

**FIAP GRADUAÇÃO**

# DATA SCIENCE

BIG DATA ARCHITECTURING & DATA INTEGRATION

Prof. Dr. Renê de Ávila Mendes

# Objetivos da disciplina

**DISCIPLINA:** Big Data Architecturing & Data Integration

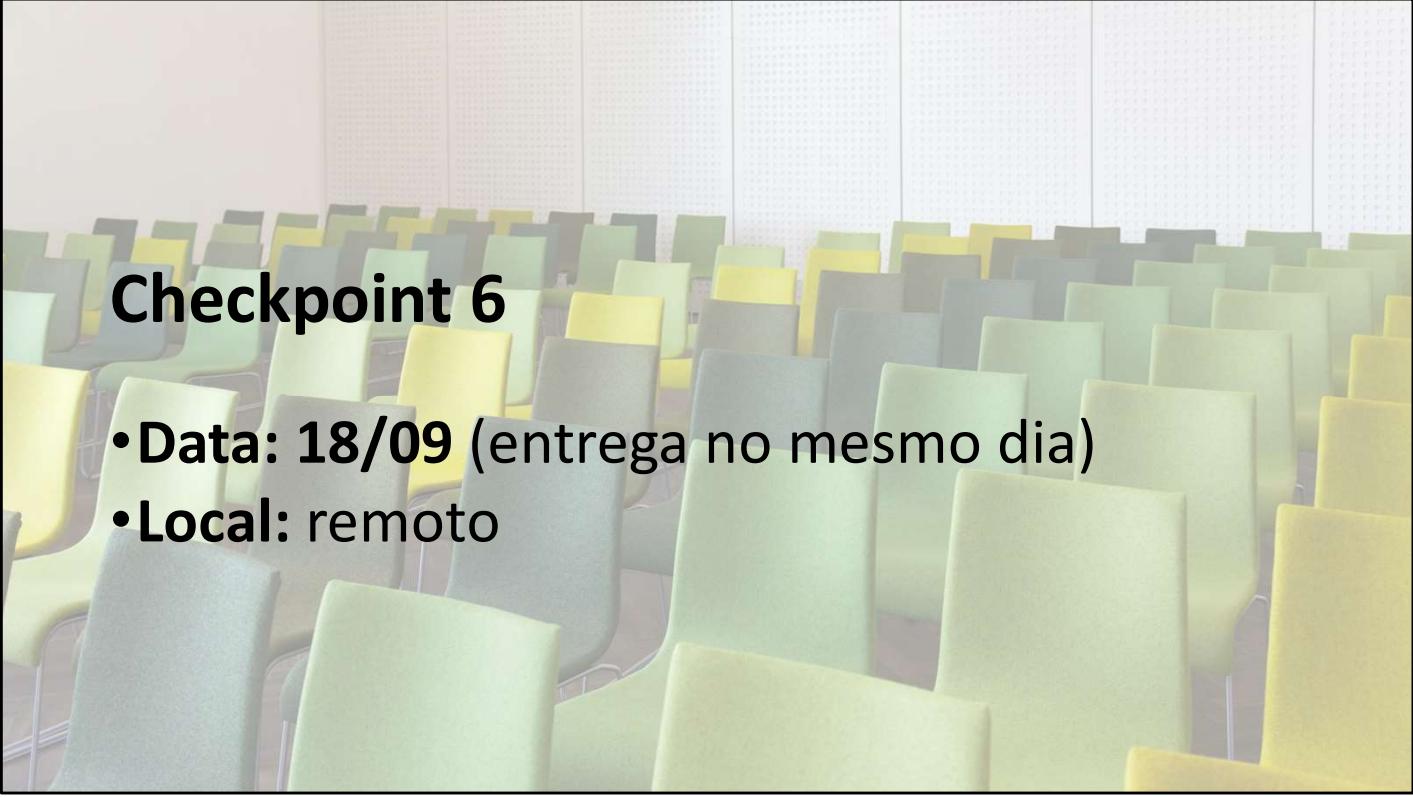
**OBJETIVOS:** Entenda as principais **arquiteturas** para ingestão, processamento e análise de grandes volumes de dados. Conheça as principais **ferramentas** open-source de Big Data como Hadoop, MapReduce, Spark, Sqoop, NiFi, Flume, Kafka, Zookeeper, HBase, Hive e as **integre** com as ferramentas de **extração, transformação e carga** de dados em modelos dimensionais. Entenda conceitos sobre **computação paralela e distribuída**, aplicação do Hadoop e bases Apache e arquiteturas *serverless* e desacopladas. Veja como **visualizar os dados** estruturados ou não estruturados com ferramentas de Self-Service Business Intelligence como PowerBI, utilizando as melhores práticas de **visualização de dados**.

## **Assuntos – 2º Semestre**

- Arquiteturas para Big Data
- Data Pipelines
- Conceito de Data Lake
- PIG, HIVE, Spark, Data Streaming

## Próximas aulas

- **11/09 - Revisão de conteúdo**
- 18/09 - Checkpoint 6
- 25/09 - Aula para preparação para Challenge Sprint 4
- 02/10 - EDM aula 1
- 09/10 - EDM aula 2
- 16/10 - Segurança e Auditoria aula 1
- 23/10 - Segurança e Auditoria aula 2
- 30/10 - LGPD aula 1
- 06/11 - LGPD aula 2
- 13/11 - Kick-off Global Solutions
- 20/11 - Fériado
- 27/11 - Global Solutions
- 04/12 - Substitutiva



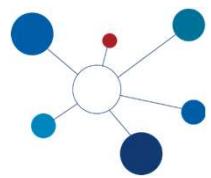
## **Checkpoint 6**

- **Data:** 18/09 (entrega no mesmo dia)
- **Local:** remoto

# REVISÃO DE CONCEITOS



# Alguns conceitos iniciais



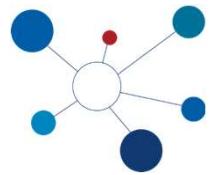
## Sistema

Pode ser definido como aquilo que é feito por pessoas e que possa ser configurado com hardware, software, dados, humanos, processos, procedimentos, facilidades, materiais e outras entidades que ocorram naturalmente ou como uma combinação de **elementos que interagem para alcançar um objetivo**, cujo entendimento pode ser tornado mais claro pela utilização de um substantivo associativo, a exemplo de "sistema de vôo", que pode ser substituído simplesmente por seu sinônimo, tal como avião, helicóptero ou outro sistema capaz de voar.

ISO. *ISO/IEC 12207:2008 Systems and software engineering—software life cycle processes*. [S.I.]: International Organization for Standardization, 2008.

IEEE. *IEEE 15288-2004 Adoption of ISO/IEC 15288:2002 Systems Engineering—System Life Cycle Processes*. [S.I.]: IEEE Computer Society, 2004.

# Alguns conceitos iniciais

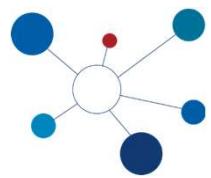


## Arquitetura

Pode ser definida como aqueles **conceitos ou** aquelas **propriedades fundamentais de um sistema** incorporados em seus elementos, relacionamentos ou ainda em seus princípios de projeto e evolução. A definição utiliza a disjunção “ou” para abranger as filosofias de arquitetura como conceito de um sistema na mente de uma pessoa e de arquitetura como percepção das propriedades de um sistema. Neste sentido uma arquitetura é abstrata, requerendo artefatos que a descrevam e documentem, o que é então definido como Descrição da Arquitetura.

ISO. *ISO/IEC 42010:2011 Systems and software engineering — Architecture description*. [S.I.]: International Organization for Standardization, 2011.

# Alguns conceitos iniciais

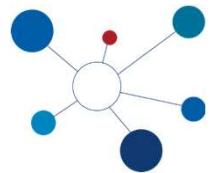


## Framework

No contexto de Tecnologia da Informação, pode ser definido como **classes ou quadros cooperativos que tornam um projeto reutilizável** para uma classe específica de software.

IEEE. *IEEE 1517-2010 IEEE Standard for Information Technology - System and Software Life Cycle Processes - Reuse Processes*. [S.l.]: IEEE Computer Society, 2010.

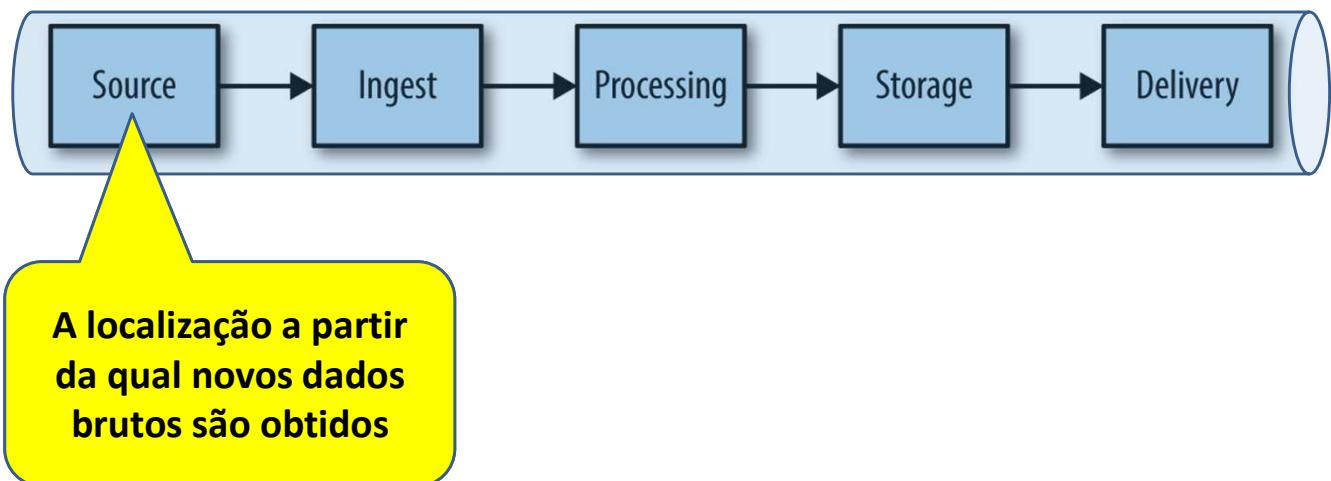
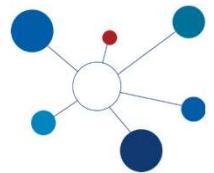
# O pipeline de dados analíticos



Os dados não se formatam por si mesmos para fins de análise. Eles passam por uma série de passos que envolvem a coleta a partir das origens, o tratamento para as formas necessárias à análise e finalmente sua disponibilização na forma de resultados para serem consumidos. Estes passos podem ser pensados como um “tubo”. Este modelo ajuda a entender onde aplicar cada tecnologia.

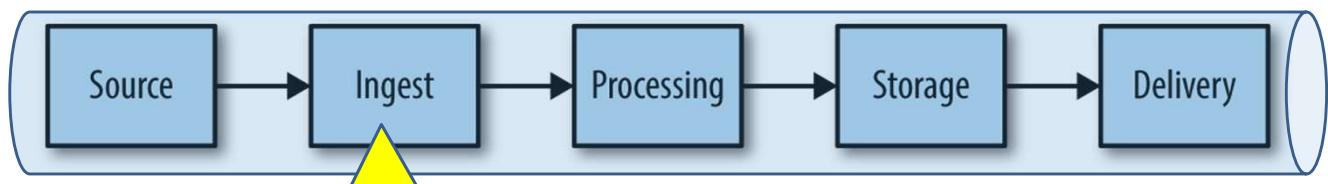
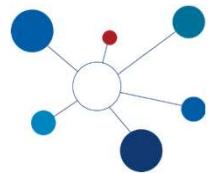
TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

# O pipeline de dados analíticos



TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

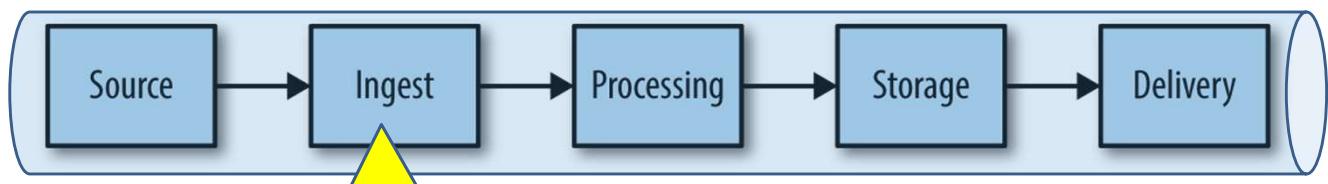
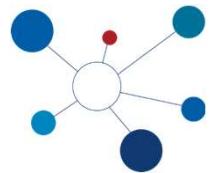
# O pipeline de dados analíticos



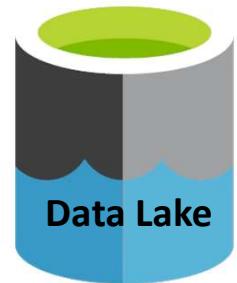
Os processos computacionais responsáveis pelo recebimento dos dados brutos da origem para que possam ser processados.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

