Tunisian republic
Ministry of Higher Education
and Scientific Research

University of Carthage
Higher Institute of Applied Sciences and Technology by Mateur

Project Graduation

Introduced to

The Higher Institute of Applied Sciences and Technology of Mateur

In order to obtain the Applied License in
Computer science

university path: ***Computer System and Software***

by

Wael JABER & Nour Ridha DHAOUADI

Rebuild a web site

Defended on June 25, 2021 before the jury commission:

Ms.	**Bchira BEN MABROUK**	President
Mr.	**Slim AMRI**	Reporter
Mr.	**Jamel SLIMI**	Supervisor
Ms.	**Imen JENDOUBI**	Company Supervisor

Academic year 2020-2021n *Order number:IT27*

Contents

End of Studies Project

Anis, Benna
anis.benna@incedo.com

Firas, Jaber
firas.jaber@incedo.com

March 2022

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]

1.1 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. The internship took place fully remotely at Incedo Services GmbH for five months starting from the 1st of February 2022 to the 30th of June 2022 with the purpose of further developing the existing project as well as slowly migrating it to a micro-services architecture. [2]

1.2 Company overview

This section introduces the host company and its offered services.

1.2.1 About Incedo

1.2.2 Incedo Services

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 in LinkedIn. 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 (Incedo Lead Generator) as a SaaS (Software as Service), 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.

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 application is deployed on a single server.

1.4.3 Proposed solution

The solution to these problems is refactoring the whole application to separate sub applications (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 (or user story) from start to finish. They do this by using a kanban board and continuously improving their flow of work.

1.5.4 The choice for ILG

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 a kanban board where they flow from one stage of the workflow(column) to the next. Common workflow stages are To Do, In Progress, In Review, Blocked, and Done. And since this project is very susceptible to changes from outside (LinkedIn), Kanban offered more flexibility than Scrum so that's why the team went with it instead.

Chapter 2

State of the art

Introduction

This chapter will present and study various concepts such as Automation, Web Scraping, Version Control, DevOps, State Machine with an emphasis on the DevOps terminology such as the CI/CD pipelines. 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

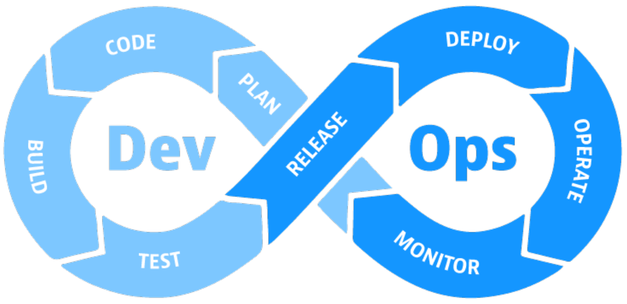
2.2.1 Definition

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 (IAC).

2.2.2 Lifecycle



2.2.3 Container management

Containers have become an integral part of DevOps over the past couple of years but what is a container 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 which always executes in an isolated environment. 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 ?

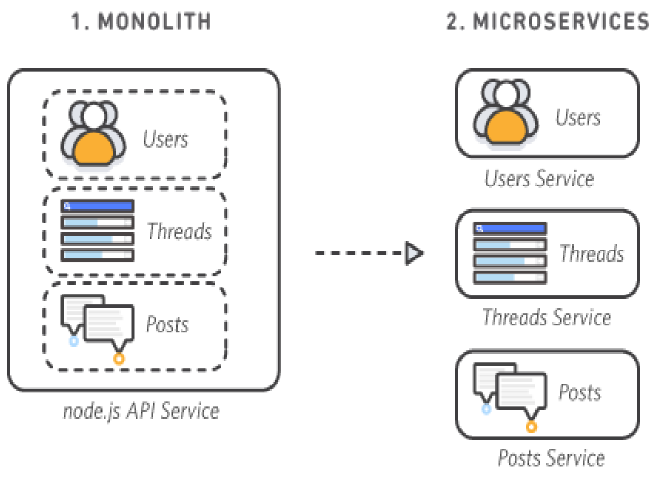
Container management is the process for automating the creation, deployment and scaling of containers. Container management tools such as Docker and Podman facilitate the addition, replacement and organization of containers.

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.

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



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 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.

2.4.1 Version Control: Git vs SVN

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

Table 2.1: Comparative study between existing methodologies

	Description	**Advantages**
Git	Git is a distributed version control system which means that when coloning 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.

At the end, both are great options. However, we will be using Git which is the VCS the company uses.

2.4.2 Container management: Docker vs Podman

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

Table 2.2: 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 daemon-less 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.

As a result, we will also be sticking to Docker in this project.

2.4.3 Browser automation: Playwright vs Puppeteer

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

Table 2.3: Playwright vs Puppeteer 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.

Chapter 3

Analysis and specification of requirements:

3.1 Analysis

3.2 Requirements

3.2.1 SprintOne

3.2.1.1 Sprint Story and Objective

3.2.1.2 Sprint Analysis

3.2.1.3 Sprint Kanban Board

3.2.1.4 Sprint Review

3.2.2 SprintTwo

3.2.2.1 Sprint Story and Objective

3.2.2.2 Sprint Analysis

3.2.2.3 Sprint Kanban Board

3.2.2.4 Sprint Review

3.2.3 SprintThree

3.2.3.1 Sprint Story and Objective

3.2.3.2 Sprint Analysis

3.2.3.3 Sprint Kanban Board

3.2.3.4 Sprint Review

Chapter 4

Realization :

4.1 General architecture

4.2 Work environment

4.3 Difficulties encountered

Chapter 5

Conclusion :

5.1 General conclusion

Webography

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy.* San Val, 1995.

[2] Veronika Földváry Ličina, Toby Cheung, Hui Zhang, Richard de Dear, Thomas Parkinson, Edward Arens, Chungyoon Chun, Stefano Schiavon, Maohui Luo, Gail Brager, Peixian Li, Soazig Kaam, Michael A. Adebamowo, Mary Myla Andamon, Francesco Babich, Chiheb Bouden, Hana Bukovianska, Christhina Candido, Bin Cao, Salvatore Carlucci, David K.W. Cheong, Joon-Ho Choi, Malcolm Cook, Paul Cropper, Max Deuble, Shahin Heidari, Madhavi Indraganti, Quan Jin, Hyojin Kim, Jungsoo Kim, Kyle Konis, Manoj K. Singh, Alison Kwok, Roberto Lamberts, Dennis Loveday, Jared Langevin, Sanyogita Manu, Cornelia Moosmann, Fergus Nicol, Ryozo Ooka, Nigel A. Oseland, Lorenzo Pagliano, Dušan Petráš, Rajan Rawal, Ramona Romero, Hom Bahadur Rijal, Chandra Sekhar, Marcel Schweiker, Federico Tartarini, Shin-ichi Tanabe, Kwok Wai Tham, Despoina Teli, Jorn Toftum, Linda Toledo, Kazuyo Tsuzuki, Renata De Vecchi, Andreas Wagner, Zhaojun Wang, Holger Wallbaum, Lynda Webb, Liu Yang, Yingxin Zhu, Yongchao Zhai, Yufeng Zhang, and Xiang Zhou. Development of the ASHRAE Global Thermal Comfort Database II. *Building and Environment*, 142:502–512, sep 2018.