Middleware – Cloud Computing – Übung

Christopher Eibel, Michael Eischer, Tobias Distler

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)

www4.cs.fau.de

Wintersemester 2017/18

Überblick

Hybrid Clouds & Virtualisierung
- Einführung
- Aufbau einer virtuellen Maschine

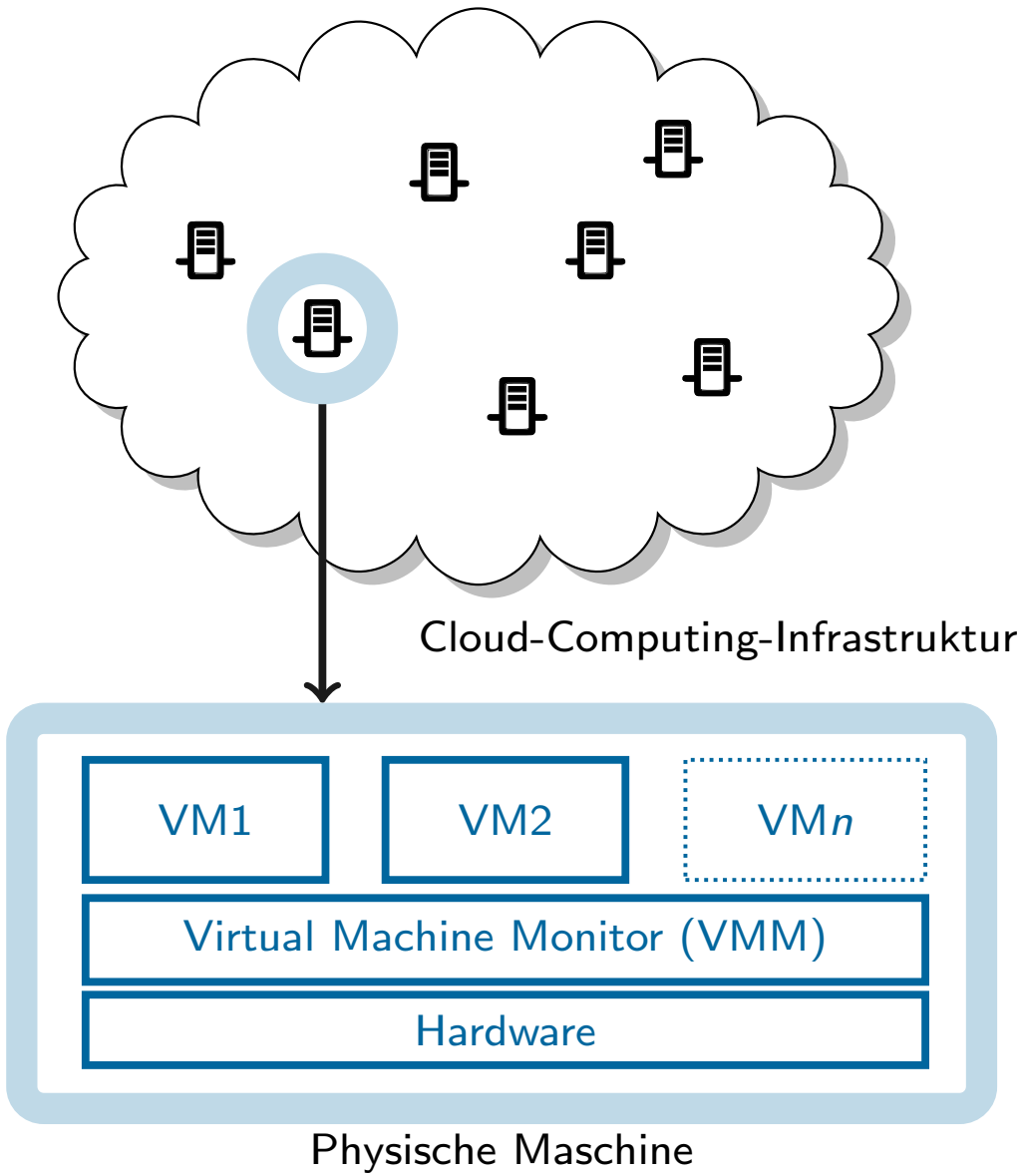
Amazon Web Services
- Überblick
- Elastic Compute Cloud (EC2)
- Simple Storage Service (S3)
- Amazon CloudWatch
- Amazon Java SDK

Aufgabe 2
- Aufgabenstellung
- OpenStack

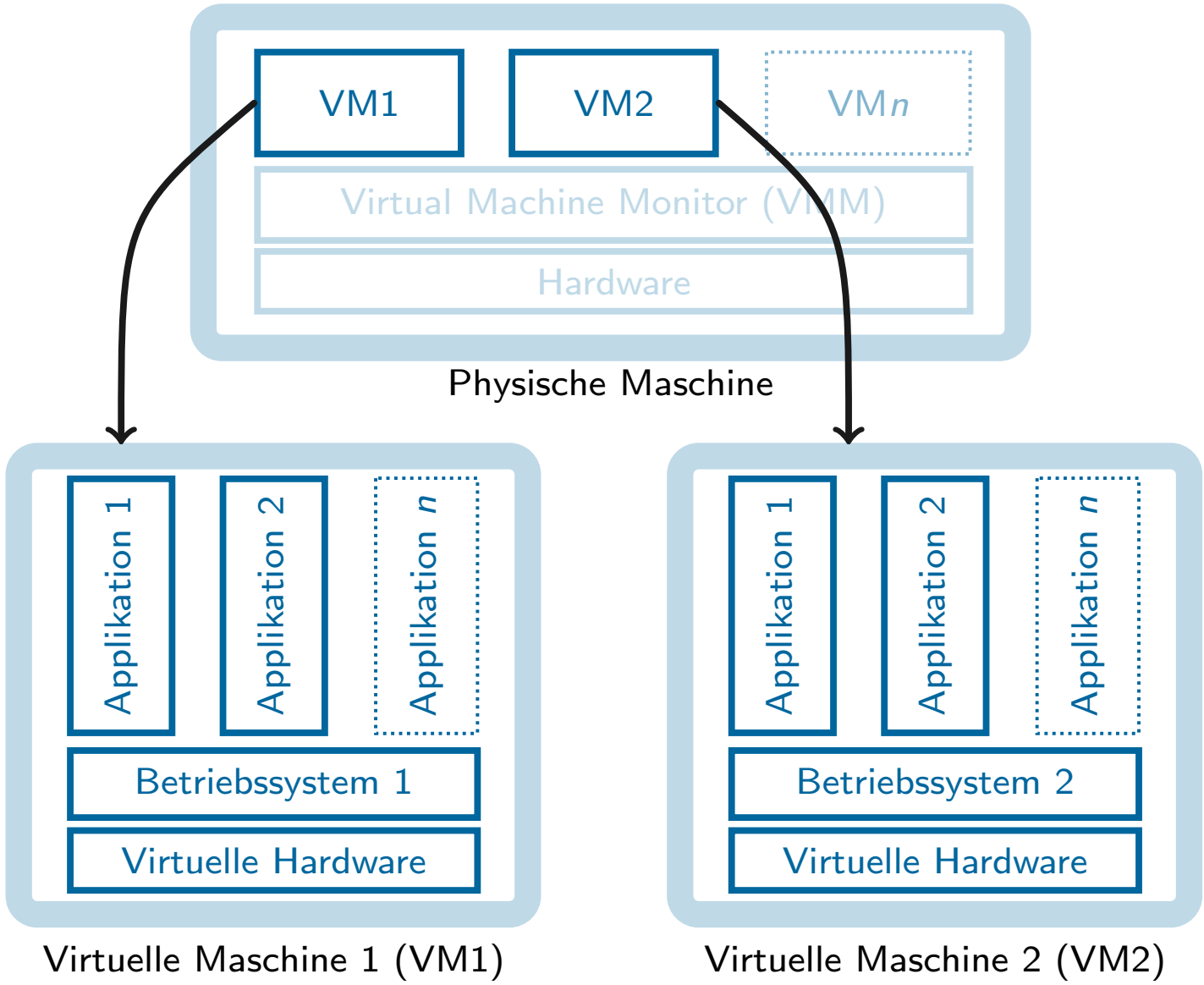
Hybrid Clouds

- 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, Ausdehung 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

