Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Continuous Scraping and its Management in an Industry Setting

Author: Chih-Chieh Lin (2700266)

1st supervisor:	Dr. Adam S.Z. Belloum	University of Amsterdam
daily supervisor:	Wuyou Liu	Dashmote B.V.
2nd reader:	Zhiming Zhao	University of Amsterdam

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

July 12, 2022

"I am the master of my fate, I am the captain of my soul"

from Invictus*, by William Ernest Henley*

Abstract

Web data scraping is applied in various fields and in different purposes. Reducing the human intervention through scraping workflow is critical to improve the scraping efficiency. The goal of this paper is to build an continuous (automatic) scraping pipeline that includes schedulers triggering the scraping pipeline automatically and also relevant features for achieving less human monitoring and actions. This application or pipeline is deployed on the cloud using Amazon Web Services. Moreover, for the maintenance, developing and operations, we design and implement a CI/CD pipeline by Github Actions to make the whole scraping pipeline more automatic. Our work automates the scraping pipeline with scheduler, and automates the deployment procedure through CI/CD pipeline which not only set a foundation for integration work afterward but also promote the collaboration among team members.

Keywords: *Web Scraping, Continuous Integration, Continuous Deployment, Workflow Orchestration*

Acknowledgements

This project was finished during my internship at Dashmote among Data Platform team and Sourcing team. This internship not only horn my technical skills but also gave the chance to investigate and explore in the academic areas. I would like to express my appreciation to Dashmote to give me the opportunity to join them to do this project. I am much obliged to my daily supervisor, Wuyou Liu for her guidance, feedback and all the support she gave. Also, I do appreciate the supports from my colleagues, Fucheng Zhen at Sourcing team and Adrián Villanueva Martínez at Data Platform team. I also want to express the depth of my gratitude to Dr.Adam S.Z. Belloum for all the guidance he gave me and the advice on finishing this project. Moreover, I am grateful to Dr. Zhiming Zhao for his valuable feedback and insights for this thesis.

Last but not least, thanks to my family and my friends for their trust and support.

Contents

List of Figures

List of Tables

Glossary

AWS	Amazon Web Services
CI/CD	The core of a DevOps methodology which refers to continuous integration, continuous delivery and continuous deployment
DevOps	A set of practices that combines software development (Dev) and IT operations (Ops)
Docker	A set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers
ECR	Elastic Container Registry: a fully managed Docker container registry service provided by AWS, which makes it easy to store, share, and deploy container images.
ECS	Elastic Container Service: a fully managed container orchestration service provided by AWS, which helps users easily deploy, manage, and scale containerized applications.
EventBridge	A serverless event bus service that you can use to connect your applications with data from a variety of sources.
Github Actions	GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows user to automate your build, test, and deployment pipeline.
Lambda Function	A compute service that lets users run code without provisioning or managing servers.
Pull Request	an event that takes place in software development when a contributor/developer is ready to begin the process of merging new code changes with the main project repository.
S3	Simple Storage Service (S3) is a service offered by Amazon Web Services (AWS) that provides object storage through a web service interface.
YAML	A data serialization language that is often used for writing configuration files

1 Introduction

Living in a data driven world nowadays, we have no choice but to consume and generate great amount of data every day. Zhu et al.(5) defined *Data Explosion* in their work as “large scale of data is generated rapidly, and stored in computer systems.” In order to use this large scale of data efficiently, developers need to develop new or advanced method to explore, extract, and process the data.

The data created across the web is abundant and useful for various purposes, e.g., academic, marketing, scientific purposes. People might intend to collect and analyze data from multiple websites, so that **Web Data Scraping** have become the crucial technologies for the purpose of extracting data from the web. However, web scraping can be time-consuming and resource-consuming, especially when it is performed manually (6). Therefore, a web scraper which simulates human browsing the web to automatically extract data are being activity research and engineered.

As a company providing data product and market strategy for global beverage brand clients like Red-Bull and Coca-Cola, *Dashmote*[1] must apply the efficient methods to extract data in order to carry out the data processing and analytic afterward so that they are able to deliver products on time. The way Dashmote obtain the associated data is to scrape it from the online food ordering and delivery platform such as Ubereat, Deliveroo, Just eat. It should be an efficient method with automated manner to some extent.

This thesis aims to take a *continuous* (which means we get rid of on-demand model, and pursue a continued, automated, scheduled, stable process) scraping project at Dashmote, which upgrade the automatic level of web scraping, and design the CI/ CD pipeline for such scraping program deployment. We identified the problem and the challenges in the web scraping field and also during the developing of the automatic workflow. Moreover, it

[1] https://dashmote.com/

help us scale the scraping procedure by deploying services on AWS automatically, which also increases the potential of integration and extendability of our work.

This thesis is structured as follows. I begin by delineating the background information with technical and business perspective, followed by the research questions in Chapter 2. Next, I describe relevant technologies and associated academic papers in the fields of web scraping and automatic workflow orchestration in Chapter 3. Next, the proposed continuous scraping pipeline (a scheduled automation workflow) and the CI/ CD pipeline architecture for scraper program are introduced in Chapter 4. Afterward, the reflection including the comparison and discussion, and the future work of this project are presented om Chapter 5. The summary concluding this work is provided in Chapter 6. The terminologies in this paper are explained in the Glossary and the implementation details are listed in Appendix.

2 Background

2.1 Problem Statement

Web data scraping is applied in various fields and for different purposes, such as stock price monitoring, flight ticket price comparing, daily news summarizing, and academic purpose. However, there are plenty of problems we may confront during developing and maintaining tools for scraping. Glez-Peña et al. (7) discussed several potential challenges and problems for web scraping tools development. The comprehensive relevant detail for the challenges are introduced in the next section. In this section, several problems from technical perspective along with the ones from the business perspective are emphasized here in order to understand our motivation of this work.

2.1.1 Technical Perspective

- Manual screening is time-consuming.
- Manual running scraper program increases the human cost.
- Scraper program run in the deployed cluster may not be scalable when dealing with large scale data.

These problems highlights the difficulty the engineers were previously confronted with at the company where I was doing the internship.

The need for a “manual” intervention requires the availability of engineers to monitor and run the scraper program in order to make sure that the data will be available on time. That is to say, the engineers should check the scraper process periodically. Once the process has finished, the scraping data (result) should be inspected by the engineers to confirm if the data is valid for the following processing. Although, the scraper program is run by a click action or a command entered by the engineer (partly automatically), the whole procedure consumes human concentration and valuable engineer resource. Moreover, these human intervention should be avoidable by other automatic strategies.

As for the scalability issue, scraping source and condition may be different which depends on the client requirements. For instance, the platforms and countries which are asked to scrap may be differed according to different clients. As a result, the scraping performance vary in different requirements. To be more specific, the scale of data is affected by the clients' requirements and will influence scraping performance. From the scalability perspective, the scraping process can be able to smoothly run most cases with small or medium amount of data, however, it may fail when the cluster encounter large scale data.

2.1.2 Business Perspective

- Wasting time on screening reduces the throughput of data processing and delivery.
- Intensive human manual care and action for the scraping procedure may decrease the potential profits for the company, since releasing engineer resources from manual care can make them doing other tasks to increase the productivity of the whole company.
- Deploying and Maintaining an exclusive cluster for scraping purpose is expensive, especially in the condition of other data processing and modeling purposes are deploying on the AWS[1].

Each problem or situation in business consideration listed above can be relevant to the one engineers met in the technical part. Moreover, these difficulties affected the frequencies of delivering the data products to the clients. Therefore, the solution to solve these dilemma is to have stable deliveries of products or even to have more frequent deliveries. For instance, the company used to deliver the products to one of clients once a month; we expected to deliver the product twice a month to satisfy the growing demand of customers. That is to say, we must scrap the data from the resources twice in a month which need more efforts (manual care) by engineer team. However, it is hard to achieve the goal of having more frequent deliveries with an inefficient scraping process.

To summarize, the inefficiency of delivering data products caused by the problems listed above makes more frequent refresh for deliveries unachievable. Therefore, to satisfy the request from customer and also to have a stable throughput, a more generic, automatic and continuous scraping method is mandatory.

[1]Amazon Web Services: https://aws.amazon.com/about-aws/

2.2 Research Questions

According to the challenges and problem we encountered in developing the continuous scraping pipelines, three research questions are stated here to clarify the direction of this work. The research questions and rational motivations of each question are claimed as follows.

- RQ1: *What are the challenges in developing and maintaining continuous pipeline for web dat a scraping?*
  - I desire to figure out the challenges we might encounter in developing our solution and also in maintaining pipeline. I try to summarize the potential and actual problems during our developing and maintaining procedure.
- RQ2: *What are the challenges for web data scraping, and what are the potential solutions for them?*
  - In addition to continuous pipeline for scraping procedure, I try to focus on scraping part itself and further explore to determine the corresponded problems and challenges.
  - The possible challenges caused by the increase of data volumes, heterogeneous resources, legal and policy issues.

3

Related Work

In this chapter, the state-of-the-art tools, relevant technology and associated academic paper are discussed. First, the related work for web data scraping are provided. Then, the potential tools and combination of cloud services for fulfilling our automation workflow are considered.

3.1 Web Data Scraping

Since the main task of this project is to develop an automatic workflow which scrapes the data through the website, the usage, the knowledge and the background in the relevant field must be discussed. In general, the main purpose of all web scraping is obviously - extracting the data we are interested in from the web, and transform them into understandable structure like database, spreadsheets, json etc (8). Undoubtedly, the extracted data in certain field that experts or researchers are interested in is usable and valuable. As a result, web data scraping can be applied in various field, e.g., online price comparison, website change detecting, government data gathering, research, brand monitoring.

In this section, we begin by providing the definition of web scraping to clarify the scope of field. Next, we describe several frameworks and tools that are state-of-the-art and used by current companies or developers. Then, the challenge and the difficulties of web scraping are delineated. Finally, the ethics and the social consideration for web scraping are described in the last part of this section.

3.1.1 Definition

The outsider or even the developer outside of this field may be confused by the terms "*Web Crawling*" and "*Web Scraping*". The answer can be shortly illustrated[1] - web crawling is to find or explore URLs or links on the web, while web scraping is to extract the data from one or several website (9).

Broucke and Baesens (10) describe that the difference between these two terms are relatively vague since lots of programmers use both terms interchangeably. They define "crawler" as a program which navigate web pages with or even without purpose to explore the contents offered by a site. Also, "web scraping" for them is referred to create agents to download, parse and arrange data in an automated way.

Massimino considered *web crawling* and *information scraping* as two distinct methodologies: the former is the automated navigation of a series of internet-based references, while the latter is the automated procedure of processing and transforming semi-structured data into a structured format data (11).

To summarize, we can differentiate these two terms by their different purposes and outputs: web crawling navigates web page with or without specific end goal but aims to find URLs as the main output elements, while we scraping aims to extract data from specific, known, and target websites to store or transform it into structured format.

3.1.2 Web Scraping Structure

Before broadly discussing the nowadays technique for web data scraping, we first provide how web scraping works by simplifying into 3 steps as follows (7, 12).

1. **Site access**: This first step refers to the communication between web scraper (request) and the host (i.e. response from web server) through the HTTP protocol.

2. **HTML parsing & content extracting**: The web scraper will extract the certain content after retrieving the HTML document. There are plenty of tools and methods can be applied to complete the extraction, e.g., selector-based languages, regular expression matching, DOM[2] structure based parsing libraries, programming logic or machine learning approaches.

3. **Output building**: The aim of this final stage is to transform the extracted web content into structured format so that is valid for subsequent processing and storage.

[1]https://www.zyte.com/learn/difference-between-web-scraping-and-web-crawling/
[2]Document Object Model

3.1.3 Web Scraping Technique

Here we describe the tools and frameworks for web data scraping. The researchers classify them in different ways. Massimino classified it into three types: Fully custom applications, Semicustom programs and Commercial sotware services (11). Similarly, Glez-Peña *et al.* structured them into three categories: libraries for genereal-purpose programming language, frameworks, desktop based environments (7). Diouf *et al.* proposed two categories: Ready-made Tools (which include *Browser extensions* and *Software and Platforms*) and The libraries of programming languages (6). We can easily claim that the standards of categorizing mainly depend on whom the target customers of the tools are, the customizability, and programming skill level required. Also, we provide three classified types with examples as follows in the Table .

	Description	**Examples**
Libraries	This category is the most primitive and with highest customizability of the three. Plenty of languages provides libraries for web scraping, e.g., NodeJS, Java, PHP, Python.	Python: *BeautifulSoup*[1] Java: *jsoup*[2] NodeJS: *Apify*[3] PHP: *Goutte*[4]
Framework	The framework is more integrative solution comparing to libraries. That is to say, we must integrate several different libraries for various tasks, e.g., access web sites, parse HTML, extract contents. As a result, this type of tools integrates these tasks and packages them into a framework.	Python: *scrapy*[5] Java: *jaunt*[6] NodeJS: *Cheerio*[7] Ruby: *Kimurai*[8]
Ready-used Tools	We divided this type into two sub-categories: *Browser extension* and *Software.* These ready-used tools are well-developed, yet with limited extendability and customizability.	Browser extension: *Web Scraper*[9], *Data Scraper*[10], *Dexi.io*[11] Software: *Import.io*, *Mozenda web*, *WebExtractor360*

Table 3.1: Categories of Web Scraping Technique

3.1.4 Challenge of Web Scraping

In the stage of constructing web data scraper or executing scraping, plenty of technical challenges and potential issues are encountered and solved. The purpose for this section is not to go in the details of each difficulty but it is just limited to enumerate possible challenges and have an overview of them.

- **Changeable Web Site Structure & Dynamic Content**:
  Once the web site owner changes the structure of web page due to improving user experience or publishing new features, the web structure may be changed. In this situation, the old scraper may not work for the updated web pages. Also, some web pages generate the content dynamically using JavaScript (or AJAX[1]) may also cause problems (12, 13).

- **Anti Scraping**:
  Web sites' owners apply several methods to detect and block the robots for scraping, e.g., CAPTCHA[2], reversed Turing test, IP blocking(due to large amount of request, or intend to restrict the requests for each single one IP), so that they are able to protect their valuable resource or data at their sites.

- **Login Requirement**:
  Some specific data or protected information is only visible by the certain roles which login to the accounts with the credential of accessibility.

- **Honeypot Traps**:
  This Honeypot is also an useful tool for web scraper detection and prevention which the same as the anti scraper. However, we list it here to emphasize its influence. A honeypot is able to trap crawler or scraper in an "spider trap" which may be a infinite deep directory tree or an inescapable loop collecting useless data (14, 15).

