**Continuous Testing of a Java Web App in using Selenium**

Here are the pre-requisites for this article:

1. Visual Studio Team Services (VSTS) Account with one Team Project created in it (it can be of any process template)

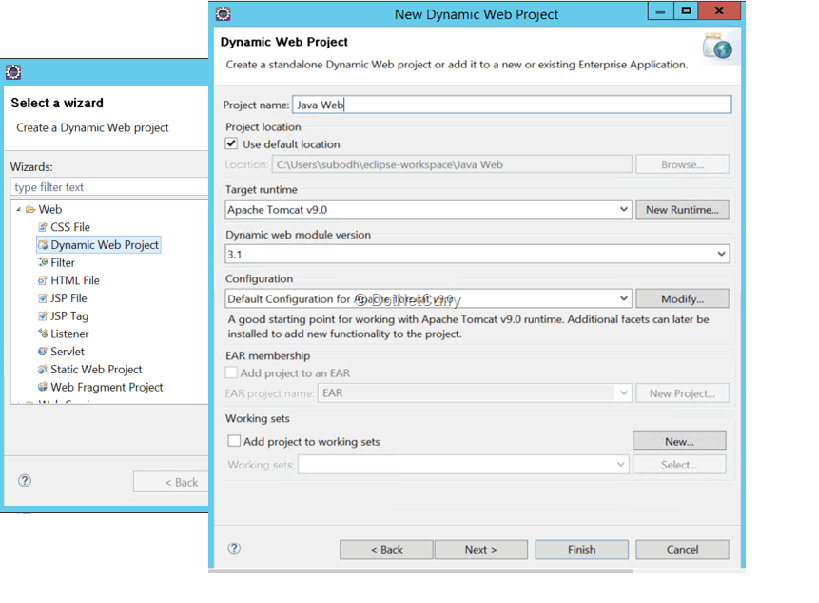
2. Account for Azure Portal to create Java Web App

3. Java SDK installed and added in JAVA\_HOME environment variable.

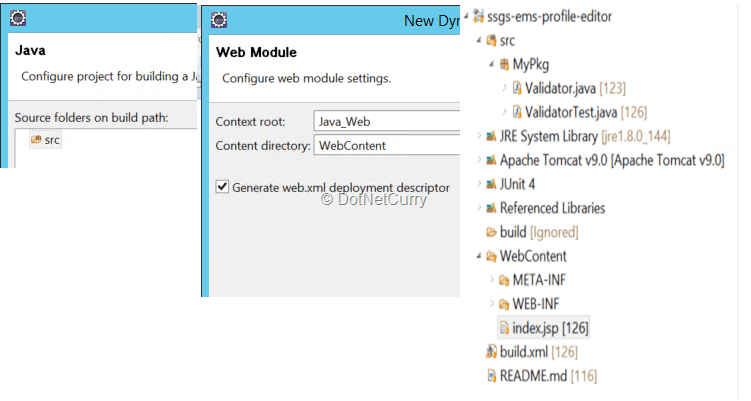
4. Eclipse IDE, Ant and Apache Tomcat (if we want to view the application using Tomcat)

## Using Eclipse with VSTS

1. Using Eclipse IDE create a Dynamic Web Project



2. I have added index.jsp in the WebContent folder which is in the src folder



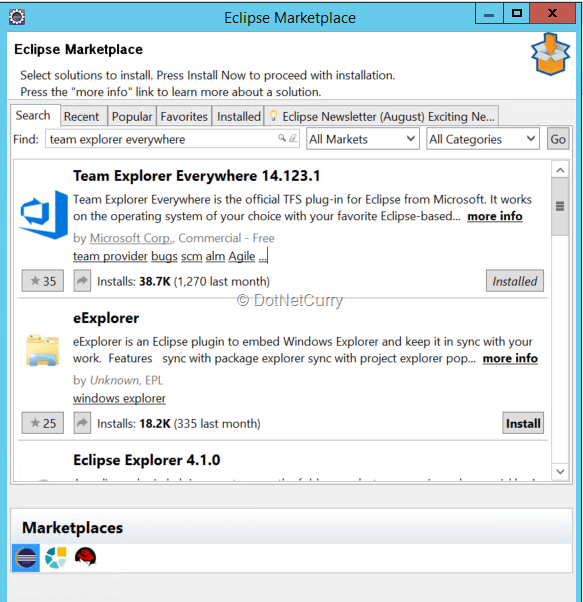
3. Let us add some code to the project.

