LEVANDO IA PARA PRODUÇÃO

UNIDADE III
MICROSERVICES E COMO UTILIZAR PARA PRODUÇÃO

Elaboração

Paulo Vitor Pereira Cotta

Natasha Sophie Pereira

Produção

Equipe Técnica de Avaliação, Revisão Linguística e Editoração

SUMÁRIO

MICROSERVICES E COMO UTILIZAR PARA PRODUÇÃO

UNIDADE III

CAPÍTULO 1

MICROSERVICES

O que é *Microservice*?

O *Microservice*, ou arquitetura de microsserviços, é uma forma de estruturar a solução de um problema como sendo uma coleção de serviços ligeiramente acoplados, ou sem acoplamento, que implementam capacidades empresariais. Essa arquitetura permite que as soluções sejam entregues durante a produção do sistema, ou seja, possibilita a implantação contínua de aplicativos grandes e complexos, fazendo com que a organização evolua sua tecnologia.

É possível entender que a arquitetura de microsserviços é uma nova maneira para a criação de *software*. Como o nome sugere, microsserviços se referem a um *software* dividido em pequenos *softwares* ou micropacotes que têm responsabilidades reduzidas e são independentes.

Para exemplificar, vamos entender que você possui uma loja virtual feita em Python que implementa diversas funcionalidades e estruturas, tais como: cadastro de clientes, gerência de estoque e de pedidos, catálogo de produtos, e outras funcionalidades. Sendo assim, o sistema da loja virtual se torna grande, volumoso e complexo, pois possui várias funcionalidades embutidas. Para implementar um sistema desse tipo utilizando uma arquitetura de microsserviços, é possível dividir o *software* em diversos *softwares* menores e independentes, o que leva à ideia de pequenos pacotes, cada um com funcionalidade predefinida. Por exemplo, um *software* para cadastrar clientes, outro para gerenciar estoque, e assim por diante. Dessa forma, em vez de a empresa possuir uma aplicação em Python única e monolítica, são utilizadas diversas e pequenas aplicações, que se comunicam entre si para formar a loja virtual, o que demonstra que é uma comunicação e uma melhora para manter a aplicação com alta disponibilidade.

Resumindo, a arquitetura de *microservices* permite que você tenha um único *software* formado por pequenas partes.

Protocolos Rest HTTP 1.1

Representational State Transfer (REST, do inglês) ou Transferência de Estado Representativo é um estilo de desenvolvimento de *softwares* baseado nos protocolos HTTP 1.1, de modo que o envio e o recebimento de pacotes são baseados nos métodos do próprio HTTP 1.1, que são: GET, POST, DELETE, PUT, HEAD, TRACE, CONNECT e OPTIONS.

Para interagir com as URIs (*Universal Resource Identifier*), são utilizados os métodos HTTP. Note que as URIs são escritas como substantivos, e os métodos HTTP são escritos como verbos. Em outras palavras, os métodos HTTP são responsáveis alterar os recursos identificados pelas URIs. Alguns princípios básicos devem ser seguidos para que possamos ter o domínio e trabalhar com a tecnologia REST, a saber:

- uso adequado dos métodos;
- uso adequado de URLs (*Universal Resource Locator*);
- uso de códigos corretos para sucesso e falha;
- uso adequado de cabeçalhos HTTP; e
- interligação entre vários recursos diferentes.

É possível passar pelo endereço tanto uma URL quanto uma URI, visto que tudo o que é passado pelo endereço é interpretado pelo HTTP.

URL e URI

URL (*Universal Resource Locator*) é utilizado para fornecer um caminho e/ou endereço para identificação, e essa identificação é única. Uma URL (padronização) pode ser considerada uma URI (envio dos dados) de certa forma. URI (*Universal Resource Identifier*) pode ser utilizado para qualquer coisa, como caminho, dar nome a objetos etc.

Seguem os padrões necessários para a construção de um URI:

- formato do URI;
- autorização de Projeto URI;

- modelagem e estilo de código;
- arquétipos de recursos da URI;
- designer padronizado dos caminhos URI; e
- URI padrão designer de consulta.

