HITACHI
Inspire the Next

Driving Pentaho Data Integration (PDI) Project Success with DevOps

HITACHI
Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes
5/2019	1.0	Beppe Raymakers	Original
2/2020	1.1	Beppe Raymakers Megan Brown	Update to 9.0

Contents

This page intentionally left blank.

Overview

This document introduces the Pentaho Data Integration (PDI) DevOps series, consisting of best practices documents whose main objective is to provide guidance on creating an automated environment to iteratively building, testing, and releasing a PDI solution. This can be faster and more reliable, resulting in a high-quality solution that meets customer expectations at a functional and operational level.

DevOps is a set of practices centered around communication, collaboration, and integration between software development (Dev) and IT operations (Ops) teams, automating the processes between them. Its main objective is to reduce the time between committing a solution increment and having that increment available in production, while maintaining high quality.

Our intended audience is Pentaho administrators and developers, as well as IT professionals who help plan software development.

The information in this document covers the following versions:

Software	Version(s)
Pentaho	7.x, 8.x, 9.0

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

Introduction to DevOps

When you are using Agile solution development principles, the goal of each development iteration or sprint is to produce a fully functioning piece of software that has been thoroughly tested and is ready to release into the solution that is being developed. These short intervals are there to deliver results quickly and incrementally, using frequent feedback loops to keep the project on the right track and to maximize the value of the solution.

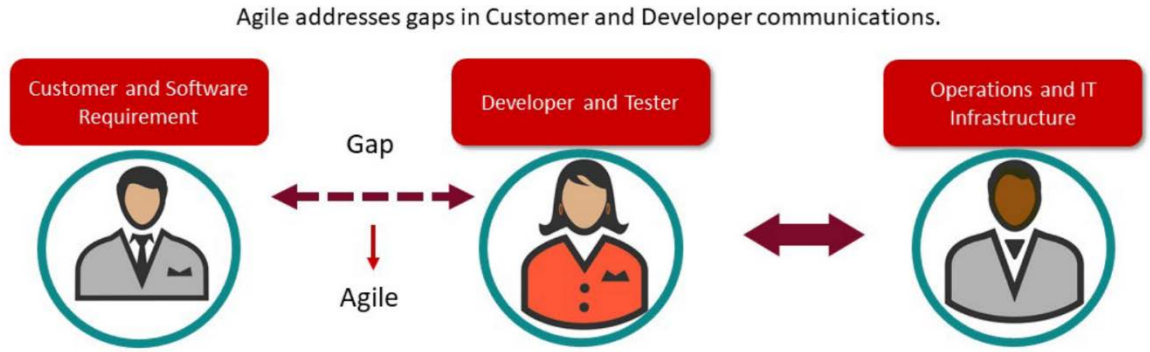

Figure 1: Agile Development

Delivering on such promises requires more than having such an Agile feedback loop at the functional level. To accelerate project success, we need the same continuous feedback and release process at the code level of the solution. This guarantees that the new solution increment is properly working according to technical specifications and that we can reliably integrate the iteration into the production environment.

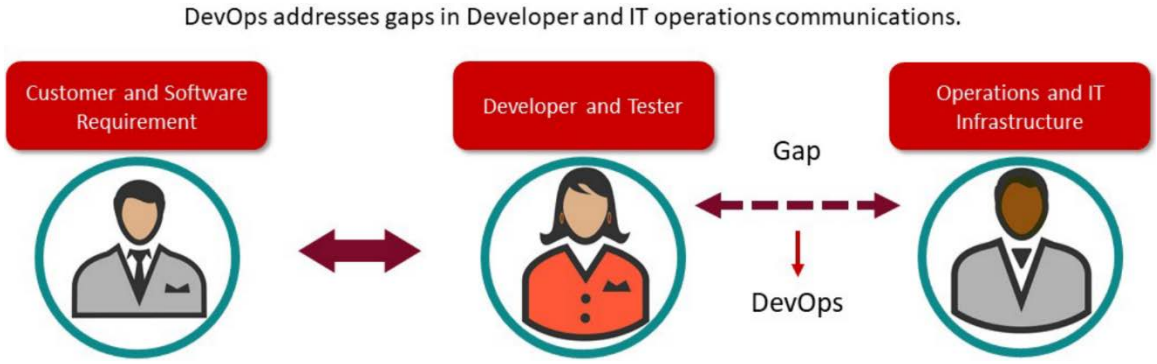

Figure 2: DevOps

The set of practices intended to reduce the time between committing an increment to a solution and the increment being placed into production, while ensuring high quality, is called DevOps. Using Agile and DevOps methodologies in tandem drives project success and productivity:

- Agile for planning and governance.
- DevOps for development, testing, deployment, and operations.

Key DevOps concepts:

- DevOps is a methodology to create software solutions.
- DevOps is based on the integration between software developers and system administrators.
- DevOps helps to manufacture software faster, with higher quality, lower cost, and a very high frequency of releases.

The following diagram depicts the DevOps lifecycle and identifies some tools that can support your Pentaho project with adopting a DevOps strategy. These are just examples for illustrative purposes only, and some of the tools will be discussed later in this and following documents in the series.

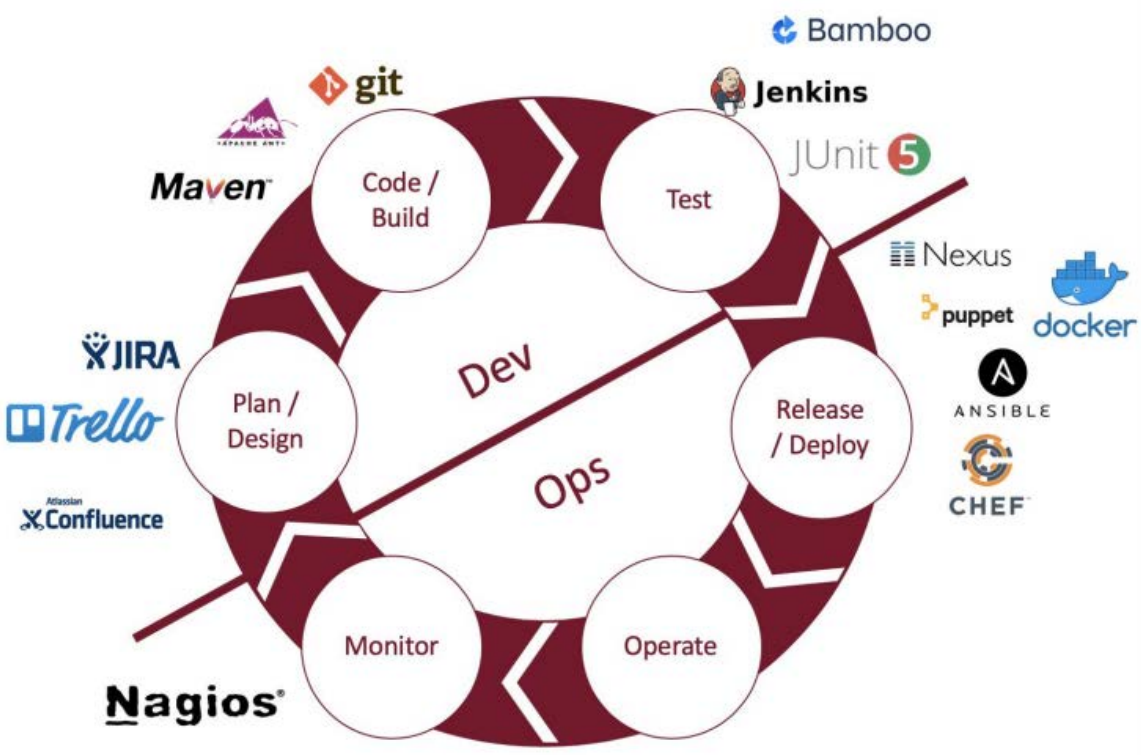

Figure 3: DevOps Lifecycle and Tools

DevOps Maturity Phases

There are several phases in the process of adopting DevOps in an organization.

You can find details on these topics in the following sections:

- Project Foundations
- Continuous Integration (CI)
- Continuous Delivery
- Continuous Deployment (CD)

Here is a representation of the different DevOps maturity phases:

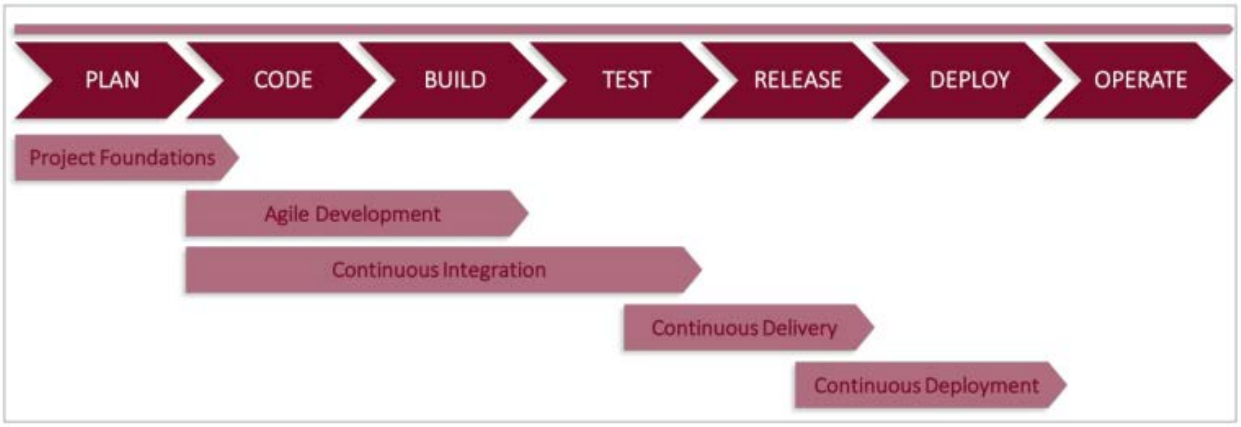

Figure 4: DevOps Maturity Phases

Project Foundations

Onboarding on your project's DevOps journey starts with setting up the right project foundations for your PDI solution on which you can build your development, CI, and CD strategy. Such project foundations consist of a set of best practices for starting your PDI project, including project structure and the project's technical governance and lifecycle management strategy.

Starting your Data Integration (DI) project means planning beyond the data transformation and mapping rules to fulfill your project's functional requirements. A successful DI project proactively incorporates design elements for a DI solution that not only integrates and transforms your data in the correct way, but does so in a controlled manner.

You should consider such nonfunctional requirements (NFRs) before starting the actual development work, to make sure your developers can work in the most efficient way. Forcing such requirements in at the end of the project could be disruptive and hard to manage.

All these best practices regarding setting up your project foundations are discussed in detail in *PDI Project Setup and Lifecycle Management*, the second document in our DevOps series, and the main points are summarized in the following subsections of this document.

Version Control System (VCS)

A team collaborating on a DI solution requires a shared repository of artifacts. Store your DI project content (PDI jobs and transformations, external SQL or other scripts, documentation, or master input files) in a central, version-controlled repository. This repository should also support other team collaboration and release management capabilities.

Although the Pentaho repository offers a basic revision control system, it is not intended to replace common market tools, such as Git. If you need the advanced features of an enterprise VCS, we recommend that you use one.

The preferred and recommended option to integrate your Pentaho solution into your organization's IT standards is by using an **enterprise version control system (VCS)**, such as Git, SVN, or BitBucket.

We recommend using a VCS in your development and test environments to:

- Integrate with the IT standards defined in your organization.
- Simplify collaboration between team members by providing a central location for your DI solution's artifacts.
- Standardize the solution structure.
- Keep track of changes and revert changes when needed.
- Make use of VCS branching functionality to implement a new feature in an isolated area that allows you to prove that it works before it gets merged into the main code base. Branching also allows more than one developer to implement features in parallel without creating code conflicts and allows you to separate production-ready code from untested code. User branching wisely, based on your active projects and your organization model.
- Make use of other advanced VCS features to support the lifecycle management of your project, such as merging, tagging, and so forth.
- Support code integration happening in the CI automatic lifestyle.

Content and Configuration Management

Your DI solution will run in different environments as it moves through the development and test lifecycle. This means that you need to separate the PDI artifacts and code (for example, the PDI jobs and transformations) from the configurations (the connection details, directory paths, and other variables) that make the code run properly. Whenever you invoke the solution, you use a specific environment configuration.

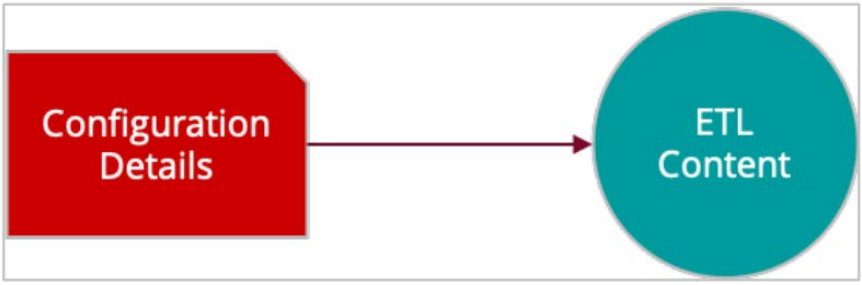

Figure 5: Running ETL Content with Different Environment Configurations

When you are developing a project that will be running in various environments, it is not only important to separate the project code and configuration, but also to stick to a consistent solution for configuration management. Every environment that your project will be running on will have a dedicated configuration setup, and we recommend you keep this environment configuration in a VCS.

When you use the ignore feature of the version control tool, the local configuration for developers is not checked, so every developer can have a different configuration. It is helpful, however, to keep a default local configuration checked in so that developers joining the team can create a copy of that for their local environment and get started quickly. Switching between configurations can then be as easy as setting an environment variable pointing to the folder containing the desired environment configuration. If that environment variable is not set, the location of the local environment is assumed.

The ETL must be designed in such a way that all external references (such as database connections, file references, and all internal calls to jobs and transformations) are made using variables and parameters, which get their values from the configuration files part of the project configuration. When the developer parameterizes his ETL, the solution becomes environment-agnostic, a key attribute powering the solution's lifecycle management.

Structured Solution Approach

Starting a successful DI project involves working in a structured and repeatable fashion which is agreed upon by the team. This includes agreeing on a structure for your content and configuration repositories and various standards that define how the development and governance of the solution will be managed.

Typically, this structured solution approach needs:

- A folder and file structure for your content and configuration VCS repositories.
- An organized way to manage environment configuration (PDI variables and parameters), including settings that are environment-aware, such as usernames, passwords, database connections, and so on.
- The ability to launch Spoon for a specific project and environment configuration.
- An approach for multiple developers to have a local development environment.
- Naming standards for your jobs and transformations or other solution artifacts.
- A logging and monitoring approach, and an error-handling approach.
- A structured approach to documenting the solution.
- An approach to solution restartability whenever an error occurs.

You can capture some of these agreed-upon practices in a **Development Guidelines** handbook. This handbook should include guidelines and standards that must be followed by the entire team when developing the solution, Other standards are captured in things like **templated VCS repositories** that showcase how to structure those in terms of files and folders.

Still other structures and approaches will be captured in a **DI framework**. The DI framework is a set of jobs and transformations that acts as a wrapper around your actual DI solution, taking care of all setup, governance, and control aspects of your solution. By abstracting these concepts from the actual DI solution, the development team can focus on the actual jobs and transformations.

Continuous Integration (CI)

Continuous integration is often the first step of a DevOps implementation, and is the most common level achieved. It supports the Agile project delivery methodology with a software development lifecycle that has great emphasis on test automation to check that the application is not broken whenever new iterations are integrated into the central solution repository.

Development teams need to continuously merge their changes back to the central solution repository as often as possible. This way, the team's changes can be validated by creating a solution build and running automated tests against the build, giving the developer immediate feedback and the ability to easily fix bugs. This allows you to avoid integration problems that happen when people wait for release day to merge their changes into the central repository.

Continuous Delivery

Continuous delivery builds on continuous integration to make sure that you can release new iterations to the customer quickly and in a sustainable way. This means that on top of automating your testing, you can also automate your release process. As a result, you can deploy your application at any point in time without much human intervention, up to the point of the actual environment deployment where manual intervention is still needed.

Continuous Deployment (CD)

Continuous deployment goes one step further than continuous delivery by providing an end-to-end solution deployment into production without any human intervention. As a result, every solution increment that passes all stages of your software development lifecycle is automatically released to your end users. This way, continuous deployment accelerates the feedback loop with your end users by contributing to a continuous release process.

DevOps Benefits

Before the use of DevOps practices, software development and operations were composed of separate teams working in relative siloes, making the end-to-end process slower and very inefficient.

DevOps's practices try to address these challenges by establishing a collaborative and cross-functional ecosystem. Its main objective is to promote and improve the collaboration between all development and operations to jointly carry out the different stages of a software development lifecycle. You will be able to go from early stages of planning through delivery and automation of the delivery process, so you can:

- Improve quality and increase the frequency of deployment.
- Decrease the amount and severity of failures with software releases.
- Improve resolution procedures and mean time to recovery.
- Improve solution quality and realize a faster time to market.

Project Environments

Adopting the DevOps development and operations lifecycle for your project requires several project environments. At the beginning of a project, you need to figure out which environments are needed, and how they are implemented and hosted.

There should always be at least isolated development environments for each developer, and a test environment with a continuous integration server running tests. The integration environment is necessary if the project requires user acceptance testing. The staging environment is required if the project team must provide a space for long-running acceptance testing and/or content promotion and development support.

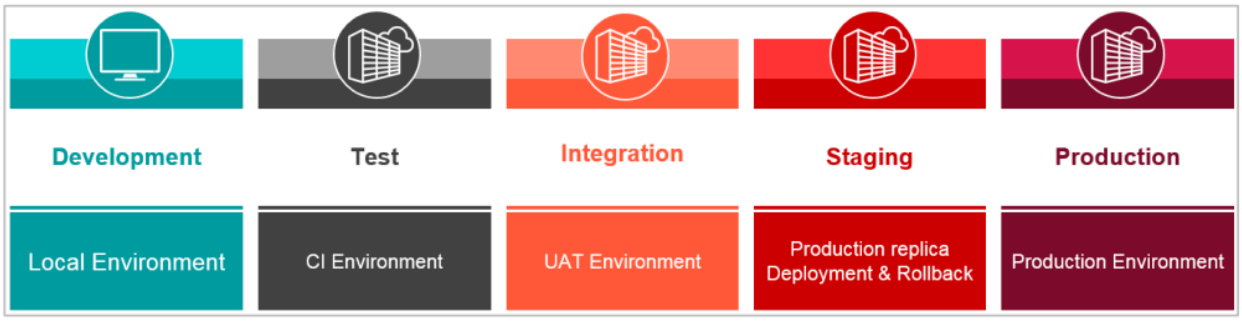

Figure 6: Project Environments

You can find details on these topics in the following sections:

- Development Environment
- Test Environment
- Integration Environment
- Staging Environment
- Production Environment

Development Environment

This is a shared-nothing environment local to each developer. Each developer has their own instances of databases, Hadoop clusters, BI servers, and so on. This is often achieved by running local instances of databases, or VMs for more complex setups like a single-node Hadoop cluster in a VM.

Test Environment

This is an isolated environment where the continuous integration server runs the test suite. Again, the test environment has its own set of databases, Hadoop clusters, and so forth. The test environment is isolated, so the test suite can restore a “clean slate” state and load known data fixtures without interfering with anyone’s work. We recommend that building this environment be a repeatable automated process, so the test server can be easily moved or expanded to multiple worker machines using consistent configuration.

Integration Environment

This is an isolated environment where successful builds are automatically deployed to on a regular basis. This is the best place to experience the current state of the solution. Its main purpose is to make sure that deployment works as expected, and to enable user acceptance testing. Like with the test environment, it is crucial to be able to replicate the integration environment from scratch using some variant of provisioning technology.

Staging Environment

This is an isolated environment that closely mimics the characteristics of the production environment. It is regularly provisioned with a dump of the state of the production environment. The staging environment's purpose is to closely simulate the deployment and rollback procedure. It is also sometimes used for longer user acceptance tests, since it tends to be stable for longer periods of time. If that use case is the dominant one, this environment is often referred to as the quality assurance (QA) environment. As with the test and integration environments, it is beneficial to be able to restore a staging environment from scratch.

Production Environment

This is the environment the solution runs on productively. Access to this area is usually heavily regulated. The developers of a solution are usually not accessing this area directly. Some knowledge about this environment is necessary, though: hardware specifications (CPU, HD I/O throughput, RAM), network characteristics, availability of an internet connection, or any exiting or enforced configuration options do inform architecture decisions in solution design.

Related Information

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

- Nonfunctional Requirements (NFRs)
- Pentaho Components Reference
- Pentaho Customer Portal
- Pentaho Documentation
- Training and Certification – Hitachi Vantara Global Learning