NOVA IMS Information Management School

MDSAA

Master Degree Program in

Data Science and Advanced Analytics

Development of an Application to Run Integration Tests on a Data Pipeline

Pedro Henrique Medeiros de Oliveira

Internship Report

presented as a partial requirement for obtaining the Master's Degree Program in Data Science and Advanced Analytics

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

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

DEVELOPMENT OF AN APPLICATION TO PERFORM INTEGRATION TESTS ON A DATA PIPELINE

by

Pedro Henrique Medeiros de Oliveira

Internship report presented as a partial requirement for obtaining the Master's degree in Advanced Analytics, with a Specialization in Business Analytics

Supervisor: *Mauro Castelli, Ph.D.*

October 2022

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledge the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Porto, 21st October 2022

DEDICATION

I would like to dedicate this internship project report to my family. To my parents, my grandparents, and my brothers, for always being by my side, especially through difficult times, always encouraging and believing in me.

ACKNOWLEDGEMENTS

I would like to especially thank my supervisors and my internship team, as without them the development of the project would not have been possible. Their guidance and support carried me through all stages of the internship project.

ABSTRACT

As new technologies continue to surge, the ever-growing complexity of modern software products and architectures significantly increases the difficulty and development costs of a fully tested application. As agile methodologies enforce faster delivery of new features, a tool to automatically test the different components of an application has become an essential prerequisite in many software development teams.

In this context, this report describes the development of an application for the automatic execution of integration tests. The application was developed during a data engineering internship at Xing, a very well-established German career-oriented professional networking platform. At the time of writing Xing counts more than 19 million users, most of them from Germany, Austria, and Switzerland.

The internship project was carried out in a team focused on data engineering projects and, following the Kaban methodology, an application was developed to automatically perform integration tests on the different components involved in the creation of a type of update on the platform. The application was also coupled to the tool used by the team in the adopted software development continuous integration and delivery practices

This report describes the developed project, which successfully achieved the proposed objectives, and delivered as a final product an application that will serve as a framework to perform integration tests, in an automated way, in the data pipelines for the creation of updates on the platform.

KEYWORDS

Integration tests; CI/CD; Data pipeline; Big data; Software development

INDEX

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
CI/CD	Continuous Integration and Continuous Delivery
DACH	German-speaking Europe: Deutschland (Germany), Austria, and Confoederatio Helvetica (Switzerland)
DevOps	Combination of Software Development and Operations practices and tools
ETL	Extract, Transform and Load
QA	Quality Assurance
PR	Pull Request
OS	Operating system

1. INTRODUCTION

The current transition from building applications based on a monolithic architecture hosted on a single server, to modern design patterns with multiple microservices, packaged up into orchestrated containers, and hosted in a distributed cloud environment, has been significantly increasing the complexity of modern software development (Carey, 2021). As new technologies continue to surge, especially with the advances in the uses of Big Data, many applications are now built using various languages and tools, and are expected to process real-time data, as a prerequisite for the success of a company's business.

This complexity oftentimes gets in the way of delivering value to customers, and as the adoption of modern agile methodologies in software development enforces the delivery of new versions of the software more rapidly, programmers now embrace development practices for continuous integration and delivery, intending to automate the flow of more reliable software updates. Techniques and tools for automating tests, to detect errors quickly, are an important part of the cycles of modern software production.

With every new version of a software product, new features are added. And when new features are added, the program becomes more complex, and consequently, it becomes harder to add new features without breaking anything. A piece of software's inevitable growth in complexity increases its development costs because it takes more time to properly test and fix the found defects, making the software more difficult and time-consuming to maintain (Arnon, 2018).

Software manufacturers know that testing is an essential prerequisite for the successful implementation of a software product and are now allocating more time and resources to testing technologies. As the cost of manual testing is oftentimes as expensive, difficult, and time-consuming as developing the product itself, the use of an automated testing tool can reduce the cost and improve the efficiency and effectiveness of software testing (Li & Wu, 2004; Tassey, 2002).

A test automation tool affects and serves nearly all stakeholders of a software development organization, including project managers, testers, DevOps people, software architects, and not only the developers. However, as every software organization and project is different, adopting practices, tools, and techniques that don't fit the team's skills or needs can cause failures in the automation project (Arnon, 2018).

For these reasons, the internship project reported here had the objective of developing a specific tool for the automatic execution of integration tests, developed utilizing the technologies used by the company, and with the purpose of verifying the compliance of various software components used in one of the data processing pipelines from which the team was responsible. This testing tool was also created with the aim of being integrated into CI/CD pipelines, facilitating the development, verification, and delivery of new software features to production.

1.1. Company Overview

Xing is a German career-oriented social networking platform on which businesses and professionals can interact with one another. It was founded in Hamburg in 2003 and is primarily focused on the German-speaking market (Germany, Austria, and Switzerland). With offices located in 5 different counties in Europe, Xing counted more than 19 million registered users in 2021[1].

Xing is one of the most trusted and popular professional social networking platforms in the DACH region, and professionals from all kinds of different industries can come together, find jobs, colleagues, cooperation partners, new assignments, experts, and generate business ideas.

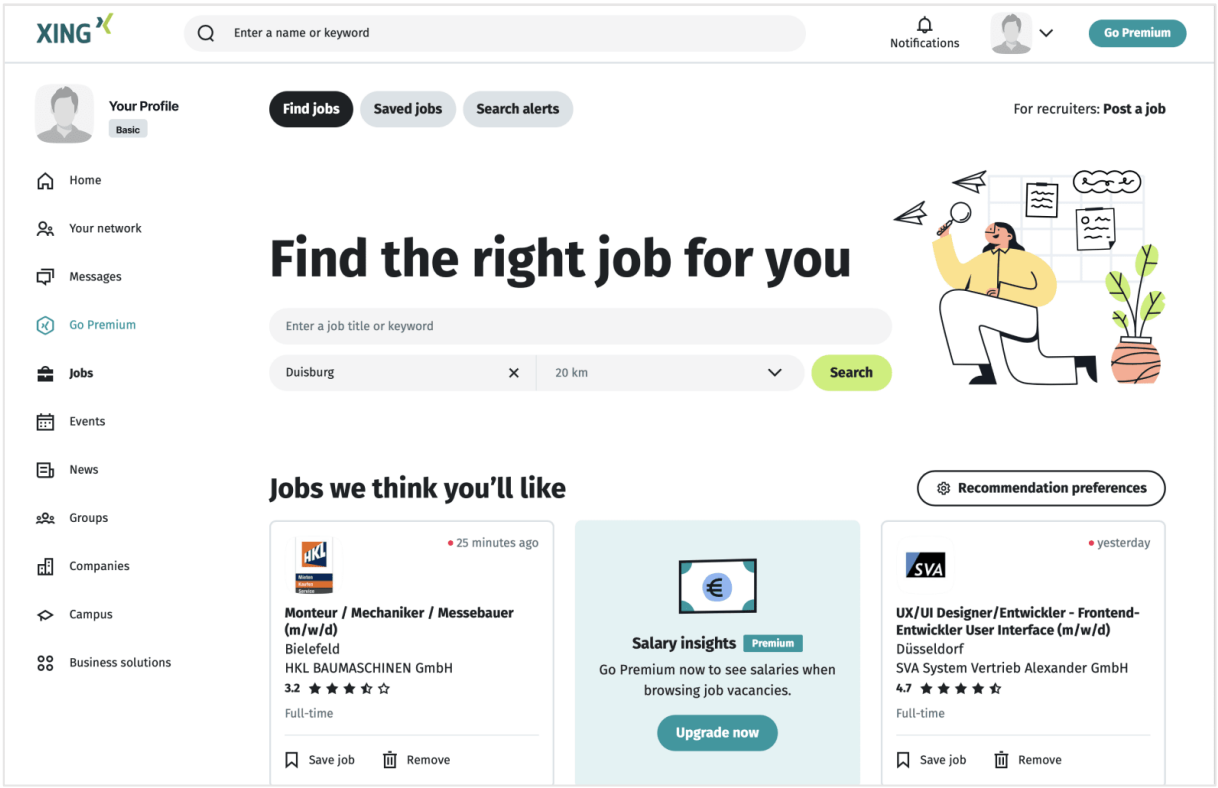

Figure 1.1 – Example of the Xing platform page

1.2. Internship Overview

The data engineering internship program lasted a total of 1 year, and took place from February 15, 2021, to February 15, 2022. From this period, the last 3 months were fully allocated to the development of the internship project. The internship was conducted in a team composed of three data engineers, two data scientists, one quality assurance engineer, a project owner, and a time leader.

[1] https://www.new-work.se/en/newsroom/press-releases/2020-xing-continues-growth-course-with-19-million-members-in-german-speaking-countries

The team was responsible for many data-intense microservices, and their corresponding data products, in the platform. The main tasks were developing and maintaining new functionalities in the different data streaming applications responsible for processing and generating the different types of updates on the platform, updating and populating the index used for user research, and improvements in the user recommendation system.

