Faculty Informatics
Bachelor of Science – Business Information Systems

Bachelor Thesis

Development of a distributed cloud-based system
for crawling public real estate relevant data
for a large German real estate portal

Name:	Nguyen, Viet Hoa
Student number:	2896037
Email:	nguyenviethoa95@gmail.com
Submission date:	12.07.2019
Cooperating partners:	Immowelt AG
	Axel Schwanke
	Maxim Fridental
First examiner:	Prof. Dr. Michael Zapf
Second examiner:	Prof. Dr. Friedrich Stappert

Acknowledgements

My bachelor thesis was written from March 2019 to August 2019 in the product management department at Immowelt AG
I would like to express my most sincere thanks to the Big Data team which always provided me a nice working environment, with a lot of help, and support. A special thanks for my advisors Axel Schwanke, Maxim Fridental, and my colleague Stefan Nagel for their professional advices.
A special thanks to my advisor Prof. Dr. Michael Zapf for being always ready to help, and to support my work with patience.

Declaration

“I affirm that this bachelor thesis was written by myself without any unauthorized third-party support. All used references, and resources are clearly indicated. All quotes, and citations are properly referenced. This thesis was never presented in the past in the same or similar form to any examination board. I agree that my thesis may be subject to electronic plagiarism check. For this purpose, an anonymous copy may be distributed, and uploaded to servers within, and outside the Nuremberg Institute of Technology”

Nuremberg, 12.07.2019

____

Nguyen, Viet Hoa

Table of Contents

List of Abbreviations
List of Figures
List of Tables

1 Introduction **1**
1.1 Immowelt AG 1
1.2 About the "Baukarte" project 1
1.3 Purpose of the web crawler 2
1.4 Research method 3
1.5 "Amtsblatt" Data Source 1
1.6 Structure of the thesis 6
2 Theoretical Background **7**
2.1 Web Crawling 7
2.1.1 Motivation 7
2.1.2 Concept of crawling 8
2.1.3 Types of web crawlers 9
2.2 Serverless Computing 9
2.2.1 Definition 10
2.2.2 Serverless architecture 11
2.2.3 Serverless Application Programming Model 12
2.2.4 AWS Serverless Services in comparison 12
2.3 Function as a Service 15
3 Design, and Conception **18**
3.1 Requirements 18
3.2 Crawler design 19
3.2.1 Initializer Module 22

3.2.2 Website Watcher Module........22
3.2.3 Hyperlink Collector........23
3.2.4 Hyperlink Processor Module........25
3.2.5 Data Storage........26
3.3 Serverless deployment development........26
3.3.1 Purpose........27
3.3.2 Goal........27
3.3.3 Concept........27
4 Implementation........29
4.1 Development environment, and Frameworks........29
4.2 Implementation of the core functions........30
4.2.1 Initializer Module........30
4.2.2 Website Watcher Module........31
4.2.3 Hyperlink Collector Module........32
4.2.4 Hyperlink Processor Module........32
4.2.5 Job Queue........33
4.2.6 Adopted AWS Services........36
4.3. CI/CD Pipeline........41
4.3.1 Frameworks, and development environment.........41
4.3.2 AWS Resources CI/CD pipeline........43
4.4 Evaluation........**Error! Bookmark not defined.**
5. Conclusion........ Error! Bookmark not defined.
References........53

List of Abbreviations

API	Application Programming Interface
AWS	Amazon Web Service
AWS EC2	Amazon Elastic Compute Cloud
AWS S3	AWS Simple Storage Services
AWS SQS	Amazon Simple Queue Service
CLI	Command Line Interface
FaaS	Function as a Service
HTML	Hypertext Markup Language
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
IaC	Infrastructure as Code
URL	Uniform Resource Locator

List of Figures

Figure 1: Project concept 2
Figure 2: Phrases of the research conducted in the actual project 3
Figure 3: Bochum Official Journal HTML Code 1
Figure 4: Screenshot of Bochum Official Journal 2
Figure 5: Screenshot of Göttingen Official Journal 2
Figure 6: Screenshot of Göttingen Official Journal Year 2019 3
Figure 7: Screenshot of Bremen Official Journal 4
Figure 8: Bremen's Official Journals 2019 5
Figure 9: Flow chart of a crawler. Source: [36] (p.4) 8
Figure 10: Gartner Hype Cycle for Emerging Technologies. Source: Gartner 10
Figure 11: Serverless application layers 11
Figure 12: FaaS Processing Model 16
Figure 13: Workflow of the web crawler 19
Figure 14: A generic work queue. Adapted: Burns [12] 20
Figure 15: Components of web crawler 21
Figure 16: Workflow of Website Watcher Module 23
Figure 17: Workflow of the Hyperlink Collector Module 24
Figure 18: Workflow of Hyperlink Processor 26
Figure 19: Initializer Module implemented with AWS Services 31
Figure 20:Website Watcher Module implemented with AWS Services 31
Figure 21.Hyperlink Collector Module implemented with AWS Services 32
Figure 22: Hyperlink Processor Module implemented with AWS Services 33
Figure 23: SQS as Lambda's trigger on the left handside 33
Figure 24: AWS SQS as trigger for Lambda function 34
Figure 25: Simplify architecture of a running Lambda function 36
Figure 26: Configurations of AWS Lambda Function 37
Figure 27: Result after invocation of Lambda function 37
Figure 28: GitLab CI/CD pipelines 41
Figure 29: Terraform API call 42
Figure 30: Terraform project repository in GitLab 43
Figure 31: Deployment pipeline for AWS Infrastructure 44
Figure 32: Deployment pipeline - AWS Lambda function 46
Figure 33: Screenshot - PDF folders stored in S3 Bucket 50
Figure 34: Screenshot - Content of Berlin's folder 50
Figure 35: Json structure - PDF metadata 51

List of Tables

Table 1: AWS S3 Pricing for region eu-central-1 (Frankfurt) at the time of writing.......39
Table 2: Project structure of AWS Lambda function.................**Error! Bookmark not defined.**

1 Introduction

This chapter gives an overview of the company Immowelt AG, explain the motivation behind this project, and discusses the scope of my work.

1.1 Immowelt AG

Immowelt AG offers complete IT solutions for the real estate industry. The core business of the company is the real estate portals *immowelt.de*, *immowelt.at*, and *immowelt.ch* as well as *bauen.de*, *ferienwohnung.com*, and *wohngemeinschaft.de*. This reflects in the second business area of the company - the development of CRM software for the real estate agents *estateOffice*, *estatePro*, and *immowelt i-Tool*. The main portal *immowelt.de* went online in 1996. It allows the private owner to offer or to rent private real estate property. Besides, the user will find extensive information on the topics of living, building, and financing, as well as price overview of the real estate market [1]. In 2000, DataConcept GmbH was renamed Immowelt AG. In 2015, Immowelt AG merged with its competitor Immonet GmbH under the name Immowelt Holding AG. Today more than 500 employees work in two locations in Nuremberg, and Hamburg [2].

1.2 About the “Baukarte” project

With the vision of being the number one real estate portal in Germany, it is essential to continually come up with innovative features, which has not been developed by any other real estate portals. One of those features could be the interactive building profile for Germany.
With this feature, the real agent estate should be able to keep track of the upcoming plan for new buildings with a visual representation on an interactive map. Nowadays, interactive maps used not only on geo-data-specialized websites but also on real estate portals and become a trend for visualizing mass data. Building permits, and public real estate plans in the neighborhood often have a significant impact on the prices of real

estate property. For example, l, and value often goes up with planned increases in residential building permits.

Project concept

The end-product of the project is planned to be a new feature on the immowelt.de website for real estate agents. The official announcement should be collected automatically (1). After that, the building permit will be extracted (2), the raw text will be tokenized (3), transformed (3), and stored in the database (3). The information will be visualized on an interactive map (4).

Figure 1: Project concept

The project team consisted of three students from the Nuremberg Institute of Technology (Technische Hochschule Nürnberg Georg-Simon-Ohm). The company representative was the project's product owner from the product management Department and interacted closely with the student team. The first student took responsibility for the presentation layer, which visualizes building a permission based on Google Map API. The second student developed an automatic text mining pipeline to create visualizing components from building permissions in PDF. This thesis contributes to the first step of the project "Automated data collection". It should deliver an automated mechanism to collect the data for the application. The overall architecture of the "Baukarte" project can be viewed in Appendix A and B.

1.3 Purpose of the web crawler

The use of the web crawler is inevitable when it comes to collecting massive data set. The use case for the web crawler implemented in this thesis is to extract information from an official announcement containing new building permissions. While running web crawler on a local machine is fine or do-once tasks, and a small amount of data, where the crawling process can be triggered manually. However, this is not a sustainable, and reliable solution for retrieving a huge amount of data. Web crawler can be optimized with deploying into the cloud to reduce operational management and increase parallelism. Cloud computing also provides greater flexibility in term of computing capacity, and IP address.

1.4 Research method

The purpose of this section is to introduce the methodology for developing a distributed web crawler using cloud services. The research attempts to experiment and design a possible solution for the distributed web crawler, and practices for adopting cloud services. The solutions should also include technical implementation fulfilling the practices of continuous development, and deployment process. As such, the thesis is aimed to deliver a solution for a practical problem, and "the problem cannot be proven mathematically, and tested empirically" [3]. The research method of this thesis is inclined toward the methodology for the system development research published by Nunamaker, Chen, and Purdin [3]. Figure 2 depicted the five different phrases of system development research, and corresponding phases in the project.

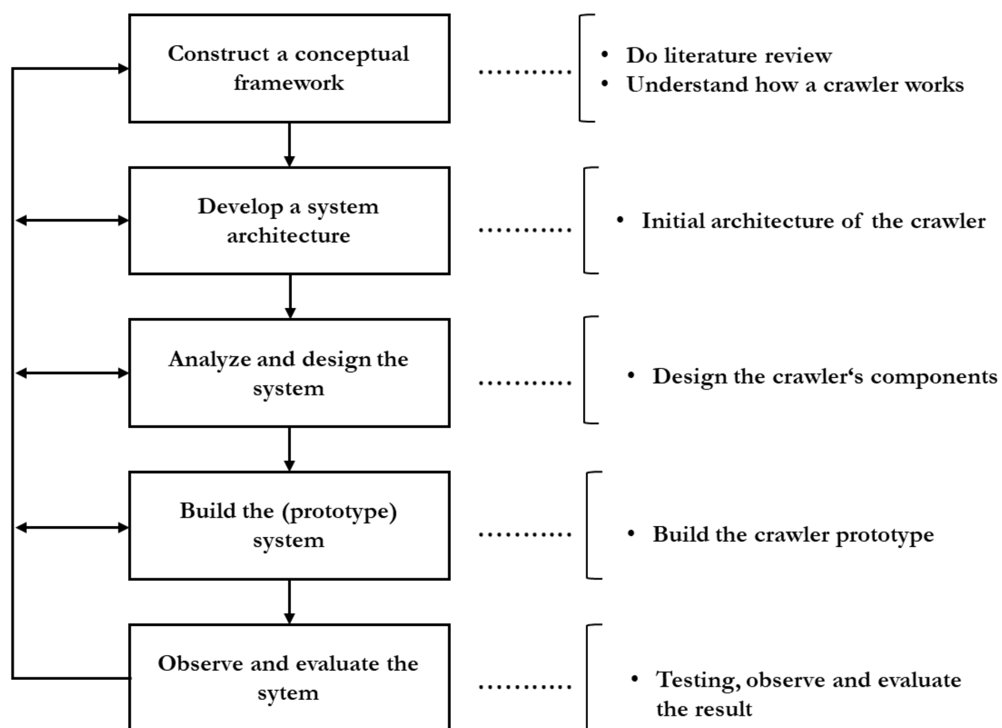


Figure 2: Phrases of the research conducted in the actual project
Adapted from: Nunamaker et al.

First, and foremost, a literature review is carried out to study how a web crawler works, and how should it be designed. The result of the first stage is a reference architecture show in Figure 9 in section 2.2.1. The second stage, which is displayed in Figure 15 in section 3.2, designs the architecture to be implemented. Functional, and non-functional requirements must also be defined, and identified in this stage, which are presented in section 3.1. The third stage involves designing the crawler components. The fourth stage presented in chapter 4 involves in translating the chosen design into code. Lastly, the developed system will be evaluated, and compared with the objective at the beginning, the test results will be summarized in chapter 5.

1.5 “Amtsblatt” Data Source

In Germany, a municipality (“Stadt”, “Landkreis”, “Gemeinde”) has the responsibility to publish their official journals (Amtsblatt). In these journals, the information about new regulations, construction projects of the municipal council can be found. An example of a typical official can be found in Appendix B. It also includes the real estate relevant information such as land-use plan, development plan, and construction zoning map. To some extent, these journals can be viewed on their website, and are available for download in PDF format. Some journals do not have an online version and are available only in printed version.

In this section, three different “Amtsblatt” website that provides a list of “Amtsblatt” (official journal) will be compared. The sites we analyzed were Bonn Amtsblatt[1], Bochum Amtsblatt[2], Bremen Amtsblatt[3]. These websites represent the most common official journal website layout. In the section below, a general analysis of the HTML layout will be presented for each of the three websites.

a. Bochum Official Journal

As can be seen in Figure 4, the layout consists of a site header with general information, searching, and navigation. The horizontal navigation on the left-h, and side enables the user to navigate to other sites of the city municipality council. The yellow region is the main content of the page, which displays the list of official journals chronologically. The page has a basic Hyper Text Markup Language (HTML) structure, all the official journals are ordered in a list using the <ul> tag. Each of <ul> tag contain a nested <a> tag which redirected the user to the webpages where the official journals can be seen.

```
▼<ul class="bulletlist">
  ▼<li>
    ▼<span>
       <a href="/C12571A3001D56CE/vwContentByKey/W2BD4H8D437BOCMDE/$FILE/
       amtsblatt_24_2019.pdf" target="_blank" title="Hinweis: Die pdf-Datei
       öffnet sich in einem neuen Fenster." class="link-download">Ausgabe 24 /
       2019 vom 17. Juni 2019</a> == $0
     </span>
   </li>
 </ul>
```

Figure 3: Bochum Official Journal HTML Code

This type of website can be classified as a simple listing style website, where all of the documents are listed directly in the main page. This basic HTML structure makes it easier to identify the official journals through the <a> tags.

[1] https://www.bonn.de/service-bieten/aktuelles-zahlen-fakten/amtsblatt.php
[2] https://www.bochum.de/amtsblatt
[3] https://www.amtsblatt.bremen.de/

Figure 4: Screenshot of Bochum Official Journal

b. Göttingen Official Journal

The screenshot in Figure 5 shows the site layout of Göttingen Official Journal. As can be seen, the official journals list cannot be viewed directly on the main page. They can be only detected if the user clicks on one of the three boxes ordered by year.

Figure 5: Screenshot of Göttingen Official Journal

Once the user clicks on the boxes, they are redirected to another website, where the list of documents can be seen, and are available to download, as seen in Figure 6.

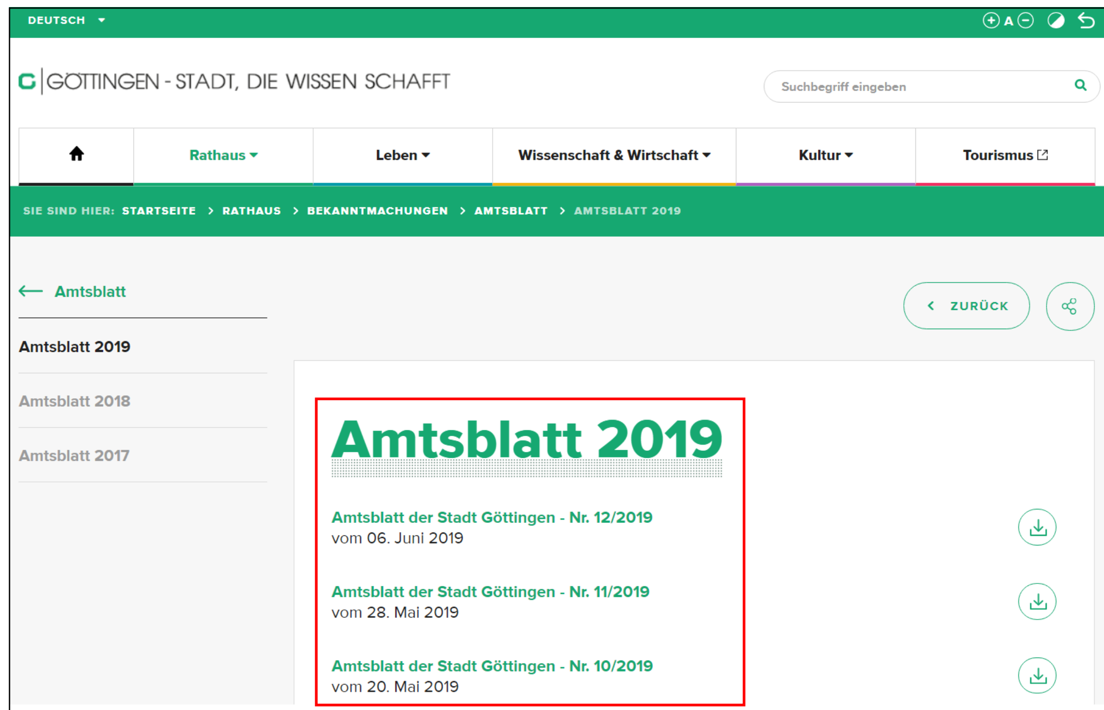


Figure 6: Screenshot of Göttingen Official Journal Year 2019

c. Bremen Amtsblatt

In the screenshot of Bremen Official Journal, the list of official journals is also not available on the main page. First, the user is required to choose a year in from the dropdown list (marked yellow) and clicks on "Suche" button. Once a year is chosen, the page loads again, and list of documents of the chosen year are showed underneath the dropdown list in Figure 8.

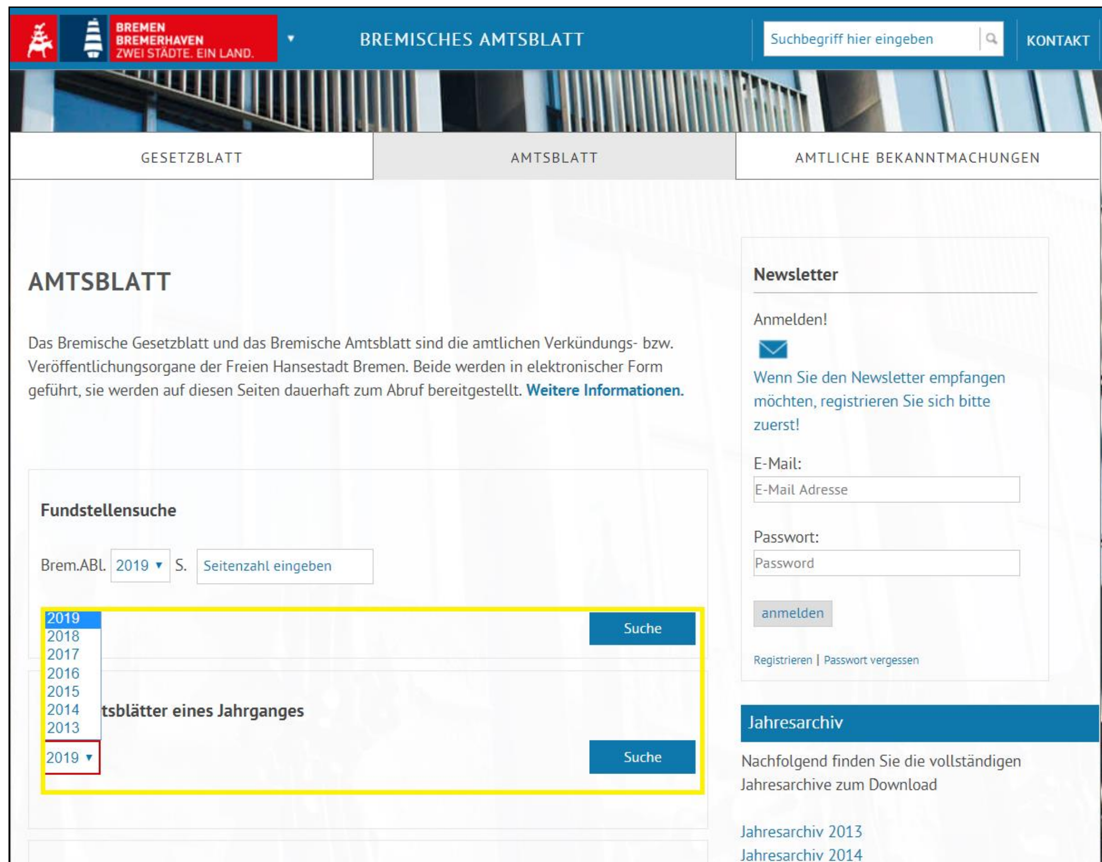


Figure 7: Screenshot of Bremen Official Journal

This responsive web design of this official journal website, which contains the JavaScript elements such as the search box makes it extremely complicated to be crawled. Because the link to the PDF files cannot be found directly in the HTML code on the main page. Firstly, the crawler must be able to automatically identify that there is a dropdown list in the HTML code. Secondly, it must simulate a user action of choosing the link in the dropdown, and a click on the search button. This set of action could be accomplished on a single website with some web browser automation tool such as Selenium [4]. However, this solution also not generic when it is applied to several websites. Because the automation tool can only work with the condition that the ID of the dropdown element is provided. It would be very time consuming to manually investigate the HTML code in order to retrieve the ID of each dropdown element of each website. Furthermore, It would be impossible to write a generic crawler fulfilled this requirement.

[4] https://www.seleniumhq.org/

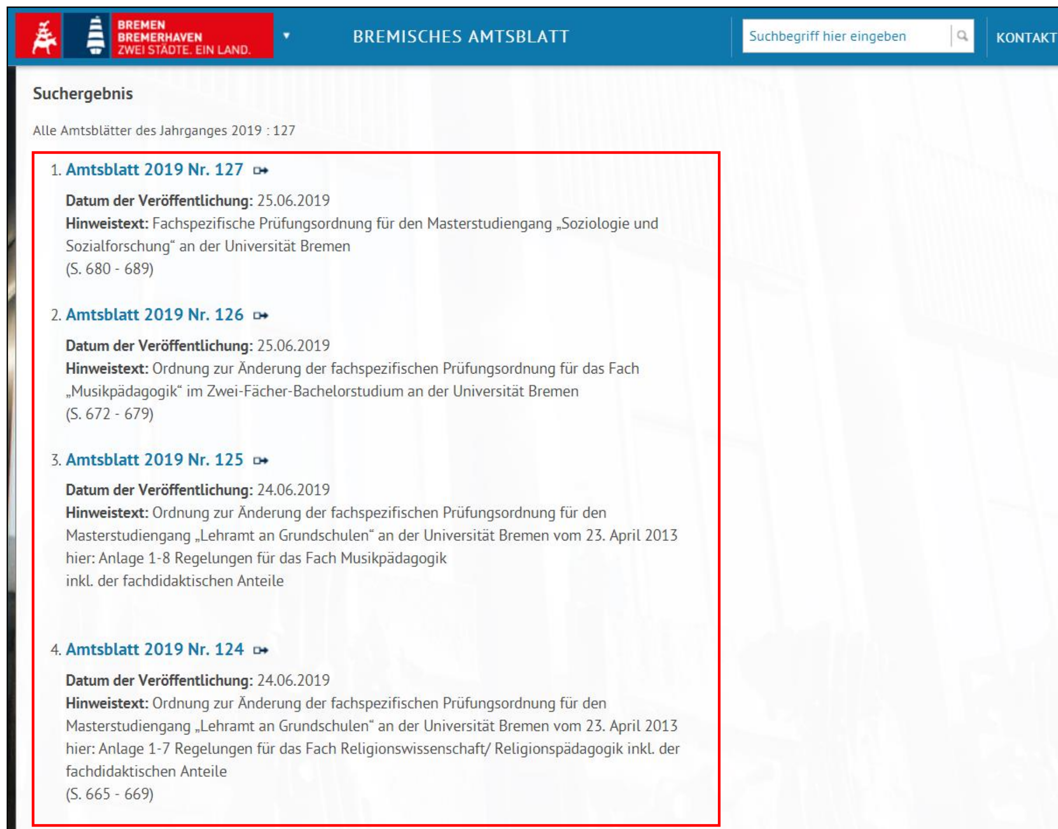


Figure 8: Bremen's Official Journals 2019

d. Scope of the thesis

The examples mentioned above only cover a small fraction of possible website layouts available on the Internet. Because there is no standard for designing an official journal site. The wide variety of website layout makes it impossible to make a generic crawler that is suitable for all web layouts. It would be very unpractical to review and handle all kinds of website layouts. Notably, the website layout that has some JavaScript elements in it requires individual crawler configuration. In limited time of a bachelor thesis, it is unrealistic to be able to develop a crawler for three types of website layouts. As a result, only 50 official journal websites of 50 municipal councils will be chosen to be crawled. These websites share the same overall layout of the Bochum Official Journal site, in which the list of documents is available directly on the main page. This type of HTML structure contains no redirecting to other pages allows for easy identification, and retrieval.

