## **Enunciado do Projeto: Sistema de Geração e Distribuição de Relatórios Assíncronos**

**Objetivo:** Desenvolver um sistema distribuído composto por três microservices que gerenciam a geração e distribuição assíncrona de relatórios, utilizando Apache Kafka, RabbitMQ, **Spring Cloud Stream** e o @Async do Spring. O sistema deve permitir que usuários solicitem relatórios baseados em dados do banco de dados e recebam notificações quando estiverem prontos para download.

**Requisitos Essenciais:**

1. **Comunicação Assíncrona:** Toda a comunicação entre os microservices e a notificação de status de relatório deve ser assíncrona.
2. **Apache Kafka:** Utilizado para a comunicação de eventos de solicitação de relatório e status de geração, **integrado via Spring Cloud Stream**.
3. **RabbitMQ:** Utilizado para a distribuição de relatórios gerados (por exemplo, enviando o link de download), **integrado via Spring Cloud Stream**.
4. **@Async do Java (Spring):** Utilizado para tarefas que podem ser executadas em background dentro de um microservice, como a geração do relatório em si.
5. **Download de Relatórios:** Os relatórios gerados devem ser armazenados e disponibilizados para download. O conteúdo do relatório deve ser baseado em dados do banco de dados.
6. **Estrutura de Microservices:** O sistema deve ser dividido em 3 microservices.
7. **Cobertura de Testes:** Mínimo de 90% de cobertura de testes unitários e de integração.

### **Detalhamento dos Microservices:**

**1. Microservice: report-request-service (API REST)**

* **Responsabilidade:** Receber as solicitações de relatórios dos usuários.
* **Tecnologias:** Spring Boot, Spring Web, **Spring Cloud Stream (com binder Kafka)**.
* **Funcionalidades:**
  + Endpoint REST (POST /reports/request) para receber a solicitação de um novo relatório.
    - A solicitação deve conter parâmetros para filtrar os dados do relatório (ex: dataInicial, dataFinal, tipoRelatorio).
  + Ao receber uma solicitação, deve validar os parâmetros e, se válidos, **publicar um evento no Kafka para o tópico report-requests utilizando o Spring Cloud Stream (via um StreamBridge ou interface Source/Processor configurada)**.
  + Armazenar no banco de dados local um registro da solicitação de relatório com o status inicial (PENDING).
  + Endpoint REST (GET /reports/{id}/status) para o usuário verificar o status de uma solicitação de relatório.
* **Banco de Dados:** Pequeno banco de dados (ex: H2, PostgreSQL) para armazenar os metadados das solicitações de relatório (ID, status, tipo, parâmetros).

**2. Microservice: report-generation-service (Consumer de Kafka, Produtor de Kafka/RabbitMQ)**

* **Responsabilidade:** Consumir as solicitações de relatórios do Kafka, gerar os relatórios e disponibilizá-los.
* **Tecnologias:** Spring Boot, **Spring Cloud Stream (com binder Kafka e RabbitMQ)**.
* **Funcionalidades:**
  + **Consumir mensagens do tópico Kafka report-requests utilizando o Spring Cloud Stream (via anotação @StreamListener ou @InboundChannelAdapter)**.
  + Ao consumir uma solicitação, usar @Async para iniciar a geração do relatório em background.
  + A geração do relatório deve:
    - Consultar um banco de dados (pode ser o mesmo do report-request-service ou um simulado para fins de relatório).
    - Processar os dados e gerar o relatório em um formato específico (ex: CSV, PDF simples). O conteúdo do relatório deve ser dinâmico e baseado nos dados recuperados.
    - Armazenar o relatório gerado em algum lugar acessível (ex: sistema de arquivos local, S3 simulado, ou um endpoint REST de um terceiro serviço de storage).
  + Após a geração bem-sucedida, atualizar o status da solicitação no banco de dados para GENERATED.
  + **Publicar um evento no Kafka para o tópico report-status-updates com o ID da solicitação e o novo status (GENERATED), utilizando o Spring Cloud Stream.**
  + **Publicar uma mensagem no RabbitMQ para uma fila report-downloads contendo o ID da solicitação e o link/caminho para download do relatório gerado, utilizando o Spring Cloud Stream.**

