



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

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

VM Virtual Machine

IaC Infrastructure as Code

Table of Contents

Glossary	iii
General Introduction	1
1 Context of the work	2
Introduction	3
1.1 General framework of the internship	3
1.2 Company overview	3
1.2.1 About Incedo	3
1.2.2 Incedo's services	4
1.3 Stating the problem	4
1.4 Assessment of the case	5
1.4.1 Describing the work procedure	5
1.4.2 Criticizing the current state	5
1.4.3 Proposed solution	5
1.5 Development Methodology	6
1.5.1 Agile methodology	6
1.5.2 Scrum methodology	6
1.5.3 Kanban methodology	6
1.5.4 The choice for ILG	7
1.5.5 Unified Modeling Language	7

Conclusion	7
2 State of the art	8
Introduction	9
2.1 Automation and Web Scraping concepts	9
2.1.1 Automation	9
2.1.2 Web Scraping	9
2.1.3 Relationship between the two	9
2.2 DevOps	9
2.2.1 Definition	9
2.2.2 Lifecycle	10
2.2.3 Container management	11
2.2.4 CI/CD pipelines	11
2.3 Microservices	12
2.3.1 Definition	12
2.3.2 Characteristics of Microservices	12
2.3.3 Benefits of Microservices	13
2.4 Software Testing	14
2.4.1 Definition	14
2.4.2 Types of Tests used	14
2.5 Comparative Analysis	15
2.5.1 Version Control: Git vs SVN	15
2.5.2 Container management: Docker vs Podman	15
2.5.3 Browser automation: Puppeteer vs Playwright	16
2.5.4 Database: MongoDB vs PostgreSQL	17
2.5.5 Database: GraphQL vs REST	18
2.5.6 Deployment: Docker Swarm vs Kubernetes	19
2.5.7 Deployment management: Kubectl vs Helm	20

Conclusion	21
3 Analysis and specification of requirements	22
Introduction	23
3.1 Requirements	23
3.1.1 Dashboard	23
3.1.2 Lead generating	26
3.2 Conception	27
3.2.1 Architecture	27
3.2.2 Tasks & Queues	29
3.2.3 Scraper & Automation	31
3.2.4 Data Layers	36
Conclusion	37
4 Realization	38
Introduction	39
4.1 Hardware setup	39
4.1.1 Cloud provider	39
4.1.2 Kubernetes Cluster	41
4.1.3 Proxy Servers	42
4.2 Software setup	43
4.2.1 Manual configuration	43
4.2.2 Automating the process	43
4.3 Deployment process	45
4.3.1 Adopted strategy	45
4.3.2 Linking the cluster to GitLab	46
4.3.3 Continuous Integration	46
4.3.4 Continuous Deployment	47

4.3.5	Custom Helm Package	48
4.3.6	Our deployed application	50
4.4	Monitoring	57
4.4.1	Monitoring the cluster nodes	57
4.4.2	Monitoring the logs	58
4.5	Technologies	61
4.6	Difficulties encountered	69
4.6.1	Infrastructure as code	69
4.6.2	Stateful Kubernetes	69
	Conclusion	70

General Conclusion	71
---------------------------	-----------

Appendices

Kubernetes dashboard	a
--------------------------------	---

List of Figures

1	Logo of Incedo Services GmbH	3
2	Selection of Incedo clients	4
3	ILG project Kanban board	7
4	DevOps and the application lifecycle	10
5	Monolith to microservices example	12
6	Use case diagram	23
7	Overview of the ILG services	27
8	Cloud infrastructure of the application	29
9	Task queues	30
10	Diagram detailing the producers and the consumers	31
11	Campaign scraper sequence diagram	32
12	Lead scraper sequence diagram	33
13	Diagram detailing the producers and the consumers	34
14	Diagram detailing the producers and the consumers	35
15	Entity Relationship Diagram	36
16	Logo of IONOS	39
17	Ionos cloudpanel dashboard	40
18	Standard VM sizes in IONOS	41
19	RAM Optimized VM sizes in IONOS	41

20	Extract from the Ansible playbook	44
21	The Ansible hosts configuration	45
22	GitLab Kubernetes Agent	46
23	Extract from the CI pipeline	47
24	Extract from the deployment Makefile	47
25	Extract from the custom Helm Chart	48
26	Extract from the custom Helm package's pipeline	49
27	GitLab's package registry	49
28	Login page	50
29	Campaigns Overview page	51
30	Add Campaign form	51
31	Edit Campaign form	52
32	View Users page	52
33	Edit Campaign form	53
34	View Users page	53
35	Add User form	54
36	Users Add credit form	54
37	Campaigns Records page	55
38	Campaigns Analytics page	55
39	Campaigns Analytics page	56
40	Viewing the metrics endpoints with Prometheus UI	57
41	Monitoring the cluster nodes with Grafana	58
42	Kibana dashboard home screen	59
43	Fluentd configuration extract	60
44	Indexing the logs in Kibana	60
45	Logo of Git	61
46	Logo of Docker	61

47	Logo of Playwright	61
48	Logo of NestJS	62
49	Logo of ReactJS	62
50	Logo of PostgreSQL	62
51	Logo of Sequelize	63
52	Logo of GraphQL	63
53	Logo of GitLab	63
54	Logo of Nginx	64
55	Logo of Renovate	64
56	Logo of Docker Compose	64
57	Logo of VSCode	65
58	Logo of Linux	65
59	Logo of Makefile	65
60	Logo of MicroK8s	66
61	Logo of Kubernetes	66
62	Logo of Ansible	66
63	Logo of Squid Proxy	67
64	Logo of GitLab Agent for Kubernetes	67
65	Logo of Helm	67
66	Logo of Prometheus	68
67	Logo of Grafana	68
68	Logo of The LaTeX Project	68
69	Creating a block storage in IONOS	69
70	Monitoring the pods with the Kubernetes dashboard	a

List of Tables

1	Comparative study between Git and SVN	15
2	Comparative study between Docker and Podman	16
3	Puppeteer vs Playwright highlights	17
4	Comparative study between MongoDB and PostgreSQL	18
5	Comparative study between GraphQL and REST [5]	19
6	Comparative study between Docker Swarm and Kubernetes .	20
7	Comparative study between Kubectl and Helm	21
8	Create Campaign's use case textual description	25
9	All the application's Microservices	28
10	The application's tasks or jobs	30
11	Core cluster nodes	42
12	Scalable cluster nodes	42
13	List of proxy servers	42

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.” [6]



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 codebases as well as respond faster to bugs and know exactly what caused them in the first place.

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 to finish. They do this 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.

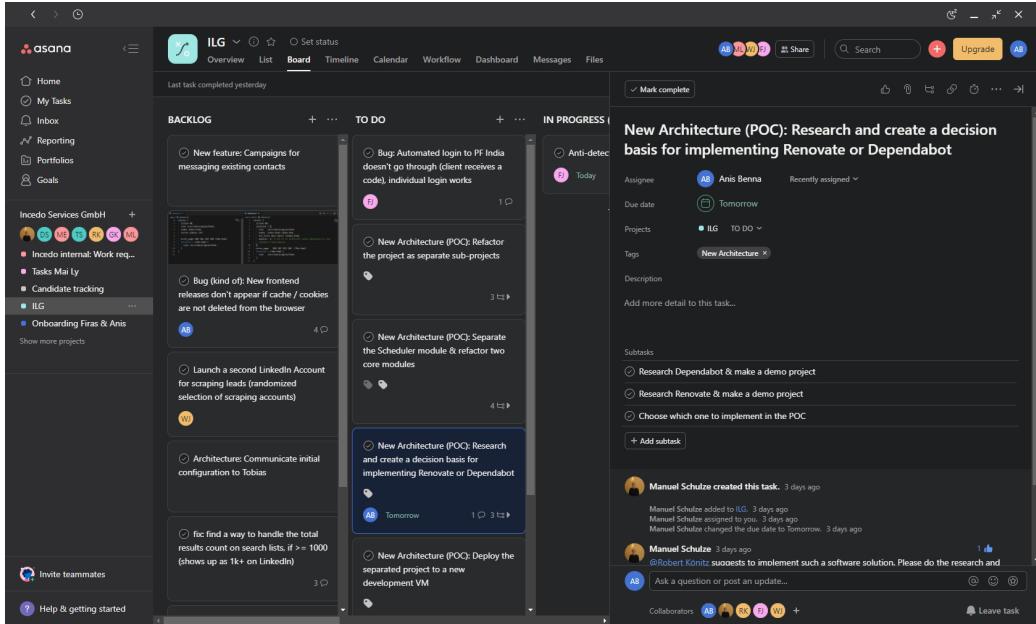


Figure 3: ILG project Kanban board

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.

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

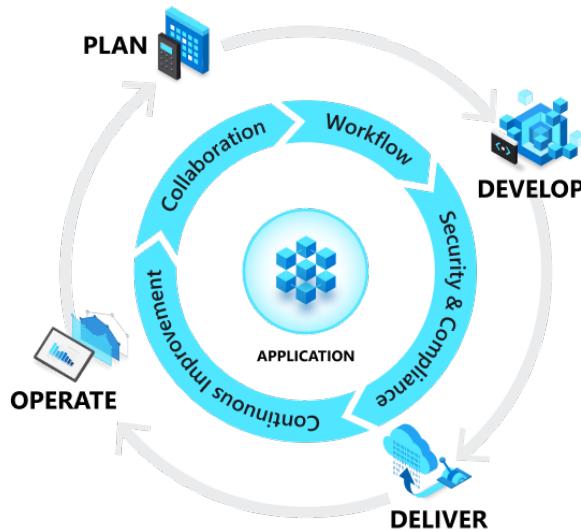


Figure 4: DevOps and the application lifecycle

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. [1]

- **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 to 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 [2]. 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 [4]. 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. [9]

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. [13]

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

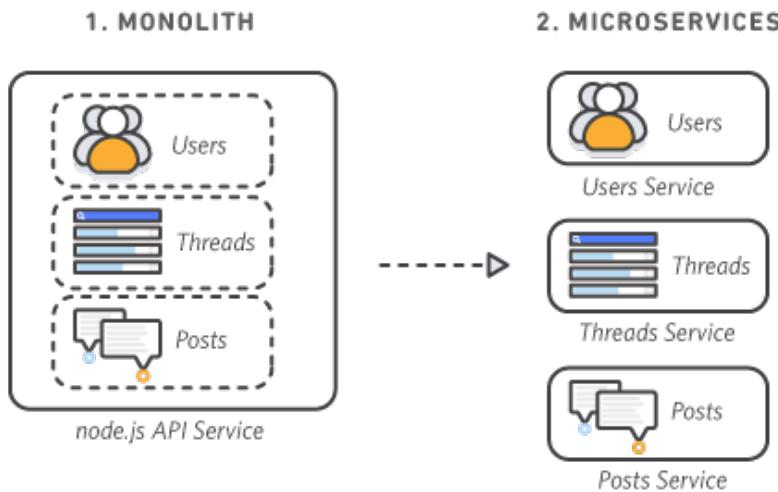


Figure 5: Monolith to microservices example

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. Also it makes refactoring and adding code a lot easier so it prevent the developer from breaking existing feature when adding new one. 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 below :

2.4.2 Types of Tests used

- **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 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 common 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 [5]

	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	<ul style="list-style-type: none"> • Straightforward to install • Takes less time to learn • Works with the Docker CLI 	<ul style="list-style-type: none"> • 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 specification and the requirements of the application. This chapter will describe the product's main features and their conception that should be met for the end result.

3.1 Requirements

Our requirements are divided into two parts:

- The public client portal of the application or dashboard.
- The lead generating happening in the background.

