Research Continuous Integration & Delivery

Test and delivery your application automatically

Tobias Derksen

Fontys School of Technology
Informatics
Module Software Factory (SOFA)

Venlo, 28th December 2017

Contents

Chapter 1

Introduction & Approach

This document deals with a research about continuous integration and delivery systems made for the SoFa project "Fontys Module Management System" (FMMS).

At first, the requirements on such a system are collected from both the customer who has to use the software and the developers who has to develop the software. The customer is interested in getting a high quality product while the developers don't want to deal with recurring tasks over and over again. Recurring tasks usually can be automated so the developers can focus on developing instead of deployment or testing issues.

After collecting the requirements, some functional demands on the continuous integration system are developed and weighted by its importance. Those demands are than assessed on four different continuous integration and delivery systems. Finally a fitting system is chosen and an infrastructure is recommended.

Chapter 2

What do we need

2.1 How to collect requirements

The requirements are collected by interviewing the developers of the software and the customer. Because of their different roles and interests in this project, they have different expectations on a continuous integration and delivery system. Those differences will end the end sum up to a complete image which actually increases the products quality.

The requirements are split up into continuous integration and continuous delivery requirements. Despite the fact that those systems usually come together, the processes are technically different and they fulfill different business goals. Nevertheless for this research a software is wanted which combines both parts, but for the requirements it makes sense to split them.

Continuous integration usually builds the software and checks the quality by performing some tests, for example unit tests or coverage tests.

Continuous delivery is responsible for bringing the software to the customer. For FMMS this means, that the system should pull the latest version of the docker images, then stops eventually running instances and starts new ones with the new images.

2.2 Requirements for Continuous Integration

2.2.1 Automatically run on repository push

Identifier	**2.2.1**
Title	Automatically run on repository push
Specification	The continuous integration pipeline should automatically run when one or more commits are pushed to the repository.
Rationale	If the system is responsible to start of the integration process, there is no way to prevent or forget to run the tests.
Source	Customer, Developers
Priority	High

2.2.2 Docker Container

Identifier	**2.2.2**
Title	Docker Container
Specification	The continuous integration system should run as a docker container.
Rationale	When the system runs using docker platform independence is inherited by the platform support of docker. Therefor it can be run on multiple operation systems without any changes to the system itself. Furthermore the setup can easily reused for other projects by simpling creating a new instance with the same container image.
Source	Customer, Developers
Priority	High

2.2.3 Support for repository provider

Identifier	**2.2.3**
Title	Support for repository provider
Specification	The continuous integration system should include support for all common versioning software.
Fit criterion	The system includes support for at least the following software: • Git • Subversion A support using plugins is acceptable.
Source	Developers
Priority	High

2.2.4 Work with different build tools

Identifier	**2.2.4**
Title	Work with different build tools
Specification	The continuous integration system should include support for different build tools.
Fit criterion	The system includes support for at least the following build tools: • NodeJS & NPM • Java & Maven • Docker A support using plugins is acceptable.
Source	Developers
Priority	High

2.2.5 Notifications

Identifier	**2.2.5**
Title	Notifications
Specification	The continuous integration system should be able to send notification if something unexpected happens. For example, if a pipeline run failed.
Fit criterion	Notifications should at least be sent via email. Other technologies for example Slack are optional.
Source	Developers
Priority	High

2.2.6 Present quality check results

Identifier	**2.2.6**
Title	Present quality check results
Specification	The continuous integration system should collect and present the results of the quality checks made during the build pipeline run.
Source	Customer, Developers
Priority	Medium

2.2.7 Sonarqube integration

Identifier	**2.2.7**
Title	Sonarqube integration
Specification	The continuous integration system should include support for sonarqube to check code quality and common bugs.
Source	Customer
Priority	Optional

2.2.8 User management with access rights

Identifier	**2.2.8**
Title	User management with access rights
Specification	The continuous integration system should be able to handle multiple user accounts with different access rights. The access rights should be separated by projects or globally valid.
Source	Developers
Priority	Low

