HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Cloud Deployment and Architectural Refactoring Lab

Studienarbeit

Technischer Bericht

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2013

Autoren: Marcel Tinner, Daniel Zigerlig

Betreuer: Prof. Dr. Olaf Zimmermann

20. Dezember 2013

Abstract

Cloud Computing hat sich in den letzten Jahren vom Trendthema zur wichtigen Architekturalternative für Entwicklung und Betrieb von Web-Anwendungen und anderer Software entwickelt.
Daher war es nur eine Frage der Zeit, bis dieses Thema auch im Informatikstudium an der HSR behandelt wird. Ab dem Frühlingssemester 2014 wird deshalb das Modul „Cloud Computing“ angeboten. Für die Vorbereitung dieses Moduls, insbesondere der Übungslektionen, werden Beispielanwendungen benötigt, welche die gelehrten Cloud-Konzepte (wie z.B. Cloud Computing Patterns) veranschaulichen und die technische Umsetzung der Konzepte und Patterns bei Cloud-Providern demonstrieren.
Im Rahmen dieser Studienarbeit wurden die drei PaaS-Anbieter Heroku, CloudBees und Google App Engine, die für den Übungsbetrieb besonders geeignet sind, detailliert analysiert. Um die Cloud-Konzepte sowie die spezifischen Eigenschaften der ausgewählten Cloud-Provider zu testen und zu veranschaulichen, wurden mehrere Beispielapplikationen entwickelt.
Die vorliegende Arbeit stellt einfache Tests mit den Anbietern sowie sieben neu erstellte Applikationen vor, welche die verschiedenen Cloud-Eigenschaften, wie das Map Reduce Pattern, das Verhalten von Sockets, den Unterschied von MySQL zu NoSQL-Datenbanksystemen aufzeigen sowie eine Kostengegenrechnung ermöglichen.
Die Analyse der Cloud-Umgebungen orientierte sich an einem in Rahmen der Arbeit erstellten Kriterienkatalog. Dieser Kriterienkatalog entwickelte sich während der Arbeit iterativ und inkrementell aus Vorgaben des Betreuers, einer Literaturrecherche sowie Erkenntnissen und Erfahrungen, die während Entwicklung und Test der Beispielanwendungen gewonnen wurden. Die vierzehn Kriterien im Katalog beschreiben wichtige Eigenschaften der PaaS-Anbieter und zeigen Vor- sowie Nachteile. Der Katalog kann als Entscheidungshilfe bei der Wahl eines geeigneten Cloud-Anbieters dienen.

Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selbst und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Regeln zitiert haben,
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum:

Rapperswil 19.12.2013

Name, Unterschrift:

Marcel Tinner

Daniel Zigerlig

Danksagungen

Wir danken folgenden Personen für Ihre Unterstützung während der Studienarbeit:

- Prof. Dr. Olaf Zimmermann für die Betreuung unserer Studienarbeit
- Danilo Bargen für viele interessante und hilfreiche Inputs während der gesamten Arbeit
- Unseren Partnerinnen für Geduld, Rat und motivierende Worte.
- Manuela Kaufmann für die grammatikalische Korrektur

Inhaltsverzeichnis

1 Management Summary

1.1 Ausgangslage

Cloud Computing hat sich in den letzten Jahren vom Trendthema zur wichtigen Architekturalternative für Entwicklung und Betrieb von Web-Anwendungen und anderer Software entwickelt. Neben traditionellen IT-Anbietern wie HP, IBM und Microsoft sind Internetfirmen wie Amazon und Google im Public Cloud Markt aktiv; verschiedene Open Source Assets erlauben den Aufbau von Private Clouds. Die Cloud-Anbieter unterscheiden sich stark hinsichtlich ihrer Preismodelle und der zugesicherten Dienstgüte. Anwendungsarchitekten und Entwickler sind daher mit einer Vielzahl neuer Designoptionen konfrontiert, z.B. nichtrelationale Speichertechniken (NoSQL), Message-Oriented Middleware mit At-Least-Once Delivery sowie Server-, Speicher- und Netzwerkvirtualisierung. Nicht alle Entwurfsmuster eignen sich für Cloud-Anwendungen; mit den Cloud-Ressourcen muss sparsam und fehlertolerant umgegangen werden.

Erstmals ab dem Sommersemester 2014 wird die HSR deshalb ein Aufbaumodul Cloud Computing anbieten. Für die Durchführung bestimmter Übungslektionen in diesem Modul werden noch Beispielanwendungen benötigt, welche die Unterschiede zu nicht-cloudbasierten Systemen aufzeigen.

1.2 Zielsetzung

Im Rahmen dieser Studienarbeit entwickelten wir mehrere Beispielanwendungen. Diese Anwendungen zeigen cloudspezifische Eigenschaften auf und sollen in den Übungsstunden zum Einsatz kommen. Entwickelt werden die Applikationen mit Java. Aufgrund des funktionalen Reichtums und der schnellen Verfügbarkeit werden Public Paas-Provider verwendet. Das Deployment erfolgt auf den Providern CloudBees, Heroku sowie Google App Engine.
Da sich Anwendungen im Betrieb in der Cloud-Umgebung teilweise anders verhalten als in traditionellen IT-Infrastrukturen, sammeln und analysieren wir Informationen zum Verhalten. Typische Eigenschaften und Erklärungen sind im Tutorium zusammengefasst.
Des weiteren erstellen wir einen Kriterienkatalog mit den gesammelten Informationen. Der Kriterienkatalog hat die Aufgabe, die Auswahl von Cloud-Providern anhand von technischen und organisatorischen Eigenschaften zu unterstützen. Er dient auch als Checkliste bezüglich der kritischen Erfolgsfaktoren und technischen Risiken von Cloud-Nutzungsprojekten sowie des Umstellungsbedarfes für Anwendungen, die aktuell noch in traditionellen Hosting-Infrastrukturen betrieben werden. Der Katalog dient damit als Entscheidungshilfe und Leitfaden für zukünftige Anwendungsarchitekten, welche im PaaS-Bereich entwickeln möchten.

1.3 Resultat

Das Endprodukt dieser Studienarbeit lässt sich in zwei Teile trennen, in Beispielanwendungen und ein Tutorium.

Es ist vorgesehen, dass die Beispielapplikationen in den Übungslektionen im Modul Cloud-Computing zum Einsatz kommen. Eine realisiert eine einfache und kompakte System-Selbstauskunft. Diese Selbstauskunfts-Applikation zeigt dem Architekten (oder: Cloud-Nutzer) nach dem Deployment sämtliche potentiell relevanten Informationen der Cloud-Plattform und der deployten Anwendung an (z.B. bei der Nutzung von Platform-as-a-Service).

Eine weitere Anwendung, welche im Rahmen der vorliegenden Arbeit entwickelt wurde, ist eine reine Java-Implementierung des bei einigen prominenten Cloud-Anbietern anzutreffende Map Reduce Pattern. Diese Implementierung hat das Ziel, die abstrakt-konzeptionellen Darstellung dieses Cloud Computing Patterns in der Literatur zu illustrieren und damit das Verständnis des Patterns zu vertiefen.

Die dritte Beispielanwendung demonstriert Interprozesskommunikartion mit TCP/IP Sockets. Dabei zeigt sie, dass sich der Einsatz von Sockets im Bereich der PaaS-Programmierung stark unterscheidet im Vergleich zur herkömmlichen IT-Infrastruktur.

Schliesslich entwickelten wir mehrere Anwendungsbeispiele, welche den Unterschied von relationalen zu nicht-relationalen Datenbanksystemen aufzeigen und die Auswirkungen dieser Unterschiede auf die Anwendungsprogrammierung demonstrieren.

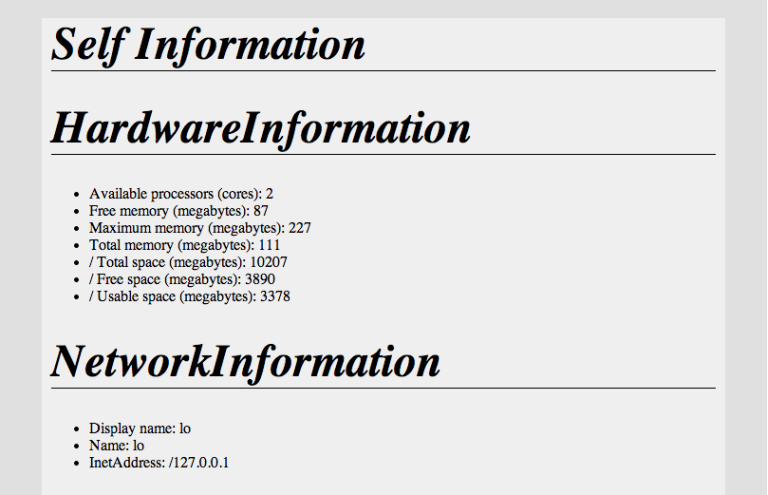


Abbildung 1 - Selbstauskunfts-Applikation

Das zweite Endprodukt der vorliegenden Arbeit ist ein umfassendes Tutorium. Dieses Tutorium beschreibt die Nutzung der PaaS-Provider und erklärt die Arbeit mit Schritt-für-Schritt Anleitungen. Spezifische Eigenschaften im Cloud-Bereich werden erläutert und teilweise durch praktische Analysen gezeigt.

Das Tutorium beinhaltet zudem den Kriterienkatalog. Dieser beschreibt in 14 Kriterien wichtige Eigenschaften von CloudBees, Heroku und Google App Engine. Diese Provider werden den Kriterien gegenübergestellt und verglichen.

1.4 Ausblick

Die vorliegende Studie in Kombination mit den Beispielapplikationen, dem Tutorium mit den Anleitungen sowie des Kriterienkatalogs, hilft dem Dozenten bei der Vorbereitung und Durchführung des Cloud Computing Moduls an der HSR.

Ausserdem sind die Vergleichskriterien für Cloud-Provider und die gesammelten Erfahrungen zu Umstellungsbedarf und technischen Erfolgskriterien und Risiken für Cloud-Projekte sicherlich auch für Cloud-Interessierte in der Industrie relevant. Wir hoffen deshalb auf viele interessierte Leser für Bericht und Tutorium.

2 Analyse der Anforderungen

2.1 Allgemeine Beschreibung

Diese Studienarbeit hat gemäss Aufgabenstellung im Kapitel „G Aufgabenstellung“ zum Ziel, Architekturkonzepte anhand mehreren Beispielapplikationen darzustellen und die Erkenntnisgewinne im Tutorium zu dokumentieren. Dieses Tutorium soll dem Anwendungsentwickler helfen, bestehende oder neue Applikationen für die Cloud-Umgebung anzupassen und stellt zudem hilfreiche Informationen zur Verfügung.

2.1.1 IaaS - Infrastructure as a Service

Ein IaaS-Anbieter stellt dem Kunden einen Zugang zu seinen Ressourcen wie beispielsweise Rechnern, Netzwerke sowie Speicher zur Verfügung. Dabei ist der Anwender für den Betrieb der Software selbst verantwortlich [Chr132].

2.1.2 PaaS - Platform as a Service

Ein PaaS-Anbieter stellt dem Kunden einen Zugang zu Programmier- oder Laufzeitumgebungen zur Verfügung. Somit kann ein Programmierer seine erstellten Programme auf eine PaaS-Platform laden und dort ausführen [Chr133].

2.1.3 SaaS - Software as a Service

Ein SaaS-Anbieter stellt dem Kunden einen Zugang zu einer Software bzw. einer Software-Sammlung zur Verfügung [Chr134]. Bekannte Beispiele von SaaS sind Dropbox oder Google Docs.

2.1.4 Public Cloud

Ermöglicht den Zugang zu Cloud-Infrastruktur über das Internet. Die Nutzung der Infrastruktur steht für die Öffentlichkeit zur Verfügung. Kunden bezahlen nur den Verbrauch ohne eigene Infrastrukturen anschaffen zu müssen.

2.1.5 Private Cloud

Ermöglicht den Zugang zu Cloud-Infrastruktur innerhalb der eigenen Organisation. Das heisst, die Cloud ist von der Organisation selbst zu verwalten.

2.1.6 OSSM

Nach der Definition OSSM betrachten wir den Begriff Cloud. Die Definition setzt sich folgendermassen zusammen.

- **O**n-demand: Der Server ist bereits vorhanden und bereit für das Deployment.
- **S**elf-service: Kunden können bestimmen, was sie wann wollen.
- **S**calable: Kunden bestimmen, wie viel sie wollen und können dies, falls notwendig, regulieren.
- **M**easureable: Es existiert eine transparente Abrechnung, welche sämtliche genutzten Ressourcen beinhalten.

2.2 Fokus

Der Fokus dieser Studienarbeit liegt darin, Anwendungsentwickler und Softwarearchitekten, welche eine neue oder bestehende Applikation in einer Cloud-Plattform überführen möchten, mit Tipps zu unterstützen. Dabei fokussieren wir uns auf Angebote im Public-PaaS-Bereich. Dieser ist frei zugänglich und adressiert viele Anwender. Wir konzentrierten uns dabei auf die drei Anbieter Heroku, CloudBees und Google App Engine. Unsere Erkenntnisse sind nach weiteren Recherchen teilweise auch auf andere Cloud-Anbieter abbildbar.

2.3 Anbieterevaluation

Es gibt mittlerweile unzählige Cloud-Anbieter im Internet. Für die Arbeit während der Studienarbeit wurden Public-PaaS-Anbieter gewählt, welche gegebenenfalls auch in den Übungsstunden des Cloud Computing Moduls eingesetzt werden können. Die gewählten Anbieter eignen sich besser für Anwendungsszenarien, in denen Kreditkartendaten nicht preisgegeben werden wollen. Ausserdem unterstützen sämtliche Anbieter die Programmiersprache Java, welche unter den Studenten der HSR durch Grundlagenmodule bekannt ist.

Folgende Anbieter wurden evaluiert:

- CloudBees [Clo132]
- Google App Engine [Goo13]
- Heroku [Her13]

3 Hauptergebnisse

3.1 Tutorium Überblick

Das erstellte Tutorium besteht aus folgenden Kapiteln:

- Kriterienkatalog
- Abrechnungsmodelle
- Cloudspezifische Eigenschaften
- Einfache Java SE Applikationen in der Cloud
- Diagnoseanwendung Selbstauskunft
- Nicht-relationale Datenbanksysteme
- Nutzung von TCP Sockets in der Cloud
- HTTP Zeitüberschreitung
- Map Reduce Pattern
- Rechenintensive Operationen in der Cloud
- Anleitungen

3.1.1 Kriterienkatalog

Es ist schwer einen objektiven Vergleich der Cloud-Anbieter zu erhalten. Deshalb zeigt dieses Kapitel einen Kriterienkatalog, welcher einen messbaren Vergleich der verschiedenen Cloud-Anbieter präsentiert. Diesen Kriterienkatalog stellten wir aus Literaturrecherchen sowie Erkenntnissen und Erfahrungen während der Entwicklung der Beispielanwendungen zusammen. Die folgenden Kriterien beschreiben wichtige Eigenschafen der PaaS-Provider und zeigen Vor- sowie Nachteile. Der Katalog dient als Entscheidungshilfe bei der Wahl eines geeigneten Cloud-Anbieters.

K1: Einstiegshilfen
K2: Dokumentation
K3: Sprachen und Frameworks
K4: Deployment in die Cloud
K5: Anpassungsfähigkeit der Serverinstanz
K6: Handhabung
K7: On-demand
K8: Self-Service
K9: Scalable
K10: Measurable
K11: Domain Verwaltung
K12: Nutzung von Sockets
K13: Selbstauskunft
K14: Datenbanksysteme

3.1.2 Abrechnungsmodelle

In der Cloud-Umgebung gibt es zwei verschiedene Abrechnungssysteme. Dieses Kapitel beschreibt die Systeme von Google App Engine, Heroku sowie CloudBees. Ausserdem zeigt eine Beispiel-Applikation, welche Kosten bei Abfragen auf die Datenbank bei Google App Engine entstehen.

3.1.3 Cloudspezifische Eigenschaften

Entscheidet sich ein Anwendungsentwickler seine Applikationen bei einem Cloud-Anbieter auszuführen, verändern sich gewohnte Ausführungs- und Managementeigenschaften. Dieses Kapitel des Tutoriums beschreibt spezifische Eigenschaften. Es zeigt auf, welche Überlegungen ein Entwickler durchführen muss und wie sich der Betrieb der Applikation verändert. Zu diesen Eigenschaften gehören beispielsweise der Lebenszyklus einer Applikation, sowie auch die Einrichtung eines eigenen Domainnamens.

3.1.4 Einfache Java SE Applikationen in der Cloud

Neben Java Webprojekten gibt es in der Cloud auch die Möglichkeit, einfache Java SE Applikationen auszuführen. Dies können beispielsweise Prozesse sein, welche einmalig ausgeführt werden, oder aber auch Hintergrund-Tasks, welche länger dauern. Dieses Kapitel beschreibt, wie man eine solche Applikation erstellt, auf was zu achten ist und zeigt zwei Beispiele an den PaaS-Providern Heroku und CloudBees.

3.1.5 Nicht-relationale Datenbanksysteme

Die in diesem Kapitel beschriebenen Beispielapplikationen zeigen dem Entwickler eine Übersicht zu den verschiedenen Datenbanken im Cloud Bereich. Dies veranschaulichen wir durch eine MySQL Applikation, deren Funktionalität anschliessend mittels MongoDB und Google Datastore nachgebaut wird. Dafür verwenden wir MySQL und MongoDB von CloudBees, sowie Datastore von Google. Als Ausgangssituation besteht eine MySQL Applikation. Diese beinhaltet eine Join-Operation, eine typische Operation, welche jedoch bei nicht-relationalen Datenbanksystemen nicht zur Verfügung steht. Mit diesen Applikationen wird veranschaulicht, was umgestellt und auf was verzichtet werden muss, wenn man mit einer nicht-relationalen Datenbank arbeitet.

3.1.6 Nutzung von TCP Sockets in der Cloud

Möchte ein Anwendungsentwickler oder Softwarearchitekt in seiner Anwendung mit Sockets arbeiten, unterscheidet sich dies wesentlich sobald er dies im Cloud-Umfeld ausführen möchte. Dieses Kapitel beschreibt mit Beispielapplikationen, wie man einerseits intern, aber auch extern mit Socket-Applikationen arbeiten kann. Sämtliche Beispiele beziehen sich auf die PaaS-Provider Heroku sowie CloudBees. Entwickelt wurde ein Server-Socket, welches bei einer Anfrage eine HTTP-Antwort sendet. Erklärt wird zudem, wie die Weiterleitung der Daten funktioniert.

3.1.7 HTTP Zeitüberschreitung

Eine Webanwendung in der Cloud, welche breit skaliert ist und nebenläufig arbeitet, ist nur wirksam, wenn die Antwortzeiten kurz sind. Die in dieser Arbeit behandelten PaaS-Anbieter intervenieren, sobald eine Applikation nach einer überschrittenen Zeitspanne nicht mehr antwortet. Dieses Kapitel beschreibt mit einer Beispielanwendung, wie die einzelnen Anbieter vorgehen.

3.1.8 Map Reduce Pattern

Map Reduce ist ein Verfahren, welches zur Verarbeitung von grossen Datenmengen eingesetzt wird [Map13] [Chr131]. Das Verfahren besteht aus zwei Phasen, der Aufteilung aller Daten in kleinere Stücke (Map) und das Zusammenfügen der errechneten Zwischenergebnisse (Reduce). Der Vorteil von Map Reduce ist, dass man das Verfahren sehr gut parallelisieren kann [LBl13]. Da sich Anwendungen in der Cloud einfach horizontal skalieren lassen, ist dieses Verfahren in diesem Zusammenhang gut umzusetzen. Somit sind auch nicht zahlreiche Rechenmaschinen im eigenen Serverraum notwendig.

3.1.9 Rechenintensive Operationen in der Cloud

In diesem Kapitel wird dem Entwickler gezeigt, dass sich nicht alle Methoden in der Cloud gleich verhalten wie lokal. Es gibt Methoden, die in der Cloud verhältnismässig sehr viel mehr Rechenzeit benötigen. Die erstellte Applikation führt mehrere Methoden aus und zeigt dabei die Ausführungszeit an. Das Resultat mit den im Rahmen dieser Studienarbeit getesteten Methoden zeigt, dass die Ausführung der SecureRandom-Methode nach dem Deployment auf einen PaaS-Anbieter mehr Zeit beansprucht als lokal.

3.1.10 Anleitungen

Der zehnte Abschnitt des Tutoriums beinhaltet Schritt-für-Schritt Anleitungen. Die Anleitungen beschreiben die Benutzung der gewählten PaaS-Anbieter. Dazu gehören beispielsweise das Erstellen von ersten Hello-World Programmen, das Deployment dieser zum Anbieter und das Nutzen der von den Providern zur Verfügung gestellten Tools und Plug-Ins.

3.2 Beispielapplikationen

Im Rahmen dieser Semesterarbeit entstanden mehrere Beispielanwendungen, welche in den folgenden Unterkapiteln beschrieben sind.

3.2.1 Diagnoseanwendung Selbstauskunft

Damit sich ein Entwickler eine Übersicht über die von ihm verwendete Infrastruktur bei einem Cloud-Anbieter verschaffen kann, haben wir eine Selbstauskunftsapplikation erstellt. Die Grundidee war, dass sich die Applikation sämtliche für den Entwickler potentiell relevante Informationen vom aktuell deployten Server bezieht und diese dann via Servlet direkt im Browser anzeigt.

Abbildung 2 - SelfInformation Applikation Grundidee

Eine grosse Menge an Informationen können mit Hilfe von Java erfragt werden, gruppiert wurden die Informationen in folgende Teilbereiche:

- Hardware-Informationen
- Netzwerk-Informationen
- Systemeigenschaften
- Umgebungsvariablen

3.2.1.1 Problem JVM

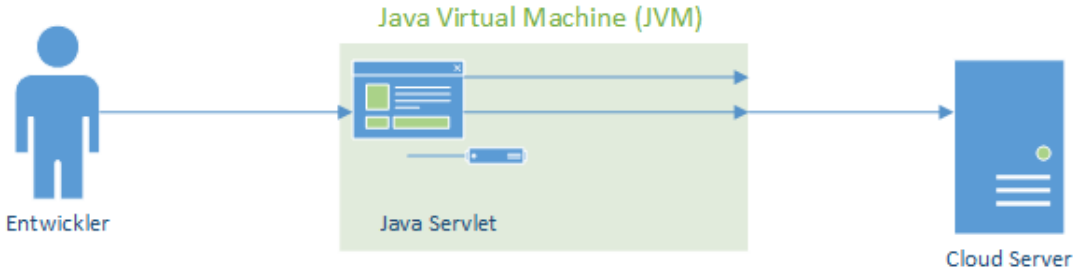


Abbildung 3 - SelfInformation Problem JVM

Durch die Nutzung von Java können jedoch nur Informationen der JVM ausgelesen werden. Die effektive Grösse des Arbeitsspeichers bleibt dem Java-Programmierer verborgen. Auszulesen ist nur die Grösse des Speichers, welcher der JVM zur Verfügung gestellt wird.

3.2.1.2 Endresultat

Die Applikation wurde als Dynamic Web Project erstellt. Im Presentation Layer befindet sich ein Servlet. Der Business Logic Layer besteht aus POJO's, somit lässt sich die Applikation unkompliziert in andere Umgebungen integrieren.

Nach dem Deployment auf einen PaaS-Provider erscheint folgende Übersicht.
Die Ausgabe ist je nach Provider unterschiedlich. Bei Google App Engine können Informationen wie *NetworkInformation* und *EnvironmentVariables* nicht ausgelesen werden.

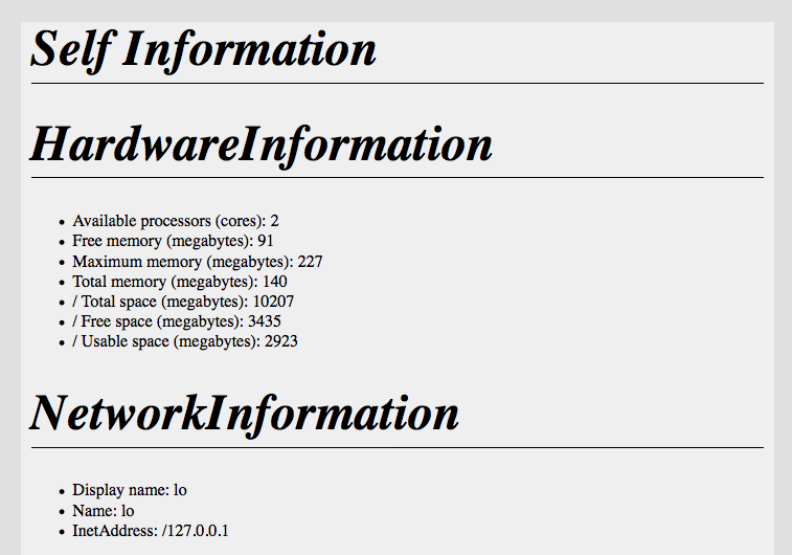


Abbildung 4 - SelfInformation Endresultat

3.2.2 Nicht-relationale Datenbanksysteme

Folgende Beispielapplikationen zeigen dem Entwickler eine Übersicht zu den verschiedenen Datenbanken im Cloud Bereich. Dies veranschaulichen wir durch eine MySQL Applikation, deren Funktionalität anschliessend mittels MongoDB und Google Datastore nachgebaut wurde. Dafür verwenden wir MySQL und MongoDB von CloudBees, sowie Datastore von Google. Als Ausgangssituation besteht eine MySQL Applikation. Diese beinhaltet eine Join-Operation, eine typische Operation, welche jedoch bei nicht-relationalen Datenbanksystemen nicht zur Verfügung steht. Mit diesen Applikationen wird veranschaulicht, was umgestellt und auf was verzichtet werden muss, wenn man mit einer nicht-relationalen Datenbank arbeitet.

3.2.2.1 Aufbau der MySQL-Applikation

Die Applikation beinhaltet zwei Tabellen. Eine für den Kunden (Customer) und eine für den Vertrag (Contract). Customers und Contracts werden über den Browser hinzugefügt.

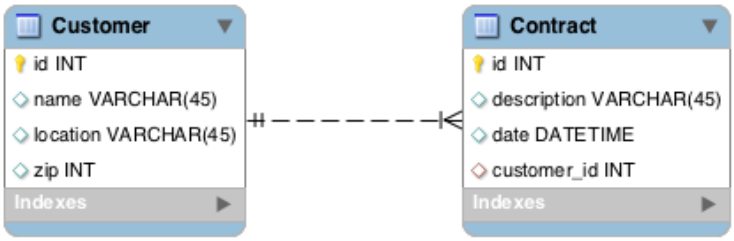


Abbildung 5 - ContractManagement Tabellen

Danach wird in der Applikation einen Join über die zwei Tabellen erzeugt. Dabei wird der Customer durch die Id in der Contract Tabelle referenziert.

3.2.2.1.1 Endresultat

Das nachfolgende Bild zeigt die Ansicht der Applikation nach einem Aufruf im Browser. Auf der linken Seite ist die Tabelle mit den Kunden zu sehen. Die Tabelle der Verträge befindet sich auf der rechten Seite. Da bei einem Vertrag nur die Id eines Kunden eingetragen ist, wird durch einen Join die dritte Tabelle erzeugt, welche unterhalb zu sehen ist.

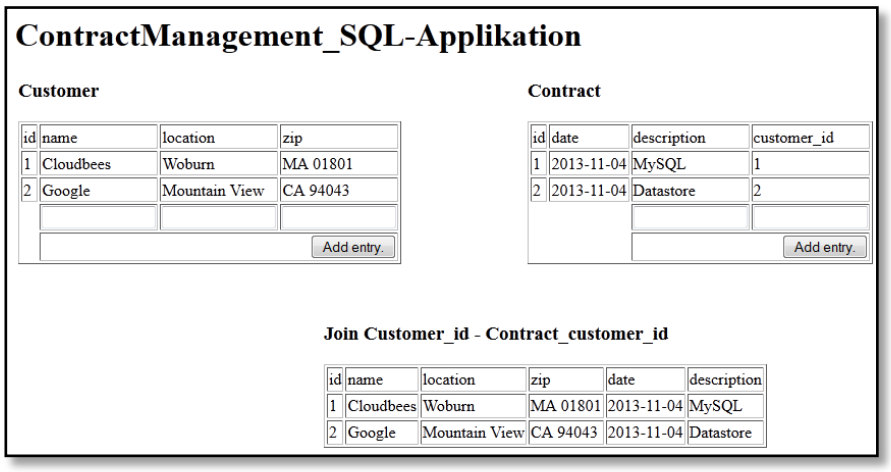


Abbildung 6 - ContractManagement Browser Output

3.2.2.2 MongoDB

MongoDB ist eine dokumentorientierte Key-Value Datenbank [Chr13] , welche Sammlungen von BSON[1] Dokumenten verwaltet [BSO13] [Mon13]. Sämtliche Objekte lassen sich beliebig verschachteln. MongoDB gehört zur verbreitesten NoSQL-Datenbank [SKe13].

3.2.2.2.1 MongoDB Data Model

Eine Instanz von MongoDB beinhaltet mehrere Datenbanken. Jede Datenbank besteht aus verschiedenen Collections. Diese wiederum bestehen aus verschiedenen Dokumenten. Ein Dokument setzt sich zusammen aus mehreren Key-Value-Pairs und hat ein dynamisches Schema. Dies bedeutet, dass unterschiedliche Dokumente in derselben Collection nicht dasselbe Schema aufweisen müssen, sondern verschieden sein dürfen.

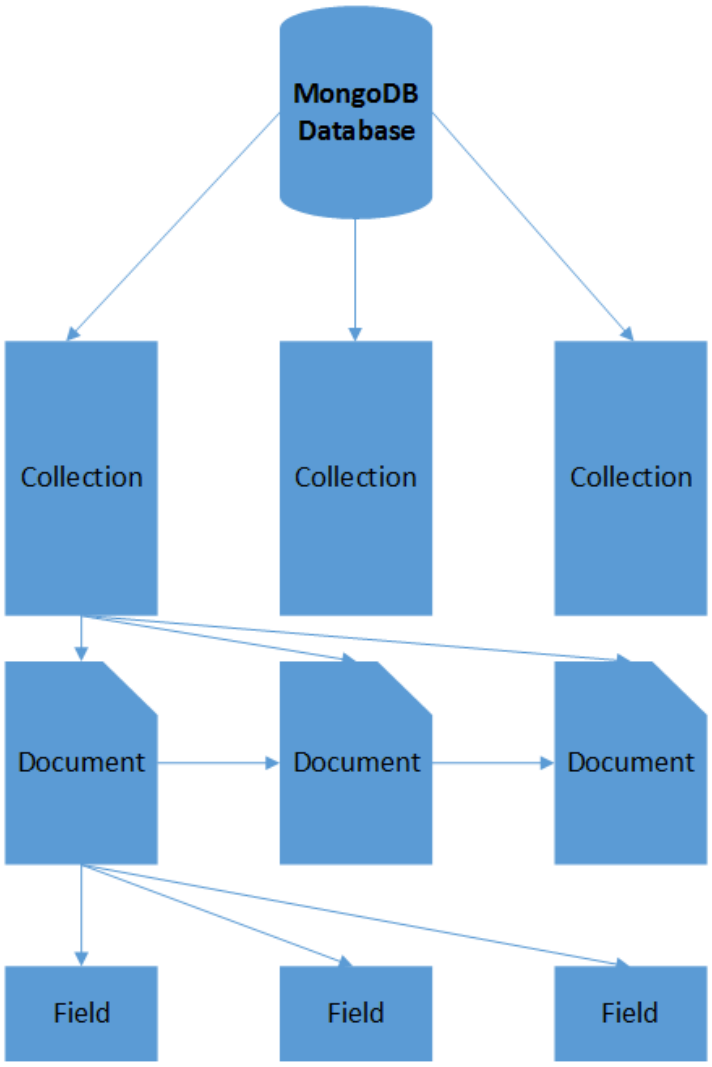


Abbildung 7 - MongoDB Aufbau

[1] "binary-encoded serialization of JSON-like documents"

3.2.2.2.2 Implementierung Join Operation

MongoDB erlaubt es, Dokumente zu verschachteln. Dabei kann einem Dokument als Attribut ein weiteres Dokument angegeben werden. Dadurch braucht es keine Anfragen auf verschiedene Collections.
Man muss sich in einem solchen Fall jedoch bewusst sein, dass man die Daten redundant speichert. Das heisst im Falle einer Änderung, wie in unserem Fall eines Customers, muss man beide Collections editieren. Dies, weil es sich nicht um eine Referenz handelt. Zudem existiert ein gekoppelter Lebenszyklus.

Nachfolgend das JSON-Objekt eines Contract Eintrages.

```
1  {
2    "_id": {
3      "$oid": "527519439c653cec20b7292e"
4    },
5    "description": "Contract 1",
6    "customer": {
7      "_id": {
8        "$oid": "527519399c653cec20b7292d"
9      },
10     "name":"Customer A",
11     "location": "City",
12     "zip": "1234"
13   },
14   "date": "2013-11-02"
15 }
```

Quellcode 1 - JSON Embedded Document

3.2.2.3 ContractManagement Google Datastore

Bei Google Datastore handelt es sich um einen verteilten, Key-Value Datenspeicherdienst [Cre13]. Dieser besitzt keine relationale Schemata wie beispielsweise MySQL. Jeder Applikation bei Google App Engine wird eine Instanz eines Datastores zugewiesen. Google Datastore unterstützt ACID-Transaktionen und nutzt Optimistic Concurrency [Wha13].

3.2.2.3.1 Entitäten

Datenobjekte in Google Datastore nennt man Entitäten. Diesen können ein oder mehrere Attribute (Eigenschaften) hinzugefügt werden [Ent13]. Diese Attribute können vom Typ Integer, Floating-points, Strings, Datumswerten oder binäre Daten sein.
Entitäten besitzen zudem einen eindeutigen Schlüssel. Dieser setzt sich aus einer Entitäts-ID und einem Typ zusammen. Die Entitäts-ID wird vom Datenspeicher bereitgestellt, kann aber auch selbst erzeugt werden. Über die Entitäts-ID kann die Entität gezielt und schnell abgefragt werden. Es kann aber auch über den Typ eine Abfrage erstellt werden. Diese liefert jedoch alle Entitäten mit dessen Typ. Möglich wäre auch, dass Entitäten wiederum Entitäten enthalten, somit wird eine Hierarchie aufgebaut. Eine weitere Möglichkeit wäre auch, eine Entitäts-ID als Attribut einer anderen Entität zu nehmen.

3.2.2.3.2 Implementierung Join Operation

Bei einem Datenspeicher wie Datastore gibt es, anders als bei relationalen Datenbanksystemen keine Joins. Dadurch ist es nicht möglich diese Operation einzusetzen , um Einträge zu verknüpfen.
Dieses Problem wurde durch folgenden Beispielcode gelöst. Innerhalb einer Iteration über sämtliche Contract-Einträge gelangt man an die Id's der Customers. Durch diese kann man mittels einer weiteren Abfrage den dazugehörenden Kunden ermitteln.

```
1  public ArrayList<CDAR_CustomerContractJoin> getJoin() {
2    ArrayList<CDAR_CustomerContractJoin> joinList =
               new ArrayList<CDAR_CustomerContractJoin>();
3    ArrayList<CDAR_Customer> customersList = getCustomers();
4
5    for (CDAR_Contract con : getContracts()) {
6      int conRefID = 0;
7      try {
8        conRefID = Integer.parseInt(con.getRefID());
9      } catch (NumberFormatException e) { }
10
11     for (CDAR_Customer cus : customersList) {
12       int cusID = 0;
13       try {
14         cusID = Integer.parseInt(cus.getID());
15       } catch (NumberFormatException e) { }
16
17       if (cusID == conRefID && cusID != 0) {
18         joinList.add(new CDAR_CustomerContractJoin(
                                      Integer.toString(cusID), cus.getName(),
                                      cus.getLocation(), cus.getZip(),
                                      con.getDate(), con.getDescription()));
19
20       }
21     }
22   }
23   Collections.sort(joinList, new CDAR_CustomerContractJoin());
24
25   //Columnnames
26   joinList.add(0, new CDAR_CustomerContractJoin("customer_id", "name",
                                     "location", "zip",
                                     "date", "description"));
27
28   return joinList;
```

Quellcode 2 - Datastore Join-Operation

```
1  public ArrayList<CADR_Customer> getCustomers() {
2    ArrayList<CDAR_Customer> customerList = new ArrayList<CDAR_Customer>();
3    Query q = new Query("Customer");
4    PreparedQuery pq = datastore.prepare(q);
5
6    for (Entity result : pq.asIterable()) {
7      customerList.add(new CDAR_Customer(
         result.getProperty("id").toString(),
         result.getProperty("name").toString(),
         result.getProperty("location").toString(),
         result.getProperty("zip").toString()));
8    }
9
10   Collections.sort(customerList, new CDAR_Customer());
11
12   //Columnnames
13   customerList.add(0, new CDAR_Customer("id", "name", "location", "zip"));
14   return customerList;
15 }
```

Quellcode 3 - Datastore getCustomers

3.2.3 Nutzung von TCP Sockets in der Cloud

Möchte man in der Cloud-Umgebung eine Socket-Applikation instanzübergreifend laufen lassen, muss die Applikation umgebaut werden. Da sich neben der eigenen Applikation auch viele Andere auf derselben Maschine befinden, kann eine Applikation nicht einfach auf einen Port binden und diesen besetzen. Dieser wäre dann für alle anderen Nutzer auf diesem Server nicht mehr belegbar. Deshalb wird jeder Applikation vom Cloud-Provider ein dynamischer Port zugewiesen. Die Applikation kann intern auf diesen Port binden und kommunizieren. Von aussen kann via Port 80 zugegriffen werden. Das Port-Mapping wird durch den Cloud-Provider übernommen.

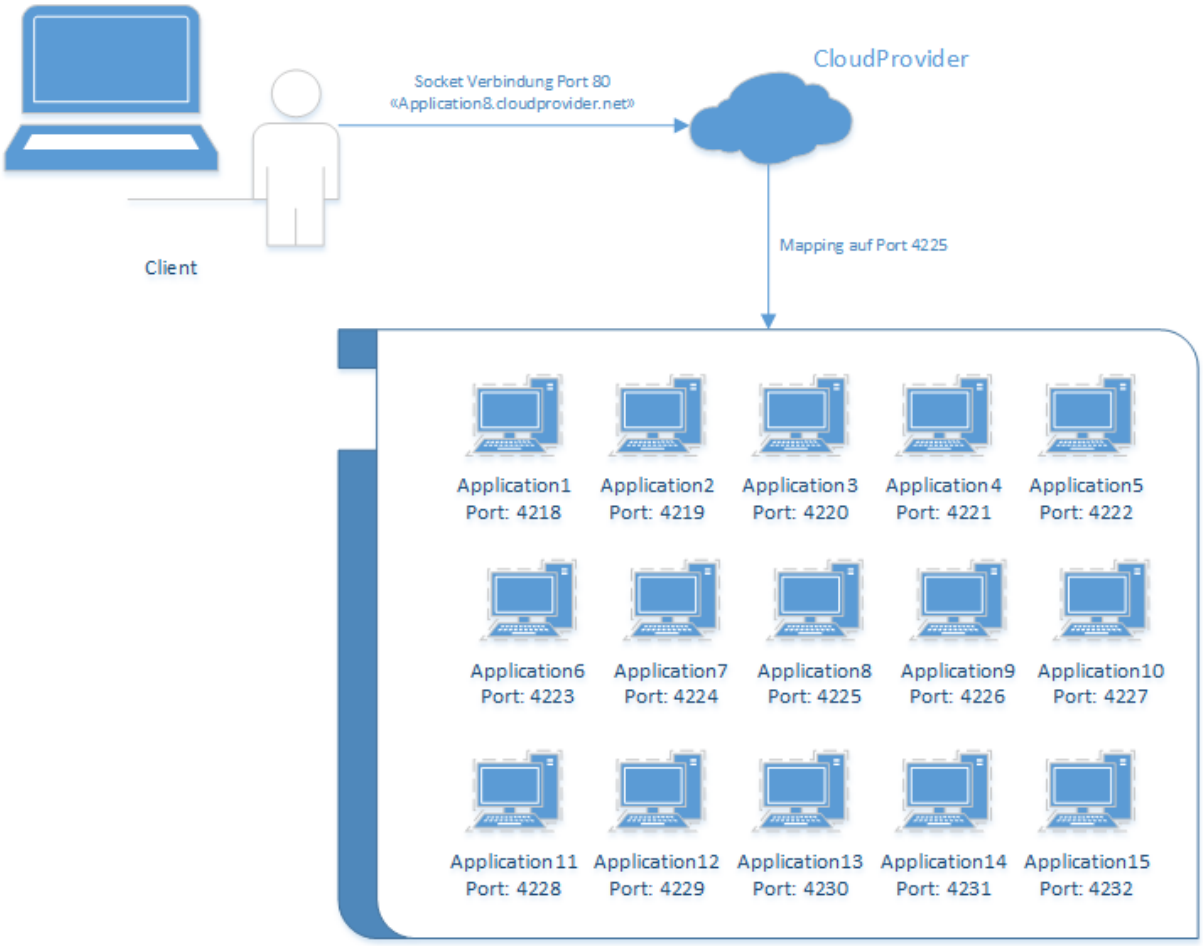


Abbildung 8 - Socket Portbinding

Dieser Port verändert sich bei jedem Deployment. Bei unseren Tests bei den Anbietern Heroku und CloudBees hat sich herausgestellt, dass nur HTTP-Traffic weitergeleitet wird [Fre13] [Pro13]. Zudem wird pro Applikation nur ein Port zugewiesen. Möchte man über mehrere Ports kommunizieren, ist ein Umbau der Applikation notwendig.

3.2.3.1 Umbau der Applikation

Damit das Port-Mapping durch den Cloud-Provider funktioniert [Fin13], muss es sich beim Traffic um HTTP-Daten handeln. Dies kann an unserem Socket-Beispiel leicht eingebaut werden, indem der Server eine HTTP 1.1 Antwort zurücksendet. Aufgerufen wird der Socket nicht über ein Client-Socket, sondern direkt über den Browser.

3.2.3.1.1 Server Socket

Durch das Einfügen der HTTP 1.1 Headerinformationen wird HTTP-Traffic erzeugt.

```
1   Date now = new Date();
2   String text = "Time is " + now.toString();
3
4   out.write("HTTP/1.1 200 OK\r\n");
5   out.write("Server: Custom Heroku Java Server\r\n");
6   out.write("Content-Length: " + text.length() + "\r\n");
7   out.write("Content-Language: en\r\n");
8   out.write("Connection: close\r\n");
9   out.write("Content-Type: text/plain\r\n");
10  out.write("\r\n");
11
12  out.write(text + "\r\n");
```

Quellcode 4 - Server Socket HTTP 1.1

3.2.3.1.2 Zugriff auf die Portnummer

Die Applikation muss zur Laufzeit wissen, welcher Port ihr zugewiesen wurde. Die einzelnen Cloud-Provider handhaben dies unterschiedlich.

3.2.4 HTTP Zeitüberschreitung

Eine Webanwendung in der Cloud, welche breit skaliert ist und nebenläufig arbeitet, ist nur wirksam, wenn die Antwortzeiten kurz sind. Vorgeschlagener Wert, welcher beispielsweise Heroku angibt, wäre bei 500ms. Dies ist nicht nur im Sinne der Benutzer der Anwendung, sondern auch der Anbieter, da auf diese Art schneller Ressourcen wieder freigegeben werden. Dies ist der Grund, warum einige Anbieter intervenieren, falls eine Anfrage zu lange dauert, und senden deshalb eine Fehlermeldung an den Client, sodass der Request abbricht.
Im Rahmen unserer Studienarbeit haben wir nebst dem Studium der Dokumentation auch einen praktischen Test durchgeführt, um zu sehen, welche Anbieter nach welcher Zeit eingreifen und eine Fehlermeldung senden.

3.2.4.1 Wartendes Servlet

Seitens der Cloud wurde eine Web Anwendung geschrieben, welche beim Aufruf 180 Sekunden wartet, bis sie eine Antwort sendet. Beim Aufruf dieses Servlets kann der Cloud Provider nun eingreifen und den Request abbrechen, sowie eine Fehlermeldung schalten. Getestet wurde dies mit Unit Tests.
In Unit Tests wird das Servlet geöffnet, falls die Antwort nun vor Ablauf der 180 Sekunden zurückkommt und es sich hierbei nicht um die korrekte Antwort handelt, gilt der Unit Test als fehlgeschlagen.

3.2.4.2 Resultat

Ein Eingriff findet bei Google App Engine sowie Heroku statt. Bei Heroku beträgt die maximale Zeit für einen Request 30 Sekunden [Req13]. Bei Google App Engine hat das Servlet bis zu 60 Sekunden Zeit, um eine Antwort zu senden [Url13]. Bei CloudBees wird auch nach drei Minuten Wartezeit noch die korrekte Antwortseite gesendet.

	Zeit
CloudBees	-
Google App Engine	60s
Heroku	30s

Tabelle 1 - HTTP Requesttimeout Zeitübersicht

Diese Zeiten werden von Google sowie Heroku fix gesetzt. Sie lassen sich nicht umstellen.

3.2.5 Map Reduce Pattern

Um das Map Reduce Pattern zu veranschaulichen und das Verhalten in der Cloud aufzuzeigen, schrieben wir ein Programm, welches das Map Reduce Pattern beinhaltet.

In diesem Map Reduce Beispiel sind eine grosse Anzahl an Agenten vorhanden. Diese Agenten haben einen Namen und eine zurückgelegte Strecke. Die Strecke besteht aus einem Start und einem Endpunkt. Start- sowie Endpunkt sind Ländernamen. Die Aufgabe des Map Reduce besteht in diesem Fall darin, alle Agenten nach dem Namen zu gruppieren und dessen Wege zusammenzufassen.

3.2.5.1 Map-Phase

In der Map-Phase teilt sich die ganze Datenmenge in kleinere Collections auf. Danach wird in der kleineren Collection über jeden Eintrag iteriert. Falls der Eintrag (Key) noch nicht in einer Zwischenliste (Key/Value) ist, fügt man diesen Eintrag hinzu.

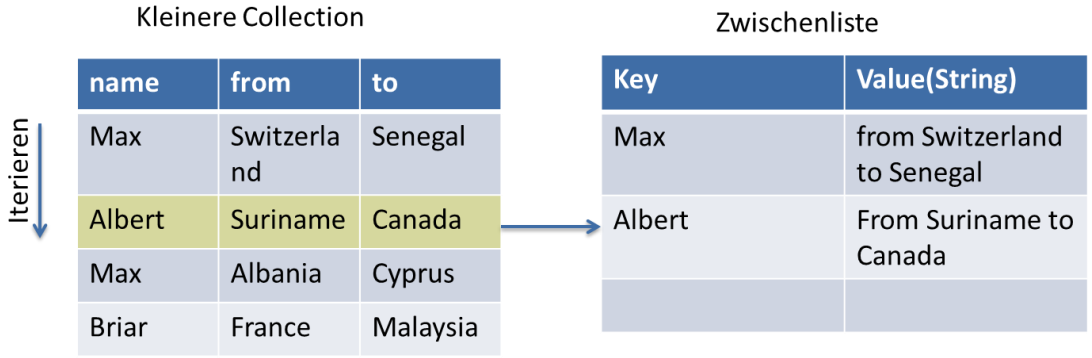


Abbildung 9 - Map Reduce Pattern: Map Phase

Falls der Eintrag jedoch bereits in der Zwischenliste existiert, wird nur der Value ergänzt. Das Iterieren über die einzelnen kleineren Collections kann jeweils parallelisiert werden. Somit findet der folgende Ablauf in mehreren Threads statt, für jede einzelne Collection Einen. Alle Threads verfolgen den gleichen Ansatz, jedoch ausschliesslich in ihrer eigenen Collection.

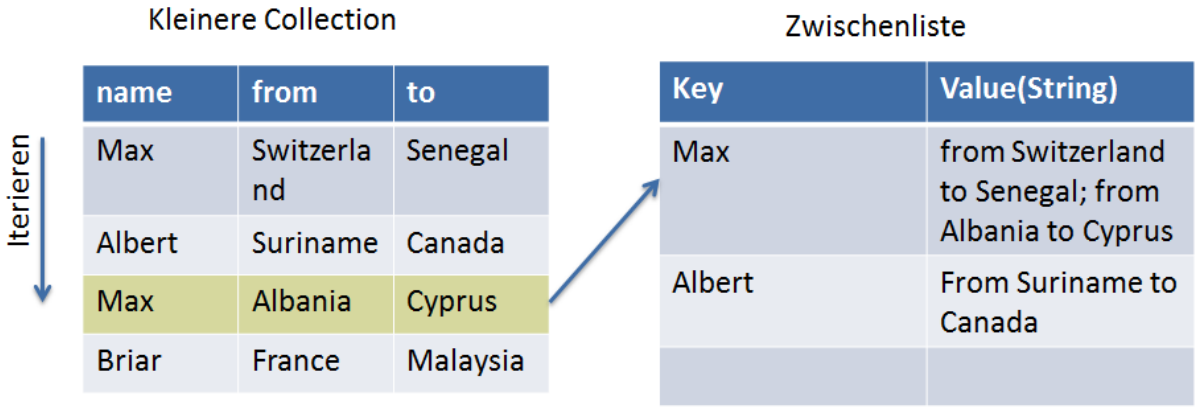


Abbildung 10 - Map Reduce Pattern: Map Phase, Ergänzung

3.2.5.2 Reduce-Phase

Nachdem die Zwischenlisten vervollständigt sind, beginnt die Reduce-Phase. In dieser Phase Reduziert man alle einzelnen Zwischenlisten zu einer Gesamtliste. In diesem gezeigten Beispiel verhält sich die Reduce-Phase sehr ähnlich der Map-Phase. Das Iterieren über die Zwischenlisten findet in diesem Beispiel seriell statt.
Falls der Zwischeneintrag (Key) noch nicht in einer Gesamtliste (Key/Value) vorhanden ist, wird dieser Zwischeneintrag hinzugefügt. Andernfalls wird nur der Value erweitert.

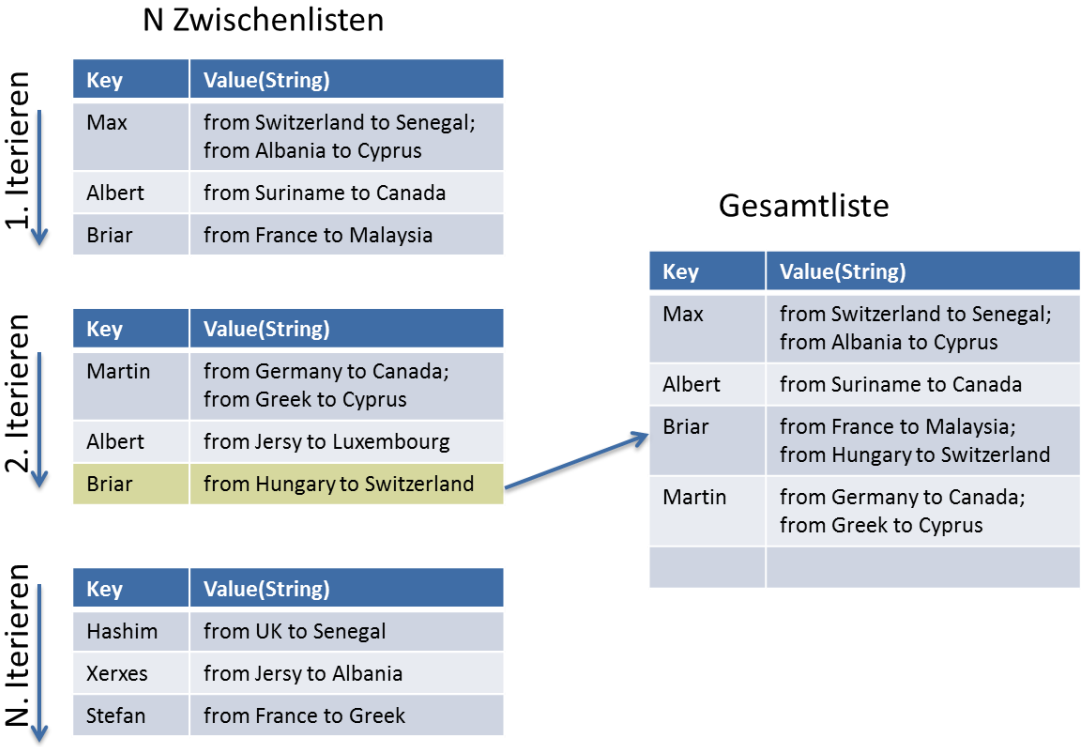


Abbildung 11 - Map Reduce Pattern: Reduce Phase

3.2.5.3 Arbeitsfluss

Folgende Grafik zeigt den ganzen Arbeitsfluss. So stellen Abbildung 9 und Abbildung 10 das Feld „Füge Teilinput in einer Map zusammen (map2)“ in untenstehender Grafik in detaillierter Form dar. Ebenso ist Abbildung 11 die umfänglichere Darstellung des Feldes „Fügt alle Maps in Liste zu einer Map zusammen (reduce)“. Felder mit der gleichen Farbe, sind in unserem Map Reduce, Methoden in der gleichen Java-Klasse. Die Wörter in Klammern widerspiegeln die Methodennamen im Code.

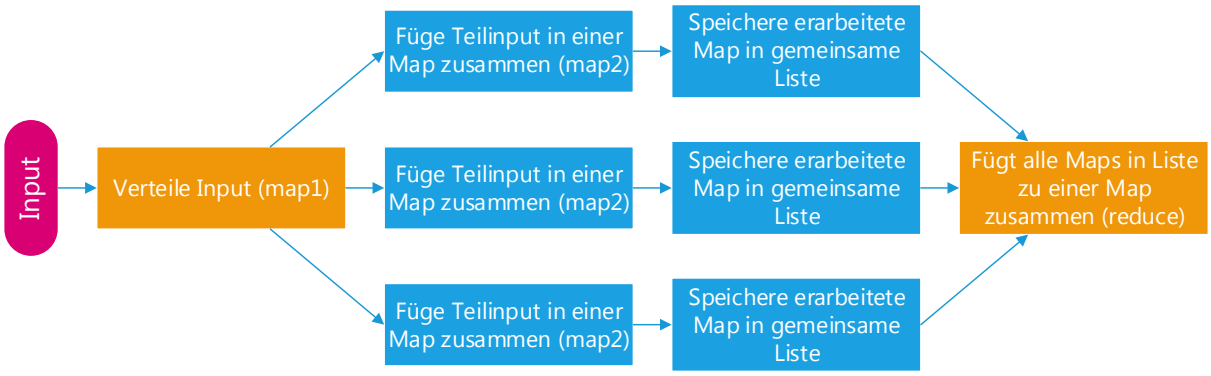


Abbildung 12 - Map Reduce Workflow

3.2.6 Rechenintensive Operationen in der Cloud

Diese Applikation zeigt dem Entwickler, dass sich nicht alle Methoden in der Cloud gleich verhalten wie lokal. Es gibt Methoden, die in der Cloud verhältnismässig sehr viel mehr Rechenzeit benötigen.

Lokal:

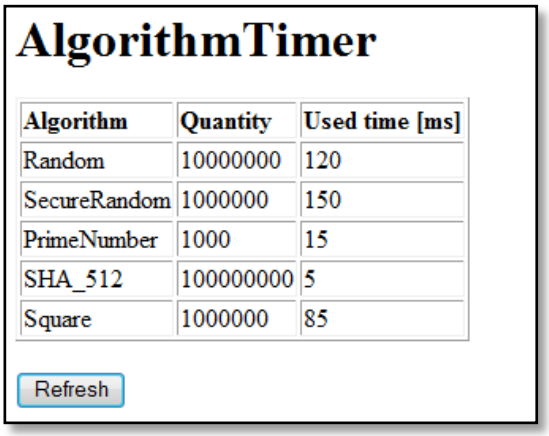


Abbildung 13 - Algorithm Timer lokal

CloudBees:

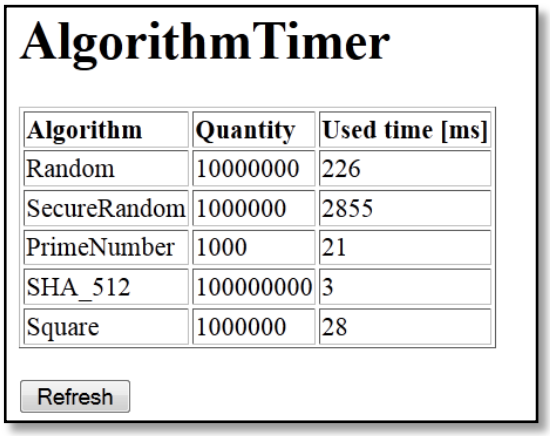


Abbildung 14 - Algorithm Timer – CloudBees

Google App Engine:

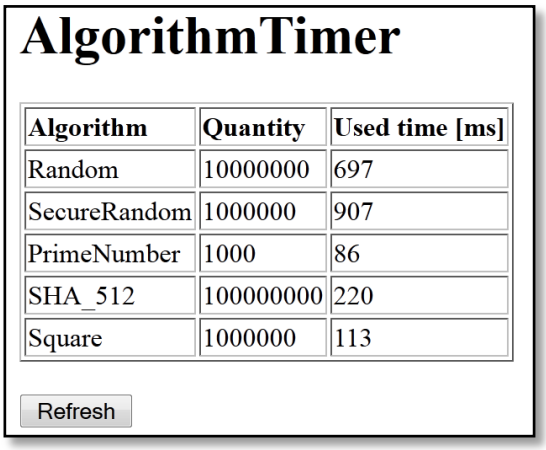


Abbildung 15 - AlgorithmTimer - Google App Engine

Heroku:

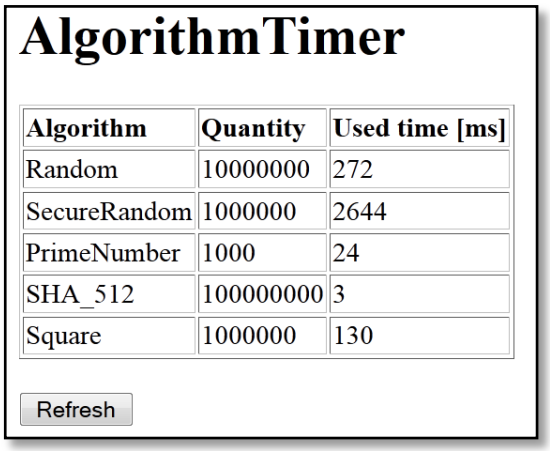


Abbildung 16 - AlgorithmTimer - Heroku

3.2.6.1 Applikationsaufbau

Der Applikationsaufbau ist sehr simpel gehalten. Die Struktur ist so aufgebaut, dass leicht weitere Algorithmen hinzuzufügen sind. Bei der Applikation wurden keine Optimierungen vorgenommen. Die Zeitmessungen sind an die Applikationsaufrufe gebunden und liefern bei einem zweiten Aufruf andere Resultate.
Dieses Projekt implementiert das Pattern „Template Method“. Allerdings ist der Aufruf der Methode direkt im den Konstruktor ausgelagert. Die Funktionalität dieses Projektes setzt dies so voraus.

