Docker Driven Continuous Delivery

On the gaps between tooling

Jeroen Peeters

A thesis presented for the degree of
Master of Science

Supervised by:
Professor H. Dekkers

The University of Amsterdam
September 2016

I, Jeroen Peeters, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

tbd.

Acknowledgements

tbd

Table of Contents

Chapter 1

Introduction

tbd.

Chapter 2

Literature review

2.1 Introduction

tbd.

2.2 Tools

2.3 Process

2.4 People

2.5 Methodoloy

Chapter 3

What is Continuous Delivery

In this chapter I will discuss what people generally understand by the term Continuous Delivery.

3.1 Continuous Integration

Continuous Delivery is the natural evolution of Continuous Integration (CI). Practicing Continuous Integration is an absolute necessity before you can start with Continuous Delivery.

CI focuses on integrating different software branches into a main line. This generally occurs when developers make changes to the main line in their development environments.

To do CI one needs at least the following systems:

1. Revision Control System
2. Continuous Integration Server

Figure 3.1 depicts the dependency relationship between the CI systems.

3.1.1 Revision Control System

The RCS covers the integration of code branches into a main line.

3.1.2 Continuous Integration Server

A CI-server, sometimes referred to as the build server, automatically performs the build process when a code branch is integrated into the main line. This ensures that the software in the main line can still be build according to predefined rules. Furthermore it ensures that the change doesn't depend on development specific environments, reducing the 'it builds on my machine'-problem. Preferably the CI-server also executes tests to ensure that previous functionality is still intact.

