

# times-series

## Series temporais

BANCO INFLUXDB

nomes:

Luan B Sarmento

Anthony Guilherme Mucelini

Rafaela Eduarda de Oliveira Barreiros

Jhonatan da Silva Margaf

Gabrieli Cavalcante Boeira

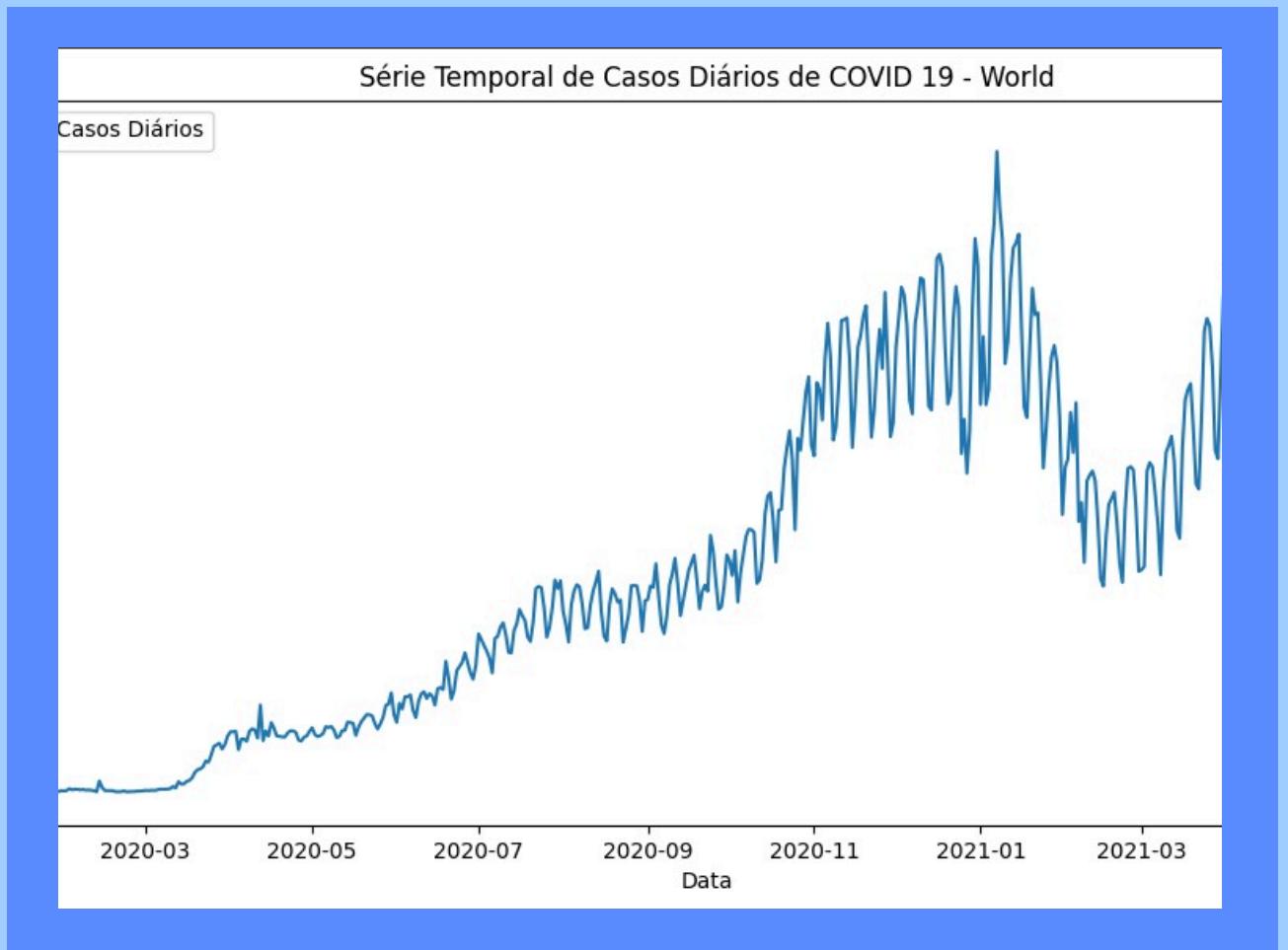


# Motivação

**Séries temporais referem-se a conjuntos de dados que são coletados, registrados ou observados ao longo do tempo.**

**Esses dados são organizados em uma sequência cronológica, onde cada ponto de dados está associado a um momento específico no tempo, de forma periódica e constante.**

**Essa organização permite que os pontos sejam estabelecidos de forma equidistante em gráficos – em intervalos denominados frequências de série – facilitando o estabelecimento de uma correlação entre cada análise.**



# Pontos positivos de sua utilização

## Alta Performance de Escrita e Leitura

- Como funciona: Ele usa uma engine de armazenamento chamada TSM (Time Structured Merge Tree). Diferente de uma B-Tree (usada em SQL tradicional), a TSM é otimizada para adicionar dados sequenciais rapidamente e compactá-los agressivamente.
- Resultado: Você consegue inserir métricas de sensores ou logs de servidores em tempo real com latência de milissegundos.



# Schema Flexível (Schema-on-Write)

- Tags (Indexadas): Metadados para filtrar dados (ex: host, region).
- Fields (Não Indexados): Os dados brutos/valores (ex: temperatura, uso\_cpu).
- Flexibilidade: Se você quiser adicionar uma nova Tag no meio do projeto, basta enviar o dado com ela. O banco se adapta automaticamente.



# Windowing & Downsampling (Consultas por Janelas)

- Agregação: Funções como `window()`, `aggregateWindow()` facilitam calcular médias, máximos ou mínimos em janelas de tempo (ex: média de CPU a cada 5 minutos) sem queries complexas.
- Retention Policies: Você pode configurar o banco para apagar automaticamente dados detalhados após 7 dias, mas manter uma versão summarizada (downsampled) por 1 ano.

```
from(bucket: "sensores")
|> range(start: -1h) // Última hora
|> filter(fn: (r) => r._measurement == "temperatura")
|> aggregateWindow(every: 5m, fn: mean) // Média a cada 5 min
```

# Pontos negativos

- Relações Complexas (SQL/JOINs):
- Problema: Não suporta chaves estrangeiras e tem performance ruim em JOINs. O modelo exige dados repetidos (desnormalizados).
- Busca Textual (Logs/Strings):
- Problema: Não indexa o conteúdo de textos (mensagens de log). Buscas por palavras-chave são lentas e não têm classificação de relevância.



