Neue Welten

Java-Enterprise in der Cloud

Marek Iwaszkiewicz, Michael Schütz

In den letzten Monaten war Cloud Computing das vorherrschende Thema in verschiedenen Magazinen, Blogs und sogar auf Veranstaltungen wie der CeBIT. Welche Möglichkeiten hier für bestehende Java-Enterprise-Anwendungen existieren und welche Konsequenzen sich aus einer Migration auf eine Cloud ergeben, wurde dabei jedoch zumeist außer Acht gelassen. Dieser Aspekt soll im Rahmen dieses Artikels anhand eines einfachen praktischen Beispiels diskutiert werden.

Im Rahmen des Artikels soll aufgezeigt werden, wie ein auf Spring basierendes Java-Enterprise-Projekt mit geringem Aufwand auf die *Google App Engine* [GAE] migriert wird und wie dabei weiterhin die Java Persistence API (JPA) als Persistenzframework genutzt werden kann. Dabei wird die Entscheidung für die Wahl der Cloud beleuchtet und diskutiert, welche Probleme und Anforderungen sich für die zu migrierende Java-Enterprise-Anwendung ergeben.

Die Pizza4U-Anwendung

Grundlage des Beispiels bildet das imaginäre Start-up-Unternehmen Pizza4U, welches durch den Einstieg von Investoren über die nötigen finanziellen Mittel verfügt, Fernseh-Werbekampagnen zu finanzieren. Bereits der erste ausgestrahlte Werbespot erreicht ein großes Publikum – führt jedoch auch zum Zusammenbruch des Servers. Der erhoffte Effekt der Kampagne ist hinfällig und potenzielle Neukunden sind abgeschreckt. Um einen weiteren Zusammenbruch des Servers zu vermeiden, muss schnellstmöglich eine dynamisch skalierbare und dennoch kostengünstige Lösung her.

Der in Abbildung 1 dargestellte Technologie-Stack der Pizza4U-Anwendung umfasst Technologien wie Spring 3 und JPA/Hibernate im Backend sowie Flex und BlazeDS im Frontend. Die Infrastruktur basiert auf der MySQL-Datenbank und JBoss als Applikationsserver. Der Build-Prozess wird mittels Maven organisiert.

Im Folgenden wird untersucht, welche Probleme und Anforderungen sich im Rahmen der eingesetzten Technologien ergeben. Um ein Gefühl für mögliche Probleme bei der geplanten Umstellung zu bekommen, werden zunächst die beteiligten Kerntechnologien beleuchtet und das Thema Cloud Computing vorgestellt.

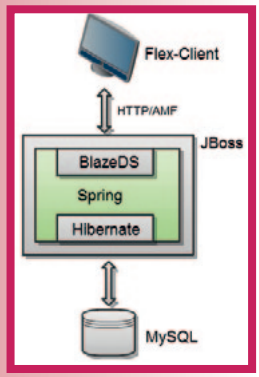

Abb. 1: Pizza4U-Infrastruktur

Um auf den bestehenden Paradigmenkonflikt zwischen Googles Persistenzschicht Bigtable und dem relationalen Datenmodell einzugehen, wird diesem Thema ein eigener Abschnitt gewidmet. Dieser umfasst neben der Beschreibung von Bigtable auch die Beschreibung des DataNucleus-Frameworks, welches eine JPA-Implementierung für die Persistenzschicht der GAE bereitstellt.

Ab in die Wolke

Cloud Computing beschreibt den Ansatz, Programme und Infrastruktur über das Internet zu hosten. Das angestrebte Ziel ist die Reduzierung von Kosten sowie die Vermeidung von Abhängigkeiten und Engpässen. Die Vorteile im Überblick:

- Kostenersparnis durch reduzierte Administrationskosten und durch eine verbrauchsabhängige Abrechnung,
- Risikominimierung aufgrund fehlender Hosting-Verträge,
- Flexibilität/Skalierbarkeit, da Ressourcen auch für kurze Zeitspannen schnell aufgestockt werden können.

Den Vorteilen stehen die folgenden Nachteile gegenüber:

- Sicherheitsbedenken bezüglich des Auslagerns von Unternehmensdaten,
- geringere administrative Möglichkeiten bei Problemen,
- Verwendung proprietärer Frameworks.

