CAPACITAÇÕES

JAVA PARA A PDPJ-Br

JAVA AVANÇADO

E-BOOK 4

Ronaldo Pinheiro Gonçalves Junior

Unifor | JUSTIÇA DO TRABALHO Conselho Superior da Justiça do Trabalho | Tribunal Superior Eleitoral | STJ SUPERIOR TRIBUNAL DE JUSTIÇA | JUSTIÇA 40 | PNUD | JUSTIÇA FEDERAL Conselho da Justiça Federal | CNJ CONSELHO NACIONAL DE JUSTIÇA

INTRODUÇÃO

OBJETIVO DA TRILHA

Introduzir os conceitos de Containers e Virtual Machines (VMs), abordar as tarefas de documentação e versionamento de APIs REST e realizar a execução de APIs REST, utilizando Docker Containers e Amazon Elastic Kubernetes Service (Amazon EKS).

CONTEÚDOS DA TRILHA

10. Execução de APIs REST, utilizando Docker Containers;
11. Documentação de APIs REST e versionamento de APIs REST;
12. Criação e manutenção de logs de execução, utilizando Apache Log4j2, e execução de APIs REST, utilizando Amazon Elastic Kubernetes Service (Amazon EKS).

INDICAÇÃO DE MATERIAL COMPLEMENTAR

INDICAÇÃO 1
Tipo: Tutorial
Título: Dockerizing a Spring Boot Application
Link: **https://www.baeldung.com/dockerizing-spring-boot-application**

INDICAÇÃO 2
Tipo: Tutorial (Workshop)
Título: AWS Elastic Kubernetes Service (EKS) workshop
Link: **https://www.eksworkshop.com/docs/introduction**

10 EXECUÇÃO DE APIs REST, UTILIZANDO DOCKER CONTAINERS

10.1 O que é Docker?

O uso de APIs REST encoraja o desenvolvimento de uma arquitetura com baixo acoplamento, através da definição de recursos autônomos e uso de princípios simples do protocolo HTTP. Isso implica em um maior isolamento entre serviços, uma maior facilidade no processo de manutenção, maior escalabilidade etc. Essas características incentivam o uso de tecnologias de virtualização, como o Docker, pois permitem que desenvolvedores isolem serviços e aplicações com facilidade, a fim de aumentar a portabilidade e facilitar o processo de implantação.

O Docker é uma popular plataforma de virtualização de containers, que permite que equipes de desenvolvimento possam trabalhar em um *software*, sem se preocupar com a consistência entre os múltiplos sistemas distintos. A criação e a configuração de um container permitem que desenvolvedores possam usar, empacotar e implantar o *software* de maneira simplificada e eficiente. Iremos aprender a executar essas tarefas nesta seção, mas, por ora, precisamos entender o que é um container.

10.2 Containers x Virtual Machines (VMs)

Para melhor entender como funcionam as tecnologias de virtualização, suponha que um integrante de uma equipe de desenvolvimento possui uma máquina física com 4 CPUs (unidade central de processamento), 8GB de RAM (memória primária) e com um sistema operacional MacOS. Este desenvolvedor pode isolar os recursos físicos da máquina em múltiplos recursos virtuais. Por exemplo, é possível criar duas VMs (Virtual Machines); uma, com 1 CPU e 2GB de RAM; e outra, com 2 CPUs e 1GB de RAM. Em vez de controlar diretamente os recursos físicos, as VMs entram em contato com uma camada de virtualização da máquina física (ou máquina hospedeira) que passa a funcionar como uma máquina independente. Cada VM terá um sistema operacional próprio. Isto é, uma VM pode ter um sistema Linux e a outra, Windows.

Considere o cenário em que a equipe de desenvolvimento decida realizar a instalação e a configuração de seu sistema de *software*, em cada máquina dos desenvolvedores. Isso significa que cada ambiente esteja com todos os componentes necessários instalados, como servidor de aplicação, servidor de autenticação, servidor de banco de dados e assim por diante. Caso integrantes da equipe de desenvolvimento possuam sistemas operacionais diferentes, é provável que sigam passos diferentes, durante a instalação e a configuração dos componentes, o que pode levar a um problema de consistência.

