Relatório de trabalho no OrchFlow

Angelo Silva [*] Dante Mesquita Neto [†] Jônatas Bonventti [‡] Pedro [§]

2018, v-1.0.0

Resumo

Durante manutenção no orquestrador java OrchFlow, viu-se a necessidade da refatoração em cima da implementação existente. Pontos de mudança com alto acoplamento foram trabalhados para, futuramente, proporcionar mais facilidade na hora de incrementar a aplicação e apliar correção de erros.

Palavras-chave: neo4j. controladores. java. floodlight.

Introdução

Devido a tomadas de decisão imaturas, partes do software acabaram ao longo da primeira implantação do sistema levando um rumo não focado em qualidade, mas sim em simplesmente finalizar a implantação. Através de técnicas simples de desenvolvimento de software e devops, tentamos reescrever pequenos trechos da aplicação, refatorando o sistema.

1 Mudanças implementadas

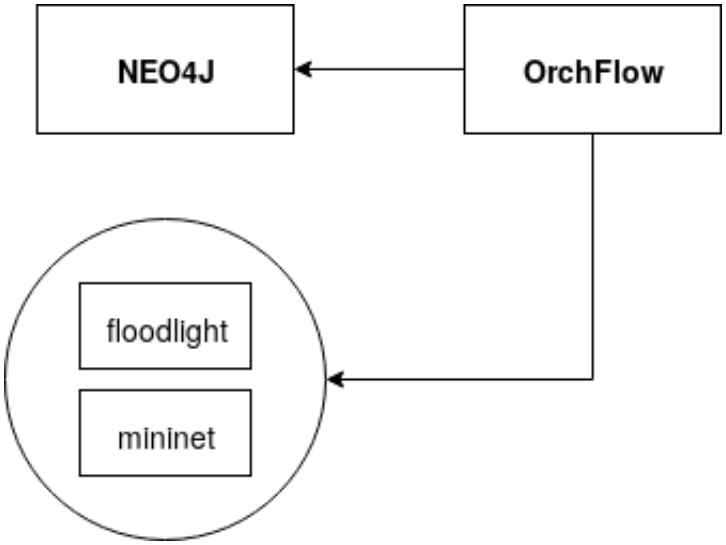
1.1 Docker e contêineres

Para melhor acolhermos os desenvolvedores da plataforma, utilizamos contêineres linux sendo gerenciados por docker e docker-compose para conseguirmos rodar a aplicação, conseguimos controlar partes do software dentro de contêineres mas outras não, floodlight e mininet foram duas dependências que não puderam ser controladas de forma separada a tempo da entrega do projeto.

A arquitetura final dos contêineres ficou como descrito na figura 1.1.

[*] angelorodriigo.rs@gmail.com
[†] dantemesquitaneto@gmail.com
[‡] jonatasbvt@yahoo.com
[§] pedro_vkf@hotmail.com

Figura 1 – Arquitetura dos contêineres



1.1.1 Contêiner neo4j

O contêiner do neo4j fica responsável por manter uma instância do banco de dados, poderíamos trabalhar com um cluster de contêineres, mantendo diversas instâncias sendo controlada pelo docker-compose, aproveitando-se de benefícios como scaling e loadbalance.

1.1.2 Imagem do floodlight com mininet

O contêiner do floodlight com o mininet é onde o problema está, em um cenário de contêineres, cada um deve rodar apenas um serviço linux, porém, durante a definição dos contêineres, foram encontrados problemas de comunicação entre um contêiner floodlight e um mininet, portanto, uma imagem limpa e simples do mininet com floodlight foi mantida, por não fazer parte da aplicação OrchFlow, mas sim, ser um cliente.

1.1.3 Contêiner do OrchFlow

O contêiner do OrchFlow vai manter uma instância da aplicação rodando em uma imagem com base em java e tomcat, se comunicando com o contêiner do neo4j e da imagem do floodlight teste.

1.1.4 Receptibilidade de desenvolvedores

Implementando uma infraestrutura baseada em contêineres conseguimos reduzir consideravelmente a receptibilidade de novos programadores no projeto, como cada serviço fica contido dentro do seu próprio contêiner, configurar um ambiente de desenvolvimento se torna algo até mesmo monótono, não se perde mais tempo tentando entender configurações mágicas que, uma vez não documentadas, são esquecidas.

1.2 Acoplamento de configurações

Um outro problema que foi notado ao iniciar o projeto de manutenção do OrchFlow, que, somado a uma arquitetura baseada em contêineres poderia tornar o desenvolvimento mais rápido, é o acoplamento de configurações, existem chaves de API e configurações

de banco de dados direto no código da aplicação, isso torna a arquitetura OrchFlow um impedimento para novos programadores. Para solucionar o problema, criamos um pacote, centralizando todas as configurações do sistema.

1.3 Padrões de nomeação

Foi trabalhada durante a manutenção do sistema um novo padrão de nomeação, problemas com nomeação de classes e camadas foram solucionados, unificando o idioma para o inglês americano.

1.4 Síndrome das janelas quebradas

Durante o desenvolvimento do projeto, notamos um problema que acontece na maioria dos projetos de software, o projeto começa bem, com uma curva de trabalho e implementação altíssima, porém ao longo do tempo, conforme a base de código cresce, trechos de código ficam para trás no que toca cuidado e boas práticas.

1.4.1 Padrão

Padrões surgiram para trazer abertura para desenvolvedores, assim como a experiência do usuário, temos também a experiência do desenvolvedor ao trabalhar em um projeto, a partir do momento onde cada pessoa que escreve detalhes de implementação, o faz conforme imagina, temos um problema de pai, somente uma pessoa conhece como aquele pedaço de código funciona, por tal, durante a manutenção do projeto, foram discutidos padrões de desenvolvimento de software, desde a arquitetura do sistema até a indentação de parâmetros de um método.

1.4.2 Comentários

Segundo Martin (2008, p. 51),

> Nada por ser tão útil quanto um comentário bem colocado. Nada consegue amontoar um módulo mais do que comentários dogmáticos e supérfluos. Nada pode ser prejudicial quanto um velho comentário mal feito que dissemina mentiras e informações incorretas.

Um comentário em um código tem uma série de problemas, um dos mais identificados dentro da base de código do OrchFlow foi código velho, código que por algum motivo estava sendo trabalhado e foi deixado para trás, isso trás dúvidas quanto os detalhes de implementação.

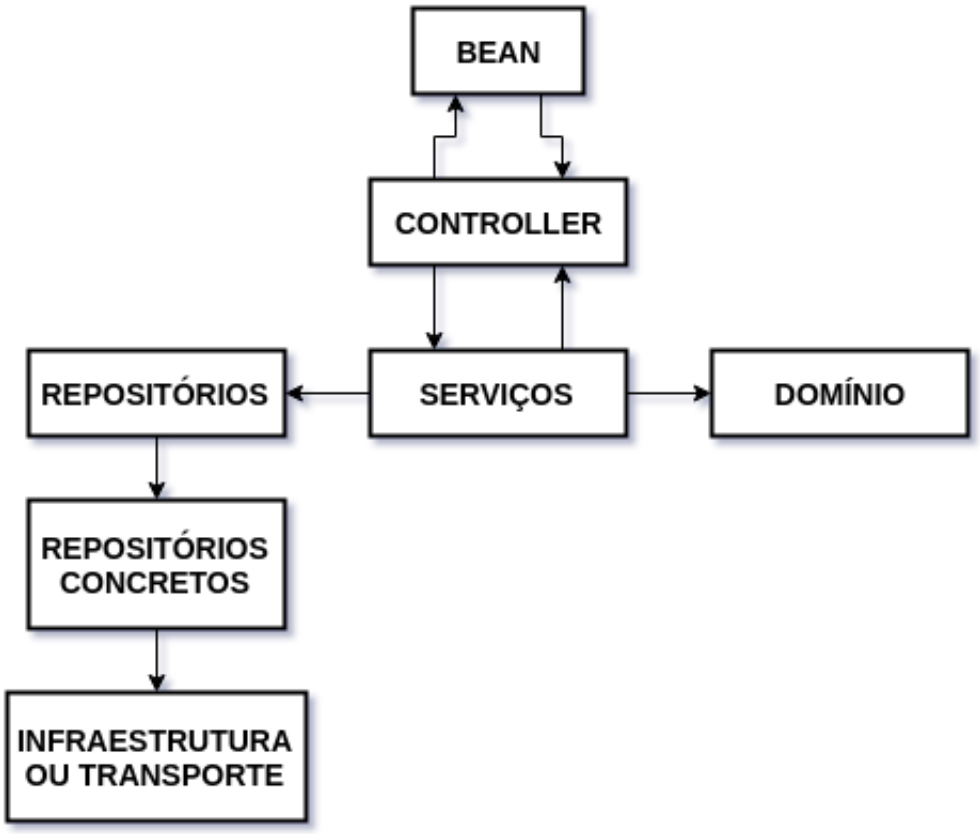
Muitas vezes um código está tão mal escrito, que muitos programadores desejariam um código em cima do comentário o explicando, se é necessário um comentário explicando como aquele código funciona, provavelmente o código não foi capaz de expressão sua intenção através de boa nomeação e estruturação.

2 Mudanças futuras ou em andamento

2.1 Problemas na separação de camadas

A grande dificuldade em manter o projeto está em sua arquitetura, a separação de camadas é feita de forma errônea, se comunicando diretamente com dados através das controllers, com regras de negócio no mesmo lugar. Uma arquitetura que poderia solucionar o problema é separar as camadas e suas responsabilidades como descrito na figura 2.1.

Figura 2 – Arquitetura de camadas



3 Bean

Como não foi pensado sair da arquitetura padrão do OrchFlow, trocando para algum outro framework, foi decidido por manter os beans como um DTO, uma camada de transporte dos dados da aplicação até a View. Na aplicação, os beans do Java simplesmente mantém os dados para serem trabalhados na view, eles se comunicam passando as informações para a controller, simulando uma requisição http.

4 Controle

A camada de controle, trabalha de forma a fazer uma validação básica, trabalhando normalmente em cima de tipos, verificando se a requisição possui todo os dados necessários. Ela se comunica com os serviços da aplicação, requisitando os casos de uso.

5 Domínio

A camada de domínio é a mais importante dentro de uma aplicação, é nela que as regras de negócio estão inseridas, elas são utilizadas também como entidades, sendo passadas e retornadas pelos repositórios.

6 Serviços

Na camada de serviços, são implementados os casos de uso da aplicação, utilizando, podem ser colocadas nessa camada regras de validação de dados, ela se comunica com a camada de domínio da aplicação, chamando as regras de negócio e também com os repositórios na hora de se comunicar externamente.

7 Repositórios

A camada de repositório, se comunica com aplicações externas, é comum encontrarmos nessa camada, chamadas para persistência de dados, porém, podem ser serviços externos diferentes, como um API, um broker, banco de dados, entre diversos outros, é uma boa prática chamar nessa camada um repositório concreto, que possui comunicação com a camada de infraestrutura para o método de persistência desejado.

8 Infraestrutura

A camada de infraestrutura é a camada de comunicação, é nela que ficam as chamadas para os serviços externos da aplicação, a troca da infraestrutura de um usuário de banco de dados para uma chamada em uma API não deve alterar o funcionamento de outra camadas por exemplo. Para se adequar ao OrchFlow, as chamadas a API do neo4j, por exemplo, poderiam ser implementadas aqui.

Referências

MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1. ed. [S.l.]: Prentice Hall PTR, 2008. Citado na página 3.