Varianten von Cloud Computing

Als Grundlage für die bevorstehende Anwendungsmigration sollen zunächst die verschiedenen Ausprägungen des Cloud Computing kurz vorgestellt werden:

- Infrastructure as a Service (IaaS) oder Cloud Hosting,
- Platform as Service (PaaS),
- Software as a Service (SaaS).

Obwohl die Abgrenzungen der genannten Ausprägungen nicht detailliert spezifiziert sind, bilden sie Teilmengen voneinander. Bei *IaaS* kann der eigene virtuelle Server im Baukastensystem integriert werden, sodass in diesem Fall eine explizite Administration erfolgen muss. Die *Amazon Elastic Compute Cloud* (EC2) sowie *Elastic Hosts* sind zwei Beispiele für diese Art der Cloud.

PaaS bietet eine integrierte Laufzeitumgebung für Anwendungen. Die Bereitstellung von Ressourcen erfolgt hier nach Bedarf. Beispiel für diese Art der Cloud sind *Windows Azure* von Microsoft und die *App Engine* von Google.

SaaS ist ein bekanntes Geschäftsmodell, in welchem Software nicht als Produkt, sondern als Dienst bereitgestellt und vertrieben wird. Beispiele hierfür sind vielfältig, unter anderem zählen hierzu *Google Docs* und *Sharepoint Online*.

Da Pizza4U seine bereits bestehende Anwendung in einer Cloud laufen lassen und sich selbst nicht um die Konfiguration des Cloud Hostings kümmern möchte, benötigt es eine PaaS als Cloud-Umgebung, in welcher komplett auf die vom Anbieter bereitgestellte Infrastruktur gesetzt wird. Flexible Skalierbarkeit und ein geringer Administrationsaufwand sind dabei die beiden Hauptargumente für die Wahl der Cloud-Umgebung.

Warum Google?

Die Google App Engine eignet sich für eine Standard-Integration einer Java-Anwendung. Zur Unterstützung von Java-basierten Webanwendungen setzt die GAE auf den Servlet-Container *Jetty* [Jetty]. Die Plattform ist intuitiv zu bedienen, leicht skalierbar und erfordert keine tiefer gehenden Administrationskenntnisse. Weiterhin setzt Google die App Engine auch für andere Anwendungen mit großen Datenmengen und Zugriffen ein, wie z. B. Google Earth und YouTube. So ist davon auszugehen, dass das System skaliert, hochverfügbar ist und sich in der Praxis bewährt hat – ideal also für ein Start-up und damit für Pizza4U.

Kosten?

Der Einsatz der GAE ist kostenlos für ein begrenztes tägliches Kontingent an Ressourcen. Kosten entstehen erst nach der Überschreitung einer dieser Grenzen. Die Werte der ver-

brauchten Ressourcen werden alle 24 Stunden zurückgesetzt. In Tabelle 1 sind vorhandene Ressourcen wie CPU-Rechenzeit oder Bandbreite inklusive ihrer jeweiligen Kontingente und Kosten aufgelistet.

Resource	Budget	Unit Cost	Paid Quota	Free Quota	Total Daily Quota
CPU Time	n/a	$0.10/CPU hour	n/a	6.50	6.50 CPU hours
Bandwidth Out	n/a	$0.12/GByte	n/a	1.00	1.00 GBytes
Bandwidth In	n/a	$0.10/GByte	n/a	1.00	1.00 GBytes
Stored Data	n/a	$0.005/GByte day	n/a	1.00	1.00 GBytes
Recipients Emailed	n/a	$0.0001/Email	n/a	2,000.00	2,000.00 Emails
Max Daily Budget:	n/a				

Tabelle 1: Kosten einzelner Ressourcen

Aufgrund der im Rahmen liegenden durchschnittlichen Zugriffszahlen bietet sich für Pizza4U die kostenfreie Variante der GAE an. So würden lediglich in Stoßzeiten Kosten anfallen. Sollte sich die Anwendung allerdings irgendwann dauerhaft im Markt etabliert haben und permanent hohe Zugriffe aufweisen, wird diese Entscheidung überdacht werden müssen.