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://awstcocalculator.com/`

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

Amazon Web Services (AWS)

Database

DynamoDB
Predictable and Scalable NoSQL Data Store

ElastiCache
In-Memory Cache

RDS
Managed Relational Database

Redshift
Managed Petabyte-Scale Data Warehouse

Storage & CDN

S3
Scalable Storage in the Cloud

EBS
Networked Attached Block Device

CloudFront
Global Content Delivery Network

Glacier
Archive Storage in the Cloud

Storage Gateway
Integrates On-Premises IT with Cloud Storage

Import Export
Ship Large Datasets

Cross-Service

Support
Phone & email fast-response 24X7 Support

Marketplace
Buy and sell Software and Apps

Management Console
UI to manage AWS services

SDKs, IDE kits and CLIs
Develop , integrate and manage services

Analytics

Elastic MapReduce
Managed Hadoop Framework

Kinesis
Real-Time Data Stream Processing

Data Pipeline
Orchestration for Data-Driven Workflows

Compute & Networking

EC2
Virtual Servers in the Cloud

VPC
Virtual Secure Network

ELB
Load balancing Service

WorkSpaces
Virtual Desktops in the cloud

Auto Scaling
Automatically scale up and down

DirectConnect
Dedicated Network Connection to AWS

Route 53
Scalable Domain Name System

Deployment & Management

CloudFormation
Templated AWS Resource Creation

CloudWatch
Resource and Application Monitoring

Elastic Beanstalk
AWS Application Container

IAM
Secure AWS Access Control

CloudTrail
User Activity Logging

OpsWorks
DevOps Application Management Service

CloudHSM
Hardware-based key storage for compliance

App Services

CloudSearch
Managed Search Service

Elastic Transcoder
Easy-to-use Scalable Media Transcoding

SES
Email Sending Service

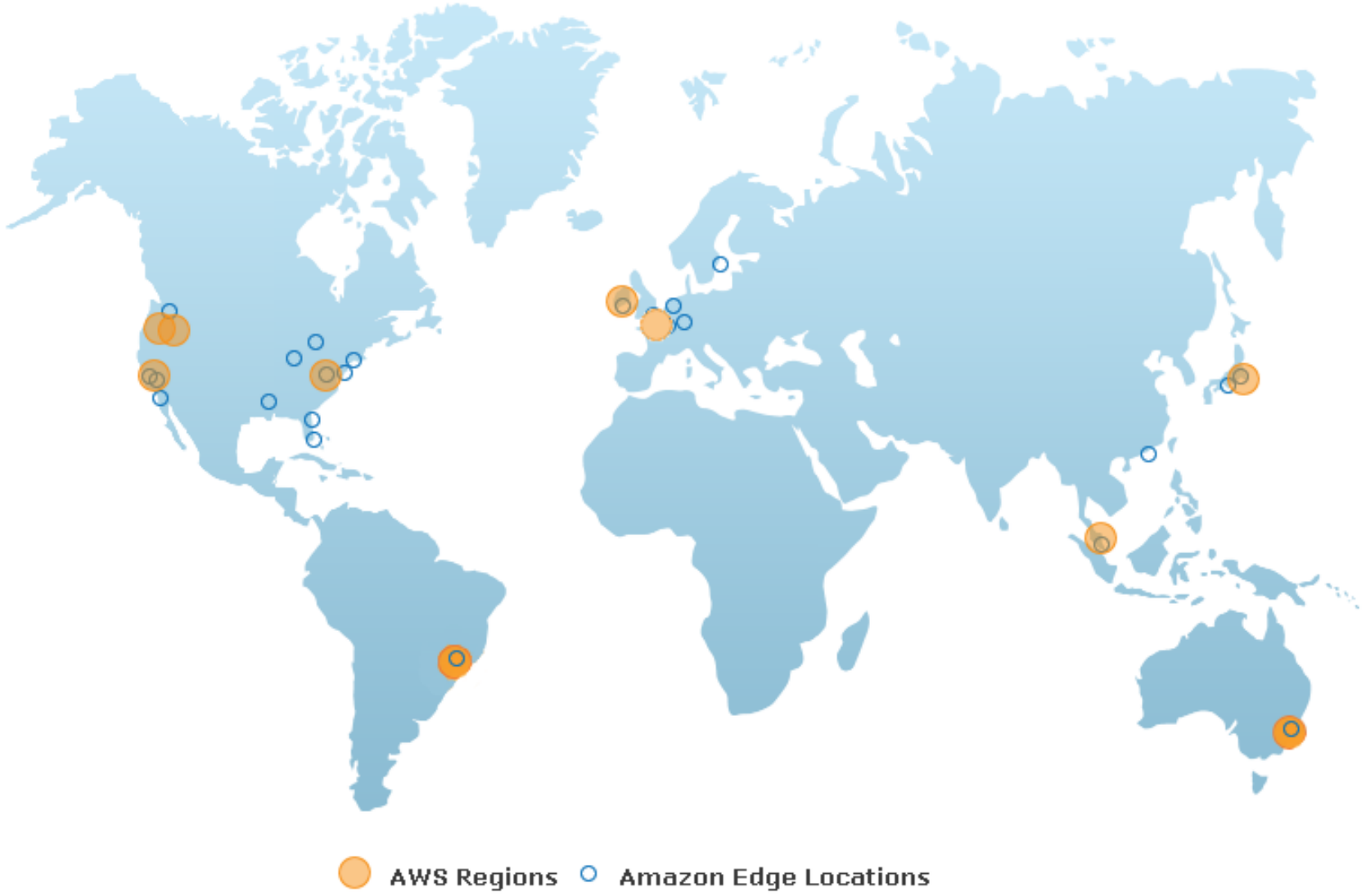
SNS
Push Notification Service

SQS
Message Queue Service

SWF
Workflow Service for Coordinating App Components

AppStream
Low-latency Application Streaming

Amazon Web Services (AWS)



Amazon Web Services: Betriebsumgebung

- Benutzung der Amazon Web Services (u. a.) über Web-Oberfläche (Login-Informationen werden in einer separaten E-Mail mitgeteilt)
  → `https://console.aws.amazon.com`
- Ablegen der Credentials zum API-Zugriff in Datei empfehlenswert für spätere Übergabe an Programme, welche auf die API zugreifen

1) Anlegen der privaten Konfigurationsdatei `~/.aws/aws.conf`

```
> mkdir ~/.aws
> touch ~/.aws/aws.conf
```

2) Zugriffsrechte einschränken

```
> chmod 600 ~/.aws/*
```

3) Erstellen von `AWS_ACCESS_KEY` und `AWS_SECRET_ACCESS_KEY` über die Web-Oberfläche: → `http://tinyurl.com/access-keys`, Reiter „Access Keys“
   → Eintragen in `aws.conf`:

```
export AWS_ACCESS_KEY_ID=<schluessel_id>
export AWS_SECRET_ACCESS_KEY=<privater_schluessel>
```

Amazon Web Services: Betriebsumgebung

- Nach dem Erstellen von `~/.aws/aws.conf` kann die Konfigurationsdatei geladen werden

```
> source ~/.aws/aws.conf
```

- AWS CLI: AWS-Befehlszeilen-Schnittstelle

```
export PATH="$PATH:/proj/i4mw/pub/aufgabe2/aws/bin"
```

  - 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: `> aws configure`
    - Setzen der Credentials (`AWS_ACCESS_KEY` und `AWS_SECRET_ACCESS_KEY`)
    - Setzen der Region (`eu-west-1`)

- Liste der verfügbaren AWS-Kommandozeilen-Tools:

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

Amazon Elastic Compute Cloud (EC2)

- Voraussetzungen für die Instanziierung einer virtuellen Maschine
  - Amazon Machine Image (AMI, Liste: `aws ec2 describe-images`)
  - EC2-Schlüsselpaar
- 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

- 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
```

- Starten einer Linux-Instanz:
  - AMI: `ami-acd005d5` [↪ Amazon Linux AMI], Instanz-Typ: `t2.nano`
  - Schlüsselname: `gruppe0`, Nutzdatenfeld mit String füllen: `Hello World.`
  - Security-Group legt Port-Freigaben fest; Basis-Security-Group (u. a. mit Port-Freigabe für SSH) bereits im AWS-Account vorhanden (Name: `i4mw`)
    (→ in Web-Oberfläche konfigurierbar: `http://tinyurl.com/aws-sgroups`)
  - VPC inkl. Subnet-Id nötig; existiert bereits im zur Verfügung gestellten AWS-Account; Finden der IDs z. B. auch über `aws ec2 describe-subnets`
    (→ in Web-Oberfläche konfigurierbar (optional): `http://tinyurl.com/aws-vpcs`)

```
> aws ec2 run-instances --instance-type t2.nano --key gruppe0
        --user-data="Hello world." --image-id ami-acd005d5
        --subnet-id subnet-0eab946a --security-group-ids sg-1dec2066
```

Amazon EC2: Zugriff auf eine Instanz

- Überprüfen des Status der Instanz mit `aws ec2 describe-instances`; auch: Abfrage der öffentlichen IP-Adresse ($\rightarrow$ `PublicIpAddress`)
- Sobald die Instanz den Boot-Vorgang abgeschlossen hat, erfolgt der Zugriff auf die Maschine mittels SSH

```
> ssh -i ~/.aws/gruppe0.pem -l 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 Log-Dateien (`/var/log/*`)
- 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-xxxxxxxx`)
- Unter Kenntnis dieser ID kann die Instanz mit `aws ec2 terminate-instances` beendet werden:

```
> aws ec2 describe-instances
(...)
> aws ec2 terminate-instances i-xxxxxxxx
```

- Kontrolle: `https://console.aws.amazon.com/ec2/home`
- Bitte stets sicherstellen,
  dass **keine unbenutzten** Instanzen laufen!

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 (Bilddaten)

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`)
  - Firefox-Plugin (z. B. S3Fox, `https://addons.mozilla.org/de/firefox/addon/s3fox/`)
  - 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:

  `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-1.11.210/lib/aws-java-sdk-1.11.210.jar`

  3rd-Party-Bibliotheken:

  `/proj/i4mw/pub/aufgabe2/aws/aws-java-sdk-1.11.210/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. Instanziierungs-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 die Umgebungsvariablen `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`

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

List<Tag> tags = new LinkedList<Tag>();
tags.add(tag);

spec.setTags(tags);

spec.setResourceType("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();
request.withImageId("ami-9ca607e5")
  .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.util.GregorianCalendar, java.util.TimeZone
GregorianCalendar e = new GregorianCalendar(TimeZone.getTimeZone("UTC"));
GregorianCalendar s = new GregorianCalendar(TimeZone.getTimeZone("UTC"));
s.add(GregorianCalendar.MINUTE, -2); // Datenpunkte ueber 2-Min.-Intervall

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

- Weiterführende Links
  - `http://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_concepts.html`
  - `http://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_GetMetricStatistics.html`

Amazon Java SDK: CloudWatch

Metrik abrufen (Fortsetzung)

com.amazonaws.services.cloudwatch.model

```
// Festlegung, dass nur Durchschnittswerte abgefragt werden
List<String> stats = new LinkedList<String>();
stats.add("Average");

// Request zum Holen der Werte einer Metrik zusammensetzen und absenden
GetMetricStatisticsRequest req = new GetMetricStatisticsRequest()
  .withStatistics(stats)
  .withPeriod(60)
  .withStartTime(s.getTime())
  .withEndTime(e.getTime())
  .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());
}
```

Amazon Java SDK: CloudWatch

Eigene Metrik propagieren

com.amazonaws.services.cloudwatch

```
// Dimension festlegen
Dimension dimension = new Dimension()
  .withName("PerInstanceMetrics")
  .withValue("i-xxxxxxx");

// Konkreten Wert fuer eigene Metrik setzen
MetricDatum datum = new MetricDatum()
  .withMetricName("TotalRequests")
  .withUnit(StandardUnit.None)
  .withValue(42.0)
  .withStorageResolution(1)   // Aktivieren einer hochaufloesenden Metrik
  .withDimensions(dimension);

// Metrik propagieren
PutMetricDataRequest request = new PutMetricDataRequest()
    .withNamespace("MW/TEST")
    .withMetricData(datum);

PutMetricDataResult response = cw.putMetricData(request);
```

