Arquitetura básica de dados orientada à eventos e uso de Webhooks

# Apresentação

A arquitetura de dados orientada a eventos é uma maneira de tratar dados de maneira mais flexível e ágil, reagindo a cada evento de maneira independente. Isso é contrário ao processamento em batch, em que os dados são agrupados e processados em lotes.

Exemplos de uso de arquitetura de dados orientada a eventos incluem:

Sistemas de detecção de fraudes: quando um evento suspeito é detectado, uma ação é imediatamente tomada para evitar perdas financeiras.

Sistemas de monitoramento de máquinas: eventos gerados por sensores são processados em tempo real para detectar problemas antes que eles causem danos maiores.

Aplicativos de logística: eventos gerados por GPS e sensores são processados em tempo real para otimizar rotas e prever entregas.

Sistemas de alerta de emergência: eventos são processados em tempo real para detectar situações de emergência e enviar alertas imediatos.

Já o processamento em batch é adequado para casos em que a latência não é crítica e a quantidade de dados é grande, como:

Análise de dados de log de uma empresa: os dados são agrupados e processados em lotes todas as noites para produzir relatórios diários.

Análise de dados de vendas: os dados são agrupados e processados mensalmente para produzir relatórios de vendas.

Processamento de grandes conjuntos de dados climáticos: dados de sensores são agrupados e processados em lotes para produzir modelos climáticos precisos.

Em resumo, a escolha entre arquitetura de dados orientada a eventos e processamento em batch depende das necessidades do negócio, incluindo a necessidade de latência, volume de dados e requisitos de recursos. A arquitetura de dados orientada a eventos é uma opção para situações em que a velocidade é importante, enquanto o processamento em batch é uma opção para situações em que é necessário processar grandes volumes de dados.

# Webhooks

## O que são Webhooks?

Webhooks são uma maneira de enviar informações de um sistema para outro de forma automatizada. Um aplicativo envia uma solicitação HTTP para um URL especificado quando algo de interesse ocorre, permitindo que outra aplicação saiba imediatamente sobre o evento.

A ingestão de dados passiva é uma forma de coletar dados sem interação direta com o sistema de origem. Em vez disso, os dados são coletados automaticamente através de fontes como arquivos, logfiles ou APIs.

Exemplo prático da criação de um webhook com Flask:

import flask

# Inicialização da aplicação Flask

app = flask.Flask(\_\_name\_\_)

# Definição da rota "/webhook" com suporte a requisições HTTP POST

@app.route("/webhook", methods=["POST"])

def handle\_webhook():

# Recupera o conteúdo da requisição como um dicionário em Python

data = flask.request.get\_json()

# Imprime o conteúdo da requisição

print("Received data:", data)

# Retorna uma resposta HTTP simples

return "OK"

# Verifica se o script está sendo executado como um módulo principal

if \_\_name\_\_ == "\_\_main\_\_":

# Inicia a execução da aplicação

app.run()

Este script demonstra um simples aplicativo Flask em Python, onde:

A primeira linha importa o pacote Flask, que é o coração da aplicação.

Na segunda linha, a variável 'app' é inicializada como uma instância de Flask, passando name como argumento. name é uma variável especial do Python que retorna o nome do módulo corrente.

A seguir, a função handle\_webhook é decorada com @app.route, que especifica a rota (URL) para a qual ela deve responder. No caso, a rota é "/webhook" e a função só responderá a requisições HTTP do tipo POST.

A função handle\_webhook é chamada quando uma requisição POST é enviada para a rota "/webhook". Dentro da função, o corpo da requisição é recuperado com flask.request.get\_json(), que retorna o conteúdo da requisição como um dicionário em Python.

Finalmente, a função imprime o conteúdo da requisição e retorna "OK" como resposta HTTP.

O último bloco de código verifica se o script está sendo executado como um módulo principal, e, nesse caso, inicia a execução da aplicação com app.run(). Isto permite que o script seja executado como um aplicativo standalone, sem ser necessário importá-lo como um módulo em outro script.

Este exemplo cria um servidor Flask que escuta por solicitações HTTP no endpoint /webhook. Quando uma solicitação POST é recebida, a função handle\_webhook extrai os dados da solicitação e os imprime na tela.

