Desafio de Desenvolvimento

Você deve desenvolver um microsserviço de validação de ameaças em potencial, com o objetivo de descartar URLs legítimas utilizando uma whitelist de expressões regulares. A whitelist é formada por expressões aplicáveis a clientes específicos e por expressões globais, aplicáveis a todos os clientes. A funcionalidade deve ser disponibilizada através de operações baseadas em mensagens assíncronas.

O microsserviço deve ser desenvolvido em Java e deve ter integração com o message broker RabbitMQ e com o banco de dados MySQL. Os três componentes (microsserviço, RabbitMQ e MySQL) devem ser executados em containers Docker. As expressões regulares devem seguir o padrão Java nativo.

Ambiente

A aplicação deve receber as seguintes variáveis de ambiente (environment variables). Sua utilização é descrita nas seções seguintes. O valor das variáveis deve ser especificado no arquivo **docker-compose.yml**, também descrito adiante.

- INSERTION_QUEUE
- VALIDATION_QUEUE
- NUMBER_OF_VALIDATION_CONSUMERS
- RESPONSE_EXCHANGE
- RESPONSE_ROUTING_KEY
- RABBITMQ_HOST
- RABBITMQ_PORT
- RABBITMQ_VHOST
- RABBITMQ_USERNAME
- RABBITMQ_PASSWORD
- JDBC_URL

Operações

A interação com o microsserviço é feita através de mensagens assíncronas. Não é necessário disponibilizar uma API HTTP.

A aplicação deve fornecer duas operações:

1. Inserção de expressão regular na whitelist

Ao receber uma mensagem de requisição da fila $INSERTION_QUEUE, a aplicação deve inserir na sua base de dados um novo registro.

Formato da mensagem de requisição:

```
{"client": <string/nullable>, "regex": <string>}
```

Uma requisição com o campo "client" nulo (null) deve ser interpretada como uma inserção na whitelist global, aplicável a todos os clientes.

Resposta: nenhuma resposta é necessária.

2. Validação de URL

Ao receber uma mensagem de requisição da fila $VALIDATION_QUEUE, a aplicação deve utilizar a whitelist apropriada para determinar se a URL está na whitelist ou não.

Formato da mensagem de requisição:

```
{"client": <string>, "url": <string>, "correlationId": <integer>}
```

O campo "client" deve ser utilizado para selecionar as expressões regulares aplicáveis. Uma expressão regular da whitelist é aplicável a uma requisição caso tenha sido inserida para o mesmo cliente da requisição ou caso tenha sido inserida na whitelist global.

Resposta: a resposta deve ser enviada para o exchange $RESPONSE_EXCHANGE com a routing key $RESPONSE_ROUTING_KEY.

Formato da mensagem de resposta:

```
{"match": <boolean>, "regex": <string/nullable>, "correlationId": <integer>}
```

O campo "match" da resposta deve ter o valor true caso ao menos uma expressão regular da whitelist seja compatível com a URL fornecida na requisição, e nesse caso o campo "regex" deve conter essa expressão regular; caso nenhuma expressão regular da whitelist seja compatível com a URL, o campo "match" deve ter o valor false, e o campo "regex" deve ser

nulo (null); em caso de múltiplas expressões regulares compatíveis, o campo "regex" deve conter qualquer uma delas.
O campo "correlationId" da resposta deve ter o mesmo valor contido na requisição, sendo relevante apenas para o usuário do microsserviço conseguir correlacionar uma mensagem de requisição com uma mensagem de resposta.

Base de dados

As expressões regulares inseridas através da operação de inserção devem ser persistentes, ou seja, devem resistir a reinicializações da aplicação. Os dados devem ser armazenados em um banco de dados MySQL. O acesso a ele é configurado na aplicação através da variável de ambiente JDBC_URL, incluindo na própria URL o usuário e a senha. Campos tipo string podem ser limitados a 128 caracteres. Durante a inicialização, a aplicação deve garantir a existência das estruturas necessárias (tabelas, índices etc.), criando-as caso não existam.

Mensagens

Deve ser utilizado como message broker o RabbitMQ. O acesso a ele é configurado na aplicação através das variáveis de ambiente RABBITMQ_HOST, RABBITMQ_PORT, RABBITMQ_VHOST, RABBITMQ_USERNAME e RABBITMQ_PASSWORD. Adicionalmente, deve ser possível configurar a quantidade de consumidores simultâneos para a operação de validação, através da variável de ambiente NUMBER_OF_VALIDATION_CONSUMERS. Durante a inicialização, a aplicação deve garantir a existência dos componentes necessários (exchanges, queues, bindings etc.), criando- os caso não existam.

Build e execução

A partir do diretório raiz do projeto, deve ser possível fazer o build e inicializar os três componentes com os seguintes comandos, executados em sequência:

mvn clean install

docker-compose up

A imagem Docker do microsserviço deve ser construída a partir de um Dockerfile, utilizando como base a imagem openjdk:8-jre-alpine. O arquivo docker-compose.yml fornecido deve ser utilizado sem modificações.

Como todos os containers são inicializados simultaneamente, é permitido utilizar *busy waiting* na inicialização do microsserviço para aguardar o final da inicialização do RabbitMQ e do MySQL.

Tecnologias

- Java 8
- Apache Maven
- Spring Framework
- Spring AMQP
- RabbitMQ
- MySQL
- Docker
- Docker Compose

Entrega

Deve ser entregue um arquivo compactado contendo todo o código fonte da aplicação, de acordo com a estrutura abaixo, através do endereço de e-mail maite.roman@axur.com

Diretórios e arquivos

src/

Dockerfile

docker-compose.yml (fornecido)

pom.xml

Avaliação

O resultado será avaliado com base na qualidade interna do código (boas práticas de programação e design), na correção dos resultados e na eficiência (throughput, uso de memória etc.). É esperado que o microsserviço suporte até algumas dezenas de milhares de expressões regulares cadastradas na whitelist.

Caso algum item do enunciado não possa ser cumprido por dificuldades técnicas, incluir no arquivo compactado entregue um arquivo readme.txt justificando o motivo e explicando como a dificuldade foi contornada, bem como instruções detalhadas para build e execução caso não seja possível seguir à risca o procedimento descrito acima.