Middleware – Cloud Computing – Übung

Aufgabe 2: Hybrid Cloud

Wintersemester 2020/21

Michael Eischer, Laura Lawniczak, Tobias Distler

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.cs.fau.de

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Überblick

Hybride Cloud

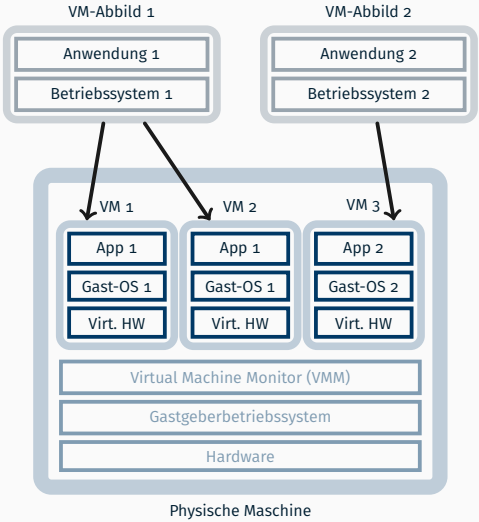
Hybride Cloud & Virtualisierung

Hybride Cloud

- Öffentliche Cloud: Cloud-Dienste frei für jeden verfügbar
  - *aaS: „X as a Service“-Gedanke
  - Scheinbar unbegrenzte Ressourcen
- 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
- **Hybride Cloud:** Mischform aus privater und öffentlicher Cloud
  - Sicherheitskritische Teile einer Anwendung laufen nur in der privaten Cloud
  - Skalierbarkeit, Ausdehnung auf öffentliche Cloud (z. B. beim Auftreten von Lastspitzen)

Virtuelle Maschinen in der Cloud

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

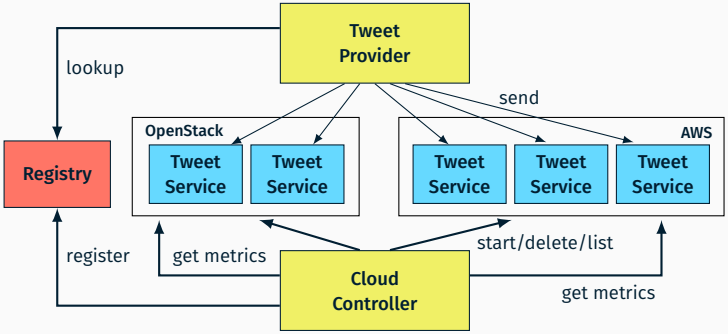


Hybride Cloud

Aufgabe 2

Aufgabe 2: Hybride Cloud

- Bereitgestellten Tweet-Service in hybrider Cloud ausführen
  - Ein bis maximal zwei VMs in privater Cloud
  - Öffentliche Cloud für Lastspitzen
- Teilaufgaben
  - Cloud-Controller für manuelle **Cloud-Ansteuerung** (VMs starten, beenden, auflisten)
  - **Lastverteilung** für Tweet-Anfragen im Provider, VMs per Registry abfragen
  - Erweiterter Cloud-Controller zur **dynamischen Skalierung** der VMs (nur 7.5 ECTS)



Aufgabe 2: Hybride Cloud

- Public Cloud: Amazon Web Services
  - Limitiertes Guthaben: 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/`
- Private Cloud: OpenStack-Umgebung des Lehrstuhls
  - Ressourcen der drei Node-Controller sind **beschränkt**
  - Jederzeit auf faire Verwendung achten
- OpenStack-Infrastruktur
  - Bitte sendet bei Problemen oder Ungereimtheiten schnellstmöglichst eine E-Mail an `i4mw-owner@lists.cs.fau.de`

Achtung!

Bitte stets sicherstellen,
dass **alle unbenutzten** Instanzen beendet (gelöscht) werden!

