AMIS

Edisonbaan 15

Postbus 24

3430 AA Nieuwegein

T +31(0) 30 601 60 00

E info@amis.nl

I amis.nl

BTW nummer NL8117.70.400.B69

KvK nummer 30114159

# Oracle Continuous Delivery Pipeline

## Overview, considerations and tools for

## Oracle Database and Fusion Middleware

**Authors**

Robbrecht van Amerongen

Laurus Keijzer

**Version / status**

Version 1 / Final

**Date**

25 March 2016

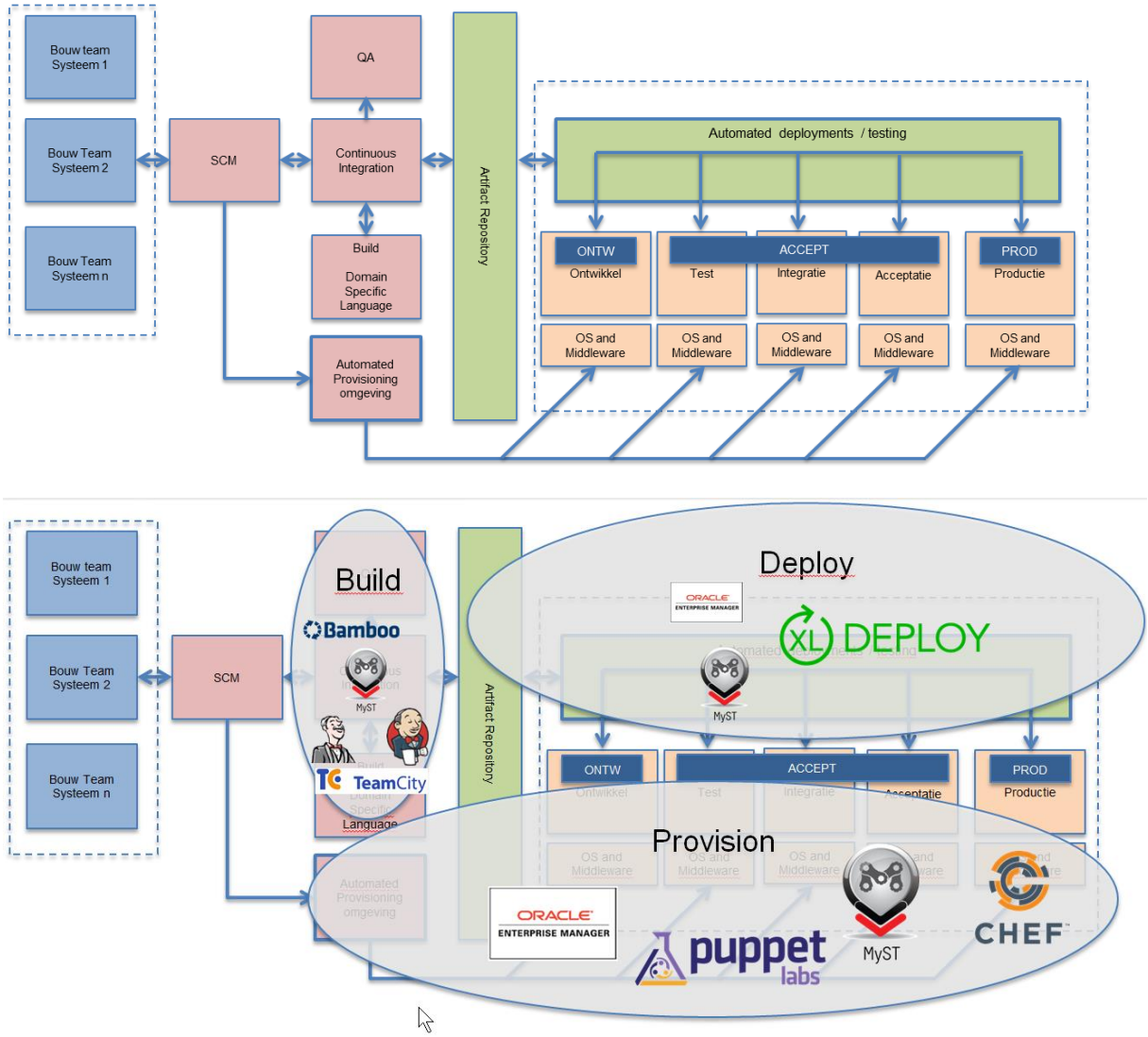
**Filename**

overview continuous delivery and devops approach amis.docx

# Index

# 1 Management Summary

It is important to consider the broad range of tools and product for creating your Continuous Delivery Pipeline. From this perspective we share our vision on the Continuous Delivery Pipeline specifically for the Oracle stack. This document is  an overview and guide for the complete delivery pipeline and shows our preferred product for build, provisioning and deployment of Oracle components. This document is specifically suitable for organisations who are looking for a complete new set of tools for their continuous delivery process and development teams. We realize the details of the continuous delivery process and the preferred tools need to be tailored for your organisational processes and products. This document gives guidance in the usage of our preferred tools within the Oracle tool stack.




