Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia Departamento de Computação

DEVOPS

Chaos Game

Professor Delano Beder

Fernando Kiyoshi Kaida 769667

Engenharia de Computação

1. Introdução

Este trabalho foi feito com o intuito de converter uma aplicação em imagens para serem executadas dentro de containers no Docker, que serão orquestrados por Docker Compose, e analogamente com minikube e Helm.

O jogo do caos é um método iterativo de gerar fractais. A partir de um ponto inicial e um polígono em um plano, seleciona-se um vértice aleatóriamente e então calcula-se um novo ponto, cuja distância até o ponto inicial é proporcional à deste com o vértice. Repetindo o algoritmo, um fractal será gerado. Manipulando as regras, como restrição no processo de seleção de vértices, é possível gerar novas imagens. Então foi desenvolvido uma linguagem com o objetivo de facilitar o processo.

2. Requisitos

A aplicação foi desenvolvida em um Ubuntu 20.04.6 com Docker v26.1.3, Docker Compose 2.36.2, minikube v1.36.0 e Helm v3.18.4. É necessário que seja incluído a seguinte entrada no arquivo /etc/hosts: '192.168.49.2 k8s.local'.

3. Guia de instalação.

Para executar a aplicação utilizando Docker e Docker Compose, deve-se executar o comando **'docker compose up'** no terminal na raiz do projeto e acessar **'localhost:80'** em um navegador. Para encerrar, deve-se executar **'docker compose down –rmi all -v'** para deletar as imagens e os volumes.

Para executar com o minikube e Helm, foi escrito feito um script **helm.sh** que deve ser executado na raiz do projeto. Ele possui quatro comandos: **'bash helm.sh setup'**, construir as imagens e carregá-las no minikube e montar uma pasta do projeto contendo exemplos prontos no minikube; **'bash helm.sh start'**, que irá inicializar abrir um dashboard mostrando o estado da aplicação, que poderá ser acessada em **'k8s.local'**; **'bash helm.sh stop'**, que irá finalizar a aplicação; e **'bash helm.sh remove'**, que irá deletar as imagens do minikube e na máquina local, e encerrará o minikube.

4. Guia rápido de uso

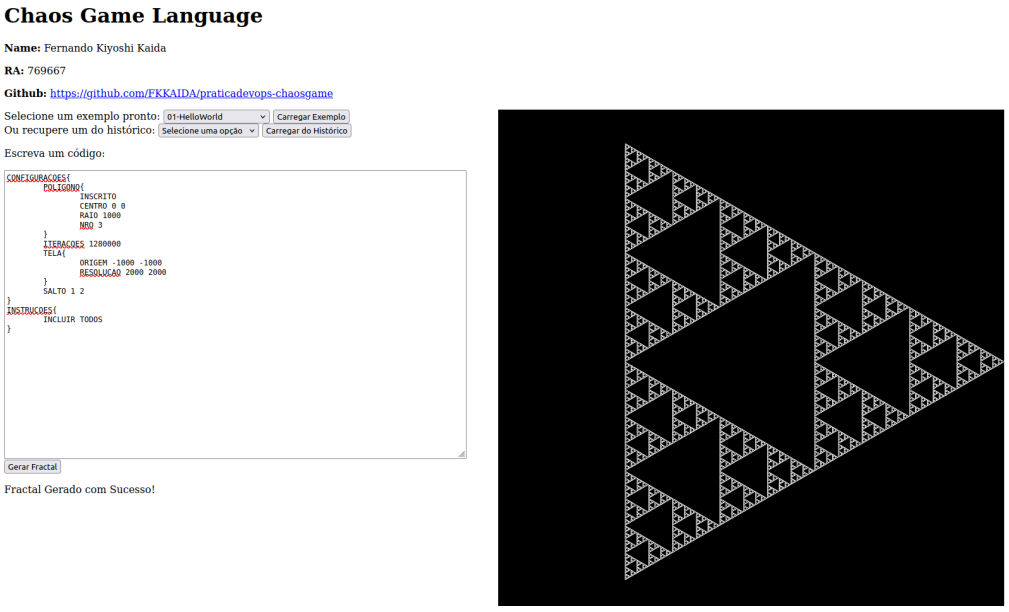

Figura 1 - Página da aplicação com o primeiro exemplo executado.

A página contém uma textArea em que o usuário pode inserir o código e um botão ‘gerar fractal’. Se o código estiver correto, uma imagem será carregada à esquerda e uma mensagem confirmando o sucesso aparecerá. Caso contrário, uma mensagem de erro irá aparecer, expondo a natureza do erro e a linha.

Acima possui dois menus do tipo dropdown. O primeiro serve para carregar códigos prontos elaborados para demonstrar as funcionalidades da linguagem. O segundo, para carregar códigos e imagens bem sucedidos, nomeados com a data em que foram executados no formato AAAA-MM-DD–HH-MM-SS.

5. Estrutura do Projeto

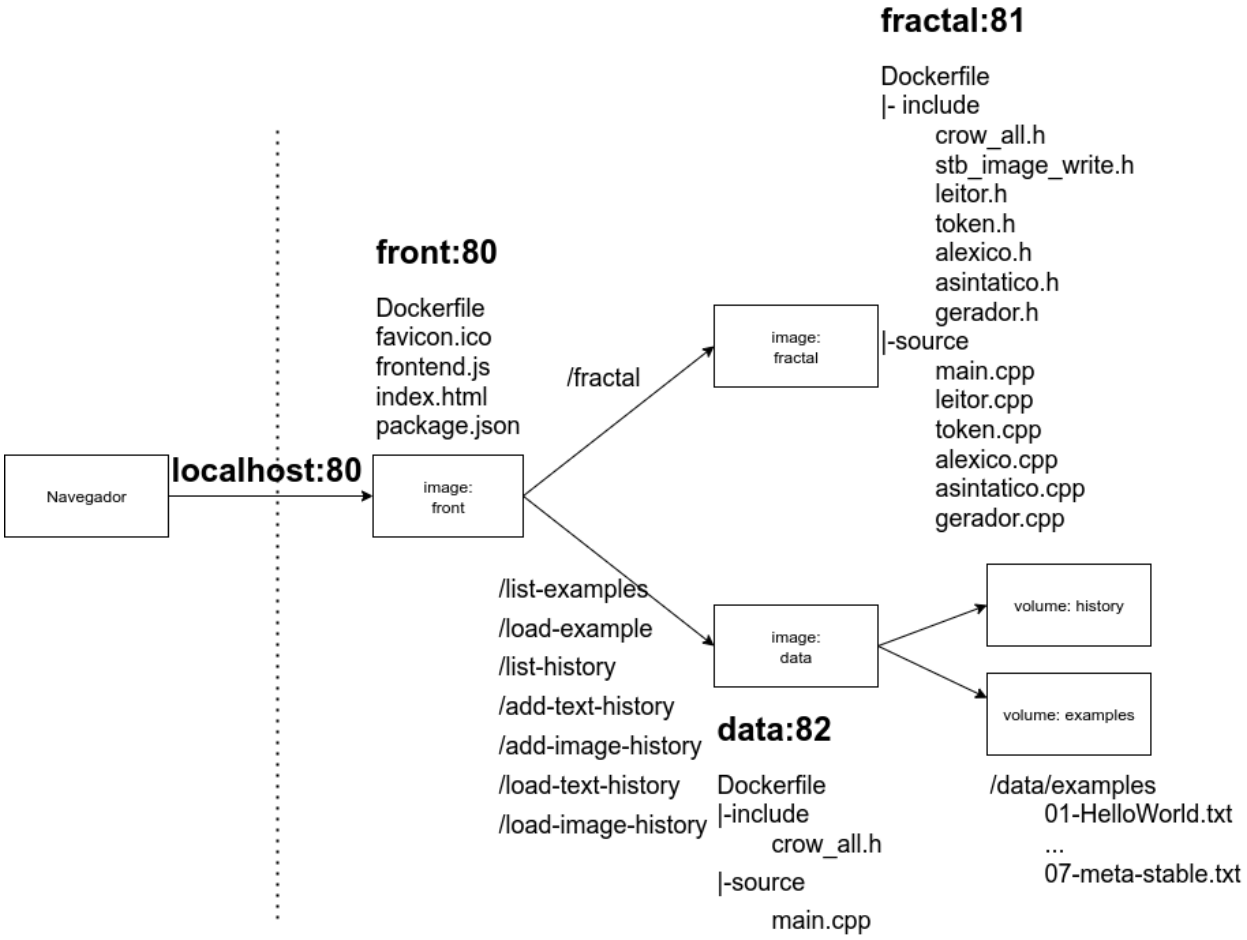

Figura 2 - Diagrama da organização da aplicação.

A aplicação faz o uso de três containers e a figura 3 exibe como eles interagem entre si e os arquivos envolvidos na construção das respectivas imagens, que estão em cada uma das pastas com os mesmos nomes. O arquivo compose.yaml é responsável pela orquestração: constrói as imagens, atribui as variáveis de ambiente de cada container, armazenando portas e ips, e mapeia as redes, volumes e portas.

5.1. front

A imagem ‘front’ foi construída em cima da imagem node:16.14.0-alpine e foi utilizando node.js

para montar o servidor. package.json contém os pacotes utilizados: express, node-fetch, asio e cors. frontend.js contém o código do servidor, que repassa os comandos feitos o usuário na página index.html (figura 1) carregada no navegador, acessível pelo mapeamento da porta 80 do container com a máquina local, para os outros servidores e retorna os resultados.

5.2. fractal

A imagem ‘fractal’ foi construída em cima da imagem alpine:3.22 e o servidor foi feito utilizando C++ e asio-dev e crow para lidar com as requisições http. O ‘front’ faz uma requisição contendo o código na textArea de index.html e retorna uma imagem no formato binário ou uma mensagem de erro sobre o código.

5.3. data

Possui as mesmas configurações de construção de ‘fractal’. Possui dois volumes: ‘examples’, que é montado em cima da pasta /data/examples e contém códigos prontos e que podem ser requisitados pelo o usuário e carregados na textArea; e ‘history’, que é gerenciado pelo docker e toda vez que uma imagem é gerada com sucesso, o código utilizado e ela são salvos no volume, nomeados com a data AAAA-MM-DD–HH-MM-SS do momento, como arquivo de texto (‘txt) e binário (‘.bin’).

5.4. minikube

Para executar a aplicação no minikube, foi desenvolvido um Helm chart na pasta /charts que busca seguir a figura 2. Nesta pasta possui pasta Charts.yaml que define as três partes descritas como dependências e que estão em suas respectivas , que são implantadas em três pods separados. Na pasta templates possui um arquivo ingress.yaml que mapeia a porta 80 do pod contendo a imagem ‘front’ com ‘k8s.local’, que foi por sua vez mapeado com o ip do minikube, permitindo que um usuário na máquina local acesse a aplicação.

Cada imagem será executada dentro de um container dentro de pods separados. Cada uma das três partes estão em pastas separadas na pasta /charts/charts. Cada um possui um deployment.yaml dita as configurações do pod, que cria um container contendo as respectivas imagens; e um service.yaml, que permite que os pods contendo as imagens ‘fractal’ e ‘data’ seja acessíveis para ‘front’, e este para o ingress.

‘data’ possui dois volumes, ‘examples’ e ‘history’, que são configurados com examples-pv.yaml, examples-pvc.yaml, history-pv.yaml e history-pvc.yaml. Eles são montados em cima dos diretórios /examples e /history do minikube respectivamente. Nota-se que como os exemplos estão na máquina local, primeiro deve-se montar /data/examples em /examples.

6. Linguagem

O propósito da linguagem desenvolvida é simplificar a criação de fractais pelo jogo do caos. Ela é dividida em duas etapas definidas com as palavras chaves: ‘CONFIGURACOES’ e ‘INSTRUCOES’.

‘CONFIGURACOES’ define, entre chaves (‘{‘ e ‘}’), espera as subetapas ‘POLIGONO’, ‘ITERACOES’, ‘SALTO’ e ‘TELA’, descritos na tabela 1.

Em ‘INSTRUCOES’ é feita a parte de seleção de vértices de cada iteração, utilizando ‘INCLUIR’ e ‘EXCLUIR’. É possível selecionar o vértice diretamente ou iterações passadas. Há o suporte de comparação de vértices e condicionais.

Palavra Chave	Formato	Descrição
CONFIGURACOES	CONFIGURACOES{ … }	Define as configurações antes de executar o algoritmo
POLIGONO	POLIGONO { … }	Define os vértices que serão escolhidos. Possui o modo ‘INSCRITO’ e ‘MANUAL’
INSCRITO	INSCRITO CENTRO < INT> <INT> RAIO <INT> NRO <INT>	Define um polígono de ‘NRO’ lados inscrito de uma circunferência com ‘CENTRO’ de coordenadas xy e ‘RAIO’.
MANUAL	MANUAL VERTICE…	Define os vértices de forma manual com qualquer número de ‘VERTICE’s.
VERTICE	VERTICE <INT> <INT>	Define um vértice de coordenadas xy com peso 1.
VERTICE	VERTICE <INT> <INT> <INT>	Define um vértice de coordenadas xy com o terceiro argumento.
ITERACOES	ITERACOES <INT>	Define o número de vezes que ‘INSTRUCOES’ será executado.
SALTO	SALTO <INT> <INT>	Define o salto na razão <INT>/<INT>.
SALTO	SALTO <FLOAT>	Define o salto na razão <FLOAT>.
TELA	TELA { ORIGEM <INT> <INT> RESOLUCAO <INT> <INT>}	Define a tela de visualização, com o canto inferior esquerdo definido por ‘ORIGEM’ e com resolução ‘RESOLUCAO’ em pixels.
INSTRUCOES	INSTRUCOES { … }	Define a etapa de seleção de vértices.
INCLUIR	INCLUIR (TODOS | VERTICE | ANTERIOR)	Inclui vértices com base na palavra chave escolhida.
EXCLUIR	EXCLUIR (TODOS | VERTICE | ANTERIOR)	Exclui vértices com base na palavra chave escolhida.
TODOS	TODOS	Seleciona todos os vértices definidos em ‘POLIGONO’

VERTICE	VERTICE(<INT>) (+ | - <INTt>)* (+|- [<INT>, …])*	Seleciona o vértice definido. Possui dois argumentos opcionais: o primeiro é o deslocamento, que seleciona um vizinho; e o segundo permite que selecione múltiplos vértices com um vetor de deslocamentos. Nota-se que para selecionar o vértice original, deve-se incluir o deslocamento '0'. Os argumentos opcionais operam em lógica circular.
ANTERIOR	ANTERIOR(<INT>) (+ | - <INTt>)* (+|- [<INT>, …])*	Seleciona o vértice escolhido em uma iteração anterior definida com um número positivo maior que 0. Possui dois argumentos opcionais: o primeiro é o deslocamento, que seleciona um vizinho; e o segundo permite que selecione múltiplos vértices com um vetor de deslocamentos. Nota-se que para selecionar o vértice original, deve-se incluir o deslocamento '0'.
IF	IF(...) {(INCLUIR | EXCLUIR)}	Condicional. Suporta 'ELSE' e 'ELSE IF(){}'
(== | !=)	(VERTICE(<INT>) (+|-<INT>) | ANTERIOR(<INT>) (+|-<INT>)) (== | !=) (VERTICE(<INT>) (+|-<INT>) | ANTERIOR(<INT>) (+|-<INT>))	Comparação de igualdade e diferença entre dois vértices. Pode ser vértices definidos ou vértices de iterações passadas. Suporta deslocamento

Tabela 1 - Palavras chaves com argumentos e descrições da linguagem.

Assume-se que no começo de uma iteração não há nenhum vértice escolhido, e no final, é feito um sorteio com base no peso do vértices escolhidos.

Assume-se que o vértice inicial é o 'VERTICE(0)' e que na primeira iteração ANTERIOR(<INT>) são o 'VERTICE(0)'.

```
CONFIGURACOES{
        POLIGONO{
                INSCRITO
                CENTRO 0 0
                RAIO 1000
                NRO 5
        }
        ITERACOES 1280000
        TELA{
                ORIGEM -1000 -1000
                RESOLUCAO 2000 2000
        }
        SALTO 0.5
}
INSTRUCOES{
        INCLUIR TODOS
        EXCLUIR ANTERIOR(2) + [2,3]
}
```

Figura 3 - Exemplo de código.

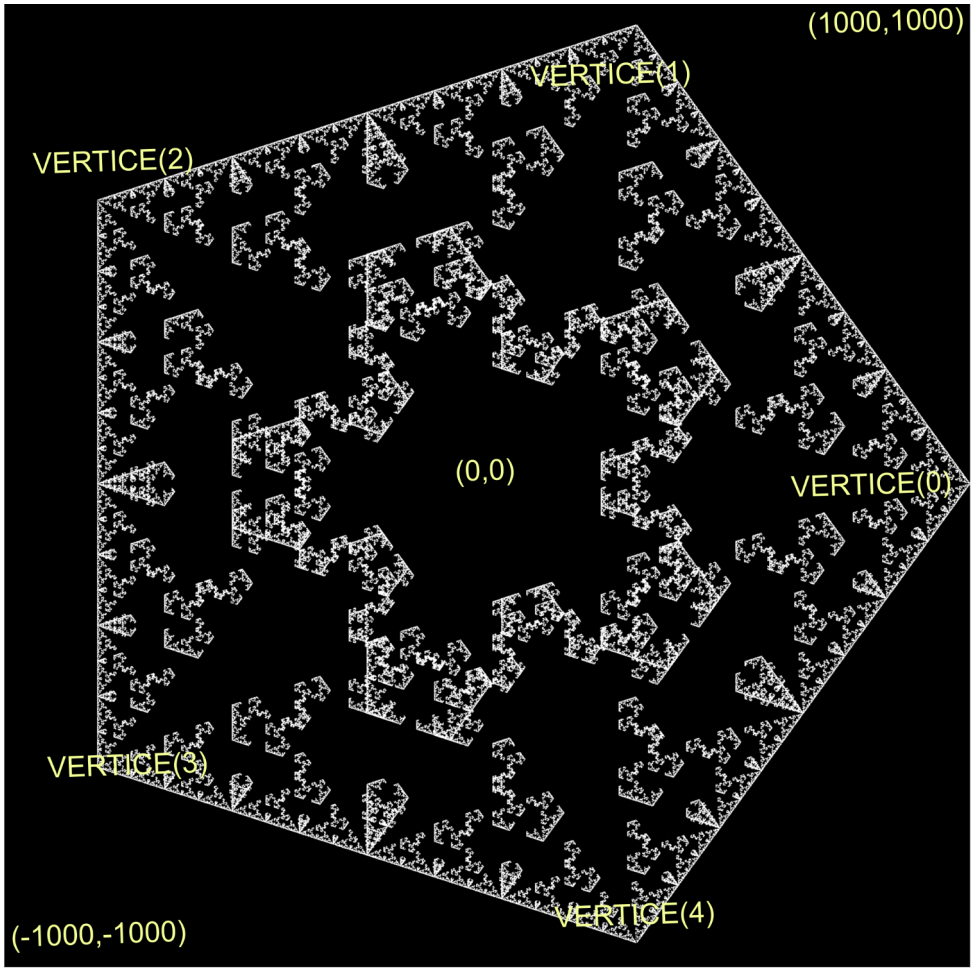

Figura 4 - Imagem gerado pelo código da figura 3 com as coordenadas e vértices destacadas para exemplificar a linguagem.

Na aplicação há sete exemplos prontos que exploram um pouco as funcionalidades da linguagem.

7. Problemas

Foram reconhecidos alguns problemas na aplicação:

- Se no final de ‘INSTRUCOES’ não houver nenhum vértice para a seleção, o algoritmo falha, mas não retorna uma mensagem de erro como deveria.
- O sinal de ‘+’ e ‘-’ deve estar separado do inteiro em ‘VERTICE’ e ‘ANTERIOR’. Por exemplo, a expressão “VERTICE(1)+1” ou “VERTICE(1)-1” resulta em erro pois “+1” e “-1” são interpretados como “1” e “-1”, mas casos como “VERTICE(1) + 1”, “VERTICE(1) ++1”, “VERTICE(1)- 1” e “VERTICE(1)+-1” comportam-se da maneira esperada.