NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação

Master Program in Information Management

Data Lake em Viticultura

Big Data management na Agricultura

Francisco Miguel Marques Pires

Trabalho de Projeto apresentado como requisito parcial para obtenção do grau de Mestre em Gestão de Informação

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

DATA LAKE EM VITICULTURA

BIG DATA MANAGEMENT NA AGRICULTURA

por

Francisco Miguel Marques Pires

Trabalho de Projeto apresentado como requisito parcial para a obtenção do grau de Mestre em Gestão de Informação, Especialização em Gestão dos Sistemas e Tecnologias de Informação

Orientador: Miguel de Castro Simões Ferreira Neto

Outubro 2017

DEDICATÓRIA

Ao meu ***Avô*** Valdemar que nos abandonou mais cedo,
dedico todo o meu trabalho em sua memória.

AGRADECIMENTOS

Agradeço à minha família,

Aos meus pais,

À minha irmã,

Aos meus avós,

À minha namorada,

Agradeço ao professor Miguel de Castro Simões Ferreira Neto,

Ao André Barriguinha,

Todo o apoio e acompanhamento durante o meu percurso de desenvolvimento do projeto.

RESUMO

O volume de dados criado pelas inúmeras fontes tem aumentado a um nível visivelmente crescente. No âmbito de *Big Data*, o setor vitícola não é exceção, englobando uma grande quantidade de dados de diferentes tipos para análise, desde dados meteorológicos, dados sensoriais ou até dados do mercado de venda.

Com este projeto pretende-se implementar uma arquitetura moderna de dados cujo principal componente é o *Data lake.* A ideia base consiste em ter um único repositório com dados do setor vitícola, de forma a termos uma visão centrada dos dados numa única plataforma.

Este desafio consiste em identificar as diferentes fontes de dados necessárias para a elaboração do *data lake*, as suas características, incluindo a conceção de desenho e implementação da arquitetura, os processos existentes no *data lake*, e a exploração das tecnologias pertencentes ao ecossistema *Hadoop* que melhor se adaptam aos dados.

PALAVRAS-CHAVE

Big Data; Data Lake; Hadoop; Viticultura; Portugal

ABSTRACT

The volume of data created by innumerous sources has increased at an unprecedented rate. In Big Data, the wine sector is no exception, encompassing a large amount of data of different types for analysis, from meteorological data, sensory data or even wine market data.

With this project, is intended to implement a modern data architecture whose main component is a Data lake. The basic idea is to have a single repository with data from the wine sector, so that we have a focused view of the data on a single platform.

This challenge consists in identifying the different sources of data required for the elaboration of the data lake, its characteristics, including the design and implementation of the architecture, the existing processes in the data lake, and the exploitation of technologies belonging to the Hadoop ecosystem that best adapt to the data.

KEYWORDS

Big Data; Data Lake; Hadoop; Viticulture; Portugal

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

BI&A	*Business Intelligence e Analytics*
DAX	*Data Analysis Expressions*
DW	*Data Warehouse*
EDW	*Enterprise Data Warehouse*
GFS	*Google File System*
HDFS	*Hadoop Distributed File System*
HDP	*Hortonworks Data Platform*
IoT	*Internet of Things*
IVV	Instituto da Vinha e do Vinho, I.P.
JDBC	*Java Database Connectivity*
JSON	*JavaScript Object Notation*
NDVI	*Normalized Difference Vegetation Index*
NDWI	*Normalized Difference Water Index*
ODBC	*Open Database Connectitivty*
SGBD	Sistema de Gestão de Bases de Dados
SNIRH	Sistema Nacional de Informação de Recursos Hídricos
SQL	*Structured Query Language*
SSMS	*SQL Server Management Studio*
UDFs	*User-Defined Functions*
YARN	*Yet Another Resource Negotiator*

1. INTRODUÇÃO

O presente capítulo concede uma introdução genérica sobre o tema "*Data Lakes* em Viticultura" com o intuito de o enquadrar com as temáticas que o rodeiam. Inicialmente será realizado o enquadramento e identificação do problema, *framework* teórica, a relevância e importância do estudo bem como os objetivos titulados a este estudo.

1.1. *ENQUADRAMENTO E IDENTIFICAÇÃO DO PROBLEMA*

Com a corrente explosão tecnológica, a nossa sociedade tem gerado dados a uma taxa sem precedentes, aumentando a um ritmo nunca antes visto, emergindo das mais variadas fontes sem formatação específica, o que eleva o grau de complexidade em extrair, armazenar e manipular toda a informação que nos rodeia (Granite, 2015). Na Era digital, estes dados são conhecidos como *Big Data*.

Na indústria da agricultura, começam a sentir-se os primeiros efeitos de *Big Data*. Com a Internet das Coisas (*IoT*) o setor da agricultura tende também a ser informatizado, por exemplo, através da implementação de sensores no solo, da utilização de tratores e equipamentos capazes de gerar bastante informação em tempo real aos agricultores (Rijmenam, 2014).

Para gerir este tipo de informação, o principal objeto de investigação neste projeto consiste em identificar as fontes de dados e suas características, assim como os requisitos e a arquitetura necessária para implementar um *data lake* com dados provenientes da agricultura no setor da viticultura e enologia.

Por este tema ser recente e ainda alvo de investigações e desenvolvimentos, torna-o pouco utilizado por parte das organizações, pois existe ainda a necessidade de maturar, provar a usabilidade e demonstrar os benefícios de implementar uma arquitetura como um *data lake*.

Delineando cronologicamente os acontecimentos, na década de 70, o aparecimento do Sistema de Gestão de Base de Dados (SGBD) (Cood, 1969) foi desenvolvido para facilitar o acesso à informação, permitindo armazenar grandes volumes de dados. Na década seguinte, surgiu o Armazém de Dados, mais conhecido por *Data Warehouse* (*DW*), para fins de "*Reporting and Analytics*" (relatórios e desenvolvimentos analíticos), sendo a principal componente utilizada na área de *Business Intelligence e Analytics* (*BI&A*).

Na presença de *Big Data*, surge a necessidade de armazenar dados de forma diferente de como é armazenada atualmente. James Dixon, *CTO* do Pentaho, introduziu pela primeira vez, o conceito de *Data Lake* (Woods, 2011), uma nova arquitetura capaz de organizar e construir os sistemas da próxima geração. A arquitetura é denominada também por "*Modern Data Architetcure*" cujo componente principal é o *data lake* (O'Brien, 2015).

Com um *data lake*, os dados são todos carregados "*as-is*", ou seja, sem fazer qualquer transformação previamente. A ideia é unificar todos os dados da organização apenas num único local, evitando a separação dos dados em aplicações ou departamentos. Assim, os dados são facilmente disponibilizados a qualquer pessoa da organização e, posteriormente são moldados através de ferramentas consoante as necessidades para que estes tenham significado e valor. O *data lake* visa complementar os repositórios que as empresas já possuem, como por exemplo um *Enterprise Data Warehouse* (Teradata & Hortonworks, 2014).

A novidade e atualidade deste tópico é tão recente que os primeiros livros que defendem uma arquitetura sobre o *data lake* apenas surgiram em 2016. Um dos livros (*Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump)* pertence ao prestigiado autor Bill Inmon, considerado como o pai do *Data Warehouse* ("'The Father of Data Warehousing,'" 2007).

Sendo o termo *Big Data* cada vez mais aplicável a diversas organizações, o *data lake* tende a ser uma estrutura que futuramente será utilizada com frequência. Por isso, existe a necessidade de dar os primeiros passos para perceber e implementar um *data lake* num negócio em particular, com as devidas especificidades - o setor da viticultura.

Existem inúmeros dados no âmbito da agricultura que se encontram dispersos pelas mais diversas instituições e um dos principais benefícios do presente projeto consiste em disponibilizar uma infraestrutura de dados (*Data Lake*) onde fosse possível iniciar a sua concentração e posterior disponibilização.

O levantamento do estado da arte incluirá a investigação dos requisitos para construir um *data lake*; que tipos de dados existem para responder aos objetivos; as suas vantagens ao utilizá-los na agricultura; o que difere um *data lake* de um *DW*; e por fim, a arquitetura com uma visão genérica e detalhada.

Como a arquitetura dependerá das ferramentas utilizadas no *data lake*, será também investigado o *software* mais adequado para o implementar. Como hipótese existem três

grandes distribuidores da *framework Hadoop, Cloudera, HortonWorks* e *MapR*. O projeto visa utilizar uma das três ferramentas para pôr em prática o pretendido.

1.2. *FRAMEWORK TEÓRICA*

Os primeiros casos de sucesso da implementação de um *data lake* surgiram em empresas de grande dimensão como a *Google* ou a *Yahoo* (Teradata & Hortonworks, 2014). O principal objetivo destas empresas centrava-se em conseguir armazenar grandes volumes de dados, incluindo dados não estruturados e realizar transformações e modelos analíticos sobre os mesmos – o que foi possível devido às características de um *data lake*.

Com o aparecimento de *Big Data* em empresas de menor dimensão que até à data tinham apenas investido em *EDW's*, estas começam agora a investir em *data lakes* para complementar a sua arquitetura atual de dados (Woods, 2014). Através dos grandes distribuidores de *Hadoop*, a prática de implementar um *data lake* torna-se mais acessível para as organizações, sendo possível expandir o conceito em diversas áreas, existindo provas concretas dos benefícios que cada setor conseguiu atingir.

O projeto a implementar aplica técnicas semelhantes aos casos de uso referidos pelos distribuidores de *Hadoop*, mas numa área menos dominante como a viticultura.

A implementação de um *data lake* não segue qualquer diretiva específica (Teradata & Hortonworks, 2014) pois depende de vários fatores como a área de negócio, tipos de dados ou objetivos a serem cumpridos. Contudo, é possível delinear em traços gerais as etapas do processo. Inicialmente, será necessário reunir os dados das diversas fontes de dados. Após esta projeção, passaremos a identificar a arquitetura necessária que irá suportar todos os dados do setor vitícola. A arquitetura dependerá das ferramentas utilizadas no *data lake*, e da ferramenta base onde o *data lake* é executado. Os dados serão passíveis de transformações que serão efetuadas através de ferramentas que funcionam diretamente sobre a *framework Hadoop*, como por exemplo *NiFi* ou *Hive*. Finalmente, será disponibilizada a informação sob a forma de *dashboards* para a análise dos dados.

O contributo que se pretende alcançar com o projeto destina-se à partilha de conhecimento incidente na implementação de uma arquitetura moderna de dados, com tecnologias recentes que foram utilizadas por empresas e projetos de maior dimensão.

1.3. *RELEVÂNCIA E IMPORTÂNCIA DO ESTUDO*

Como referido no capítulo anterior, inicialmente, apenas empresas de grande dimensão é que adotaram uma arquitetura moderna de dados como o *data lake.* A razão principal fundamenta-se no facto de somente estas empresas possuírem novos tipos e grandes volumes de dados.

Depois de surgirem os principais distribuidores de *Hadoop*, empresas de menor dimensão mas com projetos relevantes, adotaram também os mesmos princípios para a gestão dos dados. Com o atual projeto, é importante seguir o mesmo conceito, embora o volume de dados não seja da mesma grandeza dos projetos referidos. No entanto, tal não representará um problema, porque um *data lake* tanto é aplicável para dados na ordem dos *MB* (*Megabytes*) ou *PB* (*Petabytes*) (Lang, 2014). Não estando em causa o volume de dados, e sendo maioritariamente dados estruturados provenientes do setor da viticultura, continua a ser importante aplicar o conceito neste setor menos dominante, pelas mais variadas razões.

A primeira razão centra-se no facto de serem dados incrementais, uma vez que existirá o crescimento contínuo de dados provenientes da meteorologia, preço dos vinhos, fatores de produção da vinha e do vinho, entre outros. Portanto, não tendo "*Big Data*" hoje, não significa que não venhamos a ter amanhã. Como segunda razão, em Portugal estas práticas ainda não são implementadas e, como tal existe a necessidade de impulsionar esta área de estudo com novos casos práticos, para que se incentive e estimule a descoberta de novo conhecimento. A terceira, advém de todos os benefícios que um *data lake* pode fornecer à organização ou, neste caso, ao projeto. A título de exemplo, existem os benefícios provenientes da *framework Hadoop*, que permitem que a solução seja rentável no âmbito de processamento e armazenamento distribuído (Swoyer, 2015). Com o *data lake* será possível capturar e armazenar dados de vários tipos em grande escala, transformar os dados, definir a estrutura dos dados apenas quando são usados - chamado "*Schema-on-read",* realizar novos tipos de processamento de dados e análises recaindo num tema em específico (Teradata & Hortonworks, 2014).

Com este projeto, extrair-se-á informação que até à data estaria "escondida" pois será possível analisar todo o histórico de dados e portanto, poderão estabelecer-se relações entre dados que adicionam valor à informação existente.

1.4. *OBJETIVOS DE ESTUDO*

Este projeto apresenta como objetivo final a disponibilização de um ponto de acesso aos dados do setor vitícola, utilizando uma arquitetura de dados moderna, recorrendo à *framework Hadoop.*

Espera-se contribuir através do desenvolvimento do *data lake* com novas descobertas para a comunidade científica desta área. Para o cumprimento deste objetivo, é necessário identificar as fontes de dados, as suas caraterísticas, e os requisitos necessários para desenhar a arquitetura de um *data lake* para que, posteriormente seja implementado.

Será igualmente importante identificar as fontes de dados que influenciam o setor vitícola, para cruzar diretamente as diferentes variáveis no repositório de dados a ser implementado. Inerente à questão principal do projeto, existem subquestões associadas como as seguintes: que tipos de dados existem e que são necessários (identificação e caracterização das fontes de dados), quais as vantagens em utilizar o *data lake* no setor vitícola; quais os requisitos necessários para implementar o *data lake*; quais as vantagens ao utilizar o *data lake* ao contrário de um *DW*; quais os processos existentes desde a entrada dos dados no *data lake* até à utilização por parte do utilizador final; qual o *software* que melhor se aplica para este tipo de projeto.

2. REVISÃO DA LITERATURA

Neste capítulo são analisados temas que irão possibilitar a construção de um *Data Lake*. Nomeadamente, o impacto de *Big Data* na viticultura, as diferenças de um *data lake* em relação ao tradicional *Data Warehouse*, requisitos necessários à implementação do *Data Lake*, toda a componente *Hadoop* e ferramentas associadas e os principais processos e componentes da arquitetura do *Data Lake*.

2.1. *BIG DATA NO SETOR DA VITICULTURA*

O setor vitícola é uma das áreas da agricultura que mais tem recebido atenção por parte de grandes organizações como a *IBM*, que recentemente lançou um caso de estudo sobre a adega - *E.&J. Gallo Winery* em *Napa Valley*, na Califórnia - Estados Unidos da América.

Com o auxílio do supercomputador *Watson* da *IBM*, foi possível utilizar um novo sistema de irrigação que permite utilizar menos água na rega das vinhas e aumentar o rendimento na colheita. Tal foi possível, ao aplicar análises avançadas sobre os dados, sob a forma de imagens satélite, combinado com tecnologias inovadoras no campo (IBM Corporation, 2014).

Este tipo de iniciativa permitiu poupar grandes quantidades de dinheiro na indústria da agricultura e na conservação da água (Vanian, 2016).

Existe também um projecto europeu a decorrer, denominado "*BigDataEurope Horizon 2020 project*" com características semelhantes ao pretendido com o atual projeto.

"*Horizon 2020*" é o maior projeto europeu de investigação e inovação com um capital perto dos 80 biliões, dos quais 64 milhões são dedicados à agricultura de precisão e à revolução digital no setor, enquanto 30 milhões são dedicados à implementação da *IoT* num projeto piloto "*Smart farming and food security*" (Michalopoulos, 2016).

No âmbito deste projeto, a autora *Maritina Stavrakaki*, defende que no setor vitícola está presente um dos quatro "*V's*" de *Big Data*, a variedade (Stavrakaki, 2016). Existe uma variedade de dados neste setor que está presente em todo o processo da produção do vinho. Este tópico foi também abordado no *workshop "Big data for food, agriculture and forestry: opportunities and challenges"* que decorreu em Paris, Setembro 2015.

Parte da dificuldade deste tema centra-se na uniformização dos dados e nos formatos existentes que estes podem adquirir e, no fim do processo retirar informação útil que possa influenciar decisões estratégicas e operacionais por parte da organização. A complexidade passa por perceber a dificuldade inerente à recolha de informação e de que forma é possível manipular os dados obtidos para uma fácil interpretação e visualização da informação.

Também em França não passa despercebida a este tema, sendo um dos maiores produtores mundiais de vinho e, com o aparecimento de empresas nos anos 2000 dedicadas ao processamento de dados na agricultura, surgiu a *Smag* – uma das maiores empresas francesas fornecedoras de Sistemas de Informação na agricultura.

As tecnologias da *Google* associadas ao termo *Big Data* para o processamento de grandes volumes de dados, são tidas como vantagem por parte de empresas como o *Facebook*, pois possuem uma grande quantidade de dados para análise, algo que no setor vitícola pode parecer não tão abundante e por essa razão este tipo de técnicas ainda são pouco usadas. Segundo o CEO da *Smag* "*Facebook génère plus de données en deux jours, que nous en un an. Avec W millions d'hectares, nous commençons tout juste à avoir besoin de ces technologies" ("O Facebook gera mais dados em dois dias do que nós num ano. Com 10 milhões de hectares, estamos apenas a começar a necessitar destas tecnologias") (Agro Presse Hebdo, 2015, p. 1).* Contudo, existe um aumento em número dos dados produzidos por maquinarias, como tratores, pulverizadores, ceifeiras, e também dados provenientes de sensores incorporados no campo, como sensores de stress hídricos, ou sensores aéreos como *drones* e imagens satélites (MR, 2015).

Atualmente, já existem plataformas de informação pagas onde é possível os agricultores introduzirem os dados sobre as suas colheitas e obterem as mais variadas análises que a plataforma oferece. A título de exemplo, a empresa de maquinaria agrícola e pioneira em *Big Data* na agricultura, *John Deere*, criou recentemente um portal que permite aos agricultores acederem aos dados produzidos pelas suas máquinas, permitindo observar vários indicadores, desde a manutenção das máquinas, visões dos campos, até ao planeamento de tarefas. A tabela 1 demonstra algumas das funcionalidades da aplicação.

Máquinas	Terrenos	Trabalhos
Diagnóstico e manutenção remota	Radar meteorológico	Planificação geral dos trabalhos
Histórico de Manutenção	Visão geral dos campos	Controlo do tempo otimizado
Sistema JDLink (telemática)	Gestor de ficheiros de configuração	Otimização de frotas
Localização/Histórico das localizações	Partilha das linhas de guiamento	Relatórios para faturação
Diversas opções de transferência de dados	Analisador de terreno	Resumo executivo das operações diárias

Tabela 1 - Funcionalidades *MyJohnDeere*

Portugal é também um dos países com uma presença forte no setor vitícola, estando representado no topo de indicadores como a exportação, produção e consumo. Faz todo o sentido apostar na evolução das tendências atuais sobre o setor, para que possamos ser também um dos líderes ao tirar partido das mais recentes tecnologias e revolucionar as metodologias utilizadas nos processos de produção de vinho.

Segundo os dados mais recentes do Instituto da Vinha e do Vinho, I.P., Portugal foi o sétimo maior exportador mundial em 2014 e com taxas de crescimento anuais positivas desde 2010, esperando-se ainda um aumento de cerca de 8% na produção do vinho (Machado, 2015). A tabela 2 ilustra os valores de exportação e importação de Portugal assim como a variação em percentagem do crescimento nas duas vertentes.

Intra + Extra UE	2007	2008	2009	2010	2011	2012	2013	2014
Exportações	595.987	575.966	544.011	614.380	656.918	707.458	720.794	725.487
Δ(%)		-3.4%	-5.5%	12.9%	6.9%	7.7%	1.9%	0.7%
Importações	63.257	80.363	79.099	89.493	84.435	122.572	122.399	125.804
Δ(%)		27,0%	-1,6%	13,1%	-5,7%	45,2%	-0,1%	2,8%

Tabela 2 - Portugal - Exportações e importações de vinho (x1000€)
Fonte: Adaptado (IVV/INE)

Em Abril de 2016, Évora acolheu uma iniciativa bastante importante no tema que concerne à modernização do setor Vitícola, o primeiro *workshop* em Portugal de "Viticultura Inteligente".

O *workshop* contemplou várias palestras no âmbito da viticultura de precisão e apresentações de projetos europeus com tecnologias inovadoras neste campo. Com esta iniciativa, pretende-se sensibilizar os empresários agrícolas e público em geral, sobre o tema da agricultura inteligente suportada pela gestão de informação que pode trazer ganhos de eficiência na utilização de recursos, alargando o conceito de precisão, em que uma empresa agrícola passa por um processo de transformação digital, em que os dados são convertidos em informação - essencial ao suporte na tomada de decisões ("CCDR-A com êxito no POR; Alentejo consegue 100% de absorção dos fundos," 2016, p. 10).

Os exemplos descritos são mais uma prova que deverá emergir uma nova arquitetura de dados onde seja possível enfrentar as atuais dificuldades da complexidade e variedade dos dados provenientes do setor vitícola. Os dados demonstrados pelo IVV são também um excelente indicador para fazer iniciar esta aposta em Portugal.

2.2. *DEFINIÇÃO DE DATA LAKE*

O termo adjacente à nova arquitetura moderna de dados é o *Data Lake*. Este conceito é uma componente fundamental que permite complementar os atuais repositórios de dados das organizações, armazenando toda a informação do negócio, podendo lidar com vários tipos de dados, usando tecnologias inovadoras como a *framework Hadoop*. O *data lake* é a evolução das arquiteturas de dados existentes.

Esta evolução provém da necessidade de responder aos novos desafios de *Big Data*, não só no aumento exponencial dos dados, comprovado pela IDC/EMC (IDC/EMC, 2014), mas da vasta variedade que tem surgido nos últimos anos.

O *data lake* na viticultura é uma inovação na gestão de dados que promete reduzir os custos de processamento e armazenamento e permite efetuar novos tipos de análise que outrora não era possível com os tradicionais *Data Warehouses*.

Um *Data Lake*, normalmente mas não necessariamente, é implementado em *Hadoop*, pois é um sistema de ficheiros distribuído que pode acomodar dados de diferentes formas e tamanhos (Swoyer, 2015). Todas as suas definições estão relacionadas diretamente com o *Apache Hadoop* e todo o seu ecossistema de projetos *Open Source*, o que deriva da

pergunta principal - como construir um *Data Lake* usando as potencialidades *Hadoop* (Teradata & Hortonworks, 2014).

As organizações que usam o *Data Lake*, visam complementar e expandir os seus processos existentes de gestão de dados, de forma a tirar partido de novas análises que podem fazer a dados que antes não tinham qualquer valor ou significado.

Em suma, é um repositório bastante flexível e escalável capaz de receber todo o tipo de dados, estruturados ou não estruturados, por um preço bastante baixo comparativamente aos atuais repositórios.

O seguinte capítulo introduz as principais diferenças entre um *Data Lake* e um *Data Warehouse*.

2.3. *DATA LAKE VS DATA WAREHOUSE*

Para comparar efetivamente as características de um *Data Lake*, é utilizado frequentemente o *Data Warehouse*. Não só porque é uma arquitetura de dados mas também porque é o repositório mais comum usado em empresas que disponibilizam "*Business Intelligence*". A tabela 3 ilustra as principais características de ambas as arquiteturas.

Dimensões	***Enterprise Data Warehouse***	***Data Lake***
Utilizadores	Centenas até milhares de utilizadores em concorrência – Utilizadores de empresas	Alguns utilizadores – Cientistas de dados
Workload	Processamento *Batch*	Processamento *Batch* em larga escala
Esquema	***Schema on Write* –** esquema definido antes dos dados serem armazenados, para posteriormente serem escritos	***Schema on Read* –** esquema definido depois dos dados serem armazenados
Escala	Pode ser expansível para grandes volumes de dados mas com um custo moderado	Desenhado para obter um baixo custo de armazenamento
	Os dados são acedidos através de	Os dados são acedidos

Métodos de acesso	SQL e ferramentas de BI, suportadas por sistemas de *reporting* e *analytics*	através de programas desenvolvidos por programadores, sistemas baseados em SQL e outros métodos
SQL	ANSI SQL, Propriedades ACID	Programação flexível envolvendo SQL
Dados	Dados estruturados/ transformados	Dados "*as-is*" sem qualquer transformação. Estruturados, semiestruturados, não estruturados
Acesso	*Seeks*	*Scans*
Complexidade	*Joins* complexos	Processamento complexo
Segurança	Madura	Em desenvolvimento
Custo/Eficiência	Uso eficiente de CPU/IO	Baixo custo de armazenamento e processamento

Tabela 3 - Diferenças entre um *DW* e um *Data Lake*
Fonte: Adaptado de *Putting the Data Lake to Work: A Guide to Best Practices (Hortonworks & Teradata)*

As característas diferem entre os dois repositórios, mas mais uma vez, é importante salientar que o *Data Lake* não deve substituir um *Data Warehouse*. Ambos são diferentes e devem ser usados para aquilo que foram desenhados. Devem agir os dois em conformidade, complementando-se, pois ambos têm pontos fortes e fracos. Tal como a Figura 1 demonstra, se a organização necessitar de requerimentos que se encontram entre ambas as soluções, entra-se numa dicotomia de escolhas entre as duas arquiteturas.

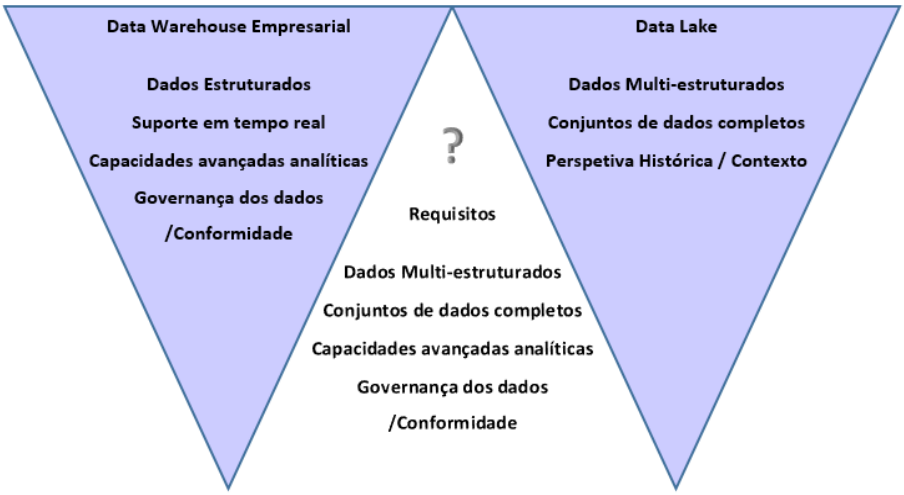


Figura 1 - Ambas as soluções não vão de encontro com os requerimentos necessários
Fonte: Adaptado de: *The Contextual Data Lake (SAP)*

A solução recai sobre um modelo híbrido – Figura 2 – que preenche as lacunas nos requerimentos que o negócio tem, sendo o modelo constituído pelos dois repositórios (*The Contextual Data Lake*, 2015). Este modelo expande o valor do tradicional *DW* combinando os dados não estruturados com os estruturados, e o acesso total a todos os dados existentes, tanto no *DW* como no *Data Lake*. Com o modelo híbrido é possível atingir um maior nível de eficiência ao distribuir a carga de processamento e otimização dos dados, como a transformação e integração.

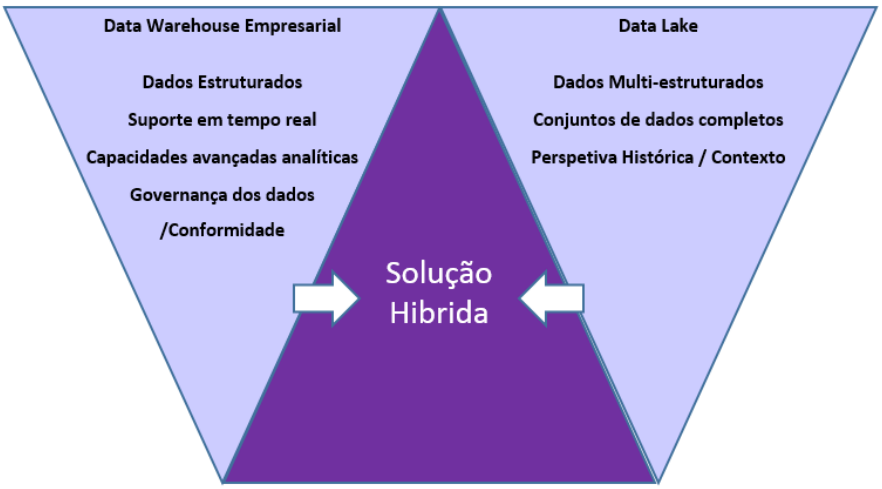


Figura 2 - Solução híbrida que preenche as lacunas
Fonte: Adaptado de: *The Contextual Data Lake (SAP)*

A par das características de um *Data Lake*, as potencialidades desta arquitetura podem ser definidas através das seguintes capacidades (Lang, 2014), (*The Contextual Data Lake*, 2015), (O'Brien, 2015):

- Armazenar e processar dados em grande escala por um baixo preço;
- Armazenar vários tipos de dados no mesmo repositório;
- Permite transformar os dados aliviando o *ETL* feito no *DW*;
- Permite aceder a todos os dados, desde os originais até aos transformados;
- Permite alimentar diretamente um *DW*;
- Definir a estrutura dos dados no momento em que são usados – o denominado *Schema on Read*, ideal para dados não estruturados ou semiestruturados cujo seu valor ainda é desconhecido;
- Permite realizar novos tipos de processamento sobre os dados;
- Permite realizar análises que numa base de dados relacional não era possível, como análises não relacionais, baseadas em grafos ou análises de rede (*"Network Analysis"*);
- Realizar análises a casos de uso em temas específicos.

Com a comparação realizada entre as arquiteturas, foi possível descrever as potencialidades/benefícios de ambas, sendo a próxima etapa vital identificar os requisitos necessários à implementação do *Data Lake*.

2.4. *REQUISITOS DA IMPLEMENTAÇÃO DO DATA LAKE*

Nos pontos anteriores foi revelada a importância de *Big Data* no setor vitícola e os benefícios que um *Data Lake* pode ter com dados provenientes da viticultura. Considerando que uma iniciativa em *Big Data* e que uma arquitetura baseada em *Hadoop* representa um valor para a organização e que possui um conjunto de benefícios ao complementar os repositórios atuais existentes, a fase seguinte consiste em identificar os requisitos para a construção de um *Data Lake*, a ser utilizado como protótipo.

Para que a construção seja possível, a sua arquitetura está assente na *framework Hadoop*, o que nos leva ao primeiro requisito que consiste em identificar um dos distribuidores da *framework* como uma solução *Open Source*. Entre as possibilidades temos as três mais conhecidas tecnologias, *Cloudera*, *Hortonworks* e *MapR*.

Tendo em conta que no *Data Lake* existem vários processos, desde a captura de dados até ao seu tratamento e disponibilização de informação, estarão também associadas, inevitavelmente, outras tecnologias em *Apache* relacionadas com o *Hadoop* para uma própria gestão dos dados. Como nenhum dos distribuidores é capaz de oferecer uma ferramenta que permita uma gestão e integração customizada é necessário adicionar outras tecnologias para que seja possível retirar o máximo valor possível de um *Data Lake*. Tal só é suscetível de acontecer, conhecendo a natureza dos dados para que seja escolhida a tecnologia mais adequada ao negócio. Identificadas todas as ferramentas necessárias, sendo em *Apache*, ou tecnologias externas (como por exemplo para *reporting*) é possível fazer uma estruturação da arquitetura. A Figura 3 ilustra o ecossistema *Hadoop* e os seus projetos *Open Source* relacionados, dedicados à computação distribuída e ao processamento em grande escala.

Figura 3 - Ecossistema *Hadoop* e projetos relacionados

Uma vez utilizada a *framework Hadoop*, será necessário perceber o seu funcionamento, nomeadamente o seu sistema de ficheiros *HDFS (Hadoop Distributed File System)* e a camada de processamento efetuada pelo *MapReduce*.

Os próximos requisitos estão associados aos processos existentes no *Data Lake*. São requisitos e também boas práticas que devem ser implementadas em projetos reais.

No âmbito de *"Data Governance"* – uma área que cobre temas como a segurança e privacidade, qualidade dos dados, monitorização, usabilidade, integração e disponibilidade

serão requisitos primários a ter em conta numa estrutura como um *Data Lake*. Sendo os dados um componente chave, não faria sentido disponibilizar dados que não fossem coerentes ou corretos pois originaria um mal entendimento nos passos seguintes de exploração e análise. Tanto a vertente "*Governance*" como a segurança dos dados, não são aditivos na implementação, nem opcionais (Swoyer, 2015).

Swoyer defende ainda que ter uma ferramenta de gestão de metadados é um dos componentes mais importantes que deve existir num *Data Lake*. A título de exemplo se existirem milhões de ficheiros, como é que um analista encontra um único ficheiro se não tiver os caracteres exatos do mesmo? Uma ferramenta de metadados permite ter controlo do histórico de um ficheiro, como por exemplo, conhecer a sua proveniência, os seus movimentos ao longo do tempo, e todos os processos que incidiram sobre os dados.

No que diz respeito à segurança e privacidade, é uma componente importante, que ainda se encontra em desenvolvimento. Alguns dos pilares da segurança como administração, autenticação, autorização e proteção dos dados deverão ser tidos em conta numa missão que visa a proteção da informação, um recurso valioso do *Data Lake*.

2.5. *IDENTIFICAÇÃO DO DISTRIBUIDOR HADOOP*

Como referido no capítulo anterior, escolher um distribuidor *Hadoop* é um requisito necessário para implementar um *Data Lake*.

O crescimento *Hadoop* originou diferentes versões nas ferramentas relacionadas, o que dificulta a compatibilidade das ferramentas com o próprio *Hadoop* (Dirk deRoos, Paul Zikopoulos & Rafael Coss, 2014). Esta dificuldade surge ainda mais acentuada caso o utilizador escolha o percurso de compilar os projetos diretamente do *Apache*. Assim surgem os distribuidores, sendo um caminho alternativo atraente, pois contribuem para a resolução deste problema apresentando os vários componentes num produto apenas, como um pacote que inclui não só o *Hadoop* mas outros projetos como o *Hive*, *HBase*, *Pig*, entre outros (Turkington, 2013).

Listando os distribuidores mais importantes e influenciadores do mercado, temos:

- ***Cloudera Distribution for Hadoop*:** Considerado um dos melhores distribuidores do mercado, *Cloudera* foi o primeiro distribuidor comercial de *Hadoop*, e também um grande contribuidor de todo o ecossistema. Contém diversos projetos *Apache* além do *Hadoop*, como *Hive*, *Pig*, *Hbase* e outros menos conhecidos como *Mahour* e *Whir*. Os pacotes disponíveis para *download* oferecem serviços mais, ou menos completos, dependendo do pacote que o utilizador escolher. *Cloudera Manager* é um exemplo de um pacote completo, que funciona com uma licença anual, onde o utilizador pode usufruir dos demais projetos descritos com suporte técnico incluído. No entanto, é possível fazer *download* de uma versão menos completa do *Cloudera* que não tem encargos monetários e permite usufruir algumas das funcionalidades que o pacote completo oferece.

