

Überblick

Hybrid Clouds & Virtualisierung
Einführung
Aufbau einer virtuellen Maschine

Amazon Web Services
Überblick
Elastic Compute Cloud (EC2)
Simple Storage Service (S3)
Amazon CloudWatch
Amazon Java SDK

Aufgabe 2
Aufgabenstellung
OpenStack

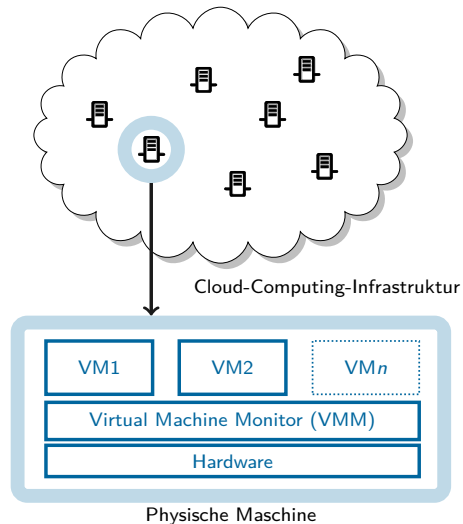


Hybrid Clouds

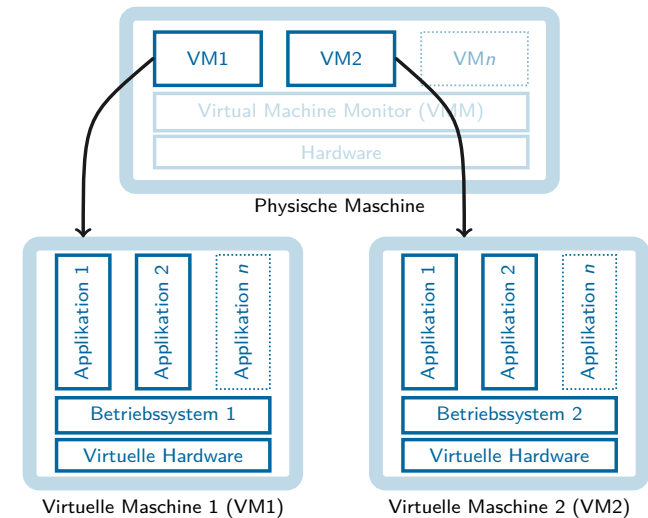
- Public Cloud: Cloud-Dienste frei für jeden verfügbar
 - *aaS: „X as a Service“-Gedanke
 - Umfangreiche Kostenmodelle
- Private Cloud: IT- bzw. Cloud-Dienste werden z. B. von einem Unternehmen oder einer Einrichtung selbst betrieben
 - Interne Nutzung: Datenschutz und IT-Sicherheit
 - Aber auch: Bereitstellung von eigenen Ressourcen für öffentliche Nutzung
- **Hybrid Cloud:** Mischform aus Private und Public Cloud
 - Sicherheitskritische Teile einer Anwendung laufen nur in der Private Cloud
 - Skalierbarkeit, Ausdehnung auf (kostenintensivere) Public-Cloud-Dienste (z. B. beim Auftreten von Lastspitzen)



Virtualisierung als Grundlage für Cloud Computing



Aufbau einer virtuellen Maschine



Aufbau einer virtuellen Maschine

- Notwendige Betriebsmittel
 - Physische Maschine und Gastgeberbetriebssystem („Host“)
 - Virtualisierungssoftware, die den Virtual Machine Monitor bereitstellt
 - **Abbild der zu betreibenden virtuellen Maschine**
- Aufbau des Abbilds einer virtuellen Maschine
 - Meta-Informationen (spezifisch, je nach Virtualisierungssoftware)
 - **Dateisystem**, beinhaltet für gewöhnlich:
 - Kern des zu virtualisierenden Gastbetriebssystems („Guest“)
 - User-Space-Komponenten des Gastbetriebssystems
 - Daten
- Analogie zur Objektorientierung
 - Das statische Abbild einer virtuellen Maschine entspricht einer **Klasse**
 - Eine im Betrieb befindliche virtuelle Maschine ist die **Instanz** eines solchen Abbilds

Überblick

Hybrid Clouds & Virtualisierung
Einführung
Aufbau einer virtuellen Maschine

Amazon Web Services
Überblick
Elastic Compute Cloud (EC2)
Simple Storage Service (S3)
Amazon CloudWatch
Amazon Java SDK

