LTI
Let's Solve

Whitepaper

Continuous Integration Using Jenkins

Author: Sapana Kejadiwal

A Larsen & Toubro Group Company

Contents

1 Abstract

In a software development project, it is a natural process for several developers and testers to work on different modules. Each developer or tester will be assigned a different module, so it is extremely important to test and build the modules to ensure they work seamlessly with each other, with zero bug interference.

Frequent changes in code? Multiple code check-ins? Manual testing? Multiple integrations? Continuous Integration (CI) is here to rescue.

CI is a practice that requires developers to integrate their code into a shared repository frequently, which leads to multiple integrations per day by different developers. The goal here is to ensure that every change in the code is built and tested immediately by an automated build, which in turn gives an instant picture if something breaks.

This white paper will explain how we achieve continuous integration & automation, using Selenium as test automation tool, Jenkins as CI tool, GitHub as a shared repository, and JIRA as Defect management tool.

Major takeaways from this paper would be adoption to part of Devops, its seamless integration, and automation process and its benefits.

2 Introduction

Over the last two decades, software industries are following a traditional way of delivering frequent changes in software development process, which has its own limitations/problems like missing on deadlines, more time to market, poor quality of product, etc.

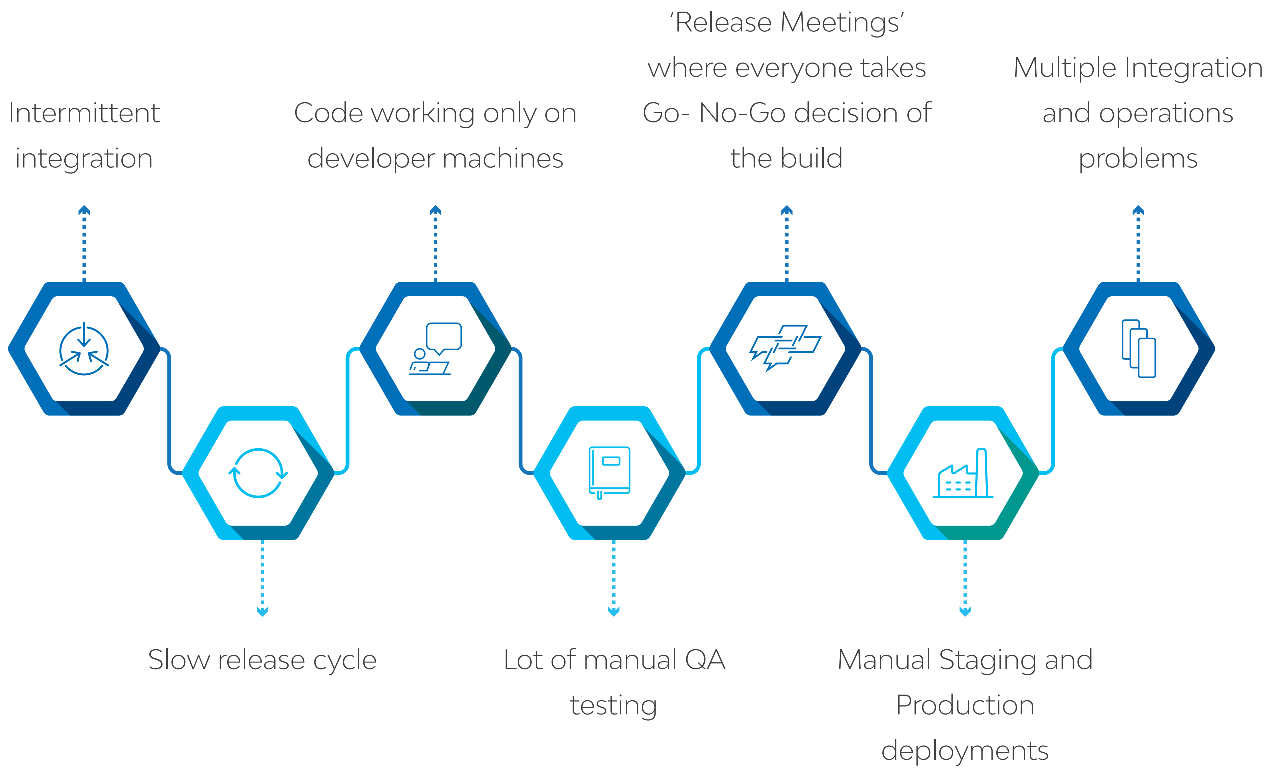
With technology innovations continuously growing, overcoming some of these limitations, software industries have caught up with the latest digital trends. One of which is Continuous Integration, which is an emerging game changer and surpasses the challenges faced with traditional software development process.

Continuous Integration (CI) primarily deals with the automation of development processes, and build/code integration/test automation. CI becoming such a well-accepted development practice, has also proved how much more productive the development process can be with automation. It has become the catalyst for continuous delivery (CI) practices - extending the automation through the entire software pipeline, through staging and into production.

This paper is intended for beginners to get the overall understanding of CI, Continuous Deployment and Continuous Delivery, how small/large enterprises leverage CI/CD, and how the road to Continuous Integration/CD is paved with its own set of challenges.

3 Problem Statement

Traditional way of delivering rapid changes in software development projects consists of:


Intermittent integration
Slow release cycle
Code working only on developer machines
Lot of manual QA testing
'Release Meetings' where everyone takes Go- No-Go decision of the build
Manual Staging and Production deployments
Multiple Integration and operations problems

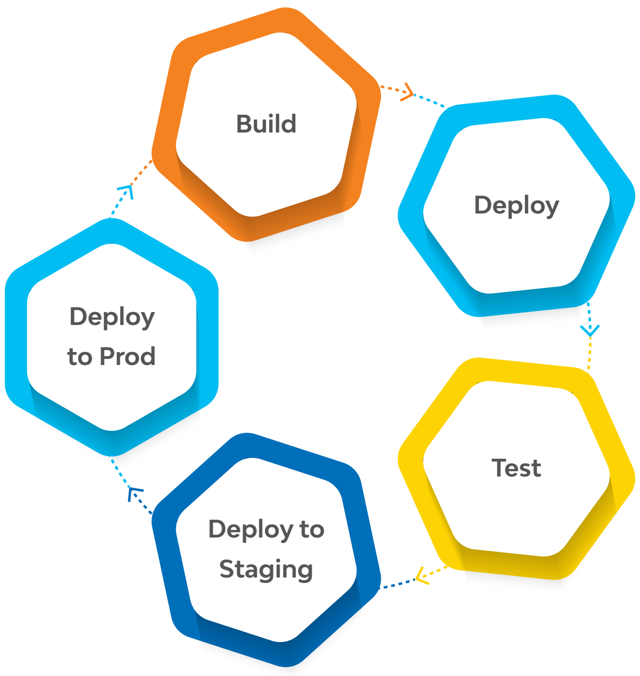
All these results in poor quality of the product, more time to market, more cost, less customer satisfaction, and ultimately adding less value to business.

4 How to solve this?

Continuous Integration is here to salvage from all these problems. CI is way to increase code quality by continuously integrating the code of different developers on central repository. CI servers trigger build and compilation processes automatically, and notify as soon as something goes wrong. So our tests automatically get executed as and when there is a change in the code. We can have CI server automatically deploying our code to staging and production, if all the tests in given branch are green. This way we can achieve faster build execution, early detection of bugs, increased code quality and faster time-to-market.

5 Overview of Continuous Integration

“Continuous Integration” or “CI” is a concept, which has been taken as a standard or a parameter nowadays, within our IT services and development. The concept of continuous integration means automating the overall deployment process for an application, after a code has been committed, so as to identify and address the pain points before they become serious issues. So each time a product code is developed, it goes through an automatic CI process. There are several tools for Continuous Integration to help trigger these builds and tests.



6 Continuous Integration Practices

To make up effective CI and to work above processes smoothly, we have to have some standard practices to be followed from the beginning of our project.

- Maintain a single-source repository
- Make your build self-testing
- Automate the build
- Everyone commits to the mainline every day
- Every commit should build the mainline on an integration machine
- Fix broken builds immediately
- Keep the build fast
- Make it transparent
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Automate deployment

7 Continuous Testing

Continuous Testing is integral part of CI process. Each time some code changes happen and checked in into a shared repository, an automated test suite run, which is developed using any automation testing tool like Selenium. This complete process can be achieved using CI tool like Jenkins.

8 Selenium – An Automation Testing Tool

In order to create automated testing in a Continuous Integration (CI) environment, with the objective of creating a Continuous Testing (CT) flow, we need to familiarize with automation and testing tools. There are lots of tools and techniques that can be integrated in a CI environment. One of the widely used open source tools is Selenium.

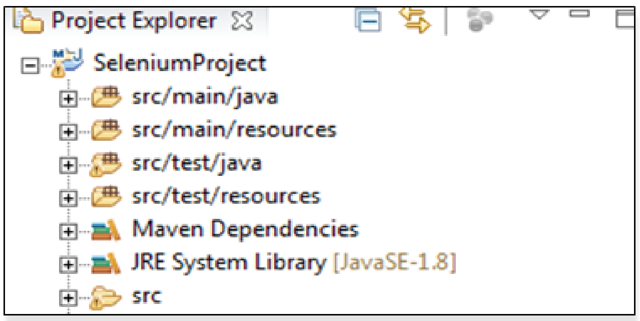
Selenium automates browsers support different languages like Java, Python, Ruby, php, perl and javascript with different browsers and operating systems. We use Selenium in conjunction with continuous integration, to ensure web applications are automatically tested via scripts as opposed to manually. Selenium can be used with TestNG, which is a rich testing framework and used for reporting, parallel testing, grouping test of cases, etc.

9 Selenium and Maven Integration

Apache Maven is a software project management and comprehension tool. It is formally known as a Build tool. Based on the concept of a Project Object Model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information that is called Central repository. Maven has its own repository, where it keeps all plugin, jars, etc. in commonplace in .m2 repository.

