students@fh-ooe

Proposal

Master of Science

Implementation of an enterprise service bus (ESB) with OpenShift und Camel

by

Ing. Thomas Herzog B.Sc

Supervisor (FH):	DI (FH) Peter Kulczycki
Supervisor (Gepardec):	Dr. Erhard Siegl
External company	Gepardec
Handed in:	December 8, 2017

1. Statement of the problem

Large enterprises work with several independent applications and systems which have the need to interexchange data. The data is represented in several formats and versions, which increases the complexity of the communication between the applications and systems. Another challenges are the configuration management and the maintenance of the integrations of the applications and systems to each other, which affects the whole development life cycle and release management.

The concepts of DevOps, PaaS, IaaS and microservice architecture differ to the classic development life cycle, release management and a server infrastructure where DevOps brings development and IT operations together and the classic server infrastructure will be replaced by platform as a service (PaaS) or infrastructure as a service (IaaS) or more likely both.

An enterprise service bus (ESB) is an enterprise pattern which in a large enterprise is often used to connect applications and systems together. There are several frameworks out there such as JBoss Fuse [1], which is based on a jboss application server, which helps to design implement and run an ESB. Openshift is a open source cloud system where Gepardec thinks could be used to realize an ESB or at least several aspect of it. The idea is to split a monolithic ESB application into microservices and host them in an openshift cloud, where openshift provides features for monitoring, deploying, scaling and securing microservices.

The following questions shall be answered in the master thesis:

1. How can components, internal and external microservices be integrated in an openshift environment with as little configuration as possible ?
2. How can different versions and stages of microservices be managed in an openshift environment ?
3. How to secure microservices hosted in an openshift environment ?

Along with using an cloud environment such as openshift changes will have to be made in the development life cycle, release management and interaction between the departments and service providers, because the classic approaches will not apply anymore when microservices are hosted in an cloud environment such as openshift.

[1] `https://developers.redhat.com/products/fuse/overview/`

2. State of the art

In my opinion today there are two established architectures for software. On the one hand there is the monolithic architecture, where the whole software is managed and organized in a single source repository, with one single complex build and deployment, and on the other hand there is the very popular micro service architecture, where the software components are organized, build, test and deployed independently from each other and therefore can be managed better.

The upcoming of cloud solutions such as Openshift have become very popular these days as they simplify life cycle of an infrastructure the services run on, because cloud solutions such as Openshift abstract us from the actual infrastructure, what is a main factor when using Platform as a Service (PaaS) solutions. Features such as load balancing, triggered deployments, isolated application processes, integrated security and templates allow developers to define an infrastructure by one or more templates which when instantiated represent our infrastrucutre hosted in Openshift.

The good integration of Continuous Integration/Deployment via Jenkins build server in Openshift allows to use Openshift as the platform to build, test and host services, where either a Jenkins Pipeline can create, trigger and test an Openshift build and/or deployment, or Openshift gets triggered by an hook and starts an build and/or deployment itself.

This redefines the infrastructure developers are used to build, test and host their services on. There is not only an application server anymore, there is also build and deploy mechanisms, a load balancer, integrated security and monitoring. In a common development, release and application life cycle, there is the build and deploy separated from the service load balancing, which is separated from the monitoring. There is a very good integration of several aspects of an development, release and application life cycle in Openshift, where the whole build, deploy and how to host the services in the infrastructure can be defined via templates and is therefore reproducible.

Openshift should allow the integration of an ESB application, where Openshift takes part of several aspects of the development, release and application life cycle and the ESB application will have to consider the micro service architecture approach in its software architecture to be able to be well integrated in Openshift.

3. Deduction

It will be analyzed how an ESB application can be designed with a micro service architecture and how this ESB application can be integrated in Openshift, where maybe some aspects of an ESB are implemented with Openshift mechanisms. The development and release life cycle can be fully integrated in Openshift where a single build server environment will become obsolete, but nevertheless Jenkins is fully integrated in Openshift an can be used to configure an Openshift cluster.

This strong integration of the development and release life cycle in Openshift will require a changes in the management of the source code, the build definitions, tests and the release management as well. The separation of an ESB application into isolated services with their own development and release life cycle will bring the possibility of partial releases, but will also increase the complexity when it comes to the interaction between the services.

While the management of an ESB application, which is separated into isolated services, increases in complexity, the complexity of the build, release and necessary infrastructure of the service will become less complex, because of the available features in cloud solutions such as Openshift. Instead of spending time with a hard to maintain and hard to evolve infrastructure more time can be spent for management and organization of the development and release life cycle which becomes more important and complex when there are several independent services involved.

Another major aspect is security and how secrets are accessed, where Openshift contains a secret management where secrets of different types such as *ssh-auth, basic-auth or string-data* can be managed and injected into the Docker containers in several ways to provide the secret for a process. With Openshift the management of the security can be completely separated from the application development and referenced secrets can be represented by placeholders (e.g. environment variables) which get injected by Openshift during container start. During development time, no developer needs access to any secret anymore.

