Um Guia Para a Modelagem e Desenvolvimento de Aplicações Big Data Sobre o Modelo de Arquitetura Zeta

Kassiano Matteussi, Paulo Souza, Julio Anjos, Claudio Geyer
kjmatteussi, prrsjunior, jcsanjos, geyer@inf.ufrgs.br

INSTITUTO DE INFORMÁTICA
UFRGS

ERAD/RS 2017
XVII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul

Sobre

Autores

M.Sc. Kassiano Matteussi

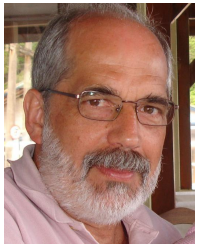
Doutorando

B.Sc. Paulo Souza

Mestrando

Ph.D. Julio Anjos

Pesquisador Associado



Ph.D. Claudio Geyer

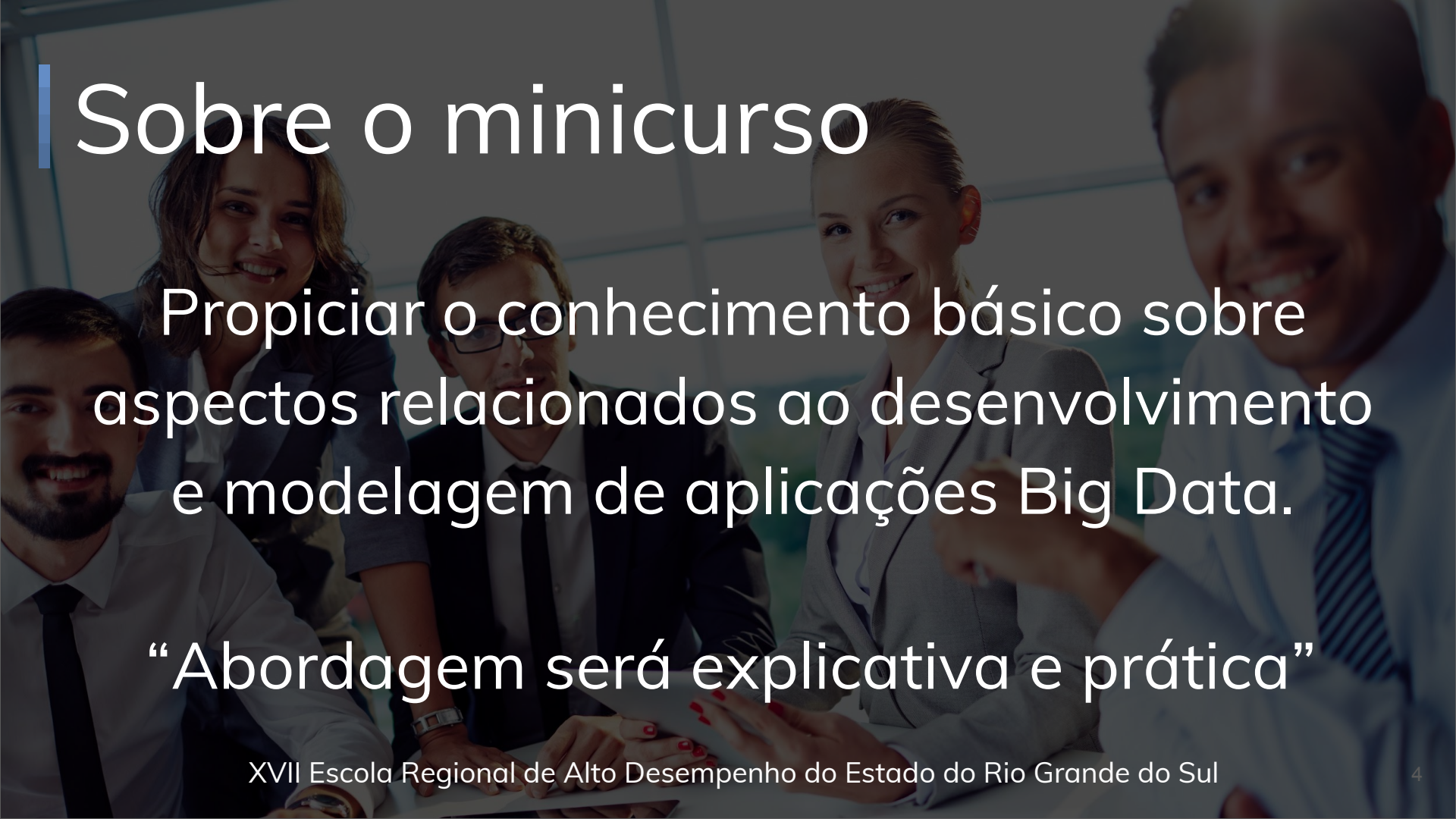
Professor Titular

GPPD

Grupo de Processamento Paralelo e Distribuído

- Sistemas Distribuídos
- Computação Ubíqua
- Big Data
- Sistemas híbridos
- Tolerância a falhas
- Escalonamento
- Cloud
- Gerenciamento de Recursos
- Cidades Inteligentes
- Inteligência Artificial

Sobre o minicurso



Propiciar o conhecimento básico sobre aspectos relacionados ao desenvolvimento e modelagem de aplicações Big Data.

“Abordagem será explicativa e prática”

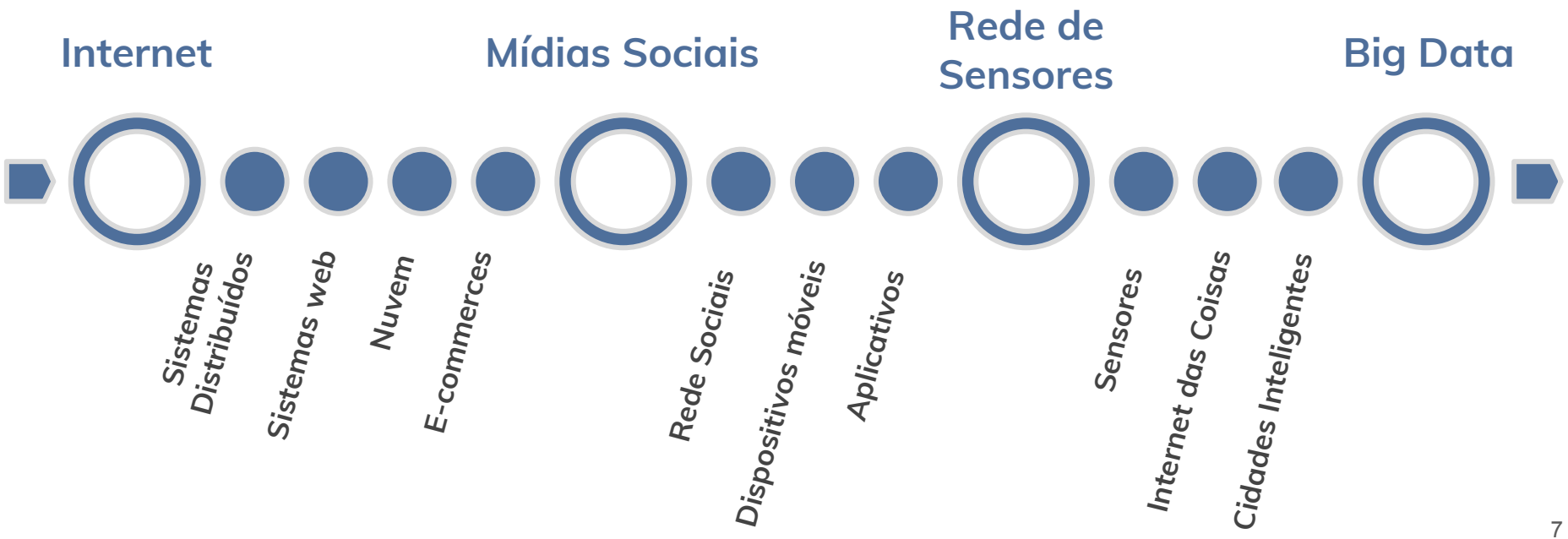
Agenda

- Introdução
- Big Data - Oportunidades e Perspectivas
- A arquitetura ZETA
- O Ecossistema Big Data vs ZETA
- Hands-on: Modelando uma Aplicação Big Data

Introdução

Introdução

A evolução das mídias sociais e a instrumentação da população.

