Software Development: Start-Up Development Cycle

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Samer Nahed Kadih

Spring 2023

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

Briana Morrison, Department of Computer Science

Software Development: Start-Up Development Cycle

CS 4991 Capstone Report, 2023

Samer Kadih
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
snk6ej@virginia.edu

Abstract

A Baltimore, Maryland-based tech start-up that recently acquired early-stage venture capital funding is developing a suite of services to ensure data cleanliness and data quality when moving big data. With different parts of the stack abstracted for convenience, I was tasked with building APIs so that they may adequately communicate with each other. In order to keep up with the fast pace of development in the start-up space, it is necessary to build thorough automated testing for continuous integration (CI). I used FastAPI and Starlette to provide a seamless framework for building and testing APIs, as well as Insomnia for accessing and logging raw HTTP traffic when simulating the consumer-end. The ability to actively keep track of and modify API services based on different teams' needs from the back-end is invaluable to the success and pace of that team. As the company matures, it will make abstractions between services more clear-cut, so that there are minimal changes to and greater reliability for API services.

1 Introduction

It is hard to imagine on the eve of the tech revolution in the mid-1980s, just over thirty years ago, that only a handful of communities made use of the internet. Only privileged groups of researchers and academics had the skills to communicate with their peers over the then-recently developed Advanced Research Projects Agency Network ("ARPANET", 2020).

The first step to commodifying the internet was to break down its barriers to entry. In 1989, Berners-Lee started developing a suite of protocols to standardize how computers communicate with one another. This set of protocols would go on to be known as the World Wide Web, and the potential that it sowed spawned an economic upswing driven by high investor sentiment.

As the world wide web developed, so did the complexity of commercial products. The software development cycle is nowadays abstracted as a timeline of milestones that conform to their own set of protocols. As a software engineer, one might classify themselves as a front-end or back-end developer. With higher granularity, a React or Python developer. In order to build a product for use by the general public, it is important not only to conform to the protocols adopted by the world wide web, but to develop on one's own set of protocols when appropriate.

Commercial applications tend to involve a sophisticated assortment of logical units, which themselves might comprise complex subunits of logic. Teams involved in the development of start-up technologies put an emphasis on agile methodologies, since demonstrating a Minimum Viable Product (MVP) and meeting product deadlines is necessary for raising funds. Scrum is a specific framework for agile product development that values collaboration and efficiency. At the heart of Scrum, high-level projects are broken down into sprints, which consist of product backlogs, sprint backlogs, and sprint goals (D. Fernandez and J. Fernandez, 2008).

In order to keep up with the pace of development, software engineers should sink as little time as possible digging through the documentation of in-house modules. Teams that thrive with this methodology do their due diligence in high-level system design, so that critical modules can support intuitive APIs. Thus, building and digesting protocols for naturally

communicating between services are important skills for software engineers to develop.

The services that I chiefly worked on are the startup's flagship tools for data integration. These include safely migrating large volumes of data from one database management system (DBMS) to another, a tool to compare data stores with different sources to target anomalies between the two, as well as a tool to write "checks" for data that will handle anomalous data accordingly as it moves downstream. These tools serve as a protocol for consumers to move their data around with ease and peace of mind, or as a way to run preliminary checks on different versions of their data to protect against data drift. Moreover, we wanted to put power in the hands of data scientists by allowing them to use their own languages and protocols when using our services.

2 Related Works

The aforementioned tools for ensuring data quality act on top of database management systems. Two popular ways of modeling data are by a relational database or a document database. A relational database stores structured data that conforms to a predefined schema, whereas a document database stores documents that may contain unstructured data. PostgreSQL and MongoDB are examples of a relational DBMS and document DBMS respectively. Thus, it is worthwhile to explore the functionality that these systems provide out of the box.

PostgreSQL is similar to MySQL, but more complicated in that it offers more complex data types and functionality out of the box. In terms of validating data whenever a service inserts or updates a record, PostgreSQL provides functionality for specifying that values in a column must satisfy a Boolean expression ("Constraints", n.d.). However, the nature of these constraints is limited by the data types and their respective operations in question. For example, a column that only accepts integer fields can be constrained to a range of values by using the supported operators for inequalities "<" and ">", or any other form of static verification.

MongoDB supports unstructured data by design, though still provides a method catered to established, large-scale businesses for establishing schema validation ("Schema Validation", 2020). Similar to PostgreSQL, document data that is stored or updated must conform to pre-defined constraints.

Apart from database management systems, there are third-party businesses that offer services similar to ours. For example, Talend is a data management service that provides functionality for unifying data from multiple sources. It supports a variety of APIs critical to businesses with dynamic data applications that need to abstract away the headache involved with moving data around ("Data Preparation Solutions", n.d.). They also provide a service for defining data constraints, though it is an interface of tools already supported by the corresponding DBMS, so it is limited in the same capacity. This is similar but not quite as powerful as the validation engine that we are developing, which will allow for more freedom in the nature of the constraints.

3 Project Design

Over the course of my summer internship, I made use of tools that abstract away much of the boilerplate logic involved with building web-services. Django is a high-level, Python-based web-framework, which provides the skeleton of our application. We made a design decision in using Uvicorn as the web-server, as it supports the Asynchronous Server Gateway Interface (ASGI). The ASGI is a convention for forwarding multiple, asynchronous requests within our application. Starlette is an ASGI framework built on top of Uvicorn. We integrated Django with FastAPI, which is a framework built on top of Starlette that provides useful features when working with high volumes of data, such as data validation and serialization.