3.2.6.1.1 Gemeinsame abstrakte Algorithmusklasse

Das Projekt besteht neben einem Servlet, welches für die ganze HTML-Darstellung zuständig ist, auch aus einer gemeinsamen abstrakten Klasse, von welcher alle Algorithmen erben. Sie berechnet die für den Algorithmus benötigte Zeit und dient als Basiskonstrukt für die Algorithmen.

```
1   /*
2    * abstract Class for all algorithm classes
3    */
4   public abstract class CDAR_CPUAlgorithm{
5
6     /*
7      * attribut:
8      * algorithm --> name of the algorithm e.g. SecureRandom
9      * quantity --> quantity of loops e.g. 100000
10     * totalTime --> needed time for process in [ms] e.g. 500
11     */
12    private String algorithm;
13    private int quantity;
14    private long totalTime;
15
16    public CDAR_CPUAlgorithm(int quantity){
17      this.setAlgorithm(this.getClass().getSimpleName().substring(5));
18      this.setQuantity(quantity);
19
20          long startTime = System.currentTimeMillis();
21          executeAlgorithm();
22          setTotalTime(System.currentTimeMillis() - startTime);
23    }
24
25    private void setQuantity(int quantity) {
26      this.quantity = quantity;
27    }
28
29    private void setAlgorithm(String algorithm) {
30      this.algorithm = algorithm;
31    }
32
33    protected abstract void executeAlgorithm();
34
35    protected void setTotalTime(long totalTime) {
36        this.totalTime = totalTime;
37    }
38
39    public String getAlgorithm() {
40      return algorithm;
41    }
42
43    public long getTotalTime() {
44      return totalTime;
45    }
46
47    public int getQuantity() {
48      return quantity;
49    }
50  }
```

Quellcode 5 – abstrakte Algorithmenklasse

3.2.6.1.2 Algorithmusklasse

Der jeweilige Algorithmus wird in einer eigenen Klasse ausgeführt. Diese Klasse erbt die Grundzüge der abstrakten Algorithmusklasse. Deshalb sind in dieser Klasse nur noch der eigentliche Algorithmus und die Anzahl der Durchläufe zu implementieren. Folgender Beispielcode zeigt die Implementation einer solchen Klasse. Diese Klasse verwendet den Algorithmus „Secure Random“.

```
1   /*
2    * this class generate a given quantity of securerandom integers
3    * secure random is a cryptographically strong subclass of random
4    */
5   public class CDAR_SecureRandom extends CDAR_CPUAlgorithm{
6
7     public CDAR_ SecureRandom(int quantity){
8           super(quantity);
9     }
10
11    @Override
12    protected void executeAlgorithm() {
13      java.security.SecureRandom sR = new java.security.SecureRandom();
14      for (int i = 0; i < getQuantity(); i++){
15        sR.nextInt();
16      }
17    }
18  }
```

Quellcode 6 –Algorithmenklasse

4 Schlussfolgerung

4.1 Zusammenfassung

In dieser Arbeit analysierten wir Public PaaS Cloud-Anbieter. Wir fokussierten uns dabei auf Heroku, CloudBees und Google App Engine. Mehrere Programme deployten wir auf die drei Anbieter und zogen daraus unsere Erkenntnisse. Es entstanden Programme wie eine Selbstauskunfts-Applikation, die sämtliche potentiell relevante Systeminformationen der Cloud-Plattform und der deployten Anwendung anzeigt. Eine weitere Anwendung demonstriert das Map Reduce Pattern und zeigt dessen Funktionalität in der Cloud auf. Des weiteren entwickelten wir eine Beispielanwendung, welche Interprozesskommunikation mit TCP/IP Sockets demonstriert. Auf Basis dieser gewonnenen Informationen stellten wir einen Kriterienkatalog zusammen, welcher massgebende Hinweise und Informationen über die Funktionalität und Qualitätseigenschaften der drei Provider aufweist.

Im zweiten Teil dieser Arbeit schrieben wir ein umfassendes Tutorium, dieses enthält Schritt-für-Schritt Anleitungen um alle unsere Applikationen nachzubauen und allenfalls zu individualisieren. Ebenso verfassten wir Anleitungen für alle notwendigen Werkzeuge der Provider zur produktiven Nutzung der Cloud.

4.2 Ausblick

Prof. Dr. Olaf Zimmermann plant einige Anwendungen im Cloud Computing Modul in den Übungsstunden einzusetzen. Wir hoffen, dass unsere Arbeit auch andere Leser finden wird, da das Thema gegenwärtig sehr aktuell ist und wir zudem Beispiele zeigen, welche bei Recherchen im Internet nur teilweise oder schwer auffindbar sind. Die Beispielapplikationen selbst sind sehr rudimentär gehalten, sie sind grundsätzlich in allen Bereichen ausbaubar. Der Fokus lag jedoch darin, die jeweiligen Aspekte im Bezug zur Cloud-Plattform zu zeigen.

A. Verzeichnis der Abbildungen und Tabellen

A.1. Abbildungsverzeichnis

A.2. Tabellenverzeichnis

B. Glossar

ACID
: Zu Deutsch Atomarität, Konsistenz, Isoliertheit und Dauerhaftigkeit beschreiben Eigenschaften betreffend Verarbeitungsschritte welche bei Datenbankmanagement-systemen erwünscht sind.

BSON
: Binary JSON ist eine binär kodierte Serialisierung von Dokumenten, die ähnlich wie JSON-Dokumente erzeugt werden. BSON hat zusätzliche Datentypen wie beispielsweise „Date“.

Cloud
: Cloud bezeichnet den Ansatz, IT-Infrastruktur dynamisch dem Benutzer zur Verfügung zu stellen.

CloudBees
: Cloud-Provider mit Infrastruktur bei Amazon (http://www.cloudbees.com)

Datastore
: Nicht-relationaler Key-Value Datenspeicher von Google.

Deployment
: Deployment nennt man den Prozess der Installation von Software auf PCs und Servern.

Google App Engine
: Cloud-Provider mit eigener Infrastruktur (http://developers.google.com/appengine)

Heroku
: Cloud-Provider mit Infrastruktur bei Amazon (http://www.heroku.com)

IaaS
: Infrastructure as a Service

Join
: Relationale Verknüpfung zwischen Tabellen in einer Datenbank.

JSON
: JavaScript Object Notation

MongoDB
: MongoDB ist eine dokumentenorientierte Open-Source-Datenbank.

MySQL
: MySQL ist ein verbreitetes relationales Datenbanksystem.

NoSQL
: NoSQL bezeichnet nicht-relationale Datenbanksysteme.

Optimistic Concurrency
: Optimistic Concurrency ist ein Verfahren, welches Zugriffe auf gleiche Daten konfliktarm sowie ohne Inkonsistenzen regelt.

PaaS
: Platform as a Service

POJO
: Plain Old Java Object

SaaS
: Software as a Service

Servlet
: Ein Servlet ist eine Java-Instanz, welche innerhalb eines Webservers Anfragen von Clients engegen-nehmen, verarbeiten und beantworten kann.

Socket

Ein Socket ist ein Software-Komponent, mit welchem man sich verbinden kann.

TCP

TCP ist in einem TCP/IP Netzwerk für den Transport und die Sicherung der Daten zuständig.

C. Literatur

[BSO13] *BSON*. BSON. bsonspec.org, zuletzt geprüft 2013.

[Bui13] *Buildpacks*. Heroku. https://devcenter.heroku.com/articles/buildpacks, zuletzt geprüft 2013.

[Bui131] *Buildpacks*. Heroku. https://devcenter.heroku.com/articles/third-party-buildpacks, zuletzt geprüft 2013.

[Chr13] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 107.

[Chr131] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 96.

[Chr132] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 41.

[Chr133] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 44.

[Chr134] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 49.

[Cli13] *ClickStart*. CloudBees. http://wiki.cloudbees.com/bin/view/RUN/ClickStart, zuletzt geprüft 2013.

[Clo132] *CloudBees*. CloudBees. http://www.cloudbees.com/#slide-1, zuletzt geprüft 2013.

[Com13] *Community*. CloudBees. https://github.com/CloudBees-community, zuletzt geprüft 2013.

[Cre13] *Create a Java Web Application using Embedded Tomcat*. Heroku. https://devcenter.heroku.com/articles/create-a-java-web-application-using-embedded-tomcat, zuletzt geprüft 2013.

[Dyn131] *Dynos*. Heroku. https://devcenter.heroku.com/articles/dynos, zuletzt geprüft 2013.

[Ent13] *Entities, Properties, and Keys*. Google. https://developers.google.com/appengine/docs/java/datastore/entities, zuletzt geprüft 2013.

[Fin13] *Finding IP/address/port*. CloudBees. https://developer.cloudbees.com/bin/view/Main/Finding+out+app+port+and+hostname, zuletzt geprüft 2013.

[Fre13] *Frequently Asked Questions*. CloudBees. http://wiki.cloudbees.com/bin/view/RUN/FAQ, zuletzt geprüft 2013.

[Goo13] *Google App Engine*. Google. https://developers.google.com/appengine/?hl=de, zuletzt geprüft 2013.

[Her13] *Heroku*. Heroku. https://id.heroku.com/login, zuletzt geprüft 2013.

[LBl13] Bläser L. (FS 2013): 01_Threads. In: *Parallele Netzwerkprogrammierung*. HSR.

[Map13] *MapReduce*. Wikipedia. http://de.wikipedia.org/wiki/MapReduce, zuletzt geprüft 2013.

[Mav131] *Maven*. Apache. http://maven.apache.org/, zuletzt geprüft 2013.

[Mon13] *MongoDB*. Wikipedia. http://de.wikipedia.org/wiki/MongoDB, zuletzt geprüft 2013.

[New13] *New Relic*. CloudBees. http://www.cloudbees.com/platform-service-newrelic.cb, zuletzt geprüft 2013.

[Pro13] *Process Types and the Procfile*. Heroku. https://devcenter.heroku.com/articles/procfile, zuletzt geprüft 2013.

[Req13] *Request Timeout*. Heroku. https://devcenter.heroku.com/articles/error-codes#h12-request-timeout, zuletzt geprüft 2013.

[SKe13] Keller S. (FS 2013): NoSQL_v2. In: *Datenbanken 2*. HSR.

[Soc13] *Socket*. Wikipedia. http://de.wikipedia.org/wiki/Socket_%28Software%29, zuletzt geprüft 2013.

[Url13] *Url Fetch*. Google. https://developers.google.com/appengine/docs/java/urlfetch/, zuletzt geprüft 2013.

[Wha13] *What is Google App Engine*. Google. https://developers.google.com/appengine/docs/whatisgoogleappengine, zuletzt geprüft 2013.

D. Projektplanung

D.1. Iterationsplanung

Die Iterationsplanung orientiert sich grob am Rational Unified Process (RUP) und ist unterteilt in eine *Analyse-*, *Realisierung*s- sowie *Auswertungsphase*.

Iteration	***Dauer***	***Beschreibung***
Analyse	**2 Wochen**	Projektsetup, Einarbeitung in Cloud-Provider, Vorbereitungen & Planungen
Realisierung 1	**2 Wochen**	Beginn Kriterienkatalog, Deployment erster Applikationen, Realisierung Selbstauskunfts-Applikation
Realisierung 2	**2 Wochen**	Dokumentation und Studium betreffend cloudspezifischen Eigenschaften, erste Arbeiten am Bericht
Realisierung 3	**2 Wochen**	Implementierung des Map Reduce Patterns, Realisierung der Ressourcen Monitorapplikation
Realisierung 4	**2 Wochen**	Realisierung von Socket Applikationen auf Heroku sowie CloudBees. Implementierung von Unit Tests betreffend HTTP-Requesttimeout
Realisierung 5	**2 Wochen**	Fertigstellung der Ressourcen Monitor-applikation. Umsetzung von Änderungen des Betreuers im Bericht. Grobe Fertigstellung der Dokumente.
Auswertung	**2 Wochen**	Finalisierung der Dokumentation sowie Erstellung der HSR Artefakte.

Tabelle 1 - Projektiterationsbeschreibung

Im Projektverwaltungstool (siehe D.4 Tools) ist ergänzend eine detaillierte Arbeitspaketplanung mit aktuellem Arbeitsstatus verfügbar.
Jede Iteration wird jeweils von einem Meilenstein abgeschlossen.

D.2. Meilensteine

ID	*Meilenstein*	*Termin*	*Beschreibung*
M1	**Ende Analyse**	**27.9.2013**	Die Aufgabenstellung wurde gem. Auftrag klar definiert und die Projektinfrastruktur ist aufgesetzt. Erste Einarbeitungen wurden vorgenommen und eine grobe Projekt-planung besteht.
M2	**Ende Realisierung 1**	**11.10.2013**	SelfInformation Applikation implementiert und in einer ersten Version lauffähig. Kriterienkatalog Struktur sowie erste Beispiele erstellt.
M3	**Ende Realisierung 2**	**25.10.2013**	Grundstruktur der Dokumentation erstellt. Erste spezifische Eigenschaften der Cloud Umgebung erkannt und dokumentiert.
M4	**Ende Realisierung 3**	**8.11.2013**	Ressourcen Monitorapplikation geplant. Implementation des Map Reduce Patterns mittels Java.
M5	**Ende Realisierung 4**	**22.11.2013**	Lauffähige Socket Applikationen bei den Anbietern Heroku und CloudBees.
M6	**Ende Realisierung 5**	**6.12.2013**	Ressourcen Monitorapplikation fertig-gestellt.
M7	**Ende Auswertung**	**20.12.2013**	Technischer Bericht und Tutorium fertiggestellt.
M8	**Abgabe HSR Artefakte**	**20.12.2013**	Das A0-Poster sowie die Kurzfassung der Studienarbeit sind dem Betreuer zugestellt.
M9	**Abgabe Bachelorarbeit**	**20.12.2013**	Alle abzugebenden Artefakte sind dem Betreuer zugestellt worden.

Tabelle 2 - Meilensteine

D.3. Meetings

Während der gesamten Projektdauer findet jeweils am Mittwoch um 11 Uhr ein wöchentliches Statusmeeting statt. Die Sitzungen werden jeweils von einem Studenten geführt sowie von Beiden protokolliert.
Das Projektteam stellt die Agenda der jeweiligen Sitzung bis spätestens am vorangehenden Dienstagabend bereit. Das Sitzungsprotokoll wird dem Betreuer jeweils bis zum folgenden Donnerstagabend zugestellt.

D.4. Tools

D.4.1. Projektverwaltung

Für die komplette Projektplanung, die Zeitrapportierung sowie das Issue-Management wird Redmine eingesetzt.
Der aktuelle Stand der Arbeiten am Projekt kann durch den Betreuer jederzeit eingesehen werden und wird vom Projektteam aktiv aktualisiert.

D.4.2. Entwicklungsumgebung

Zur Entwicklung der Beispielapplikationen wurde Eclipse eingesetzt.

E. Iterationsassessment

Iteration	*Status*	*Bemerkung / Begründung*
Analyse		Erste Cloud-Deployments verbunden mit langer Fehlersuche.
Realisierung 1		Realisierung der ersten eigenen Applikation hat gut funktioniert.
Realisierung 2		-
Realisierung 3		Deployment von externen Applikationen hat leider nicht funktioniert. Ticket bzw. Auftrag wurde gestrichen.
Realisierung 4		-
Realisierung 5		Viel Arbeit betreffend Dokumentation.
Auswertung		-

Tabelle 3 - Iterationsassessment

Tabelle mit den Iterationen und verschiedenen Stati mit Bemerkungen und Begründungen

F. Auswertung Zeitrapportierung

F.1.1. Aufwand pro Person und Woche

Mitglied	2013-38	2013-39	2013-40	2013-41	2013-42
Marcel Tinner	12.75	15.50	22.00	16.75	24.15
Daniel Zigerlig	8.25	18.50	25.10	20.45	19.00
Gesamtzeit	*21.00*	*34.00*	*47.10*	*37.20*	*43.15*

Tabelle 4 - Zeitrapport - Kalenderwochen 38-42

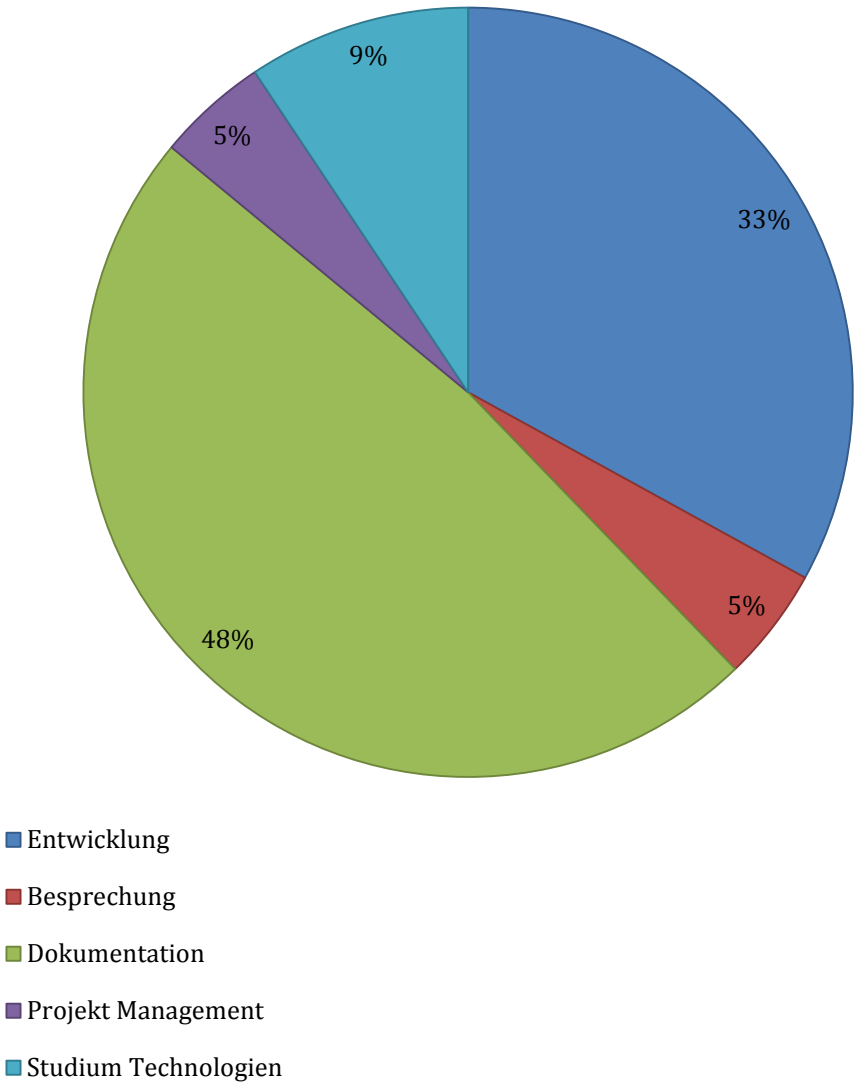
Mitglied	2013-43	2013-44	2013-45	2013-46	2013-47
Marcel Tinner	23.70	22.15	24.10	12.25	34.50
Daniel Zigerlig	26.95	17.75	17.00	19.50	33.50
Gesamtzeit	*50.65*	*39.90*	*41.10*	*31.75*	*68.00*

Tabelle 5 - Zeitrapport - Kalenderwochen 43-47

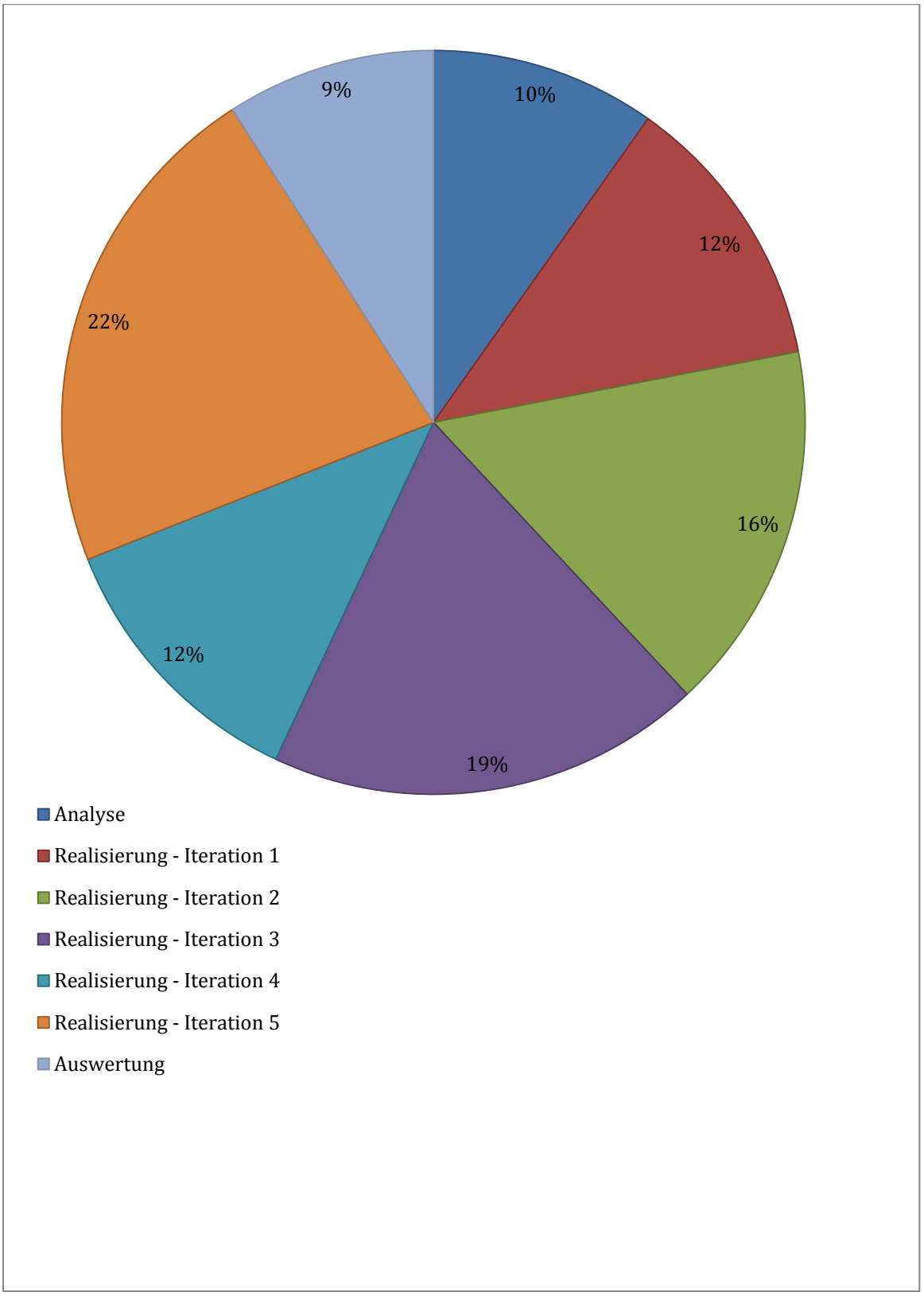
Mitglied	2013-48	2013-49	2013-50	2013-51	*Gesamtzeit*
Marcel Tinner	7.75	39.50	35.50	14.00	*304.60*
Daniel Zigerlig	10.25	38.50	30.50	16.75	*302.00*
Gesamtzeit	*18.00*	*78.00*	*66.00*	*30.00*	*606.60*

Tabelle 6 - Zeitrapport - Kalenderwochen 48-51

F.1.2. Zeitaufwand pro Aufgabenbereich



F.1.3. Zeitaufwand pro Iteration



G. Aufgabenstellung

Die folgenden drei Seiten enthalten die offizielle Aufgabenstellung dieser Studienarbeit.

Aufgabenstellung Studienarbeit
Marcel Tinner, Daniel Zigerlig

Cloud Deployment and Architectural Refactoring Lab

1. Auftraggeber und Betreuer

Diese Studienarbeit wird in Zusammenarbeit mit dem Studiengang Informatik durchgeführt.

Betreuer HSR:

Prof. Dr. Olaf Zimmermann, Institut für Software, ozimmerm@hsr.ch

2. Ausgangslage

Cloud Computing hat sich in den letzten Jahren vom stark gehypten Trendthema zur ernstzunehmenden Architekturalternative für Entwicklung und Betrieb von Web-Anwendungen und anderer Software entwickelt. Neben traditionellen IT-Anbietern wie Microsoft, HP und IBM sind Internetfirmen wie Amazon, Google und Salesforce im Public Cloud Markt aktiv; zahlreiche Open Source Assets erlauben den Aufbau von Private Clouds. Anwendungsarchitekten und Entwickler sind daher mit einer Vielzahl neuer Designoptionen und Technologien konfrontiert, z.B. nichtrelationale Speichertechniken (NoSQL), Message-Oriented Middleware mit At-Least-Once Delivery, und Servervirtualisierung.

Zum Frühjahrssemester 2014 wird die HSR deshalb ein Aufbaumodul Cloud Computing anbieten. Für die Durchführung bestimmter Übungslektionen in diesem Modul werden noch passende Beispielsanwendungen gesucht, welche auf ausgewählten Cloud-Providern deployed werden können.

3. Ziele und Aufgabenstellung

Ziel dieser Studienarbeit ist es, Java-Beispielanwendungen zur Verwendung in den Übungen zur Vorlesung Cloud Computing (ab FS 2014) zu erstellen und reproduzierbar auf ausgewählte Cloud-Provider zu deployen. Aufgabe der Beispielanwendungen ist es, die zentralen Cloud-Konzepte zu veranschaulichen und die Cloud-Provider vergleichbar zu machen z.B. bzgl. der definierenden Cloud-Eigenschaften (OSSM), Entwicklungs- und Betriebsaufwand sowie Service Levels.

Die Cloud-Provider werden gemeinsam mit dem Betreuer ausgewählt. Stand zum Projektstart:

http://www.cloudbees.com

https://www.heroku.com/ (PaaS)

http://www.openstack.org/software/start (private cloud)

http://projects.puppetlabs.com/projects/pupet

evtl. auch Google App Engine, IBM Cloud Offerings und/oder Microsoft Windows Azure. Änderungen können im Rahmen des Projektmanagements vereinbart werden.

Aufgrund der geplanten Verwendung der erstellten Programme sind folgende Software-Qualitätsattribute von besonderer Bedeutung: Wartbarkeit (Lesbarkeit des Codes), Flexibilität (Konfigurierbarkeit), Benutzerfreundlichkeit (der Skripte und Beispielkonfigurationen und der Dokumentation), Interoperabilität und Portierbarkeit (der Beispielprogramme). Eine nicht cloudbasierte JEE-Beispielanwendung wird zu Beginn der Arbeit zur Verfügung gestellt.

4. Ergebnis

Das Ergebnis besteht neben lauffähigen Beispielsanwendungen auch aus einem Bericht mit einem Kriterienkatalog, mit welchem man die verschiedenen Provider vergleichen kann.

5. Unterstützung

Die erwartete und effektiv erhaltene Unterstützung wird durch die Studenten in Sitzungsprotokollen definiert und im BA-Bericht dokumentiert.

6. Zur Durchführung

Mit dem HSR-Betreuer finden wöchentlich Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf zu veranlassen.

Alle Besprechungen, bei denen eine Vorbereitung durch den Betreuer nötig ist, sind von den Studenten mit einer Traktandenliste vorzubereiten. Beschlüsse sind in einem Protokoll zu dokumentieren.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. Arbeitszeiten sind zu dokumentieren.

Vorstudie, Anforderungsdokumentation und Architekturdokumentation sollten im Laufe des Projektes mittels Milestone mit dem Auftraggebern und dem Betreuer in einem stabilen Zustand abgenommen werden. Zu den abgegebenen Arbeitsresultaten wird ein vorläufiges Feedback abgegeben. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Dokumentation.

7. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Die Dokumentation ist vollständig auf CD/DVD in zwei Exemplaren abzugeben. Zudem ist eine kurze Projektresultatdokumentation im Wiki von Prof. Zimmermann zu erstellen. Weiterhin erwünscht ist die Erstellung eines kurzen Videos.

8. Termine

Siehe auch HSR-Webseiten.

18.09.2013	Beginn der Studienarbeit, Ausgabe der Aufgabenstellung durch die Betreuer
16.12.2013	Abgabe A0-Poster
16.12.2013	Abgabe Kurzfassung Studienarbeit für Betreuer
20.12.2013	Kurzfassungs-Überarbeitung an Studiengangssekretariat
20.12.2013	Abgabe Bericht

Allfällige weitere Termine sind am Sekretariat der Abteilung Informatik zu erfragen und sollten entsprechend in einem Sitzungsprotokoll dokumentiert werden.

9. Beurteilung

Eine erfolgreiche Studienarbeit zählt 8 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert.

Siehe http://studien.hsr.ch/allModules/23498_M_SAI.html für die Modulbeschreibung der Studienarbeiten.

Für die Beurteilung sind die HSR-Betreuer verantwortlich unter Einbezug des Feedbacks der Projektpartner.

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/5
2. Berichte (Abstract, Mgmt Summary, techn. u. persönliche Berichte) sowie Gliederung, Darstellung und Sprache der gesamten Dokumentation.	1/5
3. Inhalt*)	3/5

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit festgelegt.

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Studienarbeiten.

Rapperswil, den 1. Oktober 2013

Prof. Dr. Olaf Zimmermann
Institut für Software
Hochschule für Technik Rapperswil

H. Meetingprotokolle

Protokoll (18.09.2013, 11:00 – 12:15)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Aktueller Stand der Vorbereitung der Cloud-Vorlesung und der Übungen
- Diskussion zur genauen Aufgabenstellung
- Logistik, Administratives, Organisatorisches, Termine (ausgehend von den Vorgaben der Studiengangsleitung, siehe z.B. Web HSR intern)
- Ihre Fragen
- Nächste Schritte

Beschlüsse:

- Wöchentliche Meetings immer am Mittwoch, 11Uhr, Zimmer: C.3106
- Bücher: Cloud Computing Pattern als Lernliteratur und allenfalls Blogs
- Cloudfähigkeit von Applikationen/Software analysieren
- PaaS im Public-Cloudbereich als Schwerpunkt
- Zielgruppe: Softwarearchitekten die für Clouduser arbeiten.
- Probleme bei der Arbeit mit den Cloud-Anbietern dokumentieren
- Fünf Webanwendungen in der Cloud sollen am Schluss lauffähig sein
    - VSS
    - Zimmermann
    - DDD
    - (Namics-App von Challenge Projekt)
- Erstellen einer Auflistung von Empfehlungen betreffend Cloud-Anbietern mit Kriterien und Vergleichen
- Festlegung der Aufträge bis nächste Woche, siehe Punkt „Aufgaben“
- Projektablauf-Methodik durch Studierende wählbar
- Virtual Server wird durch Herrn Zimmermann beantragt

Aufgaben:

- Projektablaufmethodik festlegen
- Einrichten des Virtual Servers
- Grober Projektplan
    - In der 2. Woche konkreter Projektplan erstellen
- Mit Heroku und CloudBees experimentieren
- Aufgabenstellung der SA formulieren
- DDD auf Cloud deployen (Maven, JBoss)
- Dokument Abläufe und Regelung der Studien- & Bachelorarbeiten

Sitzungsvorbereitung (25.09.2013, 11:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Besprechung SA Aufgabenstellung
- Auswahl der Projektmethodik
- Kurze Besprechung des Sitzungsprotokolles vom 18.09.2013
- Stand bisheriger Arbeiten.
    - Heroku
    - CloudBees
- Besprechung Projektplan
- Weiterer Verlauf

Fragen

- Verlauf des Projektes/Aufgabenstellung
- Erstellen von Anleitungen (Zielgruppe)
- Anzahl zu behandelnde Übungen
- Abgabe der erarbeiteten Übungen
- Unterlagen der Vorlesung bzw. Übungsinput/Vorgabe
- über 16h pro Woche

Protokoll (25.09.2013, 11:00-13:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Besprechung SA Aufgabenstellung
- Auswahl der Projektmethodik
- Kurze Besprechung des Sitzungsprotokolls vom 18.09.2013
- Stand bisheriger Arbeiten
    - Heroku
    - CloudBees
- Besprechung Projektplan
- Weiteres Vorgehen

Beschlüsse:

- Traktanden jeweils vor Dienstagabend, 17:00 Uhr eingereicht
- Protokoll jeweils bis folgenden Donnerstagnachmittag versendet
- Voraussichtlich fünf Webanwendungen erstellt
    - vier bestehende Anwendungen welche womöglich noch angepasst werden müssen
    - zusätzliche Webanwendung im Eigenbau (Gegenrechnung an Anbieter / Selbstauskunft)
- Code in Englisch geschrieben
- Bericht in Deutsch verfasst
- Pro Woche zwei neue Kriterien für Kriterienkatalog eruiert
- Deployte Anwendungen testen
- Am Ende des Projektes 10-25 Kriterien erstellt

Aufgaben:

Studenten:

- Kriterienkatalog erstellen
- Vergleichskriterien der Cloud-Anbieter illustrieren
- Kriterien bei den Cloud-Anbietern testen, analysieren und dokumentieren
- Projektplan für die nächsten zwei Wochen anpassen
- Unsere SA-Aufgabenstellung in die von Herr Zimmermann integrieren
- Entwicklung der Selbstauskunfts-Webapplikation beginnen
- Herr Zimmermann im Redmine freischalten
- Google App Engine als mögliche Alternative bei genügender Zeit im Projekt anschauen

Betreuer:

- SA-Aufgabenstellung an Studenten senden.
- Erstellte Anleitungen lesen und kommentieren

Sitzungsvorbereitung (2.10.2013, 11:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- SA Aufgabenstellung
- Kriterienkatalog
- SelfInformation Applikation
- Anleitung Google App Engine
- Weiterer Verlauf

Fragen

- Erstellung Wiki-Eintrag
- Erstellung Video
- Google-Engine Anleitung

Protokoll (02.10.2013, 11:00-12:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- SA Aufgabenstellung
- Kriterienkatalog
- SelfInformation Applikation
- Anleitung Google App Engine
- Weiterer Verlauf

Beschlüsse:

- Kriterienkatalog Frageteil ausführlicher
- Kriterien nach Auftrittsreihenfolge sortiert
- Anfang des Meetings Zusammenfassung über geleistete Arbeit und Probleme
- Werbevideo für Studienarbeit, kurz und prägnant
- WIKI zum Schluss der SA mit Abstract und Video

Aufgaben:

Studenten:

- Bei Kriterienkatalog eine Einleitung zur Frage hinzufügen
- Definitiven Namen der Studienarbeit überlegen
- SelfInformation erweitern mit xml-Funktionalität
- Weitere Kriterien dokumentieren
- OSSM-Kriterien dokumentieren
- Kurz in NoSQL und Map Reduce einlesen
- Möglichkeit der eigenen URI's mit den Cloud-Anbietern eruieren
- Eigene URI-Verwaltung beim Cloud-Anbieter in Kriterienkatalog aufnehmen
- State-Machine-Diagram für Visualisierung des DeployProzesses bzw. Lebenszyklus der Applikation erstellen
- SelfInformation-App anpassen mit C Funktionen erweitern
- Vereinbarung lesen und besprechen
- Dyno-Konzept verstehen und dokumentieren

Betreuer:

- SA-Aufgabenstellung anpassen.
- IFS Bsp.-Anwendungen bis 16.10 beschaffen

Sitzungsvorbereitung (9.10.2013, 11:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Meeting am 31. November
- Name Studienarbeit
- Überblick geleistete Arbeit
- SelfInformation
    - Umgebungsvariablen
    - XML Integration
- C-Funktionen / Bash-Befehle SelfInformation-Applikation
- CloudBees -> Amazon EC2 Api
- Bericht
    - Formatierung und Aufbau
    - Kriterienkatalog
    - Dyno Konzept
    - Lebenszyklus Cloud-Applikation
    - Eigener Domainname
    - Inhalt allgemein bis zum jetzigen Zeitpunkt
- Map Reduce / NoSQL
- Weiterer Verlauf
    - Projektplan
    - Dyno Konzept CloudBees/Google App Engine
    - evtl. SelfInformation Python (psutil)
    - Dokumentation SelfInformation

Fragen

- Bericht (siehe Traktanden)
    - Eigener Domainname (wohin zeigt A-Record?)
- Literaturverzeichnis/Quellenverzeichnis
- SelfInformation Python (psutil)
- OSSM Kriterien
- Zitieren aus anderer BA/SA

Protokoll (09.10.2013, 11:00-12:15)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Meeting am 31. Oktober
- Name Studienarbeit
- Überblick geleistete Arbeit
- SelfInformation
    - Umgebungsvariablen
    - XML Integration
- C-Funktionen / Bash-Befehle SelfInformation-Applikation
- CloudBees -> Amazon EC2 Api
- Bericht
    - Formatierung und Aufbau
    - Kriterienkatalog
    - Dyno Konzept
    - Lebenszyklus Cloud-Applikation
    - Eigener Domainname
    - Inhalt allgemein bis zum jetzigen Zeitpunkt
- Map Reduce / NoSQL
- Weiterer Verlauf
    - Projektplan
    - Dyno Konzept CloudBees/Google App Engine
    - evtl. SelfInformation Python (psutil)
    - Dokumentation SelfInformation

Beschlüsse:

- Meeting vom 30. Okt. auf 25. Okt. verschoben. (12:00 Uhr)
- SA-Titel wird auf „Cloud Deployment & Architectur Refactoring Lab“ geändert
- SelfInformation-Applikation
    - C-Funktionalität entfällt
    - psutil max. eine Stunde investieren
- Map Reduce verschoben um eine Woche
- NoSQL
    - eigenes Servlet mit JDBC und mindestens einem Join
    - Google geringer priorisiert

Aufgaben:
Studenten:

- Bis spätestens morgen(10.10.13) erwünschte Änderungen an der SA-Aufgabenstellung mitteilen
- SelfInformation-Applikation
    - XML-Code der validieren
    - XML in JSON parsen
    - Einsatz von psutil untersuchen
    - Relevanz der Applikation dokumentieren
- Einarbeiten in NoSQL
    - Möglichkeiten betreffen File Storage evaluieren
    - Mindestens zwei NoSQL-Kategorien implementieren
    - Vorteile und Änderungen mit NoSQL dokumentieren

Betreuer:

- angepasste SA-Aufgabenstellung heute (09.10.13) an Studenten senden.
- SA-Aufgabenstellung beim Studiengangsekretariat einreichen
- SA-Bericht bis zum nächsten Meeting durchlesen
- Link bezüglich „seven Databases in seven Weeks“ senden

Sitzungsvorbereitung (16.10.2013, 11:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Überblick geleistete Arbeit
- SelfInformation
    - Package Layering
    - Dokumentation
- JDBC-Servlet
- Bericht Besprechung
- Map Reduce / NoSQL
- Weiterer Verlauf
    - Bericht
        - App Deployment Heroku Toolbelt
        - Kriterienkatalog: Anleitung/Dokumentation trennen
        - Dyno Konzept allgemein
        - NoSQL Dokumentation
    - CRUD Servlet MongoDB

Fragen:

- Dokumentation JDBC Servlet
- Dokumentation MongoDB

Protokoll (16.10.2013, 11:00-12:15)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Überblick geleistete Arbeit
- SelfInformation
  - Package Layering
  - Dokumentation
- SQLJoin-Servlet
- Bericht Besprechung
- Map Reduce / NoSQL
- Weiterer Verlauf
  - Bericht
    - App Deployment Heroku Toolbelt
    - Kriterienkatalog: Anleitung/Dokumentation trennen
    - Dyno Konzept allgemein
    - NoSQL Dokumentation
  - CRUD Servlet MongoDB

Beschlüsse:

- Map Reduce verschoben um eine Woche
- NoSQL
  - Nur kurze Anleitung zu NoSQL
  - Prioritätenverteilung
    - MongoDB
    - Google App Engine Datastore
    - Apache Cassandra
    - Riak
    - LinkedIn
    - Azure

Aufgaben:

- NoSQL
  - Applikationen nach oben erwähnten Prioritäten erstellen
  - Anleitung zur NoSQL Applikation erstellen
- Kurze Anleitung zum SQL-Servlet erstellen
- SelfInformation
  - Technische Eigenschaften dokumentieren
  - Beschreibung für Anwender erstellen
  - JUnit Implementation
- Überarbeitung Bericht gemäss Kommentare Betreuer
- Map Reduce
  - Szenario ausdenken (Map Reduce Pattern ohne Hadoop Overkill)
  - Machbarkeit Map Reduce und MongoDB evaluieren
- „Erklärung“ unterschreiben

Sitzungsvorbereitung (23.10.2013, 11:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Überblick geleistete Arbeit
- Bericht Überarbeitung
- ContractManagement DataStore
- ContractManagement MongoDB
- Dokumentation ContractManagement
- Dokumentation SelfInformation
- SelfInformation UnitTests
- SelfInformation wichtige Methoden
- Map Reduce MongoDB Evaluation
- Map Reduce Anwendungsbeispiel
- Weiterer Verlauf

Protokoll (23.10.2013, 11:00-12:15)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Bericht Überarbeitung
- ContractManagement DataStore
- ContractManagement MongoDB
- Dokumentation ContractManagement
- Dokumentation SelfInformation
- SelfInformation UnitTests
- SelfInformation wichtige Methoden
- Map Reduce MongoDB Evaluation
- Map Reduce Anwendungsbeispiel
- Weiterer Verlauf

Beschlüsse:

- Abgabe Servlets an Dozent am 25.10.2013
    - Code Review
    - Nutzung in Konferenz
- ContractManagement
    - Mindestens 3 Spalten pro Tabelle
    - Fehleingaben bis in den Data Access Layer weitergeben und dort behandeln
- Map Reduce
    - Eigenbau des Patterns in Java
    - Insurance Versicherungsagenten welche Kunden vor Ort besuchen

Aufgaben:

Studenten:

- JUnit Tests
    - überprüfen ob Methode einen Wert zurückgibt
    - überprüfen ob Applikation ohne Exceptions durchläuft
- ContractManagement
    - MongoDB Dokumentieren warum ID-Wert anders
    - Refactoring allgemein
    - Spaltennamen Join-Tabelle ändern
    - Namen spezifisch anpassen
    - Key Value Einarbeitung und Überarbeitung
    - Weitere Attribute hinzufügen
- Map Reduce
    - Nachbau mit Java
- Abklären
    - JBoss Nutzungsmöglichkeit in Heroku
    - Standalone-Applikation in den Clouds
    - Unittests-Möglichkeiten in den Clouds

- Applikationen S. Keller
    - Analysieren und Deployment auf Cloud Provider
- Dokumentation
    - Überarbeitung ab Freitag

Betreuer:

- Codereview
- ContractManagement-Dokumentation auf Vollständigkeit prüfen

Sitzungsvorbereitung (25.10.2013, 12:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Überblick geleistete Arbeit
- ContractManagement
- SelfInformation
- Map Reduce Nachbau
- Standalone Java Applications
- Unit Tests in der Cloud
- Weiterer Ablauf

Fragen:

- Abgabe der Applikationen
- ContractManagement Anzeige der Spaltennamen
- Map Reduce Java Applikation Umfang
- ContractManagement MongoDB Verschachtelung (Löschen/Editieren von Customers geplant?)

Protokoll (25.10.2013, 12:00-13:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Überblick geleistete Arbeit
- ContractManagement
- SelfInformation
- Map Reduce Nachbau
- Standalone Java Applications
- Unit Tests in der Cloud
- Weiterer Verlauf

Beschlüsse:

- Abgabe SelfInformation (Heroku) an Dozent am 28.10.2013
- ContractManagement
    - Validierung (Existenz der referenzierten CustomerID) nicht in der obersten Schicht
- Map Reduce
    - Kein generischer Eigenbau
- Standalone Java Applications
    - Versuch mit Sockets aus VSS-Vorlesung
- Unit Tests in der Cloud
    - Praktische Recherche
- Bericht
    - Strukturierung in drei Teile

Aufgaben:

- SelfInformation für Heroku anpassen und exportieren
- Bericht anpassen gemäss Korrektur O. Zimmermann
- Bericht restrukturieren
- Java Sockets aus VSS-Vorlesung in Cloud laufen lassen
- Map Reduce implementieren und dokumentieren

Sitzungsvorbereitung (6.11.2013, 11:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Überblick geleistete Arbeit
- SelfInformation Heroku Abgabe
- ContractManagement
    - Validierung ob Customer existiert
- InsuranceAgentWay
- Standalone Java Applications
    - Sockets
- Unit Tests in der Cloud
    - Ticket verschoben
- Keller Applikationen
- Dokumentation
    - Struktur der beiden Dokumenten
    - Standalone Java Applications
    - Insurance Agent Way
- Kurzbericht IAAS-Klausurtagung
- Weiterer Verlauf

Protokoll (6.11.2013, 11:00-12:30)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Überblick geleistete Arbeit
- SelfInformation Heroku Abgabe
- ContractManagement
    - Validierung ob Customer existiert
- InsuranceAgentWay
- Standalone Java Applications
    - Sockets
- Unit Tests in der Cloud
    - Ticket verschoben
- Keller Applikationen
- Dokumentation
    - Struktur der beiden Dokumenten
    - Standalone Java Applications
    - Insurance Agent Way
- Kurzbericht IAAS-Klausurtagung
- Weiterer Verlauf

Beschlüsse:

- Bei Problemen mit Keller-App Kontakt aufnehmen
    - NeoMapApp fällt weg
- AppFog als weitere PaaS Alternative
- Wissensaustausch mit Student aus Stuttgart
- Abgabe Bericht und sämtlicher SA relevanten Daten per 25. November an Betreuer
- Statische IP's innerhalb Cloud-Applikation wenig sinnvoll

Aufgaben:

- Keller Applikationen in Cloud deployen
- HTTP-Requesttimeout evaluieren
- Kriterium CPU Intensive Rechenoperationen evaluieren
- Unit Tests in Cloud umsetzen
- Client- und Serversocket innerhalb Cloud-Applikationen austesten
- Ressourcen-Monitorapplikation entwickeln und in Cloud deployen Als erster Schritt Logging Möglichkeiten evaluieren
- Map Reduce Refactoring durchführen
    - Anzahl Threads anzeigen
- Bericht anpassen
- Mit Bing Shao Kontakt aufnehmen

Sitzungsvorbereitung (13.11.2013, 11:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Überblick geleistete Arbeit
- Map Reduce Refactoring
- Ressourcen-Monitorapplikation
- Socketapplikation
- Unit Test in der Cloud
- CPU-Intensive Rechenoperationen
- HTTP-Requesttimeout
- Keller Applikationen
- Besprechung abgegebener Dokumente bzw. Applikationen
- Kurzbericht MS Windows Azure Schulung an der ETH
- Verlegung des Meetings vom 20.11.2013
- Weiterer Verlauf

Protokoll (13.11.2013, 11:00-12:30)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Überblick geleistete Arbeit
- Ressourcen-Monitorapplikation
- Socketapplikation
- Unit Test in Cloud
- CPU-Intensive Rechenoperationen
- HTTP-Requesttimeout
- Keller Applikationen
- Besprechung abgegebener Dokumente bzw. Applikationen
- Kurzbericht MS Windows Azure Schulung an der ETH
- Verlegung des Meetings vom 20.11.2013
- Weiterer Verlauf

Beschlüsse:

- Ressourcen-Monitorapplikation
    - Informationen in Log-Files schreiben
    - keine Darstellung als Übersicht nötig
    - u. U. Verwendung von Annotations falls nicht zu aufwendig
- Meeting am 20.11.2013 wird auf 22.11.2013, 12:00 Uhr verschoben
- Azure Account möglich
    - Ende SA Azure austesten falls Zeit vorhanden
- CPU-Auslastung auslesen über Java möglich
- Verwendung und Installierbarkeit der Applikationen auf Übungsrechnern testen

Aufgaben:

- Keller Applikationen in Cloud deployen
- eigene Annotations / Hilfsklasse für Logs erstellen
- Socketapplikation auch mit CloudBees
- Evaluieren der Möglichkeit, den HTTP Request-Timeout selbst zu setzen
- Bericht bis Freitag 15.11.2013 auf Dropbox stellen
- Die Verständlichkeit und Komplexität der Applikationen und Anleitungen überprüfen
- Codekommentare an sinnvollen Stellen im Code

Sitzung vom 22.11.2013, 12:00

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Java Annotations
- Socketapplikation CloudBees
- HTTP-Request-Timeout
- RessourcenMonitor-Applikation
- Abgabe Bericht 25.11.2013
- Bachelorarbeit
- Windows Azure

Fragen:

- Use Cases, Packagestruktur im technischen Bericht oder Tutorium?

Protokoll (22.11.2013, 12:00-13:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- Java Annotations
- Socketapplikation CloudBees
- HTTP-Request-Timeout
- RessourcenMonitor-Applikation
- Abgabe Bericht 25.11.2013
- Kriterienkatalog
- Windows Azure
- Bachelorarbeit

Beschlüsse:

- Bei genügend Zeit
    - Appfog in den Bericht integrieren
    - Azure austesten
- XML, JSON Export in RessourcenMonitor-Applikation

Aufgaben:

Studenten:

- Kriterienkatalog überarbeiten und erweitern
- RessourcenMonitor-Applikation weiter implementieren
- Bericht bis am 25.11 auf Dropbox stellen
- Cloud Foundry untersuchen
- Möglichkeit von Lifecycle-Callback evaluieren
- Review-Ziele setzen

Betreuer:

- Code-Review
- Dokumentation-Review

Sitzung vom 27.11.2013, 11:00

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- RessourcenMonitor-Applikation
- Bericht Review

Protokoll (27.11.2013, 11:00-12:30)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- RessourcenMonitor-Applikation
- IntTe Miniprojekt Deployment auf Heroku
- Bericht Review

Beschlüsse:

Folgende Beschlüsse betreffen die Dokumentation der Arbeit:

- Alternativen zu Sockets erwähnen
- Bei Hauptkapiteln jeweils eine Zusammenfassung
- Begriffe falls möglich in deutsch
- Mehr aktiv- statt passiv-Form
- Fokus auf die Zielgruppe
- Management Summary, Abstract und Danksagungen nur im Hauptbericht
- Namen der Applikationen nicht in den Kapitelüberschriften RessourcenMonitorApplikation
- Ohne Verwendung von Backend-Threads

Aufgaben:

- Umsetzen sämtlicher vom Betreuer verfassten Änderungen an den Dokumenten
- Weiterarbeit an der RessourcenMonitor-Applikation
- Dokumentation vom doppelten Lifecycle bei Google App Engine
- Dokumentation von Socket-Alternativen wie bspw. RabbitMQ

Sitzung vom 4.12.2013, 11:00

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- RessourcenMonitor-Applikation
- Bericht
    - Abstract
    - Management Summary
    - Weitere Arbeiten
    - Fragen
- Video
- Folienvergleich Aufgabenergebnisse, JEE-Api Abdeckungscheck
- Code Refactoring
- Termine

Fragen:

- Verschiedene Fragen bezüglich Bericht bzw. Dokumentation

Protokoll (4.12.2013, 11:00-12:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- RessourcenMonitor-Applikation
- Bericht
    - Abstract
    - Management Summary
    - Weitere Arbeiten
    - Fragen
- Video
- Folienvergleich Aufgabenergebnisse, JEE-Api Abdeckungscheck
- Code Refactoring
- Termine

Beschlüsse:

- Im Management Summary die Applikationen genauer beschreiben
- Metriken (LOC, etc.) der Applikationen in den Anhang des Tutoriums
- Voraussetzungen sowie eingesetzte Libraries pro Applikation angeben
- Installationsanleitungen zu erstellten Applikationen erstellen
- Im technischen Bericht zu den Kriterien nur eine Auflistung, keine genaue Beschreibung
- Erneute Abgabe des technischen Berichts zwecks Review

Aufgaben:

- JBoss Applikationen aus Enterprise Computing Woche 6 und 10 deployen
- Eigene Applikationen deployen und dokumentieren (IntTe Testate, etc.)
- Weiterarbeit an der Dokumentation

Sitzung vom 11.12.2013, 11:00

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- SA-Titel
- Kurzbeschreibung
- Poster
- Technischer Bericht

Protokoll (11.12.2013, 11:00-12:00)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- SA-Titel
- Kurzbeschreibung
- Poster
- Technischer Bericht

Beschlüsse:

- SA-Titel "Cloud Deployment & Architectural Refactoring Lab"
    - Klassenbeschreibung mit "CDAR" in Ordnung

Aufgaben:

- Liste von Software mit Versionsnummern an Betreuer bis 18.12.2013
- Korrekturen sämtlicher Dokumente
- Technischer Bericht sowie Tutorium abschliessen

Sitzung vom 18.12.2013, 11:00

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- SA-Abgabe
- Cloud Computing Installationen
- Poster
- Kurzfassung
- Schlussfolgerung
- Video

Protokoll (18.12.2013, 11:00-11:45)

Sitzungsteilnehmer:

- Olaf Zimmermann
- Marcel Tinner
- Daniel Zigerlig

Traktanden:

- SA-Abgabe
- Cloud Computing Installationen
- Poster
- Kurzfassung
- Schlussfolgerung
- Video

Beschlüsse:

- Inhalt der Abgabe-CD
    - Technischer Bericht (pdf/Word)
    - Tutorium (pdf/Word)
    - Technischer Bericht und Tutorium zusammen (pdf)
    - Sämtliche erstellten Applikationen
    - Werbevideo
    - Poster

Aufgaben:

- Liste von Software mit Versionsnummern an Betreuer bis 18.12.2013
- Korrekturen der abgegebenen Dokumente

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Cloud Deployment and Architectural Refactoring Lab

Studienarbeit

Tutorium

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2013

Autoren: Marcel Tinner, Daniel Zigerlig

Betreuer: Prof. Dr. Olaf Zimmermann

20. Dezember 2013

Inhaltsverzeichnis

Teil I

Tutorium / Arbeitsergebnis / Einarbeitung Cloud

1 Einleitung

Vorliegendes Tutorium informiert Anwendungsentwickler und Softwarearchitekten, bestehende oder neue Applikationen für die Cloud-Plattform anzupassen. Ausserdem stellt es Informationen betreffend Nutzung von Cloud-Infrastruktur zur Verfügung.

1.1 Fokus

Der Fokus dieser Studienarbeit liegt darin, Anwendungsentwickler und Softwarearchitekten, welche eine neue oder bestehende Applikation in einer Cloud-Plattform überführen möchten, mit Tipps zu unterstützen. Dabei fokussieren wir uns auf Angebote im Public-PaaS-Bereich. Dieser ist frei zugänglich und adressiert viele Anwender. Wir konzentrierten uns dabei auf die drei Anbieter Heroku, CloudBees und Google App Engine. Unsere Erkenntnisse sind nach weiteren Recherchen teilweise auch auf andere Cloud-Anbieter abbildbar.

1.2 Ziel

Ziel dieses Tutoriums ist es, Anwendungsentwickler sowie Softwarearchitekten in den Bereich der Public-PaaS-Provider einzuführen. Dabei sollen unsere Anwendungen einen Überblick verschaffen, worauf bei der Entwicklung von Cloud-Programmen zu achten ist.

1.3 Aufbau des Tutoriums

Neben den Beschreibungen der Applikationen zeigen wir dem Anwendungsentwickler und Softwarearchitekten auch Quellcode und Screenshots. Der im Tutorium enthaltene Quellcode zeigt die relevanten Stellen der Beispiele. Die zahlreichen Screenshots verbildlichen zudem die Ausgabe der Programme und Programmabläufe. Die Grafiken aus den Webseiten der Provider haben die Aufgabe, einen schnellen Einblick in den jeweiligen Kernbereich zu veranschaulichen.

2 Kriterienkatalog

Es ist schwer einen objektiven Vergleich der Cloud-Anbieter zu erhalten. Deshalb zeigt dieses Kapitel einen Kriterienkatalog, welcher einen messbaren Vergleich der verschiedenen Cloud-Anbieter präsentiert. Diesen Kriterienkatalog stellten wir aus Literaturrecherchen sowie Erkenntnissen und Erfahrungen während der Entwicklung der Beispielanwendungen zusammen. Die folgenden Kriterien beschreiben wichtige Eigenschafen der PaaS-Provider und zeigen Vor- sowie Nachteile. Der Katalog dient als Entscheidungshilfe bei der Wahl eines geeigneten Cloud-Anbieters.
Die einzelnen Kriterien beinhalten jeweils eine Fragestellung, eine Beurteilung pro Anbieter sowie ein Fazit. Ausserdem sind sie in ihrer Wichtigkeit eingestuft. Durch diese Einteilung sind wichtige Kriterien sofort ersichtlich.

K1: Einstiegshilfen

Da der Start in ein neues Produkt oft am meisten Schwierigkeiten darstellt, ist eine ausführliche Dokumentation wichtig.

Frage: Gibt es Einstiegs-Tutorials für die Verwendung der Cloud? Wie hilfreich sind die Anleitungen? Werden mehrere Wege gezeigt z.B. über die Console und über eine IDE?

Wichtigkeit: mittel-hoch

CloudBees: Der Provider gestattet einen guten Einstieg in sein Produkt. Sie bieten gut verständliche Schritt-für-Schritt Anleitungen, welche das Deployment einer Applikation beschreiben. Diese Anleitungen beschreiben neben dem eigenen CloudBees Software Development Kit auch die Nutzung des Eclipse-Plugins.

Google App Engine: Der Benutzer findet sich durch die Anleitungen beim Anbieter sehr schnell auf der Platform zu Recht. Auch die Anleitung, welche das Arbeiten mit Eclipse und dem Google App Engine Plugin beschreibt, ist hilfreich.

Heroku: Dieser Cloud-Anbieter stellt eine ausführliche Anleitung zur Verfügung. Teilweise müssen Tools installiert werden, die nicht von Heroku stammen, wie beispielsweise Maven [Mav131]. Auch diese wiederum verweisen auf eine Anleitung. Das Deployment und Management ist über die Konsole möglich. Zusätzlich existiert ein Eclipse-Plugin, dessen Anleitung lässt jedoch kleinere, teilweise wichtige Schritte aus.

Fazit: In diesem Punkt sind CloudBees und Google App Engine ausgereifter als Heroku. Bei sämtlichen Anbietern sind die Einstiegshilfen schnell zu finden. Jedoch sind die Anleitungen bei Heroku in einzelnen Punkten zu knapp gehalten. Im Allgemeinen ist inklusive dem Einrichten des jeweiligen SDK's bei allen Anbietern ein „Hello World" innerhalb von 25 min möglich.

K2: Dokumentation

Zu einem guten Cloud-Anbieter gehört auch eine hilfreiche Entwickler-Dokumentation welche die Nutzung des Dienstes beschreibt.

Frage*:* Wie umfangreich beschreibt der Anbieter wie man sein Produkt nutzen kann? Wie hilfreich sind diese Informationen?

Wichtigkeit*:* mittel-hoch

CloudBees: Alle Überbegriffe sind zentral auf einer Seite aufgelistet. Über die Unterpunkte kann man gezielt auf die entsprechenden Artikel zugreifen. Die einzelnen Punkte enthalten hilfreiche Hinweise und Beispiele. Bei einzelnen Artikeln verweist CloudBees jedoch schnell auf den eigenen Support. Die ganze Dokumentation ist übersichtlich und verständlich aufgebaut.

Google App Engine: Die Dokumentation von Google ist gut gehalten. Google bietet sehr viele Informationen an. Alle Dokumentationen sind informativ, dadurch bleiben keine Fragen offen. Unter diesem grossen Informationsangebot leidet jedoch die Übersichtlichkeit.

Heroku: Der Anbieter stellt eine gute Dokumentation über sein Produkt zur Verfügung. Auch die Fehlerdokumentation ist nicht zu kurz gekommen. Unterhalb der Anleitungen sind zudem mögliche Fehler mit den entsprechenden Lösungsvorschlägen aufgelistet. Ebenfalls eine Liste von Fehlercodes ist vorhanden. Die Problembehebung ist nach unseren Recherchen jedoch nicht beschrieben. Alternativ zur Dokumentation steht ein Support zur Verfügung.

Fazit: Im Ganzen sind alle Dokumentationen der drei Konkurrenten gleich gut. Alle Dokumentationen sind hilfreich und informativ.

K3: Sprachen und Frameworks

Da oft schon die Programmiersprache vorausgesetzt ist, ist die Auswahl der unterstützen Programmiersprachen in der Cloud ein wichtiges Kriterium.

Frage: In welchen Sprachen kann entwickelt werden? Welche Frameworks werden unterstützt? Gibt es Einschränkungen oder Erweiterungen?

Wichtigkeit: ausschlaggebend

CloudBees: CloudBees kann mit Java angesprochen werden. Auch das Web Applikation Framework Rails wird unterstützt. Eine Erweiterung bietet JRuby, womit jeglicher Ruby-Code in Java übersetzt werden kann.

Google App Engine: Google App Engine hält sich in der Mitte mit den unterstützen Sprachen.

Programmier- Skriptsprache	Frameworks
Go (Experimental)	Django
Java	Spring MVC, Struts 2
PHP (Preview)	
Python	Django
Scala	

Tabelle 1 - Sprachen Google App Engine

Heroku: Heroku ist flexibel in Bezug auf die Programmiersprache.

Programmier- Skriptsprache	Frameworks
Clojure	
Java	Spring, Play
Node.js	
Python	Django
Ruby	Rails
Scala	

Tabelle 2 - Sprachen Heroku

Fazit: Bekannte Programmiersprachen wie Java unterstützen alle Cloud-Anbieter. Google App Engine bietet noch weitere Programmiersprachen auf Testbasis an. Auf Heroku stehen neben Java auch einigen anderen Programmiersprachen zur Auswahl.

K4: Deployment in die Cloud

Sobald ein Projekt im Gange ist, ist das Ziel die Applikation möglichst schnell zu deployen. Deswegen muss der Vorgang für den Cloud-Anwender einfach und effizient sein. Zudem sollte das Deployen nicht nur über die Konsole möglich sein.

Frage: Welche Möglichkeiten stehen zur Verfügung um Applikationen auf die Cloud zu deployen? Worauf muss geachtet werden?

Wichtigkeit*:* mittel

CloudBees: Bei CloudBees besteht die Webseite, über die Konsole sowie auch über Eclipse ein Projekt zu deployen. Über die Webseite ist ausschliesslich das Deployment von War-Files möglich. Nutzt man die Konsole können zusätzlich auch direkt die Projekte eingecheckt werden. Mit Eclipse wird dies über das Git-Repository von CloudBees gelöst.

Google App Engine: Das Deployment über die Google App Engine Webseite wird nicht unterstützt. Für das Deployen mit Eclipse steht über das Plugin eine zusätzliche Auswahl zur Verfügung. Über diese Auswahl kann danach das Google App Engine-Projekt deployt werden. Das direkte Deployen eines War-Files wird über ein SDK unterstützt.

Heroku: Bei diesem Cloud-Provider stehen mehrere Varianten zur Auswahl. Es besteht die Möglichkeit das War-File über Eclipse hochzuladen. Die Datei muss jedoch bestimmte Bedingungen erfüllen, welche nicht standartmässig gegeben sind. Eine andere Möglichkeit ist es, das Maven-Projekt über ein Commit von Eclipse hochzuladen. Auch über die Konsole ist es möglich, ein Projekt zu deployen.

Fazit: CloudBees und Heroku bieten beinahe dieselben Möglichkeiten, ein File oder Projekt zu deployen. Ein Vorteil ist jedoch, dass bei CloudBees ohne Plug-Ins und weiteren Tools der Nutzer ein War-File hochladen kann. Ein Nachteil von Heroku sind die Bedingungen die für das War-File gelten. Google App Engine bietet das deployen direkt über das Plug-In und die Konsole an.

K5: Anpassungsfähigkeit der Serverinstanz

Entscheidet sich ein Anwendungsentwickler spezielle Frameworks und Libraries in seiner Applikation zu benutzen, sind unter Umständen die eingerichteten Serverinstanzen des PaaS-Anbieter nicht geeignet.

Frage: Existieren Möglichkeiten die Umgebung gemäss eigenen Vorstellungen zu konfigurieren?

Wichtigkeit: Mittel

Heroku: Dieser Anbieter bietet sogenannte Buildpacks. Ein Buildpack definiert, wie die Serverinstanz konfiguriert werden soll. Vordefiniert durch Heroku existieren folgende Buildpacks:

- Ruby
- Node.js,
- Clojure
- Python
- Java
- Gradle
- Grails
- Scala
- Play

Die genannten Buildpacks sind konfigurierbar. Hierzu finden sich im Internet viele Open-Source “Custom-Buildpacks” [Bui13] [Bui131].

Google App Engine: Soweit unsere Recherchen gezeigt haben bietet dieser Anbieter keine Möglichkeiten an, die Konfiguration anzupassen.

CloudBees: Vordefinierte ClickStarts helfen dem Applikationsentwickler, seine gewünschte Konfiguration zu laden. Auf einem öffentlichen GitHub-Repository finden sich viele freiverfügbare ClickStarts. Diese lassen sich nach Belieben konfigurieren und anpassen [Cli13] [Com13].

Fazit: Heroku sowie CloudBees ermöglichen es dem Entwickler, die Serverinstanz nach Wunsch zu verändern. CloudBees bietet gegen Heroku eine grössere Menge an vordefinierten ClickStarts bzw. Buildpacks an. Beide Anbieter haben eine grosse Community, welche bei Problemen gewillt ist, zu helfen.

K6: Handhabung

Beinahe keine Anwendung ist für jeden Anwender nach Mass. Allerdings gibt es in vielen Anwendungen Abläufe, welche man besser umsetzen könnte.

Frage: Gibt es Programme, Einstellungen, Abläufe oder Ähnliches, die beim Cloud-Anbieter positiv oder negativ auffallen, bzw. die Handhabung verschlechtern? Was hätte besser gelöst werden können?

Wichtigkeit: gering

CloudBees: Vor dem Deployment über das SDK wird auf Updates überprüft. Falls welche gefunden werden, kann der Vorgang eine gewisse Zeit in Anspurch nehmen. CloudBees ist der einzige Anbieter, welcher das Deployment über das Webinterface anbietet.

Google App Engine: Die Dokumentation ist sehr ausführlich, dabei kann ein Neueinsteiger schnell die Übersicht verlieren.

Heroku: Das Löschen eines Projektes bei Heroku über die Internetseite ist umständlich. Um das Projekt zu löschen, muss zuerst auf dasjenige Projekt geklickt werden, danach muss jedoch trotzdem noch der Name des zu löschenden Projektes angegeben werden. Dies kann wiederum auch als Sicherheitsfeature angesehen werden.

Fazit: Jeder Anbieter hat seine Vor- und Nachteile. Bei keinem Provider haben wir massgebende Probleme bei der Handhabung festgestellt.

K7: On-demand

Der Vorteil von Clouds ist unter anderem, dass die Infrastruktur nicht aufgesetzt werden muss, sondern direkt zur Verfügung steht.

Frage: Wie schnell steht der Service bereit? Werden Installationen auf dem eigenen Rechner vorausgesetzt?

Wichtigkeit: hoch

CloudBees: Dieser Anbieter stellt den Service nach einer kurzen Registration direkt zur Verfügung. Das Erstellen von neuen Projekten auf der Cloud erfolgt über die Webplattform oder über das zu installierende CloudBees-SDK und steht danach direkt zur Verfügung.

Google App Engine: Nach einer Registration kann bei Google App Engine sofort gestartet werden. Allerdings setzt der Anbieter ein Plug-In für das Entwickeln von Java-Applikationen voraus, alternativ steht auch ein SDK zur Verfügung.

Heroku: Bei diesem Provider steht nach wenigen Klicks und einer Registration der Service bereit. Über die Webplattform oder das Eclipse Plug-In von Heroku kann man neue Cloud-Projekte erstellen.

Fazit: In diesem Punkt sind alle Anbieter auf gleichem Niveau. Die Services stehen schnell und betriebsbereit zur Verfügung.

K8: Self-Service

Da der Kunde den Server nicht mehr selber aufsetzen und installieren muss, soll ihm doch noch die Möglichkeit offen gehalten werden, weitere Services zu beantragen.

Frage: Was kann genutzt werden? Wie kann dieser Service gelöst werden?

Wichtigkeit: mittel-hoch

CloudBees: Über die Webseite von CloudBees können die einzelnen Services aufgeschaltet werden. Ausserdem stellt CloudBees mehrere Plug-Ins auf der Webseite zur Integration bereit.

Google App Engine: Bei diesem Anbieter wird alles über das Web-Interface gelöst. Dort werden die Projekte verwaltet und neue hinzugefügt.

Heroku: Bei diesem Provider ist es möglich alle Plug-Ins über die Konsole aufzuschalten. Neben der Konsole steht beinahe die gleiche Funktionalität auf der Webseite zur Auswahl.

Fazit: Heroku bietet hier zusätzlich die Möglichkeit alles über die Konsole aufzuschalten. Ansonsten bieten alle drei Konkurrenten dasselbe Angebot.

K9: Scalable

Da die Leistung einer einzelnen Ressource auf welcher die Applikation läuft oft nicht ausreicht, sollten die Ressourcen skalierbar sein.

Frage: Wie kann skaliert werden? Wird automatisch skaliert? Was kann skaliert werden?

Wichtigkeit: hoch

CloudBees: Bei CloudBees können die einzelnen Services wie Datenbank, Applikation sowie Jenkins um eine bis zwei Stufen aufgerüstet werden. In der Gratisversion sind die zugeteilten Ressourcen für die Applikation vordefiniert. Die Ressourcen für die Applikationen in den kostenpflichtigen Versionen werden hingegen automatisch skaliert. Ebenso ist es möglich, die Plug-Ins aufzurüsten.

Google App Engine: Google App Engine skaliert die Anzahl der Prozesse automatisch. Die einzige Einstellung die vorgenommen werden kann, ist der einzelne Prozessspeicher sowie dessen Rechenleistung.

Heroku: Bei diesem Anbieter wird über die Dynos skaliert. Das System mit den Dynos ist ähnlich, jedoch skalieren sie nicht automatisch. Die Dynos können vertikal(scale up) wie auch horizontal(scale out) skaliert werden [Dyn131].

Fazit: Bei allen Cloudanbietern sind Skalierungen möglich. Wer allerdings die Skalierung automatisch reguliert haben will, sollte eher CloudBees bzw. Google App Engine in Betracht ziehen.

K10: Measurable

Falls Dienstleistungen genutzt werden, welche Kosten, sollte eine Übersicht über die Kostenaufstellung vorhanden sein.

Frage: Ist eine Kostenübersicht verfügbar? Kann daraus entnommen werden, wann was genutzt wurde?

Wichtigkeit: mittel

CloudBees: Bei CloudBees besteht z.B. die Möglichkeit über den „New Relic“ eine Übersicht über die genutzten Ressourcen zu erhalten [New13]. Der zu betrachtende Zeitabschnitt kann angepasst werden und dadurch ist die Dauer und die Menge der genutzten Dienstleistung ausfindig zu machen.

Google App Engine: Bei Google App Engine ist der Monitor bereits integriert und bietet etwa die gleiche Funktionalität wie CloudBees an.

Heroku: Die bietet viele Plug-Ins an, darunter u.a. auch „New Relic“. Es gibt auch die Gratisversion von „New Relic“, allerding ist das Hinterlegen von Kreditkartendaten notwendig. Auch die alternativen Plug-Ins bieten Monitorfunktionen an, jedoch auch nur unter der Angabe von Kreditkartendaten.

Fazit: Wer ohne seine Kreditkartendaten den Cloud-Service nutzen will, steht bei CloudBees und Google App Engine mehr zur Verfügung. Für wen dies unwesentlich ist, der kann bei Heroku mehrere Plugins testen, welche die Monitorfunktionen anbieten.

K11: Domain Verwaltung

Für die effektive Nutzung einer Applikation möchte der Anwender meist seine eigene Domain einsetzen. Damit kann er die Cloud-Applikation z.B. auf www.myurl.ch/AnyApplication aufrufen.

Frage*:* Welche Möglichkeiten für das Setzen der eigenen Domain werden angeboten?

Wichtigkeit*:* hoch

Bei sämtlichen Cloud-Anbietern besteht die Möglichkeit, über das Web-Interface in den Einstellungen der gewünschten Applikation eigene Domains hinzuzufügen.

Dies sind jedoch nur die Domains, auf welche der Server reagiert, beinhaltet jedoch in der Regel kein DNS Hosting.

A-Records können problematisch werden, da sie aufgrund der Einschränkung von DNS auf eine fixe IP zeigen, diese sich wiederum bei Cloud-Anbietern wechseln kann.

K12: Nutzung von Sockets

Viele Applikationen nutzen Sockets als Kommunikationsmittel. Diese können als Server- sowie auch als Client agieren.

Frage: Ermöglichen die Paas-Provider das Binding von Ports? Besteht die Möglichkeit eine Verbindung via Socket herzustellen?

Wichtigkeit: niedrig

CloudBees und Heroku: Diese Provider ermöglichen das Hosting eines Server-Sockets in der Applikation. Als Übertragungsprotokoll muss jedoch HTTP verwendet werden. Ein praktischer Test ist im Kapitel „8 Nutzung von TCP Sockets in der Cloud“ beschrieben.

Google App Engine: Im Rahmen unserer Studienarbeit liess sich bei diesem Anbieter kein Socket realisieren.

Fazit: Socketapplikationen in Cloud-Plattformen unterscheiden sich im Nutzverhalten massgeblich von traditionellen IT-Infrastrukturen. Der Applikationsentwickler ist durch die Nutzung des HTTP-Protokolls und nur eines Ports massgeblich eingeschränkt.

K13: Selbstauskunft

Eine Applikation beziehungsweise der Programmierer interessiert sich für die darunterliegenden Systemeigenschaften. Zu diesen gehören unter Umständen beispielsweise Festplattengrössen, Arbeitsspeicher aber auch Netzwerkinformationen sowie Umgebungsvariablen.

Frage: Welche Informationen geben die PaaS-Provider über das darunterliegende System bekannt?

Wichtigkeit: hoch

CloudBees und Heroku: Diese Provider ermöglichen es folgende Informationen, programmatisch durch ein Java Programm auszulesen:

- Hardware Informationen
- Netzwerk Informationen
- System Eigenschaften
- Umgebungsvariablen

Genauere Details zu den erfragten Informationen sind im Kapitel „6 Diagnoseanwendung Selbstauskunft“zu finden. Diese beinhalten auch den dynamischen Port, welcher jeder Applikation zugewiesen wird. Benötigt wird er im Kapitel „8 Nutzung von TCP Sockets in der Cloud“ für das Erstellen eines Sockets.

Google App Engine: Dieser Anbieter stellt nur eine eingeschränkte Java Funktionalität zur Verfügung. Deswegen konnten die Netzwerk Informationen nicht ausgelesen werden.

Fazit: Die ausgelesenen Informationen erweisen sich als sehr interessant. Teilweise sind sie auch notwendig für die Realisierung von Programmfunktionalitäten wie zum Beispiel einem Socket. Es ist erstaunlich, wie viele Informationen ausgelesen werden können und wie sie sich in kurzer Zeit bereits wieder verändert haben.

K14: Datenbanksysteme

Datenbanksysteme werden nicht nur in traditionellen IT-Infrastrukturen verwendet. Auch im Bereich der PaaS-Programmierung finden sich verschiedene Datenbanksysteme.

Frage: Welche Datenbanksysteme bietet der Anbieter an?

Wichtigkeit: mittel-hoch

CloudBees: Neben MySQL Datenbanken bietet dieser Anbieter auch den Zugriff auf NoSQL und Database-As-A-Service (DBaaS) Instanzen an. Hierzu zählen MongoHQ, Cloudant, FoxWeave sowie RabbitMQ.

Google App Engine: Eigene Implementierungen wie Google Cloud SQL sowie Datastore sind bei diesem Anbieter zu finden. Bei Datastore handelt es sich um eine Objektdatenbank. Bei Google Cloud SQL handelt es sich um ein ähnliches Datenbanksystem wie MySQL.

Heroku: *Dieser Anbieter bietet 20 verschiedene Datenbankservices an. Darunter von relationalen Datenbanksystemen wie MySQL bis NoSQL Services wie MongoHQ oder MongoLab* [Dat134].

Fazit: Die grösste Auswahl an unterschiedlichen Datenbanksystemen findet man beim PaaS-Anbieter Heroku. Sämtliche Anbieter ermöglichen es dem Anwendungsentwickler, seine Datenbankabfragen nach einem relationalen oder aber auch nach einem dokumentbasierten Ansatz zu programmieren.

2.1 The Twelve-Factor App

Heutzutage wird Software oft als Service beziehungsweise sogenannten Web-Applikationen oder SaaS an den Benutzer gebracht. Die Webseite "The Twelve-Factor App" beschreibt eine Programmiermethodik zur Erstellung von SaaS-Applikationen, welche nachfolgende Punkte erfüllen [The13].

- Nutzung von deklarativem Format für Installations-Automation.
- Klare Darstellung von Nutzung des darunterliegenden Systems. Dies führt zu maximaler Portabilität zwischen verschiedenen Ausführungsumgebungen.
- Sind geeignet für das Deployment in moderne Cloud-Plattformen.
- Halten den Unterschied zwischen Entwicklungs- und Ausführungsumgebung möglichst klein.
- Sind ohne signifikanten Änderungen skalierbar.

Die Webseite richtet sich in erster Linie an Applikations-Entwickler, welche SaaS-Applikationen programmieren.

2.2 Erkenntnis

In diesem Kapitel sind grundlegende Kriterien von PaaS-Clouds aufgelistet. Es ist wichtig seine Aufgabe bereits im Vorhin zu kennen. Daraus sind die für diese Aufgabe ungeeigneteren Cloud-Providern zu eliminieren. Nicht alle Cloud-Anbieter stellen die gleichen Funktionalitäten zur Verfügung. Jede Cloud hat bestimmte Einschränkungen wie zum Beispiel, dass die gewünschte Programmiersprache nicht unterstützt wird.

3 Abrechnungsmodelle

Im Rahmen dieser Studienarbeit begegneten uns zwei Abrechnungsmodelle. Namentlich die Abrechnung nach Laufzeit sowie nach genutzten Ressourcen.

3.1 Abrechnung nach Laufzeit

Die Cloudanbieter Heroku und CloudBees verrechnen dem Kunden die Nutzung von dedizierten Instanzen. Hierbei unterscheiden sich Services wie auch die Applikations-Instanzen. Läuft eine Applikation in einer Instanz, bezahlt der Kunde folglich die Laufzeit der Instanz am Ende des Monats. Zusätzlich kommt er für die Nutzung von Services, wie beispielsweise einer MongoDB Datenbank, auf. Wie in der nachfolgenden Abbildung von Heroku ersichtlich ist, kann sich der Kunde für verschiedene Grössen der Datenbank entscheiden. Er bezahlt dann monatlich je nach Grösse einen fixen Betrag. Daher ist die Menge der Abfragen auf die Datenbank, irrelevant in Bezug auf die effektiven Kosten.

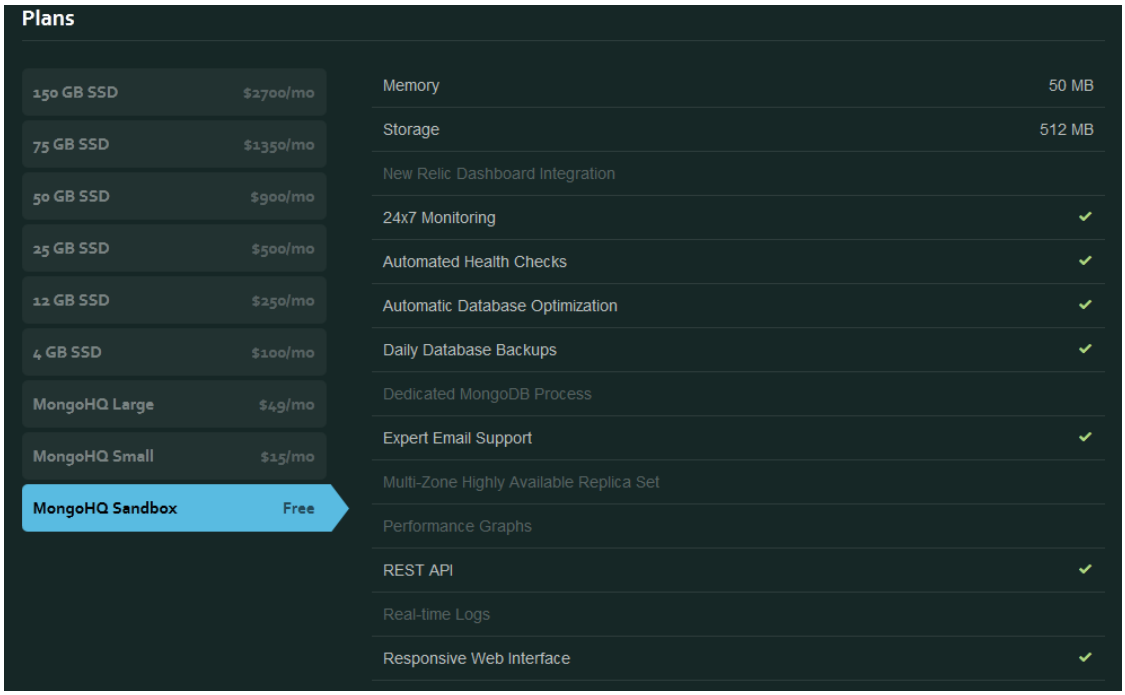


Abbildung 1 - Heroku, MongoHQ Kostenlevel

Für den Kunden der Paas-Provider mit diesem Kostenmodell sind somit die Kosten bereits im Voraus klar kalkulierbar. Es empfiehlt sich, die Applikationen in Bezug auf Rechenlaufzeit und Datenspeicher effizient zu programmieren. Dadurch führen auch viele Aufrufe und Abfragen auf Services ohne Skalierung zu keinen Mehrkosten. Demzufolge unterscheidet sich dieses Modell zum Nachfolgenden, welches in dieser Hinsicht kostenintensiver ist.

3.2 Abrechnung nach genutzten Ressourcen

Google App Engine verrechnet dem Benutzer die Nutzung von Ressourcen. Der Benutzer hat die Möglichkeit, sämtliche Ressourcen des Anbieters zu nutzen. Überschreitet sich jedoch der Gebrauch eine Tageslimite (Quota), muss der Kunde für weitere Operationen bezahlen.

Nachfolgend ein Auszug des Preismodelles von Google App Engine [Pri131].

Hosting	Quota	Kosten nach überschreiten der Quota
Instanzen auf Abruf	28 Stunden	$0.08 / Stunde
Reservierte Instanzen	keine	$0.05 / Stunde
Datastore Speicherplatz	1 GB	$0.12 / GB
Ausgehende Datenmenge	1 GB	$0.12 / GB
Eingehende Datenmenge	1 GB	Kostenlos
APIs		
Datastore API	50'000 read/write/small-Operationen	$0.09 / 100'000 write Operationen $0.06 / 100'000 read Operationen $0.01 / 100'000 small Operationen

Tabelle 3 - Kostenübersicht Google App Engine

3.3 Gegenrechnung der genutzten Ressourcen

Nun kann dem Cloud-Anbieter bei der Abrechnung vertraut werden, oder man loggt die Nutzung der Ressourcen parallel zum Anbieter selbst. Folgende Beispielapplikation loggt kostenpflichtige Aktionen von Google App Engine.

Es gibt diverse Ressourcen welche Google App Engine anbietet und somit auch geloggt werden müssten, jedoch sind nicht alle Logeinträge mit gleicher Schwierigkeit zu realisieren. Datenbankvorgänge sind zum Beispiel einfacher zu loggen als die benötigten Instanzen der Applikation (siehe auch „4.1 Ressourcen Zuteilung“). Deshalb fokusieren wir uns bei dieser Applikation auf das Loggen von Datastore-Abfragen. Diese sind trivial und für erste Gegenrechnungen gut geeignet.

Die Java Applikation zeichnet alle Lese- und Schreiboperationen der Datenbank auf. Die Applikation bezieht sich auf einige API Call Operationen, welche folgende Kosten (read, writes) mit sich tragen:

API Call	Datastore Operations
Entity Get(per entity)	1 read
New Entity Put (per entity)	2 writes + 2 writes per indexed property value
Existing Entity Put (per entity)	1 write + 4 wites per modified indexed property value
Entity Delete (per entity)	2 writes + 2 writes per indexed property value

Tabelle 4 - API Calls

Um die Applikation einfach und generisch zu halten, verzichten wir auf das Logging „composite index value“.
Nachfolgend ein Beispiel der genutzten Ressourcen bezogen auf Google Datastore.

Resource	Used	Free	Billable	Charge
Datastore Writes $0.90/Million Ops	0.01	0.05	0.00	$0.00
Datastore Reads $0.60/Million Ops	0.01	0.05	0.00	$0.00

Abbildung 2 - Google App Engine, Datastore Kostenrechnung

3.3.1 Endresultat

Die Applikation besteht aus einem Web Applikation Project von Google. Dieses beinhaltet zwei Servlets. Das eine Servlet ist für die Darstellung der Logs zuständig, das Andere für die Erstellung der Logs.
Nach dem Deployment auf Google ergibt sich nachfolgende Darstellung der Servlets im Browser.

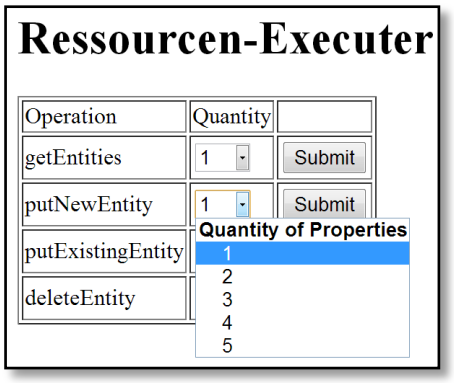


Abbildung 3 - Datastore Logs

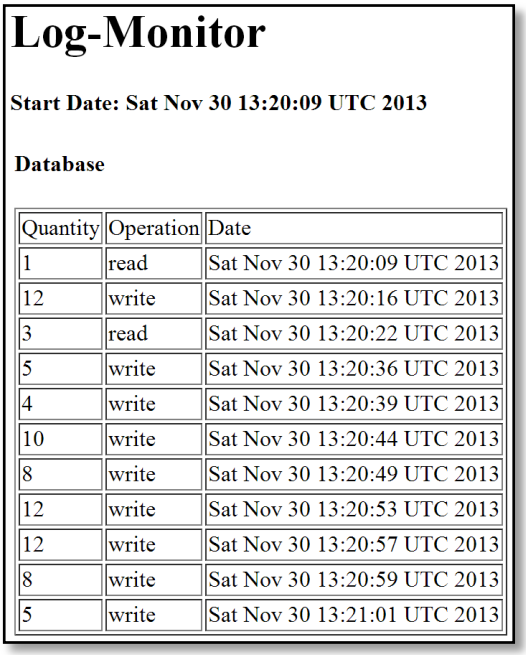


Abbildung 4 - Datastore Log-Ersteller

3.3.2 Leistungsnachweis

Google bietet ein Leistungsnachweis an. Dies ermöglicht einen Vergleich mit dem aufgezeichneten Log der Beispielapplikation.

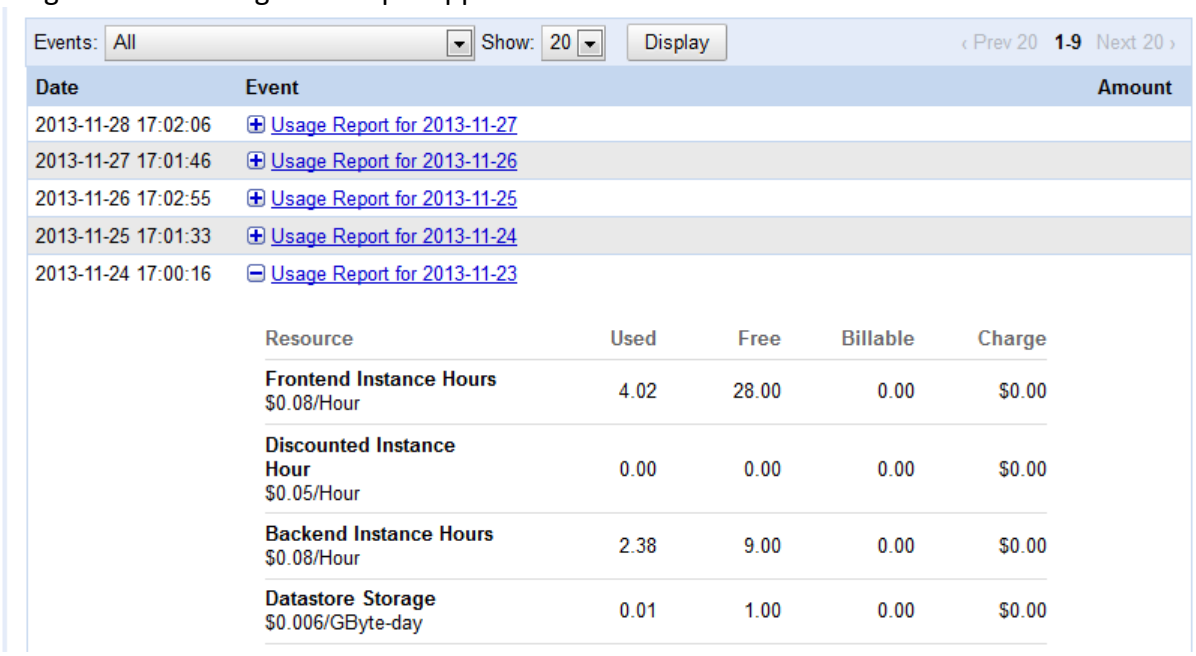


Abbildung 5 - Leistungsnachweis Google App Engine

3.3.3 Ausgabemöglichkeiten

Der Business Logic Layer wurde so konzipiert, dass man zwischen einer HTML und XML Ausgabe wählen kann.

3.3.3.1 Beispiel Ausgabe HTML

Die HTML Ausgabe erzeugt eine Tabelle mit allen Logs.

```
1  <table  border="1">
2    <tr><td>Quantity</td><td>Operation</td><td>Date</td></tr>
3    <tr> <td>10</td> <td>write</td> <td>Mon Dec 02 19:10:12 CET 2013</td> </tr>
4    <tr> <td>10</td> <td>write</td> <td>Mon Dec 02 19:10:12 CET 2013</td> </tr>
5    <tr> <td>10</td> <td>write</td> <td>Mon Dec 02 19:10:12 CET 2013</td> </tr>
6  </table>
```

Quellcode 1 - SelfInformation HTML Output

3.3.3.2 Beispiel Ausgabe XML

Der erzeugte XML Code wurde mit dem W3C-XML-Validator als korrekt geprüft. Anbei ein XML-Auszug eines Log-Beispiels [Val13].

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <CDAR_Logs>
3    <Log>
4      <operation>write</operation>
5      <quantity>10</quantity>
6      <time>Mon Dec 02 19:10:12 CET 2013</time>
7    </Log>
8    <Log>
9      <operation>write</operation>
10     <quantity>10</quantity>
11     <time>Mon Dec 02 19:10:22 CET 2013</time>
12   </Log>
13   <Log>
14     <operation>read</operation>
15     <quantity>1</quantity>
16     <time>Mon Dec 02 19:10:34 CET 2013</time>
17   </Log>
18 </CDAR_Logs>
```

Quellcode 2 - RessourcenMonitor XML Output

3.4 Erkenntnis

Anhand des Logs der Beispielapplikation beziehungsweise mittels des Leistungsnachweises des Providers, zeigt sich die Nutzung der Ressourcen. Daraus lassen sich die Kosten errechnen respektive anzeigen. Wichtig ist in beiden Kostenmodellen eine effiziente Programmierung. Da im zweiten Modell für jede einzelne Ressource abgerechnet wird, kann eine unstrukturierte und ineffizient entwickelte Applikation schnell teuer werden. Der Entwickler sollte demnach die vorhandenen Ressourcen in verantwortungsvoller Weise nutzen.

4 Cloudspezifische Eigenschaften

In diesem Kapitel sind spezifische Eigenschaften der Cloud-Infrastruktur erklärt. Falls nicht anders gekennzeichnet, beziehen sich diese Eigenschaften auf sämtliche Cloud-Anbieter.

4.1 Ressourcen Zuteilung

Bei Heroku und CloudBees wird einer Applikation eine gewisse Menge von Ressourcen zugeteilt. Möchte man die Applikation nun skalieren, kann man weitere Kontingente dazu schalten. Schlussendlich bezahlt mal die Anzahl der Mengen.
Bei Google App Engine hingegen wird automatisch skaliert. Man hat keinen dedizierten Ressourcenraum. Die Bezahlung erfolgt anhand der Nutzung. Diese beinhaltet die Anzahl der Datenbankabfragen, wie auch die zeitliche Nutzung der Dienste.

4.1.1 CloudBees – App-Cell

Bei CloudBees bezahlt man pro sogenannten App-Cell je Stunde. Eine App-Cell besteht aus 128 MB Arbeitsspeicher sowie einem Achtel vom Server Core. Die Anzahl der benötigten App-Cells kann man berechnen, indem man die Grösse der Applikation in 128MB Blöcke aufteilt. Entscheidet man sich für die kostenpflichtige Version von CloudBees, besteht die Grösse einer App-Cell aus bis zu 4 Gb.
Hat man beispielsweise eine Applikation welche 200MB Speicher braucht, wird für die Ausführung zwei App-Cells benötigt [Pri13].

4.1.2 Google App Engine – Quotas

Bei Google App Engine hat man keine dedizierten Ressourcen. Der Ersteller der Applikation kann sich selber Safety Quotas in Form von täglichen sowie minütlichen Begrenzungen setzen. Diese definieren, wie viele Ressourcen die Applikation maximal in diesem Zeitraum nutzen darf. Der Applikation werden dann bis zu dieser festgesetzten Limite automatisch die benötigten Ressourcen zugeteilt. Für eine genauere Kostenübersicht siehe Kapitel „3 Abrechnung“.

4.1.3 Heroku – Dyno

Bei Heroku können 1x-Dynos und 2x-Dynos beantragt werden. Bei einem 1x-Dyno wird 512MB RAM und eine CPU, bei einem 2x-Dyno das Doppelte zur Verfügung gestellt (1024MB) [Dyn13] [Dyn131]. Falls ein Dyno nicht ausreicht, können zusätzliche 1x bzw. 2x-Dynos aktiviert werden. Der Benutzer kann entweder über die Konsole oder über die Website von Heroku die Anzahl der Dynos regulieren.

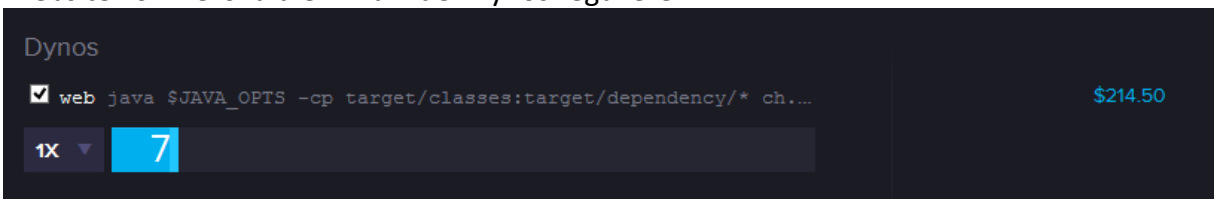


Abbildung 6 - Webseite Heroku - Dyno regulieren

Bei einem Dyno kann der Arbeitsspeicher bis zum Dreifachen überzogen werden. Allerdings verschlechtert sich die Leistung bei Überschreitung des Wertes. Ein Dyno, welcher das Dreifache überschreitet, wird mit einem Errorcode beendet und danach neugestartet.

Folgende weitere Eigenschaften weisen Dynos auf [Dyn131].

- Horizontale Skalierbarkeit: Die Anzahl der Dynos, die einer Anwendung zugeordnet sind, können jederzeit erhöht oder gesenkt werden.
- Routing: Die Router verfolgen den Standort aller laufenden Dynos und leiten den HTTP-Traffic zu ihnen weiter.
- Dyno-Management: Alle laufenden Dynos werden überwacht. Dabei wird sichergestellt, dass sie weiterhin aktiv bleiben. Falls ein Dyno abstürzt, wird er entfernt und durch einen neuen Dyno ersetzt.
- Verteilung und Redundanz: Eine Applikation mit mehr als nur einem Dyno läuft unter Umständen an unterschiedlichen physischen Standorten. Damit wird sichergestellt, falls ein Dyno ausfällt, die anderen Dynos weiterlaufen und somit die Applikation aktiv bleibt.

4.2 Lebenszyklus Cloud Applikation

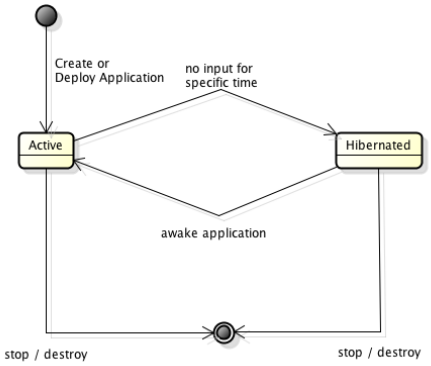


Abbildung 7 - Lebenszyklus Cloud Applikation

Nach dem Deployment einer Cloud-Applikation befindet sich die Applikation im *Active-Modus* [War13] [FAQ13] [App13]. Ein Aufruf dieser Applikation wird dann sofort behandelt.

Wird eine Applikation über einen gewissen Zeitraum nicht mehr benutzt, geht sie automatisch in einen *Hibernated*- bzw. *Sleep-Modus* (Begriff bei Heroku: *Asleep*).

Wird sie aufgerufen solange sie sich im Sleep-Modus befindet, muss sie vor dem Gebrauch erst aufgeweckt werden. Dies dauert je nach Provider einige Sekunden.
Diese Eigenschaft ist vor allem bei kostenlosem Gebrauch der Dienste üblich. Der Anbieter kann die Ressourcen zwischenzeitlich für andere Applikationen nutzen. Entscheidet sich der Benutzer für einen kostenpflichtigen Dienst, ist die Deaktivierung des *Sleep-Modus* möglich.

Die Idle-Zeiten bis zum Umschalten in den Sleep-Modus unterscheiden sich je nach Anbieter, genaue Zahlen konnten für CloudBees und Heroku gefunden werden:

- CloudBees: 2 Stunden
- Heroku: 1 Stunde

4.2.1 Alternativen

Um dem Sleep-Modus zu entgehen und die Applikation ständig aktiv zu halten, existieren folgende Möglichkeiten:

- Nutzung eines kostenpflichtigen Dienstes.
- Abfragen ob die Applikation bald in den Sleep-Modus wechselt [Int131].
- Gelegentliches Speichern von Werten und Logs.

4.3 Domainname

Möchte der Entwickler seinen eigenen Domainnamen für die eigene Cloud Applikation nutzen, geschieht die Einrichtung grundsätzlich identisch wie bei einem normalen Webhoster. Als erstes muss dem Cloud-Anbieter die gewünschte Domain mitgeteilt werden, damit dieser die Weiterleitung von diesem Domainnamen erlaubt. Danach kann ein CNAME- beziehungsweise ein A-Record erstellt werden.

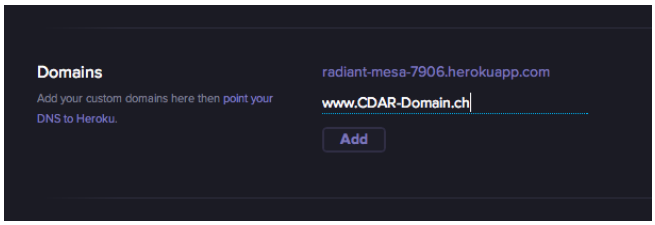


Abbildung 8 - Domain hinzufügen Heroku

4.3.1 CNAME Record

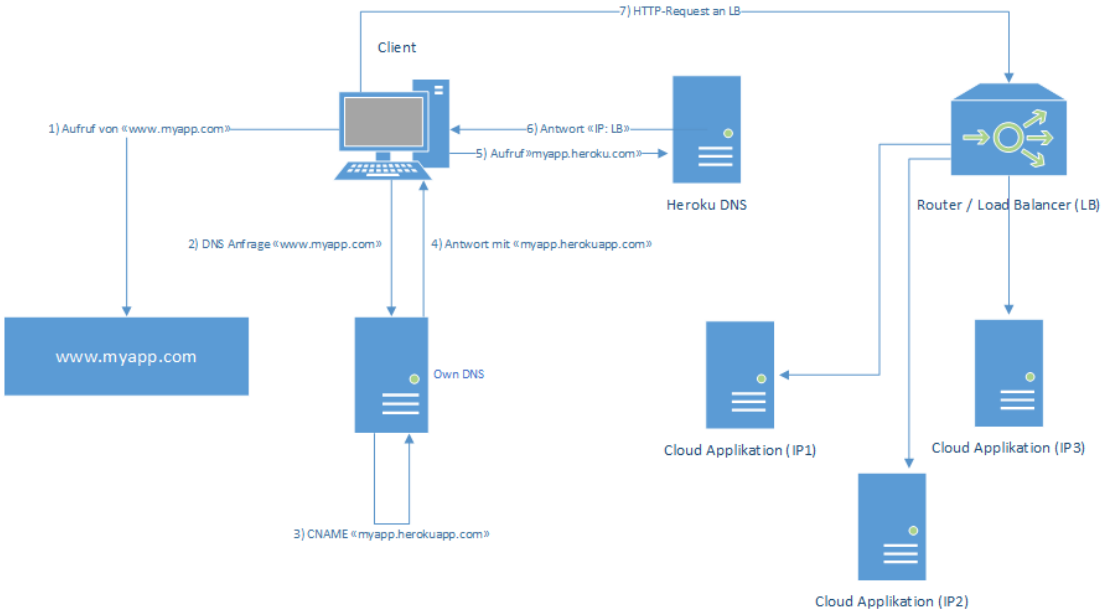


Abbildung 9 - CNAME Record

Ein CNAME wird beim DNS-Provider hinzugefügt. Der CNAME zeigt in diesem Fall auf die URI der eigenen Applikation. Nach dem Updaten des DNS-Records dauert es eine Weile, bis der DNS-Eintrag propagiert wurde. Dies dauert im Normalfall einige Stunden, kann aber unter Umständen bis zu einem ganzen Tag dauern.

Im Bereich der Cloud-Applikationen kann es im Vergleich zu normalen Hosting-Angeboten eher zu einem Wechsel des Servers kommen. Die Wahrscheinlichkeit, dass die effektive IP einer Applikation wechselt, ist somit höher. Im Falle der Nutzung eines CNAME's ist dies jedoch irrelevant, da der Cloud-Anbieter selbst für die korrekte Weiterleitung seiner Subdomain zur Applikation zuständig ist.

4.3.2 A-Record

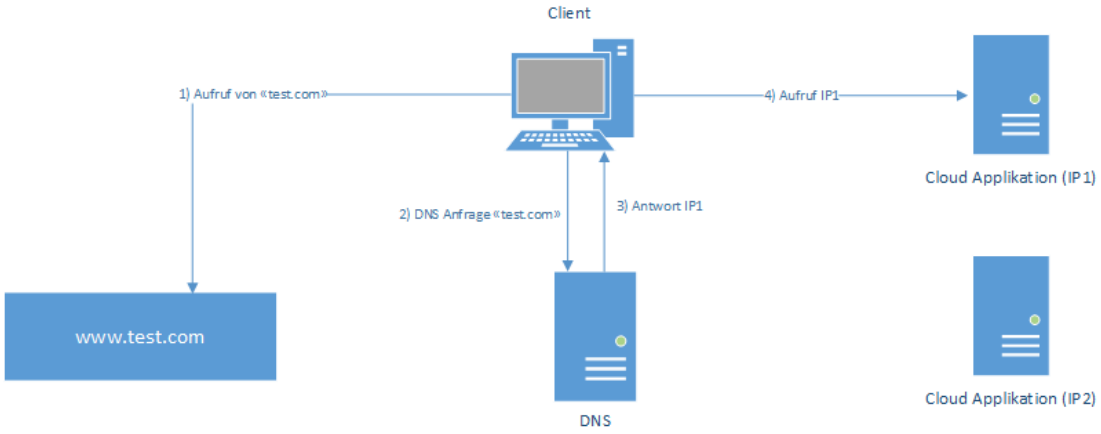


Abbildung 10 - A Reord

Eine Limitierung von DNS ist, dass man CNAMES nur für das Zeigen auf Subdomains nutzen kann. Möchte man jedoch zum Beispiel „test.com“ auf die eigene App weiterleiten, kann man dies nicht mit einem CNAME bewerkstelligen, sondern muss einen A-Record nutzen.

Für die Nutzung im Cloud-Umfeld bestehen für das Erstellen von A-Records drei Möglichkeiten. Diese werden im Folgenden erklärt.

4.3.3 Manueller A-Recordeintrag

A-Record auf die IP setzen, auf der die Applikation momentan läuft. In diesem Fall ist der Administrator der Applikation jedoch selbst dafür verantwortlich, dass diese Records stets auf dem aktuellen Stand sind. Handelt es sich zusätzlich um viele Einträge, kann der Arbeitsaufwand sehr schnell hoch werden.

Zudem besteht ein Caching-Nachteil. Ändert man einen A-Record kann es unter Umständen bis zu einem Tag dauern bis die Änderung propagiert ist. Kommt nun ein Client über einen DNS welcher noch die alte IP im Cache hatte, kann er die Applikation nicht benutzen [Dns131].

4.3.4 A-Record Service

Einen Service nutzen, welcher die A-Records erstellt und jeweils updatet [Int13] [Dns13].
Auch bei dieser Lösung hat man das Caching-Problem der DNS-Server, da es sich um normale A-Records handelt welche wieder durch das ganze Netz propagiert werden müssen.

4.3.5 Weiterleitungs-Service

Durch die Nutzung eines Service, welcher beim Aufruf eine HTTP-Weiterleitung macht, kann man das Caching-Problem umgehen. Jedoch stösst man hier auf einen Single Point of Failure. Bricht ein Server des Weiterleitungs-Service ein, kann man über den A-Record die Applikation nicht aufrufen.

4.4 Erkenntnis

Die gewonnenen Erkentnisse können unter Umständen sehr wichtig für Betreiber von Applikationen bei einem PaaS-Provider sein. Man muss sich beispielsweise bewusst sein, dass eine Applikation, welche lange nicht mehr aufgerufen wurde, eine Weile braucht, bis sie antwortet. Dies als Folge des Sleep-Modus, welcher der Cloud-Anbieter nach einer gewissen Zeit einstellt.
Die verschiedenen Möglichkeiten in Bezug auf das A-Record Caching-Problem sind interessant und sollten besonders im Hinblick auf eine mögliche Migration zu einem Cloud-Provider evaluiert werden.

5 Einfache Java SE Applikationen in der Cloud

Neben Java Webprojekten gibt es in der Cloud auch die Möglichkeit, einfache Java SE Applikationen auszuführen. Dies können beispielsweise Prozesse sein, welche einmalig ausgeführt werden, oder aber auch Hintergrund-Tasks, welche länger dauern.
Im zweiten Teil dieses Dokumentes befinden sich Anleitungen für das Erstellen einer HelloWorld-Applikation sowie das Deployment auf die jeweiligen Cloud Providern.
Im Rahmen unserer Recherchen hat sich gezeigt, dass es zurzeit unter Google nicht möglich ist, ein Java Projekt anzulegen, welches keine Servlets enthält.

5.1 Heroku

5.1.1 Ausgangslage

Ausgangslage ist ein Java Projekt mit einer einfachen Klasse *HelloWorld*, welche in der Main-Methode „Hello World“ ausgibt [Run13]. Die Klasse befindet sich im Package „*ch.hsr.helloWorld*“.

```
1  package ch.hsr.helloWorld;
2
3  public class HelloWorld {
4    public static void main(String[] args) {
5      System.out.println("Hello World");
6    }
7  }
```

Quellcode 3 - Heroku Hello Word

5.1.2 Application Assembler Maven Plugin

Um die Applikation nach dem deployen zu starten, benötigt es ein Skript, welches die Applikation ausführt. Nach dem Umwandeln des Projektes in ein Maven-Projekt, wird diesem das Application Assembler Maven Plugin hinzugefügt. Dieses Maven Plugin generiert automatisch Skripte zum Starten der Java Applikation.

```
1  <plugin>
2    <artifactId>appassembler-maven-plugin</artifactId>
3    <version>1.1.1</version>
4    <configuration>
5      <assembleDirectory>target</assembleDirectory>
6      <programs>
7        <program>
8          <mainClass>ch.hsr.HelloWorld.HelloWorld</mainClass>
9          <name>helloWorld</name>
10       </program>
11     </programs>
12   </configuration>
13   <executions>
14     <execution>
15       <phase>package</phase><goals><goal>assemble</goal></goals>
16     </execution>
17   </executions>
18 </plugin>
```

Quellcode 4 - Maven Heroku Hello World

5.1.3 Lokaler Test

Nun kann man mit *mvn package* das Projekt bauen und anschliessend mit folgendem Befehl ausführen:

Linux/OSX:

```
1  sh target/bin/helloWorld
```

Windows:

```
1  target\bin\helloWorld.bat
```

Ausführen in der Cloud
Nach dem Deployment in die Cloud kann die Applikation durch folgenden Befehl gestartet werden:

Linux/OSX:

```
1  heroku run "sh target/bin/helloWorld"
```

Windows:

```
1  heroku run "target\bin\helloWorld.bat"
```

5.1.3.1 Alternativ: Procfile

Mittels eines Procfiles, welches sich im Root-Verzeichnis der Applikation befindet, kann zudem die Applikation direkt nach dem Deployment gestartet werden. Der Inhalt der Procfile Datei (ohne Endung) besteht in unserem Fall aus folgendem Befehl:

```
1  helloWorld: sh target/bin/helloWorld
```

Nach dem Deployment kann der Applikation ein Dyno zugewiesen und die Applikation gestartet werden. Dies erfolgt durch folgenden Befehl:

```
1  heroku ps:scale helloWorld=1
```

5.1.4 Ausgabe ansehen

Über den Befehl *heroku logs* kann die Ausgabe der Applikation angesehen werden.

5.1.5 Probleme

Leider haben wir beim Ausführen der Applikation jeweils eine „Error connecting to Process“ Fehlermeldung erhalten. In den Logs war nur ersichtlich, dass es sich um den Fehlercode R99 handelt. Dieser Fehlercode definiert interne Plattform-Errors [Err13].

5.2 CloudBees

5.2.1 Ausgangslage

Ausgangslage ist ein Java Projekt mit einer einfachen Klasse *HelloWorld*, welche in der Main-Methode „Hello World“ ausgibt [Clo13]. Die Klasse befindet sich im Package „*ch.hsr.helloWorld*“.

```
1  package ch.hsr.helloWorld;
2
3  public class HelloWorld {
4    public static void main(String[] args) {
5      System.out.println(”Hello World”);
6    }
7  }
```

Quellcode 5 - CloudBees Hello World

5.2.2 Ausführbare Jar-Datei

Für das Deployment in die Cloud benötigt es eine ausführbare Jar-Datei. Nachdem man das Projekt in Eclipse gestartet hat, kann mit einem Rechtsklick auf Export, die Datei als ausführbare Jar-Datei exportiert werden. Vor dem Export muss die gewünschte Applikation ausgewählt werden.

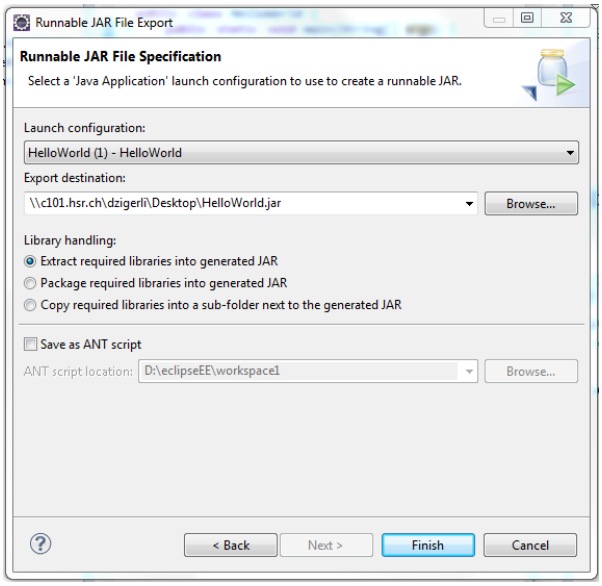


Abbildung 11 - Ausführbare Jar-Datei

5.2.3 Deployment

Über das CloudBees-SDK kann das Package mit folgendem Befehl deployed werden [JVM13]:

```
1  bees app:deploy –t java –R class=ch.hsr.HelloWorld.HelloWorld –R java_version=1.7
   HelloWorld.jar
```

Die Ausgabe kann nun entweder über das Webinterface von CloudBees oder über das SDK mit dem Befehl *bees app:tail APP_ID* angesehen werden.

5.3 Erkenntnis

Nach unseren Recherchen unterstützen nicht alle Cloud-Provider einfache Java SE Applikationen. Bei Google App Engine konnte keine Beispielapplikation realisiert werden. Heroku gibt zwar eine Anleitung zu diesem Thema, funktioniert hat dies im Rahmen unserer Studienarbeit nicht. Lediglich ein Worker-Thread konnte für die Erstellung eines Sockets erstellt werden. Somit bleibt zu hoffen, dass es sich in diesem Fall um eine vorübergehende Störung handelt und dies zwischenzeitlich wieder möglich ist.

6 Diagnoseanwendung Selbstauskunft

Damit sich ein Entwickler eine Übersicht über die von ihm verwendete Infrastruktur bei einem Cloud-Anbieter verschaffen kann, haben wir eine Selbstauskunftsapplikation erstellt. Die Grundidee war, dass sich die Applikation sämtliche für den Entwickler potentiell relevante Informationen vom aktuell deployten Server bezieht und diese dann via Servlet direkt im Browser anzeigt.

Abbildung 12 - SelfInformation Applikation Grundidee

6.1 Java Applikation

Eine grosse Menge an Informationen können mit Hilfe von Java erfragt werden. Gruppiert wurden die Informationen in folgende Teilbereiche:

- Hardware-Informationen
- Netzwerk-Informationen
- Systemeigenschaften
- Umgebungsvariablen

6.1.1 Problem JVM

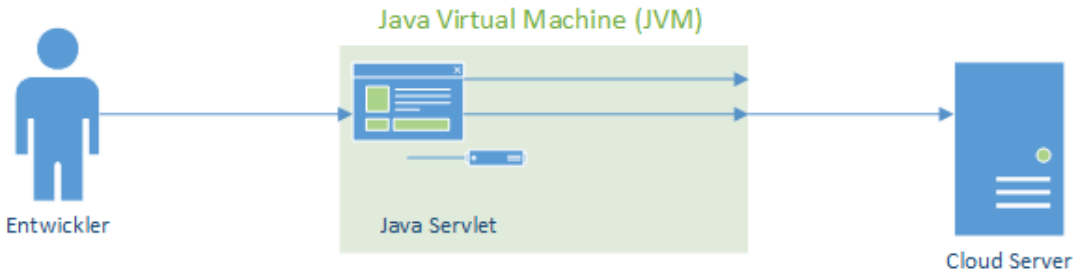


Abbildung 13 - SelfInformation Problem JVM

Durch die Nutzung von Java können jedoch nur Informationen der JVM ausgelesen werden. Die effektive Grösse des Arbeitsspeichers bleibt dem Java-Programmierer verborgen. Auszulesen ist nur die Grösse des Speichers, welcher der JVM zur Verfügung gestellt wird.

6.2 Endresultat

Die Applikation entsteht von einem Dynamic Web Project. Im Presentation Layer befindet sich ein Servlet. Der Business Logic Layer besteht aus POJO's, somit lässt sich die Applikation unkompliziert in andere Umgebungen integrieren.

Die HTML-Ausgabe des SelfInformation-Servlets zeigt das untenstehende Bild.
Die Ausgabe ist je nach Provider unterschiedlich. Bei Google App Engine können Informationen wie NetworkInformation und EnvironmentVariables nicht ausgelesen werden.

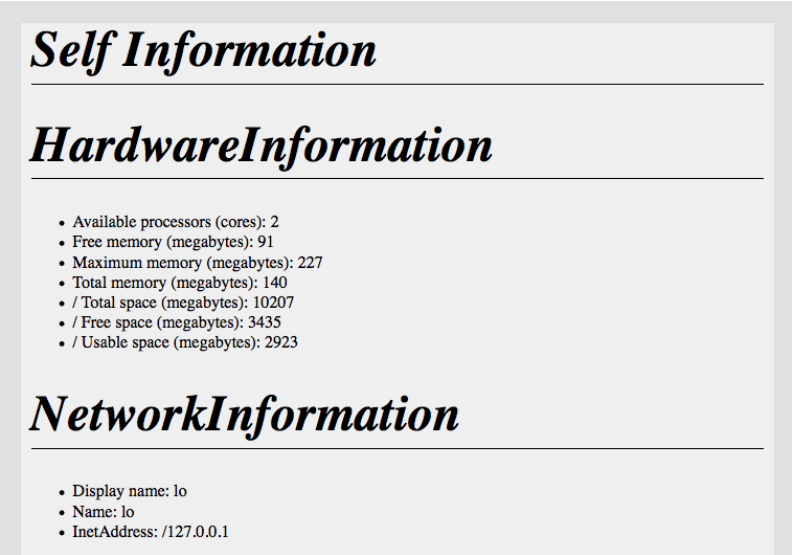


Abbildung 14 - SelfInformation Endresultat

6.2.1 Ausgabemöglichkeiten

Der Business Logic Layer ist so konzipiert, dass man zwischen einer HTML, XML und JSON Ausgabe wählen kann.

6.2.1.1 Beispiel Ausgabe HTML

Die HTML Ausgabe generiert einen *<h1>-Tag* sowie ein *<ul>-Listing* mit den verschiedenen Einträgen als *<li>-Eintrag*.