Relation with Selenium:

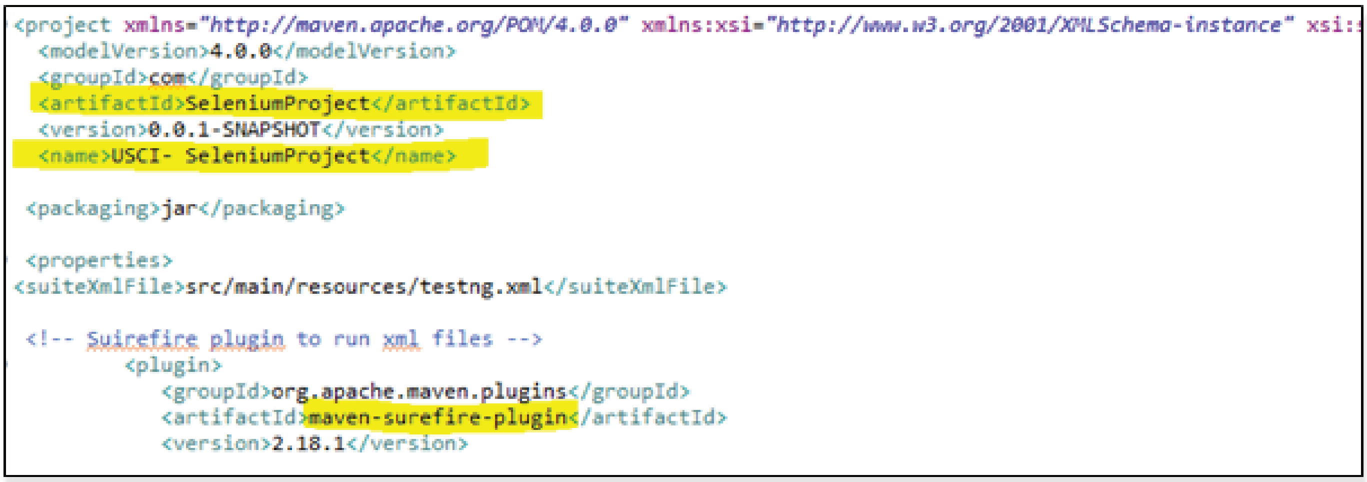
Maven project can be created for writing script and creating dependency-using POM.xml once dependency is set, Maven will download all the dependent jar files automatically. Maven also automates process of creating initial project structure.



Key components of pom.xml file:

Project configuration details

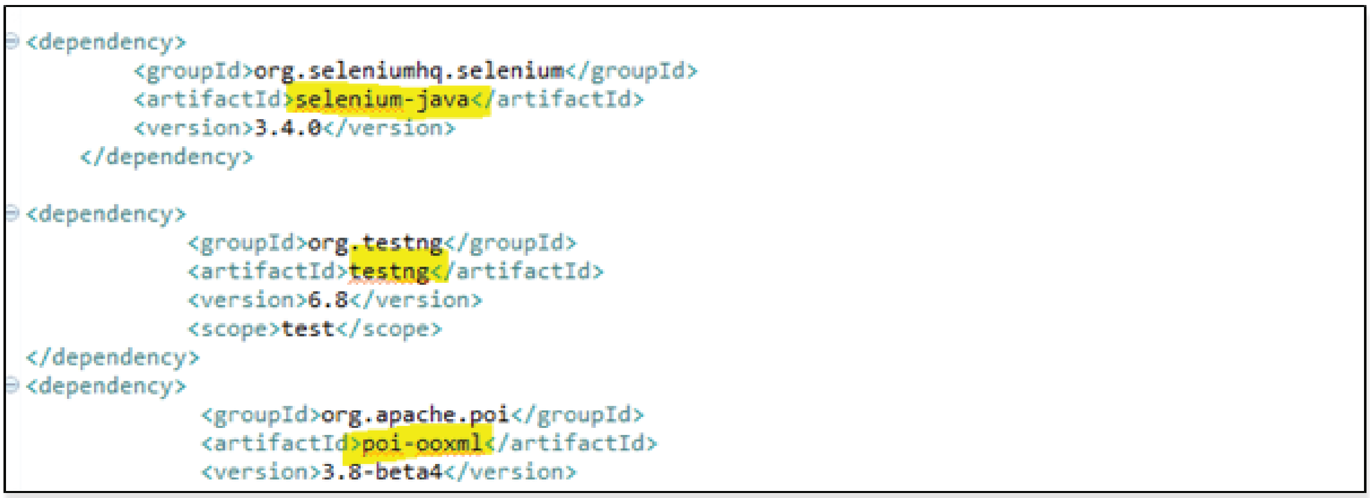
Information such as name of the project, artifact id, group id, .xml file path, plugin details like Surefire plugin, fortysix plugin to send an automatic email after build success, etc.is mentioned in pom.xml file as depicted below:



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
  <modelVersion>4.0.0</modelVersion>
  <groupId>com</groupId>
  <artifactId>SeleniumProject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>USCI- SeleniumProject</name>

 <packaging>jar</packaging>

 <properties>
<suiteXmlFile>src/main/resources/testng.xml</suiteXmlFile>

 <!-- Suirefire plugin to run xml files -->
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.18.1</version>
```

Dependencies

Dependencies are the libararies, which are required by the project. For example Log4j jars, Apache Poi jars, Selenium Jars, etc. Dependencies are mentioned in the pom.xml like this.



```
<dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>3.4.0</version>
    </dependency>

<dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.8</version>
            <scope>test</scope>
</dependency>
<dependency>
             <groupId>org.apache.poi</groupId>
             <artifactId>poi-ooxml</artifactId>
             <version>3.8-beta4</version>
```

10 Jenkins: Continuous Integration Tool

Jenkins is a leading open source continuous integration server built, with Java. For the last few years, it holds the top position in DevOps toolchains due to being free, open source and modular. It is used to build and test software projects continuously, making it easier to integrate changes to the project. It can execute Apache Ant and Apache Maven-based projects, as well as arbitrary shell scripts and Windows batch commands. It provides hundreds of plugins to support building and testing any project, virtually.

Advantages of Jenkins

- **Continuous Integration and Continuous Delivery**

As an extensible automation server, Jenkins can be used as a simple CI server or turned into the continuous delivery hub for any project.

- **Easy installation**

Jenkins is a self-contained Java-based program, ready to run out-of-the-box, with packages for Windows, Mac OS X and other Unix-like operating systems.

- **Easy configuration**

Jenkins can be easily set up and configured via its web interface, which includes on-the-fly error checks and built-in help.

- **Plugins and Extensible**

With hundreds of plugins in the Update Center, Jenkins integrates with practically every tool in the continuous integration and continuous delivery tool chain.

- **Distributed**

Jenkins can easily distribute work across multiple machines, helping drive builds, tests and deployments across multiple platforms faster.

Because of all above advantages of Jenkins, it is preferred over other CI tools, which are mentioned below.

Advantages of Jenkins

Apart from Jenkins, there are several tools for Continuous Integration, and most of them are open source.

- Cruise Control
- Bamboo
- Buildbot
- Travis CI

11 GIT: Version Control System

Git is the most widely used modern version control system in the world today, which allows multiple persons to safely work on the same project, without hampering other team members. Git is free and open source designed to handle everything from small to very large projects, with speed and efficiency.

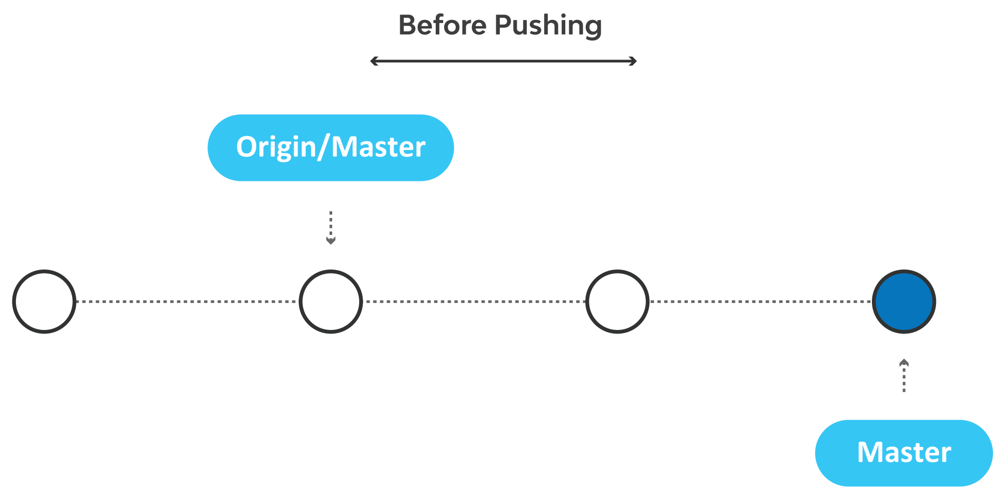
It is distributed and disconnected i.e. every checkout is actually a clone of master repository and every developer has its own repository with its own branches and own commits. We can clone working the copy of a local repository from Git server, which can also add and commit the test scripts that are developed locally and push our changes to the Git.

Git is also used by the Ops team as a version control for automation to store scripts, tools and software configuration.

