
Vergleich eines Usecases mit Serverless Technologie gegenüber Spring Boot Technologie am Beispiel von Instant Payments

Silas Hoffmann

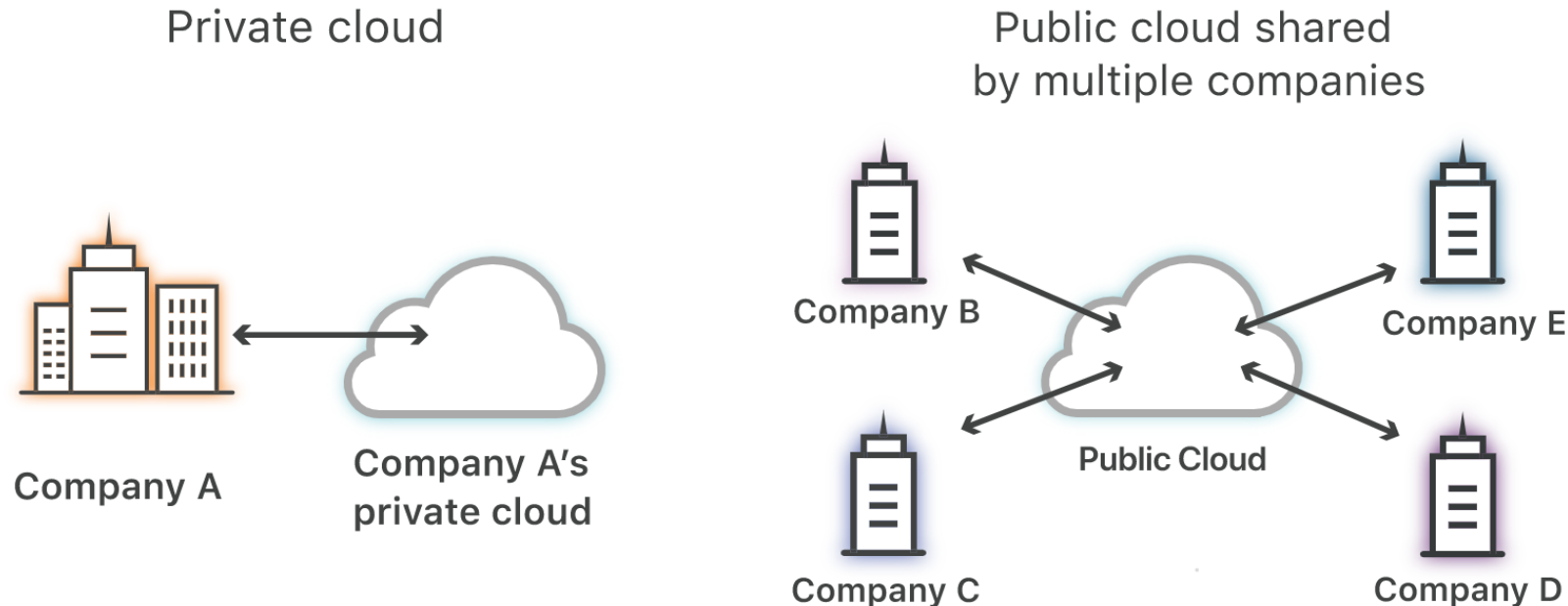
12 September 2021

Inhalt

- Zielsetzung
- Begriffserklärung
 - JBoss
 - Serverless
- Vergleich: Serverless / Application Server
- Implementierung Prototyp
- Lasttest
 - Durchführung
 - Analyse / Fazit
- Optimierungspotenzial

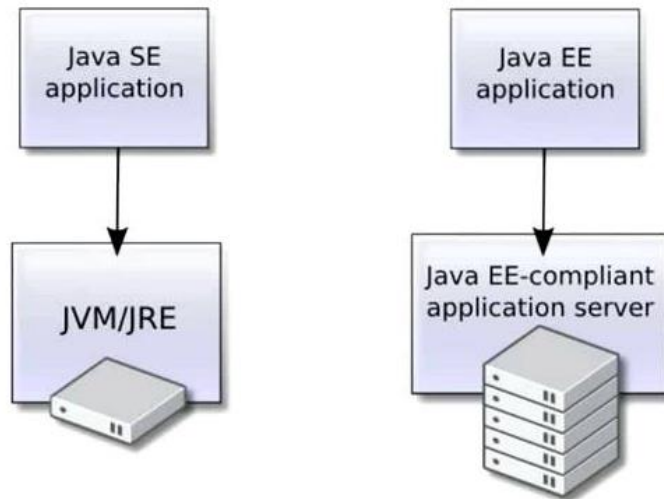
Zielsetzung

- Definierten Anwendungsfalls ermitteln (Instant-Payments)
- Cloudfähigkeit von Spring Boot und Serverless Tech. Vergleichen
- Container Startupzeiten / Verarbeitungsgeschwindigkeiten evaluieren
 - Startupzeiten müssen absolut minimal sein um fachliche Timeout bei Instant-Payments zu vermeiden (End-to-End max. 7 Sekunden)



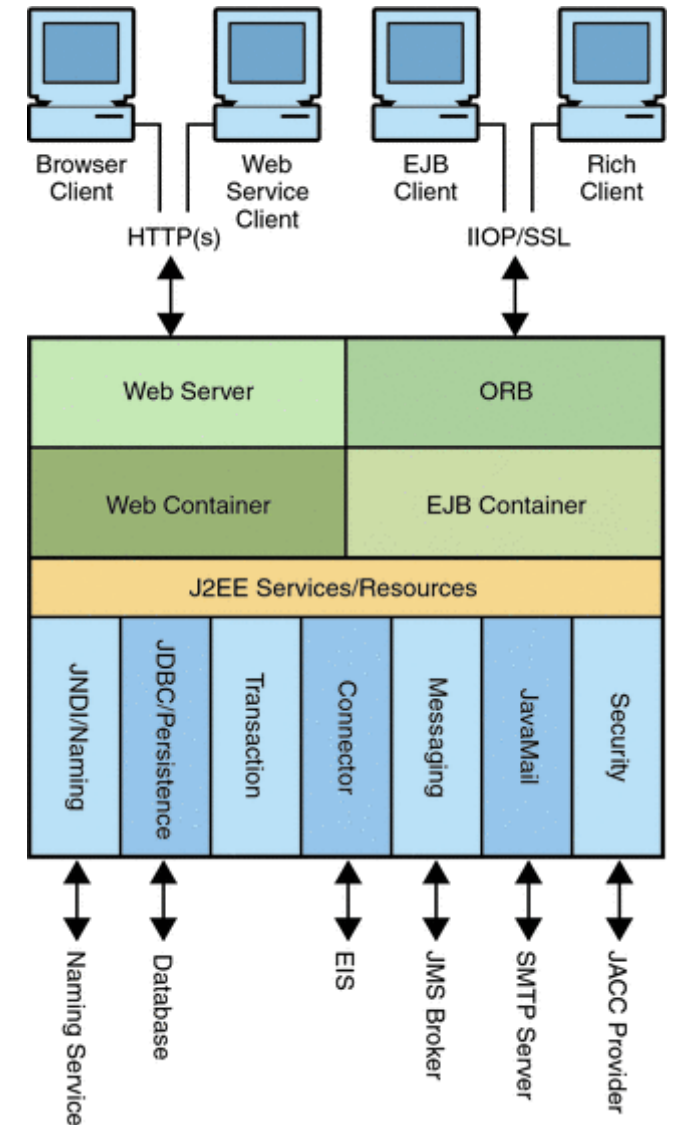
JBoss / J2EE – Begriffserklärung

- Application Server implementieren JEE-Standard
- Deployment standardisierter modularer Komponenten
 - Artefaktbasiert (Pre-Compiled)
 - .jar / .war Dateien auf Server installiert



Quelle: https://i.ytimg.com/vi/KUXdQd_14fU/maxresdefault.jpg

Quelle: <https://docs.oracle.com/cd/E19900-01/819-4741/abfas/index.html>



Serverless - Begriffserklärung

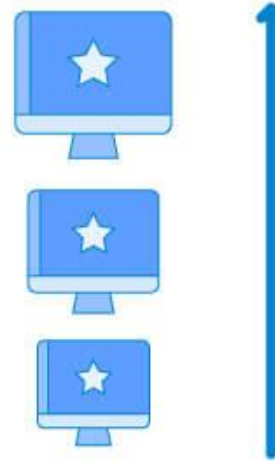
- Synonym: Function-as-a-Service / Serverless Functions
- Business Logik wird in einem Skript programmiert (Function)
- Intern auf Skript-Sprachen zurückgegriffen
 - Keine Kompilierung
 - Interpretation der Befehle zur Laufzeit
- Anwendungen laufen in Containern
- Ein Container ist gleich ein Verarbeitungsthread (Parallelisierungsgrad = 1)
- Sowohl in Jboss & Cloud wird Parallelisierung über Loadbalancer verteilt

Überschrift:
Parallelisierung

Verti: jboss
Hori: cloud

VERTICAL SCALING

Increase size of instance
(RAM, CPU etc.)



