INSTITUT SUPÉRIEUR DES ÉTUDES TECHNOLOGIQUES DE NABEUL
المعهد العالي للدراسات التكنولوجية بنابل

Computer Engineering Department

Code: DSI/BA-JF/22

SPECIALTY

INFORMATION SYSTEM DEVELOPMENT

END OF STUDIES PROJECT

ENTITLED

INCEDO LEAD GENERATOR

HOST ORGANISM

INCEDO SERVICES GMBH

RED DREAM SOLUTIONS SURAL

REALIZED BY

ANIS BENNA

FIRAS JABER

ACADEMIC SUPERVISOR

MRS. MAHASSEN KHEMIRI

PROFESSIONAL SUPERVISOR

MR. MANUEL SCHULZE

MR. WAEL JABER

ACADEMIC YEAR 2021/2022

Acknowledgments

We would like to take a moment to thank all of those involved in finalizing this work, whether directly or indirectly.

Our profound gratitude goes to the entire **Incedo Services GmbH team** for their warm welcome, invaluable support and for accompanying us throughout this wonderful journey. Special thanks to our company supervisor and product owner **Mr. Manuel Schulze** who has been very involved with us almost daily throughout the long period of the internship, his trust in us has truly allowed us to come very far during this project. We also appreciate **Wael Jaber** for initially showing us the ropes and for his continuous guidance. Our deepest gratefulness also goes to **Mr. Robert Könitz** and **Mr. Tobias Schulze** for their huge efforts in contributing to our weekly mentoring sessions amongst other things, their input has truly been a huge source of knowledge for both of us and will be forevermore. We would also like to thank both of **Mrs. Mai Ly Moll** and **Mrs. Xenia Harter** for their efforts in business development and for organizing the delightful team events that we gladly had the occasion to participate in, as well as for their almost daily engagements that don't fall short from our supervisor's. We also extend our thanks to the rest of the Incedo team **Mr. Joachim Liedtke**, **Mr. Mohamed Elloumi** and **Mr. Gabor Kluge**.

We also wish to acknowledge the great support of our university supervisor **Mrs. Mahassen Khemiri** for her assistance, availability and valuable spot-on advice. Her faith in us has truly allowed us to keep working on this project stress-free while confident this report will turn out better than initially intended.

Our heartfelt appreciation goes to the jury members who agreed to evaluate our work and we hope they find it up to standard. We equally thank all of the teachers of the Higher Institute of Technological Studies of Nabeul (ISETN) who contributed to our education during these wonderful two and a half years.

Abstract

The Incedo Lead Generator is a proprietary software solution of Incedo Services GmbH that's offered in a software as a service modal. It scrapes LinkedIn and uses it as a medium to directly automate the generating of leads from a campaign using the LinkedIn sales navigator feature.

To start off, we have two big roles to play. **The first** one is to develop new features for the existing software solution while continuously monitoring the changes LinkedIn makes (since we rely on scraping). **The second** and the most important one is to migrate ILG from a monolithic application to a **microservices** architecture in order to make it infinitely scalable. We also have to find the ideal environment to deploy our newly created microservices to, in order to ensure their intra-communication.

We start by breaking our application into various NestJS microservices, build the corresponding docker images then create a shared Helm package for the microservices. Second of all, we setup our own **self-managed Kubernetes cluster with MicroK8s and ansible scripts**, configure it appropriately to support our specific use-case, then deploy our application stack to both staging and production.

Lastly, using **GitLab's CI/CD pipelines** and also **GNU's Makefiles** allows us to automate the whole process from start to finish. So that in the end, one commit on the main branch is enough to deliver our changes to the clients, which is the standard that **DevOps compliant** applications should follow.

Glossary

ILG Incedo Lead Generator

SaaS Software as a Service

POC Proof of Concept

PO Product Owner

VM Virtual Machine

IaC Infrastructure as Code

SDL Software Development Lifecycle

CI Continuous Integration

CD Continuous Delivery

NFS Network File Server

Table of Contents

List of Figures

List of Tables

General Introduction

Companies are always looking for ways to grow and it is the more so especially in the IT world where technology evolves at an insane rate. The main way of achieving growth for any company is to gain the trust of their customers whose satisfaction is at the center of its concerns. And what's better to gain customers' trust than improving the software quality, added value as well as delivery times (Time To Market) or in other words, adhere to the DevOps principles.

It is in light of this background that our End of Studies Project was designed. The goal of the project is to build on the already existing proprietary SaaS (Software as a Service) developed by the company Incedo Services GmbH; it is about further developing the software by delivering new features, fixing bugs and migrating the whole application infrastructure. Since the previous deployment is bottlenecked due to the increase in customers, we have to make the software in question infinitely scalable by separating it to multiple microservices while ensuring the service to service communication. It is imperative on top of that that we setup a fully automated DevOps workflow for continuously monitoring the state of the services and where any changes made to the codebase are delivered to customers on the fly. For that, we have to make the deployment on cloud servers that we additionally need to setup, configure and manage separately.

This report contains four chapters, the first of which is dedicated to introducing our project where we present the general framework of the internship, the host company as well as the assessment of the work. The second chapter is devoted to defining the basic concepts in our project in addition to making comparative studies between the various tools where we explain some of our technical choices where relevant. The third chapter will pick up the pace by setting out the analysis and specifications of the functional and non-functional requirements in order to specify the objectives we want to achieve. The forth and final chapter will then present the implementation phase where we go a bit more into the technical details of the project such as the hardware and software setup in addition to the deployment process in addition to the incorporation of DevOps practices.

Chapter 1

Context of the work

Introduction

This chapter introduces the general context of this report. We start by presenting the frame of the project as well as the host company. Then comes the enumeration of the problems which led to the realization of the project. We wrap it up by defining the methodology we've followed to carry out our work.

1.1 General framework of the internship

This project was carried out within the frame of obtaining a bachelor's degree in Computer Science at the Higher Institute of Technological Studies of Nabeul (ISETN). The internship took place fully remotely at Incedo Services GmbH for five months starting from the 26th of January 2022 to the 30th of June 2022 with the purpose of further developing an existing software solution of the company as well as migrating its infrastructure for scalability and to adhere to modern DevOps principles.

1.2 Company overview

This section introduces the host company **Incedo Services GmbH** as well as the services it offers.

1.2.1 About Incedo

"We are a young software development and consulting firm located in Stuttgart. We help our clients to develop exciting digital products and solutions and we also love to bring our own ideas into life from time to time." [2]

Figure 1: Logo of Incedo Services GmbH

1.2.2 Incedo's services

Consulting

Incedo helps its clients overcome two main challenges:

- Develop a product that somebody actually needs and that creates enough value to form a profitable business case.
- Ensuring that (large) IT-projects are finished in time & budget with a result that satisfies the actual users/clients of the developed software solution.

Development

Incedo develops software for clients, as well as for the company itself.

Figure 2: Selection of Incedo clients

1.3 Stating the problem

Incedo -having ambitious goals to grow over the next 2 to 4 years- wants to win new clients and strategically develop the existing ones. Winning new clients starts with generating new leads. Previously, Incedo has worked with an Austrian start-up (motion group) that provides automated lead generation through LinkedIn at the cost of 0.85 € per requested contact. Although one big project was closed and several leads were generated, Incedo started using the LinkedIn Sales Navigator with a more targeted approach, but nevertheless wanted to automate the approach and increase its outreach. Having launched the first version of the ILG as a SaaS, Incedo was satisfied for a while. Over the time however, as more and more clients were interested in the ILG, the current architecture couldn't handle the load properly. Therefore, it was our task to re-design and implement a new scalable architecture instead of the old one.

1.4 Assessment of the case

1.4.1 Describing the work procedure

The work on any project must first of all be preceded by a thorough study of the existing ones which undermines the strengths and weaknesses of the current system, as well as the business decisions that should be taken into account during the conception as well as the realization.

1.4.2 Criticizing the current state

After studying the existing, we can determine its limitations:

- Bugs always tend to happen whenever the LinkedIn website changes (due to scraping).
- Since cron jobs are running for the whole day, bugs are hard to respond to fast enough because we can only deploy once at the end of day.
- It is hard to test the whole workflow because of how the app works (looking for certain changes in the LinkedIn interface after certain buttons are clicked for example) meaning that the dev environment is lacking.
- It cannot scale well enough since the whole monolithic application is deployed on a single server.
- It is very hard to read the logs of the application since we have to SSH into the VM every time and use the correct grep command.

1.4.3 Proposed solution

The solution to these problems is refactoring the whole application to separate sub-applications or microservices where the automation and scraping processes can be scaled independently of the other parts of the application. This way, it will be easier to maintain and scale the different services as well as respond faster to bugs and know exactly what caused them in the first place. We of course have have to create and organize the deployment pipelines from start to finish in addition to creating the any hardware or software resources that we need for the deployment in the cloud. We also suggest going further by adding recent monitoring solutions for all of our infrastructure in order to increase the availability and the reliability of the application and ultimately improve the experience of the clients using ILG.

1.5 Development Methodology

1.5.1 Agile methodology

Agile is a structured and iterative approach to project management and product development. It recognizes the volatility of product development, and provides a methodology for self-organizing teams to respond to change without going off the rails.

1.5.2 Scrum methodology

Scrum teams commit to completing an increment of work, which is potentially shippable, through set intervals called sprints. Their goal is to create learning loops to quickly gather and integrate customer feedback. Scrum teams adopt specific roles, create special artifacts, and hold regular ceremonies to keep things moving forward.

1.5.3 Kanban methodology

Kanban is all about visualizing your work, limiting work in progress, and maximizing efficiency (or flow). Kanban teams focus on reducing the time a project takes from start by using a Kanban board to continuously improve their flow of work. To explain more in details, Kanban is based on a continuous workflow structure that keeps teams nimble and ready to adapt to changing priorities. Work items —represented by cards— are organized on the board where they flow from one stage of the workflow or column to the next. Common workflow stages are To Do, In Progress, In Review, and Done.

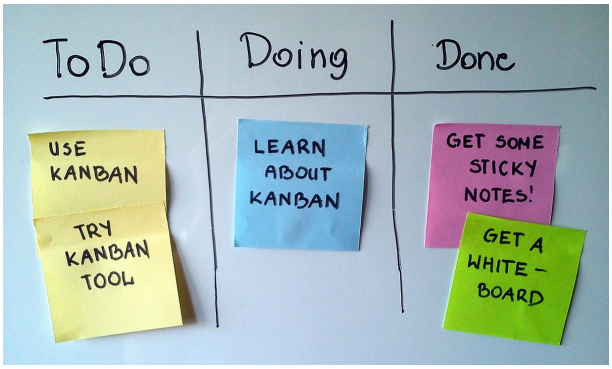


Figure 3: Simple Kanban Board

1.5.4 The choice for ILG

Since this project is very susceptible to changes from outside (LinkedIn), Kanban offers the most flexibility in comparison to Scrum so that's why we went with it instead.

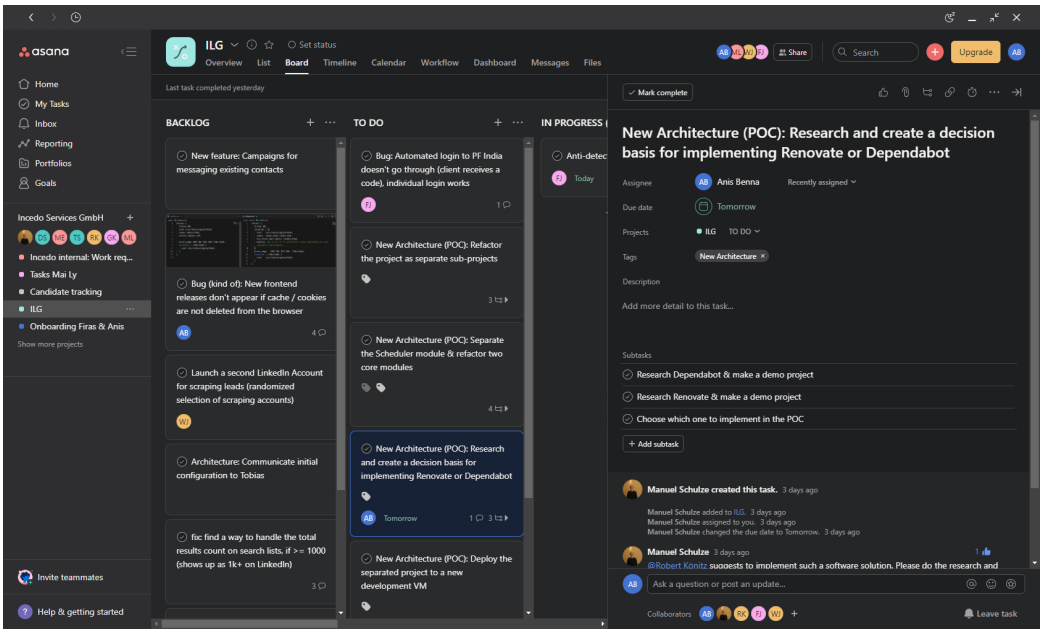


Figure 4: ILG project Kanban board

1.5.5 Unified Modeling Language

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. In our case, we used UML to design the top level view of the systems therefore we only used the Use Case and Sequence diagrams.

Conclusion

In this chapter, we presented not only the host organization but also the general context of the project and why it's needed. We then criticized the current state of the ILG application and discussed the possible development methodologies. Lastly, we picked the most ideal choice for us which is Kanban.

Chapter 2

State of the art

Introduction

This chapter will present and study various concepts such as browser automation, web scraping, version control and DevOps with an emphasis on key DevOps terminologies. We will talk about the most used tools in the market as well as which ones we settled on after making a comparison.

2.1 Automation and Web Scraping concepts

2.1.1 Automation

At its core, automation is leaving menial and recurring tasks to the automaton (or machine) in order to do more meaningful work as a human in the meantime. Implementing automation improves the efficiency, reliability and the speed of tasks that previously took humans a lot of time.

2.1.2 Web Scraping

Web scraping is the process of automatically extracting content and data from a website. Although data extraction is done in a brute way by reading texts from HTML elements, it is used in a variety of legitimate digital businesses like search engines, price comparison sites and market research companies. So contrary to what some might think, web scraping is completely legal.

2.1.3 Relationship between the two

Web scraping simplifies the process of extracting data and the automation process helps repeating the recurring task of extraction. As a result, the combination of scraping data, storing it and automating the whole process is getting very popular, especially with new technologies and tools arising in order to do just that.

2.2 DevOps

2.2.1 Definition

DevOps is the set of practices, techniques and tools used to speed up the Software Development Lifecycle (SDL) by bringing together two historically separate functions of development and operations.

Development refers to writing code and software, whereas operations refer to provisioning servers, configuring them as well as deploying the apps to them, amongst other things.

DevOps teams focus on automating all of the above. The key terminologies around DevOps are Continuous Integration, Continuous Delivery and Infrastructure As Code and automation.

2.2.2 Lifecycle

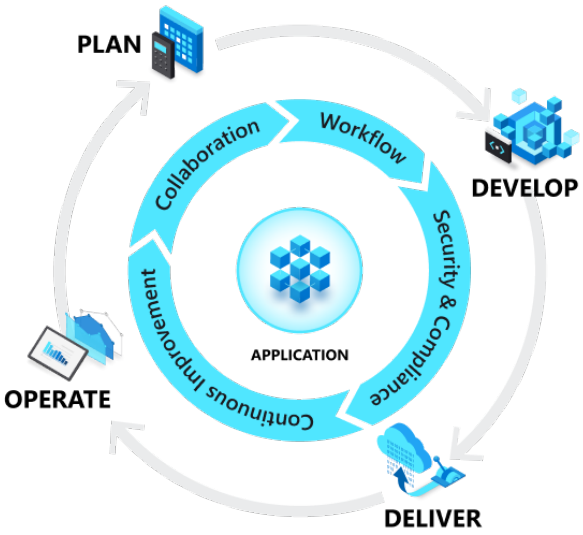


Figure 5: DevOps and the application lifecycle

DevOps -according to Microsoft Azure- is the main influencer of the application lifecycle throughout its plan, develop, deliver, and operate phases. All of the phases rely on one another and they are not role-specific. In a true DevOps culture, each role is involved in each phase to some extent. [3]

- **Plan:** In this first stage, DevOps teams define and describe the main features of the system being created. They accomplish that by many ways. Some of which are creating backlogs, tracking bugs, using Kanban boards and visualizing progress with dashboards.
- **Develop:** This step includes all aspects of coding, testing, reviewing and code integration. It also includes creating build artifacts that can be deployed into various environments. Automation of mundane and manual tasks plays a heavy role in this phase through automated testing and continuous integration.

- **Deliver:** This is the process of deploying application into various environments such as staging or production consistently and reliably. It also includes deploying and configuring the infrastructure that makes up the environments. The DevOps team defines a release management process with automated gates or checkpoints the move the applications between stages in order to deploy in an efficient, scalable, repeatable and controllable way.
- **Operate:** The operate phase involves maintaining, monitoring and troubleshooting the application in the production environment. The purpose of this phase is ensuring high availability, system reliability and zero downtime. This requires alerting and full visibility into the application.

2.2.3 Container management

Containers have become an integral part of DevOps over the past couple of years but what are containers exactly and how do we manage them?

What is a container ?

Simply said; container is a packaging format for software applications that are akin to a very lightweight virtual machine (very broad analogy) which always executes in an isolated environment [4]. What this implies is that said containers can be easily copied to and run on different machines with high reliability, lower costs and high efficiency.

What is container management ?

Container management is the process for automating the creation, deployment and scaling of containers [5]. Container management tools such as Docker and Podman facilitate the addition, replacement and organization of containers.

2.2.4 CI/CD pipelines

A continuous integration and continuous deployment (CI/CD) pipeline is a series of steps that must be performed in order to deliver a new version of software. CI/CD pipelines are a practice focused on improving software delivery throughout the software development life cycle via automation. [6]

2.3 Microservices

2.3.1 Definition

Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams. [7]

Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.

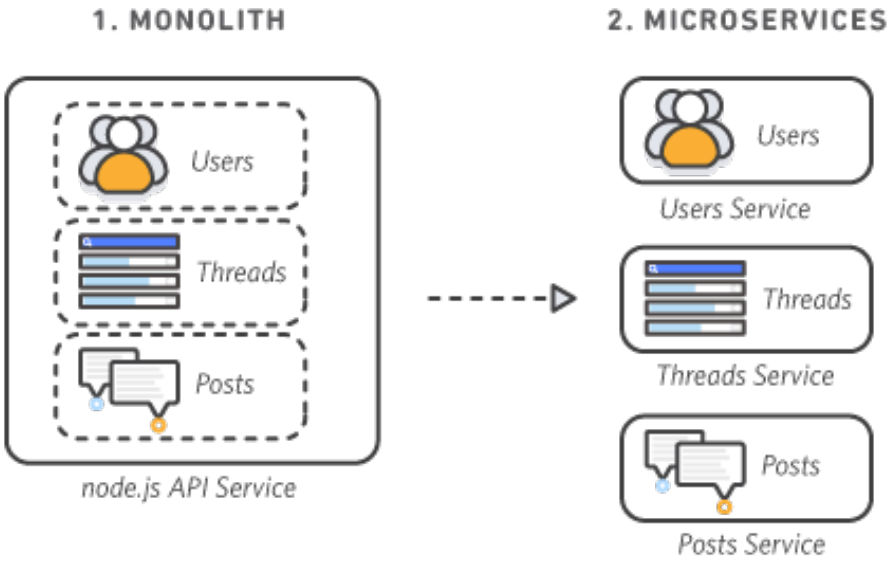


Figure 6: Monolith to microservices example

2.3.2 Characteristics of Microservices

- **Autonomous:** Each component service in a microservices architecture can be developed, deployed, operated, and scaled without affecting the functioning of other services. Services do not need to share any of their code or implementation with other services. Any communication between individual components happens via well-defined APIs.
- **Specialized:** Each service is designed for a set of capabilities and focuses on solving a specific problem. If developers contribute more

code to a service over time and the service becomes complex, it can be broken into smaller services.