4. Prototype

An ESB application will be implemented and integrated in Openshift. The application will not only be hosted in Openshift, the application will also be build and released via Openshift mechanisms.

The actual functionality will not be the focus of the prototype, the focus will be on the integration of the organization of the source, configuration, build and release of the ESB application in the cloud environment Openshift. The prototype shall be capable of answer the in section 1 raised questions and shall prove that an ESB can be well integrated in the cloud environment Openshift and that not only the application itself can be hosted in an cloud environment such as Openshift but also it's development and release lifecycle can be integrated in Openshift.

5. Empric

TODO

6. Induction

TODO

A. Target audience

The target audience of the master thesis will be especially software developers who are working on enterprise software, microservices or software which could be hosted in an cloud environment such as Openshift. Also cloud architects, software product managers and CTO's are part of the target audience because the master thesis could help or inspire them to integrate their software applications in a cloud environment.

Therefore that a key factor of DevOps is the joint work of the infrastructure department and the development department, the master thesis could also be interesting for persons in charge of hosting infrastructure such as service providers like IT&Tel, because the master thesis could help them to understand the developer point of view when an ESB application is hosted in Openshift.

Parts of the master thesis could help people to understand the interconnection between containerization (Docker), CaaS (Kubernetes) and PaaS and why there is a need for such technologies these days.

B. Structuring

This sections deals with the structuring and the extend of the master thesis.

1. Abstract ***(1 page)***
2. Introduction ***(2 pages)***
    2.1. Motivation
    2.2. Objectives
3. Infrastructure as Code (IaC) ***(~7 pages)***
    3.1. The need for IaC
    3.2. Core concepts
4. Containerization with Docker ***(~7 pages)***
    4.1. The need for containerization
    4.2. Core concepts
    4.3. IaC with Docker Compose
    4.4. Virtualization vs Containerization

5. Container as a Service (CaaS) with Kubernetes ***(~7 pages)***
    5.1. Need for container orchestration
    5.2. Core concepts
    5.3. IaC with kubernetes templates
    5.4. VM orchestration vs container orchestration
6. PaaS with Openshift ***(~7 pages)***
    6.1. Need for application orchestration
    6.2. Core concepts
    6.3. IaC with openshift templates
    6.4. Development and release lifecycle integration
    6.5. Secret and configuration management
    6.6. Application orchestration vs container orchestration
7. Enterprise Service Bus (ESB) ***(~5 pages)***
    7.1. Need for an ESB
    7.2. Core concepts
    7.3. Challenges of hosting an ESB
8. Prototype ESB in Openshift ***(~10 pages)***
    8.1. Source code management
    8.2. Configuration management
    8.3. Build and release management
    8.4. ESB Infrastructure in Openshift
9. Conclusion ***(1 page)***
10. Prospects ***(1 page)***

C. Milestones

This section deals with the planning of the work on the master thesis which is organized by milestones.

To-Date	Task	Milestone
01/20/18	Learn the ropes of docker	CI/CD in openshift
	Learn the ropes of openshift	
	Learn the ropes of CI/CD in openshift	
01/31/18	Specify application build artifacts	Abstract build/release concept
	Specify application integration in Openshift	
	Specify application release in Openshift	
02/15/18	Study/write IaC	Chapter 'IaC'
02/28/18	Study/write Docker	Chapter 'Docker'
03/15/18	Study/write Kubernetes	Chapter 'CaaS - Kubernetes'
03/31/18	Study/write Openshift	Chapter 'PaaS - Openshift'
04/15/18	Study/write ESB	Chapter 'ESB'
05/15/18	Implement ESB servcies	Prototype Implementation
	Implement build for services	
	Implement release/rollback for services	
06/01/18	Write Prototype	Chapter 'Prototype'

D. Literature index

This section deals with the literature used to create the master thesis.

1. *DevOps with Openshift*
   Authors: Stefano Picozzi, Mike Hepburn & Noel OĆonnor
   ISBN-13: 978-1491975961

2. *Infrastructure as Code: Managing Servers in the Cloud*
   Author: Kief Morris
   ISBN-13: 978-1491924358

3. *Building Microservices: Designing Fine-Grained Systems*
   Author: Sam Newman
   ISBN-13: 978-1491950357

4. *Enterprise Integration Patterns*
   Author: Gregor Hohpe & Bobby Woolf
   ISBN-13: 978-0321200686

5. *Management of Integration Architecture Using Switchyard As an example* Authors: Christoph Gostner, B.Sc
   Master thesis, FH-Hagenberg - Software engineering, 2013

6. *https://docs.openshift.com/container-platform/3.5/welcome/index.html*
   Author: RedHat
   OpenShift Online Container Platform 3.5 Documentation

7. *https://docs.docker.com/*
   Author: Docker
   Docker Online Documentation

8. *https://martinfowler.com*
   Author: Martin Fowler
   Martin Flowers online Blog