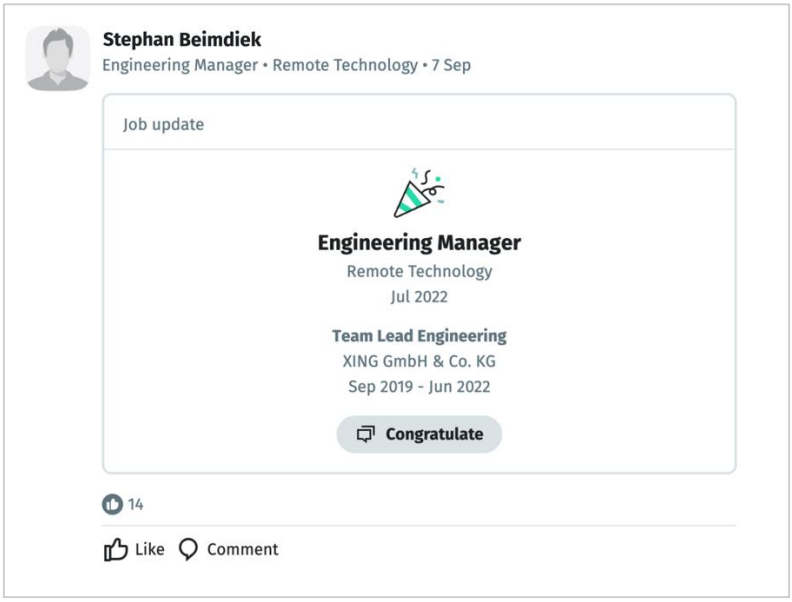

Figure 1.2 – Example of a user update card in the platform

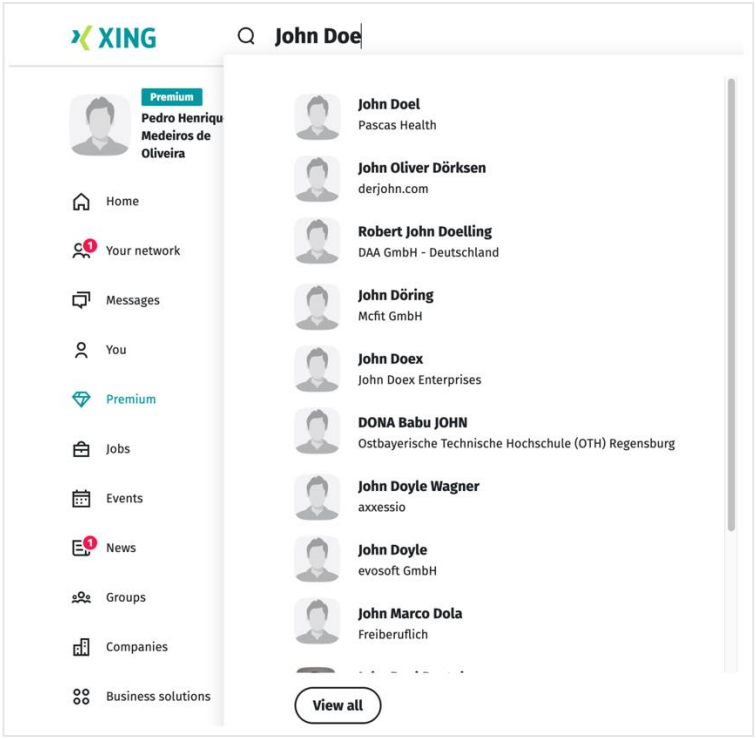

Figure 1.3 – Example of user search in the platform

1.3. Internship Project Objectives

The scope of the internship project was the development of a tool to run integration tests in one of the data pipelines responsible for creating updates on the platform. Among the various types of different updates that exist, for reasons of complexity, the project focused on testing the integration of the different modules in the data pipeline for the creation of Company Anniversary updates.

Company Anniversary type updates are generated on the platform when a user completes 1, 3 or 5 years working for a company. When an update is created, a card is generated in the feed of the user's friends who completed the anniversary of working at that company. The following figure illustrates a Company Anniversary Update card

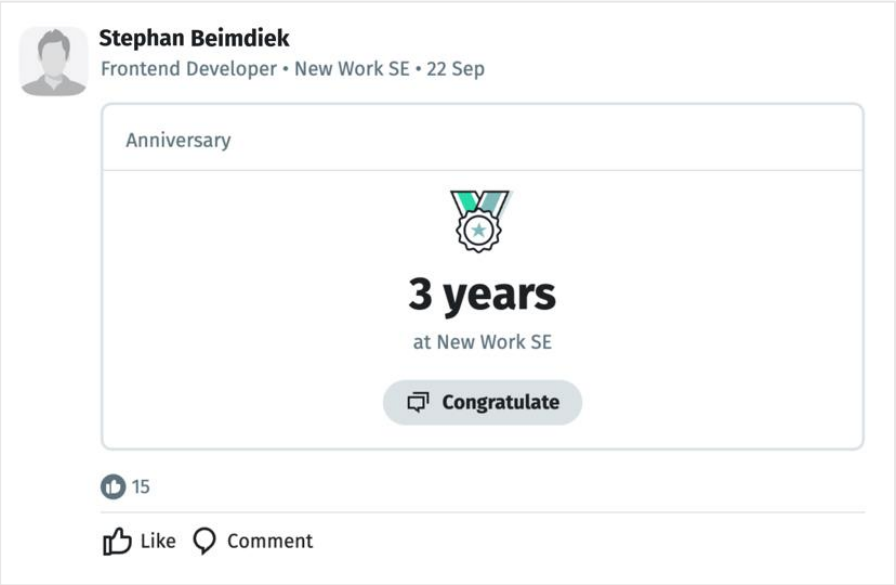

Figure 1.4 – Example of a Company Anniversary update card

The objective was to create a modular project that could be expanded over time, to cover all the test cases of the various pipelines of the different types of updates existing on the platform. It was also essential that the project could be compatible with the integration and continuous development practices adopted by the team's engineers.

2. PROBLEM DESCRIPTION

Usually, more than 8.000 Company Anniversary updates are processed every day. For each of these updates created, cards are, by default, shown in the feeds of all friends of the users for whom the updates were generated. This means that the total number of cards displayed on the platform is equivalent to the number of updates created times the number of users' friends.

The processing of this type of update involves a flow of data that passes through several applications until the final card is shown in the users' feed. At each stage of the pipeline, the data is processed by specific applications, and the processing is done according to the business logic the respective applications are responsible for.

As it is a complex process that involves different applications, logics, and technologies, and which also reaches a significant number of users, any error that is mistakenly introduced in any component of the pipeline can possibly generate problems that directly affect thousands of users in the platform. Possible bugs could be updates improperly created for users who have disabled this functionality from their profiles, updates created with incorrect data, or even updates not created at all.

The main applications involved in this data pipeline are covered by unit tests. However, unit tests alone can only verify the operation of specific components in isolation, not the integration of the results from the various parts and components involved in the pipeline. It is possible that bugs go undetected by unit tests, and that they cause serious problems in the production environment. This should be avoided and was the main reason for the development of this project.

2.1. COMPANY ANNIVERSARY UPDATES PIPELINE

The Company Anniversary Updates pipeline starts with a Scala application that runs a Spark job every day during the first hours of the morning. Initially, the batch job collects data from Hive tables from all users of the platform and executes a series of computations to validate and calculate which users have a company birthday on that date. Then, for each one of those users, a message is sent to a Kafka topic. This workflow is coordinated with the use of Apache Oozie.

The picture below shows the number of Company Anniversary Kafka messages processed in a day.

Figure 2.1 – Company Anniversary updates processed in a day

Subsequently, another application consumes and processes the messages sent to the Kafka topic in real time. This application is also developed in Scala and uses Akka Streams for the asynchronous processing of streaming data. It is in this application where users' privacy settings are checked, and whether they are blacklisted or not. If all validations are successful, the information about the update is stored in a MySQL table and sent in an AMQP message.

Client applications, belonging to other teams in the company, then consume these AMQP messages and are responsible for all processing until the creation of Company Anniversary Update cards on the platform. When a card is created, its data is stored in databases, and it is from this data that it is possible to verify its main information.

As can be seen in the following picture, the integration tests of this internship project covered only a part of the data pipeline. The choice of the tested components is due to the fact that it is in these components where the most complex logic occurs, and also where the most frequent implementations are inserted.