Aufgabe 2: Hybride Cloud

- Gemeinsame Schnittstelle: `MWCloudPlatform`
  - Instanzen starten / beenden / auflisten
  - Metriken der Instanzen abrufen

- `MWCloudPlatformAWS`: Betrieb des Dienstes in AWS EC2
  - Java 11 & Java-Bibliotheken bereits in vorkonfiguriertem Image (ami-0a6c83a43215dedd6) enthalten
  - Passende Konfigurationsparameter `userdata` übergeben
  - Metriken aus AWS CloudWatch abfragen
- `MWCloudPlatformOpenStack`: Betrieb des Dienstes in OpenStack Nova
  - Erzeugung und Konfiguration eines eigenen VM-Abbilds
    - Installation des Grundsystems
    - Hinzufügen von Java, Java-Bibliotheken für Dienst

    ↪ Schritt-für-Schritt Anleitung im Übungsteil „Erstellen eines VM-Abbilds in OpenStack“
  - Metriken aus Gnocchi abfragen

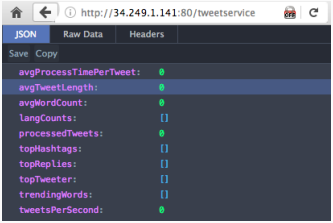
- Hinterlegen des JAR-Archivs des Tweet-Service auf AWS S3

Testen der Dienstlauffähigkeit

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

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

  → Innerhalb der VM unter `localhost` erreichbar



- Direkter Zugriff über den Web-Browser möglich
- Instanz nicht erreichbar?
  → Konfiguration der cloudseitigen Firewall durch die Security Groups kontrollieren
- Logs per SSH einsehen

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

  - Benutzername für AWS: `ec2-user`, für OpenStack: `cloud`
  - Bei Anmeldeproblemen Benutzernamen und SSH-Key kontrollieren
  - Überprüfen, ob Java-Prozess läuft: `> ps aux | grep java`
    oder `> sudo systemctl status i4mw-service`
  - Fehlersuche: Protokolle durchsuchen mit `> sudo less /var/log/syslog`
    oder `> sudo journalctl -u i4mw-service`

Amazon Web Services

Überblick

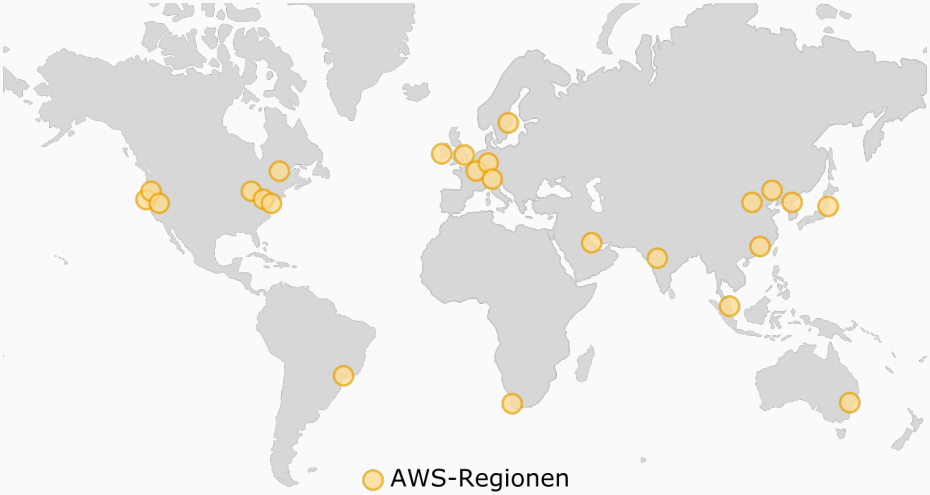
Amazon Web Services (AWS)

- 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
  - Elastic Block Storage (EBS) – Bereitstellung VM-Abbilder und Datenträger
  - Simple Storage Service (S3) – Netzwerkbasierter Speicher-Dienst
  - CloudWatch – Überwachungsfunktionen für AWS-Dienste

