UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

THOMÁS TORREÃO DE BRITO BASTOS

ORQUESTRANDO E AUTOMATIZANDO A IMPLANTAÇÃO DE UM DATA LAKE COM ANSIBLE

Recife
2025

THOMÁS TORREÃO DE BRITO BASTOS

ORQUESTRANDO E AUTOMATIZANDO A IMPLANTAÇÃO DE UM DATA LAKE COM ANSIBLE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Orientador: Ricardo Massa Ferreira Lima

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Bastos, Thomás Torreão de Brito.
Orquestrando e automatizando a implantação de um data lake com ansible / Thomás Torreão de Brito Bastos. - Recife, 2025.
22 p. : il., tab.

Orientador(a): Ricardo Massa Ferreira Lima
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Sistemas de Informação - Bacharelado, 2025.
8,5.
Inclui referências.

1. Ansible. 2. Automação. 3. Data Lake. 4. DevOps. 5. Implantação. 6. Orquestração. I. Lima, Ricardo Massa Ferreira. (Orientação). II. Título.

000 CDD (22.ed.)

THOMÁS TORREÃO DE BRITO BASTOS

ORQUESTRANDO E AUTOMATIZANDO A IMPLANTAÇÃO DE UM DATA LAKE COM ANSIBLE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Aprovado em: 03/04/2025

BANCA EXAMINADORA

_____

Prof. Dr. Ricardo Massa Ferreira Lima (Orientador)

Universidade Federal de Pernambuco

_____

Prof. Dr. Kiev Santos da Gama (Examinador Interno)

Universidade Federal de Pernambuco

_____

Orquestrando e automatizando a implantação de um *Data Lake* com Ansible

Thomás Torreão de Brito Bastos[1], **Ricardo Massa**[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Recife, PE – Brasil

{ttbb,rmfl}@cin.ufpe.br

Abstract. *Many organizations have sought to implement Data Lakes to extract insights from their data in order to support their strategic decision-making. However, deploying Data Lakes is a challenging and labor-intensive task due to the large number of technologies involved and their interactions. This work aims to demonstrate how Ansible can assist in orchestrating and automating the complete deployment of a Data Lake, eliminating the manual installation and configuration of its multiple components. To this end, a small distributed Data Lake was designed, comprising the tools such as: Hadoop, Hive, MariaDB, PDI, PSQL, and Trino, and deployed using Ansible in a test environment consisting of six virtual machines, following the development of the necessary Ansible artifacts. The results showed that Ansible successfully completed the entire installation and configuration process of the Data Lake components in just 16 minutes, resulting in a functional deployment. Ansible proved to be a practical and efficient solution for orchestrating and deploying complex systems such as a Data Lake.* ***Keywords****: Ansible. Automation. Data Lake. DevOps. Deploy. Orchestration.*

Resumo. *Muitas organizações têm buscado implementar Data Lakes (lagos de dados) para extrair de seus dados informações que possam ajudar na tomada de decisões estratégicas. Porém, a implantação de Data Lakes é uma tarefa difícil e trabalhosa, devido à grande quantidade de tecnologias envolvidas e suas interações entre si. O objetivo deste trabalho é mostrar como o Ansible pode auxiliar a orquestrar e automatizar a implantação de um Data Lake por completo, eliminando a instalação e configuração manual das múltiplas partes do mesmo. Para isso, foi projetado um pequeno Data Lake distribuído, composto por ferramentas como: Hadoop, Hive, MariaDB, PDI, PSQL e Trino, que foi implantado pelo Ansible em um ambiente de testes composto por seis máquinas virtuais, depois do desenvolvimento dos artefatos Ansible necessários. Os resultados mostraram que o Ansible foi capaz de fazer todo o processo de instalação e configuração das partes do Data Lake em apenas 16 minutos, resultando em uma implantação funcional. O Ansible foi mostrado como uma solução prática e eficiente para orquestrar e implantar sistemas complexos como um Data Lake.* ***Palavras-chave****: Ansible. Automação. Data Lake. DevOps. Implantação. Orquestração.*

1. Introdução

Os dados são fundamentais para as organizações, pois facilitam a tomada de decisões estratégicas e aprimoram o desempenho institucional por meio de análises efica-

zes. Em contextos onde o volume de informações é muito grande (*Big Data*), apresentando desafios significativos de gestão e interpretação dos dados, surgiram os *Data Lakes*: repositórios versáteis que armazenam os dados em seu formato bruto, possibilitando um refinamento posterior conforme as necessidades analíticas da organização [NAMBIAR e MUNDRA 2022].

No entanto, a implantação (ou *deploy*) e gestão de um *Data Lake* são tarefas desafiadoras, por conta da instalação e configuração dos diversos componentes envolvidos, e em diversas máquinas. É comum a utilização de *clusters* (agrupamentos) de máquinas para que o funcionamento das aplicações seja escalonável e consiga lidar com grandes volumes de dados, aumentando assim a quantidade de ambientes a serem configurados. Implantações, quando feitas de forma manual, onde de máquina em máquina e de comando em comando se instalam e se configuram as aplicações, costumam ser trabalhosas e fadadas a erros, resultando em atrasos. E, com quantas mais máquinas se lida, como em grandes *clusters*, esse processo pode se tornar bastante difícil; estudos como o de [KAUSHIK e GUPTA 2017] reforçam essa realidade.

Ferramentas de automação, como o Ansible, Puppet e Chef, utilizam a ideia de Infraestrutura como Código, tornando possível a definição do estado desejado para um sistema através de código descrito em arquivos [CHIARI et al. 2022]. O objetivo deste Trabalho de Conclusão de Curso (TCC) é demonstrar como a ferramenta de automação Ansible pode ser utilizada para a orquestração e automação do processo de implantação de um *Data Lake* distribuído, reduzindo o tempo e os riscos inerentes a um processo manual. Para esse fim, será montado um *Data Lake* composto pelos seguintes programas principais: Hadoop, Hive, Hue, Pentaho Data Integration (PDI) e Trino, e de alguns programas auxiliares como: MariaDB e PostgreSQL (PSQL).

Para alcançar esse objetivo, elegem-se os seguintes objetivos específicos:

- Desenvolver e implementar *Playbooks* com o Ansible, para instalar e configurar os componentes do *Data Lake*;
- Descobrir se uma implantação automatizada é performática;
- Demonstrar a complexidade da implantação de um *Data Lake* e como o Ansible pode simplificar o processo.

Sendo assim, as perguntas de pesquisa que orientam este estudo são:

- Quais as vantagens de utilizar o Ansible para a implantação de um *Data Lake* em comparação à abordagem manual?
- O quão rápida é a execução de uma implantação com Ansible?
- Quais desafios técnicos e operacionais são encontrados com o uso do Ansible em uma implantação de um sistema complexo como um *Data Lake*?

2. Fundamentação teórica

2.1. *Data Lake*

O termo *Data Lake* foi cunhado pela primeira vez por James Dixon em 2010, que descreveu como um repositório natural de dados em seu estado bruto, em contraste com os tradicionais sistemas estruturados, como *Data Warehouses*, que funcionam como "água engarrafada" [Dixon 2010]. O conceito foi desenvolvido em meio ao cenário moderno de *Big*

Data, onde grandes volumes de dados, estruturados e não estruturados, e de diversas fontes são gerados ou capturados constantemente. Para extrair o conhecimento destes complexos e heterogêneos conjuntos de dados, sua importância é evidenciada, tendo a capacidade de consolidar esses conjuntos de dados em um único sistema [AZZABI et al. 2024].

