2 Übungsaufgabe #2: OpenStack-Private-Cloud

In dieser Aufgabe soll die Effizienz des in Aufgabe 1 implementierten Web-Service mit Hilfe eines Cache gesteigert werden. Der Cache dient dazu, die für die Berechnung eines kürzesten Pfads zwischen zwei Nutzern vom Facebook-Dienst geholten bzw. erzeugten Daten lokal zu verwalten, und bei zukünftigen Pfadberechnungen auf diese zurückzugreifen. Auf diese Weise lässt sich die Anzahl der Aufrufe am Facebook-Service reduzieren. Der erweiterte Pfad-Dienst soll anschließend auf der privaten *OpenStack*-Cloud des Lehrstuhls 4 ausgeführt werden.

2.1 Cache (für alle)

Die Implementierung des Cache soll sich nicht an einem bestimmten Anwendungszweck orientieren (Verwaltung von Daten für den Pfad-Dienst), sondern einen allgemeinen Dienst zur Verwaltung von Schlüssel-Wert-Paaren bereitstellen. Der Cache (`MWCache`) wird daher auch in einer eigenen Java-Instanz gestartet. Der Zugriff erfolgt über eine Hilfsklasse `MWCacheClient`, die per REST angebunden ist und folgende Schnittstelle bereitstellt:

```
public class MWCacheClient {
    public void addObject(String key, String value);
    public String getObject(String key) throws MWNoSuchKeyException;
}
```

Mittels `addObject()` lässt sich ein neues Schlüssel-Wert-Paar (beides Zeichenketten) zum Cache hinzufügen. Existiert bereits ein Eintrag für den angegebenen Schlüssel, wird der gespeicherte Wert durch den neuen ersetzt. Bestehende Einträge können über die Methode `getObject()` unter Verwendung des Schlüssels angefordert werden. Existiert kein Eintrag für den angegebenen Schlüssel, wird eine `MWNoSuchKeyException` geworfen.

2.1.1 Festlegung der Datentypen und Nachrichten

Zuerst sind die Datentypen und Nachrichtenformate, die beim Cache-Zugriff über REST zum Einsatz kommen sollen, in einer Datei `cache.xsd` zu spezifizieren. Der innere Aufbau der Datentypen und Nachrichten ist freigestellt, ebenso ihre Anzahl. Es ist jedoch darauf zu achten, dass die verwendeten Nachrichten das mögliche Auftreten von Fehlern während der Ausführung berücksichtigen (z. B. durch den Einsatz eines Status-Attributs oder die Verwendung eines gesonderten Nachrichtenformats für Fehlermeldungen). Sobald die Datentypen und Nachrichtenformate festgelegt sind, lassen sich mit dem Tool `xjc` aus der Datei `cache.xsd` die korrespondierenden Java-Klassen erzeugen (siehe Tafelübung).

Aufgaben:

→ Definition der Datentypen und Nachrichten in einer Datei `cache.xsd`
→ Erzeugung der Hilfsklassen mittels `xjc` in einem Subpackage `mw.cache.generated`

2.1.2 Implementierung des Cache-Diensts

Die Cache-Implementierung wird von der Klasse `MWCache` bereitgestellt. Diese implementiert die `Provider`-Schnittstelle für Web-Services, da sie direkt die Payloads der eintreffenden Anfragen verarbeiten soll.

```
public class MWCache implements Provider<Source> {
    public Source invoke(Source source);
}
```

Der Web-Service-Endpunkt sorgt dafür, dass für jede an den Cache-Service gerichtete Anfrage die Methode `invoke()` aufgerufen wird, unabhängig davon, ob es sich um einen `addObject()`- oder einen `getObject()`-Aufruf handelt. Eine einfache Möglichkeit, Anfragen beider Varianten voneinander zu unterscheiden, ist die verwendete HTTP-Methode: Zum Anlegen von Objekten sollen HTTP-`PUT`-Anfragen zum Einsatz kommen, das Abfragen von Objekten erfolgt dagegen mittels HTTP-`GET`.
Jedem im Cache verwalteten Objekt ist eine eindeutige *Objekt-URL* zugeordnet, die sowohl beim Anlegen als auch beim Auslesen des Objekts verwendet wird. Diese setzt sich typischerweise aus der URL des Cache-Diensts sowie dem Schlüssel des adressierten Objekts zusammen, z. B. `http://localhost:12345/cache/my-object-key`.

Aufgabe:

→ Implementierung der Klasse `MWCache`

Hinweise:

- Die Klassen `WebServiceContext` und `MessageContext` ermöglichen das Auslesen von HTTP-Headern.
- Das Auslesen von Objekten soll auch durch die Eingabe der Objekt-URL in einem Browser möglich sein.