# InfluxDB

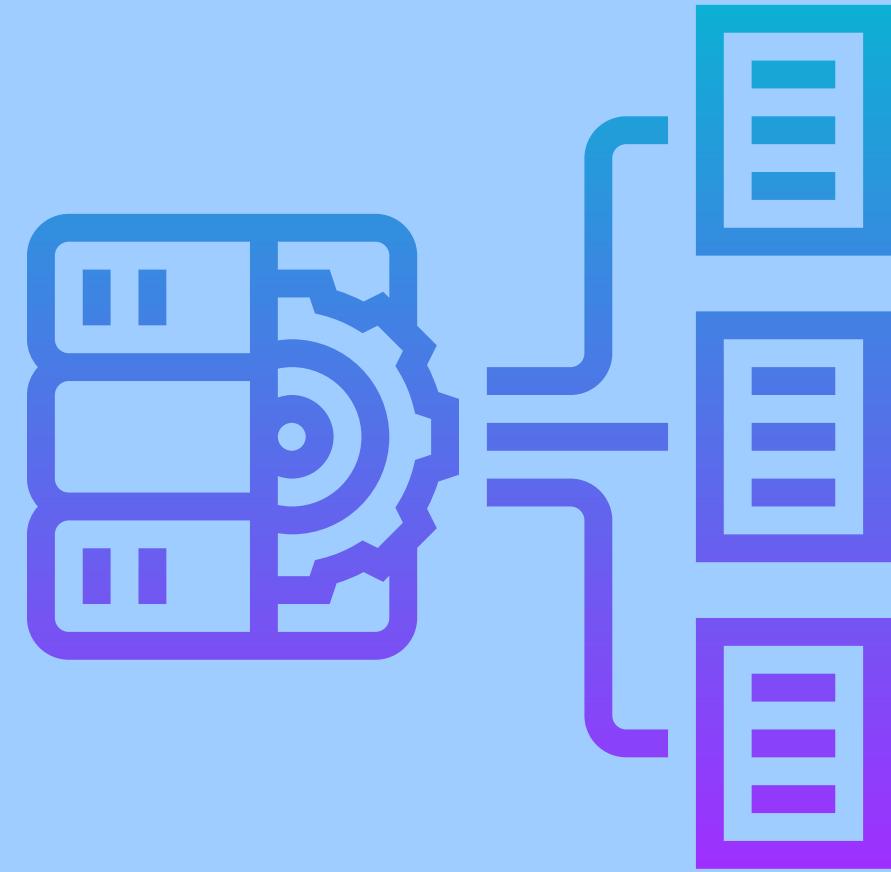
# Uma Abordagem Diferente do SQL Tradicional

O InfluxDB não foi projetado como um banco de dados relacional comum.

Ele é otimizado para uma tarefa específica: ingerir milhões de pontos de dados por segundo.

Para alcançar essa velocidade e eficiência de armazenamento, ele utiliza um modelo de dados onde o tempo é o componente central e indispensável.

Diferente de bancos tradicionais onde chaves primárias são IDs numéricos, no InfluxDB, cada registro de dado deve, obrigatoriamente, estar associado a um carimbo de data e hora exato.

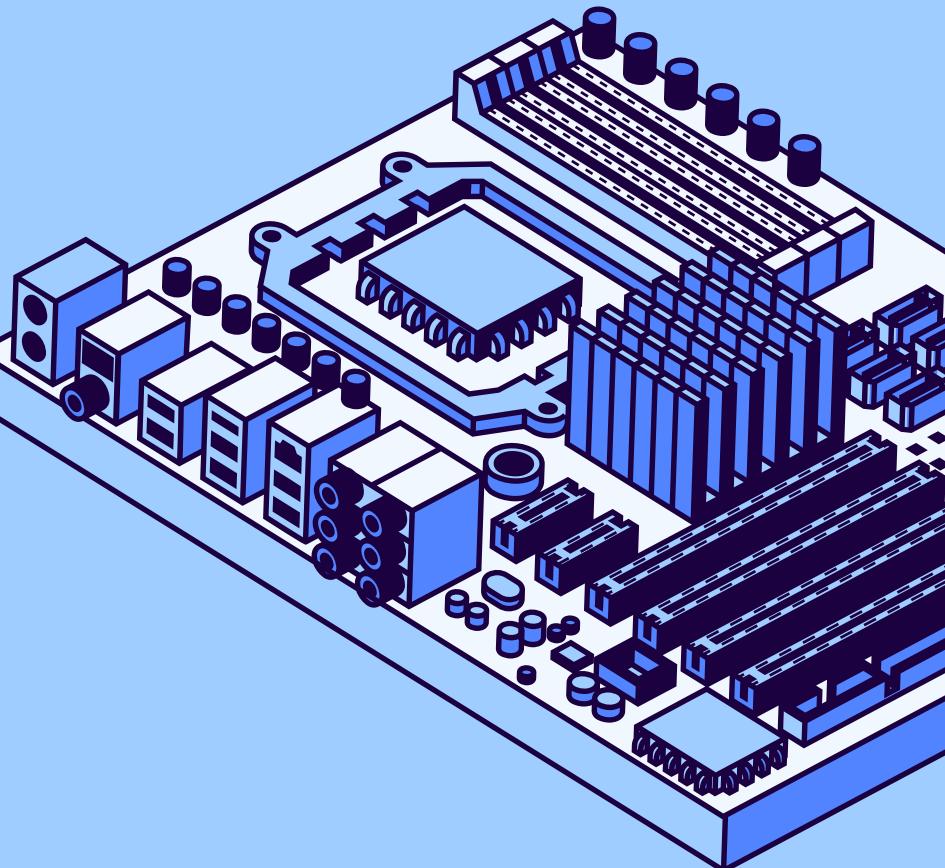


# Os Componentes de uma Série Temporal

O InfluxDB organiza as informações recebidas em quatro componentes principais. Para ilustrar, usaremos um exemplo de uma linha de dados no formato de texto que o banco recebe (Line Protocol), simulando o monitoramento de um servidor:

```
temperatura_servidor,host=servidor01,regiao=us-eastvalor=45.21678886400000000000
```

Esta linha única contém todas as informações necessárias para o banco de dados. Nos próximos slides, vamos detalhar cada uma dessas quatro partes.