Segundo [COUTO et al. 2019], por ser um desenvolvimento recente no contexto de *Big Data*, não existe ainda uma definição universalmente aceita do que é um *Data Lake*. No mesmo estudo, foi encontrado que a definição acadêmica de *Data Lake* mais citada, na literatura acadêmica, é a de [TERRIZZANO et al. 2015]. De acordo com a definição mencionada, um *Data Lake* é um conjunto de repositórios centralizados que contém vastas quantidades de dados brutos (estruturados ou não), descritos por metadados, organizados em conjuntos de dados identificáveis e disponíveis sob demanda. Os dados são utilizados para apoiar a descoberta e análise dos dados e a criação de relatórios. Foi em torno desta definição que o *Data Lake* do presente trabalho foi projetado.

Em comparação com o conceito de *Data Warehouse* (Armazém de Dados), um *Data Lake* possui vantagens de escalonamento e a habilidade de armazenar dados em vários formatos e sem o uso de *schemas* (esquemas de dados), pois utiliza uma abordagem de pós-processamento dos dados, sendo o processamento feito apenas quando as informações precisam ser extraídas do conjunto de dados [TERRIZZANO et al. 2015, NAMBIAR e MUNDRA 2022]. Por conta desses fatores, um *Data Lake* costuma ser menos custoso e trabalhoso.

2.2. Orquestração e Automação

A automação pode ser definida como o ato de buscar executar tarefas repetitivas sem ou com pouca intervenção humana, para aumentar a eficiência das mesmas e reduzir erros. Já a orquestração é a automação levada a um patamar mais alto, coordenando e gerenciando múltiplas tarefas automatizadas, para executar um fluxo de trabalho ou processo mais abrangente. É importante observar a ordem de execução das tarefas automatizadas, para que o processo como um todo possa acontecer sem problemas, pois algumas coisas dependem de outras [DILMEGANI 2025].

2.3. Infraestrutura como Código

Infraestrutura como Código (IaC) se refere à prática de utilizar código ao invés de configurações e processos manuais para provisionar infraestrutura. A sigla vem do termo em inglês, "Infrastructure as Code" [RED HAT 2022]. Os benefícios de IaC são a redução do tempo de implantação, reprodutibilidade e maior facilidade na gestão, configuração e manutenção dos sistemas de TI [CHIARI et al. 2022].

A utilização de IaC minimiza a frequência de erros humanos que costumam acontecer durante uma implantação, porém, os arquivos de código IaC podem crescer bastante, e ser gerenciados por diferentes pessoas, resultando em problemas como a implantação de erros ou defeitos que causem falhas nas implantações, problemas que requerem a utilização de técnicas de engenharia de *software* nas práticas de IaC [CHIARI et al. 2022].

IaC é considerado como um pilar fundamental para a implementação de práticas de DevOps (ver Subseção 2.4), pois acelera a entrega de aplicações e serviços para os usuários. Ferramentas de automação como o Ansible, Terraform e Puppet discutidas na Subseção 2.5 operam sob o paradigma de IaC [RAHMAN et al. 2018].

2.4. DevOps

Segundo o artigo [YARLAGADDA 2021], DevOps envolve um conjunto de atividades ou práticas integradas utilizadas na automação e na interligação dos processos de desenvolvimento de software com os profissionais de TI, com o objetivo de construir, testar e lançar entregáveis de forma rápida e confiável. DevOps é um termo integrado que se refere a desenvolvimento e operações e, culturalmente, representa uma interconexão entre desenvolvedores e operadores, cujas funções eram originalmente baseadas em silos.

Tradicionalmente, desenvolvedores tinham que implantar manualmente vários programas depois da criação das máquinas. DevOps chegou para resolver o problema com uma automação inteligente dos sistemas [YARLAGADDA 2021], pois a prática de automação é parte essencial da filosofia de DevOps, e o uso de ferramentas de automação é indissociável da mesma. Logo, a utilização de Ansible, Chef, Puppet, entre outros, tornou-se essencial dentro do movimento. Sendo assim, a introdução de DevOps nas organizações trouxe a automação da maioria das operações em infraestruturas de TI [YARLAGADDA 2021], resolvendo os problemas de complexidade em lidar com múltiplas máquinas.

2.5. Ansible

O Ansible é uma ferramenta de orquestração e automação de ambientes de TI, que auxilia no provisionamento, configuração e gerenciamento de máquinas de forma eficiente.

O Ansible foi desenvolvido em torno dos seguintes princípios [ANSIBLE 2025]:

- Arquitetura *agent-less*: não é necessário instalar agentes nas máquinas-alvo (ou seja, sem necessidade de programas pré-instalados), facilitando o uso e manutenção;
- Simplicidade: utiliza "*Playbooks*" (descrito abaixo) escritos na linguagem YAML, resultando em um código que parece uma documentação. Também é decentralizado em seu acesso aos sistemas remotos;
- Escalabilidade e flexibilidade: facilmente escala os sistemas e suporta múltiplos sistemas operacionais e plataformas;
- Idempotência: quando o sistema está no estado desejado o Ansible não muda nada.

Uma das características mais importantes na comparação do Ansible com outras ferramentas de automação é justamente sua característica *agent-less*, enquanto ferramentas como *SaltStack*, *Chef* e *Puppet* usam *agents*, ferramentas como o Ansible e Terraform não as utilizam. Essa abordagem *agent-less* aumenta a compatibilidade com múltiplos sistemas, reduz a complexidade e simplifica o uso. Entre as ferramentas *agent-less*, o *Terraform* é mais voltado para o provisionamento de infraestrutura, enquanto o Ansible é mais voltado para o gerenciamento de configuração, como a instalação e configuração de programas em ambientes já provisionados.

Pelas qualidades mencionadas, o Ansible foi escolhido como a plataforma de automação para o desenvolvimento deste projeto. No Ansible, existem alguns conceitos cujos termos serão mencionados adiante, e consequentemente necessitam de explicação; são eles [ANSIBLE 2025]:

- Ansible *Playbooks*: são arquivos no formato da linguagem YAML. Esses arquivos contém as instruções do programa, chamadas de *Tasks*. Chamaremos de *Playbooks* os múltiplos arquivos que serão criados;
- Ansible *Tasks*: as menores unidades de ação de um *Playbook*. Por exemplo, uma Task pode definir como será feita a criação de um diretório no remoto;
- Ansible *Modules*: são programas que executam *Tasks* no remoto, por exemplo, o módulo "apt" serve para executar instalações ou outras ações envolvendo o gerenciador de pacotes de mesmo nome, presente em vários sistemas Linux;
- Ansible *Roles*: permitem o carregamento automático de variáveis, arquivos, *Tasks* e outros artefatos Ansible em um sistema de pastas padronizado, permitindo a fácil reutilização e compartilhamento dos mesmos. Um *Role* vai possuir um conjunto de *Playbooks*;
- Ansible *Tags*: a depender da estrutura dos *Playbooks* e dos *Roles*, existe a necessidade de só executar uma parte dos mesmos, assim é possível utilizar *Tags* para executar apenas a parte desejada do programa. Por exemplo, se um *Role* for separado entre *Playbooks* de instalação e de configuração, e a instalação já tiver sido feita, convém usar *Tags* para executar apenas a parte de configuração.

Os conceitos descritos serão escritos da seguinte forma no decorrer do artigo: *Playbook(s)*, *Task(s)*, *Module(s)*, *Role(s)* e *Tag(s)*.

2.6. Sistemas distribuídos

Na definição de [COULOURIS et al. 2013], um sistema distribuído é um modelo de sistema computacional no qual os componentes de programas estão localizados em diferentes máquinas, comunicando-se e coordenando suas ações por meio de mensagens ou outros mecanismos de interação.

Os principais benefícios de sistemas distribuídos são [RICHMAN 2023]: (I) escalabilidade horizontal, um *Data Lake* precisa lidar com crescentes volumes de dados. Com uma arquitetura distribuída, é possível adicionar mais máquinas ao *cluster* (para aumentar sua capacidade de processamento e armazenamento) conforme a necessidade, sem precisar reestruturar as máquinas existentes; (II) desempenho, com o uso de ferramentas de armazenamento e consultas distribuídas, os conjuntos de dados podem ser divididos em partes menores, a serem processadas simultaneamente em diferentes máquinas, resultando em ganhos de desempenho; (III) resiliência, um sistema distribuído torna possível a replicação dos dados nas diferentes máquinas. Consequentemente, problemas em uma máquina não comprometem a integridade do sistema como um todo.

Apesar das vantagens, a estrutura apresenta alguns desafios, como questões de complexidade e consistência: instalar e configurar múltiplos programas em múltiplas máquinas, garantindo a consistência de versões e configurações (de programa, de sistema e de conectividade entre as máquinas) é um processo complexo e trabalhoso, dependendo do tamanho do sistema.

2.7. Trabalhos relacionados

A atual seção elenca trabalhos com temática semelhante, que serviram como base para o desenvolvimento do presente projeto, que se assemelham com o mesmo pelo fato de que neles foram aplicadas ferramentas de automação para a instalação, configuração e gerenciamento de infraestrutura de Tecnologia da Informação (TI).

No artigo [SINGH e PURWAR 2019], é discutida a importância da implantação rápida e consistente de serviços para reduzir custos, mencionando como os times de DevOps estão sob grande destaque para este fim. O estudo evidencia como a utilização de ferramentas de automação, como o Chef, é capaz de reduzir o tempo do processo de implantação na Nuvem de horas para minutos. O estudo se diferencia do presente projeto pelo fato de que focou em conceitos gerais e na demonstração da importância da automação para a eficiência operacional, enquanto este projeto apresenta a utilização prática da ferramenta Ansible na implantação de um *Data Lake* complexo, demonstrando a redução do tempo de implantação.

O artigo [ELRADI 2023] faz uma análise similar, mostrando a automação do processo de instalação de uma aplicação *web* utilizando o Ansible. O artigo encontrou uma redução de 5,6 vezes no tempo de implantação da aplicação com a automação em relação a uma instalação manual. Em comparação com o presente projeto, o artigo mencionado analisou o uso do Ansible em uma implantação mais simples em apenas uma máquina, enquanto este projeto analisou uma implantação consideravelmente mais complexa, envolvendo múltiplas máquinas que formam um *Data Lake*.

Em [KAUSHIK e GUPTA 2017] foi comparado o tempo entre uma implantação manual e uma implantação automatizada com Ansible, em uma máquina, e entre uma implantação manual e uma implantação automatizada com Ansible em um *cluster* de máquinas. O estudo encontrou vantagens consideráveis na utilização do Ansible, incluindo uma boa redução no tempo de implantação. Em comparação com o presente projeto, o estudo não realizou a implantação em um sistema complexo, e não fez uma demonstração detalhada da configuração do Ansible.

3. Metodologia

A presente seção descreve como foi desenvolvido o projeto, os componentes escolhidos, o ambiente de testes, a configuração do Ansible, a orquestração dos processos e os testes realizados para a validação do sucesso da automação; com o objetivo de descrever o processo de implantação de um *Data Lake* funcional, automatizado utilizando o Ansible e outras ferramentas de código aberto. A Figura 1 apresenta a *pipeline* do projeto.

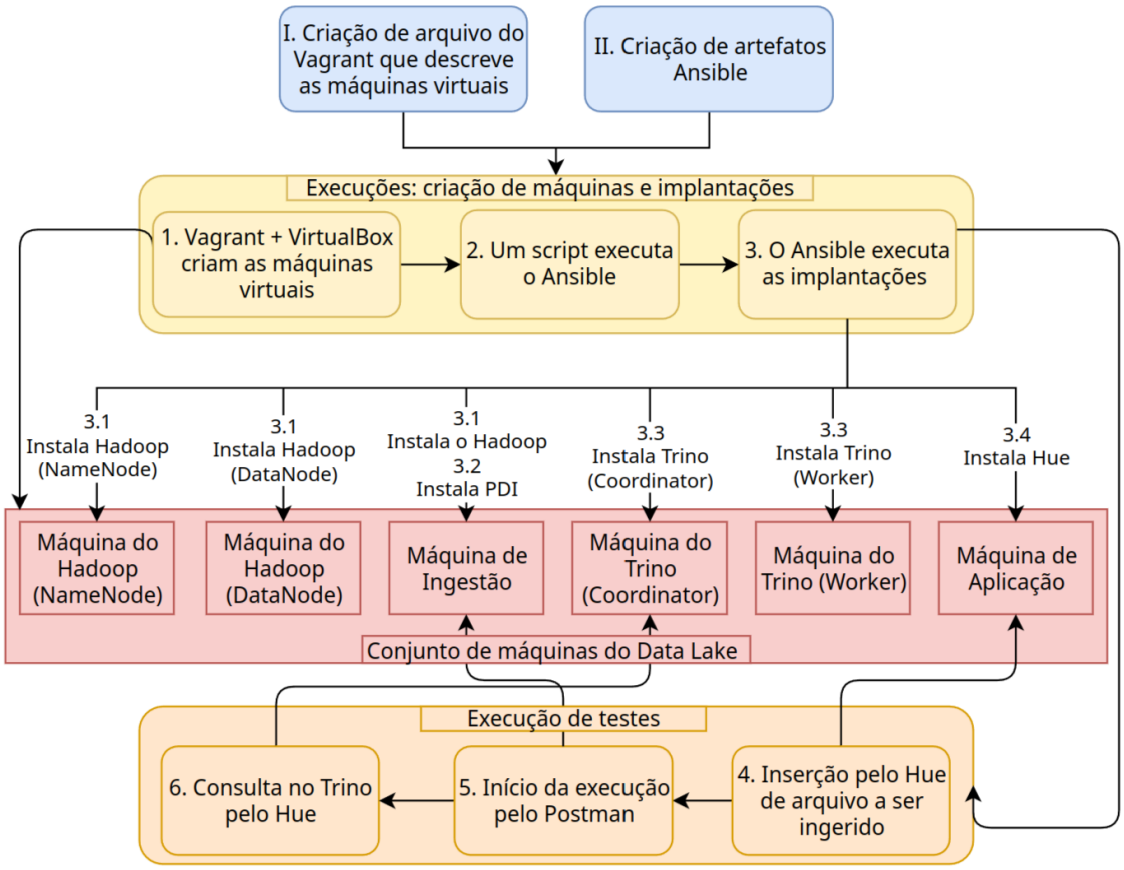

Figura 1. *Pipeline* do projeto

3.1. Componentes do *Data Lake*

Para a construção do *Data Lake*, foram escolhidos os seguintes componentes, com suas descrições (tiradas de suas respectivas documentações), justificativas para a escolha e funções no *Data Lake* projetado:

- Hadoop
  - Descrição: é um *framework* para processamento distribuído de grandes volumes de dados em *clusters* de computadores usando modelos de programação simples, é projetado para escalar de servidores únicos para milhares de máquinas e ser tolerante a falhas. De acordo com o estudo de [COUTO et al. 2019], entre as escolhas para montar a arquitetura de *Data Lakes*, o ecosistema Hadoop é o mais utilizado;
  - Função no projeto: foi escolhido para o armazenamento distribuído dos dados, em seu HDFS (*Hadoop Distributed File System*), servindo como base para o *Data Lake*.
- Hive
  - Descrição: acompanha o Hadoop e é um programa de *Data Warehouse* projetado para ler, escrever e gerenciar grandes conjuntos de dados distribuídos usando sintaxe SQL. O Hive armazena metadados (descrições dos dados armazenados) no *Metastore*;

  - Função no projeto: será utilizado por conta do *Metastore*, ou seja, para guardar os metadados descritivos dos dados armazenados no Hadoop, para que o Trino saiba como realizar consultas no HDFS. O *Metastore* necessita de algum banco de dados, para isso foi escolhido o MariaDB.
- PDI
  - Descrição: é uma ferramenta responsável por processos de ETL (extração, transformação e carga de dados) que facilitam o processo de captura, limpeza e armazenamento de dados em um formato uniforme e consistente. O PDI vem acompanhado de um programa chamado Carte, que permite executar os artefatos do PDI através de uma requisição HTTP;
  - Função no projeto: orquestrar a ingestão de um arquivo no formato CSV. Essa ingestão consistirá em: extrair (o CSV do local inicial para um local de processamento), transformar (de CSV em Parquet, um formato de arquivos eficiente para uso em *Data Lakes*) e inserir os dados necessários no Hadoop, Hive e Trino.
- Trino
  - Descrição: é um mecanismo de consulta SQL distribuído projetado para consultar grandes conjuntos de dados distribuídos em uma ou mais fontes de dados heterogêneas;
  - Função no projeto: servirá como o motor de busca que fará as consultas no HDFS, consultando o *Metastore*. Foi escolhido por oferecer agilidade, flexibilidade e otimização do desempenho das consultas.
- Hue
  - Descrição: é um assistente SQL para consultas em bancos de dados e *Data Warehouses*;
  - Função no *Data Lake*: servirá para simular uma aplicação, onde serão feitos os seguintes passos: a inserção inicial de um arquivo CSV no Hadoop, e, após a ingestão, consultas no Trino, retornando os dados inseridos no *Data Lake*. Em conjunto com o Hue, será utilizado o programa Postman para dar início ao processo de ingestão, fazendo uma requisição HTTP no Carte. O Hue depende de um banco de dados relacional para suas operações, para isso foi escolhido o PSQL.

As escolhas foram feitas baseando-se em critérios como compatibilidade, código aberto e simplicidade de uso.

3.2. Ambiente de testes

Foram utilizados, em combinação, os programas *Vagrant* (que permitem a configuração de máquinas em forma de código) e *VirtualBox* (programa que cria máquinas virtuais) – para a criação de seis máquinas virtuais simulando o *Data Lake* distribuído, onde serão testadas a instalação e configuração dos programas, usando o Ansible.

Na definição das máquinas necessárias, foi levada em consideração a necessidade de simular apenas um pequeno *cluster*, representando as partes mínimas para uma instalação funcional, de maneira que fique evidente a possibilidade de uso dos *Playbooks* na instalação de um *cluster* distribuído real. Usualmente, em um *cluster* de aplicações como o Hadoop e o Trino, tem-se pelo menos uma máquina dedicada para a coordenação/gerenciamento e outra(s) dedicada(s) para o armazenamento/processamento de dados. Por isso, foi decidido criar uma máquina para o *NameNode* (coordena o *cluster*) e

outra para o *DataNode* (armazena os dados) do Hadoop. Já no caso do Trino, uma para o *Coordinator* (coordena o *cluster*) e uma para um *Worker* (executa tarefas e processa os dados).

Assim, o ambiente ficou dividido da seguinte forma:

- Máquina de ingestão onde ficará o PDI;
- Maquina do *NameNode* do Hadoop, Hive e MariaDB;
- Máquina do *DataNode* do Hadoop;
- Máquina do *Coordinator* do Trino;
- Máquina do *Worker* do Trino;
- Máquina de aplicação onde ficará o Hue.

Ver na Tabela 1 as especificações das máquinas criadas. Foram escolhidos 1 GB de memória RAM e 1 vCPU (CPU virtual) como especificação mínima e a RAM dos programas ajustada adequadamente. A máquina do *NameNode* do Hadoop possui um pouco mais de RAM, pois vai comportar três programas (Hadoop, Hive e MariaDB).

Tabela 1. Tabela de máquinas

Função	Hostname	IP	vMEM	vCPU	S.O.
Máquina do Ingestor	tcc-ingestor	192.168.57.2	1024MB	1	Ubuntu 20
Máquina do Hadoop (Namenode)	tcc-hadoop-namenode	192.168.57.3	1280MB		
Máquina do Hadoop (Datanode)	tcc-hadoop-datanode1	192.168.57.4	1024MB		
Máquina do Trino (Coordinator)	tcc-trino-coordinator	192.168.57.5			
Máquina do Trino (Worker)	tcc-trino-worker1	192.168.57.6			
Máquina da Aplicação	tcc-aplicacao	192.168.57.8			

A Figura 2 representa o fluxo de operações para a ingestão, inserção e consulta dos dados no *Data Lake*.

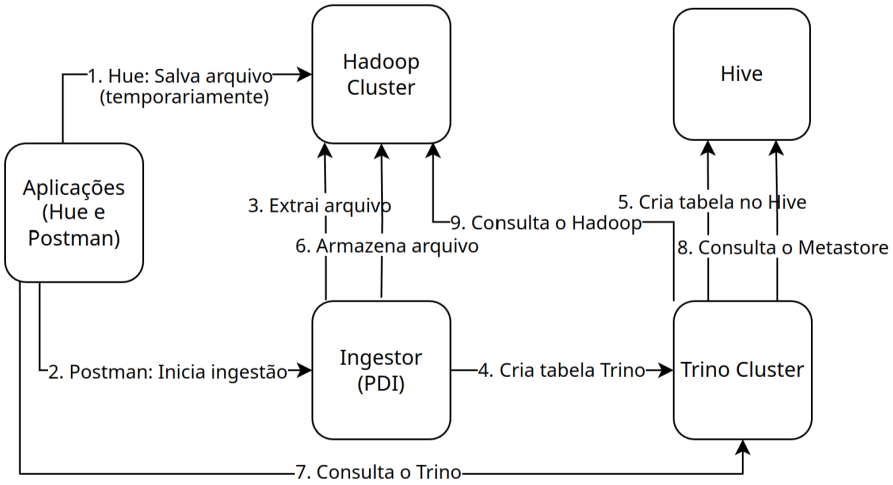

Figura 2. Fluxo de operações

A Tabela 2 representa o cenário da implantação distribuída, onde são mostradas as máquinas, suas aplicações (versões e modo de instalação) e as dependências das aplicações (versões e modo de instalação). A criação das máquinas do ambiente foi realizada pelo *Vagrant* em aproximadamente quatro minutos.

Tabela 2. Tabela de instalações

Máquinas		Aplicações	Versões/ Tags	Métodos de instalação	Dependências	Versões (dependências)	Métodos de instalação (dependências)
Máquina de ingestão		PDI	9.2	Instalação nativa	Java (OpenJDK)	1.8.0_442	Pacote apt
Cluster do Hadoop	Máquina do Namenode	Hadoop	2.10.2				
		Hive	2.3.9		MariaDB	15.1	
	Máquina do Datanode	Hadoop	2.10.2		Java (OpenJDK)	1.8.0_442	
Cluster do Trino		Trino	454		Java (Temurin)	22.0.2	
Máquina da aplicação		Hue	20250314-140101	Docker	PSQL	12	

3.3. Configuração do Ansible

3.3.1. Inventário

Como primeiro passo, deve-se criar um inventário (representado parcialmente na Figura 3) onde serão descritas várias informações de cada máquina do cluster, como: IP, *hostname*, porta SSH, usuário da máquina-alvo, etc. Esses dados serão utilizados pelo Ansible para fazer as conexões SSH nas máquinas.

A possibilidade de agrupar máquinas no inventário se mostrou útil, pois com o uso das mesmas é possível organizar melhor as execuções das *Tasks*, separando as que devem executar, por exemplo, em todo o *cluster* do Hadoop das que devem executar apenas no *NameNode* do Hadoop.

```yaml
HadoopNamenode:
  hosts:
    tcc-hadoop-namenode.datalake.local:
      short_name: tcc-hadoop-namenode
      ansible_host: 192.168.57.3
      ansible_port: 22
      ansible_user: vagrant

HadoopDatanode:
  hosts:
    tcc-hadoop-datanode1.datalake.local:
      short_name: tcc-hadoop-datanode1
      ansible_host: 192.168.57.4
      ansible_port: 22
      ansible_user: vagrant

Hadoop:
  children:
    HadoopNamenode: {}
    HadoopDatanode: {}
```

Figura 3. Fragmento do inventário mostrando o uso de grupos

3.3.2. *Roles*

Para construir um *Role*, em primeiro lugar, se cria uma pasta "*roles*", e dentro as pastas dos *Roles* a serem desenvolvidos. Para o projeto, foram criados um total de quatro *Roles*: "hadoop-Hive", "*hue*", "pdi" e "trino". Cada um agrupa os *Playbooks* relevantes à sua função e similaridade. Por exemplo, como o Hadoop e o Hive são interdependentes, possuem muitas variáveis em comum, e por isso foram colocados juntos num *Role*. A estrutura da pasta "*roles*" pode ser vista na Figura 4.

Para visualizar a estrutura de um *Role*, vejamos o exemplo do *Role* "pdi", dentro da pasta dele criamos as pastas "*defaults*", "*files*", "*tasks*", "*templates*" e "*vars*" (ver Figura 5). Existem mais possibilidades de pastas previstas no conceito de *Roles*, porém não serão abordadas neste projeto.

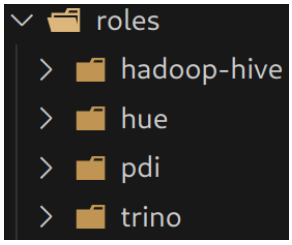

Figura 4. Pasta "*roles*"

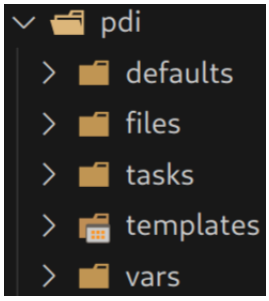

Figura 5. *Role* "pdi"

Na pasta "*defaults*" é criado um arquivo "main.yml" (ver Figura 6), onde são definidas as variáveis a serem utilizadas pelas *Tasks*. Ele será lido automaticamente ao chamar o *Role*. Na pasta "vars", são colocados arquivos com variáveis (como na pasta "*defaults*"), porém, as variáveis nela têm uma prioridade maior do que as da pasta "*defaults*".

```
3   # Linux
4   PENTAHO_USUARIO: pentaho
5   PENTAHO_GRUPO: pentaho
6   PENTAHO_SENHA: "$6$N0bPAiw.vWY7C3fi$0JUGfSjDuvt9sH43EVD1G/RfJSxJ2v
7
8   # Programa
9   PENTAHO_VERSAO: 9.2 # Versões suportadas: ver vars/main.yml
10  PENTAHO_CARTE_USUARIO: carte
11  PENTAHO_CARTE_SENHA: password
12  PENTAHO_CARTE_PORTA: 8080
13  PENTAHO_MEMORIA_MINIMA: 160
14  PENTAHO_MEMORIA_MAXIMA: 320
15
16  # Modo de download
17  PENTAHO_MODO_DE_DOWNLOAD: local # remoto ou local
18  PENTAHO_DESTINO_DO_DOWNLOAD_LOCAL: "{{ role_path }}/files" # se o
19
20  # Caminhos
21  PENTAHO_CAMINHO_BASE: /usr/local/pdi
22  PENTAHO_CAMINHO_DATA_INTEGRATION: /usr/local/pdi/data-integration
```

Figura 6. Fragmento do arquivo de variáveis do *Role* "pdi"

Na pasta "*files*" são colocados os arquivos necessários para o funcionamento dos *Playbooks*, como arquivos que serão copiados para a(s) máquina(s) alvo (ver Figura 7).

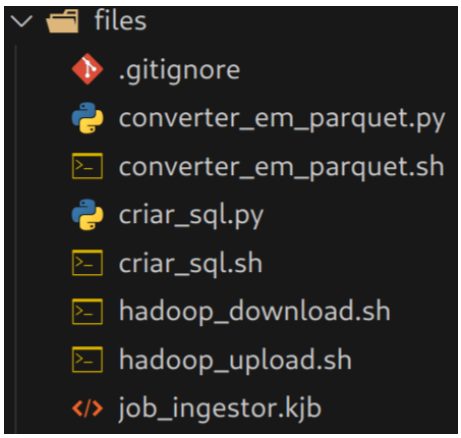

Figura 7. Fragmento da pasta de arquivos do *Role* "pdi"

Na pasta "*tasks*" (ver Figura 8) é criado o arquivo principal "main.yml", que será chamado automaticamente ao chamar o *Role*. Para tudo ficar mais organizado e encur-

tar os arquivos YAML, também foram criados outros arquivos que serão chamados pelo arquivo principal.

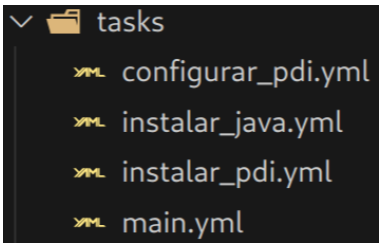

Figura 8. Pasta das *Tasks* no *Role* "pdi"

Na Figura 9 pode ser visto como o arquivo principal chama os outros arquivos de *Tasks*, e como *Tags* podem ser utilizadas para selecionar apenas o que se quer executar, ao passar essas *tags* para o comando que chama o *Role*.

```
3   - import_tasks: instalar_java.yml
4     tags: pdi-java
5
6   - import_tasks: instalar_pdi.yml
7     tags: pdi-instalar
8
9   - import_tasks: configurar_pdi.yml
10    tags: pdi-configurar
```

Figura 9. YAML principal de *Tasks* do *Role* "pdi"

Os outros arquivos da pasta "*tasks*" foram divididos entre arquivos de instalação dos programas e arquivos de configuração dos mesmos. Na Figura 10 pode-se ver um fragmento do YAML de instalação do PDI, que executa vários *Modules* como "*group*", "*user*" e "*file*". Como exemplo de orquestração, note que o uso do segundo *Module* requer a existência de um usuário "hadoop" na máquina do PDI, resultando assim na necessidade da execução do *Role* do Hadoop antes deste.

```yaml
- name: GROUP | Criar grupo
  group:
    name: "{{ PENTAHO_GRUPO }}"
    state: present

- name: USER | Criar usuário
  user:
    name: "{{ PENTAHO_USUARIO }}"
    group: "{{ PENTAHO_GRUPO }}"
    password: "{{ PENTAHO_SENHA }}"
    shell: /bin/bash
    state: present

- name: USER | Adicionar grupo hadoop ao grupo do pentaho
  user:
    name: pentaho
    groups: hadoop
    append: yes
```

Figura 10. Fragmento do YAML de instalação do PDI

Na pasta "templates" (ver Figura 11), como na pasta "*files*", são colocados arquivos que serão copiados para a(s) máquina(s) alvo, porém, no caso dos *templates* (modelos), é utilizada a linguagem Jinja2 (linguagem de geração de modelos dinâmicos), que utilizará as variáveis definidas na pasta "defaults" para gerar os arquivos finais.

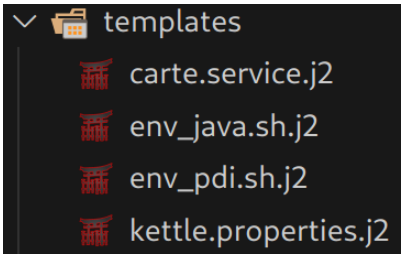

Figura 11. Pasta de *templates* do *Role* PDI

Características gerais dos *Roles* desenvolvidos:

- Instalações: a maioria dos programas são "instalados nativamente" (instalação diretamente no sistema operacional, sem utilização de *containers* ou emulação). A exceção foi o Hue, que é instalado utilizando o *Docker*;
- Configuração: criação de pastas onde serão guardados arquivos, como *scripts* de execução, criação dos arquivos que configuram os programas; criação de serviços para que as aplicações iniciem sozinhas, etc;
- Utilização de *templates* para maior facilidade em modular os arquivos que serão criados nas máquinas;
- Envio de arquivos auxiliares para completar as instalações.

Sumário dos *Roles* desenvolvidos:

- O *Role* “hadoop/hive” instala e configura as aplicações Hadoop, Hive, MariaDB e suas dependências, como Java;
- O *Role* “hue” instala e configura as aplicações Hue e PSQL;
- O *Role* “pdi” instala e configura as aplicações PDI e sua dependência, o Java;
- O *Role* “trino” instala e configura as aplicações Trino e sua dependência, o Java.

Seguem alguns pontos reflexivos sobre *Roles*:

- A estrutura modular dos *Roles* demonstrou ser de grande auxílio para gerenciar a complexidade dos *Roles*, que podem ficar bastante grandes;
- As *Tags*, aliadas à separação das *Tasks* em *Playbooks* agrupados por função (por exemplo, entre *Playbooks* de instalação e configuração) também são de grande serventia, pois, com elas, é possível selecionar um grupo menor de *Tasks* a ser executado durante o processo de desenvolvimento dos *Roles*;
- O uso de arquivos Jinja2 na pasta “*templates*” foi de grande utilidade, pela flexibilidade que eles permitem, especialmente nos *clusters* do Hadoop e Trino, pois parâmetros como IPs variam entre as máquinas;
- Mesmo organizando as variáveis por *Roles*, o número delas tendem crescer rapidamente, resultando em dificuldades para gerir-las.

3.3.3. Desempenho

É interessante a possibilidade de evitar parte da variabilidade inerente às conexões de rede, tanto das máquinas-alvo quanto dos provedores dos arquivos, baixando arquivos de instalação maiores e os colocando nas pastas “*files*” dos *Roles*, apenas extraindo-os nas máquinas-alvo. Sendo assim, o teste de execução será feito com os arquivos do Hadoop, Hive, Trino e PDI (3,16 *gibibytes* de tamanho total) já baixados na máquina que executará o Ansible. Ao mesmo tempo, a capacidade de baixar os arquivos diretamente nas máquinas-alvo ou no computador que roda o Ansible foi implementada nos *Roles*, e a escolha dentre os dois métodos pode ser feita apenas modificando uma variável, ilustrando assim a flexibilidade do programa em aceitar múltiplas estratégias de execução.

É interessante a possibilidade de aprimorar o desempenho do Ansible modificando algumas configurações no arquivo “ansible.cfg”, que o Ansible carrega automaticamente. Esse arquivo também pode modificar várias outras características da ferramenta. O projeto será executado com as configurações vistas na Figura 12.

```
[defaults]
interpreter_python = /usr/bin/python3
gathering = smart
fact_caching = jsonfile
fact_caching_connection = .facts_cache
fact_caching_timeout = 21600
pipelining = True
forks = 16
display_skipped_hosts = false

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

Figura 12. Arquivo "ansible.cfg"

3.3.4. Orquestração

Para o correto funcionamento da implantação em uma única execução dos *Roles*, é necessário orquestrar corretamente a ordem de execução dos mesmos. Para isso, é necessário considerar as dependências entre os componentes do *Data Lake*. Por exemplo, como já mencionado, o *Role* do PDI necessita da prévia instalação do Hadoop na máquina de ingestão; bem como o *Role* do Trino depende da instalação do Hive para que o Trino, ao ser iniciado, possa acessar o Hive.

Na Figura 13, observa-se o *Playbook* "principal": "deploy.yml", que chama os *Roles* nas máquinas-alvo. Com o uso de *Tags*, a ordem neste arquivo não vai importar, pois, junto com o controle dos *hosts*, é possível modificar quais *Roles* vão executar, e em quais máquinas. Por exemplo, para executar apenas o *Role* do Trino, nas máquinas do grupo "Trino", usa-se o comando: "ansible-playbook -i inventory/tcc-datalake.yml deploy.yml –tags "trino""

```
- name: Deploy do role hadoop/hive
  hosts: Hadoop,Hive,Ingestor
  become: true
  gather_facts: true
  roles:
    - role: hadoop-hive
      tags: hadoop

- name: Deploy do role pdi
  hosts: Ingestor
  become: true
  gather_facts: true
  roles:
    - role: pdi
      tags: pdi
```

Figura 13. Fragmento do *Playbook* "principal"

Para a simplificação do processo, foi criado um *script* na linguagem *Shell* auxiliar (ver Figura 14). Ele serve para encapsular os comandos Ansible que precisam ser rodados. Assim, uma implantação completa pode ser iniciada com apenas um comando. A combinação do *Playbook* "principal", *Tags* e o *script* permite uma implantação mais simples e adaptável.

```
ble > deploy.sh
1   #!/bin/bash
2
3
4   ansible-playbook -i inventory/tcc-datalake.yml deploy.yml --tags "hosts"
5   ansible-playbook -i inventory/tcc-datalake.yml deploy.yml --tags "hadoop"
6   ansible-playbook -i inventory/tcc-datalake.yml deploy.yml --tags "pdi"
7   ansible-playbook -i inventory/tcc-datalake.yml deploy.yml --tags "trino"
8   ansible-playbook -i inventory/tcc-datalake.yml deploy.yml --tags "aplicacao"
```

Figura 14. *Script Shell* auxiliar

4. Resultados

Com o desenvolvimento dos artefatos do projeto concluído, foi realizada a criação das seis máquinas virtuais descritas na Tabela 1 por um comando Vagrant, e depois foi executada a automação do Ansible nas mesmas, utilizando o *script* visto na Figura 14. Um fragmento do retorno da execução do Ansible pode ser visualizado na Figura 15.

```
→  ansible ./deploy.sh

PLAY [Atualizar /etc/hosts] ********************

TASK [Gathering Facts] *************************
ok: [tcc-aplicacao.datalake.local]
ok: [tcc-ingestor.datalake.local]
ok: [tcc-trino-worker1.datalake.local]
ok: [tcc-hadoop-namenode.datalake.local]
ok: [tcc-hadoop-datanode1.datalake.local]
ok: [tcc-trino-coordinator.datalake.local]

TASK [TEMPLATE | Atualizar /etc/hosts] *********
changed: [tcc-ingestor.datalake.local]
changed: [tcc-aplicacao.datalake.local]
changed: [tcc-hadoop-datanode1.datalake.local]
changed: [tcc-hadoop-namenode.datalake.local]
changed: [tcc-trino-worker1.datalake.local]
changed: [tcc-trino-coordinator.datalake.local]
```

Figura 15. Pequeno fragmento da implantação em curso

Para testar o êxito da implantação, foi feito um teste fim a fim do *Data Lake* implantado seguindo o plano da *pipeline* descrito na Figura 1. Inicialmente, foi inserido um arquivo com dados aleatórios no formato CSV, pela interface do Hue, na pasta "/upload" do Hadoop, um local temporário antes do início da ingestão (Figura 16). Em seguida, foi efetuada uma requisição HTTP pela aplicação Postman, solicitando ao Carte – responsável por coordenar os processos do PDI – o início da ingestão do arquivo (Figura 17). Para verificar o andamento da ingestão, foi consultada a *interface* do Carte, que oferece

a visualização dos processos executados no PDI (Figura 18). O uso do Postman e, como previamente mencionado, o uso do Hue, substituem, no cenário de testes deste projeto, aplicações mais robustas que as organizações normalmente utilizam para iniciar operações de ingestão e consulta de dados e acompanhar o andamento das ingestões em *Data Lakes*.

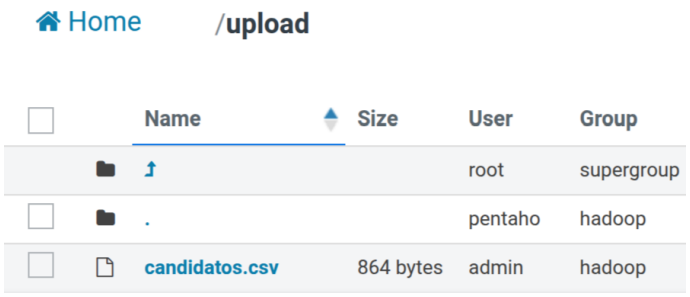

Figura 16. Pela interface do Hue: arquivo CSV inserido no Hadoop

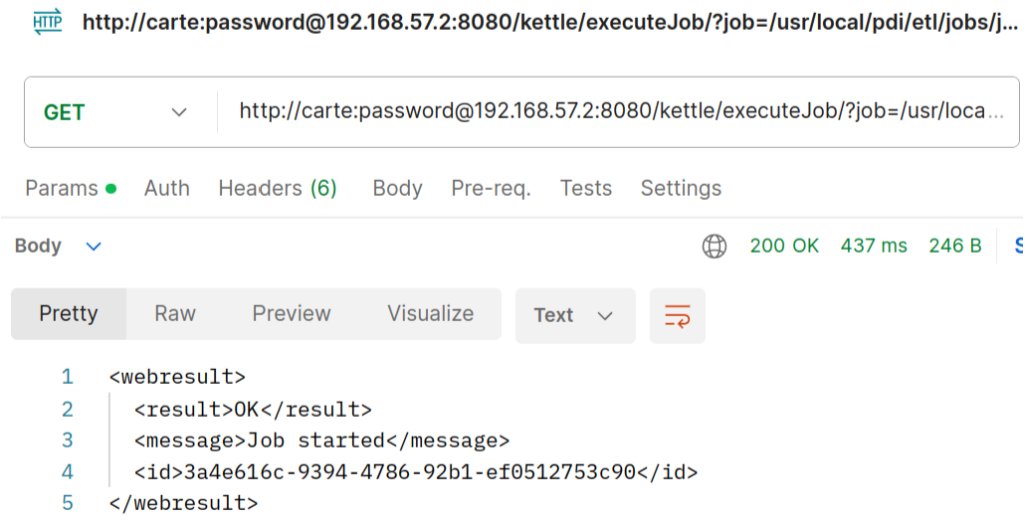

Figura 17. Requisição HTTP feita ao Carte pelo Postman: ingestão iniciada

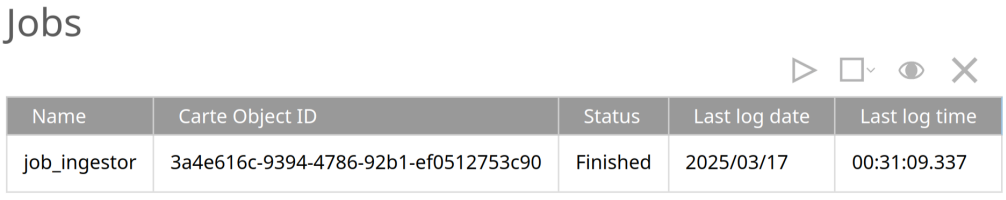

Figura 18. Andamento da ingestão: *Job* concluído com sucesso

Com a verificação do fim da ingestão, foi executada uma consulta SQL pelo Hue, que buscou os dados no Trino, que é o motor de consulta do *Data Lake* projetado. Com o retorno dos dados, foi assim confirmada a correta inserção dos dados no sistema, como

visto na Figura 19. Ficando assim evidente o sucesso de toda a operação de implantação do *Data Lake* feita com o Ansible.

	candidato_cpf	candidato_nome	candidato_cargo	coligacao_nome	coligacao_partidos
1	48392017465	João Pereira	Vereador	União pelo Futuro	PDB e PDE
2	75029382146	Maria Cardoso	Deputado	Progresso Nacional	PDA e PDG
3	19283746501	José Oliveira	Senador	Aliança Democrática	PDC e PDI
4	84710293856	Ana da Costa	Presidente	Frente Popular	PDD e PDJ
5	30519487263	Pedro Martins	Deputado	Brasil Forte	PDF e PDH
6	67839201452	Luiza Ribeiro	Vereador	União pelo Futuro	PDB e PDE
7	24938175026	Carlos Souza	Senador	Progresso Nacional	PDA e PDG
8	81350692741	Gabriela Gomes	Presidente	Aliança Democrática	PDC e PDI
9	39284710563	Felipe dos Santos	Deputado	Frente Popular	PDD e PDJ
10	57192038462	Amanda da Silva	Vereador	Brasil Forte	PDF e PDH

Figura 19. Retorno do Hue: conjunto inserido com sucesso no *Data Lake*

Todo o processo de instalação e configuração dos componentes do *Data Lake* (Hadoop, Hive, Hue, MariaDB, PDI, PSQL e Trino) foi concluído em aproximadamente 16 minutos, resultando em uma rápida e funcional implantação com apenas um comando. Este tempo foi obtido com a utilização das configurações de desempenho vistas na Figura 12.

Para a comparação do tempo de execução obtido com o tempo estimado que a implantação manual de um *Data Lake* similar levaria, foi feita uma pesquisa de opinião, seguindo a utilização de um método comum em projetos de pesquisa – as opiniões de especialistas descritas por [GARCIA 2010]. Para este propósito, foi elaborada uma pergunta: "De acordo com sua experiência, quantas horas você estima que são necessárias para o processo de implantação manual de um *Data Lake*, em um grupo de 6-10 máquinas, realizado por uma pessoa com experiência em implantações de *Data Lakes*?", que foi feita a dois especialistas escolhidos com base nas suas experiências de anos com *Data Lakes*. A Tabela 3 apresenta os resultados.

Tabela 3. Estudo com especialistas

Especialista	**Tempo de experiência com *Data Lakes***	**Resposta à pergunta**
Especialista 1	7 anos	80 horas
Especialista 2	9 anos	64 horas

Fonte: autor (2025)

Com o tempo de execução de 16 minutos obtido no teste de implantação do presente projeto, e a média de 72 horas obtida pelas estimativas dos especialistas, calculou-se que a automação reduziu o tempo de implantação em mais de 99%. Porém, é importante salientar que os artefatos Ansible requerem manutenção constante para que estejam sempre atualizados, e suas variáveis devem ser ajustadas para cada implantação nova; esses fatores não se encontram presentes no cálculo. Essa redução do tempo de implantação

de que o Ansible é capaz também fica evidente – embora em menor escala – em outros trabalhos acadêmicos estudados, como os da lista a seguir:

- A análise de [ELRADI 2023], o tempo de implantação de uma aplicação *web* foi reduzido de 45 minutos, quando utilizado um processo manual, para 8 minutos com o uso do Ansible, representando uma redução de 82,2%;
- No estudo de [KAUSHIK e GUPTA 2017], o uso do Ansible resultou no primeiro caso (em uma máquina), em uma redução do tempo de implantação de aproximadamente 4,3 horas (manual) para aproximadamente 2,4 horas (com Ansible), representando uma redução de 44,19%. Já no segundo caso (em mais de uma máquina), o tempo foi reduzido de aproximadamente 6,5 horas (manual) para aproximadamente 2,8 horas (com Ansible), ou seja, uma redução de cerca de 56,92%.

É possível que as escolhas de desempenho descritas na Subseção 3.3.3 tenham resultado na redução mais significativa encontrada no presente trabalho em relação aos dois estudos citados.

Os artefatos criados para os fins deste projeto estão disponíveis no GitHub, no seguinte endereço: `https://github.com/thomastbb/tcc-projeto`. Para a replicação do experimento apresentado neste projeto, é recomendada a instalação do Ansible (v2.18.3), *Vagrant* (v2.4.3) e *VirtualBox* (v7.1.6) em uma máquina Linux, pois foram as versões utilizadas para o mesmo.

5. Conclusão

Este trabalho descreveu a utilização do Ansible para orquestrar e automatizar o processo de implantação de um *Data Lake* em um grupo de máquinas.

Como resultados, foi encontrado que a utilização do Ansible proporcionou uma redução considerável: do nível de complexidade da implantação de um *Data Lake*; do tempo de implantação em 99%, de 72 horas estimadas para 16 minutos. O uso do Ansible garantiu consistência nas configurações em todas as máquinas, facilidade de replicação e evitou inúmeros comandos manuais e erros humanos.

Como limitações do projeto, listam-se os seguintes pontos:

- Os *clusters* distribuídos do Hadoop e Trino poderiam utilizar mais máquinas de armazenamento e processamento, para melhor representar a realidade de um sistema distribuído, mas, os recursos limitados da máquina a ser utilizada para o projeto não permitiram a criação de muitas máquinas virtuais;
- Não foi possível explicar detalhadamente o funcionamento dos artefatos do Ansible e o processo de construção dos mesmos, pois o artigo seria demasiadamente longo;
- A comparação de desempenho poderia ser mais fundamentada, por exemplo, não foi feita uma implantação manual no mesmo ambiente, para uma comparação com as estimativas dos especialistas. Além disso, o número de opiniões de especialistas obtido foi limitado.

Como lições aprendidas durante o desenvolvimento do projeto, listam-se os seguintes pontos:

- Gestão de variáveis: a facilidade com que o número de variáveis em um projeto como este cresce, evidencia a necessidade de gerenciá-las e agrupá-las;

- Ordem de execução: é importante mapear dependências entre componentes antes de estruturar os *Playbooks*;
- Organização dos *Playbooks*: o uso de estruturas como a de *Roles* é essencial para reduzir o nível de complexidade que surge de um projeto como este.

Este trabalho contribui para a área de pesquisa mostrando como uma tecnologia de automação, como o Ansible, pode ajudar a orquestrar e automatizar processos complexos, como a implantação de um *Data Lake* composto por *clusters* distribuídos e com múltiplas máquinas com diferentes funções, reduzindo consideravelmente o tempo gasto em tarefas manuais desnecessárias e simplificando a complexidade do processo.

Como trabalhos futuros, elencam-se as seguintes sugestões para a complementação do presente trabalho ou o desenvolvimento de trabalhos similares:

- Os artefatos do Ansible podem ser expandidos para, por exemplo: (I), dar suporte a outros tipos de sistemas operacionais além do Ubuntu, como o *Alpine* e o CentOS; (II), dar suporte a mais versões de cada ferramenta utilizada; (III), dar suporte a implantações de *clusters* mais complexos que utilizam mais de um *NameNode* do Hadoop ou *Coordinator* do Trino;
- Adaptar e testar os artefatos do Ansible em uma implantação de um *Data Lake* real e avaliar os resultados;
- Comparar o desempenho do processo automatizado com um processo manual, em uma análise mais profunda.

Referências

ANSIBLE (2025). Ansible community documentation. `https://docs.ansible.com/`. Última atualização em 25 de fevereiro de 2025. Acesso em: 8 mar. 2025.

AZZABI, S., ALFUGHI, Z., e OUDA, A. (2024). Data lakes: A survey of concepts and architectures. *Computers*, 13:183.

CHIARI, M., PASCALIS, M., e PRADELLA, M. (2022). Static analysis of infrastructure as code: a survey.

COULOURIS, G., DOLLIMORE, J., KINDBERG, T., e BLAIR, G. (2013). *Sistemas Distribuídos: Conceitos e Projeto*. BOOKMAN, GRUPO A, 5ª edition. E-book.

COUTO, J., BORGES, O., RUIZ, D., MARCZAK, S., e PRIKLADNICKIP, R. (2019). A mapping study about data lakes: An improved definition and possible architectures. pages 453–458.

DILMEGANI, C. (2025). Orchestration vs. automation: Which one to choose in 2025? `https://research.aimultiple.com/orchestration-vs-automation`. Publicado em 4 de fevereiro de 2025. Acesso em: 8 mar. 2025.

Dixon, J. (2010). Blog post: Pentaho, hadoop, and data lakes. Acesso em: 22 de mar. de 2025.

ELRADI, M. D. (2023). Ansible: A reliable tool for automation. *Electrical and Computer Engineering Studies*, 2(1, No. 4).

GARCIA, V. C. (2010). *RiSE reference model for software reuse adoption in brazilian companies*. PhD thesis, Universidade Federal de Pernambuco, Pernambuco, Brasil. Acesso em 24 de mar. 2025.

KAUSHIK, S. e GUPTA, S. (2017). Implementation of open stack through ansible. 6(5). Retrieval Number ES036066517/170BE/ESP.

NAMBIAR, A. e MUNDRA, D. (2022). An overview of data warehouse and data lake in modern enterprise data management. *Big Data and Cognitive Computing*, 6(4):132.

RAHMAN, A., MAHDAVI-HEZAVEH, R., e WILLIAMS, L. (2018). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108.

RED HAT (2022). What is infrastructure as code (iac)? Acessado em 13 de abril de 2025.

RICHMAN, J. (2023). What are distributed architectures: 4 types key components. Última atualização em 17 de outrubro de 2024. Acesso em: 22 de mar. de 2025.

SINGH, R. e PURWAR, R. K. (2019). Cloud automation with configuration management using chef tool. *International Journal of Engineering Research & Technology*.

TERRIZZANO, I. G., SCHWARZ, P. M., ROTH, M., e COLINO, J. E. (2015). Data wrangling: The challenging yourney from the wild to the lake. In *Conference on Innovative Data Systems Research*.

YARLAGADDA, R. T. (2021). Devops and its practices. *SSRN Electronic Journal*, 9:2320–2882.