Optional Project

Introduction

Following you can find the documentation for the optional project of the course IC4302 - Databases II. The project consists of three main components: the S3 Spider, the Downloader, and the SparkJob. These components work together to process data from an S3 bucket, retrieve information from an external API, and perform data transformations and indexing using Spark and Elasticsearch. The project is designed to run in a Kubernetes environment and utilizes RabbitMQ for messaging, MariaDB for database operations, and various Python and Scala libraries for data processing.

The project aims to provide students with hands-on experience in working with real-world data processing scenarios, integrating multiple technologies, and deploying applications in a containerized environment.

Team Members

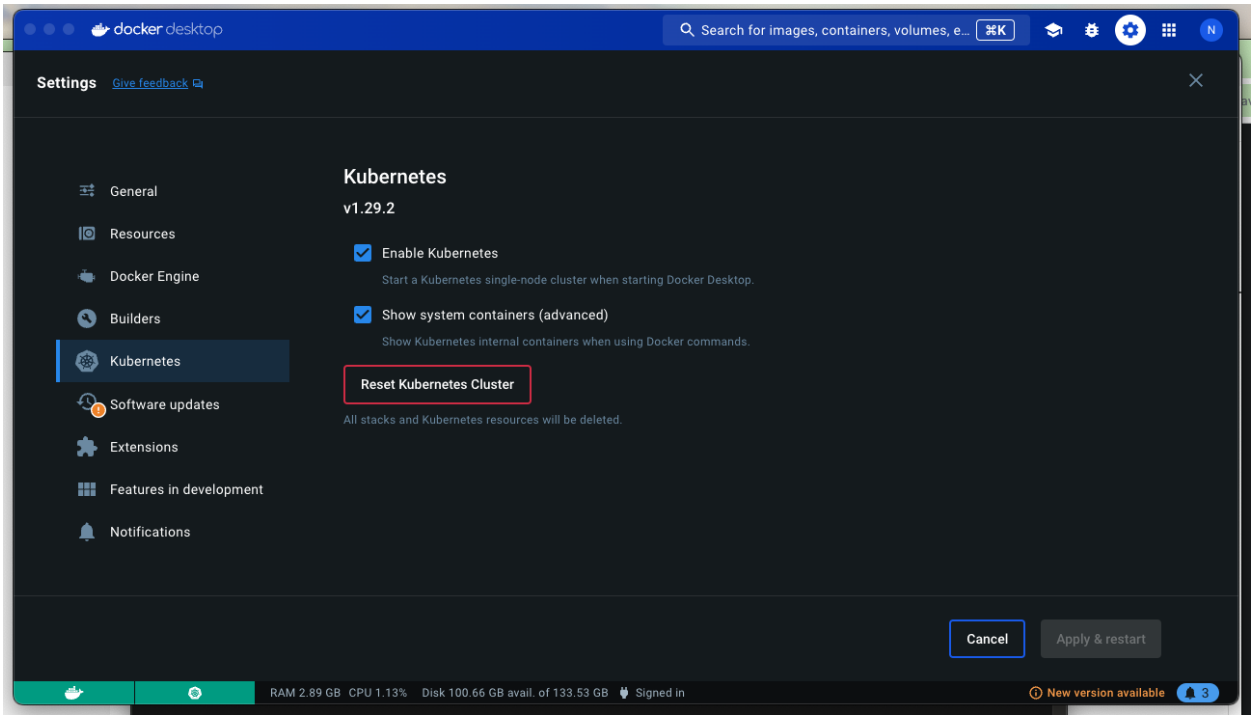
- Victor Aymerich
- Anthony Barrantes
- Fabricio Solis
- Melanie Wong
- Pavel Zamora

Next up, you will find the requirements to run the project, the steps to execute it, testing examples, unit tests and the recommendations and conclusions we have gathered from the project.

Requirements

The requirements for the project are the following:

- Create an user in DockerHub
- Install Docker Desktop, if you are using MacOS, please make sure you select the right installer for your CPU architecture.
- Open Docker Desktop, go to **Settings > Kubernetes** and enable Kubernetes.



K8s

- Install Kubectl
- Install Helm
- Install Visual Studio Code
- Install Lens

AWS Access Information

Bucket: 2024-02-ic4302-gr1

Access Key: AKIAQ2VOGXQDTWAX4PUY

Secret Key: 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

After installing all the necessary requirements described before, in order to execute the optional project you must proceed with the following steps:

Building the docker images

There is a script to build the docker images, to execute it in a bash shell execute:

```
cd ./PO/docker
./build.sh nereo08
```

Change **nereo08** to your DockerHub username

Please take a look on the script contents to make sure you understand what is done under the hood. This script will build the images for the S3 Spider, Downloader and SparkJob components. They are the base of the project. It is important to have the DockerHub username set correctly in the script to ensure the images are pushed to the correct repository.

Helm Charts

Configure

- Open the file **PO/charts/application/values.yaml**
- Replace **nereo08** by your DockerHub username

```
config:
  docker_registry: nereo08
```

Install

Execute:

```
cd ./PO/charts
./install.sh
```

Here we are installing the components in the Kubernetes cluster. The script will install the RabbitMQ, MariaDB, and Elasticsearch components, as well as the S3 Spider, Downloader, and SparkJob components. It is important to have the DockerHub username set correctly in the script to ensure the images are pulled from the correct repository.

Uninstall

Execute:

```
cd ./PO/charts
./uninstall.sh
```

This script will uninstall the components from the Kubernetes cluster. It is important to have the DockerHub username set correctly in the script to ensure the correct components are removed.

Access Debug Pod

```
# copy the name that says debug from the following command
kubectl get pods
# then replace debug-844bb45d6f-9jt45 by that name
kubectl exec --stdin --tty debug-844bb45d6f-9jt45 -- /bin/bash
```

In case you need to access the debug pod to check the logs or execute some commands, you can use the previous command to access it.

Execute Spark

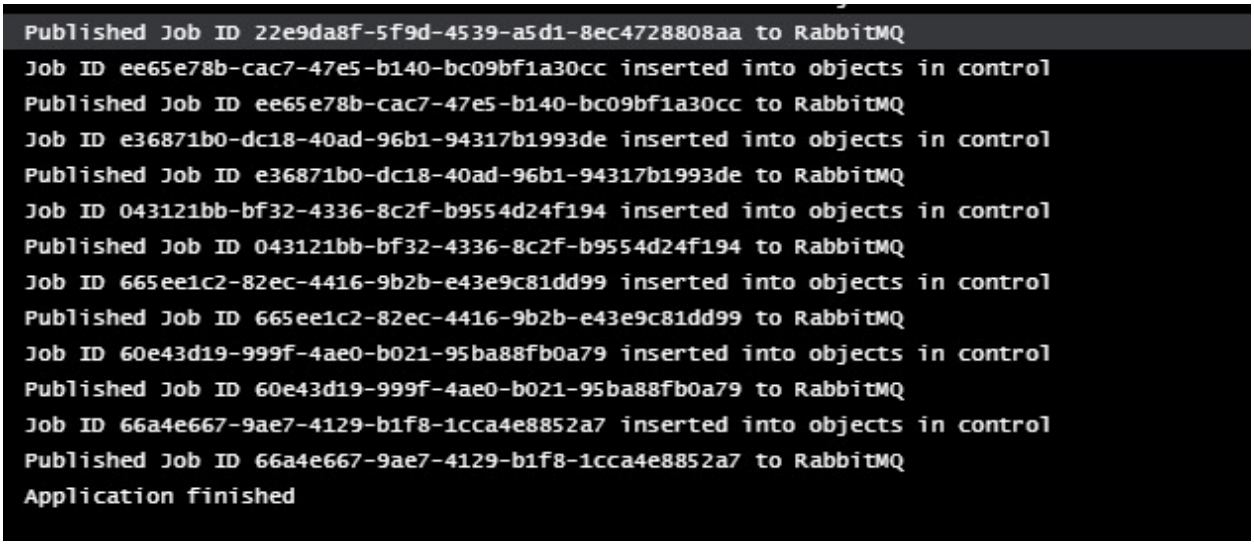
```
cd /opt/spark/
bin/spark-shell
```

Now that Spark Shell is up and running, you can execute the contents of the file **PO/charts/application/scala/app.scala**

Testing

To test the whole project together, you can follow the next steps: First you need to do the build and install steps, after that to verify that the project is working correctly you can follow the next steps:

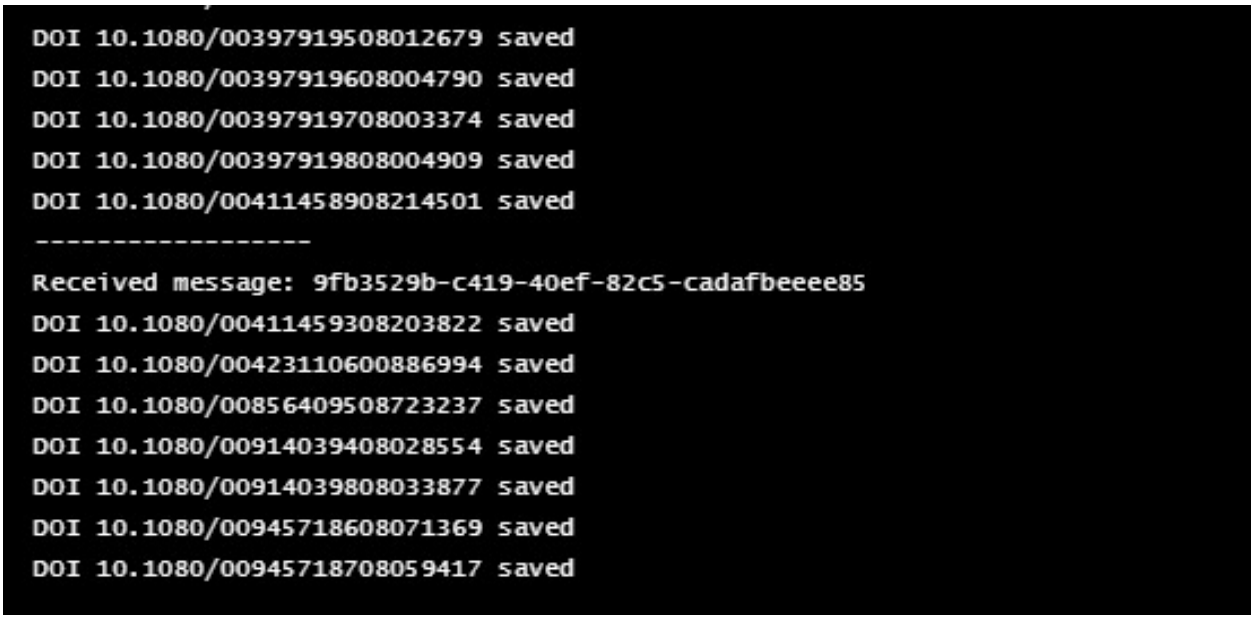
Check Docker images are running, you can check this in the Docker Desktop application. In this area it is important to check the different images, you can check the RabbitMQ, MariaDB, and Elasticsearch pods are running in the Kubernetes cluster.



Spider S3 Logs

Atter that you can check in Lens that the pods are running correctly, here you can check the logs of the pods to see if there are any errors and the status of the pods.

After that you can check the logs of the downloader pod to see if the data is being processed correctly.



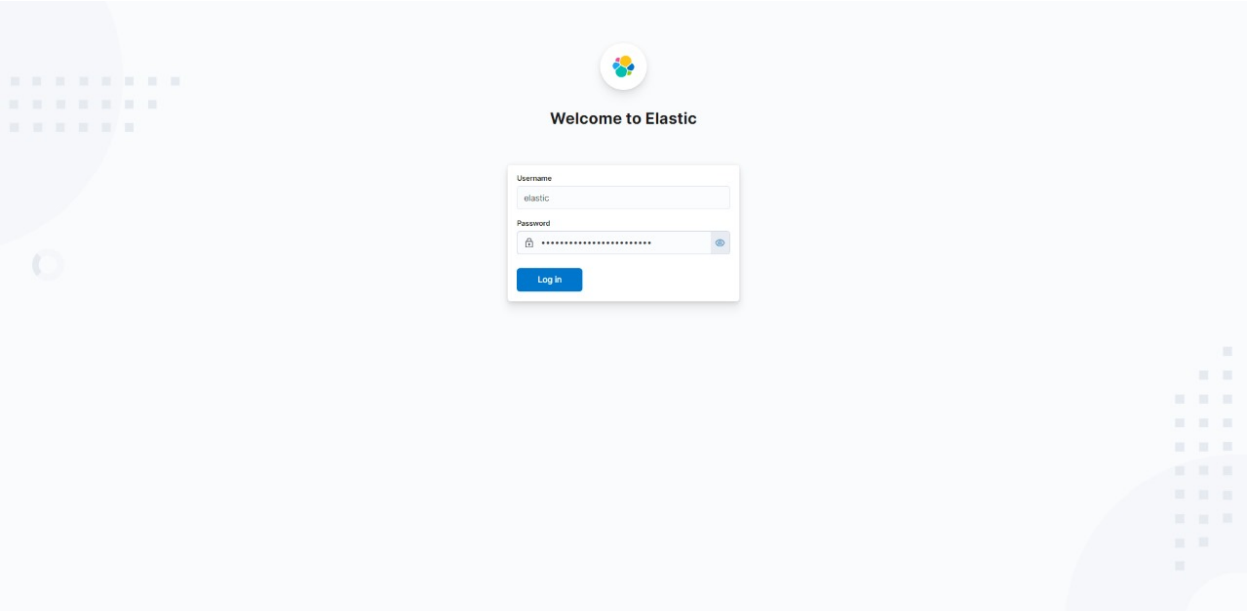
Downloader Logs

Finally you can check the logs of the SparkJob pod to see if the data is being indexed correctly in Elasticsearch.

```
24/08/24 05:30:12 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://spark-job-28741290-twpdt:4040
Spark context available as 'sc' (master = local[*], app id = local-1724477430890).
Spark session available as 'spark'.
Starting the Spark job...
SparkSession initialized successfully
Found 422 JSON files in the directory: /data
Processing file: /data
```

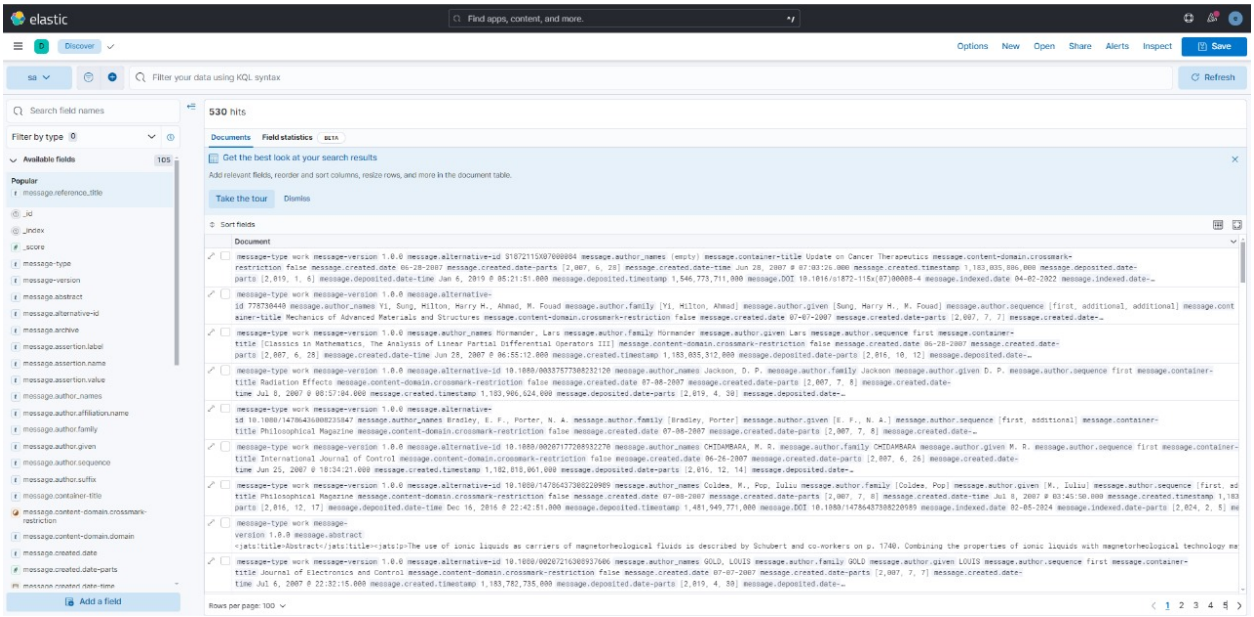
SparkJob Logs

Once you have checked all the logs and the data is being processed correctly, you can consider to go to Elasticsearch by forwarding the IC4302 KB pod and here check if the data is being indexed correctly. The user on Elasticsearch is **elastic** and the password can be obtained from the spark pod's environment variables.



ElasticSearch Main Page

Once you enter the Elasticsearch, you head towards the Discover tab and create a data visualization to see the data that is being indexed. This can be done by creating a new index pattern using the name "data" and creating the new visualization to see the stored information.



Elasticsearch

Recommendations and Conclusions

Recommendations

1 **Learn Python Programming**

- **Recommendation**: It is important to start by learning Python programming, knowing how to run scripts and understand code structures will help you follow and understand what is being done.
- **Reason**: The S3 Spider component and the Downloader component are written in Python, so understanding the basics will help you follow the code and make minor adjustments if any change is needed.

2 **Familiarize Yourself with Environment Variables**

- **Recommendation**: You also should understand what environment variables are, how they work and how to set them up. You will need to set certain environment variables for the components of the project to work correctly, for example, some database credentials and API URLs.
- **Reason**: Environment variables are used to pass important configuration settings to the application without hardcoding them into the script.

3 **Learn the Basics of SQL**

- **Recommendation**: Getting a basic understanding of SQL is highly recommended, it is the language used to interact with the databases. You should learn how to run queries, such as selecting data from tables or updating records and understand the functioning of them.
- **Reason**: The project in multiple sections interacts with databases, mainly MariaDB, and understanding SQL will help you see how it inserts, retrieves and updates data.

4 **Get Familiar with MariaDB**

- **Recommendation**: Learn how to access and use MariaDB, which the the main database used. You should learn the basics, including knowing how to connect to it, view tables, and check data for example.
- **Reason**: You will need to know how to connect to MariaDB to ensure certain components are working correctly, as well as to verify that data is being updated and stored properly.

5 **Understand RabbitMQ Basics**

- **Recommendation**: Learn what RabbitMQ is, what is it used for and how it works as a message broker that passes messages between different parts of an application.
- **Reason**: The S3 Spider and Downloader both work beside RabbitMQ, sending and consuming messages with RabbitMQ, so understanding this process will help you see how jobs are being processed.

6 **Learn How to Use a Kubernetes Cluster**

- **Recommendation**: It is very important to get a basic understanding of Kubernetes, what is their function and how it works, as the whole project is designed to run in a Kubernetes cluster.
- **Reason**: Knowing how to deploy and manage the application in Kubernetes will help you run the project in a real-world environment.

7 **Understand API Interactions**

- **Recommendation**: It is important to learn what an API (Application Programming Interface) is, and understand how a component uses it to fetch data from the CrossRef API.
- **Reason**: The downloader interacts with the CrossRef API to retrieve information for each DOI. Understanding this process is key to seeing how the component works.

8 **Learn How to Use Helm Charts**

- **Recommendation**: Learn how to use Helm charts, which are used to deploy the application in Kubernetes. It is relevant to understand how they work and what it their function so you can deploy the application correctly.
- **Reason**: The project uses Helm charts to deploy the application in Kubernetes. Knowing how to use Helm charts will help you manage the deployment of the application.

9 **Understand Spark and Scala Basics**

- **Recommendation**: Learn the basics of SparkSQL and Scala, as the project uses Spark to process data and Scala is the language used to write the SparkJob consequently. Basic knowledge of these technologies will help you understand the code and make changes if necessary.
- **Reason**: In the SparkJob component, the code is written in Scala so for you to understand the structure and how everything works basic knowledge is needed.

10 **Get Familiar with Elasticsearch Indexing**

- **Recommendation**: Study the basics of Elasticsearch, focusing on how to create and manage indices where data will be stored.
- **Reason**: After transforming the data with Spark SQL, the results are saved in an Elasticsearch index. Understanding Elasticsearch indexing will help you verify that the data is being stored correctly and know how to retrieve it.

11 **Learning the difference between CronJob and a Deployment**

- **Recommendation**: Understand the difference between a CronJob and a Deployment in Kubernetes, as the project uses both to run the components.
- **Reason**: The project uses a CronJob to run the S3 Spider and SparkJob components at specific times, and a Deployment to run the Downloader component. Knowing the difference between the two will help you understand how the components are being executed.

12 **Ensure Environment Variables Are Set**

- **Recommendation**: Verify that the environment variables (`XPATH`, `ELASTIC_USER`, `ELASTIC_PASS`, `ELASTIC_URL`) are properly set before running the script.
- **Reason**: These variables are essential for the script to locate the JSON data file and connect to Elasticsearch. Missing or incorrect variables can lead to errors during execution.

13 **Check Elasticsearch Connectivity**

- **Recommendation**: Confirm that the Elasticsearch cluster is accessible from the machine running the script and that the provided URL (`ELASTIC_URL`) is correct.
- **Reason**: Proper connectivity ensures that Spark can successfully index and retrieve data from Elasticsearch, avoiding connection failures.

14 **Verify Spark and Elasticsearch Versions**

- **Recommendation**: Ensure that the versions of Spark and Elasticsearch are compatible with the Elasticsearch Spark connector being used.
- **Reason**: Compatibility issues can lead to runtime errors or unexpected behavior. Using compatible versions avoids integration problems.

15 **Validate JSON Data Path**

- **Recommendation**: Double-check the JSON data path provided in the `XPATH` environment variable to make sure it points to a valid JSON file.
- **Reason**: An incorrect path will cause errors when the script attempts to read the data, preventing the script from executing successfully.

16 **Review Elasticsearch Index Settings**

- **Recommendation**: Verify that the index settings and parameters in the `SparkConf` configuration match your Elasticsearch setup.
- **Reason**: Ensuring that settings like `es.index.auto.create` and authentication parameters are correct helps avoid indexing issues and ensures proper data handling.

17 **Optimize Spark Configurations**

- **Recommendation**: Review and adjust Spark configurations for performance, such as memory settings and parallelism, based on your dataset size.
- **Reason**: Proper configurations can enhance the performance of Spark jobs, especially for large datasets, leading to faster execution times.

18 **Check Data Format and Consistency**

- **Recommendation**: Ensure that the JSON data is well-formatted and consistent with the expected schema used in SQL queries.
- **Reason**: Inconsistent or improperly formatted data can lead to errors during data processing and

querying.

19 **Monitor Resource Usage**

- **Recommendation**: Keep an eye on resource usage (CPU, memory) while running the script to ensure it operates within acceptable limits.
- **Reason**: Monitoring helps prevent performance bottlenecks and ensures that the script runs efficiently, particularly for large datasets.

20 **Test with Sample Data**

- **Recommendation**: Run the script with a small sample of data before processing the full dataset to validate the functionality.
- **Reason**: Testing with smaller datasets helps identify and fix issues early, reducing the risk of errors during full-scale execution.

21 **Document Configuration and Steps**

    - **Recommendation**: Document the configuration parameters and execution steps clearly for future reference and ease of use.
    - **Reason**: Clear documentation helps users understand the setup and execution process, making it easier to troubleshoot and replicate the environment.

Conclusions

1 **Optional Projects Enhance Learning and Skill Development** Projects being part of a course make learning much more rewarding, as they allow practical applications of the theoretical knowledge gained. However, optional projects like this one provide an additional layer of complexity and challenge, which can significantly enhance learning and skill development. This project, in particular, covers a wide range of technologies and concepts we had not seen before, consequently not having knowledge about them, making it an excellent opportunity for us to develop our understanding and expertise in various areas.

2 **Diverse Databases Require Specialized Skills and Techniques** The project highlights the necessity of acquiring distinct skills and techniques to manage various types of databases effectively. By working with various databases, we gained valuable experience in handling different database systems, understanding their unique features, and optimizing data storage and retrieval processes. This exposure to diverse databases can help us develop a well-rounded skill set and adapt to different data management requirements in our future careers.

3 **Spider/Crawler Tools Simplify Data Acquisition from Multiple Sources** The use of Spider/Crawler tools within the project demonstrates their effectiveness in automating data collection from various sources such as APIs, web pages, and data storage buckets. These tools significantly reduce the time and effort required for data gathering, making them indispensable for large-scale data processing tasks.

4 **Collaboration Across Technologies Enhances Project Outcomes** Integrating various technologies, such as RabbitMQ for messaging, Kubernetes for deployment, and Elasticsearch for data indexing, demonstrates the importance of cross-technology collaboration. This project demonstrated to us how combining different tools and platforms can result in a more appropiate and efficient system all together.

5 **Scalability is Essential for Modern Applications** The project emphasizes the need for scalability in modern applications. By leveraging Kubernetes and containerization, the project ensures that the system can handle increased loads and expand as needed, which has been demonstrated, is vital for maintaining performance in dynamic environments.

6 **Continuous Learning and Adaptation are Key to Success** Learning and adapting to new

technologies and methodologies are essential for success in the rapidly evolving field of data science and software development, which is exactly our degree. This project provides an excellent opportunity for us to enhance skills, explore new technologies, and learn about the latest trends in the industry.

7 **Real-World Applications Provide Valuable Experience** Working on projects that simulate real-world scenarios provides valuable experience and prepares students like us for the challenges we may face in our professional careers. This project offers a practical learning experience that can help develop skills and knowledge needed to succeed in the field of data science and software development. In our professional careers we will face similar challenges where a application uses multiple technologies we do not know about to learning will be constant, for that reason this project is a great opportunity to learn how to solve them.

8 **Feedback and Iteration Improve Project Outcomes** Working together and getting feedback from your teammates are key elements to making any project a success. When classmates team up, we can mix our habilities, share ideas, and tackle the challenges. Receiving feedback from our colleagues and teachers can help us identify areas for improvement and refine our work to achieve better outcomes. This iterative process of working together and improvement is essential for developing high-quality projects and achieving success in the project.

9 **Reading and Understanding Code is Essential** Reading and understanding code is an essential skill for Computer Science majors as us. By reviewing and analyzing the codebase of this project, we could gain insights into best practices, coding standards on the technologies we did not know about, and design patterns used in real-world applications. This experience helped us improve our coding skills, learning new techniques, and developing a deeper understanding of various components of the project.

10 **Documentation and Learning to Search are Key** Documentation and learning how to navigate the internet in search of solutions are essential skills for students like us. By documenting the work, we can keep track of our progress, plus it helps with sharing our knowledge with others, and refering back to previous solutions if needed. Additionally, knowing how to search for information online may help us find solutions to problems we encounter, learn information about technologies we are not familiar with, and integrating new tools into our projects. The more information we can find, the more we can learn and apply to our projects.

11 **Spark and Elasticsearch Configuration** The code configures a `SparkConf` to connect to an Elasticsearch cluster and sets the required credentials.

12 **Initialization of SparkContext and SparkSession** `SparkContext` and `SparkSession` are initialized using the previously defined configuration.

13 **Data Reading** Data is read from a JSON path specified in the environment variable `XPATH`.

14 **SQL Queries** SQL queries are executed to format dates and concatenate author names.

15 **Saving Data to Elasticsearch** The results of the SQL queries are saved to Elasticsearch using the `data` index.

16 **Usage of** `SQLContext` Although `SQLContext` is created, it is not explicitly used in the script. It may be possible to remove it if only `SparkSession` is used.

17 **Indexing in Elasticsearch** Data is indexed in Elasticsearch with the `es.index.auto.create` option enabled, allowing automatic index creation if it does not exist.

18 **Environment Variables** Elasticsearch configuration relies on environment variables, which is a good practice for maintaining flexibility and security.

19 **Date Formats** Date transformations in SQL queries ensure dates are stored in a readable format.

20 **Name Concatenation** Author names are concatenated to consolidate information into a single field.

Components

S3 Spider

Overview

This script automates the processing of objects stored in an Amazon S3 bucket. For each object, it extracts DOIs, creates "jobs" with a list of DOIs, saves these jobs to a MariaDB database, and publishes the unique job IDs to RabbitMQ. Once an object is processed, it is marked in the database to prevent it from being processed again.

The script performs the following key steps:

1. All files are fetched from the S3 bucket.
2. For each file, if it has not been processed: - The file content is fetched. - DOIs are extracted and jobs are created. - Each job is inserted into the MariaDB database and its ID is published to RabbitMQ. - The file is marked as processed.
3. The connection to RabbitMQ is closed and the application terminates.

Configuration

- **RabbitMQ**: A connection is established with RabbitMQ using the provided credentials. A queue with the name specified in RABBITMQ_QUEUE is declared.
- **S3 Client Configuration**: The S3 client is configured using AWS credentials. If the credentials are not configured correctly, the script will terminate with an error.

Dependencies

- **boto3**: The AWS SDK for Python, used to interact with Amazon S3.
- **pika**: A RabbitMQ client library for Python, used to interact with RabbitMQ.
- **mariadb**: A MariaDB client library for Python, used to interact with MariaDB.
- **uuid**: A library used to generate unique identifiers for jobs.
- **re**: A library used to work with regular expressions in Python.
- **os**: A library used to interact with the operating system and handle environment variables.
- **sys**: A library used to work with system-specific parameters and functions.
- **time**: A library used to work with time-related functions in Python.

Environment Variables

- **RABBITMQ**: RabbitMQ server address.
- **RABBITMQ_PASS**: Password to access RabbitMQ.
- **RABBITMQ_QUEUE**: Name of the RabbitMQ queue to which the messages will be sent.
- **BUCKET**: Name of the S3 bucket where the objects to be processed are located.
- **ACCESS_KEY** and *SECRET_KEY*: AWS access credentials.
- **MARIADB_USER**: MariaDB user.
- **MARIADB_PASS**: MariaDB password.
- **MARIADB**: MariaDB server address.
- **MARIADB_DB**: Name of the database in MariaDB.
- **MARIADB_TABLE**: Name of the table where jobs are stored in MariaDB.
- **PROCESSED_TABLE**: Name of the table where processed objects are marked.

Main Functions

- **mariadb_connection()** Establishes a connection to the MariaDB database and returns the connection object and cursor.
- **is_object_processed(file_key)** Checks if an object in S3 has already been processed using a stored procedure in MariaDB.
- **mark_object_as_processed(file_key)** Marks an object as processed in MariaDB using a stored procedure, this storages the key of the object in the table processed_objects.

- **extract_dois(text)** Extracts valid DOIs from the contents of a file using a regular expression.
- **get_file_content(bucket, key)** Gets the contents of a file stored in an S3 bucket.
- **create_jobs(data, job_size)** Creates "jobs" by splitting the list of DOIs into batches of the specified size. In the program case we use 10 job size.
- **list_all_objects(bucket)** Lists all objects in an S3 bucket, handling pagination if necessary.
- **insert_job_to_mariadb(job_id, dois)** Inserts job information into the MariaDB database using a stored procedure.

Execution of Tests

Test execution happens while the Docker image is being built, but this is an example of running the downloader unit tests.

```
sfabricito@DESKTOP-67IMC3K:~/Documents/University/Bases_de_datos_II/PO$ py docker/s3-spider/app/test.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

Spider

–

testing

Downloader

Overview

The downloader is a Python application designed to process jobs from a message queue, interact with a database, retrieve data from an external API, and store the results. It operates within a Kubernetes environment and performs the following key steps:

1. **Message Consumption**:
   - The downloader connects to a RabbitMQ queue, where it waits for messages. Each message represents a job that needs to be processed.
2. **Database Interaction**:
   - When a job message is received, the downloader connects to a MariaDB database and updates the job status to "in-progress". This ensures that the job is marked as active while it is being processed.
   - It then retrieves a list of DOIs (Digital Object Identifiers) associated with the job from the database.
3. **API Data Retrieval**:
   - For each DOI, the downloader makes a call to the CrossRef API using the DOI as a parameter. This API returns metadata about academic articles, books, or other scholarly content in JSON format.
   - The JSON response from the API is then saved to disk as a file, with the filename being an MD5 hash of the DOI.
4. **Handling Missing Data**:
   - If the API call does not return data for a particular DOI, that DOI is ignored, and its information is recorded in a "skipped" list within the MariaDB database.

5 **Job Completion**:

- After processing all DOIs associated with a job, the downloader updates the job status in MariaDB to "done" and records the completion time.
- The downloader then waits for the next job message from the RabbitMQ queue and repeats the process.

Configuration

The Downloader component requires configuration settings for RabbitMQ, MariaDB, and the CrossRef API. These settings are provided through environment variables, which specify the connection details for each service.

Dependencies

- **pika**: A Python RabbitMQ client library that allows the downloader to interact with the message queue.
- **mariadb**: A MariaDB client library that enables the downloader to connect to the database and execute queries.
- **requests**: A library used to make HTTP requests to the CrossRef API and retrieve data.
- **hashlib**: A library used to generate MD5 hashes for file names.
- **os**: A library used to interact with the operating system and handle file operations.
- **json**: A library used to parse JSON responses from the CrossRef API.
- **datetime**: A library used to work with dates and times in Python.

Environment Variables

The Downloader component requires the following environment variables to be set:

- **RABBITMQ_HOST**: The hostname of the RabbitMQ server.
- **RABBITMQ_USER**: The username for authenticating with the RabbitMQ server.
- **RABBITMQ_PASS**: The password for authenticating with the RabbitMQ server.
- **RABBITMQ_QUEUE**: The name of the RabbitMQ queue to consume messages from.
- **DB_HOST**: The hostname of the MariaDB server.
- **DB_PORT**: The port number of the MariaDB server.
- **DB_USER**: The username for authenticating with the MariaDB server.
- **DB_PASS**: The password for authenticating with the MariaDB server.
- **DB_NAME**: The name of the MariaDB database to store the data in.

Main Functions

- create_connection_pool(): Creates a connection pool to the MariaDB database using the provided credentials.
- execute_query(query, params=None): Executes a SQL query on the MariaDB database using the provided parameters.
- get_doi_information(doi_id): Retrieves information about a DOI from the CrossRef API.
- save_json(name, data): Saves JSON data to a file with the specified name.
- update_info(job_id,doi): Updates the status of a job in the MariaDB database.
- callback(ch, method, properties, body): Callback function that processes messages from the RabbitMQ queue.

Unit Testing

This unit test isolates the following functions and checks its correctness by providing various input values and

asserting the corresponding output.

- **test_get_doi_information()**: Tests the **get_doi_information(doi_id)** function to confirm that it accurately retrieves and processes DOI information.
- **test_save_json()**: Verifies the **save_json(name, data)** function's ability to correctly serialize data into a JSON file and store it at the designated location.

Unit Testing

Unit test verifies the correct behavior of vital individual functions to ensure that the Spider works properly in execution time.

- *test_extract_dois(): Test the behavior of the **extract_dois(text)* with a example text.
- *test_create_jobs(): Ensure that the function **create_job(data, job_size)* is working perfectly.

Execution of Tests

Test execution happens while the Docker image is being built, but this is an example of running the downloader unit tests.

```
sfabricito@DESKTOP-67IMC3K:~/Documents/University/Bases_de_datos_II/PO$ py docker/downloader/app/test.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

Downloader

–

testing

SparkJob

Overview

The Spark job is a Scala script designed to process JSON data, perform SQL transformations, and interact with Elasticsearch for indexing. It operates within a Spark environment and performs the following key steps:

1. **Spark Configuration**:
   - The job initializes `SparkConf` to set up the configuration for connecting to Elasticsearch. It configures essential settings such as Elasticsearch nodes, authentication details, and index creation options.
2. **Context Initialization**:
   - The script creates a `SparkContext` using the configured `SparkConf`. This context is crucial for executing Spark operations.
   - A `SparkSession` is then initialized using the `SparkContext`. This session serves as the entry point for working with Spark SQL.
3. **Data Reading**:
   - JSON data is read from a path specified by the environment variable `XPATH`. The data is loaded into a DataFrame using `spark.read.json`.
4. **SQL Transformations**:

- The script defines and executes SQL queries to transform the data:
    - **Date Formatting**: It formats dates in the `message.indexed.date-time` and `message.created.date-time` fields into `MM-dd-yyyy`.
    - **Name Concatenation**: It concatenates the `message.author.family` and `message.author.given` fields into a single `message.author_names` field.

5 **Data Indexing**:

- The transformed data is saved to Elasticsearch using the `saveToEs` method. The data is indexed under the `data` index.

6 **Resource Management**:

- After processing, the `SparkContext` and `SparkSession` are stopped to release resources.

Configuration

Dependencies

The Spark job requires the following dependencies to be included in the project:

- **elasticsearch-spark-30_2.12-8.14.3**: The Elasticsearch connector for Spark 3.0, used to interact with Elasticsearch.

Environment Variables

The Spark job requires the following environment variables to be set:

- **XPATH**: The path to the JSON data file.
- **ELASTIC_USER**: The username for authenticating with the Elasticsearch cluster.
- **ELASTIC_PASS**: The password for authenticating with the Elasticsearch cluster.
- **ELASTIC_URL**: The URL of the Elasticsearch cluster.

References

- [1] "Dockerfile reference," Docker Documentation. [Online]. Available: https://docs.docker.com/reference/dockerfile/#overview. [Accessed: Aug. 15, 2024].
- [2] "kubectl commands," Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands. [Accessed: Aug. 15, 2024].
- [3] "MariaDB Documentation," MariaDB Knowledge Base. [Online]. Available: https://mariadb.com/kb/en/documentation/. [Accessed: Aug. 15, 2024].
- [4] "RabbitMQ Tutorials," RabbitMQ. [Online]. Available: https://www.rabbitmq.com/tutorials. [Accessed: Aug. 15, 2024].
- [5] "Spark SQL, DataFrames and Datasets Guide," Apache Spark. [Online]. Available: https://spark.apache.org/docs/latest/sql-programming-guide.html. [Accessed: Aug. 15, 2024].