## Table of Contents

## 1 DOCUMENT INTRODUCTION

As part of the POC and learning exercise I have done this project and hosted in GCP environment. This document walks through the steps and the thought process involved behind this pet project.

Feel free to add or pass on your comments.

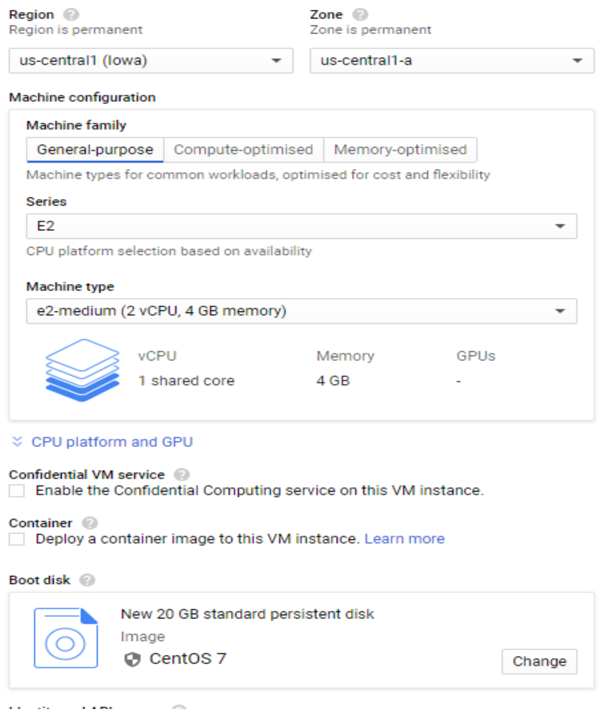
**Set up Kafka Cluster with Zookeeper - Write a Rest API to publish and Consume the messages.**

This is done using a GCP VM instances inorder to make the demo seamless.

Below are the steps executed inorder to get a kafka cluster with 2 brokers and a zookeeper instance all in a separate VM. They all in a Region -1 with static ips configured and firewall settings rules defined in such a way that required ports are exposed.

#### 1.1.1.1 Provision VM instance in GCP and install Kafka (for brokers and Zookeeper)

Created a new cluster **Kafka-Cluster1 - Cluster ID: FeyYsUgjRia39bbRhJtVqQ** . Below is the configuration used for all the nodes.



Region
Region is permanent
us-central1 (Iowa)

Zone
Zone is permanent
us-central1-a

Machine configuration

Machine family
General-purpose | Compute-optimised | Memory-optimised
Machine types for common workloads, optimised for cost and flexibility

Series
E2
CPU platform selection based on availability

Machine type
e2-medium (2 vCPU, 4 GB memory)

| vCPU | Memory | GPUs |
|---|---|---|
| 1 shared core | 4 GB | - |

CPU platform and GPU

Confidential VM service
☐ Enable the Confidential Computing service on this VM instance.

Container
☐ Deploy a container image to this VM instance. Learn more

Boot disk
New 20 GB standard persistent disk
Image
CentOS 7
Change

Install openjdk8 and wget inorder to download and run the kafka binaries.

*sudo yum -y install java-1.8.0-openjdk*

*sudo yum -y install wget*

*wget https://mirrors.estointernet.in/apache/kafka/2.6.0/kafka_2.12-2.6.0.tgz*

*tar -xzf kafka_2.12-2.6.0.tgz*

*sudo vi .bash_profile*

add an entry to the bin so that kafka shell commands are accessible from root.
*$HOME/kafka_2.12-2.6.0/bin*

### 1.1.1.2 Configure Zookeeper

I have brought up an another VM instance in the cluster for Zookeeper and executed the below commands to set up ZK.