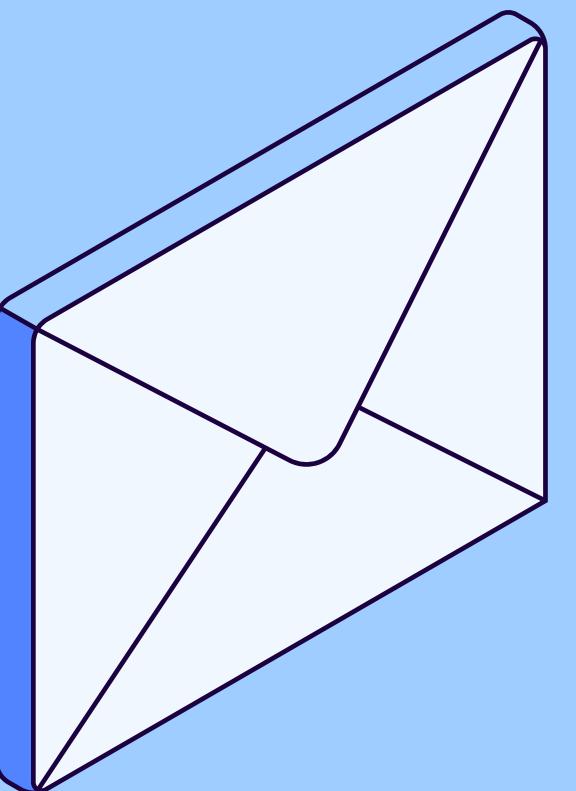
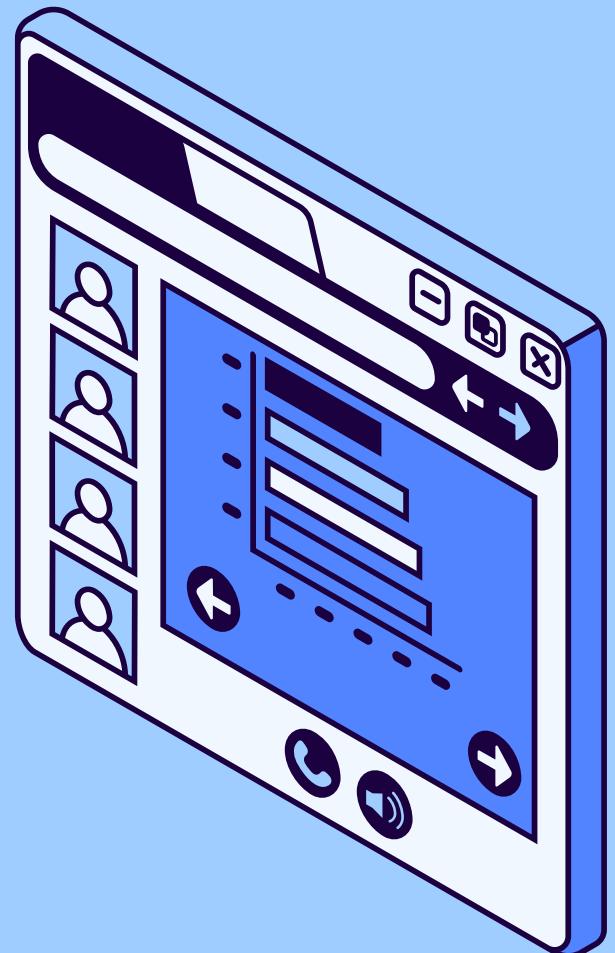
# Timestamp

O Timestamp é o coração de um banco de dados de séries temporais.

No nosso exemplo, ele é representado pelo número longo no final da linha (**16788864000000000000**), que é a data e hora exata em nanossegundos.

Ele funciona como o índice primário universal do InfluxDB.

Diferente de bancos SQL que geralmente buscam primeiro por IDs de clientes ou produtos, o InfluxDB é otimizado para buscar primeiramente pelo "quando" o evento ocorreu.



# Measurement / Medição

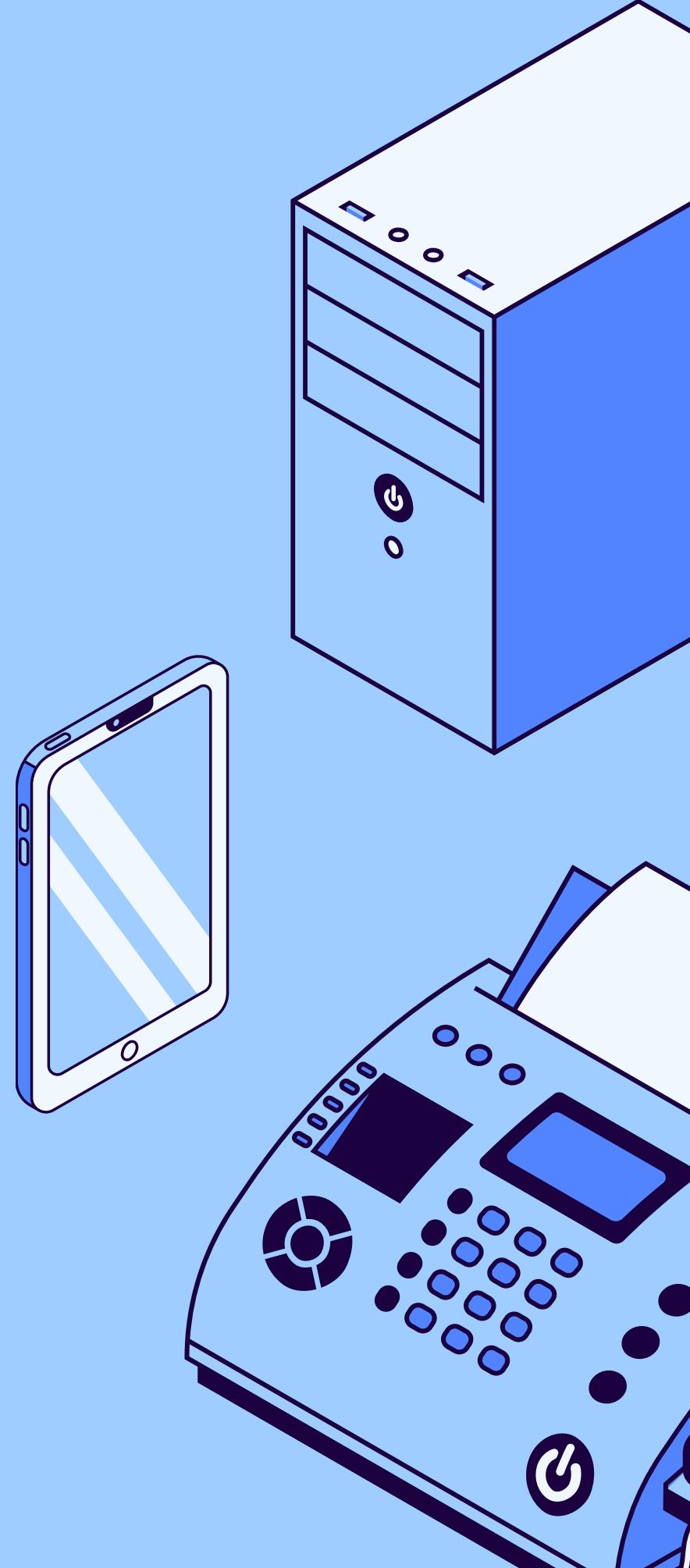
O Measurement é o contêiner de alto nível para os dados que estão sendo coletados.

No nosso exemplo, é a primeira parte da linha: **temperatura\_servidor**.

Ele indica "**o quê**" estamos monitorando de forma geral.

Pode ser comparado, de forma simplificada, ao nome de uma tabela em um banco de dados relacional tradicional.

Todos os dados relacionados à temperatura dos servidores serão agrupados sob este nome.



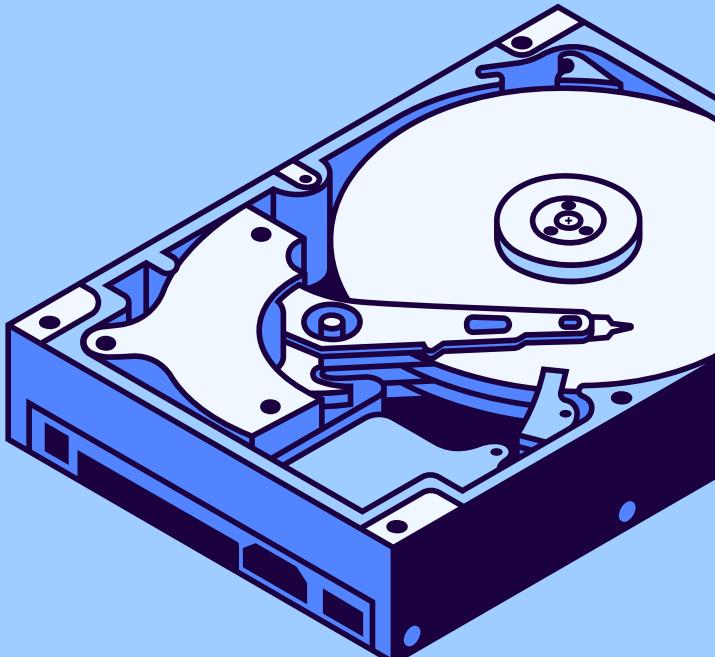
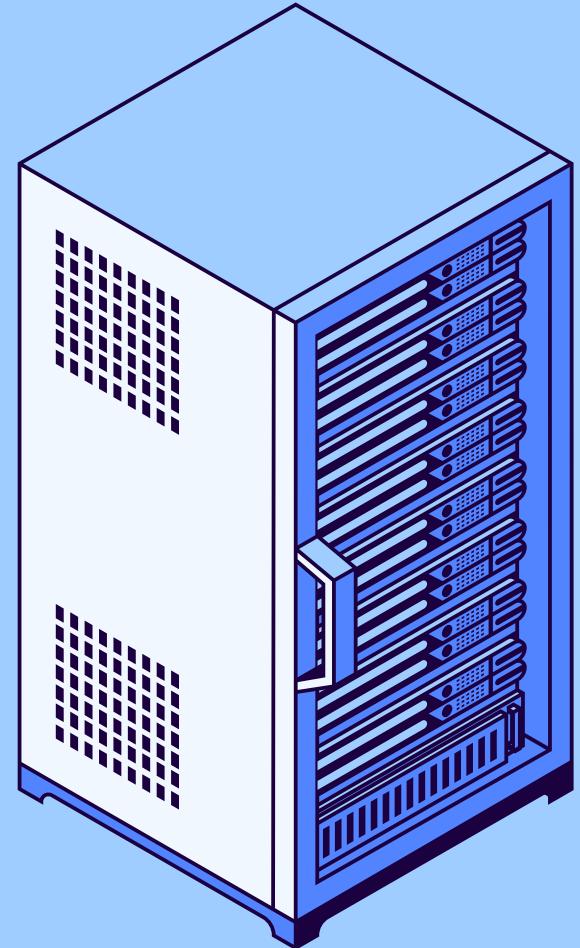
# Tags

As Tags são fundamentais para o modelo. Elas descrevem a origem ou o contexto do dado. São informações que mudam raramente, conhecidas como metadados.

No nosso exemplo, as tags são: **host=servidor01** e **regiao=us-east**.

O ponto mais importante sobre as Tags é que elas são indexadas.

Isso significa que o banco de dados constrói índices específicos para elas, permitindo buscas e filtragens extremamente rápidas (usadas na cláusula **WHERE** das consultas). É através das tags que diferenciamos uma série temporal da outra.



# Fields



Os Fields são os dados brutos que estão sendo efetivamente medidos e que mudam constantemente ao longo do tempo. São os números ou booleanos que queremos analisar.

No nosso exemplo, o field é: **valor=45.2**.

Uma diferença crucial em relação às Tags é que os Fields não são indexados por padrão. O banco é projetado para encontrar a série temporal correta usando as Tags e o Tempo, e então entregar os valores dos Fields associados a ela. Eles são a carga útil da informação.

# Estrutura em Disco



Agora que entendemos o modelo lógico, precisamos entender como o InfluxDB grava isso no disco. O banco precisa resolver dois problemas conflitantes simultaneamente:

1. Alta Taxa de Escrita: Capacidade de receber milhões de dados novos por segundo sem travar o sistema.
2. Armazenamento Eficiente: Guardar esse volume imenso de dados históricos ocupando o mínimo de espaço em disco possível através de compressão.

Para resolver este conflito, o InfluxDB utiliza uma arquitetura de armazenamento em dois estágios principais: o WAL e o TSM.

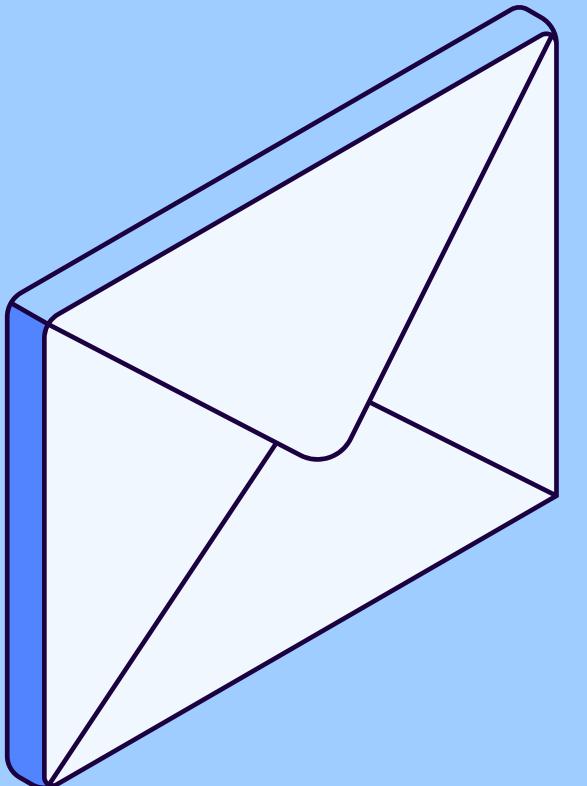
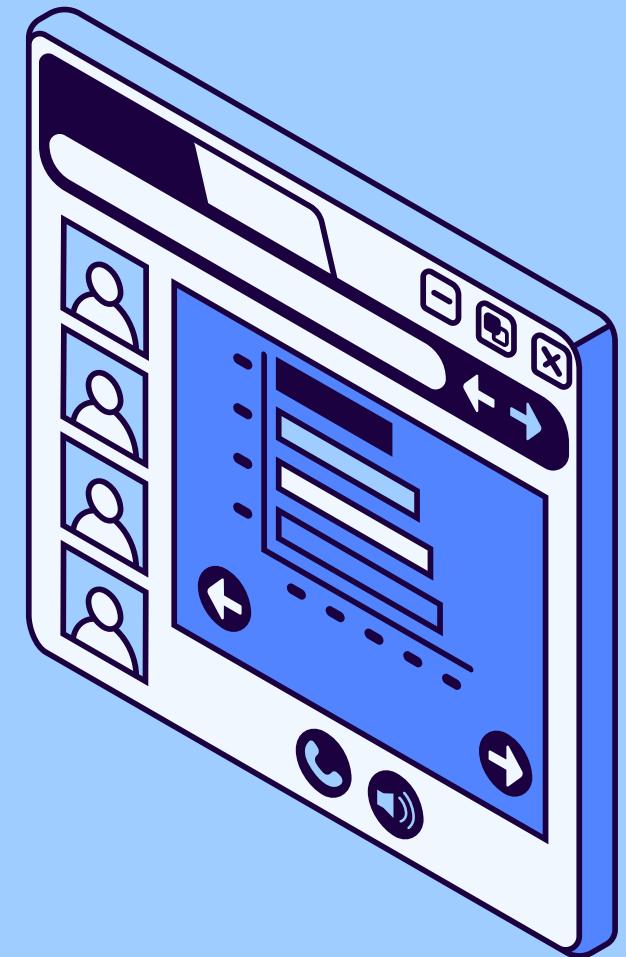
# O WAL

