# TouchTest™ Jenkins CI for Android Tutorial

SOASTA TouchTest™ Jenkins CI for Android Tutorial

## Table of Contents


**About This Tutorial** ..... **1**

**Jenkins/TouchTest Prerequisites** ..... **2**

Android and Ant ..... 2

Configuring the Android Device ..... 2

Test Composition Prerequisites ..... 3

**CloudTest Continuous Integration Support** ..... **4**

**Installing the SOASTA CloudTest Jenkins/Hudson Plugin** ..... **5**

**Installing the GitHub Plugin** ..... **8**

**Static vs. Dynamic Instrumentation** ..... **10**

Jenkins Workflows for TouchTest ..... 10

Dynamic Instrumentation of an APK file ..... 11

Static Instrumentation of an Android Project ..... 12

**Creating a New Jenkins Job** ..... **13**

**Update the Android Project** ..... **16**

**Using the CloudTest Jenkins Plugin, MakeAppTouchTestable** ..... **18**

Using MATT on an APK (Dynamic) ..... 19

Using MATT on a Project (Static) ..... 21

**Building the APK File** ..... **23**

Build the APK File using Ant (Dynamic) ..... 24

Build the APK File using Ant (Static) ..... 25

**Install the APK File to the Device(s) (All Workflows)** ..... **27**

**Playing the Composition** ..... **28**

**Building the Project with Jenkins** ..... **30**

**Inspecting Test Results in Jenkins** ..... **31**

**Appendix I: Importing Source Controlled SOASTA XML** ..... **I**

Importing CloudTest Objects ..... I

**Appendix II: Jenkins Plugin, Transaction Thresholds** ..... **IV**

## About This Tutorial

This tutorial guides the user through the process of using the Jenkins continuous integration tool combined with the CloudTest Jenkins/Hudson Plugin in tandem with an example Android project and preconfigured CloudTest test compositions.

This tutorial provides guidance for two audiences:

- Users who would like to add Android testing using Ant to a pre-existing Jenkins setup
- Users who are Android Developers starting out with TouchTest who would also like to add TouchTest to a continuous integration setup

**TIP:** If your organization is not already using Jenkins—refer to the documentation on the Use Jenkins page to get started.

In the steps presented in this tutorial a Jenkins job will be defined that can get project source code, make it or its compiled APK TouchTestable, deploy it, and use the mobile app to silently play a test composition in CloudTest and on the specified devices. Finally, CloudTest results are inspected inline in Jenkins, also via the CloudTest Jenkins/Hudson Plugin.

**Note:** A version of the Droidfish chess game, an open-source project available from GitHub, has been customized for use in this tutorial. You can, of course, substitute your own mobile app's source.

## Jenkins/TouchTest Prerequisites

This guide presumes that the Android developer has the following configuration:

### Android and Ant

- This tutorial uses Apache Ant with Android SDK. Learn more about the Android SDK here, and Apache Ant here.
    - The Minimum Android Version supported for use with TouchTest™ is 2.3.3 (Gingerbread).
    - The Minimum Apache Ant version required for using the SOASTA CloudTest Jenkins Plugin is 1.8.0 or later.
- The Jenkins GitHub Plugin is used to retrieve the project used in this tutorial (instructions for installing it are presented below). Refer to the git site here.
- The native app example in this tutorial, Droidfish, uses the Android NDK, although this is not a requirement of TouchTest itself. Unless you have a different Android native mobile app you'd like to use with this tutorial, install the Android NDK toolset into your environment before proceeding. The NDK Library is deployed into the workflow at the appropriate time (either using Ant or the CloudTest Jenkins Plugin, MATT module).

### Configuring the Android Device

Each Android device must have the TouchTest Agent app installed and Connected at runtime (if mobile web testing will occur then the TouchTest Web app is also required).

**TIP:** Refer to the "Installing the TouchTest Agent on a Device (All Users)" section of the TouchTest Android Tutorial to configure your mobile device for use within your CloudTest instance.

Each Android device should have the following Settings:

In the device settings, tap Developer Options and check the USB Debugging box.

In the device settings, tap "Security" and then tap to check the Unknown sources box.

**Note:** TouchTest Agent must be installed on the device. Instructions to install it are included below.

### Test Composition Prerequisites

The test compositions you will use must already exist on the CloudTest instance that you specify and you must use the correct SOASTA Repository path to invoke them. The test composition specifies the device(s) that it will run on.

If you're a new TouchTest user, refer to the following documentation before proceeding with this tutorial.

- Basic TouchTest recording is covered in the TouchTest™ Android Tutorial.

The test clips shown in the result dashboards at the end of this tutorial were created using the GitHub version of Droidfish used in this guide simply by following the steps presented in the following two Advanced Tutorial sections:

  - Create a Simple TouchTest Clip – King Gambit Declined
  - Advanced Clip Editing – Fool's Mate

**Note:** Note that the Droidfish app (rather than Stockfish) and alternate names are used for the test composition in this Android version; Composition for Droidfish1 and 2, respectively. These apps are based on the same underlying chess game engine and the clip creation steps are interchangeable (e.g. between Stockfish and Droidfish).

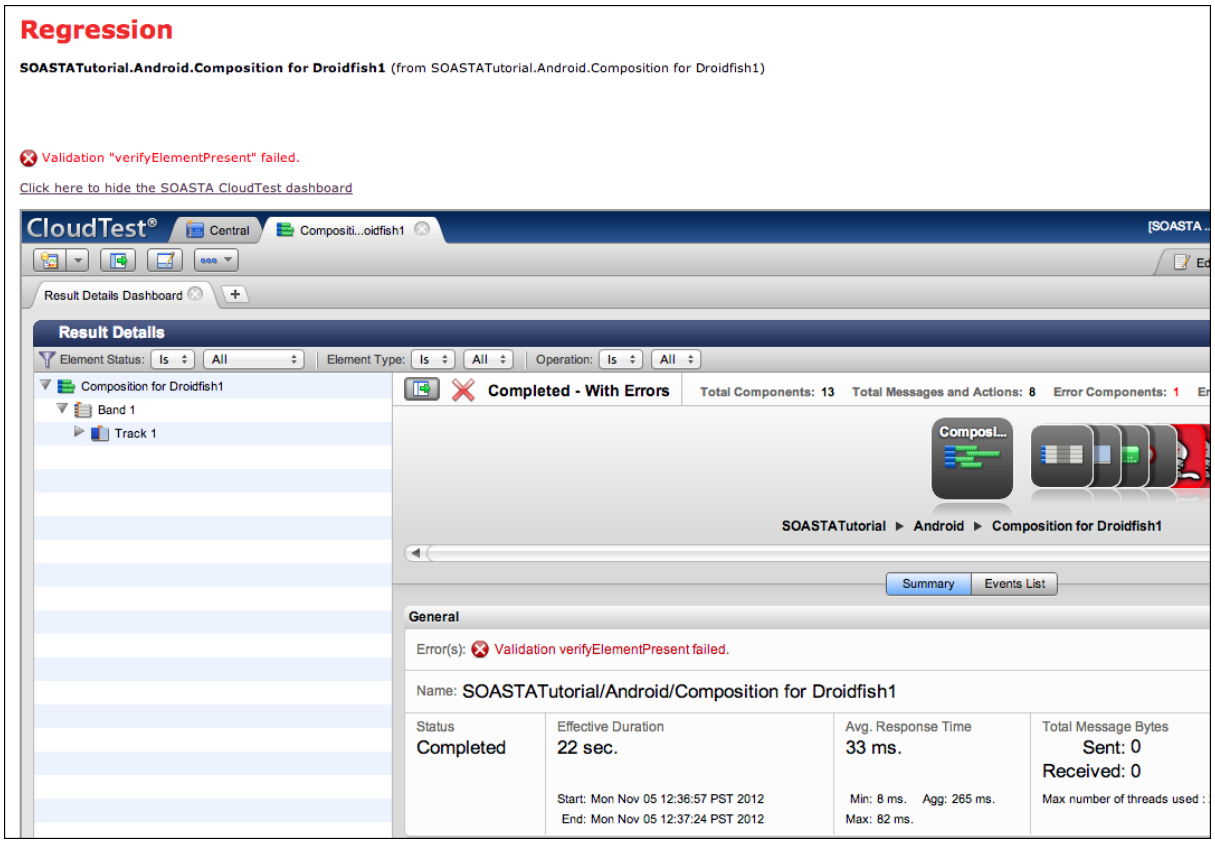
## CloudTest Continuous Integration Support

SOASTA CloudTest includes first-class support for including test output in build reports for Jenkins and Hudson via the SCommand utility. Additionally, the Jenkins/Hudson Plugin provides visual integration with CloudTest dashboards within Jenkins itself.

CloudTest automatically outputs JUnitXML-compatible test output that is the basis for test result display in Jenkins. Since Jenkins provides out-of-the-box JUnit support these result details from a given test composition run in Jenkins will appear on the corresponding Test Results page in Jenkins.

By placing all its XML into a directory that we also provide to Jenkins as a part of defining a given job, we can easily display these JUnit-friendly test results inside Jenkins. Using this configuration, a Jenkins Test Result detail page will display details from CloudTest.

While it is not necessary to install the CloudTest Jenkins/Hudson Plugin to utilize CloudTest's JUnitXML-formatted output in Jenkins in this manner, plugin installation adds the capability to also jump to specific test composition errors in the CloudTest Result Details dashboard from within Jenkins.



