Azure Databricks Best Practice Guide

Azure Databricks (ADB) has the power to process terabytes of data, while simultaneously running heavy data science workloads. Over time, as data input and workloads increase, job performance decreases. As an ADB developer, optimizing your platform enables you to work faster and save hours of effort for you and your team.

1. **Use Cache table/dataframe for re-usable tables or confirmed dimentions.**

   cache() is an Apache Spark transformation that can be used on a DataFrame, Dataset, or RDD when you want to perform more than one action. cache() caches the specified DataFrame, Dataset, or RDD in the memory of your cluster's workers. Since cache() is a transformation, the caching operation takes place only when a Spark action (for example, count(), show(), take(), or write()) is also used on the same DataFrame, Dataset, or RDD in a single action.

```
df=spark.table("input_table_name")
df.cache.take(5)                    # Call take(5) on the DataFrame df, while also caching it
df.count()                          # Call count() on the DataFrame df
```

```
df1=spark.read.parquet(input_path1)
df2=spark.read.parquet(input_path2)
df1.cache()                                            # Cache DataFrame df1

joined_df = df1.join(df2, df1.id==df2.id, 'inner')    # Join DataFrame df1 and df2
filtered_df = joined_df.filter("name == 'John'")       # Filter the joined DataFrame for the name "John"
df1.count()                                            # Call count() on the cached DataFrame
filtered_df.show()                                     # Show the filtered DataFrame filtered_df
```

```
df=spark.table("input_table_name")
df.cache.count()                    # Call count() on the DataFrame df, while also caching it
df.count()                          # Call count() on the DataFrame df
df.filter("name=='John'").count()
```

   You should call count() or write() immediately after calling cache() so that the entire DataFrame is processed and cached in memory.

2. **Create partitions on every table and for fact tables use partition column on key join column like country_code, city, market_code**

   Delta tables in ADB support partitioning, which enhances performance. You can partition by a column if you expect data in that partition to be at least 1 GB. If column cardinality is high, do not use that column for partitioning. For example, if you partition by user ID and there are 1M distinct user IDs, partitioning would increase table load time. Syntax example:

```
CREATE TABLE events (
DATE DATE
,eventId STRING
,eventType STRING
,data STRING
) USING delta PARTITIONED BY (DATE)
```

```
-- create a partitioned table and insert a few rows.
USE salesdb;
CREATE TABLE customer(id INT, name STRING) PARTITIONED BY (state STRING, city STRING);
INSERT INTO customer PARTITION (state = 'CA', city = 'Fremont') VALUES (100, 'John');
INSERT INTO customer PARTITION (state = 'CA', city = 'San Jose') VALUES (200, 'Marry');
INSERT INTO customer PARTITION (state = 'AZ', city = 'Peoria') VALUES (300, 'Daniel');
```

```
-- Use the PARTTIONED BY clause in a table definition
> CREATE TABLE student(university STRING,
                       major      STRING,
                       name       STRING)
        PARTITIONED BY(university, major)
```

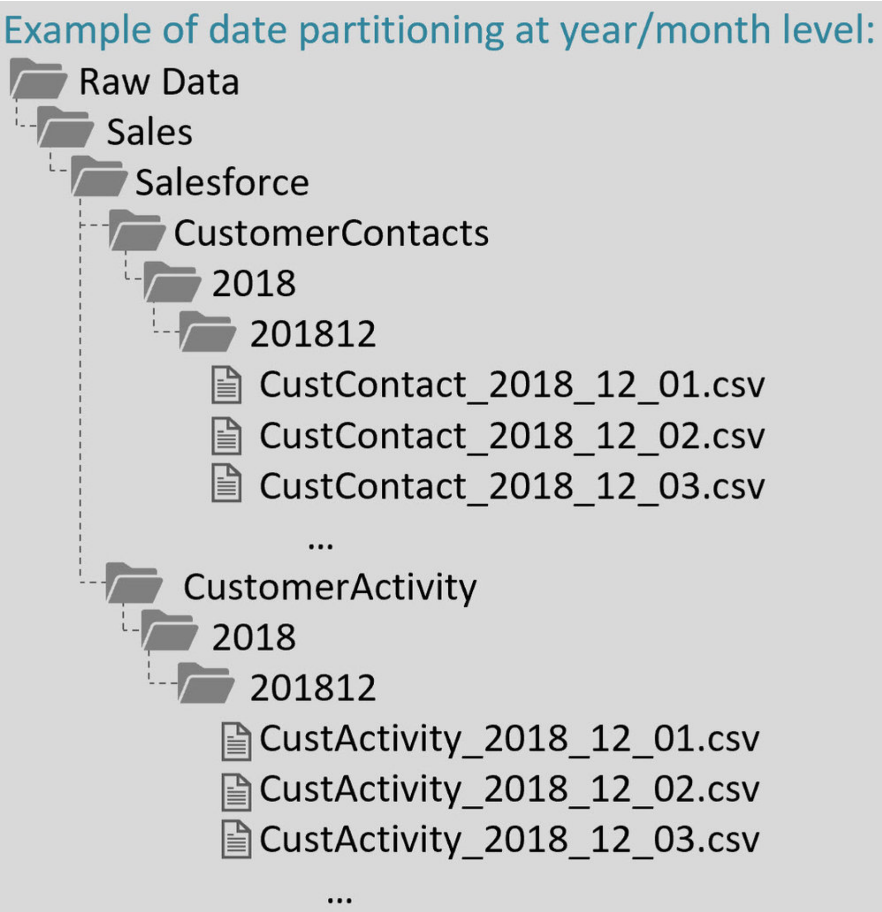
3. **Land data in Blob Store/ADLS partitioned into separate directory**

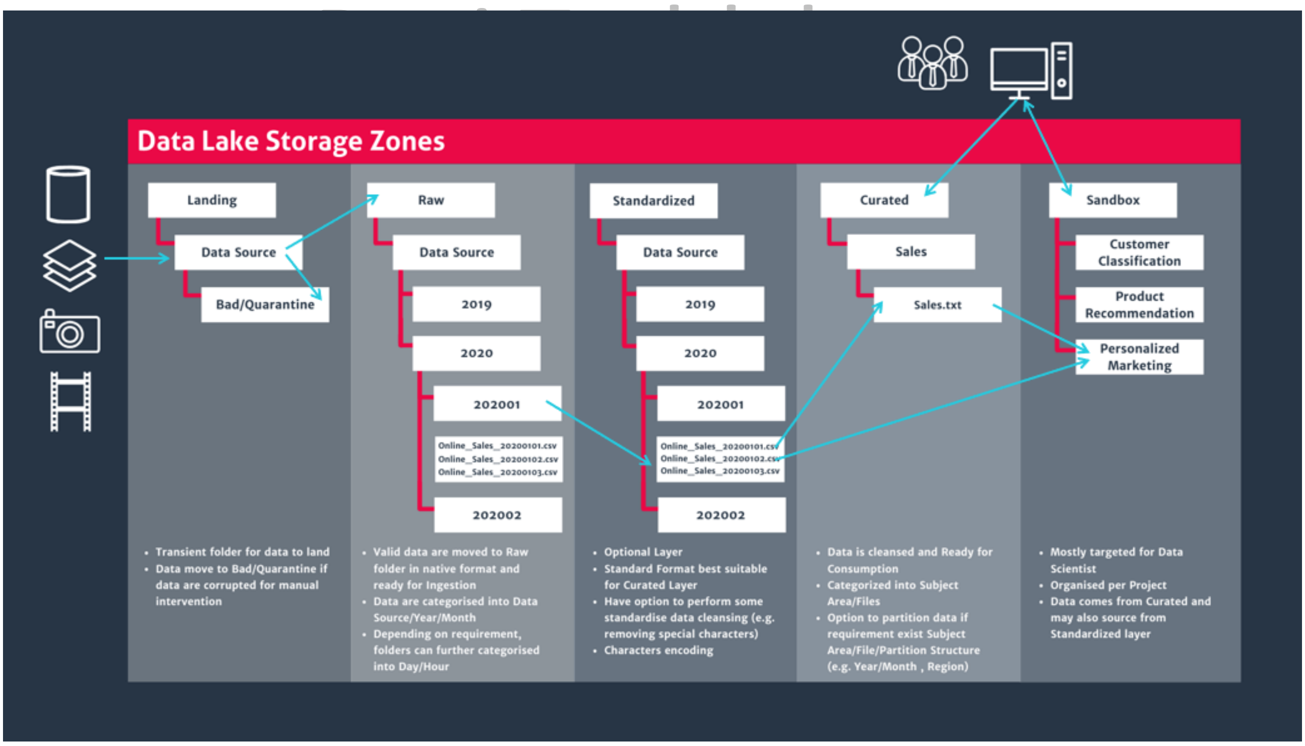
   Avoid high list cost on large directories like Hierarichal folder structure

A general template to consider might be the following layout:

{Region}/{SubjectMatter(s)}/{yyyy}/{mm}/{dd}/{hh}/

In this example, by putting the date at the end of the directory structure, you can use ACLs to more easily secure regions and subject matters to specific users and groups





4. Use Delta Lake performance features like OPTIMIZE with ZORDER

Z-Ordering (multi-dimensional clustering)

Z-Ordering is a technique to colocate related information in the same set of files. This co-locality is automatically used by Delta Lake on Databricks data-skipping algorithms to dramatically reduce the amount of data that needs to be read. To Z-Order data, you specify the columns to order on in the ZORDER BY clause:

```sql
SQL

OPTIMIZE events
WHERE date >= current_timestamp() - INTERVAL 1 day
ZORDER BY (eventType)
```

5. **Enable Auto Optimize option for all staging tables.**

Enable Auto Optimize

You must explicitly enable Optimized Writes and Auto Compaction using one of the following methods:

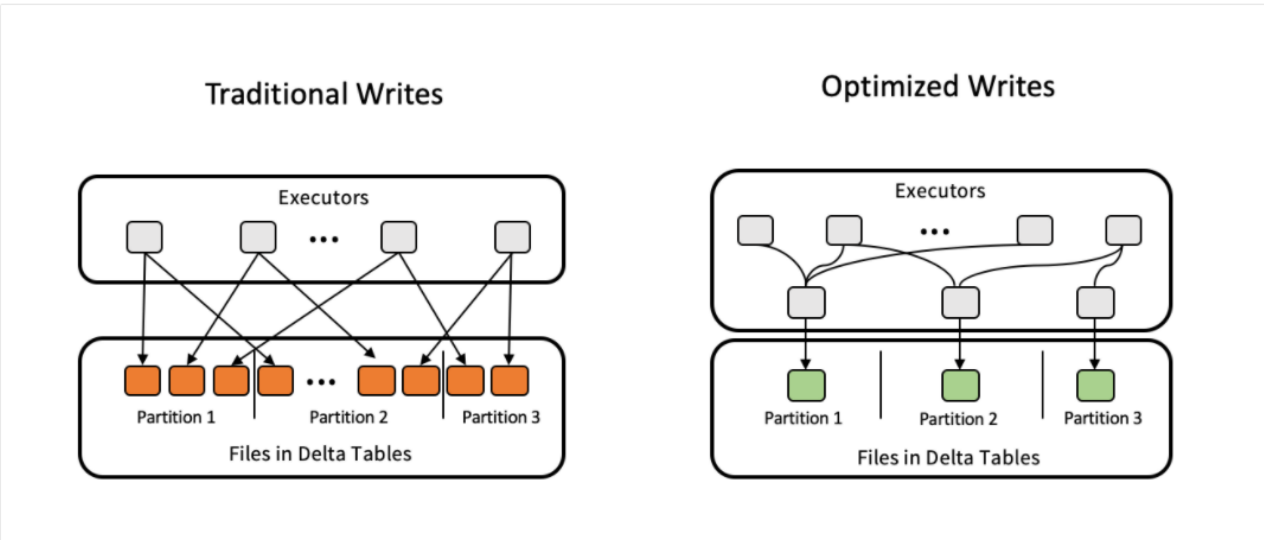
➢ **New table**: Set the table properties delta.autoOptimize.optimizeWrite = true and delta.autoOptimize.autoCompact = true in the CREATE TABLE command.

➢ Auto Optimize consists of two complementary features: **Optimized Writes** and **Auto Compaction**.

➢ **CREATE TABLE** student (id INT, name STRING, age INT) TBLPROPERTIES (delta.autoOptimize.optimizeWrite = **true**, delta.autoOptimize.autoCompact = **true**)

How Optimized Writes work

Databricks dynamically optimizes Apache Spark partition sizes based on the actual data, and attempts to write out 128 MB files for each table partition. This is an approximate size and can vary depending on dataset characteristics.



NOTE:

- ➢ Databricks does not support Z-Ordering with Auto Compaction as Z-Ordering is significantly more expensive than just compaction.
- ➢ Auto Compaction generates smaller files (128 MB) than **OPTIMIZE** (1 GB).
- ➢ Auto Compaction greedily chooses a limited set of partitions that would best leverage compaction. The number of partitions selected will vary depending on the size of cluster it is launched on. If your cluster has more CPUs, more partitions can be optimized.
- ➢ To control the output file size, set the Spark configuration **spark.databricks.delta.autoCompact.maxFileSize**. The default value is 134217728, which sets the size to 128 MB. Specifying the value 104857600 sets the file size to 100MB.

- ➢ spark.sql("set spark.databricks.delta.autoCompact.enabled = true")

6. Decide partition size (block size default is 128MB). Based on that it will create no of files at table.

Table size	Target file size	Approximate number of files in table
10GB	256MB	40
1TB	256MB	4096
2.56TB	256MB	10240
3TB	307MB	12108
5TB	512MB	17339
7TB	716MB	20784
10TB	1GB	24437
20TB	1GB	34437
50TB	1GB	64437
100TB	1GB	114437

7. Use hints for improving query performance like BROADCAST.

Join hints

Join hints allow you to suggest the join strategy that Databricks Runtime should use. When different join strategy hints are specified on both sides of a join, Databricks Runtime prioritizes hints in the following order: BROADCAST over MERGE over SHUFFLE_HASH over SHUFFLE_REPLICATE_NL. When both sides are specified with the BROADCAST hint or the SHUFFLE_HASH hint, Databricks Runtime picks the build side based on the join type and the sizes of the relations. Since a given strategy may not support all join types, Databricks Runtime is not guaranteed to use the join strategy suggested by the hint.

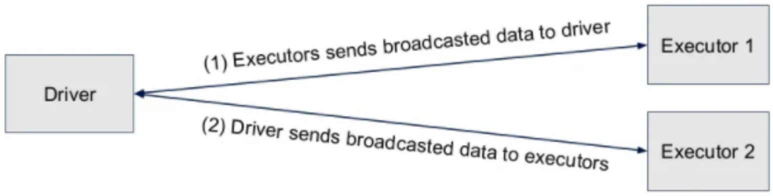
Join hint types

- **BROADCAST**

  Use broadcast join. The join side with the hint is broadcast regardless of autoBroadcastJoinThreshold. If both sides of the join have the broadcast hints, the one with the smaller size (based on stats) is broadcast. The aliases for BROADCAST are BROADCASTJOIN and MAPJOIN.

Broadcasting in Spark

- Spark's broadcasting mechanism is inefficient
  - Broadcasted data goes through the driver
  - Too much broadcasted data can run the driver out of memory



- **MERGE**

  Use shuffle sort merge join. The aliases
  for MERGE are SHUFFLE_MERGE and MERGEJOIN.

- **SHUFFLE_HASH**

  Use shuffle hash join. If both sides have the shuffle hash hints, Databricks Runtime chooses the smaller side (based on stats) as the build side.

- **SHUFFLE_REPLICATE_NL**

  Use shuffle-and-replicate nested loop join.

```
-- Join Hints for broadcast join
> SELECT /*+ BROADCAST(t1) */ * FROM t1 INNER JOIN t2 ON t1.key = t2.key;
> SELECT /*+ BROADCASTJOIN (t1) */ * FROM t1 left JOIN t2 ON t1.key = t2.key;
> SELECT /*+ MAPJOIN(t2) */ * FROM t1 right JOIN t2 ON t1.key = t2.key;

-- Join Hints for shuffle sort merge join
> SELECT /*+ SHUFFLE_MERGE(t1) */ * FROM t1 INNER JOIN t2 ON t1.key = t2.key;
> SELECT /*+ MERGEJOIN(t2) */ * FROM t1 INNER JOIN t2 ON t1.key = t2.key;
> SELECT /*+ MERGE(t1) */ * FROM t1 INNER JOIN t2 ON t1.key = t2.key;

-- Join Hints for shuffle hash join
> SELECT /*+ SHUFFLE_HASH(t1) */ * FROM t1 INNER JOIN t2 ON t1.key = t2.key;

-- Join Hints for shuffle-and-replicate nested loop join
> SELECT /*+ SHUFFLE_REPLICATE_NL(t1) */ * FROM t1 INNER JOIN t2 ON t1.key = t2.key;
```

Broadcast Join vs. Shuffle Join

Broadcast Join	Shuffle Join
Avoids shuffling the bigger side	Shuffles both sides
Naturally handles data skew	Can suffer from data skew
Cheap for selective joins	Can produce unnecessary intermediate results
Broadcasted data needs to fit in memory	Data can be spilled and read from disk
Cannot be used for certain outer joins	Can be used for all joins

Where applicable, broadcast join should be **faster** than shuffle join

8. **Use Repartition hints for balancing partitions.**

Partitioning hint types

- **COALESCE**

  Reduce the number of partitions to the specified number of partitions. It takes a partition number as a parameter.

- **REPARTITION**

  Repartition to the specified number of partitions using the specified partitioning expressions. It takes a partition number, column names, or both as parameters.

- **REPARTITION_BY_RANGE**

  Repartition to the specified number of partitions using the specified partitioning expressions. It takes column names and an optional partition number as parameters.