HORIZONTAL SCALING

(Add more instances)

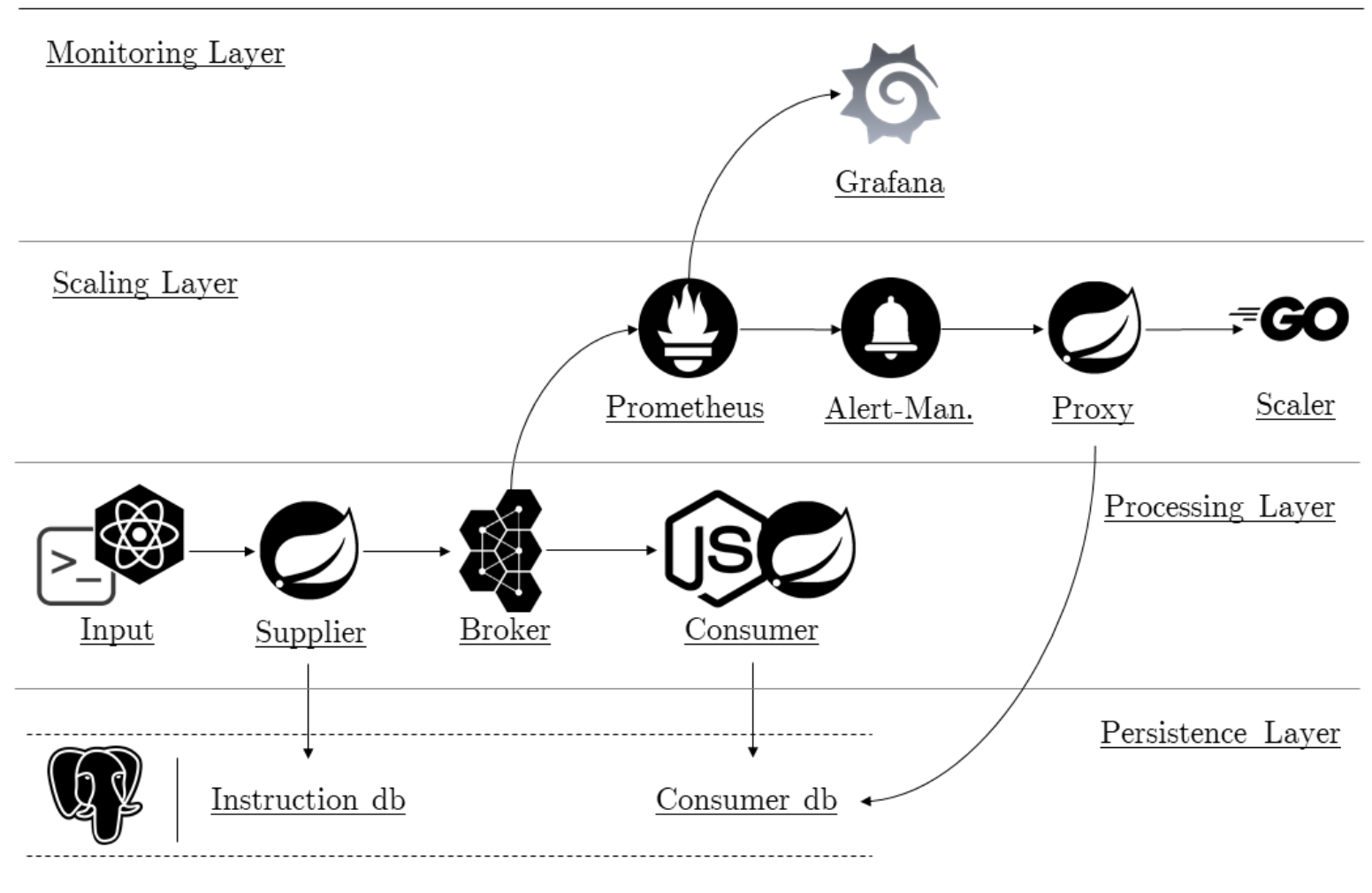


*FaaS platforms execute your code
only when needed and scale
automatically...*

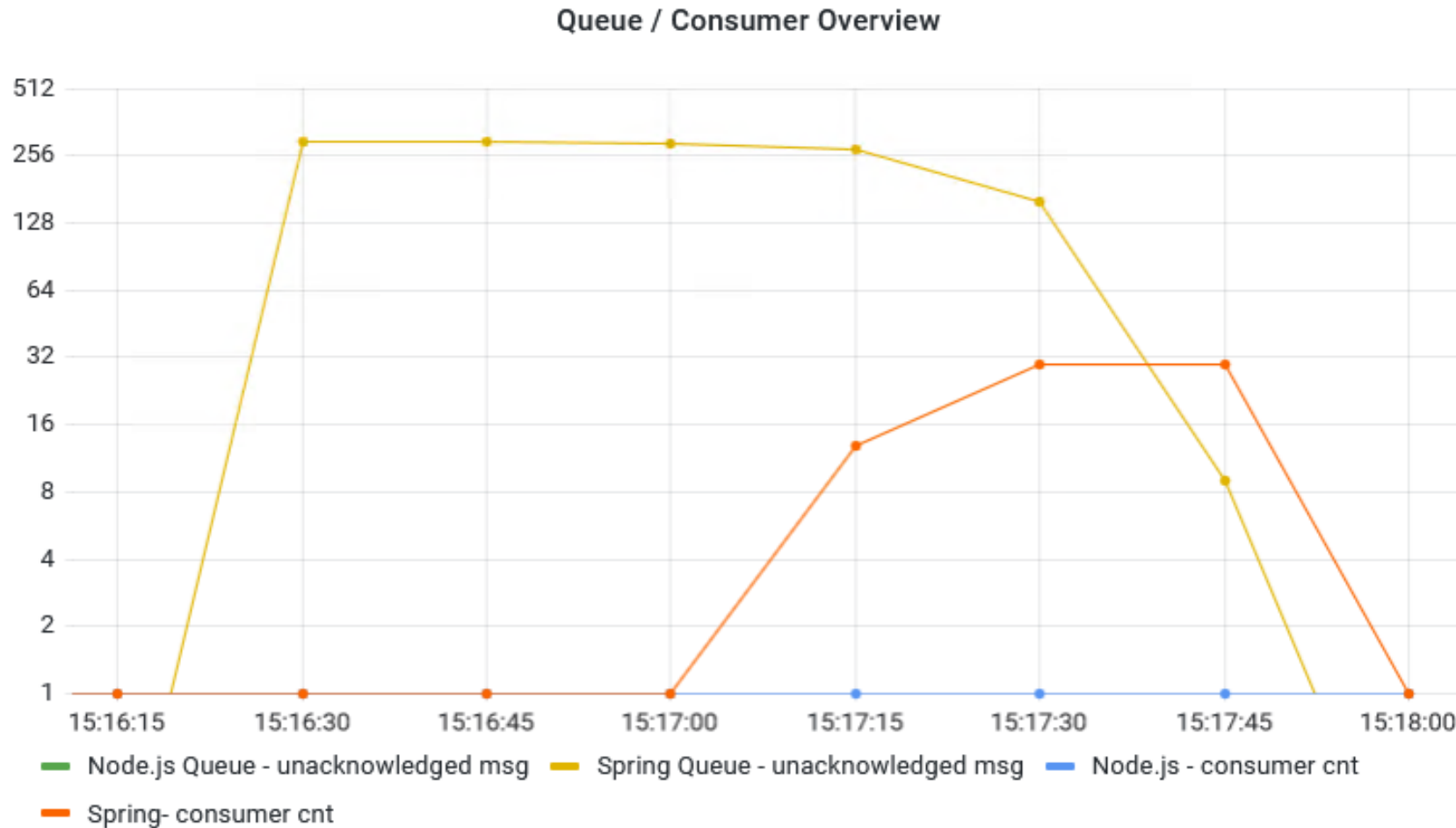
Continuous Delivery in Java

Implementierung des Prototypen (Instant-Payment)

- Scaling Layer:
 - Startet bei Bedarf neue Verarbeitungscontainer
- Processing Layer:
 - Input ist eine Instant-Payment Nachricht
- Consumer
 - implementiert Dummy Business-Logik mit DB-Zugriff
 - Jede Instanz = ein Docker Container
 - Consumer = Skalierungsobjekt

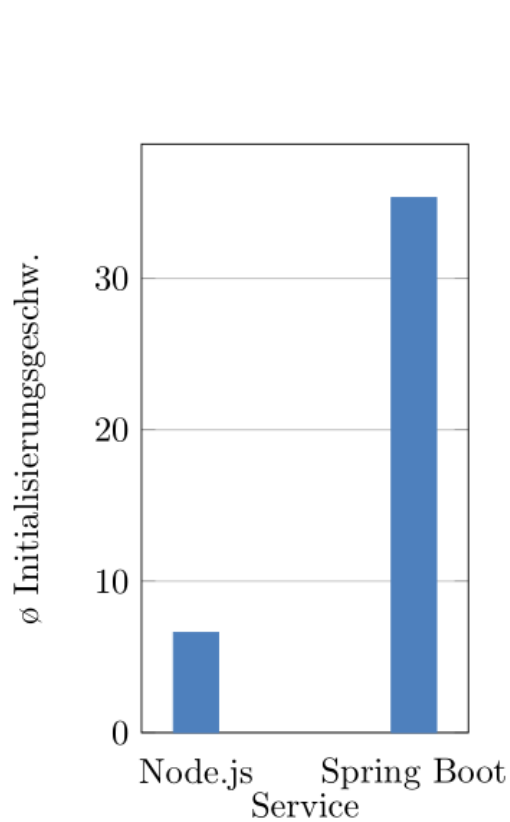


Dynamische Skalierung in der Cloud (Bsp. Prototyp)

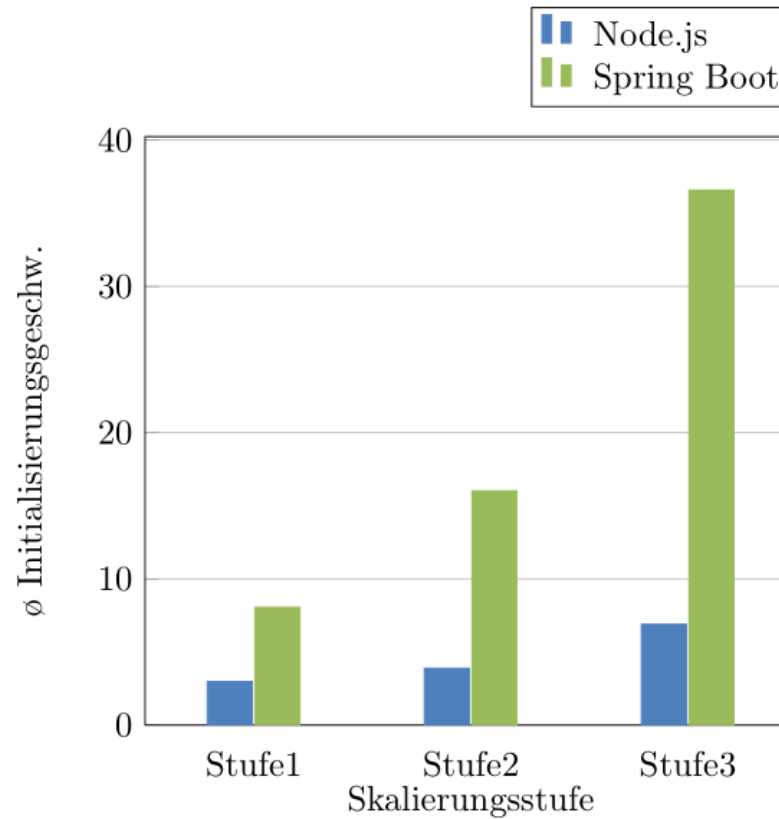


Quelle: Hoffmann – Bsc. Thesis

Ergebnisanalyse / Fazit

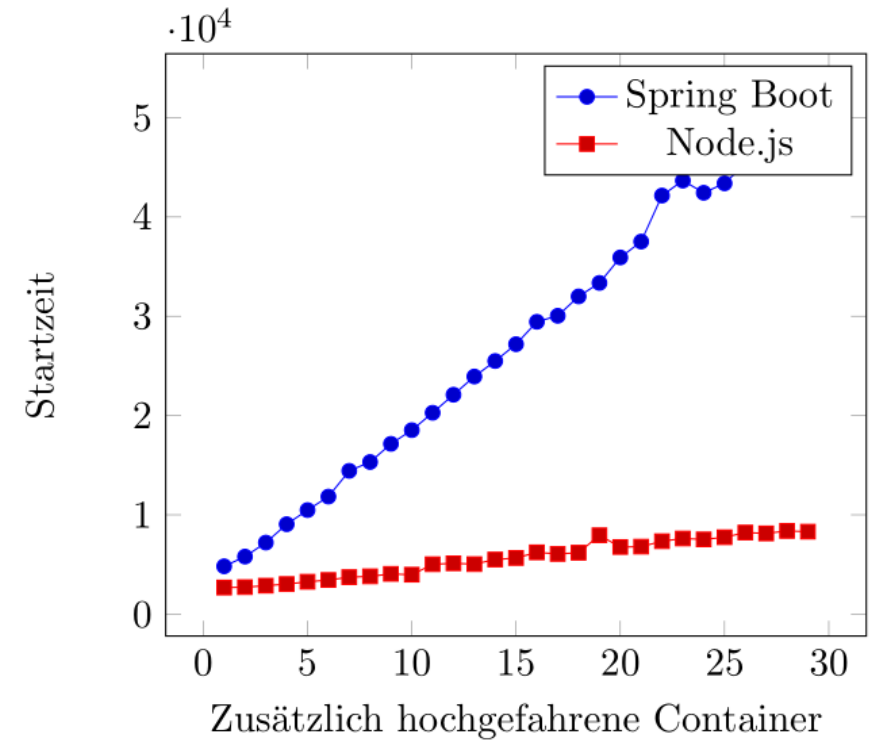


- Node.js: 6,9 Sek.
- Spring: 36,6 Sek.



Stufen ab denen Skalierer neue Instanzen startet

- Stufe 1: 15 Msg. -> 5 Container
- Stufe 2: 30 Msg. -> 10 Container
- Stufe 3: 100 Msg. -> 30 Container



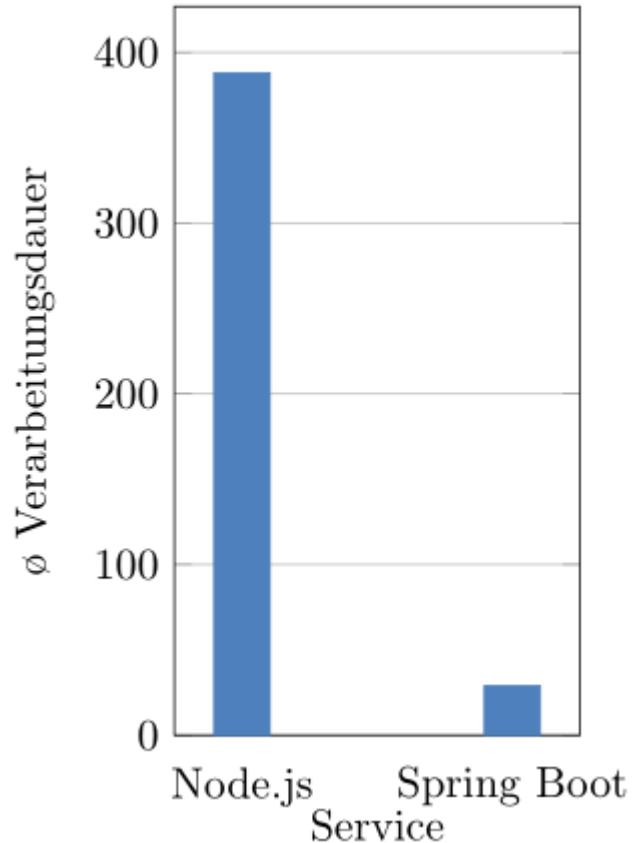
Quelle: Hoffmann – Bsc. Thesis

Linearer Anstieg:

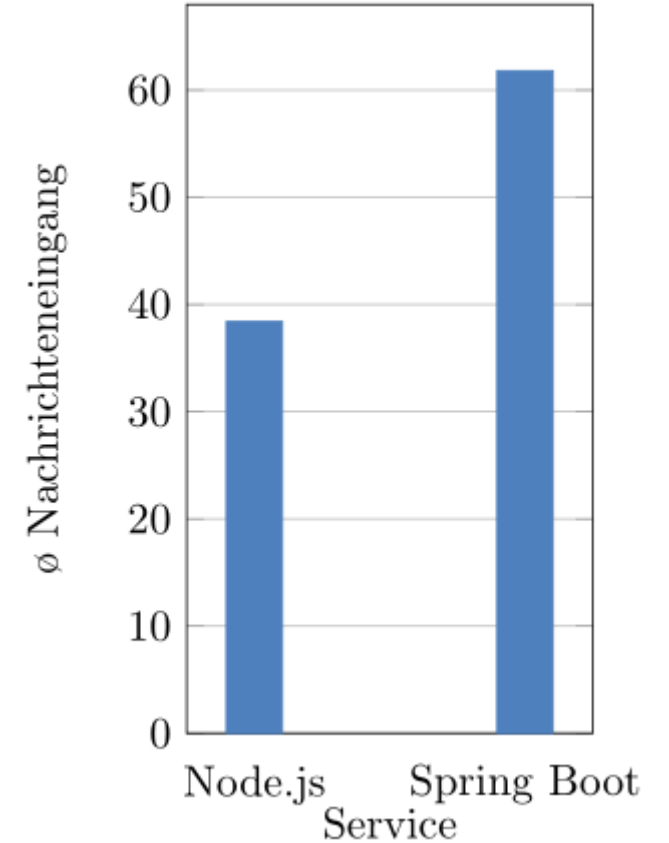
- Node.js: 194 Millis pro Container
- Spring: 1611 Millis pro Container

Ergebnisanalyse / Fazit

- Node.js mit besserem Skalierungsverhalten
- Spring Boot mit besserer Verarbeitungsgeschwindigkeit
- Unterschied beim Nachrichteneingang vernachlässigbar



- Node.js: 388 Millis.
- Spring: 29 Millis.



- Node.js: 38,4 Millis.
- Spring: 61,8 Millis.

Quelle: Hoffmann – Bsc. Thesis

Optimierungspotenzial

Docker	Spring
Ressourcenoptimierung	Spring-Bean - Optimierung der Initialisierungsphase
Ausführungsreihenfolge	
Design For Failure (chaos monkey etc.)	

Thesis - Unterlagen

Gesammelte Unterlagen verfügbar unter:

<https://github.com/derMacon/serverless-bsc-thesis>

Thesis – PDF verfügbar unter:

https://github.com/derMacon/serverless-bsc-thesis/blob/main/thesis/thesis_main.pdf

Vergleich eines Usecases mit Serverless Technologie gegenüber Spring Boot Technologie am Beispiel von Instant Payments

Silas Hoffmann

12 September 2021

Vergleich – Serverless / Application Server

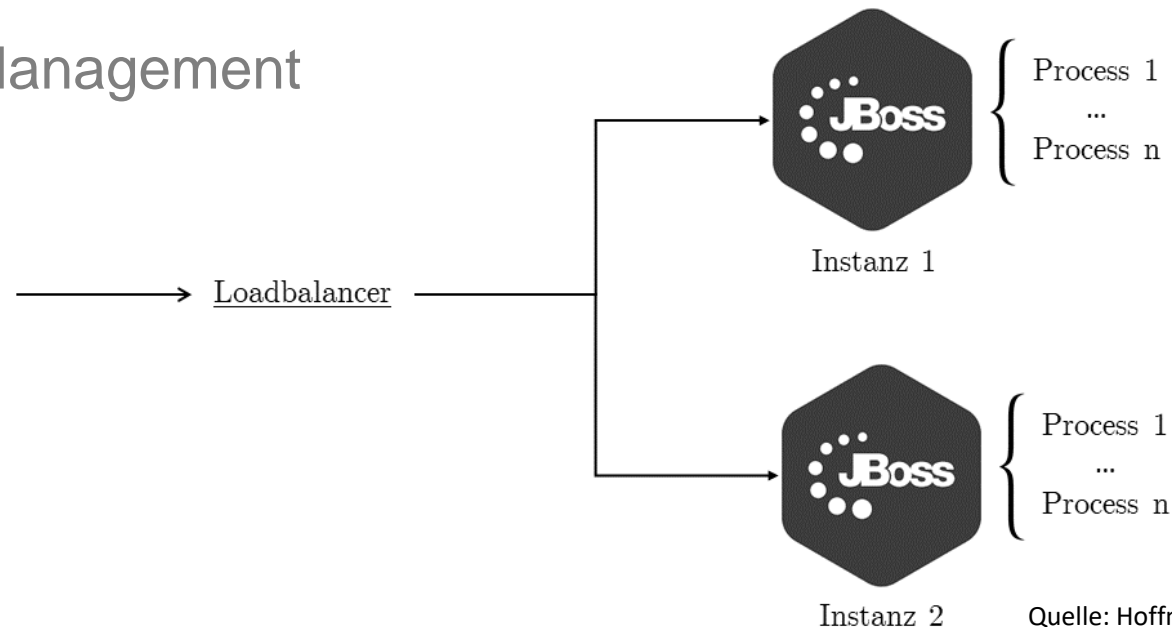
Eigenschaft	Serverless	Application Server
Skalierung innerhalb einer Produktivumgebung	✓	🌀
Unabhängiges Deployment	✓	✗
Skalierte Entwicklung	✓	✗
Konfigurationsoverhead	🌀	✓
Performance	✓	✓

