**Databricks JOBS Issues with Mitigations**

**Databricks Jobs Issues Comparison Table with Mitigations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Jobs Issue | One-Liner Description | Where It Typically Arises | Jobs Areas Most Affected | Mitigation Strategies |
| 1 | Inefficient Scheduling | Overlapping or poorly staggered jobs overload clusters or warehouses. | Workflow orchestration | Clusters, SQL Warehouses | Use job dependencies; stagger schedules; monitor concurrency limits. |
| 2 | Missing Alerts and Notifications | Job failures go unnoticed due to lack of configured alerts. | Production pipelines | Pipelines, Monitoring | Configure email or webhook alerts on failure or SLA breach. |
| 3 | Dependency Drift | Libraries or environment dependencies change, causing previously working jobs to fail. | Cluster updates, library installation | Clusters, Jobs | Pin library versions; use init scripts; validate environments before deployment. |
| 4 | Credential Exposure | Secrets are hard-coded into job configurations or notebooks. | Job setup, automation scripts | Jobs, Clusters, Notebooks | Store credentials in Databricks Secrets; reference them securely in configs. |
| 5 | Excessive Retries | Jobs retry too many times, masking root causes and increasing costs. | Retry policy configuration | Clusters, Pipelines | Set appropriate retry limits; monitor logs to resolve recurring failures. |
| 6 | Inefficient Cluster Configuration | Jobs run on oversized or misconfigured clusters, wasting resources. | Cluster selection for jobs | Compute Resources | Use job clusters with autoscaling and auto-termination; right-size workers and drivers. |
| 7 | Lack of Parameterization | Hard-coded paths and configurations make jobs inflexible across environments. | Notebook jobs, production pipelines | Jobs, Notebooks | Use parameterized workflows; define variables for environment-specific configs. |
| 8 | Lack of Audit Trail | No clear record of who modified or triggered jobs, complicating accountability. | Multi-user environments | Jobs, Pipelines | Use job history and Git-backed repos; review audit logs regularly. |
| 9 | Inefficient Data Writes | Jobs write unoptimized small files or unpartitioned data, degrading performance. | Data ingestion and processing jobs | Delta Tables, Storage | Use coalesce and partitioning strategies to optimize file sizes. |
| 10 | Orphaned Active Runs | Stuck or orphaned runs keep consuming compute, driving up costs. | Long-running or interrupted workflows | Clusters, Pipelines | Enable timeout thresholds; monitor active runs; terminate stalled jobs proactively. |

**Quick Reference**

* **Jobs:** Scheduled or triggered workflows executing notebooks, JARs, or Python scripts.
* **Pipelines:** Orchestrated multi-task workflows.
* **Job Clusters:** Dedicated clusters created for job execution.
* **Retries:** Automatic attempts to re-run failed tasks.
* **Alerts:** Notifications triggered on failures or SLA violations.

**Example Mitigation Commands and Configurations**

**Configure Alerts on Failure:**

* In the Job UI > *Edit Job* > *Notifications*, set:
  + Email on failure
  + Slack webhook
  + PagerDuty integration

**Set Retry Policies:**

json

CopyEdit

"max\_retries": 3,

"min\_retry\_interval\_millis": 60000

**Use Parameterized Notebooks:**

python

CopyEdit

dbutils.widgets.text("input\_path", "")

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

**Secure Credentials with Secrets:**

python

CopyEdit

dbutils.secrets.get(scope="prod-secrets", key="storage-key")

**Enable Auto-Termination on Job Clusters:**

* In cluster settings for the job:

json

CopyEdit

"autotermination\_minutes": 20

**Optimize Data Writes:**

python

CopyEdit

(

df.repartition(20)

.write

.format("delta")

.mode("overwrite")

.option("overwriteSchema", "true")

.save("/mnt/delta/optimized\_data")

)

**Audit Job History:**

* In the UI: *Jobs* > *Job Runs* > *View Details*
* Or via REST API to export run logs.