1.6 Structure of the thesis

This thesis is arranged into fives chapters including this chapter:

Chapter 2 – Theoretical Background introduces the terminology, technologies, and concepts which will be used throughout this thesis. It introduces terms like cloud computing, serverless computing, and web crawling.

Chapter 3 – Design and Conception provides the necessary designs for the implementation in chapter 4. First, it addresses the expectations for a project-specified web crawler. From the system perspective, some functional requirements must be addressed with care to guarantee that the crawler meets all the functionalities needed. The functional requirements support the design process of the web crawler later in this chapter. The design presents the overall architecture of the web crawler and details about each module. The design for the deployment pipeline of the crawler are also introduced at the end this chapter.

Chapter 4 - Implementation describes the process of transforming the web crawler's design in previous chapter with AWS Services. In addition, it also describes how the deployment pipeline is implemented.

Chapter 5 – Conclusion examines the web crawler and the deployment pipeline in terms of performance. Based on the examinations, the limitations and suggestions for further improvements, before it ends the thesis with a personal conclusion about the thesis.

2 Theoretical Background

This chapter lays the theoretical foundation for the design, and implementation of the web crawler in this thesis. The outline of this chapter is as followings: the first section will introduce the motivation of web crawling in this thesis and gives a brief description of how web crawlers work under the hood. The second section of this chapter will introduce Serverless Computing as an umbrella term. The last section reviews two important technologies Serverless and Function as a Service and explains how these technologies can be utilized to build our application.

2.1 Web Crawling

2.1.1 Motivation

With the exponential growth of information sources available on the World Wide Web, exploring web data becomes an integral part of many big enterprises, it can range from collecting customer opinions about products, exploring data for scientific research or even to build an application on top the data collected. Furthermore, the unstructured information from Web pages needs to be transformed into structured information that can be used in a subsequent stage of analysis. An automated program as known as a web crawler, which scans through the web, and downloads the pages which can be reached by the links, is the key to massive data collection. The two main reasons are:

A user needs to think, grab the mouse, point to the link, click on it, and finally copy paste content of a web page, whereas a computer program can perform this in milliseconds. It is straightforward to use an automated program to request, and parse web content.

Using a web browser to search web content is a visual and intuitive but not very useful way of gathering massive data from the World Wide Web since the content rendering process is long-running.

The largest application field of web crawler is for commercial search engines such as Google, Bing, and Yahoo!, which are used by millions of users daily around the world. Google, for example, developed its crawler known as Googlebot, which traverses web pages by following hyperlinks, and stores web documents that are later indexed to optimize the search process. Although Googlebot is believed to be the first large-scale web crawler in the world, the history of web crawler can be traced back long before the launched of Google in 1997.

2.1.2 Concept of crawling

Figure 9 shows the flow of a basic crawler. In this most straightforward form, a crawler starts from a set of seed pages (URLs), and then uses the links within them to fetch other pages. The link in these pages are, in turn, extracted, and the corresponding pages are visited. The unvisited URLs are called the frontier. The list is initialized with the seed URLs which may be provided by the user or another program.

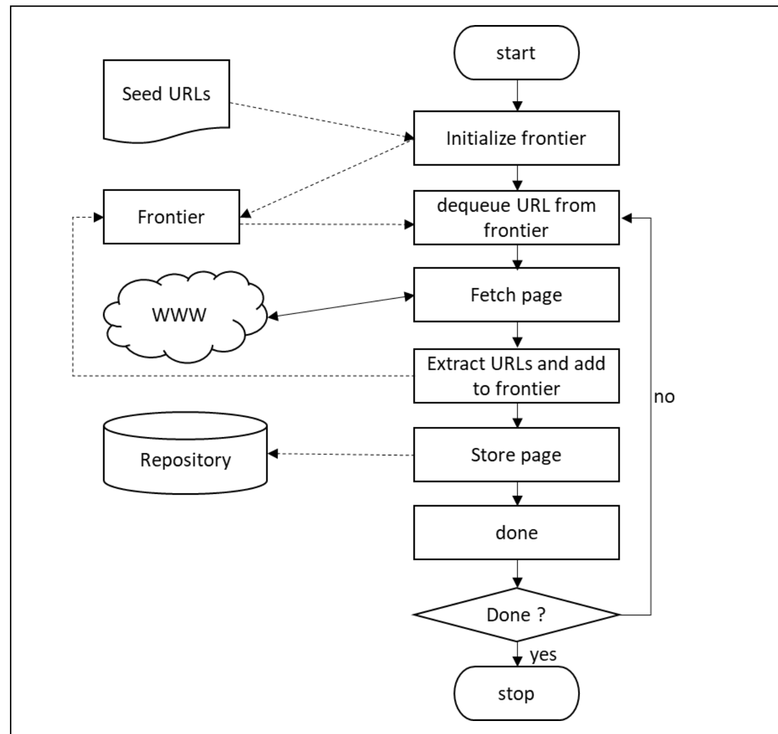


Figure 9: Flow chart of a crawler. Source: (p.4)

In each iteration of its main loop, the crawler picks the next URL from the frontier, fetches the page corresponding to the URL through HTTP, parses the retrieved page to extract its URLs, adds newly discovered URLs to the frontier, and stores the page in a

local disk repository. The crawling process may be terminated when a certain number of pages have been crawled.

2.1.3 Types of web crawlers

Web crawlers may differ from each other in the way they crawl web pages. This is mainly related to the final application that the web crawling system will serve. Crawlers can be classified into four categories [4]:

Focused Crawler is for discovering and retrieving web pages that are related to a specific area of interest. The relevance of the web pages must be determined before being crawled. This kind of crawler requires less hardware and bandwidth resources.

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. The crawler also replaces old, and less important pages by new, and more relevant pages. The advantage of incremental crawler is that data enrichment is achieved.

Distributed crawler utilizes distributed computing to distribute the workload on many crawlers. There is a central server manages the communication, and synchronization of the workers. Distributed crawler is robust against system crashes, and other events, and is adaptable to many crawling applications.

Parallel crawler executes multiples crawlers simultaneously. This kind of crawler offers the solution for retrieving web pages content in a reasonable amount of time.

2.2 Serverless Computing

It is worth to mention that the focus of this thesis is on Amazon Web Services (AWS), because Immowelt AG has chosen AWS as their main computing platform. With the ambition to migrate some of the core infrastructures into the cloud environment. It is practical to have a look at the existing cloud computing models in advance. One of the emerging models is Serverless Computing. As such, it is recommended that IT company start to learn its opportunities, and limits, identify best practices, and pilot test cases to build knowledge, and skills.

Serverless is listed in the top 10 Trends and Impacting Infrastructure & Operation for 2019 by Gartner in one of their recent articles [5]. It is predicted to become a mainstream between 2010, and 2022, with 10% of IT organizations already using serverless

computing. This can be reflected in the fact that the three largest cloud providers all have their own platform for serverless computing [5] [6] [7]. This reflects the explosion of interest in Serverless. Gartner classified serverless as being in its initial cycle phase, "innovation trigger".

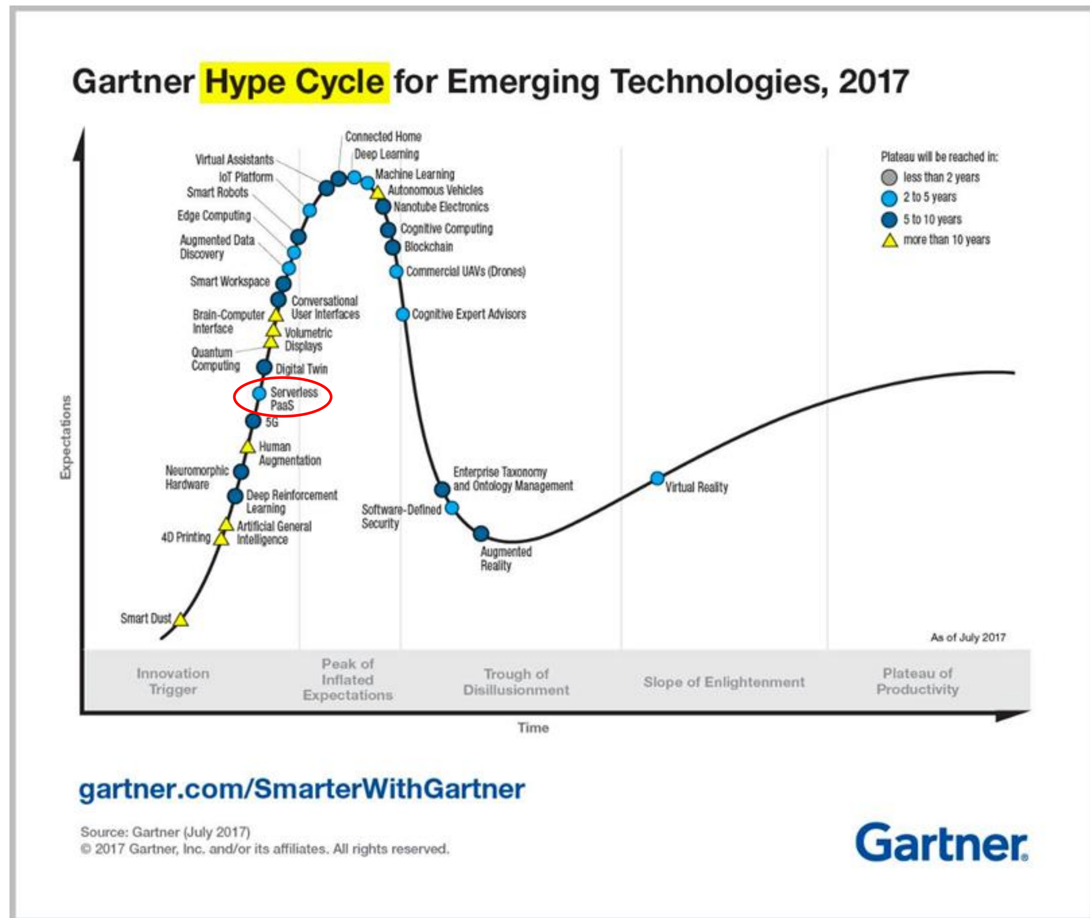


Figure 10: Gartner Hype Cycle for Emerging Technologies. Source: Gartner

2.2.1 Definition

Serverless Computing is an emerging paradigm for cloud computing, it is hard to find a universal definition. Due to the lack of terminology, Serverless, and FaaS are often used

[5] AWS Serverless https://aws.amazon.com/serverless/
[6] Microsoft Azure Serverless https://azure.microsoft.com/en-us/solutions/serverless
[7] Google Cloud https://cloud.google.com/serverless

interchangeably by the user community [6]. However, it is worth differentiating between FaaS, and Serverless Computing.

Serverless Computing is a broad notion refers to a cloud-computing execution model in which the application runs on servers that are fully managed by a third party [7]. In other words, Serverless Computing is a cloud computing model in which code is run as a service without the need to maintain or create the infrastructure. As a result, Serverless application can be considered as a cloud-native application, which is deployed on an abstracted infrastructure owned, and managed by cloud providers.

Serverless covers a range of techniques, and technologies including *Backend as a Service* (BaaS), and *Function as a Service* (FaaS) [8]. Backend as a Service is a cloud model in which developers outsource all the backend components of a web or mobile application. Typical BaaS Services are user authentication, database management, remote updating, and push notifications, as well as cloud storage, and hosting [9]. Function as a Service will be discussed thoroughly in section 2.3.2 as it will be used intensively in our application.

2.2.2 Serverless architecture

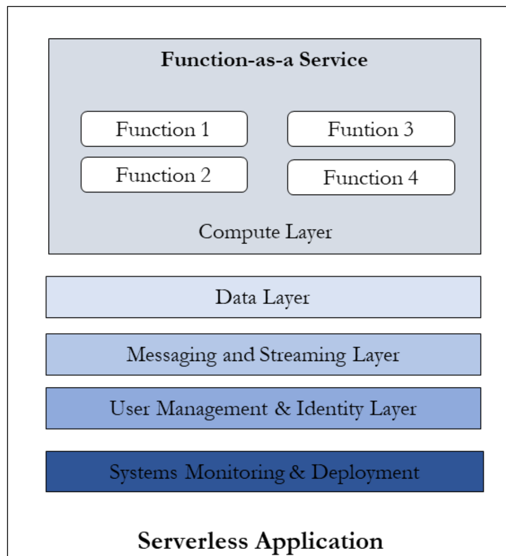


Figure 11: Serverless application layers

Serverless architecture relies completely on cloud technology. Following the white paper about AWS Well-Architected Framework published by AWS in 2018 [10] as shown in Figure 11, a serverless application can be decomposed into a five layers model to facilitate good design in the cloud including:

- Compute layer manages requests from external systems, controlling access, and ensuring requests are appropriately authorized. It contains the runtime environment that the business logic will be deployed and executed by
- Data Layer manages persistent storage from within a system. It provides a mechanism to store states that the business logic will need.
- Messaging, and Streaming Layer: The messaging layer manages communications between components. The streaming layer manages real-time analysis and processing of streaming data.

- User Management and Identity Layer provides identity, authentication, and authorization for both external, and internal customers of the application.
- System Monitoring and Deployment manages system visibility through metrics, and creates contextual awareness of how it operates, and behaves over time.

2.2.3 Serverless Application Programming Model

Besides the benefits of scalability without additional configuration. Serverless computing has effects not only on how applications are executed but also how they are developed.

Development, and Debugging

Local debugging is complicated. Because the application is expected to be executed in the cloud, and sometimes being triggered by other cloud services, it requires a lot of time to reproduce a production execution environment in local debugging. Serverless application can be debugged, and test locally by writing custom wrapper. Integration test can be done by adopting practices such as mock or stub test.

Infrastructure Management, and Deployment

The serverless application is not only about functions, it contains multiple resources such as database, queue service, log monitoring, and security credentials. AWS provides web console, AWS CLI, and AWS Serverless Application Model (AWS SAM) for managing serverless resources. AWS SAM is a good choice for managing multiple resources application since it is written over AWS CloudFormation, a native AWS tool for defined AWS Resources.

2.2.4 AWS Serverless Services in comparison

It is worth to get familiar with existing services before choosing the suitable one for development. The three primary computing services of AWS Serverless, namely AWS Batch, AWS EC2, and Lambda were taken into consideration for the deployment of the web crawler. Each service was experimented to gain insights about its strengths and drawbacks.

a. AWS Batch

AWS Batch [8] is a fully-managed service by AWS, which will provision the optimal quantity, and type of compute resources. AWS Batch has the following component. Job queue contains the jobs, which is an independent task to be executed with multiple inputs. These jobs are defined with a job definition, which is basically a Docker image encapsulating the business logic code. AWS Batch monitors the ongoing jobs, and job queues , and can auto-scale cluster capacity depending on workload. AWS Batch uses Elastic Container Service (ECS) for orchestrating Docker containers for running tasks. AWS does not charge an extra fee for the batch jobs, they only charge for the resources which Batch jobs are using such as EC2 Instance, S3 Storage. With AWS Batch, the crawler can be implemented as followed. The business logic of the crawler can be defined in a job definition, the seed URLs can be put in the job queues. Once started the job will receive one URL from the job queues and perform the crawling process.

Advantage

AWS Batch is efficient for provisioning the optimal used of computing resources that are required for performing the batch jobs. Because it is fully integrated with AWS Platform so it utilized the networking, scaling of the AWS.

Disadvantages

As each Batch job is a containerized application, Batch expects the Docker skill from the developer. The lack of experience with Docker might lead to some hidden problem. In addition, it is worth to mention that the user interface is confusing for developers who are not familiar with AWS. Because it is less intuitive when compared to other AWS Services. One important point is that AWS Batch doesn't have a job history that supports monitoring, which makes it extremely difficult to monitor its behavior. Furthermore, Batch is a less popular services of AWS. The development with AWS Batch is cumbersome with little support from the community. Last but not least, there are less documentation about the operational behavior of Batch, which might be hard for troubleshooting, and fixing the application.

b. AWS EC2

AWS EC2[9] is a virtual cloud infrastructure service, which provides on-demand computing resources (instances) to create powerful servers in the cloud. AWS EC2 instances can be scaled in terms of processing power, and computing memory.

[8] https://aws.amazon.com/batch/

[9] https://aws.amazon.com/ec2/

Advantages

The most significant advantage of EC2 is its ability to scale horizontally with increasing workload. It is what makes EC2 very attractive services for hosting provider. The instant setup of newer server instances in minutes with a click of a button allows.

Disadvantages

Although the setting up for a single EC2 machine can be easy, being able to scale up EC2 is not as much convenient. In order to scale out the EC2 Instances in a cost-efficient manner requires predicting the incoming workload. Besides, the cost model of AWS EC2 is also not very flexible. The user has a large upfront payment even if they do not fully use the instances in the purchase length. It is also hard to decide which instance types is suitable, it is sometimes forced to get bigger instances only more CPU or RAM are needed. In general, the entire configuration, and setup process demands comprehensive technical knowledge, and requires proper training. The learning curve of EC2 can be steep and can take some time to be fully familiar with this service.

c. AWS Lambda in conjunction with SQS

AWS Lambda [10] is a Serverless FaaS runs backend code without configuring and managing a platform or infrastructure.

Advantages

The basic advantage of Lambda is that the user only pays for the time the function running, and the resources it need to execute. AWS Lambda functions are billed by the millisecond of CPU time. The main distinguishing feature of Lambda rapid development. AWS Lambda enables faster prototype, and the developer must spend less time on operational issue. Furthermore, the building, and deploying of Lambda is straightforward, and is simple when compared to deploying an entire server. The developers can even write the code directly in the AWS Lambda Console. It is a good starting point for developer with less or no experience with developing in AWS. Since it is easier for developer to concentrate on the application logic.

Disadvantage

The lack of control over environment should be considered before using Lambda, the developers are not able to custom install packages or software on the running

[10] https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html

environment. AWS Lambda functions also suffered from cold start, the delay by the first function invocation or invocation after a long idle time. The debugging and testing of Lambda can be troublesome but can be alleviated by using some framework.

d. Conclusion

Each of the services mentioned above has a niche where its suits best. AWS Batch is a new computing platform, which can have a lot of potentials but need to be study careful before usage. EC2 in conjunction with Elastic Load Balancing is also excellent for web hosting. For example, if EC2 is used to host a webpage, the computing resources being used is constant. However, the cost to use EC2 is unpredictable since we cannot calculate exactly the crawling workload.

Based on the weakness, and the strength of each service, AWS Lambda in conjunction with AWS SQS was chosen for the deployment of the web crawler, because it is easy to get familiarized with and quick to develop a first prototype. Furthermore, the usage of AWS Lambda has raised a lot of interest at the company since it is a most popular Function as a Service, an emerging cloud computing service.

2.3 Function as a Service

Function as a service (FaaS) is a new service offered by cloud providers. This is supported by the fact that the "cloud function" products such as AWS Lambda[11], Azure Functions[12], and Google Cloud Functions[13]. The idea behind FaaS is that the developers can develop, execute, and manage their own code without having to handle the under lying complexity. The developer only needs to write his function code and define the function with an event. The occurrence of this event will trigger the execution of the function, as known as invocation. After the function finishes an execution it will be immediately terminated.

This makes FaaS cost-effective in comparison with other computing models, because the user is only charged for the time a function is running. Launching a web server on a virtual machine, for examples Amazon EC2, will be charged per hour, even if there are no requests come in. The second benefit of FaaS is auto-scaling is handled completely by the cloud providers. This will allow the application to be more responsive since it is designed to be scaled at any unpredictable workload. In general, a serverless might be a good choice

[11] https://aws.amazon.com/lambda/
[12] https://azure.microsoft.com/en-us/services/functions/
[13] https://cloud.google.com/functions/

for a workload that is asynchronous, easy to parallelize into independent units of work. It can also good at handling infrequent or sporadic dem, and.

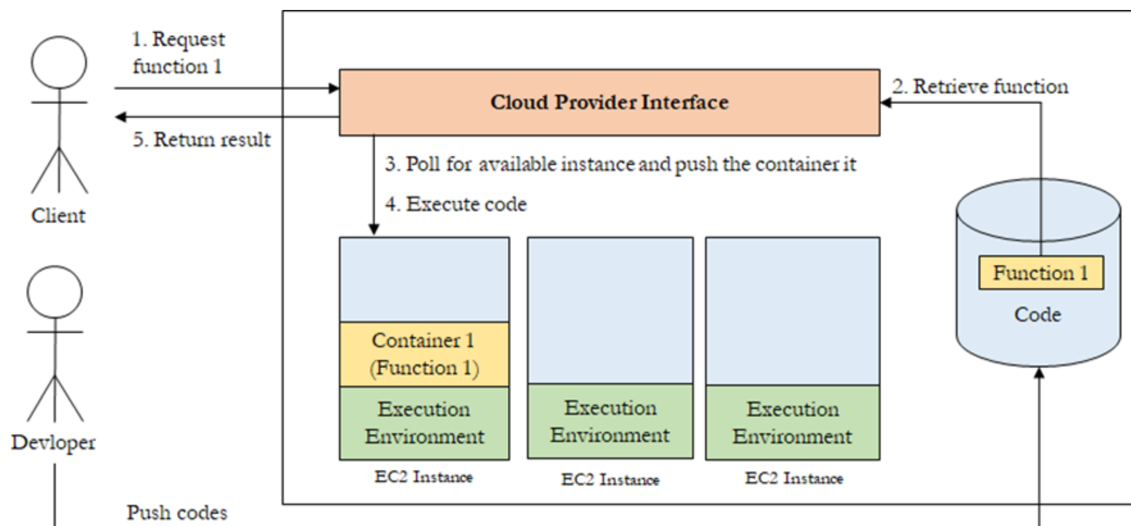


Figure 12: FaaS Processing Model

The execution model behind FaaS is described in Figure 12. Whenever there is a new incoming request through the API, an available instance within the pool is used to handle that request. The corresponding code of the requested will be retrieved from a code repository and pushed into a container. The container is then pushed to the chosen instance and executed. The result is then returned to the client, who sent the request through the cloud provider interface (API). The code is maintained and pushed into code repository by the developer.

The "cold start" problem arises from this execution model in other word the first execution of function always takes longer time. Because the container has to be spun up, and the code has to be loaded into the container. The next execution won't take as much time as the first one since the container is already "warm". However, after a long period of idle time, the container will be terminated, and the delay problem will happen again.

Conclusion

In conclusion, serverless computing in conjunction with Function as a Service leveraged with task queueing for batch processing is a promising technology to implement batch processing on premise. However, there are still open questions related to this architecture, namely:

- How to design a serverless architecture?

- How to meaningfully implement the business logic with the lambda function on cloud platforms?
- How to deploy a serverless application fast on cloud platforms?

3 Design and Conception

This chapter will present the design of the crawler. Chapter 3 will be organized as following: Section 3.1 will present the requirements. Section 3.2 will give an overview of the system's architecture and explain the role of each component. Section 3.3 will propose a deployment pipeline for the applications.

3.1 Requirements

The requirement analysis aims to identify the most important specifications for the web crawler and to consider the feasibility of them. The requirement analysis was done as follows. At the beginning stage of the project, the product owner proposed the initial ideas of the project in the kick-off meeting. The software requirements analysis is conducted independently based on the product owner's ideas. Although the requirement analysis and architecture choice were done, they are not completely fixed. During the development, the product owner, and the team Big Data were involved all the time and introduced additional requirements.

These are the requirements from both functional, and non-functional perspectives:

- The application must be deployed and run completely on the cloud environment to avoid IP being blocked.
- The crawler must behave politely and following the robot.txt rules.
- The results of the crawling process must be according to the use case:
  - Unstructured Data must be stored in the S3 object store.
  - Metadata, logging data must be stored in a relational database.
- The crawler should be designed and implemented generically in term of the monitoring of crawling jobs.
- The interface for monitoring crawling jobs has the following specification:

  - Enables the monitoring following metrics: the number of successful, deferred, and failed jobs. Deferred jobs are jobs that have a temporary error and needs to be relaunched.
  - Allows users to keep track of the crawling history: last crawling history, crawling schedule plan.
  - References to the cost of the crawler history.
- The programming language of choice should be Python. Furthermore, the crawler should be easily written and debugged locally.

