COEN 6731: Distributed Software Systems

Assignment – III

Submitted to – YAN LIU

Submitted by –Rohan K (40196377)

Date of submission – April 18, 2023

Task 1. Install rabbitmq server

For this assignment I have set up a RabbitMQ server for the development purpose using docker-compose. I have downloaded the official rabbitmq docker image to run it.

Prerequisites for this was to have docker and docker-compose installed by running the below commands:

$ docker -v and $ docker-compose -v

Further I have created a docker-compose.yaml file with the below content –

docker-compo... CommonConfig... MessagePubli... TestConsur

```yaml
version: "3.6"
# https://docs.docker.com/compose/compose-file/
services:
  rabbitmq:
    image: 'rabbitmq:3.6-management-alpine'
    ports:
      # The standard AMQP protocol port
      - '5672:5672'
      # HTTP management UI
      - '15672:15672'
    environment:
      # The location of the RabbitMQ server.  "amqp" is the protocol;
      # "rabbitmq" is the hostname.  Note that there is not a guarantee
      # that the server will start first!  Telling the pika client library
      # to try multiple times gets around this ordering issue.
      AMQP_URL: 'amqp://rabbitmq?connection_attempts=5&retry_delay=5'
      RABBITMQ_DEFAULT_USER: "ron"
      RABBITMQ_DEFAULT_PASS: "ron"
    networks:
      - network
networks:
  # Declare our private network.  We must declare one for the magic
  # Docker DNS to work, but otherwise its default settings are fine.
  network: {}
```

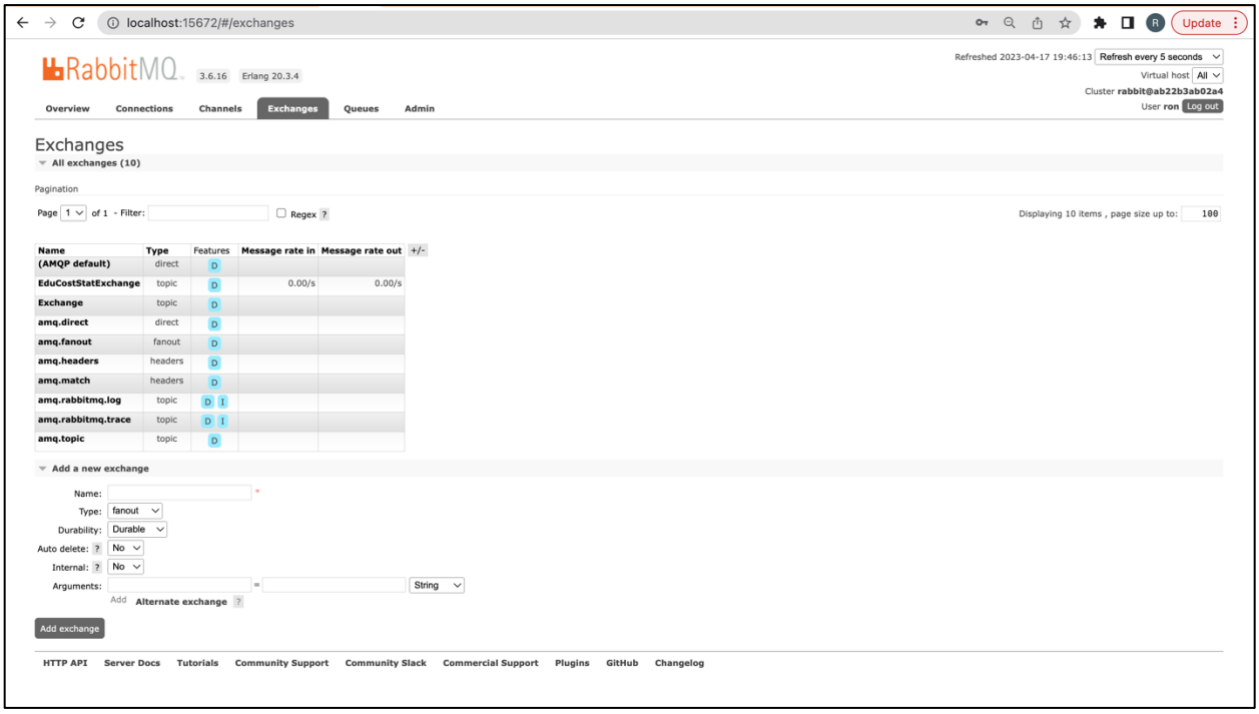
As seen, the rabbitmq user name and password is – “ron”