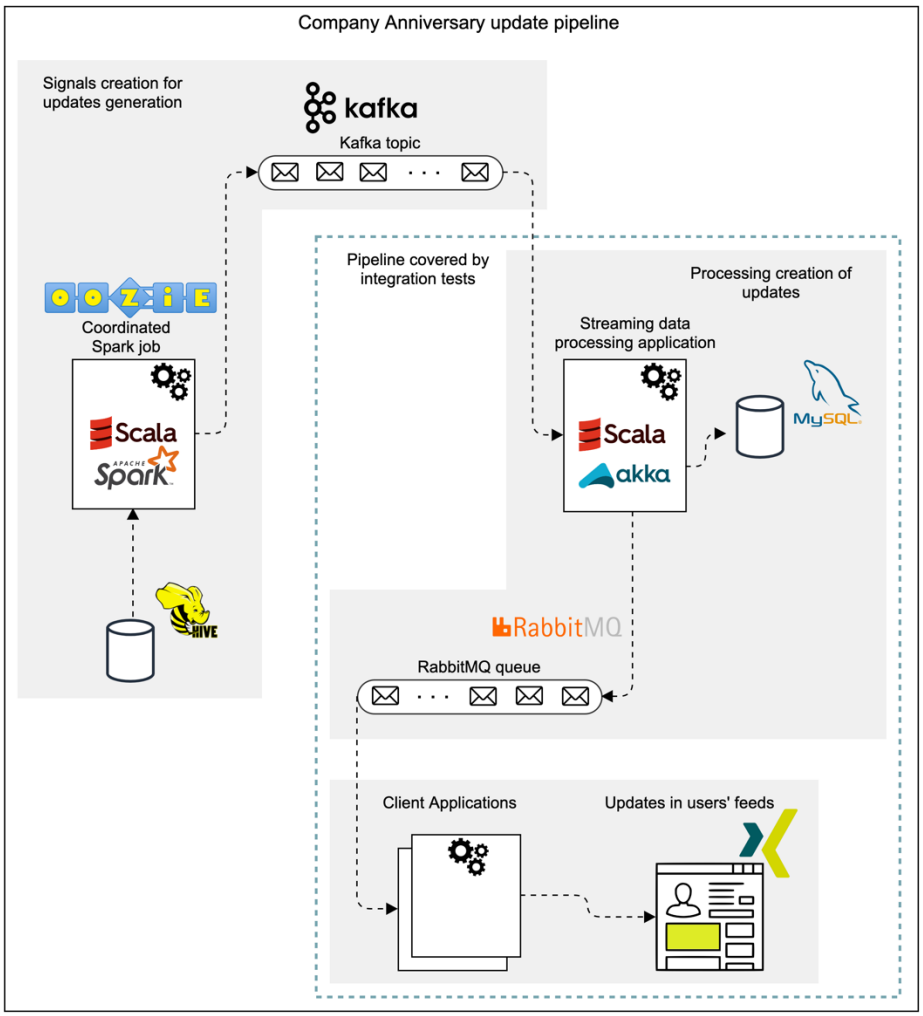

Figure 2.2 – Company Anniversary updates pipeline

3. BACKGROUND

3.1. SOFTWARE TESTING

Software testing is a series of processes designed to ensure computer code does what it was designed to do. It is a method to check whether the actual software product matches the expected requirements without doing anything unintended. Software should be predictable and consistent, presenting no surprises to users.

Testing software can be viewed as the destructive process of trying to find the errors in a program (whose presence is assumed). It involves the execution of software/system components using manual or automated tools to verify one or more properties of interest. The objective of software testing is to identify gaps, errors, or missing requirements in contrast to expected requirements.

Testing a program adds value to it by raising its quality and reliability by finding and removing errors. A successful test suite is one that furthers progress in this direction by a diligent exploration for errors, with the purpose of establishing some degree of confidence that the program does what it is supposed to do and does not what it is not supposed to do (Meyers et al., 2012).

Software testing is critical because it can identify errors or bugs in the software early so that they can be solved before the delivery of the software product. Properly tested code ensures security, reliability, and high performance which further results in cost-effectiveness, time saving, and customer satisfaction (Hamilton, 2022).

3.1.1. Unit tests

Unit testing is a software development process in which the smallest testable parts of a software application, also called units, are independently and individually tested. Unit tests focus on testing the individual modules in total isolation against specifications generated as part of the system design. it is an important part of the development process because if done correctly, it can help detect early flaws in code which may be more difficult to find in later testing stages (Mili & Tchier, 2015).

This testing methodology is done during the development process by the software developers and enables them to think through the design of the application and what has to be done before writing code. Testing code early on can help to define what each part of the code is responsible for and facilitate developers to stay focused and create better designs.

Unit tests are also the first level of software testing, which are performed in earlier stages than the other testing methods such as integration tests. This testing method normally comprises three stages: Arrange, Act, and Assert. The first step is setting the testing units and preparing the test's prerequisites. The second step is when the script is executed, and the code is tested. The final step is for verifying the results.

3.1.2. Integration tests

Integration testing is a level of software testing where the different units, components, or modules of a software application are combined and tested as a group. It tests the interface between the modules, to determine its correctness and expose faults in the interaction between the integrated units. Once all modules have been unit-tested, integration testing is performed.

Once all units have been developed, the system is put together according to its design and tested to ensure that the integration of the units meets the system-wide specification. Integration testing aims to diagnose faults in the design of the system, estimate the reliability of the software product, or remove enough faults that the system's reliability is raised beyond the required threshold (Mili & Tchier, 2015).

3.2. CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY

Continuous Integration and Continuous Delivery is a software engineering practice that automates application build, testing, deployment, as well as infrastructure provisioning. It is a best practice in agile methodologies that lets software development teams focus on meeting business requirements while ensuring code quality and software security. With CI/CD, code changes go through an automated pipeline that handles the repetitive build, test, and deployment tasks, and alerts if an issue is found(Sacolick, 2022) .

A CI/CD pipeline is an automated process composed of a series of steps that must be performed to deliver a new version of software. It compiles incremental code changes by developers and packages them into software artifacts. Automated testing verifies the integrity of the software and automated deployment pushes it immediately into production. CI/CD pipelines enforce the agile concept of development in small interactions, create a fast feedback loop for developers and enable them to deliver higher-quality code faster.

The Continuous Integration part of a CI/CD pipeline refers to the development practice where team members use a version control system to integrate their work frequently to the same location, such as a main branch. Each integration is verified by an automated build and tested to detect integration errors as fast as possible. This process can significantly reduce integration problems and allows teams to develop cohesive software more rapidly (Fowler, 2006).

Whereas Continuous Integration is focused on automatically building and testing code, Continuous Delivery is focused on automating the entire software release process. It is a software development practice responsible for deploying all types of changes, including new features, bug fixes, experiments, and configuration changes, into production in an automated manner. It represents the final stages of a CI/CD pipeline and turns integrated code into production software (Humble & Farley, 2010).

3.3. VERSION CONTROL

Version control is the practice of tracking and managing changes to software code. A tool that manages and tracks different versions of software source code over time is referred to as a version control

system. A version control system enables teams to develop and maintain a repository of content, provide access to historical editions, and record all changes in the code by keeping track of every modification made. If an error was introduced, developers can turn back and compare earlier versions of the code to fix the mistake (Loeliger & McCullough, 2012).

3.3.1. Git

Git is a free and open-source distributed version control system that is used to track changes in the source code, enabling multiple developers to work on non-linear development. Git enables software development teams to work on many different copies of a project codebase independently. These copies are called “branches” and can be created, merged, and deleted, allowing teams to experiment and test, with little compute cost, before merging the introduced changes into the main branch.

Git is software that runs locally, and the files and their history are stored on Git repositories. A Git repository is simply a database containing all information needed to retain and manage the revisions and history of a project. In Git, a repository contains a complete copy of the entire project throughout its lifetime, and it maintains a set of configuration values within each repository (Loeliger & McCullough, 2012).

3.3.2. GitHub

GitHub is a cloud-based platform where developers can upload a copy of a Git repository. But more than just uploading to Git repositories, it enables collaboration across teams on different projects. GitHub provides a centralized location to share the repository, a web-based interface to view it, and features like Pull Requests, Forking, Issues, Projects, and GitHub Wikis that allows team members to specify, discuss and review changes to the code more effectively

In GitHub, whenever a developer is building a new piece of functionality from a repository code, it will create a branch to work on it. A branch is a version of the repository, also described as an independent series of commits. Commits are like a snapshot of the entire repository at that point in time. When working on implementing a new feature on a branch, developers update their repositories gradually saving the changes by committing them (Beer, 2018).

When developers want to introduce changes to a repository, they normally do so by creating a pull request. A GitHub pull request is a mechanism to notify team members a feature has been added from a forked branch. Once a pull request is opened, collaborators can discuss and review potential changes and add follow-up commits before the introduced changes are merged into the base branch.

