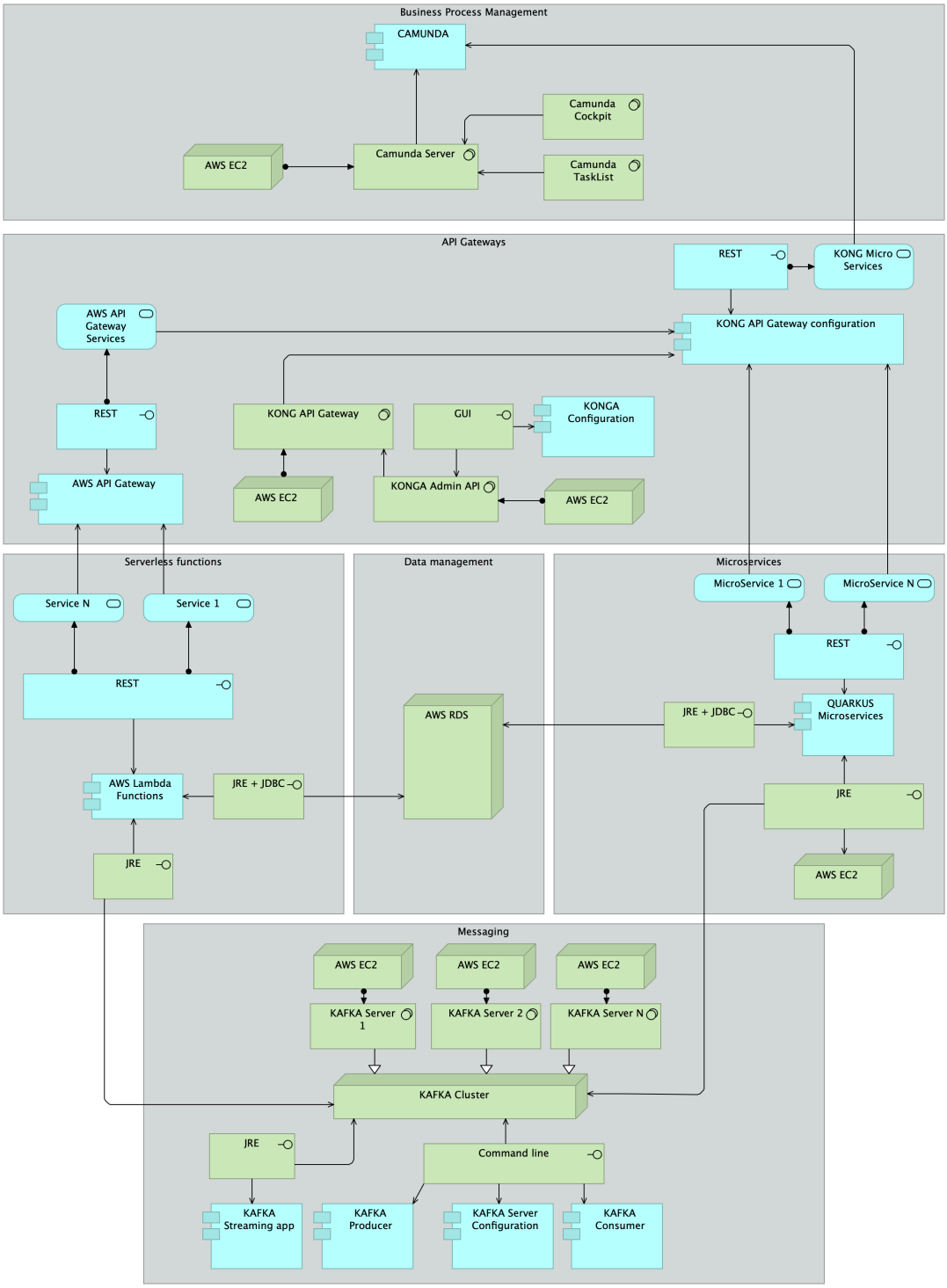
DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Tutorials for practical classes

Instituto Superior Técnico, Universidade de Lisboa

Integração Empresarial / Enterprise Integration

The following tutorials describe, in detail, the expected technological implementation using examples. The target integration architecture is overviewed in the following ArchiMate model (Application and Technological layers).

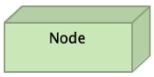


ArchiMate used elements:

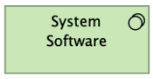
An application component represents an encapsulation of application functionality aligned to implementation structure, which is modular and replaceable.

An application interface represents a point of access where application services are made available to a user, another application component, or a node.

An application service represents an explicitly defined exposed application behavior.



A node represents a computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources.



A System software represents software that provides or contributes to an environment for storing, executing, and using software or data deployed within it.

A technology interface represents a point of access where technology services offered by a node can be accessed.

The serving relationship represents that an element provides its functionality to another element.

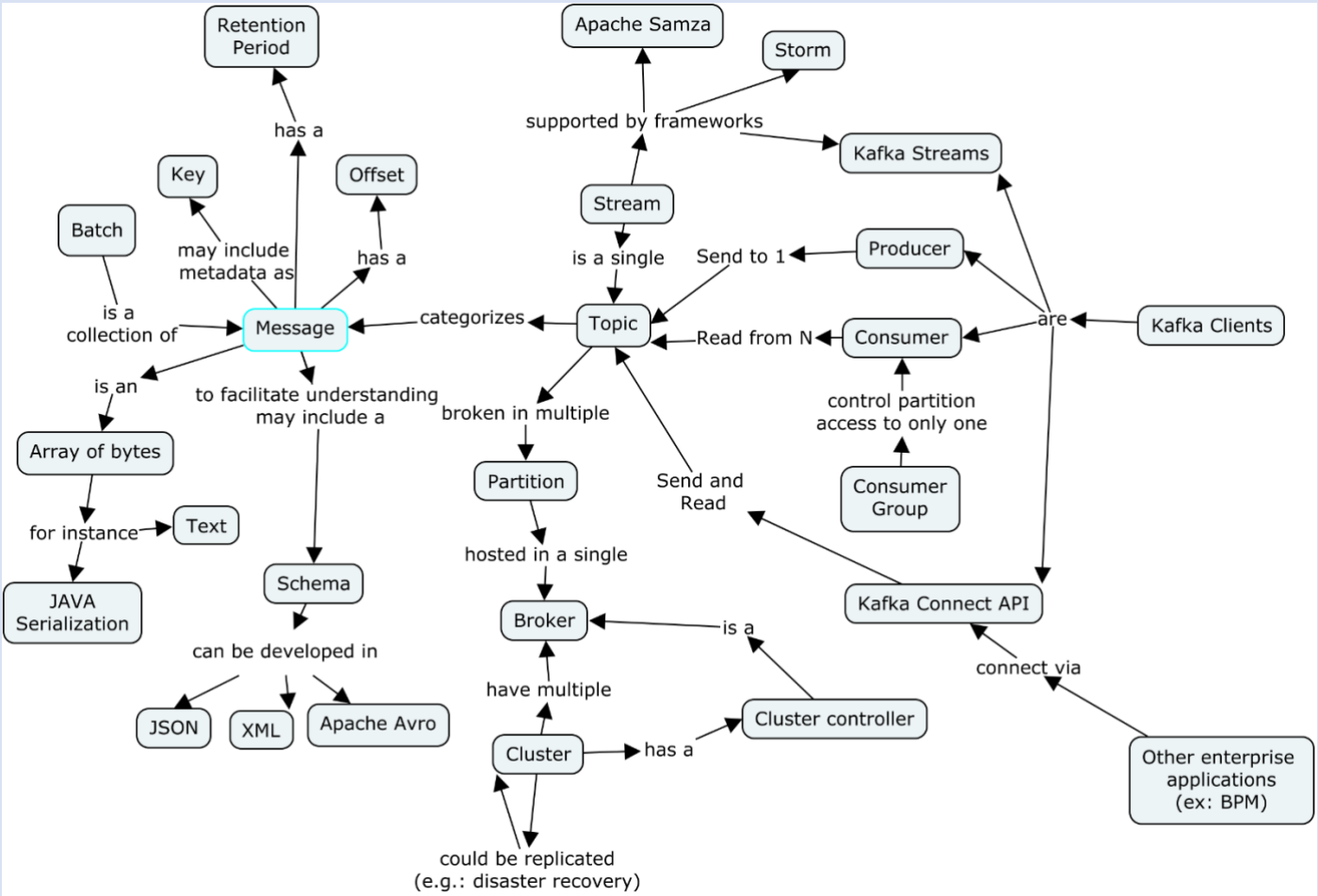
The assignment relationship represents the allocation of responsibility, performance of behavior, storage, or execution.

The specialization relationship represents that an element is a particular kind of another element.

DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to show how to operate the basics of a remote Kafka service: installation, starting, accessing, testing, stopping and backup. Remote installation step by step is presented, where the remote installation requires an Amazon AWS account. The following concepts are contained in Kafka:



The following contents is presented in this document.

Contents

A. Creating AWS Academy account and AWS private key

The goal of this section is to activate your AWS account and create your private key that will allow you to access your EC2 instance.

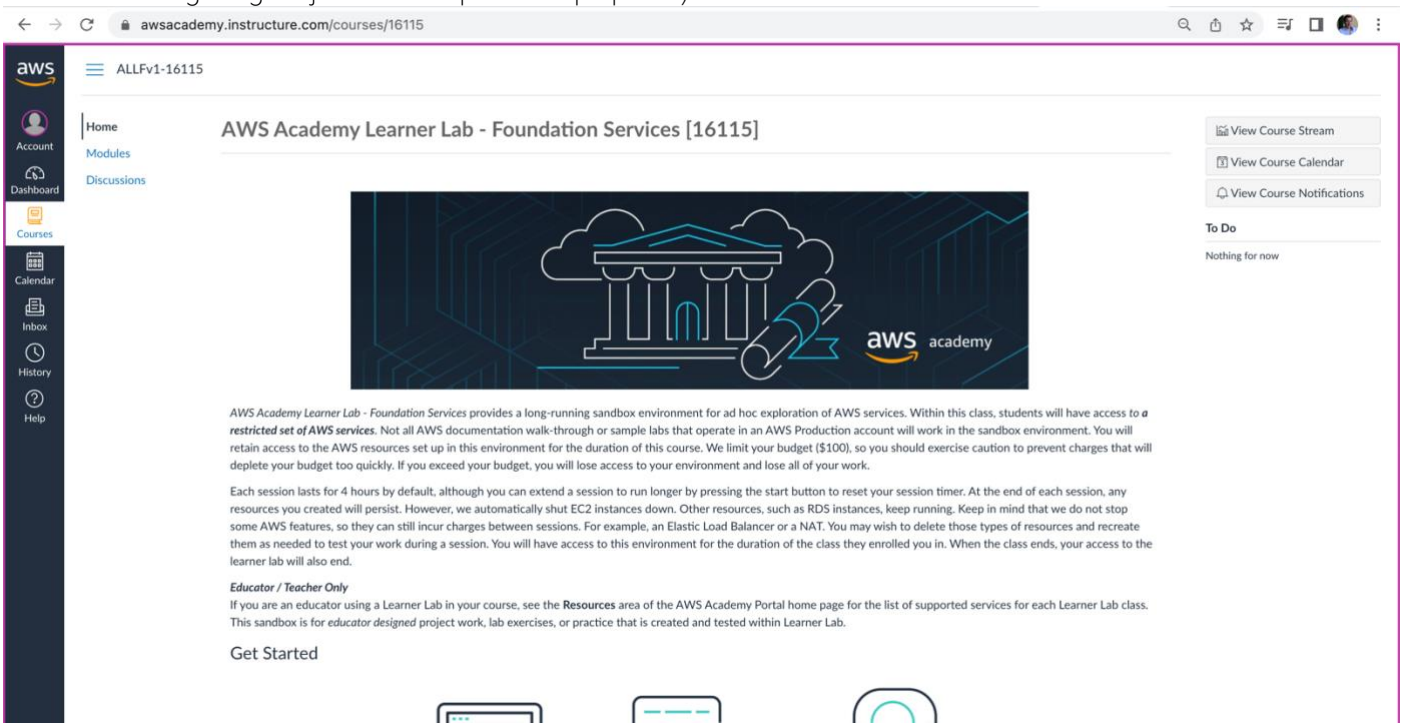
A.0. Create an amazon AWS Academy account using the invitation email sent by Amazon Academy on the faculties request.

While you are in the process of account creation be careful with your graduation date – define it, at least, accordingly with the end of semester date.

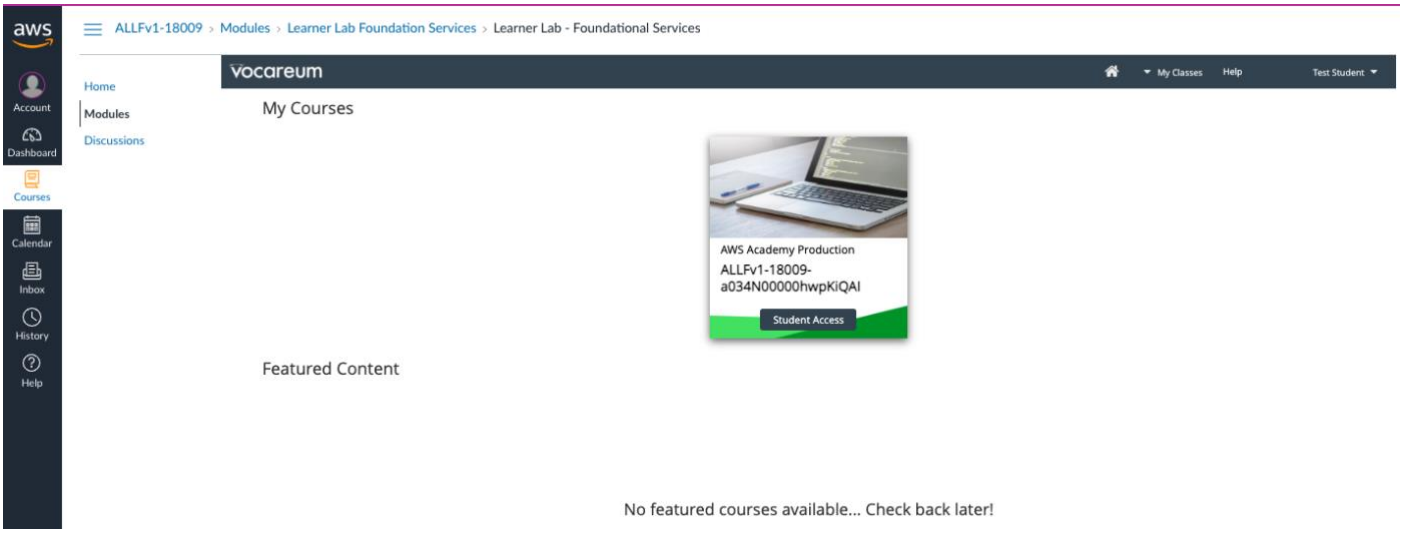
Important remark: *the usage of cloud resources implies costs. An EC2 AWS instance is a machine stored in an Amazon AWS facility that is using the usual computational resources: CPU, memory, storage, communications, energy, physical space, among others. This is a reality that any organization faces daily.*

Therefore, you should be aware that while your AWS instance is started, your AWS account also starts to be billed. We absolutely recommend you to always stop your instances when you are not working with them. Always manage your resources wisely!

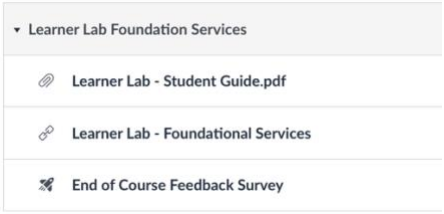
A.1. After registration, accept the terms, and then you can access your account and have a similar canvas (the following image is just for exemplification purposes).



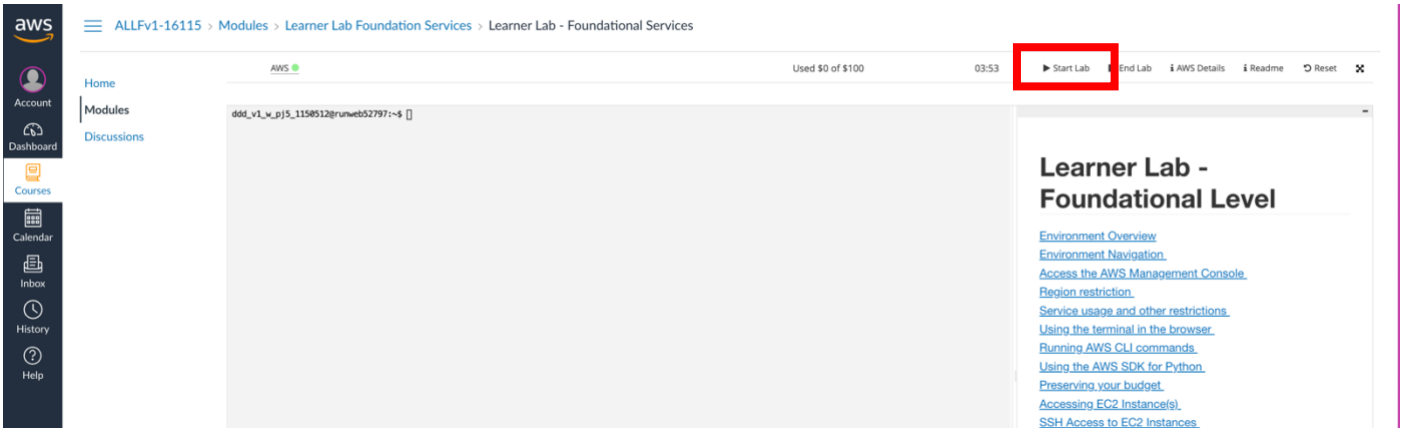
A.2. Select the available course (the following image is just for exemplification purposes):



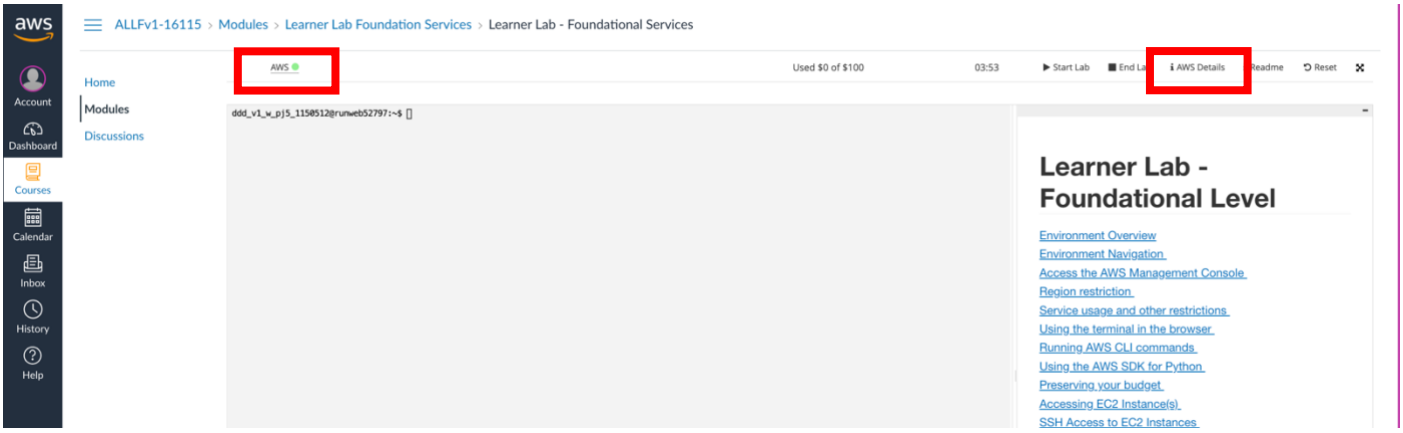
A.3. Select “Modules” and then “Learner Lab – Foundational Services”



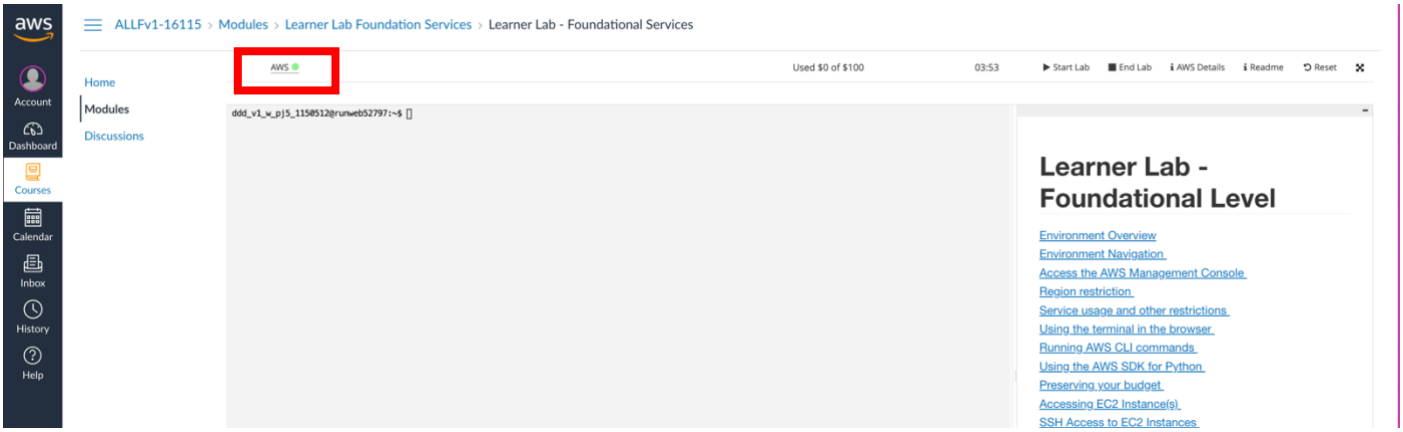
A.4. Click “**Start Lab**” on the right side



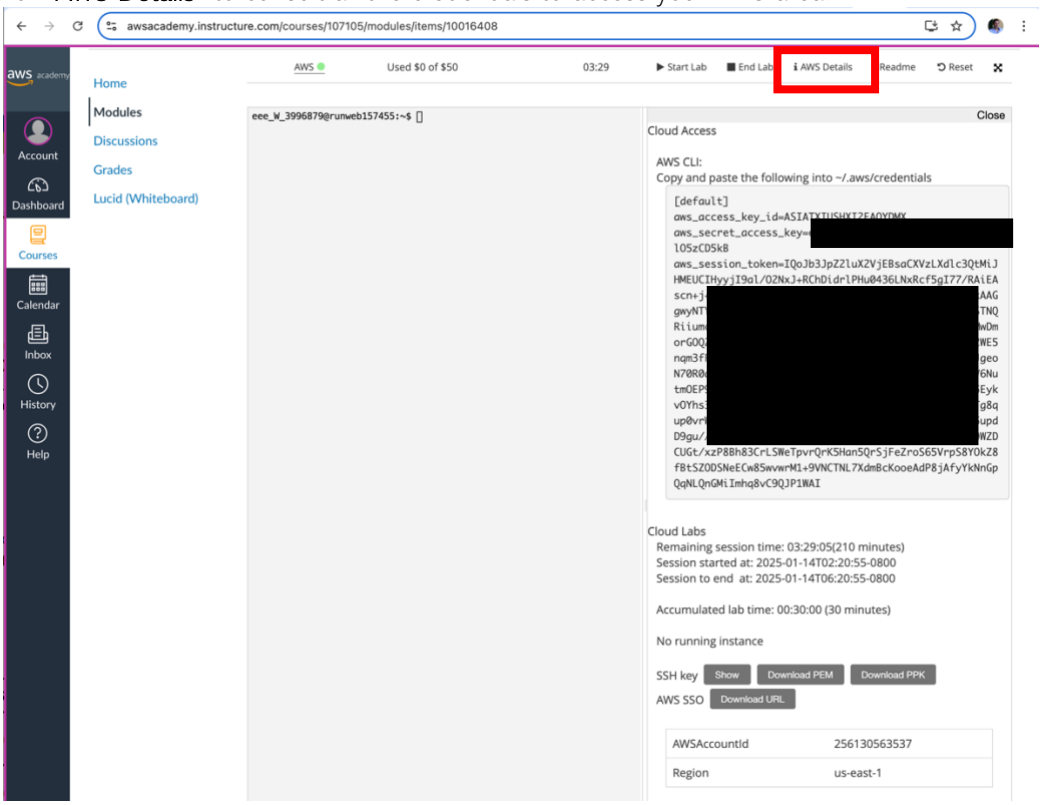
A.5. Then, when the indicator is green, click “**AWS Details**” to access AWS Management Console on the right side. As indicated by the red boxes below. Store the returned key file in your computer for later use. You can choose .pem to use with OpenSSH and/or .ppk to use with PuTTY. If you are unsure store the keys in both formats on your computer.



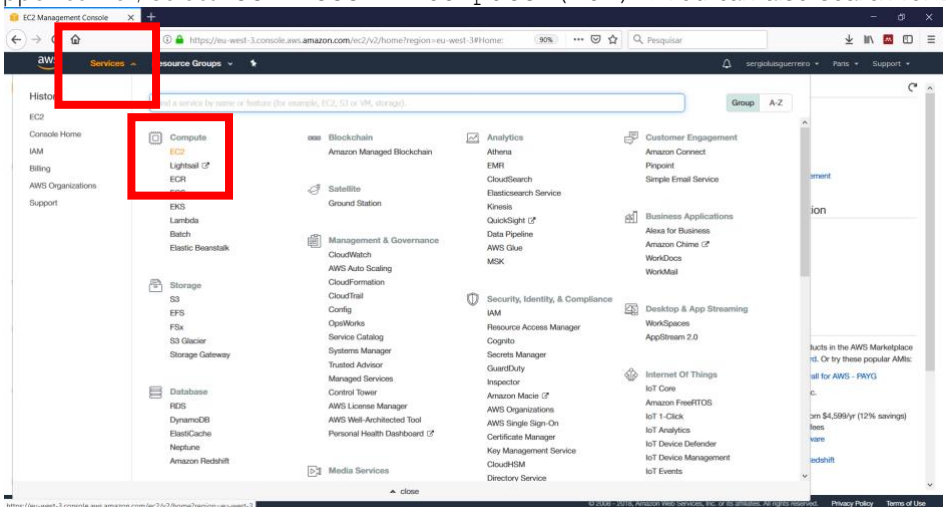
A.6. Then, click "**AWS**" to access AWS Management Console on the left side. As indicated by the red box below.



A.7. Then, click "**AWS Details**" to consult all the credentials to access your AWS area.



A.8. In the left upper corner, select "`Services -> Compute (EC2)`". *You can also search for EC2.*

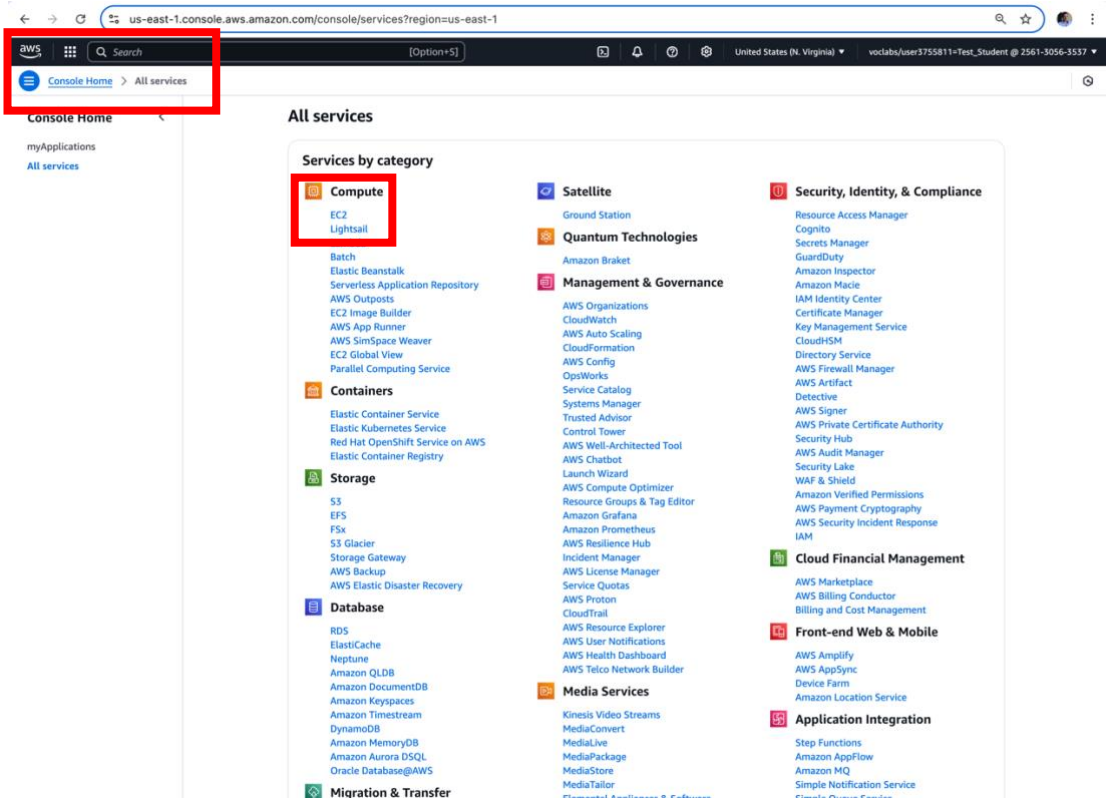


You will obtain the AWS EC2 dashboard. All the options are vertically organized in the left side.

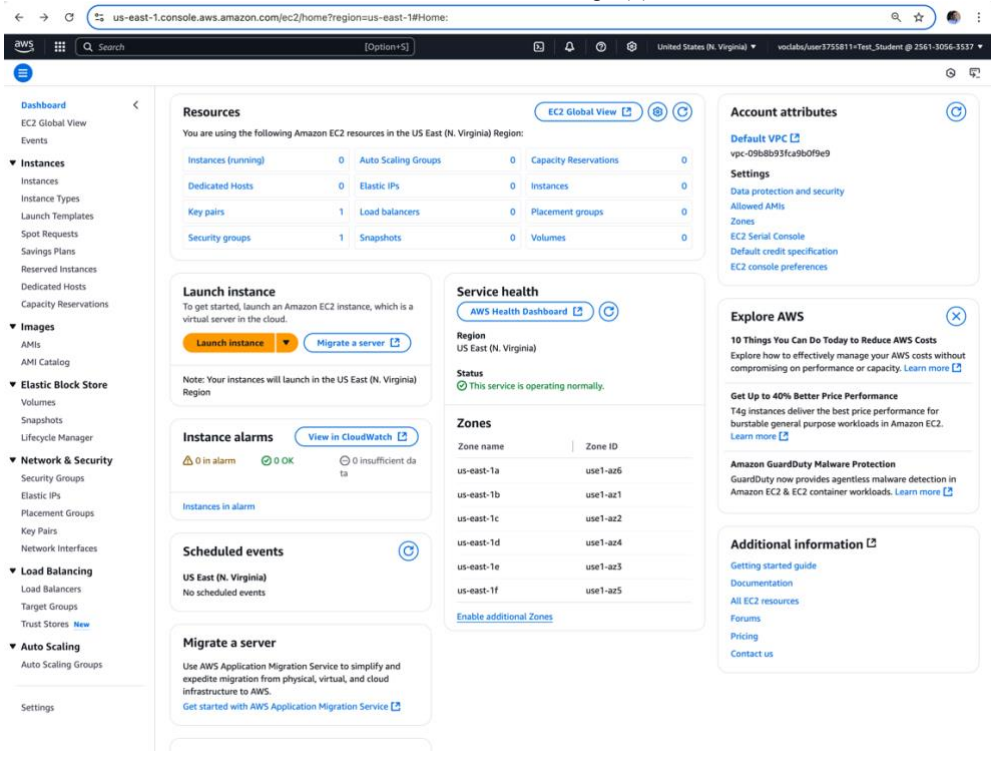
B. Creating and launching an AWS EC2 instance

The goal of this section is to create your first EC2 instance using the available Amazon Machine Images (AMI) available. This requires the previous execution of section A.

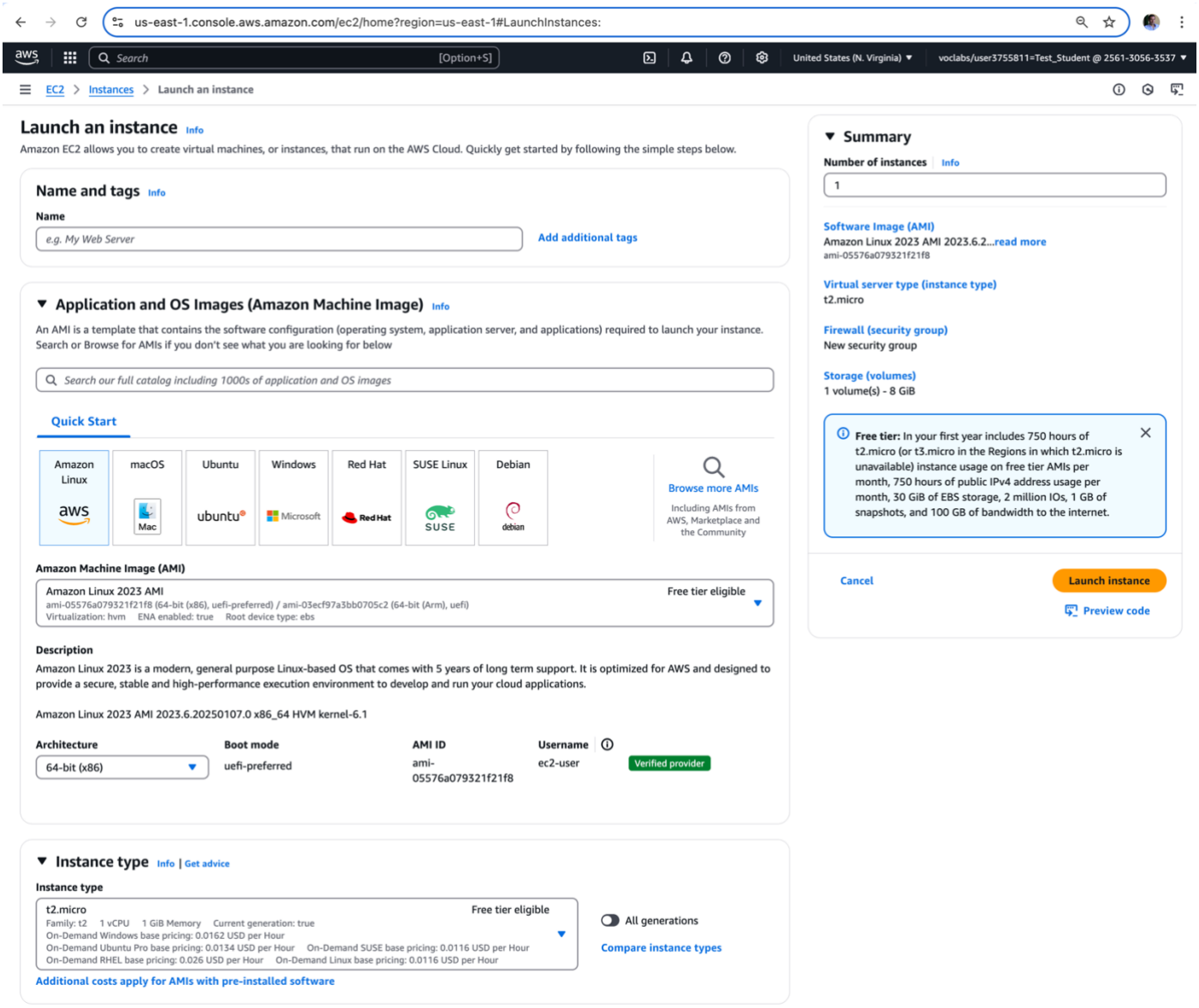
B.1. Login in your account, then access the AWS console, and in the left upper corner, select "`All Services -> Compute (EC2)`".



B.2. Select "`Instances -> Instances`". The following appearance will be shown.



B.3. Select "`Launch Instance`" and a similar interface will be presented as shown below. The goal is to define the Amazon Machine Image (AMI) that will contain all the software configuration required to launch your instance.



B.4. Use the following configurations:

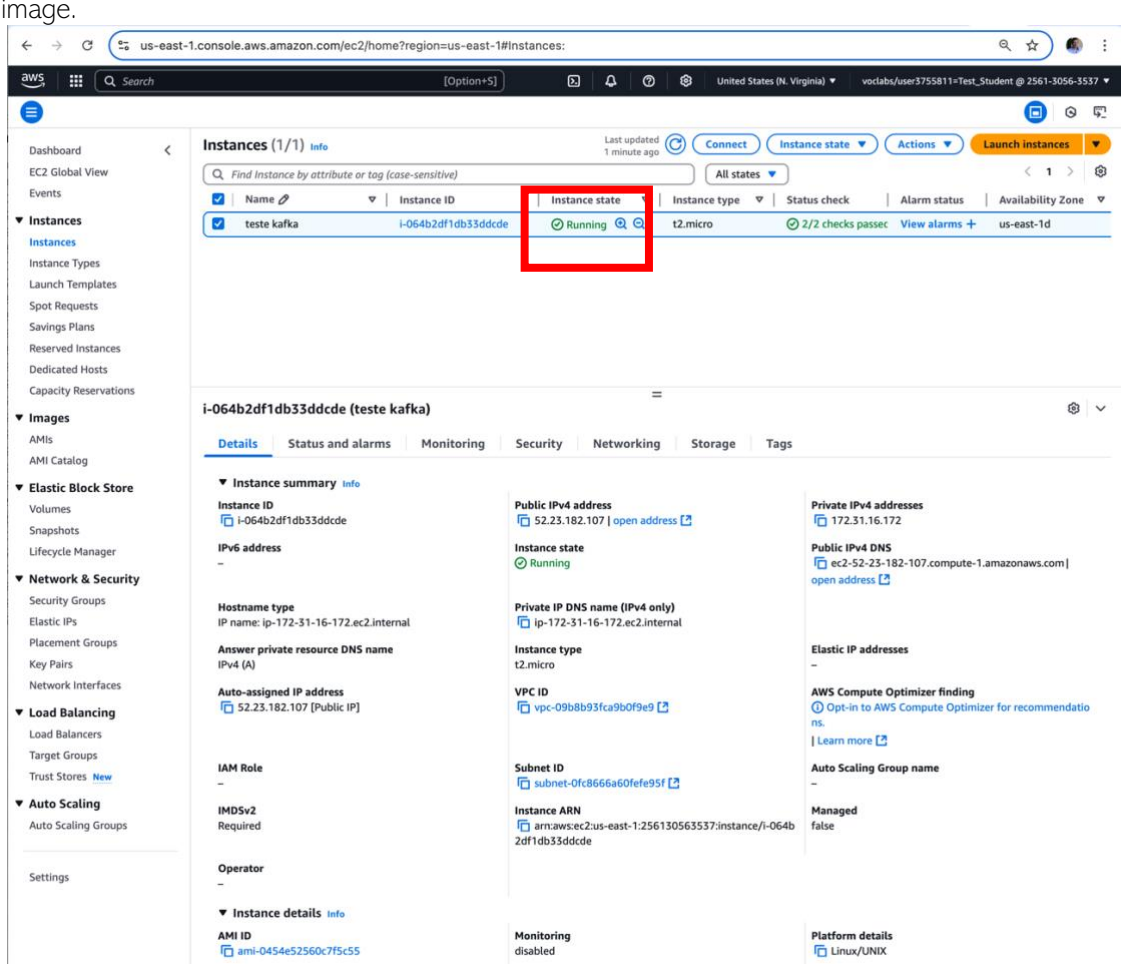
Step 1 – Choose an Amazon Machine Image (AMI)	select "Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type" (available on January 2025) with "64-bit (x86)"
Step 2 – Choose an Instance Type	select "t2.micro"
Step 3 – Configure Instance Details	check if number of instances = 1 Keep all the remaining options as default
Step 4 – Storage	keep the default size
Step 5 – Key Pair (login)	Key Pair name = vockey
Step 6 – Add Tags	Key = application Value = kafka
Step 7 – Configure Security Group *(it could be configured later in the instance console or*	Select "Create a new security group" Security group name: launch-Kafka Select "Add rule" Type = "Custom TCP Rule" Protocol = TCP Port range = 2181

since the creation using terraform)	Source = Anywhere (0.0.0.0, ::/0) Description = zookeeper Select "Add rule" Type = "Custom TCP Rule" Protocol = TCP Port range = 9092 Source = Anywhere (0.0.0.0, ::/0) Description = kafka Select "Add rule" Type = "SSH" Protocol = TCP Port range = 22 Source = Anywhere (0.0.0.0, ::/0) Description = remote access
Step 8 – Review Instance Launch	

B.5. Press "Launch" in the bottom of the page

Your AWS account starts to be billed at this point.

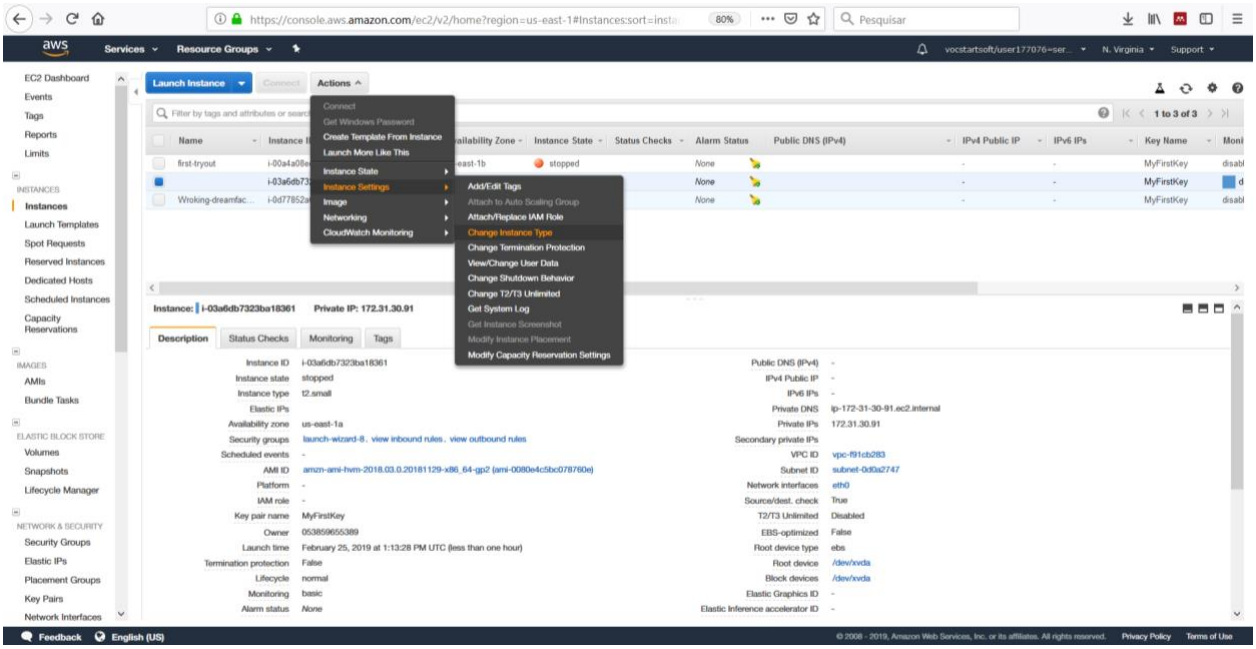
B.6. Go to EC2 dashboard as explained in A.1., and you will find an appearance similar with the following image.



You can check the IPv4 and v6 public address, and the state of the instance. In this example it is *running*.

B.7. Stop the instance, checking the row with the desired instance and then pressing “Instance state -> Stop”. Wait for the instance State to change to stopped.

B.8. To allow a proper future execution of Kafka, change the type of instance, by choosing the following options, and then choose **t2.small.**



C. Access the AWS EC2 instance using PuTTY (for windows only)

The goal of this section is to describe how your AWS EC2 instance can be accessed using an SSH client. It will be useful to install, configure and manage the Kafka server.

C.1. Install PuTTY from https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

C.2. *(optional – only if the key is a .pem file)* Convert your AWS private key created in section A, using PuTTYgen. For that, execute PuTTYgen from start menu.

C.3. *(optional – only if the key is a .pem file)* Then, in type of key to generate choose: RSA

C.4. *(optional – only if the key is a .pem file)* Press "Load" and locate your AWS private key stored in step A.4. (with .pem file extension)

C.5. *(optional – only if the key is a .pem file)* Press "Save private key" and choose "Yes" when asking to save without passphrase

C.6. *(optional – only if the key is a .pem file)* Store with the same name as the AWS private key (with .ppk file extension)

C.7. Start PuTTY from start menu

C.8. Fill hostname with "user_name@public_dns_name", where user_name = ec2-user and public_dns_name= <as presented in B.7.>

Public DNS name is changed whenever an AWS EC2 instance reboot is performed.

C.9. Choose SSH connection type and check if port = 22

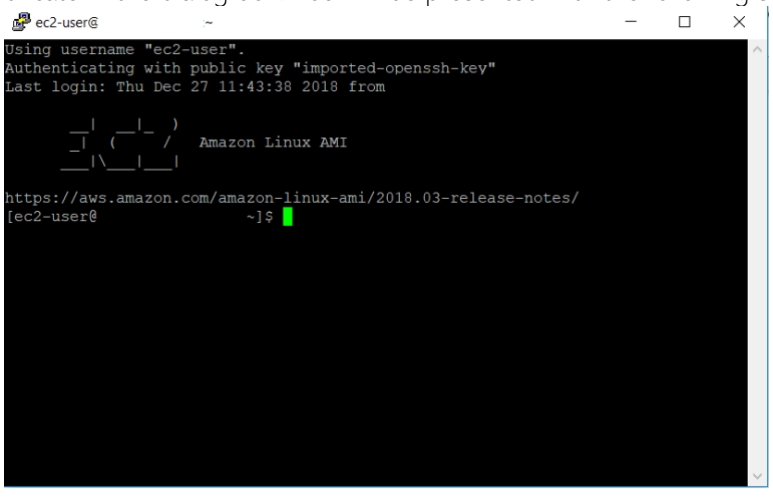
C.10. You can configure PuTTY to automatically send "keepalive" data at regular intervals to keep the session active. This is useful to avoid disconnecting the instance from session inactivity. In the Category panel, choose Connection and enter the required range in the Seconds between keepalives field. For example, if the session disconnects after 10 minutes of inactivity, enter 180 to configure PuTTY to send keepalive data every 3 minutes.

C.11. In the Category panel, choose Connection, choose SSH, choose Auth, select "Browse" and choose the .ppk file from step C.6.

Start your AWS instance.

C.12. In the Category panel, choose Session, and then select "Open"

C.13. Confirm the certificate in the dialog box. You will be presented with the following similar screen:



C.14. Update your AWS instance with the command:

```
sudo yum update
```

C.15. Check JDK packages available with the command:

```
sudo yum search "java-17"
```

C.16. Execute the command (or another equivalent package)

```
sudo yum install java-17-amazon-corretto-devel.x86_64
```

C.17. Check if JAVA is currently version 17 with the command:

```
java -version
```

If any other version is referred you may remove it, e.g., first check the installed java versions with the command:

```
yum list installed |grep java
```

and then remove the ones that you don't need with the command

```
sudo yum remove java-1.7.0-openjdk
```

C.18. Execute the command:

```
exit
```

Stop your AWS instance.

D. Access the AWS EC2 instance using ssh (for linux/macOS) and scp (for linux/macOS)

The goal of this section is to describe how your AWS EC2 instance can be accessed using an SSH client in linux. It will be useful to install, configure and manage the Kafka server.

Start your AWS instance.

Public DNS name is changed whenever an AWS EC2 instance reboot is performed.

D.1. In a new terminal session, change directories to the location of the private key file that you created when you created the AWS instance, for instance:

```
[oracle@soabpm-vm ~]$ cd /media/sf_SharedFolderForOracleSOA
```

C.19. Use the following command to set the permissions of your private key file so that only you can read it.

```
[oracle@soabpm-vm sf_SharedFolderForOracleSOA]$ chmod u=rwx,g=,o= myKeyAWS.pem
```

C.20. Use the ssh command to connect to the instance. You specify the private key (.pem) file and user_name@public_dns_name. For example, if you used Amazon Linux 2 or the Amazon Linux AMI, the user name is ec2-user. In

```
[oracle@soabpm-vm sf_SharedFolderForOracleSOA]$ ssh -i myKeyAWS.pem ec2-user@YOURIP
The authenticity of host 'X.X.X.X (X.X.X.X)' can't be established.
RSA key fingerprint is 25:1c:ef:ee:c1:87:61:e9:ea:f6:e3:45:1e:3a:63:73.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'X.X.X.X' (RSA) to the list of known hosts.
Last login: Fri Feb  8 14:59:07 2019 from Y.Y.Y.Y

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

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
5 package(s) needed for security, out of 6 available
Run "sudo yum update" to apply all updates.
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... already running as process 11211.
[ec2-user@ip-X-X-X-X ~]$
```

C.21. Update your AWS instance with the command:

```
sudo yum update
```

C.22. Check JDK packages available with the command:

```
sudo yum search "java-17"
```

C.23. Execute the command (or another equivalent package)

```
sudo yum install java-17-amazon-corretto-devel.x86_64
```

C.24. Check if JAVA is currently version 17 with the command:

```
java -version
```

If any other version is referred you may remove it, e.g., first check the installed java versions with the command:

```
yum list installed |grep java
```

and then remove the ones that you don't need with the command

```
sudo yum remove java-1.7.0-openjdk
```

C.25. Execute the command:

```
exit
```

C.26. To transfer files from your machine to your AWS instance, use the scp application in linux. For instance:

```
scp -i my-key-pair.pem /path/SampleFile.txt ec2-user@c2-198-51-100-1.compute-1.amazonaws.com:~
```

C.27. To transfer files from your AWS instance to your machine, use the scp application in linux. For instance:

```
scp -i my-key-pair.pem ec2-user@ec2-198-51-100-1.compute-1.amazonaws.com:~/SampleFile.txt ~/SampleFile2.txt
```

Stop your AWS instance.

E. Access the AWS EC2 instance using FileZilla (for windows and macOS)

The goal of this section is to describe how your AWS EC2 instance can be accessed using an SFTP client. It will be useful to transfer files between your computer and AWS EC2 instance and back.

E.1. Install FileZilla from https://filezilla-project.org/download.php?type=client
E.2. Open FileZilla.
E.3. Go to the FileZilla Edit -> settings, and on the left, click SFTP.
E.4. Add a new private key. (Your .ppk key)
E.5. If you are using a .pem key you must convert it accordingly with steps C.2. to C.6., otherwise it will not work
E.6. At the top in the Quickconnect bar (similar with the image below), put your Public DNS in the host, ec2-user, port 22 (Port 22 is SFTP rather than FTP, AWS will kick back FTP.), and NO PASSWORD.

Start your AWS instance.

E.7. Click Quickconnect.
E.8. File transfer to AWS EC2 instance is only a matter of dragging files from the left to right side.

Stop your AWS instance.

F. Installing manually Kafka in the AWS EC2 instance

The goal of this section is to describe the required steps to install the Zookeeper and Kafka servers in your AWS EC2 instance.

Start your AWS instance.

F.1. Access your AWS EC2 instance using previous steps of this tutorial.

F.2. To change for your home directory, type the command:

```
cd
```

F.3. Download the Kafka binary version from https://kafka.apache.org/downloads. Recommended version is kafka_2.13-3.9.0.tgz (in January 2025).

```
wget https://downloads.apache.org/kafka/3.9.0/kafka_2.13-3.9.0.tgz
```

F.4. Download the ZooKeeper binary version from https://zookeeper.apache.org/releases.html. Recommended version is apache-zookeeper-3.9.3-bin.tar.gz (in January 2025).

```
wget https://dlcdn.apache.org/zookeeper/zookeeper-3.9.3/apache-zookeeper-3.9.3-bin.tar.gz
```

F.5. Then, type the command:

```
ls -ltr
```

to check if you have both kafka and zookeeper files available.

F.6. Update your AWS instance with the command:

```
sudo yum update
```

F.7. Execute the command (or another equivalent package)

```
sudo yum install java-17-amazon-corretto-devel.x86_64
```

F.8. To extract the zookeeper files, type the command:

```
tar -zxf apache-zookeeper-3.9.3-bin.tar.gz
```

F.9. Move the new zookeeper directory to system typing the command:

```
sudo mv apache-zookeeper-3.9.3-bin /usr/local/zookeeper
```

F.10. Create a new directory using the command:

```
sudo mkdir -p /var/lib/zookeeper
```

F.11. Create the baseline zookeeper server configuration with the command:

```
cat > /usr/local/zookeeper/conf/zoo.cfg << EOF
```

and then write the following content to the file directly in the command line:

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
EOF
```

F.12. Start ZooKeeper server with the command:

```
sudo /usr/local/zookeeper/bin/zkServer.sh start
```

F.13. To extract the kafka server, type the command:

```
tar -zxf kafka_2.13-3.9.0.tgz
```

F.14. Move the new kafka directory to system typing the command:

```
sudo mv kafka_2.13-3.9.0 /usr/local/kafka
```

F.15. Create a new directory using the command:

```
sudo mkdir /tmp/kafka-logs
```

F.16. Start Kafka with the command:

```
sudo /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server.properties
```

Hint 1: if you are short in the instance memory reduce the JVM usage with the command:

```
export KAFKA_HEAP_OPTS="-Xmx256M -Xms256M"
```

Hint 2: you can always check if Zookeeper and Kafka servers are started correctly with the command:

```
ps -ef |grep java
```

Stop your AWS instance.

Exercise 1: To automate the start of Zookeeper and Kafka servers with instance login write both starting commands in the file:

```
.bash_profile
```

G. Testing Kafka locally in the AWS EC2 instance

The goal of this section if to test if the previous installation in section E. was executed successfully. The production and consumption of Kafka messages are tested. It is considered that in this environment the servers and the clients are installed in the same machine.

Start your AWS instance.

G.1.Access your AWS EC2 instance using the steps C.7. to C.12.
G.2.First, install telnet package in your AWS EC2 instance typing the command:

```
sudo yum install telnet.x86_64
```

G.3.Test ZooKeeper installation typing the command:

```
telnet localhost 2181
```

then write the command:

```
srvr
```

and it is expected to receive the following output:

```
[ec2-user@ ~]$ telnet localhost 2181
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
srvr
Zookeeper version: 3.4.12-e5259e437540f349646870ea94dc2658c4e44b3b, built on 03/27/2018 03:55 GMT
Latency min/avg/max: 0/0/0
Received: 1
Sent: 0
Connections: 1
Outstanding: 0
Zxid: 0x0
Mode: standalone
Node count: 4
Connection closed by foreign host.
```

G.4.Test Kafka installation, with the creation of a new topic named "test", typing the command:

```
sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test
```

G.5.Type the command to check if the topic "test" is created successfully:

```
sudo /usr/local/kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic test
```

G.6.Type the command to produce messages to the topic "test":

```
sudo /usr/local/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

And write some text messages, one per line. Then, to finish press Ctrl+D.

G.7.Type the command to consume messages from the topic "test":

```
sudo /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning
```

All the sent messages from topic "test" will be presented. Then, to finish press Ctrl+C.

Exercise 2: Try creating different PuTTY sessions for producer and consumer and check the run-time production and consumption of messages.

Stop your AWS instance.

H. Testing Kafka in the AWS EC2 instance remotely

The goal of this section if to test if the previous installation in section E. was executed successfully. The production and consumption of Kafka messages are tested. It is considered that in this environment the servers and the clients are installed in different machines.

This section requires the previous installation of Kafka, as described in section E.

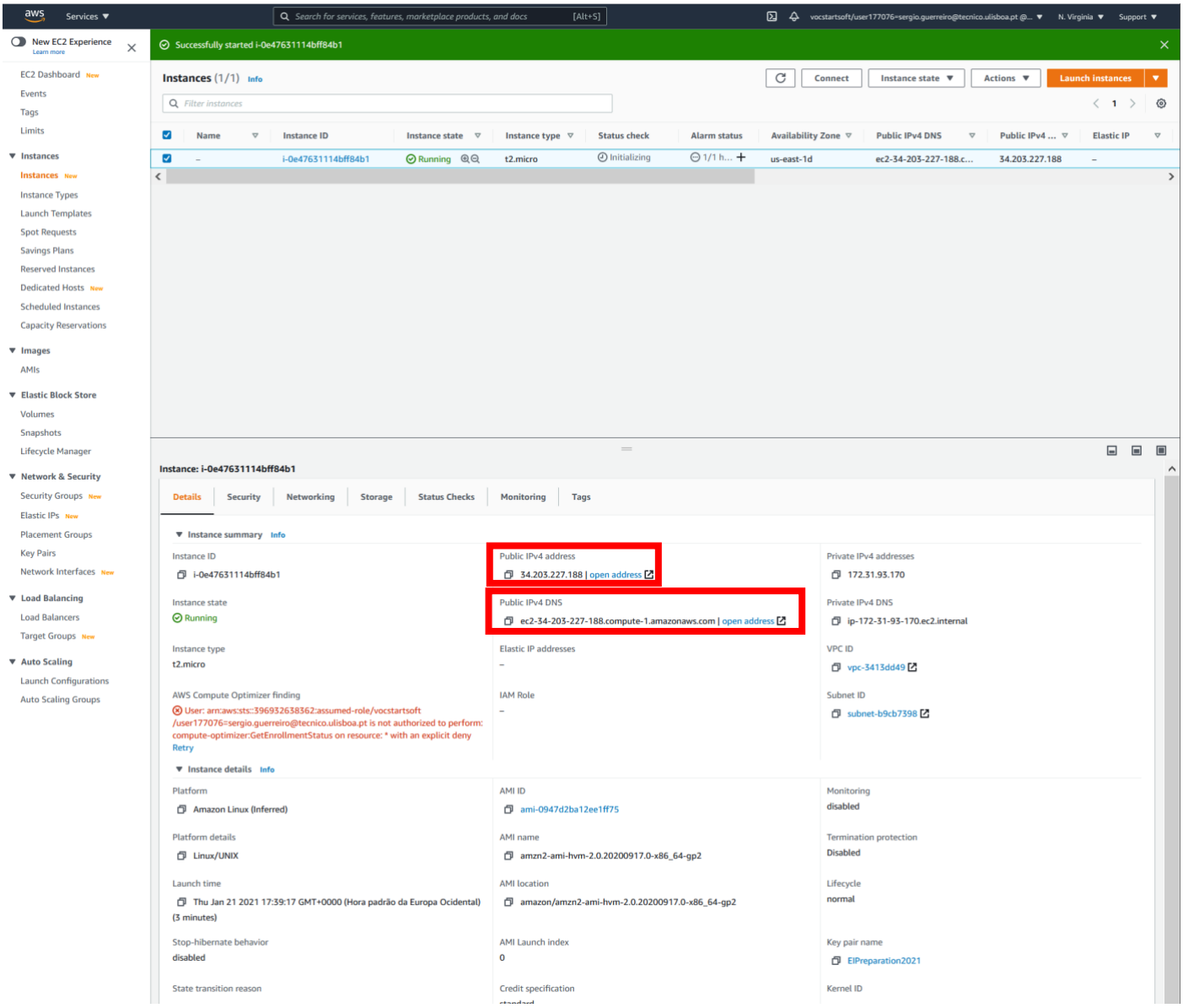
Start your AWS instance.

H.0. Access your AWS EC2 instance using the steps C.7. to C.12.

H.1. From your local docker container, from local macOS, local linux, or other AWS EC2 linux environment, check if you can connect to the AWS EC2 instance Zookeeper server using the following command:

```
telnet YourPublicIP 2181
```

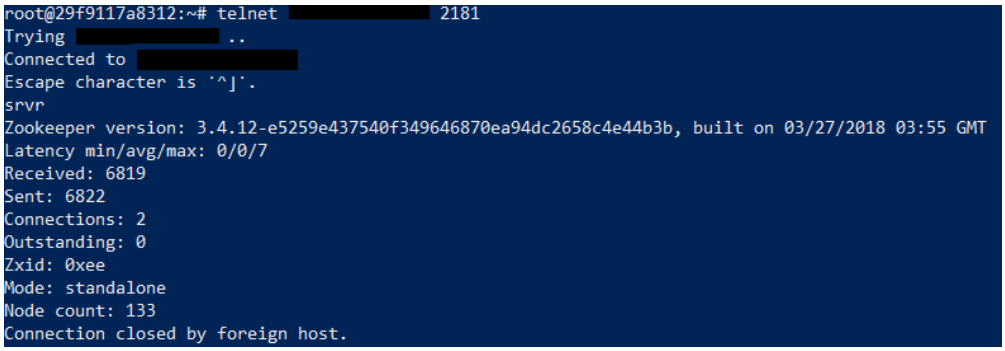
remember that *YourPublicIP* can be found in the AWS EC2 dashboard as presented below:



After that, send the command:

```
srvr
```

You should be able to see an interaction similar with the following:



Hint 3: Public IP or Public DNS Name could be.

H.2. Using the FileZilla, download the file /usr/local/kafka/config/server.properties from your AWS EC2 instance.

H.3. Edit the previous file adding the following uncommented text. Where *Your Public DNS Name* can be found in the AWS EC2 dashboard as presented above.

```
…
############################# Socket Server Settings #############################

# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
#listeners=PLAINTEXT://:9092
listeners=PLAINTEXT://Your Public DNS Name:9092
…
```

Hint 4: Public DNS name is changed whenever an AWS EC2 instance reboot is performed, therefore reconfiguration of this file could be needed often.

H.4. Save the file and upload, using FileZilla, for your AWS EC2 instance in /usr/local/kafka/config/server.properties.

Hint 5 regarding H.2., H.3. and H.4.: another option to change the content of the server.properties file is using the vi application directly in the AWS EC2 environment.

