# Tutorial – Creating JAVA microservices to use remote Kakfa service and Cloud Database

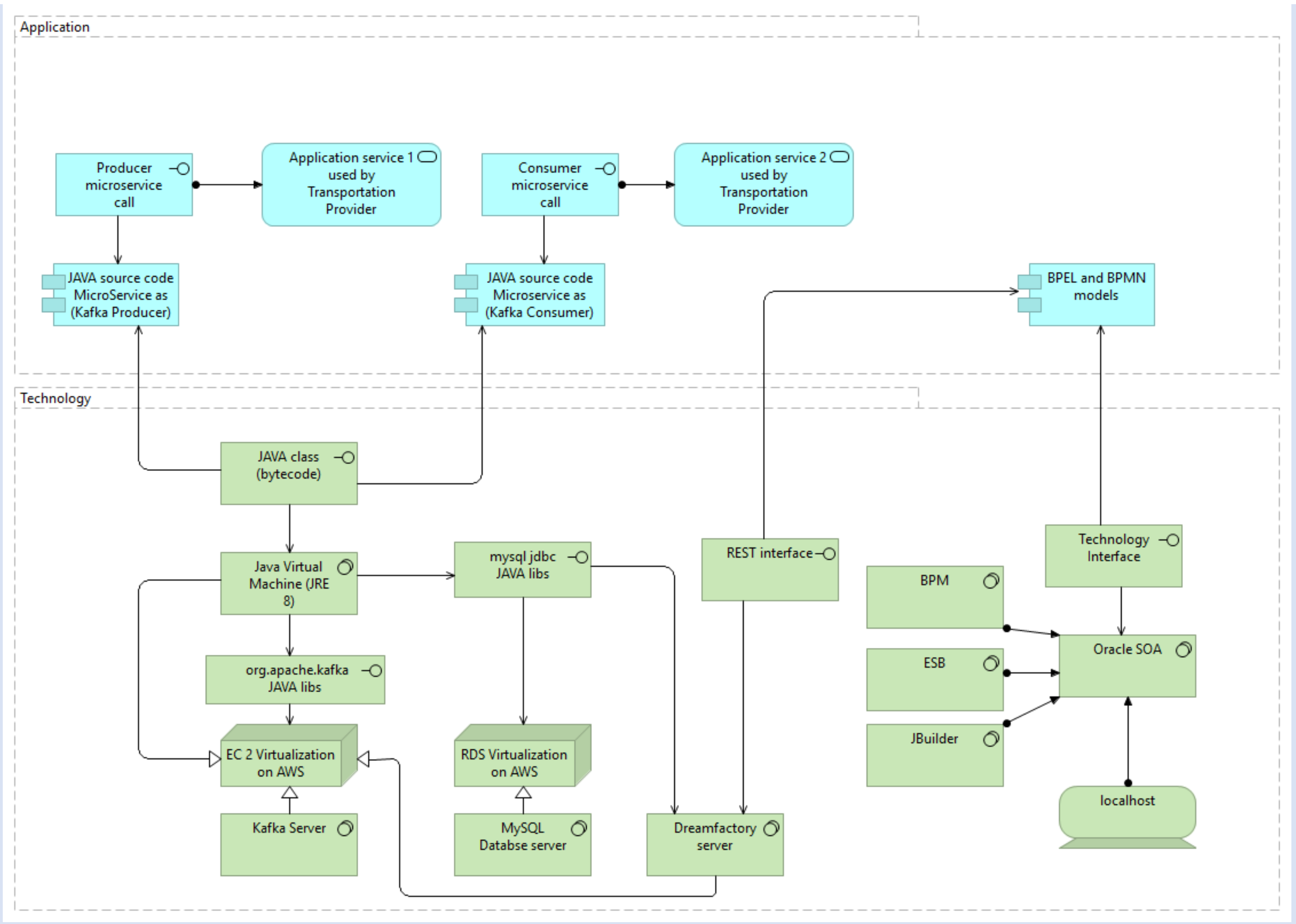
Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to show how to develop JAVA microservices that can interact with Kafka, and, with a database available in a cloud environment. The project should be built from these indications.
The usage of a cloud database requires an Amazon AWS account.
The previous Tutorial "Deploying Kakfa service in Amazon AWS EC2" is used as a service used by Kafka Producer and Kafka Consumer.

The following picture overviews the technological and application server dependencies that are introduced by this tutorial.

*Description:*
Any application service that requires the production or consumption of Kafka messages, in a predefined topic, invokes JVM and refers to the JAVA class and the required libraries, 1 and 2 are exemplified, more application services could be used. *Producer depends on org.apacha.kafka libs and Consumer depends on org.apacha.kafka libs and mysql.jdbc libs.*
In this example, both Producer and Consumer are configured to use the same Kafka server installed on EC2 Amazon AWS. After that, the Consumer propagates the received messages to an RDS Amazon database configured as a MySQL service. This architecture allows a light production and consumption of messages, and then storing the received messages for future use in a different repository.
Then, a Dreamfactory server is configured to deploy a REST interface that exposes the datamodel of the RDS Amazon database. All the following REST operations are allowed: GET, POST, PUT, PATCH, DELETE. These REST operations are consumed by the BPEL and BPMN models that are deployed in ORACLE SOA platform.

*Scalability:*
JAVA microservices could be developed and deployed on-the-fly accordingly with business needs.
EC2 virtualization could be partitioned and clustered if more performance is demanded.
RDS virtualization could be escalated, creating different SCHEMAS, or DATABASES instances, for different Application services.
A single DreamFactory server could be configured for multiple SCHEMAS, or DATABASES instances. Exposing each configuration as a different service with its own WEBAPP_id in the URL.



Application
Producer microservice call
Application service 1 used by Transportation Provider
Consumer microservice call
Application service 2 used by Transportation Provider
JAVA source code MicroService as (Kafka Producer)
JAVA source code Microservice as (Kafka Consumer)
BPEL and BPMN models
Technology
JAVA class (bytecode)
Java Virtual Machine (JRE 8)
mysql jdbc JAVA libs
REST interface
Technology Interface
BPM
ESB
JBuilder
Oracle SOA
org.apache.kafka JAVA libs
EC 2 Virtualization on AWS
RDS Virtualization on AWS
Kafka Server
MySQL Databse server
Dreamfactory server
localhost

The following contents is presented in this document.

## Contents

## A. Prerequisites for using Maven in Eclipse:

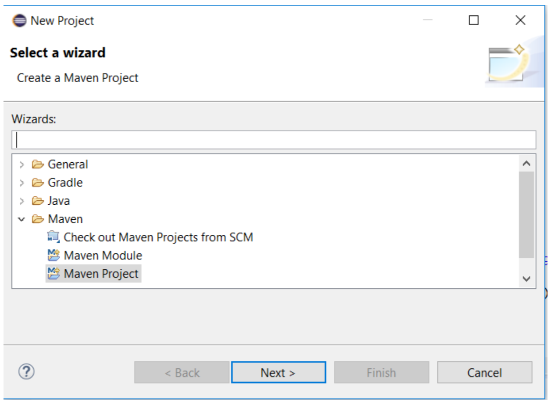
A.0. On the top menu bar, open Window -> Show View -> Other
A.1. In the Show View window, open Maven -> Maven Repositories
A.2. In the window that appears, right-click on Global Repositories and select Go Into
A.3. Right-click on "central (http://repo.maven.apache.org/maven2)" and select "Rebuild Index"
A.4. Note that it will take a while to complete the download
A.5. Once indexing is complete, Right-click on the project -> Maven -> Add Dependency and start typing the name of the project you want to import (such as "hibernate").
A.6. The search results will auto-fill in the "Search Results" box below.

Then to install Maven plugin in Eclipse:
Help -> install new software -> http://download.eclipse.org/technology/m2e/releases Add

## B. Create a JAVA Kafka Producer (for windows): example of

The goal of this section is to create a micro service in JAVA to produce messages for a Kafka service that has already been provisioned and a topic has been created.

1. Create a Maven project. File > New > Project…


New Project

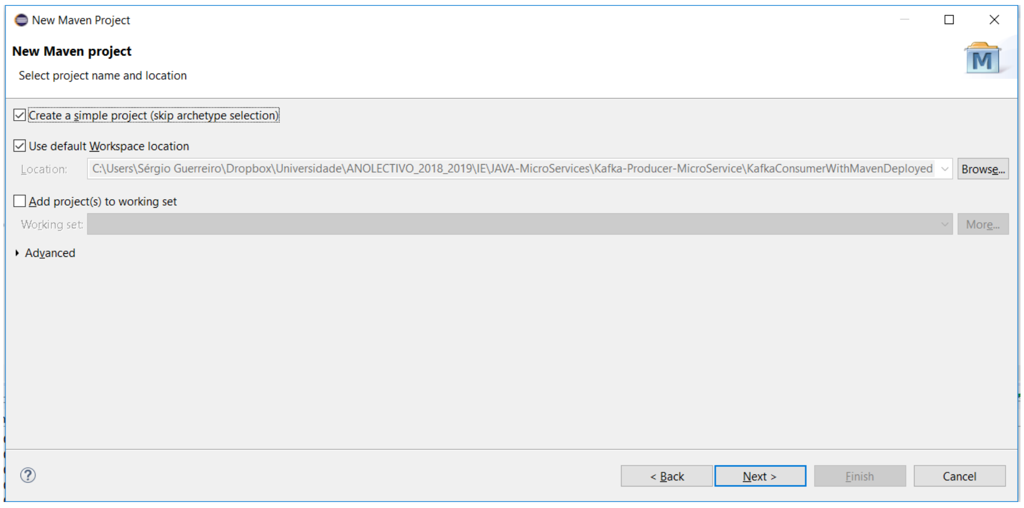
**Select a wizard**

Create a Maven Project

Wizards:

- General
- Gradle
- Java
- Maven
  - Check out Maven Projects from SCM
  - Maven Module
  - Maven Project

2. Check the option "Create a simple project (skip archetype selection) and then press Next


New Maven Project

**New Maven project**

Select project name and location

☑ Create a simple project (skip archetype selection)

☑ Use default Workspace location

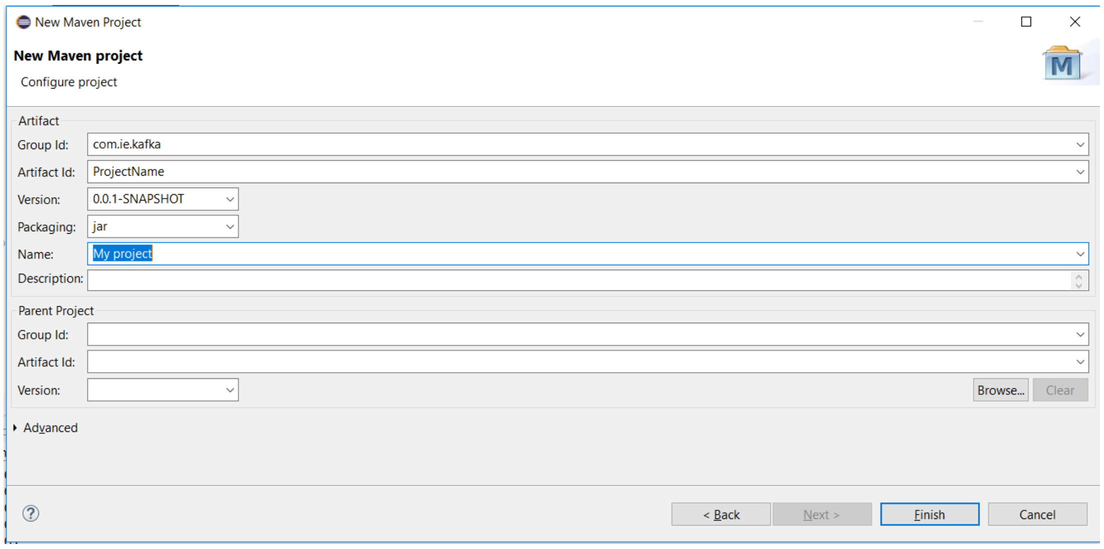
Location: C:\Users\Sérgio Guerreiro\Dropbox\Universidade\ANOLECTIVO_2018_2019\IE\JAVA-MicroServices\Kafka-Producer-MicroService\KafkaConsumerWithMavenDeployed

☐ Add project(s) to working set

Working set:

Advanced

Use the following configuration (consult Appendix: URLS with other resources further details). And then press Finish.


New Maven Project

**New Maven project**

Configure project

Artifact

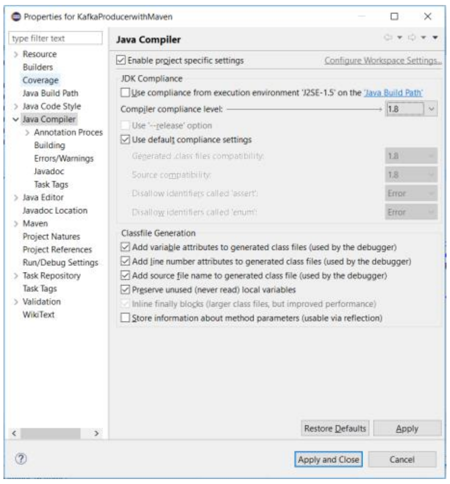
| | |
|---|---|
| Group Id: | com.ie.kafka |
| Artifact Id: | ProjectName |
| Version: | 0.0.1-SNAPSHOT |
| Packaging: | jar |
| Name: | My project |
| Description: | |

Parent Project

| | |
|---|---|
| Group Id: | |
| Artifact Id: | |
| Version: | |

Advanced

3. In Project > Properties > Java compiler check that compilation is done using java 8



Java Compiler

[✓] Enable project specific settings

JDK Compliance

[ ] Use compliance from execution environment 'J2SE-1.5' on the 'Java Build Path'

Compiler compliance level: 1.8

[ ] Use '--release' option

[✓] Use default compliance settings

Generated .class files compatibility: 1.8

Source compatibility: 1.8

Disallow identifiers called 'assert': Error

Disallow identifiers called 'enum': Error

Classfile Generation

[✓] Add variable attributes to generated class files (used by the debugger)

[✓] Add line number attributes to generated class files (used by the debugger)

[✓] Add source file name to generated class file (used by the debugger)

[✓] Preserve unused (never read) local variables

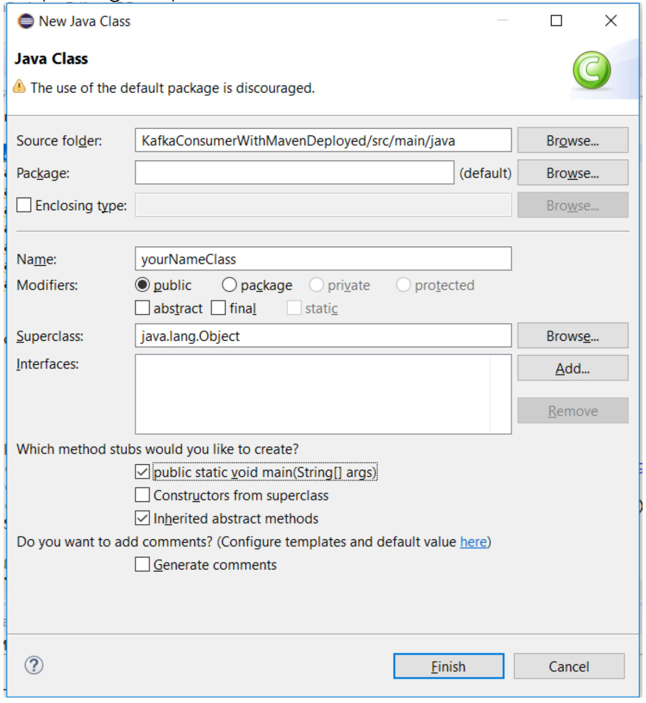
[✓] Inline finally blocks (larger class files, but improved performance)

[ ] Store information about method parameters (usable via reflection)

4. Define the following Maven dependency in pom.xml of your project:

```
<dependencies>
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.1.0</version>
</dependency>
</dependencies>
```

5. Right click in your project in the package explorer and create a new JAVA class with the following configuration:



Java Class

The use of the default package is discouraged.

Source folder: KafkaConsumerWithMavenDeployed/src/main/java

Package: (default)

[ ] Enclosing type:

Name: yourNameClass

Modifiers: [● public] ○ package ○ private ○ protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

6. Study the following source code and adapt it to your new class:

```
import java.util.Properties;
import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.serialization.LongSerializer;
import org.apache.kafka.common.serialization.StringSerializer;

public class ClientProducerKafka {
       public static void main(String[] args) {
               Properties kafkaProps = new Properties();
               kafkaProps.put("bootstrap.servers", "YOURAWSIP:9092");
               kafkaProps.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
               kafkaProps.put("value.serializer","org.apache.kafka.common.serialization.StringSerializer");
               KafkaProducer<String, String> producer = new KafkaProducer<String, String>(kafkaProps);

               //Fire-and-forget
               System.out.println("Fire-and-forget starting...");
               ProducerRecord<String, String> record = new ProducerRecord<>("EXERCISE3", "Precision
Products", "France");
               try
               {  producer.send(record); }
               catch (Exception e) { e.printStackTrace();}
               System.out.println("Fire-and-forget stopped.");

               //Synchronous send
               System.out.println("Synchronous send starting...");
               ProducerRecord<String, String> record2 =
               new ProducerRecord<>("EXERCISE3", "Precision Products", "France2");
               try {   System.out.println("Result get =" + producer.send(record2).get()); }
               catch (Exception e) {e.printStackTrace();   }
               System.out.println("Synchronous stopped.");
       }
}
```

7. Compile and try your project.

## C. Create a JAVA Kafka Consumer (for windows): example of

The goal of this section is to create a micro service in JAVA to consume messages from a Kafka service that has already been provisioned, a topic has been created, and messages are being sent to the topic.

1. Create a new Maven project (accordingly with section B).

2. In Project > Properties > Java compiler check that compilation is done using java 8

3. Define the following Maven dependency in pom.xml of your project:

```
<dependencies>
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.1.0</version>
</dependency>
</dependencies>
```

4. Create a new Java class (accordingly with section B)

5. Study the following source code and adapt it to your new class:

```
import java.util.Collections;
import java.util.Properties;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

public class ClientConsumerKafka {

       public static void main(String[] args) {
              // TODO Auto-generated method stub


              Properties props = new Properties();
              props.put("bootstrap.servers", "yourAWSIP:9092");
              props.put("group.id", "CountryCounter");
              props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
              props.put("value.deserializer","org.apache.kafka.common.serialization.StringDeserializer");
              KafkaConsumer<String, String> consumer = new KafkaConsumer<String,String>(props);
              consumer.subscribe(Collections.singletonList("EXERCISE3"));
              try {
                     while (true)
                     {
                            /*
       consumers must keep polling Kafka or they will be considered
       dead and the partitions they are consuming will be handed to another
       consumer in the group to continue consuming. The parameter we pass, poll(),
       is a timeout interval and controls how long poll() will block if data is not available
       in the consumer buffer. If this is set to 0, poll() will return immediately;
       otherwise, it will wait for the specified number of milliseconds for data to arrive
       from the broker.
                             */
                     ConsumerRecords<String, String> records = consumer.poll(100);
                     for (ConsumerRecord<String, String> record : records)
                      {
              System.out.printf("topic = %s, partition = %s, offset = %d,customer = %s, country = %s\n",
                            record.topic(), record.partition(), record.offset(),
                            record.key(), record.value());
                      }
                     }
              }
              finally {consumer.close();}
       }
}
```

6. Compile and try your project.

## D. Producing and Consuming an XML message (for windows): example of

The goal of this section is to present how to send an XML message to your consumer.

1. Start the Eclipse Consumer as explained before in this tutorial.

2. Produce the following XML message in docker client

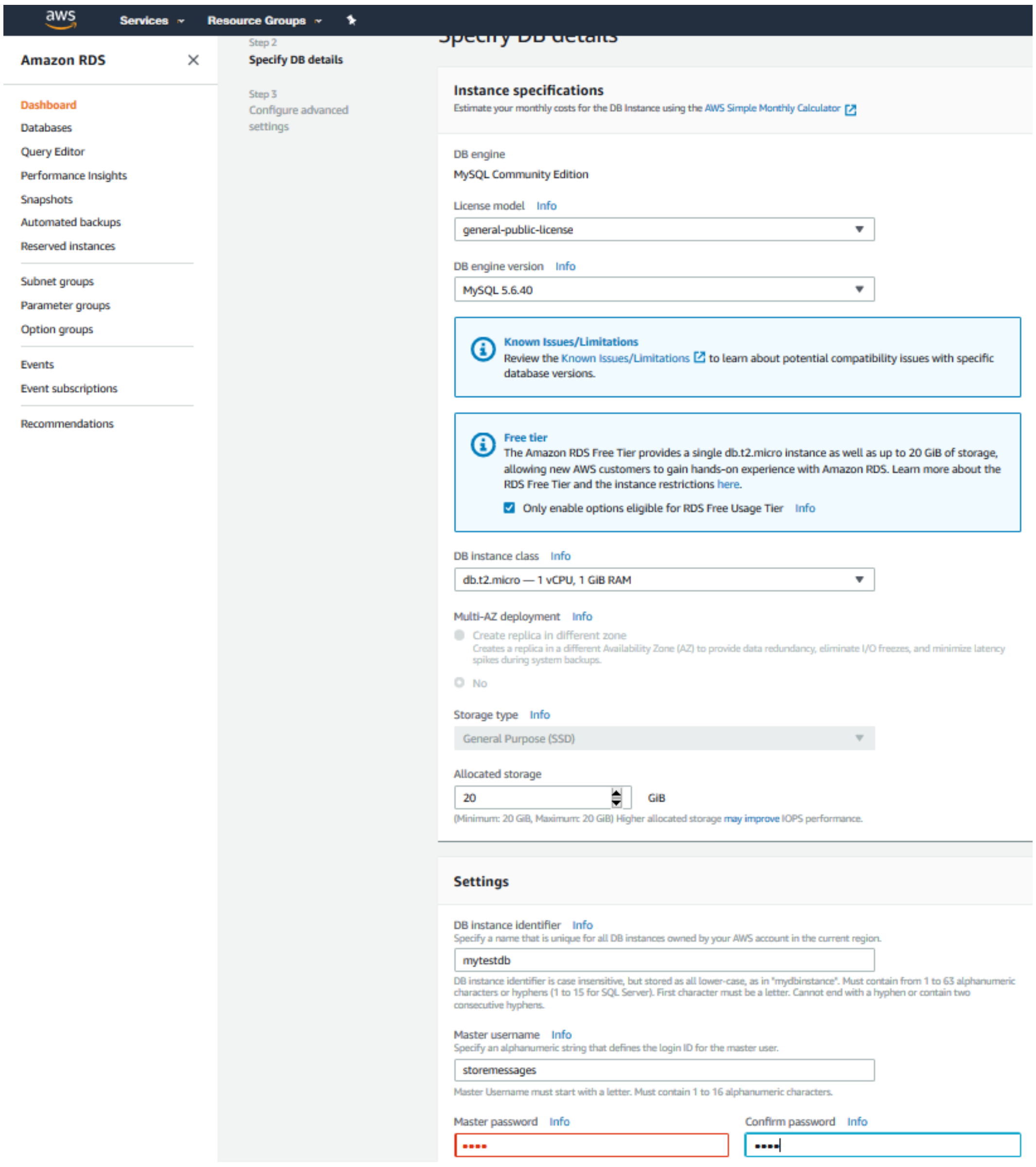
```
<?xml version="1.0" encoding="UTF-8" ?><!-- This is an XML document example --><invoice
xmlns="http://example.org" nr="220706" date="2015-07-22T11:05"><seller tax-
id="500600"><name>Restaurant</name></seller><buyer tax-id="555111"><name /></buyer><totals
currency="EUR"><value>100</value><tax>30</tax></totals><items><item><value>8</value><desc>Meal</desc></item><
item><value>2</value><desc>Water</desc></item></items></invoice>
```

3. Verify the message received in the consumer.

## E. Create an AWS RDS Database, accessing it, and inserting data: example of

The goal of this section is to exemplify how to create your own cloud database in AWS, to access its contents through a database workbench, and then creating a JAVA microservice to consume Kafka messages that are then inserted in the cloud database.

1. Go to your AWS account console and click on Database RDS -> My SQL and select the following configurations in step 2:



**Instance specifications**

Estimate your monthly costs for the DB Instance using the AWS Simple Monthly Calculator

DB engine
MySQL Community Edition

License model Info
general-public-license

DB engine version Info
MySQL 5.6.40

**Known Issues/Limitations**
Review the Known Issues/Limitations to learn about potential compatibility issues with specific database versions.

**Free tier**
The Amazon RDS Free Tier provides a single db.t2.micro instance as well as up to 20 GiB of storage, allowing new AWS customers to gain hands-on experience with Amazon RDS. Learn more about the RDS Free Tier and the instance restrictions here.

✓ Only enable options eligible for RDS Free Usage Tier Info

DB instance class Info
db.t2.micro — 1 vCPU, 1 GiB RAM

Multi-AZ deployment Info
Create replica in different zone
Creates a replica in a different Availability Zone (AZ) to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.
No

Storage type Info
General Purpose (SSD)

Allocated storage
20 GiB
(Minimum: 20 GiB, Maximum: 20 GiB) Higher allocated storage may improve IOPS performance.

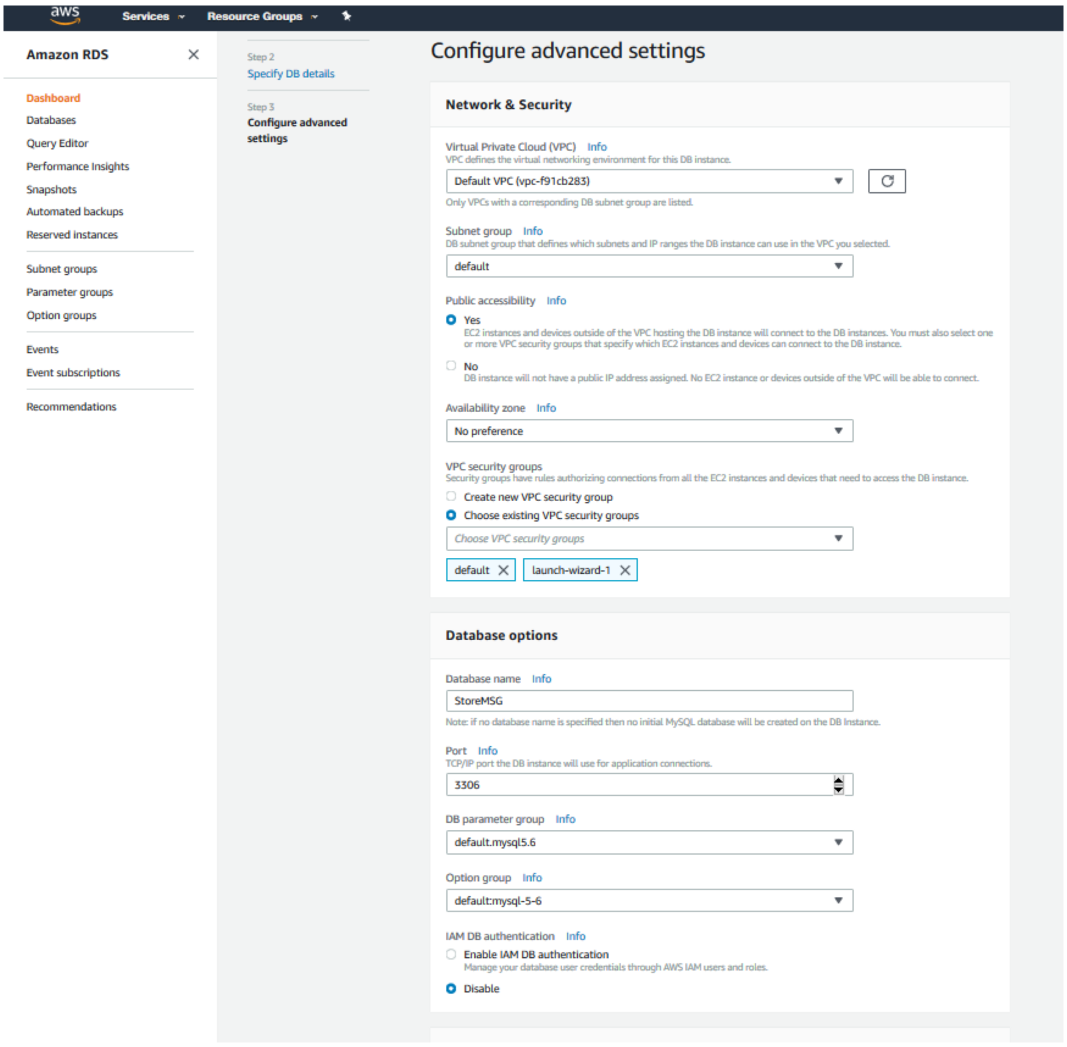
**Settings**

DB instance identifier Info
Specify a name that is unique for all DB instances owned by your AWS account in the current region.
mytestdb
DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance". Must contain from 1 to 63 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Cannot end with a hyphen or contain two consecutive hyphens.

Master username Info
Specify an alphanumeric string that defines the login ID for the master user.
storemessages
Master Username must start with a letter. Must contain 1 to 16 alphanumeric characters.

Master password Info
••••

Confirm password Info
••••



Configure advanced settings

**Network & Security**

Virtual Private Cloud (VPC) Info
VPC defines the virtual networking environment for this DB instance.
Default VPC (vpc-f91cb283)
Only VPCs with a corresponding DB subnet group are listed.

Subnet group Info
DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.
default

Public accessibility Info
[✓ Yes]
EC2 instances and devices outside of the VPC hosting the DB instance will connect to the DB instances. You must also select one or more VPC security groups that specify which EC2 instances and devices can connect to the DB instance.
No
DB instance will not have a public IP address assigned. No EC2 instance or devices outside of the VPC will be able to connect.

Availability zone Info
No preference

VPC security groups
Security groups have rules authorizing connections from all the EC2 instances and devices that need to access the DB instance.
Create new VPC security group
[✓ Choose existing VPC security groups]
Choose VPC security groups
default ✕ launch-wizard-1 ✕

**Database options**

Database name Info
StoreMSG
Note: if no database name is specified then no initial MySQL database will be created on the DB Instance.

Port Info
TCP/IP port the DB instance will use for application connections.
3306

DB parameter group Info
default.mysql5.6

Option group Info
default:mysql-5-6

IAM DB authentication Info
Enable IAM DB authentication
Manage your database user credentials through AWS IAM users and roles.
[✓ Disable]

2. Check if the database is started.

3. Use your MySQL database client (for example MySQL workbench), connect to your RDS AWS database and execute the following scripts:

```
DROP DATABASE IF EXISTS collectedMSGs;
CREATE DATABASE IF NOT EXISTS collectedMSGs;
```

```
USE collectedMSGs;
DROP TABLE IF EXISTS Message;

CREATE TABLE Message( offset INTEGER PRIMARY KEY,
                      groupId VARCHAR(100) NOT NULL,
                      contentKey VARCHAR(100),
                      contentValue VARCHAR(1000));
```

4. Create a new Maven Project in Eclipse.

5. Download the mysql/J connector from https://dev.mysql.com/downloads/connector/j/ and add it to the external library of your project.

6. Use the following dependency in pom.xml

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.13</version>
</dependency>
```

7. Study the following source code and adapt it to your new class:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Properties;

import org.apache.kafka.clients.consumer.CommitFailedException;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.TopicPartition;

public class ClientConsumerBDCloud {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Properties props = new Properties();
		props.put("bootstrap.servers", "yourAWSIP:9092");
		props.put("group.id", "1CountryCounter");
		props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
		props.put("value.deserializer","org.apache.kafka.common.serialization.StringDeserializer");
		props.put("auto.commit.offset","false"); //to commit manually
		KafkaConsumer<String, String> consumer = new KafkaConsumer<String,String>(props);
		consumer.subscribe(Collections.singletonList("EXERCISE3"));
		Connection conn = null;
		boolean bd_ok = false;
		try {
			Class.forName("com.mysql.cj.jdbc.Driver");
			conn = DriverManager.getConnection("jdbc:mysql://yourAWSDBIP:3306/YOURDATABASENAME",
							"YOURMasterUSERNAME", "YOURPASSWORD");
			bd_ok = true;
		}
		catch (SQLException sqle) {   System.out.println("SQLException: " + sqle); }
		catch (ClassNotFoundException e) {   e.printStackTrace();}

		try {
			while (true)
			{
				ConsumerRecords<String, String> records = consumer.poll(100);
				for (ConsumerRecord<String, String> record : records)
				{
				System.out.printf("topic = %s, partition = %s, offset = %d,customer = %s,
				country = %s\n",record.topic(), record.partition(), record.offset(),
				record.key(), record.value());

				if (bd_ok)
				{
					PreparedStatement s;
					s = conn.prepareStatement ("insert into Message values(?,?,?,?)");
					s.setInt(1, new Long(record.offset()).intValue());
					s.setString(2, props.getProperty("group.id"));
					s.setString(3, record.key());
					s.setString(4, record.value());
					s.executeUpdate();
					s.close();
				}
			}
			//Commit Current Offset
			try { consumer.commitSync(); }
			catch (CommitFailedException e) { System.out.printf("commit failed", e);}
			}
		} catch (SQLException e) {   e.printStackTrace(); }
		finally {
			consumer.close();
			try {
				conn.close();
			} catch (SQLException e) {   e.printStackTrace();  }
		}
	}
}
```

8. Compile and try your project.

## F. Deploying JAVA services in EC2: example of

1. Create an EC2 Amazon linux image instance
2. Check that JDK 8 is available in the new created instance with the following commands:

```
[ec2-user@ip-  - -  ~]$ java -version
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
[ec2-user@ip- - - - ~]$ javac -version
javac 1.8.0_191
```