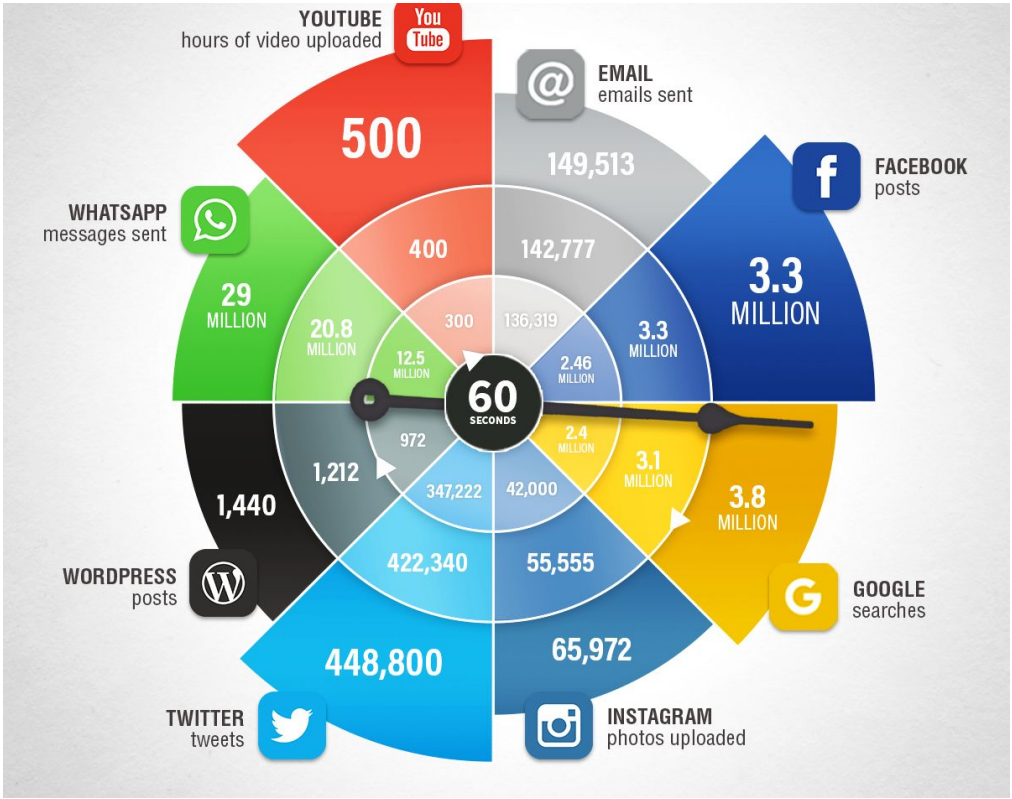


Introdução

Big Data

1. O grande volume de dados gerado pelas mais variadas fontes e características. Os 5 Vs do Big Data.
2. A importância do valor da informação para a indústria
3. A busca por soluções eficientes e de baixo custo
4. O desenvolvimento de aplicações e a complexidade envolvida

A movimentação de dados em 60 segundos

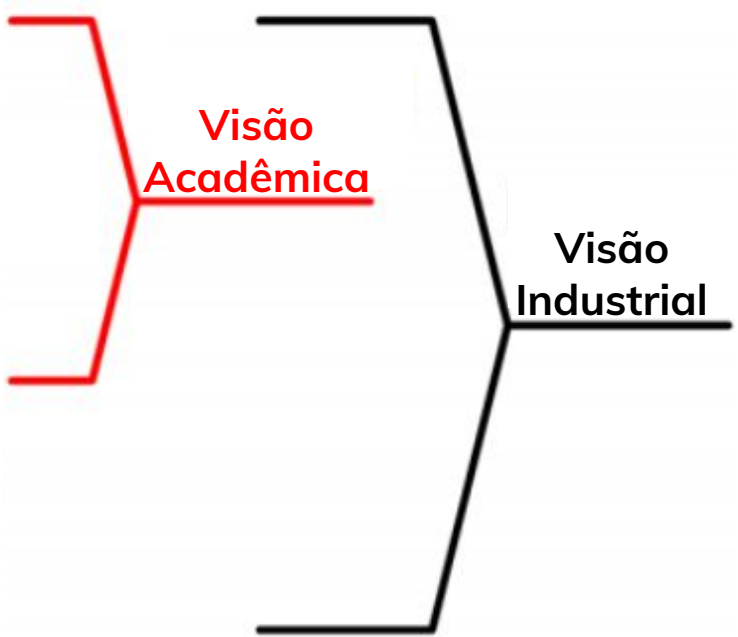


Smart insights, 2017

Introdução

Os 5 Vs do Big Data

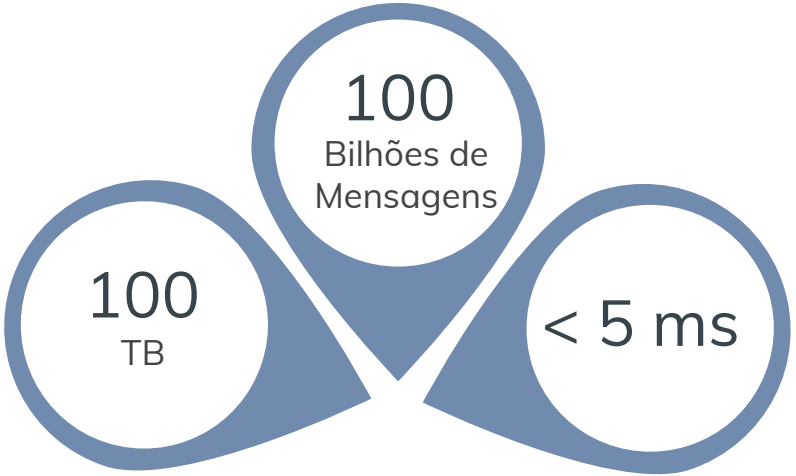
1. **Volume:** condiz ao tamanho e à quantidade de dados;
2. **Velocidade**: velocidade com que os dados são produzidos; e o quão rápido os mesmos devem ser processados;
3. **Variedade:** aos tipos de dados produzidos, uma vez que os mesmos possuem naturezas diversas;
4. **Veracidade:** confiabilidade sobre a fonte de dados - define o nível de incerteza da informação retirada do dado;
5. **Valor:** à extração de conhecimento de um dado e o quanto isso poderá ser agregado;



Big Data

Uber Eats

- Detectar fraudes
- Share my ETA (localização em tempo real)



Big Data

Oportunidades e Perspectivas

Perspectivas
Big Data

- Apoio à tomada de decisão
- Complexidade na modelagem e desenvolvimento de aplicações
- A escolha de técnicas e algoritmos é uma tarefa custosa
- Necessidade de um modelo de arquitetura abrangente

Oportunidades

Big Data

- Otimização de infraestrutura
- Otimização de recursos
- Redução de custos
- Grande demanda do mercado
- Soluções para a modelagem e desenvolvimento de aplicações Big Data

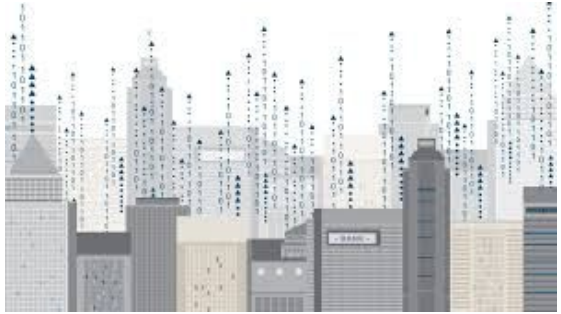
Zeta

Uma Arquitetura **Conceitual** de **Alto Nível**

A Arquitetura Zeta

Modelagem e desenvolvimento de soluções Big Data

Desafios no desenvolvimento de Soluções Big Data ligados a Indústria e a Academia



Zeta reduz a complexidade inerente ao desenvolvimento de soluções Big Data

Um modelo padronizado e ajustável a ser seguido.

Zeta

A Arquitetura e Suas Camadas

- Orquestração e Gerenciamento de Recursos
- Armazenamento Distribuído
- Frameworks
- Arquitetura da Solução
- Aplicação
- Sistemas Virtualizados
- Armazenamento em Tempo Real

O Ecossistema BIG DATA Sob a Perspectiva da Arquitetura ZETA

Arquitetura de solução

Arquitetura da Solução

Modelos de processamento

Processamento em Lote (Batch Jobs)

- Processamento em larga escala;
- Os dados são recebidos/coletados e armazenados para posterior processamento;
- Os trabalhos (jobs) são submetidos em sequência e em grande quantidade;
- Exemplos: sistemas para o cálculo de folhas de pagamento e faturamento.

