Hochschule Karlsruhe - Technik und Wirtschaft

Fakultät für Informatik und Wirtschaftsinformatik
Fachgebiet Wirtschaftsinformatik

MASTERTHESIS

Skalierung von Eclipse Scout Applikationen in der Cloud

von	Herrn Thomas Schweigler
Arbeitsplatz	BSI Business Systems Integration Deutschland GmbH, Frankfurt am Main
Erstbetreuer	Prof. Dr. Jürgen Zimmermann
Zweitbetreuer	Prof. Dr. Andreas Schmidt
Abgabetermin	30.04.2014

Karlsruhe, 30.04.2014

Vorsitzender des Prüfungsausschusses

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die hier vorgelegte Master-Thesis selbstständig und ausschließlich unter Verwendung der angegebenen Literatur und sonstigen Hilfsmittel verfasst habe. Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde zur Erlangung eines akademischen Grades vorgelegt

Frankfurt, den 28.04.2014 _____

Unterschrift

Inhaltsverzeichnis

Abkürzungen

GoF Gang of Four
IaaS Infrastructure as a Service
PaaS Platform as a Service
SaaS Software as a Service
SNS Simple Notification Service
SQS Simple Queue Service
NIST National Institute of Standards and Technology
PaI Platform-as-Infrastructure
QoS Quality of Service

1 Einleitung

1.1 Motivation

Eclipse Scout ist ein Framework zur Entwicklung moderner, serviceorientierter Geschäftsapplikationen, das in den letzten Jahren immer mehr an Bedeutung gewonnen hat. Das belegen steigende Downloadzahlen und eine immer größer werdende Community. In den letzten Jahren ist nicht nur die Anzahl der mit Eclipse Scout erstellten Applikationen stark angestiegen, sondern auch die Anzahl der Nutzer hat sich erhöht, die auf eine Eclipse Scout Anwendung zugreifen. Das hat zur Folge, dass immer mehr Nutzer gleichzeitig auf eine Eclipse Scout Applikation zugreifen, was wiederum zu Einbußen bei der Performance führt. Eine weitere Herausforderung ist, dass immer mehr Fachabteilungen den Einsatz von Eclipse Scout evaluieren möchten, ohne Aufwände in der IT-Abteilung zu generieren. Bisher ist jedoch für den Betrieb einer Eclipse Scout Anwendung ein eigener Applikationsserver notwendig.

Parallel zu der Entwicklung von Eclipse Scout erhält das Thema Cloud einen immer größeren Stellenwert (vgl. Bit). Was im Jahr 2006 mit den Amazon Webservices begann hat sich inzwischen zu einem rasant wachsenden Milliardenmarkt entwickelt. Immer mehr Software-Anbieter stellen Ihre Produkte als Cloud-Lösung zur Verfügung (vgl. Pra) und es werden immer größere Beträge in die Cloud-Infrastruktur investiert (vgl. McK). Damit Applikationen, die auf Basis von Eclipse Scout erstellt wurden mit den steigenden Nutzerzahlen umgehen können, ist es notwendig diese skalierbar zu machen. Da Skalierbarkeit eine der Kernkompetenzen von Cloud-Angeboten ist, bietet sich eine Verbindung der Skalierbarkeit von Eclipse Scout Applikationen mit deren Fähigkeit zum Betrieb innerhalb einer Cloud an.

1.2 Ziel der Arbeit

Ziel dieser Masterthesis ist es, das Eclipse Scout Framework so zu erweitern, dass damit Applikationen erstellt werden können, die skalierbar sind und in möglichst vielen Platform as a Service (PaaS) Clouds ohne großen Aufwand betrieben werden können. Um die Anforderungen definieren zu können, die Eclipse Scout Applikationen an eine PaaS Cloud stellen, soll die Architektur von Eclipse Scout Applikationen hinsichtlich der betriebsnotwendigen Eigenschaften der Laufzeitumgebung untersucht und denkbare Möglichkeiten für eine Unterstützung bei der Skalierung beschrieben werden. Anhand dieser Kriterien soll eine Auswahl an PaaS Anbietern evaluiert und Implementierungen für zwei dieser PaaS Anbieter erstellt und beschrieben werden. Abschließend sollen Skalierungsmessungen zeigen, ob eine Demo-Applikationen, die auf dem angepassten Framework basiert, in einer der beiden ausgewählten PaaS Clouds skaliert. Im Rahmen dieser Masterthesis soll nur der Applikationsserver betrachtet werden. Der Webserver ist bezüglich der Skalierung und dem Betrieb in der Cloud nicht Teil dieser Arbeit.

2 Analyse und Konzeption

Im vorhergehenden Kapitel 1 wurde beschrieben, warum das Eclipse Scout Framework um eine Unterstützung für die Erstellung skalierbarer Applikationen und den Betrieb der Applikationen in einer PaaS Cloud erweitert werden soll. Dieses Kapitel zeigt, wie die Architektur von Eclipse Scout Applikationen beschaffen ist und inwieweit sie sich von der Architektur anderer cloudfähiger Applikationen unterscheidet. Weiterhin werden die Eigenschaften von Cloud-Systemen beschrieben. Im darauffolgenden Kapitel 3 werden aus der bisherigen Architektur der Eclipse Scout Applikationen und den Eigenschaften der Cloud Anforderungen definiert, die eine Laufzeitumgebung erfüllen muss, um Eclipse Scout Applikationen skalierbar betreiben zu können. Zudem werden im 3. Kapitel verschiedene PaaS Anbieter gegenüber der definierten Anforderungen geprüft und bewertet.

2.1 Architektur von Eclipse Scout Applikationen

Die Architektur von Eclipse Scout Applikationen wird durch das Eclipse Scout Framework vorgegeben. Hierfür werden durch das Eclipse Scout Framework verschiedene Komponenten bereitgestellt. Da das Eclipse Scout Framework nur den Rahmen für eine Applikation bereitstellt, wird in dieser Masterthesis die Demo-Applikation BahBahChat [1] [2] als Referenz-Applikation verwendet. Diese Implementiert eine kleine Chat-Applikation und bietet durch die beidseitige Kommunikation zwischen Server und Client eine gute Basis, um Skalierungseffekte zu beobachten.

1 http://www.bsiag.com/scout/bahbahchat-eclipse-scout-tutorial-eclipsecon-2012/
2 https://github.com/BSI-Business-Systems-Integration-AG/org.eclipsescout.demo/tree/3.9/bahbah

2.1.1 Komponenten

Das Eclipse Scout Framework stellt Modelle und Implementierungen für einen Applikationsserver, einen Webserver und zwei Rich-Clients zur Verfügung. Werden alle durch das Eclipse Scout Framework bereitgestellten Komponenten verwendet, so entsteht eine Applikation mit einer erweiterten Client-Server Architektur. Da die mit Eclipse Scout erstellten Anwendungen meist Geschäftsanwendungen sind, die zur Datenverarbeitung eingesetzt werden, kann die Architektur um eine Datenbank ergänzt werden. Zusammengesetzt entsteht daraus die in der Abbildung 2.1 dargestellte 4-Schichten Architektur. Diese zeigt in blau dargestellt, die von Eclipse Scout bereitgestellten Komponenten und in orange Komponenten von Drittanbietern, auf die zugegriffen wird oder die auf die von Eclipse Scout bereitgestellten Komponenten zugreifen. Die erste Schicht ist für die Datenhaltung verantwortlich. Die Datenhaltungsschicht wird, wie durch die orange Hinterlegung ersichtlich ist, selbst nicht durch das Eclipse Scout Framework implementiert, ihre Anbindung jedoch durch eine vordefinierte Schnittstelle vereinfacht. In der 2. Schicht befindet sich der Applikationsserver. Dieser beinhaltet die Geschäftslogik und ist für die Anbindung der Datenbank und der Schnittstellen verantwortlich. Zudem ist er der Ansprechpartner für den Web-Server und die Rich-Clients. Der Web-Server wiederum befindet sich in der 3. Schicht, der Präsentationsschicht. Er stellt je nach Endgerät ein Web-Frontend bereit, welches speziell für Desktop-Geräte bzw. Laptops, Tablets oder Smartphones optimiert ist. Die beiden Rich-Clients beinhalten sowohl Komponenten der 3. als auch der 4. Schicht. Das heißt, in ihnen steckt die Präsentationslogik und die Visualisierungstechnik. Neben den Rich-Clients sind auch die Browser der verschiedenen Endgeräte in der 4. Schicht zu finden. Sie visualisieren das vom Webserver bereitgestellte Web-Frontend.

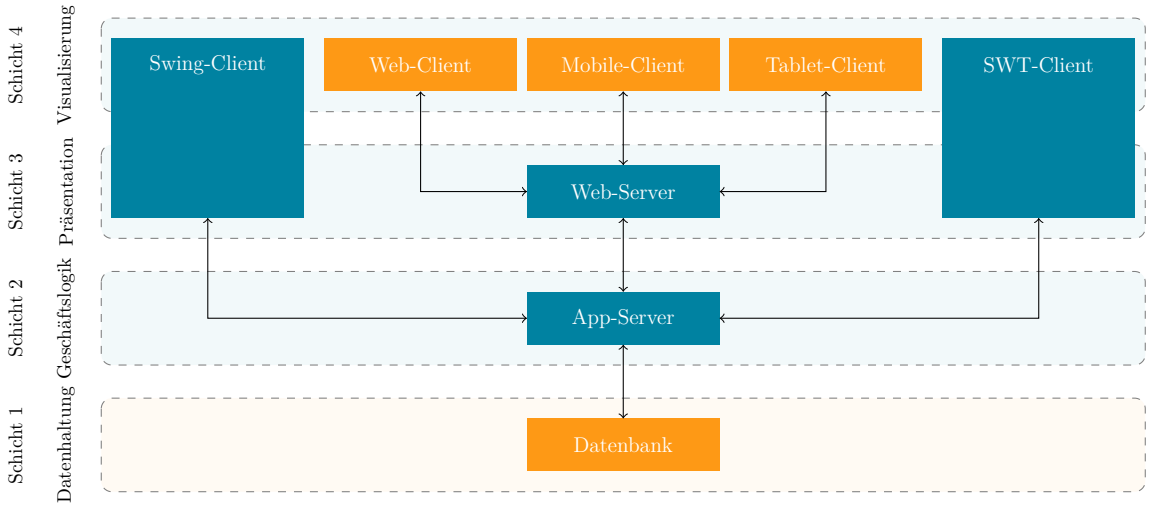

Abb. 2.1: Architektur einer Eclipse Scout Applikation

2.1.2 Bestandteile der Komponenten

Aus technischer Sicht basieren alle vier Komponenten, die durch das Eclipse Scout Framework bereitgestellt werden, auf der OSGi Plattform. Das bedeutet die einzelnen Komponenten bestehen aus verschiedenen Bundles. Diese Bundles können wiederum logischen Paketen zugeordnet werden. Diese Pakete sind in der Grafik 2.2 zu sehen. Auf der linken Seite ist der Aufbau der Rich-Clients zu sehen, rechts der Aufbau des Web-Servers und der untere Block zeigt die Server-Applikation. Die Kommunikation zwischen Client und Server erfolgt über einen Service-Tunnel. Die Bestandteile der einzelenen Komponenten werden im Folgenden beschrieben.

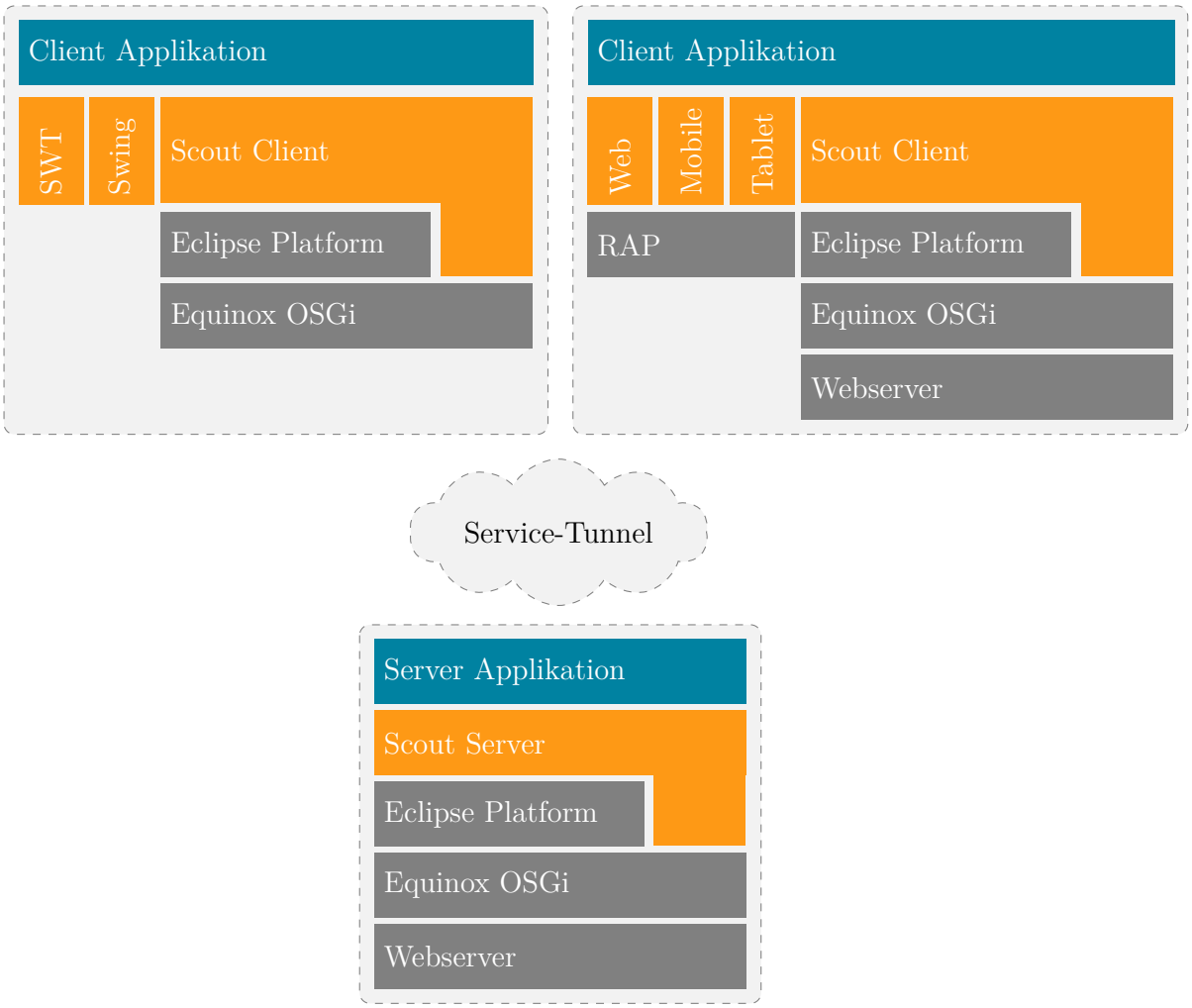

Abb. 2.2: Architektur Eclipse Scout, Angelehnt an (Cla11)

Applikationsserver

Bei dem Applikationsserver wird Eclipse Equinox, und damit auch der Rest der Applikation, innerhalb eines Servlet Containers betrieben. Die Bundles aus denen der Applikationsserver besteht gehören entweder zur Eclipse Plattform, dem Eclipse Scout Server oder der Server Applikation. Die Eclipse Plattform stellt Basis-Funktionalitäten bereit, die vom Eclipse Scout Server genutzt werden. Dem Eclipse Scout Server werden Bundles zugeordnet, die vom Eclipse Scout Framework bereitgestellt werden. Der Server Applikation werden Bundles zugeordnet, die vom Entwickler erstellt werden. In der Abbildung 2.3 ist dies nochmals schematisch dargestellt. Das Zusammenspiel der Bundles innerhalb der Server-Komponente wird über die OSGi Service Registry geregelt. Hier registrieren die verschiedenen Bundles die Services, die sie anbieten. Einzelne Services können priorisiert und somit auch ausgetauscht werden (All07).

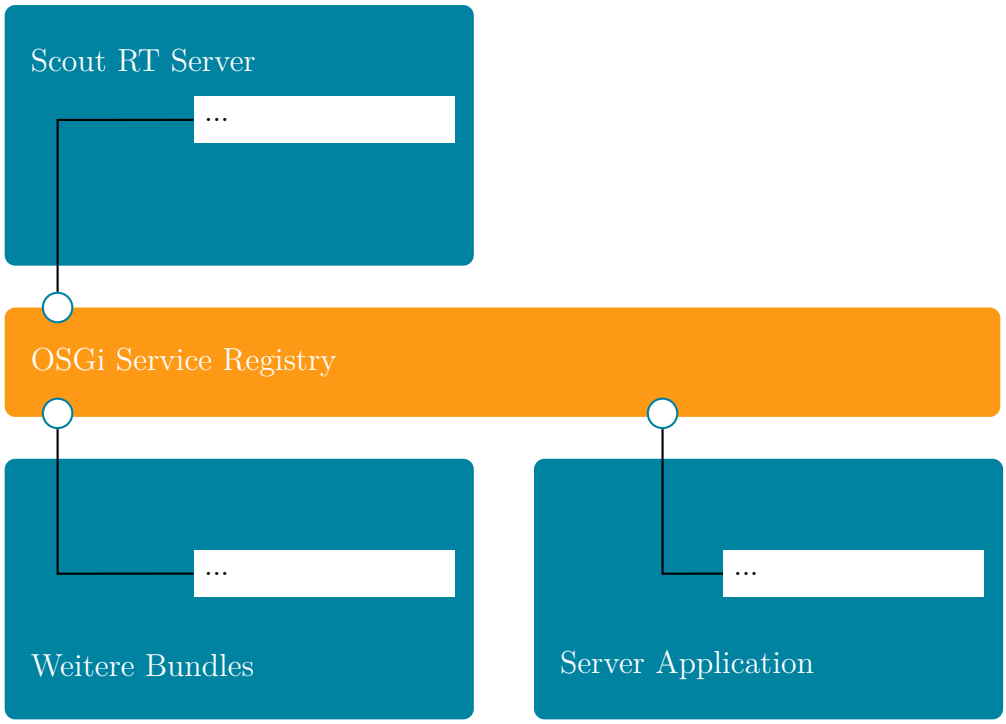

Abb. 2.3: Bundles der Serverapplikation

RAP-Webserver

Bei dem RAP-Webserver wird analog zum Applikationsserver Eclipse Equinox innerhalb eines Servlet Containers betrieben. Die Bundles, die der Webserver nutzt unterscheiden sich jedoch von denen des Applikationsservers. Zwar nutzt der Webserver ebenfalls verschiedene Bundles der Eclipse Plattform, besitzt aber weder Bundles des Scout Servers noch der Server Applikation. Stattdessen werden Bundles des Scout Clients, der Client Applikation und Eclipse RAP genutzt, um den Webserver zu betreiben. Dieser ist dann in der Lage Web-Oberflächen optimiert für Web-, Mobile-, und Tablet-Clients bereitzustellen (vgl. Abb. 2.4).

Rich-Clients

Die Rich-Clients basieren ebenfalls auf Eclipse Equinox. Im Gegensatz zu dem Applikations- und dem Webserver werden diese jedoch nicht innerhalb eines Webserver betrieben sondern als eigenständige Applikation. Sie laden neben der Eclipse Plattform den Scout Client. Zusätzlich wird je nach Art des Rich-Client noch die GUI Komponente für SWT oder SWING eingebunden.

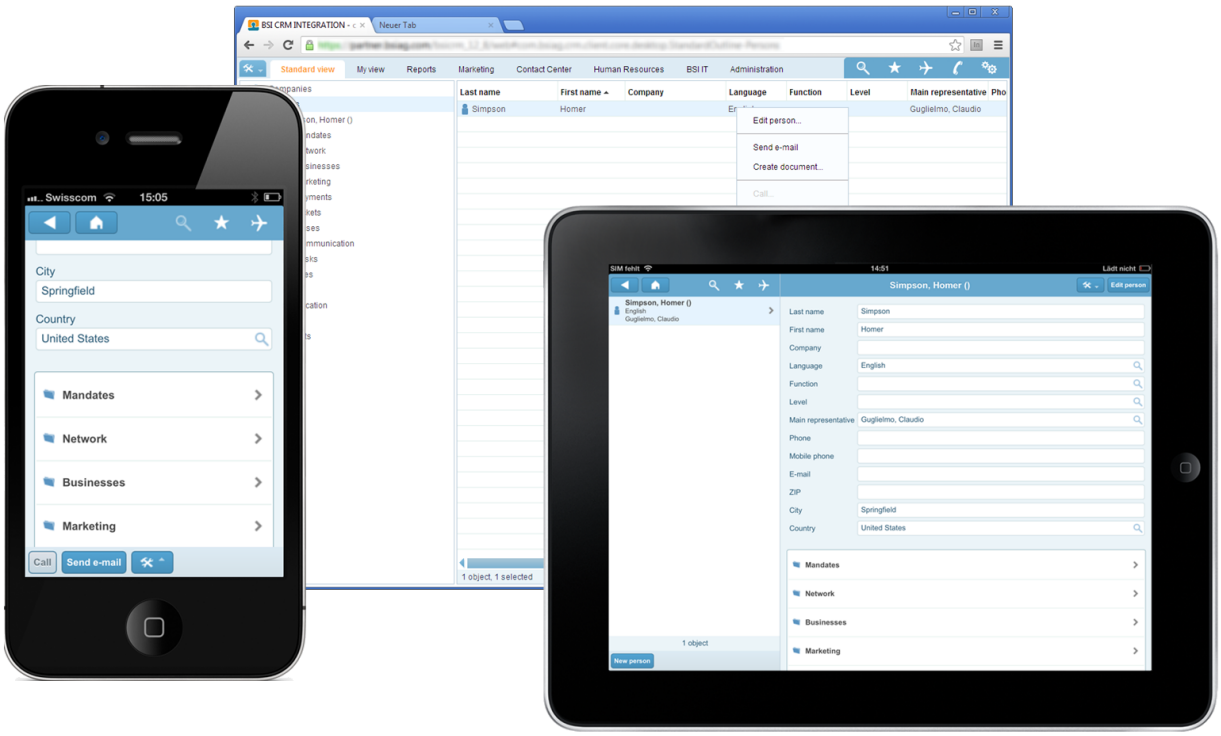

Abb. 2.4: Eclipse Scout Web-Frontends

2.1.3 Kommunikation der Komponenten

Um aus den zuvor beschriebenen einzelnen Komponenten eine lauffähige Applikation zu machen, ist es notwendig, dass die einzelnen Komponenten untereinander kommunizieren. Die Kommunikationswege und -richtungen sind in der Abbildung 2.1 bereits zu sehen. Die Kommunikation zwischen Client und Applikationsserver sowie zwischen Web-Server und Applikationsserver wird über einen HTTP-Basierten Service Tunnel abgewickelt. Die Kommunikation zwischen dem Webserver und den Web-Clients findet über einfaches HTTP statt. Betrachtet wird hier nur die Kommunikation zwischen Client und Applikationsserver, da nur die Server-Applikation skaliert werden soll.

Vom Client zum Applikationsserver

Die Kommunikation zwischen dem Applikationsserver und den Rich-Clients findet über eine SOAP Schnittstelle statt, welche als Service Tunnel bezeichnet wird. Sie ist prinzipiell zustandslos, es wird jedoch für jeden Client innerhalb des Servers eine Session angelegt, die im Folgenden als ServerSession bezeichnet wird. Diese ServerSession wird zur Identifizierung des Client und als Cache für Daten genutzt. Die ServerSession der Rich-Clients wird innerhalb der HTTP-Session abgelegt (Orab). Die ServerSession der Clients, die über den Web-Server mit dem Applikationsserver kommunizieren, werden innerhalb der Server-Applikation gecached. Die Kommunikation kann sowohl synchron als auch asynchron geschehen. Die Adresse des Applikationsservers wird dem Rich-Client in einer Konfigurationsdatei mitgeteilt.

Vom Applikationsserver zum Client

Mit der Richtung der Kommunikation zwischen dem Applikationsserver und den Rich-Clients ändert sich auch das Kommunikationsmuster, da der Server die Clients nicht direkt adressieren kann. Deshalb werden Nachrichten für die Clients auf dem Server gesammelt. Der Client ruft die Nachrichten bei jedem Service-Aufruf und in regelmäßigen zeitlichen Abständen vom Server ab.

2.1.4 Datenspeicherung innerhalb des Applikationsservers

Um die Performance zu erhöhen, werden innerhalb der Server-Applikation Daten zwischengespeichert, damit unnötige Zugriffe auf die Datenbank vermieden werden. Gibt es Änderungen an der Datenbasis werden die zwischengespeicherten Daten erneuert. Betroffen hiervon sind die HTTP-Session, die `ClientNotificationQueue`, der `CodeTypeStore` und der `AccessControlStore`. In anderen Komponenten werden ebenfalls Daten zwischengespeichert. Da im Rahmen dieser Masterthesis jedoch nur die Server-Applikation skaliert werden soll, wird auch nur diese hier betrachtet.

HTTP-Session

Das Eclipse Scout Framework nutzt die HTTP-Session, um darin die ServerSession abzulegen. Innerhalb der ServerSession werden Daten gecached, die dem Client zugeordnet sind. Zudem wird sie genutzt, um den Client zu identifizieren. Die HTTP-Session selbst ist ein Service, der von dem Servlet-Container angeboten wird, in dem die Applikation läuft. Das bedeutet, dass die Daten, die in der HTTP-Session gespeichert werden, innerhalb des Servlet-Containers abgelegt werden.

ClientNotificationQueue

Im vorhergehenden Abschnitt 2.1.3 wurden Nachrichten angesprochen, die vom der Server-Applikation an die Clients ausgeliefert werden. Diese Nachrichten werden ClientNotifications genannt und innerhalb der Klasse `ClientNotificationQueue` zusammen mit einem Filter und einer Liste mit Clients, an die die Nachricht bereits ausgeliefert wurde, in einer Queue gespeichert. Welche Klassen dabei involviert sind, ist in der Abbildung 2.5 zu sehen. Diese zeigt den `ClientNotificationService`, welcher über das Interface `IClientNotificationService` den Clients zugänglich gemacht wird. Der `ClientNotificationService` besitzt eine Instanz der `ClientNotificationQueue`. Diese wiederum hat eine Liste, in der Instanzen von `QueueElement` abgelegt werden. Jedes `QueueElement` wiederum enthält eine Instanz einer Klasse, die das Interface `IClientNotification` implementiert, eine Instanz einer Klasse, die das Interface `IClientNotificationFilter` implementiert und eine `WeakHashMap` in der Instanzen von Klassen des Interfaces IServerSession beinhaltet. Das bedeutet, dass zu jeder Nachricht, die an einen Client ausgeliefert werden soll, wird ein neues `QueueElement` in

der `ClientNotificationQueue` abgelegt. Innerhalb dieses `QueueElement` wird die Nachricht an sich abgelegt sowie ein Filter der bestimmt, an welche Clients die Nachricht ausgeliefert werden soll und wie lange die Nachricht gültig ist. Zudem kann in dem Filter festgelegt werden, ob die Nachricht nur an einen Client ausgeliefert werden soll oder an alle Clients. Um bei einer Nachricht, die an alle Clients ausgeliefert werden soll, feststellen zu können, an welche Clients die Nachricht bereits ausgeliefert wurde, werden in dem `QueueElement` innerhalb der `WeakHashMap` alle Server Sessions der Clients gespeichert, die die Nachricht schon abgerufen haben. Der Auswahlprozess, der entscheidet welche Nachrichten an einen Client ausgeliefert werden, ist in der Abbildung 2.6 zu sehen. Dieser Auswahlprozess wird bei jedem Aufruf der Methode `getNextNotifications` der `ClientNotificationQueue` ausgeführt. Dabei wird für jedes Element aus der internen Queue geprüft, ob dieses noch aktiv ist. Ist dies nicht der Fall, wird es aus der Queue gelöscht und mit dem nächsten Element weitergemacht. Ist das Element noch aktiv wird geschaut, ob die ClientNotification schon einmal an diesen Client ausgeliefert wurde. Wurde es bereits an diesen Client ausgeliefert, wird einfach mit dem nächsten Element fortgefahren. Wurde das Element an diesen Client noch nicht ausgeliefert, so wird geprüft, ob diese Element überhaupt für diesen Client bestimmt ist. Ist dies der Fall, wird das Element in einer Liste zur Auslieferung zwischengespeichert. Handelt es sich bei dem Element um eine ClientNotification, die für alle Clients gedacht ist, so wird die Auslieferung an diesen Client in dem Element vermerkt, andernfalls wird sie aus der Queue gelöscht und mit dem nächsten Element fortgefahren. Abschließend werden alle vorgemerkten Elemente an den Client übergeben.

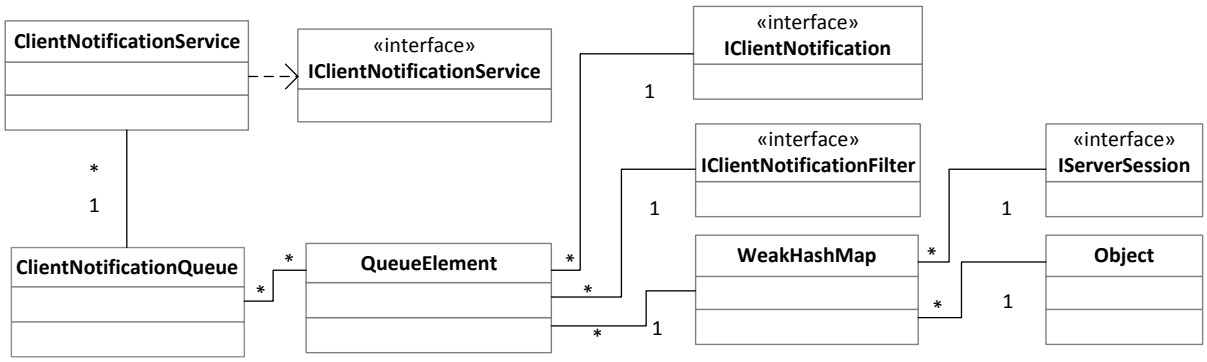

Abb. 2.5: UML - Client Notification Speicherung im Server

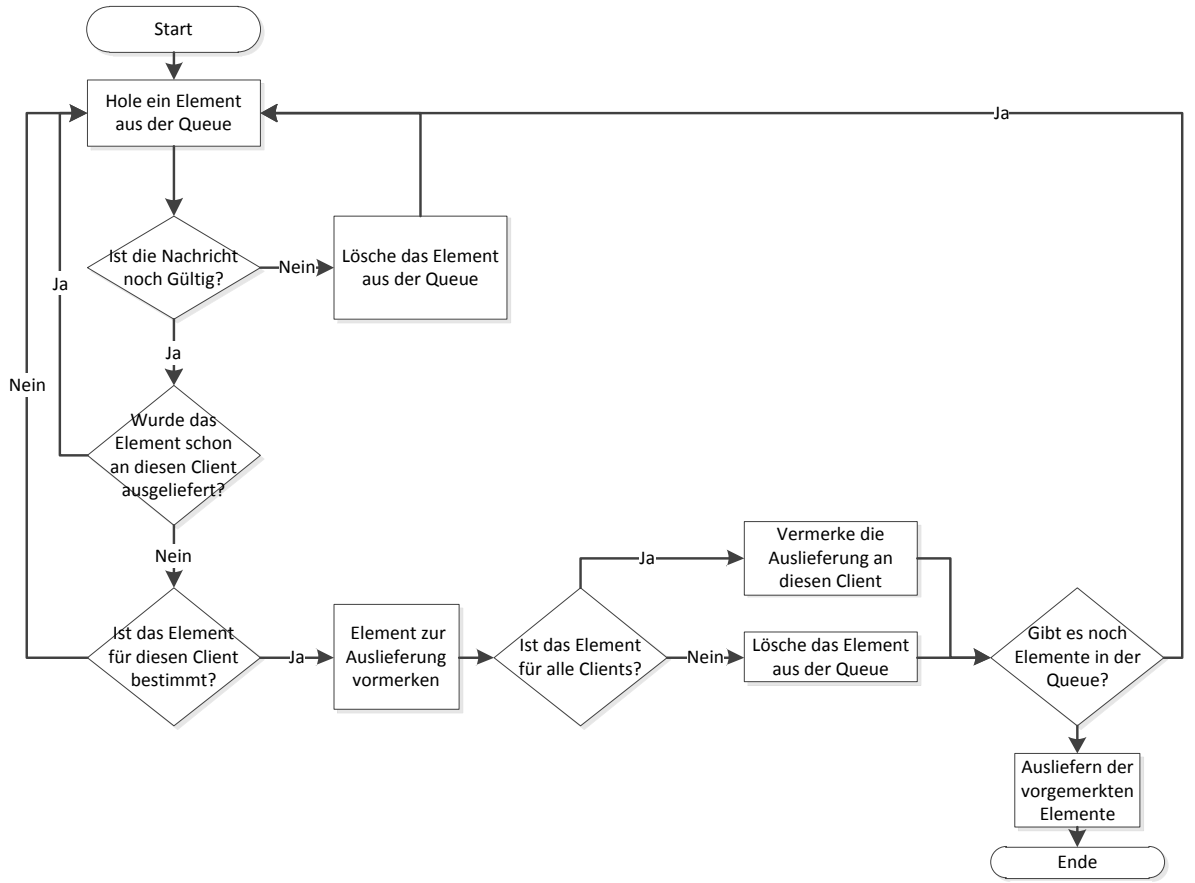

Abb. 2.6: Ablauf - Auswahl der auszuliefernden Nachrichten

CodeTypeStore

Der CodeTypeStore ist eine Klasse, in der CodeTypes gecached werden. CodeTypes werden in Eclipse Scout für SmartFields und SmartColumns verwendet. Ändert sich ein oder mehrere CodeTypes, werden diese neu in den Cache geladen und es wird eine ClientNotification an alle Clients versendet, um sie über die Änderung zu informieren.

AccessControlStore

Im AccessControlStore werden die Berechtigungen der Nutzer gecached. Der AccessControlStore bietet die Möglichkeit, die Berechtigungen entweder für alle oder für eine bestimmte Menge an Nutzern zu löschen. Nachdem der gesamte Cache oder einzelne Elemente daraus gelöscht wurden, wird eine ClientNotification an die betroffenen Clients geschickt, um diese über die geänderten Berechtigungen zu informieren.

2.2 Architektur von Cloud-Systemen

Im vorhergehenden Abschnitt wurde die Architektur von Eclipse Scout Applikationen und ihrer Bestandteile untersucht. Dieser Abschnitt gibt einen Überblick über die Eigenschaften von Cloud-Systemen und bestehenden Architektur-Modellen.

2.2.1 Merkmale der Cloud

Definitionen und Beschreibungen der Merkmale einer Cloud gibt es viele. So definiert Christian Braun zusammen mit seinen Co-Autoren die Cloud wie folgt:

> „Unter Ausnutzung virtualisierter Rechen- und Speicherressourcen und moderner Web-Technologien stellt Cloud Computing skalierbare, netzwerk-zentrierte, abstrahierte IT-Infrastrukturen, Plattformen und Anwendungen als on-demand Dienste zur Verfügung. Die Abrechnung dieser Dienste erfolgt nutzungsabhängig.“ (Bau11, S. 4)

Borko Furth beschreibt die Entwicklung von Computer Systemen in Phasen. Demnach sieht er Cloud Systeme als Nachfolger von Grid Systemen, die gegenüber ihrem Vorgänger geteilte Ressourcen einfach zugänglich und skalierbar über das Internet bereitstellen. (vgl. FE10, S. 3-4). Auch Nikolaos Preve beschreibt in seinem Buch „Grid Computing“ Cloud-Systeme als die nächste Generation der verteilten Systeme (vgl. Pre11, S. 174)

Meist wird jedoch die Definition des National Institute of Standards and Technology (NIST) verwendet:

> „Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.“ (MG11, S. 2)

Dazu hat das NIST fünf wesentliche Eigenschaften, drei Service-Ebenen und vier Bereitstellungsmodelle definiert, die im Folgenden näher beschrieben werden. (vgl. MG11, S. 2)

Wesentliche Eigenschaften

Die vom NIST definierten Eigenschaften einer Cloud sind On-demand self-service, Broad network access, Resource pooling, Rapid elasticity und Measured service. Diese wurden in dem Buch „Cloud Computing - Web-basierte dynamische IT-Services“ übersetzt und wie folgt beschrieben:

> „Diensterbringung auf Anforderung: Dienste sind auf Anforderung und selbständig von Konsumenten ohne erforderliche menschliche Interaktion mit dem Anbieter nutzbar.
>
> Netzwerkbasierter Zugang: Dienste können netzwerkbasiert in Echtheit durch Verwendung von Standardtechnologien abgerufen werden.
>
> Ressourcen Pooling: Ressourcen sind in Pools konsolidiert und erlauben eine parallele Diensterbringung für mehrere Nutzer (Mandanten), die dem tatsächlichen Bedarf eines jeden Nutzers angepasst ist.
>
> Elastizität: Ressourcen werden schnell und in verschiedenen, auch fein granularen Quantitäten, zur Verfügung gestellt und erlauben so die Skalierung von Systemen. Dem Nutzer gegenüber entsteht die Illusion unendlich verfügbarer Ressourcen.
>
> Messbare Dienstqualität: Dienste sind quantitativ und qualitativ messbar, so dass eine nutzungsabhängige Abrechnung und Validierung der Dienstqualität gegeben ist.“ (Bau11, S. 5-6)

Im wesentlichen lässt sich also sagen, dass Cloud-Dienste voll automatisiert über standardisierte Netzwerke in Echtzeit angeboten werden. Dabei sind die Dienste immer für eine Vielzahl an Kunden ausgelegt und können deren Bedürfnissen angepasst werden. Die verwendeten Ressourcen lassen sich einfach skalieren und bieten eine messbare Dienstqualität, sodass eine nutzungsabhängige Abrechnung möglich ist.

Von diesen Eigenschaften sind das Ressourcen Pooling und die Elastizität für diese Masterthesis besonders wichtig, denn aus Ihnen lassen sich die Anforderungen an Skalierung und Lastverteilung ableiten.

Skalierung & Lastverteilung

Eine wesentliche Eigenschaft von Cloud-Systemen ist, wie im vorhergehenden Abschnitt beschrieben, dass sie je nach Bedarf skalieren. Damit werden zwei Ziele verfolgt. Zum einen ist es so möglich auf eine ansteigende Last zu reagieren und die Performance der Applikation auf einem gleich bleibenden Level zu halten. Zum anderen werden bei abnehmender Last weniger Ressourcen in Anspruch genommen, was die Betriebskosten senkt. Bei der Art der Skalierung werden zwei verschiedene Formen unterschieden. Bei der vertikalen Skalierung werden einzelnen Instanzen mehr Ressourcen zur Verfügung gestellt. Das heißt, die Server bzw. die virtuellen Maschinen bekommen mehr Ressourcen, meist in Form von Prozessorleistung und/oder Arbeitsspeicher, zugeteilt. Somit können die Anfragen schneller verarbeitet werden. Bei der horizontalen Skalierung werden einzelne Instanzen geklont und parallel betrieben. So lassen sich die Anfragen und damit die Last auf mehrere Instanzen verteilen. Der große Vorteil der vertikalen Skalierung gegenüber der horizontalen Skalierung ist, dass meist keine Änderungen an der Applikation vorgekommen werden müssen. Der Nachteil ist, dass der Skalierungseffekt durch technische Faktoren, wie die maximale Anzahl an Prozessoren oder Arbeitsspeicher, begrenzt ist. Der horizontalen Skalierung sind hingegen keine technischen Grenzen gesetzt. Um bei der horizontalen Skalierung die Anfragen, die von den Client kommen auf die einzelnen Server-Instanzen verteilen zu können, sind Systeme zur Lastverteilung erforderlich. Diese werden Load-Balancer genannt. Load-Balancer lassen sich in ihrem Verhalten, Anfragen zu verteilten, grob in zwei Kategorien einteilen. Diese sind Load-Balancer mit Sticky-Sessions und Load-Balancer ohne Sticky-Sessions (TTC+03). Bei Load-Balancern mit Sticky-Session wird die Anfrage eines Clients immer an den Server weitergeleitet, der auch die initiale Anfrage bearbeitet hat. Das hat den Vorteil, dass der Webserver Daten zu dem Client speichern kann und diese auch bei jeder Anfrage des Clients auch Verfügbar sind, da sich der bearbeitende Server nicht ändert. Der Nachteil ist, dass bei Ausfall oder Abschaltung des Servers auch die gespeicherten Daten verloren gehen. Anders ist es bei Load-Balancern ohne Sticky-Session. Diese verteilen die Anfragen der Clients nach festgelegten Algorithmen. Zum Beispiel nach Auslastung der Server. Der Nachteil dieser Methode ist, dass keinerlei Daten innerhalb der Applikationsserver gespeichert werden dürfen, die relevant für die Beantwortung von Anfragen der Clients sind, da ungewiss ist, welche Instanz die Anfrage bearbeitet. Der Vorteil ist, wenn keine relevanten Daten innerhalb einer Instanz gespeichert werden, können Server-Instanzen nach belieben hoch- und heruntergefahren werden, ohne dass Auswirkung bei Bearbeitung von Clientanfragen auftreten.

Service-Ebenen

Cloud-Angebote werden üblicherweise in drei Service-Ebenen unterteilt (vgl. Mah13, S. 48, vgl. VRMCL08, S. 51). Diese Unterteilung wird auch von der Bitkom (vgl. Inf09, S. 22) und des NIST (vgl. MG11) verwendet. Wie sich die drei Service-Ebenen in Bezug zu IT-Leistungen und Zielgruppe verhalten, ist in der Abbildung 2.7 zu sehen.

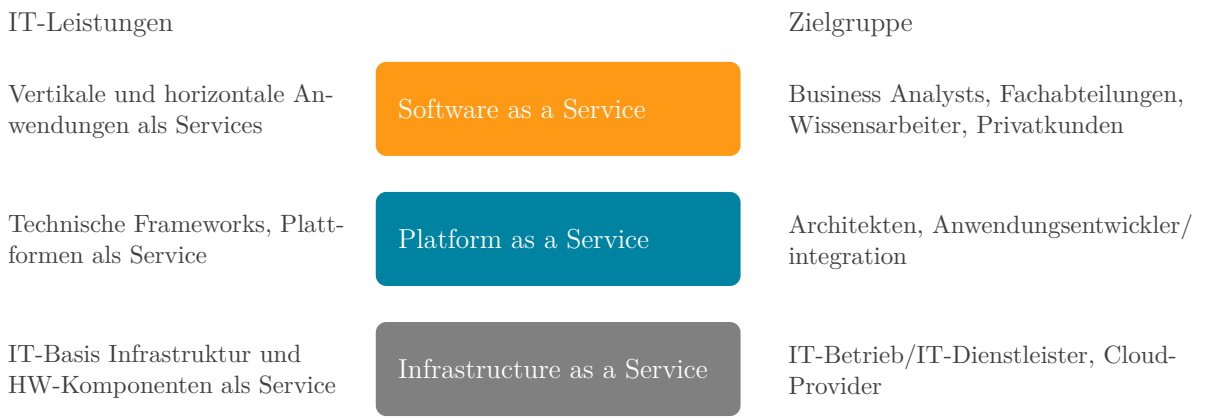

Abb. 2.7: Ebenen von Cloud Services, Quelle: (vgl. Inf09, S. 23)

Infrastructure as a Service (IaaS) beinhaltet die Bereitstellung von Rechen- , Speicher- und Netzwerk-Infrastruktur (vgl. Inf09, S. 22). Diese Ressourcen werden an die Kunden, je nach Anbieter, mehr oder weniger verbrauchsabhängig vermietet. Dabei hilft Virtualisierung physische Ressourcen zu bündeln oder zu teilen (vgl. DWC10, S. 28).

Platform as a Service (PaaS) bietet eine Entwicklungsumgebung. Diese ist je nach Anbieter mit verschiedenen Diensten ausgestattet. Diese Dienste unterstützen beim Entwickeln, Installieren, Testen oder Betrieb von Applikationen. (vgl. TSB10, S. 684). Weitere Merkmale einer PaaS Cloud sind die Unterstützung bei der Skalierung und Lastverteilung sowie das Angebot weiterer Dienste wie z. B. zur Speicherung von Daten (vgl. Kep12, S. 8).

Software as a Service (SaaS) stellt dem Endkunden Software über das Internet zu Verfügung, sodass diese nicht mehr lokal auf dem eigenen Computer installiert werden muss. (vgl. Bau11, S. 37) Das hat den Vorteil, dass die Software schnell Verfügbar ist, keine Arbeit für Wartung entsteht und die Kosten meist geringer sind als bei herkömmlicher Software (vgl. Mah13, S. 50).

Deployment Model

Das Deployment Model einer Cloud gibt an, wo sich die zur Bereitstellung des Cloud-Service verwendeten Ressourcen physisch befinden. Um zu beschreiben, unter welchem Deployment Model eine Cloud-Lösung betrieben wird, wurden die vier Begriffe Public, Private, Hybrid und Community Cloud definiert. (vgl. MG11, S. 3, vgl. Mah13, S. 51).

Private Cloud Die Private Cloud wird innerhalb des Unternehmens betrieben, welches die Cloud-Lösung nutzen möchte. Für das Unternehmen hat diese Lösung gleich mehrere Vorteile. Zum einen kann die Bereitstellung von Infrastruktur erheblich optimiert und Kosten eingespart werden. Zum anderen können die unternehmensinternen Kunden sicher sein, dass ihre Daten das Unternehmen nicht verlassen und somit auch den Sicherheitsrichtlinien des Unternehmens genügen (vgl. DWC10, S. 2).

Public Cloud Die Public Cloud ist ein Angebot eines frei zugänglichen Anbieters. Dieser macht seine Dienste offen über das Internet für jedermann zugänglich (vgl. Fra). Hier wird die Kontrolle über die zugrunde liegende Infrastruktur komplett abgegeben und der Kunde hat sich an die Richtlinien und Preise des Anbieters zu richten. Sie kann, im Gegensatz zu der Private Cloud, von einer Vielzahl von Kunden genutzt werden. Aus diesem Grund ist die Public Cloud auch das am weiten verbreitetste Deployment Model.

Community Cloud Die Community Cloud ist ein Zusammenschluss von Ressourcen befreundeter Organisationen zu einer gemeinsam genutzten Cloud. Dieses Deployment Model bietet sich besonders für die Unternehmen an, die ähnliche Anforderungen und Aufgaben an die an die Cloud stellen. Mit diesem Deployment Model ist es möglich, ähnliche Vorteile wie in der Public Cloud zu genießen und dabei einen Großteil der Kontrolle zu behalten.

Hybrid Cloud Die Hybrid Cloud ist laut NIST eine Mischung aus Private, Public oder Community Cloud (vgl. MG11, S. 3), dargestellt in der Abbildung 2.8. So können die Vorteile der einzelnen Deployment Models verknüpft genutzt werden. Sicherheitskritische Anwendungen können herbei im privaten Teil betrieben werden, Anwendungen die nicht sicherheitsrelevant sind können im Public Teil betrieben werden.

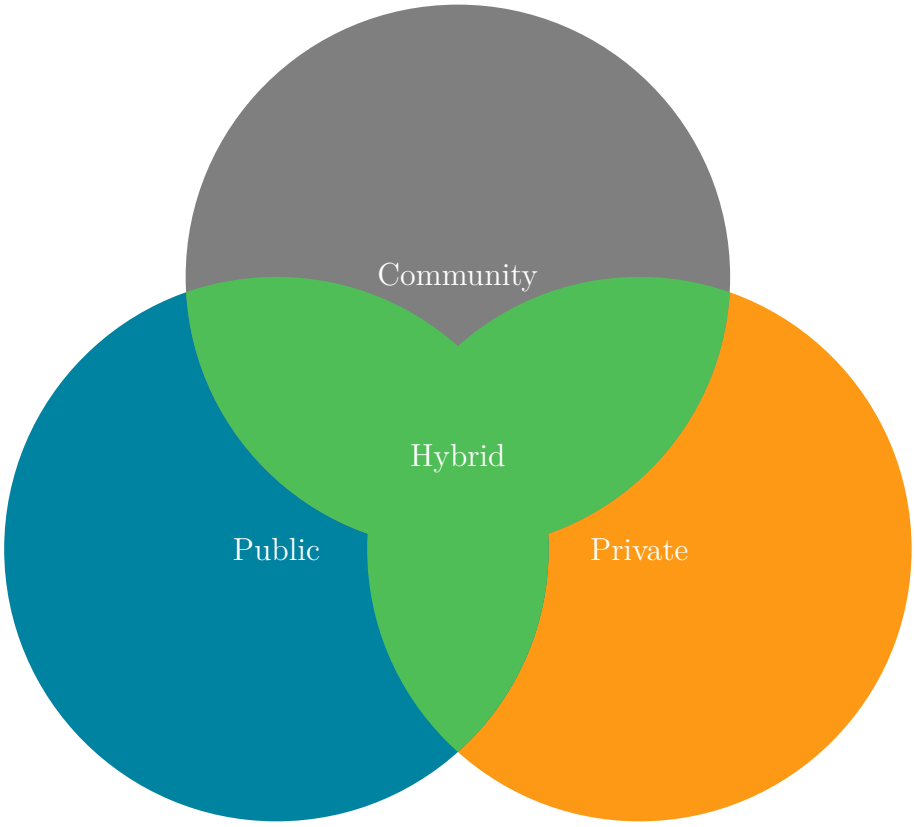

Abb. 2.8: Cloud Deployment Modelle

Services

Services sind Dienste, die innerhalb einer Cloud genutzt werden können. Welche Dienste genutzt werden können hängt stark vom Cloud Anbieter, der Service Ebene und dem Deployment Model ab. Bei der Nutzung einer IaaS-Cloud können neben der vom Cloud Anbieter bereitgestellten Dienste auch eigene Dienste auf der gemieteten Infrastruktur eingerichtet werden. Innerhalb einer PaaS Cloud obliegt es dem Cloud Anbieter, welche Dienste er anbietet. Bezüglich des Deployment Model ist es in einer Private Cloud eher möglich weitere Dienste zu erhalten, als in einer Public Cloud, da die Kontrolle über die Infrastruktur im eigenen Unternehmen liegt.

Einige Dienste werden von Drittanbietern als SaaS Angebot bereitgestellt. Bei der Nutzung dieser Dienste spielt das Hosting eine wichtige Rolle, da die Kommunikation bzw. der Datenverkehr aus einer Cloud heraus meist reglementiert und kostenpflichtig ist. Aus diesem Grund ist es von Vorteil, wenn der Anbieter des Dienstes sein SaaS Angebot in der selben Cloud betreibt, wie der Nutzer.

2.2.2 Typische Architekturen

Nachdem der vorhergehende Abschnitt 2.2.1 die Merkmale einer Cloud beleuchtet hat, werden in diesem Abschnitt Architekturen typischer Cloud-Applikationen betrachtet, um für die Erweiterung des Eclipse Scout Framework bewährte Ansätze mit einfließen zu lassen.

Web-Applikationen

Einfache Web-Applikationen sind wohl die am weitest verbreitetsten Applikationen, die in einer Cloud betrieben werden. Trieu C. Chieu beschreibt eine typische Architektur von Web-Applikationen, wie sie in der Abbildung 2.9 zu sehen ist (vgl. CMKS09, S. 284). Das heißt, er sieht eine beliebige Anzahl an Webserver, die hinter einem Load Balancer platziert werden, welcher dann die Kommunikation mit den Clients regelt. Zudem wird typischerweise eine Datenbank von den Webservern genutzt. Ferner beschreibt er die Präsenz eines Service Monitors, der die Auslastung der Webserver im Blick behält und diese an den Load Balance weitergibt, welcher über einen Skalierungsalgorithmus weitere Maßnahmen bestimmt. Änderungen werden an die Provisionierung weitergegeben, um eine bedarfsabhängige Abrechnung zu erstellen.

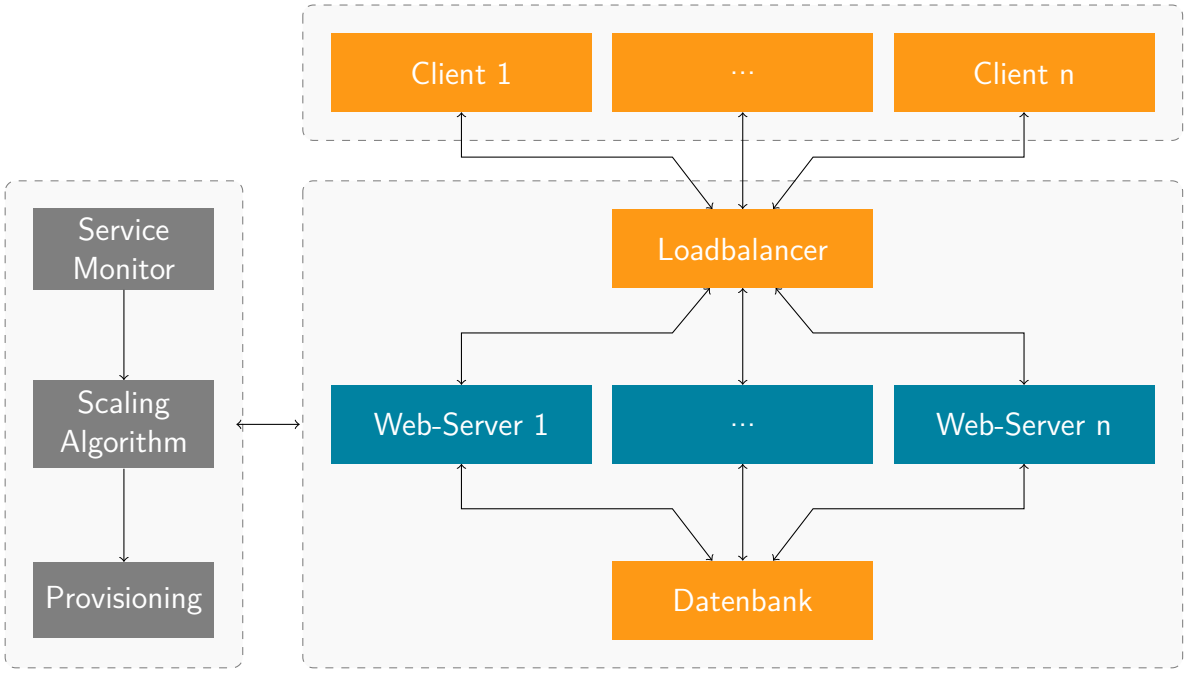

Abb. 2.9: Architektur typischer PaaS Applikation, Quelle: (vgl. CMKS09, S. 284)

OSGi in der Cloud

Mit dem RFC 183 (All13) verfolgt die OSGi Alliance einen ganz anderen Ansatz, um Applikationen in der Cloud skalierbar zu machen. Unter dem Namen OSGi Cloud Ecosystem wird eine Architektur beschrieben, bei der nicht die ganze Applikation auf mehrere Server-Instanzen verteilt, sondern der modulare Aufbau von OSGi Applikationen genutzt wird, um die einzelnen Bundles auf mehrere Server-Instanzen zu verteilen. einen möglichen Aufbau dieser Architektur hat David Bosschaert auf der EclipseCon 2013 vorgestellt (Dav10). Dieser ist in der Abbildung 2.10 zu sehen. Hierfür werden die in der Cloud laufenden Server-Instanzen mit dem OSGi Framework ausgestattet und ihnen eine Funktion zugeordnet. Eine Instanz bekommt dabei die Aufgabe, die zur Verfügung stehenden Bundles aus dem Bundle Repository an die anderen Instanzen zu verteilen, in der Grafik als Provisioner bezeichnet. Die restlichen Server-Instanzen erhalten je nach Applikation, die sie ausführen sollen, Bundles zugewiesen.

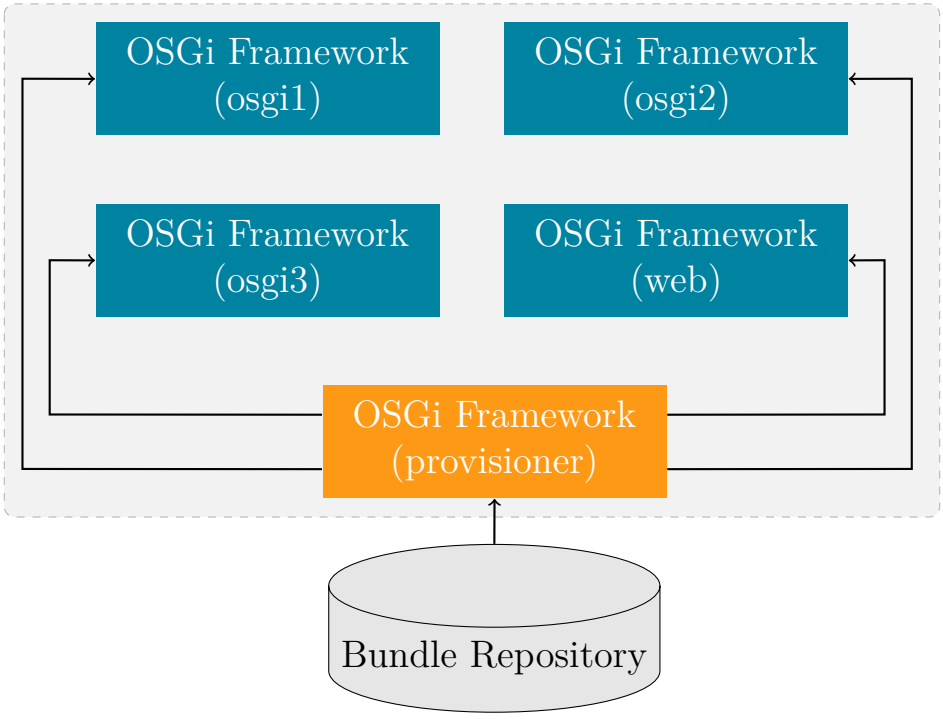

Abb. 2.10: Architektur OSGi Cloud Ecosystem, Quelle: Vgl. (Dav10)

Diese Architektur ist für Eclipse Scout Applikationen jedoch nicht geeignet, da das Eclipse Scout Framework zu wenig Bundles besitzt, um bei der Verteilung der Bundles entstehenden Skalierungseffekte nutzen zu können.

2.2.3 Abgrenzung Legacy Migration

Zu der Migration von Legacy Anwendungen in die Cloud, gibt es bereits verschiedene Ansätze. Die Herausforderung dieser Masterthesis ist jedoch, ein Framework so zu erweitern, dass die damit erstellten Applikationen ohne großen Aufwand in der Cloud betrieben werden können. Dennoch lohnt es sich einen Blick auf die bestehenden Ansätze zur Migration von Legacy Anwendungen zu werfen, da hier ähnliche Ziele verfolgt werden. Der IT Analyst Gartner hat für die Migration von Legacy Anwendungen fünf Ansätze erarbeitet, die bei der Migration helfen sollen (Woo11, vgl.).

Rehost

Mit Rehost ist eine einfache Verschiebung des Hostings in die Cloud gemeint. Hierbei wird die Applikation ohne Anpassungen in eine IaaS Cloud umgezogen. Bis auf die zugrunde liegende Hardware und eventuell das Betriebssystem bleibt der übrige Applikationsstack unverändert bestehen.

Replace

Die alte Applikation zu verwerfen und mit einer als SaaS Lösung zu ersetzen, wird von Gartner unter dem Begriff Replace verstanden. Von der alten Applikation werden ausschließlich die Daten in das neue System migriert.

Refactor

Beim Refactoring wird die Applikation soweit angepasst, dass diese in einer PaaS Lösung betrieben werden kann. Dabei kann es notwendig sein, andere Frameworks oder Sprachen einzusetzen. Die Applikationskonfiguration muss angepasst und die Datenbasis auf das neue System migriert werden.

Rebuild

Die Methode Rebuild ist der Extremfall des Refactoring. Hierbei wird die komplette Applikation neu implementiert. Bei der Implementierung wird dann darauf geachtet, dass

die Applikation auf der gewünschten PaaS Cloud betrieben werden kann. Als Risiko sieht Gartner hier die Festlegung auf einen Anbieter.

Revsise

Unter Revsise versteht Gartner eine Modernisierung der Applikation, um diese portierbar zu machen und anschließend per Refactor oder Rehost in die Cloud zu verschieben. Diese Methode lohnt sich jedoch nur selten. Die Vorteile, dieser meist kostenintensive Umstellung der Applikation, können laut Gartner oft nicht genutzt werden.

Zusammenfassende Betrachtung

Lucas Carlson, der Gründer der AppFog PaaS, schreibt in seinem Buch Programming for PaaS über die für die PaaS Migration relevanten Methoden Refactor und Rebuild (Car13). Dabei mach er sich Gedanken über die Probleme, die bei einer Migration einer Legacy Anwendung auftreten können. Die von Ihm angesprochenen Punkte sind:

- Datei Hosting
- Sessions Management
- Caching
- Asynchrone Prozesse
- SQL
- NoSQL

Interessant im Bezug auf die Anpassung des Eclipse Scout Frameworks sind hier die Lösungen für das Session Management. An dieser Stelle schlägt Herr Carlson vor, die Session verschlüsselt in einem Cookie, in einer SQL oder NoSQL Datenbank abzulegen.

Von diesen fünf Methoden ist die Methode Reviese diejenige, die am ehesten auf das Zutrifft, was das Ziel dieser Masterthesis angeht, da hier die notwendige Portierbarkeit enthalten ist. Der Aufwand hält sich hier in Grenzen, da nur das Framework angepasst wird und keine ganze Applikation. Zudem kann hier der Vorteil der Portierbarkeit voll ausgenutzt werden, um die mit dem Framework erstellten Applikationen nicht auf einen Anbieter festzulegen.

2.3 Architektur-Anpassungen

Die vorhergehenden Abschnitte haben gezeigt wie Eclipse Scout Applikationen aufgebaut sind und welche Eigenschaften die einzelnen Komponenten haben. Zudem wurden die Eigenschaften von Cloud-Systemen untersucht und gängige cloudfähige Architekturen beschrieben. Mit dem Ziel dieser Masterthesis, Eclipse Scout Applikationen skalierbar auf mehreren PaaS Angeboten betreiben zu können, sind Anpassungen am Eclipse Scout Framework notwendig. Diese werden im Folgenden beschrieben.

2.3.1 Skalierung

Die Grundprinzipien der Skalierung und der damit verbunden Lastverteilung wurden im Abschnitt 2.2.1 bereits erläutert. Um eine uneingeschränkte Skalierung der Applikationen zu gewährleisten, die mit dem Eclipse Scout Framework erstellt wurden, wird eine horizontale Skalierung der Server-Applikation eingesetzt. Dabei soll die Applikation fähig sein ohne einen Load Balancer mit Sticky Sessions korrekt zu arbeiten. Dem entgegen steht die in den Abschnitten 2.1.3 und 2.1.4 angesprochene Datenhaltung innerhalb des Applikationsservers. Laut dem Buch „Cloud Computing Patterns“ gilt es bei der Skalierung von Komponenten mit Datenhaltung die Daten über alle Server-Instanzen zu synchronisieren (vgl. FR11, S 168). Die Daten die innerhalb des Applikationsservers gespeichert werden, wurden im Abschnitt 2.1.4 aufgelistet. Die durch die Skalierung entstehenden Probleme der gespeicherten Daten werden im Folgenden beleuchtet.

HTTP-Session

Da die HTTP-Session innerhalb des Servlet-Containers liegt und jede Server-Instanz ihren eigenen Servlet-Container besitzt, ist es notwendig die zuvor in der HTTP-Session abgelegten Daten über alle Server-Instanzen zu synchronisieren. Hierfür kommen ein gemeinsames Dateisystem, eine Datenbank oder ein verteilter Cache in Frage. Aus Gründen der Performance, soll hier der verteilten Cache eingesetzt werden.

Neben der Funktion des Caches hat die HTTP-Session auch die Funktion der Identifikation des Clients erfüllt. Diese kann die HTTP-Session beim Einsatz mehrerer Server-Instanzen auch nicht mehr erfüllen, denn bei der Betrachtung der Verarbeitung von HTTP-Sessions innerhalb des Servlet-Container fällt auf, dass die Zuordnung der HTTP-Session über

ein Cookie mit dem Namen JSESSIONID gehandhabt wird, welches beim Client gespeichert wird. Dieses Cookie wird von dem Servlet-Container jedes mal gesetzt, wenn auf die HTTP-Session zugegriffen wird und der Client noch kein Cookie mit dem Namen JSESSIONID hat, oder die gefundene JSESSIONID nicht in der internen Ablage gefunden wird. Wird einen Load-Balancer ohne Sticky-Sessions eingesetzt, so kommt der Client bei jeder Anfrage auf eine andere Server-Instanz und erhält damit bei jedem Aufruf eine neue JSESSIONID. Daraus wiederum folgt, dass zuvor in der HTTP-Session gespeicherte Daten nicht mehr zugeordnet werden können und somit verloren gehen. Die Lösung ist hier ein eigenes Cookie zu setzen, dass den Client identifiziert.

ClientNotificationQueue

Im Abschnitt 2.1.3 wurde bereits beschrieben, dass zur Kommunikation zwischen dem Applikationsserver und der Clients ClientNotifications in einer Queue auf dem Server abgelegt werden. Beim Einsatz mehrerer Server-Instanzen kann es dazu kommen, dass sich ein Client über längere Zeit nicht auf eine bestimmte Server-Instanz verbindet und die ClientNotifications deshalb nicht ausgeliefert werden können. Daher ist es erforderlich, dass alle ClientNotifications zwischen allen Server-Instanzen synchronisiert werden. Da alle Instanzen beim jedem Aufruf eines Clients auf diese Queue zugreifen müssen, macht eine zentralisierte Ablage in einem verteilten Cache hier keinen Sinn. Deshalb bleibt in jeder Server-Instanz die lokale Queue bestehen und es werden nur die Änderung an dieser Queue den anderen Server-Instanzen mitgeteilt. Hierfür soll eine Message Queue eingesetzt werden von der die Server-Instanzen über der Kommunikationsmuster Publish/Subscribe ihre Nachrichten erhalten. Damit ist es möglich eine asynchrone Kommunikation zwischen den Server-Instanzen zu gewährleisten ohne, dass diese sich gegenseitig kennen.

CodeTypeStore & AccessControlStore

Eine weitere Herausforderung ist die im Abschnitt 2.1.4 abgesprochene Datenhaltung innerhalb des `CodeTypeStore` und dem `AccessControlStore`. In diesen werden Daten aus der Datenbank gecached. Nach einer Änderung an den Daten werden diese aus der Datenbank aktualisiert und anschließend die Clients über die Änderung benachrichtigt. Hier besteht die Möglichkeit die Daten entweder ähnlich wie die ServerSession im synchronisierten Cache abzulegen oder analog zu den ClientNotifications nur die Änderungen zwischen

den Server-Instanzen zu kommunizieren. Beide Lösungen können mit den zuvor angesprochenen Diensten, wie dem verteilten Cache oder der Message Queue, umgesetzt werden. Das heißt, für diese Anforderung sind keine gesonderten Änderungen an der Architektur notwendig. Wie die Synchronisation der gecachten Daten schlussendlich implementiert wird, ist im Abschnitt 4.1 beschrieben.

Resultierende Architektur

Die bezüglich der Skalierung notwendigen Änderungen an der Architektur von Eclipse Scout Applikationen sind in der Abbildung 2.11 zu sehen. Diese beinhaltet die angesprochenen Änderungen gegenüber der bisherigen Architektur aus der Abbildung 2.1. Neu hinzugekommen ist hier die Möglichkeit, beliebig viele Applikationsserver zu betrieben. Diese müssen hinter einem Load Balancer platziert werden, damit alle Clients einen einheitlichen, gleich bleibenden Kommunikationspunkt haben. Weiterhin sind die beiden zusätzlichen Systeme, den verteilten Cache und die Message Queue, neu in die Architektur aufgenommen worden.

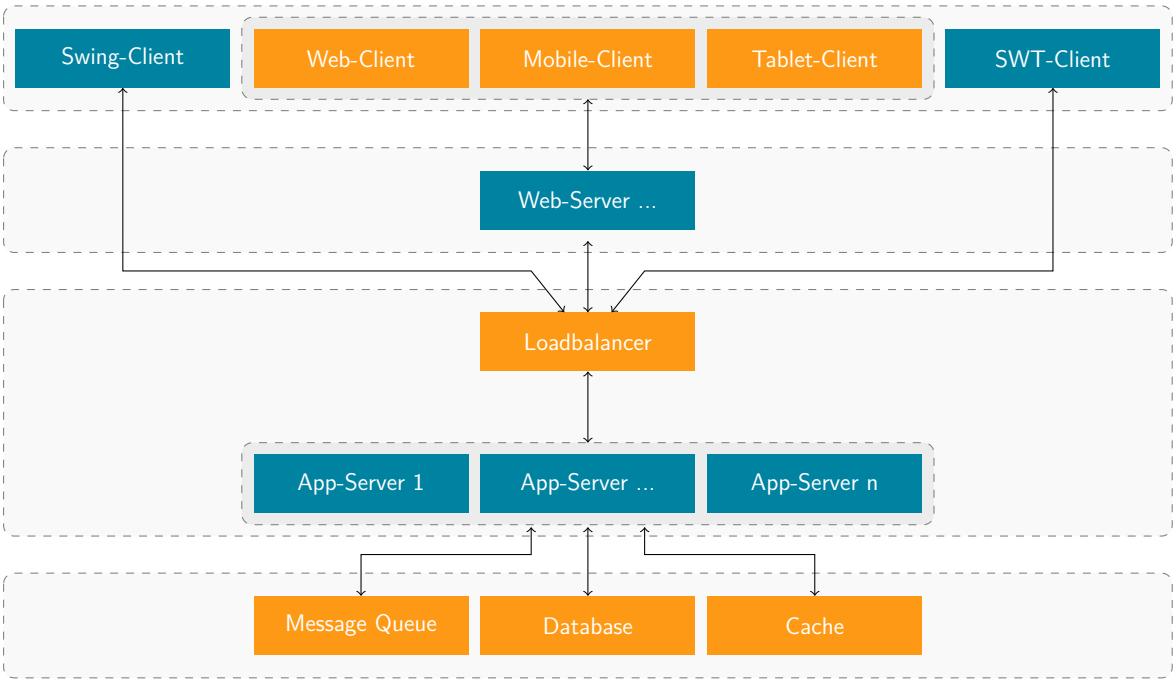

Abb. 2.11: Angepasste Architektur einer Eclipse Scout Applikation

2.3.2 Lose Kopplung

Neben der Skalierbarkeit der mit dem Eclipse Scout Framework erstellten Applikationen soll auch eine einfache Portierung auf verschiedene PaaS Clouds möglich sein. Diese Fähigkeit halte ich für sehr wichtig, da die Entwicklungen in einem sich so schnell veränderndem Markt, wie dem der Cloud, nur sehr schwer abzuschätzen sind. Das Eclipse Scout Framework verfolgt eine Langzeitstrategie. Aus diesem Grund ist es umso wichtiger den Entwicklern eine Möglichkeit zu geben die Vorteile von PaaS Cloud zu nutzen ohne sich an einen bestimmten Anbieter zu binden. Hier hilft die serviceorientierte und modulare Architektur die durch das Eclipse Scout Framework bereitgestellt wird. Wie im Abschnitt 2.1.2 beschrieben wurde, besteht die Server-Applikation aus mehreren Bundles, die ihre Services in der OSGi Service Registy registrieren. Genau dieses Konstrukt hilft hier die Kopplung zwischen den einzelnen PaaS Angeboten und dem Eclipse Scout Framework möglichst gering zu halten. Die Anbindung des verteilten Caches und der Message Queue werden nicht direkt im Eclipse Scout Framework implementiert, sondern in eigenen Bundles, die je nach PaaS Anbieter der Applikation hinzugefügt oder ausgetauscht werden können. Hierfür werden, wie in der Abbildung 2.12 zu sehen, im Scout RT Server Bundle zwei neue Services definiert die Applikation für den Single-Node-Betrieb einsetzten. Soll die Applikation skaliert werden, können einfach zwei neue Bundles hinzugefügt werden, die diese Services überschreiben.

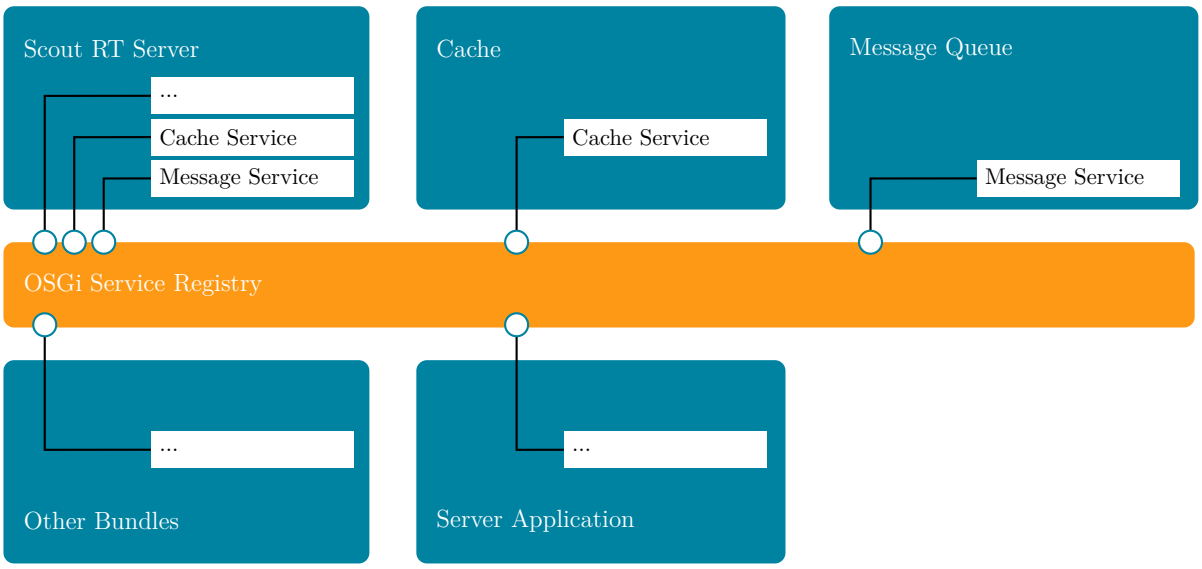

Abb. 2.12: Bundles der Serverapplikation

3 PaaS Anforderungen und Anbieter

Das vorhergehende Kapitel 2 hat die Architektur von Eclipse Scout Applikationen und die Eigenschaften ihrer Komponenten untersucht. Weiterhin wurden die Eigenschaften von Cloud-Systemen beschrieben. Daraus wurde eine Architektur entworfen, die es möglich macht, Applikationen, die mit dem Eclipse Scout Framework erstellt wurden, skalierbar in verschiedenen PaaS Clouds zu betreiben. Dieses Kapitel beschäftigt sich mit den Anforderungen, die aus der angepassten und bestehenden Architektur sowie den im Kapitel 1 gesetzten Ziele entstehen. Anhand dieser Anforderungen werden ausgewählte PaaS Anbieter untersucht und bewertet. Abschließend werden zwei PaaS Anbieter ausgewählt, für die im darauffolgenden Kapitel 4 Implementierungen erstellt werden.

3.1 Anforderungen an die PaaS Cloud

Wie im Abschnitt 1.2 beschrieben, ist es das Ziel dieser Arbeit Applikationen, die mit Eclipse Scout erstellt wurden, skalierbar auf PaaS Angeboten ausführbar zu machen. Im Abschnitt 2.1 wurde die bisherige Architektur von Eclipse Scout Applikationen beschrieben und im Abschnitt 2.2 die Architektur von Cloud-Systemen. Dieser Abschnitt beschäftigt sich mit den Anforderungen, die an die PaaS Cloud gestellt werden, unter Berücksichtigung der im Abschnitt 2.1 und 2.2 beschriebenen Architektur Voraussetzungen und der im Abschnitt 1.2 gesetzten Ziele. Bei den Anforderungen wird zwischen harten Anforderungen, die zwingend erfüllt sein müssen und weichen, optionalen Anforderungen unterschieden.

3.1.1 PaaS Form

Unter dem Begriff PaaS werden viele verschiedene Lösungen angeboten. Das NIST definiert die PaaS Cloud wie folgt:

> "The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment." (MG11)

Damit meint das NIST demnach Angebote, die es dem Kunden ermöglichen, selbst erstellte oder erworbene Applikationen auf Basis vom Anbieter unterstützten Programmiersprachen, Bibliotheken und Services zu betreiben. Hierbei hat der Kunde keine Kontrolle über die zugrunde liegende Infrastruktur. Diese recht weit gefasste Definition lässt es zu, dass unter dem Begriff PaaS, neben den klassischen gehosteten Cloud Angeboten, mit ihren verschiedenen Deployment Modellen auch Angebote fallen, die lediglich einen Container bereitstellen, der alles enthält was die Applikation benötigt und selbst anschließend in einer Cloud Infrastruktur betrieben wird. Ein Beispiel hierfür ist Docker. Eines der im Abschnitt 1.2 definierten Ziele dieser Masterthesis ist, dass die mit Eclipse Scout erstellten Applikationen ohne großen Aufwand in einer PaaS Cloud betrieben werden können. Meiner Meinung nach kann dieses Ziel nur unter Verwendung einer Public PaaS Cloud erreicht werden, bei der sich der Anbieter auch um das Hosting kümmert, da bei allen anderen Lösungen durch die fehlende Plattform oder Infrastruktur hohe Installationsaufwände entstehen. Ob der PaaS Anbieter seine eigene Infrastruktur oder die eines anderen Cloud Anbieters nutzt, ist hierbei nicht von Belang. Aus diesen Prämissen ergibt sich die harte Anforderung für ein Public PaaS Angebot mit Hosting.

3.1.2 Java

Das Eclipse Scout Framework basiert auf Java, ebenso wie die mit Eclipse Scout erstellten Applikationen. Das macht es erforderlich, dass die PaaS Lösung ebenfalls Java unterstützt. Das Eclipse Scout Framework setzt die Java-Version 1.6 voraus. Empfohlen wird jedoch die Version 1.7 (vgl. Ecl). Die in dieser Masterthesis genutzte Referenz-Applikation BahBahChat setzt die Java-Version 1.7 voraus. Ein weiterer Anspruch an die von der PaaS bereitgestellte Java-Installation ist der volle Zugriff auf alle Java-Klassen. Dieser wird unter anderem benötigt, um die von Eclipse Scout Applikationen verwendete OSGi Implementierung Equinox starten zu können.

3.1.3 Servlet Container

Neben der Unterstützung von Java ist es für den Betrieb einer Eclipse Scout Server-Applikation notwendig, innerhalb der PaaS Cloud einen Servlet-Container zur Verfügung zu haben, in dem die Applikation betrieben werden kann. Hierbei muss der Servlet Container in der Lage sein, mit Servlets in der Version 2.5 umgehen zu können. Wünschenswert ist, dass der PaaS Anbieter einen Servlet Container ohne weiteren Installationsaufwand bereitstellt.

3.1.4 Skalierung und Lastverteilung

Die Skalierung der Applikation ist eine Kernanforderung dieser Masterthesis. Die beiden Formen der Skalierung wurden bereits im Abschnitt 2.2.1 erläutert. Da die vertikale Skalierung durch die einer Maschine zur Verfügung stehenden Ressourcen beschränkt ist, kommen nur PaaS Angebote in Betracht, die eine horizontale Skalierung anbieten. Dabei ist es für die Auswahl nicht entscheidend, ob die Skalierung automatisch oder manuell erfolgt. Ebenfalls nicht entscheidungsrelevant ist die Fähigkeit der Load-Balancer mit Sticky-Sessions umgehen zu können, da auf diese, zugunsten der im Abschnitt 2.2.1 genannten besseren Skalierungseigenschaften ohne Sticky-Sessions, bewusst verzichtet wird.

3.1.5 Verteilter Cache

Im Abschnitt 2.3.1 wurde beschrieben, dass ein verteilter Cache benötigt wird, um die ServerSession über alle Server-Instanzen zu synchronisieren. Daraus ergibt sich die Anforderung, dass die PaaS, auf der eine Eclipse Scout Applikation skalierbar betrieben werden soll, einen verteilten Cache zu Verfügung stellen muss.

3.1.6 Message Queue

Ebenfalls im Abschnitt 2.3.1 wurde festgelegt, dass zur Verteilung von Nachrichten über alle Server-Instanzen eine Message Queue benötigt wird. Da in der Cloud eine elastische Skalierung üblich ist und somit die Server-Instanzen je nach Bedarf gestartet und wieder gestoppt werden, muss die Message Queue fähig sein, mit einem flexiblen Publish/Subscribe Pattern umgehen zu können. Das heißt, dieses System muss zudem in der Lage sein,

die Nachrichten an die Server-Instanzen zu verteilen ohne dass dieses die Server-Instanzen kennt. Ebenso darf das ungeplante Beenden von Server-Instanzen keine Auswirkungen auf die Verteilung der Nachrichten haben.

3.1.7 Datenbank

Da die meisten mit Eclipse Scout erstellten Anwendungen Business Applikationen sind, die eine Verarbeitung von Daten mit einschließen, ist es erforderlich, dass auch die PaaS Cloud, in der die Applikationen betrieben werden soll, eine Datenbank zur Verfügung stellt. Auch die in dieser Masterthesis verwendete Demo-Applikation benötigt zur User-Verwaltung eine Datenbank. Die Daten von Geschäftsapplikationen stehen meist in enger Beziehung. Das macht den Einsatz eines relationalen Datenbanksystems erforderlich. Eclipse Scout bietet von sich aus Connectoren zu Derby und MySQL an. Um keinen großen Aufwand in die Anbindung weiterer Datenbanken stecken zu müssen, wäre die Unterstützung für Derby oder MySQL innerhalb der PaaS Cloud wünschenswert.

3.1.8 Zusammenfassung der Anforderungen

Aus den zuvor beschrieben Anforderungen lassen sich somit die folgenden harten Anforderungen definieren:

- Public PaaS Cloud mit Hosting (H1)
- Java 1.7 (H2)
- Servlet Container mit Servlet 2.5 Unterstützung (H3)
- Dienst für verteilten Cache (H4)
- Flexiblen Dienst zum Synchronisieren von Nachrichten (H5)
- Relationale Datenbank (H6)

Als weiche Anforderungen wurden die folgenden Punkte definiert:

- Vorinstallierter Servlet Container (W1)
- Automatische horizontale Skalierung (W2)
- MySQL oder Derby als Datenbank (W3)

Hinter den einzelnen Anforderungen ist in Klammer eine Kennung geschrieben. Diese dient der Zuordnung bei der Überprüfung der Anforderungen im Abschnitt 3.4. Harte Anforderungen haben das Prefix H, weichen Anforderugen haben das Prefix W.

3.2 PaaS Anbieter

Anbieter von PaaS Angeboten gibt es inzwischen viele (vgl. Paa). Der CTO der Firma Cloudcor, Khazret Sapenov, hat in einer ständig aktualisierten Liste (vgl. Kha) 65 Anbieter von PaaS Cloud Systemen zusammen getragen (Stand Feb. 2014). Von diesen 65 PaaS Anbietern unterstützen laut der Liste 35 Anbieter Java. Um Anbieter auszuschließen, die offensichtlich nicht in Frage kommen, wurde eine Vorauswahl anhand der harten Anforderung H1 getroffen. Untersucht werden also nur Anbieter, die eine Public Cloud Lösung mit Hosting anbieten. Herausgestrichen wurden doppelte Einträge, nicht mehr verfügbare Angebote und Anbieter deren Fokus stark auf die Nutzung ihrer eigenen Software ausgelegt ist. Die so ausgeschiedenen Anbieter sind im Anhang A.1 in der Tabelle A.1 zusammen mit dem Grund für das Ausscheiden aufgelistet. Die nicht ausgeschiedenen und somit für die Analyse verbleibenden Anbieter sind in der Tabelle 3.1 aufgelistet. Diese sind potentiell in der Lage Eclipse Scout Applikationen zu betreiben. Sie werden im Folgenden hinsichtlich der im Abschnitt 3.1 beschriebenen Anforderungen untersucht.

Unternehmen	**Produkt**
Amazon	Elastic Beanstalk
CenturyLink Inc.	AppFog
Clever Cloud SAS	Clever Cloud
CloudBees	Run@Cloud
Google	Google AppEngine
Hivext Technologies	Jelastic
Microsoft	Windows Azure
Oracle	Cloud Java
Pivotal	Cloud Foundry
Red Hat	OpenShift
Salesforce	Heroku

Tab. 3.1: Java PaaS Public Cloud Anbieter

3.2.1 Amazon - Elastic Beanstalk

Amazon war der Pionier, was das Bereitstellen von Ressourcen in Form einer Cloud-Lösung an geht. Was mit den Amazon Webservices auf der IaaS-Ebene angefangen hat, wird mit Elastic Beanstalk auf der PaaS-Ebene fortgeführt. Amazon Elastic Beanstalk setzt auf der bestehenden IaaS Landschaft von Amazon auf. Damit sind auch alle anderen Cloud Angebote von Amazon mit Amazon Elastic Beanstalk nutzbar. Von Haus aus unterstützt werden die Programmiersprachen Java, .NET, Node.js, PHP, Phyton und Ruby (vgl. Ama10, Seite 4). Als Plattformen für Java Applikationen werden virtuelle Maschinen mit Linux Betriebssystem wahlweise in 32 oder 64 Bit Architektur mit vorinstalliertem Tomcat 6 oder Tomcat 7 angeboten. Die zur Verfügung stehenden Java Versionen sind Java 1.6 und Java 1.7 (vgl. Ama10, Seite 19-20). Für das dynamische Starten und Beenden von Instanzen bietet Amazon eine Auto Scaling Funktion (vgl. Ama10, Seite 17) an und unterstützt somit eine automatische horizontale Skalierung. Eine vertikale Skalierung wird manuell über die genutzte EC2 Instanz angeboten. Weiterhin bietet Amazon unter dem Namen Amazon ElastiCache (vgl. Ama10, Seite 411-412) zwei Services zum zentralen Cachen von Daten an. Bei der Nutzung von ElastiCache kann zwischen Memcached und Redis gewählt werden. Für das instanzenübergreifende Versenden von Nachrichten stellt Amazon den Simple Notification Service (SNS) und den Simple Queue Service (SQS) bereit. Diese lassen sich jedoch nicht mit akzeptablem Aufwand für eine flexible Publish/Subscribe Architektur nutzen, da für jeden Subscriber eine eigene SQS Queue angelegt werden muss. Diese Queues müssen beim Hochfahren einer Server-Instanz angelegt und beim Herunterfahren wieder gelöscht werden. Beim Hochfahren einer Server-Instanz kann die Queue beim Starten der Applikation angelegt werden kann, beim Herunterfahren hingegen fehlen die auslösenden Events. Das bedeutet, die Applikation wird direkt ausgeschaltet und hat keine Möglichkeit mehr die Queue zu löschen. Damit scheidet die Verwendung von SNS und SQS für Eclipse Scout aus. Eine weitere Möglichkeit ist es, den Service von CloudAMQP zu nutzen. Dieser Service bietet eine SaaS Cloud-Lösung auf Basis von RabbitMQ an und wird innerhalb der Amazon EC2 Cloud gehosted (vgl. cloa). Für die Speicherung von Daten hat Amazon gleich mehrere Lösungen parat. Es werden sowohl SQL als auch NoSQL Datenbanken unterstützt (vgl. Ama10, Seite 38). Amazon RDS bietet Zugriff auf MySQL, Orcale und Microsoft SQL Datenbanken. Die Amazon Cloud hat den großen Vorteil, dass sehr viele Anbieter von SaaS Angeboten diese in der selben Infrastruktur betreiben. Oft kann bei den Anbietern sogar gewählt werden, aus welchem Rechenzentrum der Service bezogen werden soll, um eine optimale Performance zu erreichen.

3.2.2 CenturyLink Inc. - AppFog

AppFog ist das PaaS Angebot der CenturyLink Inc., einem der größten Telekommunikationsunternehmen der USA. Als Basis für ihr PaaS Angebot wird Cloud Foundry eingesetzt. Die unterstützten Programmiersprachen sind Ruby, Node.js, PHP, Phyton und Java. Die eingesetzte Java-Version ist Java 1.7 (vgl. Appb). Als Servlet-Container wird zur Zeit nur der Tomcat 6 angeboten. Eine Unterstützung von Tomcat 7, TomcatEE, GlassFish und JBoss sind jedoch in Planung. Dabei wird der Servlet-Container automatisch installiert, wenn die Applikation installiert wird. Zur Skalierung und Lastverteilung macht AppFog in der Dokumentation keine detaillierten Angaben. Der Preisübersicht lässt sich jedoch entnehmen, dass eine vertikale Skalierung über die Auswahl des Produktes erfolgen kann und jedes Produkt eine horizontale Skalierung zulässt. Dies bestätigt auch die Konfigurationsseite für die installierten Applikationen. Wie in der Grafik 3.1 zu sehen ist, kann die Applikation, die zur Verfügung stehende Anzahl an Instanzen und die jeweilige verfügbare Menge an Arbeitsspeicher fix eingestellt werden. Eine automatische Skalierung ist hier nicht möglich. Als Dienst für einen synchronisierten Cache bietet AppFog Redis an. RabbitMQ wird als Dienst zum Verteilen von Nachrichten angeboten. Für die Anbindung der IronMQ gibt es ein Add-on. Als Datenbanken stehen MySQL, MongoDB und PostgreSQL zur Verfügung. Das Hosting erfolgt derzeit in der Amazon AWS Cloud. Ein Hosting Angebot für die HP OpenStack Cloud befindet sich zur Zeit in der Erprobungsphase (vgl. Appa, vgl. Appc).

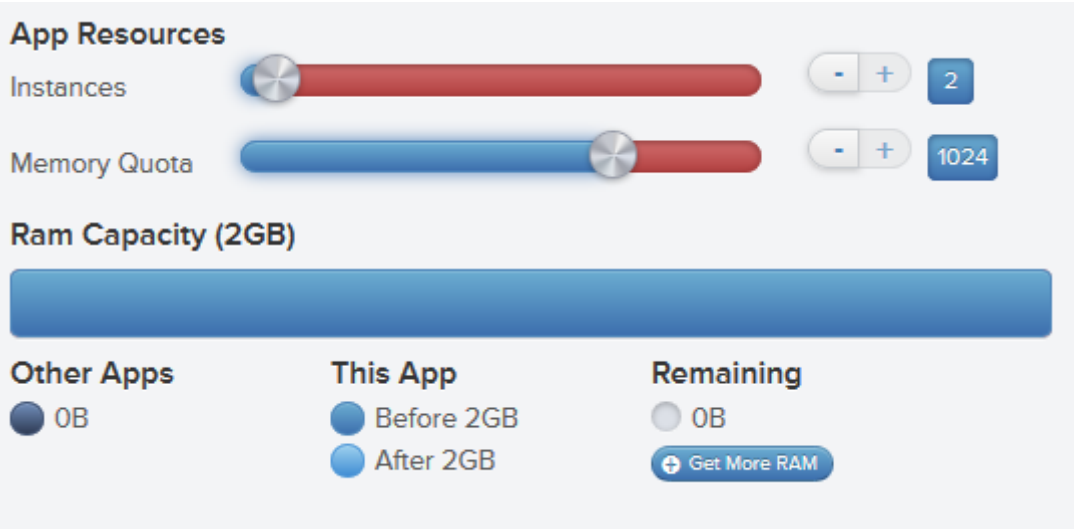

Abb. 3.1: AppFog Skalierung

3.2.3 Clever Cloud SAS - Clever Cloud

Die Clever Cloud ist ein Angebot des gleichnamigen französischen Anbieter Clever Cloud SAS. Unterstützte Programmiersprachen sind Java, PHP, Scala, Node.js, Phyton, Ruby und Go (vgl. Clob). Bei der Java Version kann zwischen Java 1.6 und Java 1.7 gewählt werden (vgl. Cloc). Servlet-Container bietet die Clever Cloud gleich eine ganze Palette an. Diese sind in der Tabelle 3.2 aufgelistet. Was die Skalierung angeht, bietet Clever Cloud sowohl horizontale als auch vertikale Skalierung an. Bei der horizontalen Skalierung kann eine minimale und maximale Anzahl von Instanzen angegeben werden, zwischen denen automatisch skaliert werden soll. Bei der vertikalen Skalierung stehen 7 Typen bereit, die sich in der Ausstattung von CPU und RAM unterscheiden. Auch hier kann eine automatische Skalierung eingestellt werden (vgl. Cloe). Ein Cache-Service oder eine Message Queue werden von Clever Cloud nicht angeboten. Als Datenbanken stehen MySQL, MongoDB und PostgreSQL zur Verfügung. Gehostet wird die Clever Cloud bei der Iliad Group.

Apache Tomcat 4.1	Jetty 6.1	Jboss 6.1	Glassfish 3.1	Resin 3.1
Apache Tomcat 5.5	Jetty 7.6	Jboss AS 7.1		
Apache Tomcat 6.0	Jetty 8.1			
Apache Tomcat 7.0	Jetty 9.0			

Tab. 3.2: Clever Cloud - Verfügbare Servlet Container, Quelle: (Clod)

3.2.4 CloudBees - Run@Cloud

CloudBees hat sich auf Java-Entwickler spezialisiert und bietet als einziger Anbieter eine Unterstützung für den gesamten Entwicklungsprozess. Die Plattform vom CloudBees läuft vollständig in der Amazon Cloud, was den Zugriff auf Services von Amazon stark vereinfacht. CloudBees bietet als vorinstallierte Laufzeitumgebungen Tomcat 6, JBoss 7.02 und JBoss 7.1 an (vgl. Cyr13). Weitere Laufzeitumgebungen sind mit einer eingeschränkten Unterstützung wählbar. Zudem ist es möglich, über CloudBees ClickStack eine eigene Laufzeitumgebung zu nutzen. Die vorinstallierte Java Version ist 1.7 (vgl. Nea13). Es können aber auch die Versionen 1.6 und 1.8 verwendet werden. Skalieren kann CloudBees nur horizontal. Das kann allerdings vollautomatisch erfolgen und komfortabel per Weboberfläche konfiguriert werden (vgl. Mic13). Für die Synchronisierung der Session bietet CloudBees

mit dem Service AppSessionStore (vgl. Cyr14a) eine einfache Möglichkeit, die HTTP-Session zwischen den Servlet-Container-Instanzen zu synchronisieren. Für die Synchronisation von Caches kann der Service von Memcachier genutzt werden. Für die Verteilung von Nachrichten kann auf den externen Dienstleister CoudAMQP zurückgegriffen werden (vgl. Viv13). Als Datenbank kann die von CloudBees angebotene MySQL-Datenbank verwendet werden (vgl. Cyr14b).

3.2.5 Google - AppEngine

Die Google AppEngine gibt es seit 2008 und ist damit eine der ältesten PaaS Lösungen. Anfangs wurde nur Python unterstützt, im Laufe der Zeit ist jedoch eine Unterstützung für Java, PHP und Go hinzugekommen. Java ist in der Version 1.7 verfügbar, allerdings nur in einem gesicherten Sandbox-Betrieb. Innerhalb dieser Sandbox ist der Zugriff auf Java-Klassen nur möglich, wenn diese in der von Google festgelegten Whiteliste gelistet sind (vgl. Goob). Laut dem Online-Magazin InfoQ verwendet die Google AppEngine Jetty als Servlet Container (vgl. Cra09). Die aktuelle Version ist jedoch nicht dokumentiert. Die Skalierung geschieht horizontal und wird von Google voll automatisch vorgenommen (vgl. Gooa). Zur Synchronisation von Caches wird Memcached angeboten. Einen Dienst zum Verteilen von Nachrichten wird nicht unterstützt. Das Hosting wird von Google direkt übernommen.

3.2.6 Hivext Technologies - Jelastic

Jelastic ist ein Exot unter den PaaS Plattformen. Sie vermarkten sich selbst als Platform-as-Infrastructure (PaI). Unter diesem Namen kombinieren Sie PaaS und IaaS Angebote. Hivext Technologies bietet mit Jelastic lediglich die Plattform an, der Betrieb und das Hosting wird von Drittanbietern übernommen. Als javafähige Applikationsserver werden Tomcat 6, Tomcat 7 Tomcat EE, Jetty 6 und GlassFish 3 (vgl. jela) angeboten. Verfügbare Java-Versionen sind Java 1.6 und Java 1.7 (vgl. jeld). Bei Jelastic werden Anfragen mittels Sticky Sessions an die Applikationsserver verteilt (vgl. jelb). Zusätzlich können Sessions über eine spezielle Repliklations-Funktion zwischen zwei Tomcat-Instanzen synchronisiert werden. Soll dennoch eine Synchronisation der Sessions über alle Instanzen erfolgen, ist es möglich, Sessions mittels Memcached direkt zwischen allen Tomcat-Instanzen zu synchronisieren (vgl. jelc). Einen Service zur Verteilung von Nachrichten wird von Jelastic

nicht direkt angeboten. Stattdessen gibt es eine Anleitung zur Konfiguration von JMS über GlassFish (vgl. Mar13). Für die Skalierung bietet Jelastic die Möglichkeit sowohl horizontal als auch vertikal zu Skalieren. Hierbei ist jedoch nur die vertikale Skalierung automatisierbar. Die horizontale Skalierung muss manuell vorgenommen werden. Als Datenbanken können MySQL, MariaDB, PostgreSQL, MongoDB und CouchDB verwendet werden.

3.2.7 Microsoft - Windows Azure

Mit Windows Azure bietet Microsoft eine PaaS Lösung mit einen Fokus auf .Net Applikationen. Dennoch unterstützt sie weitere Programmiersprachen wie Node.js, Java, PHP, Python und Ruby. Welche Java Versionen zur Verfügung stehen ist von Microsoft nicht dokumentiert. Eine Anleitung lässt jedoch darauf schließen, dass Java in der Version 1.6 und 1.7 unterstützt wird. Selbiges gilt für die unterstützten Servlet Container. Auch hier ist nur einer Anleitung zu entnehmen, dass ein Tomcat 7 auf einer VM installiert werden kann (vgl. Mica). Skaliert wird sowohl horizontal als auch vertikal voll automatisch (vgl. Micb). Als synchronisierten Cache kann die hauseigene Lösung Windows Azure-Cache verwendet werden (vgl. Micd). Für das Verteilen von Nachrichten wird der Windows Azure Service Bus angeboten (vgl. Micc). Als Datenbank kann die Windows Azure SQL Database angebunden werden. Gehostet wird das Angebot in der microsofteigenen Infrastruktur.

3.2.8 Oracle - Cloud Java

Das PaaS Angebot von Oracle beschränkt sich ausschließlich auf Java-Applikationen. Als Servlet-Container kommt WebLogic 10.3.6 zum Einsatz. Die verfügbaren Java Versionen sind Java 1.6 und Java 1.7 (vgl. TC12, Seite 1.8-1.9). Eine Skalierung im herkömmlichen Sinn bietet Oracle nicht an. Es kann lediglich aus drei verschiedenen Produkten gewählt werden die wahlweise eine, zwei oder vier WebLogic-Instanzen bereitstellen (oraa). Somit ist eine manuelle horizontale Skalierung möglich. Zum synchronisierten Speichern von Daten in einem Cache schließt Oracle das Oracle Coherence In-Memory Data Grid für die Java Cloud explizit aus. Auch die Verwendung von JMS wird explizit ausgeschlossen (vgl. TC12, Seiten A1-A3). Da Oracle ausschließlich auf ihrer eigenen Infrastruktur arbeitet, können auch keine Services von Drittanbieter performant angebunden werden. Als Datenbank wird der Oracle Database Cloud Service angeboten.

3.2.9 Pivotal - Pivotal One

Pivotal One ist ein Produkt der Pivotal Initiative. Die Pivotal Initiative ist eine Kooperation der EMC Corporation mit VMWare. Unterstützte Programmiersprachen sind Node.js, Java und Ruby (vgl. Foua). Die Plattform kann hier nicht, wie bei den meisten anderen Anbietern, beim Anlegen der Applikationsumgebung ausgewählt werden, sondern muss über einen sogenannten Buildpack installiert werden (vgl. Fouc). Der von Pivotal genutzte vorkonfigurierte Buildpack für Java Applikationen schließt Java in der Version 1.7 ein und enthält Tomcat in der 7. Version. Skaliert wird bei Povital One sowohl horizontal als auch vertikal, jedoch nur manuell (vgl. Foub). Über den Cloud Foundry Services Marketplace können verschiedene Dienste genutzt werden. Für synchronisiertes Caching stehen sowohl Memcached Cloud als auch Redis Cloud zur Verfügung. Als Message Queue können die Dienste von CloudAMPQ und IronMQ angebunden werden. Verfügbare Datenbanken sind MySQL, MongoDB und PostgreSQL. Bei der Auswahl des Hosting Partner ist es möglich zwischen der AWS Elastic Beanstalk Cloud und den hauseigenen Pivotal Web Services zu wählen.

3.2.10 Red Hat - OpenShift

OpenShift ist das PaaS Angebot von Red Hat. Als Software-Basis wird die Open Source Software OpenShift Origin eingesetzt. Gehostet wird OpenShift in der Amazon Cloud (vgl. MRS11). Programmiersprachen, die unterstützt werden, sind Java, Phyton, PHP, Perl, Ruby und Node.js (vgl. Opec). Als Applikationsserver für Java-Anwendungen steht der JBoss Application Server 7, die JBoss Enterprise Application Platform 6, Wildfly 8, und der JBoss Enterprise Web Server wahlweise mit Tomcat 6 oder Tomcat 7 zur Verfügung (vgl. Opea). Die zur Verfügung stehenden Java-Versionen werden von OpenShift nicht angegeben. Die vorinstallierte Java-Version ist jedoch, wie in der Abbildung 3.2 zu sehen, 1.7 (OpenJDK). Skaliert wird bei OpenShift nur horizontal, das allerdings automatisch, unter Angabe der minimalen und maximalen Anzahl an Instanzen (vgl. Opeb). Als Cache-Service kann Redis eingesetzt werden. Für die Nutzung einer Message Queue wird eine Integration von IronMQ über den Cloud-Service von Iron.io angeboten. Als Datenbanken können MySQL, PostgreSQL und MongoDB verwendet werden. Gehostet wird OpenShift in der Amazon Cloud.

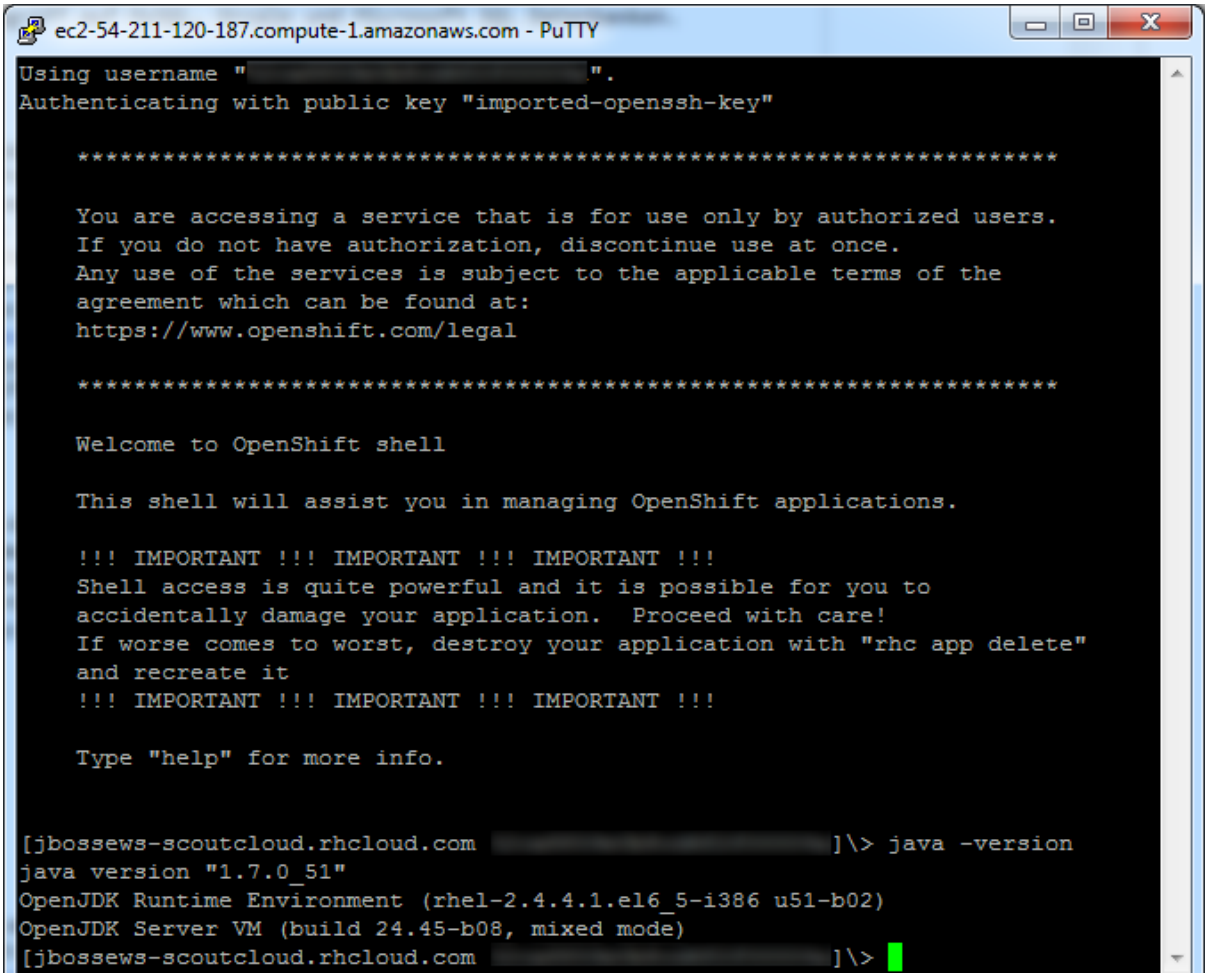

Abb. 3.2: OpenShift Java-Version

3.2.11 Salesforce - Heroku

Heroku ist das PaaS Angebot von Salesforce. Die unterstützten Programmiersprachen sind Ruby, Java, Phyton, Clojure, Scala, Node.js und Play (vgl. here). Java wird in der OpenJDK-Versionen 1.6, 1.7 und 1.8 unterstützt (vgl. herc). Die dokumentierten zur Verfügung stehenden Servlet-Container sind Tomcat 7 (vgl. her13a) und Jetty 7 (vgl. her13b). Im Rahmen der Skalierung ist es möglich sowohl vertikal (vgl. herb) als auch horizontal (vgl. herh) zu skalieren. Beides ist jedoch nur manuell möglich. Einen Cache-Service bietet Heroku selbst nicht an. Es stehen jedoch Memcached (vgl. herf) und Redis (vgl. herg) als Cloud-Lösung zur Verfügung. Den gleichen Ansatz hat Heroku auch bei der Bereitstellung von Message Queue. Es gibt kein eigenes Angebot, es kann aber auf CloudAMQP (vgl. hera) und IronMQ (vgl. herd) zugegriffen werden. Als Datenbank wird von Heroku PostgreSQL angeboten. Gehostet wird Heroku in der Amazon Cloud.

3.3 PaaS Cloud-Anbieter im Vergleich

Im Abschnitt 3.2 wurden alle relevanten PaaS Anbieter einzeln gegenüber der im Abschnitt 3.1 gestellten Anforderungen untersucht. Um nun einen Überblick über die Eigenschaften der PaaS Angebote bezüglich der gestellten Anforderungen zu erhalten, sind die Eigenschaften der einzelnen PaaS Angebote nochmals in der Tabelle 3.3 und 3.4 gegenübergestellt. Eine Aufteilung der Anbieter in zwei Tabellen ist nur der zugunsten einer besseren Übersichtlichkeit erfolgt.

Bei der Infrastruktur der PaaS Angebote setzen die meisten Anbieter auf eigene Implementierungen. Es gibt aber auch Anbieter, wie zum Beispiel Pivotal und AppFog, die auf die Open Source Plattform Cloud Foundry setzen. Eine Ausnahme unter den PaaS Anbietern stellt Jelastic dar. Jelastic stellt lediglich die PaaS als Plattform-Lösung bereit und lässt diese bei Hosting-Partnern betreiben. Das Hosting in der eigenen Infrastruktur können sich nur die größten Anbieter wie Amazon, Google, Microsoft und Oracle leisten. Die, die keine eigene Infrastruktur haben, nutzen meist die Amazon Cloud als zugrunde liegende Infrastruktur. Eine Auswahl der Hosting-Infrastruktur ist nur bei den Cloud Foundry basierten Angeboten von AppFog und Pivotal möglich.

Bei der angebotenen Java Version gibt es zwischen den Anbietern kaum Unterschiede. Java 1.7 ist die vorherrschende Java Version. Sie wird von allen elf Anbietern unterstützt (vgl. Abb. 3.3). Lediglich sechs Anbieter bieten noch Java in der Version 1.6 an. Die Java Version 1.8 ist noch nicht bei den PaaS Anbietern angekommen. Erst ein Anbieter unterstützt diese Version. Als einziger Anbieter stellt Google Java nur innerhalb einer Sandbox bereit, in der der Zugriff auf einige Java-Klassen beschränkt ist.

Abb. 3.3: Anzahl des Vorkommen der Java Version

Bei der Bereitstellung der Servlet Container sieht es ähnlich aus. Die Häufigkeit des Vorkommen der Servlet-Container ist in der Abbildung 3.4 zu sehen. Apache Tomcat wird von neun der elf Anbieter unterstützt. Nur Oracle und Google bieten Apache Tomcat als

Servlet Container nicht an. Die Version, in der Tomcat angeboten wird ist meist die Version 6 oder 7. Weiterhin werden von jeweils drei Anbietern JBoss, Jetty oder GlassFish als Servlet Container angeboten.

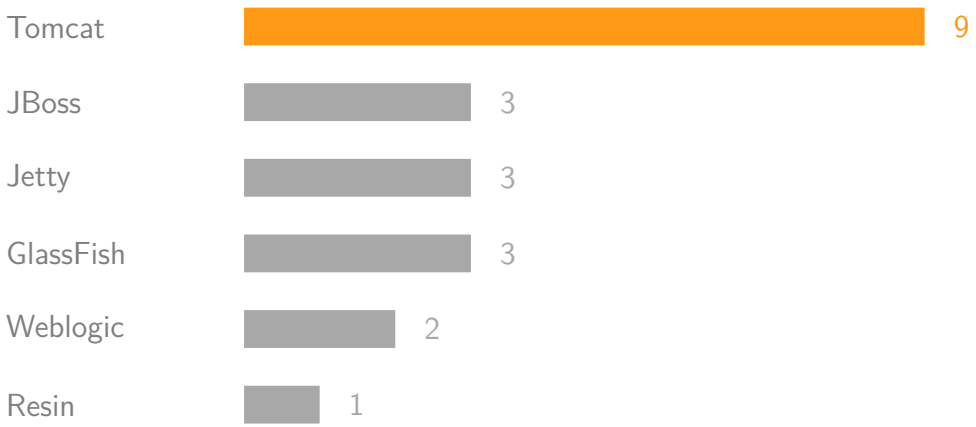

Abb. 3.4: Anzahl des Vorkommen der Servlet-Container

Bei der Skalierung gibt es große Unterschiede. Zwar gibt es bei jedem Anbieter in gewisser Weise die Möglichkeit entweder horizontal oder vertikal zu skalieren, aber welche dieser Möglichkeiten zur Verfügung steht und ob die Skalierung automatisch oder manuell passiert, ist von Anbieter zu Anbieter verschieden. So ist die Auswahl bei Oracle recht eingeschränkt, da eine eine horizontale Skalierung nur über die Änderungen des PaaS Produkts möglich ist. Bei Microsoft hingegen kann sowohl vertikal als auch horizontal vollautomatisch skaliert werden.

Unter den angebotenen Cache-Services lässt sich beobachten, dass Memcached oder Redis besonders von nahezu allen Angeboten unterstützt werden. Neben diesen beiden Cache Services gibt es lediglich zwei alternative proprietäre Angebote (vgl. Abb.3.5). Einzig das Clever Cloud Angebot hat keinen Cache Service im Angebot.

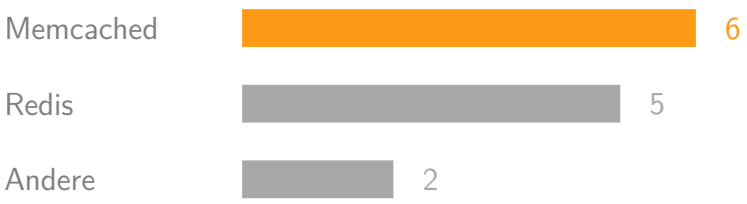

Abb. 3.5: Anzahl des Vorkommen der Cache-Services

Bei den Message Services ist die Situation recht ähnlich. Hier vorherrschend sind RabbitMQ und IronMQ. RabbitMQ wird von fünf Anbietern unterstützt und IronMQ von vier. Mal direkt in der PaaS Cloud integriert, mal über CloudAMQP oder Iron.io als SaaS Angebot eingebunden. Aber auch hier gibt es Anbieter, die auf eigene Implementierungen für eine Message Queue setzen. So haben Amazon, Microsoft und Oracle selbst erstellte Lösungen im Einsatz. Es gibt mit der Clever Cloud und der Google AppEngine aber auch Angebote, die keine Unterstützung von Message Queues bereitstellen.

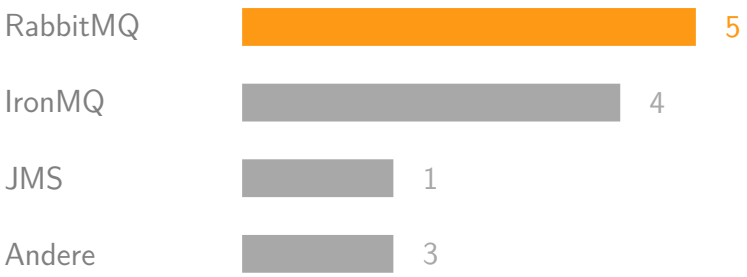

Abb. 3.6: Anzahl des Vorkommen der Message Queues

Datenbanken werden von allen Anbietern bereitgestellt. Die Grafik 3.7 zeigt die Häufigkeit der Datenbankangebote. Gut zu sehen ist, dass die Datenbank MySQL mit acht Angeboten nahezu von allen der elf PaaS Anbietern offeriert wird. PostgreSQL ist bei sechs Anbietern und damit knapp über der Hälfte verfügbar. MongoDB ist bei fünf Angeboten vertreten und damit die populärste NoSQL Datenbank.

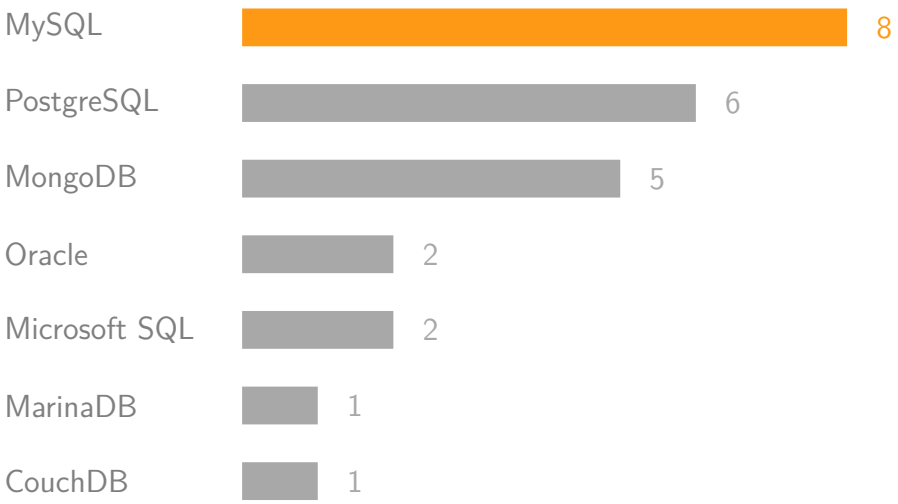

Abb. 3.7: Anzahl des Vorkommen der Datenbanken

Anbieter	Produkt	Java-Version	Servlet-Containter	Skalierung	Cache-Service	Message-Service	Datenbank
Amazon	Elastic Beanstalk	1.6, 1.7	Tomcat 6 & 7	horizontal (automatisch)	Memcached, Redis	Amazon SQS / SNS, RabbitMQ	MySQL, Oracle, SQL-Server
CenturyLink Inc.	AppFog	1.7	Tomcat 6	horizontal (manuell), vertikal (manuell)	Redis	RabbitMQ, IronMQ	MySQL, MongoDB, PostgreSQL
Clever Cloud SAS	Clever Cloud	1.7	siehe Tabelle 3.2	horizontal (automatisch), vertikal (manuell)	-	-	MySQL, MongoDB, PostgreSQL
CloudBees	Run@Cloud	1.6, 1.7	Tomcat 6 & 7, JBoss 7.2, Glassfish 3	horizontal (automatisch)	Memcached	RabbitMQ	MySQL
Google	AppEngine	1.7 (Sandbox)	Jetty	horizontal (automatisch)	Memcached	-	MySQL
Hivext	Jelastic	1.6, 1.7	Tomcat 6 & 7, Glasfish 3	horizontal (manuell), vertikal (automatisch)	Memcached	JMS	MySQL, MariaDB, PostgreSQL, MongoDB, CouchDB

Tab. 3.3: Cloudanbieter im Vergleich - Teil 1

Anbieter	Produkt	Java-Version	Servlet-Containter	Skalierung	Cache-Service	Message-Service	Datenbank
Microsoft	Windows Azure	1.6, 1.7	Tomcat 7	horizontal (automatisch), vertikal (automatisch)	Azure Cache	Windows Azure Service Bus	Windows Azure SQL Database
Oracle	Cloud Java	1.6, 1.7	Weblogic	horizontal (manuell)	Coherence Dedicated Nodes	Oracle Messaging Cloud Service	Oracle Database Cloud Service
Pivotal	Pivotal One	1.7	Tomcat 7	horizontal (manuell), vertikal (manuell)	Memcached, Redis	RabbitMQ, IronMQ	MySQL, PostgreSQL, MongoDB
Red Hat	OpenShift	1.7	Tomcat 6 & 7, JBoss AS 7, JBoss EAP6, Wildfly 8	horizontal (automatisch)	Redis	IronMQ	MySQL, PostgreSQL, MongoDB
Salfesforce	Heroku	1.6, 1.7, 1.8	Tomcat 7, Jetty 7	horizontal (manuell), vertikal (manuell)	Memcached, Redis	RabbitMQ, IronMQ	PostgreSQL

Tab. 3.4: Cloudanbieter im Vergleich - Teil 2

3.4 PaaS Cloud-Anbieter Anforderungserfüllung

Nachdem in den vorhergehenden Abschnitten ausführlich auf die einzelnen Anforderungen und die Eigenschaften der PaaS Angebote eingegangen wurde, wird in diesem Abschnitt auf die direkte Erfüllung der gestellten Anforderungen geprüft. Die Ergebnisse dieser Prüfung sind in der Tabelle 3.5 dargestellt. Ist eine Anforderung erfüllt, ist diese mit OK und einer grünen Hinterlegung gekennzeichnet. Ist eine Anforderung nicht erfüllt, so ist diese mit NOK und einer roten Hinterlegung gekennzeichnet.

Da die Auswahl der zu prüfenden PaaS Angebot bereits an der Anforderung H1 getroffen wurde, erfüllen auch alle Anbieter dieses Kriterium. Bei der Anforderung H2 geht es um die Java Version. Hier haben unterstützen alle Anbieter die geforderte Java Version 1.7 in ihrem Service. Das Angebot von Google bietet mit seiner Sandbox-Lösung jedoch nur einen eingeschränkten Zugriff auf die Java-Klassen und genügt somit nicht den gestellten Java-Anforderungen. Bei den Servlet-Containern und damit dem Kriterium H3 fällt keiner der Anbieter durch. Alle haben mindestens einen Servlet-Container im Angebot, der Servlets in der Version 2.5 verarbeiten kann. Die nächste Anforderung H4, die einen verteilten Cache voraussetzt, konnte von nahezu jedem Angebot erfüllt werden. Lediglich Clever Cloud hat keinen verteilten Cache anzubieten. Bei der Message Queue, der Anforderung H5, sieht es hingegen etwas anders aus. Hier haben Clever Cloud und Google keine Message Queue offeriert und Red Hat unterstützt zwar IronMQ, diese ist jedoch nicht ohne zusätzlichen Aufwand für eine flexible Publish/Subscribe Architektur zu nutzen. Die sechste harte Anforderung H6, die eine relationale Datenbank voraussetzt, kann dagegen wieder von jedem der Anbieter erfüllt werden.

Die erste weiche Anforderung W1 konnte von allen Anbietern, bis auf Pivotal, erfüllt werden. Hier muss vor dem ersten Deploy, über ein Building Pack, die Laufzeitumgebung erst installiert werden. Die zweite weiche Anforderung W2, die eine automatische, horizontale Skalierung voraussetzt, kann von den Cloud Foundry basierten PaaS Angeboten AppFog und Pivotal One sowie Cloud Java und Heroku nicht erfüllt werden. Die dritte und letzte weiche Anforderung W3 macht eine MySQL Datenbank zur Bedingung. Diese kann von sieben der elf betrachteten PaaS Angeboten erfüllt werden. Keine MySQL Datenbank wird von Microsoft, Oracle und Salesforce angeboten.

Damit fallen drei Anbieter für den Betrieb einer Eclipse Scout Applikation aus. Diese sind Clever Cloud, AppEngine und OpenShift. Alle anderen Anbieter sind potentiell in der Lage eine Applikation, die mit dem Eclipse Scout Framework erstellt wurde zu betreiben.

Für welche zwei dieser acht Anbieter eine Implementierung erstellt wird, hängt demnach von der Erfüllung der weichen Kriterien ab. Fünf der acht Anbieter können nicht alle weichen Anforderungen erfüllen. Übrig bleiben drei Anbieter, die es schaffen, allen harten und weichen Anforderungen zu entsprechen. Diese sind Amazons Elastic Beanstalk, CloudBees Run@Cloud und Hivetext Jelastic. Im Rahmen dieser Masterthesis soll nur für zwei Anbieter eine Implementierung erstellt werden. Von diesen drei Anbietern habe ich mich für Amazon und CloudBees entschieden, da Jelastics zum einen nur einen zweiwöchigen Testzeitraum anbietet und zum anderen nicht dem klassischen Hosting Modell von PaaS Angeboten entspricht.

Anbieter	Produkt	H1	H2	H3	H4	H5	H6	W1	W2	W3
Amazon	Elastic Beanstalk	OK	OK	OK	OK	OK	OK	OK	OK	OK
CenturyLink Inc.	AppFog	OK	OK	OK	OK	OK	OK	OK	NOK	OK
Clever Cloud SAS	Clever Cloud	OK	OK	OK	NOK	NOK	OK	OK	OK	OK
CloudBees	Run@Cloud	OK	OK	OK	OK	OK	OK	OK	OK	OK
Google	AppEngine	OK	NOK	OK	OK	NOK	OK	OK	OK	OK
Hivext	Jelastic	OK	OK	OK	OK	OK	OK	OK	OK	OK
Microsoft	Windows Azure	OK	OK	OK	OK	OK	OK	OK	OK	NOK
Oracle	Cloud Java	OK	OK	OK	OK	OK	OK	OK	NOK	NOK
Pivotal	Pivotal One	OK	OK	OK	OK	OK	OK	NOK	NOK	OK
Red Hat	OpenShift	OK	OK	OK	OK	NOK	OK	OK	OK	OK
Salfesforce	Heroku	OK	OK	OK	OK	OK	OK	OK	NOK	NOK

Tab. 3.5: PaaS Anbieter - Erfüllung der Anforderngen

4 Implementierung

Im Kapitel 2 wurde die Architektur von Eclipse Scout Anwendungen untersucht und mit Architekturen von typischen Cloud Applikationen verglichen. Auf dieser Basis wurden im vorhergehenden Kapitel 3 Anforderungen definiert, die ein PaaS Anbieter erfüllen muss, damit Eclipse Scout Anwendungen auf diesen Angeboten skalierbar und lauffähig sind. Gängige PaaS Anbieter wurden hinsichtlich der Anforderungen untersucht und bewertet. Dieses Kapitel beschäftigt sich mit den Anpassungen, die an Eclipse Scout Framework notwendig sind, um Anwendungen auf Basis von Eclipse Scout in den ausgewählten PaaS Angeboten skalierbar zu betreiben sowie den Implementierungen der Services, die von den einzelnen Anbietern bereitgestellt werden. Zudem werden die Anpassungen beschrieben die an der Demo-Applikation BahBahChat vorgenommen wurden. Weiterhin wird erklärt, wie bei den ausgewählten PaaS Anbietern eine Eclipse Scout Applikation installiert werden kann.

4.1 Anpassungen am Framework

Da das Eclipse Scout Framework eine Open Source Software ist, unterliegen Änderungen am Code bestimmten Voraussetzungen und Prozessen. Diese sind im Scout Wiki Artikel „Contributions for Scout Committers“ [1] ausführlich beschrieben.

Die notwendigen Anpassungen am Framework, welche die Architektur betreffen, wurden im Abschnitt 2.3 bereits festgelegt. Dieser Abschnitt beschäftigt sich mit den konkreten Auswirkungen dieser Anpassungen auf das Eclipse Scout Framework und den daraus resultierenden Implementierungen.

1 http://wiki.eclipse.org/Scout/Contributions_for_Scout_Committers

4.1.1 Cache Synchronisation

Im Abschnitt 2.1.3 wurde beschrieben, dass verschiedene Daten bisher in der HTTP-Session gespeichert werden. Der Zugriff auf die Daten in der HTTP-Session erfolgte bisher direkt über die Methode getSession() der Klasse HttpServletRequest. Um diesen direkten Zugriff zu entkoppeln, wurde ein Service erstellt, der sich um das Caching kümmert. Dieser wird durch das Interface `ICacheStoreService` repräsentiert, dargestellt in der Abbildung A.1. Der `ICacheStoreService` bietet Methoden an, um Daten in Cache zu speichern und wieder auszulesen sowie die Gültigkeit der im Cache abgelegten Daten zu verlängern. Dabei ist es möglich, die Daten einem bestimmten Client zuzuordnen oder die Daten unabhängig von einem Client zu cachen. In der Abbildung 4.1 sind die Abhängigkeiten zwischen dem Interface und den Implementierungen dargestellt. So implementiert der `DefaultCacheService` das Interface direkt, der `MemcachedCacheService` und der `RedisCacheService` jedoch implementieren das Interface, indem sie den `AbstractCacheStoreService` erweitern.

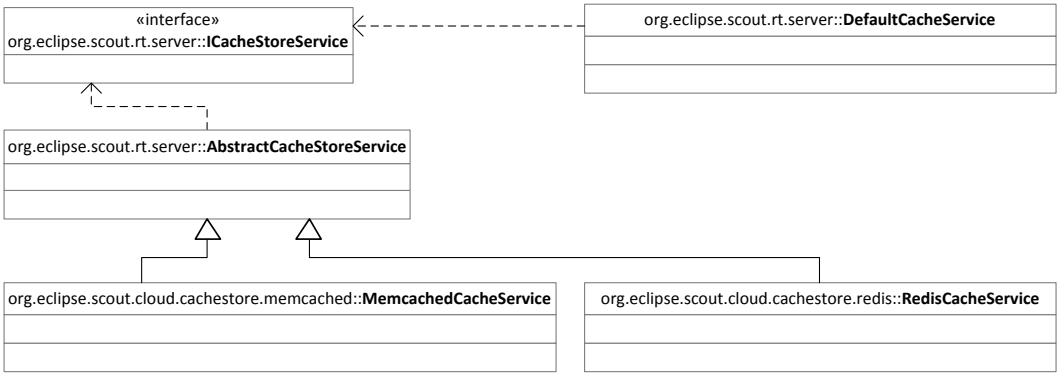

Abb. 4.1: UML Klassendiagramm - Cache Services

Der `AbstractCacheStoreService` enthält Methoden zum serialisieren und deserialisieren von Objekten. Dies ist notwendig, da das Bundle, in dem sich die Implementierung befindet, nicht immer Zugriff auf alle Klassen hat, die deserialisiert werden sollen. Wie die jeweiligen Implementierungen im Detail aussehen, ist im Abschnitt 4.3 beschrieben.

Für die Änderungen am Eclipse Scout Framework, die die Cache-Synchronisation betreffen, wurde der Bug #421614 [1] erstellt. Changes, die diesen Bug betreffen, wurden in Gerrit unter der Change-Id I9421e3f83e29e2c33b4313d484de3910dbb4b2a5 [2] gesammelt.

1 `https://bugs.eclipse.org/bugs/show_bug.cgi?id=421614`
2 `https://git.eclipse.org/r/#/c/19066/`

4.1.2 Nachrichten Synchronisation

Im Abschnitt 3.1.6 wurde bereits beschreiben, dass Nachrichten zur Synchronisation von ClientNotifications und Caches benötigt werden. Das bedeutet, dass Nachrichten verschiedener Funktionen zwischen den Server-Instanzen verteilt werden müssen. Dabei soll die Unabhängigkeit von einer bestimmten Message Queue Implementierung wie z. B. RabbitMQ oder JMS gewahrt bleiben. Eine weitere Prämisse ist, dass das Versenden und Verarbeiten von Nachrichten mit dynamisch hinzugefügten oder entfernten Server-Instanzen funktionieren muss. Weiterhin ist es notwendig, dass die versendeten Nachrichten genau ein mal an jede Server-Instanz ausgeliefert wird.

Bei der Verteilung der Nachrichten hilft das Kommunikationsmuster Publish/Subscribe. Wie dieses Implementiert wird, ist in der Grafik 4.2 zu sehen. Es gibt ein gemeinsames Thema „scoutNotificationQueue“ auf das alle Server-Instanzen ihre Nachrichten publizieren. Abonniert eine Server-Instanz dieses Thema, wird automatisch eine eigene Queue für sie angelegt. Alle Nachrichten, die auf dem Thema „scoutNotificationQueue“ publiziert werden, werden automatisch an alle Queues der Abonnenten verteilt. Das bedeutet, dass jeder Abonnent jede Nachricht genau einmal bekommt. Damit die publizierende Server-Instanz nicht seine eigenen Nachrichten verarbeitet, wird jeder versendeten Nachricht die ID der publizierenden Server-Instanz mitgegeben.

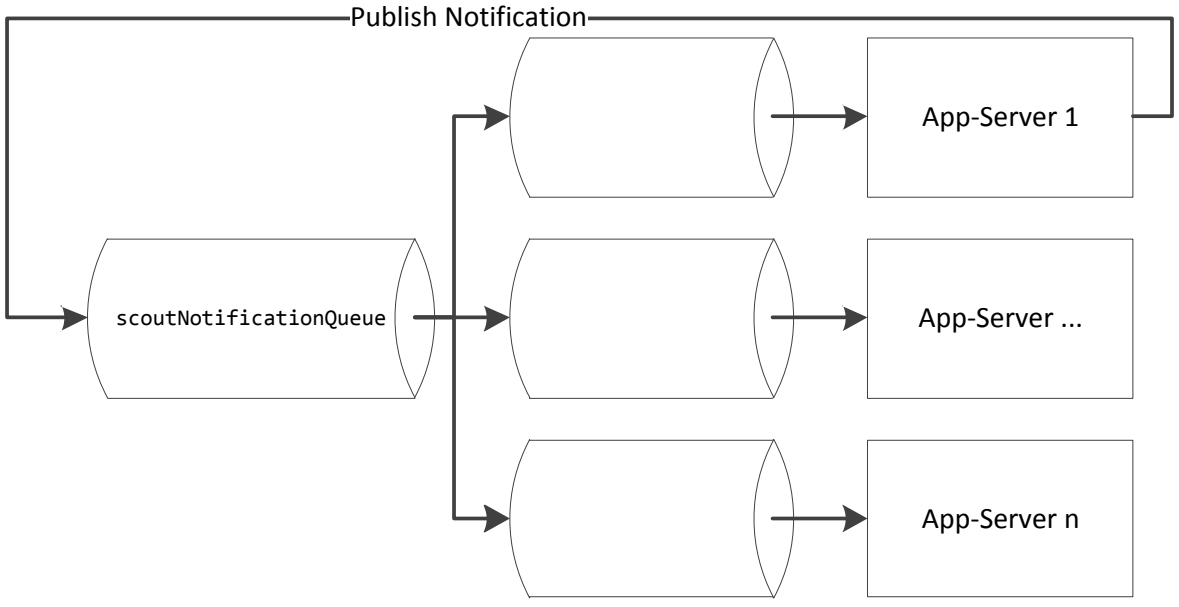

Abb. 4.2: Kommunikations-Muster - Publish/Subscribe

Innerhalb von Eclipse Scout übernimmt der neu angelegte Service `NotificationService` die Verarbeitung von Nachrichten. Dieser Service wird durch das Interface `INotificationService` repräsentiert. Die von diesem Interface angebotenen Funktionen sind in der Abbildung 4.3 zu sehen. Neben Funktionen zum Starten, Beenden und zur Statusabfrage, gibt es die Möglichkeit, dem Service Nachrichten zum Publizieren zu übergeben und empfangene Nachrichten zu verarbeiten. Zudem ist es möglich, dem Service sog. Listener hinzuzufügen und wieder zu entfernen. Diese Listener sind Java-Klassen, die das Interface `IDistributedNotificationListener` implementieren. Mit Hilfe des durch die Listener implementierten Beobachter-Pattern ist der `NotificationService` in der Lage mit verschiedenen Typen von Notifications umzugehen. Zudem macht er es so möglich, dass die Entwickler von Eclipse Scout Applikationen dieses Konstrukt auch für eigene Nachrichten, die über alle Server-Instanzen verteilt werden sollen, verwenden können.

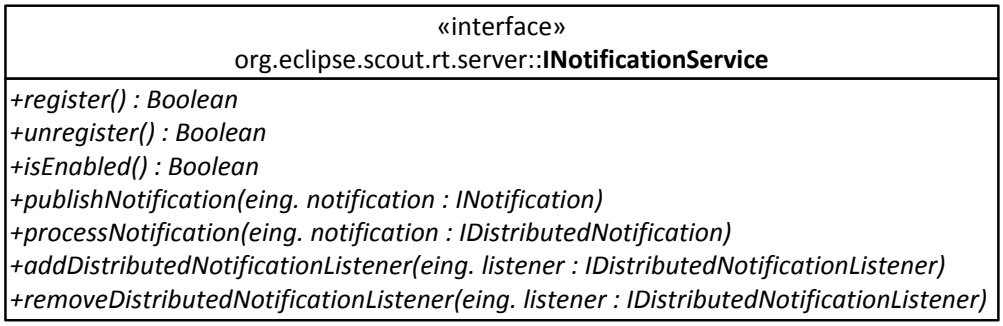

Abb. 4.3: UML Klassendiagramm - INotificationService

Um die Entkopplung zwischen dem `NotificationService` und den Implementierungen der Message Queues zu gewährleisten, wurde ein weiteres Service-Interface angelegt. Dieses nennt sich `IPubSubMessageService` und ist in der Grafik 4.4 zu sehen. Neben Methoden zum Abonnieren und Beenden eines Abonnements enthält das Interface eine Methode, welche Nachrichten entgegen nimmt und an die Message Queue schickt. Für die-

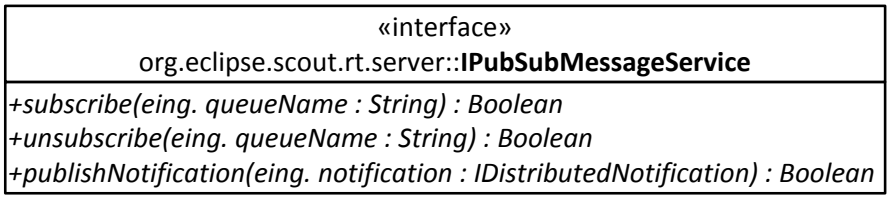

Abb. 4.4: UML Klassendiagramm - INotificationService

ses Interface können nun Implementierungen für die konkreten Message Queues erstellt und über die OSGi Service Registry verfügbar gemacht werden. Die Abhängigkeiten zwischen den Interfaces `INotificationService`, `IDistributedNotificationListener` und `IPubSubMessageService` sowie den dafür erstellten Implementierungen, sind in der Abbildung 4.5 dargestellt. Für das Interface `INotificationService` gibt es nur eine Implementierung, den `NotificationService`. Dieser kann, falls notwendig, von Eclipse Scout Entwicklern ausgetauscht werden. Für die meisten Fälle sollte diese eine Implementierung jedoch vollkommen ausreichen. Weiterhin zeigt die Grafik, dass zwei Implementierungen für das Interface `IPubSubMessageService` erstellt wurden. Eine für RabbitMQ und eine für ActiveMQ. Diese sind im Abschnitt 4.3 näher beschrieben. Der `NotificationService` lädt bei der Initialisierung die Implementierung von `IPubSubMessageService` von der OSGi Service Registry. Zu jeder der beiden Implementierungen von `IPubSubMessageService` gibt es zudem noch ein einen `NotificationListener`. Dieser läuft in einem eigenen Thread und wartet auf eingehende Nachrichten von der Message Queue. Die erhaltenen Nachrichten werden dann an den `NotificationService` zur Verarbeitung weitergegeben. Weiterhin zeigt die Abbildung drei Klassen, die das Interface `IDistributedNotificationListener` implementieren. Jede dieser Klassen verarbeitet Nachrichten eines bestimmten Zwecks.

Bei der Verarbeitung der Nachrichten kann es vorkommen, dass weitere Services aus der OSGi Service Registry angesprochen werden müssen. Einige Services setzen voraus, dass eine ServerSession geladen ist, um diese aufzurufen. Da die Nachrichtenverarbeitung unabhängig von den Clients und ihren ServerSessions funktionieren muss, wird beim Start der Server-Applikation zusätzlich eine Backend-ServerSession angelegt. Zur Verwaltung dieser Session wurde der Service `BackendService` angelegt. Dieser legt bei Bedarf eine neue Backend-ServerSession an und speichert diese im verteilten Cache.

Für die Änderungen am Eclipse Scout Framework, die das Versenden von Nachrichten betreffen, wurde der Bug #421615 [1] erstellt. Changes, die diesen Bug betreffen wurden in Gerrit unter der Change-Id I9421e3f83e29e2c33b4313d484de3910dbb4b2a5 [2] gesammelt. Das ist die selbe Change ID, wie bei den Änderungen, die den Cache betreffen, da hier beim Commiten ein Fehler passiert ist.

1 https://bugs.eclipse.org/bugs/show_bug.cgi?id=421615
2 https://git.eclipse.org/r/#/c/19066/

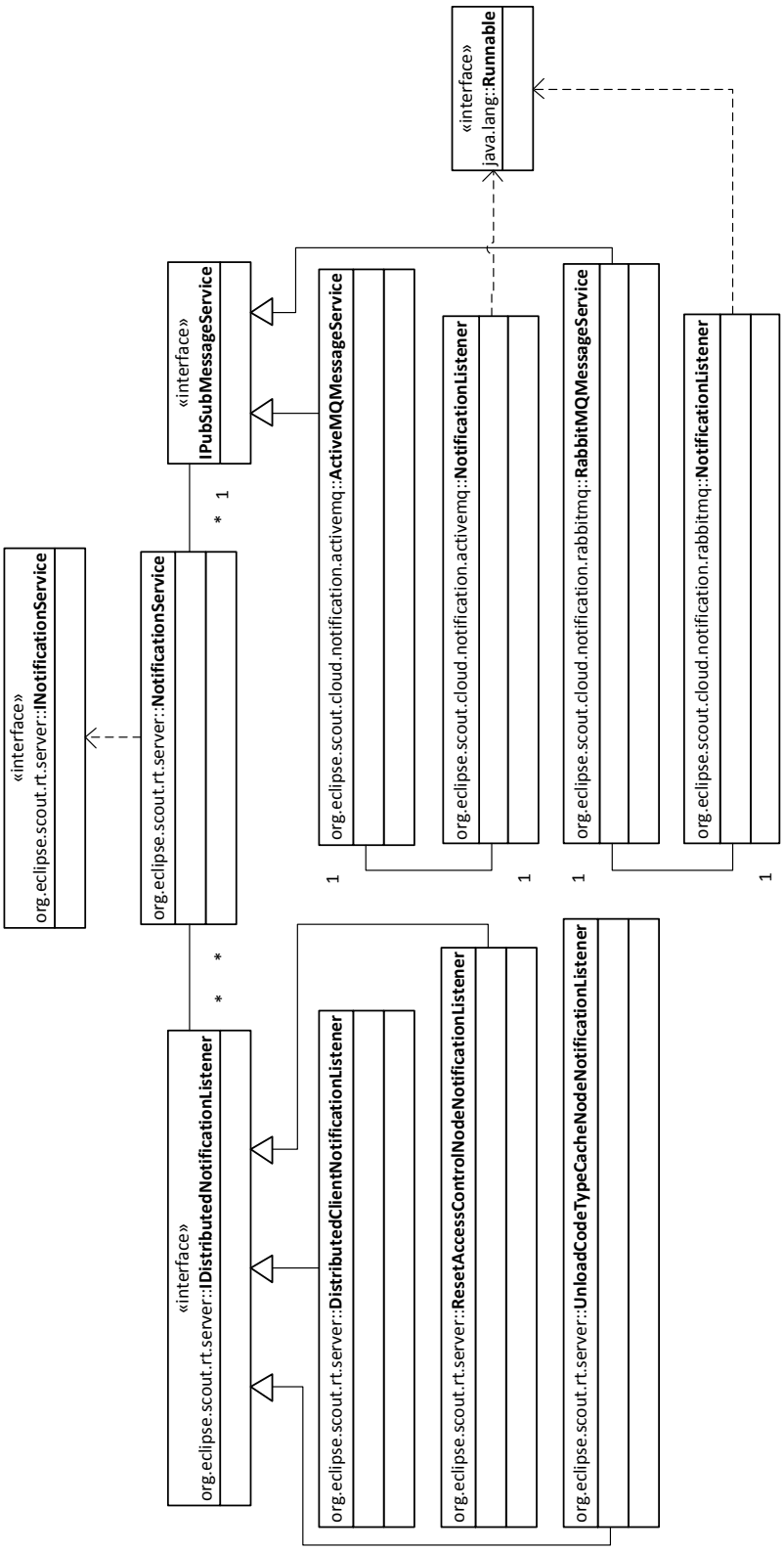

Abb. 4.5: UML Klassendiagramm - Nachrichten

Synchronisation der ClientNotifications

Wie die Speicherung und Verteilung von ClientNotifications bisher in Eclipse Scout abgewickelt wurde, ist im Abschnitt 2.1.3 beschrieben. Warum das Eclipse Scout Framework an dieser Stelle angepasst werden muss, ist im Abschnitt 3.1.6 erläutert. Statt die ClientNotifications lediglich innerhalb der serverinternen Queue abzulegen, sollen diese, falls gewünscht, über den `NotificationService` an alle anderen Server-Instanzen verteilt werden. Damit die ClientNotifications über alle Server-Instanzen identifiziert werden können, wurde das `QueueElement`, in dem die ClientNotification in der internen Queue abgelegt wird, um eine eindeutige ID erweitert.

Um die Synchronisierung der ClientNotifications über alle Server-Instanzen zu implementieren, sind mehrere Änderungen an der `ClientNotificationQueue` notwendig. In der Abbildung 4.6 sind die Methoden zu sehen, die die `ClientNotificationQueue` anbietet. Die blau hinterlegten Methoden sind dabei neu hinzugekommen. Die Methode `putDistributedNotification` nimmt neue ClientNotifications entgegen, steckt sie in ein `QueueElement` und verteilt dieses über den `NotificationService` an alle anderen Server-Instanzen. Anschließend gibt sie das erzeugte `QueueElement` an die entsprechende Methode `putNotification` weiter. Diese legt das übergebene `QueueElement` in der internen Queue ab. Die Methode `removeNotification` nimmt die ID eines `QueueElement` entgegen und löscht dieses aus der lokalen Queue. Die Methode `updateNotification` ergänzt ein `QueueElement` aus der lokalen Queue mit IDs von Clients, die die Nachricht bereits abgerufen hatten.

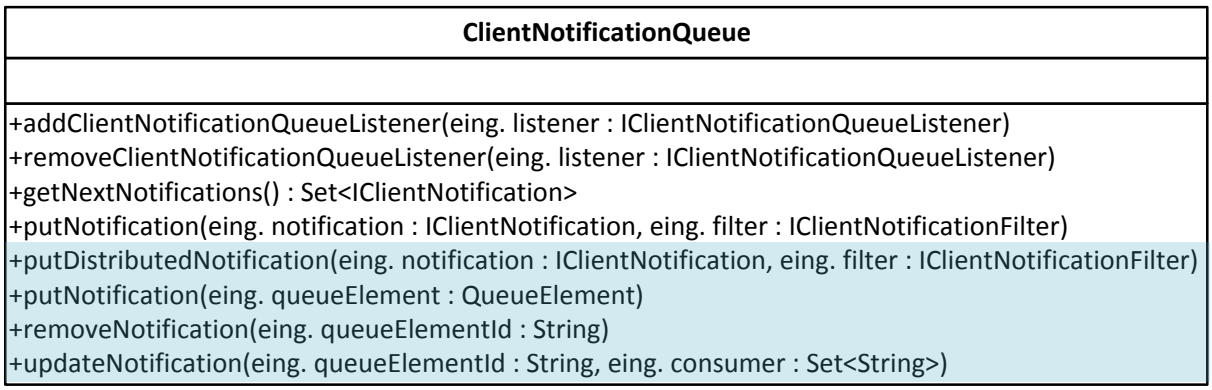

Abb. 4.6: UML Klassendiagramm - ClientNotificationQueue

Neben den Methoden, die der `ClientNotificationQueue` hinzugefügt wurden, wurde die bestehende Methode `getNextNotifications` angepasst. Wie diese bisher intern gearbeitet hat ist in der Abbildung 2.6, im Abschnitt 2.1.3, zu sehen. Hinzugefügt wurden, wie in der Abbildung 4.7 zu sehen zwei Prozessschritte. Zum einen wird nach dem Löschen eines Elements aus der internen Queue allen anderen Server-Instanzen über den NotificationService mitgeteilt, dass diese das Element auch aus ihrer lokalen Queue löschen können. Zum anderen werden alle Server-Instanzen informiert, nachdem eine Nachricht, die für alle Clients bestimmt ist, von einem Client konsumiert wurde. Dieser Mechanismus gewährleistet, dass Nachrichten nicht doppelt ausgeliefert werden. Für den Fall, dass ein Client, aufgrund einer verzögerten Synchronisation, dennoch eine Nachricht doppelt erhält, wurde innerhalb des Clients ein Cache angelegt, in dem die IDs der Nachrichten aus den letzten zehn Minuten gespeichert werden. Erhält der Client eine neue Nachricht, wird vor der weiteren Verarbeitung geprüft, ob die Nachricht innerhalb der letzten zehn Minuten schon einmal verarbeitet wurde.

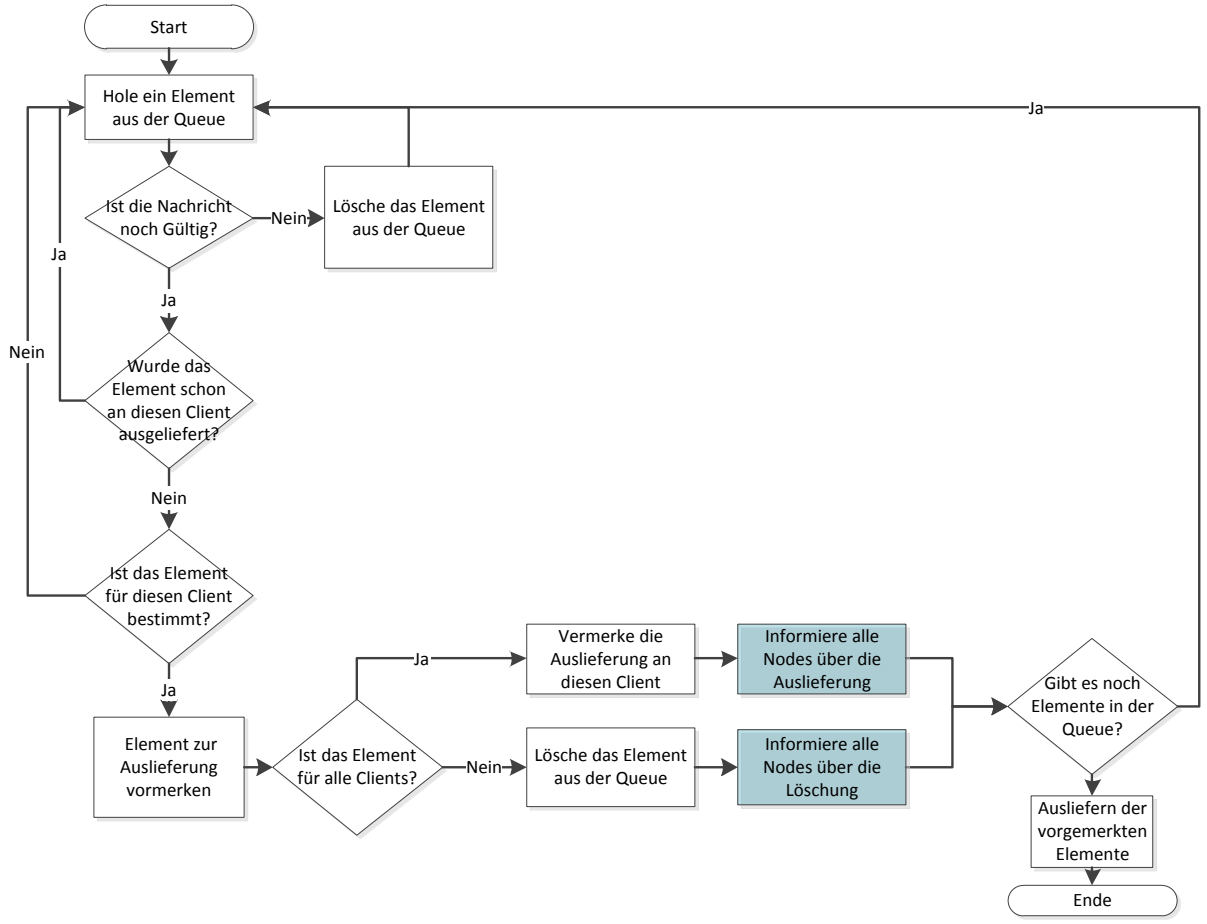

Abb. 4.7: UML Klassendiagramm - ClientNotificationQueue

Für das Versenden der Nachrichten über neue, geänderte und zu löschende QueueElemente wurden drei Nachrichten-Typen angelegt. Alle drei Nachrichten-Typen werden vom `DistributedClientNotificationListener` entgegengenommen und an die entsprechenden Methoden der `ClientNotificationQueue` weitergeleitet. Weiterhin wurde das Interface `IClientNotification` angepasst. Diesem wurden Methoden zum Setzen und Entnehmen der ID des Server-Nodes, der die Nachricht entgegengenommen und die Queue gelegt hat und des Server-Nodes, der die Nachricht letztendlich an den Client ausgeliefert hat. Hierzu wurde die Klasse `AbstractClientNotification` mit den entsprechenden Attributen ergänzt.

Synchronisation des CodeTypeStore

Im CodeTypeStore werden, wie in Abschnitt 2.1.4 beschrieben, CodeTypes gecached. Um alle Server-Instanzen über Änderungen an den CodeTypes zu informieren, wurde die Notification `UnloadCodeTypeCacheNodeNotification` angelegt. Diese kann eine Liste der geänderten CodeTypes enthalten und wird vom `UnloadCodeTypeCacheNodeNotificationListener` über den NotificationService entgegen genommen. Dieser Listener ruft die Methode `reloadCodeTypesInternal` des CodeTypeStore auf. Die Methode `reloadCodeTypesInternal` setzt den internen Cache entsprechend zurück und erstellt eine ClientNotification ausschließlich in der internen Queue. Das hat den Vorteil, dass die Anzahl an versendeten Nachrichten so gering wie möglich gehalten wird.

Synchronisation des AccessControlStore

Ebenfalls im Abschnitt 2.1.4 wurde die Speicherung von Berechtigungen im AccessControlStore angesprochen. Um alle Server-Instanzen über Änderungen an den Berechtigungen zu informieren, wurde die Notification `AccessControlChangedNodeNotification` angelegt. Diese kann eine Liste der geänderten Berechtigungen enthalten und wird vom `AccessControlChangedNodeNotificationListener` über den NotificationService entgegen genommen. Dieser Listener ruft die Methode `clearCacheInternal` des AccessControlStore auf. Diese setzt den internen Cache entsprechend zurück und erstellt eine ClientNotification ausschließlich in seiner internen Queue. Analog zur Lösung des `CodeTypeStore` hat diese Lösung dem Vorteil, dass so wenige Nachrichten wie möglich versendet werden.

4.2 Anpassungen an der Demo-Applikation

Neben den Anpassungen an dem Framework sind auch Anpassungen an der Demo-Applikation BahBahChat notwendig. Um die im Abschnitt 4.1 beschriebenen Änderungen am Eclipse Scout Framework in der BahBah-Chat Applikation nutzen zu können, muss, bis die Funktionalität im Luna Release verfügbar ist, die erstellte Update-Site in das Target-File der Demo-Applikation eingetragen werden. Die erforderlichen Einträge hierfür sind im Listing A.1, im Anhang A.3, zu sehen.

Die BahBahChat Applikation setzte bisher auf eine integrierte Derby Datenbank, um die Chat-User zu speichern. Die Nutzung von DerbyDB ist in keiner PaaS Cloud möglich. Aus diesem Grund wurde die DerbyDb durch MySQL abgelöst. Hierfür wurde der Eclipse Scout JDBC-Adapter ebenfalls über das Target-File eingebunden und der MySQL Adapter in der Manifest als benötigtes Bundle hinzugefügt. Im gleichen Zug wurde der `DerbySqlService` durch den `MySqlSqlService` abgelöst. Die BahBahChat Applikation prüft bei jedem Start der Applikation, ob die Tabelle `tabusers` bereits existiert. Ist dies nicht der Fall, wird diese angelegt und der Benutzer `admin` mit dem Passwort `admin` angelegt. Die hierzu notwendigen SQL-Befehle wurden auf die MySQL Syntax umgestellt. Zusätzlich wird der Benutzer `user` mit dem Passwort `user` angelegt, damit direkt nach dem Start der Applikation mit dem Chat zwischen zwei Benutzern begonnen werden kann.

Damit der NotificationService gestartet wird, muss dieser in der `ServerApplikation` angestoßen werden. Hierfür wurde ein Job implementiert, der dies erledigt. Weiterhin wurde die `ServerSession` der BahBahChat Applikation erweitert, sodass die angesprochene `BackendSession` erstellt werden kann.

Im Abschnitt 2.1.4 wurde die Datenhaltung innerhalb des Applikationsservers angesprochen. Im Abschnitt 4.1.2 wurde dies für den Frameworkanteil mit Hilfe von Nachrichten gelöst. Die BahBahChat Applikation speichert ebenfalls Daten zwischen. In dem Service `UserProcessService` wird ein Set von den aktuell am Server angemeldeten Nutzern gespeichert. Um allen Server-Instanzen eine neue Anmeldung mitzuteilen, wurde die `RegisterUserNotification` angelegt, die nach erfolgreicher Anmeldung an die Message Queue verschickt wird. Über den Listener `RegisterUserNotificationListener` werden die Nachrichten empfangen und die neuen User dem `UserProcessService` hinzugefügt. Analog dazu werden Nutzer, die sich abmelden über die `UnregisterUserNotification` und den `UnregisterUserNotificationListener` wie-

der aus den `UserProcessService` Services aller Server-Instanzen gelöscht. Die Listener werden beim Start der `ServerApplication` auf dem `NotificationService` registriert.

Ebenso mussten Anpassungen bezüglich des Caching durchgeführt werden. Der `BasicForwardSecurityFilter` der BahBahChata Applikation griff direkt auf die HTTP-Session zu. Der Zugriff auf HTTP-Session ist jedoch nicht mehr erlaubt, da diese mit dem Service `ICacheStoreService` abgelöst wurde. Der `BasicForwardSecurityFilter` wurde entsprechend angepasst.

Damit in der Applikation sichtbar wird welche Nachricht von welcher Server-Instanz entgegen genommen und wieder ausgeliefert wurde, wurden der Tabelle mit den Chat-Nachrichten zwei neue Spalten hinzugefügt. Die Spalten enthalten die ersten fünf Zeichen der ID der Server-Instanz sowie eine farbliche Markierung zur besseren optischen Darstellung. Die Abbildung 4.8 zeigt den Swing-Client des Benutzers admin im Chat mit dem Benutzer user. Wie an den Spalten Receiving Node und Providing Node zu sehen ist, wurde die erste Nachricht von der Server-Instanz mit der ID 1b78c an den Client ausgeliefert obwohl diese von dem Client des Benutzer user an die Server-Instanz mit der ID 25d25 ausgeliefert wurde. Ähnlich sieht es bei der zweiten Nachricht aus, die vom Benutzer user dem Benutzer admin geschickt wurde. Diese wurde von der Server-Instanz mit der ID 1b78c entgegengenommen und von der Server-Instanz mit der ID 65e0e an den Client des Benutzers admin ausgeliefert. Das zeigt, dass die Clients mit mehreren Server-Instanzen kommunizieren und es keine Rolle dabei spielt welche Server-Instanz die Nachrichten entgegen nimmt oder verteilt. Alle Nachrichten werden über alle Server-Instanzen synchronisiert.

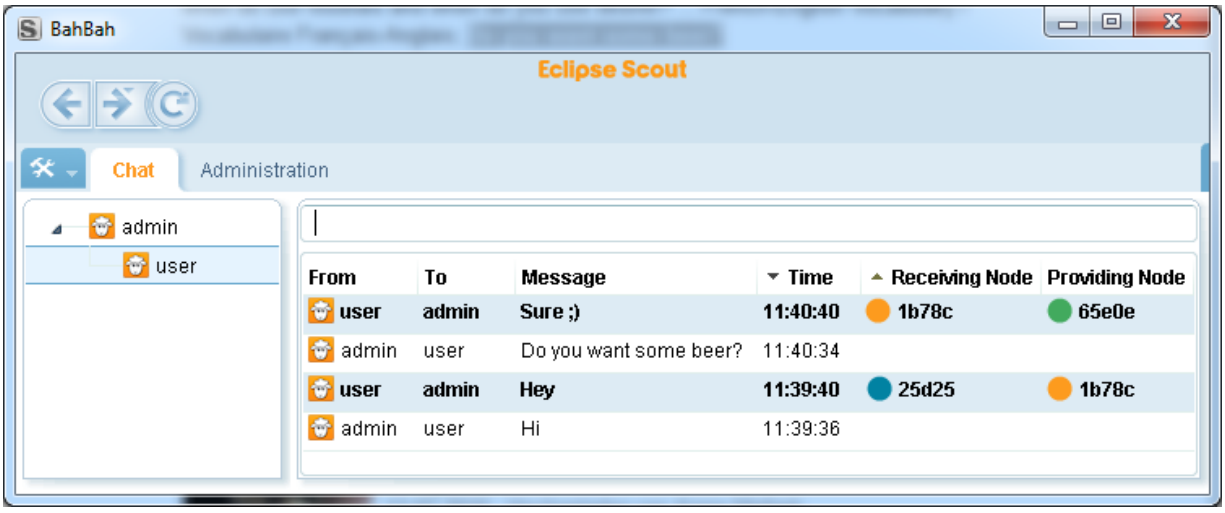

Abb. 4.8: BahBahChat - Chat Dialog

4.3 Service Implementierungen

Um die im Abschnitt 4.1 beschriebenen Anpassungen am Eclipse Scout Framework nutzen zu können, müssen Implementierungen für den Cache und Message Service erstellt werden. Im Rahmen dieser Masterthesis wird für jeden Service eine Default-Implementierung angefertigt, die im Single-Node-Betrieb eingesetzt werden können. Für Amazon Elastic Beanstalk und CloudBees Run@Cloud müssen Implementierungen für Memcached und RabbitMQ erstellt werden. Zusätzlich zu diesen Implementierungen werden Implementierungen für Redis und JMS konstruiert. Redis wird von vielen PaaS Anbietern und bietet somit eine gute Alternative zu Memcached. JMS wird unter den PaaS Anbietern zwar nur von Jelastic genutzt, doch in Cluster-Umgebungen ist JMS häufig im Einsatz. Zudem ist mit den zusätzlichen Implementierungen kein großer Aufwand verbunden.

Das Ziel ist es die Implementierungen für die PaaS spezifischen Services nicht direkt in Scout zu integrieren, sondern diese über den Eclipse Marketplace zur Verfügung zu stellen. Das hat den Vorteil, dass das Framework für Single-Node-User keinen unnötigen Ballast enthält und Änderungen oder Erweiterungen wesentlich flexibler eingearbeitet werden können. Um die in dieser Masterthesis erstellten Implementierungen zu beziehen, wurde die Update-Site `http://tools.bsiag.com/marketplace/cloud` erstellt (Vgl. Abb. 4.9). Diese ist zwar noch nicht vom Eclipse Marketplace indexiert, stellt aber bereits die Implementierungen für Memcached, Redis, RabbitMQ und ActiveMQ zur Verfügung. Die Implementierungen für den Singe-Node-Betrieb hingegen werden direkt im Eclipse Scout Framework integriert, damit die Applikationen auch ohne Erweiterungen betrieben werden können. Um die konkreten Cache und Messages Services ansprechen zu können, müssen die Verbindungs- und Authentifizierungsparameter in der Conig.ini der Server-Applikation eingetragen werden. Sind diese Daten vor der Installation nicht verfügbar sondern erst innerhalb der Laufzeitumgebung, so kann ein neuer Service erstellt werden, der den alten Service erweitert aber die Parameter selbst setzt. Dies ist zum Beispiel bei Cloud Foundry basierten PaaS Angeboten notwendig, da diese die Verbindungs- und Authentifizierungsdaten in einem JSON-Objekt in einer Umgebungsvariable ablegen. Damit der neu implementierte Service anstelle des erweiterten Services genutzt wird, muss dieser lediglich mit einer höheren Priorität in der Plugin.xml der Server-Applikation eingetragen werden.

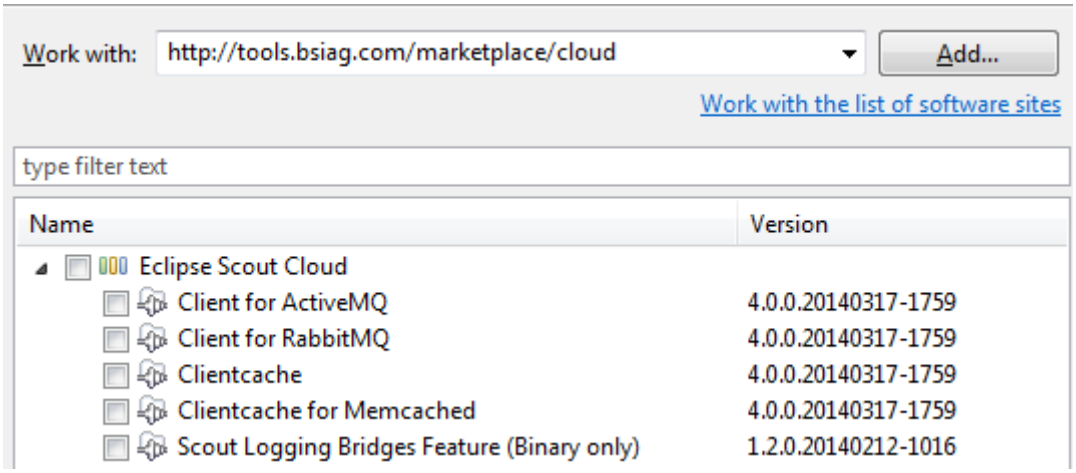

Abb. 4.9: Update Site - Service Implementierungen

4.3.1 Single-Node-Cache

Für den Single-Node-Betrieb wurde die Klasse DefaultCacheService angelegt. Diese nutzt zum cachen der clientbezogenen Daten weiterhin die HTTP-Session. Die Daten, die unabhängig vom Client gecached werden sollen, werden in einer Objekt-Liste innerhalb des Cache-Services abgelegt. Um auch bei den Daten, die in der internen Objekt-Liste gespeichert werden, eine zeitliche Gültigkeit festlegen zu können wurde der Datentyp CacheElement angelegt. Diese Klasse ist mit ihren Attributen und Methoden in der Abbildung 4.10 dargestellt. Die Default-Implementierung des Cache Service wird vom Eclipse Scout Framework in der OSGi Service Registry ohne Angabe eines Rang registriert. Das hat den Vorteil, dass im Single-Node-Betrieb keine weitere Konfiguration notwendig ist. Sobald die Applikation mit mehreren Server-Instanzen betrieben werden soll, wird der Default-Service mit einer Implementierung, die mit einem höheren Rang registriert wurde, überschrieben.

org.eclipse.scout.rt.server.commons::**CacheElement**
-m_value : Object -m_creationTime : Long -m_expiration : Long
+CacheElement(eing. value : Object, eing. expiration : Integer, eing. creationTime : Long) : CacheElement +Operation1(eing. value : Object, eing. expiration : Integer) : CacheElement +getCreationTime() : Long +getValue() : Object +isActive() : Boolean +resetCreationTime() +setExpirationTime(eing. expiration : Integer)

Abb. 4.10: Sequenzdiagramm Notifications

4.3.2 Single-Node-Messages

Für den NotificationService ist keine gesonderte Implementierung für den Single-Node-Betrieb notwendig. Dieser prüft bei der Initialisierung, ob eine Implementierung von IPubSubMessageService über die OSGi Service Registry verfügbar ist. Ist dies nicht der Fall, werden Nachrichten, die über den NotificationService publiziert werden, einfach verworfen. Weitere Mechanismen zur Verarbeitung von Messages im Single-Node-Betrieb sind nicht notwendig, da zum einen keine Nachrichten versendet werden können und zum anderen durch das Fehlen der Implementierungen von IPubSubMessageServic kein Service mehr die Verarbeitung von Nachrichten auf dem NotificationService anstößt.

4.3.3 Memcached

Für die Implementierung des Memcached Cache Service wurde ein eigenes Bundle mit dem Namen `org.eclipse.scout.cloud.cachestore.memcached` erstellt. Der Zugriff auf Memcached erfolgt über die Bibliothek `net.spy.memcached`[1]. Diese Bibliothek wird über die Update-Site der 3rdParty Tools[2] bereitgestellt. Für die Konfiguration der Parameter müssen folgende Angaben in der `Config.ini` gemacht werden:

- org.eclipse.scout.cloud.cachestore.MemcachedCacheService#url=
- org.eclipse.scout.cloud.cachestore.MemcachedCacheService#port=
- org.eclipse.scout.cloud.cachestore.MemcachedCacheService#auth=
- org.eclipse.scout.cloud.cachestore.MemcachedCacheService#username=
- org.eclipse.scout.cloud.cachestore.MemcachedCacheService#password=

Bei der Implementierung des Memcached Services wurde die Methode `touch` verwendet. Diese ist erst ab der Memcached Version 1.4.2 verfügbar. Alle betrachteten PaaS Anbieter konnten diese Anforderung erfüllen.

1 `https://code.google.com/p/spymemcached`
2 `https://scout.bsiag.com/repository/tools.3rdParty/`

4.3.4 Redis

Für die Implementierung des Redis Cache Service wurde ein eigenes Bundle mit dem Namen `org.eclipse.scout.cloud.cachestore.redis` erstellt. Für den Zugriff auf Redis wird die Bibliothek jedis[1] genutzt. Für die Konfiguration der Parameter müssen folgende Angaben in der `Config.ini` gemacht werden:

- org.eclipse.scout.cloud.cachestore.redis.RedisCacheService#host=
- org.eclipse.scout.cloud.cachestore.redis.RedisCacheService#port=
- org.eclipse.scout.cloud.cachestore.redis.RedisCacheService#auth=
- org.eclipse.scout.cloud.cachestore.redis.RedisCacheService#password=

4.3.5 RabbitMQ

Für die Implementierung des RabbitMQ Notification Service wurde ein eigenes Bundle mit dem Namen `org.eclipse.scout.cloud.notification.rabbitmq` erstellt. Für den Zugriff auf RabbitMQ wird die Bibliothek `jedis`[2] genutzt. Für die Konfiguration der Parameter müssen folgende Angaben in der `Config.ini` gemacht werden:

- org.eclipse.scout.cloud.notification.rabbitmq.RabbitMQMessageService#host=
- org.eclipse.scout.cloud.notification.rabbitmq.RabbitMQMessageService#auth=
- org.eclipse.scout.cloud.notification.rabbitmq.RabbitMQMessageService#user=
- org.eclipse.scout.cloud.notification.rabbitmq.RabbitMQMessageService#password=

Pro Server-Instanz werden zwei Verbindungen zum RabbitMQ Server aufgebaut. Eine, um Nachrichten zu publizieren und eine, um Nachrichten zu empfangen. Dies ist wichtig, wenn RabitMQ als Service, z. B. von CloudAMQP, bezogen wird. Dort sind die Angebote in der Anzahl der zugelassenen Verbindungen reglementiert. Bei Tests mit CloudAMQP ist weiterhin aufgefallen, dass bei längeren Verbindungen Timeouts auftreten. Aus diesem Grund wird die Verbindung zum Versenden der Nachrichten immer nur dann aufgebaut,

1 `https://github.com/xetorthio/jedis`
2 `https://www.rabbitmq.com/java-client.html`

wenn eine Nachricht versendet werden muss. Unmittelbar danach wird die Verbindung wieder getrennt.

4.3.6 ActiveMQ

Für die Implementierung des ActiveMQ Notification Service wurde ein eigenes Bundle mit dem Namen `org.eclipse.scout.cloud.notification.rabbitmq` erstellt. Für den Zugriff auf ActiveMQ wird die Bibliothek `activemq-all-5.9.0.jar`[1] genutzt. Für die Konfiguration der Parameter müssen folgende Angaben in der `Config.ini` gemacht werden:

- org.eclipse.scout.cloud.notification.activemq.ActiveMQMessageService#host=
- org.eclipse.scout.cloud.notification.activemq.ActiveMQMessageService#port=
- org.eclipse.scout.cloud.notification.activemq.ActiveMQMessageService#password=
- org.eclipse.scout.cloud.notification.activemq.ActiveMQMessageService#username=
- org.eclipse.scout.cloud.notification.activemq.ActiveMQMessageService#auth=

4.4 Deploy der BahBahChat Applikation

In dem Abschnitt 3.4 wurden zwei Anbieter ausgewählt, für die, im Rahmen dieser Masterthesis, Implementierungen für die angebotenen Cache und Message Services erstellt und die Beispiel Applikation BahBahChat betrieben werden soll. Die Ausgewählten PaaS Angebote sind Amazon Elastic Beanstalk und CloudBees Run@Cloud. Die Implementierungen der konkreten Services, die zum Betrieb der Applikation in den ausgewählten Angeboten notwendig ist, wurden bereits im Abschnitt 4.3 beschrieben. In diesem Abschnitt geht es darum, wie die Applikation bei den einzelnen Anbietern betreiben werden kann und was dabei zu beachten ist. Zuvor wird noch erläutert, wie die Demo-Applikation in einer lokalen Umgebung getestet werden kann.

1 `http://activemq.apache.org/activemq-590-release.html`

4.4.1 Lokale Umgebung

Um die Skalierung der BahBahChat Applikation lokal testen zu können, wurden, neben dem ursprünglichem Product-File `bahbah-server-dev.product`, drei weitere Product-Files angelegt. Diese starten die Server-Applikation jeweils einmal hinter den Ports 8081, 8082 und 8083. Zur gebündelten Nutzung dieser drei Server-Instanzen ist ein Load Balancer notwendig. Eine einfache Möglichkeit bietet hier der Apache HTTP-Server. Mit der Virtual Host Konfiguration aus dem Listing 4.1 kann dieser als Load Balancer eingestellt werden. Zudem muss das Modul `mod_proxy` in der `httpd.conf` aktiviert werden. Der Apache HTTP-Server verteilt somit die eingehenden Anfragen der Reihe nach an die eingetragenen Server-Instanzen. Die Anzahl der eingetragenen Server-Instanzen ist hierbei nicht beschränkt.

```
<VirtualHost *:80>
  ProxyRequests off
  ServerName localhost
  <Proxy balancer://cluster>
    BalancerMember http://localhost:8081
    BalancerMember http://localhost:8082
    BalancerMember http://localhost:8083
  </Proxy>
  ProxyPass / balancer://cluster/
</VirtualHost>
```

List. 4.1: Apache - Load Balancer Konfiguration

Auch in der lokalen Umgebung müssen ein Datenbank, ein Cache und ein Message Queue Service angebunden werden, damit die Applikation fehlerfrei arbeitet. Die zu verwendende Datenbank kann direkt bei MySQL[1] heruntergeladen werden. Installationen für den lokalen Betrieb eines Redis-Server können von redis.io[2] bezogen werden. Soll Memcached lokal genutzt werden, so kann entweder Memcached[3] direkt genutzt werden oder alternativ CouchBase[4]. Möchte man keine Software installieren, so können SaaS Angebote genutzt werden, wie zum Beispiel redislabs, MemCachier oder CloudAMQP.

1 http://dev.mysql.com/downloads/
2 http://redis.io/download
3 http://memcached.org/
4 http://www.couchbase.com/download

4.4.2 Amazon Elastic Beanstalk

Amazon stellt alle sein Cloud Dienste im ersten Jahr, innerhalb eines begrenzen Kontingents, kostenlos bereit. Hierfür werden pro EC2 Service 750 Stunden pro Monat zur Verfügung gestellt[1], was für den Dauerbetrieb einer Einheit eines Services vollkommen ausreicht. Nach der Anmeldung kann der Nutzer über ein übersichtliches Web-Frontend neue Laufzeitumgebungen zu seinem Account hinzufügen. Beim Anlegen der Laufzeitumgebung kann bequem über ein Formular die gewünschte Plattform konfiguriert und ein WAR-File angeben werden, welches auf der Umgebung deployed werden soll. Beim dem Deploy der BahBahChat Applikation auf Amazon Elastic Beanstalk gab es einige Schwierigkeiten. Das von Eclipse Scout bereitgestellte Process-Servlet[2] konnten nicht aufgerufen werden. Stattdessen wurde die Anfrage vom Tomcat mit einer 404 Statusmeldung quittiert. Die Fehlersuche gestaltete sich sehr schwierig und langwierig, da in den Log-Files keine Unstimmigkeiten gefunden werden konnten. Über die Aktivierung der Equinox "Framework Controlls"konnte festgestellt werden, dass die OSGi Implementierung fehlerfrei läuft. Um festzustellen welche Bundles Aktiv sind, wurde die Apache Felix Webconsole in das BahBahChat Projekt eingebunden und beim Start von OSGi geladen. In der Webconsole war zu sehen, dass die Bundles `org.eclipsescout.demo.bahbahchat.server` und `org.eclipsescout.demo.bahbahchat.shared` den Status Resolved hatten und damit nicht aktiv waren. Bei dem Versuch die Bundles über die Webconsole zu starten wurde ein Stacktrace in den Logfiles erzeugt, welcher auf die falsche Java-Version hinwies. Das Updaten der Java-Version einer Amazon Elastic Beanstalk Instanz ist über eine Konfigurationsdatei möglich, die innerhalb des WAR-Files im Ordner `.ebextensions` abgelegt und in die Dateiendung `.config` haben muss. Um das Oracle JDK 7 nutzen zu können, muss die Konfiguration aus dem Listing A.2 verwendet werden. Für die Nutzung von OpenJDK 7 kann die Konfiguration aus dem Listing A.4 verwendet werden. Hierbei ist zu beachten, dass als Betriebssystem Linux in der 64 Bit Variante eingesetzt werden muss. Dieser Fehler ist aufgetreten, da bei der Einrichtung der Laufzeitumgebung keine Angabe über die Tomcat oder Java-Version gemacht wurde und Amazon per Default Java 1.6 verwendete. Diese wurde zwischenzeitlich von Amazon umgestellt. Elastic Beanstalk verwendet nun Tomcat 7 und Java 1.7 bei der Erstellung der Laufzeitumgebung. Eine Auswahl der Version ist jedoch weiterhin nicht möglich. Der Zugriff auf die einzelnen Ser-

1 `http://aws.amazon.com/de/free/`
2 `http://scoutcloud.elasticbeanstalk.com/process`

vices von Amazon bedarf etwas Konfigurationsarbeit, denn die Zugriffe auf die einzelnen Services sind über sogenannte Security Groups geregelt. Soll von einem Service auf einen andern zugegriffen werden, muss sich der zugreifende Service in einer Security Group befinden, die bei dem Service, auf den zugegriffen werden soll, als berechtigt eingetragen ist.

Cache

Als Cache-Service kann Amazon ElastiCache verwendet werden. Dieser bietet die Auswahl zwischen Redis und Memcached. Der Service kann nach belieben skaliert werden. Hier habe ich mich für Memcache entschieden, da somit die Services für Amazon und CloudBees identisch sind, was den Entwicklungsprozess stark vereinfacht. Um den Zugriff von der EC2-Instanz, auf der die Server-Applikation läuft, auf den Memcached Cache zu erlauben, muss die Security Group der EC2-Instanz dem Cache hinzugefügt werden.

Message Queue

Die von Amazon angebotene Message Queue mit dem Namen SQS ist alleine nicht in der Lage das Publish/Subscribe Pattern zu implementieren. Kombiniert man den SQS mit einem weiteren Service von Amazon, dem SNS, so kann das Publish/Subscribe Pattern implementiert werden, allerdings nicht flexibel. Aus diesem Grund habe ich mich entschieden, das Angebot von CloudAMQP zu nutzen. CloudAMQP stellt RabbitMQ als Service in der Amazon Cloud bereit.

Datenbank

Eine MySQL Datenbank wird bei Amazon über den AWS RDS Service angeboten. Mit Hilfe eines Wizards kann die Datenbank detailliert konfiguriert werden. Auch hier ist der Zugriff über die Security Groups reglementiert. Sollen innerhalb der Amazon Landschaft weitere Services, wie zum Beispiel die EC2 Instanz auf die Datenbank zugreifen, so muss die Security Group, in der die EC2 Instanz steckt, der Datenbank hinzugefügt werden. Soll von außerhalb der Amazon Landschaft, etwa vom PC des Entwicklers aus, auf die Datenbank zugegriffen werden, so muss die IP des Internetanschlusses der Security Group zugefügt werden.

4.4.3 CloudBees Run@Cloud

CloudBees bietet für das Laufzeitangebot Run@Cloud vier sogenannte AppCells kostenlos an, in denen jeweils eine Applikation laufen kann. Soll eine AppCell skaliert werden, muss auf einen bezahlten Plan umgestiegen werden. Neue AppCells können per Web-Wizard, CloudBees SDK, Eclipse Toolkit oder einem eigenem ClickStack erstellt werden. Am schnellsten geht es mit dem Web-Wizard. Bei CloudBees muss lediglich über ein Webformular die Plattform ausgewählt und ein Namen für die Applikation vergeben werden und schon ist die Umgebung einsatzbereit. Das WAR-File kann, wie bei dem Angebot von Amazon, bequem über ein Web-Formular deployed werden. Im Gegensatz zu Amazon hat man bei CloudBees zusätzlich noch die Möglichkeit, eine Jenkins-Instanz zu konfigurieren, die den Source-Code aus einem Git-Repository lädt, baut und anschließend automatisch deployed. Das ist ein Alleinstellungsmerkmal von ClouBees und wird von vielen Entwicklern sehr geschätzt.

Cache

CloudBees bietet mit dem Session Store die Möglichkeit, alle HTTP-Sessions über alle Instanzen zu synchronisieren. Die Demo Applikation BahBahChat könnte also ohne ein zusätzliches Cache-Bundle bei CloudBees deployed werden und würde dennoch korrekt skalieren. Dieser Service kostet jedoch 40$ pro Monat. Da ich davon ausgehe, dass viele Eclipse Scout Entwickler sich erst einmal einen Einblick verschaffen und dafür kein Geld ausgeben wollen, habe ich mich entschieden, hier die von CloudBees angebotene Integration von MemCachier zu nutzen[1]. MemCachier bietet den Cache-Service Memcached als Cloud-Lösung an.

Message Queue

Auch bei der Message Queue hat CloudBees keine eigene Lösung, bietet hier jedoch eine Integration von CloudAMQP über das CloudBees Ecosystem an. Auch die Anbindung von CloudAMQP kann komfortabel über ein Webformular erledigt werden. Automatisch wird ein Account bei CloudAMQP angelegt und die Zugangsdaten bereitgestellt.

1 `http://wiki.cloudbees.com/bin/view/RUN/Memcache`

Datenbank

Bei der Datenbank ist die Konfiguration ähnlich einfach. Über ein Web-Formular kann ausgewählt werden, wie groß die Datenbank sein soll und unter welchem Namen diese verwaltet wird. Anschließend können die Zugangsdaten zu der MySQL-Datenbank entweder über das Web-Frontend abgefragt oder in der laufenden Applikation über den Context ausgelesen werden.

5 Skalierungstest

Im vorhergehenden Kapitel 4 wurde beschrieben, welche Anpassungen am Eclipse Scout Framework vorgenommen wurden, um damit erstellte Applikationen skalierbar zu machen. Weiterhin wurde beschrieben, wie die Demo-Applikation BahBahChat angepasst werden musste und wie diese bei den beiden PaaS Angeboten deployed werden kann. In diesem Kapitel geht es darum, wie sich die BahBahChat Applikation unter Last verhält und ob eine Skalierung der Applikation auch eine Verbesserung der Performance mit sich bringt. Hierfür sollen Lasttests durchgeführt werden. Laut Daniel Menascé helfen Lasttests bei der Messung des Quality of Service (QoS) der Performance basierend auf tatsächlichen Kundenverhalten (Men02, S. 2). Als relevante Messgrößen sieht er hier die Antwortzeit der Software unter einer bestimmten Anzahl der von gleichzeitig zugreifenden Benutzer.

5.1 Testgrundlage

Da die Verwendung von Rich-Clients für die Lasttests, wegen der Installationsarbeiten und der Test-Steuerung, sehr aufwändig ist, wurde ein Servlet erstellt, dass eine typische Kommunikation eines Clients mit dem Server simulieren soll. Einer der gängigsten Use-Cases von Eclipse Scout Anwendungen ist, dass ein Service Daten aus der Datenbank holt, diese aufbereitet und dem Client zur Darstellung übermittelt. Zusätzlich dazu soll die Skalierbarkeit des Caches und der Message Queue getestet werden. Hierzu wurde der BahBahChat Applikation ein neues Servlet hinzugefügt. Dieses holt aus einer Datenbanktabelle 100 Datensätze a 10 Spalten, wobei jede Zelle einen String mit 36 Zeichen enthält. Diese insgesamt 1000 Strings werden im Anschluss alphabetisch sortiert in einem Set abgelegt. Danach werden weitere 250 Strings, die ebenfalls aus 36 Zeichen bestehen, aus dem verteilten Cache geladen. Diese 1250 Strings werden mit jeweils 4 Zeichen HTML-Code

zur Formatierung angereichert und von dem Servlet ausgegeben. Damit erzeugt das Servlet bei jedem Aufruf eine Datenlast von etwa 50KB. Zudem wird bei jedem Aufruf des Servlets eine Nachricht mit der ID der Server-Instanz an die restlichen Server-Instanzen versendet. Diese zählen intern jede empfangene Nachricht mit, sodass jede Server-Instanz die Anzahl der Requests aller Server-Instanzen kennt. Tritt ein Fehler bei der Bearbeitung des Servlet auf, wird das Servlet nicht mit dem Response-Code 200 abschließen. Der Erfolg der Lasttests hängt nicht nur von der Zeit ab, die eine Server-Instanz braucht, um die Anfragen zu beantworten, sondern auch von der Korrektheit der Antwort. Ob die Datenbank- und Cache-Abfrage korrekt funktioniert, kann zum einen am Response-Code des Servlets geprüft werden und zum anderen an der Größe der übermittelten Daten. Die korrekte Übermittlung der Nachrichten kann mit Hilfe der Zählung der Nachrichten festgestellt werden. Sind die gezählten Requests pro Server-Instanz aus allen Instanzen identisch, so kann davon ausgegangen werden, dass alle Nachrichten korrekt versendet und verarbeitet wurden.

5.2 Testszenarien

Um belastbare Ergebnisse zu Erhalten werden die Lasttests mit zwei verschiedenen Tools durchgeführt, für die jeweils eigene Testszenarien erstellt werden. Das erste Tool ist der Clouddienst von Blitz.io, mit dem Lasttests direkt aus der Cloud heraus gefahren werden können. Das zweite Tool ist die Software LoadUI, mit der Lasttests von der eigenen Maschine aus gestartet werden können. Mit der Software LoadUI wurden Probemessungen durchgeführt, um Anhaltspunkte für die maximale Anzahl an Requests herauszufinden, die eine Server-Instanz verarbeiten kann, damit die im Folgenden beschriebenen Testszenarien mit aussagekräftigen Größen arbeiten können. Hierzu wurde die Anzahl der gleichzeitigen Requests pro Sekunde stetig erhöht, bis die Applikation keine stabilen Antwortzeiten mehr liefern konnte. Dies wurde bei 15 Requests pro Sekunde erreicht. Für die Tests mit Blitz.io wurden zwei Testszenarien erstellt, die in der Tabelle 5.1 zu sehen sind. Sie unterscheiden sich lediglich in der Anzahl der eingesetzten Server-Instanzen. Die Anzahl der gleichzeitigen Requests zum Start und zum Ende des Tests können bei beiden Szenarien gleich bleiben, da Blitz.io die Anzahl der gleichzeitigen Requests im Laufe des Tests linear erhöht. Kostenbedingt werden nur zwei Durchläufe pro Szenario durchgeführt. Die Anzahl der User am Ende des Tests wurde bei beiden Testszenarien gleich hoch gesetzt um die Ergebnisse besser vergleichen zu können. Die Höhe der User wurde auf das Fünffache

der maximalen Kapazität einer einzelnen Instanz gesetzt und anschließend nochmals um ein Drittel erhöht, um auch eine Überforderung der fünf Server-Instanzen zu provozieren (15*5 = 75; 75 * (4/3) = 100). Zudem wird ein Timeout von 10 Sekunden eingestellt, was Requests abbricht, die länger als diese Zeit dauern.

Nr.	Anbieter	Instanzen	Dauer	Start User	Ende User	Timeout
B1	Amazon	1	60s	1	100	10s
B2	Amazon	5	60s	1	100	10s

Tab. 5.1: Testszenarien - LoadUI

Für die Tests mit LoadUI wurden vier Testszenarien erstellt, die in der Tabelle 5.2 zu sehen sind. Sie unterscheiden sich in der Anzahl der eingesetzten Server-Instanzen und der Anzahl der Requests, die pro Sekunde auf den Load Balancer abgesetzt werden. Hierbei werden auf einer Server-Instanz erst 15 Requests pro Sekunde abgesetzt und im Anschluss daran 20 Requests pro Sekunde. Das Gleiche, wird nochmals mit fünf gleichzeitig betriebenen Server-Instanzen wiederholt, mit 75 Requests pro Sekunde beim ersten Test und mit 100 Requests pro Sekunde beim zweiten Test. Insgesamt werden bei jedem Testdurchlauf 150 Requests abgesetzt. Die untere Anzahl der Requests pro Sekunde, die von einer Instanz verarbeitet werden muss, wurde anhand der zuvor beschriebenen Probemessungen festgelegt. Die obere Anzahl beträgt vier Drittel der unteren Anzahl, um zu zeigen dass die Instanz mit der Verarbeitung von mehr Anfragen überfordert ist. Das bedeutet, wenn maximal etwa 15 Requests pro Sekunde von einer Instanz verarbeitet werden können, so sollte die Instanz mit 20 Requests pro Sekunde überfordert sein. Bei einer Skalierung auf 5 Instanzen wird erwartet, dass 75 Requests pro Sekunden von 5 Instanzen verarbeitet werden können. Bei 100 Requests pro Sekunde sollte auch diese nicht mehr mit der Verarbeitung nachkommen.

Test-Nr.	Anbieter	Instanzen	Requests/Sekunde	Requests Gesamt
L1	Amazon	1	15	150
L2	Amazon	1	20	150
L3	Amazon	5	75	150
L4	Amazon	5	100	150

Tab. 5.2: Testszenarien - LoadUI

Die Applikationsserver der BahBahChat Demo-Applikation laufen bei Amazon auf EC2 Instanzen der Größe t1.micro. Die Rechenkapazität dieser Maschine wird mit bis zu zwei EC-Recheneinheiten angegeben.

5.3 Messsoftware und Umgebung

Wie zuvor bereits erwähnt, kommen zwei verschiedene Tools zur Durchführung der Lasttests zum Einsatz. Die besonderen Eigenschaften dieser beiden Tools werden im Folgenden näher beschrieben.

5.3.1 Blitz.io

Gemessen wird die Antwortzeit mit der Software Blitz[1] der Firma Spirent Communications. Blitz ist ein Lasttest Dienst aus der Cloud für die Cloud. Die Software wird als SaaS angeboten und bietet Last- und Performancetests mit bis zu 50'000 Usern. Getestet werden können Webseiten, Web-Applikationen und Web-APIs. Um mit Blitz diese Dienste testen zu können, muss zuvor die Domain autorisiert werden. Dies ist notwendig, um Missbrauch zu vermeiden. Hierzu muss eine Webseite auf der Domain eingerichtet werden die über `http://meine.domain/[BlitzAccountID]` zu erreichen ist und als Antwort auf alle Fragen, 42 zurück gibt. Alternativ kann auch die Index-Seite mit dem Meta-Tag `<meta name="blitz"content="[BlitzAccountID]»` ausgestattet werden. Um die entsprechenden Domains `http://scoutcloud.elasticbeanstalk.com` und `http://appserver.scoutcloud.cloudbees.net` zu autorisieren, wurde ein WAR-File erstellt, welches eine entsprechende Datei enthält. Dieses wurde bei den beiden Anbietern deployed und bei Blitz.io über das Webformular aus der Abb. 5.1 autorisiert.

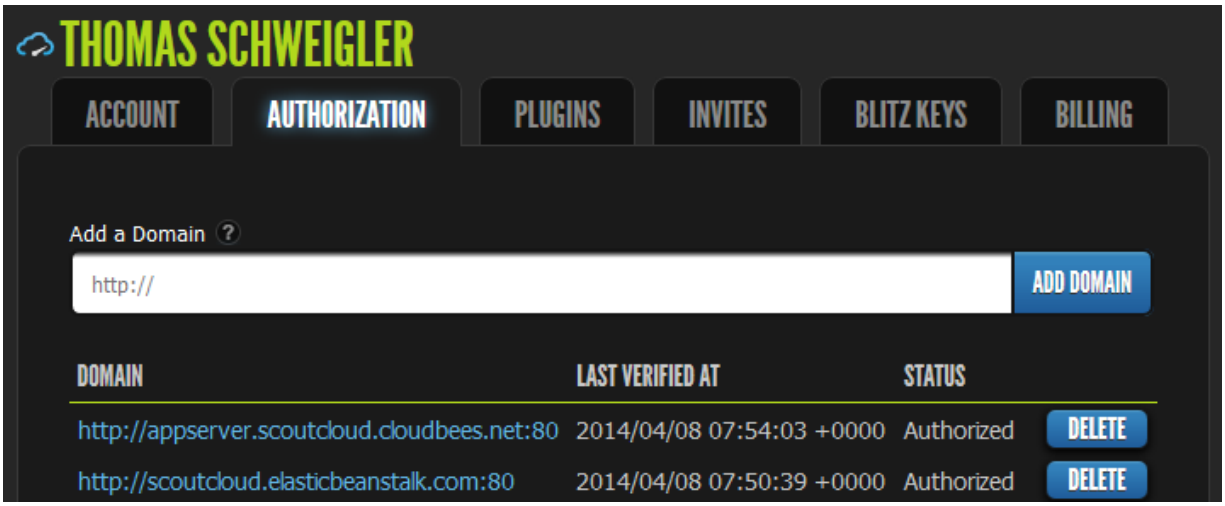

Abb. 5.1: Blitz - Autorisieren der Domains

1 `https://www.blitz.io/`

5.3.2 LoadUI

Die Software LoadUI der Firma SmartBear Software ist eine klassische Desktop-Applikation. Die Testszenarien können bei LoadUI bequem per Drag&Drop und der Verknüpfung von Komponenten zusammen gestellt werden (vgl. Abb. 5.2). Hier stehen z. B. verschiedene Last-Generatoren zur Verfügung, die nach unterschiedlichen Kurven die Last generieren. Eine Authentifizierung der zu testenden URLs ist bei LoadUI nicht notwendig. Bei der Verwendung von LoadUI kommt es stark auf die Umgebung an, in der die Software betrieben wird. Mit einer Installation der Software auf einer Maschine im Firmennetzwerk der BSI AG konnten keine belastbaren Ergebnisse hinsichtlich der Skalierung erzielt werden. Gründe hierfür können Latenzen zwischen Europa und der USA sein. Mit einer Installation der Software auf einer VM in der Amazon Cloud konnten hingegen plausible Resultate erzielt werden. Hierfür wurde eine Instanz einer m3.large Maschine in der Amazon EC2 Cloud mit Windows Server 2012 für die Tests eingerichtet und mit der Testsoftware LoadUI bespielt.

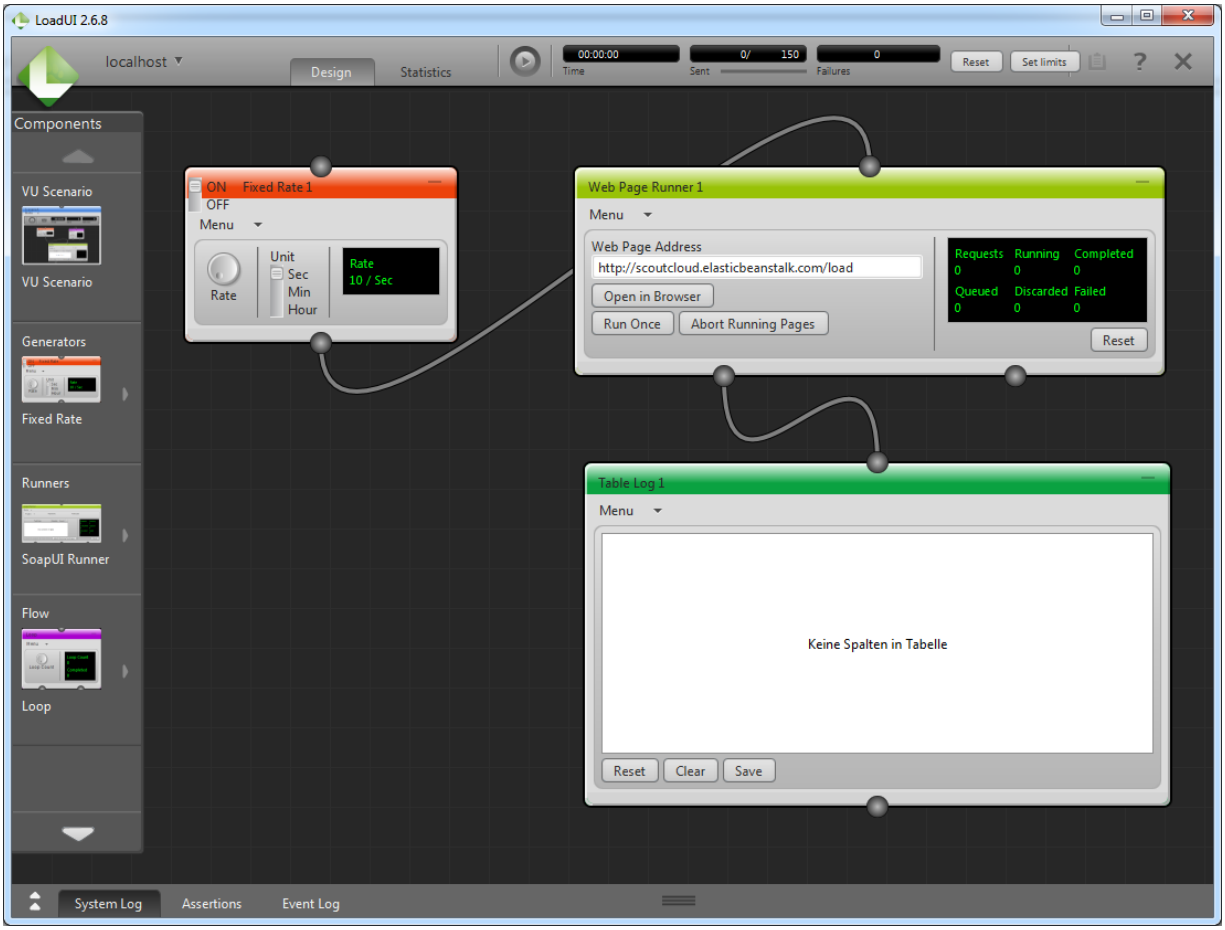

Abb. 5.2: LoadUi - Konfiguration

5.4 Messung

Dieser Abschnitt beschreibt die Messungen und ihre Ergebnisse, unterteilt nach den beiden verwendeten Test-Tools.

5.4.1 Blitz.io Messungen

Die Messungen mit Blitz.io wurden für jedes Szenario zweimal wiederholt. Bei den Messergebnissen liefert Blitz.io alle 2,5 Sekunden einen Statusbericht. Dieser enthält die aktuelle Anzahl an Usern, die gleichzeitig auf die Applikation zugreifen sowie die Anzahl der Hits, Timeouts und Errors, die bis dahin abgeschlossen wurden bzw. aufgetreten sind. Eine Auflistung der Ergebnisse ist im Anhang A.6 zu finden. Ebenfalls wird die durchschnittliche Zeit angegeben, die für die in den letzten 2,5 Sekunden abgeschlossenen Requests benötigt wurde. Da bei keiner der Messungen Errors erzeugt wurden, sind diese in keiner der folgenden Messungen beschrieben. Die in diesem Abschnitt dargestellten Diagramme enthalten auf der linken Y-Achse immer den aktuell betrachtete Einheit, auf der rechten Y-Achse die Anzahl der gleichzeitig zugreifenden User und auf der X-Achse die Zeitintervalle. Um die Tendenzen der Messergebnisse übersichtlicher darzustellen, wurden aus den beiden Messungen, die für jedes Testszenario durchgeführt wurden, ein Mittelwert gebildet.

Die folgende Abbildung 5.3 zeigt die Antwortzeiten, die beim den beiden Messungen des Testszenario B1 erzielt wurden. Gut zu sehen ist, dass zum Anfang der Messung die Antwortzeiten deutlich unter einer Sekunde liegen. Ab etwa der 15. Sekunde und 25 gleichzeitigen Usern erhöht sich die gemessene Antwortzeit und liegt deutlich über einer Sekunde.

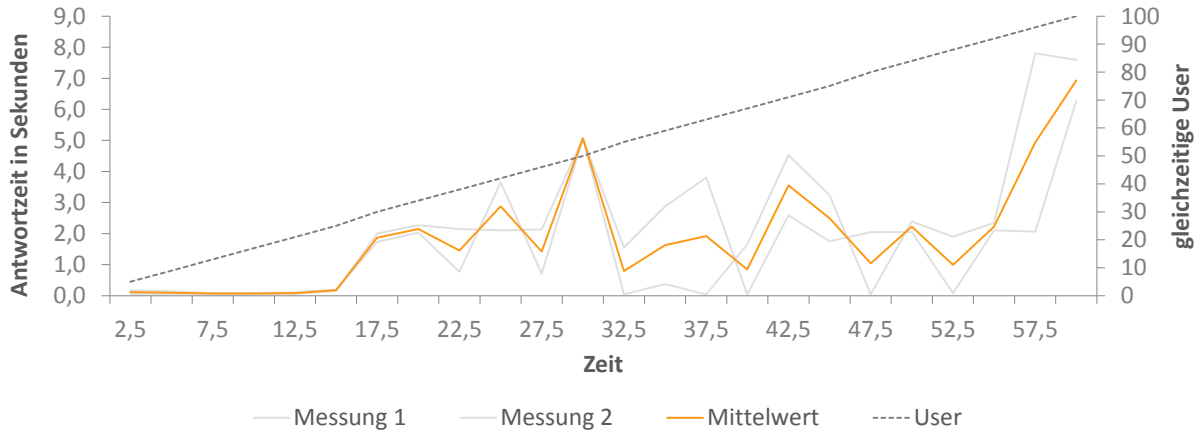

Abb. 5.3: Blitz.io - Antwortzeit - 1 Instanz

Die in der Abbildung 5.3 dargestellten Messergebnisse sind alleine jedoch nur wenig aussagekräftig, da Antwortzeiten von über 10 Sekunden, aufgrund des eingestellten Timeouts, nicht berücksichtigt werden. Um hier Aussagen über die Belastbarkeit der Applikation treffen zu können, müssen die erzielten Hits und Timeouts pro Sekunde mit in die Betrachtung einbezogen werden. Die Hits pro Sekunde, dargestellt in der Abbildung 5.4, steigen zu Anfang stark an, um anschließend wieder stark zu fallen.

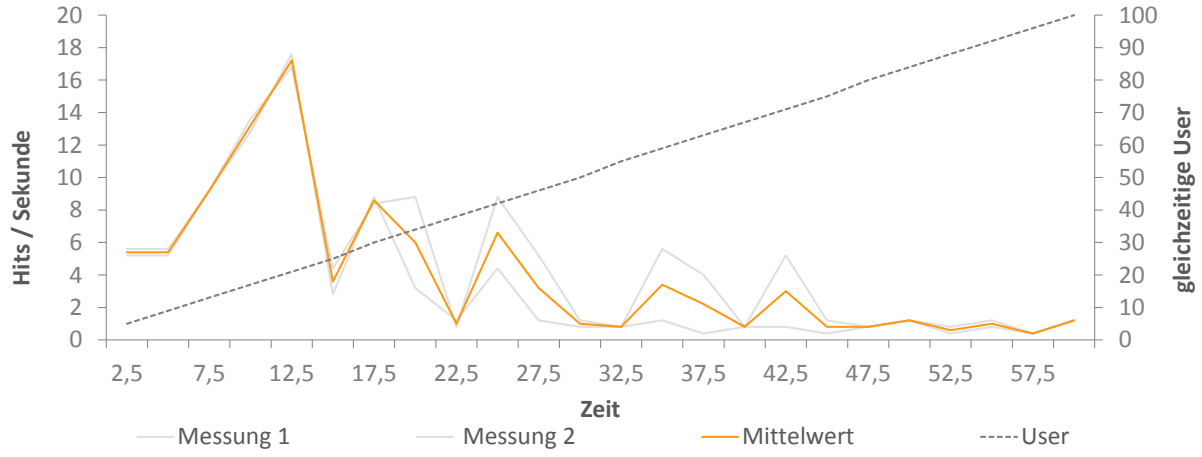

Abb. 5.4: Blitz.io - Hits / Sekunde - 1 Instanz

Bei den Timeouts verhält es sich umgekehrt. Wie in der Abbildung 5.5 zu sehen, treten bis zum Zeitpunkt 22,5 bei beiden Messungen keine Timeouts auf. Anschließend werden jedoch in jedem Zeitintervall Timeouts verzeichnet, wobei sich die Anzahl der Timeouts pro Sekunde in der Tendenz immer weiter erhöht.

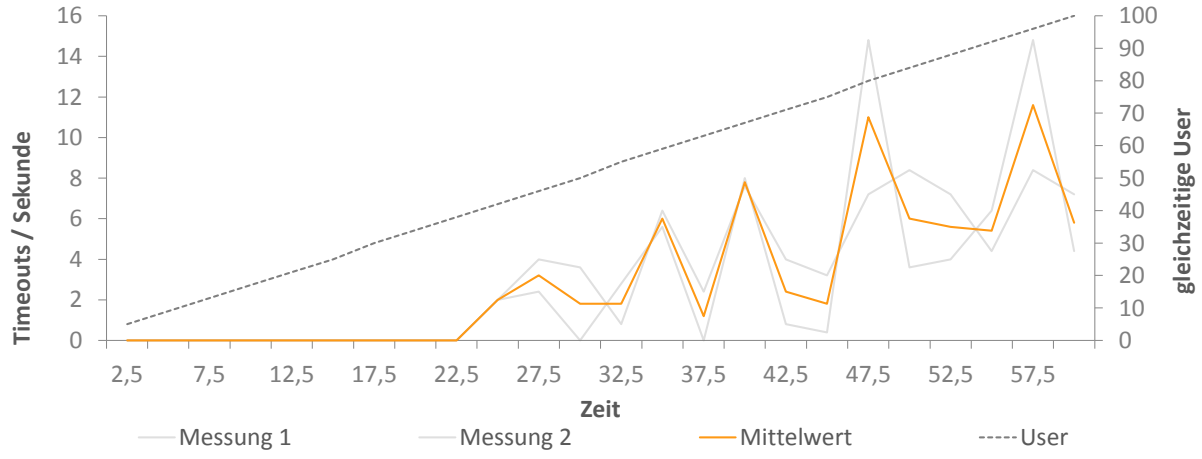

Abb. 5.5: Blitz.io - Timeouts / Sekunde - 1 Instanz

In der Abbildung 5.6 sind die Antwortzeiten aus den beiden Messungen des Testszenario B2 zu sehen. Diese liegen bis zum Zeitpunkt 52,5 mit 88 gleichzeitigen Usern stetig unter 200 Millisekunden. Im Anschluss daran steigt die Antwortzeit an. Im Mittel auf bis zu 1,12 Sekunden.