2.1.3 Implementierung des Cache-Client und Erweiterung des Pfad-Diensts

Im nächsten Schritt soll nun die Klasse `MWCacheClient` für den Zugriff auf den Cache-Dienst implementiert werden. Hierbei ist vor allem dafür zu sorgen, dass beim Aufruf von `addObject()` bzw. `getObject()` die korrekten Anfragen (HTTP-Methode, Objekt-URL) erzeugt und an den Cache-Dienst gesendet werden.
Abschließend kann der Pfad-Dienst so erweitert werden, dass er Aufrufe am Facebook-Dienst nur noch tätigt, falls die angeforderten Daten sich nicht bereits im Cache befinden. Gleiches gilt für die Ergebnisse von Pfadberechnungen. Auf welche Weise der Pfad-Dienst dabei Daten im Cache ablegt ist freigestellt.

Aufgaben:

→ Implementierung der Klasse `MWCacheClient`
→ Erweiterung des Pfad-Diensts, so dass er den Cache sinnvoll nutzt

2.2 Erweiterter Cache (optional für 5,0 ECTS)

Die bisherige Implementierung des Cache-Diensts verwaltet Objekte in einem flachen Namensraum und beschränkt sich darauf, nur einen einzigen Wert pro Schlüssel zu speichern. Diese Einschränkung ist (wie die Teilaufgabe 2.1.3 gezeigt hat) besonders dann umständlich, wenn z. B. Arrays im Cache abgelegt werden sollen. In dieser Teilaufgabe soll der Cache-Dienst daher so erweitert werden, dass ein in Ansätzen hierarchischer Namensraum genutzt werden kann. Hierzu muss der `MWCacheClient` um die beiden folgenden Methoden erweitert werden:

```
public void addBucket(String key);
public Map<String, String> getBucket(String key) throws MWNoSuchKeyException;
```

Mit Hilfe der Methode `addBucket()` lässt sich ein *Bucket* in den Cache einfügen. Bei einem Bucket handelt es sich um eine durch einen eigenen Schlüssel eindeutig identifizierte Datenstruktur, die mehrere Objekte aufnehmen kann, vergleichbar mit einem Verzeichnis und den darin enthaltenen Dateien. Bei seiner Erzeugung ist ein Bucket leer. Um ihn mit Objekten zu füllen, kommt die bereits existierende Methode `addObject()` zum Einsatz, bei deren Aufruf die Objekt-URL durch den Bucket-Schlüssel erweitert ist, z. B. `http://localhost:12345/cache/my-bucket-key/my-object-key`.
Das Auslesen von in einem Bucket abgelegten Objekten erfolgt entweder einzeln durch den Aufruf von `getObject()` über die jeweilige Objekt-URL oder in einem einzigen Aufruf von `getBucket()` über die Bucket-URL. Als Rückgabewert liefert `getBucket()` eine `Map`, in der alle im angeforderten Bucket enthaltenen Objekte als Schlüssel-Wert-Paare gekapselt sind.

Der Einsatz von Buckets ermöglicht es nun, die Nutzung des Cache durch den Pfad-Dienst zum Beispiel an all jenen Stellen sauber zu implementieren, an denen `String`-Arrays zum Einsatz kommen, insbesondere also bei der Verwaltung der vom Facebook-Dienst per `getFriends()`-Methode gesammelten Freundeskreise.

Abschließend soll dafür gesorgt werden, dass der Neustart des Cache-Diensts nicht zum Verlust des Zustands führt. Hierzu müssen alle vom Cache verwalteten Objekte und Buckets persistent auf der Platte gesichert und beim Start des Diensts von dort eingelesen werden. In welchem Format die Daten persistent gemacht werden ist freigestellt, es ist jedoch darauf zu achten, dass das Einlesen des Zustands möglichst effizient erfolgt.

Aufgaben:

→ Erweiterung des Cache-Diensts für die Verwendung von Buckets
→ Anpassung der Cache-Nutzung des Pfad-Diensts
→ Implementierung einer persistenten Speicherung des Cache-Zustands

Hinweise:

- Der Fall einer weiteren Verschachtelung von Buckets muss nicht betrachtet werden.
- Es soll weiterhin möglich sein, Objekte außerhalb von Buckets anzulegen und auf diese zuzugreifen.

2.3 OpenStack-Cloud (für alle)

In dieser Teilaufgabe sollen der Pfad-Dienst, inklusive Cache-Dienst, auf der privaten *OpenStack*-Cloud des Lehrstuhls 4 ausgeführt werden. Dazu ist es notwendig, zunächst eine entsprechend angepasste Betriebssysteminstallation zu erstellen, diese auf den Cloud-Dienst hochzuladen und schließlich zu starten.

2.3.1 Einrichtung des Betriebssystems

Um die beiden Dienste in der Cloud-Umgebung ausführen zu können, muss zunächst ein Betriebssystemabbild erstellt werden, das unter OpenStack lauffähig ist (siehe Tafelübung). Als grundlegendes Betriebssystem ist dabei Debian „Jessie“ vorgesehen. Das eigene Abbild soll unter Zuhilfenahme der Linux-Live-CD *Grml* erzeugt werden. Ein spezielles Abbild mit dem Namen `GRML-2014.11-amd64` steht dafür bereits zur Verfügung, so dass davon eine Instanz gestartet werden kann. Darüber hinaus ist über die OpenStack-Weboberfläche ein leeres,

2 GB großes Volume zu erstellen. Dieses Volume dient als Basis für das eigene Abbild und sorgt dafür, dass darin vorgenommene Änderungen persistent gehalten werden und auch nachträglich noch jederzeit Änderungen vorgenommen werden können. Innerhalb der Grml-Live-Umgebung kann über das Gerät `/dev/vdb` auf das Volume zugegriffen werden, nachdem es per OpenStack-Weboberfläche der laufenden Grml-Instanz zugewiesen (engl. *attached*) wurde. Im abschließenden Schritt (Teilaufgabe 2.3.3) soll aus diesem Volume ein eigenes Abbild erzeugt und gestartet werden.

Die Einrichtung des Betriebssystems erfolgt in mehreren Teilschritten, wobei zunächst ein neues Dateisystem angelegt und eingehängt wird, um überhaupt Dateien in dem Volume speichern zu können. Nach der Installation einer Minimalversion von Debian mittels `debootstrap` sollen dann die Installation mit Hilfe des vorgefertigten Skripts unter `/proj/i4mw/pub/aufgabe2/post-debootstrap.sh` innerhalb der `chroot`-Umgebung vervollständigt und notwendige Hilfsprogramme (siehe Tafelübung) nachinstalliert werden. Abschließend ist noch ein SSH-Zugang einzurichten, um auf der virtuellen Maschine später Diagnose- und Wartungsarbeiten ausführen zu können. Da in einer Cloud-Umgebung zudem sehr viele Instanzen eines Systems vorhanden sein können, soll die Authentifizierung sicher per SSH-Key statt herkömmlicher Passwörter erfolgen.

Aufgaben:

→ Erzeugung eines leeren, 2 GB großen Volume über die OpenStack-Weboberfläche

→ Starten der Grml-Live-Umgebung und Einrichten von Debian auf dem Volume

→ Konfiguration des Betriebssystems, so dass das System später von OpenStack ausgeführt werden kann und ein Anmelden per SSH möglich ist

2.3.2 Installation der Web-Services

Nach dem Abschluss der grundlegenden Konfiguration des Betriebssystems, sollen der Pfad- sowie der Cache-Dienst in das Abbild integriert werden. Da der Betrieb von Java-Programmen eine Laufzeitumgebung erfordert, welche bei Debian nicht standardmäßig installiert ist, muss diese noch über den Paketmanager installiert werden.

Aufgaben:

→ Installation der Java-Laufzeitumgebung (Paketname: `openjdk-7-jdk`) mit Hilfe des Debian-Paketmanagers

→ Installation des Pfad- sowie des Cache-Diensts in das Verzeichnis `/opt/mwcc-services` des Volume

→ Konfiguration des Betriebssystems, so dass sowohl der Pfad- als auch der Cache-Dienst automatisch beim Hochfahren gestartet werden

2.3.3 Ausführung auf der OpenStack-Private-Cloud

Abschließend soll das in den Teilaufgaben 2.3.1 sowie 2.3.2 erstellte und modifizierte Volume in ein Abbild umgewandelt werden und auf der Private Cloud des Lehrstuhls 4 installiert und ausgeführt werden. Zur Fehlersuche können über die OpenStack-Weboberfläche unter `https://i4cloud1.informatik.uni-erlangen.de/horizon/project/instances/` (Spalte „Actions“→ Button „More → Eintrag „View Log“) die Meldungen der jeweils gestarteten virtuellen Maschine abgerufen werden. Der in Teilaufgabe 2.3.1 installierte SSH-Zugang kann ebenfalls zu Diagnosezwecken verwendet werden.

Aufgaben:

→ Erzeugen und Hochladen eines Abbilds aus dem Volume über die OpenStack-Kommandozeilenwerkzeuge

→ Ausführung des Abbilds als Instanz vom Typ „i4.tiny“ und Test auf korrekte Funktionsweise

Hinweise:

- Die Zugangsdaten für die OpenStack-Weboberfläche (Nutzername und Passwort) entsprechen den Zugangsdaten für das SVN-Repository der eigenen Gruppe.
- Um zu vermeiden, dass es zu Kapazitätsengpässen auf der I4-Cloud kommt, ist darauf zu achten, dass gestartete Instanzen auch wieder beendet werden. Es ist für diese Aufgabe nicht erforderlich, mehr als eine Instanz gleichzeitig zu betreiben.

Abgabe: am 18.11.2015 in der Rechnerübung