Automated testing and continuous integration
Build and test your software at the same time

Table of contents

Introduction 2
Who should read this document? 2
  Build managers 2
  Developers and testers 2
  Managers 2
Testing in CI 2
Types of advanced CI tests 3
  Deployment tests 3
  Integration/System tests 3
  Smoke tests/Acceptance tests 3
  Functional tests 3
  Load/Stress tests 3
Triggered builds 4
Nightly builds 4
Weekly builds 4
Reporting 5
Additional considerations for scheduled builds 5
Continuous integration and testing with HP 5
What's next? 7
For more information 7
Introduction

Many development teams are turning to continuous integration (CI) as a means to improving the quality of their software, and reducing the amount of time it takes to deliver it. CI provides feedback about the quality of a build as soon as possible. It reduces the risk associated with integrating code back into the source code repository by encouraging developers to commit their code frequently, by building the code as soon as it’s checked in, and running unit tests on the resulting modules. The developer receives immediate feedback about whether the change prevents the code from compiling or whether it introduced unintended consequences in the modified module, or related modules.

CI systems can be further configured to deploy the newly built modules to a production, or production-like environment, and then perform further testing. This brings the following additional benefits:

- Know immediately whether everything necessary to deploy the modules has been checked in
- Data-driven automated testing can rapidly test a number of real-life scenarios
- Quickly discover if the change created any adverse effects in the production system

This paper will explore how developers can take their CI to the next level by introducing automated testing to their processes.

Who should read this document?

The intended audience is anyone who will benefit from introducing testing into their CI systems. Here are some examples:

Build managers

Build managers will learn how they can configure their CI implementation to run tests, and when those tests should be run.

Developers and testers

Developers and testers will learn about the types of tests they can run as part of CI.

Managers

Managers will find out about additional information that CI can provide, which can be used to monitor the quality and readiness of their products.

Testing in CI

The keys to successful CI are speed and frequency. Speed, because we want to run our builds and unit tests as soon as code is checked in, and we want to deploy and perform testing as soon as the software is built. Frequency, because we want to check in our code regularly, and see the differences from the previous check-in. The more frequently we do this, the easier it is to detect and resolve problems.

However, thorough testing is time consuming and some long-running tests can take 12 to 24 hours, or even longer, to complete. On one hand, we want to run as many tests as possible, but on the other hand, we need to know as soon as possible if our change is responsible for any problems.

In order to get the best of both worlds, we need to consider two different build types: builds triggered by code changes and scheduled builds. Builds that are triggered by code changes can respond quickly enough in order to:

- Compile the source code
- Run unit tests
Nevertheless, adding steps such as deploying the built modules to a production system may cause a delay in getting feedback from the CI system. So you should also configure scheduled builds, that run regularly, but which are not triggered by a code change. This will allow you to:

- Compile the source code
- Run unit tests
- Deploy to a production system
- Run advanced tests

You can schedule these builds whenever you choose—daily, nightly, weekly, etc. You may want to choose a combination, for example, schedule a nightly build every night of the working week, which runs sanity tests, and schedule a longer-running build on the weekend, which will perform a full build, deploy it, and run a full battery of tests on the deployed system.

**Types of advanced CI tests**

There are different types of advanced tests that can be run on a CI system. This section will describe the most common ones.

**Deployment tests**

Deployment tests ensure that the installation worked, that all modules are present and correct, and that the system is configured correctly.

**Integration/System tests**

Unit tests generally test a single module, and if that module is dependent on other modules, they are emulated using mocks. Integration tests, also called system tests, perform similar tests to unit tests, but instead of working against mock objects and emulated services, they work against real implementations.

These tests can only be performed on a system where all of the modules have been deployed and configured correctly.

**Smoke tests/Acceptance tests**

Smoke tests, also known as acceptance tests, test basic functionality to ensure that further, more intensive testing can be performed. For example, if a banking application contains a feature for transferring money from one account to another, a smoke test might check that the option to transfer money exists, and that money can be transferred from one checking account to another. It will not try to transfer money from different account types, or from different customers’ accounts, etc.

**Functional tests**

Functional tests execute scenarios from a user’s perspective, by simulating the user’s behavior. If we take the previous example of transferring money between accounts, we might want to create a battery of functional tests to check before transferring money between different accounts belonging to the same customer and between accounts belonging to different customers.

Functional tests are performed directly against the user-interface of the application being tested, and may also be performed on the API layer.

**Load/Stress tests**

Load tests are designed to ensure that the system performs correctly when subjected to different loads. The system is usually tested by generating a surge or drop in the number of concurrent users, or by running different scenarios concurrently. These tests are also referred to as stress tests when abnormally high loads are placed on the system, which is expected to fail gracefully.
**Triggered builds**

A triggered build is initiated by a developer checking code into the baseline. The CI system monitors check-ins, and starts a build (or queues the build, depending on the configuration) if no other build is running. This build typically causes the code to compile, and unit tests are run on the products of the build. Triggered builds must finish as quickly as possible, so that the system is ready to run the next build.

To keep the response of the CI system as crisp as possible, only unit tests should be run as part of a triggered build.

**Nightly builds**

It is common practice to configure the CI system to perform regular scheduled builds once per day. This is usually done late in the evening, or at a time when coding activity is low. Some organizations might run this kind of build twice per day, perhaps during the evening and at lunchtime. It usually consists of a full build, compiling all of the modules in the product, and running all of the unit tests.

Where possible, the nightly build should be configured to deploy the built modules to a clean production system, and perform integration and functional testing. Remember that you may also need to perform some initialization steps, such as creating database tables, etc, in addition to simply deploying the modules. You might need to package the modules into a full installation, and then run the installation on the target production system.

Once this has been done, the system is ready to be tested. The CI system should be configured to run:

- Deployment tests
- Integration/system tests
- Smoke/acceptance tests

This is a good compromise for teams wishing to get as much feedback as quickly as possible, without holding up developers. If the build is quick enough, consider adding some short integration and functional tests to the build, in order to take advantage of the automatic deployment and get more feedback.

**Weekly builds**

Many development organizations schedule a build once a week or so in order to perform a full battery of tests on the system. This allows the team to get a complete picture of the quality of the system. CI systems should be configured to run:

- Deployment tests
- Integration/system tests
- Smoke/acceptance tests
- Functional tests
- Load and stress tests

The build and all of the tests should run to completion, and reports should be generated by the time developers are ready to start work again. Weekly builds are typically run over the weekend, which is the largest window available when developers are not working.

It is recommended to run shorter functional tests before longer ones. Some functional tests can take a long time to run because of:

- Time to configure the system or to prime the database, and the life, before the actual test is run
- Number of times, or iterations, the test is run. Each iteration runs the same test with a different set of data
- Time to access databases, or other systems which might have latency

Running shorter tests first provides feedback more quickly.
Reporting

When a test is run, its results must be communicated back to the CI system. The tests need to report their results in a format that the CI system understands, and the CI system will collate the results and present them to the user. Ideally, the results should contain enough information to understand why the test failed, without the developer or the tester having to open the test to find out what it does.

Additional considerations for scheduled builds

If you are not already running scheduled builds, you will need to ensure that users can't check code in while the build is taking place. If your CI system supports queued builds, this is not a problem, because the CI system will simply wait till the current build finishes before running the next one. If not, you will have to declare a “check-in freeze” to make sure developers don't check code in at this critical time. Some source-code management systems provide the ability to lock users out for a period of time. If your system does not have this capability, you might need to request developers’ cooperation.

If you are working with development teams that are distributed over different time zones, you will need to ensure you run the scheduled build at a time that is least inconvenient to all developers.

Continuous integration and testing with HP

HP has released a plugin for Jenkins and Hudson, called HP Application Automation Tools, which allows HP Unified Testing (11.00 and later) tests and HP LoadRunner 11.50 scenarios to be run as part of a CI build.

You can introduce both GUI and API functional tests into your CI build, and also run LoadRunner scenarios to determine how the system behaves under load.

You can run tests from HP Application Lifecycle Management (ALM) and HP Quality Center. If you wish, you can also run tests from the file system. You can select specific tests, or even select a folder, and the plugin will run all of the tests in the folder.

The plugin can be configured to run the tests on the build machine (master or slave), or a machine can be specified. If your tests are stored in HP ALM or HP Quality Center test set, you can configure the CI system to use the machine specified in the test set.
Figure 1. Configuring a Jenkins project to run HP tests

The plugin also collates the test results, which are then displayed as part of the Jenkins build report.

Figure 2. Results of HP test runs in Jenkins

With the HP Application Automation Tools plugin for Jenkins and Hudson, you can decide exactly which tests you want to run, and the order in which they should run, and they will run seamlessly as part of the CI configuration.
What's next?

Introducing automated testing to your CI environment is easy. This section summarizes five easy steps to get up and running with HP UFT and Jenkins.

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: Get the tools</td>
<td>If you have Jenkins or Hudson, you can go to the “Manage Jenkins/Hudson” link, and click “Manage plugins.” In the “Available” tab, look for “HP Application Automation Tools” and install it. You will also need automated testing tools, such as HP Unified Functional Testing (UFT), HP Business Process Testing (BPT), and HP LoadRunner.</td>
</tr>
<tr>
<td>Step 2: Get additional training</td>
<td>To jumpstart your productivity using the HP Application Automation Tools plugin and the automated testing tools, you may need some additional training, which is offered by HP or third-party training vendors.</td>
</tr>
<tr>
<td>Step 3: Configure your builds</td>
<td>You are now ready to start configuring the builds that you want to run and create the tests. Decide when the build should be scheduled, which tests you want to run as part of the build, and figure out the order in which they should run. Add one or more jobs to Jenkins, and configure them to run the tests you selected.</td>
</tr>
<tr>
<td>Step 4: Run a test build</td>
<td>Run a build which executes all of the tests to make sure that everything is configured correctly. Ensure that you can see the results of the tests in Jenkins.</td>
</tr>
<tr>
<td>Step 5: Roll out to production</td>
<td>You can now introduce the new build steps to your production instance of Jenkins.</td>
</tr>
</tbody>
</table>

For more information

To read more about HP Unified Functional Testing, go to [hp.com/go/uft](http://hp.com/go/uft).

Get connected

[hp.com/go/getconnected](http://hp.com/go/getconnected)

Current HP driver, support, and security alerts delivered directly to your desktop