Arquitetura básica de dados orientada à eventos e uso de Webhooks

Apresentação

A arquitetura de dados orientada a eventos é uma maneira de tratar dados de maneira mais flexível e ágil, reagindo a cada evento de maneira independente. Isso é contrário ao processamento em batch, em que os dados são agrupados e processados em lotes.

Exemplos de uso de arquitetura de dados orientada a eventos incluem:

Sistemas de detecção de fraudes: quando um evento suspeito é detectado, uma ação é imediatamente tomada para evitar perdas financeiras.

Sistemas de monitoramento de máquinas: eventos gerados por sensores são processados em tempo real para detectar problemas antes que eles causem danos maiores.

Aplicativos de logística: eventos gerados por GPS e sensores são processados em tempo real para otimizar rotas e prever entregas.

Sistemas de alerta de emergência: eventos são processados em tempo real para detectar situações de emergência e enviar alertas imediatos.

Já o processamento em batch é adequado para casos em que a latência não é crítica e a quantidade de dados é grande, como:

Análise de dados de log de uma empresa: os dados são agrupados e processados em lotes todas as noites para produzir relatórios diários.

Análise de dados de vendas: os dados são agrupados e processados mensalmente para produzir relatórios de vendas.

Processamento de grandes conjuntos de dados climáticos: dados de sensores são agrupados e processados em lotes para produzir modelos climáticos precisos.

Em resumo, a escolha entre arquitetura de dados orientada a eventos e processamento em batch depende das necessidades do negócio, incluindo a necessidade de latência, volume de dados e requisitos de recursos. A arquitetura de dados orientada a eventos é uma opção para situações em que a velocidade é importante, enquanto o processamento em batch é uma opção para situações em que é necessário processar grandes volumes de dados.

Webhooks

O que são Webhooks?

Webhooks são uma maneira de enviar informações de um sistema para outro de forma automatizada. Um aplicativo envia uma solicitação HTTP para um URL especificado quando algo de interesse ocorre, permitindo que outra aplicação saiba imediatamente sobre o evento.

A ingestão de dados passiva é uma forma de coletar dados sem interação direta com o sistema de origem. Em vez disso, os dados são coletados automaticamente através de fontes como arquivos, logfiles ou APIs.

Exemplo prático da criação de um webhook com Flask:

import flask

# Inicialização da aplicação Flask

app = flask.Flask(\_\_name\_\_)

# Definição da rota "/webhook" com suporte a requisições HTTP POST

@app.route("/webhook", methods=["POST"])

def handle\_webhook():

# Recupera o conteúdo da requisição como um dicionário em Python

data = flask.request.get\_json()

# Imprime o conteúdo da requisição

print("Received data:", data)

# Retorna uma resposta HTTP simples

return "OK"

# Verifica se o script está sendo executado como um módulo principal

if \_\_name\_\_ == "\_\_main\_\_":

# Inicia a execução da aplicação

app.run()

Este script demonstra um simples aplicativo Flask em Python, onde:

A primeira linha importa o pacote Flask, que é o coração da aplicação.

Na segunda linha, a variável 'app' é inicializada como uma instância de Flask, passando name como argumento. name é uma variável especial do Python que retorna o nome do módulo corrente.

A seguir, a função handle\_webhook é decorada com @app.route, que especifica a rota (URL) para a qual ela deve responder. No caso, a rota é "/webhook" e a função só responderá a requisições HTTP do tipo POST.

A função handle\_webhook é chamada quando uma requisição POST é enviada para a rota "/webhook". Dentro da função, o corpo da requisição é recuperado com flask.request.get\_json(), que retorna o conteúdo da requisição como um dicionário em Python.

Finalmente, a função imprime o conteúdo da requisição e retorna "OK" como resposta HTTP.

O último bloco de código verifica se o script está sendo executado como um módulo principal, e, nesse caso, inicia a execução da aplicação com app.run(). Isto permite que o script seja executado como um aplicativo standalone, sem ser necessário importá-lo como um módulo em outro script.

Este exemplo cria um servidor Flask que escuta por solicitações HTTP no endpoint /webhook. Quando uma solicitação POST é recebida, a função handle\_webhook extrai os dados da solicitação e os imprime na tela.