# O pipeline de dados analíticos

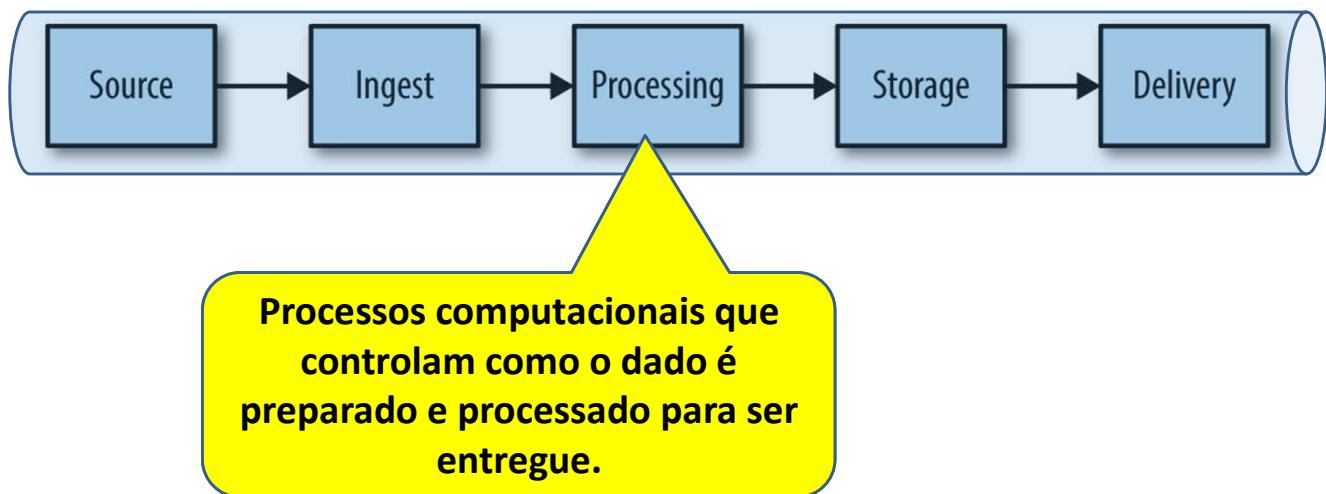
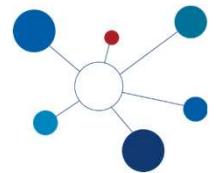


Os processos computacionais responsáveis pelo recebimento dos dados brutos da origem para que possam ser processados.



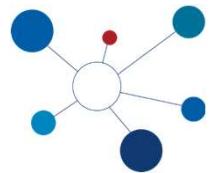
A fase de ingestão deste modelo relaciona-se ao conceito Data Lake, abordado em seguida.

# O pipeline de dados analíticos



TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

# O pipeline de dados analíticos



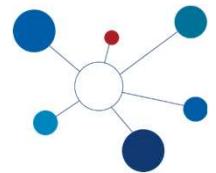
**TRANSIENTE X PERSISTENTE**

Os vários locais onde os dados ingeridos, os dados intermediários e os resultados finais são armazenados.

O armazenamento pode ser transiente (o dado reside na memória somente por um período finito de tempo) ou persistente (o dado é armazenado definitivamente).

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

# O pipeline de dados analíticos

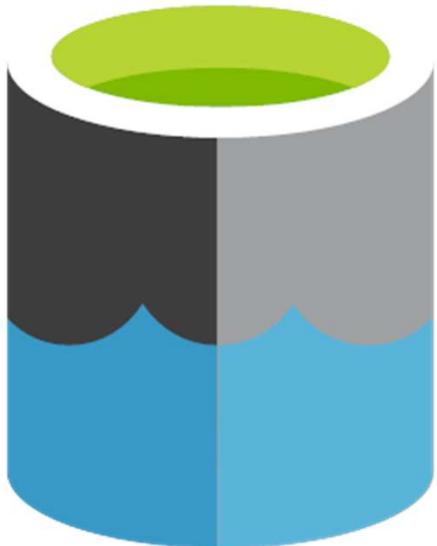


Como o dado é entregue ao consumidor, seja para análise ou para ser consumido por outros processos.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

# Data Lake - Conceituação

Ingest

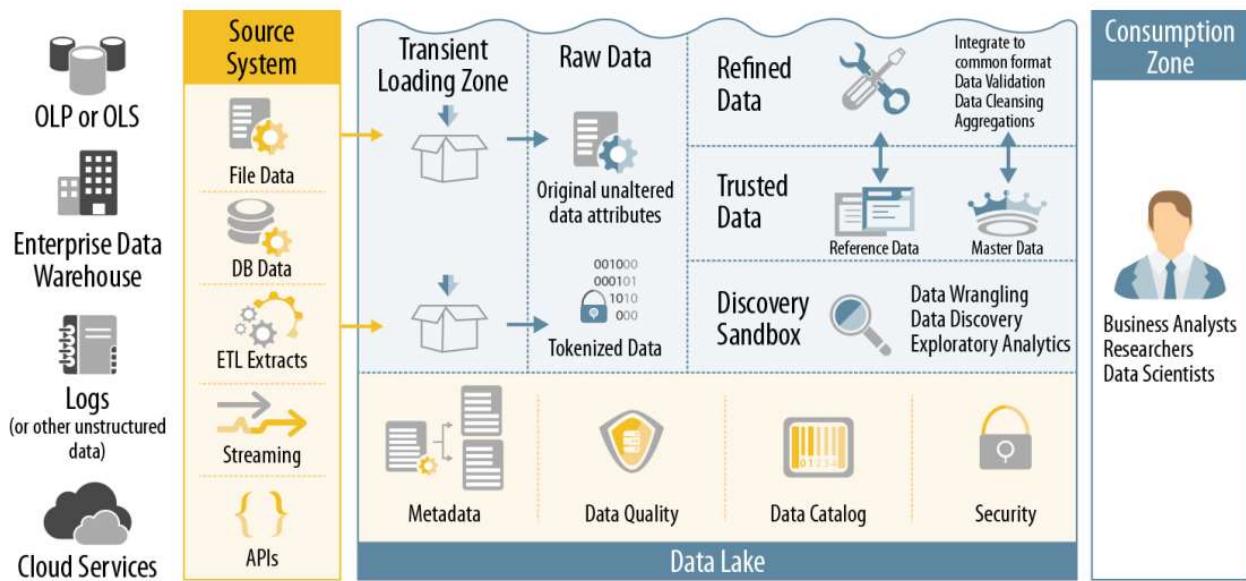


Repositório **centralizado** para armazenamento de todos os dados de uma corporação, independentemente da origem e do formato. Recebe dados estruturados, semiestruturados e desestruturados.

Como todos os dados são bem-vindos, os data lakes são uma abordagem emergente e poderosa para os desafios da integração de dados em um EDW (Enterprise Data Warehouse) tradicional, especialmente quando as organizações se voltam para aplicativos móveis e baseados em nuvem e IoT.

LAPLANTE, Alice e SHARMA, Ben. **Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases**. O'Reilly Media, Inc., 2016.

# Data Lake - Arquitetura

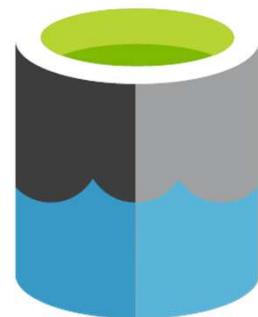


LAPLANTE, Alice e SHARMA, Ben. **Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases**. O'Reilly Media, Inc., 2016, p14.

## Data Lake – Características essenciais

Ingest

Deve ser um repositório único de dados, normalmente armazenado em HDFS (*Hadoop Distributed File System*).



Ao armazenar os dados em HDFS o formato original do dado é preservado e as mudanças do ciclo de vida do dado são capturadas. Esta característica é especialmente útil para atender as necessidades de compliance e de auditoria.

LAPLANTE, Alice e SHARMA, Ben. **Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases**. O'Reilly Media, Inc., 2016.

## Data Lake – Características essenciais

Ingest

Deve ter recursos de orquestração e agendamento de tarefas. Pode utilizar o Apache YARN, que faz parte do framework Hadoop.



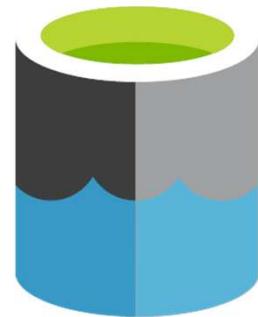
A execução da carga de trabalho é um pré-requisito para o Enterprise Hadoop, e o YARN fornece gerenciamento de recursos e uma plataforma central para fornecer operações consistentes, segurança e ferramentas de governança de dados em clusters Hadoop, garantindo que os fluxos de trabalho analíticos tenham acesso aos dados e ao poder de computação de que precisam.

LAPLANTE, Alice e SHARMA, Ben. **Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases**. O'Reilly Media, Inc., 2016.

## Data Lake – Características essenciais

Ingest

Deve conter recursos (aplicações e *workflows*) para consumir, processar e agir sobre os dados.



O fácil acesso do usuário é uma das marcas de um data lake, devido ao fato de que as organizações preservam os dados em sua forma original. Sejam estruturados, não estruturados ou semiestruturados, os dados são carregados e armazenados como estão. Os proprietários de dados podem facilmente consolidar dados de clientes, fornecedores e operações, eliminando obstáculos técnicos — e até mesmo políticos — ao compartilhamento de dados.

LAPLANTE, Alice e SHARMA, Ben. **Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases**. O'Reilly Media, Inc., 2016.

## Data Lake – Uso

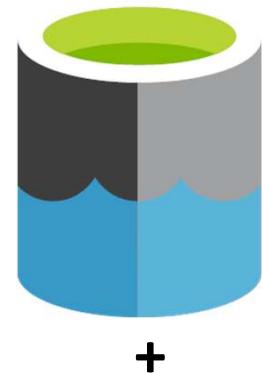
Ingest



LAPLANTE, Alice e SHARMA, Ben. **Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases**. O'Reilly Media, Inc., 2016.

Ingest

## Data Lake – Uso



+

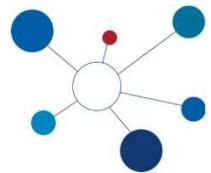


!

**Governança  
de Dados**

LAPLANTE, Alice e SHARMA, Ben. **Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases**. O'Reilly Media, Inc., 2016.

## Tipos de processamento



**Lote (*batch*)**

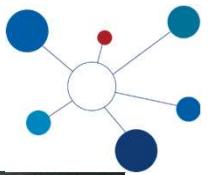
**Fluxo (*stream*)**

Uma observação final importante: o conceito de data lake como é usado hoje é destinado ao processamento em lote, onde a alta latência (tempo até os resultados ficarem prontos) é apropriada. Dito isso, o suporte para processamento de latência mais baixa é uma área natural de evolução para data lakes, portanto, essa definição pode evoluir com o cenário tecnológico.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

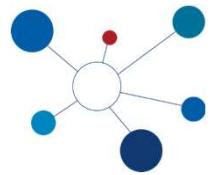
# Tipos de processamento

- Processamento em lote

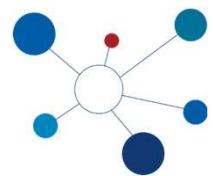


# Tipos de processamento

- Processamento em fluxo

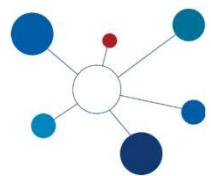


# Tipos de processamento



- Processamento em lote
  - Computa o **volume completo de dados** gerando visões a partir dele
  - Os volumes de dados são **grandes**
  - As **restrições de tempo** de resposta para a análise são menos exigentes
- Processamento em fluxo
  - Computa apenas o **volume de dados mais recente**
  - Requerido por aplicações de análise em **tempo real, ou quase real**
  - A variável tempo de resposta, ou latência, torna-se um **fator crítico** para o sucesso

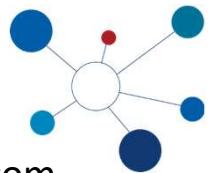
## **Big Data – Por que uma arquitetura ?**



- Necessidade de inserir e pesquisar grandes volumes de dados com uma latência aceitável;

A escolha de arquitetura para sistemas Big Data surge da necessidade de inserir e pesquisar dados em grandes volumes e com uma latência aceitável para o contexto do sistema. Mesmo o escalamento dos tradicionais bancos de dados e até mesmo a inclusão de recursos de processamento assíncrono não são capazes de responder satisfatoriamente a problemas tais como corrupção de dados, tolerância a falhas e alta disponibilidade.

(MARZ; WARREN, 2015).



## Big Data – Por que uma arquitetura ?

- Necessidade de inserir e pesquisar grandes volumes de dados com uma latência aceitável;
- O escalamento dos tradicionais bancos de dados e mesmo a inclusão de recursos de processamento assíncrono não respondem satisfatoriamente a problemas de corrupção de dados, tolerância a falhas e alta disponibilidade;

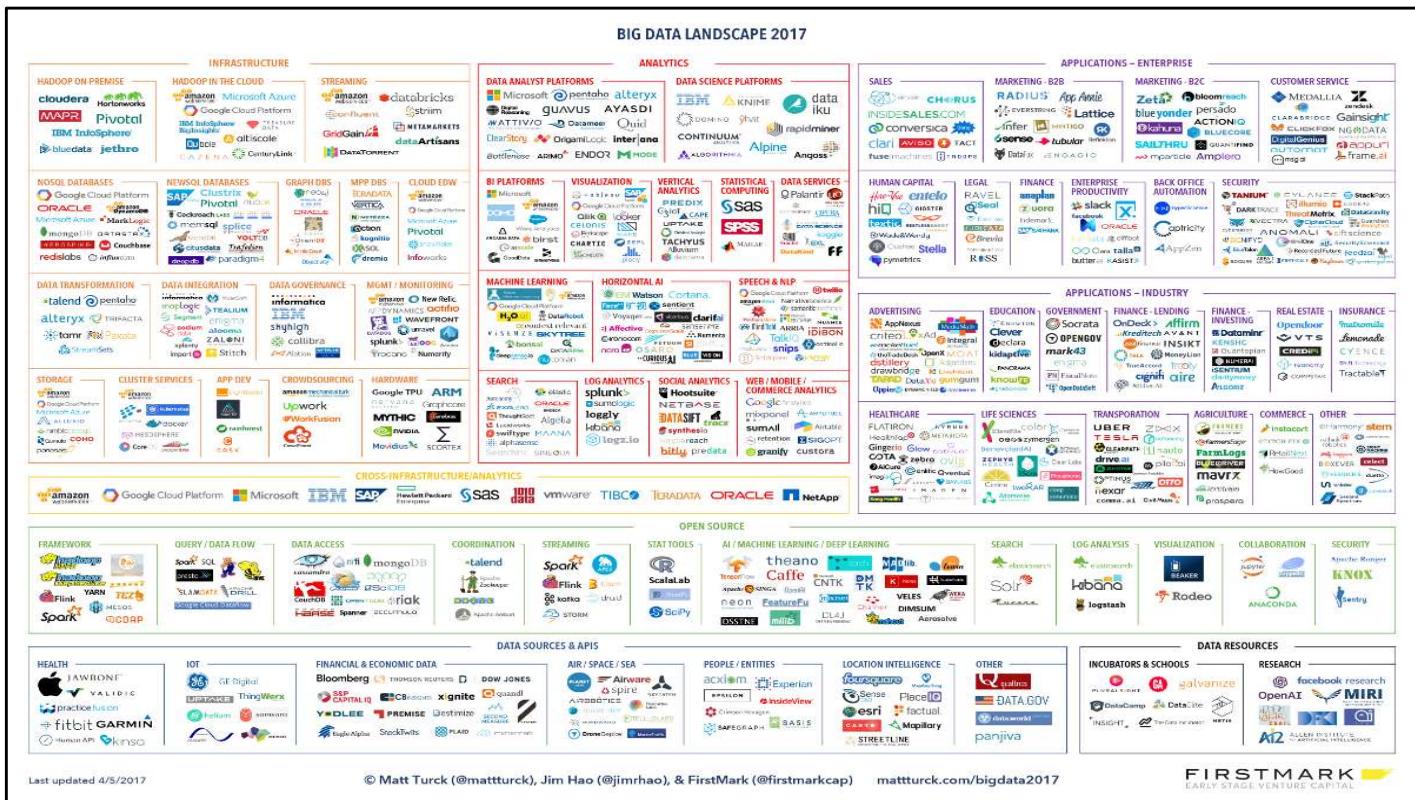
A escolha de arquitetura para sistemas Big Data surge da necessidade de inserir e pesquisar dados em grandes volumes e com uma latência aceitável para o contexto do sistema. Mesmo o escalamento dos tradicionais bancos de dados e até mesmo a inclusão de recursos de processamento assíncrono não são capazes de responder satisfatoriamente a problemas tais como corrupção de dados, tolerância a falhas e alta disponibilidade.

(MARZ; WARREN, 2015).

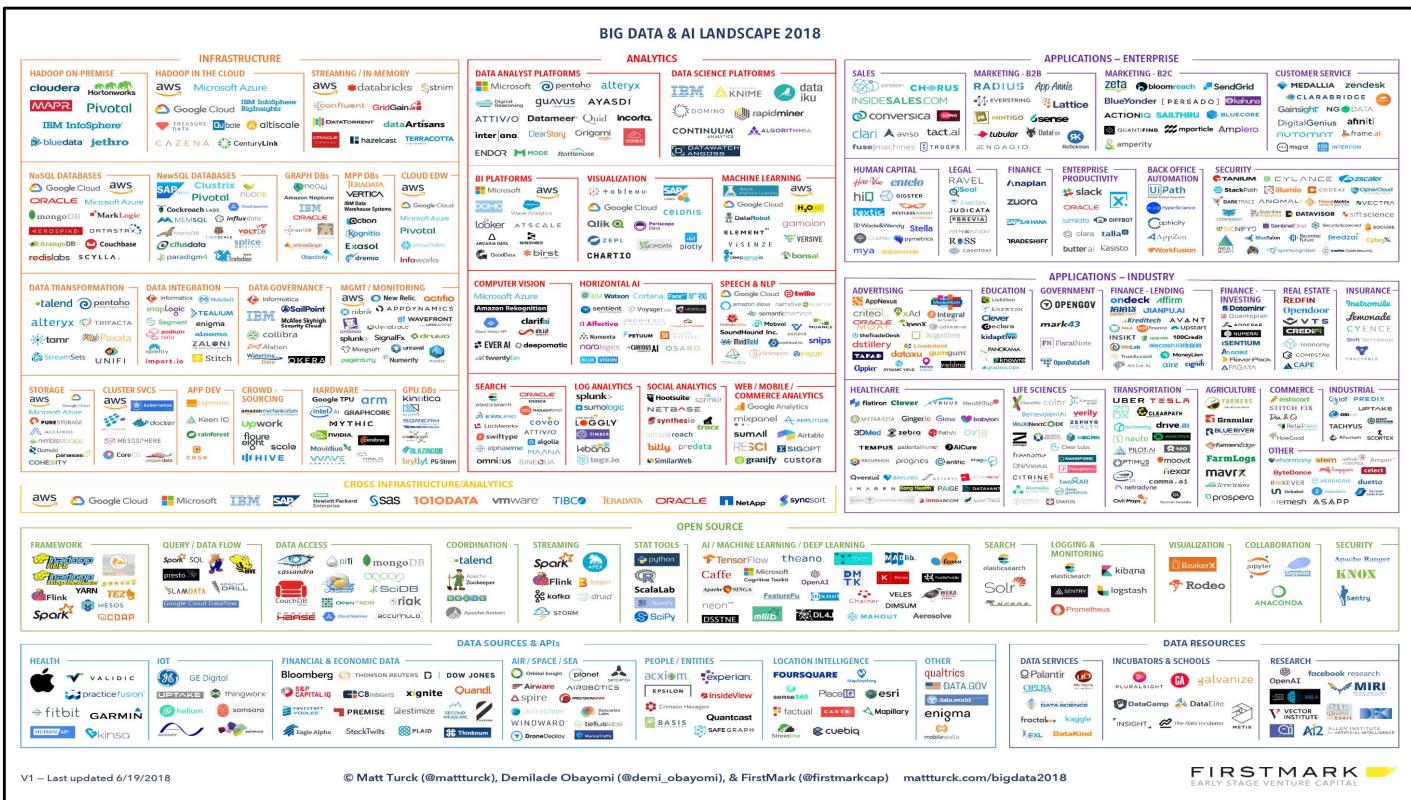
## **Big Data – Por que uma arquitetura ?**



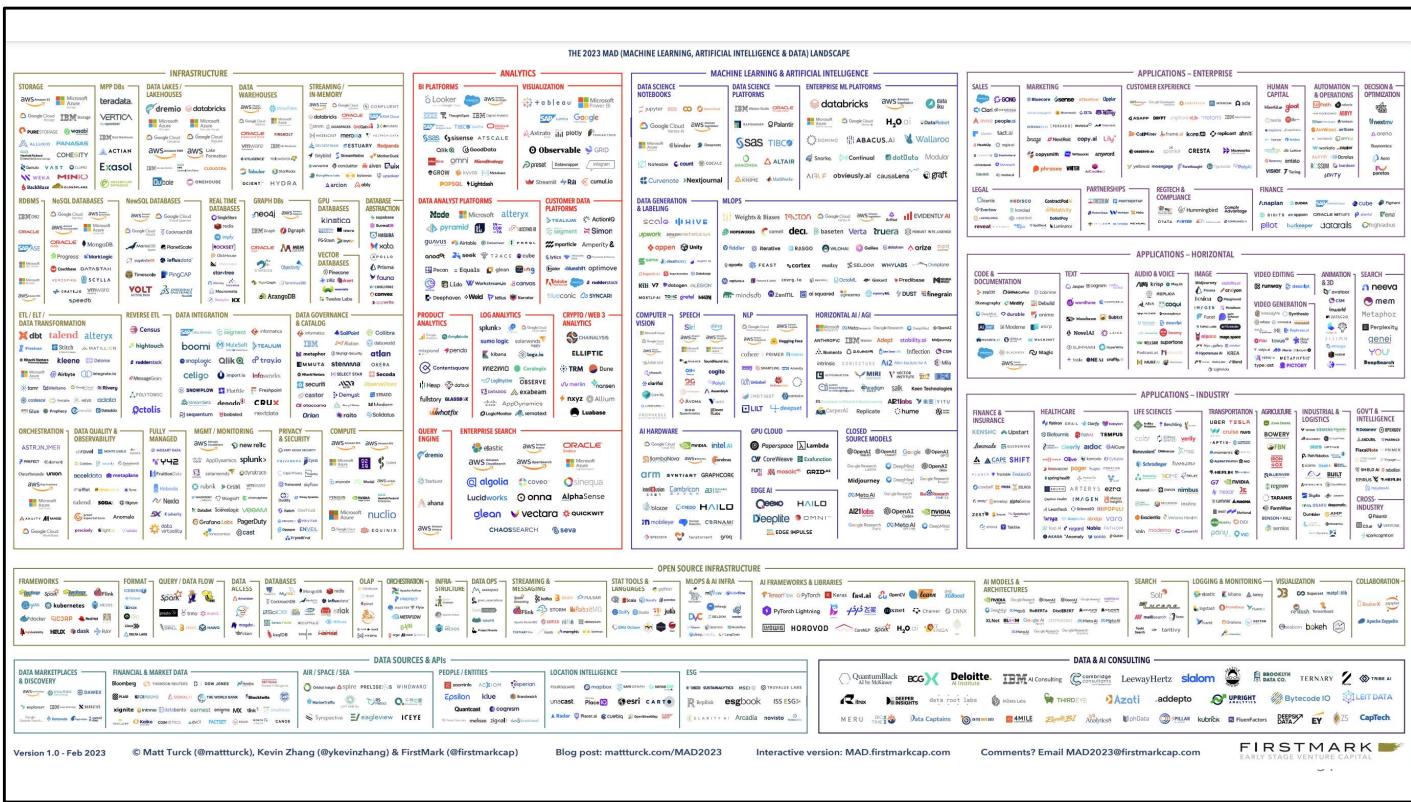
- Necessidade de inserir e pesquisar grandes volumes de dados com uma latência aceitável;
- O escalamento dos tradicionais bancos de dados e mesmo a inclusão de recursos de processamento assíncrono não respondem satisfatoriamente a problemas de corrupção de dados, tolerância a falhas e alta disponibilidade;
- E ...



Um importante aspecto que diferencia o fenômeno Big Data do fenômeno conhecido como Business Intelligence é a oferta de ferramentas Open Source, muitas delas desenvolvidas e disponibilizadas por grandes empresas geradoras de dados, tais como Facebook, Yahoo!, Twiter e LinkedIn, tornando a implementação de soluções e as iniciativas analíticas em Big Data acessíveis (FAN; BIFET, 2013).

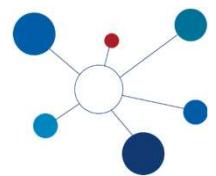


Fonte: <http://mattturck.com/>



## **Big Data – Principais arquiteturas**

- Kappa
- Lambda
- Liquid

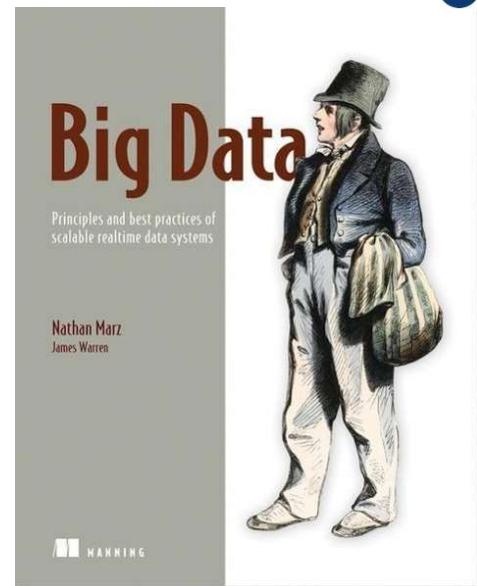


# ARQUITETURA LAMBDA

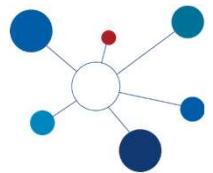


## Big Data – Arquitetura Lambda

- Marz e Warren (2013)
- Implementa funções arbitrárias em um conjunto arbitrário de dados
- Resposta em uma baixa latência pela combinação de diferentes ferramentas e técnicas
- Divide o sistema em três camadas dependentes e complementares



MARZ, N.; WARREN, J. Big Data: Principles and best practices of scalable realtime data systems. [S.I.]: Manning Publications Co., 2015.

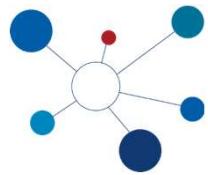


## Lambda – Premissa

Se um sistema de dados qualquer responde perguntas consultando os dados, então o sistema mais genérico pode responder questões consultando todos os dados.

A arquitetura Lambda é a proposta de Marz e Warren (2015) para a implementação de funções arbitrárias em um conjunto arbitrário de dados (pesquisa = função(todos os dados)) obtendo uma resposta em uma baixa latência pela combinação de diferentes ferramentas e técnicas. A arquitetura foi proposta em 2013 por Nathan Marz sendo o primeiro padrão arquitetônico para processamento em lote e em tempo real.

MARZ, N.; WARREN, J. *Big Data: Principles and best practices of scalable realtime data systems*. [S.I.]: Manning Publications Co., 2015.



## Lambda – Premissa

Se um sistema de dados qualquer responde perguntas consultando os dados, então o sistema mais genérico pode responder questões consultando todos os dados.

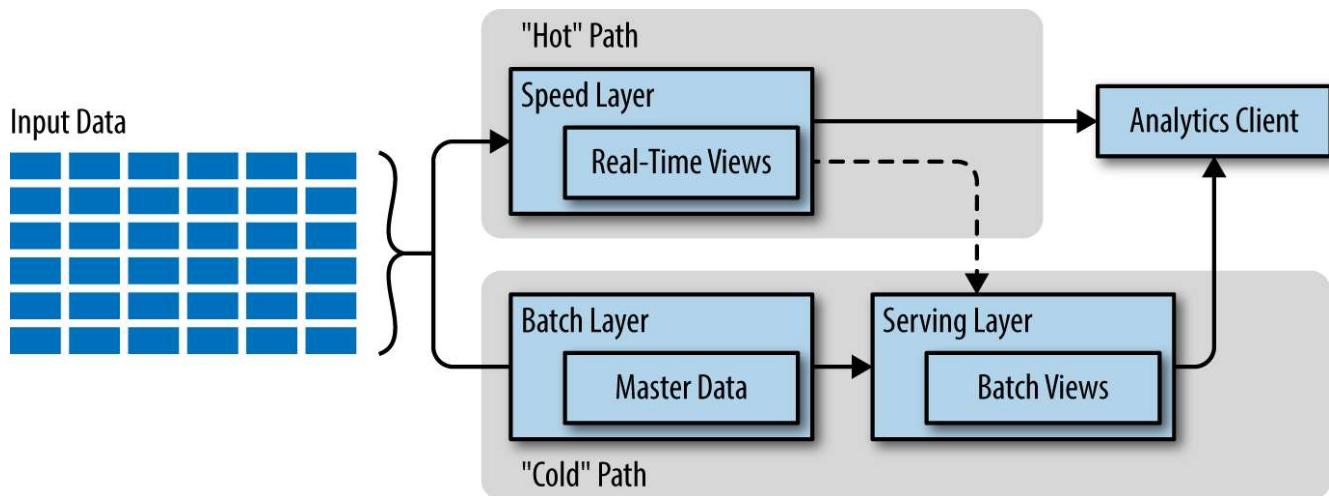
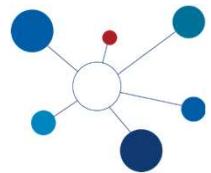
A definição genérica para um sistema de dados pode ser expressada como:

***consulta = função (todos os dados)***

O princípio de funcionamento da arquitetura Lambda é a divisão de um sistema Big Data em três camadas dependentes e complementares: camada batch, camada serving e camada speed. Cada camada atende a um conjunto de propriedades e é construída sobre as funcionalidades da camada mais próxima.

MARZ, N.; WARREN, J. *Big Data: Principles and best practices of scalable realtime data systems.* [S.I.]: Manning Publications Co., 2015.

# Lambda – Visão geral



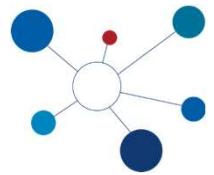
Na arquitetura Lambda há dois caminhos para o dado fluir pelo “tubo” de dados analítico:

- Um caminho quente (hot path) no qual dados sensíveis a latência fluem para serem rapidamente consumidos por clientes analíticos;
- Um caminho frio (cold path) pelo qual todos os dados fluem e são processados em lotes que podem tolerar grandes latências até que o resultado seja produzido.

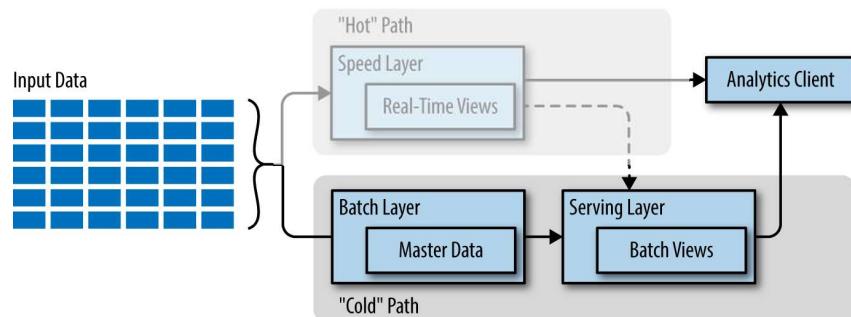
Em certas implementações a camada serving pode hospedar dados das visões real-time e das visões batch.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

## Lambda – Batch – Cold path



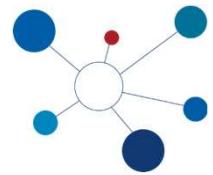
- Dados imutáveis (mudanças somente por novas entradas)
- Permite recomputação de qualquer ponto no tempo
- Resultados extremamente precisos
- Alta latência



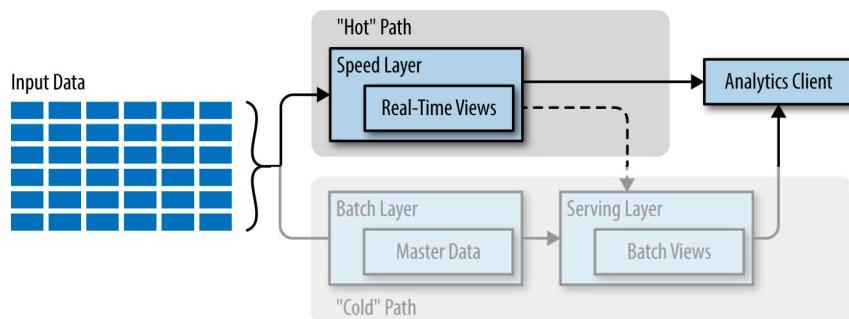
Quando o dado flui pelo caminho frio este dado é imutável. Quaisquer mudanças no valor dos dados são refletidas por novos dados, com um atributo temporal (timestamp), sendo armazenado no sistema junto com os valores antigos. Esta abordagem permite ao sistema recomputar o então atual valor para qualquer ponto no tempo dos dados coletados. Como o caminho frio pode tolerar uma grande latência até que os resultados sejam produzidos, a computação pode ser feita em grandes conjuntos de dados, e o tipo de cálculo pode ser demorado. O objetivo do caminho frio pode ser sumarizado como: tome o tempo que precisar, mas devolva um resultado extremamente acurado.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

## Lambda – Stream – Hot path



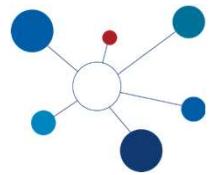
- Dados mutáveis (podem ser corrigidos na hora)
- Menos acurácia em troca de respostas rápidas



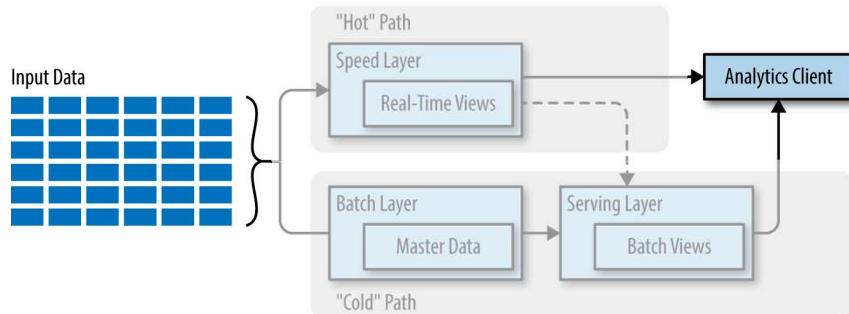
Quando o dado flui pelo caminho quente, estes dados são mutáveis e podem ser atualizados na hora. Neste caminho há uma restrição de latência para os dados, uma vez que os resultados são esperados em tempo quase real. O impacto desta restrição de latência é que os tipos de cálculos que podem ser executados são aqueles que podem ser executados rápido o suficiente. Isto pode implicar a troca de um algoritmo que ofereça acurácia perfeita por um que ofereça um resultado aproximado. O objetivo no caminho quente pode ser resumido assim: negociar um pouco de acurácia nos resultados para garantir que o resultado seja obtido rápido o suficiente.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

## Lambda – Cliente



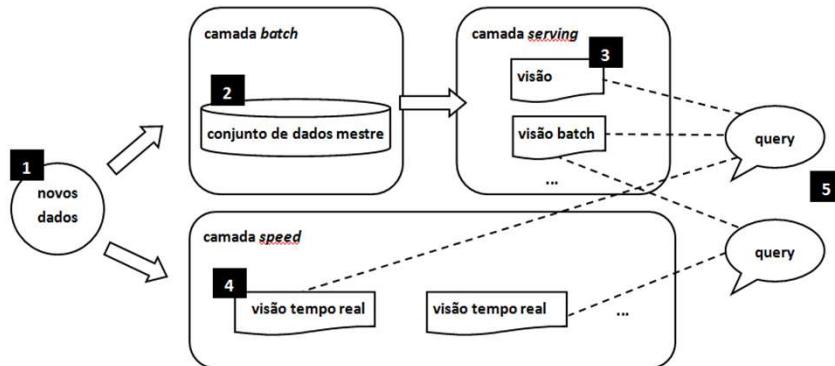
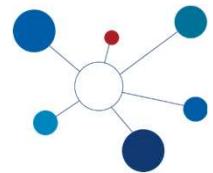
- Define o critério da obtenção do resultado
- Se tiver mais tempo, escolhe o resultado mais preciso
- Se não puder esperar, pesquisa por um resultado menos preciso e que represente um intervalo de tempo menor



Os caminhos frio e quente convergem finalmente para uma aplicação analítica cliente. O cliente precisa escolher o caminho a partir do qual ele obterá o resultado. Ele pode escolher obter dados menos precisos mas mais atualizados a partir do caminho quente ou ele pode usar um resultado mais antigo mas mais preciso a partir do caminho frio. Um importante componente da decisão relaciona-se à janela de tempo para a qual somente o caminho quente tem o resultado, uma vez que o caminho frio ainda não computou seu resultado. Olhando por este ponto de vista, o caminho quente tem resultados por somente uma pequena janela de tempo e seus resultados são finalmente atualizados pelos resultados mais precisos obtidos pelo caminho frio. Esta abordagem tem o efeito de minimizar o volume de dados com o qual o caminho quente precisa lidar.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

# Lambda - Resumo



$visão batch = função(todos os dados)$

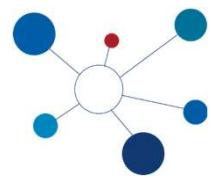
$visão tempo real = função(visão tempo real, novos dados)$

$query = função(visão batch,visão tempo real)$

HAUSENBLAS, M.; BIJNENS, N. What is the Lambda Architecture? <http://lambda-architecture.net>

- (1) todos os dados que entram no sistema vão para as camadas *batch* e *speed* para serem processados;
- (2) a camada *batch* é responsável por gerenciar o conjunto de dados mestre (conjunto de dados imutáveis e crescentes) e por pré-processar as visões *batch*;
- (3) a camada *serving* indexa o conteúdo das views *batch* para ser acessado pelas *queries*;
- (4) a camada *speed* lida com dados recentes e complementa o resultado da camada *batch*;
- (5) as consultas obtêm dados das visões *batch* e das visões de tempo real

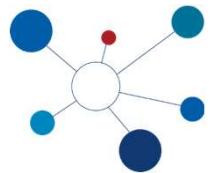
## Lambda – Críticas à arquitetura



### Acurácia do processamento *stream (hot path)*

O processamento em tempo real não necessariamente será aproximado, menos poderoso e mais sujeito a perdas. Os *frameworks* de processamento em tempo real são menos maduros, mas oferecem a mesma garantia semântica que os *frameworks* de processamento em lote.

KREPS, J. *Questioning the Lambda Architecture: The Lambda Architecture has its merits, but alternatives are worth exploring.* 2014. Disponível em: <<https://www.oreilly.com/ideas/questioning-the-lambda-architecture>>. Acesso em: 11 Junho 2016.

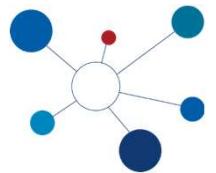


## Lambda – Críticas à arquitetura

### Complexidade de manutenção de código das camadas *batch e speed*

A lógica da transformação tem que ser implementada duas vezes, uma para cada camada. A separação do processamento em duas camadas implica o uso de diferentes *frameworks* distribuídos, tais como Hadoop e Storm, que possuem suas especificidades de codificação. Recomenda-se a adoção de um ou de outro processamento: em lote caso não haja sensibilidade à latência, ou em tempo real, caso a baixa latência seja um requisito imprescindível.

KREPS, J. *Questioning the Lambda Architecture: The Lambda Architecture has its merits, but alternatives are worth exploring.* 2014. Disponível em: <<https://www.oreilly.com/ideas/questioning-the-lambda-architecture>>. Acesso em: 11 Junho 2016.



## Lambda – Críticas à arquitetura

### Integração de resultados

Como o resultado final é uma composição de resultados de visões *batch* e *speed*, a sincronização entre estas duas visões torna-se um fator chave para que se evitem resultados redundantes ou ausentes. O armazenamento dos resultados das camadas *batch* e *speed* em repositórios diferentes deixa o sistema em um estado conhecido como **persistência poliglota**, exigindo a agregação dos resultados das views para atendimento das consultas.

Na arquitetura Lambda, a camada *batch* é responsável por processar todo o conjunto de dados para gerar as visões *batch* enquanto que à camada *speed* cabe a responsabilidade de processar os novos dados que chegam ao sistema gerando a partir destes as visões *speed*. As pesquisas são feitas numa composição das visões *batch* e *speed*. Tão logo o dado é processado na camada *batch* o resultado é armazenado nas visões *batch* e retirado das visões *speed*.

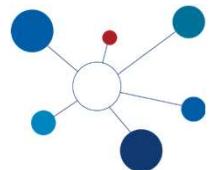
Como solução para os problemas de sincronização e agregação, Vanhove propõe que os dados que chegam à camada *speed* sejam identificados com uma etiqueta (tag), chamada de  $T_n$ . O processamento desta camada gera como resultado (visão *speed*) $T_n$ . Quando o processamento da camada *batch* é concluído, os dados resultantes são movidos para a visão *batch* e o sistema muda para uma nova etiqueta,  $T_{n+1}$ , para todos os novos dados. Os dados de (visão *speed*) $T_{n-1}$  são então apagados.

VANHOVE, T. et al. Managing the synchronization in the lambda architecture for optimized big data analysis. *IEICE Transactions on Communications*, The Institute of Electronics, Information and Communication Engineers, v. 99, n. 2, p. 297–306, 2016.

# ARQUITETURA KAPPA



# Big Data – Arquitetura Kappa



- Jay Kreps (2013)
- Todo o processamento é responsabilidade de uma camada paralelizada de processamento em **fluxo**
- Baseada em **logs**

**LinkedIn** Engineering

## The Log: What every software engineer should know about real-time data's unifying abstraction

Jay Kreps December 16, 2013

I joined LinkedIn about six years ago at a particularly interesting time. We were just beginning to run up against the limits of our monolithic, centralized database and needed to start the transition to a portfolio of specialized distributed systems. This has been an interesting experience: we built, deployed, and run to this day a distributed graph database, a distributed search backend, a Hadoop installation, and a first and second generation key-value store.

One of the most useful things I learned in all this was that many of the things we were building had a very simple concept at their heart: the log. Sometimes called write-ahead logs or commit logs or transaction logs, logs have been around almost as long as computers and are at the heart of many distributed data systems and real-time application architectures.

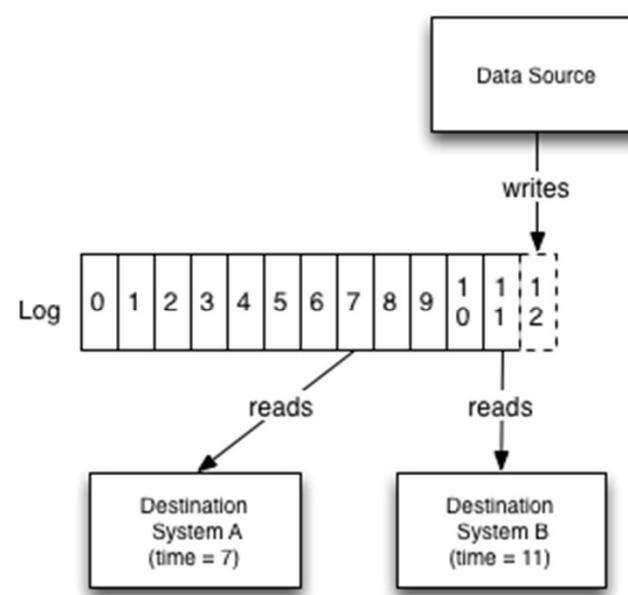
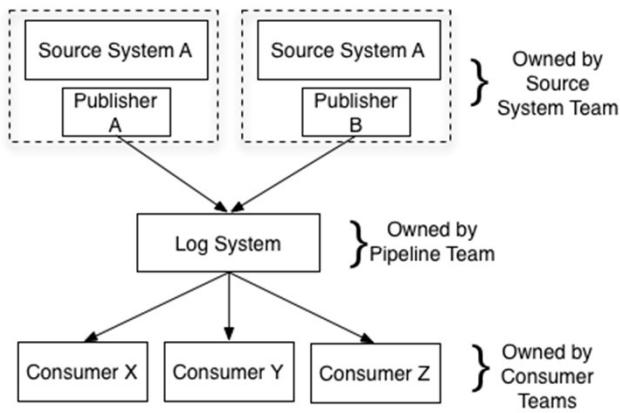
You can't fully understand databases, NoSQL stores, key value stores, replication, paxos, hadoop, version control, or almost any software system without understanding logs; and yet, most software engineers are not familiar with them. I'd like to change that. In this post, I'll walk you through everything you need to know about logs, including what is log and how to use logs for data

- Baseada em uma fila central escalável;
- Todos os dados (eventos) são armazenados nesta fila pelos nós produtores (aplicações);
- Cada evento é imutável e armazenado na sequência em que é gerado;
- O evento só pode ser alterado por um novo evento sendo adicionado à fila;
- Os nós consumidores consomem os eventos assincronamente e podem reprocessar;
- Os eventos são retidos por um período de tempo determinado

KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction.

<https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying>

# Kappa – Tudo é log



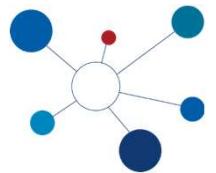


# Kappa – O que é log ?

```
jkreps-mn:~ jkreps$ tail -f -n 20 /var/log/apache2/access_log
::1 -- [23/Mar/2014:15:07:00 -0700] "GET /images/apache_feather.gif HTTP/1.1" 200 4128
::1 -- [23/Mar/2014:15:07:04 -0700] "GET /images/producer_consumer.png HTTP/1.1" 200 86
::1 -- [23/Mar/2014:15:07:04 -0700] "GET /images/log_anatomy.png HTTP/1.1" 200 19579
::1 -- [23/Mar/2014:15:07:04 -0700] "GET /images/consumer-groups.png HTTP/1.1" 200 268
::1 -- [23/Mar/2014:15:07:04 -0700] "GET /images/log_compaction.png HTTP/1.1" 200 41414
::1 -- [23/Mar/2014:15:07:04 -0700] "GET /documentation.html HTTP/1.1" 200 189893
::1 -- [23/Mar/2014:15:07:04 -0700] "GET /images/log_cleaner_anatomy.png HTTP/1.1" 200
::1 -- [23/Mar/2014:15:07:04 -0700] "GET /images/kafka_log.png HTTP/1.1" 200 134321
::1 -- [23/Mar/2014:15:07:04 -0700] "GET /images/mirror-maker.png HTTP/1.1" 200 17054
::1 -- [23/Mar/2014:15:08:07 -0700] "GET /documentation.html HTTP/1.1" 200 189937
::1 -- [23/Mar/2014:15:08:07 -0700] "GET /styles.css HTTP/1.1" 304 -
::1 -- [23/Mar/2014:15:08:07 -0700] "GET /images/kafka_logo.png HTTP/1.1" 304 -
::1 -- [23/Mar/2014:15:08:07 -0700] "GET /images/producer_consumer.png HTTP/1.1" 304 -
::1 -- [23/Mar/2014:15:08:07 -0700] "GET /images/log_anatomy.png HTTP/1.1" 304 -
::1 -- [23/Mar/2014:15:08:07 -0700] "GET /images/consumer-groups.png HTTP/1.1" 304 -
::1 -- [23/Mar/2014:15:08:07 -0700] "GET /images/log_cleaner_anatomy.png HTTP/1.1" 304
::1 -- [23/Mar/2014:15:08:07 -0700] "GET /images/log_compaction.png HTTP/1.1" 304 -
::1 -- [23/Mar/2014:15:08:07 -0700] "GET /images/kafka_log.png HTTP/1.1" 304 -
::1 -- [23/Mar/2014:15:08:07 -0700] "GET /images/mirror-maker.png HTTP/1.1" 304 -
::1 -- [23/Mar/2014:15:09:55 -0700] "GET /documentation.html HTTP/1.1" 200 195264
```

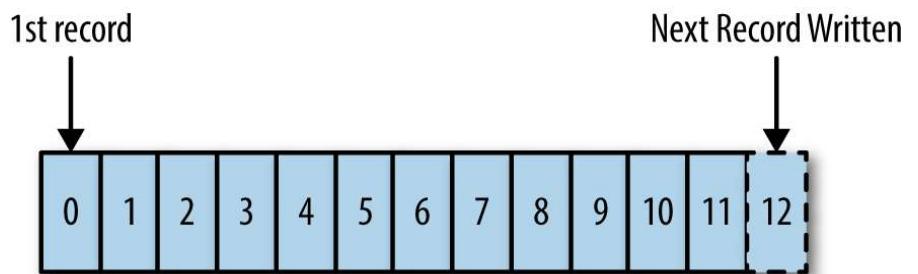
Nossa tendência é a associação imediata com logs como os do Apache.

**KREPS, Jay. I Heart Logs: Event Data, Stream Processing, and Data Integration.** "O'Reilly Media, Inc.", 2015.



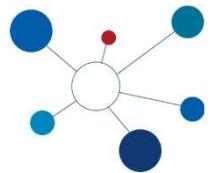
## Kappa – O que é log ?

- Uma sequência de registros ordenados por tempo;
- *Append-only*;
- Semelhante a logs de bancos de dados (*commit logs ou journals*);
- JSON ou qualquer conteúdo.



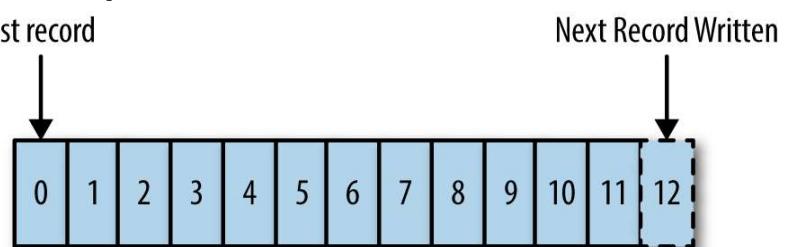
Cada retângulo representa um registro que foi anexado ao log. Os registros são armazenados na ordem em que foram anexados. As leituras procedem da esquerda para a direita. Cada entrada anexada ao log recebe um número de entrada de log sequencial exclusivo que atua como sua chave exclusiva. O conteúdo e o formato dos registros não são importantes para os propósitos desta discussão. Para ser concreto, podemos apenas imaginar cada registro como um blob JSON, mas é claro que qualquer formato de dados serve.

KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration.**" O'Reilly Media, Inc.", 2015.



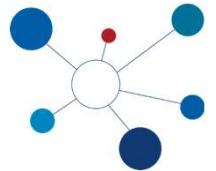
## Kappa – O que é log ?

- A posição do log confere a noção de tempo;
- É um tipo de tabela ou de arquivo onde os registros são ordenados por tempo;
- Uma estrutura de dados abstrata, não um arquivo de texto;
- Registram **o que** acontece e **quando** acontece;



A ordenação dos registros define uma noção de “tempo”, já que as entradas à esquerda são definidas como mais antigas que as entradas à direita. O número da entrada de log pode ser considerado como o “carimbo de data/hora” da entrada. Descrever essa ordenação como uma noção de tempo parece um pouco estranho a princípio, mas tem a propriedade conveniente de ser desacoplado de qualquer relógio físico específico. Essa propriedade se tornará essencial quando chegarmos aos sistemas distribuídos.

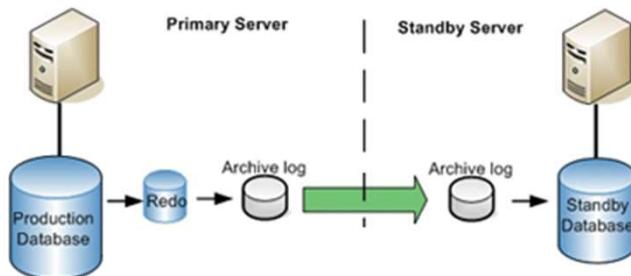
KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration.** "O'Reilly Media, Inc.", 2015.



## Log – Onde é usado

Em bancos de dados:

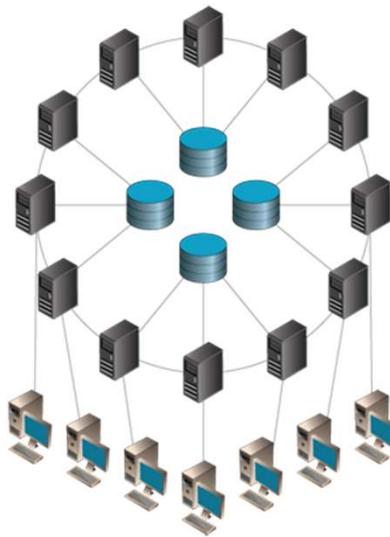
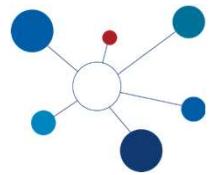
- O log é usado como um mecanismo de publicação/assinatura para transmitir dados para outras réplicas;
- O log é usado como um mecanismo de consistência para executar as atualizações que são aplicadas em múltiplas réplicas.



Exemplo: replicação entre um banco Oracle primário e seu banco *standby*. Note que os dados trafegam por meio de logs (archived redo logs).

KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration.** "O'Reilly Media, Inc.", 2015.

## Log – Onde é usado



Em sistemas distribuídos:

- A natureza temporal de uma sequência de logs é a garantia de que os resultados serão obtidos em ordem em um sistema distribuído.
- Princípio de replicação da máquina de estados:  
***"Se dois processos idênticos e determinísticos começarem no mesmo estado e receberem as mesmas entradas na mesma ordem, eles produzirão a mesma saída e terminarão no mesmo estado."***

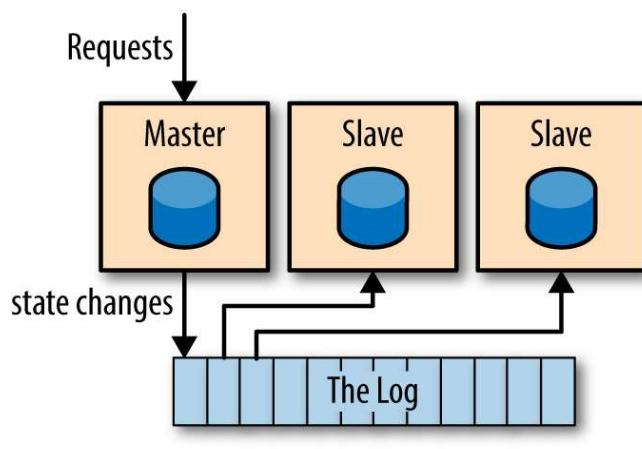
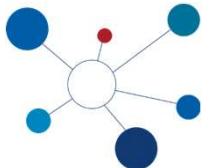
Os mesmos problemas que os bancos de dados resolvem com logs (como distribuir dados para réplicas e concordar com a ordem de atualização) estão entre os problemas mais fundamentais para todos os sistemas distribuídos.

...

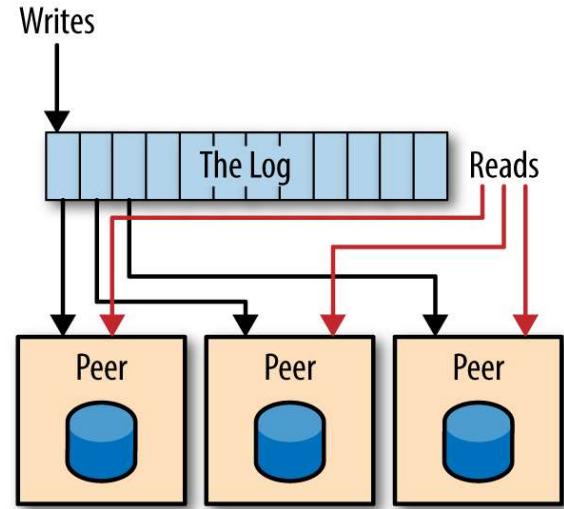
Uma das coisas bonitas sobre isso é que os números de entrada de log discretos agora agem como um relógio para o estado das réplicas - você pode descrever o estado de cada réplica por um único número: o registro de data e hora para a entrada de log máxima que ela processou . Duas réplicas ao mesmo tempo estarão no mesmo estado. Assim, esse carimbo de data/hora combinado com o log captura exclusivamente todo o estado da réplica. Isso fornece uma noção de tempo discreta e orientada a eventos que, ao contrário dos relógios locais da máquina, é facilmente comparável entre diferentes máquinas.

KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration.** "O'Reilly Media, Inc.", 2015.

## Log – Onde é usado



**Primary Backup**

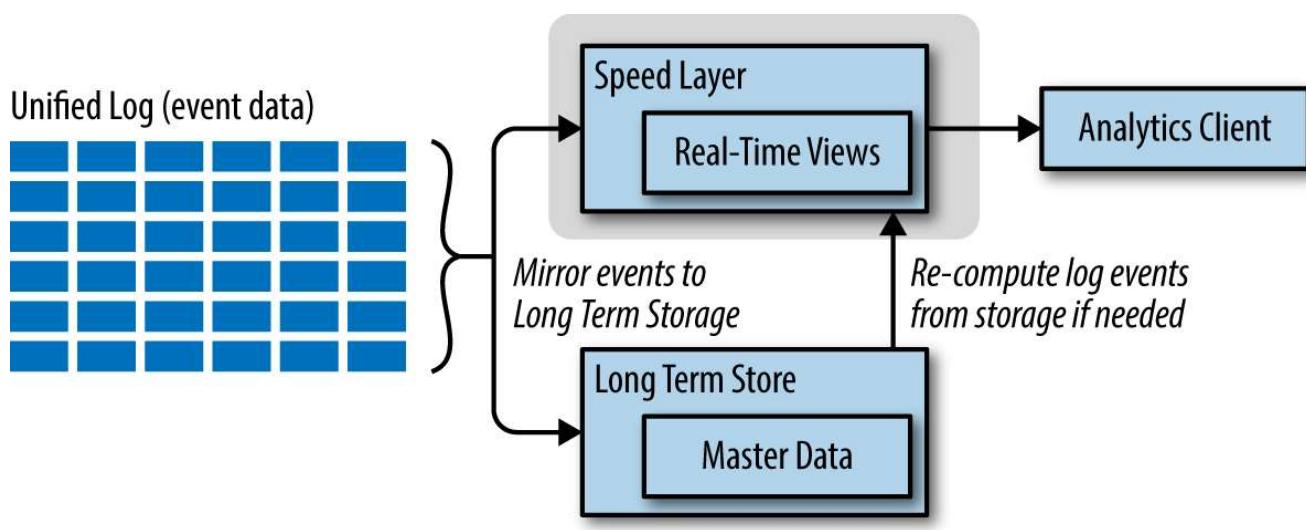
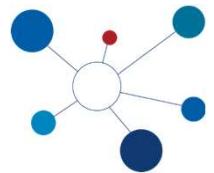


**State Machine Replication**

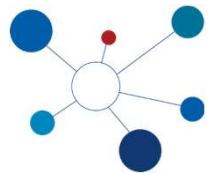
No modelo de backup primário, um nó mestre é escolhido para lidar com todas as leituras e gravações. Cada gravação é postada no log. Os escravos se inscrevem no log e aplicam as alterações que o mestre executou em seu estado local. Se o mestre falhar, um novo mestre é escolhido entre os escravos. No modelo de replicação da máquina de estado, todos os nós são pares. As gravações vão primeiro para o log e todos os nós aplicam a gravação na ordem determinada pelo log.

KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration.** "O'Reilly Media, Inc.", 2015.

# Kappa – Descrição



TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

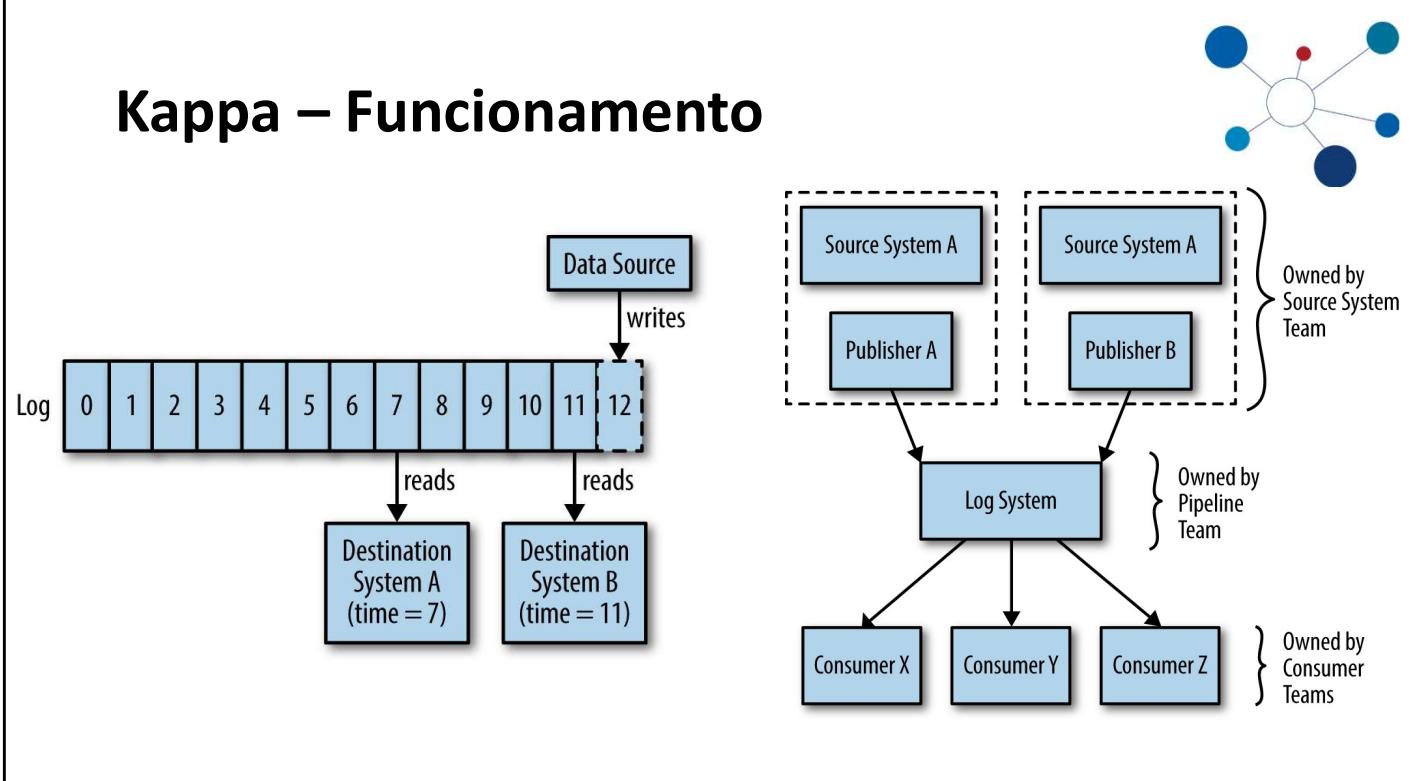


## Kappa – Descrição

- Baseada em uma fila central escalável;
- Todos os dados (eventos) são armazenados nesta fila pelos nós produtores (aplicações);
- Cada evento é imutável e armazenado na sequência em que é gerado;
- O evento só pode ser alterado por um novo evento sendo adicionado à fila;
- Os nós consumidores consomem os eventos assincronamente e podem reprocessar;
- Os eventos são retidos por um período de tempo determinado.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark.**" O'Reilly Media, Inc., 2017.

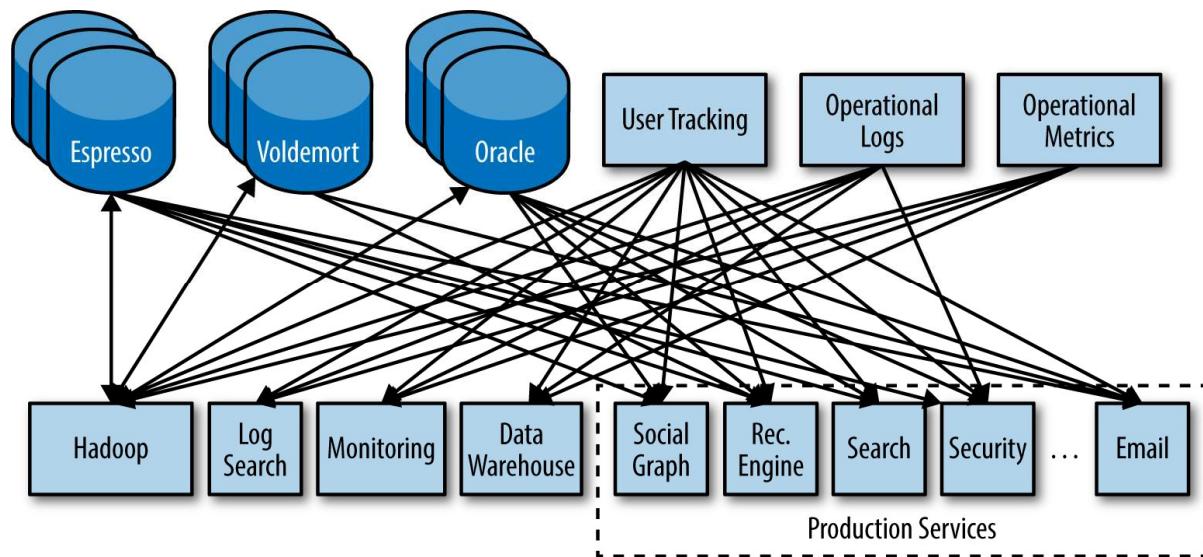
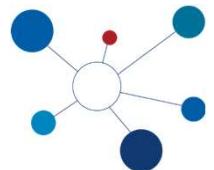
# Kappa – Funcionamento



KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction.

<https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying>

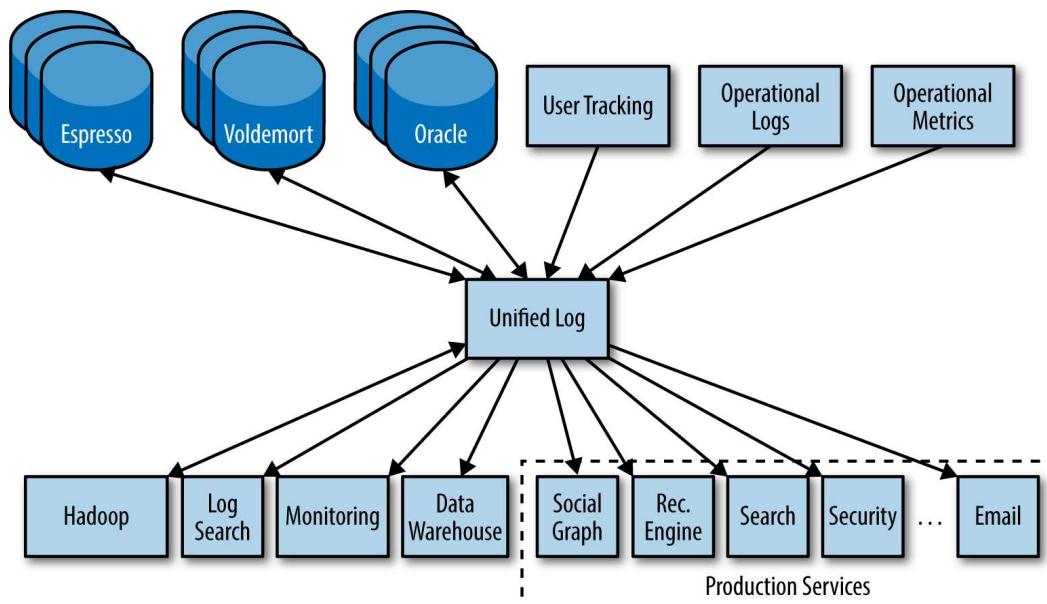
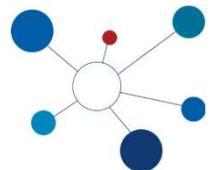
## Kappa – Aplicações



KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction.

<https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying>

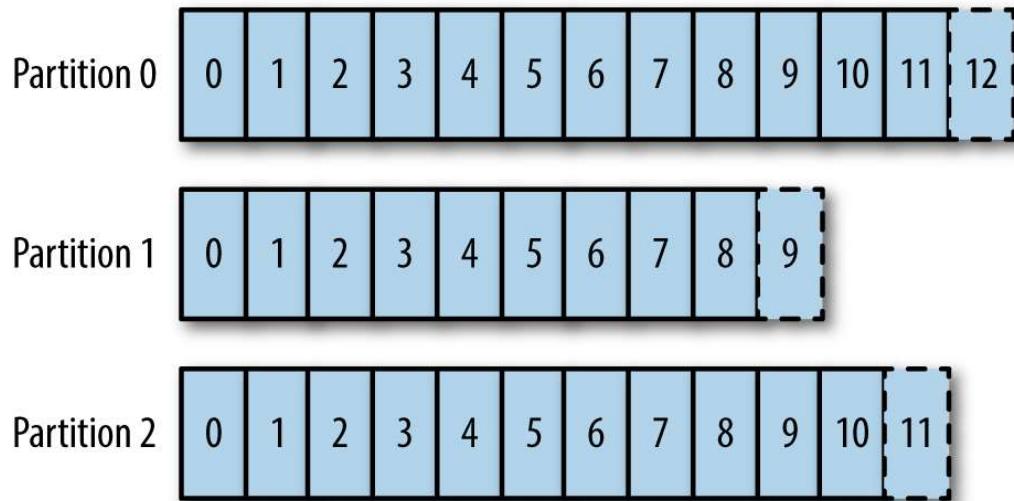
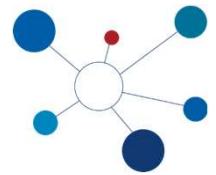
## Kappa – Aplicações



KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction.

<https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying>

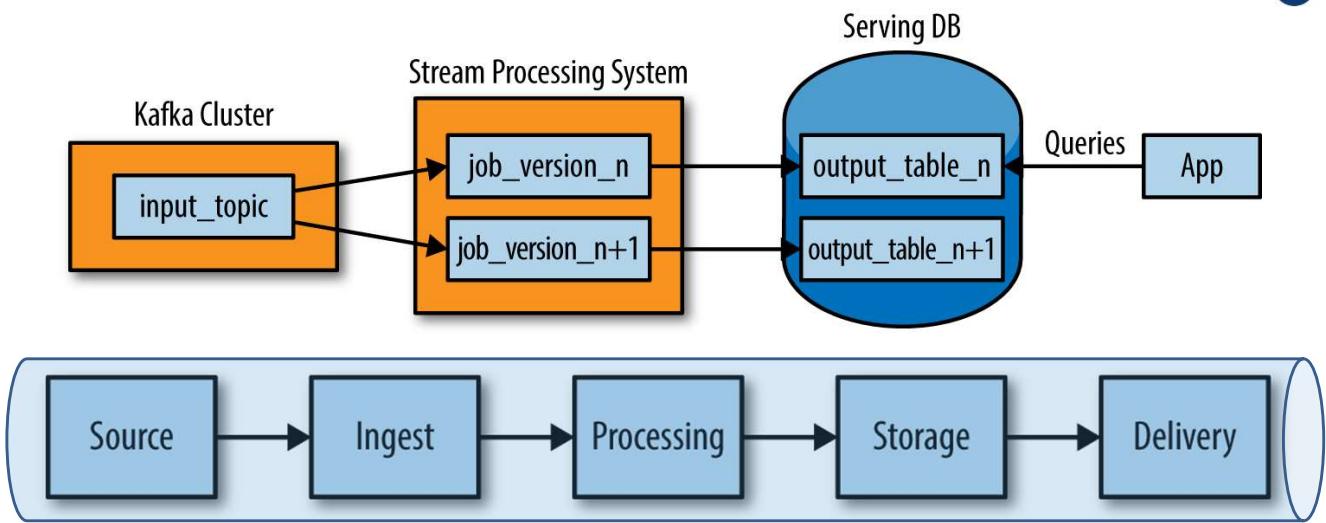
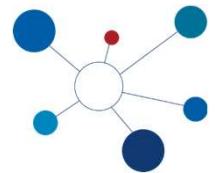
## Kappa – Escalamento do log



KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction.

<https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying>

# Kappa – Implementação



KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction.

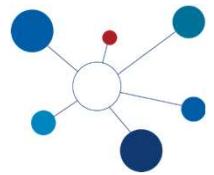
<https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying>

# ARQUITETURA LIQUID



# Big Data – Arquitetura Liquid

- Fernandez e outros (2015)
- *Stack* de integração de dados
- Oferece acesso de baixa latência
- Suporta processamentos em lote e em tempo quase real
- Suporta processamento incremental, eficiência de custo e alta disponibilidade.



## Liquid: Unifying Nearline and Offline Big Data Integration

Raul Castro Fernandez,<sup>1</sup> Peter Pietzuch  
Imperial College London  
{rc3011, prp}@doc.ic.ac.uk

Jay Kreps, Neha Narkhede, Jun Rao  
Confluent Inc.  
{jay, neha, jun}@confluent.io

Joel Koshy, Dong Lin, Chris Riccomini, Guozhang Wang  
LinkedIn Inc.  
{jkoshy, dolin, criccomini, gwang}@linkedin.com

### ABSTRACT

With more sophisticated data-parallel processing systems, the new bottleneck in data-intensive companies shifts from the back-end data systems to the front-end data integration stacks required for the pre-processing of data for back-end applications. The use of back-end data systems with different access latencies and data integration requirements poses new challenges that current data integration stacks based on distributed file systems—proposed a decade ago for batch-oriented processing—cannot address.

In this paper, we describe *Liquid*, a data integration stack that provides low latency data access to support near real-time in addition to batch applications. It supports incremental processing,

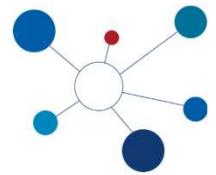
on commodity shared-nothing clusters with scalable distributed file systems (DFS) such as GFS [14] or HDFS [32] in order to produce data for back-end systems [22].

While in the past processing performance was limited, a new breed of data-parallel systems such as Spark [42] and Storm [36] has helped mitigate processing bottlenecks in back-end systems. As a result, the pendulum has swung back, making the data integration stack perhaps critical. Perhaps, for *nearline* data processing systems, i.e., back-end systems that are stream-oriented and therefore require low-latency, high-throughput data access, the use of a DFS as the storage layer increases data access latency, thus impacting the performance of applications.

Many organizations today use a **MAPREDUCE** stack for data intensive processing.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).

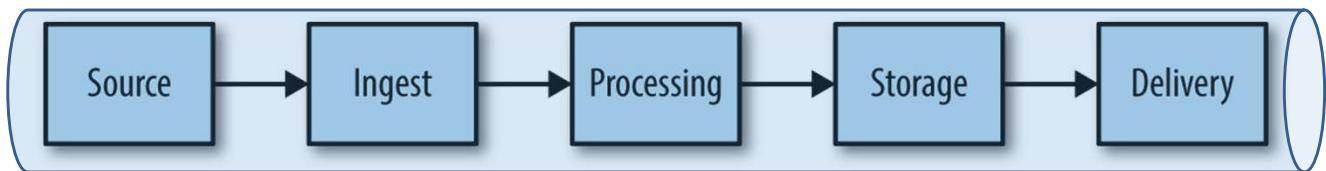
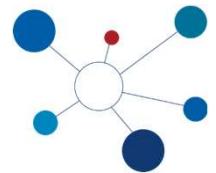
## Tipos de processamento



**Lote (batch)**

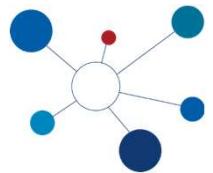
**Fluxo (stream)**

## Tipos de processamento



**Lote (batch)**

**Fluxo (stream)**  
**online      nearline**



# Tipos de processamento

## Online

Executado por aplicações que oferecem sua resposta com uma latência de milissegundos.

## Nearline (near online)

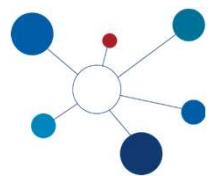
Executado por aplicações que oferecem sua resposta com uma latência de segundos.

Quanto antes oferecerem o resultado maior é o valor para os usuários.