[1]https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[2]https://jsoup.org/
[3]https://www.npmjs.com/package/apify
[4]https://github.com/FriendsOfPHP/Goutte
[5]https://scrapy.org/
[6]https://jaunt-api.com/
[7]https://cheerio.js.org/
[8]https://github.com/vifreefly/kimuraframework
[9]https://webscraper.io/
[10]https://dataminer.io/
[11]https://www.dexi.io/
[1]Asynchronous JavaScript And XML: https://www.w3schools.com/js/js_ajax_intro.asp
[2]Completely Automated Public Turing test to tell Computers and Humans Apart

- **Trade-off between Accurate Performance and Automation**:
  Although scraping web data in an high level automated manner is efficient, human's feedback is crucial for increasing the accuracy of extracted data. As a result, the balance between automation and human intervention must be considered when developing the scraper so that the quality and accuracy of extracted data can be ensured (16).

- **Scalability**:
  A scalable and reliable scraper is difficult to develop since the extracting procedure may fall by several patterns in the web site (17). Glez-Peña *et al* also mentioned that dealing with large volumes of information through scraping is challenging (7).

  Moreover, it is sensible to develop scraper for different target data or information. For instance, for an e-commerce platform, the tasks can be simply separated by: discovering product, and extracting product pages (18). As a result, numerous and various types of scrapers are developed. The strategy to manage this numerous scrapers to make the extracting procedure more scalable can be challenging.

  As the complexity increasing in the procedure of web data scraping, cloud computing services are suggested to use. Distributing the load among computational units makes a more efficient resource management procedure, and even more scalable and reliable (16).

- **Heterogeneity**: For the heterogeneity of data format or data type, with large number of pages using JavaScript and CSS[1], the increasing of heterogeneity of web pages can cause the challenges for scraping (12).

  For any of resource that developers and researchers are looking for in the web, they may encounter lots of heterogeneous and independent resources (7).

3.1.5 Ethics Consideration

The extracted data from web scraping may be influential or even harmful for related individuals or even society. For instance, the data which is related people's or companies' privacy, especially in medical domain and financial domain. As a result, the ethics, social, and legal consideration should be satisfied when using web scraping.

The *Web Ethics* concept had been proposed when Web technique evolved. Eichmann organized the web ethic for different roles e.g. service agent, user agent. Also, web spider

[1]Cascading Style Sheets: https://developer.mozilla.org/en-US/docs/Web/CSS

was included in his works; several guidelines and exclusion for each role were provided, even in the nineties of the last century (19).

Massimino provided the social considerations from several perspectives for the researchers who intend to utilize the web scraping to extract data - A systematic third-party review for use of second-hand data is missing in practice, while self-discipline is common nowadays scientific research. Also, when the websites expose certain signs (e.g., Spider traps, authorization, CAPTCHAs) that they are not welcome for the scraper, the researchers should consider any of these explicit disapproval of web scraping and obtain the consent from the host of the site. Moreover, the web site performance or other functionality may be affected by the scraper which requests at high frequency. Therefore, Massimino suggested the researchers to restrict their request frequency to no more than two requests per second (11).

Upadhyay *et al* also considered legal consideration and scraping ethics in web data extraction field. They regard web scraping as privilege but not a right. The operator of web scraping tools must conform to the term of use provided by the web site host. Also, they suggested not to scrap encrypted data but only scrap for the public available data so that avoid the risk of violating the law. In other words, it is safe to assume that scraping the data from the site requiring login and password for accessibility is illegal (12).

To summarize, we should always take ethics, law, society into consideration, and think twice before scraping any of website. As Mitchell described, the script or the program of scraping technique should not be runs against every site you can find (17).

3.2 Schedulable Recurring Automation Workflows Tools

In this section, we discuss several potential tools that are able to solve our problem by creating automatic triggered workflow running our scraping programs with cloud computing services. Moreover, the schedule feature for periodically executing the scraper is required. We first consider the tools provided by three different cloud computing service providers, i.e. Amazon Web Services (AWS), Microsoft Azure [1], Google Cloud Platform (GCP) [2].

3.2.1 Automation Workflow Structure using Cloud Computing Services

Cloud computing services platforms provide various types of services and functions that can be utilized in our developing of automation workflow structure. We aim to construct

[1]https://docs.microsoft.com/en-us/azure
[2]https://cloud.google.com/gcp

a schedulable automated workflow that can run different scrapers repeatedly in a fixed period. In Table 3.2, we list the requirements should be fulfilled; and associated them with the tools that support them in different cloud computing platform respectively.

Requirement	**Relevant Tools**
A scheduler to trigger the workflow periodically	– AWS: EventBridge[1] – Azure: Timer Trigger[2] – GCP: Cloud Scheduler[3]
A serverless service that connects and extends cloud services, deploys the containers	– AWS: Lambda Function[4] – Azure: Azure Functions[5] – GCP: Cloud Functions[6]
A orchestration service to manage and run the scraper containers	– AWS: ECS[7] – Azure: Azure Container Instances[8] – GCP: Google Kubernetes Engine[9] (GKE)
A tool that can used for monitoring the scraping procedure by presenting the logs	– AWS: CloudWatch[10] – Azure: Azure Container Instances[11] – GCP: Cloud Logging[12]

Table 3.2: Relevant Tools for Schedulable Automated Workflow

Based on the tools and services listed above, we are able to construct the automated workflow using different platforms. We provide the workflow structure of each platform in Figure 3.1 for AWS, Figure 3.2 for Azure, Figure 3.3 for GCP. Each service provided by the platforms may match to another equivalent or similar service in another platform (20). The structures for AWS and Azure are similar or even identical while GCP's shows slight

[1]https://docs.aws.amazon.com/eventbridge/
[2]https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-scheduled-function
[3]https://cloud.google.com/scheduler
[4]https://aws.amazon.com/lambda/
[5]https://docs.microsoft.com/en-us/azure/azure-functions/
[6]https://cloud.google.com/functions
[7]https://aws.amazon.com/ecs
[8]https://azure.microsoft.com/en-us/services/container-instances
[9]https://cloud.google.com/kubernetes-engine
[10]https://aws.amazon.com/cloudwatch/
[11]https://azure.microsoft.com/en-us/services/container-instances
[12]https://cloud.google.com/logging

different. The Cloud Functions triggered by scheduler in GCP should be invoked by Pub/Sub[1] service.

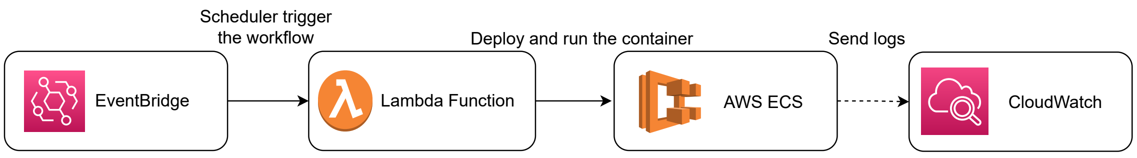


Figure 3.1: Amazon Web Services Automation Workflow

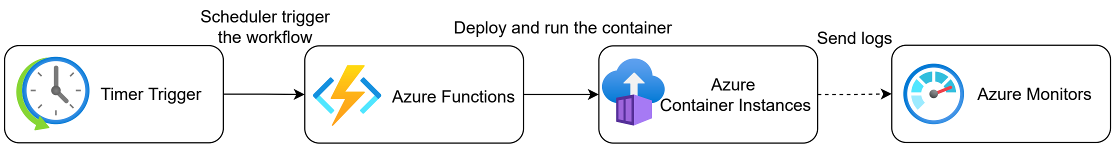

Scheduler trigger the workflow
Deploy and run the container
Send logs
Timer Trigger
Azure Functions
Azure Container Instances
Azure Monitors

Figure 3.2: Microsoft Azure Automation Workflow

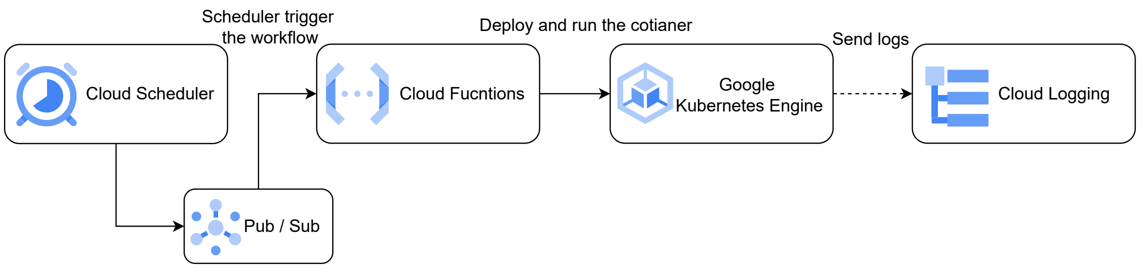


Figure 3.3: Google Cloud Platform Automation Workflow

3.2.2 Workflow Orchestration

In the Section 3.2.1, we considered the structures of the workflow to be a lighter and simpler design. Since the core program, the scraper, run in container, the most important point here is to develop the workflow in a fast and efficient way so that it can be put into production as soon as possible.

However, if we consider integrating other procedures (e.g. the quality assurance, data completeness test, subsequent data processing) into our workflow structure, there are also other advanced workflow application provided by cloud service providers.

[1]https://cloud.google.com/scheduler/docs/tut-pub-sub

For instance, AWS Step Functions[1], Azure Logic Apps[2], Workflows[3] provide users with such platform which is able to connect, integrate different cloud services in a automated workflow by writing less code and focusing on the business logic and functionality (21). Moreover, these workflow orchestration service free developers from taking care of maintaining, managing, scaling and monitoring solutions created with these services (22).

Moreover, there are other workflow orchestration tools and platforms that providing services in the marketplace nowadays, such as Luigi[4], Airflow[5], Kubeflow[6], MLflow[7], Argo[8]. These orchestration tools allow us to orchestrate individual steps, schedule tasks and manage dependency resolution. The workflows defined by these tools describe the dependencies among a set of tasks which encodes in a Directed Acyclic Graph (DAG) (23, 24).

In addition to the workflow tools for business purposes which are widely utilized for business data processing, several tools have been developed for scientific workflow which are mainly used for modeling and running scientific experiments, e.g. Montage[9], Pegasus[10], Makeflow[11], Kepler[12] (25, 26, 27).

Finally, there are also plenty challenges for workflow orchestration, e.g., *architectural challenges* which ask workflow management system support flexibility, extensibility and portability architecturally, *integration challenges* which focus on the issue when executing workflow over cloud resources, *computing challenges* which emphasize the decision of cloud instance type (28).

[1]https://aws.amazon.com/step-functions/
[2]https://docs.microsoft.com/en-us/azure/logic-apps
[3]https://cloud.google.com/workflows
[4]https://luigi.readthedocs.io/en/stable/
[5]https://airflow.apache.org/
[6]https://www.kubeflow.org/
[7]https://mlflow.org/
[8]https://argoproj.github.io/argo-workflows/
[9]http://montage.ipac.caltech.edu/docs/grid.html
[10]https://pegasus.isi.edu/
[11]https://cctools.readthedocs.io/en/latest/makeflow/
[12]https://kepler-project.org/

4

Project Implementation

In this chapter, the comprehensive implementation procedure and details for this project are described. We first illustrate the concrete requirements according to the problem statements in the Section 2.2, so that we can have an overview and expected results of this project. After clarifying the requirements base on the problem, the continuous pipeline structure will be illustrated. The service and the components using in the pipeline provided by AWS are also involved in the subsection. Followed by the continuous pipeline architecture, the scraper program and the relevant features for it will be introduced. The CI/CD pipeline developed by Github Actions will be illustrated at the last part of this chapter. The idea of creating and maintaining an organized CI/CD pipeline is to make whole solution become more complete and automatic.

4.1 Project Requirements

According to problem statement described before in Section 2.1, we have an overview of challenges the engineers and the company are confronted with. Moreover, we summarized the problems and provided the main goal of this project. That is, to have a more generic, automatic and continuous scraping pipeline. Here, we come up with more concrete ideas and requirements based on the main purpose of this work and the problem stated in Section 2.1. We list the requirements which satisfy the potential solution for problems proposed in both technical and business perspective.

- A stable and extendable scraping pipeline using automatic procedure which has following characteristics:
  - A reliable scheduler to trigger the specific scraper

  - A extendable scraper program to scrape the web data
  - A function of notification to inform engineer so that it can reduce the time of manual screening
  - A function to validate the result of scraping data so that it can reduce the effort of inspection by engineers
  - A scalable cluster or service to run the scraper so that it can avoid unexpectable crash
  - A well-organized storage strategy for storing the scraping result data
  - A CI/CD pipeline may be included in the workflow in order to reduce human intervention

4.2 Continuous Scraping Pipeline Design

As the requirements for this continuous scraping pipeline were clearly clarified in last section, we are now coming up with concrete architecture of the pipeline. In this section, the pipeline structure with its development procedure and mechanism will be first displayed and illustrated, followed by the rules and mechanism of schedulers for triggering pipeline. Afterwards, since we utilize lots of various web services provided by AWS[1] in order to complete our work, the developing details, operating mechanism and the knowledge for the services we choose in AWS will be well-described in the last part of this section.

4.2.1 Pipeline structure

Figure 4.1 displays the continuous scraping pipeline structure. The right part of the Figure 4.1 can be seen as the main continuous scraping pipeline while the left part can be considered as the main development procedure. We first describe the development procedure except for the main scraper and its features (which are illustrated in the section 4.3). Then the continuous scraping pipeline mechanism is described.

- **Development procedure (Steps)**

  1. **Developing scraper:** First of all, the main scraper program and its feature will be included and developed in the Github repo called *conso-scraper*.
  2. **Linting & unit tests:** To guarantee the code format and quality, linting checks and unit tests are essential before any further development.