3.2 Crawler design

The following section is organized as follows. In the first subsection, a general workflow of a web crawler is introduced, which serves as a starting point for the choice of architecture type. After that, a suitable, and high-level architecture approach is chosen based on the workflow analysis. The last section will describe how the crawler is designed in detail by explaining each of its components in term of business logic.

General Workflow

Figure 13 presents the general workflow of the crawling process in favor of a focused web crawler. It is obligatory to implement a focused web crawler, as the crawler is expected to only retrieve the official journals, and no other types of documents on a webpage in order to avoid contaminating the database with irrelevant data. The high accuracy of the PDFs discovered will help to reduce the effort spent in re-filtering the PDFs in the next phase of the "Baukarte" project, which involves in data mining, and text extraction.

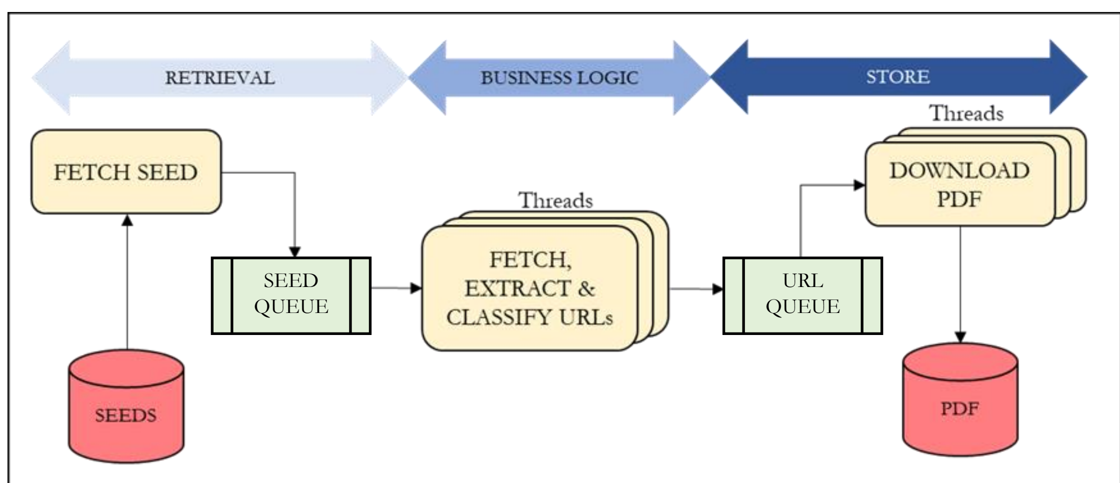


Figure 13: Workflow of the web crawler

The workflow consists of three main steps. First, the input retrieval part fetches the data from a data source, in this case, the seed URLs from the database. Secondly, the business logic processing part applies a set of predefined business logic to each item (URL). Examples can be extracting hyperlink from the HTML markup and classifying them. If the URL is predicted to be on-topic, then its links are extracted and are appended into the URLs. Thirdly, a worker will fetch the URL links, and download them into persistent storage. The workflow can be periodically repeated maybe once or twice a week in order to keep the PDF collection fresh.

Distributed message queue as a solution

Based on the workflow in Figure 13, it is easy to recognize that the crawling process has the characteristics of batch processing. Firstly, the crawling process only needs to be executed periodically. Secondly, the crawling processing can be fully automated, and therefore no human interaction is needed.

According to Burns [11], in order to reduce the processing time, a simple technique can be applied such as message queue. Figure 14 shows the idea behind the work queue system, which can be divided into four components: producer, task queue, queue manager, and a pool of workers. The producer pushes messages into the queue. Each message contains a task. Each task might be a small unit of data needed to be processed in the same way. The queue manager has the responsibility to distribute the tasks evenly to the workers. It will poll, and assign the tasks to any worker, which has the free executing capability. The random distribution is allowed, because the tasks are independent of each other, and do not need to be processed sequentially. The usage of a message queue for batch processing has three advantages:

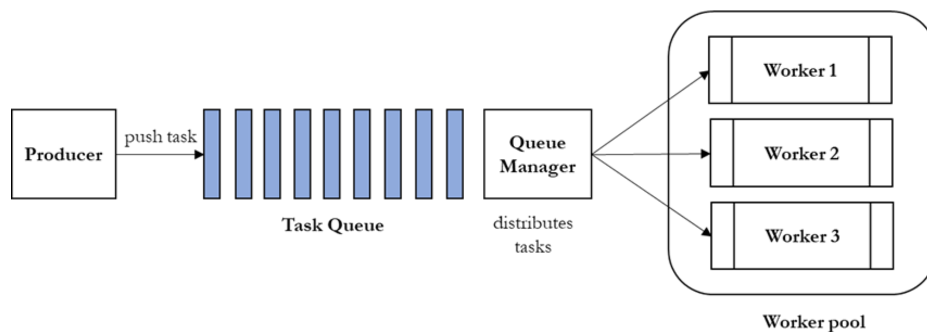


Figure 14: A generic work queue. Adapted: Burns *[11]*

Firstly, the message queue serves as a buffer between the producer and the worker, which increases the efficiency of the producer. For examples, if the producer, and the worker have different processing rates. The producer can still work asynchronously from the

worker and do not have to wait until the worker finishes the previous task to process a new task.

Secondly, the workers can be scaled up or scaled down to ensure that the work can be handled within a certain amount of time. This increase the parallelism of the web crawler.

Lastly, the message queue system also shows high compatibility to the deployment model (AWS Lambda). It allows the application to be designed following the best practices recommended by AWS [10]: singularity, scalability, and stateless. The Lambda function has a single purpose and has a concise business logic. Lambda function can be scaled up, and down based on the workload. After each execution, Lambda passes the output to another queue or other functions and terminates, which makes it completely stateless.

Based on the huge advantages of message queue shown above, it is the best candidate for the crawler architecture and is chosen to implement the web crawler. Figure 15 shows the components of the web crawler including:

- Initializer Module triggers the crawling task so that it will be started periodically without any user's interaction.
- Website Watcher Module monitors website-contents changes and informs the Hyperlink Collector Module about websites that should be crawled or re-crawled.

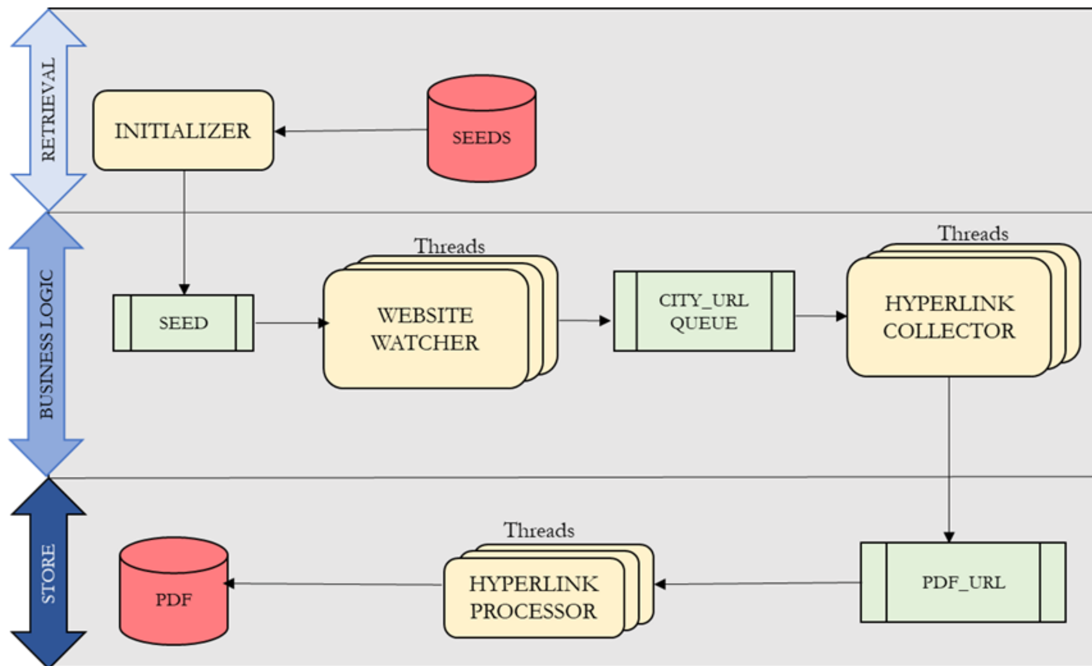


Figure 15: Components of web crawler

- Hyperlink Collector Module is in charge of parsing websites HTML markup, identifying links for PDF file, retrieving the hyperlinks, indexing, and storing eligible links in the database.

- Hyperlink Processor Module is responsible for downloading the PDF files, indexing, and storing the file in the AWS S3 Object Store with their IDs.

3.2.1 Initializer Module

The crawler is initiated by the Initializer, which executes on schedule. Once started, it will open a connection to the database read all the URLs of all city and put them into the seed queue.

3.2.2 Website Watcher Module

Function

Website Watcher Module is the component that implements its business logic. It has the responsibilities to detect the content changes in the webpage and assigns the website with changes detected to further steps. The purpose of it is to reduce the computational resources in incremental crawling round after the initial crawling round. The business logic discussed above can be visualized in Figure 16.

Initial idea

The ideal practice to identify webpage content change is to check the Last-Modified field. A website might have a Last-Modified field in their HTTP header that contains that date, and the time at which the origin server believes the resource was last modified. This content changes would signal that a new document is uploaded in the main page and added to the lists of building permissions. An example of the Last-Modified field is:

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

Problem

However, an empirical study in the set of experiment 50 webpages gave a surprising result that the Last-Modified field is not always available in the HTTP header of in our experimental web pages. Because there is no general mechanism of updates, and notifications, the initial idea cannot be attained in practice.

Solution

A simple workaround solution to this problem can be as follows. The Website Watcher Module detects changes in a web page's content by storing a copy the HTML of a webpage, and then periodically getting the current HTML, and checking it against the initial HTML. In order to save disk memory, the raw HTML markup will not be stored directly in the database. The raw content must be first filtered to remove JavaScript elements, and then the HTML tags are also removed. The filtered HTML markup is then

hashed and stored in the database. If the webpage content has recently changed, it will have a different HTML hash value to its previous hash value.

Trade-off

At first, the initial idea was intended to save computational resources, as the resource needed to check an HTTP header of a webpage is relatively low. Indeed, the solution proposed above needs a higher resource than the initial idea. Because it must do the extra steps to parse the HTML markup and create a hash value. However, it is still acceptable compared with not having the Website Watcher module. Because parsing and hashing HTML markup is still cheaper than having to parse HTML code, check up every hyperlink found in the main page to decide whether they have already existed or not.

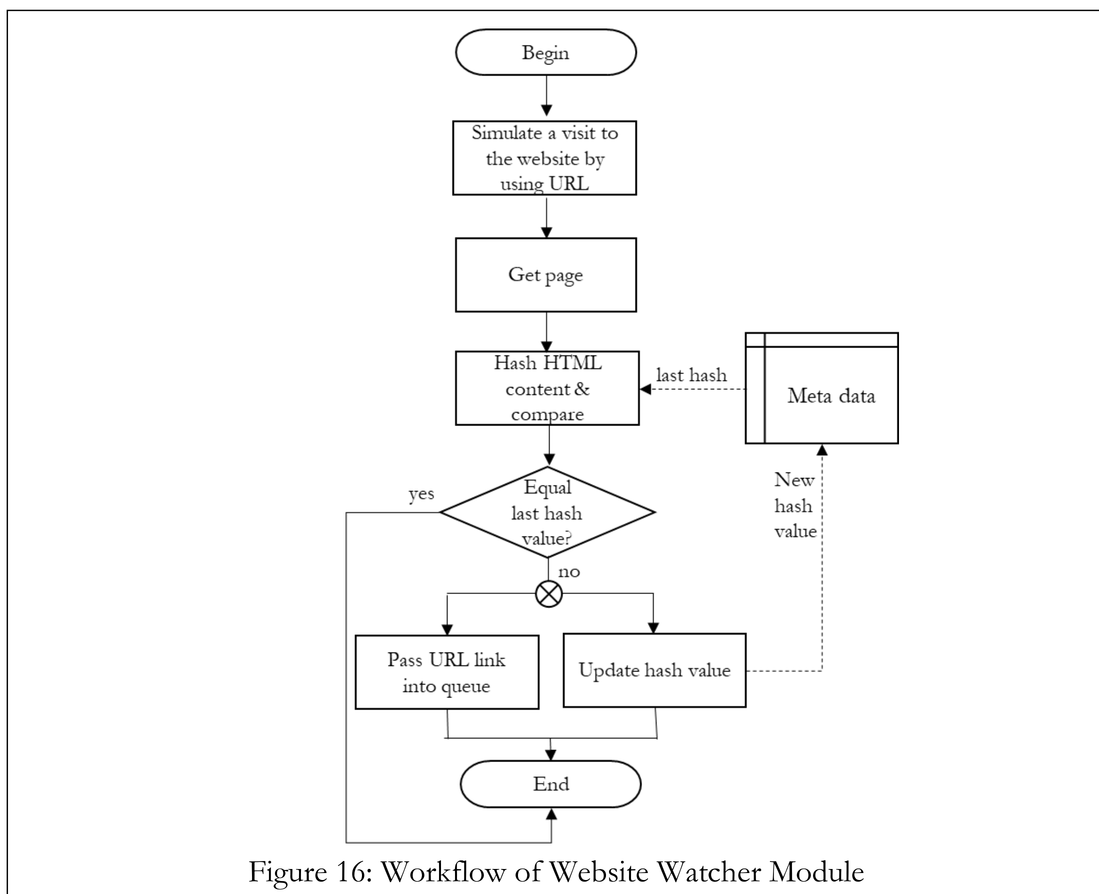


Figure 16: Workflow of Website Watcher Module

3.2.3 Hyperlink Collector

Function

The Hyperlink Collector Module is responsible for parsing and extracting hyperlinks from the Webpage. It reads the messages in the queue sent from the previous module and

processes them individually. The message from the previous module contains the name, ID of the cities need to be crawled.

Solution

Each message received will be processed with the workflow shown in Figure *17*. Firstly, a GET request is sent to the server to retrieve the HTML content of the webpage. This HTML markup is then searched for <a> tags, which specifies the hyperlinks in a HTML content. This was done by parsing the HTML to a parse tree using a Python Module. With this parse tree, all the tags that are <a> can be extracted. The second step is to canonize the relative links. For example, such a relative URL

`<a href="page2.html">Next page</a>` -

will be turned into the absolute link:

`<a href="https://www.yourwebsitename.com/pages/page2.html">Next page</a>` In the relative link, the attribute href only point a directory relative to the webpage, where the hyperlink locates.

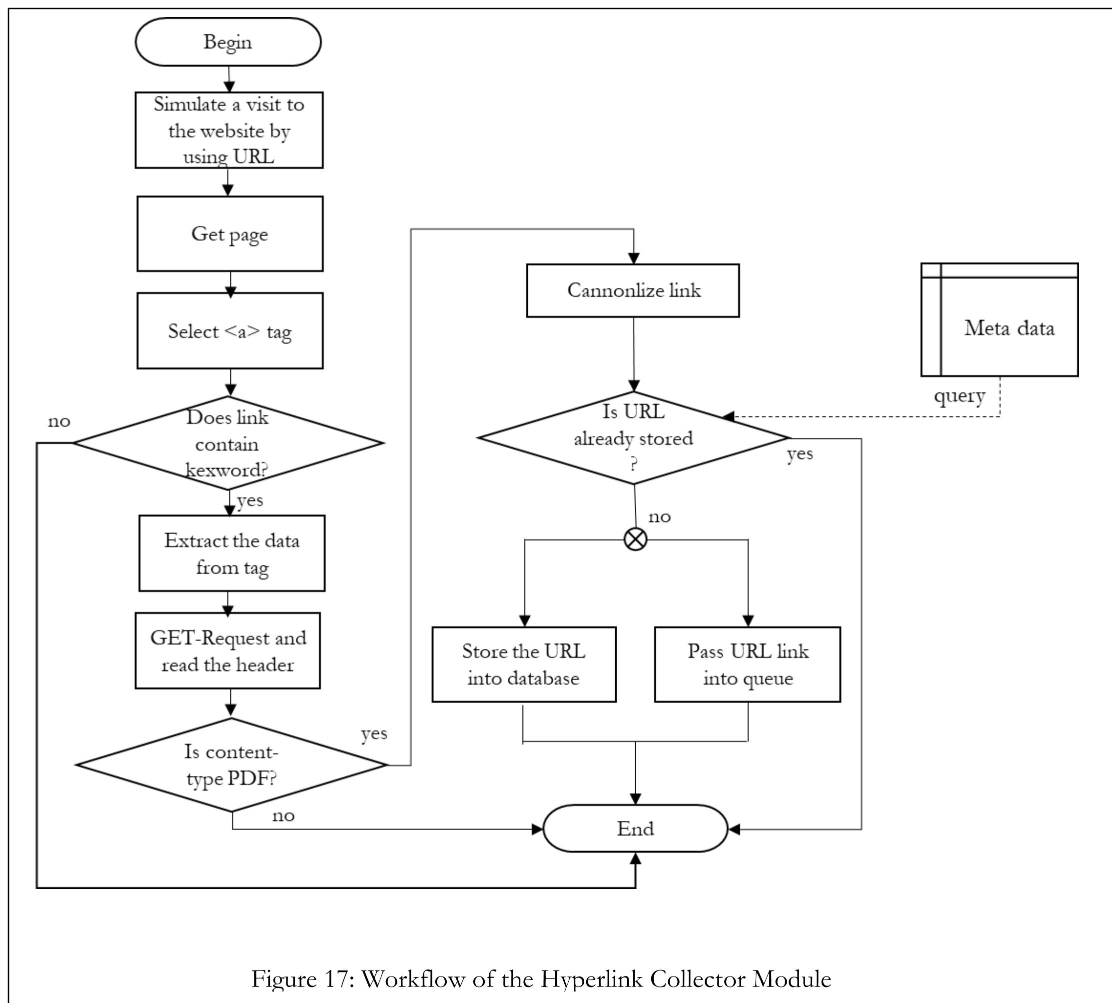


Figure 17: Workflow of the Hyperlink Collector Module

After being canonized, the href link is then added with a protocol, and domain name so that the crawler can download the document later. Because it is straightforward to search

a PDF embedded in hyperlinks, but hyperlinks extracted from webpages must be processed and filtered in numbers of ways before being thrown back into the work pool. For example, the <a> tags can also contain advertisements links, and navigations links, which are irrelevant. Filtering out documents that are probably not official announcement before retrieving it into our database reduces the consumed bandwidth and helps the crawler to be domain specific. The attributes such as text inside the tag is then compared with a predefined set of rules. If the text inside the tag satisfies all these rules, it will be passed to the next step, or else, will be filtered out, and marked as invalid. Next, the MIME type of the URL will be checked to be PDF. If the MIME type proves to be anything else than PDF, it will be ignored.
Next, the URL will be checked if they are already stored in the database. This feature avoids redundancy of URLs. It makes sure that the URL in the database is unique. Only URL links that fulfill all these conditions are stored in the database. At last, they will be passed into another queue that serves as the event source to trigger the next module – Hyperlink Processor.

3.2.4 Hyperlink Processor Module

Function

Hyperlink Processor is the components to download the PDF file. The advantages of separating the download task from the HTML parsing task in the Hyperlink Collector Module is that the crawler runs much faster and requires less bandwidth. This also reduces the time of the session. Therefore, the crawler will be less likely to be blocked from performing too many requests in short amount time.

Due to the requirements of the "Baukarte" project, the PDF files have to be stored by reference and downloaded. The project involves the process of text mining that is shown in the project concept in Figure 1. Therefore, the PDFs file have to be persistently stored in our database for further analysis. Moreover, the act of only storing PDF by reference (storing the URL where the file is located at) has a risk as follows. The file hosted at any particular URL is subject to change. This might lead to unexpected effects. For example, if the PDF content is migrated to another location, the URL links might eventually go missing or is changed to something completely irrelevant.

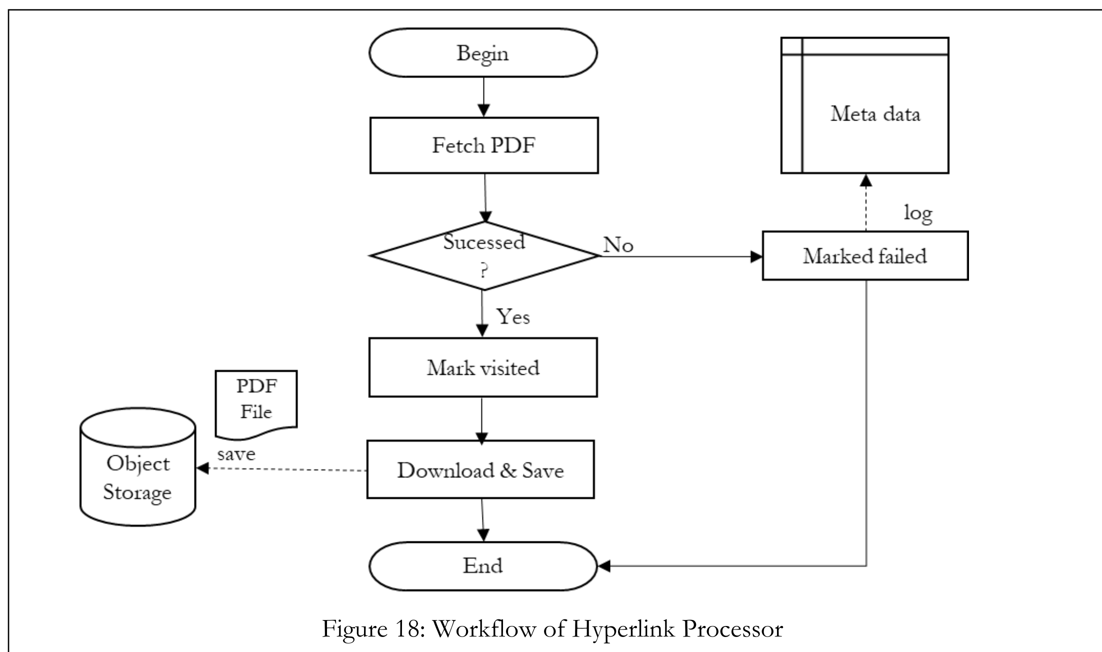


Figure 18: Workflow of Hyperlink Processor

3.2.5 Data Storage

In order to make the web scrapers useful, the ability to store and interact with large amounts of data is incredibly important. There are two main types of data to store: metadata and actual PDF documents:

Meta data The meta data stored in this project can be divided into two categories: input for the crawling process, and the output of the crawling process. The input is the list of URL seeds. The outputs are the history from the crawling process, indexes of documents we already have in your S3 bucket. The history of each PDF link crawled has to be stored to make sure that the crawler only downloads the same PDF once. It is practical to keep the information in a database. The information has to be queryable so that the links which have not been downloaded or failed to be downloaded can be found later on. The database must also handle a lot of insert operation from the crawler.

PDF Documents is the official journal found. The PDF documents need to be indexed in a way that allows them to be queryable.

3.3 Serverless deployment development

The last step in developing a web crawler is to be able to deploy it into the cloud. The deployment of a serverless application differs a lot from a self-hosted application's deployment. It is an interesting object to examine in order to gain more insights about the deployment process of a serverless application, which lack both academic resources, and established practical patterns.

3.3.1 Purpose

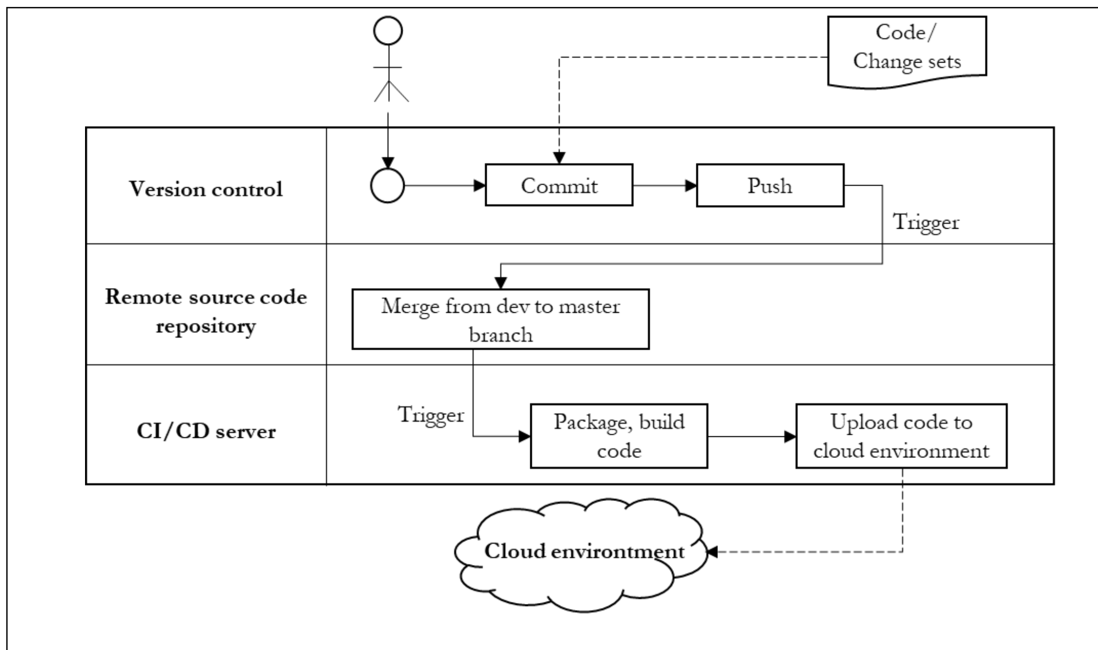
The term application deployment can have a lot of different meanings depending on the type of the application. In general, it is the process of release the application from the development environment to the production environment. In case of a serverless application, deployment means moving the code from a developer's repository to a repository managed by the cloud provider. The source code of a serverless application consists of two components: the business logic that defines the behavior of the application, and the resource's configuration code that defines the properties of the resources used by the applications such as database, message server, user authentication server, etc.

3.3.2 Goal

This is the main goal to design a simple, and automated integration, and deployment pattern both the infrastructure and the actual code are deployed. The automated integration process should support the early merge of code changes into the master branch. The automated deployment would minimize the manual errors, and reduce the time needed.

3.3.3 Concept

First, the common repository should be managed by a version control software. The source code contains both code for the business logic and the resource's configurations of the serverless application. In the build, and deployment stage, a continuous integration server should support the developer team with the ability to build and deploy to a cloud environment.



Developers create an isolated branch to work on a new feature. This branch can be merged into the develop branch for review. After tested, and reviewed the code be pushed to master branch, where it will trigger the build, and deployment process. This process is defined in a configuration file conforming to the requirement of the CI platform.
Two separate deployment pipelines are needed: one to deploy the AWS resource's configuration code, and one to deploy business logic's code. This choice was made based on the following reasons. First, the configuration files of the cloud resources need to be stored securely. As in this project, the cloud resource's configuration file will be kept in an AWS S3 bucket. This method of remote resource storing is safer than storing locally. Second, once the project is deployed for the first time, it will come into the state of code iteration, and redeployment. In short, the infrastructure will not need to be changed as often as the Lambda code and should not be coupled in the deployment of AWS Lambda code.

4 Implementation

This chapter describes the implementation based on the proposed architecture in the previous chapter. Section 4.1 presents the overall view of development environment and technologies, namely frameworks and libraries. Section 4.2 gives details about the implementation of the core functions as well as the adopted AWS Services. Section 4.3

4.1 Development environment and Frameworks

Development environment

The Python in this project runs under Python 3.6. The application was developed simultaneously under both Ubuntu 16.04 LTS, and Windows 10 Operating System.

Within the scope of a bachelor thesis, it is practical to use as many as possible "ready-to-use" technologies, in order to create a prototype fast and test it at a small scale to validate the initial architecture proposal. Therefore, the following frameworks and library are extensively used in the implementation:

BeautifulSoup[14] is a library that makes it easy to scrape information from web pages. It sits on top of an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree. The official docs are comprehensive, easy to read and provided with practical examples. Beautiful Soup comes with Python 2, and Python 3.

Requests[15] library is the de facto standard for making HTTP requests in Python. Every request raised from a web client takes the advantage of Request to communicate with the server using any one of the HTTP methods i.e., HTPP GET or HTTP POST. It abstracts the complexities of making requests behind a beautiful, simple API so that the users can focus on interacting with services and consuming data in the application.

Boto3[16] is the Amazon Web Services SDK for Python. It enables Python developers to create, configure, and manage almost all AWS resources from the Python script. Boto3 provides an easy to use, object-oriented API, as well as low-level access to AWS services.

[14] https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[15] https://2.python-requests.org/en/master/
[16] https://boto3.amazonaws.com/v1/documentation/api/latest/index.html?id=docs_gateway

The code example below how to get the lists of all buckets available in the AWS S3 in three lines of code.

```
import boto3

sqs = boto3.resource('s3')

bucket = conn.get_bucket('bucket_name')
```

4.2 Implementation of the core functions

In section 3.2, the main modules of the crawler are explained in term of function. This architecture needs to be evaluated by the feasibility, and usability. Therefore, the architecture was built into a prototype as a proof-of-concept with AWS Services. In addition, a brief introduction about the critical AWS Services adopted in the implementation are presented, namely AWS Lambda, SQS, CloudWatch and S3, because they are the key building blocks of the crawler.

One of the requirements for the implementation was the complete usage of cloud services, which can be traced back to the requirements in section 3.1. The decision was made to use build the application completely with Amazon Web Services from the business logic to database. As being one of the most popular cloud providers, Amazon Web Services offers a cloud ecosystem including compute, storage, databased, developer tools, and management tools, which can cover the complete application development life cycle.

4.2.1 Initializer Module

Figure 19 shows the components of the Initializer Module. AWS CloudWatch has the ability to set up monitoring, which enables the trigger to get triggered every few minutes or hours. In particular, Lambda functions can be triggered as a cron via AWS CloudWatch, by setting the frequency of the cron by entering the interval via the cron expression. The Amazon CloudWatch Event is the agent to schedule automated actions that are self-triggered periodically. This event will trigger the Lambda function to retrieve all cities (URLs of official website) stored in the DynamoDB seeds table and put them into the SQS queue.

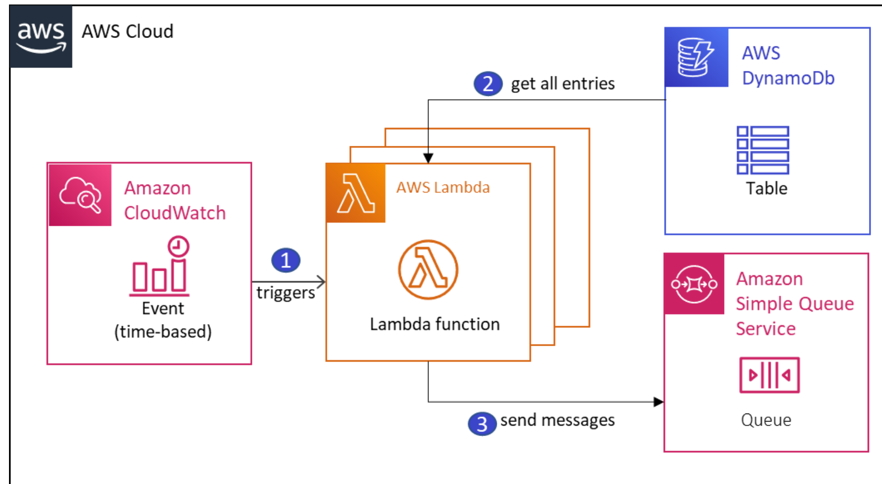


Figure 19: Initializer Module implemented with AWS Services

4.2.2 Website Watcher Module

Figure 20 describes how the Website Watcher Module is implemented. A Lambda function is chained to the SQS of the Initializer Module as a worker. When messages are pushed into the queue, the queue automatically fires up multiples Lambda Function to process these messages. Further business logic carried out by Lambda Function includes retrieving webpage content (HTML) and comparing with the previous hash value stored in DynamoDB. The webpage that has changed recently will provide a different hash value to its previous hash value. This will be the factor to decide whether a city (URL) should be added to the to-be-crawled queue or not. The new hash value is stored in persistent storage.

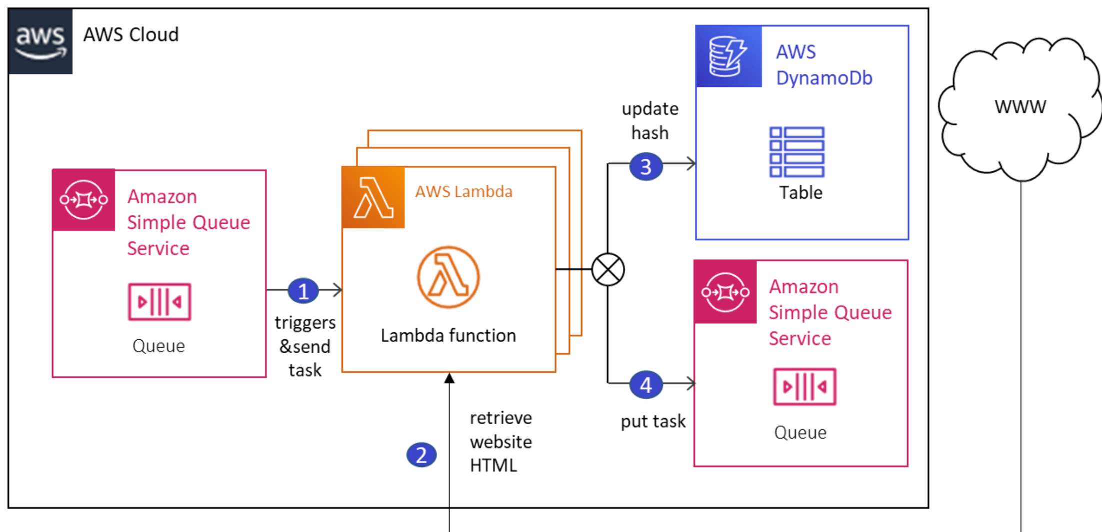


Figure 20:Website Watcher Module implemented with AWS Services

AWS DynamoDB is used to store the latest hash value of the webpage along with the previous hash value. Lastly, the URL of the web page with content changes is sent to the next SQS queue to be crawled.

4.2.3 Hyperlink Collector Module

SQS queue from the Website Watcher Module acts as an event source that triggered the Lambda function. AWS Lambda Function executes its code to perform the crawling process on the pages (URL) it received from the queue. Lambda parses the HTML content of the web pages and extracts all hyperlinks. Each link has to fulfill a set of rules to be considered as eligible URL (official announcement). Next, each hyperlink is classified as new or already existed by querying the DynamoDB table. The links classified as not already existed will be indexed with a unique identifier and stored in the DynamoDB table. They will also be sent to AWS SQS Queue as tasks for the next module.

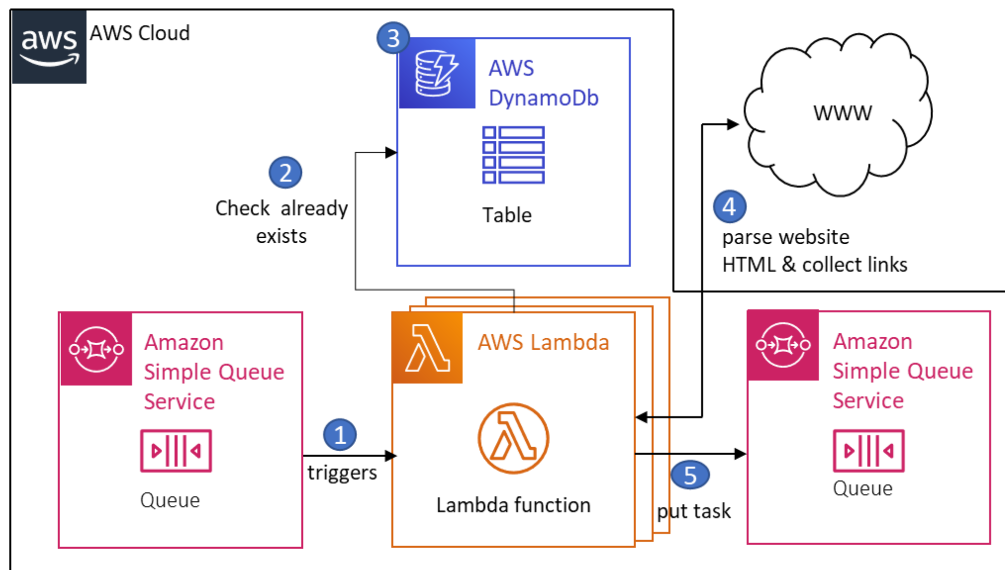


Figure 21.Hyperlink Collector Module implemented with AWS Services

4.2.4 Hyperlink Processor Module

The last step of the crawling process is performed by the Hyperlink Processor Module. AWS Lambda Function opens, and reads the message containing the URL to directly into the S3 buckets with the unique identifier created in the previous module. The file is then stored in a bucket AWS S3 Amazon Simple Storage. This ensures that each document will be saved both as referenced in the AWS DynamoDB table, and stored persistently in S3 bucket with the same identification.

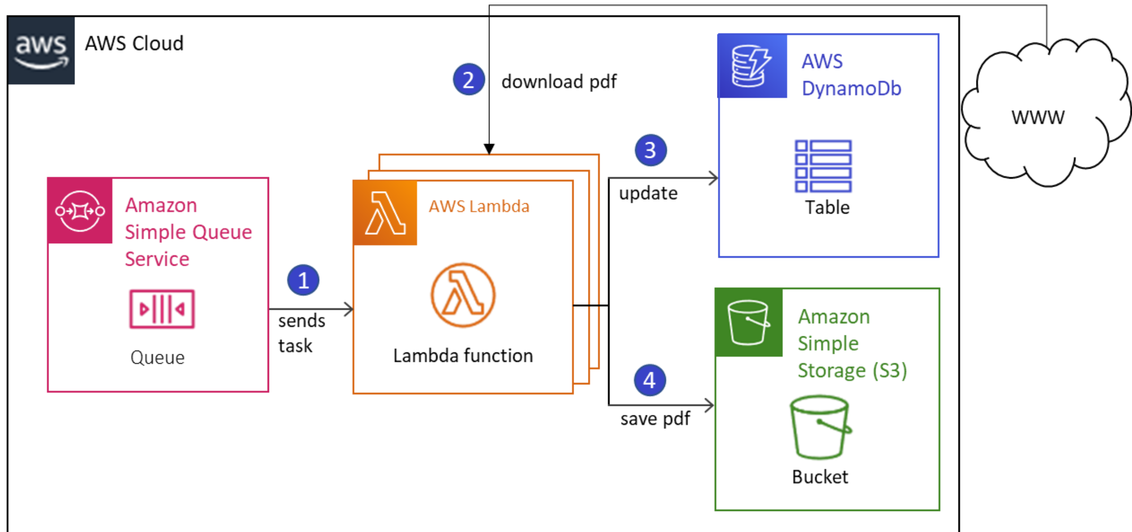


Figure 22: Hyperlink Processor Module implemented with AWS Services

4.2.5 Job Queue

A pattern can easily be noticed in the four modules above is the use of AWS SQS to trigger AWS Lambda function. This is a newly added feature of AWS Lambda, which is used to improve the parallelism of the crawler. In June 2018, AWS announced that AWS SQS is added as an event source of Lambda [12]. The SQS trigger can be added via AWS Lambda Console as shown in Figure 23. Whenever a new message arrives in the queue, the queue will automatically trigger Lambda function.

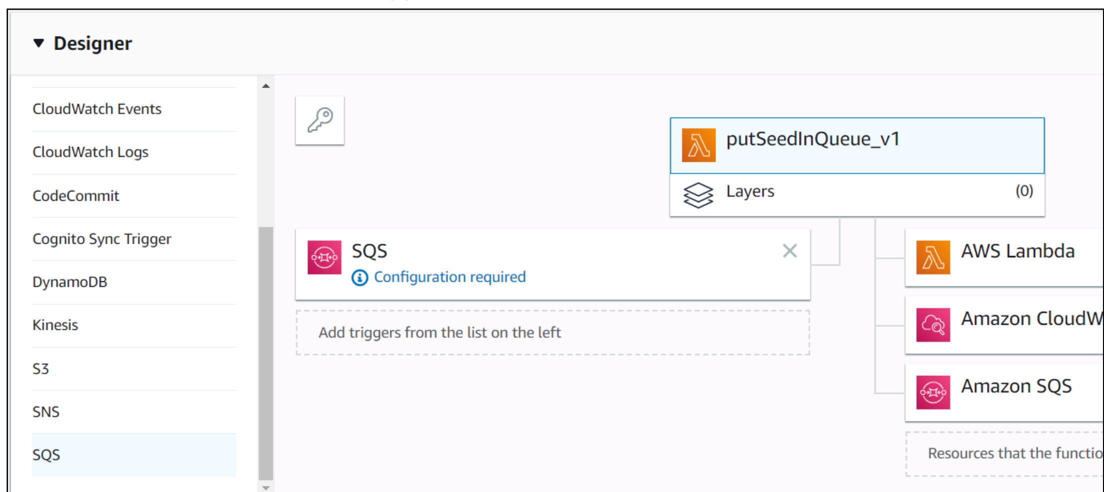


Figure 23: SQS as Lambda's trigger on the left hand-side

Advantages

First, and foremost, this design shows a great potential of serverless application. As AWS will handle all tasks that have to be implemented manually in Lambda function before. Traditionally, Lambda function has to poll the messages from the queue, wait for the

messages to arrive, process them, and delete them from the queue. With a queue acting as the broker between data producer, and worker (the durability of the process is on a reasonably good level. If the worker dies, SQS will hold onto our data, ensuring that it is possible to reach to it, whenever the workers are back to full health. The other benefit is also the scalability without additional configuration. AWS will take the responsibility of scaling up or down the numbers of Lambda functions adapted to the numbers of messages in the queue, and the predefined limit [13].

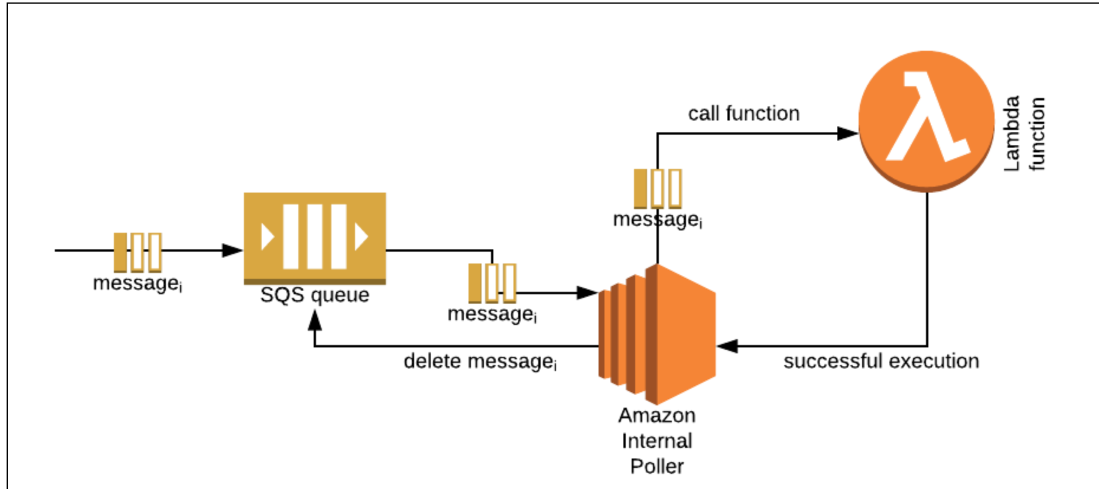


Figure 24: AWS SQS as trigger for Lambda function

The table below shows the list of queues used in our

Nr	Configuration	
1	Name	seed
	Queue type	Standard
	Publisher	putseedinQueue
	Consumer	WebsiteWatcher
2	Name	cityURLqueue
	Queue type	Standard
	Publisher	Website Watcher
	Consumer	Hyperlink Collector
3	Name	pdfURLqueue
	Queue type	Standard
	Publisher	Hyperlink Collector
	Consumer	Hyperlink Processor

Table 1: SQS Queue Configuration

f. Data Storage

The metadata is organized as a JSON tree and structured in a flattened form. For

The AWS S3 was implemented to hold with three buckets. Bucket iw_bd_demowebcrawler_lambda stores the AWS Lambda deployment packages. Bucket aws_bd_demowebcrawler_pdf bucket contains the downloaded PDFs file. Bucket aws_bd_demowebcrawler_tfstate is used to store the tfstate files, which are the configuration files for AWS resources.

4.2.6 Logs, logging management and billing.

Logs, and logging management are organized with Amazon Cloud Watch. Cloud Watch gathers detailed information relating to operations of Lambda functions. It also provides useful metrics, configurable alarm setting to notify stakeholders whenever failing, and various filters to quickly search for correct log

The statistics are collected on each run of the crawler. AWS Lambda automatically monitors Lambda function, reporting metrics through Amazon CloudWatch [14]. Lambda automatically integrates with CloudWatch Logs, and pushes all logs from your code to a CloudWatch Logs group associated with a Lambda function, which is named /aws/lambda/<function name> [15].

AWS CloudWatch Dashboard was chosen to visualize these metrics. Each metric mentioned above is shown on an individual chart. The data for the charts was sent directly from the logs of functions. The UI of the dashboard can be found in Appendix I. The dashboard was manually created by choosing the most important metrics of an AWS Lambda function. This dashboard allows the user to monitor the health of the crawler. The most important metric is the Error, and Success rate.

AWS using tags to enable the users to categorize AWS Resource and also for AWS billing reports. Tags were used with each of the web crawler resources to allow reporting using AWS Console and billing reports

4.2.7 Adopted AWS Services

This section provides a brief introduction about the important AWS Services used in the main functions, namely AWS Lambda, AWS Simple Queue Service, AWS Simple Storage S3, AWS DynamoDB, and AWS CloudWatch.

AWS Lambda Function

AWS Lambda[17] lets us run the code without provisioning or managing the servers. It lets us execute the code from few requests to even thousands in a short time, and scales up automatically as per dem, and. We need to pay as per use and need not pay when the code is not running.

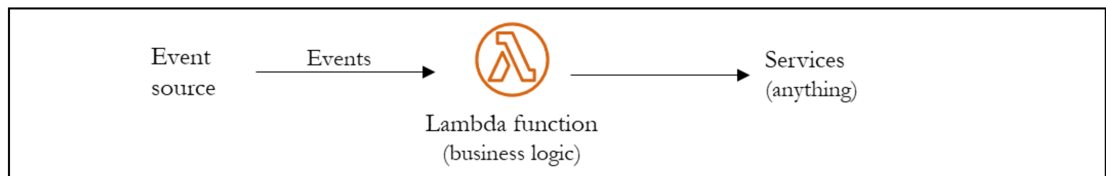
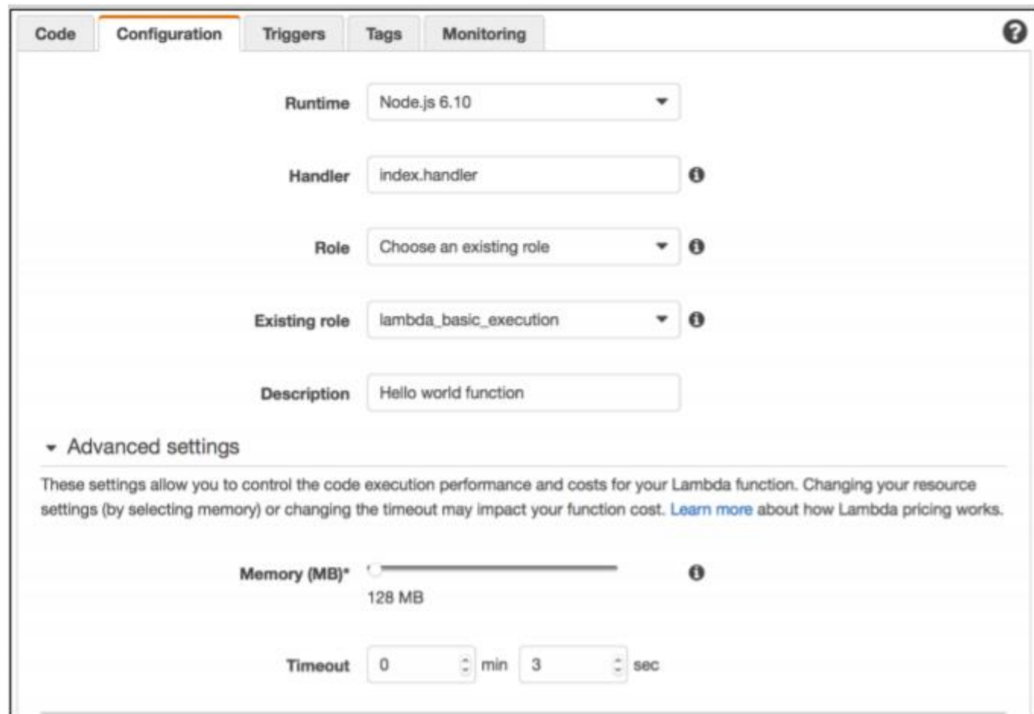


Figure 25: Simplify architecture of a running Lambda function

Configuration

The developers can set up their own configurations including Runtime Environments, Handler (name of the handler function), Role (IAM Role), and the Timeout.



[17] https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

Figure 26: Configurations of AWS Lambda Function

a. Runtime – Execution Environment

All the developer needs to do is supply the function code in one of the languages that AWS Lambda support (current Node.js, Java, C#, and Python).

b. Event source

In AWS Lambda, we can run our code response to events, such as changes to data in an Amazon S3 bucket, Amazon DynamoDB Table. AWS Lambda is also capable of processing messages in a standard Amazon Simple Queue Service (Amazon SQS) queue. Lambda polls the queue and invokes the function synchronously with an event that contains queue messages. Lambda reads the messages in batches and invokes your function once for each batch. When your function successfully processes a batch, Lambda deletes its messages from the queue. With this capability, we can use AWS Lambda, and Amazon SQS build our application at a very low cost.

c. AWS Lambda Console

AWS Lambda Console allows developers to write code directly. A sample "Hello World" function and its handler are also given to demonstrate the programming model of AWS Lambda. Through the AWS Lambda Console, the Lambda function can also be invoked for testing, and the result is displayed directly on the console as seen in

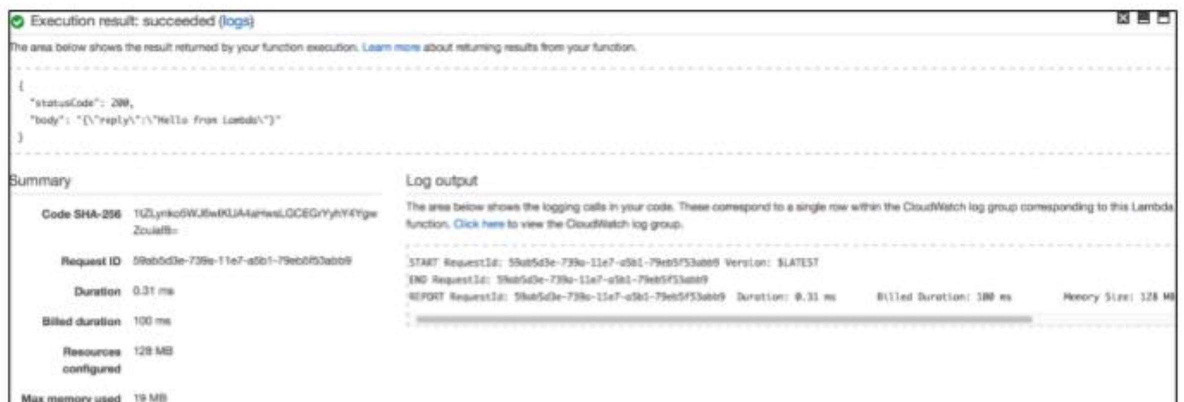


Figure 27: Result after invocation of Lambda function

Billing

AWS Lambda is charged based on the number of requests for the function (incl. test invokes from the console), and the time the code executes. First million requests a month is free.

Afterward, AWS charges $0.0000002 per request for the total number of requests across all your functions. For every GB-second used AWS charges $0.00001667 [18].

4.3.2.2 AWS SQS

AWS SQS[19] is the message queuing services provided by AWS that enables asynchronous communication. AWS SQS follows the producer/consumer paradigm. The producer can be other AWS Services, which add messages to the queuing services. The consumer subscribes to the queue can read the message from the queue and process it later.

Configuration

a. **Queue type** AWS SQS provides two types of queue:

- Standard queue is the default queue type supported by AWS SQS.
- FIFO (First in First out) queue maintains order for delivery of messages. The name of the queue should have the suffix ".fifo". It restricts 300 transactions per second and ensures that messages are delivered exactly once.

b. **Default visibility timeout**

This parameter is used when the consumer receives the message and processes it so that no other consumers have that same message. There are two possibilities:

- Once the consumer processes the message successfully, the consumer deletes the message.
- No delete call is been made until the visibility timeout expires, so the message will be available to receive a call.

c. **Message retention period**

This parameter is used to retain the message in the queue. After the message retention period retention has expired, the message will be deleted automatically. By default, the message, retention period is 4 days and can be extended up to a maximum of 14 days.

Billing

AWS offers Amazon SQS Free Tier for free with 1 million Amazon SQS requests each month. After that, the pricing for standard queue is $0.40 for 1 Million requests. The pricing for FIFO queue is $0.05 for 1 Million requests.

4.3.2.3 AWS S3

AWS Simple Storage Services[20] (S3) is a highly scalable, and available data object storage, which allows storing and retrieving all type of data from anywhere on the web. AWS S3 stores data objects in buckets. A bucket is a logical container used to identify the namespace of data objects.

[18] Price by the time of writing
[19] https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html
[20] https://docs.aws.amazon.com/s3/index.html

Folders and subfolders can also be created under the bucket to hold data objects. Every S3 data object has a unique identifier in form of URL formed by concatenating the followings components:

http://BUCKET_NAME.s3.amazonaws.com/DATA_OBJECT_KEY			
Protocol	Bucket name	S3 endpoint	Object Key

The public can have access to the data object by the URL. Access to each S3 bucket, and object are granted by, and ACL (Access Control List) which consists a series of up to 100 grants. Each grant consisting of a grantee, and permissions will control access to specific users or to groups of users. As per research, AWS S3 is considered as an excellent choice for a large choice of use cases. It ranges from storage for backups to hosting static websites.

Billing
Billing is calculated based on storage (average), data transfer in, and out, and the number of requests per month.

Storage - S3 Standard Storage	
First 50 TB / Month	$0.0245 per GB
Next 450 TB / Month	$0.0235 per GB
Over 500 TB / Month	$0.0225 per GB
Request – S3 Standard	
Data Returned by S3 Select	$0.0008 per GB
Data Scanned by S3 Select	$0.00225 per GB
PUT, COPY, POST, or LIST Requests	$0.0054 per 1,000 requests
GET, SELECT, and all other Requests	$0.00043 per 1,000 requests
Data transfer	
Data Transfer IN To Amazon S3 From Internet	
All data transfer in	$0.00 per GB
Data Transfer OUT From Amazon S3 To Internet	
Up to 1 GB / Month	$0.00 per GB

Table 2: AWS S3 Pricing for region eu-central-1 (Frankfurt) at the time of writing

4.3.2.3 AWS CloudWatch

AWS Cloud Watch[21] enables the monitoring from all AWS resources, application, and services that run on AWS, so that the developer and administrators can see the metrics and logs from AWS Resources. In order to call metrics, the following parameter must be defined. Cloud

[21] https://docs.aws.amazon.com/cloudwatch/index.html

Watch gathers detailed information corresponding to operations of Lambda functions. Moreover, various filter to search for specifying logs is provided. Figure [?] shows the graphed metric for AWS Lambda.
Cloud Watch collects logs of AWS Lambda functions, and categories them into separate groups corresponding to each function called a log group. A log group consists of multiple log messages.

CloudWatch Metrics:

a. Namespaces: a container for CloudWatch metrics. Metrics in different namespaces are isolated from each other so that metrics from different applications are not accidentally aggregated for computing statistics.
b. Metrics: represents a time-ordered set of data points that are published to CloudWatch. It can be thought of as a variable that we need to monitor, and the data points are the values of the variable over time.
c. Dimensions: a name or a value pair that uniquely identifies a metric. You can assign a maximum of 10 dimensions to a metric. Dimension help you design a structure for your statistics plan.
d. Statistics: are metric data aggregation over time specified by the user. Aggregation are made using the namespace, metric name, dimensions, and the data point unit of measure within the time period you specify.
e. Percentiles: as the name suggests, the percentile indicates the relative standing for a value in a dataset. It helps you get a better understanding of the distribution of your metric data. Percentile are used to detect anomalies.
f. Alarms: used to initiate actions on your behalf. An alarm monitors a metric over a specified interval of time and performs the assigned actions based on the value of the metric relative to a threshold over time.

AWS CloudWatch Logs

CloudWatch Logs allows users to access, monitor, and store log files from all AWS resources. It offers near real-time monitoring, and developer can search for specific phrases, values or patterns. Cloud Watch logs are managed services which can be provisioned with no extra purchase from within an AWS account.

AWS CloudWatch Events

Cloud Watch Events allows users to consume a near real time stream of events when changes to their AWS environment takes place. These event changes can subsequently trigger notifications or other actions.

4.3. CI/CD Pipeline

The expected outcome of this section will be an automated CI/CD pipeline from a local development environment to AWS. This section is organized as follows, the first subsection will provide a brief introduction to the tools, and technology used in the DevOps pipeline. The second subsection focuses on DevOps pipeline for AWS Lambda Functions, which are the main components of our application.

4.3.1 Frameworks, and development environment.

The main technical requirements from the company are the usage of GitLab as a shared code repository, which is in already in used, and the usage of Terraform to provision AWS resources. A brief introduction to the tools used in the thesis is provided below.

PyCharm IDE[22] is an integrated development environment, specifically for the Python language. PyCharm IDE has a lot of plugins supports a lot of languages used in this project including Python, HashiCorp Configuration Language (HCL) used to write Terraform Templates, and YAML file used to define GitLab CI Pipeline. Moreover, PyCharm is also fully integrated with git, ssh and shell console.

GitLab[23] is a hosted Git service like GitHub. GitLab provides a stand-alone server that can be deployed on-premises or in the cloud, and commendably most of the development of their service product is open source. GitLab in conjunction with Git is the code repository, and source control management being used at Immowelt AG besides Microsoft Team Foundation Server (TFS). For this reason, Git/GitLab was undoubtedly chosen in this project considering our project does not involve with any form of Microsoft's .NET development environment.

GitLab CI/CD[24] is a continuous integration tool built to use with GitLab. The fundamental concepts for Gitlab's approach to CI. Every repository has a single pipeline configuration, declared in a .gitlab-ci.yml file. Every commit to the repo will trigger a run of this pipeline. GitLab CI/CD organizes pipelines in stages. A stage consists of one or multiple jobs. The jobs of a stage are executed concurrently. A consecutive stage is only started if all jobs of the previous stage finished successfully.

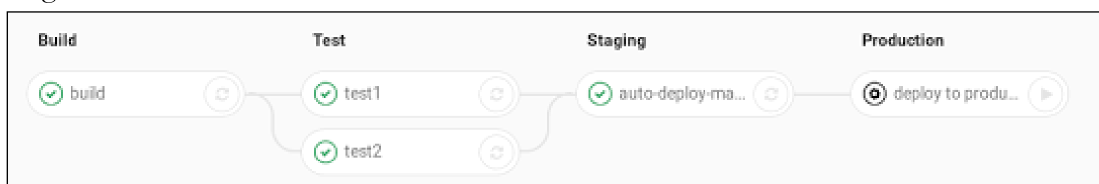


Figure 28: GitLab CI/CD pipelines

[22] https://www.jetbrains.com/pycharm/
[23] https://about.gitlab.com/stages-devops-lifecycle/
[24] https://docs.gitlab.com/ee/ci/

GitLab Runner [25] GitLab CI/CD provides an open source GitLab Runner, an automation server used to schedule, coordinate, and triggers these tasks. Moreover, GitLab CI/CD also provides an interactive web application to visualize the deployment process.

Infrastructure as Code IaC is the practice for managing infrastructure by code following the rise of cloud computing. As it is easy to provision servers, databases, and other infrastructure with a few buttons click, mistakes can be easily made provision a complex application. IaC replaces the traditional manual configuration on cloud provider 's interface with the practice of provision and manage IT infrastructure through the use of source code. The configuration files will be maintained the same as actual code. The AWS – native tool for IaC is CloudFormation, Google Cloud, and Microsoft Azure also provide their own implementations.

Terraform

Terraform[26] is an Infrastructure as Code (IaC) management tool that helps provision cloud infrastructures in a declarative way. Terraform by HashiCorp is a multivendor solution that is gaining momentum, this is also the standard at Immowelt for provisioning infrastructure in the AWS Cloud. Since the infrastructure is handled as code, IaC needs applying DevOps practices to version infrastructure code, rolled back in case of a problem.

Terraform uses Terraform configuration files called .tf file to describe infrastructure. These .tf file is written in HCL (HashiCorp Configuration Language), a human-readable syntax. It can also be stored remotely, which works better in a team environment. Terraform can be used to deploy on almost all popular cloud platform such as Amazon Web Services, Microsoft Azure, Google Cloud, Digital Ocean, etc. Terraform can be installed on any machine both inside, and outside the cloud.

In order to understand how Terraform work, it is worth first to understand how AWS resources are managed. AWS provides an interface called application programming interface (API). The developer can control every resource of AWS over the API. AWS API follows the RESTful paradigms using the HTTPs Protocol. Terraform basically makes an API calls on the user behalf to AWS.

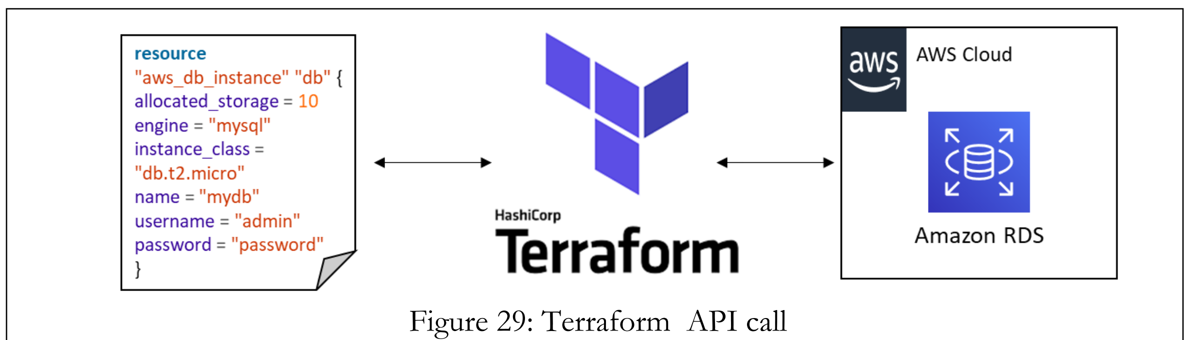


Figure 29: Terraform API call

[25] https://docs.gitlab.com/runner/
[26] https://www.terraform.io/

4.3.2 AWS Resources deployment pipeline

Project structure

Figure 30 shows the structure of the Terraform project. The project contains the configuration files (.tf files) for AWS resources and a configuration file for the deployment pipeline (.gitlab-ci.yml). The .tf files define the property of the resource such as AWS SQS, Lambda, IAM policies, etc. Each type of AWS Resources will be grouped into one .tf file. For examples, four Lambda functions of the web crawlers will be grouped together in the lambda.tf file. The file .gitlab-ci.yml describes which tasks should the GitLab CI Server executes. This file is written follows the convention published by GitLab to ensure compatibility with the GitLab CI/CD server.

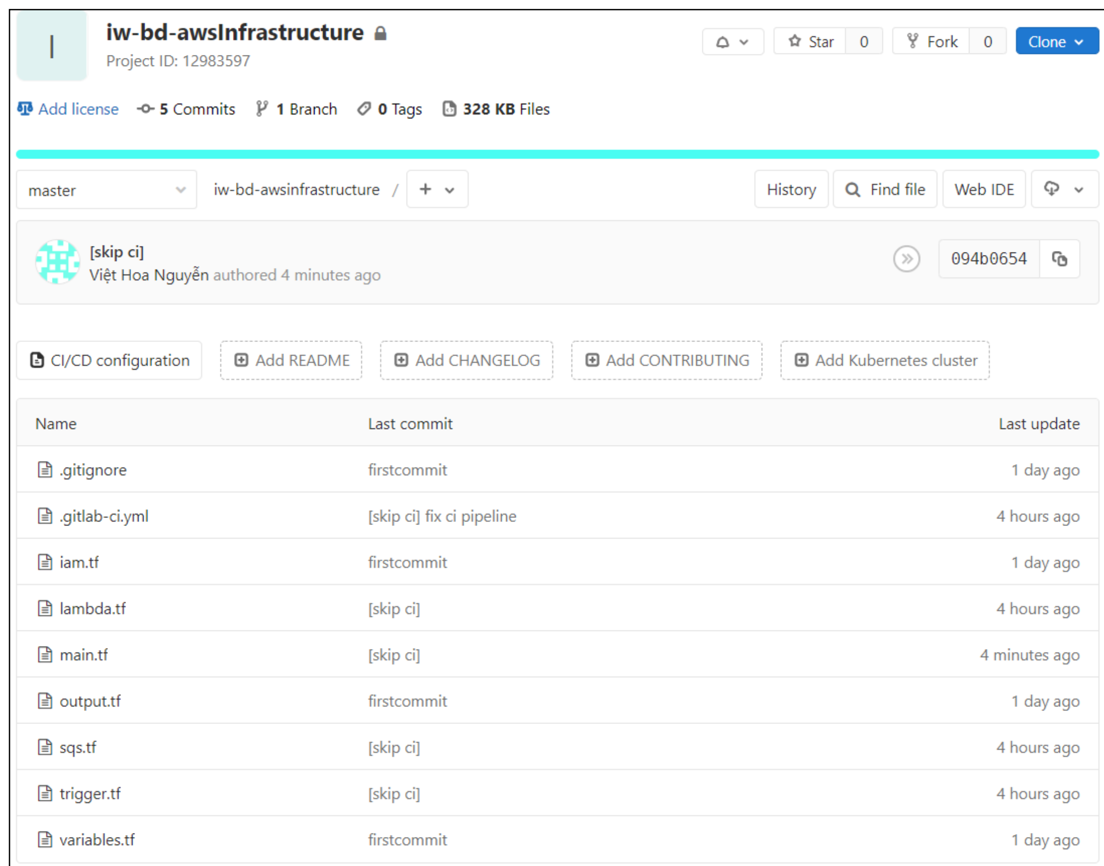


Figure 30: Terraform project repository in GitLab

Version control In this project, Git was adopted to assist the version controlling of the source code. The code is stored in a remote repository in the GitLab Server.

Deployment Figure 31 shows the workflow of the deployment process. The user made a change in the source code and commit this change locally to the master branch of the GitLab Repository. After the local changeset is taken, it then triggers the GitLab Runner deployment pipeline. The GitLab Runner takes the changeset and executes the predefined pipeline with .gitlab-ci.yml in a linux environment. The .gitlab-ci.yml contains a set command lines based on

Terraform conventions. First, the terraform plan command is executed to transform the .tf files into an execution plan. Terraform determines which new resources need to be added or which resource has to be changed. After the execution plan is created, terraform validate command will ensure that the syntax in the terraform files is correct. At last, the terraform apply command will create or update the resources according to the execution plans. Lastly, Terraform updates the state files in the AWS S3 Bucket.

State file management

After the AWS resources were created, their configurations will be saved in a .tfstate[27] file. Following the best practice, the terraform backend[28] is configurated to store the state files of the resources in an AWS S3 Bucket. AWS S3 allows file versioning, which enables to track any changes in the configurations. This is also a method to securely store state file than in a local machine. The state file is a very important file but also very fragile. For example, the state file is always loaded at the initializing of Terraform to check against any changes in the configuration file. The lost or unintentional modifications of the states file may result in the inconsistency between the state managed by Terraform, and the actual state of the resources in the cloud, which may need a lot of effort to be fixed.

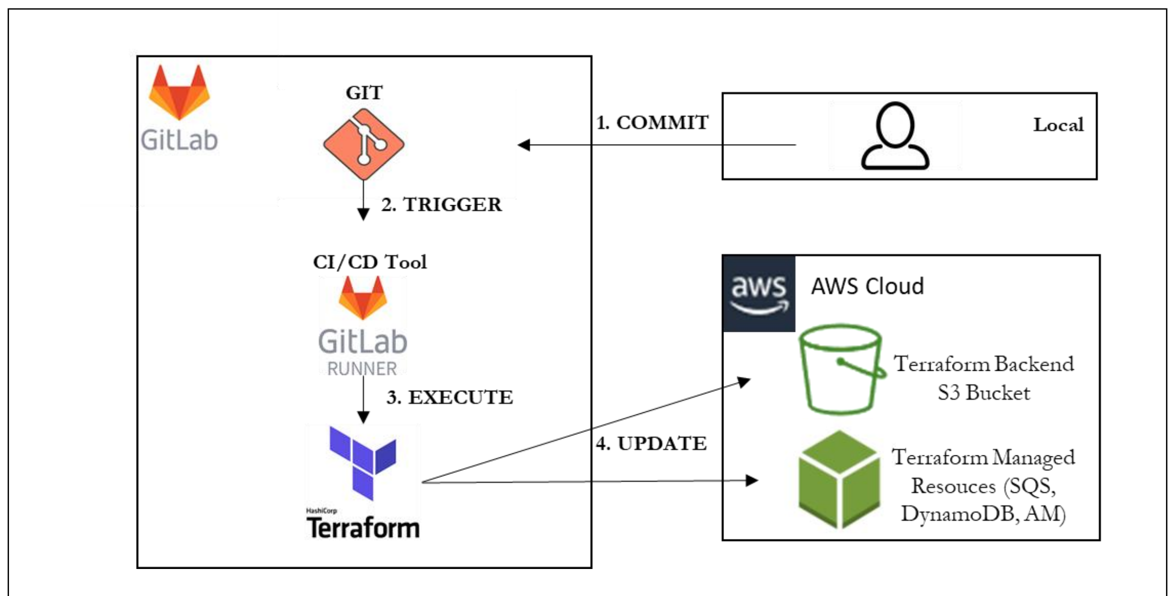


Figure 31: Deployment pipeline for AWS Infrastructure

4.3.3.2 AWS Lambda deployment pipeline

The practices described in below is applicable to all four Lambda functions of the web crawler. Each of the Lambda function will have a

[27] https://www.terraform.io/docs/state/
[28] https://www.terraform.io/docs/backends/

Source control

There is a misconception that AWS Lambda is all about a function. However, the additional components along with the code are also required for local development, debugging, and testing. First, unit tests and integration tests are still required to validate the logic and prevent regression. Second, there is a need to call 3rd party library. For example, in order to do a secure HTTP GET requests the Python library requests should be used, rather than being coded from scratch. Third, all AWS Lambda functions require a source event, and a result end. IAM role or IAM policies have to be attached to the Lambda function so that Lambda function can connect to these resources.

Managing all the modules, environment variables, testing suite, and credentials for other AWS resources can be challenging. A good, and well-defined project structure will lay a foundation for managing these dependencies. With a new project, there are many ways to organize code. The following structure shown in **Error! Reference source not found.** has worked very well in this project. All the code is placed into the src/* directory, namely the main.py script that contains code of interest. The file .gitignore specifies the intentionally untracked files, which shouldn't be commit to the remote master branch. The file .gitlab-ci.yml contains the configuration for the AWS Lambda deployment pipeline. In requirements.txt, the dependencies required in the main.py script are listed.

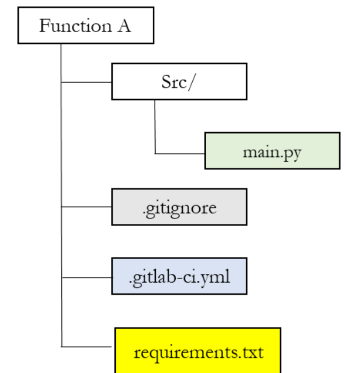


Figure 32: Project structure of AWS Lambda function

Deployment

The deployment process can be described as follows. First, and foremost, the user makes a change in the source code and commits this change to his local Git repository. After the code is reviewed and tested, it can be pushed into the master branch in the remote GitLab repository. The changes in the master branch in GitLab trigger the GitLab CI/CD pipeline, which is defined by the .gitlab-ci.yml file in the project. GitLab Runner will execute the tasks defined in the .gitlab-ci.yml which follows the conventions defined in the AWS documentation for deploying Python[29]. The deployment consists of three steps in this case package, build, and deploy. Package command will install the required modules, and dependencies written in requirements.txt. Build command means to zip the installed modules together with the Python source code to create a deployment package. The deployment package is then uploaded to a predefined AWS S3 Bucket. Package command can be done with pip, a command line tool to install Python modules. Build, and deploy are done with AWS Cli, a native AWS command line tool.

[29] https://docs.aws.amazon.com/lambda/latest/dg/lambda-python-how-to-create-deployment-package.html

Problem

The biggest problem by deploying an AWS Lambda function with Terraform and AWS S3 Bucket as code repository is that AWS Lambda function managed by Terraform cannot detect changes in the AWS S3 bucket. Therefore, it will not automatically update the code after the source code in AWS S3 has been changed. By several experiments, a workaround for this problem has been discovered. Namely, a trigger is defined in the .gitlab-ci.yml to automatically redeploy the AWS resource deployment pipeline.

First, an extra attribute must be defined in the Terraform configuration for AWS Lambda function, namely source code hash. In this case, the source code hash is created form the last modified timestamp of the AWS S3 object. After the new code has been pushed to AWS S3 bucket, the timestamp will be changed, and will not be similar to the previous hash value. The next step is to redeploy the AWS resources.

This is a dummy step because there is actually no change to the AWS resources. However, it is needed to invoke Terraform to apply the changesets from AWS S3 to AWS Lambda functions. Because, the terraform plan command checks the current configuration in the state file with the execution plan. It allows checking of current the source code hash against the previous one. If any dis-integrity is detected between the two values, Terraform will explicitly command AWS Lambda function to apply the changeset from AWS S3.

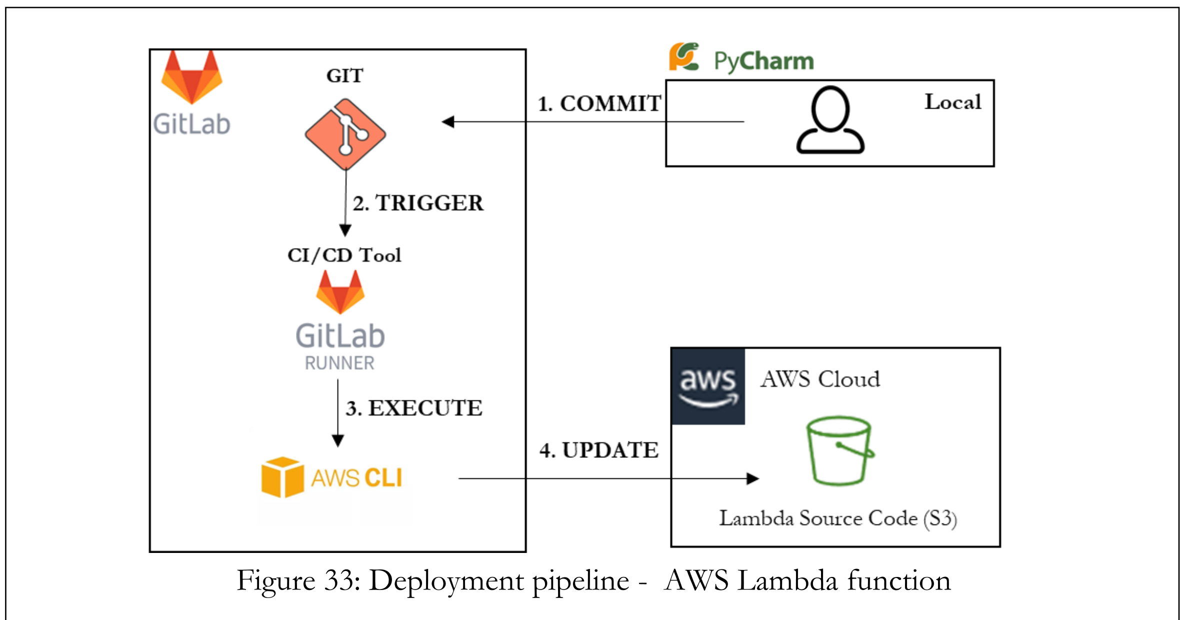


Figure 33: Deployment pipeline - AWS Lambda function

Monitoring of the deployment pipeline

Monitoring of the deployment pipeline can be easily done with the interactive visual interface of GitLab. If all deployment tasks in the .gitlab-ci.yml were successfully executed, the deployment pipeline is marked as green as success. In case of unsuccessful task, the pipeline automatically rolls back to its latest stable version.

Việt Hoa Nguyễn > iw-bd-awsInfrastructure > **Pipelines**

All 9 Pending 0 Running 0 Finished 7 Branches Tags Run Pipeline

Status	Pipeline	Triggerer	Commit	Stages	
passed	#68642797 latest		master 20eb6b3c final		00:01:52 6 days ago
failed	#68642424 latest		master 20eb6b3c final		00:01:07 6 days ago

Figure 34: Examples GitLab CI Dashboard

5. Conclusion

5.1 Evaluation

5.1.1 Web crawler

Coverage To ensure the crawler work, the outputs of the web crawlers were examined. The data retrieved were manually examined to see the coverage of the web crawler. The coverage is an indication to decide whether the web crawler is able to retrieve all the documents on the webpages. After the examinations of the S3 object storage, and the DynamoDB tables showed that the crawler has 100% coverage over the experiment 50 websites.

Duplication Manual examination was performed to ensure that there are no duplications existing in the database after each re-run of the crawler. The crawler was able to successfully identify the existing PDF documents.

Hyperlink classification Despite few errors, it is confident to confirm that the crawler is largely be able to identify the hyperlinks of official journals. It delivered an acceptable number of mistakes in classifying hyperlinks, which can be fixed manually or automatically in the text processing step of the "Baukarte" project . This problem is largely based on the fact that there is no standard in naming hyperlinks. The hard code solution can only cover a certain case.

Statistic The goal of this thesis is the design, and implementation of a distributed web crawler. It implements a prototypical web crawler which works as expected and leads to meaningful results. The resulting experiment statistics is shown in the following table.

Total numbers of seeds/city	50
Total numbers of PDF downloaded	4140
Total cost per run	$ 0,45

Tabelle 1: Web crawler's statistic

Target-Actual comparison

This prototypical implementation should be considered as a starting point for further implementations. In order to determine the functionality of the crawler, a simple comparison between the initial requirements, and the actual implementation was made, as shown in the following table.

	Target	**Actual**
The application must be deployed, and run completely on the cloud environment	x	x
Automatic deployment pipeline		x
The crawler must behave politely, and following the robot.txt rules	x	
The result of the crawling process must be according the use case:		
- Unstructured Data must be stored in the S3 object store	x	x
- Metadata, logging data must be stored in a relational database	x	x
The crawler should be designed, and implemented generically in term of the monitoring of crawling jobs	x	x
The interface for monitoring crawling jobs has the following specification:		
- Enables the monitoring following metrics: the number of successful, deferred, and failed jobs. Deferred jobs are jobs that have a temporary error and needs to be relaunched.	x	
- Allows users to keep track of the crawling history: last crawling history, crawling schedule plan.	x	
- References to the cost of the crawler history.	x	x
The programming language of choice should be Python. Furthermore, the crawler should be easily written, and debugged locally.	x	x

Table 3: Comparison between target, and actual features

Result

Figure 35 shows the folder structure of the S3 Bucket storing PDF files. Each folder has a key that is a combination between the city name and the ID of the city in the

DynamoDB. It allows the bucket to easily be searched via query and is also convenient for manual search, as city names are human-readable than a series of random numbers.

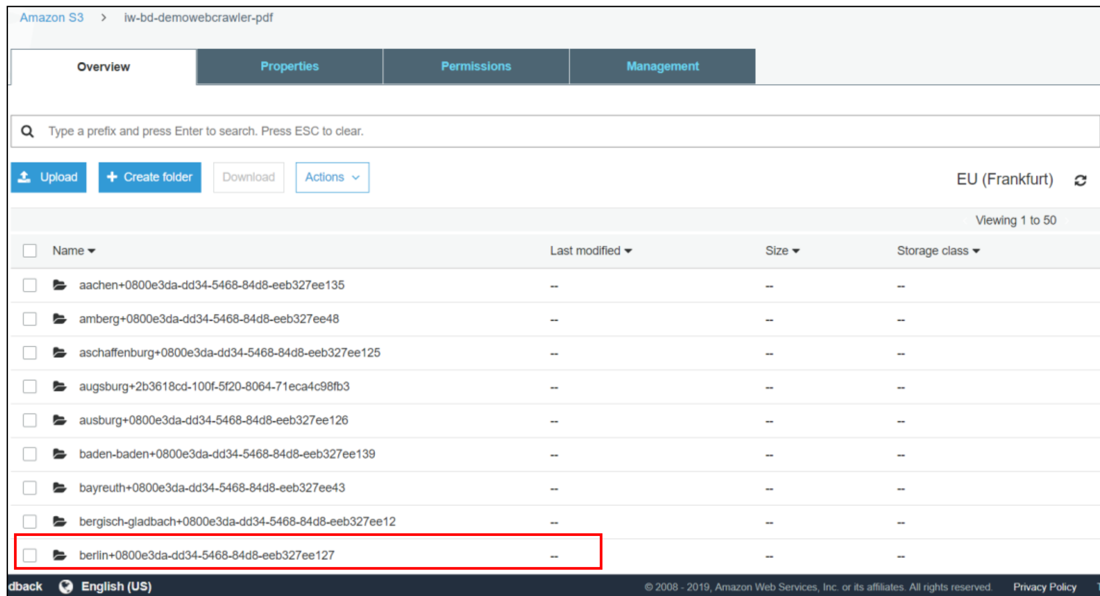


Figure 35: Screenshot - PDF folders stored in S3 Bucket

Figure 36 presents a closer look to Berlin's folder contents, which contains the official journals identified with a unique ID.

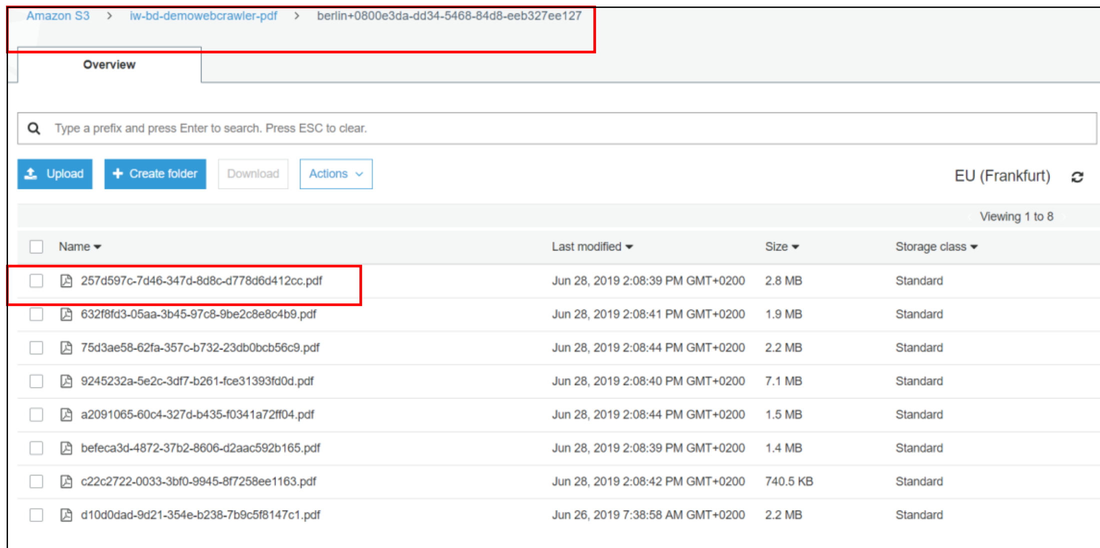


Figure 36: Screenshot - Content of Berlin's folder

With the same ID, further information about the PDF files can be retrieved from DynamoDB table as shown in Figure 37.

Figure 37: Json structure - PDF metadata

Limitations

Due to the limited time of the thesis, only a part of the crawler scheduling was done, in term of self-triggered crawling process once a week. The monitoring of the dashboard was also partially done in term of monitoring the health of AWS Lambda functions via a customized AWS CloudWatch dashboard which can be seen in the Appendix K.

A visualized crawling history, and errors couldn't be implemented due to limited time and lack of log aggregation functionality in AWS CloudWatch. Lastly, the technical implementation of the prototype application is lack of detailed database design, which might affect the efficiency of DynamoDB querying time.

Future improvement

Considering the current state of the thesis some improvements, and features could be added in the future to further enhance the coverage web crawler. First, the sources of official journal can be expanded, a lot of municipal council's websites allow the users to subscribe to RSS feed to receive newly released official journals per Email. Second, a large-scale crawling framework can be implemented to cover all of cities in German, which may require change in the requirement of the programming language. Because most the powerful framework are written mainly in Java, such as Apache Storm[30], Apache Nutch[31], Heritrix[32].

A solution for the visualization of crawling schedule, history and tasks would be to use an AWS ElasticSearch cluster, and pipe the CloudWatch logs into its for processing. Then, the Kibana interface can be used to quickly look through the logs and do basic log analytics or log aggregations. Logs can also be customized with special keywords such as "failed", "success", or "delayed", which will add some meaning information to the log.

5.1.2 Deployment pipeline

[30] https://storm.apache.org/
[31] https://nutch.apache.org/
[32] https://webarchive.jira.com/wiki/spaces/Heritrix

The implemented deployment pipeline fulfils the concept presented in section 3.3.3. The Package, Build, and Deploy are run automatically on commit. In other words, the pipeline is very user-friendly, as it can be started with a button click. The deployment is fully scripted with Terraform. CI, and CD pipelines are scripted in combination with GitLab CI. It also complies with the technology requirements at the company such as Terraform as IaC tool, and GitLab CI as CI/CD tool. The execution time of the deployment pipeline for Lambda function is estimated to be 3 minutes, and for the AWS Resources maximum 2 minutes. The execution time can vary, because the GitLab Community Runner is shared between a lot of free users. When the Runner is in high demand, the deployment pipeline might suffer from delay. Running a self-managed GitLab Runner might solve this delay problem. However, it is still very attractive considering the cost factor, since the execution of the deployment pipeline on GitLab Community Runner is free of charge.

Future improvement

The automated testing was not integrated into the deployment pipeline. This limitation can be resolved by studies of open source testing frameworks. The possible approach can be using a Python Framework called Moto [33] to create mock AWS Services and not directly on actual AWS Services.

Technology stack evaluation

Terraform is a not yet mature technologies and is still under development. Its latest version is 0.12, which means there are still plenty of changes to come in 1.0 version. The user should think carefully before getting started to rely on Terraform for provisioning cloud infrastructure. Hidden problem such as updating AWS Lambda function can be quite cumbersome to work with.

5.2 Conclusion

As proposed, a web crawler for journal officials was built, and successfully crawled with the experimental set of websites. The web crawler was implemented with a Serverless architecture using AWS Services, and aided by messages queue to increase the efficiency. In addition, the aim to implement an automated deployment pipeline was also achieved. The artifact of this thesis includes the source code of the crawler, deployment pipeline configuration, and AWS resources' configurations.

The most interesting knowledge gained over the course of the designing phase is the understanding of the complexity in developing a distributed system. This gave me a good chance to gain hand-on experiences with serverless architecture, which is very different

[33] https://github.com/spulec/moto

from the traditional monolith architecture. Especially, the monitoring of serverless application has lack of transparency and specialized monitoring tools, which makes it notoriously hard to debug errors.

It is also worth to emphasize that with the adopting of Serverless Architecture, developers will have to be involved in DevOps processes that requires additions skills such as IaC tools, and GitLab CI configurations.

During the work on this thesis, the main challenge was to be able to learn a lot of different skillsets, and technologies. This required a huge investment in time to be able to work productive with the completely new tools, because I didn't have any experiences with them beforehand. Through this thesis, I was able to extend my knowledge about not only application but also popular topics in cloud computing such as Infrastructure as Code and DevOps. This couldn't be achieved without the support of the Immowelt Big Data's Team. In conclusion, it was my pleasure to complete my thesis in a very interesting and practice-oriented way.

Bibliography

[1] "Immowelt AG," [Online]. Available: https://www.immowelt.de/immoweltag/wir/index. [Accessed 25 05 2019].

[2] "Axel Springer SE," 12 02 2015. [Online]. Available: https://www.axelspringer.com/de/presseinformationen/immowelt-und-immonet-schliessen-sich-zusammen. [Accessed 25 05 2019].

[3] Nunamaker Jr, J. F.; Chen, M,; Purdin T. D. , "Systems development in information system research," *Journal of Management Information Systems,* no. 7(3), pp. 89-106, 1990.

[4] Udapure, T.; Kale. R.; Dharmik, R, „Study of Web Crawler and its Different Types," *IOSR Journal of Computer Engineering,* Bd. 16, p. 4, 2014.

[5] "Gartner Identifies the Top 10 Trends Impacting Infrastructure and Operations for 2019," 04 12 2018. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2018-12-04-gartner-identifies-the-top-10-trends-impacting-infras. [Accessed 28 05 2019].

[6] van Eyk, E.; Iosup, A.; Seif, S. and Thömmes, M., „The SPEC Group's Research Vision on FaaS and Serverless Architectures," in *Workshop on Serverless Computing*, Las Vegas, NV, USA, 2017.

[7] Stigler, M., Beginning Serverless Computing: Developing with Amazon Web Services, Microsoft Azure, and Google Cloud, Apress Berkely, 2017.

[8] Roberts, M., „Serverless Architectures," 22 05 2018. [Online]. Available: https://martinfowler.com/articles/serverless.html. [Zugriff am 29 05 2019].

[9] Chapin, J.; Roberts, M., What is Serverless?, O'Reilly Media, Inc., 2017.

[10] A. W. S. Inc., „Amazon Web Services - AWS Well-Architected Lens-Serverless Application," 11 2018. [Online]. Available: https://d1.awsstatic.com/whitepapers/architecture/AWS-Serverless-Applications-Lens.pdf. [Zugriff am 31 05 2019].

[11] Burns, B., „Designing Distributed System," O'Reilly, Inc., 2018, p. 109.

[12] Hunt, R., „AWS Lambda Adds Amazon Simple Queue Service to Supported Event Sources," 28 06 2018. [Online]. Available: https://aws.amazon.com/blogs/aws/aws-lambda-adds-amazon-simple-queue-service-to-supported-event-sources/. [Zugriff am 05 2019].

[13] Amazon Web Services, Inc., „Understanding Scaling Behavior," [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/scaling.html. [Zugriff am 06 2019].

[14] „Using Lambda CloudWatch," AWS, [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/monitoring-functions.html. [Zugriff am 31 05 2019].

[15] "Accessing Amazon CloudWatch Logs for AWS Lambda," AWS , [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/monitoring-functions-logs.html. [Accessed 31 05 2019].

[16] Eichmann, D., "The RBSE spider: balancing effective serach against web load," in *Proceeding of the first World Wide Web Conference*, Geneva, Switzerland, 1994.

[17] IDC, "“IDC FutureScape: Worldwide IT Industry 2017 Predictions”,IDC #US41883016. MA:ID,2016".

[18] Sewak, M.; Singh; S., "Winning in the era of Serverless Computing and Function as a Service," in *2018 3rd Conference for Convergence in Technology*, Pune, India, 2018.

[19] I. Amazon Web Services, „Getting Metrics from Amazon CloudWatch," [Online]. Available: https://boto3.amazon.aws.com/v1/documentation/api/latest/guide/cw-example-metrics.html. [Zugriff am 03 26 2019].

[20] I. Amazon Web Services, „AWS Lambda Deployment Package in Python," [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda-python-how-to-create-deployment-package.html. [Zugriff am 25 05 2019].

[21] I. Amazon Web Services, „How do I build an AWS Lambda deployment package for Python?," [Online]. Available: https://aws.amazon.com/premiumsupport/knowledge-center/build-python-lambda-deployment-package/. [Zugriff am 25 05 2019].

[22] A. W. S. Inc., „Serverless Architecture with AWS Lambda".

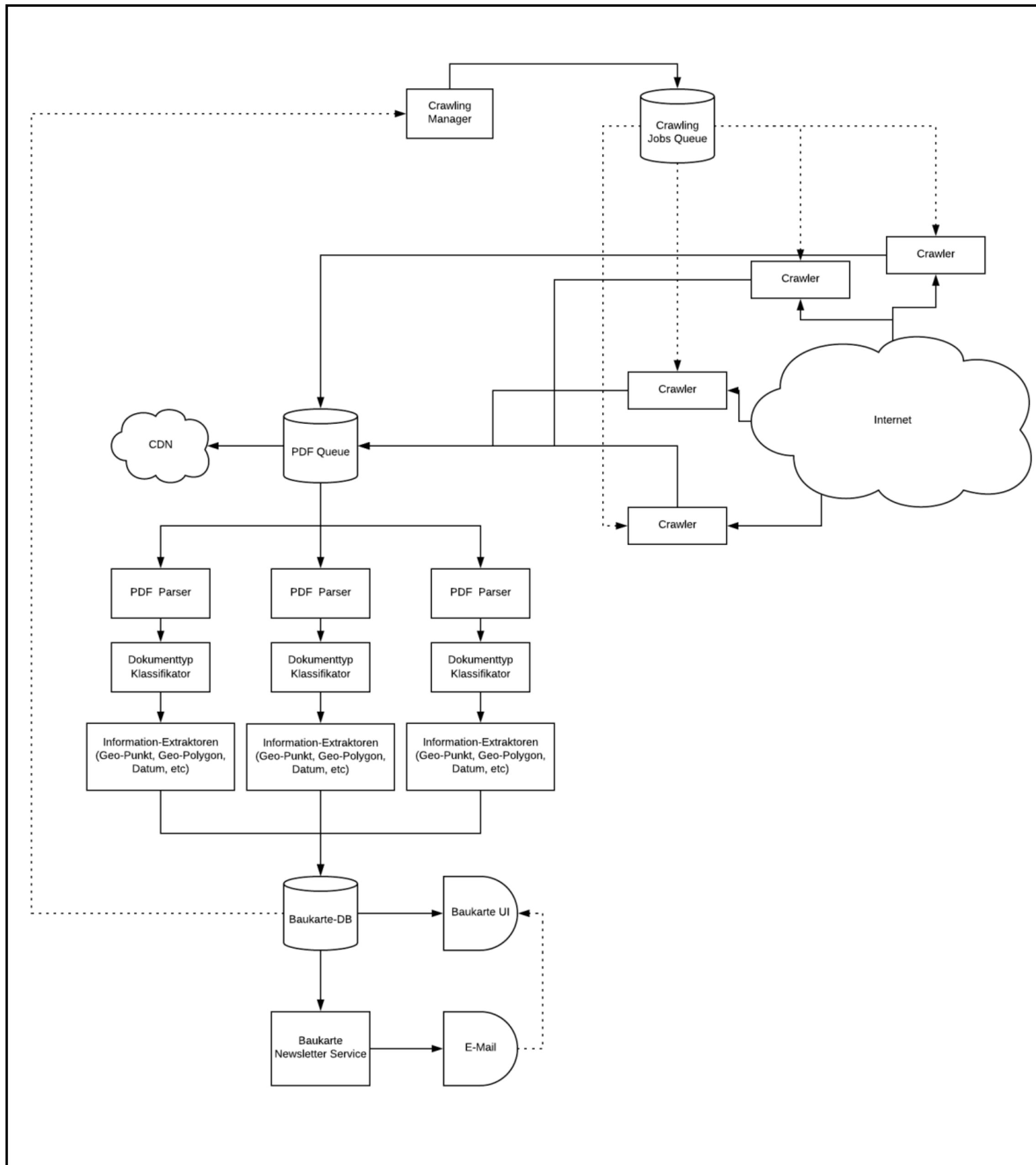
[23] Munns, C., „Serverless architecture patterns and best practices," 2017. [Online]. Available: https://www.youtube.com/watch?v=_mB1JVlhScs. [Zugriff am 27 05 2019].

[24] Pant, G.; Srinivasan, P.; Menczer, F., „Crawling the Web," in *Web Dynamics*, Berlin, Heidelberg; s.l., Springer Berlin Heidelberg, 2004, p. 4.

Appendix

Appendix A	Architecture of the “Baukarte” Project Source: Maxim Fridental (Immowelt AG)
Appendix B	Official Journal Example
Appendix C	“Baukarte” project Source: Linda Hegewald
Appendix D	Initializer Module source code
Appendix E	Python source code - Website Watcher Module
Appendix F	Python source code - Hyperlink Collector Module
Appendix G	Python source code - Hyperlink Processor Module
Appendix H	AWS Resources configuration
Appendix I	Source code - .gitlab.yml for deploying lambda function
Appendix K	Source code - .gitlab-ci.yml file for deploy AWS resources

Appendix A: "Baukarte Project Architecture"



Appendix B: Official Journal Example

Amtliche Mitteilungen der Stadt Ingolstadt

Herausgegeben vom Presse- und Informationsamt
der Stadt Ingolstadt, Franziskanerstr. 7 , 85049 Ingolstadt

NR. 43 MITTWOCH, 24.10.2018

Die Stadt Ingolstadt informiert Sie über Ihre bestehenden Widerspruchsrechte bei folgenden Datenübermittlungen:

1. Melderegisterauskünfte/Datenübermittlungen an Parteien, Wählergruppen und anderen Trägern von Wahlvorschlägen im Zusammenhang mit Wahlen und Abstimmungen auf staatlicher und kommunaler Ebene in den sechs der Wahl oder Abstimmung vorangehenden Monaten. Hierzu gehören auch Abstimmungen im Zusammenhang mit Volksbegehren, Volksentscheiden sowie Bürgerentscheiden.

