15 Critical Databricks Mistakes Advanced Developers Make: Security, Workflows, Environment

Ganesh R

Azure Data Engineer

Uncover 15 critical Databricks mistakes that Advanced developers often encounter. Learn essential tips on workflow optimization, security best practices, Git integration, environment management, an effective cluster strategy, and cost-saving tips.

I continue sharing practical information about common mstakes in Databricks and ways to fix them. I recommend reading the first part: « 11 Common Databricks Mistakes Beginners Make: Best Practices for Data Management and Coding »

1. Ignoring UTC zone when calculating dates

Mistake: Many developers use current date functions without considering Databricks clusters running in the UTC zone. This results in incorrect date calculations when scripts are run late at night or early in the morning in your local time zone, which can result in data being processed for the wrong day.

```
-- Problematic approach (uses UTC) if you are using Spark SQL
% sql
SELECT * FROM TABLE
WHERE FILE_DATE >= current_date ()
```

```
# Problematic approach (uses UTC) if you are using Spark
today = datetime.now().date()
```

How to fix it:

The first way is to set the default time zone on the cluster:

```
# Set the default time zone for a Spark session
spark.conf. set ( "spark.sql.session.timeZone" , "America/New_York" )
```

Here’s how to set up your time zone for Spark:

```
from datetime import datetime
import pytz

local_tz = pytz.timezone( 'America/New_York' )          # or your local time zone
today = datetime.now(pytz.UTC).astimezone(local_tz).date()
```

Here’s how to set timezone in Spark SQL:

```
% sql
SELECT * FROM TABLE
WHERE FILE_DATE >= date (from_utc_timestamp(current_timestamp (), 'EST' ))
```

2. Working in a single environment without proper separation of development and production.

Mistake: Working in a single Databricks environment without a clear separation between Dev and Prod is a common mistake that can lead to data loss. Ideally, you should create separate Databricks workspaces for different environments (e.g., Dev, QA, Prod). It also happens that, due to inexperience, developers manually change tables for DEV and PROD in each request.

How to fix it: However, creating multiple environments is not always feasible due to budget constraints or other limitations. When multiple separate environments can't be set up, a practical compromise is logical separation within a single environment. One convenient approach is to use schema and table prefixing within your Notebooks.

It is necessary to divide the workspace with Notebooks into two folders DEV and PROD. Or to develop in personal folders, and all Notebooks that will be installed on schedule should be moved to the common PROD folder.

Start each notebook by explicitly defining environment parameters, such as schema_prefix variables. In this example, we will define the path to the Notebook and check if the name contains the word (PROD):

```
# defining DEV or PROD environment
notebook_path = (dbutils.notebook.entry_point.getDbutils().notebook().getContext
if "(PROD)" in notebook_path.upper():
        schema_prefix = "catalog.prod_schema"
else:
        schema_prefix = "catalog.dev_schema"
```

And when we use the query, and the Notebook is not located in the folder named (PROD), then we will not spoil the real tables:

```
spark.sql(f"""
DELETE FROM {schema_prefix}.clients
WHERE
        DATE = current_date()
""")
```

3. Building long chains of processes using %run commands within notebooks, instead of proper Workflow orchestration.

Mistake: Many developers working with notebooks often rely excessively on the %run magic command to execute external scripts or notebooks. While

%run is convenient and suitable for quickly loading variables or libraries

across notebooks, some developers extend this approach to build complex, lengthy processing chains. This habit results primarily from a lack of awareness of specialized workflow orchestration tools.