The most important concept of Git is Branching and Merging. Developers can work on their own feature branch without disturbing others work. Branches are very easy to create and merge back. Since these branches are local, these can be used all the time.

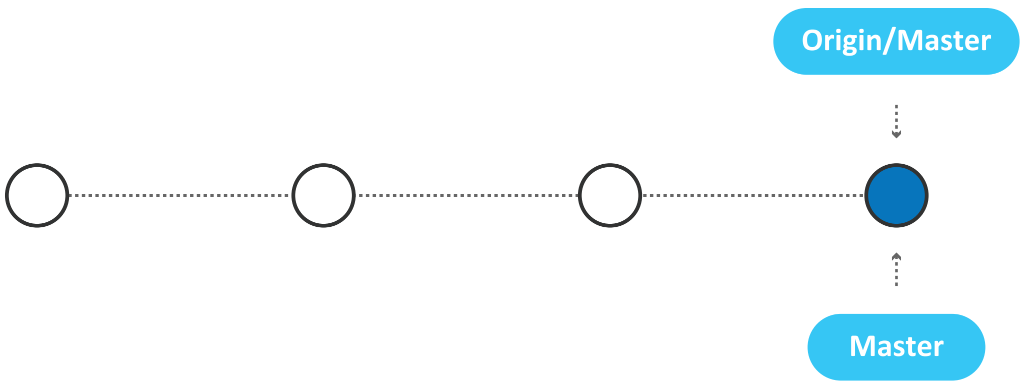
Pictorial representation of Git Usage:

Developers make some local changes and commit these changes locally using 'git commit' command.



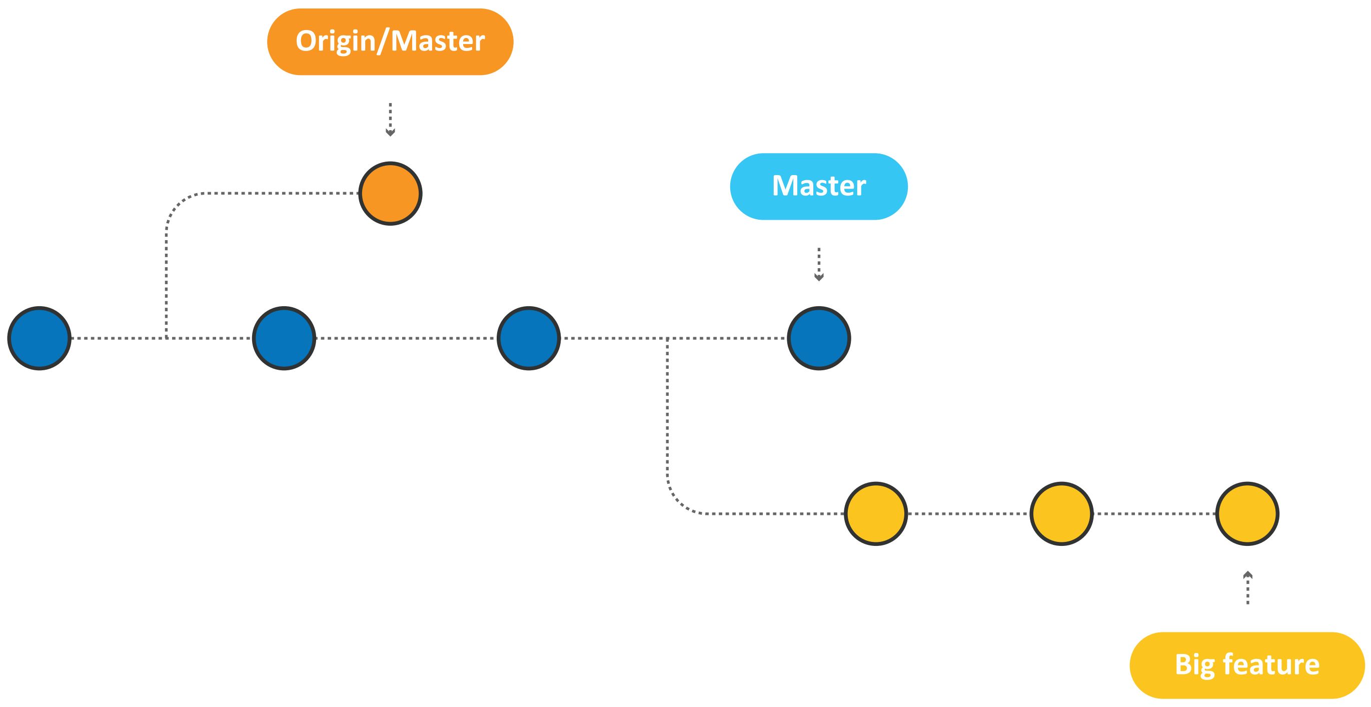
These local changes can be shared with the team and sent to the central repository or master branch, using 'git push' command as depicted in the below fig. Another team member can now retrieve the changes using 'git pull' command.

After Pushing



Branches are the widely used feature in Git. Developers work on their own feature branch so that they don't disturb anyone else who is also implementing some features. As shown in the given diagram Master branch is in blue, a little feature branch in purple and big feature branch in green. At some point, these branches are merged to master branch using 'git merge' command.

Branching and Merging



Other Version Control Tools

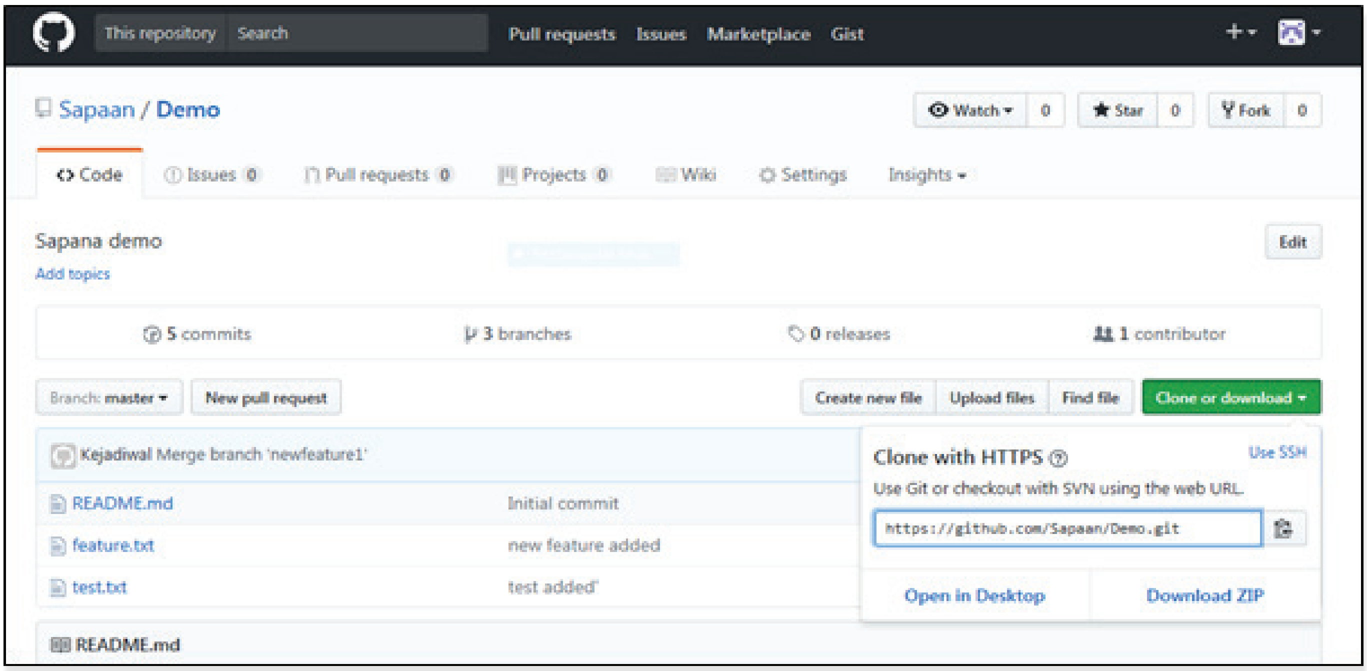
- BitBucket
- SVN
- Mercurial
- Bazaar

12 How Does Continuous Integration Work?

Here CI practices and usage of Jenkins is explained in detail to achieve effective and smooth continuous integration.

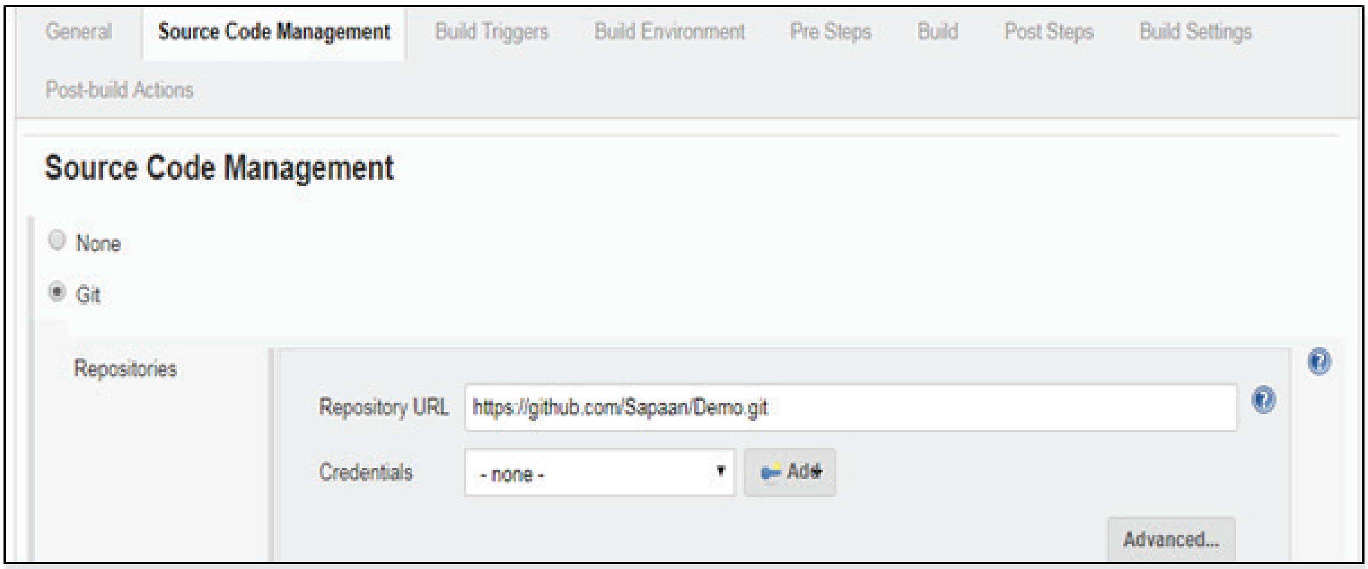
Maintain a Code Repository

Software development project involves multiple developers constantly working and pushing code files, that need to be orchestrated together to build a product. The best practice is to keep a revision control system or tool in place that helps the team to keep the latest and clean code in the repository at any point in the development process. Below is the snapshot of Github, where codebase is placed.



Automate the Build

The key aspect is to avoid keeping everything in the automated build. The build automation ensures that the development team only works on the latest source code from the repository, and compiles it every time to build the final product. We make Jenkins to periodically pull the project's source code from the remote Git Server by installing GitHub plugin, and by specifying the source code path of project in Source code management section of Jenkins.


General | Source Code Management | Build Triggers | Build Environment | Pre Steps | Build | Post Steps | Build Settings | Post-build Actions

Source Code Management

- None
- [✓ Git]

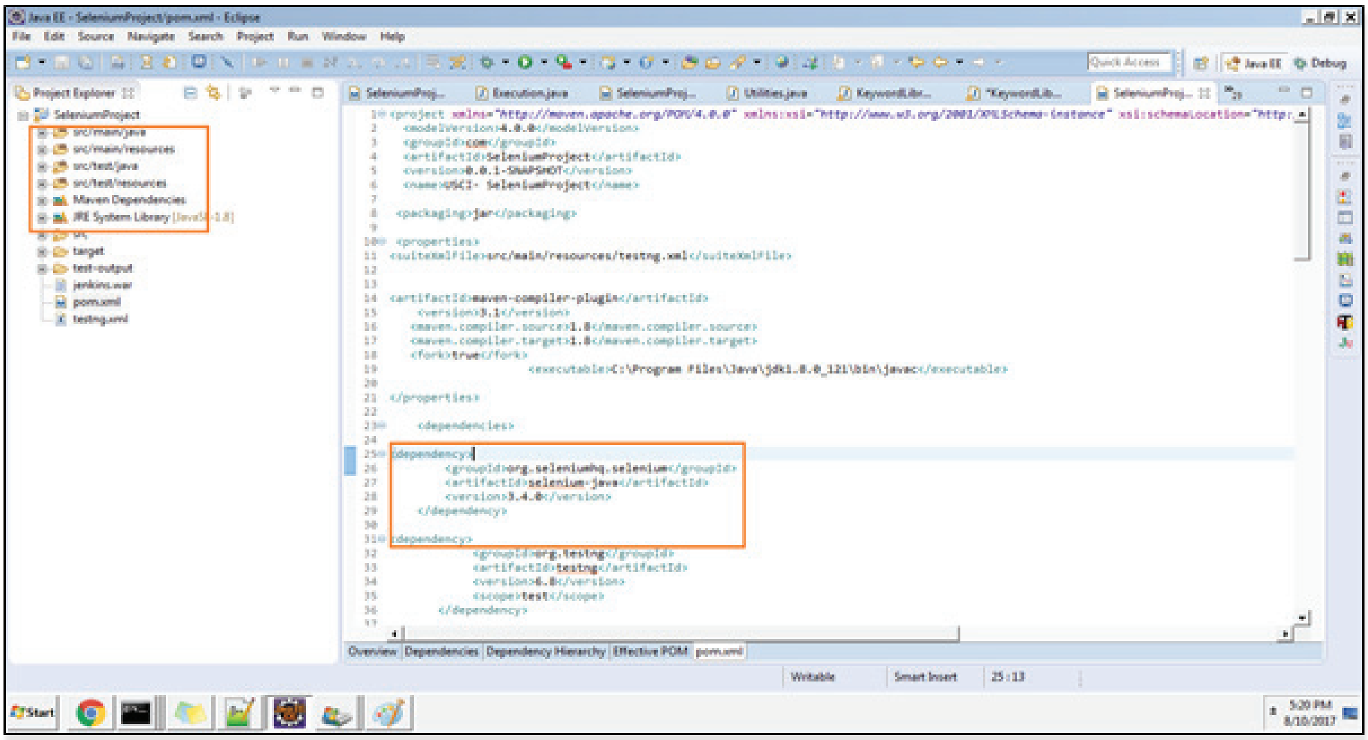
Repositories

Repository URL: https://github.com/Sapaan/Demo.git

Credentials: - none -

Keep Automation test suite ready

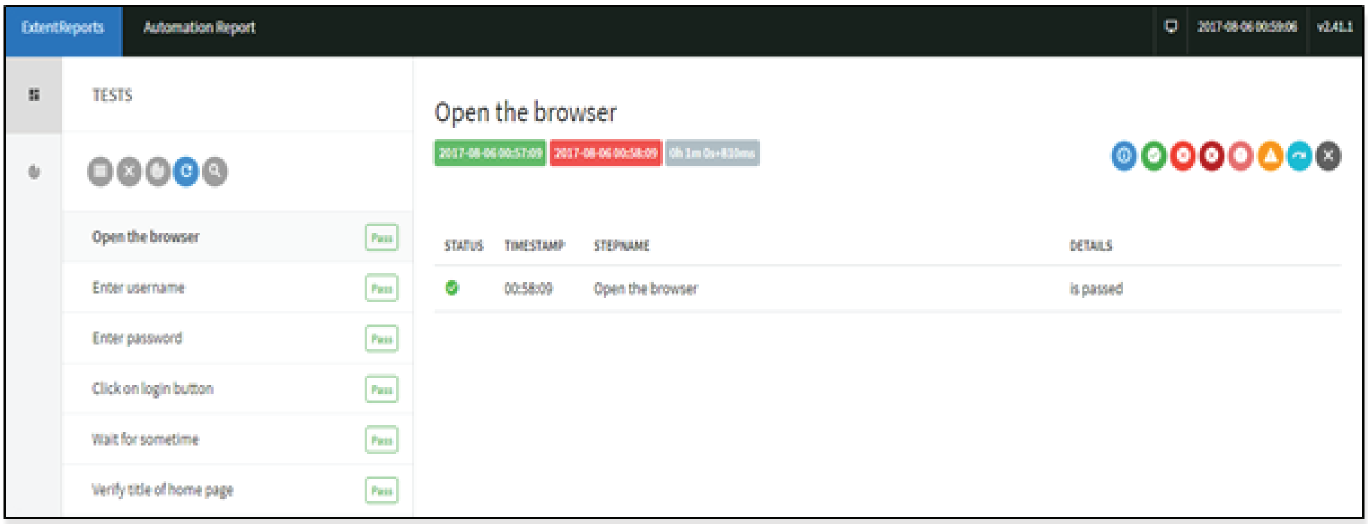
In order to execute our test cases automatically in build process, we should keep our test suite ready. Here Selenium Webdriver is used with TestNG + Maven and test cases are automated using Hybrid Framework. Once Maven is integrated with Selenium, pom.xml file is created where all project configuration details are mentioned. It also provides initial folder structure as below.