[1]https://aws.amazon.com/about-aws/

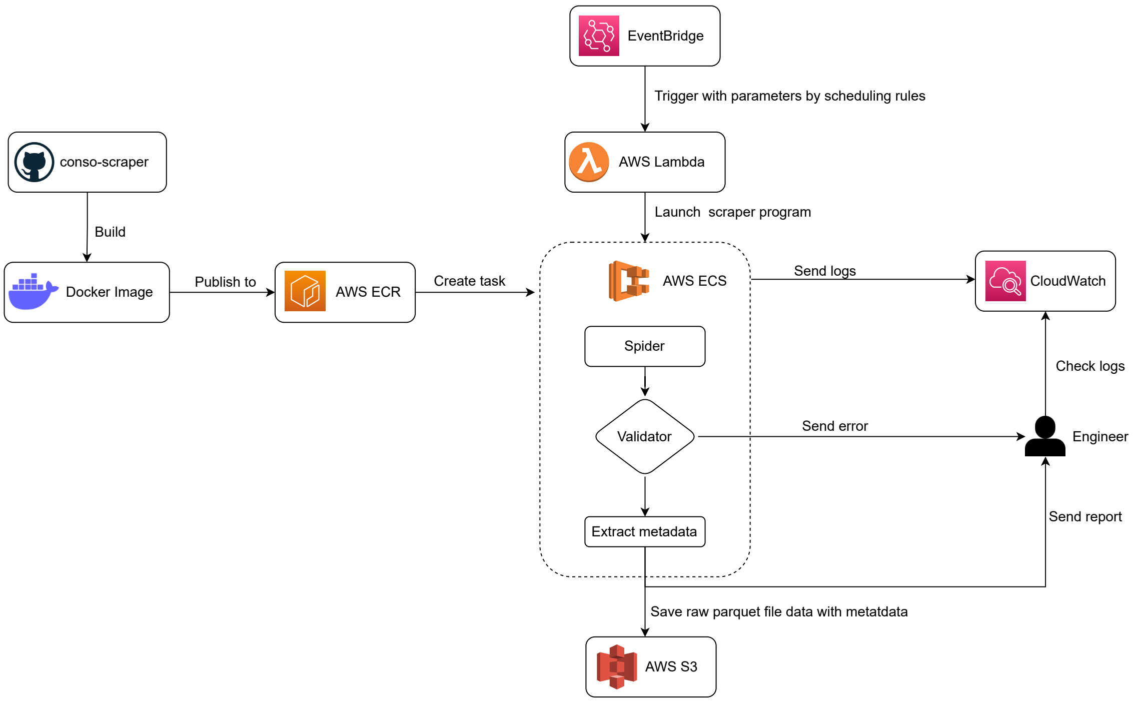


Figure 4.1: Continuous Scraping Pipeline Diagram

3. **Building Docker image:** After developing the functions and features in the repo, the Docker image will be built through *Dockerfile*. The image should contain the same environment within functions and feature as the program in the repo.
4. **Local test for Docker:** Before pushing our Docker image to AWS service, we must test the Docker image and container to make sure the container run successfully and obtain the proper result with expected log.
5. **Uploading image to ECR:** After local test for Docker container, built image will be uploaded to ECR (container registry) by running script.
6. **Creating ECS task definition:** To run the container in ECS, the ECS task definition is required. The specific mechanism and the usage details will be described in section 4.2.3.
7. **Doing cloud test (ECS):** Similar to step 4, we also want to make sure the scraper will be able to run successfully in ECS container with corresponded log in AWS CloudWatch. Therefore, testing the container in ECS before putting it into production is mandatory.

8. **Setting up schedulers:** The schedulers can be separated into two parts: EventBridge(rules to trigger the pipeline periodically) and Lambda Function(receive parameters from EventBridge and launch scraper container in ECS). The details of mechanism of schedulers will be described in Section 4.2.2.

- **Pipeline workflow (Automatic triggered workflow)**

1. **Triggering the pipeline:** EventBridge which consist of event schedule with rules will trigger the pipeline by passing the parameters to Lambda Function.
2. **Launching container in ECS:** Once receiving the parameters sent from EventBridge, Lambda Function will launch the corresponded scraper programs deployed in ECS.
3. **Running the scraper in container:** The scraper program is run in the container in ECS. We called the main scraper program as *Spider*.
4. **Sending the logs:** During the running process of scraper container in ECS, the ECS is sending the logs to CloudWatch in the mean time.
5. **Validating results:** Once the spider finish the scraping procedure, a validating process will be run in order to recognize whether the result of scraping is valid and can be used for further following processing.
6. **Sending report or error:** The validator determined if the results are valid, and then a report or error message will be sent to Slack channel to notice the engineers.
7. **Checking Logs:** If the validating report or error raise issue of the results, the engineers must take actions on checking the logs during the scraping procedure which could be investigated through ÇloudWatch.
8. **Extracting metadata:** The raw scarping result data must be stored with a set of metadata in order to describe it. Therefore, before storing the raw result data, we must extract or generate the corresponding metadata.
9. **Storing result in S3:** The final step is to upload the scraping result data to S3 buckets with metadata describing it. Also, the storing path must be specified by organizing rules so that the data will be able to extract easily.

4.2.2 Schedule rules

The scheduler is implemented by the service in AWS, EventBridge, which is a serverless event bus that can be used to connect applications with data from various sources. That is to say, EventBridge receives events, and then applies rules to route to the targets. In our case, we set up rules in EventBridge according to different resources. That is, countries, food deliver platforms, restaurants or details, are the categories for rules for different scrapers. Table 4.1 displays 2 examples of rules set up in EventBridge.

Object	**Example 1**	**Example 2**
Name of rules	dash-sourcing-DLR-BE-detail	dash-sourcing-UBE-DE-finder
Country	Belgium	Germany
Platform	Deliveroo	Ubereats
Cron expression	0 8,18 ? * THU *	0 17 * * ? *
Arranged date 1	Thu, 09 Jun 2022 08:00:00 GMT	Mon, 06 Jun 2022 17:00:00 GMT
Arranged date 2	Thu, 09 Jun 2022 18:00:00 GMT	Tue, 07 Jun 2022 17:00:00 GMT
Target	Lambda function	Lambda function

Table 4.1: Schedule rules set in EventBridge

The name of rules were specified clearly with specific country, platform, type of scraping process, in order to recognize which target to trigger, and also to let the engineers better manage the rules. The cron expression [1] plays essential role in this setting; it requires six field: minutes, hours, day of month, month, day of week, and year. As a result, having these six field provides sufficient information for a regular scheduler. For instance, the one in the example 1 in Table 4.1: " 0 8,18 ? * THU * " represents that the events will be triggered on every 8:00 and 18:00 in each Thursday every month and every year. Also, the one in example 2 in Table 4.1 represents the triggered events will be on 17:00 every day. The target of EventBridge is Lambda Function used to launch container in ECS. A certain set of parameters is passed to Lambda Function in order to run the function with correct parameters so that the launched containers also run the relevant scrapers (specific country, platform, and type). The mechanism details for Lambda Function and the rest of AWS services we utilized in this work are illustrated in the next subsection.

[1]https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/ScheduledEvents.htmlCronExpressions

4.2.3 Mechanism of AWS Service utilized

For the services in AWS we have utilized in this project except for EventBridge (described in last subsection), we describe the mechanism and how they work in our project in this subsection. The order of the content will be: *Lambda Function, ECR, ECS, CloudWatch, S3.*

- **Lambda Function (triggered to launch ECS container)**

As a service that lets developers run code without provisioning or managing servers, Lambda function has several characteristics: First, it does not require server. All the developers need to do is to write the code and upload it. This is so-called FaaS (Function as a Service) model of cloud computing. Second, it does the automatic scaling according to the size of workload. Third, Charging depends on the amount of time that code is computed, rather than on the server running time.

Based on the characteristics of Lambda Function summarized above, we come to several upsides by choosing Lambda Function in our work. First, it makes developing faster than the other requires servers. The developers can focus on the pure logic of the applications. Second, it reduces the computational cost by the charging policy and the functionality of automatic scaling. Third, it also decreases the operation management overheads by the functionality of automatic scaling(29).

After having the overview of features and benefits of developing by Lambda Function, we now illustrates mechanism and how it works in our project.

Whenever the workflow is triggered, the EventBridge will send corresponded set of parameters within a JSON file to Lambda Function. The parameters include:

  - *container_name*: To specify which container to launch
  - *cmd*: The command will be run in the container
  - *task_name*: Given task name for recognizing and using for logic in Lambda Function
  - *env*: To specify the source (the platform and the country) that will be scraped.

Once receiving the parameters above, a python program uploaded in the Lambda Function will be run with parameters, and then launch a ECS container by fargate[1] type.

- **Deploy application through ECR & ECS**

[1]Amazon ECS launch types (30)

Figure 4.2 displays the mechanism how ECR and ECS work with each other. ECR (1) can be refer to a "AWS Docker Hub" which let users store, manage and deploy Docker images. It compresses, encrypts the Docker images, and also provides versioning, tags, control of image lifecycle. ECS is a scalable service that deploys the Docker containers, pulls the Docker images from ECR When deploying. It allows users to manage and deploy the applications through API or task definition (31). Although Figure 4.2 shows the overview

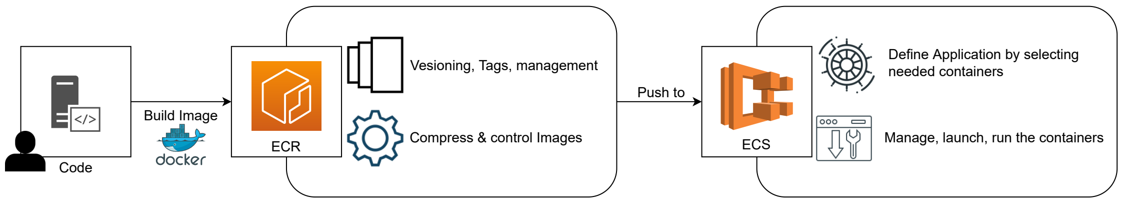

Code
Build Image
docker
ECR
Vesioning, Tags, management
Compress & control Images
Push to
ECS
Define Application by selecting needed containers
Manage, launch, run the containers

Figure 4.2: ECR to ECS mechanism (1, 2)

of workflow for ECR and ECS, we provide more details of how we deploy the application in our work as follows.

1. **Create repository**: The repository is a place to let users store Docker images in ECR. Whenever, we push or pull an image in ECR, we must specify the registry and the repository location.

2. **Build and tag Docker image**: The prerequisite of this step is to install AWS CLI, Docker, and do **Docker Login** to authenticate Docker to the registry. Afterwards, we are able to build and tag images through commands. Also, it is always good to run and test the image and container locally before pushing to ECR repository.

3. **Push the image to ECR**: After building and tagging the image, the image should be pushed to the specified repository.

4. **Create a task definition(31)**: A task definition which is written in JSON file, describes the task as how it runs the container. To be more specific, it should include which Docker images to use in containers, how much CPU and memory to use, the launch type, logging configuration etc.

After registering the task definition, we are now able to make use of the containers for the application (i.e. scraper in our case). Since the containers in ECS will be launch and run by Lambda Function, we do not need further actions or manual operation in AWS after the 4 steps mentioned above. However, we still are able to manually run the application manually by creating cluster and service through user interface in AWS manually.

- **Monitoring through CloudWatch**

The logs generated during the scraping procedure can be inspected through CloudWatch, which offered us a way to do monitoring. In addition to scraping log sent from ECS containers, the rules in EventBridge can also be monitored, and Lambda Function can be monitored as type of log streams so that we can track the scrapers in order. Moreover, CloudWatch also provides other functionality to further analyze or monitor the applications. For instance, Container Insights [1] which collects the metrics and logs of the containerized applications for troubleshooting. Alarms [2] are also provided for various purposes, such as setting threshold for errors; noticing administrator if the amount of errors exceed the threshold.

- **S3 & storage management**

With Amazon S3, we are able to retrieve, store and manage any amounts of data. The critical issue for storing data in our work is to well-organize the path of each scraping data to categorize them. As a result, we specify the storage path using: information to scrape, platform, country, date, and run id to not only make developer better categorize the result data but also let the engineers do the subsequent work easier.

4.3 Scraper and Improved Features

In this section, the tools, framework and improved features of scraper are described. We first provide a structure of the repository for developing which lets us have an overview of the whole scraper and its features. The rationale of keeping old tools (scrapy) will then be provided. Several features improved are also described namely: validator, notification and metadata extraction. Also, the rationale of developing certain features will be illustrated.

4.3.1 Git Repository Structure

We depict the Git repo structure as follow, list essential component and briefly explain their functionality in this section.

[1]https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ContainerInsights.html
[2]https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html

- properties: The config files and the JSON files for describing ECS task definition are included in the properties folder.
- pipeline: A sequential scraping pipeline is included here.
- process: It includes all the spiders which are categorized by scraping contents and platform. Also, it contains all the relevant features and extensions (e.g. metadata, notification, validator, pipeline for uploading parquets to s3 bucket).
- test: The unit tests and some sample texts for the specific platforms are placed here.
- script: It includes the scripts for setting certain platforms, countries and refresh month. Also, it consists of the entry point for running the scraper pipeline either locally or in the container. (e.g. poetry run python src/conso_scraper/scripts/run_scraper.py)
  - env.sh: The script for setting the environment parameters (e.g. platform, country, refresh month)
  - run_scraper.py: The entry point of running the scraping pipeline.
- .github/workflow: The YAML files describing the Github Actions workflow for CI/ CD pipelines are placed here. The specific details will be illustrated in next section.
- Dockerfile: The docker image setup for creating the scraper environment for either running locally or in cloud container.

```
conso_scraper
├── properties
│   └── ecs_task_definitions.json
└── src
    └── conso_scraper
        ├── pipeline
        │   └── sequential.py
        └── process
            ├── extensions
            │   └── metadata_extension.py
            ├── notification
            │   └── notification.py
            └── pipelines
                └── s3_pipeline.py
```

```
│   │   │
│   │   ├── spiders
│   │   │   ├── deliveroo
│   │   │   ├── doordash
│   │   │   ├── justeats
│   │   │   └── ubereats
│   │   └── validators
│   │       └── content_validator.py
│   ├── script
│   │   ├── env.sh
│   │   └── run_scrper.py
│   ├── utils
│   └── executor.py
├── test
├── Dockerfile
├── pyproject.toml
├── poetry.lock
└── .github
    └── workflows
```

4.3.2 Rational of the Technology Choices

The scraper program is implemented based on the scrapy[1] framework, which is one open source and collaborative framework utilized for extracting data from web pages with high flexibility and extendability. The rationale of choosing this framework is that the old scraping procedure was implemented by scrapy. Also, since the logic, the main body of scraping code, and the result data of old scraper had high adaptability to the data processing at the down streams, switching the main framework for scraper may increase the risk of occurring issue in any down stream processing. Moreover, due to its extendability, the improved features we about to describe can be developed by using the extension of it.

4.3.3 Validation procedure for scraping result data

The main purpose of validator for our scraper is to notify the engineers when there is any problem with the result of scraping data as soon as possible. Although the result of scrapped data must be checked by the quality assurance procedure which ensures the quality of the result data and also the process of scraping, there are still several checks

[1]https://scrapy.org/

that can be done earlier included in the scraper program. Another factor or reason of developing validator is that quality assurance procedure is a work with more human actions and manual care in the company. Therefore, a validator can make engineers detect the error or abnormal condition within the scraping data at the early stage. We come up with several strategies to validate the result of scraping data as follows:

> **Strategy 1**: **Evaluating the amount of HTTP status codes**: The scraping is a long requests procedure with possibility to success (i.e. status code 200, success) and fail (i.e. status code 400, bad request error). The strategy here is to check the ratio of status code 200 among all requests. If the ratio of success is low, the result data of scraping may not be usable.
>
> **Strategy 2**: **Threshold of the item size**: The size of each item will be recorded during the scraping procedure. Once the spider closed, we will be able to investigate the amount of item which size is extremely small so that can not be used afterwards. We set a threshold for item size, and evaluate the ratio between passing threshold and those do not pass.