**Near online** - Aplicativos que consultam um gráfico social, pesquisam dados, normalizam dados ou monitoram alterações são classificados como nearline: quanto mais cedo eles fornecerem resultados, maior será o valor para os usuários finais. Como os pipelines de processamento têm mais estágios, a latência de ponta a ponta também aumenta, o que torna o processamento nearline mais desafiador. Fundamentalmente, as pilhas baseadas em DFS não suportam processamento de baixa latência porque têm uma alta sobrecarga por estágio: elas são projetadas para leituras e gravações de dados granulares.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).

# Tipos de processamento

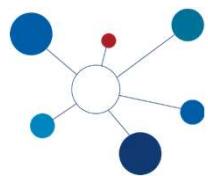


## Offline

Processamento onde a latência aceitável varia entre minutos a horas.

Neste tipo de processamento o armazenamento em DFS é aceitável e recomendado porque sua latência não compromete o processamento.

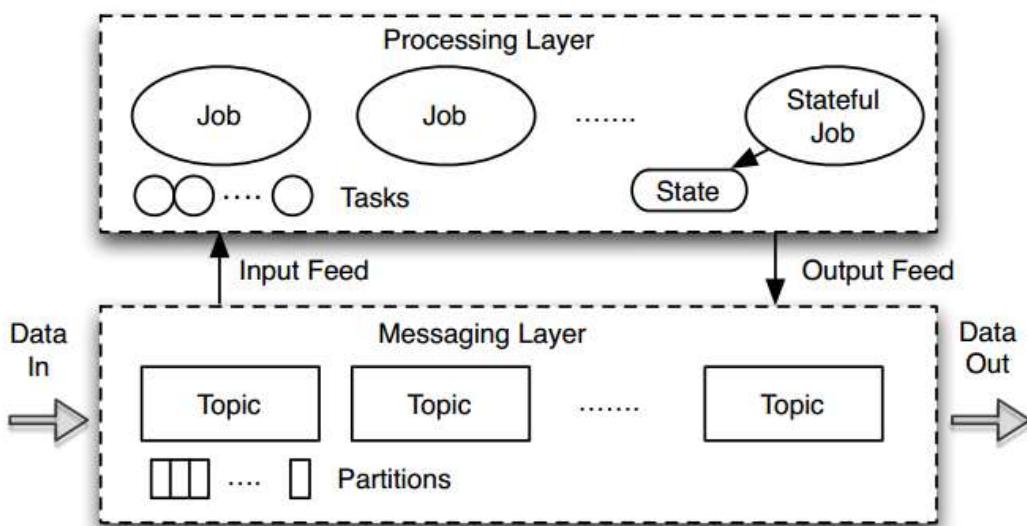
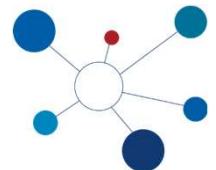
FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).



## Stateful e Stateless

- Adjetivos que descrevem se um programa, processo ou computador é capaz de anotar e lembrar um ou mais eventos precedentes em uma determinada sequência de interações.
- **Stateful** – o processo mantém a trilha histórica do estado das interações, normalmente atribuindo valores em um campo de armazenamento destinado a este fim;
- **Stateless** – o processo não mantém registro de interações anteriores e cada nova interação é processada unicamente com base nos dados fornecidos ao processo.

# Liquid – Descrição da arquitetura

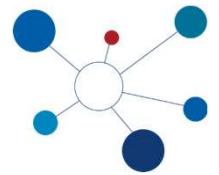


**Camada de mensagem** – provê acesso aos dados baseado em metadados, os quais permitem às aplicações consumidoras lerem dados a partir de um ponto específico no tempo. Esta camada é distribuída para escalar para grandes volumes de dados mantendo alta disponibilidade;

**Camada de processamento** – executa tarefas do tipo ETL para sistemas consumidores, garantindo um acesso aos dados com baixa latência. Esta camada pode executar processamentos arbitrários antes de entregar os dados aos sistemas consumidores finais, indo de limpeza dos dados e normalização para computação de estatísticas agregadas ou detecção de anomalias nos dados.

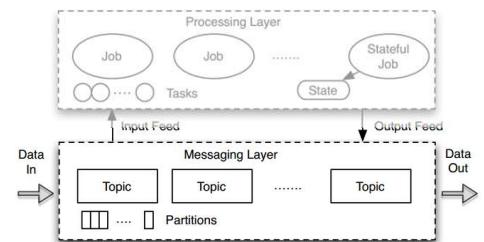
FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).

# Liquid – Descrição da arquitetura



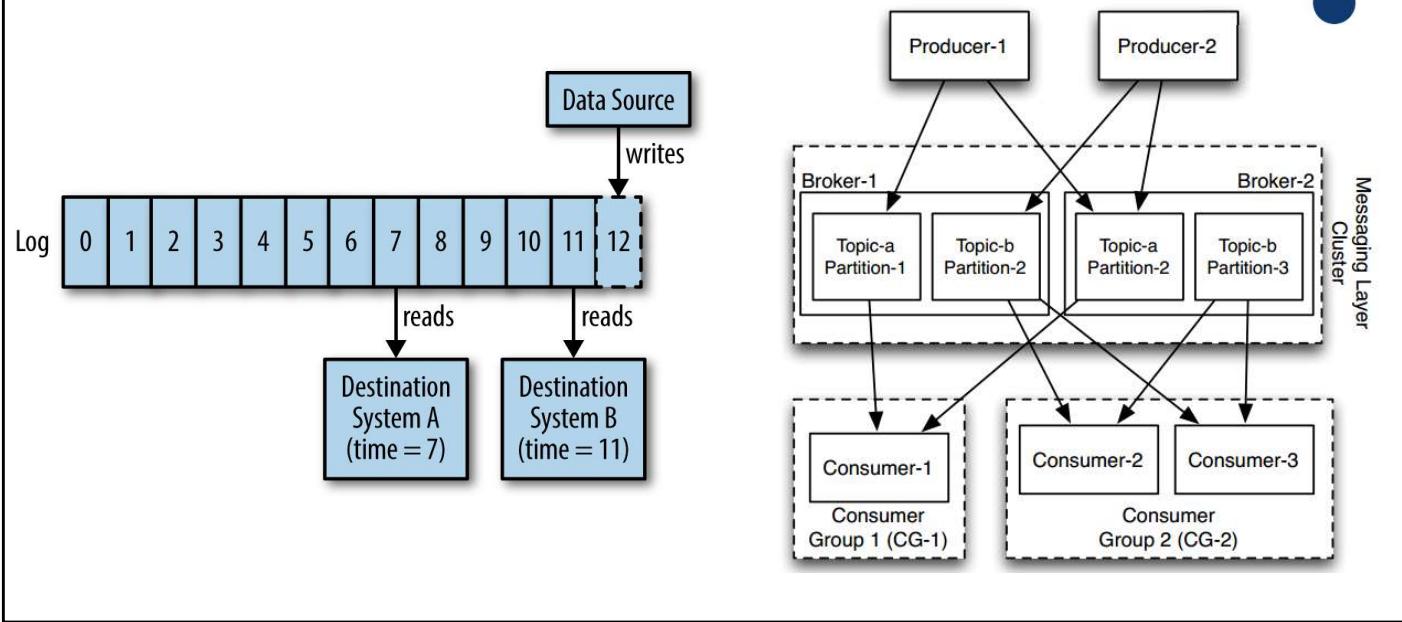
## Camada de mensagem

- Suporta a camada de processamento;
- Armazena um grande volume de dados com alta disponibilidade;
- Oferece a possibilidade de acessar os dados com base em anotações por metadados (*rewindability*).



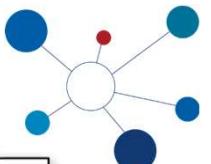
FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).

# Liquid – Camada de mensagem



Baseia-se no modelo publicador/assinante, armazenando os dados em tópicos. Pode utilizar o Apache Kafka.

# Liquid – Camada de mensagem



## Tópicos

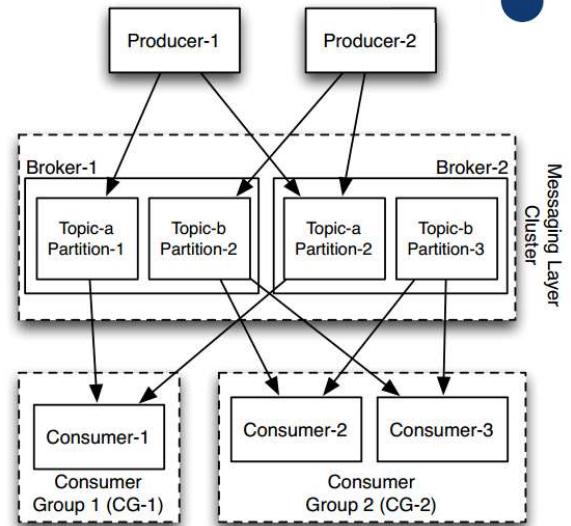
Separam os dados em categorias que façam sentido para as aplicações que produzem os dados.

## Produtores

Aplicações que produzem os eventos, os quais são armazenados em diferentes tópicos.

## Consumidores

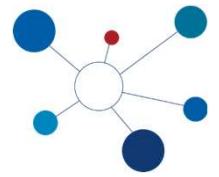
São aplicações assinantes dos tópicos, lendo dados da camada de mensagem.



**Metadata-based access.** A camada de mensagens usa um gerenciador de deslocamento altamente disponível e logicamente centralizado para manter anotações nos dados, que podem ser consultados pelos clientes. Por exemplo, os consumidores podem verificar suas últimas compensações consumidas para salvar seu progresso; após a falha, eles podem solicitar os últimos dados que processaram. Para reprocessar os dados, os clientes podem incluir metadados, como timestamps, com os deslocamentos e recuperar dados de acordo com esses timestamps armazenados anteriormente.

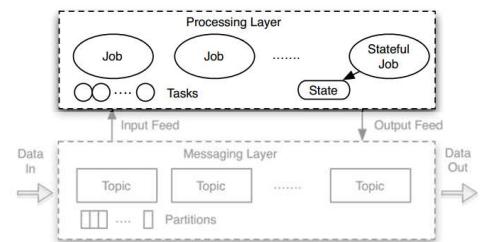
FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).

# Liquid – Camada de processamento



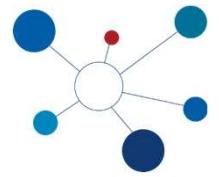
## Camada de processamento

- Executa *jobs* do tipo ETL para diferentes sistemas consumidores de acordo com modelo de processamento em fluxo que mantém o estado (*stateful*);
- Garante os níveis de serviço através do isolamento de recursos;
- Oferece resultados com baixa latência;
- Permite o processamento incremental.



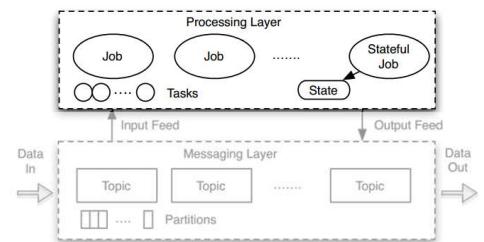
FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).

# Liquid – Camada de processamento



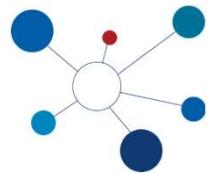
## Jobs

- São divididos em tarefas para permitir o processamento paralelo, processando diferentes partições dos tópicos;
- Podem se comunicar com outros *jobs*, formando um grafo de processamento de dados em fluxo.



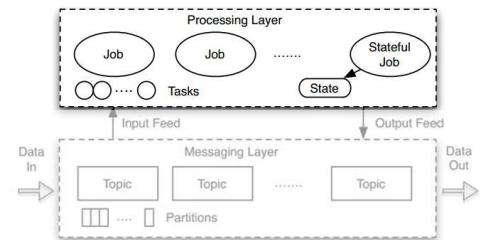
FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).

# Liquid – Camada de processamento



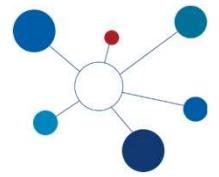
## Processamento *stateful*

- Os estados são carregados localmente por cada *job*;
- Estes estados podem ser representados de formas diversas, tais como uma janela dos dados em fluxo mais recentes, um dicionário de estatísticas ou mesmo um índice invertido utilizado em *queries*;
- Para fins de recuperação de um *job*, a camada de processamento salva o estado atual do *job* como dados derivados, armazenados na cada de mensagem.



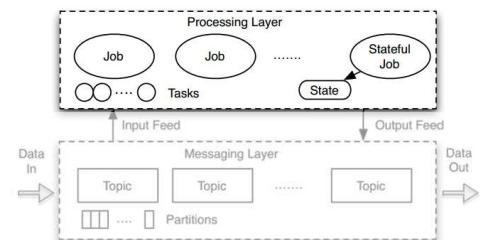
FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).

# Liquid – Camada de processamento



## Processamento incremental

- Ocorre pela combinação do estado de um *job* e dos metadados;
- O *job* pode periodicamente verificar os *offsets* que já consumiu e manter um sumário dos dados de entrada como sendo seu estado atual;
- Quando novos dados são disponibilizados, o *job* pode então ignorar os dados já processados.



FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).