O **WAL** é um arquivo de log sequencial em disco, otimizado exclusivamente para gravações extremamente rápidas.

Quando um dado novo chega ao InfluxDB, ele é gravado na memória **RAM** (para acesso imediato) e, simultaneamente, anexado ao final do arquivo **WAL**.

O objetivo principal do **WAL** é a durabilidade e segurança contra falhas.

Se o servidor desligar subitamente por uma queda de energia, os dados que estavam apenas na memória **RAM** seriam perdidos. O **WAL** funciona como um "**rascunho de segurança**", garantindo que esses dados recentes possam ser recuperados assim que o sistema reiniciar.



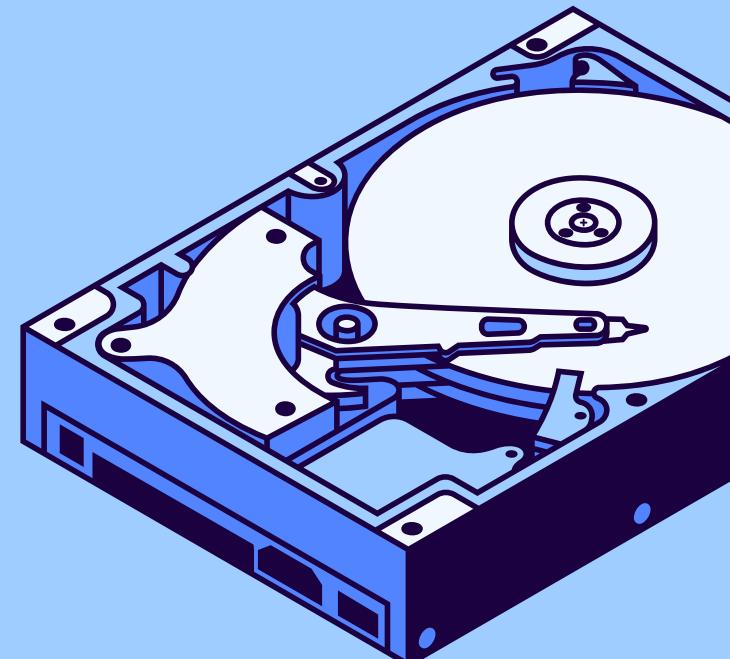
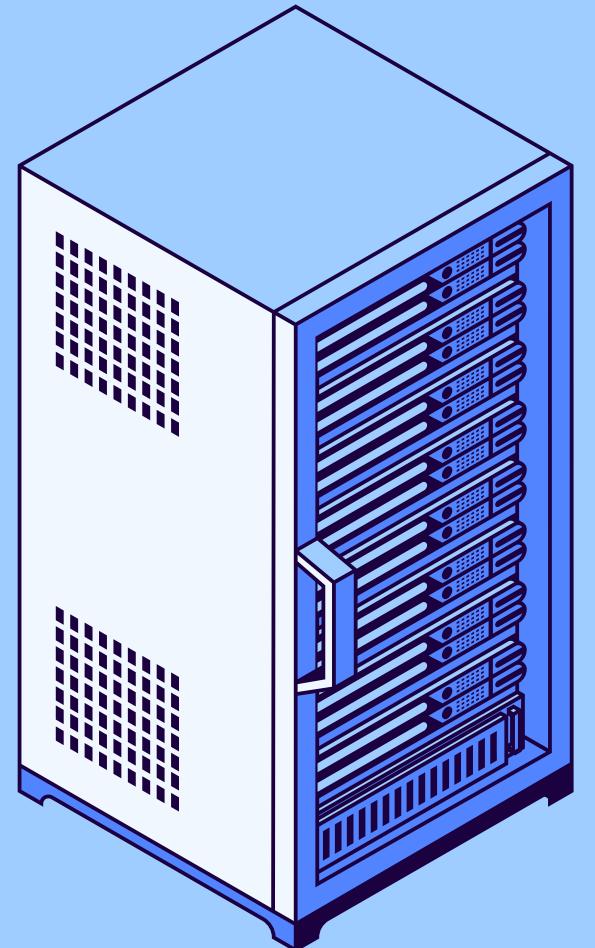
# O TSM

O WAL é rápido, mas não é eficiente em espaço. Por isso existe o segundo estágio. Periodicamente, o **InfluxDB** pega os dados acumulados na memória e no **WAL** e inicia um processo de compactação.

Ele organiza esses dados estritamente por ordem de tempo, aplica algoritmos matemáticos fortes de compressão e os salva em arquivos chamados **TSM**.

Os arquivos **TSM** são o formato de armazenamento de longo prazo. Eles são otimizados para leitura e ocupam muito menos espaço em disco do que os dados brutos.

Uma vez que um arquivo **TSM** é fechado, ele se torna somente leitura, o que torna as consultas a dados históricos muito eficientes e seguras.



# Resumo do Modelo e Estrutura



Em resumo, o **InfluxDB** utiliza um modelo lógico especializado para séries temporais, dividindo os dados em **Measurement** (o contêiner), **Tags** (contexto indexado), **Fields** (valores brutos) e **Timestamp**.

Sua estrutura física resolve o desafio de desempenho usando o **WAL** como uma camada de entrada rápida para garantir que nenhum dado seja perdido, e os arquivos **TSM** para organizar e comprimir os dados para armazenamento eficiente a longo prazo.

# Quando Usar Cada Tipo de Banco?

Existem diferentes tipos de bancos de dados, cada um adequado a necessidades específicas. Por isso, é importante usar um Fit Map para identificar em quais cenários cada tipo funciona melhor e onde não é recomendado.

No Fit Map analisamos fatores como requisitos não funcionais (latência, throughput, consistência, disponibilidade) e características dos dados (volume, velocidade, cardinalidade, retenção). Esses critérios ajudam a decidir se o banco escolhido realmente atende ao conjunto de dados que precisamos armazenar.



# Classificação

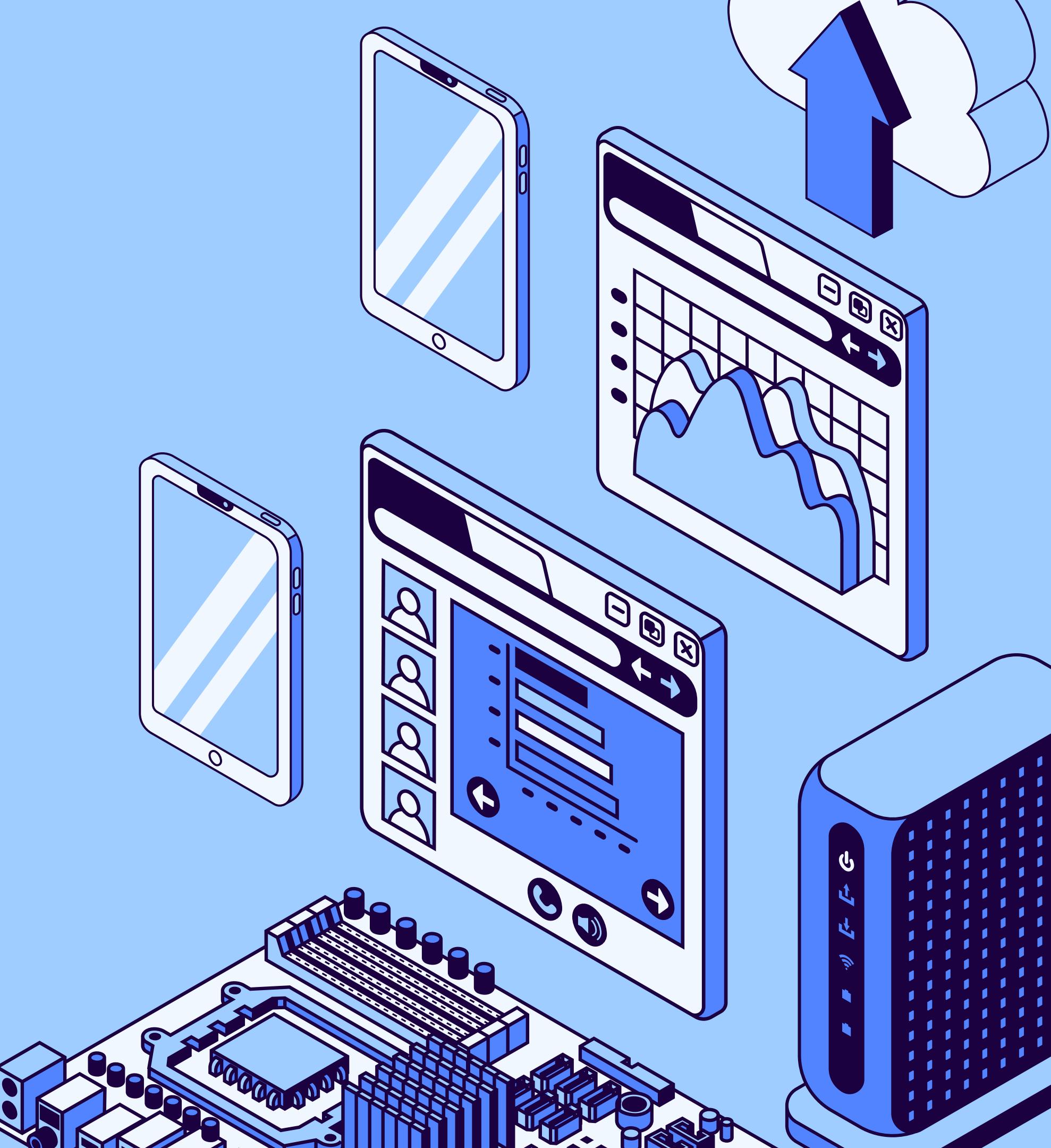
Para facilitar a visualização do Fit Map, a análise foi separada em quatro categorias: Ótimo, Bom, Aceitável e Ruim. Essa divisão ajuda a identificar rapidamente onde o banco de dados se destaca e onde apresenta limitações, tornando o Fit Map mais claro e objetivo.

**Ótimo** → Caso de uso ideal, performance excelente

**Bom** → Funciona bem, sem limitações críticas

**Aceitável** → Funciona, mas há alternativas melhores

**Ruim** → Evitar; modelo não foi projetado para isso



## **Escritas intensivas (alta taxa de ingestão)**

**Ótimo**

InfluxDB é otimizado para ingestão contínua, milhões de pontos por segundo.

## **Time-range scans**

**Ótimo**

Armazena dados temporalmente ordenados e indexados por timestamp.

## **Consultas com agregações por tempo**

**Ótimo**

É exatamente o foco: agregações sobre janelas de tempo usamos GROUP BY time().

## **Janelas móveis (sliding windows)**

**Ótimo**

Muito eficiente para métricas, IoT, logs, telemetria.

## **Acesso direto por chave**

**Bom**

Funciona bem, mas o design é orientado mais ao tempo do que a chave exata.

## **Scans sem filtro temporal**

**Ruim**

Varreduras completas do banco são lentas; não é o propósito.

## **JOINS complexos**

**Ruim**

InfluxDB não suporta JOINs como bancos relacionais; design é baseado em medições independentes.

## **Consultas textuais (full-text search)**

**Ruim**

Não foi feito para isso; pouca eficiência em strings grandes.

# Requisitos Não-Funcionais

Critério	Classificação	Justificativa curta
Latência baixa	ÓTIMO	Projetado para respostas rápidas sobre séries temporais.
Throughput muito alto	ÓTIMO	Engine otimizada para ingestão massiva.
Consistência forte	ACEITÁVEL	InfluxDB prioriza disponibilidade; não é um banco ACID.
Consistência eventual	BOM	Funciona bem em clusters distribuídos.
Alta disponibilidade	BOM	Versão empresarial tem disponibilidade superior a versar OSS.
Retenção de dados (Retention Policies)	ÓTIMO	Possui TTL, compressão e drop automático nativo.
Custo	BOM / ACEITÁVEL	Bom em versão OSS; em cluster ou maior escala pode ficar caro.

# Características dos Dados

Critério	Classificação	Justificativa curta
Heterogeneidade de esquema	BOM	Tags e fields flexíveis; modelo semi-estruturado.
Cardinalidade BAIXA/MÉDIA	ÓTIMO	Índices ficam leves, consultas rápidas.
Cardinalidade ALTA	RUIM	Alta cardinalidade nas tags destrói a performance (ex: ID único por ponto).
Volume alto (big data)	ÓTIMO	Modelo columnar + compressão nativa.
Velocidade muito alta (streams de IoT, logs)	ÓTIMO	Time-series DB foi feito para isso.
Compressão e TTL	ÓTIMO	Retention Policies + compaction automática.

# ANTI-PADRÕES (Onde NÃO usar InfluxDB / Time-Series)

Para finalizar, também é importante destacar os anti-padrões, ou seja, as situações em que o InfluxDB se torna uma escolha inadequada e pode comprometer o desempenho ou a eficiência do sistema.

## JOINS complexos entre tabelas

O modelo time-series é flat, não relacional.

## Cardinalidade extrema nas tags

Ex: criar uma tag usuario\_id para 10 milhões de usuários.

Ex: tag com valores únicos em cada inserção.  
Isso destrói o índice TSI.

## Queries ad-hoc que não dependem de tempo

InfluxDB é eficiente para consultas baseadas em tempo, como:

```
SELECT * FROM cpu WHERE time > now() - 1h
```

Mas péssimo para consultas como:

```
SELECT * FROM vendas WHERE cidade = 'Curitiba'
```

Sem filtro temporal → **scan gigante**.



# InfluxDB

## Demonstração

### IOT monitoramento

Arquivo	Descrição
<code>docker-compose.yml</code>	Sobe o InfluxDB já configurado automaticamente
<code>insert.py</code>	Envia dados contínuos de 3 sensores
<code>README.md</code>	Documentação do projeto
<code>querys (consultas)</code>	Apenas as 3 primeiras consultas utilizadas na demo

- Tags = índices automáticos no InfluxDB
- Tags usadas: `sensor_id`, `location`, `status`
- Fields não são indexados
- Fields usados: `temperatura`, `umidade`, `co2`, `bateria`

# Rodando Docker

```
PS C:\Users\luanb> cd c:/GitHub/influxdb-time-series/demo
PS C:\GitHub\influxdb-time-series\demo> docker-compose up -d
time="2025-11-23T17:12:39-03:00" level=warning msg="C:\\\\GitHub\\\\influxdb-time-series\\\\demo\\\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 1/1
  □ influxdb Pulled                               1.5s
[+] Running 2/2
  □ Network demo_default Created                 0.1s
  □ Container influxdb Started                  0.3s
```

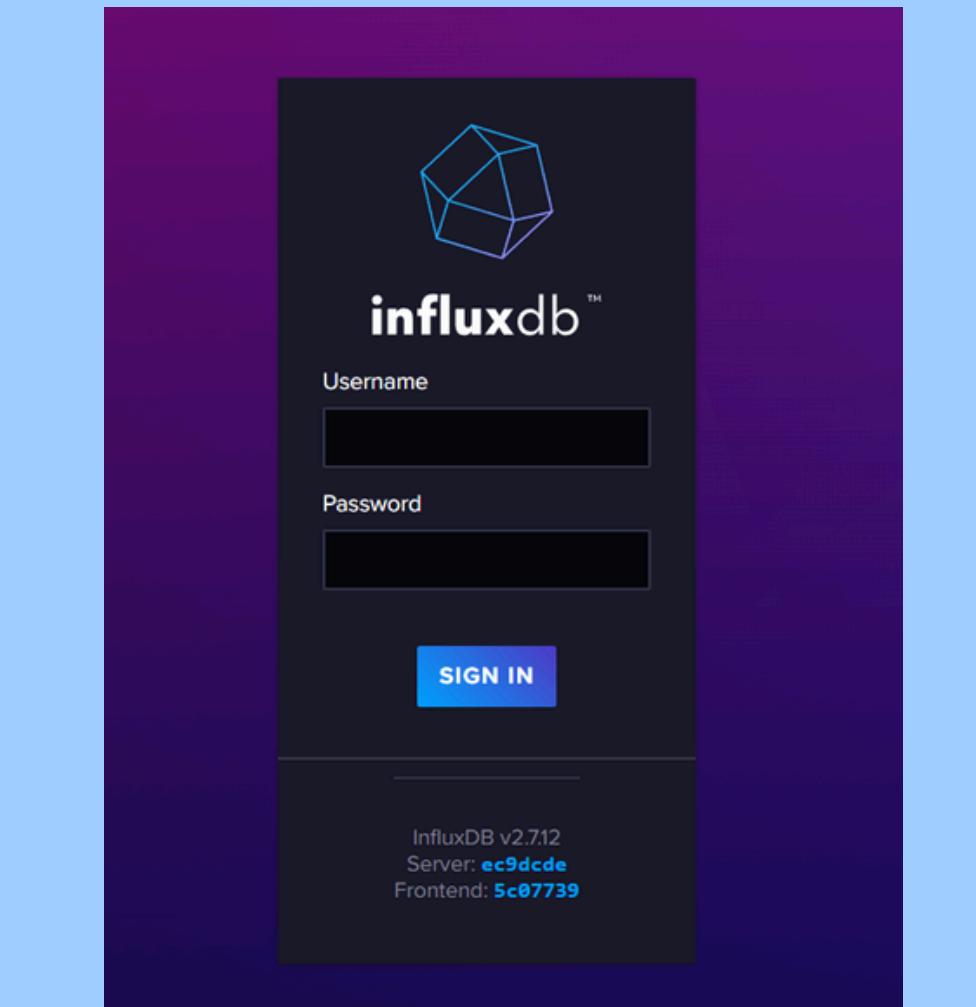
# Rodando Insert.py

```
PS C:\GitHub\influxdb-time-series\demo\scripts> python insert.py
  □ Enviando dados continuamente para o InfluxDB...
Pressione CTRL + C para parar.

2025-11-23 20:16:17.168953 -> dados enviados para 3 sensores
2025-11-23 20:16:18.182456 -> dados enviados para 3 sensores
2025-11-23 20:16:19.191929 -> dados enviados para 3 sensores
2025-11-23 20:16:20.199545 -> dados enviados para 3 sensores
2025-11-23 20:16:21.212130 -> dados enviados para 3 sensores
2025-11-23 20:16:22.230214 -> dados enviados para 3 sensores
2025-11-23 20:16:23.252676 -> dados enviados para 3 sensores
2025-11-23 20:16:24.265614 -> dados enviados para 3 sensores
```

# Acessando InfluxDB localmente

<http://localhost:8086/signin>



```
environment:  
  DOCKER_INFLUXDB_INIT_MODE: setup  
  DOCKER_INFLUXDB_INIT_USERNAME: admin  
  DOCKER_INFLUXDB_INIT_PASSWORD: admin123  
  DOCKER_INFLUXDB_INIT_ORG: demo_org  
  DOCKER_INFLUXDB_INIT_BUCKET: iot_raw  
  DOCKER_INFLUXDB_INIT_RETENTION: 7d  
  DOCKER_INFLUXDB_INIT_ADMIN_TOKEN: demo-token  
  
volumes:  
  - ./influxdb-data:/var/lib/influxdb2
```

Para rodar consultas  
Acessar Data Explorer / Script Editor

A screenshot of the InfluxDB Data Explorer interface. The top navigation bar includes 'Graph' (selected), 'CUSTOMIZE', 'Local' (selected), 'SAVE AS', and other options. The main area is titled 'Data Explorer' with a sub-instruction 'Create a query. Have fun!'. It shows a complex query builder with multiple filter stages. The first stage has 'FROM' set to 'iot\_raw' and 'measurement' set to 'iot\_data'. The second stage has 'field' set to 'battery', 'co2', 'humidity', and 'temperature'. The third stage has 'location' set to 'lab', 'office', and 'warehouse'. The fourth stage has 'sensor\_id' set to 'S3'. On the right side, there are sections for 'WINDOW PERIOD' (set to 'auto (10s)') and 'AGGREGATE FUNCTION' (set to 'mean').

# Clique Aqui para acessar consultas!

## Consulta 1 – Dados dos últimos 15 minutos

```
from(bucket: "iot_raw")
|> range(start: -15m)
```



## Consulta 2 – Filtrar por um sensor (usa TAG / INDEX)

```
from(bucket: "iot_raw")
|> range(start: -1h)
|> filter(fn: (r) => r.sensor_id == "S1")
```



## Consulta 3 – Apenas o campo temperatura

```
from(bucket: "iot_raw")
|> range(start: -1h)
|> filter(fn: (r) => r._field == "temperature")
```



2025-11-23 16:17:44 G...	2025-11-23 17:17:44 G...	2025-11-23 16:18:00 G...	20,11	temperature	iot_data	lab	S1	low
2025-11-23 16:17:44 G...	2025-11-23 17:17:44 G...	2025-11-23 16:18:20 G...	22,11	temperature	iot_data	lab	S1	low
2025-11-23 16:17:44 G...	2025-11-23 17:17:44 G...	2025-11-23 16:18:40 G...	24,84	temperature	iot_data	lab	S1	low
2025-11-23 16:17:44 G...	2025-11-23 17:17:44 G...	2025-11-23 17:16:20 G...	20,42	temperature	iot_data	lab	S1	low
2025-11-23 16:17:44 G...	2025-11-23 17:17:44 G...	2025-11-23 17:16:30 G...	26,20	temperature	iot_data	lab	S1	low
2025-11-23 16:17:44 G...	2025-11-23 17:17:44 G...	2025-11-23 17:16:40 G...	22,53	temperature	iot_data	lab	S1	low
2025-11-23 16:17:44 G...	2025-11-23 17:17:44 G...	2025-11-23 17:17:10 G...	26,51	temperature	iot_data	lab	S1	low
2025-11-23 16:17:44 G...	2025-11-23 17:17:44 G...	2025-11-23 17:17:30 G...	22,53	temperature	iot_data	lab	S1	low
2025-11-23 16:17:44 G...	2025-11-23 17:17:44 G...	2025-11-23 17:17:40 G...	25,63	temperature	iot_data	lab	S1	low



# Otimização e Tuning

## RETENTION POLICIES (RP)

é a regra que define por quanto tempo os dados ficam armazenados no banco.

### O que ela faz:

- Deleta automaticamente dados mais antigos que o período configurado.
- Controla o crescimento do volume de dados.
- Evita que o disco encha e o InfluxDB fique lento.

### Por que isso importa:

O InfluxDB é um banco de séries temporais, ou seja, você grava milhares/milhões de pontos por hora.

Sem retenção, o banco cresce sem parar e:

- consultas ficam lentas
- compactações ficam pesadas
- ocupa muito disco
- pode travar o servidor



# SHARD DURATION

Define o intervalo de tempo que cada shard (bloco de dados) cobre.

Ex: 1 shard = 1 dia, 7 dias, 30 dias etc.

## O que ela faz:

- Divide os dados em blocos menores para facilitar leitura e escrita.
- Afeta diretamente quantos arquivos o InfluxDB precisa gerenciar.
- Organiza os dados em ordem de tempo → performance.

## Por que isso importa:

- Se o shard for pequeno demais, você cria centenas de arquivos que acaba resultando em consultas lentas.
- Se o shard for grande demais, uma consulta pequena pode ter que varrer um período gigante.



# COMPRESSÃO TSM

O formato de armazenamento do InfluxDB (Time-Structured Merge Tree).

## O que ela faz:

- Compacta dados em arquivos otimizados.
- Garante organização por timestamp.
- Mantém alta performance e baixo uso de espaço.

## Por que isso importa:

Sem TSM, o InfluxDB não conseguiria:

- responder consultas rapidamente
- usar pouco disco
- suportar altas taxas de escrita
- agrupar dados por período automaticamente
- InfluxDB chega a comprimir dados em até 80–90%



# CACHE DE TAGS

Uma área de memória usada para armazenar os valores de tags e o índice das séries.

## O que ela faz:

- Acelera filtros por tag (WHERE host='server01').
- Evita reprocessar o índice toda vez que recebe dados.
- Ajuda a identificar séries rapidamente.

## Por que isso importa:

Tags definem cardinalidade (quantidade de séries únicas).

Se tiver tag demais, ou tags com valores únicos, o cache explode e o banco:

- fica lento
- usa muita RAM
- pode travar
- aumenta muito o número de séries



# MÉTRICAS DE MONITORAMENTO

Conjunto de métricas internas para acompanhar o comportamento do InfluxDB

## O que ela faz:

- Permite monitorar:
- uso de CPU e memória
- taxa de escrita
- erros de escrita
- latência de queries
- número de séries (cardinalidades)

## Por que isso importa:

Sem monitorar, você só descobre o problema quando já está tudo travado.

Com métricas, você consegue detectar:

- cardinalidade fora de controle
- compactações atrasadas
- spikes de escrita
- queries lentas
- disco enchendo por falta de retenção



# CONCLUSÃO

A otimização e o tuning no InfluxDB são fundamentais para garantir desempenho, eficiência e estabilidade em ambientes com grande volume de dados. A definição correta de Retention Policies e Shard Duration evita crescimento desnecessário do banco e melhora o desempenho das consultas.

A gestão adequada do cache de tags e o controle da cardinalidade são essenciais para evitar sobrecarga de memória e garantir consultas rápidas. Além disso, o monitoramento contínuo das métricas do servidor permite identificar gargalos e agir preventivamente, garantindo que o banco continue operando de forma estável.



# LINKS PARA REFERÊNCIAS

- <https://github.com/influxdata/influxdb>
- <https://docs.influxdata.com/>
- <https://xitoring.com/pt/blog/how-to-monitor-influxdb-server-performance/>
- [https://docs.influxdata.com/influxdb/v1/query\\_language/manage-database/](https://docs.influxdata.com/influxdb/v1/query_language/manage-database/)
- [https://docs.influxdata.com/influxdb/v1/query\\_language/manage-database/](https://docs.influxdata.com/influxdb/v1/query_language/manage-database/)
- <https://wordpress.ft.unicamp.br/revisa/series-temporaiso>
- <https://docs.influxdata.com/influxdb/v2/write-data/best-practices/schema-design>

