**DevTest(LISA)-BAMBOO INTEGRATION**

**DRAFT (1.2)**

|  |  |  |
| --- | --- | --- |
| Author | Date | Vesion |
| B Giridhar Dhanya | 21- October-2015 | 1.0 |
| B Giridhar Dhanya | 30 –November -2015 | 1.1 |
| Somanchi Subramanyam | 10-December-2015 | 1.2 |

**Contents**

[DevTest (LISA)-BAMBOO Integration 3](#_Toc437517676)

[DevTest (LISA) Mobile Test Automation Build Requirements 3](#_Toc437517677)

[Bamboo Setup 5](#_Toc437517678)

[Integrating Lisa and Bamboo 10](#_Toc437517679)

[Sample Maven script to call Ant script: 15](#_Toc437517680)

[Lisa Projects and sample Ant build scripts: 15](#_Toc437517681)

[Reports : 20](#_Toc437517682)

[LISA Ant Tasks 24](#_Toc437517683)

[junitlisa Ant Task 24](#_Toc437517684)

[junitlisareport Ant Task 26](#_Toc437517685)

[Conclusion : 27](#_Toc437517686)

DevTest (LISA)-BAMBOO Integration:

This document highlights the steps that are involved in setting up Bamboo integration with DevTest (Lisa) and running Lisa Test cases/Suites from Bamboo.

DevTest (LISA) Mobile Test Automation Build Requirements:

*Please refer to DevTest Documentation: for any specific version and configuration documentations.*

[*https://docops.ca.com//display/DTS84/Preinstallation+Steps+for+Mobile+Testing*](https://docops.ca.com//display/DTS84/Preinstallation+Steps+for+Mobile+Testing)

[*https://docops.ca.com//display/DTS90/Preinstallation+Steps+for+Mobile+Testing*](https://docops.ca.com//display/DTS90/Preinstallation+Steps+for+Mobile+Testing)

LISA would need debug builds from the Mobile Application Development Teams.

**Environment URL:**

Ensure the Mobile Application is pointing to the Environment as required to hit the backend services

* Configure your Mobile Application to point to the appropriate Environment.
* Generally the Application point to Production URLs by default
  + As required make the corresponding changes to the URL in the Mobile Application to point to
    - either QA Environment
    - or SV (Service virtualization) environment.

DevTest Mobile Testcases – can be run on a Simulator/Emulator/Devices as required by the Application team.

**Testing with Devices:**

* Ensure corresponding Device Drivers are installed and are visible in the system
  + For Android
    - Open a command prompt and run the below command
    - adp devices
    - The device should be listed and then only the DevTest testcases will run on the device
  + For iOS
    - Ensure you have setup the Developer certificates as required/documented by Apple iOS.
    - Prepare IPA Signing as required – please refer to link

<https://docops.ca.com//display/DTS90/Preinstallation+Steps+for+Mobile+Testing#PreinstallationStepsforMobileTesting-PrepareIPASigning(MacOnly)>

**Mobile App builds For Android:**

* .apk file needs to be built with debug options
* Application APK file can run on a device or emulator with not configuration changes
* Ensure the backend services URL is configured appropriately as required by the QA Team.
* To run the Android testcases on the Devices
* Please ensure you have installed the Android SDK and configured it as per the product documentation
* Please ensure the corresponding Device Drivers are installed
* Ensure you are able to see the device listed once you run the
  + adb devices

**Mobile App builds For iOS:**

* Please provide a **.app file** which needs to be **debug build**
* To run the Testcases in iOS-Simulator – the build file should be **Simulator build**
  + Please ensure the iOS-Simulator is loading the .app file manually in case you are facing any issue.
* To run the Testcases on Mobile Device
  + Please get the certificates as need and defined by Apple and set up the certificates in the System that needs to run the testcases on the device.
  + Connect the Device to the System and please verify if the .app file is getting loaded into the Device (QA Team member with DevTest Mobile knowledge can help here).

# Bamboo Setup

**Creation of Repository:**

Bamboo needs a Repository to use, so First create a Repository in BitBucket

* Create a BitBucket Account  
  Open <https://bitbucket.org/account/signup/> in your browser.  
  Sign up for a new Account or Sign up with Google Account
* Note: Remember your BitBocket Login Credentials for future use.
* Confirm your email address  
  When you create an account, Bitbucket sends an email to you to confirm your email. Confirming your email allows Bitbucket to automatically match your user account to your future source code commits.
* Create Repository

Log into Bitbucket Cloud under your individual account.

Click Repositories > Create repository or the Create new repository link.

Choose a repository Owner.

This only appears if you are creating under an account with membership in one or more teams.

Enter a Name and Description for your repository.

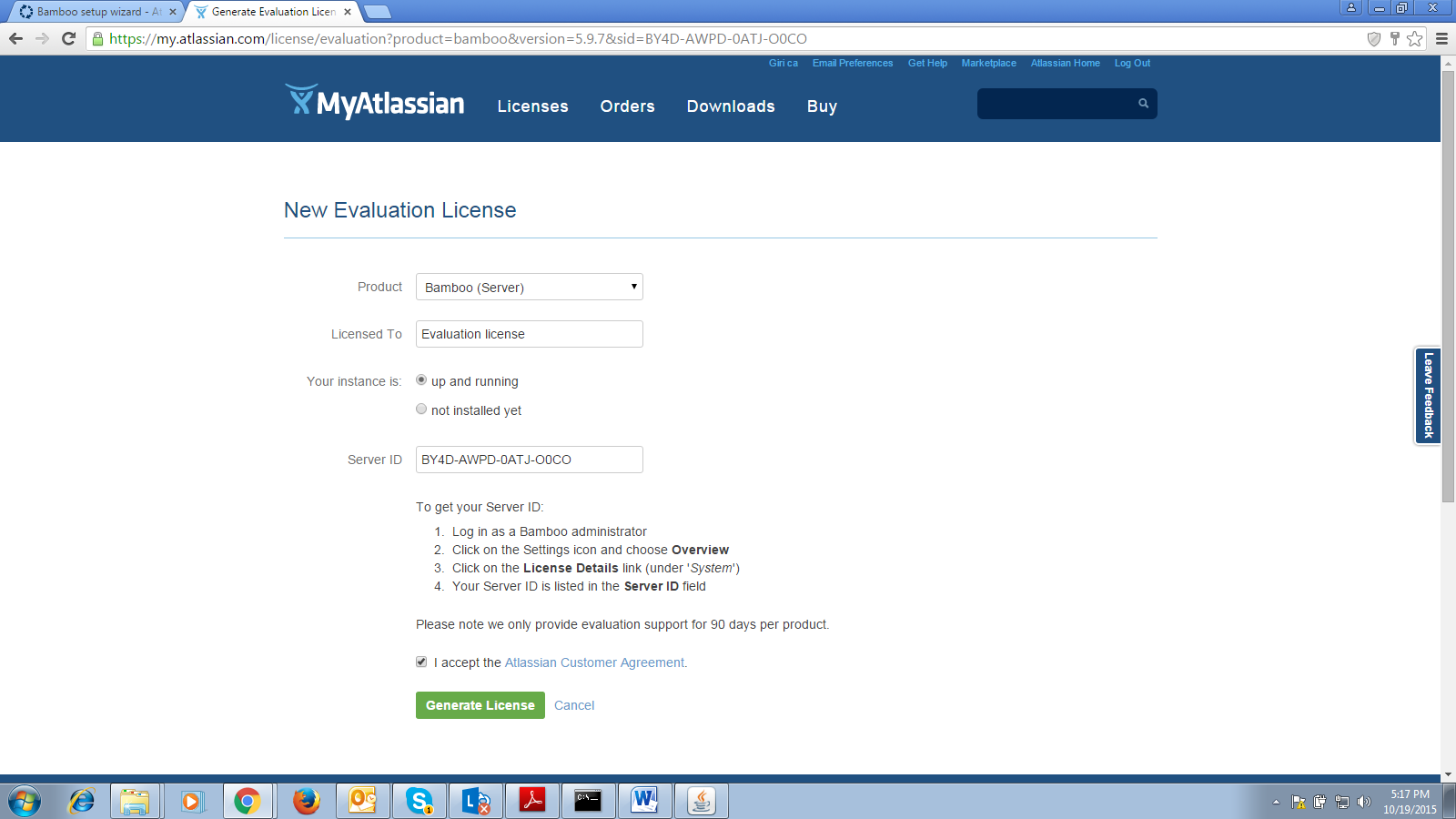
Tick Private if you want to hide your repository from the general public, so that only selected people can see it.

Select the Repository type.

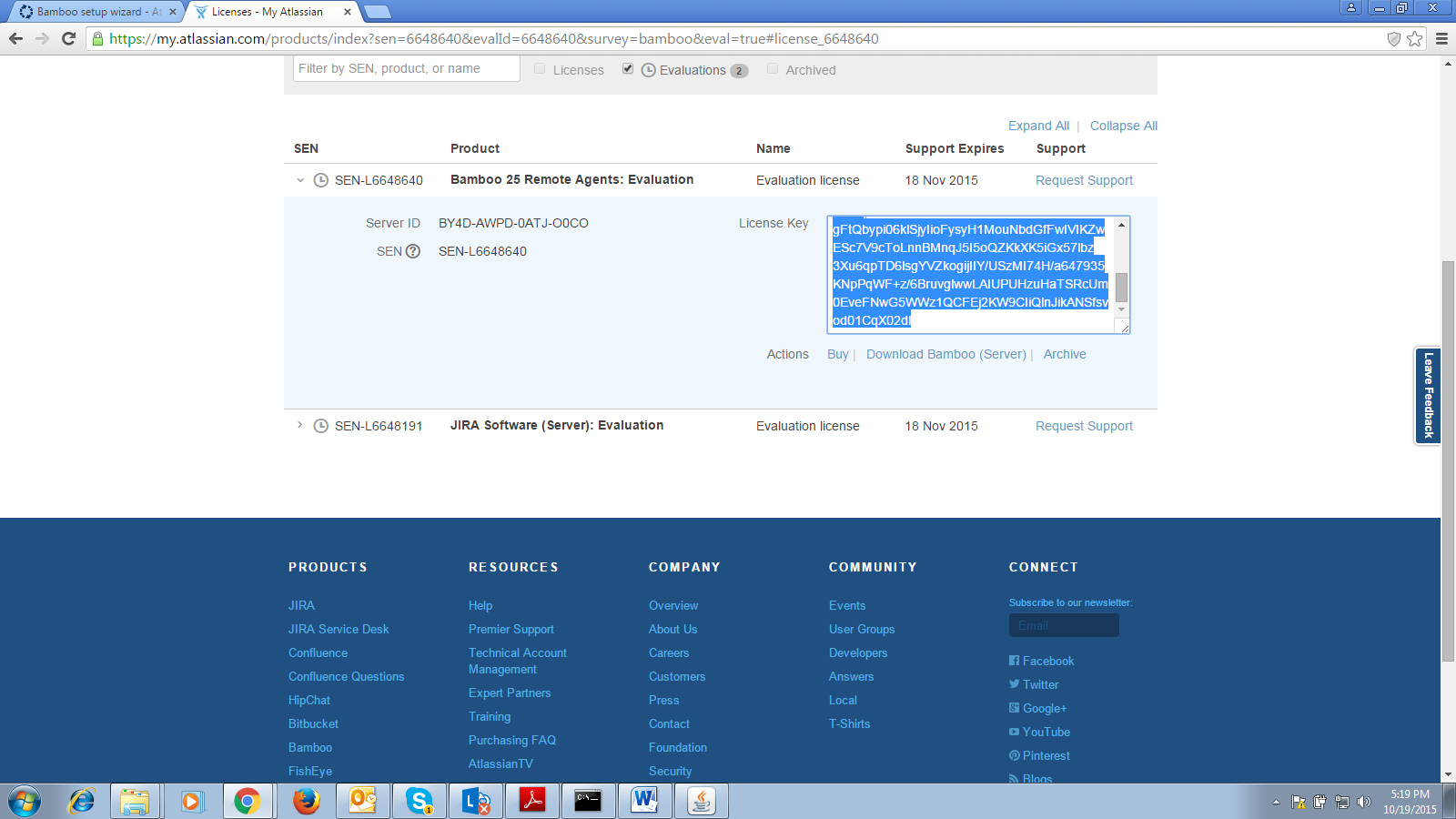
Click Create repository.