- Die Abrechnung erfolgt nach tatsächlichem Verbrauch **und** Standort
  - Betriebsstunden, Speicherbedarf
  - Transfervolumen, Anzahl verarbeiteter Anfragen
  - Standorte in Nord- und Südamerika, Europa, Südafrika und Asien-Pazifik: `https://infrastructure.aws/`
  - Berechnung der Gesamtbetriebskosten: `https://calculator.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/awscli/bin/aws
```

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

- Liste der verfügbaren AWS-Kommandozeilen-Tools

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

Amazon Web Services: Betriebsumgebung

- Ablegen der Credentials zum API-Zugriff in Datei `~/.aws/credentials`
  → Automatische Verwendung durch Programme, welche auf die API zugreifen
  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/`
  → Menü „Users“, Namen anklicken, Reiter „Security Credentials“, Abschnitt „Access Keys“
  → Eintragen in `~/.aws/credentials`

```
[default]
aws_access_key_id = <schluessel_id>
aws_secret_access_key = <privater_schluessel>
```

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

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

Amazon Web Services

Elastic Compute Cloud (EC2)

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`

- Optionales Nutzdatenfeld `user-data`
  - Base64-kodierter String
  - Maximal 16 kByte

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

```
> chmod 600 ~/.aws/gruppe0.pem
```

- VPC-Netzwerk inklusive Subnetz nötig
  → Konfiguration (optional): `https://console.aws.amazon.com/vpc/home?region=eu-west-1`
  - Existiert bereits im zur Verfügung gestellten AWS-Account
- Security-Group für Port-Freigaben einrichten
  → `https://console.aws.amazon.com/ec2/home?region=eu-west-1#SecurityGroups`
  - 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` und `::/0` (CIDR-Notation, entspricht weltweitem Zugriff) freigeben!
  - Änderungen möglich während Instanz läuft

Amazon EC2: Starten einer Instanz

- Starten einer Linux-Instanz
  - Instanz-Typ: `t2.nano`
  - AMI: `ami-0bb3fad3c0286ebd5` [↪ Amazon Linux 2 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 --filters Name=group-name,Values=i4mw \
        | grep -i -e groupname -e groupid
```

- Starten über die Kommandozeile

```
> aws ec2 run-instances --instance-type t2.nano \
         --image-id ami-0bb3fad3c0286ebd5 \
         --key gruppe0 --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`

↪ Antwort enthält auch öffentliche IP-Adresse (`PublicIpAddress`)

- Sobald der Boot-Vorgang abgeschlossen ist, erfolgt der Zugriff auf die Instanz 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>
```

- Bei Zugriffsproblemen: Boot-Meldungen über die Web-Schnittstelle oder mit `> aws ec2 get-console-output --instance-id <id> --output text` nach Fehlern durchsuchen

↪ Richtiger Benutzername für SSH verwendet?

- Innerhalb der virtuellen Maschine
  - Abrufen von Meta-Informationen mit `> ec2-metadata`
  - Enthalten Nutzdatenfeld `user-data`

Amazon EC2: Beenden einer Instanz

- Zum Terminieren einer im Betrieb befindlichen Instanz ist die eindeutige Instanz-ID notwendig
- Das Kommando `> aws ec2 describe-instances` listet die `InstanceId` (Format: `i-xxxxxxxx`)
- Unter Kenntnis dieser ID kann die Instanz beendet werden:

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

- Kontrolle: `https://console.aws.amazon.com/ec2/home`

Achtung!

Bitte stets sicherstellen,
dass **alle unbenutzten** Instanzen beendet (gelöscht) werden!

Amazon Web Services

Simple Storage Service (S3)

Amazon Simple Storage Service (S3)

- Der Simple Storage Service (S3) ist ein Netzwerk-Dateisystem
  - Einfache API
  - REST-Schnittstelle
  - Zugriffskontrolle mittels Zugriffskontrolllisten (Access Control Lists, ACLs)

- Eindeutige Identifikation von Dateien durch Bucket (Kübel) und Dateiname:
  `s3://<bucket>/<dateiname>`
- Kein hierarchischer Namensraum
  - Dateinamen mit Separator / möglich

  → Web-Konsole zeigt dies als Ordner an

- Übersetzung der S3-Adressrepräsentation in eine URL
  - S3: `s3://<bucket>/<dateiname>`
  - URL: `http://<bucket>.s3.amazonaws.com/<dateiname>`

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 Web Services

Amazon CloudWatch

Amazon CloudWatch

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

- Beispiele
  - Amazon EC2: CPU-Auslastung, gesendete/empfangene Netzwerkpakete
  - Amazon EBS: Lese- und Schreiblatenz

- Metriken: Messwerte über Zeit
  - Metriken abfragen aber auch eigene Metriken einpflegbar
  - Minutengranularität möglich
  - Ältere Daten werden aggregiert und ausgedünnt
- Alarme: Automatische Reaktion bei auffälligen Veränderungen
- Visualisierung: Darstellung der Daten in einem Dashboard möglich
  `https://eu-west-1.console.aws.amazon.com/cloudwatch` → „Metriken“

Amazon CloudWatch: Metriken

- Metriken
  - Enthalten Messwerte mit Zeitstempeln (UTC)
  - Gruppiert in *Namensräume* wie `AWS/EC2`, `AWS/EBS`, `AWS/S3`, …
  - *Dimensionen* zum Zuordnen von Datensätzen, z. B. per Instanz-ID

- Metriken für EC2 Instanzen
  - Grundlegende Überwachung (5 Minutenintervalle), kostenlos
  - Detaillierte Überwachung (1 Minutenintervalle), zusätzliche Kosten
  - Benutzerdefinierte Metriken: aus Anwendung heraus, selbst definierbar

- Abruf
  - Benötigt Start- und Endzeitpunkt sowie Aggregationszeitraum
  - Aggregation innerhalb eines Zeitraums (Period) per Minimum / Maximum / Durchschnitt / …
  - Zeitraum muss gleich oder ein Vielfaches des Erzeugungsintervall sein
  - Möglicherweise verzögert verfügbare Daten

Amazon Web Services

Amazon Java SDK

Amazon Java SDK

- Amazon stellt Java-Bibliotheken für die Verwendung der Amazon Web Services bereit
  `/proj/i4mw/pub/aufgabe2/aws-java-sdk-2.15.11`
  → Dokumentation: `https://sdk.amazonaws.com/java/api/latest/`

- Java-Packages für den Betrieb virtueller Maschinen in Amazon EC2 und Amazon CloudWatch
  - `software.amazon.awssdk.services.ec2`
  - `software.amazon.awssdk.services.cloudwatch`

- Grundlegende Verwendung des SDK
  1. Initial: Client-Objekt (z. B. Typ `Ec2Client`) erstellen und gegenüber AWS authentifizieren
  2. Anfrageparameter in Anfrageobjekt (z. B. Typ `RunInstancesRequest`) setzen

  ↪ Objekte nicht modifizierbar, Erzeugung per Builder-Pattern

  3. Anfrage über Client-Objekt abschicken
  4. Gibt Ergebnisobjekt (z. B. Typ `RunInstancesResponse`) zurück, das Ergebnis der Anfrage enthält

  ↪ Ergebnisobjekt spiegelt *Zustand zum Zeitpunkt der Antwort* wider

Amazon Java SDK: Instanziierung einer VM

- Minimal-Beispiel (analog Kommandozeilen-Beispiel)
  Beachte: Vor dem Aufruf am `Ec2Client.Builder` müssen in der Konfigurationsdatei `~/.aws/credentials` die Optionen `aws_access_key_id` und `aws_secret_access_key` gesetzt sein.