Arquitetura da Solução

Modelos de processamento

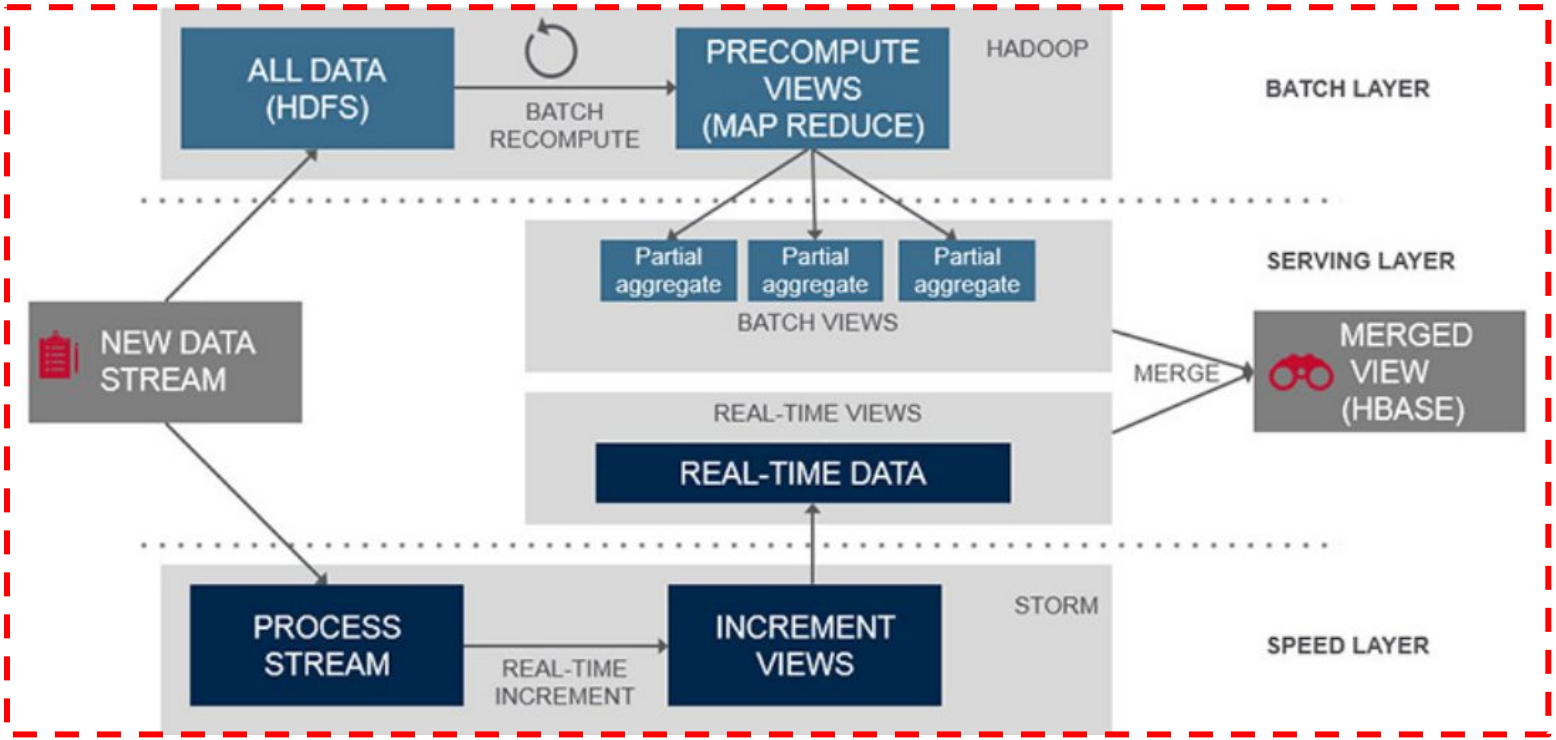
Processamento em Tempo Real (Streaming)

- Os dados são gerados e processados em tempo real e de forma incremental, utilizando janelas ou períodos móveis de tempo/número de tuplas;
- Os trabalhos (jobs) são pequenos (Micro batch);
- O processamento de dados imediato gera uma nova cadeia de processamento
- Exemplos: logs de sites e-commerce, dados de redes sociais.

Arquitetura da Solução

Arquiteturas para processamento

Lambda

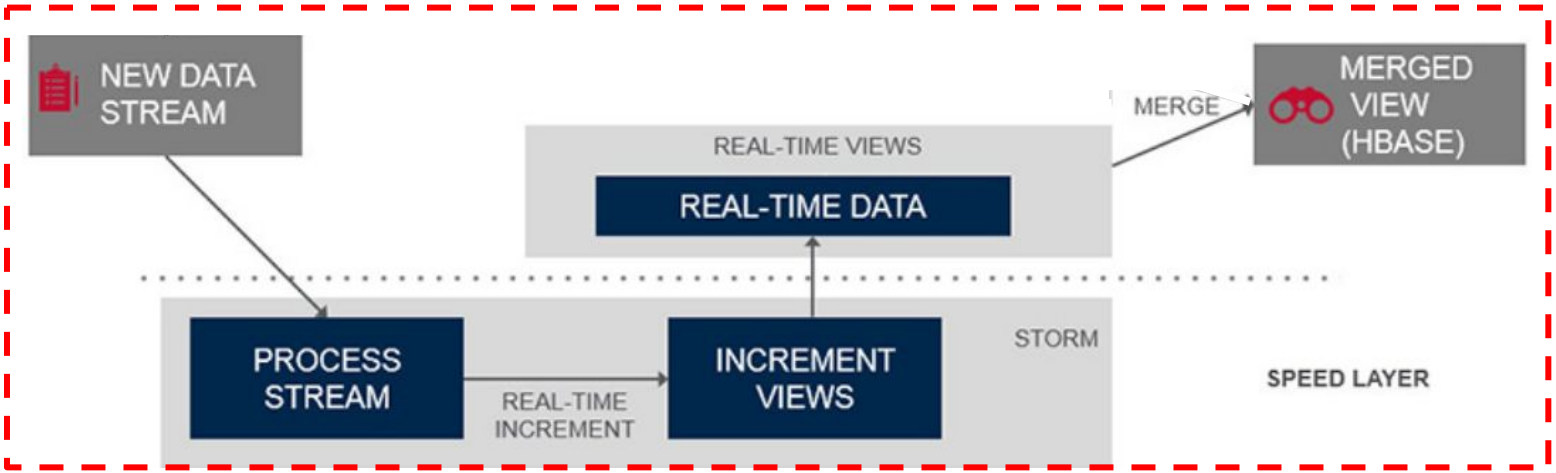


Scott, 2015

Arquitetura da Solução

Arquiteturas para processamento

Kappa



Scott, 2015

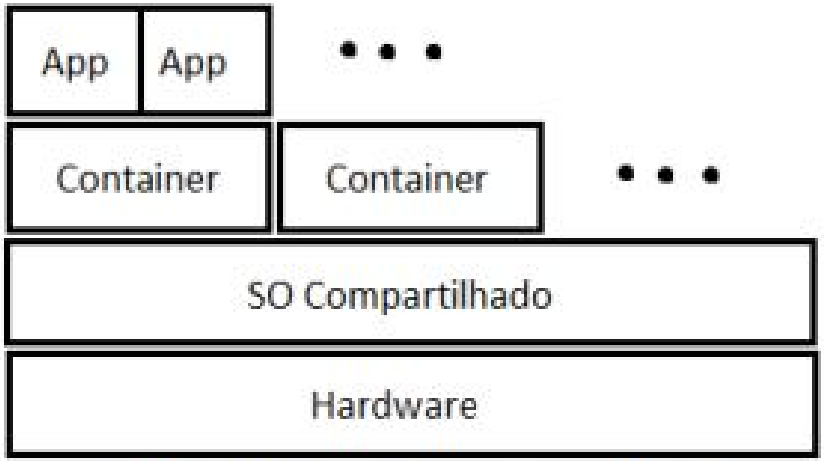
Tabela Comparativa Entre Arquiteturas

Plataforma Característica	Kappa	Lambda
Modo de Processamento	Tempo Real (Micro Batch)	Lote e Tempo Real (Batch e Micro Batch)
Reprocessamento	Somente quando o algoritmo sofrer alterações.	A cada ciclo (batch) ou quando o algoritmo sofrer alterações (Micro batch).
Acesso aos Dados	Algoritmos incrementais	Algoritmos incrementais e views sobre todos os dados
Confiabilidade	Pode variar	Batch confiável e tempo real pode variar

Sistemas Virtualizados

Sistemas Virtualizados

- Virtualização baseada em contêineres
- Desempenho semelhante ao nativo
- Suportado pelo YARN e Mesos



Virtualização Baseada em Container

Sistemas Virtualizados

Exemplos

LXD
CANONICAL

LXC

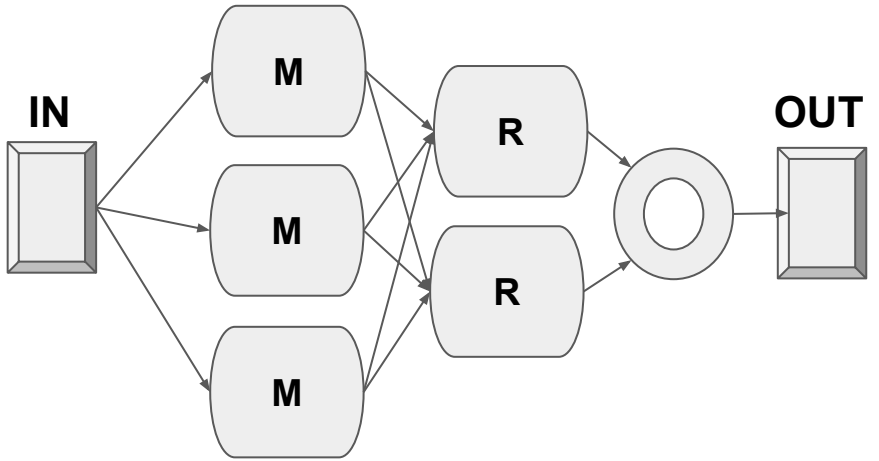
Frameworks de Processamento

XVII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul

MapReduce (2004)

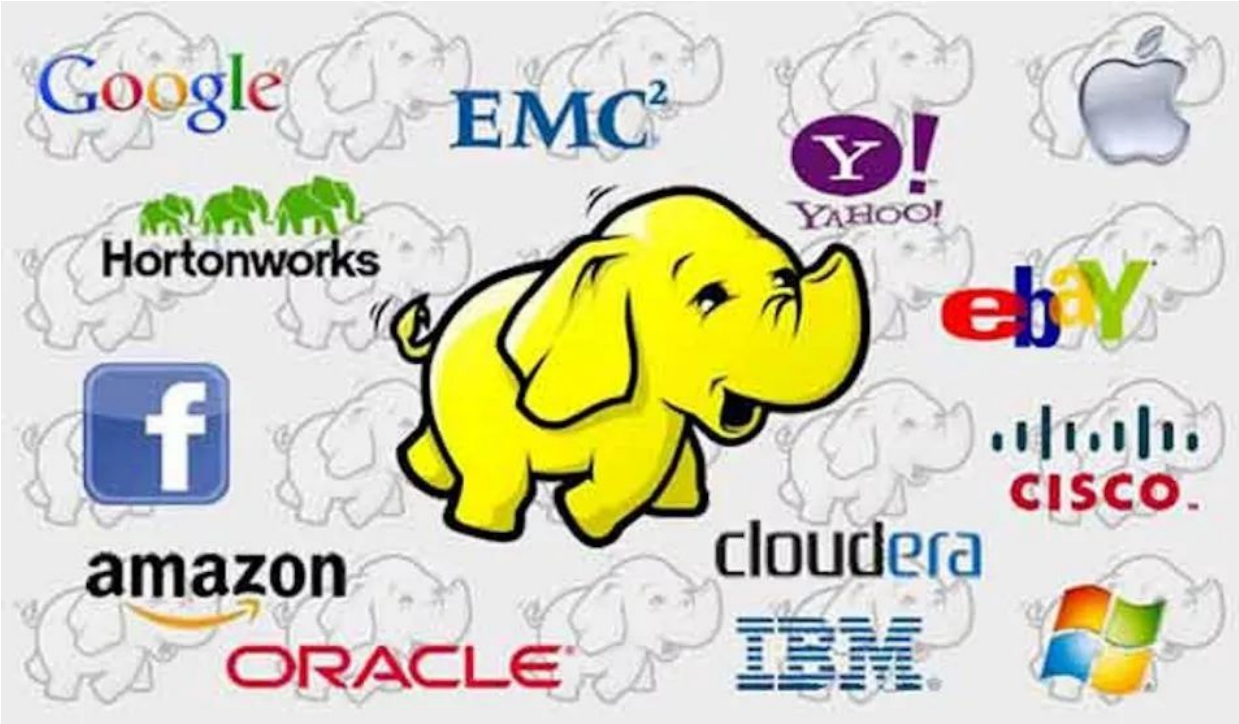
O modelo MapReduce e suas principais implementações.

- MapReduce - Batch
- Hadoop MapReduce
- Hadoop Streaming



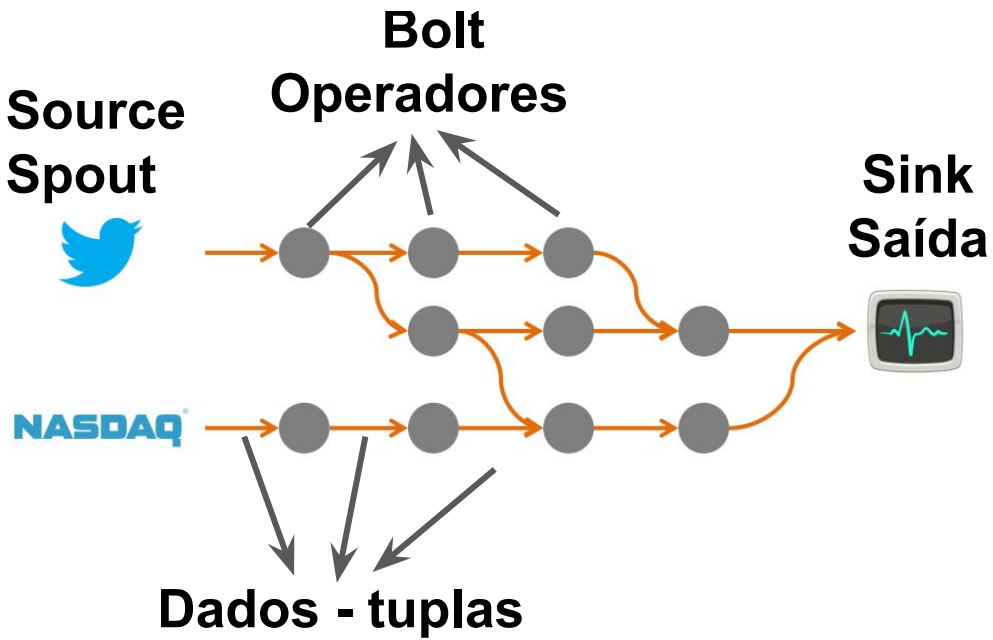
MapReduce

Exemplos



Storm (2011)

- Orientado a eventos
- Bolt - funções de processamento
- Spout - fonte de dados (tuplas)
- Toda mensagem é processada
- Suporte a múltiplas linguagens
- Kafka, YARN, HDFS



Storm

Exemplos

Spark (2014)

- Resilient Distributed Datasets (RDDs)
- Micro Batch
- Baixa latência
- Suporte a várias APIs
- Mesos e YARN
- HDFS, NoSQL, HBase e Local

Spark

Exemplos

Flink

Exemplos

- **Lote e tempo real (computação Híbrida)**
- **Suporte a Infraestrutura Heterogênea**
- **Iterações incrementais e não incrementais**
- **Suporte a múltiplas linguagens**
- **Possui mecanismos internos de otimização**
- **Não possui storage próprio**

Flink

Exemplos

Armazenamento

Armazenamento Distribuído

- HDFS
- S3

Armazenamento em Tempo Real

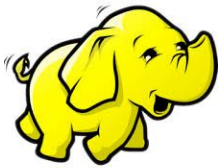
- MongoDB, Cassandra
- Pipeline-DB - relacional
- HBASE
- Geode, Ignite - in-memory

Gerenciamento de Recursos

Orquestração e Gerenciamento de Recursos

Mesos

- Gerenciamento global de recursos de um cluster.



YARN

- Otimizado para aplicações HMR e processamento em lote.

Arquitetura em Alto Nível

Sumário inerente às tecnologias previamente apresentadas

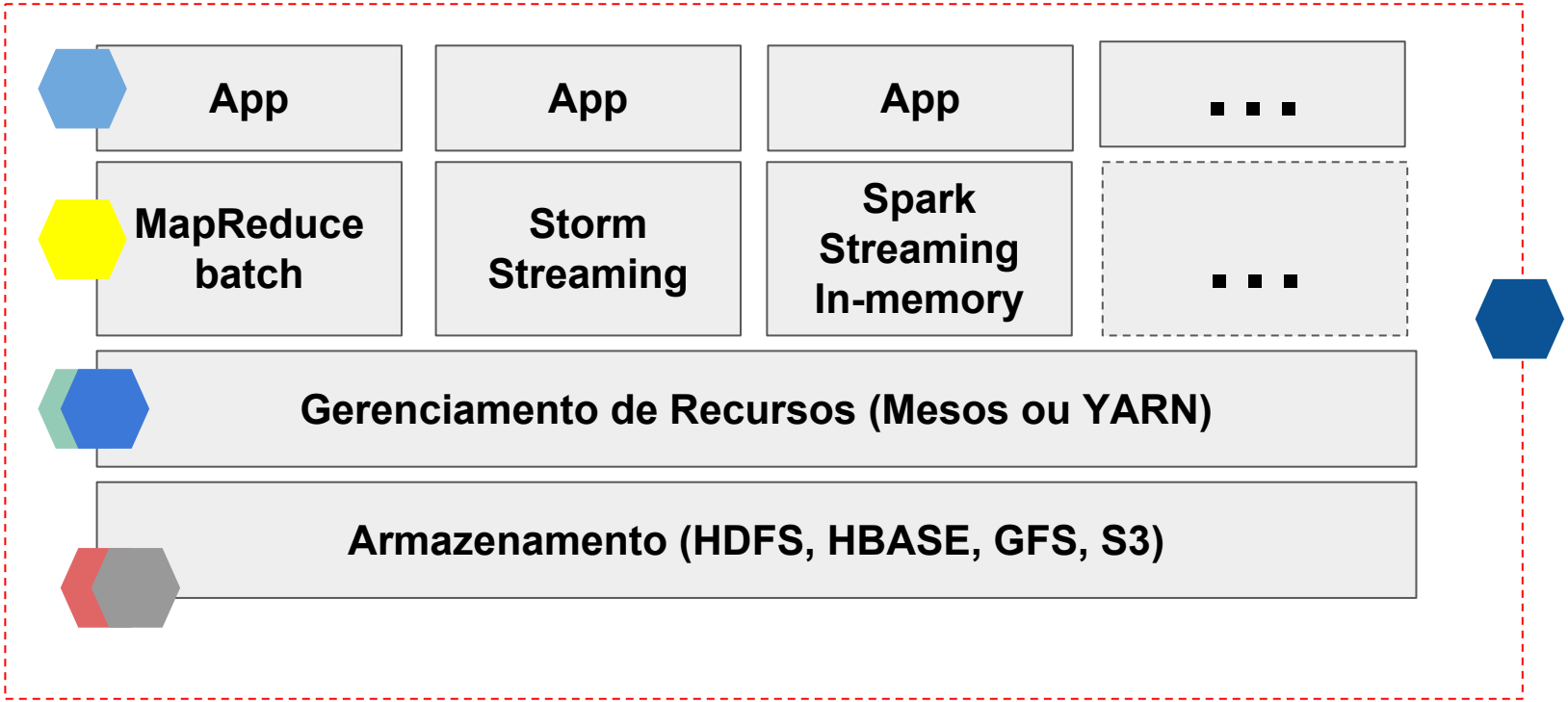


Tabela Comparativa: Frameworks de processamento

	Hadoop	Spark	Storm	Flink
Computação Incremental	Não	Sim	Sim	Sim
Modo de Processamento	Lote	Tempo Real In-memory	Tempo Real	Lote e Tempo Real
Linguagens	Java, R	Java, Scala, Python, R	Java (principal), Ruby	Java, Scala e Python
Gerenciamento	YARN, Mesos e local	YARN, Mesos e local	YARN, Mesos e local	YARN, Mesos e local
Outros	Grande Capacidade de Armazenamento; APIs para Streaming e Monitoramento	Inúmeras APIs para Criar Aplicações Iterativas e incrementais	Adequado para Aplicações em Tempo Real	Suporta Infraestrutura Heterogênea; Ideal para Aplicações IoT

Aplicação

Aplicação

Desenvolvimento de soluções (IoT, cloud, Big Data….)

- **Análise de sentimento**
- **Aprendizagem de máquina**
- **Gerenciamento de *logs***
- **Detectar fraudes**
- **Recomendação**
- **Classificação**
- **BI**
- **Agrupamento - clusterização**

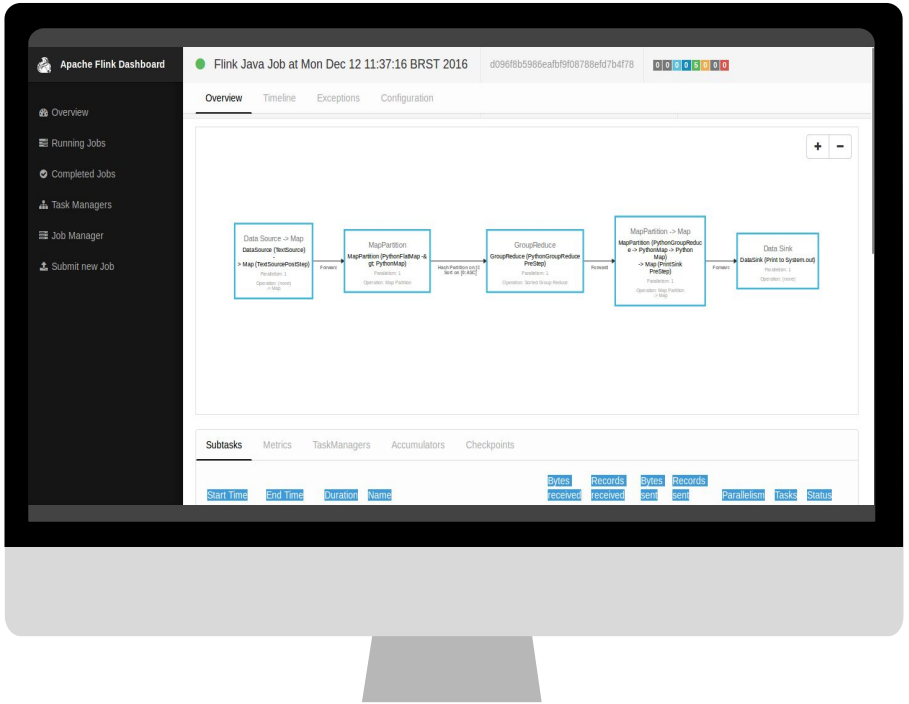
Hands-on

Desenvolvendo aplicações Big Data

Aplicação

Classificação de Tweets com Naive Bayes

- Dividida em duas etapas
  - Etapa de coleta de bigramas das bases de conhecimento negativas e positivas
  - Etapa de classificação de um tweet genérico



Streaming de tempo real por *hashtag*

Consultas e inserções constantes no BD

Hands-on

Aplicação de classificação de Tweets

Primeira etapa da aplicação

Passo um

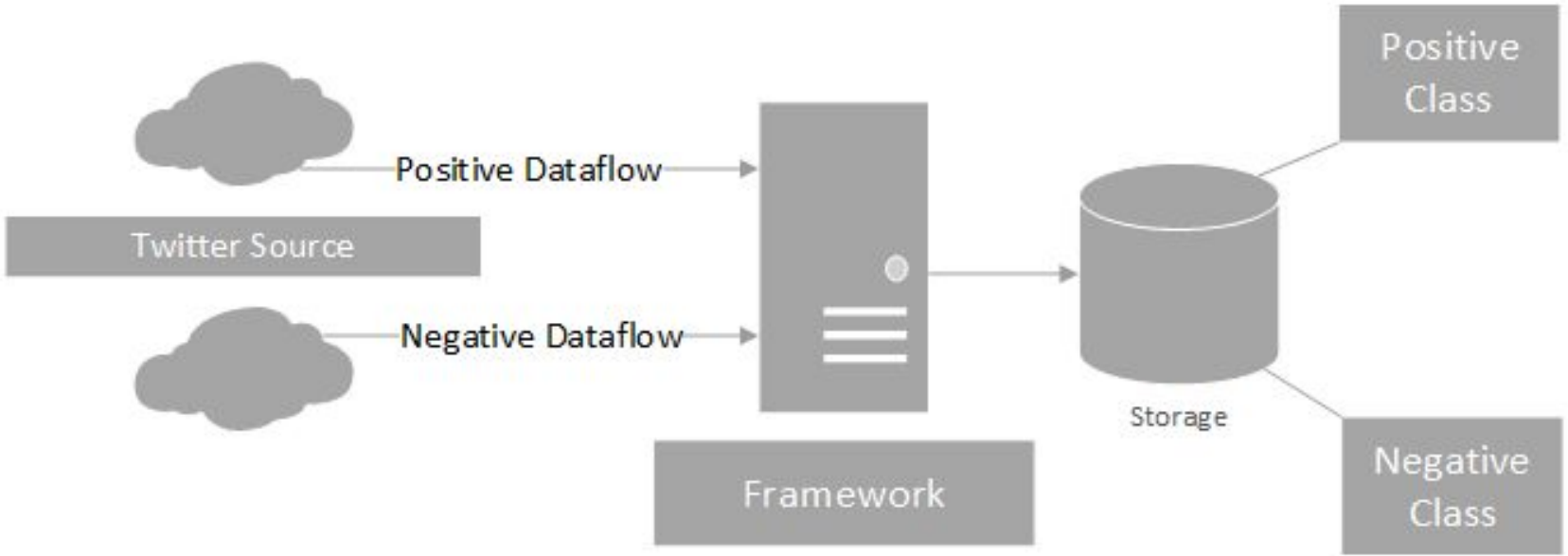
Passo três

Passo dois

Passo quatro

Primeira Etapa

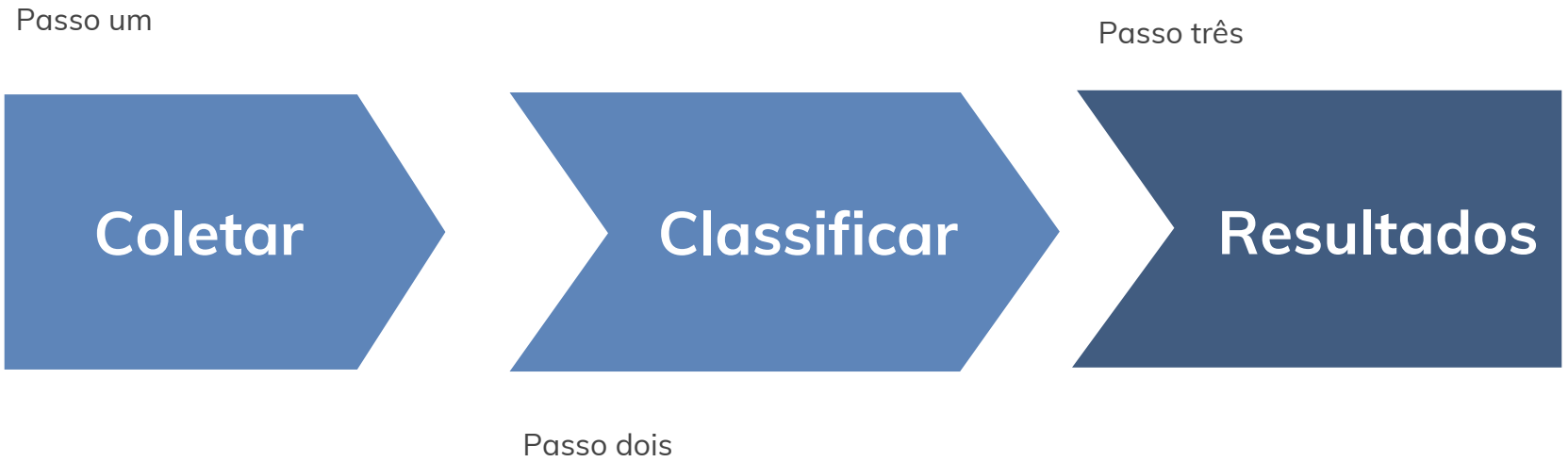
Coleta e armazenamento



Hands-on

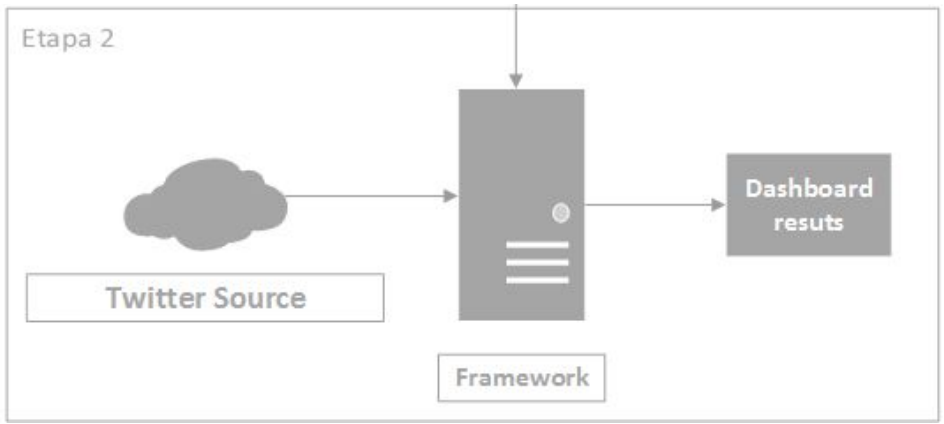
Aplicação de classificação de Tweets

Segunda etapa da aplicação



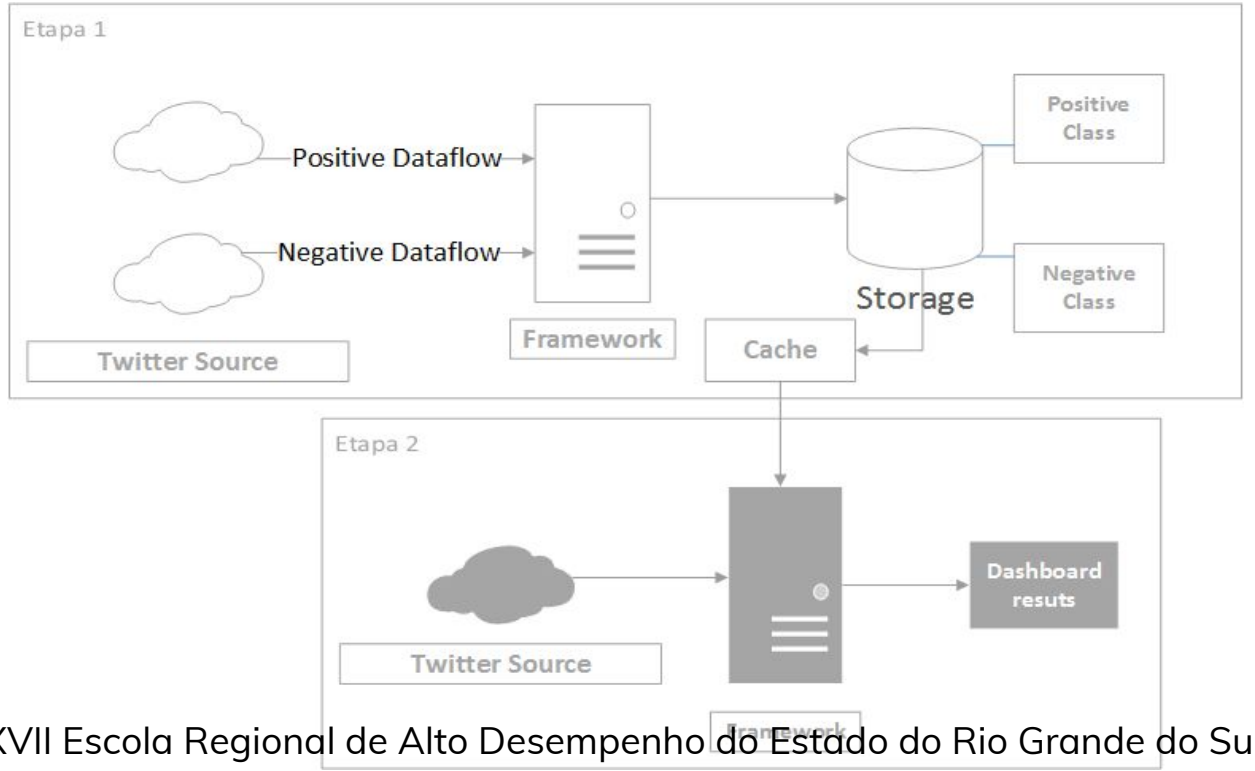
Segunda Etapa

Coleta e classificação



Aplicação Completa

Classificação de tweets

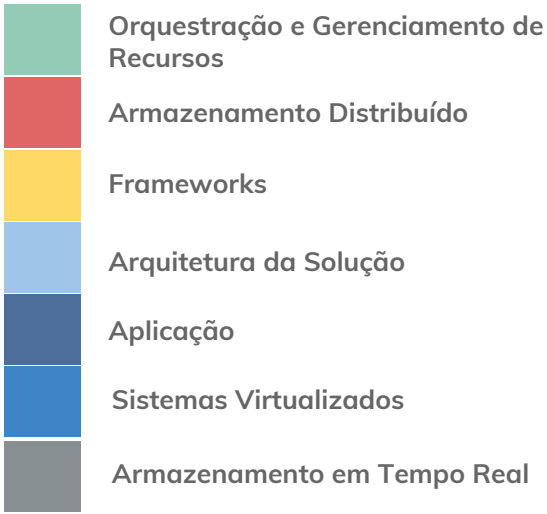


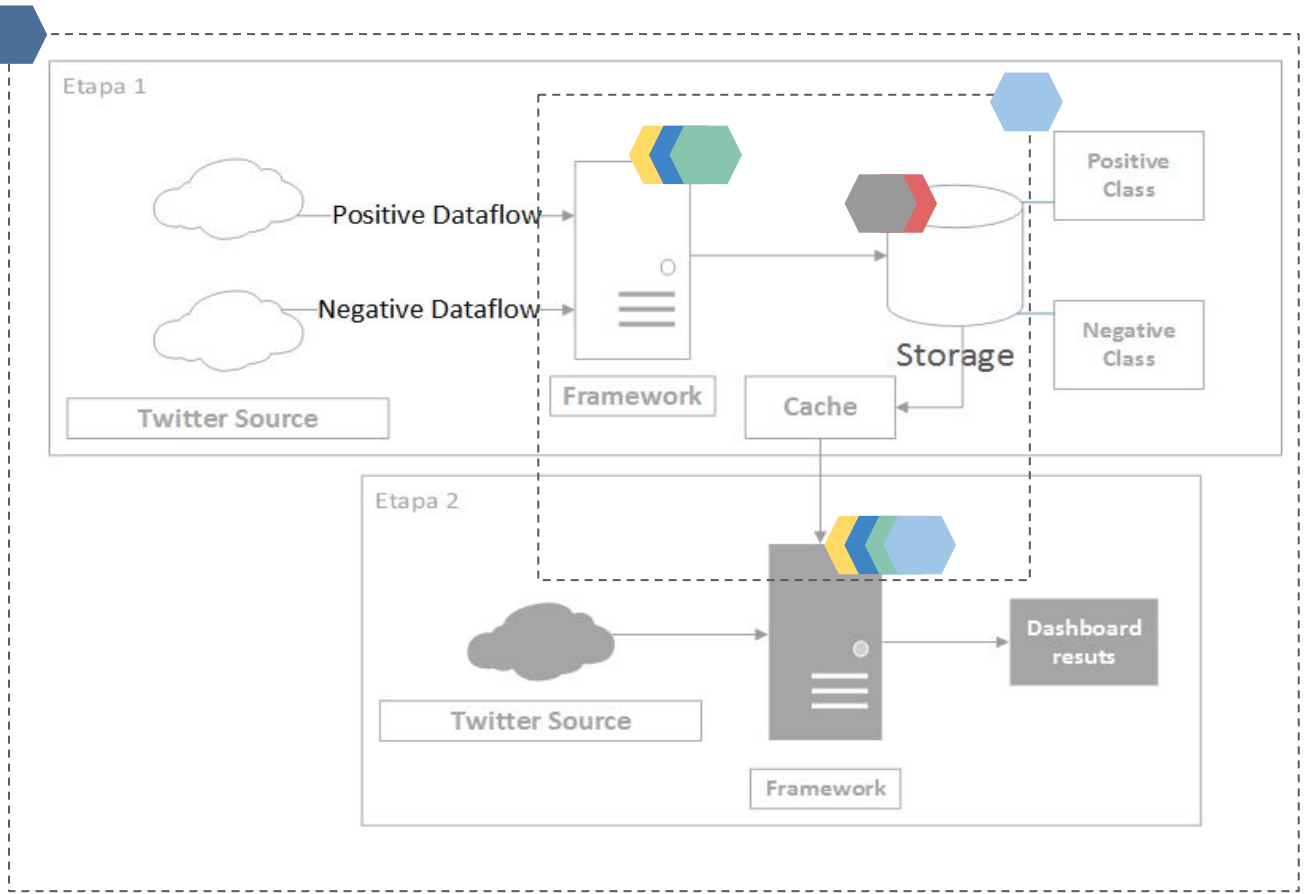
Arquitetura **Zeta**

Como selecionar as tecnologias para minha aplicação?

Agora serão apresentados tecnologias que podem ser utilizadas como solução à aplicação proposta. A fim de mostrar os conceitos propostos pela arquitetura Zeta na definição de um ambiente para uma aplicação.
Após a justificativa do uso das tecnologias será apresentado o desenvolvimento desta aplicação.

Aplicação


Orquestração e Gerenciamento de Recursos
Armazenamento Distribuído
Frameworks
Arquitetura da Solução
Aplicação
Sistemas Virtualizados
Armazenamento em Tempo Real


Etapa 1
Positive Dataflow
Negative Dataflow
Twitter Source
Framework
Storage
Cache
Positive Class
Negative Class
Etapa 2
Twitter Source
Framework
Dashboard resuts

Aplicação

Cenário Hipotético

- Desenvolvedor apenas conhece Python :(
- Não são necessários filtros durante a coleta
- Banco de dados
  - Acesso constante
  - Dados atualizados no momento que chegam
- Re-classificar após períodos de tempo

- **Balanceamento de recursos ?**
- **Fila de mensagens ?**

Frameworks de processamento

	Hadoop	Spark	Storm	Flink
Computação Incremental	Não	Sim	Sim	Sim
Modo de Processamento	Lote	Tempo Real In-memory	Tempo Real	Lote e Tempo Real
Linguagens	Java, R	Java, Scala, Python, R	Java (principal), Ruby	Java, Scala e Python
Gerenciamento	YARN, Mesos e local	YARN, Mesos e local	YARN, Mesos e local	YARN, Mesos e local
Outros	Grande Capacidade de Armazenamento; APIs para Streaming e Monitoramento	Inúmeras APIs para Criar Aplicações Iterativas e incrementais	Adequado para Aplicações em Tempo Real	Suporta Infraestrutura Heterogênea; Ideal para Aplicações IoT

Bancos de dados

	Cassandra	HBase	MongoDB
Relacional	Não	Não	Não
Distribuído	Sim	Sim	Sim (em partes)
API	Java, Python ...	Java, Python ...	Java, Python ...
Desempenho	Razoável	Superior	Superior
Tempo real	Indicado	Indicado	Indicado

Aplicação

Cenário Hipotético

	Aplicação
Arquitetura de Solução	Lambda
Framework	Flink
Base de Dados	MongoDB
Virtualização	NA
Filas	NA

Flink

API Python

	Descrição
Map	**Recebe um elemento e produz outro.** data.map(**lambda** x: x * 2, INT)
FlatMap	**Recebe um elemento e produz zero, um ou vários.** data.flat_map(**lambda** x,c: [(1,word) **for** word **in** line.lower().split() **for** line **in** x], (INT, STRING))
Filter	**Avalia uma função booleana para cada elemento e retém aqueles para os quais a função retorna true.** data.filter(**lambda** x: x > 1000)

Flink

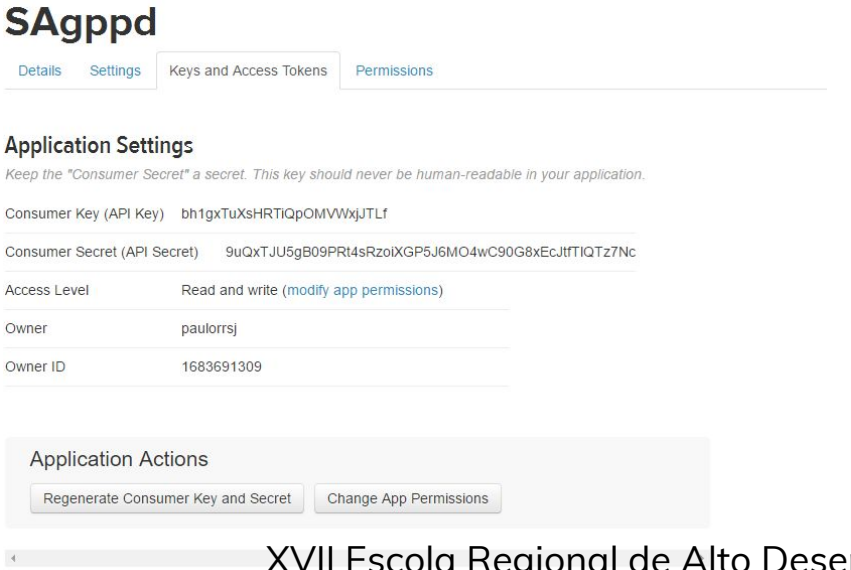
API Python

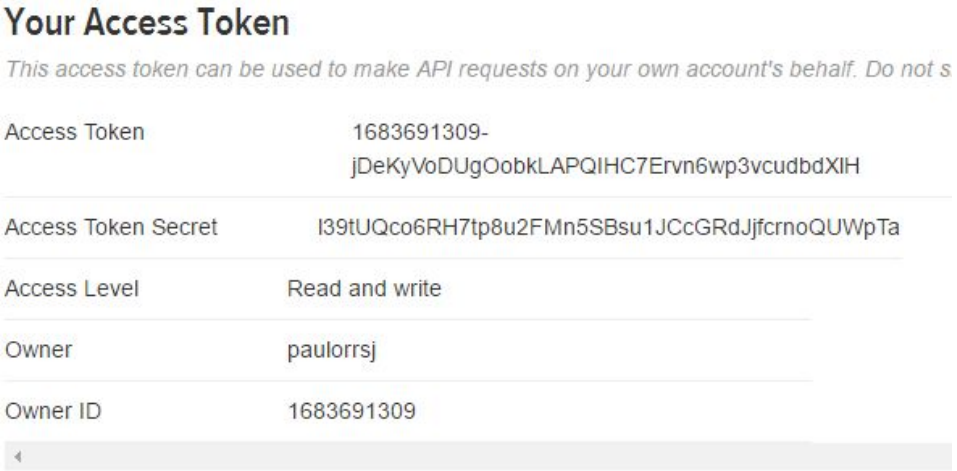
	Descrição
Reduce	**Combina um grupo de elementos em um único elemento, combinando repetidamente dois elementos em um único.** `data.reduce(lambda x,y : x + y)`
ReduceGroup	**Combina um grupo de elementos em um ou mais elementos.** `data.reduce_group(Adder(), (INT, STRING))`
Join	**Cruza dois conjuntos de dados criando todos os pares de elementos que são iguais em suas chaves. Opcionalmente, usa um JoinFunction para transformar o par de elementos em um único elemento**
Union	**Une dois datasets.** `data.union(data2)`

Implementação

Coleta dos dados

- **Coleta realizada utilizando a API do Twitter, através de uma *hashtag*.**
  - Tweepy (http://www.tweepy.org/)





Implementação

Coleta dos dados

- **Coleta realizada utilizando a API do Twitter, através de uma *hashtag*.**
  - Tweepy

```
import tweepy
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream

from pymongo import MongoClient

consumer_key = '                                '
consumer_secret = '
access_token = '
access_secret = '

l = StreamRead()
auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
stream = Stream(auth, l)

stream.filter(track=['#trump'])
```

```
class StreamRead(StreamListener):
    def __init__(self):
        super().__init__()
        self.counter = 0
        self.limit = 1000
        print("Collecting")

    def on_status(self, status):
        tweets.append(status.text)
        print(self.counter, status.text)
        thefile.write("%s;\n" % status.text)
        self.counter += 1
        if self.counter > self.limit:
            print("Reached limit")
            self.counter = 0
            runner()
            return True

    def on_error(self, status_code):
        return False
```

Implementação

Etapa de coleta de dados

- Para cada tweet coletado, gerar strings dos bigramas possíveis.
- “Bigramas são duplas de palavras em sequência.”
  - Bigramas são
  - são duplas
  - duplas de
  - de palavras
  - palavras em
  - em sequência.

```
>>> generic("Este é apenas um exemplo")
[(1, 'Este é'), (1, 'é apenas'), (1, 'apenas um'), (1, 'um exemplo')]
```

Implementação

Exemplo *word count*

```
data \
        .flat_map(lambda x, c: [(1, word) for word in x.lower().split()]) \
        .group_by(1) \
        .reduce_group(Adder(), combinable=True) \
        .map(lambda y: 'Count: %s Word: %s' % (y[0], y[1])) \
        .write_text(output_file, write_mode=WriteMode.OVERWRITE)
```

Hands-on

Flink

Inicializar os serviços do Flink

```
~/Documents/flink-1.1.3/bin$ ./start-cluster.sh
Starting cluster.
Starting jobmanager daemon on host Nulheim.
Starting taskmanager daemon on host Nulheim.
```

Importante que o jobmanager e taskmanager sejam iniciados, confirmar com o comando:

```
jps
22848 TaskManager
23192 Jps
22443 JobManager
```

Hands-on

MongoDB

Inicializar os serviços do Mongo

```
sudo service mongod start
```

Confirmar execução do serviço com:

```
sudo service mongod status
● mongod.service - High-performance, schema-free document-oriented database
   Loaded: loaded (/lib/systemd/system/mongod.service; disabled; vendor
preset: enabled)
   Active: active (running) since Seg 2017-04-03 13:25:10 BRT; 1s ago
```

Implementação

Tarefa 1

- Mapear os dados
  - Obter os bigramas
- Agrupar
  - Por bigramas iguais
- Reduzir
  - realizar a soma dos dados iguais
- Armazenar no Banco de Dados
  - Utilizar a API do Pymongo (próximo slide)

Implementação

Pymongo API

```
#Connect to mongo

client = MongoClient('localhost', 27017)

#Get database

db = client['DB_NAME']

#Get collection

cl = db['collection_name']
```

```
cl.insert({"_id":brigram, "value":value})

cl.find()

cl.update()

cl.find_one()
```

Implementação

Etapa de classificação do dado

Teorema de Bayes:

X= “Bigramas são duplas de palavras em sequência.”

C1,C2 = classe positiva e negativa

p(C1|X)*= p("bigramas são"| C1) * p("são duplas"| C1) … p("em sequência."| C1) * p(C1)

p(C2|X)*= p("bigramas são"| C2) * p("são duplas"| C2) … p("em sequência."| C2) * p(C2)

Probabilidade do tweet X pertencer a classe C1 ou C2

Implementação

Etapa de classificação do dado

- Contar a quantidade de bigramas no Tweet e produzir as tuplas.
- Para cada tupla gerada é necessário obter a probabilidade do bigrama acontecer nas classes C1, C2.
- Reduzir os dados com o produtório das probabilidades dos bigramas.
- Apresentar os resultados normalizados em um arquivo de texto.

Links úteis

API Python

- https://ci.apache.org/projects/flink/flink-docs-release-0.9/apis/python.html
- http://api.mongodb.com/python/current/tutorial.html
- https://docs.mongodb.com/getting-started/shell/query/
- http://www.tweepy.org/


?
Obrigado