  Em 2013, *Cloudera* anunciou que iria focar-se em adicionar apenas componentes adicionais no topo da *framework Hadoop*, para ser um distribuidor diferenciador do mercado (Turkington, 2013).
- ***Hortonworks Data Platfrom*:** *Hortonworks* é outro grande distribuidor com factos impressionantes. Tem a maior comunidade de "*committers*" (têm o poder de aprovar alterações de código nos projetos *Apache*) e contribuidores de código nos componentes do ecossistema *Hadoop* (Dirk deRoos, Paul Zikopoulos & Rafael Coss, 2014). *Hortonworks* surgiu em 2011, tendo sido

criado por uma equipa da Yahoo, que participou no desenvolvimento do *Hadoop*. Assim, surgiu a plataforma *HDP - Hortonworks Data Platform* que faz concorrência ao *CDH - Cloudera Distribution for Hadoop*. Ambas são similares, mas apresentam algumas diferenças, demonstradas na figura 4. Tal como a *Cloudera*, o *Hortonworks* também é um contribuidor na comunidade *Open Source*, com os seus esforços de novos desenvolvimentos. Respetivamente aos serviços que oferece, o pacote que inclui o *HDP* é inteiramente gratuita, tendo apenas serviços profissionais e de suporte pagos para a plataforma. No entanto, não vende *software* e contribui com soluções entre a comunidade *Open Source* (Turkington, 2013).

- ***MapR Technologies:*** Também este distribuidor destaca-se entre as outras opções sendo uma tecnologia diferente com os seus pontos fortes e fracos. *MapR* é também baseado em *Hadoop* com algumas modificações e melhoramentos. Foi considerado o melhor distribuidor *Hadoop* que fornece uma total proteção dos dados, com significativas vantagens fáceis de usar (Dirk deRoos, Paul Zikopoulos & Rafael Coss, 2014). Foi também o primeiro distribuidor a oferecer soluções *high-availability* (*HA*, capacidade do sistema continuar a funcionar na ocorrência de falhas imprevistas no sistema) para o *NameNode*, *JobTracker* e *HBase*. *MapR* oferece também integração com o sistema de ficheiros *NFS* e *POSIX*, sendo um melhoramento em relação ao seu atual sistema de ficheiros. Semelhante ao *CDH*, *MapR* tem um pacote (também conhecido como M3) livre de encargos mas que apenas tem algumas das funcionalidades que o pacote empresarial apresenta. A versão M7 é a mais completa, incluindo tecnologias como *HBase, Pig, Hive, Sqoop, Mahout* e muitas outras (Turkington, 2013).

Além dos três principais distribuidores listados, existem também outros que no entanto são menos referenciados, como a *Pivotal* da empresa *EMC*, *InfoSphere BigInsights* e *PureData System for Hadoop* da *IBM* e ainda a *Intel Distribuition* da *Intel.*

A figura seguinte demonstra as características das três plataformas mais conhecidas no mercado, onde podemos concluir que existem mais similaridades do que diferenças.

	Hortonworks	Cloudera	MapR
Performance e Escalabilidade			
Receção de dados	Batch	Batch	Batch e escrita em streaming
Arquitetura Metadados	Centralizado	Centralizado	Distribuído
Performance HBase	Picos de Latência	Picos de Latência	Baixa Latência frequentemente
Aplicações NoSQL	Principalmente aplicações Batch	Principalmente aplicações Batch	Batch e aplicações online/tempo real
Dependências			
Alta Disponibilidade	Recuperação de falhas única	Recuperação de falhas única	Auto recuperação perante múltiplas falhas
MapReduce (AD)	Reinicializar jobs	Reinicializar jobs	Contínuo, sem reinicializar
Upgrading	Upgrades parando a infraestrutura	Upgrades sem parar infraestrutura	Upgrades sem parar infraestrutura
Replicação	Dados	Dados	Dados + Metadados
Snapshots	Consistente só para ficheiros fechados	Consistente só para ficheiros fechados	Consistência Point-in-time para todos os ficheiros e tabelas
Recuperação de desastres	Não	Agendamento de cópia de ficheiros	Espelhamento
Capacidade de gestão			
Ferramentas de gestão	Ambari	Cloudera Manager	Sistema de controlo MapR
Suporte Volume	Não	Não	Sim
Heat Map, Alarmes,Alertas	Sim	Sim	Sim
Integração com REST API	Sim	Sim	Sim
Data and Job Placement Control	Não	Não	Sim
Acesso Dados			
Acesso a sistema de ficheiros	HDFS, leitura NFS	HDFS, leitura NFS	HDFS, leitura/escrita NFS (POSIX)
Ficheiros I/O	Apenas Incremental	Apenas Incremental	Leitura/Escrita
Segurança: (LCA)	Sim	Sim	Sim
Autenticação Wire-Level	Kerberos	Kerberos	Kerberos, Nativo

Figura 4 - *Hortonworks* VS *Cloudera* VS *MapR*
Fonte: Adaptado de Hadoop Buyer's Guide (Schneider, 2013)

Em suma, a plataforma "ideal" escolhida não o será pelas suas características pois qualquer uma delas cumpre os requisitos do projeto, mas será escolhida face aos serviços que oferece. A distribuidora *Cloudera* poderia ser uma opção caso o pacote *Cloudera Manager* não tivesse encargos. A versão gratuita do *MapR* não suporta algumas das funcionalidades como o caso das soluções *High-availability* que representam uma das vantagens em relação aos outros distribuidores. Sendo o *HDP* a única plataforma que no seu pacote gratuito concede mais funcionalidades e o acesso total à plataforma, o *Hortonworks* será a plataforma utilizada para a construção do *Data Lake*.

2.5.1. *Apache Hadoop*

Em 2003 e 2004 foram lançados dois *papers* académicos pela *Google*, que revelam o sistema de ficheiros distribuído da *Google* (*GFS*) e a implementação do *MapReduce* respetivamente. Ambos os projetos consistiram numa plataforma de processamento de dados em grande escala e altamente eficiente. No entanto o *Hadoop* foi criado por *Doug Cutting*, criador do projeto *Apache Lucene*, que consistia num motor de busca avançado *Open Source*. Assim surgiu *Hadoop* com as suas origens no projeto *Apache Nutch* que por sua vez faz parte de *Apache Lucene* (White, 2015).

Em 2008 *Doug Cutting* juntou-se à *Yahoo* que anunciou o lançamento da maior aplicação de produção do *Hadoop.* Também neste ano, *Hadoop* fez parte do projeto *Apache*, e nesta altura já era utilizado por grandes empresas como o *Facebook* ou o *New York Times* (White, 2015).

Para melhor entendimento do funcionamento da *framework Hadoop*, serão descritas as componentes que derivaram do *GFS* e *MapReduce*.

2.5.2. *Hadoop Distributed File System - HDFS*

O *Hadoop distributed file system* (*HDFS*) é um sistema de ficheiros distribuído com alta tolerância a falhas que possam ocorrer, desenhado especificamente para acomodar grandes volumes de dados através de *clusters* espalhados em diversas máquinas (também denominadas por *commodity hardware/machines*).

Quando um determinado *dataset* (conjunto de dados) supera a capacidade de armazenamento da máquina, é necessário particionar o conjunto de dados através de várias máquinas, surgindo assim o conceito de sistema de ficheiros distribuídos, responsável pela gestão de armazenamento de uma rede de máquinas.

A arquitetura implementada pelo *HDFS* é intitulada de *master/slave* em que operam dois tipos de nós, um *NameNode (master)* e vários *Datanodes (workers)* (White, 2015). O papel principal do *NameNode* é gerir o espaço no sistema de ficheiros, e gerir metadados resultantes de modificações no sistema de ficheiros. Os *DataNodes* são os "escravos" do sistema de ficheiros, e armazenam blocos de dados sob instrução do *NameNode*. Por outras palavras, todos os dados do utilizador passam pelos *DataNodes* e são armazenados em blocos, que por sua vez constituem um ficheiro, guardado fisicamente no sistema de ficheiros do *Hadoop*. O *NameNode* controla os *DataNodes* e sabe a localização dos blocos que constituem um ficheiro, devido ao *BlockReport* enviado periodicamente ao *DataNode*.

Existe um mecanismo que permite aos *DataNodes* enviarem um "*Heart Beat Signal*" ao *NameNode*. Assim, o *NameNode* sabe que pode contar com esse *DataNode* e portanto continua ativo para ser utilizado. Este mecanismo faz parte da característica de tolerância de falhas do *Hadoop*. Assim que um *DataNode* deixa de estar ativo, o *NameNode* replica os blocos contidos nesse *DataNode.* Por norma, o *HDFS* replica cada bloco de dados e é armazenado tipicamente em três máquinas separadas. Os mecanismos de segurança para eventuais falhas são um conceito bastante presente no *Hadoop*, visto que a falha é a norma em vez da exceção (Borthakur, 2013).

A figura 5 descreve a arquitetura do sistema de ficheiros distribuído do *Hadoop*, demonstrando a interação entre o *NameNode*, o cliente e o *DataNode*.

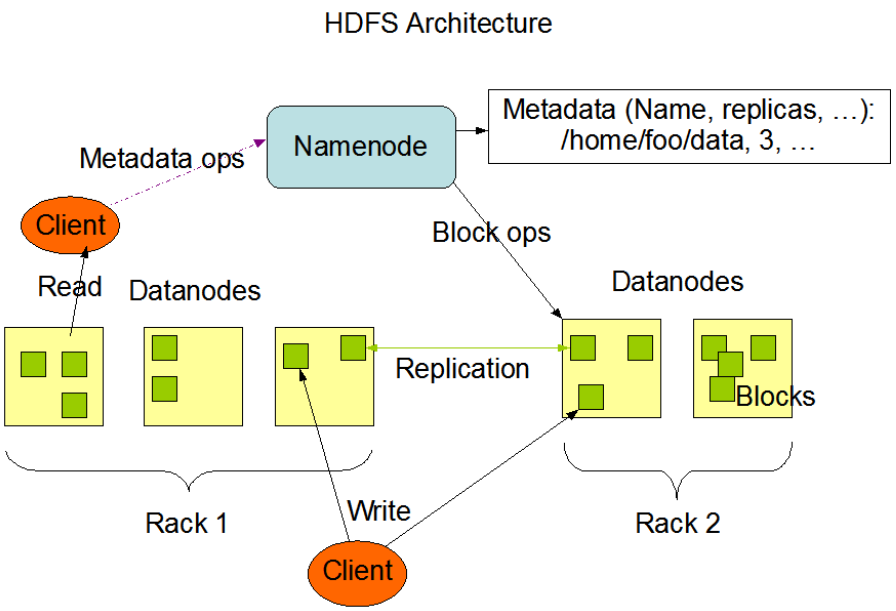


Figura 5 - Arquitetura do *HDFS*
Fonte: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#NameNode+and+DataNodes

2.5.3. MapReduce

Descrita a essência como o *Hadoop* armazena os dados, resta saber como os processa. Segundo a filosofia do *Hadoop "Moving Computation is Cheaper than Moving Data".* Esta assunção contraria a tradicional forma de processar dados, que concerne na sua movimentação até à aplicação, onde ocorre o processamento, congestionando a rede (Borthakur, 2013). Contudo, a abordagem do *Hadoop* é mais eficiente se executarmos a computação onde se encontram os dados, evitando a sua movimentação e portanto diminuindo o congestionamento de rede e aumentando o fluxo do sistema. Esta abordagem tem um impacto maior no paradigma do *Hadoop* e na presença de *datasets* de grandes volumes.

Como funciona?

Existem duas fases de processamento no *MapReduce*:

- *Map Phase* – nesta fase os dados são recebidos pelo componente "*Input Splitter*" que divide o *input* em conjuntos de pares chave-valor ("*key-value pairs"*). Os conjuntos são transformados pelo mapa gerando novos pares chave-valor.
- *Reduce Phase* – processa todos os conjuntos chave-valor provenientes da fase anterior, agrupados por chave (todos os pares com a mesma chave são atribuídos ao mesmo "*reducer"*), gerando novos pares chave-valor. O *output* é escrito fisicamente no disco.

Ambas as fases podem ser programadas através de funções, permitindo ao utilizador parametrizar o seu *input*. Existem passos intermédios opcionais utilizados para otimizar o fluxo de processamento, como por exemplo adicionar um "*combiner*" – uma fase redutora intermédia. Um *combiner* pode ser útil no caso em que uma tarefa mapa demore mais tempo que as outras tarefas mapas, porque neste caso só se começa a fase redutora quando todas as tarefas mapas tiverem sido concluídas. Se adicionarmos um *combiner* podemos ir agregando os conjuntos chave-valor ao mesmo tempo que a tarefa morosa está a ser executada. Com este passo reduzimos o trabalho do *reducer* poupando tempo de processamento. Outra função opcional e também para efeitos de otimização é o "*partitioner*" ou "*shuffler*" cujo papel principal é decidir que pares vão para cada *reducer*. A figura 6 ilustra, sumariamente, o fluxo do processamento do *MapReduce*.

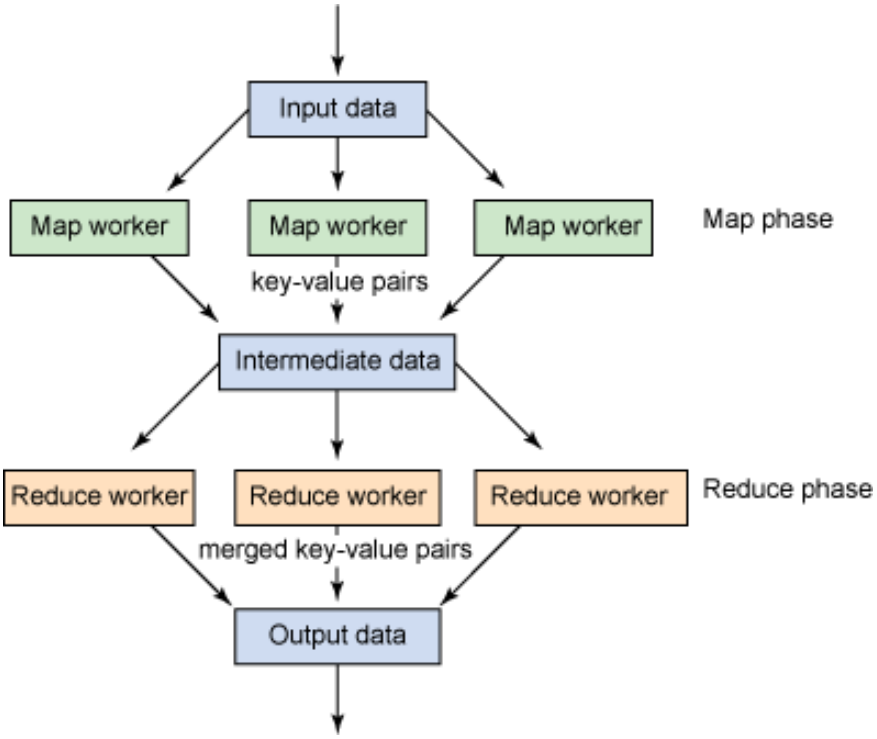


Figura 6 - Vista simplificada do processamento *MapReduce*
Fonte: http://www.ibm.com/developerworks/library/l-hadoop-3/

3. PROCESSO DE CONSTRUÇÃO DO DATA LAKE

As metodologias utilizadas englobam um conjunto de técnicas que se iniciam na identificação das fontes de dados que servirão de fonte na informação que será disponibilizada no *Data Lake*. Depois da identificação, é possível traçar um primeiro esboço do desenho da arquitetura do *data lake*. Após o desenho, são explicadas as metodologias de extração e técnicas de transformação dos dados. Estas técnicas são mapeadas com determinadas tecnologias dentro do *data lake*, e portanto, o desenho da arquitetura será refinado com mais detalhe na fase final dos processos envolvidos no *data lake*.

3.1. *ENQUADRAMENTO SETOR VITÍCOLA*

Portugal é um país rico no setor vitivinícola, tendo uma grande importância, não só através das receitas geradas pelo mercado, mas também pelo turismo envolvente nesta área, e em termos ambientais.

Os dados utilizados no projeto, provêm das várias etapas até à fase final do produto. Numa primeira fase existem os dados ambientais das culturas das vinhas. Esse tipo de dados como as características das vinhas, do solo, a enxertia, a poda, a empa ou as regas, não foram facultados. No entanto os dados da segunda fase mais operacional, foram utilizados no projeto, no que diz respeito à produção. Dados de controlo das condições em que a vinha se encontra, são capturados através de uma diversificação de sensores instalados no solo e nas plantas, que permitem deduzir uma quantidade de informação vasta, seja do solo, da planta ou do ambiente. Para complementar os dados sensoriais, foram utilizados dados meteorológicos para futuros relacionamentos de métricas, e dados satélite que permitem analisar a área territorial das castas, usufruindo de certos índices como índice de vegetação e de água no solo, bastante utilizados na agricultura.

Os dados da fase final respeitante ao produto, como por exemplo a quantidade de vinhos que uma casta produziu, características do lote, que vinhos foram produzidos, não foram fornecidos e não foi possível criar uma ligação direta entre os dados operacionais e o mercado de venda. Uma análise interessante seria relacionar os dados operacionais com o preço dos vinhos produzidos no mercado no período de análise utilizado. Não existindo essa possibilidade, foram utilizados dados do mercado de vendas atual, incluindo potenciais vinhos produzidos pelas castas Touriga Nacional e Aragonez na Herdade do Esporão.

3.2. *IDENTIFICAÇÃO E RECOLHA DE DADOS*

3.2.1. Innovine

O primeiro *dataset* corresponde a dados sensoriais de castas situadas da Herdade do Esporão, facultados pelo projeto europeu *Innovine*. Este projeto foi iniciado em 2013, tendo a duração de 4 anos e envolvendo diversos países europeus, incluindo Portugal. O Innovine tem como diretriz apoiar o setor vinícola, num cenário de alterações climáticas. Um cenário previsível que pode afetar o equilíbrio entre a produção e as variedades de uva, e também impactando negativamente as castas devido a pestes e doenças (Anne-Françoise Adam-Blondon, 2013).

A Herdade do Esporão, fundada em 1973, faz parte da histórica cidade de Reguengos de Monsaraz no Alentejo, tendo 1830 hectares, dos quais 450 hectares pertencem a vinhas. Segundo os dados fornecidos no relatório de 2015 da Herdade do Esporão, existem cerca de 194 variedades de vinhas plantadas e 37 em produção, correspondendo estas, às castas que tiveram uma melhor adaptação à região do Alentejo (*Esporão SA Relatório 2015*, 2015). Uma das castas que é utilizada neste projeto, a Touriga Nacional, é inclusivamente considerada "*ex-líbris* do Alentejo" (*Esporão SA Relatório 2015*, 2015). A outra casta utilizada também conhecida é a Aragonez.

Existem dados para estas duas castas, provenientes de sensores colocados no solo e na planta. Alguns exemplos dos dados que estes sensores recolhem são as temperaturas ao nível das folhas, dos bagos, o diâmetro dos troncos, sensores do fluxo de seiva e da folha molhada. A nível meteorológico temos como exemplo, sensores que medem a velocidade do vento, a temperatura e humidade do ar e precipitação.

Para melhor identificação da geolocalização da Herdade do Esporão e das castas que são alvo de estudo foram criados com o auxílio do *Google Earth* vários polígonos que permitem fazer uma aproximação muito real da área que a herdade do esporão e as castas ocupam.

A área denotada a verde na figura 7, ilustra toda a região da Herdade do Esporão referente a todas as vinhas. É constituído por dois polígonos perfazendo um total de 194 pontos com um tamanho global de 507 hectares. A esta área, retirando os polígonos a laranja, que correspondem a regiões que não são vinhas, como por exemplo caminhos de terra e infraestruturas circundantes pertencentes à Herdade, temos um total de 411 pontos constituídos por seis polígonos, e que perfazem um total de 24,76 hectares. Subtraindo a

região a verde pela laranja, ficamos com um total de 482,84 hectares, que é um número semelhante ao relatório emitido pela Herdade do Esporão em 2015. A casta Touriga Nacional tem 5,3 hectares e a Aragonez 5,6.

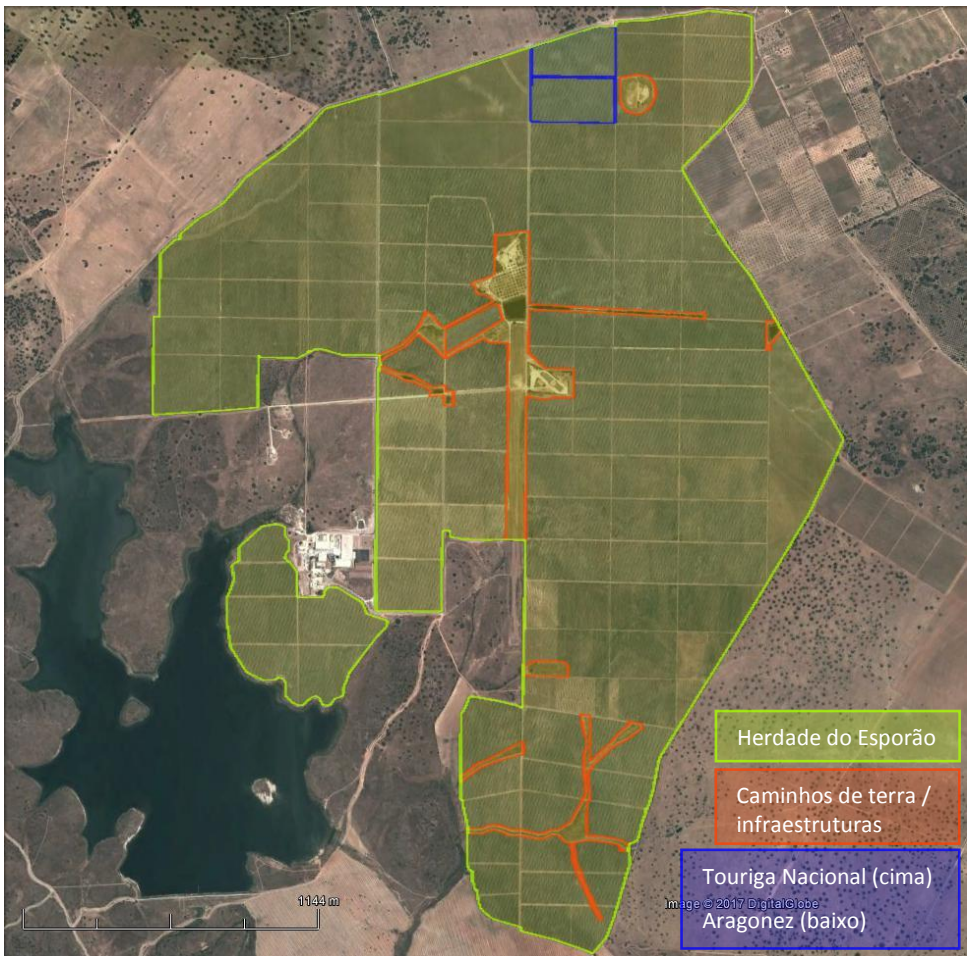


Figura 7 - Herdade do Esporão

Os dados fornecidos são representados sob a forma de um *script SQL* que permite reconstruir uma base de dados da *Innovine*, com dados correspondentes às castas já descritas. A amplitude temporal dos dados compreende-se entre 2013 e 2015.

3.2.2. Sistema Nacional de Informação de Recursos Hídricos

O segundo *dataset* compreende dados do Sistema Nacional de Informação de Recursos Hídricos (SNIRH). Os dados utilizados são hidro-meteorológicos e estão disponíveis publicamente, sendo recolhidos pela rede de monitorização e recursos hídricos do Ministério do Ambiente. A recolha de dados baseia-se nas diferentes redes meteorológicas próximas da Herdade do Esporão. A figura 8 ilustra as estações nas

proximidades da Herdade, sendo que as estações assinaladas a verde são estações ativas, a vermelho inativas e a amarelo ativas mas sem dados compreendidos entre 2013 e 2015. As estações eleitas para a recolha de dados são:

- São Manços
- Montoito
- Santiago Maior
- Reguengos
- Albufeira do Alqueva (Mourão)
- Albufeira do Alqueva
- Portel

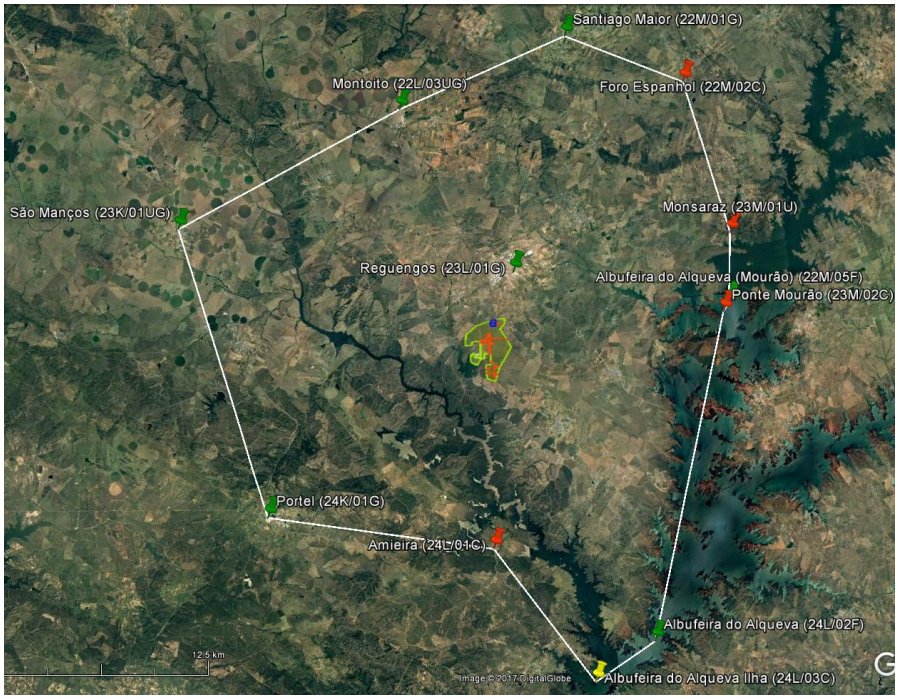


Figura 8 - Estações meteorológicas

A informação que é possível recolher no SNIRH sobre estas estações meteorológicas depende de estação para estação, pois nem todas têm dados para o mesmo período temporal, nem os mesmos parâmetros com dados. Exemplos dos parâmetros possíveis de obter são dados sobre a temperatura do ar, direção e velocidade do vento, precipitação e humidade. É possível extraí-los do *website* do SNIRH em formato *xlsx*, *csv* ou *tsv.*

3.2.3. Sentinel-2

Para se obter dados que ilustrem a área circundante ao terreno onde se localizam as castas, utilizaram-se imagens satélite provenientes do *Sentinel-2*. Este satélite pertence ao programa *Copernicus* cuja missão se foca no desenvolvimento de serviços de informação com base em satélites observacionais especializados na monitorização ambiental do planeta Terra.

O programa *Copernicus* conta já com várias missões de satélites lançados, desde o *Sentinel-1* ao *Sentinel-3* (S3A) e outras missões já planeadas como o *Sentinel-4*, *Sentinel-5*, *Sentinel-5p* e *Sentinel-6* (ESA, 2017c). Cada um dos satélites tem diferentes tecnologias e objetivos sendo que cada missão é sempre composta por uma constelação de dois satélites. No caso do *Sentinel-2* temos o *Sentinel-2A* e *Sentinel-2B* que operam em conjunto para terem uma maior cobertura e preencherem todos os requisitos necessários da missão (ESA, 2017b).

O *Sentinel-1* apresenta instrumentos como o *C-SAR* (*C-band Synthetic Aperture Radar*) capaz de operar na região das ondas microondas, permitindo a captura de informação dia e noite (ESA, 2017a). Tal é possível, uma vez que o instrumento envia sinais para a Terra através de um sensor, sendo medida a energia refletida pelos objetos cuja onda atingiu. Assim, obtém-se um satélite que não necessita da luz solar para a obtenção de dados. A desvantagem passa pela complexidade de interpretação e processamento dos dados.

O *Sentinel-2* é composto por um conjunto de tecnologias e instrumentos que permitem capturar imagens de alta resolução, tanto em terra como no mar. É utilizado para a monitorização de terra e como meio de resposta a emergências. O satélite possui um instrumento ótico (*MultiSpectral Instrument* ou *MSI*) que captura a luminosidade refletida pela terra em treze bandas que abrangem desde a luz visível e quase infravermelha - *Near Infrared* (*NIR*) às ondas curtas infravermelhas - *Short-wavelength infrared* (*SWIR*) (Fletcher, 2012). Como a tabela 4 indica, das treze bandas, quatro têm uma resolução espacial de 10 metros, seis de 20 metros e três de 60 metros. A largura de faixa do varrimento é de aproximadamente 290 km.

	S2A		S2B		
Número de Banda	**Comprimento de onda central (nm)**	**Largura de banda (nm)**	**Comprimento de onda central (nm)**	**Largura de banda (nm)**	**Resolução espacial (m)**
1	443.9	27	442.3	45	60
2	496.6	98	492.1	98	10
3	560.0	45	559	46	10
4	664.5	38	665	39	10
5	703.9	19	703.8	20	20
6	740.2	18	739.1	18	20
7	782.5	28	779.7	28	20
8	835.1	145	833	133	10
8a	864.8	33	864	32	20
9	945.0	26	943.2	27	60
10	1373.5	75	1376.9	76	60
11	1613.7	143	1610.4	141	20
12	2202.4	242	2185.7	238	20

Tabela 4 - Espetro de bandas para os sensores do *Sentinel-2* (S2A & S2B)
Fonte: Adaptado de https://sentinels.copernicus.eu/web/sentinel/technical-guides/sentinel-2-msi/msi-instrument

O tamanho das imagens depende da sua dimensão mas, pode aferir-se que rondam entre 1Gb e 10Gb por imagem, o que é um artefacto com algum peso e perfeito para um *data lake*. No entanto, como existem poucas imagens para os anos relativos aos dados dos sensores e ainda com a agravante da imagem ter uma percentagem demasiado alta de nuvens, não foram utilizadas imagens dessa fonte de dados. Porém, existe a aplicação *Sentinel Playground,* que disponibiliza imagens de alta resolução do *Sentinel-2* com uma maior periodicidade entre o espaço temporal necessário para análise. Esta aplicação assemelha-se a um *Google Maps* mas, no caso da aplicação podemos escolher o dia em que queremos ver o mapa. Existem imagens apenas a partir de 2015, sendo possível aplicar sobre o mapa combinações de bandas e consequentemente obter-se transformações espetrais da imagem. Esta particularidade permite obter análises da imagem em causa sem recorrer a programas de processamento de imagem. É também possível escolher a percentagem limite de nuvens que queremos ver representada no mapa.

No total, foram capturadas 48 imagens abrangendo 13 dias distintos, todos eles na segunda metade do ano de 2015. As "renderizações" escolhidas têm em consideração a

utilidade e importância que poderá ter na análise das imagens, no âmbito agrícola. Um exemplo de um índice utilizado é o *NDVI - Normalized Difference Vegetation Index,* representado na figura 9 pelo número 4. Este índice poderá ajudar a ter um maior rendimento de uma colheita (Mulla, 2013), monitorizando as castas e detetando eventuais secas na região. Outro exemplo é o índice *NDWI - Normalized Difference Water Index* (fig. 9, número 1), que permite identificar a concentração de água nas copas da vegetação, vulgarmente utilizado em complementaridade com o *NDVI* (Gao, 1996). Os números 2 e 3 da figura 9 correspondem à cor verdadeira e infravermelha respetivamente.

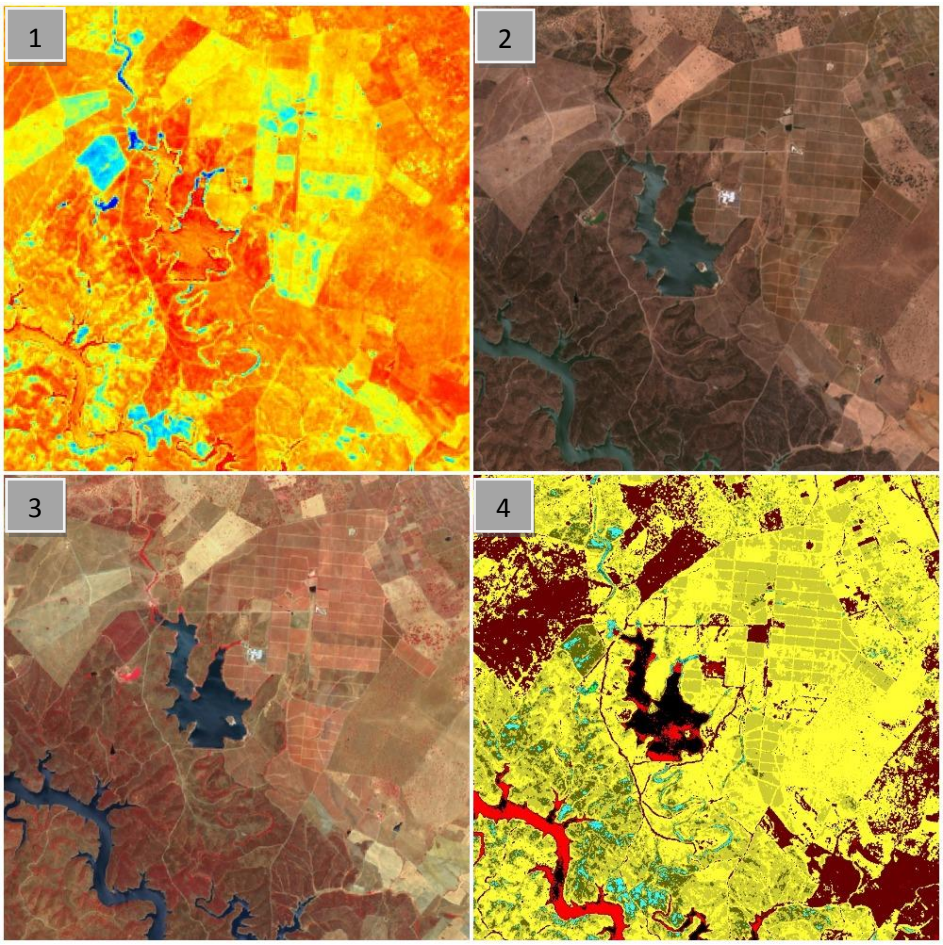


Figura 9 - Imagens capturadas 2015-07-15 – Herdade do Esporão

3.2.4. Mercado de venda

Idealmente, para o cruzamento dos dados seria interessante obter informação sobre os produtos que originaram das castas em objeto de análise, como por exemplo, os vinhos que foram produzidos nas castas Touriga Nacional e Aragonez; o seu preço no espaço temporal em que existem dados do *Innovine* e outras especificidades do produto como características do lote; unidades produzidas e vendidas, entre outras. Não foi possível obter este tipo de informação junto da Herdade do Esporão, pelo que tomou-se iniciativa de obter dados generalistas sobre o mercado de venda no momento, através de uma técnica denominada *Web Scraping*.

