HITACHI
Inspire the Next

Continuous Integration with Pentaho

HITACHI
Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes

Contents

Overview 1
Before You Begin 1
Use Case 1: Constant Changes 1
Use Case 2: Sharing the Work 1
Continuous Integration (CI) with Pentaho 2
Code and Content Repository 2
Continuous Integration Server 4
Step 1: Source Code System Definition 4
Step 2: Trigger the Process 5
Step 3: Building the Code and Performing Tests 5
Step 4: Solution Package Generation 7
Related Information 7
Finalization Checklist 8

This page intentionally left blank.

Overview

This document covers some best practices on continuous integration (CI), including how you can use CI during development, automatically building software whenever changes have been made on the system or code. CI functions as a second line of defense to let you know of errors as soon as possible, which lets you focus more on development. Jenkins is a market tool used to facilitate the integration of the components present in CI architecture.

Our intended audience is Pentaho administrators and developers, as well as IT professionals who help plan software development. The examples and instructions are geared toward a situation where you are using Git as a code repository and Maven as a building tool, but other configurations would work if you applied the same principles found throughout this document.

Software	Version(s)
Pentaho	6.x, 7.x, 8.x

The Components Reference in Pentaho Documentation has a complete list of supported software and hardware.

Before You Begin

Here are some terms you should be familiar with:

- **Continuous integration (CI)**: The processes of continually and regularly updating the code base to reflect changes made to it by developers, to keep final integration and merging of the code more simple, easy, and quick.
- **Versioning:** Creating a new version of a system or software is referred to as *versioning*. A *version control system (VCS)* is a method to keep records of changes to software over time so that you can avoid conflicts of different people saving over each other's changes and can also roll back the code to a previous version if you need.

Use Case 1: Constant Changes

The central big data administration department of Company A needs to ingest data from several data sources. They have many developers working together in the different ETL projects around the ingestion process. The managers' main concern is how to handle the continuous changes in the processes and guarantee the quality of each development.

Use Case 2: Sharing the Work

Company B has developers distributed around the world, and seeks to manage a Pentaho ETL project using Agile methodologies so that they can share the workload of their projects worldwide. Developers from different time zones working in the same projects must have a centralized repository and integration platform for the entire team.

Continuous Integration (CI) with Pentaho

Continuous integration within Pentaho projects is important to the success and lifecycle of software development. Here are the topics covered in this section:

- Code and Content Repository
- Continuous Integration Server

This end-to-end diagram shows the workflow between various stages in the CI process, and highlights the positioning of the main tools used in the field:

