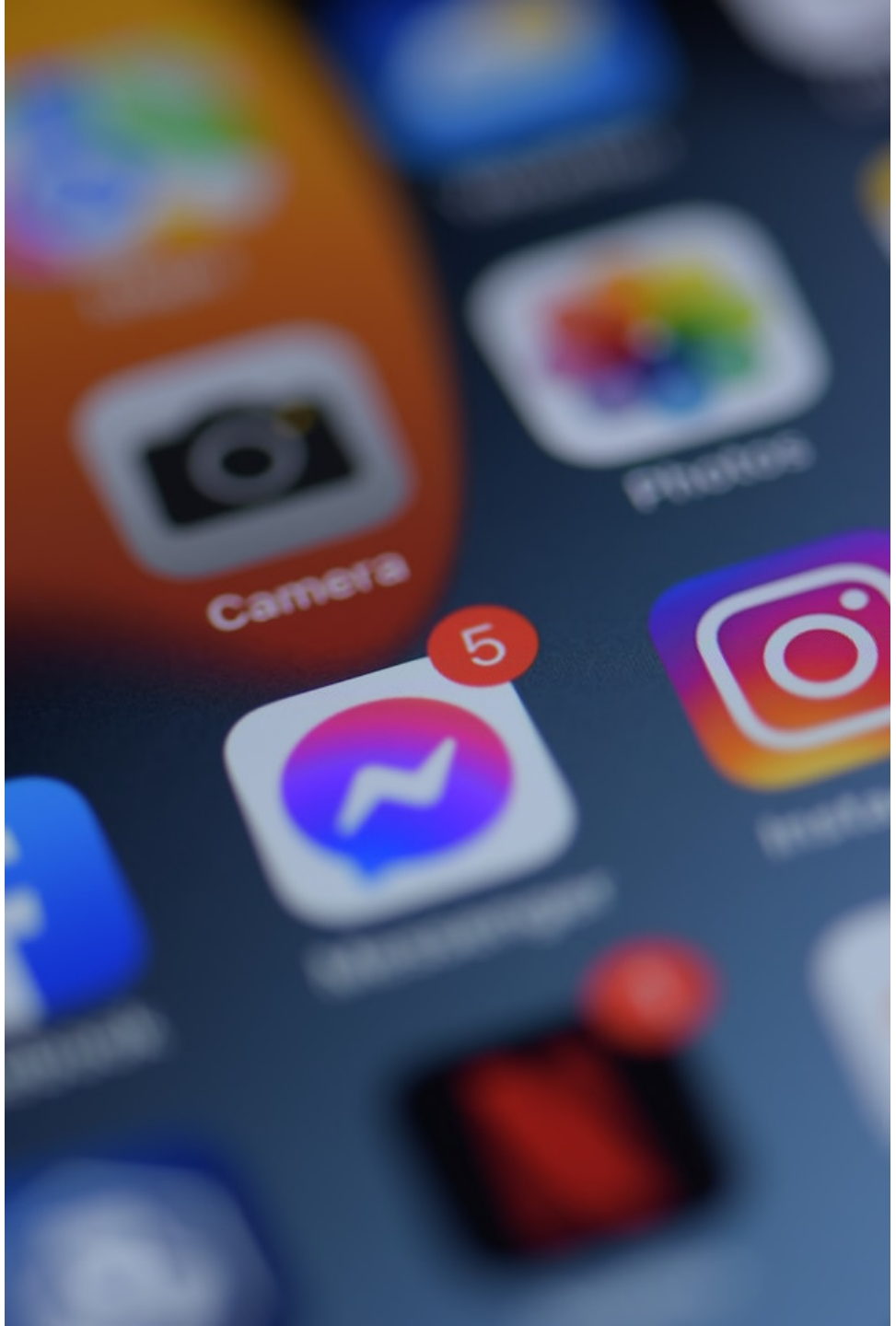


5954025 - Sistemas Distribuídos

Aula 08c - Comunicação via HTTP: Server Sent Events

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Objetivos de Aprendizagem

- Explicar o conceito de *server push* e sua importância em aplicações modernas.
- Entender o funcionamento dos Server-Sent Events (SSE).
- Implementar SSE utilizando Python e FastAPI.
- Projetar soluções baseadas em eventos e microserviços.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Revisando: Arquitetura Cliente-Servidor

- A arquitetura web clássica é baseada no modelo **PULL** (puxar).
  - O Cliente **puxa** a informação do Servidor.
  - A comunicação só ocorre após um gatilho explícito do cliente (o clique, o carregamento da página).
- Isso dificulta comunicação contínua em tempo real.
- Atualizações exigem novas requisições.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Revisando: Alternativas

Problema: Quando precisamos de dados que mudam constantemente (ex: placar de jogo, notificações), esperar que o cliente pergunte a cada segundo é ineficiente.

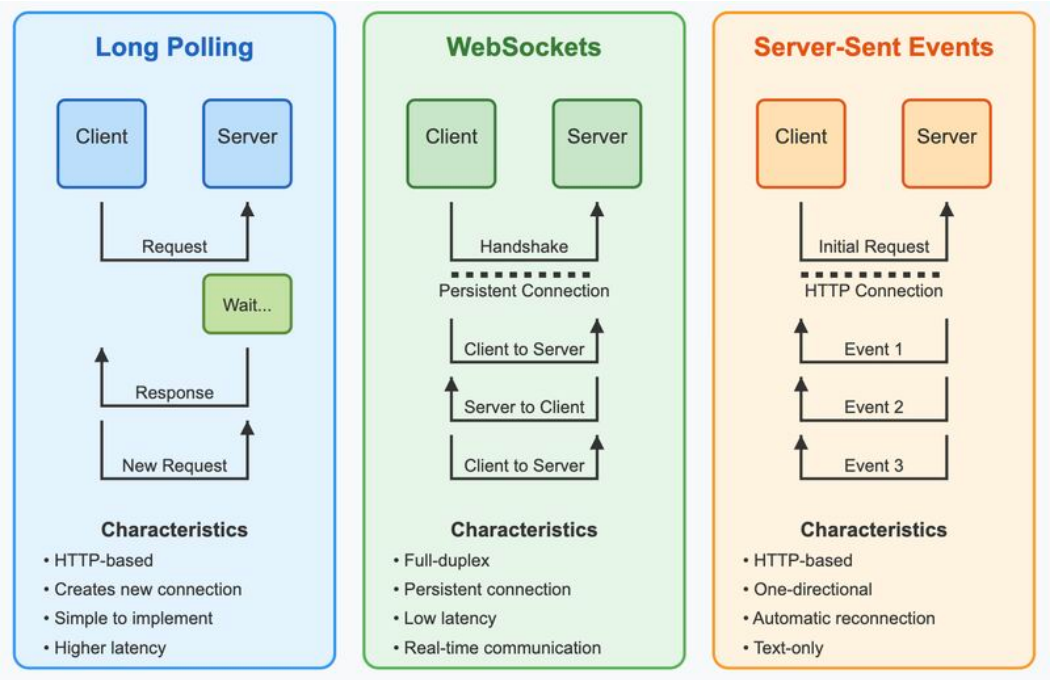


Fig. .1: Diferentes alternativas de comunicação "streaming" entre cliente e servidor. Fonte: https://www.linkedin.com/posts/app-developer_webdevelopment-techeducation-softwareengineering-activity-7324792527901933568-QwEl

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Necessidade de Tempo Real

- **Problema:** Muitas aplicações modernas não seguem o ciclo PULL. Elas exigem que a informação chegue *automaticamente*.
  - Exemplos: Chat em tempo real, placares esportivos ao vivo, notificações bancárias, feeds de notícias instantâneas.
- **Desafio:** Como fazer com que o Servidor "chame" o Cliente sem que ele tenha feito uma requisição?
- **Conceito Chave: Server Push (Empurrar)**. É a capacidade do servidor de iniciar e manter um fluxo de dados para o cliente, independentemente de uma nova requisição explícita.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Server Push