4.3.4 Notification Report

The notification feature can be influential for engineers since they do not need to keep screening or periodically checking the scraping procedure and logs. The notification message will be sent to certain Slack [1] channel with several scraping result details. The message can be considered as a short report to the engineer team and also a notification for them to start to do the tasks (e.g. quality assurance, processing, etc) after scraping.

This feature was developed by using the extension of scrapy, and the framework called Spidermon[2] which is a framework to build monitor for scrapy spiders, i.e., our scrapers.

We provide the message format and the examples in the following Table 4.2. The screenshot of this notification message will be placed in appendix.

4.3.5 Extracting Metadata

The aim of extracting metadata feature within scraper program is to pass essential information of raw scraping data to the engineers who should take over the tasks afterwards. Even though our storage strategy already specify several information of the scraping, we still need to have a short metadata to clarify the stage of scraping progress.

[1] https://slack.com/

[2] https://spidermon.readthedocs.io/en/latest/

Object	Example 1
source	UBE
spider	outlet detail
country	AU
item scraped count	26421316
request account	310060
response account	310048
response 200 account	310048
log error count	11347
start time	2022-06-18 06:02:04.492408
finish time	2022-06-18 13:54:06.107996

Table 4.2: Notification Message Format

Since the metadata here is relevant to the procedure of scraping, we developed it using the extension of scrapy. The metadata table should provide the information of whether the scraping result data is valid or not, the exact time stamp of closing time of the scraper, the id number of running scraper, the table name which represents the result type of scraping, and the stage of processing procedure. With this information, the engineers can easily recognize the status of the scraping result data. Moreover, the stages of processing procedure is specified inn the path of storage directory in s3, so that the engineers doing subsequent tasks can also place their metadata their to have better collaborating procedure and flow.

4.4 CI/CD Pipeline using Github Actions

In this section, a completed CI/ CD pipeline design and its implementation procedure details are described. CI/ CD pipeline stands for continuous integration and continuous delivery which falls under DevOps. Several CI/ CD tools are used and developed for engineers nowadays, e.g., Jenkins[1], circleci[2], TeamCity[3], Bamboo[4], GitLab[5], Github Actions[6].

[1]https://www.jenkins.io/
[2]https://circleci.com/
[3]https://www.jetbrains.com/teamcity/
[4]https://www.atlassian.com/software/bamboo
[5]https://about.gitlab.com/
[6]https://github.com/features/actions

The CI means that changes of new code are built, tested and merged to a certain repository. Moreover, the CD can be referred to continuous delivery and continuous deployments. Once CI process finish building, testing for developed code, continuous delivery ensure that it packages the environment for production which can be used by operation team. Continuous deployment refers to the final stage of pipeline that deploy the application into the production environment (32). The Figure 4.3 shows the basic structure of CI/CD pipeline we described above.

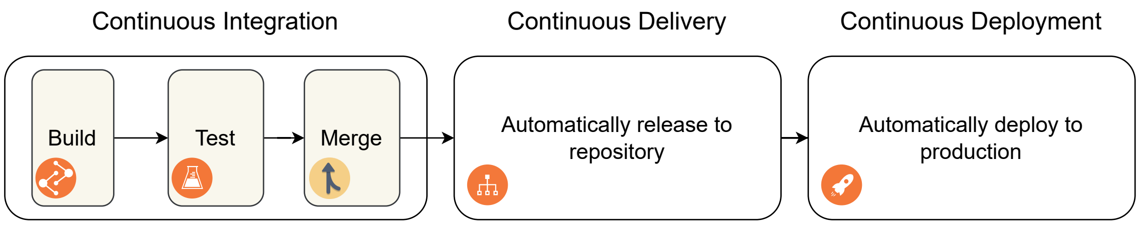


Figure 4.3: Basic CI/CD pipeline Structure (3)

The tool of CI/ CD pipeline we utilized is Github Actions. In this section, first introduce the Github Actions workflow, its characteristics and the rationale for choosing it. The components and the set-up of the Github Actions workflow are followed by the rationale of choosing. Afterwards, the commit standard for release workflow will be explained. Finally, the overview of CI/ CD pipeline structure for this project is displayed and illustrated. Also, the details for each workflow mechanism in pipeline is introduced after the pipeline structure.

4.4.1 Github Actions Workflow

Github Actions is a CI/ CD platform that allows users or to automate build, test and deployment pipeline. The developers can create workflows for the pipeline directly in the Github repository. Also, comparing to other platforms (e.g. Jenkins) using custom server, Github Actions provides serverless service which means that we do not have to maintain a server continuously to do the CI/ CD tasks. As a result, not only is the initial setup convenient but also requires less following maintenance than other platforms do.

Another advantages is the ease of use of Github Actions: since the workflows are written, developed in YAML file which is beginner friendly, it is suitable for small and startup company which engineers already have experience or fundamental knowledge in YAML file.

Moreover, it provides several different environments for building, testing, e.g., Linux, MacOS, Windows. Other than ease of use and maintenance, the extendability is also important for CI/ CD pipeline. Github Actions has its own community and provides open source platform which let users reuse the actions developed by other developers. For the cost of running CI/ CD pipeline, the billing strategy for Github Actions must be taken into consideration. Github provides certain amounts of free minutes and storage [1] which depends on the product utilized by users. To have clear overview of Github Actions we described here, we listed all the characteristics and advantages in Table 4.3.

Characteristics	**Description**
Convenience	Develop directly in Github repository
Ease of use	Written in YAML file
Server	Serverless
Environments	Linux, MacOS, Windows
Extendability	Open source actions in community
Billing strategy	Certain amount of free storage and time

Table 4.3: Characteristics of Github Actions

- **Components of Github Actions**
  Here we list several essential components required to set up Github Actions workflow:
  - **Workflows** [2]:
    A workflow refers to a configurable automated process which consists of several jobs. Workflows must be defined by YAML file, and be triggered by events.
  - **Events**:
    An event is a specific activity in a Github repository that triggers a workflow run. For example, activity can originate from GitHub when someone creates a pull request, merges to certain branch, opens an issue, or pushes a commit to a repository.
  - **Jobs**:
    A job is a set of steps in a workflow that execute on the same runner. Each

[1]https://docs.github.com/en/billing/managing-billing-for-github-actions
[2]https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions

step in a job is either a shell script that will be executed, or an action that will be executed.

- **Actions**:
  An action is a custom application for the GitHub Actions platform that performs a complex but frequently repeated task. We can also find available actions to use in workflows in Github Marketplace[1].

- **Runners**:
  A runner is a machine that runs your workflows when they are triggered. Each runner can execute one or more jobs at a time. GitHub provides Ubuntu Linux, Microsoft Windows, and MacOS runners to run workflows. The runners here refer to the environments described in the Table 4.3.

- **Creating a Workflow**
  There are two options for creating a workflow in Github repository:

  Option 1: Create the *.github/workflows* directory in the repository, and then place the developed workflow YAML files in it.

  Option 2: In the repository, go to the *Action* page and select any suggested available workflow there. Afterwards, it will create YAML file with edit page in the browser. Once the user finishes editing, committing and pushing to the repository, the workflow and the directory will be generated.

To have a customized workflow for CI/ CD pipeline for our projects, option 1 is more suitable for us.

4.4.2 Setting up Github Actions in Repository

- **Encrypted Secrets for Github Actions**[2]
  To start setting up the Github Actions workflow, encrypted secrets must be added in the environment so that the actions are able to be executed since they operate and run the program in the Github repository or use the service from AWS, Slack, Azure. In our case, Github token for accessing repository, AWS access key ID and AWS secret access key for accessing AWS services are needed.

[1]https://github.com/marketplace

[2]https://docs.github.com/en/actions/security-guides/encrypted-secrets

- **Workflows and Actions in the Repository**
  We first list all the workflows and action components in the directory as follows, and provide further explanation afterwards.

```
.github
└── workflows
    ├── _develop.yml
    ├── _main.yml
    ├── _release.yml
    ├── _pull_request_develop.yml
    ├── _pull_request_main.yml
    ├── _pull_request_release.yml
    ├── python_pytype.yml
    ├── python_pytest.yml
    ├── python_pylama.yml
    ├── python_publish_test.yml
    ├── python_black.yml
    ├── pytest_coverage.yml
    ├── git_tag.yml
    ├── git_release.yml
    ├── git_delete_tag.yml
    ├── docker_build_push.yml
    └── aws_ecs_task_definition.yml
```

The YAML files with prefix of "_" describing specific CI/CD pipeline workflows (e.g. _develop.yml, _main.yml, _release.yml describe the one for pushing events at those branches, while the other three files describe the one for pull request events on those branches.) are "caller workflow", while the ones without prefix (e.g. python_pytype.yml, python_pytest.yml etc.) are "called workflows". Although caller workflows are able to includes every job or actions in one single YAML file for an event, the reason why we separate the jobs to several different YAML is to consider the extendability, flexibility and the maintenance afterwards. To be more specific, the called workflows are reusable [1] and can be developed separately so that it is convenient for developer to debug, maintain, extend the whole workflow.

[1]https://docs.github.com/en/actions/using-workflows/reusing-workflows

4.4.3 Auto Releasing with Identical Version & Tag by Commit Standard

To make CD process automatically and smoothly, we managed to propose a strategy that does github release by specific commit format. Also, the Github release version must coincide with the one in Docker image and the container in the AWS service, i.e., ECR, ECS.

The first step of implementing this strategy is to specify the format of commits. As a result, we make the engineers conform to the conventional commits specification [1] which refers to a lightweight convention for every commit message. Afterwards, we create a release workflow which will be triggered by the pushing to release branch. This release workflow will create corresponded tag and version according to the commit message which follows the commit convention; and then do the release with them. Table 4.4 illustrates the commit message convention and the corresponded release type.

Commit Format	**Description**	**Release type**
fix:	Bug fixes	Patch release
feat:	Feature developed	Minor release
perf:	Performance changes	Major release

Table 4.4: Commit Convention for releasing

Although there are still other conventions and rules that can be used to trigger these three specific release types (e.g. style, refactor, test, chore etc.), we simplify them into three types to make the workflow procedure more convenient and be easily recognized by other engineers in the organization.

4.4.4 CI/ CD Pipeline Architecture

Figure 4.4 displays the overview of CI/CD pipeline structure for this project. In the industry of developing software application or products, the engineers develop the feature and performs all the maintenance tasks by working among branches, e.g., feature, develop, release, and main branch. This collaboration among engineers and branches is so-called Gitflow Workflow (4) which displays in Figure 4.5.

With Gitflow, the developers are able to make better DevOps practice and continuous delivery. Moreover, combining Gitflow strategy and the CI/ CD pipeline makes our developing and maintaining procedure more smooth and automatic.

[1]https://www.conventionalcommits.org/en/v1.0.0/

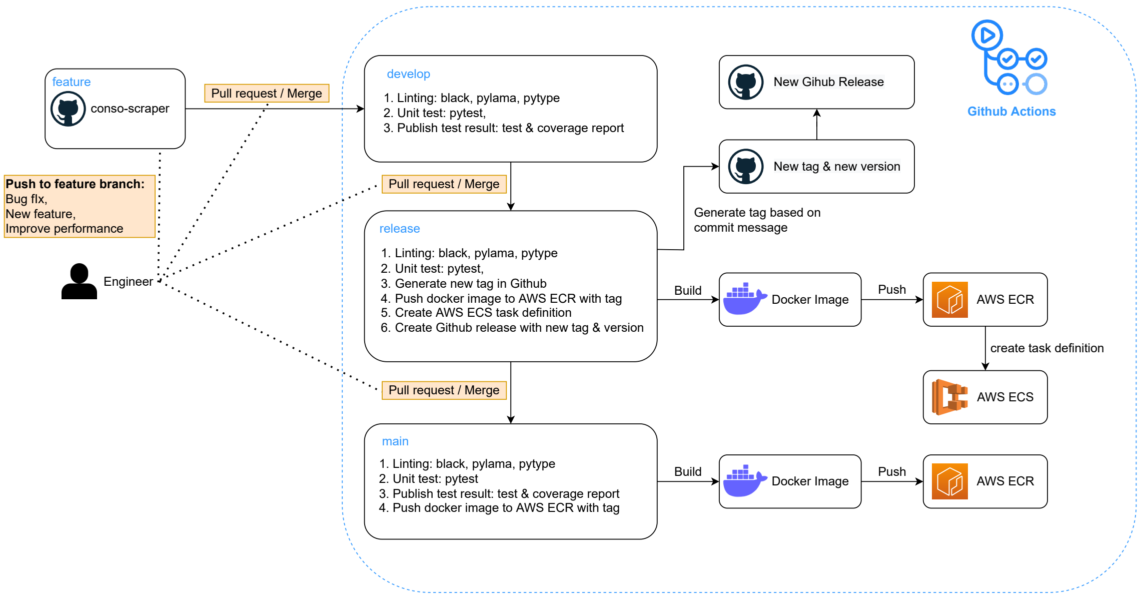

feature
conso-scraper
Pull request / Merge
Push to feature branch:
Bug fix,
New feature,
Improve performance
Engineer
develop
1. Linting: black, pylama, pytype
2. Unit test: pytest,
3. Publish test result: test & coverage report
Pull request / Merge
release
1. Linting: black, pylama, pytype
2. Unit test: pytest,
3. Generate new tag in Github
4. Push docker image to AWS ECR with tag
5. Create AWS ECS task definition
6. Create Github release with new tag & version
Pull request / Merge
main
1. Linting: black, pylama, pytype
2. Unit test: pytest
3. Publish test result: test & coverage report
4. Push docker image to AWS ECR with tag
New Gihub Release
New tag & new version
Generate tag based on commit message
Github Actions
Build
Docker Image
Push
AWS ECR
create task definition
AWS ECS
Build
Docker Image
Push
AWS ECR

Figure 4.4: Overview of CI/CD pipeline structure

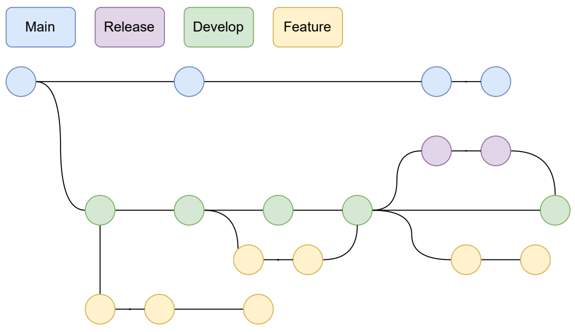


Figure 4.5: Gitflow Workflow Illustration (4)

Whenever the engineers finish developing the features and then do the push or pull request to develop branch, the workflow displayed in Figure 4.6 is triggered. It first check the code quality and the format in the Ubuntu Linux environment using the linting tools provided by python since the codes in our work are mainly developed by python. After linting tasks, it launch another runner for doing unit testing. Once all the tests passed, the Github Action bot will publish the test result in the repository and also comment with the coverage report within the certain commit or pull request. After commenting, the workflow finishes which also means all the checks pass.