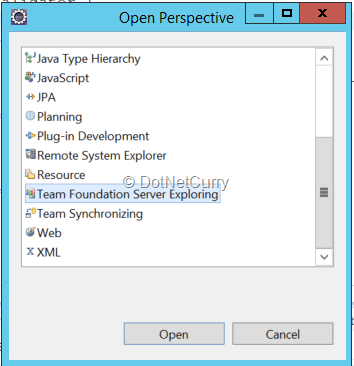
I have written a Validate method in the Validator class. I have also written code to call the method. Just some very basic functionality.



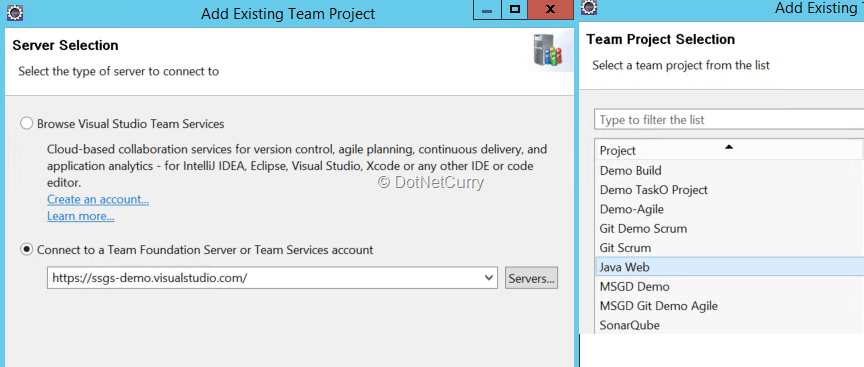
### Connect to Team Explorer Everywhere

4. As we are going to use Visual Studio Team Services as our code repository, we need to connect to VSTS. Go to help in Eclipse, browse to the Eclipse Marketplace and search for Team Explorer Everywhere. Install and then you can open perspective for Team Foundation Server Exploring.

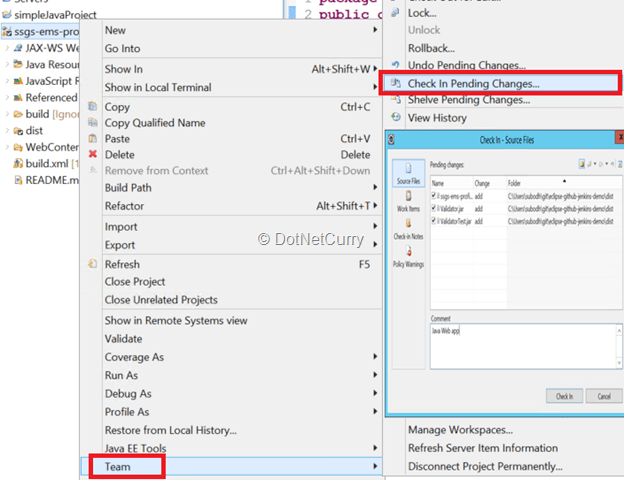




5. After adding code, we need to connect to VSTS and use TFVS as Source Control. Connect to VSTS and the Team Project in it.



6. Right click on the project > select Team > Check in pending changes. Provide the comment for check in and check in your code.