Then I ran the docker-compose up command in a terminal, waited for the command to complete its execution. This downloaded the specified rabbitmq docker image, ran the rabbitmq-admin UI on #15672 and the Server listens on #5672 for messaging. Once the server is up, below log is seen -

```
/Users/rohankodavalla/Documents/rabbitmq_test — docker-compose up        ~/Documents/rabbitmq_test — -bash        ~/Downloads — opc@final-instance:~ — -bash
(base) rohans-air:rabbitmq_test rohankodavalla$ docker-compose up
[+] Running 1/0
 ✔ Container rabbitmq_test-rabbitmq-1  Created                                                                    0.0s
Attaching to rabbitmq_test-rabbitmq-1
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:05 ===
rabbitmq_test-rabbitmq-1  | Starting RabbitMQ 3.6.16 on Erlang 20.3.4
rabbitmq_test-rabbitmq-1  | Copyright (C) 2007-2018 Pivotal Software, Inc.
rabbitmq_test-rabbitmq-1  | Licensed under the MPL.  See http://www.rabbitmq.com/
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  |                  RabbitMQ 3.6.16. Copyright (C) 2007-2018 Pivotal Software, Inc.
rabbitmq_test-rabbitmq-1  |   ##  ##      Licensed under the MPL.  See http://www.rabbitmq.com/
rabbitmq_test-rabbitmq-1  |   ##  ##
rabbitmq_test-rabbitmq-1  |   ##########  Logs: tty
rabbitmq_test-rabbitmq-1  |   ######  ##        tty
rabbitmq_test-rabbitmq-1  |   ##########
rabbitmq_test-rabbitmq-1  |                  Starting broker...
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:05 ===
rabbitmq_test-rabbitmq-1  | node           : rabbit@ab22b3ab02a4
rabbitmq_test-rabbitmq-1  | home dir       : /var/lib/rabbitmq
rabbitmq_test-rabbitmq-1  | config file(s) : /etc/rabbitmq/rabbitmq.config
rabbitmq_test-rabbitmq-1  | cookie hash    : 6nF9cUeJuIDa0AsR8rfxYg==
rabbitmq_test-rabbitmq-1  | log            : tty
rabbitmq_test-rabbitmq-1  | sasl log       : tty
rabbitmq_test-rabbitmq-1  | database dir   : /var/lib/rabbitmq/mnesia/rabbit@ab22b3ab02a4
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | Memory high watermark set to 1573 MiB (1649804902 bytes) of 3933 MiB (4124512256 bytes) total
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | Enabling free disk space monitoring
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | Disk free limit set to 50MB
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | Limiting to approx 1048476 file handles (943626 sockets)
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | FHC read buffering:  OFF
rabbitmq_test-rabbitmq-1  | FHC write buffering: ON
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | Waiting for Mnesia tables for 30000 ms, 9 retries left
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | Waiting for Mnesia tables for 30000 ms, 9 retries left
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | Priority queues enabled, real BQ is rabbit_variable_queue
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | Starting rabbit_node_monitor
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | Management plugin: using rates mode 'basic'
rabbitmq_test-rabbitmq-1  |
rabbitmq_test-rabbitmq-1  | =INFO REPORT==== 17-Apr-2023::21:24:06 ===
rabbitmq_test-rabbitmq-1  | msg_store_transient: using rabbit_msg_store_ets_index to provide index
rabbitmq_test-rabbitmq-1  |
```

Further, when navigated to http://localhost:15672 in browser and logged in to the management dashboard with “ron” as both username and password, below is the RabbitMQ admin management dashboard observed –

The UI when logged in is as below -



So, the RabbitMQ installation is successful.

Task 2. Program the producer and consumer using rabbitmq exchange topic libraries

In order to Send and Receive a message in RabbitMQ below steps were followed -

1) **Programmed the Publisher :**

docker-compo... | MessagePubli... | CreateQueues... | rabbitmq-pa... | 19

```java
package com.amqp.basic.queue;

import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Sorts;

import org.bson.Document;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Consumer;
import com.rabbitmq.client.DefaultConsumer;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

// Run me - third
public class MessagePublisher {
    public static void main(String[] args) throws IOException, TimeoutException {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setUsername("ron");
        factory.setPassword("ron");
        factory.setPort(5672);
        factory.setVirtualHost("/");


        try (Connection connection = factory.newConnection();
             Channel channel = connection.createChannel()) {

            // Declare the exchanges
            channel.exchangeDeclare("EduCostStatExchange", "topic", true);
            //channel.exchangeDeclare("EduCostStatExchange", "topic", true);
            //channel.exchangeDeclare("EduCostStatQueryThree", "topic3");
            //channel.exchangeDeclare("EduCostStatQueryFour", "topic4");
            //channel.exchangeDeclare("EduCostStatQueryFive", "topic5");

            Consumer consumer = new DefaultConsumer(channel) {
                //...
            };



            // MongoDB database and collection setup
            MongoClient mongoClient = MongoClients.create("mongodb+srv://rkodava:Dimpu1997@educoststat.ioim58e.mongodb.net/?retryWrites=
            MongoDatabase database = mongoClient.getDatabase("test");
            MongoCollection<Document> collection = database.getCollection("EduCostStatQueryFive");
            FindIterable<Document> results = collection.find();


            // Query for EduCostStatQueryOne
           /*
            int year = 2013;
            String state = "Alabama";
            String type = "Private";
            String length = "4-year";
            String expense = "Fees/Tuition";
            String message = "Message for EduCostStatQueryOne";

         // Query for EduCostStatQueryTwo
            /*
            int year = 2013;
            String type = "Private";
            String length = "4-year";
            String message = "Message for EduCostStatQueryTwo";

         // Query for EduCostStatQueryThree
```