3.1.1 Dashboard

Requirements

Here is a global use case diagram that specifies the dashboard requirements:

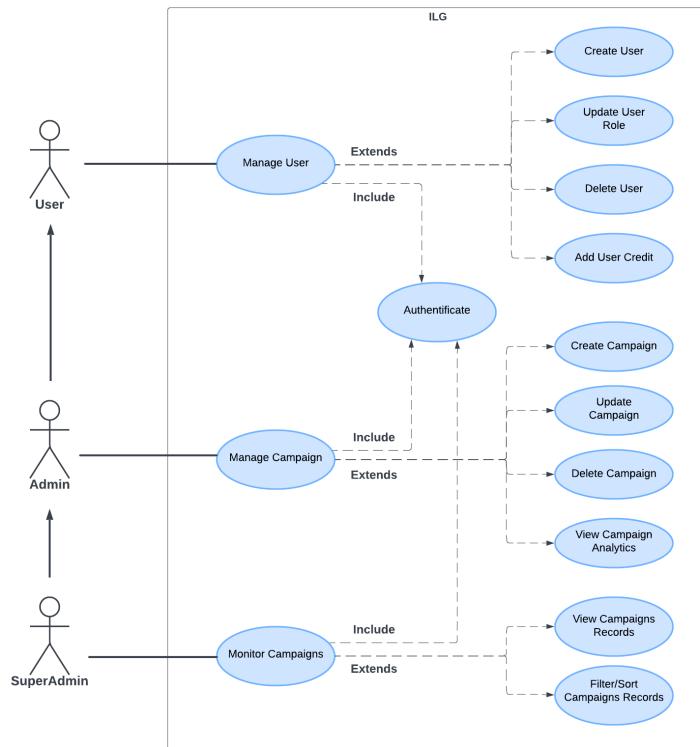


Figure 6: Use case diagram

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	Authentified User.
Post-condition	Campaign Created.
Basic path	<ol style="list-style-type: none"> 1. Navigate to campaign page. 2. Click on the "create campaign" button. 3. Fill in the needed informations 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

Requirements

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

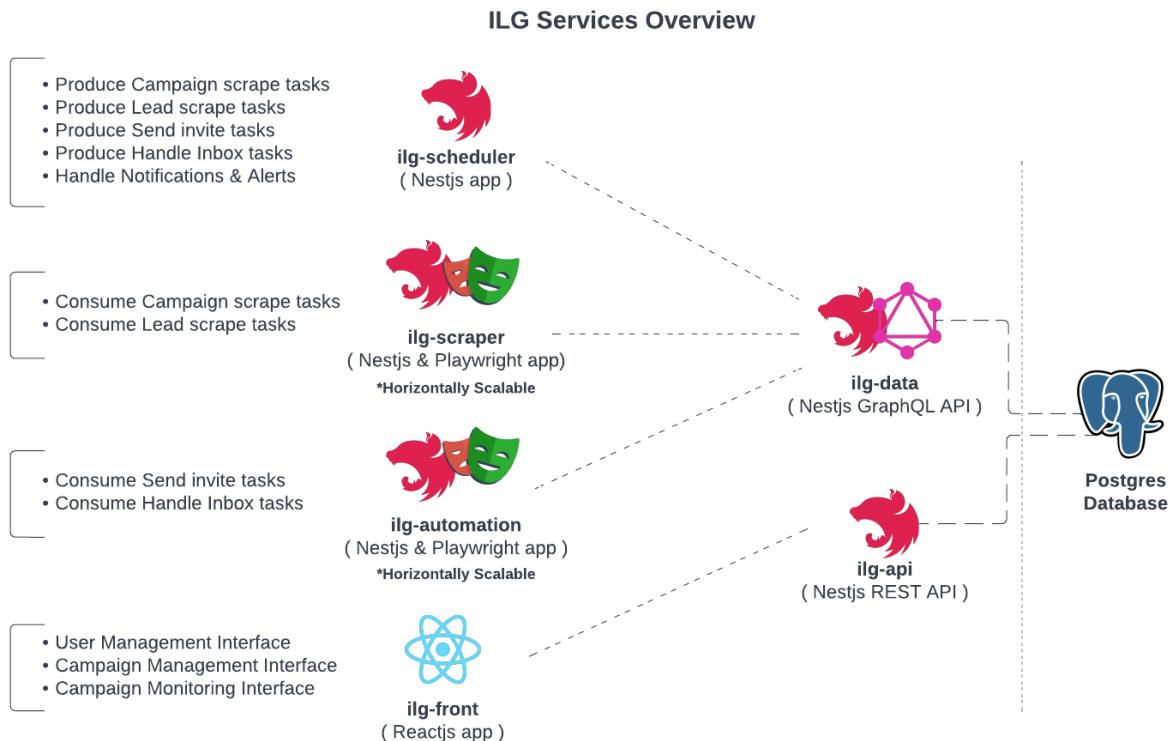


Figure 7: Overview of the ILG services

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-scrapers	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 decided to distribute our different microservices into their corresponding nodes or virtual machines depending on their shared components and scalability.

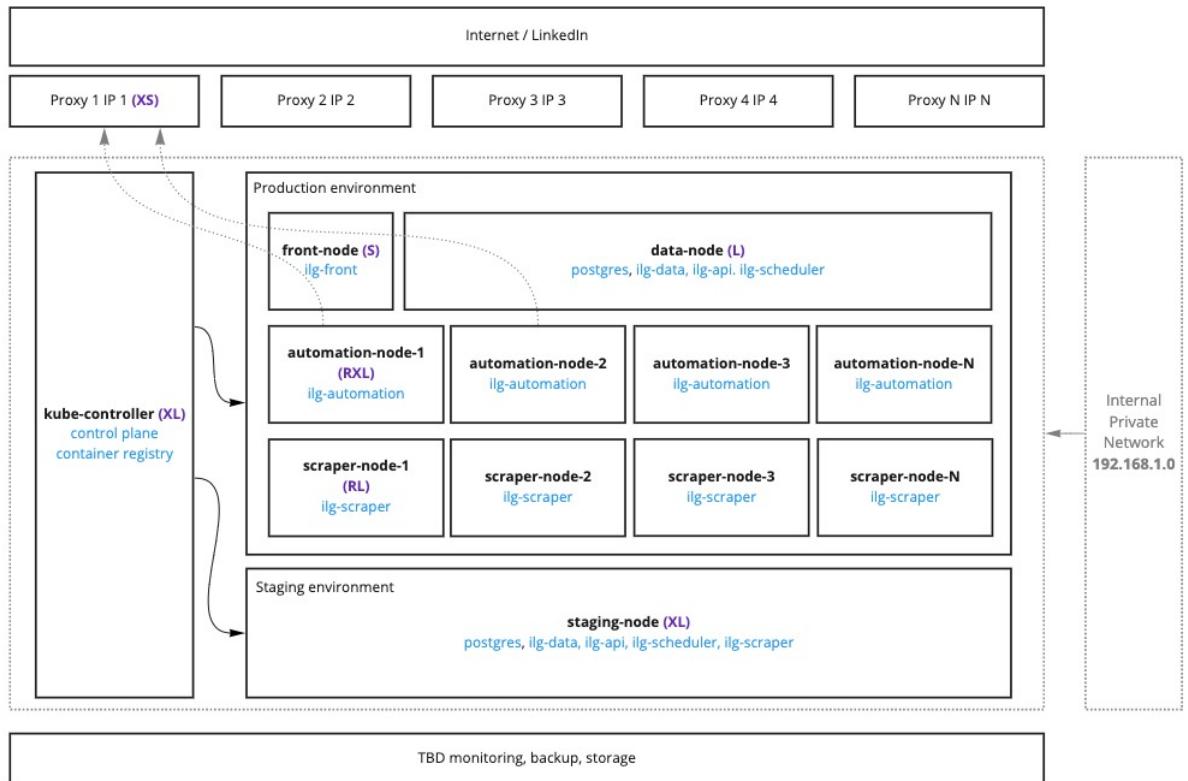


Figure 8: Cloud infrastructure of the application

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.

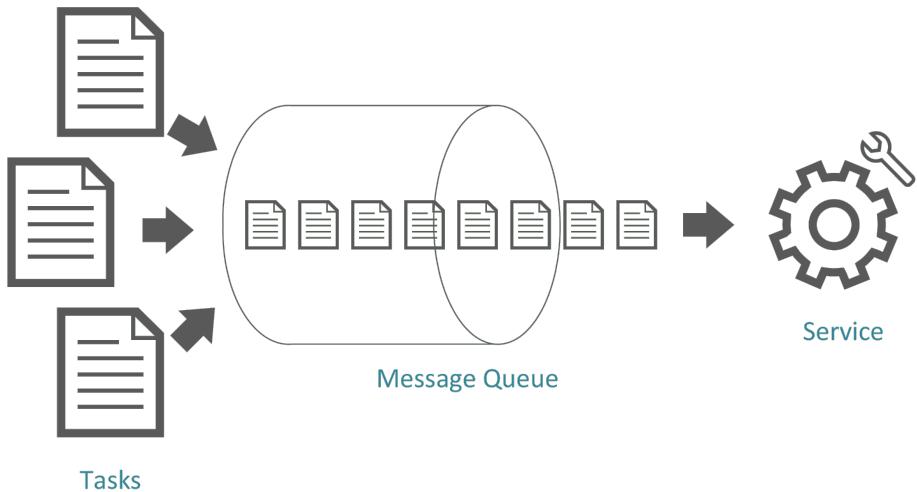


Figure 9: Task queues

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.

Producer/Consumer Diagram

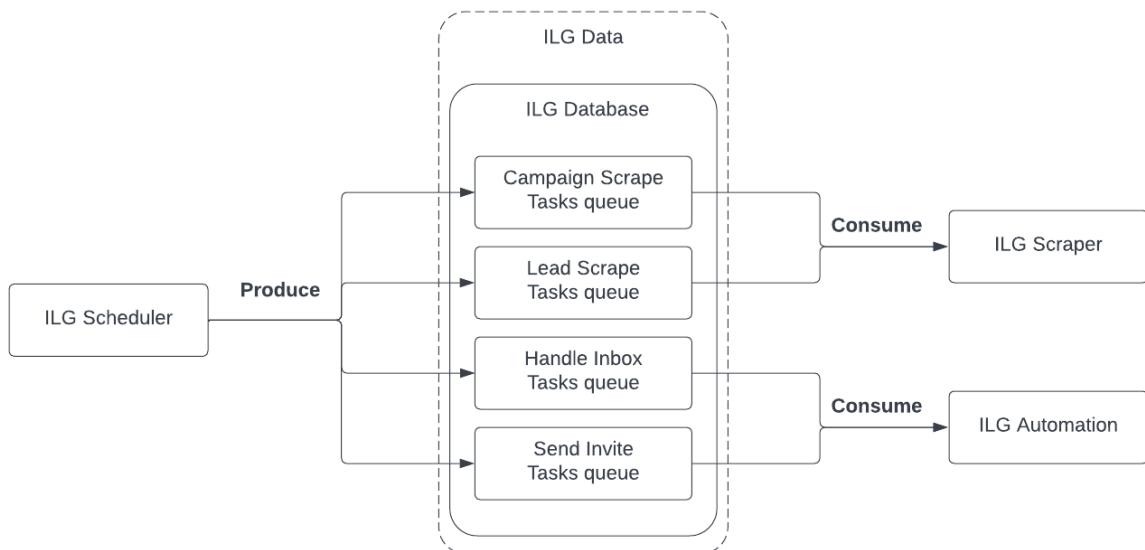


Figure 10: Diagram detailing the producers and the consumers

3.2.3 Scraper & Automation

The `ilg-scraper` and `ilg-automation` services are responsible for consuming the tasks generated on a daily basis by the system. Their main purpose is to generate leads for the various campaigns.

Scraper service flow

Campaign Scraper Sequence diagram

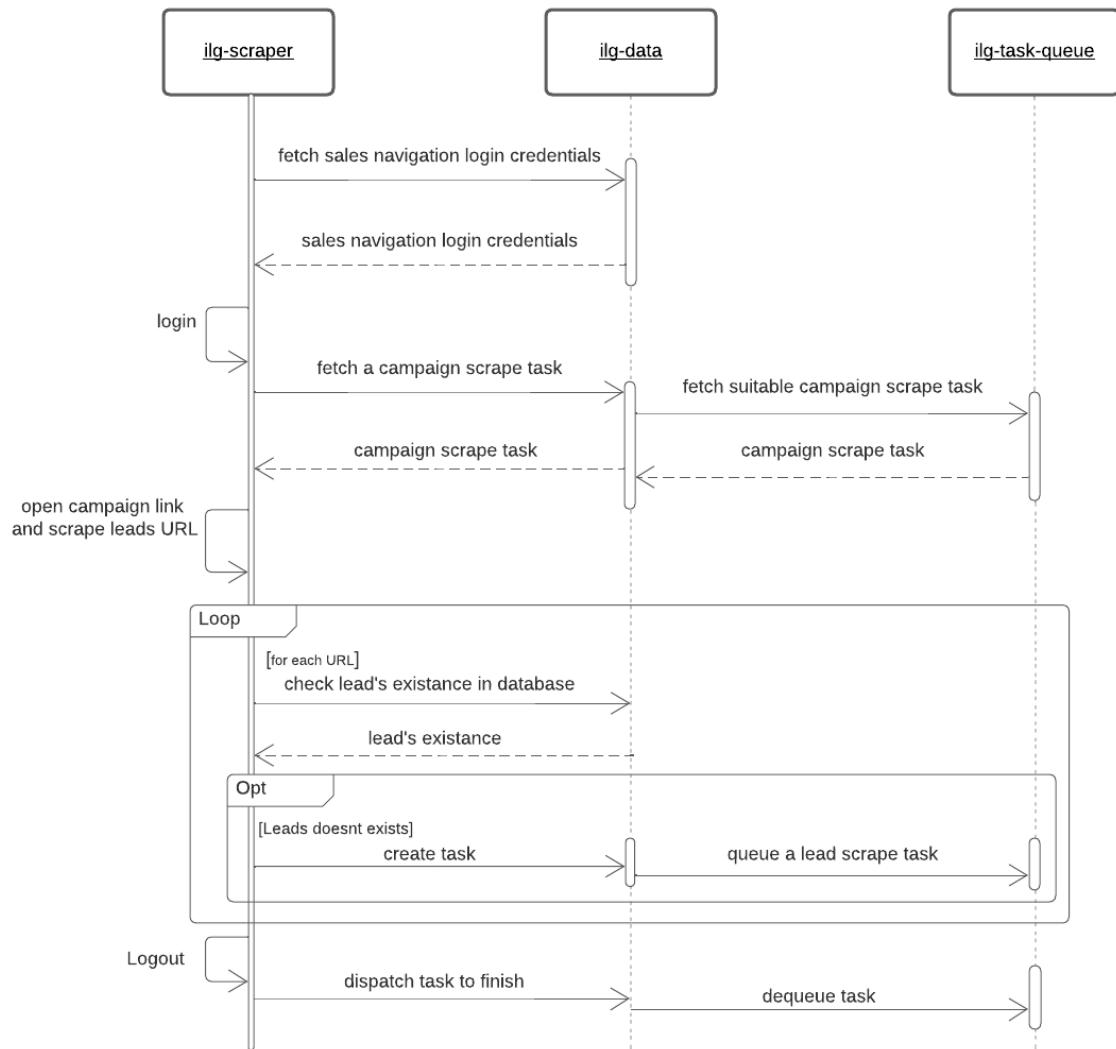


Figure 11: Campaign scraper sequence diagram

Lead Scraper Sequence diagram

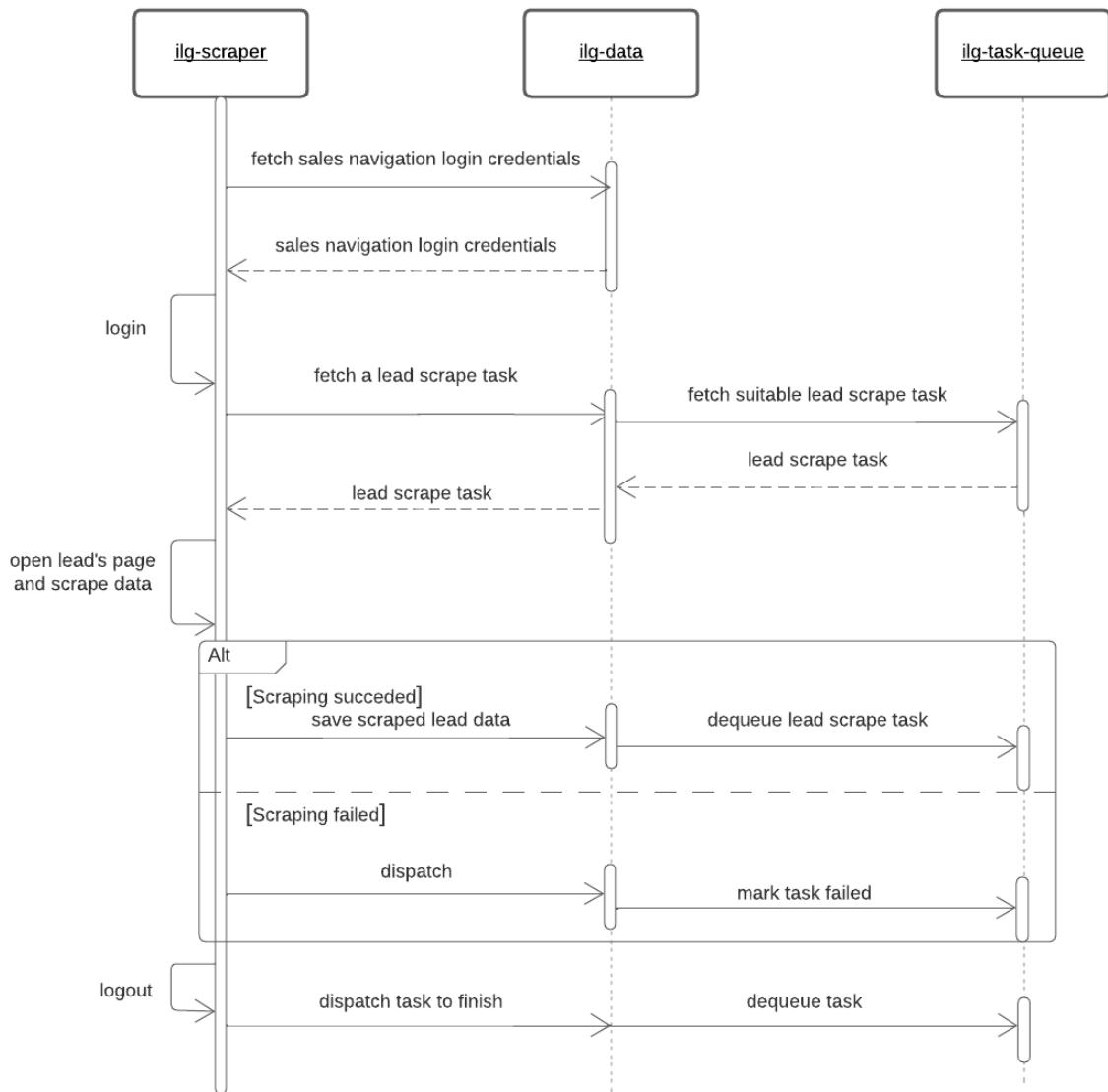


Figure 12: Lead scraper sequence diagram

Automation service flow

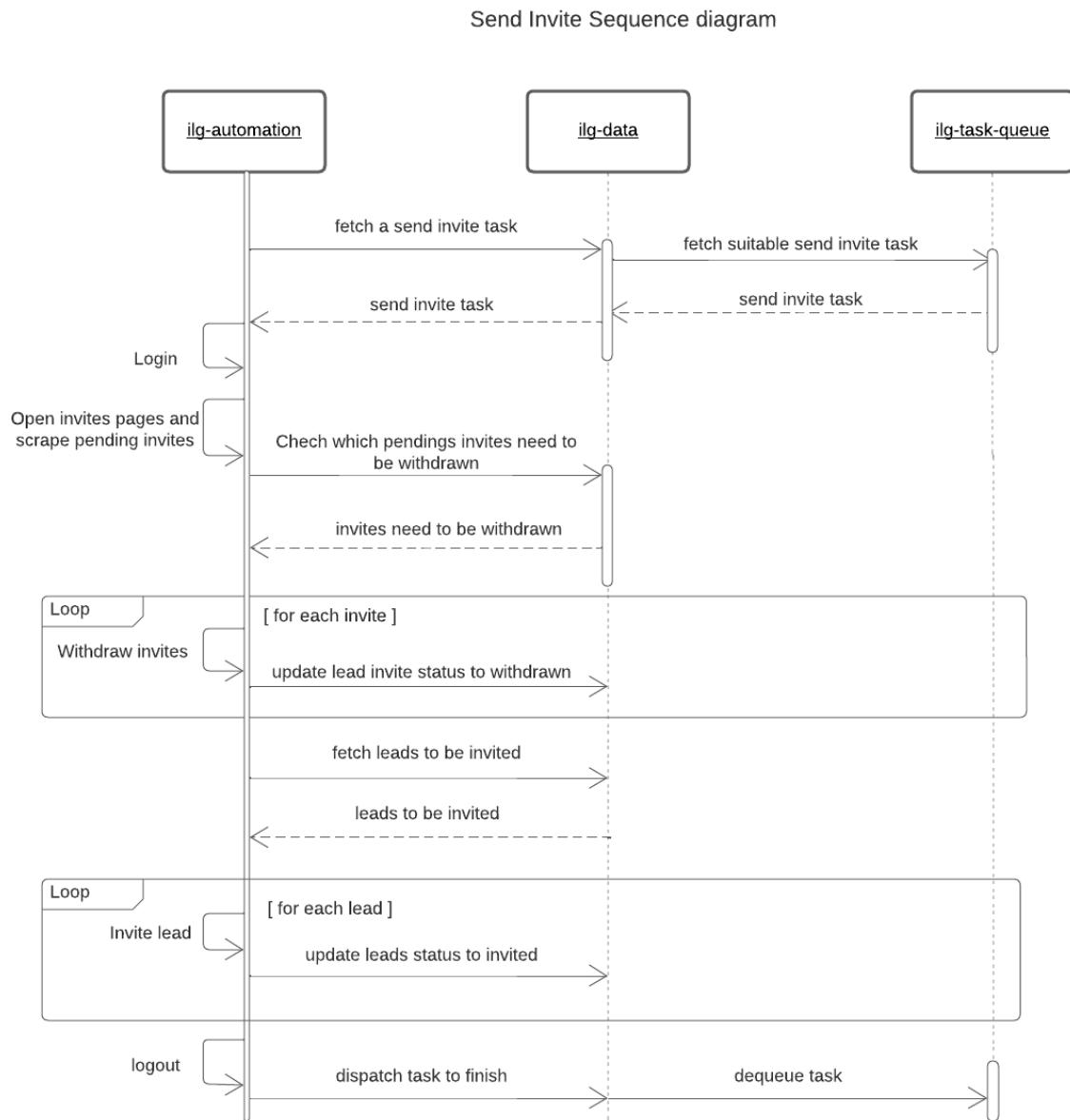


Figure 13: Diagram detailing the producers and the consumers

Send Invite Sequence diagram

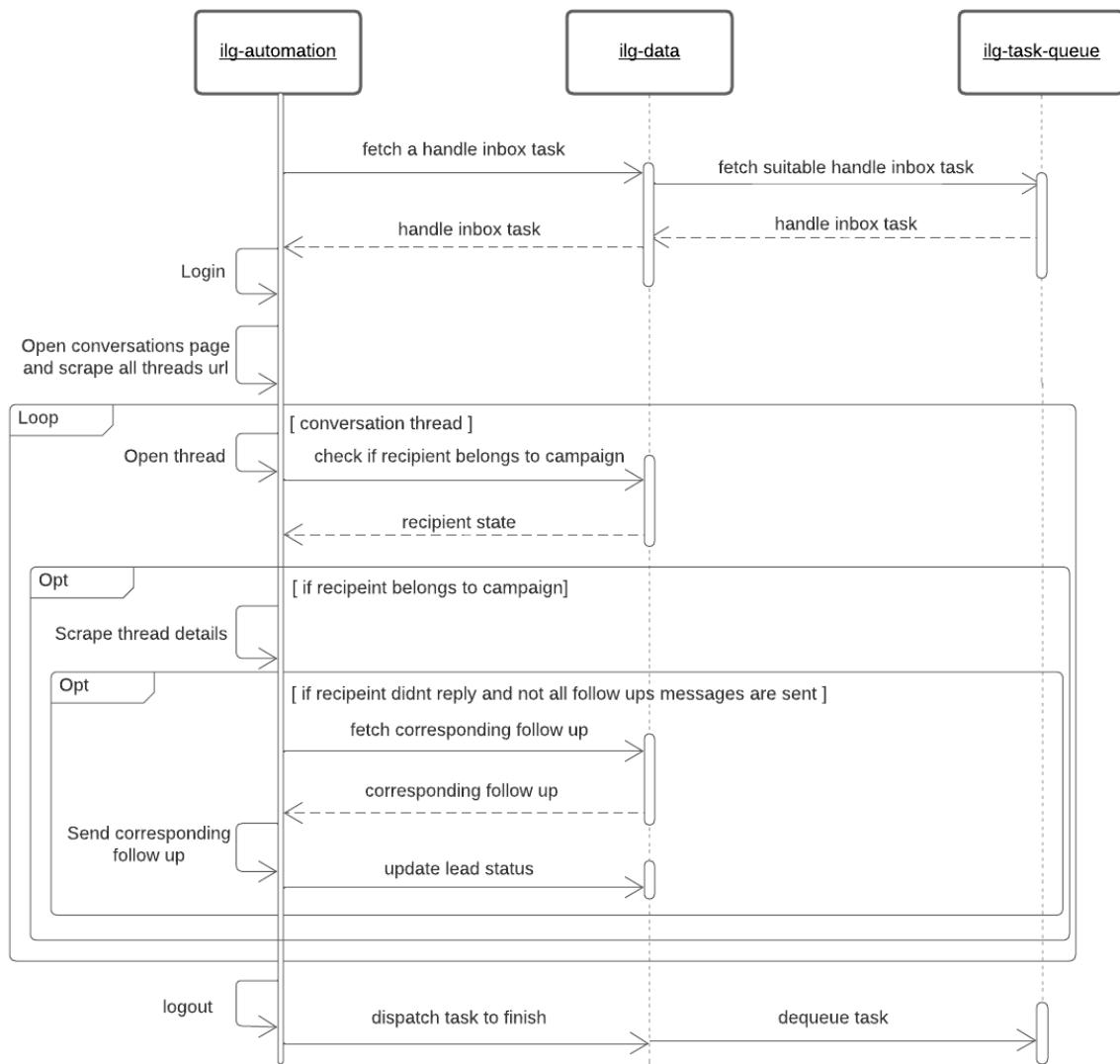


Figure 14: Diagram detailing the producers and the consumers

3.2.4 Data Layers

Schema or Entity Relationship Diagram (ERD)

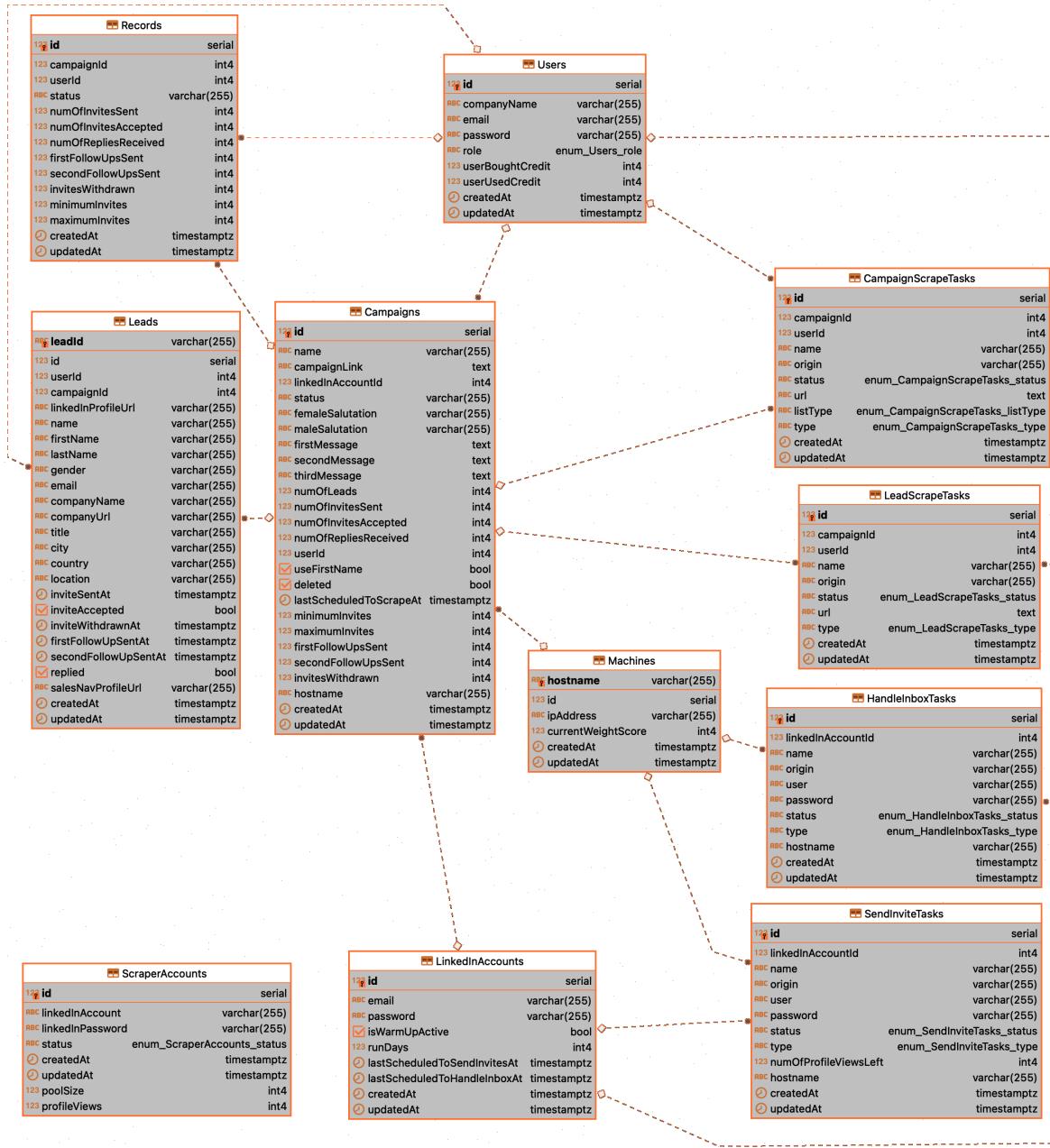


Figure 15: Entity Relationship Diagram

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.

Conclusion

In this chapter, we discussed the main features of the application and its different layers and services. 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

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. We will start with the new architecture hardware and software setup, talk a little bit about monitoring the application. Later on, we will present some of the difficulties we encountered and then list all of the technologies we've used for our work.

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 created 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. [15, Ionos is a web hosting company founded in Germany in 1988 and is currently owned by United Internet.]



Figure 16: 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.

The screenshot shows the Ionos Cloud Panel dashboard. On the left, there is a sidebar with various navigation options: Infrastructure (Servers, Images, Block Storage, Shared Storage), Network (Firewall Policies, Load Balancers, Public IP, Private Network, Public Network, VPN), Security, Backup, Management, and Costs. Below the sidebar, there is a section for Recommended help topics with links to Connecting to Linux Server via SSH, Important Security Information for Your Server, What is the KVM Console?, and Server Configuration with Cloud-Init: General Information.

The main content area has a header with a search bar, a user profile icon, and a 'ABMELDEN' button. It also displays the last login information: 'Last login: 23/05/2022 12:58:30 from 197.27.125.136 (Tunisia)'. A promotional banner for 'Cloud Backup: Switch now and benefit!' is visible, highlighting features like backup any device, encrypt crucial data, full granular scheduling, and multiple restore points. Below the banner, the 'Servers' section is shown with a table listing several servers. The table columns include Name, Status, Backup, IP, Type, OS, Warnings, and Data centre. The listed servers are:

Name	Status	Backup	IP	Type	OS	Warnings	Data centre
automation-node-1	Green	Red	82.***	RXL	Ubuntu 20.04	--	Germany
ctl.ilg.incedo.net / lead-generator.incedo...	Green	Red	212.***	XL	Ubuntu 20.04	--	Germany
data-node	Green	Red	217.***	L	Ubuntu 20.04	--	Germany
front-node	Green	Red	217.***	S	Ubuntu 20.04	--	Germany
scraper-node-1	Green	Red	82.***	RL	Ubuntu 20.04	--	Germany
staging.lead-generator.incedo.net	Green	Red	212.***	L	Ubuntu 20.04	--	Germany
usp1.ilg.incedo.net	Green	Red	212.***	S	Ubuntu 20.04	--	Germany

Figure 17: Ionos cloudpanel dashboard

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 entitled **Kubernetes Cluster**.

Standard	RAM optimized	Flex		
€5.00/month*	€8.00/month*	€16.00/month*	€24.00/month*	€50.00/month*
XS	S	M	L	XL
1 vCore	1 vCore	2 vCore	2 vCore	4 vCore
0.5 GB RAM	1 GB RAM	2 GB RAM	4 GB RAM	8 GB RAM
30 GB SSD	40 GB SSD	60 GB SSD	80 GB SSD	120 GB SSD
€100.00/month*	€160.00/month*	€240.00/month*	€360.00/month*	
XXL	3XL	4XL	5XL	
8 vCore	12 vCore	16 vCore	24 vCore	
16 GB RAM	24 GB RAM	32 GB RAM	48 GB RAM	
160 GB SSD	240 GB SSD	360 GB SSD	480 GB SSD	

Figure 18: Standard VM sizes in IONOS

Standard	RAM optimized	Flex	
€18.00/month*	€40.00/month*	€80.00/month*	€160.00/month*
RM	RL	RXL	RXXL
1 vCore	2 vCore	4 vCore	8 vCore
4 GB RAM	8 GB RAM	16 GB RAM	32 GB RAM
40 GB SSD	80 GB SSD	120 GB SSD	160 GB SSD

Figure 19: 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

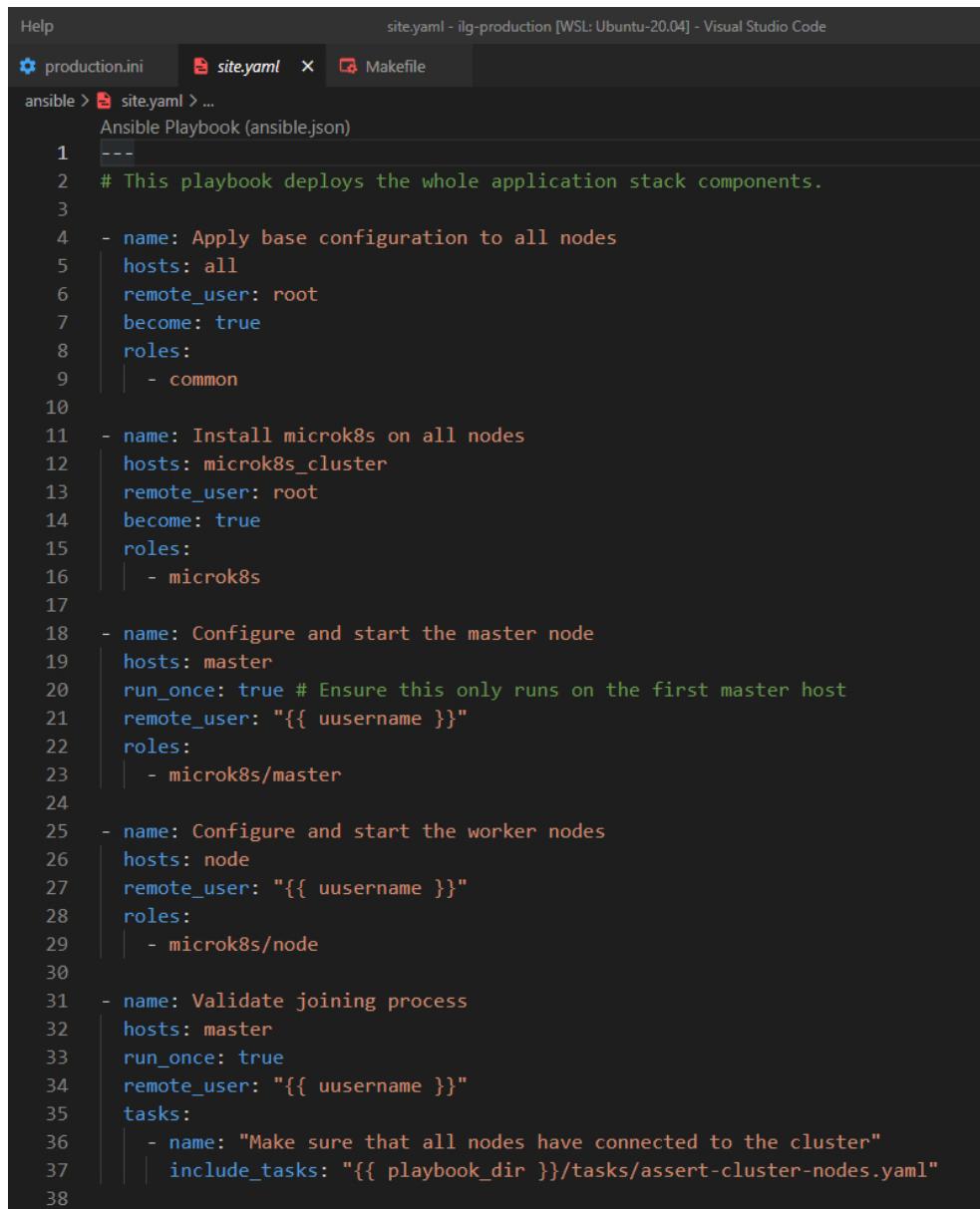
Manually configuring the above can be extremely tedious, inefficient and error prone. Therefore, we've created **Ansible** scripts or playbooks that do just that. In the end, we simply edit a configuration file and directly run the script. This has a huge advantage especially when

- 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.



The screenshot shows a Visual Studio Code interface with a dark theme. The title bar says "site.yaml - ilg-production [WSL: Ubuntu-20.04] - Visual Studio Code". The left sidebar shows "ansible > site.yaml > ...". The main editor area contains the following Ansible playbook code:

```
1  ---
2  # This playbook deploys the whole application stack components.
3
4  - name: Apply base configuration to all nodes
5    hosts: all
6    remote_user: root
7    become: true
8    roles:
9      - common
10
11 - name: Install microk8s on all nodes
12   hosts: microk8s_cluster
13   remote_user: root
14   become: true
15   roles:
16     - microk8s
17
18 - name: Configure and start the master node
19   hosts: master
20   run_once: true # Ensure this only runs on the first master host
21   remote_user: "{{ username }}"
22   roles:
23     - microk8s/master
24
25 - name: Configure and start the worker nodes
26   hosts: node
27   remote_user: "{{ username }}"
28   roles:
29     - microk8s/node
30
31 - name: Validate joining process
32   hosts: master
33   run_once: true
34   remote_user: "{{ username }}"
35   tasks:
36     - name: "Make sure that all nodes have connected to the cluster"
37       include_tasks: "{{ playbook_dir }}/tasks/assert-cluster-nodes.yaml"
38
```

Figure 20: Extract from the Ansible playbook

```

development.ini  production.ini M × Makefile
ansible > inventory > production.ini > ...
1 [master]
2 212.      hv_hostname=kube-controller hv_net_ife="ens224"
3
4 [node]
5 217.      hv_hostname=data-node hv_net_ife="ens224"
6 217.      hv_hostname=front-node hv_net_ife="ens224"
7 82.       hv_hostname=scrapers-node-1 hv_net_ife="ens224"
8 82.       hv_hostname=automation-node-1 hv_net_ife="ens224"
9 212.      hv_hostname=staging-node hv_net_ife="ens224"
10
11 [microk8s_cluster:children]
12 master
13 node
14

```

Figure 21: 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.

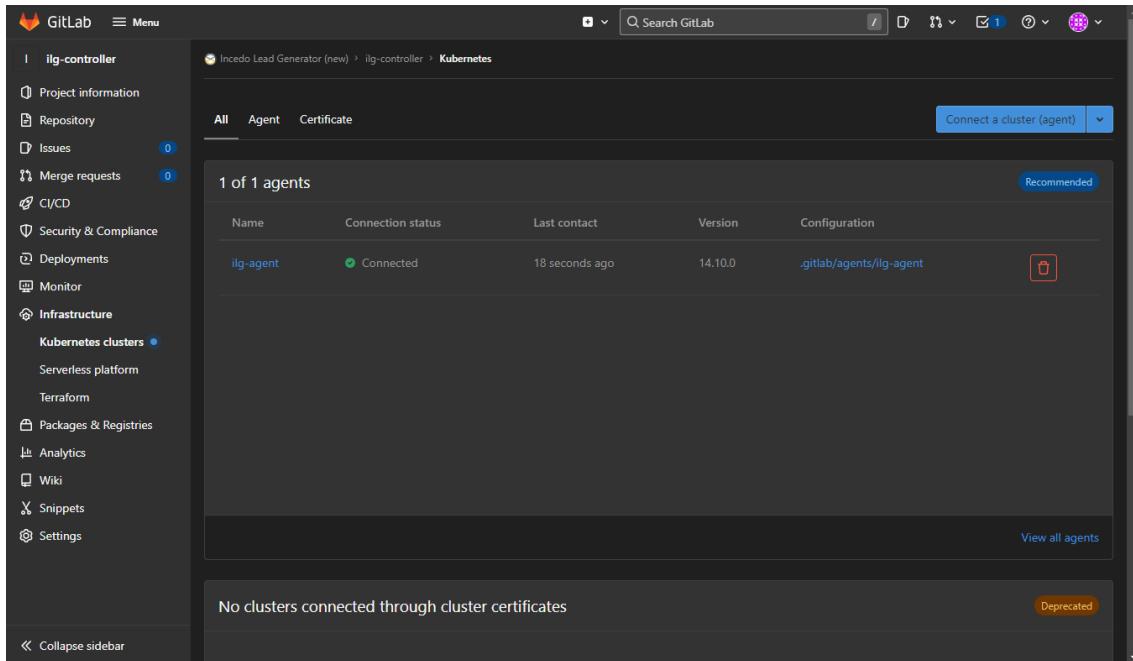
- On code changes on any microservice, we create a new Docker image that we push to a custom container registry.
- Once the build is finished, we trigger a multi-project pipeline to a central repository we will call ilg-controller.
- The central repository will interact with our Kubernetes cluster to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



The screenshot shows the GitLab web interface with a dark theme. On the left is a sidebar with various project and system navigation items. The main content area is titled 'Kubernetes' under the 'ilg-controller' project. It displays a table of agents connected to the cluster. There is one agent listed: 'ilg-agent' which is 'Connected' and last contacted '18 seconds ago'. The table includes columns for Name, Connection status, Last contact, Version, and Configuration. A blue button labeled 'Connect a cluster (agent)' is visible at the top right of the main content area. Below the table, a message states 'No clusters connected through cluster certificates' with a 'Deprecated' label.

Figure 22: 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.

```

20
27 # ----- buildPublish ----- #
28 > .export-image-version: &get-image-version...
29
30
31
32
33
34
35
36
37
38
39 build-publish-image:
40   extends: .docker
41   stage: buildPublish
42   tags:
43     - test
44   rules:
45     - if: '$CI_COMMIT_REF_NAME == "main" || $CI_COMMIT_REF_NAME == "staging"'
46   script:
47     - *get-image-version
48     - make build-and-push
49
50 # ----- deploy ----- #
51 deploy-to-staging:
52   stage: deploy
53   rules:
54     - if: '$CI_COMMIT_REF_NAME == "staging"'
55 > variables: ...
56
57 trigger:
58   project: ilg/ilg-controller
59   branch: main
60   strategy: depend
61   forward:
62     yaml_variables: true
63     pipeline_variables: false
64
65
66
67

```

Figure 23: 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.

```

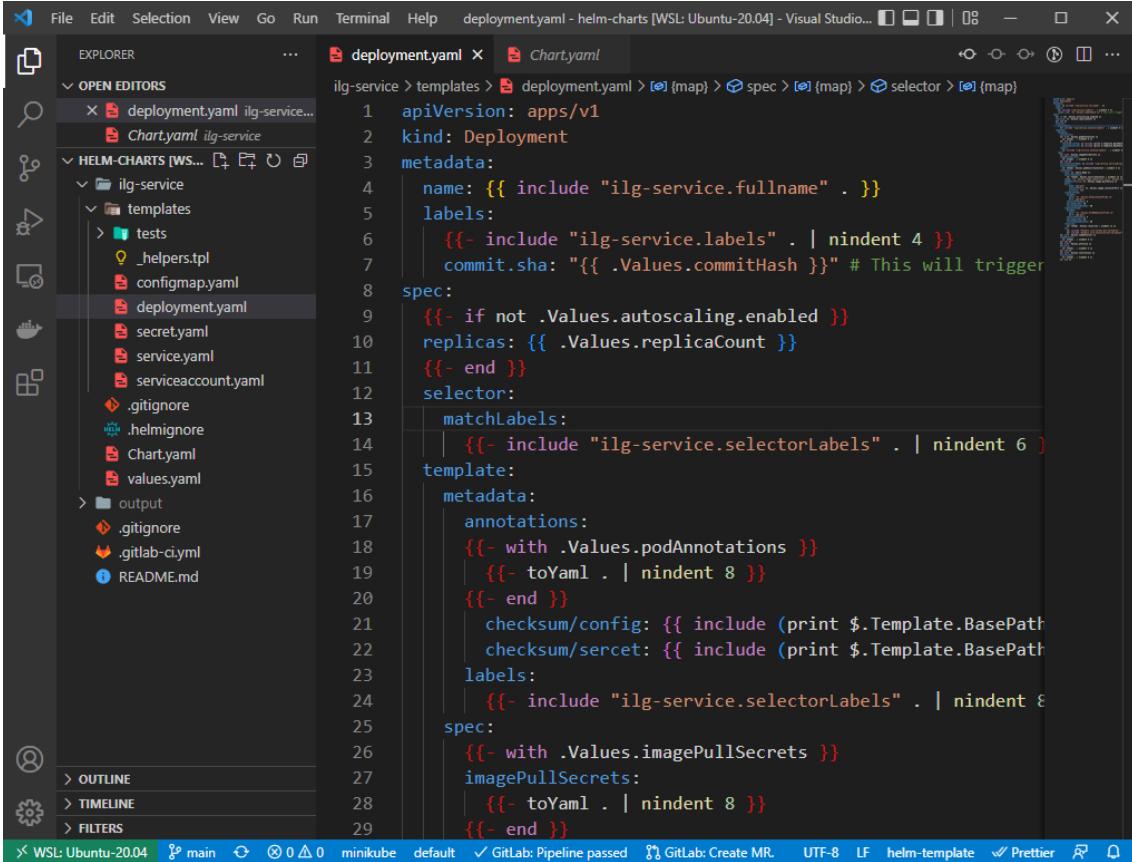
49 |> deploy: check-args set-args ## Deploy the application using helm
50   helm upgrade \
51     --install \
52     --namespace ${ENVIRONMENT} \
53     --create-namespace \
54     --set image.repository='${REMOTE_REGISTRY}/${SERVICE_NAME}' \
55     --set image.tag='${CV_IMAGE_TAG}' \
56     --set image.containerPort='${CONTAINER_PORT}' \
57     --set commitHash='${COMMIT_HASH}' \
58     --set resources.limits.cpu='${RES_LIMIT_CPU}' \
59     --set resources.limits.memory='${RES_LIMIT_MEMORY}' \
60     --set resources.requests.cpu='${RES_REQ_CPU}' \
61     --set resources.requests.memory='${RES_REQ_MEMORY}' \
62     --set nodeSelector."kubernetes\\.io/hostname"='${NODE_NAME}' \
63     --set tolerations[0].key='dedicated',tolerations[0].value='my-pool',tolerat
64     --set appLabels.environment='${ENVIRONMENT}' \
65     ${CV_ARGS} \
66     ${CV_RELEASE_NAME} ilg-helm-charts/ilg-service
67

```

Figure 24: 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.

A screenshot of Visual Studio Code showing the file structure and content of a custom Helm chart. The left sidebar shows a tree view of files: deployment.yaml, Chart.yaml, HELM-CHARTS [WSL: Ubuntu-20.04], ilg-service (containing templates, tests, helpers.tpl, configmap.yaml, deployment.yaml, secret.yaml, service.yaml, serviceaccount.yaml, .gitignore, .helmignore, Chart.yaml, values.yaml), output, .gitignore, .gitlab-ci.yml, and README.md. The main editor pane displays the deployment.yaml file, which defines a Deployment resource for an 'ilg-service'. The file includes sections for apiVersion, kind, metadata, spec, template, and annotations. The spec section contains logic for replicas based on .Values.autoscaling.enabled and includes a selector for the deployment. The template section handles annotations and checksums for the deployment.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "ilg-service.fullname" . }}
  labels:
    {{- include "ilg-service.labels" . | nindent 4 }}
    commit.sha: "{{ .Values.commitHash }}" # This will trigger a new build
spec:
  {{- if not .Values.autoscaling.enabled --}}
  replicas: {{ .Values.replicaCount }}
  {{- end --}}
  selector:
    matchLabels:
      {{- include "ilg-service.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      annotations:
        {{- with .Values.podAnnotations --}}
        {{- toYaml . | nindent 8 --}}
        {{- end --}}
        checksum/config: {{ include (print $.Template.BasePath "configmap.yaml") "YAML" | toYaml | nindent 8 }}
        checksum/secret: {{ include (print $.Template.BasePath "secret.yaml") "YAML" | toYaml | nindent 8 }}
      labels:
        {{- include "ilg-service.selectorLabels" . | nindent 8 }}
  spec:
    {{- with .Values.imagePullSecrets --}}
    imagePullSecrets:
      {{- toYaml . | nindent 8 --}}
    {{- end --}}
```

Figure 25: 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.

```

# ----- buildPackage -----
build-ilg-service:
  extends: .helm
  stage: buildPackage
  tags:
    - test
  rules:
    - 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 26: Extract from the custom Helm package's pipeline

The screenshot shows the GitLab interface for the 'Inedo Lead Generator (new)' project under the 'helm-charts' namespace. The 'Package Registry' section displays a list of packages. There are 6 packages listed, all named 'ilg-service'. Each entry includes the version number, the author ('Anis Benna'), the Helm icon, the commit hash (e.g., 635bdd9b, 0c57334c, etc.), the date it was created (2 days ago), and a trash can icon for deletion.

Version	Published by	Commit Hash	Created	Action
0.1.5	Anis Benna	635bdd9b	Created 2 days ago	Delete
0.1.4	Anis Benna	0c57334c	Created 2 days ago	Delete
0.1.3	Anis Benna	1337345b	Created 2 days ago	Delete
0.1.2	Anis Benna	408ebbe4	Created 2 days ago	Delete
0.1.1	Anis Benna	39e93a08	Created 2 days ago	Delete
0.1.0	Anis Benna	ee24d38	Created 2 days ago	Delete

Figure 27: GitLab's package registry

4.3.6 Our deployed application

With the previous work here done, we can now view our deployed application at <https://lead-generator.incedo.net> and here is how everything looks like

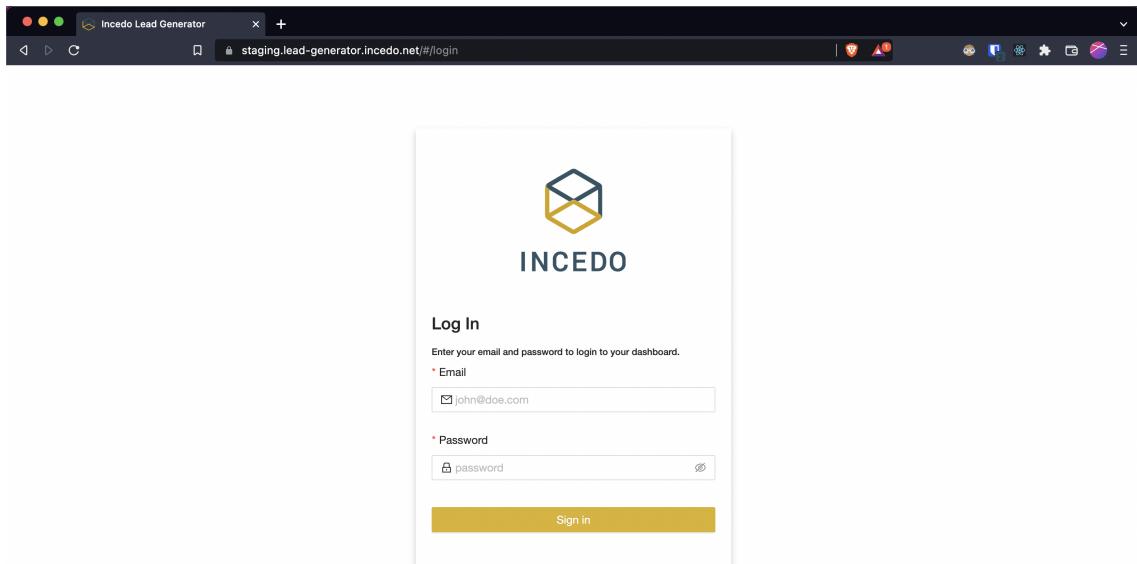


Figure 28: Login page

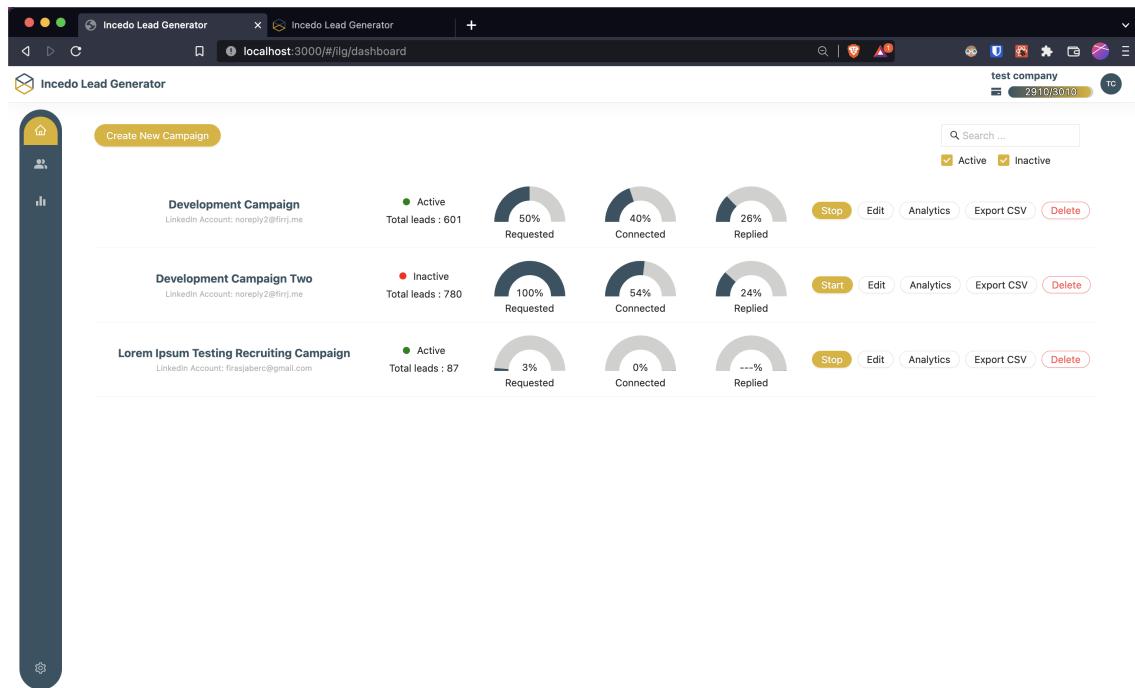


Figure 29: Campaigns Overview page

Figure 30: Add Campaign form

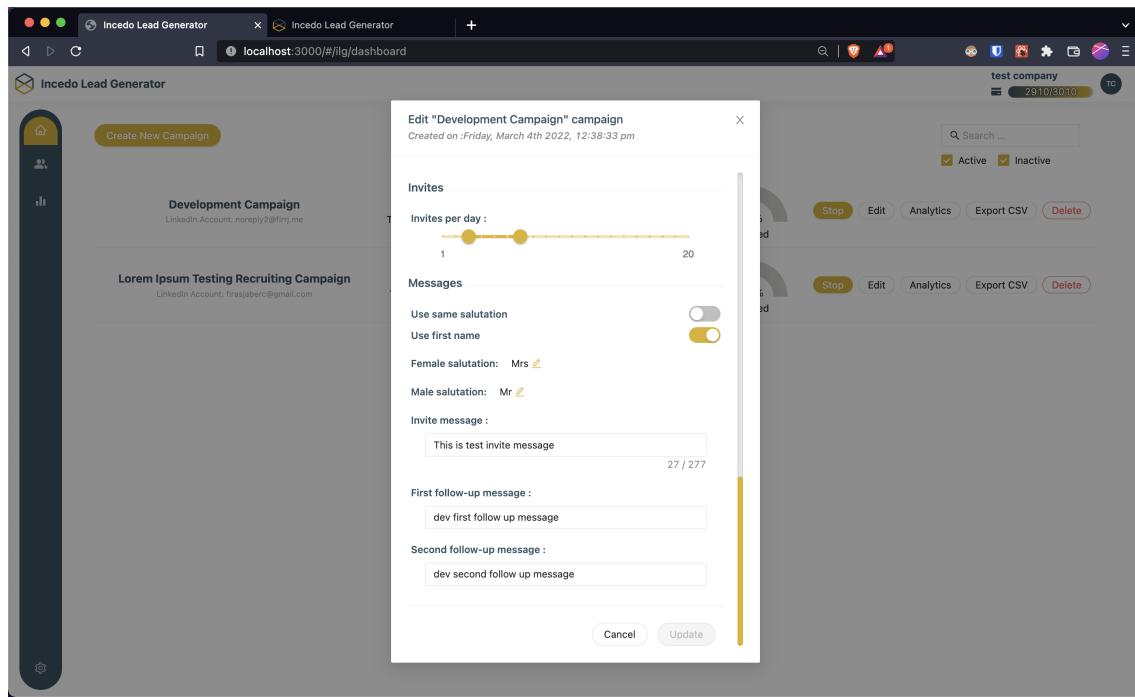


Figure 31: Edit Campaign form

Incedo Lead Generator						
Incedo Lead Generator						
Incedo Lead Generator						
Company name	E-mail	Role	Purchased credits	Used credits	Credits left	Action
Incedo Services GmbH	contact@admin	SuperAdmin	3010	100	2910	<button>Add credit</button> <button>Edit</button>
Lorem Client	email@lorem.com	User	500	0	500	<button>Add credit</button> <button>Edit</button>
Ipsum GmbH	email@ipsum.com	User	200	0	200	<button>Add credit</button> <button>Edit</button>
HR One	contact@firrj.me	Admin	0	0	0	<button>Add credit</button> <button>Edit</button>

Figure 32: View Users page

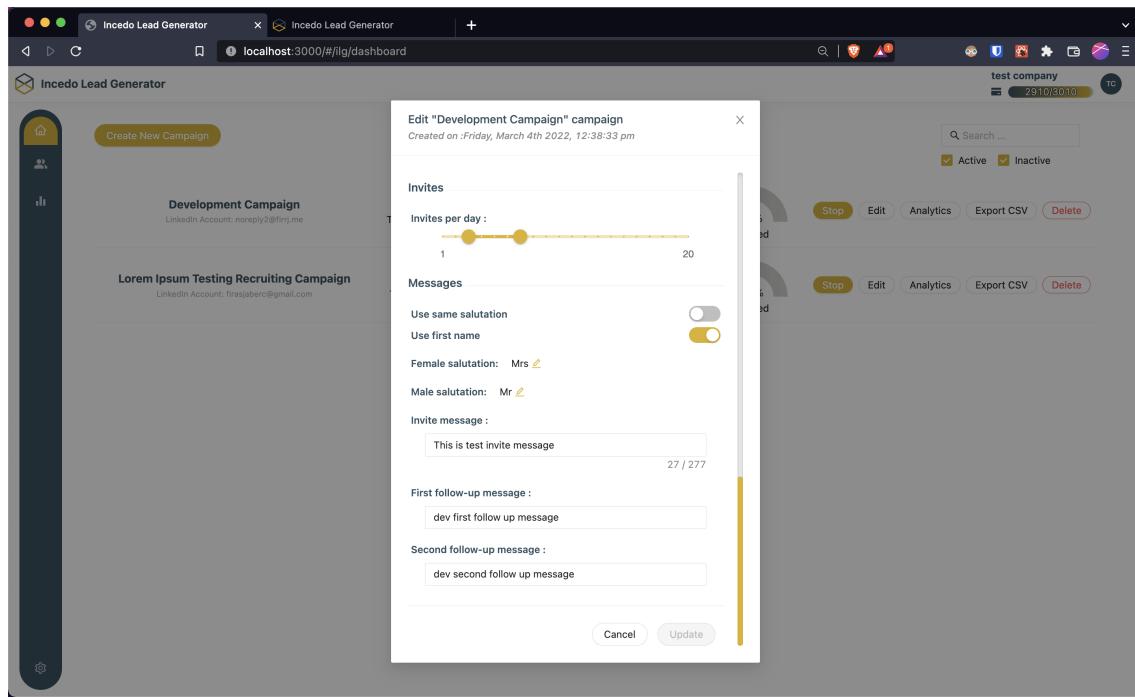


Figure 33: Edit Campaign form

Incedo Lead Generator						
localhost:3000#/ilg/users						
Incedo Lead Generator						
+ Add New User						
Company name	E-mail	Role	Purchased credits	Used credits	Credits left	Action
Incedo Services GmbH	contact@admin	SuperAdmin	3010	100	2910	Add credit Edit
Lorem Client	email@lorem.com	User	500	0	500	Add credit Edit
Ipsum GmbH	email@ipsum.com	User	200	0	200	Add credit Edit
HR One	contact@firrj.me	Admin	0	0	0	Add credit Edit

Figure 34: View Users page

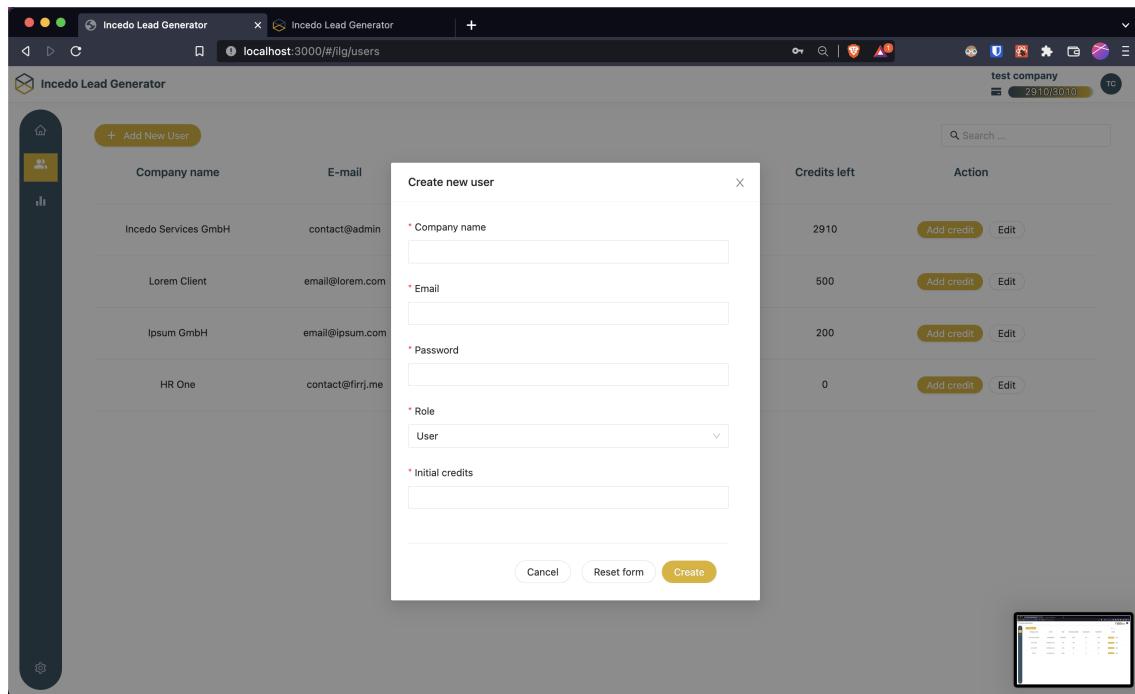


Figure 35: Add User form

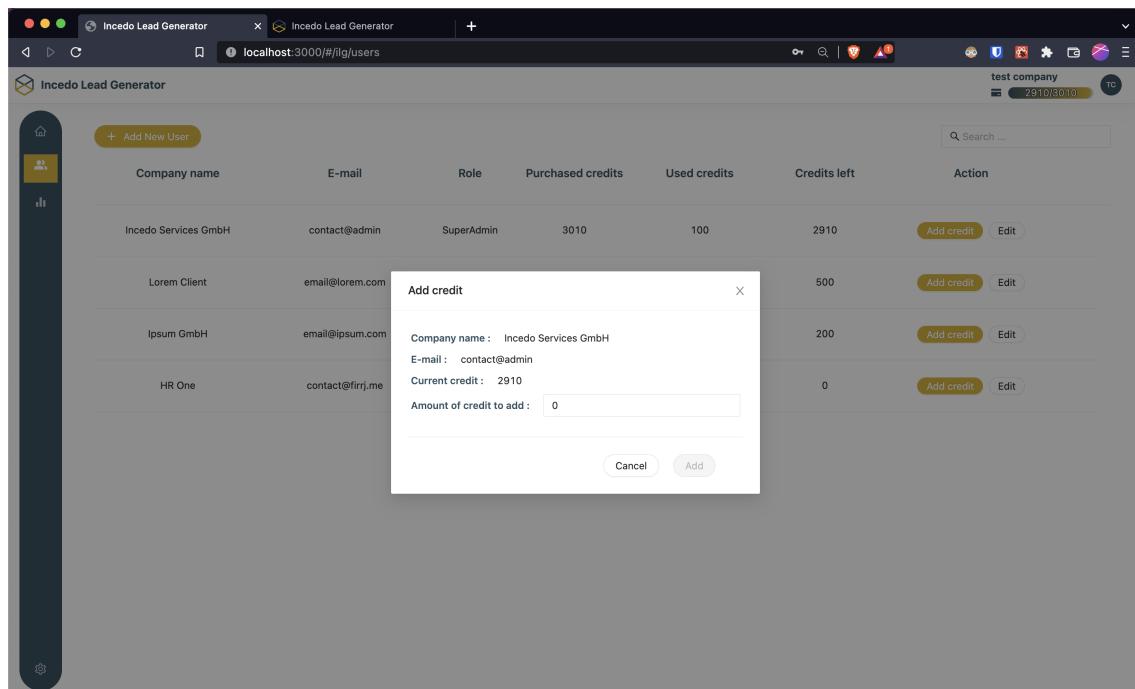


Figure 36: Users Add credit form

Account's e-mail	Campaign name	Connection invites Sent	First follow-up messages sent	Second follow-up messages sent	Invites withdrawn
contact@admin	Development Campaign Lorem Ipsum Testing Recruiting Campaign	349 0	64 0	68 0	174 0
contact@firrj.me	Development Campaign Two	237	58	49	151

Figure 37: Campaigns Records page

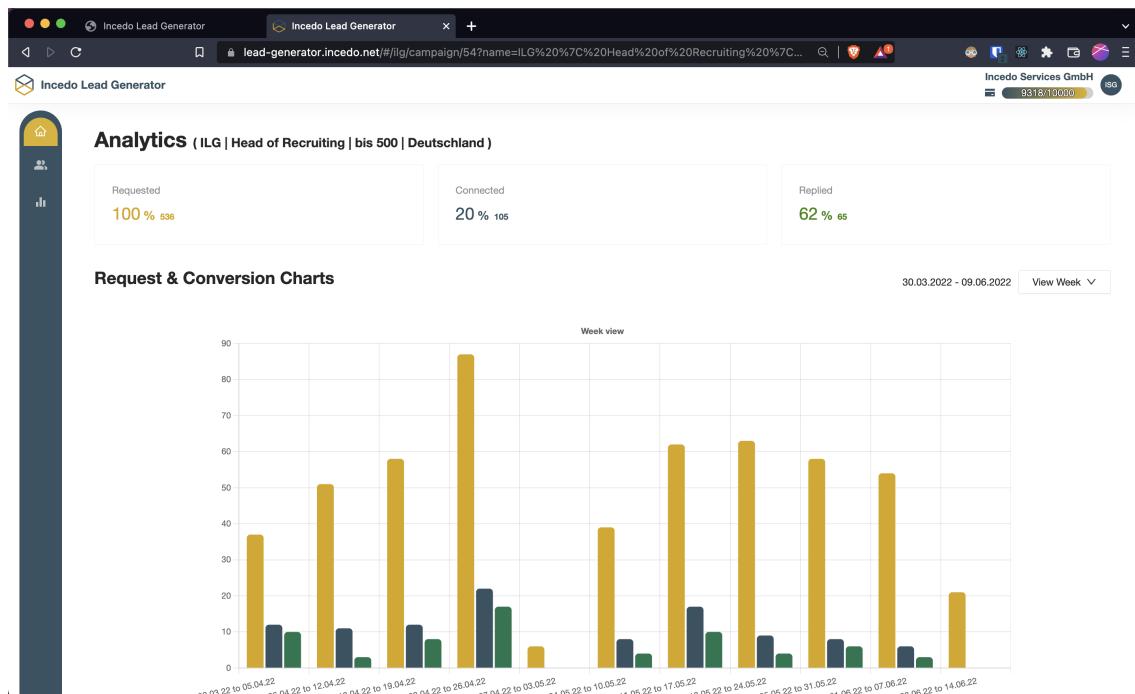


Figure 38: Campaigns Analytics page

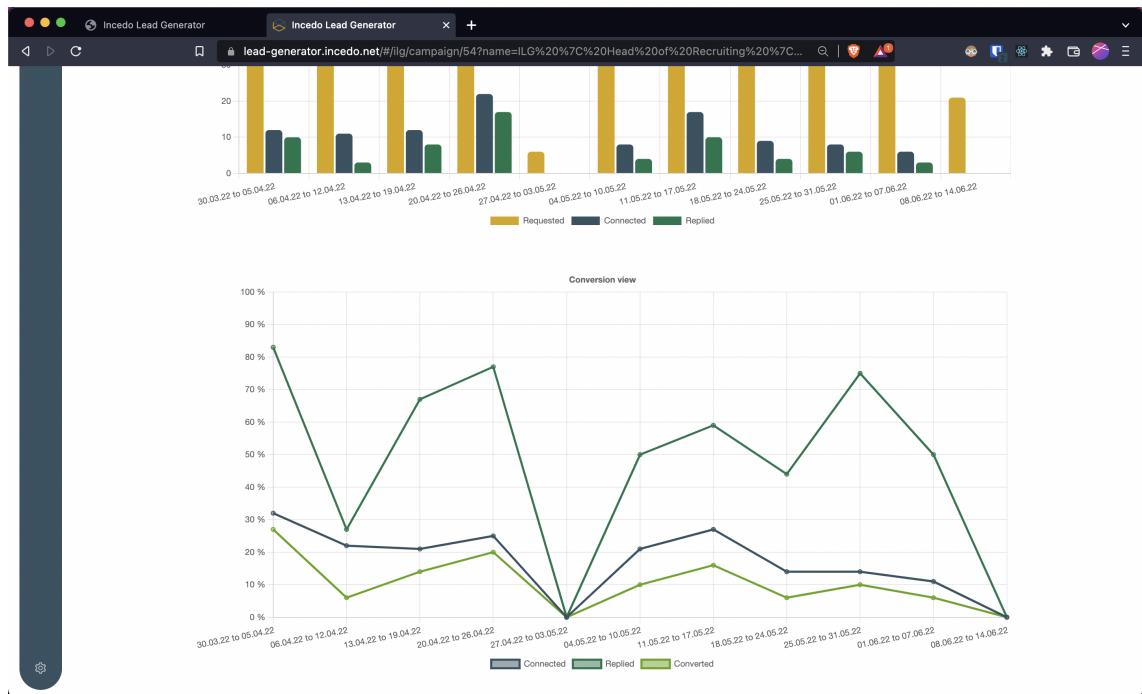


Figure 39: Campaigns Analytics page

4.4 Monitoring

In order to make the most out of our Kubernetes setup, we also setup monitoring for our whole application stack. The things we monitor vary from the cluster nodes' CPU and memory usage, to the logs of our microservices.

In order to get started, we setup the necessary manifest resources to expose the dashboard provided by Kubernetes.

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 as well as configuring the automatic sending of alerts in case of critical errors.

The screenshot shows the Prometheus UI Targets page with four serviceMonitor entries:

- serviceMonitor/monitoring/kube-prometheus-stack-alertmanager/0 (1/1 up)**
Endpoint: <http://10.1.68.131:9093/metrics>, State: UP, Labels: container="alertmanager", endpoint="http-web", instance="10.1.68.131:9093", job="kube-prometheus-stack-alertmanager", namespace="monitoring", pod="alertmanager_kube_prometheus-stack_alertmanager_0", service="kube-prometheus-stack-alertmanager". Last Scrape: 4m 59s ago, Scrape Duration: 4.673ms.
- serviceMonitor/monitoring/kube-prometheus-stack-apiserver/0 (1/1 up)**
Endpoint: <https://192.168.1.1:16443/metrics>, State: UP, Labels: endpoint="https", instance="192.168.1.1:16443", job="apiserver", namespace="default", service="kubernetes". Last Scrape: 4m 36s ago, Scrape Duration: 247.736ms.
- serviceMonitor/monitoring/kube-prometheus-stack-coredns/0 (1/1 up)**
Endpoint: <http://10.1.68.151:9153/metrics>, State: UP, Labels: container="coredns", endpoint="http-metrics", instance="10.1.68.151:9153", job="coredns", namespace="kube-system", pod="coredns-6c6478b6c-fd79f", service="kube-prometheus-stack-coredns". Last Scrape: 4m 54s ago, Scrape Duration: 3.064ms.
- serviceMonitor/monitoring/kube-prometheus-stack-kube-state-metrics/0 (1/1 up)**
Endpoint: [\(link\)](#), State: UP, Labels: (not visible). Last Scrape: (not visible), Scrape Duration: (not visible).

Figure 40: 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.

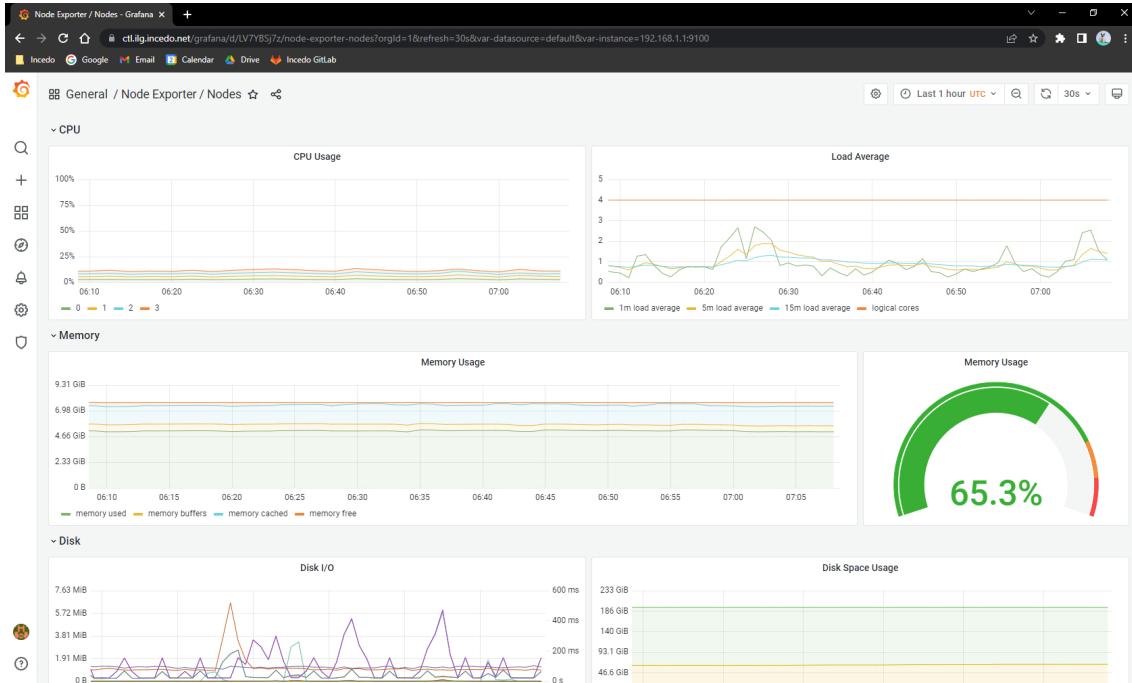


Figure 41: Monitoring the cluster nodes with Grafana

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 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 always almost used 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.

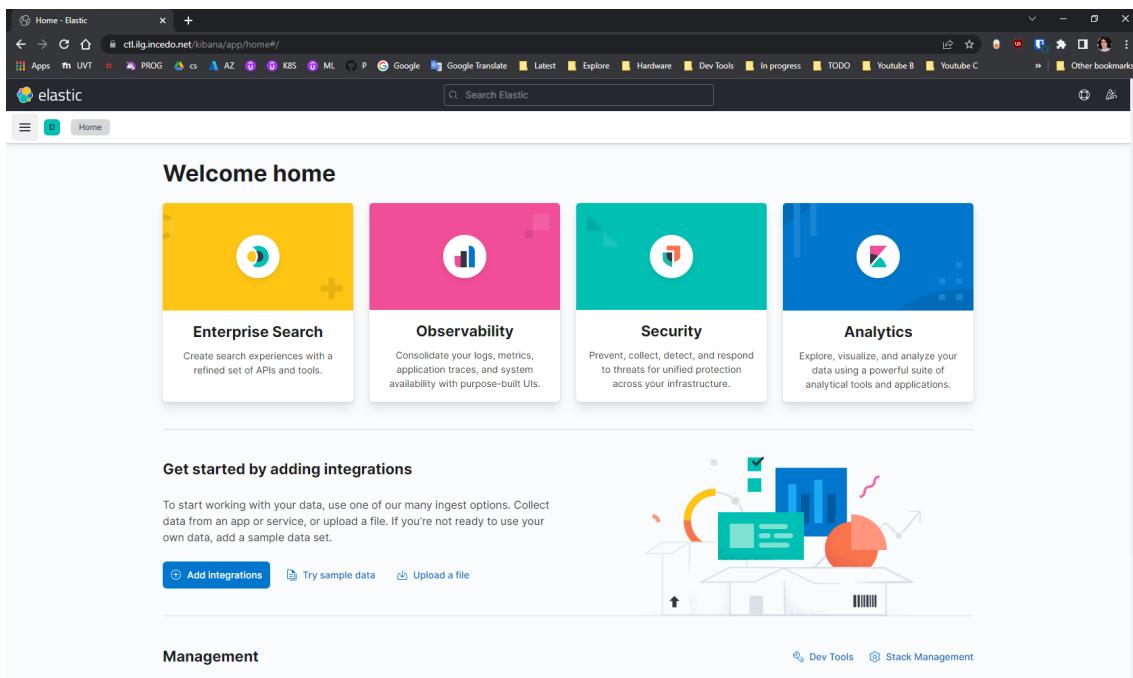


Figure 42: Kibana dashboard home screen

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 us 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.

```

# Run the DaemonSet's Pod on the tainted cluster nodes
# And implicitly on all other nodes that are not tainted
tolerations:
- key: dedicated
  value: my-pool
  effect: NoSchedule
containers:
- name: fluentd
  image: fluent/fluentd-kubernetes-daemonset:v1.9.2-debian-elasticsearch7-1.0
  env:
    - name: FLUENT_ELASTICSEARCH_HOST
      value: "elasticsearch-master.efk-stack"
    - name: FLUENT_ELASTICSEARCH_PORT
      value: "9200"
    - name: FLUENT_ELASTICSEARCH_SCHEME
      value: "http"
    - name: FLUENT_UID
      value: "0"
    - name: FLUENT_ELASTICSEARCH_LOGSTASH_PREFIX
      value: "ilg-kubernetes-cluster"
    - name: FLUENT_ELASTICSEARCH_LOGSTASH_INDEX_NAME
      value: "ilg-kubernetes-cluster"
  resources:
    limits:
      memory: 200Mi
  requests:

```

```

6   data:
7     fluent.conf: |
8       <match fluent.>
9         @type null
10        </match>
11
12       <match kubernetes.var.log.containers.**fluentd**.log>
13         @type null
14        </match>
15
16       <match kubernetes.var.log.containers.**kube-system**.log>
17         @type null
18        </match>
19
20       <match kubernetes.var.log.containers.**kibana**.log>
21         @type null
22        </match>
23
24       <source>
25         @type tail
26         path /var/log/containers/*.log
27         pos_file fluentd-docker.pos
28         time_format %Y-%m-%dT%H:%M:%S
29         tag kubernetes.*
30
31       <parse>
32         @type multi_format

```

Figure 43: 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.

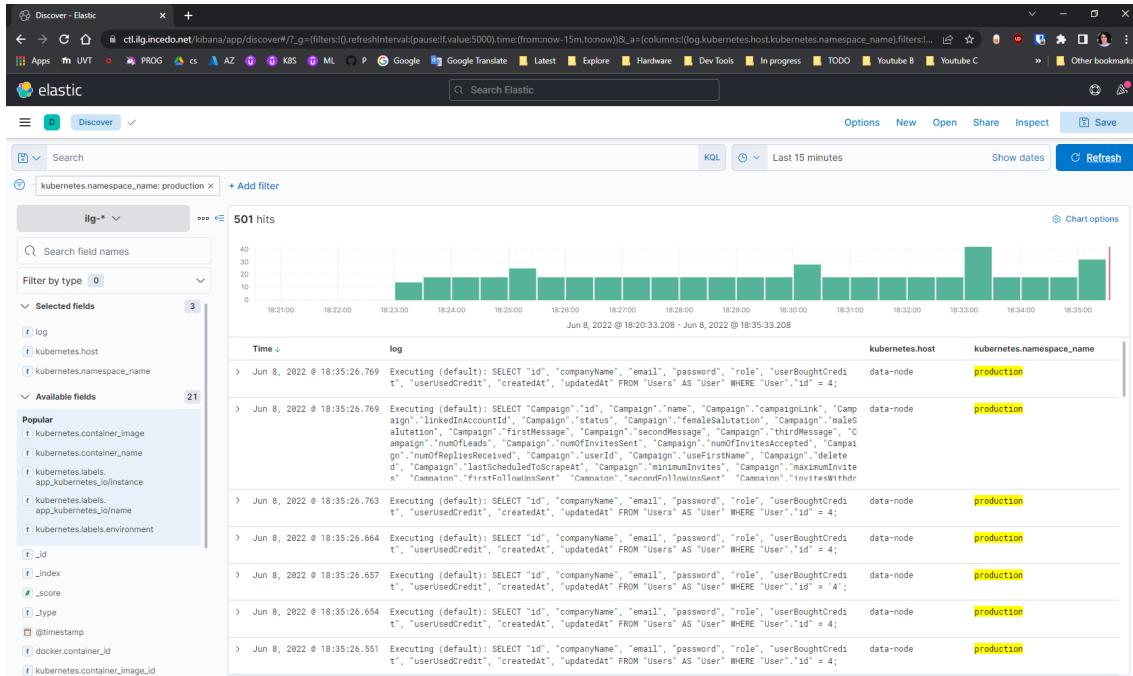


Figure 44: Indexing the logs in Kibana

4.5 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 45: Logo of Git

- **Docker:**

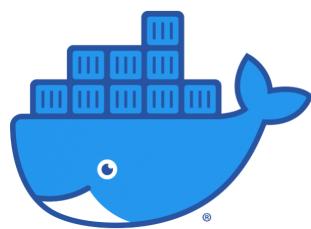


Figure 46: Logo of Docker

- **Playwright:**



Figure 47: Logo of Playwright

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



Figure 48: Logo of NestJS

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

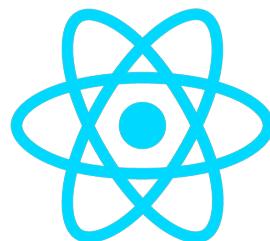


Figure 49: Logo of ReactJS

- **PostgreSQL:**



Figure 50: Logo of PostgreSQL

- **Sequelize:**

[12] A modern TypeScript and Node.js ORM for SQL databases.



Sequelize

Figure 51: Logo of Sequelize

- **GraphQL:**

[7] A query language for APIs and a runtime for fulfilling those queries with existing data.

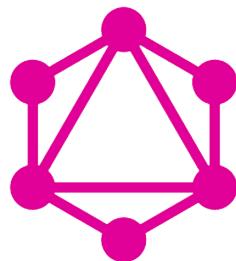


Figure 52: Logo of GraphQL

- **GitLab:**

A DevOps software that allows secure collaboration and operations in a single application.



Figure 53: Logo of GitLab

- **Nginx:**

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



Figure 54: Logo of Nginx

- **Renovate:**

A software that provides automatic dependency updates with support for multiple languages.



Figure 55: Logo of Renovate

- **Docker compose:**

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

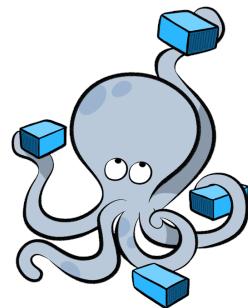


Figure 56: Logo of Docker Compose

- **Visual Studio Code:**

A code editor that's used as an entire development environment with the help of extensions.



Figure 57: Logo of VSCode

- **Linux:**

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



Figure 58: Logo of Linux

- **Make:**

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



Figure 59: Logo of Makefile

- **MicroK8s:**

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



Figure 60: Logo of MicroK8s

- **Kubernetes:**

The most powerful tool for managing containerized workloads in the cloud.



Figure 61: Logo of Kubernetes

- **Ansible:**

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



Figure 62: Logo of Ansible

- **Squid Proxy:**

[3] Squid is a caching proxy for the Web that supports HTTP, HTTPS, FTP and more.



Figure 63: Logo of Squid Proxy

- **GitLab agent for Kubernetes:**

[14] A secure and reliable way to attach a Kubernetes cluster to GitLab.



Figure 64: Logo of GitLab Agent for Kubernetes

- **Helm:**

Helm is the most popular package manager for Kubernetes.



Figure 65: Logo of Helm

- **Prometheus:**

Prometheus is a software application used for monitoring and alerting.



Figure 66: Logo of Prometheus

- **Grafana:**

Grafana is an open source solution for monitoring and analytics.



Figure 67: Logo of Grafana

- **LaTeX:**

[8] A high-quality document preparation and typesetting system for technical grade documents.



Figure 68: Logo of The LaTeX Project

4.6 Difficulties encountered

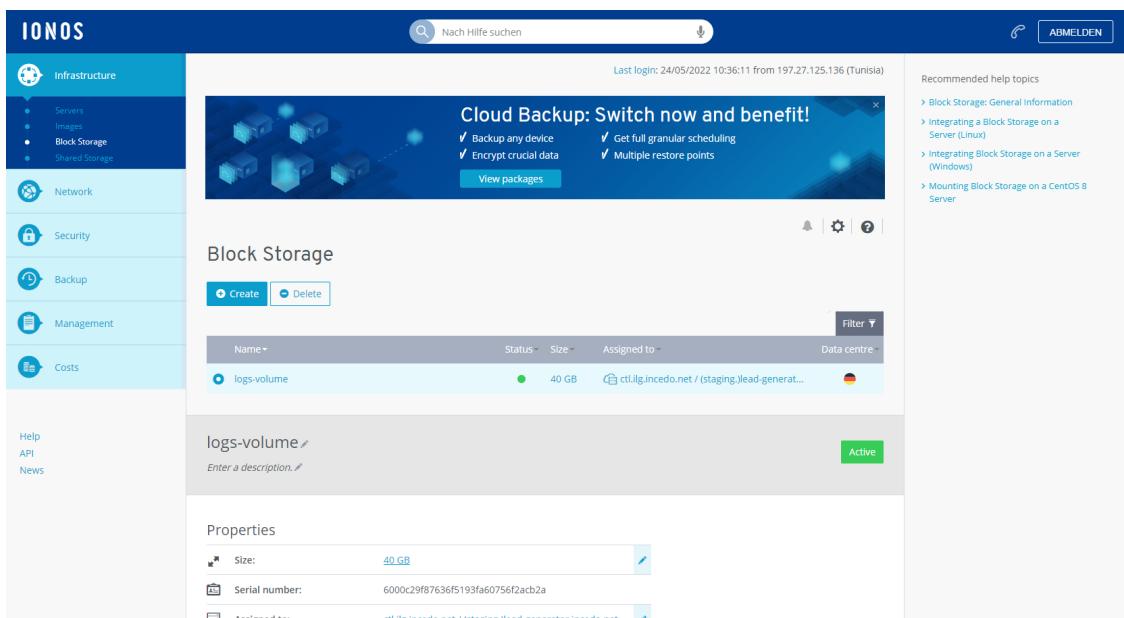
4.6.1 Infrastructure as code

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.

4.6.2 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.



The screenshot shows the IONOS cloudpanel interface. The left sidebar has a blue header 'IONOS' and a navigation menu with 'Infrastructure' selected, showing sub-options like 'Servers', 'Images', 'Block Storage', and 'Shared Storage'. Other menu items include 'Network', 'Security', 'Backup', 'Management', and 'Costs'. Below the menu is a 'Help' section with links to 'API' and 'News'. The main content area has a banner for 'Cloud Backup: Switch now and benefit!' with features like 'Backup any device', 'Encrypt crucial data', 'Get full granular scheduling', and 'Multiple restore points'. Below the banner is a 'Block Storage' section with a 'Create' button. A table lists existing storage volumes, including one named 'logs-volume' which is 40 GB and assigned to 'ctl1g.inkedo.net / (staging).lead-generator...'. The bottom part shows the 'Properties' for 'logs-volume', where the size is set to '40 GB'. There is also a 'Properties' section for the volume itself, showing its serial number and assignment details.

Figure 69: 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.

Conclusion

A successful project comes from a successful development environment as well as a clean production environment with a 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.

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] Microsoft Azure. Devops overview, 2022. <https://azure.microsoft.com/en-us/overview/what-is-devops/#devops-overview>.
- [2] Arthur Busser. What is a container?, 2020. <https://www.padok.fr/en/blog/container-docker-oci>.
- [3] Squid Cache. The official website, 2022. <http://www.squid-cache.org/>.
- [4] Beth Pariseau Emily Mell. What is container management?, 2021. <https://www.techtarget.com/searchitoperations/definition/container-management-software>.
- [5] Ronak Ganatra. Graphql vs rest apis, 2018. <https://graphcms.com/blog/graphql-vs-rest-apis>.
- [6] Incdeo Services GmbH. About incedo, 2022. <https://incedo.de/>.
- [7] GraphQL. The official website, 2022. <https://graphql.org/>.
- [8] The LaTeX Group. The official website, 2022. <https://www.latex-project.org/>.
- [9] Red Hat. What is a ci/cd pipeline?, 2022. <https://www.redhat.com/en/topics/devops/what-cicd-pipeline>.
- [10] MicroK8s. The official website, 2022. <https://microk8s.io/>.
- [11] Nest.js. The official website, 2022. <https://nestjs.com/>.
- [12] Sequelize. The official website, 2022. <https://sequelize.org/>.
- [13] Amazon Web Services. What are microservices?, 2022. <https://aws.amazon.com/microservices/>.

- [14] 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/>.
- [15] Wikipedia. Ionos, 2022. <https://en.wikipedia.org/wiki/Ionos>.
- [16] Wikipedia. Nginx definition, 2022. <https://en.wikipedia.org/wiki/Nginx>.

Appendices

Kubernetes dashboard

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

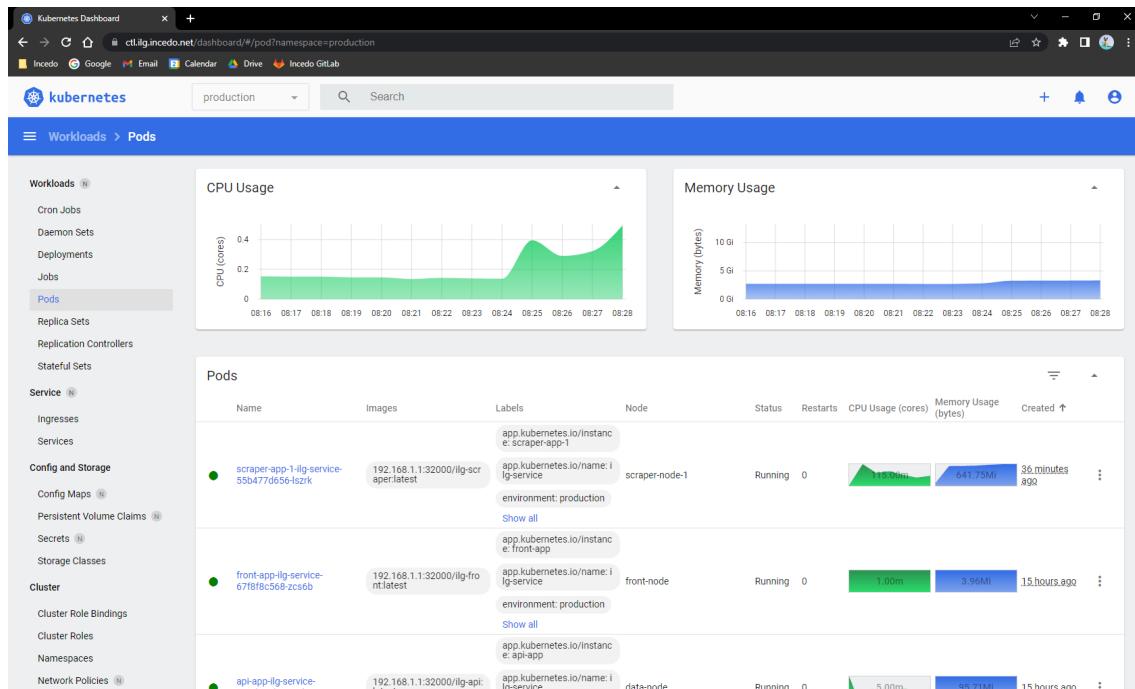


Figure 70: Monitoring the pods with the Kubernetes dashboard

a