2 Übungsaufgabe #2: Hybrid Cloud (OpenStack & AWS)

Hybride Clouds zeichnen sich dadurch aus, dass sie private und öffentliche Clouds miteinander kombinieren und dem Nutzer gegenüber als einheitliches System präsentieren. Eine gängige Vorgehensweise hierbei ist es beispielsweise eine hybride Cloud so aufzubauen, dass sie einen Dienst im Normalfall nur im privaten Teil der Cloud betreibt und nur für die Bewältigung von Lastspitzen zusätzlich auf den öffentlichen Teil zurückgreift. Ziel dieser Aufgabe ist es, die private OpenStack-Cloud des Lehrstuhls mit der öffentlichen AWS-Cloud von Amazon zu solch einer hybriden Cloud zusammenzuschließen. Als Beispielanwendung fungiert hierbei ein Dienst („Tweet-Dienst“), der gesammelte Tweets analysiert und gewonnene Statistiken wie etwa die durchschnittliche Länge von Nachrichten über eine Web-Oberfläche veröffentlicht.

2.1 Generischer Zugriff auf beide Cloud-Teile (für alle)

Zur Verwaltung der zur Verfügung stehenden Cloud-Ressourcen bietet sich eine verallgemeinerte Schnittstelle an, die nach außen unabhängig vom zugrundeliegenden Cloud-Anbieter ist. Angedacht ist hierfür eine Schnittstelle, die im Folgenden für OpenStack und AWS in jeweils eigenen Klassen zu implementieren ist:

```
public interface MWCloudPlatform {
    public MWVirtualMachine startVM(MWVirtualMachineConfig conf) throws MWCloudException;
    public MWVirtualMachine stopVM(MWVirtualMachine machine) throws MWCloudException;
    public List<MWVirtualMachine> listVMs() throws MWCloudException;
}
```

Im Fokus soll bei dieser Aufgabe die Verwaltung von virtuellen Maschinen stehen. Mit Hilfe der Methode `startVM()` kann eine virtuelle Maschine mit der in `MWVirtualMachineConfig` spezifizierten Konfiguration gestartet werden. Analog sorgt ein Aufruf von `stopVM()` dafür, dass die in `MWVirtualMachine` referenzierte virtuelle Maschine terminiert wird. Das Starten einer virtuellen Maschine soll blockierend erfolgen, d. h., es soll so lange gewartet werden, bis die Instanz als lauffähig eingestuft wird. Die Methode `listVMs()` listet alle gegenwärtig registrierten virtuellen Maschinen inkl. dem beim Aufruf aktuellen Zustand (z. B. `RUNNING`) auf. Die Klasse `MWVirtualMachine` hat mindestens Attribute für den Namen, die ID, die öffentliche IP-Adresse und den zuletzt beobachteten Zustand einer virtuellen Maschine (siehe Vorgabe im Pub-Verzeichnis). Die Methoden sollen Fehler (z. B. ungültige Konfiguration) mittels einer `MWCloudException` dem Aufrufer signalisieren. Bevor die einzelnen Methoden aufgerufen werden können, müssen die Dienste zugreifbar gemacht werden, indem eine Cloud-Anbieter-spezifische Authentifizierung erfolgt (z. B. direkt im Konstruktor).

Ein Grundgerüst für die Klasse `MWVirtualMachineConfig`, das beliebig erweitert werden darf, befindet sich im Pub-Verzeichnis. In einer weiteren Klasse `MWCloudController` ist eine Shell zu implementieren, die die Methoden der `MWCloudPlatform`-Schnittstelle für OpenStack und AWS nach außen für den Nutzer zugreifbar macht und das Testen der zu implementierenden Klassen erleichtert.