- **REBALANCE**

  The REBALANCE hint can be used to rebalance the query result output partitions, so that every partition is of a reasonable size (not too small and not too big). It can take column names as parameters, and try its best to partition the query result by these columns. This is a best-effort: if there are skews, Spark will split the skewed partitions, to make these partitions not too big. This hint is useful when you need to write the result of this query to a table, to avoid too small/big files. This hint is ignored if AQE is not enabled.

```
SELECT /*+ COALESCE(3) */ * FROM t;

SELECT /*+ REPARTITION(3) */ * FROM t;

SELECT /*+ REPARTITION(c) */ * FROM t;

SELECT /*+ REPARTITION(3, c) */ * FROM t;

SELECT /*+ REPARTITION_BY_RANGE(c) */ * FROM t;

SELECT /*+ REPARTITION_BY_RANGE(3, c) */ * FROM t;

SELECT /*+ REBALANCE */ * FROM t;

SELECT /*+ REBALANCE(c) */ * FROM t;
```

9. **Delete temporary tables after notebook execution**

Delete temporary tables that were created as intermediate tables during notebook execution. Deleting ta ook is scheduled daily.

```
spark.catalog.dropTempView("temp_view_name") //drops the table
```

```
spark.sql("drop view hvac");
```

10. **Use dbutils.fs.rm() to permanently delete temporary table metadata**

ADB clusters store table metadata, even if you use drop statements to delete. Before creating temporary tables, use dbutils.fs.rm() to permanently delete metadata. If you don't use this statement, an error message will appear stating that the table already exists. To avoid this error in daily refreshes, you must use dbutils.fs.rm().

11. **Use Lower() or Upper() when comparing strings or common filter conditions to avoid losing data**

ADB can't compare strings with different casing. To avoid losing data, use case conversion statements Lower() or Upper(). Example:

```
SELECT 'Bangalore' = 'bangalore' AS WithOutLowerOrUpper
,LOWER('BaNgalore') = 'bangalore' AS WithLower
,UPPER('BaNgaLore') = 'BANGALORE' AS WithUpper
```

12. **Use custom functions to simplify complex calculations**

If your calculation requires multiple steps, you can save time and by creating a one-step custom function. ADB offers a variety of built in SQL functions, however to create custom functions, known as user-defined functions (UDF), use Scala. Once you have a custom function, you can call it every time you need to perform that specific calculation.

13. **Use Delta tables for DML commands**

In ADB, Hive tables do not support UPDATE and MERGE statements or NOT NULL and CHECK constraints. Delta tables do support these commands, however running large amounts of data on Delta tables decreases query performance. So not to decrease performance, store table versions.

```
drop table IF EXISTS locations;
create table IF NOT EXISTS locations (id int,name string);
insert into locations
select 1,'Bangalore' union all
select 2,'Hyderabad' union all
select 3,'Chennai' union all
select 4,'Pune' union all
select 5,'Mumbai' union all
select 6,'Delhi' union all
select 7,'Vijag' union all
select 8,'Kolkatta'
```

```
delete FROM all_employee
  WHERE loc_id in (SELECT id FROM locations where name='Chennai')
```

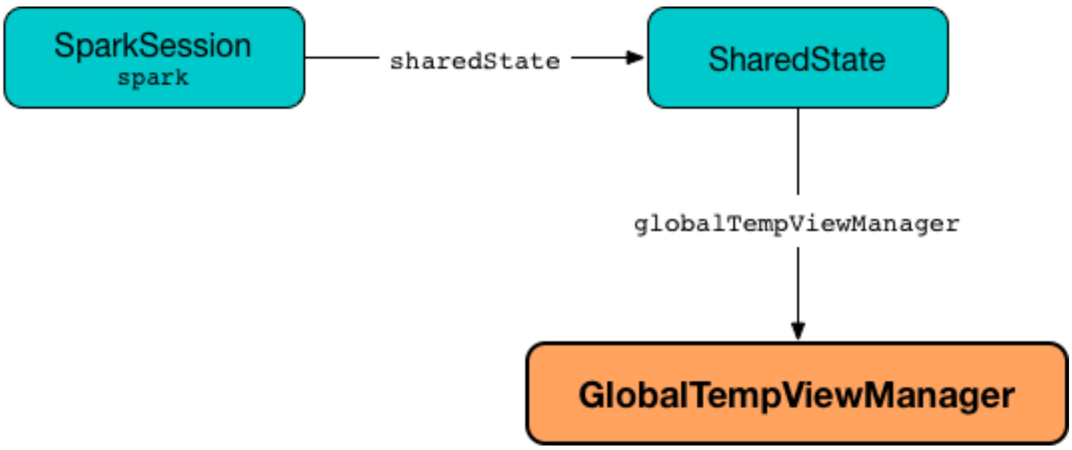
```
MERGE INTO events
USING updates
ON events.event_id = updates.event_id
WHEN MATCHED AND  updates.delete==true THEN
  delete
WHEN MATCHED THEN
  UPDATE SET events.data = updates.data
WHEN NOT MATCHED
  THEN INSERT (event_id, event_date, data,delete) VALUES (event_id,event_date, data,delete)
```

```
update all_employee
set loc_id=4
where id=3
```

14. Use views when creating intermediate tables

If you need to create intermediate tables, use views to minimize storage usage and save costs. Views are session-oriented and will automatically remove tables from storage after query execution. For optimal query performance, do not use joins or subqueries in views.

```
emp_df.createOrReplaceTempView("t_emp") # this view will be available at User Session
emp_df.createOrReplaceGlobalTempView("gv_emp")  # this view will be available at SparkSession level(cluster)
```




```
CREATE OR REPLACE VIEW V_employees as
SELECT * FROM all_employee
         WHERE working_years > 5;
```

15. Enable adaptive query execution (AQE)

AQE improves large query performance. By default, AQE is disabled in ADB. To enable it, use: set spark.sql.adaptive.enabled = true;

Enabling AQE

AQE can be enabled by setting SQL config spark.sql.adaptive.enabled to true (default false in Spark 3.0), and applies if the query meets the following criteria:

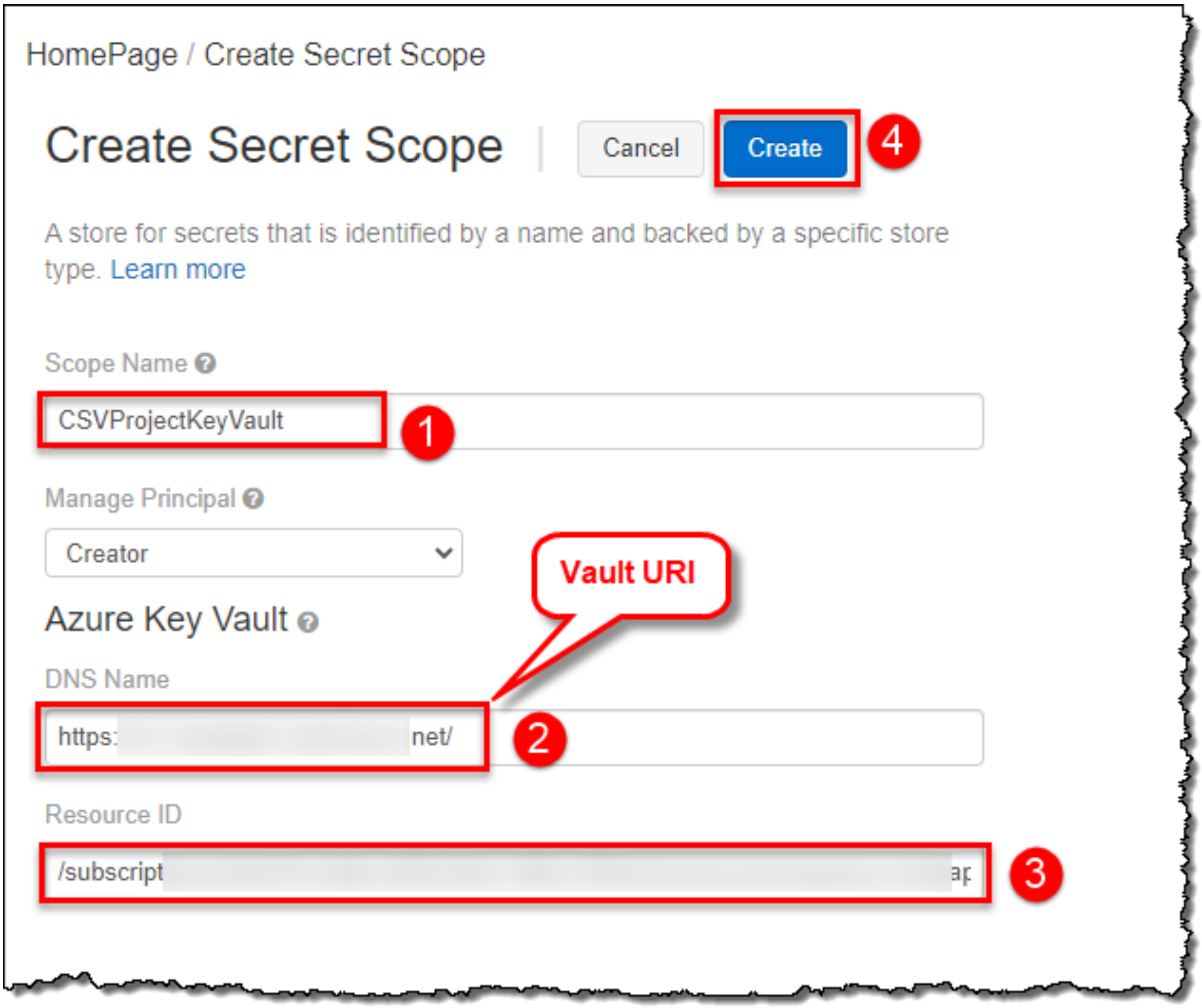
It is not a streaming query

It contains at least one exchange (usually when there's a join, aggregate or window operator) or one subquery