**Bamboo Installation and Set-Up**

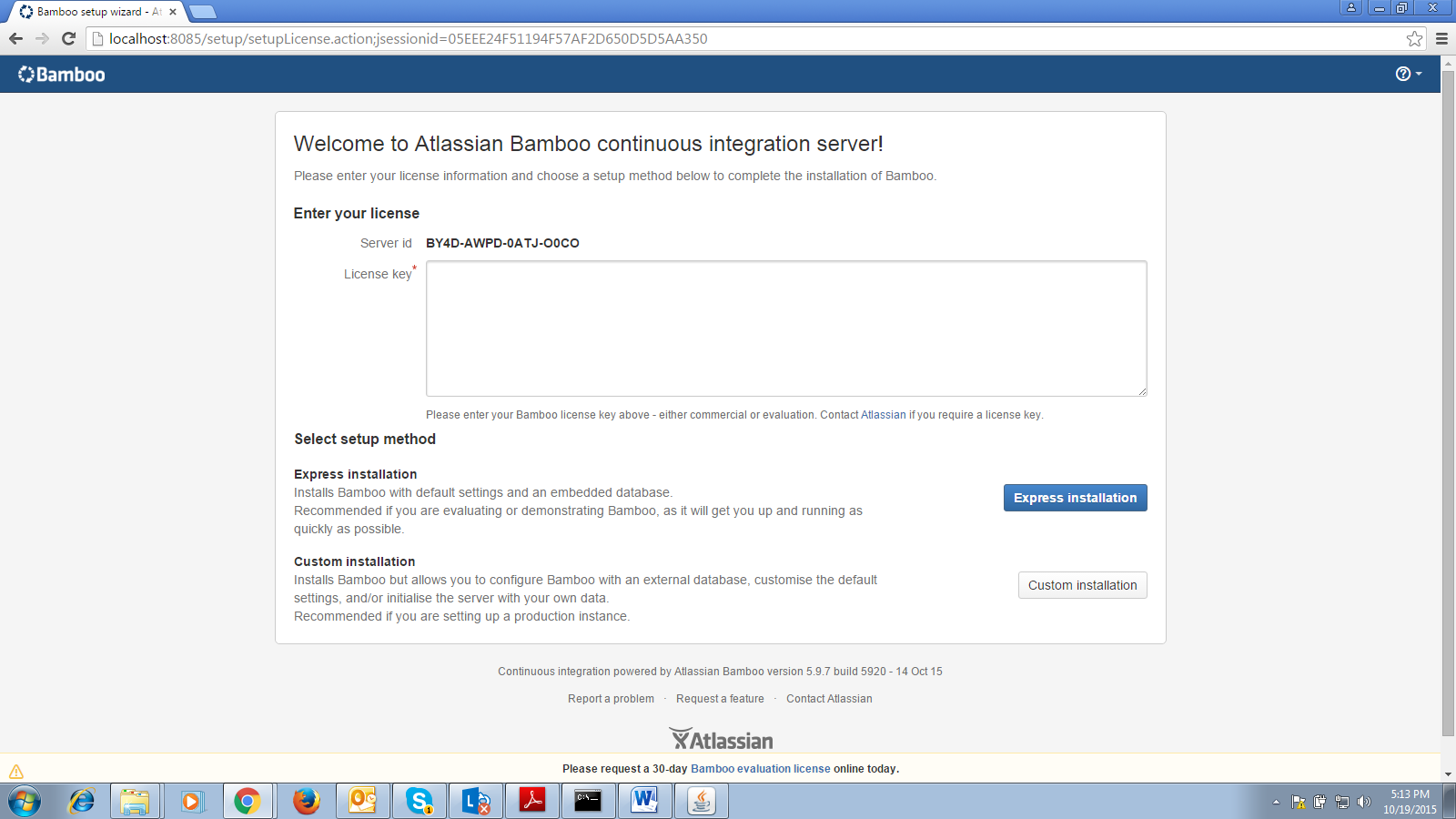
* Ensure that we have JDK (Not JRE)
* Download Bamboo  
  <https://confluence.atlassian.com/display/BAMBOO055/Installing+Bamboo+on+Windows#InstallingBambooonWindows-5.DownloadBamboo>
* Specify Destination directory and Bamboo Home directory
* Select all default options and finish the installation
* Once Installation Completed, double click on D:\Program Files\Bamboo\bin\start-bamboo.bat
* Wait till it start
* Access to <http://localhost:8085/>
* Create Evaluation license key by clicking on Atlassian link and login by using your credentials

