# Job Analytics App – Docker Deployment Guide

## Overview

This project is a Job Analytics App composed of multiple services orchestrated using Docker Compose. It includes task queuing, data extraction, storage, and monitoring components.

## Services Breakdown

### Redis

Image: redis:latest  
Purpose: In-memory data store used by Celery for task queuing. Usually used for cache memory storage for fast requests.  
Ports: 6379:6379  
Volumes: redis\_data:/data

### Redis Commander

Image: rediscommander/redis-commander  
Purpose: Web UI for managing Redis. It allows users to monitor the Redis service using a web UI on their web browser.  
Ports: 8081:8081  
Environment:  
 - REDIS\_HOSTS=local:redis:6379

### Flower

Image: Same as Celery  
Purpose: Web-based monitoring tool for Celery. Allows for the monitoring of celery workers, active and resolved tasks with the corresponding results.  
Ports: 5555:5555  
Command: celery -A celery\_app.tasks flower --port=5555 --loglevel=info  
Depends on: redis, celery

### Celery Worker

Image: Built from local Dockerfile  
Purpose: Executes background tasks from the queue.   
Command: celery -A celery\_app.tasks worker --loglevel=info -E  
Depends on: redis  
Volumes:  
 - ./celery\_app  
 - ./data\_extraction/Websites  
 - output  
 - logs  
Environment: Loaded from .docker.env

### Data Extraction

Image: Same as Celery  
Purpose: Runs the web scraping script by queuing tasks in celery. The script ends when the four websites are scraped and the result data is stored as Json files in MinIO storage.  
Command: python /app/data\_extraction/web\_scrape.py  
Depends on: redis, celery  
Volumes: Same as Celery

### MinIO

Image: minio/minio  
Purpose: Object storage (S3-compatible). In our project it is used to store all the data that is web scraped.  
Ports: 9000, 9080  
Command: minio server --console-address :9080 /data  
Volumes: minio\_data:/data  
Environment: From .docker.env

### PostgreSQL

Image: postgres:latest  
Purpose: Relational database for storing tables.  
Ports: 5432:5432  
Volumes:  
 - postgres\_data:/var/lib/postgresql/data  
 - ./docker-entrypoint-initdb.d  
Environment: From .docker.env  
Healthcheck: Uses pg\_isready

## Prerequisites

1. Docker & Docker Compose installed or Docker desktop on windows machines.
2. .docker.env file inside the root folder with:

* MINIO\_ROOT\_USER: The username for MinIO server
* MINIO\_ROOT\_PASSWORD: The password for MinIO server
* MINIO\_API: The address to the MinIO API. By default, it is "minio:9000" if using the docker-compose file provided.
* MINIO\_CONSOLE: The address to the MinIO console which is a Web UI. By default, it is "minio:9090" if using the docker-compose file provided. Trying to access the MinIO API from web browser will automatically forward to this address.
* TMP\_DIR: The folder for storing the temporary files of the various project services, these include chrome profiles and chrome driver instances. By default, it is "/app/data\_extraction/tmp" if using the docker-compose file provided.
* LOG\_DIR: The folder for storing the logging files of the various project services, these include web scraper logs and data treatment logs. By default, it is "/app/data\_extraction/log" if using the docker-compose file provided.
* PYTHONPATH: The path to the root folder to be used when python is running. This allows for absolute imports to work correctly when using the scripts in the containers. By default, it is "/app" if using the docker-compose file provided.
* CHROME\_BIN: The path to the chrome browser binary downloaded during the build of the docker image. By default, it is "/opt/chrome/chrome" if using the docker-compose file provided.
* CHROME\_DRIVER\_DIR: The path to the chrome driver binary downloaded during the build of the docker image. By default, it is ”/home/celery\_user/.local/share/undetected\_chromedriver"
* POSTGRES\_USER: The username for the Postgres database server. By default, is “root”.
* POSTGRES\_PASSWORD: The password for the Postgres database server. By default, it is “123456”
* POSTGRES\_DB: The name of the database inside the Postgres server. By default, it is “offers”
* DB\_HOST: The hostname of the Postgres host machine. By default, it is “postgres”
* DB\_PORT: The port of the Postgres host machine. By default, it is “5430”

## Instructions to Launch Docker Containers

1. Ensure Docker and Docker Compose are installed on your machine or Docker Desktop if on windows (will require WSL).  
2. Create a .docker.env file with the necessary environment variables.  
3. Navigate to the project directory.  
4. Run the following command to start the containers:  
 docker-compose -f dockercompose.dev.yaml up --build  
5. Verify that all services are running correctly.

## How the project works

Once the project image files have been built and run, all running containers should show up if using the Docker Desktop UI or by using the “docker ps” command in command shell.

The first step is setting up the databases and the celery worker. The Redis, Postgres and MinIO servers should all be up and running and exposing their ports for use. The celery worker container should activate one worker with events on for further monitoring using Flower.

You can check the logs of each container by checking the logs windows inside Docker Desktop or using the command “docker logs <container id or name>”.

You can also continuously monitor the logs by adding the “follow” flag “docker logs -f <id or name>”.

As soon as this first step ends, the web scraping should start automatically (only on first startup of project). This will queue four tasks, one for each website, and start the chrome drivers. You can monitor the logs directly from the celery worker container or wait until the end of the scraping and check the log files at “app/data\_extraction/log” or at the MinIO web scraping bucket.

If you need to queue additional tasks, please refer to the celery\_app/tasks.py file. It has all the currently defined tasks and allows celery to queue them. To run said tasks run the command “docker exec -it <container id or name> (your command)”.

For example, to run the rekrute task, navigate to the celery\_app folder using “cd celery\_app”, then “python -i tasks.py”. This will open python’s interactive mode and await your input. Now simply enter your task like this “rekrute\_task.delay()” or “emploi\_task.delay()” and it will put in the queue. To see the results either go to the Flower web UI or simply check the logs of the celery container as shown before.

## Monitoring

When running the container, the Flower UI can be accessed at the address “flower:5555”. This will allow to see all the current active workers and tasks with additional information like queues , results of tasks and concurrency of workers.

The results of tasks are stored in Redis and can be viewed at “redis:6379”.