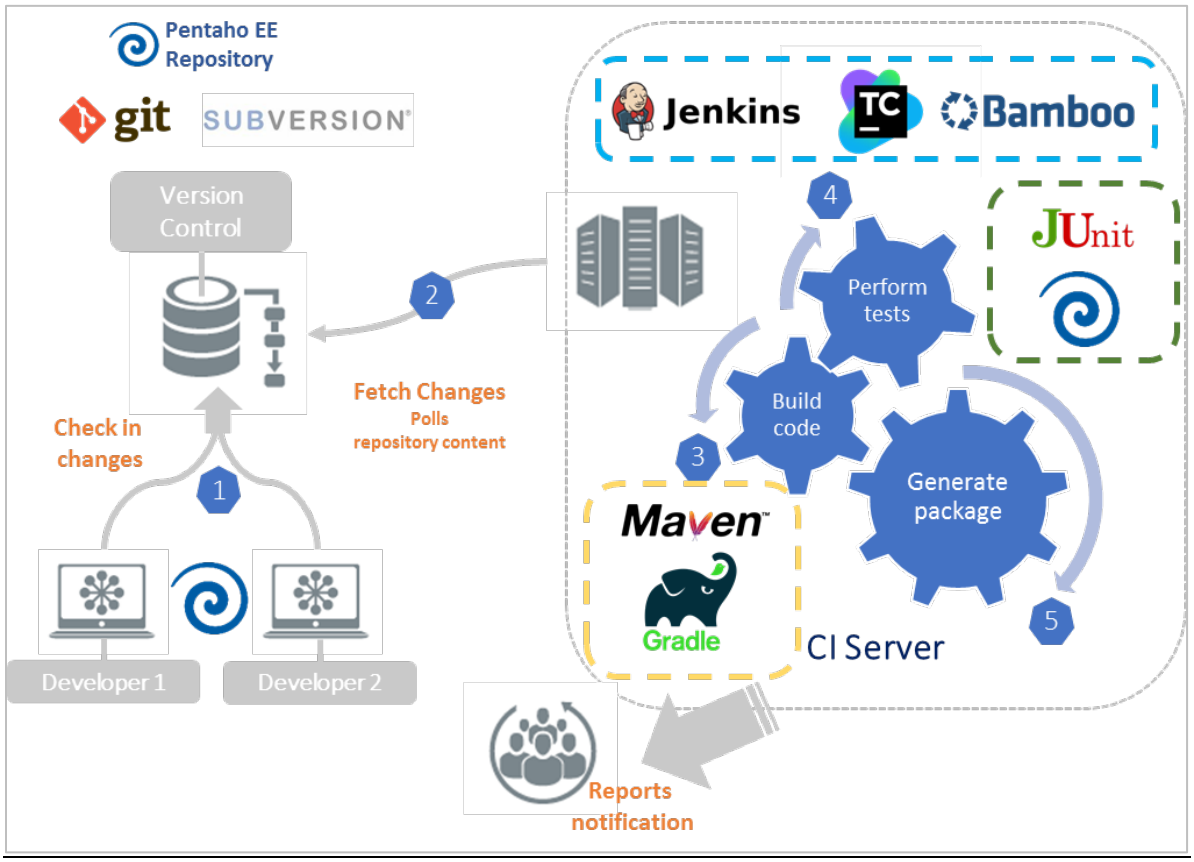

Figure 1: Continuous Integration Workflow and Tools

Code and Content Repository

A centralized repository is important to your development team's collaboration. Pentaho is equipped with an Enterprise Repository that offers basic versioning capabilities. However, if you use Pentaho Enterprise Repository, you can only use Jenkins in *scheduled* mode. This means that you would not be able to build a total CI solution, because changes in any Pentaho artifacts would not trigger automatic execution of your Jenkins job. Table 1 has a list of repository recommendations.

We recommend basic versioning for organizations that are not familiar with version control systems (VCS) or for small development teams with limited agile development requirements.

Many organizations already have third-party enterprise VCS tools like Git or Subversion.

Table 1: Repository Recommendations

Recommendation	Details
Use a Third-Party Enterprise Repository	Use this kind of repository on development environments, because those environments have better integration with CI capabilities, mature versioning systems, and more version control capabilities needed for agile developments.
Set Up Revision or VCS	Revision or VCS is mandatory. It allows you to centralize the code, track the changes made by several developers working in parallel, and prevent loss of code portions.
Organize Your Repository Structure	Organizing repository structure in a logical way is helpful in the building process. The source code of your file and folder structure should have the same logic organization as the structure in the final build package.
Place Configuration and Dependencies in Repository	Place the configuration and dependencies, including property files, DDLs, and database connection configurations, in your repository. This allows you to make fresh checkouts with minimum effort and no extra complex configuration.
Prepare Your Repository	Prepare your repository for every environment you will need, creating configuration files for each environment and using environment variables to specify the environment. You will store the configuration files in the repository, and use a custom Jenkins job/process to set up the environment automatically before every execution, according to the current environment.
Use Pentaho Server	A Pentaho server is mandatory to develop the solution in Pentaho analytics development (such as reports, dashboards, and analysis views). If your code repository is a working one, use download and upload capabilities to integrate developments.
PDI Developments	For PDI development: **File-based solution**: PDI developers check out jobs and transformations to the local drive, and then do their work. **Pentaho Enterprise Repository**: developers can work with jobs and transformations while connected to the Pentaho repository.
Customize Java Options	Use `PENTAHO_DI_JAVA_OPTIONS` to customize things such as Java options, memory, cache location, etc.

Continuous Integration Server

The CI server is the main component in the CI pipeline, orchestrating and organizing the entire process from a commit to a tested solution package ready to be used or distributed.

Jenkins is CI software that helps automate software development, and is one of the most common tools used for CI within different Pentaho deployments. Here are a few of its main features:

- Server-based system that supports many VCS integrations (such as SVN and Git)
- Offers capabilities to generate output in testing, such as solution package building
- Open source and can be adapted as needed

The first step is to create a CI project for Jenkins. The following sections break this down into distinct parts so that you can better understand each stage.

Step 1: Source Code System Definition

The server needs to be able to access the source code. Jenkins offers capabilities to integrate with many VCSs like Git and Subversion through native plugins, allowing you to connect directly to a specific URL and branch. This makes Jenkins the workspace of your project once the checkout/clone task is done.

You must download (export) code from the development environment using the import/export capabilities if you are using a Pentaho Enterprise Edition (EE) Repository and you plan to deploy your solution code to the test environment repository. Backup and Restore Pentaho Repositories has complete instructions on how to do this.

To configure the source code:

1. Select your desired CI project and click **Configure**.
2. Find the **Source Code Management** tab and select the radio button for your option.
3. Enter the Repository URL, credentials, and other information to set up integration.

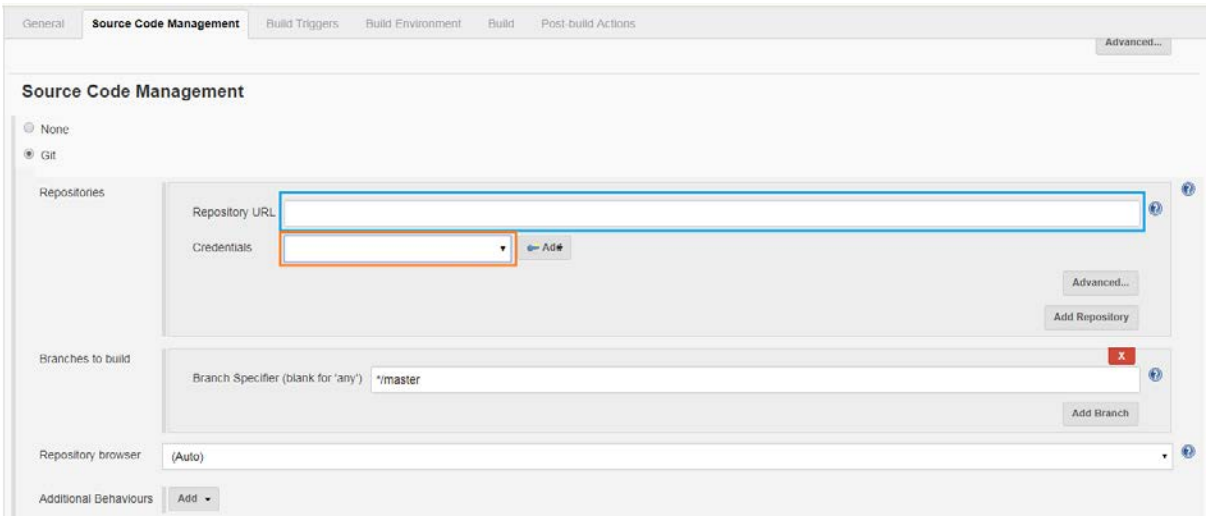

Figure 2: Source Code Management

Step 2: Trigger the Process

The execution gathers the code from the centralized code repository but there are different options you can select to trigger the process, based on your needs and CI maturity:

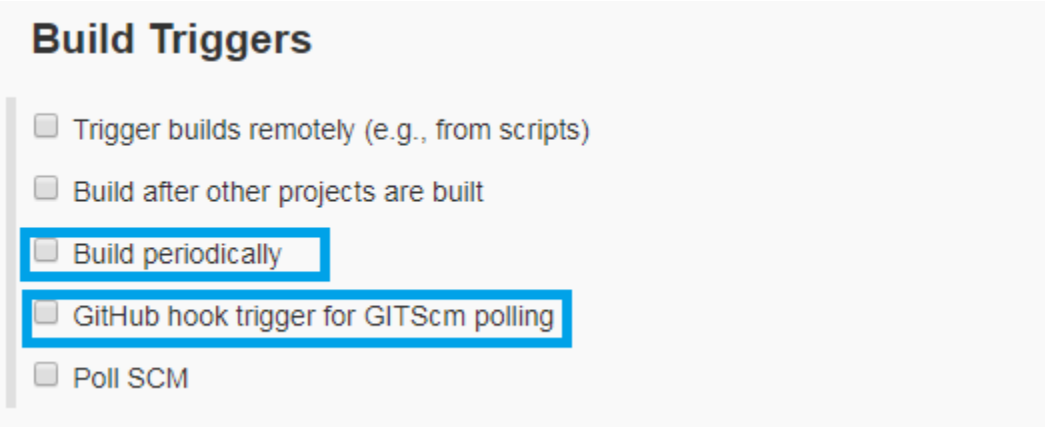

Figure 3: Process Triggers

Actively Listen for Repository Changes

CI server tools offer the ability to actively listen for repository changes through a plugin, initiating integration when the tool detects changes. Jenkins has such a plugin for Git.