Aufgaben:

→ OpenStack- bzw. AWS-spezifische Implementierung der `MWCloudPlatform`-Schnittstelle in jeweils eigenen Klassen (`MWCloudPlatformOpenStack` bzw. `MWCloudPlatformAWS`) im Package `mw.hybridcloud`

→ Vervollständigung der vorgegebenen Shell in der Klasse `MWCloudController`

Hinweise:

- Für diese Aufgabe benötigte Java-Klassen/Bibliotheken sind unter `/proj/i4mw/pub/aufgabe2` hinterlegt.
- Zum Testen der eigenen Implementierung soll in dieser Teilaufgabe für die OpenStack-Cloud das bereits bestehende Image `debian-example` zum Einsatz kommen. Für die AWS-Cloud kann beispielsweise das AMI mit der ID `ami-9ca607e5` (Region: `eu-west-1`) verwendet werden.
- Um Kapazitätsengpässe (OpenStack) und hohe Kosten (AWS) zu vermeiden, **sind nicht mehr benötigte Instanzen stets zu beenden**. Aus den selben Gründen sollen darüber hinaus jeweils nur kleine Instanzen gestartet werden (OpenStack: `i4.tiny`; AWS: `t2.nano`);
- Für OpenStack ist das Netzwerk `internal` zu verwenden; im Falle von AWS sind alle Instanzen in der Region *EU (Ireland)* zu starten; vor dem Starten von bestimmten AMIs ist ggf. eimalig ein VPC-Netzwerk mit einer eigenen Subnet-ID zu erzeugen (siehe Tafelübung).
- Die Zugangsdaten zur OpenStack-Weboberfläche (Nutzername und Passwort) wurden jeder Gruppe am Semesteranfang zusammen mit der Benachrichtigung über die erfolgreiche Gruppenanmeldung mitgeteilt.

2.2 Ermittlung der CPU-Auslastung virtueller Maschinen (optional für 5,0 ECTS)

Um erfolgreich auf Lastspitzen reagieren zu können ist es für das eigene hybride Cloud-System essentiell, diese bei ihrem Auftreten zunächst überhaupt erst zu erkennen. Eine mögliche Lösung dieses Problems, für die im Rahmen dieser Teilaufgabe die Grundlage geschaffen werden soll, besteht darin, die CPU-Auslastung der virtuellen Maschinen im System kontinuierlich zu überwachen. OpenStack bzw. AWS stellen hierfür jeweils eine eigene Schnittstelle zum Zugriff auf verschiedene Metriken zur Verfügung, über die unter anderem Informationen zur Prozessornutzung einer virtuellen Maschine abrufbar sind. Diese Informationen sollen nun durch Hinzufügen einer weiteren Methode zur `MWCloudPlatform`-Schnittstelle aus Teilaufgabe 2.1 sowie der Shell in der Klasse `MWCloudController` für die hybride Cloud nutzbar gemacht werden.

```
public Double getCPUUsage(MWVirtualMachine machine, int seconds) throws MWCloudException;
```

Die Methode `getCPUUsage()` liefert für die virtuelle Maschine `machine` die durchschnittliche CPU-Auslastung der vergangenen `seconds` Sekunden als Gleitkommawert.

Aufgabe:

→ Erweiterung der Klassen `MWCloudPlatformAWS` und `MWCloudPlatformOpenStack`
→ Erweiterung der `MWCloudController`-Shell

2.3 Bereitstellung des Tweet-Diensts (für alle)

Die Server-Implementierung des als Anwendung in der hybriden Cloud auszuführenden Diensts zur Analyse von Tweets existiert bereits als Java-Archiv. Zur Interaktion mit anderen Systemkomponenten verfügt der Tweet-Dienst über folgende REST-Schnittstelle:

`POST`	`http://<server>/tweetservice/process`	Übermittlung zu analysierender Tweets
`GET`	`http://<server>/tweetservice`	Bereitstellung der ermittelten Ergebnisse

Ziel dieser Teilaufgabe ist es, die Server-Seite des Tweet-Diensts in den beiden Teilen der hybriden Cloud bereitzustellen. Hierfür sind für AWS und OpenStack unterschiedliche Schritte erforderlich.

2.3.1 Bereitstellung des Diensts in der öffentlichen Cloud

Für Amazon EC2 ist in der Region *EU (Ireland)* ein Maschinenabbild (*ami-9ca607e5*) vorhanden, das – abgesehen von der Dienstimplementierung selbst – bereits über sämtliche Bibliotheken verfügt, die zum Betrieb des Tweet-Diensts gebraucht werden. Darüber hinaus ist das Abbild so konfiguriert, dass beim Start einer virtuellen Maschine automatisch ein Skript (`http://i4mw.s3.amazonaws.com/rc.local.i4mw`) ausgeführt wird, welches das Java-Archiv mit der Dienstimplementierung (`MWTweetService.jar`) aus Amazon Amazon S3 herunterlädt und den Dienst startet. Hierfür benötigt das Skript zusätzliche Aufrufparameter, die der jeweiligen virtuellen Maschine beim Start über das EC2-Nutzdatenfeld `user-data` zu übergeben sind. In diesem Feld sind die Aufrufparameter als Schlüssel-Wert-Paare mit dem Format `key=value` verzeichnet, die *ausschließlich* durch einzelne Semikola voneinander getrennt sind. Das Startskript wertet diese Metadaten aus und prüft, ob folgende Schlüssel existieren: `group`, `jar`; optional: `parameters`, `java_defines`; Beispiel: `group=gruppe0-bucket;jar=MWSimpleTestService.jar;parameters=MWSimpleTestService 42 4711`.

Aus den Angaben für die Schlüssel `group` und `jar` erzeugt das Startskript die URL `http://<group>.s3.amazonaws.com/<jar>`. Der Wert des Schlüssels `<group>` spezifiziert also den S3-Bucket und der Wert des Schlüssels `<jar>` den Dateinamen des in diesem Bucket abgelegten Java-Archivs. Wenn das Startskript in der Lage ist, erfolgreich auf die Daten zuzugreifen, wird im Folgenden das spezifizierte Java-Archiv auf die Instanz geladen und ausgeführt. Die Parameter `java_defines` und `parameters` werden dabei wie folgt interpretiert:

```
$ java -cp <jar>:[...] -D<java_defines> <parameters>
```

Die `main()`-Methode des Tweet-Diensts ist in der Klasse `mw.hybridcloud.MWTweetService` implementiert und erwartet als ersten Parameter die URL (`http://<server>/tweetservice`) des Diensts.

Aufgaben:

→ Erstellen eines Bucket auf Amazon S3 und dortiges Hinterlegen des Java-Archivs vom bereitgestellten Tweet-Dienst unter `/proj/i4mw/pub/aufgabe2/MWTweetService.jar`
→ Testen des Startvorgangs

Hinweise:

- Zur Laufzeit des Startskriptes (z. B. beim Start von Java) ist die öffentliche IP-Adresse der aktuellen Instanz in der Umgebungsvariable `I4MW_ADDRESS` vermerkt.
- Beim Zugriff auf die in Amazon S3 hinterlegten Daten durch die Instanzen wird auf eine Authentifizierung verzichtet, daher muss das Java-Archiv unter Kenntnis der URL öffentlich zugänglich sein.

2.3.2 Bereitstellung des Diensts in der privaten Cloud

Im Gegensatz zur öffentlichen AWS-Cloud ist für die private OpenStack-Cloud noch kein Abbild für virtuelle Maschinen des Tweet-Diensts vorhanden, sondern muss erst noch erstellt werden. Hierfür ist es erforderlich zunächst eine entsprechend angepasste Betriebssysteminstallation zu erstellen, diese auf den Cloud-Dienst hochzuladen und schließlich zu starten.

Einrichtung des Betriebssystems Um den Dienst in der Cloud-Umgebung ausführen zu können, soll im Folgenden ein eigenes OpenStack-Betriebssystemabbild erstellt werden. Als grundlegendes Betriebssystem ist dabei Debian *Stretch* vorgesehen. Das eigene Abbild soll unter Zuhilfenahme der Linux-Live-CD *Grml* erzeugt werden. Ein spezielles Abbild mit dem Namen `GRML-2017.05-amd64` steht dafür bereits zur Verfügung, so dass davon eine Instanz gestartet werden kann. Darüber hinaus ist in OpenStack 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 der laufenden Grml-Instanz zugewiesen (engl. *attached*) wurde. Im abschließenden Schritt soll aus diesem Volume später 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 im Volume speichern zu können. Nach der Installation einer Minimalversion von Debian mittels `debootstrap` erfolgt die Vervollständigung mittels des vorgefertigten Skripts `/proj/i4mw/pub/aufgabe2/post-debootstrap.sh` und Nachinstallation notwendiger Hilfsprogramme (siehe Tafelübung) innerhalb der `chroot`-Umgebung. Das Installationsskript richtet zudem einen SSH-Zugang ein, 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 & dortiges Einrichten von Debian mit Grml-Live-Umgebung

→ Konfiguration des Betriebssystems zur Ausführung auf OpenStack und Einrichten von SSH-Zugang

Installation des Web-Services Nach dem Abschluss der grundlegenden Betriebssystemkonfiguration sollen die für die Ausführung des Tweet-Dienstes benötigten Daten 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-8-jdk`) mittels Debian-Paketmanager

→ Installation der für den Tweet-Dienst benötigten Bibliotheken in das `/proj/lib`-Verzeichnis des Volume

Hinweis:

- Mit IP-Adresse `0.0.0.0` veröffentlichte Dienste sind über alle Netzwerkschnittstellen erreichbar. Damit ist keine Instanz-spezifische IP-Adresse nötig; externe Zugriffe benötigen weiterhin die öffentliche IP-Adresse.

Ausführung auf der OpenStack-Private-Cloud Abschließend soll das in den vorherigen Teilaufgaben erstellte und modifizierte Volume in ein Abbild umgewandelt werden und auf der privaten OpenStack-Cloud des Lehrstuhl 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 zuvor 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

2.4 Lasterzeugung und Lastverteilung (für alle)

Nachdem der Tweet-Dienst in beiden Teilen der hybriden Cloud lauffähig ist, besteht der nächste Schritt darin sicherzustellen, dass aktive Instanzen des Diensts (mindestens zwei pro Cloud-Teil) gleichmäßig mit Eingaben versorgt werden. Als Grundlage soll hierbei die bereits bestehende Klasse `MWTweetProvider` dienen, die so vorbereitet wurde, dass sie Tweets (siehe `/proj/i4mw/pub/aufgabe2/tweets/`) aus der Datei im übergebenen Pfad kontinuierlich von einem ebenfalls existierenden und bereits im Code-Gerüst verankerten `MWTweetReader` einliest. Noch zu implementieren ist dagegen die Entscheidung, an welche Instanz des Tweet-Diensts ein Bündel

neu eingelesener Tweets verschickt werden soll. Hierfür muss die im Provider-Code-Gerüst momentan leer implementierte Methode `getNextTarget()` entsprechend ergänzt werden. Diese Methode gibt einen Proxy in Form eines `WebTarget`-Objekts zurückgibt, der mit dem Target `http://<server>/tweetservice` initialisiert wird, wobei `<server>` von der jeweiligen IP der virtuellen Maschine abhängig ist.

Damit die einzelnen Cloud-Instanzen für den Tweet-Provider erreichbar sind, müssen ihm die Adressen also bekanntgemacht werden. Dies soll mit Hilfe der bereits in Aufgabe 1 verwendeten I4-Registry erfolgen, indem der `MWRegistryClient` wiederverwendet wird. Der `MWCloudController` aus Teilaufgabe 2.1 muss hierzu noch so erweitert werden, dass die Adressen virtueller Maschinen beim Starten in die Registry eingetragen und beim Beenden wieder ausgetragen werden. Die Struktur der Registry-Einträge im Gruppen-Slot ist frei wählbar.

Der `MWTweetProvider` soll sich die für ihn relevanten Einträge in regelmäßigen Zeitabständen neu aus der Registry holen und die entsprechenden Proxys (`WebTarget`-Objekte) zur Kommunikation mit dem Dienst ggf. neu erzeugen. Liegen ein oder mehrere Proxys vor, verteilt der `MWTweetProvider` die vom `MWTweetReader` erzeugten Anfragen an die Tweet-Dienst-Instanzen.

