Migration einer Anwendung von Jakarta EE zu Quarkus

Enterprise going Cloud Native

Sebastian Hempel · 14.09.2023

Sebastian Hempel

IT CONSULTING HEMPEL

- selbständiger Software-Entwickler seit 2003
- Java, Puppet
- Linux und OpenSource

https://it-hempel.de

@sepp@chaos.social

Eine Reise in die Cloud

Ausgangslage

Was hatten wir?

Hans - Pixabay - CC0

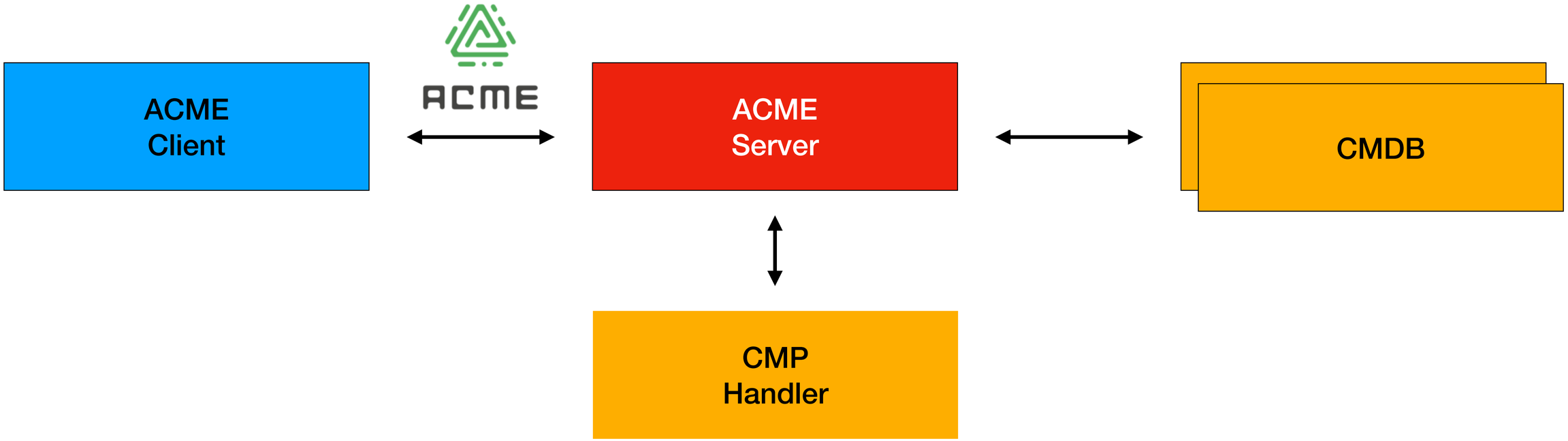
Ausgangslage

Bestehende Anwendung

- Service zur Anforderung von Zertifikaten
- ACME Server Implementierung
- erweiterte Prüfung des Certificate Signing Request (CSR)
- on premises

Ausgangslage

Infrastruktur (2018)



Ausgangslage

Technologie (2017 / 2018)

- Jakarta EE Anwendung (EAR)
- JBoss EAP Application Server (JBoss EAP 7.2, Java EE 8)
- JDK 8
- Maven Build (multi module)

JAKARTA® EE

Jakarta EE

genutzte Spezifikationen

- CDI 2.0 (Weld)
- JAX-RS 2.1 (RESTEasy)
- JPA 2.2 (Hibernate)
- JDBC
- JTA

Ausgangslage

weitere Bibliotheken

- YAML (SnakeYAML)
- BouncyCastle
- DeltaSpike (Scheduler)
- Zugriff auf Konfigurationsdatei
- Zugriff auf CredentialStore

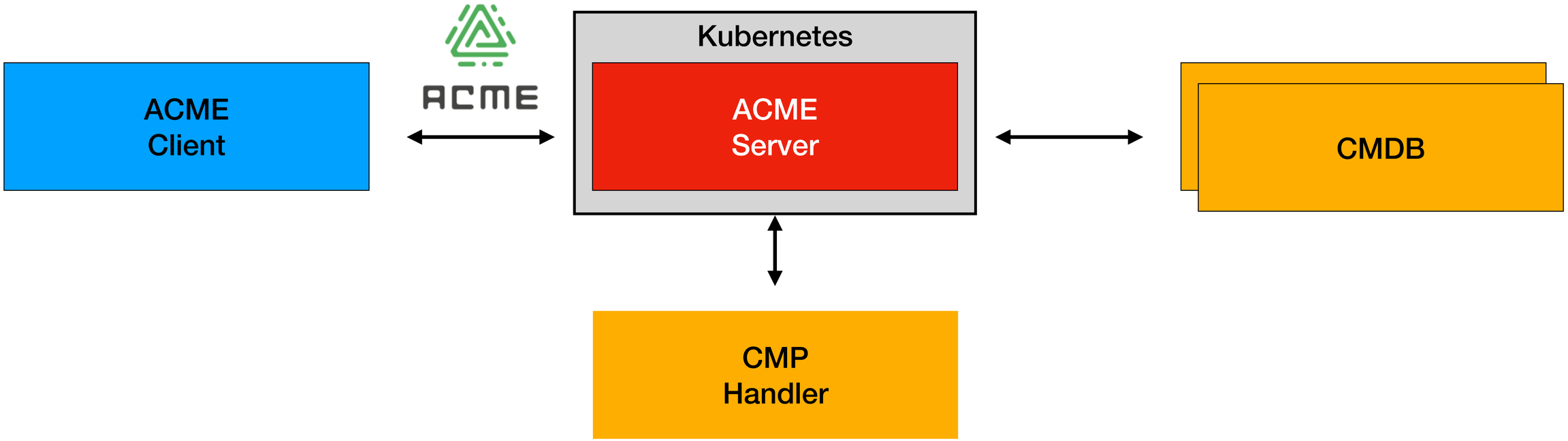
Anforderung

Let’s got to the Cloud

buttercupelles - Pixabay - CC0

Zielarchitektur

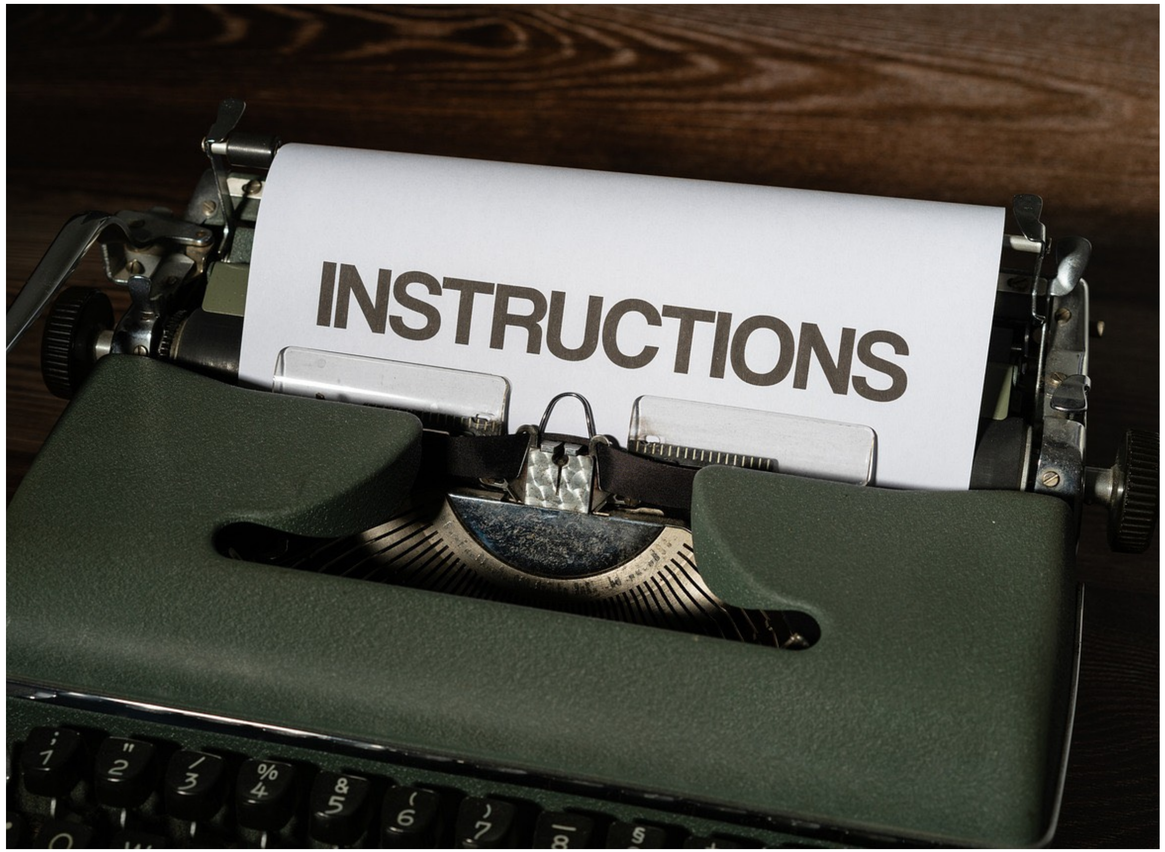
Kubernetes (2020)



Anforderungen

2020

- kein Deployment von JBoss in Kubernetes
- kein Re-Write des Codes
- (Anfangs) rückwärtskompatibel zu JBoss



viarami - Pixabay - CC0

Moderne Java Frameworks

Kandidaten

- Spring Boot (2014)
- WildFly Swarm / Thorntail (2015 - 2020)
- Micronaut (2018)
- Helidon (2018)
- Quarkus (2019)

Quarkus

Supersonic Subatomic Java

QUARKUS

Quarkus was created to enable Java developers to create applications for a modern, cloud-native world. Quarkus is a Kubernetes-native Java framework tailored for GraalVM and HotSpot, crafted from best-of-breed Java libraries and standards.

https://quarkus.io/about/

Was ist Quarkus?

- Framework basierend auf den Jakarta EE Spezifikationen
- Laufzeitumgebung und Applikation in einem JAR
- Maven Plugin für Build-Time Optimization (no reflection) Aktionen werden zur Build-Zeit „vorverlegt“
- Plugins mit Support für weitere Technologien (Kafka, S3 Storage, …)
- Entwickler Modus

Warum Quarkus?

- keine Umgewöhnung für Entwickler
- Unterstützung aller verwendeten JSRs
- Nutzung der gleichen Implementierungen wie JBoss
- „Support“ von RedHat
- Developer Joy

Wandlung zur Quarkus

Notwendige Anpassungen

Alexas_Fotos - Pixabay - CC0

Wandlung zur Quarkus

Nutzung des Maven-Plugins

```
<plugins>
    <plugin>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-maven-plugin</artifactId>
        <extensions>true</extensions>
        <executions>
            <execution>
                <goals>
                    <goal>build</goal>
                    <goal>generate-code</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
</plugins>
```

Wandlung zu Quarkus

Dependencies - Umstellung der pom

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>7.0</version>
        </dependency>
    </dependencies>
</dependencyManagement>
```

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-bom</artifactId>
            <version>1.7.1.Final</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-jsonb</artifactId>
    </dependency>
    <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarks-undertow</artifactId>
    </dependency>
    ...
</dependencies>
```

Wandlung zu Quarkus

Nutzung von JBoss Logging

```
<dependencies>
    <dependency>
        <groupId>org.jboss.logging</groupId>
        <artifactId>commons-logging-jboss-logging</artifactId>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
    </dependency>
    <dependency>
        <groupId>org.jboss.logmanager</groupId>
        <artifactId>log4j2-jboss-logmanager</artifactId>
    </dependency>
</dependencies>
```

Wandlung zu Quarkus

No Reflection

- „build-time“ CDI
- Alle benötigten Beans müssen annotiert werden

```
@Dependent
public class ThymeleafFactory {
}
```

Wandlung zu Quarkus

Datasources - andere Annotation

```
@Inject
private final EntityManager em;
```

Wandlung zu Quarkus

Datasources - application.properties statt persistence.xml

```
quarkus.datasource.db-kind=mssql
quarkus.datasource.username=SA
quarkus.datasource.password=sqlserver2019!
quarkus.datasource.jdbc.url=jdbc:sqlserver://localhost:1433;databaseName=ACME
quarkus.datasource.jdbc.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
```

Nutzen von Quarkus

Wo wir uns doch umstellen mussten.

congerdesign - Pixabay - CC0

Besonderheiten

CDI Injection

- Annotation kann wegfallen und trotzdem wird injiziert
- Quarkus möchte package private field injection

Besonderheiten

CDI Injection

```
@ApplicationScoped
public class CounterBean {

    private CounterService service;

    @ConfigProperty(name = "service.logging")
    boolean enableLog;

    CounterBean(CounterService service) {
      this.service = service;
    }

}
```

Besonderheiten

Build-Time und Run-Time Konfiguration

- Es kann nicht alles zur Run-Time konfiguriert werden.
- Gewisse Einstellungen werden nur zur Build-Time durchgeführt / beachtet
- JDBC-Treiber zur build-time

Quarkus

Was wir nicht mehr missen möchten.

moviephile - Pixabay - CC0

Quarkus Highlights

Developer Joy

- Live Coding - sehr schnelle Change Zyklen
- Dev Services - Dependencies werden hochgefahren (Testcontainers)

Quarkus Highlights

Quarkus-Test

- Tests gegen / in der laufenden Applikation
- Injection von Mocks

Quarkus Highlights

Dev-Services

- „Integration“ von Test-Containers
- Hochfahren von Mocks (WireMock)
- Injizierung der geänderten Konfiguration (Port)
- Maven + Docker = vollständige (Test-)Umgebung

qimono - Pixabay - CC0

Quarkus Highlights

Test-Profiles

- unterschiedliche Szenarien
- z.B. JPA und S3
- Alle Tests eines Profiles werden nacheinander abgearbeitet = kein Neustart

qimono - Pixabay - CC0

Quarkus Highlights

MicroProfile Config

- Konfiguration wie man es möchte
- verschiedene (hierarchische) Quellen
- Konvention für Konfigurationsdatei
- Komplexe Objekte

Quarkus Highlights

Plugin für JSON Logging

- Plugin aus dem Quarkiverse
- Erweiterung des Log Records
- Anpassung der Felder pro Log-Message möglich
- Log4j2 Unterstützung für Context

```
<dependency>
    <groupId>io.quarkiverse.loggingjson</groupId>
    <artifactId>quarkus-logging-json</artifactId>
    <version>1.1.1</version>
</dependency>
```

```
quarkus.log.json.console.enable=true
quarkus.log.json.log-format=default
quarkus.log.json.additional-
field.serviceName.value=acme-server
quarkus.log.json.additional-field.stage.value=dev
```

Fazit

Wir würden es wieder tun.

carloscuellito87 - Pixabay - CC0

Was machte Probleme?

- Maven Profil für Quarkus und JBoss
- Quarkus mit Handbremse
- wurde bald aufgegeben gegenüber einen JBoss Fork

OpenClipart-Vectors - Pixabay - CC0

Was lief gut?

schnelle Migration

- Migration konnte innerhalb von Tagen umgesetzt werden
- Kein Anpassung am eigentlichen Code (nur Annotationen)

OpenClipart-Vectors - Pixabay - CC0

Was lief gut?

Developer Joy

- Sofortige Akzeptanz durch Nutzung von Dev-Mode & Co
- „Hunger“ nach weiteren Features von Quarkus
- Sehr flache Lernkurve für Jakarta EE Entwickler

OpenClipart-Vectors - Pixabay - CC0

Was lief gut?

schnelle Nutzung von neuen Features

- Features konnten sofort genutzt werden
- Plugins haben Integration in neue Infrastruktur (S3, Kafka) erleichtert

OpenClipart-Vectors - Pixabay - CC0


Quarkus - Jakarta EE in der Cloud

Bildnachweis

- Eine Reise in die Cloud: Crysis Rubel - A Journey by Cloud - CC BY 2.0
- Jakarta EE für die Cloud: Vicuna R - Über den Wolken… - CC BY-SA 2.0