**Databricks Hands-On Lab Guide**

**Lab Overview**

This lab will guide you step-by-step to practice:

1. Creating notebook widgets & using dbutils parameters.
2. Returning JSON from a notebook to a pipeline.
3. Running jobs on job clusters vs interactive clusters.
4. Applying and testing cluster policies.
5. Building a fan-out / fan-in workflow.

**Pre-requisites**

* Access to a Databricks workspace.
* Ability to create clusters and jobs.
* Admin permissions for cluster policy creation (for section 4).
* Sample dataset in DBFS (e.g., /databricks-datasets/airlines/part-00000).

**Lab 1 – Notebook Widgets & DBUtils Parameters**

**Steps:**

1. **Create a new Notebook** in your workspace named Widget\_Demo.
2. Add a text widget:

dbutils.widgets.text("input\_path", "/databricks-datasets/airlines/part-00000")

dbutils.widgets.text("year", "2020")

1. Retrieve widget values:

input\_path = dbutils.widgets.get("input\_path")

year = dbutils.widgets.get("year")

print(f"Processing data from: {input\_path} for year: {year}")

1. Run the notebook and change widget values from the **UI**.

**Expected Output:**  
The notebook should print the updated path and year based on your inputs.

**Lab 2 – Return JSON to Pipeline**

**Steps:**

1. Create a notebook named Return\_JSON\_Demo:

import json

dbutils.widgets.text("rows\_processed", "1000")

rows\_processed = int(dbutils.widgets.get("rows\_processed"))

output\_data = {"status": "success", "rows": rows\_processed}

dbutils.notebook.exit(json.dumps(output\_data))

1. Create another notebook Caller\_Notebook:

result = dbutils.notebook.run("Return\_JSON\_Demo", 60, {"rows\_processed": "5000"})

print(f"Result from child: {result}")

1. Run Caller\_Notebook.

**Expected Output:**  
Result from child: {"status": "success", "rows": 5000}

**Lab 3 – Job vs Interactive Clusters**

**Steps:**

**Part A – Interactive Cluster**

1. Create an **interactive cluster** in the Databricks UI.
2. Attach your notebook from Lab 1 and run it.
3. Observe that the cluster stays running after execution.

**Part B – Job Cluster**

1. Create a new job in **Workflows**.
2. Add a task to run Widget\_Demo with job cluster configuration.
3. Schedule it to run once immediately.
4. Observe that the job cluster is **created at start** and **terminated automatically**.

**Expected Learning:**  
Interactive clusters persist for dev work, job clusters are ephemeral for scheduled runs.

**Lab 4 – Cluster Policies**

**Steps:**

1. Go to **Admin Console → Cluster Policies**.
2. Create a new policy:

{

"spark\_version": { "type": "fixed", "value": "10.4.x-scala2.12" },

"node\_type\_id": { "type": "allowed", "values": ["Standard\_DS3\_v2"] },

"autotermination\_minutes": { "type": "fixed", "value": 30 }

}

1. Save the policy.
2. Create a new cluster using this policy and attempt to change disallowed settings.

**Expected Outcome:**  
Disallowed fields will be locked or restricted.

**Lab 5 – Fan-out / Fan-in Workflow**

**Steps:**

1. Create three notebooks:
   * **Parent\_Notebook**:

dbutils.notebook.run("Task\_A", 60)

dbutils.notebook.run("Task\_B", 60)

dbutils.notebook.run("Task\_C", 60)

* + **Task\_A**, **Task\_B**, **Task\_C**: Each prints a unique message.

1. In **Workflows**, create:
   * Task A as the first step.
   * Tasks B & C running in parallel after Task A (**Fan-out**).
   * A final Task D that runs after both B & C complete (**Fan-in**).

**Expected Learning:**  
Parallelization speeds up pipelines; fan-in allows aggregation after parallel work.