# Red Hat Microservices

# Red Hat Microservices

# Chapter 1. General

# Chapter 2. Introduction

The design, development, packaging and monitoring of a project architected around the Microservices Model implies that we embrace new concepts, patterns and development practices.

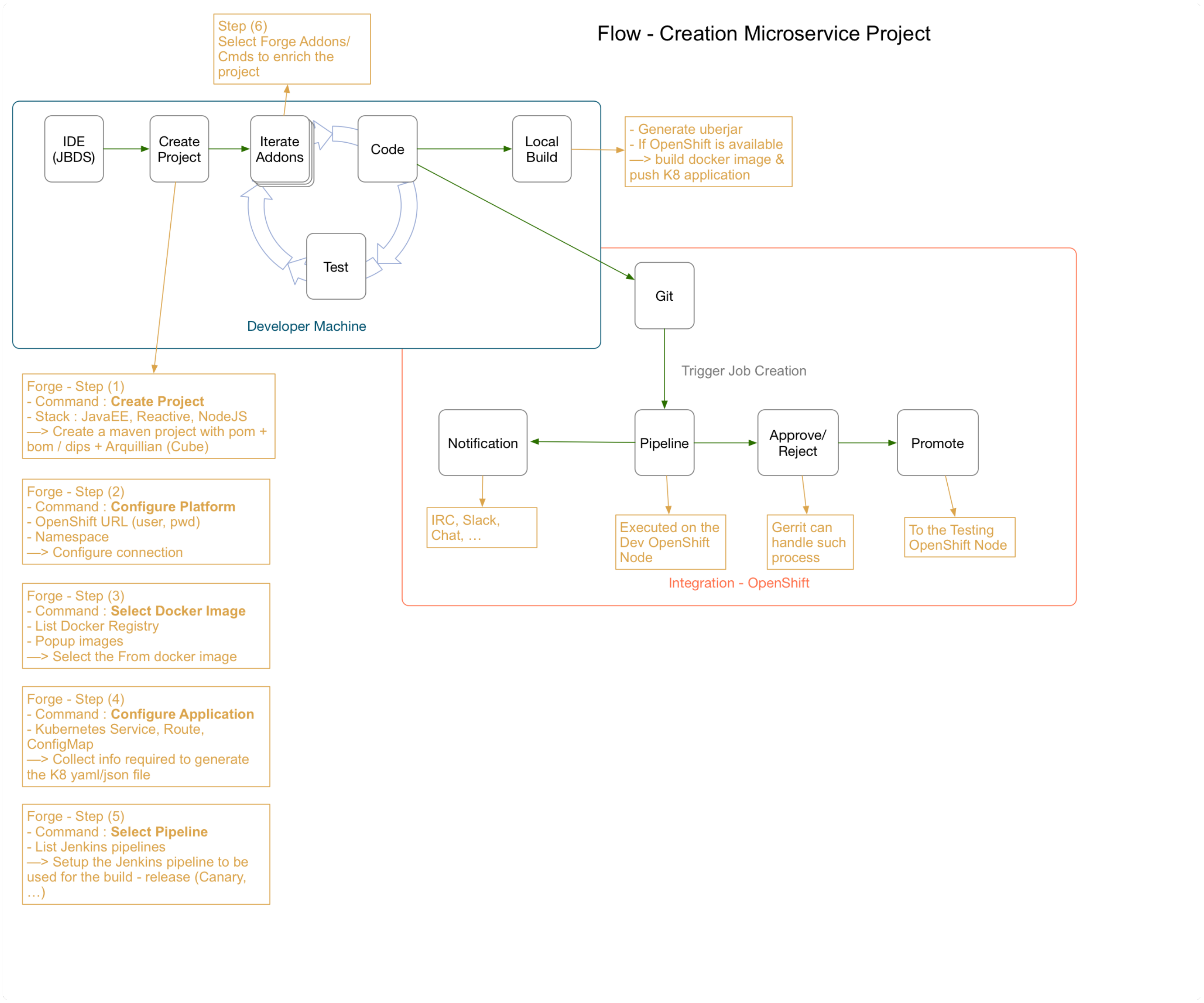
Through this documentation we will explain how you can implement the required methodology to develop Microservices, deploy them in a Cloud Native Platform and speed your delivery processes by adopting a Continuous Integration and Deployment approach.

To be continued …

# Chapter 3. Fundamental

## 3.1. Introduction

The big picture : to be explained in detail


Flow - Creation Microservice Project
Step (6) Select Forge Addons/ Cmds to enrich the project
IDE (JBDS)
Create Project
Iterate Addons
Code
Local Build
Test
Developer Machine
- Generate uberjar - If OpenShift is available —> build docker image & push K8 application
Git
Trigger Job Creation
Notification
Pipeline
Approve/ Reject
Promote
IRC, Slack, Chat, …
Executed on the Dev OpenShift Node
Gerrit can handle such process
To the Testing OpenShift Node
Integration - OpenShift
Forge - Step (1) - Command : Create Project - Stack : JavaEE, Reactive, NodeJS —> Create a maven project with pom + bom / dips + Arquillian (Cube)
Forge - Step (2) - Command : Configure Platform - OpenShift URL (user, pwd) - Namespace —> Configure connection
Forge - Step (3) - Command : Select Docker Image - List Docker Registry - Popup images —> Select the From docker image
Forge - Step (4) - Command : Configure Application - Kubernetes Service, Route, ConfigMap —> Collect info required to generate the K8 yaml/json file
Forge - Step (5) - Command : Select Pipeline - List Jenkins pipelines —> Setup the Jenkins pipeline to be used for the build - release (Canary, …)

## 3.2. Decompose

From monolith to Microservices

## 3.3. Cloud Native Platform

## 3.4. Continuous Integration & Delivery

## 3.5. Actors

#### 3.5.1. Developer

#### 3.5.2. Architect

#### 3.5.3. Builder

#### 3.5.4. Operations

# Chapter 4. Your First Cloud Application

# Chapter 5. Developer Guide

## 5.1. Design and Patterns selection

- Circuit Breaker
- Api Gateway
- Load Balancing & resilience

## 5.2. Tooling & IDE

- Apache Maven
- JBoss Forge + addons ?
- Microservices Forge commands
- OpenShift & Kubernetes Client
- Minishift
- JBDS + Addons ?

## 5.3. Containers

- WildFly Swarm
- SpringBoot
- Vert.x

## 5.4. Reactive Programming

- TODO

## 5.5. Testing

- Arquillian
- Arquillian Cube (docker)

## 5.6. Packaging

- Docker image(s)
- Kubernetes application
- Service & Routes

# Chapter 6. Builder Guide

## 6.1. Infrastructure setup

- Gogs/GitLab
- Jenkins
- Gerrit
- Taiga
- Letchat
- Slack

## 6.2. CI/CD

- Canary Release
- Jenkins Pipelines

# Chapter 7. Operations Guide

- OpenShift Nodes
- Docker Registry Setup
- SSO & Security config
- Load Balancing & HA
- Service Discovery
- Logging & Distributed tracing
- Metrics & performances
- Api Gateway
- Data Centers Topology

# Chapter 8. Vocabulary

## 8.1. Terms

- Microservices :
- Paas :
- Physical :
- Virtualized :
- Cloud :

## 8.2. Technical Glossary

| Name | Description | Links |
|---|---|---|
| Archaius (NetflixOSS) | dynamic, multi dimensional, properties framework | https://github.com/Netflix/archaius, http://techblog.netflix.com/2012/06/annoucing-archaius-dynamic-properties.html |
| Cloud Native Computing Foundation (CNCF) | Started by the Linux Foundation. "The Foundation's mission is to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self healing multi-tenant nodes." | https://cncf.io/ |
| NetflixOSS | Netflix Open Source Software Center. Many of the projects Netflix has released are used in microservices projects. Pivotal leverages them heavily. | Netflix Open Source Software Center |
|  | Prometheus | Monitoring and alerting toolkit built by SoundCloud and open sourced. Member of the CNCF. |

| Name | Description | Links |
|---|---|---|
| https://prometheus.io/ | Ribbon (NetflixOSS) | Ribbon is a Inter Process Communication (remote procedure calls) library with built in software load balancers. The primary usage model involves REST calls with various serialization scheme support. |
| GitHub - Netflix/ribbon: Ribbon is a Inter Process Communication (remote procedure calls) library with built in software… | Spring Boot | Packages Spring applications as an Uberjar. |
| Spring Boot | Uberjar | An application packaged as a jar file that contains all dependencies even dependencies like the servlet engine. |
|  | Uberjar | A jar file with all dependencies included inside |
|  | WildFly Swarm Upstream project based on WildFly. | Packages Java EE applications as an Uberjar. |
| Rightsize your JavaEE Applications | WildFly Swarm | Zipkin |
| distributed tracing system | http://zipkin.io/ | Zuul (NetflixOSS) |

# Chapter 9. Patterns

## 9.1. Monolith Architecture

TODO

- Description
- Problem
- Solution
- Example

## 9.2. Microservices Architecture

TODO

- Description
- Problem
- Solution
- Example

## 9.3. Configuration

TODO

- Description
- Problem
- Solution
- Example

## 9.4. Service Discovery

TODO

- Description
- Problem
- Solution
- Example

## 9.5. Api Gateway

TODO

- Description
- Problem
- Solution
- Example

## 9.6. Circuit Breaker

TODO

- Description
- Problem
- Solution
- Example

## 9.7. Rolling upgrade

TODO

- Description
- Problem
- Solution
- Example

## 9.8. Distributed Tracing & Logging

TODO

- Description
- Problem
- Solution
- Example

## 9.9. Instrumentation

TODO

- Description
- Problem
- Solution
- Example

## 9.10. high Availability & Loadbalancing

TODO

- Description
- Problem
- Solution
- Example

# Chapter 10. Concepts

## 10.1. Software Development Model

Basically there are 2 main paths to chose from, either you start from scratch or you decide to decompose an existing application into a set of microservices and to reuse existing. The process to create from scratch a new application is named **greenfield** while the other is **brownfield**.

As opposed to the Green Field path, we are not starting from scratch, rather we are going to focus on keeping our previous investment, and wish the team to stay focused on the same language rather than learning a new one. Again here we will start with Java EE, basing our example on an existing Java EE application that we wish to decompose, while preserving its functionality intact, keep the code untouched and allow the app to benefit from sampler footprint and its services to evolve independently.

*Reference*

- Brownfield - wiki
- Brownfield - Webopedia
- Greenfield - wiki
- Greenfield - Webopedia

## 10.2. 12-factor Apps

https://12factor.net/