Web Scraping consiste na extração de informação automática de um *website*, utilizando tipicamente um *software*. A extração é parametrizada ou através de uma linguagem de programação, ou caso o *software* tenha uma interface, através de botões e *widgets* que ajudam na parametrização do que queremos extrair, como por exemplo, fornecer apenas *URLs*, número de páginas a extrair, ou algo mais complexo em que se definem objetos e criam-se regras de extração dos dados. Utilizando esta técnica, é possível extrair dados com rapidez e de forma automática, no entanto, acresce a complexidade dos processos caso seja necessário integrar a informação de diferentes páginas ou domínios (Barreira, 2014). Por isso, o *website* escolhido é um elemento agregador de várias superfícies de supermercados atuando como um comparador de preços online – *Maiscarrinho*, fornecendo assim, informação valiosa do preço dos vinhos entre vários supermercados, como o Continente, Jumbo ou Intermarché.

A ferramenta utilizada para a extração dos dados do *Maiscarrinho* não pertence ao *data lake*, porque a *Sandbox Hortonworks* utilizada como ambiente de teste, não incorporava nativamente uma ferramenta que permitisse fazer *web scraping*. Para evitar a instalação adicional e o conflito de versões entre a *sandbox* utilizada, recorreu-se a um programa fora do ambiente do *data lake*, *Open Source* - denominado *Scrapy*.

Scrapy é um exemplo de uma ferramenta que recorre à linguagem *Python*, o que permite ter maior flexibilidade e liberdade na extração pretendida. A sua utilização é controlada exclusivamente através da linha de comandos, invocando o comando de execução de um *Spider,* que consiste numa classe criada pelo utilizador onde se encontra um *script* em *Python* com o código relativo à extração das páginas.

É de salientar que foi tido em conta um *website* que permitisse a extração dos dados, recorrendo ao protocolo de exclusão de *robots* (*The Robots Exclusion Protocol*). Isto é,

quando o *script* é executado, no momento em que se visita a página do *website* Maiscarrinho, verifica através do ficheiro *robots.txt* se pode aceder à página em questão. Exemplificando, caso o *url* a aceder seja "*https://maiscarrinho.com",* o *script* verifica primeiro o endereço "*https://maiscarrinho.com/robots.txt"* que contém o ficheiro público *robots.txt* que indica se o autor do *website* autoriza ou não que um *robot* aceda a essa página. No caso do *url* indicado, somos presenteados com a seguinte informação:

```
# www.robotstxt.org/

# Allow crawling of all content
User-agent: *
Disallow:
```

Na linha "*User-agent*", o facto de ter um asterisco significa que a secção se aplica a todos os *robots*. No "*Disallow*" caso o autor pretendesse, poderia ter secções do *site* ou mesmo todo o *site* interdito a qualquer "ataque" de um *robot*.

A particularidade de obedecer ou não ao ficheiro *robots.txt* é explicitada numa classe de configurações do *Scrapy*.

Os elementos extraídos são todos os vinhos que se encontram no *website,* incluindo vinhos da Herdade do Esporão. Imagens, preços, supermercado, nome do vinho e tipo de vinho são exemplos dos dados extraídos pelo *Spider*. Toda esta informação é armazenada numa base de dados *SQL Server*, para que possa ser introduzida posteriormente no *data lake*.

3.3. *IMPLEMENTAÇÃO*

Após a identificação das fontes de dados é possível definir a arquitetura a ser utilizada para a construção do protótipo de um *data lake*. O subcapítulo da implementação é constituído pelo desenho da arquitetura, desde a visão genérica, até ao detalhe dos seus constituintes, pelo ambiente de instalação e configuração, processos de extração de dados, segurança e por fim, visualização de dados.

3.3.1. Arquitetura de Dados - *Data Lake*

Como a *Sandbox* escolhida pertence ao *Hortonworks,* a arquitetura subjacente terá sempre o "esqueleto" que esta ferramenta integra, publicitando-a como "*Modern Data Architecture with Apache Hadoop*". As principais diferenças serão delineadas pelas ferramentas utilizadas dentro da *Sandbox* (sejam de integração, transformação ou

segurança), pelas fontes de dados utilizadas e pelas aplicações usadas para fins de análise e *reporting*, sendo esta a componente final a que o utilizador pode recorrer para a exploração dos dados.

O grupo *Hortonworks* utiliza o *Hadoop* como uma componente central e chave, para complementar os atuais sistemas de dados (Hortonworks, 2014). Tendo a particularidade de ser uma tecnologia *open source* otimizada para ser executada em paralelo em diversos servidores, é uma solução "*low cost*" e *scale-out* (expansível com novo *hardware*), para o armazenamento e processamento de dados (Hortonworks, 2014). Torna-se um alvo atrativo para empresas que têm na sua posse grandes quantidades de dados, e que não querem que esses dados sejam desperdiçados porque a sua conversão para informação e conhecimento poderá ser uma mais-valia, representando um avanço competitivo no mercado.

A primeira abordagem do desenho da arquitetura representa uma visão holística generalista do modelo funcional da arquitetura moderna de dados – figura 10. São descritas as diferentes camadas não referindo as tecnologias usadas na sua construção.

A primeira camada representa as fontes de dados (*Data Sources*), representada por dados sensoriais, documentos, bases de dados e imagens, que vão de encontro aos tipos de dados já descritos.

O próximo componente são as arquiteturas de dados, onde existe o repositório de dados atual (Base de dados), como a arquitetura moderna, que está diretamente relacionada com o repositório atual. A arquitetura moderna é composta pelo *Hortonworks Data Platform* (*HDP*), que representa a distribuição *Apache Hadoop.* Esta é a principal componente do *Data Lake*, uma *Sandbox* que age como um ambiente de teste, aprendizagem e desenvolvimento que vai de encontro com as necessidades do projeto. Assim, possui-se um protótipo numa máquina virtual, com um *cluster* de um nó (*single node cluster*) do *Apache Hadoop* e todo o seu ecossistema de projetos inerentes. Salienta-se o facto do *data lake* ser constituído apenas por um *cluster* de um nó, comportando-se como um ambiente pseudo-distribuído, devido aos componentes fulcrais do *Hadoop* como o *NameNode*, *DataNode*, *JobTracker* e *TaskTracker* funcionarem todos na mesma máquina. Só se conseguiria ter uma arquitetura com múltiplos nós, se existisse a separação do *NameNode* e um ou vários *DataNodes* em diferentes máquinas, respeitando a arquitetura *master-slave*. Este último ambiente, é o típico cenário de um *cluster Hadoop* funcionando em diversas máquinas.

No *HDP* existem variadas áreas funcionais como *Data Management* onde se armazenam e processam os dados numa camada de armazenamento expansível. Na área *Data Access* é possível aceder e interagir com os dados através de *SQL* por exemplo, bases de dados não relacionais, *streaming*, entre outras formas. *Data Governance & Integration* engloba a ingestão dos dados, auditoria e qualidade. *Security* abrange a segurança nas autenticações, autorizações e proteção dos dados. *Operations* diz respeito a monitorizações e gestão no *cluster Hadoop*, como por exemplo gestão dos utilizadores ou automatizações de processos através de agendamentos.

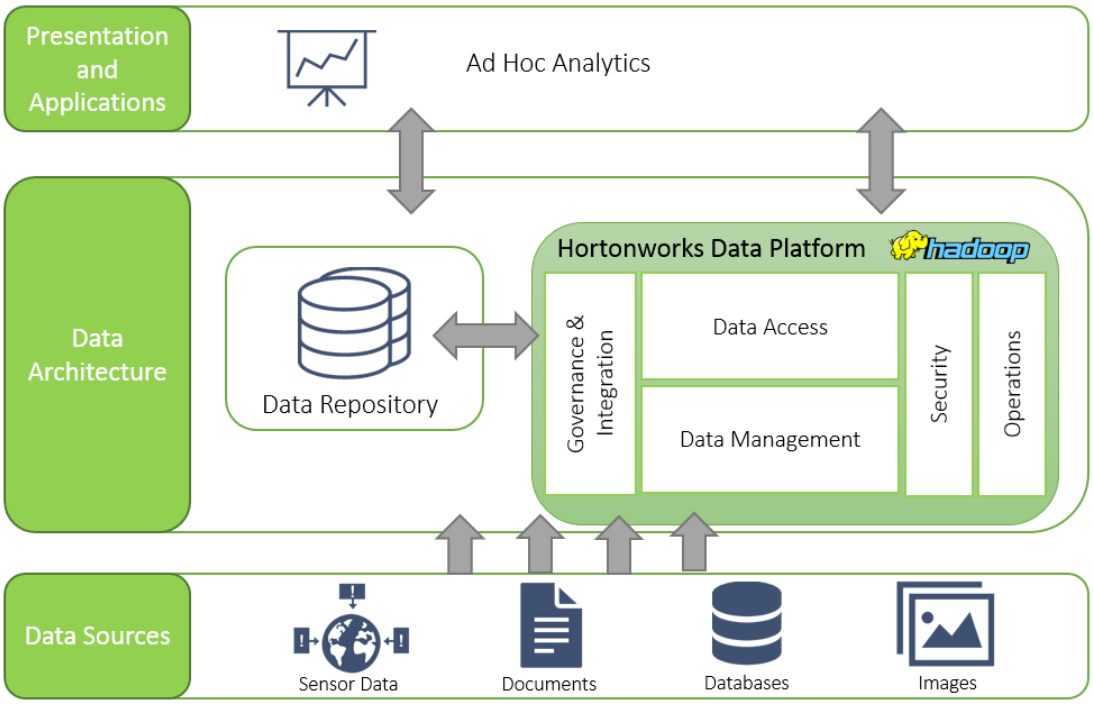


Figura 10 - Arquitetura Moderna de Dados - visão genérica

A visão detalhada do *data lake* é representada na figura 11. Apenas foram referidas as ferramentas utilizadas no projeto, porque existe uma grande variedade para diferentes propósitos que o ecossistema *Hadoop* oferece. Começando na componente *Governance & Integration,* foram utilizadas duas tecnologias da fundação *Apache – Sqoop* e *Nifi. Sqoop* é uma ferramenta utilizada para transferir eficientemente dados entre o *Hadoop* e fontes de dados estruturadas relacionais, como bases de dados. No projeto foi utilizado para a transferência de dados entre bases de dados e as tecnologias *HBase* e *Hive*. A tecnologia *Apache Nifi* tem um papel essencial nas transformações de dados e na ingestão de dados para o *HDFS*. É uma ferramenta que automatiza a transferência de dados em *Data Flows*

(fluxos de dados), vulgarmente utilizado entre sistemas e fontes de dados tornando a sua ingestão rápida, automática e simples de interagir.

Na camada de *Data Management,* o Sistema de Ficheiros Distribuído *Hadoop* é a tecnologia "*core"* que permite ter uma camada de armazenamento *scale out*, pronto para ser integrado em máquinas *low cost* com *hardware* comum (Hortonworks, 2014). O *Yarn* (*Yet Another Resource Negotiator*) é um pré-requisito do *Hadoop,* pertencendo também à arquitetura central do *data lake*, pois permite a gestão de recursos e processos que originam em operações com os dados localizados no *Hadoop*. Como ponto de acesso aos dados, foram utilizadas duas tecnologias – *Hive* e *HBase*. *Apache Hive* é a ferramenta mais utilizada e muito conhecida para aceder aos dados, tendo para o efeito, vários motores especializados para o seu funcionamento (Hortonworks, 2014). Tem um comportamento similar a uma base de dados, assim como a linguagem de interação ser muito semelhante ao *SQL* (*Structured Query Language*). A ferramenta *HBase* é uma base de dados *NoSQL* orientada a colunas, construída com padrões de baixa latência (tempos rápidos) entre os pedidos com o *HDFS*.

Outra componente presente no *data lake* é a segurança, que não deve ser descartada dada a importância que tem em qualquer negócio. Foram implementadas boas práticas nas diversas ferramentas, e foram estabelecidas políticas de segurança entre grupos de utilizadores e utilizadores, de modo a obter um melhor controlo do que cada utilizador acede e vê. Para o efeito, utilizou-se o *Apache Ranger* que ajuda a monitorizar e gerir a segurança dos dados na plataforma *Hadoop*, e o *Apache Knox* que consiste num sistema que fornece acesso centralizado e autenticação aos serviços existentes no *cluster Apache Hadoop*.

Nas Operações, a ferramenta *Ambari* atua como interface sobre todos os outros serviços que existem no *data lake*. Também é possível monitorizar certas métricas chave, em cada um dos serviços e no próprio *HDFS* e *Yarn*. O *Zookeper* é usado involuntariamente, pois é executado automaticamente, tendo como função coordenar serviços para aplicações distribuídas, assegurando a coordenação de processos entre as aplicações.

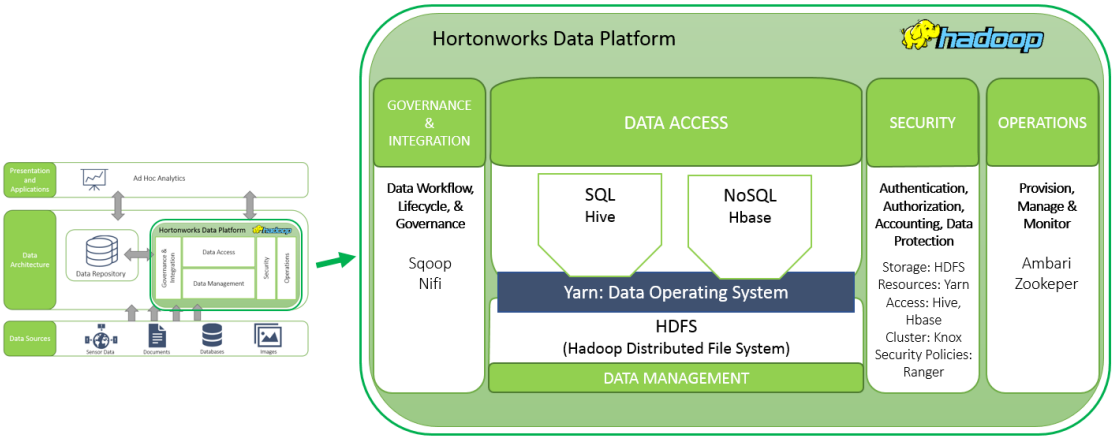


Figura 11 - *Data Lake* - visão detalhada

Dada a implementação do *Data Lake*, se não existisse pelo menos um protótipo para a visualização dos dados, seria difícil ter perceção da utilidade que estes dados podem ter após a sua devida transformação e análise. Para que o utilizador possa navegar e explorar os dados, estabeleceu-se uma conetividade entre o *data lake* e o *Power BI* sendo os dados acessíveis nesta ferramenta de visualização e "*dashboarding*". O ponto de acesso foi contruído entre a ferramenta *Hive*, pois é a que se assemelha mais a um *Data Warehouse*, através da conetividade *ODBC* (*Open Database Connectitivty*), utilizada como ponte de ligação a sistemas de gestão de bases de dados. A visão completa da arquitetura é demonstrada na figura 12.

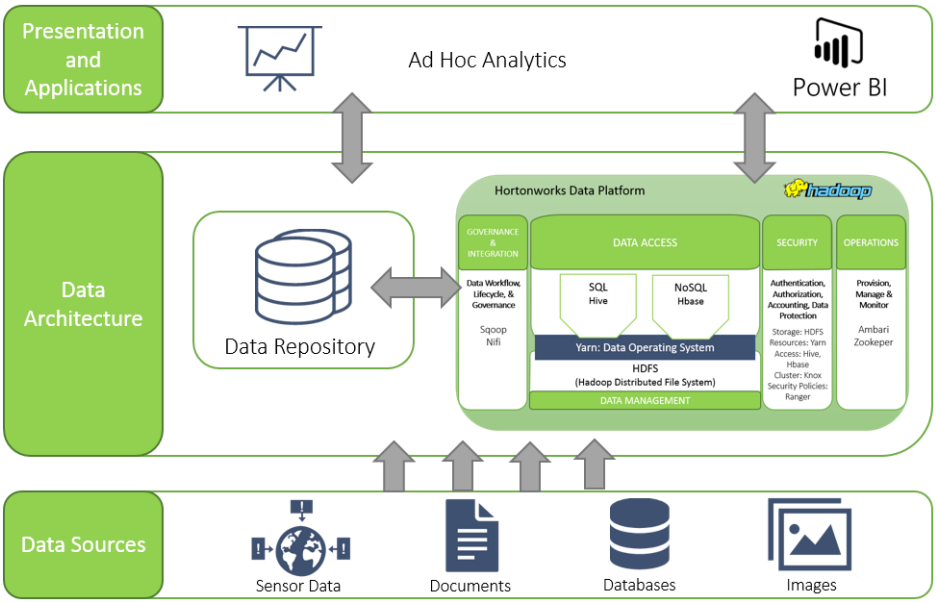


Figura 12 - *Data Lake* - visão completa

3.3.2. Ambiente de instalação e configuração

A instalação da *Sandbox Hortonworks* é "*On-Premises*", o que significa que o *software* é instalado localmente numa máquina. A versão escolhida era a mais recente no momento em que se instalou a *Sandbox,* sendo designada por *Hortonworks Data Platform 2.5.* Tendo em conta os requisitos mínimos que o *Hortonworks* refere na sua documentação, foi instalada uma máquina virtual com a *Sandbox Hortonworks*, numa máquina com as seguintes características:

- Sistema Operativo – *Windows 8.1 Pro*
- Processador – *Intel(R) Core(TM) i7-3630QM @ 2.40GHz*
- Memória – 16 *GB*
- Tipo de sistema – 64 *bits*, processador baseado em x64
- Disco Rígido – 1 *TB*

Um dos requisitos mais importantes é a memória disponível, porque para ligar certos serviços como o *HBase*, *Storm* ou *Kafka*, é necessário no mínimo *8 GB* de memória dedicados à máquina virtual. Como se utilizou o *HBase* e outros serviços, a máquina virtual foi parametrizada com 8 *GB* de memória. Para se usar a *Sandbox*, é necessário ter instalado uma das três máquinas virtuais: *VirtualBox*, *VMWare Fusion, Hyper-V*. A máquina utilizada foi a *VirtualBox*.

O facto da instalação ser *On-Premises* e apenas numa máquina, irá consumir memória física, pelo que é importante estimar o tamanho físico dos dados que queremos inserir no *data lake*, de modo a salvaguardar memória física.

O sistema operativo do ambiente virtual é o *CentOS Linux 7*, distribuído pela *Red Hat* que disponibiliza soluções baseadas em *Linux*.

O primeiro passo consiste em fazer o *download* do ficheiro a importar para o *VirtualBox.* Como tem cerca de 11.5 *GB*, não é fácil fazer a transferência sem que ocorra uma quebra de ligação de rede, pelo que se seguiu um método pouco convencional, mas mais fiável que o comum *download*, através do *web browser*. A metodologia passa por invocar um comando em *PowerShell* para transferir o ficheiro via linha de comandos. A figura 13 demonstra o comando invocado.

Figura 13 – *PowerShell* – Importação do ficheiro *Sandbox*

Assim que a importação do ficheiro para a máquina virtual estiver concluída, ao iniciar obtemos o endereço do ambiente virtual que depende do sistema instalado. Como o sistema utilizado é a *Virtualbox*, o endereço é o seguinte: ***127.0.0.1.*** Ao concatenar a porta ***8888*** ao endereço especificado e ao introduzir no *browser*, entra-se na página de boas vindas como indica a figura 14.

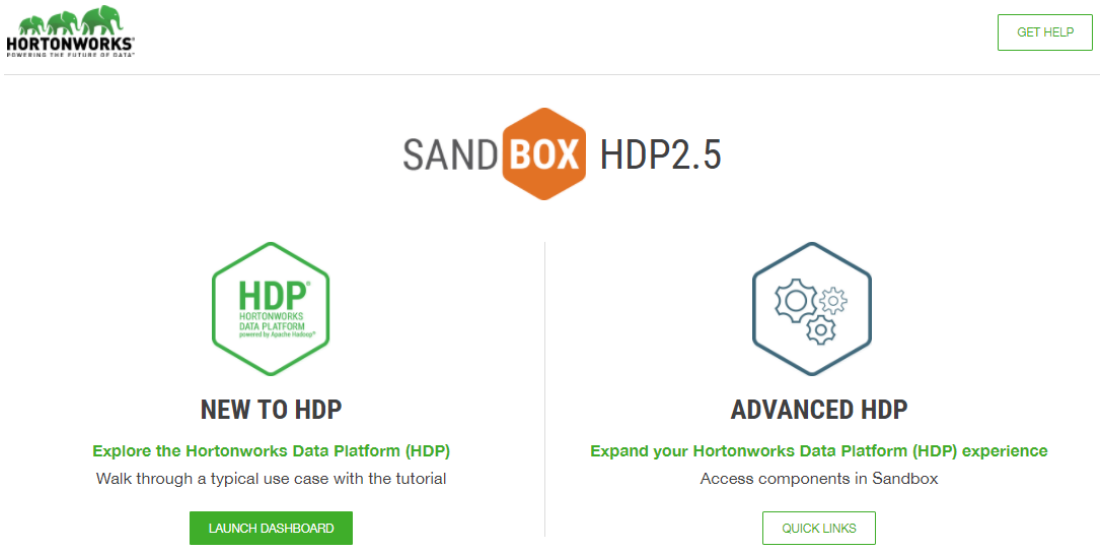


Figura 14 - Página de boas vindas - HDP 2.5

Ao clicar no botão "*Launch Dashboard*", entramos no serviço *Ambari*, que funciona como uma interface sobre os serviços da *Sandbox*. Como boa prática alterou-se o utilizador e *password* que dá acesso ao *Ambari*.

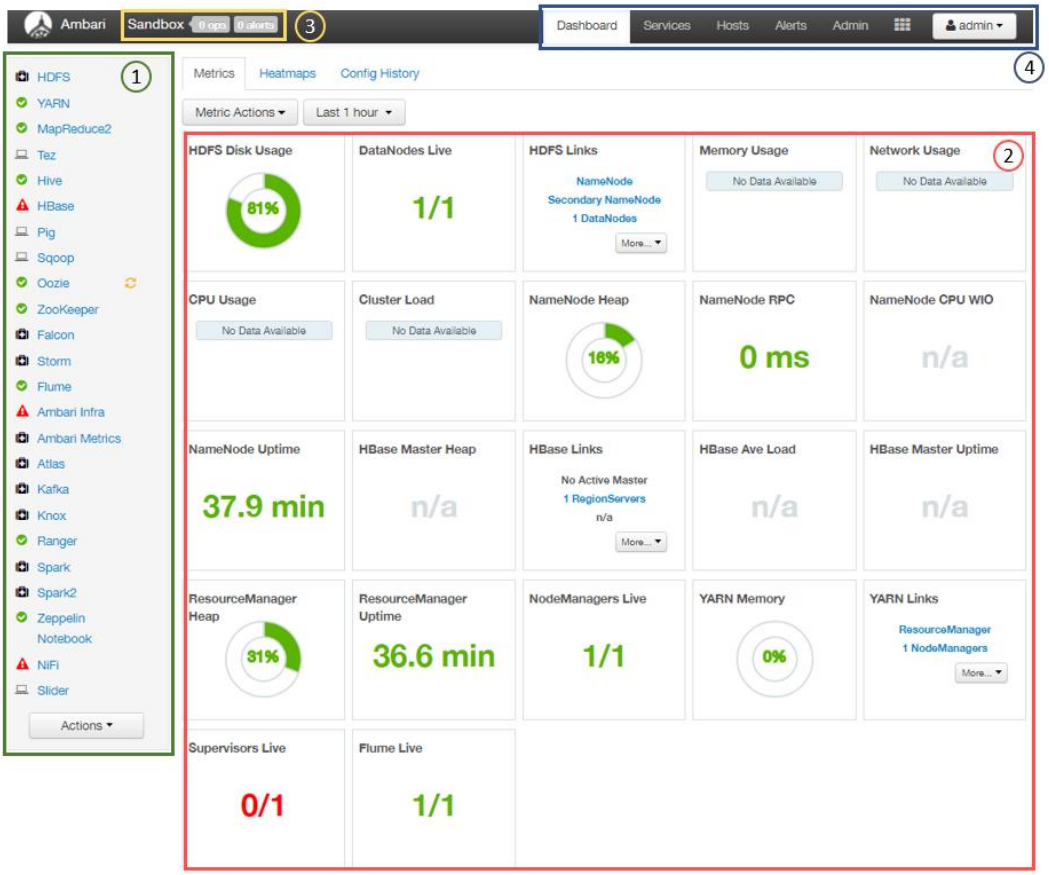


Figura 15 - *Ambari* - Página inicial

A figura 15 ilustra a página inicial da interface *Ambari*. As quatro regiões mais importantes da figura estão assinaladas. No número 1, podemos observar os serviços que estão a ser executados, ou que estão parados. Os que têm um símbolo verde significa que estão a ser executados, sendo que a figura 15 foi tirada assim que se efetua o *login,* portanto, os serviços com uma seta verde são iniciados automaticamente pelo *Ambari*. Se quisermos executar outro serviço dos que não estão em funcionamento, teremos que fazê-lo manualmente, clicando no serviço e carregando no botão iniciar. Os serviços que têm um círculo amarelo como o *Oozie*, significa que devemos reiniciá-lo, porque algum serviço relacionado com o *Oozie* ou o próprio, sofreu alterações que requerem o seu reinício. Este acontecimento é comum, devido às alterações de configurações que se fazem nos serviços.

O número 2 contém algumas métricas principais relacionadas com o sistema de ficheiros, com os *NameNodes* e *DataNodes*, *Yarn* entre outras métricas úteis para ter uma visão geral do *cluster*. Cada serviço tem também ao seu dispor métricas semelhantes como ilustra na figura 15.

A terceira região permite saber o número de operações em curso leia-se, operações que envolvam o início ou a paragem de um serviço, com o detalhe de progresso acompanhado com um *log* (registo de eventos desencadeados). Caso exista algum alerta, é também indicado nesta região.

A quarta região contém a visão do *Dashboard*, que é a selecionada e contém os serviços idênticos ao número 1, os *Hosts* que têm definições do *host* ou dos *hosts*, alertas, versões de cada serviço no botão *Admin,* e no botão semelhante a uma grelha é possível aceder a vistas parametrizáveis de serviços, como o *Hive* ou o Sistema de ficheiros *HDFS.* No botão mais à direita, encontra-se a opção de fazer *logout,* de gerir utilizadores/grupos e criar vistas acessíveis a utilizadores específicos ou grupos.

Existem duas formas de executar comandos via terminal, sendo que este método é o principal para interagir com determinados serviços, como o *Hbase*, para parametrizar ficheiros de configuração (embora também o seja através do *Ambari),* para verificar *logs,* ou até para iniciar ou terminar serviços caso a interface *Ambari* tenha algum problema ao iniciar. Uma das formas é através do *Secure Shell* (*SSH*), recorrendo a um programa fora do *data lake*, denominado *Putty*. Este programa é um *software open source* recomendado pelo *Hortonworks*. Basta colocar o endereço *127.0.0.1* e a porta *2222* e efetuar o *login* com o *username* ***root*** e *password* ***hadoop.*** Estes dados de autenticação também foram alterados posteriormente.

O outro método para executar comandos acontece através de um *Shell Web Client* com o endereço ***127.0.0.1:4200.*** Ao contrário do programa *SSH*, esta metodologia funciona via *Web.*

Com os requisitos de instalação cumpridos, podemos passar à fase seguinte com a extração dos dados para o ambiente virtual.

3.3.3. Processos de Extração de Dados

Começando no primeiro *dataset*, os dados provenientes do projeto *Innovine* foram fornecidos sob a forma de um *script* em *SQL* – ficheiro que permite reconstruir e popular a base de dados *Innovine*. A primeira abordagem consistiu em abrir o *script* no *SQL Server 2014 Management Studio* (*SSMS*) – uma ferramenta para gerir a infraestrutura *SQL*. No entanto, dada a dimensão do *script,* não existia memória suficiente para abrir no *SSMS*. Para transferir ficheiros com grandes volumes de dados, existe uma funcionalidade do *SSMS*, denominada *SqlCmd*, que permite executar ficheiros, procedimentos ou linguagem direta em *T-SQL (Transact-SQL)* via linha de comandos.

Na diretoria `C:\Program Files\Microsoft SQL Server\110\Tools\Binn` executou-se o comando: `sqlcmd -S FRANCISCO\MSSQLSERVERERP -i C:\Innovine.sql` .

No argumento "-S", invoca-se o nome da instância *SQL* e no argumento "-i" a localização do ficheiro para importação.

Após a importação é possível verificar que foi criada uma base de dados com quatro tabelas cujo diagrama é representado na figura 16.

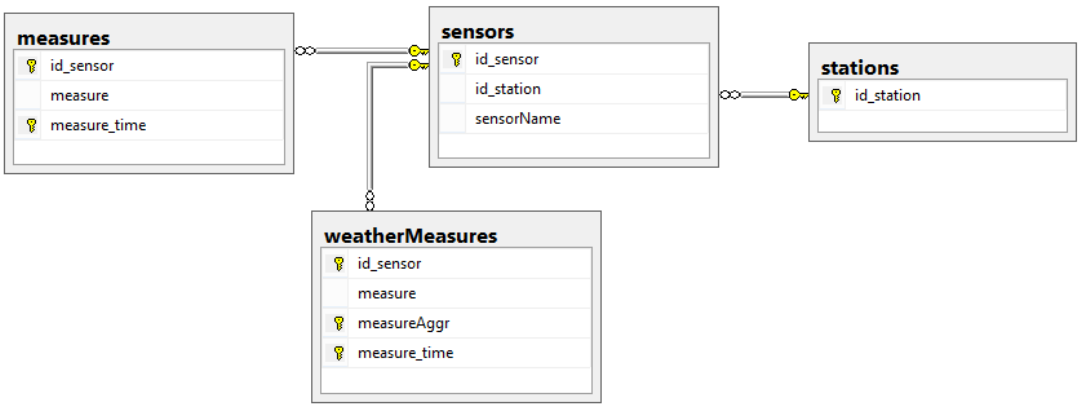


Figura 16 - Diagrama de Entidades e Relações

3.3.3.1. *Sqoop* para *HBase*

Como primeira metodologia para inserir os dados provenientes do projeto *Innovine* na *Sandbox* do *Hortonworks*, utilizou-se a ferramenta *Sqoop*, disponível para utilização dentro do *data lake*. Este serviço é o ideal para transferir os dados de uma base de dados relacional para o *Apache Hadoop* de forma automática e eficiente (Foundation, 2017).

Do lado do *SQL Server*, existem pequenas configurações a serem alteradas para garantir a conectividade em ambos os lados. Como boa prática, criou-se um utilizador "*hadoop*" no *SQL Server* com as devidas permissões de leitura e escrita.

Como requisito de conetividade, o *SQL* tem que estar parametrizado com o protocolo de rede *TCP/IP* e como o *software* está instalado localmente por defeito, esta configuração está definida com o protocolo de rede de memória partilhada (*Shared Memory Network Protocol*). Após a alteração do protocolo e executando o código da figura 17, observamos que o protocolo está agora bem definido. Os restantes dados serão necessários para a componente de ligação ao *Sqoop*.

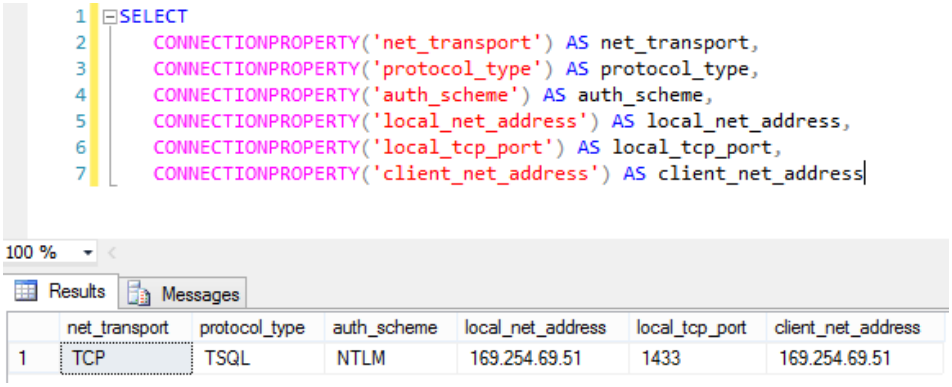


Figura 17 - Parametrização de Protocolo de Rede

O primeiro problema na transferência de dados, deve-se ao requisito do *Sqoop* necessitar de *drivers JDBC* (*Java Database Connectivity*) para estabelecer ligação ao *SQL Server*. Por defeito, o *Sqoop* não contém pré-instalado o *driver JDBC* (T. Kathleen & C. Jarek, 2013).

A instalação do *driver* em falta foi efetuada através da consola *Shell Web*, com os seguintes comandos:

```
wget https://www.dropbox.com/s/31317fp4/sqljdbc_6.0.8112.100_enu.tar.gz?dl=0
mv sqljdbc_6.0.8112.100_enu.tar.gz?dl=0  sqljdbc_6.0.8112.100_enu.tar.gz
tar -xvzf sqljdbc_6.0.8112.100_enu.tar.gz
cp sqljdbc42.jar /usr/hdp/current/sqoop-client/lib
```

Sucintamente, o código descrito copia através do *url* onde se encontra o ficheiro do *driver*, renomeia o nome do ficheiro, descompacta-o e copia para o diretório de bibliotecas do *Sqoop*.

Com os requisitos do *Sqoop* cumpridos, a próxima fase consiste na transferência de dados para o *HBase.*

Apache HBase é uma base de dados distribuída não relacional, com acesso de leitura e escrita em tempo real, ideal para um projeto que tenha tabelas com grandes volumes de dados. Como os dados do projeto *Innovine* são sensoriais e com uma periodicidade quase ao minuto, possui as características que o *HBase* está otimizado a lidar.

Tendo a ferramenta que efetua a extração dos dados do *SQL Server*, basta ter o serviço do *HBase* iniciado para este servir de destinatário.

O primeiro teste efetuado, serve para verificar se a conexão está estabelecida agora entre os três lados, ou seja, *SQL Server, Sqoop e HBase.*

O código abaixo descrito invoca o comando "*list-databases*" (lista as bases de dados), usa os dados que foram tirados na figura 17 para a conexão ao servidor *SQL*, como o endereço *IP* e a porta, e os restantes dados são de autenticação, como o utilizador "*hadoop*" que se definiu como utilizador no SQL, e uma *password* em ficheiro, uma metodologia mais segura do *Sqoop* para não colocar em pleno texto a *password* na "*String"* de conexão.

```
sqoop list-databases --connect jdbc:sqlserver://169.254.69.51:1433 --
username=hadoop --password-file file:///root/sqoop.password
```

Existem três formas diferentes de autenticação ao *SQL Server*. A primeira já explicitada é através de um ficheiro com a *password*, sendo considerada a mais segura. A segunda forma, passando o argumento "-P" o utilizador insere na linha de comandos a sua *password* não sendo visível a escrita. A terceira metodologia e menos aconselhável, passa por colocar a *password* diretamente na *String* de conexão.