H.5. Stop Kafka server using the command:

```
sudo /usr/local/kafka/bin/kafka-server-stop.sh -daemon /usr/local/kafka/config/server.properties
```

H.6. Check that Kafka server is stopped with:

```
ps -ef |grep java|grep kafka
```

H.7. Restart Kafka with the command, as in F.15.:

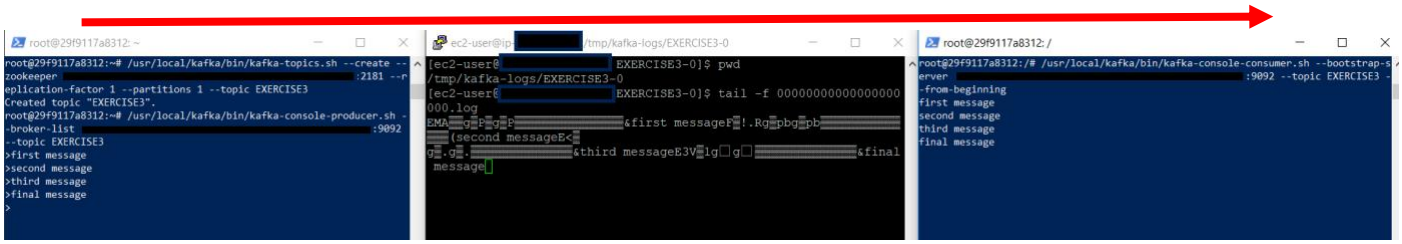
```
sudo /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server.properties
```

H.8. To test the connection, from your local docker container, from local macOS, local linux, or other AWS EC2 linux environment, type the command to check if the topic "test" is created successfully in the AWS EC2 instance:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server YourPublicDNSName:9092 --describe --topic test
```

H.9. You are now able to communicate with the AWS EC2 instance. Any other command from section G. can now be tested.

Exercise 3: Create an environment where the messages are produced from one AWS EC2 machine, sent to Kafka server in AWS EC2 instance, and then consumed by another AWS EC2 machine. Check if the flow of messages is executing correctly. Screenshot of the solution:

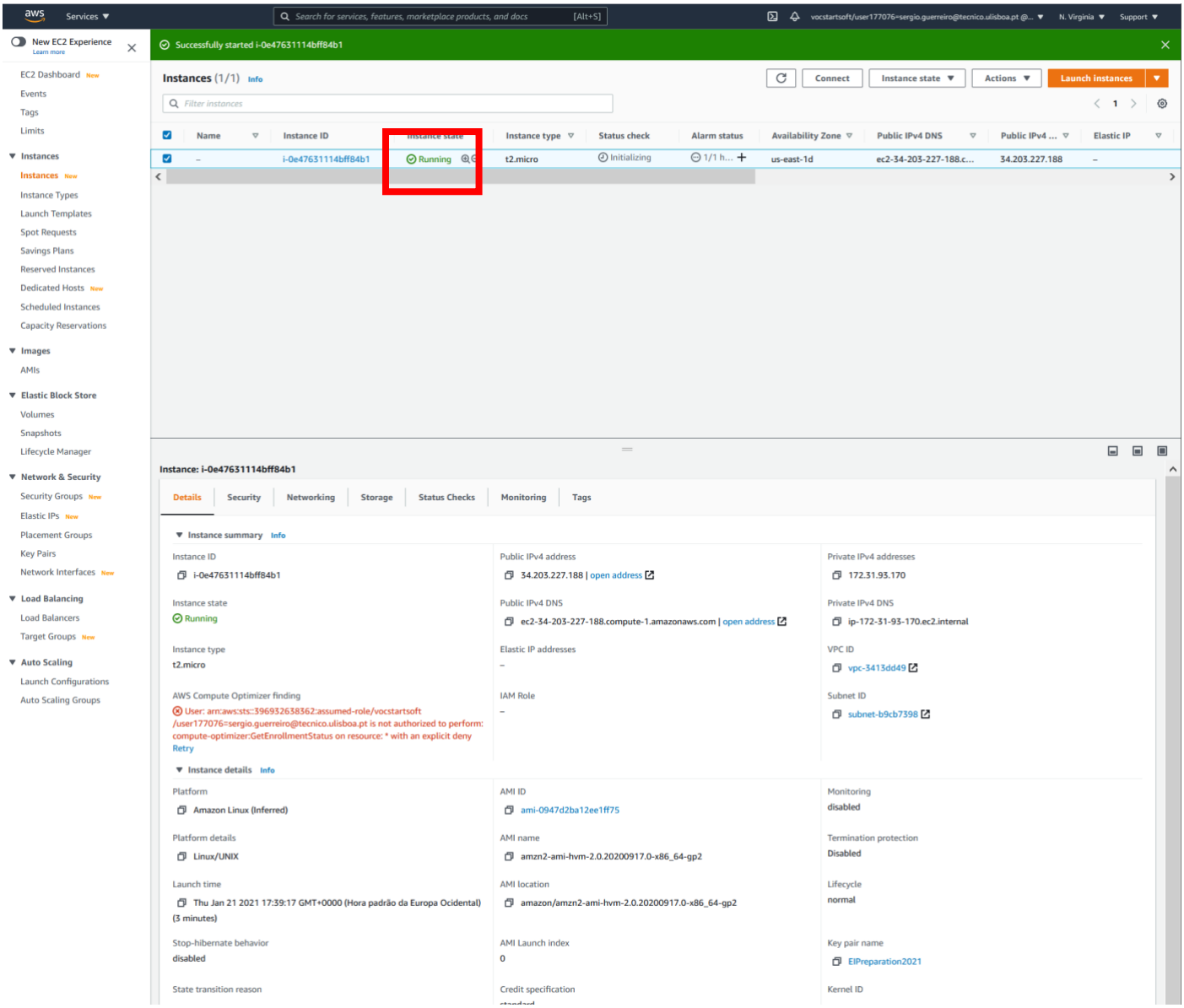


Stop your AWS instance.

I. Stopping the AWS EC2 instance

The goal of this section is to describe how an AWS EC2 instance can be stopped. This section requires the execution of section B.

I.1. Go to AWS EC2 dashboard as explained in A.1., and you will find an appearance similar with the following image.



I.2. Stop the instance, checking the row with the desired instance and then pressing “Instance state -> Stop instance state”. Wait for the instance State to change to stopped.

Other references

- Connecting to Your Linux Instance Using SSH, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html
- 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
- Where to download apache maven https://maven.apache.org/download.cgi
- How to install apache maven https://maven.apache.org/install.html

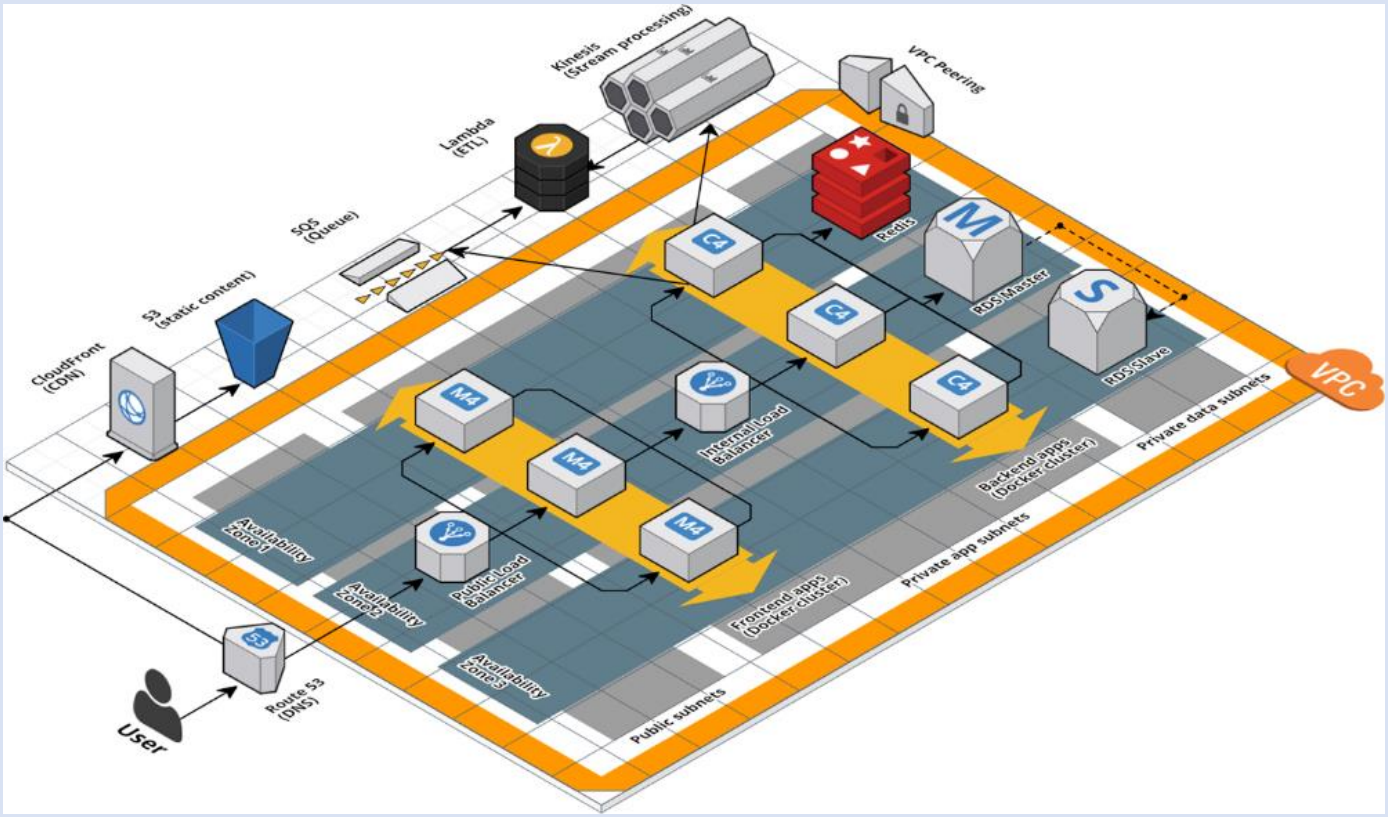
DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

HyperAutomation using Terraform: Infrastructure as Code

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to present the details related with Terraform. How to use it using a step-by-step approach. Terraform is a solution to create cloud environments using code instead of using the cloud providers User Interfaces. Therefore, the management of the cloud resources takes less effort and the creation, or update, process is faster.

The following figure taken the main literature reference in this field: *Terraform Up & Running, Writing Infrastructure as Code, Yevgeniy Brikman, 3rd edition (2022)"* exemplifies a complex infrastructure that could be created using Terraform.

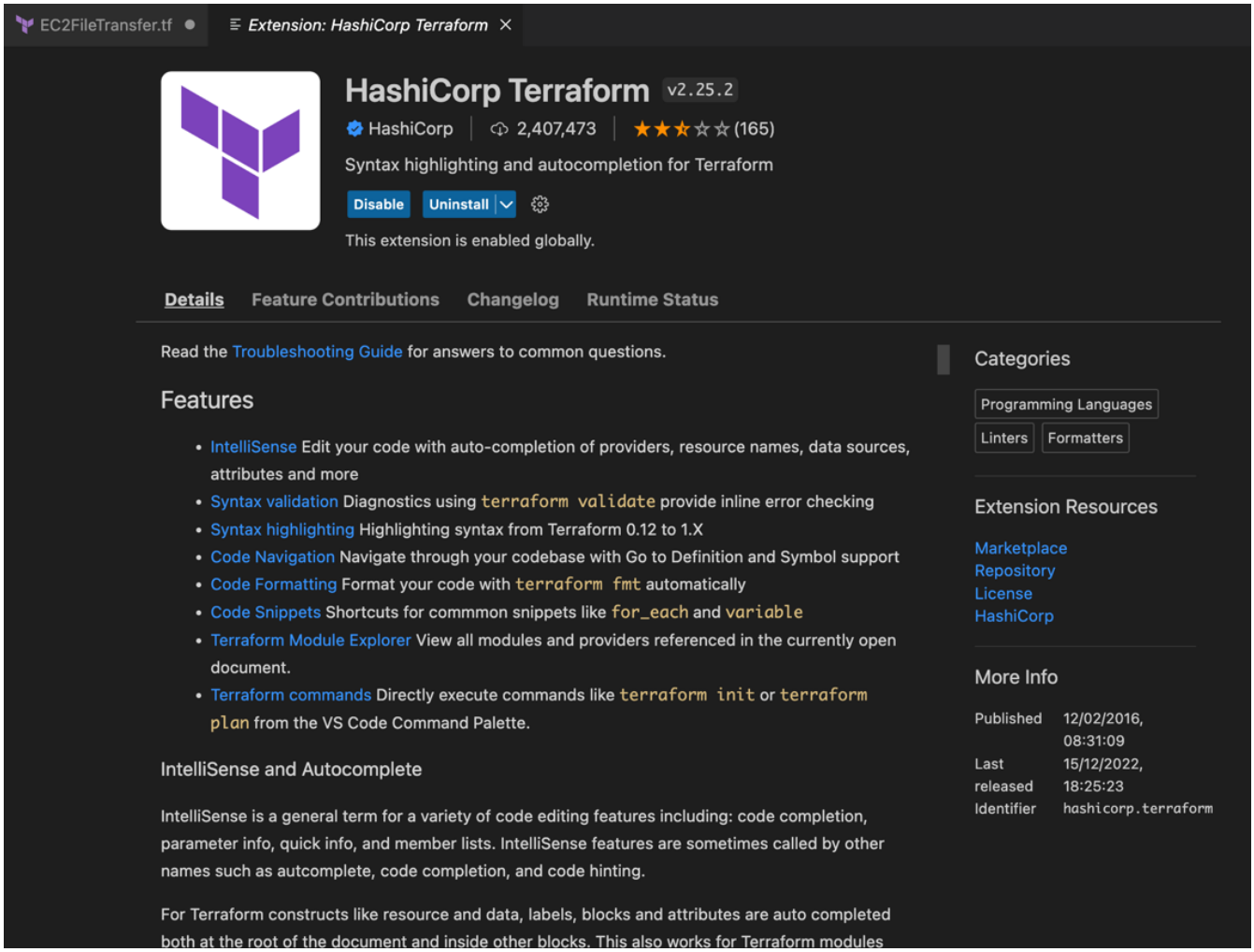


Contents

A. Terraform installation procedure

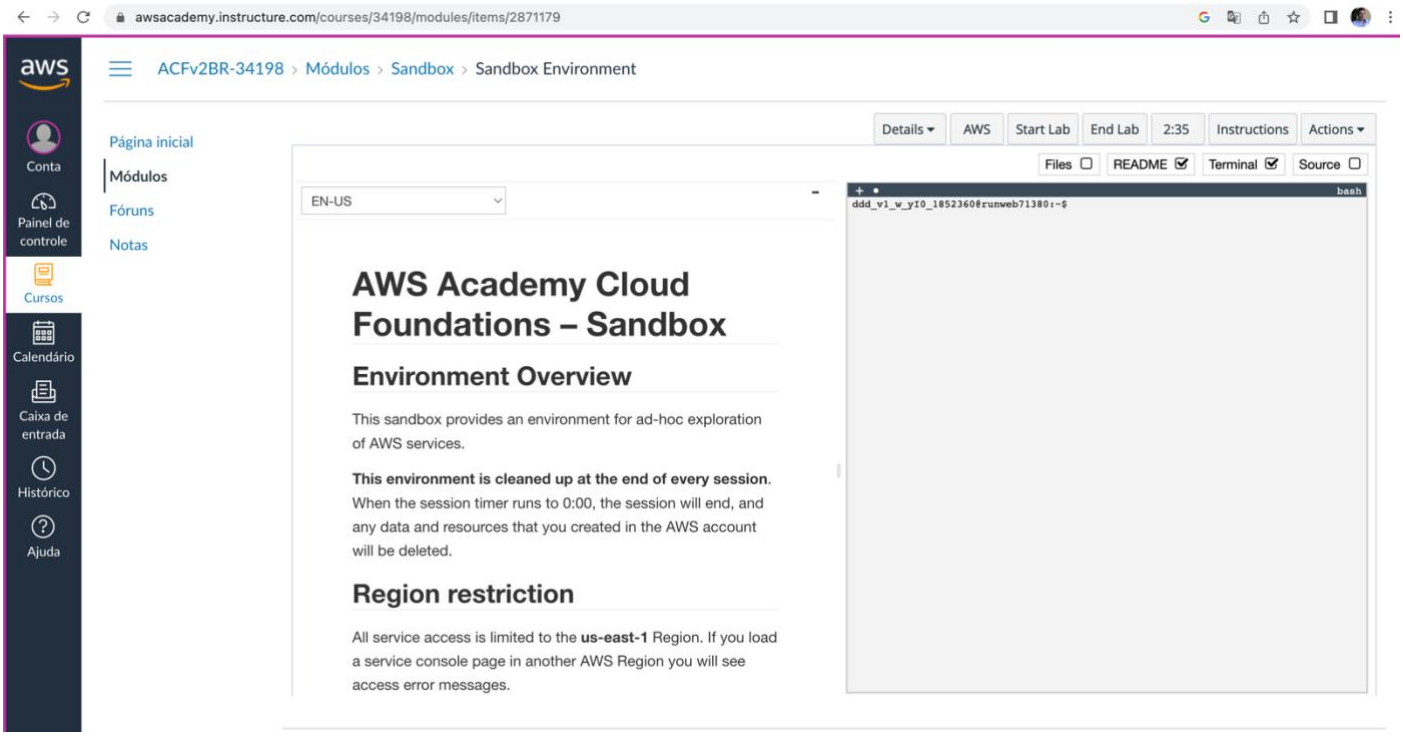
Use the URL https://developer.hashicorp.com/terraform/downloads and choose the package appropriate for your operating system and then follow the indications.

For better Terraform code rendering, you can optionally, install the Terraform extension for VSCode.

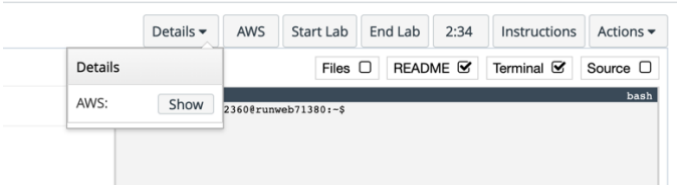


B. Obtaining the access for sandbox or AWS academy learner lab

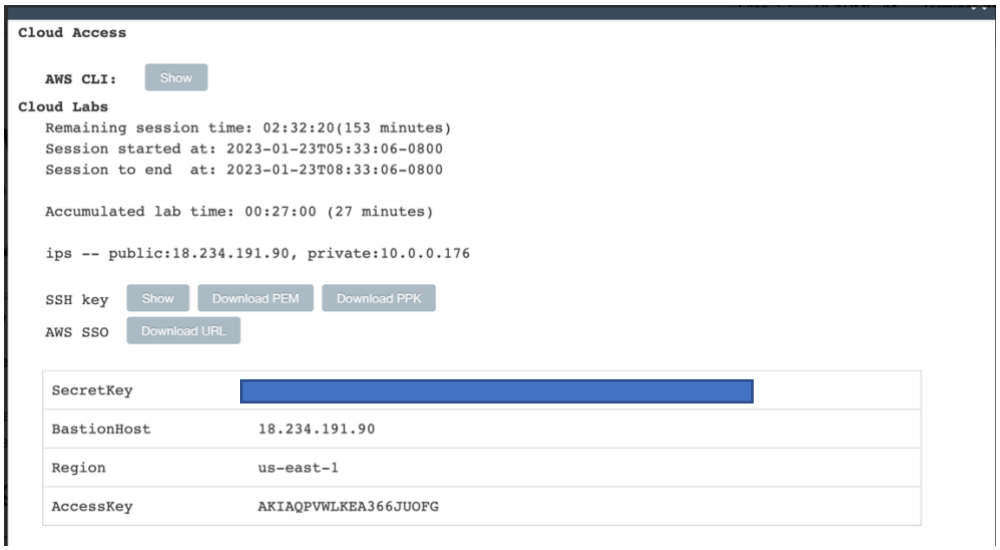
B.1. Choose your learner lab, then start it.

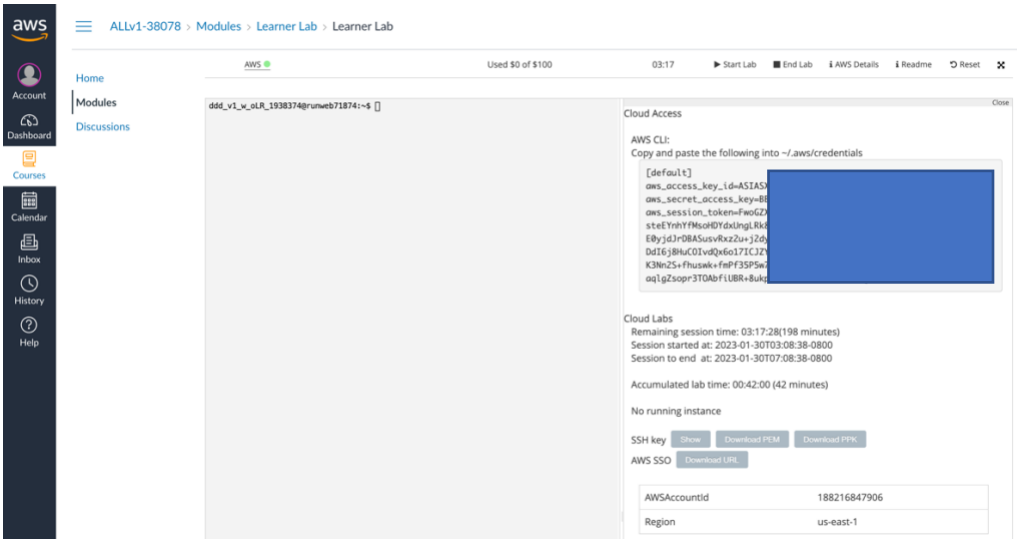


B.2. When the lab is started, select **Details**, and then **Show**:



B.3. A similar access configuration is available for you (they are both examples that could be available, the first one is secretKey/accessKey, and the second is secretKey/accessKey/token). You will have to adapt your terraform calls accordingly with the provided AWS access.




ALLv1-38078 › Modules › Learner Lab › Learner Lab
Home
Modules
Discussions
Used $0 of $100
03:17
Start Lab
End Lab
AWS Details
Readme
Reset
Cloud Access
AWS CLI:
Copy and paste the following into ~/.aws/credentials
[default]
Cloud Labs
Remaining session time: 03:17:28(198 minutes)
Session started at: 2023-01-30T03:08:38-0800
Session to end at: 2023-01-30T07:08:38-0800
Accumulated lab time: 00:42:00 (42 minutes)
No running instance
SSH key
Show
Download PEM
Download PPK
AWS SSO
Download URL
AWSAccountId
188216847906
Region
us-east-1

C. Deploying a single EC2 instance

C.1. Configure the AWS access inside the terraform script file. For that, create the following script, replacing the **access_key**, **secret_key**, **token**, **region**, the **ami**, and the **instance_type** accordingly with your needs. Save the file in text format with a **.tf** extension.

```
terraform {
  required_version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
  access_key  = "YOUR_ACCESS_KEY"
  secret_key  = "YOUR_SECRET_KEY"
  token       = "YOUR_TOKEN"
}

resource "aws_instance" "example" {
  ami           = "ami-045269a1f5c90a6a0"
  instance type = "t2.micro"

  tags = {
    Name = "terraform-example"
  }
}
```

C.2. Deploy the code with the following commands. You should use only one .tf file in each directory. A local configuration is created and then reflected in the cloud environment:

```
terraform init
terraform apply
```

or sequentially:

```
terraform init && terraform apply
```

C.3. You can verify the configuration of the current configuration using the following command:

```
terraform show
```

C.4. Clean up when you're done:

```
terraform destroy
```

Hint: to facilitate the terraform command you can also use the `-auto-approve` *option*

D. Deploying a single EC2 instance and uploading files

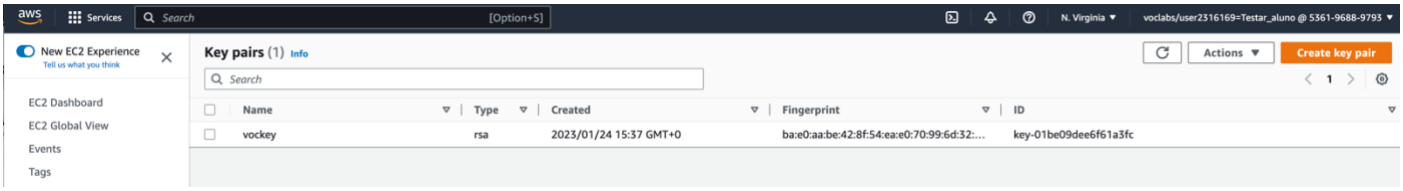
D.1. As previous, configure your AWS access keys as environment variables, and then, create the following .tf file, adapting the **region**, the **ami**, and the **instance_type** accordingly with you needs.

Notice that **access_key**, **secret_key** and **token** could be directly written in the source code.

Also get available the kafka tgz file and test.pem on your local path.

```
terraform {
  required version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region       = "us-east-1"
  access_key   = "YOUR_ACCESS_KEY"
  secret_key   = "YOUR_SECRET_KEY"
  token        = "YOUR_TOKEN"
}

resource "aws_instance" "exampleFileTransfer" {
  ami                         = "ami-045269a1f5c90a6a0"
  instance_type               = "t2.micro"
  vpc_security_group_ids      = [aws_security_group.instance.id]
  key_name                    = "vockey"
  tags = {
    Name = "terraform-example"
  }

  connection {
    type        = "ssh"
    user        = "ec2-user"
    private_key = file("test.pem")
    host        = "${self.public_dns}"
  }

  provisioner "file" {
      source      = "kafka_2.13-3.9.0.tgz"
      destination = "/home/ec2-user/kafka_2.13-3.9.0.tgz"
    }
}

resource "aws security group" "instance" {
  name = var.security_group_name
  ingress {
    from port   = 22
    to port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-example-instance"
}
```

Where **key_name** is obtained from the name of the keypairs available directly from the AWS console:



The **provisioner "file"** tag can be repeated as many times as needed, e.g.:

```
provisioner "file" {
  source      = "kafka_2.13-3.6.2.tgz"
  destination = "/home/ec2-user/kafka_2.13-3.6.2.tgz"
}

provisioner "file" {
  source      = "/root/zookeeper/apache-zookeeper-3.8.4-bin.tar.gz"
  destination = "/home/ec2-user/apache-zookeeper-3.8.4-bin.tar.gz"
}
```

Take attention to the **security resource** that allows SSH connections to your instance. If any other ingress is needed, you can add more in the same .tf file.

Also, the connection provides the **connection** configuration to enable the file upload (it uses a scp mechanism).

Also take note that provisioner is last action taken.

D.2. After changing for your configuration. Try it. Deploy the code:

```
terraform init
terraform apply
```

or

```
terraform init && terraform apply
```

D.3. Check if the file is uploaded correctly using a SSH connection directly to the EC2 instance:

```
ssh -i test.pem ec2-user@YOUR_DNS_NAME
```

To remember: Use the following command to set the permissions of your private key file so that only you can read it.

```
chmod u=rwx,g=,o= myKeyAWS.pem
```

D.4. Clean up when you're done:

```
terraform destroy
```

E. Sending commands during boot of a single EC2 instance

User_data is a shell script or cloud-init directive that will be executed during **instance creation only**.

The result of the **user_data** execution can be seen in EC2 Console (select the Instance, click Actions -> Monitor and troubleshoot -> Get system log) and you can find its execution log on the EC2 Instance itself (typically in /var/log/cloud-init*.log), both of which are useful for debugging, and neither of which is available with provisioners.

E.1. As previous, configure your AWS access keys as environment variables, and then, create the following .tf file, adapting the **region**, the **ami**, and the **instance_type** accordingly with you needs.

```
terraform {
  required_version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region      = "us-east-1"
  access key  = "YOUR_ACCESS_KEY"
  secret key  = "YOUR_SECRET_KEY"
  token       = "YOUR_TOKEN"
}

resource "aws_instance" "exampleKAFKA" {
  ami                     = "ami-045269a1f5c90a6a0"
  instance type           = "t2.small"
  vpc security group ids  = [aws security group.instance.id]
  key_name                = "vockey"

  user_data = "${file("creation.sh")}"

  user_data_replace_on_change = true

  tags = {
    Name = "terraform-example-kafka"
  }
}

resource "aws_security_group" "instance" {
  name = var.security_group_name
  ingress {
    from port   = 22
    to port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
    ingress {
    from port   = 2181
    to_port     = 2181
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
    ingress {
    from_port   = 9092
    to_port     = 9092
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr blocks      = ["0.0.0.0/0"]
    ipv6 cidr blocks = ["::/0"]
  }
}
```

```
variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-example-instance"
}
```

Where the script **user_data = "${file("creation.sh")}"** is a local file that could contain some work that you need to perform.
For instance, installing and starting ZooKeeper and Kafka, automatically!

```
#!/bin/bash
echo "Starting..."
cd
sudo wget https://dlcdn.apache.org/zookeeper/zookeeper-3.9.3/apache-zookeeper-3.9.3-bin.tar.gz
sudo tar -zxf apache-zookeeper-3.9.3-bin.tar.gz
sudo mv apache-zookeeper-3.9.3-bin /usr/local/zookeeper
sudo mkdir -p /var/lib/zookeeper
echo "tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181" > /usr/local/zookeeper/conf/zoo.cfg

sudo yum -y install java-17-amazon-corretto-devel.x86_64

sudo /usr/local/zookeeper/bin/zkServer.sh start

sudo wget https://downloads.apache.org/kafka/3.9.0/kafka_2.13-3.9.0.tgz
sudo tar -zxf kafka 2.13-3.9.0.tgz
sudo mv kafka 2.13-3.9.0 /usr/local/kafka
sudo mkdir /tmp/kafka-logs

sudo /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server.properties

echo "Finished."
```

The **egress** tag allows the EC2 instance to outbound to another machine in internet.
Recall that a provisioner is the last action taken. That is why the binary files for Zookeeper and Kafka are being downloaded instead of being received by a provisioner.

If needed, the logging of the EC2 instance provisioning can be consulted at /var/log/cloud-init-output.log

F. Change the listener of a Kafka broker deployed with terraform

F.1. Use the previous terraform file for deploying kafka broker in AWS.

```
terraform {
  required version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region      = "us-east-1"
  access_key  = "YOUR_ACCESS_KEY"
  secret_key  = "YOUR_SECRET_KEY"
  token       = "YOUR_TOKEN"
}

resource "aws_instance" "exampleKAFKAConfigured" {
  ami                     = "ami-045269a1f5c90a6a0"
  instance_type           = "t2.small"
  vpc_security_group_ids  = [aws_security_group.instance.id]
  key_name                = "vockey"

  user data = "${file("creation.sh")}"

  user_data_replace_on_change = true

  tags = {
    Name = "terraform-example-kafka2"
  }
}

resource "aws_security_group" "instance" {
  name = var.security_group_name
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr blocks = ["0.0.0.0/0"]
  }
    ingress {
    from_port   = 2181
    to_port     = 2181
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
    ingress {
    from_port   = 9092
    to_port     = 9092
    protocol    = "tcp"
    cidr blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port        = 0
    to port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-example-instance"
}
```

F.2. Then, the script for provisioning the kafka broker need to be adapted with the new configuration for the listener at file /usr/local/kafka/config/server.properties. For that the name of the EC2 instance need to be known. The instance metadata service available at https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html could be used for that purpose.

```
#!/bin/bash

echo "Starting..."
cd
sudo wget https://dlcdn.apache.org/zookeeper/zookeeper-3.9.3/apache-zookeeper-3.9.3-bin.tar.gz
sudo tar -zxf apache-zookeeper-3.9.3-bin.tar.gz
sudo mv apache-zookeeper-3.9.3-bin /usr/local/zookeeper
sudo mkdir -p /var/lib/zookeeper
echo "tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181" > /usr/local/zookeeper/conf/zoo.cfg

sudo yum -y install java-17-amazon-corretto-devel.x86_64

sudo /usr/local/zookeeper/bin/zkServer.sh start

sudo wget https://downloads.apache.org/kafka/3.9.0/kafka_2.13-3.9.0.tgz
sudo tar -zxf kafka_2.13-3.9.0.tgz
sudo mv kafka_2.13-3.9.0 /usr/local/kafka
sudo mkdir /tmp/kafka-logs

ip=`curl http://169.254.169.254/latest/meta-data/public-hostname`
sudo sed -i "s/#listeners=PLAINTEXT:\/\/:9092/listeners=PLAINTEXT:\/\/$ip:9092/g" /usr/local/kafka/config/server.properties

# due to AWS network stablishment process, check if 60 seconds is enough for your situation
(sleep 120 && sudo /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server.properties )&

echo "Finished."
```

G. Adding more Kafka brokers to cluster, and configuring each one individually

Solving this problem in small solutions:

- **First solution:** simply add the count element to your aws instance. When you execute the command "terraform apply", those correspondingly number of similar instances will be created. It's a fast and easy way of provision multiple EC2 instances. However, all the internal detailed configurations of the installed software need to be done manually EC2 instance by EC2 instance: which at scale could be unfeasible.

```
resource "aws_instance" "exampleCLUSTER" {
  ami                    = "ami-045269a1f5c90a6a0"
  instance type          = "t2.small"

  count = 4

  vpc_security_group_ids  = [aws_security_group.instance.id]
  key_name                = "vockey"

  user_data = "${file("creation.sh")}"

  user_data_replace_on_change = true

  tags = {
    Name = "terraform-example-kafka"
  }
}
```

- **Second solution:** being able to provision multiple EC2 instance, and at the same time configure all the internal detailed of each one of the EC2 instances. It has the advantage of accelerating the manual configurations by automation, and at scale. The drawback is the complexity and the lack of global configuration, i.e., configuration related with all the EC2 instances for all the EC2 instances.

```
terraform {
  required_version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region      = "us-east-1"
  access_key  = "YOUR_ACCESS_KEY"
  secret_key  = "YOUR_SECRET_KEY"
  token       = "YOUR_TOKEN"
}

variable "nBroker" {
  description = "number of brokers"
  type        = number
  default     = 3
}

resource "aws_instance" "exampleCluster" {
  ami                    = "ami-045269a1f5c90a6a0"
  instance_type          = "t2.small"
  count                  = var.nBroker
  vpc_security_group_ids  = [aws_security_group.instance.id]
  key_name                = "vockey"

  user_data     = base64encode(templatefile("creation.sh",  {
    idBroker = "${count.index}"
    totalBrokers = var.nBroker
  }))
```

```
  user_data_replace_on_change = true

  tags = {
    Name = "terraform-example-kafka.${count.index}"
  }
}

output "publicdnslist" {
  value = "${formatlist("%v", aws instance.exampleCluster.*.public dns)}"
}

resource "aws_security_group" "instance" {
  name = var.security_group_name
  ingress {
    from port   = 22
    to port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
    ingress {
    from_port   = 2181
    to_port     = 2181
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
    ingress {
    from_port   = 2888
    to_port     = 2888
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
    ingress {
    from_port   = 3888
    to_port     = 3888
    protocol    = "tcp"
    cidr blocks = ["0.0.0.0/0"]
  }
    ingress {
    from_port   = 9092
    to_port     = 9092
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-example-instances"
}
```

```
#!/bin/bash

echo "Starting..."
cd
sudo wget https://dlcdn.apache.org/zookeeper/zookeeper-3.9.3/apache-zookeeper-3.9.3-bin.tar.gz
sudo tar -zxf apache-zookeeper-3.9.3-bin.tar.gz
sudo mv apache-zookeeper-3.9.3-bin /usr/local/zookeeper
sudo mkdir -p /var/lib/zookeeper
echo "tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
maxClientCnxns=60
initLimit=10
syncLimit=5" > /usr/local/zookeeper/conf/zoo.cfg

sudo yum -y install java-17-amazon-corretto-devel.x86_64

echo ${idBroker+1} > /var/lib/zookeeper/myid

sudo /usr/local/zookeeper/bin/zkServer.sh start
```

```
sudo wget https://downloads.apache.org/kafka/3.9.0/kafka_2.13-3.9.0.tgz
sudo tar -zxf kafka 2.13-3.9.0.tgz
sudo mv kafka 2.13-3.9.0 /usr/local/kafka
sudo mkdir /tmp/kafka-logs

ip=`curl http://169.254.169.254/latest/meta-data/public-hostname`
sudo sed -i "s/#listeners=PLAINTEXT:\/\/:9092/listeners=PLAINTEXT:\/\/$ip:9092/g"
/usr/local/kafka/config/server.properties

sudo sed -i "s/broker.id=0/broker.id=${idBroker+1}/g" /usr/local/kafka/config/server.properties

sudo sed -i "s/offsets.topic.replication.factor=1/offsets.topic.replication.factor=${totalBrokers}/g"
/usr/local/kafka/config/server.properties
sudo sed -i
"s/transaction.state.log.replication.factor=1/transaction.state.log.replication.factor=${totalBrokers}/g"
/usr/local/kafka/config/server.properties
sudo sed -i "s/transaction.state.log.min.isr=1/transaction.state.log.min.isr=${totalBrokers}/g"
/usr/local/kafka/config/server.properties

# due to AWS network stablishment process, check if 30 seconds is enough for your situation
(sleep 30 && sudo /usr/local/kafka/bin/kafka-server-start.sh -daemon
/usr/local/kafka/config/server.properties )&

echo "Finished."
```

As explained in tutorial P4 - Distributed Kafka service, two global configurations are still missing in all the EC2 instances. Login in each machine and setup the following configuration:

At /usr/local/kafka/config/server.properties

```
#zookeeper connectivity (one per EC2 VM of this cluster)
zookeeper.connect=ec2-54-90-57-82.compute-1.amazonaws.com:2181,ec2-54-173-171-63.compute-
1.amazonaws.com:2181,ec2-54-236-47-54.compute-1.amazonaws.com:2181
```

And at /usr/local/zookeeper/conf/zoo.cfg

```
server.1=ec2-54-90-57-82.compute-1.amazonaws.com:2888:3888
server.2=ec2-54-173-171-63.compute-1.amazonaws.com:2888:3888
server.3=ec2-54-236-47-54.compute-1.amazonaws.com:2888:3888
```

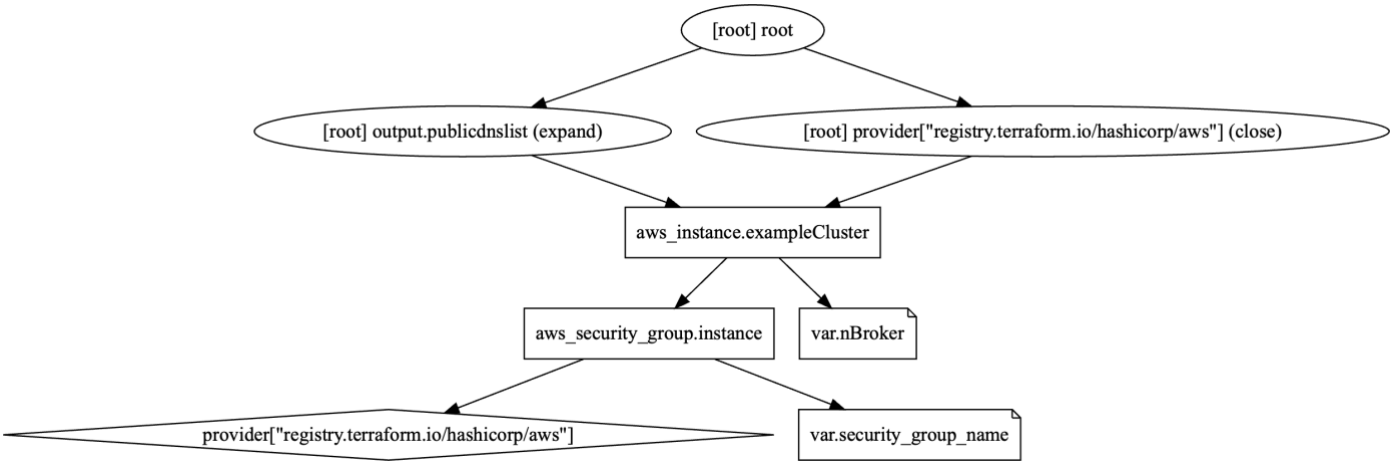
H. How to show the dependencies in terraform?

To show the dependencies of your deployment environment use the following command:

```
terraform graph -draw-cycles
```

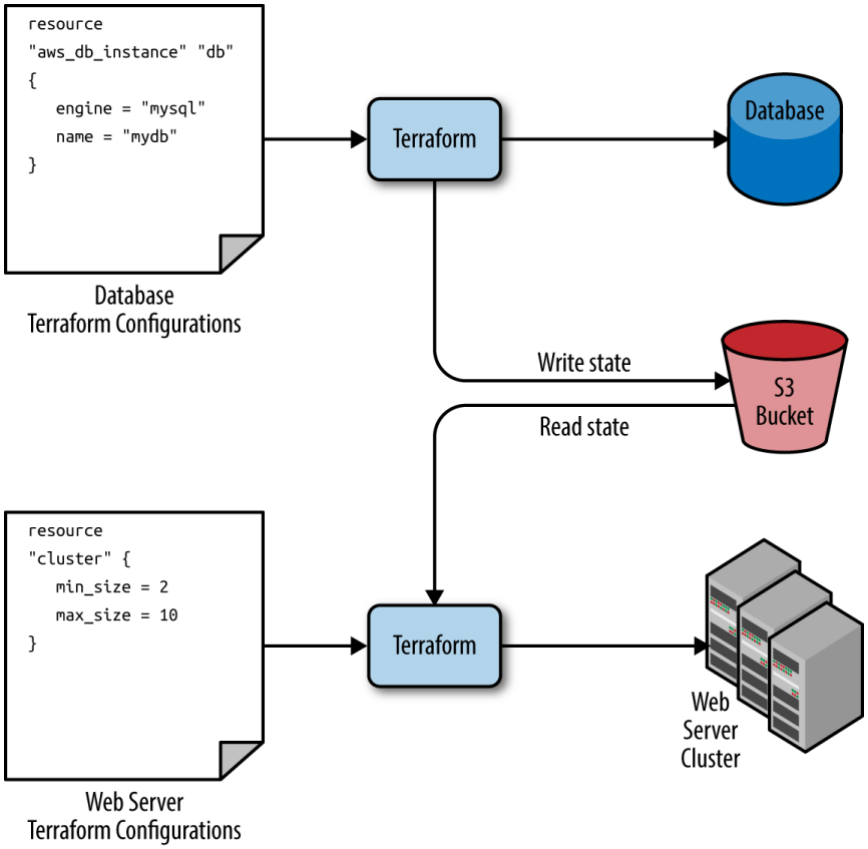
Then copy+paste the output for the graphviz, e.g., on https://dreampuf.github.io/GraphvizOnline.

Here is an example of you obtain:



I. Sharing terraform state between resources using AWS S3

The terraform state sharing allows the provisioning of an environment where **the posterior resources depend on the formers**, as depicted in the next figure. It could be done using a cloud storage or a local storage. S3 is an AWS service to store data.



Source: Brikman, Y. (2022). Terraform: Up and Running. " O'Reilly Media, Inc.".

Notice that to write the state in the storing an output should be defined in the terraform script file, i.e., to store locally the state of terraform:

```
output "publicdnslist" {
  value = "${formatlist("%v", aws_instance.exampleCluster.*.public_dns)}"
}
```

Then, you can list all the outputs, e.g.:

```
terraform state list
```

returning:

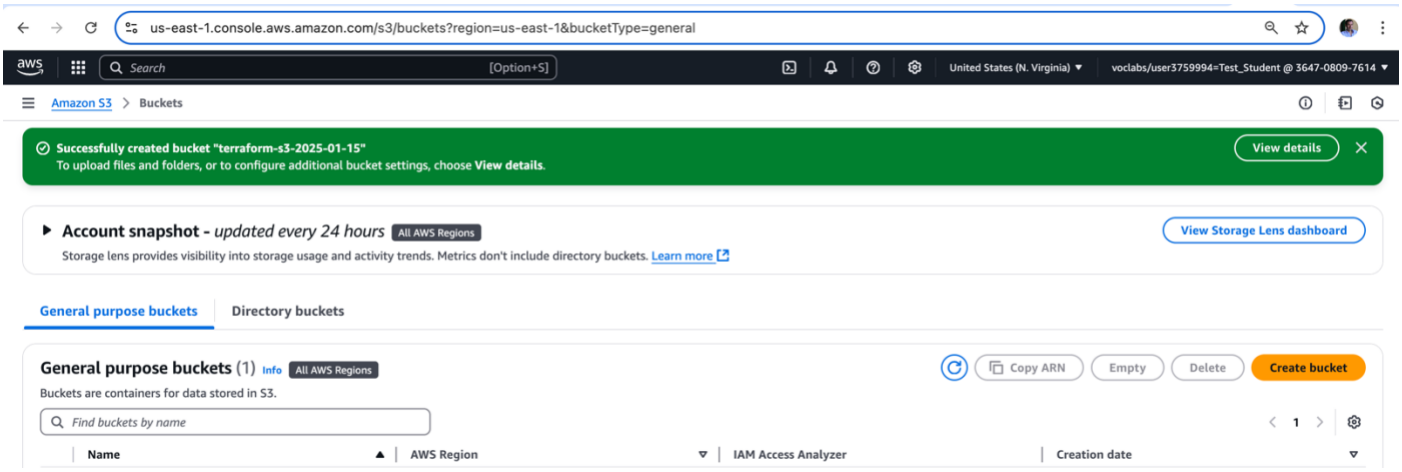
```
aws_instance.exampleCluster[0]
aws_instance.exampleCluster[1]
aws_instance.exampleCluster[2]
aws_security_group.instance
```

Or just show one of them:

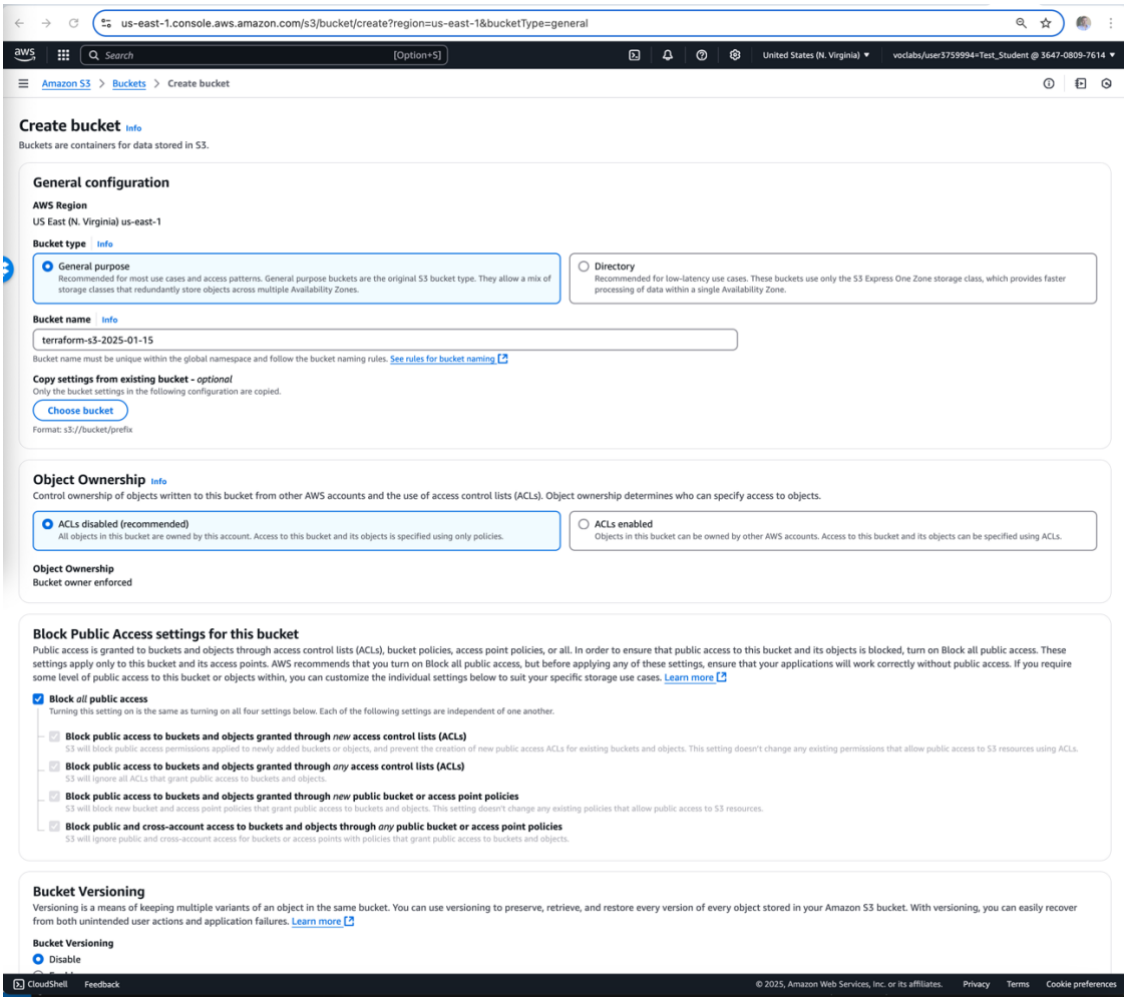
```
terraform state show 'aws_instance.exampleCluster[0]'
```

In Terraform, the state is a representation of the configuration of your infrastructure that is stored locally or remotely. Accessing the state locally involves using the terraform state command.

I.1. Choose the AWS S3 service, then, click on "**create bucket**":



Give it a unique name:



I.2. Terraform state will be stored in the S3 cloud environment. For that, firstly, create in a separate directory, a terraform file to create an AWS RDS (*e.g.*: mySQL cloud database) and store remotelly the name of the DB.

```
terraform {
  backend "s3" {
    bucket = "terraform-s3-2025-01-15"
    key= "stage/data-stores/mysql/terraform.tfstate"
    region = "us-east-1"
    access_key  = "YOUR_ACCESS_KEY"
    secret_key  = "YOUR_SECRET_KEY"
    token       = "YOUR_TOKEN"
}
  required_version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
  access_key  = "YOUR_ACCESS_KEY"
  secret_key  = "YOUR_SECRET_KEY"
  token       = "YOUR_TOKEN"
}

variable "db_username" {
  description = "The username for the database"
  type        = string
  sensitive   = true
  default     = "teste"
}

variable "db password" {
  description = "The password for the database"
  type        = string
  sensitive   = true
  default     = "testeteste"
}

variable "db_name" {
  description = "The name to use for the database"
  type        = string
  default     = "example_database_stage"
}

resource "aws db instance" "example" {
  identifier_prefix   = "terraform-up-and-running"
  engine              = "mysql"
  allocated_storage   = 20
  instance_class      = "db.t4g.micro"
  skip_final_snapshot = true
  publicly_accessible = true
  vpc_security_group_ids  = [aws_security_group.rds.id]
  db_name             = var.db_name

  username = var.db_username
  password = var.db password
}

output "address" {
  value       = aws db instance.example.address
  description = "Connect to the database at this endpoint"
}

output "port" {
  value       = aws_db_instance.example.port
  description = "The port the database is listening on"
}

resource "aws_security_group" "rds" {
  name = var.security_group_name
  ingress {
    from port   = 3306
    to port     = 3306
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port        = 0
```

```
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-rds-instance"
}
```

Execute the usual command:

```
terraform init && terraform apply
```

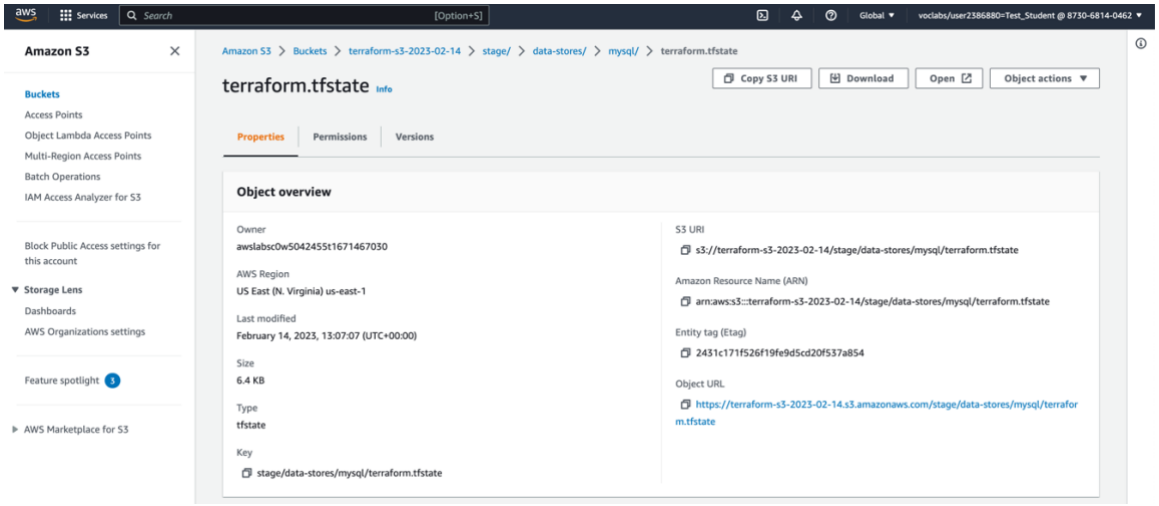
You can now check the outputs were written correctly to terraform state using the following commands:

```
terraform state list
```

```
terraform state show 'aws_db_instance.example'
```

```
terraform output
```

Moreover, if you navigate until:



You'll see that the *terraform state* of the **recently created mysql database** is stored there.

I.3. Now, in another directory, create a terraform file that can create an AWS EC2 instance writing the previous database address and port to a text file inside the EC2. The address and port are read from the state file of the previous step. With this mechanism you can create resources that are configured with parameters from the previous created resources. Facilitating the automation provisioning process!

```
terraform {
  required_version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
```

```
      version = "~> 4.0"
    }
  }
}

data "terraform_remote_state" "db" {
  backend = "s3"
  config = {
    bucket = "terraform-s3-2025-01-15"
    key= "stage/data-stores/mysql/terraform.tfstate"
    region = "us-east-1"
    access_key  = "YOUR_ACCESS_KEY"
    secret_key  = "YOUR_SECRET_KEY"
    token       = "YOUR_TOKEN"
  }
}

provider "aws" {
  region = "us-east-1"
  access_key  = "YOUR_ACCESS_KEY"
  secret_key  = "YOUR_SECRET_KEY"
  token       = "YOUR_TOKEN"
}

resource "aws_instance" "exampleCluster" {
  ami                    = "ami-045269a1f5c90a6a0"
  instance_type          = "t2.small"
  vpc_security_group_ids  = [aws_security_group.instance.id]
  key_name               = "vockey"

  user data     = base64encode(templatefile("creation.sh",  {
    address = data.terraform remote state.db.outputs.address
    port    = data.terraform_remote_state.db.outputs.port
  }))

  user data replace on change = true

  tags = {
    Name = "terraform-example-sharestate"
  }
}

resource "aws security group" "instance" {
  name = var.security_group_name
  ingress {
    from_port   = 22
    to port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}

variable "security group name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-example-instance2"
}
```

And creation.sh with the following source code:

```
#!/bin/bash
cd

echo ${address} > address.txt
echo ${port} > port.txt
```

Execute the usual command:

```
terraform init && terraform apply
```

Login in the newly created EC2 instance and verify in the **root directory** that two new files are now created with the content provided by DB configuration.

J. Creating Lambda function and API Gateway trigger using S3

J.1. To create a lambda function, written in JAVA 8, that you previously compiled locally, you need to transfer the .jar file to S3, then create the AWS lambda function resource, and finally, add an API Gateway trigger for that lambda function.

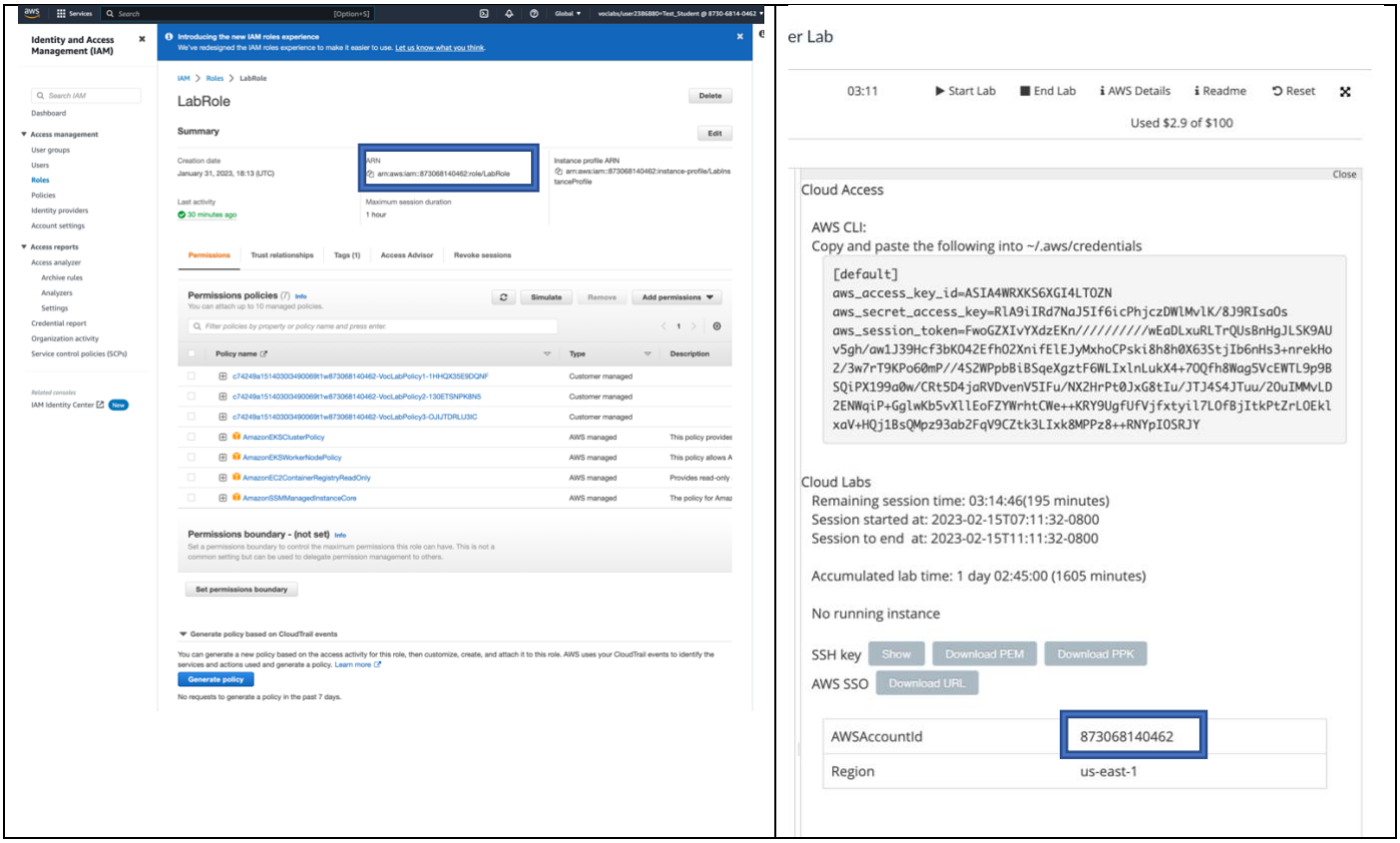
```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.15.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.1.0"
    }
    archive = {
      source  = "hashicorp/archive"
      version = "~> 2.2.0"
    }
  }
  required_version = "~> 1.2"
}

provider "aws" {
  region = "us-east-1"
  access_key  = ""
  secret_key  = ""
  token = ""
}

## S3 PART
resource "aws s3 bucket" "lambda bucket" {
  bucket = "terraform-lambda-2025-01-16"
}

 resource "aws_s3_object" "lambda_hello_world" {
   bucket = aws_s3_bucket.lambda_bucket.id
   key    = "lambda-java-example-0.0.1-SNAPSHOT.jar"
   source = "${path.module}/lambda-java-example-0.0.1-SNAPSHOT.jar"
 }

## LAMBDA FUNCTION PART
resource "aws lambda function" "hello world" {
  function_name = "HelloWorld"
  s3_bucket = aws_s3_bucket.lambda_bucket.id
  s3_key    = aws_s3_object.lambda_hello_world.key
  runtime = "java11"
  handler = "example.Hello::handleRequest"
  role = "arn:aws:iam::8730681408695462:role/LabRole"
}

resource "aws_cloudwatch_log_group" "hello_world" {
  name = "/aws/lambda/${aws_lambda_function.hello_world.function_name}"
  retention in days = 30
}

### API GATEWAY PART
resource "aws api gateway rest api" "example" {
  name        = "ServerlessExample"
  description = "Terraform Serverless Application Example"
}

resource "aws_api_gateway_resource" "proxy" {
  rest_api_id = "${aws_api_gateway_rest_api.example.id}"
  parent_id   = "${aws_api_gateway_rest_api.example.root_resource_id}"
  #path_part   = "{proxy+}"
  path_part   = "helloworldpath"
}

resource "aws api gateway method" "proxy" {
  rest api id   = "${aws api gateway rest api.example.id}"
  resource_id   = "${aws_api_gateway_resource.proxy.id}"
  http_method   = "ANY"
  authorization = "NONE"
}

resource "aws_api_gateway_integration" "lambda" {
```

```
  rest_api_id = "${aws_api_gateway_rest_api.example.id}"
  resource_id = "${aws_api_gateway_method.proxy.resource_id}"
  http method = "${aws api gateway method.proxy.http method}"

  integration_http_method = "POST"
  type                    = "AWS_PROXY"
  uri                     = "${aws_lambda_function.hello_world.invoke_arn}"
}

resource "aws_api_gateway_deployment" "example" {
  depends_on = [
    aws_api_gateway_integration.lambda,
  ]
  rest_api_id = "${aws_api_gateway_rest_api.example.id}"
  stage name  = "test"
}

resource "aws_lambda_permission" "apigw" {
  statement id  = "AllowAPIGatewayInvoke"
  action        = "lambda:InvokeFunction"
  function_name = "${aws_lambda_function.hello_world.function_name}"
  principal     = "apigateway.amazonaws.com"

  # The /*/* portion grants access from any method on any resource
  # within the API Gateway "REST API".
  source_arn = "${aws_api_gateway_rest_api.example.execution_arn}/*/*"
}

# Output value definitions
output "lambda_bucket_name" {
  description = "Name of the S3 bucket used to store function code."
  value = aws s3 bucket.lambda bucket.id
}

output "function name" {
  description = "Name of the Lambda function."
  value = aws lambda function.hello world.function name
}

output "base_url" {
  value = "${aws_api_gateway_deployment.example.invoke_url}"
}
```

Where the field role of resource "aws_lambda_function":

```
  role = "arn:aws:iam::8730681408695462:role/LabRole"
```

could be consulted on your AWS account, in the IAM service, or replacing your AWSAccountId number directly in the bold part of the field:



J.2. To test your lambda function, externally from any machine, you need to connect to AWS API Gateway, as explained in the Lambda-AWS tutorial:

```
curl -i -H "Content-Type: Application/json" --data "@body.json"  -X POST  https://<your URL>/test/helloworldpath
```

And the result obtained is the following:

```
HTTP/2 200
content-type: application/json
content-length: 43
date: Wed, 15 Feb 2023 16:36:51 GMT
x-amzn-requestid: f2027a23-803d-42d3-8537-e2fe256d7480
x-amz-apigw-id: AY6FiH8XoAMFbvA=
x-custom-header: my custom header value
x-amzn-trace-id: Root=1-63ed0a23-652880af6750972f7523767e;Sampled=0
x-cache: Miss from cloudfront
via: 1.1 cb4f40303e252a22c4df5918669814ac.cloudfront.net (CloudFront)
x-amz-cf-pop: LIS50-C1
x-amz-cf-id: CAQtv4l6vFuEzDsjQv7tj56smEc7s4ZkcEffM3VobsGSKnB-1H2y4g==

{"message":"Hello Integracao Empresarial!"}
```

P2-HyperAutomation-TERRAFORM-v1.2.docx

K. Kong and Konga deployed in AWS EC2 using Terraform

K.1. To install Kong you will need to follow the indications as explained in the Kong tutorial. They are now adapted to be hyperautomated using Terraform.

Where the .tf file can include the following instructions:

```
terraform {
  required version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region      = "us-east-1"
  access_key  = ""
  secret_key  = ""
  token = ""
}

resource "aws_instance" "exampleInstallKong" {
  ami                     = "ami-045269a1f5c90a6a0"
  instance type           = "t2.small"
  vpc security group ids  = [aws security group.instance.id]
  key name                = "vockey"

  user_data = "${file("deploy.sh")}"

  user_data_replace_on_change = true

  tags = {
    Name = "terraform-example-Kong"
  }
}

resource "aws security group" "instance" {
  name = var.security_group_name
  ingress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
  egress {
    from_port        = 0
    to port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-kong-instance"
}

output "address" {
  value       = aws_instance.exampleInstallKong.public_dns
  description = "Connect to the database at this endpoint"
}
```

And the correspondingly script with the following:

```
#!/bin/bash
echo "Starting..."