2.3.3 Benefits of Microservices

- **Agility:** Microservices foster an organization of small, independent teams that take ownership of their services. Teams act within a small and well understood context, and are empowered to work more independently and more quickly.
- **Flexible Scaling:** Microservices allow each service to be independently scaled to meet demand for the application feature it supports. This enables teams to right-size infrastructure needs, accurately measure the cost of a feature, and maintain availability if a service experiences a spike in demand.
- **Easy Deployment:** Microservices enable continuous integration and continuous delivery, making it easy to try out new ideas and to roll back if something doesn't work. The low cost of failure enables experimentation, makes it easier to update code, and accelerates time-to-market for new features.
- **Technical Freedom:** Microservices architectures don't follow a "one size fits all" approach. Teams have the freedom to choose the best tool to solve their specific problems. As a consequence, teams building microservices can choose the best tool for each job.
- **Reusable Code:** Dividing software into small, well-defined modules enables teams to use functions for multiple purposes. A service written for a certain function can be used as a building block for another feature. This allows an application to bootstrap off itself, as developers can create new capabilities without writing code from scratch.
- **Resilience:** Service independence increases an application's resistance to failure. In a monolithic architecture, if a single component fails, it can cause the entire application to fail. With microservices, applications handle total service failure by degrading functionality and not crashing the entire application.

2.4 Software Testing

2.4.1 Definition

Testing is the process of executing a program to find errors. To make our software perform well it should be error-free. It also makes refactoring and adding code later on a lot easier by preventing the developer from breaking existing feature when adding new ones.

Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features. There are multiple types of testing that we used in our application, here are their definitions:

2.4.2 Types of Tests used

- **Smoke Testing:** This type of test serves as a very shallow probe to see if the rest of the tests should proceed. An example used in the CI pipeline is to check if environment variables are set before building the docker image.
- **Unit Testing:** Unit testing is the first level of testing and is often performed by the developers themselves. It is the process of ensuring individual components of a piece of software at the code level are functional and work as they were designed to.
- **Integration Testing:** After each unit is thoroughly tested, it is integrated with other units to create modules or components that are designed to perform specific tasks or activities. These are then tested as a group through integration testing to ensure whole segments of an application behave as expected.

2.5 Comparative Analysis

In order to get started with our project, we need a wide range of tools that deal with the following areas; version control, orchestration between containers, browser automation as well as scheduled automation. For that, we did a comparative study on some of the tools the market provides.

2.5.1 Version Control: Git vs SVN

The most used tools for version control are Git and SVN. Here is a table comparing both tools.

Table 1: Comparative study between Git and SVN

	Description	Advantages
Git	Git is a distributed version control system which means that when cloning a repository, you get a copy of the entire history of that project.	Git has what's called a staging area. This means that even if you made over a 100 changes, they can be broken down to 10 commits each with their own comments and description.
SVN	SVN or Subversion is a centralized version control system. Meaning that there is always a single version of the repository that you checkout.	SVN has one central repository – which makes it easier for managers to have more of a top down approach to control, security, permissions, mirrors, and dumps.

In the end, both are great options. However, we will be using Git since the company already has a self-hosted GitLab instance that naturally uses Git.

2.5.2 Container management: Docker vs Podman

Although Docker is widely popular, Podman is also taking its share from the market. Especially on Red Hat systems.

Table 2: Comparative study between Docker and Podman

Aspect	Docker	Podman
Definition	Docker and Podman are both container management technologies used to build container images and store said images in a registry to then run them as containers in a target environment.	
Technology	Docker uses the containerd daemon which does the pulling of images then hands over the creation process to a low-level runtime named runc.	Podman uses a daemonless approach using a technology named conmon which does the heavy lifting. It also delegates the container creation to a low-level container runtime.
Specificity	Docker Desktop is a great feature for Docker which provides an easy way to build and distribute containers amongst developers.	The smallest unit in Podman is the pod. A pod is the organizational unit for containers and is directly compatible with Kubernetes.

We can see the similarities between the two and in theory, it shouldn't really matter which one we choose since both Docker and Podman are interchangeable.

2.5.3 Browser automation: Puppeteer vs Playwright

Both are Node.js libraries for browser automation. The table below compares these two on different levels.

Table 3: Puppeteer vs Playwright highlights

Category	Puppeteer	Playwright
Overview	Puppeteer makes it easy to get started with browser automation. This is in part because of how it interfaces with the browser.	Playwright is very similar to Puppeteer in many respects. The API methods are identical in most cases, and Playwright also bundles compatible browsers by default.
Community	Has a large community with lots of active projects.	Small but active community.

Since the difference is pretty minor, we will be sticking to the more recent Playwright.

2.5.4 Database: MongoDB vs PostgreSQL

The old architecture of ILG uses PostgreSQL but we have the choice to use a different database as well, so we did the following study.

Table 4: Comparative study between MongoDB and PostgreSQL

Category	MongoDB	PostgreSQL
Overview	An open-source NoSQL that stores data in JSON-like documents.	An open-source relational database system.
Storage	Stores data in documents that belong to a particular class or group.	Data is stored in tables where each table corresponds to an entity.
Type	NoSQL, meaning that data in a collection can have different structures.	Uses SQL (Standard Query Language) meaning that the schema of data cannot be changed once defined.

Since the old architecture uses PostgreSQL, we will be sticking to that in the new architecture as well for an easier migration.

2.5.5 Database: GraphQL vs REST

One of the decision we need to make is how to exchange the data in our application. For that, we compare GraphQL and REST.

Table 5: Comparative study between GraphQL and REST [1]

	Description	In depth
GraphQL	An open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data.	Describe only what's needed in queries, so no under or over fetching. It is also Schema and Type safe so it's less prone to bugs.
REST	REST (Representational State Transfer) is an architectural style that conforms to a set of constraints when developing web services.	It was introduced as a successor to SOAP APIs, follows the REST standards and is not constrained to XML.

Due to time constraints, we agreed to use both. GraphQL will be used internally between the microservices. REST on the other hand will still be used by the frontend to communicate with the backend api service.

2.5.6 Deployment: Docker Swarm vs Kubernetes

The old deployment uses Docker Compose but that will not cut it anymore since it is not very optimized for production, especially on systems that need to scale later on as is the case here. Our research leads us to choosing between either Docker Swarm or Kubernetes.

Table 6: Comparative study between Docker Swarm and Kubernetes

Aspect	Docker Swarm	Kubernetes
Overview	Docker Swarm is a lightweight, easy-to-use orchestration tool with limited offerings compared to Kubernetes. In contrast, Kubernetes is complex but powerful and it provides self-healing and auto-scaling capabilities out of the box.	
Advantages	• Straightforward to install • Takes less time to learn • Works with the Docker CLI	• Can sustain and manage large architectures and complex workloads • Has a self-healing capacity that supports automatic scaling. • Supports every operating system. • Has a large open-source community with Google's backup. • The most popular distributed system orchestrator in the world.
Disadvantages	It has been abandoned by Docker Inc. and is no longer maintained.	It has a steep learning curve and requires separate CLI tools.

Therefore, we will be deploying our application stack to Kubernetes since the potential is huge compared to the deprecated Docker Swarm.

2.5.7 Deployment management: Kubectl vs Helm

Since we're using Kubernetes, we have the option to deploy our microservices either with Kubectl or with Helm.

Table 7: Comparative study between Kubectl and Helm

Aspect	Kubectl	Helm
Overview	Kubectl is the official Kubernetes command-line tool that allows running commands against Kubernetes clusters.	Helm -in a nutshell- is the package manager for Kubernetes.
In depth	Kubectl can be used to deploy applications, inspect and manage cluster resources and also view logs. It also provides many other advanced functionality such as node tainting or setting up strategies for load balancing.	Helm helps in defining, installing and upgrading the most complex Kubernetes applications. Helm templates also provide a very clean way to avoid code duplication between similar resources in applications, such as services or deployments.

Thoroughly studying our use case, we agreed to create a uniform Helm package to ship our microservices. We also deemed it necessary to have some common configuration that we simply apply with Kubectl such as Ingress rules or letsencrypt certificates for convenience purposes and due to the lack of time.

Conclusion

In this chapter, we discussed the main concepts relevant to our project such as web scraping, automation and DevOps. Then we talked about some of the tools in the market and discussed some of our choices.

Chapter 3

Analysis and specification of requirements

Introduction

In this chapter, we are going to analyze the various specifications and requirements of the whole ILG project from development to production. We will also describe the product's main features and the conception that should be met for the end result.

3.1 Functional requirements

Since our project is fairly voluminous, we identify what we consider the necessities for ILG as follows:

- Actively editing and enhancing the portal server dashboard by implementing new features to increase customer satisfaction.
- Separating the monolithic application to multiple microservices to validate the POC (Proof of Concept) as instructed by our Product Owner before proceeding with the making of the new architecture.
- Daily monitoring of the old system by reading logs directly from the server.
- Ensuring that the lead generating is happening in the background with the newly implemented microservices architecture.
- Creating individual docker files for all microservices based on what they require.
- Creating the GitLab CI pipelines for all microservices and configuring them to build and push Docker images to a custom container registry.
- Creating a custom configurable Helm Chart for the whole ILG application stack and publishing it to GitLab's package registry.
- Working on the CD part of the pipelines by setting up a clear deployment strategy to Kubernetes using the custom Helm Chart.
- Setting up modern monitoring solutions that use the most out of our Kubernetes deployment with the most recent tools on the market.

3.1.1 Application dashboard

Use case

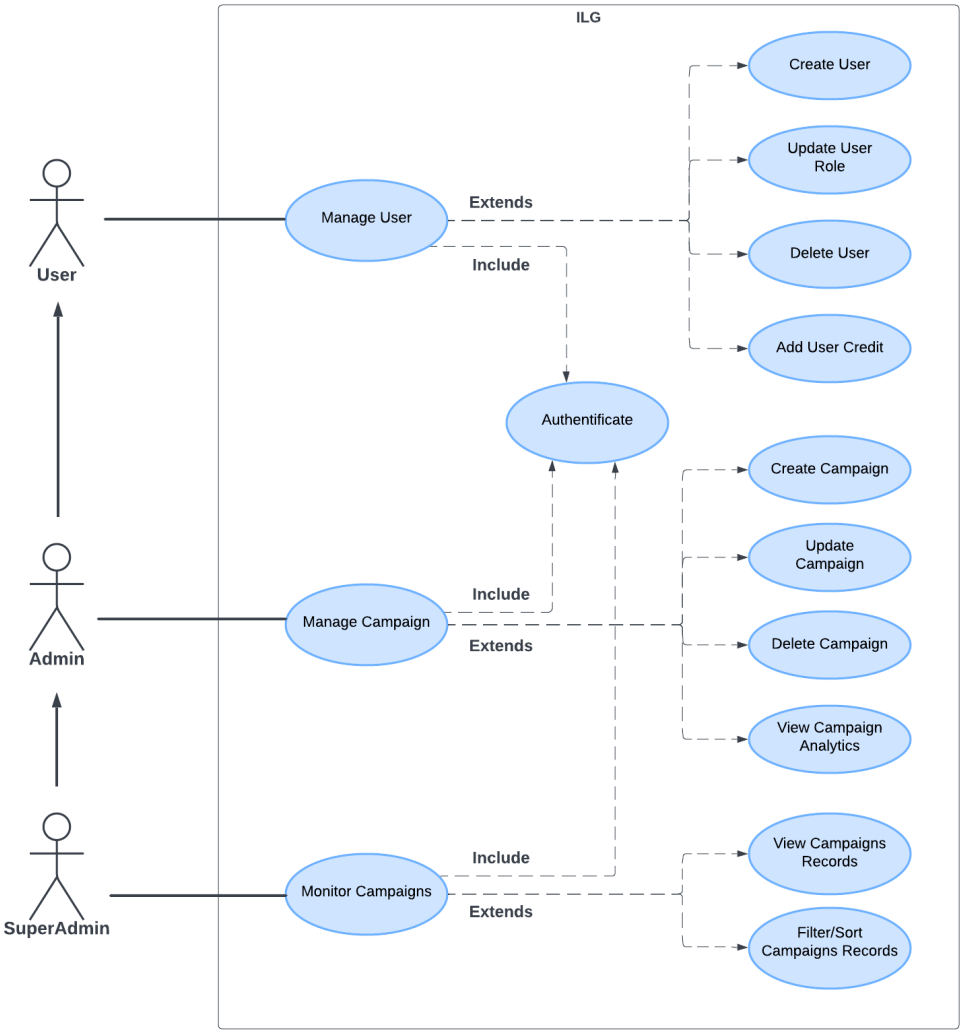


Figure 7: Use case diagram for the frontend dashboard

Actors

Mainly, there are 3 types of actors of the system:

- **Super Admin**: The main administrator with full access permissions to the system.

- **Admin:** The administrator of the system with slightly less privileges than the Super Admin.
- **User:** The main user of the system, who benefits from the different features the system offers, usually a client representative.

User management

Here are the main requirements of user management for the different type of actors:

- As a **Super Admin**, I can create, modify or delete users and admins accounts.
- As an **Admin**, I can create, modify or delete user's accounts (clients).
- As an **Admin**, I can add or remove credit from user's accounts (clients).
- As an **Admin**, I can view my user's accounts, their used and bought credits.

Campaign management

Here are the main requirements of campaign management for the different type of actors:

- As a **User**, I can create my campaign specifying the LinkedIn account credentials to be used, the campaign list, salutation messages, invite messages, follow up messages and the number of invites per day.
- As a **User**, I can update my campaigns details such as the LinkedIn account credentials, the campaign list, salutation messages, invite messages, follow up messages and the number of invites per day.
- As a **User**, I can delete my campaigns.
- As a **User**, I can activate my campaigns.
- As a **User**, I can stop my campaigns.
- As a **User**, I can export information about the scraped leads as a CSV file.

Campaign monitoring

- As a **Super Admin**, I can view the campaigns' records, on day by day basis or in a specific range of dates with information such as number of

invites sent, number of follow ups sent and the number of withdrawn invites.

- As a **User**, I can view my campaigns' analytics with information such as requested, connected, replied leads, and conversion rates.

Use case textual description

Here is an example of the textual description of one of the use cases scenarios:

Table 8: Create Campaign's use case textual description

Use Case	Create Campaign
Actor	User.
Pre-condition	Authenticated User.
Post-condition	Campaign Created.
Basic path	1. Navigate to campaign page. 2. Click on the "create campaign" button. 3. Fill in the needed information of the campaign in the popped modal form. 4. Validate the form then submit.
Exceptional path	On step 3, if the campaign sales navigator link is not valid, show "invalid campaign link" to the user and restart the step.

3.1.2 Lead generating

Definition

The lead generating is the process of **scraping** the LinkedIn accounts and **generating** leads by sending them customized invites and follow ups messages through the LinkedIn UI depending on the user's campaign preferences. So here is the requirements of both parts:

Scraping leads

- Each day, the system generates a list of URLs to scrape from a campaign (search) list and saves them as tasks in the database where each list item corresponds to a lead's profile URL.
- Each day, the leads' information are scraped from the previously generated URLs. Each scraped URL corresponds to one lead and is saved as an entry in the database.

Generating leads

- Each day, the system sends a certain number of invites to the scraped leads. The number is between a range the user specifies in the UI.
- The system sends out the first follow up message to leads that accepted the connection request but did not yet reply. The first follow up message is sent at the earliest 24 hours after the connection request has been sent.
- The second follow up message is sent out on the day after the first follow up message was sent if the lead did not reply yet.

3.2 Conception

In this section, we will dive deep into the conception and the logic behind the parts of the application:

3.2.1 Architecture

The next diagram shows the various microservices that make up our application. The bulleted lists show the main tasks of each of the services. We can see that the heavy services interact with a GraphQL data layer whereas the frontend interacts with an API layer.

We will discuss each service with more details in the upcoming sub-sections.

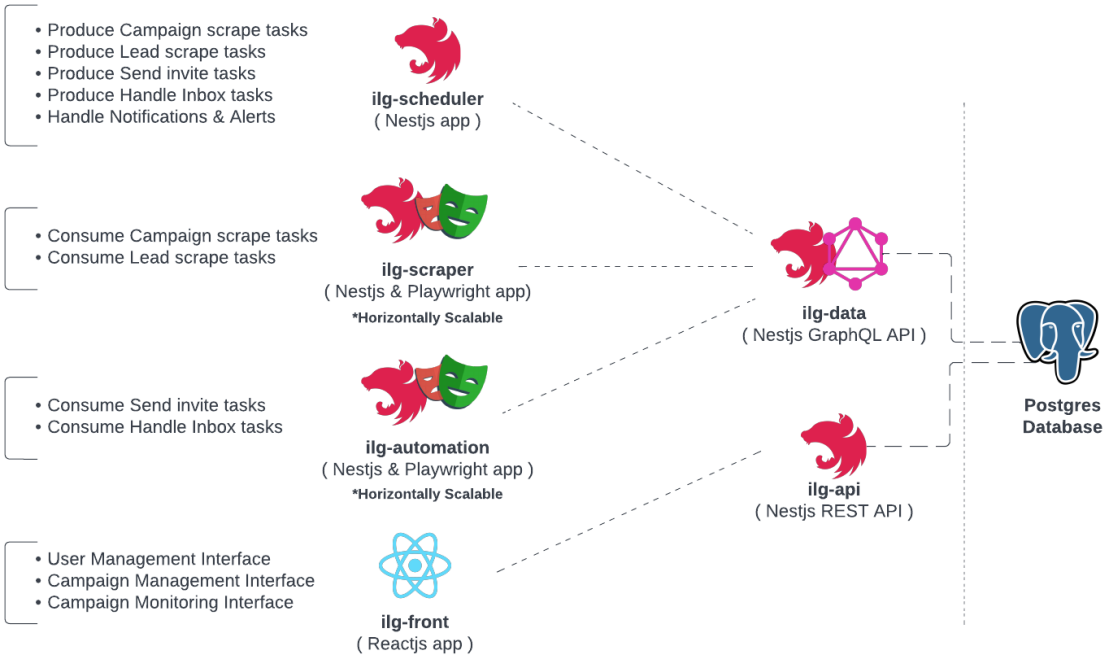


Figure 8: Overview of ILG services

Microservices

Table 9: All the application's Microservices

Name	Description	Scalable
ilg-data	A GraphQL data access layer used to feed all needed data for other services from the PostgreSQL database.	No
ilg-api	A RESTful api used to serve data to the React app dashboard.	No
ilg-front	A React app dashboard application served through NGINX.	No
ilg-scheduler	The service that's responsible for scheduling and orchestrating the tasks queues and notifications.	No
ilg-scraper	The service that's responsible for scraping leads from the campaign links generated from LinkedIn Sales Navigator.	Yes
ilg-automation	The service that's responsible for handling LinkedIn accounts' inboxes, threads and for sending invites to the leads.	Yes

Cloud Infrastructure

Since we are deploying to a Kubernetes cluster, we distribute our different microservices into various nodes or virtual machines depending on which services they are handling and whether they are scalable or not. Seeing that we're managing the cluster ourselves, we come up with a custom architecture that we do the conception of using Miro. Miro is a cloud-based white dashboard tool used for quick and efficient collaboration.

Please see the next screenshot for the architecture that we create and adopt.

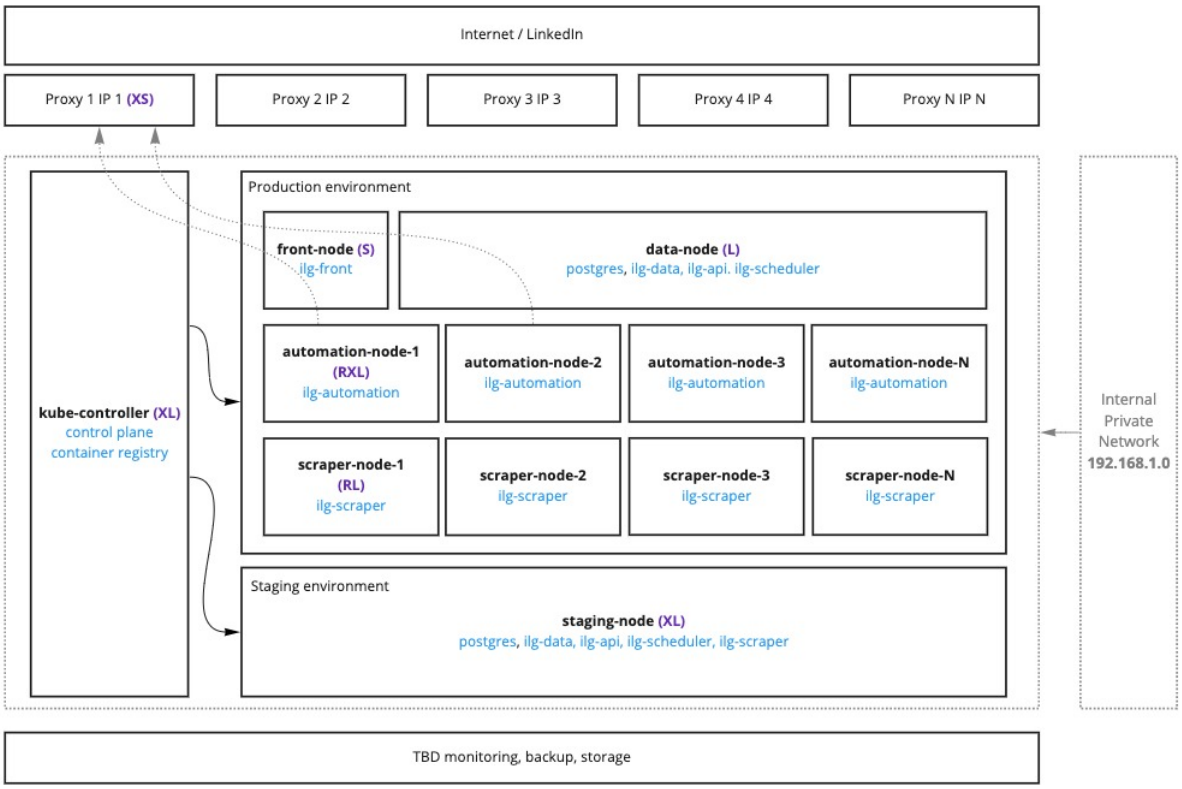


Figure 9: Custom cloud infrastructure of the application

3.2.2 Tasks & Queues

In theory

Task queues allow services to perform work in the form of asynchronous tasks outside of a user request. If an app needs to execute work in the background, it adds the tasks it needs to the task queues. The tasks are executed later, by worker services. The Task Queue service is designed for asynchronous work. Every lead scraping or generating tasks are implemented as a task queue.

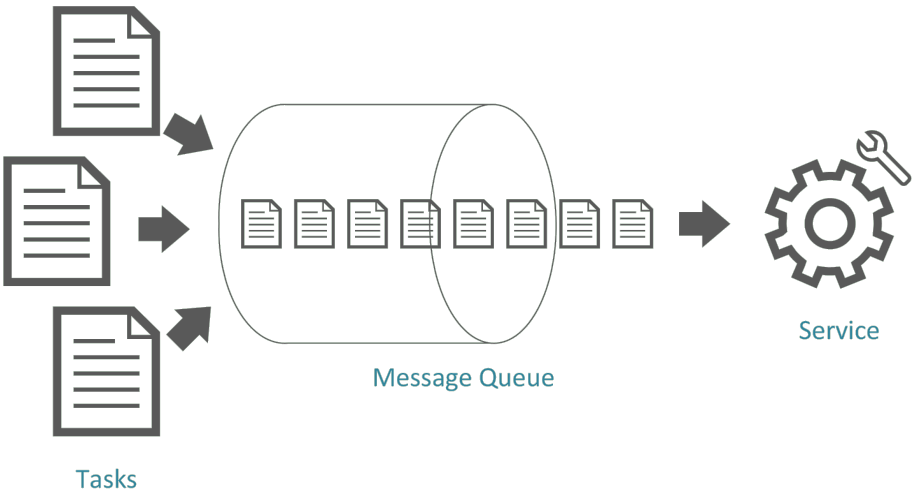


Figure 10: Task queues

Tasks queues types

Table 10: The application's tasks or jobs

Name	Description
campaign-scrape	Scrapes a list of leads from a campaign search list or URL.
lead-scrape	Scrapes the lead information from a URL and saves it in the database.
handle-inbox	Opens the conversation inbox of leads and sends them follow up messages if necessary.
send-invite	Sends a connection invite to leads through LinkedIn.

Scheduling tasks

Scheduling the different types of tasks is mainly done by the ilg-scheduler service on a daily basis according to a specific cron job we specify in code. We use the database that's accessible by ilg-data, as the store of those task queues with all of their corresponding information and state. The diagram below illustrates the different producers and consumers of the tasks in more details.

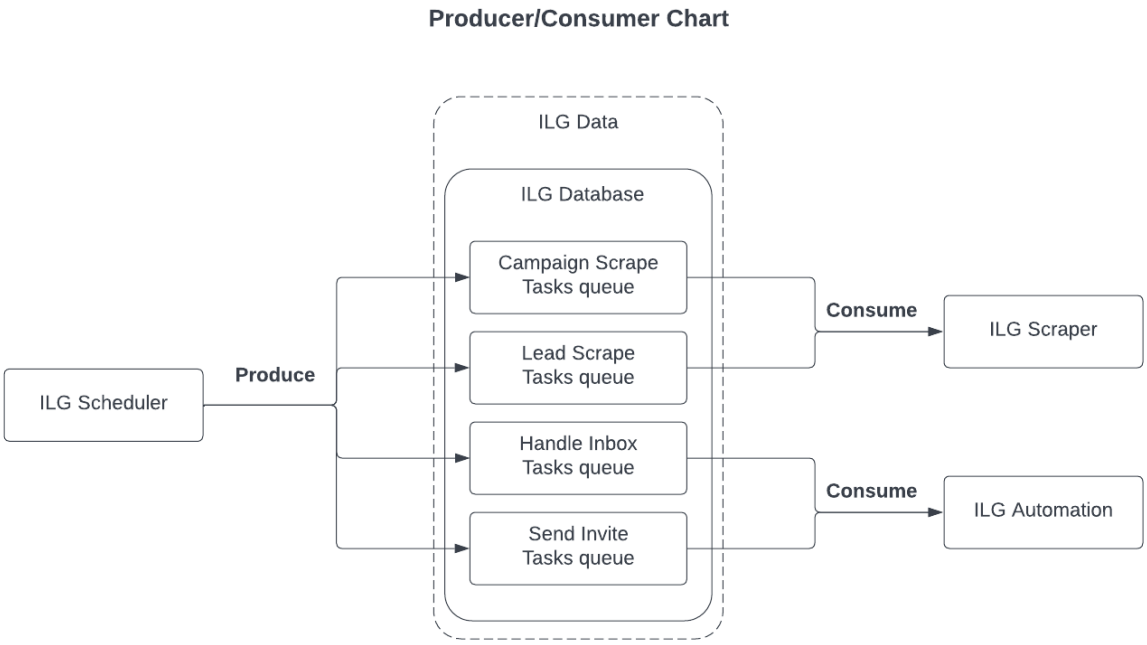


Figure 11: Chart detailing the producers and the consumers

3.2.3 Scraper & Automation

The ilg-scraper and ilg-automation services are responsible for consuming the tasks generated by the system on a daily basis.

The main purposes of the two modules boils down to:

- **Scraper:** Scrapes the LinkedIn information of various leads from a campaign link that is a valid sales navigator URL.
- **Automation:** Generates leads by sending connection requests or invites to the many LinkedIn profiles saved by the scraper using the customer's LinkedIn account.

So that in the end, if a lead replies to a customer, we register that as a valid lead that has been acquired and save a new record to be able to accurately bill our customers.

Scraper service flow

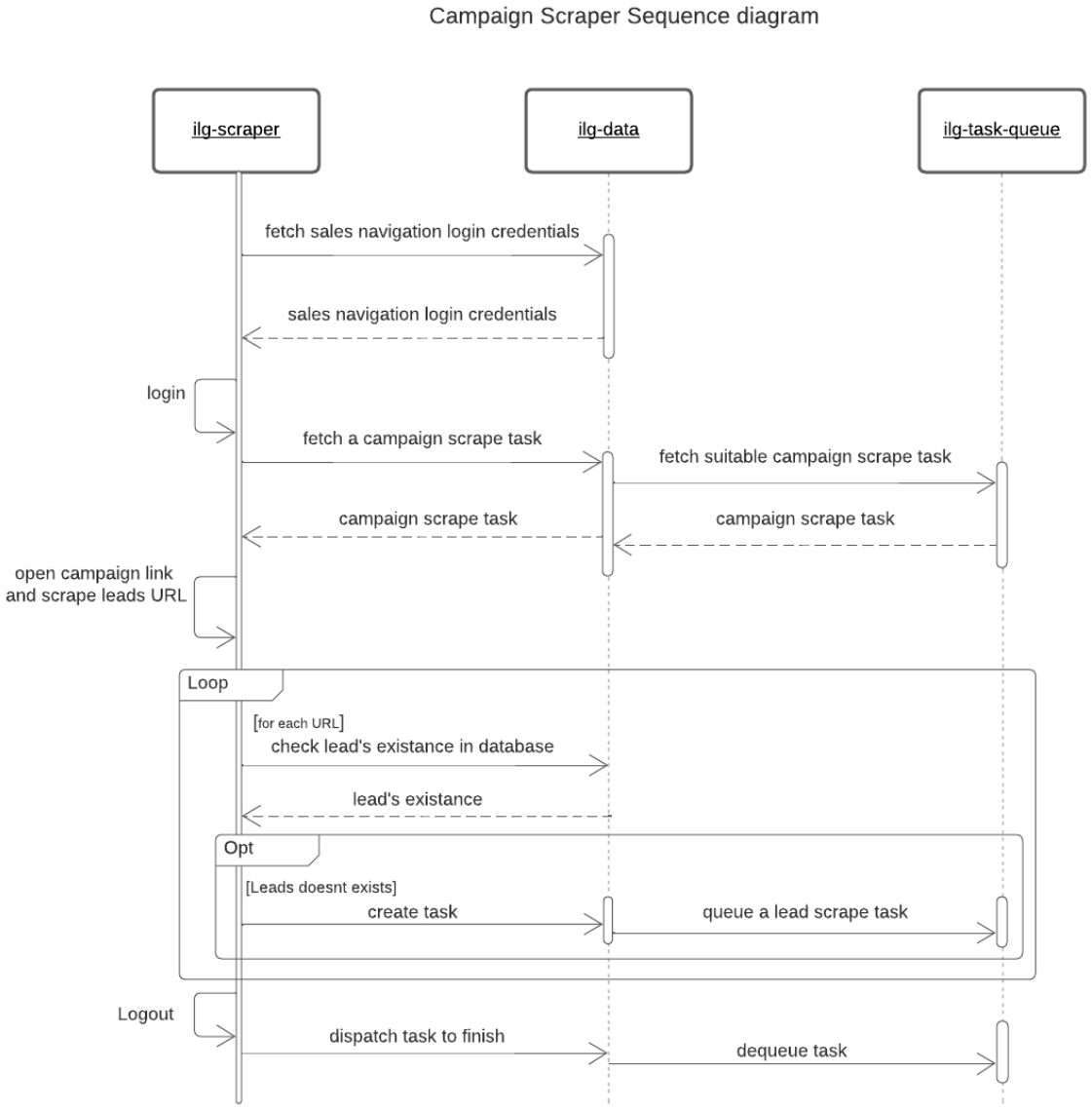


Figure 12: Campaign scraper sequence diagram

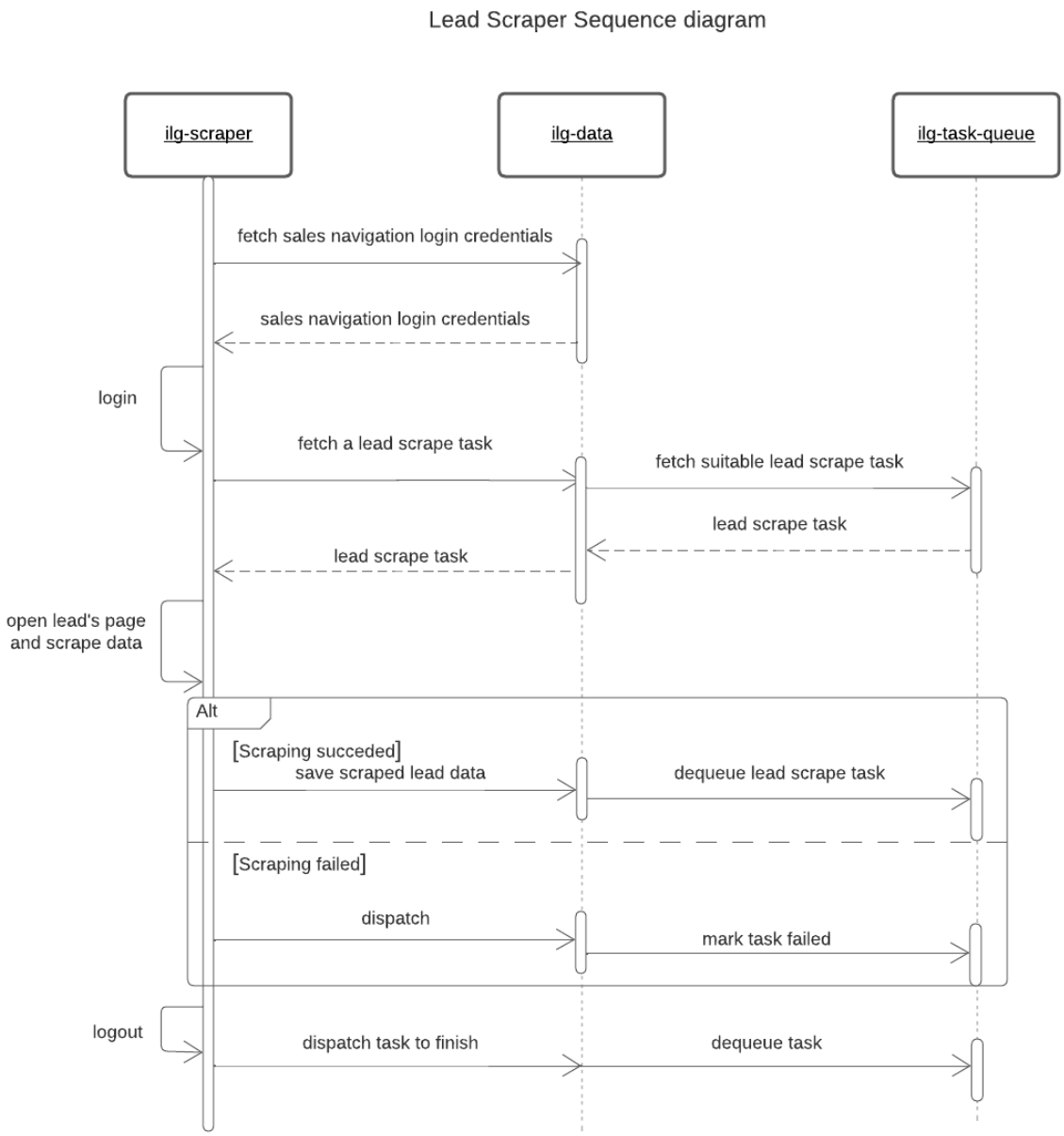


Figure 13: Lead scraper sequence diagram

Automation service flow

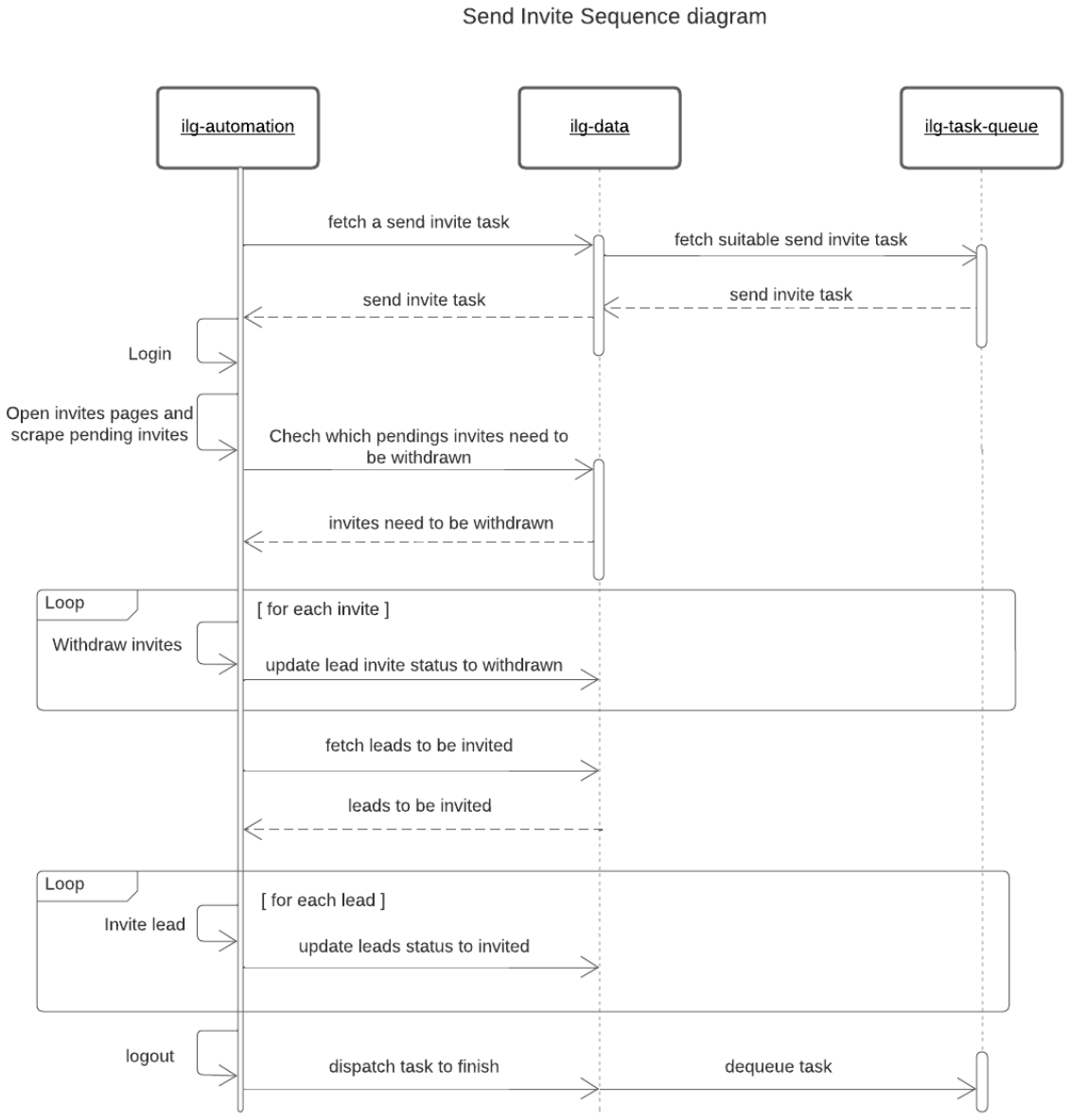


Figure 14: Diagram detailing the producers and the consumers

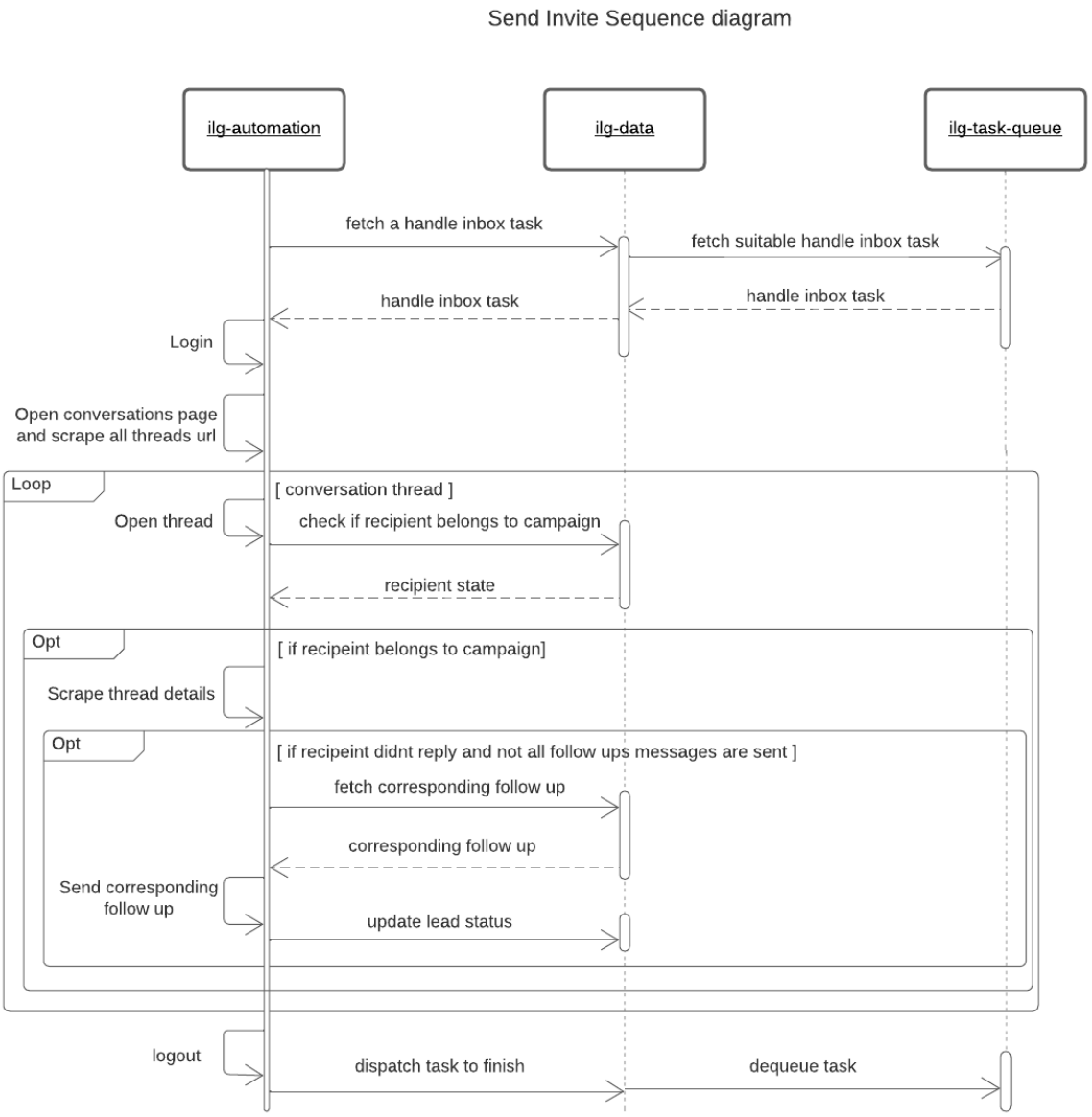


Figure 15: Diagram detailing the producers and the consumers

3.2.4 Data Layers

Schema or Entity Relationship Diagram (ERD)

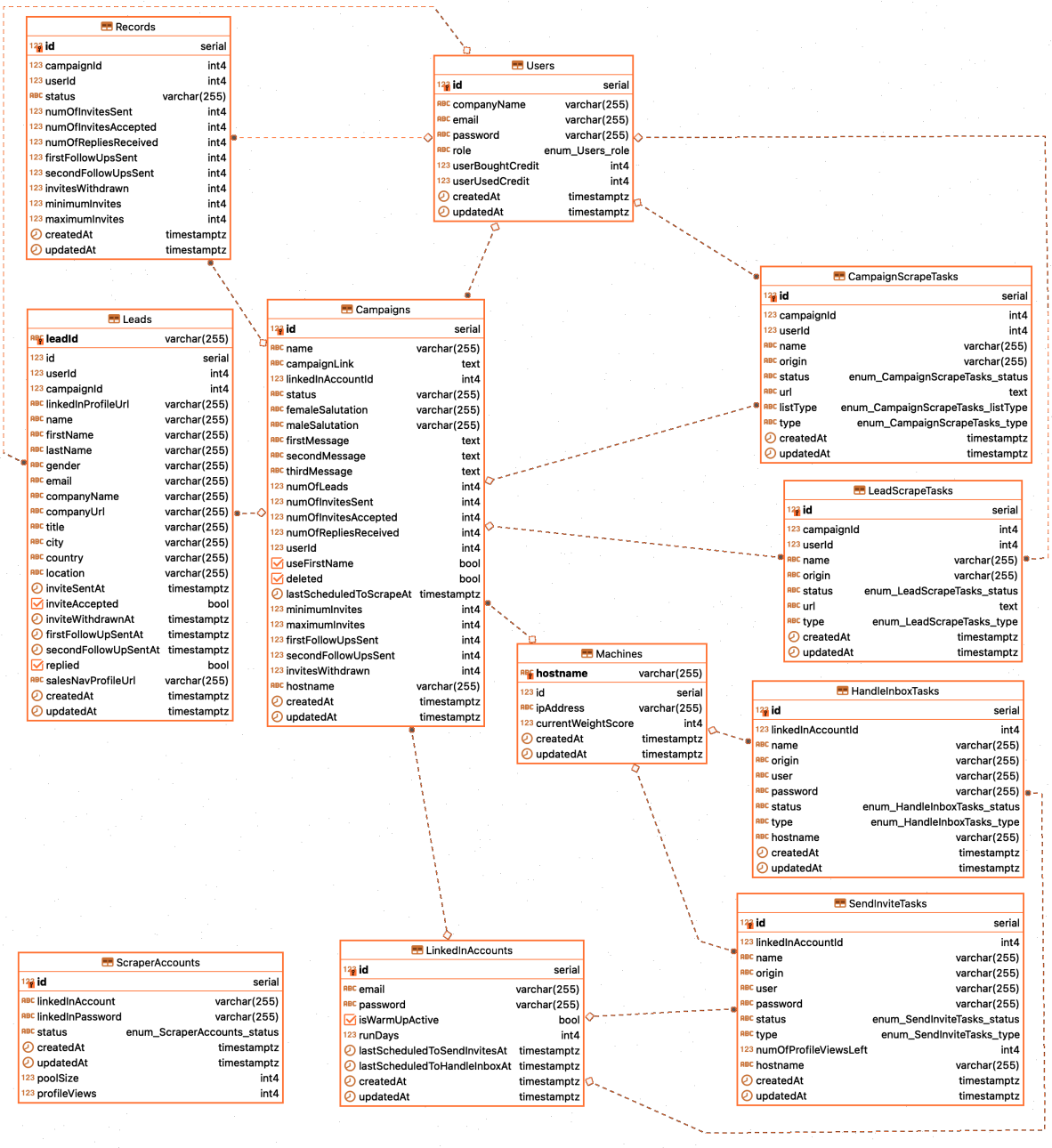


Figure 16: Entity Relationship Diagram

Communication between services

For the intra-communication between the various services, two interfaces are provided:

- **GraphQL API Layer:** This interface is used by all the internal services of the application to access the data. In other words, this layer is only reachable within the **internal private virtual network** and it is **not exposed publicly**.
- **RESTful API Layer:** This interface is used by the frontend service to access the data. In other words, this layer is **exposed publicly**. It is of course secured by a JWT token level authentication.

3.3 Non-functional requirements

Non functional requirements describe any specification that does not add direct business value but is nonetheless crucial for the good operation of a developed software. For ILG, what we should focus on is:

- **Reliability and Availability:** The software should be available 24 hours a day and 7 days a week.
- **Security:** Clients should be able to safely login to the dashboard with a strong authentication system.
- **Maintainability and Recoverability:** We should respond to LinkedIn changes all the time as fast as possible.
- **Scalability:** This is the main reason we're migrating to a new architecture in the first place.
- **Documentation:** We should always document each and every step we make extensively so that new developers can onboard on the project in a fast and efficient way, which further emphasizes reliability.

Conclusion

In this chapter, we discussed our main objectives that we should meet during the course of our internship. We also discuss the various layers and services of the application. In what follows, we will discuss the realization part with heavy focus on DevOps and the actual implementation of the new architecture.

Chapter 4

Realization: Main setup

Introduction

In this chapter, we will finally discuss the long awaited realization. This is the implementation phase where all of the work done in the previous chapters comes into picture. This section will **heavily focus on DevOps** since our primary objective has been migrating the ILG application to a new scalable architecture using modern DevOps standards. We will start with the new architecture hardware and software setup then talk about how to monitor the whole application stack.

4.1 Hardware setup

Since we agreed to proceed with the self-managed Kubernetes deployment as mentioned in **Chapter 2: State of the art**, we must create our own cluster nodes or virtual machines. The nodes in question can be separated into the core nodes that are create once, and the scalable nodes that can be created indefinitely whenever we need to scale.

4.1.1 Cloud provider

To provision the hardware resources that we need for the deployment, we're going to use the IONOS Cloud provider. [8, Ionos is a web hosting company founded in Germany in 1988 and is currently owned by United Internet.]

Figure 17: Logo of IONOS

IONOS Cloud provides -through the Ionos cloudpanel service- an easy way to create and manage virtual machines, firewall rules, private networks and various other infrastructure components.

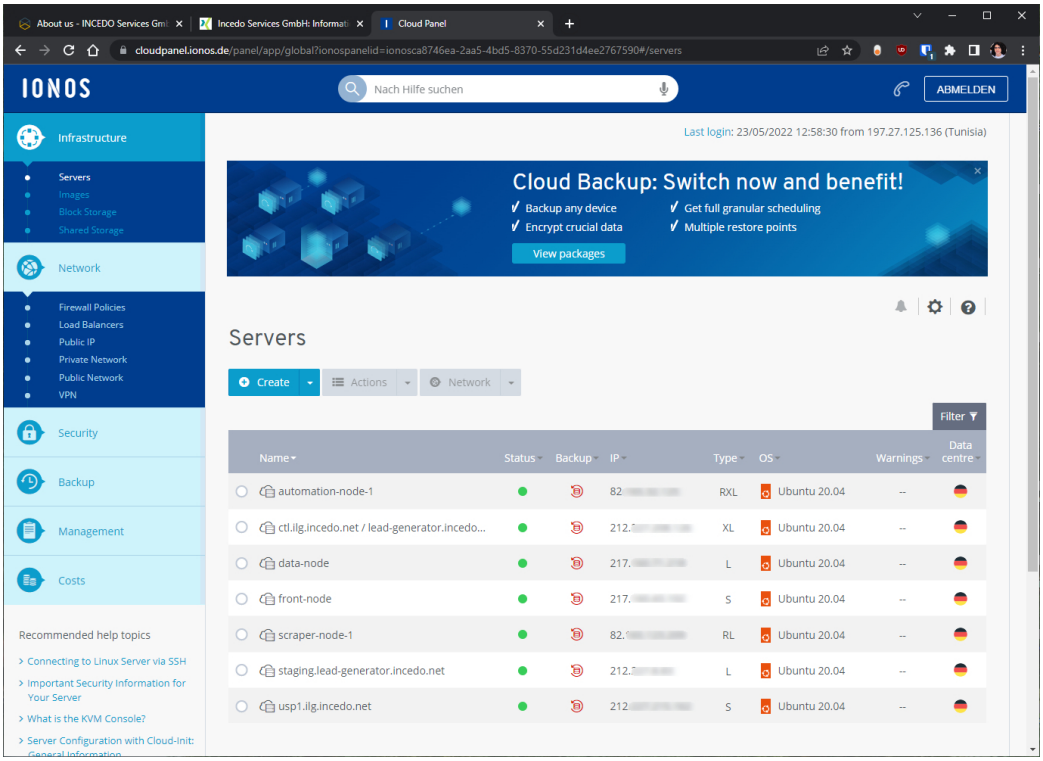


Figure 18: Ionos cloudpanel dashboard

The dashboard options of IONOS cloudpanel we use during our deployment process are:

Infrastructure:

- **Server:** To create and manage our virtual machines.
- **Block storage:** To create and connect an SSD to a single server for extra storage.

Network:

- **Firewall Policies:** To create firewall rules to protect our VMs.
- **Private Network:** To create and manage private networks for server to server communication.

The two images below show the available VM sizes in the Ionos cloudpanel dashboard and their corresponding specifications. We will refer to the sizes in the tables of the next section.

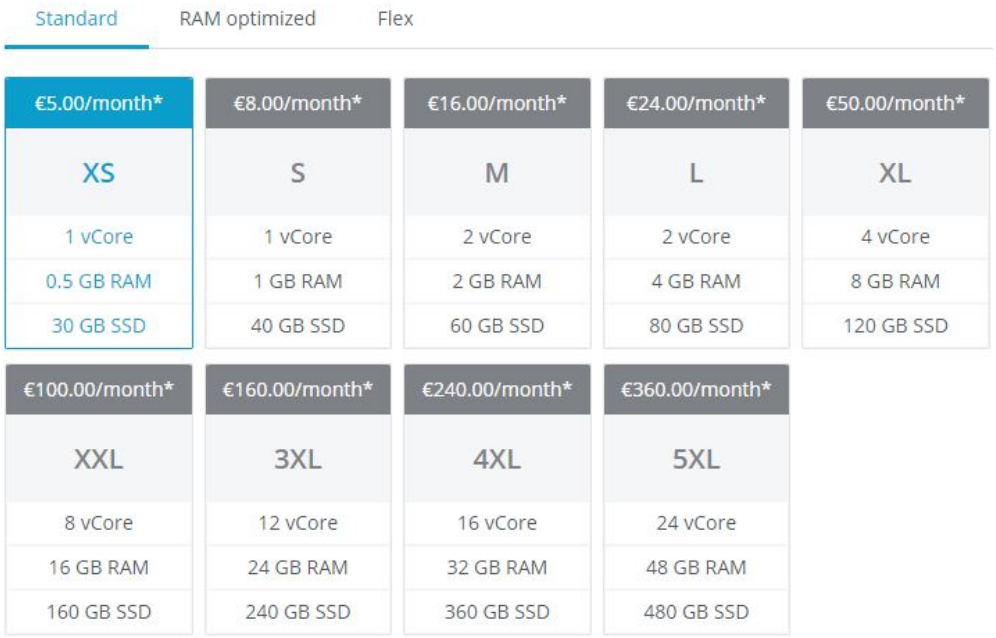


Figure 19: Standard VM sizes in IONOS

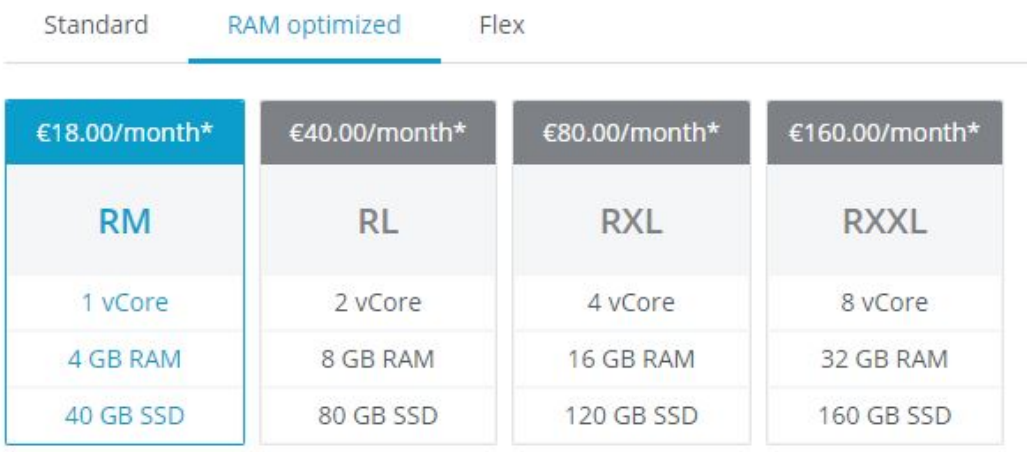


Figure 20: RAM Optimized VM sizes in IONOS

4.1.2 Kubernetes Cluster

Using the cloudpanel dashboard, it is time that we setup our cluster. The tables below show the virtual machines that we need to get started.

Table 11: Core cluster nodes

Name	Services	Size
kube-controller	The main Kubernetes cluster controller	XL
data-node	PostgresDB, ilg-data, ilg-api, ilg-scheduler	L
front-node	ilg-front	S
staging-node	PostgresDB, all other ilg services	L

Table 12: Scalable cluster nodes

Name	Services	Size
scraper-node-n	ilg-scraper	RL
automation-node-n	ilg-automation	RXL

Please note that

- What each of the services represent is already mentioned in **Chapter 3: Analysis and specification of requirements**
- The size column which stands for the specification of the virtual machine is described in the previous section.
- The number of servers for **scraper**, and **automation** nodes will be scaled manually depending on the number of clients whenever the need arises.

4.1.3 Proxy Servers

Proxies are used to ensure all requests to LinkedIn come from a single IP address in order to avoid having to write a verification code every time.

Table 13: List of proxy servers

Name	Services	Size
usp-n	Upstream proxy server running Squid	XS

4.2 Software setup

After setting up our hardware resources, we end up with raw Linux virtual machines with no configuration whatsoever. We have to create and manage our own Kubernetes instance on it to proceed any further. We will be using **MicroK8s**. But before that, we also need to setup some basic configuration.

4.2.1 Manual configuration

For a single virtual machine, we have to do the following setup:

- Create a user with sudo permissions
- Install the base packages we will be using
- Edit the hosts file so that the virtual machine recognizes the other virtual machines by their hostnames
- Install MicroK8s to have a single-node cluster running on the virtual machine
- Join the virtual machine with the master node to have a multi-node cluster.

4.2.2 Automating the process

Manually configuring the above can be extremely tedious, inefficient and error prone. Therefore, we've created **Ansible** scripts or playbooks that do just that. In the end, we simply edit a configuration file and directly run the script. This has a huge advantage especially when

- We want to create a development environment for easy prototyping.
- We want to use the exact same setup for future projects.
- We want to add nodes to the cluster later on when scaling.

The playbook in question should first of all, apply the common configuration to all nodes. Second of all, it should install MicroK8s on all nodes. After that, it needs to configure the master node appropriately. Finally, it configures the other nodes to join the cluster as worker nodes so that they can execute our services.

There are more details related to Kubernetes such as node tainting and node selectors that won't be mentioned in this report due to time constraints.

Also note that we won't get into too much details about how Ansible works in this report also due to time constraints and since it will become very lengthy. We'll only mention that the connection is established using SSH.

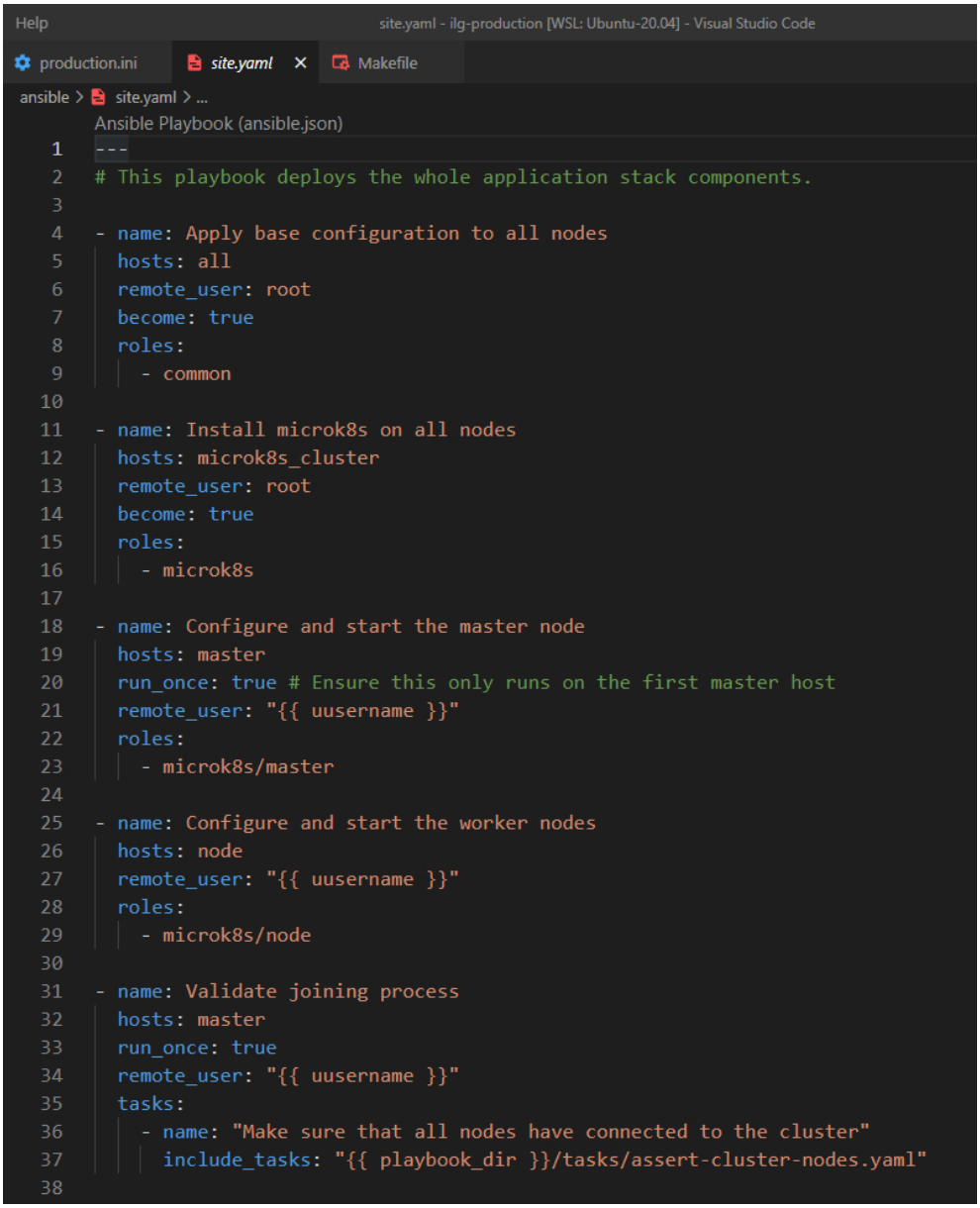


Figure 21: Extract from the Ansible playbook

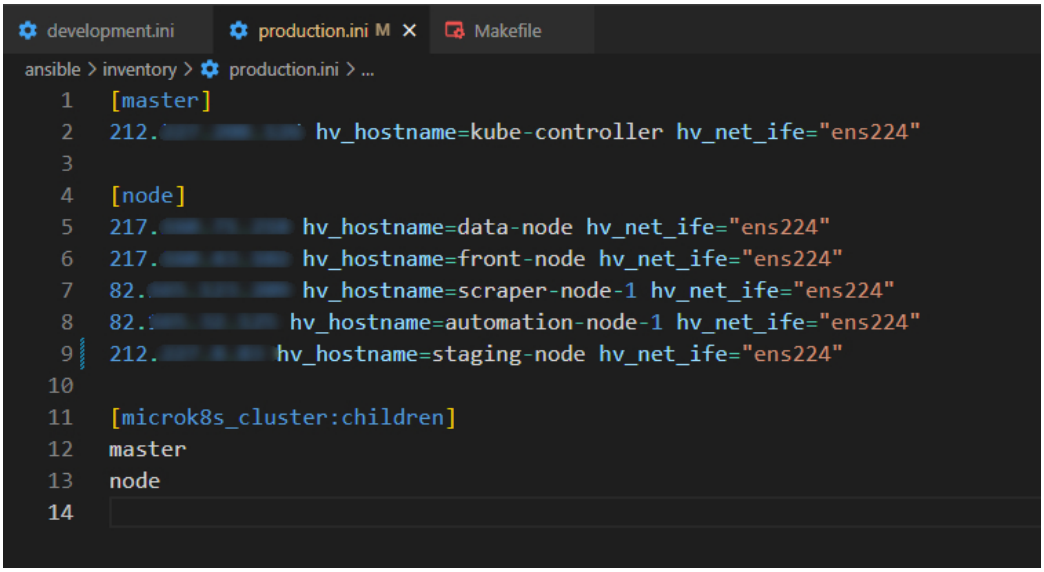


Figure 22: The Ansible hosts configuration

Note that we also use a hosts file for development where we test the setup extensively before trying it on production as seen from the figure just above.

4.3 Deployment process

We will now explain the approach we adopted to continuously integrate, deploy and deliver our application as per the best practices of DevOps previously mentioned in **Chapter 3: State of the art**. The deployment is currently supported on two different environments; the staging environment where we make sure our application works as intended, and of course the production environment.

4.3.1 Adopted strategy

Since we have multiple microservices, it would not be ideal to interact with our Kubernetes cluster from all of them whenever we make a change. For that reason we use the following steps to deploy our application.

- On code changes on any microservice, we create a new Docker image that we push to a custom container registry.
- Once the build is finished, we trigger a multi-project pipeline to a central repository we will call ilg-controller.

- The central repository will interact with our Kubernetes cluster to make the final deployment to the correct environment using a custom Helm package that we create.

This is further explained in detail in the following sub sections.

4.3.2 Linking the cluster to GitLab

Since we have a Kubernetes cluster and we also use GitLab, we will link the two using the GitLab Kubernetes Agent, or KAS for short.

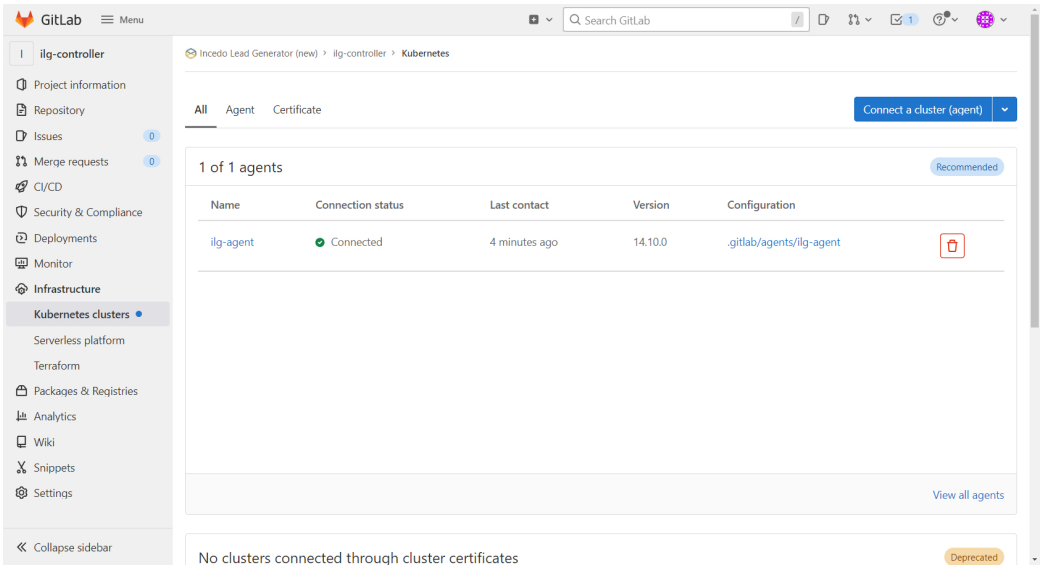


Figure 23: GitLab Kubernetes Agent

4.3.3 Continuous Integration

When we as developers make code changes to the main branch of any microservice, tests are automatically executed. If the tests are validated, a new Docker image that incorporates the changes we make will be pushed to the container registry and our deployment pipeline will then be triggered. In other words, the new code will be integrated and that actually marks the end of this step.

```yaml
# ------------------------- buildPublish ------------------------- #
.export-image-version: &get-image-version ···

build-publish-image:
  extends: .docker
  stage: buildPublish
  tags:
    - test
  rules:
    - if: '$CI_COMMIT_REF_NAME == "main" || $CI_COMMIT_REF_NAME == "staging"'
  script:
    - *get-image-version
    - make build-and-push

# ------------------------- deploy ------------------------- #
deploy-to-staging:
  stage: deploy
  rules:
    - if: '$CI_COMMIT_REF_NAME == "staging"'
  variables: ···
  trigger:
    project: ilg/ilg-controller
    branch: main
    strategy: depend
    forward:
      yaml_variables: true
      pipeline_variables: false
```

Figure 24: Extract from the CI pipeline

4.3.4 Continuous Deployment

Our ilg-controller repository (where the agent is configured) chooses the correct microservice to deploy depending on the trigger input. It then sets the necessary environment variables for it and installs it using a custom Helm package we've created for the deployment.

```
make
deploy: check-args set-args ## Deploy the application using helm
	helm upgrade \
	--install \
	--namespace $(ENVIRONMENT) \
	--create-namespace \
	--set image.repository='$(REMOTE_REGISTRY)/$(SERVICE_NAME)' \
	--set image.tag='$(CV_IMAGE_TAG)' \
	--set image.containerPort='$(CONTAINER_PORT)' \
	--set commitHash='$(COMMIT_HASH)' \
	--set resources.limits.cpu='$(RES_LIMIT_CPU)' \
	--set resources.limits.memory='$(RES_LIMIT_MEMORY)' \
	--set resources.requests.cpu='$(RES_REQ_CPU)' \
	--set resources.requests.memory='$(RES_REQ_MEMORY)' \
	--set nodeSelector."kubernetes\.io/hostname"='$(NODE_NAME)' \
	--set tolerations[0].key='dedicated',tolerations[0].value='my-pool',tolerat
	--set appLabels.environment='$(ENVIRONMENT)' \
	$(CV_ARGS) \
	$(CV_RELEASE_NAME) ilg-helm-charts/ilg-service

```

Figure 25: Extract from the deployment Makefile

4.3.5 Custom Helm Package

Since we agreed to use Helm (refer to **Chapter 2: State of the art**), we've created a package -or in terms of Helm- a chart of our own to manage the Kubernetes cluster resources.

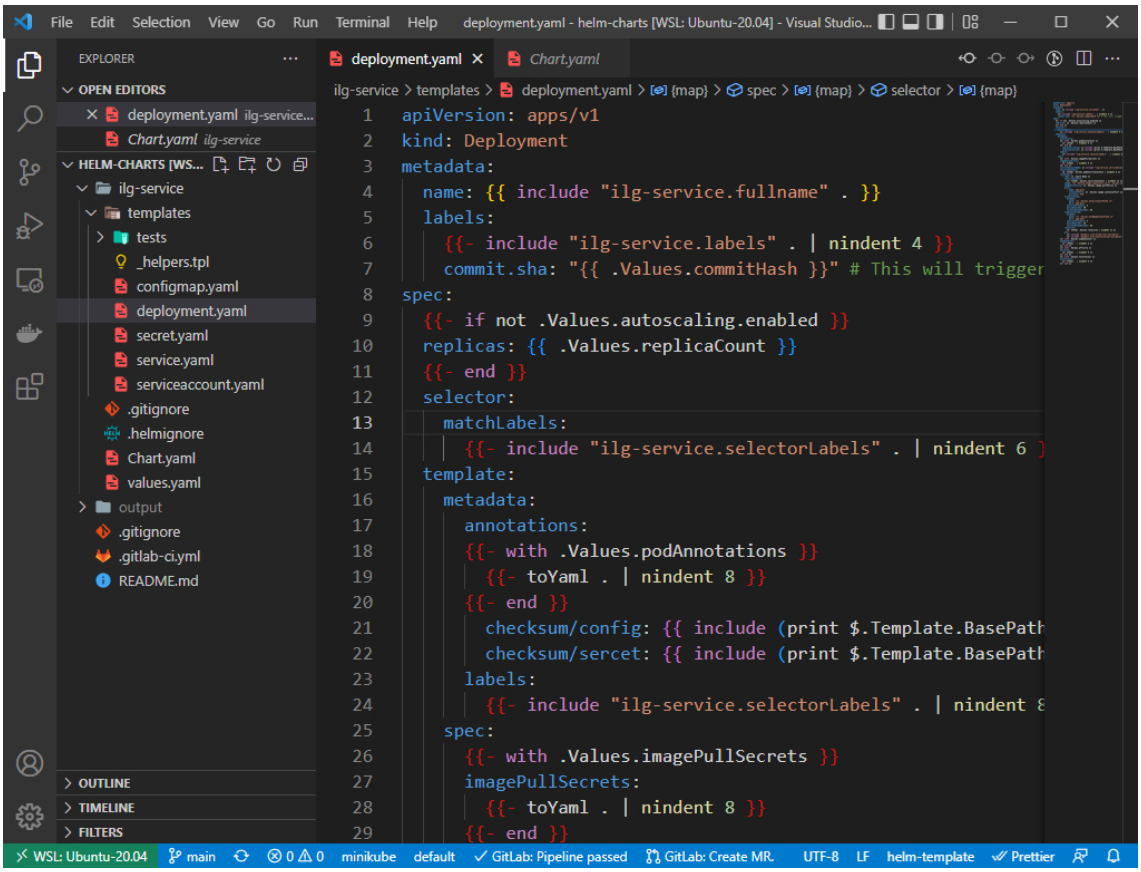


Figure 26: Extract from the custom Helm Chart

We even integrate a CI/CD pipeline for the Helm chart itself so that new packages are built automatically whenever we make changes to the manifest template files.

```yaml
# -------------------- buildPackage ---------------------- #
build-ilg-service:
  extends: .helm
  stage: buildPackage
  tags:
    - test
  rules:
    # https://regexr.com/6im81
    - if: '$CI_COMMIT_REF_NAME == "main" || $CI_COMMIT_REF_NAME =~ /^v\d+\.\d+\.\d+-?[a-zA-Z0-9_.-]*$/'
  script:
    - *get-package-version
    - helm package ./ilg-service/ --destination ./output
  artifacts:
    expire_in: 1 week
    paths:
      - output
```

Figure 27: Extract from the custom Helm package's pipeline

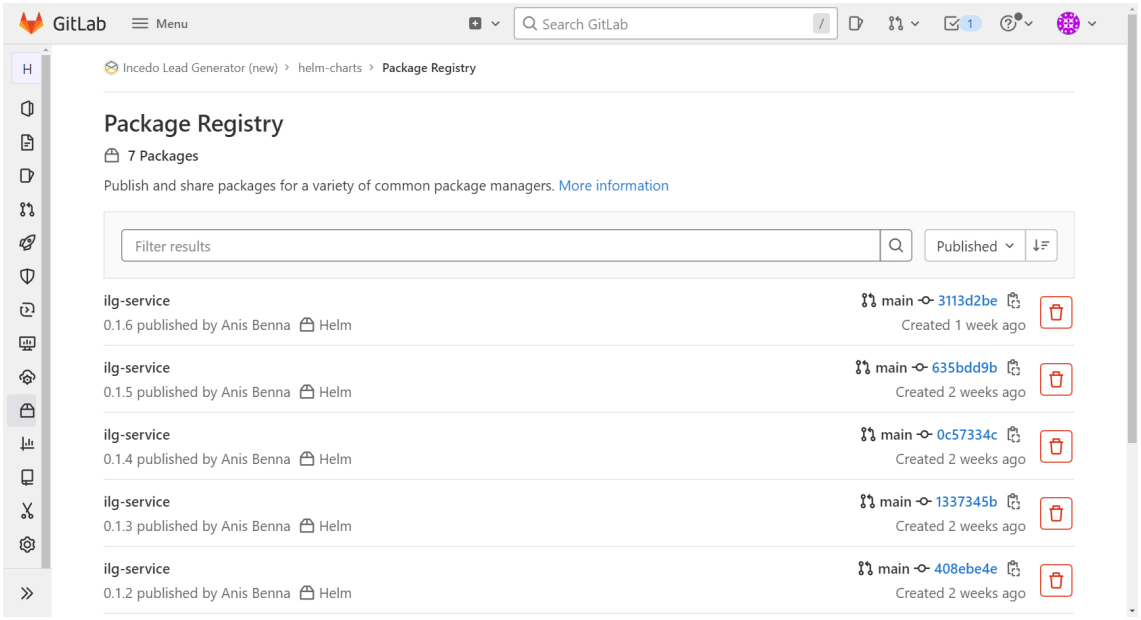


Figure 28: GitLab's package registry

4.3.6 Deployment result

And now if we trigger the CI/CD pipeline of all of our microservices (please see **Appendix c**), we can see the application dashboard deployed! In fact, we now have two live environments that are publicly available:

- **Production:** `https://lead-generator.incedo.net`
- **Staging:** `https://staging.lead-generator.incedo.net`

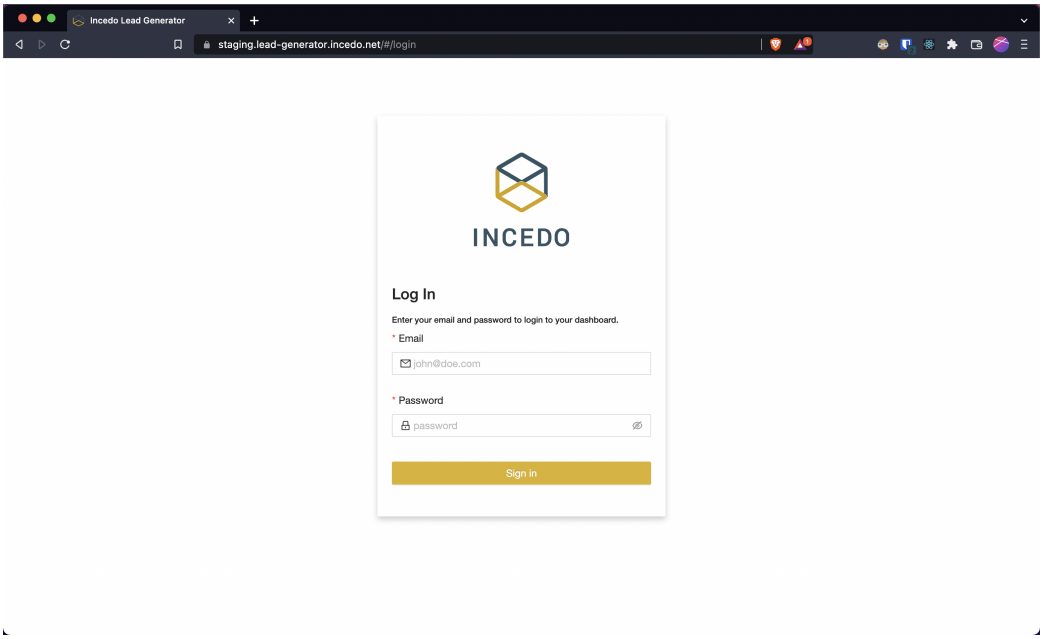


Figure 29: ILG Login page

When we login, the dashboard with the list of campaigns is the first thing we notice. We can see how many active or inactive campaigns we have and also some useful statistics about how well our campaigns are doing.

We can also manage our campaigns by adding, editing or deleting them as per the defined requirements. See **Appendix a** for more details.

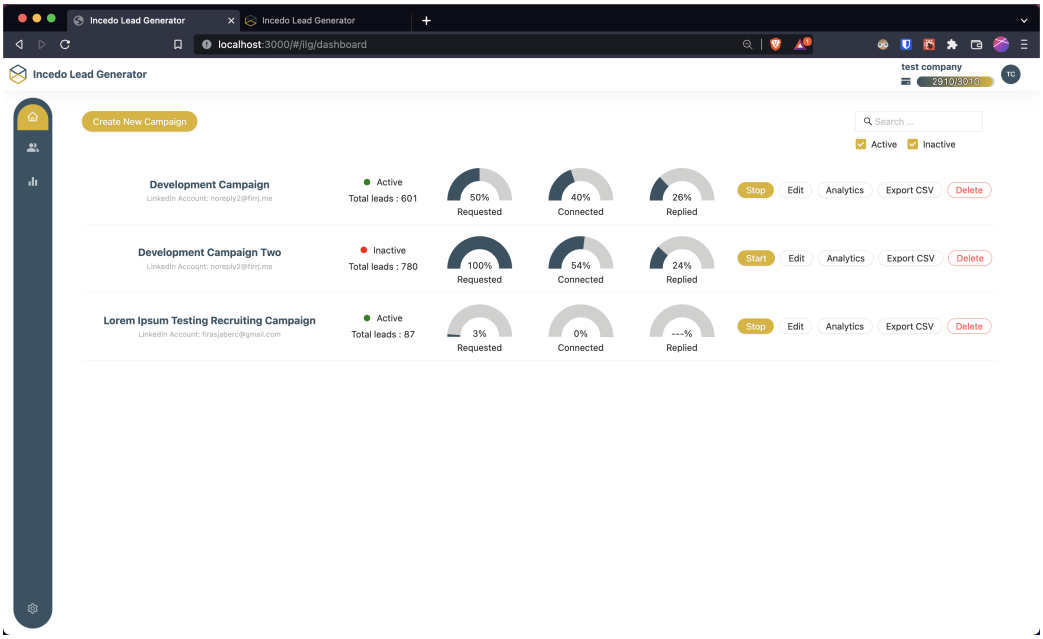


Figure 30: Campaigns overview page

We additionally have the option to manage our users as well as their credits. Also refer to **Appendix b** for more screenshots.

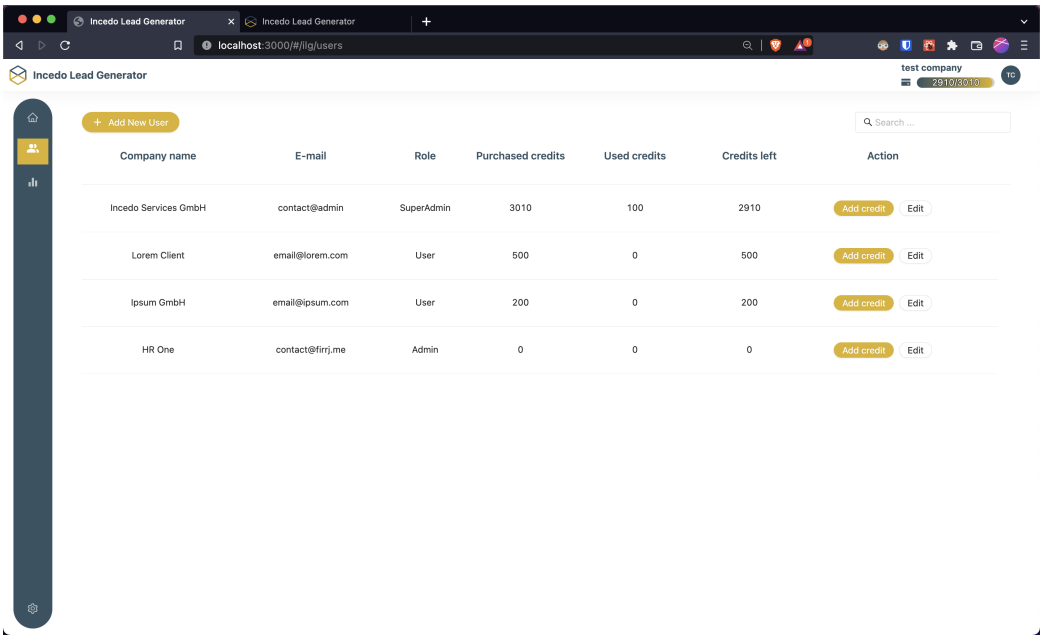


Figure 31: View users page

In order to provide visibility to see whether leads are generated, we build on the legacy code by using a records system which is basically the increment of invites sent or leads generated each day.

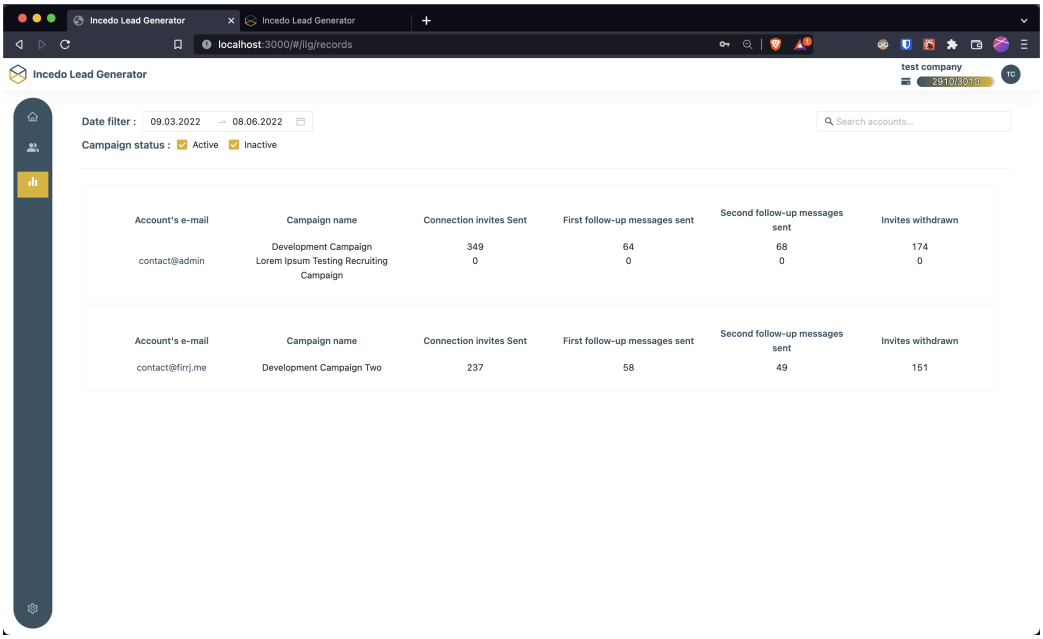


Figure 32: Campaigns Records page

As seen from the screenshot, it is not easy on the eyes and the UI is very primitive.

We therefore implement an interactive good looking analytics dashboard with ChartJS in order to make it very easy for our client to keep an eye out on their campaigns and study their conversion rate to make changes if needed to number of invites and follow ups sent per day to improve the leads conversion.

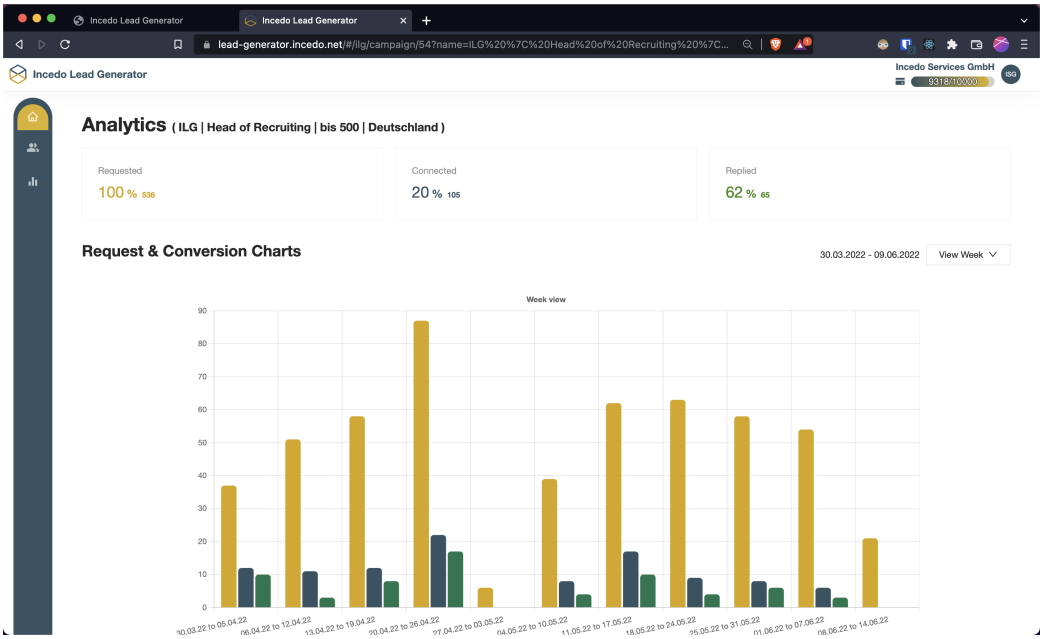


Figure 33: Campaigns Analytics page

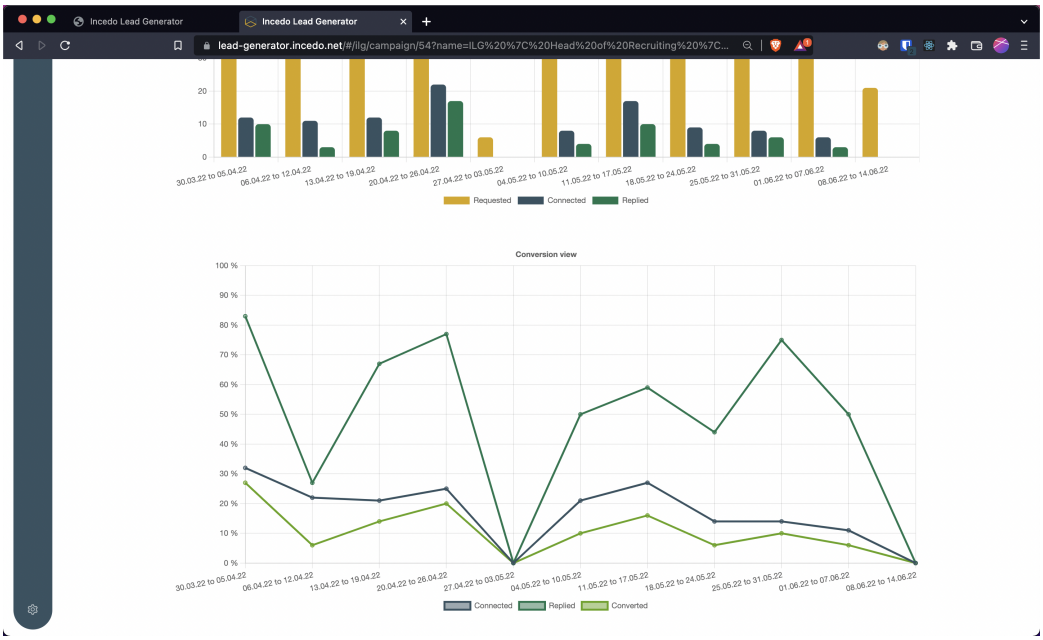


Figure 34: Campaigns Analytics page

4.4 Monitoring

The visibility definitely gets way better with dashboards. Therefore, we also setup such monitoring solutions for our whole cluster. The things we monitor vary from the cluster nodes' CPU and memory usage, to the logs of our microservices.

The first thing we do is exposing the dashboard provided by Kubernetes using manifest resources and kubectl. Please see **Appendix d**.

4.4.1 Monitoring the cluster nodes

By using and configuring the Kubernetes community's Helm packages, we can very easily deploy what's called the **Kubernetes Prometheus Stack** which will give us access to the Grafana dashboard and to the Prometheus UI dashboard.

Prometheus

Prometheus is an open-source software used for event monitoring and alerting. The way we do that is by collecting different kind of metrics and by configuring the automatic sending of alerts using rules in case of critical errors.

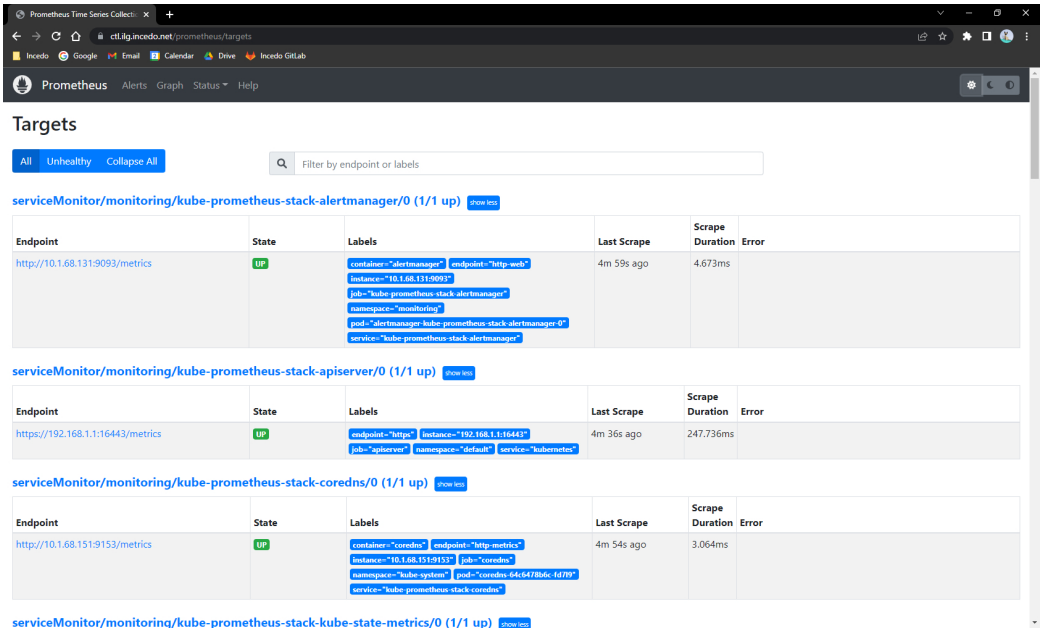


Figure 35: Viewing the metrics endpoints with Prometheus UI

Grafana

Grafana is an open-source interactive visualization web application. We use it here to monitor our cluster nodes, and most importantly, the metrics of our application collected by Prometheus.

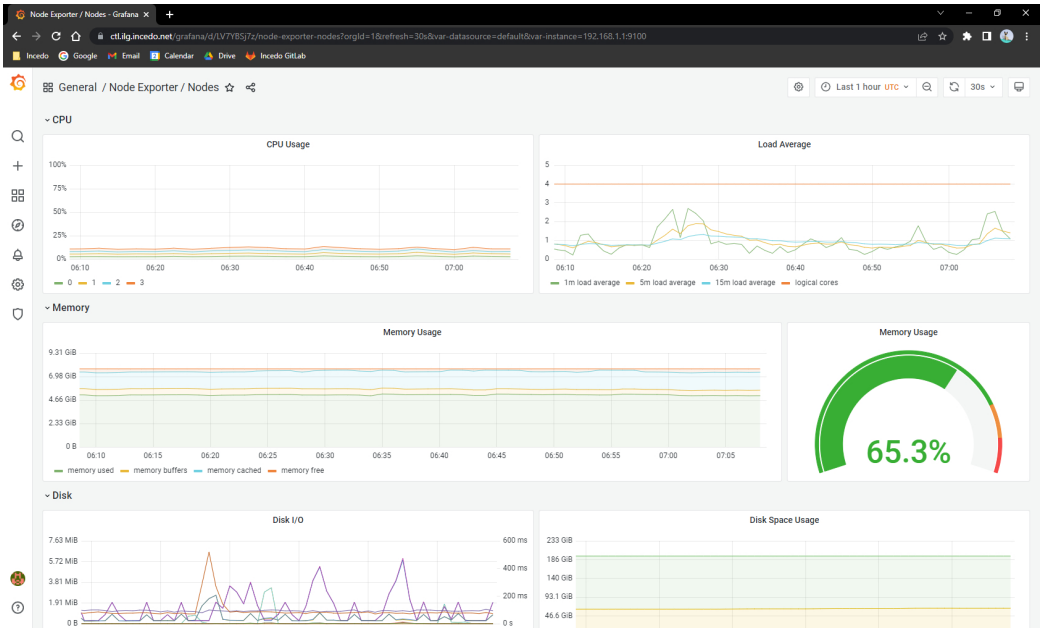


Figure 36: Monitoring the cluster nodes with Grafana

4.4.2 Monitoring the logs

In order to monitor the logs of our different microservices as well as any other applications running on the cluster efficiently, we need to

- Collect the logs from the different Pods in the cluster
- Process and parse the logs to a uniform format, often json
- Send the logs to an indexing engine for easy querying
- We may also optionally create dashboards for our logs

The best way to go about doing this is by using the EFK Stack or more expressively the Elasticsearch, Fluentd and Kibana stack.

Elasticsearch & Kibana

Elasticsearch is many things: it is often referred to as search engine, analytics engine or even a database and all of that is true. It can process a huge amount of logs with relatively minimal resources and is therefore the optimal choice for our cluster.

Kibana is a data visualization dashboard exclusively used with Elasticsearch. It is used almost always whenever there is an Elasticsearch instance involved.

By using the open-source **Elastic Stack Kubernetes Helm Charts**, we can create our Elasticsearch and Kibana setup in the cluster.

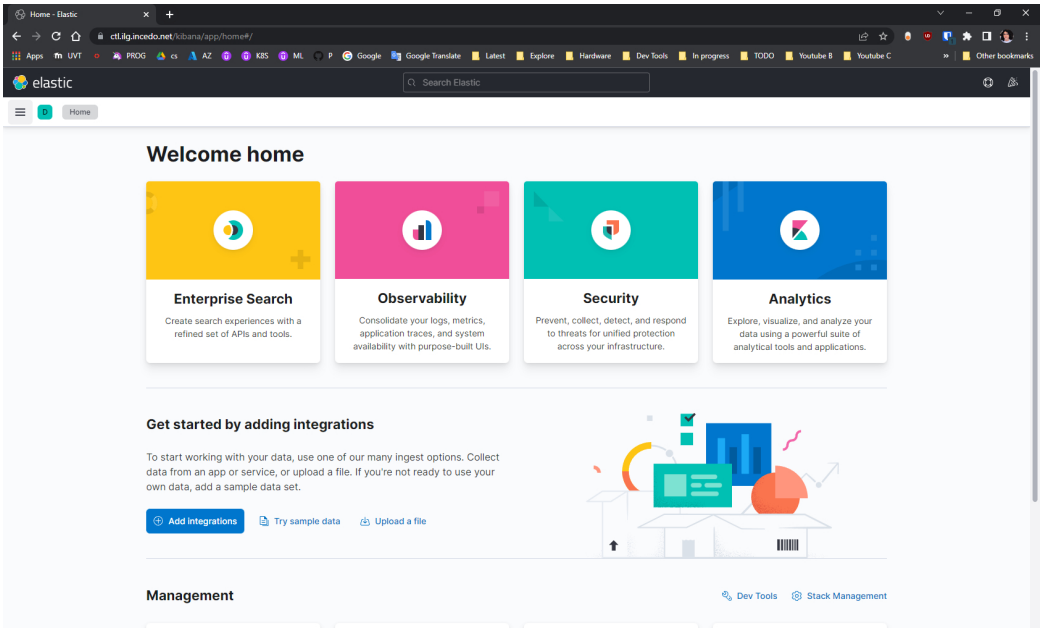


Figure 37: Kibana dashboard home screen

Fluentd

Fluentd on the other hand is a data collector. It runs in the background with the sole purpose of delivering the logs of all of our applications to any other service in a uniform way we specify. We configure it to get the logs from the standard output of our Pods, parse them to json and send them to Elasticsearch.

We deploy Fluentd the manifest way, as opposed to using Helm Charts.

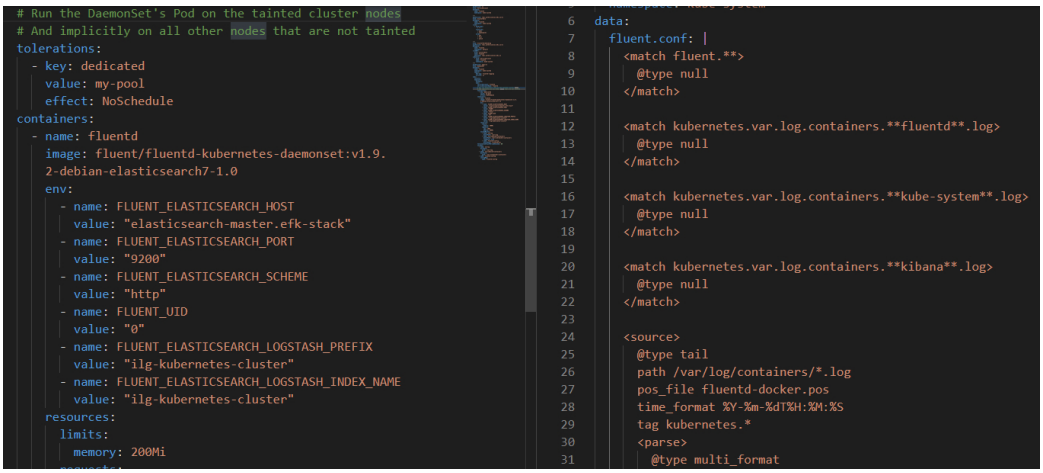


Figure 38: Fluentd configuration extract

And now we can finally use some of the advanced features of Elasticsearch such as the filtering and sorting of our logs from the Kibana dashboard.

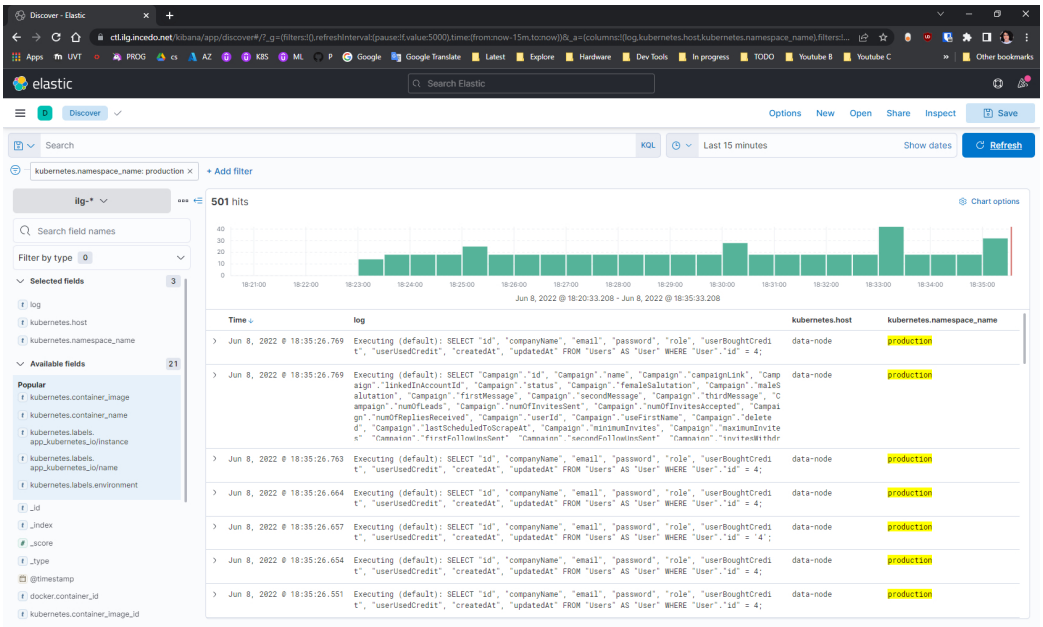


Figure 39: Indexing the logs in Kibana

Conclusion

A successful project comes from a successful development environment as well as a clean production environment with good and well thought of monitoring solutions. For this reason, we spent quite a good amount of time setting up the infrastructure in a clean and scalable way using various tools and scripts which all conform to the modern ways of how DevOps should be done. In the next chapter, we will list all of the technologies that we've used and also talk about the major difficulties that we've encountered in the whole making of the project.

Chapter 5

Realization: Environment

Introduction

A project can always look simple from the outside, however if we dive deeply into how most software is made, we can see that it's not that simple of a process. Therefore, this chapter will list all of the technologies that we've used and also some of the major technical difficulties we've come in contact with during the realization of our project.

5.1 Technologies

This part is reserved for the presentation of all of the software used in the realization of the project and includes but is not limited to; programming languages, frameworks, technologies, etc...

For a comparative analysis on some of our choices, see **Chapter 2: State of the art**.

- **Git:**

Figure 40: Logo of Git

- **Docker:**

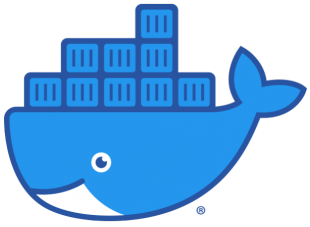


Figure 41: Logo of Docker

- **Playwright:**

Figure 42: Logo of Playwright

- **NestJS:**
  [9] A framework for building efficient, scalable Node.js web applications.

Figure 43: Logo of NestJS

- **ReactJS:**
  A front-end JavaScript library that's often considered as a framework.

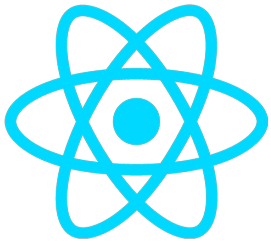


Figure 44: Logo of ReactJS

- **PostgreSQL:**

Figure 45: Logo of PostgreSQL

- **Sequelize:**
  [10] A modern TypeScript and Node.js ORM for SQL databases.

Figure 46: Logo of Sequelize

- **GraphQL:**
  [11] A query language for APIs and a runtime for fulfilling those queries with existing data.

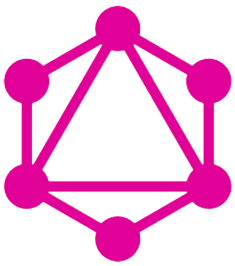


Figure 47: Logo of GraphQL

- **GitLab:**
  A DevOps software that allows secure collaboration and operations in a single application.

Figure 48: Logo of GitLab

- **Nginx:**
  [12] A multi-purpose web server can also be used as reverse proxy, load balancer, mail proxy and HTTP cache.

Figure 49: Logo of Nginx

- **Renovate:**
  A software that provides automatic dependency updates with support for multiple languages.

Figure 50: Logo of Renovate

- **Docker compose:**
  A helper tool to run applications with multiple Docker containers using a yaml format. Often used in development.

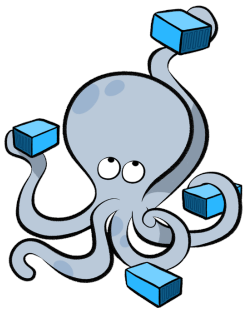


Figure 51: Logo of Docker Compose

- **Visual Studio Code:**
  A code editor that's used as an entire development environment with the help of extensions.

Figure 52: Logo of VSCode

- **Linux:**
  A Unix-like operating system for common daily use and more commonly for servers.

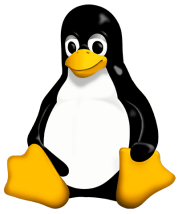


Figure 53: Logo of Linux

- **Make:**
  GNU Make is used extensively throughout the application to save useful commands for the ease of maintainability.

Figure 54: Logo of Makefile

- **MicroK8s:**
  [13] MicroK8s is a simple production-grade conformant K8s. It is Lightweight and focused.

Figure 55: Logo of MicroK8s

- **Kubernetes:**
  The most powerful tool for managing containerized workloads in the cloud.

Figure 56: Logo of Kubernetes

- **Ansible:**
  An open-source software for provisioning, configuring and managing infrastructure.

Figure 57: Logo of Ansible

- **Squid Proxy:**
  [14] Squid is a caching proxy for the Web that supports HTTP, HTTPS, FTP and more.

Figure 58: Logo of Squid Proxy

- **GitLab agent for Kubernetes:**
  [15] A secure and reliable way to attach a Kubernetes cluster to GitLab.

Figure 59: Logo of GitLab Agent for Kubernetes

- **Helm:**
  Helm is the most popular package manager for Kubernetes.

Figure 60: Logo of Helm

- **Prometheus:**
  Prometheus is a software application used for monitoring and alerting.

Figure 61: Logo of Prometheus

- **Grafana:**
  Grafana is an open source solution for monitoring and analytics.

Figure 62: Logo of Grafana

- **LaTeX:**
  [16] A high-quality document preparation and typesetting system for technical grade documents.

Figure 63: Logo of The LaTeX Project

- **Elasticsearch:**
  Elasticsearch is a search engine based on the Lucene library. Is it often used as part of a stack alongside Kibana and other tools.

Figure 64: Logo of elasticsearch

- **Kibana:**
  [17] Kibana is an free and open frontend application that sits on top of the Elastic Stack, providing search and data visualization capabilities for data indexed in Elasticsearch.

Figure 65: Logo of Kibana

- **Fluentd:**
  [18] Fluentd is an open source data collector, which lets you unify the data collection and consumption for a better use and understanding of data.

Figure 66: Logo of Fluentd

- **Microsoft Teams:**
  Microsoft Teams is a proprietary business communication platform developed by Microsoft. We use it to collaborate and share information.

Figure 67: Logo of Microsoft Teams

- **Asana:**
  Asana is a work management platform available on web and and mobile. It is used internally by Incedo for various work and project management.

Figure 68: Logo of Asana

- **Signal:**
  Signal is a cross-platform centralized e2e encrypted instant messaging service. It is used by the team to share sensitive information.

Figure 69: Logo of Signal

5.2 Difficulties encountered

5.2.1 Infrastructure as code

This not per say a technical difficulty as it is more of a limitation from the side of Ionos cloudpanel; the service of IONOS Cloud that we use the provision the needed resources. To have a full DevOps approach, even the infrastructure should be managed in code -also known as IaC (Infrastructure as Code)- using tools such as Terraform. However, Ionos cloudpanel does not provide that functionality. Therefore we had to scrape off the idea and simply create the virtual machines from the UI every time we need them.

5.2.2 Stateful Kubernetes

Another issue we had is data persistence in Kubernetes. Kubernetes is -by design- used for stateless applications. However, since we're using an Elasticsearch database, we definitely needed a way to persist our data across system reboots or crashes.

First of all, we used IONOS -our cloud provider- to create the physical Block Storage in question that we used as the data source. We started with an SSD of 40 Gigabytes then scaled it up to 80 Gigs.

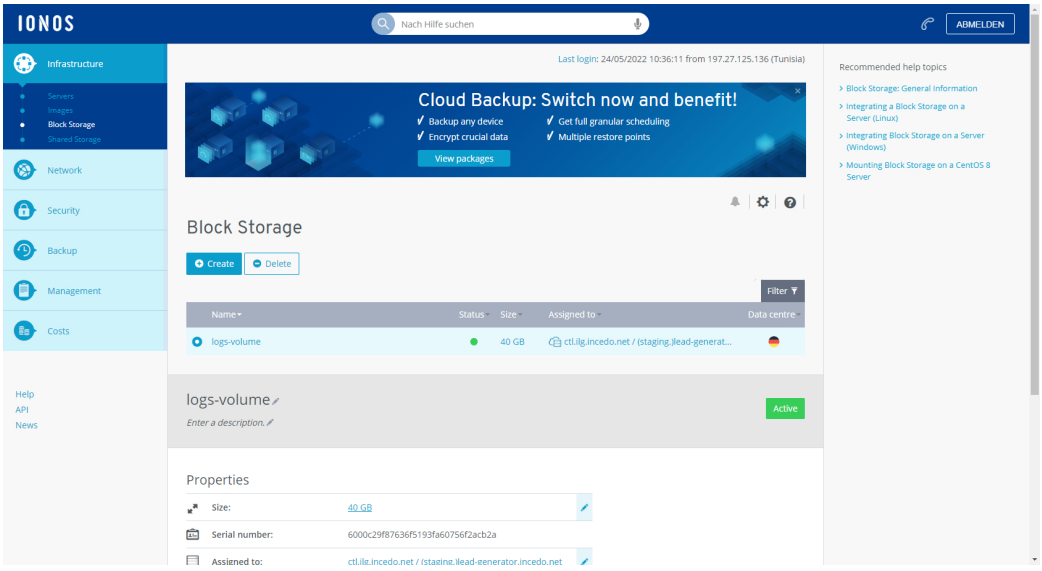


Figure 70: Creating a block storage in IONOS

Secondly, we had to logically mount the previously created block storage to a directory such as **/mnt/block** inside the controller virtual machine.

From there, we create a self-managed NFS Server on the host system that we export to the private network or in other words to all of the cluster nodes.

Now, for the stateful cluster components to use the NFS Server, we needed to setup an NFS client provisioner inside the Kubernetes cluster itself. This was as simple as using the **NFS Subdir External Provisioner** Helm Chart.

Lastly, our cluster components are now allowed to provision persistent and dynamic storage resources whenever they need it.

5.2.3 Automatic Testing

One of the keys to writing maintainable and error-prone software is to write tests across all the codebase to ensure no breaking changes happen when adding new pieces. It also makes debugging way more manageable later on. In our case however, it's challenging to write end-to-end or integration tests because most of the scraping and automation logic is dependent on interacting with LinkedIn via the browser. And since LinkedIn sets limits on how many leads profiles we can view per day, mocking and running those tests will make our scraping and automation more detectable. We still nonetheless have some parts of the codebases automatically tested such as modules integrations, helpers and authentication endpoints.

```
yarn run v1.22.17
$ jest src/modules/campaign-scraper/state-handler.spec.ts src/modules/send-invite/send-invite.service.spec.ts src/core/helpers/help
ers.spec.ts src/app.controller.spec.ts src/modules/send-invite/send-invite.service.spec.ts
 PASS  src/core/helpers/helpers.spec.ts
 PASS  src/app.controller.spec.ts
 PASS  src/modules/send-invite/send-invite.service.spec.ts
 PASS  src/modules/campaign-scraper/state-handler.spec.ts

Test Suites: 4 passed, 4 total
Tests:       152 passed, 152 total
Snapshots:   0 total
Time:        1.545 s, estimated 3 s
Ran all test suites matching /src\/modules\/campaign-scraper\/state-handler.spec.ts|src\/modules\/send-invite\/send-invite.service.
spec.ts|src\/core\/helpers\/helpers.spec.ts|src\/app.controller.spec.ts|src\/modules\/send-invite\/send-invite.service.spec.ts/i.
✨  Done in 5.63s.
```

Figure 71: ilg-scraper tests

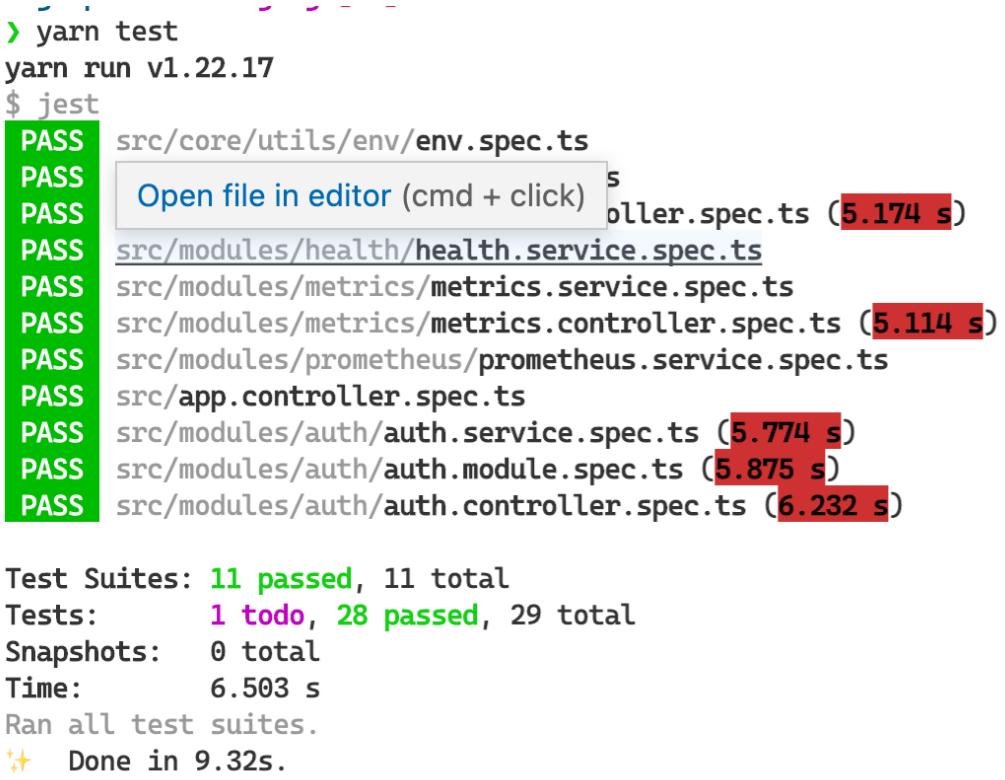


Figure 72: ilg-api tests

Conclusion

In this chapter, we listed all of the technology stack that we used for ILG as well as some of the most weighty issues we've encountered. We also listed some our solutions for the ones we could solve following the best practices of software development.

General Conclusion

Throughout the period of the internship within Incedo Services GmbH, we were committed to contributing to the growth of the company by making the Incedo Lead Generator software more successful. The objective of this project was to add value to the existing software solution by, first of all, developing more features as well as fixing bugs and, second of all, by migrating the whole infrastructure of the monolithic application to use microservices instead. The microservices in question were deployed to a self-managed instance of Kubernetes that we of course had to create and manage ourselves. The whole setup also adheres to the best practices of DevOps such as automated pipelines and continuous monitoring.

The realization of this project has allowed us to face numerous constraints; mainly time constraints, technological constraints and also COVID-19 constraints that basically forced us to do this internship fully remotely. We nevertheless had a huge chance to broaden our knowledge pool in the field of IT in general and especially in the growing field of DevOps. We were very lucky that the Incedo team encourages curiosity, learning and experimenting on the various new technologies which has motivated us to explore and develop our technical skills more and more. Some of the technologies that we've learned thanks to this project are GitLab's advanced features, Ansible, Kubernetes, Helm, Prometheus, Grafana, Nginx, GraphQL and many more...

This internship has been a great opportunity to apply our theoretical and practical knowledge acquired at the Higher Institute of Technological Studies of Nabeul and to build much more on it. Working at Incedo has truly allowed us to shine, especially considering how very pleasant it was to communicate with the team and we are extremely happy to have been recruited as permanent members of this wonderful family.

With that said, there are a couple of prospects looming on the horizon for improving the work we've done. One thing that comes to mind is to create modern looking dashboards in Kibana for the microservices' logs that are already streamed to Elasticsearch in order to get more visibility on the errors, warnings or anything that's happening at a business logic layer inside the microservices for non IT people.

To conclude, it feels like we definitely succeeded our internship in the truest meaning of the word and we are ready to keep on building and enhancing ourselves as well as to contribute hugely to the industry.

Webography

[1] Ronak Ganatra. Graphql vs rest apis, 2018. https://graphcms.com/blog/graphql-vs-rest-apis.

[2] Incdeo Services GmbH. About incedo, 2022. https://incedo.de/.

[3] Microsoft Azure. Devops overview, 2022. https://azure.microsoft.com/en-us/overview/what-is-devops/#devops-overview.

[4] Arthur Busser. What is a container?, 2020. https://www.padok.fr/en/blog/container-docker-oci.

[5] Beth Pariseau Emily Mell. What is container management?, 2021. https://www.techtarget.com/searchitoperations/definition/container-management-software.

[6] Red Hat. What is a ci/cd pipeline?, 2022. https://www.redhat.com/en/topics/devops/what-cicd-pipeline.

[7] Amazon Web Services. What are microservices?, 2022. https://aws.amazon.com/microservices/.

[8] Wikipedia. Ionos, 2022. https://en.wikipedia.org/wiki/Ionos.

[9] Nest.js. The official website, 2022. https://nestjs.com/.

[10] Sequelize. The official website, 2022. https://sequelize.org/.

[11] GraphQL. The official website, 2022. https://graphql.org/.

[12] Wikipedia. Nginx definition, 2022. https://en.wikipedia.org/wiki/Nginx.

[13] MicroK8s. The official website, 2022. https://microk8s.io/.

[14] Squid Cache. The official website, 2022. http://www.squid-cache.org/.

[15] Senior Product Manager at GitLab Viktor Nagy. A new era of kubernetes integrations on gitlab.com, 2021. https://about.gitlab.com/blog/2021/02/22/gitlab-kubernetes-agent-on-gitlab-com/.

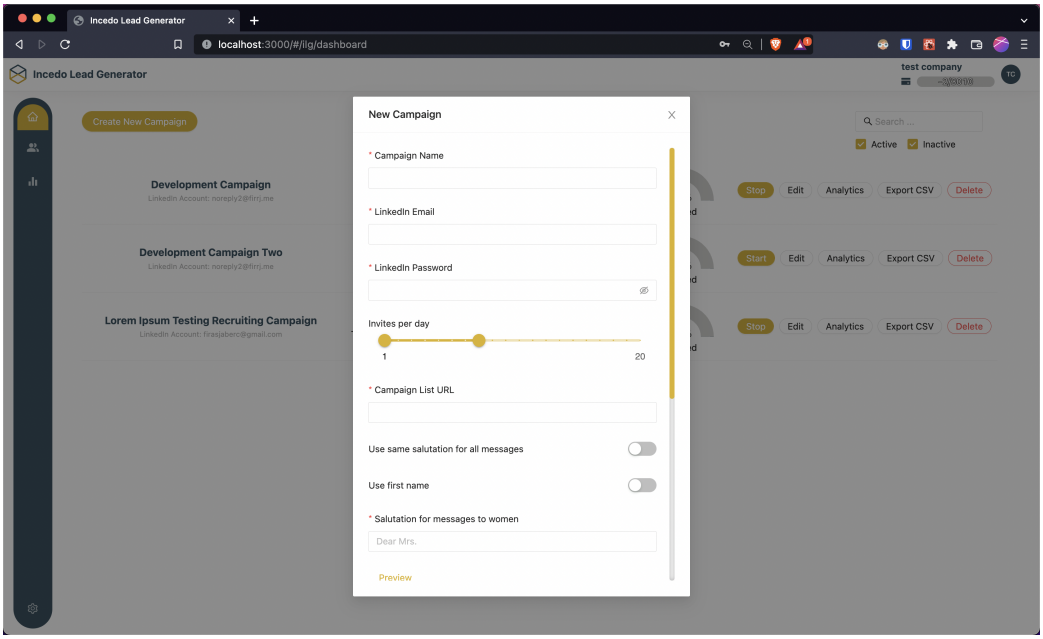
[16] The LaTeX Group. The official website, 2022. https://www.latex-project.org/.

[17] Elastic. The official website, 2022. https://www.elastic.co/what-is/kibana.

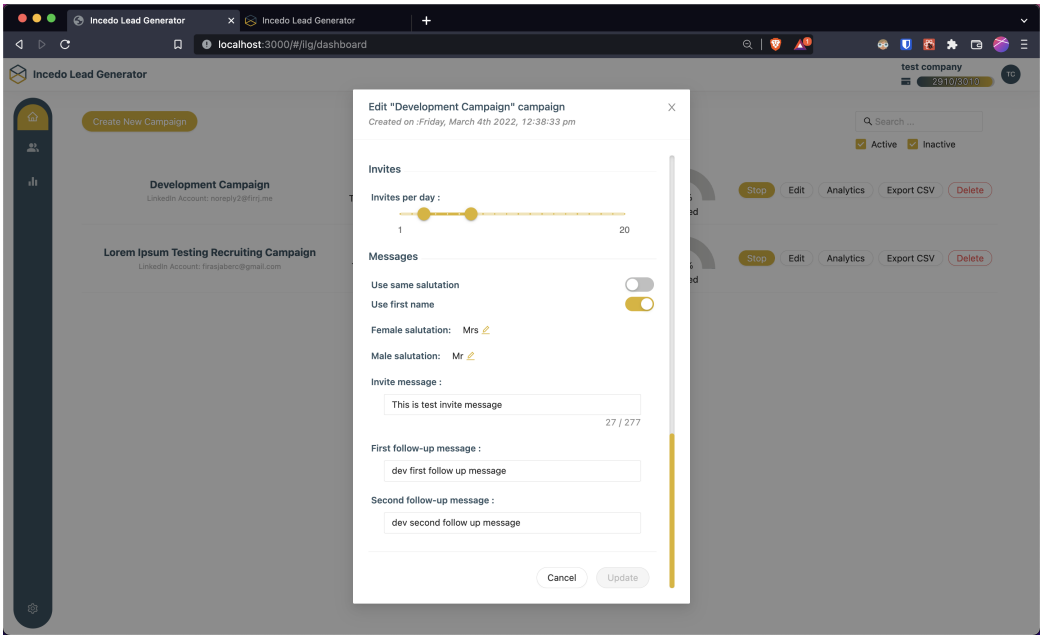
[18] Fluentd. The official website, 2022. https://www.fluentd.org/architecture.

Appendices

ILG Campaign management



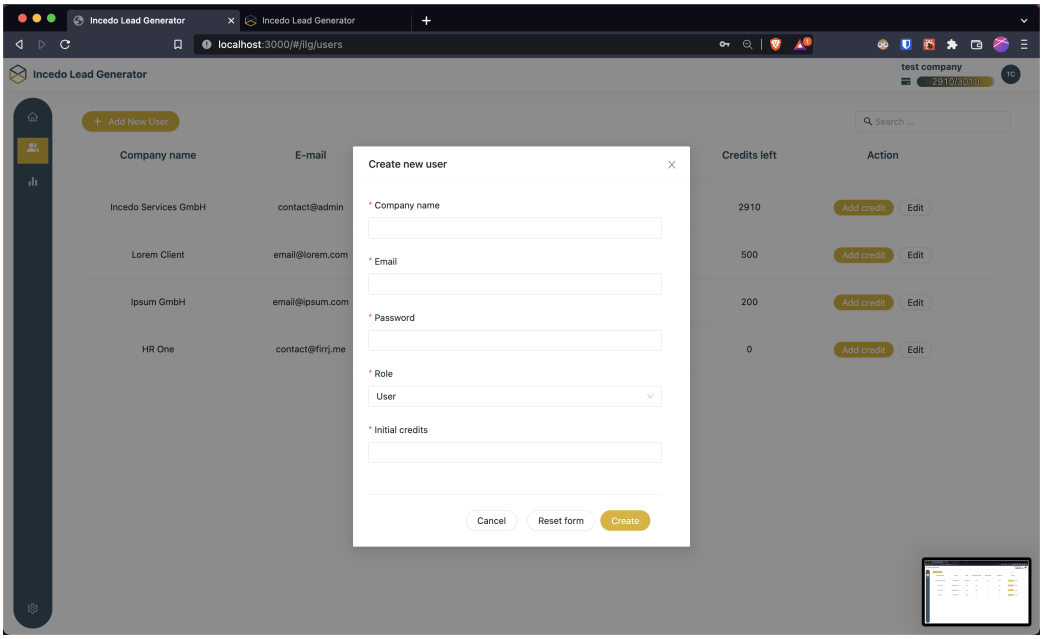
Add campaign form



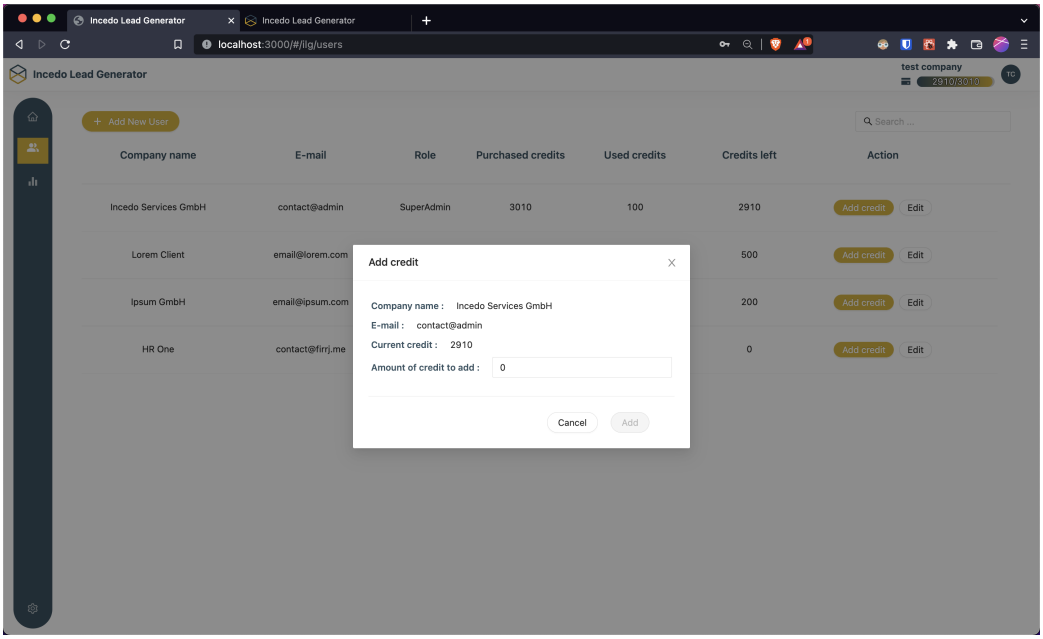
Edit campaign form

a

ILG User management



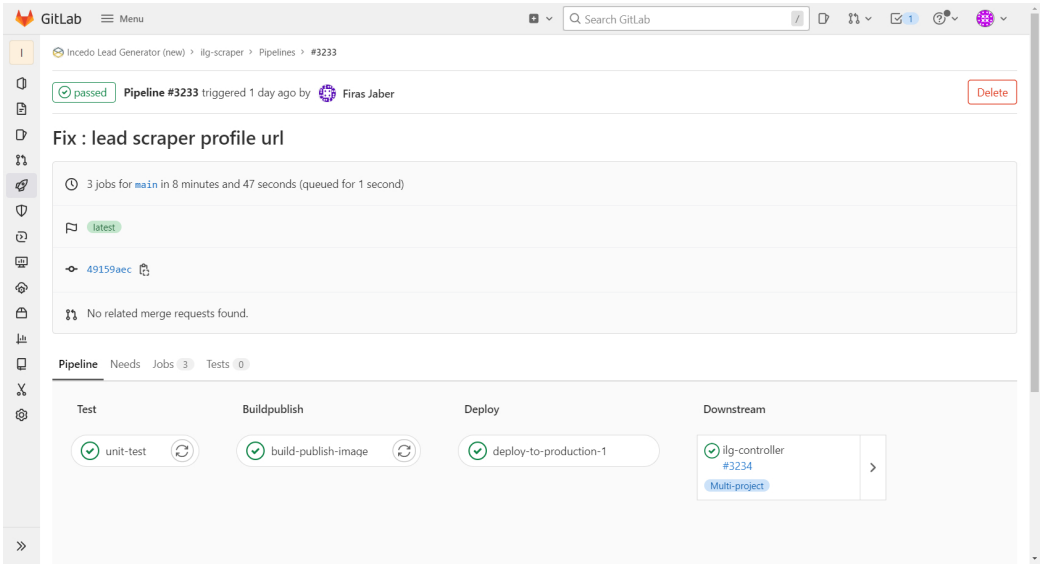
Add User form



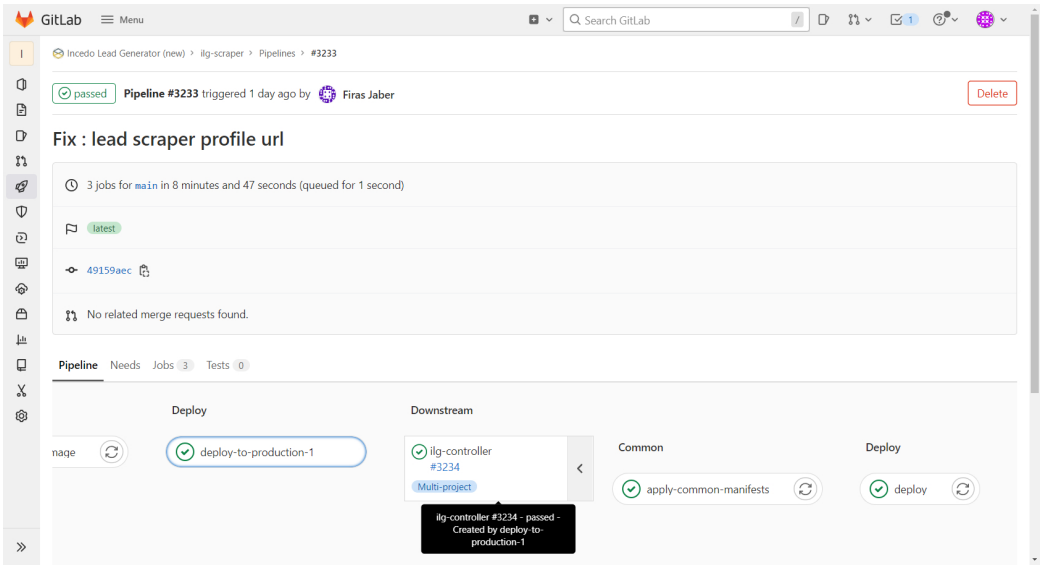
Users Add credit form

b

CI/CD pipeline of one of the microservices



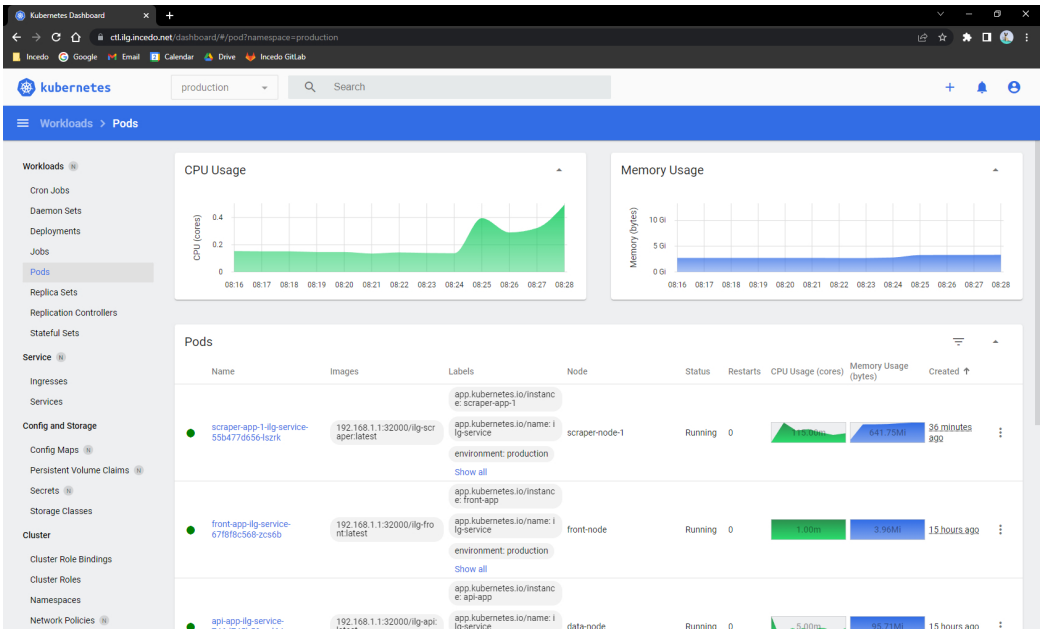
Part 1 - Same project pipeline



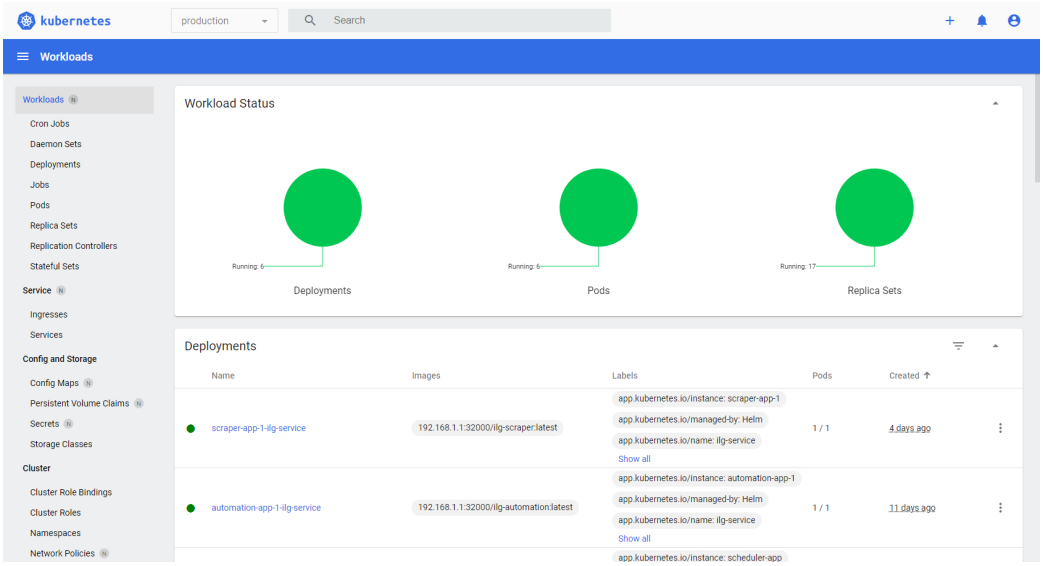
Part 2 - Multi-project pipeline

Kubernetes dashboard

The Kubernetes dashboard is a web based UI for Kubernetes clusters. It allows the management of applications running in the cluster.



Monitoring the pods with the Kubernetes dashboard



Monitoring the workloads with the Kubernetes dashboard

d