The system architecture can be broken down by the services that the startup provides. Each service is a module, written primarily in Python, that packages together different files to integrate with FastAPI and CircleCI. In this way, engineers that specialize with different services can allow themselves to silo themselves from other services while still being able to employ their functionality, and with faith that those services work as intended. Outside of the product services that this startup provides, there are also general-purpose modules whose abstraction from the rest of the system was deemed worthwhile enough to

separate. This chiefly consisted of logic to handle different database management systems such as MySQL or PostgreSQL, since part of the value proposition is in the generality of these services.

On a higher level, these services typically involve moving and scanning large volumes of data, and getting notified whenever sensitive information is gathered about or prompted of them. Apache Kafka is a platform that is built to process real-time data streams in a way that prioritizes high-throughput and low-latency. The specific protocol we used when employing Kafka was its Pub/Sub messaging service, in which different services offered by the startup "subscribe" to the a designated "producer," where all the logic is written that directly involves processing or accessing data that belongs to the consumer. Moreover, Kafka is a tool for distributing data infrastructure. The logic for a module sits inside of a "broker," also known as a "node." This node can be thought of as a unit of computing resources necessary to run the logic that we programmed into the module. An important facet of our services, though, is their reliability, which is why we balance the load of queued services among *multiple* different nodes, called a "cluster." This allows for an optimized response time, and no single node is overburdened.

When I started working for this startup, they were just getting into their next round of funding and had a few clients that agreed to test-drive their services. One client approached them with a dataset so large that it was dubbed "Big Bertha." Until this point, all data being processed on our end happened chronologically, in order of the packets of data as they are received. We realized that we needed to have a way to parallelize scanning and processing bigger databases, given the computational intensity of some of our most popular database checks. The biggest challenge we encountered was finding a way to reconcile this parallelization with the fact that the streams of data are forming their own hypothesis as they are being checked. Unfortunately, the solution to this varies depending on the check that is used on the stream of data. For example, on some statistical checks that we provide for their "protect" service, we would routinely have the different nodes communicate with each other on the expected value of the data in each node. This typically meant generating confidence intervals between the data and making sure they regularly sync up between one another.

4. Results

Over the course of three months, tools that I had a helping hand in developing were tested by small businesses. They would be able to take advantage of our suite of services in beta testing, and we would be able to regularly assess customer satisfaction. Towards the end of the internship, we had established a good rapport with a big bank based in Hong Kong, as well as a well-known U.S. fashion retailer. Due to the start-up nature of this internship, the team adhered to agile methodologies for efficiently deploying a plethora of new services and modifications to existing services. Because of this, it is difficult to differentiate any improvements to customer productivity from any one feature, but there is something to be said about the system as a whole.

The general consensus is that these services do a good job of centralizing different datastores for data-driven businesses, and lessening the barriers to entry for data analytics and exploration. Teams working with us in beta testing found that there was less required overhead for applying their data to different problems, as their data pre-processing could be automated to run over all of their data and data stores.

5. Conclusion

In conclusion, this research paper detailed the development of a suite of services for a Baltimore-based tech start-up aimed at ensuring data cleanliness and data quality when moving big data. By utilizing FastAPI, Starlette, and Insomnia, the project successfully built and tested APIs that allowed for seamless communication between different parts of the stack. The start-up's services, including data migration, anomaly detection, and data checks, provided valuable tools for businesses to manage their data efficiently. As the company matures, it plans to refine the abstractions between services for increased reliability and minimal changes. The internship experience highlighted the importance of agile methodologies, collaboration, and efficient system design in the fast-paced start-up environment.

6. Future Work

Moving forward, the startup aims to empirically determine and implement the most pressing data quality concerns by incrementing on user feedback, with an emphasis on scalability and generality. A primary concern of any startup is ensuring that they can bring value to a market in due time. By the end of my internship, the most valuable resource under development from a time-saving perspective was a Dockerized tool for the testing team, wherein they would be able to generate “flawed” data for any environment and DBMS. Docker is a tool for virtualizing software so that developers with different computing environments can ensure a consistent deployment of services. Much of the user feedback seemed to encourage services that we already had the back-end infrastructure to iterate upon, but catered to a specific functionality. Thus, a tool that could abstract away much of the boilerplate concerned with generating anomalous data would be invaluable to the team.

References

[1] D. Fernandez and J. Fernandez. 2008. Agile Project Management - Agilism Versus Traditional Approaches. Journal of Computer Information Systems 49.

[2] Living Internet. 2020. ARPANET - The First Internet. Retrieved May 11, 2023 from https://www.livinginternet.com/i/ii_arpanet.htm

[3] MongoDB. 2020. Schema Validation. Retrieved May 11, 2023 from https://www.mongodb.com/docs/manual/core/schema-validation/

[4] PostgreSQL. n.d. Constraints. Retrieved May 11, 2023 from https://www.postgresql.org/docs/current/ddl-constraints.html

[5] Talend. n.d. Data Preparation Solutions. Retrieved May 11, 2023 from https://www.talend.com/products/data-preparation/