   Rechtsgrundlagen: § 50 Abs. 1 und 5 des Bundesmeldegesetzes (BMG)

   Hinweise: Der Widerspruch kann nur bei der Meldebehörde eingelegt werden, bei der der alleinige Wohnsitz oder der Hauptwohnsitz (bei mehreren Wohnungen) besteht.

2. Melderegisterauskünfte/Datenübermittlungen an Mandatsträger, Presse oder Rundfunk über Alters- oder Ehejubiläen

   Rechtsgrundlage: § 50 Abs. 2 und 5 BMG

   Hinweise: Der Widerspruch gilt im Hinblick auf Ehejubiläen auch für den anderen Ehegatten -Lebenspartner und ist bei allen Meldebehörden einzulegen, in deren Zuständigkeitsbereich Sie mit einer Wohnung (bei mehreren Wohnungen) gemeldet sind.

3. Melderegisterauskünfte/Datenübermittlungen an Adressbuchverlage zur Herstellung von Adressenverzeichnissen in Buchform.

   Rechtsgrundlage: § 50 Abs. 3 und 5 BMG

   Hinweise: Der Widerspruch ist bei allen Meldebehörden einzulegen, in deren Zuständigkeitsbereich Sie mit einer Wohnung (bei mehreren Wohnungen) gemeldet sind.

4. Datenübermittlungen an das Bundesamt für Personalmanagement der Bundeswehr. Die Datenübermittlung erfolgt bis 31.3. eines Jahres über Personen, die im nächsten Jahr volljährig werden und die deutsche Staatsangehörigkeit besitzen.

   Rechtsgrundlagen: § 58 c Abs. 1 des Soldatengesetzes (SG) i. V. m. § 36 Abs. 2 BMG

   Hinweise: Der Widerspruch kann nur bei der Meldebehörde eingelegt werden, bei der der alleinige Wohnsitz oder der Hauptwohnsitz (bei mehreren Wohnungen) besteht. Ein etwaiger Widerspruch wird mit Vollendung des 18. Lebensjahres automatisch gelöscht. Widersprüche, die nach der bisherigen Rechtslage eingetragen wurden, behalten ihre Gültigkeit.

5. Datenübermittlungen von Familienangehörigen an öffentlich-rechtliche Religionsgesellschaften, sofern sie nicht derselben oder keiner öffentlich-rechtlichen Religionsgesellschaft angehören. Familienangehörige sind der Ehegatte oder Lebenspartner, minderjährige Kinder und die Eltern von minderjährigen Kindern. Das Widerspruchsrecht gilt nicht, sofern die Daten für Zwecke des Steuererhebungsrechts der jeweiligen öffentlich-rechtlichen Religionsgesellschaft übermittelt werden.

   Rechtsgrundlage: § 42 Abs. 1 bis 3 BMG

   Betroffene haben das Recht, den Datenübermittlungen zu widersprechen. Der Widerspruch ist an keine Voraussetzung gebunden und braucht nicht begründet zu werden. Er kann beim Bürgeramt der Stadt Ingolstadt, Rathausplatz 4, 85049 Ingolstadt eingelegt werden.

Falls der Datenübermittlung nicht widersprochen wurde, werden die Meldebehörden die genannten Daten weitergeben.

Baugenehmigung der Stadt Ingolstadt (Az.:02668 18 10)

Vorhaben/Betreff:	Dachgeschossausbau zu 3. WE
Grundstück:	Ingolstadt, Am Sunder 2
Gemarkung:	Zuchering
Flur-Nr.:	2144/2

Die Stadt Ingolstadt erteilte zu o.a. Vorhaben eine Genehmigung (Bescheid vom15.10.2018). Geplant ist der Dachgeschossausbau zu einer dritten Wohneinheit.

Als Baugenehmigungsbehörde weist die Stadt Ingolstadt alle benachbarten Grundstückseigentümer der o.a. Baumaßnahme darauf hin, dass die o.a. genehmigten Planunterlagen beim Bauordnungsamt der Stadt Ingolstadt, Spitalstr. 3, 1. Stock, Zimmer Nr. 103 (Tel.: 305-2222) zu den üblichen Geschäftsstunden eingesehen werden können. Rechtsgrundlage für diese Veröffentlichung ist Art. 66 Abs. 2 Satz 4 der Bayerischen Bauordnung (BayBO).

Rechtsbehelfsbelehrung

Gegen diesen Bescheid kann innerhalb eines Monats nach seiner Bekanntgabe Klage bei dem Bayerischen Verwaltungsgericht München erhoben werden. Dafür stehen folgende Möglichkeiten zur Verfügung:

a) Die Klage kann schriftlich oder zur Niederschrift des Urkundsbeamten bei der Geschäftsstelle erhoben werden. Die Anschriften lauten:

   Bayerisches Verwaltungsgericht München
   Postfachanschrift: Postfach 20 05 43, 80005 München
   Hausanschrift: Bayerstraße 30, 80335 München,

b) Die Klage kann bei dem Bayerischen Verwaltungsgericht München auch durch Übermittlung eines elektronischen Dokuments mit qualifizierter Signatur an das elektronische Gerichts- und Verwaltungspostfach – www.egvp.de – erhoben werden. Dabei sind die der Internetpräsenz der Verwaltungsgerichtsbarkeit zu entnehmenden Bedingungen zu beachten: http://www.vgh.bayern.de/verwaltungsgerichtsbarkeit/rechtsantragsstelle/.

Die Klage muss den Kläger, die Beklagte (Stadt Ingolstadt) und den Gegenstand des Klagebegehrens bezeichnen und soll einen bestimmten Antrag enthalten. Die zur Begründung dienenden Tatsachen und Beweismittel sollen angegeben, der angefochtene Bescheid soll in Urschrift oder in Abschrift beigefügt werden. Wenn die Klage schriftlich oder zur Niederschrift erhoben wird, sollen dieser und allen Schriftsätzen Abschriften für die übrigen Beteiligten beigefügt werden.

Hinweise zur Rechtsbehelfsbelehrung:

- Die Einlegung eines Rechtsbehelfs per einfacher Mail ist nicht zugelassen und entfaltet keine rechtlichen Wirkungen! Nähere Informationen zur elektronischen Einlegung von Rechtsbehelfen können der Internetpräsenz der Bayerischen Verwaltungsgerichtsbarkeit entnommen werden (www.vgh.bayern.de)
- Kraft Bundesrechts ist in Prozessverfahren vor den Verwaltungsgerichten grundsätzlich ein Gebührenvorschuss zu entrichten.

Bekanntmachung im Rahmen der Berichtigung des Bestandsverzeichnisses

Widmung eines Teilstückes einer Ortsstraße

Das in der Stadt Ingolstadt, Regierungsbezirk Oberbayern, gelegene Teilstück der Straße „Alban-Berg-Straße", wird laut Lageplan als Ortsstraße öffentlich gewidmet.

Die Widmungsverfügung kann bei der Stadt Ingolstadt, Technisches Rathaus, Zimmer 402, im 4. Stock, eingesehen werden.

Widmung eines beschränkt-öffentlichen Weges

Der in der Stadt Ingolstadt, Regierungsbezirk Oberbayern, gelegene Weg wird laut Lageplan als Gehweg gewidmet.

Die Widmungsverfügung kann bei der Stadt Ingolstadt, Technisches Rathaus, Zimmer 402, im 4. Stock, eingesehen werden.

INHALT

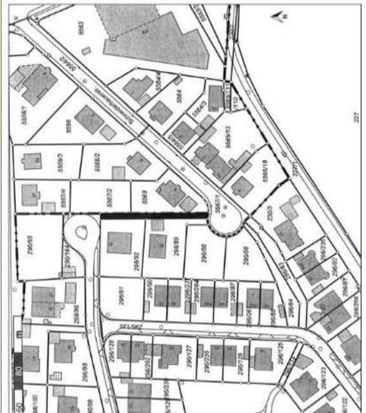
Bürgeramt
Bekanntmachung

Bauordnungsamt
Baugenehmigung

Tiefbauamt
– Widmung
– Erhebung Kostenerstattungsbetrag

ZV Zentralkläranlage Ingolstadt
Öffentliche Ausschreibung

Sparkasse Ingolstadt-Eichstätt
Aufgebot von Sparkassenbüchern u. sonstigen Sparurkunden



Erhebung eines Kostenerstattungsbetrages

Die Ausgleichs- und Ersatzmaßnahmen für Eingriffe in Natur und Landschaft für das Gebiet des Bebauungsplanes Nr. 145 H „Niederfeld" wurden abgeschlossen.

Betroffen sind die baulich nutzbaren Grundstücke am Plunderweg von Fl.Nr. 714/19 bis Wendehammer.

Gemäß Baugesetzbuch und der Kostenerstattungsbetragssatzung werden daher für o.g. Maßnahmen Kostenerstattungsbeträge erhoben, sobald die Voraussetzungen für die Verteilung des Aufwandes vorliegen.

Öffentliche Ausschreibung

Der Zweckverband Zentralkläranlage Ingolstadt, Am Mailinger Moos 145, 85055 Ingolstadt beabsichtigt folgende Leistung nach VOL/A zu vergeben:

Verwertung und Transport von kommunalem
Klärschlamm Nr. ZKA-020-2018

Einreichungstermin: 06.11.2017 um 24:00 Uhr, Ausführungsort: Ingolstadt Abwicklung der Ausschreibung über das Baureferat, Spitalstr. 3, 85049 Ingolstadt Tel. (0841) 305-2446, Fax (0841) 305-2447, E-Mail: vergabe@ingolstadt.de Auskünfte zur Ausschreibung über die Vergabeplattform www.vergabe.bayern.de

Aufgebot von Sparkassenbüchern und sonstigen Sparurkunden

Gemäß Art. 35 und 36 AGBGB wird hiermit auf Antrag der nachstehend aufgeführten Antragsteller der Inhaber ds/der jeweiligen Sparkassenbuches/Sparurkunde aufgefordert, seine Rechte unter V orlegung der Urkunde binnen drei Monaten bei der Sparkasse Ingosltadt Eichstätt anzumelden. Wird die Urkune innerhalt dieser Frist nicht vorgelegt, so wird das jeweilige Sparkassenbuch/die jeweilige Sparurkunde durch Beschluss des Vorstandes für kraftlos erklärt.

Antragsteller	Urkundennummer
Sibel Temel	3165342050

Appendix C: "Baukarte" Project

Amtliche Mitteilungen
der Stadt Ingolstadt

Baugenehmigung der Stadt Ingolstadt
(Az.:02668 18 10)

Vorhaben/Betreff: Dachgeschossausbau zu 3. WE
Grundstück: Ingolstadt, Am Sunder 2
Gemarkung: Zuchering
Flur-Nr.: 2144/2

Die Stadt Ingolstadt erteilte zu o.a. Vorhaben eine Genehmigung (Bescheid vom 15.10.2018). Geplant ist der Dachgeschossausbau zu einer dritten Wohneinheit.


```
{
    "massnahmenID" : "5d2a76b91",
    "url" : "http://www.nuernberg.de/amtsblatt0119.pdf",
    "aktenzeichen" : "BI-062-37",
    "datum_beschluss" : "2019-01-07T00:00:00Z",
    "massnahmen_art" : "Baugenehmigung",
    "beschreibung_kurz" : "Ausbau DG zu 3. Wohneinheit",
    "beschreibung_original" : "Baugenehmigung Anwesen Stadtstrasse 7 ...",
    "adressen" : [
    {
        "strasse" : "Stadtstrasse 7",
        "stadt" : "Nürnberg",
        "lat" : "49.460983",
        "lon" : "11.061859"
    },
    {
        "strasse" : "Stadtstrasse 9",
        "stadt" : "Nürnberg",
        "lat" : "49.460984",
        "lon" : "11.061858"
    }]
}
```


immo erst zu immowelt.de
Nordostpark 3, 90411 Nürnberg
Karte Satellit
Baugenehmigung
vom 07.01.2019
Nordostpark 5, Nürnberg
Umbau Immowelt AG
Baugenehmigung Anwesen Nordostpark 3 bis Nordostpark 7 wird die Immowelt AG umgebaut. Baugenehmigung Anwesen Nordostpark 3 bis Nordostpark 7 wird die Immowelt AG umgebaut. Baugenehmigung Anwesen Nordostpark 3 bis Nordostpark 7 wird die Immowelt ... mehr lesen

Official journal — Building permit — Information — Visualization

Appendix D: Initializer Module

```
1   import boto3
2
3   session = boto3.Session(region_name='xxxxx',
4                           aws_access_key_id='xxxxx',
5                           aws_secret_access_key='xxxxx')
6
7   class DynamoDBUtils:
8       def __init__(self,tableName):
9           self.tableName= tableName
10          self.conn = session.resource('dynamodb')
11          self.table = self.conn.Table(self.tableName)
12
13      def getallItemsID(self):
14          res = self.table.scan()
15          id_list = [item['ID'] for item in res['Items']]
16          return id_list
17
18      def getallItemsID, andURL(self):
19          res = self.table.scan()
20          id_list = [(item['ID'],item['url'],item['name']) for item in res['Items']]
21          return id_list
22
23  def send_message_to_queue (id, url,name):
24      session.client('sqs').send_message(
25          QueueUrl = "https://sqs.eu-central
26  1.amazonaws.com/419206837402/seeds_v1",
27          MessageBody=str(url),
28          MessageAttributes={
29              'ID':{
30                  'DataType':'String',
31                  'StringValue': str(id)
32              },
33              'Name': {
34                  'DataType': 'String',
35                  'StringValue': str(name)
36              }
37          }
38      )
39
40  def main(event,context): # Lambda Handler
41      # Create endpoint to DynamoDB Table
42      utils = DynamoDBUtils('seeds')
43      id_list = utils.getallItemsID, andURL()
44
45      for id,url,name in id_list:
46          send_message_to_queue(id,url,name)
```

Appendix E: Website Watcher Module

```
1   import hashlib
2   import boto3
3   from bs4 import BeautifulSoup as bs
4   import requests
5   from datetime import datetime
6
7   session = boto3.Session(region_name='xxxxx',
8                       aws_access_key_id='xxxxx',
9                       aws_secret_access_key='xxxxx'
10
11  class websiteHasher:
12      def __init__(self,url_id,url_to_hash,name):
13          self.url_id = url_id
14          self.url_to_hash = url_to_hash
15          self.city_name= name
16
17      def generateHash(self):
18          try:
19              cleanedHTML = HTMLSelector(requests.get(self.url_to_hash).text).selectText()
20              return hashlib.sha224(cleanedHTML).hexdigest()
21          except requests.exceptions.ConnectionError as e:
22              return None
23
24      def getLastHash(self):
25          conn = session.resource('dynamodb')
26          table = conn.Table('seeds')
27          print(self.url_id)
28          res = table.get_item(
29              Key={
30                  'ID': str(self.url_id)
31              }
32          )
33          return res['Item']['lastHashValue']
34
35      def updateHash(self,new_hash_value):
36          conn = session.resource('dynamodb')
37          table = conn.Table('seeds')
38          table.update_item(
39              Key={
40                  'ID': str(self.url_id)
41              },
42              UpdateExpression="SET lastHashValue = :var1, lastModified= :var2",
43              ExpressionAttributeValues={
44                  ':var1': new_hash_value,
45                  ':var2': str(datetime.now().date())
46              }
47          )
48
49      def compareHash(self):
50          lastHashValue = self.getLastHash()
51          currentHashValue = self.generateHash()
```

```
52          if lastHashValue != None:
53              if currentHashValue != lastHashValue:
54                  self.updateHash(currentHashValue)
55                  self.send_message_to_queue()
56
57  def send_message_to_queue (self):
58          session.client('sqs').send_message(
59              QueueUrl = 'https://sqs.eu-central-
60  1.amazonaws.com/419206837402/cityURLqueue_v1',
61              MessageBody=str(self.url_to_hash),
62              MessageAttributes={
63                  'parent_URL_ID':{
64                      'DataType':'String',
65                      'StringValue': str(self.url_id)
66                  },
67                  'Name': {
68                      'DataType': 'String',
69                      'StringValue': str(self.city_name)
70                  },
71              }
72          )
73
74
75  class HTMLSelector:
76      def __init__(self,html):
77          self.html = html
78
79      '''Select only the javascript, and styles in the html document'''
80      def selectJavascript(self):
81          soup = bs(self.html,features='html.parser')
82          return [x.extract() for x in soup.findAll(['script','style'])]
83
84      def selectText(self):
85          soup = bs(self.html,features='html.parser')
86          for x in soup.findAll(['script','style']):
87                  x.decompose()
88                  text = soup.get_text()
89          # break into lines, and remove leading, and trailing space on each
90                  lines = (line.strip() for line in text.splitlines())
91          # break multi-headlines into a line each
92                  chunks = (phrase.strip() for line in lines for phrase in line.split("  "))
93          # drop blank lines
94          text = '\n'.join(chunk for chunk in chunks if chunk)
95          # encode text
96          return text.encode('utf-8')
97
98
99  def lambda_handler(event, context):
100     for record in event['Records']:
```

```
101        url = str(record['body'])
102        payload= record['messageAttributes']
103        id= payload['ID']['stringValue']
104        name= payload['Name']['stringValue']
105        hasher = websiteHasher(id,url,name)
106        hasher.compareHash()
```

Appendix F: Hyperlink Collector Module

```
1   from bs4 import BeautifulSoup
2   import requests
3   import hashlib
4   from urllib.parse import urljoin
5   import boto3
6   from boto3.dynamodb.conditions import Key
7   from datetime import datetime
8   from botocore.exceptions import ValidationError
9   from uuid import uuid3, NAMESPACE_URL
10
11  #entrypoint to AWS
12  session = boto3.session.Session( region_name='eu-central-1',
13                                aws_access_key_id='xxxxxx',
14                                aws_secret_access_key='xxxxxx')
15
16  class Hyperlink_Collector:
17      def __init__(self,parent_url,parent_url_id,city_name):
18          self.db = session.resource('dynamodb')
19          self.sqs = session.client('sqs')
20          self. parent_url_id= parent_url_id
21          self.parent_url = parent_url
22          self.city_name=city_name
23
24      def getPage(self):
25          try:
26              s = requests.Session()
27              req = s.get(self.parent_url)
28              return BeautifulSoup(req.text, features='html.parser')
29          except requests.exceptions.RequestException:
30              return None
31
32      def isAbsoluteLink(self, link):
33          if str(link).startswith("http://") or str(link).startswith("https://"):
34              return True
35          return False
36
37      def resolveRelativeURL(self,parent_url,url):
38          return urljoin(parent_url,url)
39
40      def create_uuid (self,url):
41          parent_url_enc = self.parent_url.encode('utf-8')
42          url_enc = url.encode('utf-8')
43          hashtring = parent_url_enc + url_enc
44          url_id = hashlib.md5(hashtring).hexdigest()
45          return url_id
46
47      def create_uuid3(self,url):
```

```
48            uuid= uuid3(NAMESPACE_URL, url)
49            return str(uuid)
50
51        def insert_hyperlink(self,url,link_text,url_id):
52            self.db.Table('urlsList').put_item(
53               Item= {
54                  "downloaded": False,
55                  "downloaded_on": "None",
56                  "full_url": str(url),
57                  "parent_url":str(self.parent_url),
58                  "link_text": str(link_text),
59                  "valid": True,
60                  "visited": False,
61                  "visited_on": "None",
62                  "downloaded_tries_number":0,
63                  "uuid": str(url_id)
64               }
65            )
66
67        def mark_error (self):
68            self.db.Table('seeds').update_item(
69               Key={
70                  'ID': self.parent_url_id,
71               },
72               UpdateExpression="SET valid=:var1, history=:var2",
73               ExpressionAttributeValues={
74                  ':var1': False,
75                  ':var2': str(datetime.utcnow().isoformat())+': '+' unable to get page'
76               },
77            )
78            print('unable to get page:' + self.parent_url)
79
80        def send_message_to_queue(self, url, url_id):
81            self.sqs.send_message(
82               QueueUrl="https://sqs.eu-central-
83    1.amazonaws.com/419206837402/pdfURLqueue_v1",
84               MessageBody=str(url),
85               MessageAttributes={
86                  'ID': {
87                     'StringValue': url_id,
88                     'DataType': 'String'
89                  },
90                  'parent_URL_ID':{
91                     'StringValue': self.parent_url_id,
92                     'DataType': 'String'
93                  },
94                  'parent_URL':{
95                     'StringValue': self.parent_url,
96                     'DataType': 'String'
```

```
97                },
98                'cityName': {
99                    'StringValue': self.city_name,
100                   'DataType': 'String'
101               }
102           }
103       )
104       print(str(url)+':is added to queue')
105
106   # CHECK IF URL ALREADY EXISTS IN TABLE
107   def isURLDupilcate(self, url):
108       table= self.db.Table('urlsList')
109       res = table.query(
110           KeyConditionExpression=Key('parent_url').eq(str(self.parent_url)) &
111 Key('full_url').eq(str(url))
112       )
113       return True if len(res['Items'])!=0 else False
114
115   # CHECK IF URL POINTS TO PDF FILE
116   def is_PDFFile(self,url):
117       try:
118           h = requests.head(url, allow_redirects=True)
119           header = h.headers
120           content_type = header.get('content-type')
121           if ('pdf' in content_type) or ('PDF' in content_type):
122               return True
123       except requests.exceptions.RequestException:
124           return False
125
126    # SAVE CRAWLING HISTORY
127   def mark_last_crawled(self):
128       self.db.Table('seeds').update_item(
129           Key={
130               'ID': self.parent_url_id,
131           },
132           UpdateExpression="SET #st =:var1",
133           ExpressionAttributeValues={
134               ':var1': str(datetime.utcnow().isoformat())
135           },
136           ExpressionAttributeNames= {
137               '#st':'last_crawled'
138           }
139       )
140
141   def collect(self):
142       """
143       Searches a given website for all links related to Amtsblatt, and records all pages found
144       """
145       soup = self.getPage()
```

```
146          if soup is not None:
147              for link in soup.findAll('a'):
148                  text = str(link.getText().strip())
149                  href = str(link.get('href'))
150
151                  if self.isAbsoluteLink(href) is not True:
152                      href = self.resolveRelativeURL(self.parent_url,href)
153
154                  if self.is_PDFFile(href) :
155                      if self.isURLDupilcate(href) is False :
156                          print(href)
157                          doc_id= self.create_uuid3(href)
158                          self.insert_hyperlink(href, text,doc_id)
159                          self.send_message_to_queue(href,doc_id)
160                          self.mark_last_crawled()
161
162
163  def main(event,context):
164      print(event)
165      for record in event['Records']:
166          url = str(record['body'])
167          payload = record['messageAttributes']
168          cityURL_id = payload['parent_URL_ID']['stringValue']
169          city_name = payload['Name']['stringValue']
170          collector = Hyperlink_Collector(url,cityURL_id,city_name)
171          result= collector.collect()
172          if result is False:
173              pass
```

Appendix G: Hyperlink Processor Module

```
1   import requests
2   from datetime import datetime
3   from boto3.dynamodb.conditions import Attr
4
5   import boto3
6
7   session = boto3.session.Session(region_name='eu-central-1',
8                                   aws_access_key_id='xxxxxx',
9                                   aws_secret_access_key='xxxxxx'
10  )
11
12  class Hyperlink_Processor:
13      def __init__(self,id,url,parent_url_id,parent_url,city_name):
14          self.db=session.resource('dynamodb')
15          self.s3=session.resource('s3')
16          self.id= id
17          self.parent_url_id= parent_url_id
18          self.url=url
19          self.parent_url=parent_url
20          self.city_name=city_name
21
22      def get_pdf(self):
23          try:
24              req = requests.get(self.url,stream =True)
25              return req.content
26          except Exception as e:
27              return e
28
29      def put_object (self,content):
30          try:
31              object= self.s3.Object('iw-bd-demowebcrawler-
32  pdf',str(self.city_name)+'+'+str(self.parent_url_id)+'/'+str(self.id)+'.pdf')
33              res= object.put(Body=content)
34              print(res)
35              self.mark_downloaded()
36          except Exception as e:
37              return e
38
39      def mark_visited(self):
40          self.db.Table('urlsList').update_item(
41              Key={
42                  'parent_url': str(self.parent_url),
43                  'full_url': str(self.url)
44              },
45              UpdateExpression="SET visited=:var1, visited_on=:var2",
46              ExpressionAttributeValues={
47                  ':var1': True,
```

```
48                  ':var2':str(datetime.utcnow().isoformat())
49              }
50          )
51  
52      def mark_downloaded(self):
53          print(datetime.utcnow().isoformat())
54          self.db.Table('urlsList').update_item(
55              Key={
56                  'parent_url': self.parent_url,
57                  'full_url': self.url
58              },
59              UpdateExpression="SET downloaded=:var1, downloaded_on=:var2",
60              ExpressionAttributeValues={
61                  ':var1': True,
62                  ':var2': str(datetime.utcnow().isoformat())
63              },
64          )
65  
66      def visit(self):
67          self.mark_visited()
68          content = self.get_pdf()
69          self.put_object(content)
70  
71  def main(event,context):
72      print(str(datetime.utcnow().isoformat()))
73      for record in event['Records']:
74          url = str(record['body'])
75          payload = record['messageAttributes']
76          parent_id= payload['parent_URL_ID']['stringValue']
77          parent_url = payload['parent_URL']['stringValue']
78          city_name= payload['cityName']['stringValue']
79          id= payload['ID']['stringValue']
80          processor = Hyperlink_Processor(id,url,parent_id,parent_url,city_name)
81          processor.visit()
```

Appendix G: AWS resources configuration files

iam.tf

```
1   resource "aws_iam_role" "example_lambda" {
2     assume_role_policy = <<EOF
3   {
4     "Version": "2012-10-17",
5     "Statement": [
6       {
7         "Effect": "Allow",
8         "Principal": {
9           "Service": "lambda.amazonaws.com"
10        },
11        "Action": "sts:AssumeRole"
12      }
13    ]
14  }
15  EOF
16  }
17
18  resource "aws_iam_role_policy_attachment" "example_lambda" {
19    policy_arn = "${aws_iam_policy.example_lambda.arn}"
20    role = "${aws_iam_role.example_lambda.name}"
21  }
22
23  resource "aws_iam_policy" "example_lambda" {
24    policy = "${data.aws_iam_policy_document.example_lambda.json}"
25  }
26
27  data "aws_iam_policy_document" "example_lambda" {
28    statement {
29      sid       = "AllowSQSPermissions"
30      effect    = "Allow"
31      resources = ["arn:aws:sqs:*"]
32
33      actions = [
34        "sqs:ChangeMessageVisibility",
35        "sqs:DeleteMessage",
36        "sqs:GetQueueAttributes",
37        "sqs:ReceiveMessage",
38      ]
39    }
40
41    statement {
42      sid       = "AllowInvokingLambdas"
43      effect    = "Allow"
44      resources = ["arn:aws:lambda:eu-central-1:*:function:*"]
45      actions   = ["lambda:InvokeFunction"]
46    }
```

```
47
48   statement {
49     sid      = "AllowCreatingLogGroups"
50     effect   = "Allow"
51     resources = ["arn:aws:logs:eu-central-1:*:*"]
52     actions   = ["logs:CreateLogGroup"]
53   }
54   statement {
55     sid      = "AllowWritingLogs"
56     effect   = "Allow"
57     resources = ["arn:aws:logs:eu-central-1:*:log-group:/aws/lambda/*:*"]
58
59     actions = [
60       "logs:CreateLogStream",
61       "logs:PutLogEvents",
62     ]
63   }
64 }
```

main.tf

```
1  provider "aws" {
2    region     = "eu-central-1"
3    access_key = "${var.aws_access_key}"
4    secret_key = "${var.aws_secret_key}"
5  }
6
7  # BUCKET FOR TERRAFORM STATE
8  resource "aws_s3_bucket" "iw-bd-demowebcrawler-pdf" {
9    bucket = "iw-bd-demowebcrawler-pdf"
10   acl    = "private"
11   tags = {
12     Application="iw-bd-demowebcrawler"
13   }
14 }
15
16 # main bucket for Lambda function
17 resource "aws_s3_bucket" "iw-bd-demowebcrawler-lambda" {
18   bucket = "iw-bd-demowebcrawler-lambda"
19   acl    = "private"
20   tags = {
21     Application="iw-bd-demowebcrawler"
22   }
23 }
24
25 # DEFINE THE REMOTE REPOSITORY FOR THE .TFSTATE FILE
26 terraform {
27   backend "s3" {
28     bucket = "iw-bd-demowebcrawler-state"
29     key    = "tfstate/terraform.tfstate.json"
```

```
30    region = "eu-central-1"
31   }
32  }
```

lambda.tf

```
1   data "aws_s3_bucket_object" "putSeedInQueue_sourcehash" {
2     bucket = "iw-bd-demowebcrawler-lambda"
3     key    = "putSeedInQueue/putSeedInQueue.zip"
4   }
5   
6   resource "aws_lambda_function" "putSeedInQueue" {
7     function_name    = "putSeedInQueue_v1"
8     role             = "${aws_iam_role.example_lambda.arn}"
9     h, andler         = "main.main"
10    runtime          = "python3.6"
11    s3_bucket = "iw-bd-demowebcrawler-lambda"
12    s3_key = "putSeedInQueue/putSeedInQueue.zip"
13    source_code_hash =
14  "${base64sha256(data.aws_s3_bucket_object.putSeedInQueue_sourcehash.last_
15  modified)}"
16    timeout = 120
17    memory_size = 128
18    tags = {
19      Application="iw-bd-demowebcrawler"
20    }
21  }
22  
23  data "aws_s3_bucket_object" "websiteWatcher_sourcehash" {
24    bucket = "iw-bd-demowebcrawler-lambda"
25    key    = "websiteWatcher/websiteWatcher.zip"
26  }
27  
28  resource "aws_lambda_function" "websiteWatcher" {
29    function_name    = "websiteWatcher_v1"
30    role             = "${aws_iam_role.example_lambda.arn}"
31    h, andler         = "main.lambda_h, andler"
32    runtime          = "python3.6"
33    s3_bucket = "iw-bd-demowebcrawler-lambda"
34    s3_key= "websiteWatcher/websiteWatcher.zip"
35    source_code_hash =
36  "${base64sha256(data.aws_s3_bucket_object.websiteWatcher_sourcehash.last_m
37  odified)}"
38    timeout = 120
39    memory_size = 128
40    tags = {
41      Application="iw-bd-demowebcrawler"
42    }
43  }
44  
```

```
45  data "aws_s3_bucket_object" "hyperlinkCollector_sourcehash" {
46   bucket = "iw-bd-demowebcrawler-lambda"
47   key    = "hyperlinkCollector/hyperlinkCollector.zip"
48  }
49
50  resource "aws_lambda_function" "hyperlinkCollector" {
51   function_name    = "hyperlinkCollector_v1"
52   role           = "${aws_iam_role.example_lambda.arn}"
53   h, andler         = "hyperlinkCollector.main"
54   runtime         = "python3.6"
55   s3_bucket = "iw-bd-demowebcrawler-lambda"
56   s3_key= "hyperlinkCollector/hyperlinkCollector.zip"
57   source_code_hash =
58  "${base64sha256(data.aws_s3_bucket_object.hyperlinkCollector_sourcehash.last
59  _modified)}"
60   timeout = 300
61   memory_size = 128
62   tags = {
63    Application="iw-bd-demowebcrawler"
64   }
65  }
66
67  data "aws_s3_bucket_object" "hyperlinkProcessor_sourcehash" {
68   bucket = "iw-bd-demowebcrawler-lambda"
69   key    = "hyperlinkProcessor/hyperlinkProcessor.zip"
70  }
71
72  resource "aws_lambda_function" "hyperlinkProcessor" {
73   function_name    = "hyperlinkProcessor_v1"
74   role           = "${aws_iam_role.example_lambda.arn}"
75   h, andler         = "hyperlinkProcessor.main"
76   runtime         = "python3.6"
77   s3_bucket = "iw-bd-demowebcrawler-lambda"
78   s3_key= "hyperlinkProcessor/hyperlinkProcessor.zip"
79   source_code_hash =
80  "${base64sha256(data.aws_s3_bucket_object.hyperlinkProcessor_sourcehash.last
81  _modified)}"
82   timeout = 120
83   memory_size = 128
84   tags = {
85    Application="iw-bd-demowebcrawler"
86   }
87  }
```

sqs.tf

```
1  # CLOUDWATCH EVENT TO EXECUTE THE APPLICATION EVERY SUNDAY AT 13:00
2  resource "aws_cloudwatch_event_rule" "everyday_rule" {
3   name              = "start_crawler_trigger"
```

```
4    description       = "schedule events for crawler"
5    depends_on = ["aws_lambda_function.putSeedInQueue"]
6    schedule_expression = "cron(0 0 13 ? * SUN *)"
7    tags ={
8     Application="iw-bd-demowebcrawler"
9    }
10  }
11
12  # ATTACH THE EVENT TO PUTSEEDSFUNCTION
13  resource "aws_cloudwatch_event_target" "start_crawler" {
14    rule =  "${aws_cloudwatch_event_rule.everyday_rule.name}"
15    arn  =   "${aws_lambda_function.putSeedInQueue.arn}"
16    target_id = "putSeedInQueue"
17  }
18
19
20  resource "aws_lambda_permission" "start_crawler_per" {
21    statement_id = "AllowExecutionFromCloudWatch"
22    action = "lambda:InvokeFunction"
23    function_name = "${aws_lambda_function.putSeedInQueue.function_name}"
24    principal = "events.amazonaws.com"
25    source_arn = "${aws_cloudwatch_event_rule.everyday_rule.arn}"
26  }
27
28  # EVENT SOURCE MAPPING BETWEEN SQS, AND LAMBDA QUEUE ->
29  WEBSITEWATCHER
30  resource "aws_lambda_event_source_mapping" "endpoint_seeds_queue" {
31    batch_size        = 1
32    event_source_arn  = "${aws_sqs_queue.seeds_v1.arn}"
33    enabled           = true
34    function_name     =
35  "${aws_lambda_function.websiteWatcher.function_name}"
36  }
37
38  # EVENT SOURCE MAPPING BETWEEN SQS LAMBDA CITYURLQUEUE ->
39  HYPERLINKCOLLECTOR
40  resource "aws_lambda_event_source_mapping" "endpoint_cityURLQueue" {
41    batch_size        = 1
42    event_source_arn  = "${aws_sqs_queue.cityURLqueue_v1.arn}"
43    enabled           = true
44    function_name     =
45  "${aws_lambda_function.hyperlinkCollector.function_name}"
46  }
47
48  # EVENT SOURCE MAPPING BETWEEN SQS LAMBDA PDFURLQUEUE ->
49  HYPERLINKPROCESSOR
50  resource "aws_lambda_event_source_mapping" "endpoint_pdfURLQueue" {
51    batch_size        = 1
52    event_source_arn  = "${aws_sqs_queue.pdfURLqueue.arn}"
```

```
53  enabled         = true
54  function_name    =
55 "${aws_lambda_function.hyperlinkProcessor.function_name}"
56 }
```

trigger.tf

```
1  resource "aws_sqs_queue" "seeds_v1" {
2     name                  = "seeds_v1"
3     tags = {
4       Application="iw-bd-demowebcrawler"
5     }
6     visibility_timeout_seconds = 150
7     message_retention_seconds  = 345600
8     max_message_size           = 262144
9     delay_seconds              = 0
10    receive_wait_time_seconds  = 0
11    redrive_policy             = <<POLICY
12    {
13      "deadLetterTargetArn": "${aws_sqs_queue.seeds-deadletter.arn}",
14      "maxReceiveCount": 5
15    }
16 POLICY
17 }
18
19 resource "aws_sqs_queue" "seeds-deadletter" {
20   name = "seeds_deadletter"
21   max_message_size = 262144 #256kb
22   message_retention_seconds = 1209600 #14 days
23   visibility_timeout_seconds =90
24   receive_wait_time_seconds = 20
25   tags = {
26     Application="iw-bd-demowebcrawler"
27   }
28 }
29
30 resource "aws_sqs_queue" "cityURLqueue_v1" {
31    name                  = "cityURLqueue_v1"
32    tags = {
33      Application="iw-bd-demowebcrawler"
34    }
35    visibility_timeout_seconds = 180
36    message_retention_seconds  = 345600
37    max_message_size           = 262144
38    delay_seconds              = 0
39    receive_wait_time_seconds  = 0
40    redrive_policy             =  <<POLICY
41    {
42      "deadLetterTargetArn": "${aws_sqs_queue.cityURLqueue-deadletter.arn}",
43      "maxReceiveCount": 3
```

```
44        }
45      POLICY
46      }
47
48      resource "aws_sqs_queue" "cityURLqueue-deadletter" {
49        name = "cityURLqueue_deadletter"
50        max_message_size = 262144 #256kb
51        message_retention_seconds = 1209600 #14 days
52        visibility_timeout_seconds = 300
53        receive_wait_time_seconds = 20
54        tags = {
55          Application="iw-bd-demowebcrawler"
56        }
57      }
58
59      resource "aws_sqs_queue" "pdfURLqueue" {
60          name                      = "pdfURLqueue_v1"
61          tags = {
62            Application="iw-bd-demowebcrawler"
63          }
64          visibility_timeout_seconds = 150
65          message_retention_seconds  = 345600
66          max_message_size           = 262144
67          delay_seconds              = 0
68          receive_wait_time_seconds  = 0
69          redrive_policy             =  <<POLICY
70            {
71              "deadLetterTargetArn": "${aws_sqs_queue.pdfURLqueue-deadletter.arn}",
72              "maxReceiveCount": 200
73            }
74        POLICY
75      }
76
77      resource "aws_sqs_queue" "pdfURLqueue-deadletter" {
78        name = "pdfURLqueue_deadletter"
79        max_message_size = 262144 #256kb
80        message_retention_seconds = 1209600 #14 days
81        visibility_timeout_seconds = 300
82        receive_wait_time_seconds = 20
83        tags = {
84          Application="iw-bd-demowebcrawler"
85        }
86      }
```

Appendix H
.gitlab-ci.yml

```
1  variables:
2    AWS_ACCESS_KEY_ID: "xxxxxxx"
3    AWS_SECRET_ACCESS_KEY: "xxxxx"
4
5  image:
6    name: hashicorp/terraform:latest
7    entrypoint:
8      - '/usr/bin/env'
9      - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'
10
11 before_script:
12   - rm -rf .terraform
13   - terraform --version
14   - mkdir -p ./creds
15   - echo $SERVICEACCOUNT | base64 -d > ./creds/serviceaccount.json
16   - terraform init
17
18 stages:
19   - validate
20   - plan
21   - apply
22
23 validate:
24   stage: validate
25   script:
26     - echo "Validate"
27     - terraform validate
28 plan:
29   stage: plan
30   script:
31     - echo "Planing"
32     - terraform plan -out "planfile"
33   dependencies:
34     - validate
35   artifacts:
36     paths:
37       - planfile
38
39 apply:
40   stage: apply
41   script:
42     - echo "Applying"
43     - terraform apply -input=false "planfile"
44   dependencies:
45     - plan
```

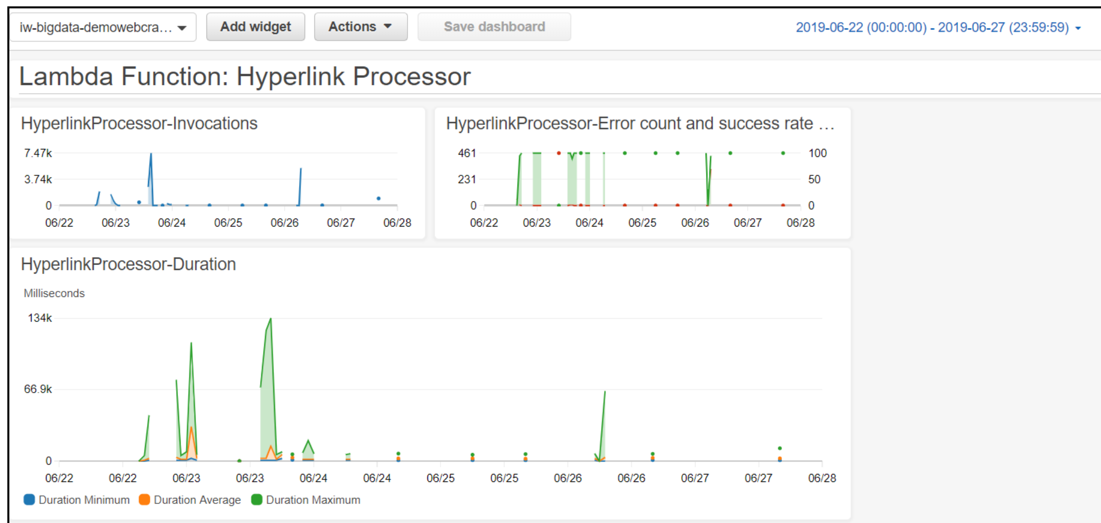
Appendix I:

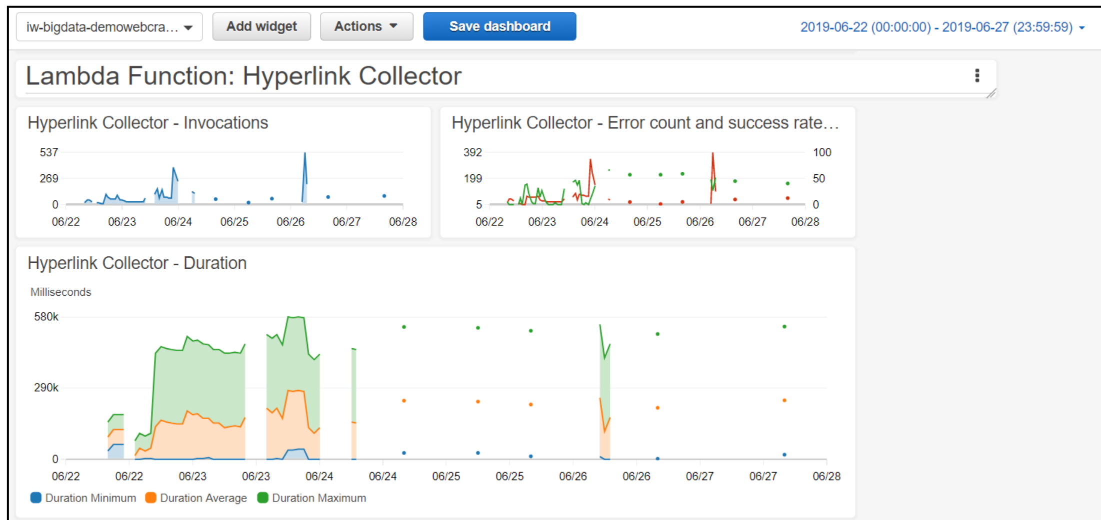
.gitlab-ci.yml

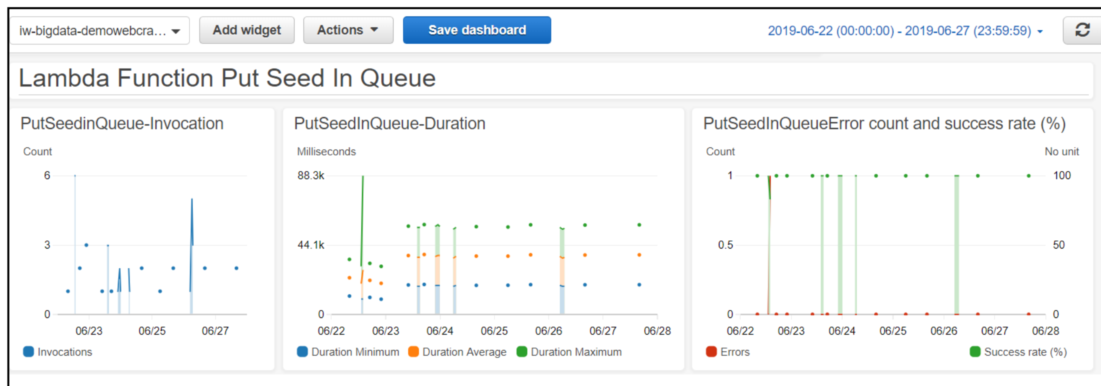
```
1  variables:
2    AWS_DEFAULT_REGION: eu-central-1 # The region of our S3 bucket
3    BUCKET_NAME: iw-bd-demowebcrawler-lambda       # Your bucket name
4    FUNCTION_NAME: hyperlinkCollector
5    AWS_ACCESS_KEY_ID: "xxxxxxxx"
6    AWS_SECRET_ACCESS_KEY: "xxxxxxxxx"
7
8  image:
9    name: hoanguyen95/terraform_python:latest # Using custom image with terraform, and
10 python installed on Ubuntu 16.04
11   entrypoint:
12     - '/usr/bin/env'
13     - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'
14
15 stages:
16   - build
17   - package
18   - deploy
19
20 build:
21   stage: build
22   script:
23     - echo "Building"
24     - pip3 install -r requirements.txt -t src/
25   artifacts:
26     paths:
27       - src/
28
29 package:
30   stage: package
31   script:
32     - cd src
33     - zip -r src.zip *
34     - echo "current dir"
35     - ls
36   artifacts:
37     paths:
38       - src/
39
40 deploy:
41   only:
42     variables:
43       - $CI_COMMIT_MESSAGE =~ /\[commits3\]/
44   stage: deploy
45   before_script:
46     - pip3 install awscli
```

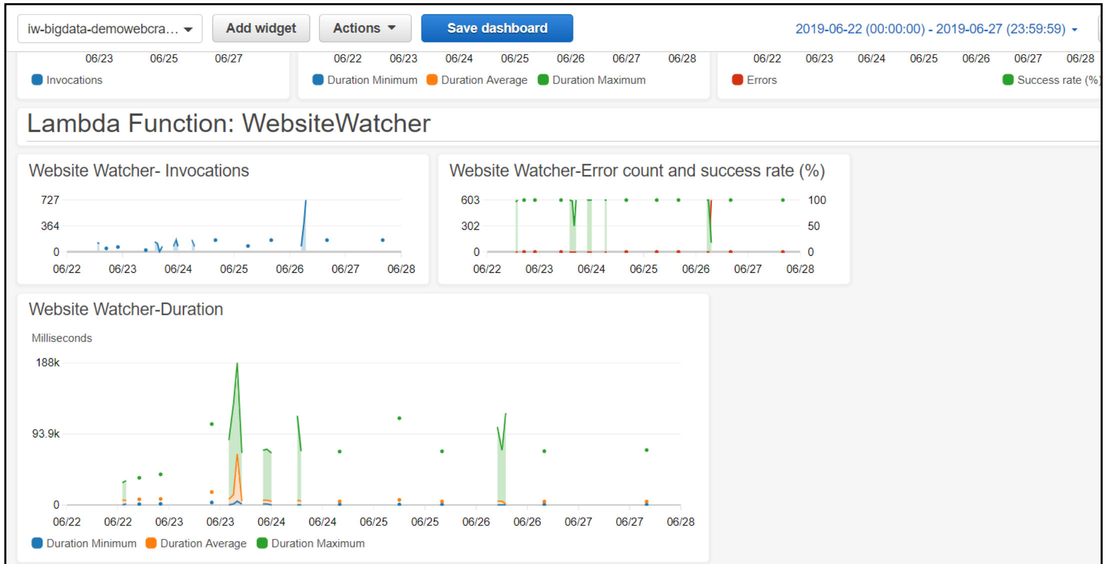
```
47     script:
48       - aws s3 cp ./src/src.zip
49   s3://${BUCKET_NAME}/${FUNCTION_NAME}/${FUNCTION_NAME}.zip
50       # redeploy AWS Infrastructure
51       - "curl --request POST --form token='xxxxxx' --form ref=master
52   https://gitlab.com/api/v4/projects/12983597/trigger/pipeline"
53     environment:
54     dependencies:
55       - package
56
```

Appendix K : AWS CloudWatch Dashboard for AWS Lambda Functions








iw-bigdata-demowebcra...
Add widget
Actions
Save dashboard
2019-06-22 (00:00:00) - 2019-06-27 (23:59:59)
Invocations
Duration Minimum
Duration Average
Duration Maximum
Errors
Success rate (%)
Lambda Function: WebsiteWatcher
Website Watcher- Invocations
Website Watcher-Error count and success rate (%)
Website Watcher-Duration
Milliseconds
188k
93.9k
Duration Minimum
Duration Average
Duration Maximum