**Oracle Continuous delivery pipeline and tools**

# 2 Challenges to address with Continuous Delivery

This chapter describes some of the challenges we face when implementing Continuous Delivery on an Oracle Technology stack.

## 2.1 What is Continuous Delivery?

The goal of continuous delivery (CD) is to help software development teams optimise their process by simultaneously automating the process of software delivery and reducing the batch size of their work. This allows organizations to rapidly, reliably, and repeatedly deliver software enhancements faster, with less risk and less cost.

Continuous Integration (CI) is the practice of automatically building and testing a piece of software; either each time code is committed by a developer or in environments with a large number of small commits, or a long-running build on a regular scheduled basis.

Continuous Delivery (CD) goes a step further to automate the build, packaging, deployment, and regression testing, so that it can be released at any time for production.

Continuous deployment takes this another step further, in that code is automatically deployed into production, rather than when the business decides to release the code.

## 2.2 Manual scripting or automating your installation, deployment, build and configuration

“Productivity of my development team was reduced by 20% due to an unstable middleware platform, at one point we were re-starting it 4-5 times per day … out of a team of ten, it cost me two FTE’s ” Integration Manager - Utility Company

“The start of the project was delayed by 3 months waiting for a suitable Dev Platform, the start of SIT was delayed even longer for similar reasons ...” Project Manager – Logistics Company

The manual process of provisioning Oracle Middleware environments, and the testing and deployment of code into these environments is one of the most painful, resource intensive and stressful parts of any Oracle Middleware implementation.

An incorrect installation is one of the most common causes of issues when developing, deploying and operating Oracle Middleware solutions. Small errors, such as misconfiguration of a middleware component, can cause issues which are difficult to diagnose and rectify; often requiring many weeks / months of man effort to resolve; resulting in project delays, cost blow out and missed milestones.

Automating the processes for the installation, configuration, build and deployment of Oracle Middleware without scripting assures, in our view, a predictable and low risk environment.

For each environment, a separate platform is created to define the environment specific configuration information. Creating a platform blueprint, platform model and provisioning a new fully operational FMW environment, is a simple three step process, which can be performed in minutes.

Reduce Risk, Decrease Costs, Speed Up Time To Market; Build and Deploy Oracle Middleware Code in Minutes!

## 2.3 Easy Questions, difficult to solve, unless automated

- How many outages have you experienced over the last month due to failure environments?
- How long does it currently take to stand up a single FMW environment?
- What is your current level of capability around Continuous Integration and Delivery?
- What do you see as the bottlenecks in your project deployment process?
- How long is the average project delivery cycle?
- Do you have a dedicated DevOps / Automation team – If so, how many FTE in team and
- who do they report to?
- What automation tools are you utilizing and how effectively are these tools delivering?
- Over the next year, how many new projects do you expect to be rolling out?
- How many different teams are involved in a typical rollout of a project?

## 2.4 Continuous Delivery and Oracle Technology

**Products and Versions**
Not all products are available at the same time. This means that you have to handle the situation where several different versions of the Oracle product stack are operational at the same time. Some products are certified to work together. Without a good architectural design and clear guidelines the landscape can evolve to a complex structure that is difficult to understand and oversee.

**Licensing**
When designing a continuous delivery pipeline you need to take the Oracle license model into account. Some architectural decisions can have a significant impact on the license structure of your environment. Sometimes it is effective to decide upon a less optimal continuous delivery model due to financial or license considerations.

**Cloud or on premises**
Even when your company has an on premises strategy it is essential to be prepared for cloud migration. Numerous companies who were not considering cloud 3 years ago, are now hosting some of their systems in the cloud. Systems that are designed with cloud concepts (multitenancy, scalability, micro services etc.. ) are easy to migrate. Ignoring this development will only postpone work and create unforeseen complexity.

**Measurements of success**
What is the measurement of success for continuous delivery. The success criteria for this diverts a little from the traditional criteria. With continuous delivery the focus will be aimed on throughput and continuous improvement. The success criteria will shift towards the measurement of adaptability, and the ability to respond to change.

**Legacy systems**
Take your current (legacy) systems in to account when implementing continuous delivery. Some of these systems will be replaced soon by the new software. But not all of them. Consider these systems in your continuous delivery strategy. Ignoring legacy systems willeventually slows the delivery process.

**Outsourcing policy**
The policy with regards to outsourcing is an important consideration when implementing the delivery pipeline. External vendors for outsourcing tend to comply to the generic tools. When you are pursuing a massive outsourcing strategy you need to think about the tools in the delivery pipeline.

**Number of systems and diversity of systems**
The number of systems and the number of different types of technology are an important factor in the size and complexity of the continuous delivery pipeline. The most common strategy in this field is to first consolidate to a limited number of different systems and solutions before consolidating to a continuous delivery process. Consolidation helps to increase quality and reduce costs for maintenance. This is also a perfect step towards cloud implementation of your current systems.

**Governance**
The result of the continuous delivery pipeline; an actual platform with a deployed number of applications on top of it, need to comply to rules and regulations for your industry. Design the continuous delivery pipeline in such a way that it is easy to deliver the traceability, roles and approvals needed for your industry. Most tools offer these kind of capabilities.

**Configuration Management**
The key for a good continuous delivery is a clear devision between data, software and configuration management. The challenge is to split the configuration form the actual software systems and be able to provision systems in these 3 parts.

# 3 Overview solution

## 3.1 Goals

The final goal for Continuous Delivery is to provide a reliable, flexible and repeatable software delivery process. Where the organization is able to release new features and the business is able to decide when to enable / implement these features in their final product. All activities must provide a value to this end product. Continuous delivery contains a self-improvement cycle where, next to the product improvement, the actual process is being made and constantly evaluated. Focus is on automating as much as possible. Especially for the repeatable processes. The final result is Automation of tasks and automation of validation.

## 3.2 Conditions

There are a couple of conditions you need to have in place to make the implementation of continuous delivery more easy.

- For the effective implementation of continuous delivery the organization needs to have a clear image of their delivery process.
- Your technology needs to support a high level of automation.
- Design for maximizing the internal dependencies within a system and minimize the dependencies between systems.
- Design with the notion that you can delete everything and build your complete infrastructure and deployment from scratch. Make sure you can re build your complete infrastructure quickly
- Practice a lot. Run your delivery process at a high frequency. To make sure the feedback cycle is short. Engineers should get fast notification on errors or failure.
- Focus on throughput and make sure the changes are delivered in small batches.
- Deliver high quality solutions. doing something right the first time is better additional work after a change.
- Focus on automation of validation. Make sure you run automated test on everything and share the results.
- Make sure your tasks are idempotent; this means that multiple runs of the same action will have the same result.
- Make the goals, process and the results visual for everybody in your organization.

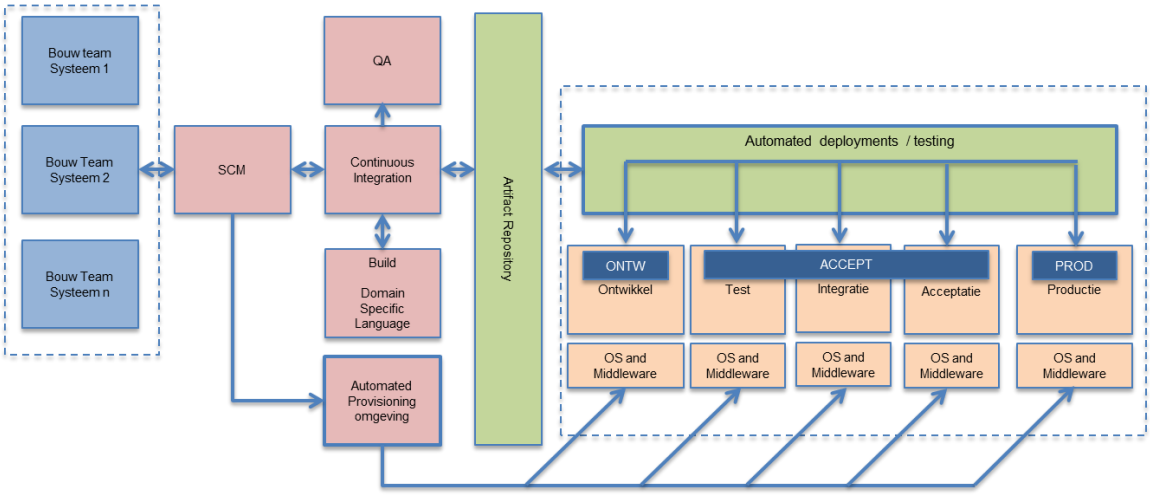
## 3.3 Structure / DevOps

Continuous delivery is often compared to DevOps. This is partly true. Continuous Delivery relies a lot on DevOps practices. The biggest obstacle for continuous delivery are siloes in the organization structure. The biggest shift is the responsibility. This responsibility is going to shift form the different application layers (OS, platform, network, applications etc.) to the end product. So you get one team responsible for the availability, quality and innovation of one specific product.

For most organizations it is quite a challenge to make this shift while maintaining current systems operational. This often means a revision of both the organization structure and the processes.

## 3.4 Components in the solution

The continuous delivery solution consists of 3 major parts; build, provision and deploy.




The build part contains the development teams, the source control system (SCM), build engine, quality engine and a preparation for platform provisioning. This is the blue and pink section in the diagram above.

The provisioning part contains the provisioning engine. This engine orchestrates the provisioning of the environments and the platform configuration management of the target environments. This part handles all the generic and individual settings for each environment and makes sure that lifecycle management and patching of the platform and operating system is executed. This is the light red part in the diagram.

And the third component in the solution is the deployment of application code, binaries and application configuration. For this part we need a binaries artifact store, a deployment engine, test engine and an orchestration tool to manage all dependencies between all tasks on the platform.

# 4 Components of Continuous Delivery solution

This chapter contains a description of the technical components for continuous delivery.

## 4.1 Development

**INTEGRATED DEVELOPMENT ENVIRONMENT**
An Integrated Development Environment (IDE) is the desktop environment for the developer and consists of an editor for the software code and a software component management system for managing the libraries in the system. The IDE is often integrated with the issue tracking system, the source control system and the build engine.

- Examples : JDeveloper, Eclipse and IntelliJ.
- For Oracle JDeveloper is the most common editor,
- Alternate Names: Software editor / Development Environment

**VERSION CONTROL SYSTEM**
A Version Control System allows multiple users to simultaneously collaborate on and record changes to a set of files or project over time.

- Examples : GIT/Subversion /Stash/CVS
- For Oracle Middleware GIT is the preferred Version Control System
- Alternate Names: Source Control System

**BINARY REPOSITORY MANAGER**
A binary repository manager is a software tool that stores binary files used and produced in software development. It centralizes the management of all of the binary artifacts generated to overcome the complexity arising from the diversity of binary artefact types, and the dependencies between them.

- Examples : Artifactory/ Nexus
- For Oracle Middleware Nexus is the preferred system.
- Alternate Names : Artifact repository

## 4.2 Build and continuous integration

**BUILD AUTOMATION TOOLS**
Build automation involves scripting or automating the process of compiling computer source code into binary code. These tools are typically invoked by a Continuous Integration Server.

- Examples: Maven/Ant/Gradle
- For Oracle Middleware Maven is the preferred component.
- Alternate Names: Domain specific Build Language

**CONTINUOUS INTEGRATION SERVER**
A Continuous integration server continuously automates the process of integrating, building, and testing of code and orchestrating and monitoring externally run jobs.

- Examples : Jenkins /Bamboo/Hudson/Team City
- For Oracle Bamboo is the preferred Continuous Integration Server.
- Alternate Names: Build Server, Build Engine, Continuous Server

## 4.3 Provision and configuration management

**CONFIGURATION MANAGEMENT SERVER**

Configuration management tools allow you to provide standard, consistent provisioning and reliable configuration and management of multiple servers.

- Example: Puppet /Chef/Salt/Ansible/MyST/Enterprise Manager
- For Oracle Either MyST (Fusion Middleware) or Puppet (Database / Weblogic) is the preferred solution
- Alternate Names: Provisioning Management Server / Infrastructure as code.

## 4.4 Deployment

**DEPLOYMENT MANAGEMENT**

Deployment management provides you with a consistent and reliable process for installing software components on your infrastructure. Including the management of the configuration of the different software components.

- Examples: XLDeploy / MyST / Enterprise Manager
- For Oracle Middleware XLDeploy is the preferred solution.
- Alternate names: Deployment engine / install agent.

## 4.5 Testing and Quality

**TEST ENGINE**

The test engine provides you with a structured approach to software testing. It handles the creation, execution and cleanup of your test environment and schedules all systems tests. Including the gathering of the test results. On this level you need to distinct unit tests, component tests, functional tests and non-functional tests.

- Examples: JMeter, Selenium, Fitness, SOAPUI
- For Oracle Middleware SOAPUI is the preferred solution for services and Selenium for User interface components.
- Alternate names: Test tool / test script / regression test .

**QUALITY ENGINE**
The quality engine provides you with a reliable process for monitoring and validating the quality of your system. The quality engine shows the current state of the systems quality and a detailed overview of the historical quality data.

- Examples: Sonar Qube , Codacy
- For Oracle Middleware Sonar Qube is the preferred solution.
- Alternate names: QA engine

## 4.6 Supporting tools

**ISSUE TRACKING**
The issue tracking system provides you with a complete overview of all issues, tasks and bugs that have been registered against the system or component. Most tools have the capability to schedule releases and assign issues to developers

- Examples: Bugzilla, Mantis, Jira
- For Oracle Middleware Jira is the preferred solution.
- Alternate names: Bug system, bug tracker

**WIKI**
The WIKI provides you with a simple way of documenting features, manuals, instructions and ideas. The main feature is that the wiki is focused on collaboration between all team members.

- Examples: Sharepoint, Confluence, Crowdbase wiki
- For Oracle Middleware Confluence is the preferred solution.
- Alternate names: WIKI system

# 5 Your cloud strategy

Moving code “seamlessly” between Cloud & On-Premise.

On the path to production, a solution will be often released dozens of times into each environment. When performed manually, the code build and deployment process is highly error prone and inconsistent, adding both additional cost and additional time to a project. This is further compounded by the fact that in most large organizations there are different individuals and teams performing these tasks in each environment.

To unlock the commercial and speed to market benefits derive the true benefits of developing in the cloud, the process of taking code developed and tested in the cloud and deploying it on-premise needs to be frictionless. The two key prerequisites to achieving this are, environment consistency across Cloud and OnPremise and having a standardized and consistent automated process for building and deploying code across all environments.

# 6 Tools: Choose fit for purpose and maturity

Any tool vendor will offer a complete solution for continuous delivery from their tool suite. However you have to consider the fitness for purpose when you choose a tool. For example: you can use Jenkins as an orchestrator of environment provisioning and deployment of applications. It will work but it is not ideal. Every tool has its own features which have been specifically designed. In addition you have to consider the full field of continuous delivery tooling. Since this is a rather new field. The toolset are rapidly evolving. So an ideal tool would help you perfectly today and can become better (or worse) in the upcoming months.

Focus on using the maximum of standard features of the tools and make sure you haven an exit strategy in place before even starting with a new tool. The field is so rapidly evolving that it is advisable to choose products with a short return on investment. And be prepared to switch tools after a while when you think other products will suit your situation better.

And finally you have to consider the maturity of the tools. Some tools are perfectly for providing continuous delivery solutions in rather strait forward environment. For advanced Oracle Fusion Middleware you need to choose a tool with a certain level of maturity. Otherwise you will be left with a lot of custom scripting and configuration. So the newest tools are not always the best solutions for Oracle middleware. And you also want to engage with tools with a larger number of engineers who know these tools.

# 7 Appendix: how to convince your team to migrate to continuous delivery

Ask them some of these questions and discover how much time is spent on each of them.

- How often are the scripts changed?
- Server Re-start Required – yes/ no
- Time to Restart Server
- Time to Validate (Mins) Error Rate
- Time to Troubleshoot
- Number of FMW Stacks
- Number of environments per stack
- Initial Configuration Changes Timing Considerations - After Basic Provisioning
- Configure OHS to map to OSB/SOA
- Apply JRF, Disable Hostname verification, OWSM Targeting, config OWSM cache
- Configure memory arguments
- Coherence Flag Setup
- Configure JDBC Persistence Store + TLOGS
- Create OPSS schema and reassociate file-based with datatabase
- Datasource connection pool change
- Deploy OSB Custom Xpath
- Configure Authentication provider
- Ongoing Configuration Changes Timing Considerations
- Apply WLS/SOA Patches (has to be applied to every machine)
- Create a new Datasource
- Create new JCA Configuration
- Define Weblogic groups, users, application stripes, grants
- Create a JMS queue
- OHS Failover from Active node to Passive Node
- Apply OHS Patches (to be applied to active and passive OHS binaries for each env)
- Apply Patch Set Updates (e.g. 11.1.1.6 to 11.1.1.7)

All manual tasks in the list above are error prone and possibly a bottleneck in the process.