```
SeleniumProject
  src/main/java
  src/main/resources
  src/test/java
  src/test/resources
  Maven Dependencies
  JRE System Library [JavaSE-1.8]
  src
  target
  test-output
  jenkins.war
  pom.xml
  testng.xml
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http:
  <modelVersion>4.0.0</modelVersion>
  <groupId>com</groupId>
  <artifactId>SeleniumProject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>USCI- SeleniumProject</name>

 <packaging>jar</packaging>

 <properties>
<suiteXmlFile>src/main/resources/testng.xml</suiteXmlFile>


<artifactId>maven-compiler-plugin</artifactId>
    <version>3.1</version>
   <maven.compiler.source>1.8</maven.compiler.source>
   <maven.compiler.target>1.8</maven.compiler.target>
   <fork>true</fork>
                   <executable>C:\Program Files\Java\jdk1.8.0_121\bin\javac</executable>

</properties>

    <dependencies>

<dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>3.4.0</version>
    </dependency>

<dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.8</version>
            <scope>test</scope>
        </dependency>
```

After execution of the test scripts, it is very important have a snapshot of our execution status as in no. of test cases passed, failed, skipped, time required to run the test, etc. Extent report, along with TestNG, helps us for providing this status in very user-friendly way as shown in the screenshot below.



Open the browser	Pass
Enter username	Pass
Enter password	Pass
Click on login button	Pass
Wait for sometime	Pass
Verify title of home page	Pass

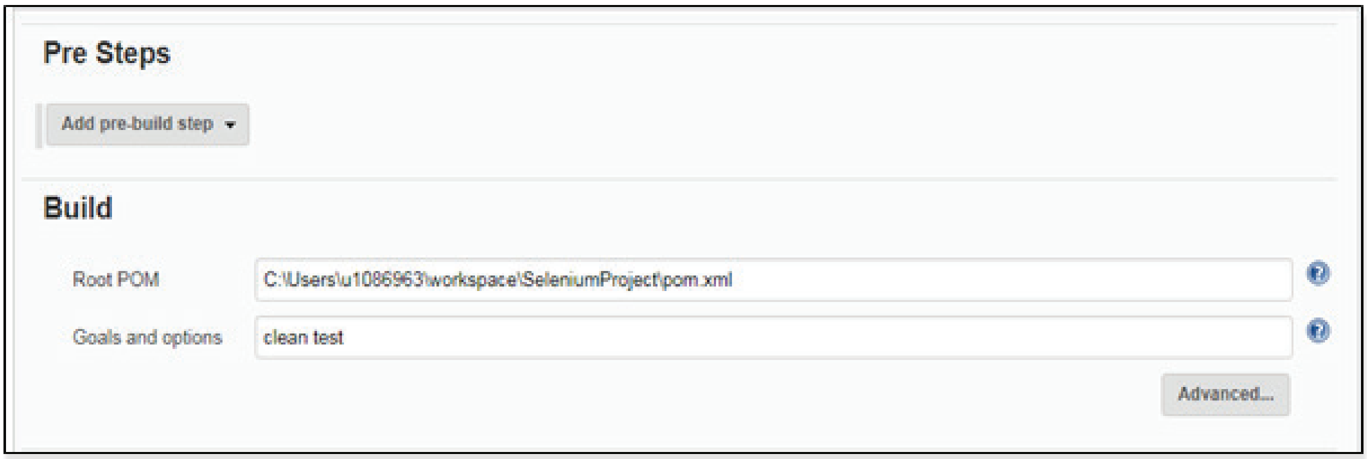
Open the browser

2017-08-06 00:57:09 2017-08-06 00:58:09 0h 1m 0s+810ms

STATUS	TIMESTAMP	STEPNAME	DETAILS
✓	00:58:09	Open the browser	is passed

Make the Build Self-Testing

For self-testing code, we need a suite of automated tests that can check a large part of the code base for a bug. A good way to catch bugs more quickly and efficiently, is to include automated tests in the build process. The tests should be able to be kicked-off from a simple command and to be self-checking. Here executable pom.xml path is given build section of Jenkins.



Root POM: C:\Users\u1086963\workspace\SeleniumProject\pom.xml

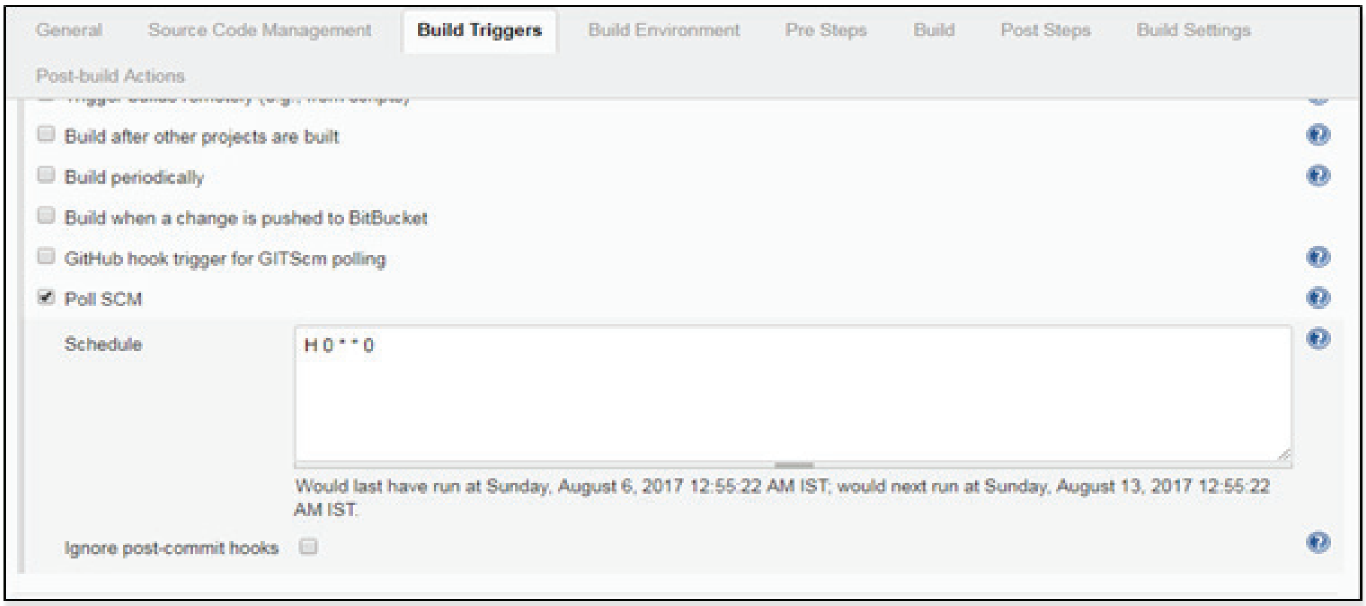
Goals and options: clean test

Keep the build fast

If the build is slow, then it would indicate a problem in the entire Continuous Integration process. CI ensures that the builds are fast, and always limited to duration.

Scheduling Build

Jenkins has an ability to schedule the builds based on different configurations. From Build Triggers, Jenkins can monitor the repository and start a build whenever any changes have been committed by choosing Poll SCM option and entering syntax of cron. We can also schedule a nightly build by selecting a periodical option and providing cron pattern. Project can also be built after other projects are built successfully through 'Build after other projects are built' option.



- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] Build when a change is pushed to BitBucket
- [ ] GitHub hook trigger for GITScm polling
- [✓] Poll SCM

Schedule

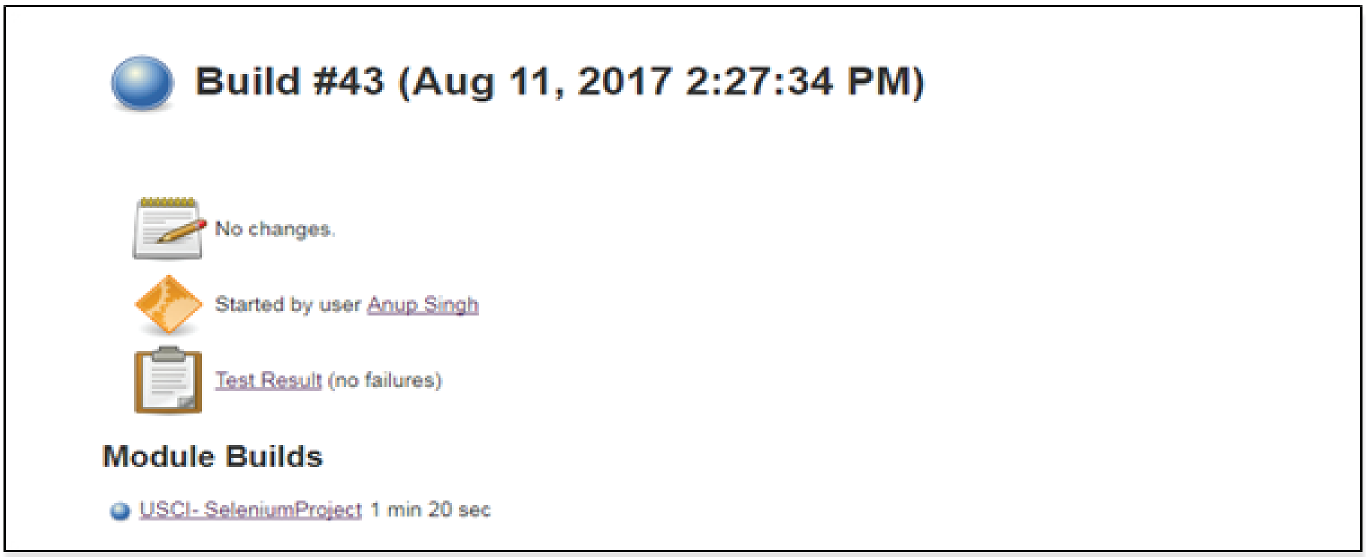
```
H 0 * * 0
```

Would last have run at Sunday, August 6, 2017 12:55:22 AM IST; would next run at Sunday, August 13, 2017 12:55:22 AM IST.

Ignore post-commit hooks [ ]

Verifying the Latest Builds

To verify the status of a build, once the build is triggered, we can check for the console output. It will show something like the below image, which has information like who started and from where the build is triggered, and build success or failure status, etc.



Build #43 (Aug 11, 2017 2:27:34 PM)

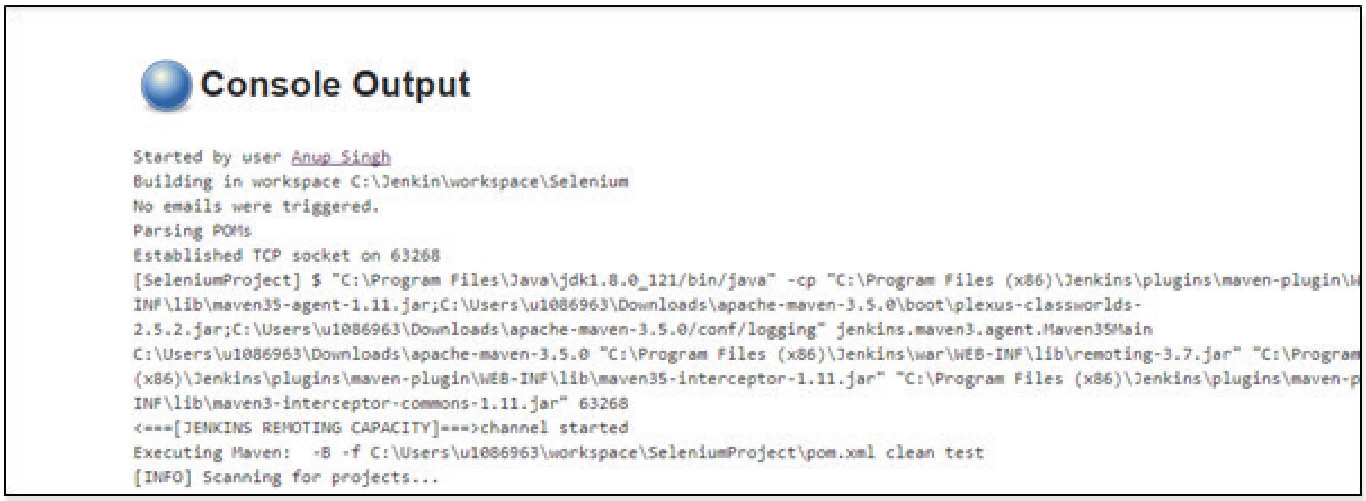
No changes.

Started by user Anup Singh

Test Result (no failures)

Module Builds

USCI- SeleniumProject 1 min 20 sec

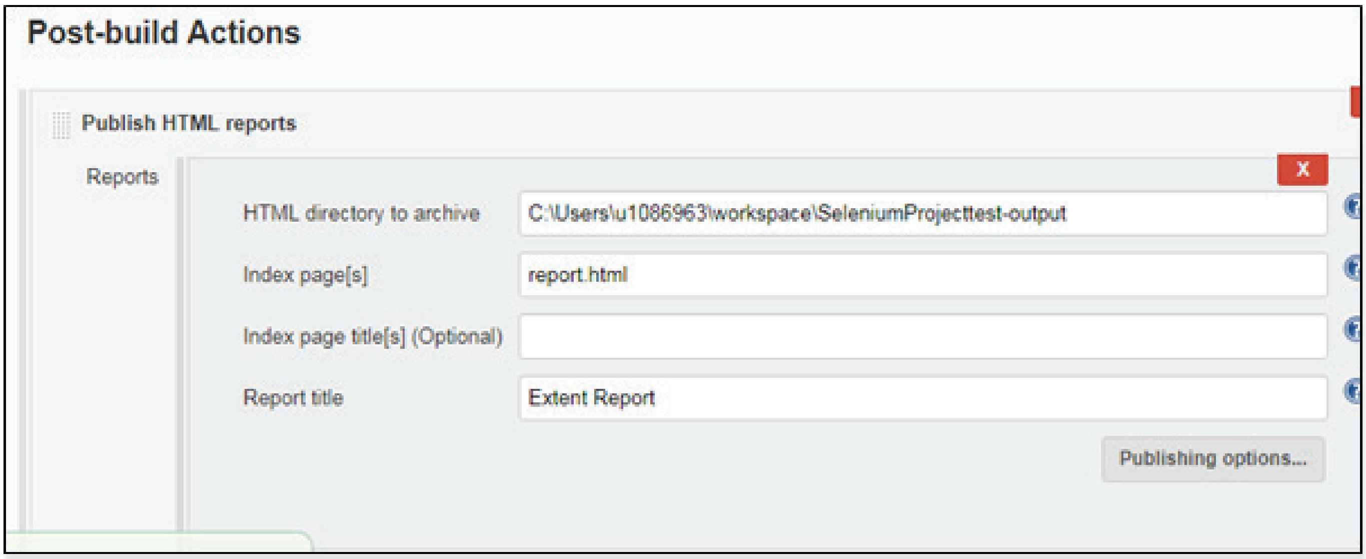


Console Output

