Phase 1

CSE354, Distributed Computing

Name:	Ali Tarek Abdelmonim	ID:	21P0123
Name:	Mohamed Mostafa Mamdouh	ID:	21P0244
Name:	Mohamed Walid Helmy	ID:	21P0266
Name:	Tsneam Ahmed Eliwa Zky	ID:	21P0284

Presented To: Dr Ayman Bahaa Dr Hossam Mohamed Eng. Alaa Hamdy Eng. Mostafa Ashraf

Phase No: (1) Group No.(11)

Date: 27 / 4 /2025

Table of Contents

Table of figures

Introduction

The advancement of technology has led to an exponential growth in the volume of online information, creating both opportunities and challenges in accessing and organizing data efficiently. To address this, our project focuses on designing and developing a Distributed Web Crawling and Indexing System using Cloud Computing.

This project aims to leverage distributed computing to perform large-scale web crawling and indexing tasks. By utilizing cloud-based virtual machines and distributed task queues, the system will efficiently crawl websites, process their contents, and create a searchable index of web pages. Emphasis is placed on ensuring fault tolerance and scalability, allowing the system to maintain functionality despite node failures and to adapt to increasing workloads.

In Phase 1, we establish the foundation of this project by refining the requirements, finalizing the technology stack, designing the system architecture, and setting up the cloud environment. This phase ensures a clear direction and robust framework for implementing the system in subsequent phases.

The following report outlines the objectives, selected technologies, detailed system architecture, and project plan for Phase 1, laying the groundwork for a scalable and resilient distributed system.

Objective and Overview

Project's purpose

The purpose of this project is to design and implement a Distributed Web Crawling and Indexing System that utilizes cloud computing technologies. By leveraging the power of distributed systems and virtualized resources, the aim is to create a scalable and efficient solution for processing and organizing vast amounts of online information.

The problem that needs to be solved

The problem being addressed stems from the rapid growth of data available on the web, which presents challenges in accessing, indexing, and managing such large volumes of information. The system is intended to be both scalable and fault-tolerant, ensuring reliability even when parts of the system experience failure. This will allow for uninterrupted operation and an adaptable infrastructure that can handle increasing workloads.

The desired functionality

The system's functionality revolves around distributed web crawling, content processing, and indexing. Crawling involves fetching and analysing web pages across multiple virtual machines, while processing transforms the extracted content into meaningful data structures. Finally, the indexing module organizes this information into a searchable format, enabling users to efficiently query and retrieve relevant web pages.

Team Roles and Responsibilities

Roles

- **Architect**: Designs the system architecture, focusing on interactions and fault tolerance strategies, this role is assigned to **Ali Tarek**
- **Crawler Lead**: Implements the distributed web crawling functionality and ensures politeness measures, this role is assigned to **Tsneam Ahmed**

- **Indexer Lead**: Develops the indexing mechanism and search functionality, this role is assigned to **Mohamed Mostafa**
- **Cloud Infrastructure Lead**: Configures and manages cloud computing resources, this role is assigned to **Mohamed Mostafa**
- **Tester/Documentation Lead**: Tests system components and ensures thorough documentation, this role is assigned to **Mohamed Walid**

Requirement Refinement

Requirement Refinement is a crucial step to ensure the project's objectives, scope, and functionalities are clearly defined and well understood. This stage involves interpreting the project specifications, identifying key requirements, and refining them for implementation. Below are the refined requirements for the Distributed Web Crawling and Indexing System using Cloud Computing.

1. Distributed Crawling

The system must efficiently distribute crawling tasks across multiple virtual machines hosted in a cloud environment. This requirement ensures scalability, as the system can process large volumes of URLs simultaneously, with the ability to add more virtual machines as needed.

Key points include:

- Assigning crawling tasks dynamically to available crawler nodes.
- Implementing mechanisms for fault tolerance to reassign tasks if a crawler node fails.
- Respecting website policies through adherence to robots.txt directives.

2. Web Page Indexing

The system must implement an indexing mechanism to process crawled web pages and create a searchable index. This includes:

- Extracting meaningful content such as text, keywords, and metadata from crawled pages.
- Structuring the data in an organized, efficient manner for fast querying.
- Ensuring scalability and reliability by utilizing cloud-based storage solutions.

3. Fault Tolerance

The system must be resilient to failures, with robust mechanisms to maintain continuous operation despite node failures. Specific strategies include:

- **Crawler Node Failure**: Monitoring crawler nodes and reassigning tasks from failed nodes to operational ones.
- **Indexer Node Failure**: Implementing strategies to ensure index integrity and recovering from node failures without data loss.
- **Data Persistence**: Storing crawled data and indexes in durable cloud storage to avoid data loss.

4. Scalability

The system must be designed to scale effortlessly as the workload increases. This involves:

- Adding more crawlers or indexer nodes to the system.
- Ensuring efficient distribution of tasks and resources.
- Using cloud-based services for dynamic scalability.

5. Politeness and robots.txt Compliance

The crawler nodes must adhere to robots.txt directives specified by websites, ensuring ethical and polite crawling practices. This includes implementing crawl delays to avoid overloading servers.

6. Basic Search Functionality

The system must provide users with a simple search interface to query the indexed content. Features include:

- Keyword-based search with relevance ranking.
- Returning URLs and metadata of matching web pages.

7. User Stories

User stories help define the functional requirements from a user's perspective. include:

- "As a user, I want to initiate a web crawl by providing a list of seed URLs."
- "As a user, I want to specify crawling parameters such as depth limit or domain restrictions."
- "As a user, I want the system to continue crawling even if some crawler nodes fail."
- "As a user, I want to search for keywords and retrieve relevant web pages from the indexed content."
- "As a user, I want to monitor the progress of the web crawling and indexing process."

Technology Selection

List of Chosen Technologies

- **Programming language**: Python (rich libraries for web crawling, indexing, and distributed computing).
- Web crawling libraries: Beautiful Soup
- Indexing/search libraries: Elasticsearch
- Cloud computing platform: GCP
- Task queue service: GCP Pub/Sub
- **Persistent storage:** Google Cloud Storage

Justification

Programming Language: Python

Python is an excellent choice for this project due to its versatility and a rich ecosystem of libraries that simplify tasks like web crawling, indexing, and distributed computing. Its clean syntax and extensive documentation allow for faster development and easier debugging. Python's compatibility with numerous frameworks and cloud-based APIs makes it ideal for implementing distributed systems efficiently.

Web Crawling Library: Beautiful Soup

Beautiful Soup is a powerful library for parsing HTML and XML documents. It allows easy navigation and extraction of data from web pages. It's lightweight, making it suitable for handling the parsing needs of our system, such as extracting text and URLs from web pages, and works seamlessly with other Python libraries for additional functionalities.

Indexing/Search Library: Elasticsearch

Elasticsearch is highly scalable and provides fast full-text search capabilities, which makes it a perfect fit for indexing and searching large datasets. Its distributed nature aligns with our system's design, ensuring fault tolerance and high availability. Elasticsearch also supports advanced querying, such as relevance ranking and aggregations, which enhances the search functionality of your system.

Cloud Computing Platform: Google Cloud Platform (GCP)

GCP provides a wide range of cloud services and is well-suited for distributed applications. It offers cost-effective virtual machines, robust networking capabilities, and scalable storage solutions, making it an optimal choice for deploying your system. GCP's integration with Pub/Sub for message queuing and its compatibility with Terraform further streamline the development process.

Task Queue Service: GCP Pub/Sub

GCP Pub/Sub: A cloud-based messaging service designed for scalability and reliability. It ensures efficient task distribution and provides strong fault tolerance guarantees. Using Pub/Sub for inter-component communication simplifies the orchestration of crawling and indexing tasks.