JBoss / J2EE – Begriffserklärung

- Expl. Eigenschaften
 - Kapselung von Datenquellen
 - Schnittstellen für Services
- Impl. Eigenschaften
 - Skalierung
 - Monitoring
 - Lifecycle Management

Application Server: „A software framework that provides facilities to create web applications and a server environment to run them.“

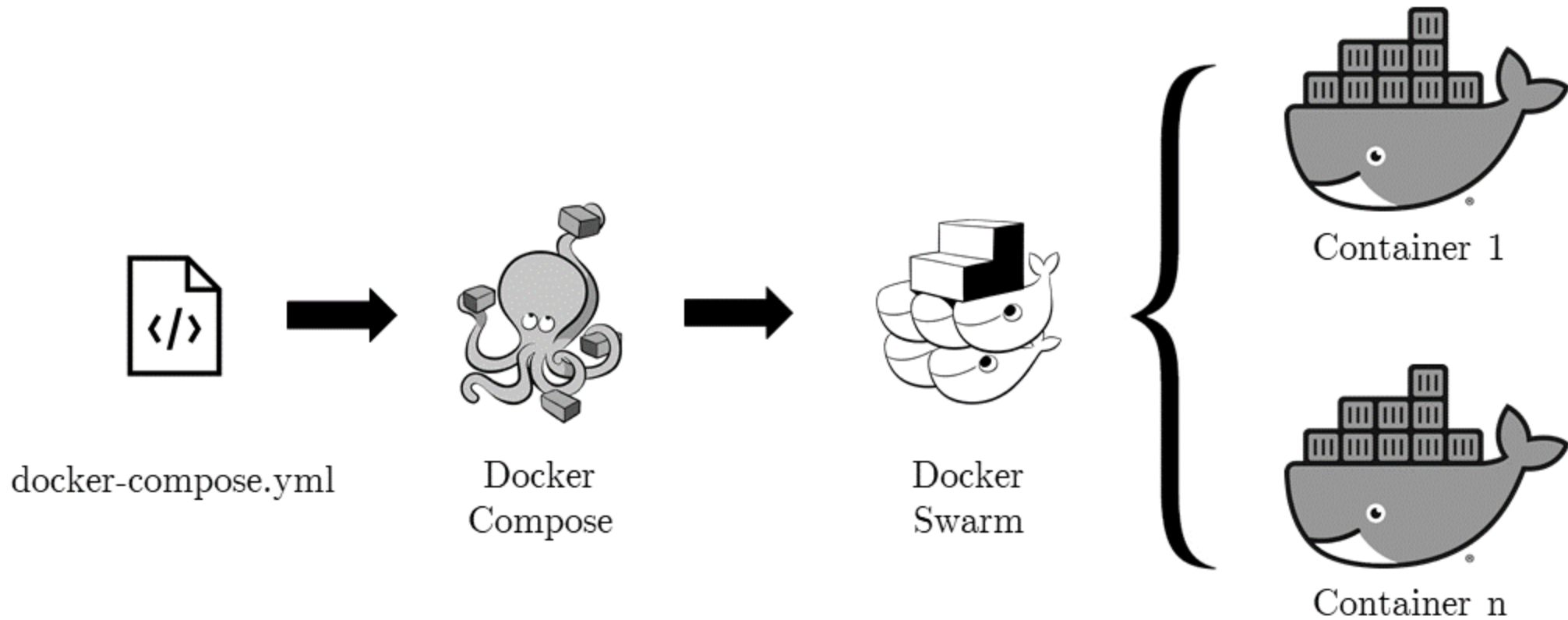
Continuous Delivery in Java



Anforderungen an Deploymentplattform

- Schnittstelle zum Verwalten der Applikation (Host)
- Runtime (Sprachenabhängig)
- Persistenz-Schicht (block store vs. Datenbank)
- Zugriff auf benötigte Middleware
- Automatisiertes Verwalten von Instanzen
- Service Discovery
- Sicherheitsvorkehrungen (z.B. Portbegrenzungen)
- Mechanismus zur Kostengenerierung

Implementierung des Prototypen



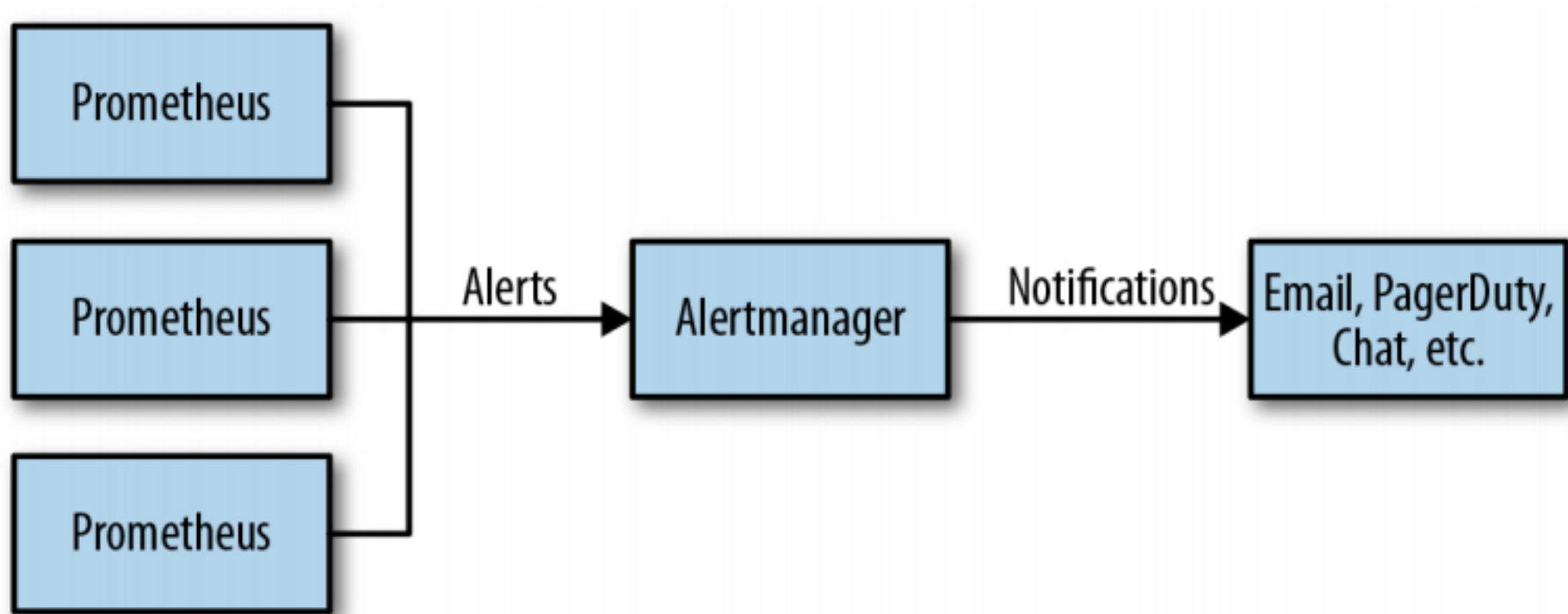
Quelle: Hoffmann – Bsc. Thesis

Skalierung - Regelsatz

$\frac{QL3}{QB2 < MC}$	UP $abs(CB0 - CB3)$	UP $abs(CB1 - CB3)$	UP $abs(CB2 - CB3)$	OK –
$\frac{QL2}{QB1 < MC \leq QB2}$	UP $abs(CB0 - CB2)$	UP $abs(CB1 - CB2)$	OK –	DOWN $abs(CB2 - CB3)$
$\frac{QL1}{QB0 < MC \leq QB1}$	UP $abs(CB0 - CB1)$	OK –	DOWN $abs(CB1 - CB2)$	DOWN $abs(CB1 - CB3)$
$\frac{QL0}{MC \leq QB0}$	OK –	DOWN $abs(CB0 - CB1)$	DOWN $abs(CB0 - CB2)$	DOWN $abs(CB0 - CB3)$
	$\frac{CL0}{CB0 == CC}$	$\frac{CL1}{CB0 < CC \leq CB1}$	$\frac{CL2}{CB1 < CC \leq CB2}$	$\frac{CL3}{CB2 < CC \leq CB3}$

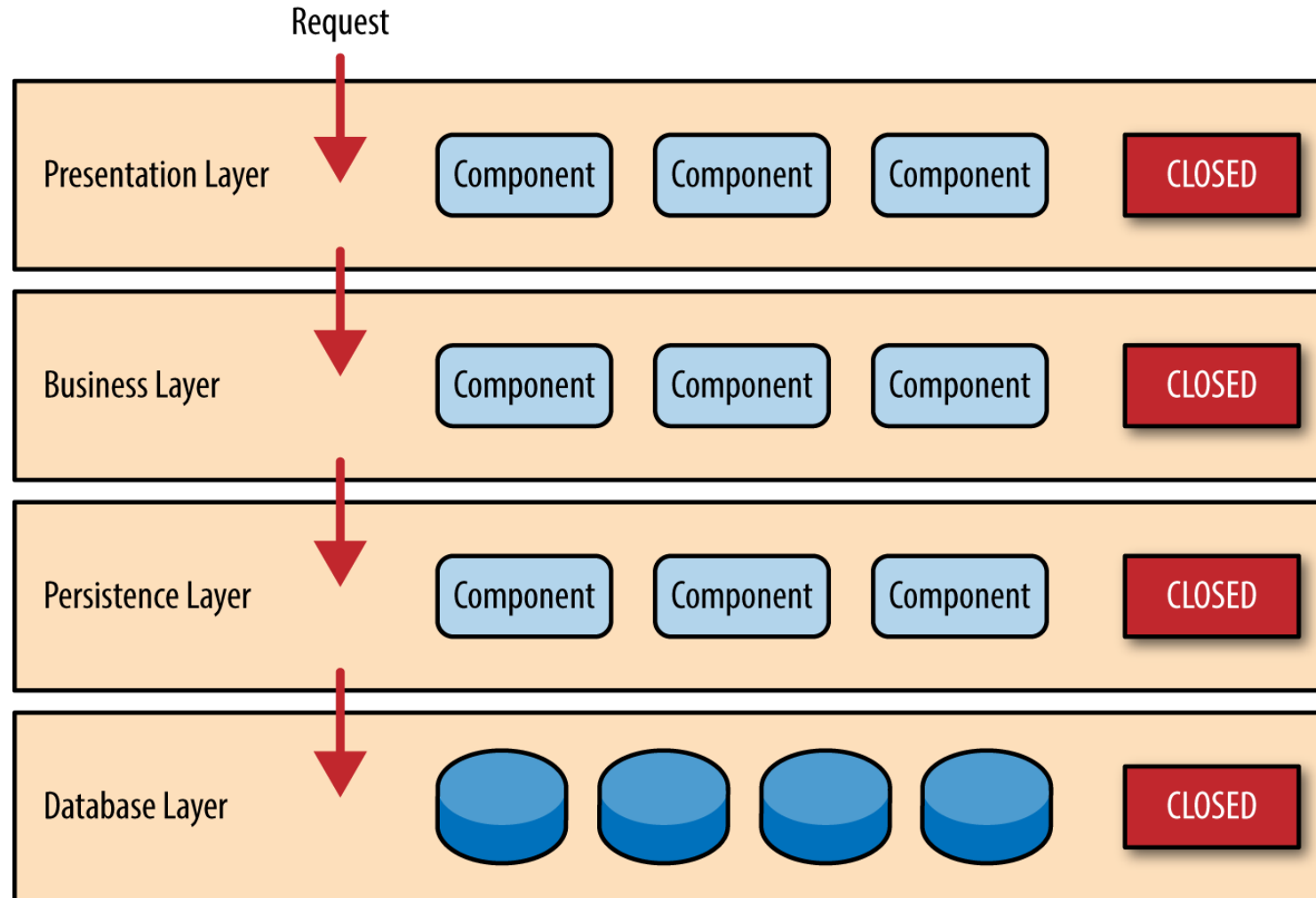
CB0=1 CB2=10 QB0=15 QB2=100 CC: Container Count
 CB1=5 CB3=30 QB1=30 MC: Message Count

Prometheus / Alertmanager



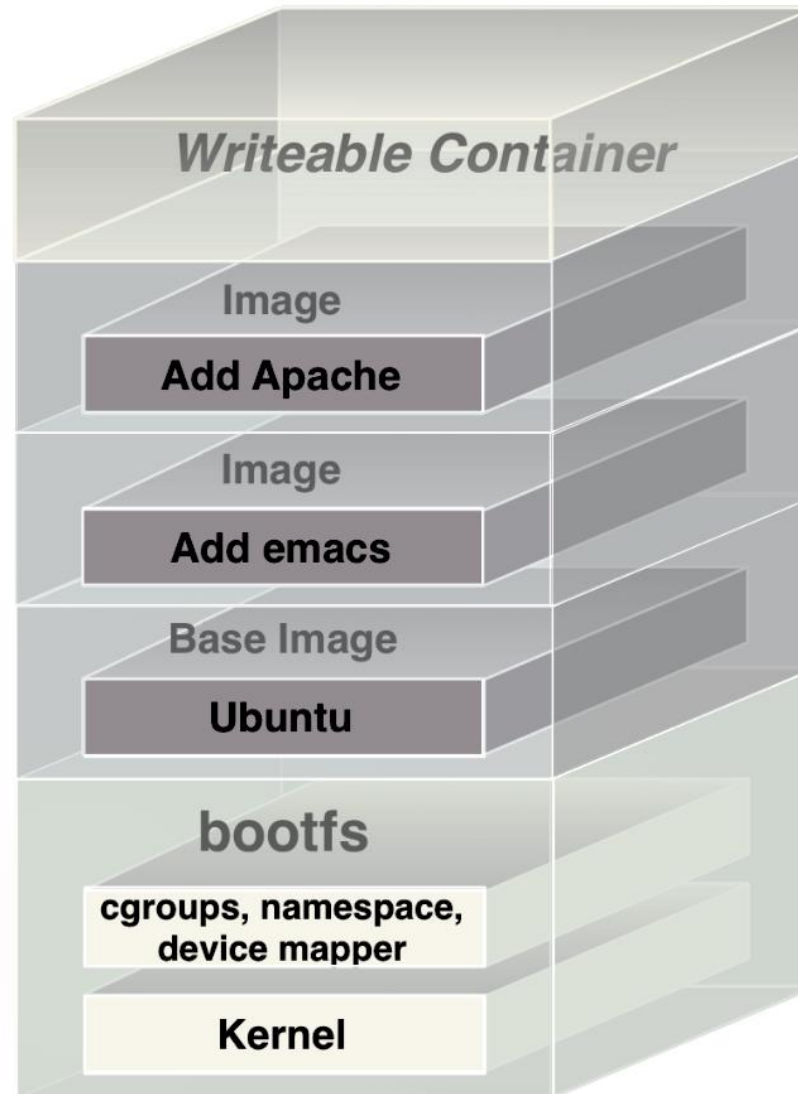
Quelle: Brazil – Prometheus: Up & Running (S. 291)

Tier - Modell



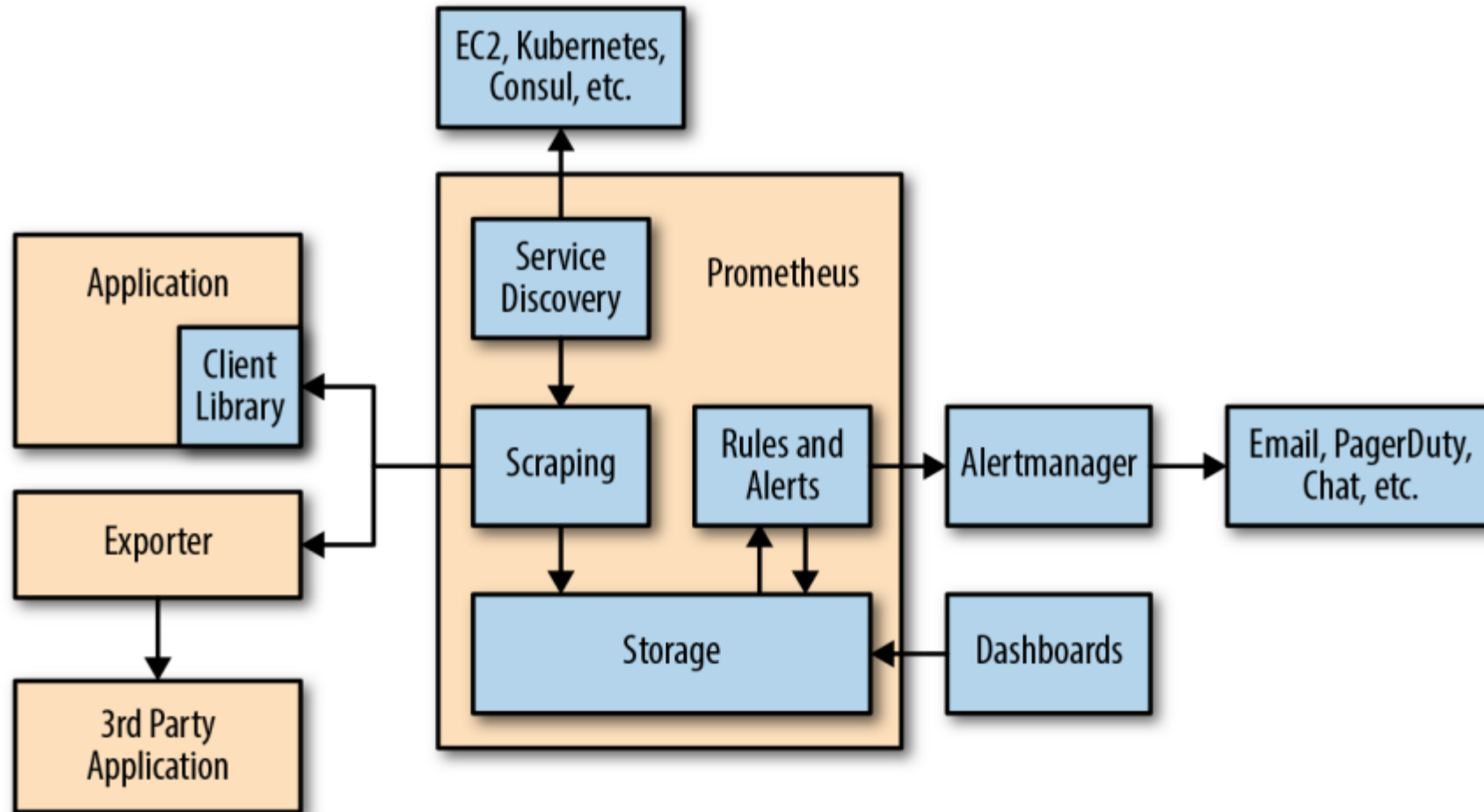
Quelle: <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>

Docker - Aufbau

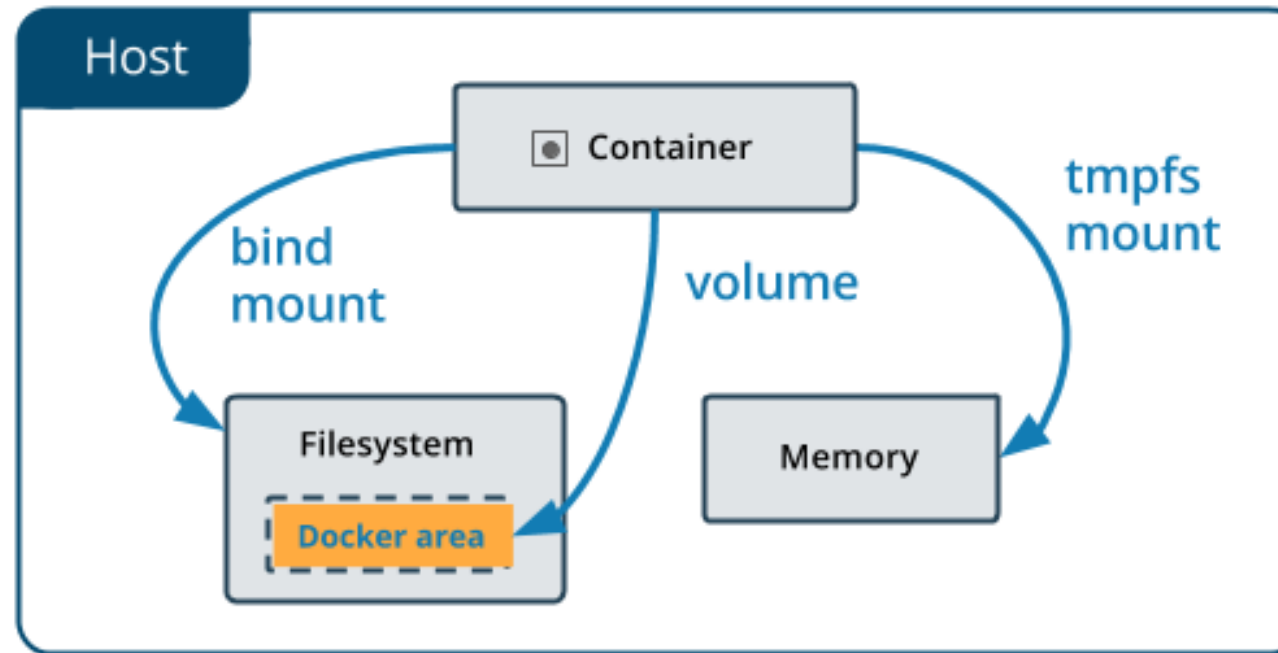


Quelle: J. Turnbull – The Docker Book (S. 72)

Prometheus - Architecture

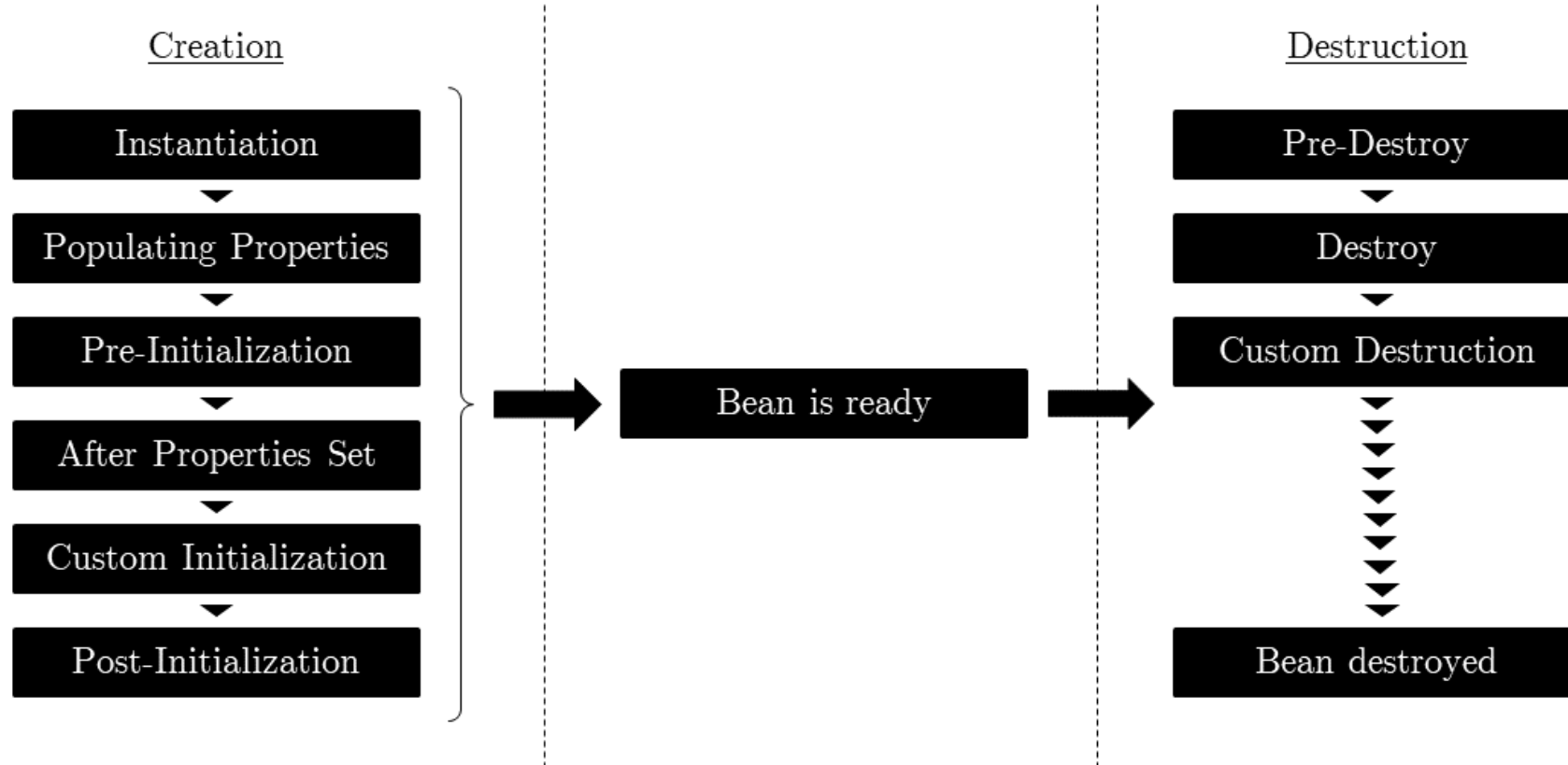


Docker – Types of mounts



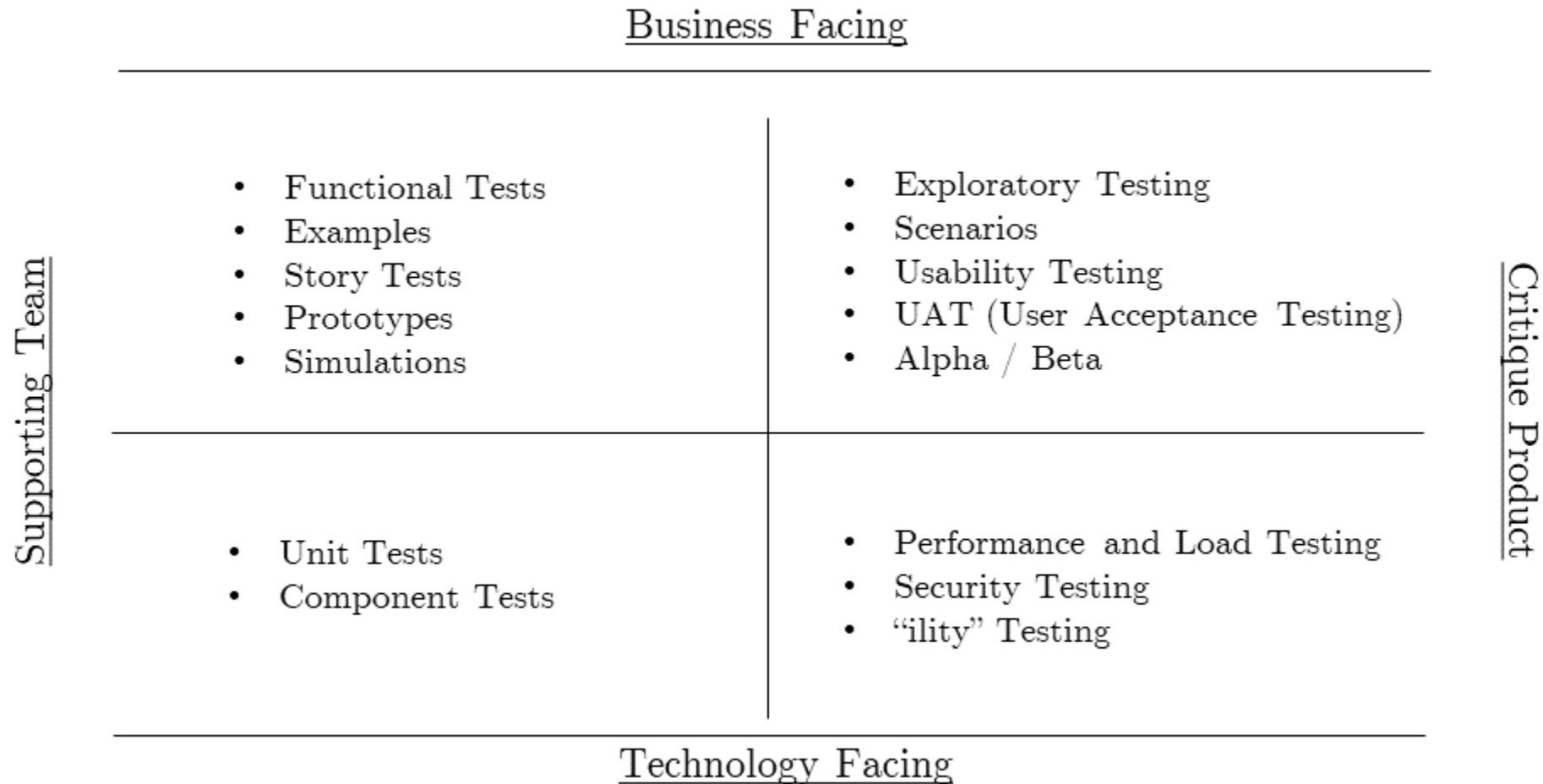
Quelle: Docker Documentation - Kapitel /storage/volumes/

Spring Bean - Lifecycle



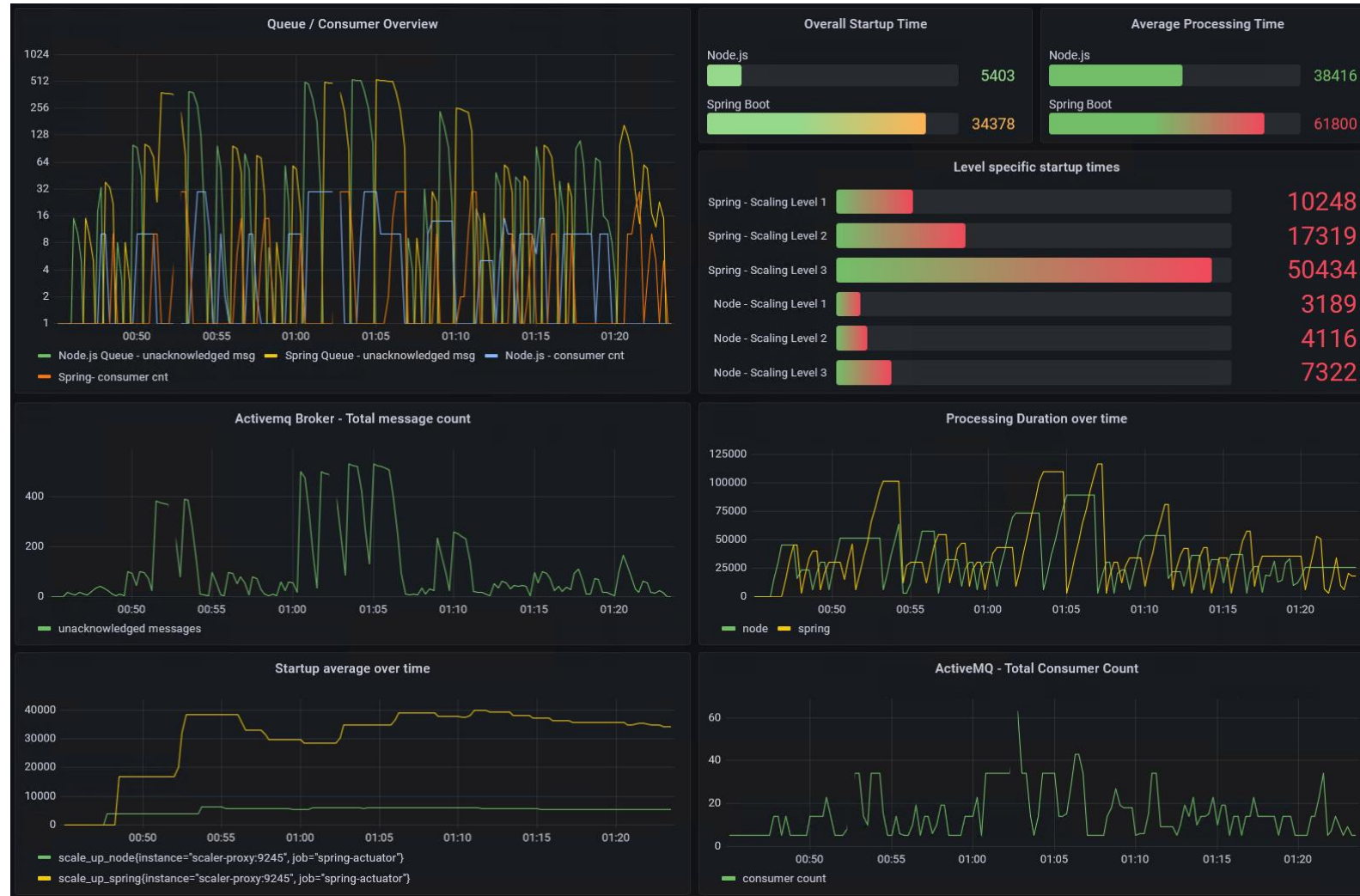
Quelle: Hoffmann – Bsc. Thesis

Agile Testing Quadrants





Quelle: Hoffmann – Bsc. Thesis

Implementierung des Prototypen



Quelle: Hoffmann – Bsc. Thesis

Activemq - Dashboard













Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Support

Queue Name Create Queue Name Filter Filter

Queues:

Name ↑	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
nodeack	0	1	205	205	Browse Active Consumers Active Producers  	Send To Purge Delete
nodequeue	0	1	2765	2765	Browse Active Consumers Active Producers  	Send To Purge Delete
persistencequeue	0	1	5530	5530	Browse Active Consumers Active Producers  	Send To Purge Delete
springack	0	1	263	263	Browse Active Consumers Active Producers  	Send To Purge Delete
springqueue	0	1	2765	2765	Browse Active Consumers Active Producers  	Send To Purge Delete

Queue Views

- Graph
- XML

Topic Views

- XML

Subscribers Views

- XML

Useful Links

- Documentation
- FAQ
- Downloads
- Forums

Copyright 2005-2015 The Apache Software Foundation.

Quelle: Hoffmann – Bsc. Thesis

Input UI

xpath

iban



payment

Randomize messages



Quantity

25



Timespan

0



Start Batch

Quelle: Hoffmann – Bsc. Thesis

Grafan - PromQL

▼ **spring-queue-size** (Prometheus) ? 📄 👁 🗑 ⋮

[Metrics browser >](#)

```
org_apache_activemq_Broker_QueueSize{brokerName="localhost", destinationName="springqueue", destinationType="Queue", instance="activemq:8080", job="services"}
```

Legend ⓘ Spring Queue - unacknow ... **Min step** ⓘ **Resolution** 1/1 ▼

Format Time series ▼ **Instant** ☐ **Prometheus** ⓘ **Exemplars** 👁

Quelle: Hoffmann – Bsc. Thesis

Prometheus - Datasource

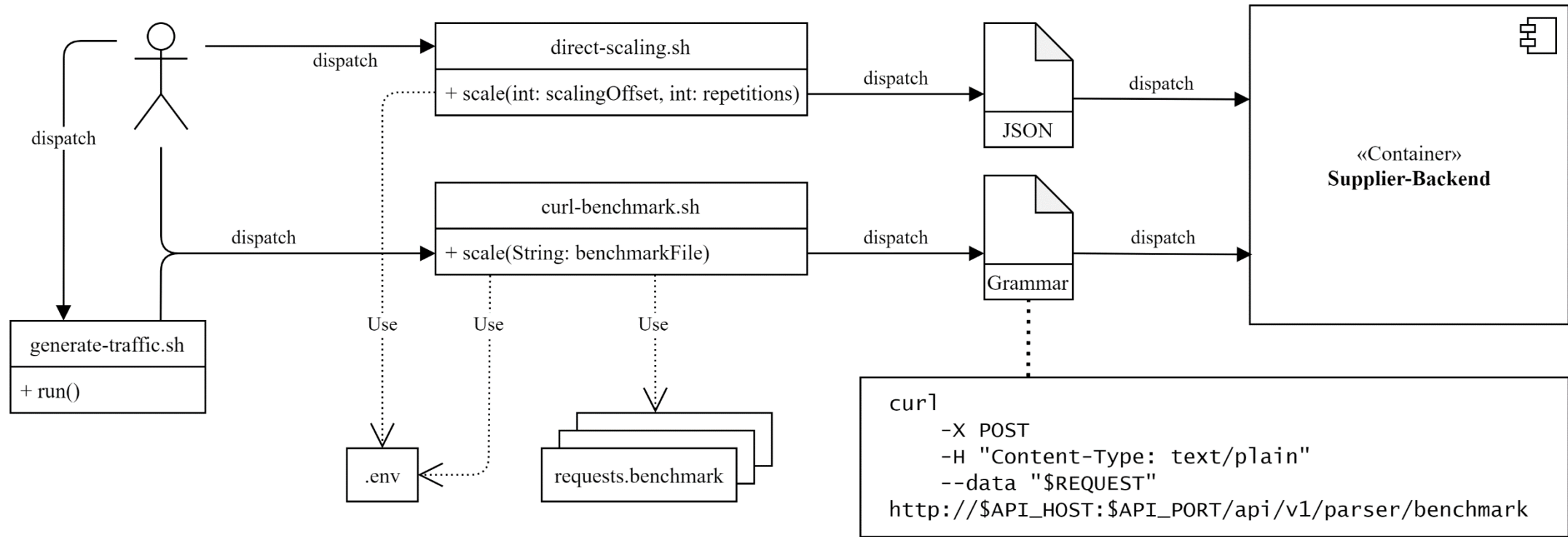
Name	<input type="text" value="Prometheus"/>	Default	<input checked="" type="checkbox"/>
------	---	---------	-------------------------------------

HTTP

URL	<input type="text" value="http://prometheus:9090"/>	
Access	<input type="text" value="Server (default)"/>	Help >
Whitelisted Cookies	<input type="text" value="New tag (enter key to add)"/>	
Timeout	<input type="text"/>	

