Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster en Ciencia de Datos

Master in Data Sience

Trabajo Fin de Máster
Master Thesis

Procesos de Datos: Integración de Herramientas de Código Abierto y Microservicios

Data Pipelines: Integrating Open Source Tools and Microservices

Victor Berrozpe Maldonado

Madrid, October/ Octubre, 2024

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid.

This Master Thesis has been deposited in ETSI Informáticos de la Universidad Politécnica de Madrid.

Trabajo Fin de Máster
Master Thesis
Máster en Ciencia de Datos
Master in Data Science

Título: Procesos de Datos: Integración de Herramientas de Código Abierto y Microservicios

Title: Data Pipelines: Integrating Open Source Tools and Microservices

October/ Octubre, 2024

Autor / Author: Victor Berrozpe Maldonado

Tutor
Supervisor:
Lutz Oettershagen

School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology

Co-Tutor UPM
Co-supervisor:
Ainhoa Azqueta

Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

Esta tesis explora el diseño e implementación de pipelines de datos modernos en la plataforma Google Cloud, con un enfoque en la identificación de componentes esenciales, metodologías y mejores prácticas para mejorar su eficiencia, confiabilidad y escalabilidad. Realizado en colaboración con Astrafy, este trabajo de investigación proporciona conocimientos críticos para la construcción de pipelines de datos automatizados. El estudio aborda específicamente las complejidades de integrar diversas herramientas de código abierto y microservicios, con el objetivo de simplificar el proceso de desarrollo mediante la propuesta de una metodología estructurada para la selección y combinación de estas tecnologías. Una contribución clave de este trabajo es la creación de un sistema de despliegue automatizado de un solo clic, que optimiza la configuración y gestión de pipelines de datos para el procesamiento de datos de facturación en Google Cloud. El sistema es evaluado a través de una serie de experimentos que comparan diversas tecnologías basadas en métricas de rendimiento como el costo, el tiempo de despliegue y la facilidad de uso. Los resultados ofrecen pautas valiosas y modelos replicables para organizaciones que buscan implementar pipelines de datos escalables y confiables en plataformas en la nube.

Abstract

This thesis explores the design and implementation of modern data pipelines within the Google Cloud Platform, with a focus on identifying essential components, methodologies, and best practices to enhance their efficiency, reliability, and scalability. Conducted in collaboration with Astrafy, this research provides critical insights into the construction of automated data pipelines. The study specifically addresses the complexities of integrating various open source tools and microservices, aiming to simplify the development process by proposing a structured methodology for selecting and combining these technologies. A key contribution of this work is the creation of an automated one-click deployment system, which streamlines the setup and management of data pipelines for processing Google Cloud billing data. The system is evaluated through a series of experiments, comparing various technologies based on performance metrics such as cost, deployment time, and ease of use. The results offer valuable guidelines and replicable models for organizations seeking to implement scalable and reliable data pipelines on cloud platforms.

Keywords

Data Pipelines, Google Cloud Platform, Microservices Integration, Open-Source Tools, Infrastructure as code

Acknowledgements

I would like to express my sincere gratitude to my supervisor at KTH, Lutz Oettershagen, and my examiner, Aristides Gionis, for their guidance and feedback at KTH, and Ainhoa Azqueta at UPM. I am also deeply thankful to Alejandro de la Cruz, my supervisor at Astrafy, for his practical insights and support.

A special thanks to the entire team at Astrafy for providing a collaborative environment, to my family and María Zuil for their support when it was most needed.

Thank you all for your contributions and encouragement.

Contents

List of Figures

List of Tables

Introduction

1.1 Background

A data pipeline is a series of processes designed for the efficient movement, transformation, and management of data from its source to its destination, such as a data warehouse. In the context of data pipelines, the technical architecture refers to the underlying framework and components that support the entire lifecycle of data as it moves through the pipeline. The key components of this architecture typically include data ingestion tools, data storage systems, data transformation and analysis engines, and often, an orchestration layer, see Figure 5.2.

There are many types of data pipelines, each serving different purposes such as data collection, real-time analytics, batch processing, etc. In this thesis, we will focus on pipelines designed for data ingestion, transformation, and communication within a cloud-based infrastructure. Specifically, we will explore pipelines that handle organizational cost or billing data, transforming it into actionable insights delivered through BI dashboards and automated notifications.

The current problem with data pipelines lies in their complexity and the challenge of integrating multiple tools and methodologies effectively. This project aims to address the complexity and challenges of integrating various open-source tools and microservices available for each step of the pipeline (Ingestion, transformation etc).

The project aligns with the core expertise and objectives of Astrafy, a company with a primary focus on Data Engineering through the use of a modern data stack. Astrafy's commitment to staying at the forefront of technological innovation, and specialization on developing custom data pipelines tailored to meet specific organizational needs makes this research particularly relevant.

1.2 Problem Statement

There exist a large amount of open source tools and methodologies that can be used for each step of a pipeline.

The goal of this thesis is to give is to provide a structured methodology for

automatically choosing and combining such tools to obtain a data pipeline for processing Google Cloud billing data. Therefore, our main question is: How can we automatically choose the right combination of tools (from a given set of possibilities) based on a given set of quality metrics to obtain and deploy data pipelines?

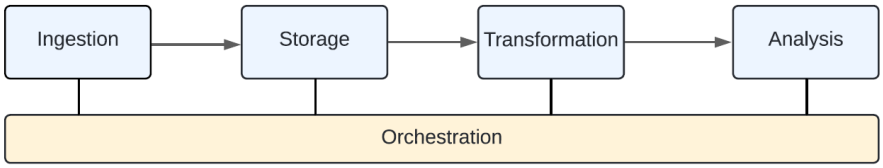

Figure 1.1: Different stages of a data pipeline.

1.3 Objectives

1. Identify and choose metrics to use in order to evaluate pipeline performance, this performance is related mainly to how much it costs compared to how hard it is to set up.
2. Identify and document the key components and methodologies for integrating open source tools and microservices to design and implement data pipelines.
3. Automate a one click deployment for data pipelines of the type mentioned in the research question.
4. Build and compare data pipelines on GCP that ingests, transforms, and communicates data, providing detailed billing information via Slack messages and BI dashboards while establishing a set of best practices.

1.4 Significance of the Study

This thesis will provide valuable guidelines on integrating microservices to build robust data pipelines, offering a framework for using and integrating each tool efficiently. It will present concrete examples of fully integrated, automatically deployed data pipelines that deliver actionable insights on organizational costs, serving as replicable models for other organizations. Additionally, the thesis will address common integration challenges, providing strategies to overcome these obstacles and ensure efficient, scalable, and reliable data pipelines on Google Cloud Platform.

1.5 Thesis Structure

This thesis is organized into several chapters, each covering different aspects of the research.

- **Chapter 1: Introduction** - This chapter provides an overview of the research topic, including the background, problem statement, research questions, objectives, significance of the study, and an outline of the thesis structure.
- **Chapter 2: Literature Review** - This chapter reviews the existing literature on data pipeline architecture, data ingestion, data storage, orchestration, analysis, and related technologies. It identifies gaps in the current knowledge that this research aims to fill.
- **Chapter 3: Methodology** - This chapter details the research design, data collection methods, data analysis techniques, ethical considerations, and limitations. It explains the approach used to address the research questions.
- **Chapter 4: Case Study Analysis** - This chapter presents detailed analyses of selected case studies, describing the data pipelines used by Astrafy and other organizations. It highlights the key components, methodologies, and best practices identified.
- **Chapter 5: Experimental Design and Results** - This chapter describes the experimental setup, including the design and implementation of prototype data pipelines. It presents the results of these experiments and discusses their implications.
- **Chapter 6: Discussion** - This chapter synthesizes the findings from the literature review, case studies, and experiments. It discusses the implications for theory and practice, outlines best practices for data pipeline design, and provides practical recommendations.
- **Chapter 7: Conclusion** - This chapter summarizes the key findings, highlights the contributions of the research, suggests areas for future work, and concludes the thesis.
- **References** - This section lists all the sources cited in the thesis, formatted according to the chosen citation style.

Literature Review and existing research

2.1 Introduction

The purpose of this literature review is to provide a comprehensive overview of existing research and developments in the field of data pipelines, with a particular focus on their integration within Google Cloud Platform (GCP). This review will explore the key components, methodologies, and best practices that contribute to the design and implementation of efficient and scalable data pipelines.

2.2 Theoretical Background

Present the key theories and concepts related to data pipelines. This might include definitions, frameworks, and foundational principles.

2.2.1 Data pipelines

Data pipelines consist of sequences of activities that process data, where the output from one component serves as the input to the next, enabling a seamless, automated flow of data from its source to its destination [1]. There are many types of data pipelines, each serving different purposes such as data collection, real-time analytics, batch processing, etc [2].

These pipelines are necessary in today's data-driven contexts, where software as a service (SaaS) and cloud computing have expanded the volume of data sources and data in general. Behind every dashboard and predictive model, a complex system of data pipelines operates unseen, driving the transformation and flow of data that makes these insights possible [3].

2.2.2 Microservices Architecture

Microservices architecture is an architectural style that structures an application as a collection of small, independent services, each responsible for a specific function within the larger system [4]. In the context of data pipelines, this architecture allows for the modularization of different stages, such as data ingestion, transformation, storage, and analysis, into distinct services that can be developed, deployed, and scaled independently. This modularity enhances the flexibility and scalability of data pipelines, allowing organizations to choose and integrate the best tools and technologies for each stage of the pipeline. Given

that there is freedom to choose the technology for each stage, open source tools are often used as they provide cheap and customizable solutions that can be tailored to meet the specific needs of each service [5], while also benefiting from continuous updates and innovations contributed by a broad developer base.

2.2.3 Metrics in the context of pipeline architecture

In the context of data pipeline architecture, metrics are used for assessing the overall performance and efficiency of the system. They offer a clear view of the pipeline's operational health, helping to ensure reliability, scalability, and optimal resource utilization. By regularly monitoring and refining these metrics, organizations can maintain and improve the effectiveness of their data processing workflows.

There exist a large amount of metrics that can be used to evaluate how good a pipeline is. Most of them can be easily measured such as cost and resource utilization, but a good pipeline evaluation also considers non objective and hard to measure metrics such as the complexity and ease of use [6].

2.2.4 Terraform

Terraform is an open-source Infrastructure as Code tool developed by HashiCorp that enables the automation of infrastructure provisioning. By using declarative configuration files, Terraform allows users to define and manage infrastructure in a consistent, repeatable manner. In the context of data pipelines, Terraform is often used to automate the setup and management of cloud resources, such as servers, databases, and storage, which are integral parts of the pipeline [7]. Its modularity and provider ecosystem allow it to interact with various cloud platforms and services, ensuring infrastructure scalability and efficient resource management throughout the data pipeline lifecycle.

2.2.5 ArgoCD

ArgoCD is a declarative, GitOps continuous delivery tool specifically designed for Kubernetes. It allows organizations to automate the deployment and management of applications in Kubernetes clusters by synchronizing the desired application state, as defined in Git repositories, with the actual state of the Kubernetes cluster [8]. In the context of data pipelines, ArgoCD is often used to ensure that infrastructure and pipeline configurations remain consistent across environments. By leveraging GitOps principles, ArgoCD enhances version control, automation, and reproducibility in the deployment and management of pipeline components.

2.2.6 Helm Charts

Helm is a Kubernetes package manager that simplifies the deployment and management of applications in Kubernetes clusters by using templates called Helm Charts. These charts define, install, and upgrade even the most complex Kubernetes applications, encapsulating resources such as services, deployments, and

configurations into reusable packages [9]. For data pipelines, Helm Charts offer a convenient way to manage complex deployments by bundling multiple services and their dependencies. This streamlines the process of deploying and scaling pipeline components while ensuring that all necessary resources are provisioned in a consistent and reproducible manner.

2.3 Pipeline Deployment Automation

Despite the advancements in CI/CD automation and data pipeline deployment within cloud environments, several key aspects remain underexplored in current research. One of the most significant gaps is the automation of infrastructure provisioning in conjunction with CI/CD processes, particularly through the use of Infrastructure-as-Code (IaC) tools such as Terraform [10]–[12]. While existing studies address various elements of pipeline automation, they often focus on the deployment of applications and fail to fully integrate the automated management of the underlying infrastructure. The automation of both layers application and infrastructure through a cloud, one-click deployment solution which would reduce manual intervention, has not been thoroughly examined.

Additionally, another area that lacks sufficient attention is the choice of metrics used to evaluate pipeline performance. Current research rarely discusses in depth which metrics should be applied to assess the effectiveness of a pipeline, making it challenging to determine the quality and efficiency of different pipeline setups. Metrics such as cost, scalability, resource utilization, and ease of use are often mentioned, but there is little consensus or structured approach in the literature on how these should be used systematically to compare and optimize pipelines. This research aims to address this gap by proposing a set of metrics tailored specifically for evaluating modern cloud pipelines.

2.4 Microservice Selection for Data Pipelines

Another gap in literature concerns the evaluation and selection of microservices for different stages of a data pipeline. While microservices architectures are widely acknowledged for their flexibility and scalability, current research does not provide a structured framework for determining which microservice is best suited for each stage of a pipeline. This issue is particularly pressing given the large number of available open source tools and services that can be integrated into pipeline architectures. As noted in [13], the abundance of microservices complicates the decision making process, with little guidance on how to compare these technologies based on key metrics.

The existing literature also does not the practical challenge of integrating these microservices into fully automated pipeline deployments. Many studies focus on isolated components or technologies, rather than offering a comprehensive solution that evaluates and selects microservices in the context of end-to-end pipeline automation. This thesis fills this gap by providing a comparative analysis of microservices for each stage of the pipeline—ingestion, transformation,

orchestration, and storage based on a defined set of operational metrics.

Methodology

3.1 Research method

In this section we will discuss literature review, case studies, and developing custom data pipelines using microservices. By reviewing existing literature and case studies from Astrafy, the study identifies best practices and open source tools for pipeline architecture. Multiple data pipelines are built and compared using metrics to determine the best tool combinations. Given that multiple data pipelines will be compared, this thesis also focuses on developing an automatic pipeline deployment system which will be used to ease the creation and comparison of the multiple pipelines. The main focus is to automate a one click deployment process for pipelines handling billing data, providing insights via Slack messages and BI dashboards. This aims to establish a methodology for choosing and integrating tools to build efficient, scalable, and reliable data pipelines.

3.2 Ethics and sustainability

The project does not raise significant ethical or sustainability concerns as it primarily focuses on technical aspects of data pipeline development. However, ethical considerations regarding data privacy and security will be taken into account during the collection and handling of data.

3.3 Limitations

- We will focus only on data pipelines for processing billing data with the goal to provide a dashboard.
- We will focus on a subset of possible metrics to evaluate our data pipelines for the use case provided by Astrafy.

3.4 Risks

Potential risks and solutions associated with this project include:

- Technical challenges in designing and implementing experimental data pipelines.

Case Study Analysis

4.1 Introduction

This section delves into the analysis of data pipelines implemented at Astrafy, a company specializing in data engineering with a focus on designing and deploying data pipelines within the Google Cloud Platform. Astrafy's expertise in modern data engineering practices makes it a compelling subject for this study, providing real world insights into the challenges and best practices associated with building and maintaining complex data infrastructures.

4.2 Description of Case Studies

Astrafy employs a variety of data pipelines designed to handle different aspects of data management, from ingestion to transformation, storage, and analysis. While the company uses numerous pipelines, this analysis focuses on a select few that exemplify the core principles of scalability and automation.

1. **GKE Cluster:**
   The principal component of Astrafy's data infrastructure is a Google Kubernetes Engine cluster, which serves as the primary platform for running applications and managing workloads. The GKE cluster provides a managed Kubernetes service that simplifies the operational aspects of running a cluster, such as scaling, updates, and maintenance, allowing Astrafy's engineers to focus on the development and optimization of data pipelines rather than the intricacies of cluster management.

2. **ArgoCD for Continuous Delivery:**
   ArgoCD is used in the continuous delivery and deployment processes at Astrafy. As an open source GitOps tool, ArgoCD automates the deployment of applications and infrastructure components by synchronizing Kubernetes resources with configurations stored in Git repositories. This approach ensures that the state of the infrastructure is always consistent with the desired configurations defined in code.

3. **Airflow for Workflow Orchestration:**
   Apache Airflow is used to orchestrate the various tasks involved in data processing. This includes scheduling and managing complex workflows that involve multiple dependencies across different services. Airflow's flexibility

allows it to integrate itself with other components in the pipeline, ensuring that data flows smoothly from ingestion to final analysis. The use of Airflow also enables Astrafy to handle both batch and real time processing workloads effectively.

4. **Istio Service Mesh:**
   To manage the communication between microservices within the GKE cluster, Astrafy utilizes Istio, a powerful service mesh technology. Istio provides enhanced security, observability, and traffic management, which are critical in a distributed microservices architecture. It also supports secure service-to-service communication, load balancing, and sophisticated routing rules, which allow for access through an external endpoint such as a url to the applications it hosts.

5. **Cloud SQL for Persistent Storage:**
   Cloud SQL is deployed as the primary relational database service, supporting the persistent storage needs of various applications within the data pipeline. By leveraging Cloud SQL, Astrafy benefits from a managed database solution that integrates seamlessly with other GCP services, providing reliable and scalable storage with minimal management overhead.

Each of these components plays a critical role in ensuring that Astrafy's data pipelines are not only functional but also scalable and maintainable. The integration of these tools and technologies into a cohesive system exemplifies best practices in modern data engineering.

4.3 Analysis

The analysis of Astrafy's data pipelines reveals several key insights into the design and operation of modern data infrastructure. By dissecting the roles and interactions of the various components, we can better understand the strategies that contribute to the success of these pipelines.

1. **GKE Cluster - Foundation of Scalability:**
   The decision to use GKE as the backbone of Astrafy's infrastructure highlights the importance of scalability in modern data pipelines. GKE's managed services allow Astrafy to dynamically scale their applications and resources in response to varying workloads, ensuring that performance remains consistent even under heavy demand. This scalability is crucial for maintaining the reliability of data pipelines as data volumes grow.

2. **ArgoCD and GitOps - Ensuring Consistency and Reliability:**
   ArgoCD's integration into the deployment pipeline demonstrates the effectiveness of GitOps practices in maintaining the consistency and reliability of infrastructure. By automating deployments and continuously synchronizing the live environment with the configurations stored in Git, ArgoCD minimizes the risks associated with manual deployments and configuration drift. This automation not only improves operational efficiency but also enhances the overall stability of the data pipelines.

3. **Airflow - Orchestrating Complex Data Workflows:**
   Airflow's role as the orchestration engine within Astrafy's pipelines is a testament to its flexibility and power in managing complex data workflows. Whether dealing with batch processing or real-time data streams, Airflow provides the necessary tools to schedule, monitor, and manage the tasks that make up a data pipeline. Its integration with other services, such as Cloud SQL and GKE, ensures that data flows are handled efficiently, with tasks being executed in the right order and at the right time.
4. **Istio - Enhancing Security and Observability:**
   Istio's deployment as a service mesh adds a critical layer of security and observability to Astrafy's microservices architecture. By managing traffic within the cluster, Istio ensures that only authorized services can communicate with each other, reducing the attack surface and improving overall security. Additionally, Istio's observability features provide deep insights into the performance and behavior of services, enabling proactive monitoring and troubleshooting of the data pipelines.

4.4 Findings

The case study analysis of Astrafy's data pipelines underscores the effectiveness of a Kubernetes-based architecture for managing complex data workflows. The integration of open-source tools like ArgoCD, Airflow, and Istio within this architecture demonstrates the power of automation and microservices in achieving scalability, reliability, and security.

Key findings:

Common Architecture: The use of a Kubernetes (GKE) cluster as the core platform allows for the seamless integration of various microservices and tools, creating a flexible and scalable environment for data processing. Open Source Technologies: Tools like ArgoCD, Airflow, and Istio are instrumental in automating deployments, managing workflows, and securing communications within the pipeline. Their open-source nature provides flexibility and customization, which are essential for meeting the specific needs of different clients. Automation reduces complexity and ensuring consistent deployments. This approach not only accelerates product deployment but also minimizes the risk of errors, making it a best practice in modern data pipeline management. These findings provide a foundation for the subsequent sections of this thesis, where the selected technologies and methodologies will be further evaluated and compared. The insights gained from Astrafy's pipelines will inform the development of best practices for building and managing data pipelines in other contexts.

Experimental Design and Results

5.1 Introduction

The purpose of this experimental design is to evaluate various open source technologies across different stages of a data pipeline. This evaluation is essential to address the research question: how can we automatically choose and integrate the right combination of tools to construct effective and scalable data pipelines? The experiments are divided into two distinct phases, each focusing on an aspect of the pipeline development process.

Phase 1: One Click Deployment Development The first phase focuses on creating a one click deployment system, an innovation developed at Astrafy, which allows for the rapid and seamless creation of data pipelines. Given the need to compare multiple pipelines, this tool is designed to minimize the complexity and time involved in building and deploying various pipeline configurations. The one click deployment system integrates multiple common components, such as GKE clusters, ArgoCD, and Istio, ensuring that pipelines can be set up with minimal manual intervention. This phase is foundational, as it establishes the infrastructure needed to quickly iterate on and test different pipeline setups.

Phase 2: Building and Comparing Data Pipelines Once the one click deployment system is in place, the second phase involves using this tool to create multiple pipelines. These pipelines will ingest, transform, and communicate data, particularly focusing on providing detailed billing information via Slack messages and BI dashboards. This phase aims to evaluate the selected open source technologies across various metrics, such as cost, scalability, complexity amongst others and to identify best practices for building modern data pipelines on GCP.

5.2 Experimental Setup

The experimental setup in this thesis is built to mirror the kind of environments uses by businesses, where data pipelines are deployed on cloud platforms like Google Cloud Platform to ensure they are secure, scalable, and reliable. While it's possible to run these technologies on a personal computer, companies dealing with large amounts of data need the power and flexibility that cloud infrastructure provides. This setup is intended to handle the real demands of data

processing. Although we're focusing on cloud solutions, the results are still relevant for self-hosted systems, where the cost isn't a factor.

5.2.1 Phase 1: One Click Deployment Development

The primary objective of Phase 1 is to develop a robust, automated system capable of deploying complete data pipelines which contain the stages mentioned in the problem statement 5.2, with minimal manual intervention. This system, is a solution designed alongside Astrafy to address the complexities associated with setting up data pipelines that are both scalable and easily replicable. This phase is crucial, as it lays the foundation for the comparative analysis of different pipeline configurations.

In modern data engineering, the process of building a data pipeline often involves a series of intricate steps, including configuring networking resources, deploying and managing Kubernetes clusters, setting up orchestration tools, and integrating various data processing components. Each of these steps can be time consuming and prone to errors, especially when performed manually. The one click deployment system is specifically designed to streamline this process by automating the deployment of the entire data stack.

The system is built using Infrastructure as Code (IaC) principles, which allow infrastructure to be defined and managed through code, rather than manual setup. By leveraging tools like Terraform and Kubernetes, the system can automate the provisioning of cloud resources, the deployment of applications, and the management of ongoing operations. This approach not only reduces the risk of configuration errors but also ensures that each deployment is consistent and repeatable.

Key Components of the One-Click Deployment System

Terraform for Infrastructure as Code (IaC):
Terraform is utilized as the primary tool for defining and managing the infrastructure required for the data pipelines. This tool is used because of its extensive modularity, which allows for reusable components. The Terraform scripts are structured to provision the following key resources on Google Cloud Platform:

- **Networking Components:** The scripts first establish the networking framework, including Virtual Private Cloud (VPC) networks, subnets, and firewall rules. These components are used for ensuring secure and efficient communication between the various services within the pipeline.
- **Google Kubernetes Engine (GKE) Cluster:** Terraform provisions a GKE cluster, which serves as the base component for hosting and orchestrating the pipeline's microservices. The GKE cluster is configured to support auto-scaling, enabling it to dynamically adjust resources based on workload demands, which will be affected by the technologies we decided to host on the kubernetes cluster.

- **Persistent Storage Solutions:** Terraform also handles the deployment of persistent storage options, such as Cloud SQL instances, which are used for managing the data required by the pipeline's components.

GKE Cluster - Foundation for Scalability and Orchestration:
The GKE cluster is the central component of the one click deployment system, providing a managed Kubernetes environment where the various microservices and tools are deployed and managed. The cluster is set up to ensure high availability and resilience, with automatic scaling features enabled to handle varying workloads. This scalability is necessary for the experimental phase, as we will use different technologies for each stage of the pipeline, each using a different amount of resources.

Additionally, the GKE cluster is configured to integrate with other GCP services, such as BigQuery for data storage and processing, and Cloud Monitoring for observing the health and performance of the deployed services. This integration is vital for maintaining the operational stability of the data pipelines and for collecting performance metrics that will be analyzed in the experimental phase.

ArgoCD - GitOps for Continuous Delivery and Deployment:
ArgoCD is employed as the continuous delivery tool within the one-click deployment system. By adopting a GitOps approach, where the entire infrastructure and application configurations are stored in Git repositories, ArgoCD automates the process of deploying and synchronizing these configurations with the running environment.

This automation ensures that any changes to the pipeline configurations are automatically applied, reducing the need for manual intervention and minimizing the risk of human error. ArgoCD also provides a rollback mechanism, allowing for quick recovery in case of deployment issues, which is crucial for maintaining the reliability of the data pipelines during testing.

Airflow for Workflow Orchestration:
Apache Airflow is integrated into the one-click deployment system to manage the orchestration of data workflows. Airflow is responsible for scheduling and executing the tasks within the data pipeline, such as data ingestion, transformation, and loading into storage. The integration of Airflow within the GKE environment allows for efficient resource utilization and ensures that the data processing tasks are performed in a timely and coordinated manner.

The Airflow deployment is configured with Kubernetes Executors, enabling it to scale out tasks across the cluster dynamically. This setup is particularly beneficial for handling large-scale data processing workloads, which are expected in the experimental phase.

Istio - Service Mesh for Security and Observability:
Istio is deployed as a service mesh to manage the internal communication between microservices within the GKE cluster. Istio provides several critical fea-

tures, including secure service-to-service communication, traffic management, and observability. These features are essential for ensuring that the microservices can communicate reliably and securely, even under heavy load.

Istio's observability features, such as distributed tracing and telemetry collection, are particularly valuable for monitoring the performance of the data pipelines. By collecting detailed metrics on service interactions, Istio enables a deeper analysis of the pipeline's performance, which will inform the selection of the most effective technologies in the later stages of the experiment.

Automated Resource Management:
The one-click deployment system also incorporates automated resource management features to optimize the use of cloud resources. For example, it includes scripts to automatically shut down idle resources during periods of low activity, which helps to minimize costs. Additionally, the system is designed to monitor resource utilization in real-time, adjusting the allocation of compute and storage resources based on the current demand.

Implementation Process of the One-Click Deployment System

The development of the one-click deployment system followed a structured approach, starting with the identification of the necessary components and the design of the deployment architecture. The implementation process involved the following steps:

- **Designing the Deployment Architecture:**
  The first step involved designing a comprehensive architecture that integrated all the necessary components, such as GKE, ArgoCD, Airflow, and Istio. This design phase focused on ensuring that the components could work together seamlessly, with clear connections between the different technologies.

- **Writing Terraform Scripts:**
  The next step was to write the Terraform scripts that would automate the provisioning of the infrastructure. This involved defining each resource in code, specifying dependencies, and setting up configurations that allowed for dynamic scaling and high availability. The scripts were tested in a sandbox environment to ensure that they could deploy the infrastructure without issues.

- **Configuring ArgoCD for GitOps:**
  ArgoCD was configured to work with the Git repositories containing the pipeline configurations. This setup included defining the synchronization policies, setting up access controls, and testing the rollback mechanisms. The goal was to ensure that any changes to the configurations could be automatically deployed to the running environment with minimal delay.

- **Integrating Airflow and Istio:**
  Finally, Airflow and Istio were integrated into the system. Airflow was con-

figured to use Kubernetes Executors, allowing it to distribute tasks across the GKE cluster. Istio was set up to manage service communication, with detailed monitoring enabled to capture performance metrics.

The following diagram explains how each main component is connected to each other.

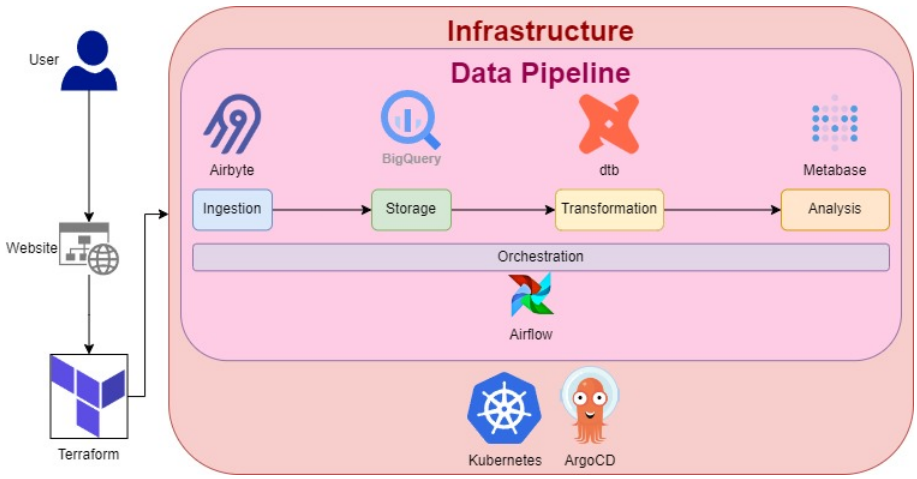

Figure 5.1: One click deployment diagram

5.2.2 Phase 2: Microservice technologies comparison and testing

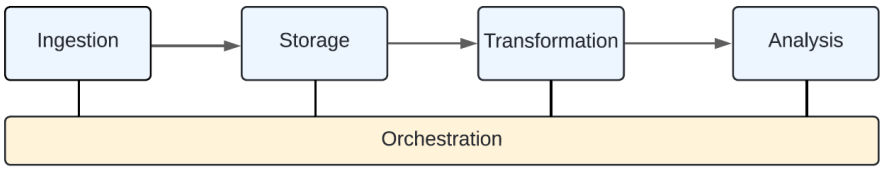

Figure 5.2: Different stages of a data pipeline.

Due to the vast number of open source technologies available for each step of the data pipeline, we are unable to test all of them. Therefore, we have selected a few key technologies for each stage to evaluate. The selected technologies can be found in the table below.

Category	Technology
Ingestion	• Airbyte • Dlt
Transformation	• dbt • Dataform
Storage	• BigQuery • Cloud Storage with BigLake
Analysis	• Metabase • Lightdash
Orchestration	• Airflow • Dagster

Table 5.1: Categories and Technologies

The choice of these technologies comes from a combination of factors: some were selected based on their frequent citation and analysis in academic papers, reflecting their recognized effectiveness in the field, while others were chosen due to their use and proven success in the case study conducted at Astrafy.

We will use the following metrics to evaluate the effectiveness of each of these technologies:

Metric	Description
Cost	The financial expense associated with using the technology.
Deployment Time	The time required to set up and deploy the technology.
Code size and manual steps	Amount of components/lines of code necessary to establish the technology, as well as the amount of manual steps needed outside of the one click deployment

Table 5.2: Metrics for Evaluating Technologies

First, we will evaluate each component individually. Then, we will assess how well they integrate with the other technologies in the pipeline.

5.2.3 Webpage to streamline pipeline creation

Knowing the technologies that well need to be integrated with one another, a small webpage was created in order to automatically generate the terraform code to create the pipeline infrastructure in a "one click deployment". This is not the primary goal of the thesis, but it will help streamline the creation of pipeline.

It works as follows:

First, the user is presented with the different stages of a pipeline, where it is possible to select which technologies you want to use for each stage.

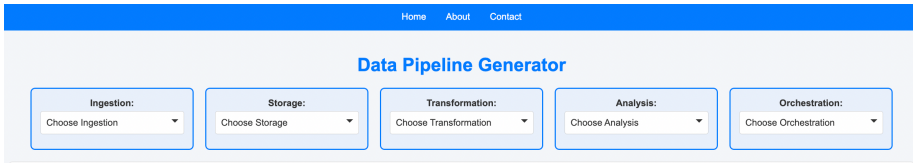

Figure 5.3: Default technology selector.

Although the pipelines are usually built with one of these technologies per stage, the user can still select multiple technologies per stage, like it is shown in the following figure.

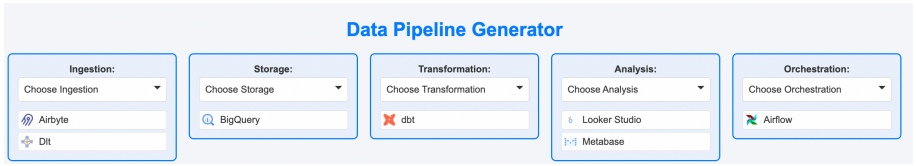

Figure 5.4: Multiple technology selection.

Once the technologies are selected, users can customize the amount of resources allocated to each one. However, it is advisable to use the default settings, as reducing resources may lead to crashes due to insufficient capacity, while increasing them could result in unnecessary costs for unused resources. That said, if a particular technology needs to run continuously without interruptions, users have the option to allocate additional resources to ensure its stability.

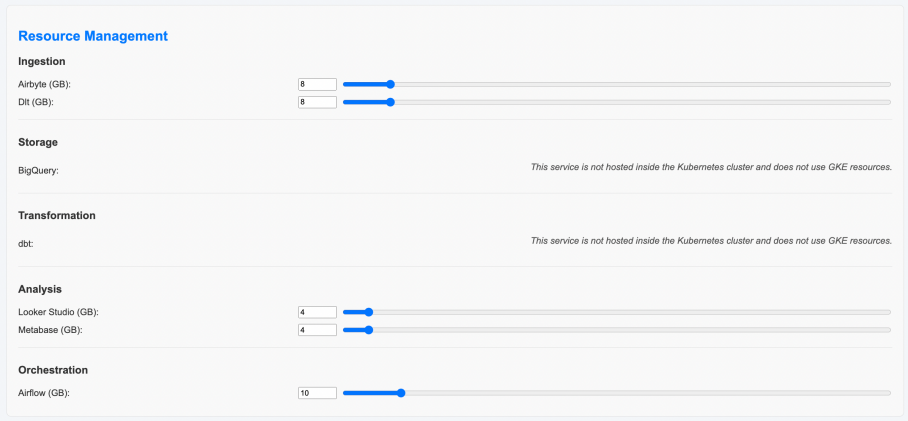

Figure 5.5: Example resource customization.

Depending on the selected amount of resources, the approximated daily costs will be calculated. These expenses are based on the baseline cost of having a running cluster in GCP, and the sum of the amount of all dedicated resources for each one of the technologies selected.

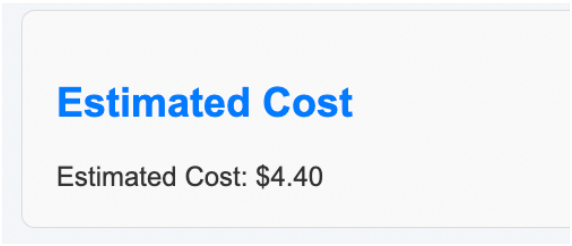

Figure 5.6: Daily cost estimation for the selected resources.

Finally, the user can configure a set of variables to integrate the pipeline into an existing GCP project. While the remaining variables are pre-configured and do not require modification, they can still be adjusted within the code if desired.

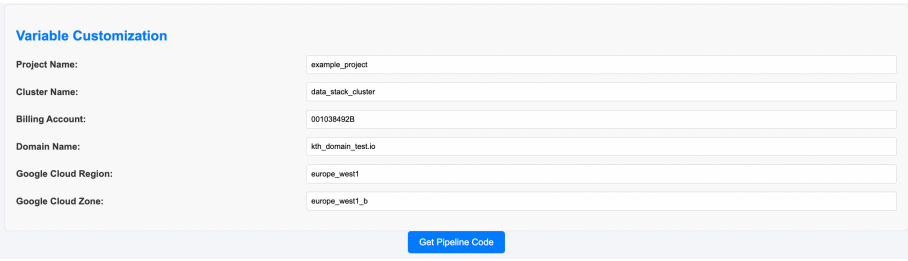

Figure 5.7: Variable customization.

After all the variable configurations are set, the user can click the "Get pipeline code" button to obtain a set of Terraform scripts that will automatically create the pipeline infrastructure. Detailed instructions on how to use these scripts,

along with any necessary manual steps for building the pipeline, are provided in the annex.

5.3 Technical challenges and solutions

During the implementation of the data pipeline architecture at Astrafy, some technical challenges were encountered. This section outlines the key challenges faced and the technical solutions used to solve them.

- **Scalability and Resource Management:** One of the primary challenges was ensuring that the infrastructure could scale based on changing workloads. Given the unpredictability of resource demand in large data pipelines, manual scaling is not an efficient solution. To address this, the system used *Google Kubernetes Engine*, which offers auto-scaling features. The infrastructure was designed to automatically allocate and deallocate resources as needed, ensuring that the system scaled in response to the workload without manual intervention.
- **Automation of Infrastructure:** Automating the provisioning of infrastructure presented a significant challenge. Initially, setting up environments manually introduced risks of configuration drift and deployment errors. This challenge was addressed by employing *Infrastructure as Code* principles through *Terraform*, enabling the full automation of infrastructure provisioning. The integration of *ArgoCD* with the GitOps approach ensured that any changes to the infrastructure or applications were automatically synchronized and deployed.
- **Orchestration of Complex Workflows:** Managing the orchestration of multiple microservices with interdependent tasks was a technical hurdle, particularly in ensuring the efficient execution of these tasks in the correct sequence. *Apache Airflow* and *Dagster* were used to solve this issue. They were configured to orchestrate tasks dynamically, distributing workloads across the GKE cluster.
- **Public Access with Istio, Load Balancer, and Domain Assignment:** A key challenge involved enabling public access to the pipeline's web interface through a custom domain, while ensuring secure and reliable routing of traffic. To achieve this, *Istio Service Mesh* was employed in conjunction with a *Google Cloud Load Balancer*, which allowed to assign an IP adress to the application so that it can later be accesed through the internet.

5.4 One click deployment example and guide

This section is a guide on how to deploy the pipeline with the code provided in the webpage displayed previously. In this case, we will use the following components for each stage.

- Ingestion: No ingestion since the data is already in bigquery
- Storage: BigQuery

- Transformation: Dbt
- Orchestration: Airflow
- Additional components: Istio, Argocd

Purpose

The goal of this accelerator is to have a one-click deployment of a pipeline that integrates the data stack we have at Astrafy.

Setting up a functional data stack requires a base architecture that involves various components. Integrating these components can often be complex and time-consuming, requiring numerous steps. This accelerator aims to deploy everything automatically while minimizing manual interventions, simplifying the setup process.

Requirements

In order to deploy this architecture, there are some prerequisites that need to be met:

- The new organization needs to have a Terraform workspace to deploy GCP resources.
- Permissions to manage GitHub apps within the client's GitHub.

Components

This accelerator uses Terraform to deploy all the resources it needs. The Terraform code can be obtained from the webpage diplayed before

The code is separated into different modules:

1. Baseline resources
2. Networking module
3. GKE module
4. ArgoCD module
5. Main instance module
6. Airflow module
7. Istio module

Baseline Resources

The baseline resources are located in the parent folder of the repository and include elements that are used by all modules and are necessary for the deployment of the resources. These elements include:

- Main project creation
- Temporary IAM permissions
- GKE namespaces
- Variables
- Versions
- Initialization of modules

The order in which modules are created is not irrelevant. First, we need to create the resources in the Networking and GKE modules, then the ones in ArgoCD and Airflow as they depend on the cluster, and finally, Istio, as it depends on ArgoCD.

Networking Module

The main goal of this module is to create a network on which many of our resources will be based.
To achieve this, we need to set up a network in which the cluster is going to be located.
To set up the network, we use the following resources:

- **module "vpc"**: Creates and configures a VPC network with subnets, secondary IP ranges, and firewall rules for a GKE cluster.
- **module "cloud-nat"**: Configures Cloud NAT to enable instances without external IPs to access the internet.
- **google_compute_global_address "private_ip_alloc"**: Allocates a global internal IP address for VPC peering.
- **google_service_networking_connection "default"**: Establishes a private connection between the VPC network and Google services using VPC peering.

GKE Module

The main goal of this module is to create a GKE cluster in GCP. Once the network and all the adjacent resources are deployed, the GKE cluster is then created in **module "gke"** using this network. Additionally, we also create a **bigquery dataset** which will be used for metering the costs and resource consumption of the cluster.

ArgoCD Module

The deployment of ArgoCD is done through a series of Helm charts. There are two main resources created in this module. The first one is the deployment of ArgoCD itself, which uses the file “values.yaml” contained within the module. The second resource is the different applications that are going to be hosted in ArgoCD. This is done by using the file **app_values.yaml**, which references a different GitHub repository containing the configuration files and Helm charts for the applications to be deployed.

In order to access this repository, first, we need to create a GitHub app and save the credentials of the GitHub app as Terraform variables to later be saved as a Kubernetes secret.

Keep in mind that, even though we are deploying Airflow and Istio now through the Helm charts, the infrastructure needed is not yet deployed and will be done in the corresponding module.

The resources created in this module are:

- **kubernetes_secret "github_app"**: Stores the credentials of the GitHub app to connect to the repository where application Helm charts are located.
- **kubernetes_secret "argocd_client_secret"**: Stores the client ID and client secret for SSO access in ArgoCD.
- **helm_release "argocd"**: Helm release for ArgoCD itself, uses the values.yaml file.
- **helm_release "application"**: Helm release for ArgoCD applications, uses the app_values.yaml file.

Main Instance Module

The main goal of this module is to create an instance where different databases and users will be created for each application. The resources created in this module are:

- **google_sql_database_instance "default"**: Main instance used for application databases.
- **output "instance_name"**: Instance name of the instance used for databases and users.
- **output "instance_ip"**: Instance IP of the instance used for databases and users.

Airflow Module

In this module, we create the following resources:

Main resources:

- **google_sql_database "airflow"**: Creates a dedicated SQL database for the Airflow application.
- **google_sql_user "db_user"**: Creates a user with specific credentials for accessing the SQL database.

Additionally, we create the following Kubernetes secrets that are stored in the cluster created in the GKE module:

- **kubernetes_secret "airflow_broker_url"**: Stores the broker URL for connecting Airflow to the Redis instance.
- **kubernetes_secret "redis_password"**: Stores the password for the Redis instance used by Airflow.
- **kubernetes_secret "webserver_config"**: Stores the Airflow webserver configuration file.
- **kubernetes_secret "airflow-db-credentials"**: Stores the database credentials for Airflow to connect to the PostgreSQL instance.
- **kubernetes_secret "fernet-key-secret"**: Stores the Fernet key for encrypting sensitive data in Airflow.

The content of these secrets is also created as resources from Terraform, with the exception of the webserver config, which uses a file contained in the same directory.

Istio Module

In this module, we use Istio to have a working URL for our ArgoCD application. In order to achieve this, the following main resources are created:

- **kubernetes_ingress_v1 "istio"**: Manages external access to services via Ingress.
- **kubernetes_manifest "istio_managed_certificate"**: Manages SSL certificates for the specified hosts.
- **kubernetes_manifest "istio_gateway"**: Configures load balancers and routes external traffic into the mesh.
- **kubernetes_manifest "backend_config"**: Sets up backend configurations, including health checks.
- **kubernetes_manifest "istio_virtual_services"**: Defines how traffic should be routed within the mesh based on specified rules and conditions.
- **google_compute_global_address "istio_ingress_lb_ip"**: Allocates a global static IP address for the Istio Ingress load balancer, ensuring a stable IP address for external access to the services.

Due to issues when planning resources, it is necessary to first create all the resources except Istio, then create Istio once everything else is created. Details on

how to do so will be provided in the “Step by Step Deployment Guide”.

Step by Step Deployment Guide

How to Deploy

In order to have a working architecture, there are some manual steps that need to be done prior to deploying resources in Terraform.

Manual Step: Creation of GitHub App and Variable Configuration

The first step in the deployment is creating a GitHub application so that ArgoCD can access the .yaml files containing the charts of each application.

To know how to create such an application, you can refer to this guide in Notion.

Once you have the following information, you need to configure them in the Terraform workspace:

GITHUB_APP_ID
GITHUB_APP_INSTALLATION_ID
GITHUB_APP_PEM_FILE

To do so, click on the “variables” section of the Terraform workspace, and then “add variable.”

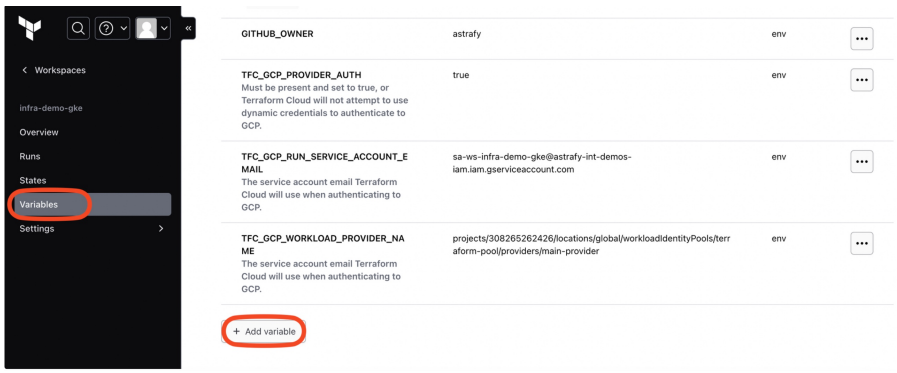

Figure 5.8: Terraform variable creation.

Then, inside the variable creation menu, for each of the variables, create a Terraform variable using the name of the variable and the content obtained in this guide.

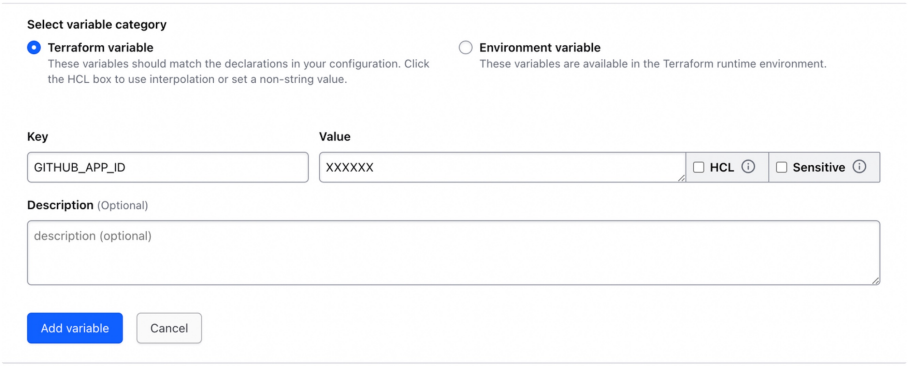

Figure 5.9: Terraform variable customization.

For the **GITHUB_APP_PEM_FILE**, you will also need to select the "sensitive" option. You don't need it in the other two variables.

Once you have performed this step correctly, Terraform should be able to create a Kubernetes secret with the three variables. You can check if it worked by looking at the contents of the Kubernetes secret using the "kubectl" command or an application such as "Lens."

You also need to perform this step for the variables related to the personal access token. These are used to connect to the repository where DAGs are created.

In your GitHub profile, go to **Settings - Developer settings - Personal access token - Fine-grained tokens**.

Once you create the token, select the resource owner.

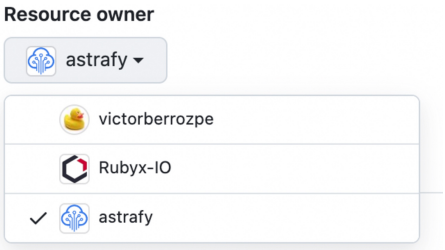

Figure 5.10: Resource owner selection.

Select only one repository, and then select the repository where the DAGs are contained. In this example, this is the DAGs repository.

You also need to give the token read-only permissions for "contents" inside repository permissions.

Once you create it, you will be given the token, which has the format:
github_pat_XXXXXXXXXXXX

You will need to copy this value and create the variable **airflow_pat_password** with the sensitive option in the same way we created the previous variables.

You will also need to create the variable **airflow_pat_user**. Inside this variable, you only need to store the token owner.

The last manual step in this deployment is to create IAP credentials for ArgoCD, Airflow, and Istio. This is done after the creation of the project.

For ArgoCD and Airflow, you need to create the credentials "Create OAuth client id" inside the credentials section of GCP.

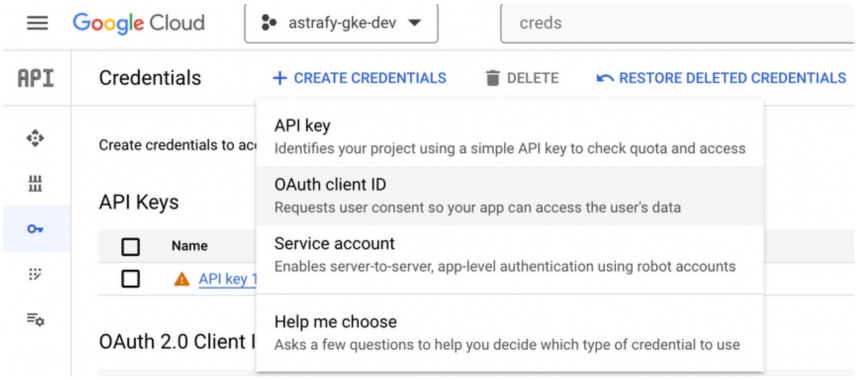

Figure 5.11: Oauth client ID creation.

For Airflow and ArgoCD, you need to have the following configurations, which will create client IDs and client secrets.

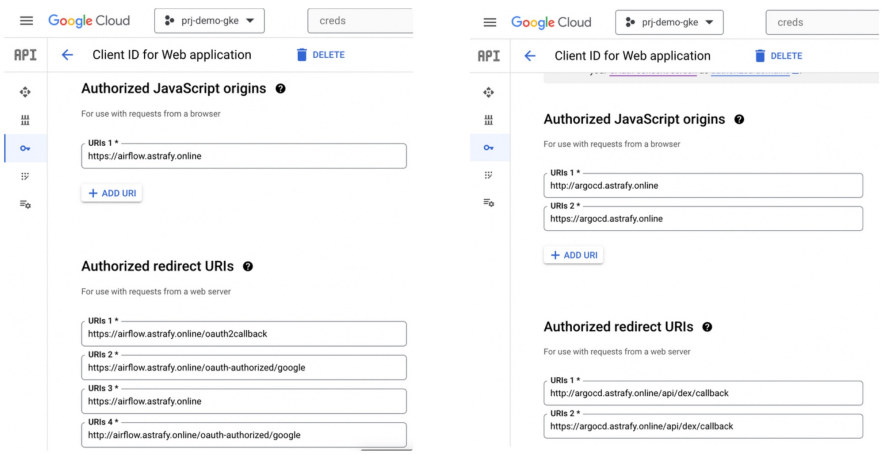

Figure 5.12: Oauth client ID customization.

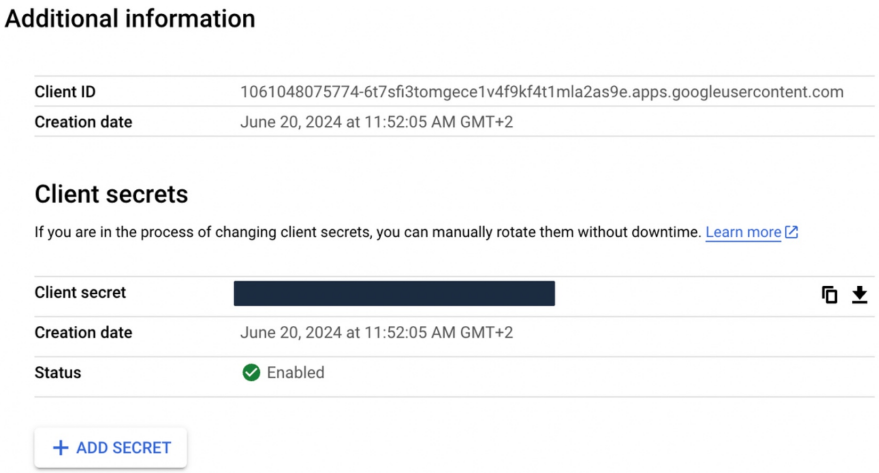

Figure 5.13: Oauth client secret.

You will need to save the client ID and client secret in the following ways:

1. **For Airflow**, replace the values contained in the **webserver_config.py** inside the **OAUTH_PROVIDERS** variable in the Terraform repository. Then copy the code from the whole file and create a Terraform variable called **webserver_config**, and paste the code as the variable value with the sensitive option. After you have done this, you should remove the file or the client ID and secret from the file **webserver_config.py** in order not to push it to a Git repo, as it contains now sensitive information.

```
OAUTH_PROVIDERS = [
{
    'name':'google',
    'token_key':'access_token',
    'icon':'fa-google',
    'remote_app': {
        'api_base_url':'https://www.googleapis.com/oauth2/v2/',
        'client_kwargs':{'scope': 'email profile'},
        'access_token_url':'https://accounts.google.com/o/oauth2/token',
        'authorize_url':'https://accounts.google.com/o/oauth2/auth',
        'request_token_url': None,
        'client_id':
        ↪  "1061048075774-98e1m410bv5g3ua9mevqqticv3kqi0rp.apps.googleusercontent.com",
        'client_secret': "XXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    }
}]
```

2. **For ArgoCD**, you will also need to save the **client_id** and **client_secret** in the Terraform variables **argocd_clientID** and **argocd_client_secret**. Make **argocd_client_secret** a sensitive variable.

3. **For Istio IAP**, you will first need to create the credentials without any additional configuration. Then, once it is created, add with the new client ID the

following authorized redirect URL: **https://iap.googleapis.com/v1/oauth/clientIds/<clie**

You will need to save the client ID and client secret into the Terraform variables:
istio_iap_client_id
istio_iap_client_secret (sensitive)

Finally, create an OAuth consent screen

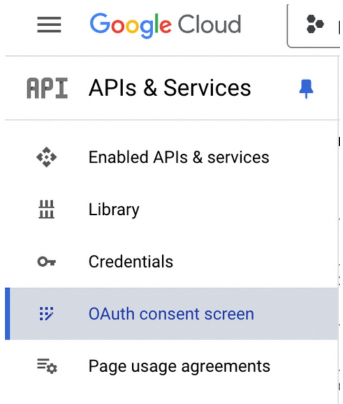

Figure 5.14: Apis and services.

As user support email and developer contact information, write your own email. For the authorized domains, you will need to include the domain that you purchased as well as the domain of the emails that will access it. In this case, we will use the following configuration:

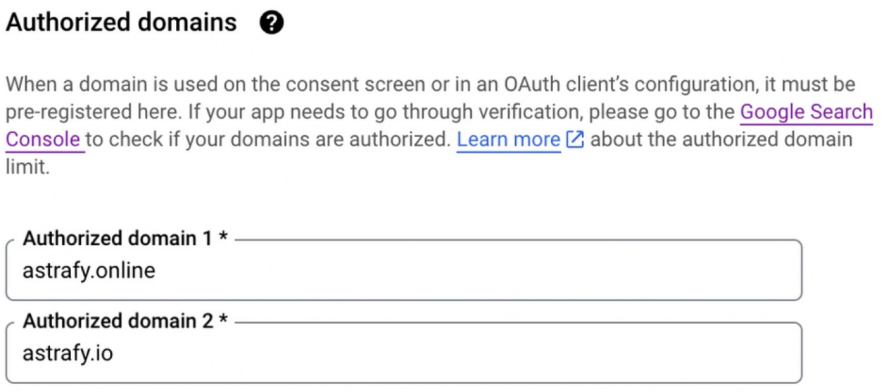

Figure 5.15: Authorised domain configuration.

Then click “save and continue” and add the following scopes.

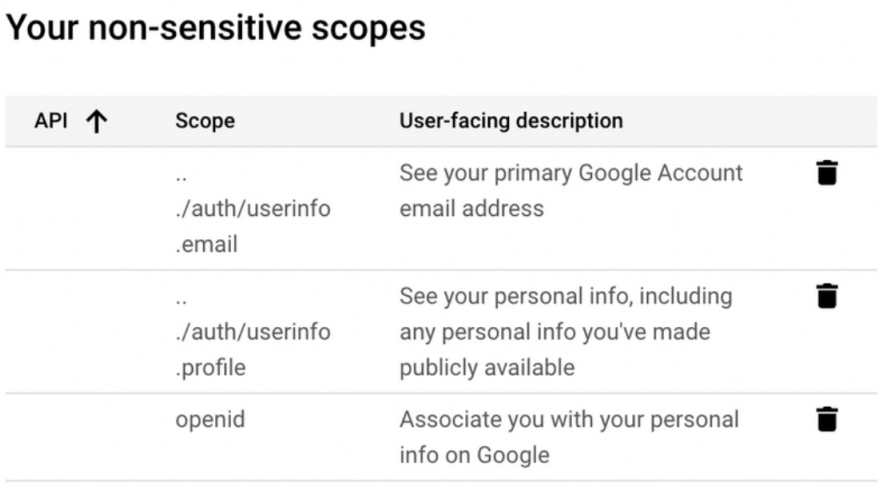

Figure 5.16: Scope configuration.

Then finish creating the resource without adding anything else.

Resource Deployment

Now that the manual steps are performed, you need to copy the project located here. Inside the **terraform.auto.tfvars**, you will need to change the variables so they are specific to the new client. You will likely need to change the following variables:

- **gke_cluster_name**
- The hosts inside **istio_ingress_configuration** and **virtual_services** to match the URL you want to use for your ArgoCD application.

Once variables are configured, the resources need to be deployed by Terraform. Due to an issue with resource creation order in Istio, we will first deploy the project itself. For that, you need to comment all modules and resources related to the main.tf file. Then, once the project is created, you need to uncomment the GKE and networking modules and apply the changes.

Once the network and GKE cluster are created, you need to deploy everything except Istio. The main.tf file should look like this:

```
1  module "gke_auth" {
2    source               =
     ↪  "terraform-google-modules/kubernetes-engine/google//modules/auth"
3    project_id           = var.project_id_gke
4    cluster_name         = var.gke_cluster_name
5    location             = var.google_cloud_region
6    use_private_endpoint = false
7
8    depends_on = [module.gke]
9  }
```

```
10
11 module "gke" {
12   source = "./gke"
13
14   organization_id     = var.organization_id
15   google_cloud_region = var.google_cloud_region
16   google_cloud_zone   = var.google_cloud_zone
17   project_id_gke      = var.project_id_gke
18   gke_cluster_name    = var.gke_cluster_name
19 }
20
21 module "airflow" {
22   source = "./airflow"
23
24   google_cloud_region    = var.google_cloud_region
25   google_cloud_zone      = var.google_cloud_zone
26   project_id_gke         = var.project_id_gke
27   gke_cluster_name       = var.gke_cluster_name
28   airflow_namespace      = var.airflow_namespace
29
30   depends_on = [module.argocd]
31 }
32
33 module "argocd" {
34   source = "./argocd"
35
36   google_cloud_region    = var.google_cloud_region
37   project_id_gke         = var.project_id_gke
38   gke_cluster_name       = var.gke_cluster_name
39   argocd_namespace       = var.argocd_namespace
40   istio_gateway          = var.istio_gateway
41   istio_namespace        = var.istio_namespace
42   use_istio_cdrs         = var.use_istio_cdrs
43
44   depends_on = [module.gke, kubernetes_secret.github_app]
45 }
46
47 # module "istio" {
48 #   source = "./istio"
49
50 #   private_cluster             = var.private_cluster
51 #   istio_ingress_configuration = var.istio_ingress_configuration
52 #   use_crds                    = var.use_crds
53 #   virtual_services            = var.virtual_services
54 #   address_name                = var.address_name
55 #   project_id_gke         = var.project_id_gke
56
57 #   depends_on = [module.argocd, module.gke_auth]
58 # }
59
```

Keep in mind that there are also resources that you need to uncomment outside of the main.tf file. You need to uncomment everything in the **k8s.tf**, **git.tf**, **providers.tf**.

When you apply Istio, the resource **google_iap_web_backend_service_iam_member.iap_user**

may fail due to the load balancer not existing yet. If this is the case, wait until the load balancer is created, modify the name in the Terraform resource, and apply the changes again. You can also comment it before applying Istio and uncomment it once Istio is ready.

To find the name of the resource, you can do the following:
Go to the load balancing section from GCP and select the load balancer.

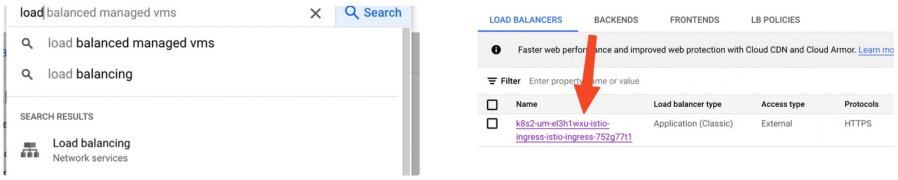

Figure 5.17: GCP Load balancer.

Then scroll down, copy the name of the load balancer, and replace the name in the Terraform code, then apply the changes.

Once the rest of the resources are created, you can proceed by uncommenting the Istio module and deploying the rest of the remaining resources.

Once everything is deployed, head out to the "IP addresses" section of GCP and copy the IP address provided by Istio.

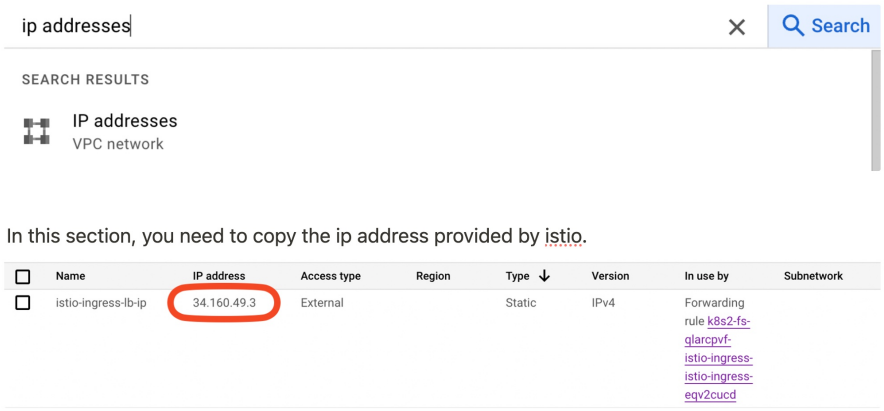

Figure 5.18: IP address provided by GCP.

Once you have the IP address, you need to configure a domain to redirect to the IP address when using a determined URL. In this example, we are going to use **namecheap.com** and the domain **astrafy.online**, but you can use any domain manager and any domain.

Make sure that the domain you configure matches the host in the **virtual_services** and **istio_ingress_configuration** variables.

```
istio_ingress_configuration = {
  allow_http = false
  hosts = [
    {
      host = "argocd.astrafy.online"
    }
  ]
}

virtual_services = {
  vs-argocd = {
    target_namespace = "argocd"
    hosts            = ["argocd.astrafy.online"]
    target_service   = "argocd-server"
    port_number      = 80
  }
}
```

In our case, we will use **"argocd.astrafy.online"**, so we need to set up the domain the following way:

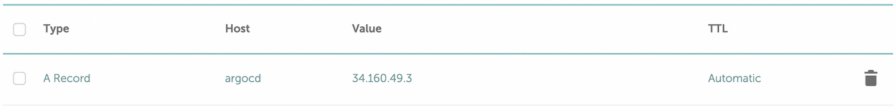

Figure 5.19: Namecheap configuration for domain.

Since we are using the domain **"astrafy.online"**, with this configuration, the end URL should be **"argocd.astrafy.online"**, which matches the host in our variable configuration.

Once this step is completed, everything is ready, but you will need to wait around 20 minutes for the SSL certificate to be provided by GCP to the domain you are trying to access. If after 20 minutes the URL does not work, you will need to sync the Istio applications inside ArgoCD. In this case, you should use localhost to connect to your ArgoCD application.

To connect locally to ArgoCD, first, connect to the cluster by doing:

gcloud container clusters get-credentials <cluster_name> –zone

Once connected to the cluster, you can access ArgoCD by forwarding the corresponding port using an application like “Lens” or the command:
kubectl port-forward svc/argocd-server -n argocd 8080:443

After accessing **http://localhost:8080/**, you will be prompted with a login screen. The credentials are:

User: admin
Password: You can get the password by performing the following command, make sure not to copy the "**kubectl -n argocd get secret argocd-initial-admin-secret -o json**

To re-sync the three Istio applications, click on the sync button of the application, and use the options **-force** and **-replace** on Istio base and ingress. For Istio discovery, a normal sync should suffice.

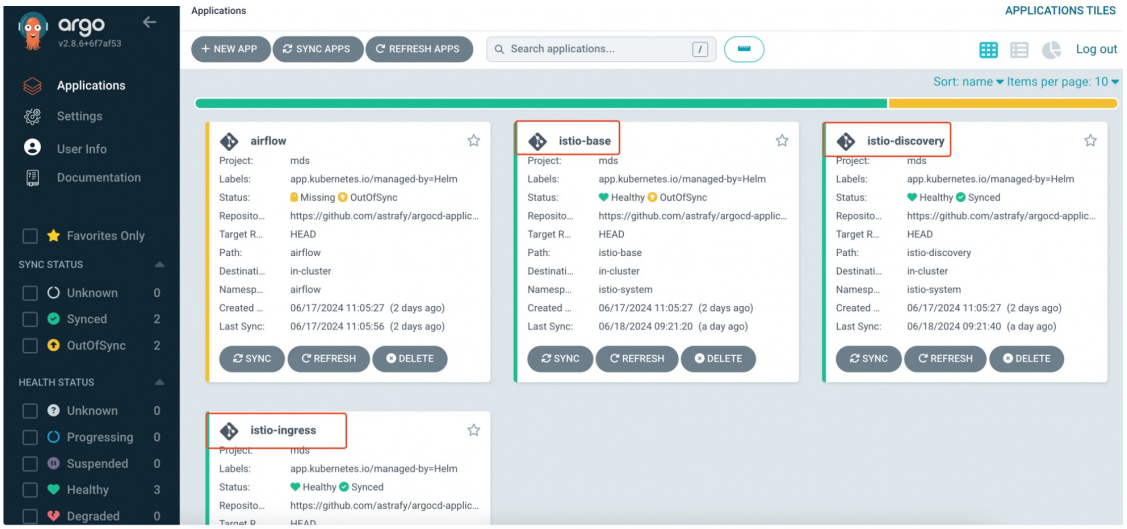

Figure 5.20: Argocd application display

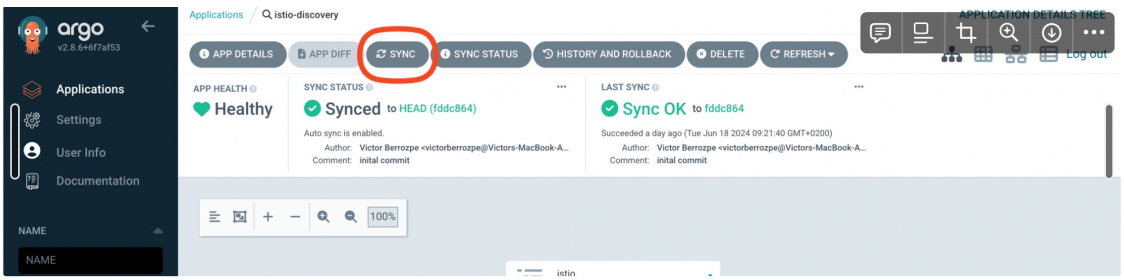

Figure 5.21: Argocd application sync

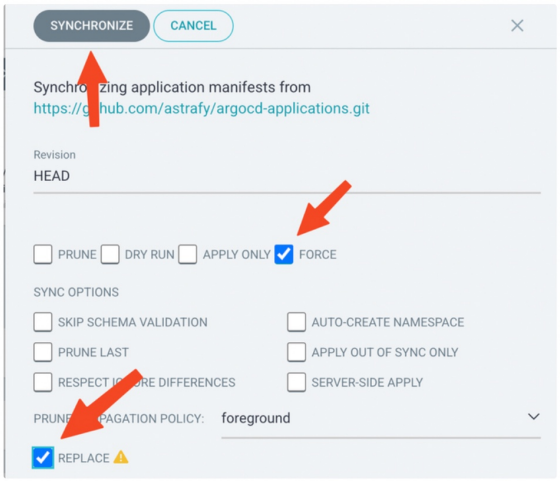

Figure 5.22: Argocd application sync configuration

Once this step is performed, it will take a few minutes to sync, then you should be able to access ArgoCD through the URL you configured.

Resource Deletion

In order to delete all the resources, the easiest approach is to manually delete the project through the Google Cloud Platform UI, as in Terraform there are many dependencies, and then delete all elements from the Terraform state list.

To achieve this, in the directory, you can perform the following steps:
terraform init

Then, you can create a bash script to execute all Terraform state rm commands. In our case, we will use nano:
nano generate_remove_commands.sh

Copy the following code:

```
\#\!/bin/bash
terraform state list | while read -r resource; do
echo "terraform state rm '\$resource'"
done > remove\_commands.sh
```

Save and exit the editor (Ctrl + X, then Y, and Enter).

Do the following commands:
chmod +x generate_remove_commands.sh
./generate_remove_commands.sh
chmod +x remove_commands.sh
./remove_commands.sh

After this, all resources should be removed from the Terraform state.

5.5 Results

Given that self hosted open source technologies do not have a direct cost associated with them, their expenses will be evaluated based on the resources they consume. We will then extract the cost of running these technologies on a Kubernetes cluster based on their resource usage. Additionally, some technologies require extra resources beyond those provided by GKE. The costs of these additional resources will be calculated separately.

Since Google Cloud billing analysis does not provide a direct breakdown of how much each application is consuming, but instead details the costs associated with each resource on Google Cloud, we need to associate all the resources for a given technology to a specific namespace. A namespace in Kubernetes is a way to organize and isolate resources, such as pods, services, and deployments, within a cluster. By assigning each technology to its own namespace, we can perform a detailed analysis by monitoring the resource usage of each namespace in GKE, allowing us to better understand the costs associated with each application.

In this section, we will evaluate the technologies selected for each step of the pipeline based on the metrics selected previously, which are cost, deployment time and complexity. The deployment time and complexity will be measured by how long it takes in the one click deployment phase, how many lines of code are needed and how many unavoidable manual steps need to be performed in order for the application to work.

Keep in mind that, in addition to the selected technologies, a GKE cluster needs to be set up, alongside a VPC network, and an application manager such as Argocd, all of which take up time and ressources. A ressource breakdown table for the additional requirements such as these can be found in the annex.

We will evaluate five key pipeline stages, categorized based on their hosting environment. The stages hosted within the cluster include orchestration, ingestion, and analysis. On the other hand, storage and transformation stages are typically hosted outside the cluster. Storage is external because it involves the database we utilize, while transformation tools are generally applied directly on these databases. We will first start by analysing the results for the technologies hosted in the cluster, and then storage and analysis.

5.5.1 Ingestion technologies

Metric	DLT	Airbyte
Cost	Minimum cost: 0.3975$ per day.	Minimum cost: 0.2575$ per day.
Deployment Time	Around 10 minutes using the one-click deployment (primarily for SQL database creation with Terraform).	Around 10 minutes using the one-click deployment (primarily for SQL database creation with Terraform).
Size and manual steps	159 lines of code for terraform resources. 2 manual steps: creating the GitLab token repository and storing it as a secret for access by the one-click deployment.	141 lines of code for terraform resources. 2 manual steps: creating the GitLab token repository and storing it as a secret for access by the one-click deployment.

Table 5.3: Comparison of DLT and Airbyte based on Evaluation Metrics

5.5.2 Analysis technologies

Metric	Lightdash	Metabase
Cost	Minimum cost: 0.10$ per day.	Minimum cost: 0.18$ per day.
Deployment Time	Less than 3 minutes .	Around 5 miutes.
Size and manual steps	173 lines of code for terraform resources. 2 manual steps: creating the GitLab token repository and storing it as a secret for access by the one-click deployment.	211 lines of code for terraform resources. 2 manual steps: creating the GitLab token repository and storing it as a secret for access by the one-click deployment.

Table 5.4: Comparison of DLT and Airbyte based on Evaluation Metrics

5.5.3 Orchestration technologies

Metric	Airflow	Dagster
Cost	Minimum cost: 0.60$ per day.	Minimum cost: 0.63$ per day.
Deployment Time	Around 15 minutes using the one-click deployment	Around 15 minutes using the one-click deployment .
Size and manual steps	159 lines of code for terraform resources. 2 manual steps: creating the GitLab token repository and storing it as a secret for access by the one-click deployment.	213 lines of code for terraform resources. 2 manual steps: creating the GitLab token repository and storing it as a secret for access by the one-click deployment.

Table 5.5: Comparison of DLT and Airbyte based on Evaluation Metrics

5.5.4 Transformation technologies

Transformation technologies take place directly on the data that you are using, and do not have to be managed by the cluster. Therefore, we cannot calculate their cost based on the amount of resources they consume on the cluster. However, although some technologies do not need to be hosted in the cluster by themselves, they may need additional components such as an orchestrator to run the transformation queries (Such as dbt), or other may directly be integrated in the cloud provider and do not need a cluster setup (Such as dataform). In order to evaluate how efficient one is compared to the other, we need to consider, in addition to the baseline efficiency of each technology, that dbt needs a prior GKE setup including an orchestrator.

Although Dataform is much cheaper given that a GKE infrastructure is not needed and the service is directly accessible within GCP, dbt has some advantages over dataform that are not reflected directly in these metrics. Dataform is not suited to run incremental tables everyday with automated workflows. Incremental tables are tables that are updated periodically by adding only new data rather than reprocessing the entire dataset each time, and they are the most common amongst the types of tables.

If you need to run incremental tables every day, the straightforward approach is to use CURRENTDATE() or any similar function in the SQL query to get the last data. If any run fails, re-executing it will mean running a different query since the date may be different. The recommended approach is to use a variable such as date, however, this is not possible with dataform. You would need to change the query manually, which is not sustainable in automated workflows.

For this reason, using the automated workflows is not suitable in most cases. However, for simple projects that do not need to use dynamic variables, using dataform over dbt has more advantages.

Regarding the deployment time, none of the technologies mentioned for this stage have a deployment time as they are not directly associated with GKE by themselves, but rather with the orchestrator.

Finally, the complexity of dataform is inferior to the complexity of dbt, as it offers much less options, and is directly integrated within GCP. However, dbt offers a larger degree of personalization and can be suited for larger, more complex pipelines.

5.5.5 Storage technologies

For this section, we compare BigQuery and Cloud Storage with BigLake, which like analysis technologies, operate outside the cluster. Both provide data storage solutions for the pipeline within GCP. We will also need to consider analysis cost in this section, as they are tied to the database the user is using.

Big Query storage costs

Storage Type	**Standard**	**Long-term**
Cost (per GB per Month)	$0.020	$0.010
Unactive Storage Duration	None	90 days

Table 5.6: Bigquery Pricing for Different Storage Types

Keep in mind that the first 10 Gb of each month in BigQuery are free of charge.

GCS storage costs

Storage Type	**Standard**	**Nearline**	**Coldline**	**Archive**
Cost (per GB per Month)	$0.020	$0.010	$0.004	$0.0012
Minimum Storage Duration	None	30 days	90 days	365 days

Table 5.7: Google Cloud Storage Pricing for Different Storage Types

Computing costs

GCS and BigQuery have the same computing costs, as both rely on BigQuery's query engine for processing data. When analyzing data stored in GCS using BigQuery, the computation costs are determined by the amount of data processed during queries, measured in bytes. While GCS stores the data, BigQuery handles the query execution, charging based on the volume of data read from the external storage and processed. The same pricing model applies whether the data resides in BigQuery's managed storage or in GCS.

The on demand computing cost for BigQUery is **$6.25 per TiB** or **$0.0061 per**

GB

An advantage that BigLake has over bigquery, is that it allows interaction with non GCP provided warehouses, such as amazons Redshift from AWS.

There is no deployment time for these data storage solutions, as they are directly managed by the cloud provider.

Google Cloud Storage with BigLake and BigQuery offer different levels of complexity based on their intended use cases. BigQuery is simpler to use for analysis tasks, as it abstracts most of the underlying infrastructure management. It is ideal for users who need to run SQL queries on large datasets without worrying about the complexities of data storage and access. On the other hand, GCS with BigLake adds an additional layer of complexity, especially when managing multiple data lakes or handling schema management across distributed storage systems. While GCS with BigLake provides flexibility for integrating various storage formats and querying across them, it requires more configuration and management compared to the straightforward experience of using BigQuery.

5.5.6 Overall price comparison for self hosted technologies

The following graph showcases the price difference of all components used in the technology evaluation. However, this does not include technologies related to storage and analysis, as these are not self hosted, and their cost is directly related to the amount of data you are storing and processing. This diagram also includes costs for Istio and ArgoCD, which are used to build the rest of applications.

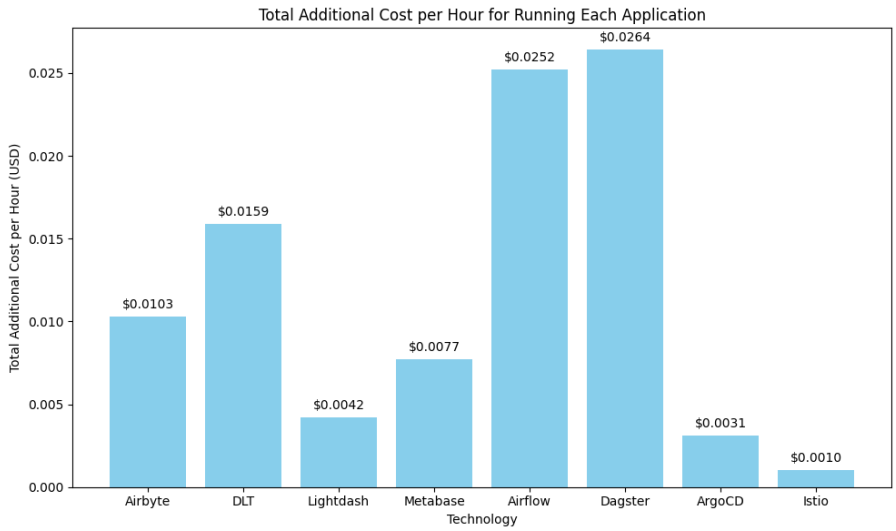

Figure 5.23: Additional GKE cost per hour for each technology.

Detailed cost calculations

6.1 Cluster cost calculation

GCP offers a variety of machine for their compute engine which is used in GKE clusters. These machines offer different combinations of cost and performance to suit a large amount of different purposes. To simplify our calculations, we will use the pricing for the standard e2 machine type, which is the cheapest option offered by GCP for their compute engine used in GKE clusters. Given that our pipeline tasks have relatively low computational demands, the e2 machine type provides sufficient capacity without incurring unnecessary costs.

Baseline cluster cost: 10 USD/hour
It offers fully automated management of the cluster lifecycle, including pod and cluster autoscaling, cost transparency, and automated optimization of infrastructure expenses.[14]

GKE utilizes Compute Engine instances as worker nodes within the cluster. The user is charged for each of these instances based on Compute Engine's pricing until the nodes are terminated. Compute Engine resources are billed on a per-second basis, with a minimum charge equivalent to one minute of usage.

The following table represent the cost per resource used of the compute engines provided by GCP. Given the nature of this thesis, we will use the standard on demand price, without any commitment.

Pricing Type	Predefined vCPUs (USD/hour)	Predefined Memory (USD/hour)
On-demand price	**$0.023993**	**$0.003216**
1-year commitment	$0.015116	$0.002026
3-year commitment	$0.010797	$0.001447

Table 6.1: E2 Machine Type Pricing in Belgium (Europe-west1)

6.2 Technologies GKE resource usage breakdown

Ingestion technologies

Name	CPU Cost (cores)	Memory Cost (GB)
airbyte-airbyte-api-server-0	0.002	0.2240
airbyte-airbyte-api-server-1	0.002	0.2031
airbyte-connector-builder-0	0.001	0.1969
airbyte-cron-cfc6c5b5b-0	0.001	0.4076
airbyte-metrics-599d9-0	0.002	0.3229
airbyte-pod-sweeper-1	0.006	0.0070
airbyte-server-7d8dd7-0	0.002	0.4023
airbyte-server-7d8dd7-1	0.001	0.4804
airbyte-temporal-696d9-0	0.025	0.1077
airbyte-webapp-9845k-0	0.000	0.0045
airbyte-webapp-9845k-1	0.000	0.0046
airbyte-worker-5d6dd-0	0.002	0.5201

Table 6.2: CPU and Memory Usage for Airbyte Components

Total CPU Cores: 0.044
Total Memory Usage: 2.883 GB
Total additional cost per hour from running the application: 0.0103$

Name	CPU Cost (cores)	Memory Cost (GB)
dlt-ingestion-api-7d9f6b5e	0.002	0.330
dlt-ingestion-worker-5a8e4c7f	0.002	0.820
dlt-transformer-builder-3c7d9e6b	0.001	0.200
dlt-cronjob-scheduler-9f8b7d6e	0.001	0.410
dlt-metrics-collector-4e5d9b7a	0.002	0.330
dlt-pod-cleaner-2f8c7d9e	0.006	0.010
dlt-main-server-7f9d8e4b	0.012	0.700
dlt-backup-server-6a9d7c8f	0.011	0.790
dlt-webapp-frontend-4b7f9d6e	0.001	0.005
dlt-webapp-backend-2e8c7b9f	0.001	0.005
dlt-data-loader-8c6f9d7e	0.022	0.610
dlt-scheduler-9e6b7d4c	0.004	0.256

Table 6.3: CPU and Memory Usage for Dlt Components

Total CPU Cores: 0.065
Total Memory Usage: 4.466 GB
Total additional cost per hour from running the application: 0.0159$

Analysis technologies

Name	CPU Cost (cores)	Memory Cost (GB)
lightdash-server-78c6d6f4	0.008	0.512
lightdash-worker-8b9d7f4a	0.007	0.256
lightdash-postgres-4e6b8c9d	0.004	0.128
lightdash-frontend-9a7e2f6b	0.003	0.192
lightdash-cronjob-5d4f3e8a	0.002	0.064

Table 6.4: CPU and Memory Usage for Lightdash Components

Total CPU Cores: 0.024
Total Memory Usage: 1.152 GB
Total additional cost per hour from running the application: 0.0042$

Name	CPU Cost (cores)	Memory Cost (GB)
metabase-server-9f8d7e3c	0.010	0.768
metabase-worker-4c2b8a6f	0.008	0.384
metabase-db-redis-6d4c9b7d	0.005	0.256
metabase-frontend-3a9d7f4b	0.006	0.192
metabase-analytics-7e8b2d3a	0.007	0.512

Table 6.5: CPU and Memory Usage for Metabase Components

Total CPU Cores: 0.036
Total Memory Usage: 2.112 GB
Total additional cost per hour from running the application: 0.0077$

Orchestration technologies

Name	CPU Cost (cores)	Memory Cost (GB)
airflow-pgbouncer-bcl	0.010	0.0556
airflow-redis-0	0.009	0.0481
airflow-scheduler-6f7c8	0.186	0.4688
airflow-statsd-7467b9	0.016	0.1102
airflow-triggerer-0	0.074	0.3693
airflow-webserver-6f6	0.008	0.7399
airflow-webserver-6f6 (replica)	0.007	0.7421
airflow-worker-0	0.162	1.8

Table 6.6: CPU and Memory Usage for Airflow Components

Total CPU Cores: 0.472
Total Memory Usage: 4.334 GB
Total additional cost per hour from running the application: 0.0252$

Name	CPU Cost (cores)	Memory Cost (GB)
dagster-scheduler-7c8d9e2f	0.010	0.220
dagster-worker-3e4a8b6d	0.150	1.600
dagster-runner-8f9d6c7b	0.112	1.350
dagster-db-redis-4b7e8c9d	0.115	0.820
dagster-frontend-2b7d8f4c	0.110	0.500

Table 6.7: CPU and Memory Usage for Dagster Components

Total CPU Cores: 0.497
Total Memory Usage: 4.490 GB
Total additional cost per hour from running the application: 0.0264$

Additional required tools

Name	CPU Cost (cores)	Memory Cost (GB)
argocd-application-controller-0	0.015	0.2983
argocd-applicationset-controller-0	0.004	0.0743
argocd-dex-server-64cfdf5	0.003	0.0739
argocd-redis-864cdff5	0.005	0.0488
argocd-repo-server-86fb66b7	0.007	0.0940
argocd-server-6fbb66f8	0.005	0.0924

Table 6.8: CPU and Memory Usage for ArgoCD Components

Total CPU Cores: 0.039
Total Memory Usage: 0.6817 GB
Total additional cost per hour from running the application: 0.0031$

Name	CPU Cost (cores)	Memory Cost (GB)
istio-ingress-6ff5d5b7	0.005	0.0542
istiod-dc665c98f	0.004	0.0736
kiali-765857fd47	0.004	0.0409
kiali-operator-d64dd4	0.003	0.0108

Table 6.9: CPU and Memory Usage for Istio Components

Total CPU Cores: 0.016
Total Memory Usage: 0.1795 GB
Total additional cost per hour from running the application: 0.0010$

6.3 Non GKE related resources

In addition to the resources directly related to GKE, there are some additional requirements in order for the applications to work. This type of requirements can be general for all applications, such as a network where the cluster is going to be based on, or technology specific resources, such as a cloud SQL database used for orchestrators.

There is a very large amount of external resources needed for the technologies to work, however the majority of these resources do not have an associated cost. In this section, we will only analyse the ones that have a non-negligible cost.

Given the large amount of resources that need to be tracked, we will only analyse networking costs, as the other costs are negligible due to the small work load necessary for these applications to work. For example, the cloud SQL database that will store users for the orchestrators, will cost less than 2 cents per month. In our case, the only resource that respresents a non-negligible cost is the networking necessary to set up the cluster as well as the load balancing rule to associate an external IP to our applications, and be able to access them from the internet.

Cost for load balancer per hour: 0.025$
Cost for vpc network per hour: 0.04$

6.4 Additional measurements

Technology	Terraform Resources
Airbyte	141
DLT	159
Airflow	158
Dagster	213
Lightdash	181
Metabase	240

Table 6.10: Number of lines of code for terraform Resources by Technology

Discussion

7.1 Synthesis of Findings

The pipelines built in this thesis do not aim to process a large amount of data, therefore their costs will not be significantly different from each other. However, this small difference in cost can be much more significant in real progress and needs to be taken into account.

The purpose of these pipelines was to store, transform and analyse billing data from GCP in small quantities (Less than 10Gb of data), evaluating the results we can observe that the best technologies in terms of pipeline architecture for each step would be the following:

- Ingestion: No data was ingested in this case since it was already provided by GCP, but in case it was necessary, Airbyte proved to be a bit cheaper, and more or less equal in terms of complexity and deployment time.
- Storage: For our example, BigQuery would be the superior alternative, as Google cloud storage only starts to shine above BigQuery if we want to store old data in large quantities which is not the case of this study. Also, since we have less than 10Gb of data, the storage in BigQuery is free.
- Transformation: Regarding transformation technologies, a lot of good points can be made for dataform, which would not need an orchestrator, and would therefore be much cheaper than owning a GKE cluster to host it. However, in this case dbt would be superior to dataform given the constraints dataform has, even though its more expensive.
- Analysis: Costwise, Lightdash has proven to be cheaper to host than metabase in the GCP cluster. Although the choice of these types of technologies can be heavily biased by the features each one provides, and depends a lot on personal preference. Still, for the purpose of this pipeline we will select lightdash as the better choice.
- Orchestration: Both orchestration tools prove to be more or less equal in terms of complexity and cost. Airflow has a more established platform with a larger community and ecosystem. On the other hand, if integration with machine learning frameworks is your priority, Dagster could be the better choice. Given that we want a simple orchestrator for this task, Airflow is the better option.

7.2 Implications

The findings of this study have significant implications for both theory and practice. Theoretically, it validates the effectiveness of microservice architectures in data engineering, reinforcing the benefits of using cloud tools to handle scalability and complexity. From a practical standpoint, this research provides actionable insights for organizations looking to implement data pipelines efficiently. It underscores the necessity of automation and modularity, making these best practices applicable to a broad range of industries.

This study has also produced a webpage that could be expanded to be used by users or companies that want to build their own hosted pipelines.

Conclusion

8.1 Summary of Findings

This research explored the construction and evaluation of modern data pipelines on Google Cloud Platform, focusing on small scale workloads like GCP billing data. The pipelines developed were evaluated across key stages: ingestion, storage, transformation, analysis, and orchestration. The findings demonstrate that technologies like Airbyte, BigQuery, dbt, Lightdash, and Airflow are optimal choices for specific pipeline stages in terms of cost efficiency and complexity. While the differences in cost are small due to the limited data used in this study, these variations can have a significant impact when pipelines are scaled for real world applications with larger datasets.

8.2 Contributions

The primary contribution of this research is a comprehensive framework for automating and managing data pipelines using cloud-native, open-source technologies. By employing Infrastructure as Code tools like Terraform, the study significantly reduces the complexity associated with setting up and maintaining these pipelines. It also highlights the strengths and limitations of various technologies in the context of small scale data, offering insights that can guide practitioners in choosing appropriate tools based on their specific use cases. Furthermore, the creation of a one-click deployment solution provides a practical, reusable tool that can be adopted by companies seeking to streamline their own pipeline setup processes.

8.3 Future Work

This study opens up several avenues for further research. Exploring how the pipelines perform under heavier workloads would provide valuable insights into their scalability and long term operational costs. Another potential area of research is to adapt the existing one click deployment tool for multi cloud environments, allowing organizations to build resilient and cost-efficient pipelines across different cloud providers. Additionally, the one click deployment tools could be expanded by asking the user a series of questions regarding the purpose of the desired pipeline and the environment it will be set in to provide a tailored selection of technologies.

Bibliography

[1] M. W. V. Alstyne, G. G. Parker, and S. P. Choudary, “Pipelines, Platforms, and the New Rules of Strategy”, en, 2016.

[2] A. R. Munappy, J. Bosch, and H. H. Olsson, “Data Pipeline Management in Practice: Challenges and Opportunities”, en, in *Product-Focused Software Process Improvement*, M. Morisio, M. Torchiano, and A. Jedlitschka, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 168–184, ISBN: 978-3-030-64148-1. DOI: 10.1007/978-3-030-64148-1_11.

[3] J. Densmore, *Data pipelines pocket reference*. O’Reilly Media, 2021.

[4] N. Dmitry and S.-S. Manfred, “On micro-services architecture”, *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.

[5] P. K. Janert, *Data analysis with open source tools: a hands-on guide for programmers and data scientists*. " O’Reilly Media, Inc.", 2010.

[6] T. Lehtonen, S. Suonsyrjä, T. Kilamo, and T. Mikkonen, “Defining metrics for continuous delivery and deployment pipeline.”, in *SPLST*, 2015, pp. 16–30.

[7] Y. Brikman, *Terraform: Up and Running*. " O’Reilly Media, Inc.", 2022.

[8] E. Utami, H. Al Fatta, *et al.*, “Analysis on the use of declarative and pull-based deployment models on gitops using argo cd”, in *2021 4th International Conference on Information and Communications Technology (ICOIACT)*, IEEE, 2021, pp. 186–191.

[9] A. Zerouali, R. Opdebeeck, and C. De Roover, “Helm charts for kubernetes applications: Evolution, outdatedness and security risks”, in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, IEEE, 2023, pp. 523–533.

[10] I.-C. Donca, O. P. Stan, M. Misaros, D. Gota, and L. Miclea, “Method for continuous integration and deployment using a pipeline generator for agile software projects”, *Sensors*, vol. 22, no. 12, p. 4637, 2022.

[11] M. Labouardy, *Pipeline as code: continuous delivery with Jenkins, Kubernetes, and terraform*. Simon and Schuster, 2021.

[12] S. Hyvämäki, “Data processing pipeline automation on cloud platform”, 2019.

[13] P. Barlas, I. Lanning, and C. Heavey, "A survey of open source data science tools", *International Journal of Intelligent Computing and Cybernetics*, vol. 8, no. 3, pp. 232–261, 2015.

[14] *Pricing | Google Kubernetes Engine (GKE)*, en. [Online]. Available: `https://cloud.google.com/kubernetes-engine/pricing` (visited on 08/27/2024).