ISEngineering

Technische Universität Berlin

Enterprise Computing: Exercise 2 – REST Server / S3 / SQS

Marco Peise

ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

Agenda

Exercise 1 Solutions

Exercise 2

- REST Server
- S3 client for AWS
- SQS client for AWS

Task 1 – REST Client

Technische Universität Berlin

Your Task is to write a HTTP client program in the programming language of your choice.

1. **Insert** an event name (corresponding to your name) and a date value via a HTTP **PUT** operation.

Endpoint: http://api-server.eu-gb.mybluemix.net/api/Calendars

Request Parameter:

{"name": <String>, // i.e.: “chrismas_marco_peise”
"date": <Date> // i.e.: “2016-12-24" }

2. **Use** the Service “timeToDate” for a given Event via a HTTP **GET**.

Endpoint: http://api-server.eu-gb.mybluemix.net/api/Calendars/timeToDate

Request Parameter:

{“eventName": <String>, // i.e.: “chrismas_marco_peise” }

Hand in your programmed lines of code for your client in ISIS.

Task 2 – HTTP Methods

Technische Universität Berlin

Review the API of Liquidfeedback (http://dev.liquidfeedback.org/trac/lf/wiki/API)

Discuss the following four HTTP Requests:

GET /issue, POST /voter, POST /interest, GET /suggestion

State for each Request an example call (endpoint, request attributes.

Please check in the following table whether the HTTP methods are idempotent and / or safe.

HTTP Methode	Idempotent	Safe	None
GET /issue			
POST /voter			
POST /interest			
GET /suggestion			

Task 2 – HTTP Methods

Technische Universität Berlin

GET /issue:

http://liquidfeedback.org/issue?issue_id=15

POST /voter:

http://liquidfeedback.org/voter?issue_id=15&initiative_23&comment='my comment'

POST /interest:

http://liquidfeedback.org/interest?issue_id=15&delete=true

GET /suggestion:

http://liquidfeedback.org/suggestion?suggestion_id=42&member_id=15

HTTP Methode	Idempotent	Safe	None
GET /issue	X	X	
POST /voter			X
POST /interest			X
GET /suggestion	X	X	

Task 3 – WebServices

Technische Universität Berlin

Given is a WebService with the following WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:s1="http://microsoft.com/wsdl/types"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://tu-berlin.de/ise/ec/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" targetNamespace="http://tu-berlin.de/ise/ec"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
<wsdl:types>
  <s:schema elementFromDefault="qualified" targetNamespace="http://tu-berlin.de/ise/ec">
    <s:import namespace="http://microsoft.com/wsdl/type/"/>
    <s:element name="GetPositionsOfSymbollnText">
      <s:complextType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="text" type="s:string"/>
          <s:element minOccurs="1" maxOccurs="1" name="symbol" type="s1:char"/>
        </sequence>
      </s:complexType>
    </s:element>
    <s:element name="GetPositionsOfSymbollnTextResponse">
      <s:complextType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
             name="GetPositionsOfSymbollnTextResult" type="tns:ArrayOfInt"/>
        </sequence>
      </s:complexType>
    </s:element>
    <s:complextType name="ArrayOfInt">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="int" type="s:int"/>
      </sequence>
    </s:complexType>
  </s:schema>
  <s:schema elementFromDefault="qualified"
    targetNamespace="http://microsoft.com/wsdl/types">
    <s:simpleType name="char">
      <s:restriction base="s:unsignedShort"/>
    </s:simpleType>
  </s:schema>
</wsdl:types>
<wsdl:message name="GetPositionsOfSymbollnTextSoapln">
  <wsdl:part name="parameters" element="tns:GetPositionsOfSymbollnText"/>
</wsdl:message>
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

Task 3 – WebServices

public int[] GetPositionsOfSymbolInText(String text, char symbol)

Request:

```
POST /AnwSysService/UebungsService.asmx HTTP/1.1
Host: ws-server.de
Content-Type: text/xml; charset=utf-8
Content-Length: 423
User-Agent: some great client
SOAPAction: "http://tu-berlin.de/ise/anwsys/GetPositionsOfSymbolInText"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
        <GetPositionsOfSymbolInText xmlns="http://tu-berlin.de/ise/anwsys/">
                <text>Anwendungssysteme macht Spass!</text>
                <symbol>a</symbol>
        </GetPositionsOfSymbolInText>
</soap:Body>
</soap:Envelope>
```

Task 3 – WebServices

Technische Universität Berlin

Response:

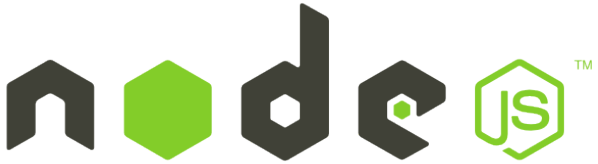
```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 519

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
        <GetPositionsOfSymbolInTextResponse xmlns="http://tu-berlin.de/ise/anwsys/">
                <GetPositionsOfSymbolInTextResult>
                        <int>19</int>
                        <int>26</int>
                </GetPositionsOfSymbolInTextResult>
        </GetPositionsOfSymbolInTextResponse>
</soap:Body>
</soap:Envelope>
```

Technische Universität Berlin

Exercise 2

Task 1 – REST Server

The task is to build and provide two Services which are described in Exercise 1 Task 1. Therefore, you may use a PaaS provider of your choice and build an RESTful Interface with an API Framework called StrongLoop in JavaScript (Node.js).



Task 1 – REST Server

Technische Universität Berlin

Service 1:

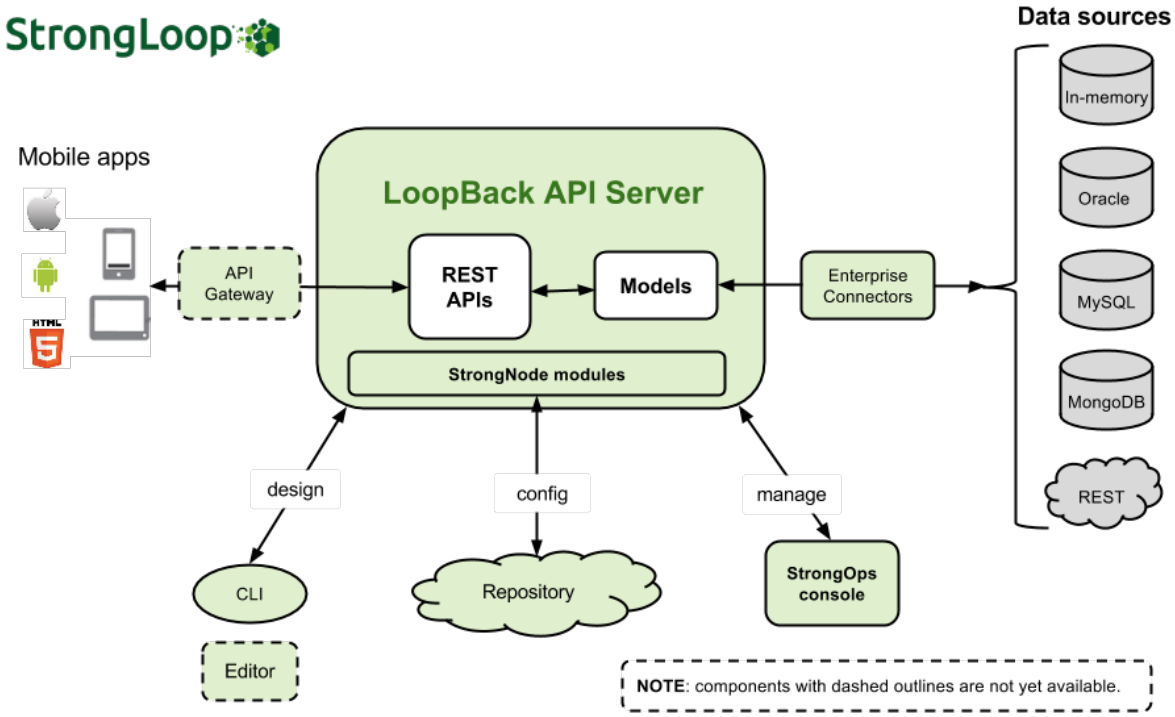
should be a HTTP PUT Operation and be able to store a given String and a Date by the client in JSON Format into a Database