```
Started by user Anup Singh
Building in workspace C:\Jenkin\workspace\Selenium
No emails were triggered.
Parsing POMs
Established TCP socket on 63268
[SeleniumProject] $ "C:\Program Files\Java\jdk1.8.0_121/bin/java" -cp "C:\Program Files (x86)\Jenkins\plugins\maven-plugin\W
INF\lib\maven35-agent-1.11.jar;C:\Users\u1086963\Downloads\apache-maven-3.5.0\boot\plexus-classworlds-
2.5.2.jar;C:\Users\u1086963\Downloads\apache-maven-3.5.0/conf/logging" jenkins.maven3.agent.Maven35Main
C:\Users\u1086963\Downloads\apache-maven-3.5.0 "C:\Program Files (x86)\Jenkins\war\WEB-INF\lib\remoting-3.7.jar" "C:\Program
(x86)\Jenkins\plugins\maven-plugin\WEB-INF\lib\maven35-interceptor-1.11.jar" "C:\Program Files (x86)\Jenkins\plugins\maven-p
INF\lib\maven3-interceptor-commons-1.11.jar" 63268
<===[JENKINS REMOTING CAPACITY]===>channel started
Executing Maven:  -B -f C:\Users\u1086963\workspace\SeleniumProject\pom.xml clean test
[INFO] Scanning for projects...
```

Publishing Execution Reports in JIRA

We can publish automated HTML reports in JIRA through Jenkins, by installing HTML Publisher plugin, and by configuring Zephyr for JIRA Test Management plugin as a post build action. Provide report URL path as a post build action in 'Publish HTML report' section.



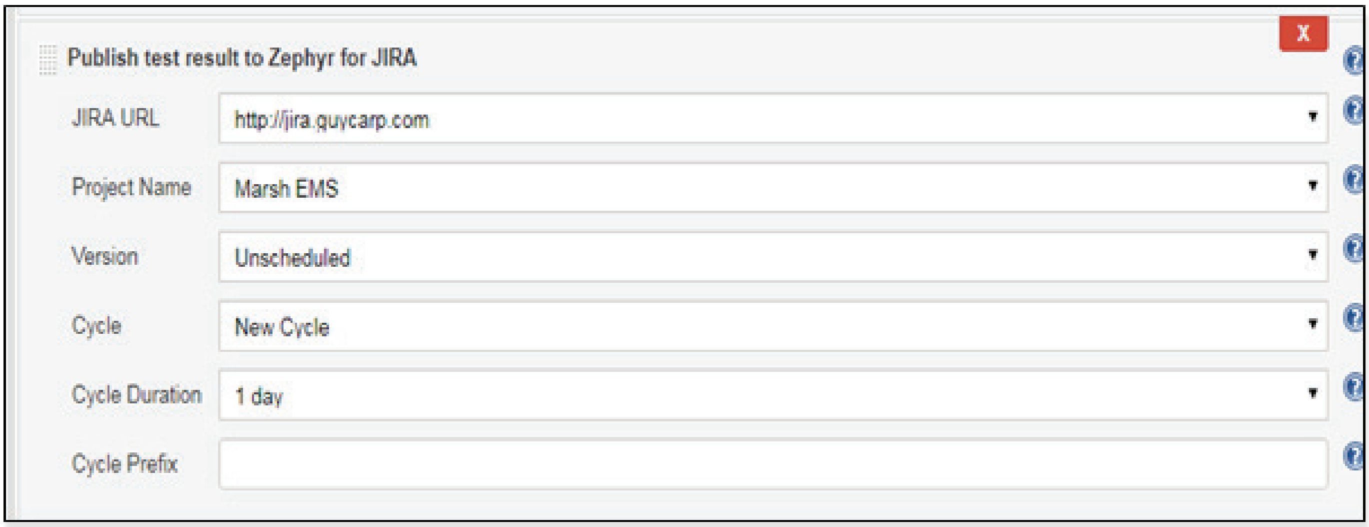
Post-build Actions

Publish HTML reports

Reports

HTML directory to archive	C:\Users\u1086963\workspace\SeleniumProjecttest-output
Index page[s]	report.html
Index page title[s] (Optional)	
Report title	Extent Report

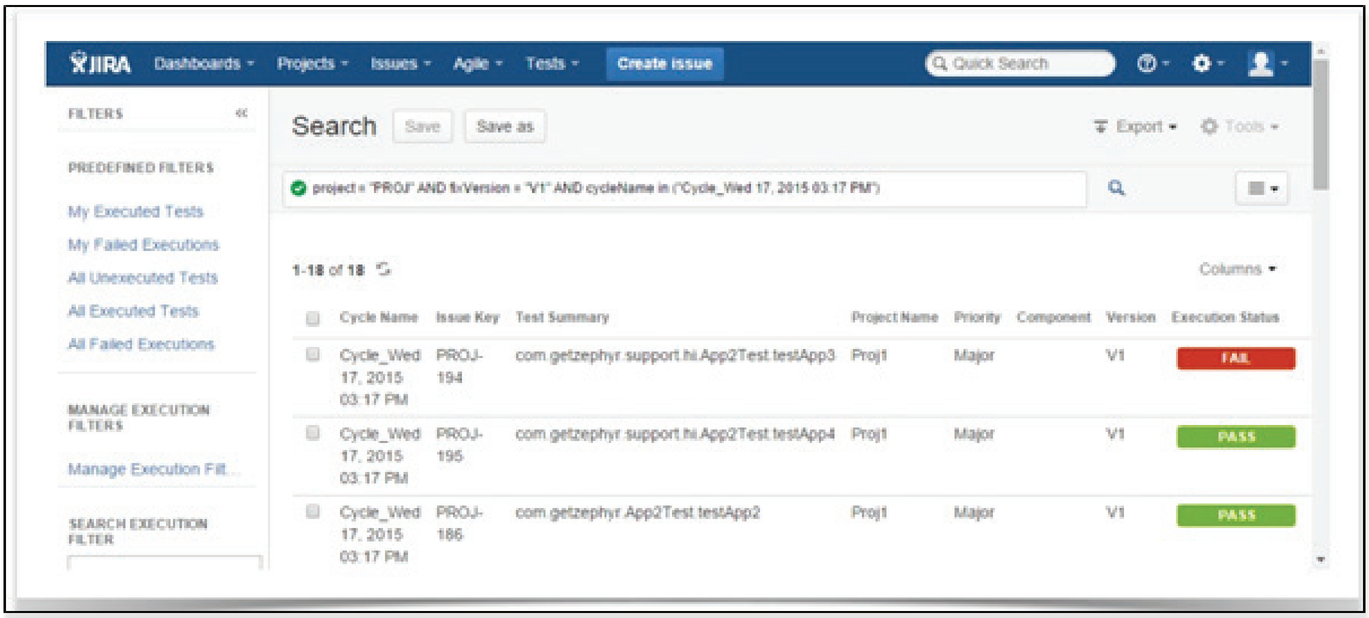
Provide Jira URL in 'Publish test result to Zephyr for JIRA' section as post build action as below:



Publish test result to Zephyr for JIRA

JIRA URL	http://jira.guycarp.com
Project Name	Marsh EMS
Version	Unscheduled
Cycle	New Cycle
Cycle Duration	1 day
Cycle Prefix	

Test Results will be published in JIRA:

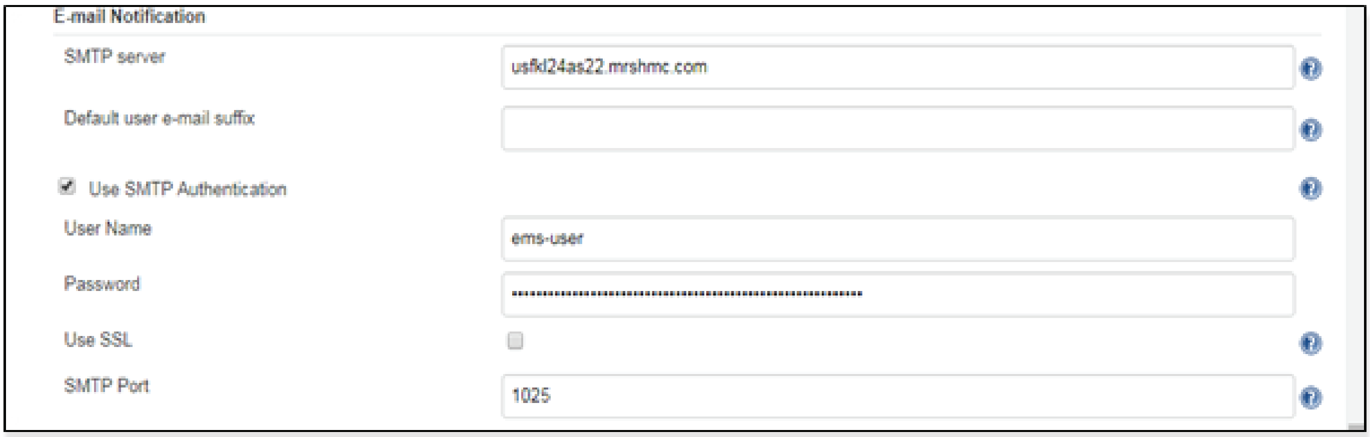


project = "PROJ" AND fixVersion = "V1" AND cycleName in ("Cycle_Wed 17, 2015 03:17 PM")

1-18 of 18

Cycle Name	Issue Key	Test Summary	Project Name	Priority	Component	Version	Execution Status
Cycle_Wed 17, 2015 03:17 PM	PROJ-194	com.getzephyr.support.hi.App2Test.testApp3	Proj1	Major		V1	FAIL
Cycle_Wed 17, 2015 03:17 PM	PROJ-195	com.getzephyr.support.hi.App2Test.testApp4	Proj1	Major		V1	PASS
Cycle_Wed 17, 2015 03:17 PM	PROJ-186	com.getzephyr.App2Test.testApp2	Proj1	Major		V1	PASS

Emails can also be triggered to a respective person after every build run by providing some email configurations like SMTP server, Port no., username and password for authentication in 'Email notification' section in Jenkins as follows:

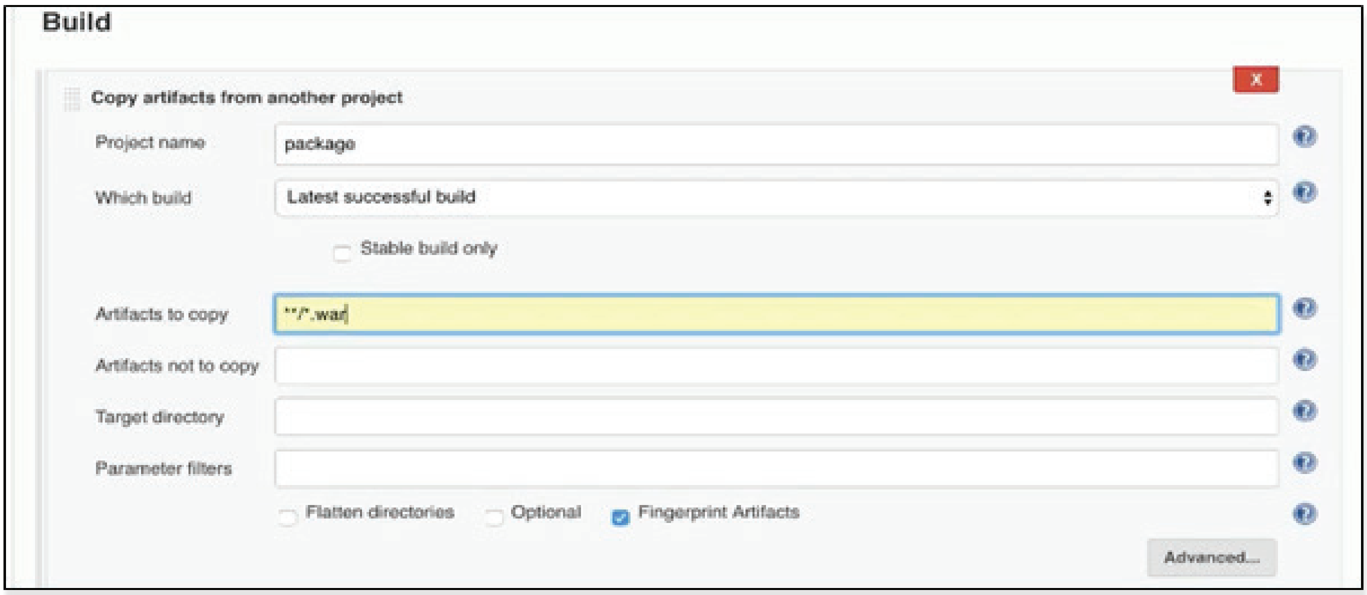


E-mail Notification

SMTP server	usfkl24as22.mrshmc.com
Default user e-mail suffix	
✓ Use SMTP Authentication	
User Name	ems-user
Password	••
Use SSL	☐
SMTP Port	1025

Automatic Deployment to Staging

Automated deployment allows developers to push the product to various environments quickly and save a lot of time. This can be done through 'Copy artifact' and 'deploy to container' plugins of Jenkins. Here we have created a separate job (Deploy to staging) to deploy the application to staging environment, which is a Tomcat server. In the Copy artefact section, we need to give project name and artifacts to copy as shown below.

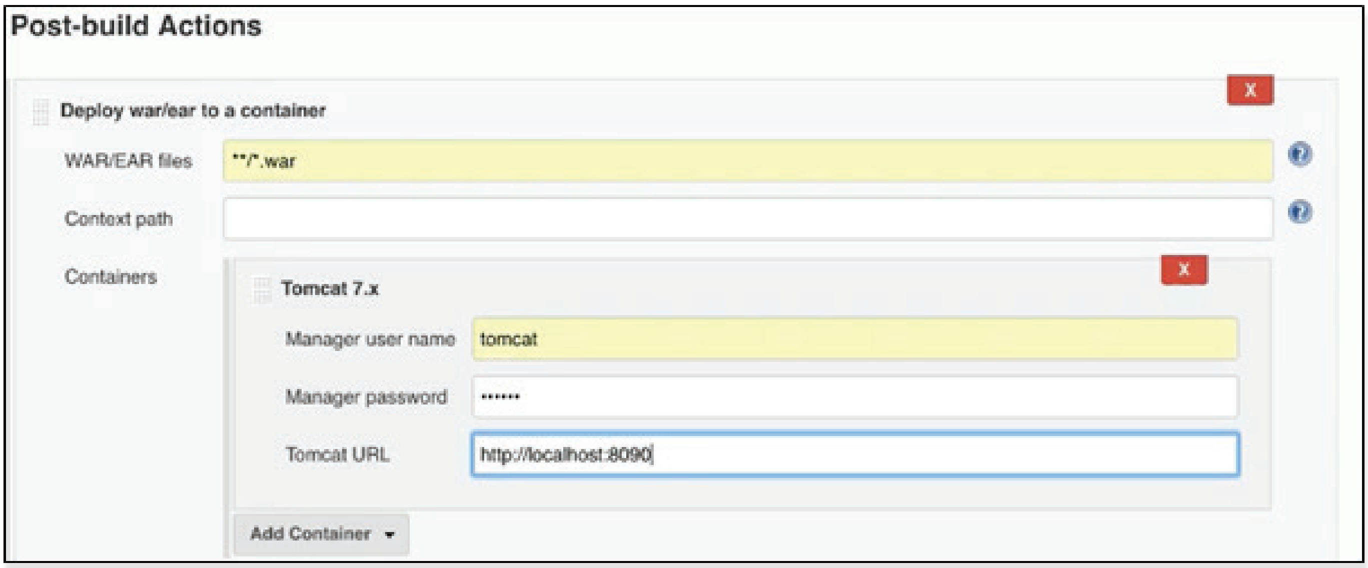


Build

Copy artifacts from another project

Project name	package
Which build	Latest successful build
	☐ Stable build only
Artifacts to copy	**/*.war
Artifacts not to copy	
Target directory	
Parameter filters	
	☐ Flatten directories ☐ Optional ✓ Fingerprint Artifacts

And in Deploy to container, need to specify container details of staging and artifacts to deploy:



Post-build Actions

Deploy war/ear to a container

WAR/EAR files: `**/*.war`

Context path:

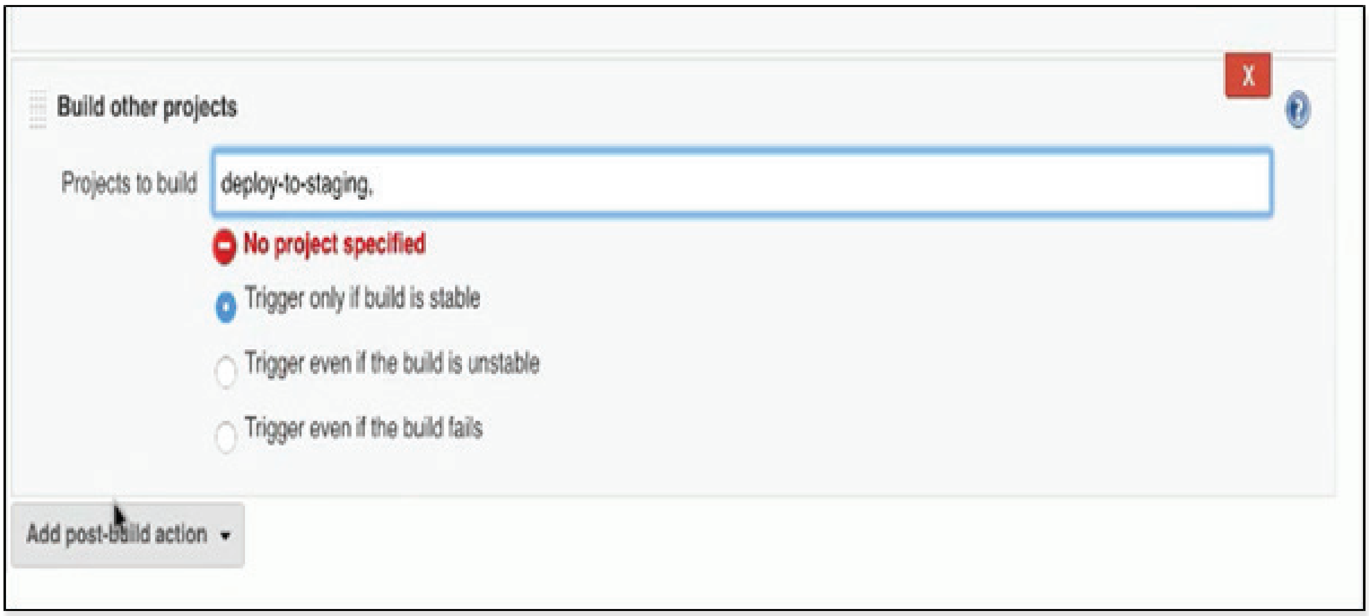
Containers: Tomcat 7.x

Manager user name: tomcat

Manager password: ••••••

Tomcat URL: http://localhost:8090

Now 'Deploy to staging' job will be automatically triggered, if upstream job runs successfully. This can be achieved by adding post build action to upstream job to trigger downstream job as follows:



Build other projects

Projects to build: deploy-to-staging,

No project specified

[✓ Trigger only if build is stable]

Trigger even if the build is unstable

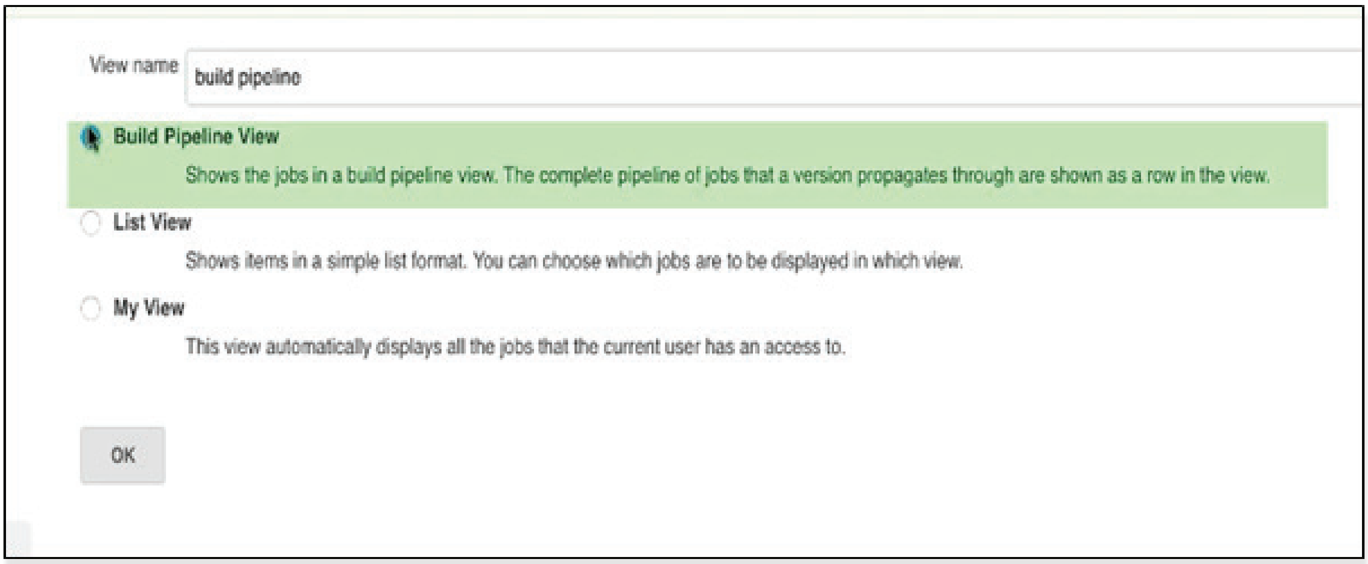
Trigger even if the build fails

And this way the build will be deployed to staging.

Jenkins Build Pipeline

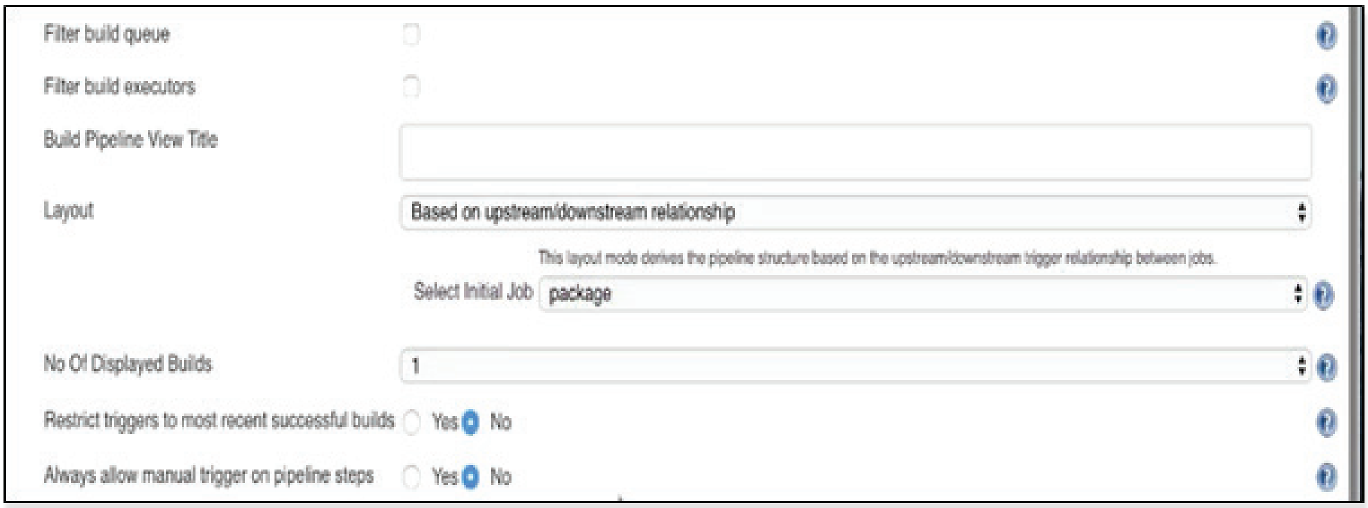
When no. of jobs (which are connected to each other) increases in Jenkins, it becomes difficult to understand the pipeline. To visualise these pipelines and to understand how they are connected, we have Jenkins 'Build Pipeline' plugin. It offers build pipeline view of all upstream and downstream jobs.

Create new view on Jenkins home page as 'Build Pipeline view', and configure the initial job to be triggered.



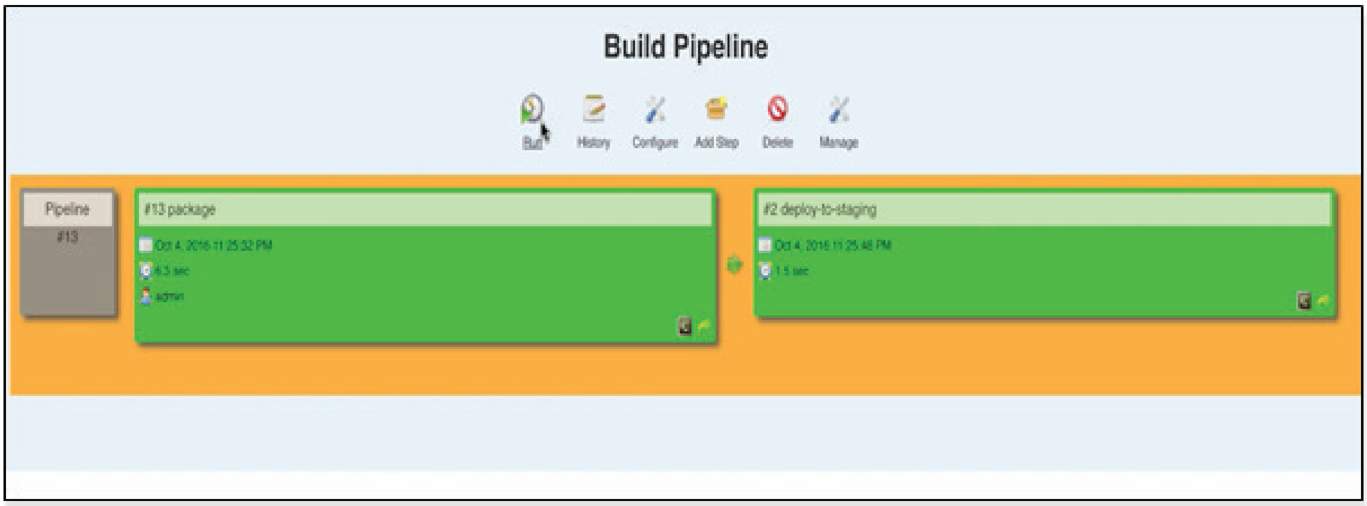
View name: build pipeline

- [✓ **Build Pipeline View**]
  Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.
- **List View**
  Shows items in a simple list format. You can choose which jobs are to be displayed in which view.
- **My View**
  This view automatically displays all the jobs that the current user has an access to.



Filter build queue	☐
Filter build executors	☐
Build Pipeline View Title	
Layout	Based on upstream/downstream relationship
	This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs.
Select Initial Job	package
No Of Displayed Builds	1
Restrict triggers to most recent successful builds	Yes [✓ No]
Always allow manual trigger on pipeline steps	Yes [✓ No]

This is how we can see build pipeline view of all upstream and downstream jobs:



Build Pipeline

Pipeline #13

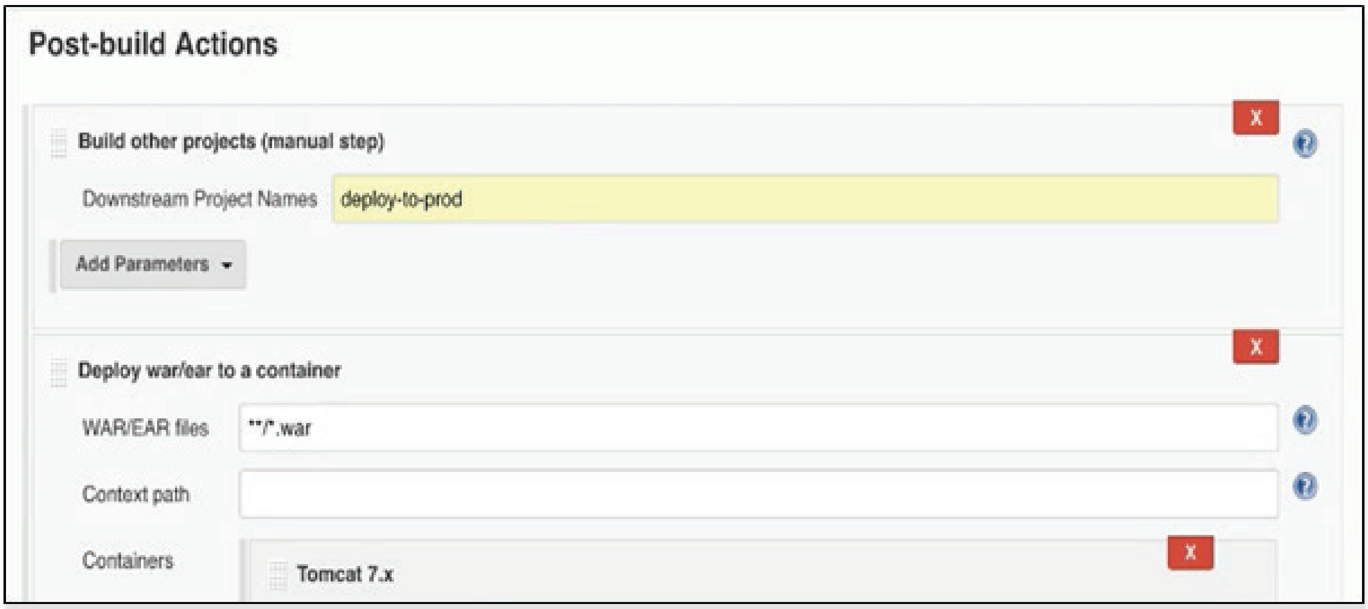
#13 package
Oct 4, 2016 11:25:32 PM
6.3 sec
admin

#2 deploy-to-staging
Oct 4, 2016 11:25:48 PM
1.5 sec

Deploy to Production

This is a final stage of build pipeline to deploy our application to production. This process can be automated, or can be done manually. We can create Production deployment environment by spinning new Tomcat server. This can be achieved by replicating staging environment binaries and configuration files from staging to production environment.

Now created another job called 'Deploy to Prod' to deploy our code in production environment and gave same configuration as Staging job like war file to be deployed and deploy to container as a post build action. This job should be triggered manually after 'deploy to staging job' completes successfully. This can be done by selecting 'Build other projects (manual step)' as a post build action in the "deploy to staging upstream" job.


Post-build Actions

Build other projects (manual step)