2.2.9 Interface should run on multiple operating systems

Identifier	**2.2.9**
Title	Interface should run on multiple operating systems
Specification	The interface of the continuous integration system should run on multiple operating systems. The easiest way to achieve this requirement is to provide an interface which runs inside a browser.
Fit criterion	The interface should be usable on at least the following operating systems: • Windows 10 • MacOS 10.11 or higher • Ubuntu 16.04 or higher
Source	Developers
Priority	High

2.2.10 Prevent check in new version if the tests are unsuccessful

Identifier	**2.2.10**
Title	Prevent check in new version if the tests are unsuccessful
Specification	The continuous integration system should be able to reject builds which do not pass the quality checks. This means that the build results are not archived, the build is marked as a failure and proper notifications will be sent.
Fit criterion	The system should check for at least the following values: • Unit tests • Test code coverage • Code style For each of the checks the system should check for a minimum value which must have met by the code before the build is considered successful.
Source	Customer
Priority	Medium

2.3 Requirements for Continuous Delivery

2.3.1 Support for different levels of stages (production, development, staging)

Identifier	**2.3.1**
Title	Support for different levels of stages (production, development, staging)
Specification	The continuous delivery system should include support to deliver artifacts to different kind of stages. The stages differ in frequency of builds and stability of code and data.
Source	Customer, Developers
Priority	Medium

2.3.2 Docker support

Identifier	**2.3.2**
Title	Docker support
Specification	The continuous delivery system should be able to fetch and deploy docker container.
Source	Developers
Priority	High

2.3.3 Work with CI

Identifier	**2.3.3**
Title	Work with CI
Specification	The continuous delivery system should work together with the CI system. This means, that after the CI system finishes a build successfully, the continuous delivery system should be able to automatically start the delivery process of the application.
Source	Developers
Priority	High

2.3.4 Produce defined environment

Identifier	**2.3.4**
Title	Produce defined environment
Specification	The continuous delivery system should be able to produce a deterministic and defined environment. This means, that the software is - after delivery - in predefined state. Therefore normally the database will be reseted and staging data will be loaded.
Rationale	To get comparable results in integration and acceptance tests, the software must be in the same state before starting the tests. If the responsibility lies in the continuous delivery system, no errors can be made by a user and all tests are comparable by default.
Source	Customer
Priority	Medium

Chapter 3

Research

3.1 Functional Requirements

Before the research can start, some functional requirements are defined to compare the different continuous integration and continuous delivery systems. Those requirements are than weighted by its importance, the higher the weight the more important is the functionality. Crucial requirements score 10 on weight.

Name	**Requirement**	**Weight**
VCS commit hooks	2.2.1	10
Docker support	2.2.2, 2.3.2	10
Git support	2.2.3	10
SVN support	2.2.3	8
NPM support	2.2.4	10
Maven support	2.2.4	10
E-Mail notifications	2.2.5	7
Check style integration	2.2.6	8
Junit support	2.2.6	9
Code coverage analysis	2.2.6, 2.2.10	7
Threshold for test coverage	2.2.10	5
Sonarqube support	2.2.7	3
User Management	2.2.8	3
Multi OS	2.2.9	7
Support for different stages	2.3.1, 2.3.4	6

Table 3.1: Functional requirements

3.2 Research different Options

3.2.1 Jenkins

Jenkins is one of the most famous continuous integration systems. It has been forked from Hudson, a famous continuous integration system developed at the Eclipse Foundation. Last year the development of Hudson stopped in favor of Jenkins. Jenkins is developed using the MIT license and is free to use for everyone.

Jenkins uses an highly extensible plugin system to provide all kind of functionality. There are thousands of Jenkins plugins available inside the community which shows that Jenkins is used very often.

Regardless that Jenkins provides Docker support, if Jenkins itself runs as a docker container the creation of docker containers during the builds are quite complex. During this research it turns out, that you need to install docker tools inside the container and mount the docker socket into the appropriate location. After that Jenkins will be able to actually build docker containers. There is no official Jenkins image available which includes docker tools and other required dependencies, so that is something which has to be developed for this project.

Additionally Jenkins integrates with Github, providing Web Hook to listen on commits and pull requests.

	Weight	**Jenkins**
VCS commit hooks	10	Yes
Docker support	10	Yes
Git support	10	Yes
SVN support	8	Yes
NPM support	10	Yes
Maven support	10	Yes
E-Mail notifications	7	Yes
Check style integration	8	Yes
Junit support	9	Yes
Code coverage analysis	7	Yes
Threshold for test coverage	5	Yes
Sonarqube support	3	Yes
User Management	3	Yes
Multi OS	7	Yes
Support for different stages	6	Yes
Sum	113	113

Table 3.2: Results of Jenkins research

3.2.2 Travis CI

Travis CI is a hosted continuous integration and delivery service which is used to build and test projects hosted on GitHub. It integrates seamless with GitHub and provides a good-looking interface which includes all important information. Despite the fact that Travis CI is technically a free software, the manufacturer provides no installation guide which makes it practically impossible to setup the software on a local infrastructure. But nevertheless Travis CI is for free for public GitHub projects, which implies to this project.

	Weight	Travis CI
VCS commit hooks	10	Yes
Docker support	10	Yes
Git support	10	Yes
SVN support	8	Yes
NPM support	10	Yes
Maven support	10	Yes
E-Mail notifications	7	Yes
Check style integration	8	Yes
Junit support	9	Yes
Code coverage analysis	7	No
Threshold for test coverage	5	No
Sonarqube support	3	No
User Management	3	No
Multi OS	7	Yes
Support for different stages	6	No
Sum	113	89

Table 3.3: Results of Travis CI research

3.2.3 TeamCity

TeamCity is a continuous integration and delivery system developed by the czech company JetBrains, well-known for their development environments. TeamCity can run as web container inside a Java EE server or as a docker container.

TeamCity includes out of the box docker support, nevertheless when TeamCity runs as a docker container, it is not able to build docker containers on the local host. This of course is a cutback in functionality. Furthermore TeamCity does not provide NPM support. NPM is a critical requirement, because the Angular framework used in frontend relies on NPM.

	Weight	TeamCity
VCS commit hooks	10	Yes
Docker support	10	Partly
Git support	10	Yes
SVN support	8	Yes
NPM support	10	No
Maven support	10	Yes
E-Mail notifications	7	Yes
Check style integration	8	Yes
Junit support	9	Yes
Code coverage analysis	7	No
Threshold for test coverage	5	No
Sonarqube support	3	No
User Management	3	Yes
Multi OS	7	Yes
Support for different stages	6	Yes
Sum	113	83

Table 3.4: Results of TeamCity research

3.2.4 Drone

Drone is a quite new continuous integration and delivery system which relies heavily on docker. Drone itself runs as a docker container and can spawn multiple workers using docker, even on remote systems. The use of docker makes Drone independent from the running operating system, so it only relies on a working docker daemon. Furthermore, docker provides feature to scale up application which makes Drone appropriate for high availability or high performance systems.

	Weight	**Drone**
VCS commit hooks	10	Yes
Docker support	10	Yes
Git support	10	Yes
SVN support	8	Yes
NPM support	10	Yes
Maven support	10	Yes
E-Mail notifications	7	Yes
Check style integration	8	Yes
Junit support	9	Yes
Code coverage analysis	7	No
Threshold for test coverage	5	No
Sonarqube support	3	No
User Management	3	No
Multi OS	7	Yes
Support for different stages	6	No
Sum	113	82

Table 3.5: Results of Drone research

3.3 Summary

The research shows, that Jenkins can fulfill all requirements. This is mainly due to its plugin system and the massive support from the community. Nearly every functionality can be integrated into Jenkins using plugins and a lot of big software companies and service providers maintain plugins to integrate Jenkins with their software or service.

	Weight	**Jenkins**	**Travis CI**	**TeamCity**	**Drone**
VCS commit hooks	10	Yes	Yes	Yes	Yes
Docker support	10	Yes	Yes	Partly	Yes
Git support	10	Yes	Yes	Yes	Yes
SVN support	8	Yes	Yes	Yes	Yes
NPM support	10	Yes	Yes	No	Yes
Maven support	10	Yes	Yes	Yes	Yes
E-Mail notifications	7	Yes	Yes	Yes	Yes
Check style integration	8	Yes	Yes	Yes	Yes
Junit support	9	Yes	Yes	Yes	Yes
Code coverage analysis	7	Yes	No	No	No
Threshold for test coverage	5	Yes	No	No	No
Sonarqube support	3	Yes	No	No	No
User Management	3	Yes	No	Yes	No
Multi OS	7	Yes	Yes	Yes	Yes
Support for different stages	6	Yes	No	Yes	No
Sum	113	113	89	83	82

Table 3.6: Summary of research results

Chapter 4

Recommendation

The recommended architecture consists of four docker containers and two external services.

All internal services will run as docker container. This improves the server's stability and security. Because the container can only access the data inside the container, if the application is hacked, they can only access the data of this particular services and nothing crucial from the operating system.

The code is hosted on GitHub and will be accessed by Jenkins, therefore Jenkins needs a appropriate access key. The setup furthermore contains a docker registry service which stores docker image so they can be pulled from other hosts.

During the build, Jenkins runs the docker commands to actually build the project parts (backend and frontend). After a successful build, the container are pushed to the registry. At this point the continuous integration parts is over. To delivery the software, Jenkins executes commands on the target host which pull the new images from the registry, stops the running services and starts them with the newer images.

The database is running on the fontysvenlo server as a service on its own. For this project the database is considered an external services, because it is managed by the server administrator.

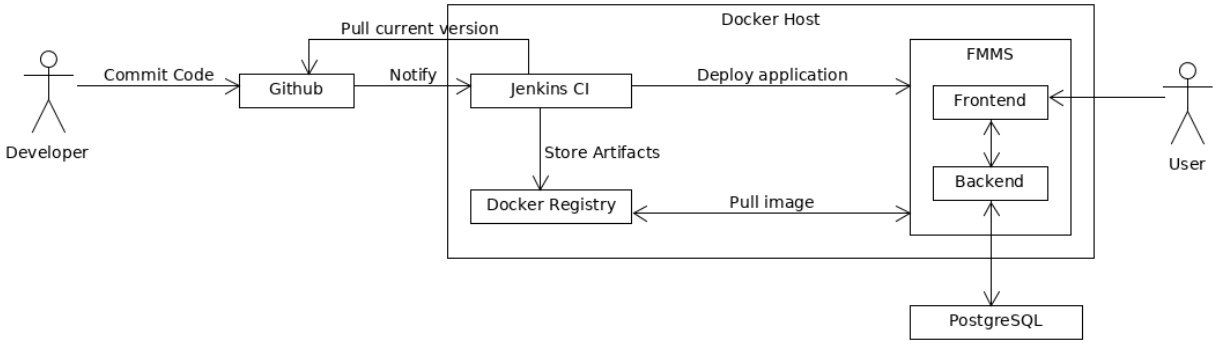

Figure 4.1: Recommended Architecture

Acronyms

CI Continuous Integration

CD Continuous Delivery

JSON Javascript Object Notation

API Application Programming Interface

References

Drone (2018). *Drone Continuous Delivery*. URL: http://docs.drone.io/ (visited on 14th January 2018).

Jenkins (2018). *Jenkins User Documentation*. URL: https://jenkins.io/doc/ (visited on 14th January 2018).

Jetbrains (2018). *TeamCity Documentation*. URL: https://confluence.jetbrains.com/display/TCD10/TeamCity+Documentation (visited on 14th January 2018).

TravisCI (2018). *Travis CI User Documentation*. URL: https://docs.travis-ci.com/ (visited on 14th January 2018).

Wikipedia (2018a). *Continuous delivery*. URL: https://en.wikipedia.org/wiki/Continuous_delivery (visited on 14th January 2018).

– (2018b). *Continuous integration*. URL: https://en.wikipedia.org/wiki/Continuous_integration (visited on 14th January 2018).