



# GERÊNCIA DE INFRAESTRUTURA PARA BIG DATA

---

Tiago Coelho Ferreto – Aula 04

Pós-Graduação em  
Ciência de Dados e Inteligência Artificial

# *Ementa da disciplina*

Introdução à arquitetura para Big Data Analytics. Visão geral sobre Infraestrutura de armazenamento de dados para Big Data. Visão geral sobre Infraestrutura de computação e de rede para Big Data. Tópicos sobre virtualização e computação em nuvem. Plataformas de Big Data na nuvem: HDFS, Hadoop e MapReduce. Estudos de caso com Spark.

# Professores

## **MARCOS TAKESHI**

Professor Convidado

Especialista em Big Data na Semantix, que atua em diversos projetos de empresas do setor financeiro, telecom, varejo e saúde. Realiza análises de arquiteturas, infraestruturas, ambientes, sistemas e ferramentas big data, visando o correto funcionamento e performance. Formado em engenharia eletrônica pela Escola de Engenharia Mauá, pós-graduado em Administração de Empresas pela FGV-SP, MBA em Big Data na FIAP, e empreendedorismo no Babson College.

## **TIAGO COELHO FERRETO**

Professor PUCRS

É professor adjunto da Pontifícia Universidade Católica do Rio Grande do Sul. Possui Doutorado em Ciência da Computação pela PUCRS (2010) com Doutorado sanduíche na Technische Universität Berlin, Alemanha (2007-2008). Tem experiência na área de Ciência da Computação, com ênfase em Redes de Computadores, atuando principalmente nos seguintes temas: computação em nuvem, grades computacionais, virtualização, processamento de alto desempenho e gerência de infraestrutura de TI.

# Encontros e resumo da disciplina

## AULA 1

Para ser um profissional de Data Science é necessário ter paciência e construir um bom Network.

Empresas tem grande interesse em processar os dados e deles extrair informação com a finalidade de monetizar.

É bom estar no meio de pessoas que saibam mais do que você, sempre você tem que estar no meio de pessoas melhores.

**MARCOS TAKESHI**  
Professor Convidado

## AULA 2

O Spark possibilita a obtenção de resultados imediatos.

É importante você saber e conseguir atuar em mais de uma frente.

Certificações podem mostrar que você tem conhecimento do assunto.

**MARCOS TAKESHI**  
Professor Convidado

## AULA 3

Nos últimos anos a gente tem, a cada ano, um novo software auxiliando no processamento de grandes volumes de dados.

Além de armazenar e processar, eu tenho que conseguir extrair valor.

O Hadoop como a principal ferramenta para trabalhar com grandes volumes de dados.

**TIAGO COELHO FERRETO**  
Professor PUCRS

## AULA 4

A redundância garante a persistência da informação.

O HDFS é a principal fonte de dados de entrada e saída do Hadoop.

Como utilizar as aplicações Sqoop e Flume.

**TIAGO COELHO FERRETO**  
Professor PUCRS

## AULA 5

MapReduce uma solução de escalonamento e capacidade de processamento.

Hadoop Streaming como implementação de funções Map e Reduce em linguagens diferentes de Java.

O Pig como linguagem alternativa para programar MapReduce.

**TIAGO COELHO FERRETO**  
Professor PUCRS

## AULA 6

O Hive trabalha com a linguagem SQL com interações através de linhas de comando em formato shell.

O Spark tem como benefícios uma melhor performance, extensibilidade e melhor suporte para outros cenários.

O componente principal do Spark é o RDD (Resilient Distributed Dataset).

**TIAGO COELHO FERRETO**  
Professor PUCRS

**Ciências de Dados e Inteligência Artificial**

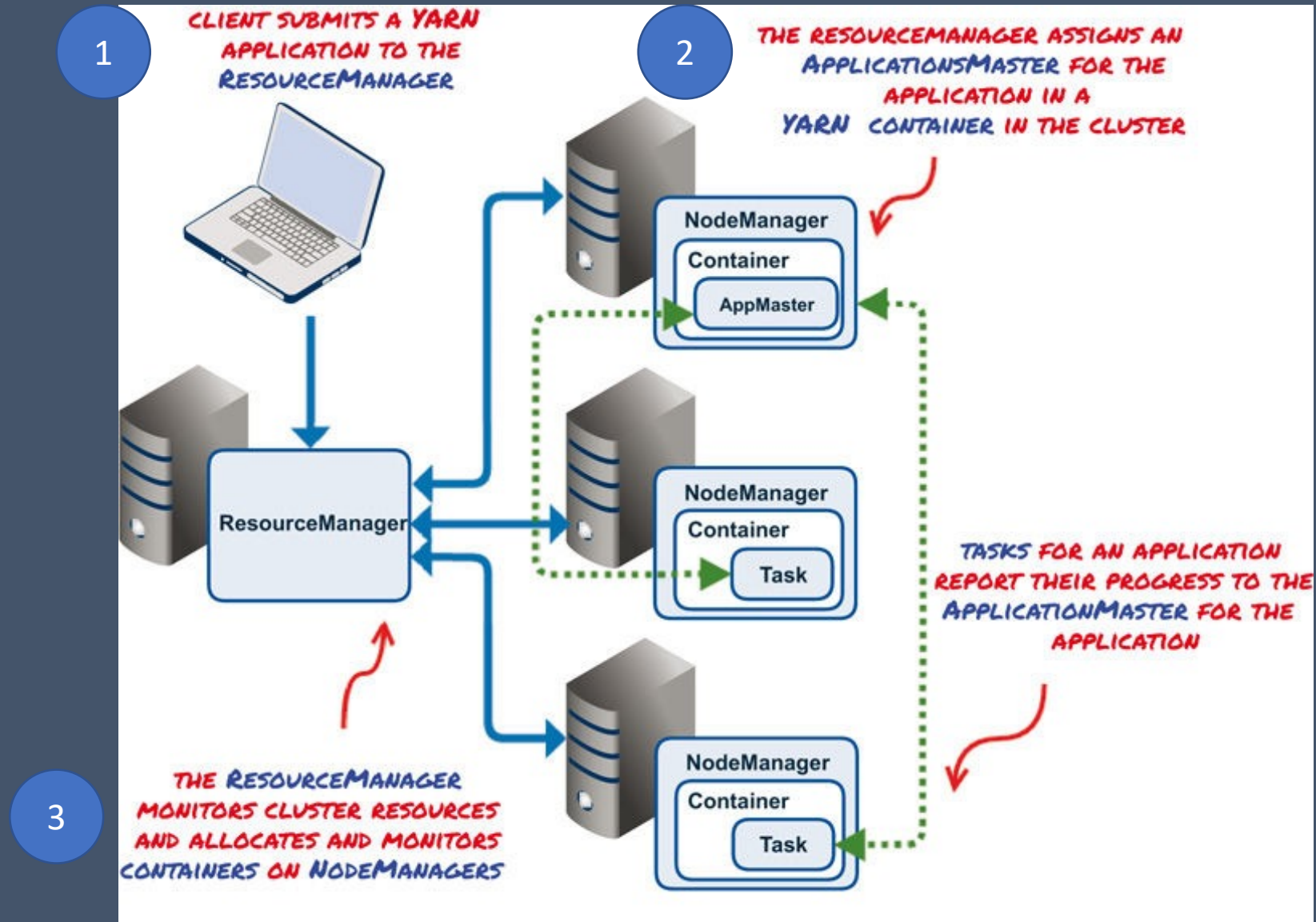
# Gerência de Infraestrutura para Big Data

YARN

Prof. Tiago Ferreto

*tiago.ferreto@pucrs.br*

# Arquitetura do cluster YARN



# Componentes do YARN

- **ResourceManager (RM)**
  - Daemon de nó mestre YARN
  - Concede recursos para aplicações na forma de **containers**
    - Unidades de computação predefinidas que consistem em memória e núcleos de CPU virtuais
  - **Monitora a disponibilidade de recursos do cluster**
    - Recebe **heartbeats** e **relatórios de status de nós escravos** (NodeManagers)
  - **Responsável pelo escalonamento no cluster e gerenciamento da concorrência**
  - Também pode ser implantado no modo **HA (alta disponibilidade)**
    - Evita se tornar um SPOF (Ponto Único de Falha)
    - Requer Zookeeper (serviço de coordenação de alto desempenho para aplicações distribuídas)

# Componentes do YARN

- **NodeManager (NM)**

- Daemon de nó escravo YARN
- Gerencia containers YARN em execução em nós de cluster
  - O primeiro container é dedicado ao **ApplicationMaster**
- Monitora o consumo e relata o progresso, status e integridade da aplicação para o RM

- **ApplicationMaster (AM)**

- Negocia com o RM por containers para executar tarefas da aplicação
- Gerencia a execução da aplicação - **orquestra os estágios de processamento (fases Map / Reduce)**



# Execução de aplicação no YARN

- **Pedidos de recursos**

- AM solicita recursos de cluster usando a **construção ResourceRequest**
  - ResourceRequests são solicitações de recursos de baixo nível enviadas ao RM em nome da aplicação que incluem o seguinte:
    - prioridade do pedido
    - quantidade de recursos de memória e CPU (vcore) necessários (por exemplo, 4 GB, 2 vcores)
    - número de recipientes necessários
    - informações de localidade de dados (por exemplo, preferências de rack e host para enviar os containers)
  - Se a solicitação for bem-sucedida, os containers solicitados serão concedidos
    - O escalonador do RM determina quando e quantos containers liberar (de acordo com a política de compartilhamento do escalonador)
- ResourceRequests são baseados em parâmetros de configuração fornecidos pelo desenvolvedor ou analista
  - Definido no arquivo *yarn-site.xml* ou fornecido pela aplicação enviada ao YARN

# Tolerância a falhas e recuperação de falhas

# Tolerância a falhas e recuperação de falhas

- **Falha de Tarefa**

- Se uma tarefa em execução falhar, o AM da aplicação reprograma a tarefa em outro nó (em conformidade com as especificações de localidade de dados definidas na solicitação de recurso)
- Uma **tarefa com falha é tentada novamente 4 vezes por padrão** (configurável)
- Depois que uma tarefa falha 4 vezes, a aplicação é considerado com falha
  - Todas as outras tarefas em execução são eliminadas
  - O status da aplicação (visto na UI do YARN ResourceManager) é definido como FALHA

# Tolerância a falhas e recuperação de falhas

- **Falha do NodeManager**

- NMs enviam heartbeats regulares (padrão: a cada 1000 ms) para o RM
- O NM é **considerado com falha se o limite de pulsação (propriedade *yarn.nm.liveness-monitor.expiry-interval-ms* ) for excedido** → padrão = 600000 ms (10 minutos)
- Tarefas em execução no NM serão consideradas como falha
- **As aplicações com o AM em execução no NM serão considerados com falha (aplicação inteira)**
- NM é removido da lista de nós ativos do YARN (lista negra) → não serão mais recipientes ou terão tarefas alocadas
- Quando o NM está ativo novamente, ele volta para a lista de nós ativos do YARN e pode ser alocado para novas tarefas

# Tolerância a falhas e recuperação de falhas

- **Falha do ApplicationMaster**

- Se o **processo ApplicationMaster** falhar, **todo a aplicação falhará**
- **A aplicação não é repetida por padrão** (pode ser modificado na *propriedade* `yarn.resourcemanager.am.max-retries` )
- Se uma aplicação for repetida, a aplicação será executada de acordo com a *propriedade* `yarn.app.mapreduce.am.job.recovery.enable` (yarn-site.xml)
  - Se falso (padrão) → todas as tarefas concluídas anteriormente na aplicação serão executadas novamente
  - Se verdadeiro → somente as tarefas falhas ou não executadas serão executadas

# Tolerância a falhas e recuperação de falhas

- **Falha do ResourceManager**

- O processo de RM e seu host são o **único ponto de falha** para um cluster YARN (se a alta disponibilidade (HA) não estiver ativada)
- Se o RM falhar e um standby não estiver disponível, nenhuma nova aplicação pode ser iniciada
  - As aplicações em execução no momento não são mais capazes de negociar recursos adicionais de que podem precisar e irão falhar
- Se HA estiver habilitado, o standby RM retomará automaticamente as funções e responsabilidades que o RM anteriormente ativo estava desempenhando, e o cluster YARN continuará a operar sem ser afetado

# Administração do YARN

# Administração do YARN

- Configuração YARN - *arquivo yarn-site.xml*
- O arquivo existe em todos os hosts que fazem parte de um cluster YARN
  - Todos os hosts podem conter o mesmo arquivo (por convenção)
  - Cada daemon lê apenas a configuração do seu serviço específico (a outra configuração é ignorada)
- Mudanças no yarn-site.xml requerem a reinicialização do daemon
- Valores padrão em: <https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-common/yarn-default.xml>

## ResourceManager



yarn-site.xml

```
<configuration>
<!--Generic YARN Properties-->
...
<!--ResourceManager Properties-->
...
<!--NodeManager Properties-->
...
</configuration>
```

## NodeManager



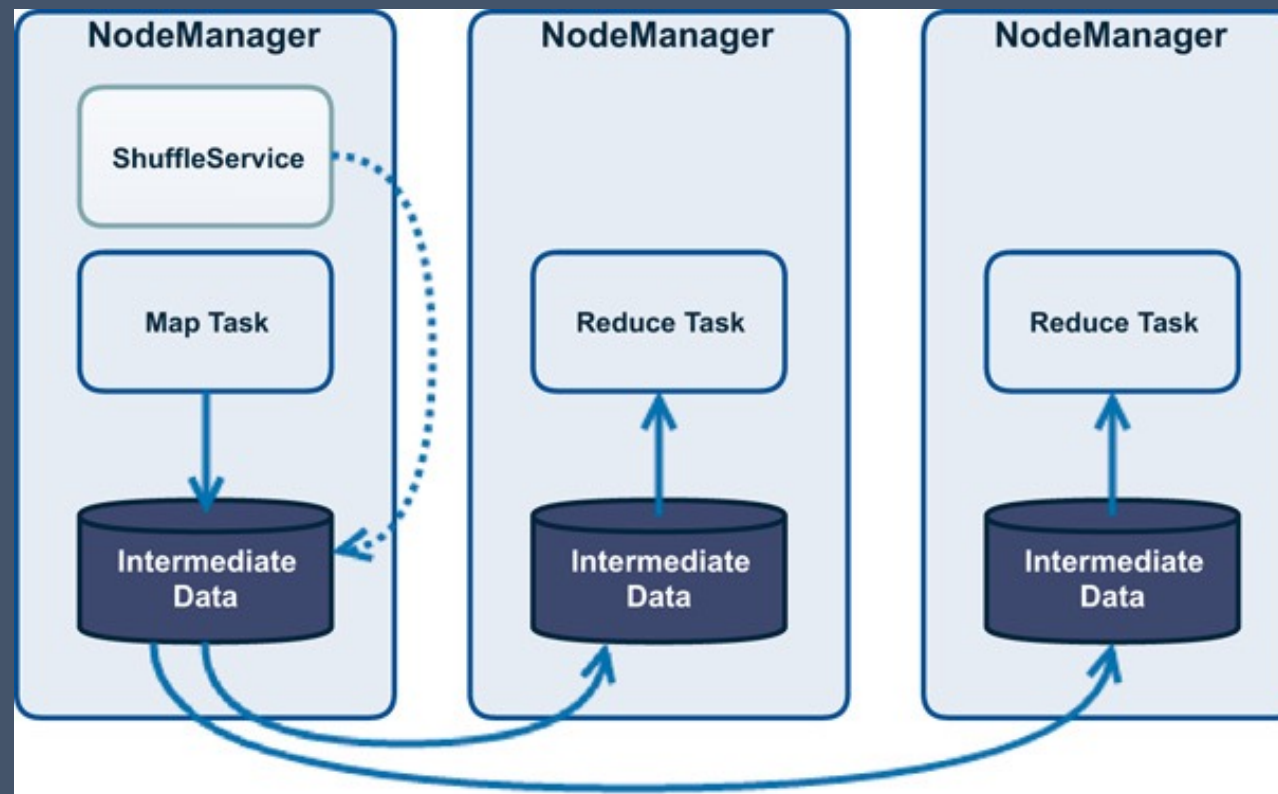
yarn-site.xml

```
<configuration>
<!--Generic YARN Properties-->
...
<!--ResourceManager Properties-->
...
<!--NodeManager Properties-->
...
</configuration>
```



# Serviços Auxiliares

- Os NodeManagers podem ser configurados para executar serviços auxiliares
  - Serviços auxiliares são executados em JVMs nos NodeManagers
- Exemplo mais comum → **ShuffleService**
  - Orquestra o processo Shuffle-and-Sort no Map Reduce



# Interface Web do YARN

- Facilita o gerenciamento de aplicações em execução no YARN
- É executado na porta 8088 do host ResourceManager
- Permite visualizar o status das aplicações (em execução, com falha, concluído), logs dos containers, etc.

The screenshot displays the Hadoop YARN web interface. At the top, the Hadoop logo is on the left, and the title "All Applications" is in the center. The browser address bar shows the URL "ec2-54-206-113-142.ap-southeast-2.compute.amazonaws.com:8088/cluster".

On the left side, there is a sidebar menu with the following items: "Cluster", "About", "Nodes", "Node Labels", "Applications", "NEW", "NEW SAVING", "SUBMITTED", "ACCEPTED", "RUNNING", "FINISHED", "FAILED", "KILLED", and "Scheduler".

The main content area is divided into two sections: "Cluster Metrics" and "Scheduler Metrics".

**Cluster Metrics**

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
22	0	1	21	1	896 MB	56.25 GB	0 B	1	80	0	2	0	0	0	0	0

**Scheduler Metrics**

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory 32, vCores 1>	<memory 11520, vCores 16>

Below the scheduler metrics, there is a table showing application details. The table has columns: ID, User, Name, Application Type, Queue, StartTime, FinishTime, State, FinalStatus, Progress, Tracking UI, and Blacklisted Nodes.

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1475830215563_0022	ec2-user	Spark Pi	SPARK	default	Sun Oct 9 17:31:51 +1100 2016	N/A	ACCEPTED	UNDEFINED		ApplicationMaster	0
application_1475830215563_0021	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:44:12 +1100 2016	Sat Oct 8 16:44:32 +1100 2016	FINISHED	SUCCEEDED		History	N/A
application_1475830215563_0020	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:44:06 +1100 2016	Sat Oct 8 16:44:23 +1100 2016	FINISHED	SUCCEEDED		History	N/A
application_1475830215563_0019	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:42:45 +1100 2016	Sat Oct 8 16:43:00 +1100 2016	FINISHED	SUCCEEDED		History	N/A
application_1475830215563_0018	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:42:45 +1100 2016	Sat Oct 8 16:43:00 +1100 2016	FINISHED	SUCCEEDED		History	N/A

# Interface de linha de comando YARN (CLI)

- Comando yarn
- Utilização
  - Visualizar o status e gerência de jobs em execução em um cluster YARN
  - Submeter uma aplicação (igual ao comando hadoop jar)
  - Visualizar arquivos de log
  - Executar processos YARN
  - Visualizar informações sobre containers, filas, etc.

## **\$ yarn --help**

Usage: yarn [OPTIONS] SUBCOMMAND [SUBCOMMAND OPTIONS]  
[...]

SUBCOMMAND is one of:

### Admin Commands:

daemonlog	get/set the log level for each daemon
node	prints node report(s)
rmadmin	admin tools
scmadmin	SharedCacheManager admin tools

### Client Commands:

applicationattempt	prints applicationattempt(s) report
app application	prints application(s) report/kill app/manage long running app
classpath	prints the classpath to the hadoop jar and the required libs
cluster	prints cluster information
container	prints container(s) report
envvars	display computed Hadoop environment variables
jar <jar>	run a jar file
logs	dump container logs
nodeattributes	node attributes cli client
queue	prints queue information
schedulerconf	Updates scheduler configuration
timelinereader	run the timeline reader server
top	view cluster information
version	print the version

...

### Daemon Commands:

nodemanager	run a nodemanager on each worker
proxyserver	run the web app proxy server
registrydns	run the registry DNS server
resourcemanager	run the ResourceManager
router	run the Router daemon
sharedcachemanager	run the SharedCacheManager daemon
timelineserver	run the timeline server

SUBCOMMAND may print help when invoked w/o parameters or with -h.

# YARN – Matando uma aplicação

- Ao executar o YARN no modo de cluster, sair do processo de invocação não mata o aplicativo
- Precisa usar o comando yarn

```
$ yarn application -list
```

```
...
```

```
Total number of RUNNING applications : 1
```

```
Application-Id                Application-Name ...
```

```
application_1475830215563_0024 org.apache.spark.examples.SparkPi ...
```

```
$ yarn application -kill application_1475830215563_0024
```

```
...
```

```
Killed application application_1475830215563_0024
```

# Agregação de log no YARN

- Agregação de log - **agrega logs de aplicativos e os move para HDFS para armazenamento de longo prazo**
  - Configuração ativada em `yarn.log-aggregation-enable = true` (arquivo de configuração `yarn-site.xml`)
- Por padrão, os logs das aplicações são armazenados no sistema de arquivos local do NodeManager durante a execução das tarefas
  - Pode ser acessado (durante a execução) pelo link ApplicationMaster para a aplicação no YARN ResourceManager
    - Redireciona para o host NodeManager relevante para os logs de tarefa (ou container)
  - Os logs das aplicações concluídas ainda ficam disponíveis por meio da UI da web
- Os logs agregados podem ser acessados usando o comando `yarn`

# Acessando registros agregados usando yarn

```
$ yarn logs -applicationId application_1475830215563_0024
```

```
Container: container_1475830215563_0024_01_000002 on ...
```

```
=====
```

```
LogType:stderr
```

```
Log Upload Time:Sun Oct 09 07:02:00 +0000 2016
```

```
LogLength:69033
```

```
Log Contents:
```

```
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
```

```
16/10/09 07:01:50 INFO SignalUtils: Registered signal handler for TERM
```

```
16/10/09 07:01:50 INFO SignalUtils: Registered signal handler for HUP
```

```
16/10/09 07:01:50 INFO SignalUtils: Registered signal handler for INT
```

```
16/10/09 07:01:51 INFO SecurityManager: Changing view acls to: yarn,ec2-user
```

```
16/10/09 07:01:51 INFO SecurityManager: Changing modify acls to: yarn,ec2-user
```

```
16/10/09 07:01:51 INFO SecurityManager: Changing view acls groups to:
```

```
16/10/09 07:01:51 INFO SecurityManager: Changing modify acls groups to:
```



# MRJobHistory Server

- O YARN não coleta métricas específicas de aplicações MapReduce
  - Foco em ser uma estrutura genérica e extensível (além do MR)
- *MRJobHistory* Server é responsável por coletar e armazenar métricas de aplicações MapReduce
  - Implementado como um daemon separado
  - Interface Web na porta 19888

# MRJobHistory Server

JobHistory

ec2-54-206-113-142.ap-southeast-2.compute.amazonaws.com:19888/jobhistory

hadoop

JobHistory

Logged in as: dcwaho

Application

About Jobs

Tools

Retired Jobs

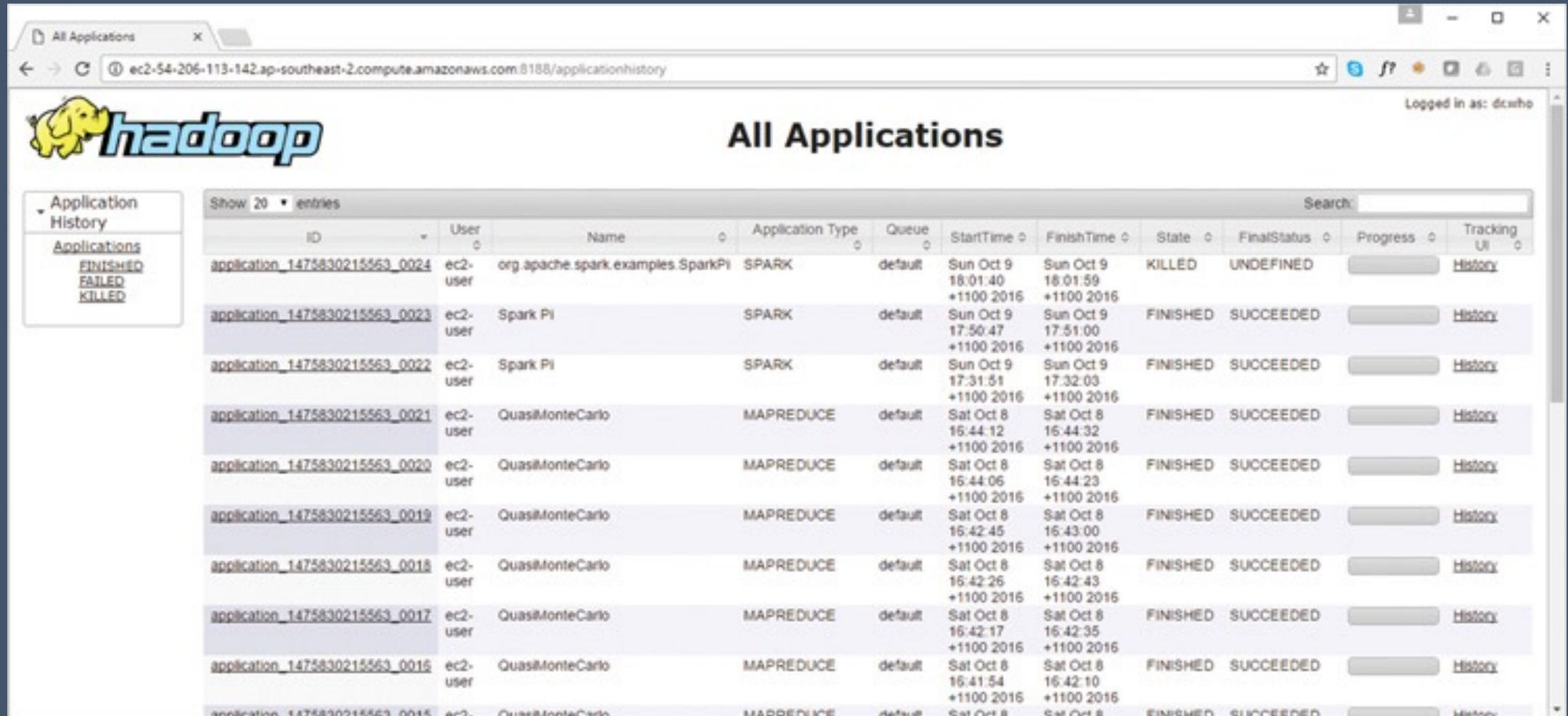
Show 20 entries

Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed
2016.10.08 05:44:12 UTC	2016.10.08 05:44:20 UTC	2016.10.08 05:44:32 UTC	job_1475830215563_0021	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	16	16	1	1
2016.10.08 05:44:06 UTC	2016.10.08 05:44:11 UTC	2016.10.08 05:44:23 UTC	job_1475830215563_0020	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	32	32	1	1
2016.10.08 05:42:45 UTC	2016.10.08 05:42:49 UTC	2016.10.08 05:43:00 UTC	job_1475830215563_0019	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	16	16	1	1
2016.10.08 05:42:26 UTC	2016.10.08 05:42:31 UTC	2016.10.08 05:42:43 UTC	job_1475830215563_0018	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	32	32	1	1
2016.10.08 05:42:17 UTC	2016.10.08 05:42:21 UTC	2016.10.08 05:42:35 UTC	job_1475830215563_0017	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	16	16	1	1
2016.10.08 05:41:54 UTC	2016.10.08 05:41:59 UTC	2016.10.08 05:42:09 UTC	job_1475830215563_0016	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	16	16	1	1
2016.10.08 05:41:09 UTC	2016.10.08 05:41:14 UTC	2016.10.08 05:41:27 UTC	job_1475830215563_0015	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	16	16	1	1
2016.10.08 05:40:58 UTC	2016.10.08 05:41:03 UTC	2016.10.08 05:41:14 UTC	job_1475830215563_0014	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	32	32	1	1
2016.10.08 05:40:19 UTC	2016.10.08 05:40:24 UTC	2016.10.08 05:40:35 UTC	job_1475830215563_0013	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	24	24	1	1
2016.10.08 05:39:56 UTC	2016.10.08 05:40:01 UTC	2016.10.08 05:40:12 UTC	job_1475830215563_0012	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	16	16	1	1
2016.10.08 05:39:18 UTC	2016.10.08 05:39:26 UTC	2016.10.08 05:39:37 UTC	job_1475830215563_0011	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	16	16	1	1
2016.10.08 05:39:10 UTC	2016.10.08 05:39:17 UTC	2016.10.08 05:39:30 UTC	job_1475830215563_0010	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	16	16	1	1
2016.10.08 05:38:53 UTC	2016.10.08 05:39:00 UTC	2016.10.08 05:39:12 UTC	job_1475830215563_0009	QuasiMonteCarlo	ec2-user	default	SUCCEEDED	16	16	1	1

# YARN Timeline Server

- Fornece **informações genéricas sobre aplicações concluídas** que são independentes de estrutura (por exemplo, tentativas de execução, escalonamento, utilização de recursos e informações de containers)
- Implementado como um daemon separado
  - Pode ser executado em um host diferente do ResourceManager
- Usa um armazenamento de dados LevelDB (armazenamento de valores-chave) para armazenar seus dados
- Dispõe uma interface web na porta 8188 e uma API REST
- O Hadoop 3 apresenta o Timeline Server V.2 com alguns aprimoramentos
  - Maior escalabilidade
  - Usa Apache HBase como o armazenamento de apoio primário (suporta um tamanho maior do que LevelDB)
  - <https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/TimelineServiceV2.html>

# YARN Timeline Server



The screenshot shows the Hadoop YARN Timeline Server web interface. The browser address bar displays the URL: `ec2-54-206-113-142.ap-southeast-2.compute.amazonaws.com:8188/applicationhistory`. The page title is "All Applications". The Hadoop logo is visible in the top left. A sidebar on the left shows "Application History" with filters for "FINISHED", "FAILED", and "KILLED". The main content area displays a table of applications with columns: ID, User, Name, Application Type, Queue, StartTime, FinishTime, State, FinalStatus, Progress, and Tracking UI. The table shows 10 applications, all of which are "FINISHED" and "SUCCEEDED".

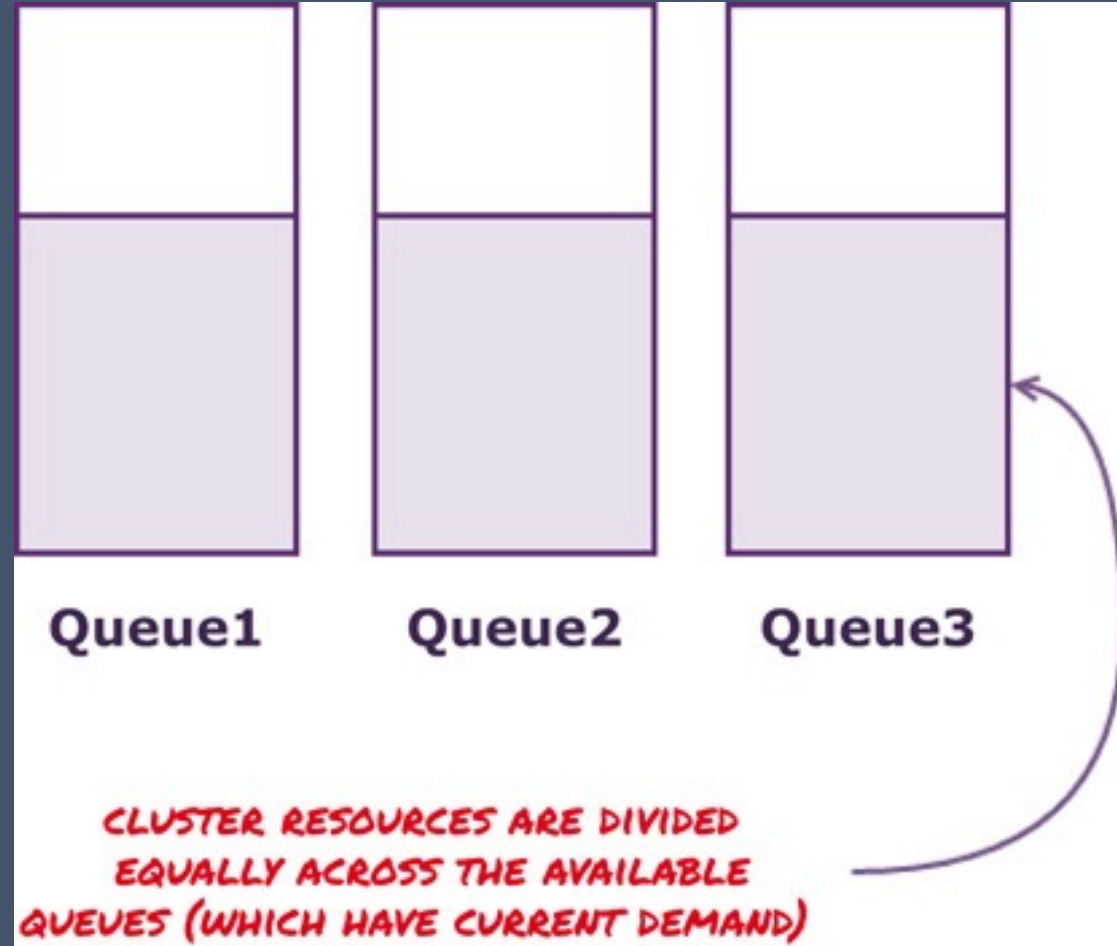
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1475830215563_0024	ec2-user	org.apache.spark.examples.SparkPi	SPARK	default	Sun Oct 9 18:01:40 +1100 2016	Sun Oct 9 18:01:59 +1100 2016	KILLED	UNDEFINED		<a href="#">History</a>
application_1475830215563_0023	ec2-user	Spark Pi	SPARK	default	Sun Oct 9 17:50:47 +1100 2016	Sun Oct 9 17:51:00 +1100 2016	FINISHED	SUCCEEDED		<a href="#">History</a>
application_1475830215563_0022	ec2-user	Spark Pi	SPARK	default	Sun Oct 9 17:31:51 +1100 2016	Sun Oct 9 17:32:03 +1100 2016	FINISHED	SUCCEEDED		<a href="#">History</a>
application_1475830215563_0021	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:44:12 +1100 2016	Sat Oct 8 16:44:32 +1100 2016	FINISHED	SUCCEEDED		<a href="#">History</a>
application_1475830215563_0020	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:44:06 +1100 2016	Sat Oct 8 16:44:23 +1100 2016	FINISHED	SUCCEEDED		<a href="#">History</a>
application_1475830215563_0019	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:42:45 +1100 2016	Sat Oct 8 16:43:00 +1100 2016	FINISHED	SUCCEEDED		<a href="#">History</a>
application_1475830215563_0018	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:42:26 +1100 2016	Sat Oct 8 16:42:43 +1100 2016	FINISHED	SUCCEEDED		<a href="#">History</a>
application_1475830215563_0017	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:42:17 +1100 2016	Sat Oct 8 16:42:35 +1100 2016	FINISHED	SUCCEEDED		<a href="#">History</a>
application_1475830215563_0016	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:41:54 +1100 2016	Sat Oct 8 16:42:10 +1100 2016	FINISHED	SUCCEEDED		<a href="#">History</a>
application_1475830215563_0015	ec2-user	QuasiMonteCarlo	MAPREDUCE	default	Sat Oct 8 16:41:40 +1100 2016	Sat Oct 8 16:41:56 +1100 2016	FINISHED	SUCCEEDED		<a href="#">History</a>

# Escalonamento no YARN

# Escalonamento de aplicações no YARN

- O escalonamento de aplicações permite que várias aplicações sejam executadas simultaneamente, compartilhando recursos de memória e computação distribuída do cluster
- **Políticas de escalonamento**
  - ***FIFOScheduler (padrão)*** - aloca recursos de acordo com a ordem de chegada
    - Não permite que aplicativos de longa execução coexistam com aplicativos de curta execução ou cumpram os SLAs de aplicativos
  - ***FairScheduler*** - distribui recursos do cluster de forma igual ou justa entre *filas* definidas e aplicações
  - ***CapacityScheduler*** - limita o uso de recursos com base em usuários e filas (objetivo semelhante ao FairScheduler - garantir justiça e estabilidade do cluster)

# Fair Scheduler






# Fair Scheduler

- Ativado no yarn-site.xml no RM e NM

```
<configuration>
...
<!--the class to use as the resource scheduler. -->
<property>
<name>yarn.resourcemanager.scheduler.class</name>
<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.Fair
Scheduler</value>
</property>
<!--path to allocation file describing queues and their properties-->
<property>
<name>yarn.scheduler.fair.allocation.file</name>
<value>fair-scheduler.xml</value>
</property>
...
</configuration>
```



Contém a configuração do FairScheduler



# Fair Scheduler

- As filas podem ser definidas com **diferentes acordos de nível de serviço (SLAs)**
- Permite especificar uma configuração de fila padrão e políticas de mapeamento
- Permite atribuir recursos mínimos garantidos para diferentes filas, bem como filas de ponderação (weighting queues)
- Os usuários também podem ter políticas específicas aplicadas
- FairScheduler também implementa o conceito de ***preempção***
  - Os containers alocados para filas de aplicações de baixa prioridade podem ser efetivamente substituídos por aplicações de prioridade mais alta (em execução nas filas de produção)

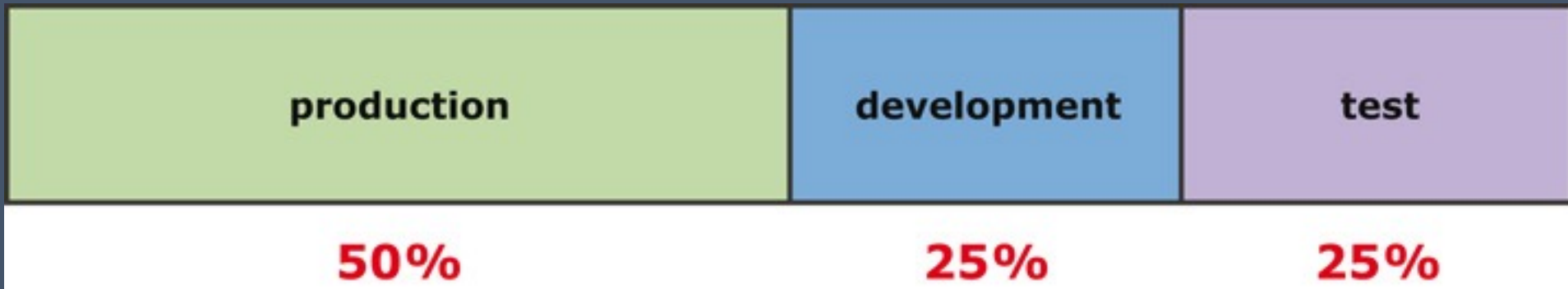
# fairscheduler.xml

```
<?xml version="1.0"?>
<allocations>
  <queue name="Queue1">
    <minResources>10000 mb,0vcores</minResources>
    <maxResources>90000 mb,0vcores</maxResources>
    <maxRunningApps>50</maxRunningApps>
    <maxAMShare>0.1</maxAMShare>
    <weight>2.0</weight>
    <schedulingPolicy>fair</schedulingPolicy>
  </queue>
  <queue name="Queue2">
    <minResources>5000 mb,0vcores</minResources>
    <maxResources>40000 mb,0vcores</maxResources>
    <maxRunningApps>100</maxRunningApps>
    <maxAMShare>0.1</maxAMShare>
    <weight>1.0</weight>
    <schedulingPolicy>fair</schedulingPolicy>
  </queue>
</allocations>
</xml>
```

Mais detalhes: <https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/FairScheduler.html>

# Capacity Scheduler

- CapacityScheduler é um escalonador muito mais simples do que o FairScheduler
- Basta definir as filas, incluindo uma fila padrão, e atribuir uma porcentagem dos recursos de cluster disponíveis à fila
- Limites rígidos em vcore e memória alocada para uma fila também podem ser definidos



# Capacity Scheduler

- Configurando o Capacity Scheduler no yarn-site.xml

```
<configuration>
...
<!--the class to use as the resource scheduler. -->
<property>
<name>yarn.resourcemanager.scheduler.class</name>
<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler
.capacity.CapacityScheduler</value>
</property>
...
</configuration>
```

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>prod,dev,default</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.prod.capacity</name>
    <value>20</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.capacity</name>
    <value>40</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.default.capacity</name>
    <value>40</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.maximum-capacity</name>
    <value>75</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.queue-mappings</name>
    <value>u:devuser:dev,u:produser:prod</value>
  </property>
</configuration>
```

Arquivo de configuração:  
capacity-scheduler.xml

# Capacity Scheduler

- Mudanças no arquivo de configuração requerem a atualização das informações no ResourceManager

```
$ yarn rmadmin -refreshQueues
```

# Fila do escalonador e informações de capacidade

- Interface Web do ResourceManager (página do escalonador)

The screenshot displays the Hadoop Resource Manager Web Interface, specifically the scheduler page. The browser address bar shows the URL: `ec2-54-206-113-142.ap-southeast-2.compute.amazonaws.com:8088/cluster/scheduler`. The page title is "NEW,NEW\_SAVING,SUBMITTED,ACCEPTED,RUNNING Applications".

**Cluster Metrics**

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
28	0	2	26	24	36.56 GB	56.25 GB	0 B	24	80	0	5	0	0	0	0	0

**Scheduler Metrics**

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:32, vCores:1>	<memory:11520, vCores:16>

**Application Queues**

**Legend:** Capacity (grey), Used (green), Used (over capacity) (orange), Max Capacity (light grey)

- root: 65.0% used
- + default: 0.0% used
- dev: 106.3% used
- prod: 112.5% used

**Application List**

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1475830215563_0028	devuser	QuasiMonteCarlo	MAPREDUCE	dev	Mon Oct 10 20:57:20 +1100	N/A	RUNNING	UNDEFINED	<div></div>	ApplicationMaster	0

**Ciências de Dados e Inteligência Artificial**

# Gerência de Infraestrutura para Big Data

YARN

Prof. Tiago Ferreto

*tiago.ferreto@pucrs.br*



**Ciências de Dados e Inteligência Artificial**

# Gerência de Infraestrutura para Big Data

## HDFS

Prof. Tiago Ferreto

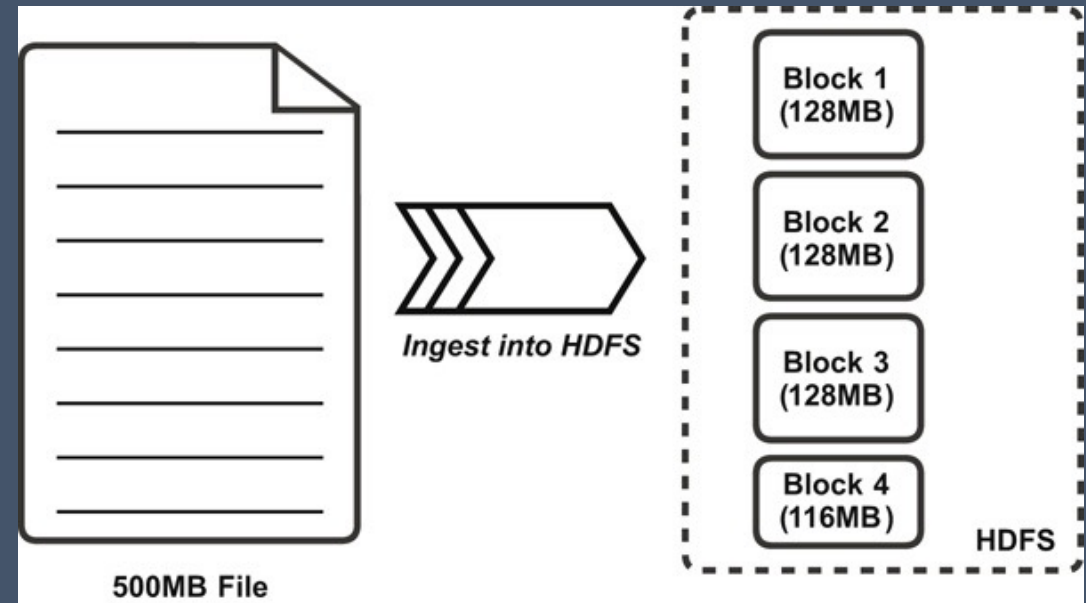
*tiago.ferreto@pucrs.br*

# HDFS

- HDFS é a principal fonte de dados de entrada e saída do Hadoop para operações de processamento de dados
  - Outros sistemas de arquivos também são suportados
- Inspirado no artigo GoogleFS (2003)
  - Foco no suporte aos requisitos de armazenamento dos mecanismos de busca
- Características
  - Escalável (economicamente)
  - Tolerante a falhas
  - Usa hardware comum
  - Suporta alta concorrência
  - Favorece a alta demanda por largura de banda em relação ao acesso aleatório de baixa latência

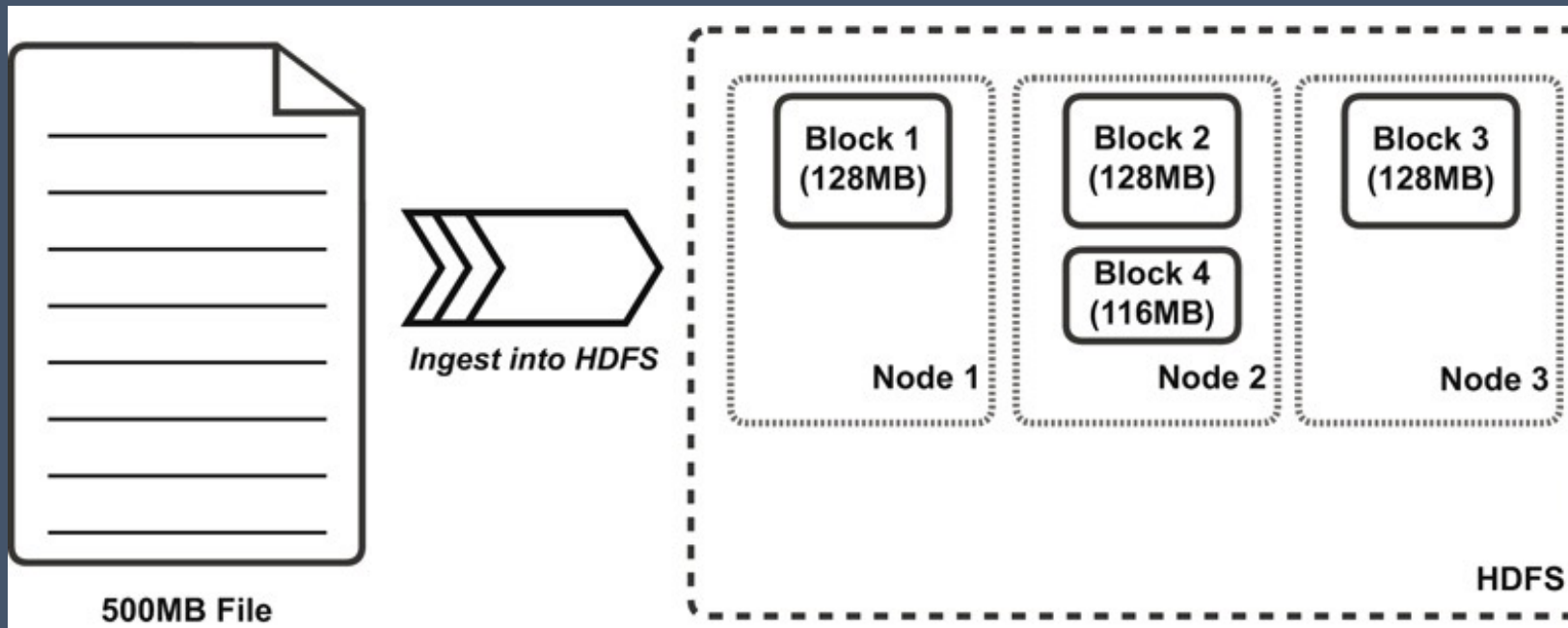
# Arquivos, blocos e replicação

- HDFS é um sistema **de arquivos virtual**
  - Ele aparece como um único sistema, mas seus dados estão localizados em vários locais diferentes
  - Opera sobre sistemas de arquivos nativos (por exemplo, ext3, ext4, xfs)
- Os dados armazenados no HDFS são imutáveis - **não podem** ser atualizados após serem escritos
  - Sistema de arquivos **WORM** (escrever uma vez, ler várias)
- Os arquivos são **divididos em blocos** quando ingeridos no HDFS
  - Tamanho padrão = 128 MB (configurável)



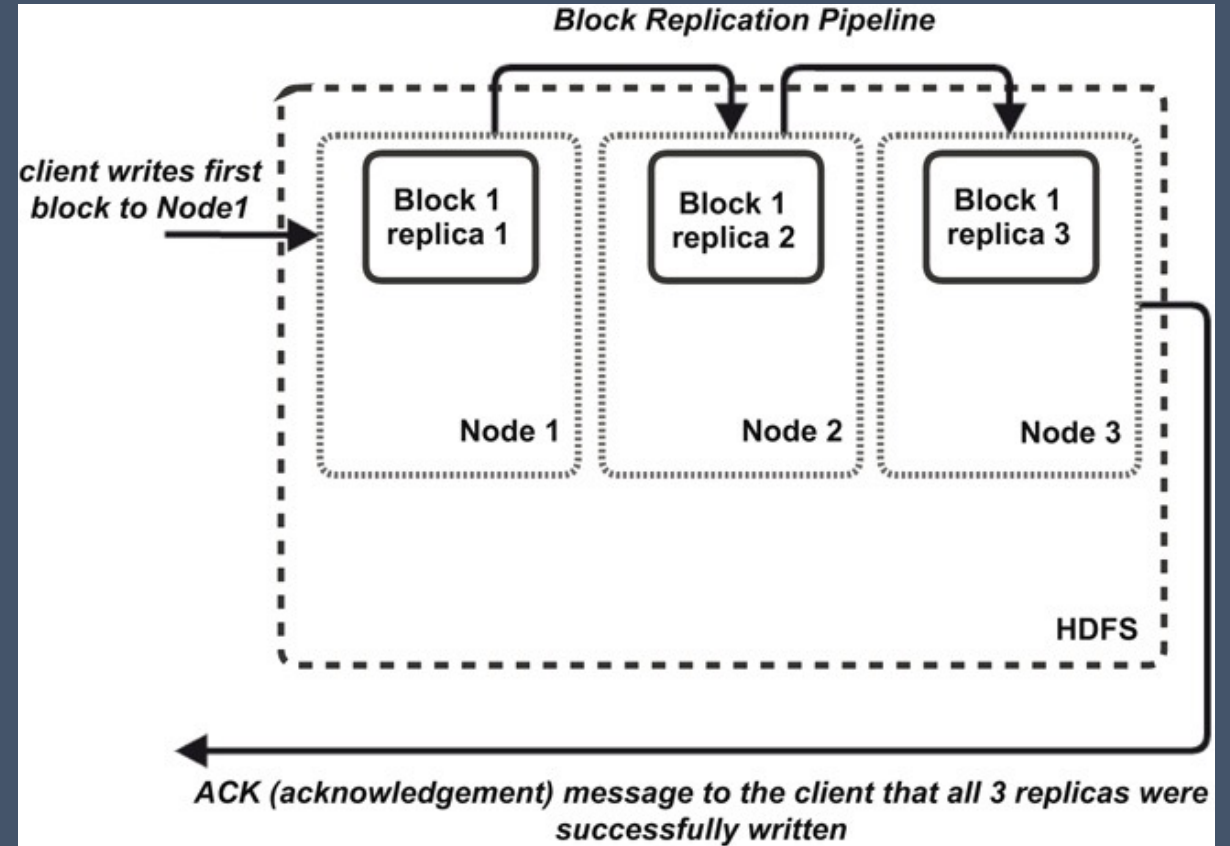
# Arquivos, blocos e replicação

- Os blocos são **distribuídos**
  - Os blocos são distribuídos entre os nós escravos no cluster após a ingestão (considerando um cluster de vários nós)
  - Isso permite: nenhum compartilhamento e processamento paralelo de dados



# Arquivos, blocos e replicação

- Os blocos são **replicados**
  - A replicação ocorre de acordo com um fator de replicação pré-configurado
    - Normalmente definido como 3 (em um ambiente de cluster totalmente distribuído com 3 ou mais nós)
    - Em um cluster Hadoop pseudo-distribuído, esse valor é definido como 1 (há apenas um DataNode)
- A replicação acontece na ingestão
- Objetivos para replicação
  - Aumenta as oportunidades de localização de dados
  - Fornece tolerância a falhas



# Recuperação de falhas de DataNode ou bloco

- Cada objeto no HDFS tem um fator de replicação
- NameNode obtém inventários de bloco regulares (relatórios de bloco) de cada DataNode no cluster
  - O intervalo padrão é de 6 horas (dfs.blockreport.intervalMsec em hdfs-site.xml)
  - NameNode verifica quais blocos estão corrompidos ou sub-replicados (possivelmente devido a uma falha de DataNode)
- NameNode também recebe heartbeats regulares para verificar a saúde do DataNode
  - O intervalo de tempo padrão é de 3 segundos (dfs.heartbeat.interval em hdfs-site.xml)
  - NameNode aguarda até 10 heartbeats perdidos (30 segundos) antes de assumir que um DataNode foi perdido
- Quando o NameNode detecta que um bloco não tem o número certo de réplicas, ele instrui um DataNode (com uma réplica válida) a replicar esse bloco para outro nó

# NameNode

- Processo de nó mestre HDFS - coordena o sistema de arquivos distribuído
- Gerencia os metadados do sistema de arquivos
  - Contém todos **os objetos de diretórios e arquivos** com suas propriedades e atributos (ACLs - Listas de Controle de Acesso) - define usuários ou grupos com acesso aos objetos
  - **Armazenado na memória** - consultas do cliente com baixa latência
  - **Usa funções de snapshot e journaling** para garantir durabilidade e consistência em caso de erros
  - Inclui as **localizações dos blocos** que compõem os arquivos em HDFS
    - Apenas representação da relação entre arquivos e blocos no HDFS
- Consultas de clientes (via CLI, MapReduce, Spark ou outro aplicativo)
  - Os clientes interagem com o NameNode para obter informações dos blocos e acessar diretamente os DataNodes → NameNode não é usado na leitura / gravação de dados (Evita gargalos)

# DataNodes

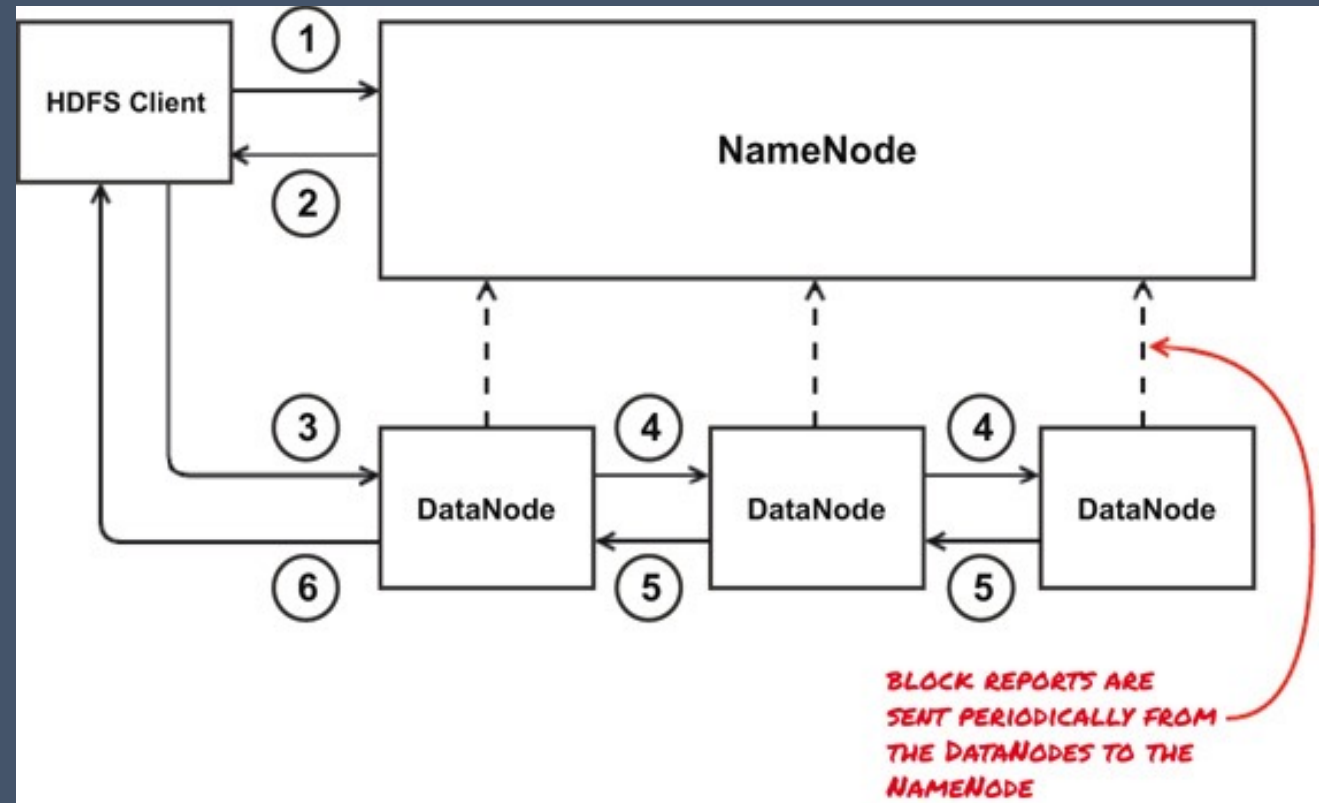
- Nós de cluster nos quais os blocos HDFS são armazenados e gerenciados
- Responsabilidades
  - Participa do **pipeline de replicação de blocos**
  - **Gerencia volumes e armazenamento locais**
  - Fornece **relatórios de blocos** para o NameNode
    - Mensagens regulares com um inventário dos blocos armazenados no DataNode
- **Os checksums** são calculadas na ingestão no HDFS e são mantidas com os blocos
  - DataNode recalcula e compara os checksums periodicamente e relata incompatibilidades com o NameNode
    - Como o sistema de arquivos é imutável, os checksums nunca mudam!
- Os blocos são armazenados em volumes locais nos DataNodes
- DataNodes armazenam apenas blocos HDFS. Não há informações sobre a estrutura dos arquivos ou diretórios → Não é possível reconstruir o sistema de arquivos se os metadados do NameNode forem perdidos!



# HDFS — ESCRITA E LEITURA DE ARQUIVOS

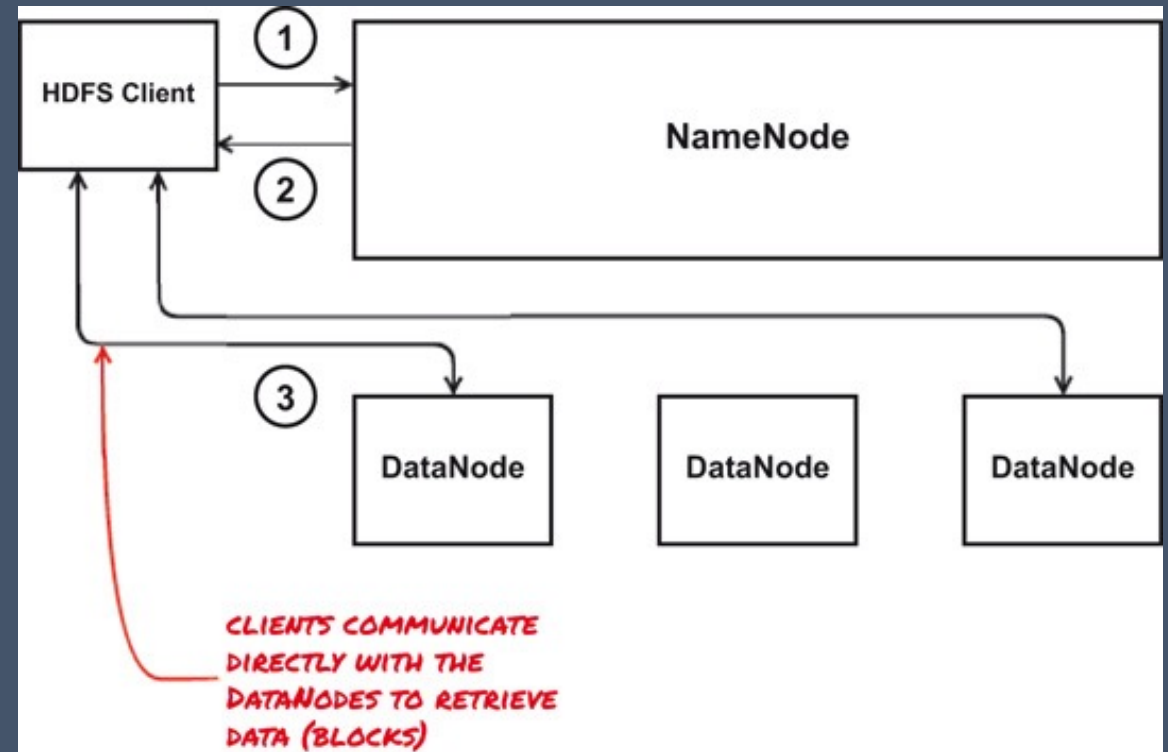
# Gravando arquivos em HDFS

1. HDFS Client solicita a gravação de um bloco de arquivo.
2. NameNode responde ao cliente com o DataNode no qual gravar o bloco.
3. O cliente solicita a gravação do bloco no DataNode especificado.
4. DataNode abre um pipeline de replicação de bloco com outro DataNode no cluster e esse processo continua até que todas as réplicas configuradas sejam gravadas.
5. Uma confirmação de gravação é enviada de volta por meio do pipeline de replicação.
6. O cliente é informado de que a operação de gravação foi bem-sucedida.



# Lendo arquivos do HDFS

1. HDFS Client solicita a leitura de um arquivo
2. NameNode responde à solicitação com uma lista de DataNodes contendo os blocos que compõem o arquivo (todas as réplicas)
  1. Se um DataNode não estiver disponível ou um bloco estiver corrompido, o Cliente pode usar outra réplica
3. O cliente se comunica diretamente com DataNodes para recuperar blocos para o arquivo.



# HDFS – METADADOS E ESTRUTURA INTERNA

# Metadados do NameNode

- Componente mais crítico no HDFS
  - Contém: link entre blocos e DataNodes, e arquivos e estrutura de diretórios e atributos
- Reside na memória para pesquisa rápida pelos clientes
- Exemplo (representação conceitual)

object	block_id	seq	locations	ACL	Checksum
/data/file.txt	blk_00123	1	[node1,node2,node3]	-rwxrwxrwx	8743b52063 ..
/data/file.txt	blk_00124	2	[node2,node3,node4]	-rwxrwxrwx	cd84097a65 ..
/data/file.txt	blk_00125	3	[node2,node4,node5]	-rwxrwxrwx	d1633f5c74 ..

# Quantidade e tamanho dos arquivos

- Recomendação
  - HDFS prefere poucos arquivos com maior tamanho
    - Cada objeto no HDFS (arquivo ou diretório) consome aproximadamente 150-200 bytes de memória no NameNode → menos arquivos maiores são preferidos em relação a muitos arquivos menores
- Considerando a ingestão de 10 GB no HDFS com tamanho de bloco de 128 MB

Files	Name Entries	Block Metadata	Total Objects
10 × 1GB files	10	80	90
10,000 × 1MB files	10,000	10,000	20,000

- Os arquivos podem ser concatenados na ingestão

# Listas de controle de acesso e permissões no HDFS

- Objetos HDFS têm ACLs associados
  - Define o proprietário do objeto e as permissões
- Usa uma máscara de permissões de acesso (Unix)
  - Bits com permissão para: 4-leitura (r), 2-gravação (w) e 1-execução (x)
  - Execução é usado apenas para diretórios (HDFS não tem executáveis)



- Exemplo

- `$ hdfs dfs -chown ferreto /data/books`
- `$ hdfs dfs -chmod 777 /data/books`



- Cuidado: a segurança no HDFS é considerada fraca!
  - É aconselhável usar métodos de segurança adicionais (por exemplo, Kerberos) em clusters de produção

# Estruturas e consistência no disco

- A representação dos metadados do NameNode em disco consiste em dois componentes:
  - **Arquivo fsimage**
    - um instantâneo point-in-time dos metadados sem os locais de bloco específicos
    - normalmente só é gravado no final da recuperação ou pelo SecondaryNameNode
  - **Arquivo edits**
    - contém atualizações para os metadados
    - atualizado para cada mudança no sistema de arquivos - novos dados, dados excluídos, permissões modificadas (semelhante a um log de transações em um banco de dados tradicional)



# Recuperação de NameNode

- Processo de recuperação do NameNode - acontece na inicialização do NameNode
  1. O snapshot do fsimage está montado
  2. Edits (atualizações) são aplicadas em sequência
  3. Um novo fsimage é criado para o próximo processo de recuperação
  4. Novos arquivos edit são criados para capturar novas alterações pós-recuperação
- Após o processo de recuperação, DataNodes começam a enviar seus relatórios de blocos para o NameNode
  - NameNode começa a associar locais de bloco para as réplicas na representação dos metadados em memória
- Os locais dos blocos podem mudar devido às operações de replicação ou rebalanceamento → persistem apenas na memória e não são gravados no arquivo fsimage ou edits

# Modo de segurança (SafeMode)

- Em modo de segurança o HDFS permite apenas operações de leitura
  - Usado durante os processos de inicialização e recuperação do NameNode

```
[ec2-user@ip-172-31-15-54 hadoop]$ sudo -u hdfs bin/hadoop fs -chmod 777 /tmp
chmod: changing permissions of '/tmp': Cannot set permission for /tmp. Name node
is in safe mode.
[ec2-user@ip-172-31-15-54 hadoop]$
```

# SecondaryNameNode

- Processo opcional localizado em um host diferente do NameNode
- **Verifica** o sistema de arquivos periodicamente
  - Executa uma operação de recuperação em nome do NameNode primário (executa a mesma sequência de operações que o NameNode primário)
  - Obtém o arquivo **fsimage**, aplica atualizações do arquivo edits e cria um novo arquivo **fsimage**
  - O novo arquivo **fsimage** é substituído no NameNode principal
    - Encurta as operações de recuperação e reduz o espaço em disco consumido pelos arquivos edits
- SecondaryNameNode não é uma solução HA (alta disponibilidade)!
  - Ele fornece apenas um local de armazenamento alternativo para a representação em disco dos metadados do NameNode em caso de falha do NameNode primário
  - HA é habilitado usando um Standby NameNode

# HDFS - INTERFACES DE ACESSO

# Interagindo com HDFS

- Principais interfaces de acesso
  - Shell do sistema de arquivos (hadoop fs ou hdfs dfs)
  - API Hadoop Filesystem Java
  - Interfaces de proxy RESTful - HttpFS e WebHDFS
- HDFS é baseado no padrão POSIX
  - Usa convenções POSIX encontradas em Unix / Linux para representações de arquivos e diretórios
- O shell HDFS usa verbos semelhantes aos comandos de FTP (put, get, etc)
- O HDFS não tem conceito de diretório atual (não existe o comando cd) → todo comando começa a partir de um caminho relativo começando no diretório inicial do usuário no HDFS (/ user / <username>)

# Shell HDFS - Upload (ou ingestão de um arquivo)

- Exemplo:

- Carregando um arquivo local chamado warandpeace.txt em um diretório existente no HDFS chamado / data / books

- `$ hadoop fs -put warandpeace.txt /data/books/`

- Comandos sinônimos

- `$ hadoop fs -copyFromLocal warandpeace.txt /data/books`

- `$ hdfs dfs -put warandpeace.txt /data/books`

# Shell HDFS - Baixando um arquivo

- Muitos aplicativos não podem interagir diretamente com o HDFS → é necessário recuperar o arquivo do HDFS
- Exemplo:
  - Recupera um arquivo chamado report.csv de / data / reports no HDFS e coloca no diretório atual do usuário
  - `$ hadoop fs -get /data/reports/report.csv`

# Shell HDFS – Listagem do conteúdo do diretório

- Exemplo:
  - Para listar o conteúdo de / data / reports
  - `$ hadoop fs -ls /data/reports`



# Shell HDFS – Excluindo Objetos

- Exemplo:
  - Para excluir report.csv de / data / reports no HDFS
  - `$ hadoop fs -rm /data/reports/report.csv`
  - Para excluir todo o diretório / data / reports
  - `$ hadoop fs -rm -r /data/reports`

# Shell HDFS

- Documentação com todos os comandos
  - <http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html>

# Pasta de lixo do HDFS

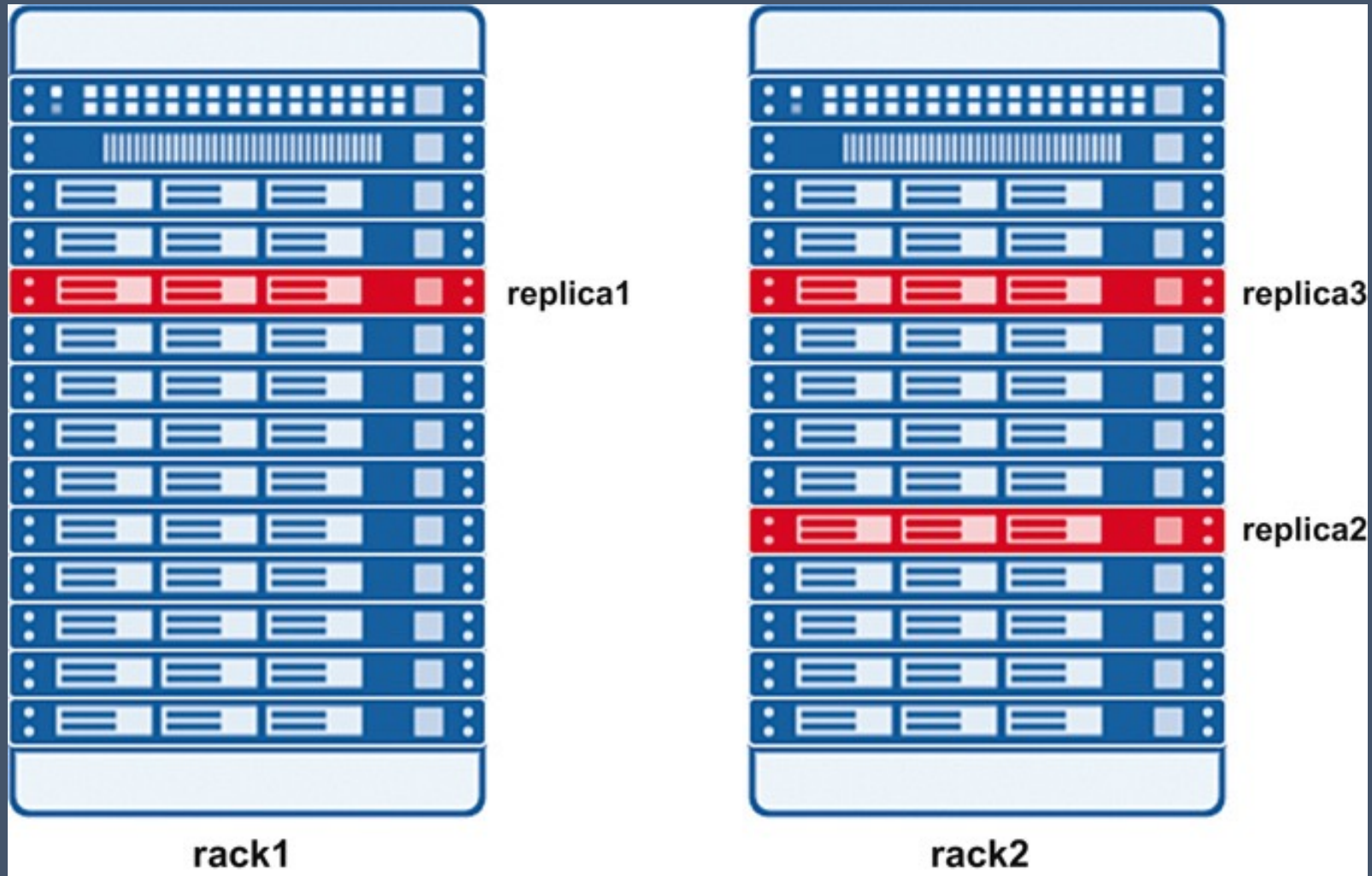
- HDFS tem o conceito de uma pasta de lixo ou lixeira
- Configurado por um parâmetro denominado `fs.trash.interval` (arquivo `hdfs-site.xml`)
  - Define a quantidade de tempo (em minutos) para manter um objeto excluído em um diretório de Lixeira oculto antes de ser removido permanentemente do sistema de arquivos
  - O padrão é 0 → todas as exclusões são imediatas e irreversíveis

# HDFS - TÓPICOS AVANÇADOS

# HDFS Rack Awareness

- Permite que o HDFS compreenda a topologia de cluster que pode incluir vários racks de servidores ou vários data centers e orquestre a colocação dos blocos adequadamente
- Os dados são dispersos em racks ou data centers para fornecer maior tolerância a falhas em caso de falha de rack, switch ou rede ou até mesmo interrupção do data center
- Estratégia de colocação de blocos (com rack awareness habilitado)
  1. A primeira réplica de um bloco é colocada em um nó do cluster
  2. A segunda réplica de um bloco é colocada em um nó que reside em um rack diferente da primeira réplica
  3. A terceira réplica, assumindo um fator de replicação padrão de 3, é colocada em um nó diferente no mesmo rack que a segunda réplica

# Exemplo

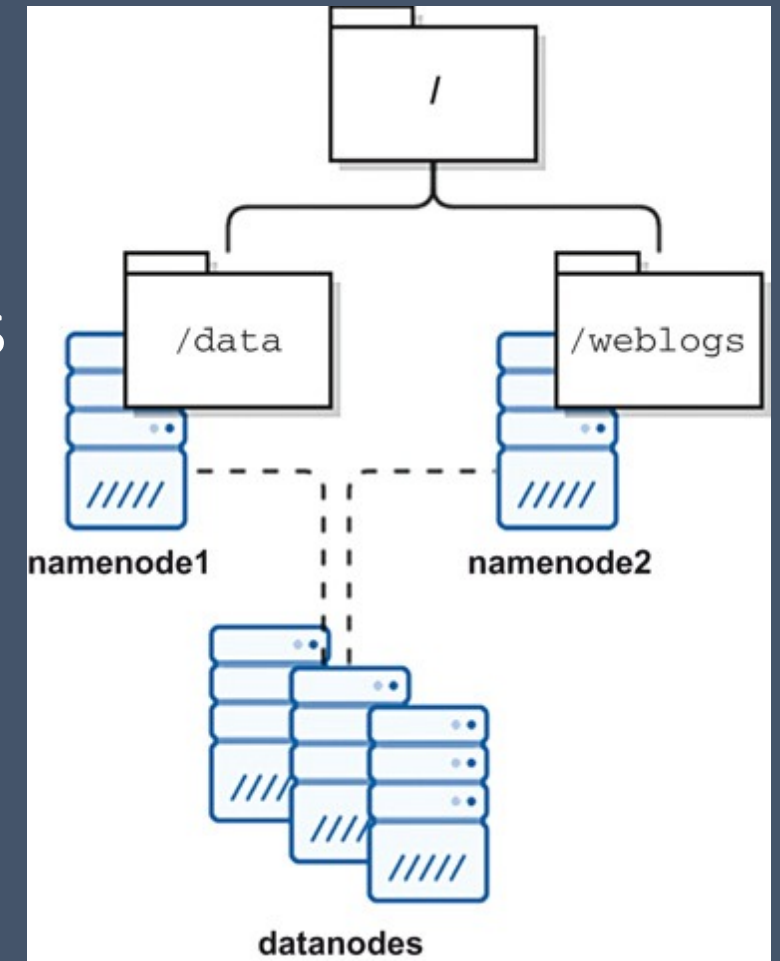


# HDFS com alta disponibilidade (HA)

- Objetivo: suportar falhas no NameNode (HW ou SW), atualizações ou alterações de configuração
- SecondaryNameNode - lida apenas com a redução do tempo de recuperação e fornece um local de armazenamento alternativo para os metadados do NameNode
  - NÃO é uma solução HA!
- HA em HDFS → 2 NameNodes - um NameNode ativo e um em espera
  - Se o NameNode ativo falhar, o Standby NameNode assume o controle do sistema de arquivos, gerencia as solicitações de leitura e gravação do cliente e atualizações de metadados
  - Standby NameNode também executa funções de checkpoint (fornecidas pelo SecondaryNameNode)
    - SecondaryNameNode torna-se desnecessário!

# Federação HDFS

- Permite que um cluster HDFS tenha muitos NameNodes ativos
  - Cada NN gerencia uma parte diferente do namespace do sistema de arquivos
  - Obrigatório quando a quantidade de metadados HDFS não cabe na memória de um único servidor
- NameNodes são independentes uns dos outros (funcionam de forma independente)
- DataNodes ainda armazenam blocos de todos os volumes de namespace
- Diferentes namespaces / NameNodes são especificados no arquivo core-site.xml





# Cache HDFS

- Permite o armazenamento de dados acessados com frequência em cache de memória off-heap nos DataNodes
  - Off-heap refere-se a objetos que não são gerenciados como parte de um heap Java (não é sujeita ao Garbage Collector do Java)
  - O acesso à memória geralmente será mais rápido do que o acesso ao disco
- O cache HDFS permite aos usuários:
  - Manter conjuntos de dados fixos em cache
  - Armazenar arquivos e diretórios específicos em cache
- Requer memória suficiente nos DataNodes

# Snapshots HDFS

- Criação de uma ou mais imagens point-in-time (instantâneos) de um diretório HDFS
- Inclui subdiretórios
- Pode ser usado como backup ou para fins de auditoria
- Para que um diretório seja snapshotted, ele deve ser criado como snapshottable

```
$ hdfs dfsadmin -allowSnapshot /data
```

- Cria um snapshot

```
$ hdfs dfs -createSnapshot /data snapshot_on_20161117
```

```
Created snapshot /data/.snapshot/snapshot_on_20161117
```

- A criação de um snapshot é instantânea (não há cópias de bloco)
- Snapshots podem ser comparados entre si

```
$ hdfs snapshotDiff /data snapshot_on_20161117 snapshot_on_20161118
```

```
Difference between snapshot snapshot_on_20161117 and snapshot  
snapshot_on_20161118 under directory /data:
```

```
M .
```

```
+ ./stop-word-list.csv
```

# HDFS Archiving

- Criação de arquivos compactados (semelhantes aos arquivos tar)
  - arquivos har → contêm metadados e arquivos de dados
- Ao contrário dos snapshots, o arquivamento cria réplicas físicas inteiras das partes desejadas do sistema de arquivos
- Criação de um archive Hadoop
  - `$ hadoop archive -archiveName data.har -p /data /backups`

**Ciências de Dados e Inteligência Artificial**

# Gerência de Infraestrutura para Big Data

## HDFS

Prof. Tiago Ferreto

*tiago.ferreto@pucrs.br*

**Ciências de Dados e Inteligência Artificial**

# Gerência de Infraestrutura para Big Data

Ingestão de dados

Prof. Tiago Ferreto

*tiago.ferreto@pucrs.br*

# Ingestão de Dados

- Como capturar dados produzidos de sistemas de origem em tempo real?
  - Exemplos: web logs, bancos de dados, sensores
- A interface padrão HDFS não é prática o suficiente para cenários mais complexos
- Ferramentas
  - Flume
  - Sqoop
  - Interfaces HDFS RESTful
    - WebHDFS
    - HttpFS

# Flume

- Projeto do ecossistema Hadoop
  - <https://flume.apache.org/>
- Desenvolvido originalmente pela Cloudera
- Objetivo: capturar, transformar e ingerir dados em HDFS usando um ou mais agentes
- Caso de uso típico: captura de arquivos de log ou weblogs de um servidor web e encaminhamento para HDFS à medida que são gerados



# Motivação do Flume

- Superar as desvantagens do comando put do HDFS e transferir "dados de streaming" dos geradores de dados para sistemas armazenamento centralizado (especialmente HDFS) com menos atraso
- Problemas com put
  - Permite a transferência de apenas um arquivo por vez
    - Os geradores de dados geram dados em uma taxa muito mais alta
  - Requer que os dados sejam empacotados e prontos para o upload
    - Os servidores da web geram dados continuamente

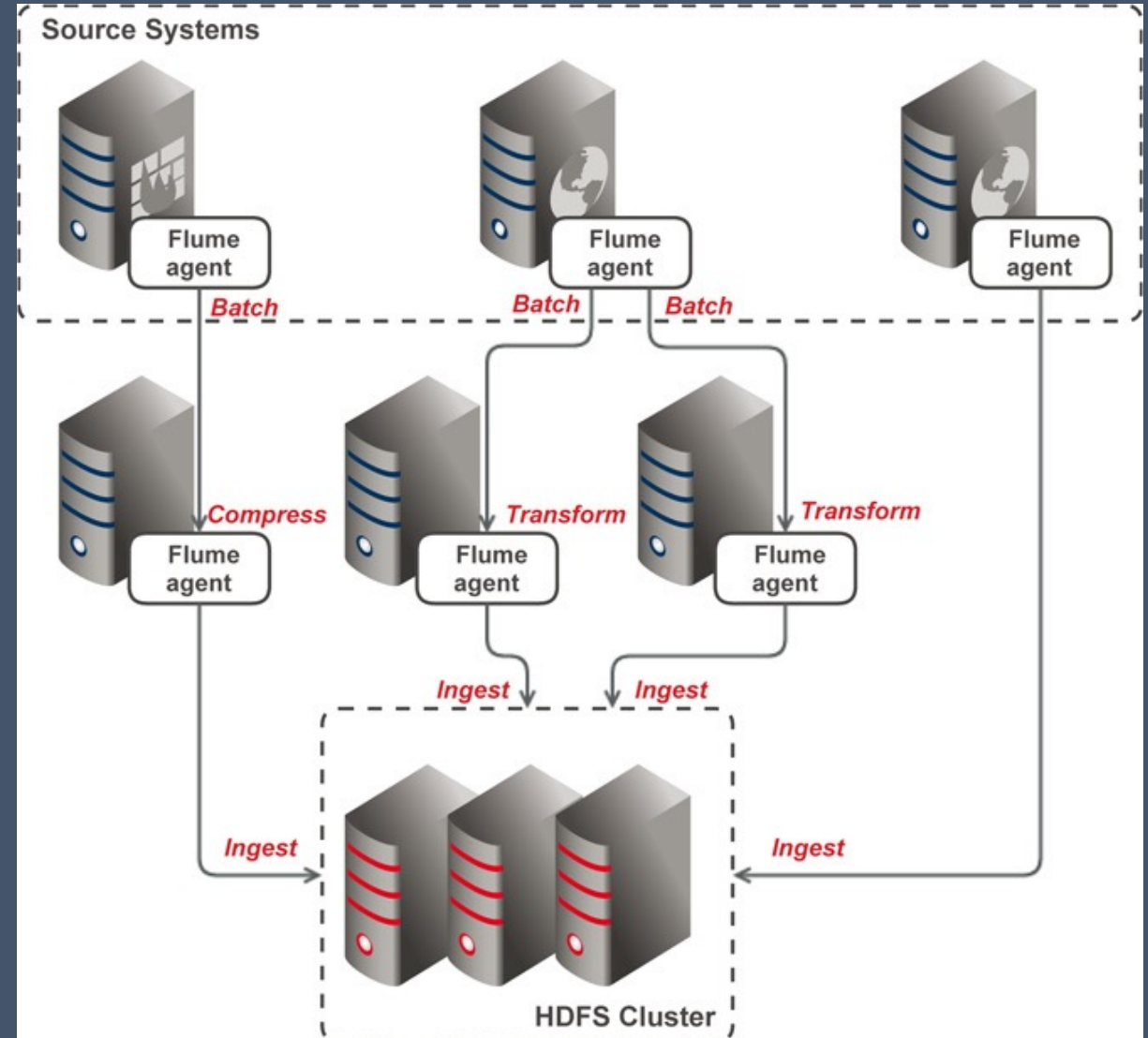


# Características do Flume

- Altamente confiável, tolerante a falhas, escalável, gerenciável e personalizável
- Fornece um processo eficiente para ingerir dados de log de vários servidores em um armazenamento centralizado (ex. HDFS)
- Permite a importação de grandes volumes de dados de eventos produzidos por sites de redes sociais como Facebook e Twitter e sites de comércio eletrônico como Amazon
- Suporta um grande conjunto de fontes e tipos de destino
- Suporta fluxos multi-hop, fluxos fan-in fan-out, roteamento contextual, etc
- Pode ser escalado horizontalmente

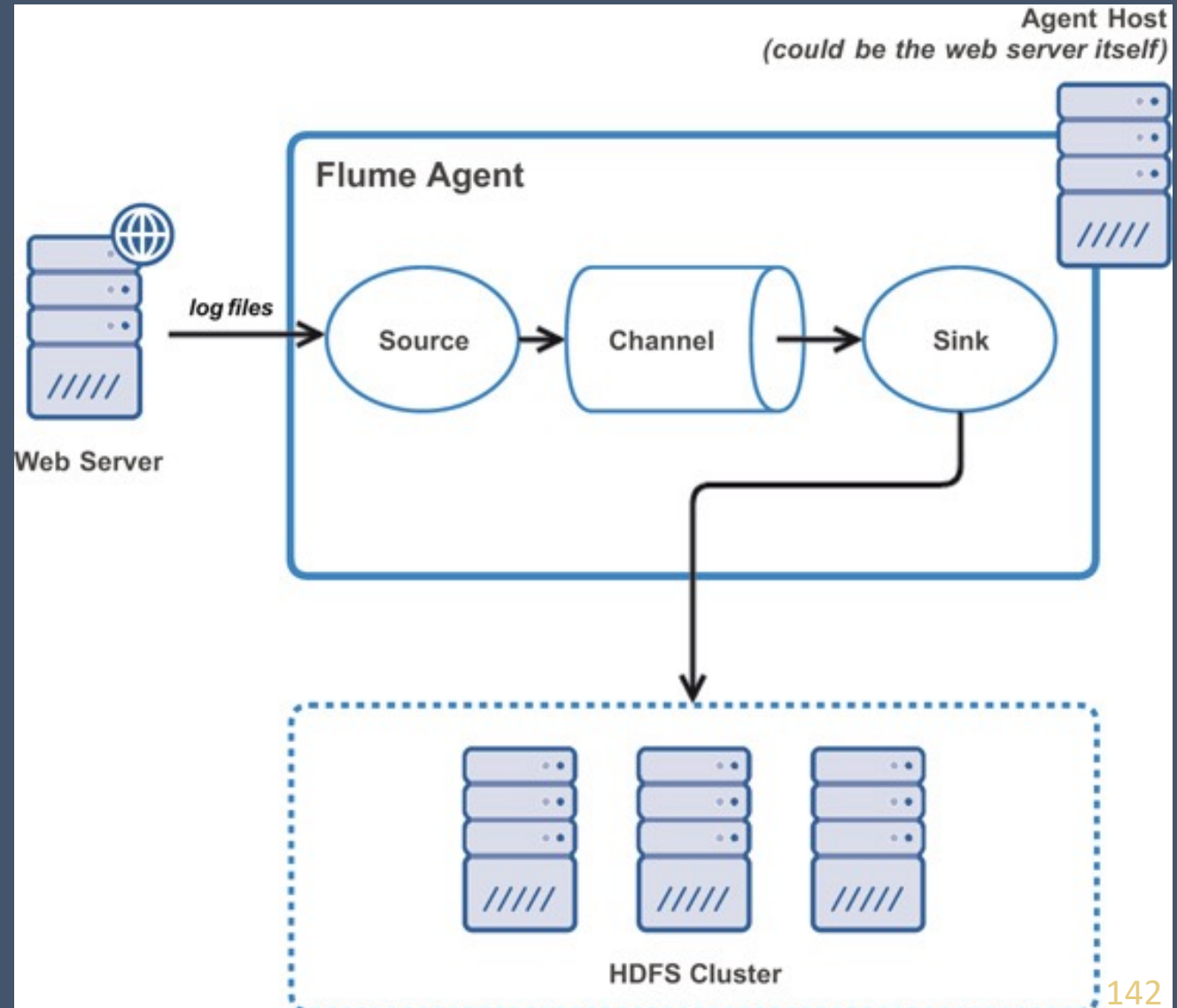
# Arquitetura do Flume

- Implementado usando um ou mais agentes
- Os agentes conectam uma fonte de dados ao HDFS ou outro agente
- Os agentes podem ser encadeados ou usados em paralelo (escalabilidade horizontal ou tolerância a falhas)
- Os agentes podem realizar operações de dados “em voo” (por exemplo, compressão, criptografia, lote de eventos, etc)



# Arquitetura do Agente Flume

- Os agentes do Flume contêm um *sink*, um *source* e um channel
- Implementado como um processo daemon independente (JVM)
- O agente recebe dados (eventos) de clientes ou outros agentes e os encaminha para seu próximo destino (sink ou agente)



# Evento Flume

- Um evento é a unidade básica dos dados transportados dentro do Flume
- Ele contém um conjunto de bytes que deve ser transportado da origem para o destino, acompanhada de cabeçalhos opcionais



# Source

- Indica de onde os dados devem ser recebidos
- Exemplos
  - HTTP - usado para consumir dados de serviços RESTful usando métodos POST e GET
  - Syslog - protocolo de logs para capturar eventos do sistema
  - JMS - Java Message Service
  - Kafka - popular plataforma de mensagens de código aberto
  - Avro - framework de serialização de dados multiplataforma e de código aberto para Hadoop
  - Twitter - fonte do Flume que se conecta à API de streaming do Twitter para baixar tweets continuamente

# Channel

- Fila entre a fonte (source) e o coletor (sink) do agente
- Flume implementa uma arquitetura transacional para confiabilidade (suporta operações de *rollback* e *recovery*)
- Configurações possíveis: **in-memory** ou **durável**
- Canais duráveis usam armazenamento persistente (disco) para manter o estado (integridade transacional)
- Exemplos de canais duráveis
  - Canal de arquivo
  - Canal JDBC
  - Canal Kafka

# Sink

- Especifica para onde enviar dados (normalmente HDFS, mas também pode ser outro agente ou outro sistema de arquivos, como S3)
- Exemplos
  - HDFS - sink mais comum usado para ingerir dados em HDFS
  - Hive - insere dados em HDFS enquanto atualiza as partições Hive e o metastore Hive
  - Hbase - armazenamento de dados NoSQL construído em HDFS
  - Kafka - popular plataforma de mensagens de código aberto
  - Solr - plataforma de pesquisa de código aberto

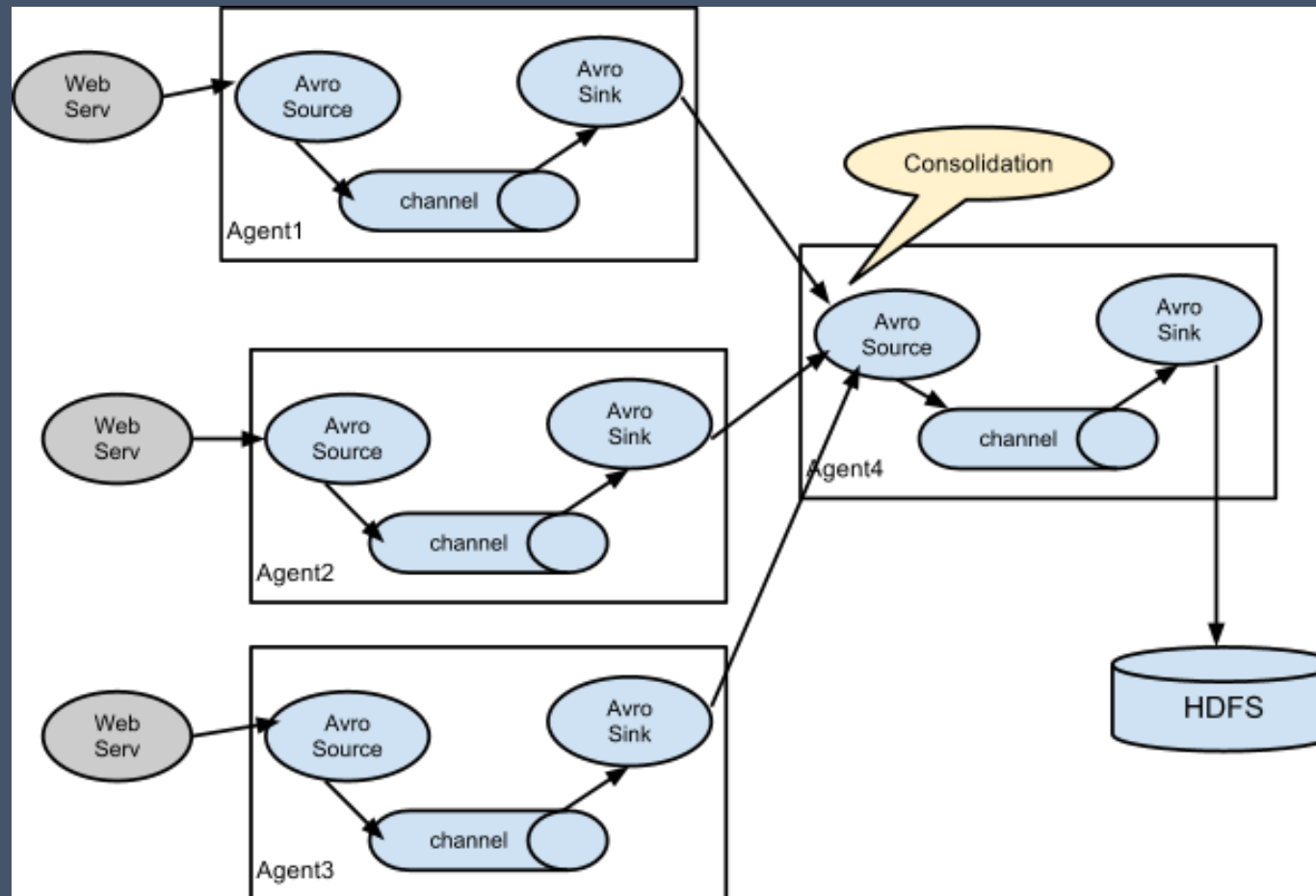
# Componentes Adicionais

- Interceptadores (Interceptors)
  - Alteram / inspecionam / descartam eventos Flume que são transferidos entre o source e o channel
  - Exemplos
    - Interceptador de timestamp: insere nos cabeçalhos do evento o tempo em milissegundos em que o evento é processado
    - Host Interceptor: insere o nome do host ou endereço IP do host em que este agente está sendo executado
- Seletores de canal (Channel selectors)
  - Determina qual canal deve ser escolhido para transferir os dados no caso de vários canais
  - Tipos: replicação, multiplexação ou customizada
- Sink Processors
  - Invoca um coletor específico do grupo selecionado de coletores
  - Permite a criação de caminhos de failover para coletores (sinks) ou balanceamento de carga dos eventos entre múltiplos sinks a partir de um canal
  - Tipos: padrão (single sink), failover, balanceamento de carga



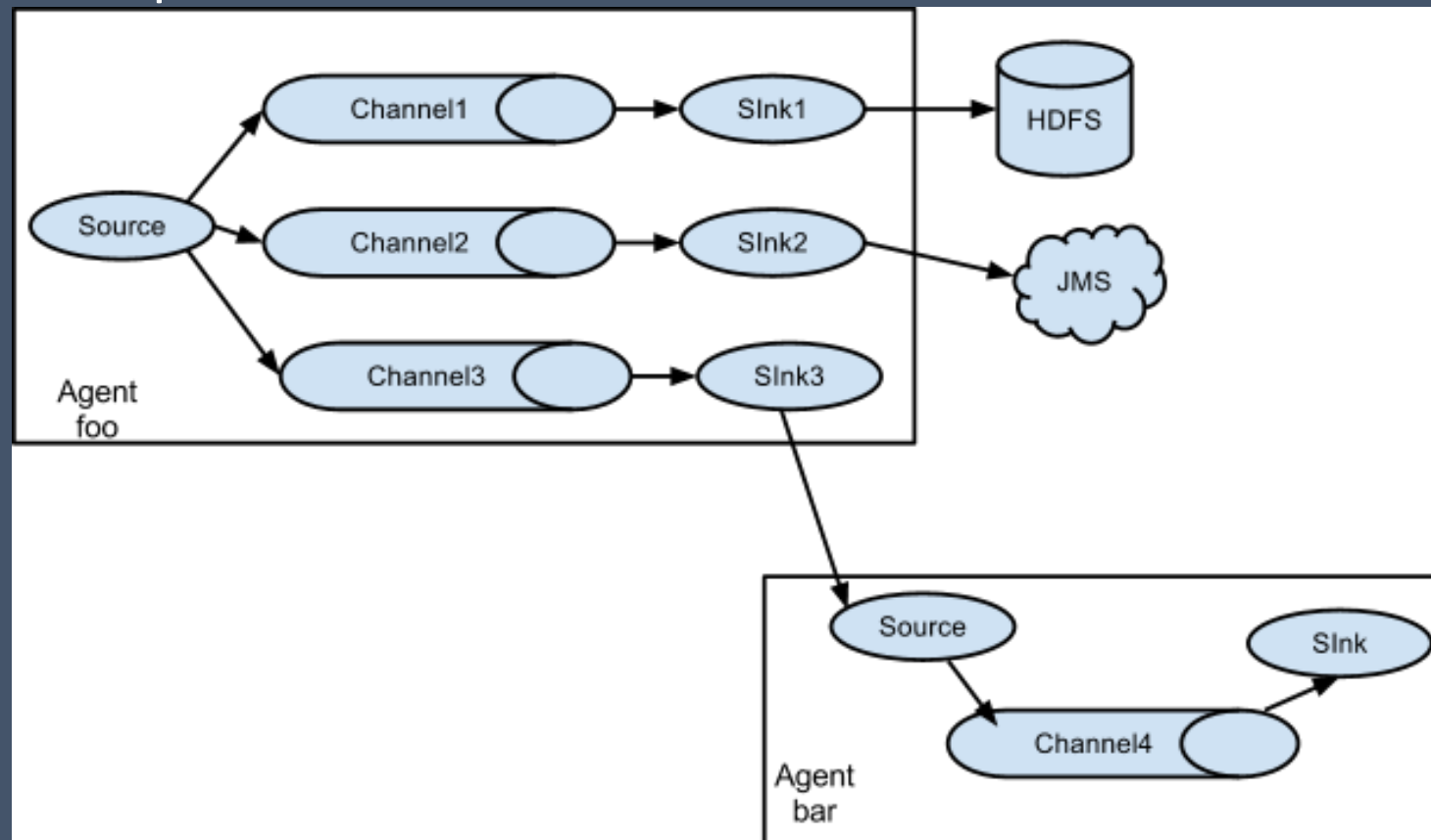
# Cenários do Flume – coleta de Logs

- Os clientes de produção de log enviam dados para alguns agentes consumidores que estão conectados ao subsistema de armazenamento



# Cenários do Flume - Multiplexação

- Envio de eventos para diferentes canais e coletores
  - Replica o evento para todos os canais ou multiplexa com base no atributo de um evento específico



# Flume – Configuração básica

```
# list the sources, sinks and channels for the agent
<Agent>.sources = <Source>
<Agent>.sinks = <Sink>
<Agent>.channels = <Channel1> <Channel2>

# set channel for source
<Agent>.sources.<Source>.channels = <Channel1> <Channel2> ...

# set channel for sink
<Agent>.sinks.<Sink>.channel = <Channel1>
```

# Exemplo do Flume

```
agent1.sources = source1
agent1.sources.source1.type = exec
agent1.sources.source1.command = tail -F /tmp/events

agent1.channels = channel1
agent1.channels.channel1.type = memory

agent1.sinks = sink1
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /flume/events
agent1.sinks.sink1.hdfs.filePrefix = events-

agent1.sources.source1.channels=channel1
agent1.sinks.sink1.channel=channel1
```

SQOOP

# Sqoop



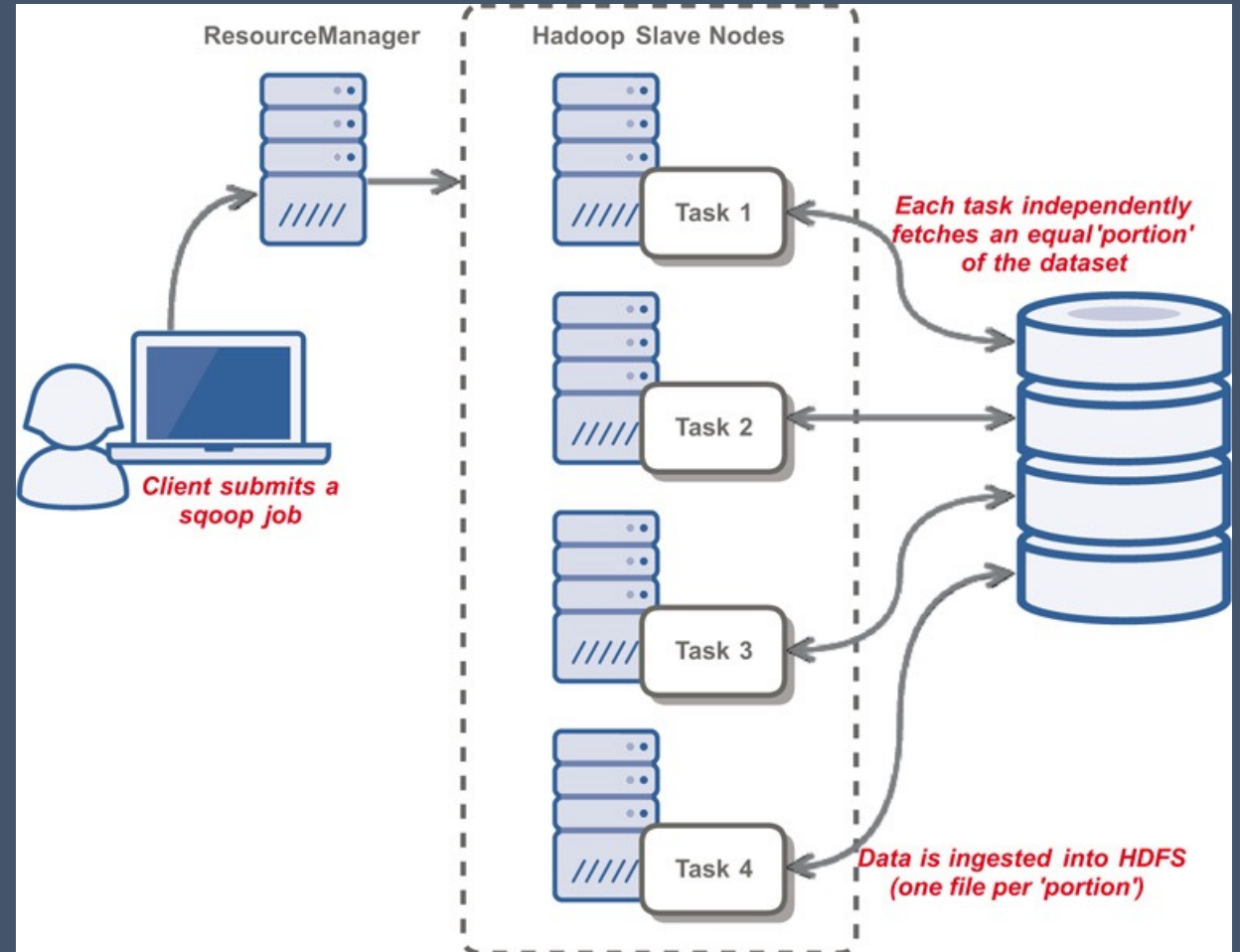
- Projeto Apache
  - <http://sqoop.apache.org/>
- Sqoop → “SQL para Hadoop e Hadoop para SQL”
- Desenvolvido originalmente pela Cloudera
- Objetivo: pegar dados de um banco de dados relacional e ingerir esses dados em arquivos (normalmente arquivos com delimitadores) no HDFS
- Também pode ser usado para enviar dados do Hadoop para um banco de dados relacional
- Integração com o Hive (abstração SQL para MapReduce)

# Operações comuns do Sqoop

- Listagem de bancos de dados e tabelas em um sistema de banco de dados
- Importar uma única tabela de um sistema de banco de dados, incluindo
  - Especificação de quais colunas importar
  - Especificação de quais linhas importar usando uma cláusula WHERE
- Importar dados de uma ou mais tabelas usando uma instrução SELECT
- Importações incrementais de uma tabela em um sistema de banco de dados (importando apenas o que mudou desde um estado anterior conhecido)
- Exportação de dados do HDFS para uma tabela em um sistema de banco de dados remoto

# Funcionamento do Sqoop

- Implementado usando MapReduce (somente etapa Map)
- Etapas (operação de importação)
  1. Conecta ao sistema de banco de dados usando JDBC ou um conector cliente
  2. Examina a tabela a ser importada
  3. Cria uma classe Java para representar a estrutura (esquema) da tabela especificada
  4. Usa o YARN para executar um job MapReduce (somente Map) com um número especificado de tarefas (mappers) para se conectar ao sistema de banco de dados e importar dados da tabela especificada em paralelo. O número padrão de tarefas paralelas é 4.





# Comandos Sqoop

## Available commands:

codegen	Generate code to interact with database records
create-hive-table	Import a table definition into Hive
eval	Evaluate a SQL statement and display the results
export	Export an HDFS directory to a database table
help	List available commands
import	Import a table from a database to HDFS
import-all-tables	Import tables from a database to HDFS
import-mainframe	Import datasets from a mainframe server to HDFS
job	Work with saved jobs
list-databases	List available databases on a server
list-tables	List available tables in a database
merge	Merge results of incremental imports
metastore	Run a standalone Sqoop metastore
version	Display version information

# Exemplos de importação com Sqoop

- Importando todas as tabelas

```
$ sqoop import-all-tables \  
--username root \  
--connect jdbc:mysql://localhost/mydb
```

- Importando uma única tabela para um diretório específico baseado em uma condição

```
$ sqoop import \  
--connect jdbc:mysql://localhost/mydb \  
--username root \  
--table mytable \  
--m 1 \  
--where "mykey = 'myval'" \  
--target-dir /myhdfsdir
```

# Exemplos de importação com Sqoop

- Importação incremental

```
$ sqoop import \  
--connect jdbc:mysql://localhost/mydb \  
--username root \  
--table mytable \  
--m 1 \  
--incremental append \  
--check-column id \  
--last-value 1000
```

# Exemplos de exportação com Sqoop

- A tabela a ser exportada deve ser criada manualmente e deve estar presente no banco de dados para onde deve ser exportada

```
$ sqoop export \  
--connect jdbc:mysql://localhost/mydb \  
--username root \  
--table mytable \  
--export-dir /myhdfsdir/mytable
```

WEBHDFS/HTTFS

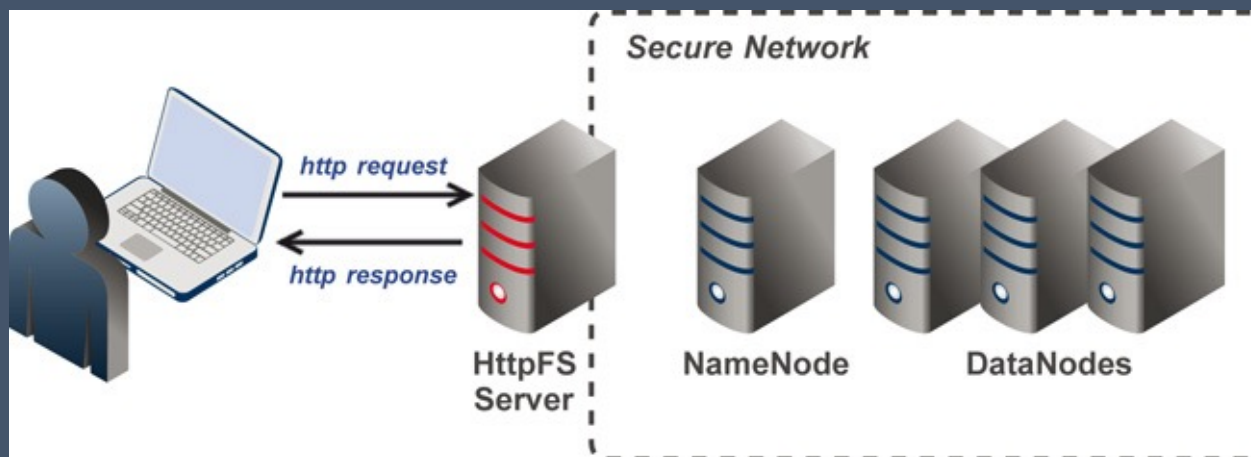
# WebHDFS

- WebHDFS fornece acesso RESTful ao HDFS usando HTTP ou HTTPS
- Suporta operações de leitura e gravação
- Utilitários de linha de comando, como wget ou curl, podem ser usados para acessar o HDFS
- Exemplo
  - ```
$ curl -i -L
```

  
<http://namenode:50070/webhdfs/v1/data/file.txt?op=OPEN>
- Limitações
  - WebHDFS não oferece suporte a implementações HDFS de alta disponibilidade (HA)
  - Os clientes devem ser capazes de acessar cada DataNode no cluster

# HttpFS

- HttpFS fornece acesso RESTful por meio de um serviço
- A solução é mais escalável, suportando implementações HDFS com HA e não exigindo acessibilidade direta do cliente a DataNodes no cluster
- O servidor HttpFS atua como um proxy, aceitando solicitações REST de clientes e enviando-as ao HDFS em nome dos clientes



# Considerações



# Ingestão de dados - considerações

- Quando um arquivo está sendo gravado no HDFS, o arquivo inicialmente aparece para os clientes como um arquivo de zero bytes no diretório de entrada com um sufixo `._COPYING_`
- Os dados são confirmados no arquivo em incrementos de tamanho de bloco (normalmente 128 MB)
  - O tamanho do arquivo aumenta em incrementos de 128 MB
- O arquivo fica visível no diretório HDFS de destino e pode ser lido (e, portanto, usado como entrada para processamento) enquanto o arquivo ainda não está completo
  - Podem ocorrer problemas ao processar arquivos incompletos
- Recomendação
  - Usar um diretório especial (por exemplo, `./incoming`) para ingerir dados
  - Assim que a operação de gravação for concluída, mover o arquivo para um diretório acessível (por exemplo, `./ingested`)
    - A operação de movimentação é (quase) instantânea - requer apenas uma operação de metadados

**Ciências de Dados e Inteligência Artificial**

# Gerência de Infraestrutura para Big Data

Ingestão de dados

Prof. Tiago Ferreto

*tiago.ferreto@pucrs.br*

**PUCRS** online  **UOL** edtech\_