4Table of Contents

1. Introduction
   1. Immowelt AG
   2. About the project “Baukarte” (Interactive building profile)
   3. Motivation
   4. Problem statement
   5. Aim of the work
   6. Methodological approach
   7. Structure of the thesis
2. Theoretical background
   1. Web Crawler
      1. Anatomy of a web crawler
      2. Types of web crawler
      3. Incremental web crawler
      4. Challenges
   2. Cloud Computing
   3. Amazon Web Services and Microsoft Azure in comparison
3. Design and Conception
   1. Requirements analysis
      1. Technical requirements
      2. Non-technical requirements
      3. The key criteria for a cloud-based solution
      4. Evaluation of candidates based on the key criteria
   2. Conception
      1. Conceptual crawler architecture
      2. Conceptual crawling process
   3. AWS and AWS Services
      1. Description of adopted services
      2. Crawler architecture on AWS
4. Implementation and realization
   1. Development environment and Frameworks
   2. Implementation of the web crawler
   3. Implementation for monitoring and scheduling
5. Conclusion
   1. Testing and evaluation of the web crawler
   2. Evaluation of the adopted AWS Services

**1.3 Problem statement**

The use case of the web crawler implemented in this thesis is to extract information from official announcement about new building permissions. While running web crawler locally is fine or do-once tasks and small amount of data, where the crawl can be triggered manually, reoccurring tasks and automatic scheduling can be optimized with deploying into cloud. Cloud also provides greater and flexibility and alternative IP address.

**1.5 Aim of the work**

This thesis proposes a Web Crawler Cloud Computing architecture which uses cloud-hosted […]

The secondary aim of the case study described in this thesis, is to determine how suitable is the Amazon Web Services suitable for web crawler and the potentials of Amazon Web Services.

**1.7 Structure of the work**

The rest of the thesis is structured as follows. Chapter 2 provides the theoretical background of web crawler and cloud computing. Chapter 3 describes the web crawler architecture. Chapter 4 focuses implementation of the web crawler in the cloud. Chapter 5 show experiments and evaluations of the web crawler. Chapter 6 summarizes result of this thesis and presents possible further works.

**2. Theoretical Background**

This chapter provides

**2.1 Web Crawler**

This chapter introduces website scraping in general.

What is Web Scraping?

In theory, web scraping is the practice of gathering data through any means other a program interacting with an API.

What is Web Crawler?

One of the way to access the Internet is through a browser, but for gathering and processing a large amounts of data an automated web scraper is needed.

The need to scrape websites arises with the increased amount of information generated of the Internet. The widely known scrapers is Google. These scrapers go through almost the whole Internet, scan every web page, extract information from it, and build an index that you can search

A Web Crawler is an automatic web object retrieval system. The two main goals of a web crawler are:

To seek out new web objects

To observe the changes in previously discovered objects.

**2.1.1 Anatomy of a web crawler**

Link Extractor will extract links from a given pages.

Media Extractor:

**2.1.2 Types of Crawlers:**

It is known that there exists many types of web crawler:

**2.1.3 Incremental Web Crawler**

The incremental crawler periodically revisits the pages. During its crawls, it may also add new pages into its data collection in order to keep the data collection fresh. In our case […]

**2.1.4 Challenges**

One of the challenges faced by web crawler is that website layout differs. It is hard to create a good-fit-for-all scraper because every website has a different layout, uses different HTML IDs to identify their layout. Moreover, many websites change their layout frequently. In these cases, the only option is to revisit the code and adapt it to changes of the target websites.

Downloading unknown files off the Internet might introduce potential malware. As always, a security mechanism to certify media downloaded is needed

IP Blocking

Most efforts to stop scrappers from accessing websites focus on detecting the difference between humans and bots.

All of the points presents challenges for data mining in general and web crawling in particular. These are the focus of this thesis.

**3.1.1 Technical requirements**

In this part the technical requirements for which must be created to overcome the challenges mentioned in the 2.1.4 Challenge.

Most websites provide a file called robots.txt, which is used to tell web crawlers what they can scrape and what they should not touch.

Let`s see one example of a robots.txt content

[….]

Integrating the a robots.txt parser and checking every URL before accessing helps automating the task of honoring the website owner`s request what to access.

Handling Logins and Cookies

Most modern websites use cookies to keep track of who is logged in and who is not. Website doing this might attempt to cut off scrapers that display abnormal behavior, such as completing forms too quickly, or to visit too many pages.