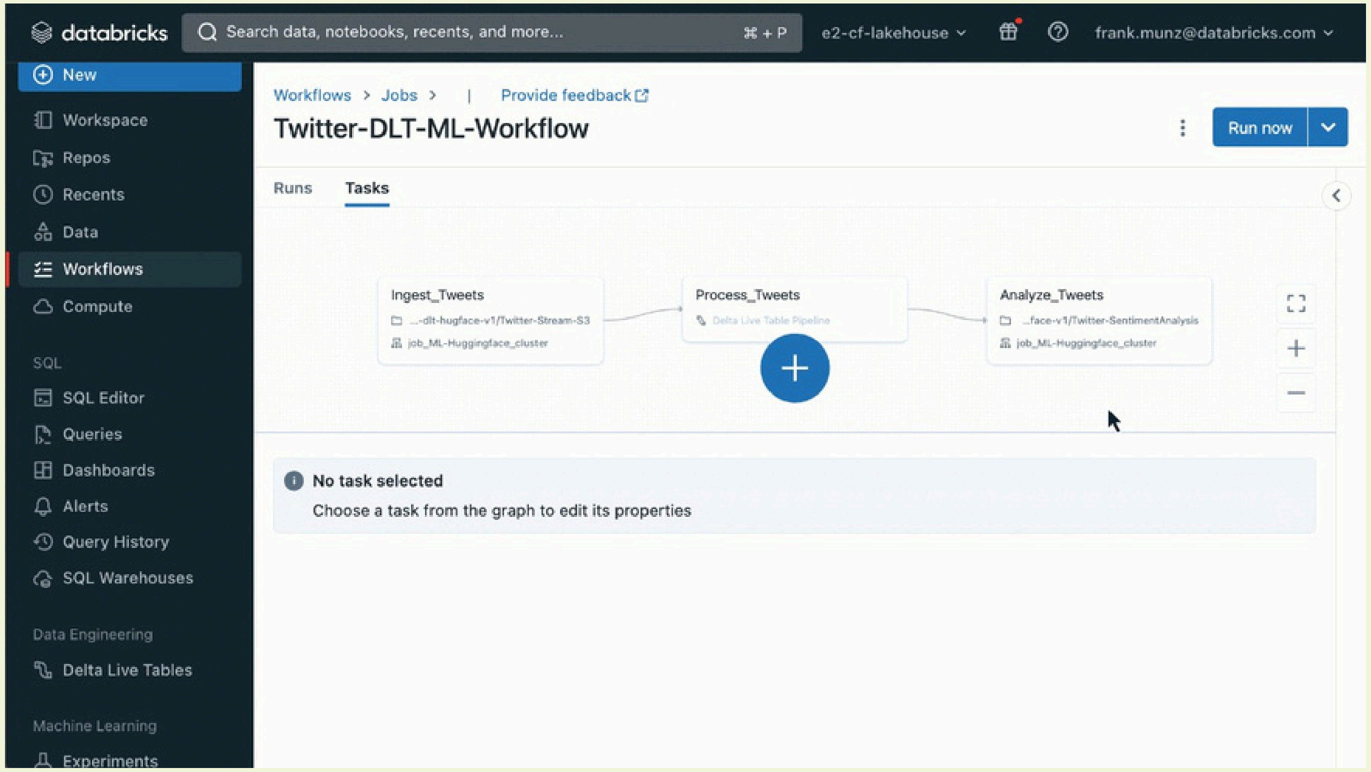
%run .variables

Although it might seem practical at first, relying heavily on %run chains can quickly become counterproductive. Such a strategy makes processes difficult to debug, track, and scale. Long %run cascades also become fragile: if one step fails, the entire process may halt unexpectedly.

How to fix it: Instead of extensive use of %run commands, it is recommended to adopt Databricks Workflows, a robust orchestration solution built directly into Databricks notebooks for seamless pipeline management. Databricks Workflows offer distinct benefits such as:

- Integrated Environment: Effortlessly orchestrate notebooks, jobs, and tasks within a unified Databricks workspace.
- Clear Task Dependencies: Define and manage dependencies visually, simplifying complex process flows.
- Fault Tolerance: Easily handle errors by setting retries, conditional logic, and alerts, improving robustness.
- Enhanced Monitoring and Logging: Access detailed logs, metrics, and performance tracking within the Databricks platform to quickly identify and address workflow issues.
- Flexible Scheduling Options: Schedule workflows based on time intervals or trigger conditions, automating execution according to business requirements.
- Simplified Scaling: Effortlessly scale and modify workflows to accommodate evolving needs without extensive refactoring.