1. Optimizing Shuffles
2. Choosing Join Strategies
3. Handling Skew Joins
4. Understand AQE Query Plans
5. The AdaptiveSparkPlan Node
6. The CustomShuffleReader Node
7. Detecting Join Strategy Change

16. Use key vault credentials when creating mount points

When creating mount points to Azure Data Lake Storage (ADLS), use a key vault client ID and client secret to enhance security.



```python
# Python code

# Get list of all scopes
mysecrets = dbutils.secrets.listScopes()
# Loop through list
for secret in mysecrets:
  print(secret.name)
```

Mount an Azure Blob storage container

```python
dbutils.fs.mount(
  source = "wasbs://<container-name>@<storage-account-name>.blob.core.windows.net",
  mount_point = "/mnt/iotdata",
  extra_configs = {"fs.azure.account.key.<storage-account-name>.blob.core.windows.net":dbutils.secrets.get(scope = "<scope-name>", key = "<key-name>")})
```

17. Query directly on parquet files from ADLS

If you need to use the data from parquet files, do not extract into ADB in intermediate table format. Instead, directly query on the parquet file to save time and storage. Example: SELECT ColumnName FROM parquet.`Location of the file`

```sql
-- Creates a CSV table from an external directory
> CREATE TABLE student USING CSV LOCATION '/mnt/csv_files';
```

```sql
%sql
-- Creates a parquet table from an external directory
CREATE TABLE student USING parquet LOCATION '/mnt/parquet_files';
```

18. Choosing cluster mode for individual jobs execution and common jobs execution.

For individual job execution use **standard mode** cluster.

For group of jobs and multiple jobs with dependency tables in parallel or sequential load choose **High Concurrency Mode.**

	Standard Mode	High Concurrency Mode
Targeted User	Data Engineers	Data Scientists, Business Analysts
Languages	Scala, Java, SQL, Python, R	SQL, Python, R
Best Use	Batch Jobs for ETL	Data Exploration
Security Model	Single User	Multi User
Isolation	Medium	High
Table-level security	No	Yes
Query Preemption	No	Yes
AAD Passthrough	No	Yes

1. Deploy a shared cluster instead of letting each user create their own cluster.
2. Create the shared cluster in High Concurrency mode instead of Standard mode.
3. Configure security on the shared High Concurrency cluster, using **one** of the following options:
    - Turn on AAD Credential Passthrough if you're using ADLS
    - Turn on Table Access Control for all other stores

Workload → Attribute ↓	Interactive	Batch
Optimization Metric: What matters to end users?	Low execution time: low individual query latency.	Maximizing jobs executed over some time period: high throughput.
Submission Pattern: How is the work submitted to ADB?	By users manually. Either executing Notebook queries or exploring data in a connected BI tool.	Automatically submitted by a scheduler or external workflow tool without user input.
Cost: Are the workload's demands predictable?	No. Understanding data via interactive exploration requires multitude of queries impossible to predict ahead of time.	Yes, because a Job's logic is fixed and doesn't change with each run.

1. **Minimizing Cost:** By forcing users to share an autoscaling cluster you have configured with maximum node count, rather than say, asking them to create a new one for their use each time they log in, you can control the total cost easily. The max cost of shared cluster can be calculated by assuming it is running X hours at maximum size with the particular VMs. It is difficult to achieve this if each user is given free reign over creating clusters of arbitrary size and VMs.

2. **Optimizing for Latency:** Only High Concurrency clusters have features which allow queries from different users share cluster resources in a fair, secure manner. HC clusters come with Query Watchdog, a process which keeps disruptive queries in check by automatically pre-empting rogue queries, limiting the maximum size of output rows returned, etc.

3. **Security:** Table Access control feature is only available in High Concurrency mode and needs to be turned on so that users can limit access to their database objects (tables, views, functions, etc.) created on the shared cluster. In case of ADLS, we recommend restricting access using the AAD Credential Passthrough feature instead of Table Access Controls.

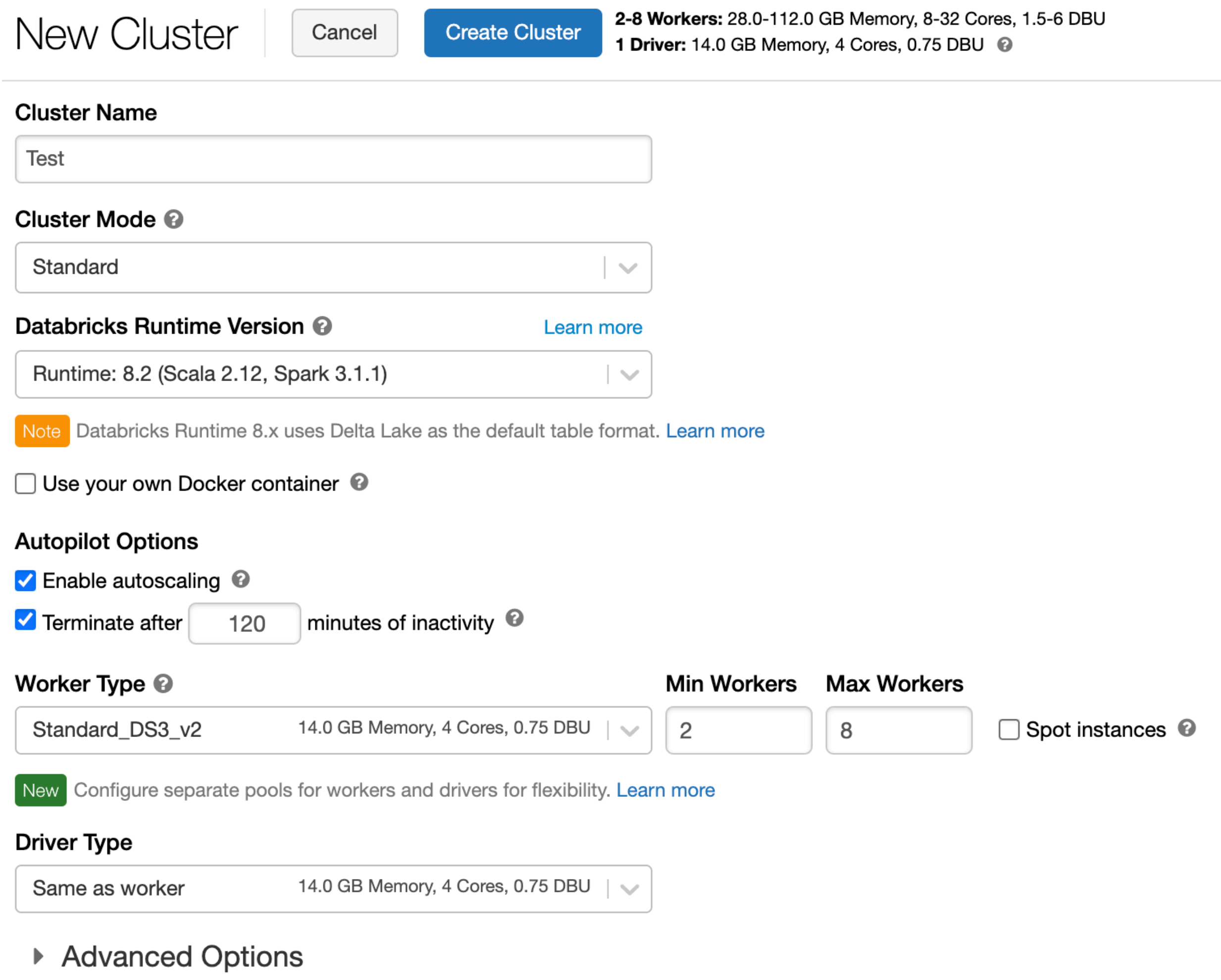
19.Arrive at Correct Cluster Size by Iterative Performance Testing

It is impossible to predict the correct cluster size without developing the application because Spark and Azure Databricks use numerous techniques to improve cluster utilization. The broad approach you should follow for sizing is:

1. Develop on a medium sized cluster of 2-8 nodes, with VMs matched to workload class as explained earlier.
2. After meeting functional requirements, run end to end test on larger representative data while measuring CPU, memory and I/O used by the cluster at an aggregate level.
3. Optimize cluster to remove bottlenecks found in step 2
    - **CPU bound**: add more cores by adding more nodes
    - **Network bound**: use fewer, bigger SSD backed machines to reduce network size and improve remote read performance
    - **Disk I/O bound**: if jobs are spilling to disk, use VMs with more memory.

Repeat steps 2 and 3 by adding nodes and/or evaluating different VMs until all obvious bottlenecks have been addressed.

