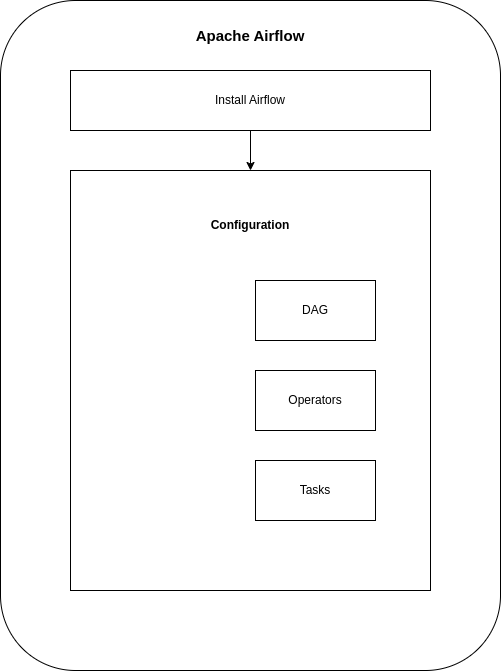
1. Apache Airflow - <https://airflow.apache.org/>

Airflow helps us with scheduling, and monitoring jobs.. Like crontab but with extra features like parallel processing, running tasks over multiple servers and a UI to visualize the DAG setup and see the task status.



* Installation process can be easy with the docker
* Configuration process is divided further into:
  + DAG
    - A DAG (Directed Acyclic Graph) is the core concept of Airflow. It's a collection of tasks arranged with dependencies and relationships to define the order in which they should run.
  + Operators
    - Operators are the building blocks of DAGs. They represent a single task. Each operator defines a specific action to be taken, such as running a Bash script, sending an email, or running a Python function.
  + Tasks
    - Tasks are instances of operators within a DAG. They are the actual process when the DAG is executed.

**Advantages:**

* **Visualization**: Airflow provides a web interface for monitoring and managing workflows, making it easy to track the progress and status of tasks, complex dependencies between tasks can be viewed as a graph in the UI.
* **Scalability and Extensibility:** Airflow supports custom operators which allowing you to extend its functionality to meet your specific needs
* **Retry Mechanism:** It has built-in support for task scheduling and retries, which helps in handling failures and ensuring data consistency