MessagePubli... CreateExchan... CreateQueues... CreateBindin... 19

```java
            // MongoDB database and collection setup
            MongoClient mongoClient = MongoClients.create("mongodb+srv://rkodava:Dimpu1997@educoststat.ioim58e.mongodb.net/?retryWrites
            MongoDatabase database = mongoClient.getDatabase("test");
            MongoCollection<Document> collection = database.getCollection("EduCostStatQueryFive");
            FindIterable<Document> results = collection.find();


            // Query for EduCostStatQueryOne
           /*
            int year = 2013;
            String state = "Alabama";
            String type = "Private";
            String length = "4-year";
            String expense = "Fees/Tuition";
            String message = "Message for EduCostStatQueryOne";

         // Query for EduCostStatQueryTwo
            /*
            int year = 2013;
            String type = "Private";
            String length = "4-year";
            String message = "Message for EduCostStatQueryTwo";

         // Query for EduCostStatQueryThree

            int year = 2013;
            String type = "Private";
            String length = "4-year";
            String message = "Message for EduCostStatQueryThree";

        // Query for EduCostStatQueryFour
            int latestYear = 2021;
            String type = "Public";
            String length = "4-year";
            int[] pastYears = new int[]{2020, 2019, 2018, 2017};
            String message = "Message for EduCostStatQueryFour";
    */
        // Query for EduCostStatQueryFive

            int year = 2019;
            String type = "Public In-State";
            String length = "4-year";
            String message = "Message for EduCostStatQueryFive";

         // Display query results
            for (Document result : results) {
                System.out.println(result.toJson());
            }
            // Publish the message to the exchange for EduCostStatQueryOne
            //channel.basicPublish("EduCostStatExchange", "Cost-" + year + "-" + state + "-" + type + "-" + length, null, message.getBy

            // Publish the message to the exchange for EduCostStatQueryTwo
            //message = "Message for EduCostStatQueryTwo";
            //channel.basicPublish("EduCostStatExchange", "Top5-Expensive-" + year + "-" + type + "-" + length, null, message.getBytes(

            // Publish the message to the exchange for EduCostStatQueryThree
           // message = "Message for EduCostStatQueryThree";
           // channel.basicPublish("EduCostStatExchange", "Top5-Economic-" + year + "-" + type + "-" + length, null, message.getBytes()

            // Publish the message to the exchange for EduCostStatQueryFour
          // message = "Message for EduCostStatQueryFour";
           //channel.basicPublish("EduCostStatExchange", "Top5-HighestGrow-2020-2022-5-" + type + "-" + length, null, message.getBytes(
           //channel.basicPublish("EduCostStatExchange", "Top5-HighestGrow-2021" + latestYear+ "-" + type + "-" + length + "-" + pastYe
            // Publish the message to the exchange for EduCostStatQueryFive
           message = "Message for EduCostStatQueryFive";
           channel.basicPublish("EduCostStatExchange", "AverageExpense-" + year + "-" + type + "-" + length, null, message.getBytes());

        }
    }
}
```

The above is a Java code for publishing a message to a RabbitMQ exchange based on data retrieved from a MongoDB database. Here's a breakdown of the code:

- The necessary libraries are imported for MongoDB and RabbitMQ.
- The main method is defined, which throws IOException and TimeoutException exceptions.
- A new ConnectionFactory object is created, which specifies the connection parameters for RabbitMQ such as the hostname, username, password, port, and virtual host.
- A new connection and channel to RabbitMQ are created and declared.
- A new Consumer object is created for the RabbitMQ channel.

- A new MongoClient object is created, which specifies the connection parameters for MongoDB such as the connection string.
- The database and collection are specified and retrieved using the MongoDatabase and MongoCollection objects.
- A MongoDB query is executed to retrieve the data from the collection specified.
- Depending on the query, the message is created with a message for a particular query result.

The message is published to the RabbitMQ exchange based on the query using the basicPublish method of the channel.

In this code snippet it is commented to indicate which part of the code corresponds to each query. However, only the code for the last query (EduCostStatQueryFive) is executed since the code for the other queries is commented out.

2) Programmed the Subscriber / Consumer :

MessageSubscriber.java

```java

import com.rabbitmq.client.*;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

// Run me - fourth
public class MessageSubscriber {
    public static void main(String[] args) throws IOException, TimeoutException {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setUsername("ron");
        factory.setPassword("ron");
        factory.setPort(5672);
        factory.setVirtualHost("/");

        String exchangeName = "EduCostStatExchange";
        //String queueName = "topic1";
        //String topic = "Cost-2013-Alabama-Private-4-year";
        //String queueName = "topic2";
        //String topic = "Top5-Expensive-2013-Private-4-year";
        //String queueName = "topic3";
        //String topic = "Top5-Economic-2013-Private-4-year";
        //String queueName = "topic4";
        //String topic = "Top5-HighestGrow-2021";
        String queueName = "topic5";
        String topic = "AverageExpense-2019-Public In-State-4-year";


        try (Connection connection = factory.newConnection();
             Channel channel = connection.createChannel()) {

            // Declare the exchange
            channel.exchangeDeclare(exchangeName, "topic", true);

            // Declare the queue
            channel.queueDeclare(queueName, true, false, false, null);

            // Bind the queue to the exchange
            channel.queueBind(queueName, exchangeName, topic);

            // Create a consumer
            DeliverCallback deliverCallback = (consumerTag, delivery) -> {
                String message = new String(delivery.getBody(), "UTF-8");
                //System.out.println(" [x] Received '" + delivery.getEnvelope().getRoutingKey()+"':'"+message + "'");
                //System.out.println("Message received and processed successfully.");

            };
            channel.basicConsume(queueName, true, deliverCallback, consumerTag -> { });
            System.out.println("Waiting for messages...");

            /*Consumer consumer = new DefaultConsumer(channel) {
                @Override
                public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte
                    String message = new String(body, "UTF-8");
                    System.out.println("Received message: " + message);
                }
            };

            // Start consuming messages
            channel.basicConsume(queueName, true, consumer);*/
        }
    }
```

The program establishes a connection to a RabbitMQ server using the ConnectionFactory class, creates a channel to communicate with the server, and declares an exchange and a queue. It then binds the queue to the exchange and consumes messages from the queue using a DeliverCallback.