→ Auch eigene Metriken im Dashboard {anzeig,visualisier}bar

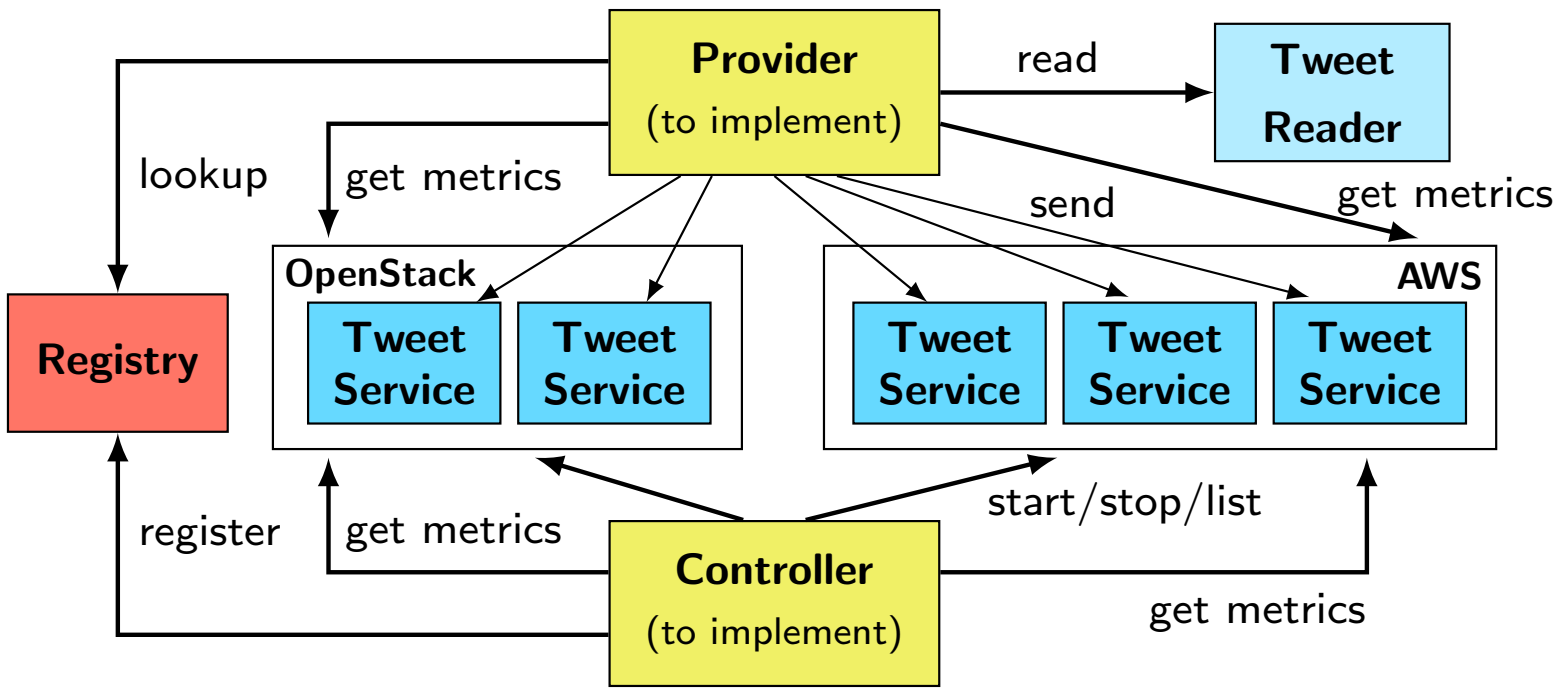
Überblick

Aufgabe 2

- Web-Service in der hybriden Cloud-Computing-Umgebung ausführen
  - Cloud-Ansteuerung
  - Basis-Technologien
    - Amazon EC2 vs. OpenStack Nova
    - Amazon CloudWatch vs. OpenStack Ceilometer/Gnocchi
    - Amazon EBS vs. OpenStack Glance/Cinder
    - Amazon S3 (vs. OpenStack Swift)
    - Lastverteilung
    - Dynamische Skalierung
- Amazon Web Services
  - Rund 20 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, stoppen, auflisten, nach Metriken abfragen)
  - Lastabhängige 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-9ca607e5) enthalten
    - Installierte Java-Version im AMI: OpenJDK Runtime Environment 8
  - Hinterlegen des JAR-Archivs des Diensts 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., 10.11., 14:00–16:00 + X Uhr **(s. t.)**
  - Hinterlegen des JAR-Archivs des Diensts auf S3

Zugriff auf OpenStack

- Web-Frontend
  - Dashboard: `http://i4cloud.informatik.uni-erlangen.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“ → „Compute“ → „API Access“
   → „Download OpenStack RC File v3“
2) RC-Datei einlesen und ausführen (sourcen)

```
$ source /path/to/<user>-openrc.sh
```

- Benutzerdaten für Login auf einer laufenden Instanz vom bereitgestellten Beispielabbild (`debian-example`): USER: `cloud` PW: `cloud`

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 projectIdentifier = Identifier.byId(<projectIdentifier>);
Identifier domainName = Identifier.byName(<domainIdentifier>);

OSClientV3 osc = OSFactory.builderV3() // Packages:
  .endpoint(<address_from_rc_file>)    // org.openstack4j.{api,openstack}
  .credentials(<user>, <pass>, domainName)
  .scopeToProject(projectIdentifier)
  .authenticate();
```

  - Endpunkt-Adresse: Variable `OS_AUTH_URL` (in RC-Datei)
  - Benutzername (`<user>`) und Passwort (`<pass>`): siehe E-Mail zur Gruppeneinteilung
  - Projekt-ID (`<projectIdentifier>`, `String`) → RC-Datei/Web-Oberfläche
  - Domänen-Name (`<domainName>`, `String`) → RC-Datei/Web-Oberfläche

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();
```

- Floating-IP zuweisen `org.openstack4j.model.{compute,network,common}`

```
List<? extends FloatingIP> ips = osc.compute().floatingIps().list();
FloatingIP floatingIp = ips.get(0);
NetFloatingIP nfIp = osc.networking().floatingip()
                                     .get(floatingIp.getId());
ActionResponse r = osc.compute().floatingIps().addFloatingIP(server,
                                       nfIp.getFloatingIpAddress());
```

Zugriff auf Metriken in OpenStack mittels Gnocchi

- REST-Anfragen → Zugriff über `WebTarget`-Objekt
- Dokumentation: `http://gnocchi.xyz/rest.html`
- Gnocchi-Endpunkt (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-Modifizerung 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 Ressource (z. B. `cpu_util`) abfragen
  → `GET`-Anfrage auf Pfad listet alle Ressourcen auf:
  `<Gnocchi-URL>/v1/resource/instance/<vm-id>`
  → Ermittlung der ID einer Ressource (im Folgenden: `<res_id>`)
- Wert einer Metrik für eine bestimmte Ressource abfragen
  → `GET`-Anfrage auf Pfad:
  `<Gnocchi-URL>/v1/metric/<res-id>/measures?start=<time>&granularity=1`
  - `<time>`: Zeitstempel (analog zu CloudWatch)
    oder relative Zeitangabe, z. B. `"-30seconds"`
  - `granularity=1`: Aggregationszeitraum über jeweils 1 Sekunde
- Ceilometer-Polling-Frequenz nach neuen Daten: 10 Sekunden
  → es wird nicht jede Sekunde ein neuer Datenpunkt erzeugt
- Rückgabe der Ergebnisse erfolgt jeweils im JSON-Format
  - Mögliche Klassen zum Parsen:
    `javax.json.Json, javax.json.stream.{JsonParser,JsonParserFactory}`
  - Dokumentation:
    `https://docs.oracle.com/javaee/7/api/javax/json/stream/JsonParser.html`

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