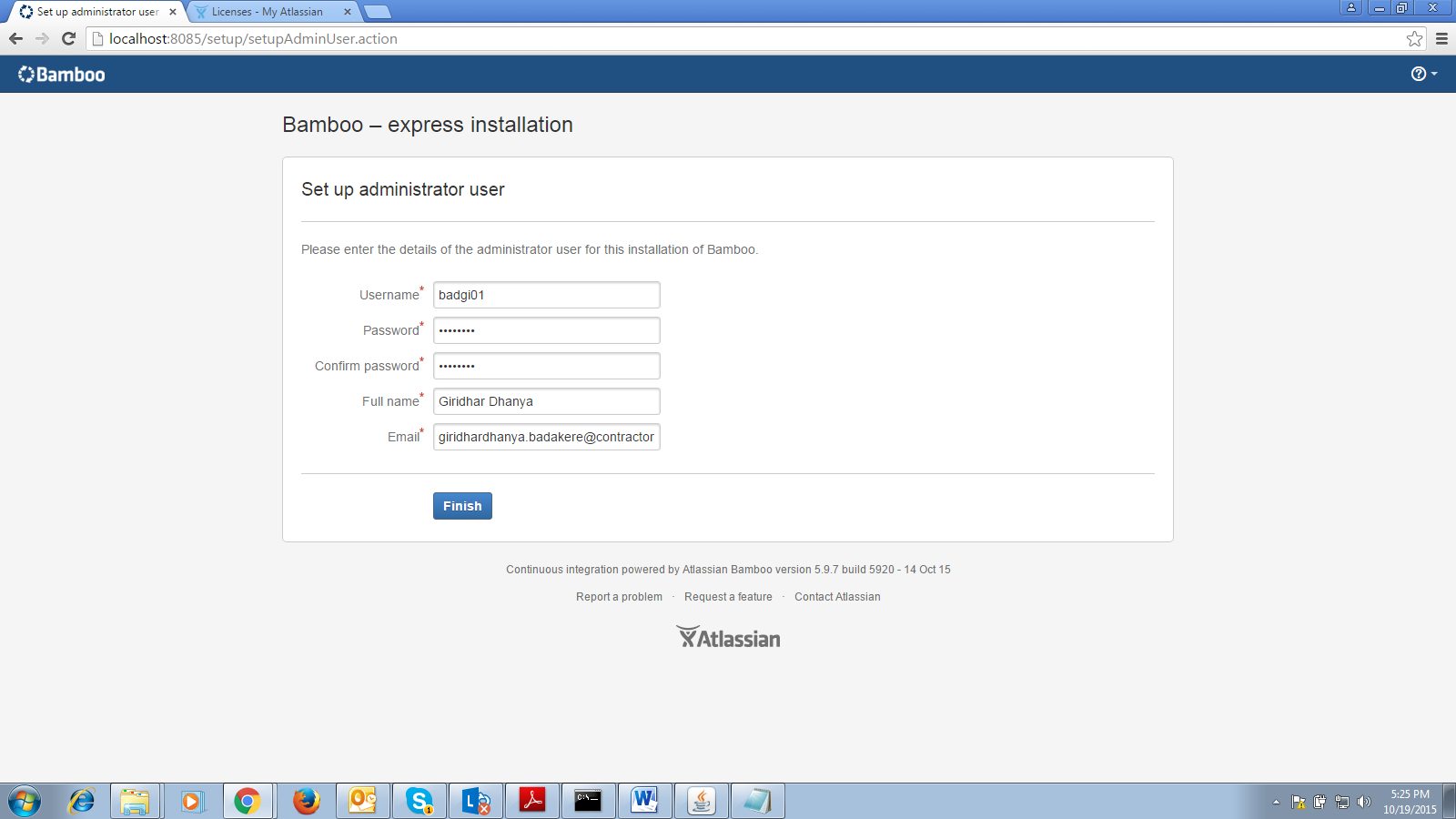


* Click on Generate License and copy License to use in Installation

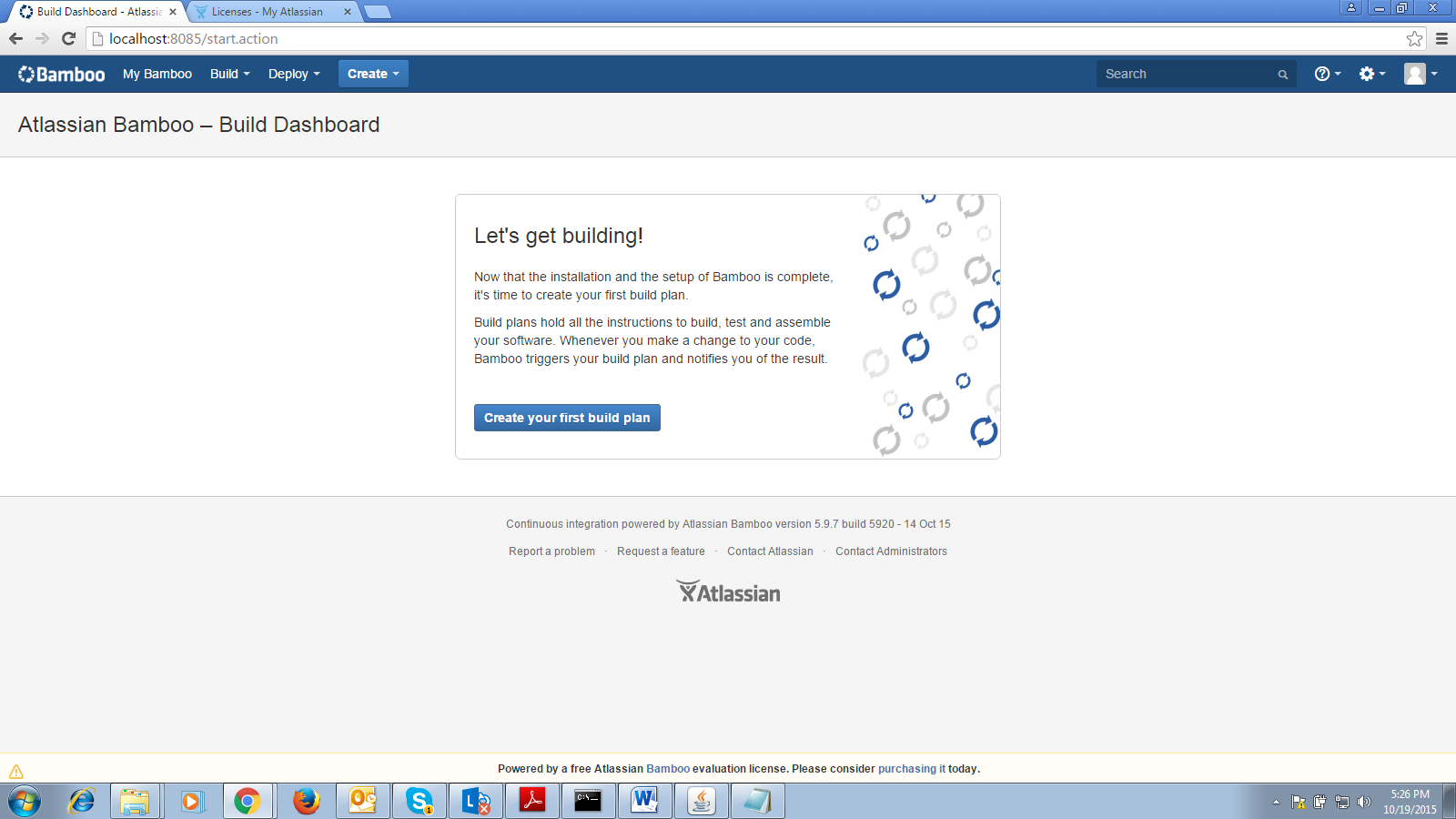


* Select “Express Installation” by using the above created Evaluation License

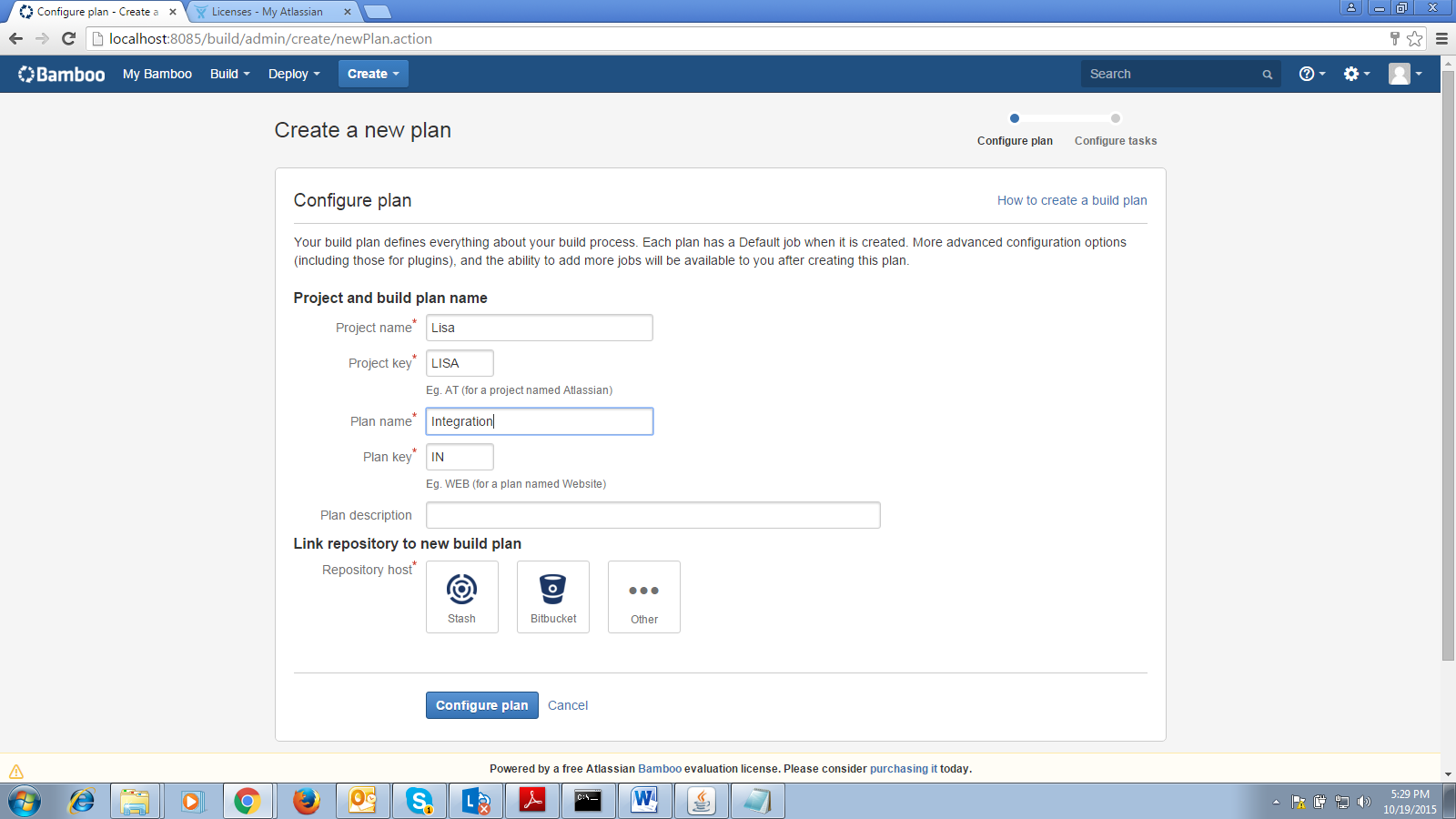




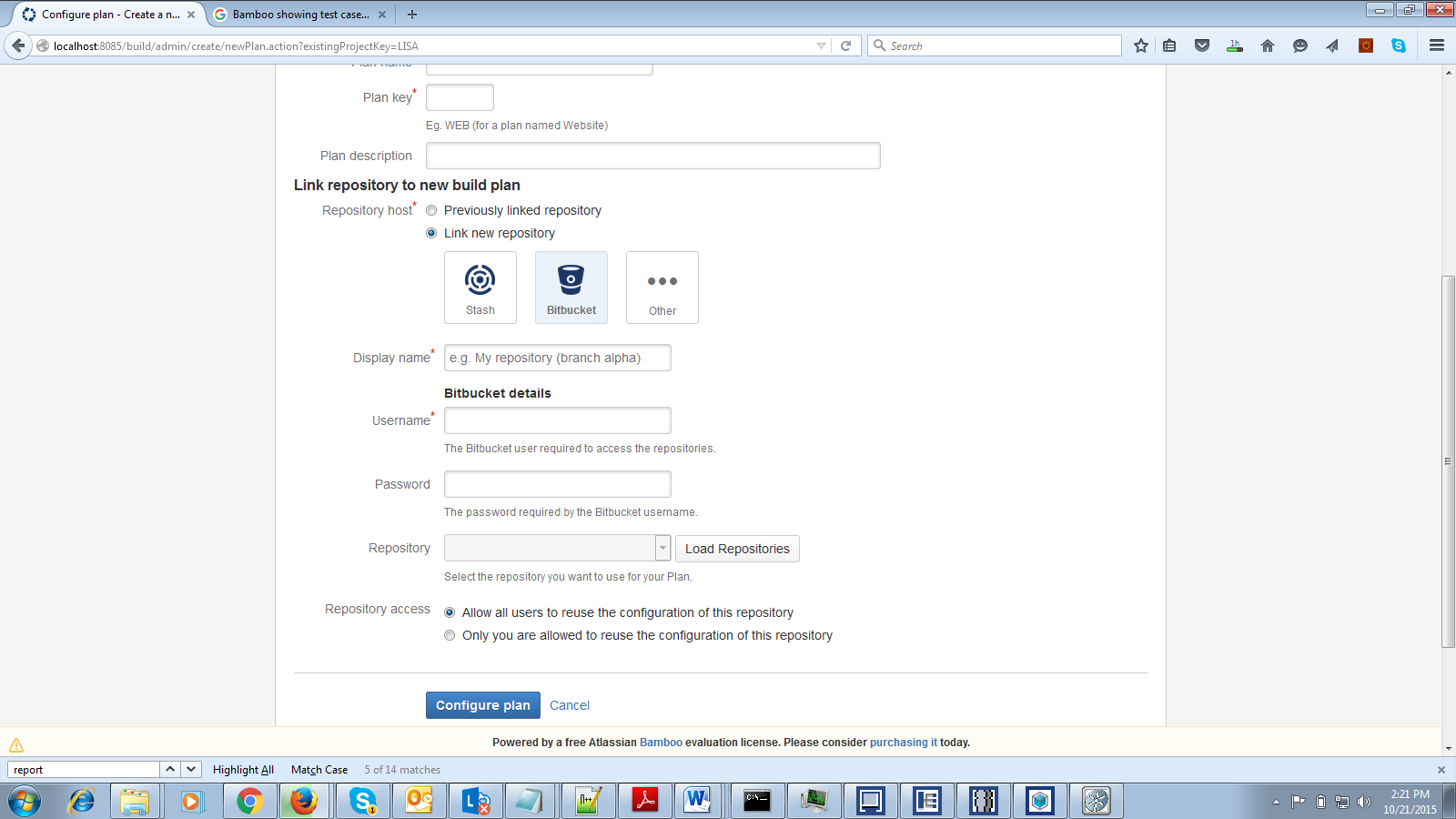
* Click on “Create your first Build Plan”



* In “Create a new plan” page , give all required details and Select BitBucket as Repository

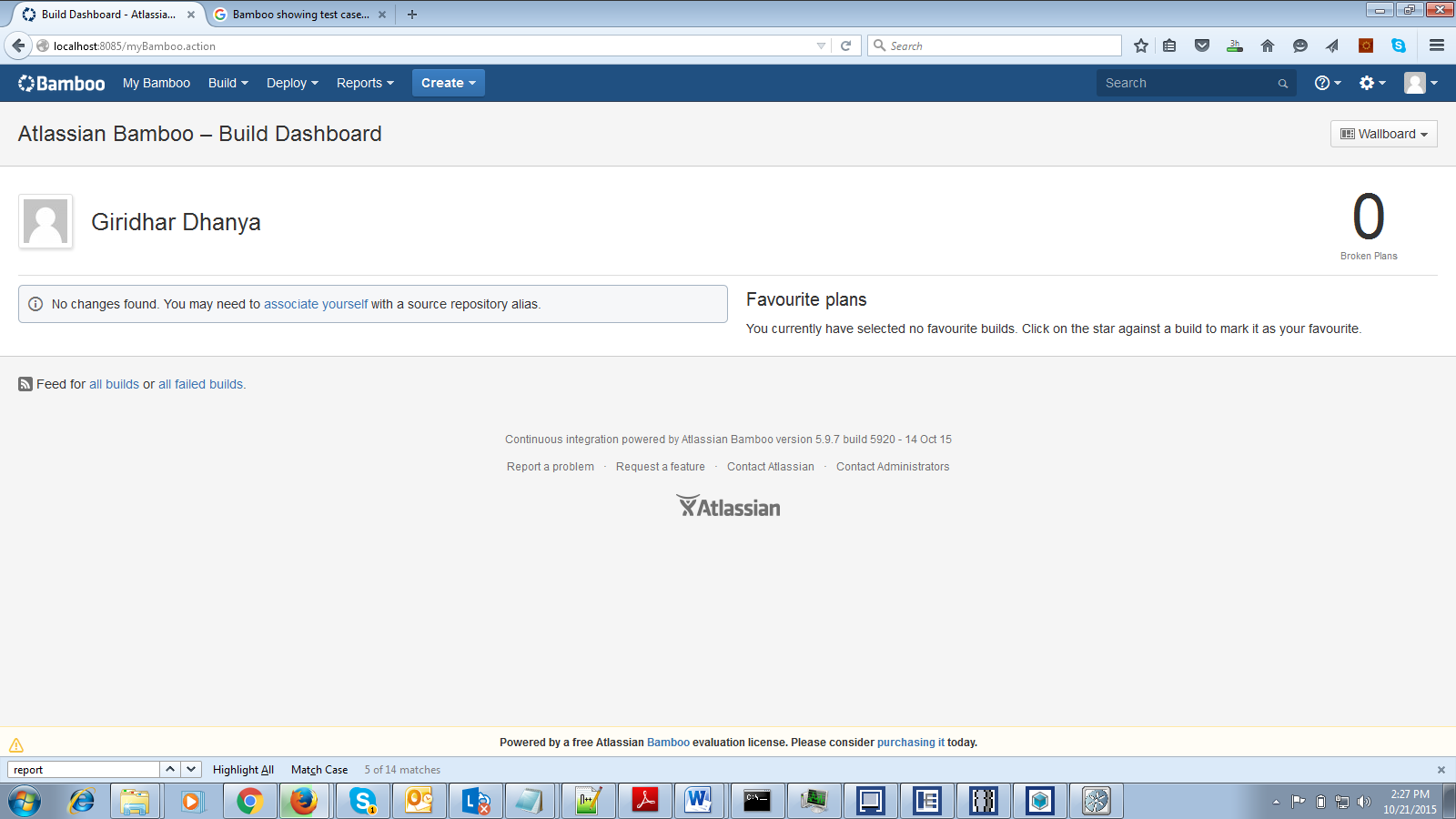


* Select BitBucket and provide Login Credentials for BitBucket (Created Above) and click on “Configure Plan”