- Initialisierung `software.amazon.awssdk.services.ec2, software.amazon.awssdk.regions`

```
Ec2Client ec2 = Ec2Client.builder()
    .region(Region.EU_WEST_1)
    .build();
```

- Setzen des Namens einer VM-Instanz `software.amazon.awssdk.services.ec2.model`

```
Tag tag = Tag.builder().key("Name").value("MyVMName").build();

TagSpecification spec = TagSpecification.builder()
    .tags(tag)
    .resourceType("instance")
    .build();

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

Amazon Java SDK: Instanziierung einer VM

- Minimal-Beispiel (Fortsetzung) `software.amazon.awssdk.services.ec2.model`

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

```
RunInstancesRequest request = RunInstancesRequest.builder()
    .imageId("ami-0bb3fad3c0286ebd5")
    .tagSpecifications(spec)
    .instanceType("t2.nano")
    .minCount(1)
    .maxCount(1)
    .keyName("gruppe0-key")
    .userData(Base64.getEncoder().encodeToString(userDataBytes)) // java.util.Base64
    // optional, detailliertere Metriken aktivieren
    .monitoring(RunInstancesMonitoringEnabled.builder().enabled(true).build())
    .securityGroupIds("sg-989f5ce3") // z.B. im Web-Interface erstellen
    .subnetId("subnet-0eab946a")     // (VPC muss Security-Group vorab zugeordnet werden)
    .build();
RunInstancesResponse response = ec2.runInstances(request);
```

- Hinweise:
  - Mittels des Objektes `response` die Instanz-ID in Erfahrung bringen
  - Auf die eigentliche Instanziierung prüfen (`DescribeInstancesRequest`)
  - Zwischen zwei Abfragen des Instanzstatus kurz warten

Amazon Java SDK: CloudWatch

- Initialisierung (ähnlich wie bei EC2) `software.amazon.awssdk.services.cloudwatch`

```
CloudWatchClient cw = CloudWatchClient.builder()
    .region(Region.EU_WEST_1).build();
```

- Metrik abrufen: Zeitintervall und Dimension festlegen
  - Erwartetes Zeitformat: ISO 8601, UTC (z. B. 2020-11-25T09:00:00Z)
  - Beispielhaftes Definieren von Anfangs- und Endzeitpunkt

```
// Packages: java.time.Clock, java.time.Instant
Instant endTime = Clock.systemUTC().instant();
Instant startTime = endTime.minusSeconds(120); // Datenpunkte ueber 2-Min.-Intervall

// Package: software.amazon.awssdk.services.cloudwatch.model.Dimension
Dimension dimension = Dimension.builder()
    .name("InstanceId")
    .value("i-xxxxxxx")
    .build();
```

- Weiterführende Links
  - 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/AWSEC2/latest/UserGuide/viewing_metrics_with_cloudwatch.html

- Metrik abrufen (Fortsetzung)

`software.amazon.awssdk.services.cloudwatch.model`

```
// Request zum Holen der Werte einer Metrik zusammensetzen und absenden
// festlegen, dass nur Durchschnittswerte abgefragt werden
GetMetricStatisticsRequest req = GetMetricStatisticsRequest.builder()
    .statistics(Statistic.AVERAGE)
    .metricName("NetworkIn")
    .dimensions(dimension)
    .namespace("AWS/EC2")
    .period(60)
    .startTime(startTime)
    .endTime(endTime)
    .build();
GetMetricStatisticsResponse res = cw.getMetricStatistics(req);