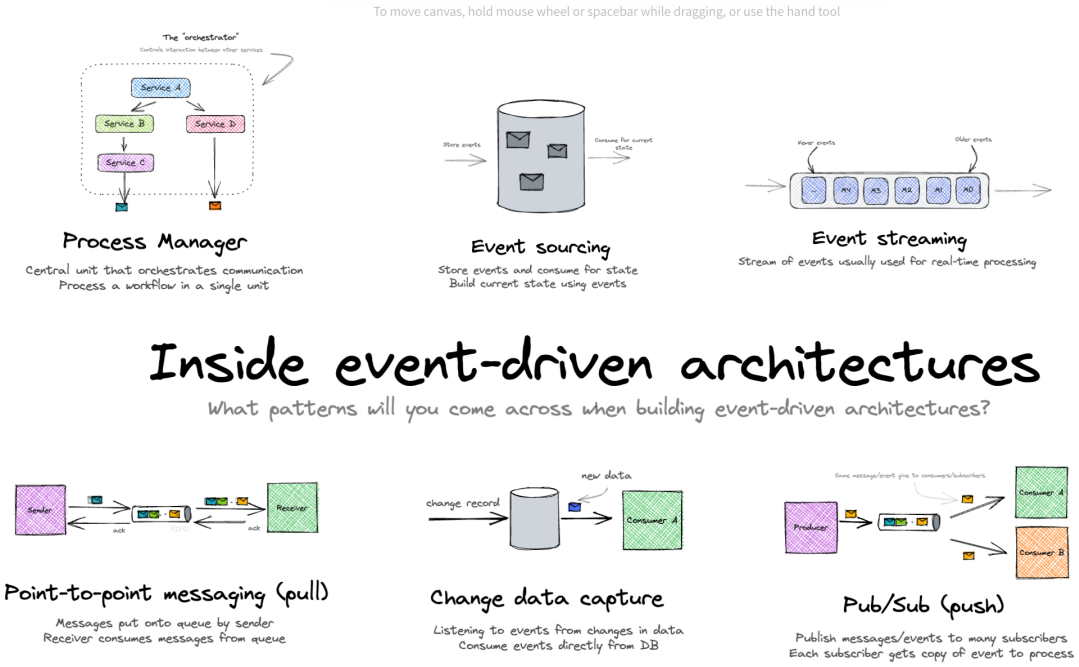
- É a capacidade do servidor de iniciar o envio de dados ao cliente, sem que este precise fazer uma nova requisição.
- Mudança de paradigma: **De Puxar para Empurrar**
- **Exemplo**: Em vez de checar sua caixa de entrada a cada 5 minutos (*Polling*), você recebe uma notificação (*Push*). O servidor "empurrou" a informação para você.
- **Importância**: Permite que as aplicações sejam reativas, respondendo instantaneamente às mudanças do estado do sistema.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Arquitetura Orientada a Eventos

- Sistemas reagem a eventos (mudanças de estado).
- Eventos são fatos **imutáveis** que representam ações ou ocorrências.
- Permitem desacoplamento entre componentes.

> Na imagem: Diferentes arquiteturas orientadas a evento por David Boyne



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Server Sent Events (SSE)

- É um mecanismo HTTP que permite ao servidor enviar dados em stream contínuo para o cliente, mantendo uma única conexão aberta.
- **Funcionamento**: O navegador abre uma conexão e a mantém viva. O servidor simplesmente escreve os eventos no fluxo de saída (stdout) dessa conexão.
  - É mais **simples de implementar** que WebSockets quando você só precisa enviar dados do Servidor para o Cliente (**unidirecional**).
- SSE permite envio contínuo de dados do servidor para o cliente.
- Baseado em texto simples.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

SSE vs. WebSockets: Qual a diferença?

SSE (Server-Sent Events):

- **Direção:** Unidirecional (Servidor $\rightarrow$ Cliente).
- **Protocolo:** HTTP.
- **Uso Ideal:** Feed de notícias, notificações, placares em tempo real.

WebSockets:

- **Direção:** Bidirecional (Servidor $\leftrightarrow$ Cliente).
- **Protocolo:** WS (WebSocket Protocol).
- **Uso Ideal:** Jogos multiplayer, chats em tempo real onde ambos os lados precisam enviar dados constantemente.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Funcionamento do SSE: Como o cliente "Lê" um Evento?

- **MIME Type:** O servidor deve configurar o **Content-Type** como **text/event-stream**. Isso diz ao cliente: "Atenção, este é um fluxo de eventos!"
- **Formato do Dado:** Os dados são enviados em pares chave-valor.
  - **data:**: Contém a informação real que queremos transmitir.
  - **event:** (Opcional): Permite classificar o tipo de evento (ex: **user_login**).
  - **Finalização:** Cada evento deve ser seguido por duas quebras de linha (**\n\n**) para sinalizar ao cliente que um pacote de dados foi concluído.

```
id: 1
event: update
data: {“valor”: 100}
retry: 5000
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

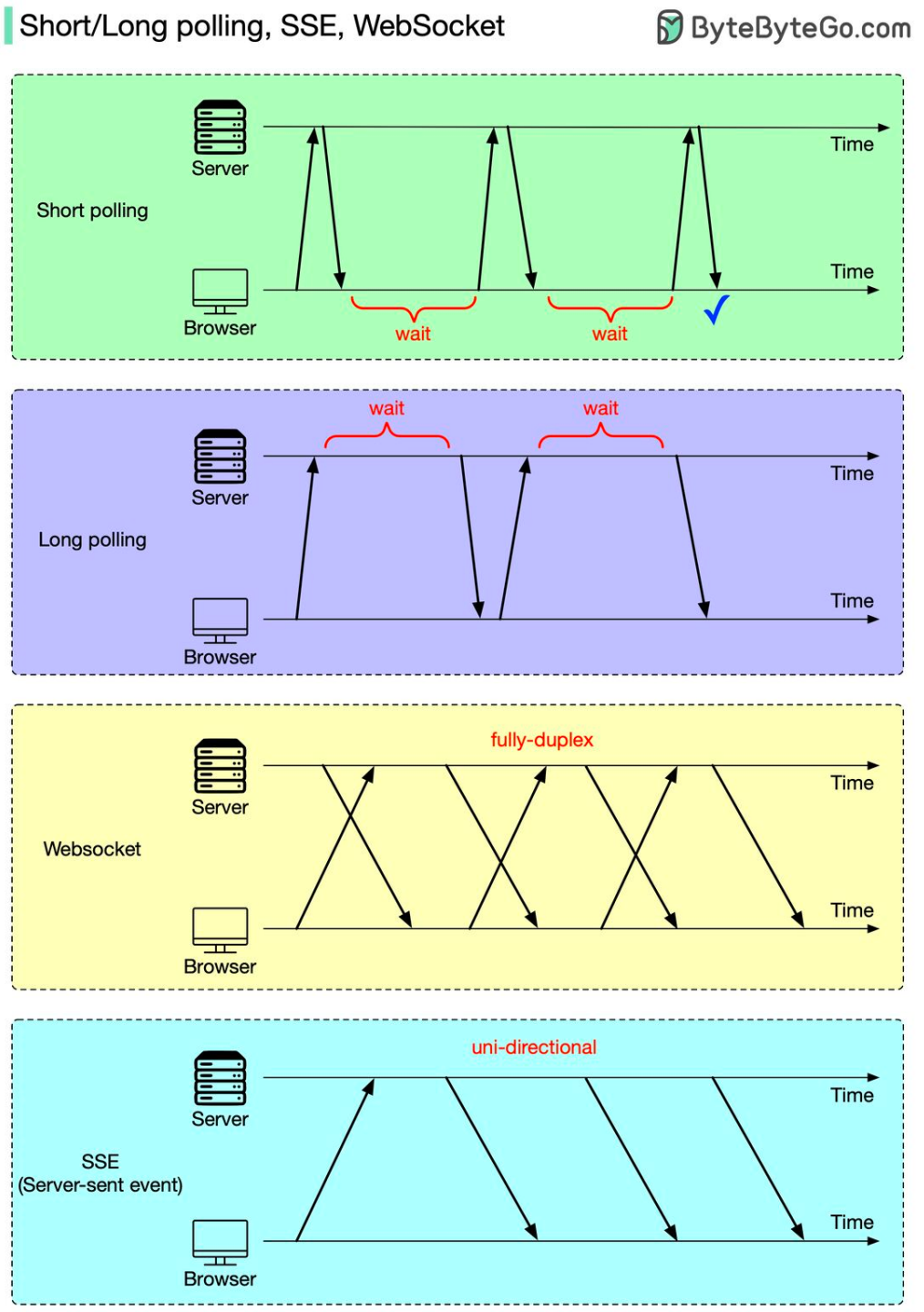
Vantagens e Limitações do SSE

Vantagens:

- Simples de implementar.
- Baseado em HTTP.
- Baixo overhead.
- Ideal para notificações e dashboards.

Limitações:

- Comunicação apenas em uma direção.
- Não serve para aplicações altamente interativas.



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo Prático - Preparação

Criação do venv

```
python -m venv sse
```

```
source sse/bin/activate
```

Instalação do pacote FastAPI

```
pip install fastapi
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo Prático - Servidor FastAPI: app.py

```python
from fastapi import FastAPI
from fastapi.responses import StreamingResponse
from fastapi.middleware.cors import CORSMiddleware
import time
from datetime import datetime
import uvicorn

app = FastAPI()
app.add_middleware(CORSMiddleware, allow_origins=["*"]) # Obrigatório para teste local com HTML

def gerar_dados():
    while True:
        # Obtém a data e hora atual
        agora = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
        # Formato SSE (IMPORTANTE: duas quebras de linha no final)
        yield f"data: {agora}\n\n"
        # Envia a cada 1 segundo
        time.sleep(1)

@app.get("/data")
def enviar_dados():
    return StreamingResponse(gerar_dados(), media_type="text/event-stream")
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo Prático - Cliente HTML e JavaScript: client.html

```html
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <title>Exemplo SSE Simples</title>
</head>
<body>
    <h1>Data Atual (SSE)</h1>
    <p id="data">Carregando...</p>
    <script>
        // Cria conexão com o servidor SSE
        const fonte = new EventSource("http://127.0.0.1:8000/data");
        // Recebe os dados enviados pelo servidor
        fonte.onmessage = function(event) {
            document.getElementById("data").textContent = event.data;
        };
        // Tratamento de erro (opcional)
        fonte.onerror = function() {
            document.getElementById("data").textContent = "Erro na conexão.";
        };
    </script>
</body>
</html>
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo Prático - Executar o Servidor

No terminal, dentro da pasta do projeto, execute:

```
fastapi run app.py
```

Se tudo estiver correto, o servidor vai iniciar (veja mensagem padrão no terminal) e você poderá receber as mensagens na página **client.html**

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Evolução Arquitetural

Problema: Em um sistema grande, mais de um serviço pode gerar eventos.

- Se o serviço A precisa notificar o cliente, e ele chama diretamente o endpoint SSE do Serviço B, há um acoplamento forte. Se o Serviço B cair, todo o fluxo para.
  - Mecanismo de Desacoplamento: Message Queues (Filas de Mensagens)
  - São intermediários robustos e persistentes (como Kafka ou RabbitMQ).
  - Eles atuam como um **"Corretor Universal"** de eventos.
  - Serviço A *publica* uma mensagem no tópico da fila ("Um pedido foi feito"). O Serviço B (e qualquer outro serviço interessado) apenas *escuta* esse tópico e reage quando a mensagem chega.
- Filas modernas (especialmente Kafka) não apenas transportam dados; elas **persistem** o log dos eventos por um período configurável.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Produtor-Consumidor

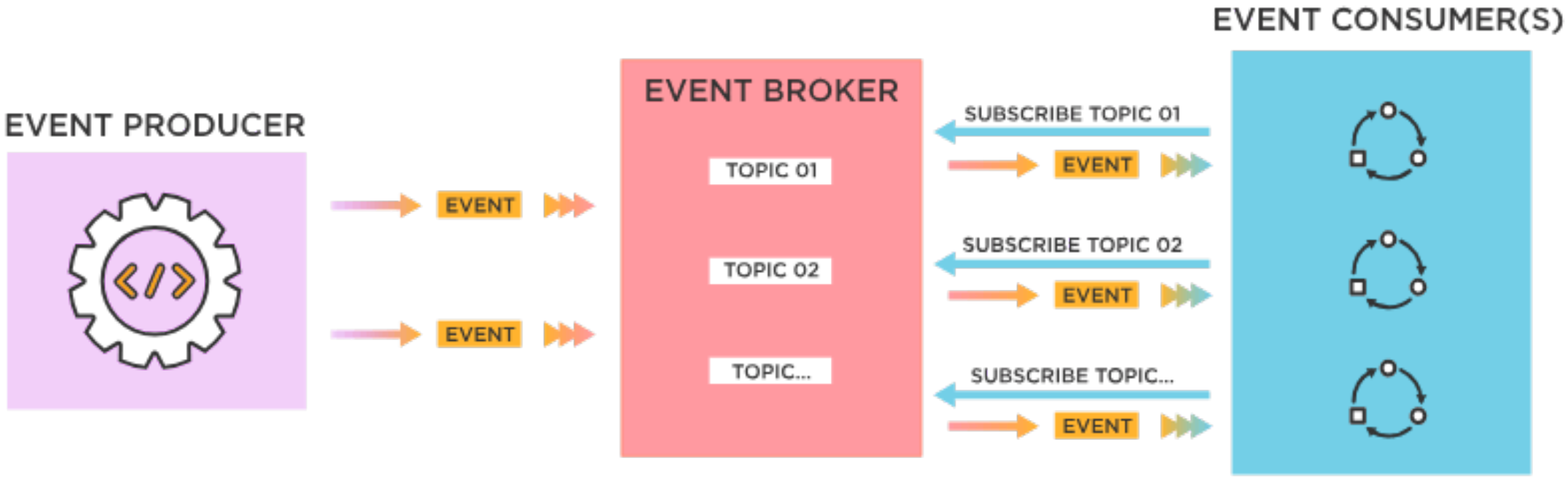


Fig. .1: Exemplo de arquitetura orientada a eventos no padrão Produtor-Consumidor.

Fonte: https://medium.com/@swatikpl44/understanding-event-driven-architecture-with-kafka-fb01c1aa1b43

> Veja mais exemplo completo em https://mpurayil.com/blog/event-driven-architecture-guide-python

Integração com Múltiplos Serviços

Padrão: Uso das filas para conectar serviços independentes que trabalham juntos em um objetivo maior.

Exemplo Prático (Pedido Online):

1. **Serviço Cliente:** Publica evento **PEDIDO_FEITO** na fila.
2. **Serviço Pagamento (Consumidor 1):** Escuta o evento, processa o pagamento e publica um novo evento: **PAGAMENTO_CONFIRMADO**.
3. **Serviço Estoque (Consumidor 2):** Escuta o evento de pagamento, reserva os itens e publica: **ESTOQUE_RESERVADO**.
4. **Serviço Notificação (Consumidor 3):** Escuta o evento de estoque, e *só então* usa SSE para notificar o cliente final.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Conclusão e Próximos Passos

Resumo:

- Server-Sent Events (SSE) permitem comunicação contínua do servidor para o cliente utilizando HTTP.
- SSE é uma abordagem simples, eficiente e adequada para cenários unidirecionais, como notificações e monitoramento.
- A integração de SSE com arquiteturas de microserviços permite construir sistemas escaláveis e reativos.

Material Adicional:

- Introducing Server-Sent Events in Python - Towards Data Science
- MDN Web Docs — Server-Sent Events (EventSource API e exemplos práticos).
- Vídeo Comunicação em tempo real com SSE com Python

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Lab Prático

Implementar um canal de comunicação persistente, simulando um servidor que recebe mensagens e as ecoa de volta para o cliente.

Siga o passo-a-passo em:
https://denmartins.github.io/labs/2026-04-28-sd-sse

> Créditos da imagem: https://unsplash.com/@safarslife

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício e Questões

Exercício Prático

Contexto: Uma empresa deseja desenvolver um sistema simples de acompanhamento de atividades em sua plataforma digital. Esse sistema deve exibir, em tempo real, eventos como:

- Pedido realizado
- Pagamento aprovado
- Erro no sistema

Objetivo: Desenvolver uma aplicação utilizando SSE que permita:

- Envio contínuo de eventos pelo servidor
- Recebimento automático desses eventos pelo cliente
- Exibição dinâmica dos eventos em um feed na interface

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questões para estudo

- Defina o conceito de server push e explique como ele difere do modelo tradicional de comunicação HTTP.
- Compare as técnicas de polling, long polling e Server-Sent Events (SSE), destacando suas principais diferenças em termos de eficiência e uso de recursos.
- Descreva o funcionamento do objeto **EventSource** no JavaScript e seu papel em aplicações SSE.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questões para estudo (cont.)

Uma empresa deseja desenvolver um sistema de monitoramento em tempo real para acompanhar a temperatura de equipamentos industriais. O sistema deve exibir atualizações contínuas em um dashboard web, sem necessidade de interação do usuário.

Com base nesse cenário:

- a) Indique qual tecnologia (Polling, SSE ou WebSockets) é mais adequada e justifique sua escolha.
- b) Explique como essa tecnologia funciona no contexto cliente-servidor.
- c) Discuta uma limitação dessa tecnologia e como ela poderia ser resolvida.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br