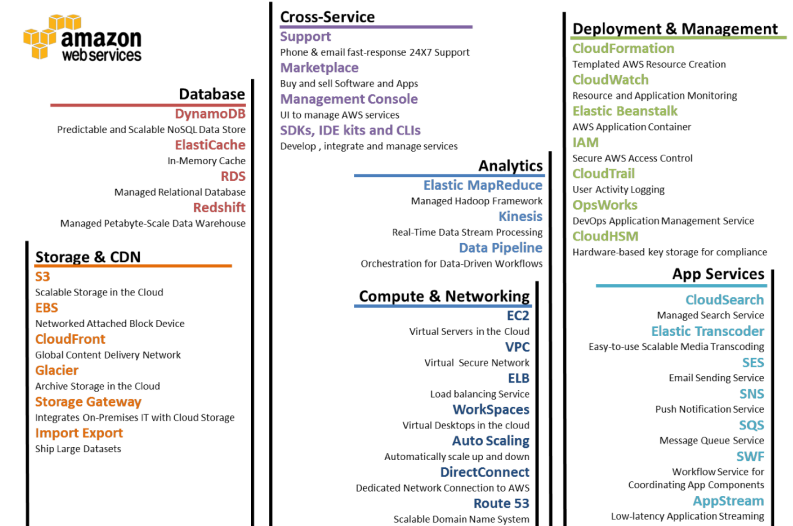
Aufgabe 2
Aufgabenstellung
OpenStack

Amazon Web Services (AWS)

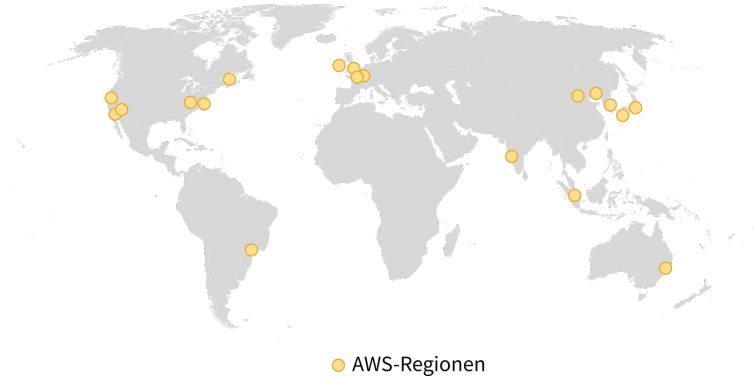
- Die Amazon Web Services bestehen aus Diensten, die den Aufbau komplexer Systeme in einer Cloud-Infrastruktur ermöglichen
- Dienste (Auszug):
 - Elastic Compute Cloud (EC2) – Betrieb virtueller Maschinen
 - Simple Storage Service (S3) – Netzwerkbasierter Speicher-Dienst
 - Elastic Load Balancing – Lastverteilung für EC2
 - Elastic Map Reduce – MapReduce-Framework basierend auf EC2 und S3
 - DynamoDB – Key-Value-Store basierend auf Dynamo
- Die Abrechnung erfolgt nach tatsächlichem Verbrauch **und** Standort
 - Betriebsstunden, Speicherbedarf
 - Transfervolumen, Anzahl verarbeiteter Anfragen
 - Standorte in Nord- und Südamerika, Europa und Asien-Pazifik
 - Berechnung der Gesamtbetriebskosten: <https://awstccalculator.com/>

G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian et al.
Dynamo: Amazon's Highly Available Key-value Store.
In *Proc. of the 21st Symposium on Operating Systems Principles*. ACM, 2007.

Amazon Web Services (AWS)



Amazon Web Services (AWS)



Amazon Web Services: Betriebsumgebung

- Benutzung der Amazon Web Services (u. a.) über Web-Oberfläche
→ `https://i4mw-gruppeXX.signin.aws.amazon.com/console`
(XX durch eigene Gruppennummer ersetzen)
↪ Login-Informationen befinden sich in der Gruppeneinteilungs-E-Mail
↪ Immer die Region eu-west-1 verwenden

■ AWS CLI: AWS-Befehlszeilen-Schnittstelle

```
alias aws=/proj/i4mw/pub/aufgabe2/aws/bin/aws
```

- Diesen Befehl am besten in die Datei `~/.profile` eintragen, damit die AWS CLI nach jedem CIP-Pool-Login funktionieren
- Python-Werkzeug zum Zugriff auf sämtliche AWS-Dienste
- Konfiguration: Setzen der Zugangsdaten und Region. Siehe nächste Folie

