

# CloudAMQP

---

## Contents

|  |   |
|--|---|
| Introduction .....                                 | 2 |
| Setting Up the Environment .....                   | 3 |
| Connecting to CloudAMQP .....                      | 3 |
| Step 1: Import the Required Libraries .....        | 3 |
| Step 2: Define RabbitMQ Connection Parameters..... | 3 |
| Step 3: Create a RabbitMQ Connection.....          | 4 |
| Step 4: Publish and Consume Messages.....          | 4 |
| Publisher .....                                    | 4 |
| Consumer .....                                     | 5 |
| Conclusions .....                                  | 5 |

## Introduction

CloudAMQP is a platform to host RabbitMQ servers, RabbitMQ is one of the most known message brokers know in the industry. A message broker is a module that transforms a message from the sender format to the receiver format protocol.

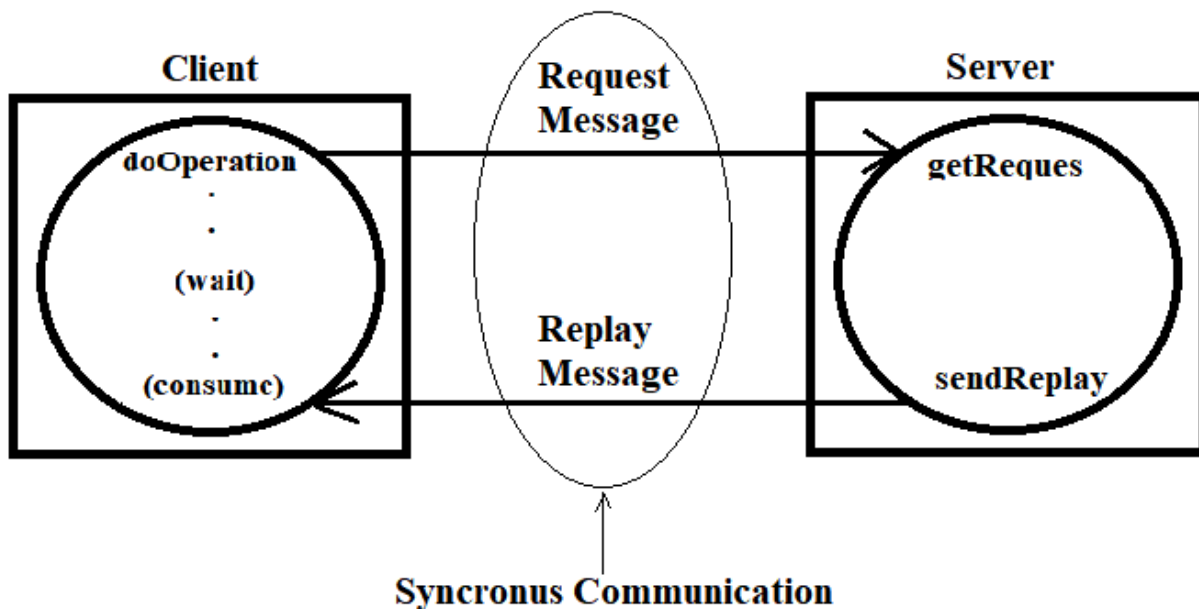
There are several reasons for choosing this platform, which are presented in the following:

- Management of RabbitMQ instance – CloduAMQP is providing a fully monitoring system which is ensures proper operation of each instance deployed on the platform.

- Easy to use – In only three easy to understand steps an instance of a message broker can be deployed and up and running. The three steps mentioned are: creating an account on the platform, creating a RabbitMQ server and completing a setup for the instance.

- Free to use – For a developing a small project, with few messages communication, this platform offers a free to use plan with up to 100 queues at a time and maximum 10 000 queued messages.

A visual exemplification of the system:



## Setting Up the Environment

Before we dive into the details of connecting our project to CloudAMQP, you should have the following prerequisites in place:

- Sign up for a CloudAMQP account.
- Create a RabbitMQ instance in CloudAMQP.
- Obtain the connection details, such as the RabbitMQ server hostname, port, username, password, and other necessary credentials.
- Download and install any code editor.
- Basic knowledge of RabbitMQ core concepts.

For diversity reasons we will implement our Publisher using Python and our Consumer using C#.

## Connecting to CloudAMQP

Now, let's go through the steps to connect your Ballerina application to CloudAMQP's RabbitMQ service.