The next figure illustrates the described series of actions:

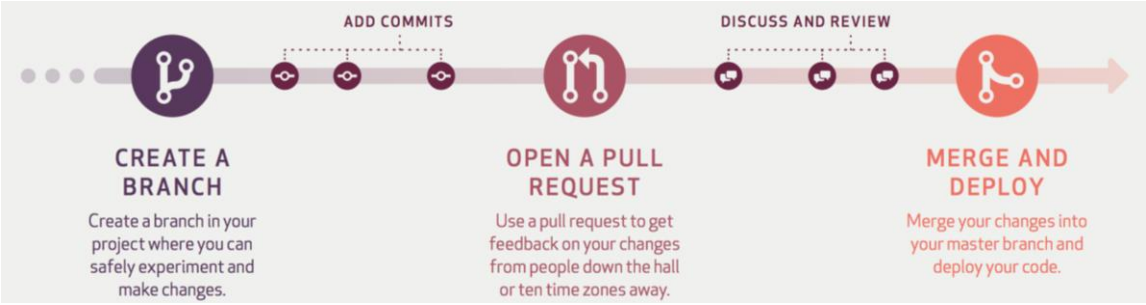

Figure 3.1 – GitHub pull request flow

3.4. CONTAINERIZED APPLICATIONS

Software containerization is a form of virtualization, where an application is packaged up with all its dependencies so that it can run consistently and uniformly on any infrastructure. Containerized applications run in isolated runtime environments called containers. Containers encapsulate an application with all its dependencies, including system libraries, configuration files, and binaries.

A containerized application is essentially a fully packaged computing environment since everything the applications need to run is present and insolated in its container. This form of packaging makes the application portable by enabling it to run consistently across various types of infrastructures. With containerization technologies, software developers can create applications and run them on different setups and hosting environments (Datadog, 2021).

Containerizing an application addresses the problem software developers often face to make applications run consistently across different hosting environments. With traditional methods, an application is developed in a specific computing environment, which when transferred to a new location, frequently results in errors caused by the differences between the systems. Containerization allows applications to be written once and run almost anywhere (IBM Cloud Education, 2019).

Unlike virtual machines, containers don't include their own operating systems. Multiple containerized applications running on the same host machine share the same OS provided by the system. Not needing to attach an OS to the application makes containers exceptionally lightweight and very fast to run. Users can have multiple isolated applications in separate containers, and to scale an application, more instances of containers can be added in a fraction of a second.

3.4.1. Docker

Docker is an open-source platform for building, deploying, running, updating, and managing software in packages called containers on servers and the cloud. Docker is a container technology that creates a complete solution for the creation and distribution of containers. The Docker platform is composed of two distinct components: the Docker Hub, a service for distributing containers; and the Docker Engine, which is responsible for creating and running containers (Mouat, 2016).

A Docker container is a standardized unit of software that encapsulates application source code with the operating system libraries and dependencies so that the application runs reliably and quickly one different computing environments. A Docker container image is a lightweight, standalone, executable package software that includes everything needed to run the application.

A Docker image is a read-only template, made up of a collection of files, that contains a set of instructions for creating a container that can run on the Docker platform. Docker container images become containers at runtime when they run on Docker Engine, isolating the software instance from its environment and ensuring that it performs uniformly regardless of the differences between the hosting systems (Miell & Sayers, 2019).

3.4.2. Kubernetes

Kubernetes is an open-source container orchestration platform for automating the deployment, scaling, and management of containerized applications. As software applications grow to span several containers deployed across multiple servers, operating them becomes more difficult and complex. Kubernetes manages this complexity by providing an open-source API to control where and how those constrainers will run.

The platform offers a range of solutions to create applications that are easily controlled and deployed, automating operational tasks of container management and includes native commands for deploying applications, monitoring, rolling out changes to the applications, and scaling applications up and down.

Kubernetes orchestrates clusters of virtual machines and schedules the containers to run on those virtual machines. It automatically manages service discovery, tracks source allocation, incorporates load balancing, and scales based on computing utilization. Kubernetes checks the health of individual resources, automatically restarting or replicating containers when needed (Microsoft Azure, 2019).

3.5. BIGDATA TECHNOLOGIES

The term “big data” refers to extensive data collections and volumes that are too enormous or complex to be dealt with by traditional data processing software. Big data is a combination of structured, semi-structured, or unstructured data collected by organizations, that contains larger variety, arriving in increasing volumes and with greater velocity from different sources.

Big data is typically measured in terms of terabytes or petabytes, and these large volumes of data can be used to solve business problems that were previously unable to be tackled. The vast collection of data grows at an exponential pace over time and has led to the development of numerous big data-based technologies capable of enough to store, process, analyze and extract information from an immense set of extremely complex data structures. Big data technologies are focused on analyzing and handling large amounts of real-time and batch-related data (Perez, 2020 B.C.E.).

3.5.1. Apache Hadoop

Apache Hadoop is an open-source software framework with utilities for the distributed processing of massive amounts of data across a network of many computers through programming models. It is designed to scale up from a single machine to thousands of servers, each offering storage and computation capabilities. There are three main components in the Apache Hadoop framework: Hadoop Distributed File System (HDFS), MapReduce, and YARN (White, 2015).

HDFS is a distributed storage solution used by Hadoop applications. It employs a cluster of machines to implement a distributed file system that provides high-performance access to data across highly scalable clusters. Hadoops splits and replicates files into large blocks and distributes them across machines in a cluster. Replication in HDFS increases data availability by reducing the chances of failures and data loss (Kumar & Shindgikar, 2018).

MapReduce is a software programming model and framework used to write applications capable of processing large sets of data in parallel stored in HDFS. A MapReduce program works in two phases: Map and Reduce. Map tasks are responsible for splitting and mapping data, converting it into another set of data. Reduce tasks take the output from a map as an input, and shuffle and reduce the data by performing summary operations. The most important advantage of MapReduce is its scalability, making it easy to write an application and scale it to run over several machines in a cluster (Taylor, 2022).

YARN stands for Yet Another Resource Negotiator and is the job scheduling and resource management technology in the Hadoop distributed framework. It is responsible for dynamically allocating system resources to the multiple application running on the Hadoop cluster, a capability designed to improve resource utilization and application performance. YARN also supports scheduling methods to be executed on the different cluster nodes. YARN enables applications to work with massive amounts of data in an efficient manner (Stedman, 2020).

3.5.2. Apache Hive

Apache Hive is an open-source data warehouse that enables reading, writing, and managing large data set files stored in the Hadoop Distributed File System or in other distributed storage systems. Hive provides a Hive query language (HiveQL), which is quite similar to standard SQL, that converts its queries into a series of jobs that execute on a Hadoop cluster through MapReduce. The underlying MapReduce interface with HDFS is difficult to program directly, and Hive is designed to make it easier by enabling developers to simpler queries that are translated into map and reduce functions (Cloudera, 2010).

Hive provides a centralized metadata store, enabling programmers to apply a table structure to massive amounts of unstructured data. Hive metastore is simply a relational database, and it stores information about tables and schemas to easily query large volumes of distributed data. It acts as a central repository for Hive metadata, as all of its stored information becomes part of the Hive architecture (Rutherglen et al., 2012).

3.5.3. Apache Spark

Apache Spark is a unified computing engine platform and a set of libraries for parallel data processing on computer clusters, designed to be fast and general-purpose. One of the main features Spark offers for speed is the ability to run computations in memory, extending the popular MapReduce model to efficiently support more types of computations. Spark covers a wide range of workloads that previously required separated distributed systems, making it easy and inexpensive to combine different data processing types (Chambers & Zaharia, 2018).

Spark is designed to be highly accessible, offering simple APIs in multiple widely used programming languages, such as Python, Scala, and Java. It includes rich built-in libraries for diverse data processing tasks and runs anywhere from a single machine to a cluster of thousands of servers. This makes it an appropriate system for big data processing or incredibly large scale (Chambers & Zaharia, 2018; Karau et al., 2015).

Although Spark supports multiple workload types, it is widely used for its fast batch computing capabilities to process large amounts of data through parallelized operations. A Spark job is a parallel computation action for data processing consisting of a sequence of stages that are composed of tasks. One example of a Spark job is an ETL data processing pipeline (Grah, 2021).

3.5.4. Apache Kafka

Apache Kafka is a streaming platform: a messaging system that allows applications to publish and subscribe to streams of data, store and process them in real time. It is primarily used to build real-time streaming data pipelines and applications that interact with the data streams. It combines messaging, storage, and stream processing to enable storage and analysis of both historical and real-time data (Amazon Web Services, 2019).

As streaming data is continuously generated by several data sources, which typically produce data simultaneously, the Kafka platform is able to handle this constant influx of data and process the data sequentially and incrementally. The Kafka API provides functions to publish and subscribe to streams of records, process the influx of real-time data, and store streams of records in the order in which they were generated.

The streams of messages that are part of a specific category are referred to as a Kafka topic. In Kafka, the data is stored in the form of topics. Kafka topics are divided into a configurable number of parts, known as partitions. Producer applications publish their data to Kafka topics, and consumer applications read data from these topics. Topic partitions provide scalability to the system since they allow multiple consumers to read data from the same topic in parallel.

Kafka works in a modern distributed system that runs as a cluster and can scale to handle all the applications in even the most complex use cases. Instead of running multiple individual messaging brokers, it provides a central platform that can scale elastically to handle all the streams of data from a pipeline. Kafka is also a true storage system capable of storing message data for as long as needed. This provides huge benefits in using it as a connecting layer since data delivery is guaranteed: its data is replicated and persistent (Narkhede et al., 2017).