Here is a summary of what the program does:

- Imports the necessary classes from the RabbitMQ Java client library, as well as IOException and TimeoutException from the standard Java library.
- Defines a class called MessageSubscriber with a main method that takes no arguments.
- Creates a ConnectionFactory object and sets the host, username, password, port, and virtual host to connect to a RabbitMQ server running on localhost.
- Defines an exchange name, a queue name, and a topic name.
- Establishes a connection to the RabbitMQ server and creates a channel for communication.
- Declares the exchange using the exchangeDeclare method of the Channel object, specifying the exchange name, type ("topic"), and durability (true).
- Declares the queue using the queueDeclare method of the Channel object, specifying the queue name, durability (true), autoDelete (false), exclusive (false), and additional arguments (null).
- Binds the queue to the exchange using the queueBind method of the Channel object, specifying the queue name, exchange name, and topic name.
- Defines a DeliverCallback that prints the message received from the queue to the console.
- Consumes messages from the queue using the basicConsume method of the Channel object, specifying the queue name, autoAck (true), the DeliverCallback, and a CancelCallback that does nothing.
- Prints "Waiting for messages..." to the console to indicate that the program is waiting for messages.

The commented out code shows an alternative way of consuming messages using a DefaultConsumer object, which implements the Consumer interface and defines a handleDelivery method that is called when a message is received.

Q 1, 2, 3 of assignment -

So, now we can confirm that the producer code retrieves the datasets from each collection from the MongoDB cloud service for each topics listed in the table above.

Now, it is required to program Exchange, Queue and Binding. Below is the demonstration –

<u>Declare an Exchange</u>

```
MessagePubli...   CreateExchan...   CreateQueues...   rabbitmq-pa...   19
/*package com.amqp.exchanges;

package com.amqp.exchanges;

import com.amqp.basic.queue.CommonConfigs;

//Run me first
public class CreateExchange {
  public static void main(String[] args) throws IOException, TimeoutException {

    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    factory.setUsername("ron");
    factory.setPassword("ron");
    factory.setPort(5672);
    factory.setVirtualHost("/");

    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    //Create a topic exchange
    channel.exchangeDeclare("EduCostStatExchange", BuiltinExchangeType.TOPIC, true);

    channel.close();
    connection.close();
  }
}
```

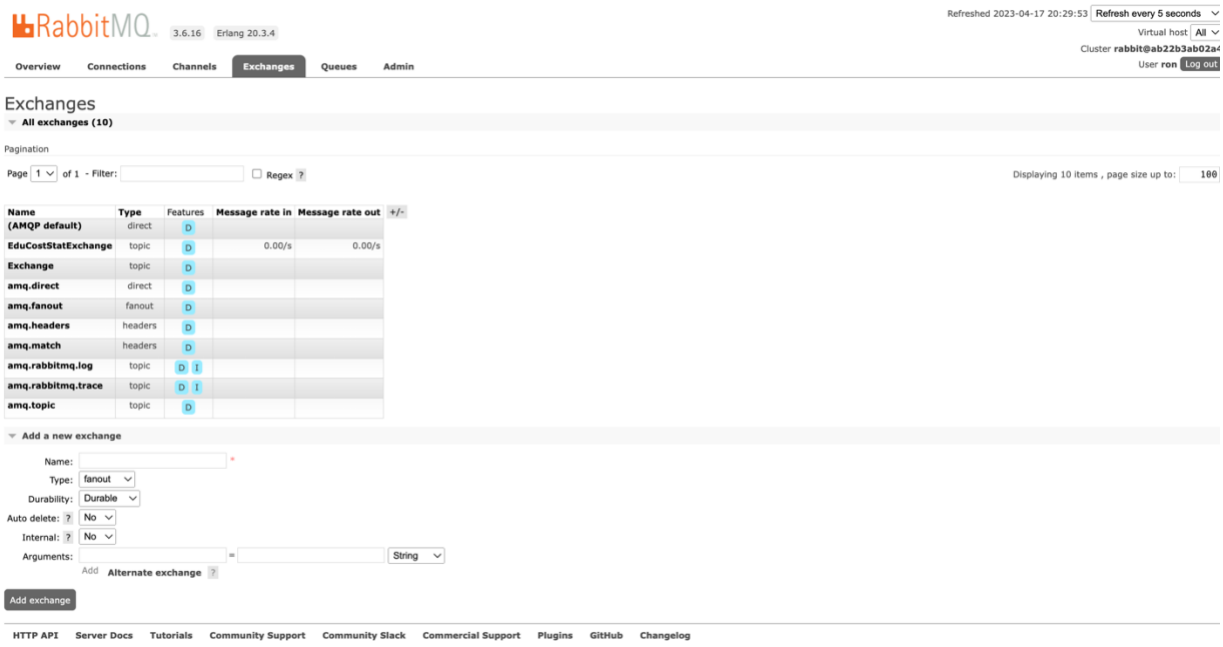
The above code is an example of how to create an exchange in RabbitMQ using the Java client library.

The ConnectionFactory is used to create a new connection to RabbitMQ, and the Connection is used to create a new channel for communication. The Channel is used to create the exchange using the exchangeDeclare() method.

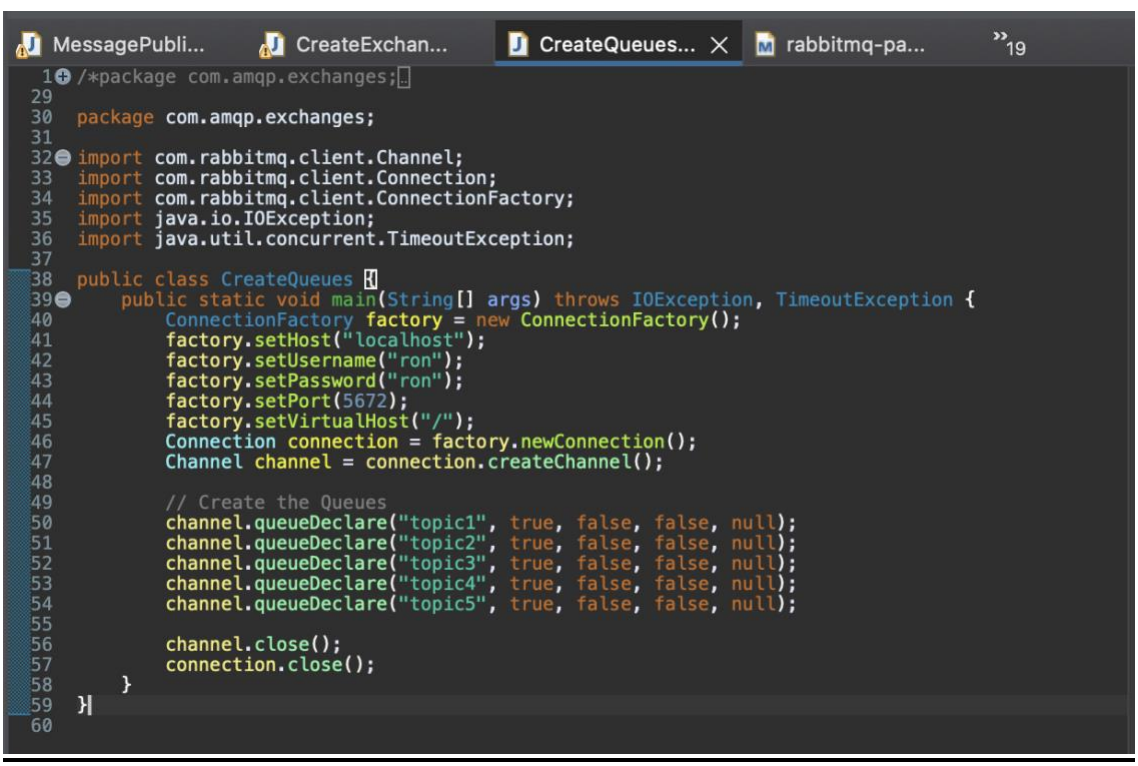
As per the assignment architecture a topic exchange named **EduCostStatExchange** is created with BuiltinExchangeType.TOPIC. The boolean value true specifies that the exchange is durable, which means that it will survive a RabbitMQ broker restart. Finally, the Channel and Connection are closed to free up resources.

This code can be run as a standalone program to create the exchange on the RabbitMQ server. Once the exchange is created, producers can send messages to the exchange and consumers can consume messages from the exchange using the same exchange name.

The exchange – “EduCostStatExchange” can be verified on RabbitMQ server as shown below –



Declare a Queue

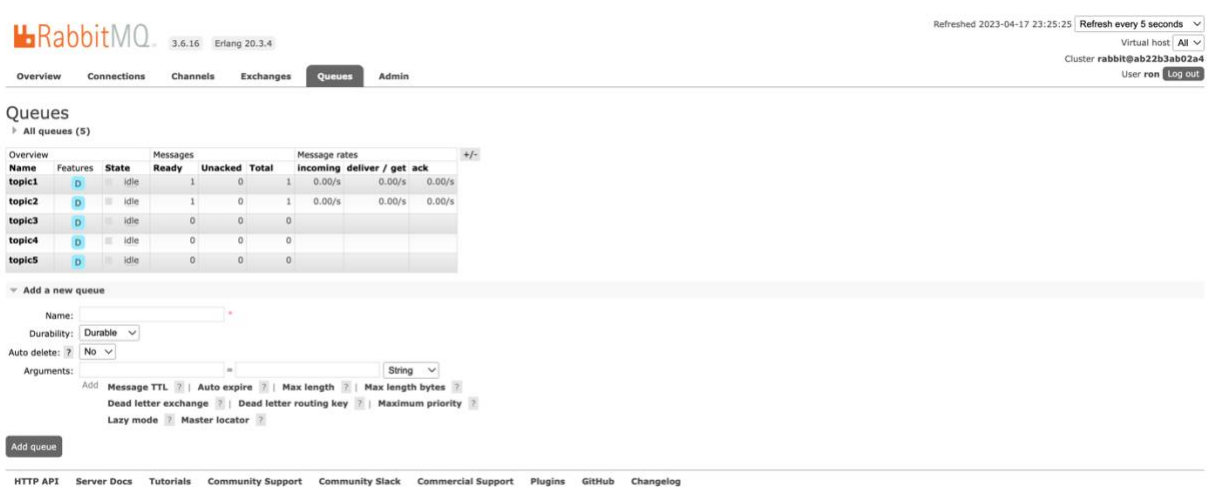


```
MessagePubli...   CreateExchan...   CreateQueues...   rabbitmq-pa...   19
/*package com.amqp.exchanges;...

package com.amqp.exchanges;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import java.io.IOException;
import java.util.concurrent.TimeoutException;

public class CreateQueues {
    public static void main(String[] args) throws IOException, TimeoutException {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setUsername("ron");
        factory.setPassword("ron");
        factory.setPort(5672);
        factory.setVirtualHost("/");
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();

        // Create the Queues
        channel.queueDeclare("topic1", true, false, false, null);
        channel.queueDeclare("topic2", true, false, false, null);
        channel.queueDeclare("topic3", true, false, false, null);
        channel.queueDeclare("topic4", true, false, false, null);
        channel.queueDeclare("topic5", true, false, false, null);

        channel.close();
        connection.close();
    }
}
```