### Write a JUnit Test and execute it in Eclipse IDE

7. We have created and checked in code for our Web Project. Now is the time to write the JUnit test.

JUnit is a framework for Java Programming Language. It helps in writing tests for functionality tests, and also for Test Drive Development (TDD).

If you are new to Unit Testing, check these tutorials:

[Live Unit Testing in Visual Studio 2017](https://www.dotnetcurry.com/visualstudio/1363/live-unit-testing-visual-studio-2017)

[NUnit Testing with Visual Studio 2015](https://www.dotnetcurry.com/visualstudio/1352/nunit-testing-visual-studio-2015)

[Smart Unit Tests in Visual Studio 2015](https://www.dotnetcurry.com/visualstudio/1106/visual-studio-2015-smart-unit-tests)

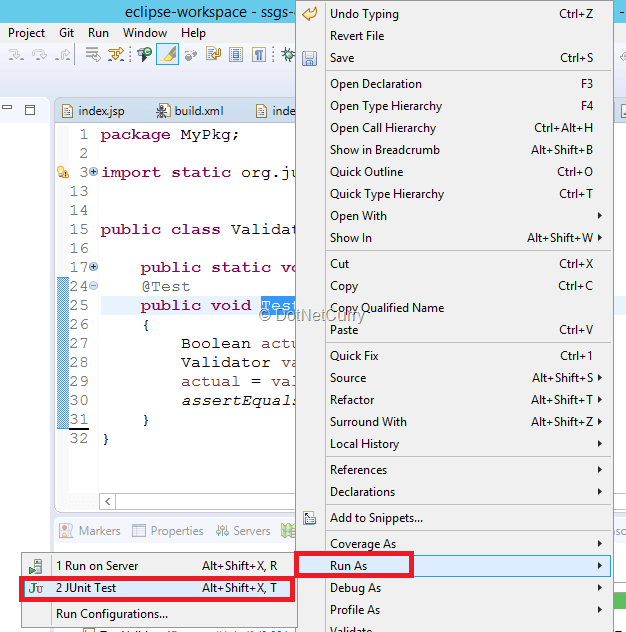
8. There are a few annotations for writing JUnit Tests. Firstly, we need to download the .jar files for JUnit. Provide the required import statements.

|  |
| --- |
| import static org.junit.Assert.assertEquals;  import org.junit.Test;  import org.junit.runner.JUnitCore;  import org.junit.runner.Result; |

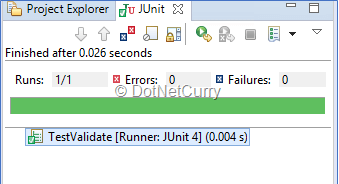
9. Let us write the actual code for the JUnit test.

|  |
| --- |
| public class ValidatorTest {  public static void main()    @Test   public void TestValidate()  {      Boolean actual, expected=true;      Validator val = new Validator();      actual = val.Validate("Gouri", "Sohoni");      assertEquals(expected,actual);  }  } |

10. We can execute the JUnit in Eclipse IDE itself. Right click on the test > select Run As > JUnit Test and you can see the result in runner.