De acordo com a definição da norma RFC3986, o formato definido para montar um URI, conforme sua sintaxe genérica, é exemplificado a seguir: URI = esquema”://”autoridade”/”caminho[“?”consulta][“#”fragmento].

O uso do separador por barras “/” é uma regra que deve ser usada para identificar hierarquia relacional, para indicar um caminho em relação aos recursos liberados hierarquicamente. Exemplo: http://localhost:8080/exemplo/pessoa/nome/paulo.

O uso de barra no final de uma URI é totalmente fora do padrão, não adiciona nenhum valor semântico para a informação e acaba mostrando que o código não está limpo o suficiente para identificar um recurso, visto que a barra estaria no final. Por exemplo: http://localhost:8080/exemplo/pessoa/nome/paulo/. Nesse caso, ficaria a pergunta: “que objeto deve chamar?”, pois não existe o objeto barra e sim “paulo”. Fica confuso e nada semântico. Ainda, hifens (“-”) devem ser usados para melhorar a legibilidade do objeto e facilitar a interpretação pelos desenvolvedores, evitando o uso de outros caracteres especiais, ou até mesmo espaços em branco no objeto para separação de palavras, por exemplo: http://localhost:8080/montadora/automovel/carro/vermelho.

O caractere especial de *underscore* (“_”) não deve ser usado, visto que confunde e deixa a informação obscura, como se fosse um sublinhado na palavra. Se necessário, utilize sempre hífen, pois, além de utilizar o padrão, ainda evitará o obscurecimento da informação.

Outra regra importante na formação da URI é o texto escrito em caixa baixa, pois a URI é *case sensitive* e pode causar problemas ao acessar os objetos detalhados nela. O detalhamento está descrito na RFC3986, que apresenta as regras *case sensitive*. Exemplo: http://localhost:8080/onibus ⇒ se colocarmos http://localhost:8080/ONIBUS, executará um “Erro 404 não encontrado”.

Quanto à regra de extensão de arquivo, não deve ser utilizada, pois deve ser configurado no *Content-Type* como cabeçalho de processamento do corpo do conteúdo.

Arquétipos de recursos

Como padrões de projeto, os arquétipos de recursos auxiliam na estruturação e comportamento, comumente encontrados em projetos de API REST. A API possui quatro tipos diferentes de arquétipos, que são:

- ***Document***: conceito singular que se assemelha a uma instância ou objeto de banco de dados. É a representação do estado de um documento e inclui campos com os valores e links para outros recursos. Um exemplo de identificação de recursos do documento poderia ser: http://localhost:8080/fabrica-tecido/tecido/vermelho.
- ***Collection***: o cliente pode propor a adição de novos objetos, e o recurso criará, ou não, uma instância. Exemplo: http://localhost:8080/fabrica-carro/onibus.
- ***Store***: repositório gerenciado pelo cliente, é um recurso de armazenamento que disponibiliza recursos HTTP para o cliente por conta própria. Por exemplo: http://localhost:8080/loja-automovel/onibus/123, que é uma requisição PUT para alterar o carro de número 123.
- ***Controller***: modelo de recursos procedural que executa funções com parâmetros e valores de retorno, entrada e saída. Por exemplo: http://localhost:8080/fabrica-chocolate/balinha/1.

Designer de caminhos URI

Cada segmento de caminho URI separado por barras representa uma oportunidade de Designer. Atribuir valores significativos para cada segmento de caminho ajuda a comunicar de forma clara a estrutura do projeto.

Via de regra, deve ser usado um substantivo singular para nomes dos documentos. Exemplo: http://localhost:8080/funcionarios/machine-learning/paulo.

A regra de uso de um substantivo no plural deve ser aplicada para nomes de coleções. Exemplo: http://localhost:8080/jogos/fm-manager/jogadores.

Assim, um substantivo no plural deve armazenar nomes no URI para identificação do recurso.

Exemplo: é correto informar http://localhost:8080/musicas/david-guetta/playlists, porém é incorreto informar http://localhost:8080/musicas/david-guetta/playlist, pois estou com um recurso que é uma coleção de dados.

Expressões verbais ou verbos devem ser usados como nomes de *Controller*, ou *Services*. Nesse caso, um URI deve identificar o recurso passado para a *Controller* ou *Service*, e ser nomeado para indicar sua ação. Exemplo: http://localhost:8080/estudante/paulo/registrar.

Já os segmentos de variáveis podem ser substituídos com base em identidades e valores. Alguns caminhos podem ser estáticos, e outros segmentos de URI são variáveis, o que significa que são automaticamente preenchidos e fornecem a singularidade do URI. E uma substituição de um URI, pode ser feita a partir da API REST selecionada e, assim que o *framework* do motor V8 do *browser* executar a substituição, pode ser repassado um identificador numérico ou alfanumérico como URI, e não se deve utilizar caracteres especiais. Exemplo: http://localhost:8080/unb/turma-7/aluno/2019 ou http://localhost:8080/unb/turma-7/professor/1paulo2.

Outra regra importante diz respeito ao uso de nomes de funções CRUD, que não devem ser usadas em um URI, uma vez que a utilização da semântica do URI deve ser apenas para identificar recursos e valores variáveis. Exemplo: http:localhost:8080/scintillan/ms/colaborador/567 – dessa forma está correto; errada será a utilização de http://localhost:8080/scintillan/ms/atualizar/567, que, além de estar fora do contexto HTTP 1.1, está mostrando qual funcionalidade do CRUD será executada, deixando exposto o código.

Designer de consulta em URI

O Designer de consultar/pesquisar fornece regras relativas à criação de uma URI. Com base na RFC3986, é opcional esse tipo de consulta em URI, sendo que deve vir após seus fragmentos. Exemplo: (URI=protocolo”://”autorização”/”caminho[“?”consultar/pesquisar][“#”parâmetros]). Como um componente, a consulta serve para a busca de um fragmento passado e retorna apenas um resultado.

A regra de trabalhar com componentes de consulta e de pesquisa pode ser usada com um conjunto de filtros. Cada componente da consulta ou da pesquisa corresponde a critérios para a pesquisa ou para retornar a tabela dos dados. É uma representação do estado de mensagem de entrada e saída. Exemplo: http://localhost:8080/teste/autor/nome=Paulo, usando um GET como requisição para retorno da busca.

A regra de paginação da consulta deve ser usada com base nos princípios do Designer de consulta e pesquisa, deve ser passada como parâmetro na coleção da URI, e, assim, a paginação será executada por demanda. Exemplo: http://localhost:8080/endereco?pagina=2&contador=50.

Código e *status* de resposta à requisição HTTP 1.1

REST API usa o processo de respostas requisitadas pelo usuário na aplicação, forma de códigos HTTP 1.1 response.

Seguem os códigos e a descrição de cada um dos retornos e forma de utilização de acordo com a RFC3986:

- Código 200: sucesso na requisição;
- Código 201: código que retorna quando executa a inserção e/ou retorno da coleção;
- Código 202: obtenção de conexão assíncrona;
- Código 204: o corpo do código está em branco de forma intencional;
- Código 301: indica que uma nova URI foi criada e inserida na aplicação;
- Código 303: retorno de dados opcionais;
- Código 304: preservação de consumo de banda;
- Código 307: indica uma URI temporária, para o cliente efetuar o acesso;
- Código 400: o caminho indicado está com erro;
- Código 401: usuário não está autorizado a enviar requisição por esta rota;
- Código 402: requisição de negação para o cliente final;
- Código 404: requisição off-line e ou não existente;
- Código 405: envia o protocolo quando o usuário envia uma informação não esperada;
- Código 406: usuário envia tipo de dados não suportado;
- Código 409: violação de objetos enviados para a requisição;
- Código 412: o processo de resposta não foi suportado;
- Código 415: mídia não suportada;
- Código 500: erro grave na aplicação.

REFERÊNCIAS

"Google's AlphaGo AI wins three-match series against the world's best Go player". TechCrunch. , 25 Maio maio 2017.

ACKLEY, D. H.; HINTON, G. E.; SEJNOWSKI, T. J. ***A learning algorithm for boltzmann machines.*** Cognitive science, Elsevier, 1985.

AIZENBERG, Igor Aizenberg,; AIZENBERG, Naum N. Aizenberg,; JOOS, P.L. ***Vandewalle. Multi-Valued and Universal Binary Neurons:*** *Theory, Learning and Applications*. Springer Science & Business Media, 2000.

BAKER, J.; DENG, Li; GLASS, Jim; KHUDANPUR, S.; LEE, C.-H.; MORGAN, N.; O'SHAUGHNESSY, D. **"*Research Developments and Directions in Speech Recognition and Understanding*",** 2009.

Balázs Csanád Csáji. ***Approximation with Artificial Neural Networks.;*** Faculty of Sciences; Eötvös Loránd University, Hungary, 2011.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. ***Representation Learning:*** *A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013*.

BENGIO, Yoshua; LECUN, Yann; HINTON, Geoffrey. ***"Deep Learning"***. Nature, 2015.

BOREN, W. L.;, HUNTER, T. B.;, BJELLAND, J. C.;, HUNT, K. R. ***Comparison of breast consistency at palpation with breast density at mammography.*** Invest Radiol, 1990.

DENG, L.; YU, D. ***"Deep Learning: Methods and Applications"*** **(PDF).** *Foundations and Trends in Signal Processing*, 2014.

HINTON, G. E.; SRIVASTAVA, N.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. R. ***"Improving neural networks by preventing co-adaptation of feature detectors"***, 2012.

HINTON, Geoffrey E.; DAYAN, Peter; FREY, Brendan J.; NEAL, Radford. ***"The wake-sleep algorithm for unsupervised neural networks"*****,** 1995.

HORNIK, Kurt. ***"Approximation Capabilities of Multilayer Feedforward Networks"***. Neural Networks, 1991.

KAPUR, Lenny. ***Neural Networks & The Backpropagation Algorithm, Explained.*** 1th ed. Spring, 2010.

KRIZHEVSKY; SUTSKEVER; HINTON. ***Methods research image.*** 2012.

LECUN *et al.*, ***"Backpropagation Applied to Handwritten Zip Code Recognition***,", Neural Computation, 1989.

LU, Z., .; PU, H., .; WANG, F., .; HU, Z., .; & WANG, L. ***The Expressive Power of Neural Networks:*** *A View from the Width*, 2017.

LUI, Bing *et al.* ***Lifelong Machine Learning:*** *Second Edition*. 2018.

MORGAN, Nelson; BOURLARD, Hervé; RENALS, Steve; COHEN, Michael; FRANCO, Horacio. ***"Hybrid neural network/hidden markov model systems for continuous speech recognition"***. *International Journal of Pattern Recognition and Artificial Intelligence*, 1998.

REFERÊNCIAS

NASCIMENTO, Rodrigo. **Afinal, o que é Big Data?**. Disponível em: <http://marketingpordados.com/analise-de-dados/o-que-e-big-data-%F0%9F%A4%96/>. Publicado em: 2017. Acesso em: 17 de jun. de 2019.

OLSHAUSEN, B. A. ***"Emergence of simple-cell receptive field properties by learning a sparse code for natural images"***. Nature, 1996.

SCHMIDHUBER, J. ***"Deep Learning in Neural Networks: An Overview".*** Neural Networks, 2015.

SHAVLIK, J. W., and ***Dietterich, T. G. Readings in Machine Learning.*** San Mateo, CA: Morgan Kaufmann Publishers, 1990.

WAIBEL, A.; HANAZAWA, T.; HINTON, G.; SHIKANO, K.; LANG, K. J. ***"Phoneme recognition using time-delay neural networks"***, 1989.

WENG, J. WENG, ; AHUJA, N. AHUJA AND; HUANG, T. S. *HUANG,* ***"Learning recognition and segmentation using the Cresceptron"*****,** 1997.