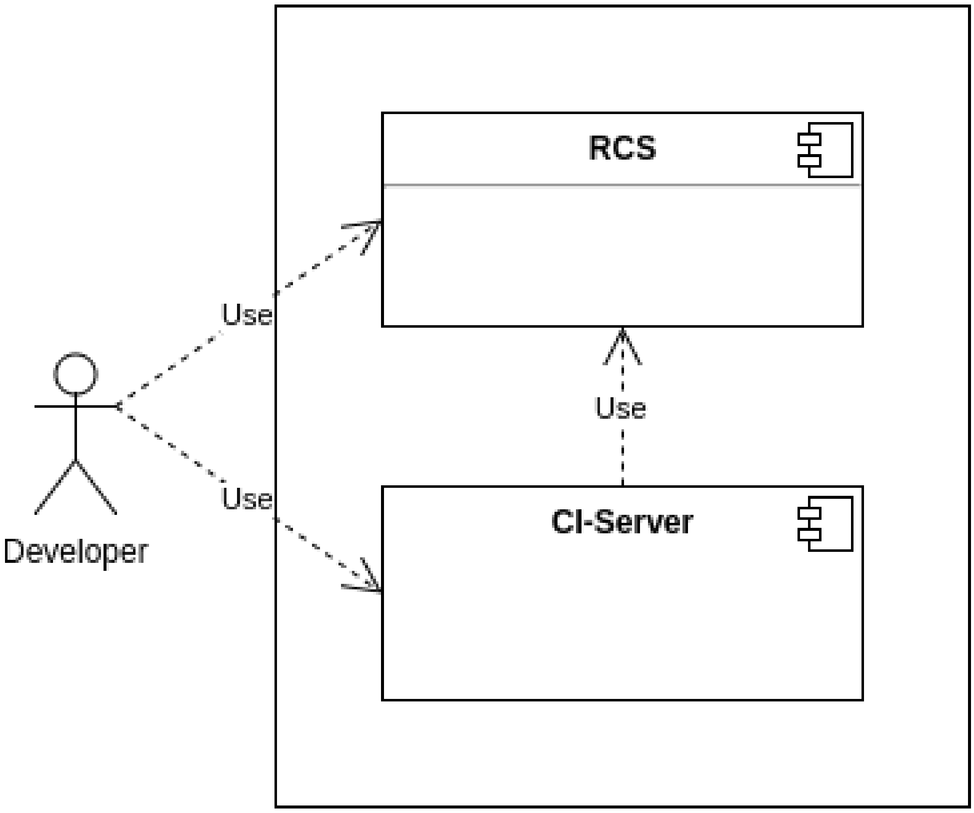

Figure 3.1: Overview Continuous Integration

Chapter 4

The development organization

In order to understand the problems at the organization it is important to have a deeper understanding of the development organization's structure. The organization is a semi-governmental IT project organization who's mission is to help other (semi-)governmental organizations with IT project management and the realization of projects. They lead by example and help the customer to shape their project according to agile principles. In this thesis we are only concerned with the department responsible for software project realization. Within the Software Delivery (SD) department project teams build software in an agile way. Because some customers are still used to work according to a waterfall approach the department plays an important role in guiding customers. The SD project team helps the customer getting familiar with Agile/Scrum principles in order for them to steer and make decisions about importance of tasks. Before a project ends up at SD it usually follows a pre-development process in which some architectural decisions are already made. This is mostly because governments have to apply to standards and regulations. Usually the software realization team is not involved in this process since the team is not yet in existence. This procedure as described here may vary per project and customer, but it usually applies. When the realization team is formed most of the fundamental decisions have already been taken.

To be able to quickly react to customer needs the development organization relies heavily on external hiring for the duration of a project. Within SD all project members are externals. This gives the organization the ability to quickly scale up or down depending on the number of active projects. However, it also implies that knowledge is easily lost. The organization tries to move people between projects as much as possible in order to retain them. In order to move people more easily between projects and bring new people up to speed more quickly the development phase is standardized within the department as much as possible. The standardization is targeted at process, tools and development frameworks and languages. This standardization is something that can change over time and is defined by SD itself. It is possible for a single project to differentiate from the standard following the “comply or explain”-principle.

The standardized process is based on Continuous Integration and Delivery (CI/CD) principles. In the next chapter we will take a closer look at the CI/CD process.

Chapter 5

Stages of Continuous Delivery

In this chapter I describe the different stages of continuous delivery that the development organization went through.

Table 5.1: Demonstration of simple table syntax.

Right Le	ft	Center De	fault
12 12		12	12
123 12	3	123	123
1 1		1	1

5.1 Stage 1: CI in a shared environment

Characteristics

.	
CI/CD Environment	Shared
Maintenance	System Administrator
Deployment	Manual
Flexibility	Static

Systems

Server	Type	Depends on
Subversion	Version Control	
Jenkins	Build Server, CI	Nexus, Sonar, Selenium
Nexus	Artifact Repository	
Sonar	Static Code Analysis	
Selenium		Deployment Server
Deployment Server		Nexus

Tools

Tool	Type	Used by	Depends on
Maven	Build	Dev, Jenkins	
Java	Language, platform	Dev, Jenkins	
Custom quality reporting	Reporting	Jenkins	Sonar, Selenium

Setup

- Setup is done by system administrators

Manual steps

Task	Depends on	Occurrence
Create build job	Jenkins	every new unit of development
Create deployment job	Jenkins	every new unit of development
Configure quality report	Jenkins, Quality reporting	every new unit of development
Maintain server configuration	Deployment Server	on configuration change
Trigger deployment	Deployment Server, Jenkins	on request of tester or stakeholder
Trigger automated tests	Deployment Server, Jenkins, Selenium	every iteration

Problems

Description	Has negative impact on
Resource sharing between all teams	Scalability
Changes and upgrades affect all teams	Stability
Teams can't change setup or install plugins	Flexibility, Usability
Teams can interfere with each other	Stability
Teams depend on sysadmins	Agility
Deployment server changes are difficult to reverse	Flexibility, Scalability
Unable to deploy multiple instances of an application	Agility, Usability

5.2 Stage 2: Automated CD in a distributed environment

Characteristics

.	
CI/CD Environment	Per team
Maintenance	Team
Deployment	Automatic
Flexibility	On-demand

- Each team has his own CI/CD environment
- The team is responsible for the environment (DevOps)
- Dynamic deployment cluster
- Application deployment is scripted
- System administrators maintain the deployment cluster
- Teams decide what their CI/CD landscape looks like

Systems

- Gitlab
- Jenkins
- Nexus
- Sonar
- Selenium
- Deployment server

Tools

- Maven
- Docker
- Custom quality reporting

Manual steps

- s1

Problems

- p1

5.3 Stage 3: — next evolution..

tbd..

5.4 Initial situation

! This needs to be placed elsewhere and rewritten !

Figure 5.2 shows the steps and interactions a developer has with build systems in order to deploy a change in the software to a target server.

Figure 5.3 shows the steps a developer needs to take in order to setup a single source repository and configure the continuous integration pipeline.

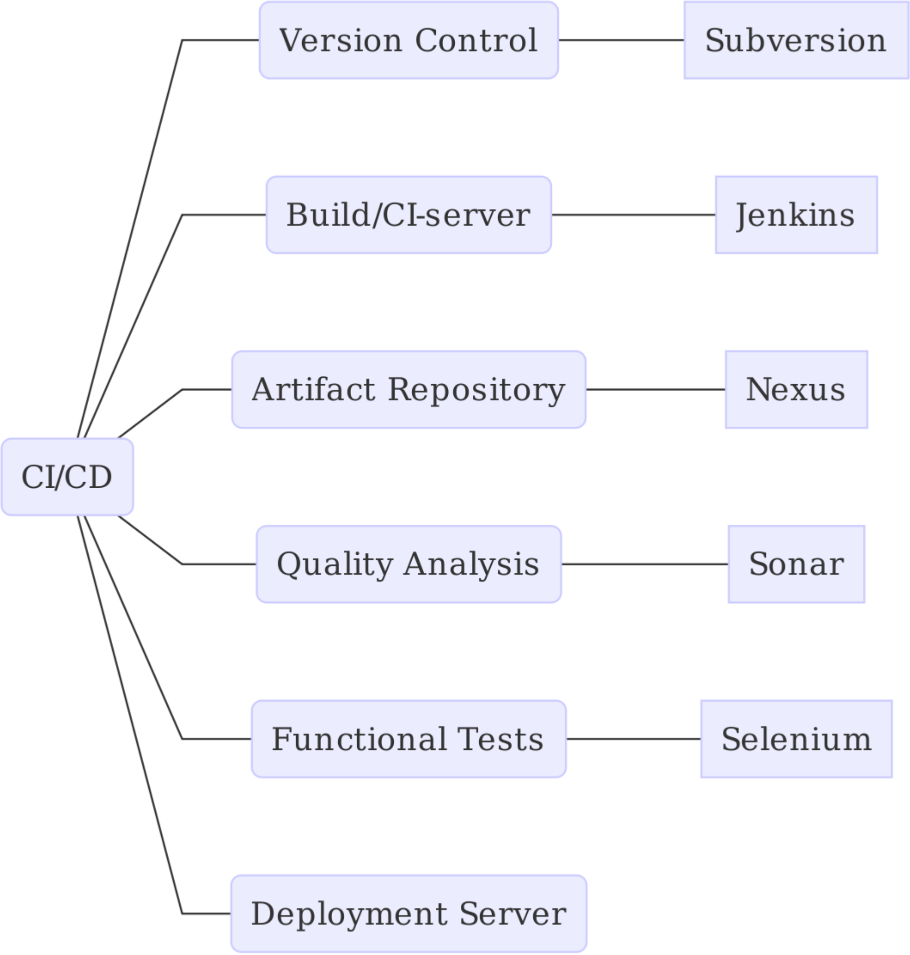

Figure 5.1: CI/CD Schematic Overview

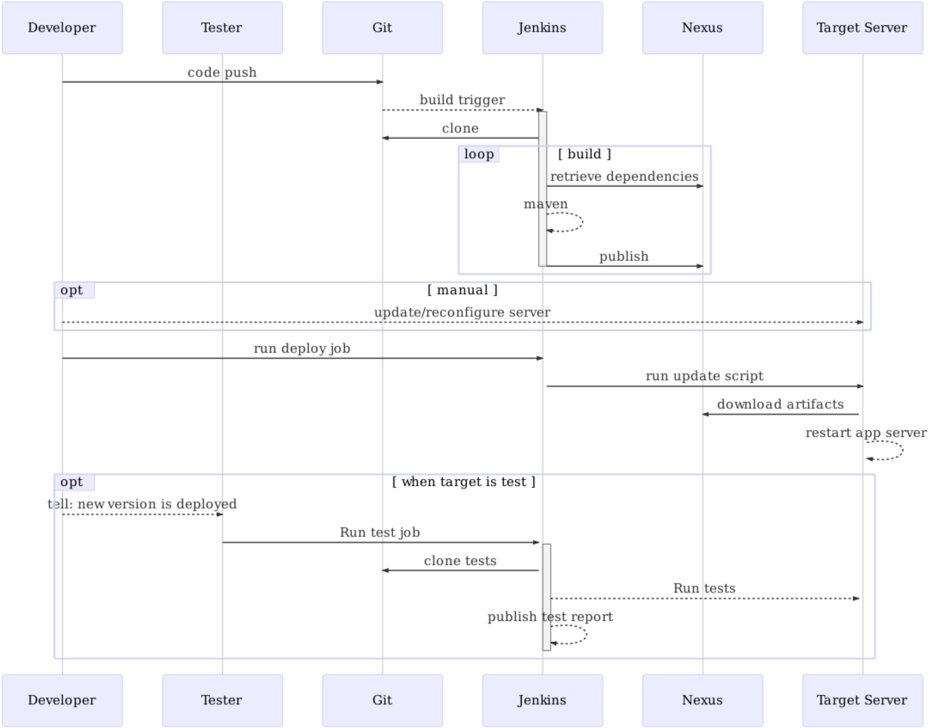

Figure 5.2: Basic CI

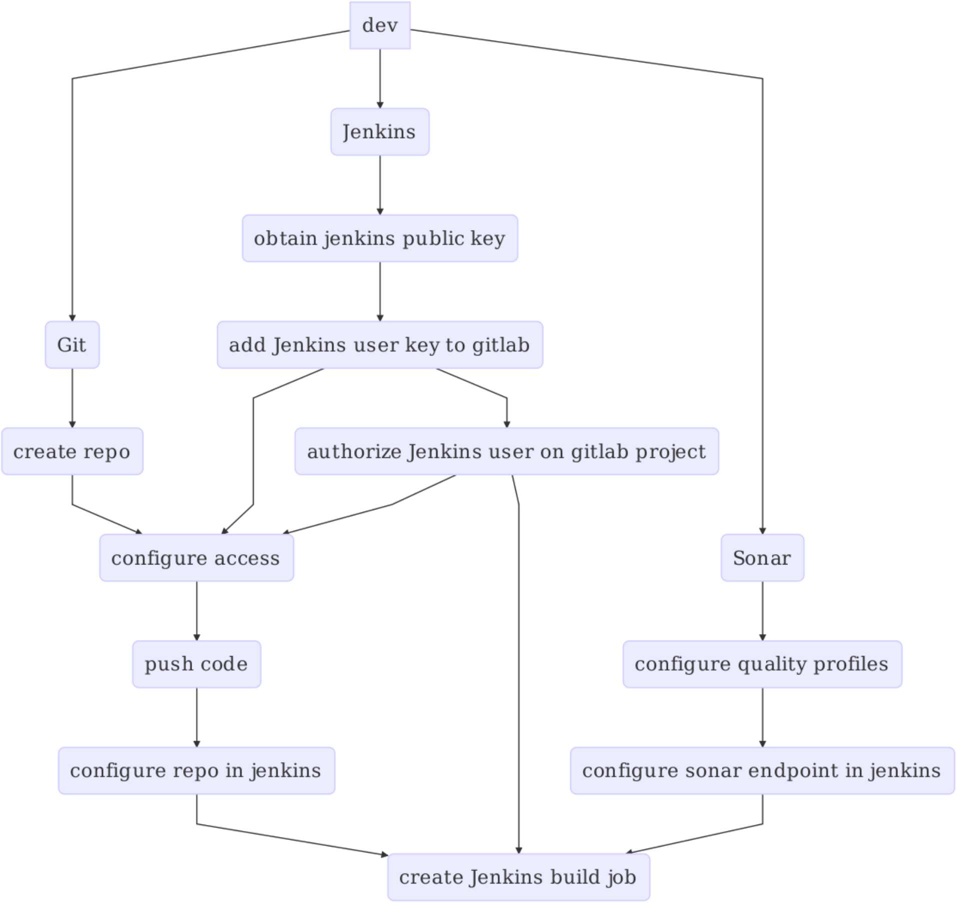
dev
Jenkins
obtain jenkins public key
Git
add Jenkins user key to gitlab
create repo
authorize Jenkins user on gitlab project
configure access
Sonar
push code
configure quality profiles
configure repo in jenkins
configure sonar endpoint in jenkins
create Jenkins build job

Figure 5.3: Basic CI setup

References