■ Liste der verfügbaren AWS-Kommandozeilen-Tools:

```
> aws help  
> aws <service> <command> help
```

Amazon Web Services: Betriebsumgebung

- Ablegen der Credentials zum API-Zugriff in Datei empfehlenswert für spätere Übergabe an Programme, welche auf die API zugreifen

↪ Konfiguration auch mit `> aws configure` möglich

- 1) Anlegen der privaten Konfigurationsdateien `~/.aws/credentials` und `~/.aws/config` mit eingeschränkten Zugriffsrechten

```
> mkdir ~/.aws  
> touch ~/.aws/credentials ~/.aws/config  
> chmod 600 ~/.aws/*
```

- 2) Erstellen von `aws_access_key_id` und `aws_secret_access_key` über die Web-Oberfläche:
`https://console.aws.amazon.com/iam/`, Menüpunkt „Users“, eigenen Namen anklicken, Reiter „Security Credentials“, Abschnitt „Access Keys“

→ Eintragen in `~/.aws/credentials`

```
[default]  
aws_access_key_id = <schlüssel_id>  
aws_secret_access_key = <privater_schlüssel>
```

- 3) Setzen der Region in `~/.aws/config`

```
[default]  
region = eu-west-1
```

Amazon Elastic Compute Cloud (EC2)

- Voraussetzungen für die Instanziierung einer virtuellen Maschine
 - Amazon Machine Image (AMI, Liste: `aws ec2 describe-images`)
 - EC2-Schlüsselpaar
 - VPC-Netzwerk
- Bei der Instanziierung muss die Größe der virtuellen Maschine festgelegt werden
 - Instanz-Typen variieren in Anzahl der CPU-Kerne, Speichergröße etc.
→ `http://aws.amazon.com/ec2/instance-types/`
 - Für Testzwecke reicht der Betrieb kleiner Instanzen aus
→ API-Name: `t2.nano`
- Nutzdatenfeld `user-data`
 - Base64-kodierter String
 - Maximal 16 kByte
 - Optional

Amazon EC2: Starten einer Instanz

Vorbereitung

- Einmalig EC2-Schlüsselpaar im Browser generieren
 - Schlüsselname wählen (z. B. gruppe0)
 - Privaten Schlüssel unter `~/ .aws/gruppe0.pem` speichern
 - <https://console.aws.amazon.com/ec2/home?region=eu-west-1#s=KeyPairs>
 - Zugriffsrechte mit `chmod` setzen

```
> chmod 600 ~/ .aws/gruppe0.pem
```
- VPC-Netzwerk inklusive Subnetz nötig
 - Existiert bereits im zur Verfügung gestellten AWS-Account
 - In Web-Oberfläche konfigurierbar (optional): <http://tinyurl.com/aws-vpcs>
- Security-Group für Port-Freigaben einrichten
 - Basis-Security-Group bereits im AWS-Account vorhanden (Name: `i4mw`)
 - **Achtung:** Erlaubt nur Kommunikation zwischen VMs in AWS
 - Für SSH externe Zugriffe über das TCP-Protokoll mit Port 22 von `0.0.0.0/0` (CIDR-Notation, entspricht weltweitem Zugriff) freigeben!
 - Änderungen möglich während Instanz läuft
 - In Web-Oberfläche konfigurierbar: <http://tinyurl.com/aws-sgroups>



Amazon EC2: Starten einer Instanz

- Starten einer Linux-Instanz:
 - Instanz-Typ: `t2.nano`
 - AMI: `ami-047bb4163c506cd98` [↔ Amazon Linux AMI]
 - Schlüsselname (`<key>`): beim Erstellen selbst gewählt (z. B. `gruppe0`)
 - Nutzdatenfeld mit String füllen (`<user-data>`): z. B. `Hello World`.
 - `<subnet-id>`: Ermitteln der ID (SubnetId) eines VPC-Subnetzes z. B. über

```
> aws ec2 describe-subnets | grep -i subnetid
```

 - `<sg-id>`: Ermitteln der ID (GroupID) der Security-Group `i4mw` z. B. über

```
> aws ec2 describe-security-groups | grep -B3 -i groupid
```
 - Starten über die Kommandozeile
- ```
> aws ec2 run-instances --instance-type t2.nano \
 --image-id ami-047bb4163c506cd98 \
 --key <key> --user-data="<user-data>" \
 --subnet-id <subnet-id> \
 --security-group-ids <sg-id>
```



## Amazon EC2: Zugriff auf eine Instanz

- Überprüfen des Status der Instanz mit `aws ec2 describe-instances`; auch: Abfrage der öffentlichen IP-Adresse (→ `PublicIpAddress`)
  - Sobald die Instanz den Boot-Vorgang abgeschlossen hat, erfolgt der Zugriff auf die Maschine mittels SSH
- ```
> ssh -i ~/ .aws/gruppe0.pem \
  ec2-user@ec2-xxx-xxx-xxx-xxx.eu-west-1.compute.amazonaws.com
```
- Bei Konflikten aufgrund erneuter Adressvergabe, alten SSH-Host-Key entfernen:
`ssh-keygen -R <server_address>`
 - Hinweise:
 - In der Betriebsumgebung der virtuellen Maschine werden mit `ec2-metadata` Meta-Informationen über das System angezeigt. Auch das Nutzdatenfeld `user-data` kann so ausgewertet werden.
 - Root-Rechte erhält man mit dem Kommando `sudo su -`
 - Debugging auf dem Live-System: Prüfen der Logs `/var/log/syslog`
 - Bei Zugriffsproblemen: Boot-Meldungen über die Web-Schnittstelle oder mit `aws ec2 get-console-output` nach Fehlern durchsuchen



Amazon EC2: Beenden einer Instanz

- Für das Terminieren einer im Betrieb befindlichen Instanz ist die eindeutige Instanz-ID notwendig
 - Das Kommando `aws ec2 describe-instances` listet die Instanz-ID in der zweiten Spalte (Format: `i-xxxxxxx`)
 - Unter Kenntnis dieser ID kann die Instanz mit `aws ec2 terminate-instances` beendet werden:
- ```
> aws ec2 describe-instances
(...)
> aws ec2 terminate-instances i-xxxxxxx
```
- Kontrolle: <https://console.aws.amazon.com/ec2/home>
  - Bitte stets sicherstellen, dass **alle unbenutzten** Instanzen beendet werden!



## Amazon Simple Storage Service (S3)

- Der Simple Storage Service (S3) ist ein Netzwerk-Dateisystem
  - REST-, SOAP- und BitTorrent-Schnittstellen
  - Zugriffskontrolle mittels Zugriffskontrolllisten (Access Control Lists, ACLs)
  - Einfache API
- Eindeutige Identifikation von Dateien durch Bucket (Kübel) und Dateiname:  
s3://<bucket>/<dateiname>
- Buckets können *nicht* geschachtelt werden
- Übersetzung der S3-Adressrepräsentation in eine URL
  - S3: s3://<bucket>/<dateiname>
  - URL: http://<bucket>.s3.amazonaws.com/<dateiname>
- Prominente Dienste, die S3 nutz(t)en:
  - Netflix
  - Dropbox
  - Twitter (Bildern)



## Amazon S3: Zugriff auf Daten

- Zugriff auf Daten in S3 im CIP-Pool via

```
> aws s3 <befehl>
```

- cp / rm / mv
- mb / rb
- ls
- ...

- Erstellen eines Bucket:

```
> aws s3 mb s3://gruppe0-bucket
make_bucket: gruppe0-bucket
```

- Speichern einer *öffentlichen* Datei im Bucket gruppe0-bucket:

```
> echo "Hello World." > foo.bar
> aws s3 cp --acl public-read foo.bar s3://gruppe0-bucket/foo.bar
upload: foo.bar to s3://gruppe0-bucket/foo.bar
```



## Amazon S3: Zugriff auf Daten

- Laden der Datei foo.bar aus dem Bucket gruppe0-bucket:

```
> aws s3 cp s3://gruppe0-bucket/foo.bar foo.bar.copy
download: s3://gruppe0-bucket/foo.bar to foo.bar.copy
```

- Löschen der Datei foo.bar aus dem Bucket gruppe0-bucket:

```
> aws s3 rm s3://gruppe0-bucket/foo.bar
delete: s3://gruppe0-bucket/foo.bar
```

- Ausführliche Liste mit Beschreibungen der s3-Befehle:

```
> aws s3 help
```

- Alternative Zugriffsmethoden:

- Browser (Amazon Web Services Console, <https://console.aws.amazon.com/s3/home>)
- Einhängen als Dateisystem (s3fs, FUSE-basiert)



## Amazon CloudWatch

- Umfangreiche Überwachungsfunktionen für viele AWS-Dienste
- Protokollierung und lange Speicherung der Daten