Persistent Storage: Google Cloud Storage

Google Cloud Storage offers a scalable and durable solution for storing crawled data and indexed content. Its high availability and multi-region storage options ensure data persistence and reliability. It integrates seamlessly with other GCP services, streamlining the data pipeline from crawling to storage and indexing.

System Architecture Design

1. High-Level Architecture Diagrams

Purpose: Provide a clear overview of the entire system and its major components.

Content:

- Include components: Client Layer (UI), Master Node, Crawler Nodes, Indexer Nodes, Task Queue (Pub/Sub), and Persistent Storage (GCS).
- Show data flow:
    - User submits crawl jobs to Master Node via UI.
    - Master Node assigns tasks to Crawler Nodes via Pub/Sub.
    - Crawler Nodes send processed content to Pub/Sub for Indexer Nodes.
    - Indexer Nodes store indexed data and serve queries from UI.

  We Used arrows to indicate communication and data exchange.

2. Component Interaction

Purpose: Illustrate how system components interact with each other during the crawling and indexing process.

Content:

- Depict interactions:
    - UI submits crawl jobs to Master Node.
    - Master Node publishes tasks to Pub/Sub, which Crawler Nodes subscribe to.
    - Crawler Nodes fetch URLs, process data, and publish results to Pub/Sub for indexing.

- Indexer Nodes subscribe to processed data, build indexes, and store them in GCS.
- UI queries indexed data via Indexer Nodes.
- Include key APIs and communication protocols.

3. Data Flow

Purpose: Show the flow of data between components throughout the system.

Content:

- Seed URLs and parameters flow from UI to Master Node.
- Master Node breaks tasks and publishes to Pub/Sub for Crawler Nodes.
- Crawler Nodes process data and publish it back to Pub/Sub for Indexer Nodes.
- Indexer Nodes build indexes and store them in GCS.
- Indexed data serves search queries from the UI.

Highlight different types of data (raw HTML, processed text, indexed content).

4. Fault Tolerance Mechanism

Purpose: Highlight strategies for handling component failures.

Content:

- Show:
  - Task reassignment from failed Crawler Nodes by the Master Node.
  - Data replication and recovery for Indexer Nodes.
  - Reliable messaging in Pub/Sub.
  - Durable storage in GCS for persistence.
  - Illustrate heartbeat monitoring and task timeout mechanisms.

5. Deployment

Purpose: Visualize the system's deployment in the cloud.

Content:

- Include cloud resources:
  - VM instances for Master Node, Crawler Nodes, and Indexer Nodes.
  - GCP Pub/Sub for messaging.
  - GCS for storage.
- Specify configuration details (e.g., VM types, network setup).

6. Sequence for a Crawl Job

Purpose: Detail the steps involved in initiating and executing a crawl job.

Content:

- Show:
  - UI sending job request to Master Node.

- Master Node assigning tasks to Crawler Nodes.
- Crawler Nodes processing URLs and sending results to Indexer Nodes.
- Indexer Nodes building the index and storing it in GCS.
- UI querying indexed data from Indexer Nodes.

More Details on System Architecture Design

1. Client Layer (User Interface)

- **Purpose:** Allows users to initiate web crawling jobs, set parameters, view progress, and search indexed data.
- Features:
    - Input seed URLs, crawl depth limit, and domain restrictions.
    - Submit queries to search indexed content.
    - Monitor progress of crawling and indexing.
    - Technology:
    - Web application built with Flask for backend and HTML/TailwindCSS for frontend.
    - Publishes crawl jobs to the UI-Master topic via GCP Pub/Sub, which the master node subscribes to.

2. Master Node (Controller/Scheduler)

- **Purpose:** Manages task distribution and system orchestration.
- Responsibilities:
- Divide seed URLs into tasks and assign them to Crawler Nodes.
- Monitor worker nodes and reassign tasks if failures occur.
- Publishcrawl tasks to the **CRAWL_TASKS_TOPIC_ID** and subscribe to the **NEW_CRAWL_JOB_SUBSCRIPTION_ID**.
- Technology:
- Python with Google Cloud Pub/Sub APIs for task distribution.
- Reads seed URLs from Google Cloud Storage and handles dynamic job submissions.

3. Crawler Nodes

- Purpose: Perform web crawling tasks.
- Responsibilities:
- Fetch web pages using **requests.**
- Parse HTML with Beautiful Soup and extract text content and new URLs.
- Save raw HTML and processed text to Google Cloud Storage.
- Publish new URLs to UI-Master topic and processed content to Crawler-Indexer topic for indexing.
- Technology:
- Python with libraries like Beautiful Soup and requests.
- Utilize GCP Pub/Sub for task communication and GCS for storage.

4. Indexer Nodes

- **Purpose:** Build and manage the searchable index.
- Responsibilities:
    - Subscribe to Pub/Sub for processed content messages from Crawler Nodes.

        - Download processed text from Google Cloud Storage.
        - Create a robust index using Elasticsearch for search functionality.
        - Handle queries from the Client Layer.
- Technology:
        - Python and Elasticsearch for indexing and search.
        - Persistent index storage in Google Cloud Storage.

5. Task Queue (Google Cloud Pub/Sub)

- **Purpose:** Facilitates communication between system components.
- Responsibilities:
        - Master Node publishes crawl tasks to **Master-Crawler topic.**
        - Crawler Nodes publish processed data to **Crawler-Indexer topic.**
        - Reliable delivery ensures fault tolerance.

6. Persistent Storage (Google Cloud Storage)

- **Purpose:** Durable and scalable storage for data.
- Responsibilities:
        - Store raw HTML, extracted text, and indexed data.
        - Ensure persistence for system fault tolerance and recovery.

Data Flow Overview

1. Job Submission:
    - Client sends crawl jobs with parameters via Pub/Sub.
    - Master Node processes and assigns tasks to Crawler Nodes.
-
2. Crawling:
    - Crawler Nodes fetch pages, extract data, and store results in GCS.
    - Publish new URLs and processed content to Pub/Sub for further processing.
3. Indexing:
    - Indexer Nodes ingest processed content from GCS and build the index.
4. Querying:
- Client queries indexed data via the Indexer Nodes.