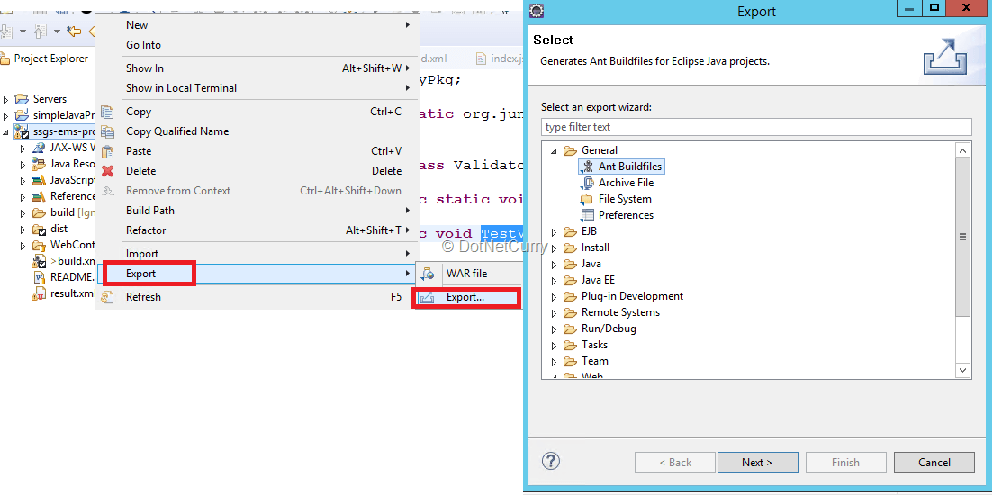


The result looks as follows:



### Add Ant for Build – xml file with targets

11. As I have already mentioned, I will be using Ant to build the code. We can Export the Ant Build files by right clicking on project and selecting Export > Ant Buildfiles



Ant provides build.xml file which consists of target and tasks. It has a project, which can have any number of targets and various tasks in it.

We can specify which is the default target (required) and the dependent target for the same. The build will then go on fetching the dependent target and execute the tasks in each one of them.

The build target can have tasks for compiling, the test target can have tasks for test execution and report task can be for creating test reports.

Following is an example of java compilation task:

|  |
| --- |
| <javac debug="true" debuglevel="${debuglevel}" destdir="build/classes" includeantruntime="false" source="${source}" target="${target}" path="src>              <classpath refid="> </javac> |

12. Following is the example for a test target:

|  |
| --- |
| <target name="test" depends="pack">     <mkdir dir="testresults"></mkdir>     <junit printsummary="yes">         <classpath>             <pathelement location="<jar" files="" path="">             <pathelement location="build/classes"></pathelement>         </pathelement></classpath>         <formatter type="xml"></formatter>         <test todir="testresults" outfile="result.xml“        name="></test>     </junit> </target> |

Observe this target depends on the pack target. The location to the .jar files is specified. I am creating a directory where the test results are stored. The test results will be put in an xml file.

Do not forget to check-in this file.

13. I have deployed the web app to Apache Tomcat server (optional) and started the server (to deploy and check if required).

14. Let us add code for Selenium test. Download selenium jar files from <http://www.seleniumhq.org/download/> and unzip the files. Add support to external jar files.

* Right click on project – Select Properties – Java Build Path
* Navigate to Libraries tab and click Add External JARS
* Add jar file from the unzipped folder and all jar files from the unzipped folder

### Add code for Selenium Test

15. The code for selenium test looks as follows:

|  |
| --- |
| @Test  public void TestSSGSWeb() throws InterruptedException{      ChromeOptions options = new ChromeOptions();      options.addArguments("--start-maximized");      WebDriver driver = new ChromeDriver(options);      driver.get("http://localhost:8080/ssgs-ems-profile-editor");      WebElement elementName = driver.findElement(By.name("uname"));      elementName.sendKeys("Gouri");      elementName.sendKeys(Keys.TAB);      WebElement elementPwd= driver.findElement(By.name("pwd"));      elementPwd.sendKeys("Sohoni");      elementPwd.sendKeys(Keys.TAB);      WebElement elementLogin =     driver.findElement(By.name("submoit"));      elementLogin.sendKeys(Keys.ENTER);      Thread.sleep(4000);      driver.quit();  } |

This code creates instance for chrome driver, navigates to the local tomcat server (later I will change the url to Java Web Service created using Azure). It also enters values for user name and password and clicks on the submit button.

This test can be executed with Eclipse IDE, it will be running on the local tomcat server (if you have installed and started the server).

Next step is to create a server-side build.

