Comunicação entre servidores backend (Flask e NodeJS) na mesma máquina

Pedro Henrique Pereira Teixeira[1]

[1]Pontificia Universidade Católica de Minas Gerais (PUCMG)
Poços de Caldas, MG

pedro.teixeira.1406145@sga.pucminas.br

Abstract. *In modern software architectures, it is common for different services to be responsible for performing different tasks, communicating with each other to compose the final application. Even in local distributed systems, such as those that run on a single machine, data traffic can follow the same principles of communication between servers, as if they were on different machines. This article explores the communication model between two backend servers — one developed in Flask (Python) and the other in NodeJS — on the same machine, illustrating how they interact using the HTTP protocol and the REST model to send data and responses.*

Resumo. *Em arquiteturas modernas de software, é comum que diferentes serviços sejam responsáveis por realizar tarefas distintas, comunicando-se entre si para compor a aplicação final. Mesmo em sistemas distribuídos locais, como aqueles que rodam em uma única máquina, o tráfego de dados pode seguir os mesmos princípios de comunicação entre servidores, como se estivessem em máquinas distintas. Este artigo explora o modelo de comunicação entre dois servidores backend — um desenvolvido em Flask (Python) e o outro em NodeJS — na mesma máquina, ilustrando como eles interagem usando o protocolo HTTP e o modelo REST para enviar dados e respostas.*

1. Modelo REST

No desenvolvimento de sistemas distribuídos, o modelo REST (Representational State Transfer) é amplamente utilizado para comunicação entre diferentes servidores. Ele utiliza o protocolo HTTP como base para troca de dados e é projetado para ser simples, escalável e eficiente. Ademais, tal metodologia se baseia em princípios como a comunicação sem estado, onde cada requisição do cliente contém todas as informações necessárias para a execução da operação, sem depender do estado do servidor entre as requisições. Além disso, a comunicação nesse modelo ocorre utilizando métodos HTTP padronizados, como GET, POST, PUT e DELETE, para realizar operações sobre os recursos, que são representados geralmente em formatos como JSON ou XML.

Neste contexto, quando dois servidores backend, um desenvolvido em Flask e o outro em NodeJS, se comunicam entre si, o servidor NodeJS pode, por exemplo, receber um arquivo no formato Excel via requisição HTTP POST, salvá-lo no sistema temporário de arquivos e, em seguida, enviar o caminho para o servidor Flask utilizando uma requisição HTTP GET ou POST, para que possa ser lido e transformado em um script MySQL padronizado. Esse tipo de interação entre os servidores é um exemplo claro de como as APIs REST funcionam em uma arquitetura de comunicação entre

sistemas distintos, mesmo quando eles estão hospedados na mesma máquina. A escolha do formato JSON para enviar o caminho do arquivo permite que a comunicação seja eficiente e fácil de integrar com outros sistemas.

2. Modelo OSI

O processo de comunicação entre os servidores segue o modelo de rede TCP/IP, que está relacionado ao Modelo OSI (Open Systems Interconnection). O Modelo OSI descreve sete camadas de abstração que ajudam a entender como os dados são transmitidos entre sistemas de maneira padronizada. As camadas do modelo OSI incluem a camada física, que cuida da transmissão dos sinais físicos, e a camada de enlace de dados, que é responsável pela comunicação dentro de uma rede local. A camada de rede trata do roteamento dos pacotes, enquanto a camada de transporte, usando protocolos como TCP, garante a entrega confiável dos dados.

Apesar dos servidores Flask e NodeJS estarem rodando na mesma máquina, o processo de comunicação entre eles segue essas camadas do modelo TCP/IP. Isso significa que, mesmo sendo uma comunicação interna, os dados são encapsulados, passam pela rede e retornam ao destino, seguindo os mesmos princípios de encapsulamento e desencapsulamento que seriam aplicados se os servidores estivessem em máquinas diferentes. Na prática, isso implica que a requisição do servidor NodeJS para o servidor Flask será encapsulada em pacotes TCP, transmitida pela rede, e ao chegar ao destino, o pacote será desencapsulado e processado pelo servidor Flask.

Esse processo de encapsulamento e desencapsulamento ocorre nas camadas de transporte e rede, com o protocolo TCP garantindo que os dados sejam entregues corretamente, sem erros. O fato de os servidores estarem na mesma máquina não elimina a necessidade de seguir essas etapas, pois o sistema operacional trata a comunicação de rede local como se estivesse realizando a troca de pacotes entre máquinas diferentes. O encapsulamento dos dados na camada de rede envolve a criação de frames, que são transmitidos pela rede local e processados conforme chegam ao destino, onde a informação pode ser extraída e utilizada conforme necessário pela aplicação.

3. Considerações finais

Em resumo, a comunicação entre dois servidores backend utilizando o modelo REST e o protocolo HTTP, mesmo estando na mesma máquina, segue as camadas do Modelo OSI e as regras do modelo TCP/IP. A utilização de pacotes de dados que são encapsulados e desencapsulados ao longo do caminho ilustra a complexidade e a robustez dos sistemas de comunicação em redes, garantindo que a troca de informações seja feita de forma confiável e padronizada, independentemente da localização dos servidores.

Figure 1. Interface desenvolvida, incluindo a área de upload do arquivo e o script MySQL gerado como resultado

Referências

FIELDING, R. T. Architectural Styles and the Design of Network-based Software Architectures. 2000. (Dissertation).

POSTEL, J. Transmission Control Protocol. RFC 793, 1981. Disponível em: https://tools.ietf.org/html/rfc793. Acesso em: 4 dez. 2024.