### Step 1: Import the Required Libraries

```
import pika
```

```
using RabbitMQ.Client;  
using RabbitMQ.Client.Events;
```

### Step 2: Define RabbitMQ Connection Parameters





You'll need to define the RabbitMQ connection parameters using the CloudAMQP credentials obtained from your CloudAMQP dashboard of the created cluster. Replace the placeholders with your specific details.

#### Overview

##### General

|            |  |
|------------|--|
| Region     | azure-arm:eastus2                            |
| Cluster    | turkey.rmq.cloudamqp.com (DNS load balanced) |
| Hosts      | turkey-01.rmq.cloudamqp.com                  |
| Created at | 2024-02-01 22:51 UTC+00:00                   |

##### AMQP details

|              |  |
|--------------|--|
| User & Vhost | jyibxztm   |
| Password     | ***   <a href="#">Rotate password</a>                    |
| Ports        | 5672 (5671 for TLS)  |
| URL          | amqps://jyibxztm:***@turkey.rmq.cloudamqp.com/jyibxztm   |

## Step 3: Create a RabbitMQ Connection

Establish a connection to the RabbitMQ server using the connection configuration defined in the previous step.

```
# CloudAMQP settings
amqp_url = 'amqps://jyitxztm:t9uvBLzfma8VJffsk8JwQVTJ9ULNT0q2@turkey.rm.cloudamqp.com/jyitxztm' # Replace this with your CloudAMQP AMQP URL

params = pika.URLParameters(amqp_url)
params.socket_timeout = 5

# Establish connection to CloudAMQP
connection = pika.BlockingConnection(params)
channel = connection.channel()

# Declare the queue (make sure it matches your CloudAMQP settings)
queue_name = 'soa'
channel.queue_declare(queue=queue_name, durable=True) # durable=True if the queue should survive a broker restart
```

```
0 references
public RabbitMQClientService(IConfiguration configuration)
{
    var factory = new ConnectionFactory() { Uri = new Uri("amqps://jyitxztm:t9uvBLzfma8VJffsk8JwQVTJ9ULNT0q2@turkey.rm.cloudamqp.com/jyitxztm") };
    _connection = factory.CreateConnection();
    _channel = _connection.CreateModel();
    _queueName = "soa";
    _channel.QueueDeclare(queue: _queueName, durable: true, exclusive: false, autoDelete: false, arguments: null);
}
```

## Step 4: Publish and Consume Messages

You can now publish messages to the RabbitMQ exchange and consume messages from queues using the rabbitmq connection created in the previous step.

Implement your business logic for message publishing and consuming here:

### Publisher

```
# Function to publish a message
def publish_message():
    name = choice(names)
    hour = randint(10, 18) # random hour between 10 and 18
    minute = randint(0, 59)
    appointment_time = (datetime.now().replace(hour=hour, minute=minute, second=0, microsecond=0) + timedelta(hours=hour)).strftime('%I:%M %p')
    message = {'name': name, 'appointment_time': appointment_time}
    channel.basic_publish(exchange='', routing_key=queue_name, body=json.dumps(message), properties=pika.BasicProperties(delivery_mode=2)) # make message persistent
    print(f" [x] Sent {name} % message")

# Continuously publish messages with random intervals around 7 seconds
while True:
    publish_message()
    time.sleep(uniform(5, 9)) # Sleep for a random time between 5 to 9 seconds, averaging around 7 seconds
```

## Consumer

```
0 references
protected override Task ExecuteAsync(CancellationToken stoppingToken)
{
    stoppingToken.ThrowIfCancellationRequested();

    _rabbitMQClientService.Consume(async (model, ea) =>
    {
        var body = ea.Body.ToArray();
        var message = Encoding.UTF8.GetString(body);
        await _websocketManager.SendMessageToAllAsync(message);
        Debug.Print($"[x] Sent: {message} \n");
    });

    return Task.CompletedTask;
}
```

## Conclusions

In this guide, we've walked you through the process of connecting the .NET Framework and Python RabbitMQ packages to CloudAMQP, a cloud-based RabbitMQ service. By following these steps, you can seamlessly integrate RabbitMQ into your application and take advantage of CloudAMQP's managed RabbitMQ service.

Message queuing with RabbitMQ is an effective way to enable communication and data exchange between your microservices, and combining it with Ballerina's ease of use and integration capabilities make it a powerful combination for developing cloud-native applications.