Schedule the Execution

You can schedule the process to be executed based on a time rule, and then build periodically. For example, you can set the rule to **nightly** if you want nightly builds.

Step 3: Building the Code and Performing Tests

You do not need to compile the Pentaho code, because the code is interpreted at execution time.

We recommend using a product like Apache Maven to resolve dependencies, define targets, or organize the logical distribution of the solution.

For extract, transform, and load (ETL) projects, the main premise is that Pentaho is used to test Pentaho. The development of the solution must include the creation of ETL processes to test the behavior, take a sample of input data, use developed functionality, and evaluate if the results match as expected. Make sure you run environmental tests such as these, before running ETL tests:

- Can I connect to the database?
- Is the Hadoop cluster up and available?
- Can I run a MapReduce process?

The main ETL items to test are those that are used in many parts of the processes:

- Mapping-Transformation (Sub-Transformation)
- Metadata Injection-driven execution
- Job and transformation execution

Place your tests in a common location within the repository to allow the CI process to execute it automatically based on certain rules. For example, everything that is placed under `rep/ETL/tests/` will be executed every time the process is triggered.

Triggering Pentaho Testing

There are many ways to perform the testing procedures. Two possible recommended ways to test Pentaho are:

Method 1: **Use built-in scripts to execute Pentaho jobs and transformations**

Use the PDI Client in the Jenkins box. The PDI Client can execute code placed locally in the filesystem. Create an execute shell on your Jenkins project to instantiate Kitchen or Pan commands to execute each `.ktr` and `.kjb` test oriented.

1. Define the environment variables needed before the PDI execution call. We recommend you use a specific `KETTLE_HOME` variable to access Pentaho configuration specifically attached to that environment.
2. Make a rule to decide which transformations or jobs should be executed by the testing procedure.
3. Handle the results properly. Building procedure depends on testing execution to decide if the build process was a success or a failure (Kitchen codes or Pan codes).

Use a secure socket shell (SSH) connection in the box where the PDI Client is installed, allowing you to isolate executions and control over permissions.

4. Execute jobs and transformations using the Pentaho server, calling REST API. In this case, the code is placed in the Pentaho EE Repository and the execution is performed by the Pentaho server placed in the test environment for this purpose. See a reference to the available API endpoints.

Method 2: **Use Maven plugins**

Enable Maven plugins, like Maven Surefire, to perform automatic testing capabilities, depending on the level of maturity and the number of tests needed.

Evaluate Testing Results

We recommend using standard output formats for testing results. Using a reporting format as a JUnit XML output format allows you to incorporate those results into Jenkins and establish success/health factors to determine if a build satisfies expectations. Using JUnit can become part of the post-build actions of your process.

Each execution on PDI has an EXIT CODE associated with it (Kitchen codes or Pan codes) that must be used to determine the results of the test.

Step 4: Solution Package Generation

Your resulting output should be a ready-to-use package with the solution after building and testing is finished. The package must be ready to be incorporated in the automatic deployment pipeline or to be deployed manually as part of your company criteria.

This packaging process can be done through script and extension points within Jenkins, or you can incorporate Apache Maven capabilities as part of the build lifecycle.

Related Information

Here are some links to information that you may find helpful while using this best practices document:

- Apache
  - Build Lifecycle
  - Maven Surefire
- Jenkins
  - Building a Software Project
  - Git Plugin
  - Jenkins Homepage
  - Scheduled Build Plugin
- Pentaho
  - Back Up and Restore Pentaho Repositories
  - Carte - API Endpoints
  - Components Reference
  - Kitchen Status Codes
  - Pan Status Codes

Finalization Checklist

This checklist is designed to be added to any implemented project that uses this collection of best practices, to verify that all items have been considered and reviews have been performed. (Compose specific questions about the topics in the document and put them in the table.)

Name of the Project:_____

Date of the Review:_____

Name of the Reviewer:_____

Item	Response	Comments
Did you maintain a code repository as the starting point of your CI workflow?	YES_____ NO_____	
Did you use Jenkins to build your test system?	YES_____ NO_____	
Did you test your solution in a clone of your production environment?	YES_____ NO_____	
Did you test your solution instead of just testing the features?	YES_____ NO_____	
Did you use Pentaho to test Pentaho?	YES_____ NO_____	