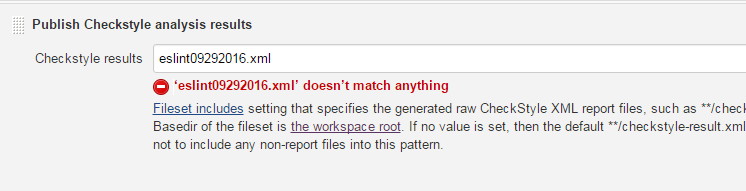
1. **JS Lint result publish**: ’Add Post-build Actions’ -> ‘publish Checkstyle analysis results’

Provide the config as following:



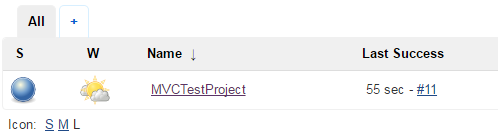
Make sure the checklist result has same file mentioned as the one mentioned above in JSLint build step as log file

1. **ESLint results publish:** It also uses Checkstyle analysis results. ’Add Post-build Actions’ -> ‘publish Checkstyle analysis results’. Provide the config as following:

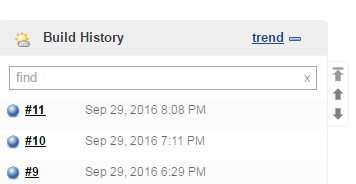


## Executing the CI build:

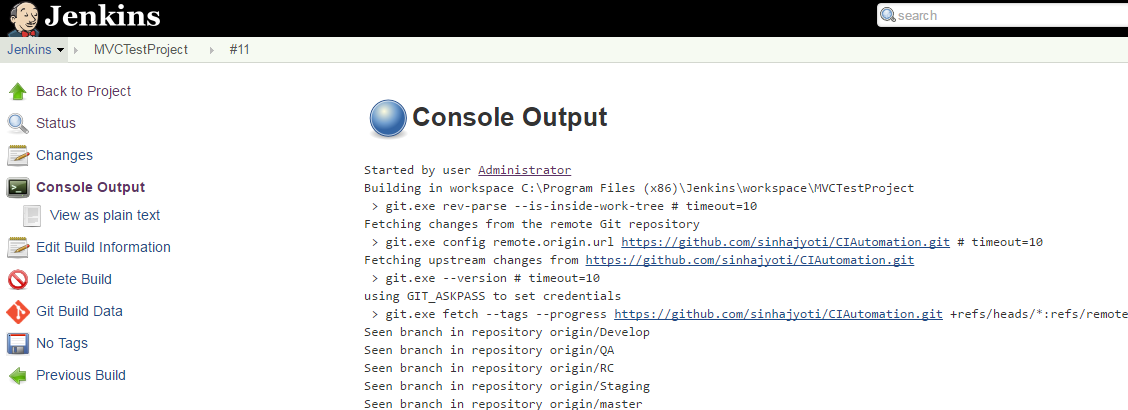
1. In the MVC code, make a change and commit->Push to ‘Develop’ branch
2. It should have triggered a build(build trigger configuration was “*Build when a change is pushed to Github*”
3. In the Jenkins dashboard, refresh the page and last success build shall be listed



1. To see the build log, hit the Project in the dashboard
2. In the left hand side navigation, “Build History” shall be listed



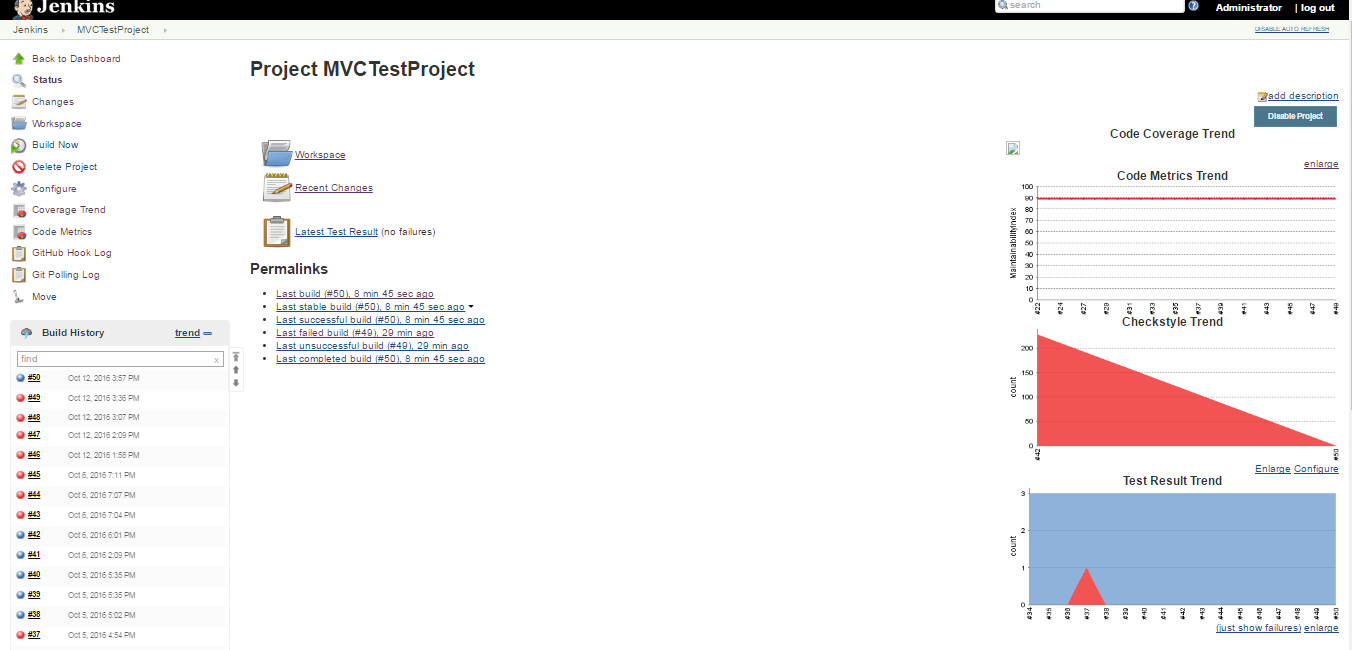
1. Click on the build number
2. Hit **Console Output,**



1. To trigger another build, in Git repo, create a PULL request using QA branch as base and from ‘Develop’
2. Merge the changes, it should trigger the step 2 listed above.

**Running the Build Reports:**

1. Click on the Project hyperlink in Jenkins dashboard
2. It would show the ‘Status’ as of latest build. Screen should look something like following:



1. To drill-down to see the details, you can click on *build number*, *Coverage trend* and *Code metrics*, *Checkstyle Warning* etc. as required.

---