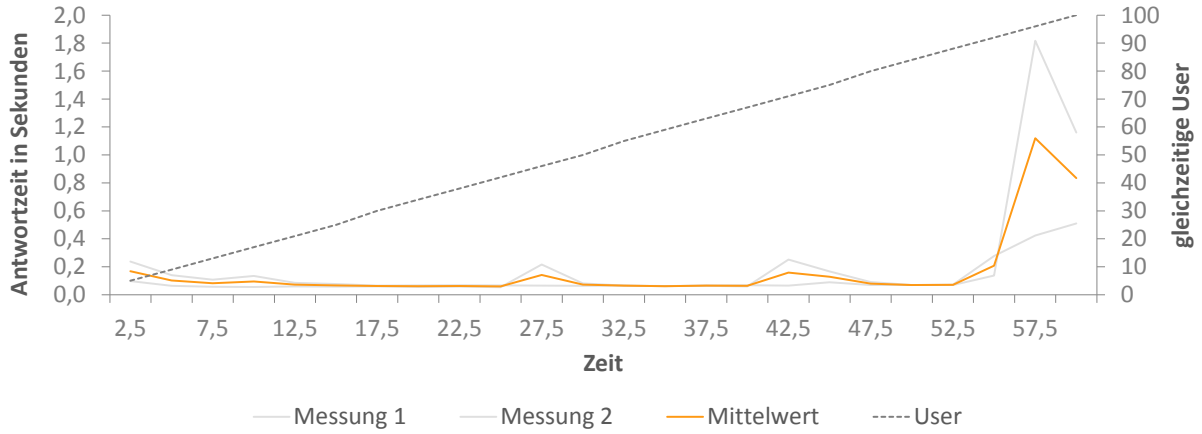

Abb. 5.6: Blitz.io - Timeouts / Sekunde - 1 Instanz

Die Hits pro Sekunde, dargestellt in der Abbildung 5.7, steigen hier ebenfalls bis zum Zeitpunkt 52,5. Danach fallen die Hits pro Sekunde stark ab. Auf eine Darstellung der Timeouts pro Sekunde kann für das Testszenario B2 verzeichnet werden, da während der Messung keine Timeouts aufgetreten sind.

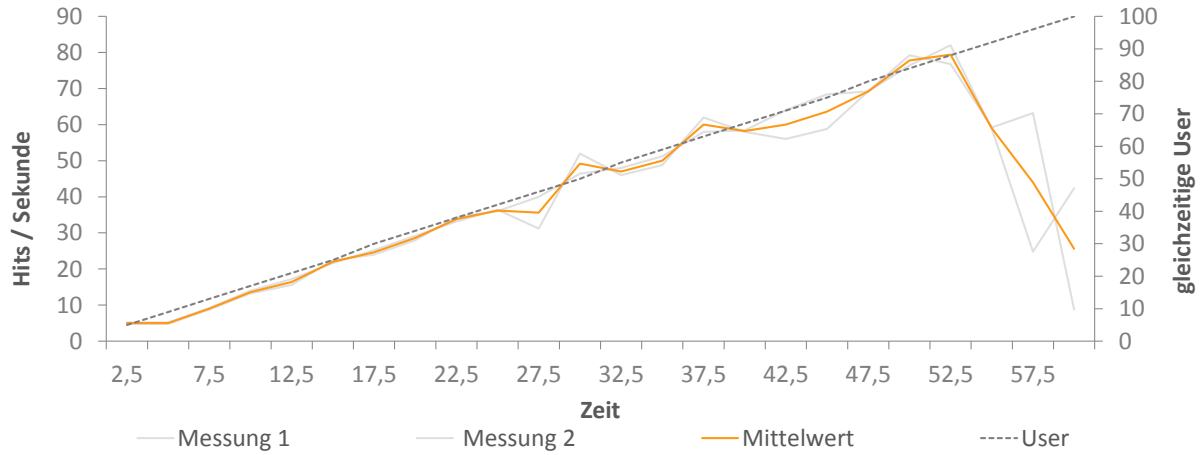

Abb. 5.7: Blitz.io - Timeouts / Sekunde - 1 Instanz

5.4.2 LoadUI Messungen

Im Folgenden werden die Messergebnisse der aufgestellten Testszenarien beschrieben. Jedes Testszenario wurde fünf mal wiederholt. Die Ergebnisse sind im Anhang im Abschnitt A.7 aufgelistet. Die hier dargestellten Diagramme sind auf der X-Achse mit den einzelnen Requests belegt und auf der Y-Achse mit der Antwortzeit in Millisekunden. Die Ergebnisse der Messungen sind mit grauen Linien dargestellt, der berechnete arithmetische Mittelwert ist mit einer orangen Linie gezeichnet.

Die Messergebnisse des Testszenario L1 sind in der Abbildung 5.8 zu sehen. Diese zeigt eine Abarbeitung der 150 Requests, welche zum Großteil unter 100ms stattfindet. Lediglich einige Ausreißer in der 2. Messung heben den Mittelwert der Antwortzeit kurzfristig über 100ms. Das arithmetische Mittel für diese Messung über alle Requests liegt bei 53,83ms bei einer Standardabweichung von 37,32.

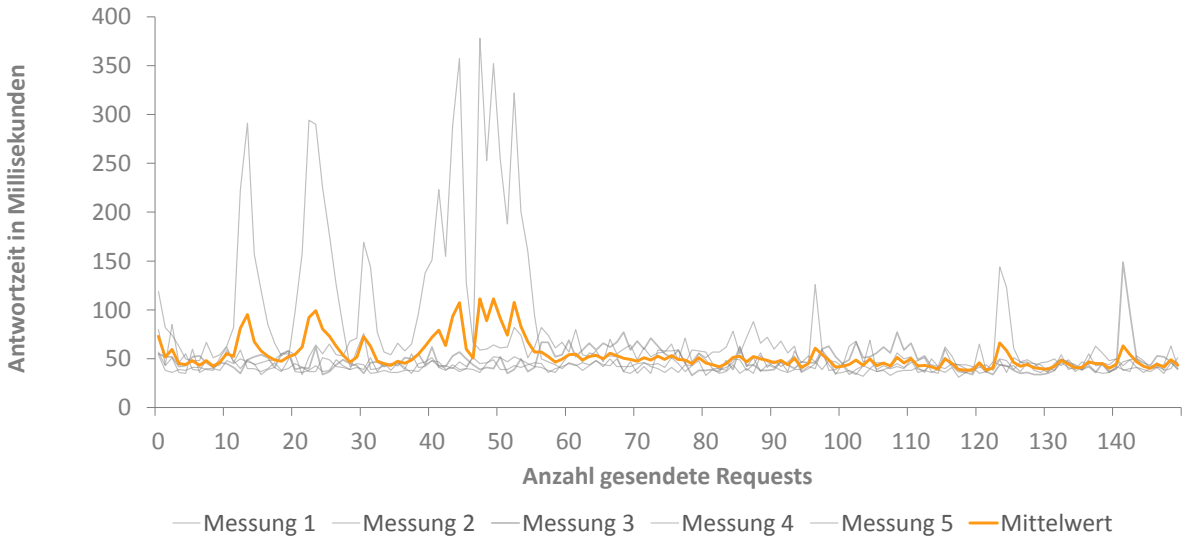

Abb. 5.8: Messergebnisse - Testszenario L1

Die Messergebnisse des Testszenario L2 sind in der Abbildung 5.9 zu sehen. Gut zu sehen ist, dass die eingehenden Requests von der Server-Instanz anfangs noch schnell beantwortet werden können. Ab dem 25. Request kam die Server-Instanz bei allen 5 Messungen nicht mehr mit der Bearbeitung hinterher und die Antwortzeiten wurden immer länger. So pendelt sich diese zwischen dem 70. und 115. Request bei etwa 10 Sekunden ein, um danach auf ungefähr 16 Sekunden anzusteigen. Das arithmetische Mittel für diese Messung über alle Requests liegt bei 6960,95 ms bei einer Standardabweichung von 5835,91.

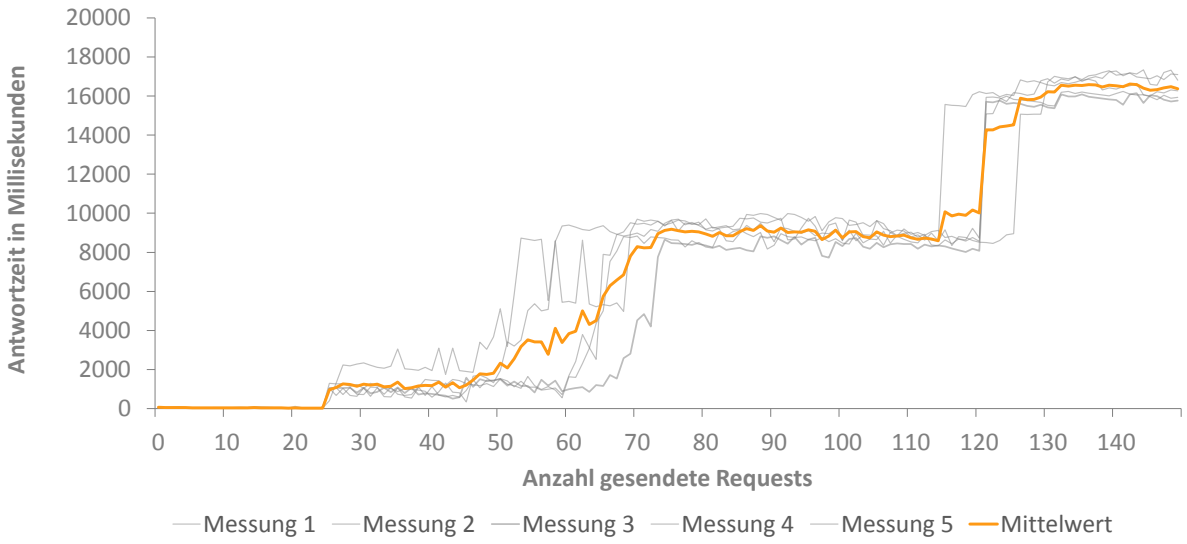

Abb. 5.9: Messergebnisse - Testszenario L2

Die Abbildung 5.10 zeigt die Messergebnisse des Testszenarios L3. Bis auf zwei Abweichungen in der ersten Messung, liegen alle Antwortzeiten zwischen 20 und 100 Millisekunden. Es lässt sich sogar eine leicht fallende Tendenz erkennen. Das arithmetische Mittel liegt bei 53,89ms bei einer Standardabweichung von 15,29.

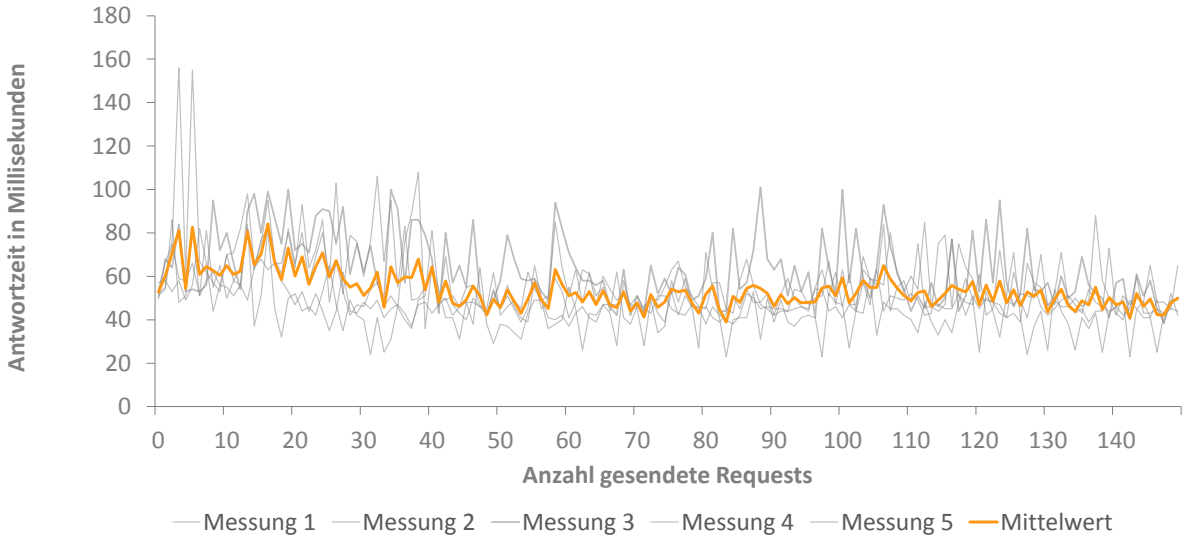

Abb. 5.10: Messergebnisse - Testszenario L3

In der Abbildung 5.11 sind die Messergebnisse des Testszenario L4 zu sehen. Diese zeigt, dass die Anfragen, die an die fünf Server-Instanzen gestellt wurden, bis zum 125. Request im Mittel in 72,39ms (σ 35,35) bearbeitet wurden. Nach dem 125. Request steigt die Antwortzeit jedoch rapide bei allen fünf Messungen auf über eine Sekunde. Das arithmetische Mittel über alle 150 Requests hinweg liegt bei 210,62ms bei einer Standardabweichung von 344,09.

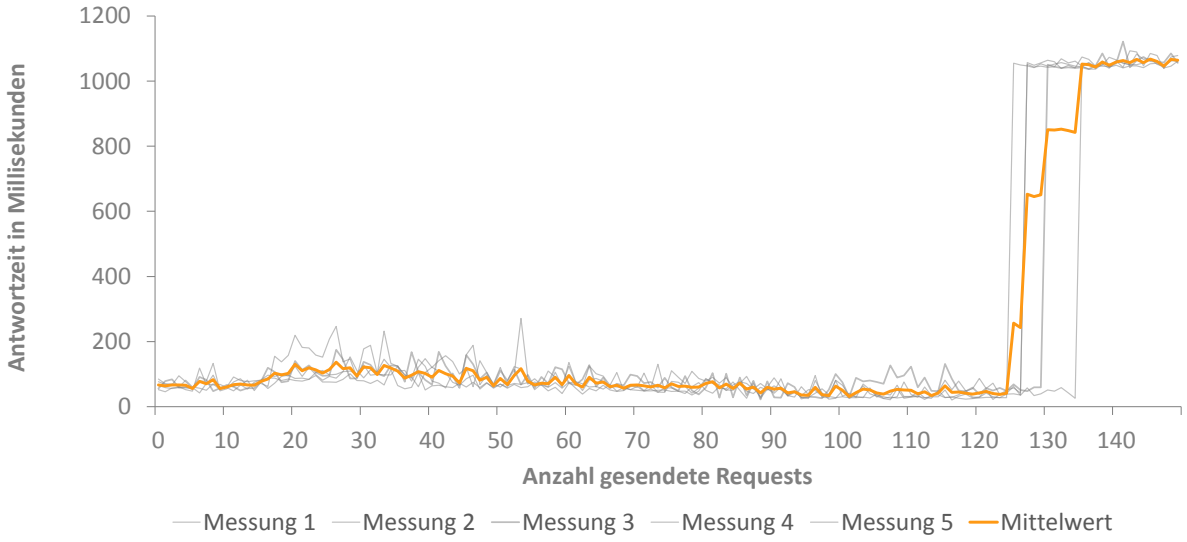

Abb. 5.11: Messergebnisse - Testszenario L4

Bezüglich der Korrektheit der Antwort der Server-Instanzen, konnte den Messergebnissen entnommen werden, dass alle gestellten Anfragen mit dem Status-Code 200 und einer Datenmenge von 50331 Bytes beantwortet wurden. Zudem konnten auch alle versendeten Nachrichten korrekt empfangen und weiterverarbeitet werden. Dies wurde überprüft, indem nach jedem Testdurchlauf ein Servlet aufgerufen wurde, dass die Anzahl der eigenen Aufrufe sowie die Anzahl der Aufrufe für jede andere Instanz auflistet. Auf eine Einbindung dieser Servlet-Ausgaben wurde verzichtet, da diese für alle durchgeführten Durchläufe, bis auf die generierten Node-Ids, nicht unterscheiden. Eine exemplarische Servlet-Ausgabe ist im Anhang A.8 zu finden.

5.5 Auswertung

Die Messungen mit Blitz.io haben gezeigt, dass eine Server-Instanz große Probleme hat Anfragen von mehr als 25 gleichzeitigen Usern zu beantworten. Nicht nur die erreichten Antwortzeiten sind angestiegen, sondern auch die Anzahl der aufgetretenen Timeouts. Wurde allerdings die Anzahl der Server-Instanzen auf fünf skaliert, konnten die Anfragen von bis zu 88 gleichzeitigen Usern mit Antwortzeiten von unter 200 Millisekunden bearbeitet werden. Timeouts traten hier keine mehr auf.

Ähnlich sieht es bei dem Messungen mit der Software LoadUI aus. Die hier durchgeführten Messungen haben gezeigt, dass eine Server-Instanz in der Lage ist 15 Requests pro Sekunde zuverlässig abzuarbeiten, bei 20 Requests pro Sekunde jedoch nicht mehr mit der Verarbeitung hinterher kommt, was sich über stark steigende Antwortzeiten bemerkbar macht. Fünf Server-Instanzen hingegen sind in der Lage 75 Requests pro Sekunde zuverlässig zu bearbeiten. Bei einer Anzahl von 100 Requests pro Sekunde geraten jedoch auch die fünf Server-Instanzen in Bedrängnis. Dies ist an der sprunghaft steigenden Antwortzeit ab dem 125. Request in der Abbildung 5.9 zu sehen.

Abschließend kann auf Basis der im vorhergehenden Abschnitt 5.4 durchgeführten Lasttests der Schluss gezogen werden, dass Applikationen, die auf dem angepassten Eclipse Scout Framework basieren, skalierbar sind und auch unter Last korrekt funktionieren.

6 Fazit und Ausblick

Nachdem sich die vorangegangenen Kapitel mit der Konzeption, Implementierung und der Skalierungsmessung befasst haben, werden in diesem Kapitel die erreichten Ergebnisse dieser Arbeit diskutiert und ein Ausblick auf mögliche zukünftige Entwicklungen gegeben.

6.1 Ergebnisse dieser Arbeit

Im Rahmen dieser Masterthesis ist es gelungen das Eclipse Scout Framework so anzupassen, dass damit Applikationen erstellt werden können, die sich ohne großen Aufwand auf verschiedenen PaaS Clouds betreiben und automatisch skalieren lassen. Um dies zu erreichen, wurde das bestehende Eclipse Scout Framework untersucht und mit Cloud-Systemen und ihren Eigenschaften verglichen. Auf dieser Basis wurden Anforderungen definiert, die eine PaaS Cloud erfüllen muss, um eine Eclipse Scout Applikation skalierbar zu betreiben. Anhand dieser Kriterien wurden verschiedene PaaS Anbieter analysiert und miteinander verglichen. Von diesen PaaS Anbietern sind Amazon, CenturyLink, CloudBees, Hivext, Microsoft, Oracle, Pivotal und Salesforce theoretisch in der Lage, Eclipse Scout Applikationen skalierbar zu betreiben. Für einen Test wurden Amazon und CloudBees ausgewählt.

Bei der Untersuchung des Eclipse Scout Frameworks wurde festgestellt, dass innerhalb der Server-Applikation verschiedene Daten zwischen gespeichert werden. Um diese Daten effizient zwischen den einzelnen Instanzen der Server-Applikation zu synchronisieren, wurde ein verteilter Cache und eine Message Queue in die Architektur von Eclipse Scout Applikationen aufgenommen. Sowohl ein verteilter Cache als auch eine Message Queue werden von den meisten PaaS Anbietern angeboten. Jedoch unterscheiden sich diese in ihrer Ausführung und Kombination. Um die mit Eclipse Scout erstellten Applikationen dennoch unabhängig von einer bestimmten PaaS Cloud zu machen, wurde die serviceorientierte Architektur von Eclipse Scout genutzt. Innerhalb des Eclipse Scout Frameworks werden,

für die Kommunikation mit dem Cache und der Message Queue, Interfaces genutzt, für die die jeweiligen Implementierungen einfach, abhängig von der genutzten PaaS Cloud, als Plugin hinzugefügt werden können. Mit der angepassten Architektur sind die Entwickler von Eclipse Scout Applikationen in der Lage, nicht nur die im Rahmen dieser Masterthesis erstellen Implementierungen für die Cache Services Memcached und Redis sowie die Message Queues RabbitMQ und ActiveMQ zu nutzen, sondern sie können genauso einfach ihre eigenen Implementierungen nutzen. Somit ist das Eclipse Scout Framework vollkommen unabhängig von den in der Laufzeitumgebung zur Verfügung stehenden Implementierungen des synchronisierten Caches und der Message Queue.

Meiner Meinung nach gewinnt das alte Paradigma „Program to an interface, not an implementation“ (GHJV98, S. 30) der Gang of Four (GoF), im Umfeld der Cloud, enorm Bedeutung. Bei der Heterogenität und Volatilität von Cloud Plattformen ist es essenziell sich nicht auf die spezifischen Implementierungen einzelner Services festzulegen, sondern ein allgemeingültiges Interface zu definieren. Wer sich vor einem Vendor lock-in schützen möchte, muss entweder auf Standards setzen oder seine Architektur flexibel halten.

6.2 Ausblick

Mit der Fähigkeit Eclipse Scout Applikationen skalierbar in verschiedenen PaaS Clouds zu betreiben, wurde lediglich der Grundstein gelegt. Einer der nächsten Schritte wird sein, den RAP-Webserver ebenfalls skalierbar in einer PaaS Cloud zu betreiben. Somit könnten nicht nur die Rich-Clients von der Skalierbarkeit profitieren, sondern auch die Web-Clients.

Auch mit dem Betrieb der Applikation in einer klassischen gehosteten PaaS Clouds ist noch nicht das Ende der Möglichkeiten erreicht. Eine Evaluierung von sogenannten Stack Angeboten, wie z. B. Docker, wäre der nächste logische Schritt. Diese verpacken die Applikationen samt aller ihrer Abhängigkeiten in einen Container. Diese Prinzip ähnelt dem der virtuellen Maschinen, hat jedoch den großen Vorteil, dass wesentlich weniger Festplatten- und Arbeitsspeicher benötigt wird. Ebenso können die Container schneller hochgefahren werden als die virtuellen Maschinen. Statt also ein vollständiges Betriebssystem zu emulieren, werden isolierte Laufzeitumgebungen, wie z. B. LinuX-Containern (LXC), innerhalb eines Betriebssystems genutzt.

Bezüglich der Service Implementierungen steht noch die Anbindung von IronMQ aus. Hier wurde bereits ein Lösungsansatz mit dem CTO von IronMQ, Travis Reeder, erarbeitet. Dieser konnte jedoch im Rahmen dieser Masterthesis nicht umgesetzt werden. Die Schwierigkeit bei IronMQ liegt darin, dass die Message Queue alle Subscriber kennen muss, um die Nachrichten ausliefern zu können. Die Server-Instanzen befinden sich jedoch hinter einem Load Balancer und können nicht direkt angesprochen werden. Für die Lösung dieser Herausforderung kann die Fähigkeit von IronMQ hilfreich sein Queues als Subscriber anderer Queues einzutragen. Hierfür würde jede Server-Instanz beim Starten eine eigene Queue anlegen und diese bei der Haupt-Queue als Subscriber eintragen. Neue Nachrichten werden auf der Haupt-Queue publiziert und von dieser an die Queues der einzelnen Server-Instanzen verteilt. Diese können die Nachrichten von ihrer eigenen Queue abrufen. Um verweiste Queues von heruntergefahrenen Instanzen zu bereinigen, muss regelmäßig eine Überprüfung der aktiven Instanzen und eine Löschung der übrigen Queues erfolgen. Dieses Prinzip sollte auch bei Amazon mit einer Kombination von SQS und SNS zu einer flexiblem Publish/Subscribe Message Queue führen. Dies gilt es zu überprüfen.

Bei Geschäftsapplikationen gibt es häufig Aufgaben, die periodisch ausgeführt werden müssen, wie z. B. das Versenden von Rechnungen. Die Abarbeitung dieser Aufgaben werden meist zeitgesteuert ausgelöst. Wird die Server-Applikation skaliert, gibt es mehrere Instanzen der gleichen Applikation. Damit nicht alle Instanzen die gleiche Aufgabe parallel ausführen, müssen sich diese untereinander abstimmen. Um hier die Eclipse Scout Entwickler noch stärker zu unterstützen, könnte im Eclipse Scout Framework ein Mechanismus integriert werden, der die Abarbeitung von zeitgesteuerten Jobs zwischen den einzelnen Server-Instanzen synchronisiert. Denkbar wäre hier ein verzögerter Start der Bearbeitung, mit einer individuellen Verschiebung pro Instanz. Zusätzlich werden vor dem Bearbeitungsbeginn die restlichen Instanzen informiert, dass die Aufgabe von ihnen nicht bearbeitet werden muss. Hierfür könnte ein Datenbank Lock, ein Cache-Eintrag oder eine Nachricht über die Message Queue genutzt werden. Dies gilt es noch genauer zu evaluieren.

A Anhang

A.1 Ausgeschiedene PaaS Anbieter

Die ausgeschiedenen PaaS Anbieter sind in der Tabelle A.1 aufgelistet.

A.2 UML Klassendiagramm - ICacheStoreService

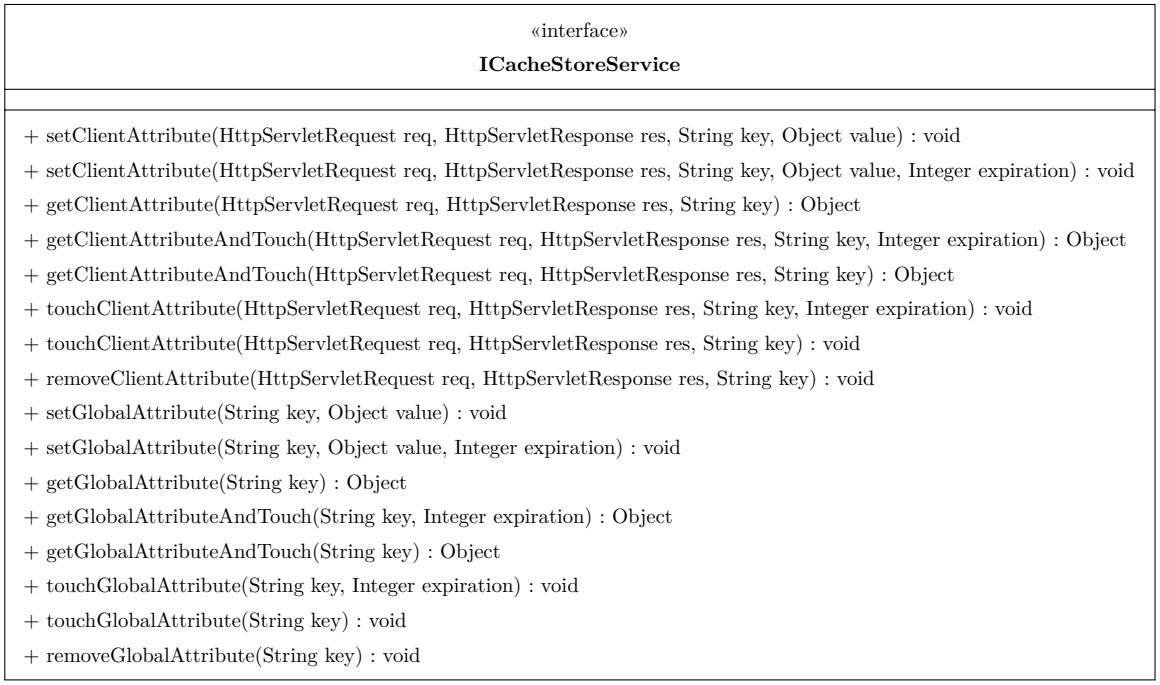

Abb. A.1: UML Klassendiagramm - ICacheStoreService

Unternehmen	Produkt	Grund des Ausscheidens	URL
ActiveState	ActiveState Stackato	Stack Anbieter, kein Public Cloud Hosting	http://www.activestate.com
AppScale, Inc.	AppScale	Private Cloud, kein Public Cloud Hosting	http://www.appscale.com
BitRock Incorporated	BitNami	Stack Anbieter, kein Public Cloud Hosting	http://www.bitnami.com
Contrail Consortium	ConPaaS	Stack Anbieter, kein Public Cloud Hosting	http://www.conpaas.eu
CumuLogic	Cumulogic	Stack Anbieter, kein Public Cloud Hosting	http://www.cumulogic.com
dotCloud	dotCloud	Stack Anbieter, kein Public Cloud Hosting	http://www.dotcloud.com
ElasticBox	ElasticBox	Stack Anbieter, kein Public Cloud Hosting	http://www.elasticbox.com
Fabasoft	Folio Cloud	Kein Public Cloud Hosting	http://www.foliocloud.com
IBM	SmartCloud	Wird eingestellt	http://www.ibm.com
mOSAIC EC	mOSAIC	Stack Anbieter, kein Public Cloud Hosting	http://www.mosaic-cloud.eu
NodeRabbit Inc.	Appsembler	Keine Java-Unterstützung	http://www.appsembler.com
Nubo9, Inc.	Nubo9	Gibt es nicht mehr	-
openlogic	CloudSwing	Stack Anbieter, kein Public Cloud Hosting	http://www.openlogic.com
Pogoapp LLC	Pogoapp	Keine Java Unterstützung	http://www.pogoapp.com
SAP	HANA Cloud Platfrom	Fokus auf eigene Software	http://www.saphana.com
SixSq.	SlipStream	Stack Anbieter, kein Public Cloud Hosting	http://www.sixsq.com
Software AG	LongJump	Kein Public Cloud Hosting eigener Applikationen	http://www.agileappslive.com
Standing Cloud	Standing Cloud	Kein Public Cloud Hosting	http://www.standingcloud.com
Tier3	Iron Foundry	Keine Java-Unterstützung, .NET Fokus	http://www.ironfoundry.org
TwoSpot	TwoSpot	Kein Public Cloud Hosting	http://twospot-paas.blogspot.ch
Uhuru Software	Uhuru PaaS	Fokus auf .NET Unterstützung für CloudFoundry und OpenShift	http://uhurusoftware.com
WSO2	WSO2 StratosLive	Fokus auf die eigene Software	http://wso2.com/cloud/stratoslive/
ActiveState	ActiveState Stackato	Doppelt in der Liste	-
Microsoft	Windows Azure	Doppelt in der Liste	-

Tab. A.1: Ausgeschiedene PaaS Anbieter

A.3 BahBahChat - Target-File

```
<location includeAllPlatforms="false" includeConfigurePhase="false"
    includeMode="slicer" includeSource="true" type="InstallableUnit">
<unit id="org.eclipse.scout.rt.source.feature.group" version="0.0.0"/>
<unit id="org.eclipse.scout.rt.rap.source.feature.group" version="0.0.0"/>
<unit id="org.eclipse.scout.rt.rap.testing.source.feature.group" version="
    0.0.0"/>
<unit id="org.eclipse.scout.rt.testing.source.feature.group" version="0.0.0"
    />
<unit id="org.eclipse.scout.rt.ui.rap.incubator.filechooser.source.feature.
    group" version="0.0.0"/>
<repository location="http://download.eclipse.org/scout/releases/4.0/cloud"/
    >
</location>

<location includeAllPlatforms="false" includeConfigurePhase="false"
    includeMode="slicer" includeSource="true" type="InstallableUnit">
<unit id="org.eclipse.scout.cloud.notification.activemq" version="0.0.0"/>
<unit id="org.eclipse.scout.cloud.cachestore.memcached" version="0.0.0"/>
<unit id="org.eclipse.scout.cloud.cachestore.redis" version="0.0.0"/>
<unit id="org.eclipse.scout.cloud.notification.rabbitmq" version="0.0.0"/>
<repository location="http://tools.bsiag.com/marketplace/cloud"/>
</location>

<location includeAllPlatforms="false" includeConfigurePhase="false"
    includeMode="slicer" includeSource="true" type="InstallableUnit">
<unit id="com.bsiag.scout.rt.server.jdbc.mysql5117.source.feature.group"
    version="0.0.0"/>
<repository location="http://tools.bsiag.com/marketplace/jdbc/4.0"/>
</location>
```

List. A.1: BahBahChat - Target-File Anpassung

A.4 Amazon Beanstalk - Konfiguration von Oracle Java 7

```
files:
  "/home/ec2-user/install-oracle-jdk.sh":
    mode: "000755"
    owner: ec2-user
    group: ec2-user
    content: |
      #!/usr/bin/env bash
      wget -O /home/ec2-user/jdk-7u25-linux-x64.rpm --no-cookies
          --no-check-certificate --header 'Cookie:gpw_e24=http://www.oracle.
          com' 'http://download.oracle.com/otn-pub/java/jdk/7u25-b15/
          jdk-7u25-linux-x64.rpm'
      rpm -Uvh /home/ec2-user/jdk-7u25-linux-x64.rpm
      alternatives --install /usr/bin/java java /usr/java/default/bin/java 3
      alternatives --set java /usr/java/default/bin/java

commands:
  execute-install-oracle-jdk-script:
    command: ./install-oracle-jdk.sh
    cwd: /home/ec2-user
```

List. A.2: AWS Beanstalk Java 7 Konfiguration

A.5 Amazon Beanstalk - Konfiguration von OpenJDK 7

```
packages:
  yum:
    java-1.7.0-openjdk: []
    java-1.7.0-openjdk-devel: []
commands:
  use_java7:
    command: alternatives --set java /usr/lib/jvm/jre-1.7.0-openjdk.x86_64/
        bin/java
```

List. A.3: AWS Beanstalk OpenJDK 7 Konfiguration

A.6 Messergebnisse - Blitz.io

	M1			M2		
timestamp	duration	hits	timeouts	duration	hits	timeouts
2,5	0,1757	3	0	0,0585	4	0
5,0	0,1399	16	0	0,056928	18	0
7,5	0,0833	39	0	0,054308	41	0
10,0	0,0842	71	0	0,058851	75	0
12,5	0,1080	115	0	0,061168	117	0
15,0	0,1982	122	0	0,156364	128	0
17,5	1,7307	144	0	2,003253	149	0
20,0	2,0316	152	0	2,270091	171	0
22,5	0,7697	155	0	2,14792	173	0
25,0	3,6563	166	5	2,104053	195	5
27,5	0,7107	169	15	2,13778	208	11
30,0	5,0570	171	24	5,073875	211	11
32,5	0,0445	173	26	1,549923	213	18
35,0	0,3788	176	42	2,873286	227	32
37,5	0,0400	177	48	3,804097	237	32
40,0	1,6550	179	67	0,036875	239	52
42,5	4,5208	181	77	2,584308	252	54
45,0	3,2379	182	85	1,752778	255	55
47,5	0,0449	184	103	2,045902	257	92
50,0	2,3882	187	124	2,055333	260	101
52,5	1,9010	188	142	0,082	262	111
55,0	2,3424	190	153	2,102	265	127
57,5	7,8028	191	174	2,066789	266	164
60,0	7,5879	194	192	6,266925	269	175