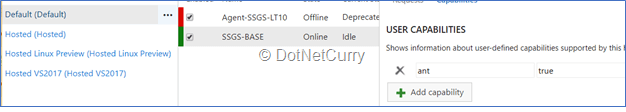
### Create Build Definition

16. We can create a build definition in VSTS by using the template for Ant.

In order to execute the build, we need a build agent. Build Agent is a computer where a specialized software is installed to run the workflow containing many tasks, as the steps for build or the release process.

VSTS can use an agent which is on premises of the customer or hosted in Azure. Build, Release Management and Test all can use the same agent. In this case I am going to use on premises build agent and not in cloud (as we need to execute JUnit tests as well as Selenium Test later).

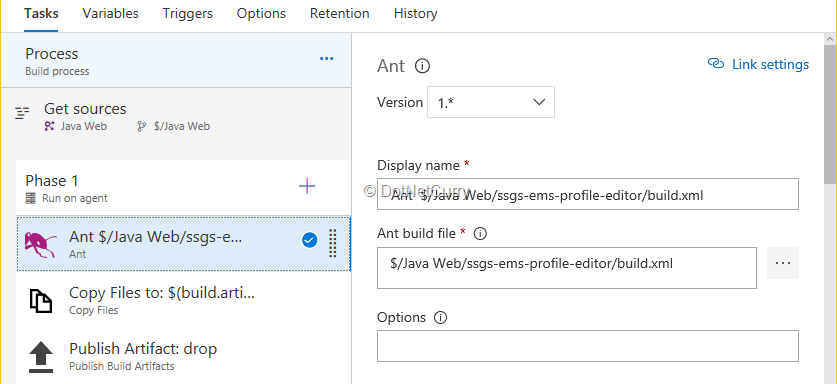
We also will have to add the capability for running ant script.



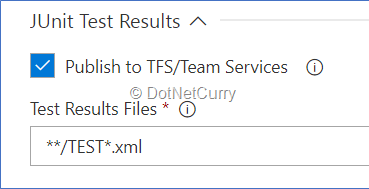
17. In order to install build agent, we have to download the zip file for agent from Agent Queues.

After un zipping the file, we can configure the agent by running config from command prompt. Provide the url for VSTS account, PAT (Personal Access Token), the work folder, pool name. Run it as an interactive process and not as a service (this is required because we will be running UI test later).

18. In the build definition for Ant, it will add tasks for Get Sources, Ant, Copy and Publish files. Provide the link to the build.xml file in ant task as follows:

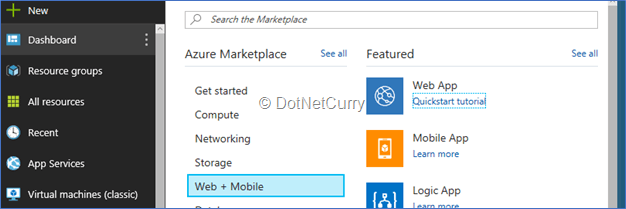


19. Click the check box for publishing test results. This will execute the tests and publish the result in the xml file.

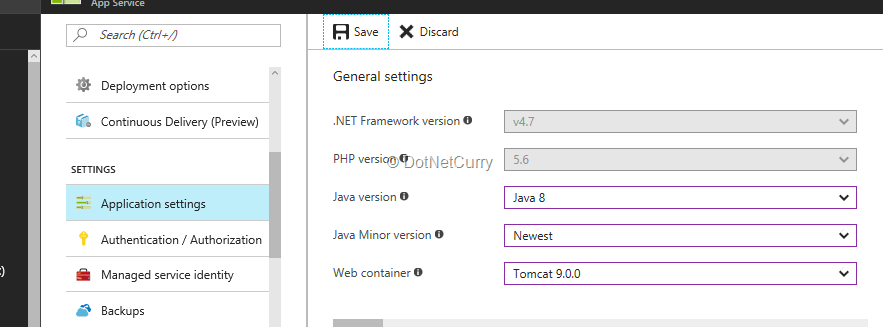


### Create Azure App Service supporting Java

20. Let us create a Java Web App Service for Azure. Login to azure portal, select New and select Web + Mobile and select Web App.



21. As we are going to use this web app service for Java, we do the following configuration:

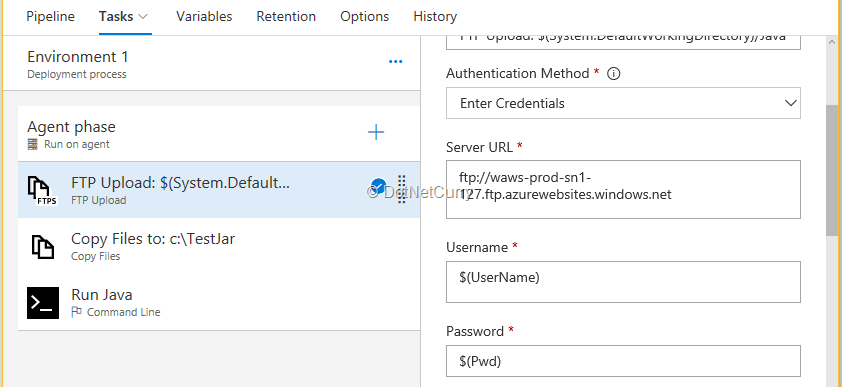


I have selected Java version as 8.0, minor as latest and Tomcat version 9.0.0. I have also provided the deployment credentials for the web app service. Now is the time to change the url for Selenium Test to the newly created Web Service App and check in this changed code.

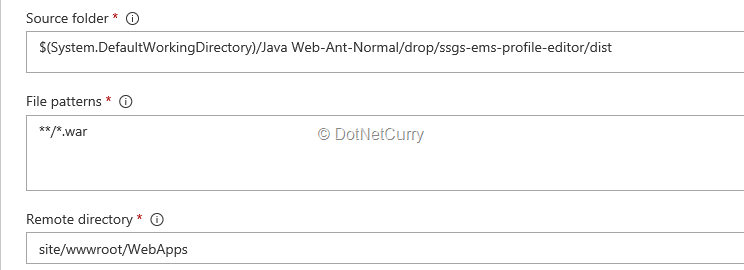
22. Trigger the build so as to create and publish the artifacts. Now we will proceed to deploying of the Web app by creating a Release.

### Create Release Definition

23. Let us create a release definition and add FTP upload task. This task will upload the application to the Web App Service. We need to provide the credentials as variables so that the actual values will not be displayed. You can add variables by selecting Variables tab.



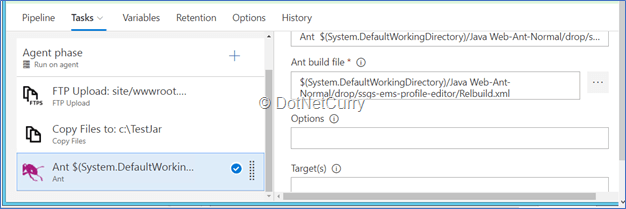
Source folder is the artifact drop folder where we need to copy the .war file. Tomcat Server requires the path to be site/wwwroot/WebApps as shown in remote directory.



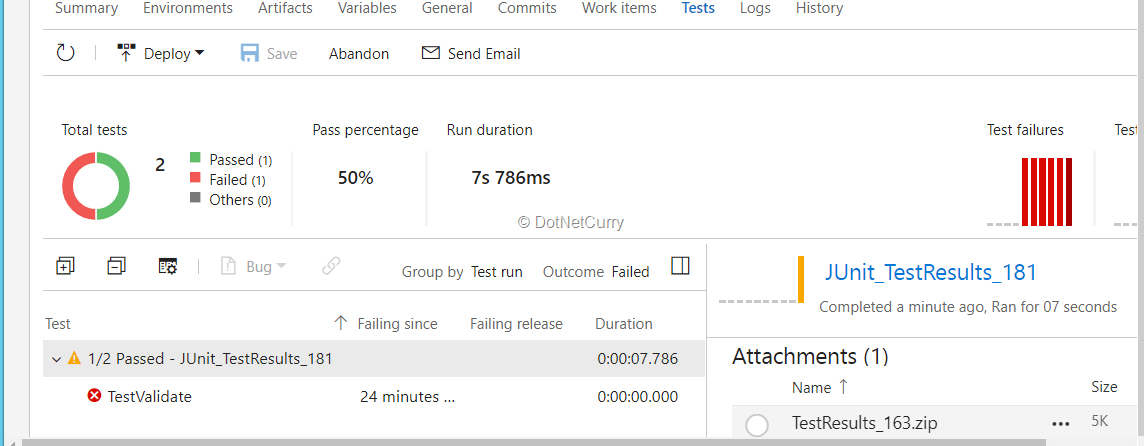
24. We need the test results to be published so we will use ant specific build file which will not do any compilation, but will only run tests and publish reports. Add another .xml file for ant and change the targets so as to have only test and reports related targets in it.

Check in the file. Ensure that this file gets dropped in the drop artifacts via build.

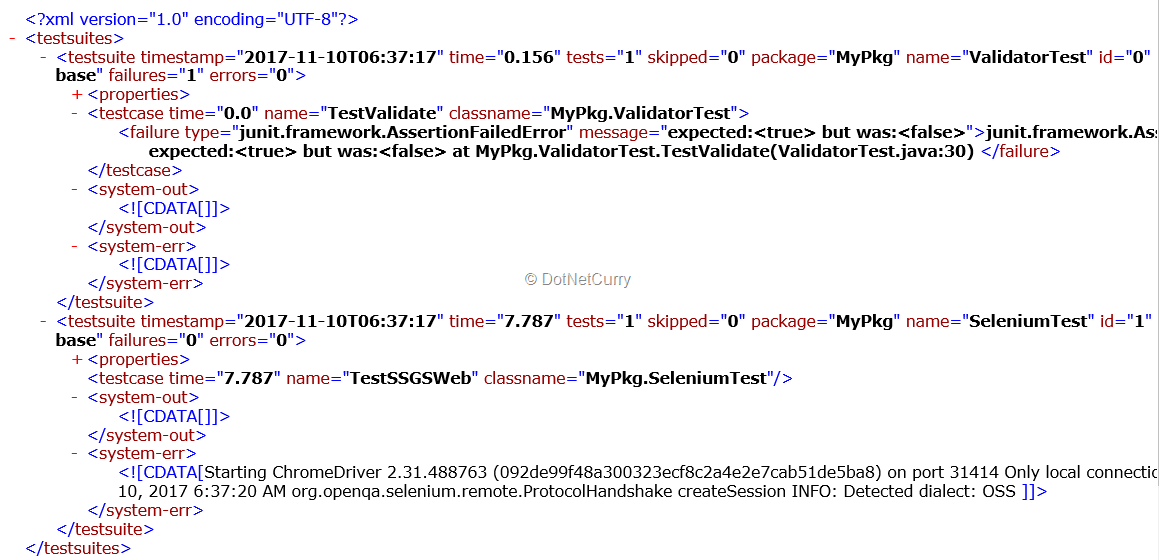
25. Add Ant build task and configure it to execute the new .xml file (the one which has targets for only test and report). Save the release definition. The release definition looks as follows:



26. Create a release. After successful upload of the web app to Web App Service, the Selenium test gets executed. The test results can be found from the Test tab of release.



I have purposely failed one of the tests. The result is shown in the summary. The zip file which is provided as attachment shows the following:



### Conclusion

In this article I discussed

* how a Java Web application can be created,
* added to Source Control on VSTS,
* how to create JUnit and Selenium Tests for it,
* how to create Azure Web App Services which supports Java,
* how to deploy the app on this service.