**3. Microservice: notification-service (Consumer de Kafka e RabbitMQ)**

* **Responsabilidade:** Notificar os usuários sobre o status de seus relatórios e disponibilizar links de download.
* **Tecnologias:** Spring Boot, **Spring Cloud Stream (com binder Kafka e RabbitMQ)**.
* **Funcionalidades:**
  + **Consumir mensagens do tópico Kafka report-status-updates utilizando o Spring Cloud Stream.**
    - Ao consumir uma atualização de status para GENERATED, pode, por exemplo, enviar um email simulado ou uma notificação interna para o usuário (pode ser apenas um log indicando a notificação).
  + **Consumir mensagens da fila RabbitMQ report-downloads utilizando o Spring Cloud Stream.**
    - Ao consumir uma mensagem, deve processar o link de download e possivelmente armazenar essa informação em um banco de dados local para que o usuário possa acessá-lo posteriormente.
  + Endpoint REST (GET /reports/{id}/download) que, dado o ID do relatório, redireciona ou serve o arquivo para download. Este endpoint deve interagir com onde o report-generation-service armazenou o arquivo.

### **Cenário de Uso (Fluxo):**

1. Um usuário (via um frontend simulado ou Postman) faz uma requisição POST para report-request-service solicitando um relatório.
2. report-request-service **utiliza Spring Cloud Stream para publicar a solicitação no Kafka** e salva o status PENDING no BD.
3. report-generation-service **utiliza Spring Cloud Stream para consumir a solicitação do Kafka**.
4. report-generation-service usa @Async para gerar o relatório, consultando dados do BD.
5. Após a geração, report-generation-service atualiza o status para GENERATED no BD, **utiliza Spring Cloud Stream para publicar um evento de status no Kafka e também para enviar o link de download via RabbitMQ**.
6. notification-service **utiliza Spring Cloud Stream para consumir o evento de status do Kafka e a mensagem do RabbitMQ**.
7. notification-service "notifica" o usuário e armazena o link de download.
8. O usuário, após ser notificado, pode então fazer uma requisição GET para notification-service (/reports/{id}/download) para baixar o relatório.

### **Requisitos Técnicos Adicionais:**

* **Docker/Docker Compose:** Para facilitar a execução e configuração de Kafka, RabbitMQ e os microservices.
* **Spring Data JPA:** Para interação com banco de dados.
* **Maven/Gradle:** Para gerenciamento de dependências.
* **Spring Cloud Stream:** Essencial para a comunicação com Kafka e RabbitMQ.
* **Testes:**
  + **Unitários:** Testes de lógica de negócio em cada serviço.
  + **De Integração:** Testes que verificam a comunicação entre os serviços (Kafka, RabbitMQ com Spring Cloud Stream), persistência de dados. Usar ferramentas como Testcontainers para facilitar testes de integração com Kafka e RabbitMQ.
  + **Cobertura:** Utilizar JaCoCo para gerar relatórios de cobertura e garantir o mínimo de 90%.

### **Desafios e Pontos de Discussão para o Mentor:**

* **Configuração do Spring Cloud Stream:** Como configurar os binders para Kafka e RabbitMQ, e os channels (entradas e saídas).
* **Serialização/Desserialização:** Como o Spring Cloud Stream lida com a conversão de objetos Java para mensagens no Kafka/RabbitMQ e vice-versa.
* **Grupos de Consumidores:** Como o Spring Cloud Stream gerencia grupos de consumidores no Kafka.
* **Transações Distribuídas (se aplicável):** Embora complexo, pode-se discutir os desafios de transações em sistemas assíncronos e como padrões como o Saga podem ajudar.
* **Idempotência:** Como garantir que as operações sejam idempotentes, especialmente ao consumir mensagens do Kafka/RabbitMQ via Spring Cloud Stream.
* **Tratamento de Erros:** Como lidar com falhas na geração do relatório ou na comunicação entre os serviços, utilizando recursos do Spring Cloud Stream (ex: ErrorChannel).
* **Monitoramento:** Como monitorar o fluxo de mensagens e o status dos serviços, incluindo métricas do Spring Cloud Stream.
* **Consistência Eventual:** Discutir o conceito de consistência eventual e como ela se aplica neste cenário.
* **Escalabilidade:** Como o sistema pode ser escalado horizontalmente?

### **Entregáveis:**

* Código-fonte completo de cada microservice em um repositório Git.
* Arquivo docker-compose.yml para facilitar a execução de todo o ambiente.
* README.md detalhado com instruções para setup e execução do projeto, além de exemplos de requisições.
* Relatório de cobertura de testes (JaCoCo).