By leveraging Databricks Workflows, teams significantly enhance flexibility, transparency, and stability, eliminating the limitations of chaining notebook executions with %run.


databricks
Search data, notebooks, recents, and more...
⌘ + P
e2-cf-lakehouse
frank.munz@databricks.com
New
Workspace
Repos
Recents
Data
Workflows
Compute
SQL
SQL Editor
Queries
Dashboards
Alerts
Query History
SQL Warehouses
Data Engineering
Delta Live Tables
Machine Learning
Experiments
Workflows > Jobs > | Provide feedback
Twitter-DLT-ML-Workflow
Run now
Runs
Tasks
Ingest_Tweets
...-dlt-hugface-v1/Twitter-Stream-S3
job_ML-Huggingface_cluster
Process_Tweets
Delta Live Table Pipeline
Analyze_Tweets
...face-v1/Twitter-SentimentAnalysis
job_ML-Huggingface_cluster
No task selected
Choose a task from the graph to edit its properties

4. Failing to grant workflow permission to other team members/groups.

Mistake: Failing to grant workflow permission to other team members or groups. By default, workflow visibility is restricted only to its creator.

How to fix it: After creating your workflow, navigate to workflow settings or permissions. Explicitly add team members or groups who require access. Define appropriate permission levels: Can Manage or Can View according to their responsibilities.

5. Forgetting to set threshold alerts or monitoring on Databricks Workflows.

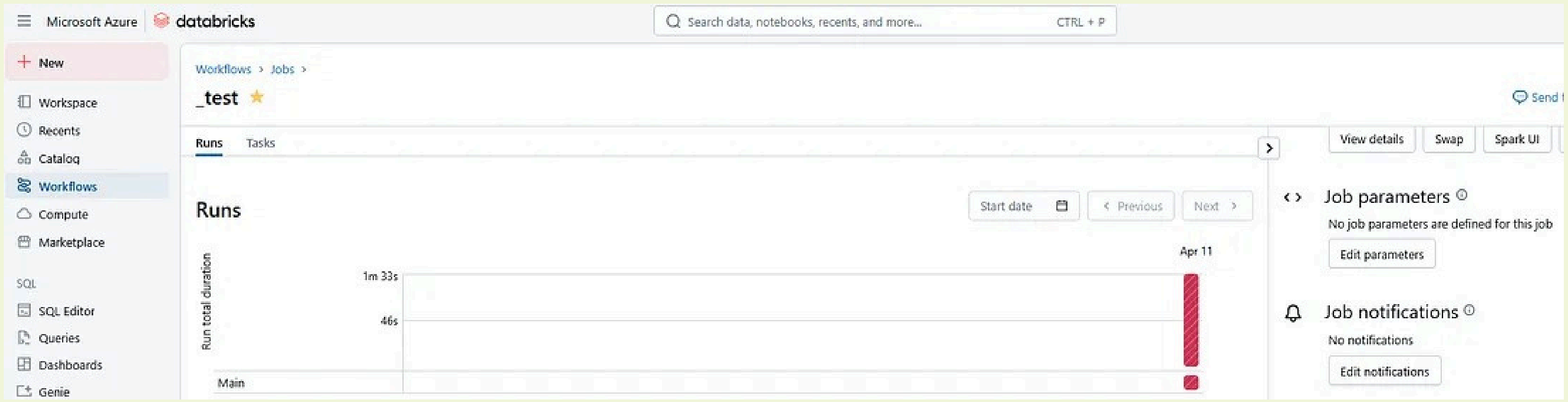
Mistake: Forgetting to set threshold alerts or monitoring parameters in Databricks workflows.

Without proper alert thresholds, workflow failures or unexpected performance degradation can go unnoticed for days, resulting in inefficient resource usage and unnecessary budget consumption.

How to fix it: Review historical execution durations of Notebook or Workflow tasks under typical operating conditions. Determine the normal run duration, then set your threshold alerts to approximately double this typical duration. Effective threshold monitoring promptly alerts teams about anomalies or stalled workflows, ensuring quick issue resolution and significant cost savings by reducing prolonged unnecessary resource use.

