Cloud Computing - Mini Project Report

Microservice communication with RabbitMQ

Submitted By:
Name – DHANESH MAHTO - PES1UG21CS808
VAIBHAV POKHRIYAL - PES1UG21CS837
VAISHNAVI K - PES1UG21CS838
V POOVARASAN- PES1UG21CS836
VI Semester
Section_K
PES University

Short Description and Scope of the Project

The microservice architecture approach involves developing an application as a collection of small, independent, and loosely coupled services that communicate with each other through APIs or message brokers. This approach provides greater flexibility, scalability, and resilience than traditional monolithic architectures.

This project involves building and deploying a microservices architecture with four components: an HTTP server, a health check endpoint, a service for inserting student records, a service for retrieving student records, and a service for deleting student records. These components will communicate with each other using RabbitMQ, a message broker.

The project also involves creating a Docker network, starting a RabbitMQ container on the network, and connecting to RabbitMQ from the producer and consumers. An HTTP server will be created to listen to requests and distribute them to the respective consumers. The consumers will use RabbitMQ clients to process the requests.

Finally, the application will be Dockerized and Mongo database will be used.

Methodology

To develop a microservices architecture that communicates via RabbitMQ, we need to first define the requirements and architecture, including the functionality of each microservice and how they will interact with each other. Then, we set upa development environment and develop/test each microservice individually.

Next, we manually create a Docker network to host the RabbitMQ image and configure the producers and consumers to connect to the RabbitMQ container. We integrate the microservices by configuring RabbitMQ exchanges and queuesto enable message passing between the services.

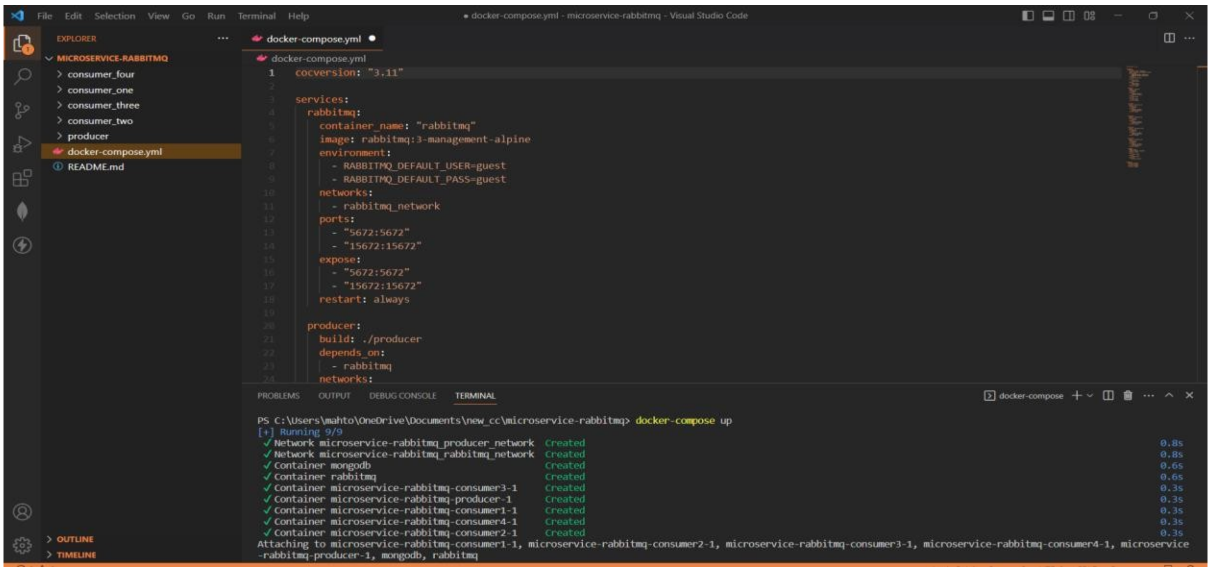
We then Dockerize the producer and consumer microservices and test the overall project using HTTP requests sent to the Flask server. Once it is tested and running smoothly, we deploy the Dockerized application to a production environment, scale the microservices as needed, and monitor and maintain the system to ensureit functions efficiently. Finally, we enhance and update the microservices as required to meet changing needs.

Testing

Postman is a widely used tool for testing APIs and can be used to test the microservices architecture created using RabbitMQ. Once the microservices are up and running, Postman can be used to send HTTP requests to the endpoints exposed by the producer microservice, such as the health_check and insert_recordendpoints.