sudo yum install -y docker

sudo service docker start

sudo docker network create kong-net

sudo docker run -d --name kong-database \
  --network=kong-net \
  -p 5432:5432 \
  -e "POSTGRES_USER=kong" \
  -e "POSTGRES_DB=kong" \
  -e "POSTGRES_PASSWORD=kongpass" \
  postgres:13

sudo docker run --rm --network=kong-net \
-e "KONG_DATABASE=postgres" \
-e "KONG_PG_HOST=kong-database" \
-e "KONG_PG_PASSWORD=kongpass" \
-e "KONG_PASSWORD=test" \
kong/kong-gateway:3.9.0.0 kong migrations bootstrap

sudo docker run -d --name kong-gateway \
--network=kong-net \
-e "KONG_DATABASE=postgres" \
-e "KONG_PG_HOST=kong-database" \
-e "KONG_PG_USER=kong" \
-e "KONG_PG_PASSWORD=kongpass" \
-e "KONG_PROXY_ACCESS_LOG=/dev/stdout" \
-e "KONG_ADMIN_ACCESS_LOG=/dev/stdout" \
-e "KONG_PROXY_ERROR_LOG=/dev/stderr" \
-e "KONG_ADMIN_ERROR_LOG=/dev/stderr" \
-e "KONG_ADMIN_LISTEN=0.0.0.0:8001, 0.0.0.0:8444 ssl" \
-e "KONG_ADMIN_GUI_URL=http://localhost:8002" \
-e KONG_LICENSE_DATA \
-p 8000:8000 \
-p 8443:8443 \
-p 8001:8001 \
-p 8002:8002 \
-p 8445:8445 \
-p 8003:8003 \
-p 8004:8004 \
-p 127.0.0.1:8444:8444 \
kong/kong-gateway:3.9.0.0

echo "Finished."
```

K.2. To install the UI Konga again you can follow the indications as explained in the Kong tutorial. They are now adapted to be hyperautomated using Terraform.

Where the .tf file can include the following instructions:

```
terraform {
  required_version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region      = "us-east-1"
  access_key  = ""
  secret_key  = ""
  token = ""
}

resource "aws_instance" "exampleInstallKonga" {
  ami                    = "ami-045269a1f5c90a6a0"
  instance_type          = "t2.small"
```

```
  vpc_security_group_ids  = [aws_security_group.instance.id]
  key_name                = "vockey"

  user data = "${file("deploy.sh")}"

  user_data_replace_on_change = true

  tags = {
    Name = "terraform-example-Konga"
  }
}

resource "aws_security_group" "instance" {
  name = var.security_group_name
  ingress {
    from port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr blocks = ["0.0.0.0/0"]
    ipv6 cidr blocks = ["::/0"]
  }
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}

variable "security group name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-konga-instance"
}

output "address" {
  value       = aws_instance.exampleInstallKonga.public_dns
  description = "Connect to the database at this endpoint"
}
```

And the correspondingly script with the following:

```
#!/bin/bash
echo "Starting..."

sudo yum install -y docker

sudo service docker start

sudo docker pull pantsel/konga

sudo docker run -d --name konga -p 1337:1337 pantsel/konga

echo "Finished."
```

L. Camunda Engine deployed in AWS EC2 using Terraform

L.1. To install Camunda Engine you will need to follow the indications as explained in the Camunda tutorial. They are now adapted to be hyperautomated using Terraform.

Where the .tf file can include the following instructions:

```
terraform {
  required version = ">= 1.0.0, < 2.0.0"

  required providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region      = "us-east-1"
  access_key  = ""
  secret_key  = ""
  token = ""
}

resource "aws instance" "exampleInstallCamundaEngine" {
  ami                     = "ami-045269a1f5c90a6a0"
  instance_type           = "t2.small"
  vpc_security_group_ids  = [aws_security_group.instance.id]
  key_name                = "vockey"

  user_data = "${file("deploy.sh")}"

  user data replace on change = true

  tags = {
    Name = "terraform-example-Camunda"
  }
}

resource "aws_security_group" "instance" {
  name = var.security_group_name
  ingress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr blocks      = ["0.0.0.0/0"]
    ipv6 cidr blocks = ["::/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-Camunda-instance"
}

output "address" {
  value       = aws instance.exampleInstallCamundaEngine.public dns
  description = "Connect to the database at this endpoint"
}
```

And the correspondingly script with the following:

```
#!/bin/bash
echo "Starting..."

sudo yum update -y

sudo yum install -y docker

sudo service docker start

sudo docker pull camunda/camunda-bpm-platform:latest

sudo docker run -d --name camunda -p 8080:8080 camunda/camunda-bpm-platform:latest

echo "Finished."
```

M. Quarkus project deployed in AWS EC2 using Terraform

M.1. To deploy a previous created and compiled Quarkus project you will need to have the image previously pushed to git, Then, use the following files for reference. Where the .tf file can include the following instructions:

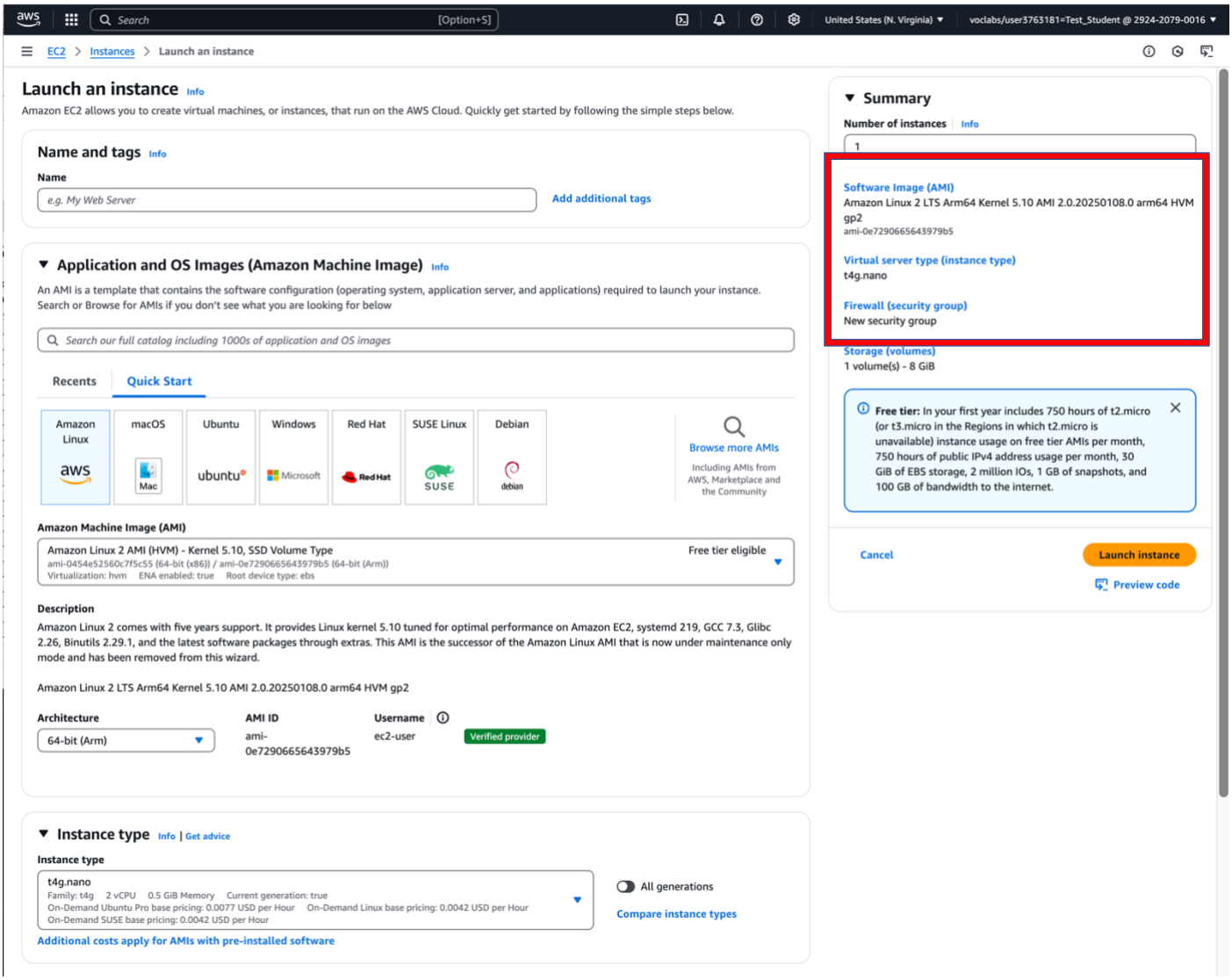
```
terraform {
  required_version = ">= 1.0.0, < 2.0.0"

  required providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region       = "us-east-1"
  access_key   = "YOUR ACESS KEY"
  secret_key   = "YOUR SECRET KEY"
  token        = "YOUR TOKEN"
}


resource "aws instance" "exampleDeployQuarkus" {
  ami                     = "ami-0e7290665643979b5" # Amazon Linux ARM AMI built by Amazon Web Services - FOR
DOCKER image COMPATIBILITY if compiled previously on ARM
  instance_type           = "t4g.nano"
  vpc security group ids  = [aws security group.instance.id]
  key name                = "vockey"

  user_data = "${file("quarkus.sh")}"

  user_data_replace_on_change = true

  tags = {
    Name = "terraform-deploy-QuarkusProject"
  }
}

resource "aws security group" "instance" {
  name = var.security group name
  ingress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-Quarkus-instance"
}

output "address" {
  value       = aws_instance.exampleDeployQuarkus.public_dns
  description = "Address of the Quarkus EC2 machine"
}
```

And the correspondingly script with the following:

```
#!/bin/bash
echo "Starting..."

sudo yum install -y docker

sudo service docker start

sudo docker login -u "YOUR DOCKER USERNAME" -p "YOUR DOCKER PASSWORD"

sudo docker pull YOUR DOCKER USERNAME/tryout1:1.0.0-SNAPSHOT

sudo docker run -d --name tryout2 -p 9000:9000 YOUR DOCKER USERNAME/tryout1:1.0.0-SNAPSHOT

echo "Finished."
```

Docker images built for ARM architecture may not work on AMD architecture machines. This is because ARM and AMD processors use different instruction sets and have different hardware architectures.

For this example, the AMI identifier and instance type were extracted from the launching instance AWS GUI.



N. Ollama deployed in AWS EC2 using Terraform

N.1. To install Ollama in a AWS EC2 instance using Terraform follows the following the **.tf file** to create the needed resources:

```
terraform {
  required_version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region      = "us-east-1"
  access_key  = "YOUR AWS ACCESS KEY"
  secret_key  = "YOUR AWS SECRET KEY"
  token = "YOUR AWS TOKEN"
}

resource "aws_instance" "exampleOllamaConfiguration" {
  ami                    = "ami-045269a1f5c90a6a0"
  instance_type          = "t2.medium"
  count                  = 1
  vpc_security_group_ids  = [aws_security_group.instance.id]

  root_block_device {
    volume_size = 24 # In Gb
  }

  key_name  = "vockey"
  user_data = "${file("creation.sh")}"

  user_data_replace_on_change = true

  tags = {
    Name = "terraform-example-ollama"
  }
}

resource "aws_security_group" "instance" {
  name = var.security_group_name
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
    ingress {
    from_port   = 11434
    to_port     = 11434
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}

variable "security_group_name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-ollama-example-instance"
}

output "address" {
  value       = aws_instance.exampleOllamaConfiguration[*].public_dns
  description = "Address of the Kafka EC2 machine with ollama"
}
```

N.2. And the following bash script to be executed after EC2 resource creation by the following script **creation.sh**:

```
#!/bin/bash
cd
sudo yum update -y

sudo curl -fsSL https://ollama.com/install.sh | sh

export HOME=$HOME:/usr/local/bin
sudo sed -i "s/\[Install\]/Environment=\"OLLAMA_HOST=0.0.0.0:11434\"\n\[Install\]/g"
/etc/systemd/system/ollama.service

sudo systemctl enable ollama
sudo systemctl start ollama

ollama pull llama3.2
```

Hint 1: Additionally, if inside the EC2 instance you would like to check the models loaded use the following command:

```
[ec2-user@ip-172-31-23-142 ~]$ ollama list
NAME           ID              SIZE      MODIFIED
llama3.2:1b    baf6a787fdff    1.3 GB    7 minutes ago
```

Hint 2: Additionally, if inside the EC2 instance you would like to check the processors loaded use the following command:

```
[ec2-user@ip-172-31-23-142 ~]$ ollama ps
NAME           ID              SIZE      PROCESSOR    UNTIL
llama3.2:1b    baf6a787fdff    2.2 GB    100% CPU     4 minutes from now
```

Hint 3: Additionally, a request example can be tested with the following remote curl command:

```
curl http://<EC2 DNS NAME>:11434/api/generate -d '{
  "model": "llama3.2",
  "prompt": "Why is the sky blue?",
  "stream": false
}'
```

The response obtained is similar with the following:

{"model":"llama3.2","created_at":"2025-02-12T15:18:11.983219976Z","response":"The sky appears blue because of a phenomenon called scattering, which occurs when sunlight interacts with the tiny molecules of gases in the Earth's atmosphere.\n\nHere's a simplified explanation:\n\n1. **Sunlight**: When the sun shines, it emits white light, which contains all the colors of the visible spectrum (red, orange, yellow, green, blue, indigo, and violet).\n2. **Atmospheric scattering**: As sunlight enters the Earth's atmosphere, it encounters tiny molecules of gases such as nitrogen (N2) and oxygen (O2). These molecules are much smaller than the wavelength of light.\n3. **Scattering**: When sunlight hits these tiny molecules, it scatters in all directions. However, not all wavelengths scatter equally. The shorter (blue) wavelengths are scattered more than the longer (red) wavelengths by the smaller gas molecules.\n4. **Blue light dominates**: As a result of this scattering effect, the blue light is dispersed throughout the atmosphere, while the red and orange light continue to travel in a straight line.\n5. **Our eyes perceive the sky as blue**: From our perspective on the surface of the Earth, we see the scattered blue light coming from all directions, which gives the sky its blue appearance.\n\nThis phenomenon is more pronounced during the daytime when the sun is overhead, and the scattering effect is greater. During sunrise and sunset, the sun's rays have to travel through more of the atmosphere, scattering off even more molecules and creating the warm hues we see.\n\nSo, in short, the sky appears blue because of the scattering of sunlight by tiny molecules in the Earth's atmosphere, which favors shorter wavelengths (like blue light) over longer wavelengths (like red light).","done":true,"done_reason":"stop","context":[128006,9125,128007,271,38766,1303,33025,2696,25,6790,220,2366,18,271,128009,128006,882,128007,271,10445,374,279,13180,6437,30,128009,128006,78191,128007,271,791,13180,8111,6437,1606,315,264,25885,2663,72916,11,902,13980,994,40120,84261,449,279,13987,35715,315,45612,304,279,9420,596,16975,382,8586,596,264,44899,16540,1473,16,13,3146,31192,4238,96618,3277,279,7160,65880,11,433,73880,4251,3177,11,902,5727,682,279,8146,315,279,9621,20326,320,1171,11,19087,11,14071,11,6307,11,6437,11,1280,7992,11,323,80836,4390,17,13,3146,1688,8801,33349,72916,96618,1666,40120,29933,279,9420,596,16975,11,433,35006,13987,35715,315,45612,1778,439,47503,320,45,17,8,323,24463,320,46,17,570,4314,35715,527,1790,9333,1109,279,46406,315,3177,627,18,13,3146,3407,31436,96618,3277,40120,13280,1521,13987,35715,11,433,1156,10385,304,682,18445,13,4452,11,539,682,93959,45577,18813,13,578,24210,320,12481,8,93959,527,38067,810,1109,279,5129,320,1171,8,93959,555,279,9333,6962,35715,627,19,13,3146,10544,3177,83978,96618,1666,264,1121,315,420,72916,2515,11,279,6437,3177,374,77810,6957,279,16975,11,1418,279,2579,323,19087,3177,3136,311,5944,304,264,7833,1584,627,20,13,3146,8140,6548,45493,279,13180,439,6437,96618,5659,1057,13356,389,279,7479,315,279,9420,11,584,1518,279,38067,6437,3177,5108,505,682,18445,11,902,6835,279,13180,1202,6437,11341,382,2028,25885,374,810,38617,2391,279,62182,994,279,7160,374,32115,11,323,279,72916,2515,374,7191,13,12220,64919,323,44084,11,279,7160,596,45220,617,311,5944,1555,810,315,279,16975,11,72916,1022,1524,810,35715,323,6968,279,8369,82757,584,1518,382,4516,11,304,2875,11,279,13180,8111,6437,1606,315,279,72916,315,40120,555,13987,35715,304,279,9420,596,16975,11,902,54947,24210,93959,320,4908,6437,3177,8,927,5129,93959,320,4908,2579,3177,570],"total_duration":72783017580,"load_duratio

n":3846465248,"prompt_eval_count":31,"prompt_eval_duration":3108000000,"eval_count":344,"eval_duration":65784000000}
%

References

URLs with other resources

- Terraform Up & Running, Writing Infrastructure as Code, Yevgeniy Brikman, 3rd edition (2022). https://www.terraformupandrunning.com/
    - The github with all the source code from the book is at: https://github.com/brikis98/terraform-up-and-running-code
- https://developer.hashicorp.com/terraform/downloads
- https://developer.hashicorp.com/terraform/tutorials/aws-get-started
- https://stackoverflow.com/questions/41596412/how-to-use-terraform-output-as-input-variable-of-another-terraform-template
- https://developer.hashicorp.com/terraform/tutorials/aws/lambda-api-gateway
- https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html
- https://registry.terraform.io/providers/hashicorp/aws/2.34.0/docs/guides/serverless-with-aws-lambda-and-api-gateway
- Ollama API documentation: https://github.com/ollama/ollama/blob/main/docs/api.md
- Ollama FAQs documentation: https://github.com/ollama/ollama/blob/main/docs/faq.md
- Ollama readme page https://github.com/ollama/ollama?tab=readme-ov-file

DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Tutorial – Distributed Kafka service

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to show how to use the Kafka in a cluster environment.

Consider that all the consumers, producers, and topics management indicated in this tutorial are executed, by the scripts delivered in the Kafka distribution, in a command line, but outside the EC2 AWS instance with your Kafka installation.
For that end, create an additional new EC2 AWS instance with Kafka installed without starting its services. Then you can create as many sessions as you need.

The following contents is presented in this document.

Contents

A. Listing all topics

```
sudo /usr/local/kafka/bin/kafka-topics.sh --list --bootstrap-server <YourIP_or_DNS>:9092

CustomerCountry
EXERCISE3
__consumer_offsets
novo_topico_automatico1
novo_topico_automatico2
novo_topico_automatico3
novotopico
test
```

B. Creating one topic with multiple partitions

```
sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server <YourIP_or_DNS>:9092 -replication-factor 1 --partitions 8 --topic clicks

Created topic "clicks".
```

C. Adding partitions for a specific topic

```
sudo /usr/local/kafka/bin/kafka-topics.sh --alter --bootstrap-server <YourIP_or_DNS>:9092 --partitions 10 --topic clicks

WARNING: If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected
Adding partitions succeeded!
```

D. Deleting one topic

```
sudo /usr/local/kafka/bin/kafka-topics.sh --delete --bootstrap-server <YourIP_or_DNS>:9092 --topic clicks

Topic clicks is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

E. Listing the count of partitions for a specific topic

```
/usr/local/kafka/bin/kafka-topics.sh --describe --bootstrap-server <YourIP_or_DNS>:9092 --topic clicks

Topic:clicks    PartitionCount:8        ReplicationFactor:1     Configs:
        Topic: clicks   Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 2    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 3    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 4    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 5    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 6    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 7    Leader: 0       Replicas: 0     Isr: 0
```

F. Producing messages to a specific topic with multiple partitions

Remember that:

```
/usr/local/kafka/bin/kafka-console-producer.sh --broker-list <YourIP_or_DNS>:9092 --topic clicks
```

G. Consuming messages from a topic with multiple partitions

Remember that:

```
/usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server <YourIP_or_DNS>:9092 --topic clicks --from-beginning
```

H. Listing the consumer groups

```
sudo /usr/local/kafka/bin/kafka-consumer-groups.sh --bootstrap-server <YourIP_or_DNS>:9092 -list

console-consumer-2817
console-consumer-57329
console-consumer-28356
```

I. Listing the consumer-ids from a specific consumer group

```
sudo /usr/local/kafka/bin/kafka-consumer-groups.sh --bootstrap-server <YourIP_or_DNS>:9092 -describe -group console-consumer-28356

TOPIC          PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
clicks         1          -               0               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks         4          -               0               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks         7          -               0               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks         0          -               1               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks         5          -               0               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks         2          -               1               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks         3          -               1               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks         6          -               0               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
```

J. Defining the group-id in a specific consumer

In command line:

```
sudo /usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server <YourIP_or_DNS>:9092 --topic clicks --from-beginning --group group-id-teste
```

In Java:

```
Define the property:
                                    props.put("group.id", "group-id-teste");

as explained in the section C. of P3-Kafka-ClientsJAVA document.
```

K. Creating two brokers in the same EC2 VM

1. Stop the kafka server if running.
2. Create the two configuration files

```
cp /usr/local/kafka/config/server.properties /usr/local/kafka/config/server-1.properties
cp /usr/local/kafka/config/server.properties /usr/local/kafka/config/server-2.properties
```

3. Edit these new files with the following set of properties (if in the same AWS EC2 instance, otherwise the port and log directory could be the same):

```
config/server-0.properties:
    broker.id=0
    listeners=PLAINTEXT://<YourIP_or_DNS>:9093
    offsets.topic.replication.factor=2
    transaction.state.log.replication.factor=2
    transaction.state.log.min.isr=2
    log.dir=/tmp/kafka-logs-0
```

```
config/server-1.properties:
    broker.id=1
    listeners=PLAINTEXT://<YourIP_or_DNS>:9094
    offsets.topic.replication.factor=2
    transaction.state.log.replication.factor=2
    transaction.state.log.min.isr=2
    log.dir=/tmp/kafka-logs-1
```

4. Start Kafka broker 1 with the command referring to server config file #1

```
    sudo /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server-0.properties
```

5. Start Kafka broker 2 with the command referring to server config file #2

```
    sudo /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server-1.properties
```

6. Open the in-bound ports 9093 and 9094 in the AWS EC2 console

7. Create a topic with replication factor=2 and multiple partitions

```
sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server <YourIP_or_DNS>:9092 -replication-factor 2 --partitions 2 --topic testing-cluster

Created topic "testing-cluster".
```

L. Checking that cluster of brokers is launched

Executing the following command:

```
                                    ps -ef |grep java |grep server
```

You should receive something similar with (one PID for each broker using each configuration):

```
root     31619     1  4 08:43 pts/0    00:00:08 java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -
XX:InitiatingHeapOccupancyPercent=35 -XX:+ExplicitGCInvokesConcurrent -Djava.awt.headless=true -Xloggc:/usr/local/kafka/bin/../logs/kafkaServer-
gc.log -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -
XX:GCLogFileSize=100M -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -
Dkafka.logs.dir=/usr/local/kafka/bin/../logs -Dlog4j.configuration=file:/usr/local/kafka/bin/../config/log4j.properties -cp
/usr/local/kafka/bin/../libs/activation-1.1.1.jar:/usr/local/kafka/bin/../libs/aopalliance-repackaged-2.5.0-
b42.jar:/usr/local/kafka/bin/../libs/argparse4j-0.7.0.jar:/usr/local/kafka/bin/../libs/audience-annotations-
0.5.0.jar:/usr/local/kafka/bin/../libs/commons-lang3-
3.5.jar:/usr/local/kafka/bin/../libs/compileScala.mapping:/usr/local/kafka/bin/../libs/compileScala.mapping.asc:/usr/local/kafka/bin/../libs/conn
ect-api-2.1.0.jar:/usr/local/kafka/bin/../libs/connect-basic-auth-extension-2.1.0.jar:/usr/local/kafka/bin/../libs/connect-file-
2.1.0.jar:/usr/local/kafka/bin/../libs/connect-json-2.1.0.jar:/usr/local/kafka/bin/../libs/connect-runtime-
2.1.0.jar:/usr/local/kafka/bin/../libs/connect-transforms-2.1.0.jar:/usr/local/kafka/bin/../libs/guava-20.0.jar:/usr/local/kafka/bin/../libs/hk2-
api-2.5.0-b42.jar:/usr/local/kafka/bin/../libs/hk2-locator-2.5.0-b42.jar:/usr/local/kafka/bin/../libs/hk2-utils-2.5.0-
```

```
b42.jar:/usr/local/kafka/bin/../libs/jackson-annotations-2.9.7.jar:/usr/local/kafka/bin/../libs/jackson-core-
2.9.7.jar:/usr/local/kafka/bin/../libs/jackson-databind-2.9.7.jar:/usr/local/kafka/bin/../libs/jackson-jaxrs-base-
2.9.7.jar:/usr/local/kafka/bin/../libs/jackson-jaxrs-json-provider-2.9.7.jar:/usr/local/kafka/bin/../libs/jackson-module-jaxb-annotations-
2.9.7.jar:/usr/local/kafka/bin/../libs/javassist-3.22.0-CR2.jar:/usr/local/kafka/bin/../libs/javax.annotation-api-
1.2.jar:/usr/local/kafka/bin/../libs/javax.inject-1.jar:/usr/local/kafka/bin/../libs/javax.inject-2.5.0-
b42.jar:/usr/local/kafka/bin/../libs/javax.servlet-api-3.1.0.jar:/usr/local/kafka/bin/../libs/javax.ws.rs-api-
2.1.1.jar:/usr/local/kafka/bin/../libs/javax.ws.rs-api-2.1.jar:/usr/local/kafka/bin/../libs/jaxb-api-
2.3.0.jar:/usr/local/kafka/bin/../libs/jersey-client-2.27.jar:/usr/local/kafka/bin/../libs/jersey-common-
2.27.jar:/usr/local/kafka/bin/../libs/jersey-container-servlet-2.27.jar:/usr/local/kafka/bin/../libs/jersey-container-servlet-core-
2.27.jar:/usr/local/kafka/bin/../libs/jersey-hk2-2.27.jar:/usr/local/kafka/bin/../libs/jersey-media-jaxb-
2.27.jar:/usr/local/kafka/bin/../libs/jersey-server-2.27.jar:/usr/local/kafka/bin/../libs/jetty-client-
9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-continuation-9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-http-
9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-io-9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-security-
9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty server-9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-servlet-
9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-servlets-9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-util-
9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jopt-simple-5.0.4.jar:/usr/local/kafka/bin/../libs/kafka_2.12-
2.1.0.jar:/usr/local/kafka/bin/../libs/kafka_2.12-2.1.0-sources.jar:/usr/local/kafka/bin/../libs/kafka-clients-
2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-log4j-appender-2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-streams-
2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-streams-examples-2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-streams-scala_2.12-
2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-streams-test-utils-2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-tools-
2.1.0.jar:/usr/local/kafka/bin/../libs/log4j-1.2.17.jar:/usr/local/kafka/bin/../libs/lz4-java-1.5.0.jar:/usr/local/kafka/bin/../libs/maven-
artifact-3.5.4.jar:/usr/local/kafka/bin/../libs/metrics-core-2.2.0.jar:/usr/local/kafka/bin/../libs/osgi-resource-locator-
1.0.1.jar:/usr/local/kafka/bin/../libs/plexus-utils-3.1.0.jar:/usr/local/kafka/bin/../libs/reflections-
0.9.11.jar:/usr/local/kafka/bin/../libs/rocksdbjni-5.14.2.jar:/usr/local/kafka/bin/../libs/scala-library-
2.12.7.jar:/usr/local/kafka/bin/../libs/scala-logging_2.12-3.9.0.jar:/usr/local/kafka/bin/../libs/scala-reflect-
2.12.7.jar:/usr/local/kafka/bin/../libs/slf4j-api-1.7.25.jar:/usr/local/kafka/bin/../libs/slf4j-log4j12-
1.7.25.jar:/usr/local/kafka/bin/../libs/snappy-java-1.1.7.2.jar:/usr/local/kafka/bin/../libs/validation-api-
1.1.0.Final.jar:/usr/local/kafka/bin/../libs/zkclient-0.10.jar:/usr/local/kafka/bin/../libs/zookeeper-
3.4.13.jar:/usr/local/kafka/bin/../libs/zstd-jni-1.3.5-4.jar kafka.Kafka /usr/local/kafka/config/server-0.properties

root     31959     1  3 08:43 pts/0    00:00:06 java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -
XX:InitiatingHeapOccupancyPercent=35 -XX:+ExplicitGCInvokesConcurrent -Djava.awt.headless=true -Xloggc:/usr/local/kafka/bin/../logs/kafkaServer-
gc.log -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -
XX:GCLogFileSize=100M -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -
Dkafka.logs.dir=/usr/local/kafka/bin/../logs -Dlog4j.configuration=file:/usr/local/kafka/bin/../config/log4j.properties -cp
/usr/local/kafka/bin/../libs/activation-1.1.1.jar:/usr/local/kafka/bin/../libs/aopalliance-repackaged-2.5.0-
b42.jar:/usr/local/kafka/bin/../libs/argparse4j-0.7.0.jar:/usr/local/kafka/bin/../libs/audience-annotations-
0.5.0.jar:/usr/local/kafka/bin/../libs/commons-lang3-
3.5.jar:/usr/local/kafka/bin/../libs/compileScala.mapping:/usr/local/kafka/bin/../libs/compileScala.mapping.asc:/usr/local/kafka/bin/../libs/conn
ect-api-2.1.0.jar:/usr/local/kafka/bin/../libs/connect-basic-auth-extension-2.1.0.jar:/usr/local/kafka/bin/../libs/connect-file-
2.1.0.jar:/usr/local/kafka/bin/../libs/connect-json-2.1.0.jar:/usr/local/kafka/bin/../libs/connect-runtime-
2.1.0.jar:/usr/local/kafka/bin/../libs/connect-transforms-2.1.0.jar:/usr/local/kafka/bin/../libs/guava-20.0.jar:/usr/local/kafka/bin/../libs/hk2-
api-2.5.0-b42.jar:/usr/local/kafka/bin/../libs/hk2-locator-2.5.0-b42.jar:/usr/local/kafka/bin/../libs/hk2-utils-2.5.0-
b42.jar:/usr/local/kafka/bin/../libs/jackson-annotations-2.9.7.jar:/usr/local/kafka/bin/../libs/jackson-core-
2.9.7.jar:/usr/local/kafka/bin/../libs/jackson-databind-2.9.7.jar:/usr/local/kafka/bin/../libs/jackson-jaxrs-base-
2.9.7.jar:/usr/local/kafka/bin/../libs/jackson-jaxrs-json-provider-2.9.7.jar:/usr/local/kafka/bin/../libs/jackson-module-jaxb-annotations-
2.9.7.jar:/usr/local/kafka/bin/../libs/javassist-3.22.0-CR2.jar:/usr/local/kafka/bin/../libs/javax.annotation-api-
1.2.jar:/usr/local/kafka/bin/../libs/javax.inject-1.jar:/usr/local/kafka/bin/../libs/javax.inject-2.5.0-
b42.jar:/usr/local/kafka/bin/../libs/javax.servlet-api-3.1.0.jar:/usr/local/kafka/bin/../libs/javax.ws.rs-api-
2.1.1.jar:/usr/local/kafka/bin/../libs/javax.ws.rs-api-2.1.jar:/usr/local/kafka/bin/../libs/jaxb-api-
2.3.0.jar:/usr/local/kafka/bin/../libs/jersey-client-2.27.jar:/usr/local/kafka/bin/../libs/jersey-common-
2.27.jar:/usr/local/kafka/bin/../libs/jersey-container-servlet-2.27.jar:/usr/local/kafka/bin/../libs/jersey-container-servlet-core-
2.27.jar:/usr/local/kafka/bin/../libs/jersey-hk2-2.27.jar:/usr/local/kafka/bin/../libs/jersey-media-jaxb-
2.27.jar:/usr/local/kafka/bin/../libs/jersey-server-2.27.jar:/usr/local/kafka/bin/../libs/jetty-client-
9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-continuation-9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-http-
9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-io-9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-security-
9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty server-9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-servlet-
9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-servlets-9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jetty-util-
9.4.12.v20180830.jar:/usr/local/kafka/bin/../libs/jopt-simple-5.0.4.jar:/usr/local/kafka/bin/../libs/kafka_2.12-
2.1.0.jar:/usr/local/kafka/bin/../libs/kafka_2.12-2.1.0-sources.jar:/usr/local/kafka/bin/../libs/kafka-clients-
2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-log4j-appender-2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-streams-
2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-streams-examples-2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-streams-scala_2.12-
2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-streams-test-utils-2.1.0.jar:/usr/local/kafka/bin/../libs/kafka-tools-
2.1.0.jar:/usr/local/kafka/bin/../libs/log4j-1.2.17.jar:/usr/local/kafka/bin/../libs/lz4-java-1.5.0.jar:/usr/local/kafka/bin/../libs/maven-
artifact-3.5.4.jar:/usr/local/kafka/bin/../libs/metrics-core-2.2.0.jar:/usr/local/kafka/bin/../libs/osgi-resource-locator-
1.0.1.jar:/usr/local/kafka/bin/../libs/plexus-utils-3.1.0.jar:/usr/local/kafka/bin/../libs/reflections-
0.9.11.jar:/usr/local/kafka/bin/../libs/rocksdbjni-5.14.2.jar:/usr/local/kafka/bin/../libs/scala-library-
2.12.7.jar:/usr/local/kafka/bin/../libs/scala-logging_2.12-3.9.0.jar:/usr/local/kafka/bin/../libs/scala-reflect-
2.12.7.jar:/usr/local/kafka/bin/../libs/slf4j-api-1.7.25.jar:/usr/local/kafka/bin/../libs/slf4j-log4j12-
1.7.25.jar:/usr/local/kafka/bin/../libs/snappy-java-1.1.7.2.jar:/usr/local/kafka/bin/../libs/validation-api-
1.1.0.Final.jar:/usr/local/kafka/bin/../libs/zkclient-0.10.jar:/usr/local/kafka/bin/../libs/zookeeper-
3.4.13.jar:/usr/local/kafka/bin/../libs/zstd-jni-1.3.5-4.jar kafka.Kafka /usr/local/kafka/config/server-1.properties
```

The topic can also be described:

```
sudo /usr/local/kafka/bin/kafka-topics.sh --describe --topic testing-cluster --bootstrap-server <Your_IP_or_DNS>:9092

Topic:testing-cluster   PartitionCount:2        ReplicationFactor:2     Configs:
        Topic: testing-cluster  Partition: 0    Leader: 1       Replicas: 1,0   Isr: 1,0
        Topic: testing-cluster  Partition: 1    Leader: 0       Replicas: 0,1   Isr: 0,1
```

M. Producing messages to a cluster of brokers

Remember that:

```
/usr/local/kafka/bin/kafka-console-producer.sh --broker-list <YourIP_or_DNS>:9093,<YourIP_or_DNS>:9094 --
topic testing-cluster
```

N. Consuming messages from a cluster of brokers

Remember that:

```
/usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server <YourIP_or_DNS>:9093,<YourIP_or_DNS>:9094 -
-topic testing-cluster
```

0. Creating N brokers in different EC2 VMs

1. Create multiple EC2 VMs adding the following inbound rules for ports: 2888 and 3888
   Hint: to facilitate the creation of multiple instance you can create an image from one instance. Then, when you launch new instances select your new image. With this approach all the kafka installation procedure is performed only once.

2. Install zookeeper and Kafka in each EC2 VM

3. If previously started, then stop the Zookeeper and Kafka servers in each EC2VM

4. In each EC2 VM change Kafka server configuration file in **/usr/local/kafka/config/server.properties** accordingly:

```
# the broker ID starting on 0 (this is for broker 2 within a 3 brokers cluster)
broker.id=2

# your EC2 VM name to be accessed from outside
listeners=PLAINTEXT://ec2-54-236-47-54.compute-1.amazonaws.com:9092

# replication factor accordingly with the number of brokers
offsets.topic.replication.factor=3
transaction.state.log.replication.factor=3
transaction.state.log.min.isr=3

#zookeeper connectivity (one per EC2 VM of this cluster)
zookeeper.connect=ec2-54-90-57-82.compute-1.amazonaws.com:2181,ec2-54-173-171-63.compute-
1.amazonaws.com:2181,ec2-54-236-47-54.compute-1.amazonaws.com:2181
```

5. In each EC2 VM change the Zookeeper server configuration file in **/usr/local/zookeeper/conf/zoo.cfg** accordingly:

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
maxClientCnxns=60
initLimit=10
syncLimit=5
server.1=ec2-54-90-57-82.compute-1.amazonaws.com:2888:3888
server.2=ec2-54-173-171-63.compute-1.amazonaws.com:2888:3888
server.3=ec2-54-236-47-54.compute-1.amazonaws.com:2888:3888
```

6. In each EC2 VM create a file with the ID of the Zookeeper server in **/var/lib/zookeeper/myid** containing only the ID of each specific Zookeeper server, example:

```
[ec2-user@ip-172-31-50-87 conf]$ cd /var/lib/zookeeper/
[ec2-user@ip-172-31-50-87 zookeeper]$ ls -ltr
total 8
-rw-r--r-- 1 root root   2 Feb  2 14:45 myid
-rw-r--r-- 1 root root   4 Feb  2 16:14 zookeeper_server.pid
drwxr-xr-x 2 root root 273 Feb  2 16:19 version-2
[ec2-user@ip-172-31-50-87 zookeeper]$ more myid
3
```

7. In each EC2 VM start the Kafka and Zookeeper servers.
8. Check the available topics:

```
sudo /usr/local/kafka/bin/kafka-topics.sh --list --bootstrap-server ec2-54-90-57-82.compute-
1.amazonaws.com:9092
```

9. Create a new topic, for example in a 3 broker cluster:

```
sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server ec2-54-173-171-63.compute-
1.amazonaws.com:9092 -replication-factor 3 --partitions 6 --topic PURCHASE
```

10. Check how the topic has been created:

```
/usr/local/kafka/bin/kafka-topics.sh --describe --bootstrap-server ec2-54-236-47-54.compute-
1.amazonaws.com:9092 --topic PURCHASE
```

11. Start a new consumer (in a different machine, VM or session) for the new topic using the predefined cluster:

```
/usr/local/kafka/bin/kafka-console-producer.sh --broker-list ec2-54-90-57-82.compute-
1.amazonaws.com:9092,ec2-54-173-171-63.compute-1.amazonaws.com:9092,ec2-54-236-47-54.compute-
1.amazonaws.com:9092 --topic PURCHASE
```

12. Start a new producer (in a different machine, VM or session) for the new topic using the predefined cluster and start sending messaging. Verify if they are consumed correctly:

```
/usr/local/kafka/bin/kafka-console-producer.sh --broker-list ec2-54-90-57-82.compute-
1.amazonaws.com:9092,ec2-54-173-171-63.compute-1.amazonaws.com:9092,ec2-54-236-47-54.compute-
1.amazonaws.com:9092 --topic PURCHASE
```

13. Try to stop one Kafka server from the cluster and recheck the topic configuration with:

```
/usr/local/kafka/bin/kafka-topics.sh --describe --bootstrap-server ec2-54-236-47-54.compute-
1.amazonaws.com:9092 --topic PURCHASE
```

14. Restart the Kafka server and execute the kafka-leader-election to refresh the topic' cluster configuration:

```
/usr/local/kafka/bin/kafka-leader-election.sh --bootstrap-server ec2-54-90-57-82.compute-
1.amazonaws.com:9092,ec2-54-173-171-63.compute-1.amazonaws.com:9092,ec2-54-236-47-54.compute-
1.amazonaws.com:9092 --election-type preferred --all-topic-partitions
```

Hints: if needed the Kafka server logs are located at: /usr/local/kafka/logs/kafkaServer.out where you can see the execution logging.

If you are using already used kafka servers, to make them as a cluster, you may need to remove all old data. Simply remove the /tmp/kafka-logs directory, and recreate it empty.

P. Broker failure test

- Launch the cluster with 2 brokers
- Create a new topic with 4 partitions

```
sudo /usr/local/kafka/bin/kafka-topics.sh --create --bootstrap-server <YourIP_or_DNS>:9092 -replication-
factor 2 --partitions 4 --topic events

Created topic "events".
```

- Check who are the leaders:

```
/usr/local/kafka/bin/kafka-topics.sh --describe --bootstrap-server <YourIP_or_DNS>:9092 --topic events

Topic:events    PartitionCount:4        ReplicationFactor:2     Configs:
        Topic: events   Partition: 0    Leader: 0       Replicas: 0,1   Isr: 0,1
        Topic: events   Partition: 1    Leader: 1       Replicas: 1,0   Isr: 1,0
        Topic: events   Partition: 2    Leader: 0       Replicas: 0,1   Isr: 0,1
        Topic: events   Partition: 3    Leader: 1       Replicas: 1,0   Isr: 1,0
```

- Produce message to the topic

```
/usr/local/kafka/bin/kafka-console-producer.sh --broker-list <YourIP_or_DNS>:9093, <YourIP_or_DNS>:9094 --
topic events
```

- Consume message from the topic <*start two different consumer sessions*>

```
/usr/local/kafka/bin/kafka-console-consumer.sh --bootstrap-server <YourIP_or_DNS>:9093, <YourIP_or_DNS>:9094
--topic events --group group-id-teste-cluster
```

- Check that messages are being balanced between the two consumers

- Stop broker id 0 with the following command:

```
sudo kill `ps aux | grep java | grep server-0.properties | tr -s " " | cut -d " " -f2`
```

- Check the change in the leaders of the cluster

```
/usr/local/kafka/bin/kafka-topics.sh --describe --bootstrap-server <YourIP_or_DNS>:9092 --topic events
```

```
Topic:events    PartitionCount:4        ReplicationFactor:2     Configs:
        Topic: events   Partition: 0    Leader: 1       Replicas: 0,1   Isr: 1
        Topic: events   Partition: 1    Leader: 1       Replicas: 1,0   Isr: 1
        Topic: events   Partition: 2    Leader: 1       Replicas: 0,1   Isr: 1
        Topic: events   Partition: 3    Leader: 1       Replicas: 1,0   Isr: 1
```

- Check that messages are still being balanced between the two consumers

Appendix: URLs with other resources

- https://github.com/confluentinc/confluent-kafka-python/issues/187
- http://cloudurable.com/blog/kafka-tutorial-kafka-failover-kafka-cluster/index.html
- if kafka brokers IDs are corrupted: https://stackoverflow.com/questions/59592518/kafka-broker-doesnt-find-cluster-id-and-creates-new-one-after-docker-restart

DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Tutorial – Kakfa Streaming in AWS

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to introduce the fundamentals of stream processing using Kafka. Three examples taken from: **Narkhede, N., Shapira, G., & Palino, T. (2017). Kafka: the definitive guide: real-time data and stream processing at scale. " O'Reilly Media, Inc."**, and are used to illustrate (i) how to develop the source code and (ii) how to test it.

First example is a simple word count example that is used to demonstrate the map/filter pattern and simple aggregates.

Second example contains a calculation of different statistics on stock market trades, which will allow us to demonstrate window aggregations.

Third example demonstrates the streaming joins.

The following contents is presented in this document.

Contents

A. Installing git in AWS EC2

In command line:

```
sudo yum install git
```

Check installation:

```
git --version
git version 2.47.1 (or later)
```

B. Installing mvn in AWS EC2

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 "17.0.14" 2025-01-21 LTS
OpenJDK Runtime Environment Corretto-17.0.14.7.1 (build 17.0.14+7-LTS)
OpenJDK 64-Bit Server VM Corretto-17.0.14.7.1 (build 17.0.14+7-LTS, mixed mode, sharing)
[ec2-user@ip-172-31-85-55 ~]$ javac -version
javac 17.0.14
```

Hint 1: if not available, use the following commands or similar:

```
[ec2-user@ip- ~]$ yum search java-17
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
================================================================================ N/S matched: java-
17 =================================================================================
java-17-amazon-corretto.x86_64 : Amazon Corretto development environment
java-17-amazon-corretto-debugsymbols.x86_64 : Amazon Corretto 17 zipped debug symbols
java-17-amazon-corretto-devel.x86_64 : Amazon Corretto 17 development tools
java-17-amazon-corretto-headless.x86_64 : Amazon Corretto headless development environment
java-17-amazon-corretto-javadoc.x86_64 : Amazon Corretto 17 API documentation
java-17-amazon-corretto-jmods.x86_64 : Amazon Corretto 17 jmods

[ec2-user@ip- ~]$ sudo yum -y install java-17-amazon-corretto-devel.x86_64
```

3. Execute the following commands in the EC2 instance:

```
[ec2-user@ip- - - -  ~]$ sudo wget https://repo.maven.apache.org/maven2/org/apache/maven/apache-
maven/3.9.9/apache-maven-3.9.9-bin.tar.gz

[ec2-user@ip- - - -  ~]$ tar xzvf apache-maven-3.9.9-bin.tar.gz

[ec2-user@ip- - - -  ~]$ cd apache-maven-3.9.9/bin

[ec2-user@ip- - - -  bin]$ export PATH=/home/ec2-user/apache-maven-3.9.9/bin:$PATH

[ec2-user@ip- - - -  ~]$ mvn -version
Maven home: /home/ec2-user/apache-maven-3.9.9
Java version: 17.0.14, vendor: Amazon.com Inc., runtime: /usr/lib/jvm/java-17-amazon-corretto.x86 64
Default locale: en_US, platform encoding: ANSI_X3.4-1968
OS name: "linux", version: "5.10.235-227.919.amzn2.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
```

C. Testing the Word count example

C.1. Create working directory

```
mkdir WordCount
```

C.2. Change to working directory

```
cd WordCount
```

C.3. Clone the source code from git

```
[ec2-user@ip- WordCount]$ git clone https://github.com/gwenshap/kafka-streams-wordcount
Cloning into 'kafka-streams-wordcount'...
remote: Enumerating objects: 57, done.
remote: Total 57 (delta 0), reused 0 (delta 0), pack-reused 57
Unpacking objects: 100% (57/57), done.
```

C.4. Change directory

```
cd kafka-streams-wordcount/
```

C.5. Change pom.xml from:

```
<dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>0.10.3.0-SNAPSHOT</version>
</dependency>
```

To a most recent library[1]:

```
<dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>3.2.0</version>
</dependency>
```

C.6. Study the following source code that is also located in:

```
/home/ec2-user/WordCount/kafka-streams-wordcount/src/main/java/com/shapira/examples/streams/wordcount/
WordCountExample.java
```

```
1   package com.shapira.examples.streams.wordcount;


2   import org.apache.kafka.clients.CommonClientConfigs;
3   import org.apache.kafka.clients.consumer.ConsumerConfig;
4   import org.apache.kafka.common.serialization.Serdes;
5   import org.apache.kafka.streams.KafkaStreams;
6   import org.apache.kafka.streams.KeyValue;
7   import org.apache.kafka.streams.StreamsConfig;
8   import org.apache.kafka.streams.kstream.KStream;
9   import org.apache.kafka.streams.kstream.KStreamBuilder;

10  import java.util.Arrays;
11  import java.util.Properties;
12  import java.util.regex.Pattern;

13  public class WordCountExample {

14  public static void main(String[] args) throws Exception{

15  Properties props = new Properties();
16  props.put(StreamsConfig.APPLICATION_ID_CONFIG, "wordcount");
17  props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
18  props.put(StreamsConfig.KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
19  props.put(StreamsConfig.VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
```

[1] The most recent libraries should be consulted at https://mvnrepository.com/

```
20  // setting offset reset to earliest so that we can re-run the demo code with the same pre-loaded data
21  // Note: To re-run the demo, you need to use the offset reset tool:
22  // https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Streams+Application+Reset+Tool
23  props.put(ConsumerConfig.AUTO OFFSET RESET CONFIG, "earliest");

24  // work-around for an issue around timing of creating internal topics
25  // Fixed in Kafka 0.10.2.0
26  // don't use in large production apps - this increases network load
27  // props.put(CommonClientConfigs.METADATA MAX AGE CONFIG, 500);

28  KStreamBuilder builder = new KStreamBuilder();

29  KStream<String, String> source = builder.stream("wordcount-input");


30  final Pattern pattern = Pattern.compile("\\W+");
31  KStream counts  = source.flatMapValues(value-> Arrays.asList(pattern.split(value.toLowerCase())))
32  .map((key, value) -> new KeyValue<Object, Object>(value, value))
33  .filter((key, value) -> (!value.equals("the")))
34  .groupByKey()
35  .count("CountStore").mapValues(value->Long.toString(value)).toStream();
36  counts.to("wordcount-output");

37  KafkaStreams streams = new KafkaStreams(builder, props);

38  // This is for reset to work. Don't use in production - it causes the app to re-load the state from Kafka
    on every start
39  streams.cleanUp();

40  streams.start();

41  // usually the stream application would be running forever,
42  // in this example we just let it run for some time and stop since the input data is finite.
43  Thread.sleep(5000L);

44  streams.close();

45  }
46  }
```

C.7. Change the location of your kafka server in the java source code:

```
/home/ec2-user/WordCount/kafka-streams-wordcount/src/main/java/com/shapira/examples/streams/wordcount/
WordCountExample.java
```

In the following source code line:

```
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "<YOUR_DNS_NAME>:9092");
```

C.8. Build the project with mvn package, this will generate an uber-jar with the streams app and all its dependencies.

```
mvn package
```

C.9. Create a wordcount-input topic:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server <YOUR DNS NAME>:9092 --create --topic wordcount-input
--partitions 1 --replication-factor 1
```

C.10. Produce some text to the topic. Don't forget to repeat words (so we can count higher than 1) and to use the word "the", so we can filter it.

```
/usr/local/kafka/bin/kafka-console-producer.sh --broker-list <YOUR_DNS_NAME>:9092 --topic wordcount-input
```

C.11. Run the app:

```
java -cp target/uber-kafka-streams-wordcount-1.0-SNAPSHOT.jar
com.shapira.examples.streams.wordcount.WordCountExample
```

C.12. View the results obtained:

```
/usr/local/kafka/bin/kafka-console-consumer.sh --topic wordcount-output --from-beginning --bootstrap-server
<YOUR_DNS_NAME>:9092 --property print.key=true


You will receive something similar with:
um      1
este    2
outro   1
testte  1teste  1
tets    1
test    4
        1
a       1
```

C.13. If you want to reset state and re-run the application (maybe with some changes?) on existing input topic, you can stop the kafka producer and consumer and then:

Reset internal topics (used for shuffle the topic and state-stores):

```
/usr/local/kafka/bin/kafka-streams-application-reset.sh --application-id wordcount --bootstrap-servers
<YOUR_DNS_NAME>:9092 --input-topics wordcount-input
```

(optional) Delete the output topic:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server <YOUR DNS NAME>:9092 --delete --topic wordcount-
output
```

D. Testing the Stock Market Statistics example

D.1. Create working directory

```
mkdir StockMarketStatistics
```

D.2. Change to working directory

```
cd StockMarketStatistics
```

D.3. Clone the source code from git

```
[ec2-user@ip- StockMarketStatistics]$ git clone https://github.com/gwenshap/kafka-streams-stockstats
Cloning into 'kafka-streams-stockstats'...
remote: Enumerating objects: 173, done.
remote: Total 173 (delta 0), reused 0 (delta 0), pack-reused 173
Receiving objects: 100% (173/173), 27.77 KiB | 3.08 MiB/s, done.
Resolving deltas: 100% (45/45), done.
```

D.4. Change pom.xml from:

```
    <properties>
        <kafka.version>2.8.0</kafka.version>
        <confluent.version>6.1.0</confluent.version>
        <avro.version>1.8.2</avro.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
```

To a most recent library[2]:

```
    <properties>
        <kafka.version>3.2.0</kafka.version>
        <confluent.version>6.1.0</confluent.version>
        <avro.version>1.8.2</avro.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
```

D.5. Study the following source code that is also located in:

```
/home/ec2-user/StockMarketStatistics/kafka-streams-stockstats/src/main/java/com/Shapira/examples/streams/
stockstats/StockStatsExample.java
```

```
1   package com.shapira.examples.streams.stockstats;

2   import com.shapira.examples.streams.stockstats.serde.JsonDeserializer;
3   import com.shapira.examples.streams.stockstats.serde.JsonSerializer;
4   import com.shapira.examples.streams.stockstats.serde.WrapperSerde;
5   import com.shapira.examples.streams.stockstats.model.Trade;
6   import com.shapira.examples.streams.stockstats.model.TradeStats;
7   import org.apache.kafka.clients.admin.AdminClient;
8   import org.apache.kafka.clients.admin.DescribeClusterResult;
9   import org.apache.kafka.clients.consumer.ConsumerConfig;
10  import org.apache.kafka.common.serialization.Serdes;
11  import org.apache.kafka.common.utils.Bytes;
12  import org.apache.kafka.streams.KafkaStreams;
13  import org.apache.kafka.streams.StreamsConfig;
14  import org.apache.kafka.streams.Topology;
15  import org.apache.kafka.streams.kstream.KStream;
16  import org.apache.kafka.streams.StreamsBuilder;
17  import org.apache.kafka.streams.kstream.Materialized;
18  import org.apache.kafka.streams.kstream.Produced;
19  import org.apache.kafka.streams.kstream.TimeWindows;
20  import org.apache.kafka.streams.kstream.Windowed;
21  import org.apache.kafka.streams.kstream.WindowedSerdes;
22  import org.apache.kafka.streams.state.WindowStore;
```

[2] The most recent libraries should be consulted at https://mvnrepository.com/

```
23 import java.util.Properties;

24 /**
   Input is a stream of trades
   Output is two streams: One with minimum and avg "ASK" price for every 10 seconds window
   Another with the top-3 stocks with lowest minimum ask every minute
25 */
26 public class StockStatsExample {

27 public static void main(String[] args) throws Exception {

28 Properties props;
29 if (args.length==1)
30 props = LoadConfigs.loadConfig(args[0]);
31 else
32 props = LoadConfigs.loadConfig();

33 props.put(StreamsConfig.APPLICATION_ID_CONFIG, "stockstat-2");
34 props.put(StreamsConfig.DEFAULT KEY SERDE CLASS CONFIG, Serdes.String().getClass().getName());
35 props.put(StreamsConfig.DEFAULT VALUE SERDE CLASS CONFIG, TradeSerde.class.getName());

36 // setting offset reset to earliest so that we can re-run the demo code with the same pre-loaded data
37 // Note: To re-run the demo, you need to use the offset reset tool:
38 // https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Streams+Application+Reset+Tool
39 props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

40 // creating an AdminClient and checking the number of brokers in the cluster, so I'll know how many
   replicas we want...

41 AdminClient ac = AdminClient.create(props);
42 DescribeClusterResult dcr = ac.describeCluster();
43 int clusterSize = dcr.nodes().get().size();

44 if (clusterSize<3)
45 props.put("replication.factor",clusterSize);
46 else
47 props.put("replication.factor",3);

48 StreamsBuilder builder = new StreamsBuilder();

49 KStream<String, Trade> source = builder.stream(Constants.STOCK_TOPIC);

50 KStream<Windowed<String>, TradeStats> stats = source
51 .groupByKey()
52 .windowedBy(TimeWindows.of(5000).advanceBy(1000))
53 .<TradeStats>aggregate(() -> new TradeStats(),(k, v, tradestats) -> tradestats.add(v),
54 Materialized.<String, TradeStats, WindowStore<Bytes, byte[]>>as("trade-aggregates")
55 .withValueSerde(new TradeStatsSerde()))
56 .toStream()
57 .mapValues((trade) -> trade.computeAvgPrice());

58 stats.to("stockstats-output", Produced.keySerde(WindowedSerdes.timeWindowedSerdeFrom(String.class)));

59 Topology topology = builder.build();

60 KafkaStreams streams = new KafkaStreams(topology, props);

61 System.out.println(topology.describe());

62 streams.cleanUp();

63 streams.start();

64 // Add shutdown hook to respond to SIGTERM and gracefully close Kafka Streams
65 Runtime.getRuntime().addShutdownHook(new Thread(streams::close));

66 }

67 static public final class TradeSerde extends WrapperSerde<Trade> {
68 public TradeSerde() {
69 super(new JsonSerializer<Trade>(), new JsonDeserializer<Trade>(Trade.class));
70 }
71 }

72 static public final class TradeStatsSerde extends WrapperSerde<TradeStats> {
73 public TradeStatsSerde() {
74 super(new JsonSerializer<TradeStats>(), new JsonDeserializer<TradeStats>(TradeStats.class));
75 }
76 }

77 }
```

D.6. Change directory

```
cd kafka-streams-stockstats
```

D.7. Build the project with mvn package, this will generate an uber-jar with the streams app and all its dependencies.

```
mvn package
```

D.8. Change to working directory

```
cd kafka-streams-stockstats
```

D.9. Create a stocks input topic and output topic:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic stocks --partitions 1
--replication-factor 1

/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic stockstats-output --
partitions 1 --replication-factor 1
```

D.10. Create the configuration file, next to you JAVA project, with the command:

```
cat > configfile << EOF
```

and then write the following content to the file directly in the command line:

```
bootstrap.servers = PLAINTEXT://<YOUR DNS NAME>:9092
EOF
```

D.11. Generate some trades so we can analyze them. Start running the trades producer and stop it with ctrl-c when you think there's enough data:

```
java -cp target/uber-kafka-streams-stockstats-1.1-SNAPSHOT.jar -DLOGLEVEL=INFO
com.shapira.examples.streams.stockstats.StockGenProducer configfile
```

D.12. Run the streams app:

```
java -cp target/uber-kafka-streams-stockstats-1.1-SNAPSHOT.jar -DLOGLEVEL=INFO
com.shapira.examples.streams.stockstats.StockStatsExample configfile
```

D.13. Check the results:

```
/usr/local/kafka/bin/kafka-console-consumer.sh --topic stockstats-output --from-beginning --bootstrap-server
<YOUR_DNS_NAME>:9092 --property print.key=true
```

D.14. If you want to reset state and re-run the application (maybe with some changes?) on existing input topic, you can stop the kafka producer and consumer and then:

Reset internal topics (used for shuffle the topic and state-stores):

```
/usr/local/kafka/bin/kafka-streams-application-reset.sh --application-id wordcount --bootstrap-servers
<YOUR_DNS_NAME>:9092 --input-topics stocks
```

(optional) Delete the output topic:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server <YOUR_DNS_NAME>:9092 --delete --topic stockstats-
output
```

E. Testing the click Stream Enrichment example

E.1. Proceed accordingly with the previous examples:

```
[ec2-user@ip- ~]$ mkdir StreamEnrichment

[ec2-user@ip- ~]$ cd StreamEnrichment/

[ec2-user@ip- StreamEnrichment]$ git clone https://github.com/gwenshap/kafka-clickstream-enrich
Cloning into 'kafka-clickstream-enrich'...
remote: Enumerating objects: 65, done.
remote: Total 65 (delta 0), reused 0 (delta 0), pack-reused 65
Unpacking objects: 100% (65/65), done.

[ec2-user@ip- StreamEnrichment]$ cd kafka-clickstream-enrich
```

E.2. Study the following source code that is also located in:

```
/home/ec2-user/StreamEnrichment/kafka-clickstream-enrich/src/main/java/com/shapira/examples/streams/
clickstreamenrich/ClickstreamEnrichment.java
```

```
1   package com.shapira.examples.streams.clickstreamenrich;
2   import com.shapira.examples.streams.clickstreamenrich.model.PageView;
3   import com.shapira.examples.streams.clickstreamenrich.model.Search;
4   import com.shapira.examples.streams.clickstreamenrich.model.UserActivity;
5   import com.shapira.examples.streams.clickstreamenrich.model.UserProfile;
6   import com.shapira.examples.streams.clickstreamenrich.serde.JsonDeserializer;
7   import com.shapira.examples.streams.clickstreamenrich.serde.JsonSerializer;
8   import com.shapira.examples.streams.clickstreamenrich.serde.WrapperSerde;
9   import org.apache.kafka.clients.consumer.ConsumerConfig;
10  import org.apache.kafka.common.serialization.Serdes;
11  import org.apache.kafka.streams.KafkaStreams;
12  import org.apache.kafka.streams.StreamsConfig;
13  import org.apache.kafka.streams.kstream.JoinWindows;
14  import org.apache.kafka.streams.kstream.KStream;
15  import org.apache.kafka.streams.kstream.KStreamBuilder;
16  import org.apache.kafka.streams.kstream.KTable;

17  import java.util.Properties;

18  public class ClickstreamEnrichment {

19  public static void main(String[] args) throws Exception {

1   Properties props = new Properties();
2   props.put(StreamsConfig.APPLICATION_ID_CONFIG, "clicks");
3   props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, Constants.BROKER);
4   // Since each step in the stream will involve different objects, we can't use default Serde

5   // setting offset reset to earliest so that we can re-run the demo code with the same pre-loaded data
6   // Note: To re-run the demo, you need to use the offset reset tool:
7   // https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Streams+Application+Reset+Tool
8   props.put(ConsumerConfig.AUTO OFFSET RESET CONFIG, "earliest");

9   // work-around for an issue around timing of creating internal topics
10  // this was resolved in 0.10.2.0 and above
11  // don't use in large production apps - this increases network load
12  // props.put(CommonClientConfigs.METADATA_MAX_AGE_CONFIG, 500);

13  KStreamBuilder builder = new KStreamBuilder();

14  KStream<Integer, PageView> views = builder.stream(Serdes.Integer(), new PageViewSerde(),
    Constants.PAGE_VIEW_TOPIC);
15  KTable<Integer, UserProfile> profiles = builder.table(Serdes.Integer(), new ProfileSerde(),
    Constants.USER PROFILE TOPIC, "profile-store");
16  KStream<Integer, Search> searches = builder.stream(Serdes.Integer(), new SearchSerde(),
    Constants.SEARCH_TOPIC);

17  KStream<Integer, UserActivity> viewsWithProfile = views.leftJoin(profiles,
18  (page, profile) -> {
19  if (profile != null)
```

```
20  return new UserActivity(profile.getUserID(), profile.getUserName(), profile.getZipcode(),
    profile.getInterests(), "", page.getPage());
21  else
22  return new UserActivity(-1, "", "", null, "", page.getPage());
23  });

24  KStream<Integer, UserActivity> userActivityKStream = viewsWithProfile.leftJoin(searches,
25  (userActivity, search) -> {
26  if (search != null)
27  userActivity.updateSearch(search.getSearchTerms());
28  else
29  userActivity.updateSearch("");
30  return userActivity;
31  },
32  JoinWindows.of(1000), Serdes.Integer(), new UserActivitySerde(), new SearchSerde());

33  userActivityKStream.to(Serdes.Integer(), new UserActivitySerde(), Constants.USER_ACTIVITY_TOPIC);

34  KafkaStreams streams = new KafkaStreams(builder, props);

35  streams.cleanUp();

36  streams.start();

37  // usually the stream application would be running forever,
38  // in this example we just let it run for some time and stop since the input data is finite.
39  Thread.sleep(60000L);

40  streams.close();


41  }

42  static public final class PageViewSerde extends WrapperSerde<PageView> {
43  public PageViewSerde() {
44      super(new JsonSerializer<PageView>(), new JsonDeserializer<PageView>(PageView.class));
45  }
46  }

47  static public final class ProfileSerde extends WrapperSerde<UserProfile> {
48  public ProfileSerde() {
49      super(new JsonSerializer<UserProfile>(), new JsonDeserializer<UserProfile>(UserProfile.class));
50  }
51  }

52  static public final class SearchSerde extends WrapperSerde<Search> {
53  public SearchSerde() {
54      super(new JsonSerializer<Search>(), new JsonDeserializer<Search>(Search.class));
55  }
56  }

57  static public final class UserActivitySerde extends WrapperSerde<UserActivity> {
58  public UserActivitySerde() {
59       super(new JsonSerializer<UserActivity>(), new JsonDeserializer<UserActivity>(UserActivity.class));
60  }
61  }
62  }
```

E.3. Change pom.xml from:

```
<dependency>
       <groupId>org.apache.kafka</groupId>
       <artifactId>kafka-clients</artifactId>
       <version>0.10.3.0-SNAPSHOT</version>
</dependency>
<dependency>
       <groupId>org.apache.kafka</groupId>
       <artifactId>kafka-streams</artifactId>
       <version>0.10.3.0-SNAPSHOT</version>
</dependency>
```

To a most recent library:

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

```
        <version>3.2.0</version>
</dependency>
<dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-streams</artifactId>
        <version>3.2.0</version>
</dependency>
```

E.4. Change the location of your kafka server in the java source code:

```
/home/ec2-user/StreamEnrichment/kafka-clickstream-
enrich/src/main/java/com/shapira/examples/streams/clickstreamenrich/Constants.java
```

In the line 8:

```
public static final String BROKER = "<YOUR_DNS_NAME>:9092";
```

E.5. Build the project with mvn package:

```
mvn package
```

E.6. Generate some clicks, searches, and profiles. Run the generator. It should take about 5 seconds to run. Don't worry about complete lack of output:

```
java -cp target/uber-kafka-clickstream-enrich-1.0-SNAPSHOT.jar
com.shapira.examples.streams.clickstreamenrich.GenerateData
```

E.7. Run the streams app:

```
java -cp target/uber-kafka-clickstream-enrich-1.0-SNAPSHOT.jar
com.shapira.examples.streams.clickstreamenrich.ClickstreamEnrichment
```

E.8. Check the results:

```
/usr/local/kafka/bin/kafka-console-consumer.sh --topic clicks.user.activity --from-beginning --bootstrap-
server <YOUR_DNS_NAME>:9092 --property print.key=true
```

E.9. If you want to reset state and re-run the application (maybe with some changes?) on existing input topic, you can stop the kafka producer and consumer and then:

Reset internal topics (used for shuffle the topic and state-stores):

```
/usr/local/kafka/bin/kafka-streams-application-reset.sh --application-id clicks --bootstrap-servers
<YOUR_DNS_NAME>:9092 --input-topics clicks.user.profile,clicks.pages.views,clicks.search
```

(optional) Delete the output topic:

```
/usr/local/kafka/bin/kafka-topics.sh --bootstrap-server <YOUR_DNS_NAME>:9092 --delete --topic
clicks.user.activity
```

F. Other tools and references

- If you prefer, install emacs in AWS EC2 using the following command:

```
sudo yum install emacs
```

- Where to download apache maven https://maven.apache.org/download.cgi
- How to install apache maven https://maven.apache.org/install.html

DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Tutorial – Cloud Database and Automated Topic Kafka creation

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to show 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.

Scalability:
RDS virtualization could be scaled, creating different SCHEMAS, or DATABASES instances, for different Application services.

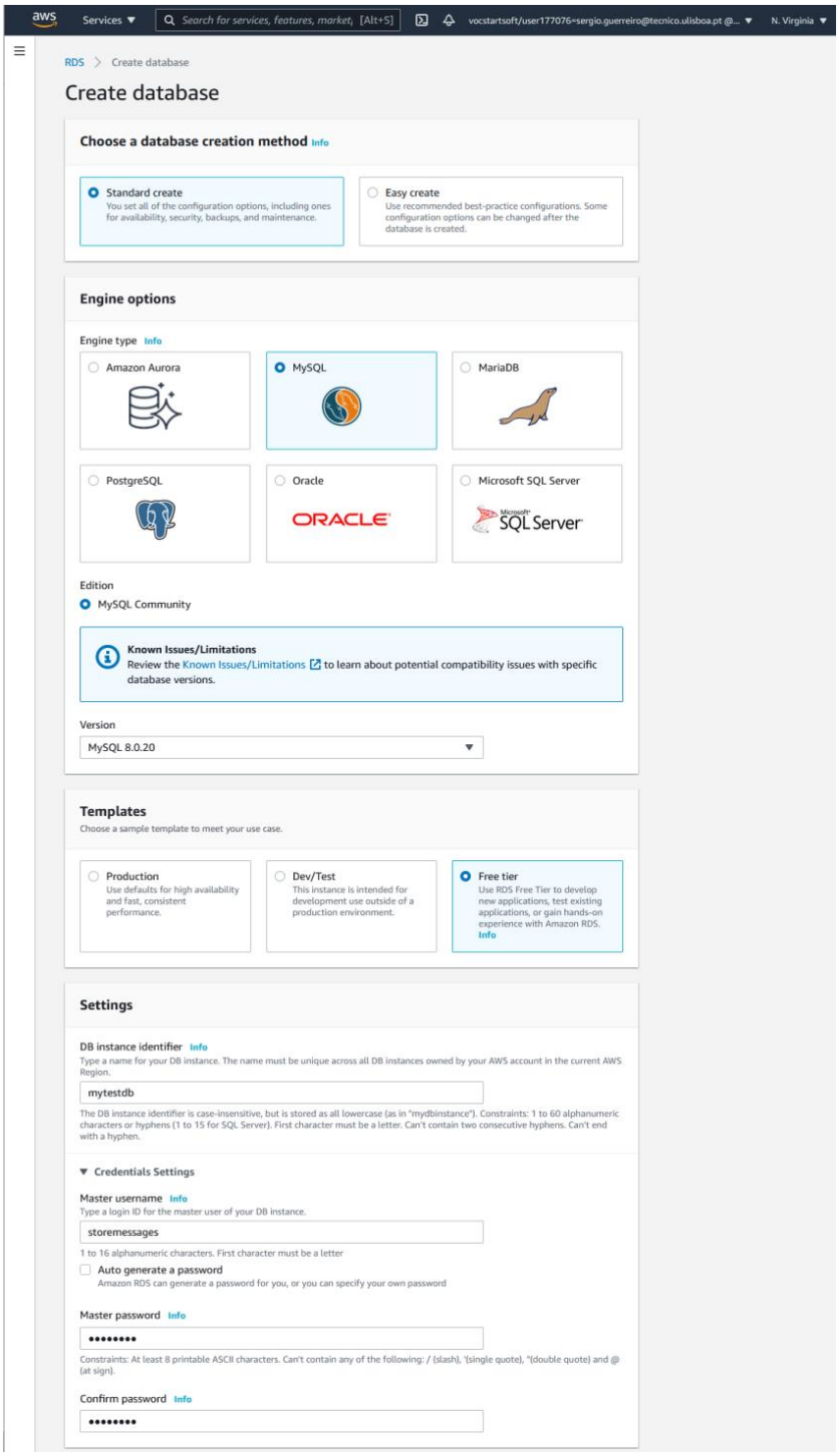
The following contents is presented in this document.

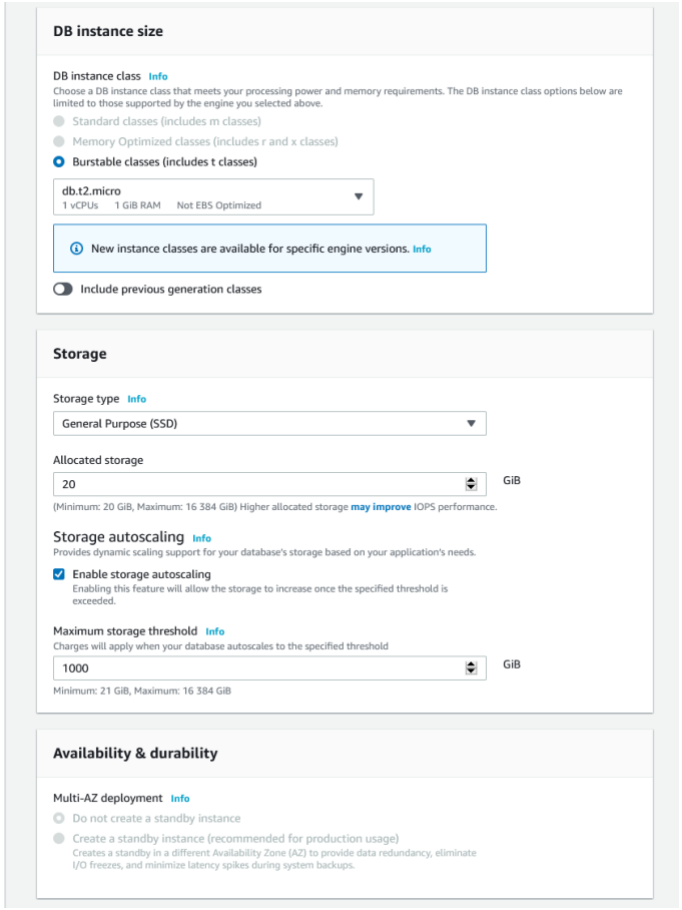
Contents

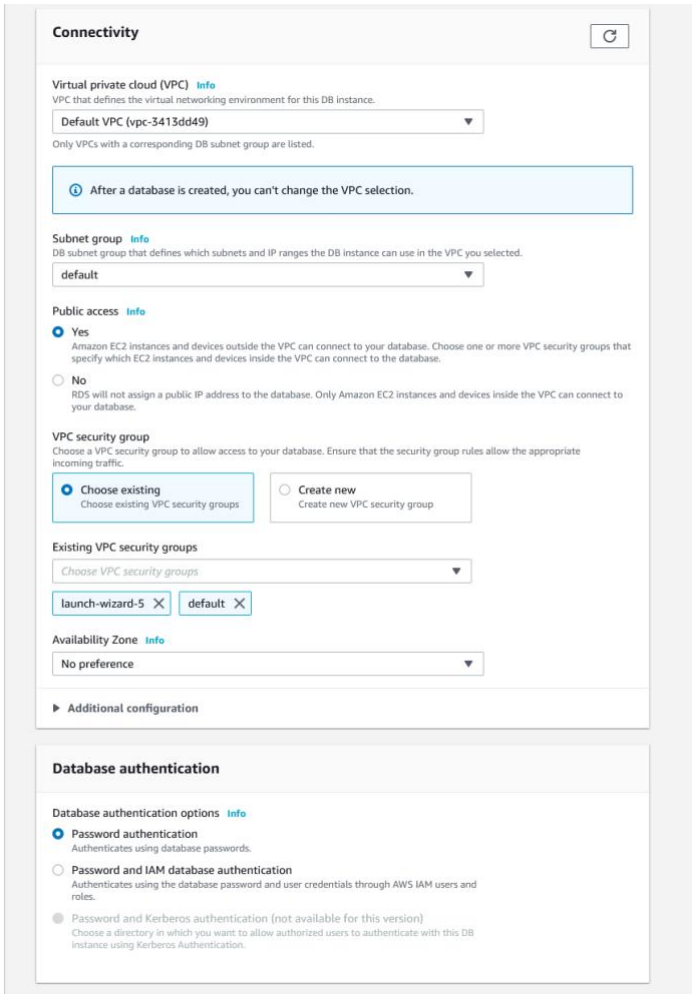
A. Create an AWS RDS Database, accessing it, and inserting data: example of

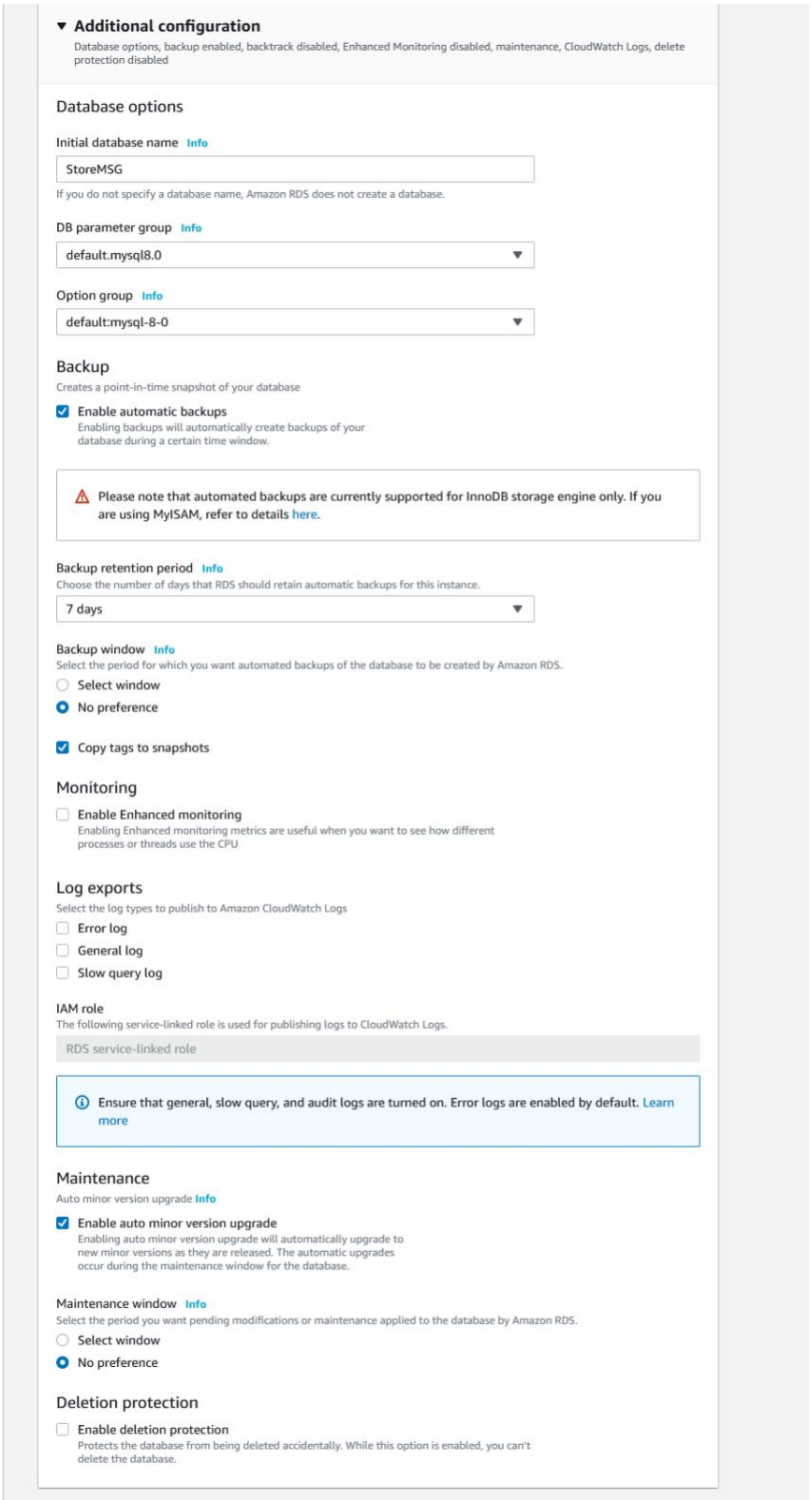
The goal of this section is to exemplify how to create your own cloud database in AWS, to access its contents through a database workbench, and then creating a JAVA microservice to consume Kafka messages that are then inserted in the cloud database.

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

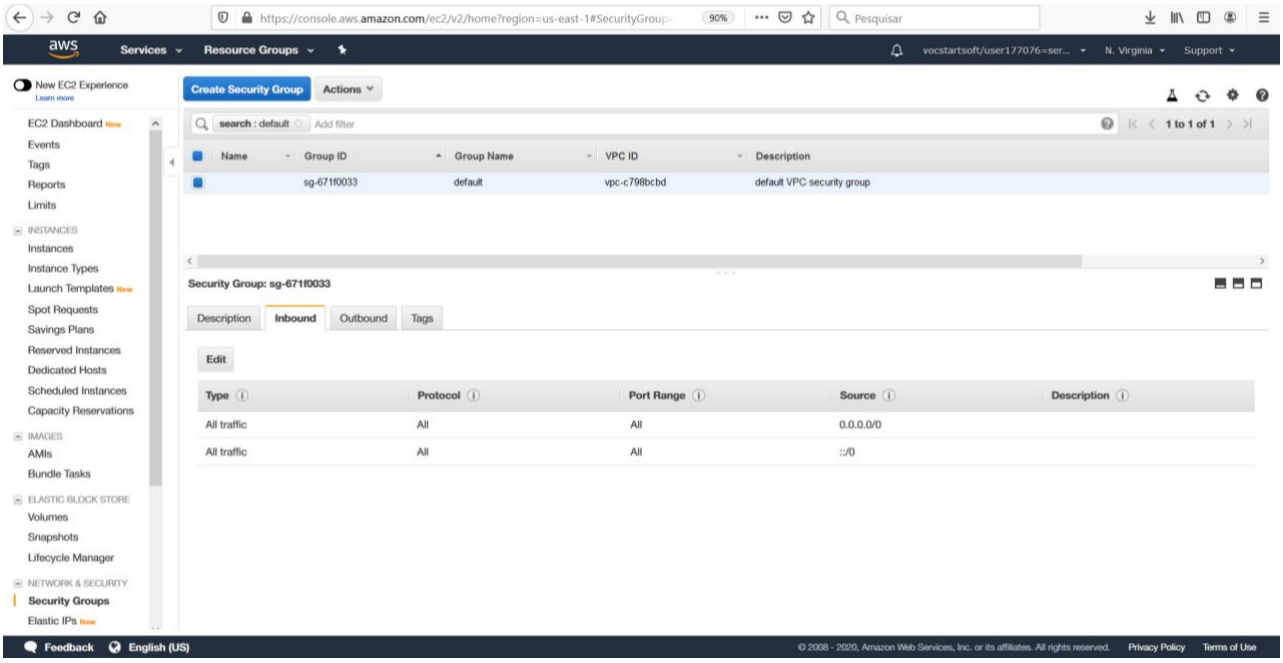



DB instance size
DB instance class Info
Choose a DB instance class that meets your processing power and memory requirements. The DB instance class options below are limited to those supported by the engine you selected above.
Standard classes (includes m classes)
Memory Optimized classes (includes r and x classes)
Burstable classes (includes t classes)
db.t2.micro
1 vCPUs 1 GiB RAM Not EBS Optimized
New instance classes are available for specific engine versions. Info
Include previous generation classes
Storage
Storage type Info
General Purpose (SSD)
Allocated storage
20
GiB
(Minimum: 20 GiB, Maximum: 16 384 GiB) Higher allocated storage may improve IOPS performance.
Storage autoscaling Info
Provides dynamic scaling support for your database's storage based on your application's needs.
Enable storage autoscaling
Enabling this feature will allow the storage to increase once the specified threshold is exceeded.
Maximum storage threshold Info
Charges will apply when your database autoscales to the specified threshold
1000
GiB
Minimum: 21 GiB, Maximum: 16 384 GiB
Availability & durability
Multi-AZ deployment Info
Do not create a standby instance
Create a standby instance (recommended for production usage)
Creates a standby in a different Availability Zone (AZ) to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.


Connectivity
Virtual private cloud (VPC) Info
VPC that defines the virtual networking environment for this DB instance.
Default VPC (vpc-3413dd49)
Only VPCs with a corresponding DB subnet group are listed.
After a database is created, you can't change the VPC selection.
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 access Info
Yes
Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more VPC security groups that specify which EC2 instances and devices inside the VPC can connect to the database.
No
RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside the VPC can connect to your database.
VPC security group
Choose a VPC security group to allow access to your database. Ensure that the security group rules allow the appropriate incoming traffic.
Choose existing
Choose existing VPC security groups
Create new
Create new VPC security group
Existing VPC security groups
Choose VPC security groups
launch-wizard-5
default
Availability Zone Info
No preference
Additional configuration
Database authentication
Database authentication options Info
Password authentication
Authenticates using database passwords.
Password and IAM database authentication
Authenticates using the database password and user credentials through AWS IAM users and roles.
Password and Kerberos authentication (not available for this version)
Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.



Hint: Later, check if you VPC configuration allows connections from anywhere in the internet. Choose the VPC Security groups and edit the configuration similarly with:



2. You can now 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));
```

DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Tutorial QUARKUS
Reactive Systems in JAVA

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to present the details related with the QUARKUS microservices framework.

The examples address the following aspects: environment set-up, creation of projects and adding elements to it, accessing databases, and controlling the database transaction, using messages in JSON format, how to use KAFKA, how to provide an OpenAPI interface, and finally, deploying in docker and deploying in AWS.

The following contents is presented in this document.

Contents

A. Setup steps for initial setup of QUARKUS environment

A.1. Clone this **repository** https://github.com/quarkusio/quarkus-quickstarts.git using GitHub desktop or web.

A.2. Install Maven **3.9.9** from https://archive.apache.org/dist/maven/maven-3/3.9.9/binaries/
(for detail on installing in different operating systems:
https://mkyong.com/maven/install-maven-on-mac-osx/
https://mkyong.com/maven/how-to-install-maven-in-windows/)

A.3. Install **docker** from https://www.docker.com/get-started and launch it.

A.4. Install **Mandrel** (for linux only) or **GraalVM** (other platforms) with **JDK 17** from https://github.com/graalvm/mandrel or https://www.oracle.com/java/technologies/downloads/#graalvmjava17 respectively
(for details on installing in different operating systems
https://github.com/graalvm/homebrew-tap
https://www.graalvm.org/latest/getting-started/linux/
https://www.graalvm.org/latest/getting-started/windows/)

A.5. Check java version with "javac -version" in command line, and change JDK if needed following these instructions:
https://mkyong.com/java/how-to-set-java_home-environment-variable-on-mac-os-x/
https://mkyong.com/java/how-to-set-java_home-on-windows-10/

A.6. In the repository base directory of step A.1. execute the following to assure that your environment is ok:

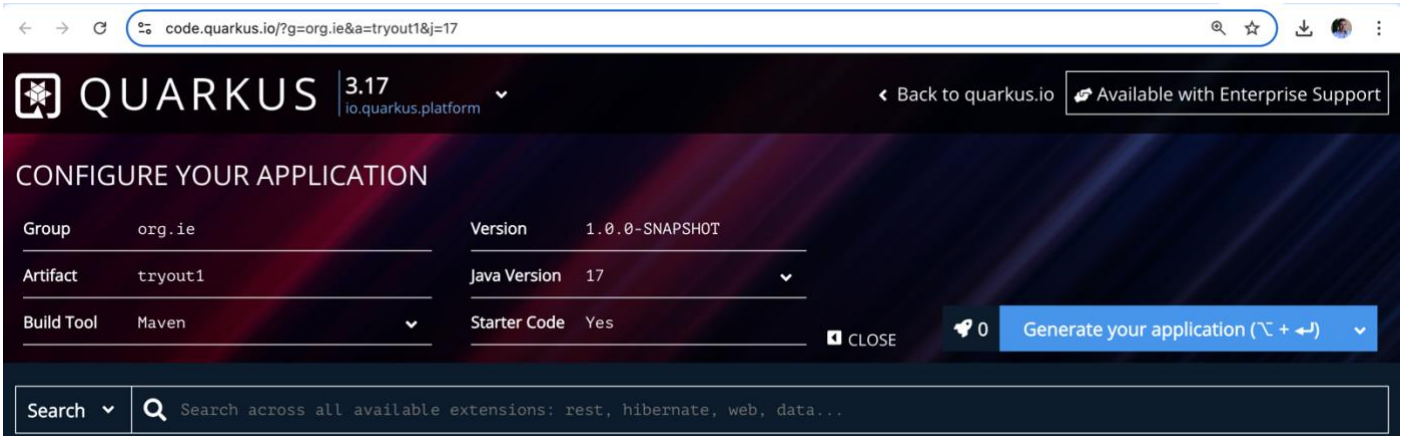
```
mvn clean install
```

A.7. (Optional) Install **mysql server** locally (optionally, for testing purposes of QUARKUS accessing data reactively) or, later, use AWS RDS.

A.8. (Optional) Install VS Code (https://code.visualstudio.com/) or other preferred IDE.

B. Create a Simple QUARKUS project with starter code

B.1. Navigate to the QUARKUS project creation website at https://code.quarkus.io and use the following configuration:



B.2. Generate, download the zip file and extract it to your local folder project

B.3. Import your newly created project to VS Code

B.4. Execute the command

```
./mvnw quarkus:dev
```

on command line or calling maven inside VS Code

B.5. Go to URL http://localhost:8080/hello and check the message that is produced.

B.6. Open the source code on your IDE, *e.g.*, VS Code and navigate to **GreetingResource.java** file. Study the source code and change the endpoint at line:

```
@Path("/hello")
```

To

```
@Path("/newapi")
```

B.7. Execute the command

```
./mvnw quarkus:dev
```

on command line or calling maven inside VS Code

B.8. Go to URL http://localhost:8080/newapi and http://localhost:8080/welcome and check the result that is produced.

C. Add a new JAVA class to and already existing QUARKUS project

C.1. Create a new class in your project with the name: **SecondClass.java**, in the same location as previous GreetingResource.java.

C.2. Update the new file SecondClass.java accordingly with this source code:

```
package org.ie;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

@Path("/api2")
public class SecondClass {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "Hello to the second API";
    }
}
```

Hint: you may also add a test class to automate the code verification

C.3. Execute the command

```
./mvnw quarkus:dev
```

on command line or calling maven inside VS Code

C.4. Go to URL http://localhost:8080/newapi and http://localhost:8080/api2 and check the result that is produced.

C.5. If needed, you can change the port, adding the following configuration in **application.properties** of your project (resources directory of your Quarkus project) :

```
quarkus.http.port=9000
```

D. QUARKUS exposed by OpenAPI interface

The aim of this example is to provide QUARKUS APIs with OpenAPI (https://quarkus.io/guides/openapi-swaggerui).

D.1. Execute the following command to add the openapi extension:

```
./mvnw quarkus:add-extension -Dextensions='quarkus-smallrye-openapi'
```

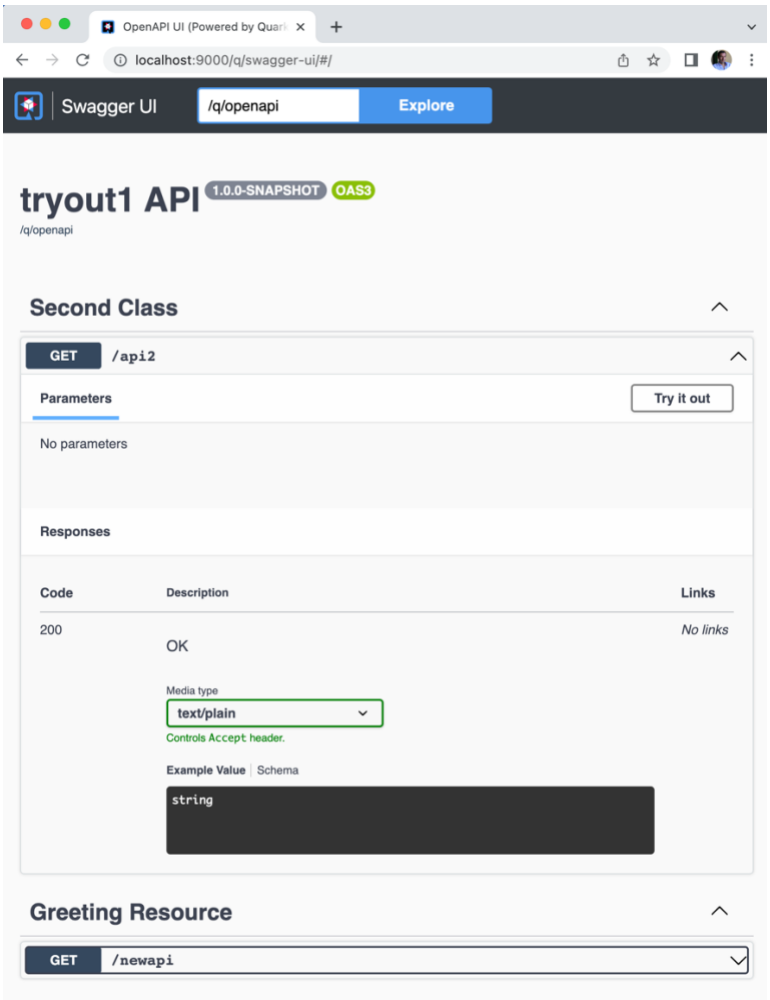
D.2. Change the end point for swagger-ui, adding the following configuration in application.properties:

```
quarkus.swagger-ui.path=swagger-ui
quarkus.swagger-ui.always-include=true
```

D.3. Execute the command on command line or calling maven inside VS Code:

```
./mvnw quarkus:dev
```

D.4. Navigate to http://localhost:9000/q/swagger-ui/ , a similar result will be presented to you:



E. QUARKUS accessing database with all CRUD Operations

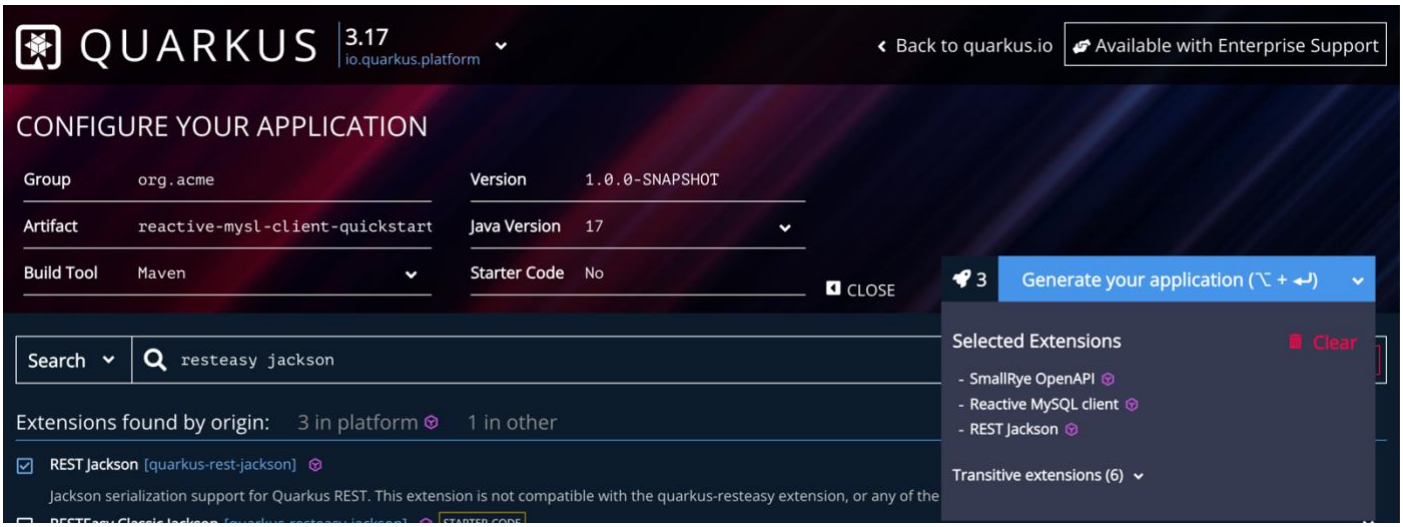
E.1. A ready-to-use MySQL server to try out this example.

```
docker run -it --name mysqlInstance -e MYSQL_ROOT_PASSWORD=password -p 3306:3306 -d mysql:latest
```

E.2. Create a MySQL table to try out the examples with the following command in MYQSL Workbench;

```
CREATE DATABASE quarkus_test;
```

E.3. Navigate to the QUARKUS project creation website at https://code.quarkus.io and use the following configuration:



E.4. Generate, download the zip file and extract it to your local folder project

E.5. Import your newly created project to VS Code

E.6. Add the following configuration in src/main/resources/application.properties of your project

```
quarkus.datasource.db-kind=mysql
quarkus.datasource.username=root
quarkus.datasource.password=<your MySQL root password>
quarkus.datasource.reactive.url=mysql://localhost:3306/quarkus_test

quarkus.swagger-ui.path=swagger-ui
quarkus.swagger-ui.always-include=true
```

E.7. Add the file Fruit.java to src/main/java/Fruit.java

```
import io.smallrye.mutiny.Multi;
import io.smallrye.mutiny.Uni;
import io.vertx.mutiny.mysqlclient.MySQLPool;
import io.vertx.mutiny.sqlclient.Row;
import io.vertx.mutiny.sqlclient.RowSet;
import io.vertx.mutiny.sqlclient.Tuple;

public class Fruit {

    public Long id;
    public String name;

        public Fruit() {
        }

        public Fruit(String name) {
            this.name = name;
        }

        public Fruit(Long id, String name) {
            this.id = id;
            this.name = name;
        }
```

```
        private static Fruit from(Row row) {
            return new Fruit(row.getLong("id"), row.getString("name"));
        }

        public static Multi<Fruit> findAll(MySQLPool client) {
            return client.query("SELECT id, name FROM fruits ORDER BY name ASC").execute()
                    .onItem().transformToMulti(set -> Multi.createFrom().iterable(set))
                    .onItem().transform(Fruit::from);
        }

        public static Uni<Fruit> findById(MySQLPool client, Long id) {
            return client.preparedQuery("SELECT id, name FROM fruits WHERE id = ?").execute(Tuple.of(id))
                    .onItem().transform(RowSet::iterator)
                    .onItem().transform(iterator -> iterator.hasNext() ? from(iterator.next()) : null);
        }

        public Uni<Boolean> save(MySQLPool client) {
            return client.preparedQuery("INSERT INTO fruits(name) VALUES (?)").execute(Tuple.of(name))
                    .onItem().transform(pgRowSet -> pgRowSet.rowCount() == 1 );
        }

        public static Uni<Boolean> delete(MySQLPool client, Long id) {
            return client.preparedQuery("DELETE FROM fruits WHERE id = ?").execute(Tuple.of(id))
                    .onItem().transform(pgRowSet -> pgRowSet.rowCount() == 1);
        }

        public static Uni<Boolean> update(MySQLPool client, Long id, String name) {
            return client.preparedQuery("UPDATE fruits SET name = ? WHERE id =
?").execute(Tuple.of(name,id))
                    .onItem().transform(pgRowSet -> pgRowSet.rowCount() == 1 );
        }
 }
```

E.8. Add the file FruitResource.java to src/main/java/FruitResource.java

```
import java.net.URI;
import jakarta.enterprise.event.Observes;
import jakarta.inject.Inject;
import jakarta.ws.rs.*;
import org.eclipse.microprofile.config.inject.ConfigProperty;

import io.quarkus.runtime.StartupEvent;
import io.smallrye.mutiny.Multi;
import io.smallrye.mutiny.Uni;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.core.Response.ResponseBuilder;
import jakarta.ws.rs.core.MediaType;


@Path("fruits")
public class FruitResource {

    @Inject
    io.vertx.mutiny.mysqlclient.MySQLPool client;

    @Inject
    @ConfigProperty(name = "myapp.schema.create", defaultValue = "true")
    boolean schemaCreate;

    void config(@Observes StartupEvent ev) {
        if (schemaCreate) {
            initdb();
        }
    }

    private void initdb() {
       client.query("DROP TABLE IF EXISTS fruits").execute()
        .flatMap(r -> client.query("CREATE TABLE fruits (id SERIAL PRIMARY KEY, name TEXT NOT
NULL)").execute())
        .flatMap(r -> client.query("INSERT INTO fruits (name) VALUES ('Orange')").execute())
        .flatMap(r -> client.query("INSERT INTO fruits (name) VALUES ('Pear')").execute())
        .flatMap(r -> client.query("INSERT INTO fruits (name) VALUES ('Apple')").execute())
        .flatMap(r -> client.query("INSERT INTO fruits (name) VALUES ('other')").execute())
        .await().indefinitely();
    }

    @GET
    public Multi<Fruit> get() {
```

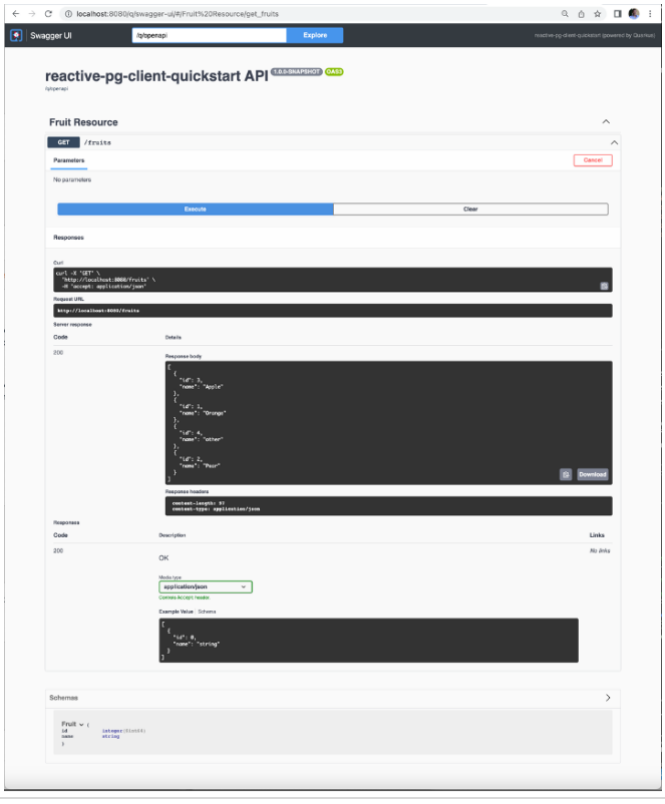
```
        return Fruit.findAll(client);
    }

    @GET
    @Path("{id}")
    public Uni<Response> getSingle(Long id) {
        return Fruit.findById(client, id)
                .onItem().transform(fruit -> fruit != null ? Response.ok(fruit) :
Response.status(Response.Status.NOT_FOUND))
                .onItem().transform(ResponseBuilder::build);
    }

    @POST
    public Uni<Response> create(Fruit fruit) {
        return fruit.save(client)
                .onItem().transform(id -> URI.create("/fruits/" + id))
                .onItem().transform(uri -> Response.created(uri).build());
    }

    @DELETE
    @Path("{id}")
    public Uni<Response> delete(Long id) {
        return Fruit.delete(client, id)
                .onItem().transform(deleted -> deleted ? Response.Status.NO_CONTENT :
Response.Status.NOT_FOUND)
                .onItem().transform(status -> Response.status(status).build());
    }

    @PUT
    @Path("/{id}/{name}")
    public Uni<Response> update(Long id , String name) {
        return Fruit.update(client, id , name)
                .onItem().transform(updated -> updated ? Response.Status.NO_CONTENT :
Response.Status.NOT_FOUND)
                .onItem().transform(status -> Response.status(status).build());
    }
}
```

E.9. Compile and execute the project:

```
cd reactive-mysql-CRUD-client-quickstart

./mvnw quarkus:dev
```

E.10. Navigate to http://localhost:8080/q/swagger-ui/ and execute the all the CRUD APIs.

Try to create, update, and delete an item.



F. Deploying a QUARKS project with docker locally

F.1. Execute the following command in your project directory:

```
./mvnw quarkus:add-extension -Dextensions="quarkus-container-image-docker"
```

F.2. Add the following configuration to application.properties in order to include all the dependencies to execute in a different environment from dev:

```
quarkus.container-image.build=true
```

F.3. Build docker image

*Hint 1: if you are not logged in on docker use the "**docker login**" command*
Hint 2: have in mind that all the network requirements for the docker image need to be met beforehand. To facilitate start, e.g., with a simple tryout project as in section B of this document.

```
./mvnw clean package
```

F.4. Check that image has been created successfully.

```
% docker image ls -a
REPOSITORY                 TAG              IMAGE ID       CREATED         SIZE
sergioguerreiro/tryout1    1.0.0-SNAPSHOT   eaf911ae3716   15 hours ago    492MB
```

F.5. Create container. Have in mind that the port available and the name of the project depends on your previous settings.

```
docker run -d --name tryout1 -p 9000:9000 sergioguerreiro/tryout1:1.0.0-SNAPSHOT
```

Test the container opening the following URL in your browser:

```
# open browser with url: http://localhost:9000/q/swagger-ui/
```

Hint 1: Later if you would like to list all the available container use the following command:

```
> docker container ls --all

CONTAINER ID   IMAGE                                     COMMAND                  CREATED         STATUS
PORTS                                       NAMES
2d4cc67021e3   sergioguerreiro/tryout1:1.0.0-SNAPSHOT    "/usr/local/s2i/run"     2 minutes ago   Up 2 minutes
8080/tcp, 8443/tcp, 8778/tcp, 0.0.0.0:9000->9000/tcp    tryout1
```

Hint 2: Later if you would like to stop the container use the following command:

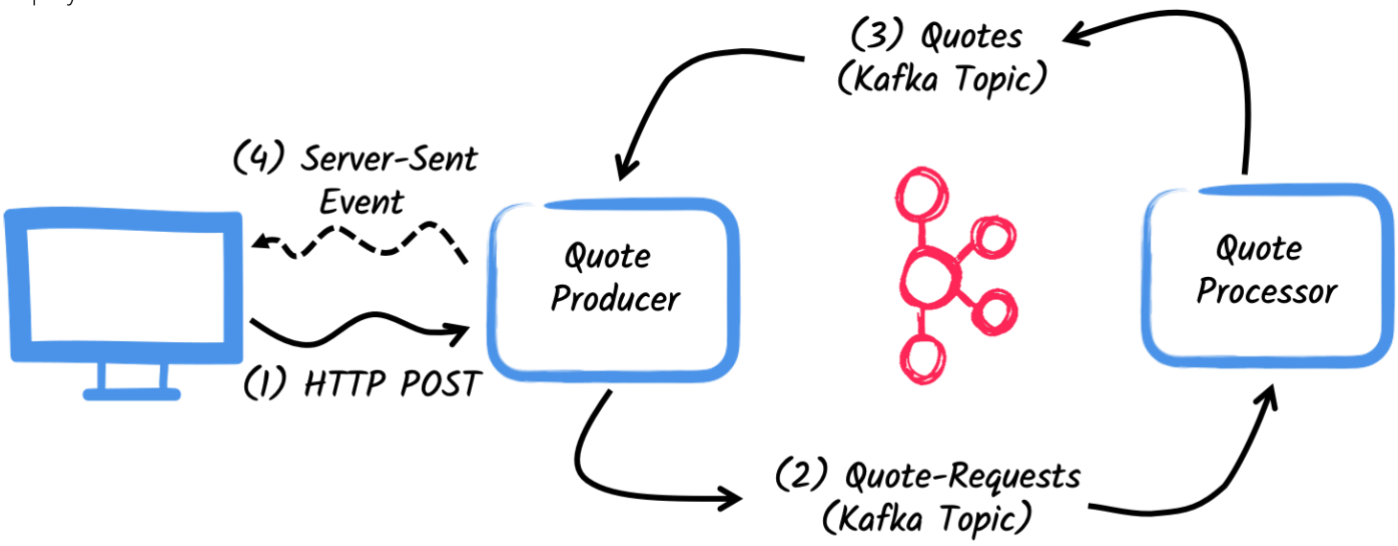
```
docker container stop <containerID_hash>
```

Hint 3: Later if you would like to restart the container use the following command:

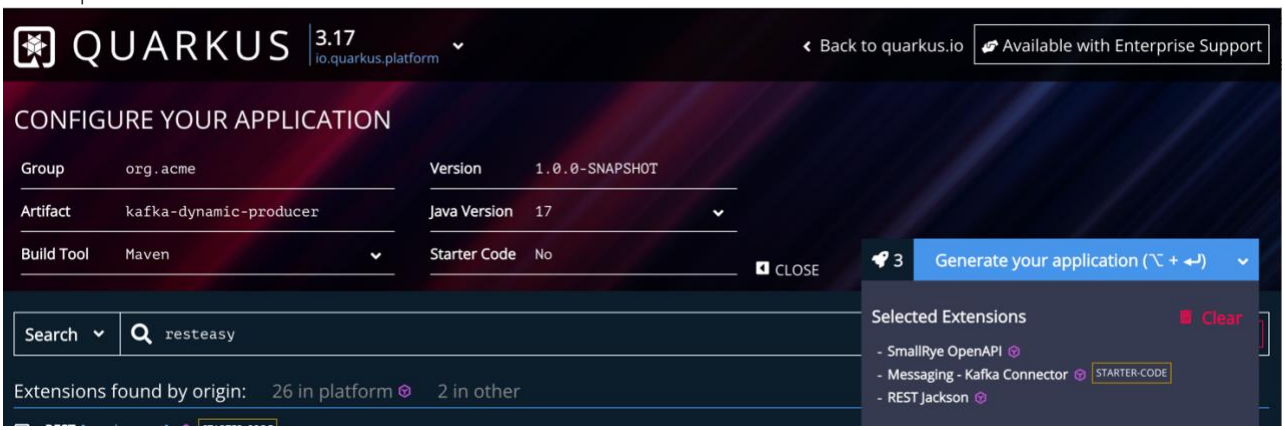
```
docker container start <containerID_hash>
```

G. QUARKUS with Kafka messaging embedded (in development environment)

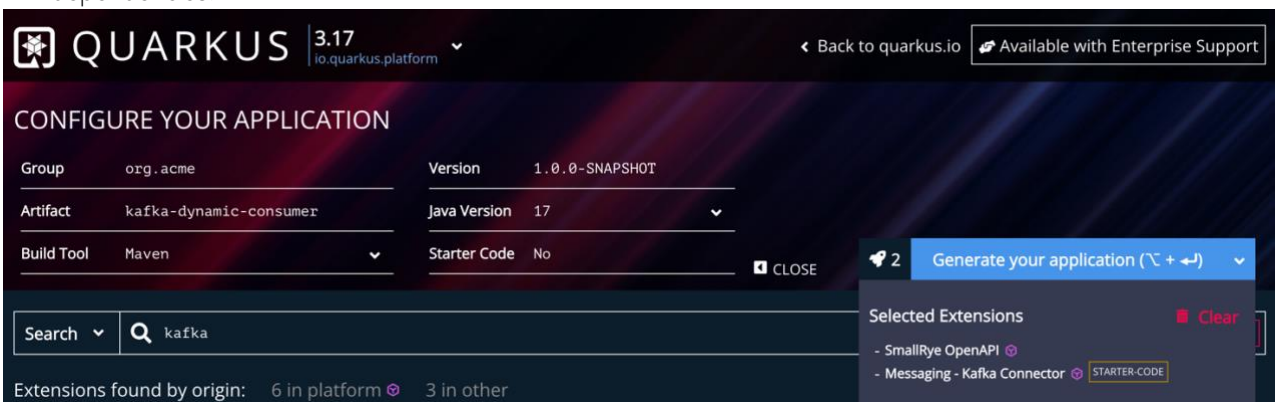
The following example aims at implementing the following messaging architecture. Kafka is embedded in QUARKUS deployment.



G.1. Create an empty project (KAFKA message producer) at https://code.quarkus.io/ with the following dependencies.



G.2.Create an empty project (KAFKA message consumer) at https://code.quarkus.io/ with the following dependencies.



G.3.In project kafka-dynamic-producer create the following JAVA classes at src/main/java/org/acme/kafka/model/Quote.java

```
package org.acme.kafka.model;

public class Quote {

    public String id;
    public int price;
```

```
    /**
    * Default constructor required for Jackson serializer
    */
    public Quote() { }

    public Quote(String id, int price) {
        this.id = id;
        this.price = price;
    }

    @Override
    public String toString() {
        return "Quote{" +
                "id='" + id + '\'' +
                ", price=" + price +
                '}';
    }
}
```

Add to kafka-dynamic-producer the **src/main/java/org/acme/kafka/producer/QuotesResource.java**

```
package org.acme.kafka.producer;

import java.util.UUID;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.POST;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

import org.acme.kafka.model.Quote;
import org.eclipse.microprofile.reactive.messaging.Channel;
import org.eclipse.microprofile.reactive.messaging.Emitter;

import io.smallrye.mutiny.Multi;

@Path("/quotes")
public class QuotesResource {

    @Channel("quote-requests")
    Emitter<String> quoteRequestEmitter;

    @Channel("quotes")
    Multi<Quote> quotes;
    /**
     * Endpoint to generate a new quote request id and send it to "quote-requests" Kafka topic using the
emitter.
     */
    @POST
    @Path("/request")
    @Produces(MediaType.TEXT_PLAIN)
    public String createRequest() {
        UUID uuid = UUID.randomUUID();
        quoteRequestEmitter.send(uuid.toString());
        return uuid.toString();
    }

    /**
     * Endpoint retrieving the "quotes" Kafka topic and sending the items to a server sent event.
     */
    @GET
    @Produces(MediaType.SERVER_SENT_EVENTS)
    public Multi<Quote> stream() {
        return quotes;
    }

}
```

Add to kafka-dynamic-producer the following **index.html** file to **/src/main/resources/META-INF/resources/**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>kafka-quickstart - 1.0.0-SNAPSHOT</title>
    <style>
        h1, h2, h3, h4, h5, h6 {
            margin-bottom: 0.5rem;
            font-weight: 400;
```

```
            line-height: 1.5;
        }

        h1 {
            font-size: 2.5rem;
        }

        h2 {
            font-size: 2rem
        }

        h3 {
            font-size: 1.75rem
        }

        h4 {
            font-size: 1.5rem
        }

        h5 {
            font-size: 1.25rem
        }

        h6 {
            font-size: 1rem
        }

        .lead {
            font-weight: 300;
            font-size: 2rem;
        }

        .banner {
            font-size: 2.7rem;
            margin: 0;
            padding: 2rem 1rem;
            background-color: #00A1E2;
            color: white;
        }

        body {
            margin: 0;
            font-family: -apple-system, system-ui, "Segoe UI", Roboto, "Helvetica Neue", Arial, sans-serif,
"Apple Color Emoji", "Segoe UI Emoji", "Segoe UI Symbol", "Noto Color Emoji";
        }

        code {
            font-family: SFMono-Regular, Menlo, Monaco, Consolas, "Liberation Mono", "Courier New",
monospace;
            font-size: 87.5%;
            color: #e83e8c;
            word-break: break-word;
        }

        .left-column {
            padding: .75rem;
            max-width: 75%;
            min-width: 55%;
        }

        .right-column {
            padding: .75rem;
            max-width: 25%;
        }

        .container {
            display: flex;
            width: 100%;
        }

        li {
            margin: 0.75rem;
        }

        .right-section {
            margin-left: 1rem;
            padding-left: 0.5rem;
        }

        .right-section h3 {
            padding-top: 0;
            font-weight: 200;
```

```
        }

        .right-section ul {
            border-left: 0.3rem solid #00A1E2;
            list-style-type: none;
            padding-left: 0;
        }

    </style>
</head>
<body>

<div class="banner lead">
    Your new Cloud-Native application is ready!
</div>

<div class="container">
    <div class="left-column">
        <p class="lead"> Congratulations, you have created a new Quarkus application.</p>

        <h2>Why do you see this?</h2>

        <p>This page is served by Quarkus. The source is in
            <code>src/main/resources/META-INF/resources/index.html</code>.</p>

        <h2>What can I do from here?</h2>

        <p>If not already done, run the application in <em>dev mode</em> using: <code>mvn quarkus:dev</code>.
        </p>
        <ul>
          <li>Add REST resources, Servlets, functions and other services in <code>src/main/java</code>.</li>
          <li>Your static assets are located in <code>src/main/resources/META-INF/resources</code>.</li>
          <li>Configure your application in <code>src/main/resources/application.properties</code>.
          </li>
        </ul>

        <h2>How do I get rid of this page?</h2>
        <p>Just delete the <code>src/main/resources/META-INF/resources/index.html</code> file.</p>
    </div>
    <div class="right-column">
        <div class="right-section">
            <h3>Application</h3>
            <ul>
                <li>GroupId: org.acme.quarkus.sample</li>
                <li>ArtifactId: kafka-quickstart</li>
                <li>Version: 1.0.0-SNAPSHOT</li>
                <li>Quarkus Version: 999-SNAPSHOT</li>
            </ul>
        </div>
        <div class="right-section">
            <h3>Next steps</h3>
            <ul>
                <li><a href="https://quarkus.io/guides/maven-tooling.html">Setup your IDE</a></li>
                <li><a href="https://quarkus.io/guides/getting-started.html">Getting started</a></li>
                <li><a href="https://quarkus.io">Quarkus Web Site</a></li>
            </ul>
        </div>
    </div>
</div>


</body>
</html>
```

Add to kafka-dynamic-producer the following quotes.html file to **/src/main/resources/META-INF/resources/**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Prices</title>

    <link rel="stylesheet" type="text/css"
          href="https://cdnjs.cloudflare.com/ajax/libs/patternfly/3.24.0/css/patternfly.min.css">
    <link rel="stylesheet" type="text/css"
          href="https://cdnjs.cloudflare.com/ajax/libs/patternfly/3.24.0/css/patternfly-additions.min.css">
</head>
<body>
<div class="container">
    <div class="card">
```

```
        <div class="card-body">
            <h2 class="card-title">Quotes</h2>
            <button class="btn btn-info" id="request-quote">Request Quote</button>
            <div class="quotes"></div>
        </div>
    </div>
</div>
</body>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
    $("#request-quote").click((event) => {
        fetch("/quotes/request", {method: "POST"})
        .then(res => res.text())
        .then(qid => {
            var row = $(`<h4 class='col-md-12' id='${qid}'>Quote # <i>${qid}</i> |
<strong>Pending</strong></h4>`);
            $(".quotes").prepend(row);
        });
    });

    var source = new EventSource("/quotes");
    source.onmessage = (event) => {
      var json = JSON.parse(event.data);
      $(`#${json.id}`).html((index, html) => {
        return html.replace("Pending", `\$\xA0${json.price}`);
      });
    };
</script>
</html>
```

G.4. Add the following configuration in src/main/resources/application.properties of your consumer project:

```
%dev.quarkus.http.port=8081

# Go bad to the first records, if it's out first access
mp.messaging.incoming.requests.auto.offset.reset=earliest

# Set the Kafka topic, as it's not the channel name
mp.messaging.incoming.requests.topic=quote-requests
```

Hint: For now, the producer project do not need any configuration in the application.properties.

G.5. In project kafka-dynamic-consumer create the following JAVA classes at
src/main/java/org/acme/kafka/model/Quote.java

```
package org.acme.kafka.model;

public class Quote {

    public String id;
    public int price;

    /**
    * Default constructor required for Jackson serializer
    */
    public Quote() { }

    public Quote(String id, int price) {
        this.id = id;
        this.price = price;
    }

    @Override
    public String toString() {
        return "Quote{" +
                "id='" + id + '\'' +
                ", price=" + price +
                '}';
    }
}
```

Add to kafka-dynamic-consumer the src/main/java/org/acme/kafka/processor/QuotesProcessor.java

```
package org.acme.kafka.processor;
```

```
import java.util.Random;

import jakarta.enterprise.context.ApplicationScoped;

import org.acme.kafka.model.Quote;
import org.eclipse.microprofile.reactive.messaging.Incoming;
import org.eclipse.microprofile.reactive.messaging.Outgoing;

import io.smallrye.reactive.messaging.annotations.Blocking;

/**
 * A bean consuming data from the "quote-requests" Kafka topic (mapped to "requests" channel) and giving out
a random quote.
 * The result is pushed to the "quotes" Kafka topic.
 */
@ApplicationScoped
public class QuotesProcessor {

    private Random random = new Random();

    @Incoming("requests")
    @Outgoing("quotes")
    @Blocking
    public Quote process(String quoteRequest) throws InterruptedException {
        // simulate some hard working task
        Thread.sleep(200);
        return new Quote(quoteRequest, random.nextInt(100));
    }
}
```

G.6. Execute in a separate command line:

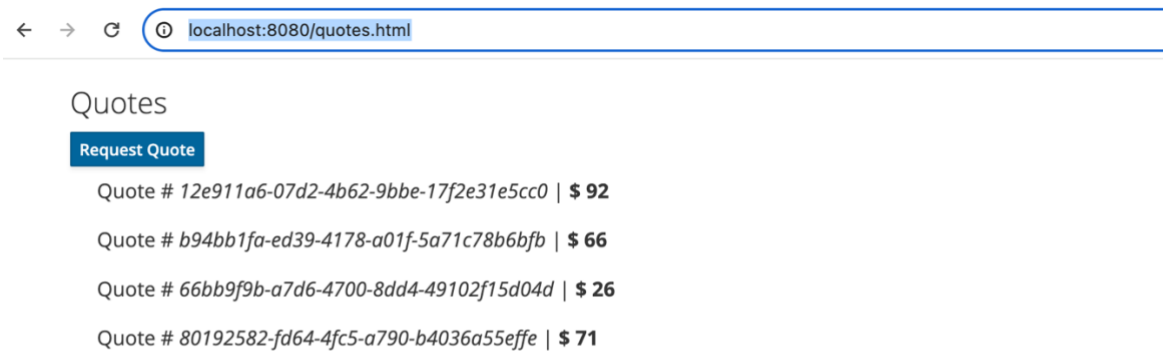
```
cd kafka-quickstart-consumer

./mvnw quarkus:dev
```

G.7. Execute in a separate command line:

```
cd kafka-quickstart-producer

./mvnw quarkus:dev
```

G.8. Navigate to http://localhost:8080/quotes.html

Then, click “Request Quote” and check in the consumer window that the message has been printed.



H. QUARKUS with Kafka messaging not embedded

Using the previous project of "**G. QUARKUS with Kafka messaging embedded (in development environment)**" execute the following configuration changes:

H.1. Add the following configuration in src/main/resources/application.properties of your consumer project. You are adding a remote integration with a previously installed Kafka server (or broker) in AWS EC2 at port 9092.

```
kafka.bootstrap.servers=<YOUR EC2 NAME>:9092

mp.messaging.incoming.requests.topic=quote-requests
mp.messaging.incoming.requests.auto.offset.reset=earliest
```

H.2. Then, add the following configuration in src/main/resources/application.properties of your producer project. You are adding a remote integration with a previously installed Kafka server (or broker) in AWS EC2 at port 9092.

```
kafka.bootstrap.servers=<YOUR EC2 NAME>:9092
```

H.3. Execute in a separate command line

```
cd kafka-quickstart-consumer

./mvnw quarkus:dev
```

H.4.Execute in a separate command line

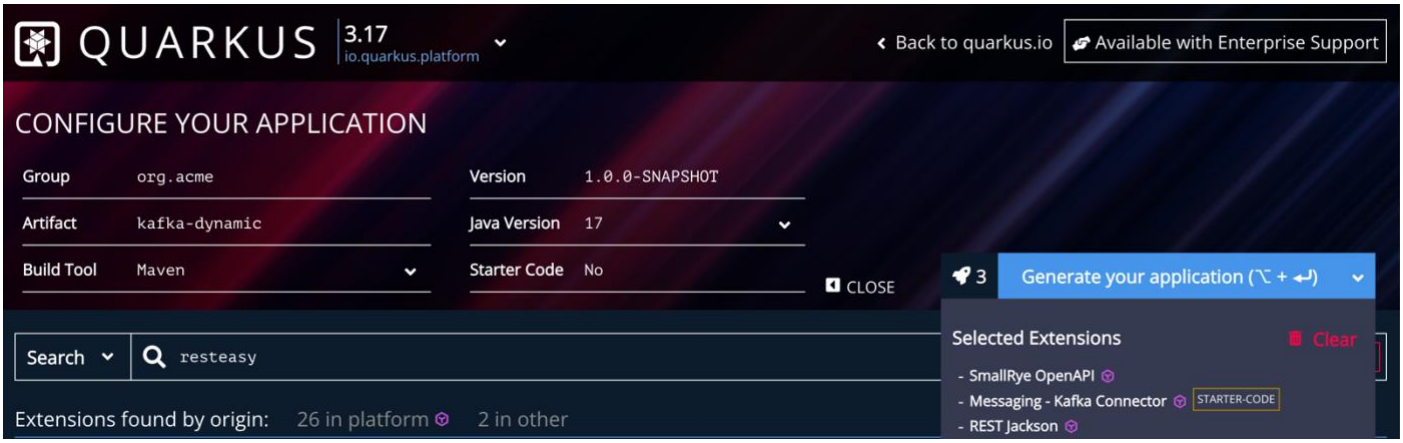
```
cd kafka-quickstart-producer

./mvnw quarkus:dev
```

H.5.Navigate to http://localhost:8080/quotes.html

I. Consuming a Kafka Topic that changes dynamically

In Quarkus Kafka, the @Incoming annotation defines a method that acts as a Kafka consumer. You are not allowed to change it dynamically at runtime, as the topic is determined at compile time. A simple workaround solution is to use the old JAVA way of dealing with consuming Kafka messages. For exemplification consider a REST API exposing a POST where the topic is received, then a new JAVA Thread is started containing the Kafka consuming source code for that topic.

I.1. Create an empty project (KAFKA message consumer) at https://code.quarkus.io/ with the following dependencies.



And then, add the following files to your project.

I.2. The KafkaProvisioningResource.java to src/main/java/org/acme/KafkaProvisioningResource.java

```
package org.acme;

import jakarta.ws.rs.POST;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;
import org.acme.model.Topic;
import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/createTopic")
public class KafkaProvisioningResource {

    @ConfigProperty(name = "kafka.bootstrap.servers")
    String kafka_servers;

    @POST
    @Produces(MediaType.TEXT_PLAIN)
    public String ProvisioningConsumer(Topic topic) {
        Thread worker = new DynamicTopicConsumer(topic.TopicName , kafka servers );
        worker.start();
        return "New worker started";
    }
}
```

I.3. The DynamicTopicConsumer.java to src/main/java/org/acme/DynamicTopicConsumer.java

```
package org.acme;

import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class DynamicTopicConsumer extends Thread  {
    private String kafka_servers;
    private String topic;

    public DynamicTopicConsumer(String topic_received , String kafka_servers_received)
```

```
    {
        topic = topic_received;
        kafka_servers = kafka_servers_received;
    }

    public void run()
        {
            try
                {
             Properties properties = new Properties();
             properties.put("bootstrap.servers", kafka_servers);
             properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
             properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
             properties.put("group.id", "your-group-id");

             try (Consumer<String, String> consumer = new KafkaConsumer<>(properties)) {
                 consumer.subscribe(Collections.singletonList(topic));

                 while (true) {
                     ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
                     for (ConsumerRecord<String, String> record : records)
                     {
                       System.out.printf("topic = %s, partition = %s, offset = %d,key = %s, value = %s\n",
                       record.topic(), record.partition(), record.offset(),
                       record.key(), record.value());
                     }
                 }
             }
         }
         catch (Exception e)
                {  System.out.println("Exception is caught");  }
    }
}
```

I.4. The Topic.java to src/main/java/org/acme/Topic.java

```
package org.acme.model;

public class Topic {
    public String TopicName;
    public Topic() {  }
    public Topic(String topicName) {  TopicName = topicName;  }
    @Override
    public String toString()
    {  return "Topic [TopicName=" + TopicName + "]";  }
}
```

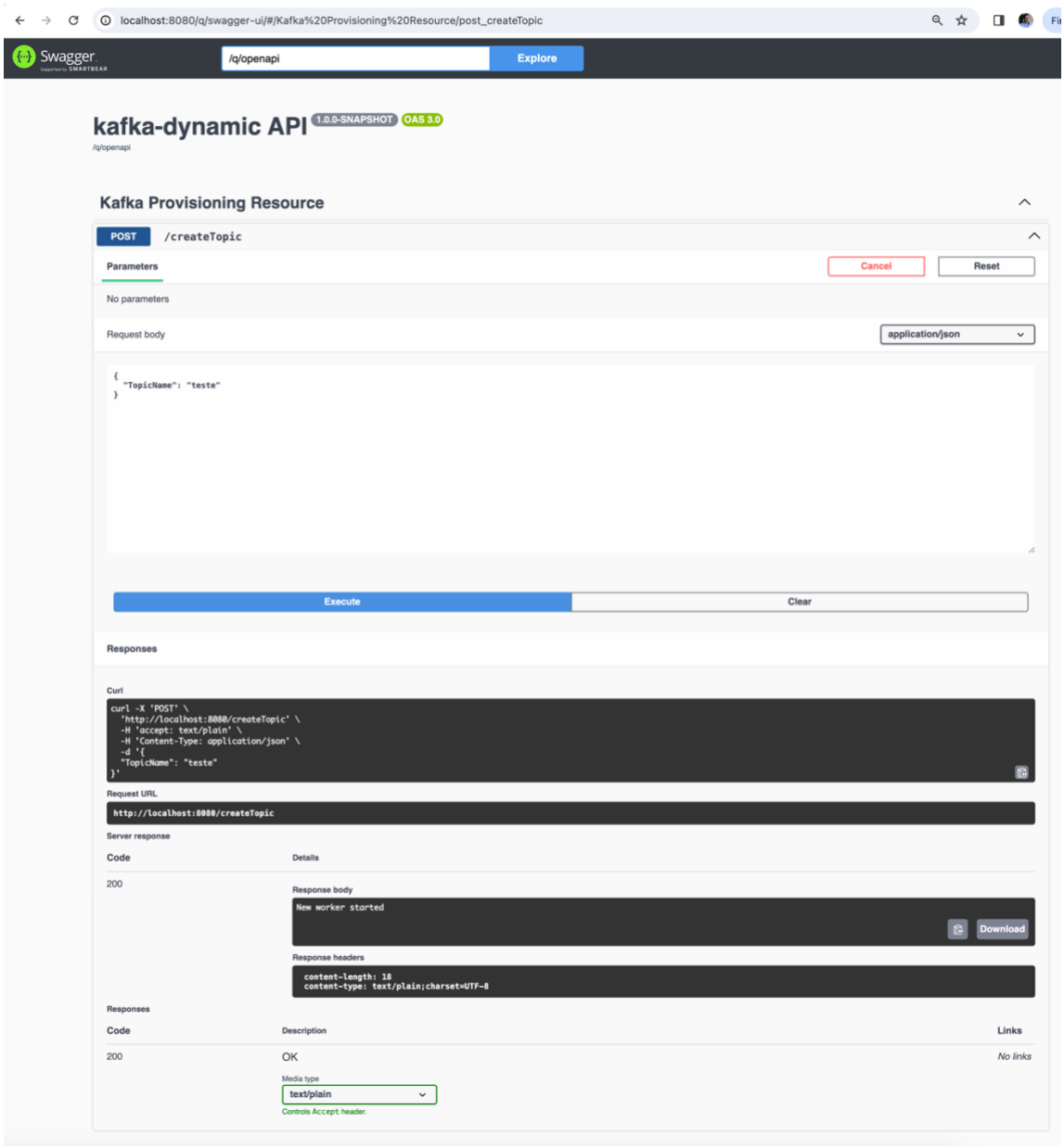
I.5. Add the following configuration in src/main/resources/application.properties

```
quarkus.swagger-ui.path=swagger-ui
quarkus.swagger-ui.always-include=true

kafka.bootstrap.servers=<YOUR KAFKA NAME>:9092
```

I.6. Execute in a separate command line

```
./mvnw quarkus:dev
```

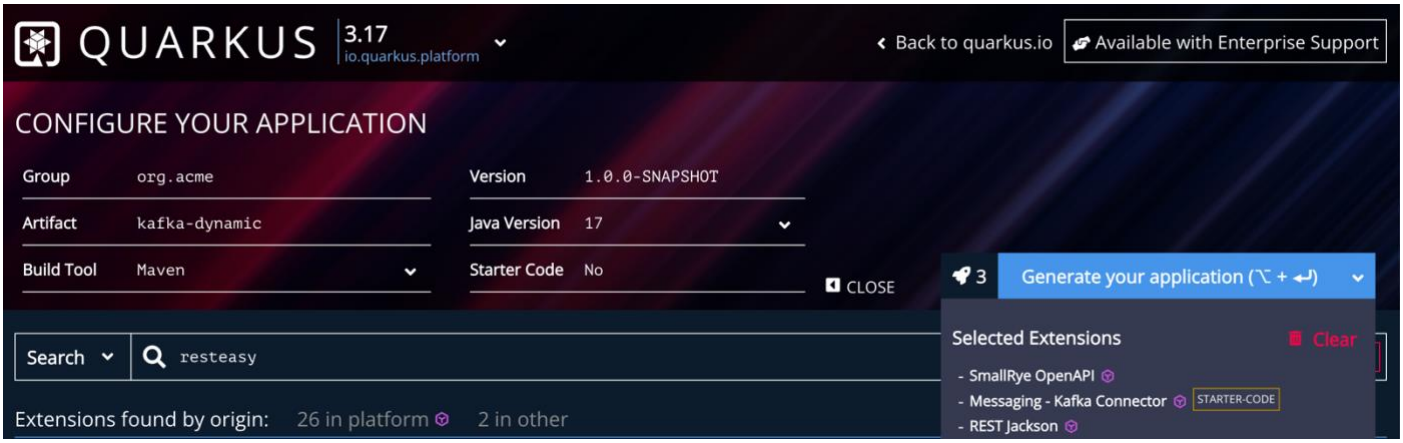
I.7. Navigate to http://localhost:8080/q/swagger-ui/ and execute the POST API. Then, check that the new kafka topic is being consumed, sending some new messages to that topic. At this moment, you can change dynamically the topic that is being consumed.


localhost:8080/q/swagger-ui/#/Kafka%20Provisioning%20Resource/post_createTopic
Swagger
/q/openapi
Explore
kafka-dynamic API
1.0.0-SNAPSHOT
OAS 3.0
/q/openapi
Kafka Provisioning Resource
POST
/createTopic
Parameters
Cancel
Reset
No parameters
Request body
application/json
{
"TopicName": "teste"
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:8080/createTopic' \
-H 'accept: text/plain' \
-H 'Content-Type: application/json' \
-d '{
"TopicName": "teste"
}'
Request URL
http://localhost:8080/createTopic
Server response
Code
Details
200
Response body
New worker started
Download
Response headers
content-length: 18
content-type: text/plain;charset=UTF-8
Responses
Code
Description
Links
200
OK
No links
Media type
text/plain
Controls Accept header.

J. Creating and Removing Kafka Topics dynamically

Before starting the kafka message consumption it is required to create the specific kafka topic. Using a microservice approach, consider a REST API exposing a POST where the topic, to be created or deleted, is received, then the library kafka-client provides APIs to perform the desired operation on a kafka cluster.

J.1. Create an empty project (KAFKA message consumer) at https://code.quarkus.io/ with the following dependencies.



And then, add the following files to your project.

J.2. The KafkaProvisioningResource.java to src/main/java/org/acme/KafkaProvisioningResource.java

```
package org.acme;

import jakarta.ws.rs.POST;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;
import java.util.Arrays;
import java.util.Properties;
import java.util.concurrent.ExecutionException;
import org.acme.model.Topic;
import org.eclipse.microprofile.config.inject.ConfigProperty;
import org.apache.kafka.clients.admin.AdminClient;
import org.apache.kafka.clients.admin.CreateTopicsResult;
import org.apache.kafka.clients.admin.NewTopic;
import org.apache.kafka.clients.admin.DeleteTopicsResult;
import org.apache.kafka.common.errors.TopicExistsException;
import org.apache.kafka.common.errors.UnknownTopicOrPartitionException;

@Path("/TopicManagement")
public class KafkaProvisioningResource {

    @ConfigProperty(name = "kafka.bootstrap.servers")
    String kafka_servers;

    @POST
    @Path("/createTopic")
    @Produces(MediaType.TEXT_PLAIN)
    public String CreateNewTopic(Topic topic) {
            Properties properties = new Properties();
            properties.put("bootstrap.servers", kafka_servers);
            properties.put("connections.max.idle.ms", 10000);
            properties.put("request.timeout.ms", 5000);
            try (AdminClient client = AdminClient.create(properties))
        {
             CreateTopicsResult result = client.createTopics(Arrays.asList(
                        new NewTopic(topic.TopicName, 1, (short) 1)  ));
             try {    result.all().get();    }
             catch ( org.apache.kafka.common.errors.TopicExistsException exc )
             {   throw exc; }
             catch (  ExecutionException | InterruptedException e )
             {   throw new IllegalStateException(e);  }
            }
        return ("New Topic created = " + topic);
    }
```

```
    @POST
    @Path("/DeleteTopic")
    @Produces(MediaType.TEXT_PLAIN)
    public String RemoveOneTopic(Topic topic) {
            Properties properties = new Properties();
            properties.put("bootstrap.servers", kafka_servers);
            properties.put("connections.max.idle.ms", 10000);
            properties.put("request.timeout.ms", 5000);
            try (AdminClient client = AdminClient.create(properties))
        {
             DeleteTopicsResult result = client.deleteTopics(Arrays.asList( topic.TopicName ));
             try {    result.all().get();    }
             catch ( org.apache.kafka.common.errors.UnknownTopicOrPartitionException exc)
             { throw exc; }
             catch (  ExecutionException | InterruptedException e )
             {   throw new IllegalStateException(e);  }
            }
        return ("New Topic deleted = " + topic);
    }
}
```

J.3. The Topic.java to src/main/java/org/acme/Topic.java

```
package org.acme.model;

public class Topic {
    public String TopicName;
    public Topic() {  }
    public Topic(String topicName) {  TopicName = topicName;  }
    @Override
    public String toString()
    {  return "Topic [TopicName=" + TopicName + "]";  }
}
```

J.4. Add the following configuration in src/main/resources/application.properties

```
quarkus.swagger-ui.path=swagger-ui
quarkus.swagger-ui.always-include=true

kafka.bootstrap.servers=<YOUR KAFKA NAME>:9092
```

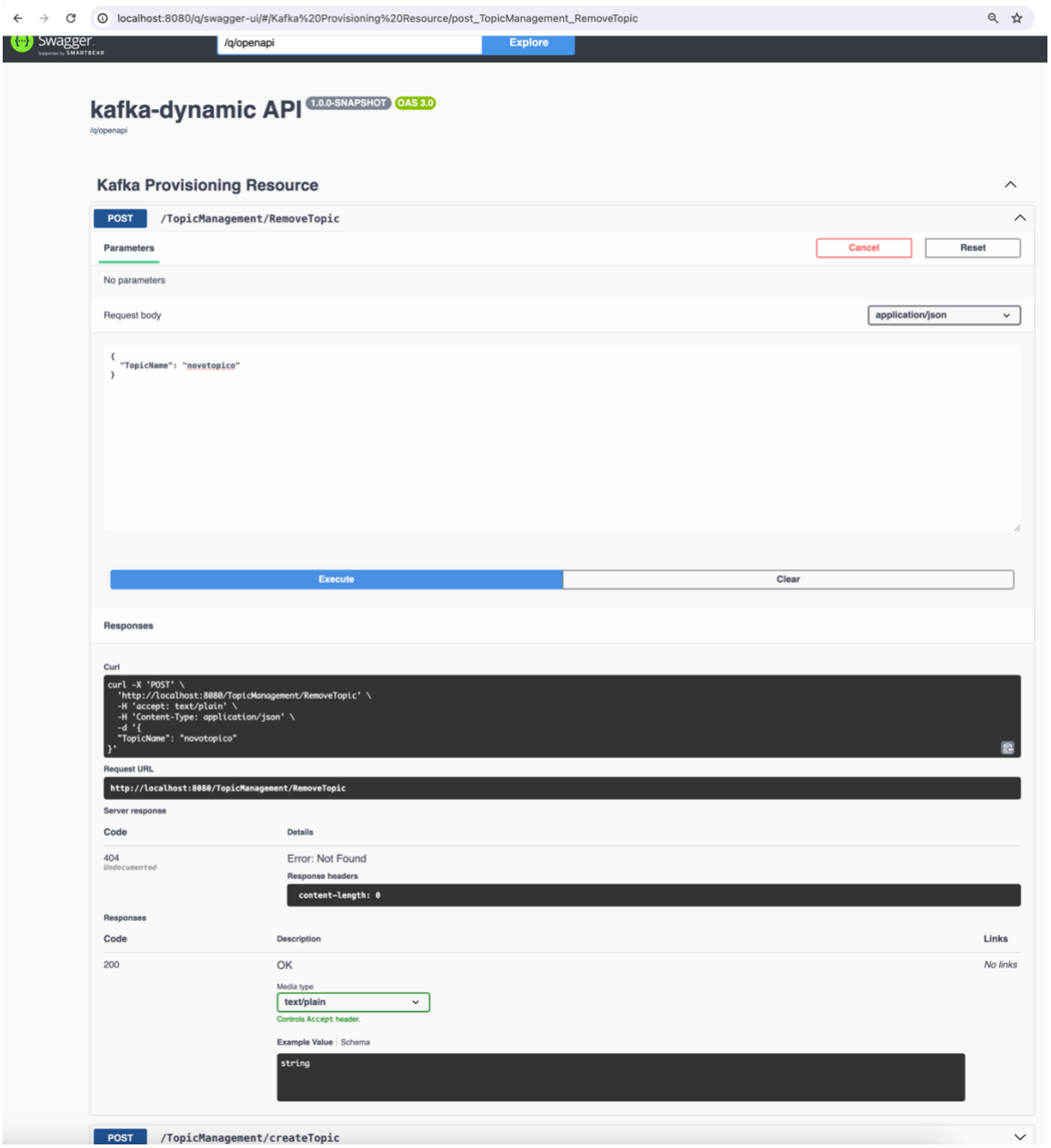
J.5. Add the following dependency in pom.xml

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

J.6. Execute in a separate command line

```
./mvnw quarkus:dev
```

Navigate to http://localhost:8080/q/swagger-ui/ and execute the POST APIs. Then, check that the new kafka topic is created or deleted, listing the topics in that kafka cluster.


localhost:8080/q/swagger-ui/#/Kafka%20Provisioning%20Resource/post_TopicManagement_RemoveTopic
Swagger
/q/openapi
Explore
kafka-dynamic API
1.0.0-SNAPSHOT
OAS 3.0
/q/openapi
Kafka Provisioning Resource
POST
/TopicManagement/RemoveTopic
Parameters
Cancel
Reset
No parameters
Request body
application/json
{
"TopicName": "novotopico"
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:8080/TopicManagement/RemoveTopic' \
-H 'accept: text/plain' \
-H 'Content-Type: application/json' \
-d '{
"TopicName": "novotopico"
}'
Request URL
http://localhost:8080/TopicManagement/RemoveTopic
Server response
Code
Details
404
Undocumented
Error: Not Found
Response headers
content-length: 0
Responses
Code
Description
Links
200
OK
No links
Media type
text/plain
Controls Accept header.
Example Value
Schema
string
POST
/TopicManagement/createTopic

K. Deploying QUARKUS in AWS EC2 environment with github and terraform!

K.1. Create your new Quarkus project as before.

K.2. Execute the following command in your project directory to add the docker dependencies:

```
./mvnw quarkus:add-extension -Dextensions="quarkus-container-image-docker"
```

K.3. Add the following configuration to **application.properties** in order to include all the dependencies to execute in a different environment from dev:

```
quarkus.swagger-ui.path=swagger-ui
quarkus.swagger-ui.always-include=true

quarkus.container-image.build=true
quarkus.container-image.push=true
```

Also, add your Git Hub accountID in the **application.properties** of your project, to identify it correctly and push it directly:

```
quarkus.container-image.group=YOUR DOCKER USERNAME
```

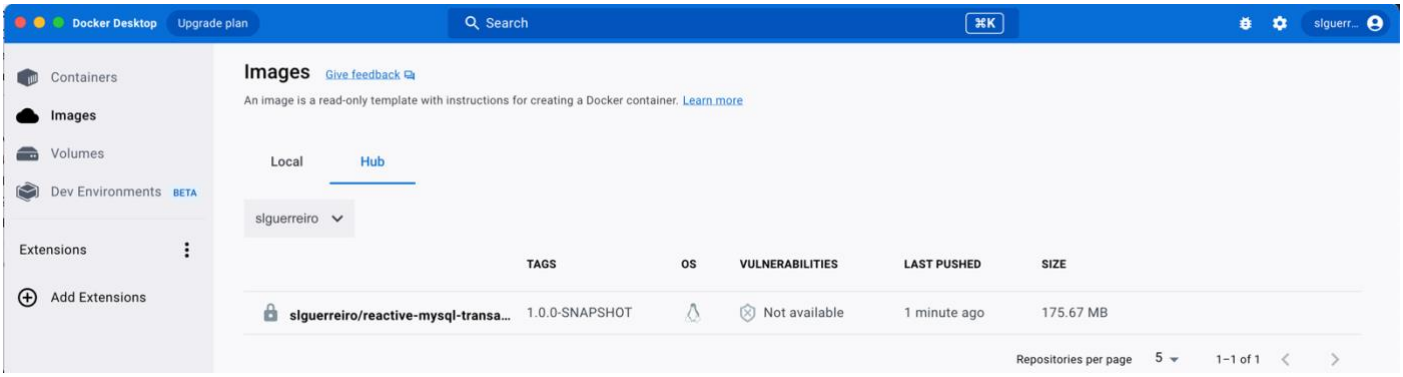
K.4. Login in your account (you can create one in http://hub.docker.com if needed)

```
docker login -u "your docker username" -p "your password"
```

K.5. Build docker image and push it directly to you github account:

```
./mvnw clean package
```

K.6. Check that image has been created successfully on github or on the docker dashboard (optional):



K.7. Now, to deploy your Quarkus project in EC2, using terraform, and based on the image previously pushed to git, create a .tf file with the following instructions. Note that the AMI to create should be compatible with the operating systems where you previously compiled the docker image.

```
terraform {
  required version = ">= 1.0.0, < 2.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region       = "us-east-1"
  access_key   = "YOUR ACESS KEY"
  secret_key   = "YOUR SECRET KEY"
  token        = "YOUR TOKEN"
}
```

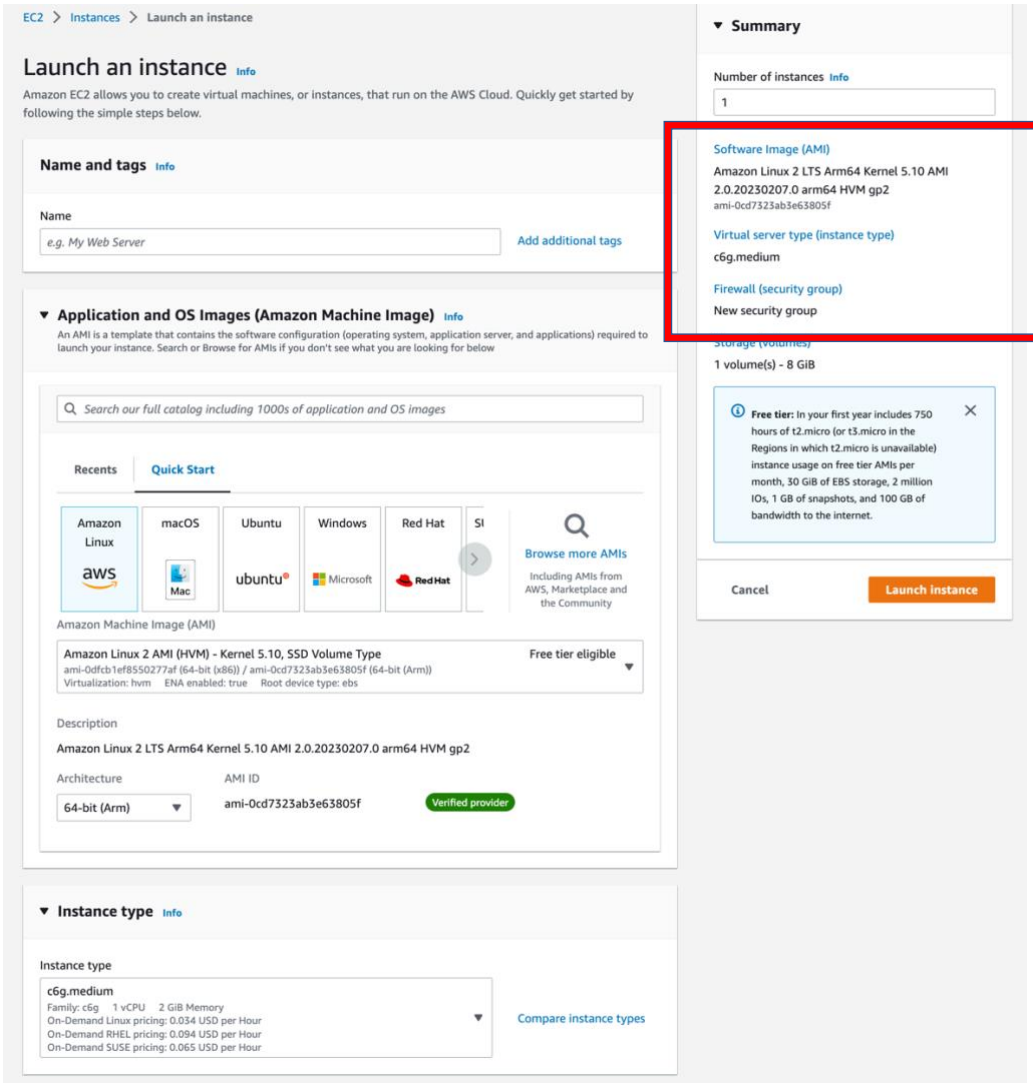
```
resource "aws_instance" "exampleDeployQuarkus" {
  ami                     = "ami-08cf815cff6ee258a" # Amazon Linux ARM AMI built by Amazon Web Services - FOR
DOCKER image COMPATIBILITY if compiled previously on ARM
  instance type           = "t4g.nano"
  vpc_security_group_ids  = [aws_security_group.instance.id]
  key_name                = "vockey"

  user_data = "${file("quarkus.sh")}"

  user_data_replace_on_change = true

  tags = {
    Name = "terraform-deploy-QuarkusProject"
  }
}

resource "aws_security_group" "instance" {
  name = var.security_group_name
  ingress {
    from port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6 cidr blocks = ["::/0"]
  }
}

variable "security group name" {
  description = "The name of the security group"
  type        = string
  default     = "terraform-Quarkus-instance"
}

output "address" {
  value       = aws_instance.exampleDeployQuarkus.public_dns
  description = "Address of the Quarkus EC2 machine"
}
```

And the correspondingly script "`quarkus.sh`" with the following:

```
#!/bin/bash
echo "Starting..."

sudo yum install -y docker

sudo service docker start

sudo docker login -u "YOUR DOCKER USERNAME" -p "YOUR DOCKER PASSWORD"

sudo docker pull YOUR DOCKER USERNAME/tryout1:1.0.0-SNAPSHOT

sudo docker run -d --name tryout2 -p 9000:9000 YOUR DOCKER USERNAME/YOUR ARTIFACT ID:1.0.0-SNAPSHOT

echo "Finished."
```

Docker images built for ARM architecture may not work on AMD architecture machines. This is because ARM and AMD processors use different instruction sets and have different hardware architectures.

For this example, the AMI identifier and instance type were extracted from the launching instance AWS GUI.


EC2 > Instances > Launch an instance
Launch an instance Info
Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.
Name and tags Info
Name
e.g. My Web Server
Add additional tags
Application and OS Images (Amazon Machine Image) Info
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below
Search our full catalog including 1000s of application and OS images
Recents
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type
Free tier eligible
ami-0dfcb1ef8550277af (64-bit (x86)) / ami-0cd7323ab3e63805f (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs
Description
Amazon Linux 2 LTS Arm64 Kernel 5.10 AMI 2.0.20230207.0 arm64 HVM gp2
Architecture
AMI ID
64-bit (Arm)
ami-0cd7323ab3e63805f
Verified provider
Instance type Info
Instance type
c6g.medium
Family: c6g 1 vCPU 2 GiB Memory
On-Demand Linux pricing: 0.034 USD per Hour
On-Demand RHEL pricing: 0.094 USD per Hour
On-Demand SUSE pricing: 0.065 USD per Hour
Compare instance types
Summary
Number of instances Info
1
Software Image (AMI)
Amazon Linux 2 LTS Arm64 Kernel 5.10 AMI 2.0.20230207.0 arm64 HVM gp2
ami-0cd7323ab3e63805f
Virtual server type (instance type)
c6g.medium
Firewall (security group)
New security group
1 volume(s) - 8 GiB
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.
Cancel
Launch instance

References

Full official documentation:

- https://quarkus.io/guides/
- https://quarkus.io/guides/container-image#building
- https://dev.mysql.com/downloads/
- https://quarkus.io/guides/reactive-sql-clients
- https://quarkus.io/guides/kafka-reactive-getting-started
- https://quarkus.io/guides/amazon-lambda
- https://aws.amazon.com/pt/blogs/aws-brasil/integracao-do-aws-lambda-com-o-quarkus/
- https://dev.to/marcuspaulo/tutorial-publish-a-quarkus-application-in-kubernetes-minikube-and-dockerhub-36nd
- https://stackoverflow.com/questions/23935141/how-to-copy-docker-images-from-one-host-to-another-without-using-a-repository
- https://quarkus.io/guides/container-image#quarkus-container-image-docker_quarkus.docker.executable-name

DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Tutorial BPMN 2.0
modelling, developing, deploying, installing, testing and operating using Camunda

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to present the details related with: Installation and execution of the Camunda platform, Designing the BPMN 2.0 models using the main constructors of the language, Deploy those previous modelled BPMN 2.0 models, Testing the models, and Operating the platform.

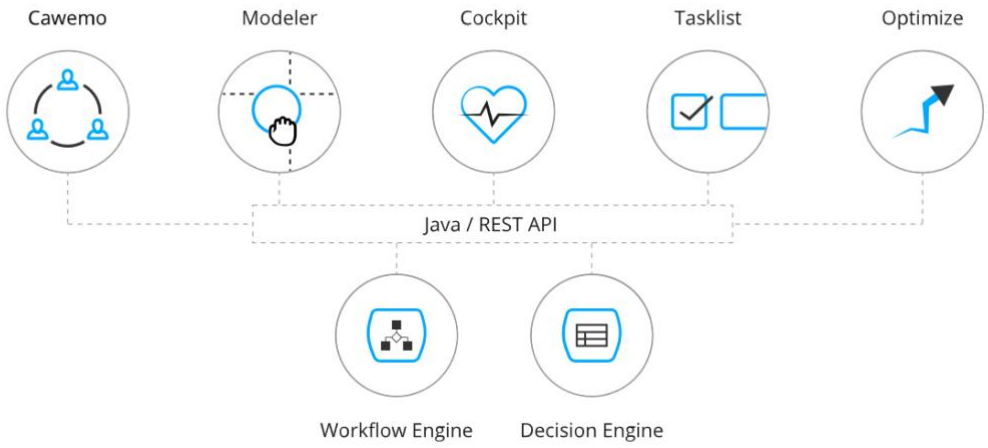
The examples address the following aspects: how to integrate a BPMN process model with an external worker written in JAVA, how to add a user task to the BPMN process model, how to add gateways to the BPMN process model, how to implement DMN rules in the BPMN process model, and how to invoke a cloud REST API.

The following contents is presented in this document.

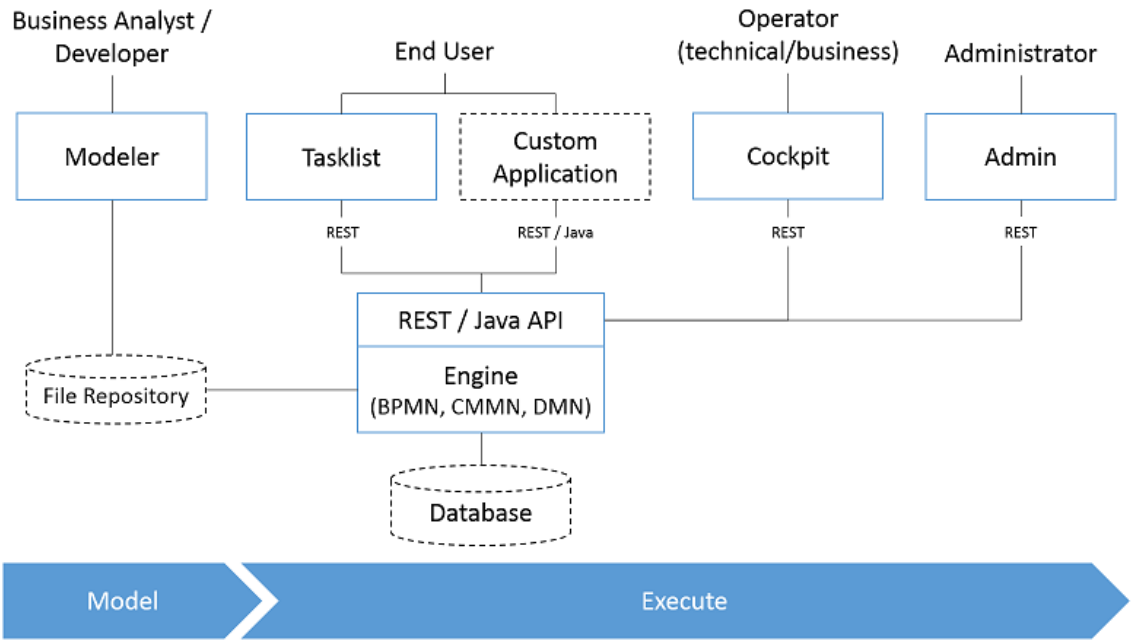
Contents

A. Camunda brief overview

Camunda encompasses the following software modules.



The Community edition that we are going to use have a basic cockpit and optimize component is also not available. The main stakeholders involved in Camunda are the following:



B. Installation and operation of Camunda Engine in AWS EC2

C.1. Create an EC2 new instance of the type: Amazon Linux 2 AMI (HVM), SSD Volume Type. The exact versions may change with time. And define the inbound rules to allow the 8080 and 22 accessed from anywhere.

C.2. Install the most recent version of camunda-modeler (**version 7 recommended**) using the download public available at: https://camunda.com/download/modeler/

C.3. Access the new EC2 instance and execute the following commands:

```
sudo yum update -y
sudo yum install -y docker
sudo service docker start
sudo usermod -aG docker ec2-user
docker ps
exit
```

C.4. Then, access again your EC2 instance and install and execute the Camunda engine with the following command:

```
docker pull camunda/camunda-bpm-platform:latest
docker run -d --name camunda -p 8080:8080 camunda/camunda-bpm-platform:latest
```

Test the installed Camunda engine opening the following URL in your browser (the default user/password is **demo**/demo):

```
# open browser with url: http://YOUR_AWS_EC2_INSTANCE:8080/camunda-welcome/index.html
```

Hint 1: Later if you would like to list all the available container use the following command:

```
PS C:\Users\Sérgio Guerreiro> docker container ls --all

CONTAINER ID        IMAGE                                  COMMAND                  CREATED             STATUS
PORTS               NAMES
44509250aa0c        camunda/camunda-bpm-platform:latest    "/sbin/tini -- ./cam…"   11 days ago         Exited
(143) 1 second ago                        camunda
```

Hint 2: Later if you would like to stop the container use the following command:

```
docker container stop <containerID_hash>
```

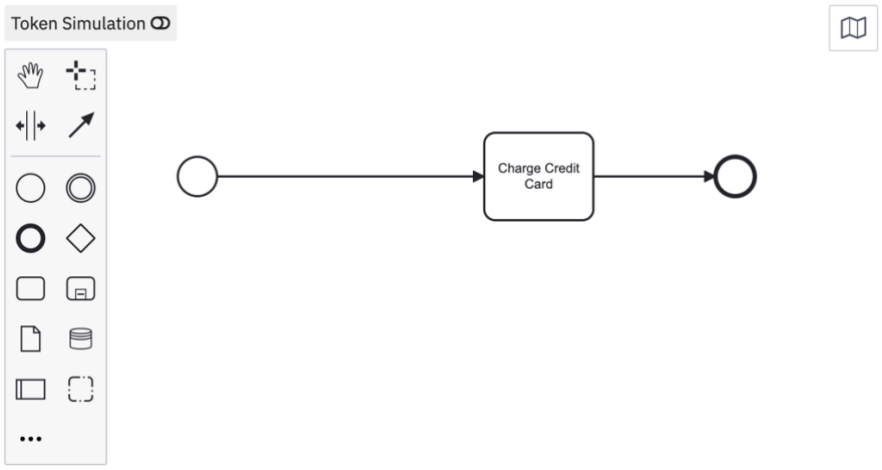
Hint 3: Later if you would like to restart the container use the following command:

```
docker container start <containerID_hash>
```

C. Add a User Task to the Process

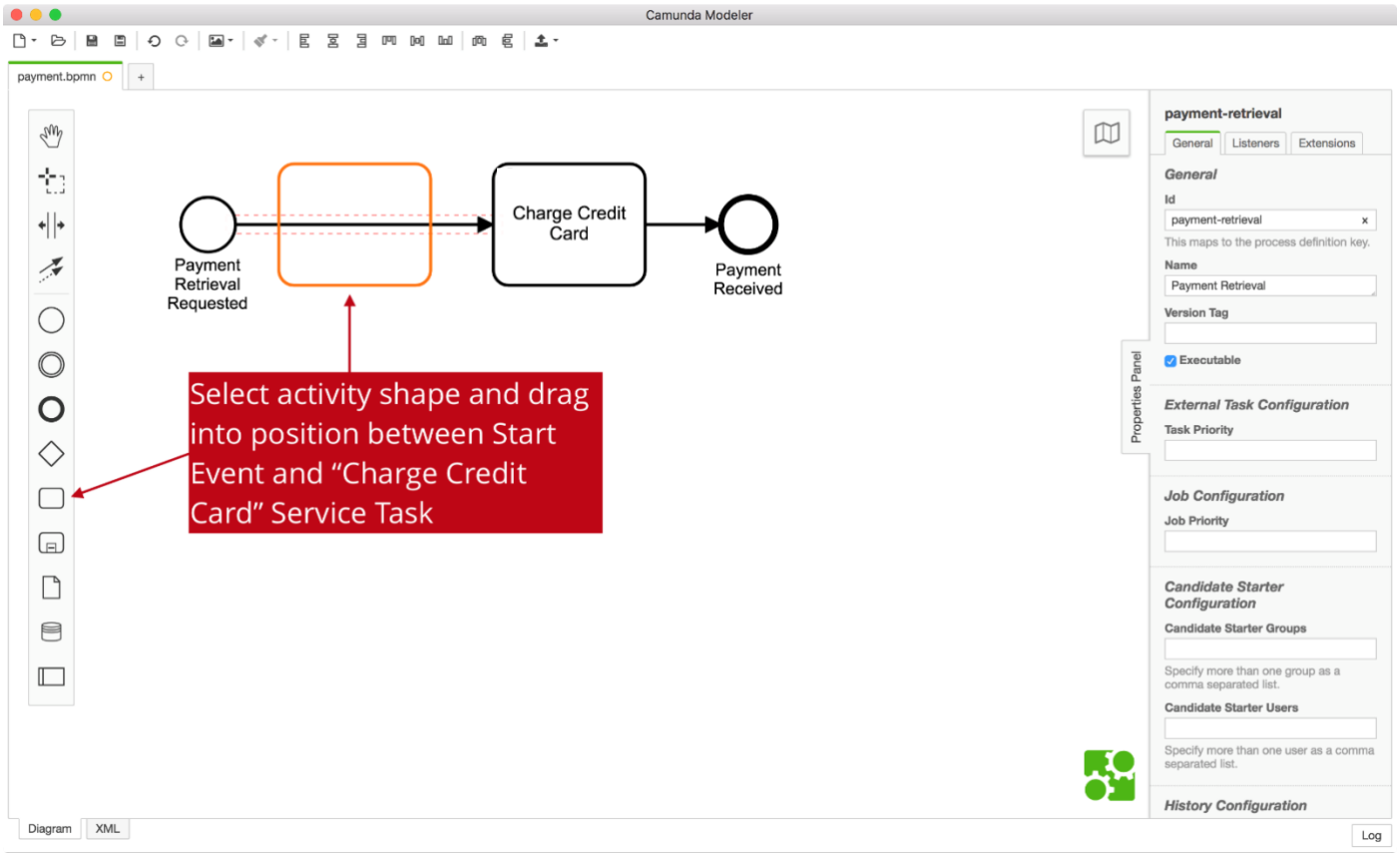
This example is based in the Camunda documentation available at https://docs.camunda.org/get-started/quick-start/user-task/ . The goal of the example is to show how to add a human task in the middle of a process execution. After deploying, go to the tasklist (all tasks) identify the "Approve payment", select the claim, and check approved and then complete. Then, check that instances and tasklist are not available anymore.

D.1. Add an abstract task with the name "Charge Credit Card" between a start and an end event.

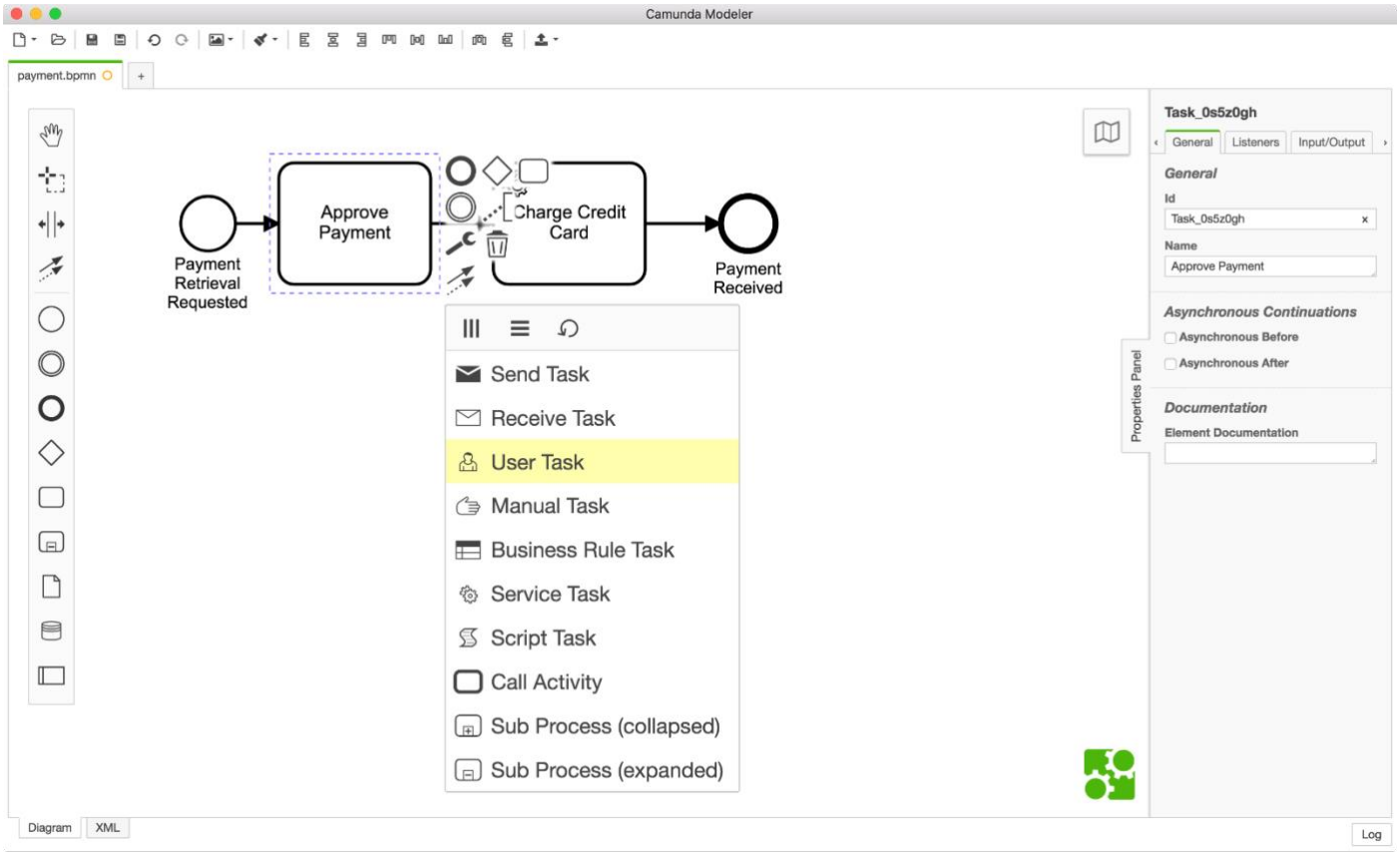


D.2. Add a User Task

Now, we want to modify our process so that we can involve humans. To do so, open the process in the Camunda Modeler. Next, from the Modeler's left-hand menu, select the activity shape (rectangle) and drag it into position between the Start Event and the "Charge Credit Card" Service Task. Name it *Approve Payment*.



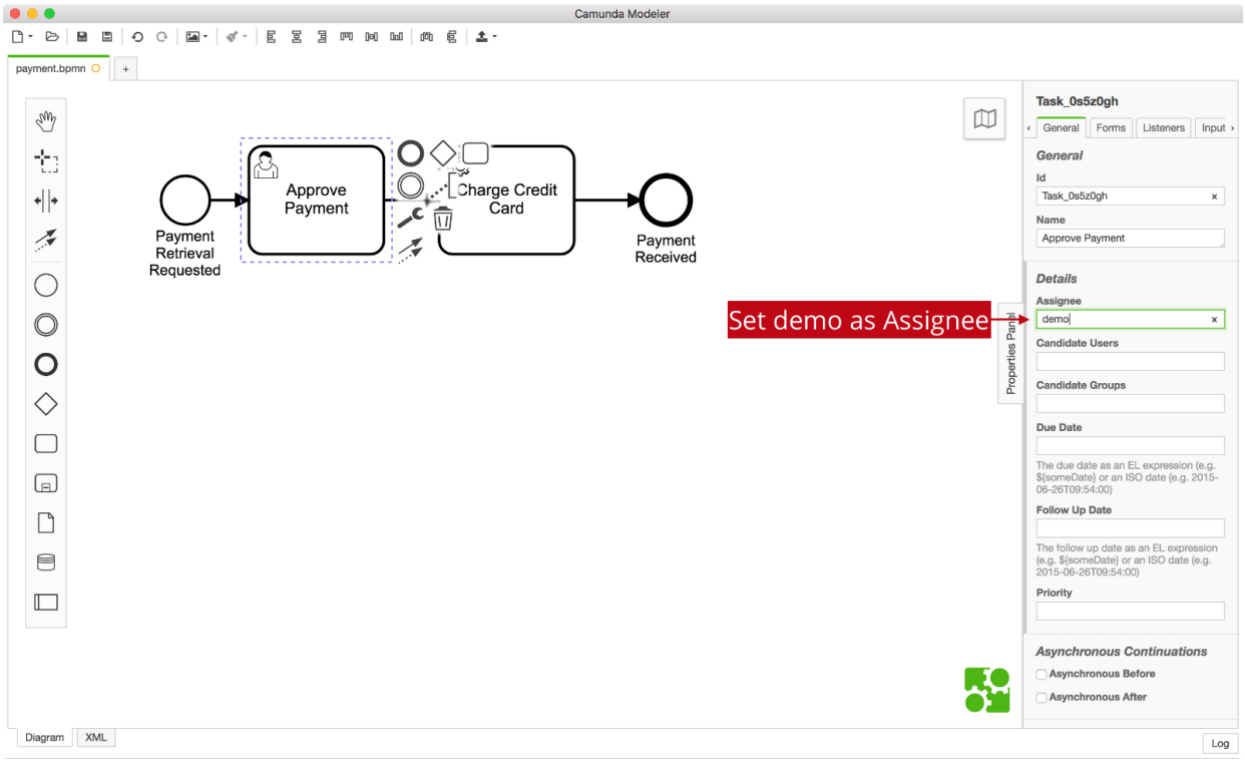
Change the activity type to *User Task* by clicking on it and using the wrench button menu.



D.3. Configure a User Task

Next, open the properties view. If the properties view is not already visible, click on the "Properties Panel" label on the right-hand side of the Modeler canvas.

Select the User Task on the canvas. This will update the selection in the properties view. Scroll to the property named `Assignee`. Type *demo*.



D.4. Configure a basic form in the User Task

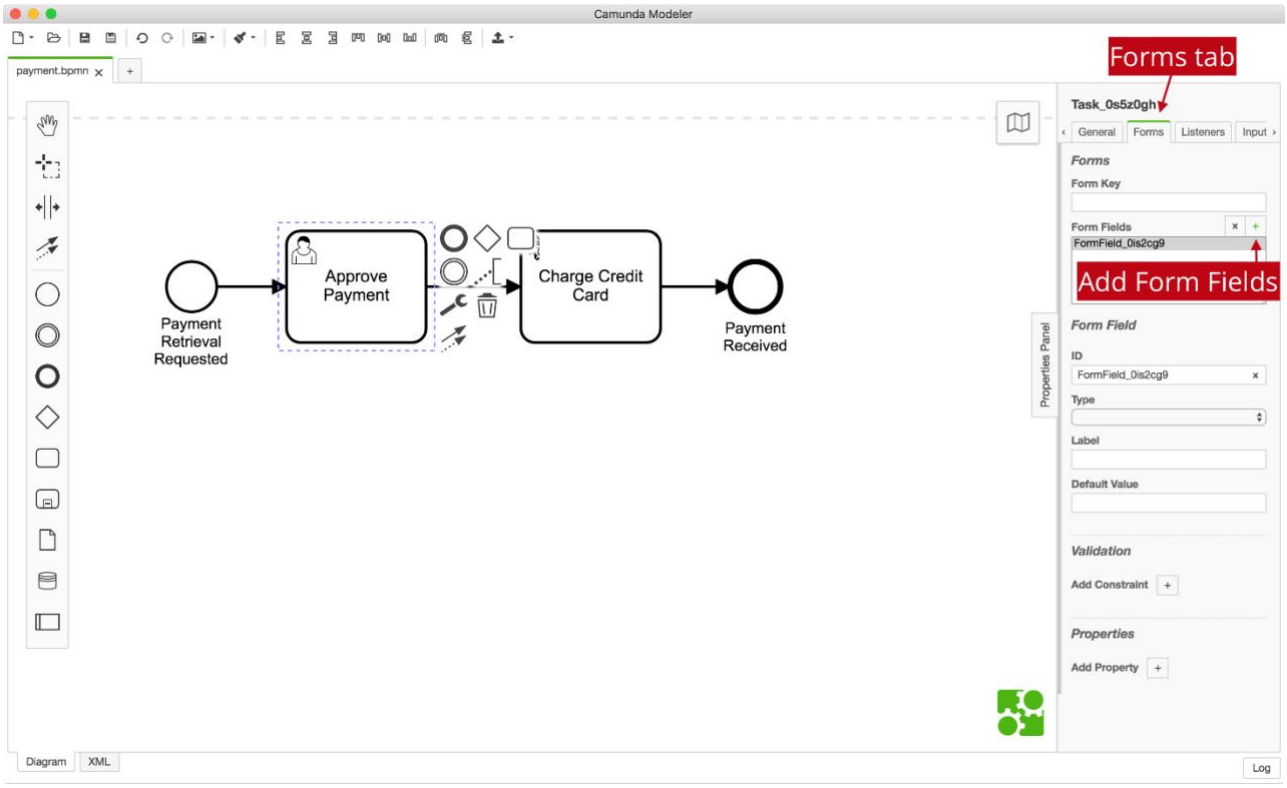
This step will also take place in the properties panel. If the panel is not already visible, click on the “Properties Panel” label on the right-hand side of the Modeler canvas.

Select the User Task on the canvas. This will update the selection in the properties view.

Click on the Tab `Forms` in the properties panel.

(version 5.8.0) -> In the type of Forms choose “Generated Task Forms”

Add three form fields by clicking on the `plus` button:

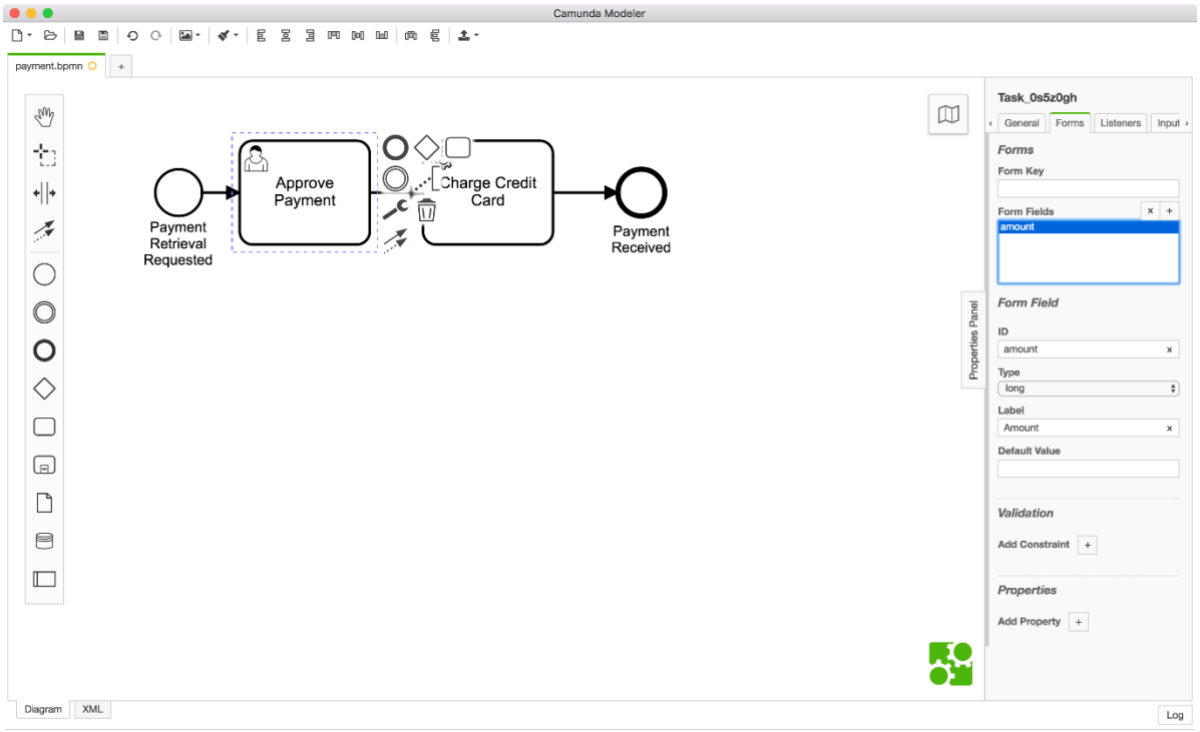


Field 1:

ID: amount

Type: long

Label: Amount

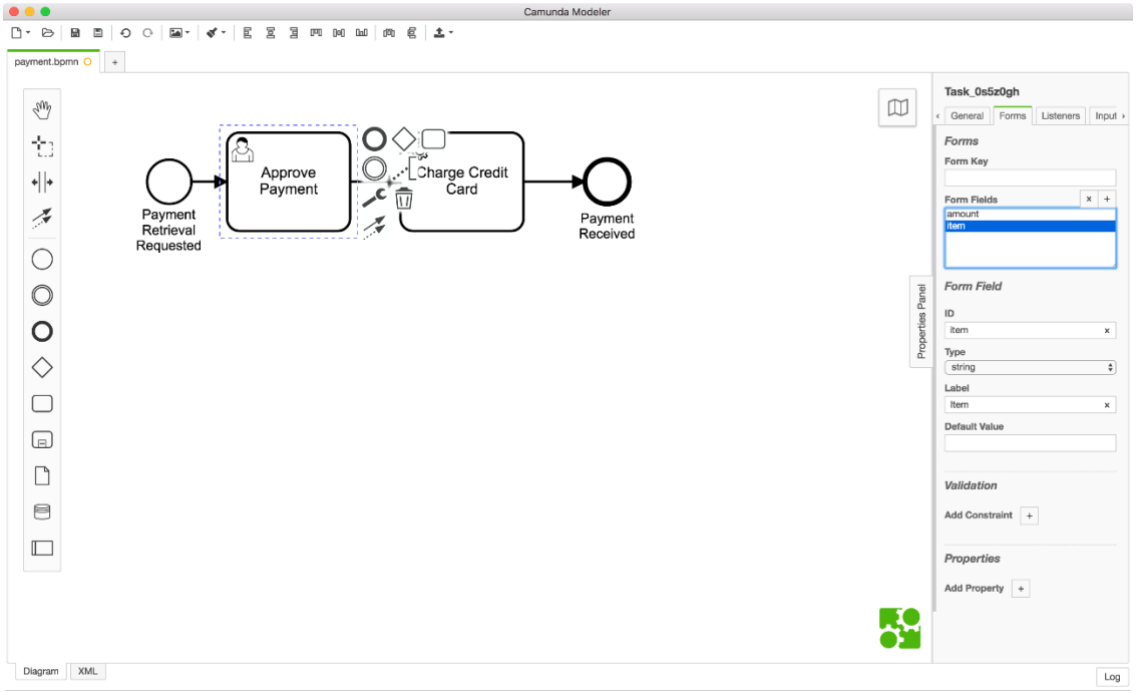


Field 2:

ID: item

Type: string

Label: Item

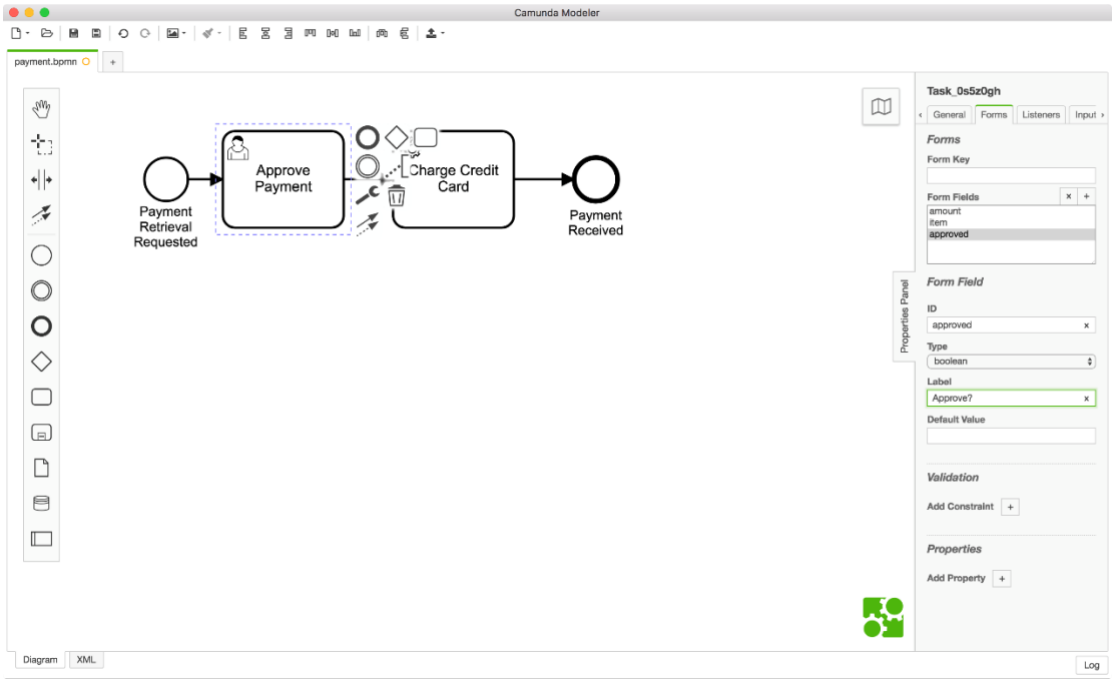


Field 3:

ID: approved

Type: boolean

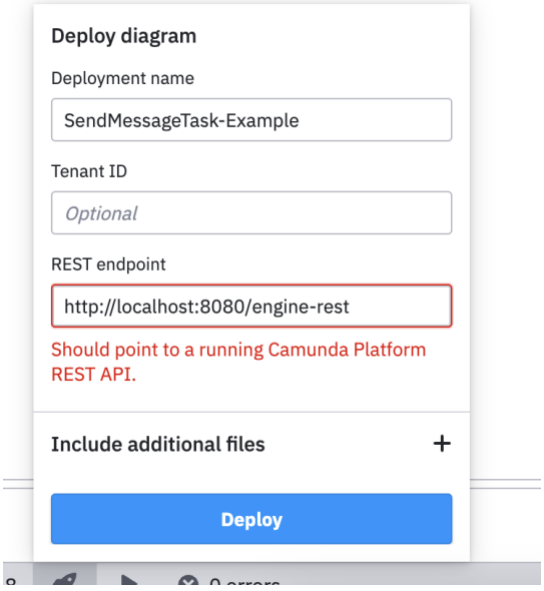
Label: Approved?



D.5. Deploy the Process

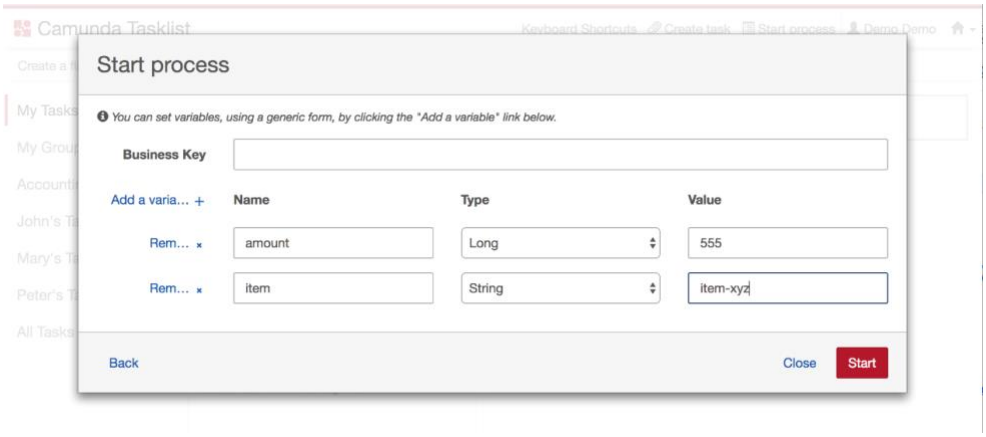
Use the `Deploy` Button in the Camunda Modeler to deploy the updated process to Camunda.

Replacing the localhost for the AWS EC2 DNS_name where you deployed the camunda-engine.



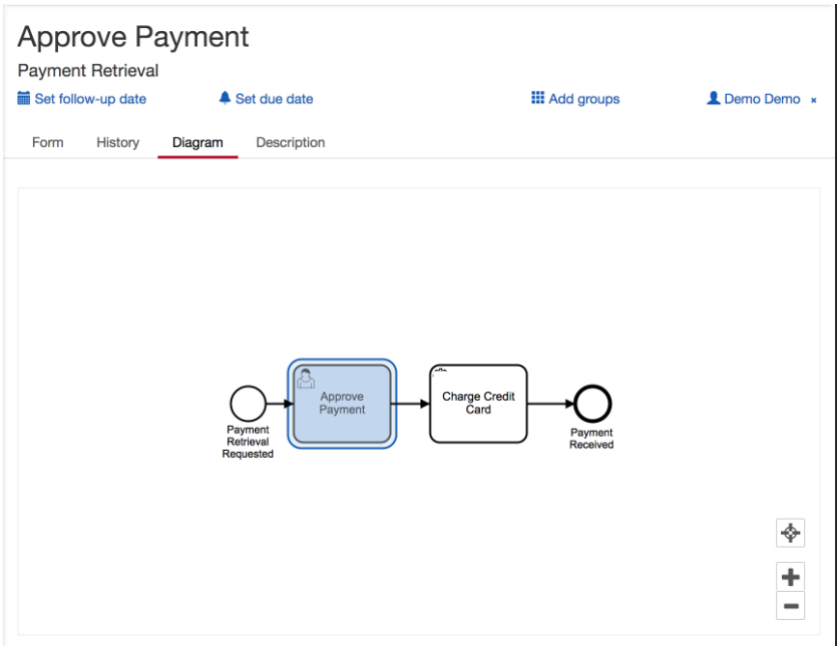
D.6. Work on the Task

Go to Camunda Tasklist and log in with the credentials “demo / demo”. Click on the button to start a process instance. This opens a dialog where you can select *Payment Retrieval* from the list. Now you can set variables for the process instance using a generic form.

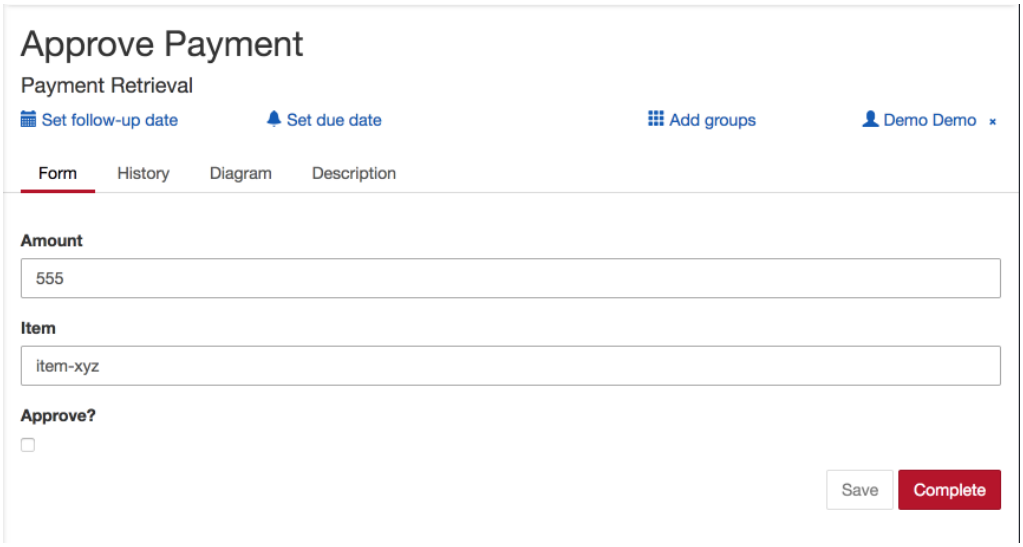


The generic form can be used whenever you have not added a dedicated form for a User Task or a Start Event. Click on the *Add a variable* button to create a new row. Fill in the form as shown in the screenshot. When you're done, click *Start*.

You should now see the *Approve Payment* task in your Tasklist. Select the task and click on the *Diagram* tab. This displays the process diagram highlighting the User Task that's waiting to be worked on.



To work on the task, select the *Form* tab. Because we defined the variables in the Form Tab in the Camunda Modeler, the Tasklist has automatically generated form fields for us.



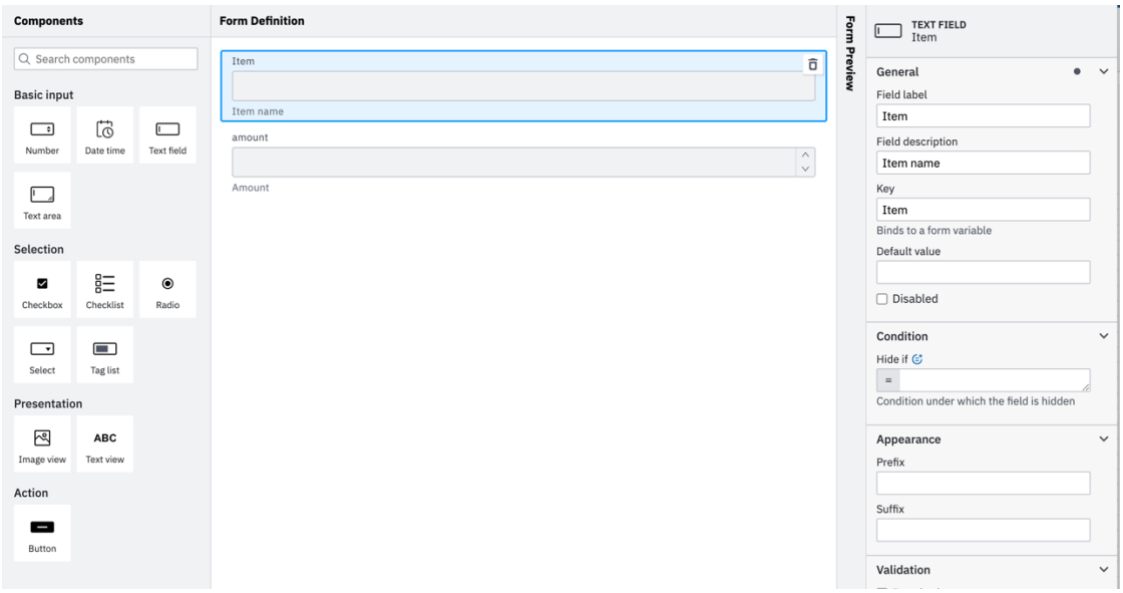
D. Yet, another option using Camunda forms

Instead of hard coding the form directly in the user task you can create the form separately, deploy it, and then refer to it in your process. However, remind that Camunda is a process engine, not an optimized platform for user interaction. When intensive user interactions are required consider the usage of solutions such as https://github.com/formio/formio.

E.1. Create a new form, using the option: File > New File > Form (Camunda Platform 7)

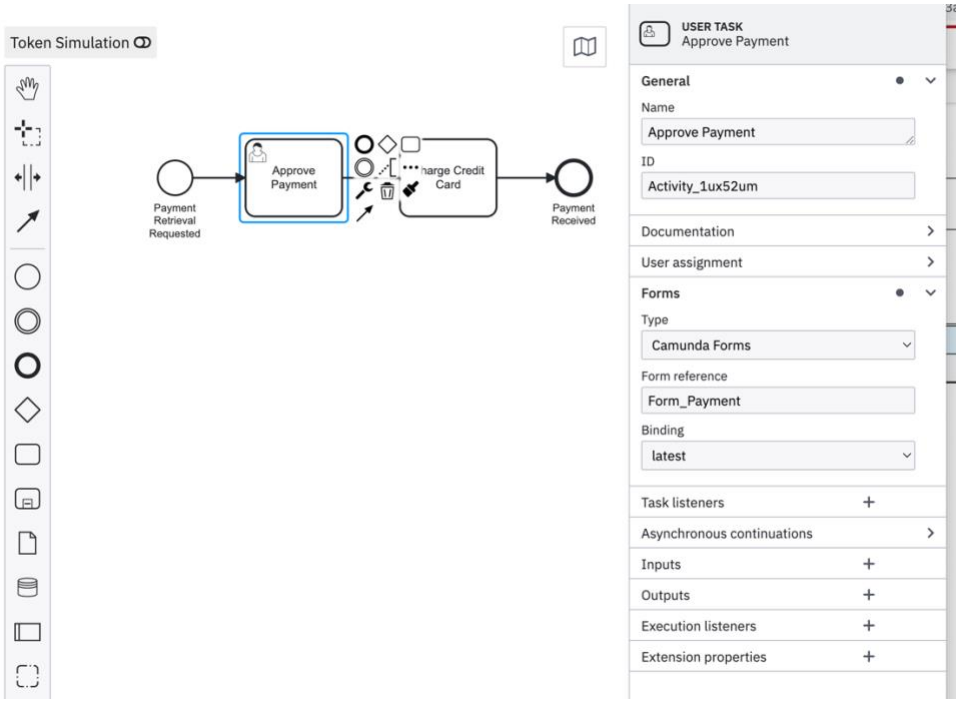
E.2. Define an ID for the entire form, i.e., Form_Payment

E.3. Add 1 text field and 1 number, similar with the following figure:



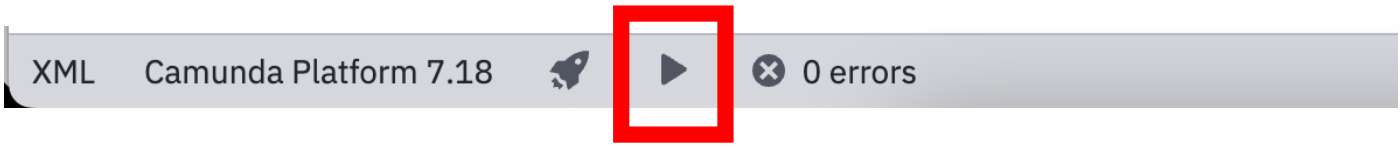
E.4. Deploy the form.

E.5. Change the configuration of the previous user task accordingly with the following figure:

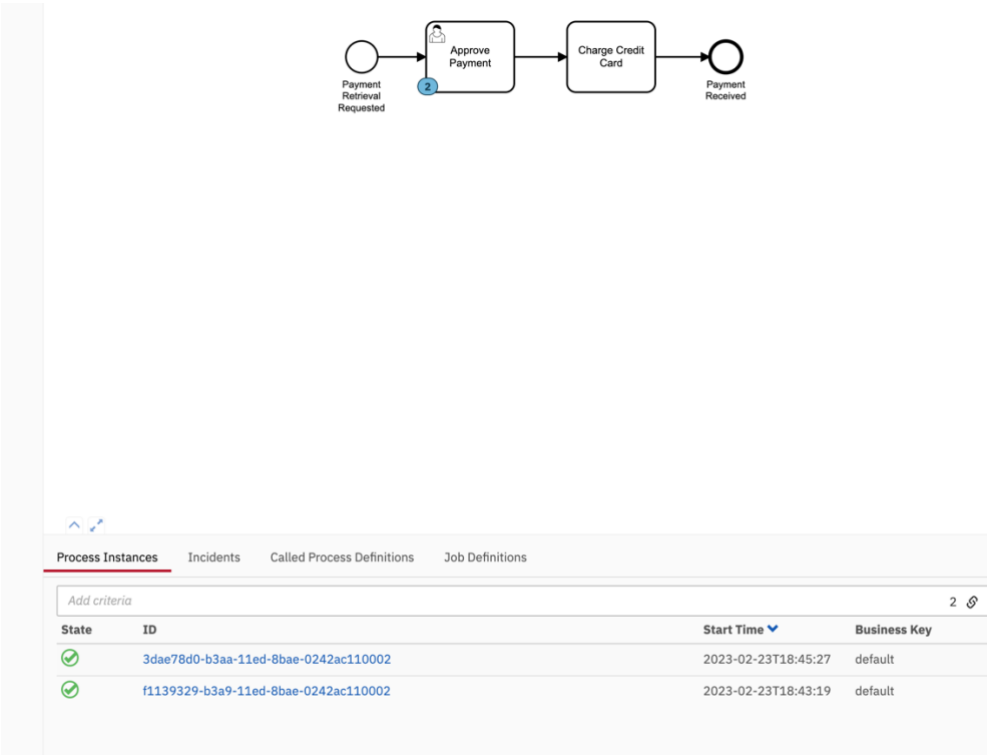


E.6. Deploy the process

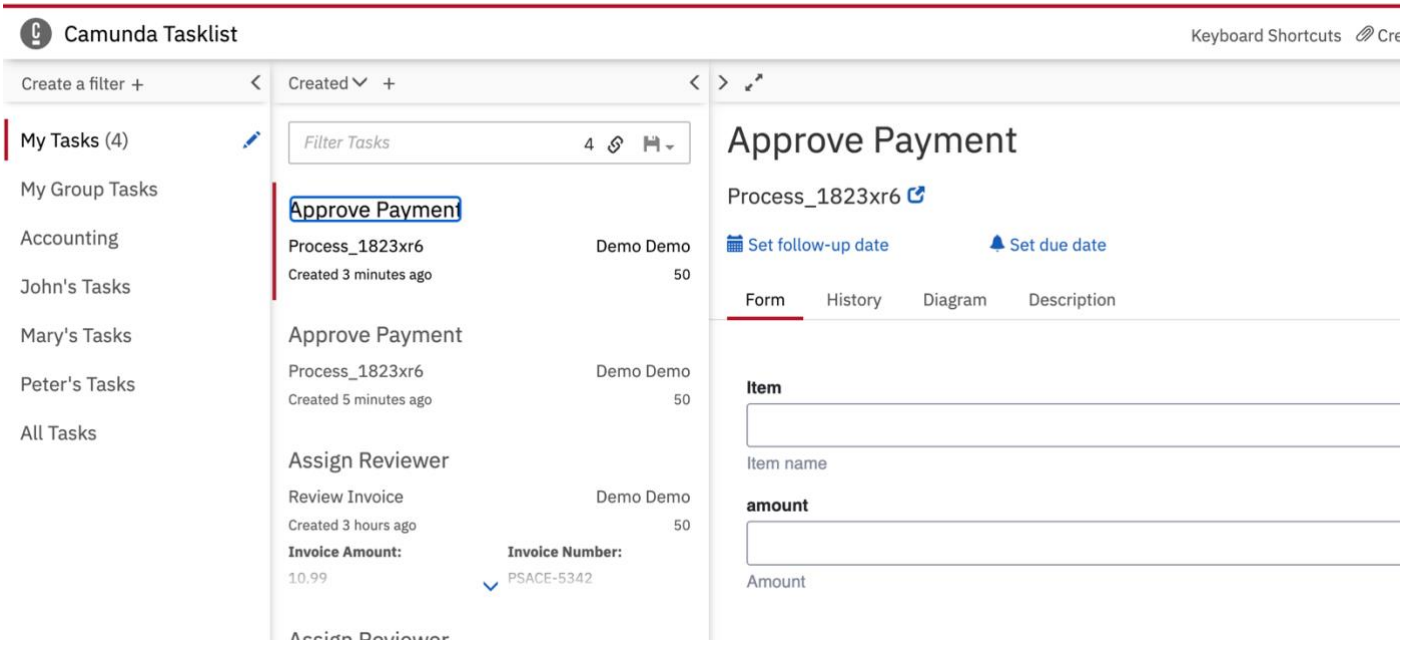
E.7. Start a process instance, using the **`Start instance`** Button in the Camunda Modeler to deploy the updated process to Camunda.



E.8. Check in the cockpit that a pending instance is waiting for a user task:



E.9. Choose your process instance, and then change to user tasks tab, and select one pending action in the task list. You can now see your form!



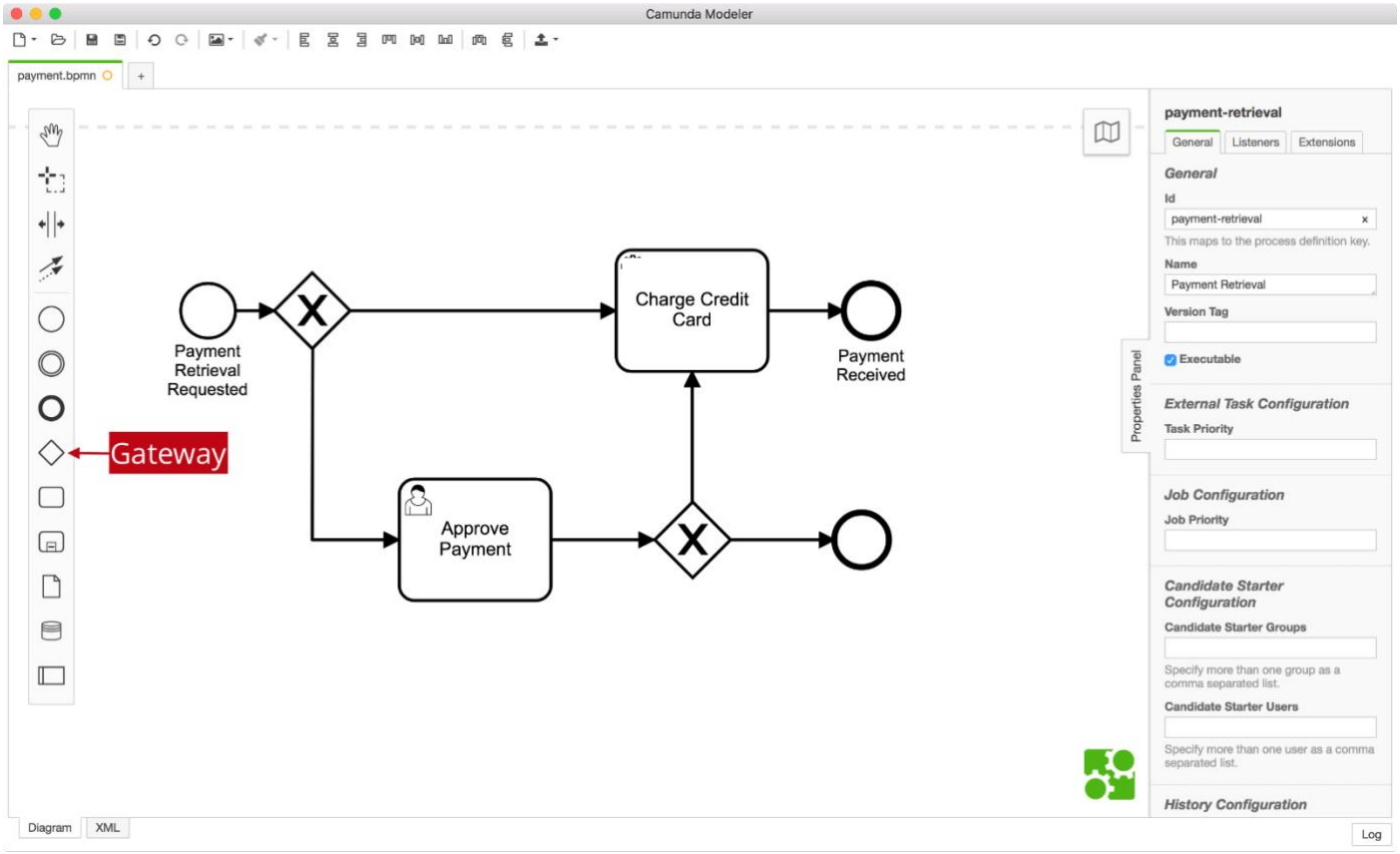
E. Add Gateways to the Process

This example is based in the Camunda documentation available at https://docs.camunda.org/get-started/quick-start/gateway/ . The goal of the example is to implement decisions in the flow of the business process, using the BPMN gateway element.

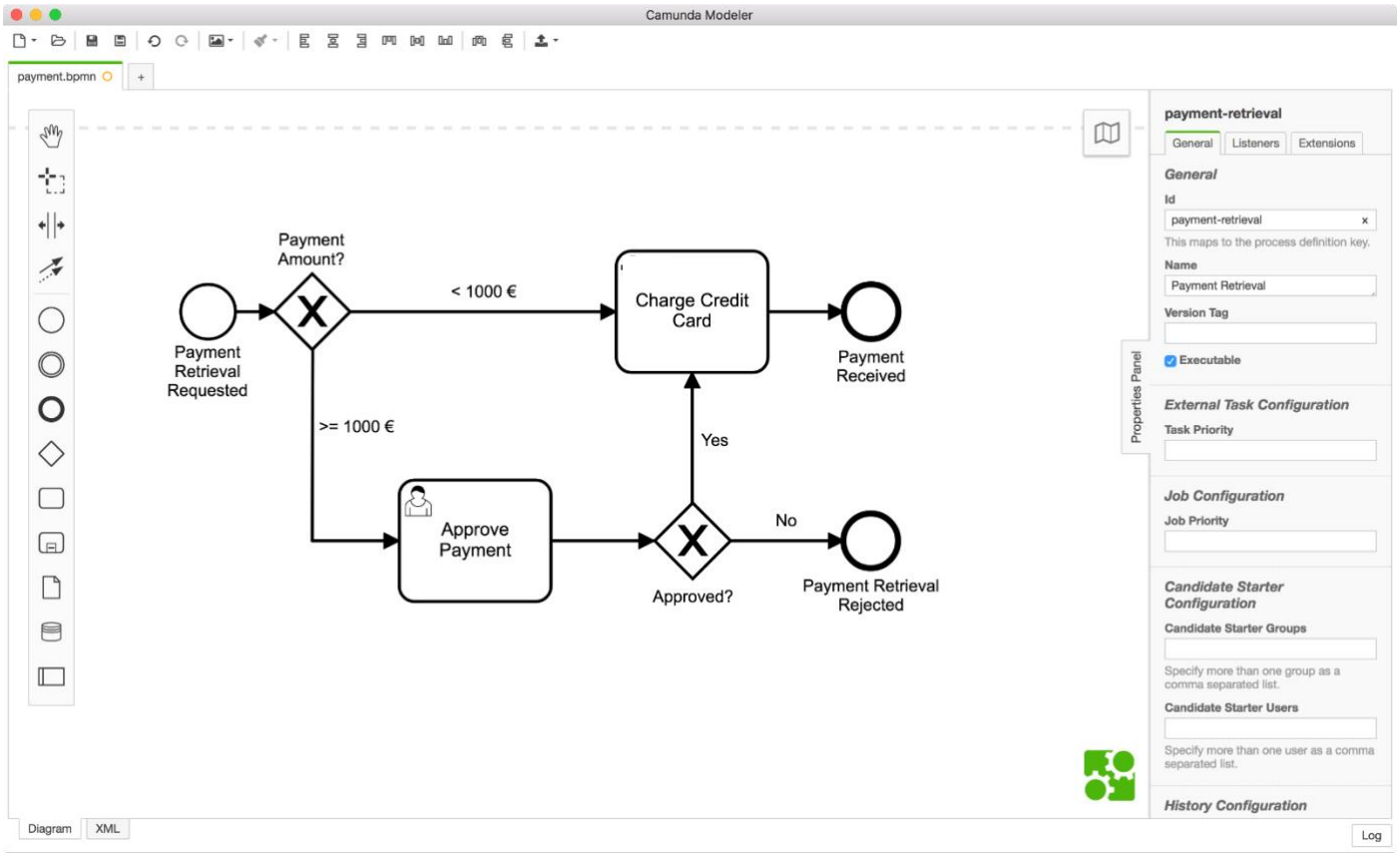
F.1. Add Two Gateways

We want to modify our process so that it is more dynamic. To do so, open the process in the Camunda Modeler.

Next, from the Modeler's left-hand menu, select the gateway shape (diamond) and drag it into position between the Start Event and the Service Task. Move the User Task down and add another Gateway after it. Lastly, adjust the Sequence Flows so that the model looks like this:

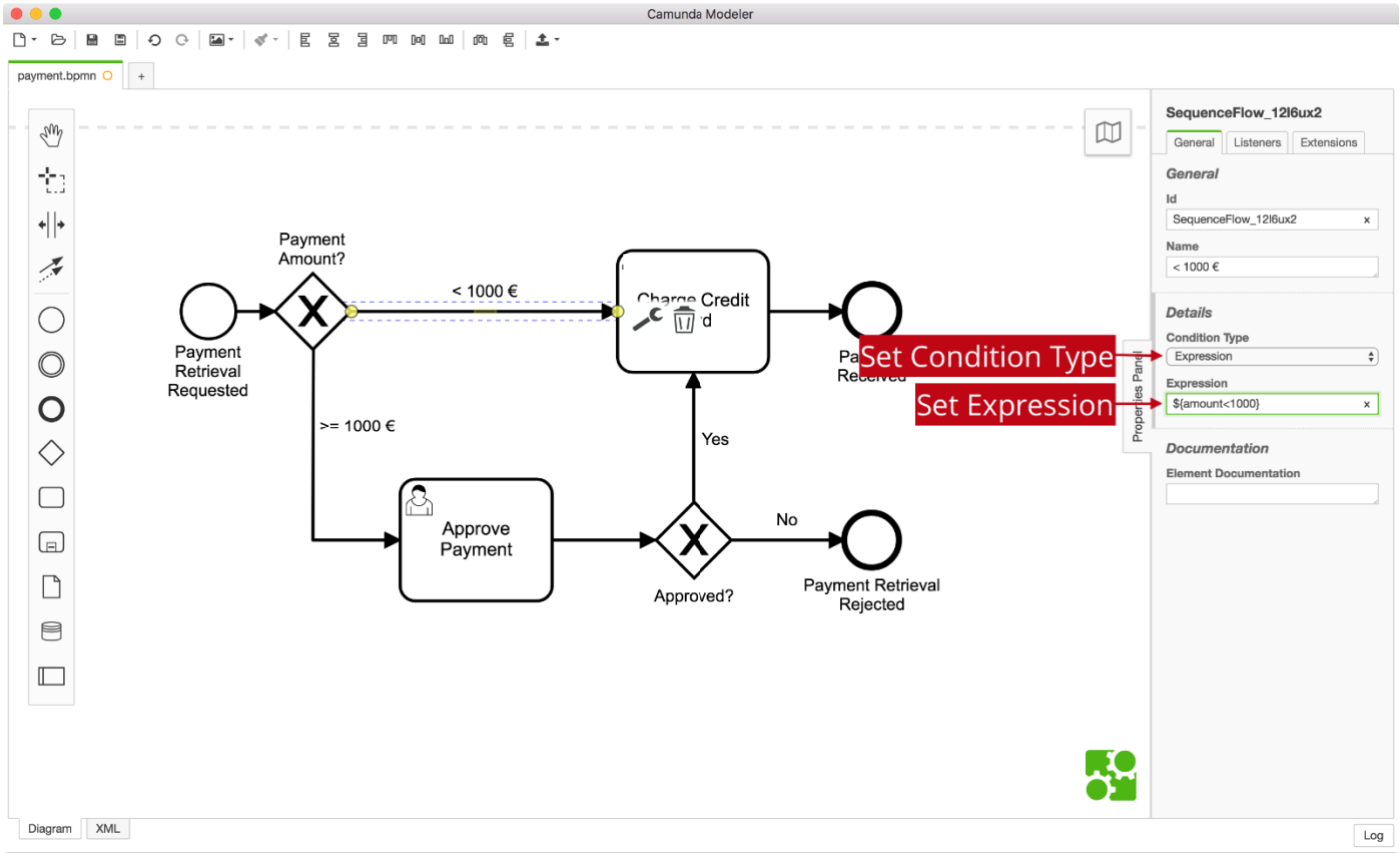


Now also name the new elements accordingly:



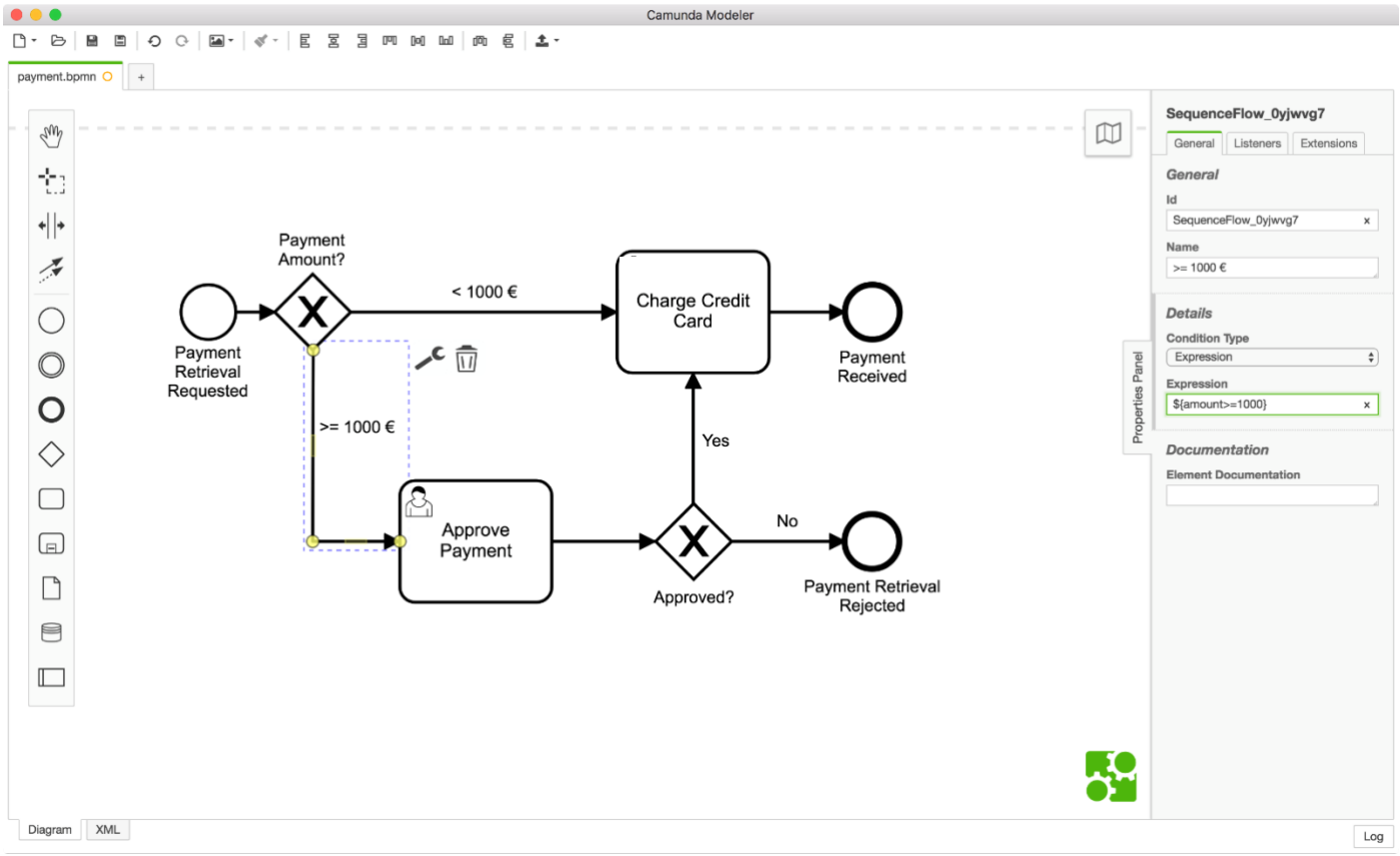
F.2. Configure the Gateways

Next, open the properties panel and select the `<1000 €` Sequence Flow after the Gateway on the canvas. This will update the selection in the properties panel. Scroll to the property named `Condition Type` and change it to `Expression`. Then input `${amount<1000}` as the Expression. We are using the [Java Unified Expression Language] to evaluate the Gateway.

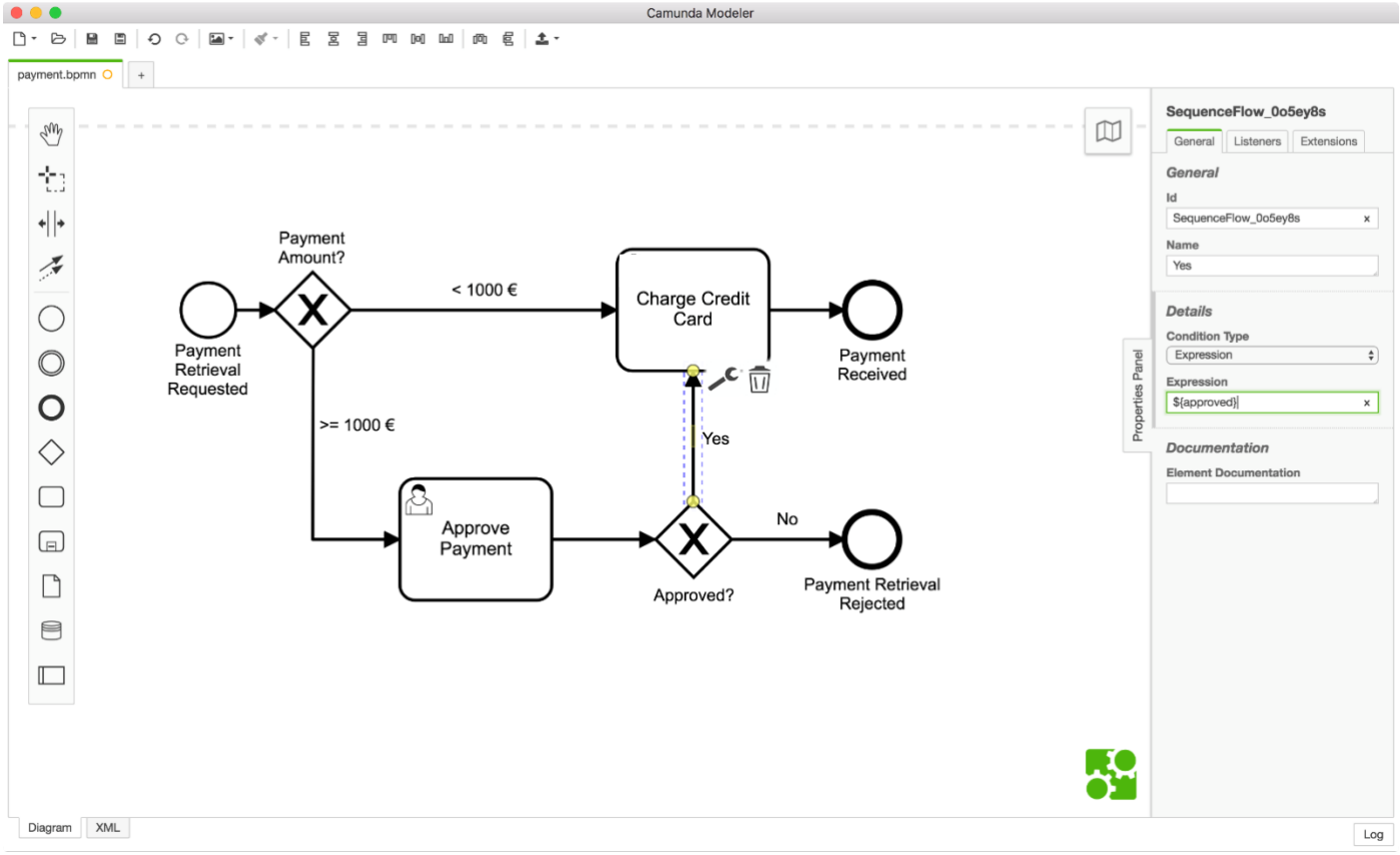


Next, change the Expressions for the other Sequence Flows, too.

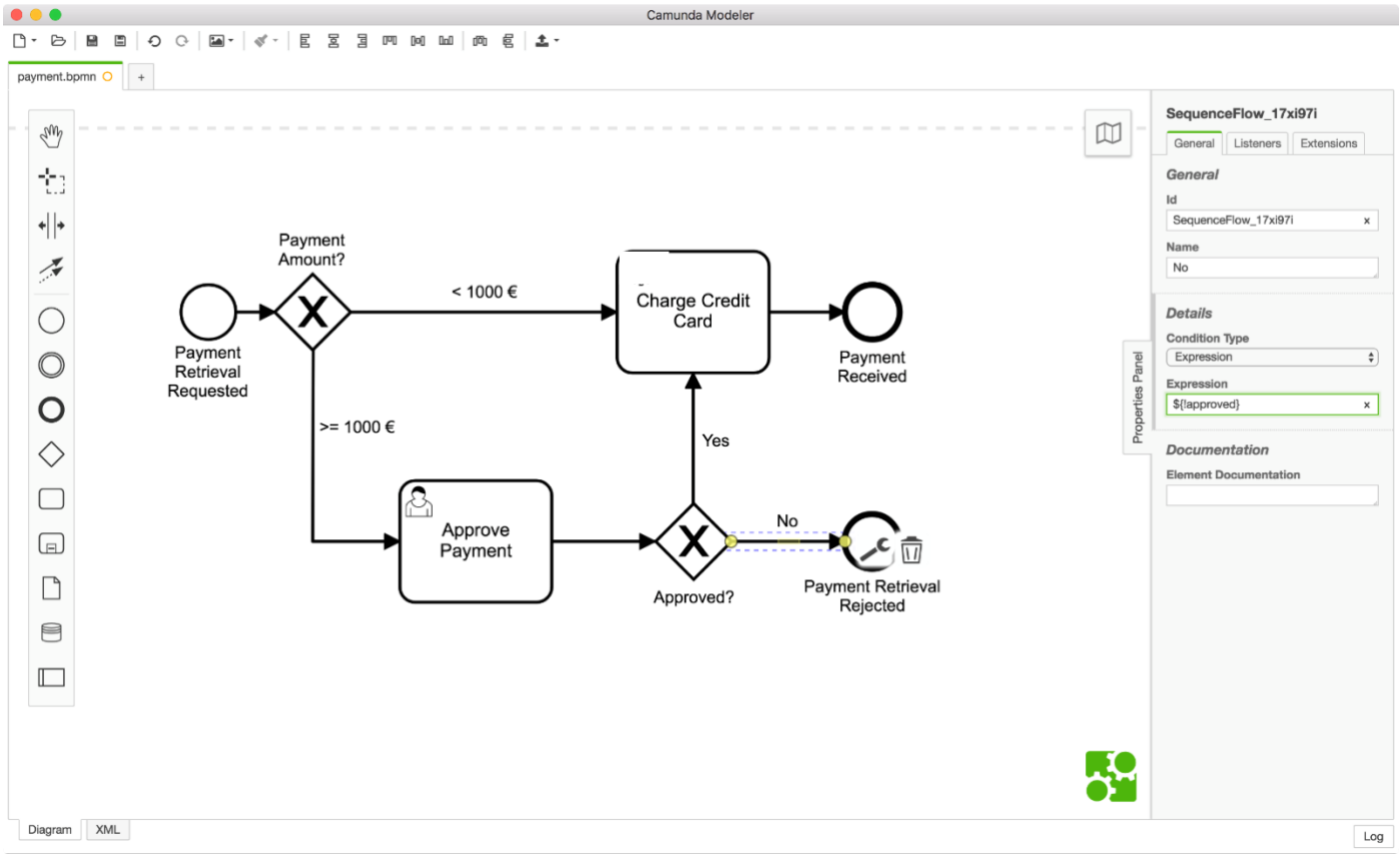
For the `>=1000 €` Sequence Flow, use the Expression `${amount>=1000}`:



For the `Yes` Sequence Flow, use the Expression `${approved}`:



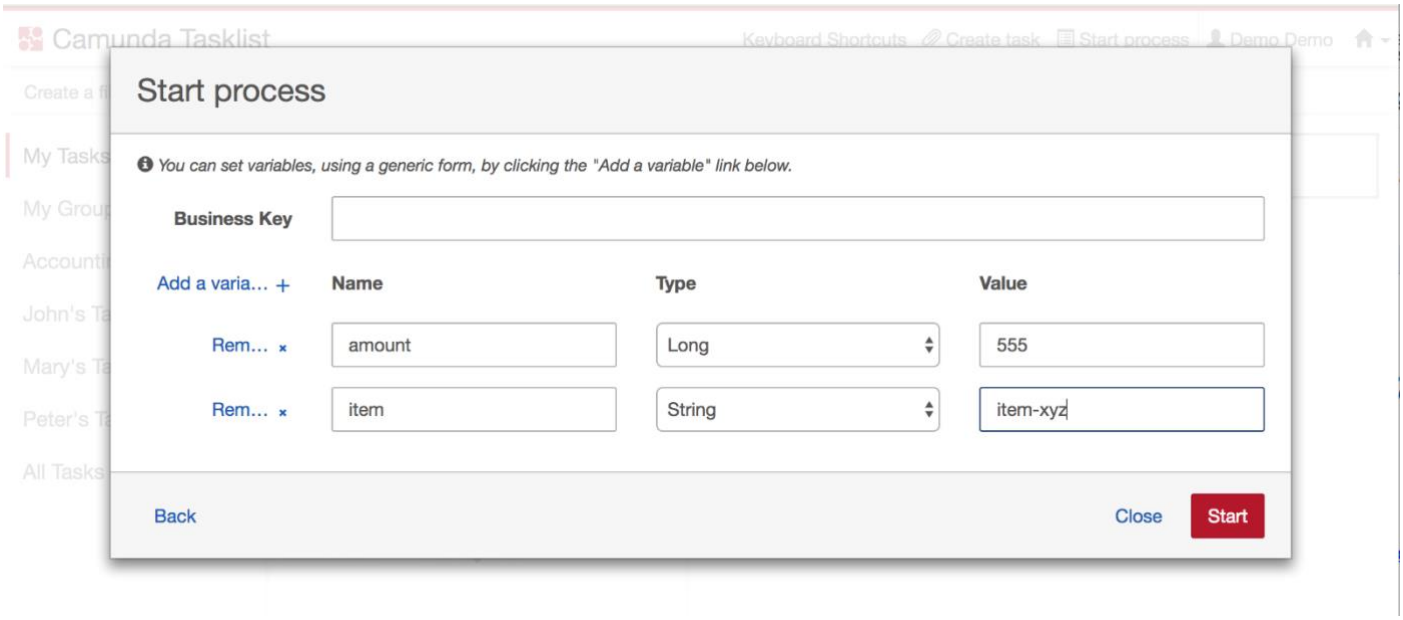
For the `No` Sequence Flow, use the Expression `${!approved}`:



F.3. Deploy the Process

F.4. Work on the Task

Go to Tasklist and log in with the credentials "demo / demo". Click on the "Start process" button to start a process instance for the *Payment Retrieval* Process. Next, set variables for the process instance using the generic form as we learned in the *User Tasks* section.



Fill in the form as shown in the screenshot and make sure you use an amount that is larger or equal to 1000 in order to see the User Task *Approve Payment*. When you are done, click *Start*.

You should see the *Approve Payment* task when you click on *All Tasks*. In this quick start, we're logged into Tasklist as an admin user, and so we can see all tasks associated with our processes. However, it's possible to create filters in Tasklist to determine which users can see which tasks based on user authorization as well as other criteria.

To work on the task, select the *Form* tab and check the *approved* checkbox so that our payment retrieval gets approved. We should see that our worker prints something to the console.

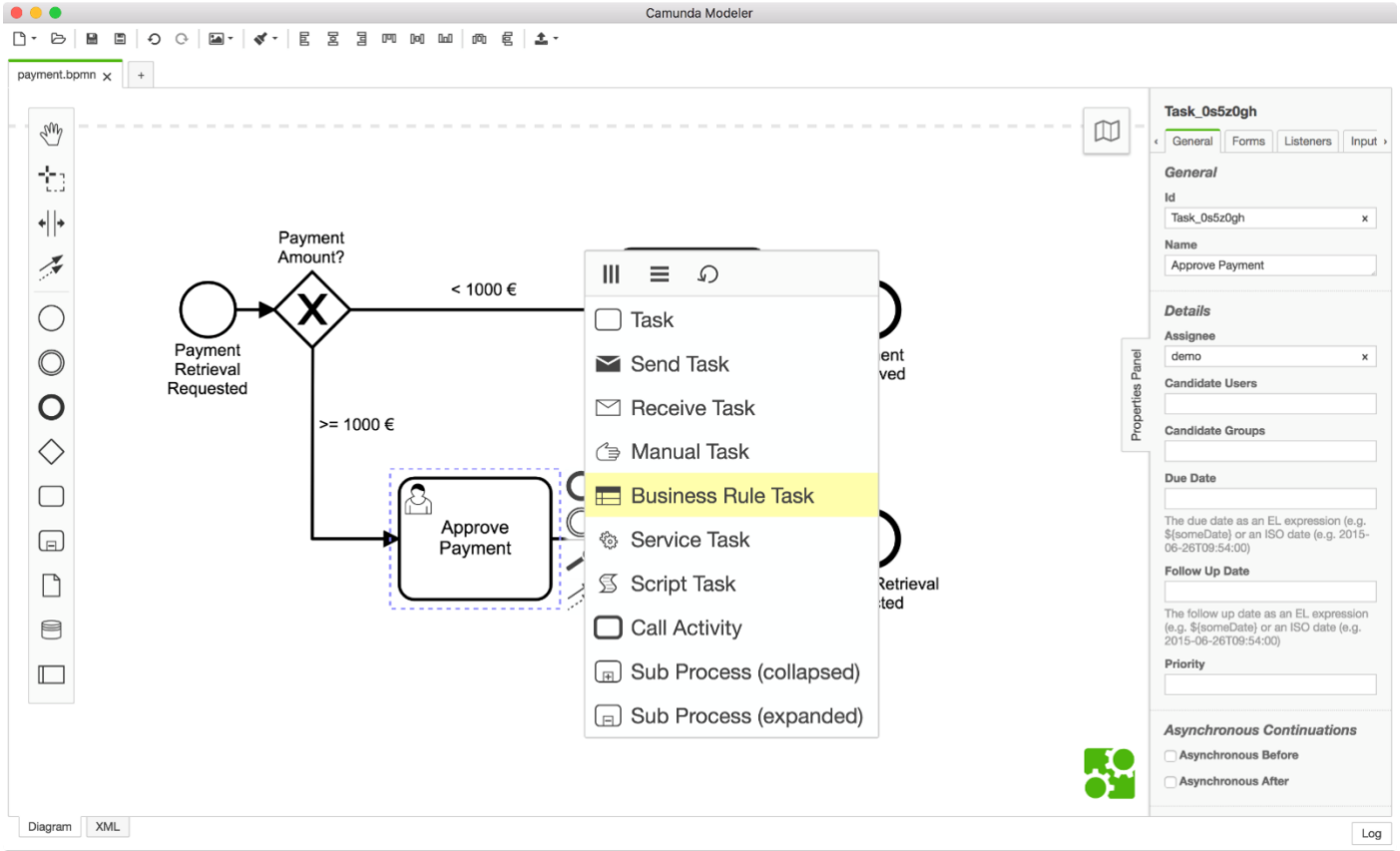
Next, repeat the same steps, and this time, reject the payment. You should also create one instance with an amount less than 1000 to confirm that the first gateway works correctly.

F. Business rule enforcement

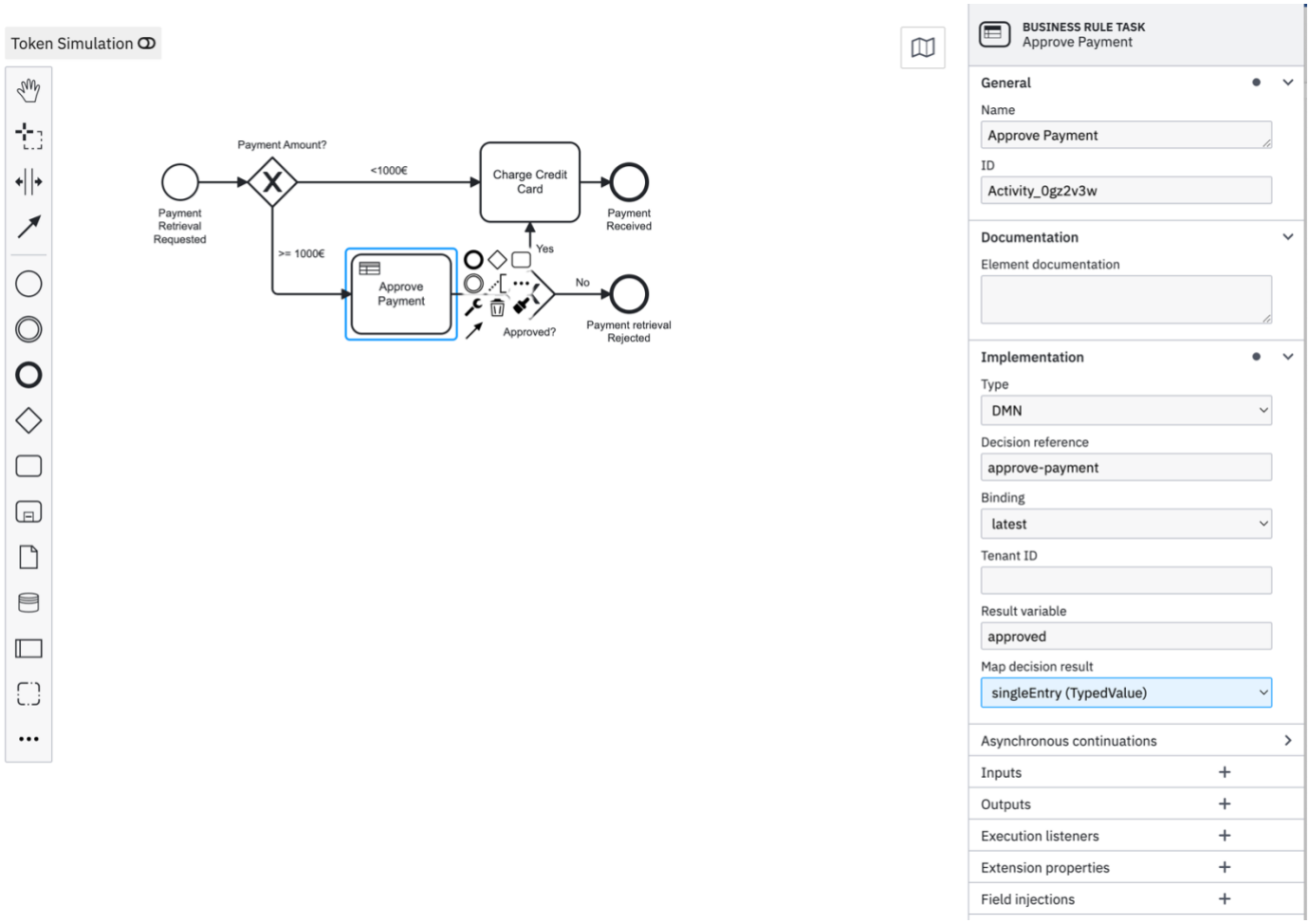
This example is based in the Camunda documentation available at https://docs.camunda.org/get-started/quick-start/decision-automation/. The goal of this example is to implement business rules using the DMN 1.1 standard.

G.1. Add a Business Rule Task to the Process

Use the Camunda Modeler to open the Payment Retrieval process then click on the Approve Payment Task. Change the activity type to *Business Rule Task* in the wrench button menu.



Next, link the Business Rule Task to a DMN table by changing `Implementation` to `DMN` and `Decision Ref` to `approve-payment` in the properties panel. In order to retrieve the result of the evaluation and save it automatically as a process instance variable in our process, we also need to change the `Result Variable` to `approved` and use `singleEntry` as the `Map Decision Result` in the properties panel.



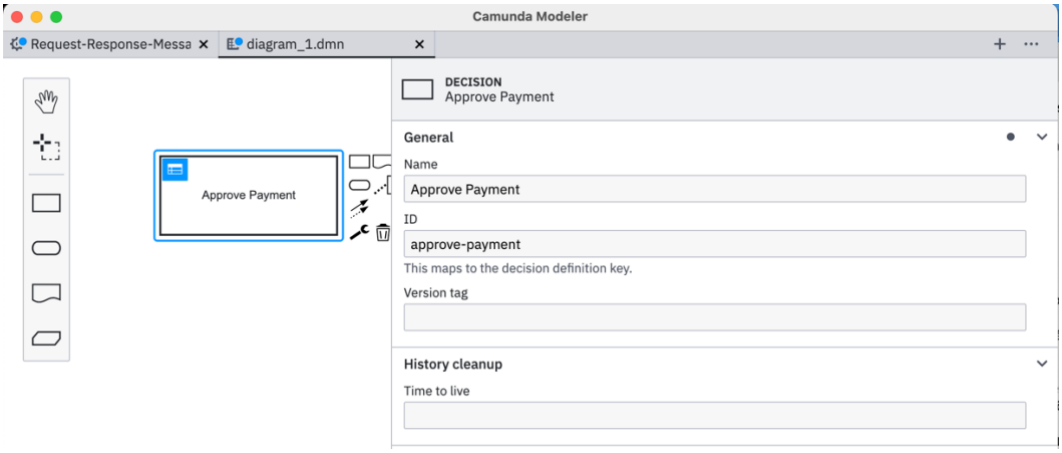
Save your changes and deploy the updated process using the `Deploy` Button in the Camunda Modeler.

G.2. Create a DMN table using the Camunda Modeler

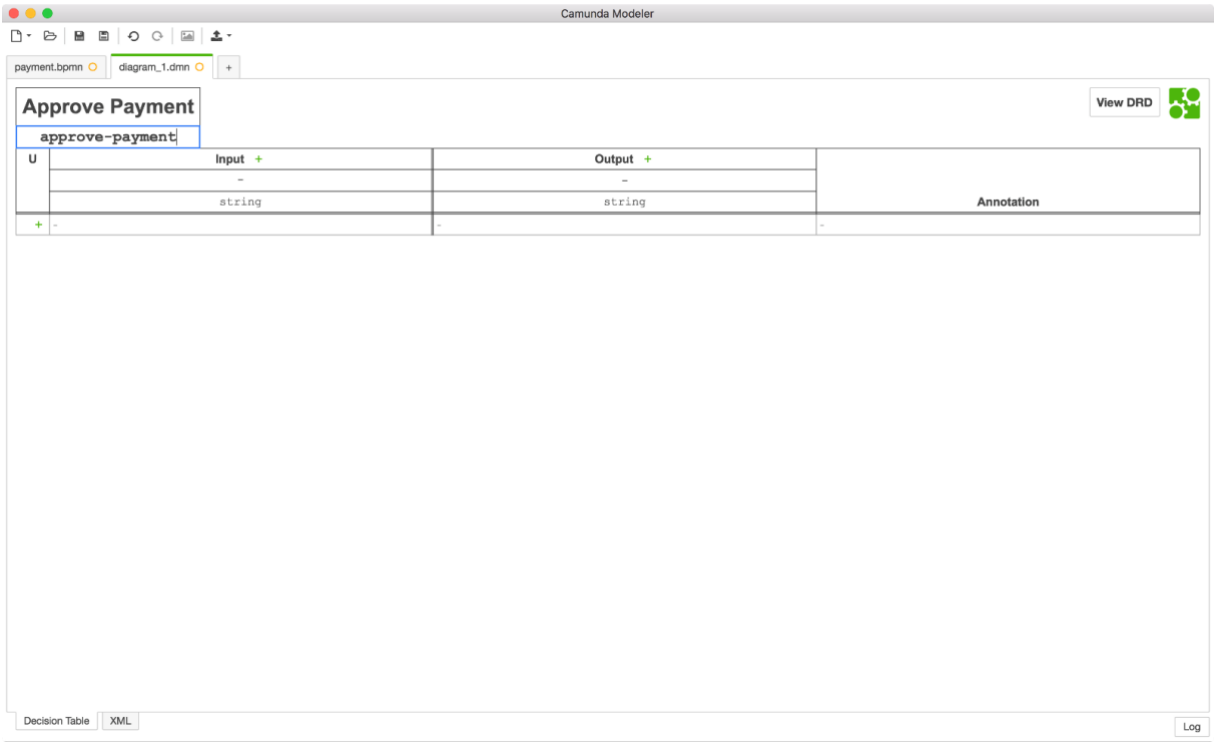
Create a new DMN table by clicking *File > New File > DMN Diagram (Camunda Platform 7)*.

G.3. Specify the DMN table

First, give the DMN table the name Approve Payment and the ID approve-payment, and then double click on it, as following.

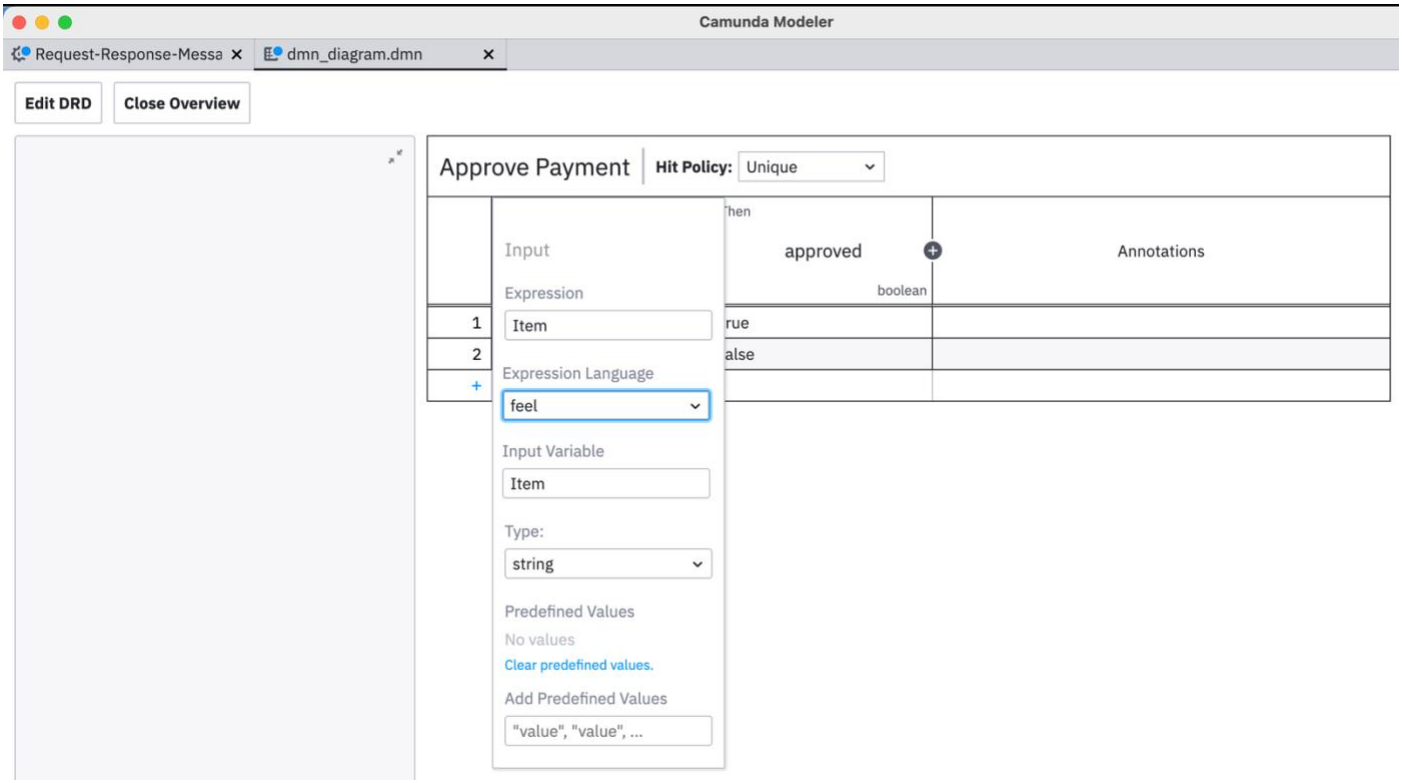


The DMN table ID will match the `Decision Ref` in your BPMN process.

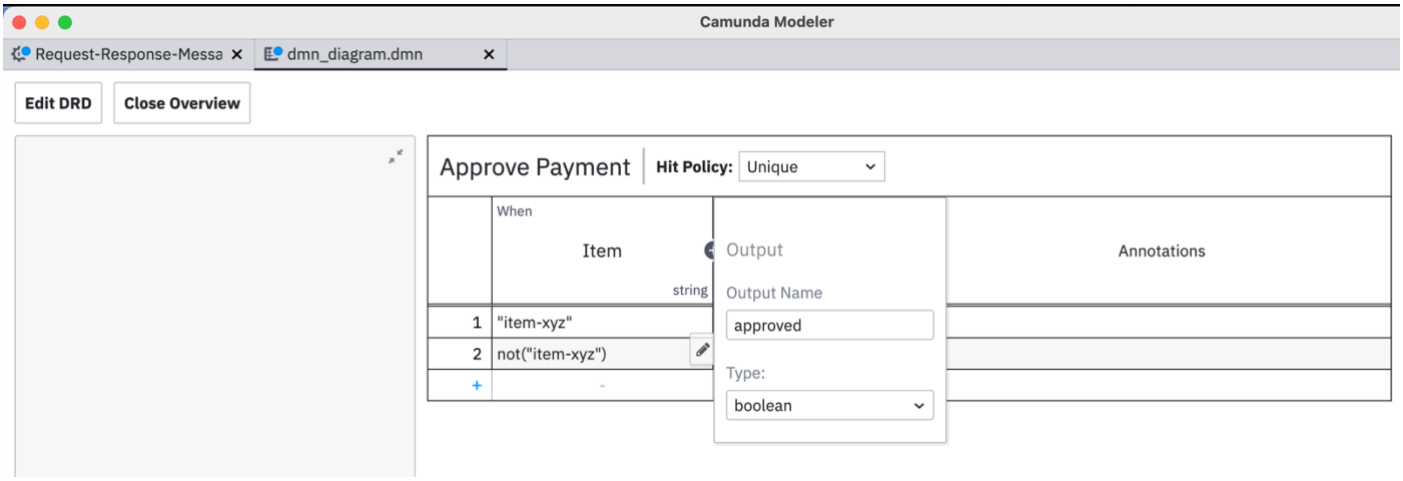


Next, specify the input expressions for the DMN table. In this example, we'll decide whether a payment is approved based on the item name. Your rules can also make use of the FEEL Expression Language, JUEL or Script. If you like, you can read more about Expressions in the DMN Engine.

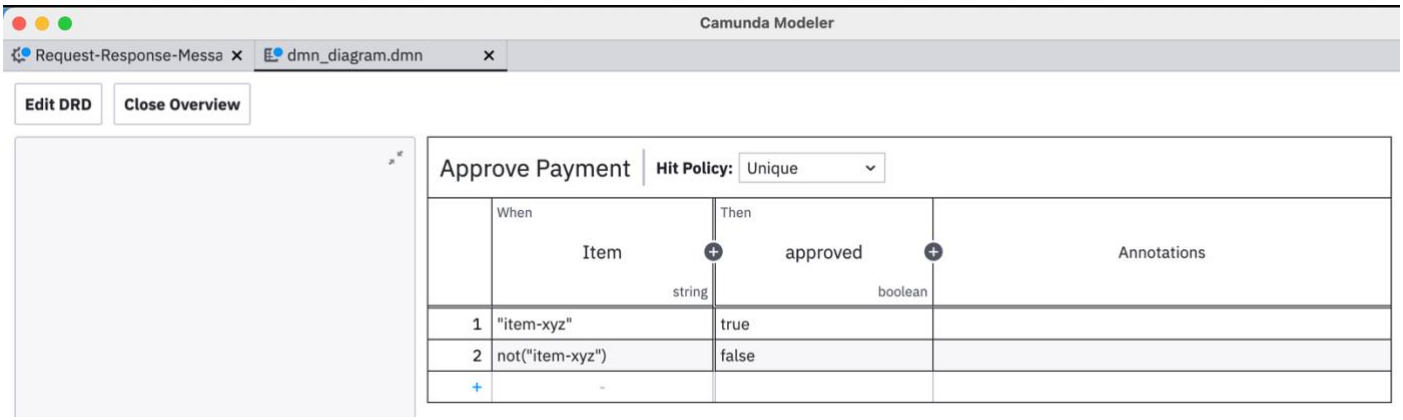
For the input column, use the following configuration:



Next, set up the output column, as following:

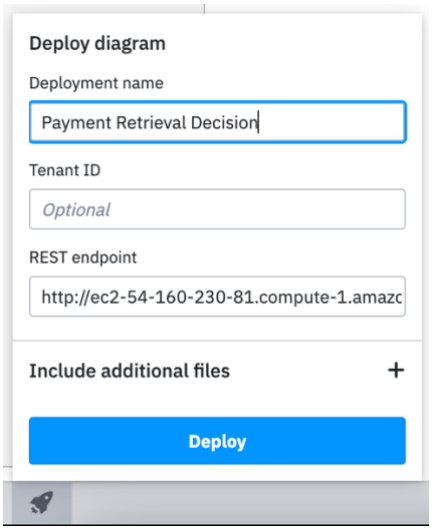


Let's create some rules by clicking on the plus icon on the left side of the DMN table. After setup, your DMN table should look like this:



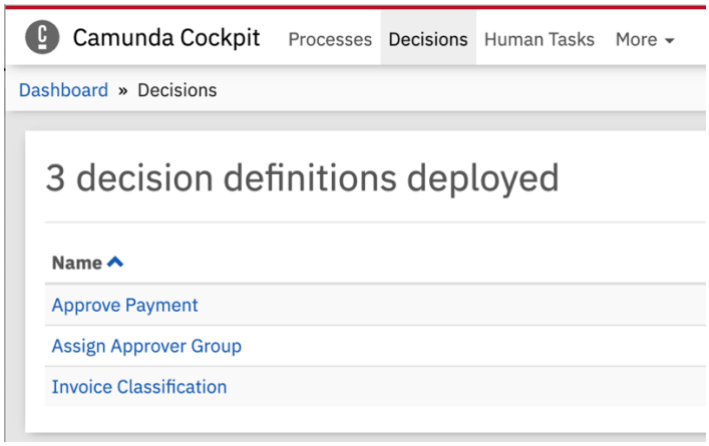
G.4. Deploy the DMN table

To deploy the Decision Table, click on the Deploy button in the Camunda Modeler, give it Deployment Name "**Payment Retrieval Decision**", then hit the Deploy button.



G.5. Verify the Deployment with Cockpit

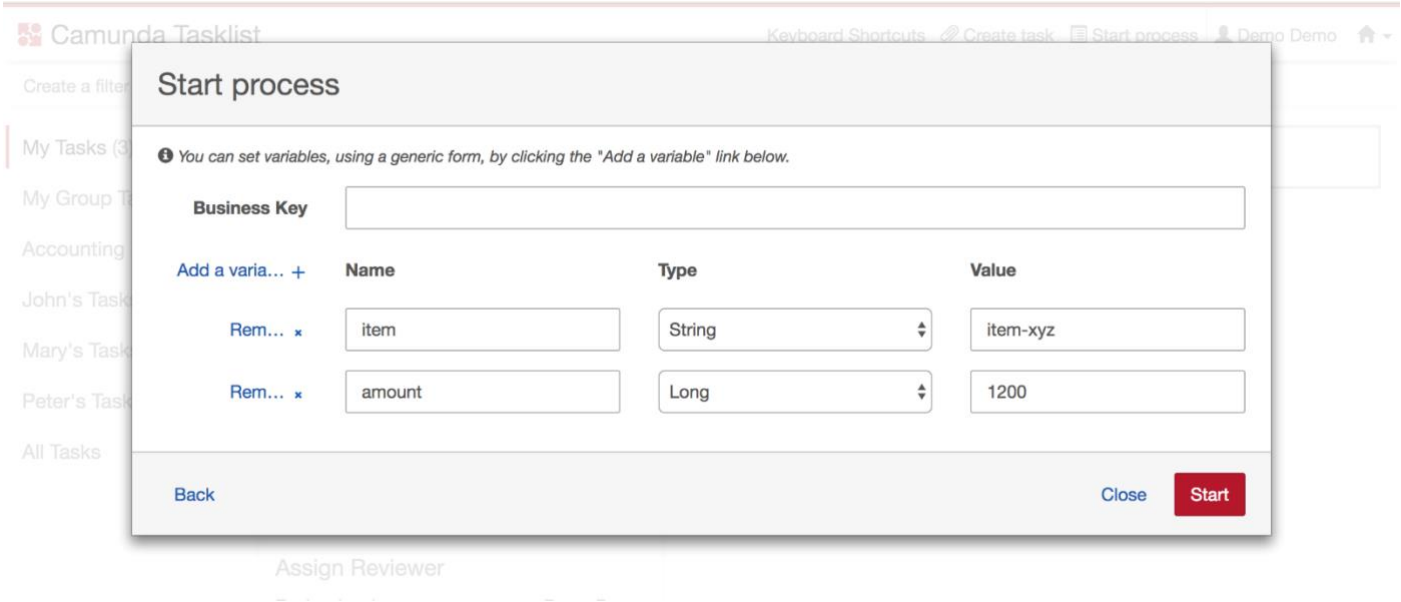
Now, use Cockpit to see if the decision table was successfully deployed. Log in with the credentials *demo / demo*. Navigate to the "Decision definitions" section. Your decision table *Approve Payment* should be listed as deployed decision definition.



G.6. Inspect using Cockpit and Tasklist

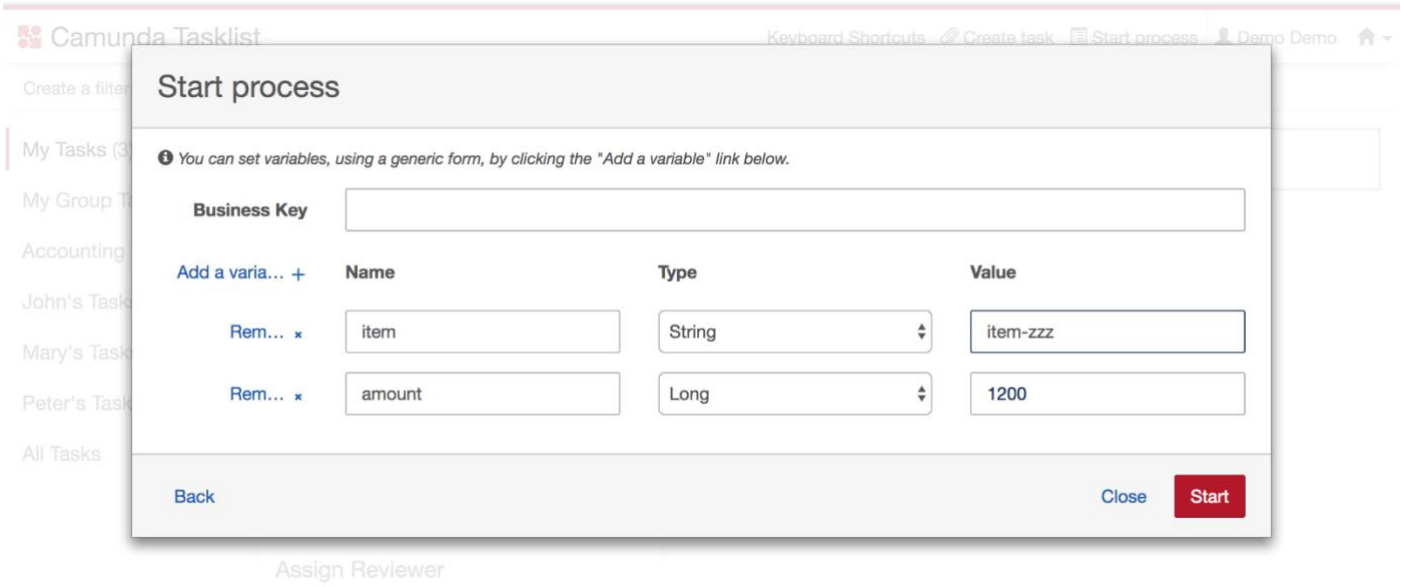
Next, use Tasklist to start two new Process Instances and verify that depending on your input, the Process Instance will be routed differently.

Click on the Start process button to start a process instance and choose the `Payment` process. Use the generic form to add the variables as follows:

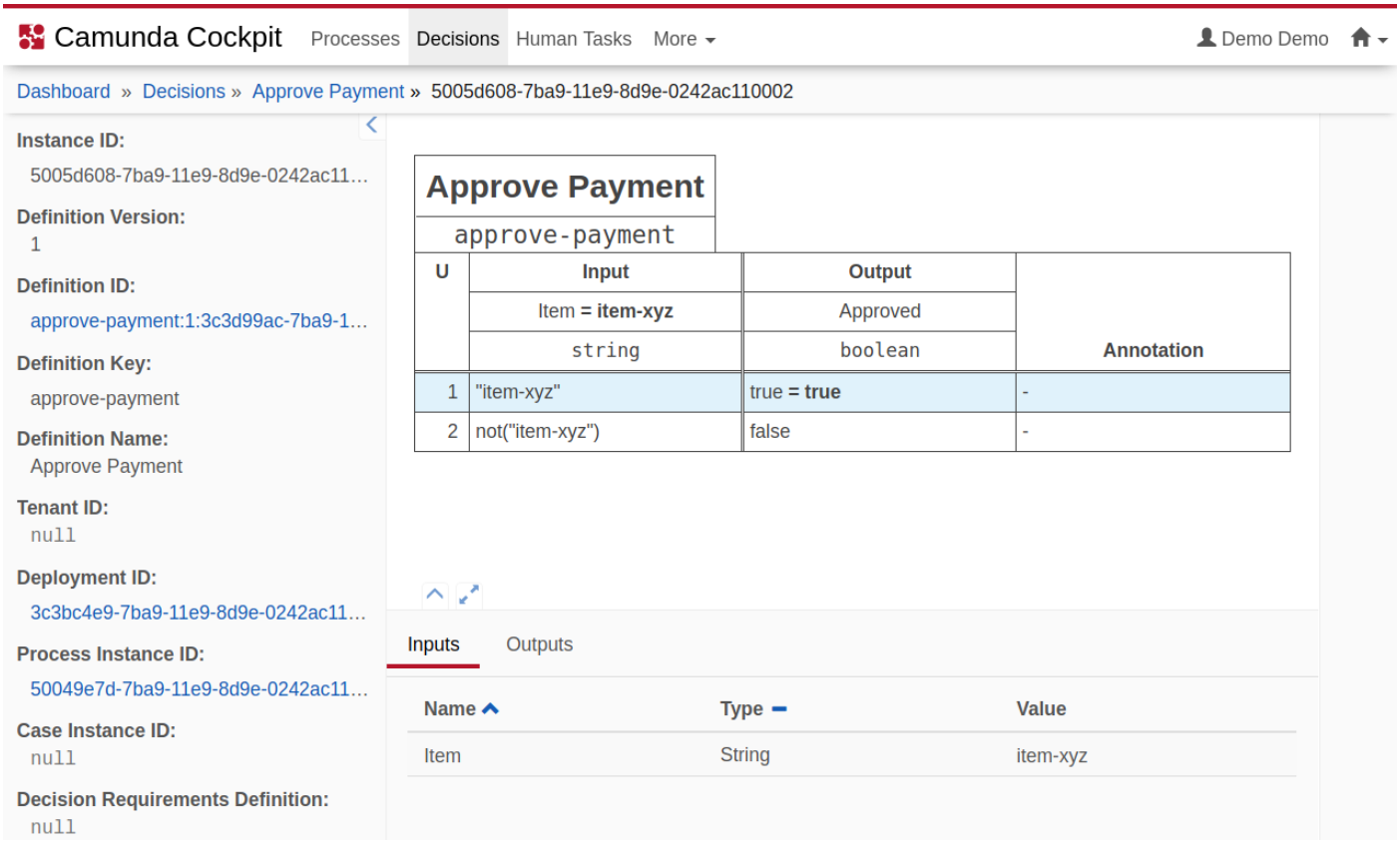


Hit the Start Instance button.

Next, click again on the Start process button to start another process instance and choose the `Payment` process. Use the generic form to add the variables as follows:



You will see that depending on the input, the worker will either charge or not charge the credit card. You can also verify that the DMN tables were evaluated by using Camunda Cockpit. Log in with the credentials *demo / demo*. Navigate to the "Decisions" section and click on Approve Payment. Check the different Decision Instances that were evaluated by clicking on the ID in the table. A single DMN table that was executed could look like this in Camunda Cockpit:



G. Integrate with a REST API installed remotely using Camunda http-connector – REQUEST ONLY

This example shows how to invoke a REST API from a **BPMN service task using the Camunda connector if indications for installation in section B were followed**. You need to create the service task element in the BPMN model as explain previously in this tutorial. Then, to invoke a REST API during the BPMN execution the following configuration in the Camunda modeller is required.

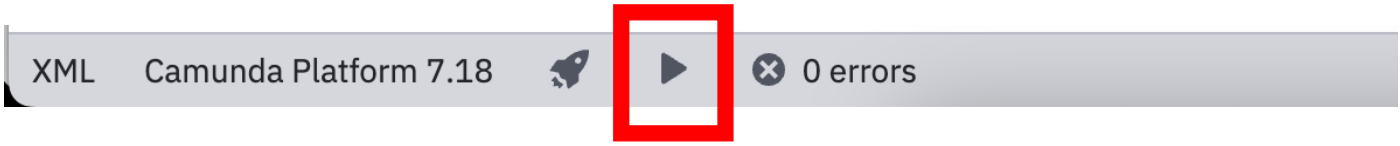
H.1. Configure your task as a service task and choose the "**Connector**" in the implementation details, the connector ID should be "**http-connector**".

H.2. Then, add three "**Connector inputs**":

a. "**method**": of type String or expression with value = POST
b. "**payload**": of type String or expression with value containing the JSON to be send with the request
c. "**url**": of type String or expression with value = endpoint previously available, *e.g.*, Kong API or AWS Gateway API, or other

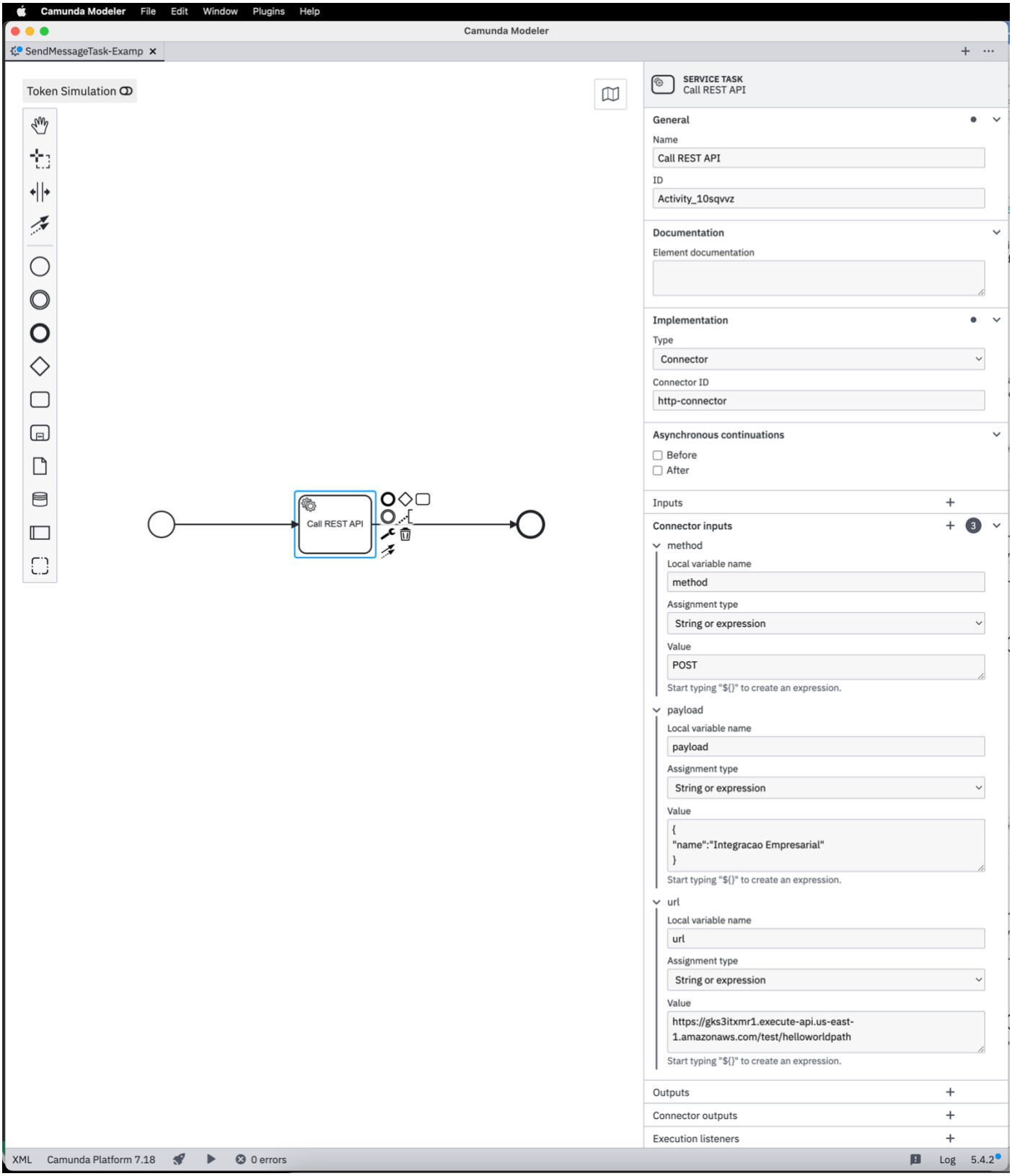
H.3. Deploy the BPMN model

H.4. Start a process instance, using the **`Start instance`** Button in the Camunda Modeler to deploy the updated process to Camunda.



H.5. Verify in the REST API endpoint that it was invoked. For instance, in AWS CloudWatch.

The previous configuration is also depicted in the following figure:


Camunda Modeler File Edit Window Plugins Help
Camunda Modeler
SendMessageTask-Examp
Token Simulation
Call REST API
SERVICE TASK
Call REST API
General
Name
Call REST API
ID
Activity_10sqvvz
Documentation
Element documentation
Implementation
Type
Connector
Connector ID
http-connector
Asynchronous continuations
Before
After
Inputs
Connector inputs
3
method
Local variable name
method
Assignment type
String or expression
Value
POST
Start typing "${}" to create an expression.
payload
Local variable name
payload
Assignment type
String or expression
Value
{
"name":"Integracao Empresarial"
}
Start typing "${}" to create an expression.
url
Local variable name
url
Assignment type
String or expression
Value
https://gks3itxmr1.execute-api.us-east-1.amazonaws.com/test/helloworldpath
Start typing "${}" to create an expression.
Outputs
Connector outputs
Execution listeners
XML
Camunda Platform 7.18
0 errors
Log
5.4.2

H. Integrate with a REST API installed remotely using Camunda http-connector – REQUEST with variable

This example shows how to invoke a REST API from a **BPMN service task using the Camunda connector if indications for installation in section B were followed** and that includes some data previously processed in the BPMN process flow into the REST API Request. You need to create the service task element in the BPMN model as explain previously in this tutorial. Then, to invoke a REST API during the BPMN execution the following configuration in the Camunda modeller is required.

If you need to add any process variable to the web service call you may add it, using the payload, as follows (*CourseName* is used to only for demonstration purposes).

Firstly, a user task is created to allow the input of the course name as text description from the end user. For that, you need to add a Task and choose the **"User Task"** option. Then, assign a user to the task. In this example is used the "demo" user. After that, choose a form of type "Generated Task Forms", and create a Form field:

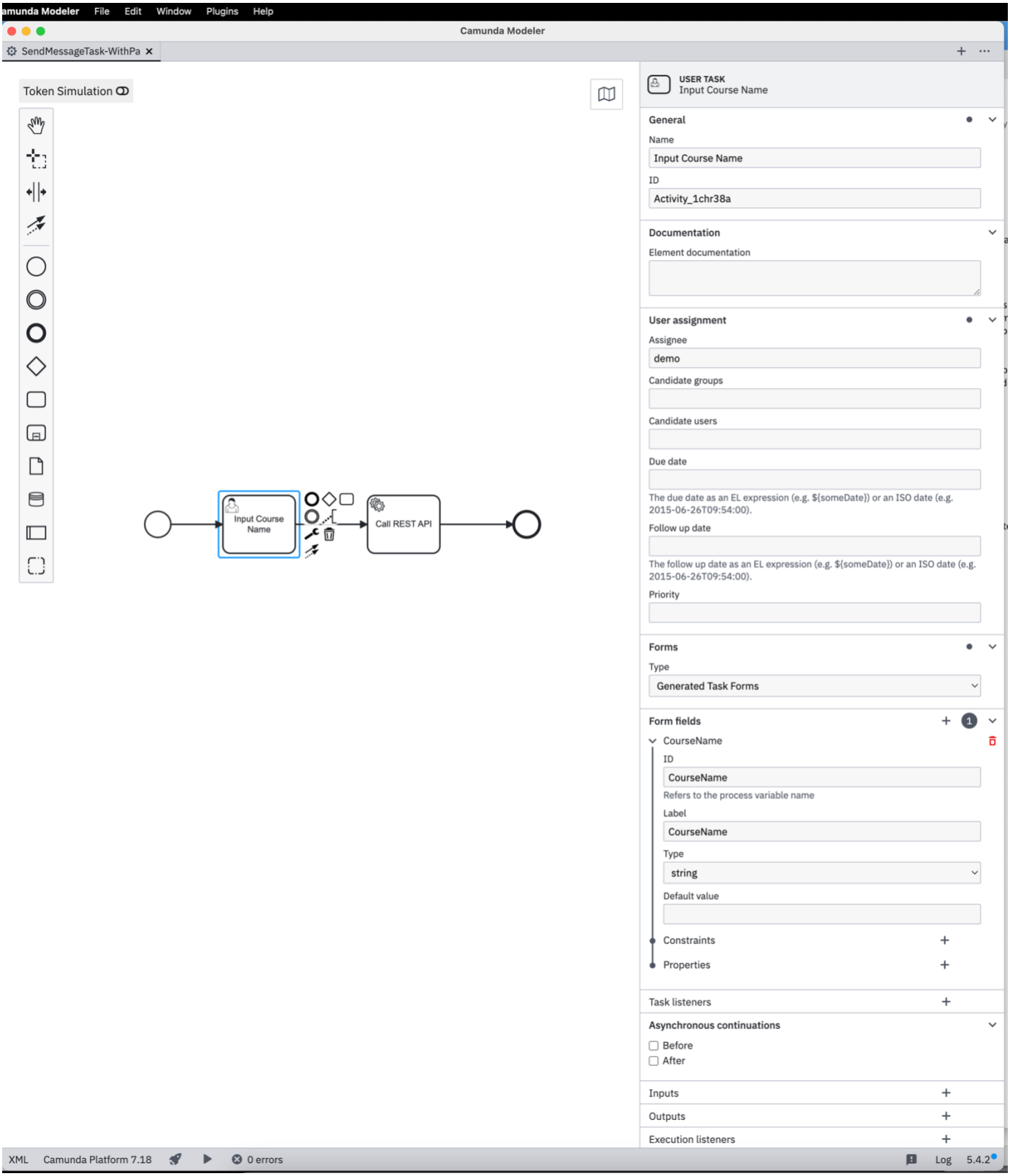
- ID = CourseName
- Label = CourseName
- Type = string

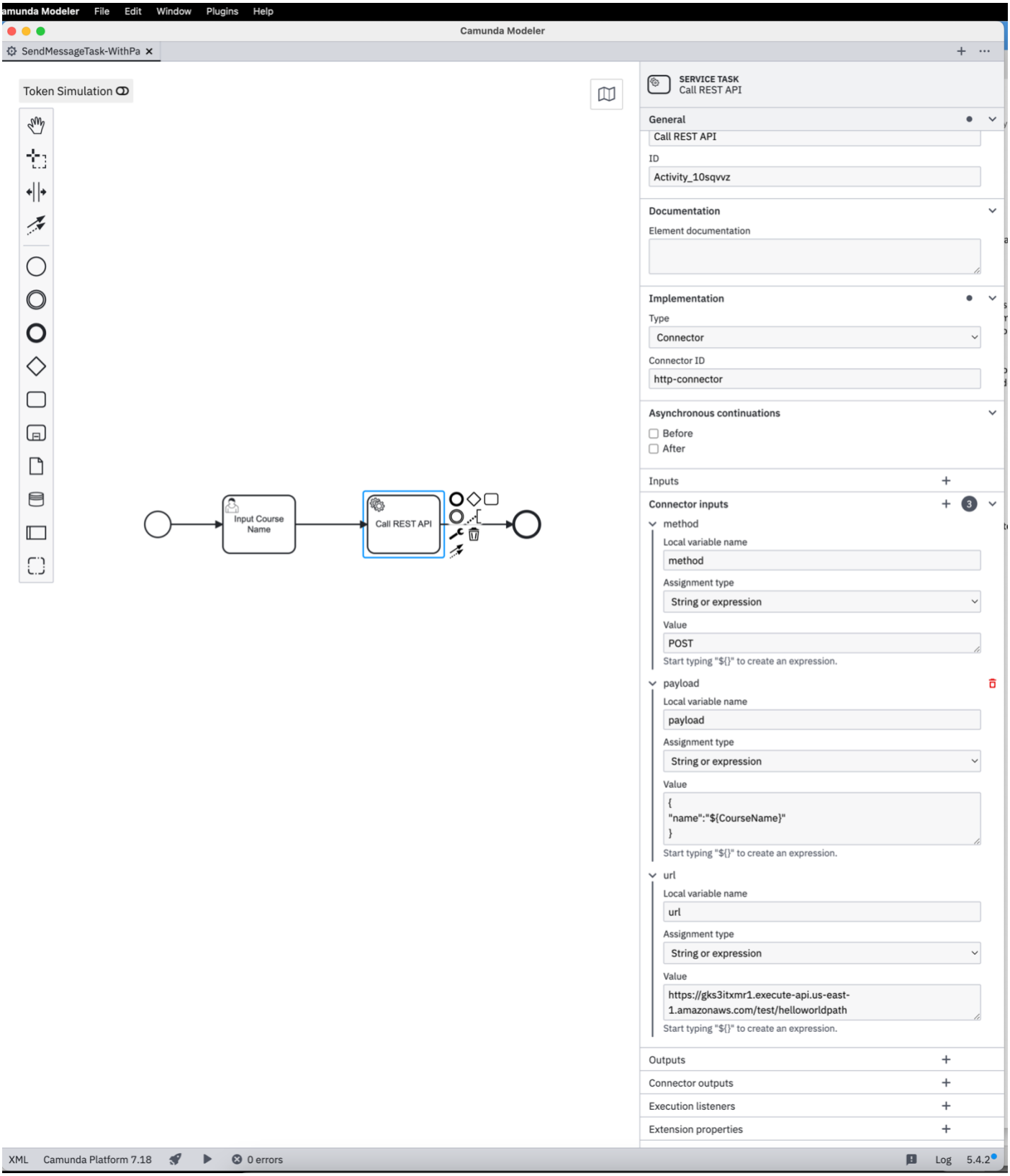
With this configuration you are now accepting the coursename from the user and allocate a variable that could be used afterwards.

Secondly, choose the "Call REST API" task and change the payload to be dynamically created on the previous **CourseName**, using the following syntax:

```
{
"name":"${CourseName}"
}
```

The previous configuration is also depicted in the following figures:

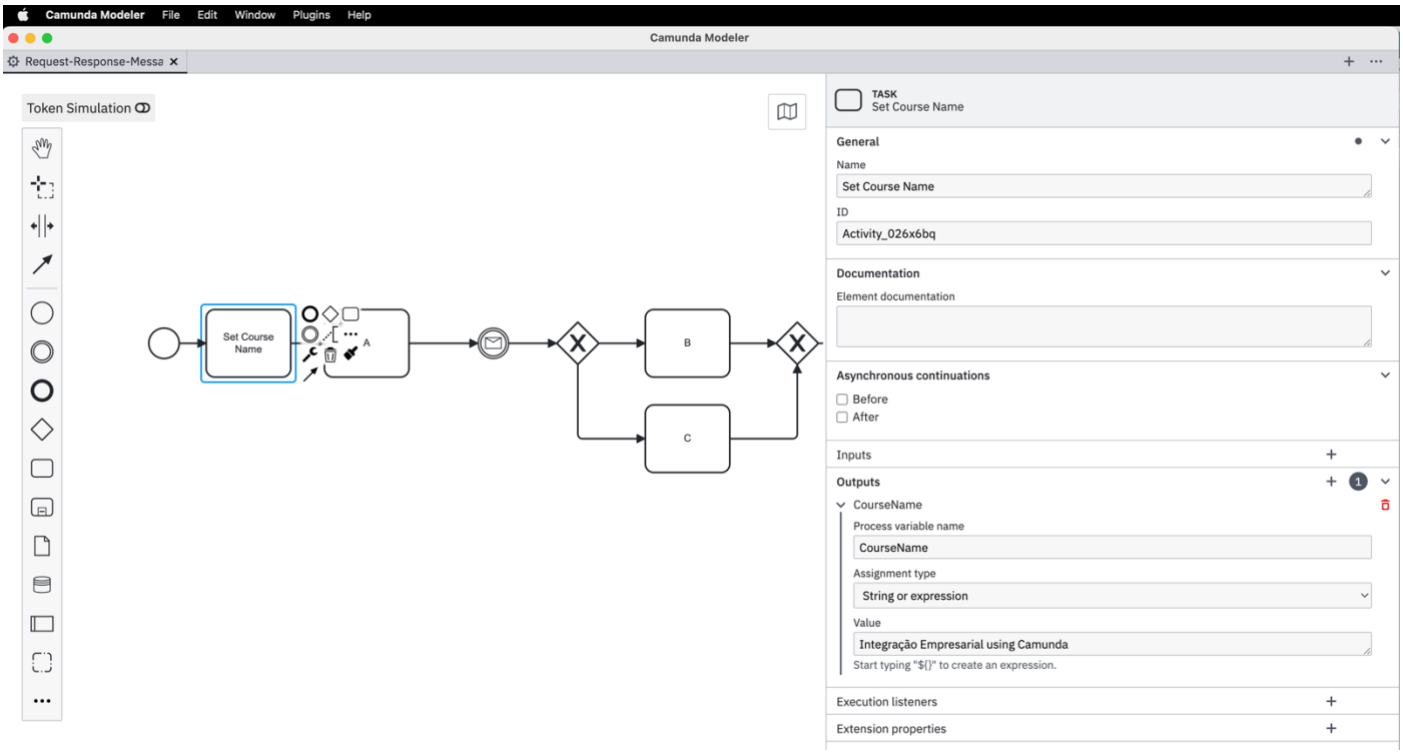

amunda Modeler File Edit Window Plugins Help
Camunda Modeler
SendMessageTask-WithPa
Token Simulation
Input Course Name
Call REST API
USER TASK
Input Course Name
General
Name
Input Course Name
ID
Activity_1chr38a
Documentation
Element documentation
User assignment
Assignee
demo
Candidate groups
Candidate users
Due date
The due date as an EL expression (e.g. ${someDate}) or an ISO date (e.g. 2015-06-26T09:54:00).
Follow up date
The follow up date as an EL expression (e.g. ${someDate}) or an ISO date (e.g. 2015-06-26T09:54:00).
Priority
Forms
Type
Generated Task Forms
Form fields
CourseName
ID
CourseName
Refers to the process variable name
Label
CourseName
Type
string
Default value
Constraints
Properties
Task listeners
Asynchronous continuations
Before
After
Inputs
Outputs
Execution listeners
XML
Camunda Platform 7.18
0 errors
Log
5.4.2


amunda Modeler File Edit Window Plugins Help
Camunda Modeler
SendMessageTask-WithPa
Token Simulation
Input Course Name
Call REST API
SERVICE TASK
Call REST API
General
Call REST API
ID
Activity_10sqvvz
Documentation
Element documentation
Implementation
Type
Connector
Connector ID
http-connector
Asynchronous continuations
Before
After
Inputs
Connector inputs
3
method
Local variable name
method
Assignment type
String or expression
Value
POST
Start typing "${}" to create an expression.
payload
Local variable name
payload
Assignment type
String or expression
Value
{
"name":"${CourseName}"
}
Start typing "${}" to create an expression.
url
Local variable name
url
Assignment type
String or expression
Value
https://gks3itxmr1.execute-api.us-east-1.amazonaws.com/test/helloworldpath
Start typing "${}" to create an expression.
Outputs
Connector outputs
Execution listeners
Extension properties
XML
Camunda Platform 7.18
0 errors
Log
5.4.2

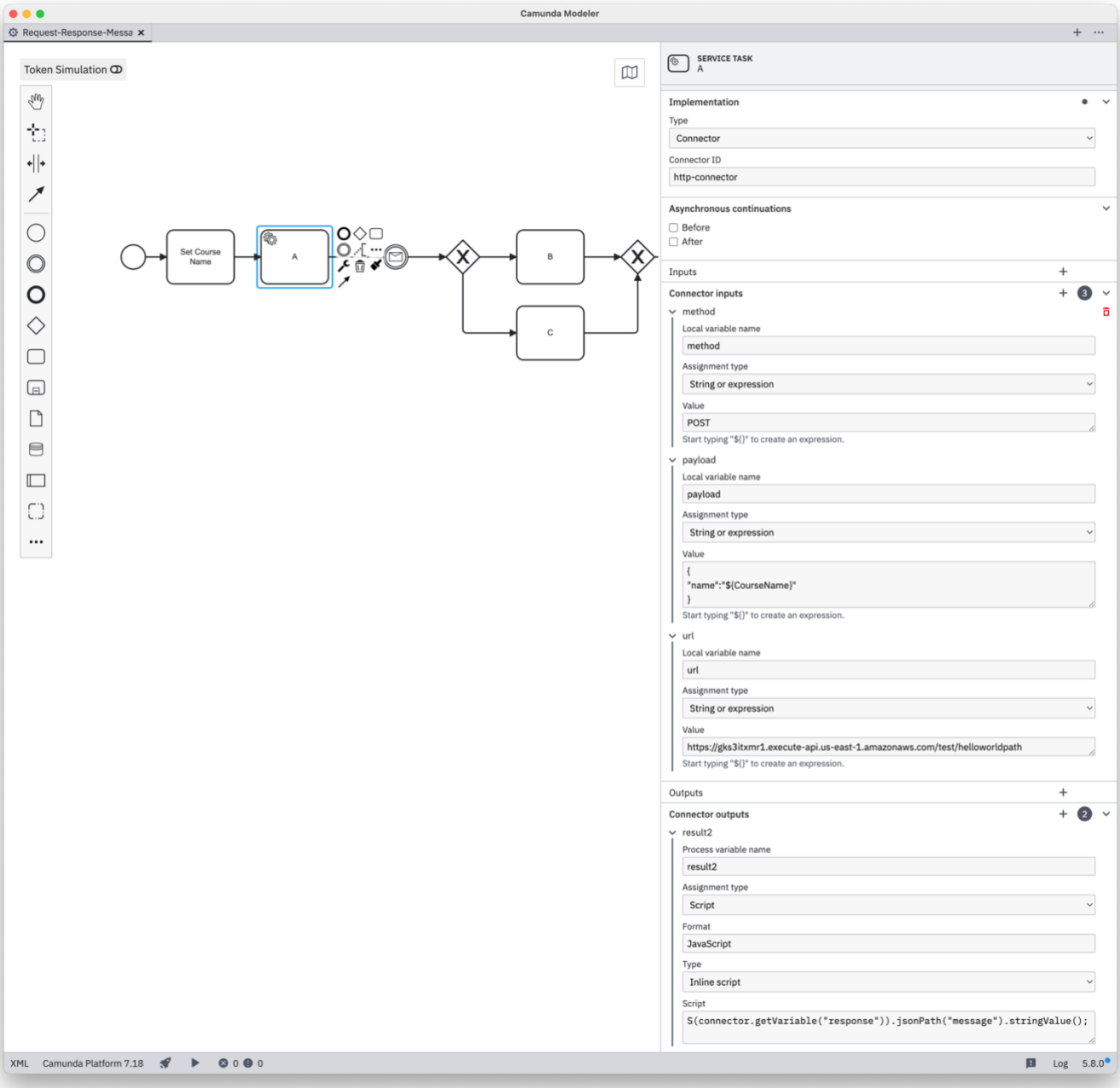
I. Integrate with a REST API installed remotely using Camunda http-connector – REQUEST & RESPONSE

For configuring the parsing of a REST API response, you may use inline *javascript* for parsing the received message, as depicted in the next figures. Many other JSON queries could be programmed, check reference manual at https://docs.camunda.org/manual/7.4/reference/spin/json/03-querying-json/.

Firstly, in this example, instead of receiving the course name from the user, we are adding the process variable "CourseName" as the output of first task, as depicted in the next figure.



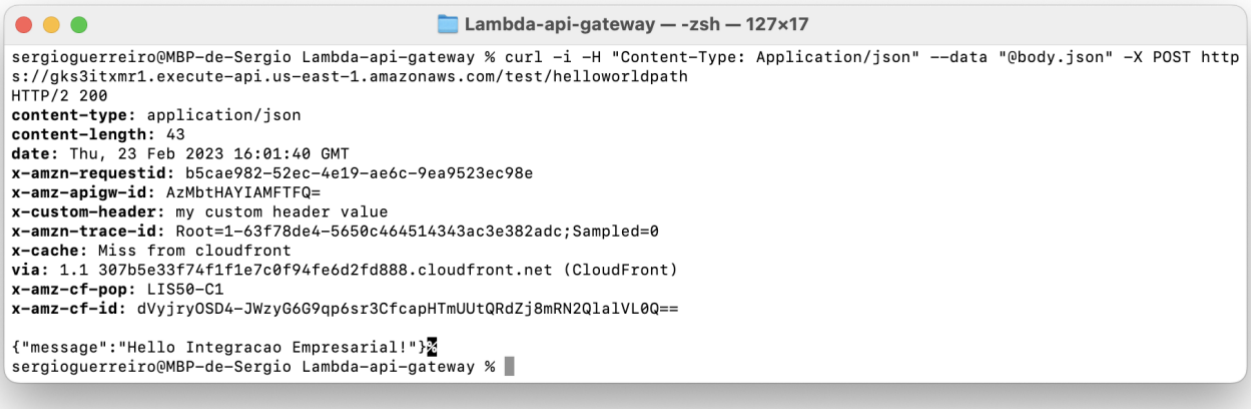
Secondly, the process variable "CourseName" is used in the payload of the connector inputs. Then, in the connector output you decode the fields that you like to receive from the response.



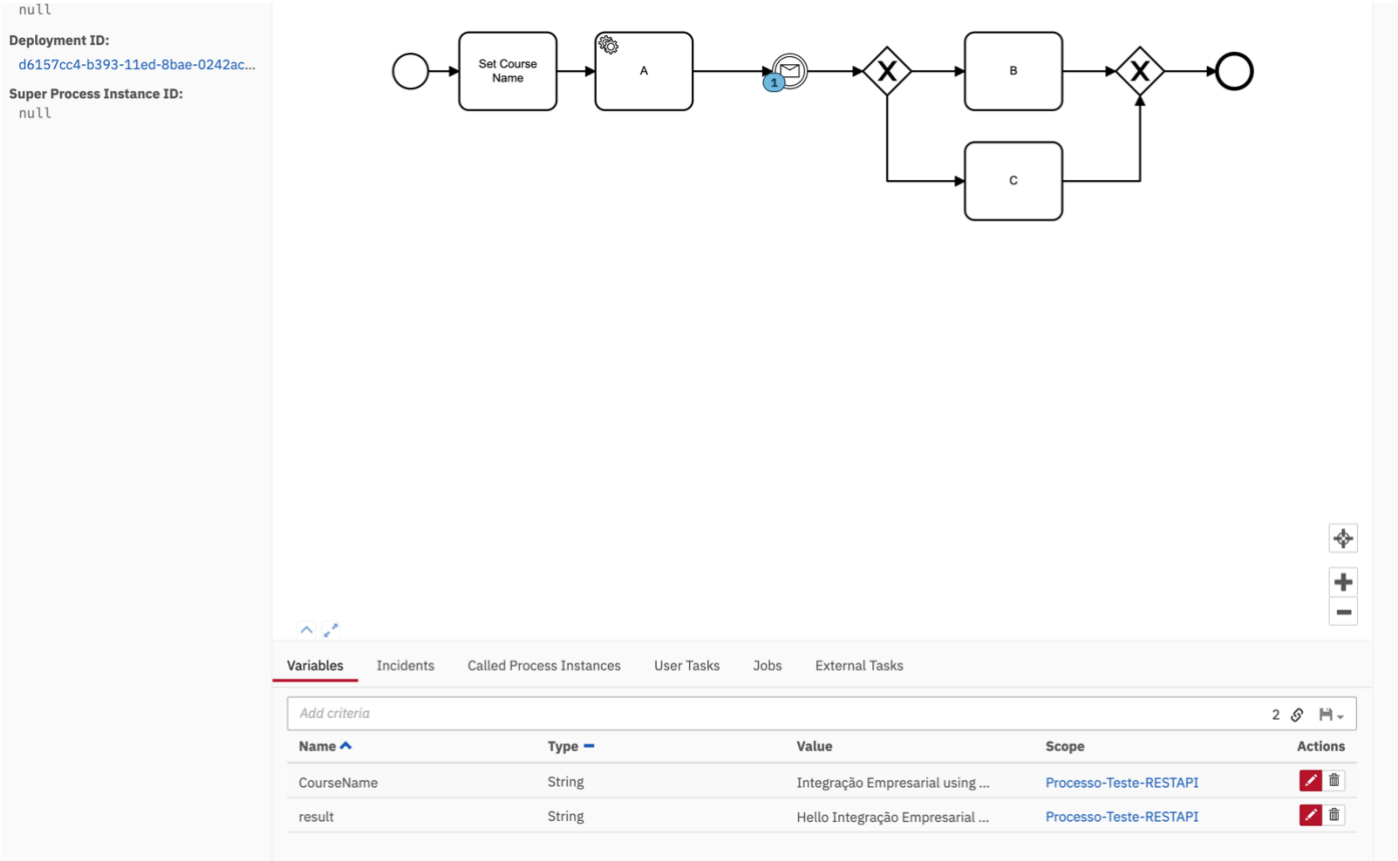
Notice that the inline script used in the connector output is reading "**message**" field from response and writing that to the **"result"** process variable:

```
S(connector.getVariable("response")).jsonPath("message").stringValue();
```

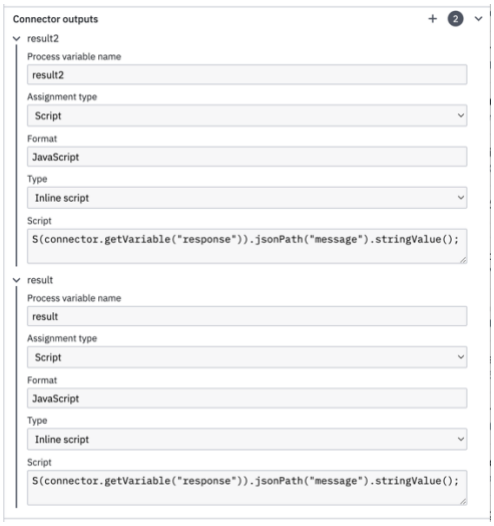
If you try to invoke your REST API using the curl command, you see the matching between the response and the name of the field "**message**".



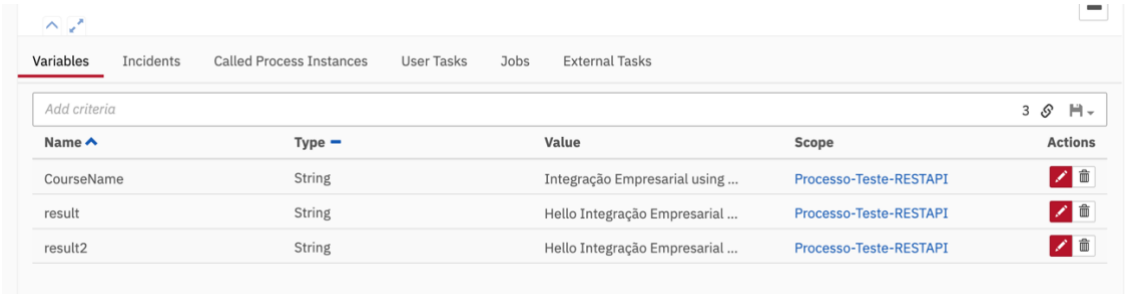
Moreover, in the Camunda cockpit, choose the process instance (an intermediate catch message event has been deliberately added to the process to hold it!), and verify that your instance now has the "result" process variable.



If you have many variables to retrieve from REST API response you only need to add more connector outputs with different process variable names, as depicted in the following figure.

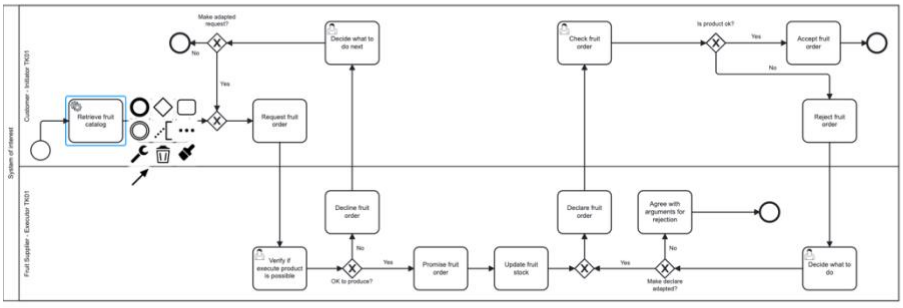


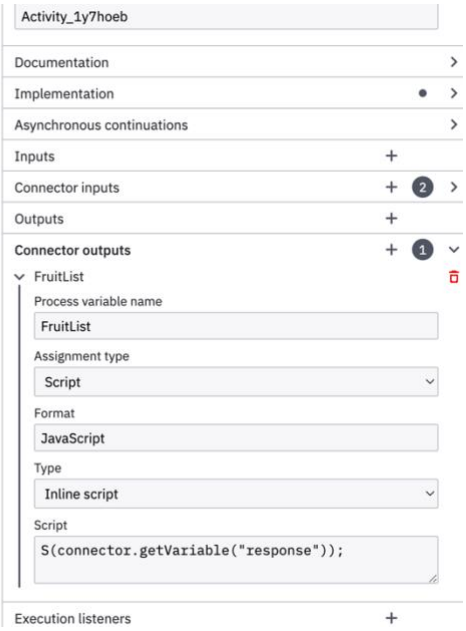
After that, you can check, in Camunda cockpit, that both variables are available.



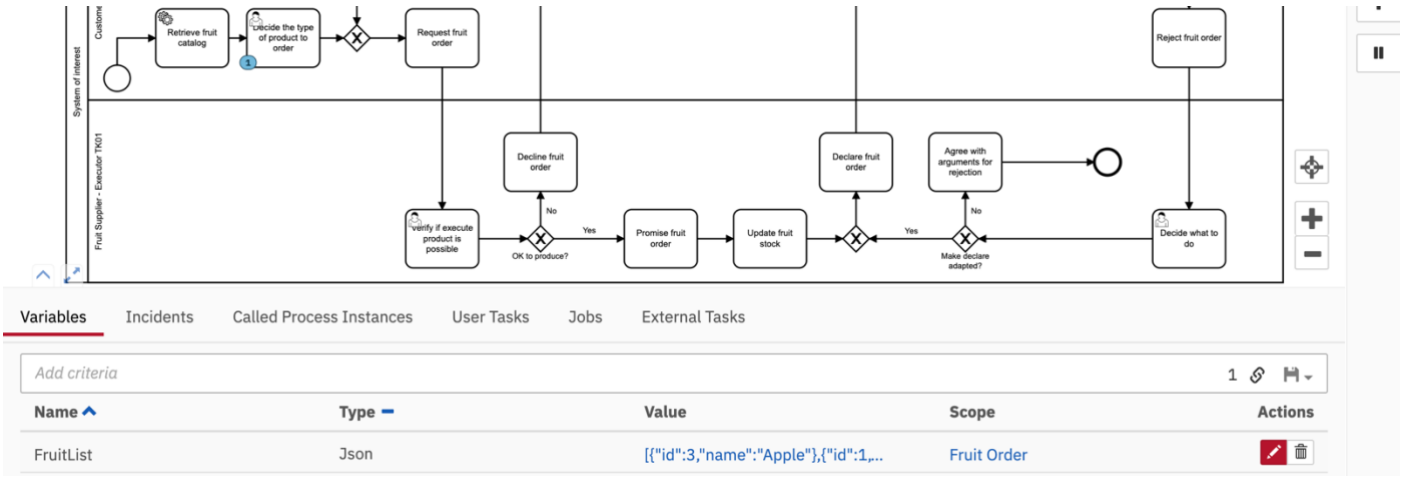
--*--

Hint: If you would like to debug a json message that has been received by a service task, you can use the following source code in the connector output:





And then, start an instance and check in the Camunda cockpit the message has been received correctly.

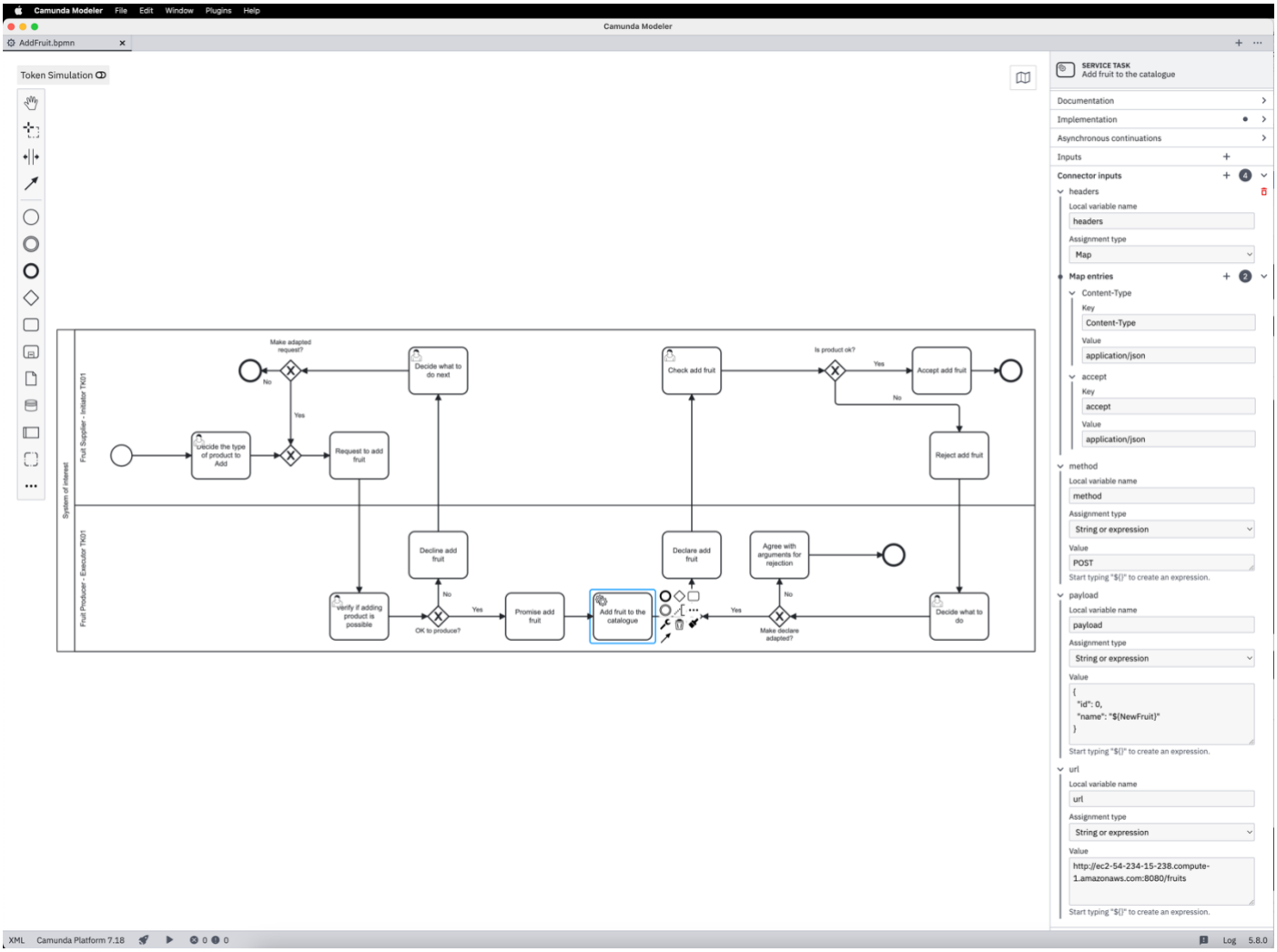


J. Integrate with a REST API installed remotely using Camunda http-connector – REQUEST & RESPONSE & Headers fields

When needed more information can be added to an API REST. For instance, if you need to specify the messages media, Headers can be included in the connector inputs. Considering the following POST request:

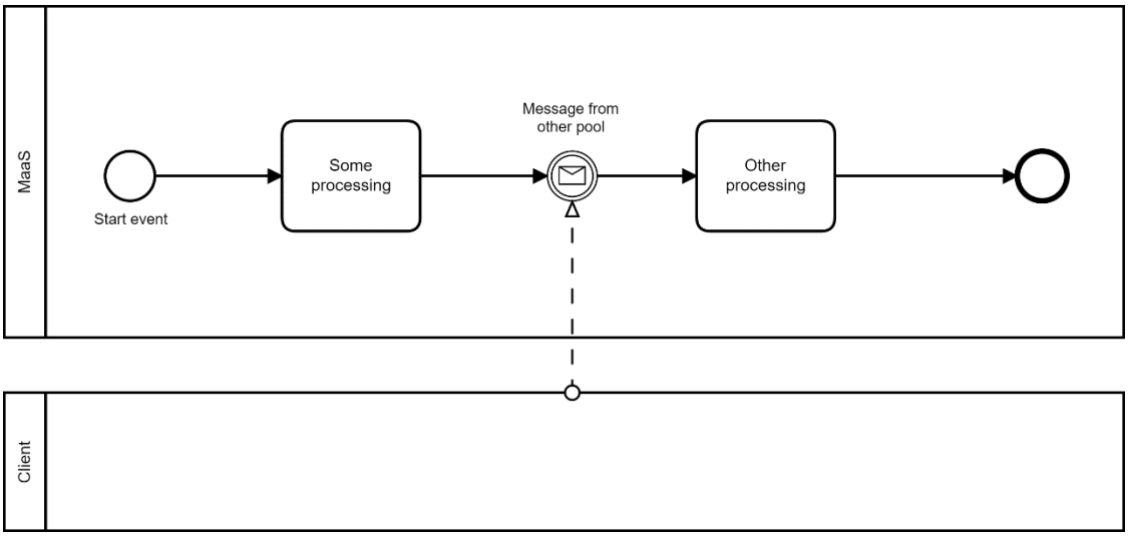
```
curl -X 'POST' \
  'http://ec2-54-234-15-238.compute-1.amazonaws.com:8080/fruits' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 0,
  "name": "sd"
}'
```

The same can be configured in a Camunda service task accordingly:

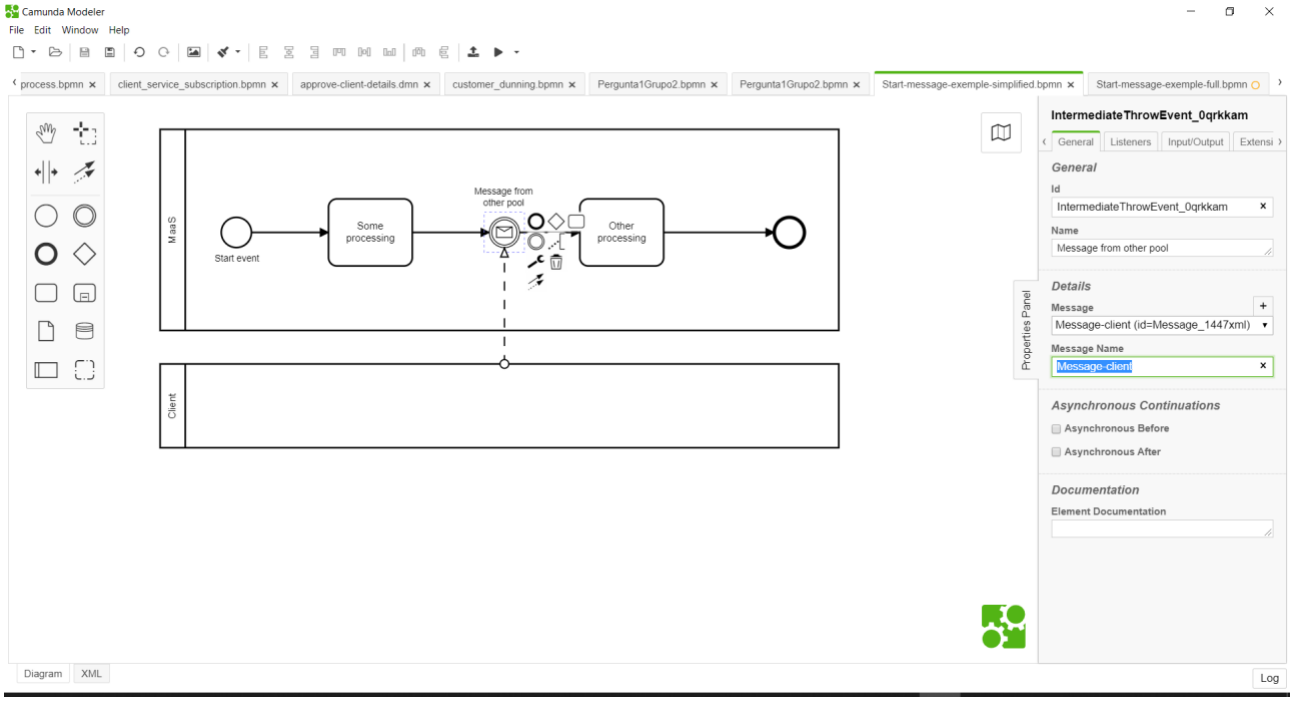


K. Using an intermediate message catch event inside a process.

L.1. Design the following collaboration in Camunda Modeller:

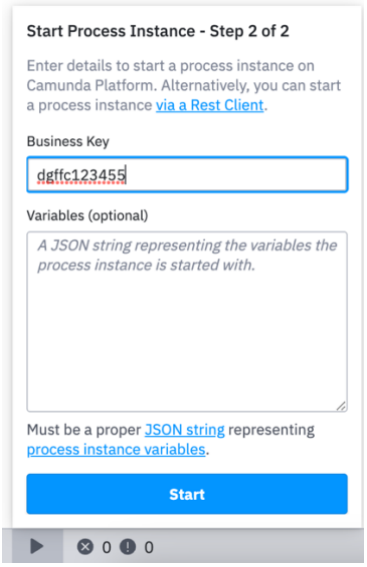


Where the intermediate message catch event should have a configuration similar with the following:

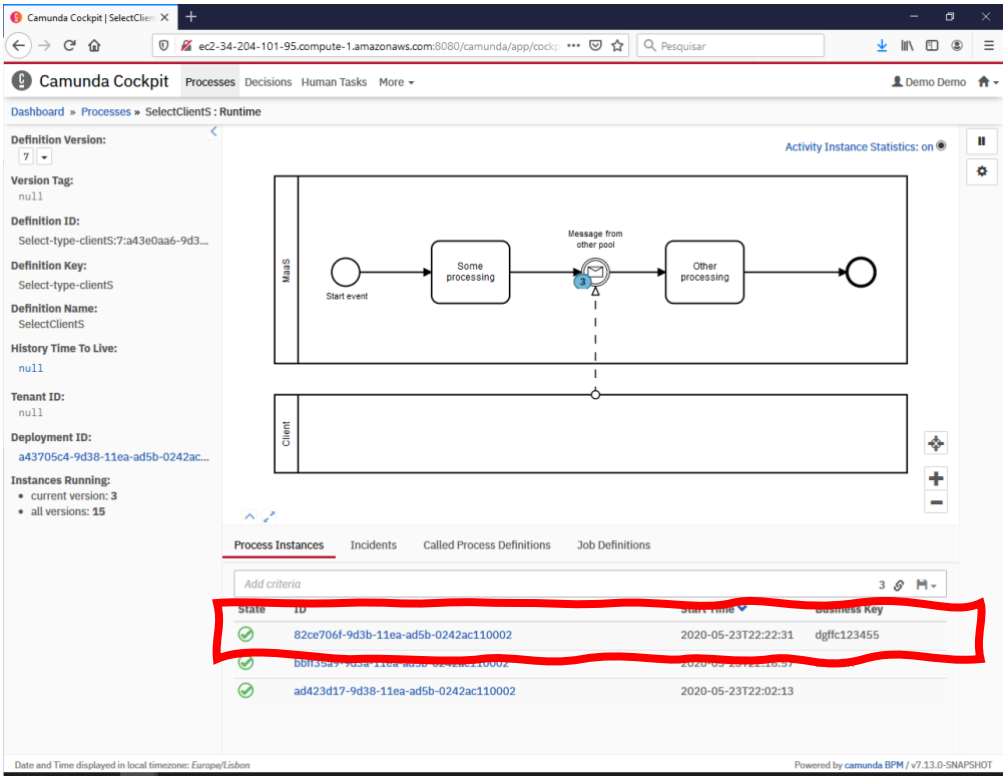


L.2. Deploy the model in your Camunda engine.

L.3. Start a new process instance with new configuration, where the **businessKey** is filled with any unique identifier.



You can verify the creation of the instance in the camunda cockpit. You will obtain something similar with the following business key as defined:



L.4. Then, the instance is expecting a message. To test the catch of a REST message, use a similar curl command:

```
curl -H "Content-Type: application/json" --data "@body.json" -X POST http://<YOUR CAMUNDA EC2 DNS>:8080/engine-rest/message
```

where bodyStart.json defines the message listener and the correlation with the key for your instance and sending some process variables:

```
{
  "messageName" : "Message-client",
  "businessKey" : "dgffc123455",
  "processVariables": {"amount": {"value":240,"type":"long"}},
  "resultEnabled" : true
}
```

References

Full official documentation:

- https://docs.camunda.org/manual/
- https://docs.camunda.org/javadoc/camunda-bpm-platform/7.12/

Testing the basics of the Camunda:

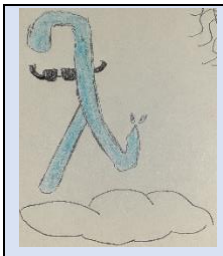
- https://docs.camunda.org/get-started/quick-start/
- Videos tutoriais: https://camunda.com/learn/videos/

URLs with other resources

- https://camunda.com/products/
- https://github.com/camunda/camunda-bpm-examples
- Send Task - https://docs.camunda.org/manual/7.7/reference/bpmn20/tasks/send-task/
- Receive task - https://docs.camunda.org/manual/7.7/reference/bpmn20/tasks/receive-task/
- https://github.com/camunda/camunda-bpm-examples/tree/master/servicetask/rest-service
- https://github.com/rob2universe/camunda-http-connector-example
- https://docs.camunda.org/manual/7.3/guides/user-guide/#process-engine-connectors
- **https://docs.camunda.org/manual/7.4/reference/spin/json/03-querying-json/**

DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration



The goal of this document is to present the details about installing, coding, deploying and testing a lambda service in Amazon AWS Lambda service.
With AWS Lambda, you run functions to process events. You can send events to your function by invoking it with the Lambda API, or by configuring an AWS service or resource to
Function: is a resource that you can invoke to run your code in AWS Lambda. A function has code that processes events, and a runtime that passes requests and responses between Lambda and the function code. You provide the code, and you can use the provided runtimes or create your own.
Runtime: Lambda runtimes allow functions in different languages to run in the same base execution environment. You configure your function to use a runtime that matches your programming language. The runtime sits in between the Lambda service and your function code, relaying invocation events, context information, and responses between the two. You can use runtimes provided by Lambda or build your own.
Event: is a JSON formatted document that contains data for a function to process. The Lambda runtime converts the event to an object and passes it to your function code. When you invoke a function, you determine the structure and contents of the event. When an AWS service invokes your function, the service defines the event.
Concurrency: is the number of requests that your function is serving at any given time. When your function is invoked, Lambda provisions an instance of it to process the event. When the function code finishes running, it can handle another request. If the function is invoked again while a request is still being processed, another instance is provisioned, increasing the function's concurrency.
Trigger: is a resource or configuration that invokes a Lambda function. This includes AWS services that can be configured to invoke a function, applications that you develop, and event source mappings. An event source mapping is a resource in Lambda that reads items from a stream or queue and invokes a function.

The following contents is presented in this document.

Contents

A. Creating a Lambda function using VSCode IDE (Java)

(Partially based on AWS Lambda Developer Guide, 2025 Amazon Web Services, Inc., public available at https://docs.aws.amazon.com/lambda/ *)*

In this step, you start VSCode and create a Maven project. You will add the necessary dependencies and build the project. The build will produce a .jar, which is your deployment package.

A.0. Add a new **Java Project…**
A.1. Select project Type: **Maven**
A.2. Select **No Archetype**
A.3. Write **examples** as group Id
A.4. Write **lambda-java-example** as artifact Id
A.5. **Select your destination folder**
A.6. Add Java class to the project.

a. Open the context (right-click) menu for the src/main/java subdirectory in the project, choose **New**, and then choose **New File…**.add the name **Hello.java**

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;

import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Map;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.*;

public class Hello implements RequestStreamHandler{
public void handleRequest(
                        InputStream inputStream,
                        OutputStream outputStream,
                        Context context)
  {
        LambdaLogger logger = context.getLogger();
        try
        {
            JSONParser parser = new JSONParser();
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
            JSONObject responseJson = new JSONObject();
            JSONObject event = (JSONObject) parser.parse(reader);
            String name = new String();

            logger.log("input:" + (String) event.toString()+"\n");

            if (event.get("body") != null)
            {
                JSONObject bodyjson = (JSONObject) parser.parse((String) event.get("body"));
                if (bodyjson.get("name") != null)
                        name = (String) bodyjson.get("name");
            }

            JSONObject responseBody = new JSONObject();
            if (name != null)
                responseBody.put("message","Hello " + name + "!");
            else
                responseBody.put("message", "No name defined. Add {\"name\":\"name\"} in the request.");

           JSONObject headerJson = new JSONObject();
           headerJson.put("x-custom-header", "my custom header value");
           responseJson.put("statusCode", 200);
           responseJson.put("headers", headerJson);
           responseJson.put("body", responseBody.toString());
```

```
            OutputStreamWriter writer = new OutputStreamWriter(outputStream, "UTF-8");
            writer.write(responseJson.toString());
            writer.close();
        }
        catch(Exception exc)
        {
                logger.log("Error : " + exc.toString());
        }
  }
}
```

A.7. Verify that your **pom.xml** file contains the following configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>examples</groupId>
    <artifactId>lambda-java-example</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>lambda-java-example</name>

    <properties>
            <maven.compiler.source>11</maven.compiler.source>
            <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-lambda-java-core</artifactId>
            <version>1.2.2</version>
        </dependency>
       <dependency>
            <groupId>com.googlecode.json-simple</groupId>
            <artifactId>json-simple</artifactId>
            <version>1.1.1</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.4.1</version>
            </plugin>
        </plugins>
    </build>
</project>
```

A.8. Configure **JDK11** to be available on your project root command line with Apache Maven 3.9.9.

A.9. Execute the packaging procedure with

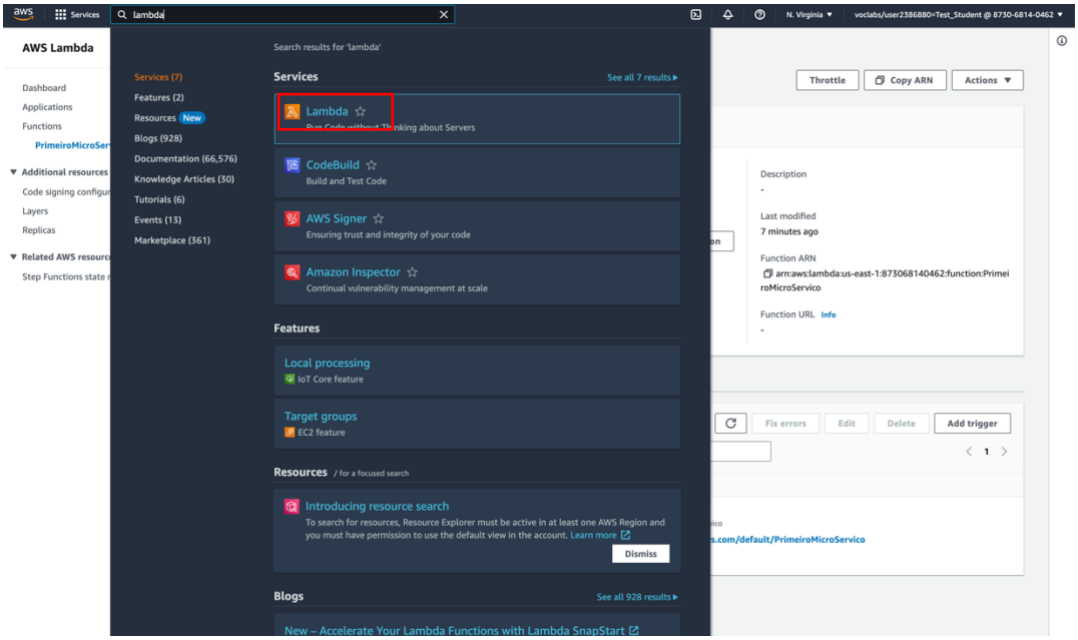
```
mvn package shade:shade
```

A.10. You will the following file in the project target folder:

```
lambda-java-example-0.0.1-SNAPSHOT.jar.
```

A.11. **Login** to AWS Educate, then choose the classroom, and press AWS console. Expand “All services” option.

A.12. Search for the AWS Lambda Service(under Compute option)



A.13. Press Create function

A.14. Choose "Author from scratch"

A.15. Define Function name as: PrimeiroMicroServico

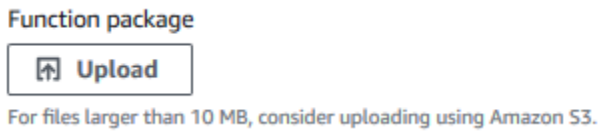
A.16. Choose JAVA 11 for Runtime

A.17. Choose x86_64 Architecture

A.18. Change Default execution role > choose Use an existing role> choose LabRole

A.19. Press Create function

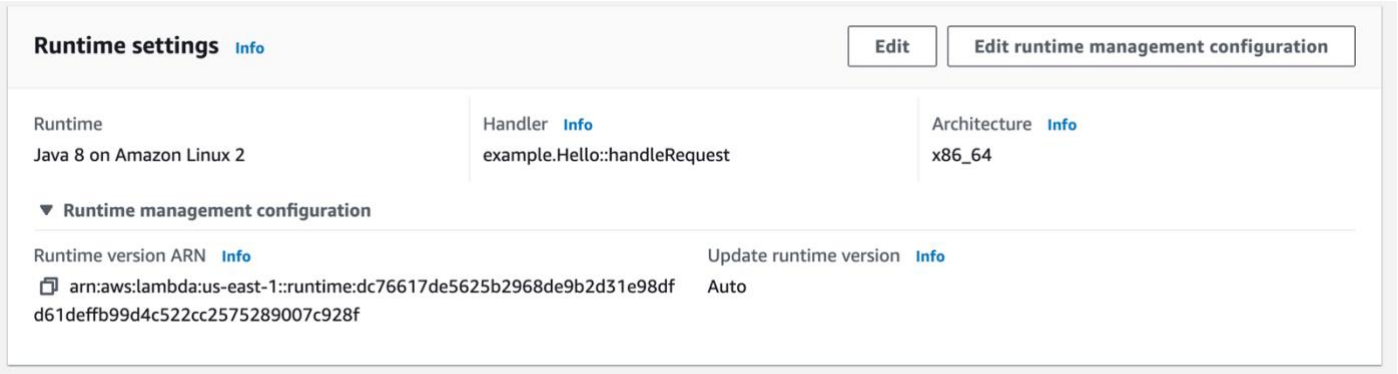
A.20. Then, in the next interface, press the Upload button:



and choose the previously created jar file:

```
lambda-java-example-0.0.1-SNAPSHOT.jar.
```

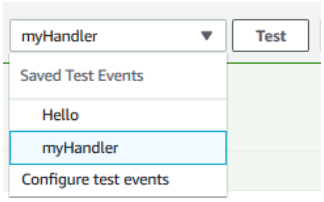
A.21. Check the Handler field with the name of the handler defined within your java class. You will obtain something similar with this:



A.22. Your lambda service is deployed!

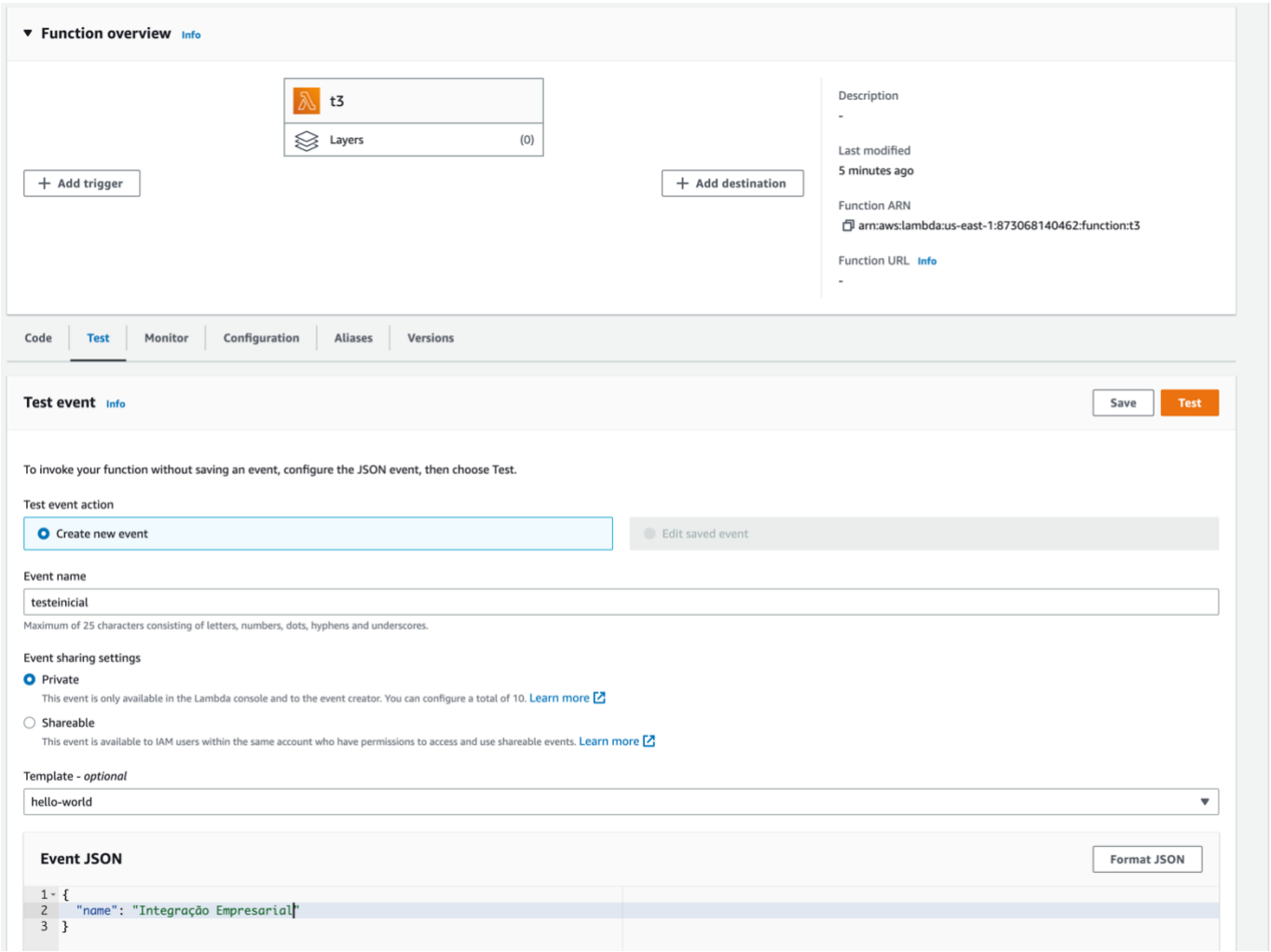
B. Testing a Lambda function internally

C.1. Choose option "Test" and select create new event



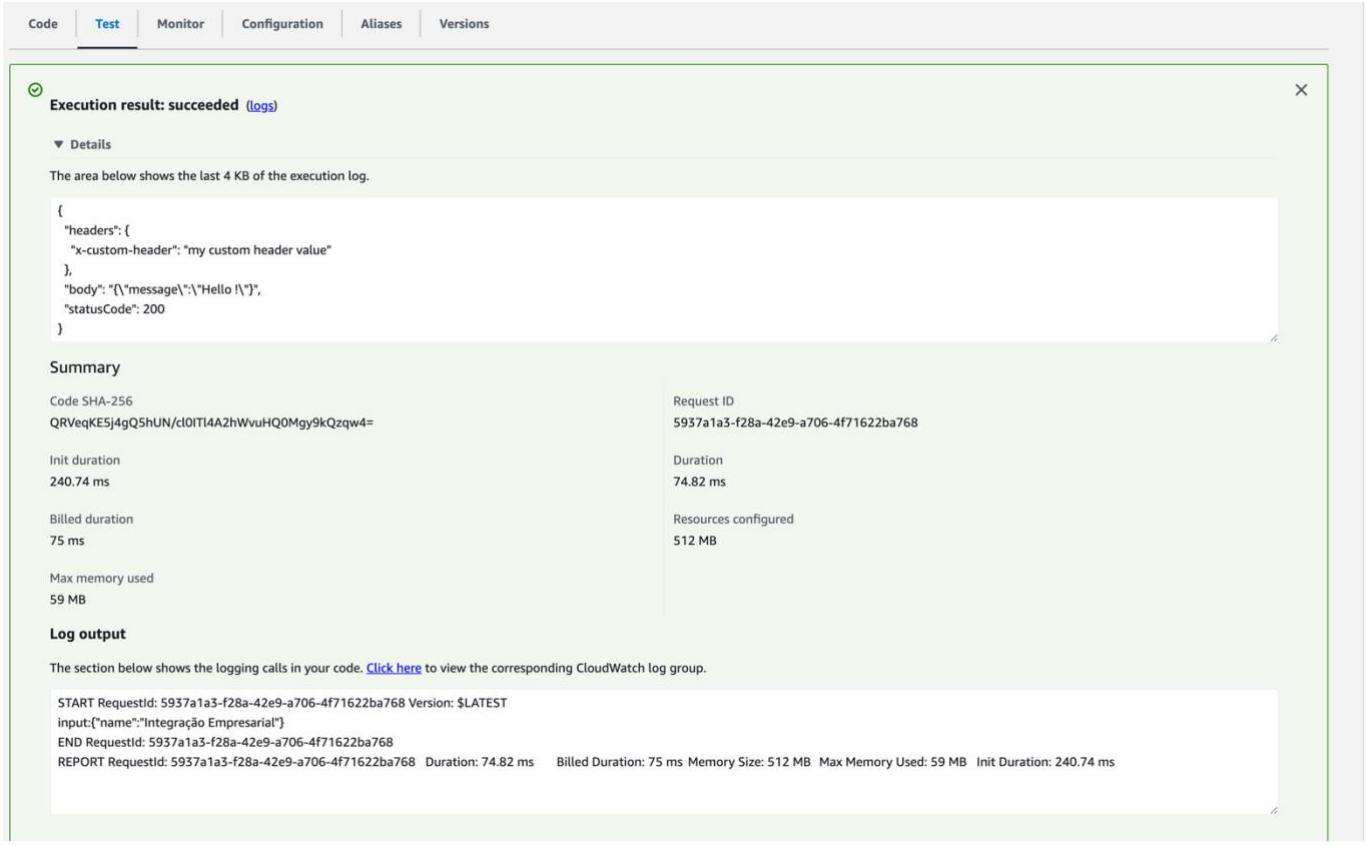
C.2. Define a name for "Event name" field

C.3. Define the JSON message accordingly with the test example, with the example of section A, define the message as follows:



C.4. Press Test button

C.5. You should receive a response similar with:


Code
Test
Monitor
Configuration
Aliases
Versions
Execution result: succeeded (logs)
Details
The area below shows the last 4 KB of the execution log.
{
"headers": {
"x-custom-header": "my custom header value"
},
"body": "{\"message\":\"Hello !\"}",
"statusCode": 200
}
Summary
Code SHA-256
QRVeqKE5j4gQ5hUN/cl0lTl4A2hWvuHQ0Mgy9kQzqw4=
Request ID
5937a1a3-f28a-42e9-a706-4f71622ba768
Init duration
240.74 ms
Duration
74.82 ms
Billed duration
75 ms
Resources configured
512 MB
Max memory used
59 MB
Log output
The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.
START RequestId: 5937a1a3-f28a-42e9-a706-4f71622ba768 Version: $LATEST
input:{"name":"Integração Empresarial"}
END RequestId: 5937a1a3-f28a-42e9-a706-4f71622ba768
REPORT RequestId: 5937a1a3-f28a-42e9-a706-4f71622ba768 Duration: 74.82 ms Billed Duration: 75 ms Memory Size: 512 MB Max Memory Used: 59 MB Init Duration: 240.74 ms

C. Invoking a Lambda function remotely

To allow a Lambda function to be invoked remotely the API Gateway need to be assigned to the trigger of the function

D.1. Access your Lambda function in AWS console as described in the previous sections

D.2. Press the Add trigger button: 

D.3. In the trigger confirmation drop-down menu choose "API Gateway"

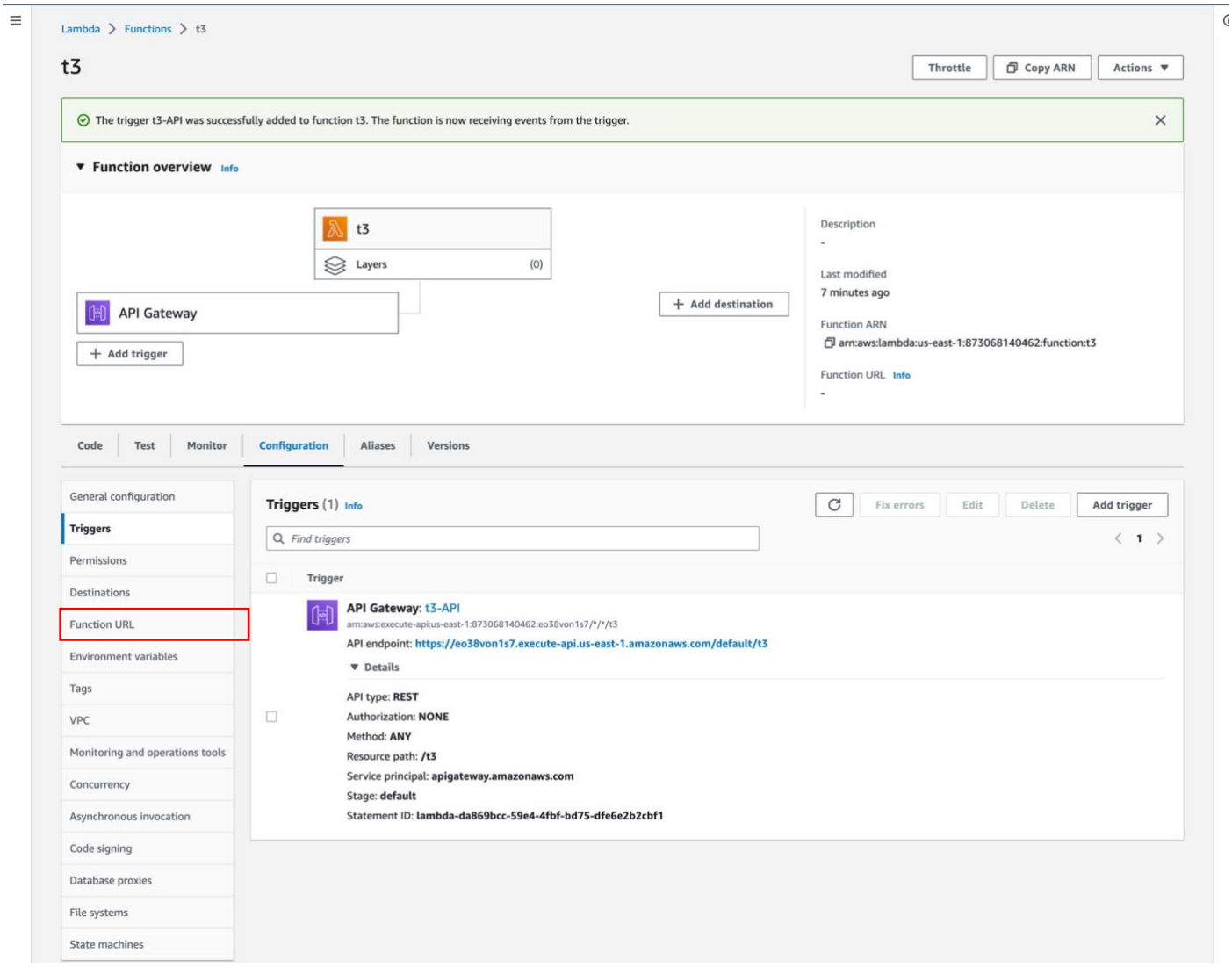
D.4. Then, in the API drop-down menu choose "Create a new API"

D.5. Choose REST API option

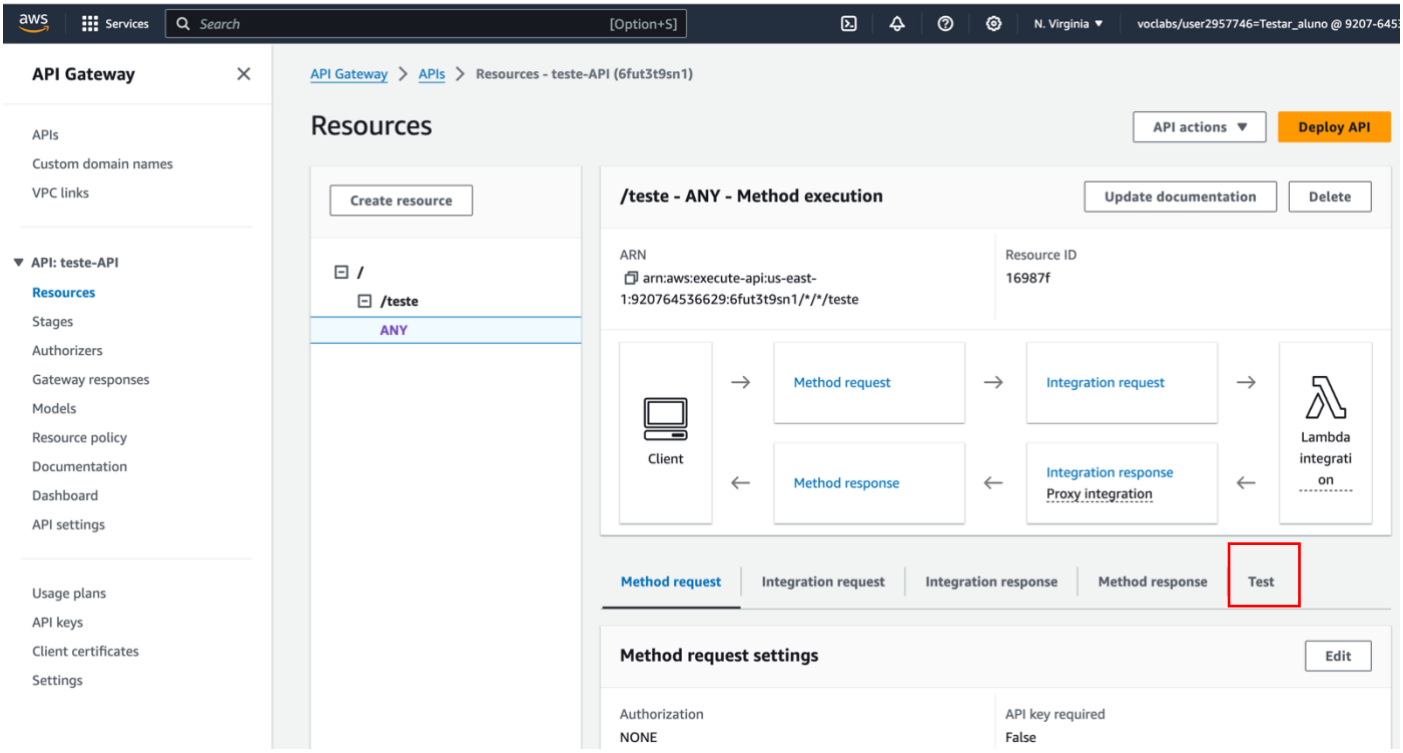
D.6. Choose the following option from Security drop down box: Open

D.7. Press Add button 

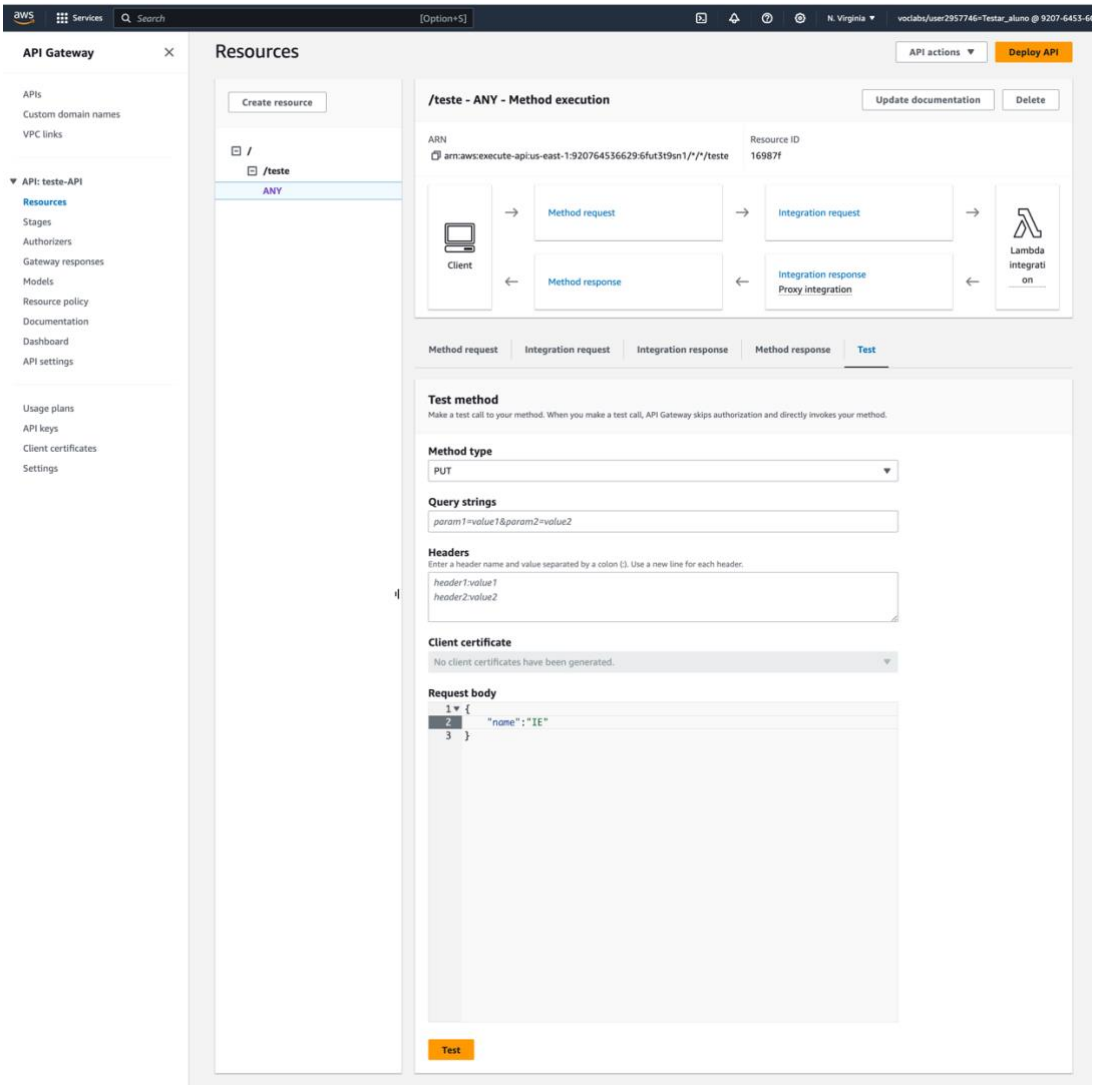
D.8. You will obtain something similar with the following image:



Click in the "t3-API" link, or similar, as identified in the red box of the image, to select the AWS API Gateway service of your lambdaservice. A similar interface is obtained:



D.9. Select the TEST option as identified in the red box of the previous image and obtain the following:



D.10. Alternatively, create the file body.json with the following contents:

```
{
  "name": "Integracao Empresarial"
}
```

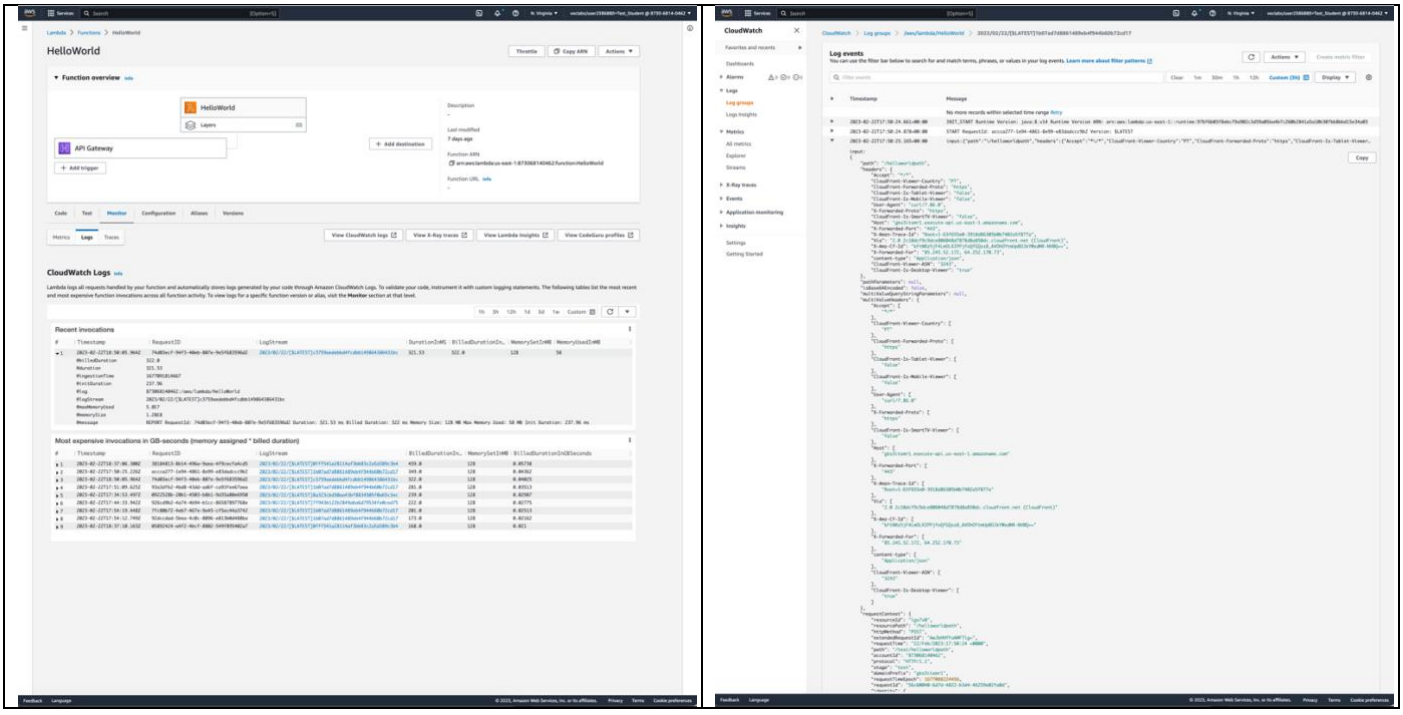
And execute the following command in another ssh or putty session (in linux like environment) where the is available as in step C.8. (e.g.: https://uwajykvb2j.execute-api.us-east-1.amazonaws.com/default/PrimeiroMicroServico):

```
curl -i -H "Content-Type: Application/json" --data "@body.json" -X POST <LAMBDA_URL>
```

It is expected that you receive something similar with:

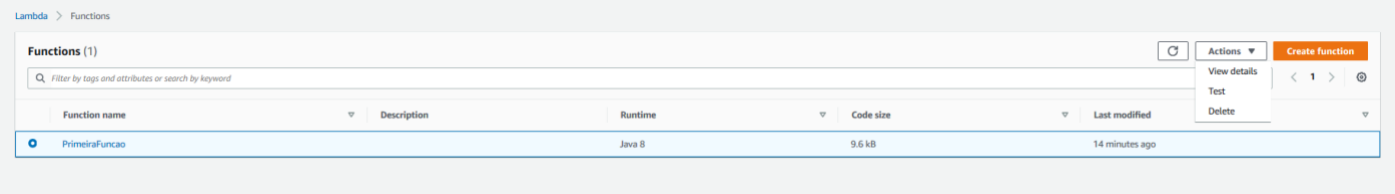
```
$ curl -i -H "Content-Type: Application/json" --data "@body.json" -X POST https://2p3fp5acxj.execute-api.us-
east-1.amazonaws.com/default/PrimeiroMicroServico

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    79  100    43  100    36     43     36  0:00:01 --:--:--  0:00:01   117
HTTP/1.1 200 OK
Date: Tue, 07 Jan 2020 22:26:07 GMT
Content-Type: application/json
Content-Length: 43
Connection: keep-alive
x-amzn-RequestId: a072be98-cefe-4847-8558-2360b6106e30
x-amz-apigw-id: F83L8FjKIAMFzrQ=
x-custom-header: my custom header value
X-Amzn-Trace-Id: Root=1-5e15057f-3a08bd2b9dd68b3d22208a89;Sampled=0

{"message":"Hello Integracao Empresarial!"}
```

The trace of the request can be verified in the "Monitor" tab of the Lambda function. Choosing the desired LogStream (left part of the following figure) will bring you to the right part of the figure with all the details of the invocation (inside AWS CloudWatch).



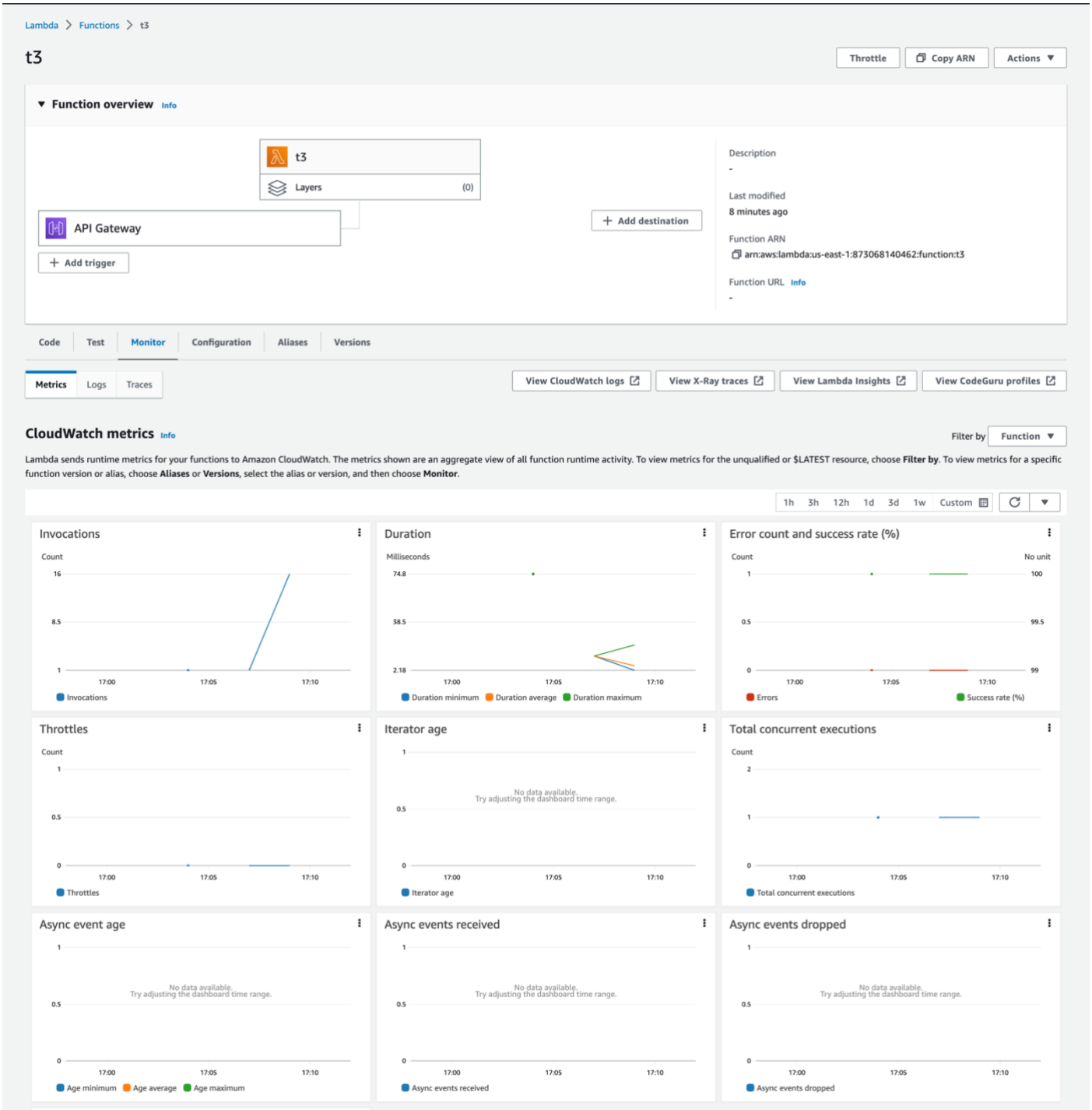
D. Delete a Lambda function

E.1. Choose the Lambda > Functions options, then choose the function to delete and then chose Delete from Action drop-down menu.



E. Checking consumption of a Lambda function

F.1. Choose the Lambda > Dashboard to view all the detail regarding each of the invocations to each of the lambda service. For a specific Function, use the option Monitoring available in all Functions.



Other references

- AWS Lambda Concepts https://docs.aws.amazon.com/lambda/latest/dg/welcome.html, available in PDF.
- Amazon API Gateway Developer Guide https://docs.aws.amazon.com/pt_br/apigateway/latest/developerguide/welcome.html, available in PDF.
- Set up an Integration Response in API Gateway, https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-integration-settings-integration-response.html
- https://www.mkyong.com/java/json-simple-example-read-and-write-json/
- https://www.tutorialspoint.com/json_simple/json_simple_quick_guide.htm
- https://www.baeldung.com/aws-lambda-api-gateway
- https://github.com/eugenp/tutorials/blob/master/aws-lambda/src/main/java/com/baeldung/lambda/apigateway/APIDemoHandler.java
- Amazon Relational Database Service User Guide - https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/rds-ug.pdf#CHAP_GettingStarted

DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Tutorial – Installing and Operating KONG Open Source

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration

The goal of this document is to demonstrate how to install and operate the Kong open source service. Kong is a platform aiming the integration between all the micro services (λ and JAVA), and the BPMN processes developed in Camunda. Moreover, a Kong management platform, named Konga, is introduced.

The following contents is presented in this document.

Contents

A. Installing and starting the Kong Open Source Service as a docker in EC2

A.1. Create an EC2 new instance of the type: Amazon Linux 2 AMI (HVM), SSD Volume Type. The exact versions may change with time. And define the inbound rules to allow the 8080, 8002 and 22 accessed from anywhere.

A.2. Access the new EC2 instance and execute the following commands:

```
sudo yum install -y docker

sudo service docker start

sudo docker network create kong-net
```

```
sudo docker run -d --name kong-database \
  --network=kong-net \
  -p 5432:5432 \
  -e "POSTGRES_USER=kong" \
  -e "POSTGRES_DB=kong" \
  -e "POSTGRES_PASSWORD=kongpass" \
  postgres:13
```

```
sudo docker run --rm --network=kong-net \
-e "KONG_DATABASE=postgres" \
-e "KONG_PG_HOST=kong-database" \
-e "KONG_PG_PASSWORD=kongpass" \
-e "KONG_PASSWORD=test" \
kong/kong-gateway:3.9.0.0 kong migrations bootstrap
```

```
sudo docker run -d --name kong-gateway \
--network=kong-net \
-e "KONG_DATABASE=postgres" \
-e "KONG_PG_HOST=kong-database" \
-e "KONG_PG_USER=kong" \
-e "KONG_PG_PASSWORD=kongpass" \
-e "KONG_PROXY_ACCESS_LOG=/dev/stdout" \
-e "KONG_ADMIN_ACCESS_LOG=/dev/stdout" \
-e "KONG_PROXY_ERROR_LOG=/dev/stderr" \
-e "KONG_ADMIN_ERROR_LOG=/dev/stderr" \
-e "KONG_ADMIN_LISTEN=0.0.0.0:8001, 0.0.0.0:8444 ssl" \
-e "KONG_ADMIN_GUI_URL=http://localhost:8002" \
-e KONG_LICENSE_DATA \
-p 8000:8000 \
-p 8443:8443 \
-p 8001:8001 \
-p 8002:8002 \
-p 8445:8445 \
-p 8003:8003 \
-p 8004:8004 \
-p 127.0.0.1:8444:8444 \
kong/kong-gateway:3.9.0.0
```

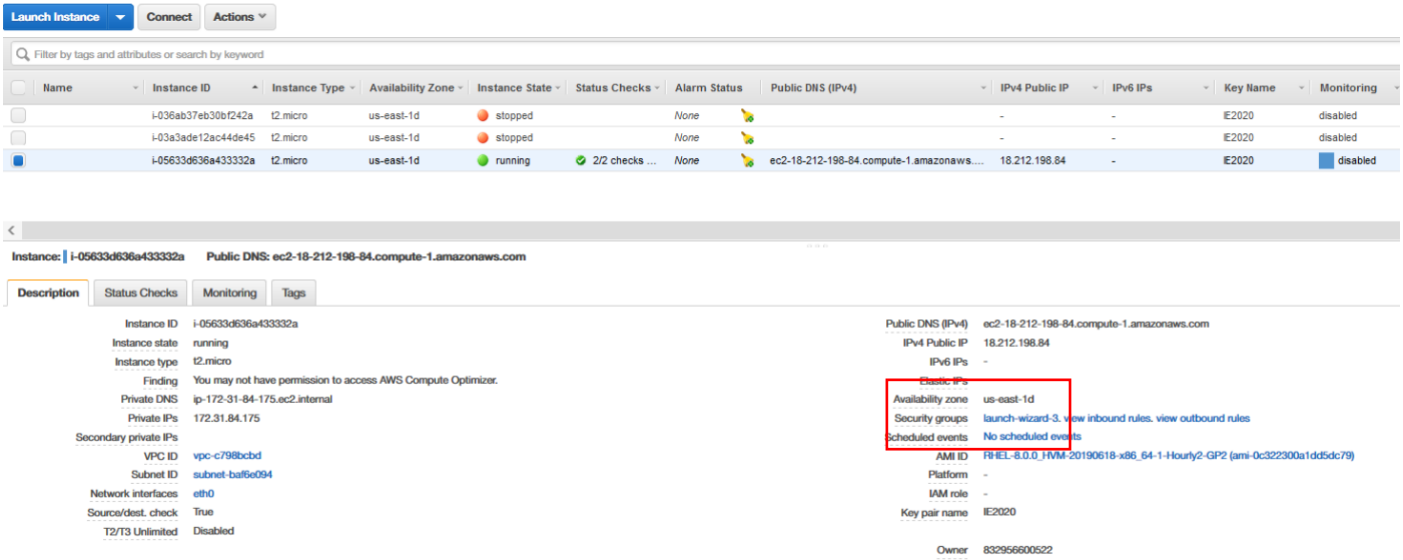
A.3. Verify your installation in EC2 instance:

```
curl -i -X GET --url http://localhost:8001/services
```

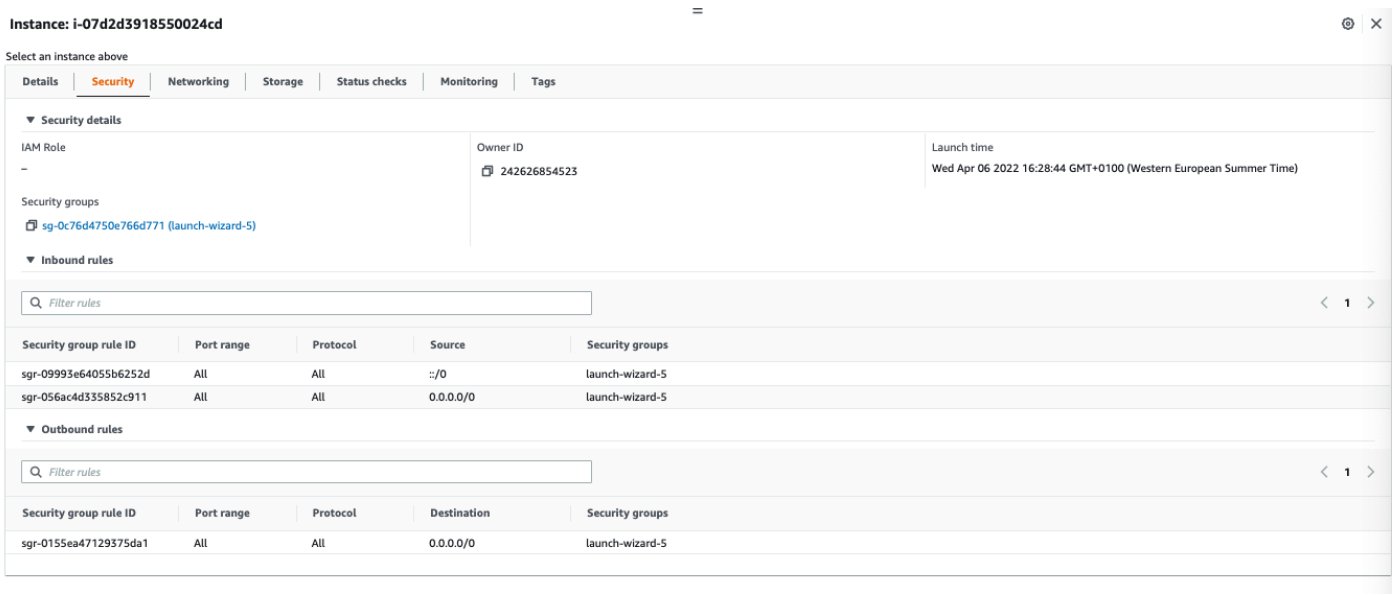
You should receive a 200 status code.

B. Configuring Kong to be used remotely

B.1. Choose you EC2 instance. Click on security-group (for example):



And change the configuration to allow all the inbound traffic:



B.2. Test with the following command from your local computer:

```
curl -i http://<YOURIP>:8000/
```

You should receive a similar response with the following:

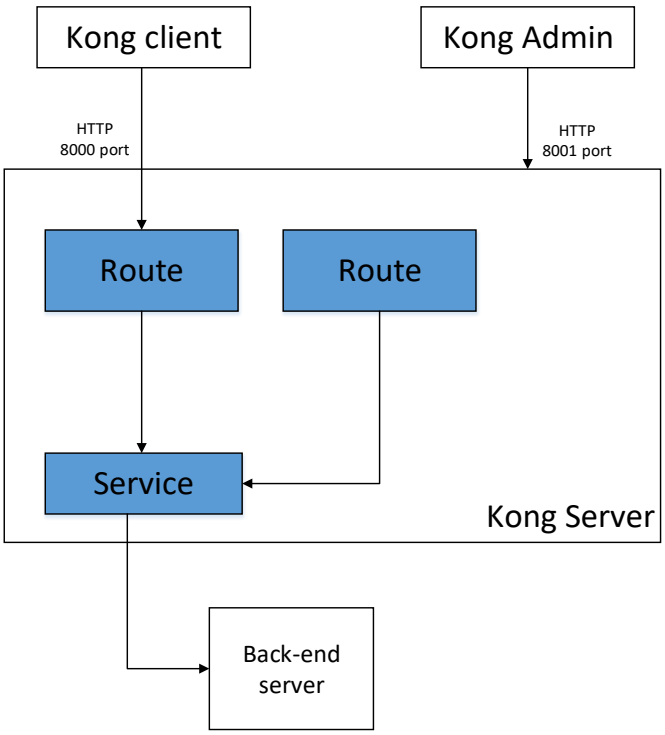
```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    48  100    48    0     0     48      0  0:00:01 --:--:--  0:00:01   192
HTTP/1.1 404 Not Found
Date: Tue, 14 Jan 2020 15:34:55 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Content-Length: 48
X-Kong-Response-Latency: 0
Server: kong/2.0.0rc2

{"message":"no Route matched with those values"}
```

C. Creating a service and a route in Kong

From Kong definitions:

- a Service; that is the name Kong uses to refer to the upstream APIs and microservices it manages.
- and a Route specify how (and if) requests are sent to their Services after they reach Kong. A single Service can have many Routes.

In Kong the ports are organized as follows: Port 8001 – is used for managing APIs and Port 8000 – for service invocation. This is depicted by the following figure:



To try-out use the following commands:

C.1. In your Kong AWS EC2 instance (for security reasons is safer to do it locally) issue the following cURL request to add your first Service (pointing to the Mockbin API) to Kong:

```
curl -i -X POST \
  --url http://localhost:8001/services/ \
  --data 'name=example-service' \
  --data 'url=http://mockbin.org'
```

You should receive a response similar to:

```
HTTP/1.1 201 Created
Content-Type: application/json
Connection: keep-alive

{
   "host":"mockbin.org",
   "created_at":1519130509,
   "connect_timeout":60000,
   "id":"92956672-f5ea-4e9a-b096-667bf55bc40c",
   "protocol":"http",
   "name":"example-service",
   "read_timeout":60000,
   "port":80,
   "path":null,
   "updated_at":1519130509,
   "retries":5,
   "write_timeout":60000
}
```

C.2. Then, add a Route for the previous Service:

```
 curl -i -X POST \
  --url http://localhost:8001/services/example-service/routes \
  --data 'hosts[]=example.com'
```

The answer should be similar to:

```
HTTP/1.1 201 Created
Content-Type: application/json
Connection: keep-alive

{
   "created at":1519131139,
   "strip path":true,
   "hosts":[
      "example.com"
   ],
   "preserve_host":false,
   "regex_priority":0,
   "updated_at":1519131139,
   "paths":null,
   "service":{
      "id":"79d7ee6e-9fc7-4b95-aa3b-61d2e17e7516"
   },
   "methods":null,
   "protocols":[
      "http",
      "https"
   ],
   "id":"f9ce2ed7-c06e-4e16-bd5d-3a82daef3f9d"
}
```

The service is now ready to be invoked. Continue to the next section of this document.

D. Invoking a service

Issue the following cURL request to verify that Kong is properly forwarding requests to your Service. Note that by default Kong handles proxy requests on port :8000. The following request indicates which is the route that you would like to call:

```
$ curl -i -X GET \
  --url http://localhost:8000/ \
  --header 'Host: example.com'
```

A successful response means Kong is now forwarding requests made to http://localhost:8000 to the url we configured in previous section and is forwarding the response back to us. Kong knows to do this through the header defined in the above curl request:

Host: <given host>

hint: If you are testing the invocation remotely from the AWS EC2 Kong instance, then you should replace the localhost by the AWS EC2 DNS name.

E. Connecting to a remote microservice

E.1. DIRECTLY TO LAMBDA BACKEND (as previous introduced in correspondingly tutorial):

```
curl -i -H "Content-Type: Application/json"\
 --data "@body.json"\
 -X POST https://2p3fp5acxj.execute-api.us-east-1.amazonaws.com/default/PrimeiroMicroServico
```

E.2. CREATE SERVICE (in the AWS EC2 Kong instance):

```
 curl -i -X POST \
  --url http://localhost:8001/services/ \
  --data 'name=invoke-lambda-service' \
  --data 'url=https://2p3fp5acxj.execute-api.us-east-1.amazonaws.com/default/PrimeiroMicroServico'\
```

E.3. CREATE ROUTE (in the AWS EC2 Kong instance):

```
 curl -i -X POST \
  --url http://localhost:8001/services/invoke-lambda-service/routes \
  --data 'hosts[]=disciplinas.com'
```

E.4. INVOKE REMOTELLY (from your local computer), considering that your JSON file should be available

```
curl -i -X POST \
  -H "Content-Type: Application/json" \
  --url http://ec2-18-212-198-84.compute-1.amazonaws.com:8000/ \
  --header 'Host: disciplinas.com'\
  --data '@body.json'
```

You are expected to receive something similar with:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    79  100    43  100    36     43     36  0:00:01 --:--:--  0:00:01   220
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 43
Connection: keep-alive
Date: Tue, 14 Jan 2020 17:16:50 GMT
x-amzn-RequestId: 39bf3f69-af84-4568-96fb-b0277ea9e896
x-amz-apigw-id: GTOceEKUoAMFc4g=
x-custom-header: my custom header value
X-Amzn-Trace-Id: Root=1-5e1df782-21cac32327af61b6064ed89a;Sampled=0
X-Kong-Upstream-Latency: 28
X-Kong-Proxy-Latency: 0
Via: kong/2.0.0rc2

{"message":"Hello Integracao Empresarial!"}
```

F. Other example of connecting to a JAVA microservice

F.1. DIRECTLY TO JAVA WEBSERVICE BACKEND (as previous introduced in correspondingly tutorial):

```
curl -i -X POST -H "Content-Type: text/xml;charset=UTF-8" \
 --url http://ec2-52-91-101-30.compute-1.amazonaws.com:9298/calculate \
 --data '@calculate.xml'
```

F.2. CREATE SERVICE (in the AWS EC2 Kong instance):

```
 curl -i -X POST \
  --url http://localhost:8001/services/ \
  --data 'name=invoke-java-service' \
  --data 'url=http://ec2-52-91-101-30.compute-1.amazonaws.com:9298/calculate'\
```

F.3. CREATE ROUTE (in the AWS EC2 Kong instance):

```
 curl -i -X POST \
  --url http://localhost:8001/services/invoke-java-service/routes \
  --data 'hosts[]=calculate.com'
```

F.4. INVOKE REMOTELLY (from your local computer), considering that your XML file should be available

```
curl -i -X POST -H "Content-Type: text/xml;charset=UTF-8"\
  --url ec2-3-86-244-114.compute-1.amazonaws.com:8000/ \
  --header 'Host: calculate.com'  \
  --data '@calculate.xml'
```

You are expected to receive something similar with:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   461    0   202  100   259    202    259  0:00:01 --:--:--  0:00:01  1054
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Date: Tue, 14 Jan 2020 21:44:18 GMT
X-Kong-Upstream-Latency: 8
X-Kong-Proxy-Latency: 3
Via: kong/2.0.0rc2

<?xml version="1.0" ?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns2:mulResponse
xmlns:ns2="http://Webservice/"><return>182</return></ns2:mulResponse></S:Body></S:Envelope>
```

G. Endpoints with parameters

G.1.To allow the access to endpoints with parameters, e.g.,

```
http://URL_OF_ENDPOINT:8080/fruits/2/bananas
```

where the exact command if you invoke it by curl is:

```
curl -X 'PUT' \
  'http://URL_OF_ENDPOINT:8080/fruits/2/bananas' \
  -H 'accept: application/json'
```

You need to create a service and route accordingly:

G.2.The Kong service need to be defined only with the base path without the service identification, for instance:

```
curl -i -X POST \
--url "http://URL OF KONG:8001/services/" \
--data "name=invoke-quarkus4-service" \
--data "url=http://URL_OF_ENDPOINT:8080/"
```

G.3.After that the Kong route need to be defined using a regular expression, containing the fixed part and the variables with type indication, for instance:

```
curl -i -X POST \
--url "http://URL_OF_KONG:8001/services/invoke-quarkus4-service/routes" \
--data-urlencode "paths[]=~/fruits/(?<id>\d+)/(?<name>\S+)" \
--data "strip_path=false"
```

id and name are stored for later use by any Kong plug-in. d+ and S+ means, one or more decimal, and one or more alphanumeric. It is here that you customize the path accordingly with you endpoint application.

G.4.After that you can use the Kong route to invoke your endpoint with parameters.

```
curl -v -i -X PUT -H "accept: application/json" \
--url "http://URL_OF_KONG:8000/fruits/2/bananas"
```

H. Listing all the available services, routes and routes associated with services in Kong

Assuming the example in Section C - Creating a service and a route in Kong, the following command retrieved the available services:

```
curl -i http://localhost:8001/services
```

the answer should be similar to:

```
HTTP/1.1 200 OK
Date: Tue, 14 Jan 2020 21:46:40 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Access-Control-Allow-Origin: *
Server: kong/2.0.0rc2
Content-Length: 1029
X-Kong-Admin-Latency: 1

{"next":null,"data":[{"host":"ec2-52-91-101-30.compute-
1.amazonaws.com","created at":1579036862,"connect timeout":60000,"id":"2158a1c8-4ae4-48e4-a312-
e464b3010104","protocol":"http","name":"invoke-java-
service","read_timeout":60000,"port":9298,"path":"\/calculate","updated_at":1579036862,"retries":5,"write_tim
eout":60000,"tags":null,"client_certificate":null},{"host":"2p3fp5acxj.execute-api.us-east-
1.amazonaws.com","created_at":1579033755,"connect_timeout":60000,"id":"783cfce7-9142-4222-9ee0-
a2b50a507cd3","protocol":"https","name":"invoke-lambda-
service","read timeout":60000,"port":443,"path":"\/default\/PrimeiroMicroServico","updated at":1579033755,"re
tries":5,"write_timeout":60000,"tags":null,"client_certificate":null},{"host":"mockbin.org","created_at":1579
033726,"connect_timeout":60000,"id":"e9f67cc4-fa1e-4a23-ade3-d5e692b7368b","protocol":"http","name":"example-
service","read_timeout":60000,"port":80,"path":null,"updated_at":1579033726,"retries":5,"write_timeout":60000
,"tags":null,"client_certificate":null}]}
```

Then, the following command retrieved the available routes:

```
curl -i http://localhost:8001/routes
```

and the answer should be similar to:

```
HTTP/1.1 200 OK
Date: Tue, 14 Jan 2020 21:47:24 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Access-Control-Allow-Origin: *
Server: kong/2.0.0rc2
Content-Length: 1318
X-Kong-Admin-Latency: 1

{"next":null,"data":[{"id":"38cd55c6-adea-4fc6-86b9-
19ca0749e4a3","path_handling":"v0","paths":null,"destinations":null,"headers":null,"protocols":["http","https
"],"methods":null,"snis":null,"service":{"id":"783cfce7-9142-4222-9ee0-
a2b50a507cd3"},"name":null,"strip_path":true,"preserve_host":false,"regex_priority":0,"updated_at":1579033770
,"sources":null,"hosts":["disciplinas.com"],"https_redirect_status_code":426,"tags":null,"created_at":1579033
770},{"id":"8d188dde-c65a-402b-b659-
d8c39054087f","path_handling":"v0","paths":null,"destinations":null,"headers":null,"protocols":["http","https
"],"methods":null,"snis":null,"service":{"id":"2158a1c8-4ae4-48e4-a312-
e464b3010104"},"name":null,"strip_path":true,"preserve_host":false,"regex_priority":0,"updated_at":1579036920
,"sources":null,"hosts":["calculate.com"],"https_redirect_status_code":426,"tags":null,"created_at":157903692
0},{"id":"cca09713-5cdf-4db1-8ab0-
606e2bd76d1c","path handling":"v0","paths":null,"destinations":null,"headers":null,"protocols":["http","https
"],"methods":null,"snis":null,"service":{"id":"e9f67cc4-fa1e-4a23-ade3-
d5e692b7368b"},"name":null,"strip_path":true,"preserve_host":false,"regex_priority":0,"updated_at":1579033736
,"sources":null,"hosts":["example.com"],"https_redirect_status_code":426,"tags":null,"created_at":1579033736}
]}[
```

Finally, the following command retrieved the available routes for a specific service:

```
curl -i http://localhost:8001/services/example-service/routes
```

and the answer should be similar to:

```
HTTP/1.1 200 OK
Date: Tue, 14 Jan 2020 16:28:47 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Access-Control-Allow-Origin: *
Server: kong/2.0.0rc2
Content-Length: 452
```

```
X-Kong-Admin-Latency: 2

{"next":null,"data":[{"id":"6b7df64c-9ed9-4d5c-87a3-
58b9935d01f4","path handling":"v0","paths":null,"destinations":null,"h
eaders":null,"protocols":["http","https"],"methods":null,"snis":null,"service":{"id":"af50ee81-66e3-4dbe-
b2f6-f25e72c549a1
"},"name":null,"strip_path":true,"preserve_host":false,"regex_priority":0,"updated_at":1579017405,"sources":n
ull,"hosts":[
"example.com"],"https_redirect_status_code":426,"tags":null,"created_at":1579017405}]}
```

I. Install Konga Open Source platform

Optionally, for administration purposes of Kong you can install Konga, which is an unofficial app. No affiliated with Kong. It offer an easy management of all the Kong concepts.

I.1. Pull the following Docker image locally on your PC:

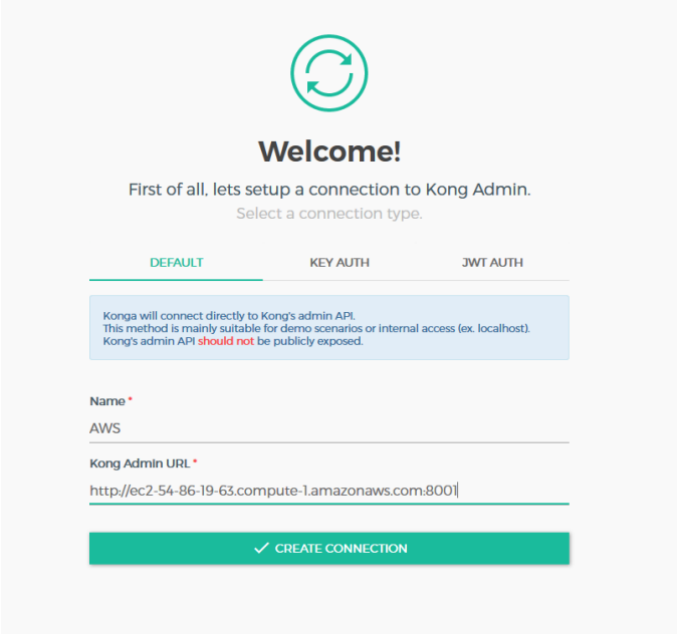
```
docker pull pantsel/konga
```

I.2. Then, execute it:

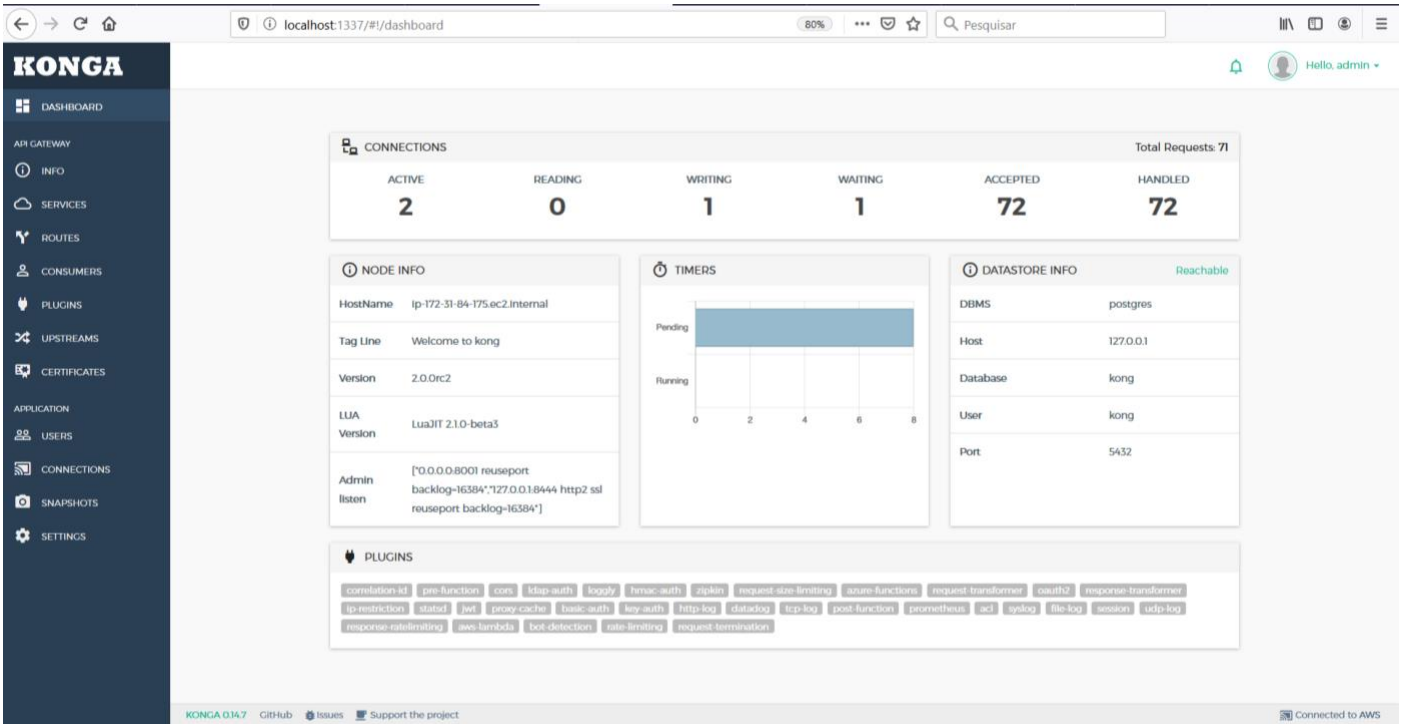
```
docker run -d --name konga -p 1337:1337 pantsel/konga
```

I.3. Open your browser with the following url: http://<ec2_DNS_NAME>:1337 and create an admin account (keep the password!)

I.4. Then, provide the Kong URL with the 8001 port reference in your browser, accordingly with the following example:



A similar result is obtained, where you can manage the Kong concepts:


localhost:1337/#!/dashboard
80%
Pesquisar
KONGA
Hello, admin
DASHBOARD
API GATEWAY
INFO
SERVICES
ROUTES
CONSUMERS
PLUGINS
UPSTREAMS
CERTIFICATES
APPLICATION
USERS
CONNECTIONS
SNAPSHOTS
SETTINGS
CONNECTIONS
Total Requests: 71
ACTIVE
2
READING
0
WRITING
1
WAITING
1
ACCEPTED
72
HANDLED
72
NODE INFO
HostName
ip-172-31-84-175.ec2.internal
Tag Line
Welcome to kong
Version
2.0.0rc2
LUA Version
LuaJIT 2.1.0-beta3
Admin listen
["0.0.0.0:8001 reuseport backlog=16384","127.0.0.1:8444 http2 ssl reuseport backlog=16384"]
TIMERS
Pending
Running
0
2
4
6
8
DATASTORE INFO
Reachable
DBMS
postgres
Host
127.0.0.1
Database
kong
User
kong
Port
5432
PLUGINS
correlation-id
pre-function
cors
ldap-auth
loggly
hmac-auth
zipkin
request-size-limiting
azure-functions
request-transformer
oauth2
response-transformer
ip-restriction
statsd
jwt
proxy-cache
basic-auth
key-auth
http-log
datadog
tcp-log
post-function
prometheus
acl
syslog
file-log
session
udp-log
response-ratelimiting
aws-lambda
bot-detection
rate-limiting
request-termination
KONGA 0.14.7
GitHub
Issues
Support the project
Connected to AWS

Other references

- Kong Inc. https://konghq.com/
- https://docs.konghq.com/install/redhat/?_ga=2.90626585.229207644.1612972455-1733373400.1609865717
- Konga https://pantsel.github.io/konga/
- https://wiki.postgresql.org/wiki/YUM_Installation
- https://www.postgresql.org/download/linux/redhat/
- https://www.cyberciti.biz/faq/install-and-setup-postgresql-on-rhel-8/
- https://docs.konghq.com/1.4.x/admin-api/
- For deploying Auth2.0 plug/in in Kong read the manual: https://docs.konghq.com/hub/kong-inc/oauth2/
- https://docs.konghq.com/gateway/2.8.x/install-and-run/docker/