NOSQL – COLUMN FAMILY

MARCIO JASINSKI

FURB

HBase

HBASE

- Banco de dados open source orientado a coluna
- Nasceu como Google Big Table e depois foi renomeado para Hbase
- Escrito principalmente em Java
- Executa sobre o Haddoop (biblioteca para processamento distribuído)
- Capaz de armazenar uma grande quantidade de dados (terabytes, petabytes) ainda ser performático
- O Hbase foca em consistência e escalabilidade
- Inicialmente descrito no Paper sobre o BigTable
- Foi criado inicialmente para Prcessamento Natural de Linguagem

HBASE

- Características
  - Low latency operations
  - Desempenho “constante” para random read e write operations
  - Armazenamento de grandes quantidades de dados em tabelas
  - Mantém resposta linear conforme a demanda aumenta (considerando um cluster escalável)
  - Permite configurar sharding entre tabelas automaticamente
  - Failover automático entre regiões
  - Diversas APIs para operar com o banco

HBASE

- Totalmente projetado para “fault tolerance” – ambiebtes onde problemas com servidores são a norma, não a exceção
- A estratégia usada para essa resiliência se chama “write-ahead-logging”
  - Escreve os dados em um log em memória antes de colocar no disco.
  - Isso permite a outros nós consultar esses logs em caso de falha com um servidor e não depender do disco
  - Como é distribuído, os nós podem confiar uns nos outros e não em um “servidor central”

HBASE – LIMITAÇÕES

- Não possibilita a execução de comandos SQL ou joins tradicionais para fácil relacionamento dentre entidades
- É um banco que utiliza muitos recursos computacionais como memória, CPU e disco de forma distribuída
- Para entregar mais, precisa ser combinado com elementos adicionais como Hive, Drill e Phoenix
- Complexo de gerenciar e de configurar
- Tem limitações quanto à flexibilidade do modelo schemaless

HBASE – ELEMENTOS DE ARMAZENAMENTO – PARTE 1

O Hbase tem alguns elementos essenciais para estruturar os dados:

- **Namespace**: Um agrupamento lógico de tabelas
- **Tabela** – Coleção de linhas (rows)
- **Row** – Coleção de família de colunas
- **Família de colunas** – Coleção de colunas
- **Coluna**: coleção de pares chave-valor
- **Célula (Cell)**– uma tupla {linha, coluna, versão}

HBASE - “SEMELHANÇAS” COM OUTROS BANCOS

- É difícil uma comparação justa mas alguns elementos podem ajudar a compreender a forma de armazenamento.

RDBMS	MongoDB	HBase
Table	Collection	Table
Row	Document	Column Family
No Equivalent	Shard	Region
GROUP_BY	Aggregation Pipeline	MapReduce
Multi-record ACID transactions	Multi-document ACID transactions	None. Requires implementation in app layer

Fonte: mongodb.com

HBASE - “SEMELHANÇAS” COM OUTROS BANCOS

- Para avaliar qual utilizar, devemos levar em consideração: Modelo de Dados, Diversidade dos Dados, Escala e Storage

HBASE	**RDBMS**
Schema-less in database	Having fixed schema in database
Column-oriented databases	Row oriented data store
Designed to store De-normalized data	Designed to store Normalized data
Wide and sparsely populated tables present in Hbase	Contains thin tables in database
Supports automatic partitioning	Has no built in support for partitioning
Well suited for OLAP systems	Well suited for OLTP systems
Read only relevant data from database	Retrieve one row at a time and hence could read unnecessary data if only some of the data in a row is required
Structured and semi-structure data can be stored and processed using Hbase	Structured data can be stored and processed using RDBMS
Enables aggregation over many rows and columns	Aggregation is an expensive operation

HBASE – ELEMENTOS DE ARMAZENAMENTO – PARTE 2

- O modelo do Hbase é formado por elementos focados em armazenamento de muitos dados:
  - Cada modelo é formado por um conjunto de tabelas
  - Cada tabela tem família de colunas e linhas
  - Cada tabela precisa ter elementos definidos como “Primary key”
  - A chave da linha funciona como “Primary Key” no Hbase
  - Todo acesso às tabelas deve ser com essa chave primária
  - Cada coluna é um atributo do objeto armazenado

HBASE – ELEMENTOS DE ARMAZENAMENTO – PARTE 3

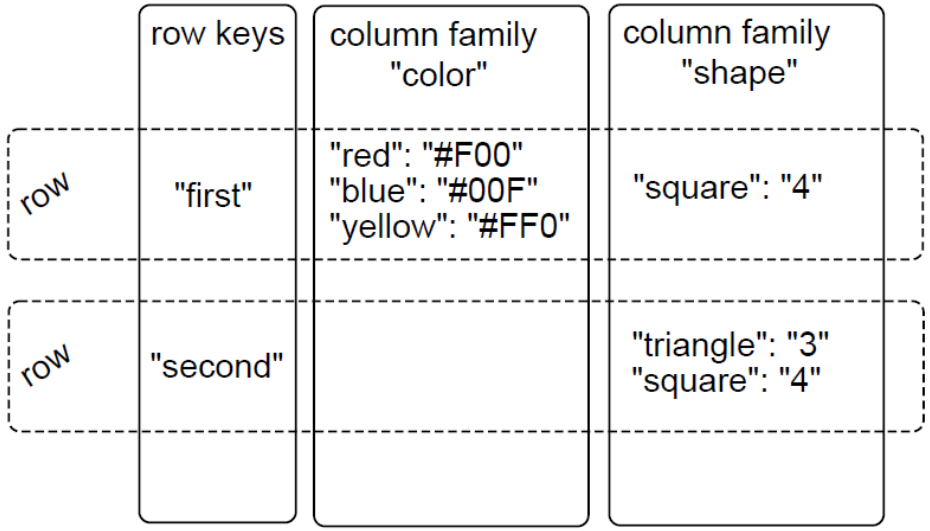
- É mais complexo do que parece:

Row Key 123.456.789-78	Column Family personal-data						Column Family professional-data			
	Column name		Column address		Column children		Column job		Column salary	
	version 1	value Joao	version 1	value Blumenau	version 1	value 2	version 1	value Manager	version 1	value 7.500,00
	version 2	value João Perdo	version 2	value Floripa					version 2	value 7.650,00
			version 3	value Florianópolis						
			version 4	value Tijucas						

*"billions of rows * millions of columns * thousands of versions = terabytes or petabytes of storage"*

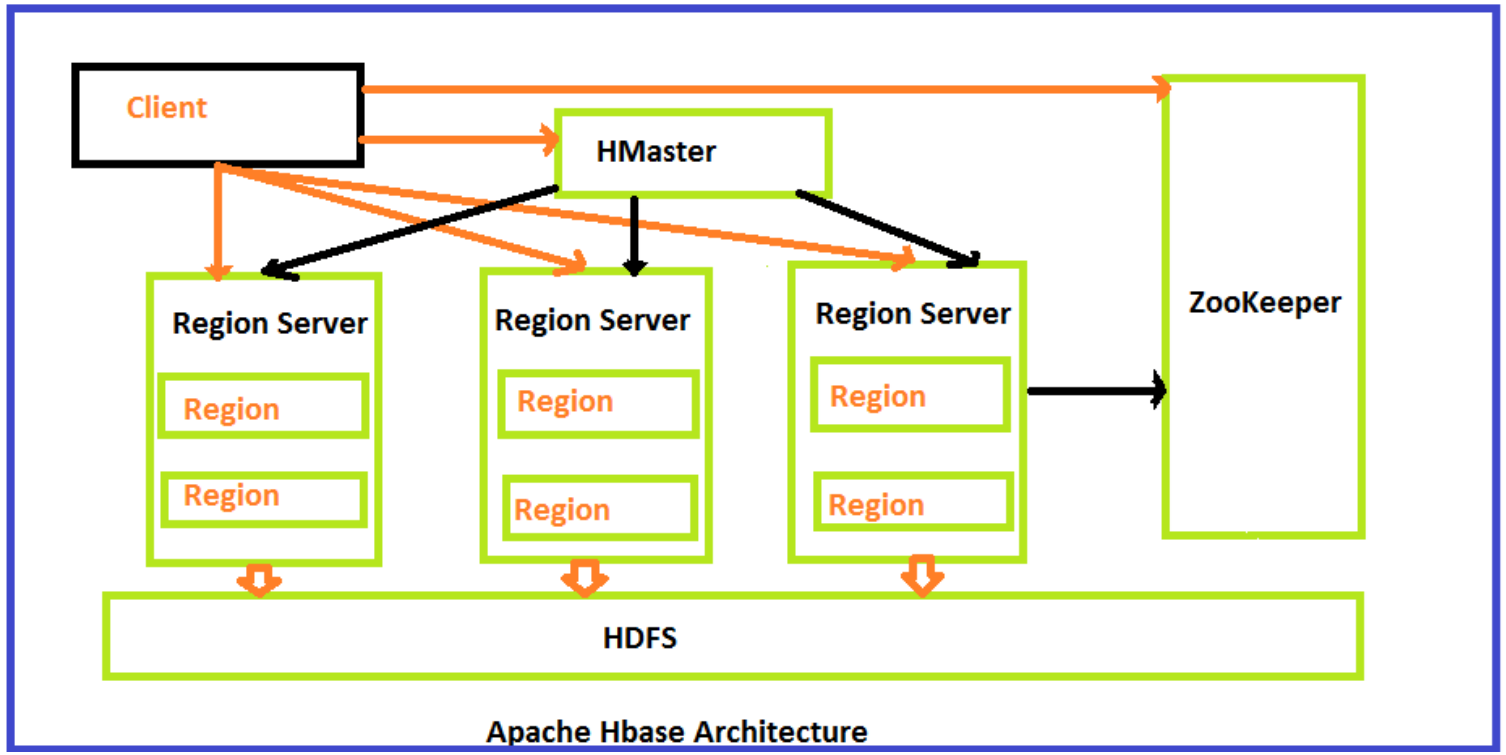
HBASE – BACK TO THE TABLE

- Uma tabela no Hbase é basicamente um grande “dicionário de dicionários” ou map of maps
- Uma tabela, é um mapa onde as chaves mapeiam para as **linhas**
- Uma linha é um mapa chave-valor, onde o valor é uma lista de **família de colunas**
- Uma família de colunas, é um mapa para uma lista de **colunas** (column qunatifiers)



HBASE – ARQUITETURA E DISTRIBUIÇÃO

- O Hbase executa sobre o HDFS e é destribado em region servers



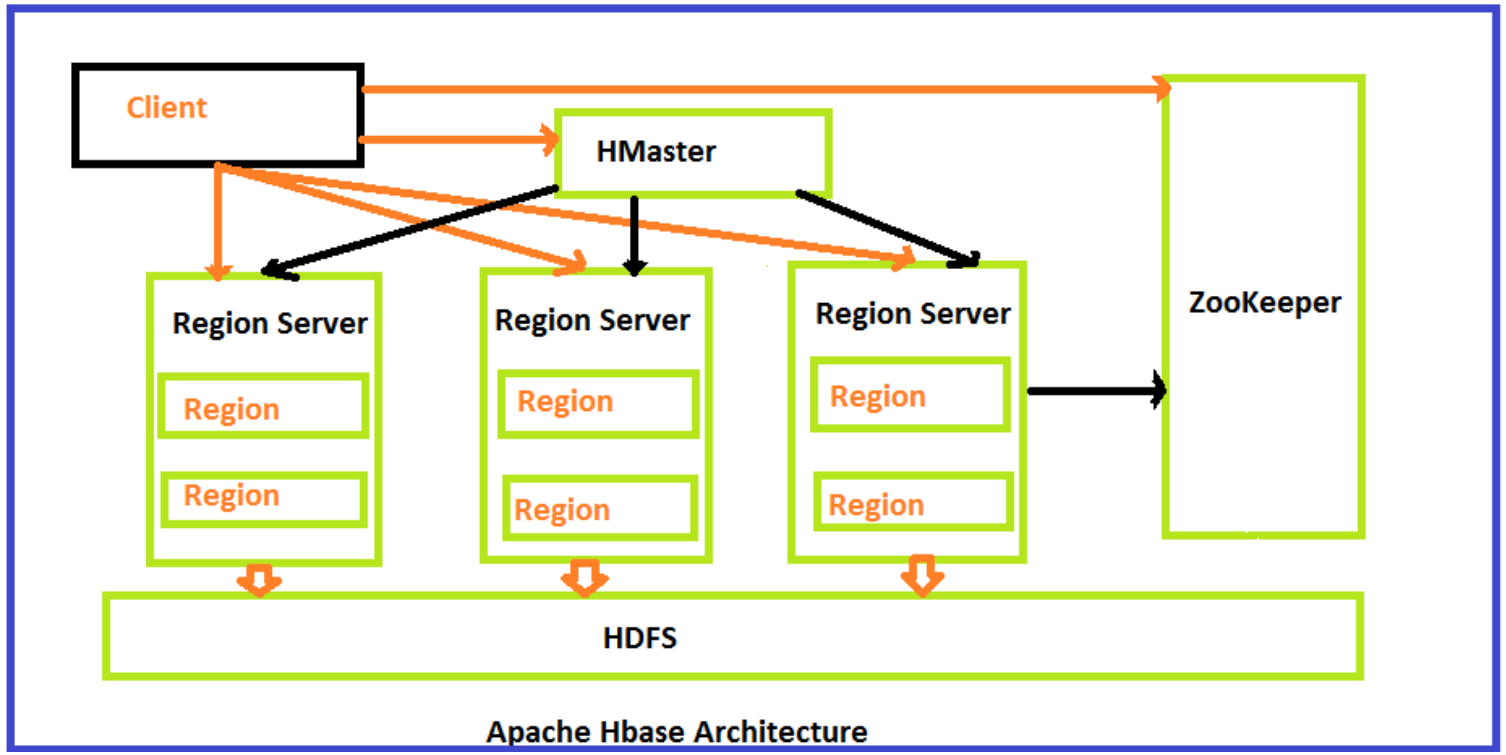
HDMaster

É o servidor principal do modelo do Hbase que atua como monitor de todas as regiões. Gerencia as mudanças de schema e alocação de regiões aos servidores.