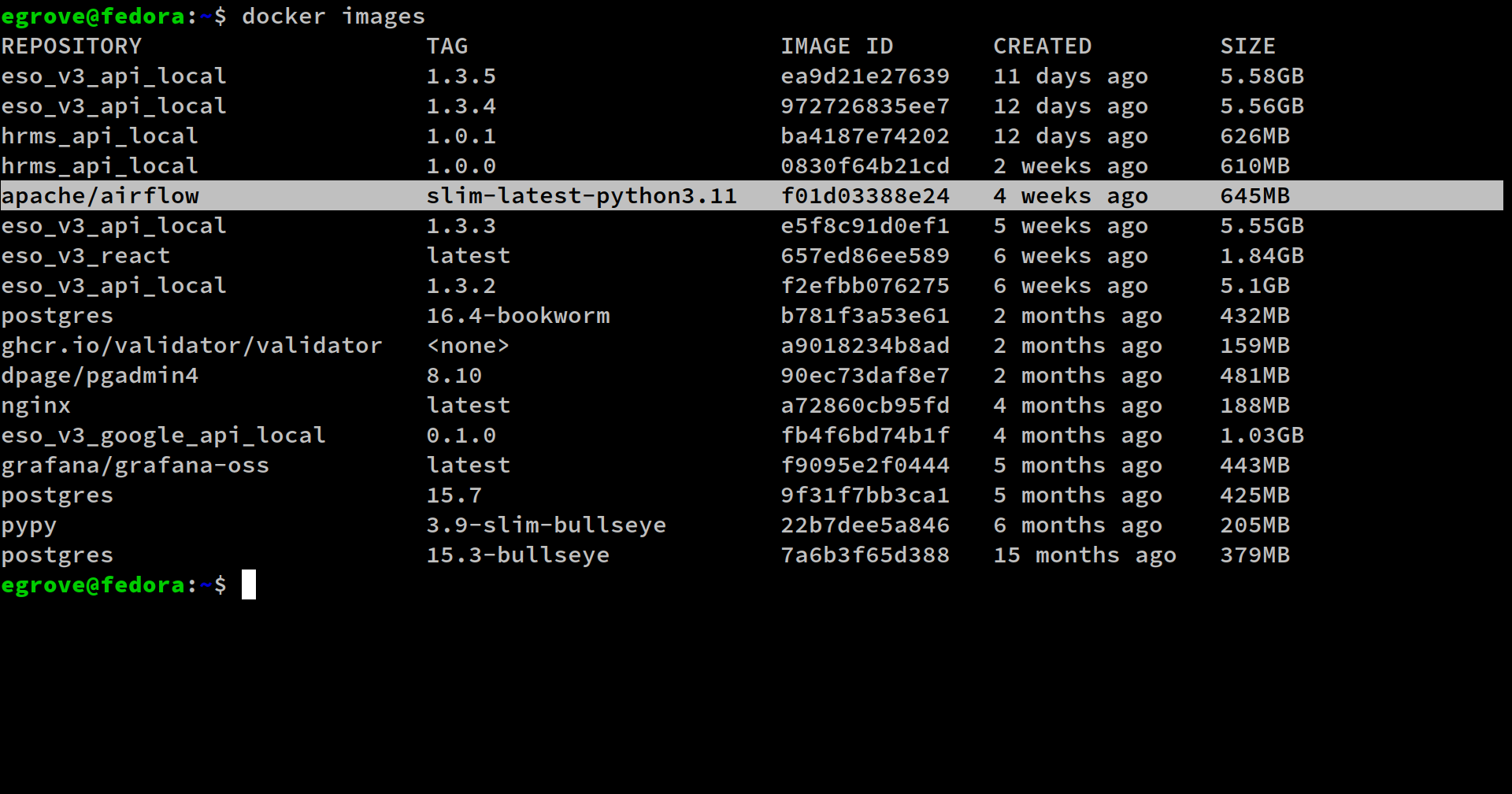
**Disadvantages**

:

* **Learning Curve:** Setting up and configuring Airflow can be complex. There is a learning curve associated with understanding and effectively using its workflow management tools.
* **Resource Requirement:** Running Airflow, especially with large workflows, can be resource-intensive and may require significant computational power.

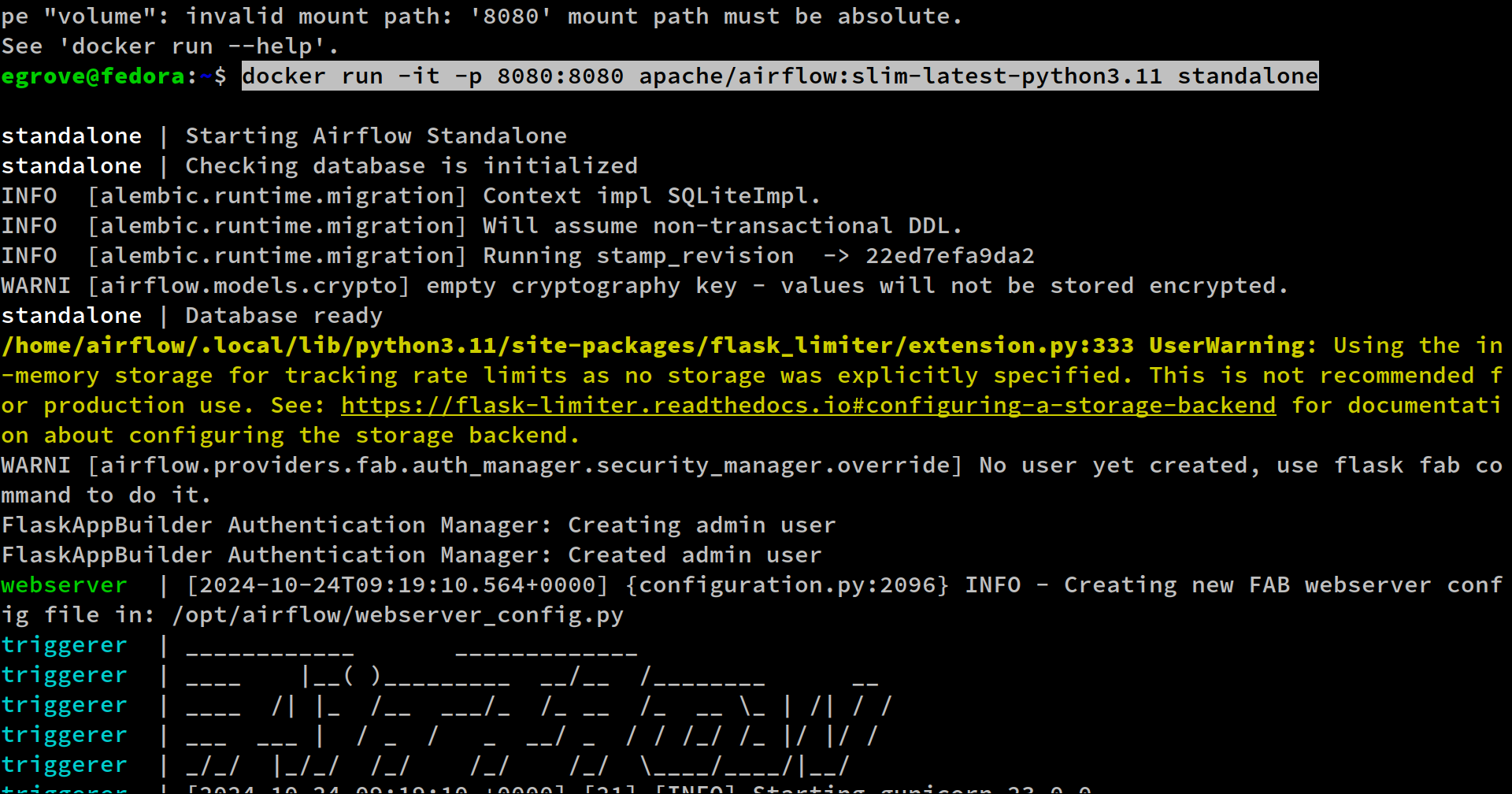
I installed a local copy of airflow using docker:

docker pull apache/airflow:slim-latest-python3.11

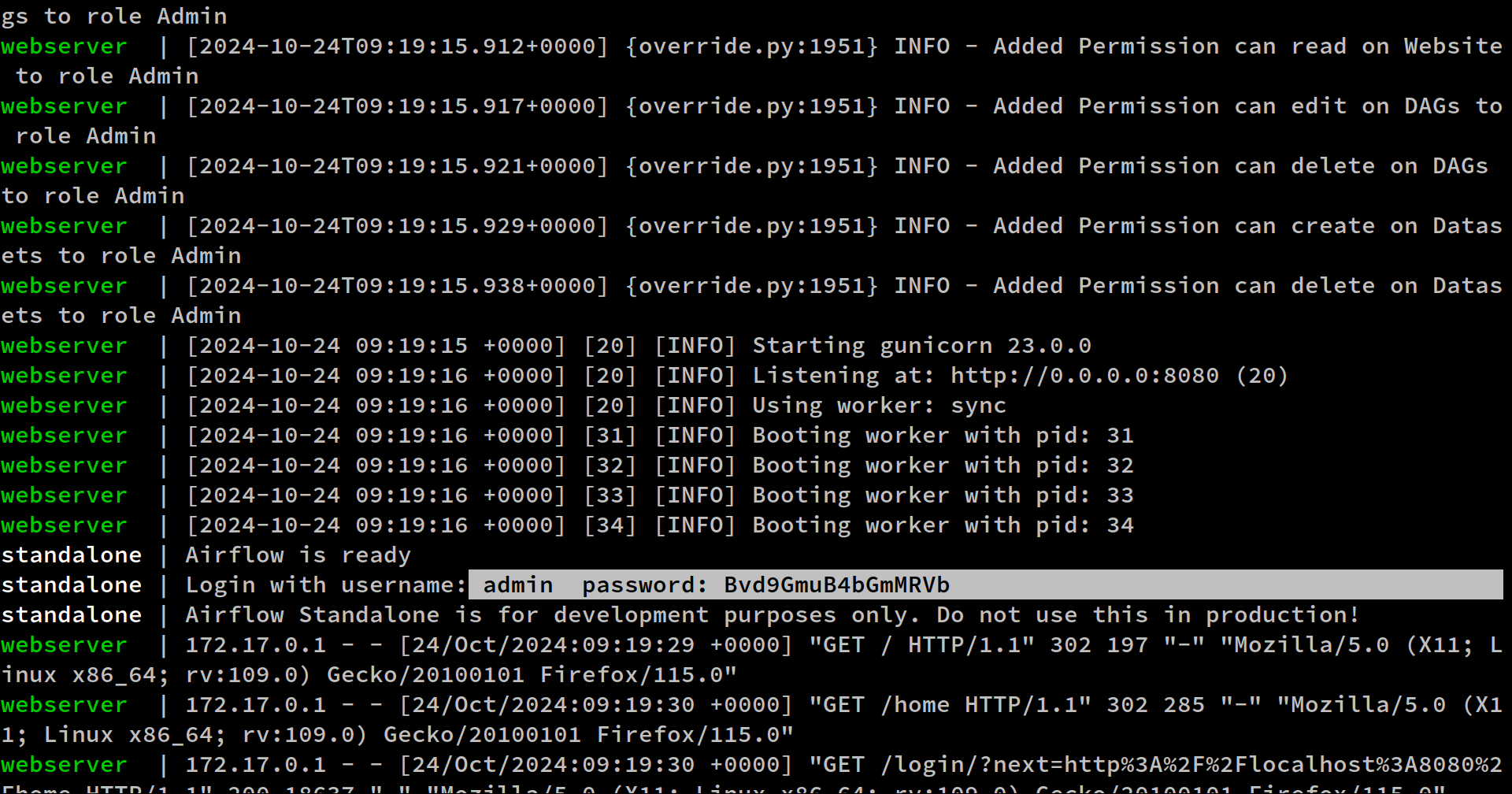


Running a standalone instance for trying out the airflow:

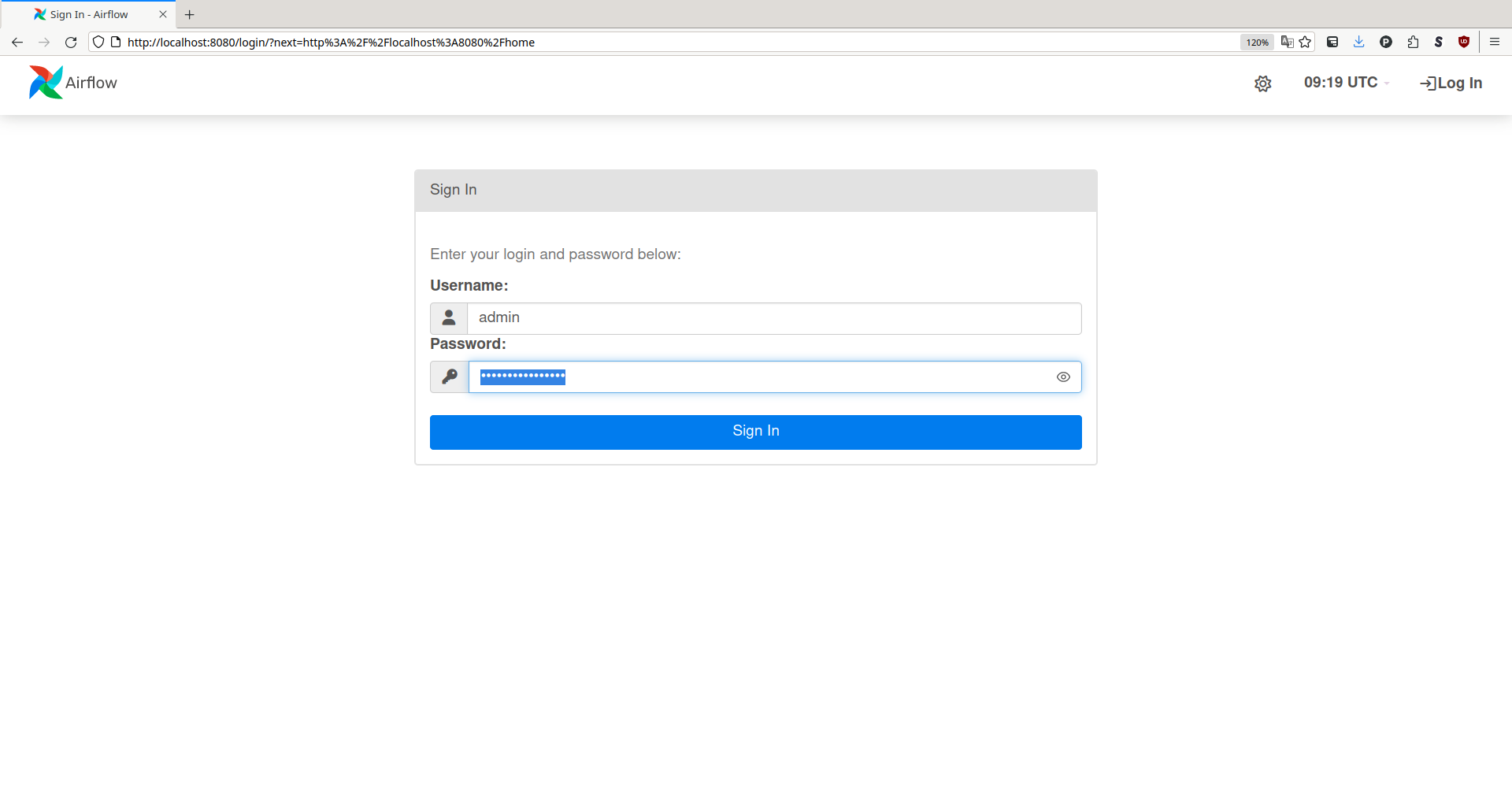
docker run -it -p 8080:8080 apache/airflow:slim-latest-python3.11 standalone

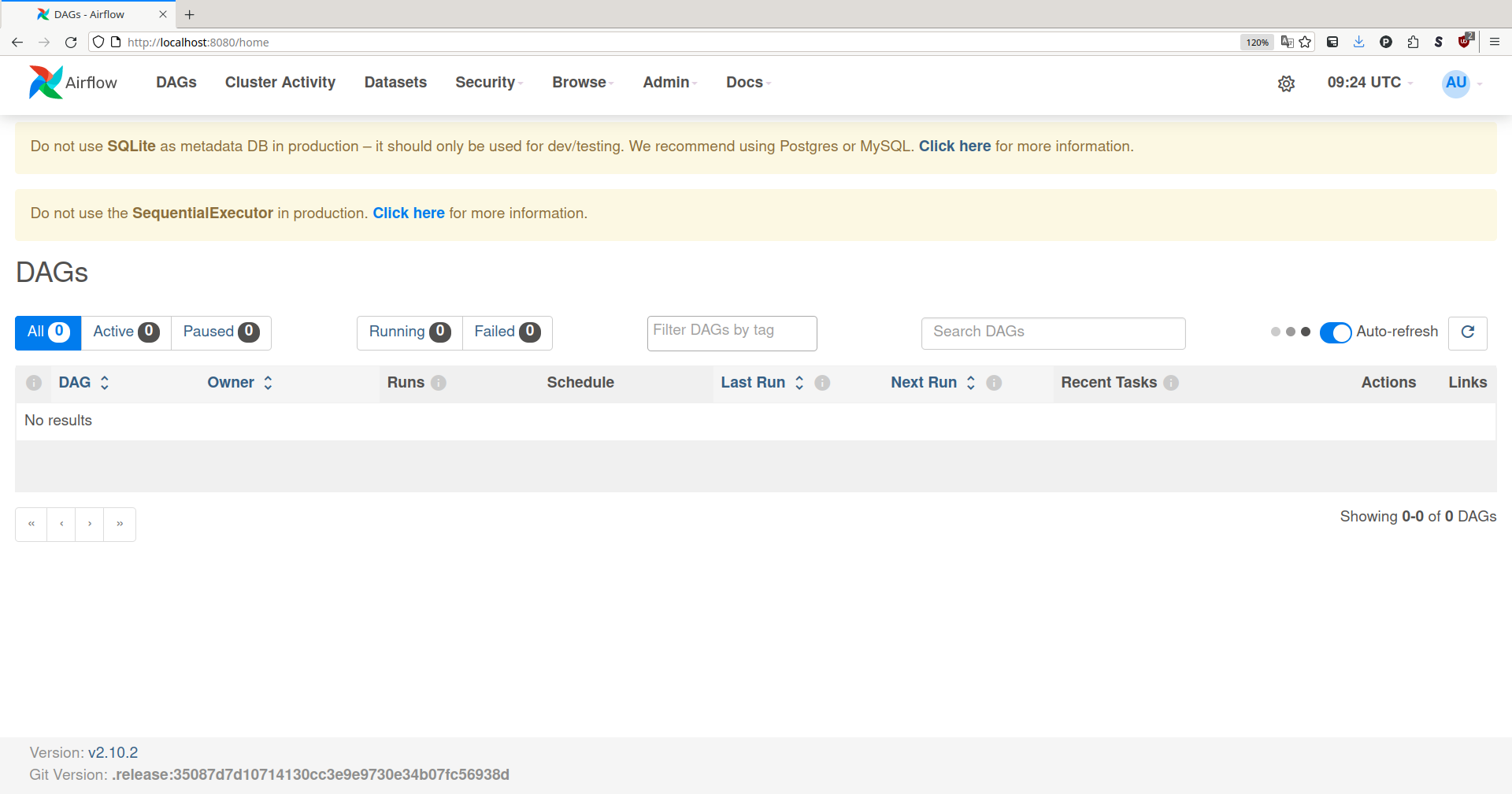


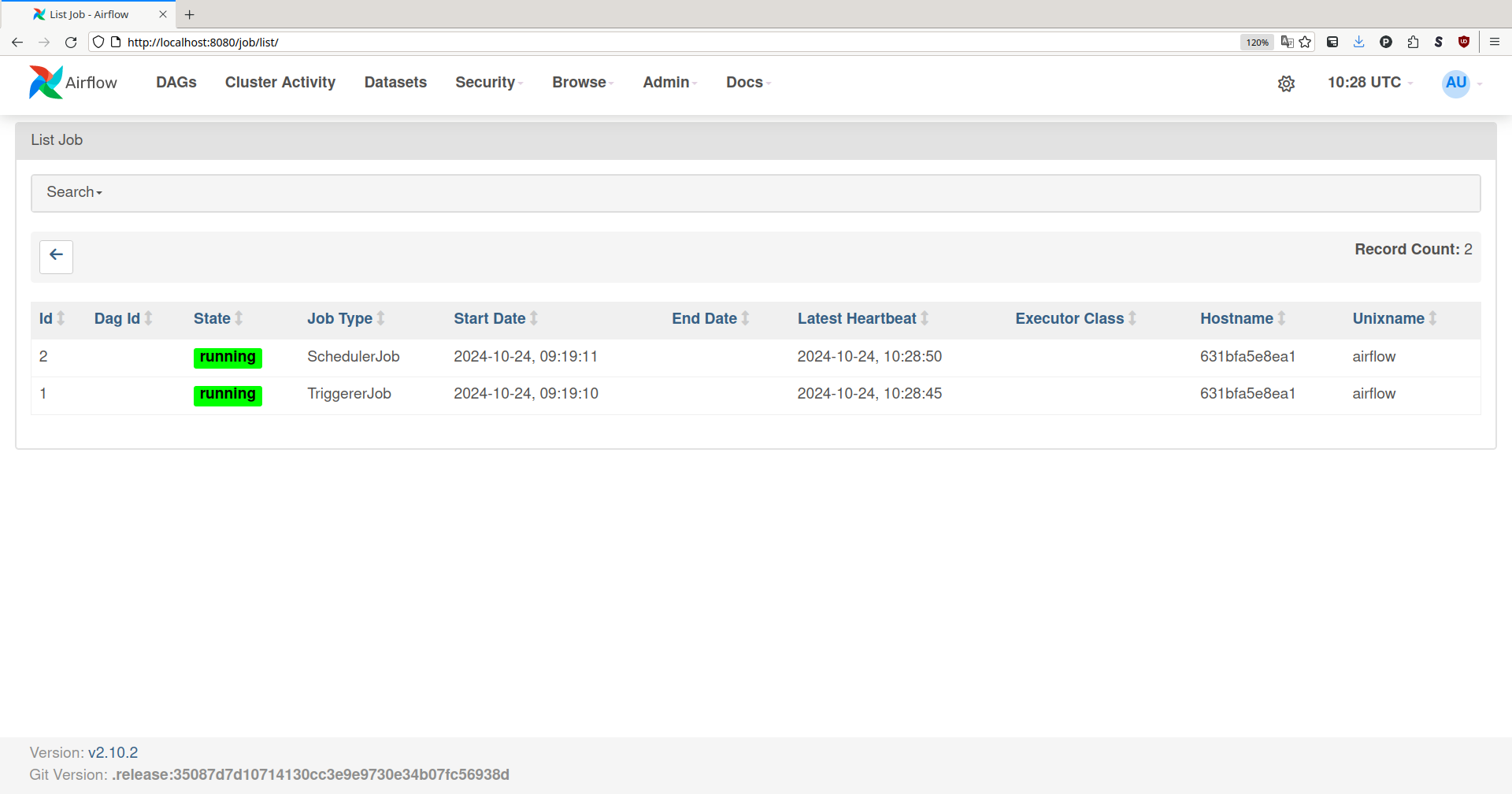
It automatically started on port 8080 and gave me the admin credentials.



I was able to visit the localhost:8080 to view the airflow UI and login using admin credentials.

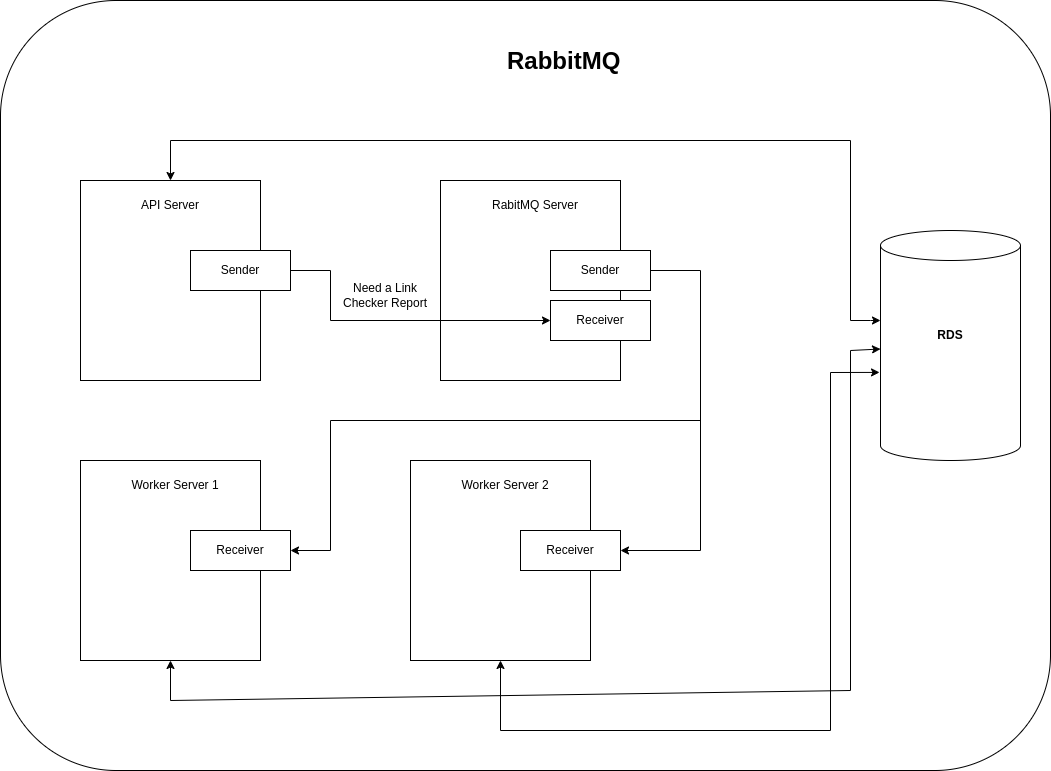






2. RabbitMQ - <https://www.rabbitmq.com/>

RabbitMQ is an open-source message broker software. It uses a protocol called AMQP (Advanced Message Queuing Protocol) to ensure reliability. RabbitMQ acts as a communication bridge between different parts of a system.



**Advantages:**

* Allows different parts of a system to operate independently
* Ensures messages are delivered even if the recipient is temporarily unavailable
* Supports multiple messaging protocols like AMQP, MQTT, and STOMP
* Can be set up for high availability and failover

**Disadvantages:**

* Setting up and managing RabbitMQ can be complex
* Need more workers to set up a distributed system.

3. Celery - <https://docs.celeryq.dev/en/main/getting-started/introduction.html>

Celery is a task queues tool that helps us to distribute work across threads or machines.

**Usages:**

* Asynchronous Tasks: Running tasks without making users wait, like sending emails or processing files.
* Scheduled Jobs: Executing tasks at certain times, like daily data backups or regular data scraping.
* Handling Long-Running Tasks: Tasks that take a long time to complete, like generating large reports or complex data processing.

**My Opinion on using Celery:**

* If we are planning to use celery in a single machine / server, it is essentially the same implementation as our current environment and does not provide much help.
* But if we are to use it to scale our worker processes on multiple servers, it can be useful.

**Tools Comparison:**

| **Feature** | **Apache Airflow** | **RabbitMQ** | **Celery** |
| --- | --- | --- | --- |
| Installation Type | Using PIP or docker | Package manager or docker | Python package |
| User Interface | Available. \* view the DAG dependencies as graph \* view the jobs in a table. | Available. \* View the channels and message passing counts | N/A |
| Logging | Have separate logging. Save as file in server. | Have separate logging. Save as file in server. | Integrate with Django logger. |
|  |  |  |  |