- Metriken: in beide Richtungen (Lesen/Schreiben) möglich

- Grundlegende Überwachung (5 Minuten), kostenlos
- Detaillierte Überwachung (1-Min.-Intervalle), zusätzliche Kosten
- Benutzerdefinierte Metriken: aus Anwendung heraus, selbst definierbar

- Alarme: Automatische Reaktion bei auffälligen Veränderungen

- Visualisierung: Darstellung der Daten in einem Dashboard möglich

<https://eu-west-1.console.aws.amazon.com/cloudwatch> → „Dashboard“

- Beispiele

- Amazon EC2: CPU-Auslastung, gesendete/empfangene Netzwerkpakete
- Amazon EBS: Lese- und Schreiblatenz
- Amazon RDS: verfügbarer Arbeitsspeicher/Speicherplatz



## Amazon Java SDK

- Amazon stellt eine Java-Bibliothek für die Verwendung der Amazon Web Services zur Verfügung

/proj/i4mw/pub/aufgabe2/aws/aws-java-sdk-1.11.639/lib/aws-java-sdk-1.11.639.jar

3rd-Party-Bibliotheken: /proj/i4mw/pub/aufgabe2/aws/aws-java-sdk-1.11.639/third-party/lib

→ <http://docs.amazonwebservices.com/AWSJavaSDK/latest/javadoc/index.html>

- Relevante Packages für den Betrieb von virtuellen Maschinen in Amazon EC2 und Amazon CloudWatch:

- `com.amazonaws.services.ec2`
- `com.amazonaws.services.cloudwatch`

- Folgende Objekte sind bei der Instanziierung einer virtuellen Maschine in EC2 beteiligt

1. Client-Objekt (Typ `AmazonEC2`)
2. Instanzierungs-Request (Typ `RunInstancesRequest`)
3. Ergebnis (Typ `RunInstancesResult`)



## Amazon Java SDK: Instanziierung einer VM

- Minimal-Beispiel (analog Kommandozeilen-Beispiel)

**Beachte:** Vor dem Aufruf am `AmazonEC2ClientBuilder` müssen in der Konfigurationsdatei `~/.aws/credentials` die Optionen `aws_access_key_id` und `aws_secret_access_key` gesetzt sein.

- Initialisierung `com.amazonaws.services.ec2`

```
AmazonEC2 ec2 = AmazonEC2ClientBuilder.standard()
 .withRegion("eu-west-1").build();
```

- Setzen des Namens einer VM-Instanz `com.amazonaws.services.ec2.model`

```
Tag tag = new Tag("Name", "MyVMName");

TagSpecification spec = new TagSpecification()
 .withTags(tag)
 .withResourceType("instance");

[...] // Fortsetzung auf der naechsten Folie
```



## Amazon Java SDK: Instanziierung einer VM

- Minimal-Beispiel (Fortsetzung) `com.amazonaws.services.ec2.model`

```
String userData = "Hello world.";
byte[] userDataBytes = userData.getBytes();
```

```
RunInstancesRequest request = new RunInstancesRequest()
 .withImageId("ami-047bb4163c506cd98")
 .withTagSpecifications(spec)
 .withInstanceType("t2.nano")
 .withMinCount(1)
 .withMaxCount(1)
 .withKeyName("gruppe0-key")
 .withUserData(Base64.encodeAsString(userDataBytes)) // com.amazonaws.util
 .withMonitoring(true) // optional, detailliertere Metriken aktivieren
 .withSecurityGroupIds("sg-989f5ce3") // z.B. im Web-Interface erstellen
 .withSubnetId("subnet-0eab946a"); // (VPC muss Security-Group vorab
 // zugeordnet werden)

RunInstancesResult result = ec2.runInstances(request);
```

- Hinweise:

- Mittels des Objektes `result` die Instanz-ID in Erfahrung bringen
- Auf die eigentliche Instanziierung prüfen (`DescribeInstancesRequest`)



## Amazon Java SDK: CloudWatch

- Initialisierung (ähnlich wie bei EC2) `com.amazonaws.services.cloudwatch`

```
AmazonCloudWatch cw = AmazonCloudWatchClientBuilder.standard()
 .withRegion("eu-west-1").build();
```

