

ASSIGNMENT – LAB 1.1 : Spark Environment

Setup & First Application

Learning Outcomes

By completing this assignment, you should be able to:

- Deploy a functional Spark environment using Docker (cluster mode).
 - Verify container state and inspect the Spark Master UI.
 - Install and configure a local PySpark environment.
 - Execute a basic Spark application using the SparkSession API.
 - Explore and interpret the Spark Application UI (jobs, stages, DAGs, executors).
-

Context

This lab simulates the initial setup of a real Spark development workflow:

- A **remote/production-like Spark cluster** running in Docker.
 - A **local PySpark setup** for development and testing.
 - A first end-to-end Spark application that uses DataFrames, transformations, and actions.
 - A **deep dive into Spark UI** to understand how Spark schedules and executes jobs.
-

Assignment Tasks

Part A – Deploy a Spark Cluster with Docker

1. Create a new working directory:

`data-engineering-course/`

2. Write a correct **docker-compose.yml** that defines:

- a Spark master

- a Spark worker
(you must infer the correct image names, environment variables, ports from course examples)

3. Start the cluster:

```
docker-compose up -d spark-master spark-worker
```

4. Verify running containers using:

```
docker-compose ps
```

5. Access the Spark Master Web UI at:

```
http://localhost:8082
```

6. Capture a screenshot of the Spark Master dashboard showing:

- master alive
 - worker connected
 - cores & memory recognized
-

Part B – Install and Validate Local PySpark

1. Create and activate a Python virtual environment.

2. Install the required packages:

- pyspark
- pandas

3. Write and run a small test script to confirm:

- SparkSession starts
- Version is printed

4. Stop the session properly.

Part C – Write & Run Your First Spark Application

1. Create a file: `lab1_hello_spark.py`

2. Implement the Spark application according to the instructions:

- initialize SparkSession
- display environment info

- create a DataFrame from static data
- show schema, preview, aggregations, filters

3. Execute the script from the terminal.

4. Save the complete console output for submission.

Part D – Explore the Spark Application UI

Run the script again, then open:

- `http://localhost:4040`

Explore the following sections and take screenshots:

- Jobs tab (overview)
- DAG visualization for one job
- Task details of one stage
- Executors tab

Everything is to be interpreted based on the Spark architecture learned in class.

Hints

- **Spark Master UI** only shows the cluster state, not your running jobs – those appear on port 4040.
- Each `.show()`, `.count()`, `.filter()` triggers a **new Spark job** → expect several jobs.
- A job is split into **stages**; a shuffle or wide transformation causes stage boundaries.
- Executors tab will show:
 - the driver
 - the executors
 - even in local mode.
- If Spark UI 4040 is empty, it means your application finished – rerun your script.
- If Docker containers are not visible, ensure:
 - Docker is running
 - Correct services are referenced in `docker-compose up`

Common Pitfalls to Avoid

- Forgetting to map ports correctly in `docker-compose.yml`.
 - Confusing **Spark Master UI (cluster UI)** with **Application UI (per job)**.
 - Using wrong Python version inside virtual environment.
 - Running the script before activating the virtual environment.
 - Editing the Docker file but not restarting the container (`up -d` won't apply changes unless recreated).
 - Expecting UI details after the script finishes – the 4040 UI disappears immediately when no application runs.
-

Deliverables (What to Submit)

1 – Spark Master UI Screenshot

- Must show 1 master + 1 worker connected
- Must show cores and RAM recognized by the worker

2 – Terminal Output of Your Spark Application

- Full output of running `lab1_hello_spark.py`

3 – Spark Application UI Screenshots

- Jobs tab
- One DAG visualization
- Stage details for any stage
- Executors tab

4 – Short Notes (3–5 bullet points)

Answer:

- What does the DAG represent?
- How many stages ran during the script?
- How many tasks per stage?
- What did you notice in the Executors tab?

- What pattern did you observe in job triggering?
-

Grading Criteria

Criterion	Points
Docker cluster fully running + UI screenshot	4
PySpark locally installed + version test	3
First Spark application working	4
Screenshots from 4040 Spark UI	5
Notes demonstrating understanding	4
Total	20 points