Performing these steps will help you to arrive at a baseline cluster size which can meet SLA on a subset of data. In theory, Spark jobs, like jobs on other Data Intensive frameworks (Hadoop) exhibit linear scaling. For example, if it takes 5 nodes to meet SLA on a 100TB dataset, and the production data is around 1PB, then prod cluster is likely going to be around 50 nodes in size. You can use this back of the envelope calculation as a first guess to do capacity planning. However, there are scenarios where Spark jobs don’t scale linearly. In some cases this is due to large amounts of shuffle adding an exponential synchronization cost (explained next), but there could be other reasons as well. Hence, to refine the first estimate and arrive at a more accurate node count we recommend repeating this process 3-4 times on increasingly larger data set sizes, say 5%, 10%, 15%, 30%, etc. The overall accuracy of the process depends on how closely the test data matches the live workload both in type and size.



Different Azure VM instance types

Compute Optimized	Memory Optimized	Storage Optimized	General Purpose
FS	DSv2	L	DSv2
H	ESv3		DSv3

Azure VM instance type information

Type	Processor	Ram	SSD Storage
FS	Haswell (Skylake not currently supported)	1 core ~2GB	1 core ~16GB
H	High-Performance	1 core ~7GB	1 core ~125GB
DSv2 (Memory Optimized)	Haswell	1 core ~7GB	1 core ~14GB
ESv3	High-performance (Broadwell)	1 core ~8GB	1 core ~16GB
L		1 core ~8GB	1 core ~170GB
DSv2 (General Purpose)		1 core ~3.5GB	1 core ~7GB
DSv3		1 core ~4GB	1 core ~8GB

- Fewer big instances > more small instances
    - Reduce network shuffle; Databricks has 1 executor / machine
    - Applies to batch ETL mainly (for streaming, one could start with smaller instances depending on complexity of transformation)
    - Not set in stone, and reverse would make sense in many cases - so sizing exercise matters
- Size based on the number of tasks initially, tweak later
    - Run the job with a small cluster to get idea of # of tasks (use 2-3x tasks per core for base sizing)
- Choose based on workload (Probably start with F-series or DSv2):
    - ETL with full file scans and no data reuse - F / DSv2
    - ML workload with data caching - DSv2 / F
    - Data Analysis - L
    - Streaming - F

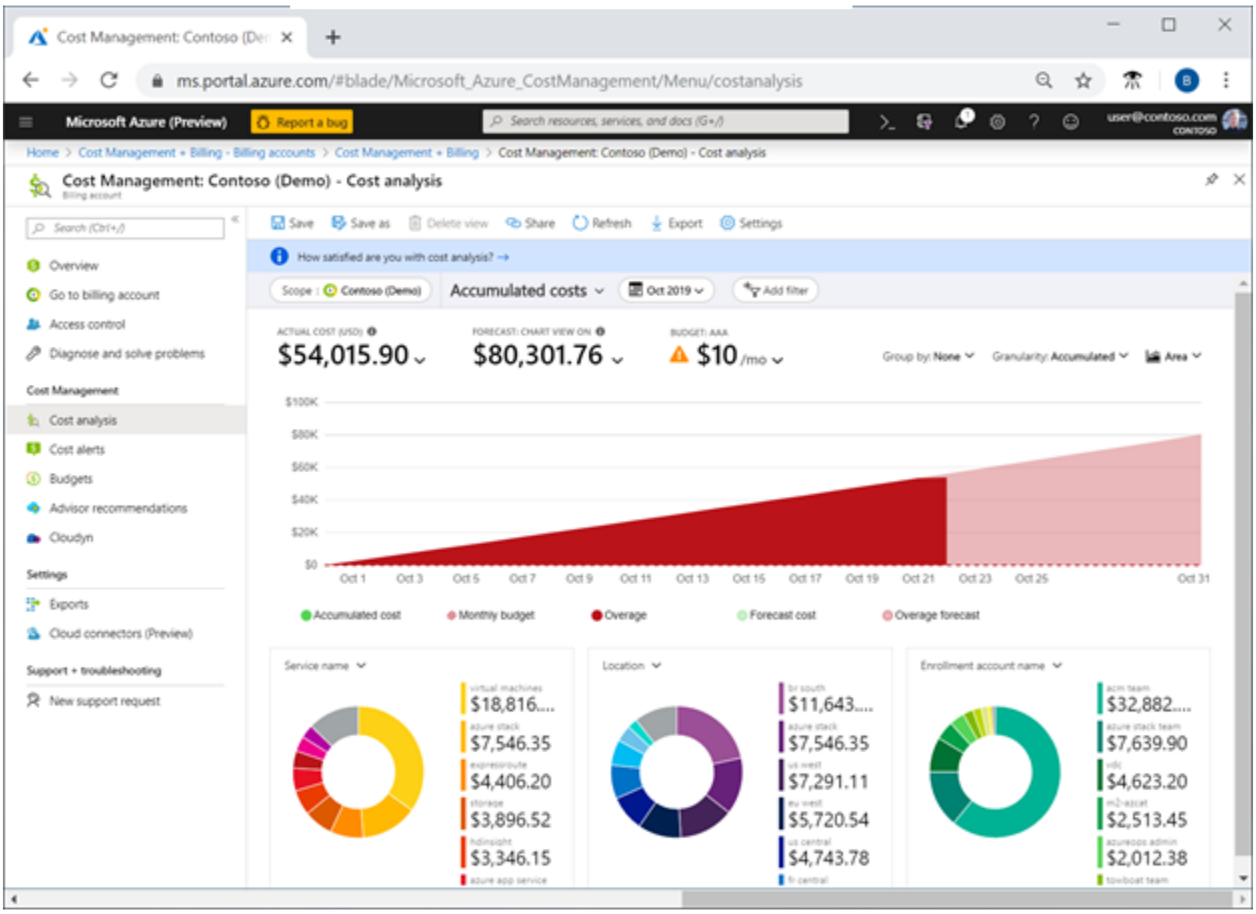
20. **Specify distribution when publishing data to Azure Data Warehouse (ADW)**

Use hash distribution for fact tables or large tables, round robin for dimensional tables, replicated for small dimensional tables. Example:
df.write \

```
.format("com.databricks.spark.sqldw") \
.option("url", "jdbc:sqlserver://
") \
.option("forwardSparkAzureStorageCredentials", "true") \
.option("dbTable", "my_table_in_dw_copy") \
.option("tableOptions", "table_options") \
.save()
```

21. Understand Databricks Pricing on individual like Compute,storage,VM and bandwidth.

Service or Resource	Pricing
DBUs	DBU pricing
VMs	VM pricing
Public IP Addresses	Public IP Addresses pricing
Blob Storage	Blob Storage pricing
Managed Disk	Managed Disk pricing
Bandwidth	Bandwidth pricing



Example 1: If you run Premium tier cluster for 100 hours in East US 2 with 10 DS13v2 instances, the billing would be the following for All-purpose Compute:

- VM cost for 10 DS13v2 instances —100 hours x 10 instances x $0.598/hour = $598
- DBU cost for All-purpose Compute workload for 10 DS13v2 instances —100 hours x 10 instances x 2 DBU per node x $0.55/DBU = $1,100
- The total cost would therefore be $598 (VM Cost) + $1,100 (DBU Cost) = $1,698.

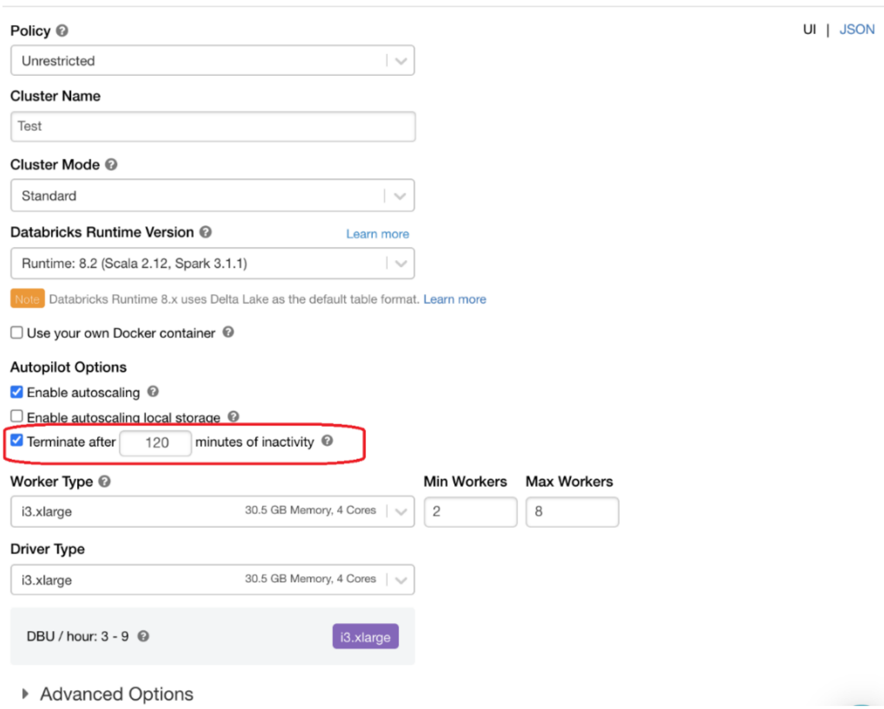
Example 2: If you run Premium tier cluster for 100 hours in East US 2 with 10 DS13v2 instances, the billing would be the following for Jobs Compute workload:

- VM cost for 10 DS13v2 instances —100 hours x 10 instances x $0.598/hour = $598
- DBU cost for Jobs Compute workload for 10 DS13v2 instances —100 hours x 10 instances x 2 DBU per node x $0.30/DBU = $600
- The total cost would therefore be $598 (VM Cost) + $600 (DBU Cost) = $1,198.