Persistenz

Nachdem GAE als Cloud-Umgebung ausgewählt wurde, sollen zunächst die geänderten Anforderungen an die Persistenz beleuchtet werden. Eine Voruntersuchung einer möglichen Migration hat ergeben, dass sich gerade die Persistenz als der entscheidende Knackpunkt herausstellen könnte, da die relationale Welt auf die der Nichtrelationalen trifft.

Der Vorteil von PaaS ist, dass man sich um die Infrastruktur in Form von Host-System und Datenbank nicht aktiv kümmern muss. Dieser Vorteil kann sich allerdings auch schnell zu einem Nachteil entwickeln, wenn man die damit verbundenen technischen Implikationen nicht kennt und demzufolge auch nicht im Griff hat. Daher ist dieses Wissen eine wichtige Voraussetzung für die Migration. Um das von Google vorgegebene NoSQL-Datenbanksystem auch mit JPA/Hibernate ansteuern zu können, wird die Verwendung von DataNucleus erforderlich.

Bigtable

Googles Cloud speichert die persistenten Daten in Bigtable [G-Labs], einem proprietären Datenbanksystem aus der Familie der NoSQL-Datenbanken. NoSQL („not only SQL“) vereint die Familie der nicht relationalen Datenbanken [NoSQL]. Neben GAE setzt Google bereits seit 2004 auf diese Art der Persistenz und setzt Bigtable unter anderem bei Anwendungen wie Google Maps, Google Book Search, Google Earth und YouTube ein. Das Datenvolumen umfasst mehrere 1000 Terabyte.

Bigtable ist ein auf Column-Store basierendes Datenbanksystem. Column-Store-Datenbanken basieren auf dem Ansatz, Daten in Muster und Spaltenfamilien zu unterteilen, was ein effizientes Speichern und Verteilen von Daten mit verwandten Attributen ermöglichen soll. Es werden mehrere Versionen eines Datenbestandes in einer Spalte verwaltet. Des Weiteren setzt Bigtable auf das in dieser Richtung weitverbreitete Map-Reduce-Prinzip [MapRed].

DataNucleus

Die Kapselung von Java-Zugriffen auf Bigtable wird durch das DataNucleus-Framework ermöglicht. DataNucleus [DaNu] ist ein Open-Source-Projekt, welches die JDO-Referenzimplementierung bereitstellt und darüber hinaus JPA- und JDO-Implementierungen für die Persistenzschicht der GAE anbietet. Die Daten werden hierbei auf Googles Bigtable abgebildet. Die JPA-Implementierung benutzt intern beim Zugriff auf Bigtable die JDO-Implementierung. Hierdurch ergibt sich die Notwendigkeit, Anpassungen an den mit JPA annotierten Klassen vorzunehmen. DataNucleus bietet hierfür einen DataEnhancer an, der die Klassen der kompilierten Entitäten erweitert. Änderungen im Quellcode fallen daher nicht an.

Restriktionen der Lösung

Einschränkungen ergeben sich nicht primär durch DataNucleus an sich, sondern durch die darunterliegende Datenstruktur von Bigtable. So gibt es beispielsweise keine Unterstützung für eine Gewährleistung der Eindeutigkeit von Tabellenwerten. Auch bei Abfragen gibt es deutliche Einschränkungen, so ist der Einsatz von SQL-Funktionen wie UPPER oder LIKE nicht möglich. Momentan gibt es auch noch keine finale JPA2.0-Implementierung von DataNucleus. Die folgenschwerste Einschränkung ist allerdings, dass der Einsatz von Many-to-many-Beziehungen und polymorphen Referenzen nicht möglich ist.

Migration

Nachdem in den vorangegangenen Kapiteln die Kerntechnologien der Zielarchitektur beschrieben wurden, widmet sich dieses Kapitel der Migration von Pizza4U auf die Google App Engine. Die Migration selbst lässt sich in die folgenden drei Schritte unterteilen.

Einrichten der GAE

Zunächst wird für die Nutzung der GAE eine Google Application ID benötigt. Dafür sind die folgenden Schritte erforderlich:

- Anlegen einer Google-Mail-Adresse [G-Mail],
- Anmeldung mit der Mail an der Google App Engine [GAE],
- Erstellung einer neuen Anwendung mit einer festen ID, hier: „spring-flex-gae“,
- initiale Validierung des Accounts beim Anlegen einer App mittels SMS.