As per the assignment architecture, there are 5 queues declared, with names topic1, topic2, topic3, topic4, topic5 corresponding to 5 topics that are listed –

Collection	Parameters	Topic (* cannot change)
EduCostStatQueryOne	Query the cost given specific year, state, type, length, expense	**Cost-[Year]-[State]-[Type]-[Length]**
EduCostStatQueryTwo	Query the top 5 most expensive states (with overall expense) given a year, type, length	**Top5-Expensive-[Year]-[Type]-[Length]**
EduCostStatQueryThree	Query the top 5 most economic states (with overall expense) given a year, type, length	**Top5-Economic-[Year]-[Type]-[Length]**
EduCostStatQueryFour	Query the top 5 states of the highest growth rate of overall expense given a range of past years, one year, three years and five years (using the latest year as the base) , type and length	**Top5-HighestGrow-[Years]**
EduCostStatQueryFive	Aggregate region's average overall expense for a given year, type and length	**AverageExpense-[Year]-[Type]-[Length]**

The queues can be verified on RabbitMQ server as shown below –



Binding a Queue with an Exchange

Bindings are an essential component of message routing in RabbitMQ. They serve as a means to establish a connection between Exchanges and Queues. It's important to note that messages are never directly published to a Queue in RabbitMQ.

When a producer sends a message to RabbitMQ, it is first routed to an exchange. Exchanges are responsible for routing messages and exist within a virtual host (vhost) in the RabbitMQ Server. They accept messages from the producer application and route them to message queues with the help of header attributes, bindings, and routing keys.

Bindings are essentially links that are set up between a queue and an exchange. They are responsible for establishing the connection between the two entities. Routing keys are message attributes that are used by the exchange to determine which queue to route the message to, based on the exchange type.

Depending on the exchange type, the exchange might examine the routing key of the message and use it to route the message to the appropriate queue.

MessagePubli... CreateExchan... CreateQueues... CreateBindin... 19

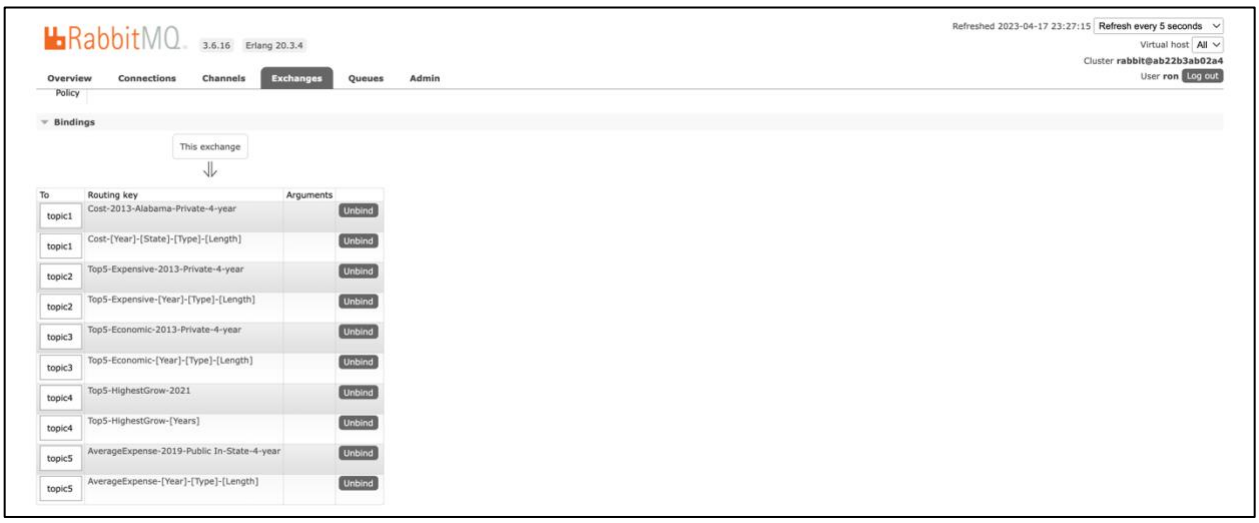
```java
package com.amqp.exchanges;

import com.amqp.basic.queue.CommonConfigs;

//Run me - Third
public class CreateBindings {
        public static void main(String[] args) throws IOException, TimeoutException {
            ConnectionFactory factory = new ConnectionFactory();
            factory.setHost("localhost");
            factory.setUsername("ron");
            factory.setPassword("ron");
            factory.setPort(5672);
            factory.setVirtualHost("/");

            Connection connection = factory.newConnection();
            Channel channel = connection.createChannel();

            // Bind queues to the exchange
        //Create bindings - (queue, exchange, routingKey)
            channel.queueBind("topic1", "EduCostStatExchange", "Cost-[Year]-[State]-[Type]-[Length]");
            channel.queueBind("topic2", "EduCostStatExchange", "Top5-Expensive-[Year]-[Type]-[Length]");
            channel.queueBind("topic3", "EduCostStatExchange", "Top5-Economic-[Year]-[Type]-[Length]");
            channel.queueBind("topic4", "EduCostStatExchange", "Top5-HighestGrow-[Years]");
            channel.queueBind("topic5", "EduCostStatExchange", "AverageExpense-[Year]-[Type]-[Length]");

            channel.close();
            connection.close();
        }
    }
```

Based on the given requirements in assignment, I have used the routing key as the topic (from assignment question table column 3).