Tab. A.2: Add caption

		M1			M2	
timestamp	duration	hits	timeouts	duration	hits	timeouts
2,5	0,09775	4	0	0,238	4	0
5,0	0,063462	17	0	0,139917	16	0
7,5	0,057134	40	0	0,107273	38	0
10,0	0,055337	75	0	0,134512	71	0
12,5	0,059446	118	0	0,085359	110	0
15,0	0,057516	172	0	0,076006	166	0
17,5	0,059077	235	0	0,065469	226	0
20,0	0,055822	308	0	0,067227	296	0
22,5	0,057639	391	0	0,068419	382	0
25,0	0,054154	482	0	0,066321	472	0
27,5	0,216943	560	0	0,064948	572	0
30,0	0,081005	690	0	0,064318	688	0
32,5	0,065127	805	0	0,06465	808	0
35,0	0,061853	927	0	0,061468	936	0
37,5	0,063588	1082	0	0,066697	1081	0
40,0	0,059985	1227	0	0,067533	1227	0
42,5	0,251315	1367	0	0,064805	1387	0
45,0	0,167129	1514	0	0,089755	1558	0
47,5	0,090995	1687	0	0,069084	1731	0
50,0	0,070081	1885	0	0,070038	1922	0
52,5	0,072919	2077	0	0,069229	2127	0
55,0	0,277581	2225	0	0,136738	2274	0
57,5	0,423588	2383	0	1,81643	2336	0
60,0	0,509721	2405	0	1,160026	2442	0

Tab. A.3: Add caption

A.7 Messergebnisse - LoadUI

	Szenario L1					Szenario L2					Szenario L3					Szenario L4				
#	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5
1	80	119	55	57	54	81	65	87	55	90	54	52	50	51	57	77	65	56	85	53
2	44	82	52	43	38	48	60	65	44	59	66	55	68	54	58	69	59	79	71	46
3	85	74	52	51	36	67	65	76	36	43	75	86	64	75	53	67	56	84	70	58
4	46	63	42	36	39	88	63	75	41	37	156	48	84	59	58	95	59	59	62	60
5	55	50	43	35	38	61	52	48	47	61	61	51	53	58	49	77	58	61	53	82
6	39	48	52	60	41	40	50	42	44	52	155	82	54	66	56	57	59	55	48	61
7	38	48	53	36	41	44	36	42	37	55	67	51	53	51	82	79	43	92	119	57
8	48	67	46	40	39	34	36	36	41	41	62	81	56	58	66	79	72	83	68	51
9	41	51	41	40	38	40	37	38	46	61	57	55	95	62	44	133	67	48	97	71
10	44	54	46	38	51	44	45	49	41	50	53	65	72	56	56	49	69	57	55	53
11	59	63	45	48	62	41	39	39	61	42	70	50	80	70	55	55	59	67	63	61
12	48	82	41	45	46	59	36	40	48	40	58	58	66	71	51	76	49	62	63	92
13	40	223	35	50	59	71	38	49	43	42	64	56	54	82	56	86	66	50	68	81
14	49	291	49	47	40	59	32	40	42	50	82	84	90	98	49	65	82	57	54	77
15	44	157	52	45	40	88	41	55	46	43	63	37	98	65	64	71	50	57	75	80
16	46	120	54	34	37	87	36	41	48	44	77	51	80	75	68	72	76	78	83	82
17	48	85	51	39	40	79	42	43	38	57	82	82	99	95	63	63	56	120	97	89
18	49	66	41	41	46	65	36	51	36	36	66	45	87	68	66	155	73	107	93	88
19	55	53	55	37	38	59	46	47	43	48	58	32	75	61	66	138	93	75	101	81
20	55	58	58	49	40	32	26	25	29	31	53	50	100	82	80	158	97	79	94	85
21	37	54	39	98	45	122	21	27	32	39	47	52	72	66	64	220	87	141	124	81
22	36	34	41	158	41	48	23	34	28	33	53	44	75	80	93	182	88	113	93	79
23	38	52	40	294	37	39	22	24	26	30	44	46	71	57	64	181	84	127	121	84
24	43	64	62	290	37	49	24	30	30	36	52	42	88	69	72	159	105	106	99	96
25	37	55	34	225	51	48	30	31	27	27	44	53	91	80	86	152	112	95	82	76
26	38	65	37	177	49	1039	412	1044	1297	1060	35	65	90	60	48	208	98	96	93	76
27	49	54	44	126	42	678	1318	1067	1273	1072	44	47	75	103	67	247	100	175	88	75
28	43	53	49	83	41	1047	2233	1054	1279	722	62	35	92	52	53	138	108	146	108	84
29	39	68	45	42	39	673	2187	1066	1155	1090	44	49	61	79	42	147	110	87	153	102
30	59	71	43	45	41	641	2291	821	1247	740	43	42	75	76	47	90	102	87	112	81
31	76	169	35	44	39	1055	2344	1079	1047	730	48	40	62	60	46	178	129	138	89	80
32	39	144	46	35	51	637	2218	780	1305	1104	45	24	74	76	54	189	97	129	113	72
33	40	77	47	36	37	1069	2118	863	1157	1091	49	41	57	106	57	102	100	102	107	82
34	43	57	42	38	46	622	2069	1069	1068	742	41	25	50	67	46	232	98	94	146	67
35	43	54	45	36	38	601	2171	841	1031	1088	45	31	100	95	51	133	98	127	121	114
36	47	66	43	36	46	1060	3057	1094	876	738	47	46	91	55	46	124	124	125	110	65
37	41	58	51	38	39	597	2048	794	1035	682	43	76	57	83	40	96	99	77	112	55
38	35	65	50	38	55	546	2005	1077	988	731	37	89	86	49	36	92	88	168	79	60
39	45	96	55	37	41	1069	1959	726	953	1123	47	108	86	50	49	62	146	106	122	104
40	47	138	44	44	42	1501	2113	781	890	677	48	36	79	55	51	105	124	83	148	52
41	51	151	62	48	48	1459	1954	758	590	1141	43	65	68	64	81	122	79	68	121	64
42	41	223	42	47	43	1443	3106	708	866	690	47	54	43	46	51	109	73	169	142	66
43	38	155	42	39	43	1193	1756	659	1317	606	50	41	80	69	49	104	58	125	157	62
44	44	289	53	41	40	1496	3096	509	853	682	42	41	57	47	51	75	103	94	139	59
45	39	357	57	37	47	1434	1947	583	809	586	46	31	65	43	46	66	69	58	101	73
46	40	128	50	40	42	1284	1906	1585	343	932	48	48	55	40	55	86	160	159	122	61
47	39	67	44	53	51	1227	1856	1116	1668	1477	48	38	86	50	56	97	189	131	62	75
48	36	59	39	378	45	1179	3407	1508	1723	1071	46	64	52	46	47	54	68	85	142	59
49	45	60	41	253	46	1283	3047	1408	1575	1474	42	38	43	45	43	73	97	105	105	74
50	52	64	40	352	49	1138	3676	1457	1325	1441	54	29	49	62	53	60	70	64	70	57

Tab. A.4: Ergebnisse LoadUI - Requests 1 - 50

	Szenario L1					Szenario L2					Szenario L3					Szenario L4				
#	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5
51	51	61	52	254	39	1530	5112	1516	1954	1546	45	38	58	42	45	70	121	112	61	70
52	47	62	35	188	39	1262	3194	1180	3415	1415	51	37	79	46	56	60	65	80	79	57
53	52	82	39	322	43	1173	5854	1385	3212	1120	45	34	68	49	46	100	72	115	123	69
54	49	74	50	200	41	1038	8734	1165	3513	1399	39	31	59	45	41	271	69	68	116	59
55	45	51	37	160	44	1648	8661	1161	5024	1108	45	43	58	62	39	91	74	98	70	62
56	42	62	41	94	46	1176	8613	826	5373	1097	65	49	59	54	58	53	64	58	86	76
57	41	82	67	50	45	978	8667	1478	5009	951	50	49	53	49	45	70	67	92	67	67
58	38	74	67	47	36	1127	5533	1180	5084	1003	41	36	49	51	49	62	49	72	60	110
59	36	61	52	44	40	950	8586	1433	8543	1052	40	38	94	59	85	106	60	122	74	91
60	45	67	54	40	42	562	5439	888	9342	729	42	40	82	53	64	64	41	115	57	61
61	54	57	69	46	45	1626	5493	994	9391	1688	37	55	71	41	51	124	73	73	76	136
62	55	80	52	44	43	1610	5394	1054	9296	2425	43	45	64	48	63	74	78	67	57	77
63	47	54	59	38	46	2312	8626	1107	9166	3804	46	26	58	63	48	60	50	84	39	71
64	56	52	66	43	44	3089	5361	858	9118	3138	41	43	58	61	62	125	82	128	56	61
65	54	58	59	48	48	4349	5216	1210	9262	2520	39	42	56	47	51	102	87	62	49	63
66	51	47	66	42	44	5005	5319	1166	9363	7900	47	47	58	60	55	94	78	90	66	54
67	53	70	62	50	43	7536	5270	1722	9061	7856	47	43	46	54	44	66	64	57	52	68
68	52	55	67	42	50	8061	5422	1549	8930	8907	47	28	42	49	62	105	61	71	47	48
69	37	60	77	42	37	8809	4973	2588	8837	9040	51	58	63	52	41	51	60	65	50	59
70	47	64	59	42	35	8764	9031	2816	8888	9512	47	46	42	47	38	66	53	99	62	53
71	35	49	68	45	40	8839	9690	4519	8993	9433	51	45	51	46	46	60	51	94	68	63
72	42	61	60	46	46	8449	9594	4839	8818	9484	45	28	43	46	44	103	49	73	50	48
73	35	52	71	44	42	8791	9645	4203	9170	9391	43	49	65	50	51	58	49	76	66	54
74	59	56	63	45	40	8758	9584	7764	9071	9591	51	34	52	49	43	131	51	59	45	44
75	58	57	55	41	36	8729	9391	8653	9491	9343	51	37	59	55	39	47	52	89	44	58
76	58	65	51	51	41	8636	9503	8475	9646	9643	45	58	52	63	53	111	47	60	78	58
77	59	48	59	45	36	8626	9655	8463	9699	9094	43	48	64	67	42	91	48	49	75	48
78	45	71	46	37	44	8278	9606	8424	9416	9496	42	59	61	55	51	72	41	94	63	48
79	59	47	33	43	43	8596	9456	8382	9527	9444	47	46	49	51	46	109	62	50	42	36
80	58	56	38	55	48	8415	9407	8484	9379	9546	45	27	51	46	46	87	74	38	57	48
81	57	51	38	49	33	8365	9707	8286	9186	9152	46	71	59	45	38	79	51	59	87	84
82	45	57	39	40	39	8266	9246	8236	9295	9089	41	54	80	57	46	91	42	103	72	76
83	39	57	37	40	35	9006	9310	8339	9184	9238	39	41	44	57	42	73	63	28	60	67
84	41	62	48	38	37	8856	9360	8128	8594	9290	43	23	44	45	40	90	52	102	55	50
85	46	78	39	50	44	9058	9160	8178	8545	9341	51	40	82	43	38	60	51	29	76	67
86	63	55	61	49	35	9158	9198	8230	8974	9743	45	41	54	58	42	72	71	90	51	77
87	42	72	43	38	40	8953	9940	8101	9372	9709	54	41	57	68	52	93	48	35	26	72
88	51	88	44	44	35	8809	9892	8049	9139	9759	48	53	72	56	50	57	41	80	73	54
89	37	66	38	56	53	9009	9992	8816	9491	9549	48	31	101	47	45	31	60	23	26	72
90	44	72	38	41	47	8165	9935	8737	9103	9504	52	50	68	45	46	61	52	78	54	55
91	41	60	39	48	42	8363	9786	8817	8553	9623	42	39	63	42	45	89	64	29	47	42
92	47	68	47	42	38	8960	9643	8618	9253	9742	44	49	68	52	45	60	52	28	86	61
93	37	56	44	47	36	8809	9983	8439	9214	8638	44	39	51	59	44	33	36	73	34	35
94	58	63	40	53	39	8760	9934	8791	9074	8689	45	37	65	50	54	44	49	61	30	47
95	46	37	47	36	39	8796	9784	8393	8975	9239	46	41	53	46	53	30	34	25	62	29
96	53	42	46	40	45	8649	9587	8648	9119	9739	45	42	62	44	47	29	53	31	38	25
97	47	40	46	45	126	8595	9838	8749	8848	9328	54	41	41	55	50	42	50	30	96	81
98	39	59	60	55	60	8754	9100	7820	8992	8630	54	23	82	63	51	55	33	26	29	43
99	48	50	63	44	38	8552	9561	7733	8951	9389	44	52	64	51	67	37	32	53	28	24
100	47	34	52	35	39	8706	9768	8539	9167	9493	46	48	52	62	47	81	28	101	82	26

Tab. A.5: Ergebnisse LoadUI - Requests 51 - 100

	Szenario L1					Szenario L2					Szenario L3					Szenario L4				
#	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5
101	42	45	45	36	44	8477	9721	8322	8621	8594	41	47	100	46	63	33	31	75	66	50
102	63	37	49	38	34	8429	9373	8682	9134	9652	47	27	57	61	47	42	28	30	34	25
103	68	34	67	35	39	8983	9425	8731	8586	9553	47	44	82	44	44	29	28	89	32	39
104	47	43	51	32	43	8481	9523	8283	8894	8864	63	43	59	57	69	70	30	78	28	64
105	49	69	52	40	40	8684	9325	8184	8880	8663	59	59	50	47	59	53	27	81	58	40
106	39	37	56	37	46	8660	9625	8484	8832	9620	58	33	64	63	56	35	39	76	30	33
107	43	39	62	38	46	9075	9477	8253	8370	9209	56	48	93	44	84	31	26	71	41	25
108	40	41	56	33	43	8475	9001	8408	9020	9102	80	45	74	51	44	26	30	127	36	22
109	42	45	77	37	56	8877	9142	8459	8823	8847	61	45	62	50	55	31	27	90	66	55
110	40	41	59	38	50	8978	9091	8432	8668	9197	53	42	54	49	58	76	30	96	32	26
111	49	46	66	38	52	9027	8796	8422	8526	8922	60	40	44	52	48	38	31	123	32	30
112	36	48	49	41	39	8779	8900	8184	8487	9014	54	34	51	49	75	29	34	60	44	32
113	38	51	53	36	38	8930	8624	8391	8843	8866	85	48	45	42	46	61	23	90	28	28
114	44	37	42	50	36	8380	8988	8304	8643	9065	46	39	57	43	46	34	34	49	26	31
115	42	35	39	42	39	8332	8589	8357	8597	9119	44	33	46	75	47	45	37	48	50	31
116	40	60	51	62	36	8632	9150	8300	15569	8752	47	40	50	79	45	82	28	132	29	52
117	45	45	43	53	41	8434	8349	8200	15520	8824	77	34	77	46	45	53	31	86	29	21
118	44	44	37	39	31	8686	8801	8108	15507	8685	51	48	44	75	52	79	26	50	37	33
119	40	44	36	36	36	8585	8752	8012	15466	8635	48	59	51	64	42	49	24	47	50	39
120	36	42	34	41	39	8754	8602	8179	16069	9235	47	50	81	59	51	55	25	59	30	27
121	65	43	46	35	41	8564	8513	8081	16219	8733	61	25	53	50	45	87	27	39	29	30
122	38	36	37	37	44	15089	8499	15698	16122	15924	42	53	86	49	50	56	29	57	43	53
123	47	43	34	41	38	15100	8449	15668	16172	15952	45	44	57	48	48	73	29	50	31	23
124	44	50	44	144	49	15840	8601	15771	15967	15905	72	32	95	47	43	38	28	58	28	38
125	42	48	43	123	36	15993	8900	15593	16075	15719	49	51	57	41	41	42	28	49	39	51
126	43	39	38	61	51	15819	8952	15643	16024	16180	47	71	46	43	62	63	1055	69	40	58
127	37	47	35	46	47	15792	15061	15596	16821	16132	52	41	52	39	49	35	1049	52	36	43
128	46	49	36	47	43	15770	15060	15497	16721	16032	51	24	82	66	41	58	1047	48	1051	1057
129	45	45	34	38	42	15719	15067	15449	16772	16092	50	37	54	57	56	34	1041	60	1043	1048
130	40	44	34	36	46	15680	15065	15552	16689	16777	70	44	53	55	46	40	1053	60	1046	1056
131	36	37	35	41	47	15519	16722	15420	16563	16878	46	26	57	47	42	52	1046	1050	1042	1064
132	45	41	38	38	50	15491	17010	15369	16516	16657	52	54	47	49	45	48	1052	1044	1044	1059
133	50	54	49	46	43	16185	16928	16081	16682	16865	71	44	60	46	49	59	1057	1040	1068	1038
134	47	39	46	41	49	16236	16879	15983	16633	16795	48	38	50	46	51	43	1051	1042	1056	1048
135	37	42	44	39	42	16131	16979	15974	16704	16995	43	26	53	50	47	26	1046	1040	1061	1040
136	42	39	39	47	38	16207	16835	16087	16805	16745	45	41	69	46	43	1053	1044	1042	1046	1074
137	55	49	42	44	45	16148	17036	15962	16859	16896	39	36	56	49	54	1055	1057	1037	1037	1067
138	43	63	36	37	46	16100	17076	15913	16763	16996	44	43	52	88	48	1038	1045	1047	1037	1046
139	44	56	45	35	46	16050	17202	15863	16307	16906	44	25	50	58	46	1061	1048	1085	1052	1046
140	36	48	37	36	44	15999	17291	15820	16408	17226	73	43	41	50	44	1043	1073	1044	1039	1044
141	40	50	40	42	50	16131	17058	15798	16361	17277	43	46	57	42	46	1048	1065	1055	1061	1062
142	47	43	149	39	38	16237	17112	15552	16469	17030	40	49	59	51	42	1040	1055	1121	1042	1056
143	49	49	98	37	40	16094	17164	16077	16524	17181	49	23	40	47	45	1047	1051	1042	1093	1049
144	50	41	50	53	41	16052	16969	16137	16636	17133	45	45	61	50	60	1047	1050	1063	1089	1083
145	42	36	46	48	39	16065	16927	15641	16056	17336	48	41	51	48	43	1041	1056	1074	1047	1067
146	41	40	43	40	37	15967	16884	16043	16008	16590	65	41	58	41	43	1053	1069	1061	1085	1068
147	46	41	53	42	41	15817	17041	15993	16215	16548	48	25	45	49	45	1055	1051	1059	1078	1060
148	38	44	52	39	35	16031	16842	15811	16165	17202	48	43	39	38	43	1040	1046	1057	1038	1049
149	40	63	47	49	46	15885	17125	15712	16316	17316	45	45	48	52	49	1046	1061	1085	1074	1066
150	51	42	40	46	39	15937	17103	15762	16268	16802	65	44	50	42	49	1061	1065	1056	1078	1058

Tab. A.6: Ergebnisse LoadUI - Requests 101 - 150

A.8 Servlet-Ausgabe der Nachrichten-Synchronisation

```
1 Node: c3200035-9c67-4127-b0a9-21819198b6da
2 sentMessagesCount: 50
3 receivedMessagesCount: 200
4 Node: 3d5d7db3-d190-42ce-b41d-fe1890a340f1 : 50
5 Node: 42fe0399-7d61-4d9e-b5f3-4048009ae6fd : 50
6 Node: c3200035-9c67-4127-b0a9-21819198b6da : 50
7 Node: e43decd3-37c0-41dc-a056-329317689f3b : 50
8 Node: 3bd5948d-0f56-4336-82d6-11f66c9a436b : 50
```

List. A.4: Servlet-Ausgabe der Nachrichten-Synchronisation

Literaturverzeichnis

[All07] OSGi Alliance. About the osgi service platform. URL: http://www.osgi.org/wiki/uploads/Links/OSGiTechnicalWhitePaper.pdf, June 2007. (Stand: 07.06.2007, Zugriff: 14.02.2014). (Zitiert auf Seite 7.)

[All13] OSGi Alliance. Osgi cloud ecoystems. URL: http://www.osgi.org/download/osgi-early-draft-2013-03.pdf, February 2013. (Stand: 28.02.2013, Zugriff: 24.02.2014). (Zitiert auf Seite 20.)

[Ama10] Configuring auto scaling using aws toolkit for eclipse. URL: http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_Java.managingappenv.as.html, January 2010. (Stand: 2014, Zugriff: 14.02.2014). (Zitiert auf Seite 32.)

[Appa] AppFog. Feature roadmap. URL: https://docs.appfog.com/roadmap. (Zugriff: 03.03.2014). (Zitiert auf Seite 33.)

[Appb] AppFog. Java. URL: https://docs.appfog.com/languages/java. (Zugriff: 26.03.2014). (Zitiert auf Seite 33.)

[Appc] AppFog. Simple and clear pricing. URL: https://www.appfog.com/pricing/. (Zugriff: 03.03.2014). (Zitiert auf Seite 33.)

[Bau11] Christian Baun. *Cloud Computing - Web-basierte dynamische IT-Services*. Springer, Berlin, Heidelberg, 2011. (Zitiert auf den Seiten 13, 14 und 16.)

[Bit] Bitkom. Umsatz mit cloud computing steigt auf fast 8 milliarden euro. URL: http://www.bitkom.org/de/markt_statistik/64086_75301.aspx. (Zugriff: 14.04.2014). (Zitiert auf Seite 2.)

[Car13] Lucas Carlson. *Programming for PaaS*. Ö'Reilly Media, Inc.", 2013. (Zitiert auf Seite 22.)

[Cla11] Guglielmo Claudio. Scout main architecture. URL: http://wiki.eclipse.org/File:Scout_main_architecture.png, November 2011. (Stand: 01.11.2011, Zugriff: 20.02.2014). (Zitiert auf Seite 6.)

[cloa] Rabbitmq as a service. URL: http://www.cloudamqp.com/. (Zugriff: 14.02.2014). (Zitiert auf Seite 32.)

[Clob] Clever Cloud. Clever cloud documentation. URL: http://doc.clever-cloud.com/. (Zugriff: 03.03.2014). (Zitiert auf Seite 34.)

[Cloc] Clever Cloud. Deploy java maven projects. URL: http://doc.clever-cloud.com/java/java-maven/. (Zugriff: 26.03.2014). (Zitiert auf Seite 34.)

[Clod] Clever Cloud. Deploy war/ear projects. URL: http://doc.clever-cloud.com/java/java-war/. (Zugriff: 03.03.2014). (Zitiert auf Seite 34.)

[Cloe] Clever Cloud. Scaling management. URL: http://doc.clever-cloud.com/clever-cloud-overview/scaling/. (Zugriff: 03.03.2014). (Zitiert auf Seite 34.)

[CMKS09] Trieu C Chieu, Ajay Mohindra, Alexei A Karve, and Alla Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *e-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*, pages 281–286. IEEE, 2009. (Zitiert auf Seite 19.)

[Cra09] Wickesser Craig. Best practices for writing scalable applications. URL: http://www.infoq.com/news/2009/08/google-chose-jetty, August 2009. (Stand: 05.09.2009, Zugriff: 07.03.2014). (Zitiert auf Seite 35.)

[Cyr13] Le Clerc Cyrille. Clickstack - application containers. URL: https://developer.cloudbees.com/bin/view/RUN/ClickStack, December 2013. (Stand:12.12.2013, Zugriff: 14.02.2014). (Zitiert auf Seite 34.)

[Cyr14a] Le Clerc Cyrille. Application session stores. URL: https://developer.cloudbees.com/bin/view/RUN/AppSessionStores, January 2014. (Stand:27.01.2014, Zugriff: 14.02.2014). (Zitiert auf Seite 35.)

[Cyr14b] Le Clerc Cyrille. Database guide. URL: https://developer.cloudbees.com/bin/view/RUN/DatabaseGuide, January 2014. (Stand:22.01.2014, Zugriff: 14.02.2014). (Zitiert auf Seite 35.)

[Dav10] Bosschaert David. Osgi cloud ecoystems. URL: http://www.osgi.org/wiki/uploads/CommunityEvent2012/OSGI%20and%20Cloud%20Computing-%20David%20Bosschaert.pdf, October 2010. (Stand: 01.10.2010, Zugriff: 24.02.2014). (Zitiert auf Seite 20.)

[DWC10] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. Ieee, 2010. (Zitiert auf den Seiten 16 und 17.)

[Ecl] Eclipse. Scout/release/luna. URL: http://wiki.eclipse.org/Scout/Release/Luna#JRE. (Zugriff: 06.03.2014). (Zitiert auf Seite 28.)

[FE10] Borko Furht and Armando Escalante. *Handbook of Cloud Computing* -. Springer, Berlin, Heidelberg, 2010. aufl. edition, 2010. (Zitiert auf Seite 13.)

[Foua] Cloud Foundry. Buildpacks. URL: http://docs.run.pivotal.io/buildpacks/. (Zugriff: 03.03.2014). (Zitiert auf Seite 37.)

[Foub] Cloud Foundry. Scaling cloud foundry. URL: http://docs.run.pivotal.io/concepts/high-availability.html. (Zugriff: 03.03.2014). (Zitiert auf Seite 37.)

[Fouc] Cloud Foundry. Tips for java developers. URL: http://docs.gopivotal.com/pivotalcf/buildpacks/java/java-tips.html. (Zugriff: 25.04.2014). (Zitiert auf Seite 37.)

[FR11] Christoph Fehling and Ralph Retter. Cloud computing patterns, 2011. (Zitiert auf Seite 23.)

[Fra] Fraunhofer. Was bedeutet public, private und hybrid cloud? URL: http://www.cloud.fraunhofer.de/de/faq/publicprivatehybrid.html. (Zugriff: 20.02.2014). (Zitiert auf Seite 17.)

[GHJV98] Erich Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns Cd: Elements of Reusable Object-Oriented Software*. Addison-Wesley professional

computing series. Addison-Wesley Publishing Company, 1998. (Zitiert auf Seite 81.)

[Gooa] Google. Best practices for writing scalable applications. URL: https://developers.google.com/appengine/articles/scaling/overview. (Stand: 12.03.2012, Zugriff: 07.03.2014). (Zitiert auf Seite 35.)

[Goob] Google. Java runtime environment. URL: https://developers.google.com/appengine/docs/java/?hl=de. (Stand: 06.06.2014, Zugriff: 07.03.2014). (Zitiert auf Seite 35.)

[hera] Cloudamqp. URL: https://devcenter.heroku.com/articles/cloudamqp. (Stand: 27.11.2013, Zugriff: 20.02.2014). (Zitiert auf Seite 38.)

[herb] Dyno size. URL: https://devcenter.heroku.com/articles/dyno-size. (Stand: 14.02.2014, Zugriff: 20.02.2014). (Zitiert auf Seite 38.)

[herc] Heroku java support. URL: https://devcenter.heroku.com/articles/java-support. (Zugriff: 20.02.2014). (Zitiert auf Seite 38.)

[herd] Ironmq. URL: https://devcenter.heroku.com/articles/iron_mq. (Stand: 28.10.2013, Zugriff: 20.02.2014). (Zitiert auf Seite 38.)

[here] Languages. URL: https://devcenter.heroku.com/categories/language-support. (Zugriff: 20.02.2014). (Zitiert auf Seite 38.)

[herf] Memcached cloud. URL: https://devcenter.heroku.com/articles/memcachedcloud. (Stand: 18.02.2014, Zugriff: 20.02.2014). (Zitiert auf Seite 38.)

[herg] Redis cloud. URL: https://devcenter.heroku.com/articles/rediscloud. (Stand: 18.02.2014, Zugriff: 20.02.2014). (Zitiert auf Seite 38.)

[herh] Scaling your dyno formation. URL: https://devcenter.heroku.com/articles/scaling. (Stand: 04.02.2014, Zugriff: 20.02.2014). (Zitiert auf Seite 38.)

[her13a] Create a java web application using embedded tomcat. URL: https://devcenter.heroku.com/articles/create-a-java-web-application-using-embedded-tomcat, November 2013. (Stand: 28.11.2013, Zugriff: 20.02.2014). (Zitiert auf Seite 38.)

[her13b] Deploy a java web application that launches with jetty runner. URL: https://devcenter.heroku.com/articles/deploy-a-java-web-application-that-launches-with-jetty-runner, October 2013. (Stand: 28.10.2013, Zugriff: 20.02.2014). (Zitiert auf Seite 38.)

[Inf09] BITKOM Bundesverband Informationswirtschaft. Cloud computing– evolution in der technik, revolution im business. URL: https://www.bitkom.org/files/documents/BITKOM-Leitfaden-CloudComputing_Web.pdf, 2009. (Zitiert auf Seite 16.)

[jela] Java application server configuration. URL: http://docs.jelastic.com/de/java-application-server-config. (Zugriff: 20.02.2014). (Zitiert auf Seite 35.)

[jelb] Lastenausgleich. URL: http://docs.jelastic.com/de/http-load-balancing. (Zugriff: 17.02.2014). (Zitiert auf Seite 35.)

[jelc] Session replication via memcached. URL: http://jelastic.com/de/docs/replication-memcached. (Zugriff: 17.02.2014). (Zitiert auf Seite 35.)

[jeld] White-paper. URL: http://blog.jelastic.com/wp-content/uploads/2013/10/White-Paper.pdf. (Zugriff: 17.02.2014). (Zitiert auf Seite 35.)

[Kep12] Ben Kepes. Understanding the cloud computing stack saas, paas, iaas. *CLOUD U: Understanding the Cloud Computing Stack SaaS, PaaS and IaaS,[Online]*, 3:17, 2012. (Zitiert auf Seite 16.)

[Kha] Sapenov Khazret. Cloud platforms (paas). URL: http://docs.google.com/spreadsheet/ccc?key=0AiIXCd1D_TmGdFluZEJQakV5M0QwWXNWaXREcWR0Q0E. (Stand: 31.10.2013, Zugriff: 14.02.2014). (Zitiert auf Seite 31.)

[Mah13] Zaigham Mahmood. *Cloud Computing - Methods and Practical Approaches*. Springer, Berlin, Heidelberg, 2013. aufl. edition, 2013. (Zitiert auf den Seiten 16 und 17.)

[Mar13] Sprava Marina. Jms application deployment tutorial for beginners. URL: http://blog.jelastic.com/2013/08/08/jms-app

lication-deployment-tutorial-for-beginners/, August 2013. (Stand:08.08.2013, Zugriff: 20.02.2014). (Zitiert auf Seite 36.)

[McK] Joe McKendrick. More than one-third of it budgets now spent on cloud: Survey. URL: http://www.forbes.com/sites/joemckendrick/2012/04/11/more-than-one-third-of-it-budgets-now-spent-on-cloud-survey/. (Zugriff: 14.04.2014). (Zitiert auf Seite 2.)

[Men02] Daniel Menascé. Load testing of web sites. *Internet Computing, IEEE*, 6(4):70–74, 2002. (Zitiert auf Seite 68.)

[MG11] Peter Mell and Timothy Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011. (Zitiert auf den Seiten 13, 16, 17 und 28.)

[Mica] Microsoft. How to run a java application server on a virtual machine. URL: http://www.windowsazure.com/en-us/documentation/articles/virtual-machines-java-run-tomcat-application-server/. (Zugriff: 07.03.2014). (Zitiert auf Seite 36.)

[Micb] Microsoft. How to scale an application. URL: http://www.windowsazure.com/en-us/documentation/articles/cloud-services-how-to-scale/. (Zugriff: 07.03.2014). (Zitiert auf Seite 36.)

[Micc] Microsoft. Messaging. URL: http://www.windowsazure.com/en-us/services/messaging/. (Zugriff: 07.03.2014). (Zitiert auf Seite 36.)

[Micd] Microsoft. Windows azure-cache. URL: http://www.windowsazure.com/de-de/services/cache/. (Zugriff: 07.03.2014). (Zitiert auf Seite 36.)

[Mic13] Neale Michael. Database guide. URL: https://developer.cloudbees.com/bin/view/RUN/Autoscaling+explained, October 2013. (Stand:15.10.2013, Zugriff: 07.03.2014). (Zitiert auf Seite 34.)

[MRS11] Philippe Merle, Romain Rouvoy, and Lionel Seinturier. A reflective platform for highly adaptive multi-cloud systems. In *Adaptive and Reflective Middleware on Proceedings of the International Workshop*, pages 14–21. ACM, 2011. (Zitiert auf Seite 37.)

[Nea13] Michael Neale. Jvm runtime container. URL: https://developer.cloudbees.com/bin/view/RUN/Java+Container, November 2013. (Stand:11.11.2013, Zugriff: 14.02.2014). (Zitiert auf Seite 34.)

[Opea] Java application hosting. URL: https://www.openshift.com/developers/java. (Zugriff: 20.02.2014). (Zitiert auf Seite 37.)

[Opeb] Scale your applications on the web. URL: https://www.openshift.com/developers/scaling. (Zugriff: 20.02.2014). (Zitiert auf Seite 37.)

[Opec] Technology cartridges. URL: https://www.openshift.com/developers/technologies. (Zugriff: 20.02.2014). (Zitiert auf Seite 37.)

[oraa] Oracle weblogic server in the cloud. URL: https://cloud.oracle.com/java. (Zugriff: 20.02.2014). (Zitiert auf Seite 36.)

[Orab] Oracle. Interface httpsession. URL: http://docs.oracle.com/javaee/5/api/javax/servlet/http/HttpSession.html. (Zugriff: 20.02.2014). (Zitiert auf Seite 9.)

[Paa] Platform as a service - comparison list. URL: http://www.paaslist.com/. (Zugriff: 14.02.2014). (Zitiert auf Seite 31.)

[Pra] Luis Praxmarer. Die zehn wichtigsten it-trends. URL: http://www.computerwoche.de/a/2551615. (Zugriff: 14.04.2014). (Zitiert auf Seite 2.)

[Pre11] Nikolaos P Preve. *Grid computing: towards a global interconnected infrastructure*. Springer, 2011. (Zitiert auf Seite 13.)

[TC12] Jennings Terri and Murray Chuck. Using oracle java cloud service. URL: http://docs.oracle.com/cloud/latest/javacs_common/CSJSU.pdf, October 2012. (Stand: 01.02.2014, Zugriff: 14.02.2014). (Zitiert auf Seite 36.)

[TSB10] Wei-Tek Tsai, Xin Sun, and Janaka Balasooriya. Service-oriented cloud computing architecture. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 684–689. IEEE, 2010. (Zitiert auf Seite 16.)

[TTC+03] George Teodoro, Tulio Tavares, Bruno Coutinho, W Meira, and D Guedes. Load balancing on stateful clustered web servers. In *Computer Architecture*

and High Performance Computing, 2003. Proceedings. 15th Symposium on, pages 4–5. IEEE, 2003. (Zitiert auf Seite 15.)

[Viv13] Pandey Vivek. Cloudamqp. URL: `https://developer.cloudbees.com/bin/view/RUN/CloudAMQP`, July 2013. (Stand:06.07.2013, Zugriff: 14.02.2014). (Zitiert auf Seite 35.)

[VRMCL08] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008. (Zitiert auf Seite 16.)

[Woo11] Jeff Woods. *Five Options for Migrating Applications to the Cloud: Rehost, Refactor, Revise, Rebuild or Replace.* "Gartner", 2011. (Zitiert auf Seite 21.)