- Metrik abrufen: Zeitintervall und Dimension festlegen

- Erwartetes Zeitformat: ISO 8601, UTC (z. B. 2017-11-03T13:45:00Z)
- Beispielhaftes Definieren von Anfangs- und Endzeitpunkt

```
// Packages: java.time.Clock, java.time.Instant
Instant e = Clock.systemUTC().instant();
Instant s = e.minusMinutes(2); // Datenpunkte ueber 2-Min.-Intervall
```

```
// Package: com.amazonaws.services.cloudwatch.model.Dimension
Dimension dimension = new Dimension()
 .withName("InstanceId")
 .withValue("i-xxxxxxx");
```

- Weiterführende Links

- [https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch\\_concepts.html](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_concepts.html)
- [https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API\\_GetMetricStatistics.html](https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_GetMetricStatistics.html)
- [https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/viewing\\_metrics\\_with\\_cloudwatch.html](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/viewing_metrics_with_cloudwatch.html)



## Amazon Java SDK: CloudWatch

### Metrik abrufen (Fortsetzung)

com.amazonaws.services.cloudwatch.model

```
// Request zum Holen der Werte einer Metrik zusammensetzen und absenden
// festlegen, dass nur Durchschnittswerte abgefragt werden
GetMetricStatisticsRequest req = new GetMetricStatisticsRequest()
 .withStatistics(Statistic.Average)
 .withPeriod(60)
// Package: java.util.date
 .withStartTime(Date.from(s))
 .withEndTime(Date.from(e))
 .withMetricName("DiskWriteBytes")
 .withNamespace("AWS/EC2")
 .withDimensions(dimension);
GetMetricStatisticsResult res = cw.getMetricStatistics(req);

// Zeitstempel und Durchschnittswerte ausgeben
for (Datapoint dp : res.getDatapoints()) {
 System.out.printf("%s: %s\n", dp.getTimestamp().getTime(),
 dp.getAverage());
}
```



## Überblick

### Hybrid Clouds & Virtualisierung

- Einführung
- Aufbau einer virtuellen Maschine

### Amazon Web Services

- Überblick
- Elastic Compute Cloud (EC2)
- Simple Storage Service (S3)
- Amazon CloudWatch
- Amazon Java SDK

### Aufgabe 2

- Aufgabenstellung
- OpenStack



## Aufgabe 2

### Web-Service in der hybriden Cloud-Computing-Umgebung ausführen

- Cloud-Ansteuerung
- Lastverteilung
- Dynamische Skalierung

### Basis-Technologien

- Amazon EC2 vs. OpenStack Nova
- Amazon CloudWatch vs. OpenStack Ceilometer/Gnocchi
- Amazon EBS vs. OpenStack Glance/Cinder
- Amazon S3 (vs. OpenStack Swift)

### Amazon Web Services

- Rund 10 US-Dollar Guthaben pro Gruppe
- Guthaben kann lediglich für Amazon Web Services verwendet werden
- Aktuelle AWS-Kosten: <http://aws.amazon.com/pricing/>

### Globaler Systemstatus der Amazon Web Services

- Bei Störungen können (Teile der) Amazon Web Services ausfallen
- Aktueller Status: <http://status.aws.amazon.com/>

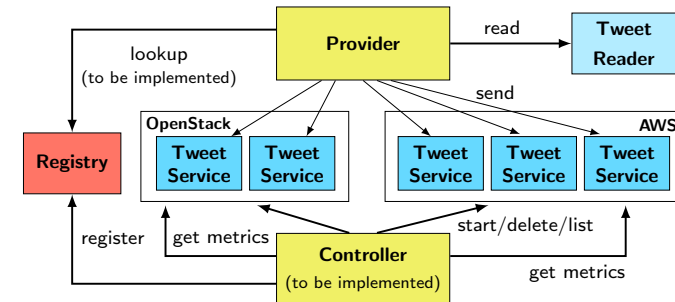


## Aufgabe 2: Hybrid Cloud

### Tweet-Service wird bereitgestellt, Registry(-Zugriff) wie gehabt

### Teilaufgaben

- Cloud-Controller für manuelle Cloud-Instanzen-Ansteuerung (VMs starten, beenden, auflisten, nach Metriken abfragen)
- Verteilung von Tweet-Anfragen im Provider
- Erweiterter Cloud-Controller zur dynamischen Steuerung der VMs



## Aufgabe 2: Hybrid Cloud

- Gemeinsame Schnittstelle: MWCloudPlatform
- Betrieb des Dienstes in Amazon EC2
  - Spezifische Klasse zur VM-Ansteuerung: MWCloudPlatformAWS
  - AMI wird bereitgestellt
  - Java & Java-Bibliotheken für Dienst
    - Java-Bibliotheken bereits im Image (ami-02918453c99a7a5c7) enthalten
    - Installierte Java-Version im AMI: OpenJDK Runtime Environment 11
  - Hinterlegen des JAR-Archivs des Dienstes auf S3
- Betrieb des Dienstes in OpenStack
  - Spezifische Klasse zur VM-Ansteuerung: MWCloudPlatformOpenStack
  - Erzeugung und Konfiguration eines eigenen VM-Abbilds
    - Installation des Grundsystems
    - Hinzufügen von Java, Java-Bibliotheken für Dienst
    - **OpenStack-Rechnerübung: Fr., 15.11., 12:00–14:00 Uhr (s. t.) in Raum 00.153-113**
  - Hinterlegen des JAR-Archivs des Dienstes auf S3



## Testen der Dienstlauffähigkeit

- Direkter Zugriff über HTTP-Anfrage (hier: GET-Anfrage)

```
> curl http://<ip-address>:<port>/tweetservice
```

- Direkter Zugriff über den Web-Browser

| JSON                    | Raw Data | Headers |
|-------------------------|----------|---------|
| Save Copy               |          |         |
| avgProcessTimePerTweet: | 0        |         |
| avgTweetLength:         | 0        |         |
| avgWordCount:           | 0        |         |
| langCounts:             | {}       |         |
| processedTweets:        | 0        |         |
| topHashtags:            | {}       |         |
| topReplies:             | {}       |         |
| topTweeter:             | {}       |         |
| trendingWords:          | {}       |         |
| tweetsPerSecond:        | 0        |         |

- Instanz nicht erreichbar
  - Einstellungen der Security Group kontrollieren

- Logs per SSH einsehen

```
> ssh -i <private_key (e.g., gruppe0.pem)> ec2-user@<ip_address>
```

- IP-Adresse z. B. über `aws ec2 describe-instances` ermitteln
- Überprüfen, ob Java-Prozess läuft: `> ps aux | grep java`
- Fehlersuche: Durchsuchen von `/var/log/syslog` oder `> journalctl -u i4mw-service`



## Zugriff auf OpenStack

- Web-Frontend
    - Dashboard: <https://i4cloud.cs.fau.de>
    - Zugangsdaten: siehe E-Mail mit Zugangsdaten
  - Kommandozeile
    - OpenStack-Client-Programm: `openstack`
    - **Vor Verwendung:** `openrc`-Datei sourcen (siehe unten)
  - Alle Kommandozeilenbefehle benötigen vorherige Authentifizierung
    - 1) Download der RC-Datei (`<user>-openrc.sh`) über Dashboard:
      - „Projekt“ → „API Access“
      - „Download OpenStack RC File“
    - 2) RC-Datei einlesen und ausführen (sourcen)
- ```
$ source /path/to/<user>-openrc.sh
```
- Benutzerdaten für Login per OpenStack-Konsole auf einer laufenden Instanz vom bereitgestellten Beispielabbild (`debian-example`):
USER: `ccloud` PASSWORD: `ccloud`



OpenStack4j

- OpenStack4j: Java-API für OpenStack-Dienste

- Bibliotheken: `/proj/i4mw/pub/aufgabe2/openstack4j`
- Dokumentation: <http://www.openstack4j.com/learn>

- Authentifizierung

```
// Package: org.openstack4j.model.common
Identifier userDomainName = Identifier.byName(<userDomainName>);
Identifier projectIdentifier = Identifier.byId(<projectIdentifier>);
OSClientV3 osc = OSFactory.builderV3() // Packages:
    .endpoint(<os_auth_url_from_rc_file>) // org.openstack4j.{api,openstack}
    .credentials(<user>, <pass>, userDomainName)
    .scopeToProject(projectIdentifier)
    .authenticate();
```

- Endpunkt-Adresse: Variable `OS_AUTH_URL` (in RC-Datei)
- Benutzername (`<user>`) und Passwort (`<pass>`):
siehe E-Mail zur Gruppeneinteilung
- Benutzer-Domänen-Name (`<userDomainName>`, String) → RC-Datei
- Projekt-ID (`<projectIdentifier>`, String) → RC-Datei/Web-Oberfläche
- `OSClientV3` ist an Thread gebunden → Neuen Client für anderen Thread per `OSFactory.clientFromToken(osc.getToken())` erzeugen



OpenStack4j: VMs erstellen

- Konfiguration (ähnlich zu AWS-API) über ServerCreate-Objekt

```
ServerCreate sc = Builders.server() // org.openstack4j.model.compute.api
    .<config_option1>
    .<config_option2>[...].<config_optionN>.build();
```

→ Konfigurieren von Instanzname, Instanztyp (Flavor-**ID**), Abbild-**ID**, Keypair, Netzwerk-**ID**, Security-Group, UserData (Kodierung mittels `java.util.Base64`)

- Boot mit Konfiguration (Aufruf blockiert, bis VM aktiv ist)

```
Server server = osc.compute().servers()
    .bootAndWaitActive(sc, <max_wait_time_in_ms>);
```

- Statusabfrage

`org.openstack4j.model.compute.Server.Status`

```
String serverId = server.getId();
Status st = osc.compute().servers().get(serverId).getStatus();
```



OpenStack4j: Floating-IP zuweisen und abfragen

- Floating-IP zuweisen

`org.openstack4j.model.compute.common`

```
List<? extends FloatingIP> ips = osc.compute().floatingIps().list();
FloatingIP floatingIp = ips.get(0);
// [...] unbenutzte Ip mit (floatingIp.getInstanceId() == null) suchen
ActionResponse r = osc.compute().floatingIps().addFloatingIP(server,
    floatingIp.getFloatingIpAddress());
```

- Floating-IP abfragen

`org.openstack4j.model.compute.common`

```
String publicIp = "";
List<? extends Address> vmAddresses = server.getAddresses()
    .getAddresses("internal");

for (Address address: vmAddresses) {
    if (address.getType().equals("floating")
        && address.getVersion() == 4) {
        publicIp = address.getAddr();
        break;
    }
}
```



Zugriff auf Metriken in OpenStack mittels Gnocchi

- REST-Anfragen → Zugriff über WebTarget-Objekt
- Dokumentation: <https://i4mw.cs.fau.de/openstack/gnocchi/rest.html>
- Gnocchi-Endpoint (Gnocchi-URL) über Kommandozeile ermitteln

```
> openstack catalog list
```

→ Suche nach „gnocchi“ und „i4region“

- HTTP-Header (Schlüssel-Wert-Paare) muss für alle Gnocchi-Anfragen zur Authentifizierung gesetzt werden

- Schlüssel (<key>): „X-Auth-Token“
- Wert (<value>): Zunächst Token anfordern

```
String authToken = osc.getToken().getId();
```

- Prinzip der Header-Modifizierung bei REST-Anfragen

```
Response r = target.request()
    .header(<key>, <value>).post(Entity.text("test"));
```



Zugriff auf Metriken in OpenStack mittels Gnocchi

- Instanz-gebundene ID einer Metrik (z. B. `cpu_util`) abfragen

→ GET-Anfrage auf Pfad listet alle Metriken auf:

```
<Gnocchi-URL>/v1/resource/instance/<vm-id>
```

- Ermittlung der ID einer Metrik (im Folgenden: <measure-id>)
- Rückgabe der Ergebnisse erfolgt im JSON-Format
- Datentyp: `MWGnocchiInstanceResource.class`

- Messwerte für eine bestimmte Metrik abfragen

→ GET-Anfrage auf Pfad:

```
<Gnocchi-URL>/v1/metric/<measure-id>/measures?start=<time>&granularity=1
```

- <time>: Zeitstempel (analog zu CloudWatch)
oder relative Zeitangabe, z. B. „-30seconds“
- granularity=1: Aggregationszeitraum über jeweils 1 Sekunde
- Datentyp: `String[] [] .class`

- Ceilometer-Polling-Frequenz nach neuen Daten: 10 Sekunden

→ es wird nicht jede Sekunde ein neuer Datenpunkt erzeugt



Timer und Timer-Task

■ Timer-Klasse `java.util.Timer`

- Einfache Scheduler-Funktionalität für `TimerTask`-Objekte
- Zentrale Methoden

```
void schedule(TimerTask task, long delay);  
void scheduleAtFixedRate(TimerTask t, long dy, long period);  
void cancel();
```

- `schedule()` Einmalig auszuführenden Task aufsetzen
- `scheduleAtFixedRate()` Periodischen Task aufsetzen
- `cancel()` Timer beenden

■ Timeout-Handler-Klasse `java.util.TimerTask`

- Basisklasse für von Timer eingeplante Tasks
- Zentrale Methoden

```
abstract void run();  
boolean cancel();
```

- `run()` Task ausführen → Timeout behandeln
- `cancel()` Task bzw. Timeout abbrechen