4. METHODOLOGY

As the project described in this report was developed during the data engineering internship at Xing in a fully functional team of engineers, its development was also planned and executed according to the practices adopted by the team. During the project development period, the Kaban methodology was used in the implementation of agile and DevOps software practices in the team.

4.1. KANBAN

Kanban is a popular lean workflow management system for defining, managing and improving services that deliver knowledge work. It was developed in the 1940s by an industrial engineer at Toyota as a scheduling system for just-in-time manufacturing, and Its name comes from two Japanese words, “Kan” meaning sign, and “ban” meaning board (Atlassian, 2016).

As its name suggests, the work in Kanban is presented on boards and must be visually depicted. The process must be shown step by step using visual cues that make each task clearly identifiable. The main idea is to easily show what each step is, what expectations are, and who will take care of the tasks (Halton, 2022).

The Kanban methodology focuses on visualizing the work, to maximize efficiency by making incremental changes and increasing delivery speed. It builds a continuous delivery model where teams can release value as soon as they are ready. Its practices revolve around the kanban board, an agile project management tool that is used in conjunction with kanban cards.

4.1.1. Kanban board

A kanban board consists of columns, each one representing an activity that together compose a workflow. The board columns indicate the stage of a workflow process. Kanban cards, also called tickets, are visual elements that represent the work item and communicate its status as it moves through the workflow. Kanban teams write their tasks onto cards, usually one per card. Once on the board, they help stakeholders and team members easily understand what the team is working on (Halton, 2022).

A typical kanban board also consists of one or more swimlanes. Kanban swimlanes are horizontal lines that divide the board into sections. They are normally used to separate different work types on the same board, representing the processes that happen concurrently within a team. Swimlanes groups related tasks to create a more organized project.

The Kanban board's main function is to ensure the team’s work is visualized, their workflow is standardized, and all dependencies and blockers are immediately identified and resolved. As the kanban methodology relies upon full transparency of work, the kanban board is used as the single source of truth for a team’s work.

4.1.2. System benefits

Kanban proposes an approach for managing the flow of work with an emphasis on continuous improvement without overburdening the development team. The work is pulled into the system when the team has the capacity for it, rather than tasks being assigned from the top (Siderova, 2018).

The methodology provides planning flexibility by instructing teams to only focus on the work that is actively in progress. Once a work item is completed, the next most important work item is pulled from the top of the backlog. It also increases efficiency and productivity by shortening the lifecycle of a task, so bottlenecks can be quickly identified and addressed, ensuring that the most relevant tasks are prioritized.

Different from other agile methodologies, Kanban focuses on helping a team improve the way it develops software. A team that uses Kanban has a clear picture of what actions they use to build software, how they interact with the rest of the company, where they run into inefficiency, and how to improve over time by removing the root cause of these inefficiencies (Stellman & Greene, 2014).

4.2. PROJECT DEVELOPMENT

As the project developed was considered a product owned by the team, it was developed following the same agile practices adopted by teammates. Initially, after brainstorming sessions with the team's Product Owner and engineers, the project was conceptualized, its main attributes defined, and its execution divided into smaller tasks. These units of work were then turned into tickets, and a repository was created on GitHub in the team namespace, where the project was developed. All implementations introduced by the tickets were added to the project code through pull requests.

As per team practice, before a ticket could be added to the Kanban board, it must first go through a phase called "refinement". In this session, the ticket is presented and explained to all team members, who in turn can ask questions and make suggestions. After the explanation, teammates estimate a difficulty level for the implementation presented in the ticket. The estimated level is only used to gauge the time required to complete the task.

In the Kanban board used for the team, a new swimlane was added dedicated to the tasks related to the internship project. Thus, the project tickets were grouped into a single section of the board, in a similar way that tickets belonging to other projects for which the team was responsible were also organized into different swimlanes. After all the tickets created for the project were properly "refined", they were added to the first column of the board called “To do”.

The tickets were then gradually pulled from the “To do” column and worked on, following the flow of the kanban board. The second column of the board, labeled "In progress", contained the tickets that were currently being implemented. When the implementation was complete, a ticket was moved to the third column, "Ready for review", where it stayed until one of the team members picked it up.

After being taken by a teammate, the ticket would then be moved to the fourth column, called "In review". At this stage, the implementation of the ticket was reviewed and, if necessary, suggestions for improvements were made. After the review remarks were addressed, the ticket was moved to the

next column, "Ready for QA". The ticket stayed in this column until the team's QA Engineer was available to pick it up, at which point the ticket was then moved to the next column labeled "In QA". In this step, the implementation added by the ticket was tested, and depending on the results, the ticket was approved or not. If the ticket was not approved, the appropriate changes would have to be made. Afterward, the ticket could be moved to the following column, called "Ready to merge".

Since most of the tickets for this project represented implementations created from pull requests on GitHub, after a ticket was approved, the pull request was merged into the main branch of the project's code. The ticket was then closed, thus ending its flow through the board. The next figure illustrates the structure of the Kanban board used for the internship project.

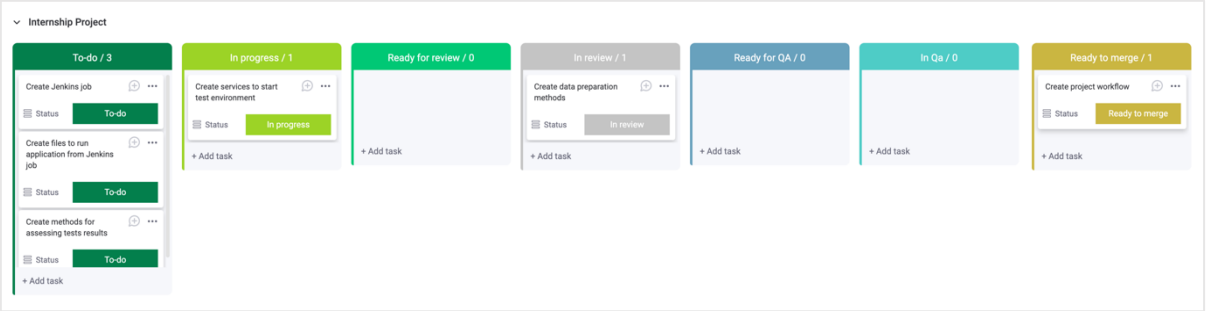

Figure 4.1 – Example of Kanban board for the internship project

Following the practices of the kanban agile software development methodology, the project was built by receiving input from the team members in the conceptualization of the tickets, and in the tasks of reviewing the code and testing the implemented features. The project was developed in a fast and organized way, delivering a quality product.

5. PROJECT DESCRIPTION

To perform the integration tests and verify how the multiple components that belong to the data workflows responsible for creating updates on the platform, an application was developed using the Scala programming language. The choice of the programming language was based on uniformity, as most of the applications for which the internship team was responsible were also developed in Scala. Using the same programming language also ensured that libraries and frameworks used in other projects could be reused, as well as team members' familiarity with understanding the code.

Due to the time constraints allocated to the development of the project, as well as the different levels of complexity in processing the different updates, the project only covered the data workflow involved in Company Anniversary updates. However, as the final objective of the application is to perform integration tests for all types of platform updates, it was developed in a modular way, so that later it will be easier to add new components and expand the test coverage.

In the application, several services were developed, each with different methods responsible for starting the test environment, creating the specific users involved in each test, as well as accessing and validating the test results. In this way, through these methods, within the test environment, the application can interact with other applications, services, and databases on the platform.

Initially, when the application tests are triggered, methods are called to start the environment where the tests will take place. This test environment is created from an internal service that allows developers to emulate a dedicated copy of the platform in a few minutes. The test environments created reside in dedicated Kubernetes clusters, where the developer has access to the different services that make up the platform. In this way, it is possible to change the Docker images used to create the containers that represent the applications within the platform.

Then, after the test environment has been initialized, another sequence of methods is called to create users with specific data for each test. After the users have been created, Kafka messages are sent to the topic from which the first application in the update creation workflow consumes, thus initiating the processing of updates. It is in this part where the functioning of the applications is really tested, as it is the results of these processes that will be verified in the tests.

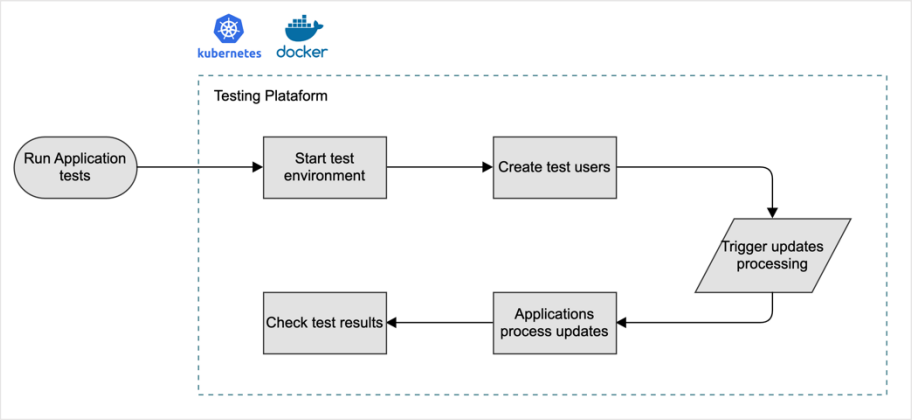

Figure 5.1 – Integration tests process

5.1. TEST CASES

At the beginning of the pipeline covered by the integration tests, messages with information about users completing their company anniversaries are sent to a Kafka topic. These messages are consumed and processed in parallel, according to the flowchart from the figure below, which illustrates the logic of processing these messages by the first application tested in the pipeline.

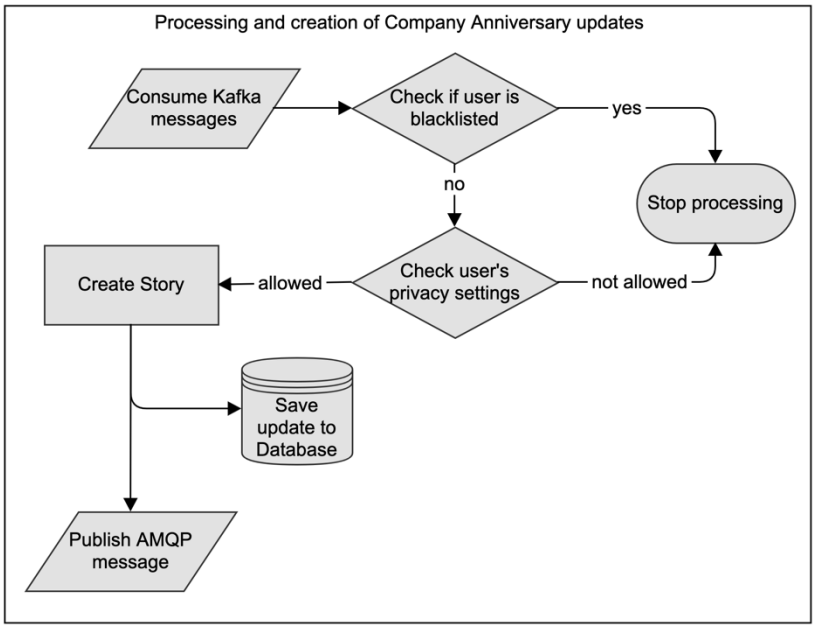

Figure 5.2 – Company Anniversary updates verification logic

When a message is consumed, a series of functions are applied using the data contained in its payload. Initially, it is checked whether the user is blacklisted or not. This verification is performed using the user's profile ID, through a method that makes an HTTP request and returns this information. If the user is blacklisted, the message processing is stopped.

Next, the user's privacy settings are checked. This verification is also done through a method that makes an HTTP request and returns this data. On the platform, users have the option to disable certain types of updates, and if they make this change in the privacy settings of their accounts, deactivated updates should not be produced. If the user has disabled Company Anniversary updates, the message processing is interrupted.

These two most important verifications create the three possible cases for the creation of Company Anniversary updates that are tested, as shown in the next figure.

Test Case	Description
Test Case 1	The user is not blacklisted and does not have Company Anniversary updates disabled. In this case, the update must be created, and the cards displayed in the timeline of his friend.
Test Case 2	The user is blacklisted, and the update should not be created
Test Case 3	The user is not blacklisted but has disabled Company Anniversary updates. The update should not be created.

Figure 5.3 – Test cases

5.2. DATA PREPARATION

To test a program, it is important to execute it on all possible inputs or combinations of inputs and observe its behavior under these circumstances. When the developed program is launched, it starts a test environment that mimics the operational environment and creates and prepares the test data so that the integration tests can be executed. The data used is a representative set that ensures that all possible test cases are covered. It is at this stage when, in the program, a series of methods are called to create users with their respective specific data for each of the test cases. The image below illustrates some of the high-level methods used to create the test users.

```
private def createUser(): Future[Long] = {
  val userData = generateUserData()
  for {
    userId <- accountsService.createAccount(userData.active_email)
    _ <- accountsService.confirmAccount(userId)
    _ <- profilesService.nameUser(userId, userData.first_name, userData.last_name)
  } yield {
    waitForUserToBeCreated(configs.userCreationWaitTime)
    userId
  }
}
def createTwoBefriendedUsers: Future[(Long, Long)] = {
  for {
    userId1 <- createUser()
    userId2 <- createUser()
    _ <- createFriendship(userId1, userId2)
  } yield (userId1, userId2)
}
def createBlacklistedUser(): Future[Long] = {
  for {
    userId <- createUser()
    _ <- accountsService.blacklistUser(userId)
  } yield userId
}
def createNotAllowedPrivacySettingUser(requestBody: UpdatePrivacySettingsBody): Future[Long] = {
  for {
    userId <- createUser()
    _ <- activitiesService.updatePrivacySettings(userId, requestBody)
  } yield userId
}
```

Figure 5.4 – Example of methods for creating test users

5.2.1. Test case 1

For the first test case, two regular users are created and then befriended. The second user is created because since they are friends when the update is created for the first user, he should receive this update on his timeline. This verification will be done in the next steps. Then, a Kafka message is sent on behalf of the first user. This is the message responsible for triggering the processing of the update.

The sequence of steps illustrated in the image below represents the methods used to create the data for this case.

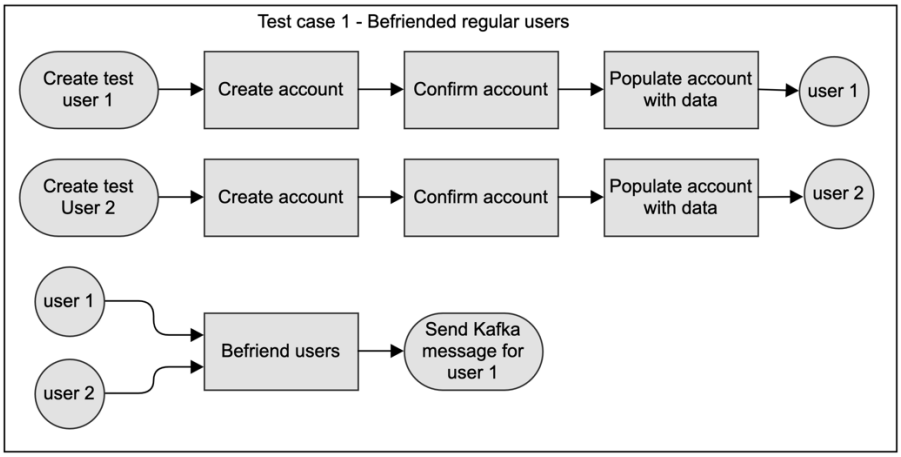

Figure 5.5 – Data preparation for Test Case 1

5.2.2. Test case 2

For the second case, only one user is created. A method responsible for blacklisting the user is called, and a Kafka message is sent to trigger the update processing. The image below illustrates the sequence of actions performed by methods to prepare the data for this case.

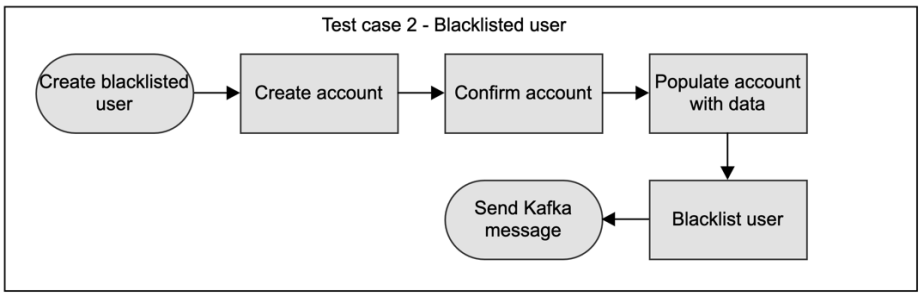

Figure 5.6 – Data preparation for Test Case 2

5.2.3. Test case 3

For the third case, similarly to Test Case 2, only one user is created. A method is called to update the user's privacy settings, disabling the creation of Company Anniversary updates. A Kafka message is then also sent to trigger the processing of the update.

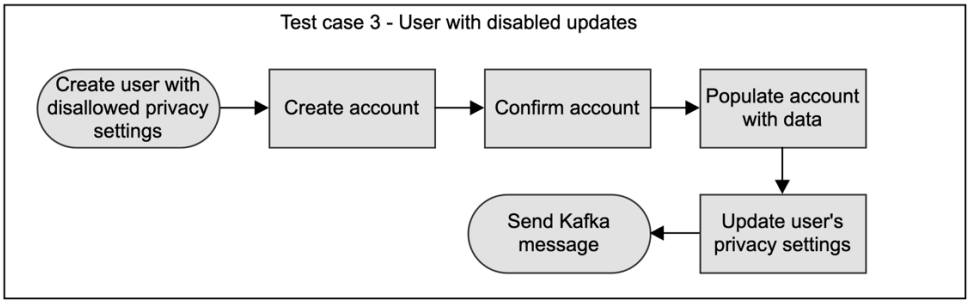

Figure 5.7 – Data preparation for Test Case 3

5.3. RESULTS ASSESSMENT

After the Kafka messages have been sent, in the previous phase of data preparation, they will be consumed by the respective applications that will process them to create the updates. After the updates are processed, the application verifies that the results are as expected. This verification is done through methods that make HTTP requests and return the data of the updates created for each user. The data retrieved from these updates is then checked, and if the results are correct, the test is successful.

For Test Case 1, it is verified that the updates were created correctly for the two befriended users. For Test Cases 2 and 3, it is checked instead if the updates were not created. The two figures below illustrate, respectively, the method used to verify if the update was created correctly or not, and the part of the program where the tests of the three cases are triggered and their results validated.

```
def newValidActivityIsCreated(activityOpt: Option[DiscoActivity], userId: Long): Boolean = {
  activityOpt.exists { activity =>
    val baseObjectUrn = extractBaseObjectUrn(activity.object_urn)
    val activityDate  = extractDate(activity.created_at)
    baseObjectUrn == "urn:x-xing:people_story_generator:stories:company_anniversary:" &&
    activityDate == LocalDate.now() &&
    activity.actor_id == userId
  }
}
```

Figure 5.8 – Method used to validate update

```
"CompanyAnniversaryTestsuite" should {
  "create activity for valid user" in {
    val kafkaPayload = CompanyAnniversary(userId1, 1, 2, 3, "Honeypot")
    sendKafkaRestMessage(kafkaPayload)

    val discoActivityUser1 = getLastDiscoActivity(userId1)
    val discoActivityUser2 = getLastDiscoActivity(userId2)

    newValidActivityIsCreated(discoActivityUser1, userId1) and newValidActivityIsCreated(discoActivityUser2, userId1)
  }
  "not create activity for blacklisted user" in {
    val kafkaPayload = CompanyAnniversary(blacklistedUserId, 1, 2, 3, "Xing")
    sendKafkaRestMessage(kafkaPayload)
    val discoActivity = getLastDiscoActivity(blacklistedUserId)

    !newValidActivityIsCreated(discoActivity, blacklistedUserId)
  }
  "not create activity for user with 'anniversary' privacy setting not allowed" in {
    val kafkaPayload = CompanyAnniversary(notAllowedPrivacySettingUserId, 1, 2, 3, "Kununu")
    sendKafkaRestMessage(kafkaPayload)

    val discoActivity = getLastDiscoActivity(notAllowedPrivacySettingUserId)

    !newValidActivityIsCreated(discoActivity, notAllowedPrivacySettingUserId)
  }
}
```

Figure 5.9 – Test cases verification

After running the tests, it is possible to check in the application logs which ones passed and which ones failed. In the example in the figure below, solely for demonstration reasons, two tests passed and one failed.

```
[info] CompanyAnniversaryTestSuiteSpec
[info] CompanyAnniversaryTestsuite should
[info]   + not create activity for blacklisted user
[info]   + not create activity for user with 'anniversary' privacy setting not allowed
[error]   x create activity for valid user
```

Figure 5.10 – Example of the test results from the application logs

5.4. CI/CD INTEGRATION

As a key component of the DevOps strategy adopted, the team uses the open-source automation server Jenkins to implement Continuous Integration and Continuous Delivery for the development of the projects it owns. With Jenkins, it is possible to create pipelines that execute a series of automated tests and builds to always check if the code is valid. Its use helps the team members test more often and deploy faster, and significantly simplifies the process of ensuring a high level of code quality and successful builds throughout all projects the team develops (Shain, 2021).

Although it is possible to run the application and execute the tests locally, the main benefit of this project is its integration with Jenkins. As part of the project, a Jenkins job was developed to enable the program to be triggered from a command written directly in a pull request on GitHub.

One of the features provided by Jenkins is the ability to connect and pull code from GitHub to trigger builds directly from a pull request. This integration is provided mainly by the Jenkins Git and GitHub plugins, and with the Jenkins GitHub integration, it is possible to use any file in the GitHub repository to trigger Jenkins jobs for every code commit.

To create the Jenkins job that triggers the integration tests by running the application, the repository URL of the project was used for the GitHub source code integration. This configuration was added to the build job, so Jenkins could pull the source code and perform operations on the files. To trigger the Jenkins job from a command on the pull request, a GitHub webhook was implemented to enable the integration between the tools.

With the job created, whenever the comment that triggers the job is added to a pull request, the Jenkins server pulls the latest code from the pull request and uses a file written in the bash programming language with a set of commands to run the application. After running the application, the test results are reported back to the pull request itself. The next picture illustrates the diagram with this connectivity setup between GitHub and Jenkins.

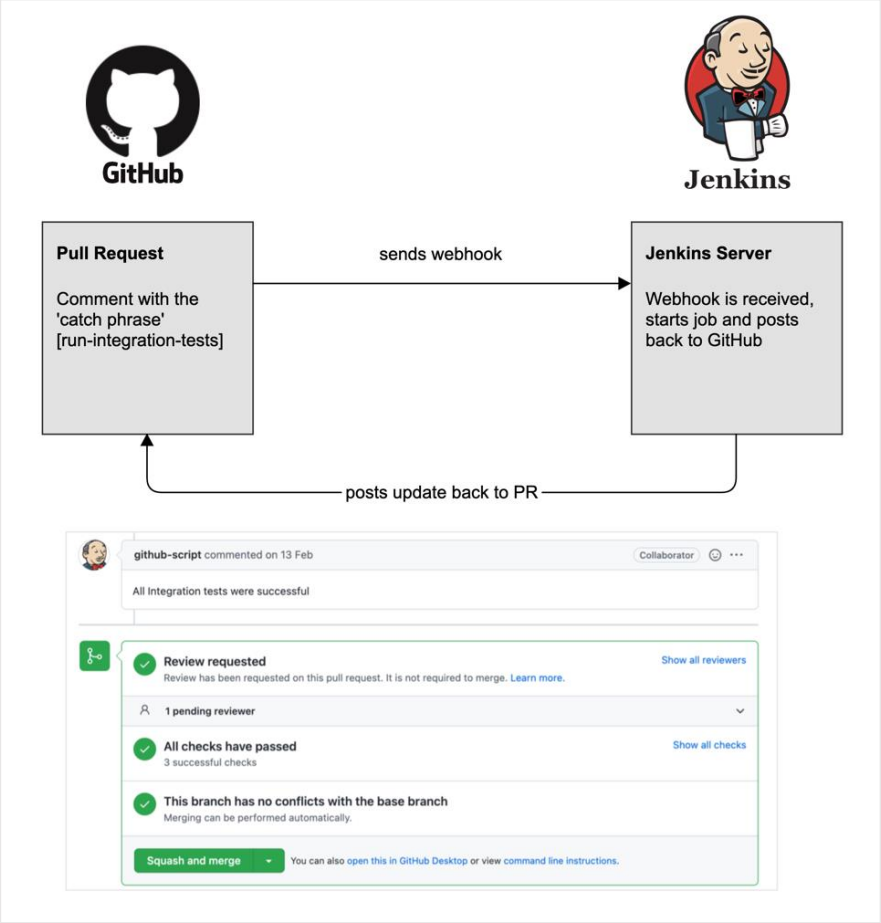

Figure 5.11 – Diagram of a pull request and Jenkins job integration

When running the integration tests from a PR, the objective is to verify that the version of the code present in the pull request does not have faults and that all tests pass. Therefore, in this case, when the testing application initially starts the test environment, it also updates the docker image of the tested application by the image generated from the last commit of the pull request it was triggered. Thus, in the test environment, the version of the application with the changes introduced in the pull request will be tested. The figure below illustrates the comment added to a pull request that triggered the integration tests and the generated message that reports the tests were successful.

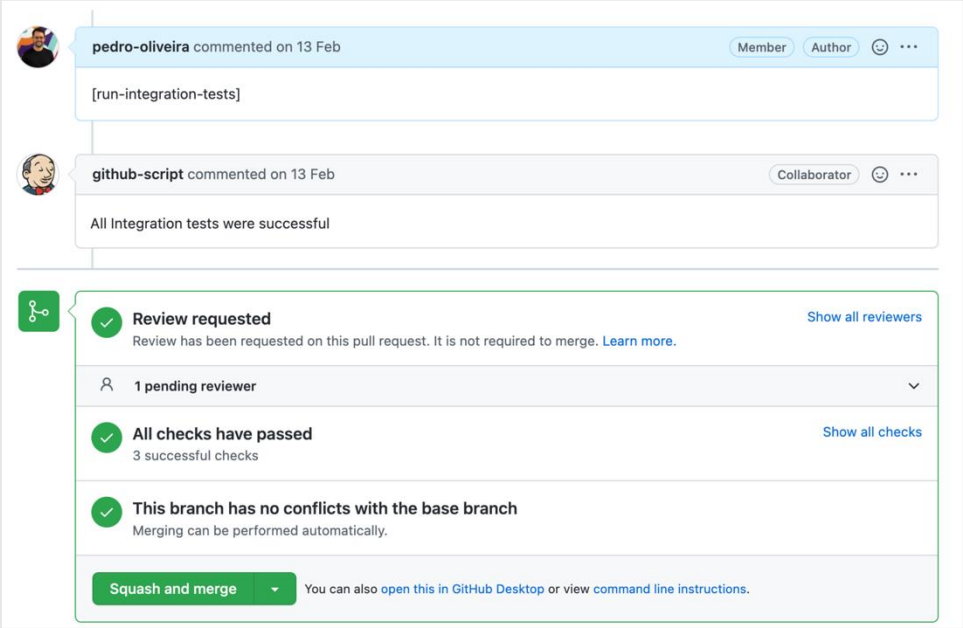

Figure 5.12 – Example of pull request with integration tests

Jenkins also provides a web console that serves as the primary input into the Jenkins system. Among the various features available, users can browse and manage the jobs defined in the system. It is possible to have access to the data of the builds of a certain job through their respective logs. In this way, whenever the application runs the integration tests through a Jenkin job, the logs of this build, also containing the application logs, are saved, which allows its verification in case any of the tests fail. The image below shows a part of the logs extracted from one build in the web console.

```
15:13:31 [info] CompanyAnniversaryTestSuiteSpec
15:13:31 [info] CompanyAnniversaryTestsuite should
15:13:31 [info]   + not create activity for blacklisted user
15:13:31 [info]   + not create activity for user with 'anniversary' privacy setting not allowed
15:13:31 [info]   + create activity for valid user
15:13:31 [info] Total for specification CompanyAnniversaryTestSuiteSpec
15:13:31 [info] Finished in 32 seconds, 203 ms
15:13:31 [info] 3 examples, 6 expectations, 0 failure, 0 error
15:13:31 [info] Passed: Total 3, Failed 0, Errors 0, Passed 3
15:13:31 [success] Total time: 58 s, completed Feb 16, 2022, 3:13:31 PM
```

Figure 5.13 – Part of logs from a Jenkins job build

6. RESULTS AND DISCUSSION

The application developed in this work achieved the proposed objectives and has already been used, mainly by the responsible QA Engineer, in the test phases of the tickets that introduce alterations or implementations to the code of the applications responsible for processing Company Anniversary Updates.

Although still with limited coverage, coupling the developed integration testing tool to a Jenkins job, with its execution through a simple command, has already shown positive results by completely eliminating the need for manual testing in most of the PRs from the applications involved in the Company Anniversary updates pipeline. In these cases, by eliminating manual checks, the time spent on testing has been significantly reduced, which in return increased the team's overall productivity.

The possibility to automatically execute testes to verify the integration of the components involved in this pipeline provided a safer and more efficient way to detect potential code failures that might not have been caught by existing unit tests. It certainly reduced the chances of introducing bugs, and, consequently, increased the team's degree of confidence to deliver new code for production.

7. CONCLUSION

The internship project described here achieved the proposed objectives by successfully developing an application to perform integration tests in a complex data pipeline. The application is serving as a testing framework providing the structure for further interactions to expand its functionalities to cover other test cases. In addition to reporting the activities developed, the objective of this work is to explore the importance of implementing atomized tests in a software production environment.

Working on this project was an excellent and rewarding experience. The internship program lasted exactly one year with the last three months devoted to project development. During the months leading up to the project, all the experience and knowledge I gained working as a data engineer intern were essential for me to have the technical capacity and theoretical knowledge to develop the project.

The project was very technically challenging, and for the development of the application, I had to study and learn many concepts and technologies that were not yet familiar to me. In particular, the synchronization of its components and their connections with other applications to start the test environment, prepare test data and verify the results was quite demanding. The integration of the Jenkins tool with GitHub to run the developed application was also complex and required many attempts.

The internship project offered benefits both for me and for the company, as the developed application will be able to be used and help different teams. I believe that the time spent developing the project was essential to put my knowledge into practice more independently, to better discover the points where I needed to improve, and to gain more experience and technical skills in the area of data engineering.

8. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

Although the internship project reported here has fulfilled all the initial objectives and reached the requirements proposed in its elaboration planning, due to capacity issues and limitations of the time allocated to the development of the project, it was only possible to build the components to perform integration tests into one of the various updates on the platform for which the internship team was responsible.

However, considering this limitation, the project was built in a generic and modular way, to be used as an expandable and dynamic integration testing framework. The project structure was designed to make it easy to extend its functionality and many of the services and methods created in it can easily be reused to test cases of the data workflows involved in the creation of other types of updates.

Therefore, as a recommendation for future work, it is suggested to expand the coverage of the integration tests developed in this project. As possible new types of updates can be created for the platform, as well as the logic of some of the applications involved in these processes can change, the testing framework should ideally be kept up to date by the teams responsible for it.

9. REFERENCES

Amazon Web Services. (2019, July 18). What is Apache Kafka? . https://aws.amazon.com/pt/msk/what-is-kafka/

Arnon, A. (2018). Complete Guide to Test Automation : Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects. In Complete Guide to Test Automation. Apress. https://doi.org/10.1007/978-1-4842-3832-5

Atlassian. (2016, August 23). Kanban - A brief introduction . https://www.atlassian.com/agile/kanban

Beer, B. (2018). Introducing GitHub : a non-technical guide.

Carey, S. (2021, November 1). Complexity is killing software developers | InfoWorld. https://www.infoworld.com/article/3639050/complexity-is-killing-software-developers.html

Chambers, B. (William A., & Zaharia, M. (2018). Spark : the definitive guide.

Cloudera. (2010). Apache Hive Guide. https://opensource.org/licenses/Apache-2.0

Datadog. (2021, November 24). Containerized Applications Overview. https://www.datadoghq.com/knowledge-center/containerized-applications/

Fowler, M. (2006, May 1). Continuous Integration. https://www.martinfowler.com/articles/continuousIntegration.html

Grah, S. (2021, November 24). 6 recommendations for optimizing a Spark job . https://towardsdatascience.com/6-recommendations-for-optimizing-a-spark-job-5899ec269b4b

Halton, C. (2022, September 28). Kanban: What It Means and How It Works in Manufacturing. https://www.investopedia.com/terms/k/kanban.asp

Hamilton, T. (2022, August 24). What is Software Testing? Definition. https://www.guru99.com/software-testing-introduction-importance.html

Humble, J., & Farley, D. (2010). Continuous Delivery : Reliable Software Releases Through Build, Test, and Deployment Automation.

IBM Cloud Education. (2019, May 15). Containerization. https://www.ibm.com/nl-en/cloud/learn/containerization

Karau, H., Konwinski, A., Wendell, P., & Zaharia, M. (2015). Learning Spark : lightening fast data analysis.

Kumar, V. N., & Shindgikar, P. (2018). Modern big data processing with Hadoop : expert techniques for architecting end-to-end big data solutions to get valuable insights.

Li, K., & Wu, M. (2004). Effective software test automation : developing an automated software testing tool. SYBEX.

Loeliger, J., & McCullough, M. (2012). Version Control with Git. www.it-ebooks.info

Meyers, G. j., Sandler, C., & Badgett, T. (2012). The Art of Software Testing.

Microsoft Azure. (2019, May 6). What is Kubernetes? . https://azure.microsoft.com/en-us/topic/what-is-kubernetes/#overview

Miell, I., & Sayers, A. H. (2019). Docker in Practice.

Mili, A., & Tchier, F. (2015). Software Testing : Concepts and operations.

Mouat, A. (2016). Using Docker : Developing and deploying software with containers.

Narkhede, N., Shapira, G., & Palino, T. (2017). Kakfa: the Definitive Guide.

Perez, E. (2020 B.C.E., May 28). How to manage complexity and realize the value of big data - Smarter Business Review. https://www.ibm.com/blogs/services/2020/05/28/how-to-manage-complexity-and-realize-the-value-of-big-data/

Rutherglen, J., Wampler, D., & Capriolo, E. (2012). Programming Hive.

Sacolick, I. (2022, April 15). What is CI/CD? Continuous integration and continuous delivery explained. https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

Shain, D. (2021, June 4). What is Jenkins? How to Use Jenkins for CI/CD and Testing - Applitools. https://applitools.com/blog/what-is-jenkins-how-to-use-jenkins-ci-testing/

Siderova, S. (2018). The Kanban Method: The Ultimate Beginner's Guide! . https://getnave.com/blog/what-is-the-kanban-method/

Stedman, C. (2020, August). Apache Hadoop YARN. https://www.techtarget.com/searchdatamanagement/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator

Stellman, A., & Greene, J. (2014). Learning Agile : understanding Scrum, XP, Lean, and Kanban.

Tassey, G. (2002). The Economic Impacts of Inadequate Infrastructure for Software Testing.

Taylor, D. (2022, September 17). What is MapReduce in Hadoop? Big Data Architecture. https://www.guru99.com/introduction-to-mapreduce.html

White, J.-M. (2015). Hadoop : the Definitive Guide.

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa