Projeto Final de Software

Monitoração de arquivos em tempo real usando o Kafka

PUC
RIO

Pedro Felipe Santos Magalhães - 2021592
Orientadora: Noemi Rodriguez
Co-orientadora: Maria Julia

Introdução

No ambiente de pesquisa que estamos inseridos temos uma demanda por monitoração de arquivos de usuários de uma plataforma de execução de simulações. Essas simulações são disparadas através de um portal e a execução ocorre em máquinas do servidor da aplicação. Assim, para que o usuário receba os dados de por exemplo um arquivo de log, é necessário que o servidor monitore esses arquivos.

Neste trabalho tivemos o objetivo de construir um software que pudesse fazer a monitoração de arquivos em tempo real, que chamamos de monitor. O objetivo é que esse monitor execute de forma desacoplada dos consumidores e que o trabalho possa ser feito por mais de uma réplica do monitor. Optamos por usar o Kafka para a comunicação entre os consumidores e monitores porque faz parte do interesse da nossa pesquisa e ele já está presente na arquitetura do software de execução das simulações.

O desenvolvimento do programa foi acompanhado de forma semanal pelas orientadoras e o escopo e progresso eram avaliado a cada iteração. Utilizamos um quadro do miro como documentação viva das reuniões guardando questões levantadas, soluções e requisitos.

Todo o projeto foi implementado em Golang e o código pode ser encontrado em:

https://github.com/Pedro-Magalhaes/projeto-mestrado

A documentação de usuário se encontra no Readme do projeto

1. Especificação do programa

Escopo

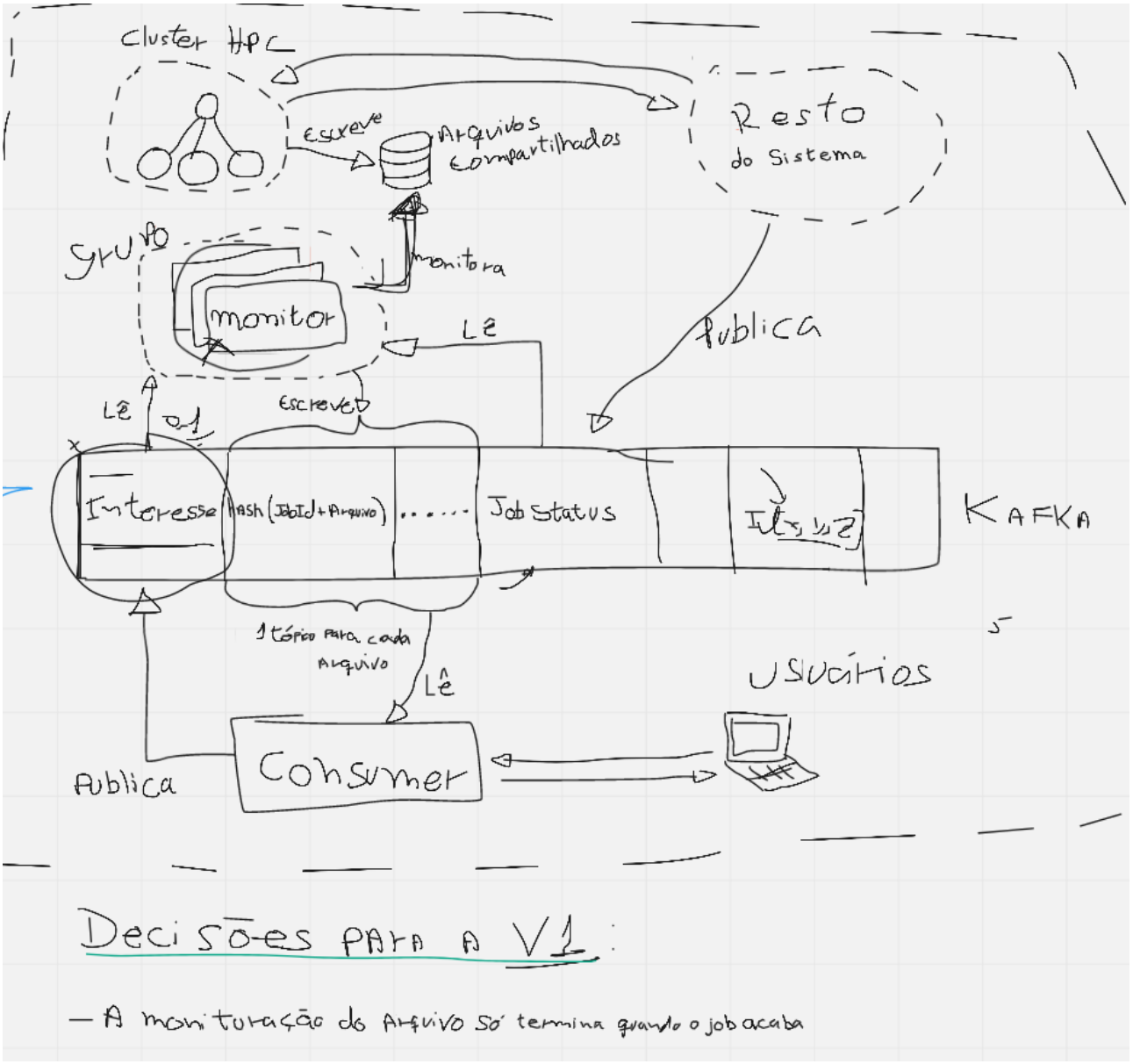
E1: Nesse trabalho decidimos implementar o monitor que apenas observa arquivos e envia o conteúdo dos arquivos para tópicos do kafka.
E2: Cada arquivo vai possuir um tópico diferente.
E3: Existe a possibilidade de uma mesma mensagem ser enviada mais de uma vez em um tópico, no caso de falha do monitor e perda do estado de um determinado arquivo. Caberá ao consumidor verificar o offset da última mensagem que leu e ignorar mensagens com offsets menores

Especificação dos requisitos

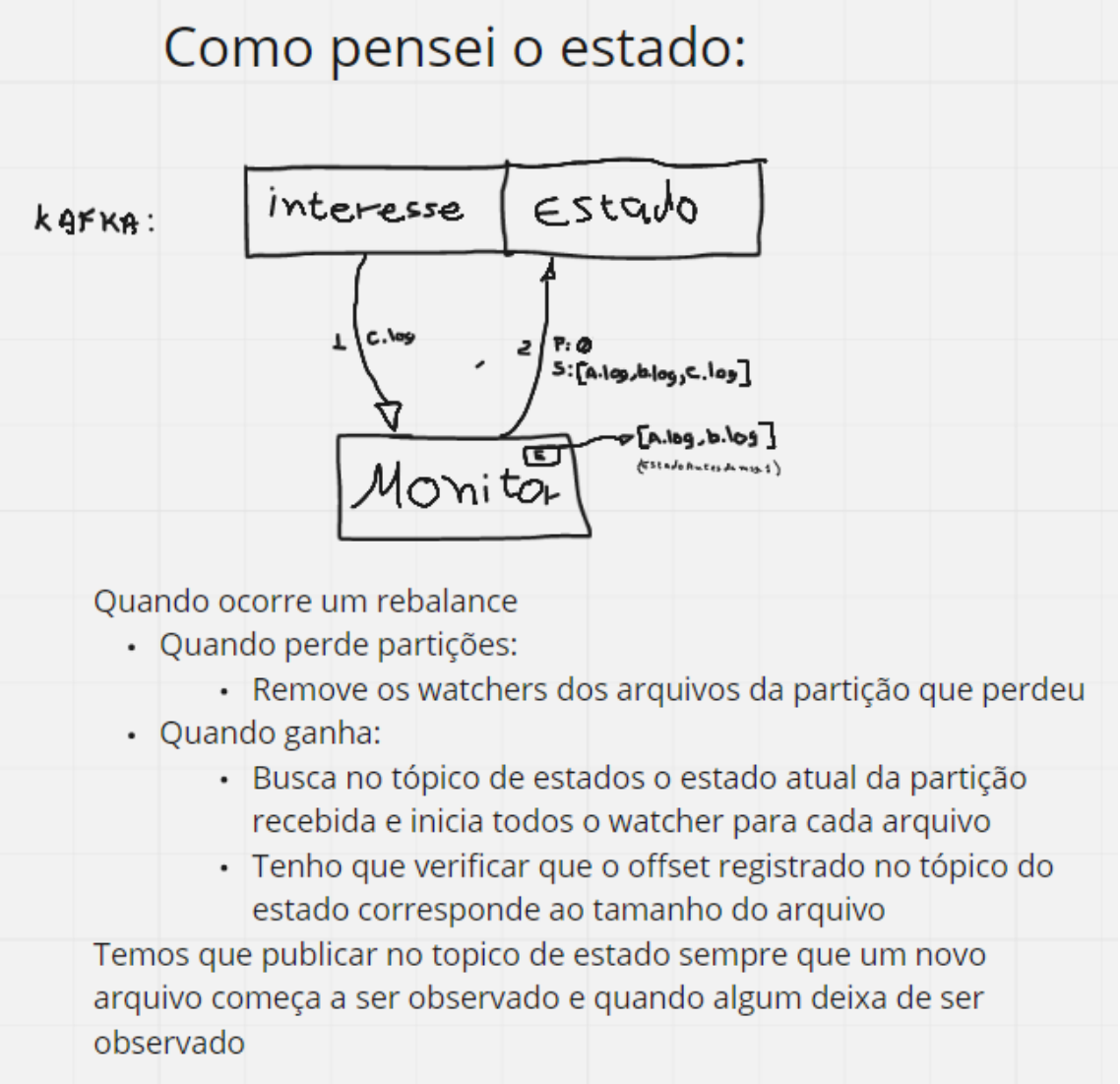
RF1: O monitor deve conseguir receber notificações de escrita nos arquivos e enviar apenas o que foi escrito desde a última vez que o monitor leu o arquivo. Assumimos que os arquivos só vão sofrer “append” e informações não vão ser sobrescritas ou deletadas do arquivo
RF2: O monitor deve conseguir rodar com outras réplicas e dividir o trabalho com elas
RF3: O monitor deve ser capaz de recuperar o estado pro caso de ele ou outro monitor falhar e o trabalho ter que ser reiniciado.
RF4: O monitor não pode enviar mensagens fora de ordem a não ser o caso descrito em E3. Pode ocorrer de termos os seguintes offset de mensagens no tópico: 1,2,3,4,5,4,5,6 mas não: 1,3,2,4,5.

Projeto do programa

1.1 • Arquitetura do sistema em que vai ser inserido



No sistema o “Consumer” (que foi implementado apenas para os testes manuais) fica responsável cadastrar no tópico de interesse os arquivos que ele deseja que sejam monitorados. Em seguida se cadastrar no tópico correspondente de cada arquivo e lá recebe as atualizações do arquivo. Um arquivo deixa de ser monitorado quando recebe uma mensagem de "descadastramento" ou quando o monitor recebe uma mensagem do “sistema” no tópico de “job status” informando que um job acabou. No caso de um job acabar, todos os arquivos vinculados a ele deixam de ser observados.



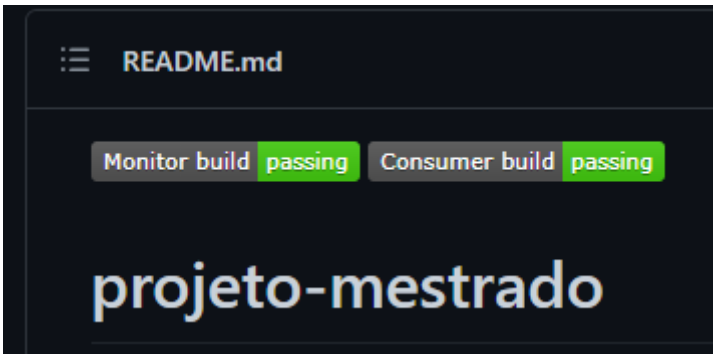
1.2 • diagramas

2. Roteiro de teste efetuado:

2.1 critérios de teste utilizados

Implementamos testes automatizados em algumas classes do monitor, no entanto como Go é uma linguagem que estou conhecendo durante o projeto as classes que possuíam mais dependências não foram testadas automaticamente pela dificuldade de injeção de dependências. Fizemos um

programa auxiliar que implementa um consumidor e uma página web para que os testes manuais fossem feitos. Com o monitor podia acompanhar os arquivos na página web, alterar na mão os arquivos que estavam sendo monitorados e observar se as mudanças estavam corretas.
implementamos no projeto giuhub um cenario de CI/CD em que todo commit gerava um pipeline de build e test e caso algum desses passos falhasse os badges do projeto ficariam vermelhos informando o erro:



2.2 descrição dos casos de teste e resultado de sua execução (laudo de testes)

Exemplo de execução dos testes automatizados:
Exemplo de teste e build quando ocorre push no repositório

Exemplo teste manual através da aplicação do consumer: