

Arquitetura Lambda e Orquestração de Pipelines

Duas Velocidades de Dados: Velocidade da Luz e Velocidade de um Lote

⚡ No mundo do Big Data, precisamos de duas velocidades para processar dados



Speed Layer

+



Batch Layer

=



Serving Layer

Abertura: Duas Velocidades de Dados



No mundo do Big Data, precisamos de duas velocidades: a velocidade da luz e a velocidade de um lote

O QUE É ARQUITETURA LAMBDA?

A **Arquitetura Lambda** é um modelo que estrutura pipelines para lidar com **dados de alta velocidade (streaming)** e **grandes volumes históricos (batch)**. Oferece uma visão completa e precisa ao unir os resultados de ambas as camadas.

Propósito da Arquitetura Lambda



Baixa Latência

Processar dados em tempo real com resposta rápida (milissegundos)



Alta Precisão

Análises históricas completas e precisas com dados consolidados



Equilíbrio

Combinar velocidade e precisão em uma única visão do negócio




Por que Arquitetura Lambda é Crítica?

Muitos sistemas precisam de **alertas imediatos** (ex: fraude em tempo real) E **análises profundas** (ex: previsão de demanda). A Arquitetura Lambda resolve esse dilema ao processar dados em duas velocidades simultaneamente, fornecendo ao negócio a melhor de ambos os mundos.

Arquitetura Lambda: As 3 Camadas

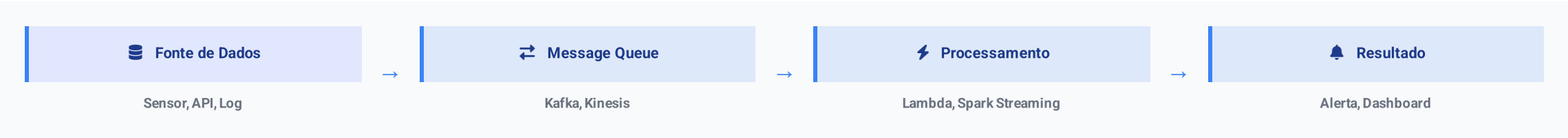


Speed Layer: Processamento em Tempo Real




Speed Layer processa dados em tempo real com **baixa latência**, fornecendo resultados rápidos mas aproximados. É ideal para alertas, dashboards em tempo real e decisões imediatas que não exigem análises complexas.

Fluxo de Processamento em Tempo Real




Tecnologias Principais

**Kafka**


Message broker para streaming de dados em tempo real. Escalável e confiável.

Uso: [Ingestão de eventos](#)

**AWS Lambda**

Computação serverless. Executa código sem gerenciar servidores. Paga-se por execução.

Uso: [Processamento leve](#)

**Kinesis**

Serviço de streaming da AWS. Alternativa ao Kafka. Integração nativa com AWS.

Uso: [Ingestão na nuvem](#)

Batch Layer: Processamento em Lotes



O **Batch Layer** processa o conjunto completo de dados históricos com alta latência, usando ferramentas distribuídas. Fornece resultados precisos e completos, ideal para análises estratégicas e modelos de Machine Learning.



Tecnologias Principais



Apache Spark

Processamento distribuído rápido. Padrão de mercado para Batch Layer



Hadoop / Hive

Processamento em larga escala. Mais lento que Spark, mas robusto



Data Warehouse

Snowflake, BigQuery, Redshift. Otimizados para queries SQL



CARACTERÍSTICAS DO BATCH LAYER

Latência: Horas a dias (alta latência)

Precisão: Alta (dados consolidados)

Análise: Complexa (agregações, ML)

Volume: Grande (histórico completo)

Custo: Alto (processamento contínuo)

Frequência: Diária, semanal ou mensal



Vantagens

- ✓ Alta precisão com dados consolidados
- ✓ Análises complexas e estratégicas
- ✓ Modelos de ML com histórico completo
- ✓ Relatórios detalhados e confiáveis
- ✓ Escalabilidade horizontal com Spark



Desvantagens

- ✗ Latência alta (não serve para alertas)
- ✗ Custo elevado de processamento
- ✗ Requer infraestrutura complexa
- ✗ Não fornece insights em tempo real
- ✗ Requer expertise em Spark/SQL

Orquestração de Pipelines: O que é um DAG?

 **O QUE É UM DAG (DIRECTED ACYCLIC GRAPH)?**

Um **DAG** é um modelo que representa um pipeline como um conjunto de **tarefas com dependências**. Cada tarefa é um nó, e as setas mostram a ordem de execução. **Directed** (tem direção), **Acyclic** (sem ciclos), **Graph** (conjunto de nós e arestas).

Componentes Principais do DAG

 **Tarefas (Tasks)**

- ▶ Unidades de trabalho do pipeline
- ▶ Exemplo: "Carregar CSV", "Limpar Nulos"
- ▶ Podem ser Python, SQL, Bash
- ▶ Cada tarefa é independente

 **Dependências**

- ▶ Regra que define ordem de execução
- ▶ Tarefa B só começa após A terminar
- ▶ Sintaxe: A >> B (Airflow)
- ▶ Garante fluxo correto

 **Exemplo: Pipeline ETL Simples**

Tarefa 1: Ingestão CSV

↓


Tarefa 2: Limpeza Dados

↓

Tarefa 3: Validação

↓

Tarefa 4: Carregamento DW

 **Próximo:** Conheça as ferramentas que implementam DAGs: **Airflow** (padrão de mercado) e **Prefect** (alternativa moderna). Qual escolher para seu projeto?

Ferramentas de Orquestração: Airflow vs Prefect

Aspecto	Airflow	Prefect
Popularidade	Muito popular (padrão de mercado, usado por Netflix, Uber, Airbnb)	Crescente (alternativa moderna, comunidade ativa)
Curva de Aprendizado	Mais íngreme (conceitos complexos, muita configuração)	Mais suave (intuitivo, documentação clara)
Flexibilidade	Muito flexível (customizável para qualquer caso)	Flexível (menos que Airflow, mas suficiente para maioria)
Comunidade	Grande (muitos plugins, respostas rápidas)	Crescente (suporte ativo, melhorando)
Caso de Uso	Pipelines complexos , grandes empresas, múltiplas integrações	Pipelines simples a médios , startups, prototipagem rápida
Instalação	Requer banco de dados, mais dependências	Mais simples, menos dependências

? Quando Usar Cada Ferramenta?

Use Airflow Se...

CENÁRIOS IDEAIS:


- ▶ Pipelines muito complexos com muitas dependências
- ▶ Precisa de alta customização e plugins específicos
- ▶ Trabalha em grande empresa com muitos usuários
- ▶ Precisa de escalabilidade horizontal
- ▶ Tem equipe com experiência em Airflow

Use Prefect Se...

CENÁRIOS IDEAIS:

- ▶ Quer começar rápido com curva de aprendizado suave
- ▶ Pipelines simples a médios (3-10 tarefas)
- ▶ Startup ou prototipagem rápida
- ▶ Prefere código Python puro sem muita configuração
- ▶ Quer melhor experiência de desenvolvimento

Exemplo: Arquitetura Lambda em Saúde

 **Contexto:** Uma clínica precisa de alerta imediato quando o nível de açúcar no sangue de um paciente atinge um pico (> 200 mg/dL), E também de análises históricas precisas para detectar padrões e ajustar o tratamento.