Moreover, we emphasize that all the modules in the CI/ CD pipeline workflow (Figure 4.6) are dependent on the up-stream components. That is, if one of the component fail at the up stream, the workflow terminate and skip all the components at the downstream. For instance, if the pylama fail, the test module is skipped and it also means the workflow fail.

For the main branch, the triggered workflow executes all the tasks that the same as the develop branch does. After finishing the test module, it build the Docker image and push the image with “latest” tag to AWS ECR. Figure 4.7 illustrates the workflow for main branch.

For the release branch, the release workflow runs all the missions which the same as the main branch does. However, the difference is that once the push commits are included with the specific format, the action will generate a new tag using the bump-up version. As a result, the tag for Docker image is generated by Github Action automatically so that it is different from the one in main branch workflow. Moreover, the triggered workflow will create ECS task definition for the Docker container deploy in ECS and release a Github release which both use the new bump-up tag and version. Finally, the version recorded in “pyproject.toml” file in the repository which describes the using packages and environment will be update by Github Action bot.

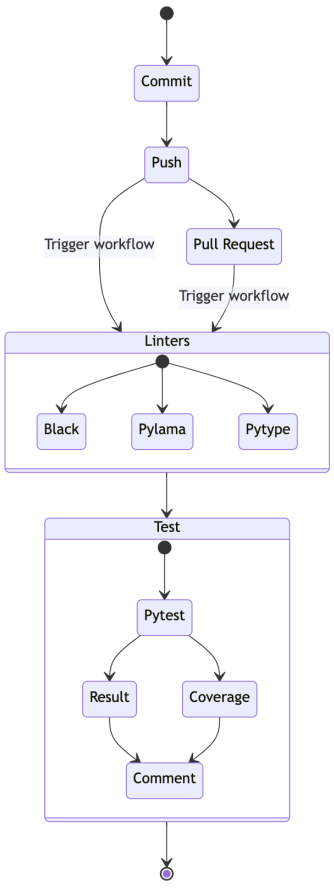


Figure 4.6: Workflow for Develop Branch

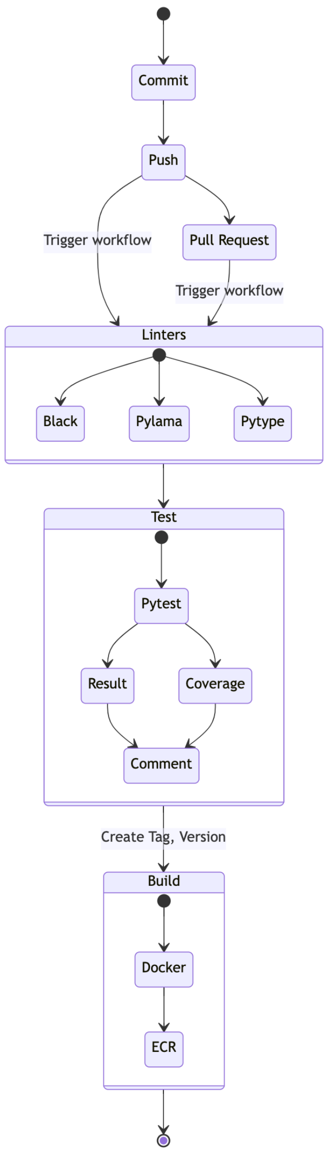


Figure 4.7: Workflow for Main Branch

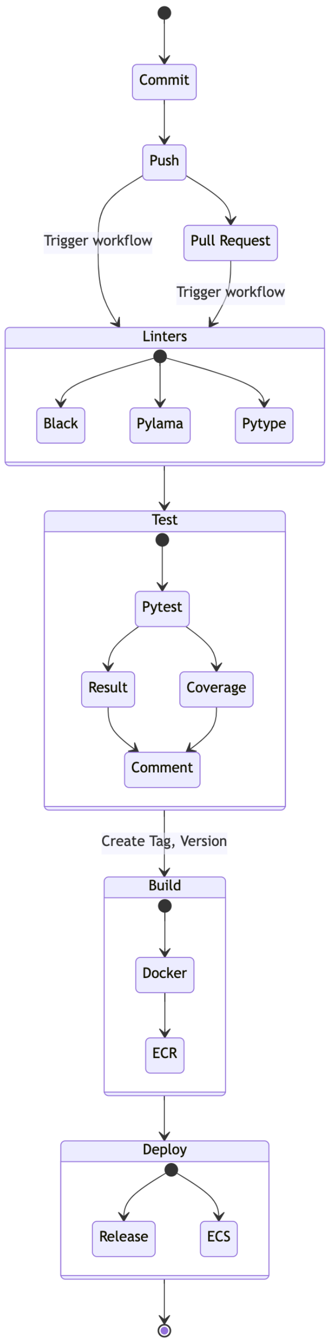


Figure 4.8: Workflow for Release Branch

5

Discussion

The automation workflow, CI/ CD pipeline and the related work combine the knowledge from technical field, academic areas, guidance and introduction in the marketplace and team's experience. To evaluate our work, the following reflection including the comparison between previous and current setup, as well as the discussion of the limitation of the current setup is provided. Also, because of the time limit and the company's plan, there is some future work can be implemented to improve and make the whole project more complete, which is illustrated in this chapter.

5.1 Reflection

5.1.1 Comparison

- **Previous Setup**
  Before developing this automation workflow and CI/ CD pipeline, the engineers take over certain takes of the scrapers which have been automated in our proposal. The different scrapers were developed and tested locally by one or more engineers. Once the scrapers had been finished, they were uploaded to the server which hosted and maintained by our company. The drawbacks of this developing model are obvious and listed as follows.

  - **Collaboration among Engineers is Limited**:
    Since the scrapers are developed locally, the engineers may focus on developing their own work and even do not upload their code to Github repository.
  - **Different Scrapers Management**:
    As the engineers developed their scraper separately and locally, it is difficult to

manage the version and the update of scrapers.

- **Hard to maintain the code format and quality**:
  Without CI/ CD pipeline, the linting tasks and tests are run locally, and may ignore by engineers. As a result, the maintenance of code format and quality are difficult.
- **Maintenance of Server**
  Some team members must take over hosting and maintaining the server, and it is the resource cost for the team.
- **Scalability & Stability**:
  It is the main reason why we develop a new strategy for scraping data. Since all the scrapers were run in one server, the crash of server caused by one scraper would affect all the other running scrapers deployed in the server. Therefore, it is the the reason for not being able to scale up and extract the data stably.
- **Extendability & Integrability**:
  Since most of other processing procedures in the company are deployed in the AWS, it is hard to integrate scraping procedure with the other if it is deployed in a separated and independent server.

• **Current Continuous Scraping**
The current continuous scraping pipeline includes scheduled automation pipeline (Section 4.2.1) and CI/ CD pipeline (Section 4.4.4), which may solve and improve the downsides of the old setup to a certain extent. These improvement are the advantages of this new scraping procedure and also some of them are the main reason to migrate the old setup to the current one. We list the upsides of the current setup as follows.

- **Promote the Collaboration among Engineers**:
  A well-organized Github repository, no need for uploading different scraper program separately, and automatic checks executed by Github Actions are all providing the conditions that make engineers collaborate more smooth and efficient.
- **Smooth Scrapers Management**:
  With the automatic release and CI/ CD pipeline in our repository, the scraper management procedure becomes smoother and easier. The tags, versions, and the changelog automatically generated by Github Actions also improve the management of version control.

  - **Ensure the Code Format and Quality**:
    Since there is no need for install package for linting test locally, all the linting checks are took over by Github Actions, so that the code format and quality can be ensured.
  - **No need for Maintaining Server**:
    The scrapers are run in separated ECS container without hosting and maintaining the server for them. This frees the engineers from taking care of the server, and lets them focus on the development and maintenance of applications.
  - **Scalability & Stability**:
    The scheduled automation workflow scale up our scraping procedure. Moreover, the notification and validating process make the engineers find the issue earlier, which also influence and increase the scale of our scraping data.
  - **Extendability & Integrability**:
    As the scraper program deployed in the AWS, it can be integrated with other processing steps. The potential integrated work will be elaborated in future work (Section 5.2)

5.1.2 Discussion

We describe the upsides and the improvement of our solution comparing to the previous setup in the last section. However, there are still some limitation and unsolved challenges in our solution. Also, some doubts for choosing the tools are raised and answered in this section. Moreover, the comparison between adding CI/ CD pipeline and without it is discussed.

- **Limitations**

The migration work is the first challenge for the team. Since most engineers who focus on developing scrapers are not familiar with ~~the~~ cloud computing services. The migration must take times for them to get used to the new tools and new way to deploy their scrapers.

Although CI/ CD pipeline automatically deploy the scraper on the cloud, it also takes more time for the whole pipeline procedure: pushing to Github repository, pushing image to ECR, creating new release and creating new task for ECS, comparing to the previous way: uploading to the server.

Moreover, though the notification reports sent to the slack channel inform the situation of the scraper process, the channel was messy and unstructured when lots of scraper program finish in the closed time period.

Despite scaling up comparing to previous setup, it is another challenge to improve the scalability for current continuous scraping workflow. Since our scrapers extract the data from several food delivery platform, they must restrict the frequency of requests according to the ethics and legal consideration mentioned in Section 3.1.5. In addition to the restriction of request frequency, there are some websites applying IP blocking which forces us to switch the IP in a IP pool. The switching procedure and the limited IPs also are the bottleneck of scalability.

- **Raised Doubts**

As we introduced several powerful automated workflow orchestration tools in Section 3.2.2, we do have advanced tools to choose in the marketplace. However, we choose to set up current workflow structure since there are still lots of steps after scraping which require engineers' manual operation. Also, some of them are not deployed to the AWS. Therefore, we come up with this automation workflow structure that is a transition state before all the other components are finished the automatic procedure and deploying to AWS.

Figure 3.1 illustrates a simplified automation workflow for our work which starts from scheduler: EventBridge, launching ECS by Lambda, to the running scraper. However, EventBridge can also trigger the event for launching ECS container. The reason for adding Lambda is for the convenience of management. With Lambda as the agent between ECS and schduler, if we have any update for launching ECS, we just need to update one Lambda function. However, without Lambda, each scheduler is matched to a independent ECS event. If any update is required, we may go through all the rules in the EventBridge to update every event.

- **Comparison between with & without CI/ CD pipeline**

Undoubtedly, with automatic checks, delivery, and deployment, CI/ CD pipeline enhances the automatic level of whole workflow. How and What exact advantages it provides are discussed here.

Without CI/ CD pipeline, the developer must go through the *Development procedure (Steps)* mentioned in Section 4.2.1. The deploying procedure is the manual operation and takes time to finish it. CI/ CD pipeline allows us to ignore several steps in the deploying procedure. Moreover, building Docker image and pulling the image from ECR for testing not only cost the computing resource locally, but also require the performance and storage of local computers owned by the engineers. As a result, CI/ CD resolve the concerns for the requirement of local computing performance and also free the computing resource locally for engineers.

5.2 Future work

Although the solution we provided can be improved in different points. We first provided potential pipeline (DAG) prototype integrating other steps using advanced workflow orchestration, i.e., Apache Airflow. Then, another potential strategy to enhance the scalability through Kafka is provided. Afterward, the improvement of the features for the scrapers is delineated. Finally, several points of enhancing the CI/ CD pipeline are described.

5.2.1 Integrating with Advanced Workflow Orchestration: Airflow

As we introduced in the Section 3.2.2, there are plenty of tools in the marketplace offering workflow orchestration service for either business or scientific purpose. We take Apache Airflow as example for potential integration work since other teams are already using it for the data pipeline tools.

Airflow was developed by Airbnb, a open source platform for developers to author, schedule, and monitor workflow. It is capable of integrating the tools or components deployed in the cloud computing service, e.g., AWS, Azure, GCP. As a result, it is suitable for us to integrate and extend the data pipeline since we are now pursuing the high automation level using AWS service. Airflow allow us to construct the DAGs which describe the queue of tasks and the dependencies together with inputs and outputs (23). The DAG structure may be more complex than the one-way simple pipeline. Figure 5.1 illustrate the potential integrating work for merging the scraper with other operators.

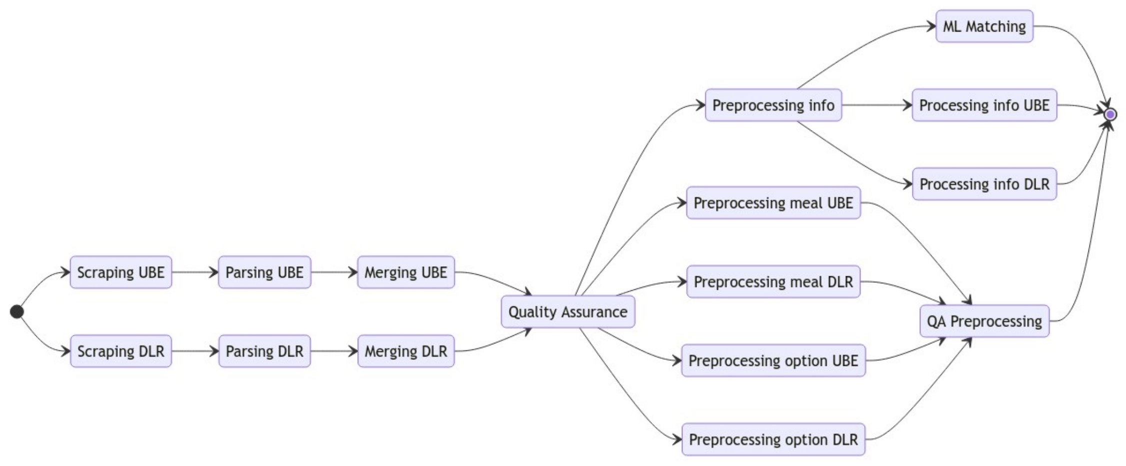


Figure 5.1: DAG Illustration of Integration for Scraping and other Components

5.2.2 Scraping with Kafka and Redis

As the scrapy is the highly extendable framework for scraping, some developers creates a scrapy project using Kafka[1] and Redis[2] for a distributed on demand scraping cluster: *Scrapy Cluster*[3]. This work utilize Kafka as data bus for other application to interact with the scrapy cluster. For instance, the scraper can be started, published, stopped, log processing, result outputting, job submitting through kafka monitor. Redis here is a high-performance in-memory database which processes and stores the incoming request queue, and records the status of scraping procedure.