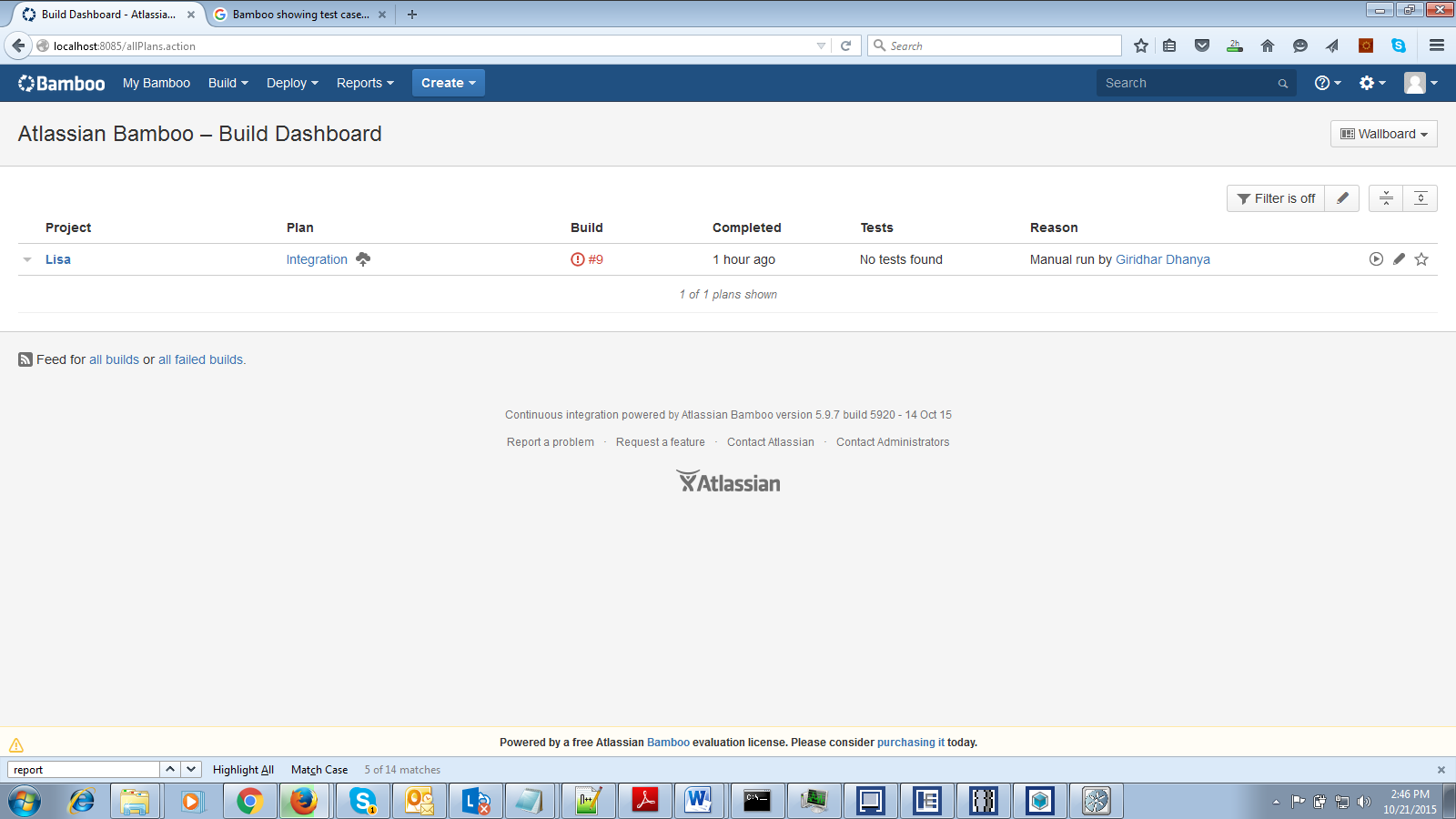


## Integrating Lisa and Bamboo

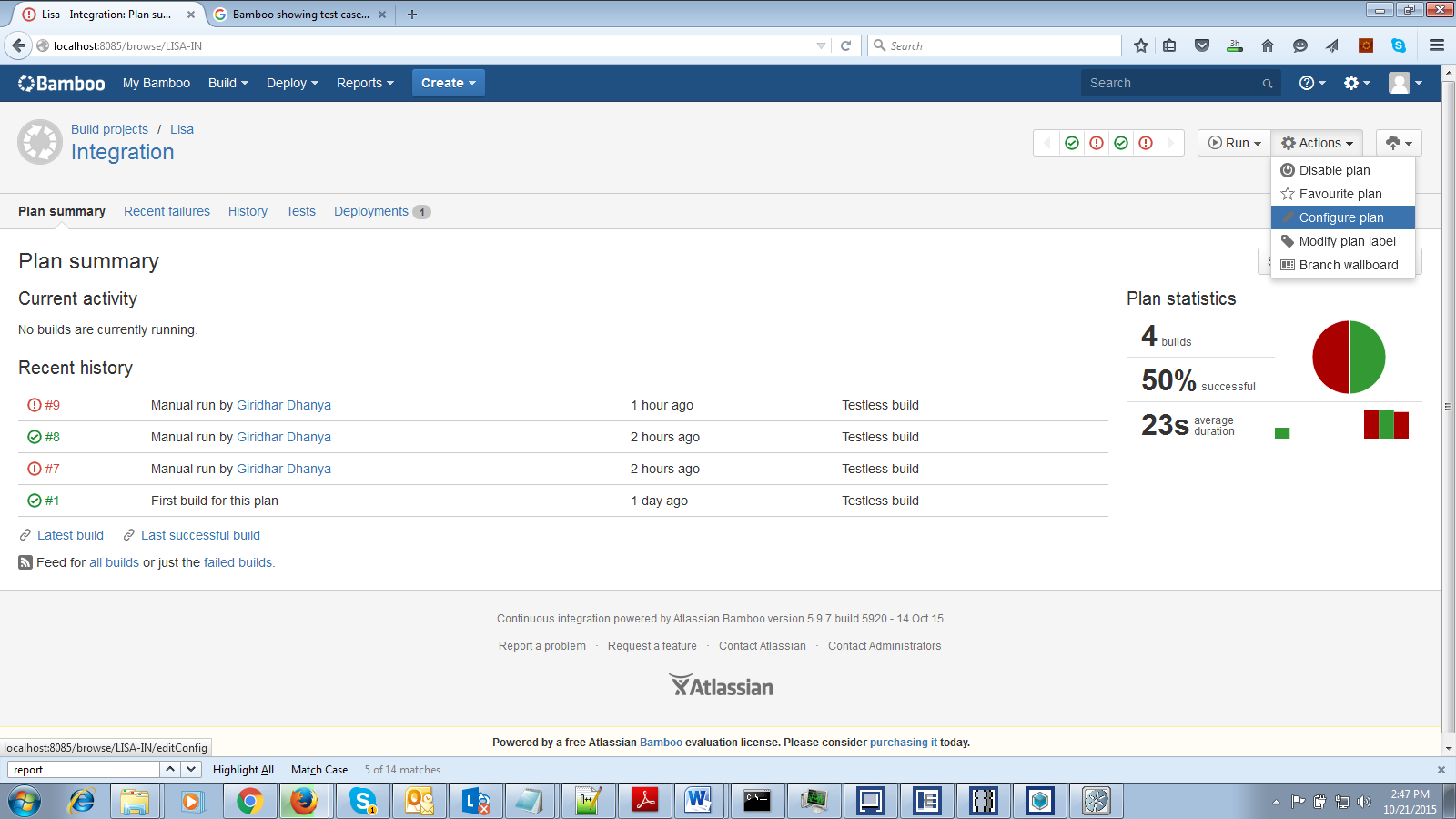
* Start Bamboo



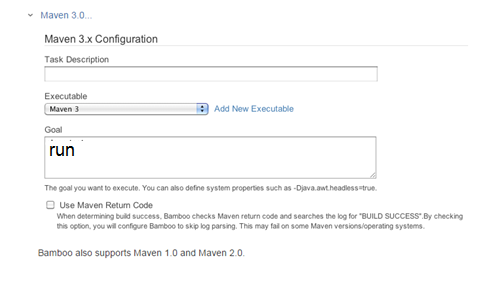
* Navigate to Build/ All Build Plans
* Click on your created plan



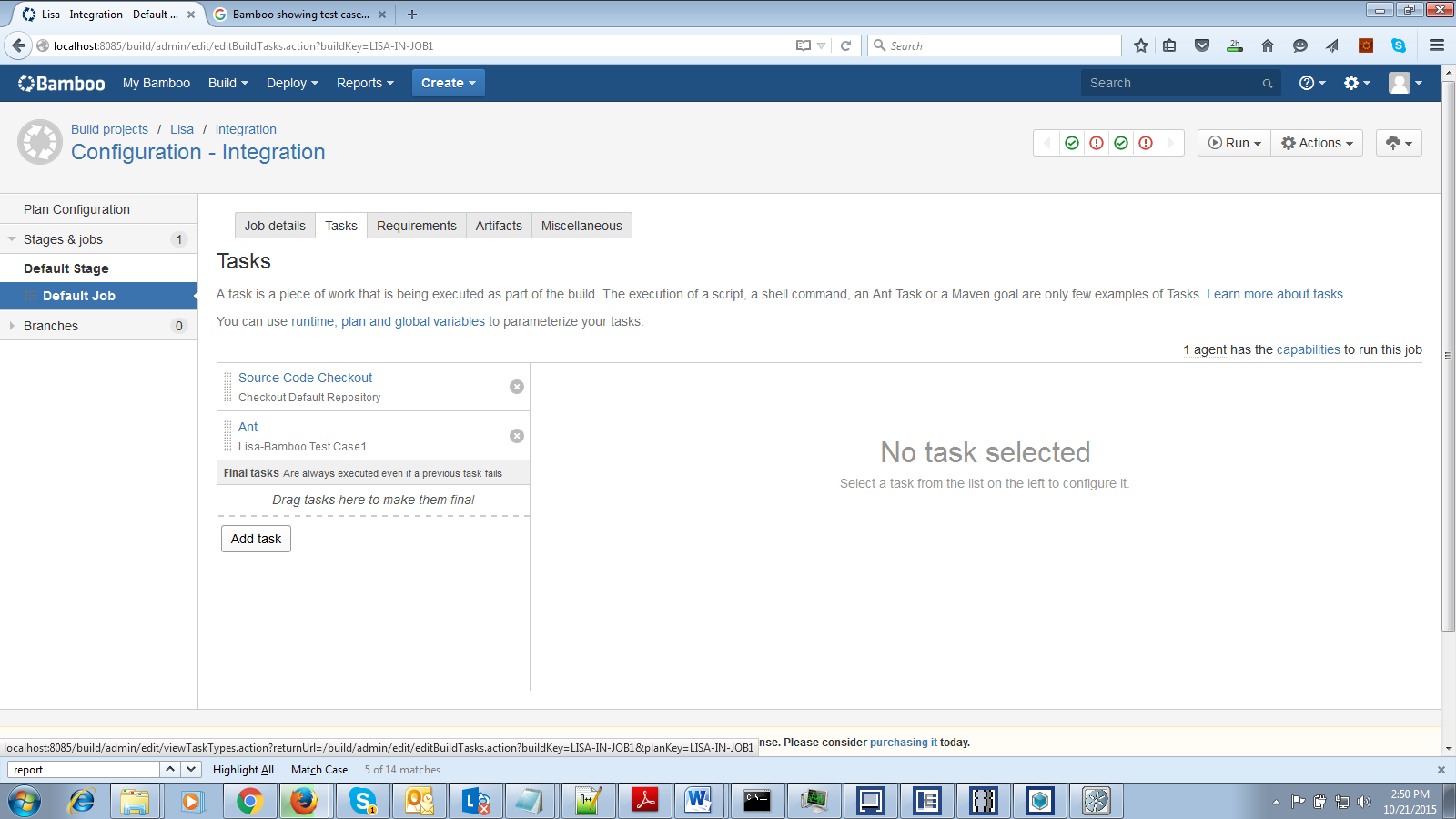
* Click on Actions / Configure Plan



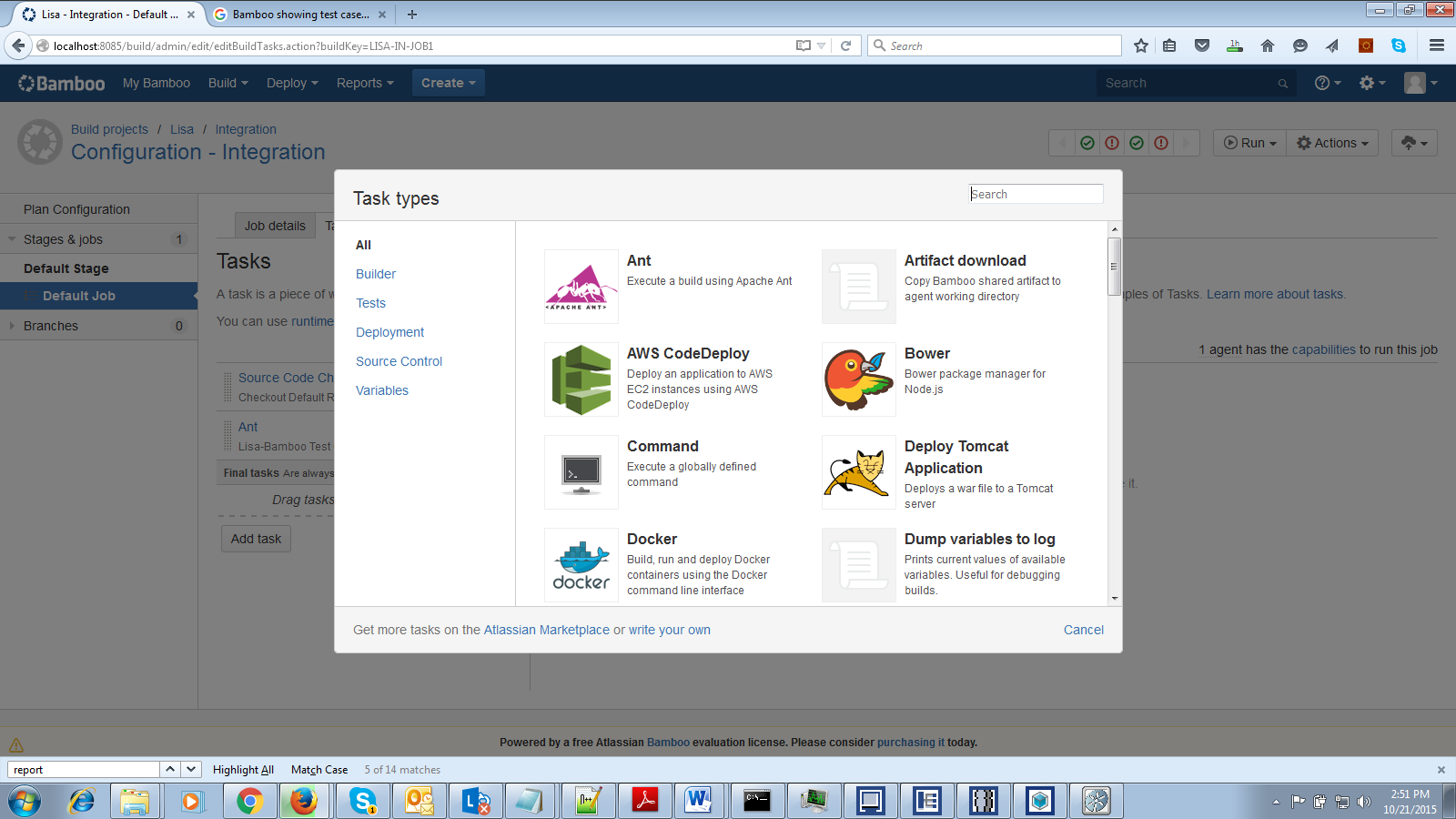
* Click on “Default Job” , “Add Task”
* If you want to use maven, then you need to use maven to call the Ant which in turn stages the DevTest test suite



You have to modify the pom.xml to call the ant task in the ‘run’ goal



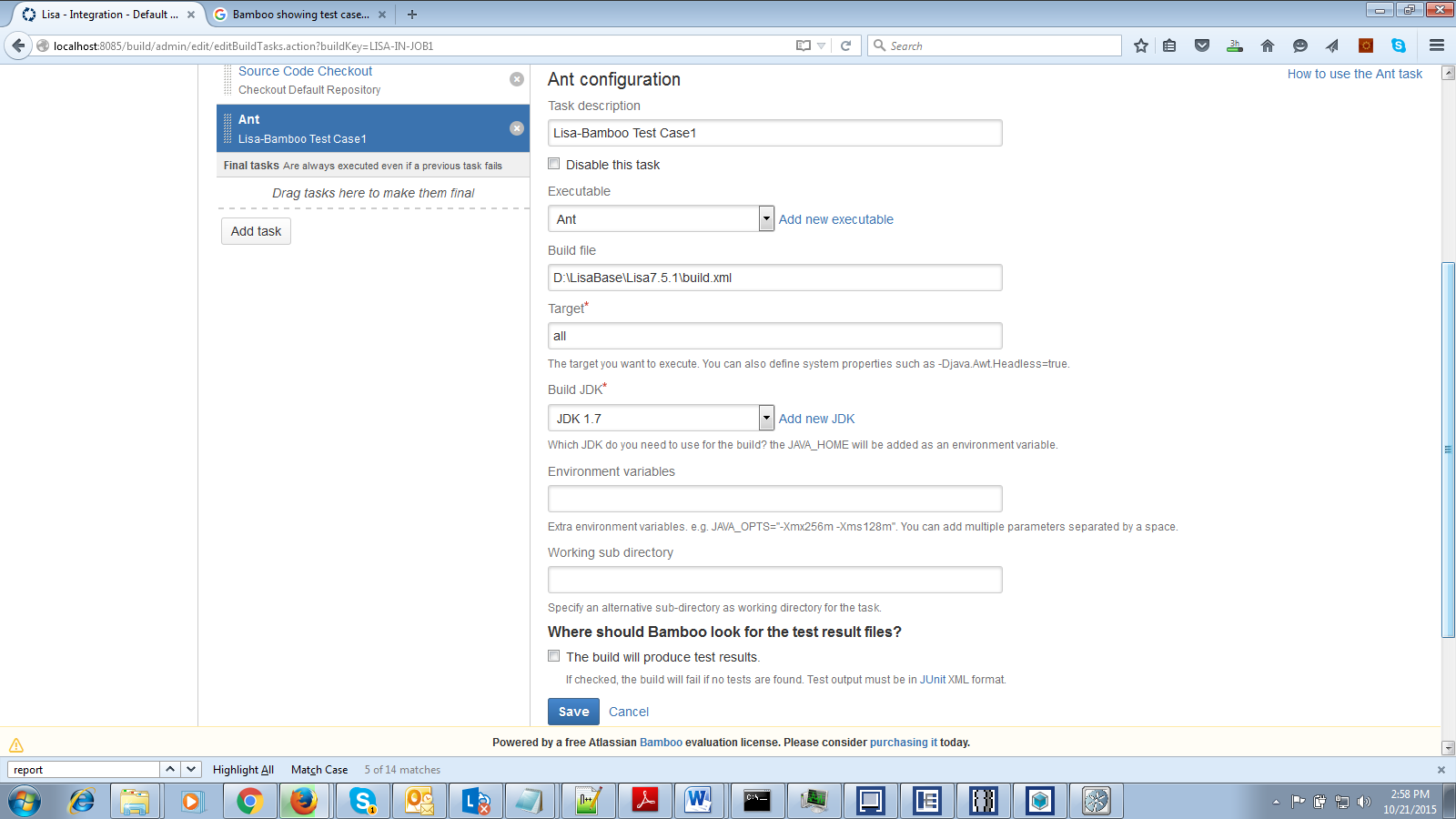
* Select “Ant”



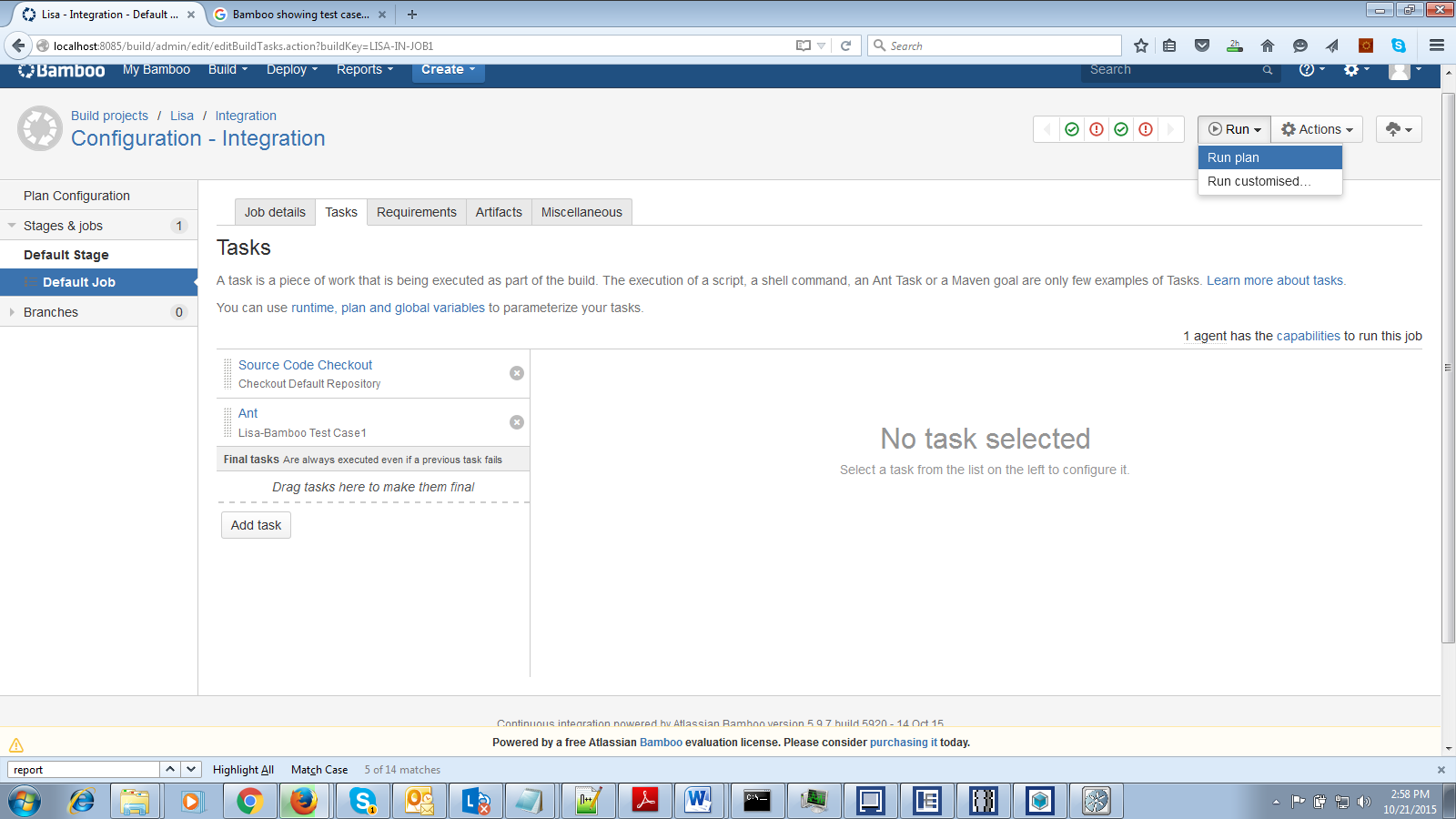
* Set Ant Configuration Page  
  Build file (Important)

Provide the Ant build file location. We can provide the build file that can trigger multiple projects or the build file of single project to just trigger the build.xml of that particular project.

Target: all

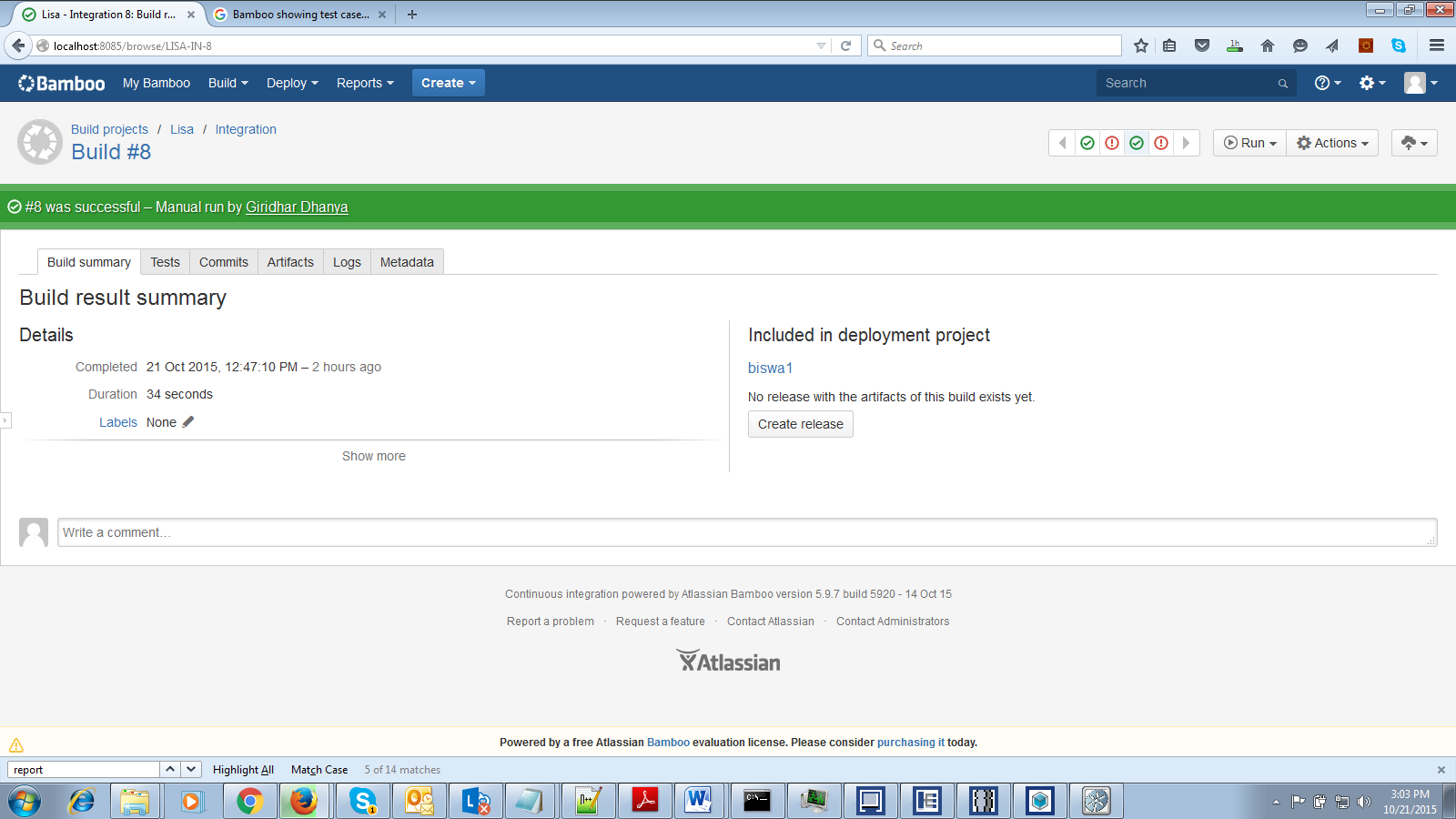


* Note: Please start all Lisa Services and Work Station.
* Once Ant Task set-up completed, Click on Run / Run Plan

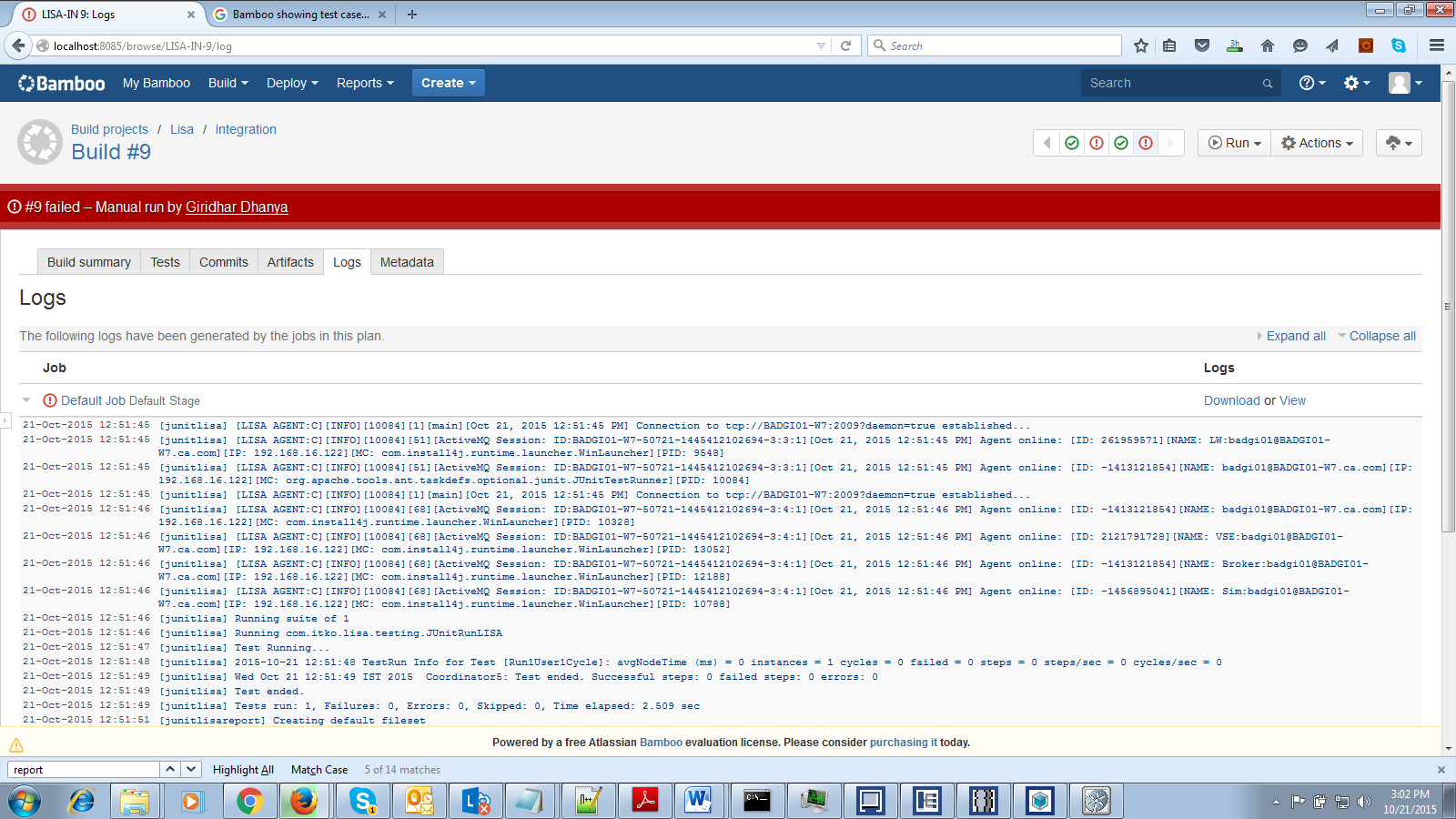


* The job number appears below and show whether the job is executed successfully or not.

If the Bamboo build job completes successfully then it shows up in ‘Blue’ Color else if the build fails it will show up in ‘Red’ Color.



* If it Fails , then we can get Error details in Log tab



## Sample Maven script to call Ant script:

In the pom.xml we need to add the following information to call the ant Script

<plugin>

<artifactId>maven-antrun-plugin</artifactId>

<executions>

<execution>

<id>myExecution</id>

<phase>deploy</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<ant target=" LisaTest">

...

</ant>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

## Lisa Projects and sample Ant build scripts:

We will use Bamboo to run Lisa Projects. We can run a single Lisa project or we can run multiple Lisa projects at a time. To do this, we need to integrate Bamboo and Lisa. You need to create a build.xml in the Lisa Project folder for the activities to be done for that project.

For example, the build.xml can mention the test cases that need to run, the settings for that particular test case to run. Please find the sample build.xml files that mention LISA test cases which can be run as junit test cases.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<!--

This is an example of how to integrate a LISA project into a continuous build system such as Bamboo. It uses

the 'junitlisa' Ant task to run LISA tests, which converts the LISA test output to look like JUnit output. LISA can

run 'native' junit tests from within LISA tests; the reverse is also true - junit can run LISA tests, as we do here.

Most continuous build systems can invoke junit via ant, so this makes integrating your LISA tests a snap.

-->

<project name="LISA Example Ant/JUnit" default="lisaTests" basedir=".">

<property name="LISA\_HOME" value="${basedir}/.."/>

<property name="testReportDir" value="${LISA\_HOME}/reports/junit"/>

<!-- pull in our custom ant tasks -->

<taskdef resource="AntTasks.properties">

<classpath>

<fileset dir="${LISA\_HOME}/lib/core" includes="\*.jar"/>

<fileset dir="${LISA\_HOME}/lib/endorsed" includes="\*.jar"/>

<fileset dir="${LISA\_HOME}/lib/shared" includes="\*.jar"/>

</classpath>

</taskdef>

<target name="lisaTests" description="Executes a LISA Test Suite as JUnit tests." >

<!-- make sure old reports are deleted and that we have a place to report... -->

<delete dir="${testReportDir}"/>

<mkdir dir="${testReportDir}"/>

<!-- run the lisa suite in junit mode -->

<junitlisa toDir="${testReportDir}"

user="admin"

password="admin">

<lisatest suite="${basedir}/Suites/AllTestsSuite.ste"/>

</junitlisa>

<!-- generate junit style html report -->

<junitlisareport toDir="${testReportDir}"/>

<echo message="The JUnit report is available at ${testReportDir}/index.html"/>

</target>

<target name="oneTest" description="Executes a LISA Test as a JUnit test." >

<!-- make sure old reports are deleted and that we have a place to report... -->

<delete dir="${testReportDir}"/>

<mkdir dir="${testReportDir}"/>

<!-- run the lisa suite in junit mode -->

<junitlisa toDir="${testReportDir}"

test="Tests/Lisa\_config\_info.tst"

stagingDoc="StagingDocs/1User0Think\_RunContinuously.stg"

user="admin"

password="admin">

</junitlisa>

<!-- generate junit style html report -->

<junitlisareport toDir="${testReportDir}"/>

<echo message="The JUnit report is available at ${testReportDir}/index.html"/>

</target>

</project>

<!--

Custom LISA ant task documentation

junitlisa is a 'drop in' replacement for junit, but designed to execute LISA tests instead of junit tests. Most

continuous build systems recognise the xml output files and will integrate the build dashboard with the test

results. You can also produce HTML reports with the junitlisareport task (or the regular junitreport task if

you want).

junitlisa is an ant task that directly subclasses the regular junit task, so any of the attributes

and nested elements that junit has, junitlisa also has. See http://ant.apache.org/manual/OptionalTasks/junit.html

There are several differences though:

- you cannot set the 'fork' attribute to 'false' (we \*always\* fork)

- you can't add nested <test> elements; use a <lisatest test="testcase.tst"/> nested element or a

test="testcase.tst" attribute instead

- you can't add nested <batchtest> elements; use a <lisatest suite="suiteFile.tst"/> nester element or a

suite="suiteFile.ste" attribute instead

- we add an implied classpath consisting of LISA\_HOME/bin/\*.jar, LISA\_HOME/lib/\*.jar,\*.zip

and LISA\_HOME/lib/endorsed/\*.jar

- we add an implied java.endorsed.dirs system property pointing to LISA\_HOME/lib/endorsed

- if no formatter is specified then we add a default formatter of type "xml"

- we default the 'showoutput' attribute to 'true'

- we default the 'printsummary' attribute to 'true'

- we default the 'maxmemory' attribute to '1024m'

junitlisa supports the following attributes in addition to the attributes inherited from junit

- suite="suitename.ste" - a LISA suite. This is the most common use case

- test="filename.tst" - a single LISA test to run

- stagingDoc="filename.stg" - the staging doc to use (will default to {{LISA\_HOME}}/defaults/Run1User1Cycle.stg )

- config="configname" which is either a named internal configuration set or a filename (.config or .properties file)

- outfile="filename" specify the filename that junitlisa uses to write reporting data. NOTE if "filename" doesn't comply with standard

naming scheme for <junitlisareport> you will need to specify a fully configured <junitreport> task instead

- registry="registryName" e.g "tcp://testbox:2010/Registry" - a pointer to the registry you want to use if

you want to stage the tests remotely. If you don't specify this or the registry name is 'local', the tests

will be staged locally

- preview="true" - output the name and description of each test case but do not execute them

lisatest nested element supports the following attributes

- suite="suitename.ste" - a LISA suite. This is the most common use case

- test="filename.tst" - a single LISA test to run

- stagingDoc="filename.stg" - the staging doc to use (will default to {{LISA\_HOME}}/defaults/Run1User1Cycle.stg )

All the attribute values can use {{curlybraces}} and LISA will resolve them in the usual fashion.

You must specify at least one test or suite, either in a nested lisatest element or the 'test' or 'suite'

attribute. Multiple tests and/or suites can be specified by adding more nested lisatest elements. The tests

and suites will be executed in the order in which they appear in the XML.

'config' and 'testRegistry' attributes are optional.

You will almost always want to specify the inherited 'toDir' attribute, although it will default to

the current working directory.

When you run a single test with the 'test' attribute, it is staged with a single vuser, run once and has 100%

think time. If you need to change this, wrap the test in a suite and specify an alternative staging doc.

junitlisareport is a subclass of the regular junitreport element except that we specify sensible defaults.

It is exactly equivalent to the following:

<junitreport todir="${testReportDir}">

<fileset dir="<todir specified in the junitreport tag>">

<include name="TEST-\*.xml"/>

</fileset>

<report format="frames"

todir="<todir specified in the junitreport tag>">/>

</junitreport>

You can specify your own fileset and report if you want though... and because it's a direct subclass of

junitreport, we also support all the attributes and nested elements that junitreport does.

You will almost always want to specify the 'toDir' attribute of the junitlisareport element however it

will default to the current working directory.

-->

For running multiple projects from a single build file, we need to ensure that we mention the targets and the build files locations in a single file such that the other project specific build.xmls are triggered. Please find the sample common build.xml .

<project name="LISAHOME" basedir=".">

<echo> BasedirMain path ${basedir} </echo>

<property name="Examples" value="D:/Lisa/examples" />

<import file="D:/old\_d\_drive/Work/Projects/Ameriprise-2014/06.LisaProjects/02.InProgress/POC/New\_LISA\_Project\_00/build.xml"/>

<target name="all">

<!--<subant target="oneTest">

<filelist dir="D:/LisaBase/Lisa7.5.1/examples" files="build.xml"/>

</subant> -->

<subant target="LisaTest">

<filelist dir="D:/LisaBase/Lisa7.5.1/POC" files="build.xml"/>

</subant>

</target>

</project>

## Reports :

Lisa Test cases run as junit test cases to produce reports if we use ANT. If we look at the ‘junitlisa’ with in the build.xml of every project, we can find the ‘testReportDir’ where the reports are saved. The reports are properly segregated depending on the success, fail or error out test cases. The reports are visible in HTML format rather than plain text which will give a better visibility.

***Example:***

*<target name="oneTest" description="Executes a LISA Test as a JUnit test." >*

*<!-- make sure old reports are deleted and that we have a place to report... -->*

*<delete dir="${testReportDir}"/>*

*<mkdir dir="${testReportDir}"/>*

*<!-- run the lisa suite in junit mode -->*

*<junitlisa toDir="${testReportDir}"*

*suite="${basedir}/Suites/AllTestsSuite1.ste"*

*test="${basedir}/Tests/Lisa\_config\_info.tst"*

*stagingDoc="${basedir}/StagingDocs/Run1User1Cycle.stg"*

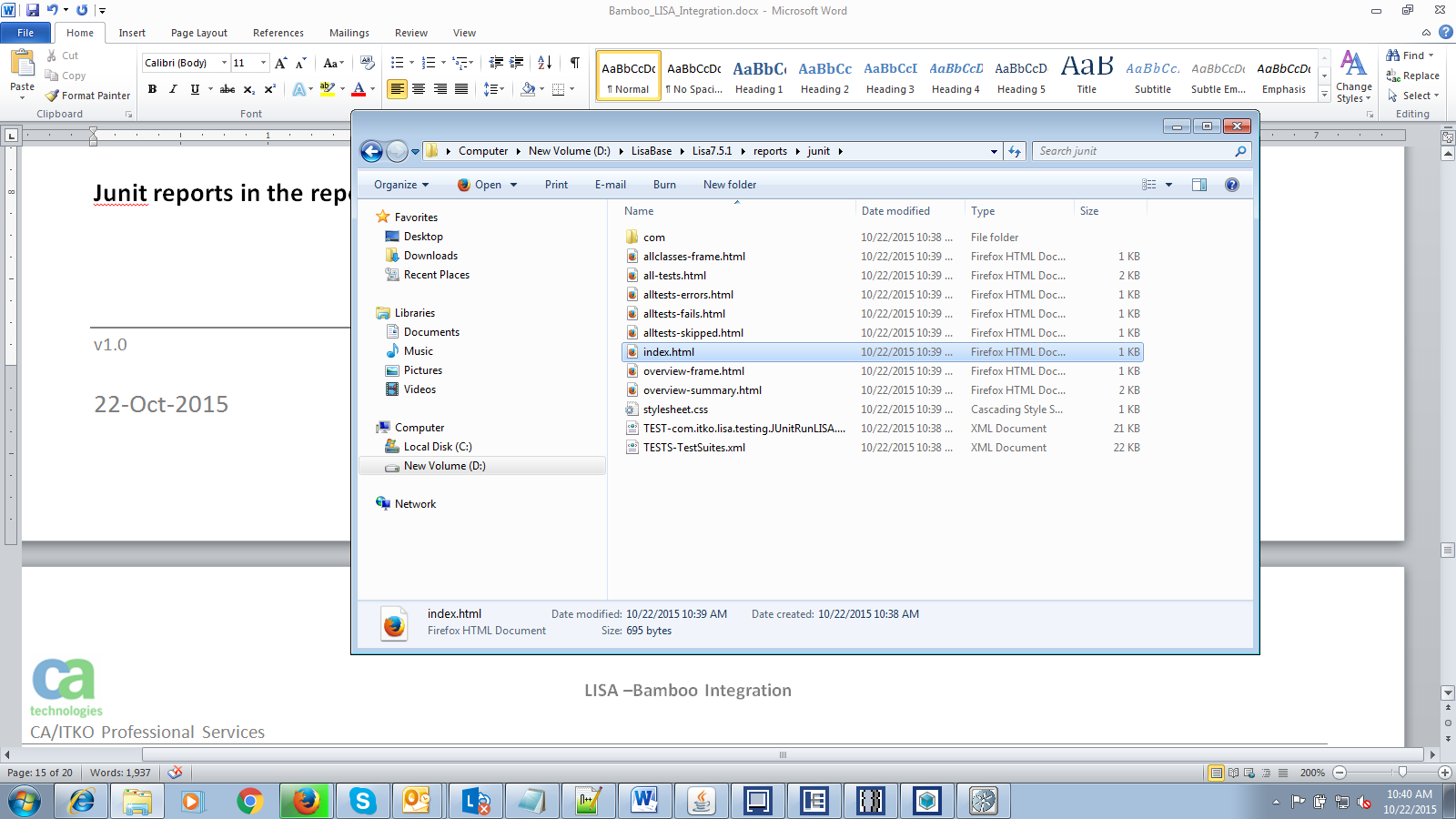
*user="admin"*

*password="admin">*

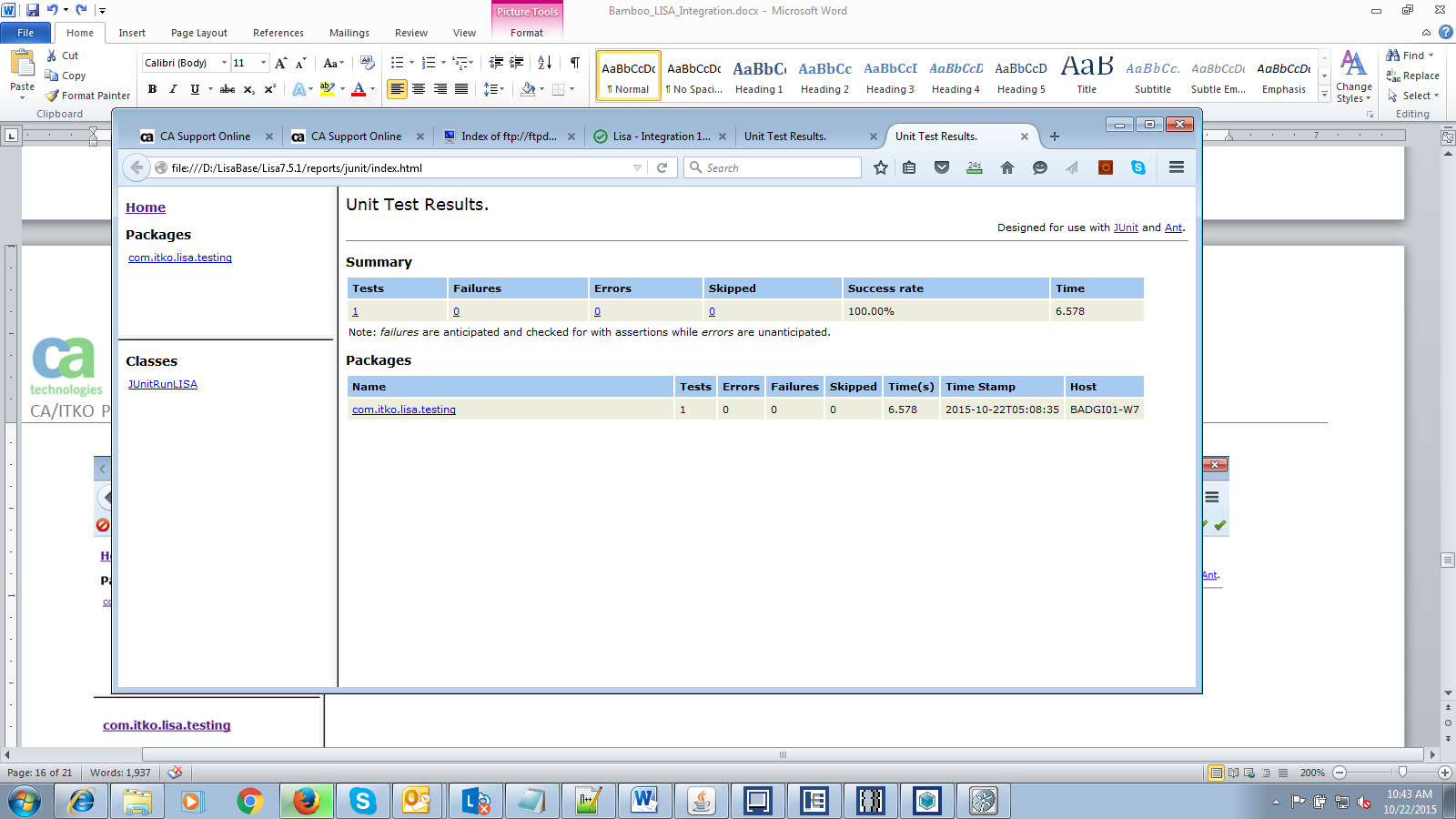
*<jvmarg value="-Dlisa.tmpdir=${lisa.temp.dir}"/>*

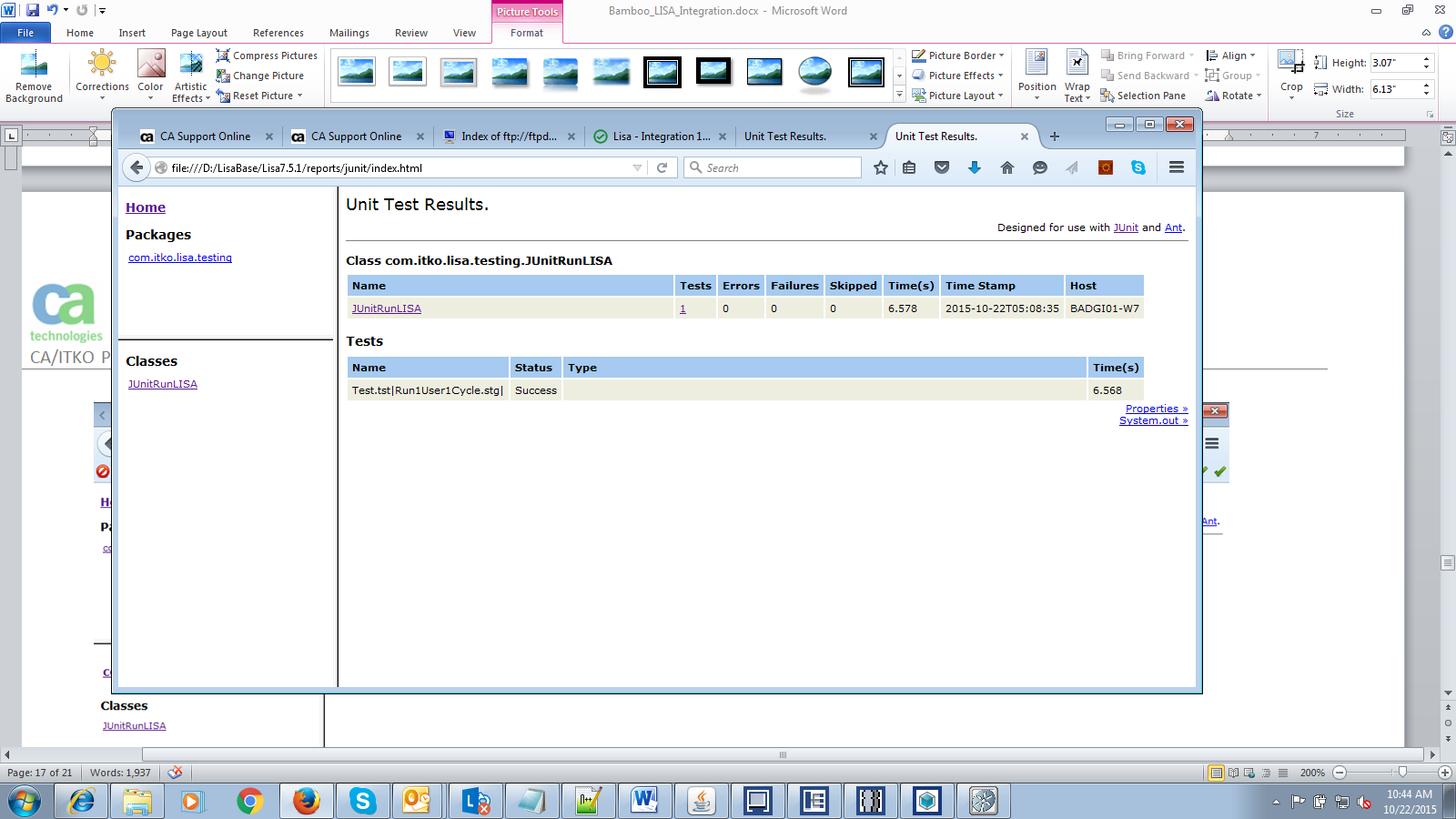
*</junitlisa>*

**Junit reports in the reports folder of Lisa**



**Sample screenshot of the reports**





We can also view reports of the test cases that have run in Lisa Server ‘Reporting’.

**Sample screenshot of the Lisa reports**



# LISA Ant Tasks

The following Ant tasks are included with LISA:

■ junitlisa

■ junitlisareport

## junitlisa Ant Task

The **junitlisa** task is a "'drop in" replacement for the JUnit task available with Ant, but it executes LISA tests instead of JUnit tests. Most continuous build systems recognize the XML output files and integrate the build dashboard with the test results.

This task is a direct subclass of the JUnit task. Therefore, the junitlisa task has the same attributes and nested elements as the JUnit task. However, be aware of the following differences in behavior:

■ You cannot set the **fork** attribute to false.

■ You cannot add nested **test** elements. Use the **test** attribute instead.

■ You cannot add nested **batchtest** elements. Use the **suite** attribute instead.

■ An implied classpath consisting of **LISA\_HOME/bin/\*.jar**, **LISA\_HOME/lib/\*.jar, \*.zip**, and **LISA\_HOME/lib/endorsed/\*.jar** is added.

■ An implied **java.endorsed.dirs** system property pointing to **LISA\_HOME/lib/endorsed** is added.

■ If no formatter is specified, then a default formatter of type **xml** is added.

■ The **printsummary** attribute is defaulted to true.

■ The **maxmemory** attribute is defaulted to 1024m.

■ The **showoutput** attribute is defaulted to true.

In addition to the attributes inherited from the JUnit task, the **junitlisa** task has the following attributes:

**suite**

The file name of a suite document.

**Example:** suite="AllTestsSuite.ste"

**test**

The file name of a test case.

**Example:** test="multi-tier-combo.tst"

**stagingDoc**

The file name of a staging document.

**Example:** stagingDoc="Run1User1Cycle.stg"

**config**

A named internal configuration set or a file name.

**Example:** config="project.config"

**Outfile**

The file name that is used to write reporting data. If the value does not comply with the standard naming scheme for junitlisareport, specify a fully configured junitreport task instead.

**Example:** outfile="report"

**registry**

A pointer to the registry to use when you want to stage the test cases remotely.

**Example:** registry="tcp://testbox:2010/Registry"

**preview**

Enables you to write out the name and description of each test case, without executing the test cases.

**Example:** preview="true"

**user**

If access control (ACL) is enabled, you can use this attribute to specify the user name.

**Example:** user="admin"

**password**

If access control (ACL) is enabled, you can use this attribute to specify the password.

**Example:** password="admin"

**mar**

The file name of a MAR document.

**Example:** mar=”example.mar”

**mari**

The file name of a MAR info document.

**Example:** mari=”example.mari”

The **junitlisa** task includes a nested element named **lisatest**. This element has the following attributes:

**suite**

The file name of a suite document.

**Example:** suite="AllTestsSuite.ste"

**test**

The file name of a test case.

**Example:** test="multi-tier-combo.tst"

**stagingDoc**

The file name of a staging document.

**Example:** stagingDoc="Run1User1Cycle.stg"

**mar**

The file name of a MAR document.

**Example:** mar=”example.mar”

**mari**

The file name of a MAR info document.

**Example:** mari=”example.mari”

The attribute values can use curly braces, which are resolved in the usual way.

You are required to specify at least one test or suite. You can specify a test or suite in a **lisatest** nested element or in the **test** or **suite** attribute. You can specify multiple tests and suites by adding more **lisatest** elements. The tests and suites are executed in the order in which they appear in the XML.

When you run a single test with the **test** attribute, the test has the following default behavior:

■ Staged with a single vuser.

■ Run once.

■ 100 percent think time.

To change this default behavior, wrap the test in a suite and specify an alternative staging document.

**Example: junitlisa Task for Test Case**

The following **junitlisa** task is configured to run a single test case on a remote registry.

<junitlisa test="MyTest.tst"

config="dev"

registry="tcp://testbox:2010/Registry"

toDir="${testReportDir}"

haltOnError="no"

errorProperty="test.failure">

<jvmarg value="-DmySystemProp=someValue"/>

</junitlisa>

## junitlisareport Ant Task

You can produce HTML-based reports with the **junitlisareport** task or the regular **junitreport** task.

The **junitlisareport** task is a subclass of the regular **junitreport** element, except that sensible defaults are specified. It is equivalent to the following code:

<junitreport todir="${testReportDir}">

<fileset dir="<todir specified in the junitreport tag>">

<include name="TEST-\*.xml"/>

</fileset>

<report format="frames" todir="<todir specified in the junitreport tag>">/>

</junitreport>

You can specify your own file set and report. Because the task is a direct subclass of **junitreport**, all the attributes and nested elements that **junitreport** has are supported.

We recommend specifying the inherited **toDir** attribute in most cases, although it defaults to the current working directory.

# Conclusion :

Integration of Lisa and Bamboo would help us build projects or mainly run the testcases for different LISA projects whenever there is change to the version control repository Bamboo job can be triggered which automatic builds and you can use Bamboo to trigger Lisa tests by either Ant tasks or Lisa Command Line Utility. This will help us to check of the health of the system under test and achieve a better quality.