Implementation of an Enterprise Service Bus with OpenShift and Camel

Ing. Thomas Herzog B.Sc

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Software Engineering

in Hagenberg

im Juni 2018

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, June 1, 2018

Ing. Thomas Herzog B.Sc

Contents

Preface

Abstract

This should be a 1-page (maximum) summary of your work in English.

Chapter 1

Introduction

1.1 Motivation

Large enterprises work with several independent applications, where each application covers an aspect of a business of an enterprise. In general, these applications are from different vendors, implemented in different programming languages and with their own life cycle management. To provide a business value to the enterprise, these applications are connected via a network and part of a business workflow. The applications have to interexchange data, which is commonly represented in different data formats and versions. This leads to an highly heterogeneous network of applications, which is very hard to maintain.

The major challenge of an IT department is the integration of independent applications into the enterprise application environment. The concept of Enterprise Application Integration (EAI) provides patterns [Hohpe and Woolf 2008], which help to define a process for the integration of applications into a heterogeneous enterprise application environment. One of these patterns is the Enterprise Service Bus (ESB), which is widely used in the industry.

Often the term ESB application is used to refer to an ESB, which connects multiple applications. But an ESB is a software architectural model, rather than an application. The term could have been established by the usage of middleware such as JBoss Fuse [Red Hat Inc. 2018a], which helps to integrate applications into an ESB. JBoss Fuse is based on the JBoss Enterprise Application Platform (JBoss EAP), where the applications are integrated in a existing runtime environment.

With the upcoming of cloud solutions such as Platform as a Service (PaaS) [Devi and Ganesan 2015, p. 2-3] it is now possible to move the platform from a dedicated environment to a cloud environment, where the cloud provider will take over responsibility for maintenance and security of the platform. The concept of Integration Platform as a Service (IPaaS) [Liu et al. 2015, p. 3] is on top of PaaS and enhances a common PaaS solution with the Integration features needed by EAI.

Thus, enterprises can now reduce the effort in implementing an ESB, integrating applications into the ESB and reducing the costs of of an ESB by using a consumption based pricing model.

1.2 Objectives

This thesis aims to implement an ESB on Openshift PaaS [Red Hat Inc. 2018b]. Commonly an ESB is implemented with the help of middleware such as JBoss Fuse, which is based on the JBoss EAP. The concepts of PaaS and IPaaS in combination with an ESB are in general new to the industry. In general the industry hosts their ESBs in their own data centers, due to the lack of trust for cloud solutions.

A main focus of this thesis is how applications internal and external can be integrated and managed in the PaaS solution Openshift. The different aspects of applications integrated into an ESB such as security, staging and versioning of will also be discussed in this thesis. The security is commonly managed by a middleware, which is in general hosted on a dedicated data center owned by the enterprise. With the usage of PaaS such as Openshift the responsibility of the security mostly shifts to the cloud provider.

// TODO: Futher writing

Chapter 2

Infrastructure as Code (IaC)

Infrastructure as Code has become very important with the upcoming of Linux containers and cloud solutions. A developer describes the intended state of the infrastructure via e.g. text files and a tool interprets the description and creates an instance of the infrastructure. With IaC there is no need for manually configuring infrastructures via Graphical User Interfaces (GUI) or Shell anymore. Infrastructures as Code is testable, reproducible and can be vendor independent, because of the abstraction level provided by the description language.

2.1 The Need for IaC

With IaC, the IT moves from the iron age, to the cloud age [Kief Morris 2016, p. 4]. The iron age of the IT, was the time where IT systems were bound to the physical hardware, and a system setup or change was a long term and complex process. In the cloud age of the IT, systems are decoupled from the physical hardware and sometimes even from the underlying operating system. Therefore the setup and change of systems are less complex and can be achieved fast.

The current speed of technological evolution does not support long term processes of infrastructure setup or maintenance anymore. Enterprises with legacy systems see themselves outrun by technological nimble competitors, who can manage infrastructures more efficiently. An Infrastructure setup and the maintenance of the infrastructure in the cloud age has to be simple, trustworthy and fast, otherwise the infrastructures can not cope with the fast changing requirements of the industry.