```
1  <h1>NetworkInformation</h1>
2  <ul>
3    <li>Display name: WAN Miniport (Network Monitor)-QoS Packet Scheduler-0000</li>
4    <li>Name: eth9</li>
5    <li>InetAddress: /fe80:0:0:0:0:5efe:c0a8:3eac%15</li>
6  </ul>
```

Quellcode 6 - SelfInformation HTML Output

6.2.1.2 Beispiel Ausgabe XML

Der erzeugte XML Code wurde mit dem W3C-XML-Validator als korrekt geprüft [Val13]. Anbei ein XML-Auszug des NetworkInformation-Bereichs.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<NetworkInformation>
  <NetworkInformation>
    <name>Display name</name>
    <value>WAN Miniport (Network Monitor)-QoS Packet Scheduler-0000</value>
  </NetworkInformation>
  <NetworkInformation>
    <name>Name</name>
    <value>eth9</value>
  </NetworkInformation>
  <NetworkInformation>
    <name>InetAddress</name>
    <value>/fe80:0:0:0:0:5efe:c0a8:3eac%15</value>
  </NetworkInformation>
</NetworkInformation>
```

Quellcode 7 - SelfInformation XML Output

6.2.1.3 Beispiel Augabe JSON

Durch die frei verfügbare Java-Bibliothek auf json.org wird der generierte XML Text durch wenige Zeilen Java Code in JSON konvertiert [JSO13].

```json
{
  "NetworkInformation": {
    "NetworkInformation": [
    {
      "name": "Display name",
      "value": "WAN Miniport (Network Monitor)-QoS Packet Scheduler-0000"
    },
    {
      "name": "Name",
      "value": "eth9"
    },
    {
      "name": "InetAddress",
      "value": "/fe80:0:0:0:0:5efe:c0a8:3eac%15"
    }
    ]
  }
}
```

Quellcode 8 - SelfInformation JSON Output

6.3 Workaround Python plist

Durch die Python Library plist ist es möglich, die effektive Grösse des Arbeitsspeichers zu ermitteln. Diese Library könnte noch weitere Informationen aus dem System auslesen. Im Rahmen dieser Studienarbeit verwendeten wir sie jedoch nur als Workaround für die Abfrage des Arbeitsspeichers.

Beispiel Heroku:

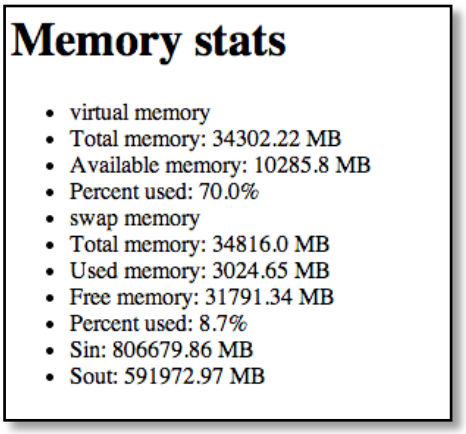


Abbildung 15 - SelfInformation Python

6.4 Erkenntnis

Die ausgelesenen Informationen erweisen sich als sehr interessant. Teilweise sind sie auch notwendig für die Realisierung von Programmfunktionalitäten wie zum Beispiel einem Socket. Es ist erstaunlich, wie viele Informationen ausgelesen werden können und wie sie sich in kurzer Zeit bereits wieder verändert haben.

Diese Beispielapplikation beinhaltet folgende Patterns:

- Execution Environment [Chr135]
- Data Access Component [Chr136]
- Dedicated Component [Chr137]

7 Nicht-relationale Datenbanksysteme

Folgende Beispielapplikationen zeigen dem Entwickler eine Übersicht zu den verschiedenen Datenbanken im Cloud Bereich. Dies veranschaulichen wir durch eine MySQL Applikation, deren Funktionalität anschliessend mittels MongoDB und Google Datastore nachgebaut wurde. Dafür verwenden wir MySQL und MongoDB von CloudBees, sowie Datastore von Google. Als Ausgangssituation besteht eine MySQL Applikation. Diese beinhaltet eine Join-Operation, eine typische Operation, welche jedoch bei nicht-relationalen Datenbanksystemen nicht zur Verfügung steht. Mit diesen Applikationen wird veranschaulicht, was umgestellt und auf was verzichtet werden muss, wenn man mit einer nicht-relationalen Datenbank arbeitet.

7.1 Aufbau der MySQL-Applikation

Die Applikation beinhaltet zwei Tabellen. Eine für den Kunden (Customer) und eine für den Vertrag (Contract). Customers und Contracts werden über den Browser hinzugefügt.

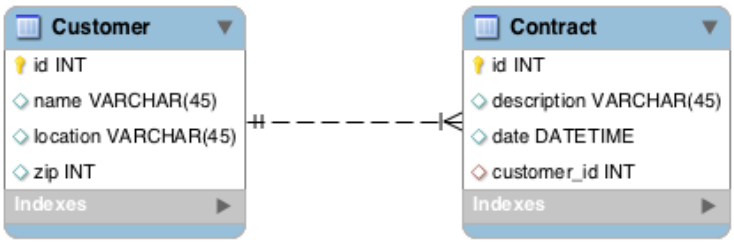


Abbildung 16 - Contract Management Tabellen

Ausserdem wird in der Applikation einen Join über die zwei Tabellen erzeugt. Dabei wird der Customer durch die Id in der Contract Tabelle referenziert.

7.2 Endresultat

Das nachfolgende Bild zeigt die Ansicht der Applikation nach einem Aufruf im Browser. Auf der linken Seite ist die Tabelle mit den Kunden zu sehen. Die Tabelle der Verträge befindet sich auf der rechten Seite. Da bei einem Vertrag nur die Id eines Kunden eingetragen ist, wird durch einen Join die dritte Tabelle erzeugt, welche unterhalb zu sehen ist.

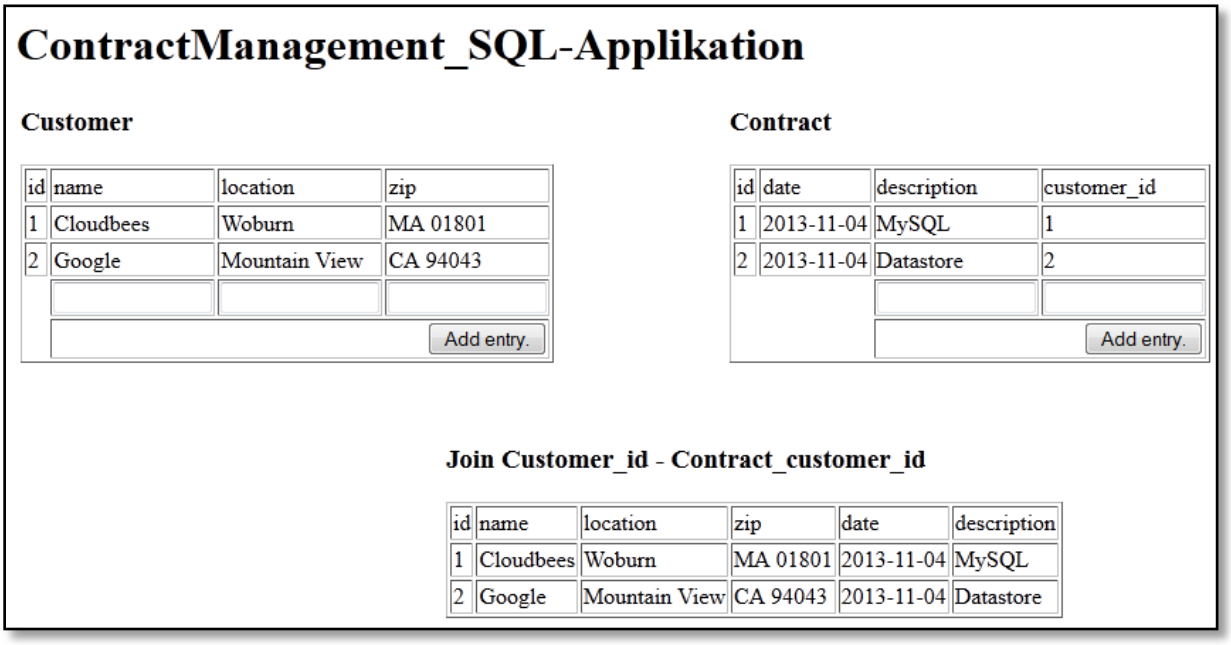


Abbildung 17 - ContractManagement Browser Output

7.3 Beispielapplikationen

7.3.1 CloudBees MySQL

Bei CloudBees besteht die Möglichkeit, mit relationalen Datenbanksystemen wie MySQL zu arbeiten [SKe12]. Somit besteht die Möglichkeit, Join-Operationen zu nutzen.

7.3.1.1 Vorbedingung

Bevor mit der MySQL Datenbank von CloudBees gearbeitet werden kann, wird die entsprechende Datenbank auf CloudBees angelegt [Dat13]. Anschliessend werden die Kunden- sowie Vertragstabelle erstellt.
Der JDBC Treiber wird ins lib-Verzeichnis des Projektes kopiert und als Library dem Build-Path des Projektes hinzugefügt.
In der Datenbankklasse werden folgende vier Attribute benötigt [MyS13]:

```
1  private Connection connect = null;
2  private Statement statement = null;
3  private PreparedStatement preparedStatement = null;
4  private ResultSet resultSet = null;
```

Quellcode 9 - MySQL Attribute

7.3.1.2 Verbindung mit CloudBees MySQL herstellen

Zuerst wird die JDBC Klasse dynamisch geladen. Anschliessend kann man sich mit folgendem Programmcode zur Datenbank verbinden.

```
1  private final String databaseConnection = "jdbc:<Server>/<Schemaname>?" +
       "user=<Benutzername>&password=<Passwort>");
2  Class.forName("com.mysql.jdbc.Driver");
3  connect = DriverManager.getConnection(databaseConnection);
4  statement = connect.createStatement();
```

Quellcode 10 - MySQL Verbindungsaufbau

7.3.1.3 Beenden der Verbindung

Folgender Code trennt die Verbindung zur Datenbank.

```
1  private void close() {
2    try {
3      if (resultSet != null) {
4        resultSet.close();
5      }
6      if (statement != null) {
7        statement.close();
8      }
9      if (connect != null) {
10       connect.close();
11     }
12   } catch (Exception e) {}
```

Quellcode 11 - MySQL Verbindung beenden

7.3.1.4 Eintrag hinzufügen

Bei bestehender Verbindung zur Datenbank können Einträge vorbereitet und ausgeführt werden.

```
1  preparedStatement = connect.prepareStatement("insert into <Schemaname>.<Tabellenname>
   values (\"+ "Hans“ + ")");
2  preparedStatement.executeUpdate();
```

Quellcode 12 - MySQL Eintrag hinzufügen

7.3.1.5 Eintrag abfragen

Die Abfrage auf die Datenbank ist ähnlich wie das Hinzufügen. Zuerst die Query vorbereiten und in das Resultset sichern. Dann über die Collection iterieren, um die einzelnen Einträge zu erhalten. Anschliessend die einzelnen Zeileneinträge auslesen. Die einzelnen Spaltennamen können über die Metadaten mit der Spaltennummer ausgelesen werden.

```
1  resultSet = statement.executeQuery("select * from <Schemaname>.<Tabellenname>");
2
3  while(resultSet.next()) {
4    CDAR_Contract c = new CDAR_Contract(
       resultSet.getString(1), resultSet.getString(2),
       resultSet.getString(3), resultSet.getString(4));
5  }
6
7  java.sql.ResultSetMetaData rsmd = resultSet.getMetaData();
8  CDAR_Contract header = new CDAR_Contract(
     rsmd.getColumnName(1), rsmd.getColumnName(2),
     rsmd.getColumnName(3), rsmd.getColumnName(4));
```

Quellcode 13 - MySQL Query

7.3.1.6 Implementierung Join-Operation

In folgendem Beispielcode wird eine einfache Joinabfrage gemacht und als Liste zurückgegeben.

Nach der Abfrage eines Select Statements, kann über das erhaltene ResultSet iteriert werden. Durch die Resultate der Abfrage werden in diesem Beispiel direkt Java Objektinstanzen von Contracts erstellt.

```
1   private ArrayList<CDAR_Contract> getContracts() throws SQLException {
2     ArrayList<CDAR_Contract> list = new ArrayList<>();
3
4     resultSet = statement.executeQuery("select cu.id, cu.name,
        co.description from feedback.customer as cu inner join
        FEEDBACK.contract as co ON cu.ID     = co.customer_id order by 1");
5
6     // Columnname
7     java.sql.ResultSetMetaData rsmd = resultSet.getMetaData();
8     CDAR_CustomerContractJoin header = new CDAR_CustomerContractJoin (
      rsmd.getColumnName(1), rsmd.getColumnName(2), rsmd.getColumnName(3),
      rsmd.getColumnName(4), rsmd.getColumnName(5), rsmd.getColumnName(6));
9     list.add(header);
10
11    while (resultSet.next()) {
12      CDAR_CustomerContractJoin  j = new CDAR_CustomerContractJoin(
          resultSet.getString(rsmd.getColumnName(1)), resultSet.getString(2),
          resultSet.getString(3), resultSet.getString(4),
          resultSet.getString(5), resultSet.getString(6));
13      list.add(j);
14    }
15  return list;
```

Quellcode 14 - MySQL Join

7.3.1.7 Was ist zu beachten

Damit Tabellen in der Datenbank angelegt werden können, muss Port 3306 offen sein. Dieser Port kann nicht geändert werden.

7.3.2 ContractManagement MongoDB

MongoDB ist eine dokumentorientierte Key-Value Datenbank [Chr13], welche Sammlungen von BSON[1] Dokumenten verwaltet [Mon13] [BSO13]. Sämtliche Objekte lassen sich beliebig verschachteln. MongoDB gehört zur verbreitesten NoSQL-Datenbank [SKe13].

7.3.2.1 MongoDB Data Model

Eine Instanz von MongoDB beinhaltet mehrere Datenbanken. Jede Datenbank besteht aus verschiedenen Collections. Diese wiederum bestehen aus verschiedenen Dokumenten. Ein Dokument setzt sich zusammen aus mehreren Key-Value-Pairs und hat ein dynamisches Schema. Dies bedeutet, dass unterschiedliche Dokumente in derselben Collection nicht dasselbe Schema aufweisen müssen, sondern verschieden sein dürfen.

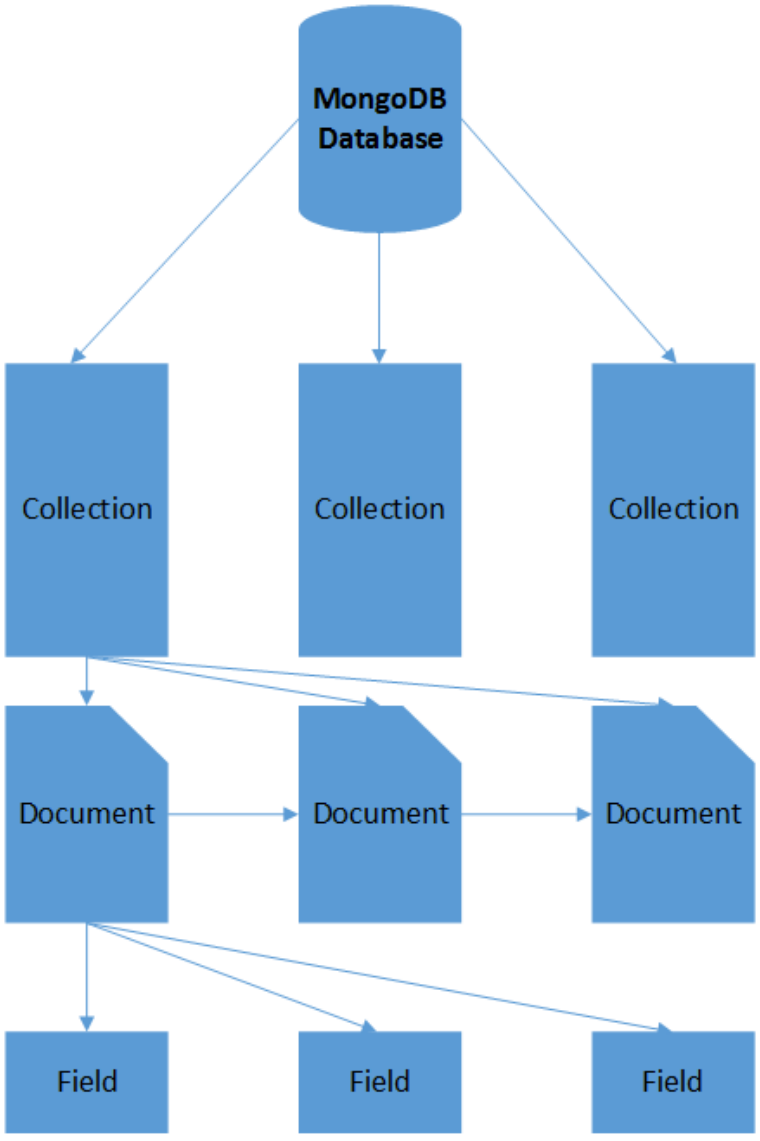


Abbildung 18 - MongoDB Data Model

[1] "binary-encoded serialization of JSON-like documents"

7.3.2.2 BSON

MongoDB speichert die Daten im BSON-Format [Das13]. BSON ist eine Erweiterung des JavaScript Object Notation (JSON) Formats. Dieses hat das Problem, dass es eine beschränkte Anzahl an unterstützten Datentypen hat. Binärdaten lassen sich nur schlecht mit JSON übertragen, da man sie vor der Übertragung noch mit Base64 serialisieren müsste. Zudem fehlt beispielsweise auch die Unterstützung von Datumswerten.

Beispiel eines Customer Objektes aus der ContractManagement-Applikation:

```
{
  "name": "Daniel Zigerlig",
  "location": "St. Gallen",
  "zip": "9000",
  "_id": {
    "$oid": "528a11d99c659bf338969d14"
  }
}
```

Quellcode 15 - BSON Customer-Objekt

7.3.2.2.1 Object ID

Ähnlich wie bei relationalen Datenbanksystemen erhält auch jedes Objekt bei MongoDB eine ID, die sogenannte _oid. Diese kann man optional selbst wählen, wird jedoch auch automatisch durch MongoDB gesetzt.

7.3.2.2.2 Verfügbare Datentypen

Datentyp	Beispiel
String	{ "string" : "myString" , "_id" : { "$oid" : "528a19a09c65ec0c9efc2351"}}
Integer	{ "integer" : 42 , "_id" : { "$oid" : "528a19a09c65ec0c9efc2352"}}
Float	{ "float" : 314000.0 , "_id" : { "$oid" : "528a19a09c65ec0c9efc2353"}}
Boolean	{ "boolean" : false , "_id" : { "$oid" : "528a19a09c65ec0c9efc2354"}}
Array	{ "array" : [1 , 2 , 3] , "_id" : { "$oid" : "528a19a09c65ec0c9efc2355"}}
Embedded-Document	{ "document" : { "embeddedDocument" : 42} , "_id" : { "$oid" : "528a19a09c65ec0c9efc2356"}}
Date	{ "date" : { "$date" : "2013-11-18T13:44:00.125Z"} , "_id" : { "$oid" : "528a19a09c65ec0c9efc2357"}}
DBRef	{ "street" : "exampleStreet" , "ref person" : { "$ref" : "DemoCollection" , "$id" : { "$oid" : "528a19a09c65ec0c9efc2358"}}}
ObjectId	{ "Demo" : "example" , "_id" : { "$oid" : "528a19a09c65ec0c9efc2359"}}

Tabelle 5 - MongoDB Datentypen

7.3.2.3 Customer und Contract Collection

Das Erstellen der Collections ist bei MongoDB nicht nötig. Jede Abfrage auf eine Collection welche noch nicht existiert, erstellt diese automatisch. Beispielsweise kann man im Data Access Layer die Funktionen für das Abfragen oder Einfügen von Customer ohne vorheriges Erstellen der Connection programmieren [Dat133]. Wird der Code nun ausgeführt, wird die Collection automatisch erstellt.

7.3.2.4 Verbindung mit MongoDB herstellen

Voraussetzung zum Aufbau einer Verbindung mittels Java zu MongoDB ist die mongo.jar im Classpath [Dat131]. Durch das Hinzufügen einer Dependency im pom.xml kann man mit Maven die jar-Datei dem Projekt hinzufügen [Get13] [Mon133].

```
1  <dependency>
2    <groupId>org.mongodb</groupId>
3    <artifactId>mongo-java-driver</artifactId>
4    <version>2.7.2</version>
5  </dependency>
```

Quellcode 16 - MongoDB Dependency

Durch nachfolgenden Befehl wird eine Verbindung zur Datenbank hergestellt. Der Port kann je nach DB-Instanz bei MongoHQ variieren. Der Port ist durch den Provider vorgegeben.

```
1  MongoURI mongoURI = new MongoURI("mongodb://user:pwd@paulo.mongohq.com:10055/dbname");
2  DB db = mongoURI.connectDB();
3  db.authenticate(mongoURI.getUsername(), mongoURI.getPassword());
```

Quellcode 17 - MongoDB Verbindungsaufbau

7.3.2.5 Eintrag in die Collection hinzufügen

Um einen Customer- oder Contract-Eintrag in die Collection hinzuzufügen, wird zuerst die Collection ausgewählt, in welcher der Eintrag erfolgen soll. Danach wird ein neues Objekt kreiert und der Collection eingefügt. Folgender Codeausschnitt beschreibt das Hinzufügen eines Contract Objektes (Attribute „description“ und „customerId“). Durch die Anwendung von *append* können verschiedene Attribute verkettet werden.

```
1  DBCollection myContractsCollection = db.getCollection("contract");
2  BasicDBObject myDbObject = new BasicDBObject("description",
   myContract.getDescription()).append("customerId", myContract.getrefID());
3  myContractsCollection.insert(myDbObject);
```

Quellcode 18 - MongoDB Eintrag hinzufügen

7.3.2.6 Implementierung Join Operation

Um zusammenhängende Einträge zu verknüpfen, existiert bei der dokumentenbasierten NoSQL-Technologie keine Join-Operation. Die Implementierung erfolgt somit manuell. Unser erster Ansatz erwies sich jedoch nach den ersten Tests als performanceintensiv. Nach diversen Anpassungen erfolgten zwei weitere Lösungen, welche performanter agieren.

7.3.2.6.1 1. Lösung - Join Operation durch mehrere Abfragen

Zuerst unsere erste Lösung, welche über die einzelnen Contracts iteriert. Pro gefundenen Contract in der Resultatmenge wird eine Abfrage auf die Customer-Collection gemacht. Diese Abfrage sucht den betreffenden Customer und gibt ihn zurück. Dies ermöglicht die Anzeige des Vertrages mit dem kompletten Namen des Kunden.

Nachteil: Für jeden Eintrag in der Contract-Collection ist eine Abfrage auf die Customer-Collection notwendig, dies benötigt jedoch unnötig viele Abfrage auf die Datenbank, was diese Lösung sehr ineffizient macht [Mon132].

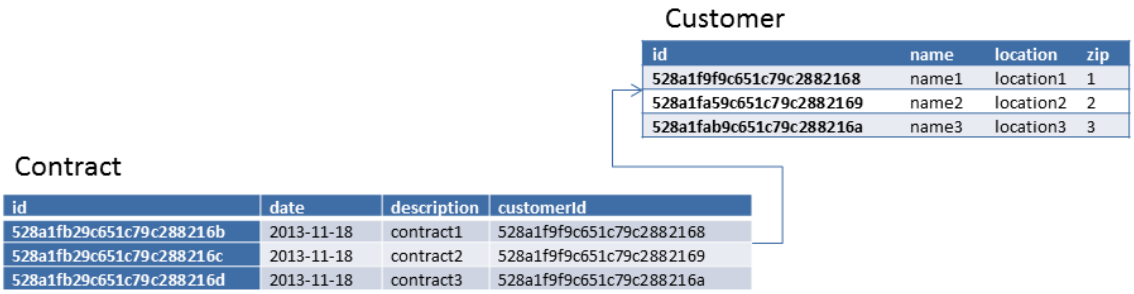


Abbildung 19 - Contract Management MongoDB Ablauf

7.3.2.6.1.1 Java Code

```
1  public ArrayList<Join> getJoinedContacts() {
2    ArrayList<Join> list = new ArrayList<Join>();
3    try {
4      DBCollection collContracts = db.getCollection("contract");
5      collContracts.distinct("description");
6      DBCursor cursorContracts = collContracts.find();
7      while (cursorContracts.hasNext()) {
8        DBObject element = cursorContracts.next();
9        String customerId = element.get("customerId").toString();
10       String description = element.get("description").toString();
11       DBObject customer = getCustomer(customerId);
12       list.add(new Join("XXX", customer.get("name").toString(), description));
13     }
14   } catch (MongoException e) {
15     e.printStackTrace();
16   }
17   return list;
18 }
```

Quellcode 19 - MongoDB Join 1. Lösung

```
1  private DBObject getCustomer(String id) throws MongoException, UnknownHostException {
2    DBCollection collCustomers = db.getCollection("customer");
3    ObjectId _id= new ObjectId(id);
4    BasicDBObject obj = new BasicDBObject();
5    obj.append("_id", _id);
6    BasicDBObject query = new BasicDBObject();
7    query.putAll((BSONObject)query);
8    return collCustomers.findOne(query);
9  }
```

Quellcode 20 - MongoDB getCustomer

7.3.2.6.2 2. Lösung - Join Operation im Memory

Um die Join Operation zwischen den zwei Collections effizienter zu gestalten, werden beide Collections komplett in eine HashMap gespeichert. Die eigentliche Join Operation erfolgt nun im Memory, da direkt über die HashMaps iteriert und verglichen wird. So sind einmalig nur zwei Abfragen für alle Dokumente der Collections nötig.
Nachdem beide Collections in je eine HashMap gespeichert worden sind, wird folgende Logik ausgeführt, um aus den zusammenhängenden Einträgen die gewünschten Objekte zu kreieren.
Diese Lösung erweist sich vor allem bei grossen Datenmengen als sehr speicherintensiv.

```
1  for (String key : contractsHashMap.keySet()) {
2    if (customersHashMap.containsKey(key)) {
3     list.add(new Join(key, customersHashMap.get(key), contractsHashMap.get(key)));
4    }
5  }
```

Quellcode 21 - MongoDB Join im Memory

7.3.2.6.3 3. Lösung – Embedded Documents

MongoDB erlaubt es, Dokumente zu verschachteln. Dabei kann einem Dokument als Attribut ein weiteres Dokument angegeben werden. Dadurch braucht es keine Anfragen auf verschiedene Collections.
Man muss sich in einem solchen Fall jedoch bewusst sein, dass man die Daten redundant speichert. Das heisst im Falle einer Änderung, wie in unserem Fall eines Customers, muss man beide Collections editieren. Dies, weil es sich nicht um eine Referenz handelt. Zudem existiert ein gekoppelter Lebenszyklus.

Nachfolgend das JSON-Objekt eines Contract Eintrages.

```
1  {
2    "_id": {
3      "$oid": "527519439c653cec20b7292e"
4    },
5    "description": "Contract 1",
6    "customer": {
7      "_id": {
8        "$oid": "527519399c653cec20b7292d"
9      },
10     "name":"Customer A",
11     "location": "City",
12     "zip": "1234"
13   },
14   "date": "2013-11-02"
15 }
```

Quellcode 22 - JSON Embedded Document

7.3.3 ContractManagement Google Datastore

Bei Google Datastore handelt es sich um einen verteilten, Key-Value Datenspeicherdienst [Cre13]. Dieser besitzt keine relationale Schemata wie beispielsweise MySQL. Jeder Applikation bei Google App Engine wird eine Instanz eines Datastores zugewiesen. Google Datastore unterstützt ACID-Transaktionen und nutzt Optimistic Concurrency [Wha13].

7.3.3.1 Entitäten

Datenobjekte in Google Datastore nennt man Entitäten. Diesen können ein oder mehrere Attribute (Eigenschaften) hinzugefügt werden [Ent13]. Diese Attribute können vom Typ Integer, Floating-points, Strings, Datumswerten oder binäre Daten sein.
Entitäten besitzen zudem einen eindeutigen Schlüssel. Dieser setzt sich aus einer Entitäts-ID und einem Typ zusammen. Die Entitäts-ID wird vom Datenspeicher bereitgestellt, kann aber auch selbst erzeugt werden. Über die Entitäts-ID kann die Entität gezielt und schnell abgefragt werden. Es kann aber auch über den Typ eine Abfrage erstellt werden. Diese liefert jedoch alle Entitäten mit dessen Typ. Möglich wäre auch, dass Entitäten wiederum Entitäten enthalten, somit wird eine Hierarchie aufgebaut. Eine weitere Möglichkeit wäre auch, eine Entitäts-ID als Attribut einer anderen Entität zu nehmen.

7.3.3.2 Customer und Contract Collection

Die Typen der einzelnen müssen sich nicht zwingend unterscheiden. Bei einer Abfrage wird eine Collection der Entitäten zurückgegeben, auf welche die Anfrage passt. Über diese Collection kann iteriert werden.

7.3.3.3 Verbindung mit Google Datastore herstellen

Mit der Google Datastore Datenbank wird über eine Service-Instanz kommuniziert [Que13]. Folgender Java-Befehl erstellt einen DatastoreService mit der Standard-Config:

```
1   DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
```

Pro Applikation ist nur ein Datastore verfügbar, auf diesen kann aber mit allen Versionen der Applikation zugegriffen werden [Pac13]. Somit gehen Daten nach einem Deployment einer neuen Version nicht verloren.

7.3.3.4 Eintrag hinzufügen

Es können nur bereits erzeugte Einträge gespeichert werden. Um die Entität zu spezifizieren, ist es möglich, weitere Properties hinzuzufügen. Abschliessend muss die Entität in die Datenbank gespeichert werden.

```
1  Entity e = new Entity("Customer");
2  e.setProperty("name", "Hans");
3  datastore.put(e);
```

Quellcode 23 - Datastore Eintrag hinzufügen

7.3.3.5 Eintrag abfragen

Die Entitäten können über deren Enitätenname abgefragt werden. Diese Abfrage wird als eine Collection zurückgeschickt, über welche iteriert wird (siehe auch Kapitel “7.3.1.6 Implementierung Join-Operation“).

```
1  Query q = new Query("Customer");
2  PreparedQuery pq = datastore.prepare(q);
3  for (Entity result : pq.asIterable()) {}
```

Quellcode 24 - Datastore Eintrag abfragen

7.3.3.6 Implementierung Join Operation

Bei einem Datenspeicher wie Datastore gibt es, anders als bei relationalen Datenbanksystemen, keine Joins. Dadurch ist es nicht möglich diese Operation einzusetzen, um Einträge zu verknüpfen.
Dieses Problem wurde durch folgenden Beispielcode gelöst. Innerhalb einer Iteration über sämtliche Contract-Einträge gelangt man an die Id's der Customers. Durch diese kann man mittels einer weiteren Abfrage den dazugehörenden Kunden ermitteln.

```
1  public ArrayList<CDAR_CustomerContractJoin> getJoin() {
2    ArrayList<CDAR_CustomerContractJoin> joinList =
               new ArrayList<CDAR_CustomerContractJoin>();
3    ArrayList<CDAR_Customer> customersList = getCustomers();
4
5    for (CDAR_Contract con : getContracts()) {
6      int conRefID = 0;
7      try {
8        conRefID = Integer.parseInt(con.getRefID());
9      } catch (NumberFormatException e) { }
10
11     for (CDAR_Customer cus : customersList) {
12       int cusID = 0;
13       try {
14         cusID = Integer.parseInt(cus.getID());
15       } catch (NumberFormatException e) { }
16
17       if (cusID == conRefID && cusID != 0) {
18         joinList.add(new CDAR_CustomerContractJoin(
                                      Integer.toString(cusID), cus.getName(),
                                      cus.getLocation(), cus.getZip(),
                                      con.getDate(), con.getDescription()));
19
20       }
21     }
22   }
23   Collections.sort(joinList, new CDAR_CustomerContractJoin());
24
25   //Columnnames
26   joinList.add(0, new CDAR_CustomerContractJoin("customer_id", "name",
                                     "location", "zip",
                                     "date", "description"));
27
28   return joinList;
```

Quellcode 25 - Datastore Join-Operation

```
1  public ArrayList<CADR_Customer> getCustomers() {
2    ArrayList<CDAR_Customer> customerList = new ArrayList<CDAR_Customer>();
3    Query q = new Query("Customer");
4    PreparedQuery pq = datastore.prepare(q);
5
6    for (Entity result : pq.asIterable()) {
7      customerList.add(new CDAR_Customer(
         result.getProperty("id").toString(),
         result.getProperty("name").toString(),
         result.getProperty("location").toString(),
         result.getProperty("zip").toString()));
8    }
9
10   Collections.sort(customerList, new CDAR_Customer());
11
12   //Columnnames
13   customerList.add(0, new CDAR_Customer("id", "name", "location", "zip"));
14   return customerList;
15 }
```

Quellcode 26 - Datastore getCustomers

7.3.3.7 Was ist zu beachten

Wenn keine Datenbankeinträge mehr zurückgegeben werden:
Möglicherweise ist das Tagespensum von Google App Engine bei der Gratisnutzung erreicht.

7.4 Erkenntnis

Bei No-SQL Datenbanksystemen stehen nicht die gleichen Funktionalitäten zur Verfügung wie bei relationalen Datenbanksystemen. Zum Beispiel werden keine Joins unterstützt. Falls trotzdem eine solche Operation auf einer No-SQL-Datenbank benötigt wird, ist dieser auf einem anderen Weg zu lösen.
Dabei ist zu beachten, dass der gewählte Weg möglichst wenige Datenbankabfragen ausführt.

Diese Beispielapplikation veranschaulicht folgende Patterns:

- Relational Database [Chr138]
- Key-Value Storage [Chr13]
- Map-Reduce (MongoDB) [Chr131]
- Pay-per-Use, Standard Google App Engine Funktionaliät [Chr139]
- Hybrid Data, MongoDB [Chr1310]

8 Nutzung von TCP Sockets in der Cloud

Ein Socket ist eine Software-Komponente, mit welchem sich eine Verbindung von einem Gerät zu einem anderem Gerät aufbauen lässt [Soc13]. Über diese Verbindung kann so eine TCP IP-Kommunikation hergestellt werden. Beim Erzeugen sind die Sockets an lokale Ports zu binden.

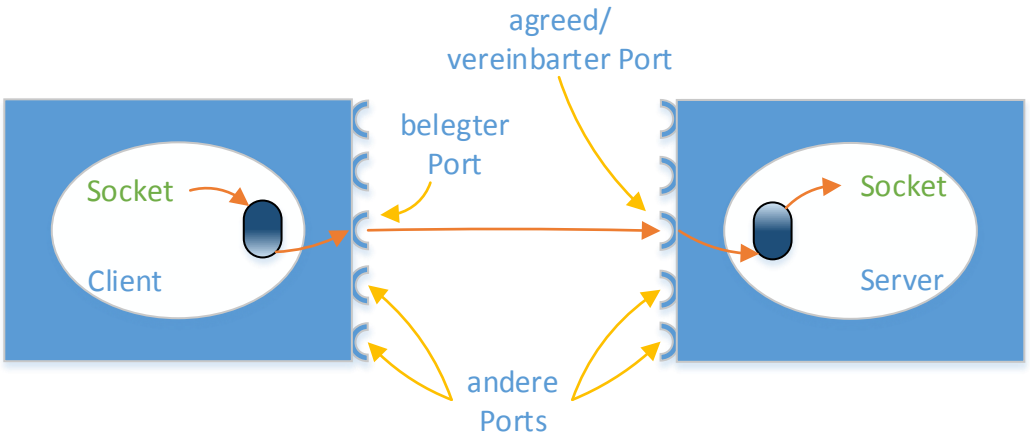


Abbildung 20 - Socketverbindung

Bei Sockets in der Cloudumgebung gibt es wichtige Unterschiede. Es ist darauf zu achten, ob sich die Socketverbindung in einer Instanz befindet, oder ob diese Socketverbindung auf eine andere Applikation übergreift. Deshalb entwickelten wir zwei verschiedene Applikationen. Die erste Anwendung kann nur in einer Applikation verwendet werden. Die zweite Applikation zeigt wie Sockets zu schreiben sind, welche auf ein anderes Devices zugreifen. In folgenden Beispielen geht es um eine einfache Anfrage, auf welche mit der Systemzeit geantwortet wird.

8.1 Server Socket

Der Server erstellt einen Socket welcher an einen Port gebunden ist. Nach dem Erstellen des Sockets steht eine Verbindungsschnittstelle zur Verfügung, auf welche verbunden werden kann. Danach ist es bereits möglich zum Server mit der IP und des ausgewählten Port eine Verbindung herzustellen.
Der Server wartet auf eine Verbindungsanfrage. Nach erfolgreichem Verbindungsaufbau (accept) sendet der Server dem Client eine Antwort. Anschliessend wird die Verbindung beendet und der Server steht auf dem gleichen Port für eine neue Verbindung zur Verfügung [JJo13].

8.2 Client Socket

Der Client verbindet auf die vom Server zur Verfügung gestellte Verbindungsschnittstellte. Dazu muss der Client aber den Port und die IP bzw. den Hostnamen wissen. Nach dem Verbindungsaufbau bekommt der Client vom Server eine Antwort

8.3 Kommunikation innerhalb einer Instanz

Diese Applikation läuft in derselben Instanz. Deshalb läuft jeweils der Server-, wie auch der Clientteil in einem eigenen Thread. Beide werden aus derselben Main-Funktion gestartet. Hierbei handelt es sich um eine reine Demonstration.

8.3.1 Server Socket

Auf der Cloudumgebung war nicht klar welche Ports für eine eigene Socketverbindung gebraucht werden kann. Folgend versuchten wir in der Applikation inkrementell von null beginnend ein Socket zu erstellen. Falls das Erstellen des Sockets fehlschlug, testen wir den gleichen Vorgang auf dem nächsthöheren Port. Diese Server-Socket-Applikation hat auf CloudBees gezeigt, dass alle Ports, welche kleiner als 1024 sind, gesperrt sind. Mit anderen Worten, es lässt sich kein Socket auf solch einen Port erstellen.

```
1  public class DaytimeServer extends Thread{
2    public static int DAYTIME_PORT = 1;
3
4    @Override
5    public  void run() {
6      while (true)
7        try {
8          System.out.println("Try to create a Socket on Port: "+ DAYTIME_PORT);
9
10         ServerSocket server = new ServerSocket(DAYTIME_PORT);
11         System.out.println("[DaytimeServer]waiting for clients...");
12
13         Socket connection = null;
14         // let it loop (wait for clients)
15         while (true) {
16           try {
17             connection = server.accept();
18             System.out.println("[DaytimeServer]Client " +
               connection.getInetAddress().getHostName() + "/" +
               connection.getInetAddress().getHostAddress() + " acepted!");
19             Date now = new Date();
20             out.write(now.toString() + "\r\n");
21             out.flush();
22             connection.close();
23           } catch (IOException e) { }
24           finally {
25             try {
26               if (connection != null) connection.close();
27             } catch (IOException e) { }
28           }
29         }
30       } catch (BindException z) {
31         System.out.println(DAYTIME_PORT + " failed");
32       } catch (IOException e) { }
33       DAYTIME_PORT++;
34     }
35   }
36 }
```

Quellcode 27 - Daytime Server Socket

8.3.2 Client Socket

In diesem Teil verbindet sich der Client Socket über den Befehl „new Socket(HOST, DAYTIME_PORT)“ mit dem vorherigen Server Socket

```
1  public class DaytimeClient extends Thread{
2
3    private static final String HOST = "localhost";
4
5    @Override
6    public  void run() {
7      Socket socket;
8      while (true)
9        try {
10         System.out.println("Try to connect to Socket: "+ HOST);
11         socket = new Socket(HOST, DaytimeServer.DAYTIME_PORT);
12
13         System.out.println(socket.getInetAddress().getHostAddress());
14
15         InetAddress inetAddr = socket.getInetAddress();
16         String xCanonicalHostName = inetAddr.getCanonicalHostName();
17
18         InputStream iS = socket.getInputStream();
19         BufferedReader br = new BufferedReader(
                   new InputStreamReader(iS));
20
21         // Read from the socket
22         String remoteTime = br.readLine();
23
24         System.out.println();
25         System.out.println("Time on " + xCanonicalHostName + " is "
                   + remoteTime);
26         System.out.println();
27
28         iS.close()
29         socket.close();
30         break;
31       } catch (IOException e) {
32       }
33     }
34   }
35 }
```

Quellcode 28 - Daytime Server Socket

8.3.3 Endresultat

Ausgabe der Applikation bei CloudBees:

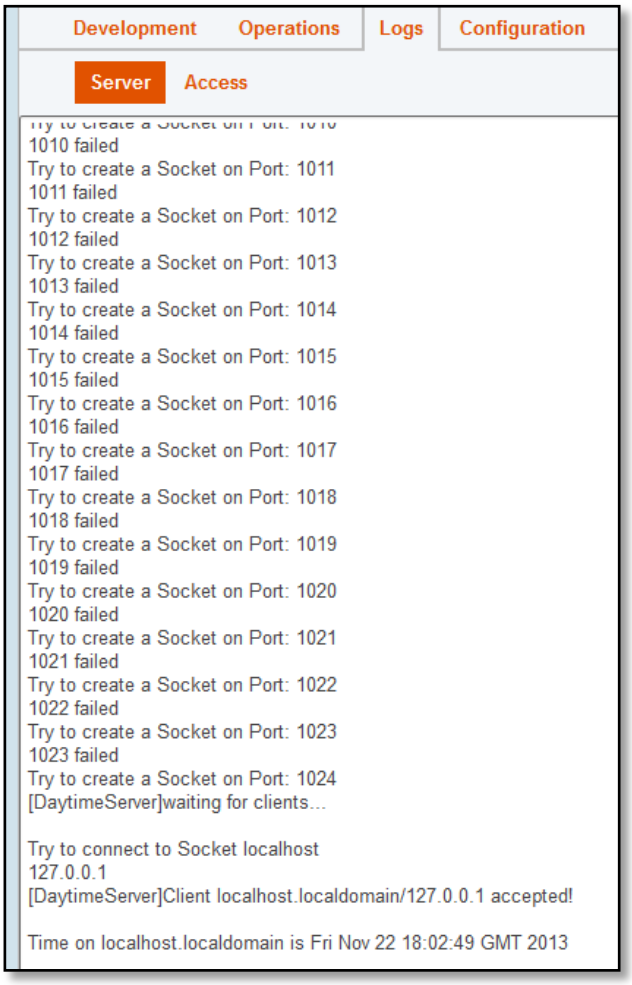


Abbildung 21 - CloudBees Output

8.4 Instanzübergreifende Kommunikation

Möchte man in der Cloud-Umgebung eine Socket-Applikation instanzübergreifend laufen lassen, muss die Applikation umgebaut werden. Da sich neben der eigenen Applikation auch viele Andere auf derselben Maschine befinden, kann eine Applikation nicht einfach auf einen Port binden und diesen besetzen. Dieser wäre dann für alle anderen Nutzer auf diesem Server nicht mehr belegbar. Deshalb wird jeder Applikation vom Cloud-Provider ein dynamischer Port zugewiesen. Die Applikation kann intern auf diesen Port binden und kommunizieren. Von aussen kann via Port 80 zugegriffen werden. Das Port-Mapping wird durch den Cloud-Provider übernommen.

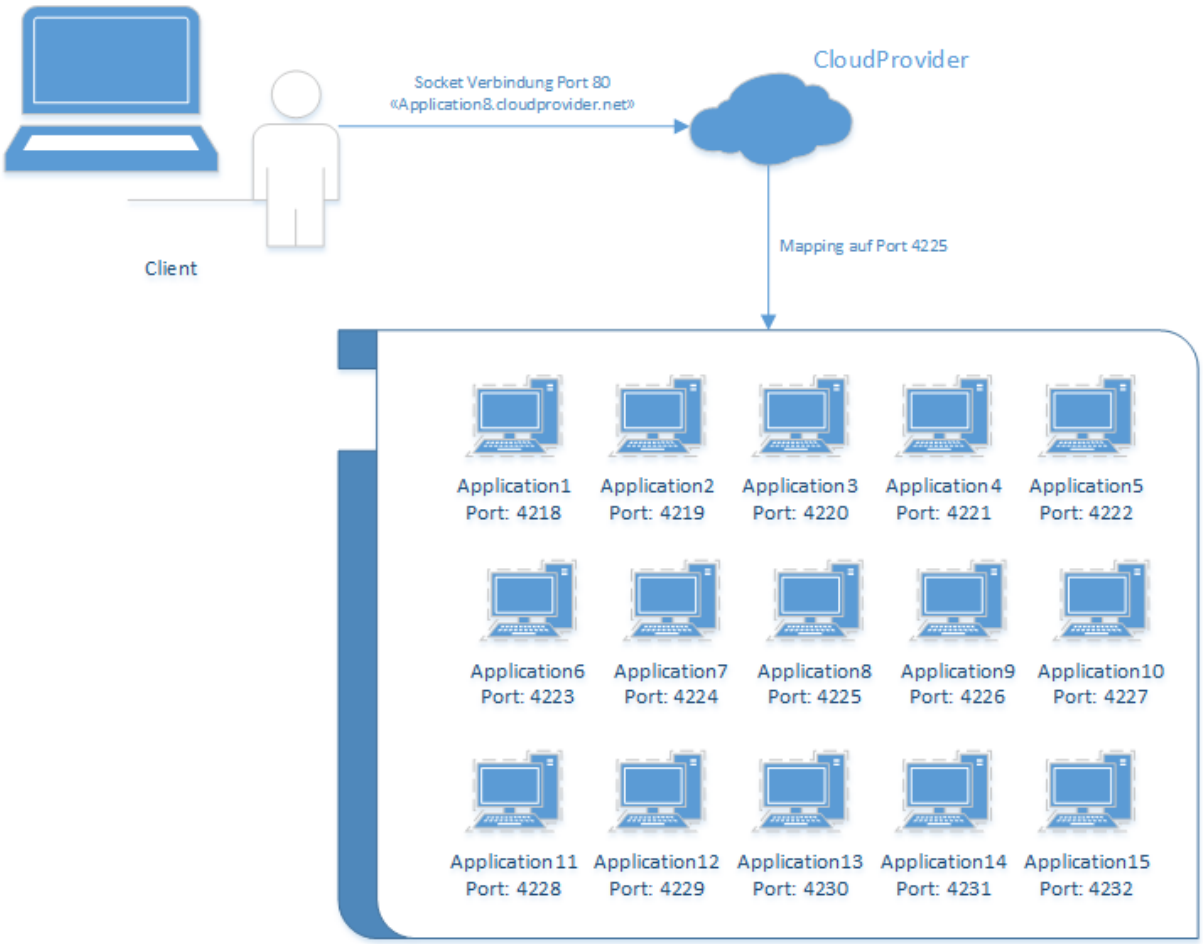


Abbildung 22 - Socket Portbinding

Dieser Port verändert sich bei jedem Deployment. Bei unseren Tests bei den Anbietern Heroku und CloudBees hat sich herausgestellt, dass nur HTTP-Traffic weitergeleitet wird [Fre13] [Pro13]. Zudem wird pro Applikation nur ein Port zugewiesen. Möchte man über mehrere Ports kommunizieren, ist ein Umbau der Applikation notwending.

8.4.1 Umbau der Applikation

Damit das Port-Mapping durch den Cloud-Provider funktioniert [Fin13], muss es sich beim Traffic um HTTP-Daten handeln. Dies kann an unserem Socket-Beispiel leicht eingebaut werden, indem der Server eine HTTP 1.1 Antwort zurücksendet. Aufgerufen wird der Socket nicht über ein Client-Socket, sondern direkt über den Browser.

8.4.1.1 Server Socket

Durch das Einfügen der HTTP 1.1 Headerinformationen kann HTTP-Traffic erzeugt werden.

```
1   Date now = new Date();
2   String text = "Time is " + now.toString();
3
4   out.write("HTTP/1.1 200 OK\r\n");
5   out.write("Server: Custom Heroku Java Server\r\n");
6   out.write("Content-Length: " + text.length() + "\r\n");
7   out.write("Content-Language: en\r\n");
8   out.write("Connection: close\r\n");
9   out.write("Content-Type: text/plain\r\n");
10  out.write("\r\n");
11
12  out.write(text + "\r\n");
```

Quellcode 29 - Server Socket HTTP 1.1

8.4.1.2 Zugriff auf die Portnummer

Die Applikation muss zur Laufzeit wissen, welcher Port ihr zugewiesen wurde. Die einzelnen Cloud-Provider handhaben dies unterschiedlich.

8.4.1.2.1 Heroku

Sobald es sich beim Deployment bei Heroku um eine Web-Anwendung handelt, setzt Heroku dieser Instanz eine Umgebungsvariable, welche den Port enthält. Durch die SelfInformation-Applikation kann dies einfach angezeigt werden.

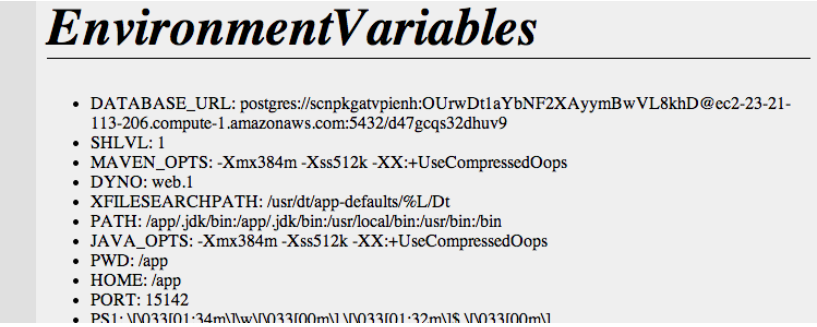


Abbildung 23 - Heroku Environment Variables

Die Abfrage des Ports kann mit folgendem Java Code durchgeführt werden:

```
1  Integer.parseInt(System.getenv().get("PORT"));
```

Da es sich beim Server Socket um eine normale Java Applikation und nicht um ein Web Application Projekt handelt, würde diese Variable nicht gesetzt werden. Um eine Setzung der Variable trotzdem zu ermöglichen, kann im Procfile angegeben werden, dass es sich um eine WebApplikation handelt. Somit wird beim Deployment ein Web-Dyno gestartet.

Inhalt Procfile:

```
1  web: sh target/bin/myApplication
```

8.4.1.2.2 CloudBees

Auch bei CloudBees wird eine Umgebungsvariable gesetzt, falls es sich um ein Web Projekt mit Tomcat-Webserver handelt. Somit wäre der Port in der Umgebungsvariable CATALINA_OPTS zu finden.

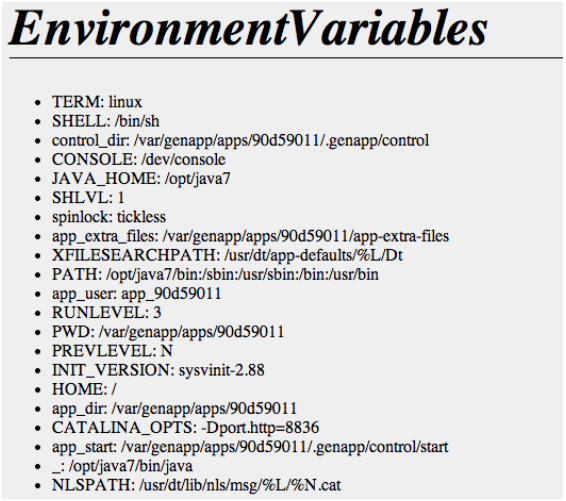


Abbildung 24 - CloudBees Environment Variables

Alternativ wird hier auch noch ein System-Property in der JVM gesetzt. Im Falle einer Web Applikation, heisst dieses Property *„port.http“*. Wird nun das Server Socket jedoch als Standalone Applikation deployed. Wird kein System-Property mit diesem Namen gesetzt.
In diesem Fall heisst das Property „app.port“ und kann mit folgendem Java Code in der Applikation abgefragt werden:

```
1    Integer.parseInt(props.getProperty("app.port"));
```

8.4.2 Endergebnis

Durch einen Aufruf im Browser sendet dieser eine Antwort mit der aktuellen Zeit.

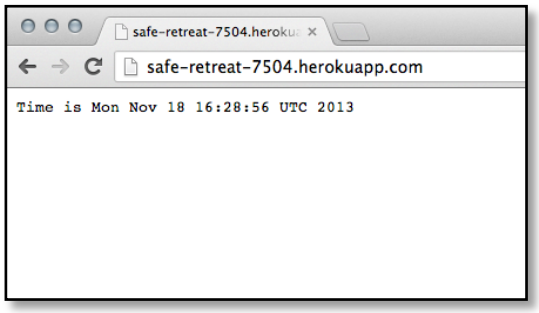


Abbildung 25 - Socket Endergebnis

8.5 Erkenntnis

Die eingeschränkte Port Nutzung sowie das festgelegte HTTP-Transportprokoll führt zu starken Einschränkungen des Anwendungsentwicklers. Alternativen bieten kostenpflichtige Services wie RabbitMQ (Heroku und CloudBees) sowie IronMQ (Heroku) [Add13] [Clo131].
Diese Beispielapplikation beinhaltet folgende Patterns:

- Execution Environment [Chr135]
- Dedicated Component [Chr137]

9 HTTP Zeitüberschreitung

Eine Webanwendung in der Cloud, welche breit skaliert ist und nebenläufig arbeitet, ist nur wirksam, wenn die Antwortzeiten kurz sind. Vorgeschlagener Wert, welcher beispielsweise Heroku angibt, wäre bei 500ms. Dies ist nicht nur im Sinne der Benutzer der Anwendung, sondern auch der Anbieter, da auf diese Art schneller Ressourcen wieder freigegeben werden. Dies ist der Grund, warum einige Anbieter intervenieren, falls eine Anfrage zu lange dauert, und senden deshalb eine Fehlermeldung an den Client, sodass der Request abbricht.
Im Rahmen unserer Studienarbeit haben wir nebst dem Studium der Dokumentation auch einen praktischen Test durchgeführt, um zu sehen, welche Anbieter nach welcher Zeit eingreifen und eine Fehlermeldung senden.

9.1 Wartendes Servlet

Seitens der Cloud wurde eine Web Anwendung geschrieben, welche beim Aufruf 180 Sekunden wartet, bis sie eine Antwort sendet. Beim Aufruf dieses Servlets kann der Cloud Provider nun eingreifen und den Request abbrechen, sowie eine Fehlermeldung schalten. Getestet wurde dies mit Unit Tests.
In Unit Tests wird das Servlet geöffnet, falls die Antwort nun vor Ablauf der 180 Sekunden zurückkommt und es sich hierbei nicht um die korrekte Antwort handelt, gilt der Unit Test als fehlgeschlagen.

9.2 Resultat

Eingegriffen wird bei Google App Engine sowie Heroku. Bei Heroku beträgt die maximale Zeit für einen Request 30 Sekunden [Req13]. Bei Google App Engine hat das Servlet bis zu 60 Sekunden Zeit, um eine Antwort zu senden [Url13]. Bei CloudBees wird auch nach drei Minuten Wartezeit noch die korrekte Antwortseite gesendet.

	Zeit
CloudBees	-
Google App Engine	60s
Heroku	30s

Tabelle 6 - HTTP Requesttimeout Zeitübersicht

Diese Zeiten werden von Google sowie Heroku fix gesetzt. Sie lassen sich nicht umstellen.

10 Map Reduce Pattern

Dieses Projekt implementiert das Map Reduce Pattern. Unser Fokus liegt aber nicht in der Generalisierung des Patterns, sondern bezieht sich auf eine konkrete Aufgabe, welche im Verlauf des nächsten Abschnittes erwähnt ist.

10.1 Map Reduce Einleitung

Map Reduce ist ein Verfahren, welches zur Verarbeitung von grossen Datenmengen eingesetzt wird [Map13] [Chr131]. Das Verfahren besteht aus zwei Phasen, der Aufteilung aller Daten in kleinere Stücke (Map) und das Zusammenfügen der errechneten Zwischenergebnisse (Reduce). Der Vorteil von Map Reduce ist, dass man das Verfahren sehr gut parallelisieren kann [LBl13]. Da sich Anwendungen in der Cloud einfach horizontal skalieren lassen, ist dieses Verfahren in diesem Zusammenhang gut umzusetzen. Somit sind auch nicht zahlreiche Rechenmaschinen im eigenen Serverraum notwendig.

Um das Pattern zu veranschaulichen und das Verhalten in der Cloud aufzuzeigen, schrieben wir ein Programm, welches das Map Reduce Pattern beinhaltet.

In diesem Map Reduce Beispiel sind eine grosse Anzahl an Agenten vorhanden. Diese Agenten haben einen Namen und eine zurückgelegte Strecke. Die Strecke besteht aus einem Start und einem Endpunkt. Start- sowie Endpunkt sind Ländernamen. Die Aufgabe des Map Reduce besteht in diesem Fall darin, alle Agenten nach dem Namen zu gruppieren und dessen Wege zusammenzufassen.

10.2 Map-Phase

In der Map-Phase teilt sich die ganze Datenmenge in kleinere Collections auf. Danach wird in der kleineren Collection über jeden Eintrag iteriert. Falls der Eintrag (Key) noch nicht in einer Zwischenliste (Key/Value) ist, fügt man diesen Eintrag hinzu.

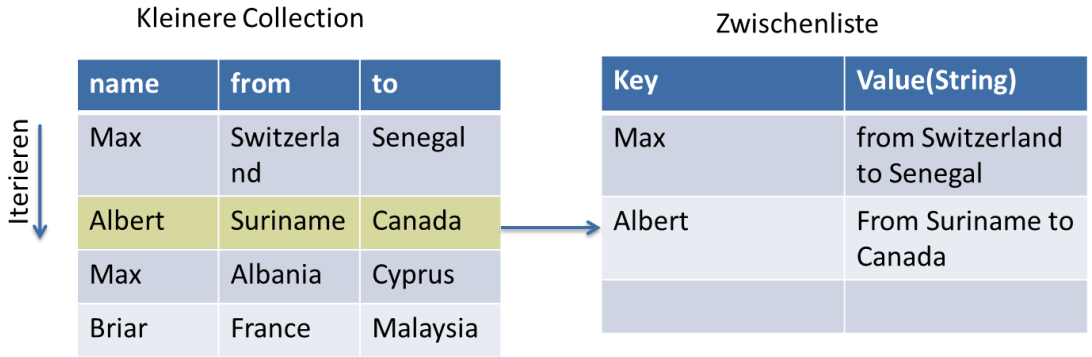


Abbildung 26 - Map Reduce Pattern: Map Phase

Falls der Eintrag jedoch bereits in der Zwischenliste existiert, wird nur der Value ergänzt. Das Iterieren über die einzelnen kleineren Collections kann jeweils parallelisiert werden. Somit findet der folgende Ablauf in mehreren Threads statt, für jede einzelne Collection Einen. Alle Threads verfolgen den gleichen Ansatz, jedoch ausschliesslich in ihrer eigenen Collection.