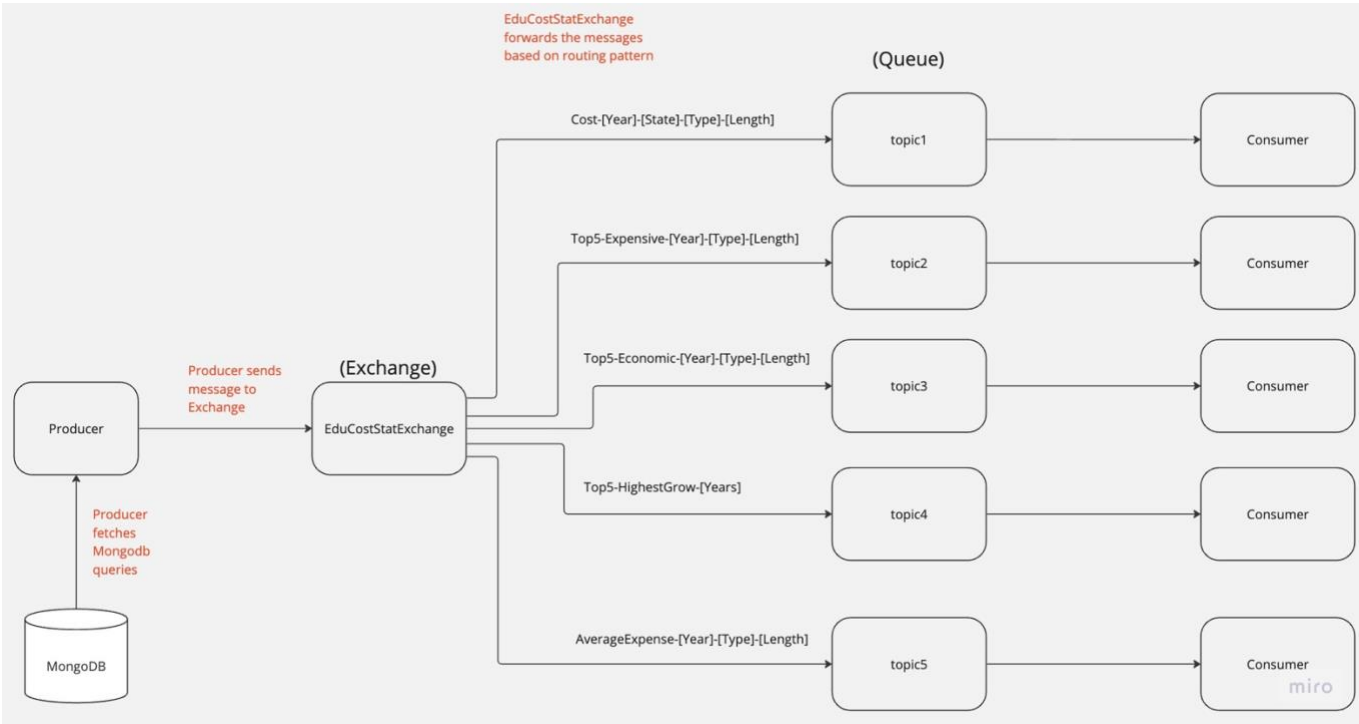
- Queues are >> topic 1,2,3,4,5
- Exchange is >> EduCostStatExchange

- Routing keys are >> Cost-[Year]-[State]-[Type]-[Length] , Top5-Expensive-[Year]-[Type]-[Length] , Top5-Economic-[Year]-[Type]-[Length] , Top5-HighestGrow-[Years], AverageExpense-[Year]-[Type]-[Length]

The bindings can be verified on RabbitMQ server as shown below –

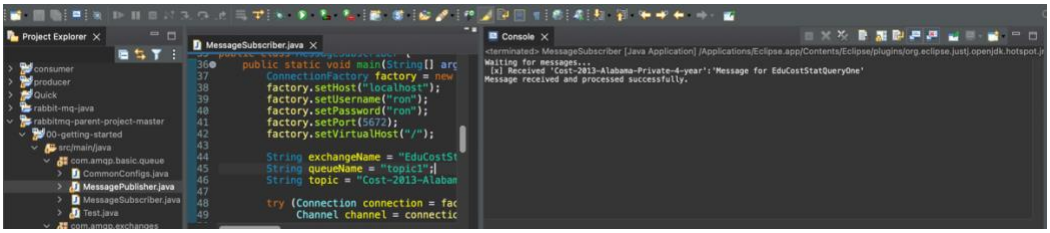


Finally, based on above codes and their workflow, below is the architecture for this code implementation –

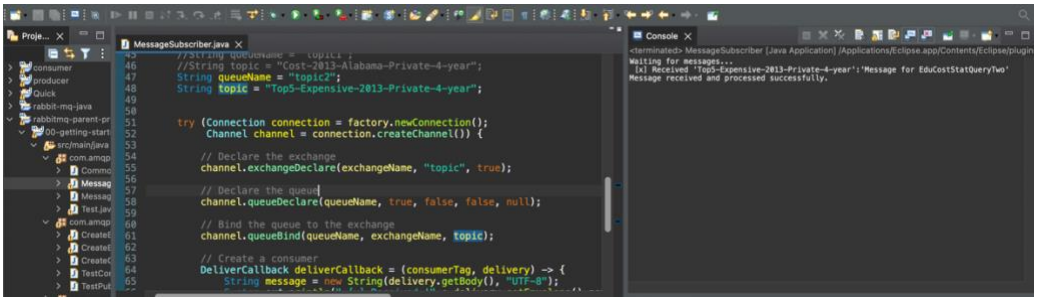


Now, below are the obtained results from the program –

Below is the subscriber console waiting and received message for the topic1 -



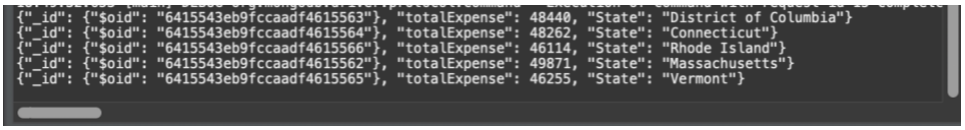
Similarly, this is for subscriber waiting and received message for topic2 –



The same will be obtained for rest of subscriber topics.