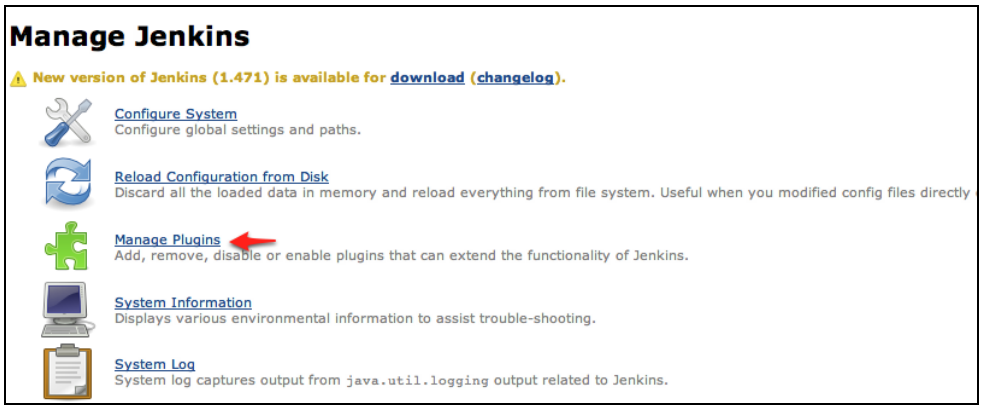

In the screenshot above, the error heading and detail text are an output of SCommand for the test shown. The hypertext link for the CloudTest dashboard is an output of the plugin, which has already been clicked to show the CloudTest dashboard. Once the Jenkins plugin is installed, The Click here... link will appear once the Post-build action box is checked.

Clicking the link, and providing CloudTest credentials, will then display the precise failure inline in Jenkins for the given CloudTest result.

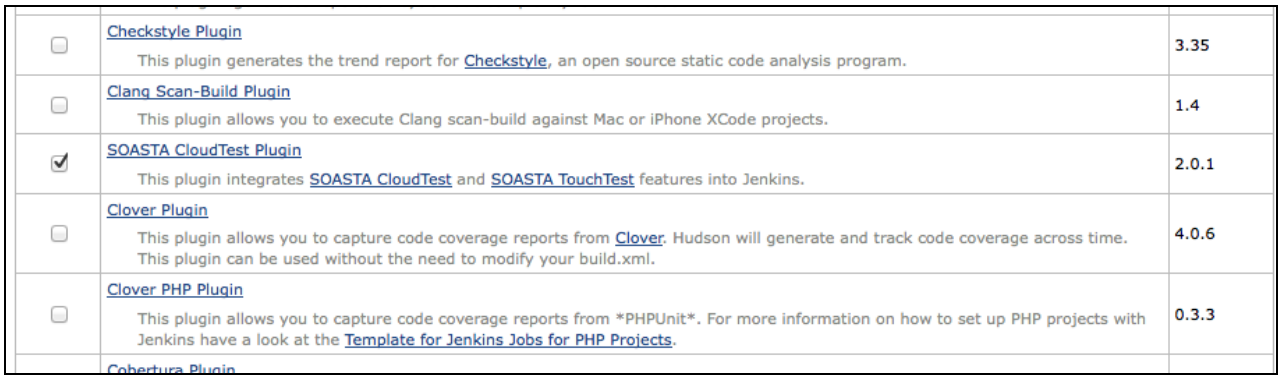
## Installing the SOASTA CloudTest Jenkins/Hudson Plugin

Use the following steps to install the CloudTest Jenkins Plugin (version 2.9 or later are required for dynamic instrumentation as well as to automatically update the CloudTest Jenkins Plugin's MakeAppTouchTestable) from within your Jenkins instance. The MATT module is auto-updated, so once it's installed the most current version is assured.

1. In Jenkins, click Manage Jenkins, and then click Manage Plugins.

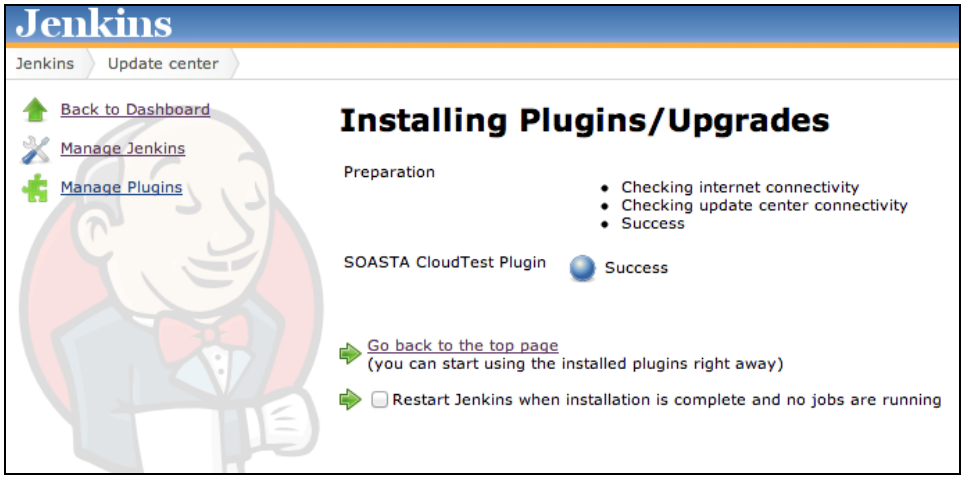



2. Click the Plugin Manager, Advanced tab.
3. Locate the SOASTA CloudTest Plugin in the list (using Cmd+F and "Soasta" is a quick way to locate it).

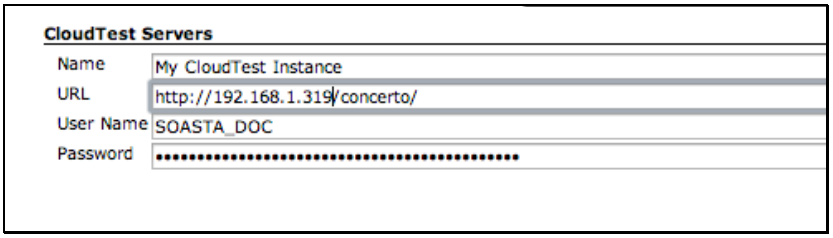



4. Click the Install without Restart button at the bottom of the page.

The Installing Plugins/Upgrades page appears and indicates success once the install completes.



Before using the CloudTest Jenkins Plugin, you will need to provide the CloudTest server URL and user credentials via the Manage Jenkins > Configure System page, CloudTest section. *We recommend creating a dedicated CloudTest account for Jenkins to use.*



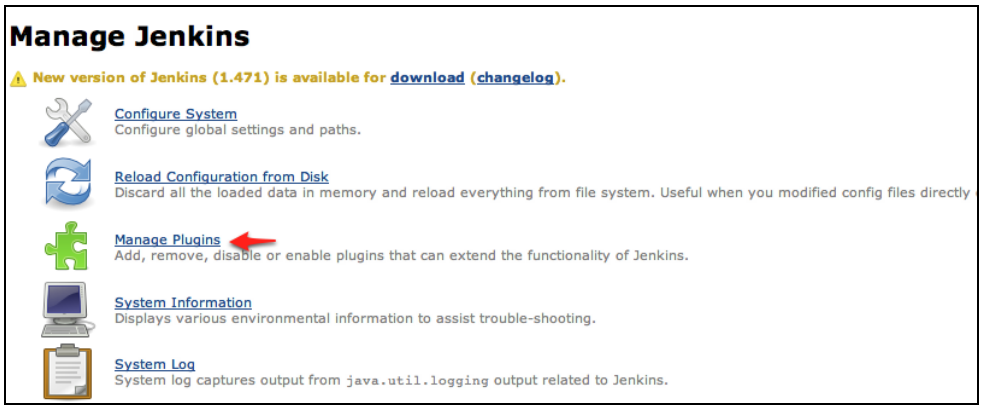
## Installing the GitHub Plugin

Use the following steps to install the GitHub Plugin from within your Jenkins instance. This will allow us to use GitHub as a Source Code Repository to retrieve the example project, Droidfish.

**TIP:** Git is a distributed version control system used for software development. It is not necessary to signup or login to GitHub in order to checkout the code using the following command. In the Source Code Management section, click Git.

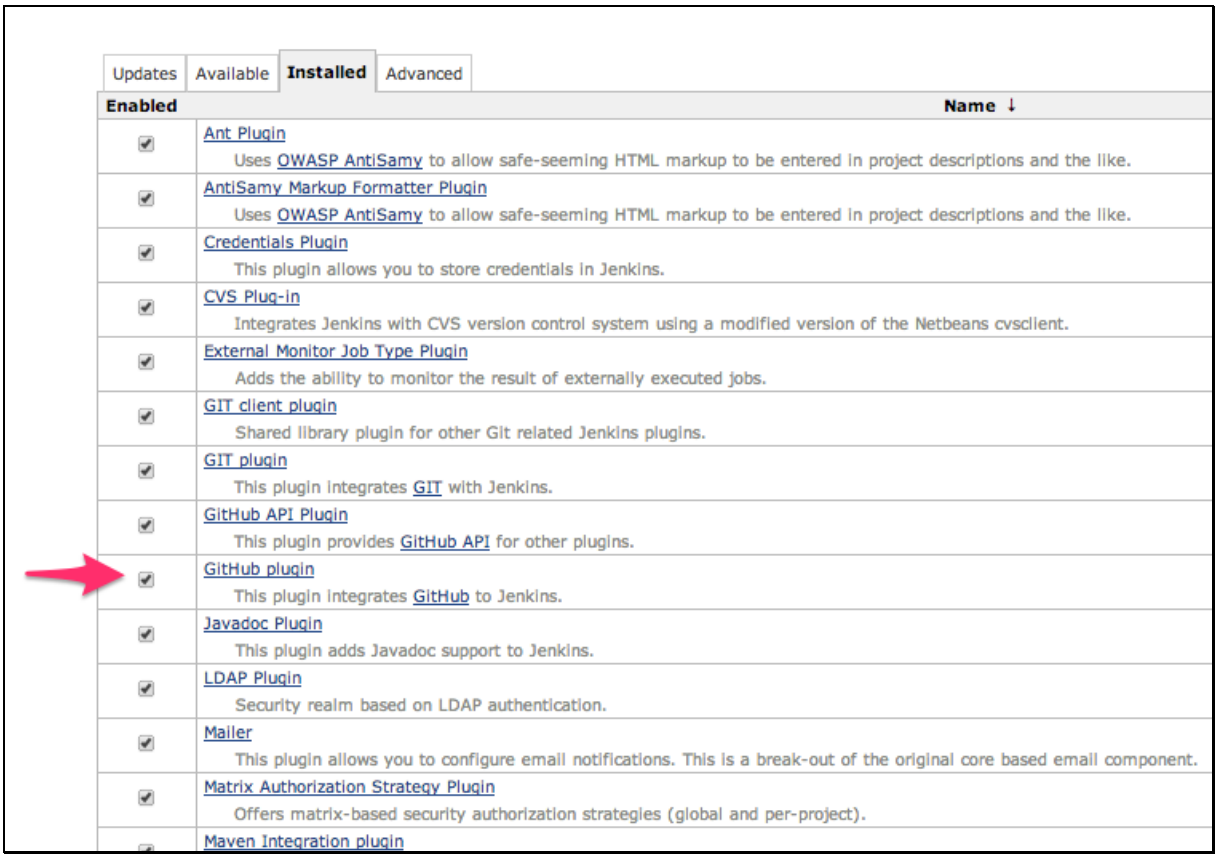
If you are using your own app, you can skip this requirement and then substitute the SCM tool and repository to use in the place of Git.

1. In Jenkins, click Manage Jenkins, and then click Manage Plugins.




2. Click the Plugin Manager, Advanced tab
3. Locate the GitHub Plugin and check it as well. Installing the GitHub Plugin will also install the GitHub Plugin.

You can verify plugin installation on the Manage Plugins, Installed tab:




**Note:** We will specify a Git repository to clone in a later step in the Jenkins job.

## Static vs. Dynamic Instrumentation

The CloudTest Jenkins Plugin's MATT module supports two instrumentation methods: static and dynamic. The MATT module plays a role in making either the Android project (static instrumentation) or its compiled APK (dynamic instrumentation) TouchTestable.

- Dynamic instrumentation occurs when MATT instruments a compiled file (i.e. an APK file).

  This method requires that you compile your Android project first to create an APK, after which it can be instrumented using SOASTA 51.07 or later (TouchTest 7040.58). Dynamic instrumentation is available for all supported Android versions.

- Static instrumentation occurs when MATT instruments an Android project. This method requires that you apply MATT to the project prior to building the APK. Static instrumentation is available in all TouchTest releases and for all supported Android versions.

1. Determine whether to instrument the mobile app using the MATT input type project or APK. The subsequent steps will differ since MATT is applied prior to compile when using static instrumentation then it is when using dynamic instrumentation.
   - To use dynamic instrumentation, follow the steps in Making the Droidfish APK TouchTestable.
   - To use static instrumentation, follow the steps in Making the Droidfish Project TouchTestable.

### Jenkins Workflows for TouchTest

The CloudTest Jenkins Plugin is used to make the mobile app TouchTestable—either by applying it to the project or to the APK file itself (as discussed below). Ant is used to build the APK and the command used depends whether MATT has already been

applied. After which, adb is used in an Execute Shell step, followed by a final pre-build step that uses the CloudTest Jenkins Plugin, Play Composition(s) command.

Because there are different possible Jenkins job workflows, the *possible* steps are presented *a la carte.* For each Jenkins job there will be (minimally):

- A First step; used to retrieve the source project (all workflows)
- In between the first and last steps, each Jenkins job will have:
  - A CloudTest Plugin, MakeAppTouchTestable step;
  - with one (or more) Build step using Ant (in Execute Shell)
  - and one (or more) Install steps using adb (in Execute Shell)
- A Last step; to Play the Composition

**Note:** Signing the app is done using the CloudTest Jenkins Plugin, MATT, Advanced Options to enter the optional parameters (refer to the relevant sections).

Note the following terminology:

- *Static instrumentation* applies MATT to the Android project file
- Dynamic instrumentation applies MATT to the APK file

**Dynamic Instrumentation of an APK file**

In this workflow, you'll apply MATT to the compiled APK file using the appbundle parameter. If you're also deploying to physical devices, you'll need to mix and match steps to do both, keeping in mind to build the APK first.

- Get the source code (in this scenario, using git)
- *Build the APK* with an Execute Shell step using Ant
- *Apply MATT* to that compiled APK using MATT's APK input type
- *Install the APK*
- *Run the composition(s)*

- *Post results (optional)*

In the following sections, choose only those steps that match your workflow.

**Static Instrumentation of an Android Project**

In this workflow, you'll apply MATT to the Android project itself, using the project input type. This is done before the APK build step. After which, add the step(s) necessary to build and install to your simulators and devices.

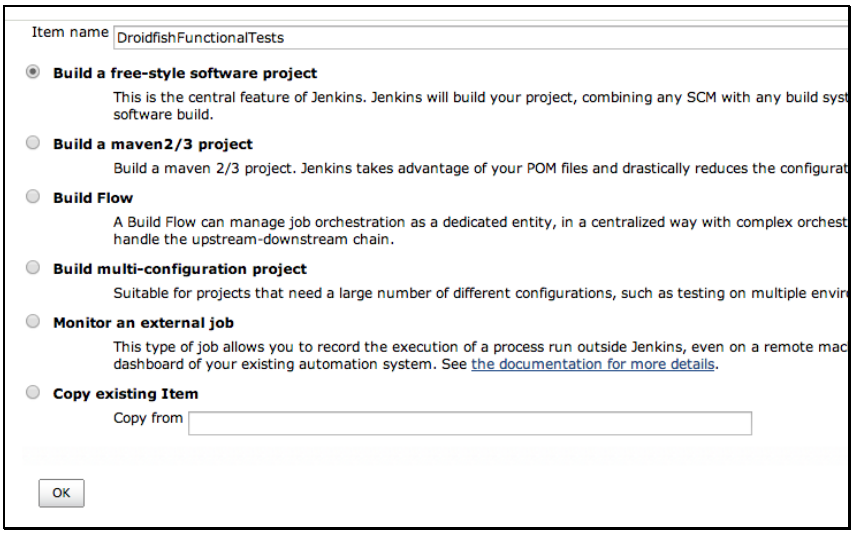
- Get the source code (in this scenario, using git)
- *Apply MATT* using the project input type (static only)
- *Build the APK* with an Execute Shell step using Ant
- Install the APK
- Run the composition(s)
- Post results (optional)

## Creating a New Jenkins Job

The Jenkins job created below will have steps that get the source code, ensures a project update using Execute Shell, uses the CloudTest Jenkins Plugin, MakeAppTouchTestable module on that project or APK, builds an APK (either before or after it is made TouchTestable) and installs the APK to a device, after which the CloudTest Jenkins Plugin's Play Composition(s) command is used to silently play a list of compositions. Finally, a Post-build action is used to publish the JUnit test result report.

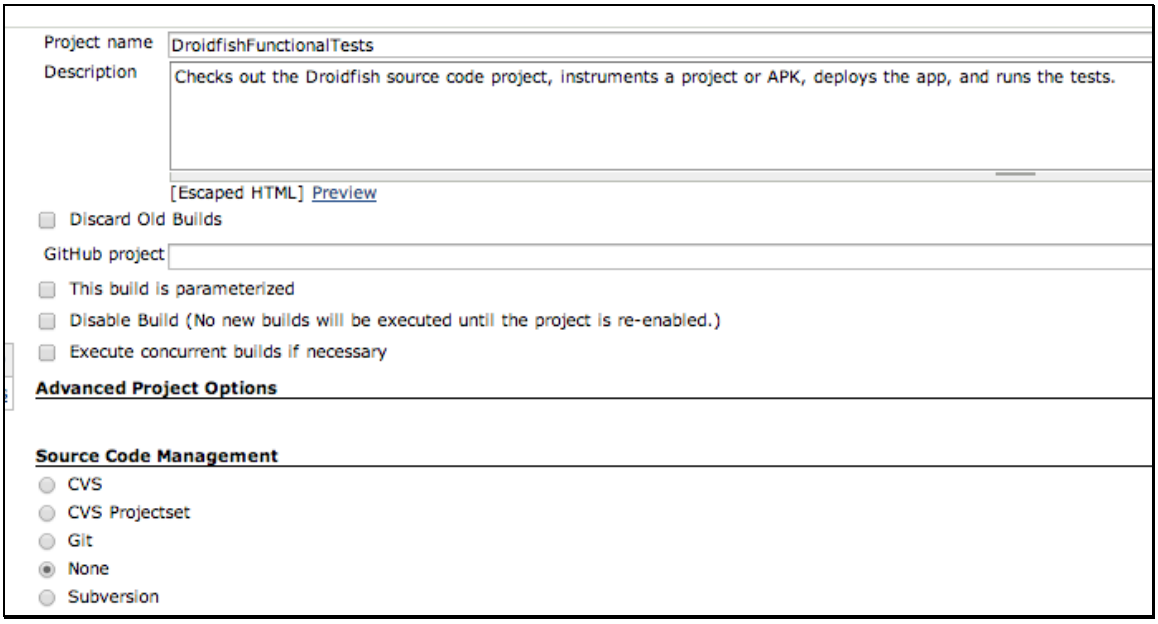
1. In the top-level Jenkins dashboard, click New Job.

The Job Details page appears.




1. Enter a job name that has no spaces (for example, *DroidfishFunctionalTests*) and check the first option, "Build a free-style software project".

   The Job Details page appears.




2. Enter a description for this job:

- For example, "Checks out the Droidfish source code project, instruments a project or APK, deploys the app, and runs the tests."

Leave the Job open and continue with the following section.

## Get Droidfish using the GitHub Plugin (All Workflows)

Next, we will add a step that will get the Droidfish Android project that will be used in the remainder of this tutorial.

The following instructions use the Jenkins GitHub Plugin (installed above). If you are using your own Source Code Management system simply select its type and enter its repository URL, as you would normally do.

With the GitHub Plugin installed in our Jenkins instance, we will add the Source Code Management step as we would with any SCM tool.

1. In the Source Code Management section, check the Git radio button.


**Note:** You can specify your own SCM tool and Repository URL here.

2. Enter the repository URL in the entry field:

   https://github.com/elitecoder/droidfishchess_android

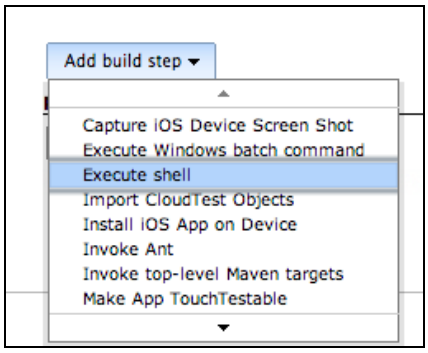

3. Click Apply.

Leave the Job open and continue with the following section or click Apply and then Save to return later.

## Update the Android Project

Now that we've added Git as our source, let's update the Android project to ensure that we will be able to build the APK file later using ant.

1. Click Add a Build Step, and select Execute Shell from the drop-down list. Note that after the step is created you can drag it to the correct place in the workflow of your Jenkins job.

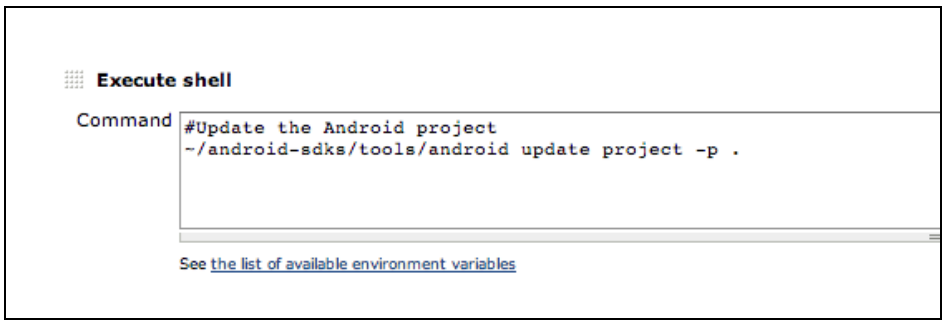



2. Next, enter the following lines in the Execute Shell field:

   #Update the Android project

   ~/android-sdks/tools/android update project -p .

The APK file is a requirement for deployment to all Android simulators and devices—in addition to being required before using the CloudTest Jenkins Plugin's MATT, APK input type.




3. Click Apply.

Leave the Job open and continue with the following section or click Apply and then Save to return later.

## Using the CloudTest Jenkins Plugin, MakeAppTouchTestable

As noted in the prerequisites above, the CloudTest Jenkins Plugin, MakeAppTouchTestable module is used to automate portions of making the app TouchTestable.

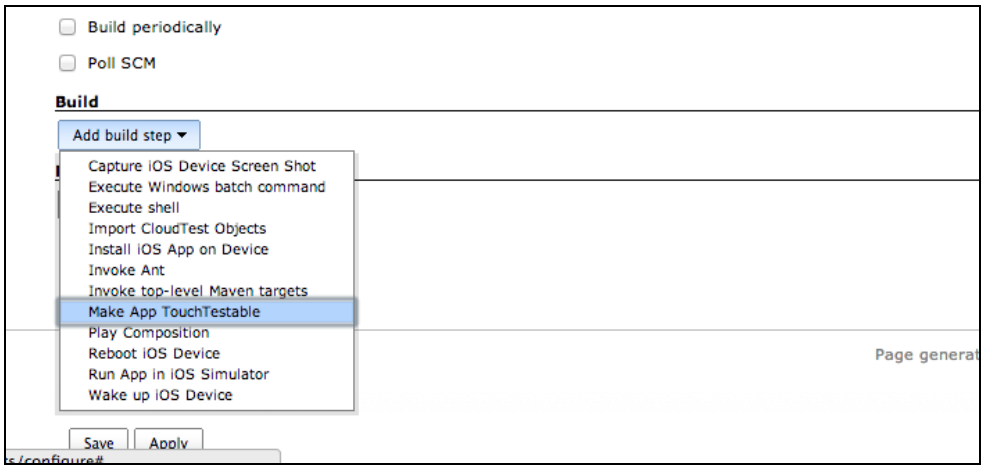
**Note:** The CloudTest user specified to run the CloudTest Jenkins Plugin, MakeAppTouchTestable module must be a user with Mobile Device Administrator rights. CloudTest Lite users have admin rights for the given device on their own instance.

## Using MATT on an APK (Dynamic)

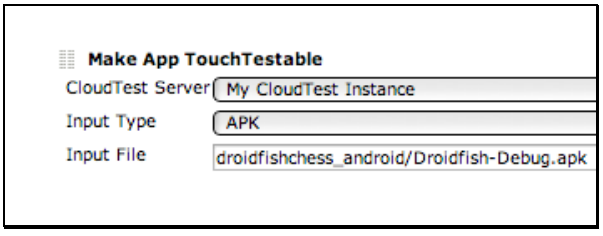
Use the following steps to dynamically instrument an APK file at the proper point in your workflow.

- If you are using dynamic instrumentation, you must first do the APK step prior to this step. But, if you'd rather follow along sequentially, you can add the MATT step now so long as you place an APK build step prior to it before building the Jenkins job.
- If you are using static instrumentation, you must apply MATT to the project prior to this step.

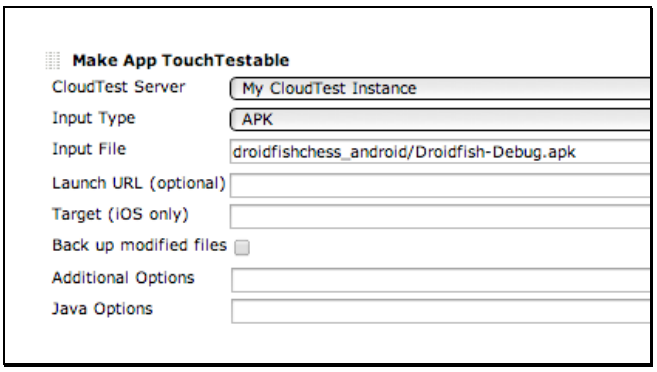
1. Add a MakeAppTouchTestable step to the job.




2. Select the Input Type, APK (e.g. the MATT equivalent is the APK parameter).




3. Enter the APK name as the Input File (from the workspace root).

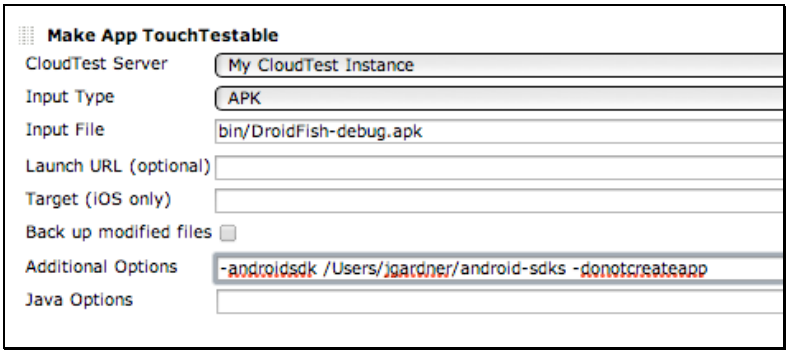



4. Click Advanced to display additional MATT configuration fields. Enter the `androidsdk` flag here and the path to the Android SDK in use. This is required when instrumenting an APK file. For example, -androidsdk ~/Development/android-sdk-macosx

5. Specify other MATT flags as required.

   For example, use either `overwriteapp` or `donotcreateapp` to control mobile app object creation. In some cases, this step can prevent Jenkins from marking the MATT step with `FAILURE` status (even though the Mobile App object on the CloudTest instance is created as expected).