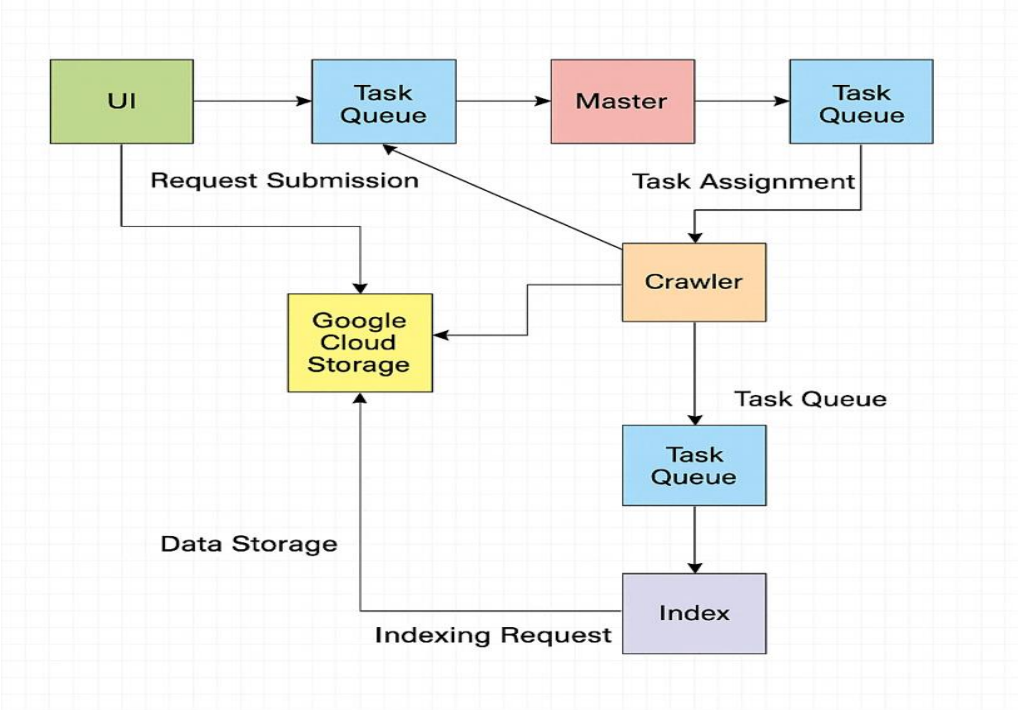

Figure 1 System Architecture Design

Project Planning

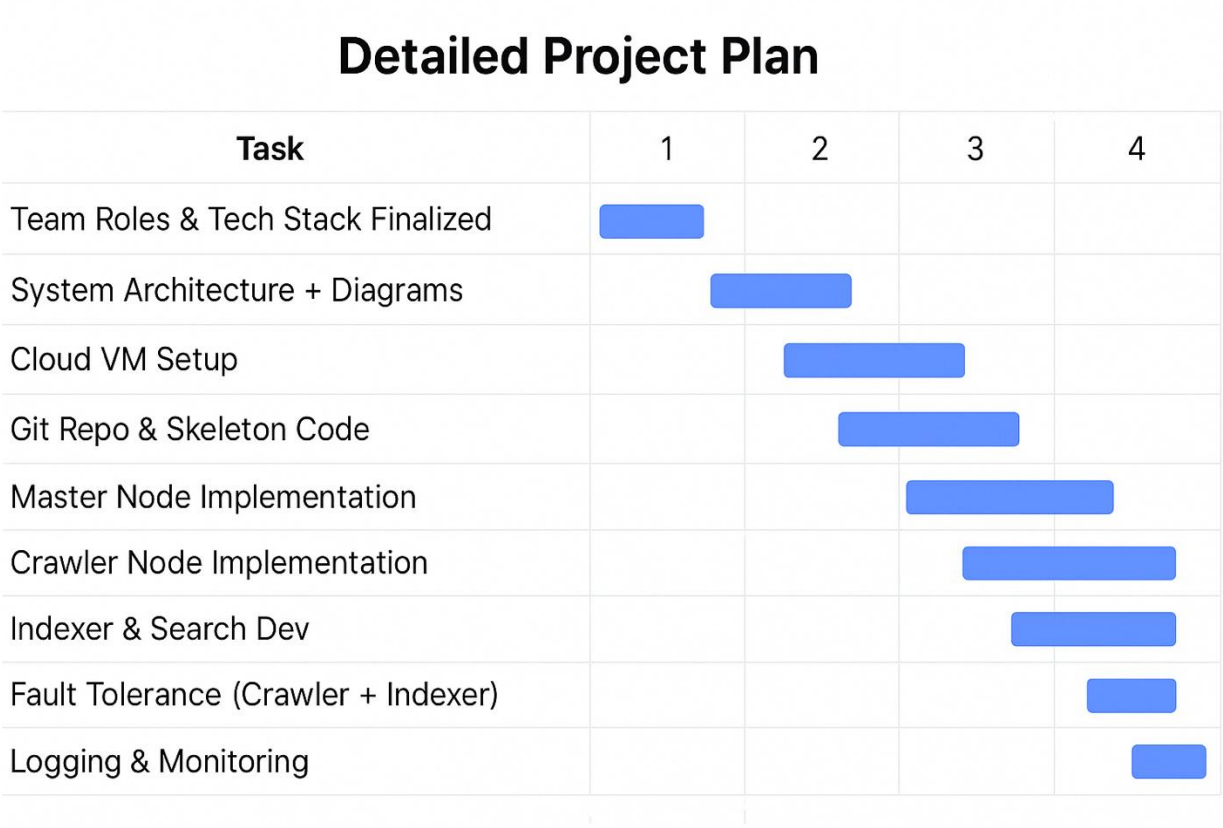

Figure 2 Project Planning

Cloud Environment Setup

Set up accounts with the chosen cloud provider.

Google Cloud Platform (GCP) under the project: `Distributed Project`.
Project ID: `sunlit-pixel-456910-j0`
Project Number: `663221054063`

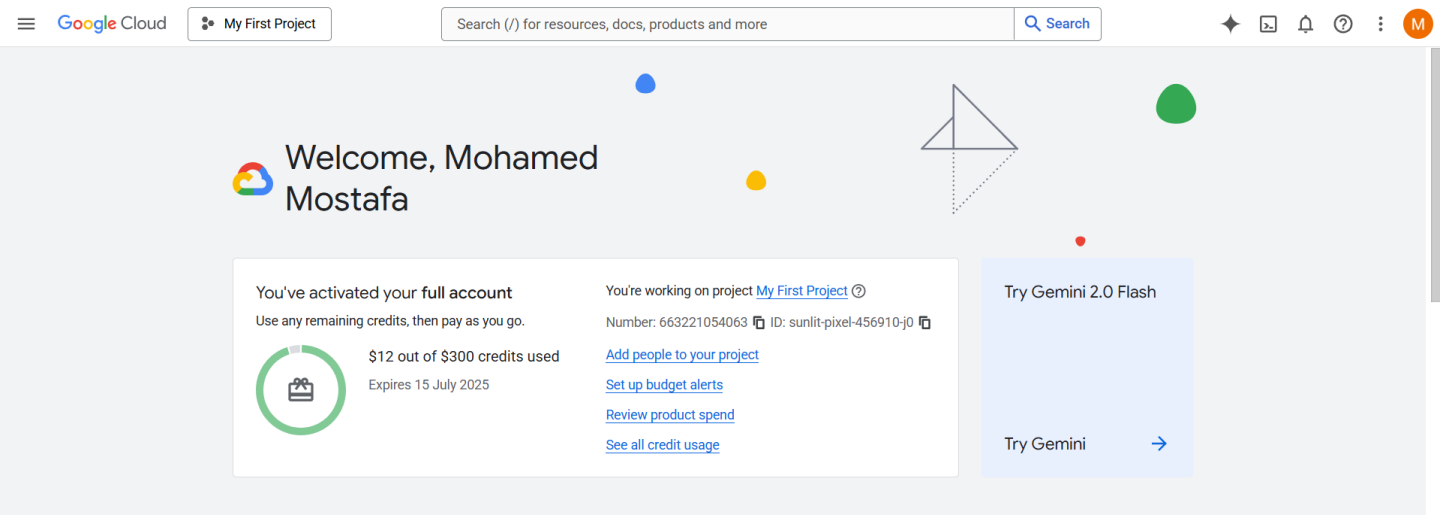

Figure 3 Cloud Environment Setup

Configure basic cloud resources (initial VMs for Master, Crawler, Indexer for testing, Task Queue service, Storage service).

1. Service Account for Instances

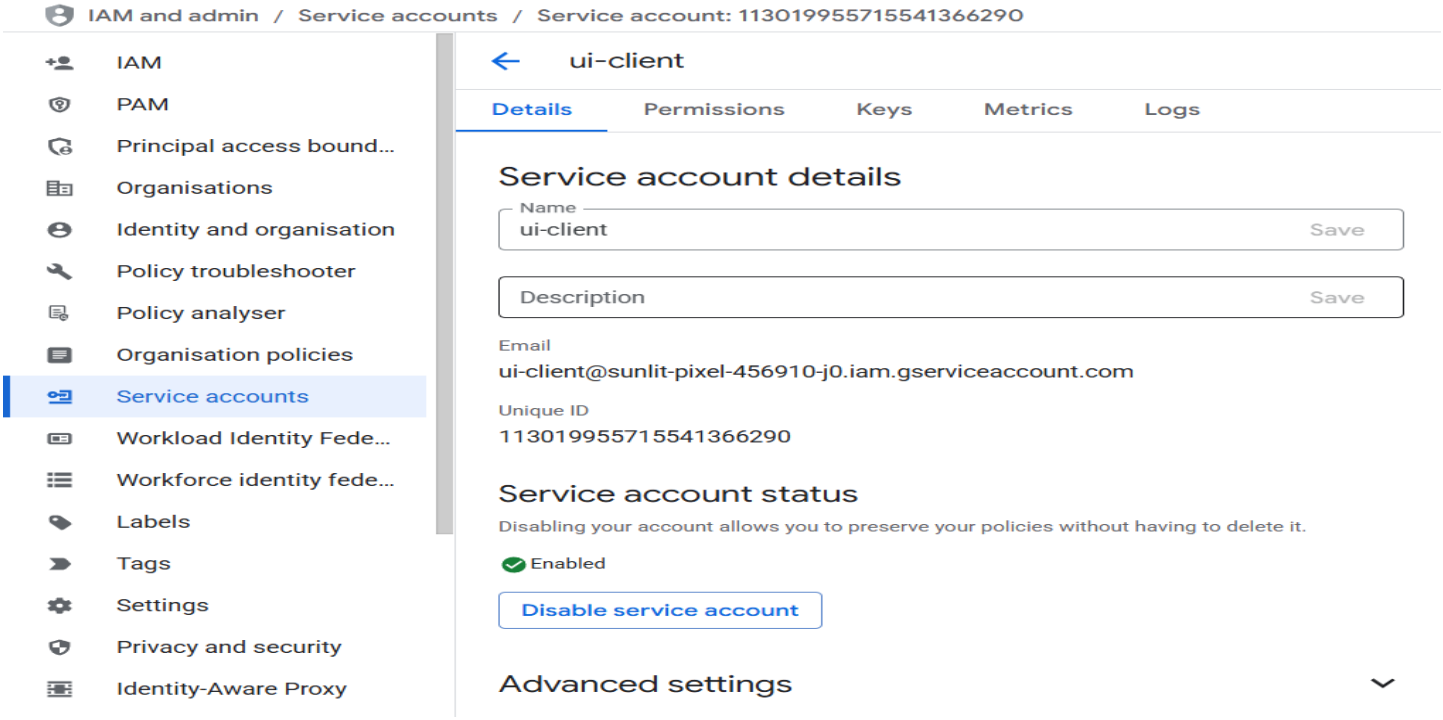

Figure 4 Service account details

2. Master VM

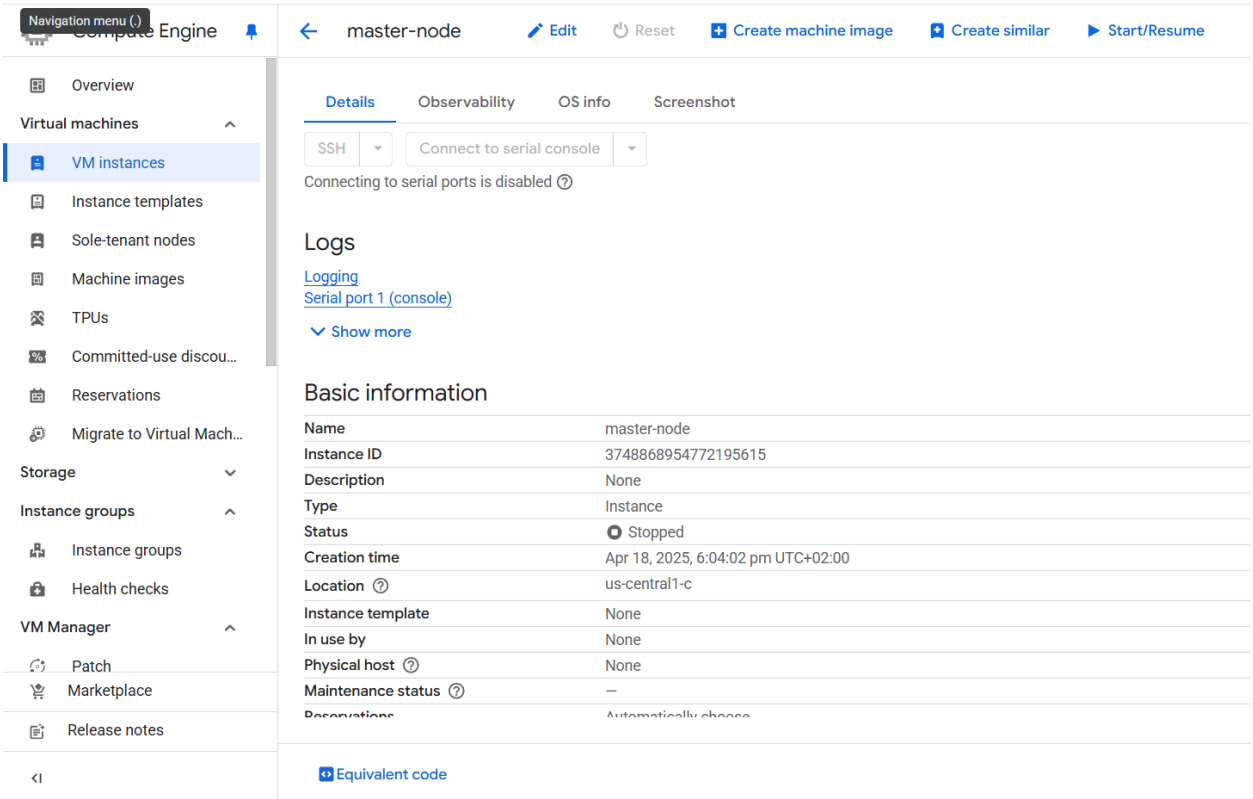

Figure 5 Master virtual machine

3. Crawler VMs

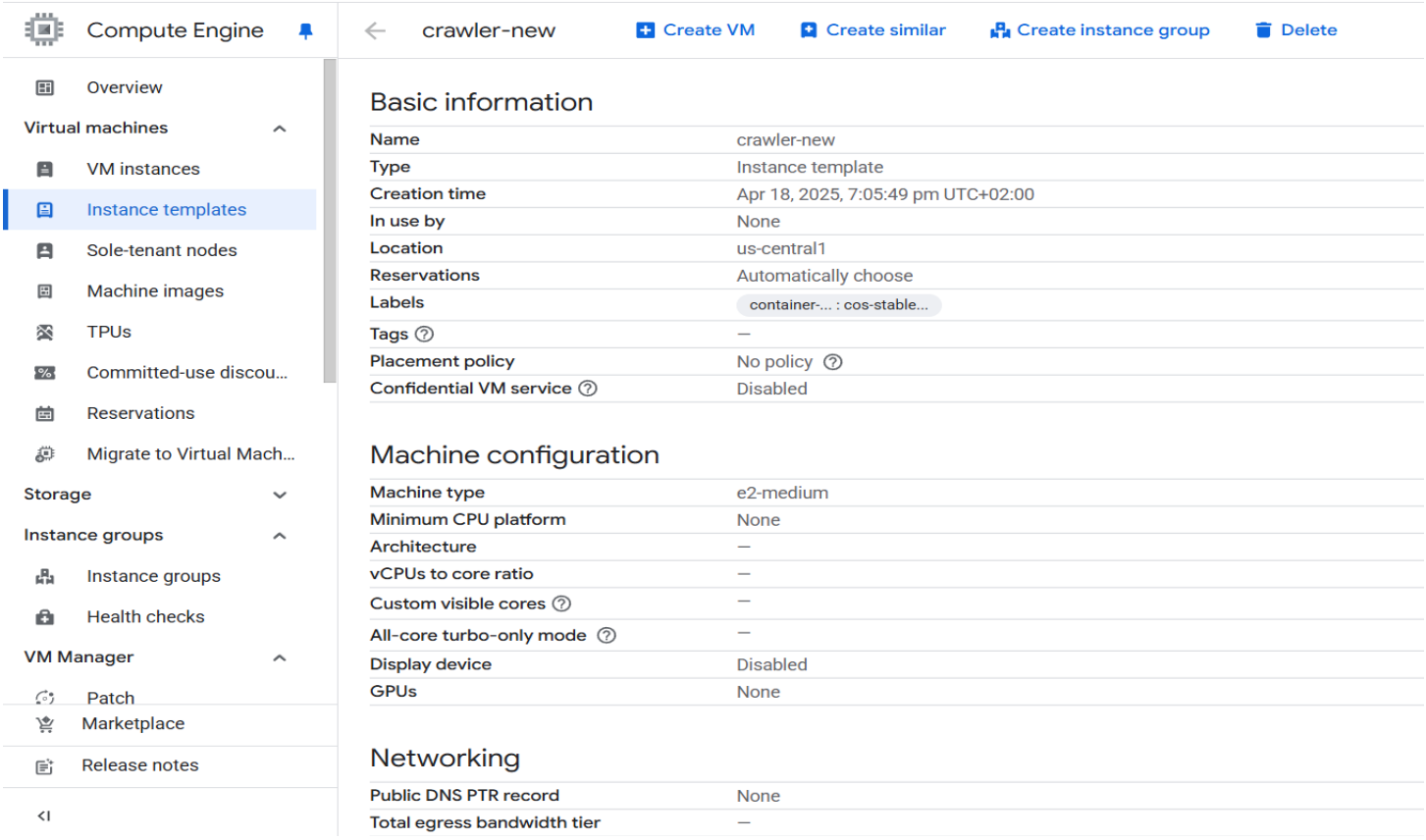

Figure 6 Crawler virtual machine

4. Indexer VMs

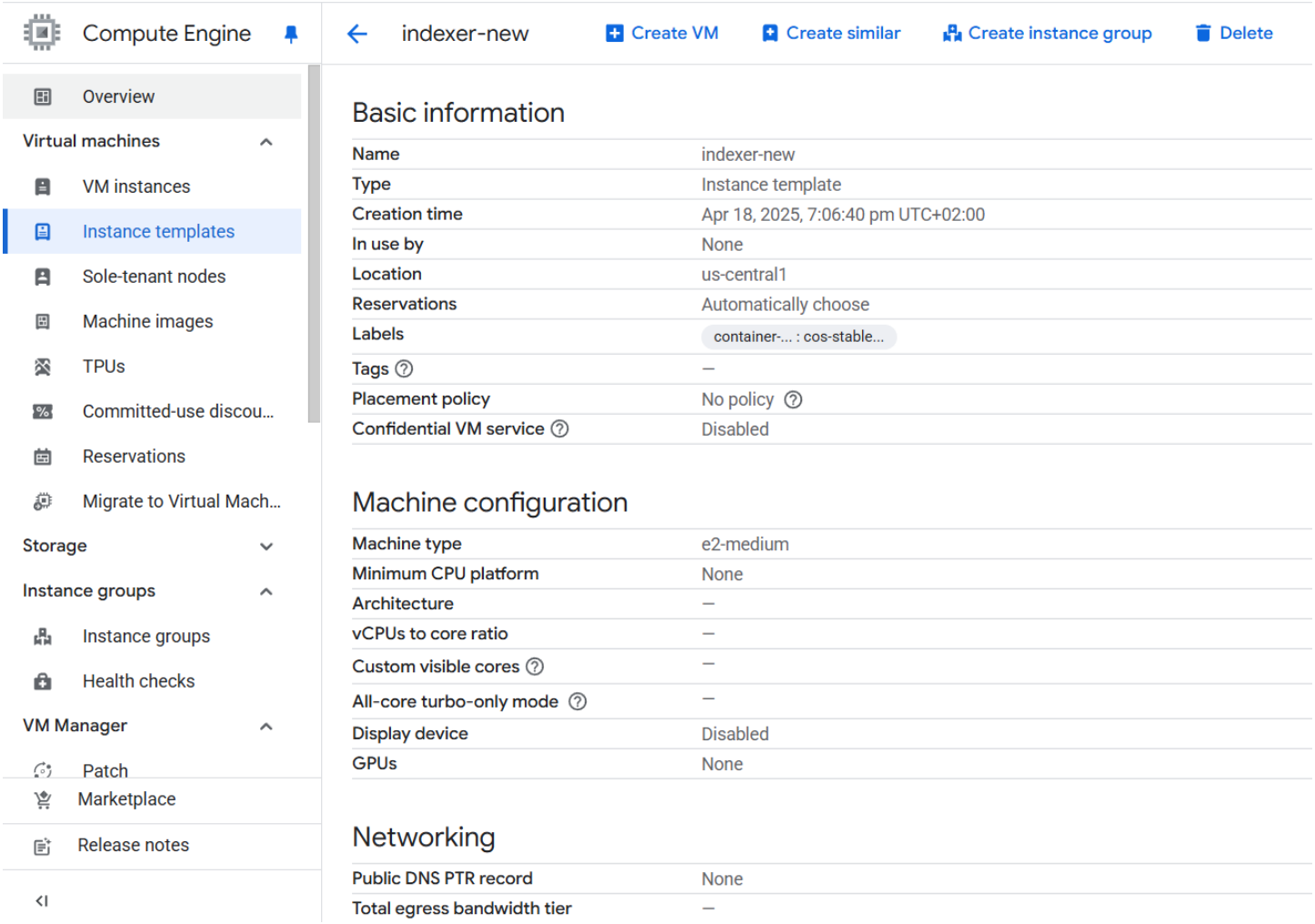

Figure 7 Indexer virtual machine

Initiate a New Web Crawl

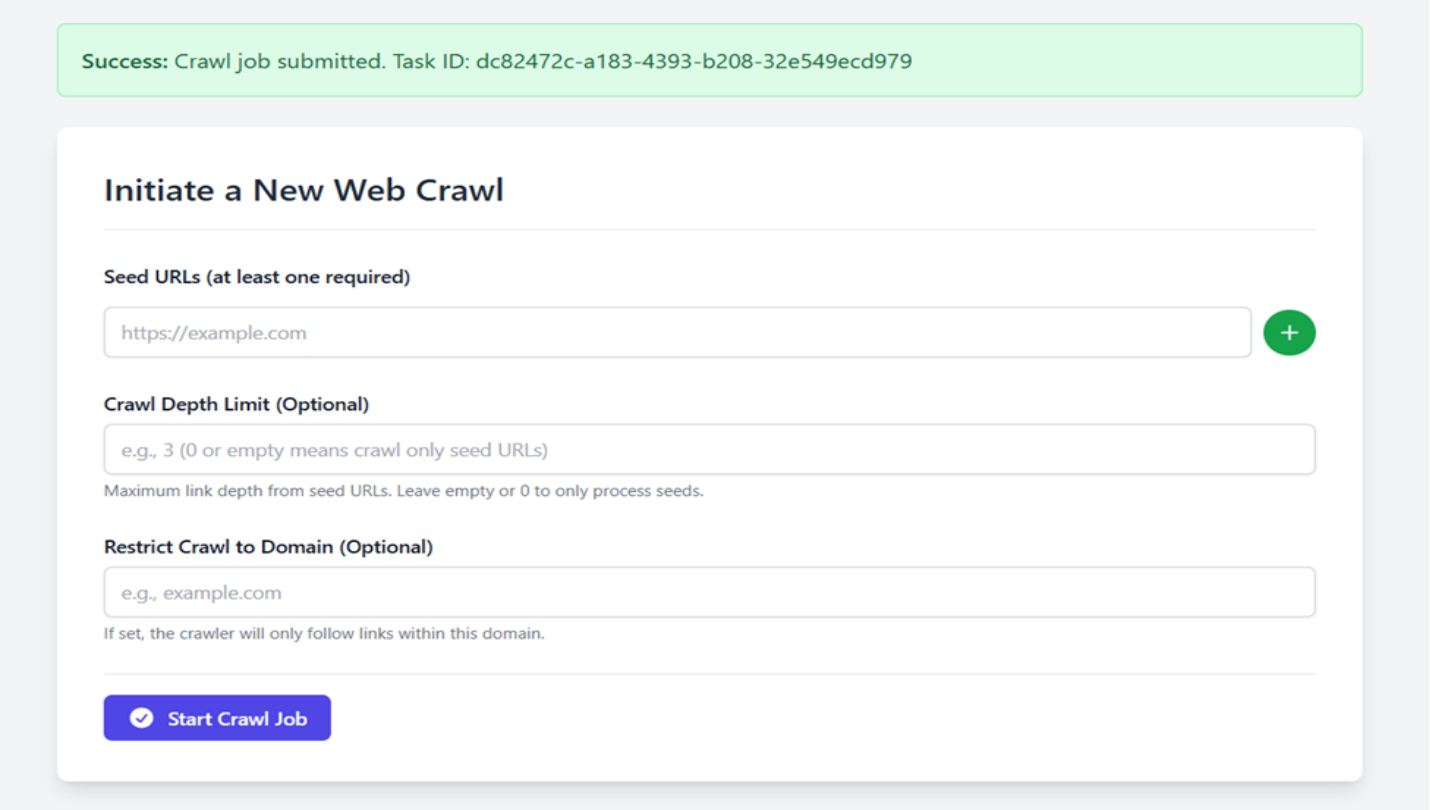

Figure 8 Example of initiate a new web crawl

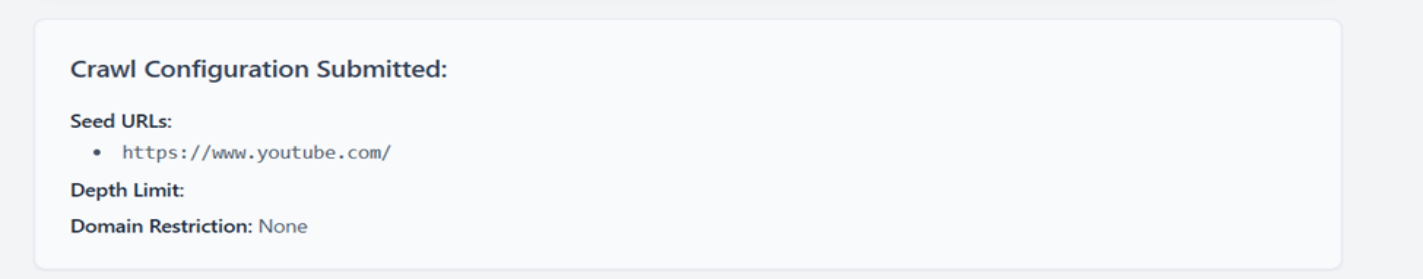

Figure 9 Result of crawl configuration submitted

Pub/Sub

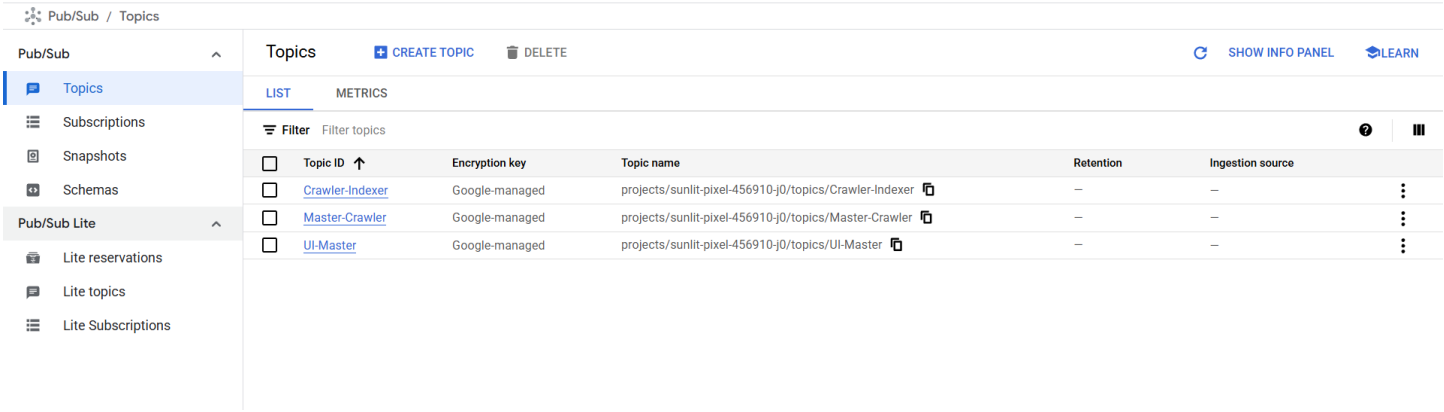

Figure 10 Pub/Sub Virtual machines

ElasticSearch

Figure 11 ElasticSearch

VM instance templates

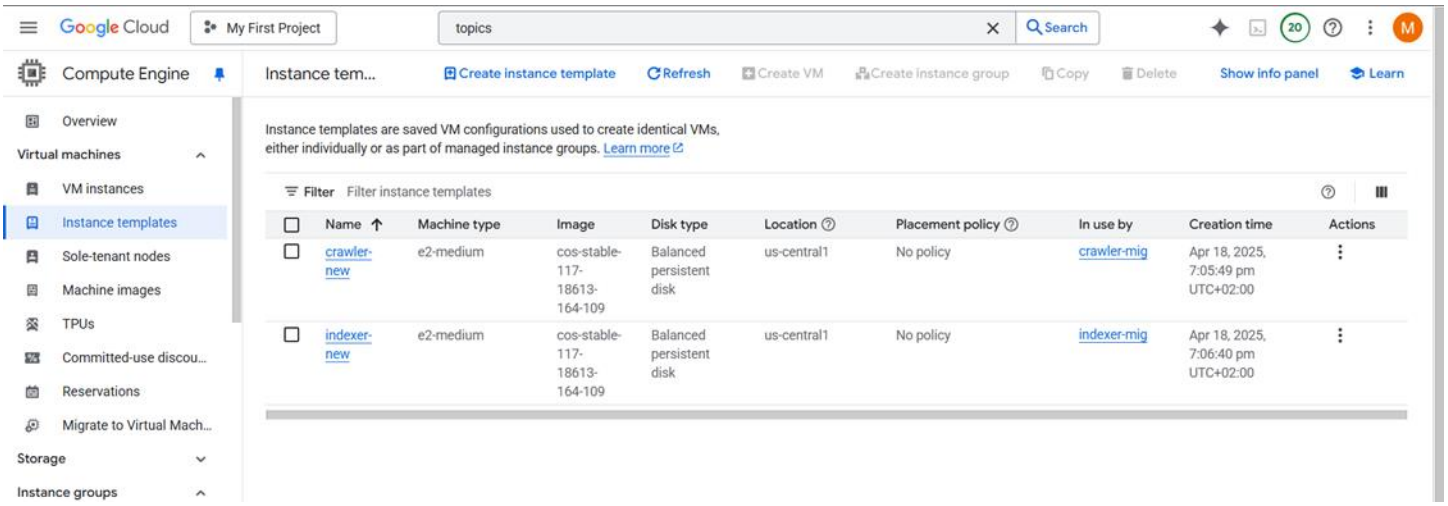

Figure 12 VM instance templates

VM instance groups

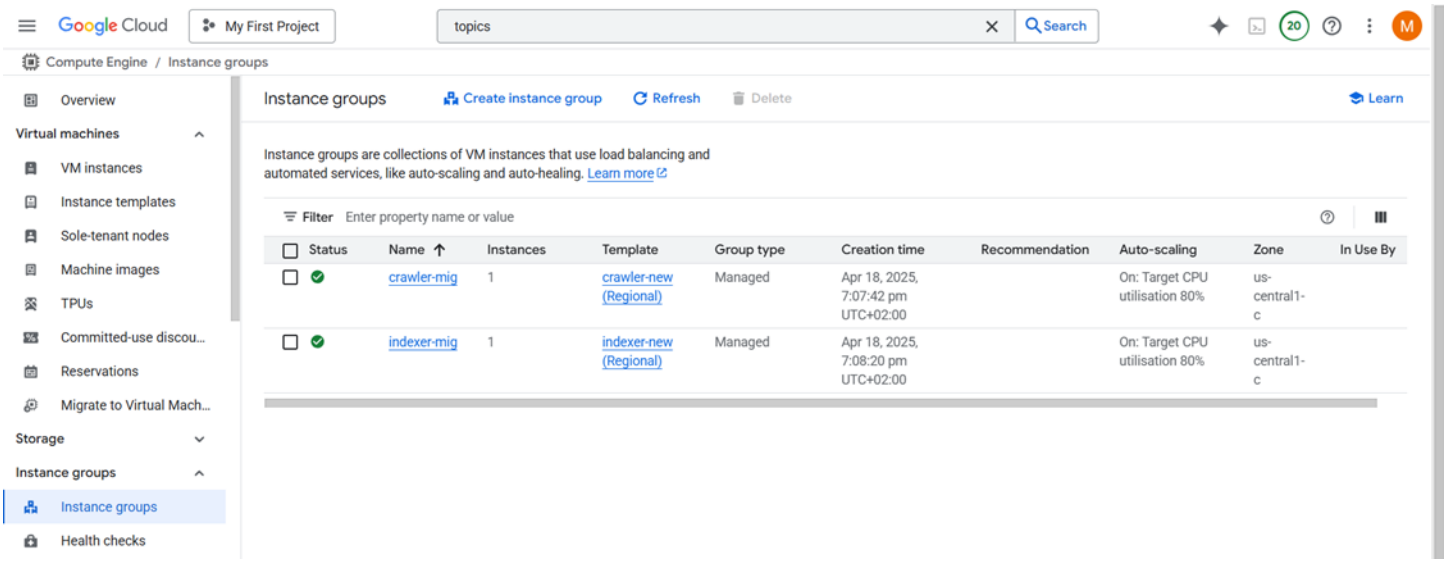

Figure 13 VM instance groups

Artifact registry repository and images

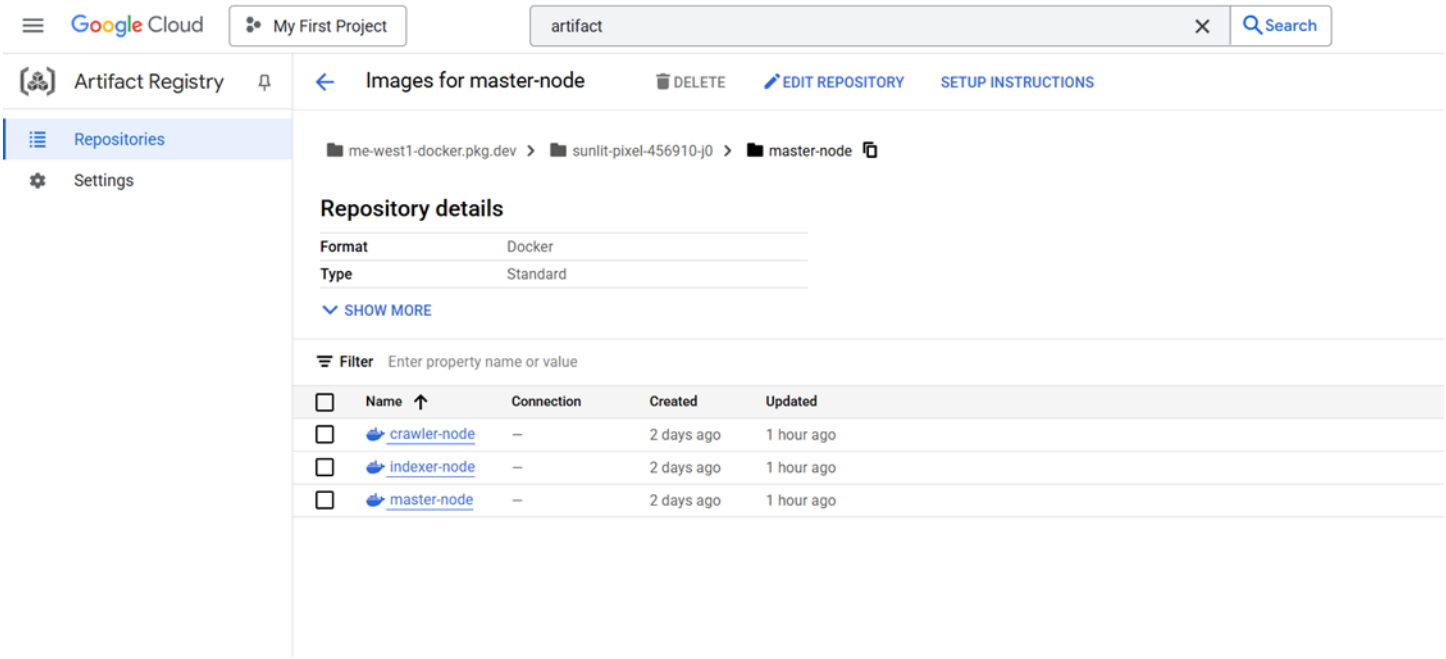

Figure 14 Artifact registry repository

Cloud Storage

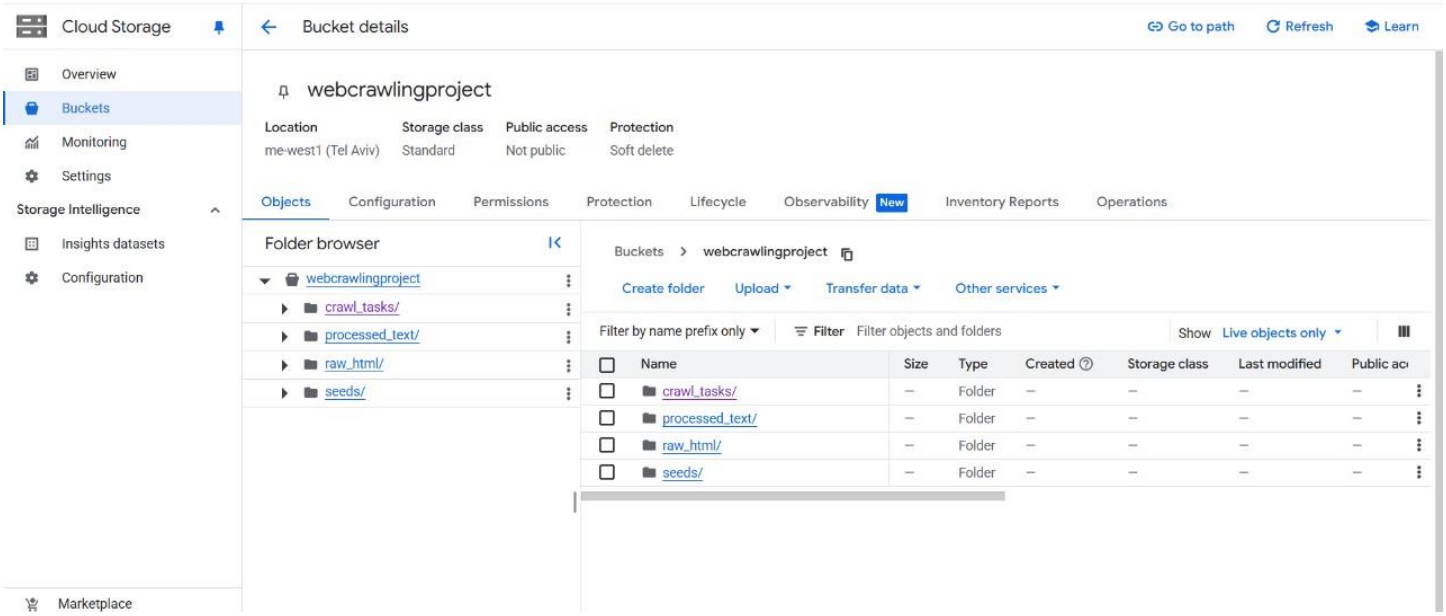

Figure 15 Google Cloud Storage

Initial Code Repository and Skeleton Code

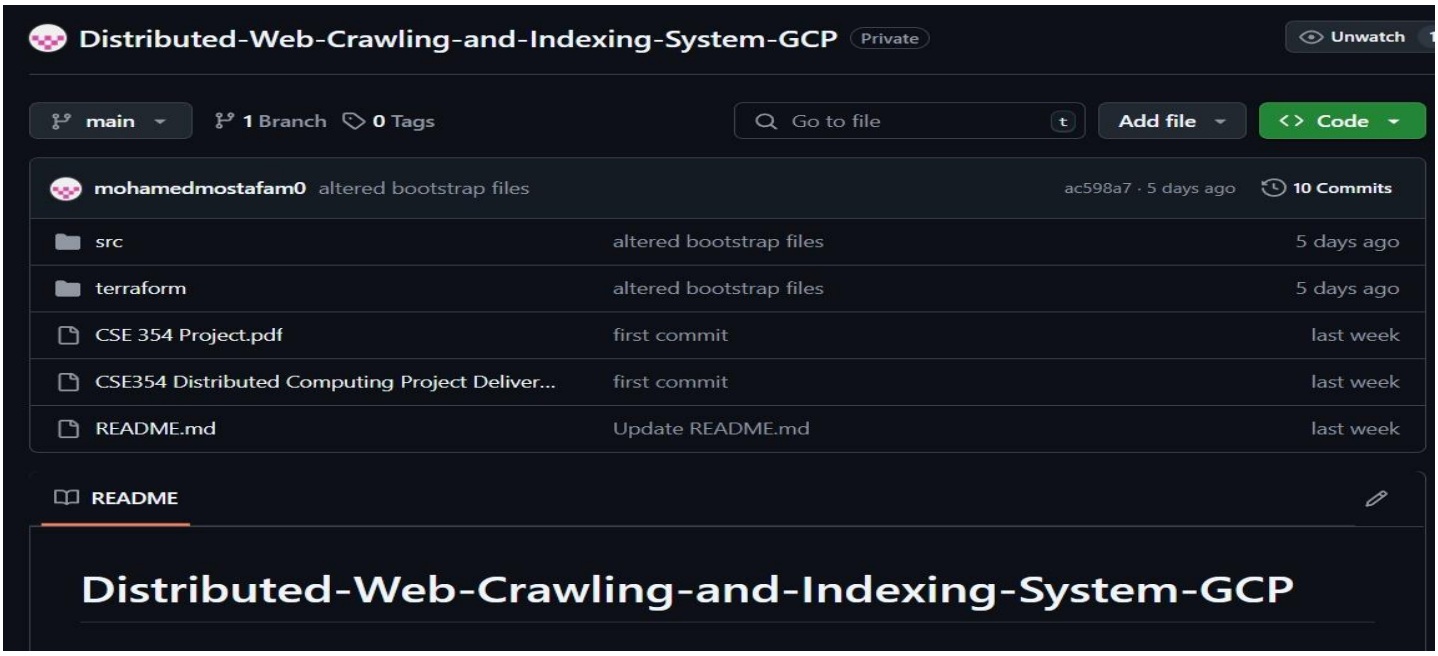

Figure 16 Code Repository Github

```
PROJECT/
├── README.md
├── CSE 354 Project.pdf
├── CSE354 Distributed Computing Project Deliverables.pdf
│
├── Documentation/
│
├── src/
│
├── scripts/
│   ├── .env
│   ├── crawler_node.py
│   ├── indexer_node.py
│   ├── master_node.py
│   └── requirements.txt
│
├── UI/
│   ├── main.py
│   └── templates/
│   └── index.html
│
├── terraform/
│   ├── Scripts/
│   │   ├── crawler_bootstrap.sh
│   │   ├── indexer_bootstrap.sh
│   │   └── master_bootstrap.sh
│   │
│   ├── compute.tf
│   ├── main.tf
│   ├── monitoring.tf
│   ├── network.tf
│   ├── outputs.tf
│   ├── provider.tf
│   ├── PubSub.tf
│   ├── storage.tf
│   ├── terraform.tfvars
│   └── variables.tf
```

Conclusion

Conclusion In Phase 1 of this project, significant progress has been made in defining the objectives, requirements, and foundational elements essential for successful implementation. We have outlined the problem to be solved, the project's purpose, and the desired functionalities, ensuring clarity in our goals. Detailed roles and responsibilities have been assigned to the team to ensure efficient collaboration. By selecting a robust technology stack, including tools like Python, Beautiful Soup, Elasticsearch, and Google Cloud Platform, the project is positioned for high performance, scalability, and fault tolerance. Furthermore, the system architecture has been meticulously designed to support distributed crawling, indexing, and deployment. These efforts lay a strong foundation for the subsequent phases, ensuring that we remain on track to achieve the project's objectives effectively and innovatively

References

https://cloud-based_web_crawler_architecture_cloud_lab_ucm.pdf

https://cloud.google.com/architecture/framework/reliability/build-highly-available-systems

https://www.robotstxt.org/

https://www.crummy.com/software/BeautifulSoup/

https://cloud.google.com/?hl=en

https://developer.hashicorp.com/terraform

https://cloud.google.com/pubsub?hl=en

https://cloud.google.com/storage?hl=en

https://cloud.google.com/appengine?hl=en