**3.1.2 Non-technical requirements:**

The usage of Amazon Web Services brought up the need for codifying server, storage, and networking infrastructure. Automated provisioning helps reduce the time it took to deploy a full-featured cloud environment. Furthermore automated deployment will significantly safer, since it will be more consistent, more repeatable, not prone to manual error in compare to using AWS Management Console or AWS CLI.

**3.2.2 Conceptual crawling process:**

After the requirements are defined, it is important to plan the application structure and behavior.

Preparation

As mentioned the chapter, before launching a crawler, the website´s term and conditions should be taken in to consideration.

Storing data

In order to make the web scrapers useful, the ability the store and interact with large amounts of data is incredibly important.

There are two main source of data to store: meta data and actual PDFs document.

There are two main ways to store PDFs documents: by reference, and by downloading the file itself. The file can be store by reference by simply storing the URL that the file is located at. This introduces several advantages:

Scrapers run much faster, and require much less bandwidth, when they don´t have to download file.

The load on the host servers is lessened by avoiding large file downloads.

However storing by reference also has its own disadvantages:

Embedding the URLs in website or application is known as hot linking and doing it is a very quick way to get you in hot water on the internet. [… For data mining application make URLs requests]

The file hosted at any particular URL is subject to change. This might lead to embarrassing effects. For example if the PDF content is needed for further research, it might eventually go missing or be changes to something completely irrelevant at a later date.

[…Conclusion]

The meta data such as URL, parent URL or domain or simply robots.txt should be stored in way that support data analyzing and indexing late on. An effective way to achieve this is through data modelling and a good choice of database type.

**3.3.1 Description of adopted services**

Our application for web crawling requires […]

AWS EC2

Each time an instance get started, a new IP address is allocated to that instance. This can be an issue, if an application needs to be accessed by end users or other system. On the other hand, we can have benefits from this.

AWS Lambda

AWS Batch

**4.1 Development environment and Frameworks**

Tesseract

Tesseract is an OCR library sponsored by Google, is widely regarded to be the best, most accurate, open source OCR system available.

Requests

Excellent for setting headers. HTTP Headers are list of attributes, or preferences, sent by you every time a request to web server is made. In order for web crawler to look like human headers should be customized.

**4.4 Proposal for deploying web crawler on Amazon Web Services**

One of the tool that help deploy infrastructure on AWS is HashiCorp´s **Terraform**. Terraform enables writing templates describing an environment, performing dry run to see what is about to happen, deploy the template, and make any adjustments where necessary-all of this without leaving the shell prompt.

*Operations:*

There is five main Terraform operation to be examine in our use case

*Validating a template*: Performing a basic syntax check

*Testing (dry-run):* displays what would happen during an actual deployment

*Initial deployment:* deploy a template

*Updating a deployment:* a change to our template will become necessary later on. Fortunately, all that is need for this is to update and re-deploy the given template.

*Removal of a deployment:* Since additional charge is not expected and the tasks are mainly in batch, it is recommended to remove AWS resources after the task is done.

*How to manage Terraform state?* Every time you run a Terraform, it records information about what infrastructure it created in a *Terraform state file*

*Downsides of using Terraform as a team on a real product*

*Shared storage for state files:* to be able to use Terraform to update your infrastructure, each of your team members needs access to the same Terraform state files. That means you need to store those files in shared location.

*Locking state files:*  As soon as data is shared, you run into a new problem: locking. Without locking, if two team members are running Terraform at the same, you may run into race conditions as multiple Terraform processed make concurrent updates to the state files, leading to conflicts, data loss, and state file corruption.

*Isolating state files:* When making changes to your infrastructure, it’s a best practice to isolate different environments. For example, when making a change in a testing or staging environment, you want to be sure that there is no way you can accidentally break production. But how can you isolate your changes if all your infrastructure is defined in the same Terraform state file.

Solution: Shared Storage for State Files

The most common technique for allowing multiple team members to access a common set of files is to put them in version control (e.g Git). With Terraform state, this is a bad idea for two reasons.

*Manual Error:* It’s too easy to forget to pull down the latest changes form version control before running Terraform or to push your latest changes to version control after running Terraform.

*Secrets:*  All data in Terraform in state files is stored in plain text. This is a problem because certain Terraform resources need to store sensitive data. For example, if you use the aws\_db\_instance resource to create a database, Terraform will store the username and password for the database in a state file in plain text.

Instead of using version control, the best way to manage shared storage for state files is to use Terraform’s built-in support Remote State Storage.