Cloud Software Plattformen

Markus Hütter

Januar 2010

Betreuer:
Dr. Andreas Thor
David Aumüller MSc

Seminar Cloud Data Management
Universität Leipzig
Institut für Informatik

Inhaltsverzeichnis

1 Einleitung **2**

2 Google App Engine **4**
2.1 Features 4
2.2 Anwendungsszenario 5

3 Weitere Cloud Software Plattformen **10**
3.1 Microsoft Windows Azure 10
3.2 Force.com 12
3.3 Vergleich 13

4 Zusammenfassung **14**

Literaturverzeichnis **14**

A **16**
A.1 app.yaml 16
A.2 application.py 16

Kapitel 1

Einleitung

Cloud Computing ist ein nebulöser Begriff, der die unterschiedlichsten Bedeutungen kennt. Eine treffende Definition liefert Simon Wardley auf der OSCON 09[1]:

> Cloud Computing ist ein allgemeiner Begriff, der eine grundlegende Veränderung in der IT zu einer Service orientierten Wirtschaft, beschreibt. [...]

Wardley hebt dabei die Analogie von Cloud Computing zur Industriellen Revolution hervor.[2]

Cloud Computing kann man in verschiedene Schichten einteilen:[3]

SaaS - Software as a Service Software die in der Cloud liegt und direkt für den Endkunden bereitgestellt wird. Für diesen entfällt die lokale Software-Installation und damit auch die Bereitstellung erforderlicher Ressourcen.

PaaS - Platform as a Service Richtet sich an Entwickler und bietet diesen eine Plattform für die sie Software entwickeln. Die Ausführung der Applikationen erfolgt in einer Laufzeitumgebung, die von der Programmierumgebung entkoppelt ist.

IaaS - Infrastructure as a Service Dem Benutzer wird eine Service orientierte, abstrahierte Sicht auf Hardware (Rechner, Massenspeicher, Netz-

[1] OSCON: Open Source Convention
[2] vgl. Vortrag *Cloud Computing - Why IT Matters* von Simon Wardley
[3] vgl. [BKNT09]

werke, ...) angeboten wobei er die Menge von bereitgestellten Ressourcen verwalten kann.

Diese Arbeit befasst sich vor allem mit dem Bereich Platform as a Service (PaaS) — insbesondere mit den drei Cloud Software Plattformen *Google App Engine*, *Microsoft Windows Azure* und *Force.com*. Der Hauptaugenmerk der Arbeit liegt jedoch bei *Google App Engine*. Es werden sowohl die technologischen Aspekte, als auch das Anwendungsszenario, eine kleine Applikation in der Cloud zur Verfügung zu stellen, beschrieben. Im Anschluss daran werden die beiden weiteren Cloud Software Plattformen *Microsoft Windows Azure* und *Force.com* vergleichend dazu vorgestellt und zum Schluss eine Gesamtbetrachtung durchgeführt.

Kapitel 2

Google App Engine

Google App Engine ist eine Cloud Software Plattform, welche von Google seit 2008 als PaaS zur Verfügung gestellt wird. Google App Engine stellt dem Anwendungsentwickler verschiedene Features zur Verfügung auf die im folgenden eingegangen wird.[1]

2.1 Features

Skalierbare Infrastruktur - Google stellt einen gewissen Teil Rechenkapazität, Speicherplatz oder Datenbanktransfers u.ä. kostenlos zur Verfügung. Werden diese Kapazitäten (*Quotas*) überschritten, kann der Anwendungsentwickler entscheiden, sich entsprechend mehr zu erkaufen. Das *mehr* an Kapazität liefert Google mit riesigen Serverfarmen und automatischer Lastverteilung, was für den Anwendungsentwickler transparent geschieht.

Python / Java - Google unterstützt für die Anwendungsentwicklung sowohl Python, als auch Java und erreicht damit eine große Entwicklergemeinde.

SDK - Google bietet eine umfangreiche Dokumentation zur Entwicklung von Anwendungen für Google App Engine und stellt auch ein Software Development Kit (SDK) bereit, mit dem Entwickler ihre Anwendungen testen und sehr einfach in die Cloud *deployen*[2] können (siehe

[1] vgl. [Goo10]

[2] die Applikation wird als Paket auf Google's Server hochgeladen

Kapitel 2.2).

Webbasierte Administrationskonsole - eine Anwendung kann über die Administrationskonsole überwacht werden — es werden Benutzungsdaten geliefert, es können Datenbanken eingesehen, Fehler oder Engpässe angezeigt und die verbrauchten Ressourcen gelistet werden (siehe Kapitel 2.2).

Persistenter Speicher - Google bietet dem Anwendungsentwickler Möglichkeiten sehr einfach auf persistenten Speicher auf Googles Servern zuzugreifen. Dies erfolgt über Tabellen mit der Abfragesprache GQL (Google Query Language).

Benutzerauthentifikation - Es werden API's[3] zur Authentifikation von Benutztern über Google Accounts zur Verfügung gestellt.

Tasks - Man kann Tasks definieren, welche im Hintergrund, unabhängig von Webrequests[4], Aufgaben erledigen und bspw. zeitgesteuert ausgeführt werden.

Image manipulation, URL Fetch, Memcache, Mail, ... - Dem Anwendungsentwickler stehen noch viele weitere nützliche API's zur Verfügung, so z.B. zur Bildmanipulation oder Nachrichtenaustausch per Email.

2.2 Anwendungsszenario

Als Teil dieser Arbeit wurde eine kleine Webanwendung geschrieben, um den Arbeitsablauf und die Komplexität der Anwendungsentwicklung für Google App Engine zu untersuchen. Ziel der Anwendung war es, zum einen die CPU Auslastung zu testen. Dafür wurde ein Algorithmus ausgeführt, der die Kreiszahl π berechnet (siehe Abbildung 2.1). Zum anderen galt es, einige der zur Verfügung stehenden Techniken zu benutzen, wie z.B. den persistenten Speicher und die Abfragesprache GQL. Hierfür wurde ein Algorithmus verwendet, der vom Benutzer eingegebene Wörter über deren Trigramme[5] vergleicht und die Ähnlichkeit bewertet (siehe Ab-

[3] Application Programming Interface - Programmierschnittstelle
[4] Benutzeranfragen an die Applikation über das Internet
[5] Sequenz aus drei zusammen stehenden Buchstaben

bildung 2.3). Dabei werden die von allen Benutzern eingegebenen Wörter gespeichert und die letzten zehn angezeigt (siehe Abbildung 2.2).

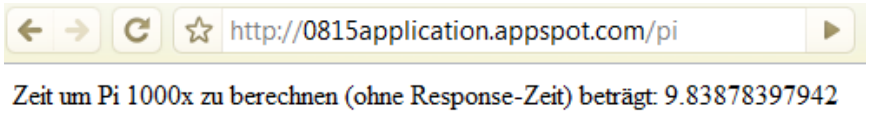


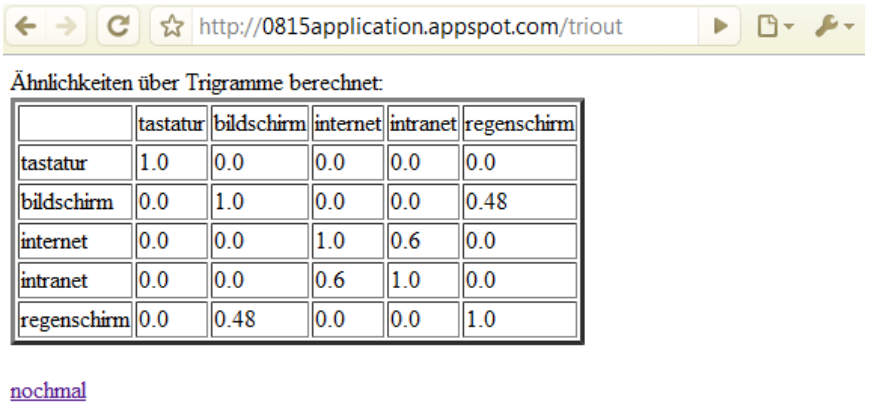
Abbildung 2.1: Berechnung von π

Abbildung 2.2: Liste zuvor eingegebener Wörter

Für die Entwicklung dieser Webanwendung wurde Python verwendet. Die Anwendung besteht aus den beiden Dateien *app.yaml*, zur Konfiguration der Anwendung und *application.py*, dem eigentlichen Quellcode (siehe Anhang A).

Das entsprechende SDK für Python liefert den Google App Engine Launcher (siehe Abbildung 2.4). Mit diesem kann eine Applikation lokal getestet werden. Dazu wird ein Entwicklungsserver gestartet, der die Applikation in einer App Engine Laufzeitumgebung simuliert. Des weiteren kann man die Konfigurationsdatei bearbeiten und die Anwendung deployen. Der App Engine Launcher bietet auch einen Button um direkt zur webbasierten Administrationskonsole (*Dashboard*) einer Anwendung zu gelangen.

Das Dashboard für die kleine Webanwendung sieht man in Abbildung 2.5.



	tastatur	bildschirm	internet	intranet	regenschirm
tastatur	1.0	0.0	0.0	0.0	0.0
bildschirm	0.0	1.0	0.0	0.0	0.48
internet	0.0	0.0	1.0	0.6	0.0
intranet	0.0	0.0	0.6	1.0	0.0
regenschirm	0.0	0.48	0.0	0.0	1.0

Abbildung 2.3: Trigramm Vergleich

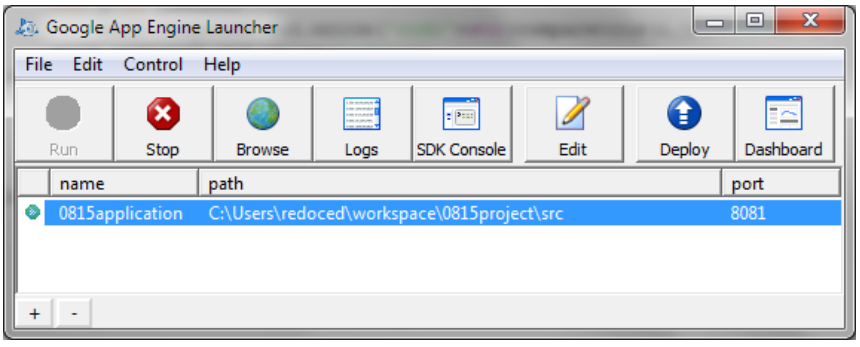


Abbildung 2.4: Google App Engine Launcher

Angezeigt werden darin Informationen über den aktuellen Status der Cloud Applikation. So sieht man hier zum Beispiel, dass in den vergangenen 15 Stunden nur einmal die Seite `/pi` (zur Berechnung von π) aufgerufen wurde, dieser eine Aufruf jedoch 85% der in dieser Zeit von der gesamten Applikation verbrauchten CPU Zeit verwendet hat. Im Dashboard sieht man also schnell, ob bestimmte Seiten viele Ressourcen verbrauchen oder gar Fehler auftreten und wie der Ressourcenverbrauch momentan ist. Details zu alledem kann man über weitere Seiten abfragen.

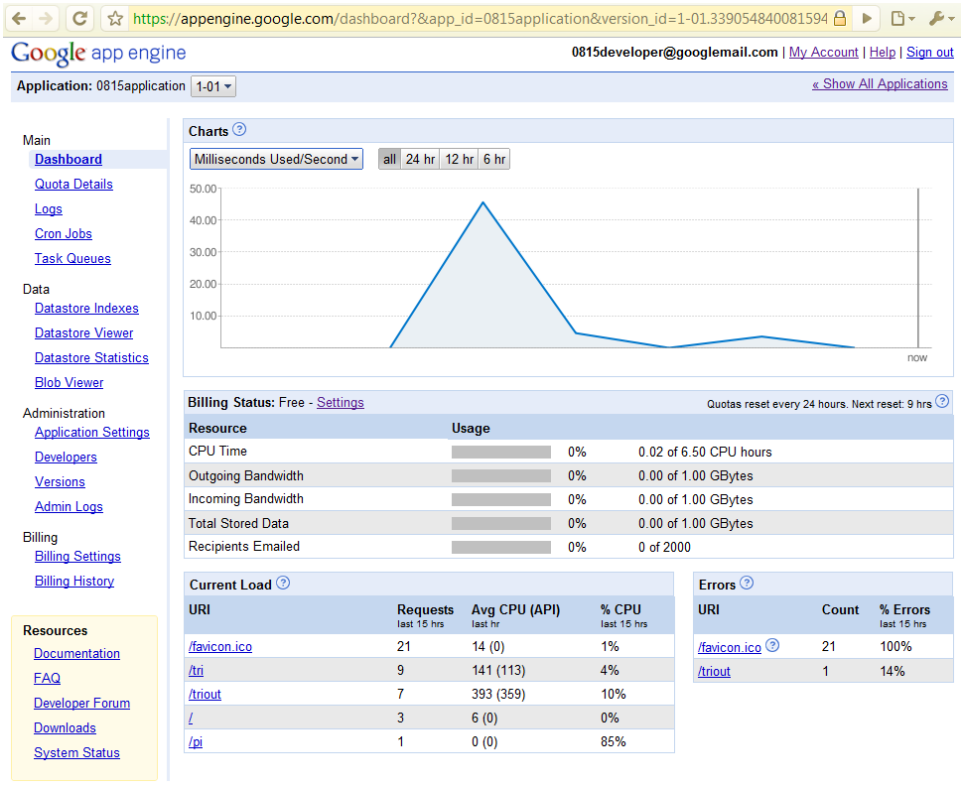


Abbildung 2.5: Dashboard

An dieser Stelle sei noch kurz hervorgehoben, dass man auch Zugriff auf den persistenten Speicher hat und entsprechend Daten abfragen oder ändern kann (siehe Abbildung 2.6) und dass eine Versionsverwaltung ermöglicht wird, über die verschiedene Versionen der Applikation hochgeladen und benutzt werden können (siehe Abbildung 2.7).

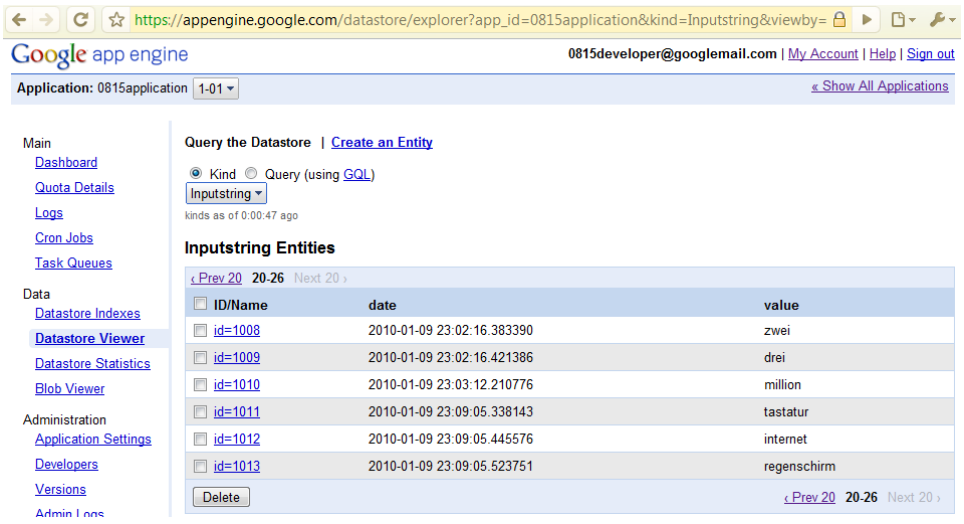


Abbildung 2.6: persistenter Speicher

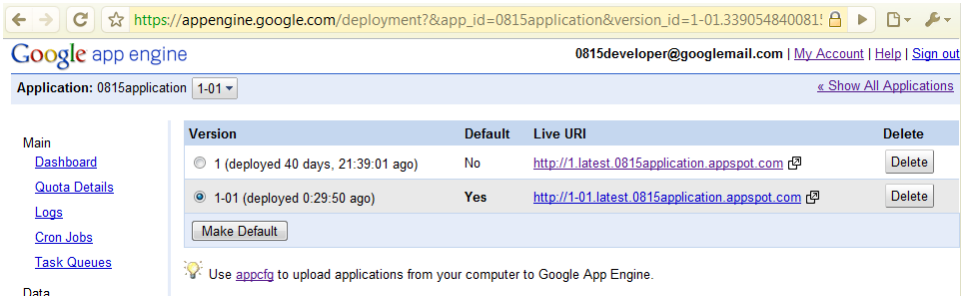


Abbildung 2.7: Versionsverwaltung

Kapitel 3

Weitere Cloud Software Plattformen

Im diesem Kapitel werden zuerst formlos die beiden Cloud Software Plattformen Microsoft Windows Azure und Force.com vorgestellt und Ähnlichkeiten oder Unterschiede zu Google App Engine herausgearbeitet. Im Anschluss folgt dann eine tabellarische Gegenüberstellung einiger Schlüsselpunkte.

3.1 Microsoft Windows Azure

Der grundsätzliche Aufbau von Windows Azure ist ähnlich dem von Google App Engine. Es gibt eine Plattform, die Windows Azure Plattform, für die Entwickler Applikationen schreiben und die über verschiedene Schnittstellen sowohl von Endnutzern, als auch von anderen Applikationen benutzt werden kann (siehe Abbildung 3.1). Analog zu Google App Engine bietet die Plattform dem Entwickler Dienste an — für persistenten Speicher gibt es hier beispielsweise Microsoft SQL Azure; für andere Dienste, wie Benutzerauthentifikation u.ä., gibt es die .NET Services und Live Services.

Als erste Besonderheit ist anzumerken, dass bei der Entwicklung von Windows Azure viel Wert auf Interoperabilität gelegt wurde. Dies spiegelt sich direkt in der Anzahl der Programmiersprachen wieder, die verwendet werden können um Applikationen zu entwickeln. So werden .NET Sprachen unterstützt, aber auch PHP, Ruby, Python und Java. Entsprechend hat man auch bei der Entwicklungsumgebung die Wahl zwischen Microsoft Vi-

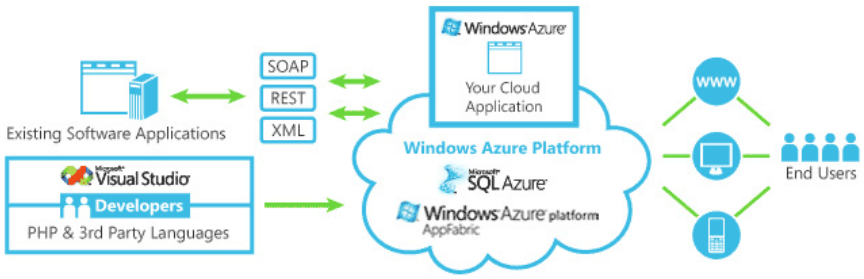


Abbildung 3.1: Microsoft Windows Azure

sual Studio oder Eclipse und bekommt eine gut integrierte Entwicklungsumgebung über SDK's bzw. Plugins.

Ein weiterer Unterschied zu Google App Engine ist die Architektur der Plattform. Die zentrale Komponente ist hier die Azure AppFabric, welche eine große Anzahl von Rechnern kapselt. In Windows Azure hat eine Applikation typischerweise mehrere Instanzen. Diese laufen jeweils in einer virtuellen Maschine mit dem Windows Server 2008 Betriebssystem mit einem zugewiesenen physischen Prozessorkern. Um auf Stoßzeiten reagieren zu können, werden vom Load Balancer neue Instanzen erstellt (siehe Abbildung 3.2).

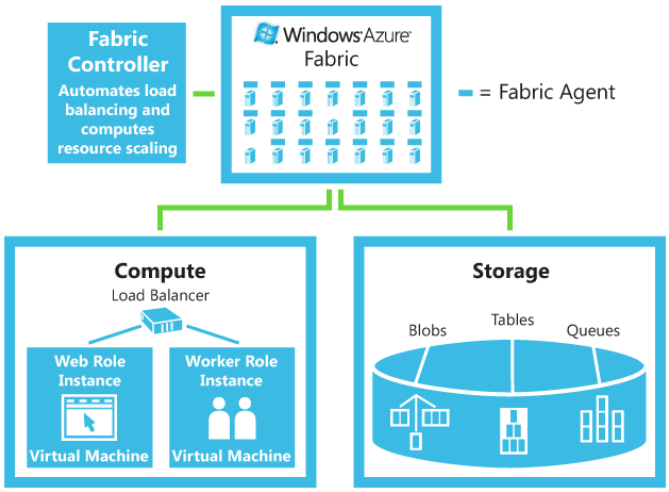


Abbildung 3.2: Azure AppFabric

Einer laufenden Instanz einer Applikation ist eine Rolle zugewiesen — entweder die Web Rolle oder die Worker Rolle. Die Virtuelle Maschine der

Web Rolle hat einen IIS[1] und kann HTTP Anfragen verarbeiten, während die Worker Rolle ihre Eingaben über Queues aus dem persistenten Speicher bezieht. Worker Rollen werden für Hintergrundaufträge gebraucht und haben damit eine ähnliche Funktion wie Tasks in Google App Engine.

Microsoft Windows Azure ist kostenlos zu testen. Werden für eine Anwendung mehr Ressourcen benötigt als der Testzugang zur Verfügung stellt, kann man sich über verschiedene Abonnements weitere Ressourcen erkaufen. Ein direkter Vergleich der Kosten für z.B. eine Stunde Rechenzeit (bei Windows Azure $0,12; bei Google App Engine $0,10) ist schwer zu beurteilen, da die zugrundeliegende Hardware unterschiedlich ist.[2]

3.2 Force.com

Force.com ist die Cloud Software Plattform von *Salesforce.com* und unterscheidet sich etwas mehr von den beiden bisher vorgestellten Plattformen. Salesforce.com begann ab 1999 CRM[3] Software als Service anzubieten. Ab 2006 veröffentlichten sie die „erste Cloud Computing-Programmiersprache“ Apex, um ihre SaaS Angebote erweitern zu können. Vor diesem Hintergrund enstand 2007 die Force.com Entwicklungs- und Betriebsplattform, welche Apex benutzt und ein Visualforce Web UI-Framework zur Verfügung stellt. Die Nähe zu ihren bisherigen SaaS Produkten ist in sofern relevant, als dass Anwendungen für die Force.com Plattform auch eher im B2B[4] Bereich angedacht sind. Dies zeigt sich vor allem durch die Preispolitik. Zwar gibt es auch bei Force.com die Möglichkeit genau eine Anwendung für maximal 100 Benutzer zu entwickeln, allerdings fallen dann Kosten von z.B. 54€ pro weiterem Benutzer und Monat an. Genau diese Abrechnung pro Nutzer schränkt die Anwendungsentwicklung ein, so dass z.B. öffentliche Applikationen, die ein jeder über das Internet erreichen kann, damit nicht möglich sind.

Ansonsten bietet Force.com wieder ähnliche Dienste, wie eine relationale Datenbank und ein Sicherheits-Framework, als auch eine integrierte

[1] Internet Information Services: Diensteplattform die Kommunikation über Netzwerke mit diversen Protokollen ermöglicht.

[2] vgl. [Mic10]

[3] Customer Relationship Management (auf Deutsch: Kundenbeziehungsmanagement)

[4] Business to Business: von Unternehmen für Unternehmen

Entwicklungsumgebung auf Basis von Eclipse.[5]

3.3 Vergleich

	Google App Engine	Microsoft Windows Azure	Force.com
Sprachen	Python, Java	.NET, PHP, Ruby, Python, Java	Apex
persistenter Speicher	nicht relational, Abfragesprache GQL	SQL Azure, relational, Teilmenge von Microsoft SQL Server	relational, Salesforce Object Query Language (SOQL), Salesforce Object Search Language (SOSL)
Nutzer	Endbenutzer, Unternehmen	Endbenutzer, Unternehmen	Unternehmen
Preispolitik	anfallende Kosten entsprechend der Ressourcennutzung	Abonnement, Kosten entsprechend Ressourcennutzung	Kosten pro Nutzer, zwei Abonnementmöglichkeiten
Testnutzung	ja, entsprechend der Quotas	ja, entsprechend des Einführungs-Abonnements	ja, eine Anwendung bis 100 Benutzer

[5] vgl. [Sal10]

Kapitel 4

Zusammenfassung

Diese Arbeit hatte zum Ziel drei Cloud Software Plattformen vorzustellen und zu vergleichen. Ein Vergleich kann sich jedoch nur auf das gröbste beschränken, da die Plattformen doch zu unterschiedlich sind und wie schon erwähnt auch unterschiedliche Zielgruppen ansprechen. Vor allem Force.com ist mit dem Hintergrund von CRM SaaS einzig auf Unternehmen ausgerichtet.

Mit den Möglichkeiten der Testnutzung bleibt abschließend zu sagen, dass alle vorgestellten Plattformen kostenlos ausprobiert werden können. Damit kann genügend evaluiert werden, ob die jeweilige Plattform den Anforderungen entspricht.