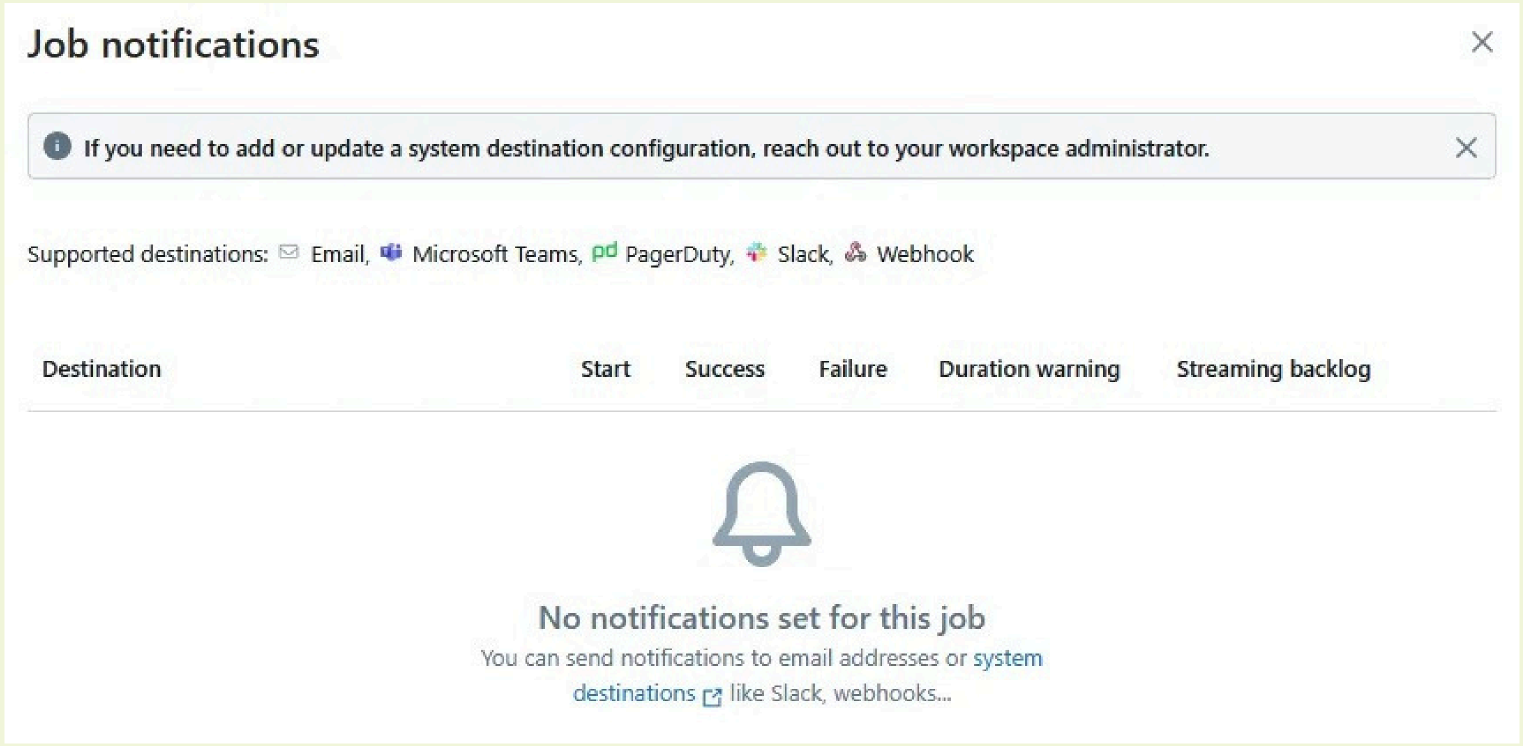
6. Ensure Your Team Receives Error Notifications in Databricks Workflows.

Mistake: In Databricks Workflows, error notifications by default go only to the workflow creator (author). However, sometimes the workflow developer forgets to change this default and assigns the notifications to a working group, or worse, removes even their contact from notification settings. As a result, when workflows fail, nobody gets notified promptly, and important issues go unnoticed.



No job parameters are defined for this job

No notifications


Job notifications

If you need to add or update a system destination configuration, reach out to your workspace administrator.

Supported destinations: Email, Microsoft Teams, PagerDuty, Slack, Webhook

Destination	Start	Success	Failure	Duration warning	Streaming backlog

No notifications set for this job

You can send notifications to email addresses or system destinations like Slack, webhooks...

How to fix it:

- Always assign a shared working group as recipients of workflow error notifications rather than to individual developers.
- Regularly audit Databricks workflows for proper notification settings to ensure no workflows are left without alert recipients.

7. Not using Job Clusters for workflows (instead mistakenly relying on interactive clusters).

Mistake: Using an interactive cluster rather than the dedicated job cluster for automated or scheduled tasks.

Databricks offers two main types of clusters:

- Interactive cluster — used early in a project or during script development. Ideal for interactive, exploratory tasks where quick modifications and manual execution are required. These clusters typically remain running as long as manually enabled.
- Job cluster — designed specifically for scheduled workflows or automated script execution. They automatically spin up when scheduled tasks start and immediately shut down upon completion, preventing unnecessary resource and budget waste.

How to fix it:

Always choose a Job cluster for scheduled scripts or workflows. Configure the Job cluster specifically for your script's resource needs.

Regularly review existing workflows to ensure they’re using Job clusters, not interactive clusters.

Using Job clusters instead of interactive clusters allows efficient resource management, reduces costs, ensures smoother operation of your scheduled processes, and prevents potential conflicts.

8. Setting up interactive Databricks clusters without auto- termination enabled.

Mistake: Setting up interactive Databricks clusters without enabling auto- termination results in clusters running continuously, even when nobody is actively using them. This oversight can quickly consume your Databricks budget unnecessarily.

How to fix it: When creating an interactive cluster, always enable the auto- termination feature in the cluster configuration. Set the recommended inactivity period (commonly 30–60 minutes), which ensures the cluster automatically shuts down after being idle for the specified timeframe.

9. Forgetting to run regular VACUUM on Delta tables.

Mistake: Forgetting to regularly run VACUUM on Delta tables can lead to accumulating unused data files. This wastes storage space and negatively impacts overall performance.

How to fix it: Regularly perform VACUUM operations to clean up old, unused data files. Or automate VACUUM by creating scheduled Databricks jobs aligned with each table's data update frequency:

```
VACUUM table_name RETAIN 168 HOURS; -- retain 7 days

VACUUM table_name RETAIN 1440 HOURS; -- retain 60 days
```

Regular VACUUM reduces storage costs, improves query efficiency, and ensures your Delta tables remain organized and optimized.

10. Storing passwords or secrets directly in code (instead of using Databricks Secrets).

Mistake: Including passwords or sensitive keys directly inside your code is a common security issue.

```
my_password = "SecretPassword123"
```

How to fix it: I propose a solution for Microsoft Azure. Key Vault Secrets is a secure cloud service for storing passwords, API keys, and other sensitive information. It helps you centralize, secure, and control access to secrets.

Typically, your cloud administrator creates a secret (such as a database password) in Azure Key Vault, and then you can securely access that secret from your Azure Databricks Notebook. This approach ensures that you are following security best practices, and sensitive information is never stored directly in your code.

Cloud Admin: Creates secret my-db-password) in Azure Key Vault (my-(keyvault).

You (in Databricks notebook): Link Azure Key Vault to Databricks using Secret Scope and access the secret from your Notebook:

```
my_password = dbutils.secrets.get(scope="my-keyvault-scope", key="my-db-password
```

11. Connecting directly to on-prem databases from cloud Notebooks.

Mistake: In some cases, companies leave access open from cloud notebooks directly to their on-premises databases or file systems. Employees who know connection details (host, port, and credentials) then connect directly. As mentioned before, employees might also mistakenly store passwords directly inside notebooks (covered previously in point 9).

Risks Associated:

Security vulnerabilities: Unauthorized access or password leak. Lack of audit trails and control: If everyone connects individually, it is difficult to track who accessed sensitive resources and when.
Performance issues: Direct queries might overload on-prem databases, causing disruptions for other users.
Compliance issues: Direct external connections might violate data regulations or policies.

How to fix it: Instead, companies should adopt centralized data loading frameworks or solutions. A correct, secure, and robust way is to use specialized tools or frameworks (such as Azure Data Factory pipelines, Apache Airflow, or securely managed ETL processes) to regularly and safely sync on-prem data to cloud storage or databases. Employees then connect notebooks and analytic applications only to pre-prepared and centrally managed datasets residing securely in the cloud.

12. Not utilizing Git version control integration.

Mistake: Your team is not using Git to keep track of script changes. That makes it hard to see who changed what, creates challenges in backing up scripts, and makes finding specific code very difficult.

How to fix it:

- Start using Git: Keep all your scripts in a Git repository. Save changes regularly: Frequently add and commit your updates to clearly track changes over time.
- Use branches: Make separate branches when working on new features or testing, so your main version remains stable.
- Train your team: Teach team members basic Git commands and explain why it’s useful.
- Easy script search: Use Git’s built-in tools to quickly find needed scripts or changes.

13. Not encrypting sensitive data when migrating from on- premise to cloud environments.

Mistake: When migrating databases from on-premise to Databricks cloud environments, developers often transfer tables containing sensitive data without implementing proper encryption. This creates significant security vulnerabilities as cloud-stored data requires stronger protection. Common issues include:

- Moving customer PII, financial data, or healthcare information in plaintext.
- Storing sensitive columns (SSNs, credit card numbers, addresses) without encryption.
- Failing to identify which table columns need encryption.
- Using actual PII data in development and testing environments.
- Disregarding company security policies for handling sensitive data.

How to fix it:

1. Implement Strong Encryption Standards:

Encrypt sensitive columns at rest and in transit using proven encryption standards like AES-256. Databricks, along with cloud providers (AWS KMS or

Azure Key Vault), offers integration for centralized key management.

```
from pyspark.sql.functions import col, sha2

encrypted_df = customer_df.withColumn("SSN_encrypted", sha2(col("SSN"),256))
```

```
%sql
SELECT
*,
 sha2(SSN, 256) AS SSN_encrypted
FROM customer
```

2. Utilize Data Masking and Anonymization for Non-production Environments:

For testing, development, or analytics environments, mask or anonymize sensitive data to minimize exposure risk.

```
%sql
SELECT
CONCAT('user', FLOOR(RAND()*10000), '@example.com') AS masked_email,  CONCAT('**** **** ****
', RIGHT(credit_card_number,4)) AS masked_credit_card FROM customer_transactions
```

14. Leaving PII Data Exposed in Files During Data Transfers.

Mistake: While developers often focus on encrypting database tables, they frequently overlook that sensitive PII data also exists in files (CSV, JSON, Parquet, Excel, etc.) being transferred to Databricks. Common oversights include:

Uploading unencrypted files containing customer information to DBFS or cloud storage Transferring sensitive files over insecure protocols (FTP, HTTP instead of SFTP, HTTPS) Neglecting to delete sensitive files after processing Failing to implement proper access controls for files containing PII

Sending files with PII data via email, which is highly insecure and violates most data protection policies

How to fix it: To prevent unauthorized access and exposure of PII data within files during transfers, the following best practices should be implemented:

- File encryption (e.g., with GPG): Always encrypt sensitive files such as CSV, JSON, Excel, or Parquet with an established encryption standard such as GPG (GNU Privacy Guard) before transferring them. This ensures that only authorized individuals or processes with the correct decryption keys can access the data, significantly reducing the risk of accidental disclosure.
- Secure protocols: Use secure transfer protocols such as HTTPS, SFTP, or SCP instead of insecure options such as FTP or HTTP. These protocols provide built-in data encryption during transfer, thereby preventing eavesdropping and unauthorized access.
- Proper access control: Implement strict role-based permissions and access controls for sensitive files on file systems, cloud storage, and DBFS. Ensure that only individuals who require access can read, download, or process these data sets.

- Automated cleanup: Set up automated processes to securely delete or archive sensitive files after processing. Avoid leaving unnecessary copies of files in storage locations (cloud storage, DBFS, etc.). Avoid sending data via email: Communicate strict policies prohibiting the use of email to send sensitive PII. Provide clear instructions and secure alternatives (e.g., links to restricted cloud storage or secure file transfer por tals).

15. Manually saving files received by email for processing by the script.

Mistake: Some employees manually download files received via email and then upload them individually to storage locations (such as a shared folder or cloud storage) for processing by Databricks notebooks or scripts. This manual process can lead to delays, mistakes, and missing data, and can reduce workflow efficiency.

How to fix it: Automate this process using Power Automate in combination with Databricks Volumes:

Set up Power Automate to automatically detect incoming emails with attachments.
Automatically save attachments directly to Databricks Volumes (the new built-in file storage in Databricks).

Databricks Notebooks can efficiently and directly access these files in volumes, ensuring simple and secure processing.
Be sure to grant the appropriate permissions to the Databricks workgroup and workspace for the volume storage.

Do you recognize yourself in any of these mistakes?

If you find yourself making any of these advanced Databricks mistakes, it’s worth reviewing the basics. Check out our previous article, “ 11 Common Databricks Mistakes Beginners Make: Best Practices for Data Management and Coding,” to make sure you’re confident in your grasp of the basics.

Follow for more content like this Azure Cloud for Data Engineering

Ganesh R

Azure Data Engineer

https://www.linkedin.com/in/rganesh0203/