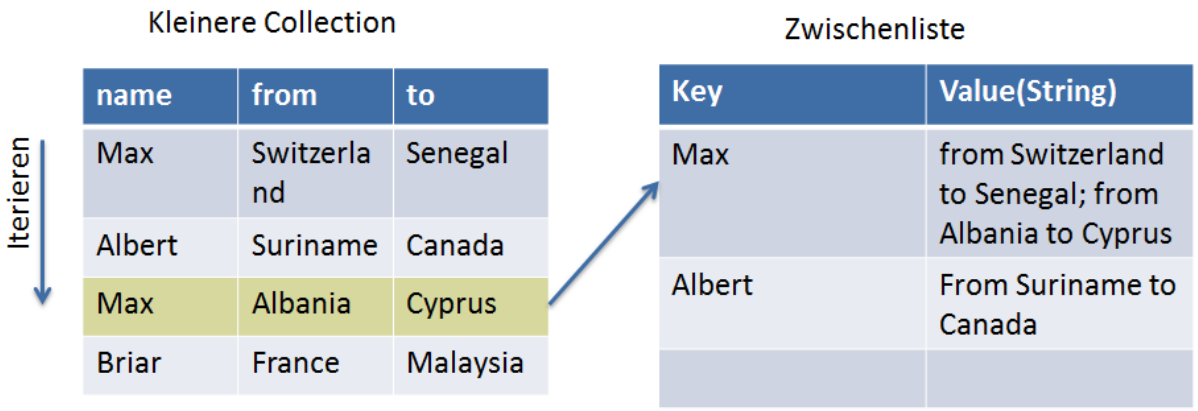


Abbildung 27 - Map Reduce Pattern: Map Phase, Ergänzung

10.3 Reduce-Phase

Nachdem die Zwischenlisten vervollständigt sind, beginnt die Reduce-Phase. In dieser Phase Reduziert man alle einzelnen Zwischenlisten zu einer Gesamtliste. In diesem gezeigten Beispiel verhält sich die Reduce-Phase sehr ähnlich der Map-Phase. Das Iterieren über die Zwischenlisten findet in diesem Beispiel seriell statt.
Falls der Zwischeneintrag (Key) noch nicht in einer Gesamtliste (Key/Value) vorhanden ist, wird dieser Zwischeneintrag hinzugefügt. Andernfalls wird nur der Value erweitert.

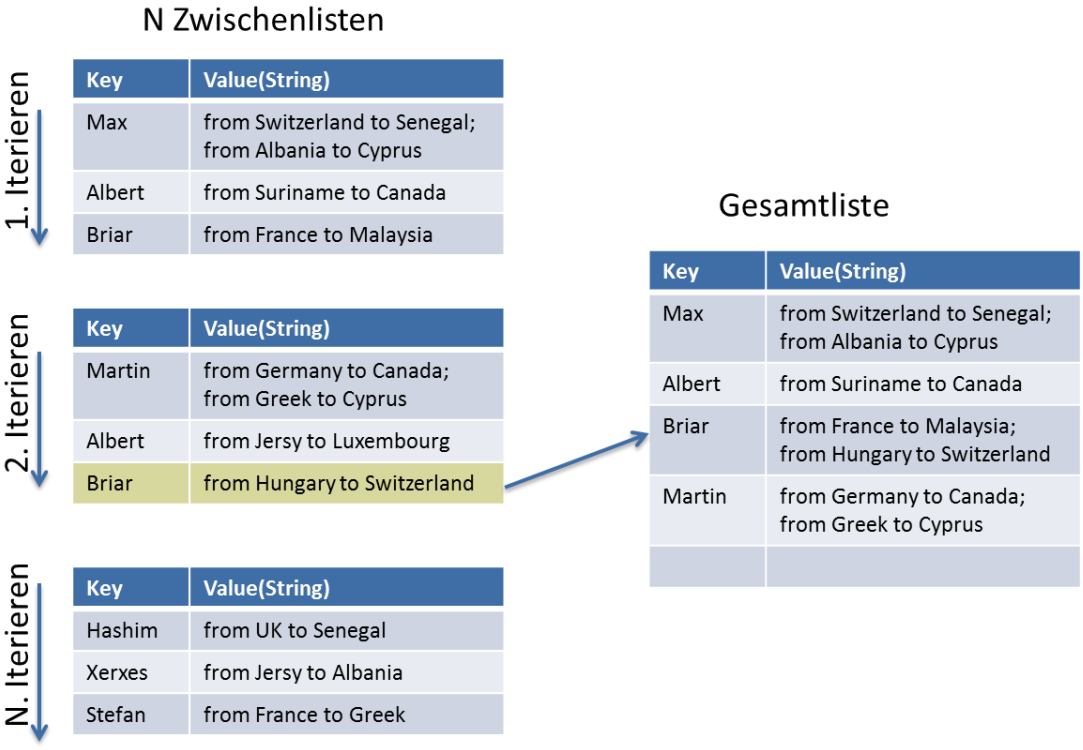


Abbildung 28 - Map Reduce Pattern: Reduce Phase

10.4 Arbeitsfluss

Folgende Grafik zeigt den ganzen Arbeitsfluss. So stellen Abbildung 26 und Abbildung 27 das Feld „Füge Teilinput in einer Map zusammen (map2)“ in untenstehender Grafik in detaillierter Form dar. Ebenso ist Abbildung 28 die umfänglichere Darstellung des Feldes „Fügt alle Maps in Liste zu einer Map zusammen (reduce)“. Felder mit der gleichen Farbe, sind in unserem Map Reduce, Methoden in der gleichen Java-Klasse. Die Wörter in Klammern widerspiegeln die Methodennamen im Code.

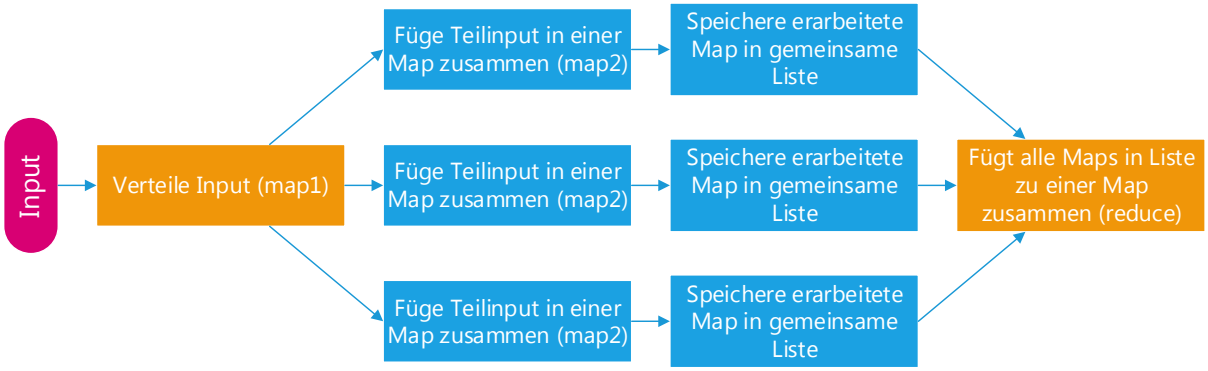


Abbildung 29 - Map Reduce Workflow

10.5 Map Reduce als Code

Für ein besseres Verständnis, wie dies im Code umzusetzen ist, sind folgend die wichtigsten Codefragmente aufgelistet.

10.5.1 Map-Phase

Die AgentList enthält alle Agenten welche sich durch einen Namen, und einen Weg auszeichnen. Der Weg wird aufgespalten in einen Startpunkt und einen Endpunkt. In folgendem Code teilt sich die AgentList in mehrere kleinere Bereiche auf. Diese werden einem WorkerThread zugeteilt. Mit anderen Worten, für jeden Bereich ist ein Thread zuständig.

```
1   static ArrayList<WorkerThread> workerList = new ArrayList<>();
2
3   public void mapFirst() throws InterruptedException{
4     for (int i = 0; i < agentList.size(); i += getThreshold()) {
5       CDAR_WorkerThread wT;
6       if (i + getThreshold() > agentList.size()) {
7         wT = new CDAR_WorkerThread(agentList, i, agentList.size() - 1);
8       } else {
9         wT = new CDAR_WorkerThread(agentList, i, i + getThreshold() - 1);
10      }
11      wT.start();
12      workerList.add(wT);
13    }
14
15    for (CDAR_WorkerThread wT : workerList) {
16      wT.join();
17    }
18  }
```

Quellcode 30 - Map Reduce-Pattern Map-Phase

10.5.2 WorkerThreads Map-Phase

Ein WorkerThread ist ein Thread, welcher für einen kleinen Bereich einer ganzen Liste zuständig ist. Seine Aufgabe besteht darin, diesen Bereich so umzustrukturieren, dass kein Name eines Agents doppelt vorkommt. Da aber die zurückgelegten Strecken der einzelnen Agenten nicht verloren gehen sollen, sind diese zusammenzufassen und durch ein Semikolon zu unterteilen. Nach dem vollenden des Vorganges, speichert der Thread seine Umstrukturierte Map in eine zentrale Liste ab.

```
1  public class CDAR_WorkerThread extends Thread {
2    private HashMap<String, String> agentMap = new HashMap<>();
3    private ArrayList<CDAR_Agent> agentList;
4    private int from,to;
5
6    //each CDAR_WorkerThread works just at a small part from the whole list (from, to)
7    public CDAR_WorkerThread(ArrayList<CDAR_Agent> agentList, int from, int to) {
8      this.agentList = agentList;
9      this.from = from;
10     this.to=to;
11   }
12
13   @Override
14   public void run() {
15     mapSecond();
16     CDAR_MapReduce.addMap(agentMap);
17   }
18
19   //Make keys unique and add Values but it just regard the range (from, to)
20   private void mapSecond() {
21     for(int i=from; i<=to;i++){
22       CDAR_Agent agent = agentList.get(i);
23       if (agentMap.containsKey(agent.getName())) {
24         agentMap.put(agent.getName(), String.format(
             "%s From Location %s to Location %s;\n", agentMap.get(agent.getName()),
             agent.getFromLocation(), agent.getToLocation()));
25       } else {
26         agentMap.put(agent.getName(), String.format(
             "From Location %s to Location %s;\n",
             agent.getFromLocation(), agent.getToLocation()));
27       }
28     }
29   }
30 }
```

Quellcode 31 - Map Reduce-Pattern WorkerThread

10.5.3 Reduce-Phase

In dieser Phase gilt es, alle Maps welche in einer zentralen Liste gespeichert sind zu einer gesamten Map zu vereinen. Ebenfalls wie im vorherigen zweiten Map-Schritt, kommen die Namen der Agenten nur noch einmal vor. Auch hier sind die Wegrouten zusammenzufassen, damit die Reiserouten der Agenten nicht verloren gehen.

```
1   static ArrayList<HashMap<String, String>> agentMaps = new ArrayList<>();
2   static HashMap<String, String> agentMap = new HashMap<>();
3
4   /*
5    * Reduce several Maps to one Map
6    * If Agentname is already in Map, put existing Agentways with new ones together
7    * else put Agentname and way into Map
8   */
9   public HashMap<String, String> reduce() {
10    for (HashMap<String, String> agent : agentMaps) {
11      for (String name : agent.keySet()) {
12        if (getAgentMap().containsKey(name)) {
13          getAgentMap().put(name, String.format("  %s  %s", getAgentMap().get(Name),
                agent.get(name);
14        } else {
15            getAgentMap().put(name, String.format(" %s", agent.get(name)));
16        }
17      }
18    }
19    return getAgentMap();
20  }
```

Quellcode 32 - Map Reduce-Pattern Reduce-Phase

10.6 Endresultat

Die Applikation läuft als Standalone. Diese Applikation basiert auf einem Java Projekt. Nach anschliessendem Deployment entsteht folgende Ausgabe.

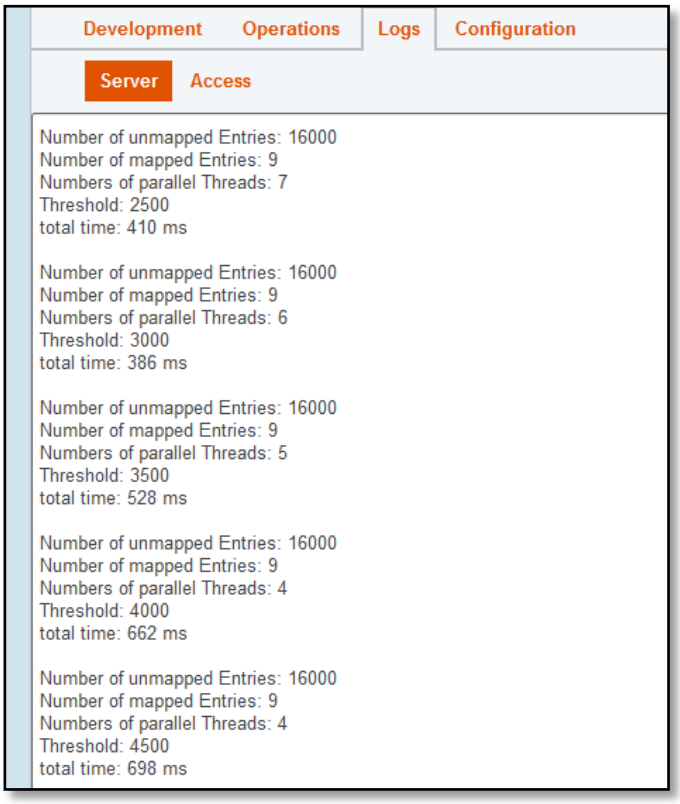


Abbildung 30 - Map Reduce Pattern: Endresultat(CloudBees)

Ein Vergleich mit den Übungsrechner unserer Schule zeigt jedoch, dass die Rechner der Schule performanter sind, als jene der Cloud-Provider

```
Number of unmapped Entries: 16000
Number of mapped Entries: 9
Numbers of parallel Threads: 7
Threshold: 2500
total time: 297 ms

Number of unmapped Entries: 16000
Number of mapped Entries: 9
Numbers of parallel Threads: 6
Threshold: 3000
total time: 365 ms

Number of unmapped Entries: 16000
Number of mapped Entries: 9
Numbers of parallel Threads: 5
Threshold: 3500
total time: 421 ms

Number of unmapped Entries: 16000
Number of mapped Entries: 9
Numbers of parallel Threads: 4
Threshold: 4000
total time: 505 ms

Number of unmapped Entries: 16000
Number of mapped Entries: 9
Numbers of parallel Threads: 4
Threshold: 4500
total time: 605 ms
```

Abbildung 31 - Map Reduce auf Übungsrechner

10.7 Erkenntnis

Durch den Vergleich zeigten wir, dass es sich für uns nicht lohnen würde, diese Applikation aus Performancegründen in der Cloud auszuführen. Zudem verbraucht die Applikation lokal, wie auch in der Cloud, erhebliche Ressourcen. Dies führt bei grösseren Applikationen dieser Art zu zusätzliche Kosten.

11 Rechenintensive Operationen in der Cloud

Diese Applikation zeigt dem Entwickler, dass sich nicht alle Methoden in der Cloud gleich verhalten wie lokal. Es gibt Methoden, die in der Cloud verhältnismässig sehr viel mehr Rechenzeit benötigen.

11.1 Endresultat

Die erstellte Applikation führt mehrere Methoden aus und zeigt danach jeweils die Ausführungszeit an. Das Resultat mit den im Rahmen dieser Studienarbeit getesteten Methoden zeigt, dass die Ausführung der SecureRandom-Methode nach dem Deployment auf einen PaaS-Anbieter mehr Zeit beansprucht als lokal.

Lokal:

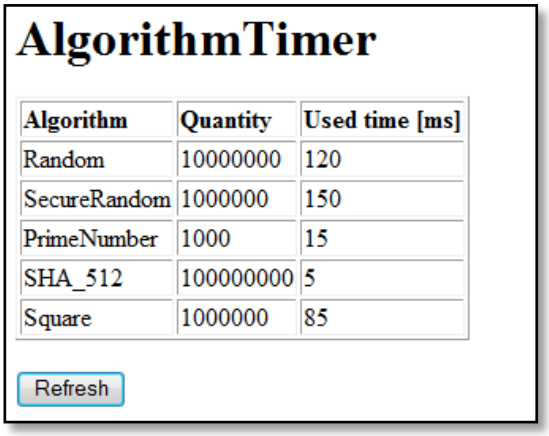


Abbildung 32 - Algorithm Timer lokal

CloudBees:

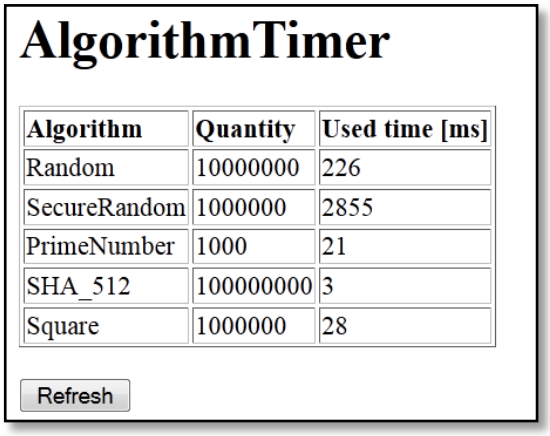


Abbildung 33 - Algorithm Timer – CloudBees

Google App Engine:

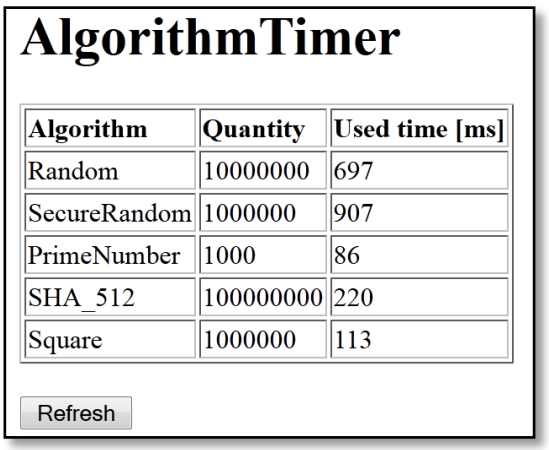


Abbildung 34 - AlgorithmTimer - Google App Engine

Heroku:

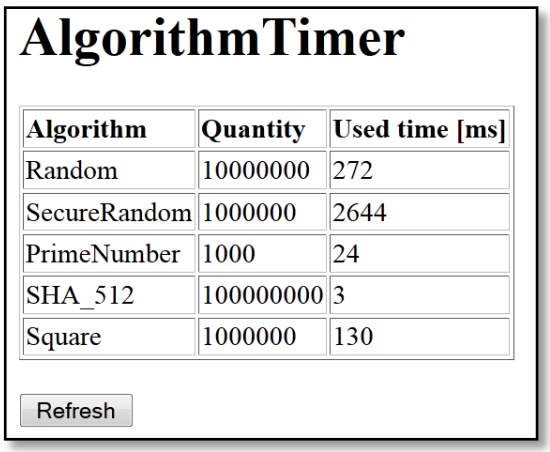


Abbildung 35 - AlgorithmTimer - Heroku

11.2 Applikationsaufbau

Der Applikationsaufbau ist sehr simpel gehalten. Die Struktur ist so aufgebaut, dass leicht weitere Algorithmen hinzuzufügen sind. Bei der Applikation wurden keine Optimierungen vorgenommen. Die Zeitmessungen sind an die Applikationsaufrufe gebunden und liefern bei einem weiteren Aufruf andere Resultate. Dieses Projekt implementiert das Template Method Pattern. Allerdings ist der Aufruf der Methode direkt im Konstruktor ausgelagert. Die Funktionalität dieses Projektes setzt dies so voraus.

11.2.1 Gemeinsame abstrakte Algorithmusklasse

Das Projekt besteht neben einem Servlet, welches für die ganze HTML-Darstellung zuständig ist, auch aus einer gemeinsamen abstrakten Klasse, von welcher alle Algorithmen erben. Sie berechnet die für den Algorithmus benötigte Zeit und dient als Basiskonstrukt für die Algorithmen.

```
1   /*
2    * abstract Class for all algorithm classes
3    */
4   public abstract class CDAR_CPUAlgorithm{
5     /*
6      * attribute:
7      * algorithm --> name of the algorithm e.g. SecureRandom
8      * quantity --> quantity of loops e.g. 100000
9      * totalTime --> needed time for process in [ms] e.g. 500
10     */
11    private String algorithm;
12    private int quantity;
13    private long totalTime;
14
15    public CDAR_CPUAlgorithm(int quantity){
16      this.setAlgorithm(this.getClass().getSimpleName().substring(5));
17      this.setQuantity(quantity);
18
19          long startTime = System.currentTimeMillis();
20          executeAlgorithm();
21          setTotalTime(System.currentTimeMillis() - startTime);
22    }
23
24    private void setQuantity(int quantity) {
25      this.quantity = quantity;
26    }
27
28    private void setAlgorithm(String algorithm) {
29      this.algorithm = algorithm;
30    }
31
32    protected abstract void executeAlgorithm();
33
34    protected void setTotalTime(long totalTime) {
35        this.totalTime = totalTime;
36    }
37
38    public String getAlgorithm() {
39      return algorithm;
40    }
41
42    public long getTotalTime() {
43      return totalTime;
44    }
45
46    public int getQuantity() {
47     return quantity;
48    }
49  }
```

Quellcode 33 – abstrakte Algorithmenklasse

11.2.2 Algorithmusklasse

Der jeweilige Algorithmus wird in einer eigenen Klasse ausgeführt. Diese Klasse erbt die Grundzüge der abstrakten Algorithmusklasse. Deshalb sind in dieser Klasse nur noch der eigentliche Algorithmus und die Anzahl der Durchläufe zu implementieren. Folgender Beispielcode zeigt die Implementation einer solchen Klasse. Diese Klasse verwendet den Algorithmus „Secure Random“.

```
1   /*
2    * this class generate a given quantity of securerandom integers
3    * secure random is a cryptographically strong subclass of random
4    */
5   public class CDAR_SecureRandom extends CDAR_CPUAlgorithm{
6
7     public CDAR_ SecureRandom(int quantity){
8           super(quantity);
9     }
10
11    @Override
12    protected void executeAlgorithm() {
13      java.security.SecureRandom sR = new java.security.SecureRandom();
14      for (int i = 0; i < getQuantity(); i++){
15        sR.nextInt();
16      }
17    }
18  }
```

Quellcode 34 –Algorithmenklasse

11.3 Erkenntnis

Wie obenstehende Screenshots zeigen, benötigen einzelne Algorithmen in der Cloud massiv länger. Ebenfalls ist überraschend, dass sich Google App Engine ganz aus dem Muster der Clouds differenziert. Deshalb ist darauf zu achten, solche Algorithmen, welche in der Cloud nicht gleich performant wie lokal sind, zu vermeiden. Diese Algorithmen beeinträchtigen den normalen Ablauf von Programmen beträchtlich und stellen einen unnötigen Flaschenhals dar.

Teil II

Anleitungen

12 Anleitungen Heroku

12.1 Heroku Toolbelt SDK

Durch die Nutzung des Heroku Toolbelt SDK's kann mit wenig Aufwand über die Kommandozeile eine Applikation in Heroku erstellt und deployed werden [Too13] [Get131].

12.1.1 Vorbedingung

- Useraccount auf heroku.com

12.1.2 Vorbereitung

Der Link zum Download findet man in der „Getting Started" Dokumentation von Heroku oder unter folgendem Link: https://toolbelt.heroku.com/

12.1.3 Installation

Installieren des Heroku Toolbelts.
Die Installation umfasst neben dem Command-Line Tool von Heroku zusätzlich noch Foreman, mit welchem man die Applikation lokal laufen lassen kann. Zudem wird git installiert, was für den Push auf Heroku benötigt wird.

12.1.4 Login

Durch den Befehl „heroku login" kann man sich bei Heroku mit seinen User-Credentials anmelden.

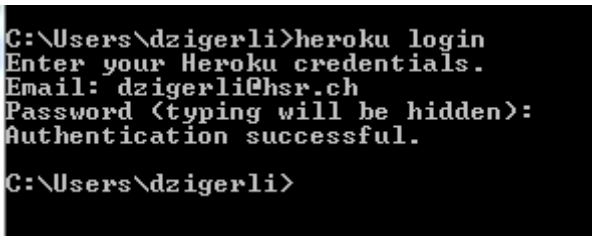


Abbildung 36 - Heroku Toolbelt Login

12.1.5 Lokales Git Repository erstellen

Befindet man sich nun in einem Verzeichnis eines Projektes, wird zuerst ein lokales Git Repository erstellt. Danach müssen noch sämtliche Dateien hinzugefügt und commitet werden, dies erfolgt mit folgenden Befehlen:

```
1  git init
2  git add .
3  git commit -m "init"
```

Quellcode 35 - lokales Git Repository

12.1.6 Applikation erstellen und deployen

Mit dem Befehl *heroku create* eine neue Heroku Applikation erstellen. Anschliessend mit folgenden Befehl den lokalen Master Branch deployen.

```
1  git push heroku master
```

12.1.7 Bestehende Applikation updaten

Mit folgenden Befehl den bestehenden Remote Branch dem lokalen Git hinzufügen:

```
1  heroku git:remote -a APPLIKATIONS_NAME
```

Beispiel:

```
1  heroku git:remote -a falling-wind-1342
```

Überprüfen der Remote-Konfiguration ist mit folgendem Befehl möglich:

```
1  heroku remote -v
```

Danach ist ein Push des lokalen Branches mit gewohntem Befehl möglich:

```
1  git push heroku master
```

12.2 Sample App Heroku mit Eclipse

Dies ist eine Anleitung für die Nutzung von Heroku mit Eclipse und dem Heroku-Plugin.

12.2.1 Vorbereitung

1. Damit Heroku zusammen mit Eclipse gebraucht werden kann, muss ein Heroku-Account (www.heroku.com) erstellt werden. Zusätzlich muss eine App im Heroku-Account erzeugt werden. Bei der Region sollte Europe ausgewählt werden, damit wird eine schnellere Verbindung garantiert.

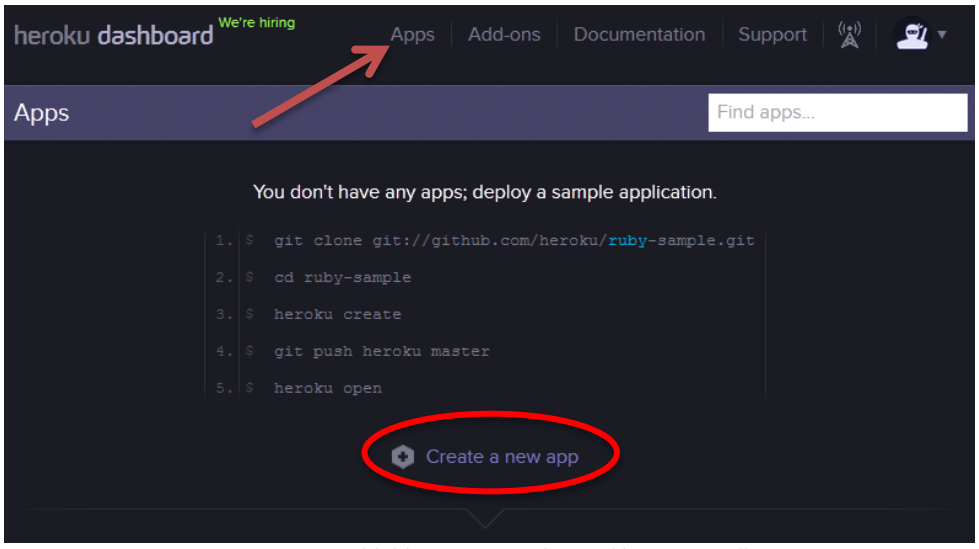


Abbildung 37 - Heroku Applikation erstellen

2. Zudem wird die EE Developer Version von Eclipse benötigt.

12.2.2 Installation der Software

1. Zuerst muss sichergestellt werden, dass JDK installiert ist.
2. Installieren des Heroku-Plugins in Eclipse
   a. Help > Install New Software

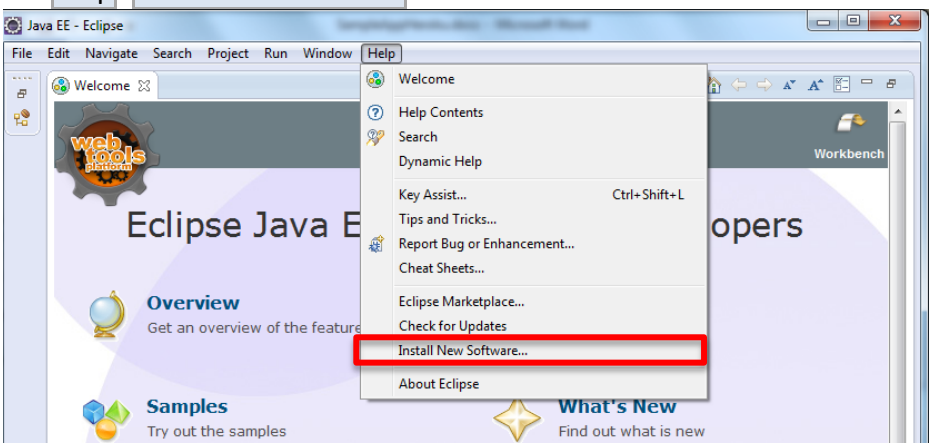


Abbildung 38 - Eclipse Install New Software

   b. Klicke auf Add um den Plugin-URL hinzuzufügen

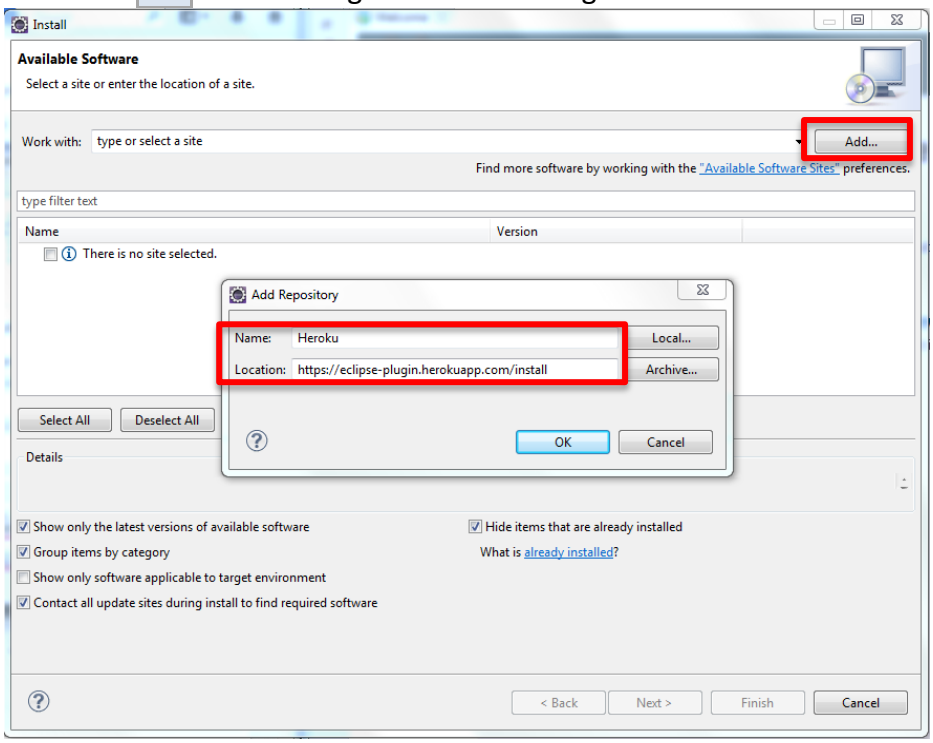


Abbildung 39 - Eclipse Repository

   c. Die URL *https://eclipse-plugin.herokuapp.com/install* bei der Location hinzufügen und einen Namen eingeben. Danach OK drücken.
   d. Das neu erschienene Plugin (Heroku Eclipse Integration) auswählen und installieren.

12.2.3 Konfiguration der Software

1. API-Schlüssel konfiguration
   a. Window>Preferences>Heroku
   b. E-Mail und Passwort des vorhin erstellten Heroku-Account eintragen. Danach auf Login klicken. Zur Sicherstellung kann der erzeugte API-Schlüssel noch validiert werden. Anschliessend auf Apply und OK.

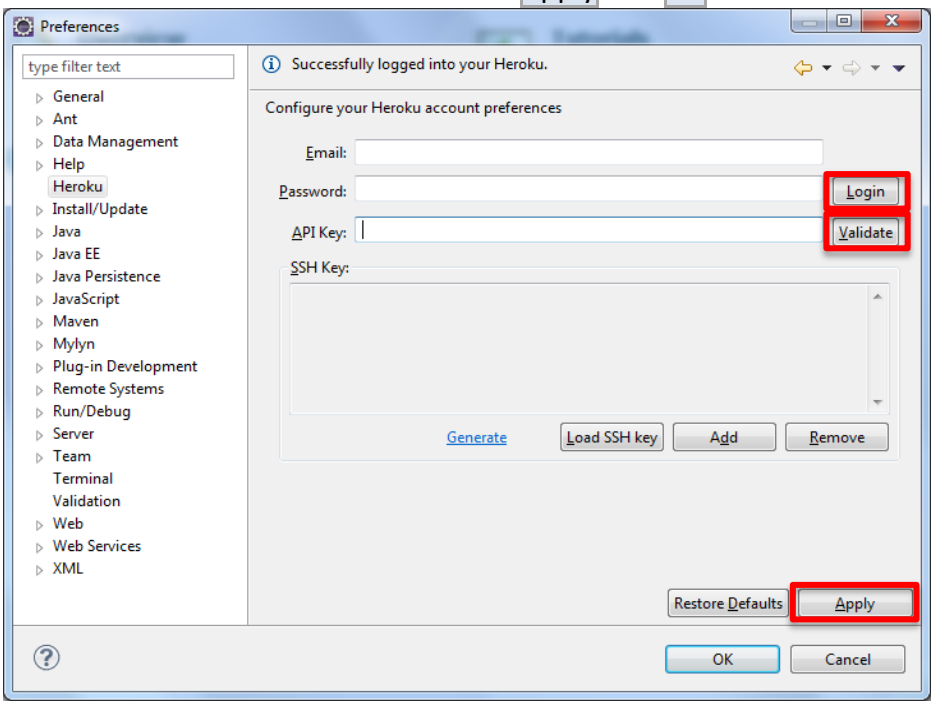


Abbildung 40 - Eclipse Heroku Konfiguration

2. SSH-Schlüssel Erzeugung
   a. Window>Preferences>General> Network Connections>SSH2
   b. Anschliessend im Reiter Key Management>Generate RSA Key…
   c. Nach dem Generieren des Schlüssels diesen speichern mit Save Private Key
   d. Danach auf Apply und OK

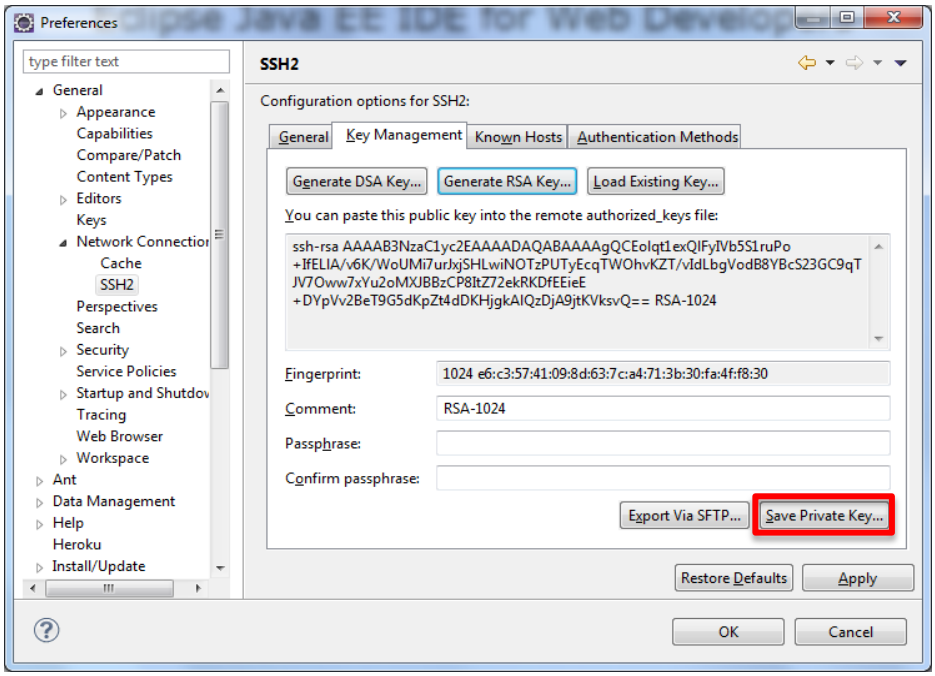


Abbildung 41 - Eclipse SSH Keys

3. Hinzufügen des SSH-Schlüsses in Heroku
   a. In diesem Schritt in das Verzeichnis Window>Preferences>Heroku wechseln um den vorhin gesicherten SSH-Schlüssel hinzuzufügen.
   b. Anschliessend auf Load SSH key klicken und den Schlüssel auswählen.
   c. Danach auf Add um den Schlüssel in Heroku anzulegen. Zum Schluss Apply und OK um den Vorgang abzuschliessen.

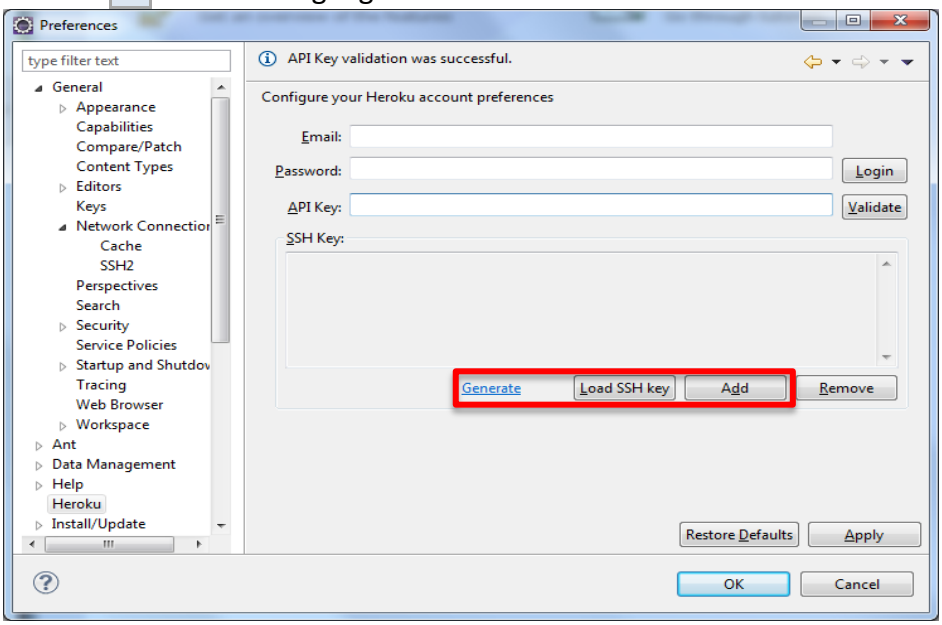


Abbildung 42 - Heroku SSH Key Add

4. Nun fehlt jedoch noch das „known_hosts“ File damit man auf das Heroku-Repository mit Eclipse zugreifen kann. Relativ einfach kann dies über ein Git-Import geschehen.
   a. File>Import
   b. Git>Projects from Git

c. URI auswählen und weiter
d. Im folgendem Fenster ist der Git URL der zu Begin erstellten App einzutragen. Diesen findet man im Heroku-Account. Danach auf Next klicken.

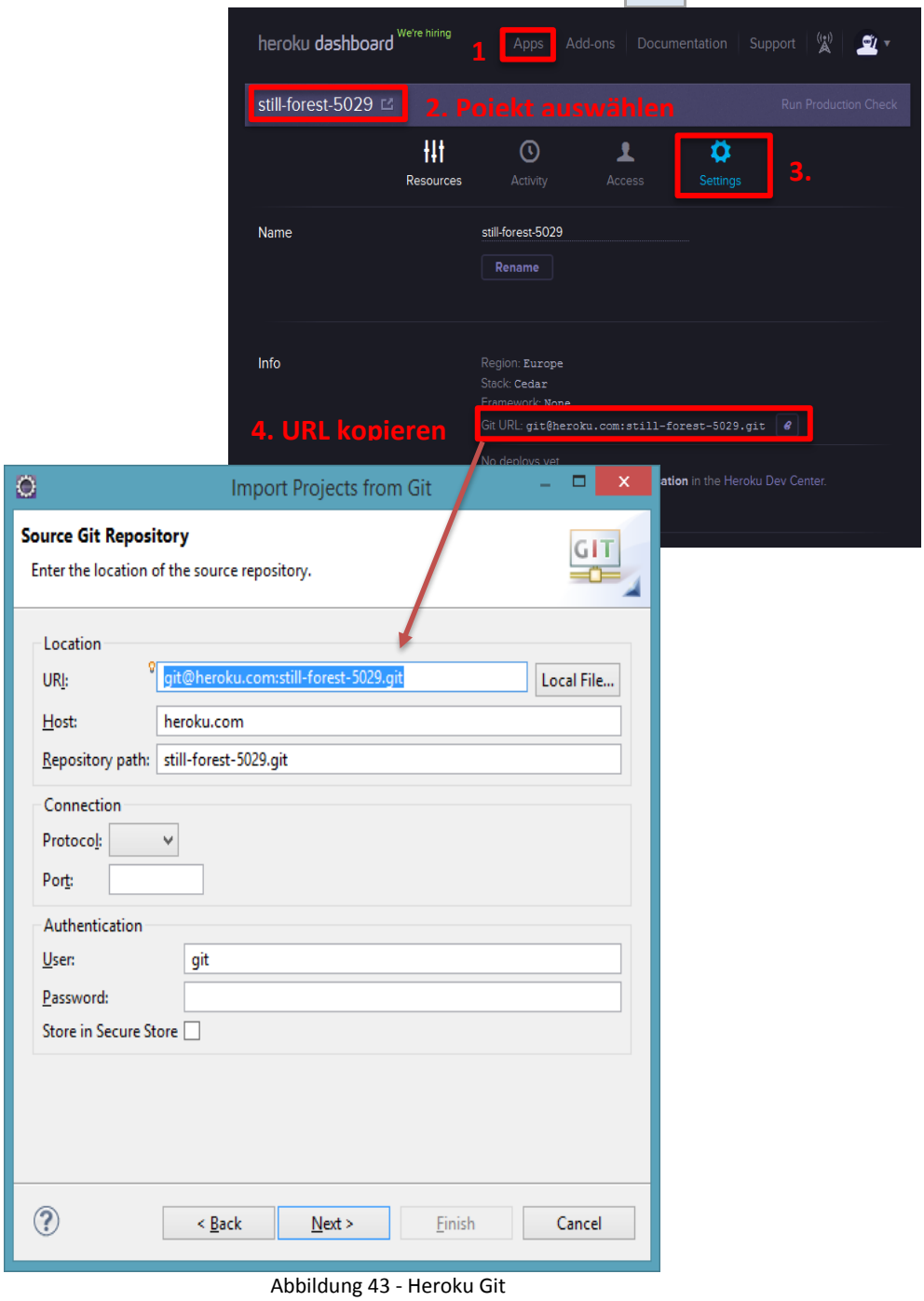


Abbildung 43 - Heroku Git

e. Anschliessend kommt folgender Dialog. Auf Yes klicken.

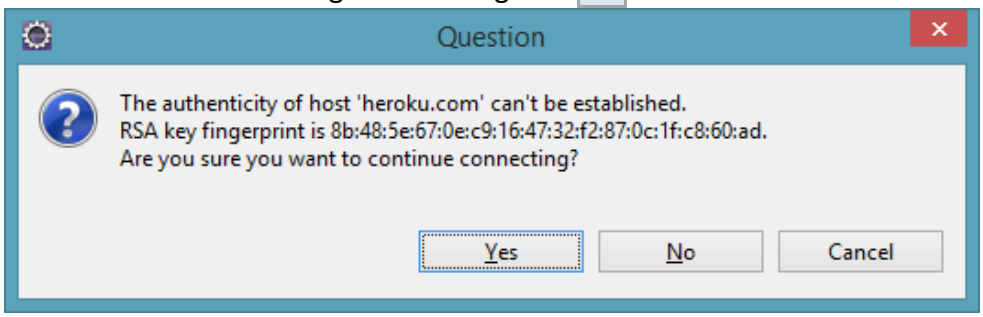


Abbildung 44 - Heroku RSA

f. Im nachstehenden Dialog ist das „known_hosts“-File zu erstellen. Deshalb auf, Yes klicken.

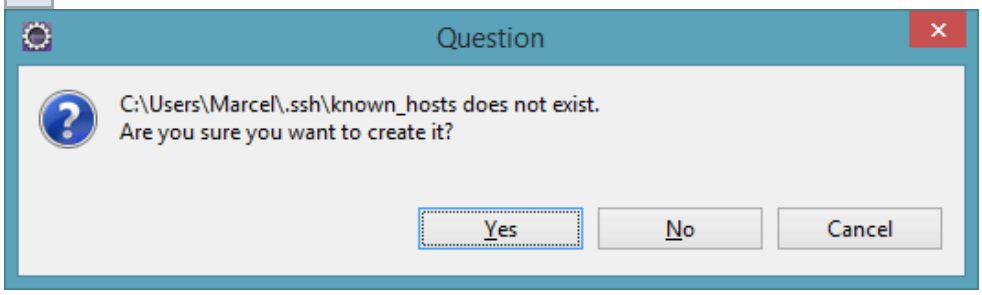


Abbildung 45 - Heroku Known Hosts

5. Anschliessend erscheint folgendes Fenster. Da jetzt das File vorhanden ist, kann der Wizard abgebrochen werden-> Cancel

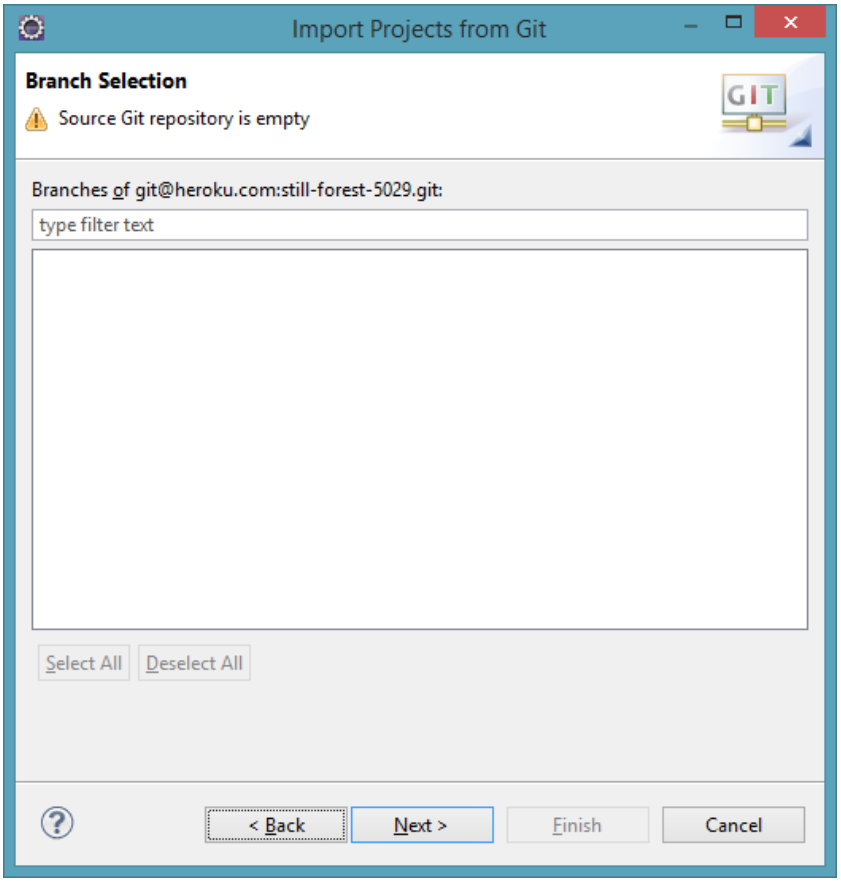


Abbildung 46 - Git Import

6. Da jetzt alle Vorbereitungen getroffen sind, kann man die App von Heroku nach Eclipse importieren.
   a. File>Import
   b. Heroku>Import existing Heroku App>Next
   c. Projekt auswählen und Next klicken
   d. Auto detected Project>Finish

7. Erstellen eines Herokuprojekts in Eclipse
   a. File>New>Other
   b. Heroku>Create Heroku App from Template
   c. Auswahl des Templates und danach auf Finish klicken.

12.2.4 Häufige Probleme

- Tritt auf wenn Punkt 4 vergessen oder falsch gemacht wurde.

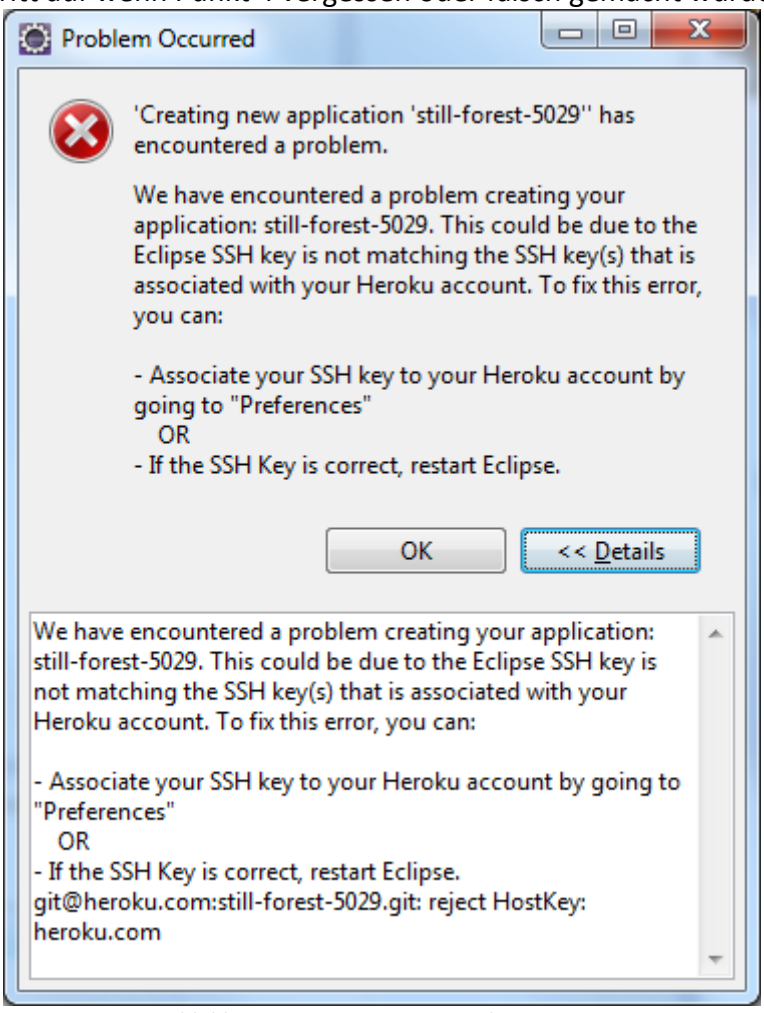


Abbildung 47 - SSH Key missmatch

- Tritt auf, wenn bereits die maximalen fünf Freeapps in Heroku existieren. Ein App bei Heroku löschen, danach funktioniert es wieder.

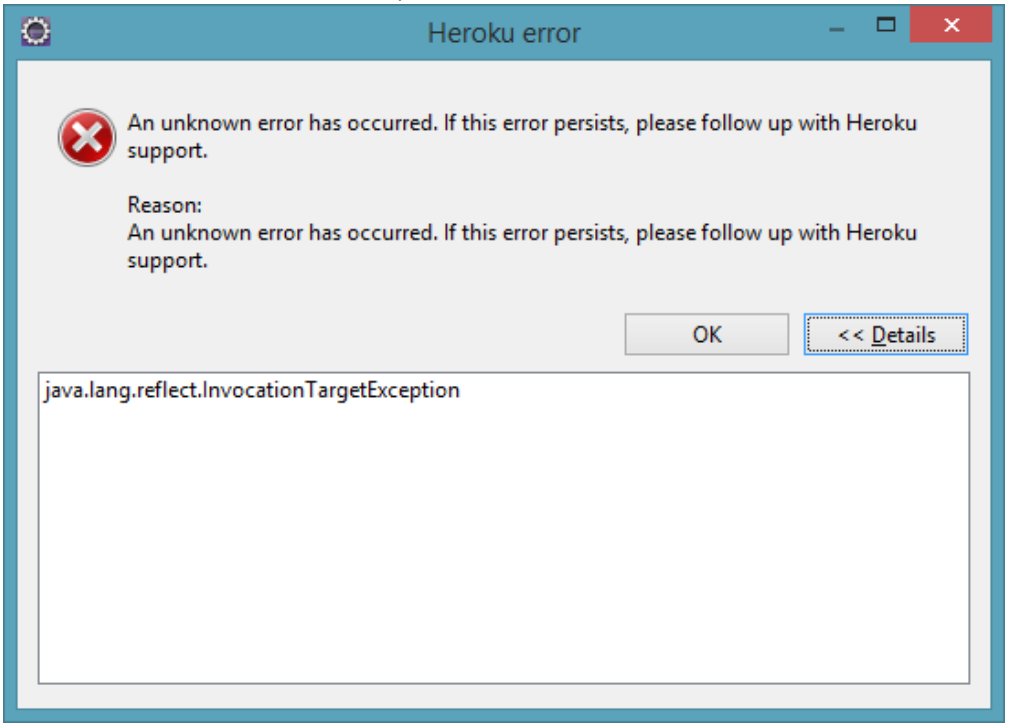


Abbildung 48 - Heroku TargetException

- Tritt auf, wenn Punkt 3.c vergessen wurde.

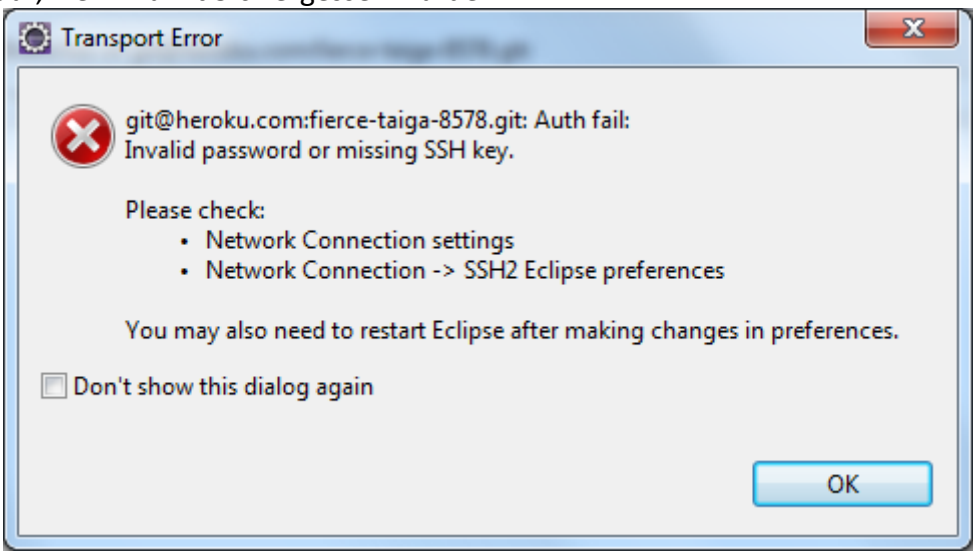


Abbildung 49 - Heroku SSH Problem

12.3 Heroku First Servlet

Diese Anleitung dient zum Erstellen eines einfachen Servlets auf Heroku. Ausserdem werden mögliche Fehler sowie deren Fehlerbehandlung erläutert [Cre13].

12.3.1 Vorbedingung

- Anleitung zu Heroku Toolbelt SDK durchgeführt bzw. als Referenz nehmen.
- EclipseEE mit Maven bzw. Maven installiert (siehe „15.1 Installationsanleitung Maven“)

12.3.2 Vorbereitung

Zuerst die notwendige Struktur der Applikation erstellen. Dazu wird ein Java-Projekt mit folgenden Ordnern erstellt. Auf die Namensvergabe sollte geachtet werden, da später auf einige Ordner referenziert wird.

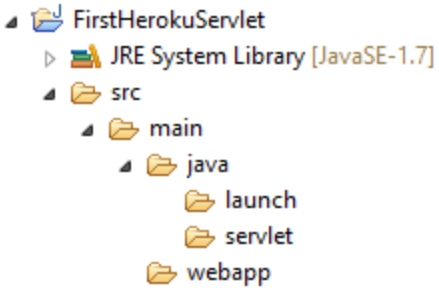


Abbildung 50 - Heroku Servlet Ordnerstruktur

12.3.3 Erstellen der Dateien

1. Im Ordern „launch“ eine Javaklasse erstellen mit folgendem Inhalt:

```
1  package launch;
2  import java.io.File;
3  import org.apache.catalina.startup.Tomcat;
4
5  public class Main {
6    public static void main(String[] args) throws Exception {
7
8      String webappDirLocation = "src/main/webapp/";
9      Tomcat tomcat = new Tomcat();
10
11     String webPort = System.getenv("PORT");
12     if(webPort == null || webPort.isEmpty()) {
13       webPort = "8080";
14     }
15
16     tomcat.setPort(Integer.valueOf(webPort));
17
18     tomcat.addWebapp("/",new File(webappDirLocation).getAbsolutePath());
19     System.out.println("configuring app with basedir: "
20 + new File("./" + webappDirLocation).getAbsolutePath());
21
22     tomcat.start();
23     tomcat.getServer().await();
24   }
25 }
```

Quellcode 36 - Embedded Tomcat

2. Klasse „HelloServlet“ mit folgendem Inhalt im Ordner „servlet” anlegen.

```
package servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(
  name = "MyServlet",
  urlPatterns = {"/hello"}
)
public class HelloServlet extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    ServletOutputStream out = resp.getOutputStream();
    out.write("hello heroku".getBytes());
    out.flush();
    out.close();
  }
}
```

Quellcode 37 - HelloServlet (heroku)

3. Der Ordner webapp erstellen und die darin enthaltene Datei „index.jsp” mit folgendem Inhalt ergänzen.

```
<jsp:forward page="/hello" />
```

4. Anschliessend die pom.xml Datei erstellen, welche folgenden Inhalt besitzt. Die Datei im root Verzeichnis des Projektes ablegen.

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSc
   hema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/maven-v4_0_0.xsd">
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>com.heroku.sample</groupId>
4    <artifactId>embeddedTomcatSample</artifactId>
5    <version>1.0-SNAPSHOT</version>
6    <name>embeddedTomcatSample Maven Webapp</name>
7    <url>http://maven.apache.org</url>
8    <properties>
9      <tomcat.version>7.0.34</tomcat.version>
10   </properties>
11   <dependencies>
12     <dependency>
13       <groupId>org.apache.tomcat.embed</groupId>
14       <artifactId>tomcat-embed-core</artifactId>
15       <version>${tomcat.version}</version>
16     </dependency>
17     <dependency>
18       <groupId>org.apache.tomcat.embed</groupId>
19       <artifactId>tomcat-embed-logging-juli</artifactId>
20       <version>${tomcat.version}</version>
21     </dependency>
22      <dependency>
23        <groupId>org.apache.tomcat.embed</groupId>
24        <artifactId>tomcat-embed-jasper</artifactId>
25        <version>${tomcat.version}</version>
26     </dependency>
27     <dependency>
28       <groupId>org.apache.tomcat</groupId>
29       <artifactId>tomcat-jasper</artifactId>
30       <version>${tomcat.version}</version>
31     </dependency>
32     <dependency>
33       <groupId>org.apache.tomcat</groupId>
34       <artifactId>tomcat-jasper-el</artifactId>
35       <version>${tomcat.version}</version>
36     </dependency>
37     <dependency>
38       <groupId>org.apache.tomcat</groupId>
39       <artifactId>tomcat-jsp-api</artifactId>
40       <version>${tomcat.version}</version>
41     </dependency>
42   </dependencies>
43   <build>
44     <finalName>embeddedTomcatSample</finalName>
45     <plugins>
46       <plugin>
47         <groupId>org.codehaus.mojo</groupId>
48         <artifactId>appassembler-maven-plugin</artifactId>
49         <version>1.1.1</version>
50         <configuration>
51           <assembleDirectory>target</assembleDirectory>
52           <programs>
53             <program>
54               <mainClass>launch.Main</mainClass>
55               <name>webapp</name>
56             </program>
57           </programs>
58         </configuration>
59         <executions>
60           <execution>
61             <phase>package</phase>
62             <goals>
63               <goal>assemble</goal>
64             </goals>
65           </execution>
66         </executions>
67       </plugin>
68     </plugins>
69   </build>
70 </project>
```

Quellcode 38 - Heroku First Servlet pom.xml

5. Ebenfalls direkt im Projekt muss das Procfile erstellt werden. Es besitzt keine Dateierweiterung.

```
1  web: sh target/bin/webapp
```

6. Zum Schluss, ebenfalls im Root-Ordner die Datei system.properties erstellen.

```
1  java.runtime.version=1.6
```

Das Projekt besitzt nun folgende Struktur.

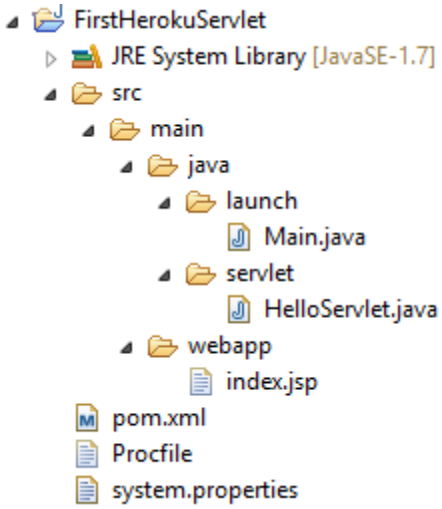


Abbildung 51 - Heroku Projektstruktur

12.3.4 Deployment

Um das Projekt zu deployen, muss es zuerst in ein Mavenprojekt konvertiert werden. Dazu gibt es zwei mögliche Varianten.

1 Die geschieht über Rechtsklick auf das Projekt `Configure`→`Convert to Maven Project`.

2 Das Projekt als „File System“ exportieren. Danach mit der Console in das Projektverzeichnis navigieren. Anschliessen den Befehl `mvn package` ausführen.

Danach kann das Projekt wie in der Anleitung Heroku Toolbelt SDK deployt werden.

12.3.5 Mögliche Fehler

12.3.5.1 Diamond operator is not supported in -source 1.5

Dazu müssen folgende zwei Dateien angepasst werden [Mav13].

1 Die Datei pom.xml mit dem folgende Inhalt ergänzen bzw. erweitern:

```
<properties>
  <tomcat.version>7.0.34</tomcat.version>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
</properties>
```

Quellcode 39 - maven tomcat

2 In der Datei system.properties muss die Versionsnummer auf 1.7 gestellt werden.

```
java.runtime.version=1.7
```

12.3.5.2 Failed to execute goal org.apache.maven.plugins…

```
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:2.3.2:compile (default-compile) on project embeddedTomcatSample: Compilation failure
[ERROR] Failure executing javac, but could not parse the error:
[ERROR] javac: invalid target release: 1.7
[ERROR] Usage: javac <options> <source files>
[ERROR] use -help for a list of possible options
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
!     Failed to build app with Maven
```

Abbildung 52 - Heroku Maven Error

Hierzu muss die Version in der Datei „system.properties“ auf 1.7 umgestellt werden.

```
java.runtime.version=1.7
```

13 Anleitungen CloudBees

13.1 War-File Deployment

Anleitung für die Nutzung von CloudBees mit dem WebGui und dem Bees SDK.

13.1.1 Vorbereitung

1. Auf CloudBees (http://www.cloudbees.com) einen Account anlegen.

13.1.2 Nutzung via WebGui

Vorbedingung: Exportiertes War-File einer Applikation sind im Kapitel „15.3 DDDSample Projektimport“ ausführlicher behandelt.

- Login auf der CloudBees Webseite
- Am oberen Bildschirmrand befindet sich die Navigation. In dieser auf den Apps Reiter klicken.
- Auf der linken Seite kann die Region ausgewählt werden, zur Verfügung stehen:
  - CloudBees: US
  - CloudBees: EU

  Bei unseren Tests konnten keine markanten Performanceunterschiede festgestellt werden.
- Über den Button „*Create Application*“ kann eine neue Applikation hinzugefügt werden. Bei diesem Anbieter darf der Name der Applikation nicht leer gelassen werden.
- Nachdem die Applikation auf dem Server erstellt worden ist, kann mittels des Buttons „Alternatively, upload a WAR file“ das gewünschte War-File auf den Server geladen werden.

13.1.3 Deployment via SDK

13.1.3.1 Projekt erstellen

- Nachdem im CloudBees Account eine neue Applikation angelegt wurde, erscheint ein Link zum Bees SDK (http://wiki.cloudbees.com/bin/view/RUN/BeesSDK).
- Auf dieser Seite findet sich der Download zum BeesSDK.
- Nach dem Entpacken in einen Ordner (keine Leerzeichen im Ordner-Pfad) kann die Konsole mit einem Klick auf die „Bees Console“-Verknüpfung gestartet werden.
- Mittels *bees create <name>* wird ein neues Projekt angelegt
- Nach einer kurzen Wartezeit wird man nach der gewünschten Region gefragt, danach müssen noch die Credentials als Überprüfung eingegeben werden
- Nun wurde im SDK-Verzeichnis ein Ordner mit dem Namen des neuen Projektes erstellt. Mit „cd <name>“ in dieses Verzeichnis wechseln und mit „bees run“ das Projekt ausführen.
    - Lokal kann man die momentan noch leere Applikation im Browser unter localhost:8080 anschauen.

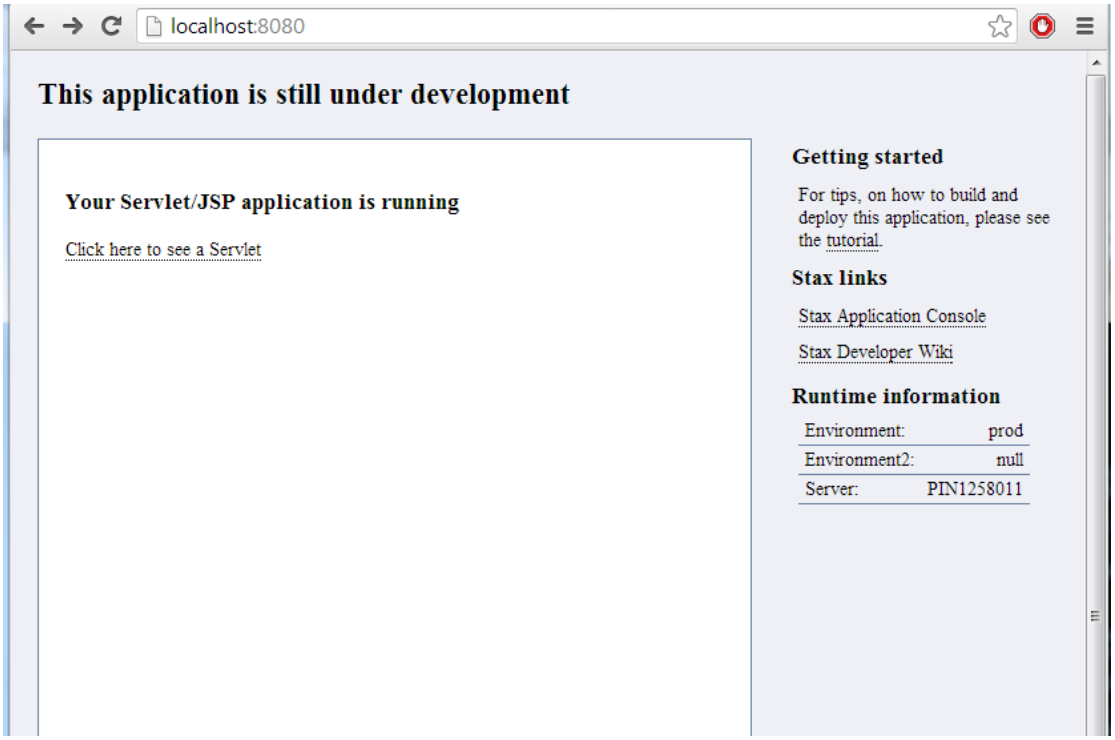


Abbildung 53 - CloudBees localhost

- Dieses Projekt kann mit den gewünschten Tools (bswp. Eclipse) bearbeitet werden und falls gewünscht, mit folgendem Befehl deployed werden: *bees deploy –a <accountname>/<projektname>*

13.1.3.2 War-Datei deployen

- Mit der Bees-Console ins Verzeichnis mit der gewünschte War-Datei wechseln.
- Datei mit folgendem Befehl deployen:

```
bees app:deploy dddsample.war -a <accountname>/<projektname>
```

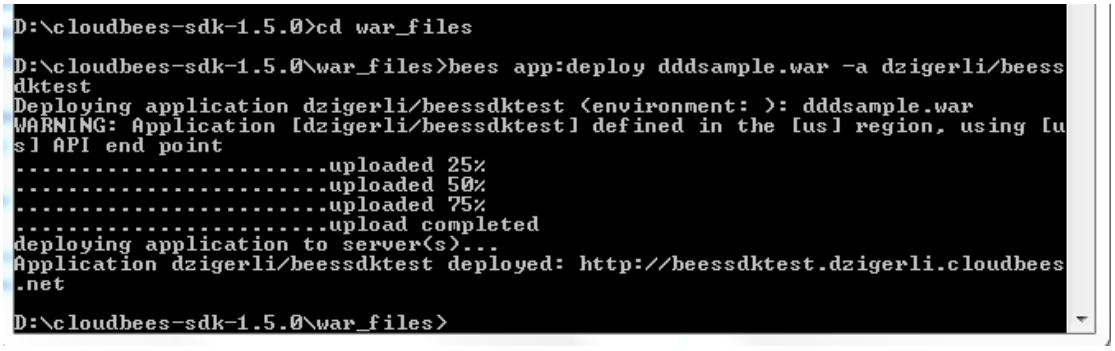
- 

Abbildung 54 - CloudBees War-Deployment

Dieser Befehl kann auf einer leeren, wie auch einer bestehenden Applikation benutzt werden.

13.2 Jar-Datei deployen über SDK

Nachdem man das Projekt in Eclipse gestartet hat, kann mit einem Rechtsklick auf Export, die Datei als ausführbare Jar-Datei exportiert werden. Vor dem Export muss die gewünschte Applikation ausgewählt werden.

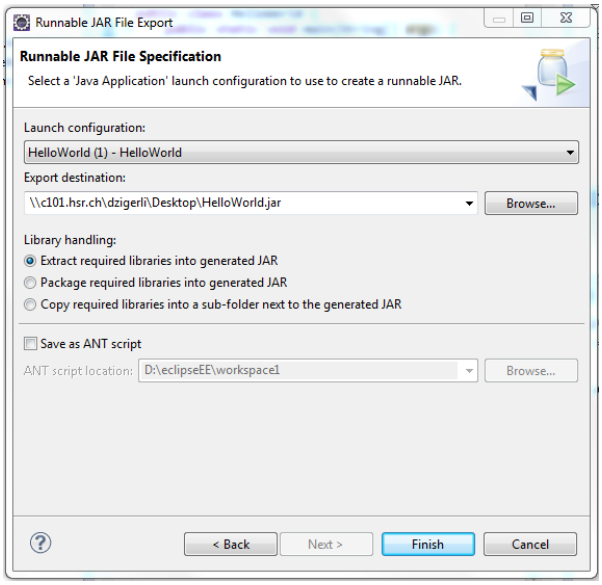


Abbildung 55 - Ausführbare Jar-Datei

13.2.1 Deployment

Über das CloudBees-SDK kann das Package mit folgendem Befehl deployed werden [JVM13]:

```
1  bees app:deploy –t java –R class=<packages>.<applicationname> –R java_version=1.7 <path
   zum Jar-File>/<Jar-Name>.jar
```

Die Ausgabe kann nun entweder über das WebUI von CloudBees oder über das SDK mit dem Befehl *`bees app:tail APP_ID`* angesehen werden.

13.2.2 Aufgetretene Probleme

13.2.2.1 Apps klicken in CloudBees

Falls die Meldung „Please subscribe to the service before accessing it.” erscheint, zuerst Subscribe bei der Spalte Free auswählen.

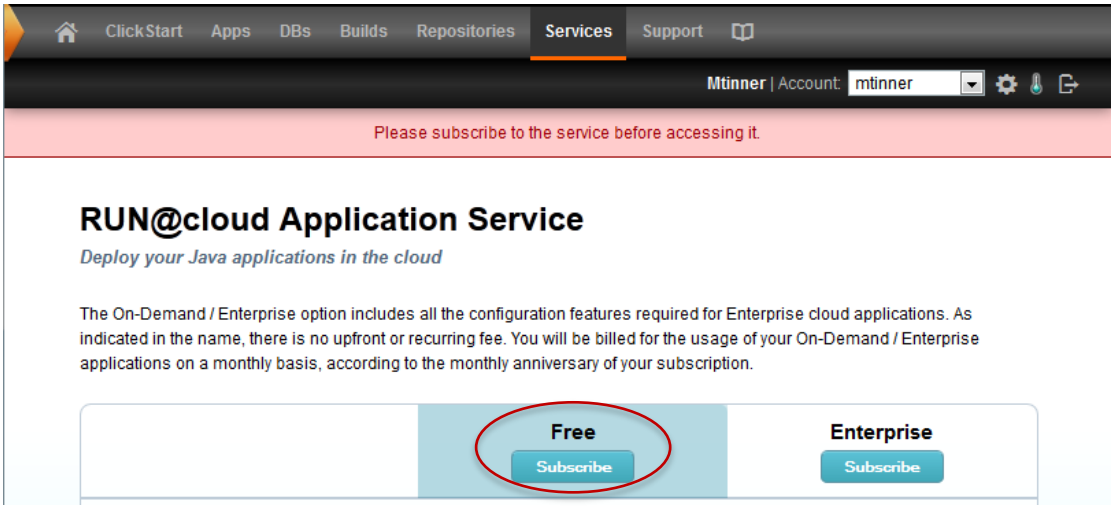


Abbildung 56 - CloudBees Application Service

13.2.2.2 Region-Auswahl im Eclipse Plugin

Im CloudBees-EclipsePlugin gibt es die Möglichkeit, die gewünschte Region auszuwählen. Nachdem aber Region EU ausgewählt und eine Applikation deployed wurde, stellte sich heraus, dass die Applikation schlussendlich trotzdem auf einen Server mit Sitz in der USA deployed wurde.

13.2.2.3 Falsches Verzeichnis

Falls folgende Fehlermeldung beim Befehl „bees run“ auftritt, befindet man sich im falschen Verzeichnis.

Abbildung 57 - CloudBees Verzeichnis error

13.3 MySQL Einrichtung

Folgende Anleitung zeigt das Erstellen und auslesen der Informationen einer MySQL Datenbank bei CloudBees.

13.3.1 Erstellen einer MySQL Datenbank

1. Nach dem Anmelden bei CloudBees steht der Reiter „DBs“ zur Verfügung, auf diesen klicken.
2. Anschliessend auf der rechten Seite auf „Add New Database“ klicken.
3. Danach erscheint das Fenster welches in folgendem Bild gezeigt wird. Alle Textfelder sind auszufüllen und anschliessend auf „Finish“ klicken.

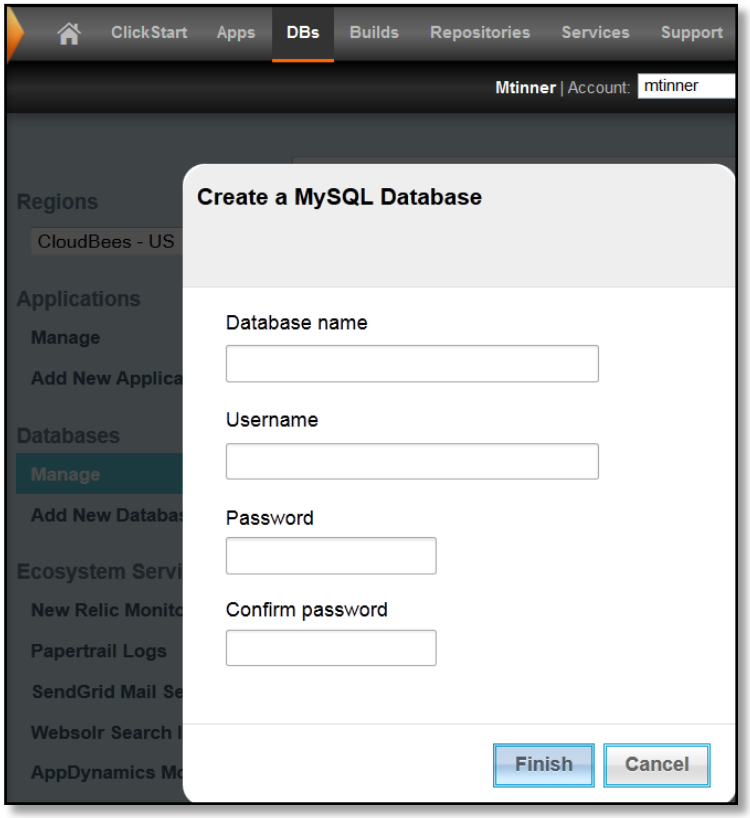


Abbildung 58 - CloudBees MySQL Database erstellen

13.3.2 Datenbankinformationen anzeigen

1. Nach dem Anmelden bei CloudBees steht der Reiter „DBs“ zur Verfügung, auf diesen klicken.
2. Anschliessend auf der rechten Seite unter dem Punkt „Databases“ auf „Manage“ klicken. Nachfolgend die gewünschte Datenbank auswählen.

Die Webseite zeigt Informationen an, welche im folgenden Bild zu sehen sind.

Abbildung 59 - MySQL Informationen

13.4 MongoHQ Einrichtung

Folgende Anleitung zeigt das Erstellen einer MongoHQ sowe deren Benutzer Datenbank bei CloudBees.

13.4.1 Erstellen einer MongoHQ Datenbank

1. Nach dem Anmelden bei CloudBees steht der Reiter „Services“ zur Verfügung, auf diesen klicken.
2. Anschliessend auf „Add services“ klicken.
3. Danach in der Spalte „RUN@Cloud Sevices“ bei MongoHQ auf „Edit Subscription“ klicken.
4. Folgend erscheint unten ersichtliche Webseite. Bei dieser auf „Subscribte“ klicken.

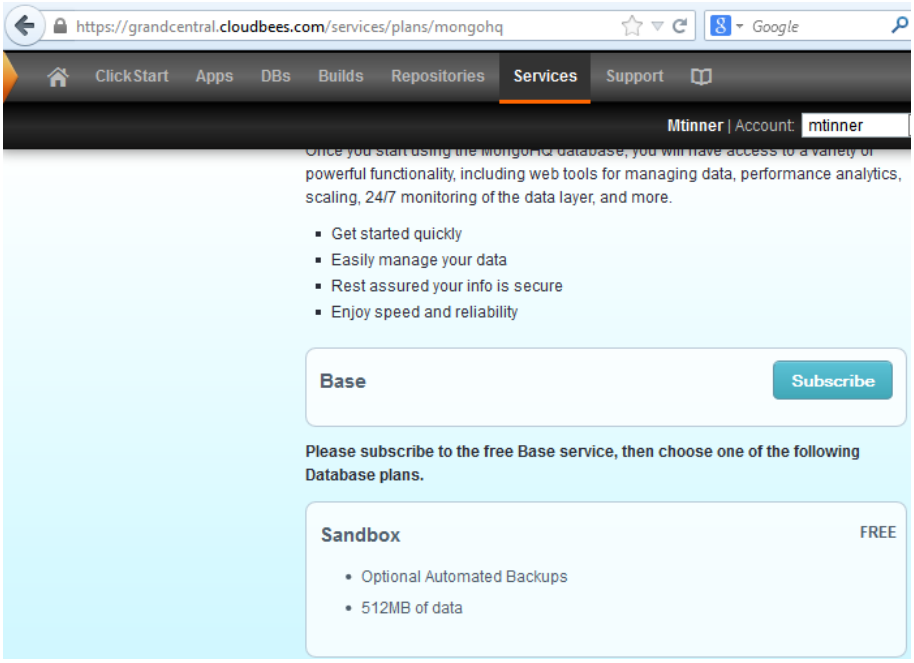


Abbildung 60 - Subscribe MongoHQ

5. Folglich auf „Create new mongodb“ klicken um eine neue Datenbank zu erstellen.
6. Auf der neu erschienen Webseite das Textfeld „Database Name“ ausfüllen und in der Dropdown-Liste „Plan“ das gewünschte Element auswählen.

13.4.2 MongoHQ Einstellungen

1. Nach dem Anmelden bei CloudBees steht der Reiter „Services“ zur Verfügung, auf diesen klicken.
2. Anschliessend bei MongoHQ auf „Manage“ und auf der folgenden Seite bei der gewünschten Datenbank auf „View“ klicken.

Abbildung 61 - MongoHQ Datenbankverwaltung

3. Folgend sind unter Admin→Users weitere Benutzer anzulegen und unter Admin→Overview können die nötigen Informationen ausgelesen werden, um eine Verbindung mit der Datenbank herzustellen.

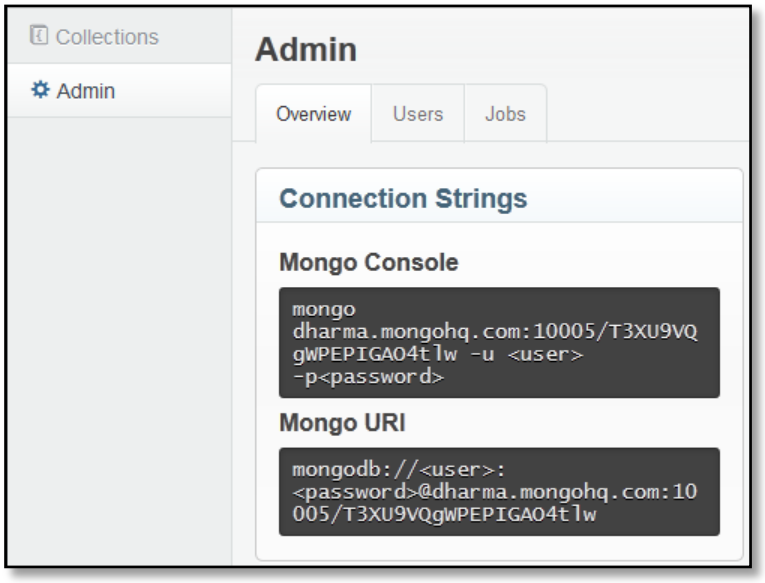


Abbildung 62 - MongoHQ Informationen

13.5 Jenkins CI

13.5.1 Vorbedingungen

Jenkins unter Subscribed Services

13.5.2 Job anlegen

Über dem Reiter Services findet man Jenkins. Alternativ kann man auch über folgende URL zu Jenkins gelangen:
http://username.ci.cloudbees.com

In der Navigation kann über „Neuen Job anlegen“ ein neuer Jenkins Job angelegt werden. Nach der Eingabe des Job Namens die Einstellung *„Free Style“-Softwareprojekt bauen* wählen.

13.5.3 Source Code Management

Bei der Konfiguration des Jobs kann unter Source-Code-Management eine URL zu einem SVN oder Git Repository angegeben werden. Nachfolgend kann bei Build-Auslöser „Source Code Management System abfragen“ gewählt werden. Somit wird der Job jeweils ausgeführt, sobald sich etwas im Repository ändert.

13.5.4 Unit Tests

Damit beim Build-Vorgang die Unit Tests ausgeführt werden können, muss man im Projekt eine Teilung von Test- und Anwendungs-Code einhalten. Danach kann über eine Dependency via Maven JUnit eingebunden werden.

13.5.4.1 Struktur Java Projekt

Abbildung 63 - Jenkins JUnit Java Projektstruktur

13.5.4.2 Anpassung pom.xml

Für die Nutzung von JUnit muss folgende Dependency in der pom-xml Datei hinzugefügt werden:

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
</dependency>
```

Quellcode 40 - Maven JUnit dependency

Zusätzlich wird das Surefile-Plugin mit folgenden Einstellungen benötigt:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
</plugin>
```

Quellcode 41 - Maven Surefile-Plugin

13.5.5 Automatisches Deployment

Unter Post-Build-Aktionen kann eingestellt werden, dass nach dem Build das Projekt automatisch deployed wird. Hierzu “Add post-build action” auswählen und danach “Deploy artifacts to my Private CloudBees Repository”.

14 Anleitungen Google App Engine

14.1 Sample App

Anleitung für die Nutzung von Google App Engine mit Eclipse und dem Google App Engine Plugin.

14.1.1 Vorbereitung

1. Um Google App Engine mit Eclipse zusammen gebrauchen zu können, muss ein Google-Account erstellt werden. Dieser kann unter folgender URL erstellt werden: https://developers.google.com/appengine/?hl=de. Zusätzlich ist das Erstellen einer App im Google-Account notwendig https://appengine.google.com/.

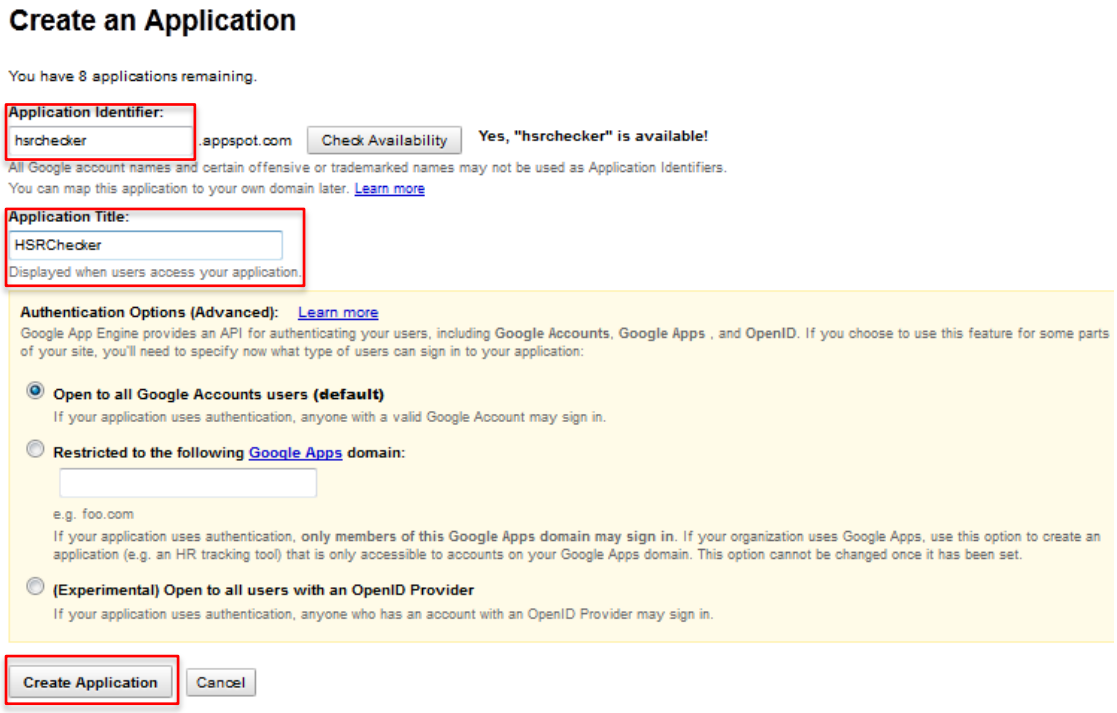


Abbildung 64 - Google App Engine Create Application

2. Zudem wird die EE Developer Version von Eclipse benötigt.

14.1.2 Installation des Plug-Ins

1. Installieren des Google App Engine-Plugins in Eclipse.
   a. Help > Install New Software

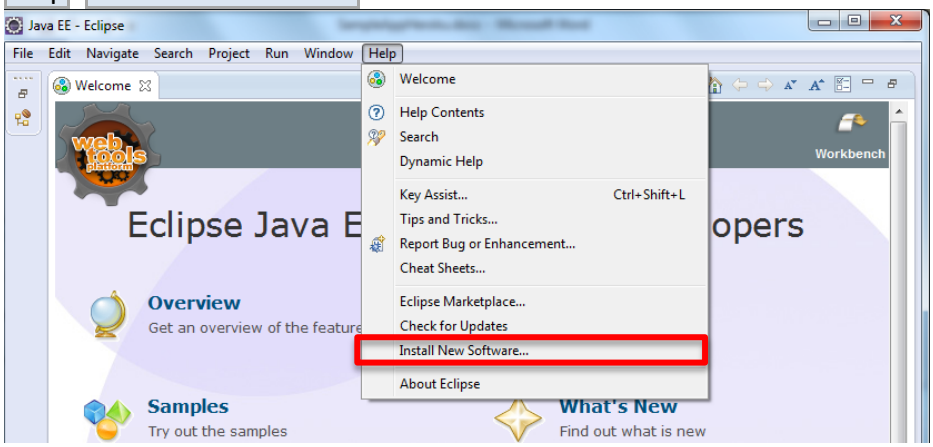


Abbildung 65 - Eclipse Install New Software

   b. Klicke auf Add um den Plugin-URL hinzuzufügen.
   c. Den URL https://dl.google.com/eclipse/plugin/4.3 bei der Location hinzufügen und einen Namen eingeben z.B. *Google App Engine*. Danach OK drücken.
   d. Die folgenden CheckBoxen selektieren und installieren.

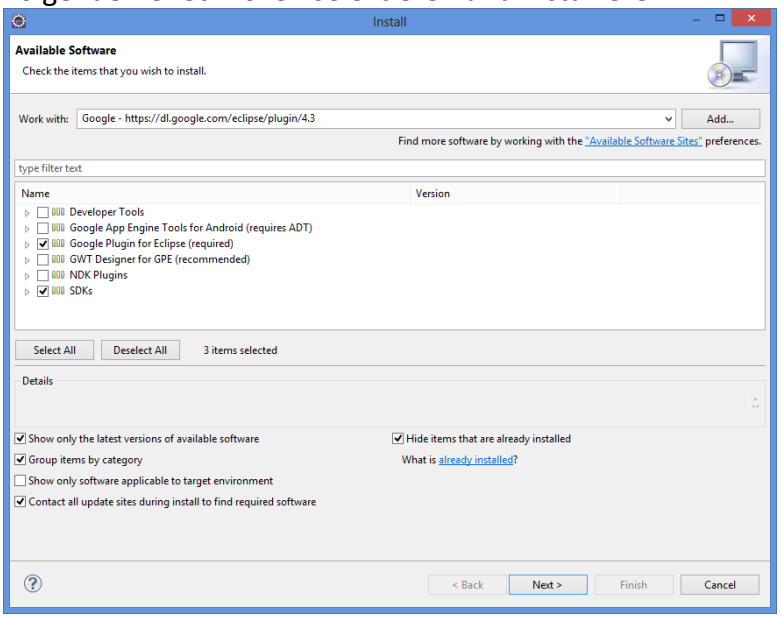


Abbildung 66 - Eclipse Google App Engine Software Packages

   e. Nach dem Neustart von Eclipse muss man sich im Eclipse-Fenster links unten bei Google anmelden.

14.1.3 Erstellen einer Anwendung

1. Erzeugen einer Beispielanwendung
   a. Erstellen eines Projektes über File>New>Other>Google>Web Application Project>Next
   b. Danach muss ein Projektname, Packagename und die App Id eingegeben werden. Auch Die CheckBox „Use Google Web Toolkit“ muss deselektiert werden.

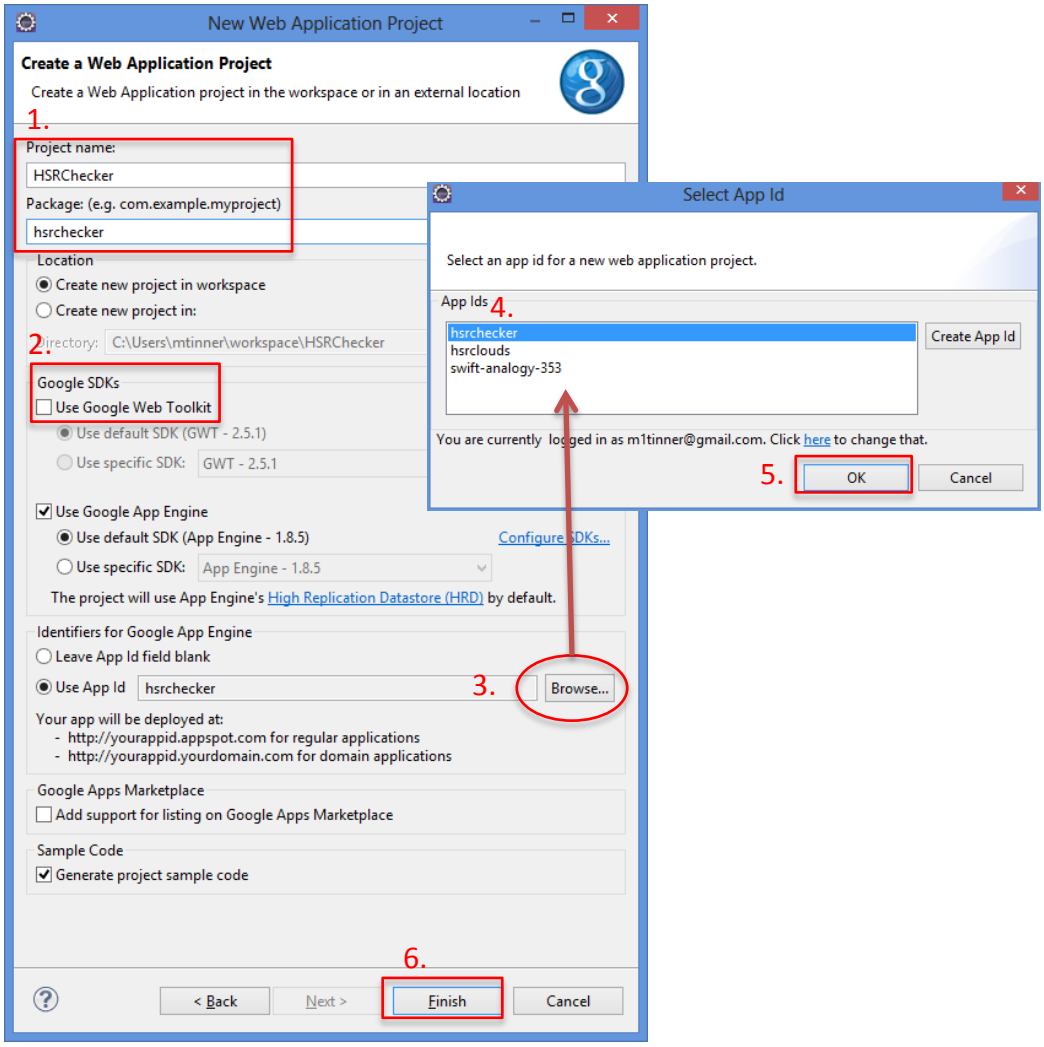


Abbildung 67 - Google App Engine Application Deployment

14.1.4 Anwendung deployen

Rechtsklick auf das Projekt Google>Deploy to App Engine>Deploy

Bei einer neu Importieren Anwendung ist anschliessend die „Application ID“ zu setzten, damit wird das Zielprojekt bei Google App Engine ausgewählt.

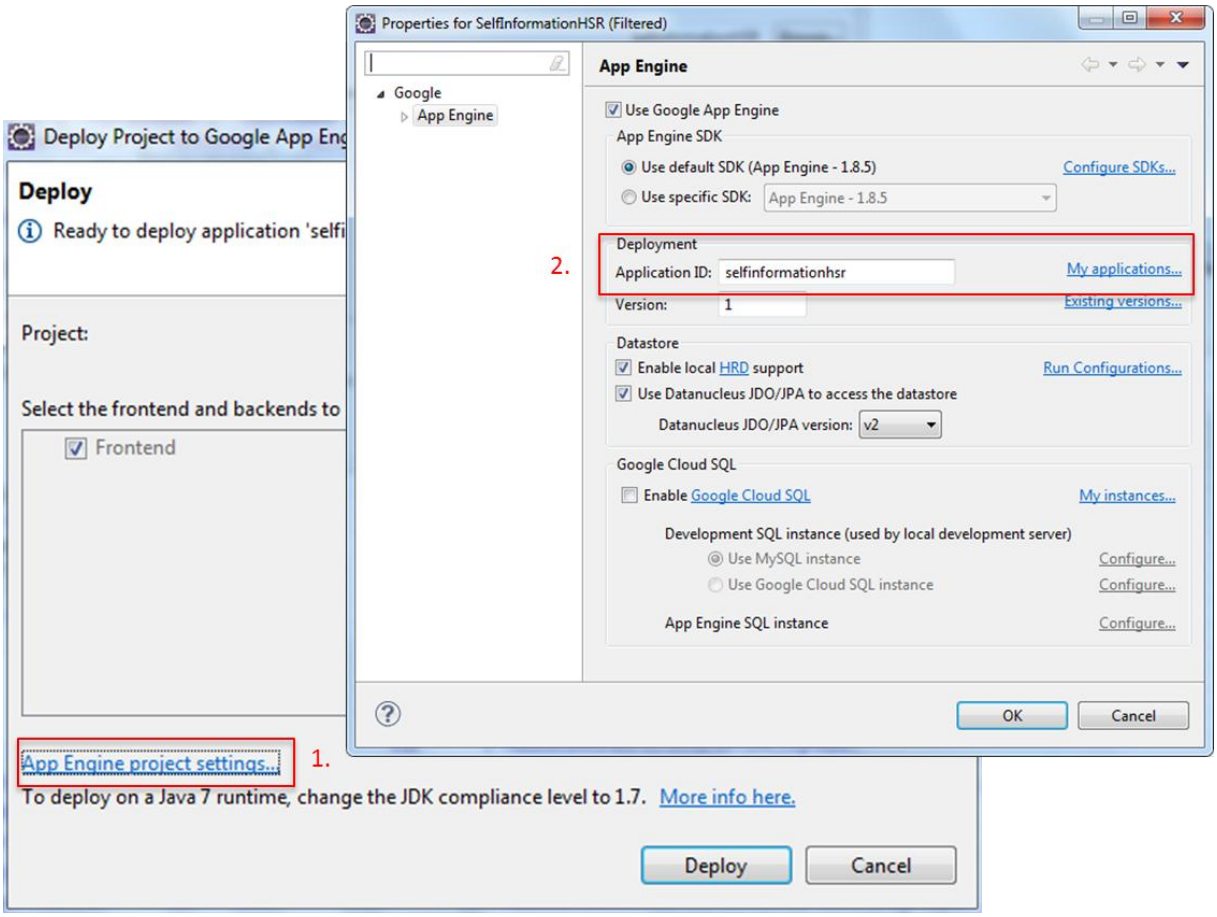


Abbildung 68 - Google App Engine Application ID

14.2 Google Data Store

Dies ist eine Anleitung für die Nutzung von Google Datastore. In dieser Anleitung werden das Erstellen, das Abfragen und das Löschen von Einträgen im Datastore von Google behandelt.

14.2.1 Vorbereitung

Um mit der Datenbank von Google zu arbeiten, muss ein Web Application Project vorhanden sein. In der Datenbankklasse wird für diese Anleitung folgendes Attribut benötigt:

```
1   DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
```

14.2.2 Erstellen von Einträgen

Um einen Eintrag abzuspeichern, muss dieser zuerst erzeugt werden. Hierbei wird lokal ein neues Entity-Objekt erzeugt mit dem Namen „Customer“. Danach kann dem Customer-Objekt ein neues Attribut hinzugefügt werden. Anschliessend kann das Objekt in der Datenbank gespeichert werden.
Es können auch mehrere Objekte mit demselben Objektnamen erzeugt werden.

```
1   Entity e = new Entity("Customer";
2   e.setProperty("name", "Hans");
3   datastore.put(e);
```

Quellcode 42 - Datastore Eintrag erstellen

14.2.3 Abfragen von Einträgen

Der im Kapitel **Error! Reference source not found.** erstellte Eintrag soll nun wieder ausgelesen werden. Hierfür wird ein Query-Objekt unter Angabe des gesuchten Elementes erzeugt. Diese Anfrage wird anschliessend auf der Datenbank abgefragt. Durch eine foreach-Schleife kann über die erhaltenen Entities die mit Customer abgespeichert worden sind, iteriert werden.

```
1   Query q = new Query("Customer");
2   PreparedQuery pq = datastore.prepare(q);
3   for (Entity result : pq.asIterable()) {
4     //do something
5   }
```

Quellcode 43 - Datastore Eintrag abfragen

14.2.4 Löschen von Einträgen

Hierbei spielt es eine wichtige Rolle, ob die Daten bereits bei Google abgespeichert sind oder nur lokal.

1. Löschen der lokalen Datenbankeinträgen:
   Im Projektverzeichnis unter `war`>`WEB-INF`>`appengine-generated` befindet sich die Datei `local_db.bin`. Wenn diese gelöscht wird, werden auch alle in der lokalen Datenbank gespeicherten Einträge gelöscht.
2. Löschen der Datenbankeinträgen bei Google Datastore:
   Bei Google App Engine auf die App welche die Datenbankeinträge anlegt. Danach unter `Datastore Viewer` können die Einträge gelöscht werden.

14.2.5 Was ist zu beachten

Wenn keine Datenbankeinträge mehr zurückgegeben werden.
Hierbei ist zu beachten, dass möglicherweise das Tagespensum von Google App Engine bei der Gratisnutzung erreicht wurde.

15 Anleitungen Diverses

15.1 Installationsanleitung Maven

1. Hinzufügen der Variable „JAVA_HOME“ zu den Umgebungsvariablen und dabei auf den JDK-Ordner verlinken.
   Hierzu Windows + Pause drücken. Anschliessend auf Advanced system settings und unter dem Reiter Advanced auf Environment Variables….

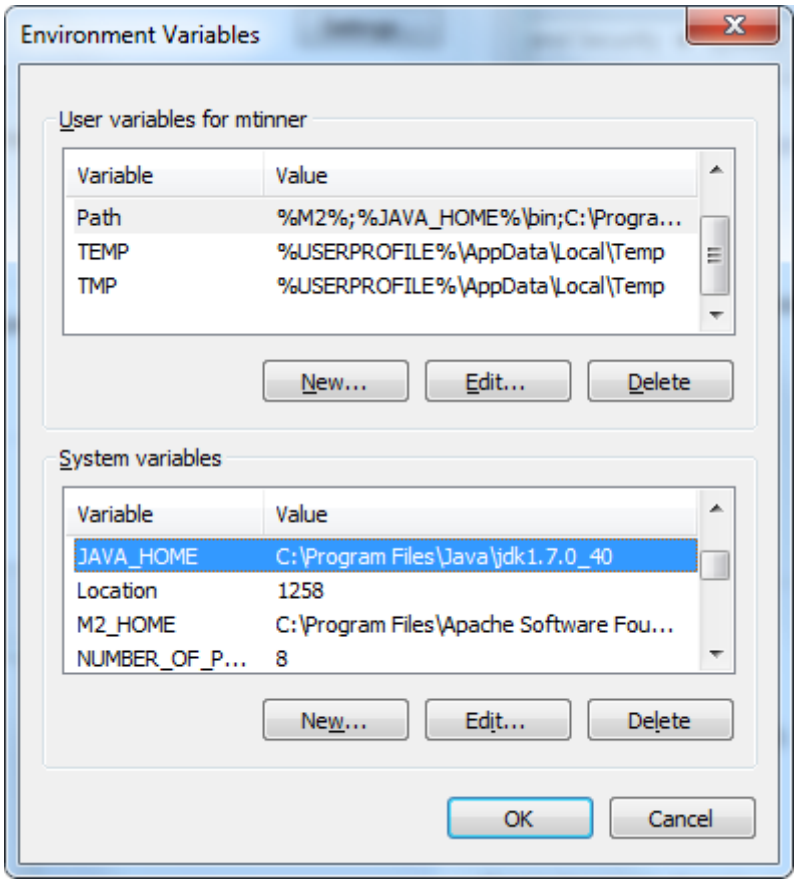


Abbildung 69 – Java Umgebungsvariable

2. Maven herunterladen von http://maven.apache.org/download.cgi, dabei die Version „apache-maven-3.1.1-bin.zip“ auswählen.
3. Datei in das Verzeichnis „C:\Program Files\Apache Software Foundation“ entzippen.
4. MAVEN_HOME als Variable setzten und dabei den Verzeichnispfad von Maven als Wert nehmen.

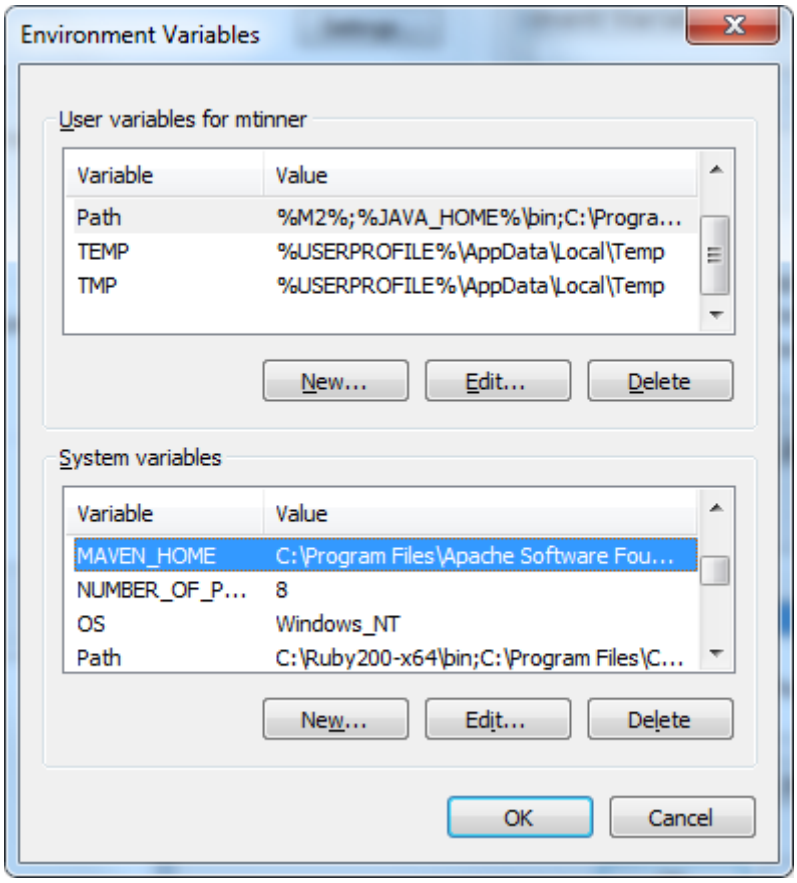


Abbildung 70 - Maven Umgebungsvariable

5. Die Systemvariable „Path“ mit dem Wert „%MAVEN_HOME%bin“ ergänzen.

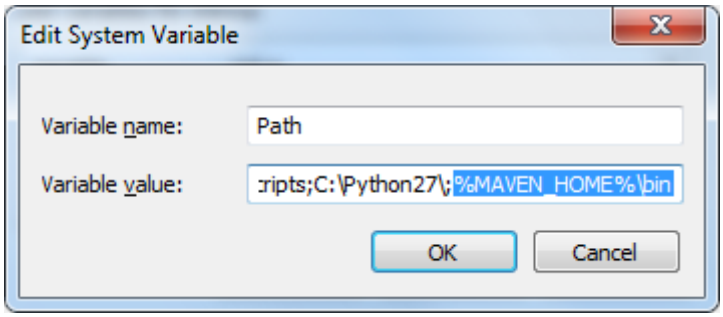


Abbildung 71 - Path ergänzen

Als Verifikation kann in der Kommandozeile den Befehl „mvn -version“ eingegeben werden. Nach erfolgreicher Eingabe sollten die Versionsnummer von Maven, sowie die des JDK erscheinen.

15.2 Import bestehender Projekte

In dieser Anleitung wird der Import der Archiv-Files aller Cloud-Anbieter beschrieben. Diese Anleitung bezieht sich ausschliesslich auf den Import bei Eclipse.

Google App Engine:

Um Programme zu verwenden, welche mit Google App Engine erstellt wurden, muss zuerst ein Plug-In installiert werden. Dies wird im Unterabschnitt „14.1.2 Installation des Plug-Ins“ beschrieben. Anschliessend kann ebenfalls diese Anleitung verwendet werden.

15.2.1 Import eines Archiv-Files

1 In Eclipse über File→Import den Import-Dialog öffnen.
2 Im Ordner General→Existing Projects into Workspace auswählen und auf Next klicken.
3 Unter dem Punkt „Select archive file“ über „Browse…“ das Archiv-File auswählen und auf Finish klicken.

15.3 DDDSample Projektimport

Einrichten des Projektes von DDDSample und Export einer war-Datei für den späteren Deploy auf einen Cloud-Dienst.

15.3.1 Vorbereitung

- Über die Webseite http://dddsample.sourceforge.net/download.html kann die DDDSample Applikation als source-package heruntergeladen werden.
- Zudem wird die EE Developer Version von Eclipse benötigt.

15.3.2 Erstellung Projekt

- In Eclipse: via File→New→New Project→General→Project ein neues Java Projekt erstellen.
- Rechtsklick auf das Projekt: Import... -> General -> File System -> Next

- From directory: Browse -> entpackter Ordner vom dddsample-src auswählen -> Ordner im linken Fenster anwählen -> Finish

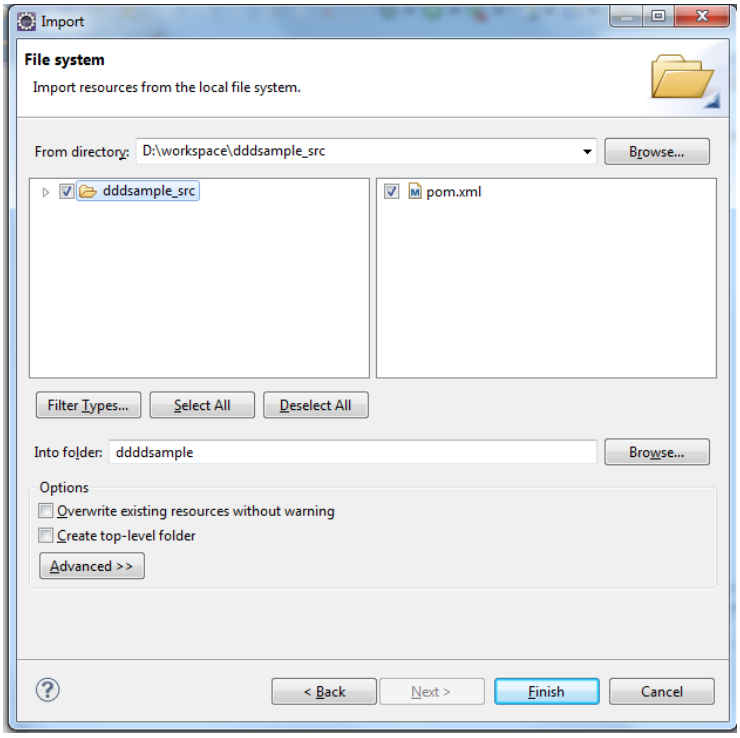


Abbildung 72 - DDDSample Projektimport

15.3.3 Konfiguration Projekt

- Mit einem Rechtsklick auf das Projekt → Configure → Convert to Maven Project die genutzten Abhängigkeiten und Einstellungen konfigurieren.

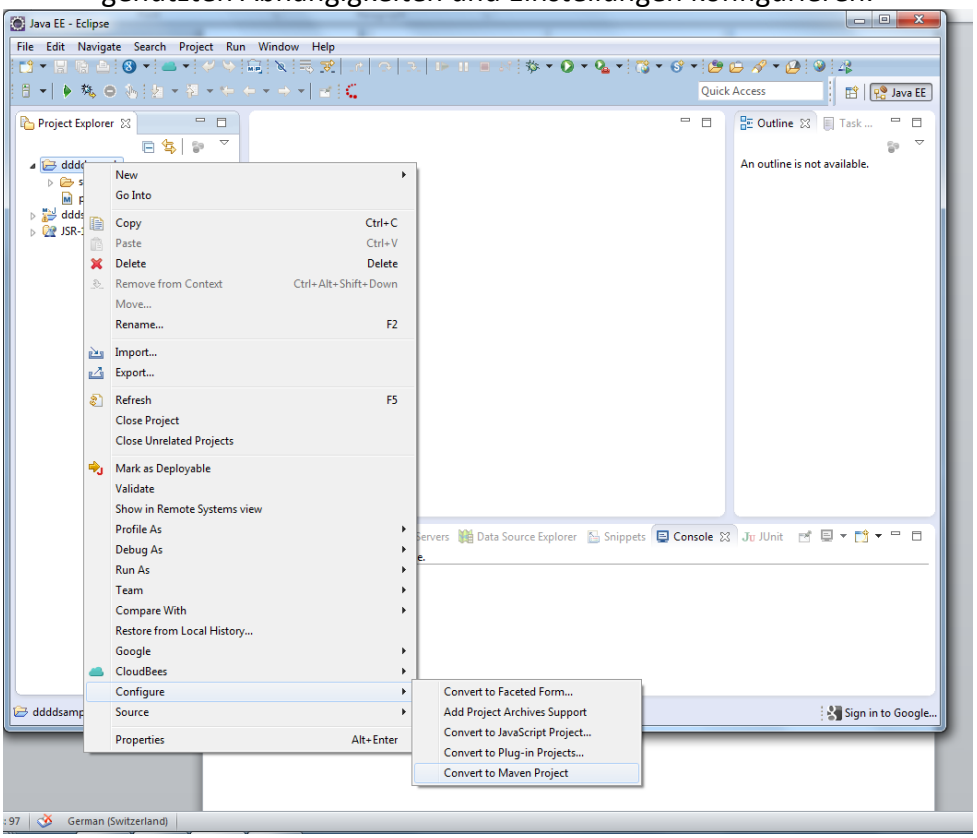


Abbildung 73 - DDDSample Projekt Konfiguration

Dies wird die externen Libraries, welche das dddsample-Projekt benutzt, herunterladen. Der Fortschritt wird am rechten unteren Bildschirmrand angezeigt.

Abbildung 74 - Maven Fortschritt

15.3.4 Export Projekt

- Nachdem das Projekt mit Maven konfiguriert worden ist, kann mittels Rechtsklick auf das Projekt → Export → WAR file eine War Datei exportiert werden.

15.3.5 Aufgetretene Schwierigkeiten

Nachdem die Maven-Conversation vollendet ist, zeigt Eclipse einen Fehler in der screencast.xml. Da das fehlerhafte Statement nur für die Anzeige eines Youtube-Videos benötigt wird, wurde dieses als Workaround auskommentiert.

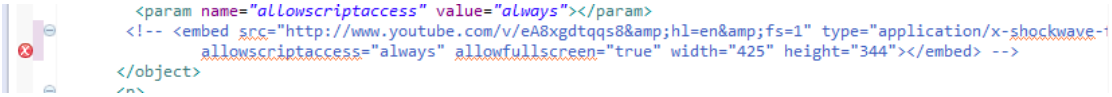


Abbildung 75 - Screencast.xml Fehler

Anhang

A Abbildungen und Tabellen

A.1 Abbildungsverzeichnis

A.2 Tabellenverzeichnis

A.3 Quellcodeverzeichnis

B Glossar

A-Record
: Zuteilung eines DNS-Namens an eine IPv4-Adresse.

ACID
: Zu Deutsch Atomarität, Konsistenz, Isoliertheit und Dauerhaftigkeit beschreiben Eigenschaften betreffend Verarbeitungsschritte welche bei Datenbankmanagementsystemen erwünscht sind.

BSON
: Binary JSON ist eine binär kodierte Serialisierung von Dokumenten, die ähnlich wie JSON-Dokumente erzeugt werden. BSON hat zusätzliche Datentypen wie beispielsweise „Date“.

Clojure
: Programmiersprache welche innerhalb der Java Virtual Machine läuft.

Cloud
: Cloud bezeichnet den Ansatz, IT-Infrastruktur dynamisch dem Benutzer zur Verfügung zu stellen.

CloudBees
: Cloud-Provider mit Infrastruktur bei Amazon (http://www.cloudbees.com)

CNAME
: Canonical name. Weist einer Domäne einen Namen zu.

Datastore
: Nicht-relationaler Key-Value Datenspeicher von Google.

Deployment
: Deployment nennt man den Prozess der Installation von Software auf PCs und Servern.

Django
: Bei Django handelt es sich um ein in Python geschriebenes Web-Framework.

DNS
: Domain Name System

Go
: Kompilierbare Programmiersprache von Google, welche Nebenläufigkeit unterstützt.

Google App Engine
: Cloud-Provider mit eigener Infrastruktur (http://developers.google.com/appengine)

Gradle
: Bei Gradle handelt es sich um ein Build-Management-Automatisierungs-Tool.

Grails
: Hierbei handelt es sich um ein Web Application Framework für die Programmiersprache Groovy.

Heroku
: Cloud-Provider mit Infrastruktur bei Amazon (http://www.heroku.com)

IaaS
: Infrastructure as a Service

Java SE
: Java Platform Standard Edition

Join
: Relationale Verknüpfung zwischen Tabellen in einer Datenbank.

JSON
: JavaScript Object Notation

MongoDB
: MongoDB ist eine dokumentenorientierte Open-Source-Datenbank.

MySQL
: Weltweit am weitesten verbreitetes relationales Datenbanksystem.

Node.js
: Serverseitige Plattform zum Betrieb von Netzwerkanwendungen. Wird häufig für die Realisierung von Webservern benutzt.

NoSQL
: MySQL ist ein verbreitetes relationales Datenbanksystem.

Optimistic Concurrency
: Optimistic Concurrency ist ein Verfahren, welches Zugriffe auf gleiche Daten konfliktarm sowie ohne Inkonsistenzen regelt.

PaaS
: Platform as a Service

PHP
: Skriptsprache welche hauptsächlich zur Erstellung von dynamischen Webseiten und Webanwendungen verwendet wird.

Play
: Hierbei handelt es sich um ein Open Source Web Application Framework.

POJO
: Plain Old Java Object

Procfile
: Konfigurationsdatei in Applikationen bei Heroku, welches dem Provider mitteilt, wie die Applikation gestartet wird.

Python
: Programmiersprache

Rails
: Ruby on Rails, Programmiersprache

Ruby
: Ruby on Rails, Programmiersprache

SaaS
: Software as a Service

Scala
: Scala ist eine funktionale und objektorienterte Programmiersprache.

SDK
: Software Development Kit

Servlet
: Ein Servlet ist eine Java-Instanz, welche innerhalb eines Webservers Anfragen von Clients engegen-nehmen, verarbeiten und beantworten kann.

Socket
: Ein Socket ist ein Software-Komponent, mit welchem man sich verbinden kann.

Spring
: Das Spring Framework ist ein quellofenes Framework für die Java-Plattform.

Struts 2
: Struts ist ein Open-Source-Framework für die Präsentations- und Steuerungsschicht von Java-Webanwendungen.

TCP

TCP ist in einem TCP/IP Netzwerk für den Transport und die Sicherung der Daten zuständig.

Tutorial

Gebrauchsanleitung

C Literatur

[Add13] *Add-Ons*. Heroku. https://addons.heroku.com/iron_mq#pro_platinum_mq, zuletzt geprüft 2013.

[App13] *App Sleeping on Heroku*. Heroku. https://blog.heroku.com/archives/2013/6/20/app_sleeping_on_heroku, zuletzt geprüft 2013.

[BSO13] *BSON*. BSON. bsonspec.org, zuletzt geprüft 2013.

[Bui13] *Buildpacks*. Heroku. https://devcenter.heroku.com/articles/buildpacks, zuletzt geprüft 2013.

[Bui131] *Buildpacks*. Heroku. https://devcenter.heroku.com/articles/third-party-buildpacks, zuletzt geprüft 2013.

[Chr13] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 107.

[Chr131] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 96.

[Chr1310] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 277.

[Chr135] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 94.

[Chr136] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing patterns. Springer. S. 168.

[Chr137] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 195.

[Chr138] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 103.

[Chr139] Christoph Fehling F.L.R.R.W.S.P.A. (2013): Cloud Computing Patterns. Springer. S. 299.

[Cli13] *ClickStart*. CloudBees. http://wiki.cloudbees.com/bin/view/RUN/ClickStart, zuletzt geprüft 2013.

[Clo13] *CloudBees SDK*. CloudBees. http://wiki.cloudbees.com/bin/view/RUN/BeesSDK, zuletzt geprüft 2013.

[Clo131] *CloudAMQP*. CloudBees. https://grandcentral.cloudbees.com/services/plans/cloudamqp, zuletzt geprüft 2013.

[Clo132] *CloudBees*. CloudBees. http://www.cloudbees.com/#slide-1, zuletzt geprüft 2013.

[Com13] *Community*. CloudBees. https://github.com/CloudBees-community, zuletzt geprüft 2013.

[Cre13] *Create a Java Web Application using Embedded Tomcat*. Heroku. https://devcenter.heroku.com/articles/create-a-java-web-application-using-embedded-tomcat, zuletzt geprüft 2013.

[Das13] *Das kleine MongoDB Einmaleins*. http://www.mongodb.com/static/downloads/Sample_MongoDB_ebook_german.pdf, zuletzt geprüft 2013.

[Dat13] *Database Guide*. CloudBees. https://developer.cloudbees.com/bin/view/RUN/DatabaseGuide, zuletzt geprüft 2013.

[Dat131] *Database References*. MongoDB. http://docs.mongodb.org/manual/reference/database-references/, zuletzt geprüft 2013.

[Dat133] *Data Acces Layer*. Wikipedia. http://en.wikipedia.org/wiki/Data_access_layer, zuletzt geprüft 2013.

[Dat134] *Data-stores*. Heroku. https://addons.heroku.com/#data-stores, zuletzt geprüft 2013.

[Dns13] *dnsimple*. Aetrion LLC. https://dnsimple.com/, zuletzt geprüft 2013.

[Dns131] *DNS*. SmarterTools Inc. http://support.powerdnn.com/kb/a604/dns-propagation-and-why-it-takes-so-long-explained, zuletzt geprüft 2013.

[Dyn13] *Dyno size*. Heroku. https://devcenter.heroku.com/articles/dyno-size, zuletzt geprüft 2013.

[Dyn131] *Dynos*. Heroku. https://devcenter.heroku.com/articles/dynos, zuletzt geprüft 2013.

[Ent13] *Entities, Properties, and Keys*. Google. https://developers.google.com/appengine/docs/java/datastore/entities, zuletzt geprüft 2013.

[Err13] *Error Codes*. Heroku. https://devcenter.heroku.com/articles/error-codes, zuletzt geprüft 2013.

[FAQ13] *FAQ*. CloudBees. http://wiki.cloudbees.com/bin/view/RUN/FAQ, zuletzt geprüft 2013.

[Fin13] *Finding IP/address/port*. CloudBees. https://developer.cloudbees.com/bin/view/Main/Finding+out+app+port+and+hostname, zuletzt geprüft 2013.

[Fre13] *Frequently Asked Questions*. CloudBees. http://wiki.cloudbees.com/bin/view/RUN/FAQ, zuletzt geprüft 2013.

[Get13] *Getting Started with Java driver*. MongoDB. http://docs.mongodb.org/ecosystem/tutorial/getting-started-with-java-driver/, zuletzt geprüft 2013.

[Get131] *Getting Started with Heroku*. Heroku. https://devcenter.heroku.com/articles/quickstart, zuletzt geprüft 2013.

[Goo13] *Google App Engine*. Google. https://developers.google.com/appengine/?hl=de, zuletzt geprüft 2013.

[Her13] *Heroku*. Heroku. https://id.heroku.com/login, zuletzt geprüft 2013.

[Int13] *Introducing the ALIAS Record*. Aetrion LLC. http://blog.dnsimple.com/zone-apex-naked-domain-alias-that-works/, zuletzt geprüft 13.

[Int131] *Interface ServletContextListener*. Oracle. http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContextListener.html#contextDestroyed%28javax.servlet.ServletContextEvent%29, zuletzt geprüft 2013.

[JJo13] Joller J. (FS 2013): VSS-U2_Lösungen. In: *Verteilte Softwaresysteme*. HSR.

[JSO13] *JSON in Java*. http://json.org/java/, zuletzt geprüft 2013.

[JVM13] *JVM Runtime Container*. CloudBees. https://developer.cloudbees.com/bin/view/RUN/Java+Container, zuletzt geprüft 2013.

[LBl13] Bläser L. (FS 2013): 01_Threads. In: *Parallele Netzwerkprogrammierung*. HSR.

[Map13] *Map Reduce*. Wikipedia. http://de.wikipedia.org/wiki/MapReduce, zuletzt geprüft 2013.

[Mav13] *Maven / doesn't use java 7*. Stack Exchange. http://stackoverflow.com/questions/8750563/maven-doesnt-use-java-7, zuletzt geprüft 2013.

[Mav131] *Maven*. Apache. http://maven.apache.org/, zuletzt geprüft 2013.

[Mon13] *MongoDB*. Wikipedia. http://de.wikipedia.org/wiki/MongoDB, zuletzt geprüft 2013.

[Mon132] *MongoDB and Java*. WordPress. http://www.zorched.net/2010/06/17/mongodb-and-java-find-an-item-by-id/, zuletzt geprüft 2013.

[Mon133] *MongoDB Hello World*. http://www.mkyong.com/mongodb/java-mongodb-hello-world-example/, zuletzt geprüft 2013.

[MyS13] *MySQLJava*. http://www.vogella.com/articles/MySQLJava/article.html, zuletzt geprüft 2013.

[New13] *New Relic*. CloudBees. http://www.cloudbees.com/platform-service-newrelic.cb, zuletzt geprüft 2013.

[Pac13] *Package Summery*. Google. https://developers.google.com/appengine/docs/java/javadoc/com/google/appengine/api/datastore/package-summary, zuletzt geprüft 2013.

[Placeholder1] *MapReduce*. Wikipedia. http://de.wikipedia.org/wiki/MapReduce, zuletzt geprüft 2013.

[Placeholder2] *What is Google App Engine*. Google. https://developers.google.com/appengine/docs/whatisgoogleappengine, zuletzt geprüft 2013.

[Pri13] *Pricing*. CloudBees. www.cloudbees.com/platform/pricing/runcloud-multi/example.cb, zuletzt geprüft 2013.

[Pri131] *Pricing*. Google. https://cloud.google.com/products/app-engine/, zuletzt geprüft 2013.

[Pro13] *Process Types and the Procfile*. Heroku. https://devcenter.heroku.com/articles/procfile, zuletzt geprüft 2013.

[Que13] *Queries*. Google. https://developers.google.com/appengine/docs/java/datastore/queries, zuletzt geprüft 2013.

[Req13] *Request Timeout*. Heroku. https://devcenter.heroku.com/articles/error-codes#h12-request-timeout, zuletzt geprüft 2013.

[Run13] *Run non-web Java dynos on Heroku*. Heroku. https://devcenter.heroku.com/articles/run-non-web-java-processes-on-heroku, zuletzt geprüft 2013.

[SKe12] Keller S. (HS 2012): JDBC_Einführung. In: *Datenbanken 1*. HSR.

[SKe13] Keller S. (FS 2013): NoSQL_v2. In: *Datenbanken 2*. HSR.

[Soc13] *Socket*. Wikipedia. http://de.wikipedia.org/wiki/Socket_%28Software%29, zuletzt geprüft 2013.

[The13] *The Twelve Factors*. http://12factor.net, zuletzt geprüft 2013.

[Too13] *Toolbelt*. Heroku. https://toolbelt.heroku.com/, zuletzt geprüft 2013.

[Url13] *Url Fetch*. Google. https://developers.google.com/appengine/docs/java/urlfetch/, zuletzt geprüft 2013.

[Val13] *Valitator*. W3. http://validator.w3.org, zuletzt geprüft 2013.

[War13] *Warmup Requests*. Google. https://developers.google.com/appengine/docs/adminconsole/instances?hl=de-DE&csw=1#Warmup_Requests, zuletzt geprüft 2013.

[Wha13] *What is Google App Engine*. Google. https://developers.google.com/appengine/docs/whatisgoogleappengine, zuletzt geprüft 2013.

D RessourcenMonitor

Folgendes Kapitel zeigt die Anforderungen und Use Cases, welche die RessourcenMonitor-Applikation erfüllen muss auf. Ebenfalls weist es die erstellte Struktur des Projektes.

D.1 Use Cases

D.1.1 Use Case Diagramm

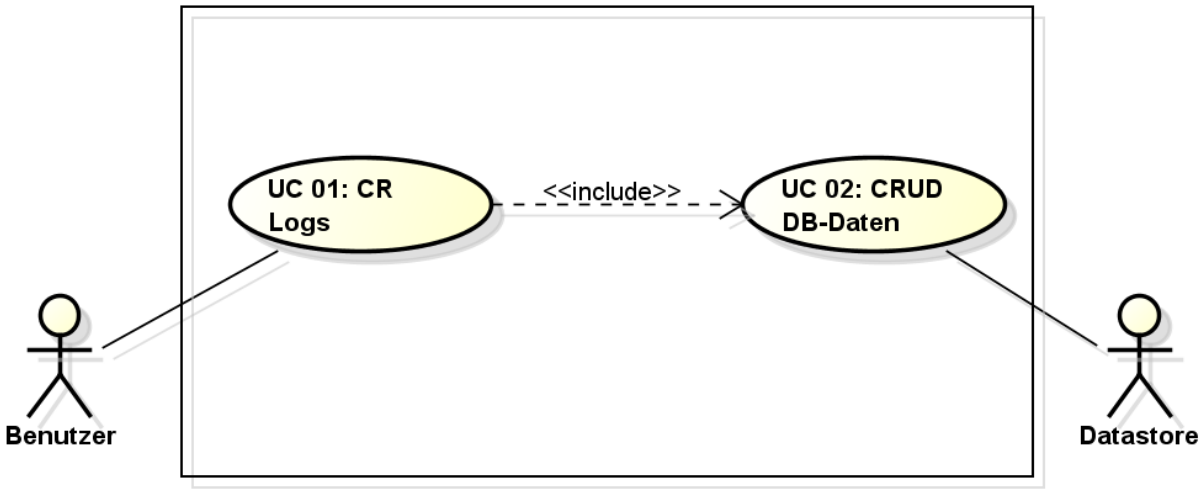


Abbildung 76 - RessourcenMonitor Use Case Diagramm

D.1.2 Aktoren & Stakeholder

- Benutzer
- Clouddienst

D.1.3 Beschreibungen

D.1.3.1 UC01: CR Contract

Der Benutzer kann Logs erzeugen und auslesen.

D.1.3.2 UC02: CR Customer

Das System fügt Einträge in die Datenbank und liest die Einträge aus der Datenbank. Die Datenbankeinträge können zudem gelöscht und verändert werden.

D.2 Packages

Die Applikation ist in drei verschiedene Schichten gegliedert.

- Presentation Layer (ch.hsr.pl)
- Business Logic Layer (ch.hsr.bll)
- Data Access Layer (ch.hsr.dal)

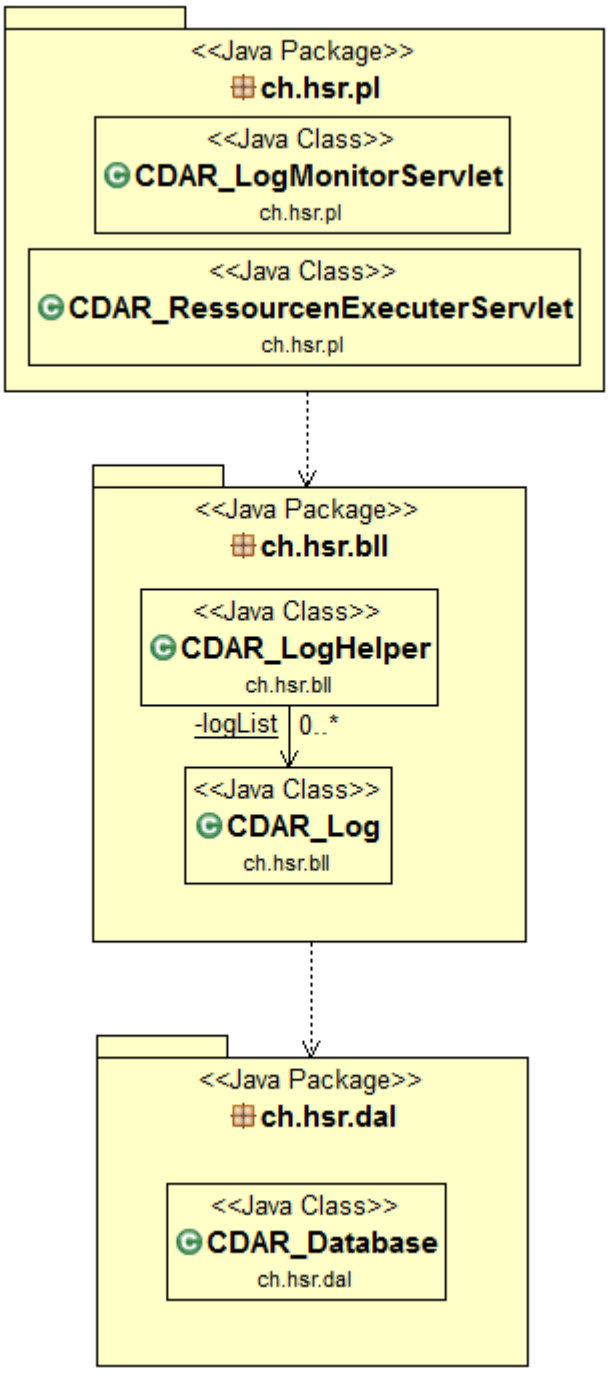
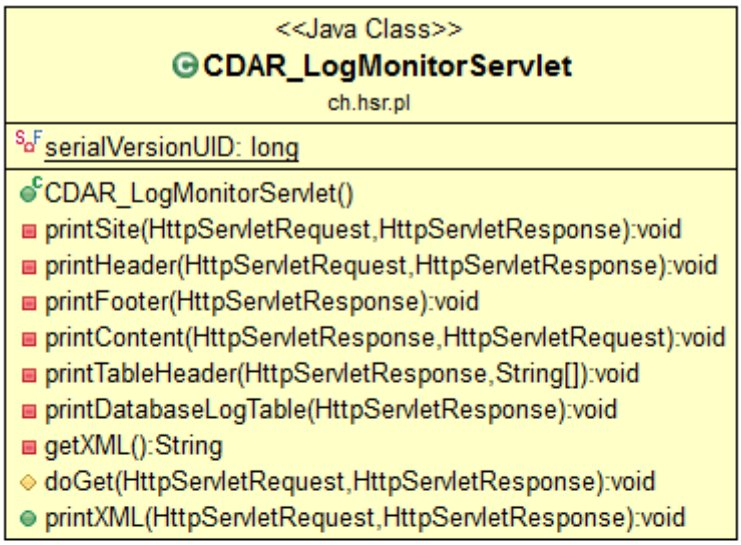


Abbildung 77 - RessourcenMonitor Packages

D.2.1 ch.hsr.pl

In dieser Schicht befinden sich die Servlets. Das Servlet „CDAR_LogMonitorServlet“ zeigt die Logs an und mit dem Servlet „CDAR_RessourcenExecuterServlet“ werden Datenbankeinträge veranlasst.



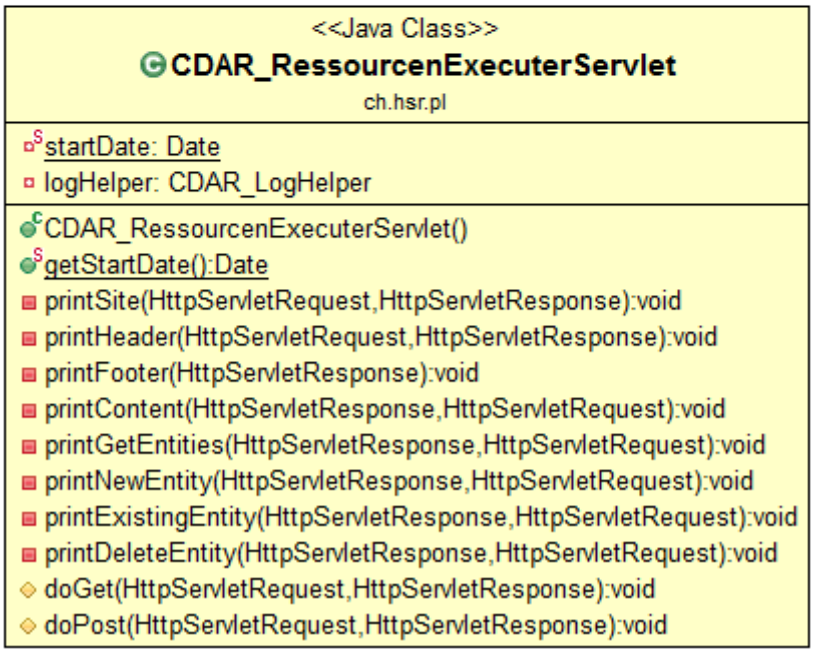


Abbildung 78 - RessourcenMonitor Presentation Layer

D.2.2 ch.hsr.bll

Im Business Logic Layer werden die Logs zwischengespeichert. Ebenfalls werden in dieser Schicht die Logeinträge erzeugt.

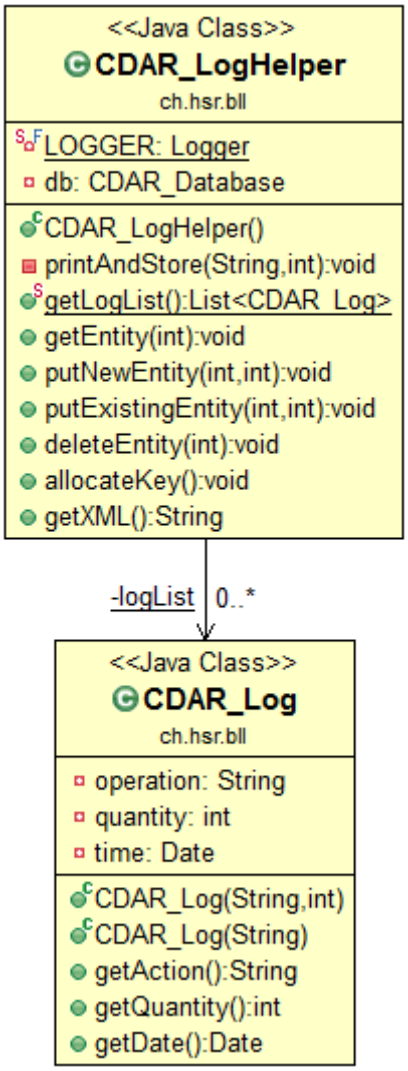


Abbildung 79 - RessourcenMonitor Business Logic Layer

D.2.3 ch.hsr.dal

Im Data Access Layer ist eine reine Kommunikation zwischen der Datenbank und dem Business Logic Layer. In diesem Layer werden die Einträge in die Datenbank eingefügt.

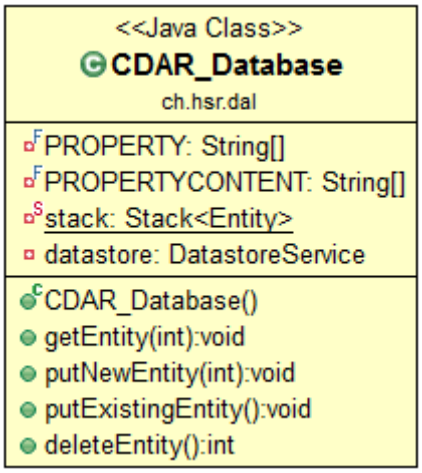


Abbildung 80 - RessourcenMonitor Data Access Layer

D.3 Installationsanleitung

Diese Anleitung bezieht sich auf das Projekt RessourcenMonitor_Google.

D.3.1 RessourcenMonitor Google App Engine

D.3.1.1 Vorbedingungen

- Applikation bei Google App Engine erstellt
- Eclipse-Plug-In von Google App Engine installiert

D.3.1.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Unterabschnitt „14.1.4 Anwendung deployen“ beschrieben, deployen.

D.3.1.3 Aufruf

Um die Servlets aufzurufen, sind folgende Erweiterungen der URL hinzuzufügen.

- Für den Aufruf des Log-Monitors „/LogMonitor“.
- Für den Aufruf des Ressourcen-Executer „/RessourcenExecuter“.

Beispiele:
http://ressourcenmonitor.appspot.com/LogMonitor
http://ressourcenmonitor.appspot.com/RessourcenExecuter

D.3.1.4 Zusätzliche Informationen

Den Namen der Applikation findet sich im Webinterface von Google App Engine.

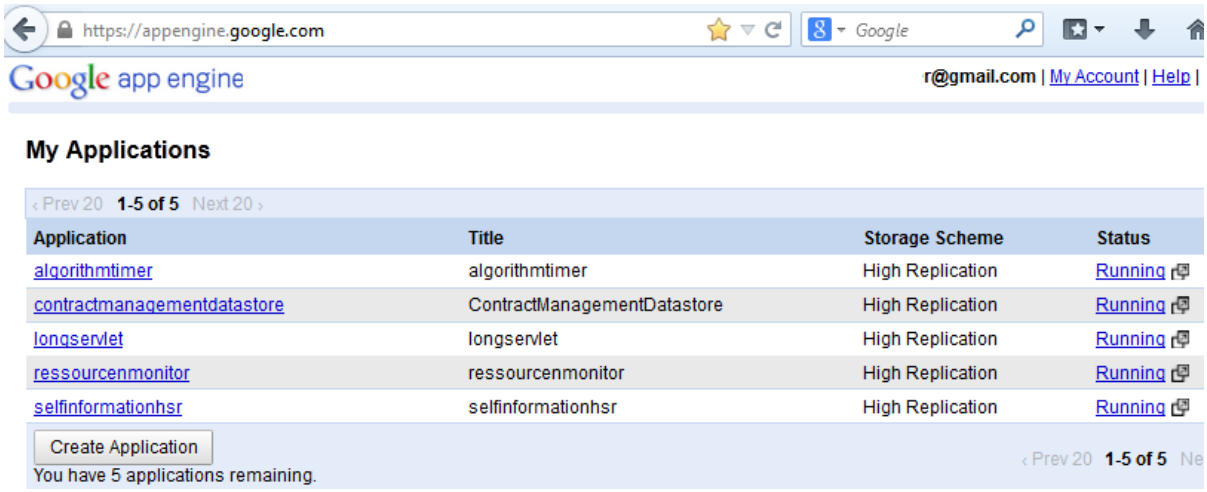


Abbildung 81 - Webinterface Google App Engine Ressourcen Monitor

E SelfInformation

Das folgende Kapitel beschreibt Anforderungen und Use Cases, welche die SelfInformation-Applikation erfüllen muss. Zudem zeigt es die erstellte Struktur des Projektes.

E.1 Funktionale Anforderungen

E.1.1 Einführung

Für die Entwickler, welche ihre Applikationen auf einen Clouddienst deployen möchten, soll eine Applikation entwickelt werden, welche Systeminformationen vom aktuellen Server anzeigt.

E.1.2 Allgemeine Beschreibung

E.1.2.1 Produkt Funktion

Die SelfInformation Applikation soll es dem Anwender ermöglichen, genaue System-informationen abzufragen.

E.1.2.2 Benutzer Charakteristik

Die Zielgruppe sind Entwickler, welche auf dem Clouddienst ihre eigenen Applikationen deployen möchten.

E.1.2.3 Annahmen

Clouddienst unterstützt Java Programme sowie Tomcat7.

E.1.3 Use Cases

E.1.3.1 Use Case Diagramm

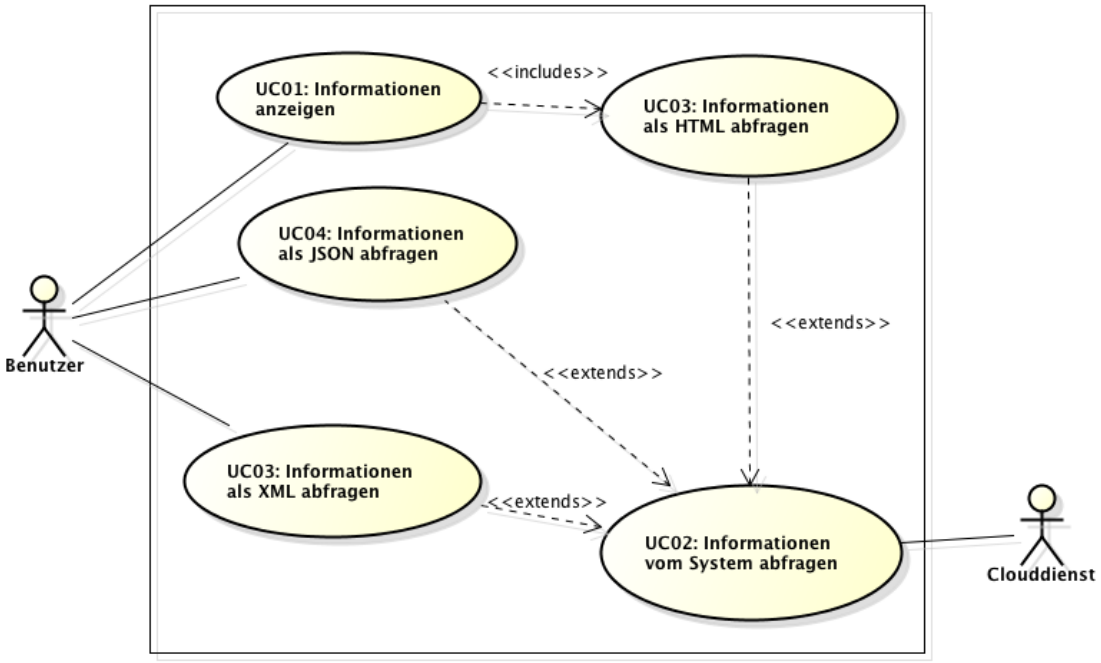


Abbildung 82 - SelfInformation Use Case Diagramm

E.1.3.2 Aktoren & Stakeholder

- Benutzer
- Clouddienst

E.1.3.3 Beschreibungen

E.1.3.3.1 UC01: Informationen anzeigen

Der Benutzer lässt sich im Browser die aktuellen Systeminformationen anzeigen.

E.1.3.3.2 UC02: Informationen vom System abfragen

Das System lädt sich die System-Informationen.

E.1.3.3.3 UC03: Informationen als HTML abfragen

Die Applikation liefert die Informationen im HTML-Format.

E.1.3.3.4 UC04: Informationen als XML abfragen

Die Applikation liefert die Informationen im XML-Format.

E.1.3.3.5 UC05: Informationen als JSON abfragen

Die Applikation liefert die Informationen im JSON-Format.

E.2 Nicht funktionale Anforderungen

E.2.1 Angemessenheit

Die Informationen, welche das System anzeigt, sind für den Benutzer interessant. Die gelieferten Informationen helfen dem Benutzer, seine Applikation für die Cloud vorzubereiten und gegebenenfalls anzupassen.

E.2.2 Interoperabilität

Die Applikation ist in einer Sprache programmiert, welche viele Cloud-Anbieter unterstützen.

E.2.3 Modifizierbarkeit

Die Applikation kann für weitere Verwendungszwecke, beispielsweise einen Webservice ohne grossen Aufwand umgebaut werden.

E.2.4 Zugriff auf Eigenschaften

Die Applikation bietet für gängige Informationen Methoden zur Abfrage an. Übrige Informationen sind im JSON, XML oder HTML zusammengefasst.

E.3 Packagestruktur

Die Applikation lässt sich in drei verschiedene Schichten unterteilen: Presentation Layer, Business Logic Layer sowie Data Access Layer.

E.3.1 Logische Architektur

Die Applikation teilt sich in folgende vier Packages auf:

- ch.hsr.pl
- ch.hsr.bll
- ch.hsr.dal
- ch.hsr.test

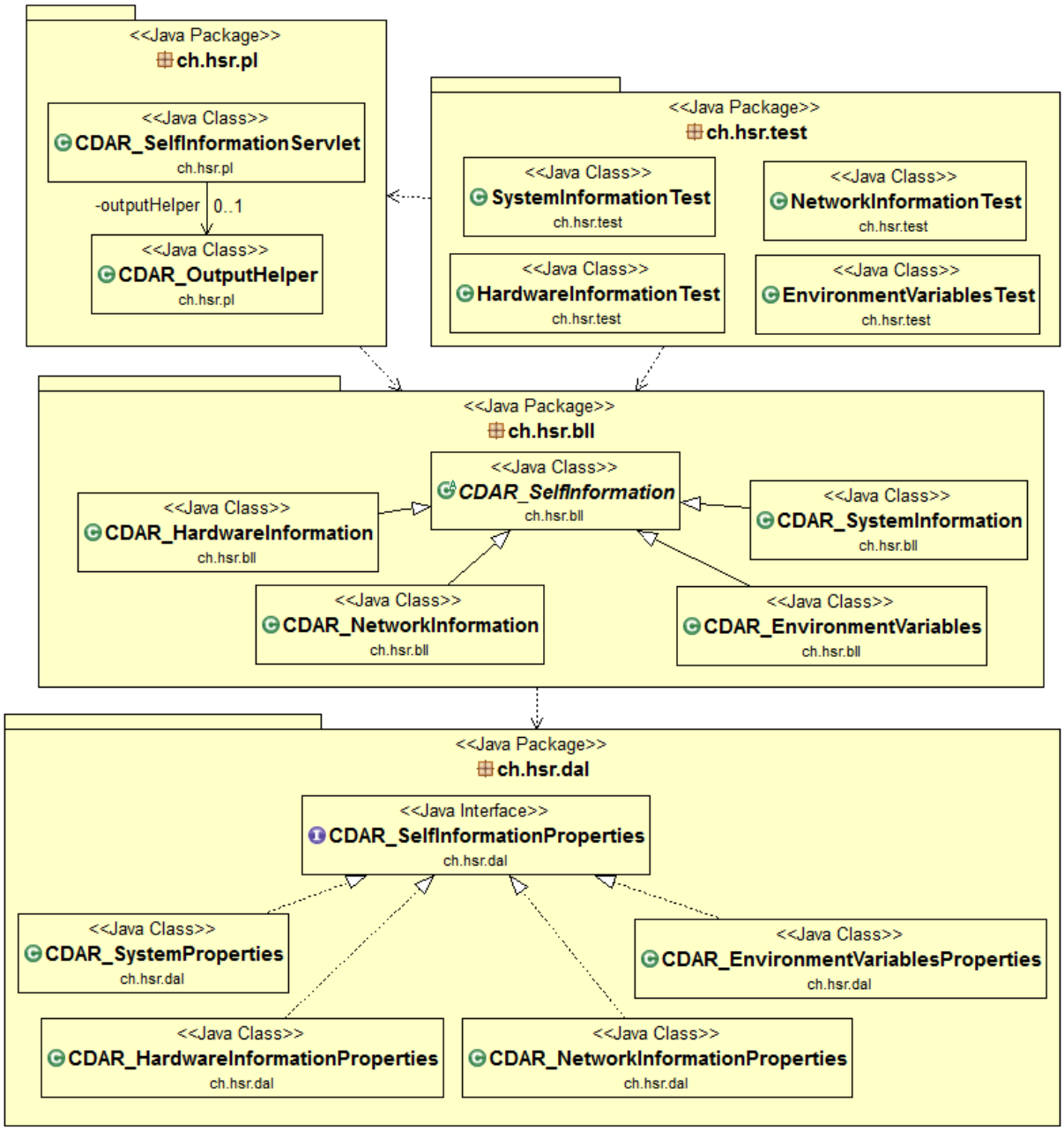


Abbildung 83 - SelfInformation Logische Architektur

E.3.2 ch.hsr.pl

Im Presentation Layer der SelfInformation Applikation befindet sich ein Servlet, welches die Informationen im HTML-Format anzeigt.

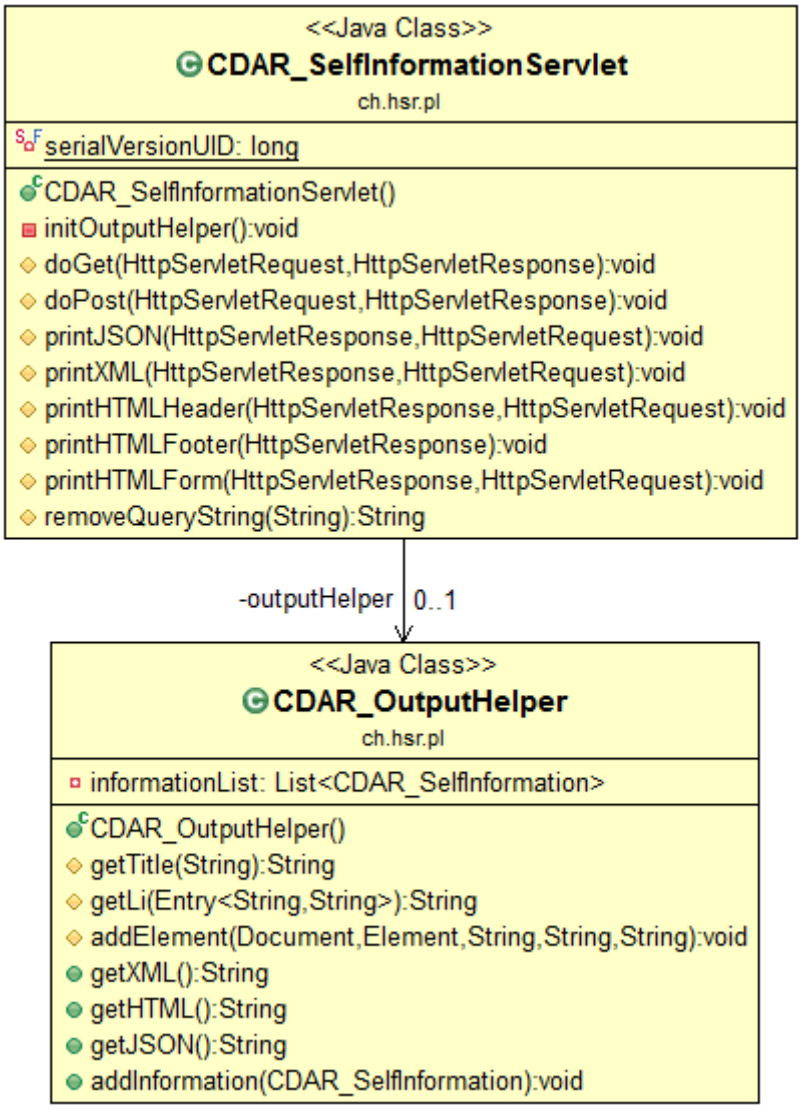


Abbildung 84 - SelfInformation Presentation Layer

E.3.3 ch.hsr.bll

Der Business Logic Layer beinhaltet Plain Old Java Object's welche über die jeweiligen Methoden Informationen zum System anbieten. Zudem ist die Ausgabe im HTML-, XML- sowie JSON-Format möglich.

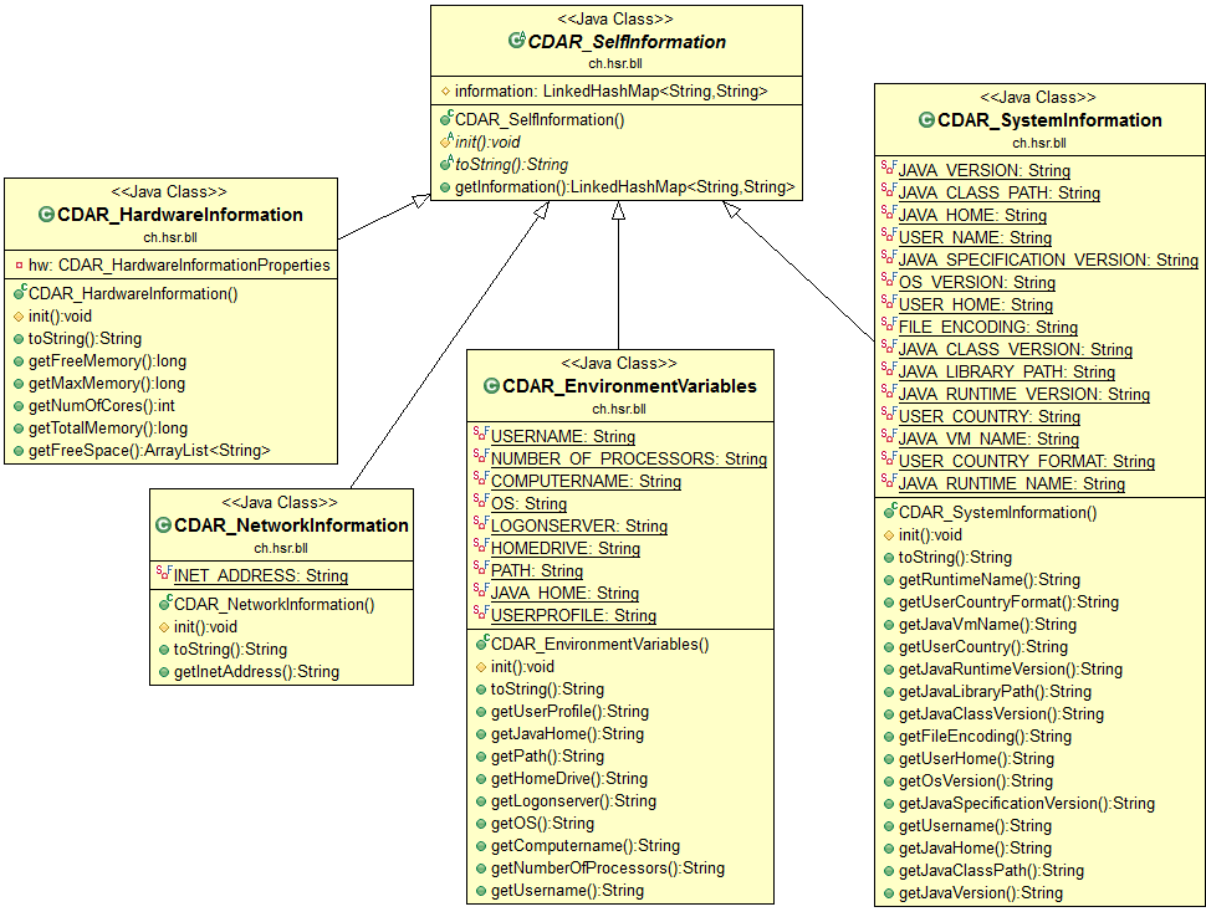


Abbildung 85 - SelfInformation Business Logic Layer

E.3.4 ch.hsr.dal

Im Data Access Layer werden Daten zum System abgefragt und als Map zurückgegeben.

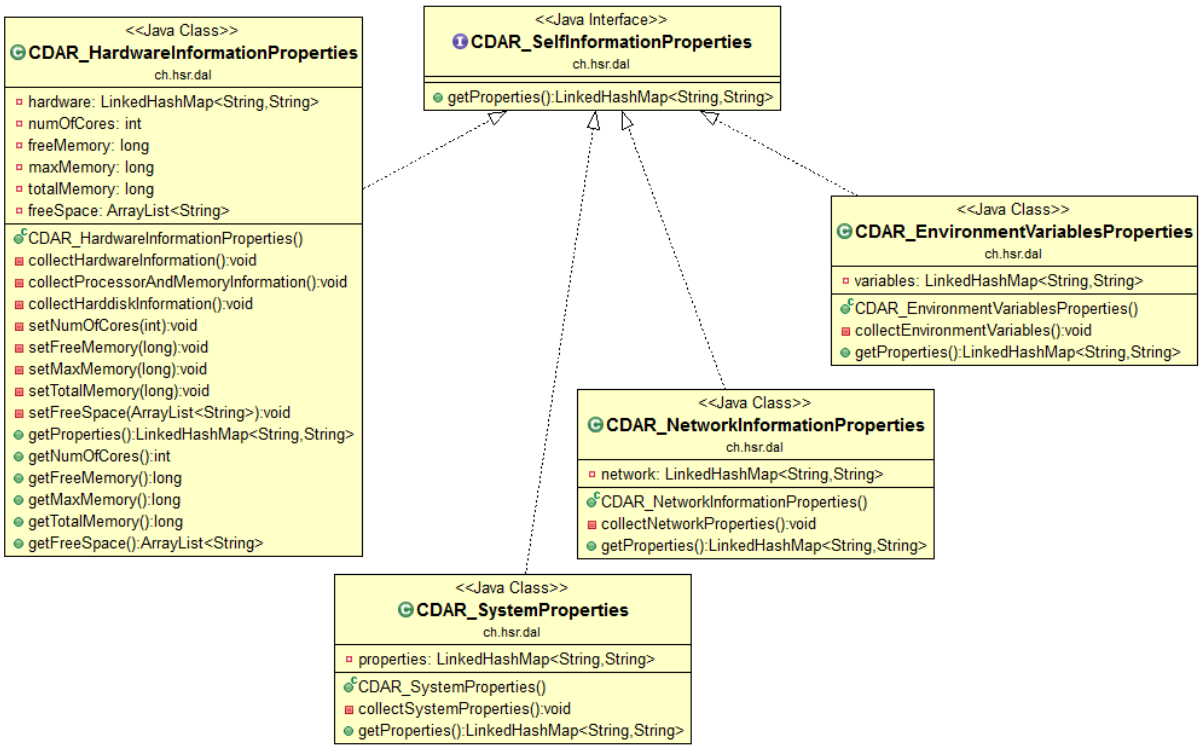


Abbildung 86 - SelfInformation Data Access Layer

E.4 Unit Tests

Um die Funktionalität der Applikation zu testen, existieren folgende Unit Tests. Da die Ausgabe der Informationen auf jedem System variiert, testeten wir lediglich auf Exceptions und die erfolgreiche Rückgabe von Daten.

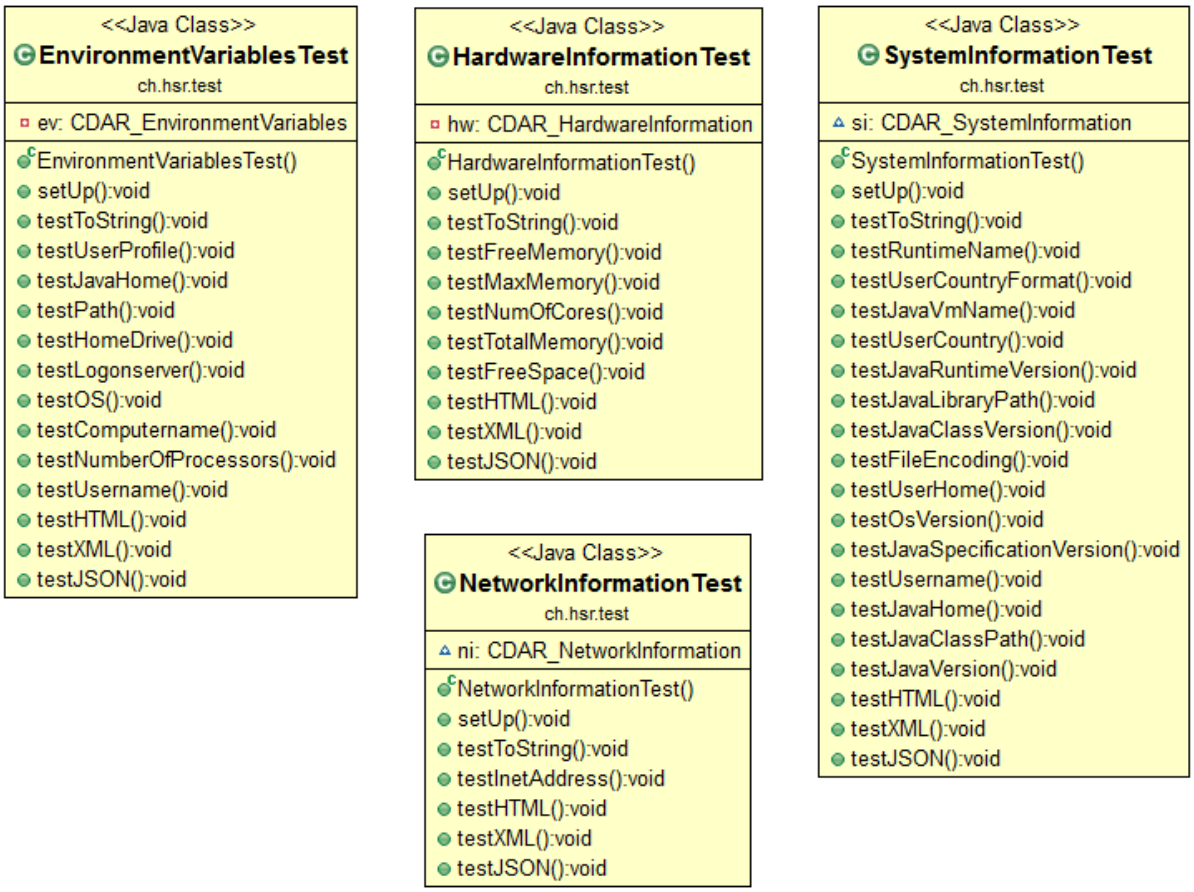


Abbildung 87 - SelfInformation Unit Tests

E.5 Installationsanleitung

Diese Anleitung bezieht sich auf die Projekte SelfInformationCloudBees, SelfInformationGoogle, sowie SelfInformationHeroku.

E.5.1 SelfInformation CloudBees

E.5.1.1 Vorbedingungen

- Applikation bei CloudBees erstellt
- Applikation als WAR-Datei exportiert

E.5.1.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Abschnitt „13.1 War-File Deployment“ beschrieben, deployen.

E.5.1.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/SelfInformationServlet“ nach der Adresse eingeben.

Beispiel:
http://selfinformation-2.dzigerli.cloudbees.net/SelfInformationServlet

E.5.1.4 Zusätzliche Informationen

Den Namen sowie Adresse der Applikation findet sich im Webinterface von CloudBees.

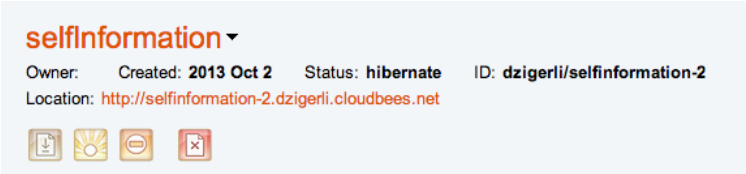


Abbildung 88 - WebInterface CloudBees SelfInformation

E.5.2 SelfInformation Heroku

E.5.2.1 Vorbedingungen

- Heroku Toolbelt installiert

E.5.2.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Abschnitt „12.1 Heroku Toolbelt SDK“ beschrieben, deployen.

E.5.2.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/SelfInformationServlet“ der URL hinzufügen.

Beispiel:
http://informationservlet.herokuapp.com/SelfInformationServlet

E.5.2.4 Zusätzliche Informationen

Den Namen der Applikation findet sich im Webinterface von Heroku.

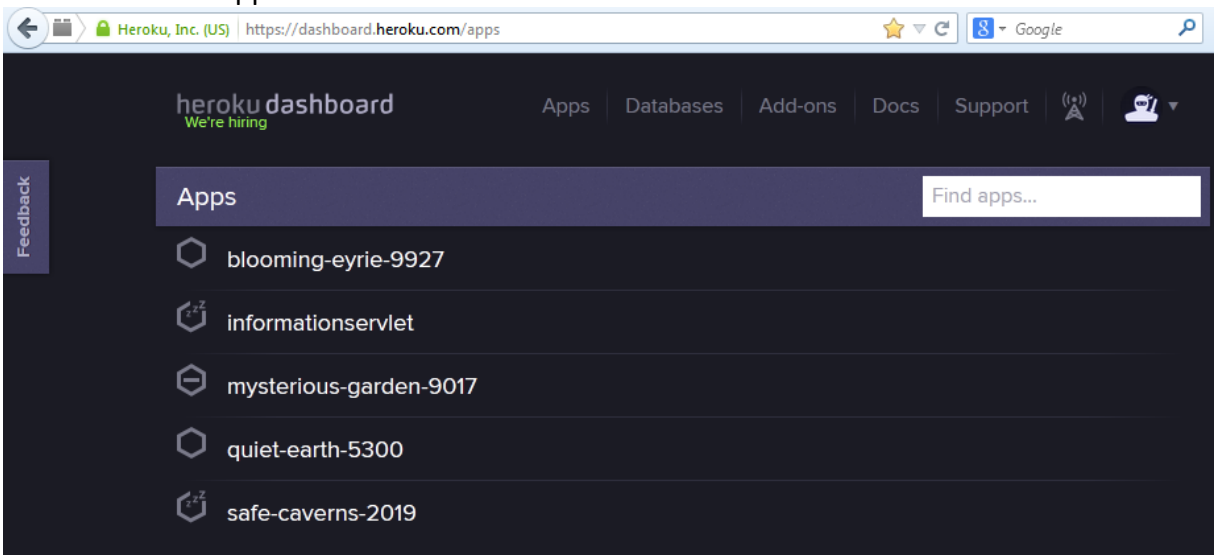


Abbildung 89 - Webinterface Heroku Self Information

E.5.3 SelfInformation Google

E.5.3.1 Vorbedingungen

- Applikation bei Google App Engine erstellt
- Eclipse-Plug-In von Google App Engine installiert

E.5.3.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Unterabschnitt „14.1.4 Anwendung deployen“ beschrieben, deployen.

E.5.3.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/SelfInformationServlet“ nach der URL hinzufügen.

Beispiel:
http://selfinformationhsr.appspot.com/SelfInformationServlet

E.5.3.4 Zusätzliche Informationen

Den Namen der Applikation findet sich im Webinterface von Google App Engine.

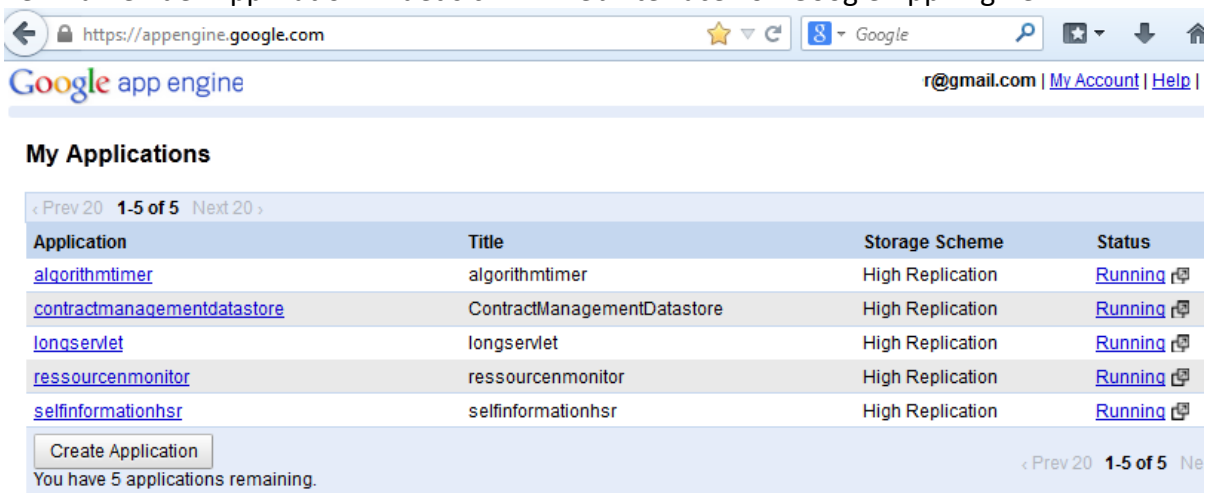


Abbildung 90 - Webinterface Google App Engine Self Information

F ContractManagement

Das folgende Kapitel beschreibt Anforderungen und Use Cases, welche die ContractManagement-Applikation erfüllen muss. Zudem zeigt es die erstellte Struktur des Projektes.

F.1 Use Cases

F.1.1 Use Case Diagramm

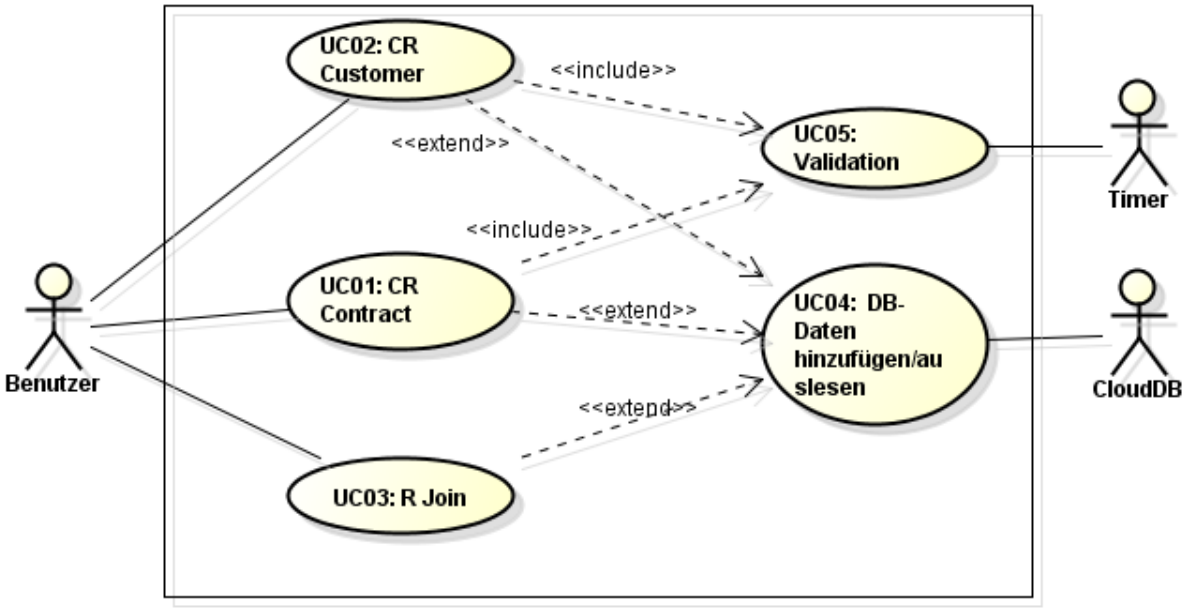


Abbildung 91 - ContractManagement Use Case Diagramm

F.1.2 Aktoren & Stakeholder

- Benutzer
- Clouddienst

F.1.3 Beschreibungen

F.1.3.1 UC01: CR Contract

Der Benutzer fügt einen Contract hinzu und kann sich diese anzeigen lassen.

F.1.3.2 UC02: CR Customer

Der Benutzer fügt einen Customer hinzu und kann sich diese anzeigen lassen.

F.1.3.3 UC03: R Join

Benutzer kann sich einen Join über zwei Tabellen anzeigen lassen.

F.1.3.4 UC04: DB-Daten hinzufügen / auslesen

Das System fügt einen Eintrag in die Datenbank und liest die Einträge aus der Datenbank.

F.1.3.5 UC05: Validation

Das System überprüft die vom Benutzer eingegebenen Daten.

F.2 Nicht funktionale Anforderungen

F.2.1 Angemessenheit

Die Applikationen zeigen wie das Arbeiten mit den drei verschiedenen Datenbanken realisiert wird. Zudem zeigen die Applikationen die Unterschiede der Datenbanken.

F.2.2 Interoperabilität

Die Applikation zeigt die Zusammenarbeit zwischen der Datenbank und der Applikation. Alle Applikationen sind in Java geschrieben, einer Sprache die viele Cloud Provider unterstützen.

F.2.3 Modifizierbarkeit

Die Applikation kann für Erweiterungen wie beispielsweise zusätzliche Spalten bzw. andere Tabellennamen ohne grossen Aufwand umgebaut werden.

F.3 Packages

Die Applikation ist in drei verschiedene Schichten gegliedert.

- Presentation Layer (ch.hsr.pl)
- Business Logic Layer (ch.hsr.bll)
- Data Access Layer (ch.hsr.dal)

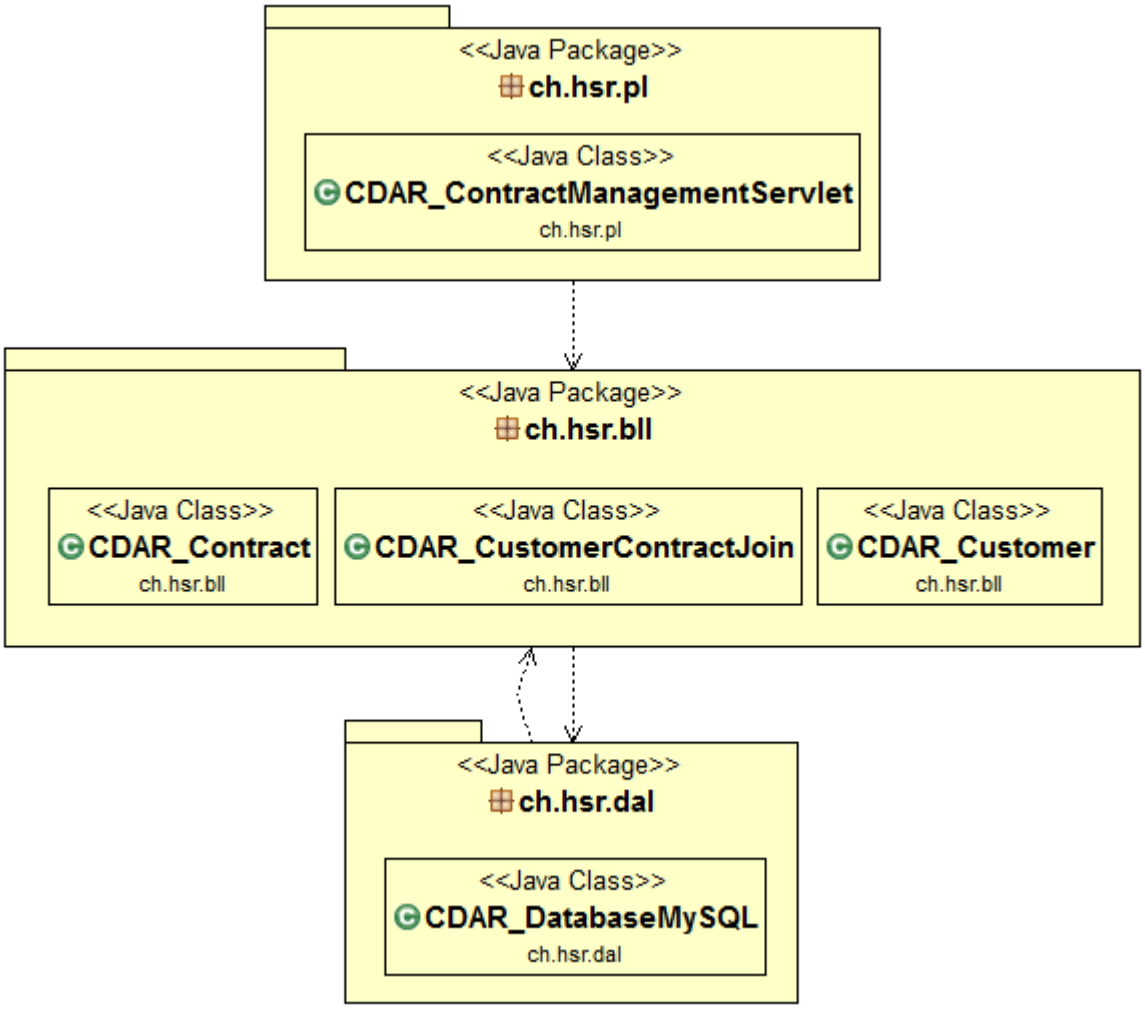


Abbildung 92 - ContractManagement Packages

F.3.1 ch.hsr.pl

In dieser Schicht befindet sich das Servlet. Das Servlet zeigt die Informationen im HTML-Format an.

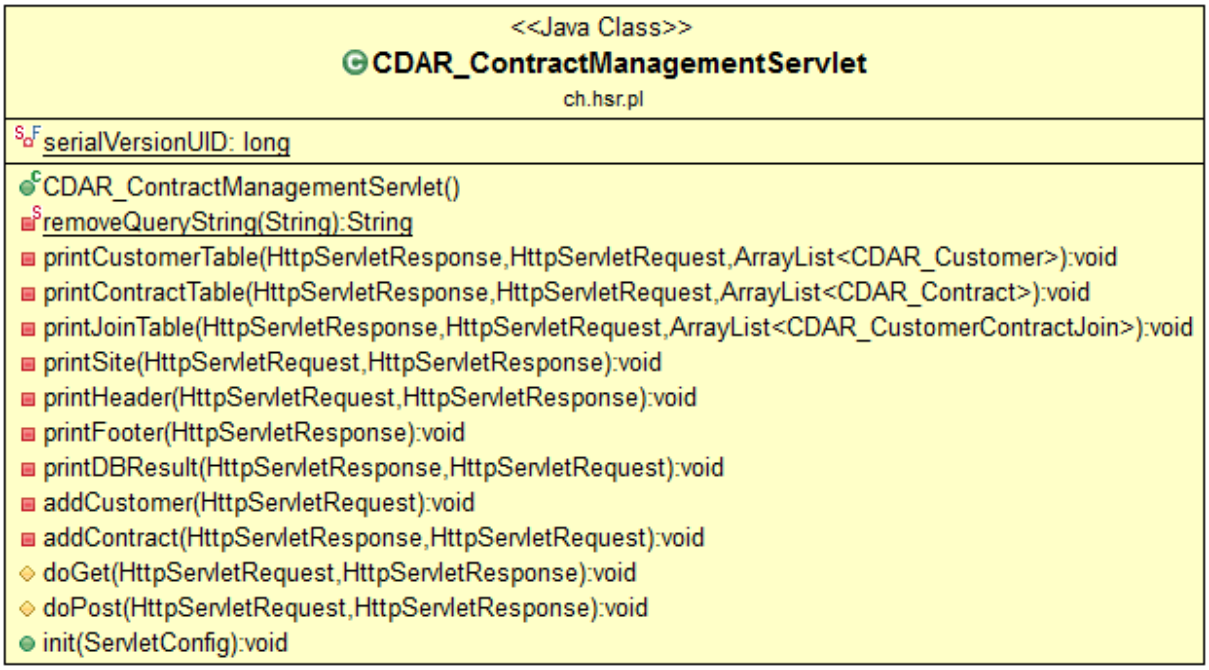


Abbildung 93 - ContractManagement Presentation Layer

F.3.2 ch.hsr.bll

Im Business Logic Layer sind alle Tabellen als Objekte abgebildet. Zudem ist der Join auch als Objekt realisiert. Die Joinklasse hat die Absicht die Join-Abfrage korrekt abzubilden.

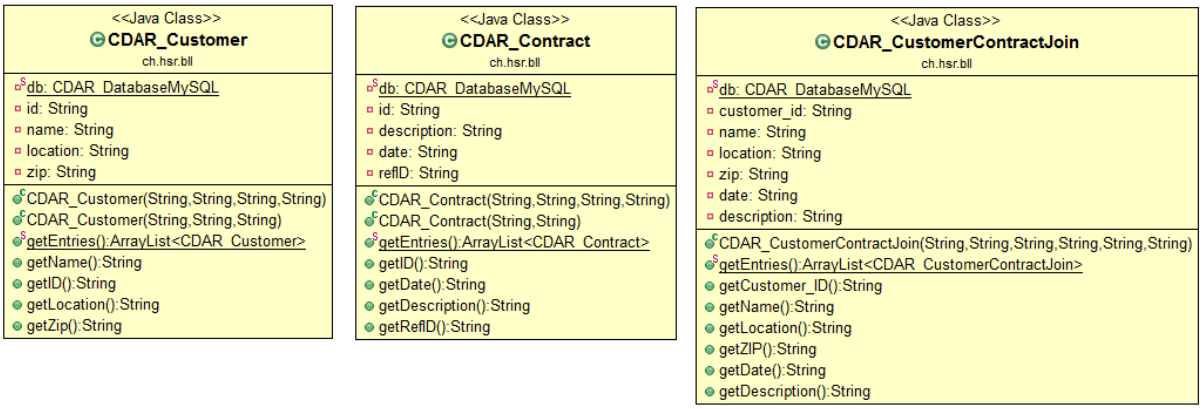


Abbildung 94 - ContractManagement Business Logic Layer

F.3.3 ch.hsr.dal

Im Data Access Layer ist eine reine Kommunikation zwischen der Datenbank und dem Business Logic Layer. In diesem Layer werden die Einträge aus der Datenbank gelesen bzw. Einträge in die Datenbank eingefügt.

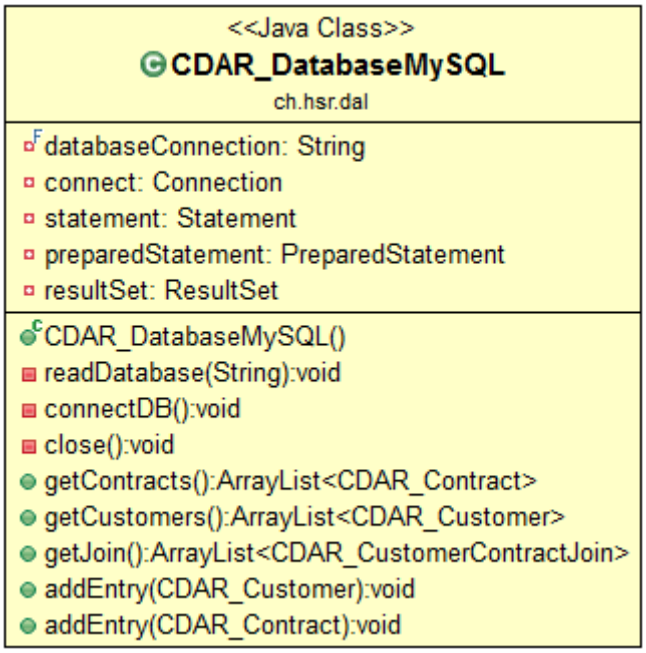


Abbildung 95 - ContractManagement Data Access Layer

F.4 Installationsanleitung

Diese Anleitung bezieht sich auf die Projekte ContractManagement_MySQL, ContractManagement_MongoDB und ContractManagement_Datastore.

F.4.1 ContractManagement MySQL

F.4.1.1 Vorbedingungen

- Applikation bei CloudBees erstellt
- MySQL Datenbank bei CloudBees erstellt
- Applikation als WAR-Datei exportiert
- ContractManagement-Code für die Verbindung zur Datenbank auf eigene Instanz angepasst (Config-File)

F.4.1.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Abschnitt „13.1 War-File Deployment“ beschrieben, deployen.

F.4.1.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/ContractManagementServlet“ nach der Adresse eingeben.

Beispiel:
http://contractmanagementsql.mtinner.eu.cloudbees.net/ContractManagementServlet

F.4.1.4 Zusätzliche Informationen

Den Namen sowie Adresse der Applikation findet sich im Webinterface von CloudBees.

Abbildung 96 - Webinterface CloudBees Contract Management MySQL

Die Anleitung für die Erstellung einer MySQL-Datenbank auf CloudBees ist im Abschnitt „13.3 MySQL Einrichtung“ zu finden.

F.4.2 ContractManagement MongoDB

F.4.2.1 Vorbedingungen

- Applikation bei CloudBees erstellt
- MongoDB Datenbank bei CloudBees erstellt
- Applikation als WAR-Datei exportiert
- ContractManagement-Code für die Verbindung zur Datenbank auf eigene Instanz angepasst (Config-File)

F.4.2.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Abschnitt „13.1 War-File Deployment“ beschrieben, deployen.

F.4.2.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/ContractManagementServlet“ nach der Adresse eingeben.

Beispiel:
http://contractmanagementmdb.dzigerli.eu.cloudbees.net/ContractManagementServlet

F.4.2.4 Zusätzliche Informationen

Den Namen sowie Adresse der Applikation findet sich im Webinterface von CloudBees.

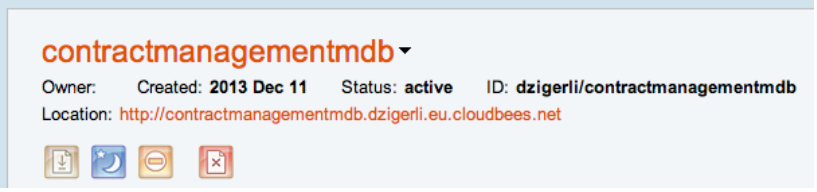


Abbildung 97 - Webinterface CloudBees Contract Management Mongo DB

Die Anleitung für die Erstellung einer MongoDB-Datenbank auf CloudBees ist im Abschnitt „13.4 MongoHQ Einrichtung“ zu finden.

F.4.3 ContractManagement Datastore

F.4.3.1 Vorbedingungen

- Applikation bei Google App Engine erstellt
- Eclipse-Plug-In von Google App Engine installiert

F.4.3.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Unterabschnitt „14.1.4 Anwendung deployen“ beschrieben, deployen.

F.4.3.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/ContractManagementServlet“ nach der URL hinzufügen.

Beispiel:
http://contractmanagementdatastore.appspot.com/ContractManagementServlet

F.4.3.4 Zusätzliche Informationen

Den Namen der Applikation findet sich im Webinterface von Google App Engine.

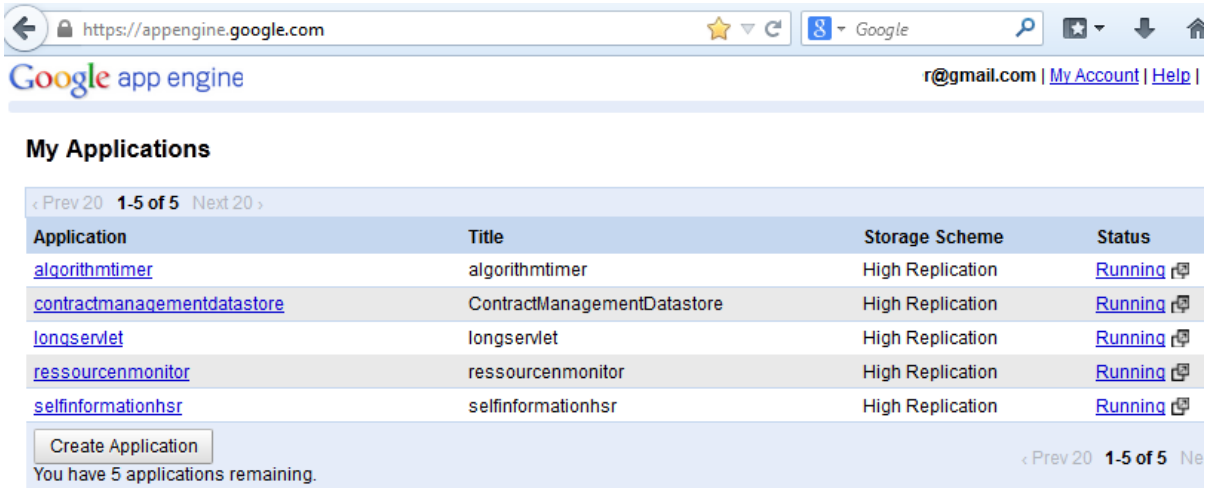


Abbildung 98 - Webinterface Google App Engine Contract Management Datastore

G Daytime

Das folgende Kapitel beschreibt die erstellte Struktur des Daytime-Projektes.

G.1 Packages

Die Applikation enthält eine Schicht (ch.hsr.daytime).

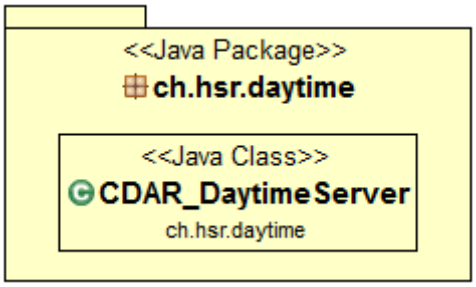


Abbildung 99 - Daytime Package

G.1.1 ch.hsr.daytime

In dieser Schicht befindet sich der Server.

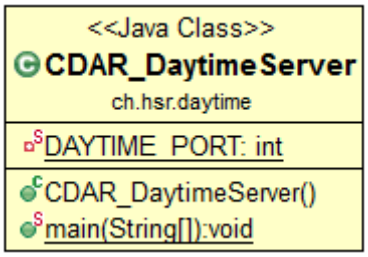


Abbildung 100 - Daytime Layer

G.2 Installationsanleitung

Diese Anleitungen beziehen sich auf die Projekte Daytime_local, DaytimeServerCloudBees und DaytimeServerHeroku. Beschrieben wird die Nutzung eines Sockets bei den beiden PaaS-Providern.

G.2.1 Daytime local CloudBees

G.2.1.1 Vorbedingungen

- CloudBees SDK installiert
- Applikation als ausführbare Jar-Datei mit dem Namen „Daytime.jar“ exportiert

G.2.1.2 Deployment

- Starten der Eingabeaufforderung
- Wechseln ins Verzeichnis der ausführbaren Jar-Datei
- Deployment der Applikation mit folgendem Befehl:

```
1  bees app:deploy -t java -R class=ch.hsr.daytime.Main -R java_version=1.7 Daytime.jar
```

G.2.1.3 Aufruf

Da dieses Projekt eine Java SE-Applikation ist, ist diese nicht direkt aufzurufen. Es aber möglich die Ausgabe der Applikation über das CloudBees-Webinterface anzuzeigen.

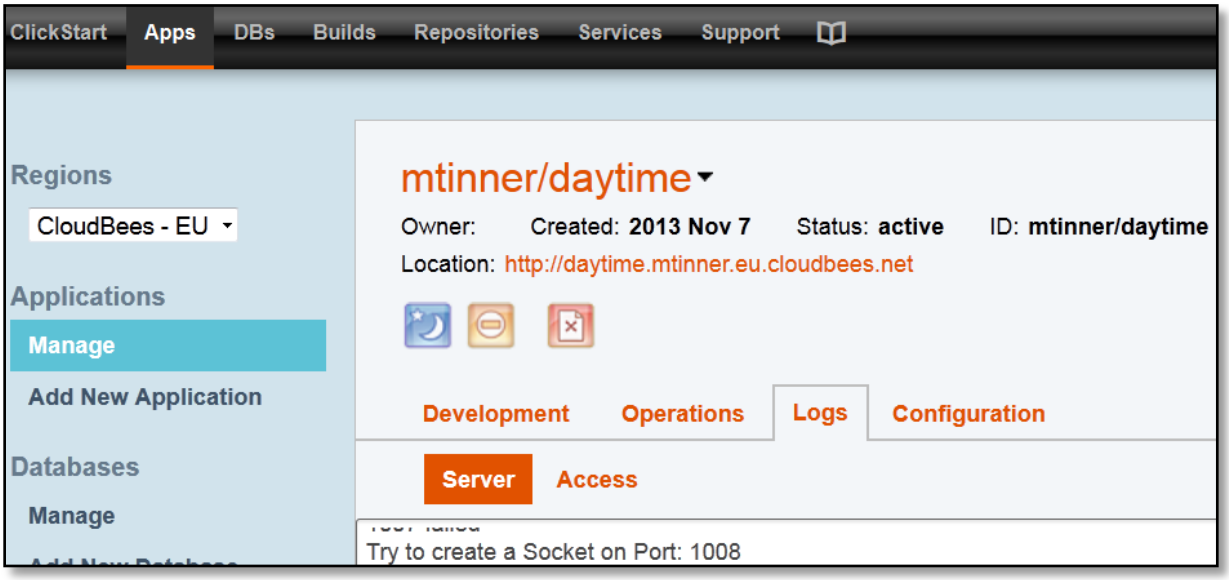


Abbildung 101 - Daytime local Ausgabe

G.2.2 DaytimeServer CloudBees

G.2.2.1 Vorbedingungen

- Applikation bei CloudBees erstellt
- CloudBees SDK installiert
- Applikation als ausführbare Jar-Datei mit dem Namen „DaytimeServerCloudBees.jar“ exportiert

G.2.2.2 Deployment

- Starten der Eingabeaufforderung
- Wechseln ins Verzeichnis der ausführbaren Jar-Datei
- Deployment der Applikation mit folgendem Befehl:

```
1  bees app:deploy -t java -R class=ch.hsr.daytime.CDAR_DaytimeServer -R java_version=1.7
   DaytimeServerCloudBees.jar
```

G.2.2.3 Aufruf

Aufruf der Applikation

- Über den Browser die Adresse der Applikation aufrufen.

G.2.2.4 Zusätzliche Informationen

Den Namen sowie Adresse der Applikation findet sich im Webinterface von CloudBees.

Abbildung 102 - Webinterface CloudBees

G.2.2.5 Ausgabe

Nach erfolgreichem Deployment und Aufruf der Applikationsadresse im Browser, gibt der Server die aktuelle Zeit zurück. Nachfolgend ein Screenshot der Applikation bei CloudBees.

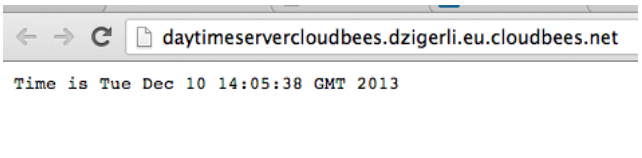


Abbildung 103 - DaytimeServer Ausgabe

G.2.3 DaytimeServer Heroku

G.2.3.1 Deployment

- Starten der Eingabeaufforderungen
- Wechseln ins Projektverzeichnis
- Deployment des Projektes nach Heroku gemäss Anleitung im Unterabschnitt „12.1 Heroku Toolbelt SDK“.

G.2.3.2 Aufruf

Aufruf der Applikation:

- Über den Browser die Adresse der Applikation aufrufen.

G.2.3.3 Ausgabe

Nach erfolgreichem Deployment und Aufruf der Applikationsadresse im Browser, gibt der Server die aktuelle Zeit zurück. Nachfolgend ein Screenshot der Applikation bei CloudBees.

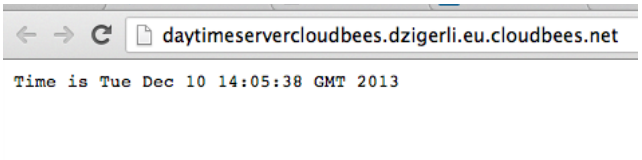


Abbildung 104 - DaytimeServer Ausgabe

H LongServlet

Das folgende Kapitel beschreibt die erstellte Struktur des LongServlet-Projektes.

H.1 Packages

Die Applikation enthält eine Schicht, den Presentation Layer (ch.hsr.pl).

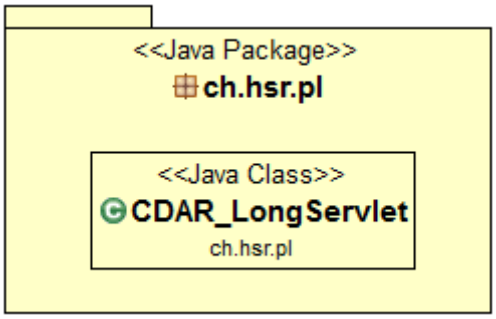


Abbildung 105 - LongServlet Package

H.1.1 ch.hsr.pl

In dieser Schicht befindet sich das Servlet. Das Servlet zeigt die Informationen im HTML-Format an.

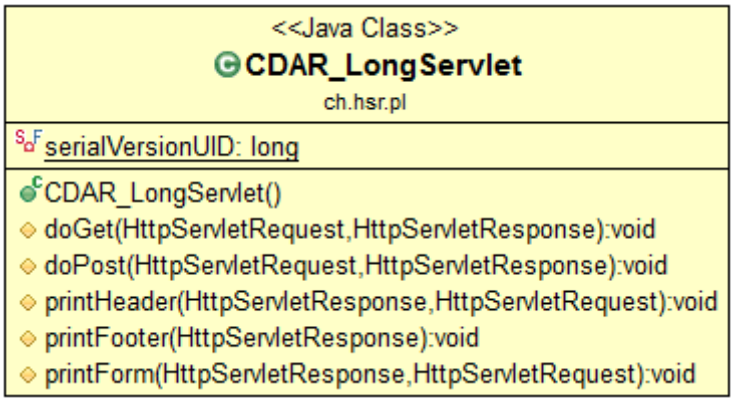


Abbildung 106 - LongServlet Presentation Layer

H.2 Installationsanleitung

Diese Anleitungen beziehen sich auf die Projekte LongServletCloudBees, LongServletGoogle und LongServletHeroku. Beschrieben sind die Vorbedingungen und die Nutzung der Applikation.

H.2.1 LongServlet CloudBees

H.2.1.1 Vorbedingungen

- Applikation bei CloudBees erstellt
- Applikation als WAR-Datei exportiert

H.2.1.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Kapitel Abschnitt „13.1 War-File Deployment“ beschrieben, deployen.

H.2.1.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/LongServlet“ nach der Adresse eingeben.

Beispiel:
http://longservlet.dzigerli.cloudbees.net/LongServlet

Zusätzlich existiert der Parameter „timeToWait“, mit welchem man dem Servlet die Anzahl Sekunden, welche es zu warten hat, mitgeben kann.

Beispiel mit 30 Sekunden Wartezeit:
http://longservlet.dzigerli.cloudbees.net/LongServlet?timeToWait=30

H.2.1.4 Zusätzliche Informationen

Den Namen sowie Adresse der Applikation findet sich im Webinterface von CloudBees.

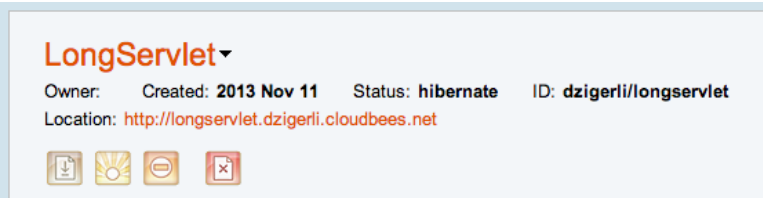


Abbildung 107 - Webinterface CloudBees LongServlet

H.2.2 LongServlet Google

H.2.2.1 Vorbedingungen

- Applikation bei Google App Engine erstellt
- Eclipse-Plug-In von Google App Engine installiert

H.2.2.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Unterabschnitt „14.1.4 Anwendung deployen“ beschrieben, deployen.

H.2.2.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/longservlet“ nach der URL hinzufügen.

Beispiel:
http://longservlet.appspot.com/longservlet

Zusätzlich existiert der Parameter „timeToWait“, mit welchem man dem Servlet die Anzahl Sekunden, welche es zu warten hat, mitgeben kann.

Beispiel mit 30 Sekunden Wartezeit:
http://longservlet.appspot.com/longservlet?timeToWait=30

H.2.2.4 Zusätzliche Informationen

Den Namen der Applikation findet sich im Webinterface von Google App Engine.

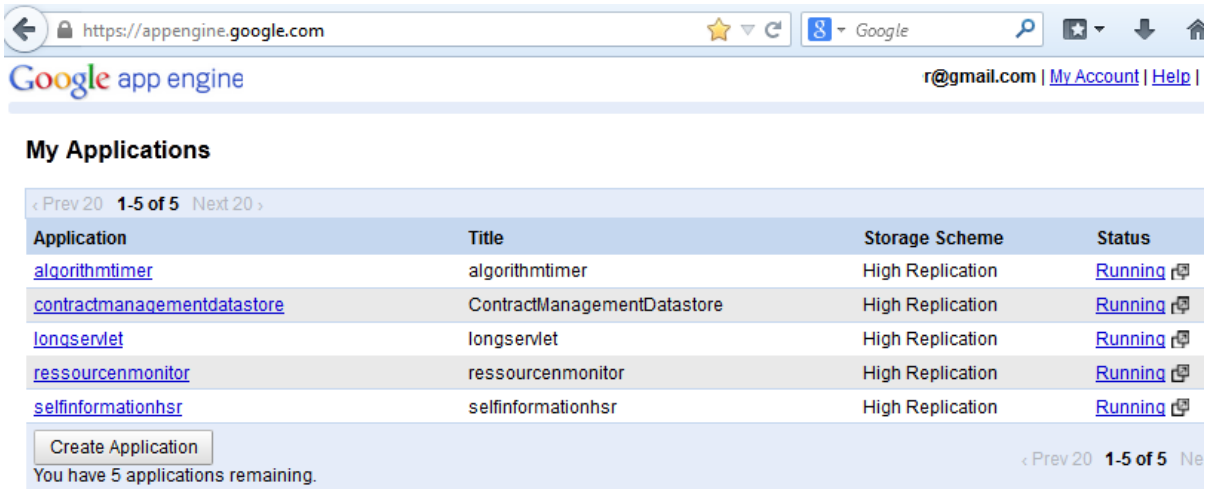


Abbildung 108 - Webinterface Google App Engine LongServlet

H.2.3 LongServlet Heroku

H.2.3.1 Vorbedingungen

- Heroku Toolbelt installiert

H.2.3.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Abschnitt „12.1 Heroku Toolbelt SDK“ beschrieben, deployen.

H.2.3.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/longServlet“ nach der URL hinzufügen.

Beispiel:
http://serene-everglades-9437.herokuapp.com/longServlet

Zusätzlich existiert der Parameter „timeToWait“, mit welchem man dem Servlet die Anzahl Sekunden, welche es zu warten hat, mitgeben kann.

Beispiel mit 30 Sekunden Wartezeit:
http://serene-everglades-9437.herokuapp.com/longServlet?timeToWait=30

H.2.3.4 Zusätzliche Informationen

Den Namen der Applikation findet sich im Webinterface von Heroku.

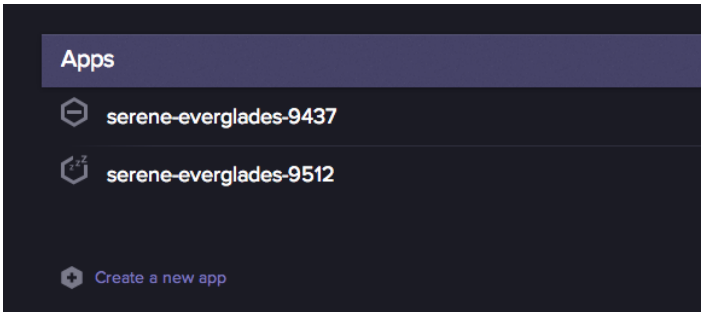


Abbildung 109 - Webinterface Heroku LongServlet

I InsuranceAgentWay

Das folgende Kapitel beschreibt die Package-Struktur der InsuranceAgentWay-Applikation.

I.1 Packages

Die Applikation ist in drei verschiedene Schichten gegliedert.

- Presentation Layer (ch.hsr.pl)
- Business Logic Layer (ch.hsr.bll)
- Data Access Layer (ch.hsr.dal)

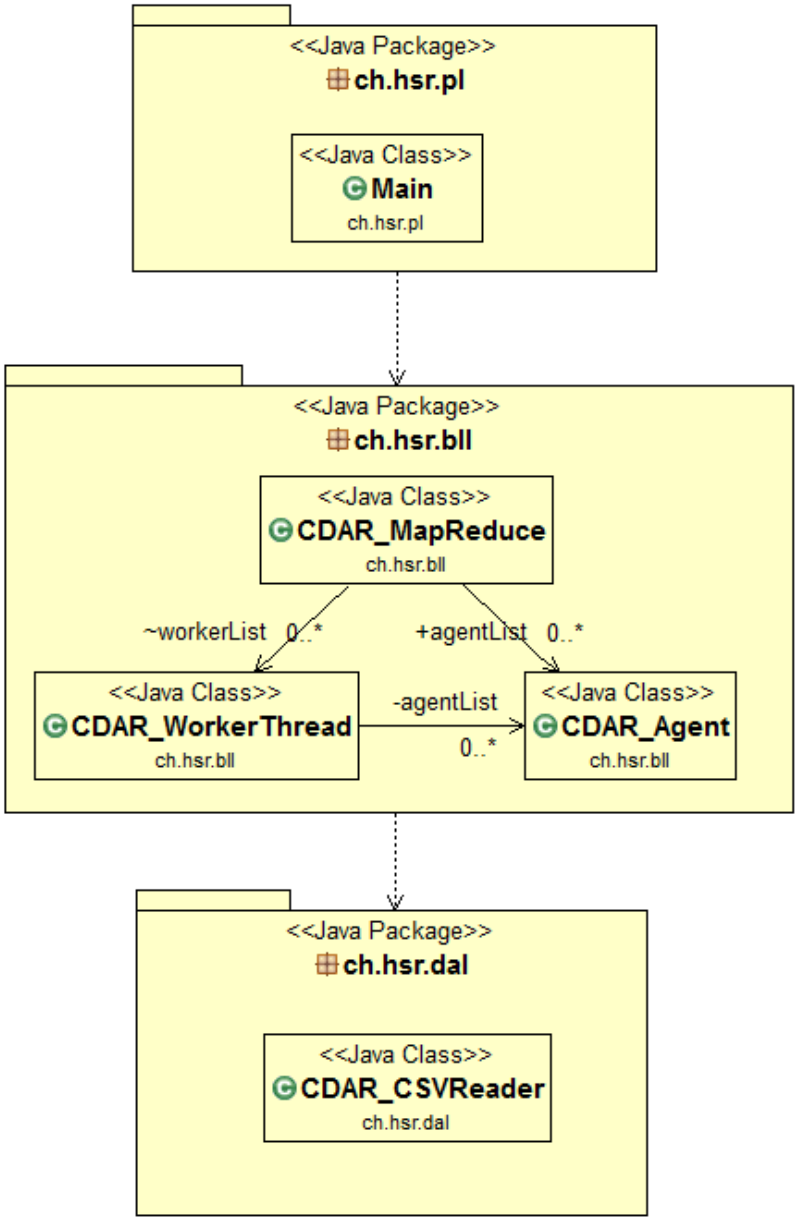


Abbildung 110 - Map Reduce Pattern Packages

I.1.1 ch.hsr.pl

In dieser Schicht befindet sich die Ausgabe. Die Main-Klasse startet das ganze Map Reduce, gibt aber auch alle Resultate, die benötigte Zeit sowie die Anzahl der verwendeten Threads aus.

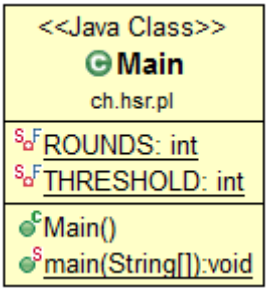


Abbildung 111 - Map Reduce Pattern: Presentation Layerch.hsr.bll

I.1.2 ch.hsr.bll

Im Business Logic Layer befindet sich der eigentliche Algorithmus von Map Reduce. Nachdem alle Daten vom Data Access Layer übergeben wurden, werden in der Klasse CDAR_MapReduce alle Einträge in Gruppensätze aufgeteilt. Jedem dieser Gruppensätze wird danach ein eigener Worker-Thread zugeteilt. Dieser Thread erstellt aus dem Datensatz eine Map(Map-Funktion). Anschliessend werden alle Datensätze wieder zusammengetragen und zu einer grossen gemeinsamen Map zusammengeführt(Reduce-Funktion). Die CDAR_Agent-Klasse spielt dabei die Basisrolle, da es sich bei den einzelnen Einträgen um Versicherungsvertreter handelt.

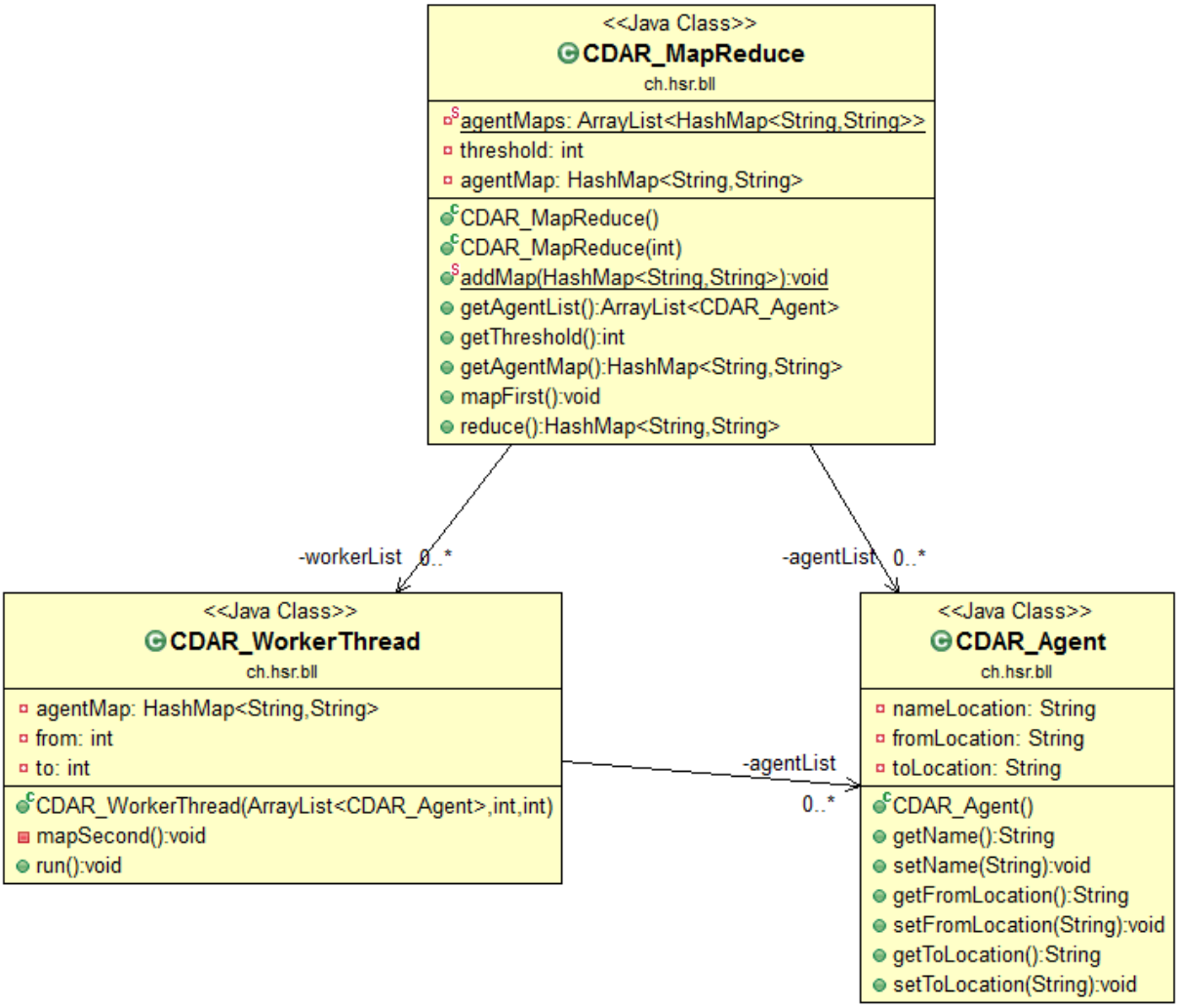


Abbildung 112 - Map ReducePattern: Business Logic Layer

I.1.3 ch.hsr.dal

Im Data Access Layer befindet die CSV Datei mit den Agenten sowie deren einzelne zurückgelegten Strecken. Ausserdem befindet sich ein CSV-Reader der alle Einträge aus der CSV-Datei ausliest.

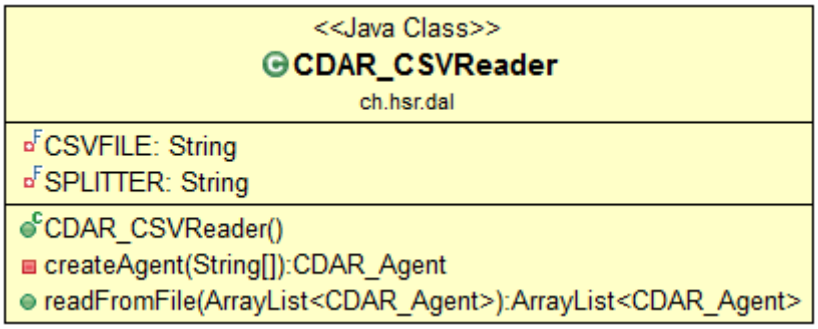


Abbildung 113 - Map Reduce Pattern: Data Access Layer

I.2 Installationsanleitung

Diese Anleitung bezieht sich auf das Projekt InsuranceAgentWay.

I.2.1.1 Vorbedingungen

- CloudBees SDK installiert
- Applikation als ausführbare Jar-Datei mit dem Namen „InsuranceAgentWay.jar“ exportiert

I.2.1.2 Deployment

- Starten der Eingabeaufforderung
- Wechseln ins Verzeichnis der ausführbaren Jar-Datei
- Deployment der Applikation mit folgendem Befehl:

```
2  bees app:deploy -t java -R class=ch.hsr.pl.Main -R java_version=1.7 InsuranceAgentWay.jar
```

I.2.1.3 Aufruf

Da es sich bei diesem Projekt um eine Java SE-Applikation handelt, ist diese nicht direkt aufzurufen. Es aber möglich die Ausgabe der Applikation über das CloudBees-Webinterface anzuzeigen.

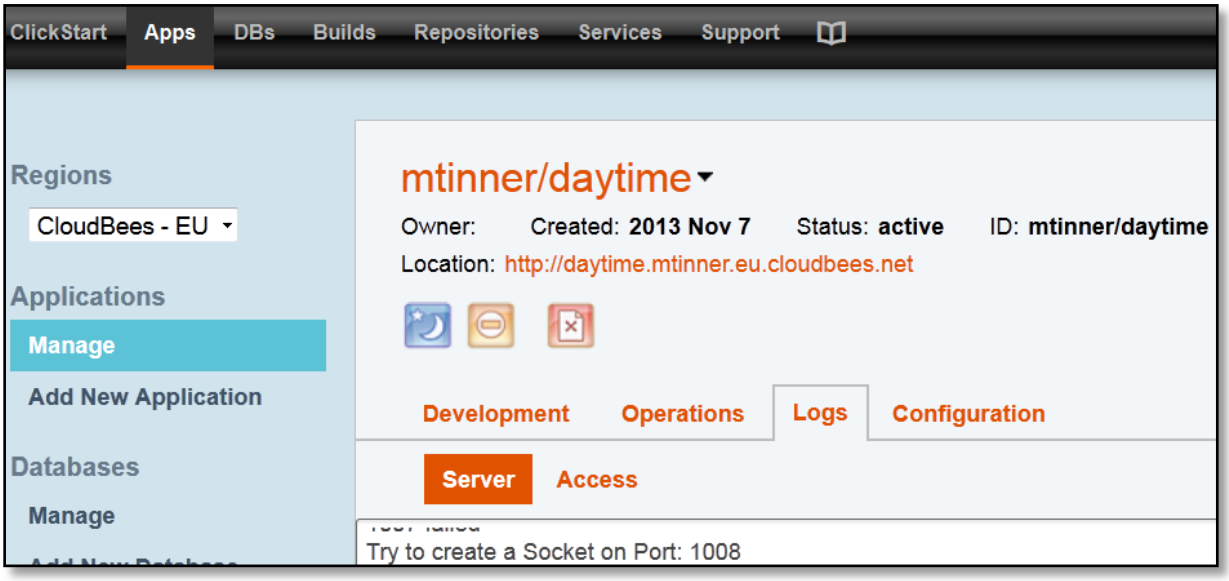


Abbildung 114 - InsuranceAgentWay local Ausgabe

J AlgorithmTimer

Das folgende Kapitel beschreibt die erstellte Struktur des AlgorithmTimer-Projektes.

J.1 Packages

Die Applikation ist in zwei verschiedene Schichten gegliedert.

- Presentation Layer (ch.hsr.pl)
- Business Logic Layer (ch.hsr.bll)

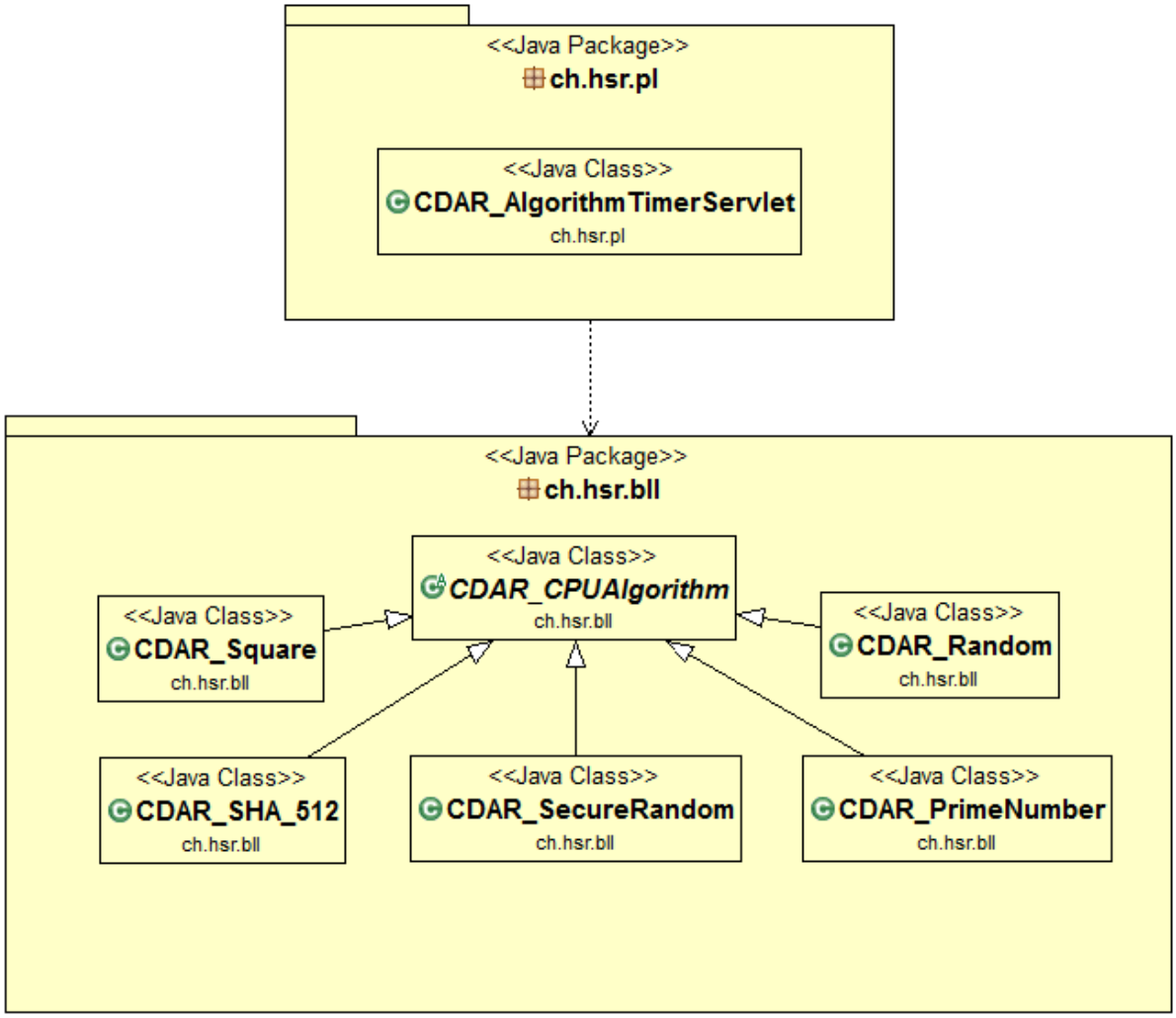


Abbildung 115 - AlgorithmTimer Packages

J.1.1 ch.hsr.pl

In dieser Schicht befindet sich das Servlet. Das Servlet zeigt die Informationen im HTML-Format an.

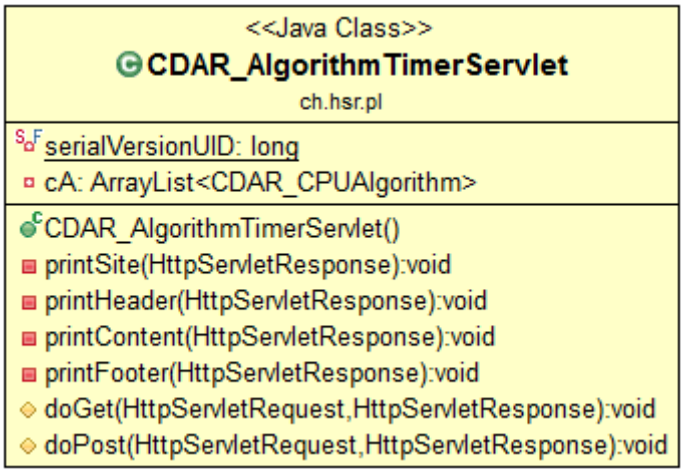


Abbildung 116 - AlgorithmTimer Presentation Layer

J.1.2 ch.hsr.bll

Im Business Logic Layer sind alle Algorithmen definiert. In dieser Schicht wird der Algorithmus durchlaufen und die dazu benötigte Zeit berechnet.

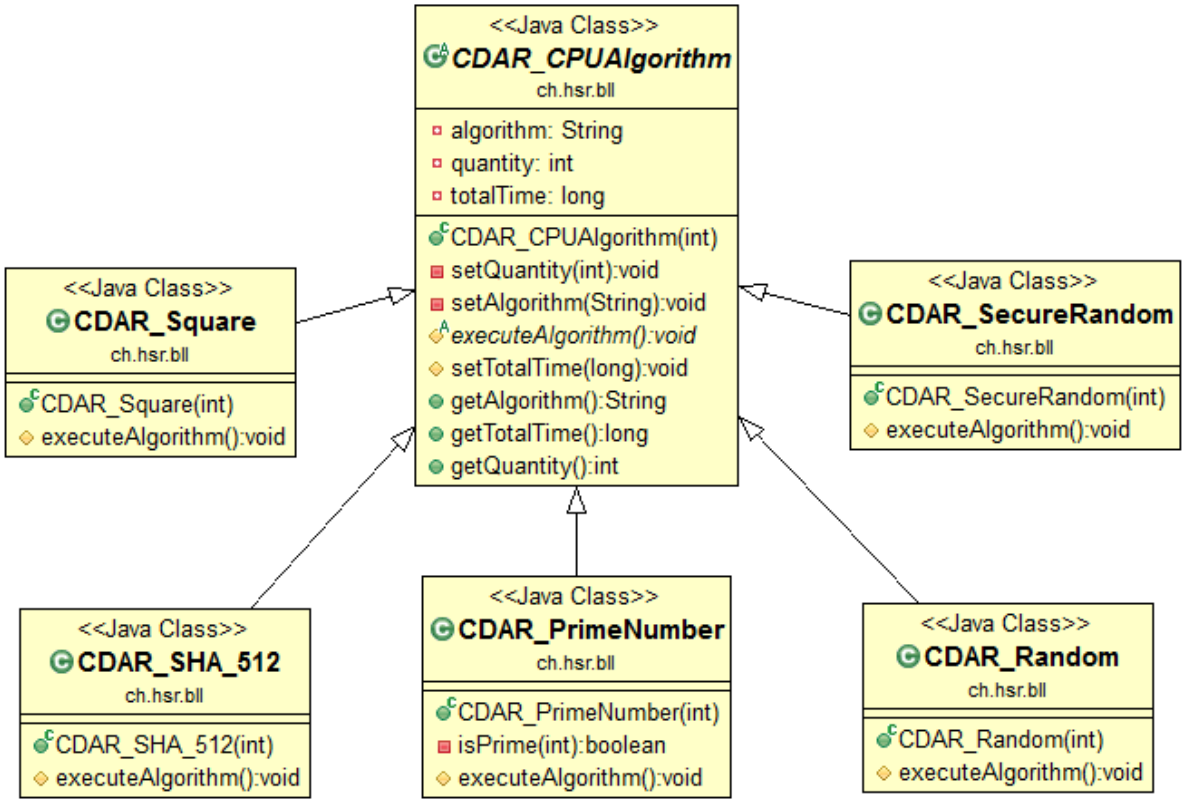


Abbildung 117 - AlgorithmTimer Business Logic Layer

J.2 Installationsanleitung

Diese Anleitungen beziehen sich auf die Projekte AlgorithmTimer_CloudBees, AlgorithmTimer_Google und AlgorithmTimer_Heroku.

J.2.1 AlgorithmTimer CloudBees

J.2.1.1 Vorbedingungen

- Applikation bei CloudBees erstellt
- Applikation als WAR-Datei exportiert

J.2.1.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Abschnitt „13.1 War-File Deployment“ beschrieben, deployen.

J.2.1.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/AlgorithmTimer“ nach der Adresse eingeben.

Beispiel:
http://algorithmtimer.mtinner.cloudbees.net/AlgorithmTimer

J.2.1.4 Zusätzliche Informationen

Den Namen sowie Adresse der Applikation findet sich im Webinterface von CloudBees.

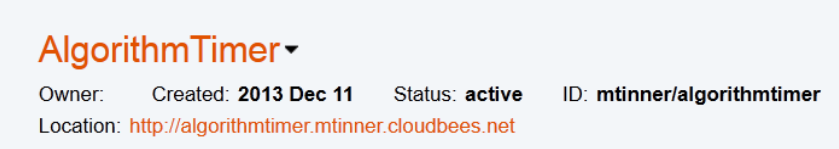


Abbildung 118 - Webinterface CloudBees AlgorithmTimer

J.2.2 AlgorithmTimer Google

J.2.2.1 Vorbedingungen

- Applikation bei Google App Engine erstellt
- Eclipse-Plug-In von Google App Engine installiert

J.2.2.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Unterabschnitt „14.1.4 Anwendung deployen“ beschrieben, deployen.

J.2.2.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/algorithmtimer“ nach der URL hinzufügen.

Beispiel:
http://algorithmtimer.appspot.com/algorithmtimer

J.2.2.4 Zusätzliche Informationen

Den Namen der Applikation findet sich im Webinterface von Google App Engine.

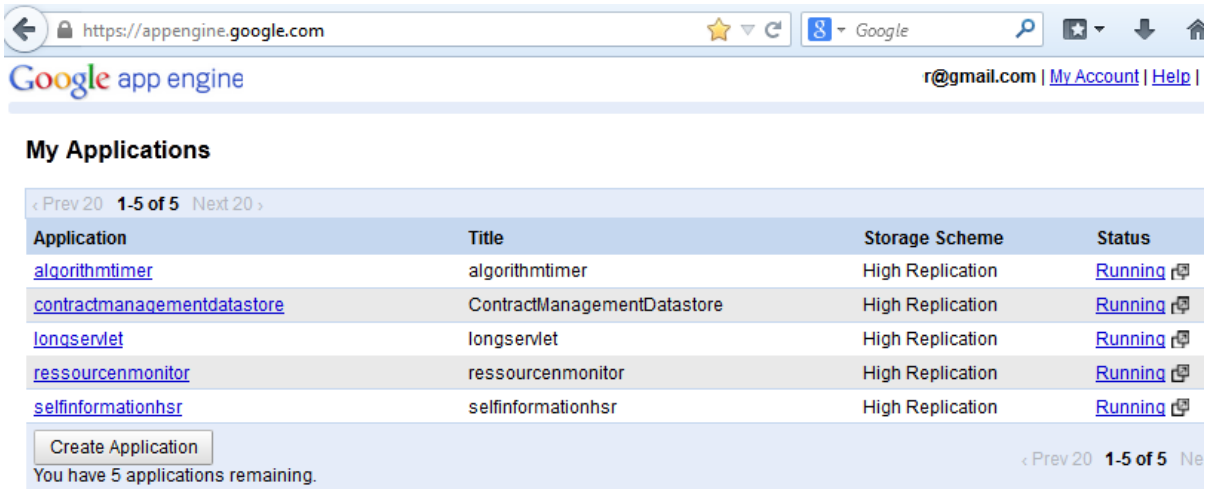


Abbildung 119 - Webinterface Google App Engine AlgorithmTimer

J.2.3 AlgorithmTimer Heroku

J.2.3.1 Vorbedingungen

- Heroku Toolbelt installiert

J.2.3.2 Deployment

Nach dem Import der Applikation (siehe 15.2 Import bestehender Projekte), kann man sie wie im Abschnitt „12.1 Heroku Toolbelt SDK“ beschrieben, deployen.

J.2.3.3 Aufruf

Um das Servlet aufzurufen, die Erweiterung „/AlgorithmTimer“ der URL hinzufügen.

Beispiel:
http://quiet-earth-5300.herokuapp.com/AlgorithmTimer

J.2.3.4 Zusätzliche Informationen

Den Namen der Applikation findet sich im Webinterface von Heroku.

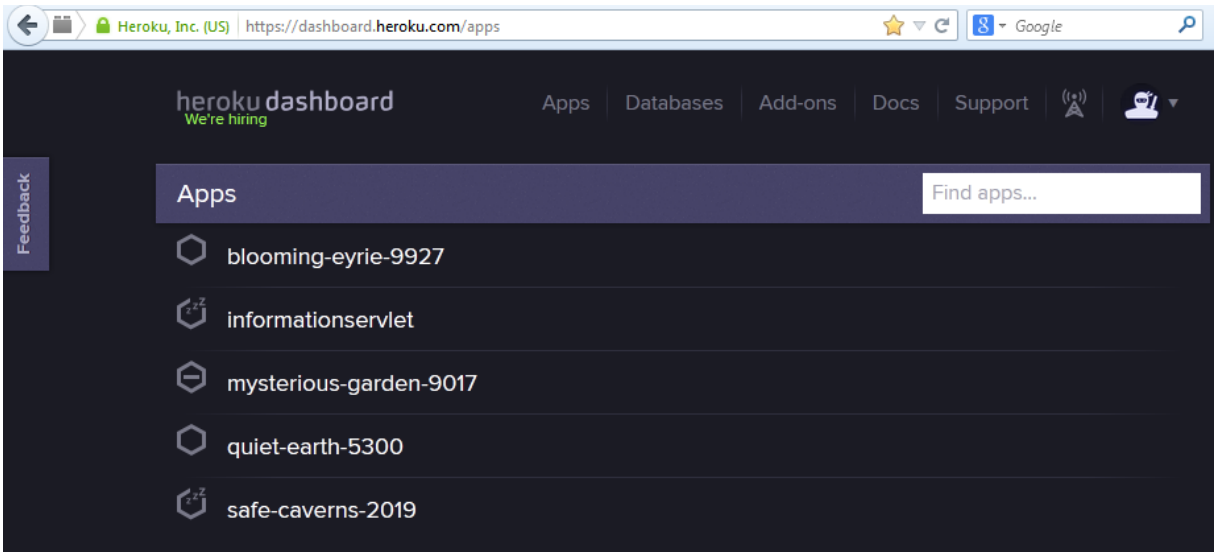


Abbildung 120 - Webinterface Heroku Algorithm Timer

K Applikationsinformationen

In folgenden Absätzen sind die Versionsnummern aller in dieser Arbeit verwendeten Programme, Libraries und Plug-Ins aufgelistet. Ebenso ist eine Übersicht mit den „Lines of Code“ der jeweiligen Applikationen zu finden.

K.1 Versionsübersicht

In untenstehender Tabelle sind alle genutzen Applikations-, Libraries- und Pluginversionen aufgelistet.

Softwarekomponente	Beschreibung / Information	Version
Java	JDK	1.7.0_40
Google App Engine	letzter Zugriff	20.12.2013
	SDK	1.8.5
CloudBees	letzter Zugriff	20.12.2013
	SDK (Toolbelt)	1.5.2
	Driver (Toolbelt)	1.3.6
	Tomcat	7.1
Heroku	letzter Zugriff	20.12.2013
	Toolbelt	3.1.0
Pom-Files	Tomcat	7.0.34
	Appassembler-maven-plugin	1.1.1
	maven-dependency-plugin	2.4
	maven-compiler-plugin	3.1
	maven.compiler.source	1.7
	maven.compiler.target	1.7
	maven-war-plugin	2.3
	org.apache.httpcomponents	4.0.1
	servlet-api	2.5
	jetty-servlet	7.6.0.v20120127
	jetty-webapp	7.6.0.v20120127
	jsp-2.1-glassfish	2.1.v20100127
	Java	1.6
Eclipse	Java EE Developers	4.3.0.v20130605-2000
	Maven Plugin	1.0.0.20130612-1742
	Google Plugin	3.4.2.v201310081834-rel-r43
	Heroku Plugin	1.0.2.201309271210
	ObjectAid Plug-In	1.1.4
	Metrics	1.3.6
Json.org	Json Library für Java	20.12.2013
Maven	Build-Management-Tool	3.1.1

Tabelle 7 - Versionsübersicht

K.2 Lines of Code

Folgende Tabelle zeigt die in dieser Arbeit programmierten Applikationen mit den zugehörigen „Lines of Code“.

Applikation	Lines of Code
SelfInformation CloudBees	812
SelfInformation Google	426
SelfInformation Heroku	558
AlgorithmTimer CloudBees	179
AlgorithmTimer Goole	176
AlgorithmTimer Heroku	201
InsuranceAgentWay	199
ContractManagement MySQL	421
ContractManagement MongoDB	458
ContractManagement Datastore	408
Daytime local	92
DaytimeServer CloudBees	50
DaytimeServer Heroku	50
LongServlet CloudBees	56
LongServlet Google	54
LongServlet Heroku	49
RessourcenMonitor Google	379
Total	*3330*

Tabelle 8 - Lines of Code