   Refer to Additional Options below for other essential flags to use here, including those related to signing the APK file.




- **Launch URL** – Same as MATT launchurl. For example: my-app://launch
- **Backup modified files** – Check this to keep backups in the project.

- **Additional options** - Enter any additional MATT command line parameters.

  Most notably, you can use MATT to add keystore, keypass, and storepass arguments to sign the dynamically instrumented APK file.

  Use the following MATT optional APK parameters

  - -keystore <keystorepath> - Path of the keystore to be used to sign the APK file..
  - -storepass <keystorepassword> - Password of the keystore to be used to sign the APK file.
  - -keypass <privatekeypassword> - Password of the private key (if different than the keystore password) to be used to sign the APK file.

  For more about using additional MATT parameters, use:

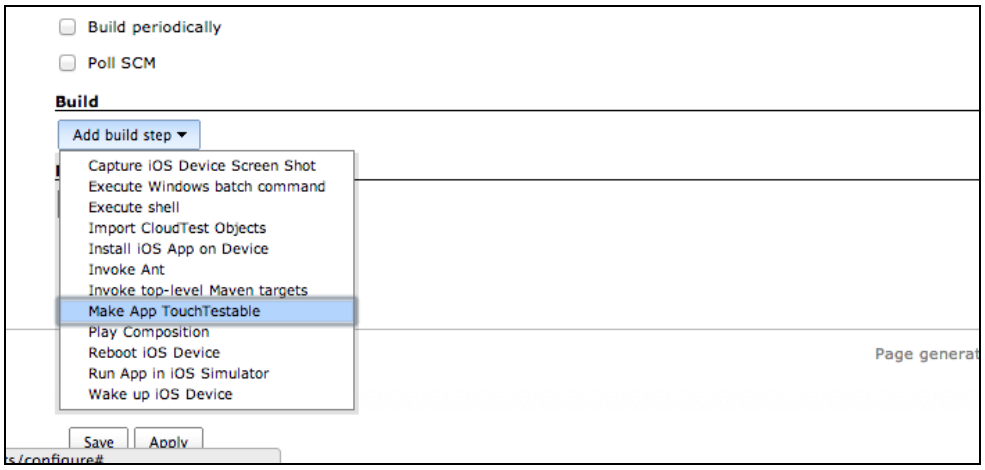
  sh MakeAppTouchTestable/bin/MakeAppTouchTestable - help

6. Save the Jenkins job.

### Using MATT on a Project (Static)

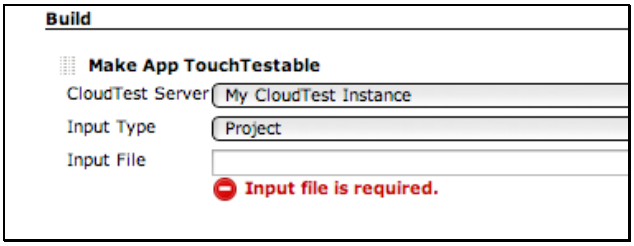
Use the following steps if you'll be instrumenting the project using the static method via the Input Type, project. This step will always proceed the build APK step while using static instrumentation.

1. Click Add a Build Step, and select Make App TouchTestable from the drop-down list.



The Make App TouchTestable form appears.

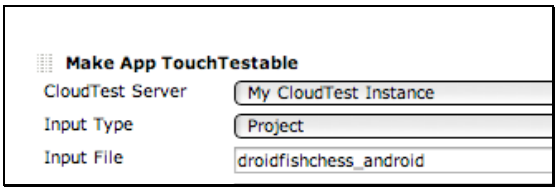
**TIP:** Click the Help icons for any row to get tip text.



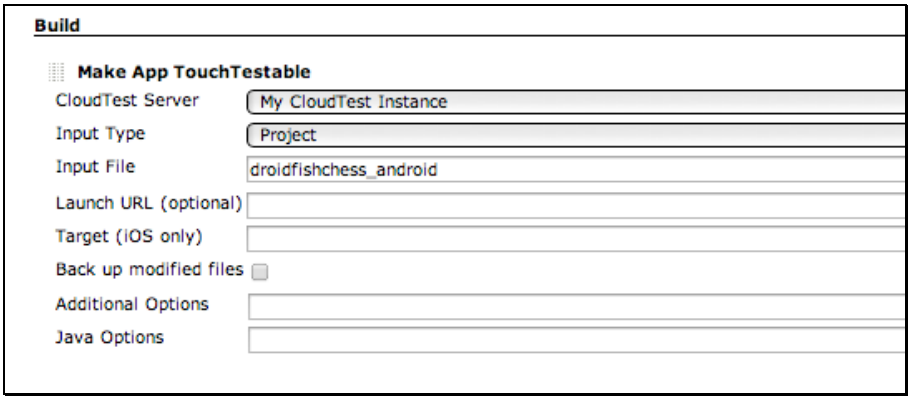

2. Select the CloudTest Server from among those configured.

**Note:** Indicating the CloudTest server was done as part of the CloudTest Jenkins Plugin, Configure System step. If not entries appear here return to Manage Jenkins > Configure System, and fill in the information in the CloudTest Servers section. Be sure to save these changes.

3. Specify the current project folder to use. "droidfishchess_android" to indicate the project folder in the Jenkins workspace.




4. Optionally, specify additional parameters by first clicking the Advanced button (page right).




5. Select the CloudTest Server from among those configured.

**TIP:** Indicating the CloudTest server was done as part of the CloudTest Jenkins Plugin, Configure System step. If not entries appear here return to Manage Jenkins > Configure System, and fill in the information in the CloudTest Servers section. Be sure to save these changes.

6. Optionally, click Advanced to display additional MATT configuration fields.

- **Launch URL** – Same as MATT launchurl. For example: my-app://launch
- **Back up modified files** – Check this to keep backups in the project folder (where build.xml resides).
- **Additional options** - Enter any additional MATT command line parameters.

  Most notably, you can use MATT to add keystore, keypass, and storepass arguments to sign the dynamically instrumented APK file.

  Use the following MATT optional APK parameters

  - -keystore <keystorepath> - Path of the keystore to be used to sign the APK file..
  - -storepass <keystorepassword> - Password of the keystore to be used to sign the APK file.
  - -keypass <privatekeypassword> - Password of the private key (if different than the keystore password) to be used to sign the APK file.

  For more about using additional MATT parameters, use:

  sh MakeAppTouchTestable/bin/MakeAppTouchTestable - help

7. Click Apply and then Save.

## Building the APK File

The APK file is a requirement for deployment to all Android simulators and devices and will be placed in the Jenkins workspace at "bin/Droidfish-Debug.apk."

Add this build step at the point in your workflow required by the instrumentation type that you chose—dynamic or static.

**Note:** The Ant commands used to build the APK differ slightly and are not interchangeable.

- If you are using dynamic instrumentation, build the APK prior to using MATT to instrument it.
- If you are using static instrumentation, apply MATT to the project prior to building the APK. While using the static build step, you will also use the -Dtouchtest.enabled=true debug then you DONT use this parameter.

**Note:** As noted in the prerequisites above, the Minimum Apache Ant version required for using the SOASTA CloudTest Jenkins Plugin is 1.8.0 or later.

### Build the APK File using Ant (Dynamic)

Use these instructions to build an APK file that has not yet been made TouchTestable using the Input Type, APK. This step will always be followed by a step that uses the CloudTest Jenkins Plugin, MATT module, APK input type. Refer to "Using MATT on an APK (Dynamic Instrumentation for Device)" above for instructions.

In this example, ant is presumed to be in the path (supply the additional path if not).

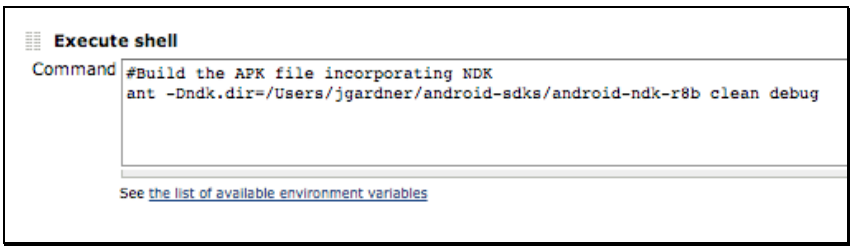
**Note:** As noted in the prerequisites above, the Minimum Apache Ant version required for using the SOASTA CloudTest Jenkins Plugin is 1.8.0 or later.

The APK file is a requirement for deployment to all Android simulators and devices and will be placed in the Jenkins workspace /bin as Droidfish-Debug.apk

1. Next, enter the following lines in the end of the Execute Shell field:

```
#Build the APK file using Ant

ant -Dndk.dir=/Users/username/android-sdks/android-ndk-r8b clean debug
```




where:

- -Dndk.dir=/Users/username/android-sdks/android-ndk-r8b - Adds the NDK Library required by Droidfish. This is not a TouchTest requirement.
- For this example, we must build with the debug parameter because this project includes no key store. We are building a debug version of the Droidfish app here that doesn't require APK signing.

  Refer to Signing Your Applications on the Android Developer site if you will not be using MATT to sign your mobile app. Otherwise, use the available MATT flags to sign the app.

### Build the APK File using Ant (Static)

Use this build step version to build an APK file that has not already made TouchTestable using the CloudTest Jenkins Plugin, MATT module, Input Type, project.

1. Next, enter the following lines in the end of the Execute Shell field:

   #Build the static APK file using Ant

   ant -Dtouchtest.enabled=true debug -Dndk.dir=/Users/username/android-sdks/android-ndk-r8b clean debug


where:

- -Dndk.dir=/Users/username/android-sdks/android-ndk-r8b - Adds the NDK Library required by Droidfish. This is not a TouchTest requirement.
- -Dtouchtest.enabled=true - This flag tells the build process to perform TouchTest enabling steps as part of the build.
- For this example, we must build with the debug parameter because this project includes no key store. We are building a debug version of the Droidfish app here that doesn't require APK signing.

  Refer to Signing Your Applications on the Android Developer site for more about signing applications. In most cases, you'll use the MATT module's Additional Options entry field to enter MATT flags, including signing.

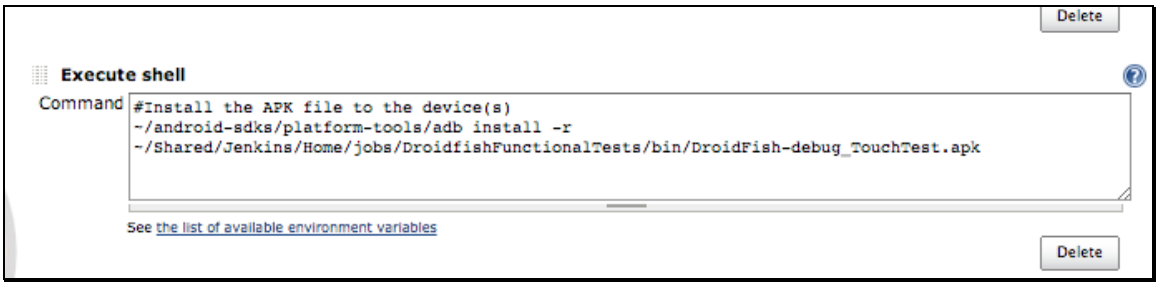
## Install the APK File to the Device(s) (All Workflows)

Now that the APK file exists we can easily deploy it to all of the tethered devices (or running emulators), without specifying the device ID by using the adb command, using adb. Use the Jenkins $WORKSPACE variable to indicate the path, if desired.

1. Create an Execute Shell step in the Jenkins job.
2. Enter the following adb install using your own file paths:

   #Install the APK file to the device(s)

   ~/android-sdks/platform-tools/adb install -r
   ~/Shared/Jenkins/Home/jobs/DroidfishFunctionalTests/bin/DroidFish-debug_TouchTest.apk



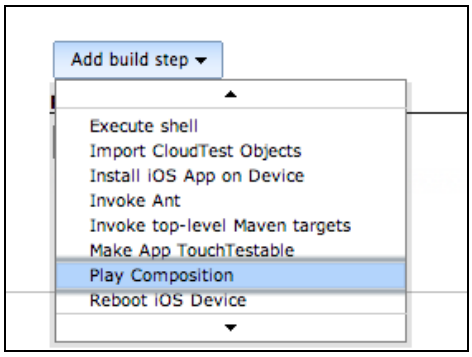

where:

- -r forces an overwrite if the mobile app already exists on a device

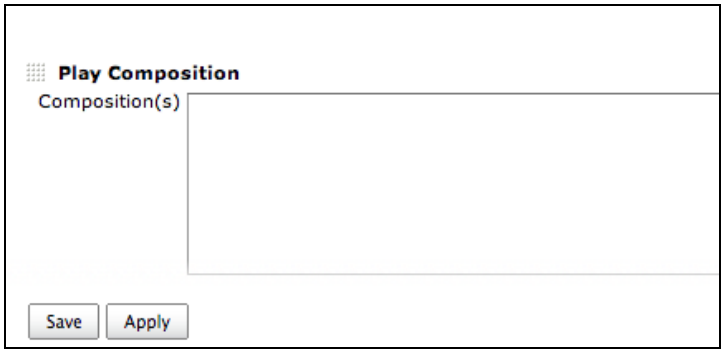
## Playing the Composition

Finally, we will add a build step using the Play Composition command.

1. Add a build step and select Play Composition from the drop down.

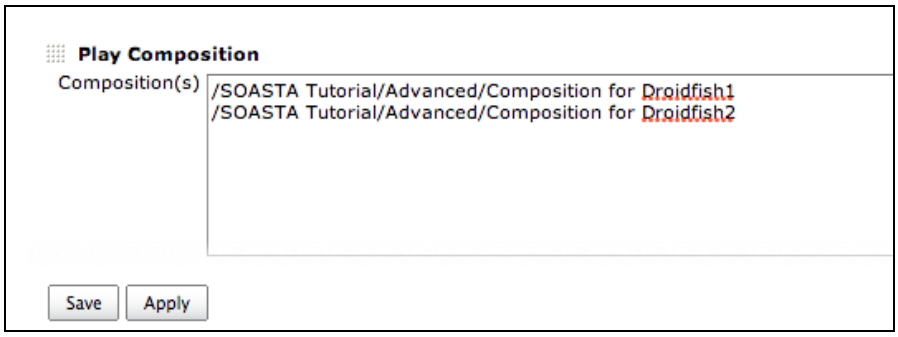



**TIP:** If you are adding more than one composition, click the down arrow in the Play Composition field to expand the entry field before entering the first composition path.




2. Enter each composition to play using its full SOASTA Repository path (shown below). For example,

```
/SOASTA Tutorial/Advanced Composition/Composition for Droidfish1
/SOASTA Tutorial/Advanced Composition/Composition for Droidfish2
```



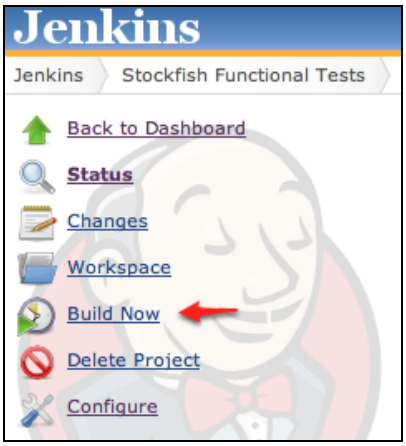

**Note:** Ensure that you have dragged all the steps into the right order before building. The order should be the same as in this tutorial, but may vary slightly depending on whether you're using devices, simulators, or both.

3. Click Save to complete the Jenkins job.

## Building the Project with Jenkins

After you click the Save button, you will be taken to the project page for the job you just created.

1. To build the project, click the "Build Now" link.




The build will start. After a short delay, you should see a progress bar appear on the left side of the page. Click this progress bar to watch the build process "live" in the Console view.

You should see the following happen:

a. Jenkins checks out the source code from Git, and runs the CloudTest Jenkins Plugin, MakeAppTouchTestable module.

b. Jenkins performs whatever Execute Shell steps you've placed to build the APK file.

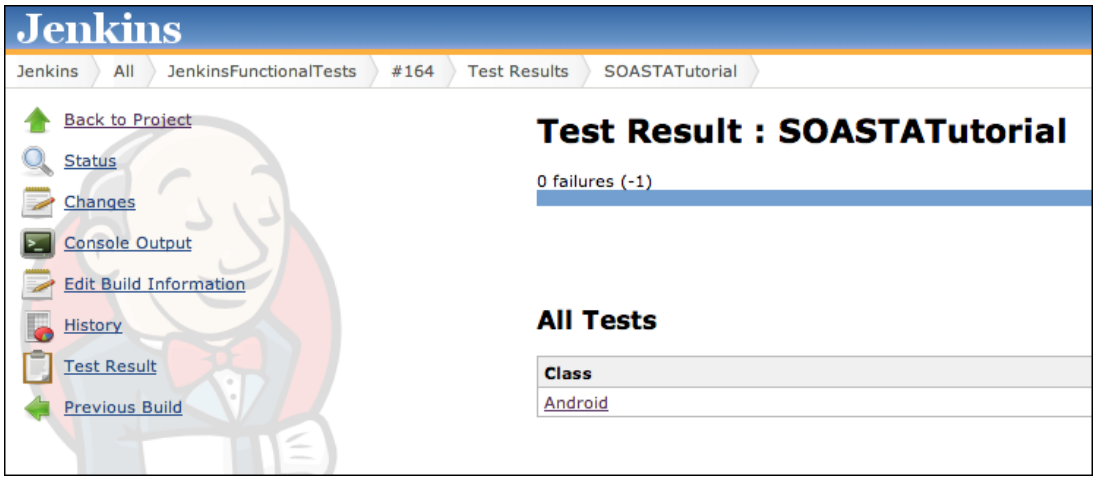
c. Jenkins plays the CloudTest compositions using the CloudTest Jenkins Plugin, Play Composition(s) command. This module utilizes the CloudTest Command Line Utility (SCommand), which is automated by the plugin.

d. On the tethered device, you should see the Droidfish app launch and run through the test steps. When the test finishes, Droidfish will exit, and the SOASTA TouchTest Agent page will re-open.

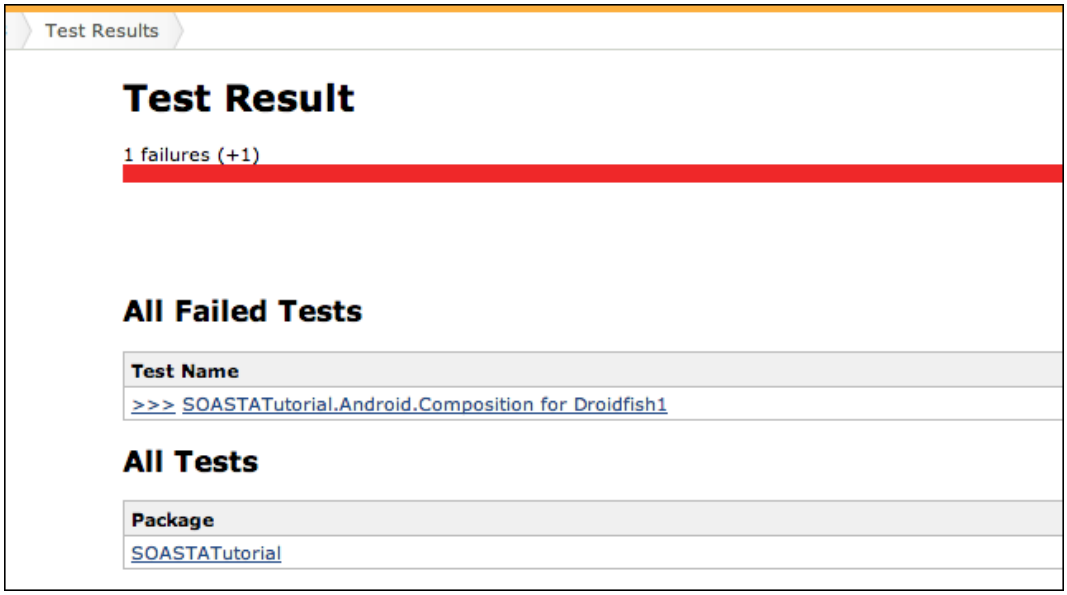
## Inspecting Test Results in Jenkins

For a successful test with no failures, the Test Result page merely lists the All Tests section with the given package (i.e. in this case the package equates to a CloudTest repository folder).

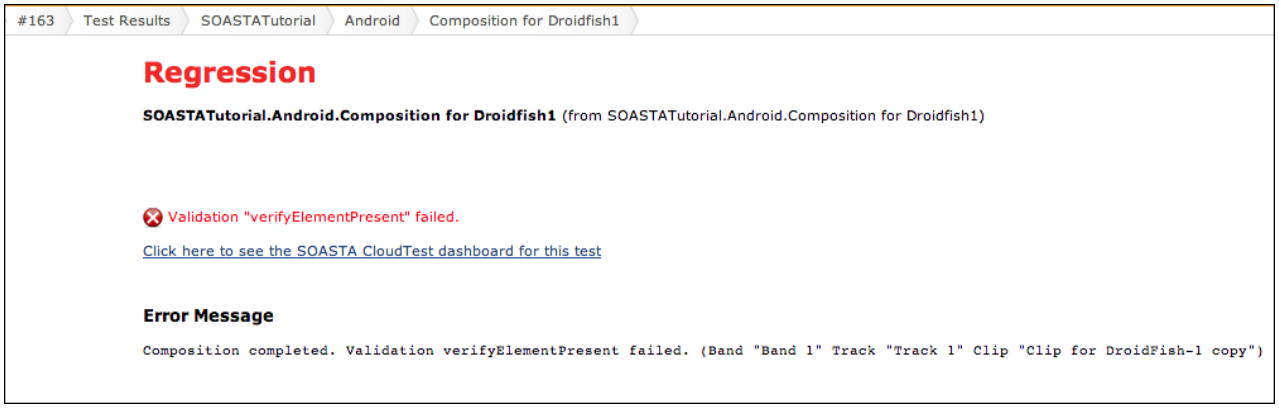

- Clicking the Package link opens the subsequent CloudTest folder.



Things get more interesting when an error in the test occurs. The subsequent SCommand output is displayed (in text) on the Jenkins Test Result page (as discussed above).



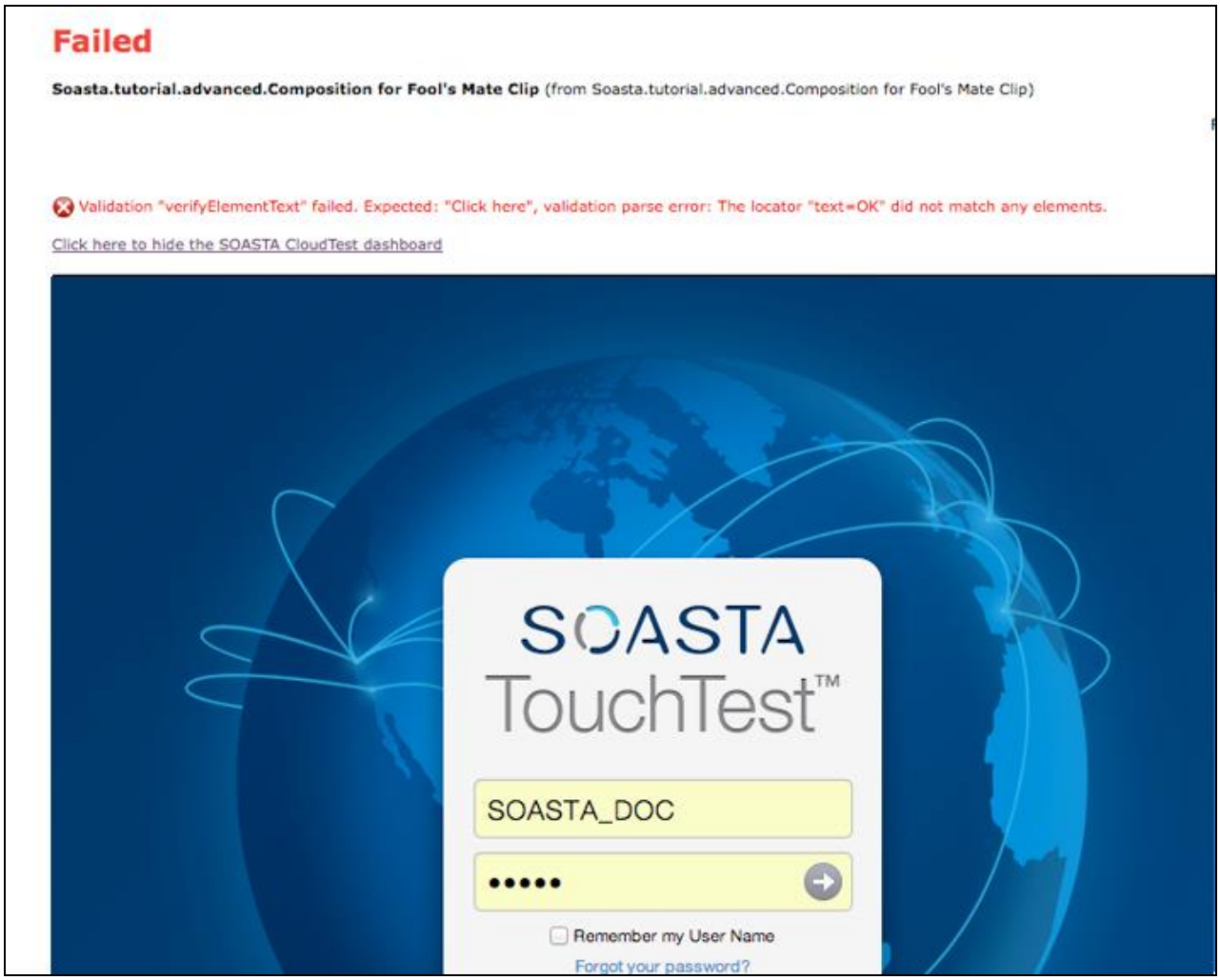

In this case, the All Failed Tests section is added with the name of the test listed with a link to more of the SCommand details. Clicking the link under the Test Name section where the composition is named displays an Error detail page (for the given error).



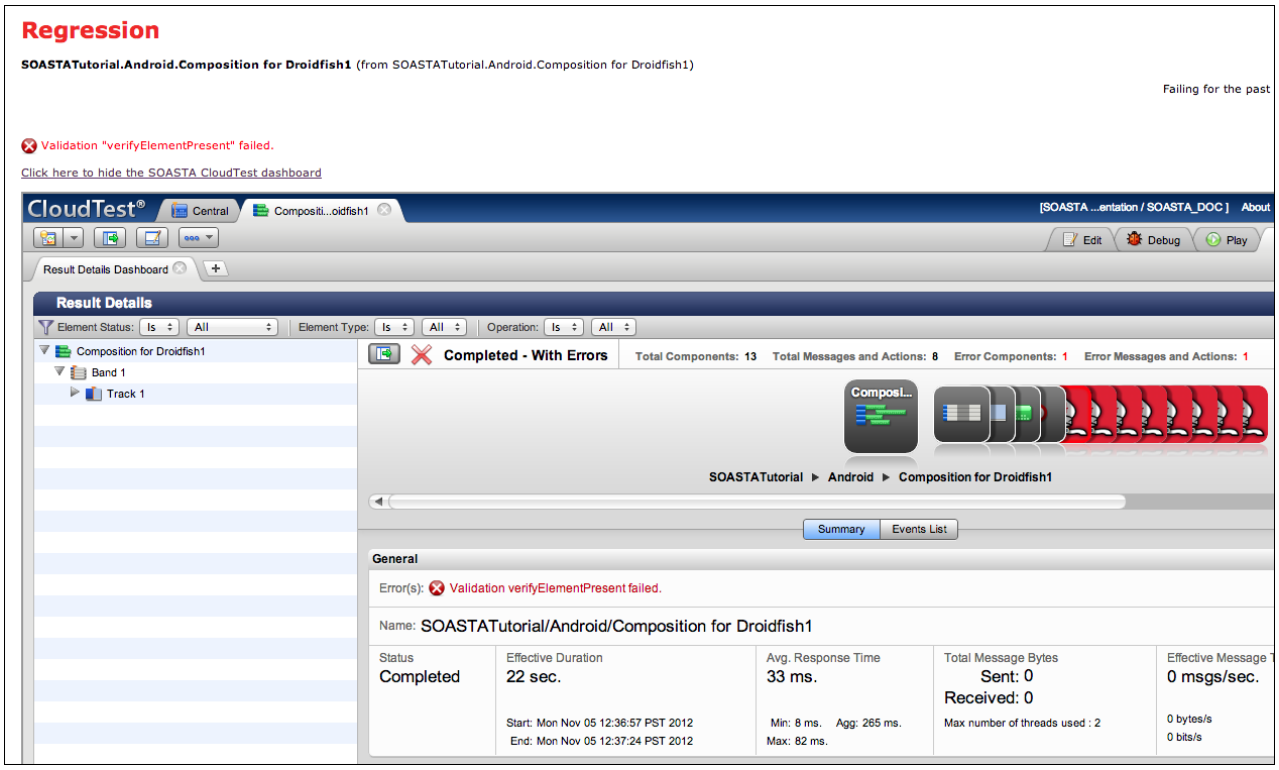

In the error above a validation in the Composition for Droidfish1 has failed.

1. To view this error in the CloudTest, Result Details dashboard, click the plugin link provided (i.e. "Click here to see the SOASTA CloudTest dashboard for this test."

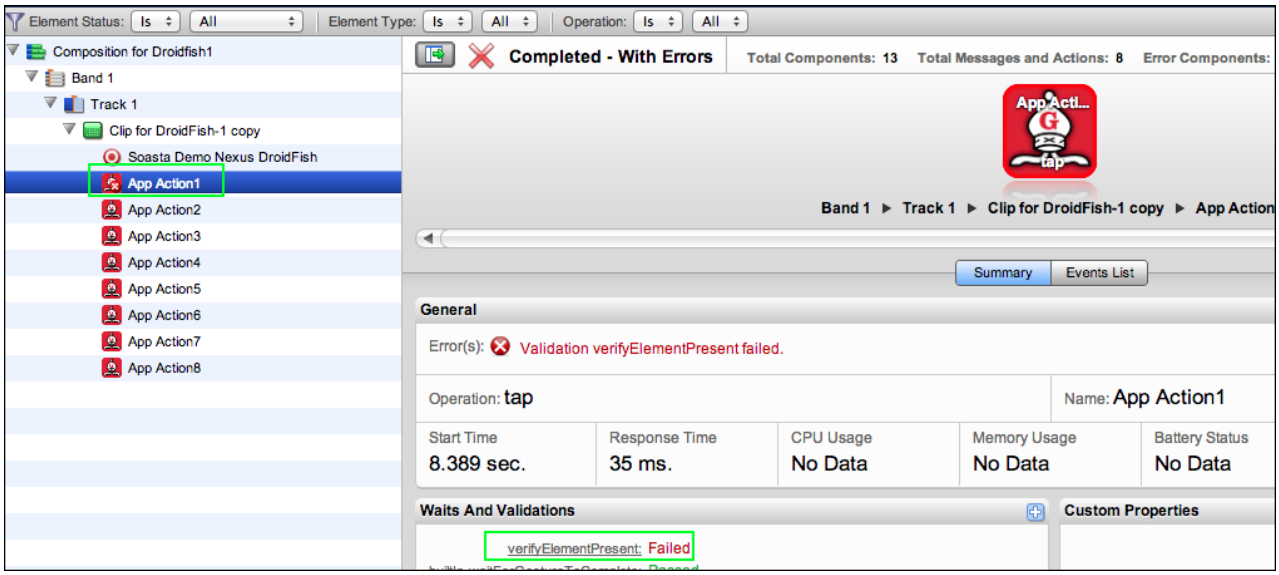
After the plugin link is clicked, enter CloudTest credentials whenever required.




- After credentials are entered, the dashboard tab opens, displays the test result, and jumps to the relevant error.




From here, the Result Details dashboard can be navigated as within any CloudTest dashboard. Refer to [Result Details Dashboard] for a quick review of Result Details features.



## Appendix I: Importing Source Controlled SOASTA XML

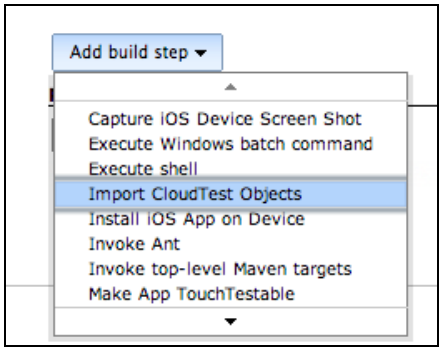
The SOASTA CloudTest Jenkins Plugin includes the ability to import SOASTA objects such as test clips from exported XML. This technique is useful in test environments where more than one tester is working as part of a team.

For example, a team might have access to the following:

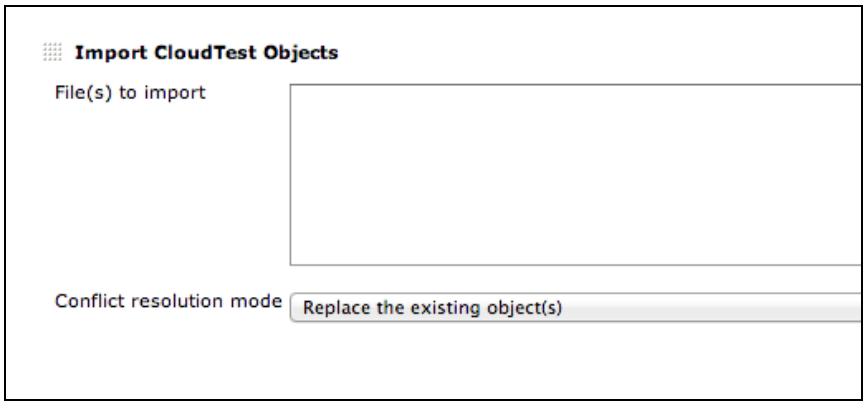
- 1 CloudTest Pro environment being used for CI, with Jenkins pointed at it.
- 1 version control system.
- 3 CloudTest Lite users, working on their own tests, do the following:
    - The CloudTest Lite users export their compositions and clips using the non-zipped XML, and check them into SVN, on a regular basis.
    - The Jenkins job checks them out from SVN, imports the XML into the CloudTest Pro environment, and then runs the compositions on that same environment. The exported objects can include dependent objects but the export should be in XML format and not zipped. Refer to Export and Import SOASTA Objects for steps.

### Importing CloudTest Objects

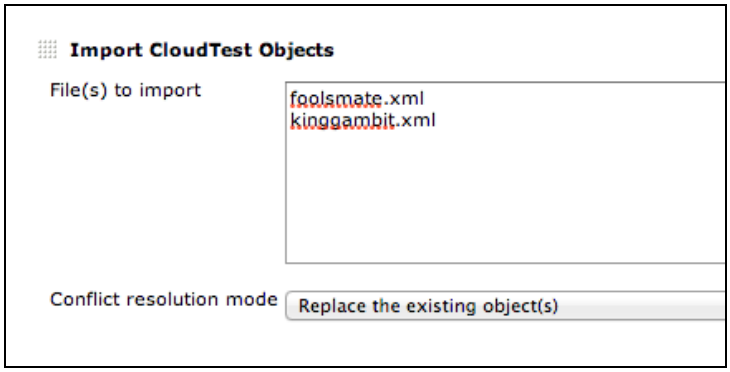
1. Export the object(s) to import and check them into the repository that also has the Xcode project.
2. Add a build step using Import CloudTest Objects. Note that after the step is created you can drag it to the correct place in the workflow of your Jenkins job.




The Import CloudTest Objects form appears. If you will be exporting more than one XML file click the down arrow to expand the form before proceeding.




1. Specify the file name of each XML file to import (one per line).




**TIP:** You can also use wildcards here. For example, if the CloudTest objects were checked into a "testobjects" folder, then entering `testobjects/*.xml` here would import them all.

2. For this scenario, accept the default conflict resolution mode, which is "Replace the existing object(s)."

3. Drag the build step into the correct position. In general, before any Play Composition steps.

4. Click Apply on the Job page.

## Appendix II: Jenkins Plugin, Transaction Thresholds

The Transaction Threshold command—which was made available in the SOASTA CloudTest Plugin, version 2.17—provides users with the ability to automate the validation of Transactions in a composition using CloudTest metrics, such as Average Response Time or Errors per Transaction.

For more details, please visit Setting Transaction Thresholds in Jenkins.

SOASTA, Inc.
444 Castro St.
Mountain View, CA 94041
866.344.8766
http://www.soasta.com