// Zeitstempel und Durchschnittswerte ausgeben
for (Datapoint dp : res.datapoints()) {
    System.out.printf("%s: %s\n", dp.timestamp(), dp.average());
}
```

OpenStack

Zugriff auf OpenStack

- Web-Frontend
  - Dashboard: `https://i4cloud1.cs.fau.de`
  - Zugangsdaten: siehe Gruppeneinteilungs-E-Mail
- Kommandozeilen-Client
  - 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 des bereitgestellten Beispielabbilds (`debian-example`):
  USER: `cloud` PASSWORD: `cloud`

OpenStack4j

- OpenStack4j: Java-API für OpenStack-Dienste
  - Bibliotheken: `/proj/i4mw/pub/aufgabe2/openstack4j-3.3`
  - Dokumentation: `https://openstack4j.github.io/learn`
- Authentifizierung

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

  - Parameter in OpenStack RC-Datei
    - Benutzer-Domänen-Name (`<user_domain_name>`): Variable `OS_USER_DOMAIN_NAME`
    - Projekt-ID (`<project_identifier>`): Variable `OS_PROJECT_ID`
    - Endpunkt-Adresse (`<os_auth_url>`): Variable `OS_AUTH_URL`
  - Benutzername (`<user>`) und Passwort (`<pass>`): siehe E-Mail zur Gruppeneinteilung
  - `OSClientV3` ist an Thread gebunden → Neuen Client für anderen Thread per `OSFactory.clientFromToken(client.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`)
  - Ersteinrichtung: Siehe Übung zum „Erstellen eines VM-Abbilds für OpenStack“

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

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

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

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

OpenStack4j: Floating-IP zuweisen und abfragen

- VM hat initial nur interne IP

→ Zugriff von extern nur mit Floating-IP möglich

- Floating-IP zuweisen

`org.openstack4j.model.{compute,common}`

```
List<? extends FloatingIP> ips = client.compute().floatingIps().list();
FloatingIP floatingIp = ips.get(0);
// [...] unbenutzte IP mit (floatingIp.getInstanceId() == null) suchen
ActionResponse r = client.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

- Datenabruf per REST-Anfragen
  - Zugriff über `WebTarget`-Objekt
  - Dokumentation: `https://gnocchi.xyz/stable_4.2/rest.html`
- Gnocchi-Endpunkt-URL (Servicetyp „`Metric`“) im Dashboard unter „API Access“ nachschlagen
- *Oder* Ermitteln der Endpunkt-URL mittels der Dienstliste von OpenStack

```
List<? extends Service> catalog = client.identity().tokens().getServiceCatalog(client.getToken().getId())
```

→ Öffentlichen (`Public`) Endpunkt des Servicetyps „`Metric`“ verwenden

- Authentifizierung bei Gnocchi-Anfragen erfolgt per HTTP-Header (Schlüssel-Wert-Paare)
  - Für *alle* Anfragen notwendig
  - Schlüssel (`<key>`): „`X-Auth-Token`“
  - Wert (`<value>`): Token von OpenStack anfordern

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

  - Header-Modifikation bei REST-Anfragen

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

Zugriff auf Metriken in OpenStack mittels Gnocchi

- Instanz-spezifische ID einer Metrik (z. B. `cpu`) ermitteln (im Folgenden: `<metric-id>`)
  → `GET`-Anfrage auf Pfad listet alle Metriken auf:
  `<Gnocchi-URL>/v1/resource/instance/<vm-id>`
  - Rückgabe der Ergebnisse erfolgt im JSON-Format
  - Datentyp: `MWGnocchiInstanceResource`
- Messwerte für eine bestimmte Metrik abfragen
  → `GET`-Anfrage auf Pfad:
  `<Gnocchi-URL>/v1/metric/<metric-id>/measures?start=<time>&granularity=10&aggregation=mean`
  - „`<time>`“: Zeitstempel (analog zu CloudWatch) **oder** relative Zeitangabe, z. B. „`−30seconds`“
  - „`granularity=10`“: Jeweils über 10 Sekunden aggregierte Datenpunkte abrufen
    - OpenStack Ceilometer sammelt bei uns alle 10 Sekunden neue Daten
    - Mögliche Aggregationszeiträume: 10 / 60 / 3600 Sekunden
  - „`aggregation=mean`“: Durchschnitt über Aggregationszeitraum
  - Datentyp: `String[][]`; pro Array-Element: Zeitstempel, Aggregationszeitraum, Wert
- CPU-Metrik gibt akkumulierte Rechenzeit zurück
  - „`aggregation=rate:mean`“: CPU-Verbrauch des aktuellen Aggregationszeitraums in Nanosekunden
  - CPU-Auslastung: Messwert / Aggregationszeitraum
    beispielsweise $7\,000\,000\,000\,ns / 10\,000\,000\,000\,ns = 70\%$