To test retrieval and deletion of records from the database, GET and DELETE requests can be sent to the corresponding endpoints. Using Postman for testing ensures that the endpoints are functioning correctly and data is being transferredbetween the producer and consumers through RabbitMQ.

docker-compose up

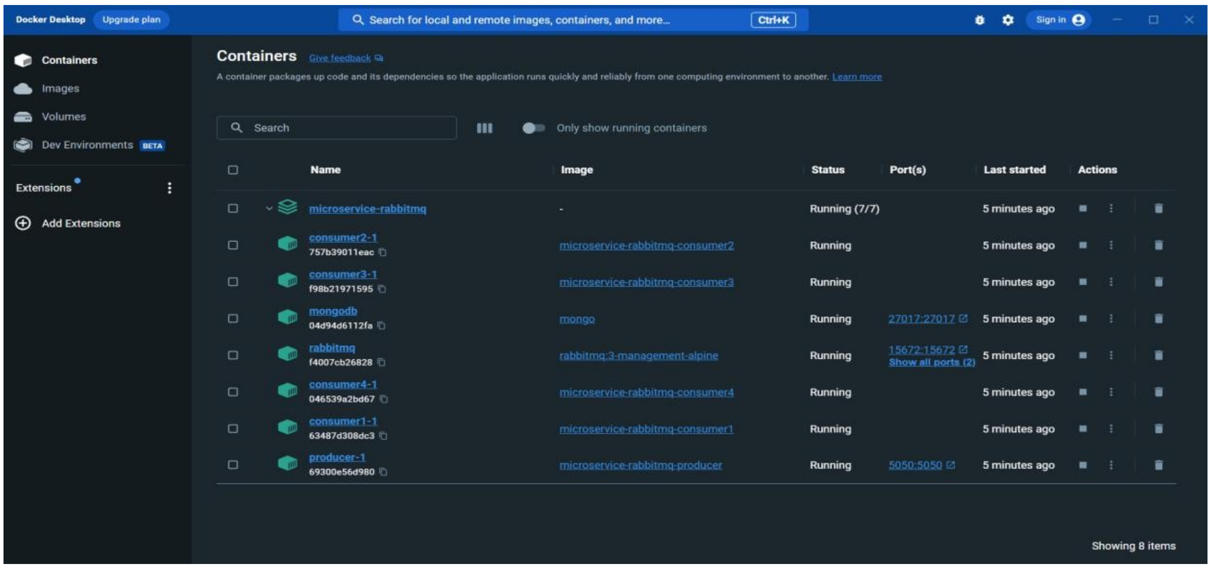


```yaml
cocversion: "3.11"

services:
  rabbitmq:
    container_name: "rabbitmq"
    image: rabbitmq:3-management-alpine
    environment:
      - RABBITMQ_DEFAULT_USER=guest
      - RABBITMQ_DEFAULT_PASS=guest
    networks:
      - rabbitmq_network
    ports:
      - "5672:5672"
      - "15672:15672"
    expose:
      - "5672:5672"
      - "15672:15672"
    restart: always

  producer:
    build: ./producer
    depends_on:
      - rabbitmq
    networks:
```

```
PS C:\Users\mahto\OneDrive\Documents\new_cc\microservice-rabbitmq> docker-compose up
[+] Running 9/9
 ✓ Network microservice-rabbitmq_producer_network  Created                                        0.8s
 ✓ Network microservice-rabbitmq_rabbitmq_network  Created                                        0.8s
 ✓ Container mongodb                               Created                                        0.6s
 ✓ Container rabbitmq                              Created                                        0.6s
 ✓ Container microservice-rabbitmq-consumer3-1     Created                                        0.3s
 ✓ Container microservice-rabbitmq-producer-1      Created                                        0.3s
 ✓ Container microservice-rabbitmq-consumer1-1     Created                                        0.3s
 ✓ Container microservice-rabbitmq-consumer4-1     Created                                        0.3s
 ✓ Container microservice-rabbitmq-consumer2-1     Created                                        0.3s
Attaching to microservice-rabbitmq-consumer1-1, microservice-rabbitmq-consumer2-1, microservice-rabbitmq-consumer3-1, microservice-rabbitmq-consumer4-1, microservice
-rabbitmq-producer-1, mongodb, rabbitmq
```