Service 2:

should be a HTTP GET Operation (name: “timeToDate”) and should be able to search for a given String within the same Database and calculate the remaining time till that event and respond within a human friendly string (like “1 month 12 days 8 hours 42 minutes 40 seconds”)

example Request: eventString: “<someStringWhichExists”
example Response: “1 month 12 days 8 hours 42 minutes 40 seconds”

Task 3 – REST Server



http://loopback.io

Tasks 1 (Prerequisites)

Technische Universität Berlin

Prerequisites for Implementation with Node.js:

- Download or clone the git repository at
  https://github.com/marcopeise/EC-Exercise2_1
- Install node.js
  https://nodejs.org/en/download/
- Open the project in WebStorm / Eclipse or your favorite JavaScript Development Environment
- “npm install” to install referenced packages
- Run node.js application locally with node server/server.js
- Deploy your code to e.g.: IBM Bluemix
  https://www.eu-gb.bluemix.net/docs/starters/install_cli.html

Task 2 – REST Client

```
module.exports = function(Calendar) {
        Calendar.timeToDate = function(eventName, cb) {


    // TODO:
    // implement a search ("findOne") operation where the name matches the
    "eventName"
    // and the response should be human readable (for example with moment.js and the
    plugin preciseDiff)
    // Fault Handling keep in mind that the string could also not be found in the DB

        cb(null, "NOT YET IMPLEMENTED");
    };
```

Task 2 – REST Client

```
{
      "name": "Calendar",
      "base": "PersistedModel",
      ...
      "properties": {
        "id": {
          "type": "string",
          "id": true
        },
            // TODO....

      },
      ...
    }
```

Task 2 – Amazon S3

Use Amazon Simple Storage Service to:

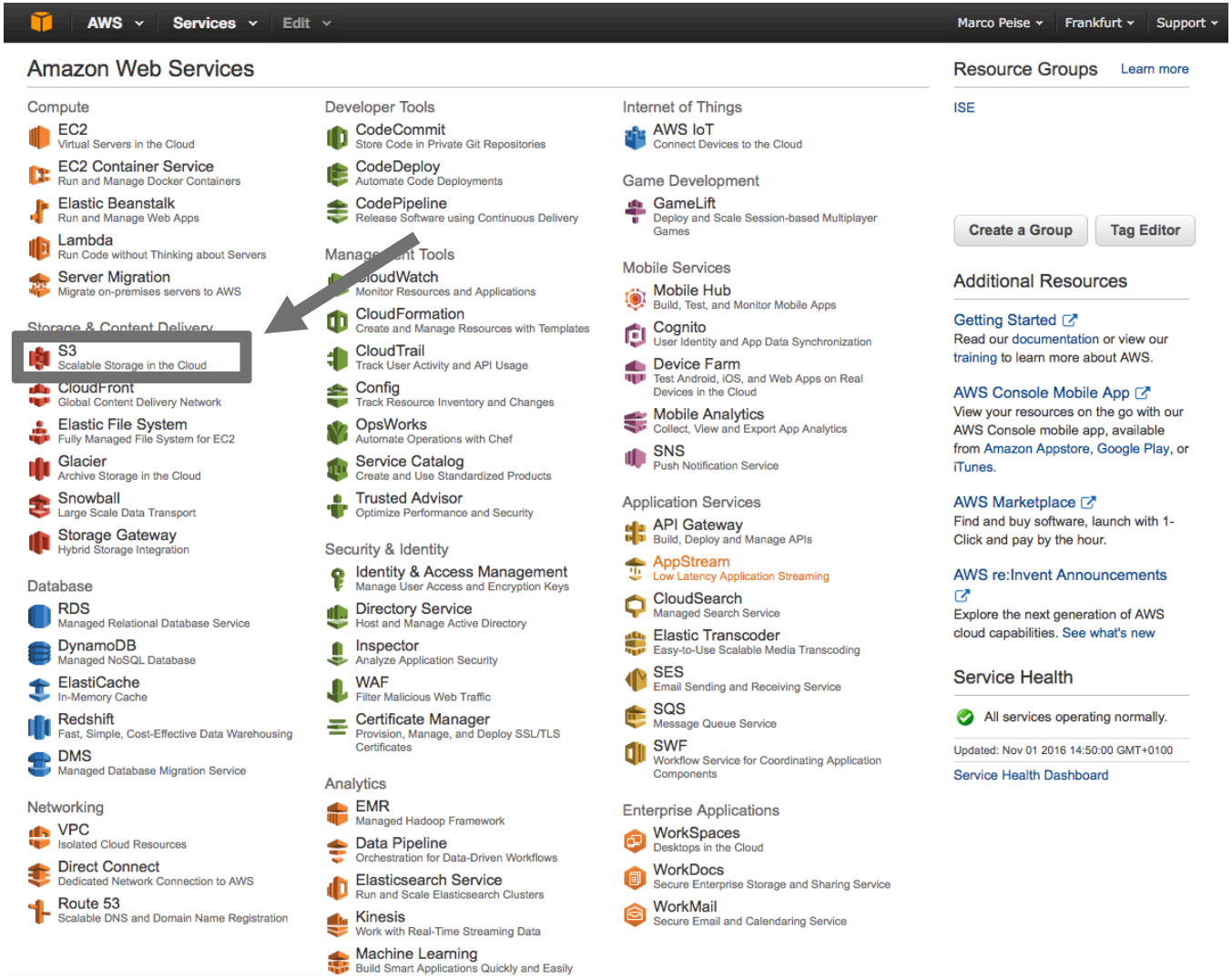
1) Create a bucket
2) Upload a text File into your S3 bucket
3) Download the File from your S3 bucket

with AWS SDK for Eclipse.

Technische Universität Berlin

AWS S3

AWS S3 - Amazon Simple Storage Service



AWS S3 - Amazon Simple Storage Service

- stores data as objects within resources called "buckets”
- as many objects as you want within a bucket
- operations: write, read, and delete objects in buckets
- size restriction: objects can be up to 5 terabytes in size
- Scalable
- 12 AWS regions (option: replication across regions)
- integration (security IAM, alerting CloudWatch, computing Lambda, database Redshift)
- Lifecycle management policies
  - move objects between storage classes via costumizable, automated rules

AWS S3 - Amazon Simple Storage Service

Amazon S3 Pricing

Pay only for what you use. There is no minimum fee. Estimate your monthly bill using the AWS Simple Monthly Calculator. We charge less where our costs are less, and prices are based on the location of your Amazon S3 bucket.

Get Started with AWS Today

Try Amazon S3 for Free

AWS Free Tier includes 5GB storage, 20,000 Get Requests, and 2,000 Put Requests with Amazon S3.

View AWS Free Tier Details »

AWS Free Usage Tier*

As part of the AWS Free Usage Tier, you can get started with Amazon S3 for free. Upon sign-up, new AWS customers receive 5 GB of Amazon S3 standard storage, 20,000 Get Requests, 2,000 Put Requests, and 15GB of data transfer out each month for one year.

Storage Pricing (varies by region)

Region: EU (Frankfurt)

	Standard Storage	Standard - Infrequent Access Storage †	Glacier Storage
First 1 TB / month	$0.0324 per GB	$0.018 per GB	$0.0120 per GB
Next 49 TB / month	$0.0319 per GB	$0.018 per GB	$0.0120 per GB
Next 450 TB / month	$0.0314 per GB	$0.018 per GB	$0.0120 per GB
Next 500 TB / month	$0.0308 per GB	$0.018 per GB	$0.0120 per GB
Next 4000 TB / month	$0.0303 per GB	$0.018 per GB	$0.0120 per GB
Over 5000 TB / month	$0.0297 per GB	$0.018 per GB	$0.0120 per GB

Except as otherwise noted, our prices are exclusive of applicable taxes and duties, including VAT and applicable sales tax. For customers with a Japanese billing address, use of AWS is subject to Japanese Consumption Tax. Learn more.

Tasks 2 + 3 (Prerequisites)

- Install the AWS plugin for Eclipse
  1. Open Help → Install New Software….
  2. Enter http://aws.amazon.com/eclipse in the text box labeled “Work with” at the top of the dialog.
  3. Select “AWS Toolkit for Eclipse” from the list below.
  4. Click “Next.” Eclipse guides you through the remaining installation steps.

Tasks 2 + 3 (Prerequisites)

- Set your AWS credentials
  - Mac/Linux: write into the file

~/.aws/credentials

  - Windows: write into the file

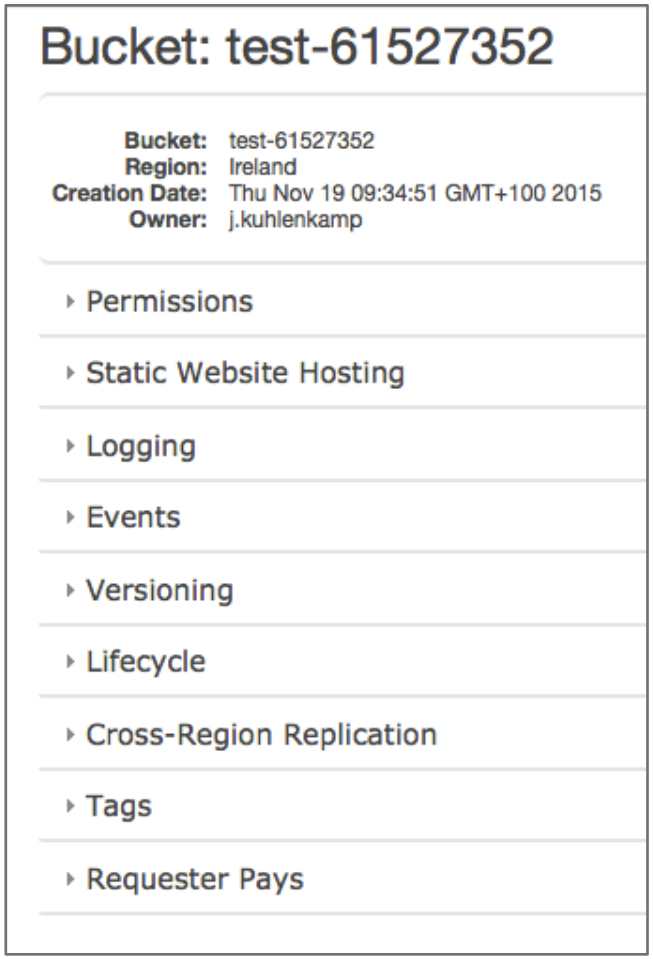
C:\Users\USERNAME\.aws\credentials

  - … the following content:

```
[default]
aws_access_key_id=enteryourkeyhere
aws_secret_access_key=enteryoursecrethere
```

(Replace the red text with your own information)

Task 2 – Amazon S3



Task 2 – Amazon S3

```
// TODO create a bucket with name "ise-tu-berlin-exercise2-",
// followed by your name or student id (MatrikelNr)
log.info("Creating a bucket (if it does not exist, yet)");
 …
// TODO Upload a text File object to your S3 bucket
// use the createSampleFile method to create the File object
log.info("Uploading an object");
 …
// TODO Download the file from S3 and print it out using the
// displayTextInputStream method.
log.info("Downloading an object");
…
```

Task 2 – Amazon S3

- Which HTTP method is used for the following AWS S3 operations?
  - createBucket
  - putObject
  - getObject
  - deleteObject

Hint: Launch the Java program with JVM option
"-Dlog4j.configuration=log4j.properties"
and log4j.properties setting "log4j.logger.org.apache.http=DEBUG"

Technische Universität Berlin

A LITTLE BACKGROUND INFO ABOUT MESSAGING BY EXAMPLE OF JMS

Point-to-Point

- The point-to-point messaging model allows JMS clients to send and receive messages both synchronously and asynchronously via virtual channels known as **queues**.
- In the point-to-point model, message producers are called **senders** and message consumers are called **receivers**.
- The point-to-point messaging model has traditionally been a pull-based or **polling-based model**, where messages are requested from the queue instead of being pushed to the client automatically.
- One of the distinguishing characteristics of point-to-point messaging is that messages sent to a queue are **received by one and only one receiver**, even though there may be many receivers listening on a queue for the same message.

Source: "Java Message Service", O'Reilly, 2nd Ed.

Publish-and-Subscribe

- In the publish-and-subscribe model, messages are published to a virtual channel called a **topic**.
- Message producers are called **publishers**, whereas message consumers are called **subscribers**.
- Unlike the point-to-point model, messages published to a topic using the publish-and-subscribe model can be received by multiple subscribers. This technique is sometimes referred to as **broadcasting** a message.
- Every subscriber receives a copy of each message. The publish-and-subscribe messaging model is by and large a **push-based model**, where messages are automatically broadcast to consumers without them having to request or poll the topic for new messages.

Source: "Java Message Service", O'Reilly, 2nd Ed.

QBorrower and QLender Example

To illustrate how point-to-point messaging works, we will use a simple decoupled request/reply example where a `QBorrower` class makes a simple mortgage loan request to a `QLender` class using point-to-point messaging. The `QBorrower` class sends the loan request to the `QLender` class using a `LoanRequest` queue, and based on certain business rules, the `QLender` class sends a response back to the `QBorrower` class using a `LoanResponseQ` queue indicating whether the loan request was approved or denied. Since the `QBorrower` is interested in finding out right away whether the loan was approved or not, once the loan request is sent, the `QBorrower` class will block and wait for a response from the QLender class before proceeding. This simple example models a typical messaging request/reply scenario.

Source: "Java Message Service", O'Reilly, 2nd Ed.

QBorrower

- The `QBorrower` class is responsible for sending a loan request message to a queue containing a salary and loan amount.
- The class is fairly straightforward: the constructor establishes a connection to the JMS provider, creates a `QueueSession`, and gets the request and response queues using a JNDI lookup.
- The main method instantiates the `QBorrower` class and, upon receiving a salary and loan amount from standard input, invokes the `sendLoanRequest` method to send the message to the queue.

Source: "Java Message Service", O'Reilly, 2nd Ed.

QLender

- The role of the `QLender` class is to listen for loan requests on the loan request queue, determine if the salary meets the necessary business requirements, and finally send the results back to the borrower.
- The `QLender` class is what is referred to as an asynchronous message listener, meaning that unlike the prior `QBorrower` class it will not block when waiting for messages. This is evident from the fact that the `QLender` class implements the `MessageListener` interface and overrides the `onMessage` method.
- The `onMessage` method first casts the message to a `MapMessage` (the message type we are expecting to receive from the borrower). It then extracts the salary and loan amount requested from the message payload, checks the salary to loan amount ratio, then determines whether to accept or decline the loan request.
- Once the loan request has been analyzed and the results determined, the `QLender` class sends the response back to the borrower.

Source: "Java Message Service", O'Reilly, 2nd Ed.

Message Correlation

- Once the message has been sent, the `QBorrower` class will block and wait for a response from the `QLender` on whether the loan was approved or denied.
- The first step in this process is to set up a message selector so that we can correlate the response message with the one we sent. This is necessary because there may be many other loan requests being sent to and from the loan request queues while we are making our loan request.
- To make sure we get the proper response back, we would use a technique called **message correlation**. Message correlation is required when using the request/reply model of point-to-point messaging where the queue is being shared by multiple producers and consumers:

```
String filter = "JMSCorrelationID = '" +
   msg.getJMSMessageID() + "'";
QueueReceiver qReceiver =
   qSession.createReceiver(responseQ, filter);
```

Source: "Java Message Service", O'Reilly, 2nd Ed.

Technische Universität Berlin

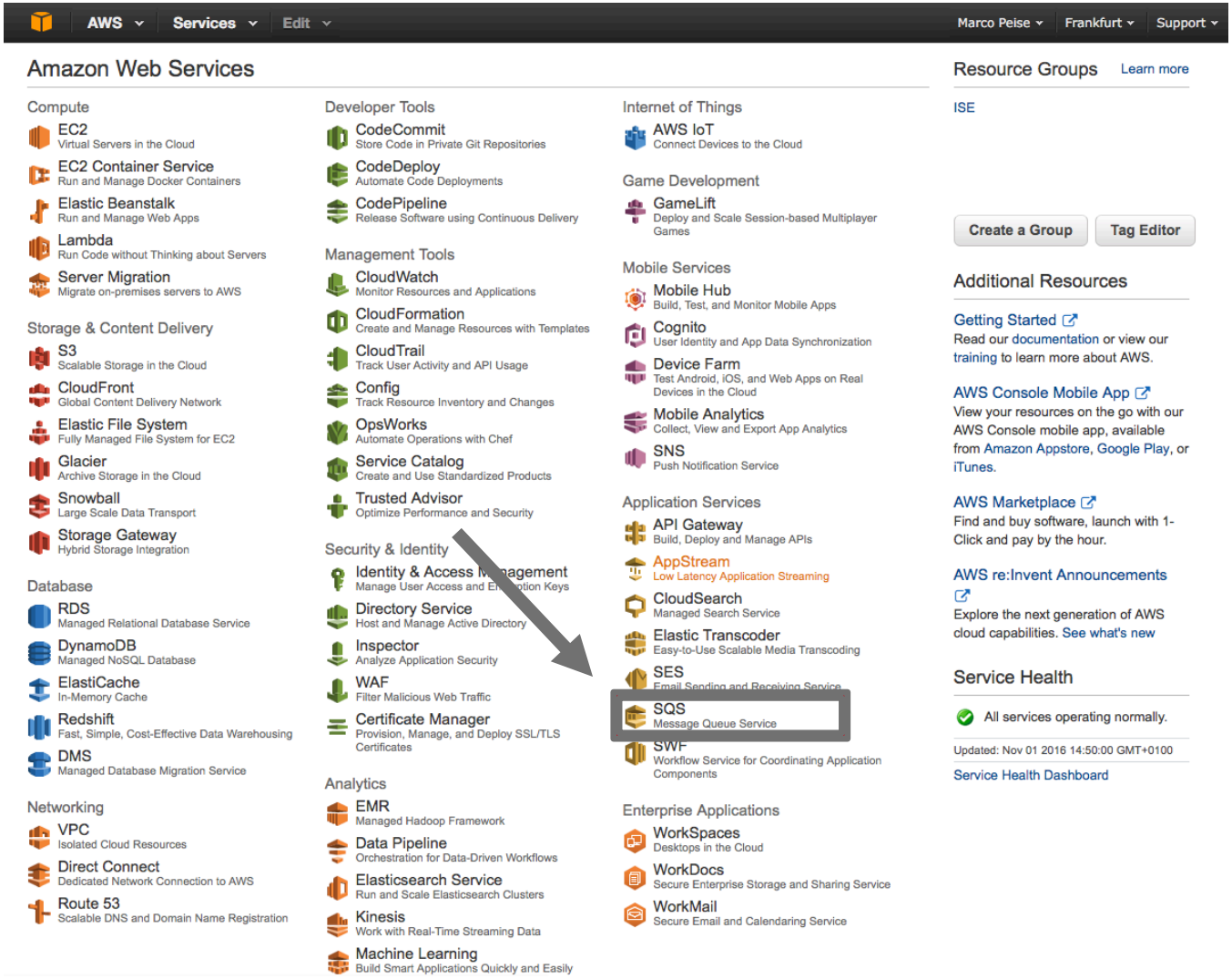
NOW, BACK TO AWS

Technische Universität Berlin

AWS SQS

ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

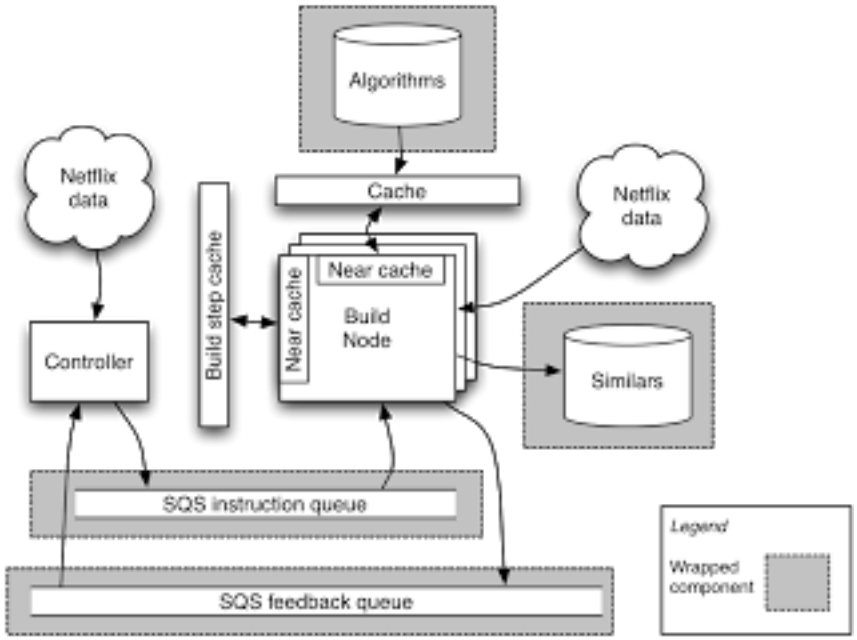
AWS SQS - Amazon Simple Queue Service



AWS S3 - Amazon Simple Queue Service

- since 2004
- guarantees at-least-once delivery
- redundancy and availability managed by storing messages on multiple servers
- ordering of messages optional (sequencing information within the messages)
- for authentication either SOAP with WS-Security or Key matching
- supports short messages up to larger messages which can either be split into segments or stored in S3
- other vendors IronMQ, StormMQ and AnypointMQ
- as a service (rather then self-server managing like IBM Websphere MQ or Microsoft Message Queuing)

AWS S3 - Amazon Simple Queue Service



Source: Netflix 2011, http://techblog.netflix.com/2011/04/more-like-this-building-network-of.html

Task 3 – Amazon SQS

Technische Universität Berlin

Please complete the borrower/lender example **with AWS SQS** (instead of JMS).

// *SqsBorrower.java*

… fill out the blanks …

// *SqsLender.java*

… fill out the blanks …

Task 3 – Amazon SQS