Aufgaben:

- → Erweiterung der Klasse `MWCloudController` um das Anmelden und Abmelden von virtuellen Maschinen über die I4-Registry, welche im `MWTweetProvider` periodisch abgerufen werden soll
- → Erweiterung der Klasse `MWTweetProvider` um die gleichmäßige Verteilung der vom `MWTweetReader` erzeugten Tweets auf die Cloud-Instanzen; falls verfügbar, unter Zuhilfenahme der in Teilaufgabe 2.2 bereitgestellten Methode für die CPU-Auslastung

2.5 Dynamische Skalierung von virtuellen Maschinen (optional für 5,0 ECTS)

Beim Starten und Beenden von Instanzen muss nun zwischen Instanzen der Private- und Public-Cloud unterschieden werden. In der Private-Cloud dürfen maximal zwei Instanzen gestartet werden. Wenn die beiden Private-Cloud-Instanzen nicht mehr ausreichend sind, sind weitere Instanzen in der Public-Cloud zu starten. In dieser Teilaufgabe gilt es, einen zusätzlichen automatischen VM-Skalierungsmodus im `MWCloudController` zu implementieren. Nach dem Programmstart ist der Cloud-Controller zunächst dafür verantwortlich, eine initiale, den Tweet-Dienst automatisch ausführende Server-Instanz in der OpenStack-Cloud zu starten (→ `startVM()`, siehe Teilaufgabe 2.1) und die Instanz anschließend in der I4-Registry zu registrieren (vgl. Teilaufgabe 2.4). Sobald die initiale Server-Instanz betriebsbereit ist, können vom `MWTweetProvider` Anfragen an die Server-Instanz gesendet werden. Im weiteren Betrieb des Systems ist der Cloud-Controller dafür zuständig, kontinuierlich die Systemlast zu überwachen und im Falle einer Über- oder Unterbeanspruchung der Server-Instanzen entsprechende Gegenmaßnahmen einzuleiten. Um festzustellen, ob der Web-Service über- oder unterbelastet ist, soll der Cloud-Controller durch periodisches Abrufen der CPU-Auslastung (siehe Teilaufgabe 2.2) die momentane Last der Server in Erfahrung bringen und anschließend mit einem selbst zu bestimmenden Schwellwert vergleichen. Bei Überbeanspruchung gilt es, neue Server-Instanzen zu starten. Stellt der Cloud-Controller hingegen eine Unterbeanspruchung des Dienstes fest, sollen nicht mehr benötigte Ressourcen freigegeben werden. Zu jedem Zeitpunkt soll mindestens eine Instanz des Tweet-Dienstes in der Private-Cloud verfügbar sein.

Damit durch kurzfristige Lastspitzen bzw. kurzfristiges Ausbleiben von Tweets keine Instanzen unnötig gestartet bzw. terminiert werden, gilt es entsprechende Vorkehrungen zu treffen. Solch ein Schutz kann z. B. durch Zähler oder Timer realisiert werden, aber auch durch Abrufen der CPU-Auslastung über ein geeignetes Zeitintervall.

Aufgabe:

- → Erweiterung der Klasse `MWCloudController` um automatisierte, dynamische VM-Skalierung

2.6 Visualisierung der erhobenen Tweet-Dienst-Daten (optional für 5,0 ECTS)

Amazon CloudWatch ermöglicht auch die Erfassung von eigenen Metriken, die aus der Anwendung heraus injiziert werden können. Der Tweet-Dienst erhebt beim Verarbeiten der Tweets statistische Daten (z. B. verarbeitete Tweets pro Sekunde), die per `GET`-Anfrage abgerufen werden können (siehe Teilaufgabe 2.3). Die als JSON übertragenen Daten werden durch die Klasse `MWTweetResponse` repräsentiert (siehe Pub-Verzeichnis). In einer neuen Methode der `MWTweetProvider`-Klasse sollen nun mittels dieser REST-Anfrage auf dem Tweet-Dienst die Daten jedes Knotens eingeholt, miteinander kombiniert und mittels Custom-Metriken nach Amazon CloudWatch geladen werden; anschließend sind diese mit Hilfe eines Dashboards im Web-Interface geeignet zu visualisieren. Im Dashboard sind nur diejenigen Daten anzuzeigen, die sinnvoll durch einen einzelnen `Double`-Wert repräsentiert werden können.

Aufgaben:

- → Sammeln und Kombinieren der von den Tweet-Dienst-Instanzen generierten Daten
- → Laden der Daten auf Amazon CloudWatch und Visualisierung in einem Dashboard der AWS-Weboberfläche

Abgabe: am 22.11.2017 in der Rechnerübung