Although this work scale the scraper instances across one or multiple machine, the bottleneck of scalability in our current solution is the request frequency restriction of the food delivery platform website, so that this work may not improve the scalability issue if we still scrape the data from specific delivery platform website. However, it is still a possible solution as long as we collect all the e-commerce websites and scrape the data from them in the future, so that the scaling up will be possible. Moreover, AWS also provide Managed Streaming for Apache Kafka (MSK)[4] to manage Kafka in the cloud service, which will be useful for this future work.

5.2.3 Improving Features

As the limitation of notification report described in Section 5.1.2, it is better to come up with a way to avoid report squeeze into one channel in a closed time period. We can either divide the notification channel by different platforms and countries, or design and develop a user interface for visualizing the scraping statue which also should be divided by platforms and countries.

Currently, the metadata for each run of scraping is store in s3, which can also be improved in a efficient and organized way. Since the storage used by this metadata is low, it does not need to store it in s3 bucket. The AWS RDS[5] which supports PostgreSQL, MySQL may be the good option for it.

Although the notification report record the number of the error for the scraping procedure, we did not handle the error when the error number is small. However, as the request number increase, the error number may increase as well. It is always good to find a way to

[1]https://kafka.apache.org/
[2]https://redis.io/
[3]https://scrapy-cluster.readthedocs.io/en/latest/
[4]https://aws.amazon.com/msk/
[5]https://aws.amazon.com/rds

solve the errors before scraper finished, even the ratio between error and request number is low. One possible way to solve it is to record the error and store such request in a list. Once the scraper finish all the other request, it re-run the error request in the list.

5.2.4 Enhancing CI/ CD pipeline

Although CI/ CD pipeline in our work almost covers all deployment procedure, the test run in the cloud (ECS) is still absent. We can add another components in the CI/ CD pipeline for release branch. Once the ECS task definition is created, Github Actions should run a small case to test the scraper.

Another point can be improve here is the versioning of the Github release and the docker image. Our design for auto releasing must bump up the version for any update conform the commit standard. However, in some situation, we would like to update but keep the version without bumping. For instance, some bug small fix does not make sense for a new version release. Therefore, a way to keep the same version but also deploy updated image to the EMR and ECS must be considered.

6

Conclusion

In this thesis, we present an scheduled automation workflow for web scraping deployed in the cloud using AWS and also the CI/ CD pipeline using Github Actions.

For the automation workflow development and operation, the rules in scheduler of AWS (EventBridge) are created by cron expression, the Lambda Function is set to connect the scheduler to ECS and also launch container in ECS, the ECS includes the containers which run the scraper program, the s3 bucket is used to store the scraping result data along with the metadata, the CloudWatch is used to Monitor the scraping procedure, the ECR is the registry we manage docker images of scrapers. Also, several features are developed in order to make engineers conveniently manage the scraper: validator, extracting metadata, and notification report.

For the CI/ CD pipeline development, we implement such pipeline using Github Actions by considering its characteristics. The complete CI/ CD pipeline workflow structures for three specific branches: develop, main, release are illustrated in Figure 4.6, Figure 4.7, Figure 4.8. The CI/ CD pipeline includes auto-checking for the code quality and format, automatic building docker image to ECR, automatic creating the ECS task definition, automatic doing Github release. Moreover, we design a strategy that bump up the version for release and image according to the commit message which conforms to the specific commit format.

The goal of this graduation project and thesis is to provide a solution to scale up the scraping by applied scheduled automation workflow, and also investigate the challenges, limitations for the web scraping and continuous automation workflow. We tried to answer and briefly conclude the research questions as follows.

- *RQ1: What are the challenges in developing and maintaining contiuous pipeline for web data scraping?*

We can summarize the answers ~~in~~ two perspectives, automation workflow relevant and web scraping relevant. For the automation workflow relevant challenges, we are first forced to confront the migration challenge when adopting the changes of developing and deploying scraper. Also, choosing appropriate cloud service tools and frameworks for such automation pipeline are critical and difficult. Moreover, the integrability and extensibility for this automation pipeline are important and must be considered. For the web scraping relevant challenges, since the automation pipeline is design for the scraper, its performance is closely related to the web scraper. Thus, the challenges for web scraper (we discuss in next paragraph) are also related to the one here.

- *RQ2: What are the challenges for web data scraping, and what are the potential solutions for them?*
  We collect literature from academic area and list 7 points in the Chapter 3: dynamic content, anti scraping, login requirement, honeypot traps, trade-off between performance and automation, scalability, and heterogeneity. For these challenges, some of them can be address at certain extent by scraper program, e.g., IP address list for IP blocking, training Tesseract [1] for CAPTCHA, HTTP *basic access authentication* for login requirement (17). Moreover, the ethics consideration discussed in Section 3.1.5 can also be seen as a challenge. However, we considered the scalability as the most critical issue in this field. Although we scale up our scraping process by constructing an automated workflow, another bottleneck emerge. Therefore, we proposed possible solution for future work to improve it in Section 5.2.2. As a professional web scraping service provider, zyte(**?**), mentioned in their white paper: there is no simple solution to these challenges (design scalable architecture, pursue high-performance and efficient scraping) (18).

To conclude, our work scales up the scraping procedure, and also automate the execution of scraping by the schedule. Also, the CI/ CD pipeline increases the automation level of our work. Moreover, deploying the scraper ~~into cloud~~ set the foundation for integrating the components in the data scraping team and the ones from other data processing teams. Therefore, it fulfil the requirement described before which scraping data in a more reliable and scalable way, and is flexible and extendable to be integrated with other components in the future.

[1]https://github.com/tesseract-ocr/tesseract

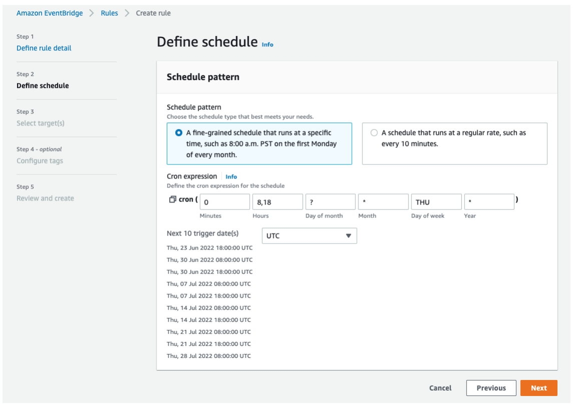
7

Appendix

A EventBridge rule setting

The setting of rules in EventBridge can be done through the user interface provided by AWS, the prerequisites are that the target (Lambda Function), the cron rules format, and the input parameters for target (in our case, a JSON file) should be prepared.

A.1 Example of creating rules



Define schedule

Schedule pattern

Schedule pattern
Choose the schedule type that best meets your needs.

[✓ A fine-grained schedule that runs at a specific time, such as 8:00 a.m. PST on the first Monday of every month.]
A schedule that runs at a regular rate, such as every 10 minutes.

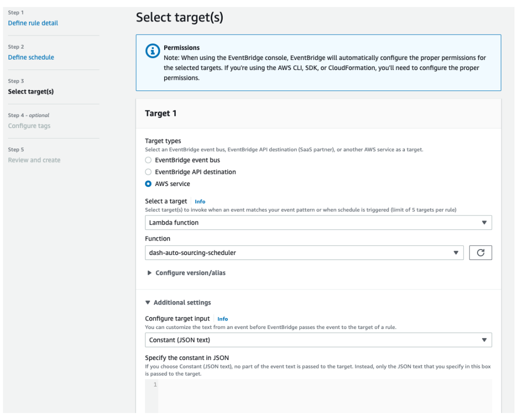
Cron expression
Define the cron expression for the schedule

	Minutes	Hours	Day of month	Month	Day of week	Year	
cron (	0	8,18	?	*	THU	*	)

Next 10 trigger date(s) UTC

Thu, 23 Jun 2022 18:00:00 UTC
Thu, 30 Jun 2022 08:00:00 UTC
Thu, 30 Jun 2022 18:00:00 UTC
Thu, 07 Jul 2022 08:00:00 UTC
Thu, 07 Jul 2022 18:00:00 UTC
Thu, 14 Jul 2022 08:00:00 UTC
Thu, 14 Jul 2022 18:00:00 UTC
Thu, 21 Jul 2022 08:00:00 UTC
Thu, 21 Jul 2022 18:00:00 UTC
Thu, 28 Jul 2022 08:00:00 UTC

Figure 7.1: EventBridge rules setting procedure 1



Select target(s)

Permissions
Note: When using the EventBridge console, EventBridge will automatically configure the proper permissions for the selected targets. If you're using the AWS CLI, SDK, or CloudFormation, you'll need to configure the proper permissions.

Target 1

Target types
Select an EventBridge event bus, EventBridge API destination (SaaS partner), or another AWS service as a target.
EventBridge event bus
EventBridge API destination
[✓ AWS service]

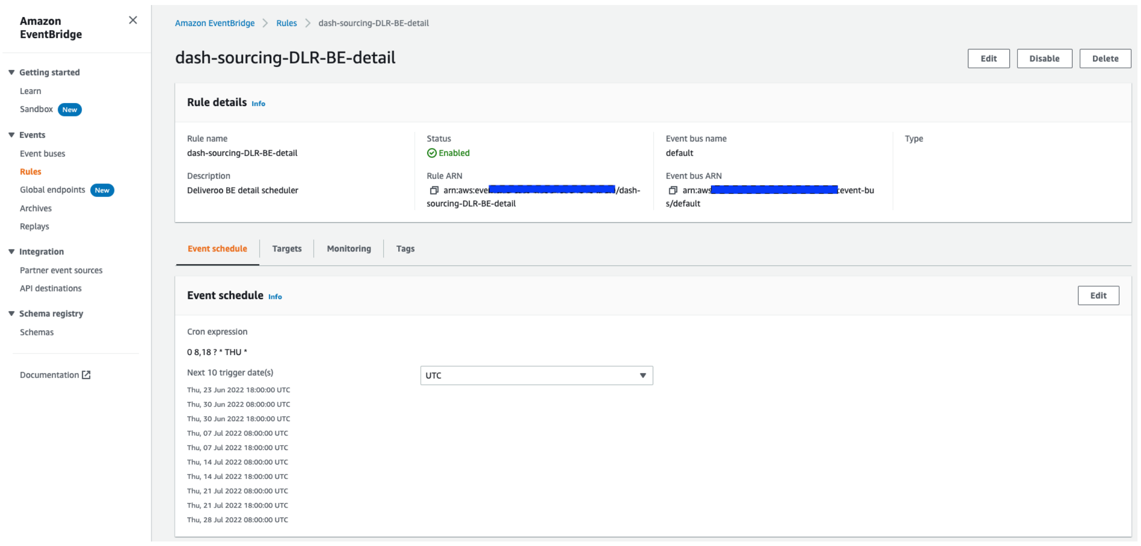
Select a target
Select target(s) to invoke when an event matches your event pattern or when schedule is triggered (limit of 5 targets per rule)
Lambda function

Function
dash-auto-sourcing-scheduler

Configure target input
You can customize the text from an event before EventBridge passes the event to the target of a rule.
Constant (JSON text)

Specify the constant in JSON
If you choose Constant (JSON text), no part of the event text is passed to the target. Instead, only the JSON text that you specify in this box is passed to the target.

Figure 7.2: EventBridge rules setting procedure 2



dash-sourcing-DLR-BE-detail

Rule details

Rule name: dash-sourcing-DLR-BE-detail

Description: Deliveroo BE detail scheduler

Status: Enabled

Rule ARN: arn:aws:eve[redacted]/dash-sourcing-DLR-BE-detail

Event bus name: default

Event bus ARN: arn:aws[redacted]:event-bus/default

Type

Event schedule

Cron expression

0 8,18 ? * THU *

Next 10 trigger date(s) UTC

```
Thu, 23 Jun 2022 18:00:00 UTC
Thu, 30 Jun 2022 08:00:00 UTC
Thu, 30 Jun 2022 18:00:00 UTC
Thu, 07 Jul 2022 08:00:00 UTC
Thu, 07 Jul 2022 18:00:00 UTC
Thu, 14 Jul 2022 08:00:00 UTC
Thu, 14 Jul 2022 18:00:00 UTC
Thu, 21 Jul 2022 08:00:00 UTC
Thu, 21 Jul 2022 18:00:00 UTC
Thu, 28 Jul 2022 08:00:00 UTC
```

Figure 7.3: EventBridge rules setting result

A.2 Result of set rules

B Lambda Function setting

Lambda function could be set up through the guide in the documentation which AWS provided, the only thing engineers should do is to develop the code put in it (in our case, a python script to launch ECS containers.

B.1 Steps of setting Lambda Function

We provide the steps of creating Lambda Functions as follows:

Step1: Open the page at Lambda console in the AWS user interface.

Step2: Create Function at the console page

Step3: Determine the Function name and set the runtime

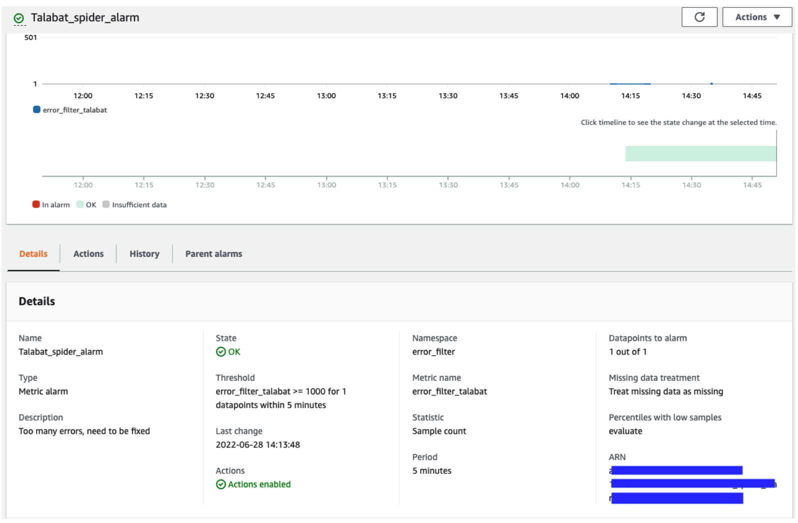
Step4: Create and Invoke the Function to test

Step5: Once the Function create, go to the code source pane, and upload out lambda function code

Step6: Save the changes

C CloudWatch Monitoring Log

C.1 CloudWatch Alarm

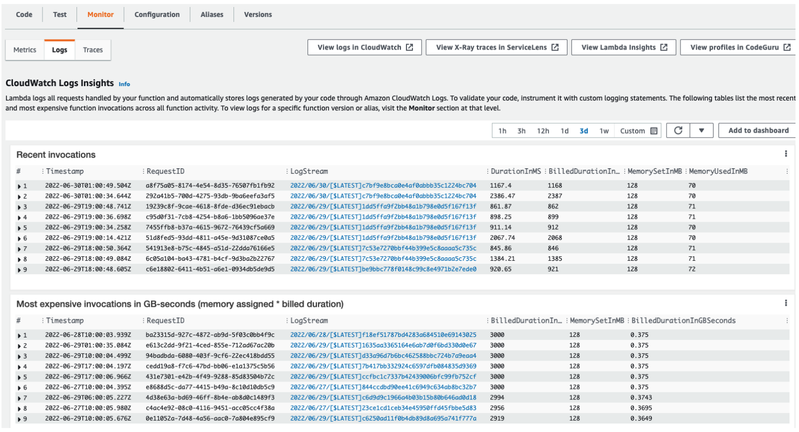


Details

Name Talabat_spider_alarm	State ✓ OK	Namespace error_filter	Datapoints to alarm 1 out of 1
Type Metric alarm	Threshold error_filter_talabat >= 1000 for 1 datapoints within 5 minutes	Metric name error_filter_talabat	Missing data treatment Treat missing data as missing
Description Too many errors, need to be fixed	Last change 2022-06-28 14:13:48	Statistic Sample count	Percentiles with low samples evaluate
	Actions ✓ Actions enabled	Period 5 minutes	ARN [redacted]

Figure 7.4: CloudWatch Alarm

C.2 CloudWatch Insight



CloudWatch Logs Insights Info

Lambda logs all requests handled by your function and automatically stores logs generated by your code through Amazon CloudWatch Logs. To validate your code, instrument it with custom logging statements. The following tables list the most recent and most expensive function invocations across all function activity. To view logs for a specific function version or alias, visit the **Monitor** section at that level.

Recent invocations

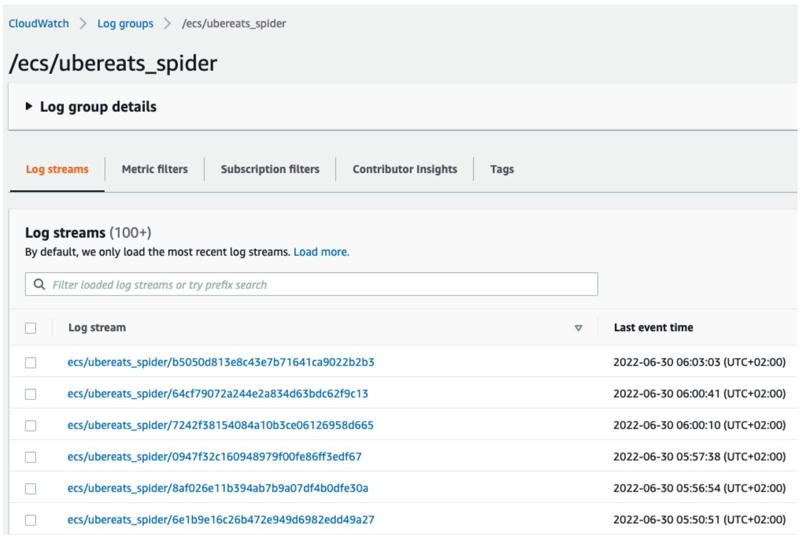
#	Timestamp	RequestID	LogStream	DurationInMS	BilledDurationIn…	MemorySetInMB	MemoryUsedInMB
1	2022-06-30T01:00:49.504Z	a8f75a05-8174-4e54-8d35-76507fb1fb92	2022/06/30/[$LATEST]c7bf9e8bca0e4af0abbb35c1224bc704	1167.4	1168	128	70
2	2022-06-30T01:00:34.644Z	292a41b5-700d-4275-93db-9ba6eefa3af5	2022/06/30/[$LATEST]c7bf9e8bca0e4af0abbb35c1224bc704	2386.47	2387	128	70
3	2022-06-29T19:00:48.741Z	19239c8f-9cae-4618-8fde-d36ec91ebacb	2022/06/29/[$LATEST]1dd5ffa9f2bb48a1b798e0d5f167f13f	861.87	862	128	71
4	2022-06-29T19:00:36.698Z	c95d0f31-7cb8-4254-b8a6-1bb5096ae37e	2022/06/29/[$LATEST]1dd5ffa9f2bb48a1b798e0d5f167f13f	898.25	899	128	71
5	2022-06-29T19:00:34.258Z	7455ffb8-b37a-4615-9672-76439cf5a669	2022/06/29/[$LATEST]1dd5ffa9f2bb48a1b798e0d5f167f13f	911.14	912	128	70
6	2022-06-29T19:00:14.421Z	51d8fed5-93dd-4811-a45e-9d31087ce0a5	2022/06/29/[$LATEST]1dd5ffa9f2bb48a1b798e0d5f167f13f	2067.74	2068	128	70
7	2022-06-29T18:00:50.364Z	541913e8-b75c-4845-a51d-22dda76166e5	2022/06/29/[$LATEST]7c53e7270bbf44b399e5c8aaaa5c735c	845.86	846	128	71
8	2022-06-29T18:00:49.084Z	6c05a104-ba43-4781-b4cf-9d3ba2b22767	2022/06/29/[$LATEST]7c53e7270bbf44b399e5c8aaaa5c735c	1384.21	1385	128	71
9	2022-06-29T18:00:48.605Z	c6e18802-6411-4b51-a6e1-0934db5de9d5	2022/06/29/[$LATEST]be9bbc778f0148c99c8e4971b2e7ede0	920.65	921	128	72

Most expensive invocations in GB-seconds (memory assigned * billed duration)

#	Timestamp	RequestID	LogStream	BilledDurationIn…	MemorySetInMB	BilledDurationInGBSeconds
1	2022-06-28T10:00:03.939Z	ba23315d-927c-4872-ab9d-5f03c0bb4f9c	2022/06/28/[$LATEST]f18ef51787bd4283a684510e69143025	3000	128	0.375
2	2022-06-29T01:00:35.084Z	e613c2dd-9f21-4ced-855e-712ad67ac20b	2022/06/29/[$LATEST]1635aa3365164e6ab7d0f6bd330d0e67	3000	128	0.375
3	2022-06-29T10:00:04.499Z	94badbda-6080-403f-9cf6-22ec418bdd55	2022/06/29/[$LATEST]d33a96d7b6bc462588bbc724b7a9eaa4	3000	128	0.375
4	2022-06-29T17:00:04.197Z	cedd19a8-f7c6-47bd-bb06-e1a1375c5b56	2022/06/29/[$LATEST]7b417bb332924c6597dfb084835d9369	3000	128	0.375
5	2022-06-29T17:00:06.966Z	431e7301-e42b-4f49-9288-85d83504b72c	2022/06/29/[$LATEST]ccfbc1c7337b42439006bfc99fb752cf	3000	128	0.375
6	2022-06-27T10:00:04.395Z	e8688d5c-da77-4415-b49a-8c10d10db5c9	2022/06/27/[$LATEST]844ccdbd90ee41c6949c634ab8bc32b7	3000	128	0.375
7	2022-06-29T06:00:05.227Z	4d38e63a-bd69-46ff-8b4e-ab8d0c1489f3	2022/06/29/[$LATEST]c6d9d9c1966a4b03b15b80b646ad0d18	2994	128	0.3743
8	2022-06-27T10:00:05.980Z	c4ac4e92-08c0-4116-9451-acc05cc4f38a	2022/06/27/[$LATEST]23ce1cd1ceb34e45950ffd45fbbe5d83	2956	128	0.3695
9	2022-06-29T10:00:05.676Z	0e11052a-7d48-4a56-aac0-7a804e895cf9	2022/06/29/[$LATEST]c6250ad11f0b4db89d8a695a741f777a	2919	128	0.3649

Figure 7.5: CloudWatch Insight

C.3 CloudWatch Log


CloudWatch > Log groups > /ecs/ubereats_spider
Log group details
Log streams
Metric filters
Subscription filters
Contributor Insights
Tags
Filter loaded log streams or try prefix search

/ecs/ubereats_spider

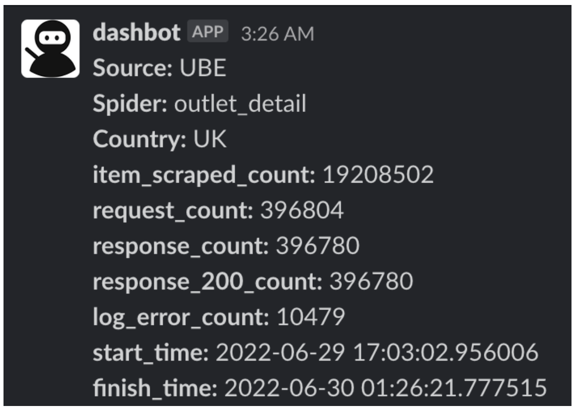
Log streams (100+)
By default, we only load the most recent log streams. Load more.

Log stream	Last event time
ecs/ubereats_spider/b5050d813e8c43e7b71641ca9022b2b3	2022-06-30 06:03:03 (UTC+02:00)
ecs/ubereats_spider/64cf79072a244e2a834d63bdc62f9c13	2022-06-30 06:00:41 (UTC+02:00)
ecs/ubereats_spider/7242f38154084a10b3ce06126958d665	2022-06-30 06:00:10 (UTC+02:00)
ecs/ubereats_spider/0947f32c160948979f00fe86ff3edf67	2022-06-30 05:57:38 (UTC+02:00)
ecs/ubereats_spider/8af026e11b394ab7b9a07df4b0dfe30a	2022-06-30 05:56:54 (UTC+02:00)
ecs/ubereats_spider/6e1b9e16c26b472e949d6982edd49a27	2022-06-30 05:50:51 (UTC+02:00)

Figure 7.6: CloudWatch Log

D Notification

D.1 Notification Report



Source: UBE
Spider: outlet_detail
Country: UK
item_scraped_count: 19208502
request_count: 396804
response_count: 396780
response_200_count: 396780
log_error_count: 10479
start_time: 2022-06-29 17:03:02.956006
finish_time: 2022-06-30 01:26:21.777515

Figure 7.7: Notification Report Example

E CI/CD with Github Actions

E.1 develop branch workflow

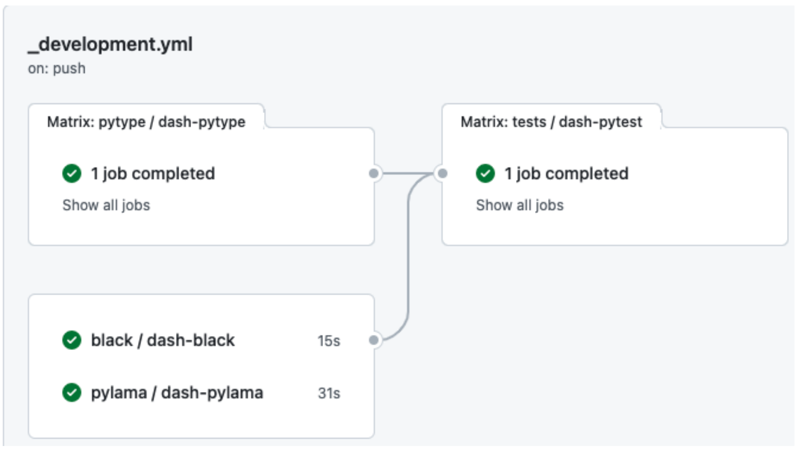
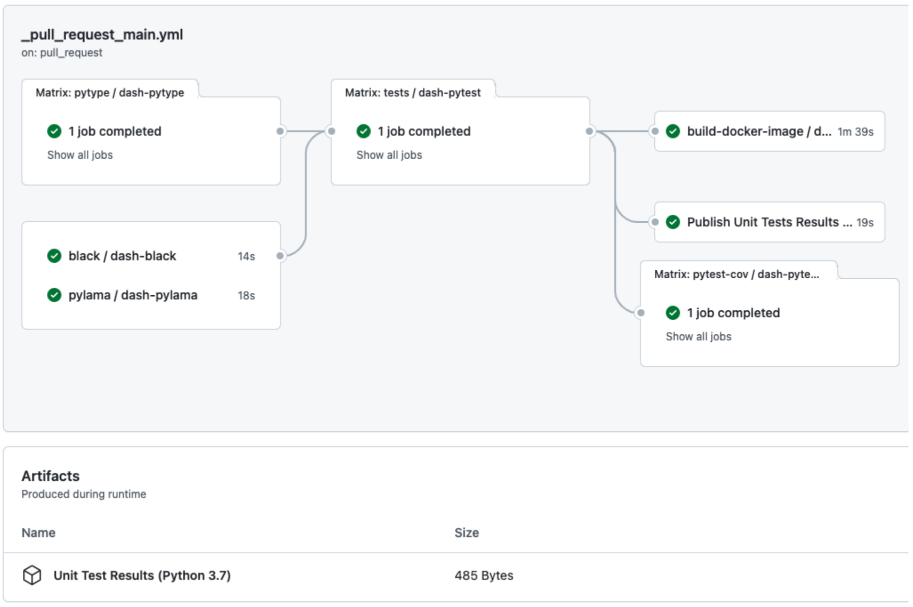


Figure 7.8: Develop Branch workflow in Github User Interface

E.2 main branch workflow



Name	Size
Unit Test Results (Python 3.7)	485 Bytes

Figure 7.9: Main Branch workflow in Github User Interface

E.3 release branch workflow

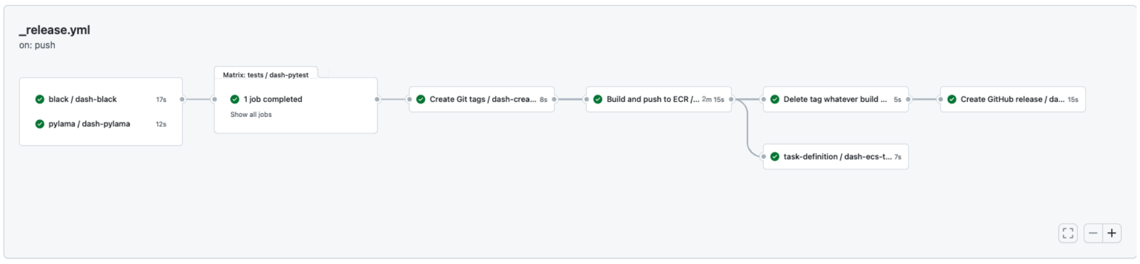
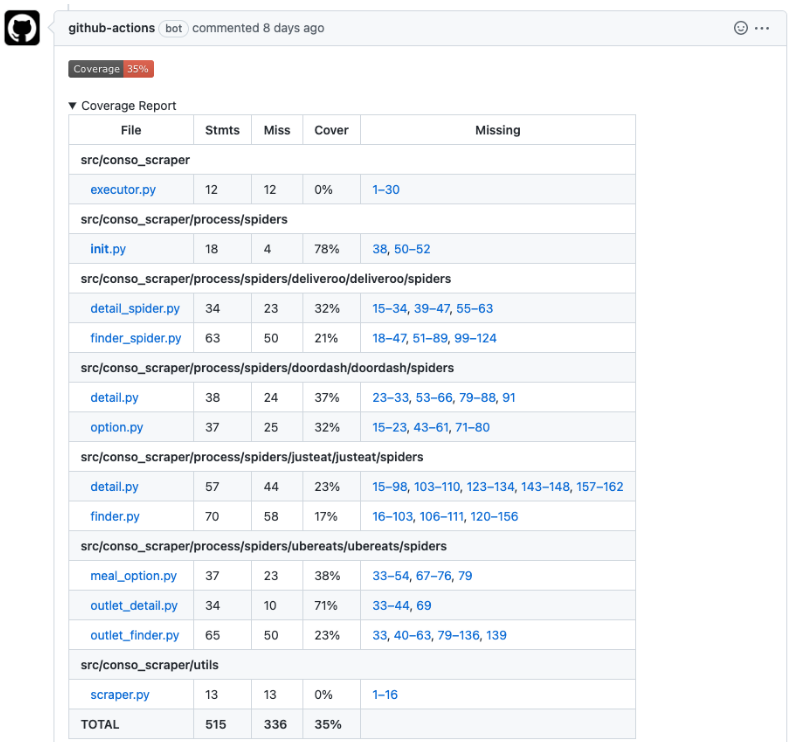

_release.yml
on: push
black / dash-black 17s
pylama / dash-pylama 12s
Matrix: tests / dash-pytest
1 job completed
Show all jobs
Create Git tags / dash-crea... 8s
Build and push to ECR /... 2m 15s
Delete tag whatever build ... 5s
Create GitHub release / da... 15s
task-definition / dash-ecs-t... 7s

Figure 7.10: Release Branch workflow in Github User Interface

E.4 Coverage Report Comment


github-actions bot commented 8 days ago

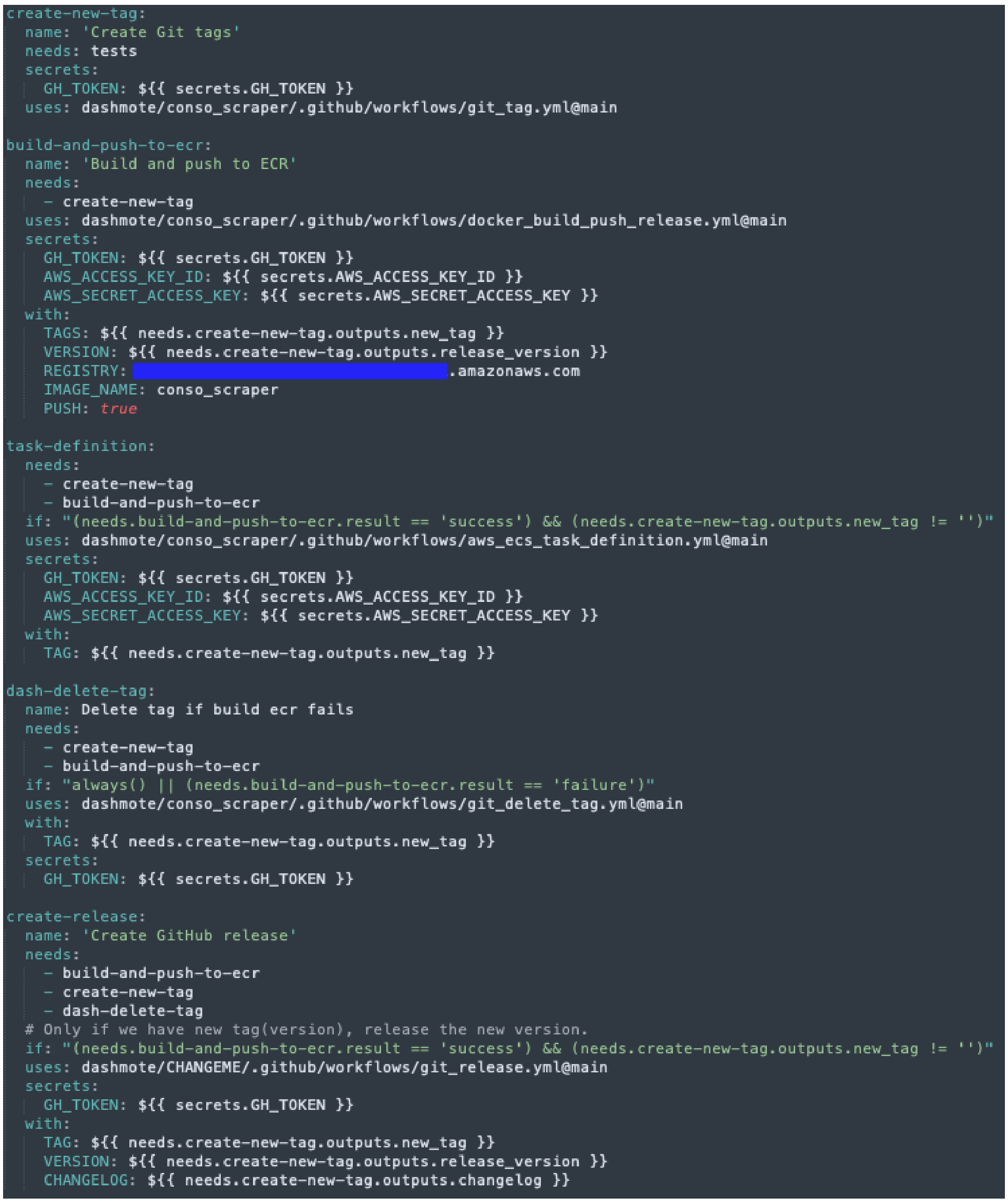
Coverage 35%

▼ Coverage Report

File	Stmts	Miss	Cover	Missing
src/conso_scraper				
executor.py	12	12	0%	1–30
src/conso_scraper/process/spiders				
init.py	18	4	78%	38, 50–52
src/conso_scraper/process/spiders/deliveroo/deliveroo/spiders				
detail_spider.py	34	23	32%	15–34, 39–47, 55–63
finder_spider.py	63	50	21%	18–47, 51–89, 99–124
src/conso_scraper/process/spiders/doordash/doordash/spiders				
detail.py	38	24	37%	23–33, 53–66, 79–88, 91
option.py	37	25	32%	15–23, 43–61, 71–80
src/conso_scraper/process/spiders/justeat/justeat/spiders				
detail.py	57	44	23%	15–98, 103–110, 123–134, 143–148, 157–162
finder.py	70	58	17%	16–103, 106–111, 120–156
src/conso_scraper/process/spiders/ubereats/ubereats/spiders				
meal_option.py	37	23	38%	33–54, 67–76, 79
outlet_detail.py	34	10	71%	33–44, 69
outlet_finder.py	65	50	23%	33, 40–63, 79–136, 139
src/conso_scraper/utils				
scraper.py	13	13	0%	1–16
TOTAL	515	336	35%	

Figure 7.11: Coverage Report Comment

E.5 Release YAML



```yaml
create-new-tag:
  name: 'Create Git tags'
  needs: tests
  secrets:
    GH_TOKEN: ${{ secrets.GH_TOKEN }}
  uses: dashmote/conso_scraper/.github/workflows/git_tag.yml@main

build-and-push-to-ecr:
  name: 'Build and push to ECR'
  needs:
    - create-new-tag
  uses: dashmote/conso_scraper/.github/workflows/docker_build_push_release.yml@main
  secrets:
    GH_TOKEN: ${{ secrets.GH_TOKEN }}
    AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
    AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
  with:
    TAGS: ${{ needs.create-new-tag.outputs.new_tag }}
    VERSION: ${{ needs.create-new-tag.outputs.release_version }}
    REGISTRY: [redacted].amazonaws.com
    IMAGE_NAME: conso_scraper
    PUSH: true

task-definition:
  needs:
    - create-new-tag
    - build-and-push-to-ecr
  if: "(needs.build-and-push-to-ecr.result == 'success') && (needs.create-new-tag.outputs.new_tag != '')"
  uses: dashmote/conso_scraper/.github/workflows/aws_ecs_task_definition.yml@main
  secrets:
    GH_TOKEN: ${{ secrets.GH_TOKEN }}
    AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
    AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
  with:
    TAG: ${{ needs.create-new-tag.outputs.new_tag }}

dash-delete-tag:
  name: Delete tag if build ecr fails
  needs:
    - create-new-tag
    - build-and-push-to-ecr
  if: "always() || (needs.build-and-push-to-ecr.result == 'failure')"
  uses: dashmote/conso_scraper/.github/workflows/git_delete_tag.yml@main
  with:
    TAG: ${{ needs.create-new-tag.outputs.new_tag }}
  secrets:
    GH_TOKEN: ${{ secrets.GH_TOKEN }}

create-release:
  name: 'Create GitHub release'
  needs:
    - build-and-push-to-ecr
    - create-new-tag
    - dash-delete-tag
  # Only if we have new tag(version), release the new version.
  if: "(needs.build-and-push-to-ecr.result == 'success') && (needs.create-new-tag.outputs.new_tag != '')"
  uses: dashmote/CHANGEME/.github/workflows/git_release.yml@main
  secrets:
    GH_TOKEN: ${{ secrets.GH_TOKEN }}
  with:
    TAG: ${{ needs.create-new-tag.outputs.new_tag }}
    VERSION: ${{ needs.create-new-tag.outputs.release_version }}
    CHANGELOG: ${{ needs.create-new-tag.outputs.changelog }}
```

Figure 7.12: Release Branch YAML (partial)

References

[1] **Amazon Elastic Container Registry (Amazon ECR)**. `https://aws.amazon.com/ecr/`. viii, 21

[2] **Amazon Elastic Container Service (Amazon ECS)**. `https://aws.amazon.com/ecs/`. viii, 21

[3] **What is a CI/CD pipeline?** `https://www.redhat.com/en/topics/devops/what-cicd-pipeline`. viii, 27

[4] **Gitflow Workflow**. `https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow`. viii, 31, 32

[5] Yangyong Zhu, Ning Zhong, and Yun Xiong. **Data explosion, data nature and dataology**. In *International Conference on Brain Informatics*, pages 147–158. Springer, 2009. 1

[6] Rabiyatou Diouf, Edouard Ngor Sarr, Ousmane Sall, Babiga Birregah, Mamadou Bousso, and Sény Ndiaye Mbaye. **Web scraping: state-of-the-art and areas of application**. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 6040–6042. IEEE, 2019. 1, 8

[7] Daniel Glez-Peña, Anália Lourenço, Hugo López-Fernández, Miguel Reboiro-Jato, and Florentino Fdez-Riverola. **Web scraping technologies in an API world**. *Briefings in bioinformatics*, **15**(5):788–797, 2014. 3, 7, 8, 10

[8] De S Sirisuriya et al. **A comparative study on web scraping**. 2015. 6

[9] SR Sreeja and Sangita Chaudhari. **Review of web crawlers**. *International Journal of Knowledge and Web Intelligence*, **5**(1):49–61, 2014. 7

[10] Seppe vanden Broucke and Bart Baesens. *Practical Web scraping for data science*. Springer, 2018. 7

[11] Brett Massimino. **Accessing online data: Web-crawling and information-scraping techniques to automate the assembly of research data**. *Journal of Business Logistics*, **37**(1):34–42, 2016. 7, 8, 11

[12] Shreya Upadhyay, Vishal Pant, Shivansh Bhasin, and Mahantesh K Pattanshetti. **Articulating the construction of a web scraper for massive data extraction**. In *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–4. IEEE, 2017. 7, 9, 10, 11

[13] **9 Web Scraping Challenges You Should Know**. `https://www.octoparse.com/blog/9-web-scraping-challenges`. 9

[14] Richard N Landers, Robert C Brusso, Katelyn J Cavanaugh, and Andrew B Collmus. **A primer on theory-driven web scraping: Automatic extraction of big data from the Internet for use in psychological research.** *Psychological methods*, **21**(4):475, 2016. 9

[15] Sean F McKenna. **Detection and classification of Web robots with honeypots**. Technical report, Naval Postgraduate School Monterey United States, 2016. 9

[16] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. **Web data extraction, applications and techniques: A survey**. *Knowledge-based systems*, **70**:301–323, 2014. 10

[17] Ryan Mitchell. *Web scraping with Python: Collecting more data from the modern web*. " O'Reilly Media, Inc.", 2018. 10, 11, 43

[18] **Whitepaper: A guide to web scraping at scale**. `https://www.zyte.com/whitepaper-ebook/scale-your-web-scraping/`. 10, 43

[19] David Eichmann. **Ethical web agents**. *Computer Networks and ISDN Systems*, **28**(1-2):127–136, 1995. 11

[20] **Compare AWS and Azure services to Google Cloud**. `https://cloud.google.com/free/docs/aws-azure-gcp-service-comparison`. 12

[21] **AWS Step Functions Overview**. `https://www.datadoghq.com/knowledge-center/aws-step-functions/`. 14

[22] **Top four benefits of Microsoft Azure Logic Apps**. `https://www.influentialsoftware.com/top-four-benefits-of-microsoft-azure-logic-apps/`. 14

[23] Devon Peticolas, Russell Kirmayer, and Deepak Turaga. **Mímir: Building and Deploying an ML Framework for Industrial IoT**. In *2019 International Conference on Data Mining Workshops (ICDMW)*, pages 399–406. IEEE, 2019. 14, 39

[24] Alexandar P Mechev, JBR Oonk, Timothy Shimwell, Aske Plaat, HT Intema, and HJA Rottgering. **Fast and reproducible lofar workflows with aglow**. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 136–144. IEEE, 2018. 14

[25] Laurens Versluis, Erwin Van Eyk, and Alexandru Iosup. **An analysis of workflow formalisms for workflows with complex non-functional requirements**. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 107–112, 2018. 14

[26] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, et al. **Pegasus, a workflow management system for science automation**. *Future Generation Computer Systems*, **46**:17–35, 2015. 14

[27] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. **A survey of data-intensive scientific workflow management**. *Journal of Grid Computing*, **13**(4):457–493, 2015. 14

[28] G Kousalya, P Balakrishnan, and C Pethuru Raj. *Automated workflow scheduling in self-adaptive clouds: Concepts, algorithms and methods*. Springer, 2017. 14

[29] **Serverless Architecture AWS Lambda: 2 Comprehensive Criteria**. `https://hevodata.com/blog/serverless-architecture-aws-lambda/`. 20

[30] **Amazon ECS launch types**. `https://docs.aws.amazon.com/AmazonECS/latest/developerguide/launch_types.html`. 20

[31] **Amazon ECS task definitions**. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definitions.html. 21

[32] **What is CI/CD?** https://about.gitlab.com/topics/ci-cd/. 27