Assignment: Integrating RabbitMQ with Spring Boot Microservices using CloudAMQP

Objective: To create a microservice-based system where one service produces messages to RabbitMQ and another service consumes those messages using the CloudAMQP platform.
Scenario: We'll consider a simple e-commerce application. One service, called the **OrderService**, receives new orders and publishes details about the order to a RabbitMQ queue. The other service, named **InventoryService**, listens to that queue, processes the order details, and updates the inventory accordingly.
Tools and Technologies:

1. Java 17+
2. Spring Boot 3.0+
3. Spring AMQP
4. CloudAMQP
5. Maven or Gradle (Based on your preference)

Steps:

1. **Set Up CloudAMQP Instance**:
    - Create an account on https://www.cloudamqp.com/ if you haven't.
    - Set up a new instance and take note of the AMQP URL provided.
2. **Setup the OrderService Microservice**:

a. Initialize a new Spring Boot project using Spring Initializer. Add dependencies: Web, AMQP.
b. In **application.properties**:
spring.rabbitmq.addresses=YOUR_CLOUDAMQP_AMQP_URL
c. Create a simple REST endpoint **/placeOrder** that takes in order details.
d. When an order is placed, convert the order object to a JSON message and send it to RabbitMQ.

```
@Autowired
private RabbitTemplate rabbitTemplate;

@PostMapping("/placeOrder")
public ResponseEntity<String> placeOrder(@RequestBody Order order) {
  // Convert order to JSON
  // Send to RabbitMQ
  rabbitTemplate.convertAndSend("order-queue", order);
  return ResponseEntity.ok("Order placed successfully");
}
```

Setup the InventoryService Microservice:
a. Initialize another Spring Boot project. Add dependencies: Web, AMQP.
b. Configuration is the same as the **OrderService** for RabbitMQ.

c. Create a listener to listen to the **order-queue**:

```
@RabbitListener(queues = "order-queue")
public void processOrder(Order order) {
  // Update inventory based on order
}
```

1. **Testing**:

a. Start both services.

b. Place an order via the **OrderService** by calling the **/placeOrder** endpoint.

c. Check if the **InventoryService** has processed the order and updated the inventory.

2. **Report**:
    - Describe the steps you took to implement the solution.
    - Share any challenges you faced and how you resolved them.
    - Highlight the advantages of using a message broker like RabbitMQ in microservice architectures.
    - Briefly describe how CloudAMQP adds value in terms of scalability and management.

Submission Guidelines:

1. Compress both microservice projects into a **.zip** file.
2. Provide a README.md file explaining how to set up and run both services.
3. Submit your solution before the deadline.

Evaluation Criteria:

1. Functionality: Does the **OrderService** publish messages and does the **InventoryService** consume and process them?
2. Code Quality: Is the code modular, readable, and maintainable?
3. Understanding: Does the report reflect a clear understanding of RabbitMQ, Spring AMQP, and microservice communication?