*Hint 1: if not available, use the following commands:*

```
[ec2-user@ip- - - -  ~]$ yum search jdk
Loaded plugins: priorities, update-motd, upgrade-helper
================================ N/S matched: jdk ================================
copy-jdk-configs.noarch : JDKs configuration files copier
java-1.6.0-openjdk.x86_64 : OpenJDK Runtime Environment
java-1.6.0-openjdk-demo.x86_64 : OpenJDK Demos
java-1.6.0-openjdk-devel.x86_64 : OpenJDK Development Environment
java-1.6.0-openjdk-javadoc.x86_64 : OpenJDK API Documentation
java-1.6.0-openjdk-src.x86_64 : OpenJDK Source Bundle
java-1.7.0-openjdk.x86_64 : OpenJDK Runtime Environment
java-1.7.0-openjdk-demo.x86_64 : OpenJDK Demos
java-1.7.0-openjdk-devel.x86_64 : OpenJDK Development Environment
java-1.7.0-openjdk-javadoc.noarch : OpenJDK API Documentation
java-1.7.0-openjdk-src.x86_64 : OpenJDK Source Bundle
java-1.8.0-openjdk.x86_64 : OpenJDK Runtime Environment
java-1.8.0-openjdk-demo.x86_64 : OpenJDK Demos
java-1.8.0-openjdk-devel.x86_64 : OpenJDK Development Environment
java-1.8.0-openjdk-headless.x86_64 : OpenJDK Runtime Environment
java-1.8.0-openjdk-javadoc.noarch : OpenJDK API Documentation
java-1.8.0-openjdk-javadoc-zip.noarch : OpenJDK API Documentation compressed in
                                      : single archive
java-1.8.0-openjdk-src.x86_64 : OpenJDK Source Bundle
ldapjdk-javadoc.noarch : Javadoc for ldapjdk
ldapjdk.noarch : The Mozilla LDAP Java SDK

[ec2-user@ip- - - -  ~]$ sudo yum install java-1.8.0-openjdk-devel.x86_64
Loaded plugins: priorities, update-motd, upgrade-helper
amzn-main                                                | 2.1 kB     00:00
amzn-updates                                             | 2.5 kB     00:00
Package 1:java-1.8.0-openjdk-devel-1.8.0.191.b12-0.42.amzn1.x86_64 already installed and
latest version
Nothing to do
```

Upload apache-maven binary file that is downloaded from URL referred in the end of this document (section Appendix: URLS with other resources).

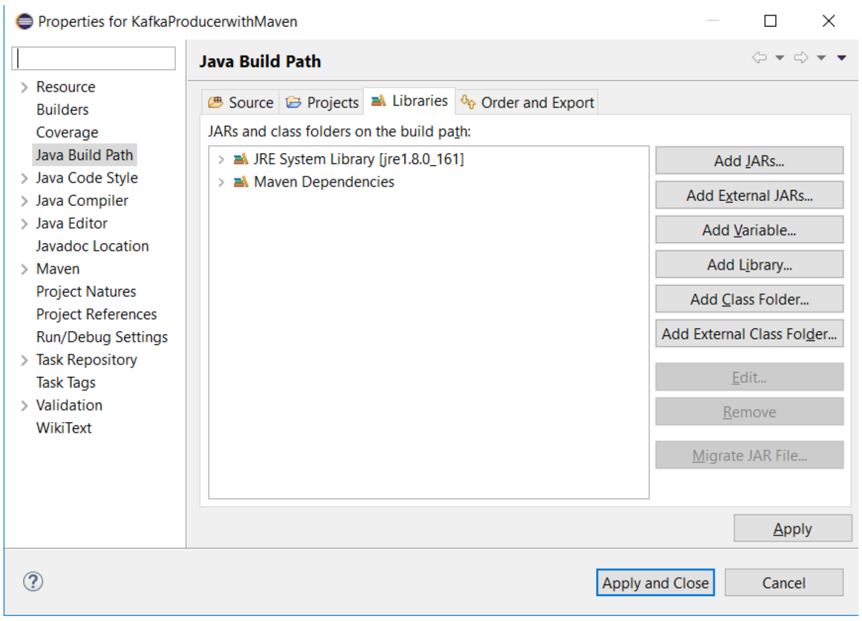
3. Execute the following commands in the EC2 instance:

```
[ec2-user@ip- - - -  ~]$ tar xzvf apache-maven-3.6.0-bin.tar.gz

[ec2-user@ip- - - -  ~]$ cd apache-maven-3.6.0/bin

[ec2-user@ip- - - -  bin]$ pwd
/home/ec2-user/apache-maven-3.6.0/bin

[ec2-user@ip- - - -  bin]$ export PATH=/home/ec2-user/apache-maven-3.6.0/bin:$PATH

[ec2-user@ip- - - -  ~]$ mvn -version
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-24T18:41:47Z)
Maven home: /home/ec2-user/apache-maven-3.6.0
Java version: 1.8.0_191, vendor: Oracle Corporation, runtime: /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.191.b12-
0.42.amzn1.x86_64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.14.77-70.59.amzn1.x86_64", arch: "amd64", family: "unix"
```

*Hint 2: To automate the maven path with instance login update the export command in the file:*

```
.bash_profile
```

4. Check in Project -> Properties -> Add Library that JAVA8 is added to the environment


Properties for KafkaProducerwithMaven

Java Build Path

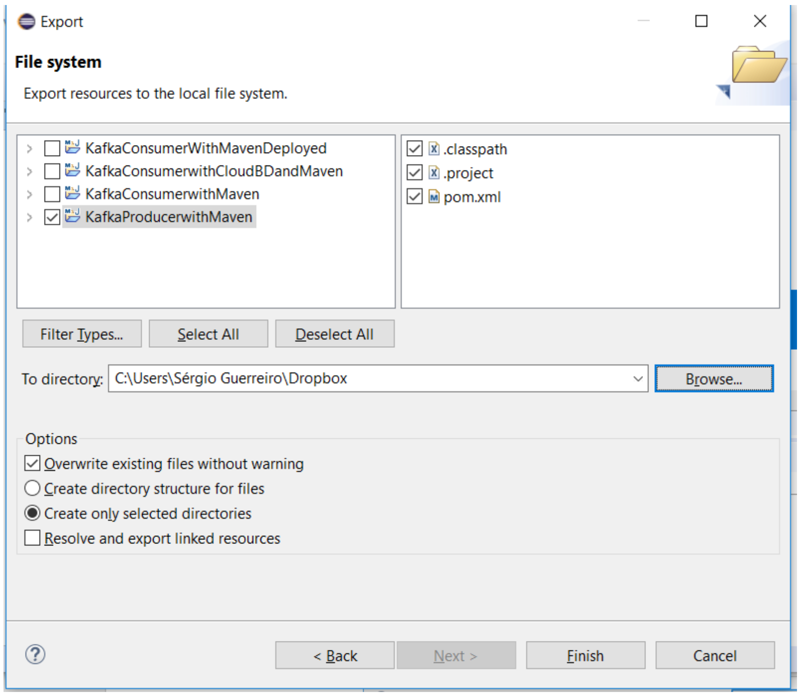
JARs and class folders on the build path:

- JRE System Library [jre1.8.0_161]
- Maven Dependencies

5. Also, in pom.xml file add the following:

```
<properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

6. Right click in the Eclipse project explorer, then Export project as a file system from Eclipse to your computer, as follows:


Export

File system

Export resources to the local file system.

- [ ] KafkaConsumerWithMavenDeployed
- [ ] KafkaConsumerwithCloudBDandMaven
- [ ] KafkaConsumerwithMaven
- [x] KafkaProducerwithMaven

- [x] .classpath
- [x] .project
- [x] pom.xml

To directory: C:\Users\Sérgio Guerreiro\Dropbox

Options

- [x] Overwrite existing files without warning
- ○ Create directory structure for files
- ● Create only selected directories
- [ ] Resolve and export linked resources

7. Upload the exported folder project to EC2 AMI using FileZillla.
8. Execute the following commands inside the exported folder project:

```
[ec2-user@ip- - - -  KafkaProducerwithMaven]$ mvn exec:java -Dexec.mainClass="ClientProducerKafka"
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< com.ie.microservices:KafkaProducerwithMaven >-------------
[INFO] Building My project 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:1.6.0:java (default-cli) @ KafkaProducerwithMaven ---
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
```

```
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Fire-and-forget starting...
Fire-and-forget stopped.
Synchronous send starting...
Result get =EXERCISE3-0@13
Synchronous stopped.
Asynchronous send starting...
Asynchronous send stopped.
```

## G. Dreamfactory installation and operation in AWS EC2

1. Download the version of linux operating system from https://www.dreamfactory.com/downloads
2. Create AWS EC2 instance – AMI and start instance.
3. SFTP the downloaded file in step 1. to your EC2 new instance.
4. If your instance has less than 1Gb of RAM, execute the following sequence of commands. Otherwise, jump directly to the command “sudo ./bitnami-dreamfactory-2.14.1-4-linux-x64-installer.run”.

```
Authenticating with public key "imported-openssh-key"
Last login: Tue Jan 15 12:03:41 2019 from kdbio47.inesc-id.pt

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/

[ec2-user@ip-172-31-85-254 ~]$ sudo yum update

[ec2-user@ip-172-31-85-254 ~]$ free -m
             total       used       free     shared    buffers     cached
Mem:           985        715        270          0         16        615
-/+ buffers/cache:         83        902
Swap:            0          0          0

[ec2-user@ip-172-31-85-254 /]$ sudo dd if=/dev/zero of=/mnt/swap.0 bs=1024 count=1048576
1048576+0 records in
1048576+0 records out
1073741824 bytes (1.1 GB) copied, 14.0105 s, 76.6 MB/s

[ec2-user@ip-172-31-85-254 /]$ sudo chmod 600 /mnt/swap.0

[ec2-user@ip-172-31-85-254 /]$ sudo mkswap /mnt/swap.0
Setting up swapspace version 1, size = 1048572 KiB
no label, UUID=ba957b3b-8175-41b2-8441-d362e082b9a7

[ec2-user@ip-172-31-85-254 /]$ sudo swapon /mnt/swap.0

[ec2-user@ip-172-31-85-254 /]$ swapon -s
Filename                                Type            Size    Used    Priority
/mnt/swap.0                             file    1048572 0       -2

[ec2-user@ip-172-31-85-254 /]$ cd

[ec2-user@ip-172-31-85-254 ~]$ sudo ./bitnami-dreamfactory-2.14.1-4-linux-x64-installer.run
----------------------------------------------------------------------------
Welcome to the Bitnami DreamFactory Stack Setup Wizard.

----------------------------------------------------------------------------
Select the components you want to install; clear the components you do not want
to install. Click Next when you are ready to continue.

NGINX Open Source [Y/n] :Y

DreamFactory : Y (Cannot be edited)

PhpMyAdmin [Y/n] :Y

Is the selection above correct? [Y/n]: Y

----------------------------------------------------------------------------
Installation folder

Please, choose a folder to install Bitnami DreamFactory Stack

Select a folder [/opt/dreamfactory-2.14.1-4]:

----------------------------------------------------------------------------
Please enter your database root user password

MariaDB and MongoDB root password :
Re-enter password :
Do you want to configure mail support? [y/N]: N
```

```
----------------------------------------------------------------------------
Setup is now ready to begin installing Bitnami DreamFactory Stack on your
computer.

Do you want to continue? [Y/n]: Y

----------------------------------------------------------------------------
Please wait while Setup installs Bitnami DreamFactory Stack on your computer.

 Installing
 0% ______________ 50% ______________ 100%
 #########################################

----------------------------------------------------------------------------
Setup has finished installing Bitnami DreamFactory Stack on your computer.

Launch Bitnami DreamFactory Stack [Y/n]: Y

[ec2-user@ip-172-31-85-254 ~]$ free -m
             total       used       free     shared    buffers     cached
Mem:           985        620        365         18         53        321
-/+ buffers/cache:        245        740
Swap:         1023          0       1023

[ec2-user@ip-172-31-85-254 ~]$ cd /opt/dreamfactory-2.14.1-4/

[ec2-user@ip-172-31-85-254 dreamfactory-2.14.1-4]$ sudo ./ctlscript.sh status
apache already running
mysql already running
mongodb already running
```

*Hint1: after an instance reboot, the dreamfactory is restarted with:*

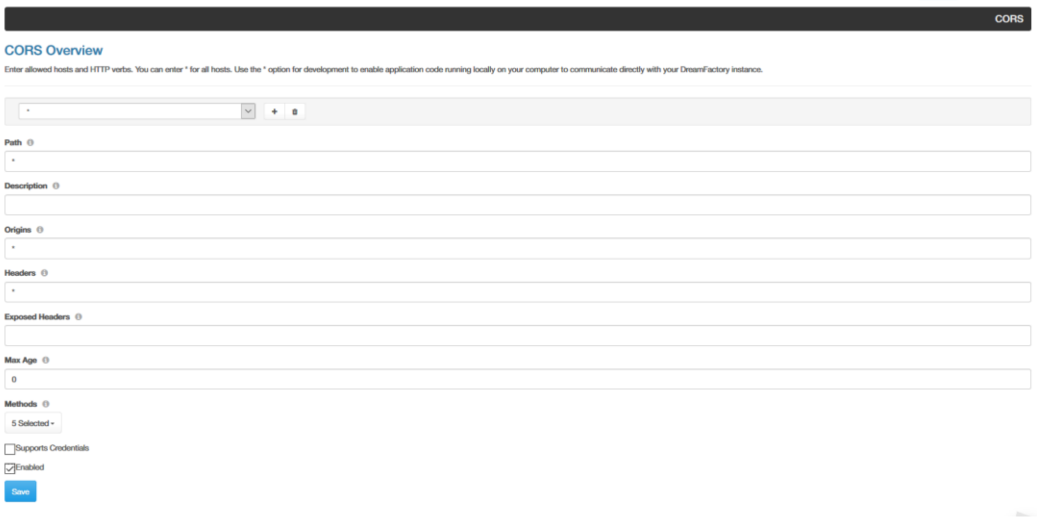
```
Using username "ec2-user".
Authenticating with public key "imported-openssh-key"
Last login: Tue Jan 15 12:07:21 2019 from kdbio47.inesc-id.pt

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
[ec2-user@ip-172-31-85-254 ~]$ cd /opt/dreamfactory-2.14.1-4/

[ec2-user@ip-172-31-85-254 dreamfactory-2.14.1-4]$ sudo ./ctlscript.sh status
apache not running
mysql not running
mongodb not running

[ec2-user@ip-172-31-85-254 dreamfactory-2.14.1-4]$ sudo ./ctlscript.sh start
/opt/dreamfactory-2.14.1-4/mysql/scripts/ctl.sh : mysql  started at port 3306
/opt/dreamfactory-2.14.1-4/mongodb/scripts/ctl.sh : mongodb started at port 27017
Syntax OK
/opt/dreamfactory-2.14.1-4/apache2/scripts/ctl.sh : httpd started at port 80

[ec2-user@ip-172-31-85-254 dreamfactory-2.14.1-4]$ sudo ./ctlscript.sh status
apache already running
mysql already running
mongodb already running
```

Your dreamfactory is now installed and running. To test the invocation of the REST interface, execute the following steps:

5. Access your dreamfactory server at http://YourAWSInstanceName:8080
6. Create your account
7. Choose Services -> Create button -> Select Service Type -> Database -> MySQL
8. Use the following configuration in Info tab:
   *Name:* TestMySQLAPI
   *Label:* for testing of TestMySQLAPI
   *Description:* this setup is for testing of TestMySQLAPI
9. Use the following configuration in Config tab:

*Host:* your Amazon RDS endpoint. It can be consulted at AWS Console -> RDS -> Databases
*Port Number:* 3306 (default)
*Database:* the name of your database previously created
*Username:* your username
*Password:* your password

10. Press Save button
11. *Optional:* You can check a set of interactive Swagger documentation, in API Docs
12. Secure your REST API, in Roles-> Create Button
13. Use the following configuration in Basic tab:
    *Name:* aws-mysql
    *Description:* this setup is for testing of TestMySQLAPI
    *Check* Active
14. Use the following configuration in Access tab:
    Press "Add new service access" in the "+" button, and choose:
    *Service:* your service defined in step 8.
    *Component:* your table name, e.g., "_table/Message/*"
    *Access:* choose the access that you would like to grant from - GET, POST, PUT, PATCH, DELETE
15. Press Save button
16. Create a new API key and associate the key with this role, in Apps -> Create Button, and use the following configuration.
    *Application Name:* MySQLAWS
    *Description:* The name of your description
    *Assign a Default Role:* your role name defined in step 13
    *App Location:* No Storage Required - remote device, client, or desktop.
    *Check* Active
17. Press the Save button and you'll be returned to the Apps index screen where the new API key can be copied! Copy the key into a text file for later reference.
18. Enable CORS (Cross-Origin Resource Sharing) for the new API, in Config -> CORS. Is will allow API-restricted traffic from all network addresses. Use the following configuration.



CORS

CORS Overview

Enter allowed hosts and HTTP verbs. You can enter * for all hosts. Use the * option for development to enable application code running locally on your computer to communicate directly with your DreamFactory instance.

Path: *
Description:
Origins: *
Headers: *
Exposed Headers:
Max Age: 0
Methods: 5 Selected
☐ Supports Credentials
☑ Enabled
Save

19. Test your REST API using the URLs:
    http://YourAWSIP:8080/api/v2/Your Service/_table/YourTable?api_key=yourAPIKey

*Hint 2: If you need to filter results then use: https://{url}/api/v2/{api_name}/_table/{table_name}?filter={filter_string} , e.g.: https://{url}/api/v2/{api_name}/_table/{table_name}?api_key=yourAPIKEY&filter=groupId%20like%201CountryCounter3*

```
Pay Attention for "\" ->
curl -v
URL:8080/api/v2/TestMySQLAPIable/_table/Message?api_key=apikey\&filter=groupId%20like%201CountryCounter
```

or using a CURL command:

```
PS C:\Users\Sérgio Guerreiro> curl -v
35.180.252.107:8080/api/v2/TestMySQLAPI/_table/Message?api_key=e114bbf45781ad62a70
dee9a16db26a86a852c2f8c44ec858699d20f46c1f7d9
VERBOSE: GET
http://35.180.252.107:8080/api/v2/TestMySQLAPI/_table/Message?api_key=e114bbf45781ad62a70dee9a16db26a86a852c2
f8c44ec858
699d20f46c1f7d9 with 0-byte payload
VERBOSE: received 403-byte response of content type application/json


StatusCode        : 200
StatusDescription : OK
Content           :
{"resource":[{"offset":182,"groupId":"1CountryCounter","contentKey":null,"contentValue":"sdssffd"},
                    {"offset":183,"groupId":"1CountryCounter","contentKey":"Precision
                    Products","contentValue":"France"},...
RawContent        : HTTP/1.1 200 OK
                    X-Frame-Options: SAMEORIGIN
                    Vary: Cookie
                    Keep-Alive: timeout=5, max=100
                    Connection: Keep-Alive
                    Content-Length: 403
                    Cache-Control: no-cache, private
                    Content-Type: application/jso...
Forms             : {}
Headers           : {[X-Frame-Options, SAMEORIGIN], [Vary, Cookie], [Keep-Alive, timeout=5, max=100],
[Connection,
                    Keep-Alive]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 403
```

*Hint 3: automate your REST API tests using Postman tool.*

## H. Creating a topic in Kafka from JAVA

Follow the similar steps presented in section “B. Create a JAVA Kafka Producer (for windows): example of” of this document with the following solution:

Maven dependencies:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka_2.11</artifactId>
    <version>0.10.2.0</version>
</dependency>
<dependency>
    <groupId>com.101tec</groupId>
    <artifactId>zkclient</artifactId>
    <version>0.9</version>
</dependency>
```

Java code:

```
import org.I0Itec.zkclient.ZkClient;
import org.I0Itec.zkclient.ZkConnection;

import java.util.Properties;

import kafka.admin.AdminUtils;
import kafka.admin.RackAwareMode;
import kafka.utils.ZKStringSerializer$;
import kafka.utils.ZkUtils;

public class KafkaJavaExample {

  public static void main(String[] args) {
    String zookeeperConnect = "zkserver1:2181";
    int sessionTimeoutMs = 10 * 1000;
    int connectionTimeoutMs = 8 * 1000;

    String topic = "my-NEW-topic";
    int partitions = 1;
    int replication = 1;
    Properties topicConfig = new Properties(); // add per-topic configurations settings here

    // Note: You must initialize the ZkClient with ZKStringSerializer.  If you don't, then
    // createTopic() will only seem to work (it will return without error).  The topic will exist in
    // only ZooKeeper and will be returned when listing topics, but Kafka itself does not create the
    // topic.
    ZkClient zkClient = new ZkClient(
        zookeeperConnect,
        sessionTimeoutMs,
        connectionTimeoutMs,
        ZKStringSerializer$.MODULE$);

    // Security for Kafka was added in Kafka 0.9.0.0
    boolean isSecureKafkaCluster = false;

    ZkUtils zkUtils = new ZkUtils(zkClient, new ZkConnection(zookeeperConnect), isSecureKafkaCluster);
    AdminUtils.createTopic(zkUtils, topic, partitions, replication, topicConfig,
RackAwareMode.Enforced$.MODULE$);
    zkClient.close();
  }
}
```

## I. Appendix: URLs with other resources

- To create a Maven project in eclipse: https://www.tech-recipes.com/rx/39279/create-a-new-maven-project-in-eclipse/
- About Kafka group-ids: https://stackoverflow.com/questions/35561110/can-multiple-kafka-consumers-read-same-message-from-the-partition
- More details about DreamFactory usage: https://blog.dreamfactory.com/create-a-mysql-rest-api-in-minutes-using-dreamfactory/
- Where to download apache maven https://maven.apache.org/download.cgi
- How to install apache maven https://maven.apache.org/install.html
- Postman tool: https://www.getpostman.com/postman
- http://wiki.dreamfactory.com/DreamFactory/Tutorials/Querying_records_with_logical_filters