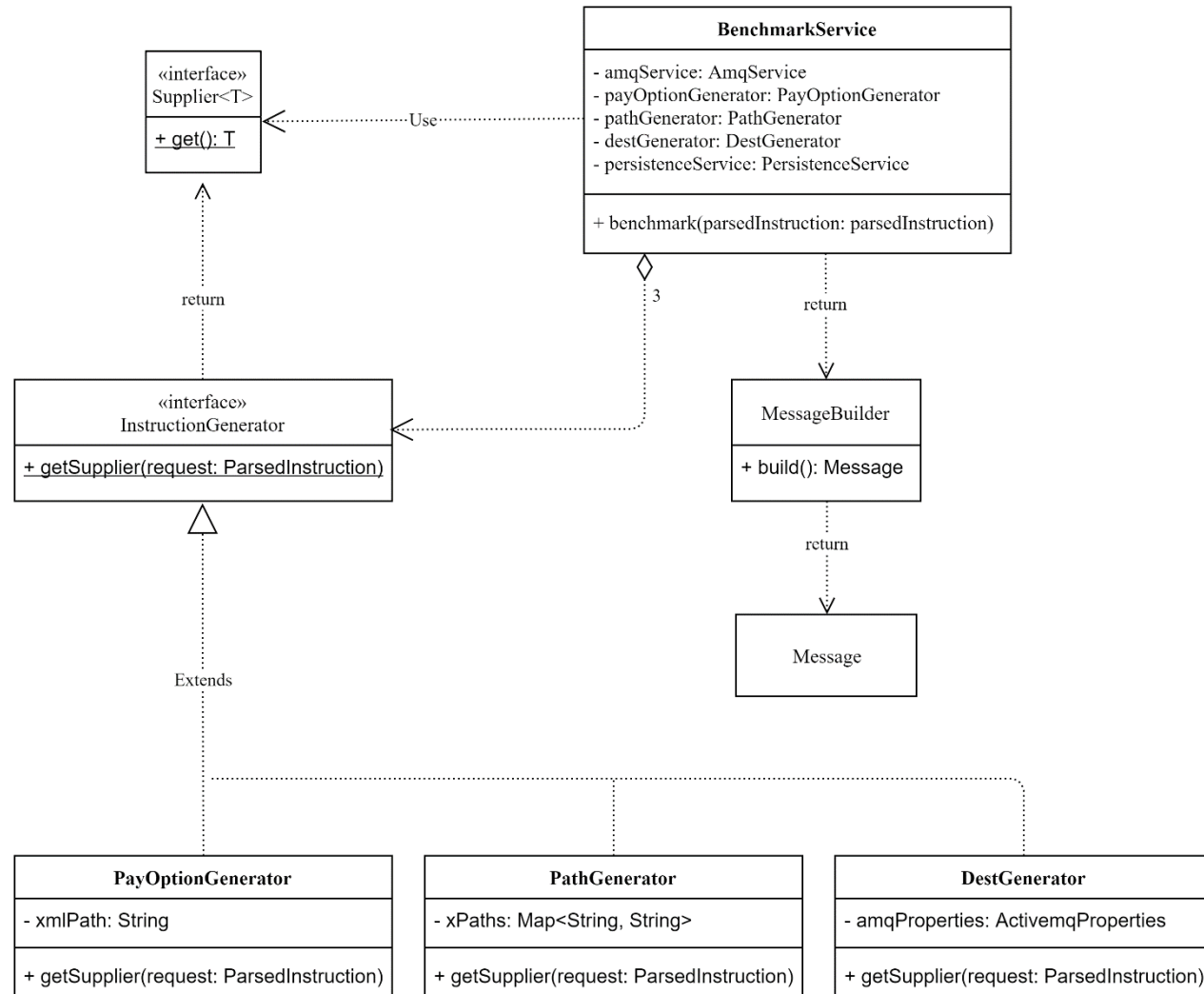
Quelle: Hoffmann – Bsc. Thesis

Input - UML



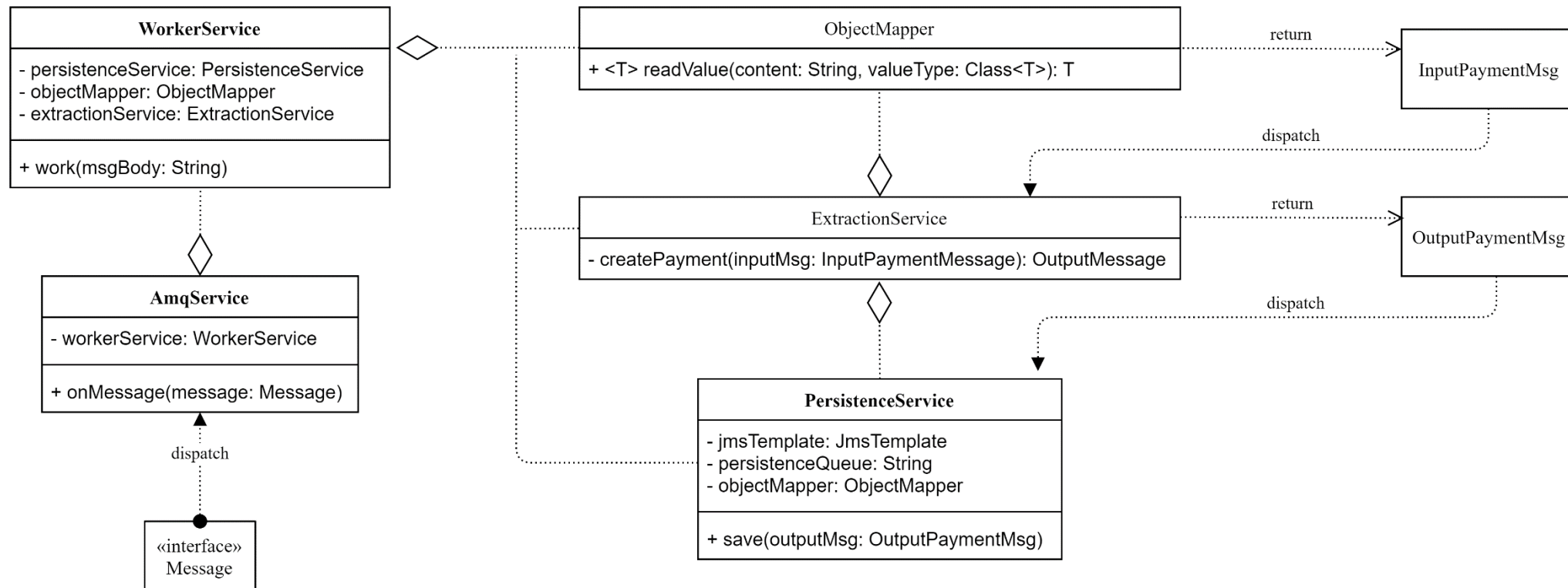
Quelle: Hoffmann – Bsc. Thesis

Supplier - UML



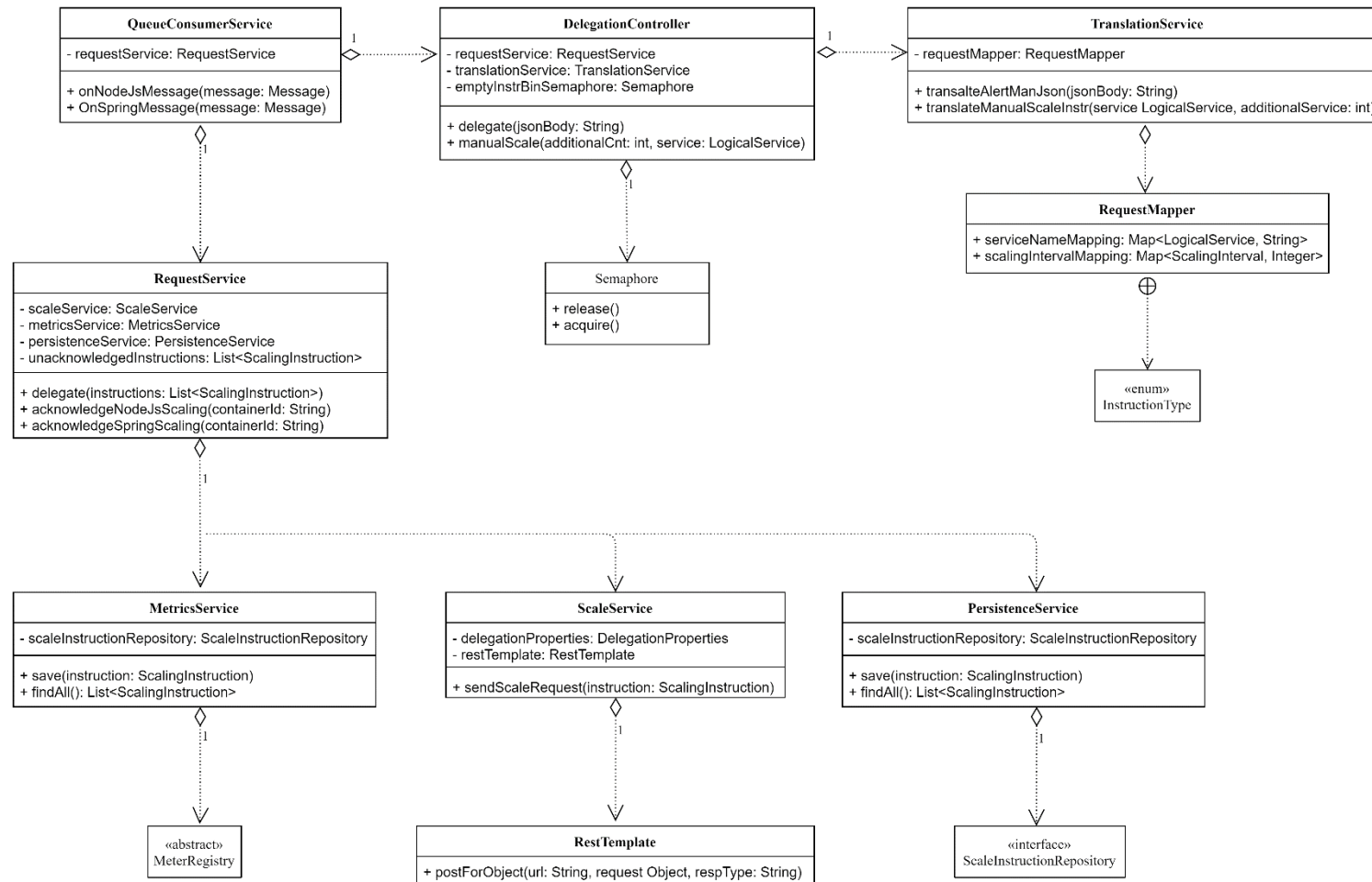
Quelle: Hoffmann – Bsc. Thesis

Consumer - UML



Quelle: Hoffmann – Bsc. Thesis

Scaler Proxy - UML



Quelle: Hoffmann – Bsc. Thesis