Migração do projeto Spira para ambiente cloud native

Vitor Guidi

Proposta de Trabalho de Conclusão de Curso
apresentado à disciplina
MAC0499
(Trabalho de Formatura Supervisionado)

Orientador: Prof. Dr. Alfredo Goldman
Coorientador: Renato Cordeiro Ferreira

São Paulo, Abril de 2023

Conteúdo

Capítulo 1

Introdução

Com o advento de serviços de hospedagem na nuvem, houve uma revolução na forma de disponibilizar aplicações. No mundo pré-nuvem, era necessário estimar a quantidade de tráfego que uma aplicação receberia, comprar hardware suficiente para tal demanda, preparar um ambiente físico para alocar os servidores, e contratar mão de obra especializada para configurar e manter a infraestrutura.

A nuvem possibilita o provisionamento imediato de infraestrutura por chamadas de API. Não há mais a necessidade de planejar capacidade, pois a nuvem é elástica e se adapta ao tráfego. O capital investido em hardware deixa de ser fixo e dispendido à priori. Ele se torna um custo variável que se ajusta ao consumo, o que facilita a criação de empreendimentos de software com baixo capital inicial.

Para que uma aplicação tire máximo proveito desse novo paradigma de infraestrutura, há uma série de boas práticas da indústria. Aplicações que seguem tais práticas são geralmente chamadas de *cloud native* (em português, nativas à nuvem). Segundo a AWS[1], pioneira da computação em nuvem, aplicações *cloud native* são compostas por microsserviços com as seguintes características: expõem APIs para consumo, são conteinerizadas, e residem numa *service mesh* (em português, malha de serviços).

Em um trabalho de iniciação científica realizado na Universidade de São Paulo [2], foram feitos experimentos com orquestração de contêineres em um *cluster* Raspberry Pi, com o propósito de hospedar uma aplicação de microsserviços num orquestrador de contêineres.

Em (Tamae , 2022) trabalho posterior de conclusão de curso[3], foi criado um conjunto de microserviços de forma a prover uma API de inferência para o projeto SPIRA.

O projeto SPIRA é um projeto de pesquisa da Universidade de São Paulo criado durante a pandemia de COVID-19 para detectar insuficiência respiratória por meio de análise de fala por modelos de aprendizado de máquina.

Os microsserviços do SPIRA foram escritos utilizando a arquitetura hexagonal, seguindo os princípios de aplicações cloud native. Entretanto, foram implantados em uma única máquina física. Não houve tempo hábil para a aplicação ser implantada em um orquestrador de contêineres com redundância, o que traz risco de indisponibilidade.

Este projeto tem como objetivo prover alta disponibilidade para a aplicação SPIRA, consolidando os resultados dos dois trabalhos citados anteriormente. Para tanto, o projeto visa implantar a aplicação no orquestrador de contêineres Kubernetes, possibilitando a existência de réplicas dos serviços em máquinas distintas.

No Capítulo 2 são apresentados os conceitos fundamentais de computação em nuvem e as tecnologias empregadas. No Capítulo 3 são discutidos os passos necessários para prover alta disponibilidade para a aplicação SPIRA. Por fim, no Capítulo 4 é apresentado o planejamento de trabalho e as datas de entrega.

[1] Amazon Web Services: what is cloud native?
[2] Cluster de Kubernetes em Raspberry Pi
[3] Building an Intelligent System to Detect Respiratory Insufficiency

Capítulo 2

Conceitos

O ecossistema cloud native se baseia em algumas abstrações na camada de infraestrutura e de aplicação. Tais abstrações são os blocos básicos de construção de aplicações cloud native.

2.1 Conceitos de computação em nuvem

2.1.1 APIs

Uma API[1], no contexto de microsserviços, é uma interface pela qual um cliente solicita a execução de alguma tarefa computacional para um servidor, por meio de um protocolo de comunicação predefinido.

No contexto do projeto SPIRA, uma API segue o padrão REST[2], pressupõe que a comunicação ocorre pelo protocolo HTTP[3], e os resultados são entregues no formato JSON[4].

2.1.2 Microsserviços

Microsserviços[5] são componentes de uma aplicação responsáveis por um conjunto pequeno de responsabilidades, expondo suas funções por meio de um contrato, geralmente uma API.

Aplicações compostas por microsserviços possibilitam que partes do sistema evoluam e sejam implantadas de forma independente. Consequentemente, o desenvolvimento de software se torna mais ágil, diminuindo a carga cognitiva das equipes de desenvolvimento.

Sendo assim, cada time precisa estar ciente apenas dos contratos com os demais serviços com que interagem, não os detalhes de implementação do sistema inteiro.

2.1.3 Máquinas virtuais

Máquinas virtuais[6] são formas de emular hardware por meio de software. Com o intuito de hospedar uma ou mais aplicações, criam-se máquinas virtuais com sistema operacional dedicado. Tais sistemas operacionais acessam o hardware de forma abstraída, embora as máquinas virtuais compartilhem o mesmo hardware.

2.1.4 Contêineres

Contêineres[7] são uma abstração com origem no projeto Linux. Tal abstração possibilita que aplicações executem compartilhando o mesmo sistema operacional de uma máquina, mas garantindo

[1] What is an API?
[2] Architectural Styles and the Design of Network-based Software Architectures, by Roy Fielding
[3] RFC 2616: Hypertext Transfer Protocol – Http 1.1
[4] The JavaScript Object Notation (JSON) Data Interchange Format – RFC 8259
[5] Microservices, by Chris Richardson
[6] What are virtual machines?
[7] What are containers?

o isolamento de recursos e das dependências de software.

Ao contrário de máquinas virtuais, contêineres distintos compartilham o mesmo kernel. Como não há a replicação do sistema operacional, diminui-se o consumo de recursos.

2.1.5 Orquestradores de contêineres

Orquestradores de contêineres[8] são responsáveis por gerenciar o provisionamento, a comunicação e a elasticidade de aplicações em contêiner.

2.1.6 Mecha de serviços

Service meshes[9] são camadas de infraestrutura que abstraem a comunicação entre serviços.

Sem elas, o desenvolvedor de aplicação é obrigado a conhecer a estrutura da rede. Dessa forma, é obrigado explicitamente escrever código para estabelecer comunicação de um microsserviço com os demais em um sistema.

Ao utilizá-las, a rede se torna transparente, possibilitando que o desenvolvedor se concentre somente na regra de negócio de seus microsserviços.

2.2 Plataforma de execução

Como discutido anteriormente, aplicações cloud native são executadas em orquestradores de contêineres. O presente projeto emprega o Kubernetes para executar essa função, e junto com ele são usadas algumas ferramentas auxiliares.

2.2.1 Kubernetes

O Kubernetes[10] é o orquestrador de contêineres mais amplamente adotado na indústria. Por tal razão, será empregado no presente projeto.

Dado um *cluster* de máquinas, sua principal função é alocar contêineres nas máquinas individuais, abstraindo a comunicação entre os microsserviços. Maiores detalhes podem ser consultados na documentação do projeto[11].

2.2.2 Recursos do Kubernetes

Um recurso é algo a ser gerenciado pelo Kubernetes. Recursos são representados no formato YAML[12]. Por meio desses arquivos, os usuários da ferramenta definem o estado desejado dos recursos.

De forma simplificada, a ferramenta funciona por meio de controladores que executam indefinidamente um laço de reconciliação: observam o estado atual do recurso, comparam com o estado desejado, e tomam as ações necessárias para que o estado desejado se torne realidade.

2.2.3 Helm

Um Helm Chart[13] é uma abstração sobre recursos de Kubernetes. É empregado para abstrair aplicações complexas compostas por múltiplos recursos distintos. Por meio de uma *template engine*[14], permite a modificação de parâmetros da aplicação de forma simplificada. Além disso, permite reutilizar recursos entre diferentes projetos.

[8] What is container orchestration? (Red Hat)
[9] What is service mesh? (Red Hat)
[10] Kubernetes
[11] Kubernetes docs
[12] YAML
[13] Helm
[14] Template engine

2.2.4 Operadores do Kubernetes

Um operador é um ponto de extensão do Kubernetes, que permite o gerenciamento de sistemas além dos recursos padrão suportados.

Os operadores são formados por definições de recurso customizadas [15](em inglês, *Custom Resource Definitions*, CRD), que representam o recurso, e por um controlador capaz de realizar as transições de estado no laço de reconciliação.

No presente projeto, os operadores serão usados para implementar as dependências da aplicação SPIRA dentro do Kubernetes.

2.2.5 Kind

O Kind[16] é uma ferramenta que permite a execução de um cluster Kubernetes com múltiplas máquinas em ambiente de desenvolvimento local.

2.3 Plataforma de dados

A plataforma de dados de aplicações cloud native tem como objetivo a persistência de estado e a comunicação entre os microsserviços. No presente projeto escolhemos as seguintes tecnologias, que podem ser executadas dentro do próprio Kubernetes.

2.3.1 MongoDB

O MongoDB[17] é um banco de dados orientado a documentos. É utilizado no SPIRA para armazenar os dados do microsserviços da aplicação.

2.3.2 MLFlow

O MLFlow[18] é utilizado no SPIRA para versionamento e armazenamento de modelos de inferência. É uma das dependências a ser provisionada para a aplicação.

2.3.3 MySQL

O MySQL[19] é um banco de dados relacional. É utilizado para armazenar metadados do MLFlow.

2.3.4 MINIO

O MINIO[20] é um sistema de armazenamento de arquivos, desenvolvido para ter uma API idêntica ao Amazon S3[21]. É uma dependência da aplicação SPIRA, utilizada para armazenar gravações dos pacientes que serão submetidas para análise e usada pelo MLFlow para armazenar modelos de aprendizado de máquina.

2.3.5 NATS

O NATS[22] é um sistema e comunicação assíncrona. É utilizada pela aplicação SPIRA na comunicação entre seus microsserviços.

[15] Custom Resource Definitions
[16] KIND
[17] MongoDB
[18] MLFlow
[19] MySQL
[20] MINIO
[21] S3
[22] NATS

Capítulo 3

Proposta

No estado atual, o SPIRA não é provisionada em um orquestrador de contêineres. Cada componente é disponibilizado com apenas uma réplica, implicando na indisponibilidade da aplicação caso um dos componentes deixe de funcionar.

Para prover alta disponibilidade para o sistema, é necessário mover seus microsserviços e dependências para o Kubernetes, em duas fases:

- Para implantar as dependências no Kubernetes, será necessário pesquisar as alternativas disponíveis entre Helm Charts e operadores. Uma vez definida a forma de prover cada dependência, será necessário implementá-las num cluster Kubernetes. O critério de sucesso da implantação das dependências consiste em realizar chamadas com sucesso para a API de cada um dos componentes.

- Para implantar os microsserviços no Kubernetes, será necessário adaptá-los para execução com múltiplas réplicas. O critério de sucesso da implementação será a execução bem sucedida dos testes de integração. Por fim, serão executados os fluxos de usuário da aplicação.

Durante o projeto, a proposta será utilizar o KIND como ferramenta para provisionar o cluster Kubernetes. Após a execução das etapas descritas acima, a aplicação terá alta disponibilidade, consolidando os resultados dos projetos anteriores.

Capítulo 4

Cronograma

O projeto será dividido nas seguintes fases, a serem executadas ao longo do período previsto pela disciplina de Trabalho de Formatura Supervisionado:

1. Levantar as melhores práticas de implantação das dependências do SPIRA, incluindo:
   (a) escolher uma metodologia de implantação para o MongoDB,
   (b) escolher uma metodologia de implantação para o MySQL,
   (c) escolher uma metodologia de implantação para o NATS,
   (d) escolher uma metodologia de implantação para o MINIO, e
   (e) escolher uma metodologia de implantação para o MLFlow.
2. Implantar as dependências do SPIRA no KIND, incluindo:
   (a) aplicar as metodologias da primeira etapa num cluster Kubernetes usando KIND; e
   (b) verificar que todas as dependências implantadas são acessíveis por meio de um contêiner de teste dentro do KIND.
3. Implantar os microsserviços do SPIRA no KIND, incluindo:
   (a) criar os recursos Kubernetes para cada um dos microsserviços do SPIRA no ambiente KIND, suportando execução de múltiplas réplicas; e
   (b) verificar que cada microsserviço consegue acessar dependências por meio dos testes de integração já implementados.
4. Testar os fluxos da aplicação, incluindo:
   (a) verificar que todos os casos de uso do SPIRA funcionam no ambiente Kubernetes, e
   (b) realizar alterações pontuais no sistema para possibilitar a integração entre os serviços, caso necessário.
5. Escrever a Monografia, incluindo:
   (a) documentar decisões arquiteturais e dificuldades enfrentadas ao longo do projeto, e
   (b) discutir resultados alcançados, em particular como eles possibilitaram garantir a alta disponibilidade do SPIRA.

Tarefa	30/05	30/06	30/08	30/10	30/11
1	x				
2		x			
3			x		
4				x	
5					x

Bibliografia

Tamae (2022) Victor Tamae. Building an intelligent system to detect respiratory insufficiency, 2022. URL https://daitamae.github.io/MAC0499/Monograph.pdf. Citado na pág. 1