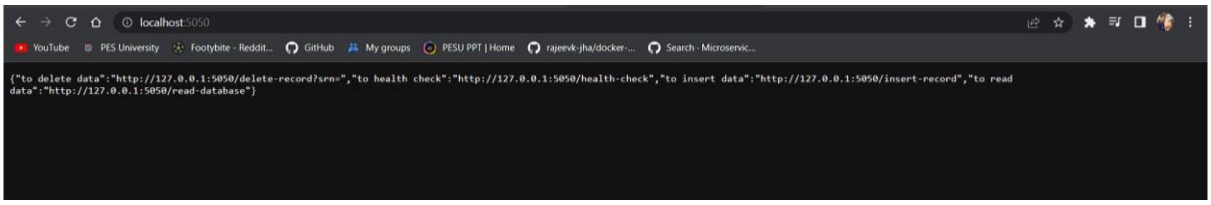
Containers

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Name	Image	Status	Port(s)	Last started
microservice-rabbitmq	-	Running (7/7)		5 minutes ago
consumer2-1 757b39011eac	microservice-rabbitmq-consumer2	Running		5 minutes ago
consumer3-1 f98b21971595	microservice-rabbitmq-consumer3	Running		5 minutes ago
mongodb 04d94d6112fa	mongo	Running	27017:27017	5 minutes ago
rabbitmq f4007cb26828	rabbitmq:3-management-alpine	Running	15672:15672 Show all ports (2)	5 minutes ago
consumer4-1 046539a2bd67	microservice-rabbitmq-consumer4	Running		5 minutes ago
consumer1-1 63487d308dc3	microservice-rabbitmq-consumer1	Running		5 minutes ago
producer-1 69300e56d980	microservice-rabbitmq-producer	Running	5050:5050	5 minutes ago

Showing 8 items

Localhost:5050

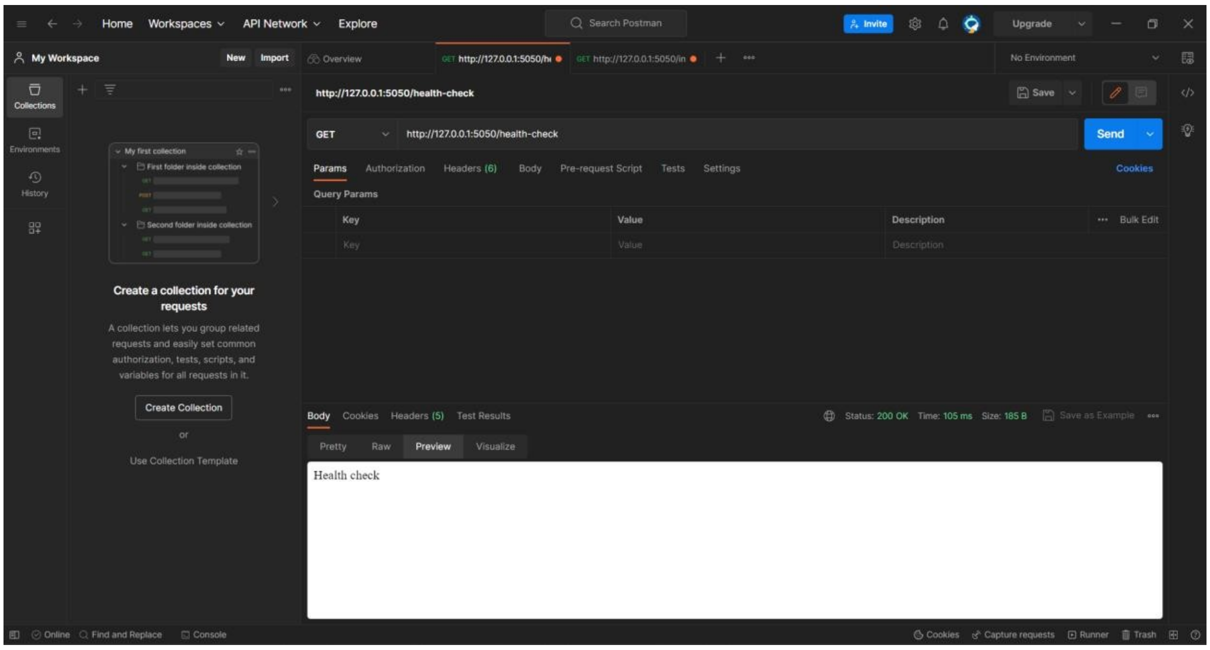


```
{"to delete data":"http://127.0.0.1:5050/delete-record?srn=","to health check":"http://127.0.0.1:5050/health-check","to insert data":"http://127.0.0.1:5050/insert-record","to read data":"http://127.0.0.1:5050/read-database"}
```

HTTP GET REQUEST

Health check

http://127.0.0.1:5050/health-check

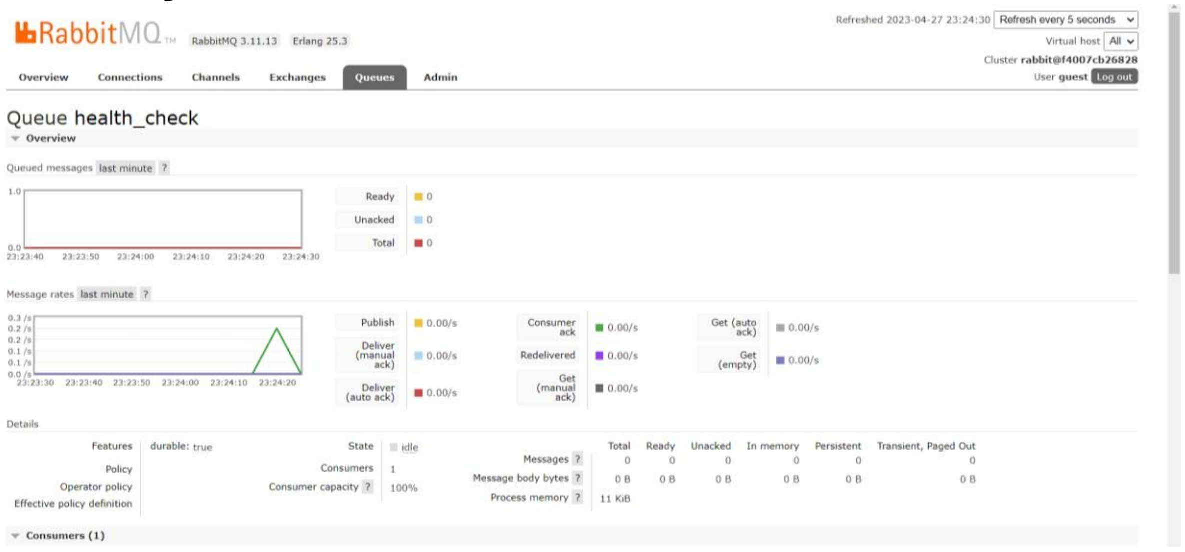


GET http://127.0.0.1:5050/health-check

Status: 200 OK Time: 105 ms Size: 185 B

```
Health check
```

RabbitMQ Console



RabbitMQ 3.11.13 Erlang 25.3

Refreshed 2023-04-27 23:24:30 Refresh every 5 seconds

Virtual host All

Cluster rabbit@f4007cb26828

User guest

Queue health_check

Overview

Queued messages last minute

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last minute

Publish	🟨 0.00/s	Consumer ack	🟩 0.00/s	Get (auto ack)	⬜ 0.00/s
Deliver (manual ack)	🟦 0.00/s	Redelivered	🟪 0.00/s	Get (empty)	🟪 0.00/s
Deliver (auto ack)	🟥 0.00/s	Get (manual ack)	⬛ 0.00/s		

Details

Features	durable: true
Policy	
Operator policy	
Effective policy definition	

State	idle
Consumers	1
Consumer capacity	100%

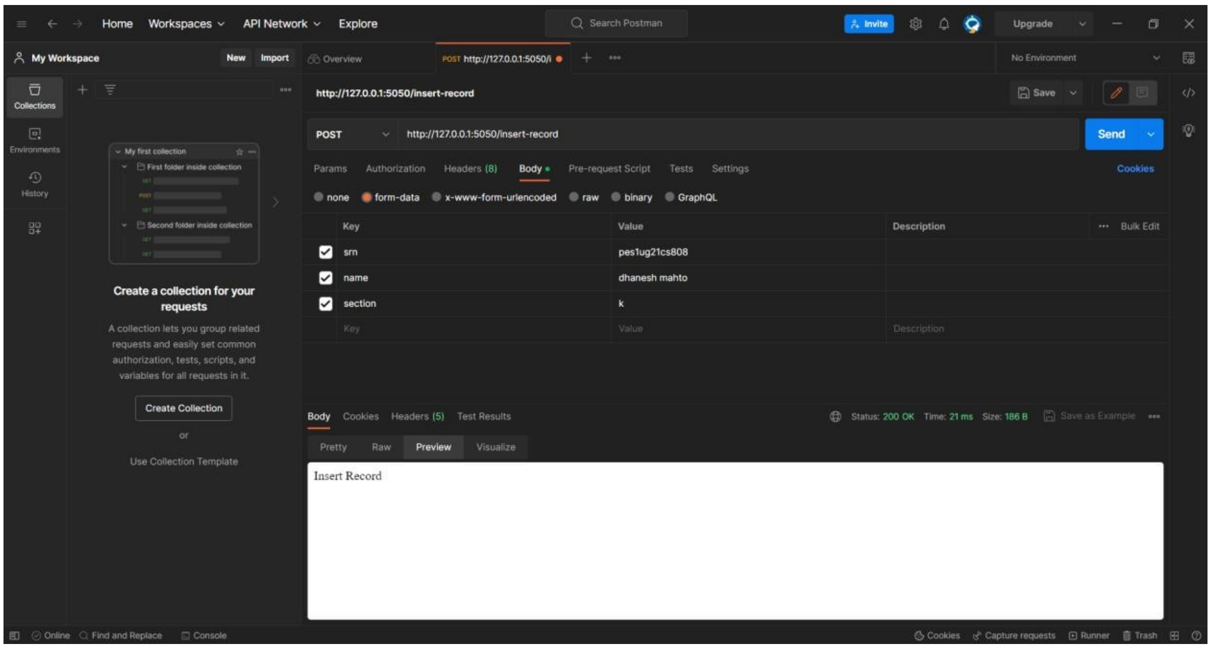
	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages	0	0	0	0	0	0
Message body bytes	0 B	0 B	0 B	0 B	0 B	0 B
Process memory	11 KiB					

Consumers (1)

HTTP POST REQUEST

Insert Record

http://127.0.0.1:5050/insert-record



POST http://127.0.0.1:5050/insert-record

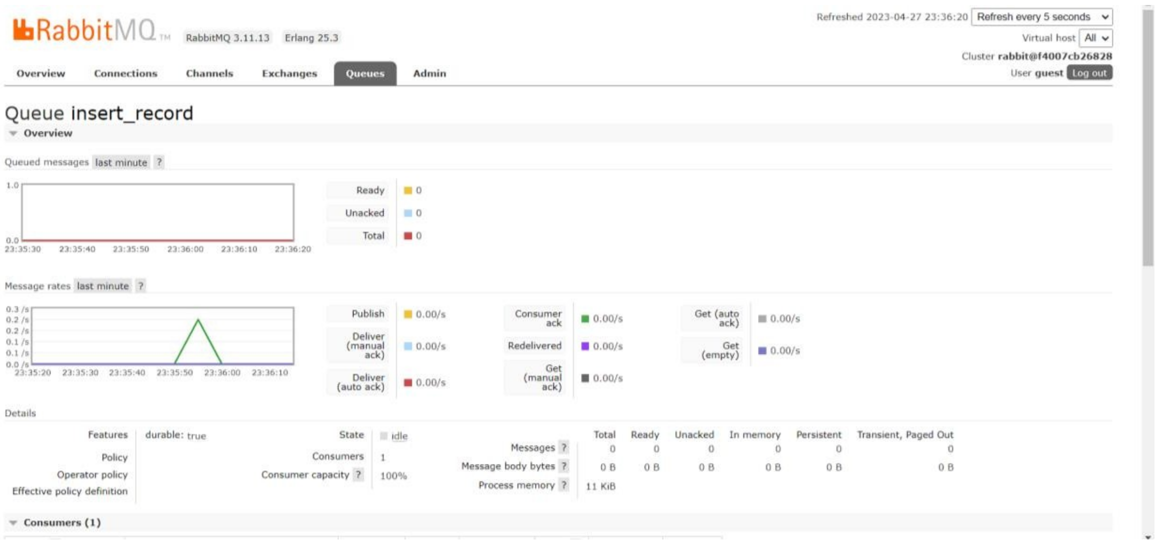
Body: form-data

Key	Value	Description
srn	pes1ug21cs808	
name	dhanesh mahto	
section	k	

Status: 200 OK Time: 21 ms Size: 186 B

```
Insert Record
```

RabbitMQ Console



RabbitMQ 3.11.13 Erlang 25.3

Refreshed 2023-04-27 23:36:20 Refresh every 5 seconds
Virtual host All
Cluster rabbit@f4007cb26828
User guest

Queue insert_record

Overview

Queued messages last minute

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last minute

Publish	🟨 0.00/s	Consumer ack	🟩 0.00/s	Get (auto ack)	⬜ 0.00/s
Deliver (manual ack)	🟦 0.00/s	Redelivered	🟪 0.00/s	Get (empty)	🟪 0.00/s
Deliver (auto ack)	🟥 0.00/s	Get (manual ack)	⬛ 0.00/s		

Details

Features	durable: true
Policy	
Operator policy	
Effective policy definition	

State	idle
Consumers	1
Consumer capacity	100%

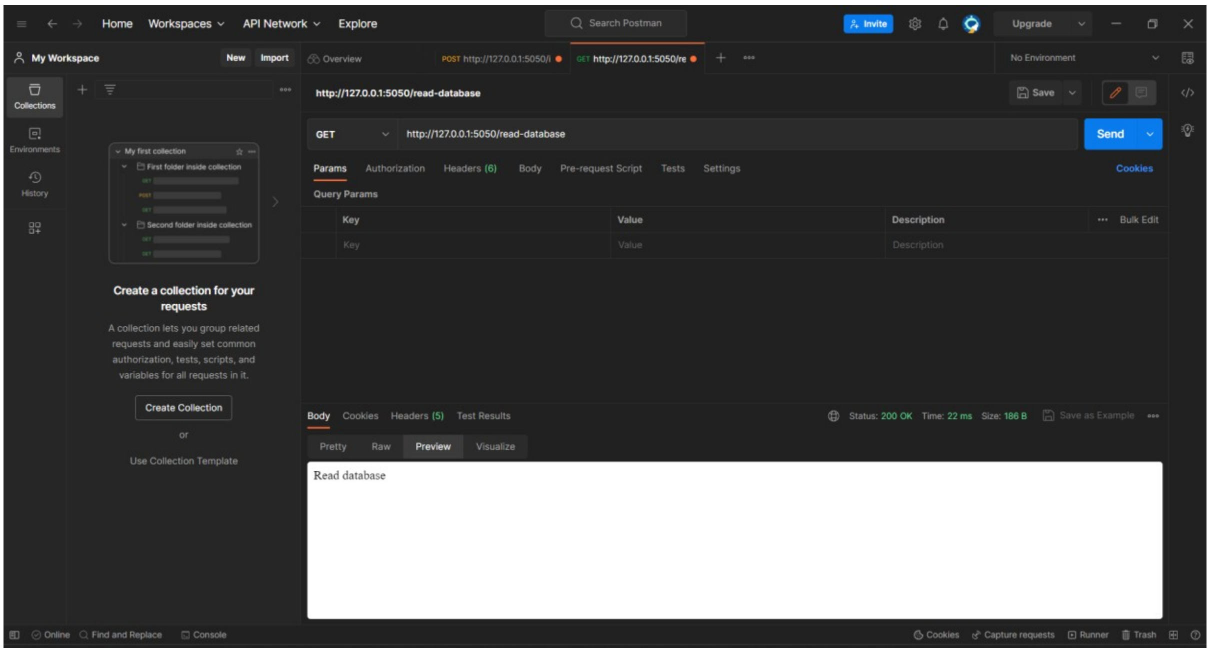
	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages	0	0	0	0	0	0
Message body bytes	0 B	0 B	0 B	0 B	0 B	0 B
Process memory	11 KiB					

Consumers (1)

HTTP GET REQUEST

Read Database

http://127.0.0.1:5050/Read-database

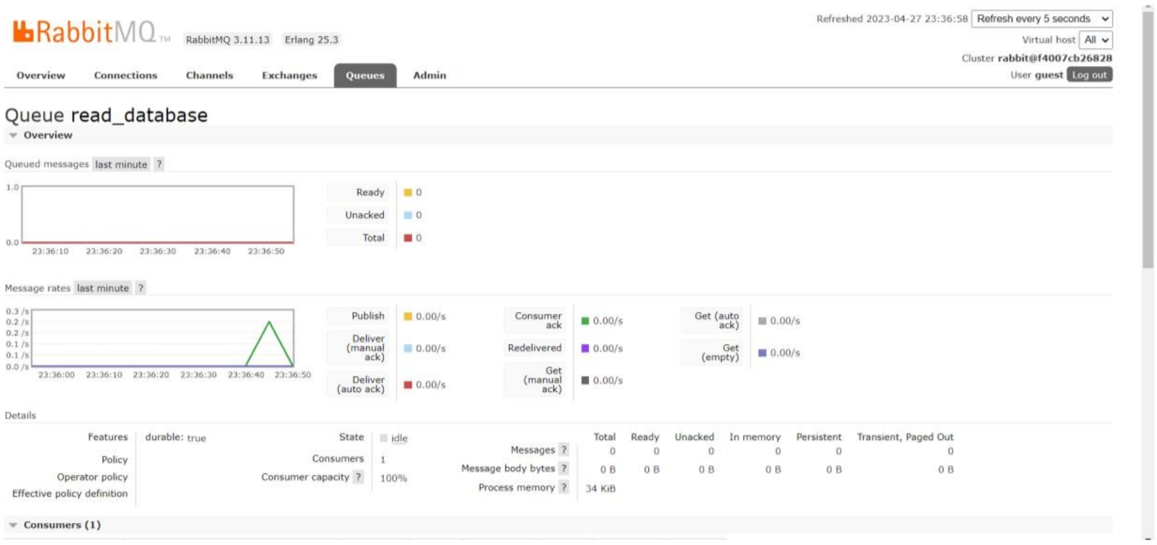


GET http://127.0.0.1:5050/read-database

Status: 200 OK Time: 22 ms Size: 186 B

```
Read database
```

RabbitMQ Console



RabbitMQ 3.11.13 Erlang 25.3

Refreshed 2023-04-27 23:36:58 Refresh every 5 seconds

Virtual host All

Cluster rabbit@f4007cb26828

User guest

Queue read_database

Queued messages last minute

Ready	0
Unacked	0
Total	0

Message rates last minute

Publish	0.00/s	Consumer ack	0.00/s	Get (auto ack)	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Get (empty)	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s		

Details

Features	durable: true
Policy	
Operator policy	
Effective policy definition	

State	idle
Consumers	1
Consumer capacity	100%

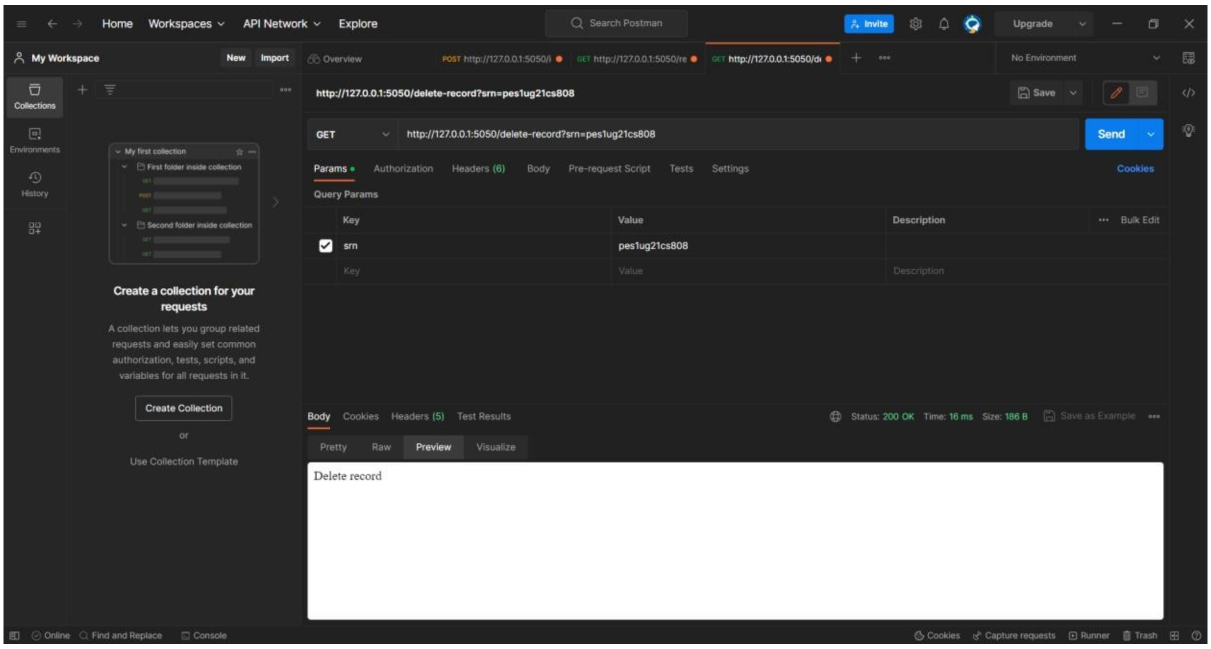
	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages	0	0	0	0	0	0
Message body bytes	0 B	0 B	0 B	0 B	0 B	0 B
Process memory	34 KiB					

Consumers (1)

HTTP GET REQUEST

Delete Record

http://127.0.0.1:5050/delete-record



GET http://127.0.0.1:5050/delete-record?srn=pes1ug21cs808

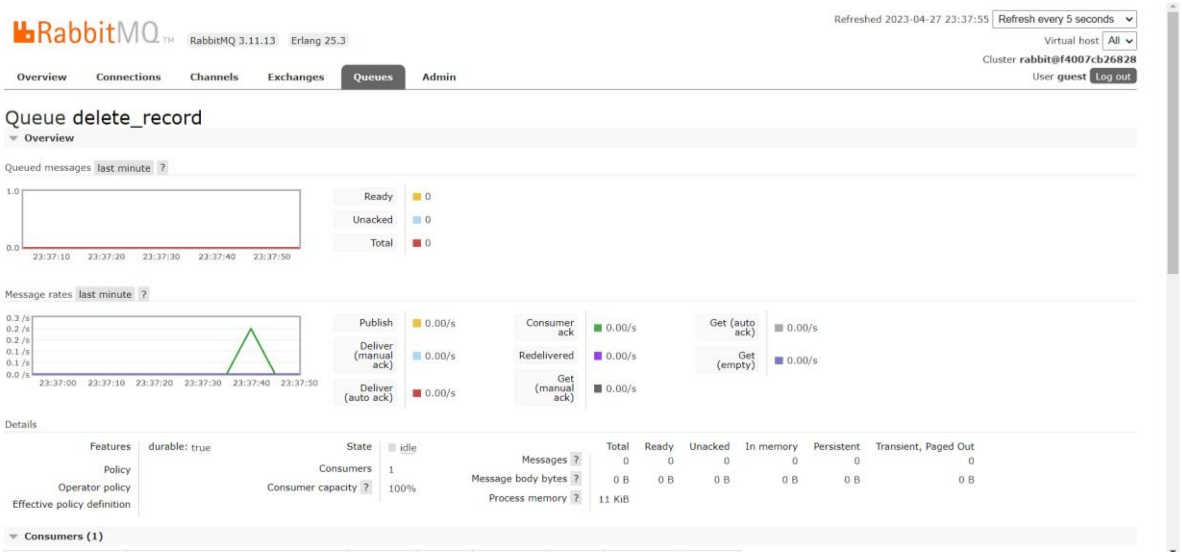
Query Params

Key	Value	Description
srn	pes1ug21cs808	

Status: 200 OK Time: 16 ms Size: 186 B

```
Delete record
```

RabbitMQ Console



Refreshed 2023-04-27 23:37:55 Refresh every 5 seconds

RabbitMQ 3.11.13 Erlang 25.3

Virtual host All

Cluster rabbit@f4007cb26828

User guest

Queue delete_record

Overview

Queued messages last minute

Ready	0
Unacked	0
Total	0

Message rates last minute

Publish	0.00/s	Consumer ack	0.00/s	Get (auto ack)	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Get (empty)	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s		

Details

Features	durable: true
Policy	
Operator policy	
Effective policy definition	

State	idle
Consumers	1
Consumer capacity	100%

	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages	0	0	0	0	0	0
Message body bytes	0 B	0 B	0 B	0 B	0 B	0 B
Process memory	11 KiB					

Consumers (1)

Results and Conclusions

This project successfully implemented a microservices architecture using RabbitMQ as a messaging system to facilitate communication between different components. Four microservices were created to handle CRUD operations on a student management database. The application was containerized using Docker and a docker-compose file was created to run all the microservices and the Mongo database.

Postman was used to test each microservice's API endpoints for functionality and proper communication with the message broker. This project highlights the advantages of microservices architecture in creating complex and scalable systems that can adapt to changing needs over time.

Using RabbitMQ as a messaging system improves system performance and reliability. By following this methodology, developers can create efficient and scalable applications to meet the needs of modern businesses.