Changed the data directory to a custom location

*vi $HOME/kafka_2.12-2.6.0/config/zookeeper.properties*

*dataDir=/home/shobithprv000/zookeeper_data*

*mkdir $HOME/zookeeper_data*

Start the ZK

*zookeeper-server-start.sh /home/shobithprv000/kafka_2.12-2.6.0/config/zookeeper.properties*

*Inorder to make sure ZK runs whenever I start the VM instance(just to make the demo look seamless), I configured zookeeper start command in the* rc.local *file and enable systemctl.*

*sudo vi /etc/rc.d/rc.local*

*/home/shobithprv000/kafka_2.12-2.6.0/bin/zookeeper-server-start.sh /home/shobithprv000/kafka_2.12-2.6.0/config/zookeeper.properties > /dev/null 2>&1 &*

*sudo chmod +x /etc/rc.d/rc.local*
*sudo systemctl enable rc-local*
*sudo systemctl start rc-local*

Test the zookeeper server : below commands show the brokers already configured with the zk instance, which in this case 0- node doesn't exist.

*kafka_2.12-2.6.0/bin/zookeeper-shell.sh 35.198.207.39:2181 ls /brokers/ids*

### 1.1.1.3 Configure Kafka Brokers

Now that we have installed the kafka and the zookeeper in the cluster , I will bring 2 instances of kafka brokers in the cluster

Created a data directory for kafka
*mkdir /home/shobithprv000/kafka_data*

Each cluster-topic configured for 2 partitions and 2 copies of a given topic in the kafka cluster. Below are the configurations that's put into the server.properties file.

*broker.rack=RACK1* (RACK2 for the second instance)
*log.dirs=/home/shobithprv000/kafka_data*
*offsets.topic.num.partitions=2 -*
*offsets.topic.replication.factor=2*
*transaction.state.log.replication.factor=1*
*transaction.state.log.min.isr=1*
*min.insync.replicas=2*

*default.replication.factor=2*

Earlier created zookeeper ip with port is assigned here.
*zookeeper.connect=10.148.0.4:2181*

*The below configuration is needed as part of the GCP official page https://cloud.google.com/solutions/processing-messages-from-kafka-hosted-outside-gcp*
*the externally advertised addresses of the Kafka brokers in the cluster differ from the internal network interfaces that Kafka uses. Which is true here because I am using my laptop as a client, I am using the the* advertised.listeners *property shown in this below config* server.properties *snippet:*

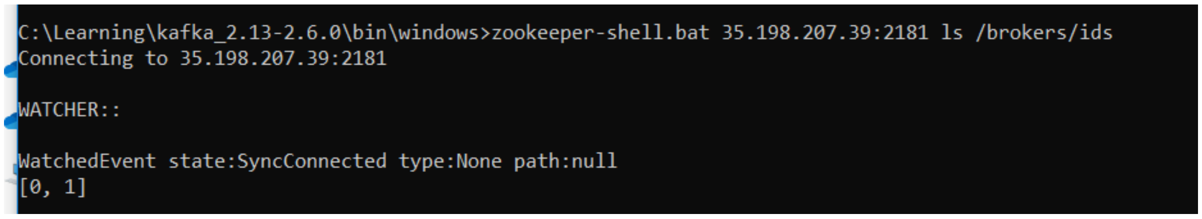
```
listeners=INTERNAL://0.0.0.0:9092,EXTERNAL://0.0.0.0:19092
listener.security.protocol.map=INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT
advertised.listeners=EXTERNAL://35.247.152.228:19092,INTERNAL://10.148.0.5:9092
inter.broker.listener.name=INTERNAL
```

*As you see I have configured the internal ports of kafka broker to be **9092** but external facing ports to be **19092** for all the vm instances.*

*Make sure the Kafka broker runs when we start the instance.*

```
sudo vi /etc/rc.d/rc.local
/home/shobithprv000/kafka_2.12-2.6.0/bin/kafka-server-start.sh /home/shobithprv000/kafka_2.12-2.6.0/config/server.properties > /dev/null 2>&1 &

sudo chmod +x /etc/rc.d/rc.local
sudo systemctl enable rc-local
sudo systemctl start rc-local
```



```
C:\Learning\kafka_2.13-2.6.0\bin\windows>zookeeper-shell.bat 35.198.207.39:2181 ls /brokers/ids
Connecting to 35.198.207.39:2181

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[0, 1]
```

As you see above I can connect to the zk and see the brokers that's configured to it

#### 1.1.1.4 *Testing the Kafka Installation*

Below are the sample commands I have used to test the set up ( all in the VM's , hence local ips and ports are used)

```
kafka-topics.sh --create --zookeeper 10.148.0.4:2181 --replication-factor 2 --partitions 2 --topic test

kafka-topics.sh --list --zookeeper 10.148.0.4:2181

kafka-console-producer.sh --broker-list 10.148.0.3:9092 --topic test

kafka-console-consumer.sh --bootstrap-server 10.148.0.2:9092 --topic test --from-beginning
```

### 1.1.2 Have a Kafka cluster on both the region / Datacentre and configure Mirror making (Open source)

I have brough up an another instance of the cluster in a different region again with 2 brokers and a zookeeper instance all in a separate VM. Done the changes to the VM as per the case 1 .

### 1.1.2.1 Configure MirrorMaker

Installed kafka version comes with version 2.0 of the MirrorMaker the same is been used for the POC. MirrorMaker is a stand-alone tool for copying data between two Apache Kafka clusters. I see it to be little more than a Kafka consumer and producer hooked together. Data will be read from topics in the origin cluster and written to a topic with the same name in the destination cluster.

I provisioned a new VM for **MirrorMaker** to keep the responsibility separate.

Steps to install the kafka and JDK is as metioned in the case1 documentaiton:

- Create a consumer configuration

```
touch mirrormakerOneConsumer.config
vi mirrormakerOneConsumer.config
```

Add the below contents to it. I am configuring the brokers from Cluster 1 **Cluster ID: FeyYsUgjRia39bbRhJtVqQ** here, so that cluster 1 is the source node of the mm and cluster 2 is the sink node.

```
bootstrap.servers=34.87.7.75:19092,35.247.152.228:19092
exclude.internal.topics=true
client.id=mirror_maker_consumer
group.id=mirror_maker_consumer
```

- Create a producer configuration

```
touch mirrormakerOneProducer.config
vi mirrormakerOneProducer.config
```

Add the below contents to it. I am configuring the brokers from Cluster 2 **Cluster ID: ajpFUr7bSPisPCL7XyIwcg** here, so that cluster 2 is where the mm should mirror the data from source

```
bootstrap.servers=35.187.253.251:19092,34.126.116.238:19092
acks=1
batch.size=50
client.id=mirror_maker_producer
```

Execute the below command to configure producer and consumer :

```
/home/shobithprv000/kafka_2.12-2.6.0/bin/kafka-mirror-maker.sh --consumer.config mirrormakerOneConsumer.config --num.streams 1 --producer.config mirrormakerOneProducer.config --whitelist=".*"
```

Test it in my local PC

```
kafka-consumer-groups.bat --bootstrap-server 34.87.7.75:19092 --describe --group mirror_maker_consumer
```

Test Delete Messages

```
kafka-console-consumer.bat --bootstrap-server 34.126.116.238:19092 --topic mirrormaker-1 --partition 1 --offset 5
```

topic partitons :

```
kafka-topics.bat --describe --zookeeper 35.198.207.39:2181 --topic mirrormaker-1
```

### 1.1.3 Explore Cruise control – open source plug-in and develop a POC for your Kafka clusters

Cruise Control is a Kafka load balancing component that can be used in large Kafka installations. Cruise Control can automatically balance the partitions based on specific conditions, and when adding or removing Kafka brokers.

I have set up the cruise control for the cluster -2 in one of the broker machines (OS- Centos -7) .

As per the documentation of cruice control I only see linux box or docker installations supported.

- Steps to configure Cruise Control

First get the git installed

```
sudo yum -y install git
```

get the java installed to the box

```
sudo yum install java-1.8.0-openjdk-devel
```

Cloned the Cruice Control from the below path

```
git clone https://github.com/linkedin/cruise-control.git && cd cruise-control/
```

and build the solution

```
./gradlew.jar
```

Copy the generated cruise control jar to kafka library

```
cp cruise-control/cruise-control-metrics-reporter/build/libs/cruise-control-metrics-reporter-2.0.132-SNAPSHOT.jar
kafka_2.12-2.6.0/libs
```

modify the kafka server.properties to configure matrics reporter from cruisecontrol

```
sudo vi kafka_2.12-2.6.0/config/server.properties
metric.reporters=com.linkedin.kafka.cruisecontrol.metricsreporter.CruiseControlMetricsReporter
```

*Modify the cruise control properties to have zookeeper and all broker references of a cluster its configured to*

```
sudo vi cruise-control/config/cruisecontrol.properties
```

