**Notebook Widgets/DBUtils params · Return JSON to pipeline · Job vs Interactive Clusters · Cluster Policies · Fan-out / Fan-in Graph**

**1. Notebook Widgets & DBUtils Parameters**

**Purpose:**  
Notebook widgets enable you to parameterize Databricks notebooks so they can run dynamically with different inputs without editing the code.

**Types of Widgets:**

* **Text widget** – For free-form input.  
  dbutils.widgets.text("param\_name", "default\_value")
* **Dropdown widget** – Select from a list of options.  
  dbutils.widgets.dropdown("param\_name", "default", ["opt1", "opt2"])
* **Multiselect widget** – Choose multiple options.  
  dbutils.widgets.multiselect("param\_name", "default", ["opt1", "opt2"])
* **Combobox widget** – Combination of text + dropdown.

**Retrieving Values:**

value = dbutils.widgets.get("param\_name")

**Use Cases:**

* Passing file paths, dates, or environment names into notebooks.
* Running the same notebook for multiple datasets.

**2. Return JSON to Pipeline**

**Purpose:**  
When using Databricks notebooks as tasks in a workflow, you may want to pass structured results to the next task.

**Key Method:**

import json

output\_data = {"status": "success", "count": 150}

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

**How It Works:**

* dbutils.notebook.exit() returns the string to the caller.
* The calling notebook or job task can capture and parse the JSON.
* Common for multi-step pipelines where downstream logic depends on upstream results.

**Example Use Case:**

* Task A ingests data and returns the number of rows processed.
* Task B decides whether to continue based on that count.

**3. Job vs Interactive Clusters**

**Job Clusters:**

* **Lifecycle:** Created automatically for a job run, destroyed afterward.
* **Use Case:** Automated workloads, scheduled jobs.
* **Benefits:** Cost efficiency, consistent environment per run.
* **Limitations:** Not for manual, exploratory work.

**Interactive Clusters:**

* **Lifecycle:** Manually created and persist until terminated.
* **Use Case:** Development, debugging, data exploration.
* **Benefits:** Faster iteration, multiple users can attach.
* **Limitations:** Higher cost if left running.

**Key Decision Factors:**

* Cost control (prefer job clusters for scheduled jobs).
* Collaboration needs (prefer interactive clusters for shared dev work).

**4. Cluster Policies**

**Purpose:**  
Enforce governance on cluster configurations to control cost, ensure compliance, and maintain security.

**Capabilities:**

* Restricting instance types.
* Enforcing auto-termination times.
* Limiting max/min worker counts.
* Locking down installed libraries or Spark versions.

**Example Policy Rule:**

{

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

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

"autotermination\_minutes": { "type": "fixed", "value": 60 }

}

**Benefits:**

* Prevents overspending on large clusters.
* Standardizes environments.
* Ensures compliance with org standards.

**5. Fan-out / Fan-in Graph**

**Fan-out:**

* **Definition:** One task produces outputs that feed into multiple parallel tasks.
* **Benefit:** Improves performance by parallelizing independent steps.

**Fan-in:**

* **Definition:** Multiple tasks converge into a single downstream task.
* **Benefit:** Aggregates results or synchronizes workflow before moving forward.

**Example Workflow:**

Task A → (Task B, Task C, Task D in parallel) → Task E

* **A → B/C/D**: Fan-out.
* **B/C/D → E**: Fan-in.

**Use Cases:**

* Data processing for multiple partitions in parallel.
* Running model training in parallel for different datasets, then aggregating results.

When preparing for interviews or practical work with Databricks:

* Practice creating a **parameterized notebook** and running it with different values.
* Experiment with **dbutils.notebook.exit** to return JSON between tasks.
* Compare **job clusters** vs **interactive clusters** in terms of cost and setup time.
* Review **cluster policies** in the admin console to understand constraints.
* Draw **fan-out/fan-in** diagrams for existing pipelines to spot optimization opportunities.