CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Research Question

1.2 Overview

In the following the structure of the thesis is outlined. Every chapter is briefly discussed, what it is about.

In the second chapter we will walk through the foundations. The chapter mentions technologies, which are used for the thesis and gives references. Furthermore it gives references to the practices which are used and are crucial for the thesis. The references are properly selected, to understand the details if they are not known and to understand what the technologies and techniques are used for. In summary those are kubernetes, continuous delivery, continuous deployment and techniques from infrastructure as code and site reliability engineering.

The third chapter is conceptual macro view to the technique nonfunction production regression testing. The text walks through the general environment and discusses the most important concepts and how they communicate with each other. The most important steps of the continous delivery pipeline are discussed and it explains how the pipeline is extended in order to have the technique of nonfunctional production regression testing. The text argues how the methodologies of nonfunctional production are embedded in the pipeline and how the pipeline must be extended.

In the fourth chapter we will get to the concrete implementation of the nonfunctional production regression testing. The chapter will go into the details of how the concept is implemented. Concretely the software deployer is described, which was implemented in the context of this thesis.

Chapter five is about the evaluation of the new approach. We will investigate the use of the technique and customized software in two different companies. The first company is Gapfish, a four year old

startup, and the software department of DIN, a company which is established for a hundred years. We are going to evaluate positive outcomes, still problematic concerns and their improvements. Another part of the evaluation is the comparison to other techniques which other companies and groups developed and tested. We differentiate in their features, advantages and disadvantages.

In the last chapter, the conclusion, the whole thesis is summarized and all the chapters are resumed. Important is the second part of the conclusion, in which we have an outlook to further improvements and how the technique can be extended to have further upgrades to delivery pipelines.

CHAPTER 2

FOUNDATIONS

CHAPTER 3

GENERAL APPROACH

Nonfunctional Production Regression Testing is the topic of the masterthesis and we will discuss the core of it in this chapter. The designation itself describes precisely what the technique and what the new practice is about. Therefore we will go shorty through every single term in the following.

Nonfunctional refers to the metrics, which we're evaluating in the test; These metrics are only non-functional and as a consequence generically applicable to multiple applications. The term production refers to the environment. The metrics, which are collected are collected in the production environment. And finally the term regression refering to the testing strategy. The metrics will be compared between two different versions and the latter version is tested for a regression, concretely a decline of the monitored metrics.

The testing technique provides some further features, which are not included in the designation. Indeed the testing technique is completely automatable and you can continuously apply it to the new versions. The testing technique is designed in respect to failing as fast as possible and inform developers.

When it comes to automation and continuous, the thoughtful reader is now probably reminded of continuous integration, delivery and deployment. This testing techniques evolved naturally from those practices and extends those. Those already established practices support the developers until the software is deployed. In contrast to that, nonfunctional production regression testing, supports the developers during after the deploy and while the software runs in production.

To understand the testing procedure in a whole and completely, it is necessary to show a complete overview of the whole testing and production environment. We will go through the steps of the pipeline and discuss it in a nutshell as you can see in the figures.

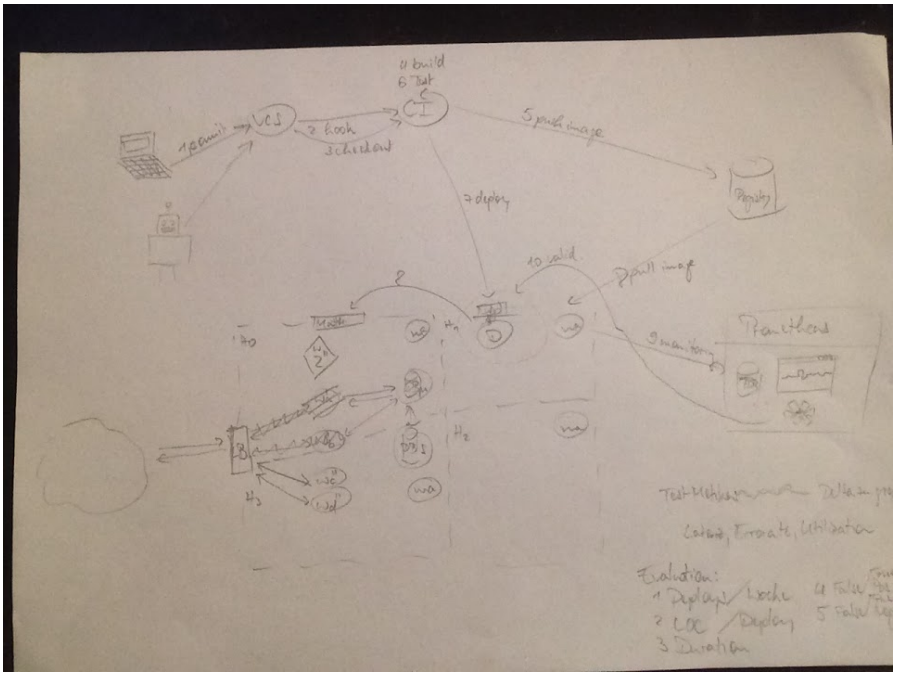

Figure 3.1 Nonfunction Production Regression Testing Overview.

The steps in the very beginning are known from the established practices continuous delivery and deployment respectively. But it is necessary to touch them and integrate them to the whole picture and outline the special characteristics for the new testing technique.

At the very beginning, there is a developer, who changed the code locally on his working machine. There is also a version control system. And in the first step the developer commits the code to the version control system. The main focus is on git. Subversion is possible, too though. In the following description we will stick to the terms and notions of git.

Next there is a continuous integration system. After the commit happened, the second step is a message to the continuous integration system. The message holds the information that a new commit exists and the continuous integration server clones the code from the version control system and checks out the specified version. Now the continuous integration system has three major jobs. The first one is to start a build process, the second is to run the tests and the third is to give the deploy signal.

In step four, namely running the build, it is typical to compile binaries, render assets and further artifacts. For our purposes it is especially necessary to build at least one or multiple docker images.

The import thing about this is, that we need to identify every docker image to a specific build. Therefore we use the commit hash, the version created by the version control system. This commit hash will follow us through the whole pipeline. This is important to be able to trace every step in the pipeline for a specific version. With this thought in mind, the docker image is tagged with the commit hash of this version and the name of the branch. Along the way it is mentioned, that the branch name is not absolutely necessary to definitely determine the version. The branch name is included for better readability for a developer and approximately recognize what the image version is about.

The continuous integration server then pushes the ready build docker image to an image registry, such as docker hub. Nevertheless this can be a private registry as well. This registry serves later as an artifact repository.

The second continuous integration step, or in total step six, the continuous integration server runs the tests. There can be multiple stages, such as unit, feature or smoke tests. Yet we do net need to recall all the details here.

The last step of the continuous integration system is to send a deploy signal. In the shown figure this is step seven. When you look at the tests, the result could on the one hand be a failure or on the other hand be successful. If the test have failed, the remaining pipeline will be cancelled and the developer will be informed. Just as you know it from a typical continuous integration system. If the tests have been successful and accordingly the build including all test stages have been successful, the continuous integration system sends the signal to deploy.

It makes sense to deploy only specific versions and not every commit. The practice which is pretty common, is that you develop new features in a seperate branch. For those version it is common to not send a deploy signal even though the branch build and tests are successful. Usually after there has been a review and a decision to deploy the changes to production, even though it is a very small change. But when the decision is made and merged into a specified branch, for instance the master branch, this version will go to production.

However, just to clarify, each built image for every single version is sent, independently of successful tests and independently of the intention to go to production, to the image repository. The reason could be a staging system and even running the tests inside the build image. But this is just a side note.

So the deploy signal is given when two requirements are fulfilled: the build and tests are successful and it is a version which is planned to go to production.

Until this point, as it was already mentioned, it is just a usual continuous delivery or deployment pipeline, which is commonly used in the development process. But since nonfunctional production regression testing is a technique, which is supposed to be completely automated, such a prior described delivery pipeline including automated deployments is precondition. From now on it is becoming interesting, hence the testing technique supports the developers post deployment in production instead of the old practices before the deploy.

The next unit is the deployer. It is the software, which is particularly implemented for this masterthesis. In the next chapter the deployer is described in detail. This chapter demonstrates how the deployer is embedded in the pipeline or in other words in the environment amongst all other tools. In the meantime tools exist, which have a similar purpose. It is crucial to have full control over the whole deployment process and as a consequence it was necessary to implement the software and have it customizable.

CHAPTER 4

IMPLEMENTATION OF THE APPROACH

CHAPTER 5

EVALUATION

5.1 Adoption of the approach

5.2 Comparison of to other approaches

CHAPTER 6

CONCLUSION

6.1 Resume

6.2 Outlook and future work