Como gerar eventos para um webhook?

Este é um exemplo de como você pode utilizar o enviar eventos para o webhook descrito acima:

import requests

import json

# URL da API webhook

url = "http://localhost:5000/webhook"

# Dados a serem enviados na requisição

data = {"message": "Hello from Python"}

# Enviando a requisição POST para a API

response = requests.post(url, json=data)

# Verificando o status da resposta

if response.status\_code == 200:

# Se o status é 200, a requisição foi bem-sucedida

print("Webhook enviado com sucesso.")

else:

# Caso contrário, houve um erro na requisição

print("Erro ao enviar webhook.")

Este script utiliza o módulo requests para enviar uma solicitação POST para o URL do webhook. Os dados a serem enviados são especificados como um dicionário Python e convertidos para JSON antes de serem enviados na solicitação. A resposta da solicitação é verificada para determinar se o webhook foi enviado com êxito ou não.

(Bônus) Framework Flask

Flask é uma biblioteca de microframework de aplicativos web para a linguagem de programação Python, que permite construir aplicativos web rapidamente e com um mínimo de configurações. É projetado para ser leve e flexível, tornando-o uma boa opção para aplicativos pequenos ou em desenvolvimento. Flask fornece as ferramentas básicas necessárias para construir aplicativos web, como suporte para gerenciamento de rotas, requisições HTTP, sessões, e muito mais, permitindo que você se concentre no desenvolvimento da lógica do seu aplicativo.

Referências

Documentação do GitHub sobre Webhooks: https://developer.github.com/webhooks/

Introdução aos Webhooks: https://sendgrid.com/blog/whats-webhook/

Documentação oficial do Flask: http://flask.pocoo.org/docs/1.0/

Guia rápido de introdução ao Flask: http://flask.pocoo.org/docs/1.0/quickstart/

Tutorial básico sobre o Flask: https://harve.com.br/blog/programacao-python-blog/introducao-e-tutorial-ao-flask-python/

Artigo sobre como criar uma API RESTful com o Flask: https://auth0.com/blog/developing-restful-apis-with-python-and-flask/

Como gerar eventos para um webhook?

Este é um exemplo de como você pode utilizar o enviar eventos para o webhook descrito acima:

import requests

import json

# URL da API webhook

url = "http://localhost:5000/webhook"

# Dados a serem enviados na requisição

data = {"message": "Hello from Python"}

# Enviando a requisição POST para a API

response = requests.post(url, json=data)

# Verificando o status da resposta

if response.status\_code == 200:

# Se o status é 200, a requisição foi bem-sucedida

print("Webhook enviado com sucesso.")

else:

# Caso contrário, houve um erro na requisição

print("Erro ao enviar webhook.")

Este script utiliza o módulo requests para enviar uma solicitação POST para o URL do webhook. Os dados a serem enviados são especificados como um dicionário Python e convertidos para JSON antes de serem enviados na solicitação. A resposta da solicitação é verificada para determinar se o webhook foi enviado com êxito ou não.

(Bônus) Framework Flask

Flask é uma biblioteca de microframework de aplicativos web para a linguagem de programação Python, que permite construir aplicativos web rapidamente e com um mínimo de configurações. É projetado para ser leve e flexível, tornando-o uma boa opção para aplicativos pequenos ou em desenvolvimento. Flask fornece as ferramentas básicas necessárias para construir aplicativos web, como suporte para gerenciamento de rotas, requisições HTTP, sessões, e muito mais, permitindo que você se concentre no desenvolvimento da lógica do seu aplicativo.

Referências

Documentação do GitHub sobre Webhooks: https://developer.github.com/webhooks/

Introdução aos Webhooks: https://sendgrid.com/blog/whats-webhook/

Documentação oficial do Flask: http://flask.pocoo.org/docs/1.0/

Guia rápido de introdução ao Flask: http://flask.pocoo.org/docs/1.0/quickstart/

Tutorial básico sobre o Flask: https://harve.com.br/blog/programacao-python-blog/introducao-e-tutorial-ao-flask-python/

Artigo sobre como criar uma API RESTful com o Flask: https://auth0.com/blog/developing-restful-apis-with-python-and-flask/