Copy dependent libraries and start the cruise control ; make sure the ZK and Kafka server is running as configured above

```
./gradlew jar copyDependantLibs
./kafka-cruise-control-start.sh config/cruisecontrol.properties 9090
```

*Cruise control should be accessible at*

*http://35.187.253.251:9090/kafkacruisecontrol/API _Cruise_Control*

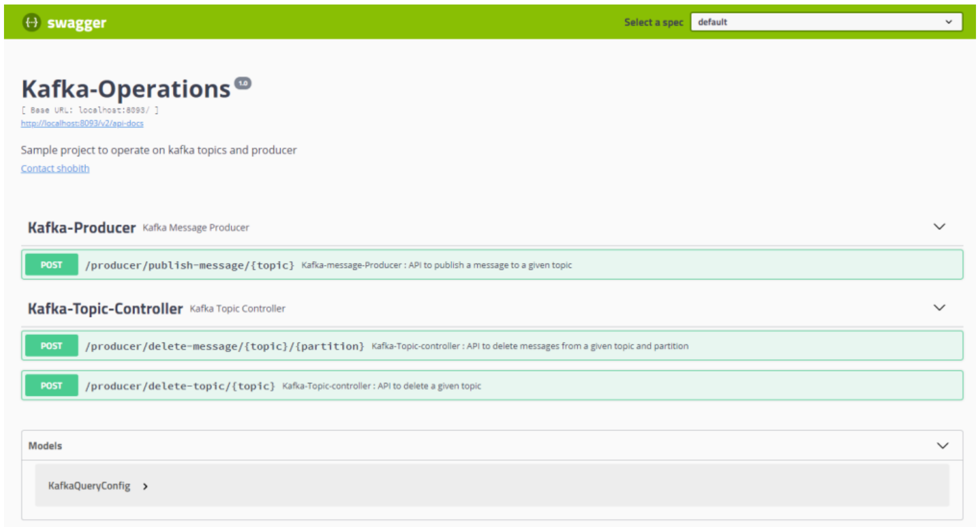
*API _Cruise_Control – Refers to many end points exposed by cruise control for querying the state,load and partition details of the cluster. It also supports POST requests which can trigger workload balance or add/decommission a list of brokers from the kafka cluster among many other things.*

*http://35.187.253.251:9090/kafkacruisecontrol/kafka_cluster_state*

### 1.1.4 Case 4 : Write a API using Kafka API to Delete the messages within the given Kafka and Delete the topic

*Attaching below the source code which consists of API's : Its not a production level code , just to get a hang of it*

kafka-producer.zip



swagger — Select a spec: default

Kafka-Operations 1.0
[ Base URL: localhost:8093/ ]
http://localhost:8093/v2/api-docs

Sample project to operate on kafka topics and producer
Contact shobith

**Kafka-Producer** Kafka Message Producer

POST `/producer/publish-message/{topic}` Kafka-message-Producer : API to publish a message to a given topic

**Kafka-Topic-Controller** Kafka Topic Controller

POST `/producer/delete-message/{topic}/{partition}` Kafka-Topic-controller : API to delete messages from a given topic and partition

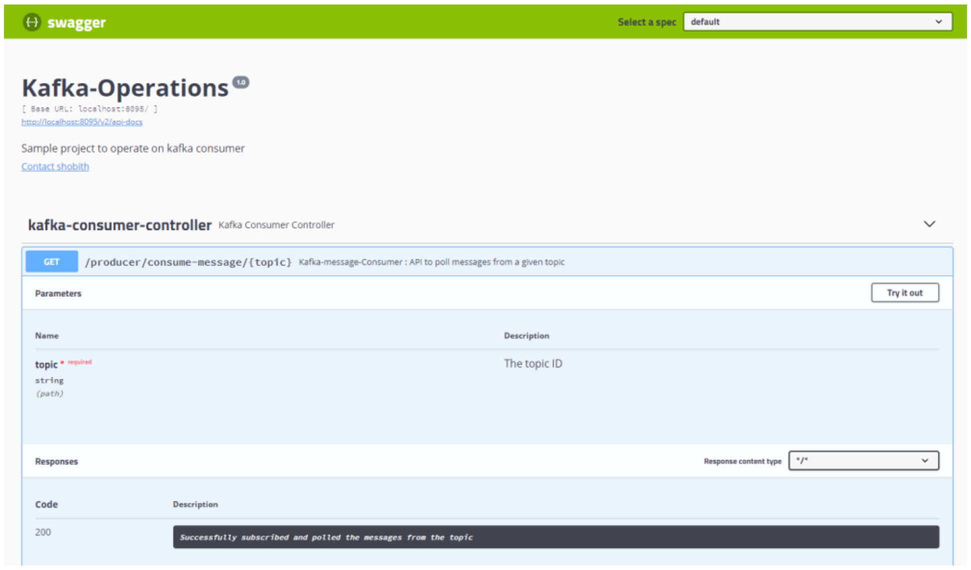
POST `/producer/delete-topic/{topic}` Kafka-Topic-controller : API to delete a given topic

Models
KafkaQueryConfig

Kafka consumer POC is attached here :

This shows 1. Spring boot – Kafka integration subscription POC and 2. Topic polling using Consumer Factory using an API endpoint

kafka-consumer.zip



swagger — Select a spec: default

Kafka-Operations 1.0
[ Base URL: localhost:8095/ ]
http://localhost:8095/v2/api-docs

Sample project to operate on kafka consumer
Contact shobith

**kafka-consumer-controller** Kafka Consumer Controller

GET `/producer/consume-message/{topic}` Kafka-message-Consumer : API to poll messages from a given topic

Parameters — Try it out

| Name | Description |
|---|---|
| topic * required string (path) | The topic ID |

Responses — Response content type: */*

| Code | Description |
|---|---|
| 200 | Successfully subscribed and polled the messages from the topic |

***Below are the cluster details of the Kafka Use case :***

*Cluster 2* ***Cluster ID: ajpFUr7bSPisPCL7XyIwcg***

| ***Machine*** | ***Local IP*** | ***Network IP*** |
|---|---|---|
| *cluster2-kafka-1* | *10.148.0.2* | *35.187.253.251* |

| | | |
|---|---|---|
| *cluster2-kafka-2* | *10.148.0.3* | *34.126.116.238* |
| *cluster2-mirrormaker* | *10.148.0.5* | *35.198.194.85* |
| *cluster2-zookeeper* | *10.148.0.4* | *35.198.207.39* |

*Cluster 1* ***Cluster ID: FeyYsUgjRia39bbRhJtVqQ***

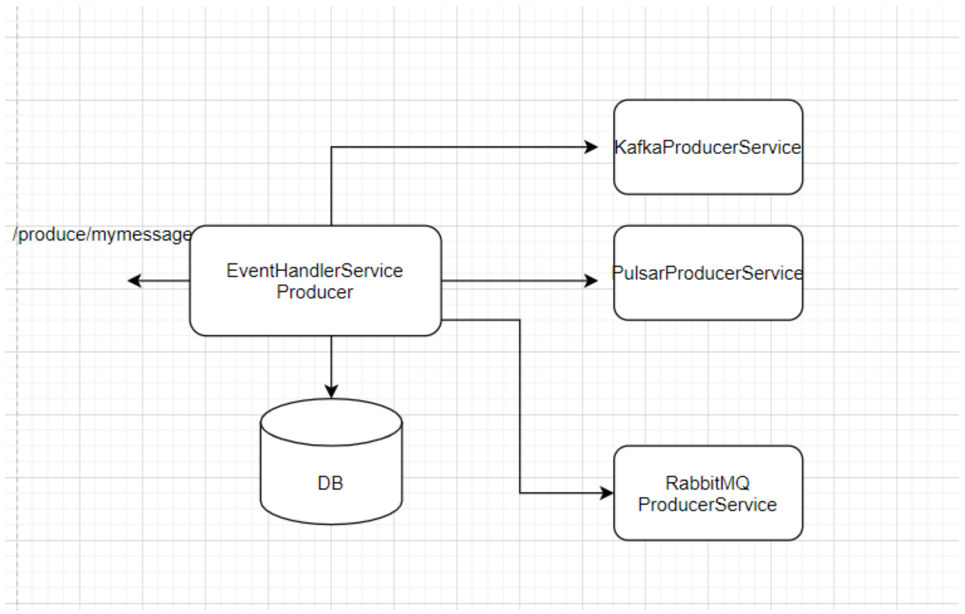
| ***Machine*** | ***Local IP*** | ***Network IP*** |
|---|---|---|
| *Cluster1-kafka-1* | *10.148.0.2* | *35.187.253.251* |
| *Cluster1-kafka-2* | *10.148.0.4* | *34.126.116.238* |
| *Cluster1-zookeeper* | *10.128.0.2* | *35.184.194.131* |

**How we can build a abstraction layer on top of Event processing :** User will have only 2 API's - Consumer or Producer API, user should not aware what we were using in the back-ground, today it might be Kafka and in future it might be Pulsar… User should not have any impact.

Basically the need here is to produce a façade layer on top of the messaging platforms so that user is unware of the underlying complexity or the structure and can be easily replaced.

**Façade Microservice:**

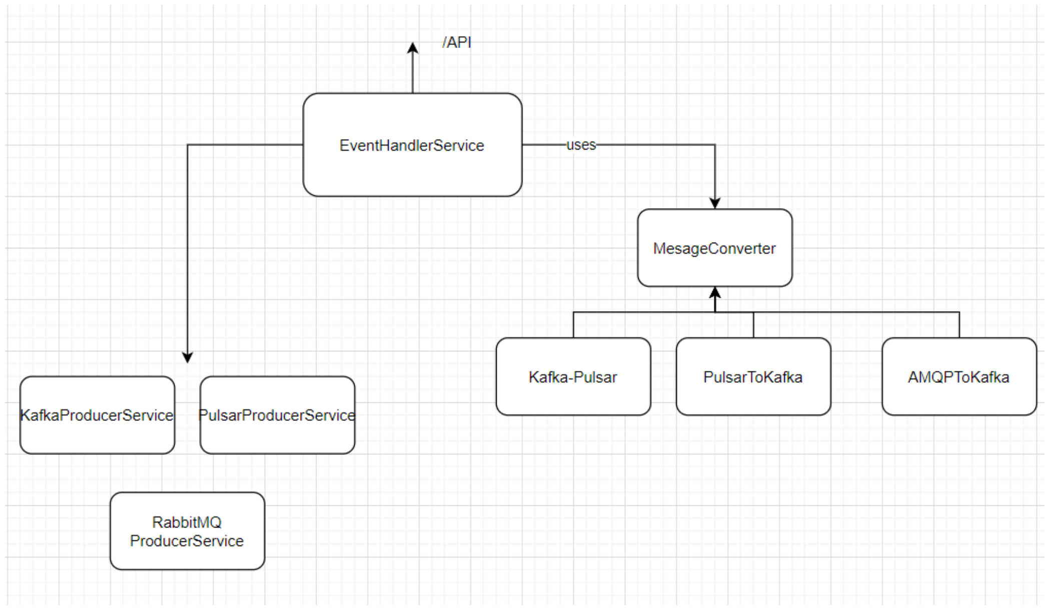
Here I will be adding an another layer of **EventHandlerService** which is responsible for handling the message conversion and configurations needed to make sure that the underlying messaging pattern is handled



As you see above

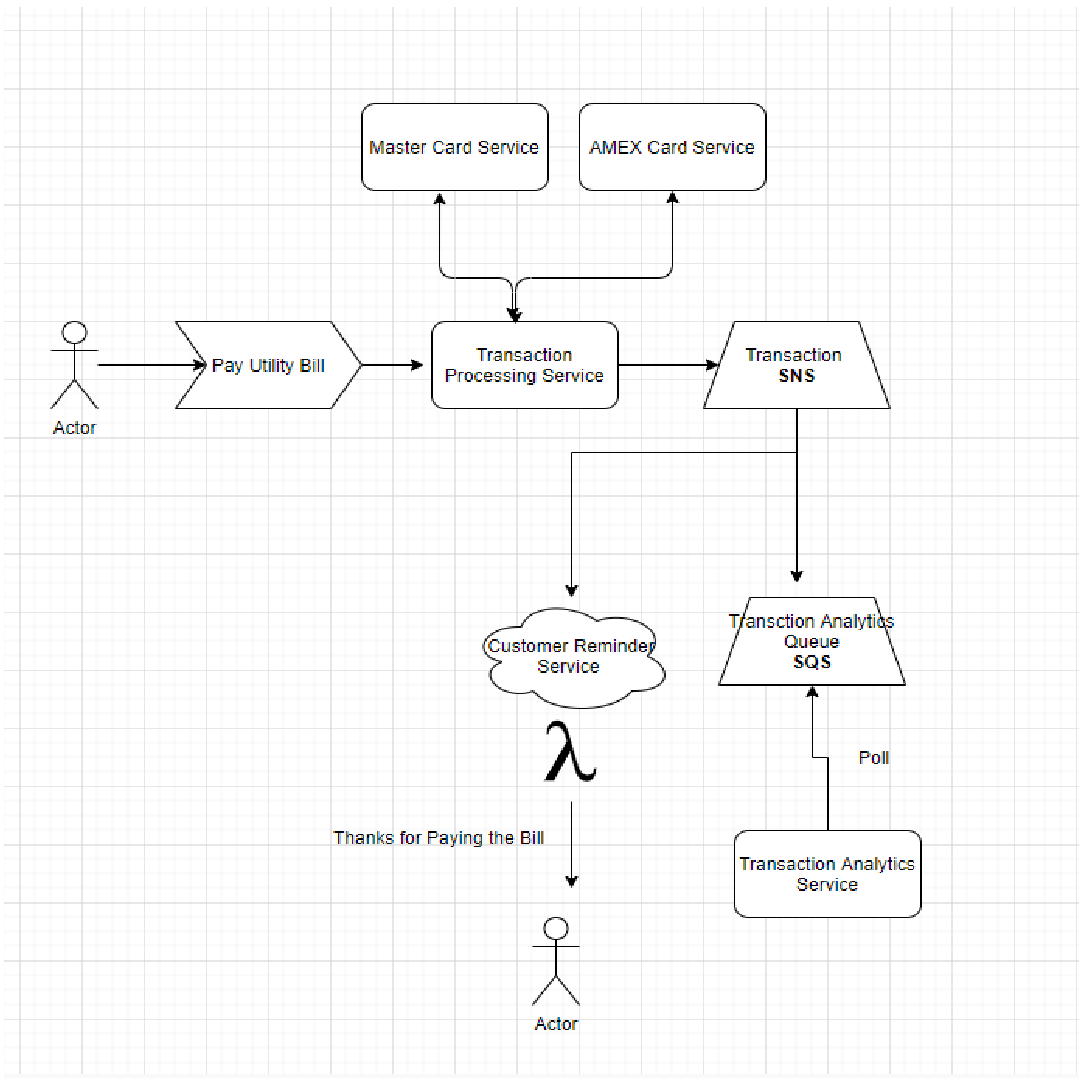
**produce/mymessage** – mymessage is a format that the customer is exposed. DB holds the configuration that is needed for the communication to the layer under it such as RabbitMQProducerService or KafkaProducerService.

This way irrespective of the underlying platform , user API is unchanged and can seamlessly work with underlying platform



MessageConverter : Is a Java Library which handles the conversion logic with respect to message and message headers

NOTE : I came to know about the launch of Kafka-on-Pulsar, or KoP, which was launched in March 2020 by OVHCloud and StreamNative. By adding the KoP protocol handler to an existing Pulsar cluster, you can now migrate existing Kafka applications and services to Pulsar without modifying the code.


Master Card Service
AMEX Card Service
Actor
Pay Utility Bill
Transaction Processing Service
Transaction SNS
Customer Reminder Service
λ
Transction Analytics Queue SQS
Poll
Thanks for Paying the Bill
Transaction Analytics Service
Actor

**When should we use SNS/SQS or Kafka:**

This is a classic trade-off of speed of development and ease of development(SQS /SNS) vs the best, very modular and rich in feature /personalized solution(Kafka), that has more overhead for the first implementation but scales better.

I personally of the opinion that if we are prototyping something, favor speed of development, AWS tools is the way to go. If the requirements are frozen and require significant scale, I would definitely take the time to use kafka. I also am a big believer in using-open-source-makes-the-world-better ;)