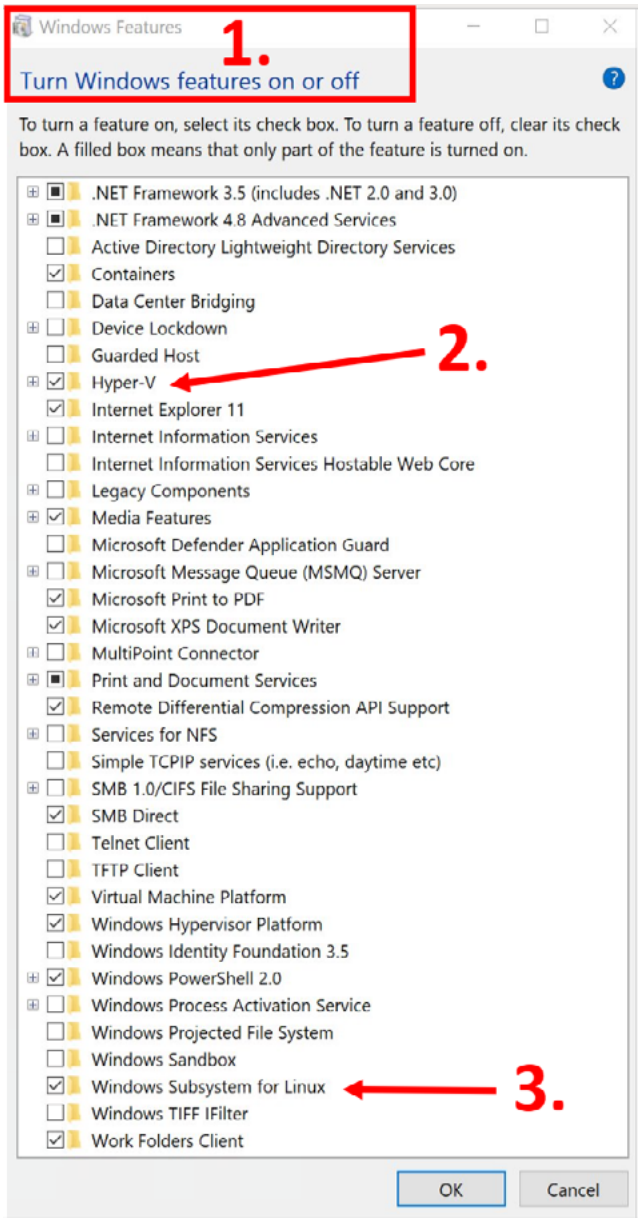
Agora imagine que a equipe decida realizar a instalação e a configuração do sistema de *software* em uma VM. Dessa forma, uma única sequência de passos consistente precisa ser executada para configurar os componentes necessários. Em seguida, os integrantes da equipe de desenvolvimento podem compartilhar a VM, independentemente do sistema operacional que usam em suas respectivas máquinas físicas.

Um container é semelhante a uma VM, pois ambos são tecnologias de virtualização. Todavia, diferente de uma VM, que possui um sistema operacional próprio por completo, um container compartilha o kernel do sistema operacional da máquina hospedeira. Isso significa que um container não precisa emular recursos virtuais, como CPU e memória. Em vez disso, utiliza o kernel da máquina física, para isolar os recursos necessários. Em outras palavras, um container possui uma camada a menos, quando comparada a uma VM, que precisa emular recursos virtuais. Dessa forma, containers tendem a ser mais eficientes na utilização de recursos, possuem um desempenho melhor e facilitam a portabilidade, pois empacotam apenas o sistema de *software* e suas dependências. Para entender esse conceito na prática, vamos agora aprender a instalar e configurar o Docker.

10.3 Instalação e configuração

Nesta seção, iremos aprender a instalar o Docker Desktop, que possui os componentes principais do Docker, para criação de containers através da linha de comando, e conta com uma interface gráfica para fácil visualização. A instalação do Docker pode variar, dependendo do sistema operacional em uso. Em um sistema Windows, é possível que o *hardware* e a versão do sistema tenham a funcionalidade “Hyper-V”. Alternativamente, é possível que seu sistema tenha a funcionalidade “WSL” ou “Windows Subsystem for Linux”. Para verificar se seu sistema tem algum desses recursos, siga os passos abaixo:

1. Busque, em seu sistema, por “Windows Features” ou “Funcionalidades do Windows”, para ativar ou desativar funcionalidades;
2. Na janela que abrir, procure, na lista, se o seu sistema possui “Hyper-V”;
3. Na mesma lista, procure se seu sistema possui “WSL”.



Elaborada pelo autor, 2024.

Ative os recursos de virtualização, conforme a imagem apresentada. Será necessário reiniciar seu sistema. Em sistemas MacOS, existem duas opções de instalação, a depender do *hardware* de processamento: uma, para chips Intel; e outra, para Apple silicon. O processo é muito semelhante, mas você deve implantar os seguintes passos, para escolher a opção apropriada[1]:

[1] Lehigh Computer Science - "Determinando a arquitetura da CPU Mac": https://docs.cse.lehigh.edu/determine-mac-architecture/

1. Clique no menu “Apple”, com o símbolo da marca;
2. Escolha a opção “About This Mac” ou “Sobre Esse Mac”;
3. Na janela que abrir, você verá uma informação de processador, que inclui “Processor Intel”, em casos de chip Intel;
4. Alternativamente, é possível que encontrar a informação de processador que inclui “Chip Apple”, em casos de Apple silicon.

Dessa forma, será possível escolher a versão de instalação correta. Finalmente, em sistemas Linux, é necessário que sua CPU tenha suporte para virtualização e um kernel de 64-bit. Sistemas Linux mais recentes atendem a esses pré-requisitos, com o hypervisor KVM utilizado por padrão. Todavia, a página do Docker conta com um passo a passo manual, para verificar se a virtualização está ativada em seu sistema para as seguintes distribuições: Ubuntu, Debian, Red Hat, Fedora e Arch[2]. Depois de confirmar que os requisitos foram atendidos, execute os passos abaixo, para instalar o Docker Desktop:

1. Acesse o *site* oficial:
   a. https://docs.docker.com/get-docker/ ;
2. Selecione a opção de instalação correspondente ao seu sistema operacional;
3. Inicie o processo de instalação.

Siga os passos de instalação, para seu sistema operacional, e execute o comando “docker --version” em um terminal, para verificar que a instalação foi bem-sucedida.

Ao executar o Docker Desktop pela primeira vez, você pode optar por utilizar as configurações padrão, para que opções eficientes sejam habilitadas de forma automática. Configurações específicas podem ser feitas futuramente, mas agora faremos a execução de um container.