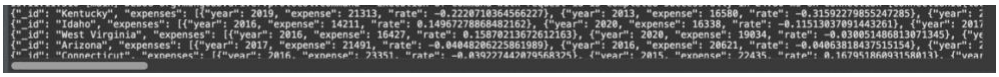
Now below is the output when publisher got data from mongodb and consumer retrieves the data-

For topic 1



For topic 2

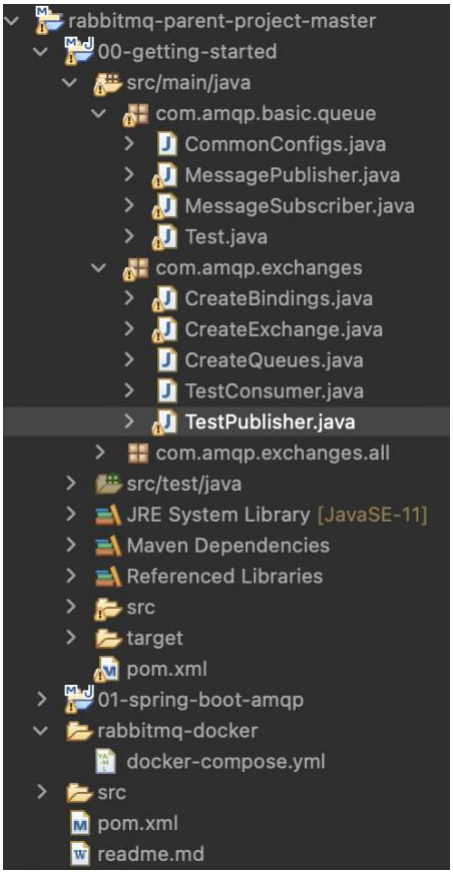
For topic 3



For topic 4

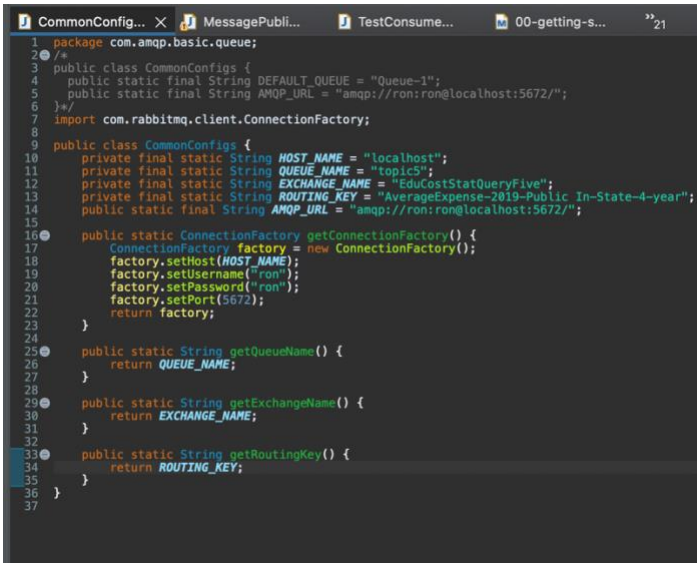
For topic 5

Codes location –



From above, we can conclude that -
The producer retrieves the datasets from each collection from the MongoDB cloud service for each topics listed in the table above. (this is done via MessagePublisher.java)

The parameters to customize each topic is set in a configuration file. –



```java
package com.amqp.basic.queue;
/*
public class CommonConfigs {
  public static final String DEFAULT_QUEUE = "Queue-1";
  public static final String AMQP_URL = "amqp://ron:ron@localhost:5672/";
}*/
import com.rabbitmq.client.ConnectionFactory;

public class CommonConfigs {
    private final static String HOST_NAME = "localhost";
    private final static String QUEUE_NAME = "topic5";
    private final static String EXCHANGE_NAME = "EduCostStatQueryFive";
    private final static String ROUTING_KEY = "AverageExpense-2019-Public In-State-4-year";
    public static final String AMQP_URL = "amqp://ron:ron@localhost:5672/";

    public static ConnectionFactory getConnectionFactory() {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(HOST_NAME);
        factory.setUsername("ron");
        factory.setPassword("ron");
        factory.setPort(5672);
        return factory;
    }

    public static String getQueueName() {
        return QUEUE_NAME;
    }

    public static String getExchangeName() {
        return EXCHANGE_NAME;
    }

    public static String getRoutingKey() {
        return ROUTING_KEY;
    }
}
```

The producer publishes the data to the exchange topics with a routing key that matches to the topic for each queue. (done via MessagePublisher.java , CreateBindings.java, CreateExchange.java, and CreateQueues.java)

The consumer receives the data from the queue based on the topic a consumer subscribed. (done via MessageSubscriber.java).