Downstream Project Names: deploy-to-prod

Deploy war/ear to a container

WAR/EAR files: **/*.war

Context path:

Containers: Tomcat 7.x

And now, when we build the initial job which is our selenium test suite, all other dependent jobs will run automatically except 'deploy to prod' as shown below. Other jobs are shown in green means they are completed successfully and "deploy to prod" is shown in blue, because this job is configured to trigger manually.

13 Key Benefits of CI

- Enables automated testing
- Minimizes risk by providing teams with real-time notification and transparency
- Increases confidence in the software
- Reduces integration problems allowing you to deliver software more rapidly
- Saves Time
- Adds Consistency
- With automated deployment, project cost is reduced by more than 50%
- Projects the use of CI more than twice, as often as those that do not use CI

14 Success Story

In our project, we have Bi-monthly releases which consists a lot of manual efforts for unit testing, deployment to different environments, system testing, staging deployment, etc. With implementation of CI, manual efforts are cut down by 75%, by automating complete process right from unit testing till deployment to staging.

Bug fix cost

Before CI, the team used to fix the bugs once QA testing is in progress. With implementation of CI, bugs got detected as and when the code got checked in, in the repository and helped in eliminating 75% of the development efforts.

Deployment cost

AES team (deployment team) used to take 4 hrs. for deployment per release, which is now reduced to 10 mins.

Cost of delay

Before CI, features are held on average 6 weeks after development completion before they are released. Now with CI in place features are released as and when they are ready..

15 Future Vision

In the future, CI will have an even greater influence than it has today. Automated environment and infrastructure creation and provisioning for cloud-based systems will mature fast. Tools like Puppet, Chef and Capistrano making automated provisioning a reality. Containers, Automated deployment, etc. are another area seeing growing adoption. Expected to have tool that grabs stack traces from production logs, and do reverse engineering of the code to auto generate the test cases in our suite. Move from Continuous Integration to Continuous Delivery, and deployment is the future.

16 Conclusion

In this white paper, we introduced Continuous Integration, Continuous Delivery and discussed how they can be used to build and release well-tested software safely and quickly, and create a better product for the customer, which is the real promise of agility. These processes leverage extensive automation and encourage constant code sharing to fix defects early. While the techniques, processes, and tools needed to implement these solutions represent a significant challenge, the benefits of a well-designed and properly used system are enormous.

17 References

- https://dzone.com/articles
- https://jenkins.io/doc/
- https://www.tutorialspoint.com/jenkins/jenkins_overview.htm
- https://en.wikipedia.org/wiki/Jenkins

18 About the Author

Sapana Kejadiwal is working as Test Lead with LTI in the MMC account. She has a total 7.5 years of Testing Experience in Manual and Automation Testing, with different domain knowledge like BFSI, MS CRM, Mobility, Insurance, etc. She has been working with LTI for 2.5 years.