IaC provides a separation of the infrastructure definition from the underlying operating system and hardware, which in turn allows the automation of the setup and change management of the infrastructure. This can lead to an almost fully automated infrastructure, where changes are applied automatically without the need of a person who has to oversea the process. A common problem which prevents infrastructures to become fully automated is the so called *Automation Fear Spiral* which is shown in figure 2.1.

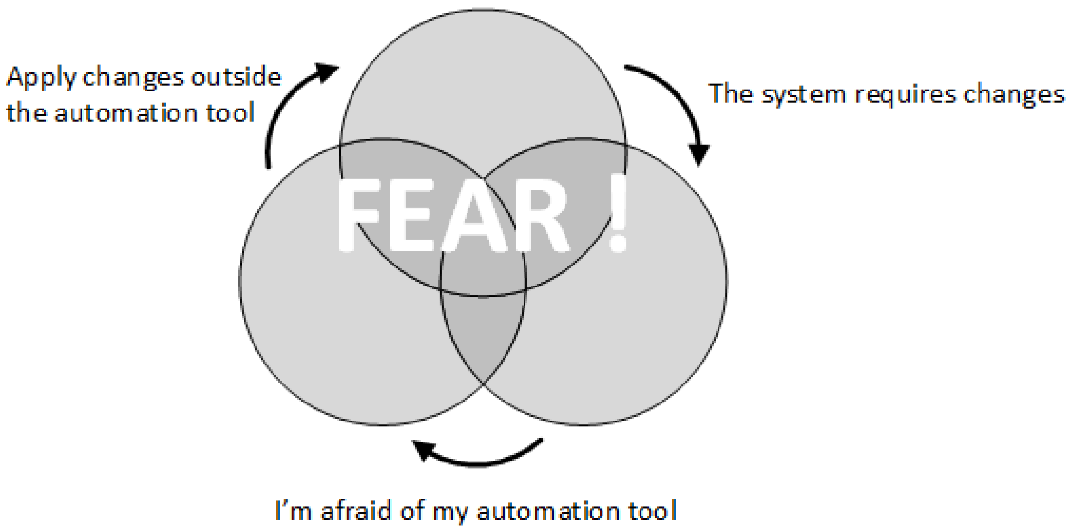
Apply changes outside the automation tool
The system requires changes
FEAR !
I'm afraid of my automation tool

Figure 2.1: Automation Fear Spiral

Because of the missing trust to the automation process and its underlying tooling, changes are applied to the infrastructure manually and outside the defined process. Thus, the infrastructure can become inconsistent, because the changes maybe haven't been added to infrastructure description or change set. If the infrastructure is tried to be reproduced, changes may be missing, because they haven't been added, or are not working because they haven't been tested.

Therefore, teams have to break this spiral to get the automated infrastructure. Additionally, changes which are defined in the infrastructure definitions or separate change sets can be managed by Version Control System (VCS), tested and are therefore consistent and less error prone.

Cloud based solutions make heavy usage of IaC, because the cloud providers can not allow the developers to tamper with the underlying system. With the definition language, the cloud providers provide an abstraction of the underlying system and an easy way to define the infrastructure for the specific cloud solution. The developers only have to describe the infrastructure and its changes, the cloud providers are responsible to create an instance of the infrastructure and to apply changes to it. Therefore, the developers can spend more time on the definition of the infrastructure their applications need.

2.2 Principles of IaC

References

Literature

Devi, T. and R. Ganesan (2015). *Platform-as-a-Service (PaaS): Model and Security Issues.* School of Computing Science and Engineering, VIT University (cit. on p. 1).

Hohpe, Gregor and Bobby Woolf (2008). *Enterprise Integration Patterns.* 11th ed. Boston, MA: Pearson Education, Inc. (cit. on p. 1).

Kief Morris (2016). *Infrastructure as Code: Managing Servers in the Cloud.* 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc. (cit. on p. 3).

Liu, Lawrence et al. (2015). *Integration Platform as a Service: The next generation of ESB, Part 1.* IBM (cit. on p. 1).

Online sources

Red Hat Inc. (2018a). *Red Hat JBoss Fuse.* URL: https://developers.redhat.com/products/fuse/overview/ (visited on 02/22/2018) (cit. on p. 1).

— (2018b). *Red Hat Openshift Container Platform.* URL: https://www.openshift.com/container-platform/features.html (visited on 02/22/2018) (cit. on p. 2).