In addition to VM and DBU charges, there will be additional charges for managed disks, public IP address, bandwidth, or any other resource such as Azure Storage, Azure Cosmos DB depending on your application.

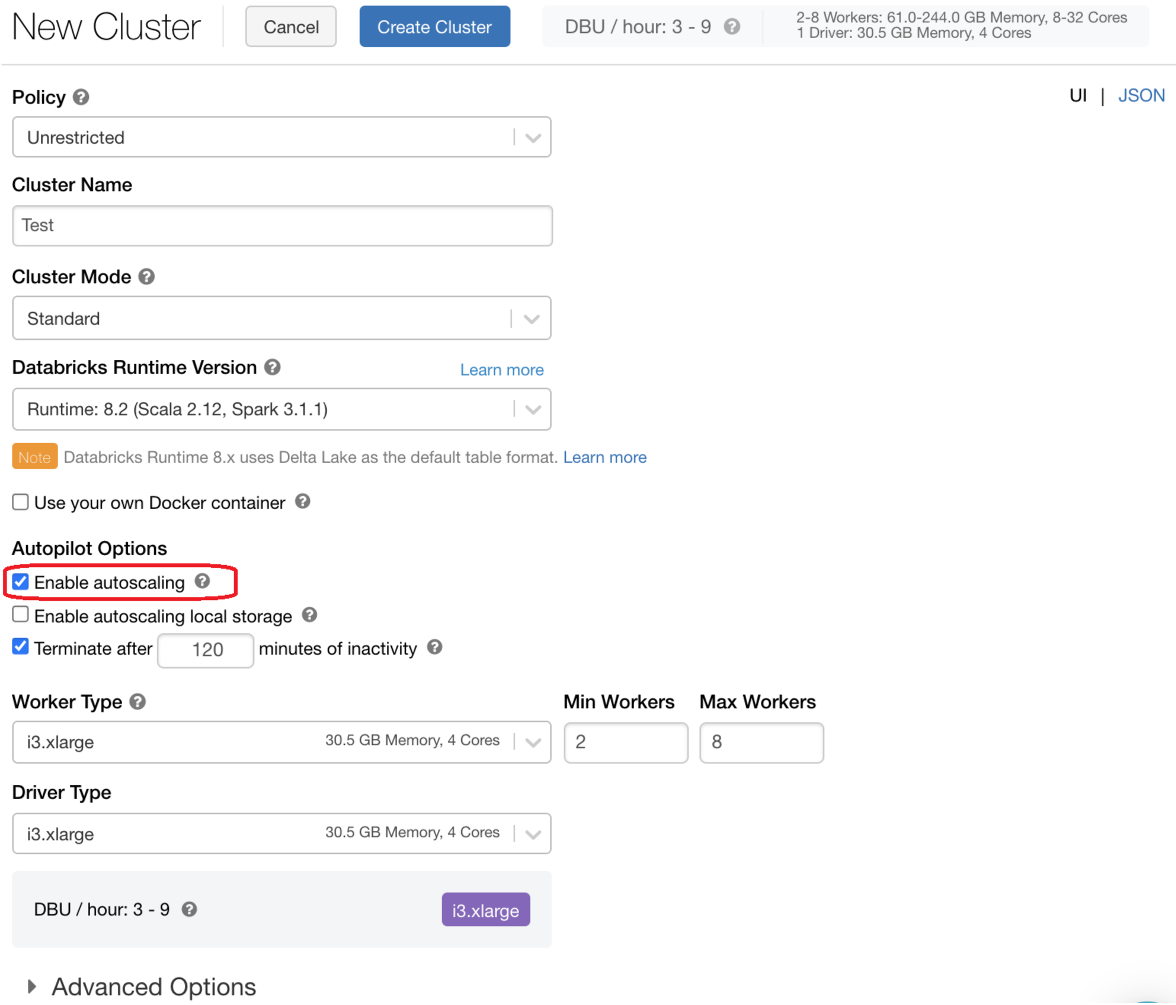
22. Customize cluster termination time

Terminating inactive clusters saves costs. ADB automatically terminates clusters based on a default down time. As different projects have different needs, it's important to customize the down time to avoid premature or delayed termination. For example: set a longer downtime for development environments, as work is continuous



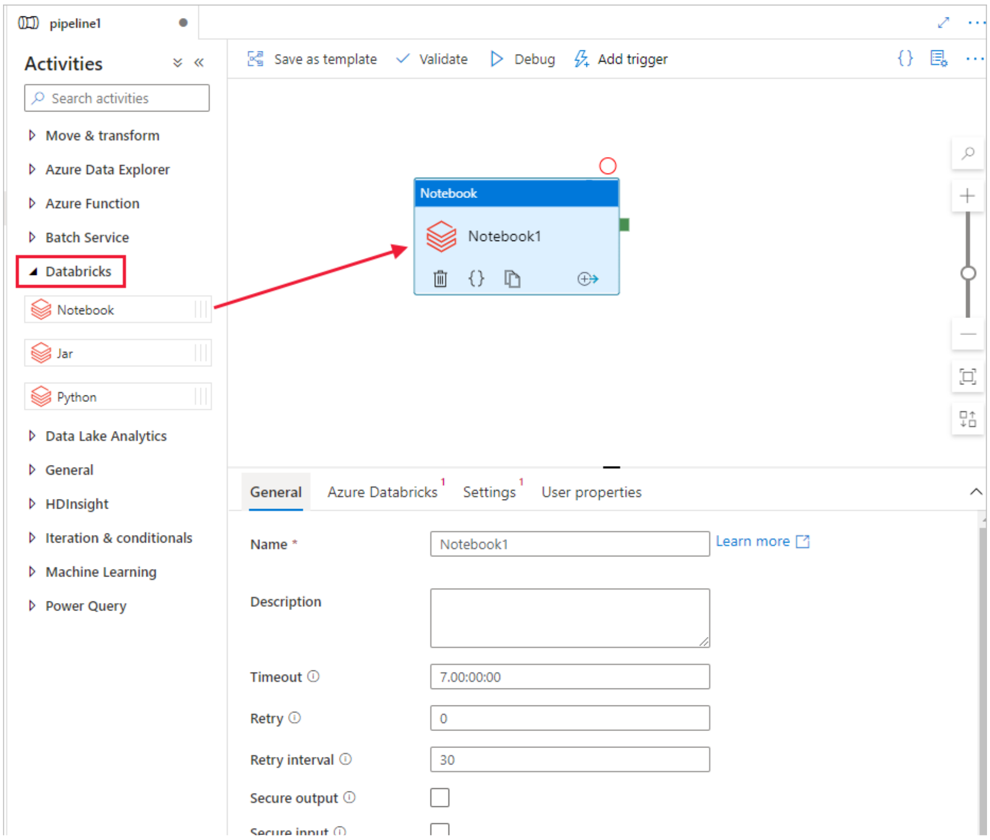
23. Enable cluster autoscaling

ADB offers cluster autoscaling, which is disabled by default. Enable this feature to enhance job performance. Instead of providing a fixed number of worker nodes during cluster creation, you should provide a minimum and maximum. ADB then automatically reallocates the worker nodes based on job characteristics.



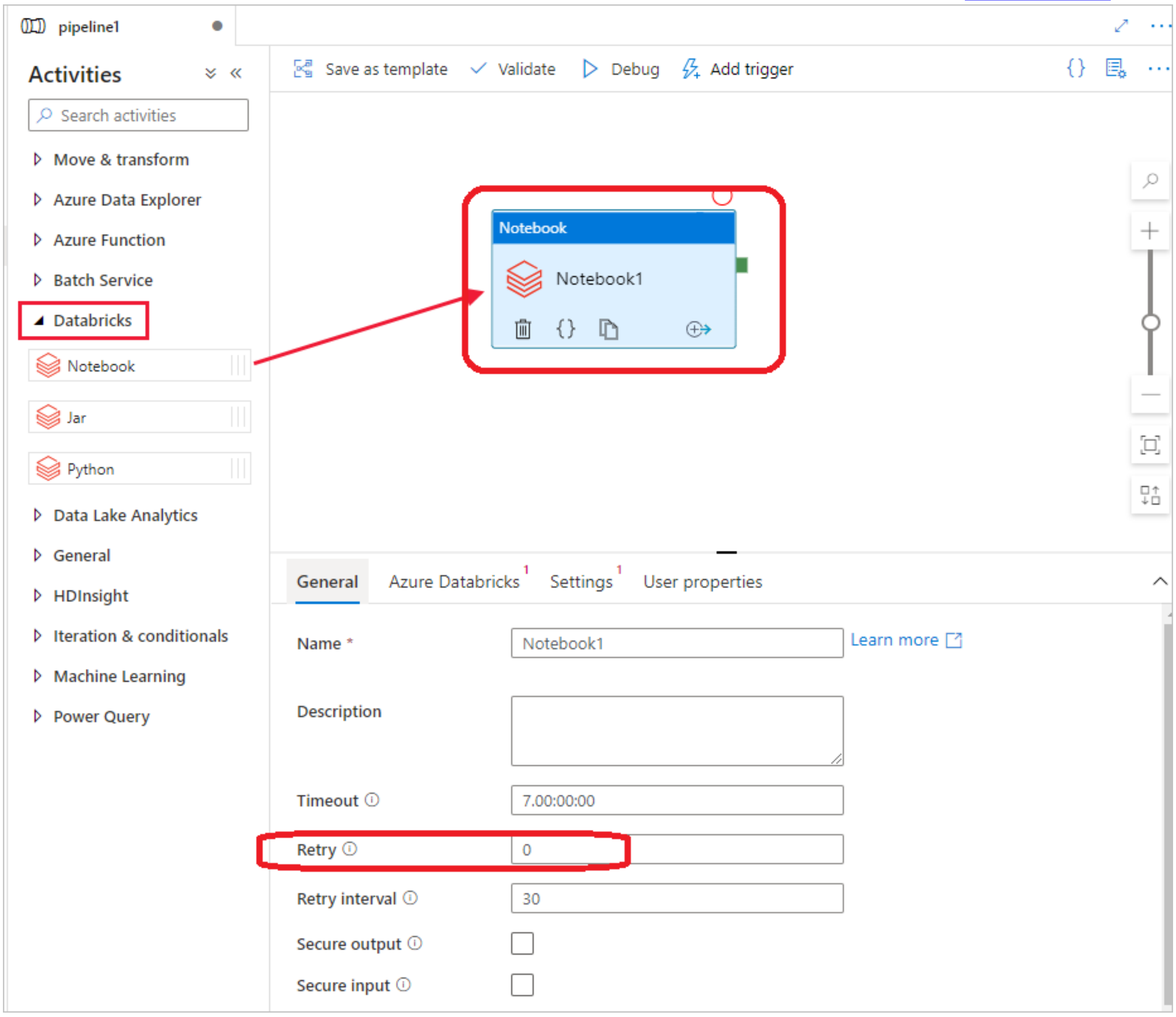
24. Use Azure Data Factory (ADF) to run ADB notebook jobs

If you run numerous notebooks daily, the ADB job scheduler will not be efficient. The ADB job scheduler cannot set notebook dependency, so you would have to store all notebooks in one master, which is difficult to debug. Instead, schedule jobs through Azure Data Factory, which enables you to set dependency and easily debug if anything fails.



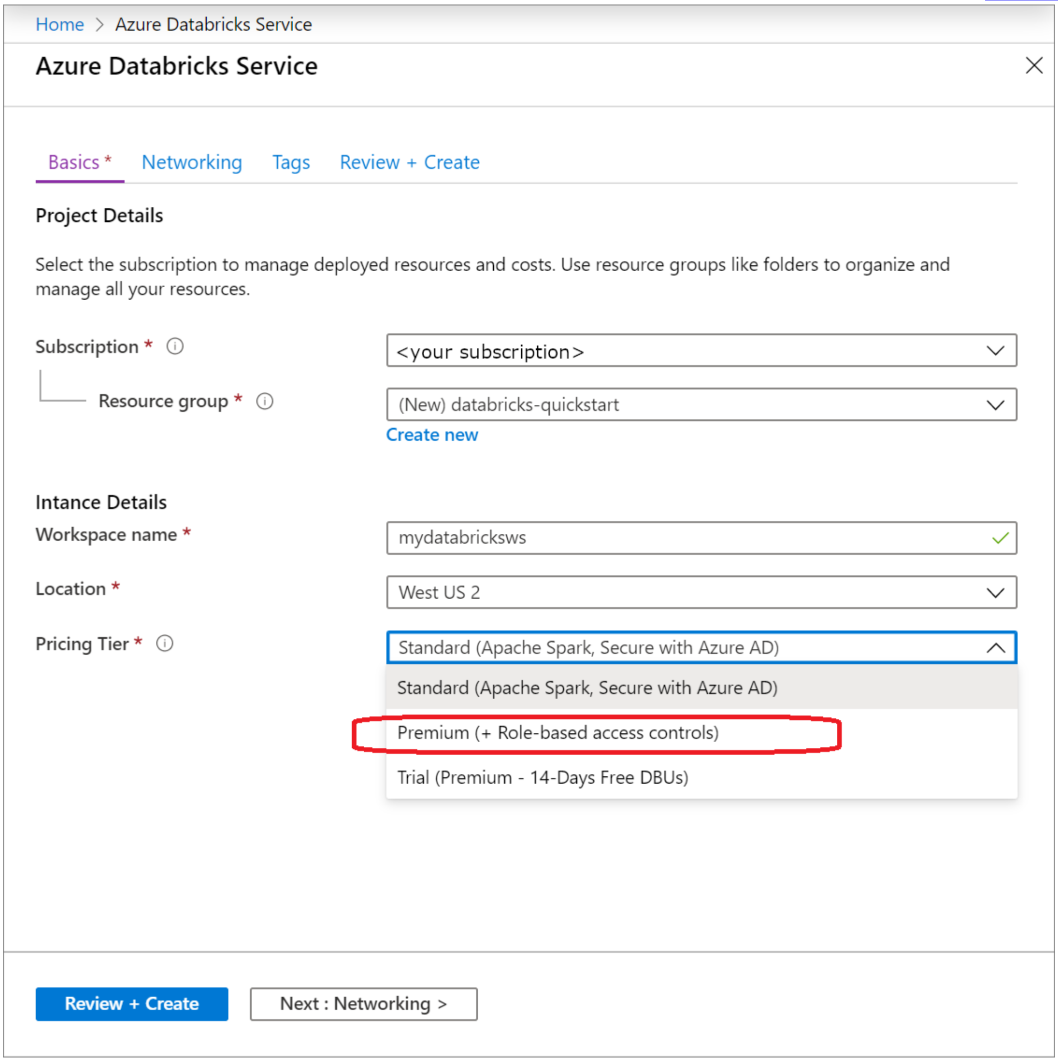
25. Use the retry feature in ADF when scheduling jobs

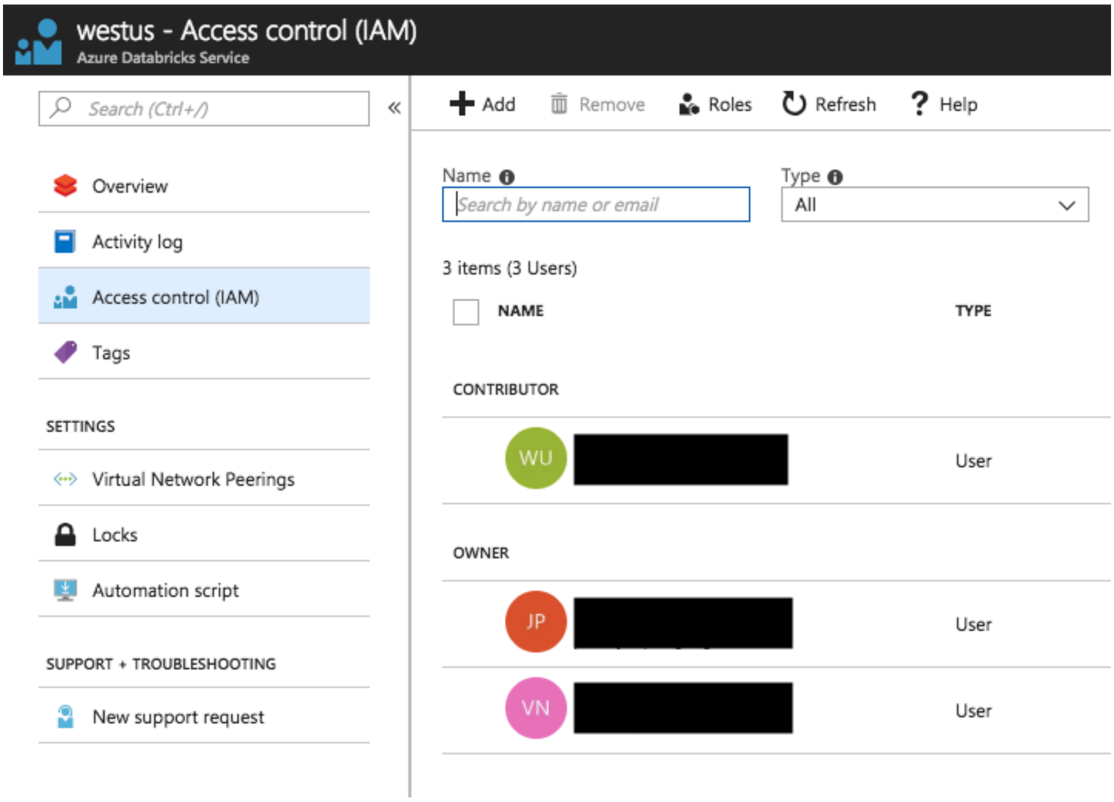
Processing notebooks in ADB through ADF can overload the cluster, causing notebooks to fail. If failure occurs, the entire job should not stop. To continue work from the point of failure, set ADF to retry two to three times with five-minute intervals. As a result, the processing should continue from the set time, saving you time and effort.

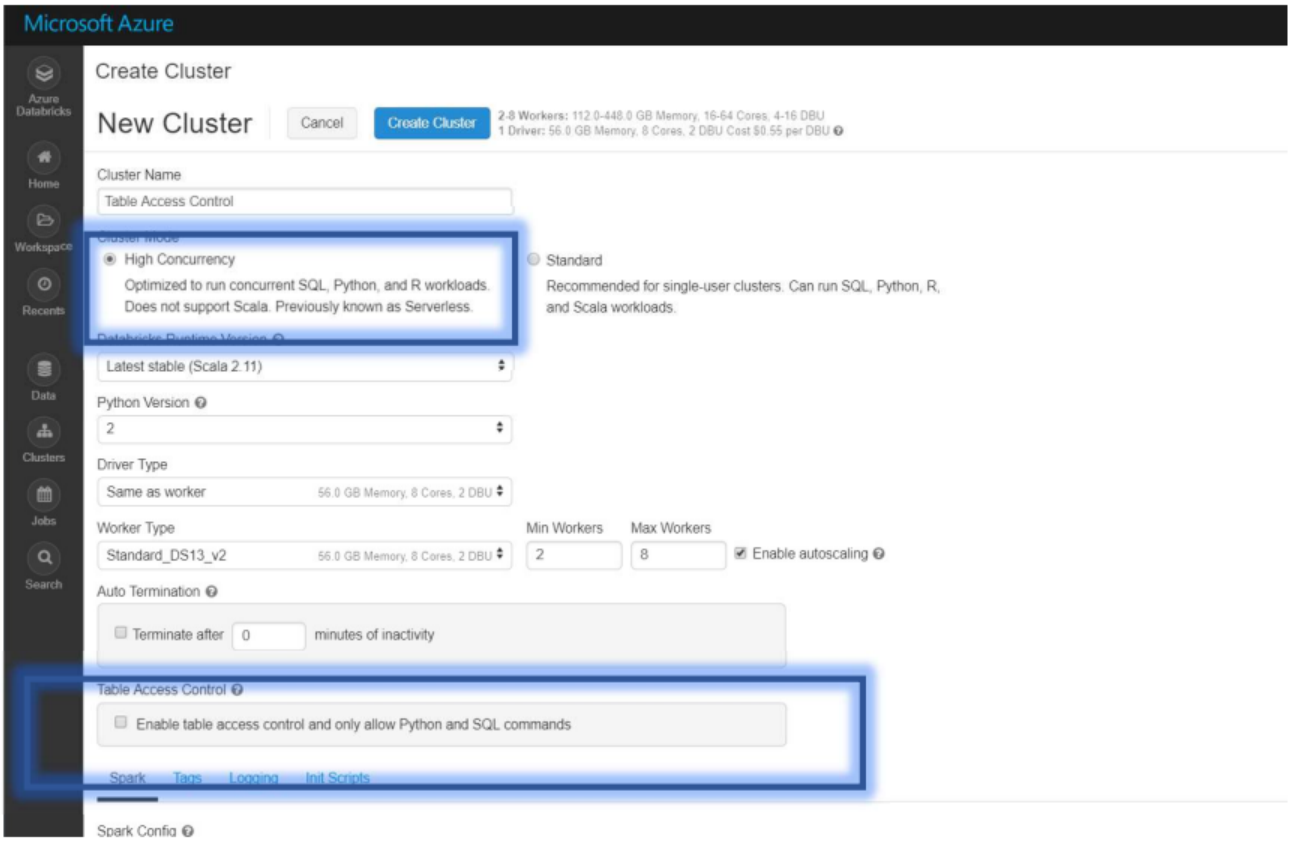


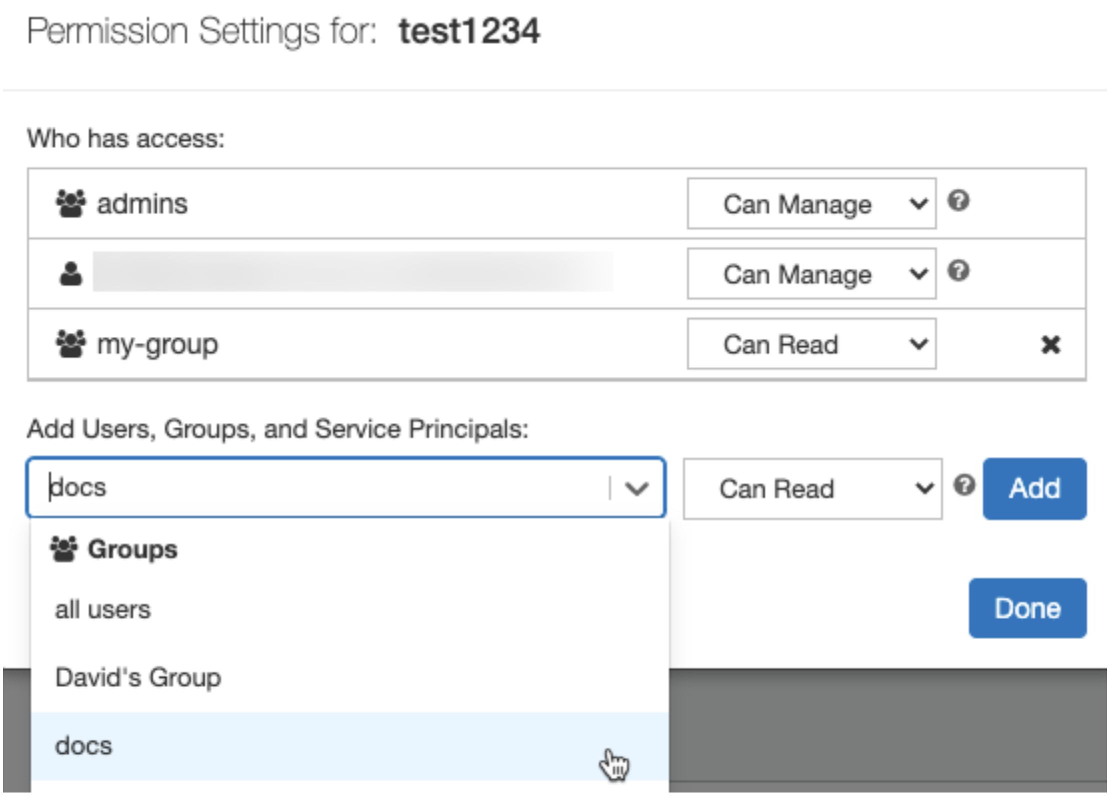
26. Consider upgrading to ADB Premium

Your business's data has never been more valuable. Additional security is a worthwhile investment. ADB Premium includes 5-level access control.


Home > Azure Databricks Service
Azure Databricks Service
Basics *
Networking
Tags
Review + Create
Project Details
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.
Subscription *
<your subscription>
Resource group *
(New) databricks-quickstart
Create new
Intance Details
Workspace name *
mydatabricksws
Location *
West US 2
Pricing Tier *
Standard (Apache Spark, Secure with Azure AD)
Standard (Apache Spark, Secure with Azure AD)
Premium (+ Role-based access controls)
Trial (Premium - 14-Days Free DBUs)
Review + Create
Next : Networking >


westus - Access control (IAM)
Azure Databricks Service
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
SETTINGS
Virtual Network Peerings
Locks
Automation script
SUPPORT + TROUBLESHOOTING
New support request
Add
Remove
Roles
Refresh
Help
Name
Search by name or email
Type
All
3 items (3 Users)
NAME
TYPE
CONTRIBUTOR
WU
[redacted]
User
OWNER
JP
[redacted]
User
VN
[redacted]
User


Microsoft Azure
Azure Databricks
Home
Workspace
Recents
Data
Clusters
Jobs
Search
Create Cluster
New Cluster
Cancel
Create Cluster
2-8 Workers: 112.0-448.0 GB Memory, 16-64 Cores, 4-16 DBU
1 Driver: 56.0 GB Memory, 8 Cores, 2 DBU Cost $0.55 per DBU
Cluster Name
Table Access Control
High Concurrency
Optimized to run concurrent SQL, Python, and R workloads. Does not support Scala. Previously known as Serverless.
Standard
Recommended for single-user clusters. Can run SQL, Python, R, and Scala workloads.
Latest stable (Scala 2.11)
Python Version
2
Driver Type
Same as worker
56.0 GB Memory, 8 Cores, 2 DBU
Worker Type
Standard_DS13_v2
56.0 GB Memory, 8 Cores, 2 DBU
Min Workers
2
Max Workers
8
Enable autoscaling
Auto Termination
Terminate after
0
minutes of inactivity
Table Access Control
Enable table access control and only allow Python and SQL commands
Spark
Tags
Logging
Init Scripts
Spark Config


Permission Settings for: test1234
Who has access:
admins
Can Manage
Can Manage
my-group
Can Read
Add Users, Groups, and Service Principals:
docs
Can Read
Add
Groups
all users
David's Group
docs
Done

Ability	No Permissions	Read	Run	Edit	Manage
View items		x	x	x	x
Create, clone, import, export items		x	x	x	x
Run commands on notebooks			x	x	x
Attach/detach notebooks			x	x	x
Delete items				x	x
Move/rename items				x	x
Change permissions					x