Literaturverzeichnis

[BKNT09] BAUN, Christian ; KUNZE, Marcel ; NIMIS, Jens ; TAI, Stefan: *Cloud Computing: Web-basierte dynamische IT-Services*. Springer, 2009

[Goo10] GOOGLE: *Developer's Guide – Google App Engine*. Version: 2010. `http://code.google.com/appengine/docs/`, Abruf: 31.03.2010

[Mic10] MICROSOFT: *Windows Azure Platform | Cloud Computing | Online Services | Data Storage*. Version: 2010. `http://www.microsoft.com/windowsazure/`, Abruf: 31.03.2010

[Sal10] SALESFORCE.COM: *Application Development with Force.com's Cloud Computing Platform - salesforce.com*. Version: 2010. `http://www.salesforce.com/platform/`, Abruf: 31.03.2010

Anhang A

A.1 app.yaml

```
application: 0815application
version: 1-01
runtime: python
api_version: 1
handlers:
- url: /.*
  script: application.py
```

A.2 application.py

```
import  math, decimal, timeit

def pi():
    decimal.getcontext().prec += 2
    three = decimal.Decimal(3)
    lasts, t, s, n, na, d, da = 0, three, 3, 1, 0, 0, 24
    while s != lasts:
        lasts = s
        n, na = n+na, na+8
        d, da = d+da, da+32
        t = (t * n) / d
        s += t
    decimal.getcontext().prec -= 2
    return +s

def get_unordered_trigrams(ain):
    a="  "+ain+"  "
    trigrams = [a[i:i+3] for i in range(len(a) - 2)]
    hist = dict()
    for i in trigrams:
        if i in hist:
            hist[i]+=1
        else:
            hist[i]=1
    return hist

def comparetris(ain, bin):
    a=get_unordered_trigrams(ain)
    b=get_unordered_trigrams(bin)
    overlap = 0
    for i in a:
        if i in b:
            overlap += min(a[i],b[i])

    total = len(ain) + len(bin)+4
    dice  = overlap * 2.0 / total
    return dice

import cgi
```

```
from google.appengine.ext import db
from google.appengine.api import users
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app

class IndexPage(webapp.RequestHandler):
    def get(self):
        self.response.out.write("""
          <html>
            <body>
              <a href="/pi">Hier wird Pi berechnet</a><br><br>
              <a href="/tri">Hier werden W&ouml;rter verglichen</a>
            </body>
          </html>""")

class PiPage(webapp.RequestHandler):
    def get(self):
        from timeit import Timer
        t = Timer("pi()",'from __main__ import pi')
        self.response.out.write("<html><body>Zeit um Pi 1000x zu berechnen "
                                "(ohne Response-Zeit) betr&auml;gt: ")
        self.response.out.write(t.timeit(number=1000))
        self.response.out.write('</body></html>')

class Inputstring(db.Model):
    value = db.StringProperty(multiline=False)
    date = db.DateTimeProperty(auto_now_add=True)

class TrigramPage(webapp.RequestHandler):
    def get(self):
        self.response.out.write("""
          <html>
            <body><a href='/'>zur&uuml;ck</a><br><br>
              Tragen sie hier die W&ouml;rter ein die zu vergleichen sind:
              <form action="/triout" method="post">
                <div><textarea name="content" rows="3" cols="60"></textarea></div>
                <div><input type="submit" value="Vergleiche"></div>
              </form>
              zuletzt eingegeben:
              <br>""")

        last = db.GqlQuery("SELECT * "
                           "FROM Inputstring "
                           "ORDER BY date DESC LIMIT 10")
        for i in last:
            self.response.out.write(i.value)
            self.response.out.write("<br>")
        self.response.out.write("""</body>
          </html>""")

class TrigramOutputPage(webapp.RequestHandler):
    def post(self):
        import re
        inputs=cgi.escape(self.request.get('content'))
        result = re.sub(r"(?simx)(?:[^\w])+", " ", inputs).split()

        self.response.out.write("<html><body>&Auml;hnlichkeiten "
                                "&uuml;ber Trigramme berechnet:<br>")
        self.response.out.write('<table border="3"><tr><td />')
        for i in result:
            self.response.out.write('<td>'+i+'</td>')
            inputstring = Inputstring()
            inputstring.value = i
            inputstring.put()
        self.response.out.write('</tr>')

        for i in result:
            self.response.out.write('<tr><td>'+i+'</td>')
            for j in result:
                self.response.out.write('<td>'+str(comparetris(i,j))+'</td>')
            self.response.out.write('</tr>')
        self.response.out.write("</table></br><a href='/tri'>nochmal</a>"
                                "</body></html>")

application = webapp.WSGIApplication([('/',       IndexPage),
                                      ('/pi',     PiPage),
                                      ('/triout', TrigramOutputPage),
                                      ('/tri',    TrigramPage)],
                                     debug=True)
def main():
    run_wsgi_app(application)

if __name__ == "__main__":
    main()
```