Projektumbau

Jede Java-GAE-Anwendung muss in ihrem WEB-INF-Verzeichnis die Datei „appengine-web.xml“ enthalten. In dieser Datei erfolgt die Konfiguration der GAE. Listing 1 enthält den kompletten Inhalt der Datei aus dem Pizza4U-Projekt. Der Name der Anwendung wird hier auf „pizza4u“, die Version auf „1“ gesetzt. Darüber hinaus sind noch eine Reihe weiterer Konfigurationsmöglichkeiten geboten. Dazu zählen zum Beispiel das Logging und die Angabe statischer Webressourcen. Außerdem bietet die Google App Engine die Implementierung einer HttpSession an, in welcher Objekte wie üblich über `session.set-`

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>spring-flex-gae</application>
  <version>1</version>
  <system-properties>
    <property name="java.util.logging.config.file"
              value="WEB-INF/logging.properties"/>
  </system-properties>
  <sessions-enabled>true</sessions-enabled>
</appengine-web-app>
```

Listing 1: appengine-web.xml

`Attribute()` abgelegt werden können. Die so abgelegten Attribute werden in dem App-Engine-Datastore abgelegt. Eine umfangreichere Beschreibung der Konfigurationsmöglichkeiten kann in [GAE2] nachgelesen werden.

Die nächste relevante Konfigurationsdatei stellt die „persistence.xml"-Datei dar. Listing 2 enthält einen Ausschnitt aus der Datei der Pizza4U-Anwendung. Hier findet man die erste wichtige Änderung am bestehenden Projekt. So wird zuerst ein von DataNucleus bereitgestellter Persistence-Provider als Bindeglied zwischen der JPA und dem App-Engine-Datastore eingebunden. Über die einzelnen Propertys erfolgt die Konfiguration von Aspekten wie Transaktionen und Factorys.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0">
  <persistence-unit name="pizza-pu" transaction-type="RESOURCE_LOCAL">
    <provider>
     org.datanucleus.store.appengine.jpa.DatastorePersistenceProvider
    </provider>
    ...
    <properties>
      <property name="javax.jdo.PersistenceManagerFactoryClass"
                value="org.datanucleus.store.appengine.
                      jdo.DatastoreJDOPersistenceManagerFactory"/>
      <property name="javax.jdo.option.ConnectionURL"
                value="appengine"/>
      <property name="javax.jdo.option.NontransactionalRead"
                value="true"/>
      <property name="javax.jdo.option.NontransactionalWrite"
                value="true"/>
      <property name="javax.jdo.option.RetainValues" value="true"/>
      <property name="datanucleus.appengine.autoCreateDatastoreTxns"
                value="false"/>
    </properties>
  </persistence-unit>
</persistence>
```

Listing 2: persistence.xml

Damit sind alle erforderlichen Anpassungen von Konfigurationsdateien abgeschlossen. Zwar folgen noch Änderungen am Build-Prozess, in die Konfigurationsdateien muss ab jetzt jedoch nicht mehr eingegriffen werden. Die verbleibenden Änderungen an der Anwendung finden vollständig im Java-Code statt.

Pizza4U basiert auf Spring in der aktuellen Version 3, mit welcher es nun möglich ist, die gesamte Spring-Konfiguration annotations-getrieben umzusetzen. Listing 3 zeigt einen Ausschnitt aus der Klasse `SpringConfig`, welche über die Spring3-Annotation `@Configuration` imstande ist, Konfigurationen am Spring-IoC-Container vorzunehmen. In diesem Fall sind die beiden mit `@Bean` annotierten Methoden relevant. Die `entityManagerFactory`-Methode konfiguriert die von der Persistenz benötigte `javax.persistence.EntityManagerFactory`. Diese Methode erstellt

```
@Configuration
public class SpringConfig {
  @Bean
  public EntityManagerFactory entityManagerFactory() {
    return Persistence.
  createEntityManagerFactory("pizza-pu").entityManagerFactory();
  }
  @Bean
  public JpaTransactionManager transactionManager() {
    JpaTransactionManager transactionManager =
      new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(entityManagerFactory());
    return transactionManager;
  }
}
```

Listing 3: SpringConfig.java

die Factory anhand der zuvor in der Datei „persistence.xml" definierten Persistence-Unit mit dem Namen „pizza-pu". Die weitere Methode liefert einen JpaTransactionManager, welcher die eben genannte Factory nutzt.

Das Domain-Modell

Abbildung 2 zeigt das einfache Domain-Modell der Pizza4U-Anwendung. Auf die Darstellung der Attribute wurde der Einfachheit halber verzichtet. Modelliert wird hier die Pizza-Bestellung, in welcher eine Bestellung einem Kunden zugewiesen und eine Menge bestellter Gerichte referenziert wird. Ein Gericht ist als abstrakte Klasse modelliert, bisher mit einer Implementierung – der Pizza.

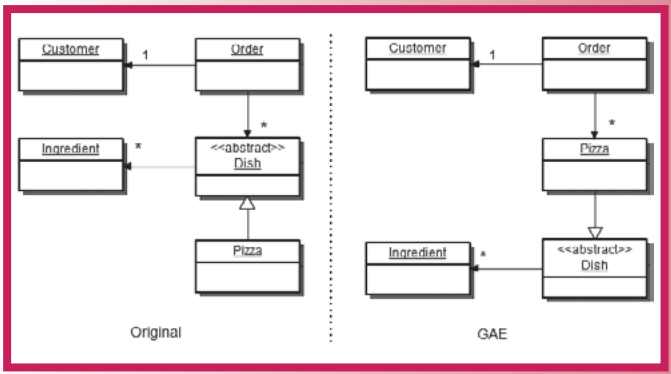

Abb. 2: Domain-Modell

Der Einsatz von DataNucleus als JPA-Persistenzframework für die GAE erfordert leider nicht nur einige Anpassungen an den Persistenz-Mappings. Ein entscheidender Nachteil der Migration ist, dass auf Persistenzebene zwischen den zu speichernden Entitäten keine polymorphen Beziehungen bestehen dürfen. Das ist in dem Pizza4U-Domain-Modell bei der Referenz von `Order` auf die abstrakte Klasse `Dish` der Fall. Daher sind wir gezwungen, auch an der Struktur selbst Änderungen vorzunehmen und die polymorphe Beziehung aufzulösen. Das ist an dieser Stelle zwar einfach, da Pizza die einzige von `Dish` erbende Klasse ist, es könnte jedoch bei komplexeren Modellen zu gravierenden Problemen führen. Würde das Modell eine weitere Spezialisierung von `Dish` enthalten (zum Beispiel `Pasta`), so müsste die `Order`-Klasse zwei Listen verwalten, welche einmal mit `Pizza` und einmal mit `Dish` typisiert sind. Damit in diesem Fall keine umfangreichen Änderungen an der Businessschicht erfolgen müssen, könnte eine Modellschicht eingeführt werden, die die polymorphen Beziehungen nach wie vor enthält. Es müsste dann zusätzlich ein Mapping zwischen den beiden Schichten implementiert werden. Der rechte Teil von Abbildung 2 stellt das für die GAE umstrukturierte Domain-Modell dar.

Im Folgenden stellen wir die in den Klassen anfallenden Änderungen der Persistenz-Mappings vor. Als erstes betrachten wir die `Order`-Klasse, deren Auszug in Listing 4 zu sehen ist. Bis auf die Änderung der Generator-Strategie für die ID musste in der Klasse nichts weiter gemacht werden. Zuvor wurde als Generator-Strategie GenerationType.AUTO genutzt, welche von der GAE nicht unterstützt wird. Eine detaillierte Beschreibung der möglichen Strategien findet man unter [G-Code].

Bezüglich des Typs der ID ist jedoch eine Einschränkung zu beachten: Wird eine Klasse im Kontext einer anderen Klasse gespeichert, so muss in dieser das ID-Attribut vom Typ `com.google.`

```
@Entity
public class Order implements Serializable {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @OneToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
  private Customer customer;

  @OneToMany(cascade = CascadeType.ALL)
  private Collection<OrderedDish> orderedDishes;
}
```

Listing 4: Die Order-Klasse

`appengine.api.datastore.Key` sein. In der `Customer`-Klasse (s. Listing 5) ist das, abgesehen von der Generator-Strategie, die einzige Änderung. Gleiches gilt für die abstrakte Klasse `Dish`.

```
@Entity
public class Customer implements Serializable {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Key key;
  private String firstname;
  private String lastname;
  private String phone;
  private String street;
  private String postal;
  private String city;
}
```

Listing 5: Die Customer-Klasse

Build

DataNucleus stellt neben der Implementierung der JPA für die GAE auch ein Maven-Plug-in zur Verfügung, mit welchem das im obigen Abschnitt über Persistenz beschriebene Enhancement der Entity-Klassen durchgeführt werden kann. Listing 6 enthält einen Ausschnitt der Datei „pom.xml“, in welcher die Einbindung und Konfiguration des DataNucleus-Maven-Plugins vorgenommen wird.

```
<plugins>
  <plugin>
    <groupId>org.datanucleus</groupId>
    <artifactId>maven-datanucleus-plugin</artifactId>
    <version>1.1.4</version>
    <configuration>
      <verbose>true</verbose>
      <api>JPA</api>
      <enhancerName>ASM</enhancerName>
      <mappingIncludes>**/domain/*.class</mappingIncludes>
      <persistenceUnitName>pizza-pu</persistenceUnitName>
      <log4jConfiguration>
        ${basedir}/log4j.properties
      </log4jConfiguration>
    </configuration>
    <executions>
      <execution>
        <phase>process-classes</phase>
        <goals>
          <goal>enhance</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
  ...
</plugins>
```

Listing 6: Ausschnitt aus der pom.xml

Administration

Deployment

Nachdem die Anwendung technisch auf die Google App Engine migriert ist, bleibt das finale Deployment samt Übernahme der IT-Infrastruktur. Google bietet dafür die folgenden Möglichkeiten an:

- Konsole,
- IDE-Plug-in (derzeit nur für Eclipse),
- Ant oder Maven.

In Abbildung 3 ist ein Beispiel für das Deployment mittels Konsole aufgezeigt. Dafür muss der entsprechende Classpath-Eintrag gesetzt sowie die Zugangsdaten konfiguriert sein. Danach ist die Anwendung unter `http://spring-flex-gae.appspot.com` zu erreichen.

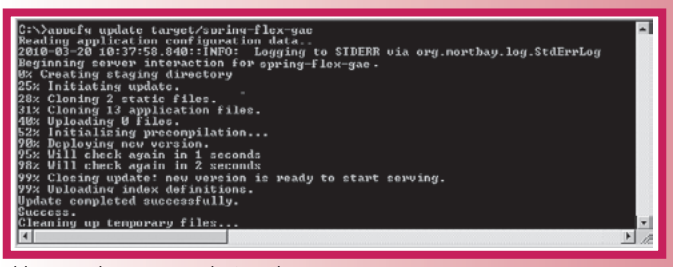

Abb. 3: Deployment mittels Konsole

Backend

Nachdem die Anwendung installiert und online erreichbar ist, soll ein Blick in die Online-Administration [GAE] das Bild abrunden. Das in Abbildung 4 abgebildete Dashboard bietet einen Überblick über Auslastung und Zugriffe auf die Anwendung sowie über die eventuell angefallenen Kosten.

Weitere von Google direkt angebotene und in diesem Artikel aus Platzmangel nicht weiter beschriebene GAE-Features [G-Code2] sind:

- Scheduling/Timer Service,
- Anbindung von Google APIs wie z. B. GoogleMaps oder Picasa,
- Verwendung von Authentisierung mittels Google Accounts,
- Datenbankabfragen mit der Google Query Language (GQL),
- Log-Datei-Auswertung mit eingebauter RegEx-Suche.

Fazit

Es ist geschafft, Pizza4U ist in der Wolke und damit hervorragend aufgestellt für die anstehende TV-Kampagne. Die dafür

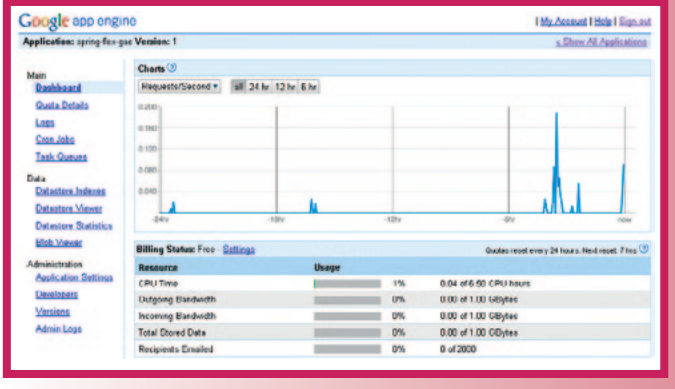

Abb. 4: Dashboard Google App Engine

benötigten Ressourcen werden bei Bedarf dynamisch zur Verfügung gestellt.

Die bei der Migration angefallenen Probleme konnten erfolgreich gelöst werden. Kritisch bleibt aber zu erwähnen, dass die Frage der erforderlichen Technologien, welche unter der GAE laufen sollen, entscheidend ist. So lässt sich auf der GAE die JPA nur mit Einschränkungen wie dem Verzicht polymorpher Referenzen verwenden. Zwar existieren auch hierfür Lösungsmöglichkeiten, die hier anfallenden Aufwände sind aber nicht immer hinnehmbar. Weiterhin wird auch der Einsatz zahlreicher Java-EE-Standardtechnologien wie EJB, JAX-WS und JMS nicht unterstützt [G-Groups]. Daraus ergibt sich, dass darauf basierende professionelle Java-EE-Anwendungen derzeit nicht auf der GAE eingesetzt werden können.

Im laufenden Betrieb ist mit keinen größeren Problemen zu rechnen, da wir mit der GAE auf ein bewährtes System setzen. Hinsichtlich des Datenaufkommens gibt es auch keine großen Bedenken, da die Pizza4U-Anwendung im Vergleich zu den bekannten großen GAE-Anwendungen wie YouTube winzige Datenmengen aufweist.

Einige durchaus sehr praxisrelevante Fragen wie z. B. die Verantwortlichkeit von Google gegenüber Datenverlusten konnten im Rahmen dieses Artikels nicht geklärt werden.

Die Anwendung ist live erreichbar unter `http://spring-flex-gae.appspot.com`. Der Code kann auf `http://berlin-incubator.org` heruntergeladen werden und lädt explizit zum Ausprobieren ein:

- SVN Checkout,
- Build: mvn clean install,
- Local deployment (build.xml).

Mit einer eigenen Application ID (Anpassung in der „appengine-web.xml“) kann die Anwendung auch direkt in die Google Cloud installiert werden. Auf der entsprechenden Wiki-Seite sind die erforderlichen Tools aufgelistet.

Die Autoren freuen sich über Feedback jeder Art.

Literatur und Links

[DaNu] DataNucleus, JPA-Implementierung, `http://datanucleus.org`

[GAE] Google app engine, GAE Account, `https://appengine.google.com`

[GAE2] GAE, Konfiguration, `http://code.google.com/intl/de-DE/appengine/docs/java/config/appconfig.html`

[G-Code] Google-Code, Verwendung von JPA mit GAE, `http://code.google.com/intl/de-DE/appengine/docs/java/datastore/usingjpa.html`

[G-Code2] Google-Code, GAE Community, `http://code.google.com/intl/de-DE/appengine/community.html`

[G-Labs] Google Labs, Bigtable: A Distributed Storage System for Structured Data, `http://labs.google.com/papers/bigtable-osdi06.pdf`

[G-Groups] Google Groups, GAE Java/Java EE-Restriktionen, `http://groups.google.com/group/google-appengine-java/web/will-it-play-in-app-engine`

[G-Mail] GMail, Neue E-Mail, `https://www.google.com/accounts/NewAccount`

[Jetty] Servlet Container, `http://jetty.codehaus.org/jetty`

[MapRed] Das Map Reduce-Prinzip, `http://de.wikipedia.org/wiki/MapReduce`

[NoSQL] NoSQL, Überblick NoSQL-Datenbanken, `http://nosql-databases.org`

Marek Iwaszkiewicz arbeitet als Software Engineer beim IT-Dienstleistungs- und Beratungsunternehmen adesso AG. Seine Schwerpunkte dort sind EJB sowie Java-Enterprise-Architekturen mit Spring. E-Mail: marek.iwaszkiewicz@adesso.de.

Michael Schütz ist Software Engineer bei der akquinet AG in Berlin und beschäftigt sich seit Jahren mit Java-Enterprise-Architekturen und -Technologien. Er ist Gründer und Moderator der Seam-Plattform auf Xing und widmet sich gerade dem Thema Java EE 6 und CDI/Weld. E-Mail: Michael.Schuetz@akquinet.de.