Para ambientes com redundância, podem haver múltiplos Hmaster servers.

HBASE – ARQUITETURA E DISTRIBUIÇÃO

- O Hbase executa sobre o HDFS e é destribado em region servers



Apache Hbase Architecture

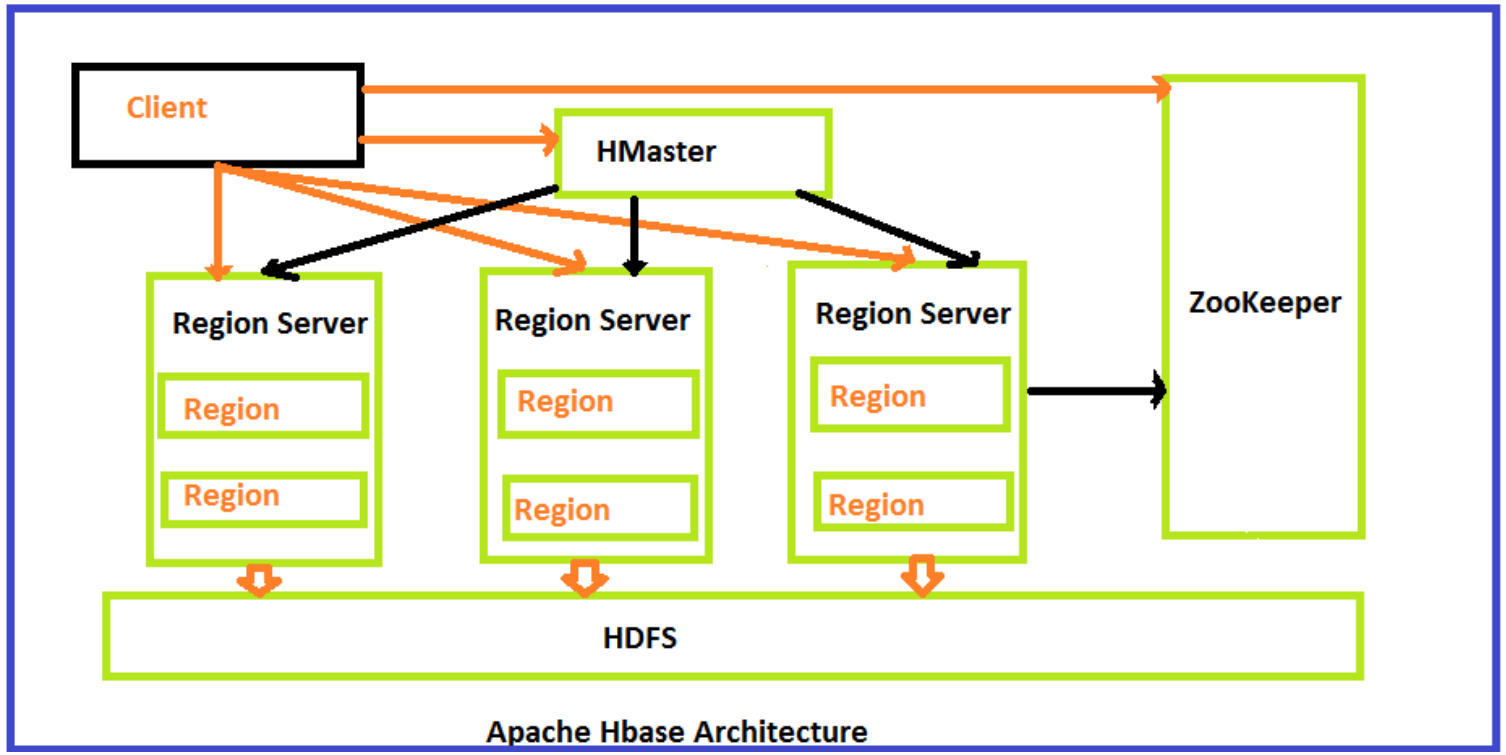
Region Server

Quando um Region Server recebe um pedido de leitura/escrita, ele direciona a requisição a uma região específica. O cliente não precisa necessariamente passar pelo Hmaster para fazer essas operações.

Region Servers executam os “data notes”

HBASE – ARQUITETURA E DISTRIBUIÇÃO

- O Hbase executa sobre o HDFS e é destribado em region servers



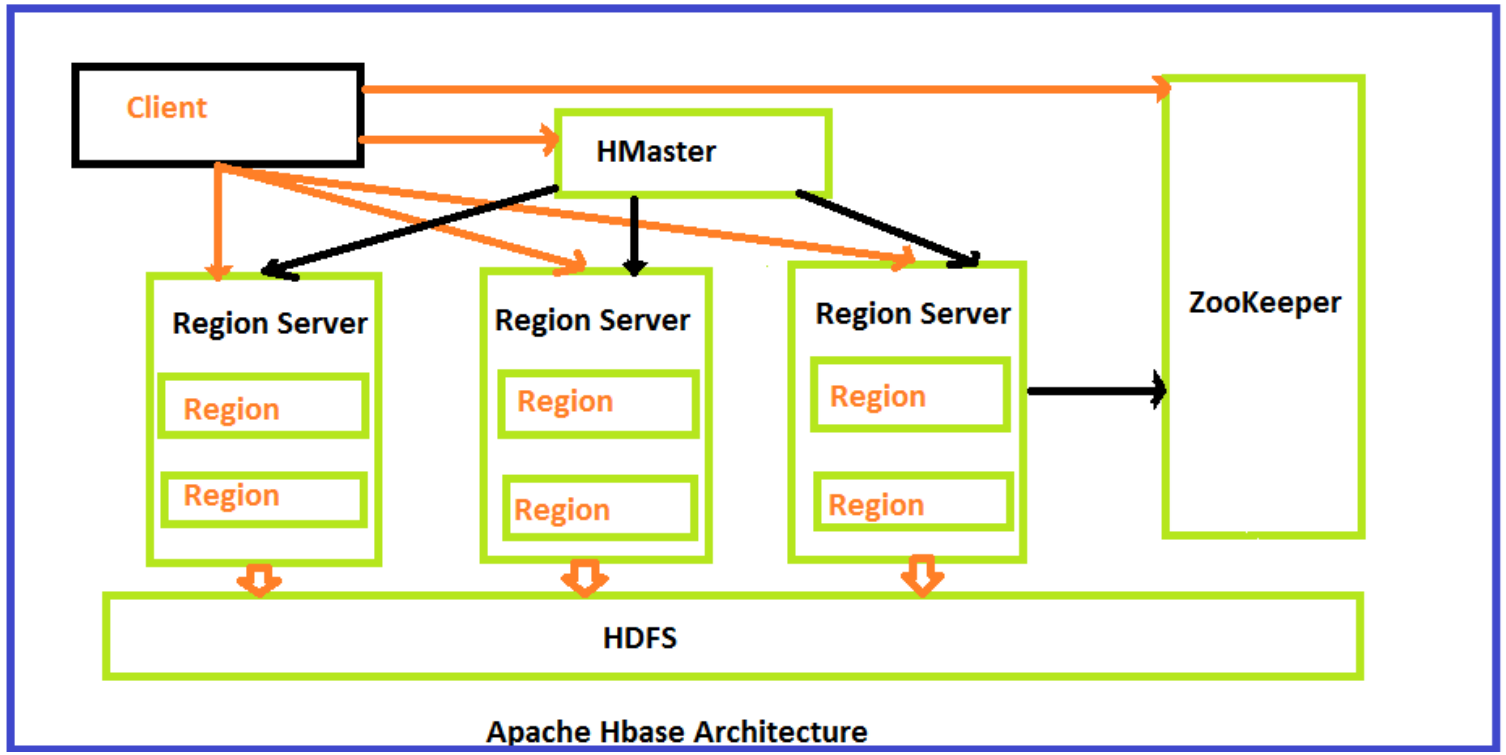
Apache Hbase Architecture

Regions

Elementos de um cluster Hbase que consistem na distribuição das tabelas e família de colunas que podem estar na memória ou no HDFS

HBASE – ARQUITETURA E DISTRIBUIÇÃO

- O Hbase executa sobre o HDFS e é destribado em region servers



Apache Hbase Architecture

ZooKeeper

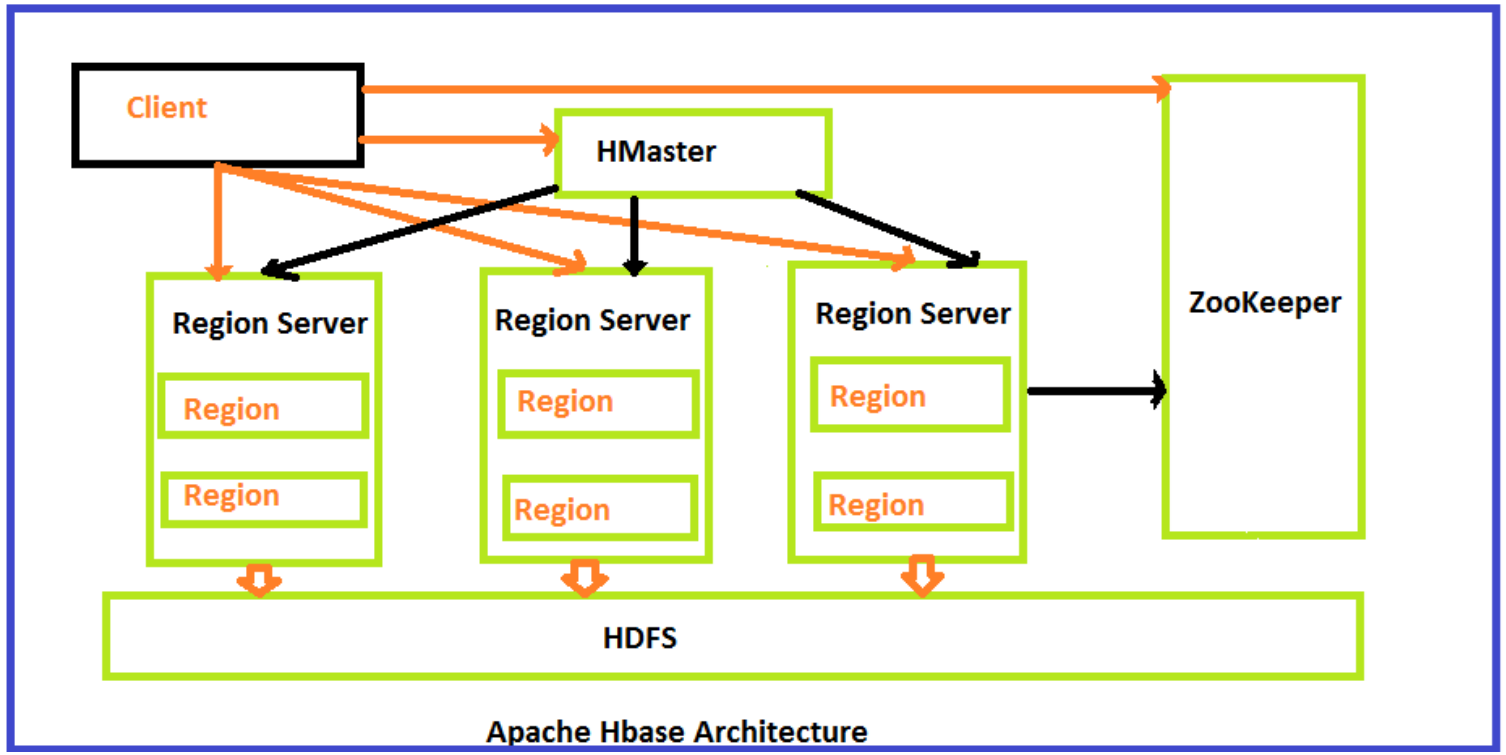
Centraliza o monitoramento e configurações para sincronização do ambiente.

É o ZooKeeper que permite ao cliente chegar até os Region Servers (pelo menos inicialmente).

Todos os servers inclusive Hmasters se registram no ZooKeeper

HBASE – ARQUITETURA E DISTRIBUIÇÃO

- O Hbase executa sobre o HDFS e é destribado em region servers



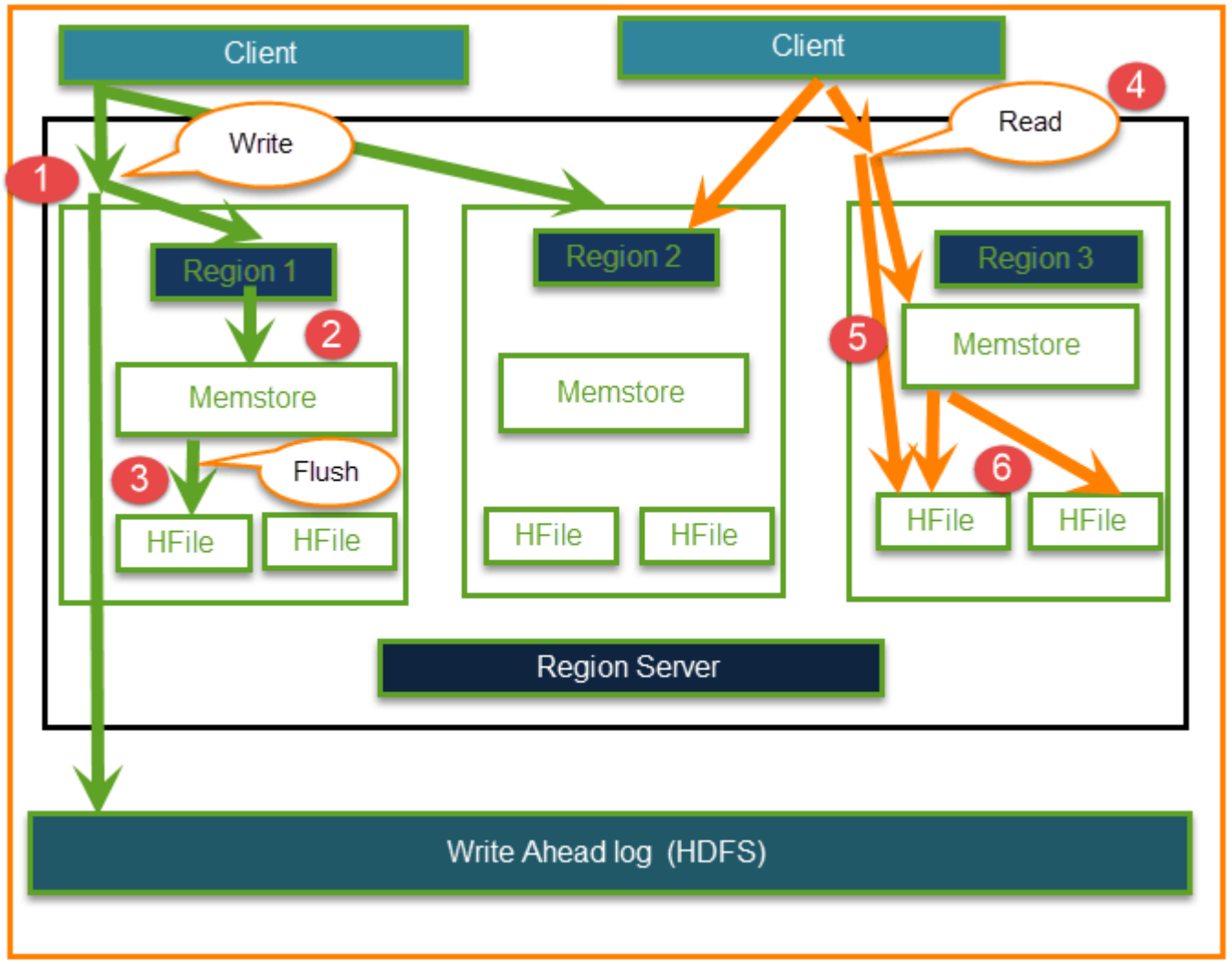
Apache Hbase Architecture

HDFS

Haddop File System é um sistema de arquivos distribuído. Ele armazena cada arquivo em mutiplos blocos espalhados pelo cluster.

Assim, oferece um alta confiança em termos de tolerância à falhas e em geral, alta performance para paralelizar grande quantidade de processamento.

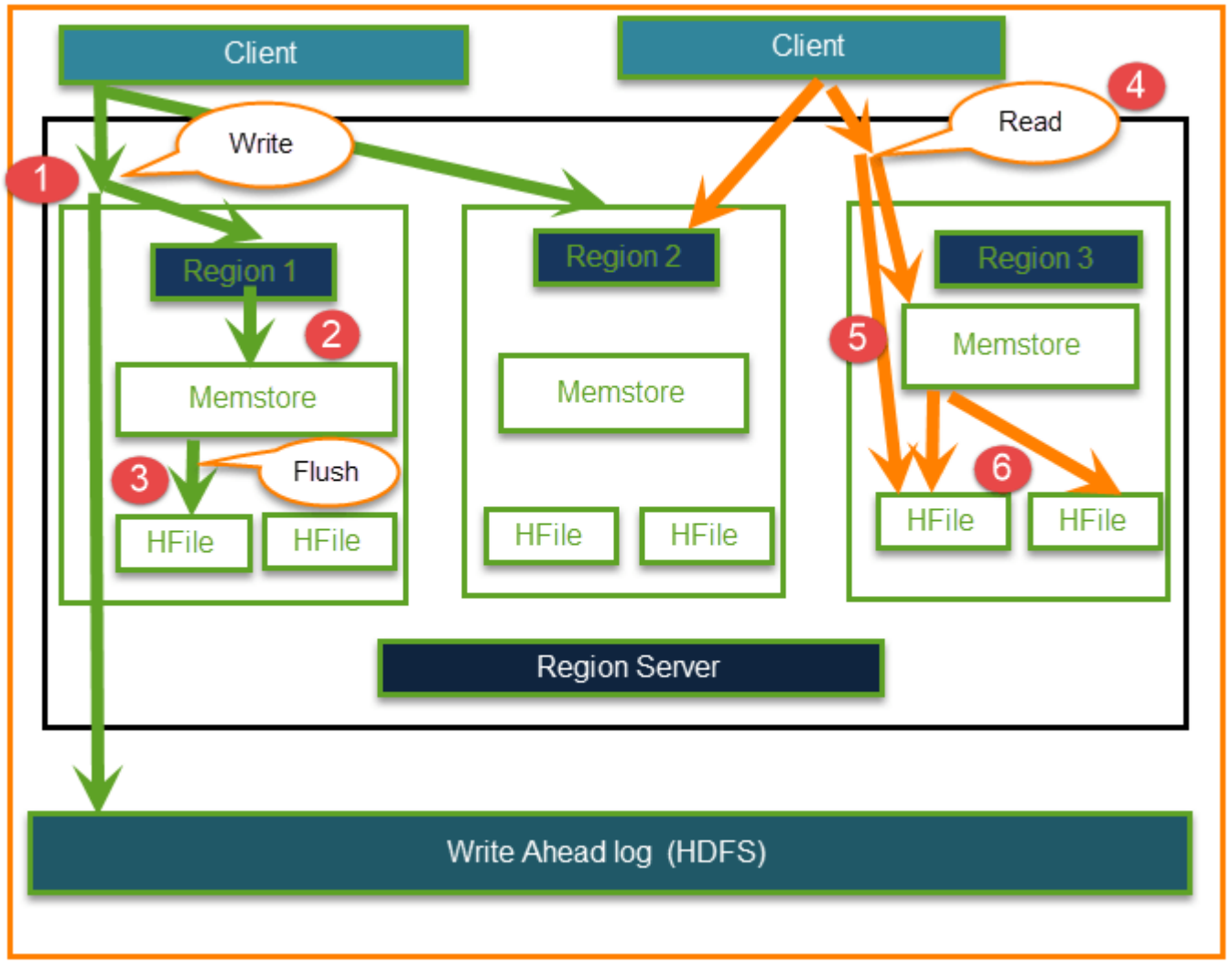
HBASE – ARQUITETURA E DISTRIBUIÇÃO



Escrita

1. Cliente notifica o region server sobre a operação; Em seguida, a operação enviada para ser gravada como logo adiantado
2. Region Server envia para a memstore para ordenação e organização dos dados.
3. Os dados da memstore são enviados para escrita no arquivo

HBASE – ARQUITETURA E DISTRIBUIÇÃO



Leitura

4. Cliente solicita leitura para os servidores de região.
5. Os dados disponíveis na Memstore são disponibilizados. Alguns dados podem ser retornados diretamente aqui.
6. Se o dado não estava na memstore, são carregados do HDFS

HBASE – ARQUITETURA E DISTRIBUIÇÃO

- Por que não usar apenas HDFS?

HBASE	HDFS
• Low latency operations	• High latency operations
• Random reads and writes	• Write once Read many times
• Accessed through shell commands, client API in Java, REST, Avro or Thrift	• Primarily accessed through MR (Map Reduce) jobs
• Storage and process both can be perform	• It's only for storage areas

Instalação e Configuração

HBASE – INSTALAÇÃO E CONFIGURAÇÃO

- Existem 3 modos de execução
  - Standalone – modelo de apenas uma máquna
  - Pseudo-distribuited – Um nó apenas que se comporta como se fosse um cluser
  - Fully distribuited – Modo de produção com um cluster de servidores trabalhando em conjunto.

HBASE – HBASE SHELL

- Ambiente para execução de comandos e administração
- Permite execução de scripts Ruby e comandos Java

```
hbase> import java.util.Date
hbase> Date.new(1218920189000).toString() => "Sat Aug 16 20:56:29 UTC
2008"
```

- Verificar e alterar configurações

```
hbase> @shell.hbase.configuration.get("hbase.rpc.timeout") => "60000«
base> @shell.hbase.configuration.setInt("hbase.rpc.timeout", 61010)
```

Data Definition Language

HBASE – NAMESPACE MANAGEMENT

- Um namespace é um agrupamento lógico de tabelas.
- É o equivalente ao conceito de um database em um RDBMS
- Deve ser utilizado para separação lógica de ambientes, como em modelos multitenant
- Existem 2 namespaces default
  - hbase – system namespace para tabelas internas do hbase
  - Default – tabelas sem namespace definido, são automaticamente alocadas no namespace default

HBASE – NAMESPACE MANAGEMENT

- Operações sobre namespace

```
create_namespace ‘<namespace name>’
```

```
drop_namespace ‘<namespace name>’
```

```
alter_namespace ‘<namespace name>’, {METHOD => 'set',
'PROPERTY_NAME' => 'PROPERTY_VALUE'}
```

HBASE – CREATE TABLE

- Tabelas no Hbase precisam ser criadas em tempo de definição do schema, ou seja, não podem ser criadas “data on write”
- Sintaxe de criação de uma tabela

```
create ‘<table name>’,’<column family>’, ’<column family>’
```

```
create ‘<namespce>:<table name>’,’<column family>’, ’<column family>’
```

- Como pode ser visto, as famílias de colunas são criadas junto com a criação da tabela
- Isso relembra um dos desafios do modelo relacional, pensar bem na estrutura inicial
- De fato, alterar as famílias de colunas pode ser bastante trabalhoso ou mesmo inviável em algumas situações
- Porém, é possível observar que tipos e tamanhos não são informados nesse processo inicial

HBASE – CREATE TABLE

- A criação de uma tabela utiliza a seguinte sintaxe

```
create 'employees', 'personal-data', 'professional-data'
```

- Para simplificar, podemos apresentar a estrutura de tabela.
- Mas na verdade, cada família de colunas pode ter linhas com chaves distantas

Row	personal-data		professional-data	
rowId	key1	key2	keyN	keyM
	Matteo	Pompeia	Manager	17.000,00

HBASE – LIST AND DISABLE/ENABLE TABLE

- Para listar as tabelas existentes no Hbase, basta utiliza rum **list** no shell

```
hbase(main):001:0 > list
TABLE
employees
branches
Companies
payroll
```

- Uma ação muito comum no Hbase é desativar uma tabela para manutenção. Como mudanças de estrutura...

- E de forma semelhante, o a reativação da tabela após concluir a manutenção. Essas operações são feitas com **enable/disable**

```
hbase(main):001:0 > disable employees
0 row(s) in 1.2760 seconds

hbase(main):001:0 > enable employees
0 row(s) in 0.4789 seconds
```

HBASE – DESCRIBE TABLE

- O **describe** é um comando que retorna a estrutura de uma tabela em termos de família de colunas:

```
hbase(main):001:0> describe 'employees'
Table employees is ENABLED
employees
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal-data', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false',
NEW_VERSION_BEHAVIOR => 'false', KEEP_DELET
ED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => 'false', DATA_BLOCK_ENCODING =>
'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0
', REPLICATION_SCOPE => '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE =>
'false', IN_MEMORY => 'false', CACHE_BLOOMS_O
N_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSION => 'NONE',
BLOCKCACHE => 'true', BLOCKSIZE => '65536
'}
…
2 row(s)
```

HBASE – ALTER TABLE

- O **alter** é o comando utilizado para modificar a estrutura das famílias de colunas de uma tabela
- Nas alterações é possível indicar também a quantidade de versões que uma família de colunas deve armazenar
- N**ão é possível** renomear uma família de colunas

```
alter <tablename>, NAME=>'<column familyname>', VERSIONS=><new version no>
```

HBASE – ALTER TABLE

- O **alter** é o comando utilizado para modificar a estrutura das famílias de colunas de uma tabela
- Idealmente, as alterações devem ser realizadas com a tabela em modo offline (ou seja, utilizando o disable/enable)
- Isso depende da configuração da chave `hbase.online.schema.update.enable`

```
hbase> disable 'employees'
Took 0.5313 seconds

hbase> alter 'employees', NAME => 'social-networks', VERSIONS => 5
Updating all regions with the new schema...
1/1 regions updated.
Done.
Took 2.2702 seconds

hbase> enable 'employees'
Took 0.8256 seconds
```

HBASE – ALTER TABLE

- O **alter** é o comando utilizado para modificar a estrutura das famílias de colunas de uma tabela
- Principais atributos do scopo de alteração
- MAX_FILESIZE
- READONLY, MEMSTORE_FLUSHSIZE
- DEFERRED_LOG_FLUSH

```
hbase> alter 'employees', READONLY
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.2140 seconds
```

HBASE – DROP TABLE

- O **drop** tem o mesmo efeito de um banco relacional, remove a tabela.
- É possível utilizar o **drop_all** que faz drop com expressões regulares!
- As tabelas devem estar desativadas para o drop ter efeito.

```
hbase> drop 'employees'
ERROR: Table employees is enabled. Disable it first.

For usage try 'help "drop"'

Took 0.0419 seconds

hbase> drop_all 'empl.*'
```

HBASE – RENOMEANDO UMA FAMÍLIA DE COLUNAS

- Não existe uma forma de renomear uma família de colunas no Hbase - pelo menos ainda não em 2020 ☹
- Então se for necessária uma alteração, a única forma é através de scripts e APIs:

1. Desativar a tabela
2. Criar a nova família de coluna
3. Copiar todos os dados para a nova família de colunas
4. Remover a família de colunas nova
5. Reativar a tabela

HBASE – COMANDOS UTILITÁRIOS

- É possível testar se uma tabela está ativa com **`is_enabled/is_disabled`**
- É possível verificar se a tabela existe com o **`exists`**
- O comando **`get_table`** permite trazer uma tabela como variável e chamar comandos de forma direta

```
Hbase> t = get_table 'employees'
hbase> t.enable
hbase> t.flush
hbase> t.disable
hbase> t.drop
```

HBASE – RENOMEANDO TABELAS

```
hbase(main):005:0> disable 'employees'
Took 0.4721 seconds

hbase(main):006:0> snapshot 'employees', 'employees2020.03.02.snapshot'
Took 0.7065 seconds

hbase(main):008:0> clone_snapshot 'employees2020.03.02.snapshot', 'italians'
Took 1.4440 seconds

hbase(main):009:0> delete_snapshot  'employees2020.03.02.snapshot'
Took 0.0495 seconds

hbase(main):010:0> drop 'employees'
Took 0.7608 seconds

hbase(main):011:0> enable 'italians'
Took 0.0183 seconds
```

CRUD &
Data manipulation

HBASE – INSERT DATA

- A inclusão de dados é possível com o comando **put** no shell do hbase
- Na prática, existem classes com drivers do Hbase para essa operação como `Htable.put(...)` e `Put.add(...)`
- Sempre é necessário informar a chave da linha a ser inserida
- A família de colunas precisa estar previamente criada

put ’<table name>’, ’row1’, ’<colfamily:colname>’ , ’<value>’

- Via shell, há uma limitação de uma família de colunas por vez

HBASE – INSERT DATA

- Exemplo de dados “bem comportados”

Row key	personal-data		professional-data	
employeeId	name	city	role	salary
20050	Matteo	Pompeia	Manager	17.000,00
20051	Silvia	Roma	Analyst	11.000,00
20052	Alex	Milano	Physicist	15.500,00

```
put  'employees', '20050', 'personal-data:name' , 'Matteo Rossi'
put 'employees', '20050', 'personal-data:city' , 'Pompeia'
put  'employees', '20050', 'professional-data:designation' , 'Manager'
put  'employees', '20050', 'professional-data:salary' , '17.000'
```

HBASE – READ DATA

- A leitura dos dados é através do método **get**

  get ’<table name>’, ’row-key’

- É obrigatório informar a chave da row desejada

```
hbase(main):017:0> get 'employees', '20050'
COLUMN                              CELL
 personal-data:city                 timestamp=1583398007168, value=Pompeia
 personal-data:name                 timestamp=1583397995831, value=Matteo
 professional-data:designation      timestamp=1583398031602, value=Manager
 professional-data:salary           timestamp=1583398044313, value=17.000,00
```

HBASE – VERSIONS

- Uma tupla {row, column, version} é o que define uma célula (cell) no HBASE.
- É possível ter inúmeras céluas onde a linha e coluna são a mesma, mas em diferentes versões
- As versões são sempre representadas em um long que representa o timestamp (java.util.Date.getTime())
- O versionamento é armazenado em ordem decrescente, assim o maior timestamp é sempre retornado primeiro
- É possível limitar a quantidade de versões de uma determinada chave/valor
- Por padrão, cada chave possui valor default de VERSIONS=1
- É possível controlar o versionamento se desejar

```
alter 't1', NAME => 'f1', VERSIONS => 5
```

HBASE – READ DATA

- O **get** permite obter as versões de uma determinada coluna

get ’<table name>’, ’row-key’, {COLUMN => ‘column-family:column’, VERSIONS => 3}

get ’<table name>’, ’row-key’, {COLUMN => ‘column-family:column’, TIMESTAMP => 1317945301466}

```
hbase(main):017:0> get 'employees', ‘1112’,{COLUMN => ‘personal-data:name’, VERSIONS => 2}
COLUMN                                CELL
 personal-data:name                   timestamp=1583397995831, value=Matteo
 personal-data:name                   timestamp=1583397995784, value=Mateo
```

- Por padrão, todas as operações retornam os dados de forma ordenada por:
  1. Linha
  2. Família de Colunas
  3. Coluna
  4. Timestamp

HBASE – READ DATA

- Para uma visão mais abrangente dos dados de uma tabela, utiliza-se a função **scan**

```
scan  '<table name>', {Optional parameters}
```

- O scan percorre a tabela e mostra os resultados da mesma. Semelhante a um SELECT * FROM TABLE
- Os atributos opcionais são:
  - TIMERANGE
  - FILTER
  - TIMESTAMP
  - LIMIT
  - MAXLENGTH
  - COLUMNS
  - CACHE
  - STARTROW e STOPROW

HBASE – READ DATA

- Para uma visão mais abrangente dos dados de uma tabela, utiliza-se a função **scan**

scan ’<table name>’, {Optional parameters}

```
hbase(main):018:0> scan 'employees'
ROW   COLUMN+CELL
 20050    column=personal-data:city, timestamp=1583398007168, value=Pompeia
 20050    column=personal-data:name, timestamp=1583397995831, value=Matteo
 20050    column=professional-data:designation, timestamp=1583398031602, value=Manager
 20050    column=professional-data:salary, timestamp=1583398044313, value=17.000,00
 20051    column=personal-data:city, timestamp=1583398007471, value=Roma
 20051    column=personal-data:name, timestamp=1583397994778, value=Silvia
 20051    column=professional-data:designation, timestamp=1583398034781, value=Analyst
 20051    column=professional-data:salary, timestamp=1583398044313, value=11.000,00
```

HBASE – READ DATA

Commando	Efeito
scan ‘italians’, {COLUMNS => personal-data:name'}	Projeta apenas os dados da FC personal-data com a chave name.
scan ‘italians’, {COLUMNS => [‘personal-data’, ‘professional-data’], LIMIT => 3, STARTROW => ‘5'}	Lista os valores para as FCs personal-data e professional data com limite de 3 registros a partir da linha 5
scan ‘italians’, {COLUMNS => ‘personal-data’, TIMERANGE => [1303668804, 1303668904]}	Consulta a partir do versionamento com timerange e projeção apenas de uma FC chamada personal-data
scan 'italians', {RAW => true, VERSIONS =>10}	O commando raw pode ser usado para apresnetar todos os dados da tabela (versões/histórico e dados removidos)

HBASE – UPDATE DATA

- A atualização dos dados é feita com o mesmo método de inserção, o **put**

```
put 'table name','row ','Column family:column name','new value'
```

- Basta substituir as chaves com um novo valor, ou mesmo indicar uma nova chave:

```
put 'employees','20050 ','personal-data:city','Florianópolis'
put 'employees','20050 ','personal-data:bloodType','A+'
```

- Porém, **não é possível adicionar uma nova família de colunas** com o put.

HBASE – COUNT DATA

- O **count** contabiliza a quantidade de linhas em uma tabela

```
count 'table name', INTERVAL => value, CACHE => value
```

- É uma operação que pode demorar significativamente pois é single thread e precisa conversar com os region servers
- Para operações de map reduce, existem utilitários adicionais
  - '$HADOOP_HOME/bin/hadoop jar rowcount'
  - org.apache.hadoop.hbase.mapreduce.RowCounter

```
hbase> count italians'
2 row(s)
Took 0.3457 seconds
=> 2
```

HBASE – COUNT DATA

- O **count** contabiliza a quantidade de linhas em uma tabela
- CACHE – Quantidade de linhas buscadas por vez durante a contagem
- INTERVAL – Determina a contagem de retorno

```
hbase(main):024:0> count 'italians', INTERVAL => 2
Current count: 2, row: 10
Current count: 4, row: 3
Current count: 6, row: 5
Current count: 8, row: 7
Current count: 10, row: 9
10 row(s)
Took 0.0201 seconds
=> 10
```

HBASE – DELETE DATA

- O **delete** é um comando mais flexível do que é comum nos tradicionais bancos de dados
  - Pode agir sobre uma versão específica de uma coluna
  - Para todas as versões de uma coluna
  - Delete todas as colunas de uma família de colunas
- Deletar uma célula específica, informando o mapeamento completo de linha, famlia de coluna, chave e timestamp

```
delete ‘<table name>’, ‘<row>’, ‘<column name >’, ‘<time stamp>’
```

- Todas as colunas de uma família de colunas. O número mínimo de parâmetros é 3 (tabela, linha e família de colunas)

```
delete 'employees', '20050', 'personal-data'
```

HBASE – DELETE DATA

- Para remover todos os dados de uma linha de uma vez, é necessário usar o **deleteall**

```
deleteall ‘<table name>’, ‘<row>’
```

- Por exemplo

```
delete 'employees', '20051'
```

HBASE – COUNTERS

- Semelhante ao que existe em bancos como Redis, o Hbase também tem suporte a contadores de forma atômica
- O **incr** atua sobre um valor de uma célula para incrementar de forma atômica
- O operador get_counter faz a leitura
- Sempre inicializar o valor através do incr e não com puts

```
create 'blog', 'pages'

incr 'blog', 'bbc', 'pages:visitors'
COUNTER VALUE = 1
Took 0.0108 seconds

get_counter 'blog', 'bbc', 'pages:visitors'
COUNTER VALUE = 1
Took 0.0077 seconds
```

HBASE – TRUNCATE

- O **truncate** é uma função de limpeza de uma tabela
- Quando executado, ele realiza as seguintes operações:
  - Disable da tabela
  - Drop da tabela
  - Recriação da tabela

```
hbase(main):011:0> truncate 'emp'
Truncating 'one' table (it may take a while):
   - Disabling table...
   - Truncating table...
   0 row(s) in 1.5950 seconds
```

HBASE – EXECUTANDO COMANDOS E SCRIPTS

- Ao invés de digitar comandos no shell, é possível executar arquivos com comandos do shell:

```
cat /tmp/italians.hbase
create 'italians', 'personal-data',  'professional-data'
put 'italians', '1', 'personal-data:name',  'Gianni Guerra'
put 'italians', '1', 'personal-data:city',  'Florence'
put 'italians', '1', '‘professional-data:role',  'Nutrição'
put 'italians', '1', '‘professional-data:salary',  '5794'
put 'italians', '2', 'personal-data:name',  'Carlo Benedetti'
put 'italians', '2', 'personal-data:city',  'Modena'
put 'italians', '2', '‘professional-data:role',  'Gestão Desportiva e
de Lazer'
put 'italians', '2', '‘professional-data:salary',  '8787’
…

hbase shell /tmp/italians.hbase
```

HBASE – EXECUTANDO COMANDOS E SCRIPTS

- Outra opção é a importação de arquivos CSV

```
hbase> create 'emp_table',{NAME => 'cf’}
```

```
$ cat /tmp/emp_data.csv
7369,SMITH,CLERK,7902,12/17/1980,800,20
7499,ALLEN,SALESMAN,7698,2/20/1981,1600,30
7521,WARD,SALESMAN,7698,2/22/1981,1250,30
7566,TURNER,MANAGER,7839,4/2/1981,2975,20

$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
-Dimporttsv.separator= ’,’ -Dimporttsv.columns=
      ’HBASE_ROW_KEY,cf:ename,cf:designation,cf:manager,cf:hire_date,
      cf:salary,cf:deptno’
      emp_table /tmp/emp_data.csv
```

Map Reduce HBase

HBASE – EXECUTANDO COMANDOS E SCRIPTS

- O map reduce no Hbase só pode ser executado com linguagens de programação.
- A função de Map Reduce está de fato no Haddop
- O Hbase permite a criação de Jobs para execução
- O MapReduce pode ser usado para processamento paralelo de muitos dados

HBASE – MAP REDUCE READ

```
Configuration config = HBaseConfiguration.create();
Job job = new Job(config, "ExampleRead");
job.setJarByClass(MyReadJob.class);     // class that contains mapper

Scan scan = new Scan();
scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
scan.setCacheBlocks(false);  // don't set to true for MR jobs
// set other scan attrs
...

TableMapReduceUtil.initTableMapperJob(
  tableName,        // input HBase table name
  scan,             // Scan instance to control CF and attribute selection
  MyMapper.class,   // mapper
  null,             // mapper output key
  null,             // mapper output value
  job);
job.setOutputFormatClass(NullOutputFormat.class);   // because we aren't emitting anything from
mapper

boolean b = job.waitForCompletion(true);
if (!b) {
  throw new IOException("error with job!");
}
```

HBASE – MAP REDUCE READ

```java
public static class MyMapper extends TableMapper<Text, Text> {

  public void map(ImmutableBytesWritable row, Result value, Context context) throws
InterruptedException, IOException {
    // process data for the row from the Result instance.
   }
}
```

HBASE – MAP REDUCE READ/WRITE

JAVA

```java
public static class MyMapper extends TableMapper<ImmutableBytesWritable, Put>  {

  public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException,
InterruptedException {
    // this example is just copying the data from the source table...
      context.write(row, resultToPut(row,value));
    }

    private static Put resultToPut(ImmutableBytesWritable key, Result result) throws IOException
 {
      Put put = new Put(key.get());
      for (Cell cell : result.listCells()) {
        put.add(cell);
      }
      return put;
    }
}
```

JAVA

```java
Configuration config = HBaseConfiguration.create();
Job job = new Job(config,"ExampleReadWrite");
job.setJarByClass(MyReadWriteJob.class);    // class that contains mapper

Scan scan = new Scan();
scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
scan.setCacheBlocks(false);  // don't set to true for MR jobs
// set other scan attrs

TableMapReduceUtil.initTableMapperJob(
  sourceTable,      // input table
  scan,             // Scan instance to control CF and attribute selection
  MyMapper.class,   // mapper class
  null,             // mapper output key
  null,             // mapper output value
  job);
TableMapReduceUtil.initTableReducerJob(
  targetTable,      // output table
  null,             // reducer class
  job);
job.setNumReduceTasks(0);

boolean b = job.waitForCompletion(true);
if (!b) {
    throw new IOException("error with job!");
}
```

HBASE – MAP REDUCE READ/WRITE

JAVA

```
public static class MyMapper extends TableMapper<ImmutableBytesWritable, Put>  {

  public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException,
InterruptedException {
    // this example is just copying the data from the source table...
      context.write(row, resultToPut(row,value));
    }

    private static Put resultToPut(ImmutableBytesWritable key, Result result) throws IOException
 {
      Put put = new Put(key.get());
      for (Cell cell : result.listCells()) {
        put.add(cell);
      }
      return put;
    }
}
```

Instalando e utilizando o HBASE

RUNNING HBASE

- https://github.com/dajobe/hbase-docker
- docker-compose -f docker-compose-standalone.yml up –d
- docker images
- docker ps
- Docker run –name hbase-furb dajobe/hbase
- docker exec -it hbase-furb /bin/bash
- root@d1e23032d936:/# hbase shell
- docker cp italians.rb hbase:/tmp/italians.rb
- $ ./bin/hbase org.jruby.Main /tmp/italians.rb

Atividade

Marcio.Jasinski@senior.com.br