Integração Sorting Hat/usVision

Proposta de trabalho

Introdução

A Arquitetura de Microsserviços (MSA) é amplamente adotada na engenharia de software. Contrapondo o monolito, ela propõe que o sistema seja dividido em microsserviços disponibilizados independentemente. Isso permite que uma única aplicação seja feita usando diferentes tecnologias em cada microsserviço, como bibliotecas, arcabouços, e até mesmo linguagens de programação. Como consequência, as aplicações que seguem a MSA são de alta complexidade, desafiando compreensão e depuração.

Para auxiliar o desenvolvimento ou até mesmo melhorar o desempenho de um software, é preciso evitar más práticas de arquitetura de microsserviços, avaliando as aplicações frequentemente. Atualmente, existem ferramentas que avaliam ou controlam questões específicas (ferramenta que limita o número de operações públicas de um serviço que utiliza determinado arcabouço, por exemplo). A usVision, por outro lado, foca-se na detecção de padrões e mau-cheiros de MSA, e pode ser utilizada para avaliar aplicações com diversas tecnologias por guiar-se apenas por métricas.

Atualmente, a usVision depende que seu banco de dados já esteja populado para funcionar. Com este trabalho, propõe-se dar suporte à inserção de dados na usVision, bem como estender seu escopo para evolucionabilidade. Como resultado, espera-se um módulo de inserção de dados por meio de uma interface gráfica. Isso porque almeja-se integrar usVision ao SortingHat, uma ferramenta de visualização de arquiteturas distribuídas. Por fim, espera-se que este trabalho contribua como um modelo de integração assíncrona e distribuída de sistemas.

Ferramenta usVision

O usVision é uma ferramenta responsável por modelar a comunicação entre os microsserviços de um sistema de software, que podem ser feitas via diferentes formas (chamadas HTTP ou mensageria) e também dependências externas, como banco de dados. A ferramenta usa essa modelagem para analisar diferentes métricas e com elas detectar padrões e anti-padrões (más práticas de arquitetura de microsserviços) por meio da geração de relatórios configuráveis. Os parágrafos seguintes explicam a estrutura do código a nível de pacotes.

O módulo app-model é responsável pela modelagem de um software que utiliza a arquitetura de microsserviços.

O pacote com.usvision.model.domain contém as interfaces e classes responsáveis pelo domínio da aplicação. Dentro deste pacote, a classe Module representa um agrupamento de microsserviços que precisam ser implementados em conjunto por haver acoplamento entre eles. MessageChannel é responsável por modelar um canal de mensageria (uma instância desta classe pode ser um tópico do Kafka, por exemplo). CompanySystem é responsável por modelar um sistema de software completo.

O pacote com.usvision.model.domain.databases possui a interface Database, que serve para modelar banco de dados. PostgreSQL é uma classe que implementa database. Neste caso, para adicionarmos novos bancos de dados, é preciso criar classes novas que implementam Database (classe MongoDB que implementa Database, por exemplo).

O pacote com.usvision.model.domain.operations permite modelar operações que podem ser feitas nos microsserviços. Nela, temos a interface Operation, que é o contrato de uma operação. Além disso, temos uma implementação dessa interface, o RestEndpoint que representa uma rota para uma chamada HTTP em um microsserviço. É possível criar novas implementações para outros tipos de operações que envolvem outros tipos de comunicação: uma classe chamada gRPC, por exemplo.

O pacote com.usvision.model.systembuilder possui um arquivo com 3 classes: SystemBuilderException, exceção personalizada para um eventual erro durante a construção de um sistema; SystemBuilder, responsável por construir uma instância de CompanySystem e MicroserviceBuilder, responsável por criar uma instância de Microservice. Como se deve imaginar, as classes com final “Builder” usam o padrão builder para construir as instâncias propostas, dada a complexidade da instanciação destes objetos.

O pacote com.usvision.model.systemcomposite possui a interface System, que serve de contrato para as classes cujas instâncias são sistemas. Também possui a interface SystemOfComponents, implementada pela classe Microsservice e implementa a interface System (pois um microsserviço é um sistema de componentes). Por fim, a classe abstrata SystemOfSystem está localizado nesse pacote e implementa a interface System (pois um sitema de sistemas é um sistema). A interface SystemOfSystem é implementada por CompanySystem. O padrão composite é utilizado nesse pacote, ao permitir a criação de objetos que contém outros objetos, seguindo a estrutura de árvore (um sistema de sistemas, segue essa mesma essa estrutura).

O pacote com.usvision.model.visitor possui a interface Visitable, utilizado pela interface System. Além disso, possui a classe abstrata Visitor. Esse pacote serve para que o padrão Visitor seja implementado, tornando as instâncias de System percorríveis pelas instâncias do Visitor.

O módulo app-reports é responsável pelos relatórios fornecidos pela usVision.

No pacote com.usvision.reports há a interface SystemRepository, que pode ser implementada por qualquer classe cuja finalidade seja carregar uma instância de System. No caso do usVision, ele é implementado por uma classe que carrega as informações consultando o banco de dados MongoDB, chamada MongoSystemRepository. Ainda neste mesmo pacote, há a classe ReportSupervisor, responsável por gerar relatórios.

O pacote com.usvision.reports.utils possuem as classes auxiliares que contribuem com a geração de relatórios. Ela possui a classe DetectorsLocator, responsável por localizar as classes que implementam a interface Detector. Além disso, contém a interface ExecutablePlan que possui métodos para planos de análises que são executáveis, que por sua vez é herdada pela interface ExecutablePlan, que possui métodos para planos que são executáveis. Por fim, este pacote possui a classe Plan, que implementa a interface EditablePlan. Este pacote também possuem as classes Report, sendo um relatório que consiste em um mapa cujo nome do padrão detectado é a chave e o valor são as instâncias dos padrões detectados (lista de ArchitectureInsight) e ReportRequest, o qual é uma requisição de relatório que possui uma lista de Detectors.

O pacote com.usvision.reports.executioner possui a interface PlanExecutioner e sua implementação SequentialPlanExecutioner, responsável por executar um plano de análise de sistema, que consiste em uma sequência de execução de Analyzers e Detectors resultando em padrões detectados (ArchitectureInsight).

O pacote com.usvision.reports.planner possui a interface Planer e sua implementação AnalyzerReusePlanner, responsável por criar um plano a partir de um reportRequest: ele cria uma instância dos Detectors do reportRequest, bem como as dependências dos Detectors (os quais são Analyzers).

O pacote com.usvision.reports.exceptions possui classes de exceção personalizadas para cada erro que pode acontecer durante a geração de um relatório.

O pacote com.usvision.analyses é responsável pela análise de um microsserviço.

O pacote com.usvision.analyses.analyzer possui a classe abstrata Analyzer, que implementa a interface Visitor descrito nos parágrafos anteriores e adiciona um método abstrato chamado getResults, responsável por devolver os resultados da análise. Todas as implementações do Analyzer representam alguma propriedade arquitetural que pode ser obtida ao percorrer pelos microsserviços e por isso possuem implementações do método visit (proveniente da interface Visitor). Um exemplo de classe que implementa Analyzer é o NumberOfClients, que representa o número de clientes de um microsserviço.

O pacote com.usvision.analyses.detector possui a classe abstrata Detector, cujas implementações representam padrões de microsserviços a serem detectados: ApiComposition é um exemplo de classe que implementa ela. A interface ArchitectureInsight também está presente neste pacote e permite instanciar o padrão detectado: ApiCompositionInstance é um exemplo de classe que implementa ela. Cada classe Detector possui como parâmetros em seu construtor os Analyzers necessários para a detecção do padrão pelo qual ela é responsável. Após rodar os Detectors através do método run, é possível obter as instâncias de ArchitectureInsight invocando o método getInstances: no caso do Detector ApiComposition, podemos obter um conjunto de instâncias de ApiCompositionInsight ao chamar este método.

Proposta

Atualmente, para carregar o sistema cujo relatório será gerado, o usVision se conecta com um banco de dados MongoDB, buscando um sistema pelo seu nome em uma coleção chamada "systems". É preciso que essa coleção já seja populada com sistemas, pois a ferramenta não possui um módulo de inserção de dados.

Com este trabalho, propõe-se criar um módulo que permita a inserção de dados através da integração com o Sorting Hat, uma ferramenta que coleta as informações dos microsserviços de um sistema por meio de repositórios no GitHub a partir de heurísticas e análises do Docker. Existem diferentes maneiras nos quais a integração desses sistemas podem ser feitas

Depois da análise de um sistema, o Sorting Hat pode publicar essas informações em um canal de mensageria no qual o usVision está inscrito. Assim, o usVision pode ler e armazenar essas informações em seu banco de dados através do seu módulo de inserção a ser criado.

Outra estratégia é utilizar um *Change Data Capture* (CDC), que serve para detectar alterações em um banco de dados e replicar em outro banco de dados. Um exemplo de ferramenta que permite implementar o CDC é o Kafka Connect. Pode-se usar ele para que cada mudança feita no banco de dados do Sorting Hat seja refletida no banco de dados do usVision, mantendo-o sempre atualizado.

Para complementar a integração entre esses sistemas, pretende-se adicionar rotas no usVision para a adição de informações necessárias para a análise, mas não são fornecidas pelo Sorting Hat. Um adicional para esta tarefa é a criação de uma interface para isso, estendendo o frontend do Sorting Hat ou criando um frontend para o usVision.

Por fim, o Sorting Hat possui uma interface que permite a visualização da arquitetura de microsserviços. Pretende-se criar uma interface gráfica para a visualização dos relatórios gerados pelo usVision, fazendo esse sistema ficar completo.

Resultados Esperados

Com este trabalho, espera-se a criação do módulo de cadastro que permite a inserção de dados através da integração com o Sorting Hat, facilitando a geração de relatórios com o usVision. Adicionalmente, espera-se a criação de interface gráfica para o cadastro de informações sobre a arquitetura de microsserviços para tudo que o Sorting Hat não consegue fornecer. Além disso, também é esperado uma interface gráfica para os relatórios gerados pelo usVision.

Através da elaboração deste trabalho, é possível usar aprendizados já adquiridos no curso e no mercado de trabalho e adquirir novos conceitos de boas práticas de código,

manutenção e integração de microsserviços. A usVision possibilita analisar arquitetura de software e entendê-la implica em entender sobre boas práticas de microsserviços. Além disso, o trabalho em si cria a comunicação entre dois serviços e implica em decisões arquiteturais em diversos níveis: nível de banco de dados, microsserviços e a interação entre os módulos em um mesmo microsserviço.