Com o teste básico efetuado com sucesso, temos a certeza que a conexão ficou bem estabelecida entre as três ferramentas.

Antes de se iniciar a extração dos dados, é necessário explicar as principais diferenças entre bases de dados relacionais e não relacionais e a arquitetura base de uma tabela em *HBase*, porque é diferente de uma tabela de uma base de dados relacional.

A primeira diferença deve-se ao modo como o *HBase* guarda a sua informação. O *HBase* é uma base de dados orientada a colunas (*column-oriented*) ao contrário do comum das bases de dados relacionais, orientada a linhas (*row-oriented*). Exemplificando com a tabela "*measures*" do *Innovine*, a tabela 5 contém dois registos.

ID_Sensor	Measure	Measure_Time
1202515	13.5000	2013-07-15 14:24:24.000
1202515	13.4000	2013-07-15 15:00:30.000

Tabela 5 - Exemplo da Tabela *Measures*

Se a tabela for orientada a colunas, são armazenados como:

1202515, 1202515, 13.5000, 13.4000, 2013-07-15 14:24:24.000, 2013-07-15 15:00:30.000

De outra forma, se for guardada orientada a linhas, como o caso do *SQL Server*, os registos são armazenados em tuplos da seguinte forma:

1202515, 13.5000, 2013-07-15 14:24:24.000,

1202515, 13.4000, 2013-07-15 15:00:30.000

A primeira abordagem tem a vantagem sobre grandes volumes de dados, porque uma coluna é armazenada num só bloco, permitindo o acesso rápido aos dados. Esta abordagem é utilizada também quando se tem dados semiestruturados ou não estruturados, no entanto, no caso dos dados *Innovine*, são estruturados por estarem armazenados numa base de dados relacional.

Os principais componentes de uma tabela *HBase* estão representados na figura 18.

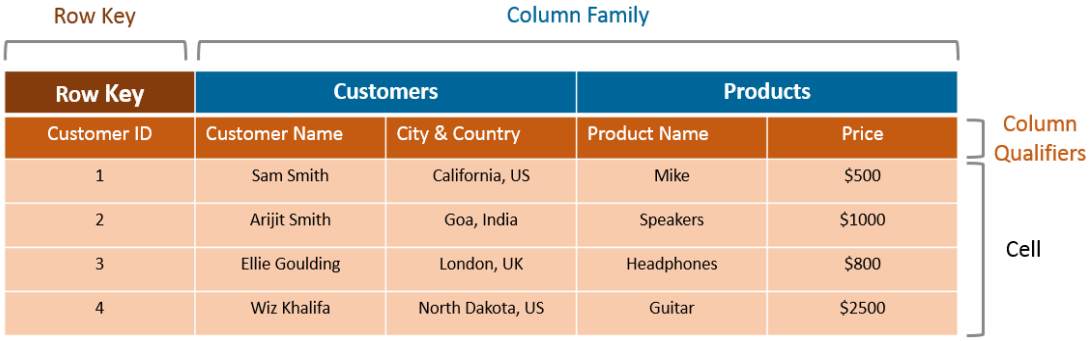


Figura 18 - Componentes de uma tabela *HBase - Exemplo*

- *RowKey* – um registo é composto por uma ou mais colunas, que é endereçado pela *Rowkey*, comportando-se como a chave de um registo. Tipicamente, a *Rowkey* é usada em pesquisas para procurar registos, tornando o acesso rápido em consultas específicas. Se a tabela fonte que estivermos a importar para o *HBase* tiver mais que uma chave, a chave composta será a *Rowkey* das chaves concatenadas por "_".
- *Column Fammily* – é composta por um conjunto de colunas formando uma família de colunas – *Column Family*. As *Column Families* são armazenadas em conjunto tornando o acesso rápido. Se necessitarmos de aceder a colunas pertencentes a uma família, o seu acesso é otimizado porque internamente, as colunas são guardadas no mesmo ficheiro, denominado *HFile* (George, 2010).
- *Column Qualifier* – é similar a uma coluna de uma tabela pertencente a uma base de dados relacional. Não existe um limite de colunas, nem existem tipos de dados sobre um valor, e as limitações em relação ao tamanho são também inexistentes (George, 2010).
- *Cell* – é a estrutura que contém o valor onde os dados são armazenados.

Outra particularidade não denotada na figura 18 é o *Timestamp* – uma combinação de data e hora guardada sempre que se insere dados numa tabela. O *timestamp* ajuda a procurar versões específicas dos dados.

Numa primeira etapa, e como boa prática num *data lake*, faz-se a extração exatamente como os dados se encontram, sem fazer qualquer tipo de transformação. O objetivo é ter uma réplica fiel dos dados fonte, exatamente como se encontram.

O *script* seguinte demonstra a criação da tabela "*measures*" no *HBase* assim como a instrução que copia os dados da tabela do *SQL Server* para a tabela criada no *HBase*.

```
create 'measures', 'measure'
sqoop import –connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table measures --
columns id_sensor,measure_time,measure --hbase-table measures --column-
family measure --hbase-row-key id_sensor,measure_time
snapshot 'measures', 'measures_snap'
disable 'measures'
drop 'measures'
```

Entrando na consola *HBase* com o comando "*hbase shell*", invoca-se o comando da criação da tabela, com o nome "*measures*" e com uma *column family* "*measure*". Na *string* de conexão, indica-se o nome da tabela fonte e as colunas que queremos importar. Depois, indica-se a tabela destino, o nome da *column family* e as colunas chave que compõem a *rowkey*. O restante código serve para "arquivar" a tabela de modo a que fique guardada mas não visível. Ao executarmos o comando "*snapshot*" a tabela é guardada mas não envolve qualquer cópia de dados, sendo o comando instantâneo. Caso queiramos recuperar o *snapshot* basta fazer o "*clone_snapshot*" e a tabela fica acessível novamente. Não implicando qualquer cópia de dados no seu restauro.

Os restantes comandos "*disable*" e "*drop*" são para remover a tabela, uma vez que já está arquivada. Antes de se efetuar o *drop*, é necessário fazer primeiro o *disable*.

Pelo mesmo método, as tabelas "*sensors*", "*stations*" e "*weatherMeasures*" foram arquivadas. O código utilizado está presente no anexo 6.1.

3.3.3.2. Otimizações na importação *Sqoop* para *HBase*

Durante a importação das tabelas notou-se que poderia haver margem para uma otimização no tempo de importação. As otimizações implementadas devem-se à modelação dos dados, ao desenho da *rowkey,* e às opções de configuração na importação dos dados através do *Sqoop*.

Em relação à modelação dos dados no *HBase*, optou-se por um modelo desnormalizado, ao contrário do que é habitual num modelo relacional. Tipicamente, num modelo relacional as tabelas estão normalizadas para evitar a duplicação dos dados e a sua redundância. Ao ter um modelo de dados normalizado, é necessário usar "*joins*" para o cruzamento dos dados, tornando por vezes o acesso lento devido a *queries* morosas. Ao ter o modelo desnormalizado, o acesso aos dados é mais rápido, e este é o modo preferencial quando se usa um esquema de dados no *HBase* (George, 2010). As relações que sejam de um para muitos (1:N) ou muitos para muitos (N:M) podem ser colapsadas, ficando-se apenas com uma tabela. A figura 19 ilustra a otimização de duas tabelas relacionais que tinham uma relação de (1:N), colapsadas apenas numa tabela em *HBase*.

Row Key		Timestamp	cf:Measure	cf:Sensor	
cq:Id_sensor	cq:Measure_time		cq:Measure	cq:Id_station	cq:Sensor_Name
1202515	2013-07-15 14:24:24.0	1489334133451	13.500	601	Tensao Bateria #601
1202515	2013-07-15 15:00:30.0	1489334133451	13.400	601	Tensao Bateria #601

Figura 19 – Tabela otimizada em *HBase*

Assim, na consulta por uma *rowkey*, obtemos informação não só do valor da métrica, como do descritivo do sensor e do identificador da estação. O mesmo método foi aplicado para o desenho da tabela que resulta da junção das tabelas *weatherMeasures* e *sensors.*

O desenho da *rowkey* surge devido a um problema conhecido de "*hotspotting*", no ato da escrita dos dados. *Hotspotting* é nada mais nada menos que a sobrecarga no momento da escrita numa determinada região, obstruindo assim a escrita dos dados na tabela *HBase*.

Para melhor entendimento, é necessário introduzir alguns conceitos dos componentes que fazem parte da arquitetura do *HBase*.

Ao armazenar os dados numa tabela *HBase*, a tabela é dividida em porções denominadas "Regiões" (*Regions*). Uma região contém todos os registos começados por uma chave de início (*Start Key*) e uma chave final (*End Key*), comportando-se como uma região contígua de dados armazenados em conjunto (George, 2010). As regiões são divididas dinamicamente quando atingem determinado limite (256 *MB*). Na criação de uma tabela, inicialmente existe apenas uma região, e só quando se atinge o limite é que a região se divide em dois na chave central (*Mid-Key*), resultando duas regiões com tamanhos idênticos. Cada região é servida apenas por um "*RegionServer*", mas um *RegionServer* pode servir várias regiões. O *RegionServer* é responsável por gerir e executar operações de escrita e leitura num conjunto de regiões, desempenhando um papel importante na escrita dos dados. A figura 20 ilustra um exemplo com três *RegionServers*, cada um assignado a duas regiões, obtendo-se uma carga uniforme ao longo dos três servidores.

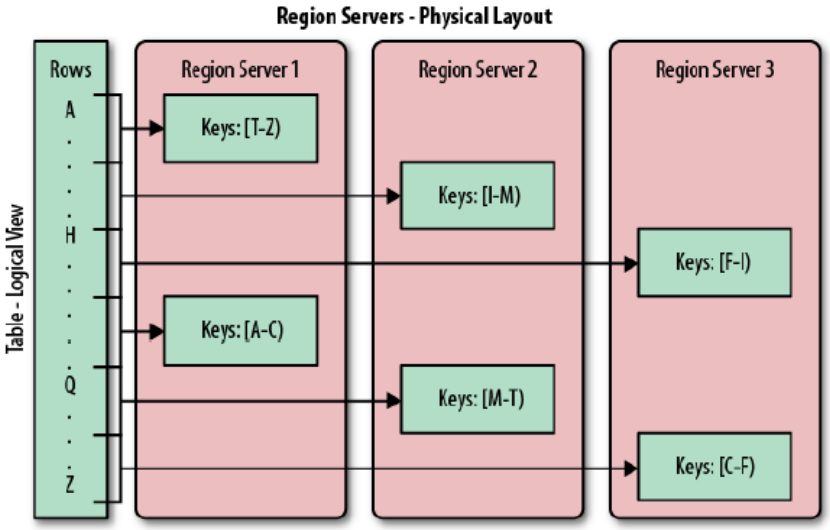


Figura 20 – *Layout* Físico – *RegionServers*
Fonte: Retirado de George, L. (2010). *HBase: The Definitive Guide*. *O'Reilly* (2nd ed., pp 27). O'Reilly Media, Inc.

Quando os dados são inseridos numa tabela, são ordenados alfabeticamente, o que permite um rápido acesso a um registo em específico se procurarmos pela sua chave individual, e também permite um acesso rápido se quisermos pesquisar por um conjunto de registos através de uma chave inicial e chave final.

O grande problema centra-se na escrita dos dados. Como estão ordenados, cria a situação de *hotspotting* devido à forma como o *HBase* armazena os registos nas regiões. Os registos com chave sequencial, quando estão a ser escritos causam a sobrecarga numa única região. Neste caso não existiria problema se uma região pertencesse a diferentes *RegionServers*, mas tal não acontece porque uma região só pode pertencer unicamente a um *RegionServer*. A figura 21 ilustra o que acontece quando se importam os dados para uma tabela *HBase* com chaves sequenciais. Enquanto um *RegionServer* atinge o esforço máximo, os outros servidores estão com carga mínima.

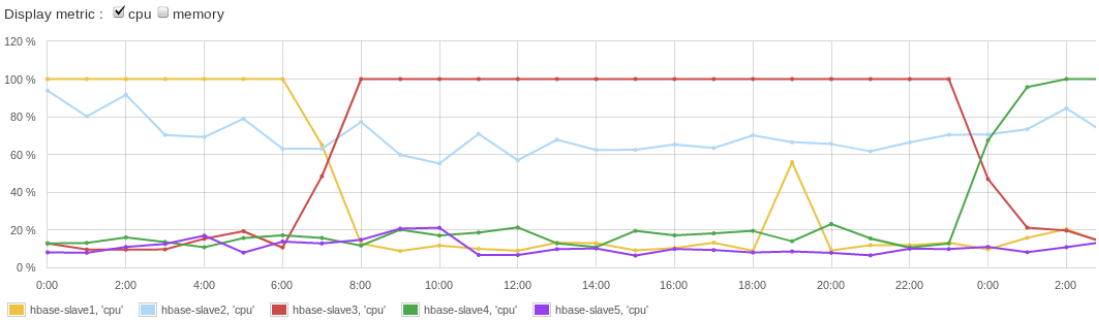


Figura 21 - *Hotspotting* num *RegionServer*
Fonte: https://sematext.com/2012/04/09/hbasewd-avoid-regionserver-hotspotting-despite-writing-records-with-sequential-keys/

Uma das soluções para resolver o problema, seria redistribuir a escrita dos dados em várias regiões usando chaves aleatórias. A metodologia para atribuir chaves aleatórias consiste em atribuir um número aleatório como prefixo a cada *rowkey*. Esta técnica chama-se "*Salting*", um termo muito conhecido no âmbito da criptografia. O objetivo passa por assignar a cada linha escrita, uma região diferente, dentro de um limite máximo de regiões (dependendo da dimensão dos dados). Aplicando este método, a figura 22 demonstra a distribuição uniforme entre todas as regiões, resultando assim num ganho significativo na otimização dos recursos e do tempo de inserção dos dados.

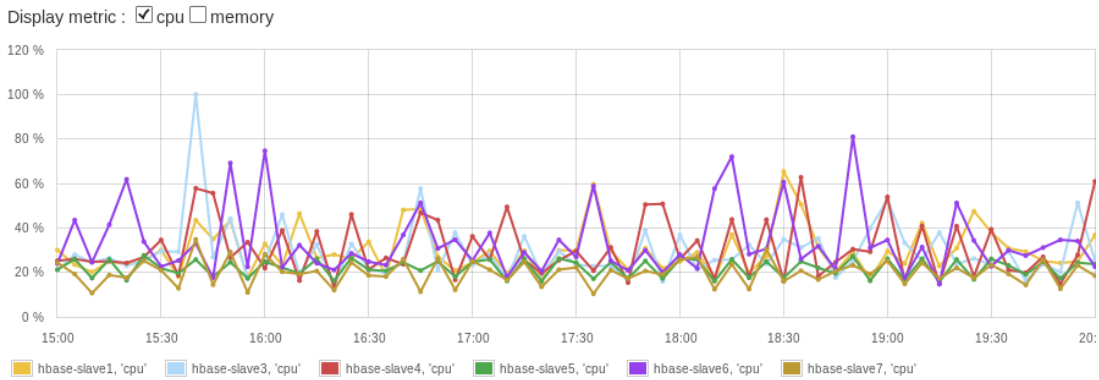


Figura 22 - Carga dos *RegionServers* distribuída uniformemente
Fonte: https://sematext.com/2012/04/09/hbasewd-avoid-regionserver-hotspotting-despite-writing-records-with-sequential-keys/

Uma das desvantagens desta abordagem deve-se ao facto dos dados serem lidos de múltiplas regiões, isto é, se pedirmos uma consulta com um determinado "*range*", estes dados estão espalhados em diversas regiões. Internamente, os *scans* são executados em paralelo, pelo que não devem degradar tanto a performance como o esperado.

Estas técnicas demonstradas resultaram de um estudo da empresa *Sematext*, especialista em consultoria em *Big Data*.

Outra situação que poderemos considerar uma desvantagem é o caso em que a tabela se encontra vazia. Se tentarmos inserir dados numa tabela vazia, mesmo com a metodologia explicada, na fase inicial de inserção, não é possível utilizar toda a capacidade do *cluster* (Soztutar, 2013). Para solucionar esta desvantagem, existe um processo denominado "*Pre-Splitting*" que consiste na divisão prévia da tabela antes da inserção de dados. Ao dividirmos a tabela em várias regiões previamente, estamos a garantir que no momento da escrita, todas as regiões estão criadas e prontas a receber dados. Ao dividir a tabela previamente, devemos conhecer com certeza a distribuição das chaves, ou corremos o risco de termos mais uma vez *RegionServers* descompensados, uns com mais ocupação de *CPU* do que outros.

Em boa prática não existe um número ótimo de regiões a ser utilizado, mas tipicamente escolhe-se um número baixo, múltiplo do número de *RegionServers* (Soztutar, 2013).

No caso dos dados *Innovine*, a tabela *measures* está ordenada por defeito pela chave composta entre o *Id_Sensor* e *Measure_Time,* o que dificulta se quisermos fazer uma divisão rigorosa em partes iguais.

Para facilitar este processo de divisão, criou-se um procedimento na base de dados *Innovine* no *SQL Server*, que cria uma coluna com o identificador do sensor, prefixada com dois *bytes* do *hash* (algoritmo que correlaciona dados de comprimento variável em dados de comprimento fixo) em *MD5* (algoritmo criptográfico de dispersão) em relação ao resultado da chave composta entre o *Id_Sensor* e *Measure_Time*.

O código seguinte descreve parte do procedimento que cria o atributo com o *hash:*

```
UPDATE measures
SET id_sensor_hashed =
CONVERT(VARCHAR(10),SUBSTRING(HashBytes('MD5',CAST(id_sensor AS NVARCHAR) +
CONVERT(VARCHAR(23),measure_time,20)),0,2) ,2)
+ '_' + CAST(id_sensor AS NVARCHAR);
```

A coluna resultante segue uma distribuição preditiva porque ao escolhermos apenas os dois primeiros *bytes* do resultado *MD5*, obtemos uma combinação de 256 valores diferentes em hexadecimal (o retorno da função *HashBytes* em *MD5* é do tipo hexadecimal), desde "00" até ao último valor "FF". Iremos usar este prefixo para a criação das divisões das tabelas *HBase.*

Um atalho para podermos saber automaticamente os pontos de divisão da tabela é usar um comando em *HBase* que retorna um registo de "x" em "x" intervalo. O número de regiões que se escolheu foram 10, por isso, teremos que dividir a tabela em 10 partes iguais. Usando o comando:

```
count 'measures', INTERVAL => 110064.6
```

Ao dividirmos o número de registos da tabela *measures* por 10 obtém-se o número 110064.6. Executando o comando descrito é devolvida uma lista de registos intervalados por esse número, descobrindo-se assim os pontos de divisão {19, 33, 4C, 66, 7F, 99, B3, CC, E6}, para efetuarmos a pré-divisão no momento da criação da tabela.

É de salientar que continua a ser possível efetuar uma consulta à procura de um sensor em específico, através da *rowkey*. Um exemplo da sua utilização é o seguinte:

```
scan 'measures', {FILTER => org.apache.hadoop.hbase.filter.RowFilter.new(
CompareFilter::CompareOp.valueOf('EQUAL'),SubstringComparator.new("1202515")
),'LIMIT' => 1500}
```

Tendo o modelo de dados otimizado, e o desenho da *rowkey* com as modificações referidas, restam algumas configurações no momento da importação das tabelas.

Realizaram-se cerca de 8 testes com configurações diferentes, cada um com 5 ensaios, para podermos realizar uma média do tempo de importação.

Todos os testes recaem sobre a importação das tabelas "*measures*" e "*sensors*" com o novo modelo descrito na figura 19.

A tabela 6 ilustra as variáveis que foram utilizadas nos testes de otimização do tempo de importação das tabelas *HBase*. A variável "*Nr. Mappers*" corresponde ao número de

mappers (relembrando as duas fases de processamento *MapReduce*) utilizados em paralelo no momento da importação dos dados. Por defeito, este valor quando não alterado no momento da importação usa quatro *mappers*, e não existe um número ótimo a utilizar porque depende também de diversas variáveis, como o tipo da base de dados ou *hardware* usado (T. Kathleen & C. Jarek, 2013). Para começar, o número de *mappers* utilizado deve ser baixo, escolhendo para o efeito apenas um no teste número oito.

A variável "*Pre-Split*" indica se a tabela foi dividida previamente com os pontos de divisão já descritos. A opção "*Hash RowKey*" indica se estamos a utilizar a tabela com o *hash* na coluna *Id_Sensor.*

A coluna "Regiões" indica o número de regiões utilizadas nos testes, sendo que por defeito é utilizado apenas uma região, porque a dimensão da tabela não é suficiente para implicar a divisão automática para outra região. Nos testes 2,3,5 e 6 foram utilizados algoritmos automáticos de divisão do *HBase*. O algoritmo *HexStringSplit* converte os registos para hexadecimal e divide as regiões com base nesses valores. O algoritmo *UniformSplit* divide o espaço das possíveis chaves uniformemente.

Teste	***Nr. Mappers***	***Pre-Split***	***Hash RowKey***	**Regiões**
1	Default (4)	Não	Não	Default (1)
2	Default (4)	Não	Não	10 (HexStringSplit)
3	Default (4)	Não	Não	10 (UniformSplit)
4	Default (4)	Não	Sim	Default (1)
5	Default (4)	Não	Sim	10 (HexStringSplit)
6	Default (4)	Não	Sim	10 (UniformSplit)
7	Default (4)	Sim	Sim	10
8	1	Sim	Sim	10

Tabela 6 - Variáveis de Otimização - tabelas *HBase*

Para cada ensaio foram medidos diversos tempos de execução, porque no momento da importação existem vários tempos que poderiam ser considerados para medir o tempo de importação das tabelas. Para cada ensaio foram medidos os seguintes tempos:

- **SHELL MR** – tempo do *Job MapReduce* registado na *Shell* como *output* da importação;
- **REAL** – tempo total passado desde o início da invocação do comando de importação, até ao fim;
- **USER** – tempo do *CPU* (fora do *Kernel*) gasto na execução do processo;
- **SYS** - tempo do *CPU* (no *Kernel*) gasto na execução do processo;
- **APP** – tempo registado pela métrica "*Elapsed*" na aplicação *MapReduce* dentro do gerenciador de *Jobs* que o *Yarn* fornece;
- **JOB** – tempo registado pela métrica "*Elapsed*" do *Job*;
- **AVG MAP** – tempo registado pela métrica "*Average Map Time*" do *Job;*
- **TT ALL MAPS** – tempo registado pela métrica "*Total Time Spent by all Map Tasks*", retirado de um conjunto de métricas associados ao *Job*;
- **CPU TIME** – tempo registado pela métrica "*CPU Time Spent*", retirado de um conjunto de métricas associados ao *Job*;

Um exemplo de cada um destes tempos medidos encontra-se no anexo 6.2, assim como os resultados detalhados de todos os ensaios.

A figura 23 ilustra os resultados obtidos em segundos, para cada um dos tempos e para cada teste, sendo que a tabela de cima é referente à média aritmética de todos os ensaios para cada teste, e a tabela debaixo regista a média de todos os testes comparativamente aos valores do melhor teste.

Sem margem de dúvida que o melhor teste efetuado é o número 8, tendo sido o melhor em todos os parâmetros à exceção da medição "*User*" e "Avg *Map*". Contudo, na medição *User*, conseguiu-se um valor mais baixo em relação à média, o que significa que mesmo assim, esta medição efetuada no teste 8 foi melhor comparativamente à média. Apenas o *Avg Map* registou um aumento face à média de aproximadamente meio segundo, o que poderemos considerar desprezável.

Com estes testes conseguimos um cenário que se destaca dentro do universo dos testes efetuados, tendo utilizado as configurações do teste 8 para a importação das restantes tabelas *HBase*.

Podemos também observar no anexo 6.2, as distribuições dos pedidos de acesso às regiões nos testes efetuados. Apenas nos últimos dois testes 7 e 8, é possível observar a distribuição uniforme nas regiões.

TESTE	TABELA	MÉDIA DOS ENSAIOS								
		SHELL MR	REAL	USER	SYS	APP	JOB	AVG MAP	TT ALL MAPS	CPU TIME
1	Measures	61,83	71,23	21,31	3,12	50,40	42,60	37,00	150,08	78,40
	Sensors	34,53	46,57	21,05	3,85	25,20	15,60	9,80	40,35	13,08
2	Measures	59,71	68,67	20,53	3,22	49,00	42,00	36,40	147,53	77,58
	Sensors	33,83	54,35	20,09	3,71	25,20	14,20	8,60	35,60	11,84
3	Measures	59,50	68,89	22,57	3,02	48,40	41,40	36,20	146,36	77,49
	Sensors	39,00	52,35	20,81	3,77	28,80	14,40	8,40	35,61	11,97
4	Measures	71,28	81,00	22,38	3,10	60,80	53,20	21,40	132,56	70,87
	Sensors	32,44	42,50	22,62	3,17	23,00	14,60	9,20	37,64	11,89
5	Measures	68,82	77,30	21,04	2,53	59,40	52,40	21,60	132,72	71,00
	Sensors	30,21	39,22	21,64	2,61	20,20	13,40	8,60	36,34	11,94
6	Measures	75,76	85,22	23,00	2,86	65,80	57,80	24,00	146,95	77,11
	Sensors	33,09	42,87	22,54	2,71	23,80	15,40	10,00	41,96	13,32
7	Measures	64,66	75,15	20,89	2,64	56,60	50,20	20,00	121,30	68,78
	Sensors	28,74	37,15	20,38	2,54	20,00	13,20	7,60	32,36	11,45
8	Measures	46,35	54,77	20,91	2,42	38,00	31,60	28,80	29,33	35,12
	Sensors	22,43	31,00	21,19	2,53	13,60	6,60	4,00	4,47	3,60

	TABELA	SHELL MR	REAL	USER	SYS	APP	JOB	AVG MAP	TT ALL MAPS	CPU TIME
MÉDIA DOS TESTES	Measures	63,49	72,78	21,58	2,86	53,55	46,40	28,18	125,85	69,54
	Sensors	31,78	43,25	21,29	3,11	22,48	13,43	8,28	33,04	11,14
TESTE 8	Measures	-17,14	-18,01	-0,67	-0,45	-15,55	-14,80	0,63	-96,53	-34,42
	Sensors	-9,35	-12,25	-0,10	-0,59	-8,88	-6,83	-4,28	-28,57	-7,53

Figura 23 - Tabela de cima - Média dos ensaios
Tabela de baixo – teste com melhor resultado face à média de todos os testes

3.3.3.3. Importações com o *NiFi*

Com a ferramenta *Apache NiFi,* criaram-se dois fluxos de dados distintos para a extração dos ficheiros *Excel* provenientes do SNIRH e para a importação das imagens satélite.

Antes de avançar com qualquer explicação dos processos de extração e transformação do *NiFi* é importante considerar que optou-se por criar uma ponte entre a *Shell Web Client*, e o disco rígido da máquina utilizada, de modo a facilitar o processo de extração dos ficheiros *Excel*, e de forma a automatizar a sua ingestão. Com esta escolha, procedeu-se à instalação de um comando *Linux* para efetuar o "*Mount*" (cria a ligação entre o sistema de ficheiros e um diretório) entre os dois ambientes. Assim, antes de utilizar qualquer processo do *NiFi* cujos ficheiros estejam no sistema de ficheiros nativo da máquina, é necessário executar o seguinte comando:

```
mount.cifs //FRANCISCO/SharedFolder /media/sharedFolder -o
 username=xxxx@outlook.com,password=xxxxxxx
```

O código acima descrito efetua o *mount* entre a pasta s*haredFolder* do sistema nativo de ficheiros da máquina e a pasta */media/sharedFolder* que corresponde à localização da pasta destino, do lado do *Hadoop.*

Para facilitar, o código foi colocado num ficheiro que o sistema operativo do lado da máquina virtual, quando arranca, lê, e por isso basta invocar o comando ***mount –a***, ficando a pasta *sharedFolder* acessível e com o conteúdo existente no *Windows*.

Numa primeira fase, efetuou-se o *download* dos ficheiros *Excel* através do *website* oficial do SNIRH. Escolheram-se apenas as estações meteorológicas que continham dados no período pretendido, perfazendo um total de 21 ficheiros, cada um para parâmetros diferentes, como a precipitação, humidade, temperatura, entre outras.

No *NiFi*, o elemento básico de um fluxo de dados (*Data Flow*), chama-se "*processor*", que pode desempenhar vários papéis, como a transformação dos dados, escrita, leitura, existindo processadores específicos para variadas tarefas. A ligação entre dois processadores faz parte do fluxo de dados porque é a ponte de ligação entre os processadores, existindo sempre a opção de ligar um processador a múltiplas ligações e consequentemente a múltiplos processadores.

A ferramenta *NiFi* tem uma interface própria no endereço ***http://localhost:9090/nifi/,*** que permite criar fluxos de dados através de componentes "*Drag & Drop*", sendo portanto uma interface amigável ao utilizador, fácil de perceber e intuitiva. Esta interface permite criar *templates*, tipicamente utilizado para separar fluxos de dados. Para facilitar o entendimento e para evitar que o fluxo se torne extenso, foram criados dois *templates*, um para os dados SNIRH e outro para as imagens satélite.

Relativamente ao *template* SNIRH, o primeiro processador é do tipo *GetFile,* que como o nome indica, vai ao encontro de um ou mais ficheiros numa determinada localização. Cada processador tem diversas configurações ajustáveis, sendo que neste primeiro, as mais importantes e utilizadas são a localização dos ficheiros na diretoria */media/sharedFolder/SNIRH* e a opção de manter o ficheiro na diretoria referida. O processador seguinte é do tipo *PutHDFS* utilizado para escrever no sistema de ficheiros *Hadoop,* os ficheiros *Excel* tal como se encontram na fonte, sem qualquer transformação, de forma a preservá-los caso exista a necessidade de os consultar. As principais parametrizações deste segundo componente são os ficheiros com a configuração do sistema de ficheiros *Hadoop*: */etc/hadoop/conf/core-site.xml,/etc/hadoop/conf/hdfs-site.xml;* a diretoria de destino no *HDFS /user/nifi/raw;* e a opção de substituir o ficheiro caso já exista. O passo seguinte é composto por um processador do tipo *ReplaceText,* que é usado para retirar as primeiras quatro linhas e as últimas três dos ficheiros extraídos. Esta

remoção de linhas é utilizada para eliminar o cabeçalho, e outros dados que não têm importância no carregamento da tabela. Para conseguir esta remoção, foi usado no tipo de remoção "*Regex Replace*", que nos dá a possibilidade de recorrer a expressões regulares, utilizadas para identificar num conjunto de caracteres qualquer padrão de interesse. A expressão utilizada foi a seguinte: **(\A(.*\r?\n){4}|(\r?\n.*){3}\z)** . No caso de sucesso ou insucesso neste passo, é sempre copiado para o *HFDS* o resultado deste processador.

No processador seguinte do tipo *Execute Script,* foi utilizado um *script* em *Python* para adicionar uma coluna aos ficheiros, de modo a que essa coluna fique preenchida com o nome do ficheiro. Uma vez que todos os ficheiros têm as mesmas colunas, só os conseguimos distinguir pelo seu nome, por isso, existiu a necessidade de introduzir esta nova coluna. O *script* encontra-se no anexo 6.3.

Do componente anterior resulta outro que escreve no *HDFS* os ficheiros, caso falhem na adição da coluna. No caso de sucesso, um processador do tipo *MergeContent* efetua a junção de todos os ficheiros num só, usando como delimitador o nome do ficheiro. Em caso de sucesso ou de falha, também é guardado este ponto intermédio no *HDFS.*

Os próximos e últimos dois passos são os que permitem a escrita do ficheiro final, numa tabela *Hive.* Optou-se por utilizar o *Hive* porque é a ferramenta que mais se assemelha a um *Data Warehouse* e, por isso, é crucial os dados estarem organizados num modelo de dados que segue as características de um *Star Schema* (modelo em estrela), para que seja possível visualizar os dados sob a forma de factos e dimensões que permitam um exploração fácil numa ferramenta de visualização de dados.

Os dois componentes que agem em conjunto e que permitem a escrita dos dados numa tabela *Hive* são os seguintes: *ConvertCSVToAvro* e *PutHiveStreaming.*

O processador *PutHiveStreaming* é totalmente dependente do *ConvertCSVToAvro*, porque os dados têm que estar no formato *Avro. Avro* é um formato de ficheiro conhecido no âmbito de *Big Data* e *streaming*. Um ficheiro deste tipo possui o esquema de dados visível em *JSON* (*JavaScript Object Notation*), contendo vários tipos de dados à disposição, incluindo tipos complexos como mapas e *arrays.*

Uma desvantagem ao usar este tipo de componente, passa por definir o esquema dos dados "*A Priori",* e consequentemente conhecer bem os dados que estamos a lidar. Outra desvantagem é o conhecimento técnico necessário para a criação de um ficheiro *Avro*.

Para ultrapassar este problema existe um processador *InferAvroSchema,* que infere dos dados o esquema em *Avro*, no entanto ao utilizar este componente os dados não eram

bem convertidos, porque na presença de valores nulos não foram convertidos para o tipo correto, e por isso, utilizou-se o *script* no anexo 6.3 para criar o esquema corretamente.

O último processador insere os dados numa tabela previamente criada no *Hive*. Basta indicar no parâmetro *Hive MetaStore URI*: *thrift://sandbox.hortonworks.com:9083*, indicar o nome da base de dados no *Hive* e o nome da tabela.

O *script* seguinte cria a tabela *weathersations:*

```
CREATE TABLE IF NOT EXISTS Innovine.weathersations(
        Data                                STRING,
        Albufeira_do_alqueva                DOUBLE,
        FLAG1                               STRING,
        Albufeira_do_alqueva_mourao         DOUBLE,
        FLAG2                               STRING,
        Montoito                            DOUBLE,
        FLAG3                               STRING,
        Portel                              DOUBLE,
        FLAG4                               STRING,
        Reguengos                           DOUBLE,
        FLAG5                               STRING,
        Santiago_o_maior                    DOUBLE,
        FLAG6                               STRING,
        Sao_mancos                          DOUBLE,
        FLAG7                               STRING,
        FileName                            STRING)
    CLUSTERED BY (Data) INTO 3 BUCKETS
    ROW FORMAT DELIMITED
    STORED AS ORC
    LOCATION  '/user/hive/Innovine/weatherstations'
    TBLPROPERTIES('transactional'='true');
```

Nesta tabela podemos encontrar a data em que foi medido determinado valor, os valores para cada estação meteorológica, intercalado por uma "*Flag*" que indica se o valor foi calculado e o nome do ficheiro que corresponde ao parâmetro do valor (precipitação, humidade, temperatura, etc).

Existem três requisitos que uma tabela tem que cumprir para o processador *PutHiveStreaming* conseguir inserir os dados diretamente numa tabela *Hive*:

- A tabela deverá estar armazenada como *ORC* (*Optimized Row Columnar);*
- A propriedade transacional tem que estar no modo "*True*";
- A tabela deverá estar particionada, sendo que neste caso encontra-se particionada pela data.

O *template* seguinte referente às imagens satélite tem um fluxo de dados mais simples que o anterior. O primeiro passo consistiu em dar estrutura a este conjunto de imagens. Para cada imagem, fez-se o *download* e colocou-se num repositório *online.* Como não é necessário fazer transformações sobre a imagem, apenas necessitamos de um *url*, para poder ser lido do lado da ferramenta de visualização de dados – *Power BI*. Assim, criou-se um ficheiro *csv,* apenas com um identificador, a data da imagem, o tipo de processamento que a imagem levou e o respetivo *url*.

Depois, aplicando a mesma metodologia que o *template* anterior, cria-se um fluxo de dados que irá ler da pasta partilhada, e desta vez utilizando o processador *InferAvroSchema* (já não existem valores nulos), podemos inferir o esquema *Avro* automaticamente para ser lido pelo próximo componente que escreve numa tabela *Hive*.

As imagens dos dois *templates* e o *script* completo para a criação das duas tabelas, encontra-se no anexo 6.3.

3.3.3.4. Importações com o *Hive*

As restantes tabelas importadas, tiveram início apenas no *Hive*, não sendo utilizadas ferramentas para transformar os dados como o *NiFi*. A principal razão resume-se ao facto de serem tabelas pontuais, que são importadas apenas uma vez. Como exemplo, temos a tabela de dimensão data - *dimDate* que está presente num ficheiro *csv* que posteriormente foi importado para a tabela *Hive*.

Desta vez, utilizou-se a funcionalidade de "*Drag & Drop*" a partir da interface *Ambari* que permite arrastar ficheiros do sistema operativo nativo para o *HDFS*.

Depois de termos o ficheiro *csv* no *HDFS* podemos então criar a tabela e proceder à inserção dos dados.

A tabela *dimDate* foi criada como externa para preservar os ficheiros caso seja apagada. Ao contrário das tabelas criadas para os processos do *NiFi*, na omissão da palavra "*External*", a tabela é "*Internal*", o que significa que quando for apagada, os ficheiros com os dados e a pasta onde se localiza também serão apagados.

Para procedermos à inserção, uma vez que não utilizamos o componente do *NiFi*, é explícito através do seguinte código a localização do ficheiro no *HDFS* e a tabela destino:

```
LOAD DATA INPATH '/user/hive/Innovine/dimDate/Dim_Date.csv' INTO TABLE
Innovine.dimDate;
```

O facto da língua portuguesa possuir acentuações nas palavras, faz com determinados caracteres sejam especiais e portanto, caso não seja alterada a definição de codificação da tabela estes caracteres são completamente ilegíveis, tornando o texto de difícil leitura. O seguinte código define o tipo codificação para *UTF-8* para lidar com estes casos especiais:

```
ALTER TABLE Innovine.dimDate SET SERDEPROPERTIES ('serialization.encoding'='UTF-8');
```

As próximas tabelas *Hive* não se pode dizer que são importadas, mas agem como "*views*" (semelhante a uma *view* em *SQL*) sobre as tabelas *HBase.* Começa-se por definir uma tabela externa sem qualquer diferença em relação às anteriores. Depois, a propriedade "*STORED BY*", é definida com o *HBaseStorageHandler,* que permite fazer a ligação entre as tabelas *HBase* com a *metastore* (repositório de metadados) do *Hive.* A fase seguinte é responsável pelo mapeamento do esquema da tabela *HBase*, ao mapear as colunas, com as definidas no *Hive*. Um exemplo genérico do mapeamento é o seguinte:

WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,f:c1,f:c2')

Para a *rowkey*, existe denotação própria passando o argumento *:key.* O "f" corresponde ao nome da família de colunas e o "c1" e "c2" correspondem aos qualificadores de colunas em *HBase.*

O código seguinte corresponde à criação da tabela *measures* que está presente no *HBase*. O *script* com as restantes tabelas encontra-se no anexo 6.4.

```
CREATE EXTERNAL TABLE Innovine.measures(id_measure STRING, measure DOUBLE)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,measure:measure')
TBLPROPERTIES ('hbase.table.name' = 'measures');
```

A tabela que foi importada pelo *NiFi* referente aos dados SNIRH, não apresenta ainda o modelo de uma tabela factual, por isso, criou-se uma tabela d*imParameter* que contém os nomes dos ficheiros, para mais tarde ser utilizado como dimensão no *Power BI*.

Após a criação da tabela, verificou-se a necessidade de recorrer a uma função incremental tal como existe no *SQL*, a funcionalidade *Identity*, que permite gerar um número incremental na inserção de registos. Sem recorrer à função *row_number()* existente no *SQL* e no *Hive*, as possibilidades de gerar um número incremental são bastante reduzidas, sendo que esta função tem um comportamento bastante lento ao ser utilizado no *Hive*, mesmo quando são poucos registos a inserir.

Para ultrapassar esta dificuldade, o *Hive* permite criar funções denominadas *UDFs* (*User-Defined Functions*). O processo para a criação de uma função *UDF* é moroso e trabalhoso porque é necessário recorrer a um programa que seja capaz de gerar um ficheiro com extensão *.jar,* assim como dominar a linguagem *Java* para produzir o código da função. Depois de produzido o código necessário e após exportar o ficheiro *.jar*, é necessário importá-lo para o *HDFS* e depois importar a função para o *Hive* de modo a ser possível utilizá-la.

Para produzir o código e exportar a classe que contém o código, utilizou-se o programa *Eclipse Luna*, sendo que o código da função resume-se ao seguinte:

```
public static void main(String args[])
    {/*Empty Main - Do nothing*/}

private LongWritable result = new LongWritable();

public UDFHiveIdentity() {
       result.set(0);
}

public LongWritable evaluate() {
       result.set(result.get() + 1);
       return result;
}
```

Para importar a função às bibliotecas já existentes do *Hive*, é utilizado o seguinte comando:

```
CREATE TEMPORARY FUNCTION sequence AS
'org.apache.hadoop.hive.contrib.udf.UDFHiveIdentity' USING JAR
'hdfs:///tmp/UDFHiveIdentity.jar';
```

Para invocar a função basta escrever *sequence()* na cláusula *"select".* O código abaixo ilustra a inserção de registos na tabela *dimParameter* usando a potencialidade da função.

```
INSERT INTO Innovine.dimParameter
select Sequence() AS Parameter_ID, filename as Parameter from (
select distinct filename from Innovine.weatherstations) a;
```

Com a *dimParameter* criada, resta fazer a otimização de colocar o nome das estações meteorológicas numa dimensão, transformar os dados das colunas em linhas na tabela original e, por fim, atribuir uma *Surrogate Key* (chave que liga a dimensão à tabela de factos) à tabela.

Para se conseguir colocar os valores das colunas em linhas recorreu-se a funções próprias do *Hive* como o "*lateral view*" e "*explode*" que facilitam esta tarefa.

Recordando o esquema original da tabela resultante dos processos aplicados no *NiFi*:

Estacao1	**Estacao2**	**EstacaoN**	**NomeFicheiro**
Val1	Val1	Val1	Temperatura
Val2	Val2	Val2	Humidade

Tabela 7 - Esquema original da tabela *Weatherstations*

Ao aplicar as funções descritas obtém-se um esquema do tipo:

Estacao1	Val1	Temperatura
Estacao1	Val2	Humidade
Estacao2	Val1	Temperatura
Estacao2	Val2	Humidade
EstacaoN	Val1	Temperatura
EstacaoN	Val2	Humidade

Tabela 8 - Esquema resultante após transformação de colunas para linhas

O código que efetua a transformação aplica no momento as *Surrogate Keys* no nome da estação e no nome do ficheiro. É possível consultá-lo no anexo 6.4, assim como o código completo da criação da função *sequence()*, e todo o *script* de criação e carregamentos de tabelas.

3.3.3.5. Scrapy

Como referido no capítulo 3.1.4 Mercado de venda, a ferramenta eleita para extrair os dados relativos aos preços dos vinhos foi o *Scrapy*. O pequeno projeto elaborado para efetuar o *Web Scraping* é composto pelos seguintes ficheiros:

- *maiscarrinhoSpider.py*
- *items.py*
- *pipelines.py*

- *settings.py*

O ficheiro *maiscarrinhoSpider.py* corresponde à classe *Spider* onde se encontra o código que efetua a extração automática das páginas. A metodologia abordada consiste em inspecionar a página do *maiscarrinho*, e no separador *Network* e *XHR* (*XMLHttpRequest*), é possível visualizar os pedidos internos realizados a um servidor *Web*, e em resposta é recebido um *script* em formato *JSON* que nos permite fazer a extração na classe *Spider*. No lado esquerdo da figura 24, podemos observar o *url* correspondente às páginas 2 e 3 da pesquisa feita por "vinhos". No lado direito da figura 24, visualizamos a vermelho, a resposta devolvida pelo servidor no formato *JSON*.

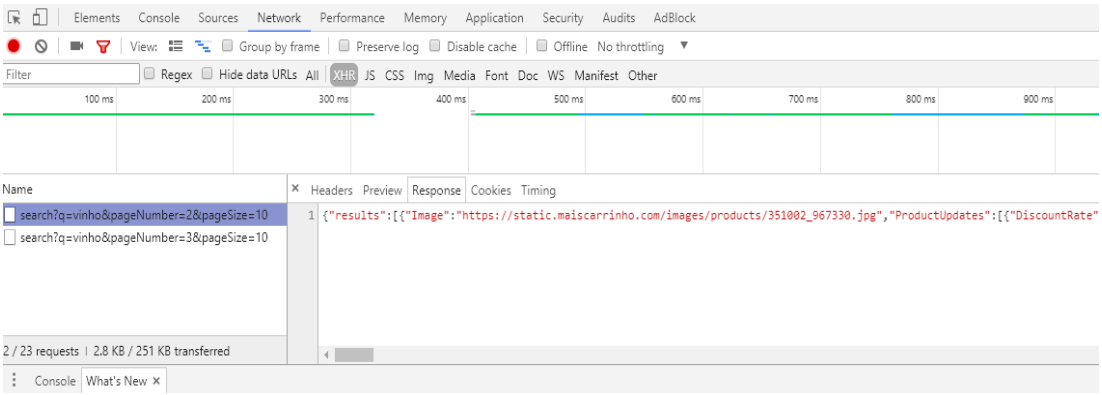


Figura 24 - Pedido *XHR* - Maiscarrinho

O método pelo qual o *website* gera uma nova página, não se realiza através de um botão de paginação, mas sim, automaticamente através do *scroll* da página, isto é, ao baixarmos a página para ver novos conteúdos, a página seguinte é mostrada automaticamente, gerada através de um novo pedido interno *XHR*. O *Spider* captura esta informação através de um ciclo em que percorre todas as páginas com conteúdo, parando quando deteta que não existe conteúdo a extrair.

A classe *items.py* contém a definição de objetos que são usados no *Spider* para guardar a informação dos elementos extraídos.

Os elementos extraídos são encaminhados para a classe *pipeline.py*, responsável por introduzir os dados em tabelas no *SQL Server*.

O modelo de dados que compõe as tabelas resultantes da extração dos dados são as seguintes:

- *Wines*
- *Supermarket*
- *WinesNames*

A tabela *Wines* representa a tabela factual onde são guardados os preços dos vinhos. *Supermarket* corresponde a uma dimensão com os supermercados existentes, e *WinesNames* representa outra dimensão com o nome dos vinhos, a sua marca e o seu tipo.

Para passarmos os dados para o *data lake*, investigou-se outra técnica de importação, que consiste na importação direta para o *Hive* através da ferramenta *Sqoop*. Utilizando esta técnica, obtém-se a vantagem das tabelas *Hive* serem criadas automaticamente no momento da importação, apenas se passarmos o parâmetro "*import-hive*".

O código seguinte ilustra o exemplo da importação da tabela *Wines* para a *fact_Wines* no *Hive*.

```
sqoop import --connect "jdbc:sqlserver://169.254.69.51:1433;
database=MaisCarrinho" --username hadoop
--password-file file:///root/sqoop.password --table Wines
--target-dir /user/hive/Innovine/fact_Wines --hive-import
--hive-table Innovine.fact_wines -m 1
```

A classe seguinte denominada *settings.py* corresponde às configurações do projeto, onde se encontra o parâmetro que obedece às regras do ficheiro *robots.txt*.

Para invocar o comando de extração via linha de comandos, o código é o seguinte:

```
activate py27
scrapy crawl SpidyWine
```

Primeiro ativamos o ambiente pré-configurado com a versão do *Python 2.7*, e de seguida, podemos efetuar o comando de extração.

O código utilizado no projeto *scrapy* encontra-se no anexo 6.5, assim como a imagem completa da fig. 24, o *output* da consola após a invocação do comando, e o código *Sqoop*.

3.3.4. Segurança

No âmbito da segurança foram implementadas ao longo do projeto boas práticas, como o exemplo da *password* em ficheiro na ferramenta *Sqoop*, mudanças de passwords das ferramentas ou até criação de utilizadores. No entanto, estas mudanças fazem parte de uma minoria de ações que se podem implementar em segurança num *data lake*.

Podem existir vários tipos de utilizador como executivos, analistas de negócio ou cientistas de dados (LaPlante & Sharma, 2016). A segurança nestes casos deve ter em conta políticas de segurança para cada tipo de utilizador. É importante separar o que cada utilizador pode ver porque pode existir informação sensível que não deve estar disponível para a grande maioria dos utilizadores. Imaginando o caso de um cientista de dados, que necessita de lidar diariamente e que pode lidar com grande parte dos dados presentes no *data lake*, este utilizador deverá ter políticas de segurança flexíveis que lhe permitem aceder aos dados com menos restrições e com mais rapidez.

É possível fazer este tipo de separação com a combinação das ferramentas *Ranger* e *Knox*.

Antes de lidar com as políticas, criou-se um utilizador *guest* para aceder à interface *Ambari*, que tem acesso a duas vistas – *HDFS* e *Hive*. Este utilizador pode consultar tabelas *Hive*, efetuar *queries*, pode ver os ficheiros presentes no *HDFS*, mas não pode alterar configurações das ferramentas pertencentes ao ecossistema *Hadoop*.

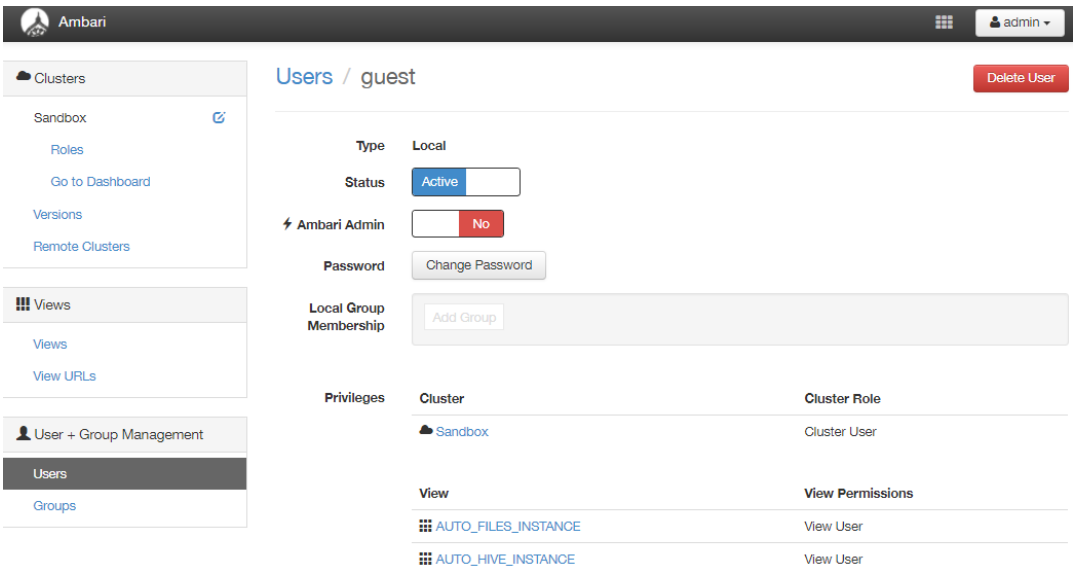


Figura 25 - Utilizador *guest* - *views*

Com o utilizador criado, a fase seguinte consiste em definir as políticas de segurança na central de administração no *Apache Ranger*. A ferramenta *Apache Knox* desempenha um papel de *gateway* (porta de acesso) aos serviços *Hadoop* como o *HDFS* e *Hive*. Por isso, um dos requisitos para o *Ranger* iniciar é o serviço *Knox*, que podemos inicializar através do *Ambari*. O serviço *Ambari Infra* também é necessário estar ligado.

Através do *url* ***http://127.0.0.1:6080/*** podemos aceder à página de autenticação do *Ranger*. Após o *login*, a imagem do anexo 6.6 ilustra a página inicial da ferramenta, onde podemos observar o gestor de serviços que contém os serviços possíveis de definir políticas de segurança. São também mostradas imagens que ilustram as configurações das políticas de segurança *Hive* assim como imagens que reproduzem o erro quando não existem permissões suficientes para aceder aos serviços.

A política ilustrada no anexo 6.6, de nome "*Data Readers + Admin*", define permissões apenas de leitura para o utilizador "*guest*", podendo este efetuar consultas do tipo "*select*", enquanto os utilizadores *hive*, *ambari-qa* e *admin*, têm total permissão sobre as tabelas *Hive*, nas quais poderão efetuar *selects*, *updates*, *creates*, *drops*, etc.

Além do *Hive* foram estabelecidas políticas de segurança nos serviços *HDFS* e *HBase*. Restando a ferramenta *NiFi*, não foi possível estabelecer as permissões por não existir o serviço instalado no *Apache Ranger*. Contudo, em versões superiores ao *HDP 2.5* a ferramenta *Ranger* já inclui nativamente o serviço *NiFi*.

3.3.5. Visualização de dados

3.3.5.1. Power BI

A jornada do *data lake* termina com a ferramenta *Power BI*, que permite aos utilizadores explorarem a informação numa experiencia imersiva rodeada de gráficos apelativos e fáceis de navegar.

O facto de se conseguir estabelecer uma ligação entre os dois ambientes – *Data Lake* e *Power BI* é um bom indício que as arquiteturas modernas de dados que lidam com "*Big Data*", estejam cada vez mais a serem postas em prática por parte das organizações.

O *Power BI* é uma ferramenta com um grande domínio e visibilidade no âmbito das ferramentas de "*Business Intelligence and Analytics*", considerada novamente líder no ano 2017, pela *Gartner* (Josh Parenteau, Rita L. Sallam, Cindi Howson, Joao Tapadinhas, Kurt

Schlegel, 2016). A figura 26 demonstra o posicionamento da *Microsoft* no quadrante *Gartner*.

Figura 26 Quadrante Mágico - *Gartner* 2017

Os requisitos para a implementação de um *dashboard Power BI* com ligação ao *data lake* são os seguintes:

- *Sandbox Hortonworks*
- *Driver* ODBC para o *Apache Hive*
- *Power BI* online ou *on-premises*
- *Dataset*

Os conjuntos de dados que alimentam o Power BI localizam-se na base de dados *Innovine,* na ferramenta *Hive*. As tabelas carregadas ao longo do projeto tiveram como destino final o *Data Warehouse* localizado no *Hive*. As tabelas assumem neste ponto um modelo em estrela (*Star Schema*), em que temos um conjunto de dimensões e um conjunto de tabelas factuais.

Após a instalação do *driver ODBC*, é necessário configurar parametrizações como o *Host* da *Sandbox*, porta, base de dados do *Hive*, mecanismo de autenticação e o utilizador. A figura 27 ilustra a configuração deste *driver*:

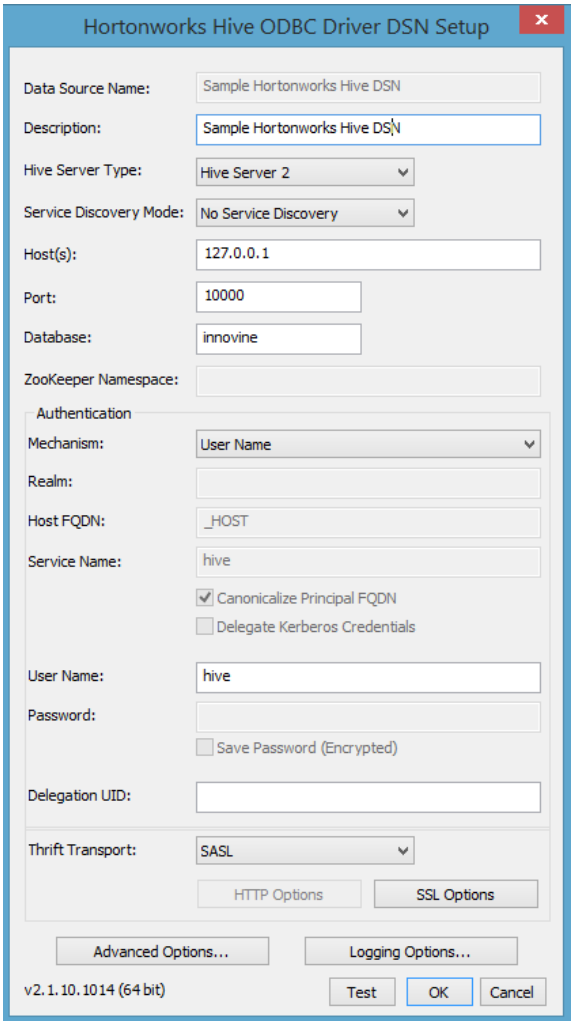


Figura 27 - *Hortonworks Hive ODBC Driver* - Configuração

Com a configuração testada com sucesso, a etapa seguinte consiste em criar o conector *ODBC* no lado do *Power BI* para se iniciar a importação dos dados. Depois, ao efetuar-se as ligações necessárias entre as tabelas obtemos o modelo final de dados ilustrado no anexo 6.7.

Foram utilizadas algumas expressões em *DAX* (*Data Analysis Expressions*) para obter novas colunas e efetuar transformações sobre os dados.

Um exemplo são as tabelas *HBase* que concatenam a chave composta com o caracter "_", e por isso recorreu-se ao *DAX* para extrair da *rowkey* os campos em separado. O código seguinte extrai da *rowkey* a parte correspondente ao descritivo do agregado.

```
aggr=LEFT(RIGHT(fact_weathermeasures[id_weathermeasure];
LEN(fact_weathermeasures[id_weathermeasure]) –
FIND("_";fact_weathermeasures[id_weathermeasure];1));
FIND("_";RIGHT(fact_weathermeasures[id_weathermeasure];
LEN(fact_weathermeasures[id_weathermeasure]) –
FIND("_";fact_weathermeasures[id_weathermeasure];1));1)-1)
```

O modelo de dados em estrela do *Power BI* é o seguinte:

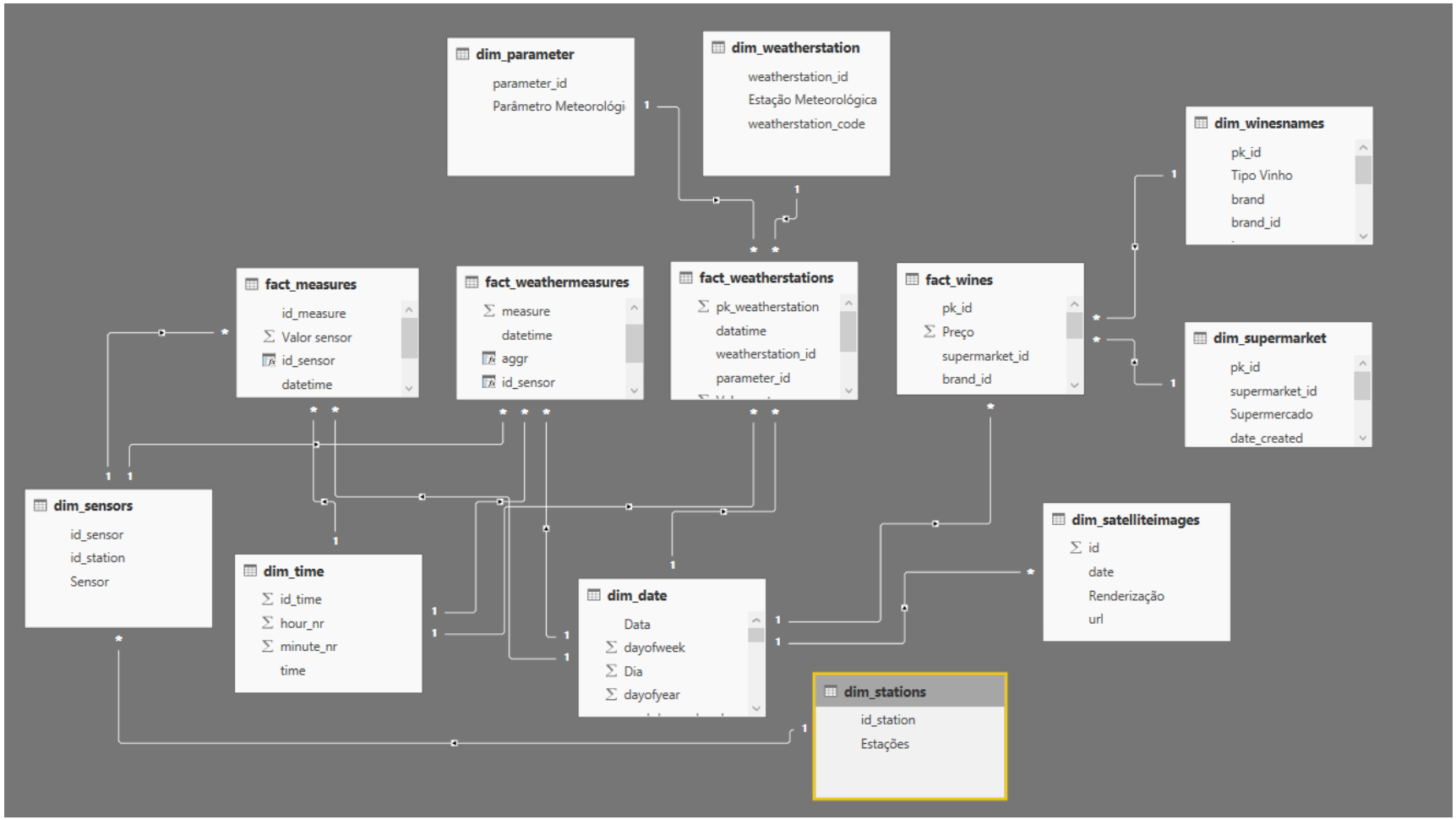


Figura 28 - *Power BI* – Modelo de dados

O *Dashboard* criado contém 3 páginas diferentes, cada uma para diferentes tipos de análise:

1. Análise Comparativa
2. Análise por Imagem
3. Análise por Mercado

Na primeira página, "Análise Comparativa", é possível cruzar os dados recolhidos do SNIRH com os dados sensoriais no terreno das castas. A figura 28 mostra informação relativa a Junho de 2015, estando selecionado o parâmetro meteorológico "Temperatura do ar horário" e o sensor "Temperatura bago 1 #601". Podemos observar que existe uma relação direta entre as duas métricas no gráfico de barras. Os valores das barras que correspondem ao valor meteorológico são bastante próximos dos valores da linha que são os valores registados pelos sensores. Este é apenas um tipo de análises que se pode retirar deste mapa, tendo muitos parâmetros à escolha. No canto superior direito observamos as médias registadas pelo sensor e pela estação meteorológica.

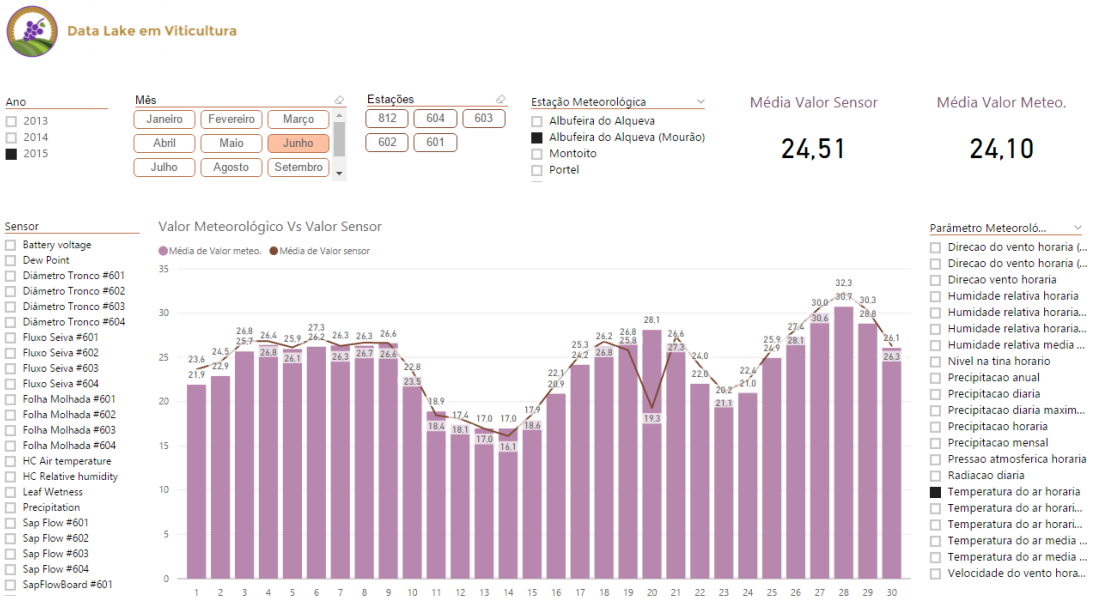


Figura 29 - *Power BI* - Análise Comparativa

capturadas, um bom indicativo do estado das castas na Herdade do Esporão. É possível escolher vários índices além do selecionado, como o infravermelho, o índice de água no solo ou até a cor verdadeira da imagem. No canto superior direito, podemos acompanhar a imagem com métricas que representam a máxima temperatura e precipitação registadas, no dia selecionado que corresponde ao dia em que a imagem foi tirada.

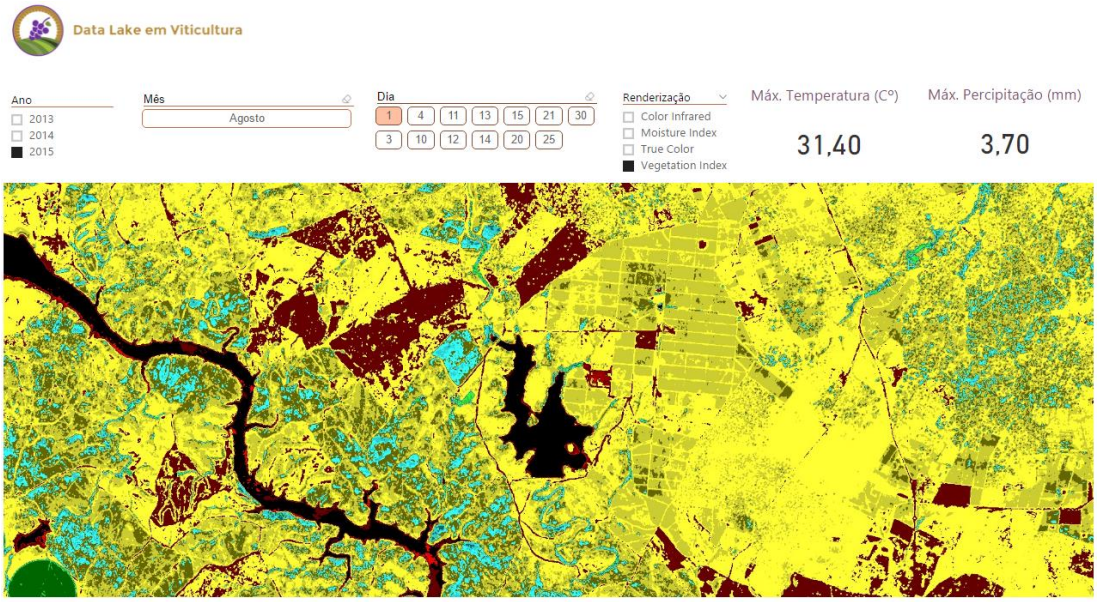


Figura 30 - *Power BI* – Análise por Imagem

A terceira e última página – "Análise por Mercado" – figura 30 – usa dados provenientes da extração efetuada com *Scrapy*, em que podemos observar o histórico dos preços dos vinhos e ficamos com uma ideia do seu preço máximo e mínimo num determinado período. No gráfico de linhas é possível subir ao nível do mês e do ano. A única desvantagem deste conjunto de dados é que não permite fazer um cruzamento direto entre os restantes, porque o período disponível do preço dos vinhos é referente apenas à segunda metade do ano 2017.

Figura 31 - *Power BI* - Análise por Mercado

4. CONCLUSÕES

Com o atual projeto implementado atingiu-se o principal objetivo de criar uma arquitetura de dados com o seu principal componente – *Data Lake*, recorrendo a tecnologias de vanguarda em que decorrem ainda inúmeras investigações sobre a sua usabilidade, no âmbito de *Big Data*.

O conjunto de dados utilizados para este projeto é proveniente do setor vitícola, uma área que não deve ser desprezada em investigações, visto ter uma grande importância em Portugal.

Conseguiu-se elaborar um projeto funcional que retrata para um conjunto de dados de várias naturezas, os mais variados processos que podem existir num *Data Lake*, desde as técnicas de importação, transformações, até à fase final de visualização de dados.

Para retratar estes processos foram utilizados um conjunto de ferramentas diferentes, desde o componente principal - *Sandbox Hortonworks*, até às ferramentas do ecossistema *Hadoop*. Foi descrito em geral as suas principais funcionalidades, embora se tenha reduzido o nível de detalhe em algumas pois cada ferramenta tem bastante material para se aprofundar.

Com o projeto concluído, é possível efetuar o cruzamento de todos os dados recolhidos através de *Dashboards* elaborados com o *Power BI*, com opção de se investigar até à fonte, pois a ideia base do *data lake* é reter todos os dados desde o início até ao fim do seu ciclo.

Em suma, a prova de conceito foi com sucesso implementada e demonstrada a sua usabilidade e versatilidade perante dados de um setor pouco comum no âmbito de soluções de *Big Data,* mas que faz todo o sentido apostar na ideia de desenvolver soluções para o setor vitícola.

Foi necessário uma forte competência técnica durante o desenvolvimento do projeto por se utilizar tecnologias de vanguarda que ainda estão em desenvolvimento e sob investigação. Muitas das vezes foi necessária uma interação com a comunidade *Hortonworks*, questionando partes técnicas do projeto.

Ao longo do projeto existiram dificuldades como instalações e erros inesperados da *Sandbox*, contudo, foi sempre possível ultrapassar estes problemas através de investigação e com a ajuda recorrente da comunidade *Hortonworks*.

4.1. *LIMITAÇÕES E RECOMENDAÇÕES PARA TRABALHOS FUTUROS*

Com este projeto, e dada a sua potencialidade de expansão, existem diversos pontos onde seria possível melhorar:

1. Alargamento das fontes de dados (nomeadamente nos processos culturais das vinhas, e dos produtos provenientes dos processos);
2. Testar o *Data Lake* com um maior volume de dados e verificar se o comportamento não se degrada nas ferramentas utilizadas;
3. Criar *workflows* de automatização com o *Apache Oozie;*
4. Explorar as possibilidades da receção dos dados em tempo real para tabelas *HBase;*
5. Utilizar o *Apache Atlas* no âmbito de "*Data Governance and Lineage*", de forma a ser possível classificar e organizar os metadados. Este é um ponto importante que não foi possível implementar porque o *Atlas* ainda não é compatível com determinadas ferramentas utilizadas no projeto como o *Sqoop* (quando o destino é o *HBase*) ou o *NiFi* (existe apenas processadores criados pela comunidade, mas nativamente, o *Atlas* não suporta o *NiFi*);
6. Automatizar a importação das imagens satélite para o *Data Lake;*
7. Automatizar a execução do *Spider* que efetua a extração *web*, utilizando ou o *task scheduler (*agendamento de tarefas*)* do sistema operativo nativo, ou recorrendo ao *ScrapingHub* que é uma ferramenta *online* derivada do projeto *Scrapy*, que permite automatizar e parametrizar a execução do *Spider* (vertente paga se existirem módulos não pertencentes ao *Scrapy*);
8. Configurar nas ferramentas utilizadas, autenticações *Kerberos* para maior segurança no *Data Lake;*
9. Criação de novos *Dashboards* consoante as análises pretendidas;
10. Análise crítica do cruzamento dos dados obtidos.

O projeto tem limitações por se utilizar uma *Sandbox* que não passa de um ambiente de desenvolvimento para testar as ferramentas do ecossistema *Hadoop*, num modo pseudo-distribuído. Devido a este facto, não se implementou políticas de armazenamento no *HDFS,* pois implicaria armazenar consoante a importância dos dados e a frequência que são acedidos, em discos rígidos, *SSDs* ou RAM, algo que não faria sentido com a máquina de ambiente de teste, que não possui os requisitos necessários para a sua implementação.

5. BIBLIOGRAFIA

About Bill : William H. Inmon, “The Father of Data Warehousing.” (2007). Retrieved May 2, 2016, from http://www.inmoncif.com/about/

Anne-Françoise Adam-Blondon. (2013). INNOVINE - Innovation in Vineyard. Retrieved April 30, 2017, from http://www.innovine.eu/

Barreira, E. da C. M. (2014). *População e Enriquecimento de Ontologias através de Web Scraping*. Instituto Superior de Engenharia do Porto.

Borthakur, D. (2013). HDFS Architecture Guide. Retrieved February 4, 2017, from http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#NameNode+and+DataNodes

CCDR-A com êxito no POR; Alentejo consegue 100% de absorção dos fundos. (2016). *Diário Do Sul*, 16. Retrieved from https://issuu.com/diariodosul/docs/ds_-_dia_29_04_2016%0A

Cood, E. F. (1969). A Relational Model of Data for Large Shared Data Banks. *IBM Research Laboratory, California*, *13*(6). Retrieved from http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

Dirk deRoos, Paul Zikopoulos, B. B., & Rafael Coss, R. M. (2014). *Hadoop for Dummies* (1st ed., Vol. 65). John Wiley & Sons, Inc.

ESA. (2017a). ESA EO Missions - Sentinel-1. Retrieved from https://earth.esa.int/web/guest/missions/esa-operational-eo-missions/sentinel-1

ESA. (2017b). Observing the Earth -Copernicus. Retrieved August 6, 2017, from http://www.esa.int/Our_Activities/Observing_the_Earth/Copernicus/Overview4

ESA. (2017c). SENTINEL Missions. Retrieved August 6, 2017, from https://earth.esa.int/web/sentinel/missions

Esporão SA Relatório 2015. (2015).

Fletcher, K. (2012). *Sentinel-2 : ESA’s optical high-resolution mission for GMES operational services*. ESA Communications ESTEC, PO Box 299, 2200 AG Noordwijk. Retrieved from http://www.esa.int/About_Us/ESA_Publications/ESA_SP-1322_2_Sentinel_2

Foundation, T. A. S. (2017). Apache Sqoop. Retrieved from http://sqoop.apache.org/

Gao, B. (1996). NDWI—A normalized difference water index for remote sensing of vegetation liquid water from space. *Remote Sensing of Environment*, *58*(3), 257–266. http://doi.org/10.1016/S0034-4257(96)00067-3

George, L. (2010). *HBase: The Definitive Guide*. *O'Reilly* (2nd ed.). O'Reilly Media, Inc.

Granite, B. (2015). *Data Lakes in a Modern Data Architecture*. Retrieved from https://www.blue-granite.com/data-lakes-in-a-modern-data-architecture-ebook-by-bluegranite

Hortonworks. (2014). A modern data architecture with Apache Hadoop: the journey to a data lake, (March), 18.

IBM Corporation. (2014). E .& J . Gallo Winery. NY. Retrieved from http://www-03.ibm.com/software/businesscasestudies/in/en/corp?synkey=M776970P79823M49

IDC/EMC. (2014). The Digital Universe of Opportunities. *IDC / EMC Report*, 17. Retrieved from http://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf

Josh Parenteau, Rita L. Sallam, Cindi Howson, Joao Tapadinhas, Kurt Schlegel, T. W. O. (2016). *Magic Quadrant for Business Intelligence and Analytics Platforms*. Retrieved from https://www.gartner.com/doc/reprints?id=1-2XXET8P&ct=160204

Lang, M. (2014). *Data Preparation in the Hadoop Data Lake*. Teradata. Retrieved from http://www.teradata.com/Resources/White-Papers/Data-Preparation-in-the-Hadoop-Data-Lake/?LangType=1033&LangSelect=true

LaPlante, A., & Sharma, B. (2016). *Architecting Data Lakes*. (S. Cutt, Ed.) (1st editio). O'Reilly Media.

Machado, T. (2015). Os Vinhos De Portugal em 2015. Instituto da Vinha e do Vinho, I.P.

Michalopoulos, S. (2016). Commission promotes smart farming to mitigate climate change. Retrieved August 23, 2017, from https://www.euractiv.com/section/agriculture-food/news/commission-promotes-smart-farming-to-mitigate-climate-change/

MR. (2015, July 15). Big data : une nouvelle révolution agricole en marche. *Hebdo, Agra Presse*, pp. 1–7.

Mulla, D. J. (2013). Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps. *Biosystems Engineering*, *114*(4), 358–371. http://doi.org/10.1016/j.biosystemseng.2012.08.009

O'Brien, J. (2015). *The Definitive Guide to the Data Lake (White Paper)*. (R. A. Lindy Ryan, Research Director, Ed.). Retrieved from http://www.teradata.com/Resources/White-Papers/The-Definitive-Guide-to-the-Data-Lake/?LangType=1033&LangSelect=true

Rijmenam, M. Van. (2014). *Think Bigger: Developing a Successful Big Data Strategy for Your Business*. AMACOM Div American Mgmt Assn.

Schneider, R. D. (2013). *Hadoop Buyer's Guide*.

Soztutar, E. (2013). APACHE HBASE REGION SPLITTING AND MERGING. Retrieved August 17, 2017, from https://br.hortonworks.com/blog/apache-hbase-region-splitting-and-merging/

Stavrakaki, M. (2016). “V” for Visualizing Vineyards and Varieties data. Retrieved December 7, 2016, from https://www.big-data-europe.eu/v-for-visualizing-vineyards-and-varieties-data/

Swoyer, S. (2015). *Data Lake Principles and Economics (White Paper)*. Retrieved from https://tdwi.org/research/2015/11/checklist-data-lake-principles

T. Kathleen, & C. Jarek. (2013). *Apache Sqoop Cookbook Unlocking Hadoop for you Relational Database*. O’Reilly Media.

Teradata, & Hortonworks. (2014). *Putting the Data Lake to Work: A Guide to Best Practices (White Paper)*. Teradata. Retrieved from http://www.teradata.com/Resources/White-Papers/Putting-the-Data-Lake-to-Work-A-Guide-to-Best-Practices/?LangType=1033&LangSelect=true

The Contextual Data Lake. (2015). SAP.

Turkington, G. (2013). *Hadoop Beginner’s Guide* (1st ed.). Birmingham, UK: Packt Publishing Ltd.

Vanian, J. (2016). How IBM is Bringing Watson to Wine. Retrieved December 7, 2016, from http://fortune.com/2016/01/09/ibm-bringing-watson-wine/

White, T. (2015). *Hadoop The Definitive Guide* (4th ed.). O’Reilly Media.

Woods, D. (2011). Big Data Requires a Big, New Architecture - Forbes. Retrieved April 23, 2016, from http://www.forbes.com/sites/ciocentral/2011/07/21/big-data-requires-a-big-new-architecture/#6e98067f1d75

Woods, D. (2014). Jump in a Data Lake. Are you leaving millions on the table? *Teradata*, 40–42. Retrieved from http://www.teradata.com/Resources/Teradata-Magazine-Articles/Jump-in-a-Data-Lake/?LangType=1033&LangSelect=true

6. ANEXOS

6.1. *SCRIPT PARA CRIAÇÃO DE TABELAS HBASE*

```
create 'measures', 'measure'
sqoop import –connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table measures --
columns id_sensor,measure_time,measure --hbase-table measures --column-
family measure --hbase-row-key id_sensor,measure_time
snapshot 'measures', 'measures_snap'
disable 'measures'
drop 'measures'
```

```
create 'sensors', 'sensor'
sqoop import –connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table sensors  --
columns id_sensor,id_station,sensorName --hbase-table sensors --column-
family sensor --hbase-row-key id_sensor
snapshot 'sensors', 'sensors_snap'
disable 'sensors'
drop 'sensors'
```

```
create 'stations', 'station'
sqoop import "-D sqoop.hbase.add.row.key=true" –connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table stations --
hbase-table stations --column-family station --hbase-row-key id_station
snapshot 'stations', 'stations_snap'
disable 'stations'
drop 'stations'
```

```
create 'weatherMeasures', 'weatherMeasure'
sqoop import "-Dorg.apache.sqoop.splitter.allow_text_splitter=true"
--connect "jdbc:sqlserver://169.254.69.51:1433;database=Innovine"
--username hadoop -- password-file file:///root/sqoop.password --table
weatherMeasures --columns id_sensor,measureAggr,measure_time,measure
--hbase-table weatherMeasures --column-family weatherMeasure
--hbase-row-key id_sensor, measureAggr, measure_time
snapshot 'weatherMeasures', 'weatherMeasures_snap'
disable 'weatherMeasures'
```

```
drop 'weatherMeasures'
```

6.2. *OTIMIZAÇÃO NAS TABELAS HBASE*

Teste 1

```
create 'measures', 'measure', 'sensor'
time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table measures
--columns id_sensor,measure_time,measure --hbase-table measures
--column-family measure
--hbase-row-key id_sensor,measure_time

time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table sensors
--columns id_sensor,id_station,sensorName --hbase-table measures
--column-family sensor
--hbase-row-key id_sensor
```

Teste 2

```
create 'measures', 'measure', 'sensor', {NUMREGIONS => 10, SPLITALGO =>
'HexStringSplit'}
time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table measures
--columns id_sensor,measure_time,measure --hbase-table measures
--column-family measure
--hbase-row-key id_sensor,measure_time

time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table sensors
--columns id_sensor,id_station,sensorName --hbase-table measures
--column-family sensor
--hbase-row-key id_sensor
```

Teste 3

```
create 'measures', 'measure', 'sensor', {NUMREGIONS => 10, SPLITALGO =>
'UniformSplit'}
time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
```

```
hadoop -- password-file file:///root/sqoop.password --table measures
--columns id_sensor,measure_time,measure --hbase-table measures
--column-family measure
--hbase-row-key id_sensor,measure_time

time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table sensors
--columns id_sensor,id_station,sensorName --hbase-table measures
--column-family sensor
--hbase-row-key id_sensor
```

Teste 4

```
create 'measures', 'measure', 'sensor'
time sqoop import "-Dorg.apache.sqoop.splitter.allow_text_splitter=true"
--conect "jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table measures
--columns id_sensor_hashed,measure_time,measure --hbase-table measures
--column-family measure
--hbase-row-key id_sensor_hashed,measure_time

time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table sensors
--columns id_sensor,id_station,sensorName --hbase-table measures
--column-family sensor
--hbase-row-key id_sensor
```

Teste 5

```
create 'measures', 'measure', 'sensor', {NUMREGIONS => 10, SPLITALGO =>
'HexStringSplit'}
time sqoop import "-Dorg.apache.sqoop.splitter.allow_text_splitter=true"
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table measures
--columns id_sensor_hashed,measure_time,measure --hbase-table measures
--column-family measure
--hbase-row-key id_sensor_hashed,measure_time

time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table sensors
--columns id_sensor,id_station,sensorName --hbase-table measures
```

```
--column-family sensor
--hbase-row-key id_sensor
```

Teste 6

```
create 'measures', 'measure', 'sensor', {NUMREGIONS => 10, SPLITALGO =>
'UniformSplit'}
time sqoop import "-Dorg.apache.sqoop.splitter.allow_text_splitter=true"
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table measures
--columns id_sensor_hashed,measure_time,measure --hbase-table measures
--column-family measure
--hbase-row-key id_sensor_hashed,measure_time

time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table sensors
--columns id_sensor,id_station,sensorName --hbase-table measures
--column-family sensor
--hbase-row-key id_sensor
```

Teste 7

```
create 'measures', 'measure', 'sensor', {SPLITS => ['19', '33', '4C', '66',
'7F', '99', 'B3', 'CC', 'E6']}
time sqoop import "-Dorg.apache.sqoop.splitter.allow_text_splitter=true"
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table measures
--columns id_sensor_hashed,measure_time,measure --hbase-table measures
--column-family measure
--hbase-row-key id_sensor_hashed,measure_time

time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table sensors
--columns id_sensor,id_station,sensorName --hbase-table measures
--column-family sensor
--hbase-row-key id_sensor
```

Teste 8

```
create 'measures', 'measure', 'sensor', {SPLITS => ['19', '33', '4C', '66',
'7F', '99', 'B3', 'CC', 'E6']}
time sqoop import "-Dorg.apache.sqoop.splitter.allow_text_splitter=true"
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table measures
```

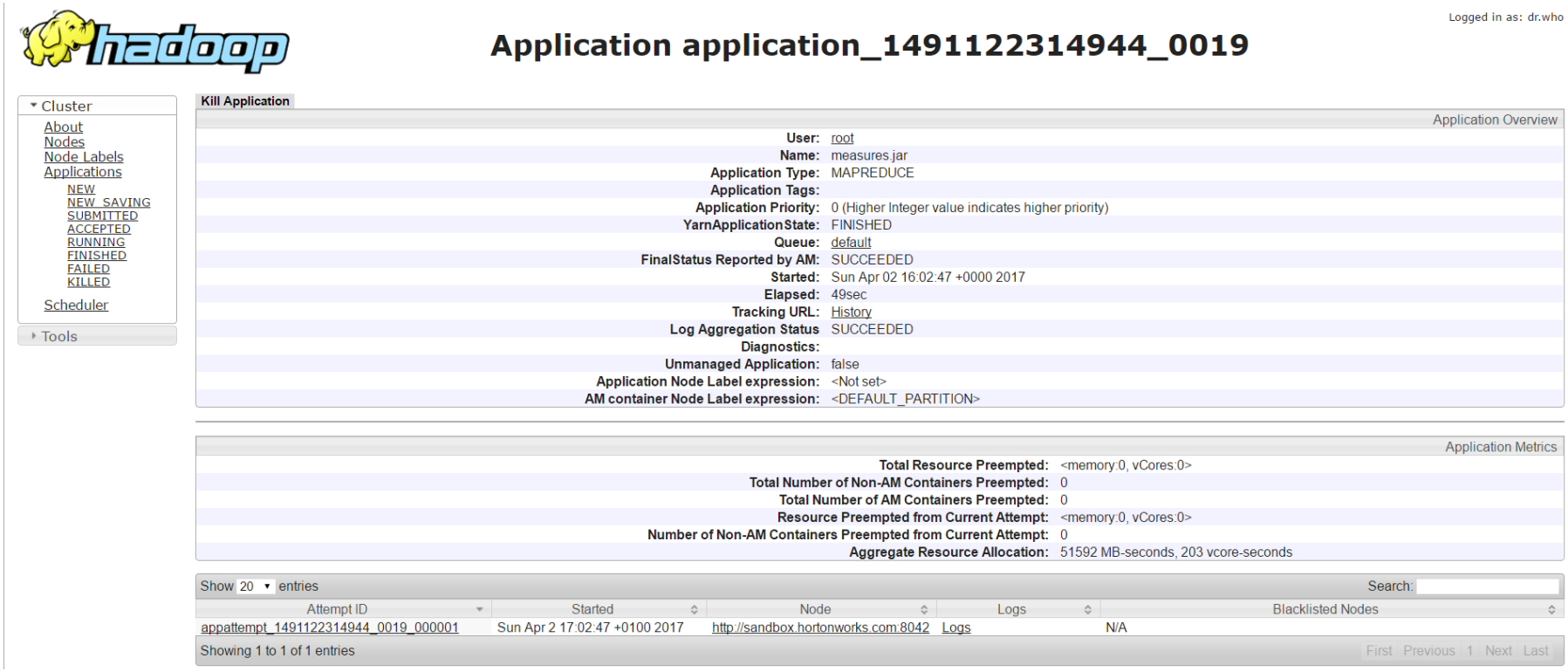
```
--columns id_sensor_hashed,measure_time,measure --hbase-table measures
--column-family measure
--hbase-row-key id_sensor_hashed,measure_time –m 1

time sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=Innovine" --username
hadoop -- password-file file:///root/sqoop.password --table sensors
--columns id_sensor,id_station,sensorName --hbase-table measures
--column-family sensor
--hbase-row-key id_sensor –m 1
```

Exemplo das métricas capturadas para os ensaios:

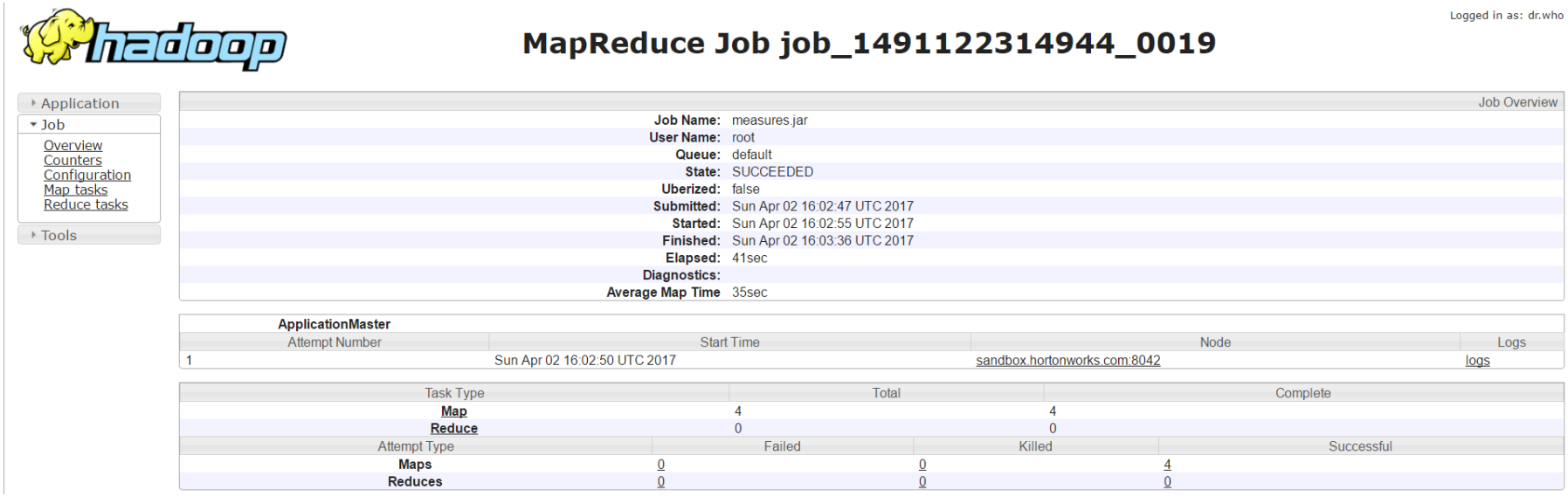
1. *SHELL MR, REAL USER* e *SYS*

```
17/04/02 16:02:48 INFO mapreduce.Job: Running job: job_1491122314944_0019
17/04/02 16:02:57 INFO mapreduce.Job: Job job_1491122314944_0019 running in uber mode : false
17/04/02 16:02:57 INFO mapreduce.Job:  map 0% reduce 0%
17/04/02 16:03:33 INFO mapreduce.Job:  map 25% reduce 0%
17/04/02 16:03:35 INFO mapreduce.Job:  map 50% reduce 0%
17/04/02 16:03:36 INFO mapreduce.Job:  map 75% reduce 0%
17/04/02 16:03:37 INFO mapreduce.Job:  map 100% reduce 0%
17/04/02 16:03:38 INFO mapreduce.Job: Job job_1491122314944_0019 completed successfully
17/04/02 16:03:39 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=786688
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=497
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=4
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=0
        Job Counters
                Launched map tasks=4
                Other local map tasks=4
                Total time spent by all maps in occupied slots (ms)=142777
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=142777
                Total vcore-milliseconds taken by all map tasks=142777
                Total megabyte-milliseconds taken by all map tasks=35694250
        Map-Reduce Framework
                Map input records=1100646
                Map output records=1100646
                Input split bytes=497
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=33678
                CPU time spent (ms)=76380
                Physical memory (bytes) snapshot=827887616
                Virtual memory (bytes) snapshot=7843360768
                Total committed heap usage (bytes)=290455552
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=0
17/04/02 16:03:39 INFO mapreduce.ImportJobBase: Transferred 0 bytes in 62.4294 seconds (0 bytes/sec)
17/04/02 16:03:39 INFO mapreduce.ImportJobBase: Retrieved 1100646 records.

real    1m13.700s
user    0m20.700s
sys     0m2.720s
```

2. *APP*


Logged in as: dr.who
Application application_1491122314944_0019
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Tools
Kill Application
Application Overview
User: root
Name: measures.jar
Application Type: MAPREDUCE
Application Tags:
Application Priority: 0 (Higher Integer value indicates higher priority)
YarnApplicationState: FINISHED
Queue: default
FinalStatus Reported by AM: SUCCEEDED
Started: Sun Apr 02 16:02:47 +0000 2017
Elapsed: 49sec
Tracking URL: History
Log Aggregation Status SUCCEEDED
Diagnostics:
Unmanaged Application: false
Application Node Label expression: <Not set>
AM container Node Label expression: <DEFAULT_PARTITION>
Application Metrics
Total Resource Preempted: <memory:0, vCores:0>
Total Number of Non-AM Containers Preempted: 0
Total Number of AM Containers Preempted: 0
Resource Preempted from Current Attempt: <memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt: 0
Aggregate Resource Allocation: 51592 MB-seconds, 203 vcore-seconds
Show 20 entries
Search:
Attempt ID
Started
Node
Logs
Blacklisted Nodes
appattempt_1491122314944_0019_000001
Sun Apr 2 17:02:47 +0100 2017
http://sandbox.hortonworks.com:8042
Logs
N/A
Showing 1 to 1 of 1 entries
First Previous 1 Next Last

3. *JOB* e *AVG MAP*



Logged in as: dr.who

MapReduce Job job_1491122314944_0019

- Application
- Job
  - Overview
  - Counters
  - Configuration
  - Map tasks
  - Reduce tasks
- Tools

Job Overview

Job Name:	measures.jar
User Name:	root
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sun Apr 02 16:02:47 UTC 2017
Started:	Sun Apr 02 16:02:55 UTC 2017
Finished:	Sun Apr 02 16:03:36 UTC 2017
Elapsed:	41sec
Diagnostics:	
Average Map Time	35sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sun Apr 02 16:02:50 UTC 2017	sandbox.hortonworks.com:8042	logs

Task Type	Total	Complete
Map	4	4
Reduce	0	0

Attempt Type	Failed	Killed	Successful
Maps	0	0	4
Reduces	0	0	0

4. *TT ALL MAP* e *CPU TIME*

hadoop

Logged in as: dr.who

Counters for job_1491122314944_0019

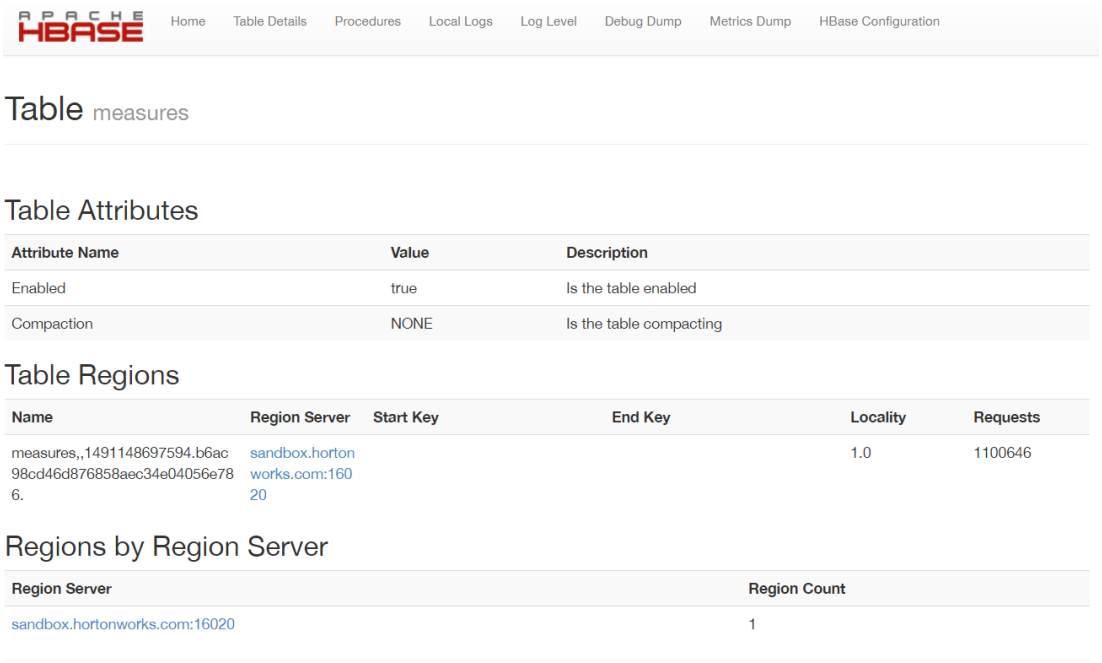
- Application
- Job
  - Overview
  - Counters
  - Configuration
  - Map tasks
  - Reduce tasks
- Tools

Counter Group	Counters			
File System Counters	Name	Map	Reduce	Total
	FILE: Number of bytes read	0	0	0
	FILE: Number of bytes written	786,688	0	786,688
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	497	0	497
	HDFS: Number of bytes written	0	0	0
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	4	0	4
	HDFS: Number of write operations	0	0	0
Job Counters	Name	Map	Reduce	Total
	Launched map tasks	0	0	4
	Other local map tasks	0	0	4
	Total megabyte-milliseconds taken by all map tasks	0	0	35,694,250
	Total time spent by all map tasks (ms)	0	0	142,777
	Total time spent by all maps in occupied slots (ms)	0	0	142,777
	Total vcore-milliseconds taken by all map tasks	0	0	142,777
Map-Reduce Framework	Name	Map	Reduce	Total
	CPU time spent (ms)	76,380	0	76,380
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	33,678	0	33,678
	Input split bytes	497	0	497
	Map input records	1,100,646	0	1,100,646
	Map output records	1,100,646	0	1,100,646
	Merged Map outputs	0	0	0
	Physical memory (bytes) snapshot	827,887,616	0	827,887,616
	Spilled Records	0	0	0
	Total committed heap usage (bytes)	290,455,552	0	290,455,552
	Virtual memory (bytes) snapshot	7,843,360,768	0	7,843,360,768
File Input Format Counters	Name	Map	Reduce	Total
	Bytes Read	0	0	0
File Output Format Counters	Name	Map	Reduce	Total
	Bytes Written	0	0	0

Tabela detalhada dos ensaios de otimizações das tabelas *HBase:*

TESTE	ENSAIO	TABELA	SHELL				MAPREDUCE			COUNTERS		MAPS (#)	
			SHELL MR	REAL	USER	SYS	APP	JOB	AVG MAP	TT ALL MAPS	CPU TIME		Job
1	1	Measures	62,43	73,70	20,70	2,72	49,00	41,00	35,00	142,78	76,38	4	19
		Sensors	40,14	51,22	19,45	3,32	29,00	14,00	8,00	32,11	11,45	4	20
	2	Measures	66,26	76,23	21,30	3,33	54,00	45,00	40,00	163,04	79,96	4	23
		Sensors	34,32	52,23	21,05	3,86	24,00	15,00	9,00	38,00	11,87	4	24
	3	Measures	58,82	67,34	21,47	3,26	51,00	43,00	37,00	149,43	75,63	4	25
		Sensors	30,63	42,15	23,75	4,61	24,00	16,00	10,00	41,31	13,30	4	26
	4	Measures	62,83	71,68	22,17	3,42	52,00	45,00	39,00	157,82	84,57	4	27
		Sensors	31,28	41,22	21,04	3,75	21,00	13,00	8,00	33,21	12,42	4	28
	5	Measures	58,81	67,21	20,32	2,88	46,00	39,00	34,00	137,28	75,46	4	29
		Sensors	36,27	46,02	19,37	3,72	28,00	20,00	14,00	57,11	16,36	4	30
2	1	Measures	60,37	70,68	21,41	3,12	50,00	43,00	37,00	150,38	78,37	4	31
		Sensors	40,90	65,52	20,45	3,60	31,00	15,00	9,00	36,84	12,44	4	32
	2	Measures	57,31	65,59	19,92	2,88	48,00	42,00	36,00	146,74	75,49	4	33
		Sensors	30,86	41,46	21,26	3,83	22,00	13,00	8,00	32,19	12,96	4	34
	3	Measures	59,22	67,77	18,95	3,06	48,00	42,00	35,00	140,43	77,19	4	35
		Sensors	29,87	40,46	19,89	4,13	23,00	15,00	9,00	39,37	10,92	4	36
	4	Measures	56,40	65,08	21,72	3,00	45,00	37,00	32,00	131,16	71,46	4	37
		Sensors	27,88	37,74	19,89	3,72	20,00	13,00	8,00	32,20	11,59	4	38
	5	Measures	64,64	74,22	20,64	4,05	54,00	46,00	42,00	168,26	85,37	4	39
		Sensors	33,63	86,56	18,35	3,23	30,00	15,00	9,00	37,41	11,28	4	40
3	1	Measures	62,11	72,11	21,35	2,65	50,00	43,00	37,00	149,74	77,62	4	41
		Sensors	42,45	52,35	19,76	3,62	31,00	16,00	9,00	39,64	13,16	4	42
	2	Measures	57,67	67,39	21,54	3,03	48,00	42,00	36,00	146,26	79,02	4	43
		Sensors	53,05	73,30	21,11	4,22	42,00	15,00	9,00	39,05	11,58	4	44
	3	Measures	58,61	67,17	21,62	2,89	46,00	39,00	35,00	140,58	75,55	4	45
		Sensors	29,16	40,72	20,55	3,35	21,00	14,00	8,00	32,35	12,49	4	46
	4	Measures	58,40	67,64	22,25	3,55	47,00	40,00	35,00	141,48	72,05	4	48
		Sensors	30,54	41,10	21,25	3,43	21,00	14,00	8,00	34,35	11,08	4	49
	5	Measures	60,72	70,14	25,48	2,97	51,00	43,00	38,00	153,72	83,21	4	50
		Sensors	39,79	54,26	21,38	4,22	29,00	13,00	8,00	32,07	11,54	4	51
4	1	Measures	74,48	84,01	21,36	3,72	66,00	57,00	27,00	164,91	68,69	6	54
		Sensors	35,55	46,00	23,34	3,23	25,00	14,00	9,00	36,55	12,43	4	55
	2	Measures	70,91	80,28	22,11	3,43	58,00	51,00	20,00	124,04	71,89	6	56
		Sensors	29,65	40,35	24,87	3,58	21,00	14,00	8,00	33,44	11,35	4	57
	3	Measures	69,97	82,09	25,23	2,91	59,00	51,00	20,00	123,88	68,26	6	58
		Sensors	33,21	42,02	21,74	3,34	23,00	15,00	10,00	40,46	12,29	4	59
	4	Measures	68,64	76,96	20,86	2,69	59,00	53,00	18,00	113,30	70,26	6	62
		Sensors	29,81	41,48	23,00	2,77	22,00	14,00	9,00	36,74	11,60	4	63
	5	Measures	72,37	81,64	22,33	2,78	62,00	54,00	22,00	136,69	75,27	6	64
		Sensors	33,96	42,63	20,14	2,94	24,00	16,00	10,00	41,02	11,76	4	65
5	1	Measures	71,33	79,63	20,87	2,53	62,00	55,00	23,00	140,67	74,02	6	66
		Sensors	30,46	39,05	20,85	2,74	19,00	12,00	8,00	32,28	9,84	4	67
	2	Measures	68,31	76,58	20,61	2,43	59,00	52,00	22,00	136,53	68,37	6	68
		Sensors	30,71	39,58	21,32	2,61	21,00	14,00	10,00	41,42	13,16	4	69
	3	Measures	66,82	74,36	20,36	2,59	58,00	51,00	20,00	123,61	69,34	6	70
		Sensors	28,41	36,88	20,52	2,68	20,00	14,00	8,00	35,34	12,54	4	71
	4	Measures	66,01	74,03	20,10	2,56	57,00	50,00	20,00	120,69	68,21	6	72
		Sensors	31,41	41,20	22,93	2,48	20,00	13,00	9,00	37,41	12,41	4	73
	5	Measures	71,61	81,24	23,28	2,56	61,00	54,00	23,00	142,13	74,48	6	74
		Sensors	30,07	39,39	22,59	2,54	21,00	14,00	8,00	35,27	11,73	4	75
6	1	Measures	78,48	87,94	23,83	2,90	68,00	60,00	25,00	155,58	78,37	6	76
		Sensors	33,15	42,57	23,08	2,61	24,00	16,00	10,00	41,60	12,99	6	77
	2	Measures	75,32	85,70	24,17	2,47	64,00	56,00	23,00	139,59	74,51	6	78
		Sensors	37,20	46,82	23,42	2,75	26,00	17,00	12,00	51,52	13,98	6	79
	3	Measures	71,83	80,78	21,39	2,92	63,00	54,00	22,00	135,59	73,50	6	80
		Sensors	29,82	38,30	21,08	3,03	22,00	14,00	9,00	37,93	13,06	6	81
	4	Measures	79,56	89,16	24,72	3,35	70,00	62,00	26,00	159,78	82,47	6	82
		Sensors	34,28	46,50	25,07	2,96	25,00	17,00	11,00	45,93	14,88	6	83
	5	Measures	73,63	82,50	20,89	2,64	64,00	57,00	24,00	144,23	76,68	6	84
		Sensors	31,00	40,14	20,07	2,22	22,00	13,00	8,00	32,83	11,71	6	85
7	1	Measures	64,31	82,67	20,12	2,44	56,00	50,00	21,00	126,08	69,80	6	89
		Sensors	31,36	40,34	20,62	2,52	19,00	13,00	7,00	30,07	10,70	6	90
	2	Measures	64,43	72,88	21,03	2,75	57,00	50,00	20,00	121,54	68,02	6	91
		Sensors	27,92	36,18	20,68	2,55	21,00	13,00	8,00	34,08	12,12	6	92
	3	Measures	65,66	74,09	21,19	2,86	58,00	52,00	20,00	123,58	69,06	6	93
		Sensors	28,38	36,87	19,92	2,80	20,00	14,00	8,00	34,25	12,15	6	94
	4	Measures	63,32	72,28	21,81	2,51	55,00	49,00	19,00	114,71	68,01	6	95
		Sensors	27,43	35,60	20,33	2,53	20,00	13,00	8,00	32,25	11,02	6	96
	5	Measures	65,60	73,85	20,27	2,61	57,00	50,00	20,00	120,58	69,01	6	97
		Sensors	28,62	36,77	20,34	2,33	20,00	13,00	7,00	31,13	11,28	6	98
8	1	Measures	48,74	56,31	20,89	2,53	40,00	33,00	31,00	31,03	35,00	6	99
		Sensors	22,85	31,06	21,31	2,56	14,00	7,00	4,00	4,67	3,94	6	100
	2	Measures	44,74	53,61	20,80	2,32	37,00	31,00	28,00	28,57	36,06	6	101
		Sensors	22,53	30,60	20,56	2,58	14,00	7,00	4,00	4,62	3,48	6	102
	3	Measures	46,72	54,92	20,81	2,62	39,00	33,00	30,00	30,57	36,21	6	103
		Sensors	22,73	31,85	20,93	2,40	13,00	6,00	4,00	4,07	3,18	6	104
	4	Measures	46,82	54,87	20,73	2,40	38,00	31,00	28,00	28,54	33,99	6	105
		Sensors	22,94	30,93	20,05	2,65	13,00	6,00	4,00	4,23	3,36	6	106
	5	Measures	44,73	53,56	21,32	2,21	36,00	30,00	27,00	27,93	34,36	6	107
		Sensors	21,10	30,59	23,11	2,45	14,00	7,00	4,00	4,77	4,05	6	108

Distribuição das regiões no **teste 1** (todas para a mesma região causando *Hotspotting*):



Distribuição das regiões no **teste 2** (todas para a mesma região causando *Hotspotting,* embora existam 10 regiões):

Table Regions

Name	Region Server	Start Key	End Key	Locality	Requests
measures,,1491155287536.ae71bfd55ccc5e5517ea693f3c59c206.	sandbox.hortonworks.com:16020		19999999	1.0	1100646
measures,19999999,1491155287536.016c8b17e5bcd76916502539e8b2b2cb.	sandbox.hortonworks.com:16020	19999999	33333332	0.0	0
measures,33333332,1491155287536.68e762fa5f02fb6b0ea56e8bdbf28ed1.	sandbox.hortonworks.com:16020	33333332	4ccccccb	0.0	0
measures,4ccccccb,1491155287536.69f071be52906c7fd198ac62b5956b0e.	sandbox.hortonworks.com:16020	4ccccccb	66666664	0.0	0
measures,66666664,1491155287536.9b8fb81c5cb4da5c5be3dc2ba543b70f.	sandbox.hortonworks.com:16020	66666664	7fffffd	0.0	0
measures,7fffffd,1491155287536.5c6dde5322d671795bb27b7965d8d0c5e.	sandbox.hortonworks.com:16020	7fffffd	99999996	0.0	0
measures,99999996,1491155287536.2bca87fd88bb79c405e37d77b80371a4.	sandbox.hortonworks.com:16020	99999996	b333332f	0.0	0
measures,b333332f,1491155287536.160591e8952ee26ea3834e2ffed9e8ff.	sandbox.hortonworks.com:16020	b333332f	cccccc8	0.0	0
measures,cccccc8,1491155287536.348b9e68ce5753790f21bd41b6fce0f5.	sandbox.hortonworks.com:16020	cccccc8	e6666661	0.0	0
measures,e6666661,1491155287536.6a0456e5988ddf75b6032f1be903fff0.	sandbox.hortonworks.com:16020	e6666661		0.0	0

Regions by Region Server

Region Server	Region Count
sandbox.hortonworks.com:16020	10

Distribuição das regiões no **teste 3** (todas para a mesma região causando *Hotspotting,* embora existam 10 regiões):

Table Regions

Name	Region Server	Start Key	End Key	Locality	Requests
measures,,1491159101584.4ddbe29109455d9379a556064926ebf1.	sandbox.hortonworks.com:16020		\x19\x99\x99\x99\x99\x99\x99\x99	0.0	0
measures,\x19\x99\x99\x99\x99\x99\x99\x99,1491159101584.210f2a194a3e64bb8a04bd3de4b0a970.	sandbox.hortonworks.com:16020	\x19\x99\x99\x99\x99\x99\x99\x99	33333332	1.0	1100646
measures,33333332,1491159101584.e529d16ff41513522219c049dd7ba650.	sandbox.hortonworks.com:16020	33333332	L\xCC\xCC\xCC\xCC\xCC\xCC\xCB	0.0	0
measures,L\xCC\xCC\xCC\xCC\xCC\xCC\xCB,1491159101584.95d35be98b4821f4f15df58571782a04.	sandbox.hortonworks.com:16020	L\xCC\xCC\xCC\xCC\xCC\xCC\xCB	ffffffd	0.0	0
measures,ffffffd,1491159101584.81d3ca448152c498fd3baa852f496545.	sandbox.hortonworks.com:16020	ffffffd	\x7F\xFF\xFF\xFF\xFF\xFF\xFD	0.0	0
measures,\x7F\xFF\xFF\xFF\xFF\xFF\xFD,1491159101584.6a4c5cf8a328c9318867b2c6129a1d06.	sandbox.hortonworks.com:16020	\x7F\xFF\xFF\xFF\xFF\xFF\xFD	\x99\x99\x99\x99\x99\x99\x96	0.0	0
measures,\x99\x99\x99\x99\x99\x99\x99\x96,1491159101584.e029ac9e31f2bbe549fff6ee245a22f3.	sandbox.hortonworks.com:16020	\x99\x99\x99\x99\x99\x99\x96	\xB3333333/	0.0	0
measures,\xB3333333/,1491159101584.d0e1a93abca5224c124f25eedd42effb.	sandbox.hortonworks.com:16020	\xB3333333/	\xCC\xCC\xCC\xCC\xCC\xCC\xC8	0.0	0
measures,\xCC\xCC\xCC\xCC\xCC\xCC\xC8,1491159101584.62025d260e4cdfd2dacee4113dc6faca.	sandbox.hortonworks.com:16020	\xCC\xCC\xCC\xCC\xCC\xCC\xC8	\xE6ffffffa	0.0	0
measures,\xE6ffffffa,1491159101584.6c4f4e411ba68334bb3f115708b11da0.	sandbox.hortonworks.com:16020	\xE6ffffffa		0.0	0

Regions by Region Server

Region Server	Region Count
sandbox.hortonworks.com:16020	10

Distribuição das regiões no **teste 4** (todas para a mesma região causando *Hotspotting*):

Table measures

Table Attributes

Attribute Name	Value	Description
Enabled	true	Is the table enabled
Compaction	NONE	Is the table compacting

Table Regions

Name	Region Server	Start Key	End Key	Locality	Requests
measures,,1491169113712.c9d0e93fcd2a2468e7149a06a2c75e37.	sandbox.hortonworks.com:16020			1.0	1514092

Regions by Region Server

Region Server	Region Count
sandbox.hortonworks.com:16020	1

Distribuição das regiões no **teste 5** (distribuído entre 7 regiões com 3 delas sem utilização):

Table Regions

Name	Region Server	Start Key	End Key	Locality	Requests
measures,,1491172488217.0592b36c9e6d2ab1c1b559914d7b8cf0.	sandbox.hortonworks.com:16020		19999999	0.0	214702
measures,19999999,1491172488217.e453c804df36d707ed5b71f37301e5be.	sandbox.hortonworks.com:16020	19999999	33333332	0.0	224476
measures,33333332,1491172488217.72920ddf3eb399872e8dc24355206624.	sandbox.hortonworks.com:16020	33333332	4ccccccb	0.0	249592
measures,4ccccccb,1491172488217.bfe58b16fa814d0a2817fea2e1305107.	sandbox.hortonworks.com:16020	4ccccccb	66666664	0.0	163641
measures,66666664,1491172488217.f8be8303cc68cdc6a17ffd937cb1d843.	sandbox.hortonworks.com:16020	66666664	7fffffd	0.0	111652
measures,7fffffd,1491172488217.a7677bfae1d42db7319bd7f8dc386064.	sandbox.hortonworks.com:16020	7fffffd	99999996	0.0	107685
measures,99999996,1491172488217.a8c5c6d5876dfc73ab61fc79af5b66d0.	sandbox.hortonworks.com:16020	99999996	b333332f	0.0	442264
measures,b333332f,1491172488217.ffd2300cba8678f95b181f7754348f48.	sandbox.hortonworks.com:16020	b333332f	ccccccc8	0.0	0
measures,ccccccc8,1491172488217.ade1072ef6b4ea23e56bb84483640c99.	sandbox.hortonworks.com:16020	ccccccc8	e6666661	0.0	0
measures,e6666661,1491172488217.a7a63fef630ef49617ed12083b3c6811.	sandbox.hortonworks.com:16020	e6666661		0.0	0

Regions by Region Server

Region Server	Region Count
sandbox.hortonworks.com:16020	10

Distribuição das regiões no **teste 6** (distribuído entre 2 regiões com as restantes sem utilização):

Table Regions

Name	Region Server	Start Key	End Key	Locality	Requests
measures,,1491175915541.288ccb4192c4f36194069be1cf648f59.	sandbox.hortonworks.com:16020		\x19\x99\x99\x99\x99\x99\x99\x99	0.0	0
measures,\x19\x99\x99\x99\x99\x99\x99\x99,1491175915541.a159f76fefd679e5f48a7a80c379377e.	sandbox.hortonworks.com:16020	\x19\x99\x99\x99\x99\x99\x99\x99	33333332	0.0	439178
measures,33333332,1491175915541.1c67b77c73a580d32fe1f6c8aef2d843.	sandbox.hortonworks.com:16020	33333332	L\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCB	1.0	1074834
measures,L\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCB,1491175915541.9e1ad049ac6d86e343bc6b09810e3f65.	sandbox.hortonworks.com:16020	L\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xCB	fffffffd	0.0	0
measures,fffffffd,1491175915541.65bd0d071177956bfd7f4404443d056c.	sandbox.hortonworks.com:16020	fffffffd	\x7F\xFF\xFF\xFF\xFF\xFF\xFF\xFD	0.0	0
measures,\x7F\xFF\xFF\xFF\xFF\xFF\xFF\xFD,1491175915541.a98c74c908949433f3a3af2affefa88d.	sandbox.hortonworks.com:16020	\x7F\xFF\xFF\xFF\xFF\xFF\xFF\xFD	\x99\x99\x99\x99\x99\x99\x99\x96	0.0	0
measures,\x99\x99\x99\x99\x99\x99\x99\x96,1491175915541.953600bb60a49392c22c527ef13d944f.	sandbox.hortonworks.com:16020	\x99\x99\x99\x99\x99\x99\x99\x96	\xB3333333/	0.0	0
measures,\xB3333333/,1491175915541.e2191aa1d7784d94161593f3c8f1a3b8.	sandbox.hortonworks.com:16020	\xB3333333/	\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xC8	0.0	0
measures,\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xC8,1491175915541.8e9d41f1db49ad386917b6c06e0babd7.	sandbox.hortonworks.com:16020	\xCC\xCC\xCC\xCC\xCC\xCC\xCC\xC8	\xE6ffffffa	0.0	0
measures,\xE6ffffffa,1491175915541.035208420ac513fbf824ffd43d963dd8.	sandbox.hortonworks.com:16020	\xE6ffffffa		0.0	0

Regions by Region Server

Region Server	Region Count
sandbox.hortonworks.com:16020	10

Distribuição das regiões nos **teste 7** e **8** (distribuição uniforme entre as 10 regiões):

Table Regions

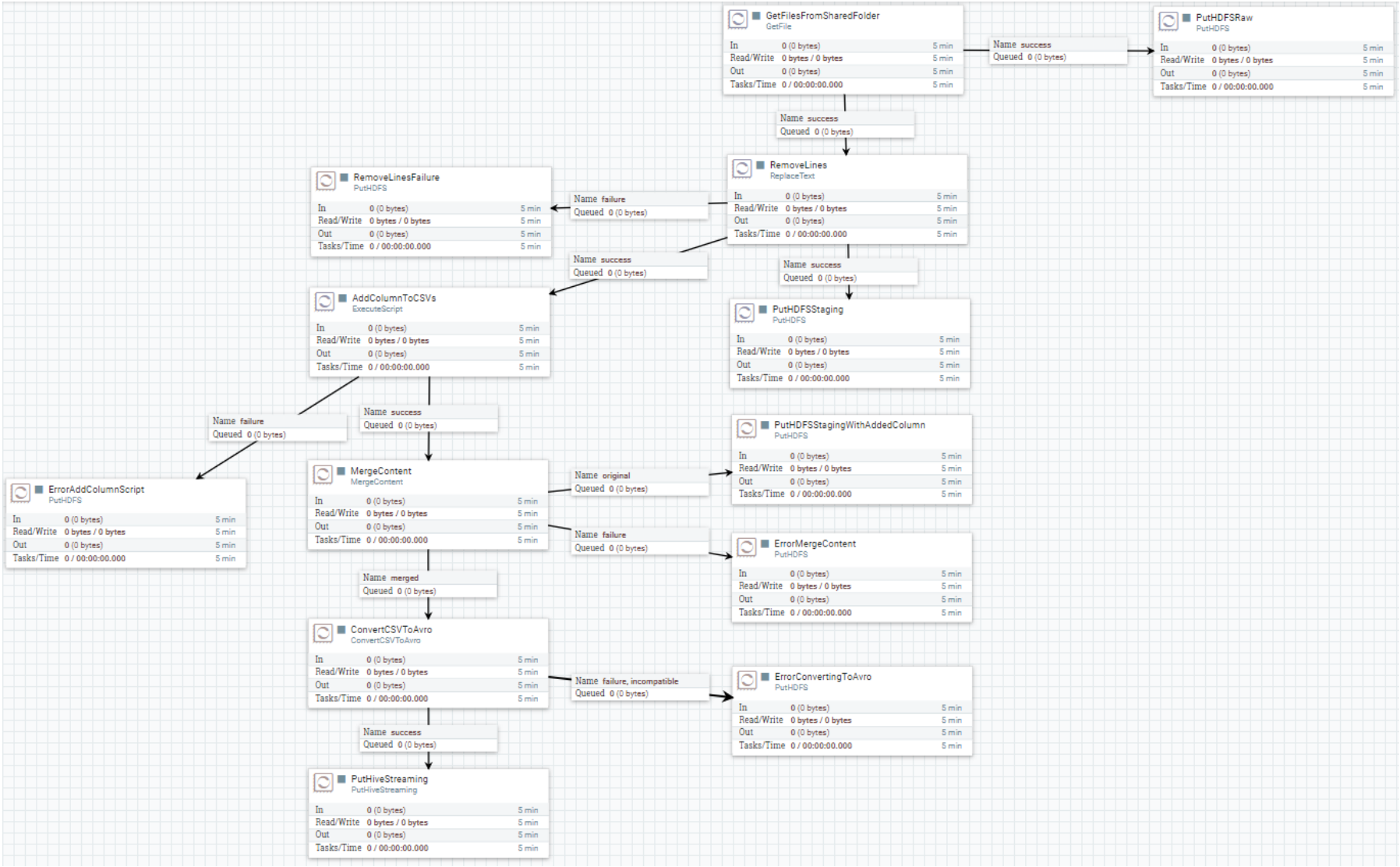
Name	Region Server	Start Key	End Key	Locality	Requests
measures3,,1503134212949.86bf113918f06992791af4aaf101ef48.	sandbox.hortonworks.com:16020		19	0.0	107351
measures3,19,1503134212949.c6485c983a0364af6985f62ede4024e0.	sandbox.hortonworks.com:16020	19	33	0.0	112238
measures3,33,1503134212949.eed54da6f55823f0b2d8fd848517fcf5.	sandbox.hortonworks.com:16020	33	4C	0.0	107555
measures3,4C,1503134212949.411e28c7b6455eb405355d791d8aef37.	sandbox.hortonworks.com:16020	4C	66	0.0	111901
measures3,66,1503134212949.e32b0c144176d363e2ed959e6d61e2f7.	sandbox.hortonworks.com:16020	66	7F	0.0	107414
measures3,7F,1503134212949.31b439ee1b0d50a9ec1adbbc2d0de509.	sandbox.hortonworks.com:16020	7F	99	0.0	111923
measures3,99,1503134212949.3279587dc293b61185bba658e83b58e3.	sandbox.hortonworks.com:16020	99	B3	0.0	111392
measures3,B3,1503134212949.63f5ba32fa178f416c5c508efe133e5c.	sandbox.hortonworks.com:16020	B3	CC	0.0	107455
measures3,CC,1503134212949.f0381344481a4b3db20b8e3d9c9b6834.	sandbox.hortonworks.com:16020	CC	E6	0.0	111814
measures3,E6,1503134212949.106dea73641a2f799278b7d99ee32e01.	sandbox.hortonworks.com:16020	E6		0.0	111603

Regions by Region Server

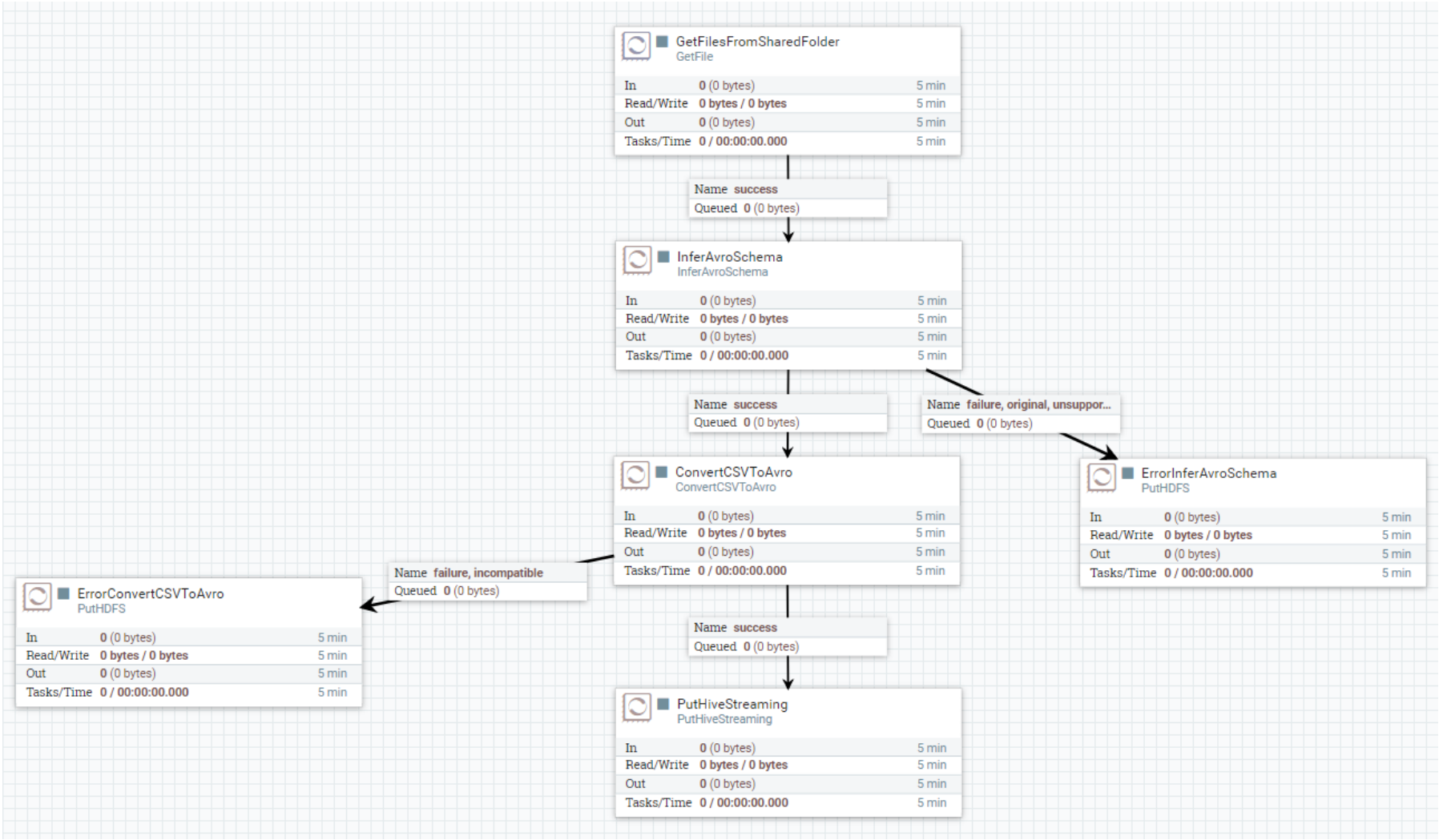
Region Server	Region Count
sandbox.hortonworks.com:16020	10

6.3. *NiFi*

***Template* SNIRH:**



Template *Satellite Images*:



***Script* utilizado na componente *AddColumnToCSVs* do *template* SNIRH:**

```
import json
import java.io
from org.apache.commons.io import IOUtils
from java.nio.charset import StandardCharsets
from org.apache.nifi.processor.io import StreamCallback

class PyStreamCallback(StreamCallback):
  def __init__(self):
      pass
  def process(self, inputStream, outputStream):
    text = IOUtils.readLines(inputStream, StandardCharsets.UTF_8)
    for line in text:
      outputStream.write(line + flowFile.getAttribute('filename').split('.')[0]+ "\n")

flowFile = session.get()
if (flowFile != None):
  flowFile = session.write(flowFile,PyStreamCallback())
  flowFile = session.putAttribute(flowFile, "filename",
flowFile.getAttribute('filename').split('.')[0])
  session.transfer(flowFile, REL_SUCCESS)
```

***Script* utilizado na componente *ConvertCSVToAvro* do *template* SNIRH:**

```
{ "type" : "record", "name" : "WeatherStation", "fields" :
 [ { "name" : "Data", "type" : [ "null", "string" ], "default" : "1900-01-01" },
   { "name" : "Albufeira_do_alqueva", "type" : [ "null", "double" ], "default" : -1.0 },
   { "name" : "Flag1", "type" : [ "null", "string" ], "default" : "N/A" },
   { "name" : "Albufeira_do_alqueva_mourao", "type" : [ "null", "double" ], "default" : -1.0 },
   { "name" : "Flag2", "type" : [ "null", "string" ], "default" : "N/A" },
   { "name" : "Montoito", "type" :  [ "null", "double" ], "default" : -1.0 },
   { "name" : "Flag3", "type" : [ "null", "string" ], "default" : "N/A" },
   { "name" : "Portel", "type" :  [ "null", "double" ], "default" : -1.0 },
```

{ "name" : "Flag4", "type" : ["null", "string"], "default" : "N/A" },

{ "name" : "Reguengos", "type" : ["null", "double"], "default" : -1.0 },

{ "name" : "Flag5", "type" : ["null", "string"], "default" : "N/A" },

{ "name" : "Santiago_maior", "type" : ["null", "double"], "default" : -1.0 },

{ "name" : "Flag6", "type" : ["null", "string"], "default" : "N/A" },

{ "name" : "Sao_mancos", "type" : ["null", "double"], "default" : -1.0 },

{ "name" : "Flag7", "type" : ["null", "string"], "default" : "N/A" },

{ "name" : "FileName", "type" : ["null", "string"], "default" : "N/A" }] }

Script* utilizado na criação das tabelas *Hive:

```
CREATE DATABASE Innovine
LOCATION  '/user/hive/Innovine';

CREATE TABLE IF NOT EXISTS Innovine.weathersations(
                    Data                                STRING,
                    Albufeira_do_alqueva                DOUBLE,
                    FLAG1                               STRING,
                    Albufeira_do_alqueva_mourao         DOUBLE,
                    FLAG2                               STRING,
                    Montoito                            DOUBLE,
                    FLAG3                               STRING,
                    Portel                              DOUBLE,
                    FLAG4                               STRING,
                    Reguengos                           DOUBLE,
                    FLAG5                               STRING,
                    Santiago_o_maior                    DOUBLE,
                    FLAG6                               STRING,
                    Sao_mancos                          DOUBLE,
                    FLAG7                               STRING,
                    FileName                            STRING)
CLUSTERED BY (Data) INTO 3 BUCKETS
ROW FORMAT DELIMITED
STORED AS ORC
LOCATION  '/user/hive/Innovine/weatherstations'
TBLPROPERTIES('transactional'='true');

CREATE TABLE IF NOT EXISTS Innovine.satelliteimages(
                    Id                                  INT,
                    Data                                STRING,
                    Renderingtype                       STRING,
                    Url                                 STRING)
CLUSTERED BY (Data) INTO 3 BUCKETS
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\u003B'
STORED AS ORC
```

```
LOCATION  '/user/hive/Innovine/satelliteimages'
TBLPROPERTIES('transactional'='true');
```

6.4. *HIVE*

Script* de criação e carregamento de tabelas *Hive:

```
CREATE EXTERNAL TABLE IF NOT EXISTS Innovine.dimDate(
        FullDate                STRING,
        DayOfWeek               TINYINT,
        DayNameOfWeek           STRING,
        DayOfMonth              TINYINT,
        DayOfYear               SMALLINT,
        WeekdayWeekend          STRING,
        WeekOfYear              TINYINT,
        MonthName               STRING,
        MonthOfYear             TINYINT,
        IsLastDayOfMonth        STRING,
        CalendarQuarter         TINYINT,
        CalendarYear            SMALLINT,
        CalendarYearMonth       STRING,
        CalendarYearQtr         STRING,
        FiscalMonthOfYear       TINYINT,
        FiscalQuarter           TINYINT,
        FiscalYear              INT,
        FiscalYearMonth         STRING,
        FiscalYearQtr           STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\u003B'
LOCATION  '/user/hive/Innovine/dimDate';

LOAD DATA INPATH '/user/hive/Innovine/dimDate/Dim_Date.csv' INTO TABLE
Innovine.dimDate;

ALTER TABLE Innovine.dimDate SET SERDEPROPERTIES ('serialization.encoding'='UTF-8');

CREATE EXTERNAL TABLE Innovine.measures(id_measure STRING, measure DOUBLE)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,measure:measure')
TBLPROPERTIES ('hbase.table.name' = 'measures');

CREATE EXTERNAL TABLE Innovine.sensors(id_sensor int, id_station int, sensorName
STRING)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' =
':key,sensor:id_station,sensor:sensorName')
TBLPROPERTIES ('hbase.table.name' = 'sensors');

CREATE EXTERNAL TABLE Innovine.weathermeasures(id_weatherMeasure STRING,
measure DOUBLE)
```

```
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,weatherMeasure:measure')
TBLPROPERTIES ('hbase.table.name' = 'weatherMeasures');

CREATE EXTERNAL TABLE Innovine.stations(id_station int, value int)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,station:id_station')
TBLPROPERTIES ('hbase.table.name' = 'stations');

CREATE TABLE IF NOT EXISTS Innovine.dimParameter(
        Parameter_ID        INT,
        Parameter           STRING
 )
CLUSTERED BY (Parameter_ID) INTO 3 BUCKETS
ROW FORMAT DELIMITED
STORED AS ORC
LOCATION  '/user/hive/Innovine/dimParameter'
TBLPROPERTIES('transactional'='true');

CREATE FUNCTION Sequence AS 'org.apache.hadoop.hive.contrib.udf.UDFHiveIdentity'
USING JAR 'hdfs:///tmp/UDFHiveIdentity.jar';

INSERT INTO Innovine.dimParameter
select Sequence() AS Parameter_ID, filename as Parameter from (
select distinct filename from Innovine.Weatherstations) a;

CREATE EXTERNAL TABLE IF NOT EXISTS Innovine.dimWeatherStation(WeatherStation_ID
INT,WeatherStation STRING,WeatherStation_Code STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES("serialization.encoding"='UTF-8')
LOCATION  '/user/hive/Innovine/dimWeatherStation';

insert into Innovine.dimWeatherStation
VALUES(1,'Albufeira do Alqueva','24L/02F'),
(2,'Albufeira do Alqueva (Mourão)','22M/05F'),
(3,'Montoito','22L/03UG'),
(4,'Portel','24K/01UG'),
(5,'Reguengos','23L/01G'),
(6,'Santiago Maior','22M/01UG'),
(7,'São Manços','23K/01UG')

ALTER TABLE Innovine. dimWeatherStation SET SERDEPROPERTIES
('serialization.encoding'='ISO-8859-1');

CREATE EXTERNAL TABLE IF NOT EXISTS Innovine.fact_weatherstations(
        PK_Weatherstation       INT,
        Data                    STRING,
        Weatherstation_id       INT,
        Parameter_id            INT,
        Value                   DOUBLE
```

```
 )
--CLUSTERED BY (Data) INTO 3 BUCKETS
ROW FORMAT DELIMITED
LOCATION  '/user/hive/Innovine/fact_weatherstations';

insert into Innovine.fact_weatherstations
SELECT Sequence() as pk_weatherstation,* FROM (
select a.data,b.weatherstation_id,c.parameter_id,a.value from (
select data, weatherstation, filename, COALESCE(value,CAST(0 AS double)) as value from
   (select data, map("Albufeira do Alqueva", albufeira_do_alqueva, "Albufeira do Alqueva
(Mourão)", albufeira_do_alqueva_mourao, "Montoito", montoito, "Portel", portel,
"Reguengos", reguengos, "Santiago Maior", santiago_o_maior, "São Manços", sao_mancos)
as map1, filename
   from Innovine.weatherstations) as t1
lateral view explode(map1) exp as weatherstation, value) a
left join Innovine.dimweatherstation b
on a.weatherstation = b.weatherstation
left join Innovine.dimparameter c
on a.filename = c.parameter ) X;

CREATE EXTERNAL TABLE IF NOT EXISTS Innovine.dimTime(
        Id_Time             INT,
        Hour_NR             INT,
        Minute_NR           INT,
        Time                STRING
 )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\u003B'
LOCATION  '/user/hive/Innovine/dimTime';

LOAD DATA INPATH '/user/hive/Innovine/dimTime/Dim_Time.csv' INTO TABLE
Innovine.dimTime;
```

Script* pertencente à classe *UDFHiveIdentity.Java:

```
package org.apache.hadoop.hive.contrib.udf;

import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.hive.ql.udf.UDFType;
import org.apache.hadoop.io.LongWritable;

/**
 * Classe que produz o equivalente a uma coluna identidade em sql, que
 * permite gerar um numero sequencial, neste caso para ser usado como uma
 * Surrogate Key de uma dimensao.
 */
@Description(name = "row_identity", value = "_FUNC_() - retorna numero
incremental comecando no numero 1")
@UDFType(deterministic = false, stateful = true)
public final class UDFHiveIdentity extends UDF
{
```

```
      public static void main(String args[])
    {/*Empty Main - Do nothing*/}

            private LongWritable result = new LongWritable();

            public UDFHiveIdentity() {
                  result.set(0);
            }

            public LongWritable evaluate() {
                  result.set(result.get() + 1);
                  return result;
            }

}
```

6.5. *SCRAPY*

Classe *maiscarrinhoSpider.py*:

```
import scrapy
import json

class WineSpider(scrapy.Spider):
    name = "SpidyWine"

    url =
'https://maiscarrinho.com/api/search?q=vinho&pageNumber=%s&pageSize=
10'
    start_urls = [url % 1]
    i = 1
    def parse(self, response):
        data = json.loads(response.body.decode('utf-8'))
        for item in data.get('results', []):
            for product in item['ProductUpdates']:
                yield {
                    'Image': item.get('Image'),
                    'SalePrice': product.get('SalePrice'),
                    'Supermarket': product['Provider'].get('Name'),
                    'Supermarket_Id': product['Provider'].get('Id'),
                    'Name': item.get('Name'),
                    'Brand': item.get('Brand'),
                    'Brand_Id': item.get('Id'),
                    'PromotionValue': product.get('PromotionValue'),
                    }

        if data.get('results', []):
            WineSpider.i += 1
            yield scrapy.Request(self.url % WineSpider.i,
callback=self.parse)
```

Classe *items.py*:

```
# -*- coding: utf-8 -*-

import scrapy

class MaiscarrinhoItem(scrapy.Item):
     Image = scrapy.Field()
     Saleprice = scrapy.Field()
     Supermarket_Id = scrapy.Field()
     Supermarket = scrapy.Field()
     Name = scrapy.Field()
     Brand_Id = scrapy.Field
     Brand = scrapy.Field()
     PromotionValue = scrapy.Field()
```

Classe *pipelines.py*:

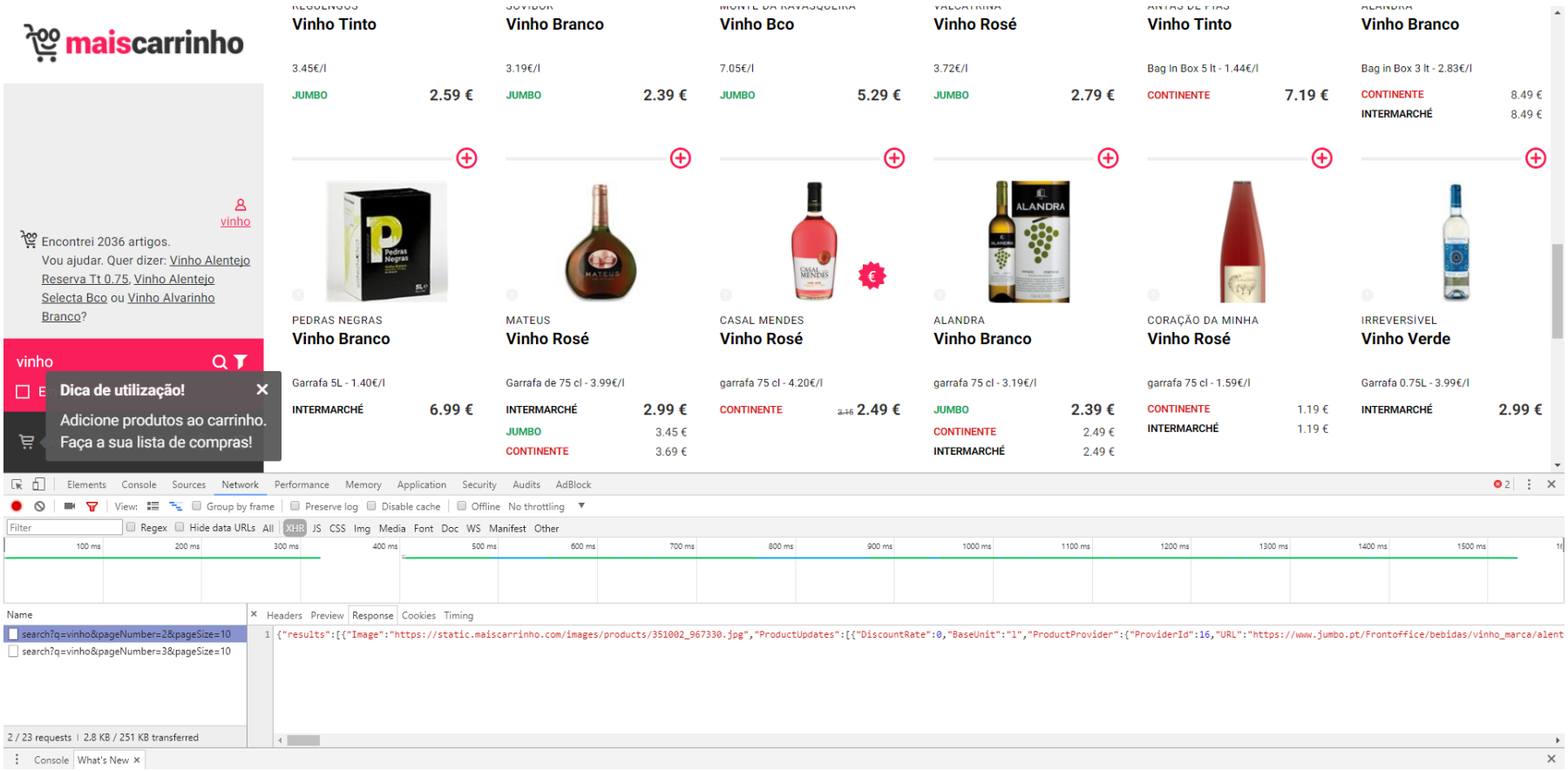
```
# -*- coding: utf-8 -*-
from os import getenv
import pymssql

#from slybot.item import create_item_version

class MaiscarrinhoPipeline(object):

    def __init__(self):
        self.conn = pymssql.connect(host='FRANCISCO\MSSQLSERVERERP',
user='FRANCISCO\Francisco Miguel', password='xxxxxxxx',
database='MaisCarrinho')
        self.cursor = self.conn.cursor()


    def process_item(self, item, spider):
            self.cursor.execute("""IF NOT EXISTS (SELECT * FROM
sys.objects WHERE Name = 'WinesNames' AND type = (N'U')) BEGIN
CREATE TABLE WinesNames (PK_ID UNIQUEIDENTIFIER, Name NVARCHAR(256),
Brand NVARCHAR(256), Brand_Id INT, Image NVARCHAR(256), Date_Created
DATETIME, Date_Modfied DATETIME, Active BIT) END""")
            self.cursor.execute("""IF NOT EXISTS (SELECT * FROM
sys.objects WHERE Name = 'SuperMarket' AND type = (N'U')) BEGIN
CREATE TABLE SuperMarket (PK_ID UNIQUEIDENTIFIER, Supermarket_Id
INT, Supermarket NVARCHAR(256), Date_Created DATETIME, Date_Modfied
DATETIME, Active BIT) END""")
            self.cursor.execute("""SELECT * FROM WinesNames WHERE
Name = %s and Brand = %s and Brand_Id = %d""",(item['Name'],
item['Brand'], item['Brand_Id']))
            result = self.cursor.fetchone()

            if result:
                print("Wine already exists!")
            else:
```

```
                try:
                    self.cursor.execute("""INSERT INTO
WinesNames(PK_ID, Name, Brand, Brand_Id, Image, Date_Created,
Date_Modfied, Active) VALUES (NEWID(),%s, %s, %d, %s,
GETDATE(),GETDATE(),1)""",(item['Name'], item['Brand'],
item['Brand_Id'], item['Image']))
                    self.cursor.execute("""DELETE FROM WinesNames
WHERE Name LIKE '%Vinagre%' OR Name LIKE '%Chou%' OR Name LIKE
'%Manga%' OR Name LIKE '%Guia%'""")
                    self.conn.commit()
                except pymssql.Error as e:
                    print ("Error when trying to insert values in
database.")

            self.cursor .execute("""SELECT * FROM SuperMarket WHERE
Supermarket_Id = %d and Supermarket = %s""",(item['Supermarket_Id'],
item['Supermarket']))
            result2 = self.cursor.fetchone()


            if result2:
                print("SuperMarket already exists!")
            else:

                try:
                    self.cursor.execute("""INSERT INTO
SuperMarket(PK_ID, Supermarket_Id, Supermarket, Date_Created,
Date_Modfied, Active) VALUES (NEWID(),%d, %s,
GETDATE(),GETDATE(),1)""",(item['Supermarket_Id'],
item['Supermarket']))
                    self.conn.commit()
                except pymssql.Error as e:
                    print ("Error when trying to insert values in
database.")

            try:
                self.cursor.execute("""IF NOT EXISTS (SELECT * FROM
sys.objects WHERE Name = 'Wines' AND type = (N'U')) BEGIN CREATE
TABLE Wines (PK_ID UNIQUEIDENTIFIER, SalePrice NVARCHAR(256),
Supermarket_Id INT, Brand_Id INT, PromotionValue NVARCHAR(256),
Date_Created DATETIME, Date_Modfied DATETIME, Active BIT) END""")
                self.cursor.execute("""INSERT INTO
Wines(PK_ID,SalePrice,Supermarket_Id,Brand_Id,PromotionValue,Date_Cr
eated,Date_Modfied,Active) VALUES
(NEWID(),%s,%d,%d,%s,GETDATE(),GETDATE(),1)""",(item['SalePrice'],
item['Supermarket_Id'], item['Brand_Id'], item['PromotionValue']))
                self.cursor.execute("""DELETE w FROM Wines w LEFT
JOIN WinesNames b ON w.Brand_id = b.Brand_Id WHERE b.Brand_Id IS
NULL""")
                self.conn.commit()
            except pymssql.Error as e:
                print ("Error when trying to insert values in
database.")

            return item

    def spider_closed(self, spider):
        self.conn.close()
```

Classe *settings.py*:

```
# -*- coding: utf-8 -*-
# Scrapy settings for Maiscarrinho project

BOT_NAME = 'Maiscarrinho'

SPIDER_MODULES = ['Maiscarrinho.spiders']
NEWSPIDER_MODULE = 'Maiscarrinho.spiders'

# Obey robots.txt rules
ROBOTSTXT_OBEY = True

ITEM_PIPELINES = {
    'Maiscarrinho.pipelines.MaiscarrinhoPipeline': 300,
}
```

Figura 25 completa:



Parte do *output* da consola após extração:

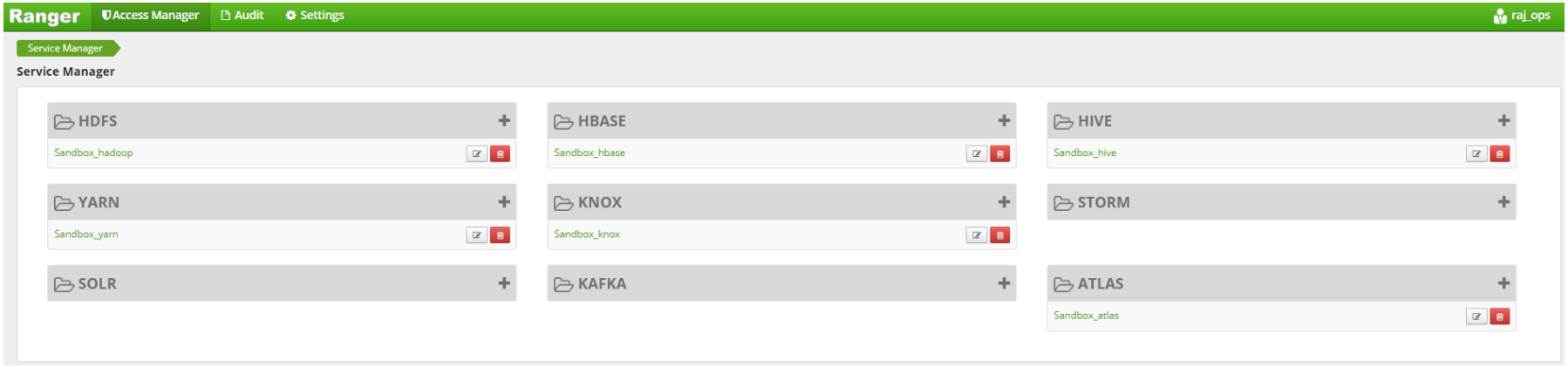
```
2017-08-21 12:54:13 [scrapy.core.scraper] DEBUG: Scraped from <200 https://maiscarrinho.com/api/search?q=vinho&pageNumber=40&pageSize=10>
{'Supermarket_Id': 1, 'SalePrice': 5.916601069947399, 'Brand_Id': 10353, 'Image': u'https://static.maiscarrinho.com/images/products/351001_2050412.jpg
', 'PromotionValue': 0.690546818833675, 'Brand': u'Mon\xe7\xe3o', 'Supermarket': u'Continente', 'Name': u'Vinho Verde Tinto'}
Wine already exists!
SuperMarket already exists!
2017-08-21 12:54:13 [scrapy.core.scraper] DEBUG: Scraped from <200 https://maiscarrinho.com/api/search?q=vinho&pageNumber=40&pageSize=10>
{'Supermarket_Id': 5, 'SalePrice': 4.809735732966988, 'Brand_Id': 10353, 'Image': u'https://static.maiscarrinho.com/images/products/351001_2050412.jpg
', 'PromotionValue': 0, 'Brand': u'Mon\xe7\xe3o', 'Supermarket': u'Intermarche', 'Name': u'Vinho Verde Tinto'}
Wine already exists!
SuperMarket already exists!
2017-08-21 12:54:13 [scrapy.core.scraper] DEBUG: Scraped from <200 https://maiscarrinho.com/api/search?q=vinho&pageNumber=40&pageSize=10>
{'Supermarket_Id': 1, 'SalePrice': 4.039625659097659, 'Brand_Id': 10372, 'Image': u'https://static.maiscarrinho.com/images/products/351001_2050491.jpg
', 'PromotionValue': 2.595986904776489, 'Brand': u'Meia Encosta', 'Supermarket': u'Continente', 'Name': u'Vinho Tinto D\xe3o'}
Wine already exists!
SuperMarket already exists!
2017-08-21 12:54:13 [scrapy.core.scraper] DEBUG: Scraped from <200 https://maiscarrinho.com/api/search?q=vinho&pageNumber=40&pageSize=10>
{'Supermarket_Id': 5, 'SalePrice': 0.7549954625692217, 'Brand_Id': 10372, 'Image': u'https://static.maiscarrinho.com/images/products/351001_2050491.jp
g', 'PromotionValue': 3.9185956363150987, 'Brand': u'Meia Encosta', 'Supermarket': u'Intermarche', 'Name': u'Vinho Tinto D\xe3o'}
Wine already exists!
SuperMarket already exists!
2017-08-21 12:54:13 [scrapy.core.scraper] DEBUG: Scraped from <200 https://maiscarrinho.com/api/search?q=vinho&pageNumber=40&pageSize=10>
{'Supermarket_Id': 16, 'SalePrice': 0, 'Brand_Id': 10372, 'Image': u'https://static.maiscarrinho.com/images/products/351001_2050491.jpg', 'PromotionVa
lue': 2.7151184435006117, 'Brand': u'Meia Encosta', 'Supermarket': u'Jumbo', 'Name': u'Vinho Tinto D\xe3o'}
2017-08-21 12:54:13 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://maiscarrinho.com/api/search?q=vinho&pageNumber=41&pageSize=10> (referer: ht
tps://maiscarrinho.com/api/search?q=vinho&pageNumber=40&pageSize=10)
2017-08-21 12:54:13 [scrapy.core.engine] INFO: Closing spider (finished)
2017-08-21 12:54:13 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 20286,
 'downloader/request_count': 42,
 'downloader/request_method_count/GET': 42,
 'downloader/response_bytes': 74782,
 'downloader/response_count': 42,
 'downloader/response_status_count/200': 42,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2017, 8, 21, 11, 54, 13, 678000),
 'item_scraped_count': 461,
 'log_count/DEBUG': 504,
 'log_count/INFO': 7,
 'request_depth_max': 40,
 'response_received_count': 42,
 'scheduler/dequeued': 41,
 'scheduler/dequeued/memory': 41,
 'scheduler/enqueued': 41,
 'scheduler/enqueued/memory': 41,
 'start_time': datetime.datetime(2017, 8, 21, 11, 53, 50, 910000)}
2017-08-21 12:54:13 [scrapy.core.engine] INFO: Spider closed (finished)
```

Código *Sqoop* referente à importação das tabelas do projeto *Scrapy*:

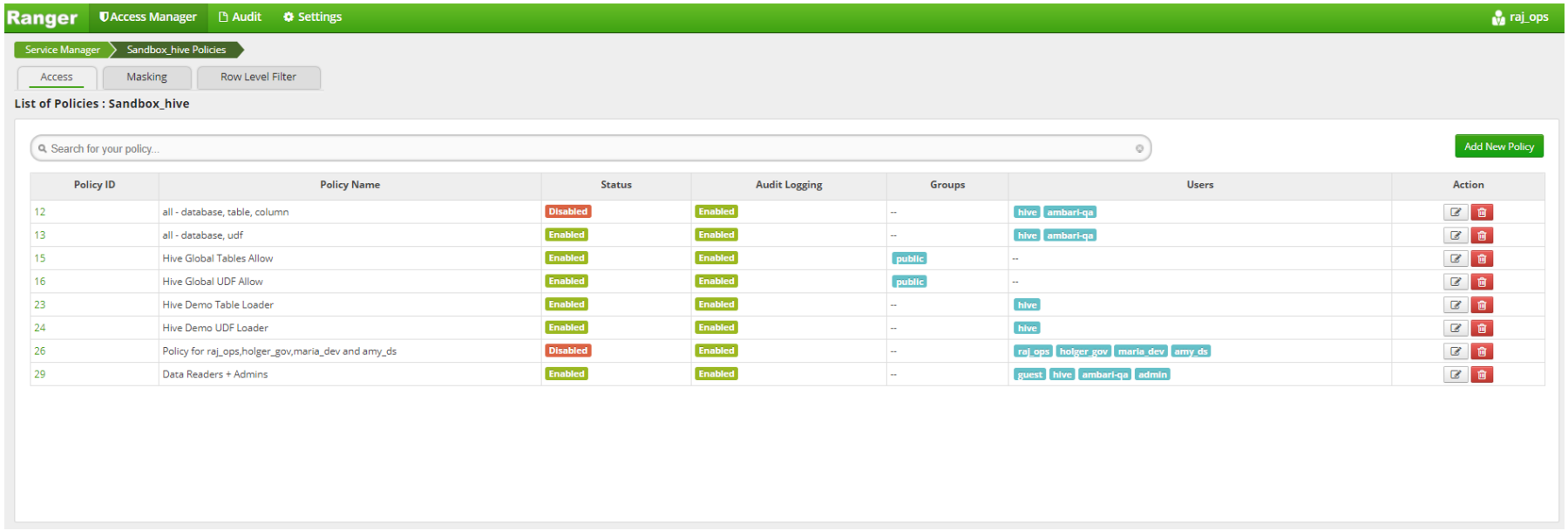
```
sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=MaisCarrinho" --username
hadoop --password-file file:///root/sqoop.password --table Wines --target-
dir /user/hive/Innovine/fact_Wines --hive-import --hive-table
Innovine.fact_wines -m 1

sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=MaisCarrinho" --username
hadoop --password-file file:///root/sqoop.password --table SuperMarket --
target-dir /user/hive/Innovine/DimSupermarket --hive-import --hive-table
Innovine.dimsupermarket -m 1

sqoop import --connect
"jdbc:sqlserver://169.254.69.51:1433;database=MaisCarrinho" --username
hadoop --password-file file:///root/sqoop.password --table WinesNames --
target-dir /user/hive/Innovine/dimWinesNames --hive-import --hive-table
Innovine.dimwinesnames -m 1
```

6.6. *SEGURANÇA*

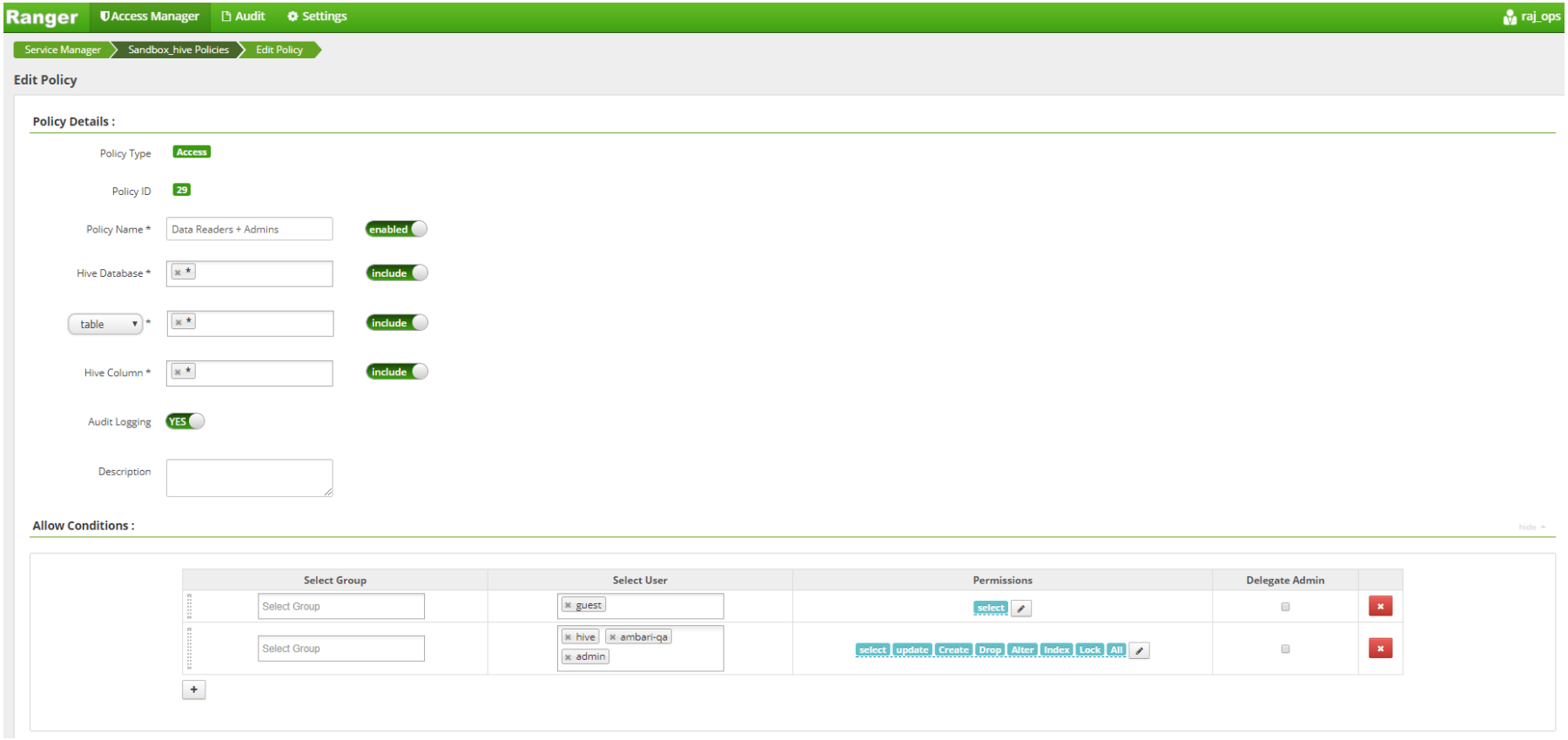
Imagem após o *login* no *Apache Ranger*:



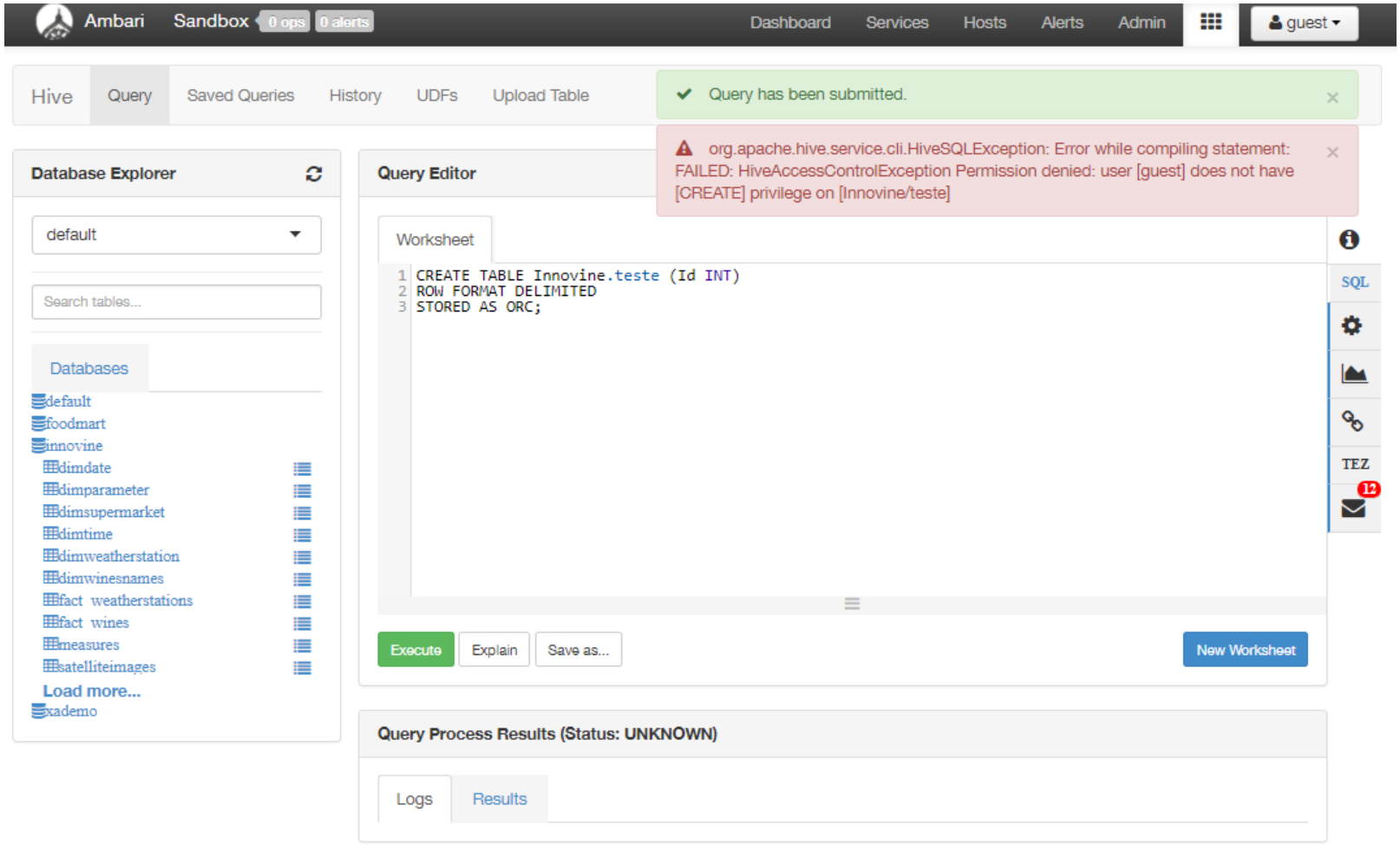
Políticas de segurança *Hive*:



Políticas de segurança *Hive* - configurações:



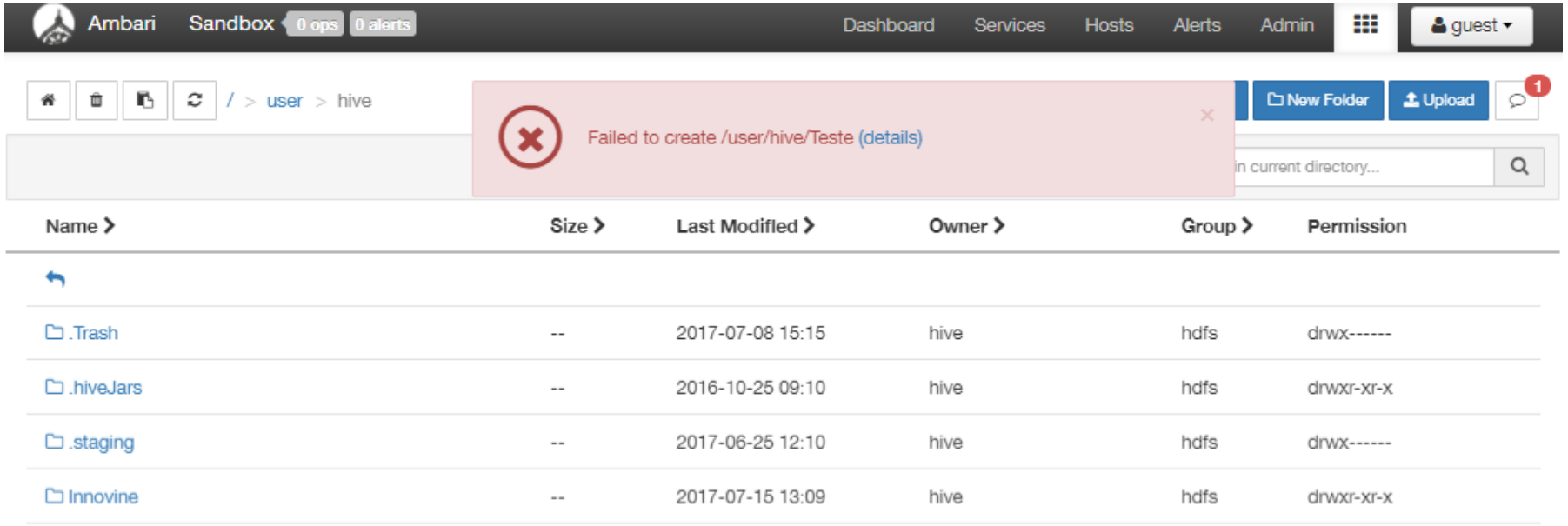
Criação de tabela Hive negada para o utilizador *guest*:



Criação de tabela HBase negada para o utilizador *guest*:

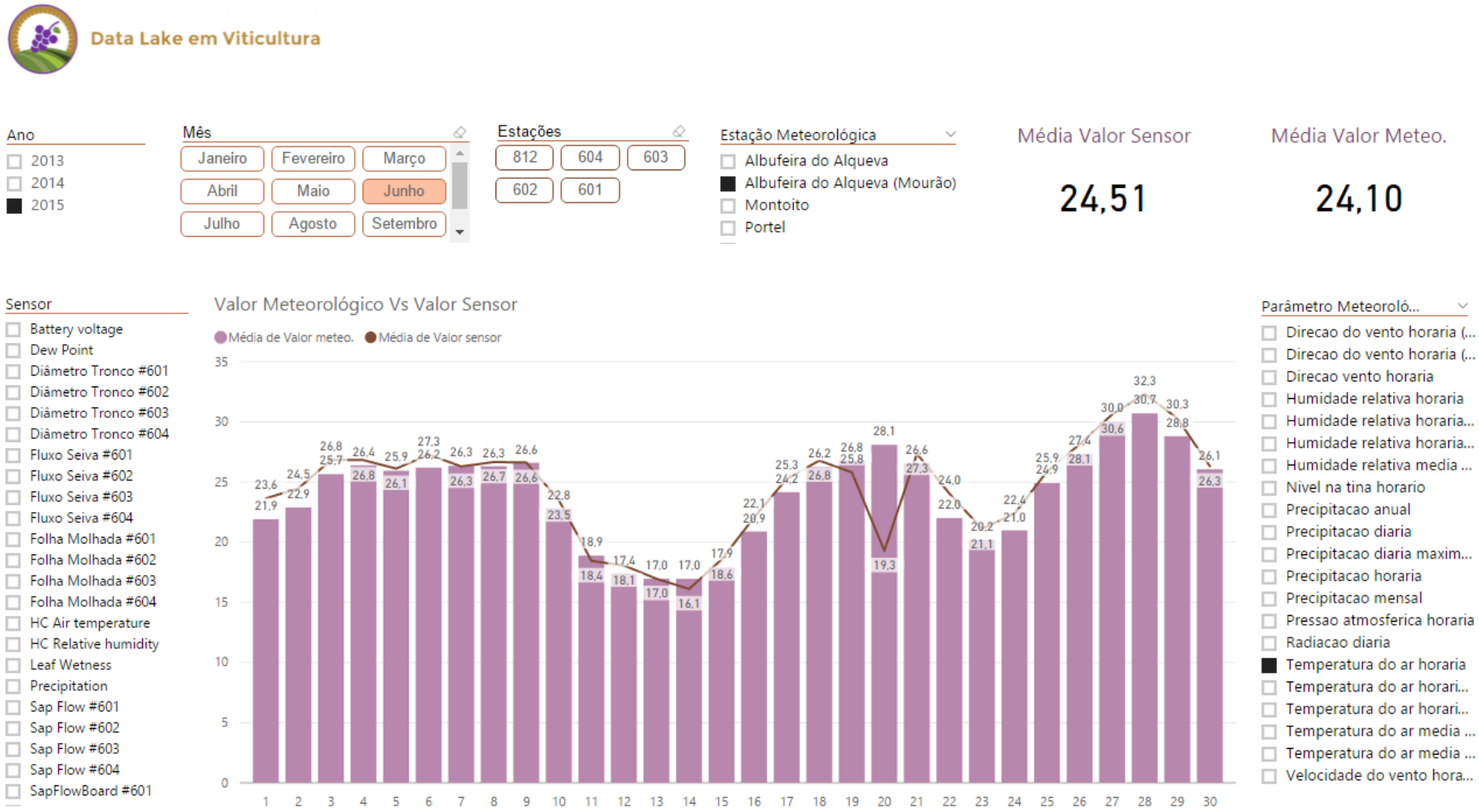
```
hbase(main):003:0> create 'measures2','measures2'

ERROR: org.apache.hadoop.hbase.security.AccessDeniedException: Insufficient permissions for user 'guest' (action=create)
```

Criação de pasta no HDFS negada para o utilizador *guest*:

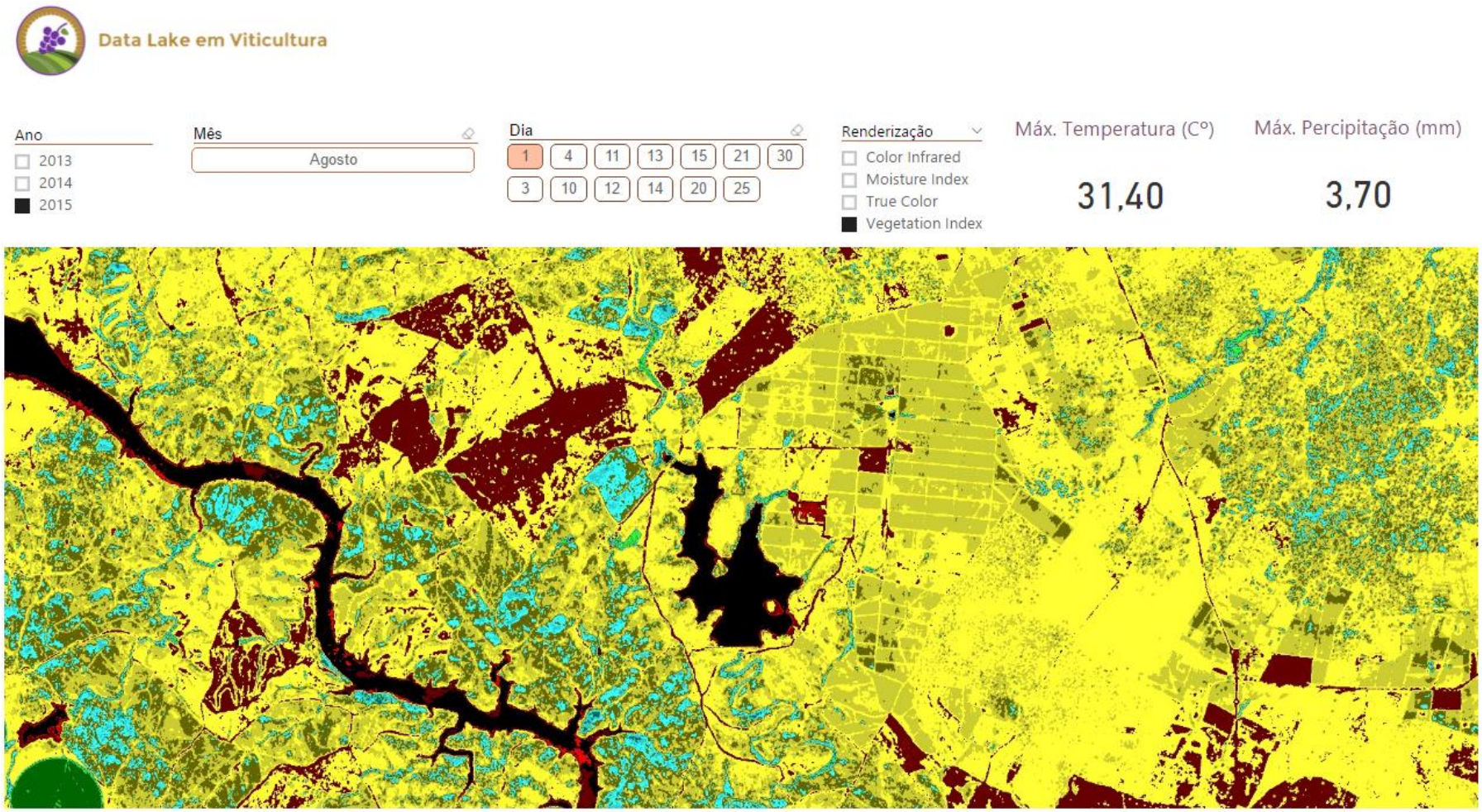


6.7. *POWER BI*

Página 1 – Análise Comparativa:



Página 2 – Análise por Imagem:



Página 3 – Análise por Mercado: