# Domain Pulse

Architectural Awareness

# Table of Contents

# 1 Introduction

Domain Pulse was originally conceptualised as an exploratory project with the goal of determining whether an application that could leverage sentiment analysis to provide meaningful insight into a generic area of interest to support decision making, could exist and be feasible in principle.

Throughout Domain Pulse's development lifecycle, the project evolved and focussed as we identified areas of potential application, useful and novel features and metrics, and target audiences. Gradually, Domain Pulse solidified into a stunning, and more importantly, useful, dashboard-style web application which makes insight from sentiment analysis more accessible to a wider variety of users from a number of different contexts - from business, to hospitality and entertainment, to research, and more.

This document describes the architectural design of the application. First, important quality requirements are identified and discussed, followed by an architectural diagram of the system. Next, the various architectural patterns employed in the design of Domain Pulse are highlighted and comprehensively explained with regard to how each pattern and structure enables our application to meet its quality requirements, as well as point out additional benefits associated with such design choices. Finally, we conclude with a discussion about our deployment strategy and how it is enabled by our architecture.

# 2 Quality Requirements

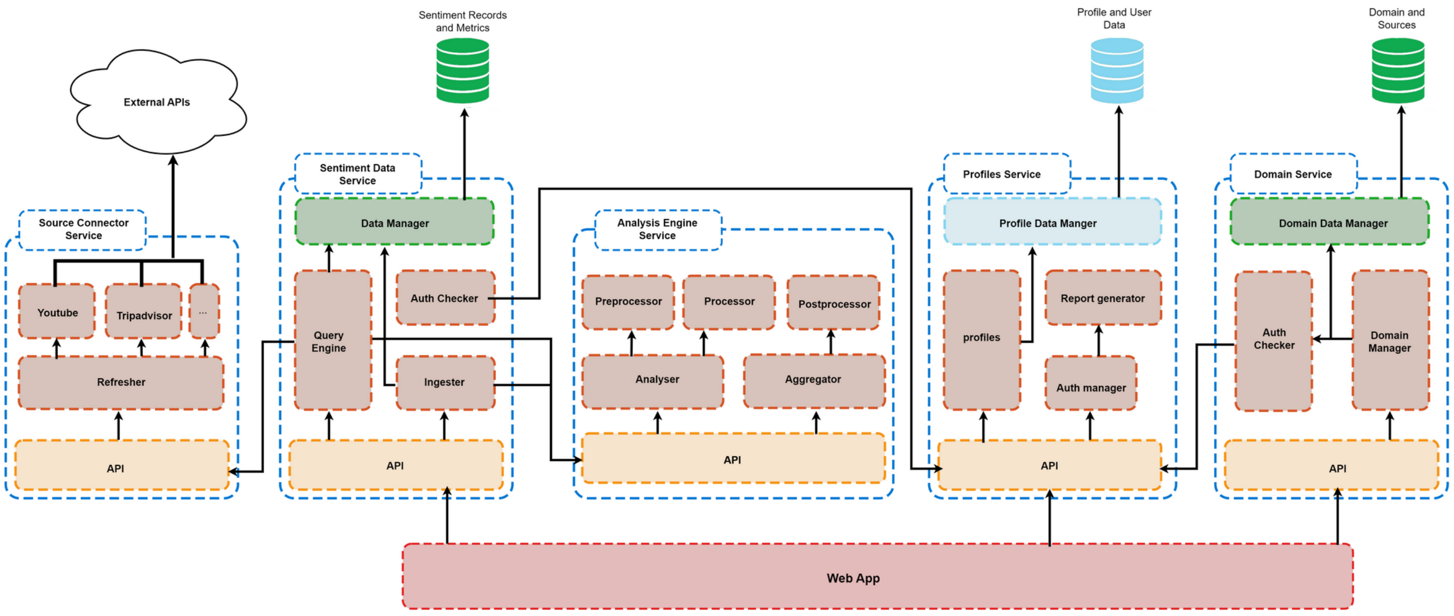
Domain Pulse, at its core, is all about making sentiment analysis more accessible. Domain Pulse gives users the power to gain and leverage novel insights from review and comment data through machine learning, who otherwise would not have the capacity to conduct this type of analysis. Hence, Domain Pulse is intentionally designed to be highly **usable** and useful to a wide user-base, with a variety of technical skills and experience.

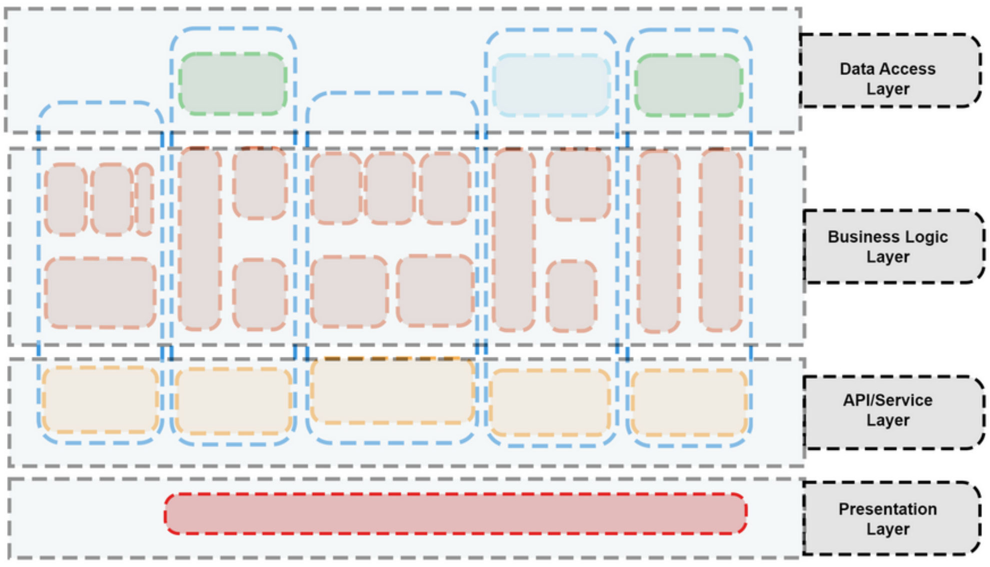
Domain Pulse performs sentiment analysis of large quantities of textual data, and due to the usability requirements of our application, we deem it to be crucial that Domain Pulse remains performant even under the stress of many concurrent users - slow response and analysis teams make an application frustrating and unusable, hence, crucial quality requirements of our system are **scalability** and **performance**, in order to support our most important quality requirement of **usability**.

When Domain Pulse was conceived, it was to be a working prototype demonstrating that such a sentiment analysis tool can exist and provide meaningful insight to its users. As Domain Pulse evolved, **modifiability** emerged as a crucial quality requirement - features such as new sentiment metrics, data sources, and insight avenues should be able to be added with minimal impedance as the product matures and new needs of users are identified. Furthermore, **modifiability** (a component of which is **maintainability**) is a crucial requirement for any project following the iterative-enhancement rhythm of the Agile methodology.

Finally, all modern web-applications require solid security. Attacks on server-hosted applications are extremely common and prevalent, and the consequences of not adequately being prepared for these attacks can be catastrophic. Compromise of user data confidentiality, system integrity, and application availability are serious concerns that have real world consequences for users, as well as the reputation and success of our Domain Pulse. As such, Domain Pulse naturally considers **security** as an important quality requirement too.

# 3 Architectural Diagram





## 4 Architectural Patterns

Domain Pulse uses a sophisticated, but simple and fit for purpose architecture that minimises coupling between components and enables the compliance with our quality requirements, while ensuring that the structure is intuitive and cohesive. A number of effective and suitable architectural patterns are employed in order to meet these goals.

In accordance with a **Service-Oriented Architecture**, Domain Pulse consists of five independently deployable subsystems - enabling both vertical and horizontal scaling, aided by automatic load balancing. By distributing processing load across independent subsystems, **scalability** of the application is improved.  Furthermore, this service isolation promotes interoperability between components using uniform interaction protocols (HTTP), between the need for complex integration of unrelated services which might have a negative impact on **performance**. This approach ensures Domain Pulse meets its quality requirements of **scalability** and **performance** in an intelligent and sustainable manner.

Additionally, this **Service-Oriented architecture** naturally lends itself to using the most appropriate database for each relying service, eliminating impedance as a result of limiting ourselves to a single DBMS on a single machine. For example, our Warehouse service which handles unstructured text data can make use of a document-based NoSQL database whereas our Profiles service, which works with highly structured and consistent data can make use of an Object-Relational Database. This flexibility enables **modifiability**. Furthermore, by leveraging different database technologies for different project services, we can take advantage of the **security** benefits of different database paradigms without having to commit to the exclusive use of one DBMS. For example, by leveraging an ORDBMS for the persistence of user Profile data, we can make use of an ORM mapping to protect against SQL Injection attacks.

In order to achieve further **modifiability**, the overarching **Service-Oriented Architecture** is supported by the use of other architectural patterns. Each individual service is structured according to a **Layered Architecture**, where each service consists of an API, Business Logic, and Data layer. Consequently, not only are the five independent services separated according to function, within each service, clear separation of concerns is achieved. This realises maximal modifiability of application components in both the horizontal and vertical axes.

To further reduce coupling between our services and the databases, we leverage the **Repository Pattern** - which places a well-defined interface between the business logic code and the underlying database system. This allows for the easy ‘swap out’ of database systems should the need arise, with minimal changes to the rest of the codebase, further emphasising the **modifiability** of the application. Additonally, the use of the Repository pattern as described previously ensures that all interaction with the database happens through uniform, predefined interfaces. This reduces the risk of unintentional or malicious interaction with the database from unrelated services, which is a benefit to **security**.

Tying together all of the above, Domain Pulse’s most crucial underlying process is its Sentiment Analysis pipeline, which handles the process whereby sentiment data is collected from a source, cleaned, preprocessed, analysed, persisted and aggregated. This process is structured as a virtual **Pipe-And-Filter architecture** in the sense that the flow of data from collection to analysis is handled by a series of computational stages each responsible for a different task (ex: collection, preprocessing, analysis, persistence, aggregation). These stages/filters are split across a total of three independent services (the Source Connector, Data Warehouse/Query Engine, and Analysis Engine). By distributing this data and computation intensive load we promote **scalability** of this extremely important process while effectively enabling pipelining of sentiment data processing tasks, improving **performance**. Each stage of the ‘pipe’ logically (and in some cases physically) separate from the others, and each stage is responsible for a different portion of the process (ex: collection, preprocessing, analysis, persistence, etc). This means that a change to one part of the pipeline has no impact on the others - further promoting **modifiability** and **maintainability**.

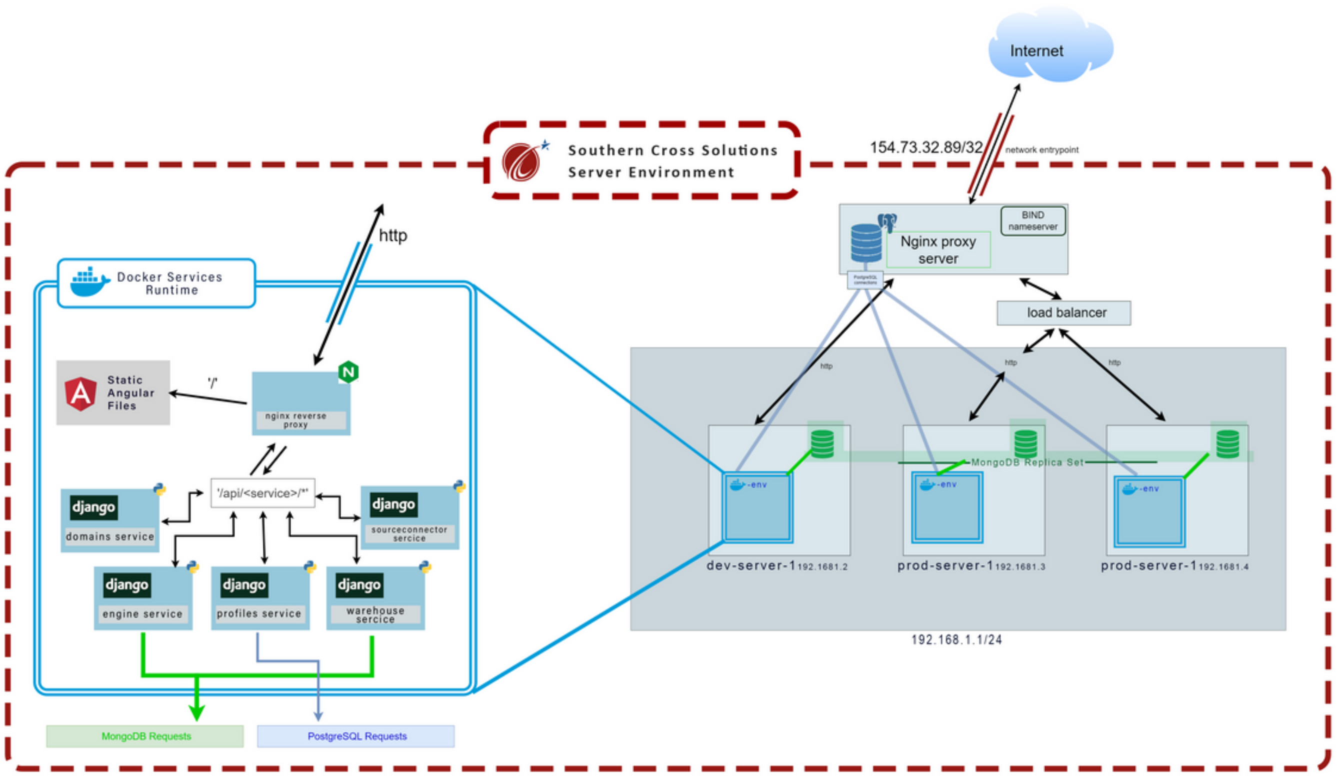
Our frontend design is rooted in the **MVC pattern**, assisted with the NGXS frontend library. We use this pattern because it fosters consistency and simplicity in our user interface. We have a single component of the application which stores all of the state of the client UI, serving as the single point of truth. We have subcomponents of the UI select the sections of the monolithic state which they are concerned about, and they simply transform this data to be shown to the user. This approach not only ensures a cohesive and clear user interface but also significantly enhances **modifiability**. By consolidating all state and state operations within a singular point in the application, we've greatly improved the developer experience, enabling seamless modifications to the codebase. This has enabled us to allow the user to remain on a single page and access all of the functionality our platform has to provide.

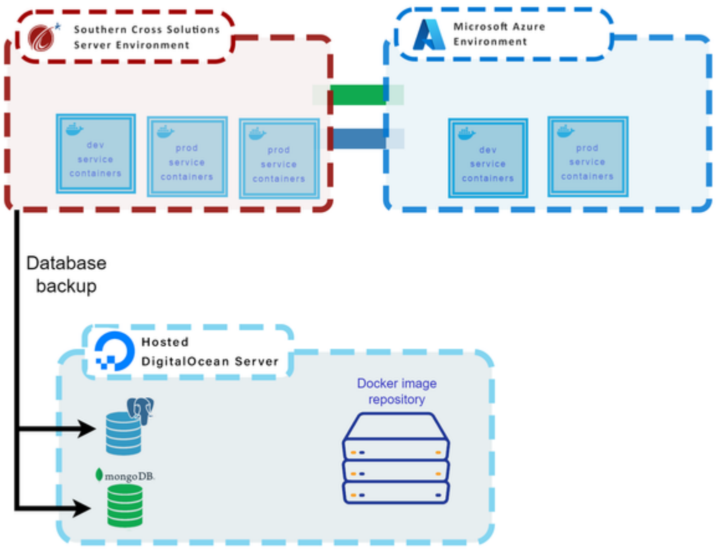
For access into Domain Pulse, an authentication system has been developed, which of course, handles sensitive credentials. Furthermore, many users, specifically business users, may want the domains and sources they view to be highly protected for marketing tactic related reasons. This introduces the need for security within our system. Our architecture helps enable these security quality requirements within our system, again through the use of our **Service-Oriented Architecture**. This is due to **Service-Oriented Architecture** helping separate the services, and therefore decouple the sensitive information used within these services, meaning that a security breach of a single service, does not result in the security breach of the entire system. This improves overall system **security**.

# 5 Deployment

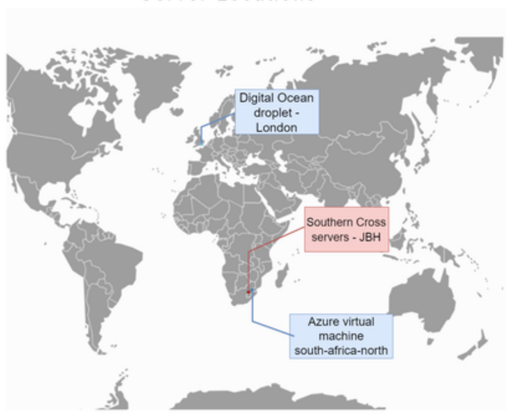
Note that not only are our services built with scalability in mind, but so are our databases. We use redundant database nodes to replicate our database such that not a single database daemon is overloaded with requests. This database replication also increases performance, as we can make sure that when we scale services to separate regions, the database from which the services pull data, is physically close, reducing round trip response times. When a new database node is created, it is connected to the replica set, which acts as a hive. When a service wants to connect to a database, it first connects to the replica set, which will then choose the node it will connect to based on which one has the lowest latency (closest).

The diagrams below describe Domain Pulse's deployment strategy.





Server Locations


Digital Ocean droplet - London
Southern Cross servers - JBH
Azure virtual machine south-africa-north