10.4 Execução de containers

Para realizar a execução de um container no Docker, abra um terminal para utilizar o comando “docker container run -ti ubuntu”. Esse comando buscará pelo container “ubuntu”, em seu sistema local. Caso não encontre, o Docker realizará o *download* do repositório de imagens. Este container, em específico, é uma distribuição Ubuntu do sistema operacional Linux. O parâmetro “-ti” informado com o comando de execução do container indica que queremos um terminal interativo, assim que o container estiver em execução.

[2] Docker Docs - “Determinando o suporte de virtualização KVM”: https://docs.docker.com/desktop/install/linux-install/#kvm-virtualization-support

Uma vez que o container esteja em execução, é possível verificar se o processo foi bem-sucedido, através do comando “cat /etc/issue”. Como estamos em um container Ubuntu, esse comando irá imprimir a versão do sistema operacional. Para sair do terminal interativo, você pode apertar as teclas CTRL + C. Note que isso encerrará a execução do container. Para sair sem encerrar a execução do container, utilize as teclas CTRL+P+Q.

Vamos, agora, tentar executar um container para execução de projetos Java por meio do comando “docker container run -ti eclipse-temurin”. Para listar todos os containers em execução, execute o comando “docker container ls”.

10.5 Criação de images, utilizando Dockerfile

A imagem de um container é a definição de um empacotamento do *software* e suas dependências. Em outras palavras, podemos definir e configurar tudo que deve ser executado. Uma vez que a imagem esteja pronta, podemos dividi-la com a equipe de desenvolvimento e utilizá-la para criar várias instâncias consistentes de containers. Nos exemplos de criação de container da seção anterior, utilizamos as imagens “ubuntu” e “eclipse-temurin”. No entanto, podemos criar nossas próprias imagens, utilizando um Dockerfile.

```
FROM eclipse-temurin:21

RUN mkdir /opt/app

COPY projeto-0.0.1-SNAPSHOT.jar /opt/app

CMD [“java”, “-jar”, “/opt/app/projeto-0.0.1-SNAPSHOT.jar”]
```

Um Dockerfile descreve uma imagem de container Docker. O exemplo acima usa o comando “FROM”, para basear nossa imagem a partir da imagem do “eclipse-temurin:21”, que possui o ambiente de execução Java 21. Em adição, executamos o comando “mkdir”, para criar o diretório “/opt/app”, para implantar nosso projeto. Em seguida, utilizamos o “COPY”, para copiar nosso JAR para o recém-criado diretório no container. Finalmente, utilizamos a sintaxe de execução de comandos “CMD”, para executar nosso projeto.

Em resumo, a partir desse Dockerfile – um arquivo sem extensão que leva este exato nome – é possível:

1. criar um ambiente Java de execução;

2. criar um diretório de implantação;

3. copiar o arquivo executável do projeto; e

4. executar o projeto.

Podemos criar um container, a partir de uma imagem, e tudo acontecerá conforme definido na imagem. Para criar a imagem a partir de um Dockerfile, como o exemplo anterior, execute o comando “docker image build -t projeto:0.0.1 .”. É possível confirmar a criação da imagem, através do comando “docker image ls”. Finalmente, para executar um container, utilizando a imagem criada, basta executar o comando “docker container run -ti -p 8080:8080 projeto:0.0.1”. Neste comando, o parâmetro “-p” tem dois números – o primeiro define que a porta “8080” da máquina hospedeira será associada à porta “8080” do container, conforme define o segundo número. Nesse caso, ambas as portas são 8080, mas podemos configurar outros valores de portas. Em outras palavras, uma chamada ao endereço “localhost:8080” será direcionada para o projeto em execução no container. Você pode testar o funcionamento do container, através de um navegador web.

10.6 Criação de imagens, utilizando Docker Compose

Uma forma alternativa de criação de imagens do Docker é através do comando compose. Em um arquivo no formato YML, é possível definir serviços e configurar seus detalhes de forma semelhante ao que realizamos por meio de um Dockerfile. Podemos, inclusive, utilizar outras imagens ou até mesmo arquivos Dockerfile, para a construção de uma imagem utilizar Docker Compose. Veja, a seguir, um exemplo de arquivo docker-compose.yml:

```
version: ‘2’

services:

  app:

    image: projeto:0.0.1

    ports:
```

```
    - "8080:8080"
  build:
    context: .
  container_name: app
  depends_on:
    - db
  environment:
    - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/CasosJudiciais
    - SPRING_DATASOURCE_USERNAME=postgres
    - SPRING_DATASOURCE_PASSWORD=Seguro10!
    - SPRING_JPA_HIBERNATE_DDL_AUTO=update
db:
  image: 'postgres:16.2'
  container_name: db
  environment:
    - POSTGRES_USER=postgres
    - POSTGRES_PASSWORD=Seguro10!
    - POSTGRES_DB=CasosJudiciais
  expose:
    - 5432
```

No exemplo apresentado, dois serviços são definidos como containers: o "app", que utiliza a imagem que criamos na seção anterior; e o "db", um container dedicado para o PostgreSQL. Para criar e executar essa imagem como um container, basta executar o comando "docker-compose up".

10.7 Repositórios de images (Docker Hub e Docker Registries)

As imagens criadas anteriormente podem ser compartilhadas em repositórios públicos ou privados de imagens. O Docker Hub é um serviço de repositório público oferecido pelo Docker. Para compartilhar uma imagem no Docker Hub, execute o comando “docker login” e informe suas credenciais do Docker Hub. Uma vez que o *login* seja bem-sucedido, podemos executar o comando “docker push”, para enviar uma imagem. É necessário que o nome da imagem inclua o nome do proprietário, para que o caminho do repositório de imagens seja identificado de forma única no Docker Hub. Por exemplo, para enviar a imagem com nome “ronaldounifor/projeto:0.0.1” para o Docker Hub, o comando seria “docker push ronaldounifor/projeto:0.0.1”.

Além do Docker Hub, outros serviços de repositório de imagens populares são usados para armazenar e compartilhar imagens, como o Amazon Elastic Container Registry (ECR), o Google Container Registry (GCR), entre outros. Alguns desses repositórios são públicos e outros são privados, mas de uma forma geral são denominados de Docker Registries. O uso de tecnologias de container facilita o processo de implementação, implantação e manutenção de um sistema de *software*.

11 DOCUMENTAÇÃO DE APIs REST E VERSIONAMENTO DE APIs REST

11.1 HATEOAS

Após disponibilizar um serviço através de APIs REST, usuários finais ou demais agentes que desejam consumir o serviço precisam saber alguns detalhes da API, como o propósito do serviço, quais recursos estão disponíveis ou até mesmo exemplos de usos da API. Esses e outros detalhes são encontrados na documentação de uma API REST.

A documentação de APIs REST é o local onde uma equipe de desenvolvimento pode definir os esquemas de dados para requisições e respostas. Isto é, que formatos e tipos de dados são aceitos e gerados pelo serviço. A equipe de desenvolvedores pode definir também o que significam os códigos de *status* HTTP que são enviados nas respostas. Dessa forma, consumidores da API REST podem verificar e confirmar suas interpretações. Podemos ainda descrever os métodos de autenticação e autorização que estão em uso.

A documentação de uma API REST pode ser um artefato importante durante a navegação entre os recursos dessa API, especialmente quando existem múltiplas formas de navegar entre os recursos. Neste contexto de conexões complexas entre recursos podemos definir o HATEOAS, "Hypermedia As The Engine Of Application State" ou "hipermídia como motor do estado da aplicação". Quando uma API REST gera uma resposta que segue o princípio HATEOAS, não apenas os recursos são representados na resposta, mas também os *links* recomendados para que o consumidor utilize, a fim de dar continuidade ao uso da API REST. Existem outras formas de se disponibilizarem esses *links*, como o uso de URLs bem definidas e documentadas. Vamos ver um possível exemplo de resposta que segue o princípio HATEOAS:

```
{
  "numero": 101012,
  "descricao": "Caso 101012",
  "decisao": "A",
  "_links": {
    "self": { "href": "/api/casos/101012" },
    "atualizarCaso": { "href": "/api/casos/101012", "method": "PUT" },
    "excluirCaso": { "href": "/api/casos/101012", "method": "DELETE" }
  }
}
```

Note que é possível definir *links* de forma explícita, através do princípio HATEOAS. Nesse exemplo, um *link* para atualizar o caso e outro para excluir o caso. Esta alternativa representa uma forma dinâmica de se consumir a API de casos judiciais, sem que haja a necessidade de verificar URLs no código ou na documentação.

11.2 Documentação Swagger (Open API)

Uma linguagem popular para a documentação de APIs REST é o Swagger, também conhecido como Open API. Na Open API, desenvolvedores podem documentar uma descrição geral da API em consideração, definir quais endpoints estão disponíveis, descrever informações sobre os parâmetros de uma requisição, fornecer exemplos concretos de requisições e respostas e assim por diante.

A documentação Open API pode ser visualizada em um navegador web ou por meio de ferramentas de desenvolvimento, como o Swagger UI ou extensões e plugins de IDEs. Vamos ver agora como gerar uma documentação Open API de forma automática.

11.3 Gerando automaticamente a documentação Open API

O primeiro passo para gerar uma documentação Open API, automaticamente, em nosso projeto Spring Boot, é escolher uma ferramenta auxiliar, conforme mencionado na seção anterior. Nesse caso, iremos utilizar o Open API do SpringDoc, que pode ser adicionado através da inclusão da seguinte dependência Maven, em nosso projeto:

```
<dependency>
<groupId>org.springdoc</groupId>
<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
<version>2.4.0</version>
</dependency>
```

Em seguida, iremos definir uma documentação Open API, em nossa aplicação Spring Boot, através da anotação:

```
@OpenAPIDefinition(info = @Info(title = “Casos Judiciais OpenApi”, version = “1”, description = “API do projeto Casos Judiciais”))
```

Essa anotação inclui um título para nossa documentação, uma versão (falaremos, em breve, sobre versionamento) e uma descrição geral da API REST. Vamos utilizar nossa API REST de arquivos, para realizar a documentação automática. Modifique as anotações da classe para definir que o retorno será do tipo JSON e defina um rótulo ou "tag" para nossa API de arquivos:

```
@RequestMapping(value = "/api/arquivos", produces = {"application/json"})
@Tag(name = "api-arquivos")
```

Neste momento, iremos definir algumas notações opcionais adicionais, para aprender como incluir uma descrição geral de cada recurso e explicar os códigos de *status*, durante o retorno do método "uploadArquivo":

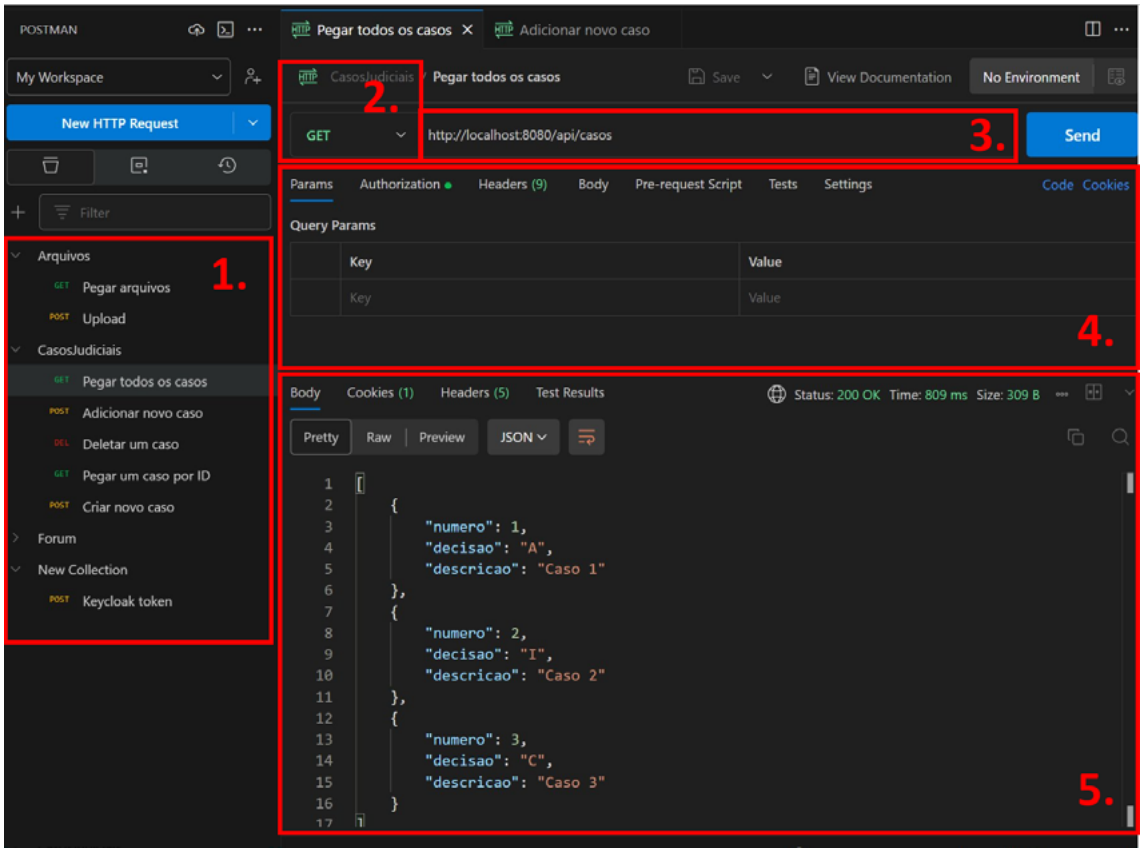
```
@Operation(summary = " Realiza o upload de um arquivo", method = "POST")
@ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "Sucesso!"),
        @ApiResponse(responseCode = "500", description = "Erro!")})
@PostMapping(value = "/upload", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
```

Note que a primeira anotação define uma descrição geral da operação de *upload*. A segunda anotação descreve o que significa cada código de *status* incluso nas respostas e a terceira e última anotação define que o recurso da API, no caminho "/upload", precisa receber como parâmetro uma mídia do tipo MULTIPART_FORM_DATA_VALUE. A figura a seguir mostra a documentação gerada automaticamente, que pode ser acessada no endereço "http://localhost:8080/swagger-ui/index.html#/":

Elaborada pelo autor, 2024.

11.4 Ferramenta Postman

Durante o desenvolvimento de um serviço, a fim de testar uma API REST de uma maneira mais detalhada, faz-se necessária uma ferramenta cliente capaz de gerar requisições e visualizar respostas, de forma eficiente e intuitiva. Um exemplo popular de um desses clientes é o Postman.



Elaborada pelo autor, 2024.

A ferramenta Postman pode ser usada para testar uma API, verificar a documentação e assim por diante. A figura acima mostra algumas das principais atividades realizadas nessa ferramenta:

1. Exploração de coleções de requisição;
2. Definição de verbos HTTP para requisições;
3. Definição de endereço para consumo de API;
4. Visualização e gerenciamento da requisição e seus detalhes;
5. Visualização e gerenciamento da resposta e seus detalhes.

Caso ainda não tenha feito, recomenda-se a execução do caderno de atividades da primeira trilha, onde temos um passo a passo de instalação e configuração do Postman.

11.5 O que é versionamento semântico?

Ao produzir novas funcionalidades, corrigir *bugs* ou defeitos, aplicar melhorias estéticas ou de desempenho, a documentação de uma API REST pode ficar defasada. Logo, realizar o versionamento de uma API pode ajudar a identificar as funcionalidades presentes em determinado momento. Uma convenção popular adotada entre desenvolvedores é o versionamento semântico, onde se atribuem números, a depender do trabalho que foi realizado e das modificações aplicadas.

No versionamento semântico, encontramos, comumente, três números separados por dois pontos, no formato "0.0.1". O primeiro número faz referência à versão principal ou "major version". Esse número apenas é incrementado quando o conjunto de mudanças faz com que a nova versão seja incompatível com as versões anteriores. O segundo número faz referência à versão secundária ou "minor version", que, em geral, significam novas funcionalidades ou um conjunto de correções significativo. O último número faz referência à versão de correção ou "patch", onde são encontradas apenas correções pontuais de *bugs* ou problemas específicos, como melhorias de segurança.

11.6 Alternativas de implementação de versionamento de REST APIs

Muitas formas alternativas podem ser usadas para implementar versionamento de APIs REST. Possivelmente, a mais comum é através do endereço URL. Por exemplo, o controlador de casos judiciais pode definir o recurso em um endereço "/api/casos/v1". Nesse caso, para acessar a segunda versão, poderíamos acessar o endereço "/api/casos/v2".

> As maiores vantagens dessa alternativa de versionamento é que versões distintas podem seguir em execução e desenvolvedores e usuários têm um controle explícito sobre a versão que gostariam de consumir. Atualmente, o PJe (Processo Judicial Eletrônico) usa esse tipo de implementação[3].

A segunda principal maneira de implementar versionamento de APIs REST é através de um parâmetro no cabeçalho de requisições. Por exemplo, podemos incluir "X-API-Version: 1", no cabeçalho de uma requisição, para acessar a primeira versão da API. Podemos facilmente modificar esse valor para "X-API-Version: 2", para acessar a segunda versão, e assim por diante. Essa alternativa não é tão explícita quanto a primeira, mas deixa a URL mais limpa.

Mais uma alternativa que pode ser usada é a definição de parâmetros de consulta no endereço de requisição, para acessar uma versão específica da API. Essa alternativa é semelhante à primeira, mas simplifica o caminho da URL trazendo a versão para o final do endereço, como,

[3] Padrões de API do Processo Judicial Eletrônico (PJe): https://docs.pje.jus.br/manuais-basicos/padroes-de-api-do-pje

por exemplo: “/api/casos/1?version=1”. A desvantagem é que deixa a URL, no geral, mais longa. Uma última alternativa de implementação de versionamento de APIs REST é no domínio, onde podemos implantar versões distintas de uma API, em domínios distintos, como “v1.cnj.br/api/casos” e “v2.cnj.br/api/casos”. Nesta última, temos que manter dois domínios distintos e gerenciar duas implantações diferentes, mas temos uma separação clara dos recursos de uma API como um todo.

12 CRIAÇÃO E MANUTENÇÃO DE LOGS DE EXECUÇÃO, UTILIZANDO Apache Log4j2, E EXECUÇÃO DE APIs REST, UTILIZANDO AMAZON ELASTIC KUBERNETES SERVICE (Amazon EKS)

12.1 Criação e manutenção de logs de execução, utilizando Apache Log4j2

A sequência de tarefas ou atividades executadas durante o uso convencional de uma API REST envolve os seguintes passos:

1. uma requisição é gerada;
2. a requisição é recebida pelo serviço;
3. a requisição é processada;
4. uma resposta é gerada; e
5. a resposta é enviada para o consumidor do serviço.

Neste cenário tradicional, após inúmeras requisições serem atendidas dessa maneira, não existe a possibilidade de estudarmos quais operações foram bem-sucedidas e quais falharam. Veremos, agora, que o uso de uma tecnologia de logs pode trazer grandes vantagens.

A ocorrência dos mais diversos tipos de erros durante a execução de múltiplos serviços pode resultar na perda de requisições ou até mesmo em inconsistências em um banco de dados. Com o uso de uma tecnologia de logs, como o Apache Log4j2, podemos registrar certos estados de componentes do nosso interesse. Ou seja, podemos registrar todos os momentos em que nosso código produz uma resposta apropriada, com sucesso, e/ou quando ocorrem falhas.

O uso de logs nos permite revisitar o histórico de operações que foram processadas pelo sistema de *software*, a fim de melhor entender seu funcionamento. Esta tarefa é semelhante ao processo de depuração de código, onde investigamos o comportamento do código, durante o fluxo de execução. Todavia, como logs ficam registrados, podemos realizar essa investigação de modo assíncrono e para um período estendido de funcionamento de um serviço. Para utilizar o Log4j2 em nosso projeto, o primeiro passo é adicionar as dependências ao projeto:

```
        <dependency>
                <groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j-api</artifactId>
<version>2.23.1</version>
</dependency>
<dependency>
<groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j-core</artifactId>
<version>2.23.1</version>
</dependency>
```

Podemos configurar nosso projeto para manter um registro de mensagens de nível “info” ou superior. Ou seja, por meio de um arquivo de configuração, fazemos a configuração do padrão de registro. O arquivo a seguir mostra um exemplo do arquivo de configuração “log4j2.xml”:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="info">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

Finalmente, é possível definir locais, em nosso código Java, para registrar mensagens de informação “info” e de erro “error”. No exemplo abaixo, um Logger é utilizado, para registrar o resultado do processamento de um *upload* de arquivo, quando o processo é bem-sucedido e quando algum erro ocorre.

```java
@RestController
@RequestMapping(value = "/api/arquivos", produces = {"application/json"})
@Tag(name = "api-arquivos")
public class ArquivoController {

    private static final Logger logger = LogManager.getLogger(ArquivoController.class);


    @PostMapping(value = "/upload", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    public ResponseEntity<String> uploadArquivo(@RequestParam("file") MultipartFile file) {
        if (file.isEmpty()) {
            logger.error("Erro ao fazer upload: Arquivo vazio");
            return new ResponseEntity<>("Selecione um arquivo para fazer upload", HttpStatus.BAD_REQUEST);
        } try {
            //...
            logger.info("Arquivo carregado com sucesso");
            return new ResponseEntity<>("Arquivo carregado com sucesso", HttpStatus.OK);
        } catch (IOException e) {
            logger.error("Erro ao fazer o upload do arquivo");
            return new ResponseEntity<>("Falha ao fazer upload do arquivo", HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    //...
}
```

Elaborada pelo autor, 2024.

12.2 O que é Kubernetes?

No contexto de desenvolvimento de *software* utilizando tecnologias de containers, ferramentas como o Docker podem facilitar o processo de instalação e configuração de componentes e suas dependências em containers. Uma vez que uma imagem de um container seja criada, vários containers podem ser rapidamente provisionados. Isto é, em um momento de maior demanda, é possível replicar um container diversas vezes, até que a demanda seja melhor atendida. De maneira análoga, em momentos de baixa demanda, as réplicas dos containers podem ser desligadas para economizar recursos. Chamamos de escalabilidade a característica de uma aplicação que tem a capacidade de aumentar ou diminuir seus recursos automaticamente, a depender da demanda e do número de requisições. No contexto de containers, uma forma fácil e eficiente de prover escalabilidade é através da orquestração de containers. Algumas das principais ferramentas que realizam a orquestração de containers são o Docker Swarm e o Kubernetes.

O Docker Swarm é uma solução de simples configuração que já vem integrada ao Docker Desktop. Já o Kubernetes, desenvolvido pela Google e agora mantido pela *Cloud Native Computing Foundation* (CNCF), é altamente escalável e tem uma arquitetura mais robusta, modular e distribuída. Em geral, pode-se dizer que o Docker Swarm é uma ótima opção para

projetos pequenos, enquanto o Kubernetes é uma boa opção no geral, mas é recomendado, especialmente, para orquestração de containers em alta escala.

De uma maneira simplificada, podemos dizer que, na orquestração de containers, são encontrados dois tipos de componentes: o mestre, ou *master*, responsável por gerenciar um *cluster*; e o trabalhador, ou *worker*, nó onde os containers são implantados. Vamos aprender, agora, como instalar e configurar nossa própria solução no Amazon EKS.

12.3 Instalação e configuração

A instalação e a configuração de um serviço como o Kubernetes, em um ambiente local, envolve a instalação de múltiplas máquinas, a configuração de rede entre elas e a configuração de uma plataforma de containers, como o Docker. Essa abordagem, conhecida como "bare metal", pode ser demorada e exige um determinado conhecimento de sistemas operacionais, redes, virtualização e assim por diante. Uma alternativa popular e que usaremos em nosso material é o serviço EKS (*Elastic Kubernetes Service*) da Amazon.

O serviço de Kubernetes EKS da Amazon é público e pode ser utilizado com maior facilidade, de três formas principais: o console de UI (*User Interface*) da plataforma AWS (*Amazon Web Services*), para uma opção visual simplificada de criação; a ferramenta eksctl, uma solução CLI (*Command Line Interface*) da Amazon, para simplificar configurações mais complexas; ou a ferramenta Terraform, uma solução opensouce de IaC (*Infrastructure as Code*), para gerenciar o EKS.

Nas próximas seções, iremos investigar como realizar a criação e a manutenção de *clusters* EKS, usando a ferramenta de linha de comando da Amazon com a eksctl. O caderno de atividades tem um passo a passo de instalação dessa ferramenta e dos pré-requisitos que devem ser atendidos. Você pode também utilizar a interface gráfica da AWS diretamente ou instalar a ferramenta Terraform.

12.4 Criação e manutenção de clusters EKS

O serviço EKS da Amazon gerencia os nós *master* de um *cluster* e precisamos apenas criar os nós *workers*. Todavia, para criar um *cluster*, é necessário possuir uma conta da AWS, utilizar um VPC (*Virtual Private Cloud*) padrão, ou criar uma, configurar um usuário no serviço IAM (*Identity and Access Management*). Nosso caderno de atividades mostra o passo a passo de configuração desses pré-requisitos.

Uma vez que o ambiente esteja criado, caso você opte por utilizar a interface, será necessário criar um Control Plane, que envolve a escolha de um nome, uma região, um VPC e opções de segurança. Como iremos utilizar a ferramenta eksctl, isso tudo é feito automaticamente, por meio do comando "eksctl create cluster", que utiliza os valores padrão e agiliza o processo de criação e manutenção do *cluster*. Para criar um *cluster* personalizado, podemos definir

as informações como nome, versão, região, nome do grupo, tipo da máquina EC2 (*Elastic Compute Cloud*) e quantidade de nós no próprio comando eksctl:

```
eksctl create cluster --name projeto --version 1.29 --region us-east-1 --nodegroup-name linux-nodes --node-type t2.micro --nodes 2
```

Nesse caso, estamos criando um *cluster* chamado "projeto" e estamos utilizando máquinas do tipo t2.micro para os nós. Vale ressaltar que a razão para a escolha dessas máquinas é o Free Tier. No primeiro ano de uso da Amazon, podemos utilizar alguns recursos de maneira gratuita, mas tome cuidado para não gerar custos inadvertidamente, dado que os serviços da Amazon são pagos. Dentre os comandos de manutenção do *cluster*, temos o de limpeza, onde recursos alocados são removidos:

```
eksctl delete cluster --name projeto
```

12.5 Criação e manutenção de recursos computacionais (nodes e Fargate)

Uma vez que nosso foco está na criação dos recursos computacionais, o primeiro passo é decidir o tipo de recurso que iremos adotar. O primeiro tipo, os nodes, são instâncias EC2 que executam containers Kubernetes. Nessa opção, temos que entender bem o serviço de infraestrutura da Amazon para configurar e gerenciar detalhes, como dimensionamento automático, balanceamento de carga e monitoramento. Essa é uma boa opção para *clusters* que precisam de maior controle. A segunda opção é o Fargate, um serviço *serverless* que permite a execução direta de containers na Amazon. Dessa forma, não existe a necessidade de utilizar ou manter instâncias EC2 ou se preocupar com a infraestrutura – esta parte fica também sob o controle da Amazon – e desenvolvedores podem se preocupar apenas com a aplicação.

Para realizar a criação de nodes ou criação de um perfil Fargate, use os respectivos comandos:

```
eksctl create nodegroup --cluster projeto --name linux-nodes --node-type t2.micro --nodes 2 --region us-east-1

eksctl create fargateprofile --cluster projeto --name develop --namespace develop --region us-east-1
```

Podemos atualizar essas informações com novos parâmetros, mantendo os comandos apresentados, mas substituindo "create" por "update". Além disso, podemos remover os recursos com os exatos mesmos comandos, porém trocando "create" por "delete".

12.6 Configurações de armazenamento, rede e segurança

Ao utilizar o serviço EKS, é comum encontrar aplicações que utilizem um armazenamento na AWS, como, por exemplo, através do serviço EBS (*Elastic Block Store*). Podemos configurar o armazenamento de um node, durante o momento de criação através do parâmetro "--node-volume-size", informando o tamanho do volume que queremos utilizar. Atualmente, o Free Tier, da Amazon, permite até 30GB de SSD no EBS.

Quanto à configuração de rede, como utilizamos um VPC, uma configuração básica já se encontra em uso. Todavia, é possível especificar a sub-rede do VPC durante a criação de um cluster, adicionando o parâmetro "--subnet" com o ID da sub-rede. Podemos ainda configurar o DNS (*Domain Name System*) interno de um *cluster* através do parâmetro "--dns-cluster". Os valores "public" e "private" habilitam e desabilitam o DNS interno, respectivamente.

Finalmente, um último passo de configuração envolve segurança. A ferramenta eksctl pode habilitar uma autenticação via OIDC (*OpenID Connect*) através da inclusão do "--with-oidc", no comando de criação de *cluster*. É possível ainda configurar um *cluster*, para restringir o acesso baseado em papéis:

```
eksctl create cluster --name projeto --region us-east-1 --managed --fargate --with-oidc --ssh-access --node-type t2.micro --nodes 2
```

Essas configurações de segurança podem ser feitas diretamente no serviço IAM da Amazon, conforme exemplo no caderno de atividades.

12.7 Gerência de clusters

As principais tarefas durante o gerenciamento de um *cluster*, no EKS, são as tarefas de criação, atualização e remoção de *clusters*, conforme vimos nos exemplos das seções anteriores. Todavia, vale mencionar também que podemos mudar os números mínimo e máximo de nós em um *cluster*, através dos parâmetros "--node-min" e "--node-max", respectivamente. Além disso, caso um desenvolvedor queira atualizar uma versão do Kubernetes ou aplicar patches de segurança a um *cluster*, a ferramenta eksctl facilita esse processo por meio do comando "eksctl upgrade cluster –name projeto". Vimos, agora, a última parte do conteúdo da capacitação de Java Avançado. Após realizar as práticas contidas no caderno de atividades, caso ainda tenha interesse nos demais serviços da Amazon, veja como utilizar o serviço CloudWatch e AWS X-Ray, para monitorar e diagnosticar o desempenho de um *cluster* em produção.

REFERÊNCIAS

BOUNA, P. **The Ultimate Swagger Tools Course**: Build OpenAPI with Ease. 1ª edição. Editora Packt Publishing, 2022.

BRETET, A. **Spring MVC Cookbook**: Over 40 Recipes for Creating Cloud-ready Java Web Applications With Spring MVC. 1ª edição. Editora Packt Pub Ltd, 2016.

MIELL, Ian; SAYERS, A. **Docker in Practice**. 2ª edição. Editora Manning Publications, 2019.

NICKOLOFF, J; KUENZLI, S. **Docker in Action**. 2ª edição. Editora Manning, 2019.

PIPER, B; CLINTON, D. **AWS Certified Solutions Architect Study Guide**. 3ª edição. Editora Sybex Inc, 2021.

POULTON, N. **Docker Deep Dive**: Zero to Docker in a single book. 1ª edição. Editora Kindle, 2023.

POULTON, N. **The Kubernetes Book**. 1ª edição. Editora Kindle, 2024.

ROYAL, P. **Building Modern Business Applications**: Reactive Cloud Architecture for Java, Spring, and PostgreSQL. 1ª edição. Editora Apress, 2022.

SAYFAN, G. **Mastering Kubernetes**. 4ª edição. Editora Packt, 2023.

SUBBU, A. **RESTful Web Services Cookbook**: Solutions for Improving Scalability and Simplicity. 1ª edição. Editora O'Reilly - Yahoo Press, 2010.

THE APACHE SOFTWARE FOUNDATION. **User's Guide**: "Apache Log4j 2 v.2.3.2". Publicação eletrônica, 2021. Disponível em: https://logging.apache.org/log4j/2.3.x/log4j-users-guide.pdf/. Acesso em: 29 mar. 2024.