1 Speed Layer: Alerta Imediato (< 1 segundo)

FLUXO

Sensor → Kafka → AWS Lambda → Alerta

LÓGICA

IF glicemia > 200 THEN alerta

VANTAGENS

Latência mínima, custo baixo

LIMITAÇÕES

Análise simples, falsos positivos



2 Batch Layer: Análise Histórica (1-24 horas)

FLUXO

Data Lake → Spark Job → Análises

LÓGICA

Média, padrões, modelo ML

VANTAGENS

Análise complexa, alta precisão

LIMITAÇÕES

Latência alta, custo maior



3 Serving Layer: Dashboard Médico

O MÉDICO VÊ

- ✓ Alerta em tempo real
- ✓ Histórico 30 dias
- ✓ Previsão de risco

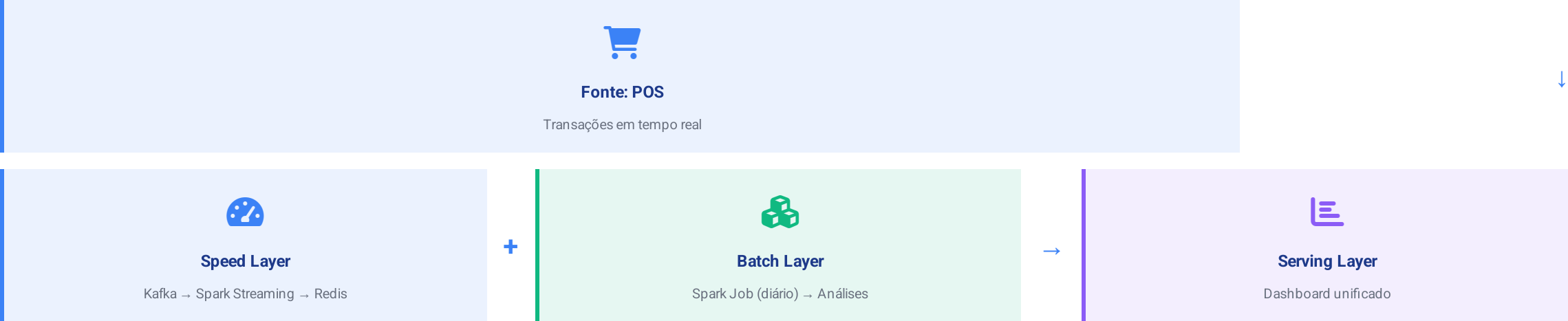
TECNOLOGIAS

Grafana, API REST, Banco de Dados

 **Por que Arquitetura Lambda é Essencial em Saúde?**
Speed Layer garante reação rápida a emergências. Batch Layer garante diagnóstico preciso. Juntos, fornecem ao médico a melhor decisão: agir rápido quando necessário, com base em dados precisos e históricos.

Exemplo: Arquitetura Lambda em Varejo

Cenário: Uma rede de lojas precisa de dashboard em tempo real de vendas (para gerentes) E análises profundas de padrões de compra (para estratégia). Arquitetura Lambda resolve esse dilema.



Speed Layer (Tempo Real)

- ▶ Agregação de vendas (últimos 5 min)
- ▶ Cálculo de GMV por loja
- ▶ Contagem de transações
- ▶ Latência: Segundos

Batch Layer (Histórico)

- ▶ Análise de cesta de compras
- ▶ Segmentação de clientes
- ▶ Previsão de demanda (ML)
- ▶ Latência: Horas/dias

O que cada pessoa vê no Serving Layer?

Gerente de Loja

Dashboard em tempo real: Vendas agora. Alertas: Vendas abaixo da meta.


Diretor de Categoria

Análise de cesta: Quais produtos vendem juntos. Previsão: Quanto estocar.

CFO

Relatório de receita: GMV por período. Previsão: Faturamento próximos 3 meses.

Atividade: Criação de um DAG - Parte 1



Cenário: Vocês precisam criar um pipeline ETL simples: 1) Ingestão de um arquivo CSV de vendas, 2) Limpeza de linhas duplicadas, 3) Validação de qualidade, 4) Armazenamento no Data Warehouse. Defina as tarefas, dependências e documentação.



Passos 1-3: Estruturando o DAG

- 1

Definir as Tarefas (Tasks)


Listar as 4 tarefas necessárias: Ingestão, Limpeza, Validação, Carregamento. Para cada tarefa, descrever: nome, função, entrada e saída esperada.
- 2

Definir as Dependências

Desenhar ou descrever a ordem de execução: Ingestão → Limpeza → Validação → Carregamento. Usar sintaxe: tarefa_1 >> tarefa_2 >> tarefa_3 >> tarefa_4
- 3

Elaborar o Código do DAG (Pseudo-código ou Python)

Escrever a estrutura do DAG em Python ou pseudo-código. Incluir: definição das tarefas, dependências e configurações básicas (schedule, owner, retries).



Próximo: Na continuação, você vai documentar o fluxo, identificar pontos de monitoramento, ver um exemplo completo de código Python e validar sua solução com um checklist.

Atividade: Criação de um DAG - Parte 2

→ **Continuação:** Passos 4-5 da atividade de criação do DAG simplificado

4

Documentar o Fluxo

Criar uma documentação curta (5-10 linhas) descrevendo: propósito do DAG, o que cada tarefa faz, frequência de execução, e contato responsável. Exemplo: "Pipeline que ingere vendas diárias, limpa duplicatas, valida qualidade e carrega no DW. Executa às 2am. Responsável: João Silva."

5

Identificar Pontos de Monitoramento

Definir 2-3 pontos críticos: SLA (tempo máximo), teste de volume (% mínima de linhas), e alerta (quando falhar). Exemplo: "Alerta se Ingestão demorar > 30 min (SLA). Validação falha se dados caírem < 90% do original (volume). Email para João se qualquer tarefa falhar."

</> EXEMPLO: ESTRUTURA COMPLETA DO DAG EM PYTHON

```
# Tarefas
tarefa_1 = Task(funcao="ingestao_csv", timeout=1800, sla=timedelta(minutes=30))
tarefa_2 = Task(funcao="limpeza_dados")
tarefa_3 = Task(funcao="validacao_qualidade", min_volume=0.9)
tarefa_4 = Task(funcao="carregar_warehouse")
# Dependências
tarefa_1 >> tarefa_2 >> tarefa_3 >> tarefa_4
# Configuração e Alertas
dag = DAG(id="pipeline_vendas", schedule="0 2 * * *", owner="joao_silva")
dag.on_failure_callback = send_email_alert
```

✓ CHECKLIST DE VALIDAÇÃO FINAL

- | | |
|-----------------------------------------------------------|-------------------------------------------------------|
| <input type="checkbox"/> Nome do DAG descritivo | <input type="checkbox"/> Propósito claro (1-2 linhas) |
| <input type="checkbox"/> Cada tarefa tem descrição | <input type="checkbox"/> Dependências estão claras |
| <input type="checkbox"/> Frequência de execução definida | <input type="checkbox"/> SLA e alertas configurados |
| <input type="checkbox"/> Contato responsável identificado | <input type="checkbox"/> Testes de qualidade inclusos |

Monitoramento e Controle de Qualidade

Um pipeline sem monitoramento é como um avião sem instrumentos: você não sabe se está indo bem até cair. Monitoramento garante que seus dados fluem com qualidade, pontualidade e confiabilidade.



SLA (Service Level Agreement)

O QUE É?

Acordo de tempo máximo para que uma tarefa execute. Se exceder, gera alerta.

EXEMPLO

Tarefa de ingestão não pode demorar mais de 30 minutos

IMPLEMENTAÇÃO

```
tarefa_1.sla = timedelta( minutes=30 )
```

BENEFÍCIO

Detecta gargalos e atrasos antes que causem problemas



Teste de Volume

O QUE É?

Verifica que não perdemos muitos dados durante transformação

EXEMPLO

Após limpeza, dados não podem cair abaixo de 90% do original

IMPLEMENTAÇÃO

```
if (linhas_finais / linhas_originais) < 0.9: raise  
Exception( "Perdemos dados!")
```

BENEFÍCIO

Detecta bugs de transformação rapidamente



Alertas e Notificações

O QUE É?

Notificação automática quando algo dá errado no pipeline

EXEMPLO

Enviar email se tarefa falhar ou SLA for violado

IMPLEMENTAÇÃO

```
on_failure_callback= send_email_alert on_retry_limit=3
```

BENEFÍCIO

Reação rápida a problemas antes que afetem usuários



Boas Práticas de Monitoramento em DAGs

- ✓ **Defina SLAs realistas:** Baseados em histórico de execução, não em otimismo
- ✓ **Teste volume em cada etapa:** Não apenas no final do pipeline
- ✓ **Monitore qualidade de dados:** Não apenas sucesso/falha de tarefa
- ✓ **Configure alertas em cascata:** Alertas diferentes para severidade diferente
- ✓ **Documente pontos de monitoramento:** Deixe claro por que cada métrica importa

Conclusão: Engenharia de Dados Confiável

✓ A Engenharia de Dados se resume a construir sistemas confiáveis. Arquitetura Lambda fornece o modelo, DAGs fornecem o mecanismo.

💡 Conceitos-Chave Aprendidos

Speed Layer

Processa dados em tempo real com baixa latência. Ideal para alertas e dashboards imediatos, mas com análise limitada.

Batch Layer

Processa dados históricos com alta precisão. Ideal para análises complexas, ML e relatórios estratégicos.

DAG (Orquestração)

Modelo que representa pipelines como tarefas com dependências. Garante execução correta, monitoramento e recuperação de falhas.

Equilíbrio

Arquitetura Lambda une velocidade (Speed) e precisão (Batch) em uma única visão. O melhor de ambos os mundos.

☑ Checklist de Validação: Você Aprendeu?

- ☑ Explicar a diferença entre Speed Layer e Batch Layer
- ☑ Desenhar um DAG com tarefas e dependências
- ☑ Definir SLAs e testes de qualidade em DAGs
- ☑ Identificar quando usar Arquitetura Lambda
- ☑ Comparar Airflow vs Prefect
- ☑ Aplicar Arquitetura Lambda em seu domínio

★ Reflexão Final

Dados são o novo ouro, mas apenas se forem **confiáveis, precisos e acessíveis**. A Arquitetura Lambda e a Orquestração de Pipelines são as ferramentas que transformam dados brutos em **inteligência de negócio**. Domine esses conceitos e você será capaz de construir sistemas de dados que **impactam realmente** o negócio.