Databricks vs. Snowflake Analysis
A Case Study in Managed Service vs. Software as a Service

Summary

We evaluated Snowflake vs. Databricks for a common use case similar and nine (9) functional areas.

Performance

For the performance metrics, we see that with a bit of basic tuning on the Databricks side, things are slightly more performant with Snowflake. This is normal practice. It is very common to see Snowflake OOTB 2-4x faster than Databricks and then with a bit of tuning Databricks gets close to where Snowflake is.

Performance Item	Snowflake	Databricks	Efficiency Gained
Common Use Case	Execution = 1min 8 sec ($.065)	Xlarge = 6 min 2 sec ($.077)	18%
		Xlarge2 = 3 min 21 sec ($.086)	33%

Functional

However, for anyone who has run any decent sized data and analytics organization, ***HW/SW/Cloud costs are the minority of your budget.*** You must consider the Level of Effort (LoE) impact to your organization.

Having personally ran a 250 person Fortune-100 Data and Analytics shop and having spent the last seven (7) years as Hadoop Systems Integrators (and the last 3 years as the top hadoop SI in North America), we know this very well and we hear this all the time - *"**but Snowflake is so much easi**er".*

So, let's look at costs by functional areas, let's look at the where:

Role	Role Item	Snowflake Hours	Snowflake $'s	Databricks Hours	Databricks $'s	$'s gained	Efficiency Gained	Value
1	Func – Software Mgmt	160	$12,800	1,000	$80,000		84.0%	Low
	Func – Cluster Mgmt	240	$19,200	2,400	$192,000		90.0%	
	Func – Environment Mgmt	320	$38,400	4,000	$480,000		92.0%	
	Platform Admin Role Total	**720**	**$70,400**	**7,400**	**$752,000**	**$681,600**	**90.6%**	
2	Func – Security Mgmt	2,500	$200,000	4,500	$360,000		44.4%	Medium
	Func – Job Optimization	160	$12,800	2,500	$200,000		93.6%	
	Func – Actual ETL Dev	16,150	$1,938,000	17,000	$2,040,000		5.0%	
	Data Engineer Role Total	**18,810**	**$2,150,800**	**24,000**	**$2,600,000**	**$449,200**	**17.3%**	
3	Func – Reporting / BI	14,560	$1,164,800	16,000	$1,280,000		9.0%	High
	Func – Analysts/Power Users	22,500	$1,800,000	30,000	$2,400,000		25.0%	
	Func – AI / ML	9,600	$1,152,000	8,000	$960,000		-20.0%	
	Citizen Role Total	**46,660**	**$4,116,800**	**54,000**	**$4,640,000**	**$523,200**	**11.3%**	
Total		**66.190**	**$6,338,000**	**85,400**	**$7,992,000**	**$1,654,000**	**20.7%**	

The total difference in the above mid-sized organization is 33 heads vs. 43 heads.

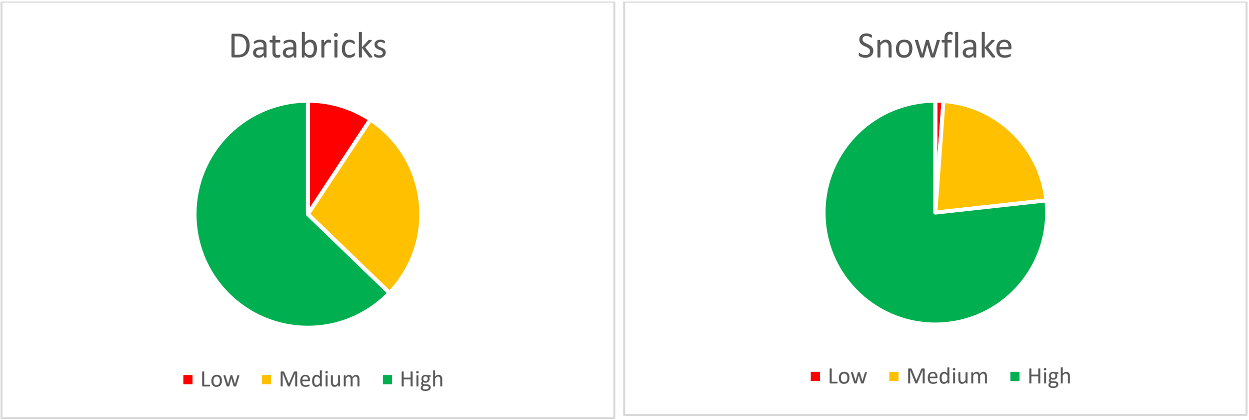
The obvious difference is in the platform administrator and the job optimization. Snowflake there is no platform administrator and no job optimization. But there are savings beyond these roles. Databricks still wins in the traditional Machine Learning space (though that may change with Containerized Services..TBD).

In all estimates, we use $80/hr (50% onshore, 50% offshore), except the data science we use $120/hr. The numbers shown are annual costs. Please note: **There will be variance in your organizational savings by moving to Snowflake instead of Databricks. Not all companies will have the same savings, but all companies will have savings – and it will be significant.**

Resource Allocation:

We do not recommend removing the 10 heads – good people are hard to find. We recommend placing those folks into the higher value work (delivering answers to the business). Here is how the work delivery of low (platform admin roles), medium (data engineer roles) and high (citizen roles) differs between Databricks and Snowflake for a mid-sized organization:

Resource Deployment by Value



Functional – Software Management

Snowflake

Snowflake is a managed service that handles software management, maintenance, upgrades, and tuning. Snowflake is considered a serverless offering, meaning users don't install, configure, or manage software. Snowpark does have library management that needs to be considered.

a. **Snowflake Data Cloud** – nothing, all software is fully managed.
b. **Snowpipe** – nothing, all software is fully managed.
c. **Snowpark** – all software is fully managed, but developers' groups that utilize python, scala or java libraries require maintenance of those libraries.

Databricks

The Databricks platform consists of clusters, workspaces and catalogs, among other things. Jobs run on clusters, which are tied to a specific version of Databricks. This runtime version dictates the python, scala, spark and R versions that the users' jobs will run on.

Software management in Databricks involves ensuring the dependencies, libraries, and packages in the runtime cluster meet the needs and currency required for users' notebooks and jobs. Individual users may include notebook libraries, init scripts and environment variables where the runtime libraries are not sufficient.

A typical implementation will have many clusters which may span several versions of Databricks runtime. Databricks releases an LTS version every 6 months, and they are good for 3 years.

Upgrading Databricks is a periodic process the platform administrator must perform. The majority of effort is ensuring that your legacy code will run on the new release. Please see the Databricks documentation for the how-to: https://docs.databricks.com/en/release-notes/runtime/12.x-migration.html.

Here is a very High-level overview of how to perform:

a. Review system environment properties, features and libraries in the new release
b. Review which features and libraries you currently use
c. Ensure there is no conflict with any user defined libraries, packages and dependencies with the target Databricks runtime version.
d. Create a new cluster with the new version.
e. Repoint the desired workspaces to use the newly versioned cluster.
f. Test to ensure equivalency.
g. Perform the same in the next higher environment (e.g. test, production, dr, etc.).

Small organizations will have 4-5 clusters to keep software currency on, medium sized organizations will have 15-20 clusters and large organizations will have 40-50 clusters. We saw one organization that had 129 databricks clusters (and each had its own hive metastore)! Currency becomes exponential challenging with the number of clusters to manage.

For Snowflake, software currency involves reviewing release notes for new features.

For Databricks, software currency involves reviewing release notes for new features and upgrading the software.

Annual currency cost:

Org Size	Snowflake Hrs	Snowflake $'s	Databricks Hours	Databricks $'s
Small	80	$6,400	400	$32,000
Medium	160	$12,800	1,000	$80,000
Large	320	$25,600	2,500	$200,000

Functional – Cluster Management

Snowflake

Snowflake's approach to cluster management is automated and abstracted, simplifying the process and allowing users to focus on queries and analytics without worrying about cluster configuration.

a. **Cluster Setup**

   No need to create, configure, or manage clusters manually. Simply pick T-shirt size warehouse.

b. **Cluster Scaling**
   When configuring warehouse, autoscaling can be enabled which will dynamically adjust the compute resources of a virtual warehouse based on the workload demand.

Databricks

Databricks allows for more customization but also requires more manual configuration for creating and auto scaling of the clusters.

a. **Cluster Setup**
   Platform administrators are responsible for configuring and managing their own clusters within the Databricks environment. This includes specifying the runtime version, instance types, and cluster configurations. Databricks also allows platform administrators to customize cluster configurations. Following are the specific steps.

   I. Create Cluster (Specify cluster name, Databricks runtime environment and cluster mode
   II. Configure node type (Specify CPU and memory configurations)
   III. Customize the configuration (Set spark configuration for specific use case and optionally enable the Spark UI and Jupiter notebooks to facilitate monitoring and interactive analysis.)

b. **Cluster Scaling**
   Platform administrators must select the node types and the number of scale-out nodes for the cluster. Databricks has two types of autoscaling: cluster autoscaling and enhanced autoscaling. Cluster autoscaling scales the cluster based on the number of pending tasks, while enhanced autoscaling scales the cluster based on the CPU and memory utilization.

c. **Instance / Obj Store Management**

   Databricks creates instances in your space and terminates those after use (although the lowest auto termination with DBX 12 is 10 minutes). Additional work is required to allow Databricks to create instances (IAM). Databricks administrators are continually working to determine the ec2 / azure instance spend that is occurring behind the scenes to ensure the costs are kept under control.

One caveat is the Databricks serverless offering that came out May 2023. This offering provides serverless (managed services style) compute for SQL and MLFlow (the eval part of ML). This will be the minority of Databricks clusters in practice. This does lower the cluster management cost of Databricks.

Snowflake: As a managed service, there is no cluster setup or management. The administrator periodically evaluates how many warehouse are available for the users, along with their warehouse spend to ensure it is a good set.

Databricks: The platform administrator creates and destroys clusters with various configuration sets for users to attach to. The administrator also periodically evaluates DBU and instance spend. Instance/IAM/Terraform labor costs to organizations are higher. Organizations with a greater variety of cluster types will see a greater variance in the amount of additional resources needed to manage.

Annual Cluster Management Cost:

Org Size	Snowflake Hrs	Snowflake $'s	Databricks Hours	Databricks $'s
Small	120	$9,600	1,000	$80,000
Medium	240	$19,200	2,400	$192,000
Large	320	$25,600	5,200	$416,000

Functional – Environment Management

Snowflake

a. **Creation of Disaster Recovery**
   Cross Region Replication in Snowflake makes creating and replicating data in DR environments a breeze.

b. **Creation of Test and Development Environments**
   Using Zero Copy Cloning Snowflake can clone database/table seamlessly into other environments.

c. **Metadata Management**
   Snowflake automatically creates metadata for data residing both externally (S3, Azure, GCP) and internally (within Snowflake), stores it as a key-value pair (dictionary), and makes it available via the Information Schema

Databricks

a. **Creation of Disaster Recovery**

Databricks requires more steps after setting up cross region environment in CSP

   i. Update your Databricks notebooks and jobs to the new storage paths in the replicated region.
   ii. Migrate the existing data.
      a. Update the connection strings or access keys to access new replicated storage.

b. Creation of Test and Development Environments

Databricks requires manual steps to create additional environments from production environment.

   i. Create a DB workspace.
   ii. Import notebooks.
   iii. Update configuration parameters.
   iv. Configure libraries for data processing.
   v. Replicate data.
   vi. Replicate metadata.

c. **Metadata Management**

Requires enablement of Unity Catalog for the metadata management. Unity Catalog is available in updated versions of Databricks. Data with the Unity Catalog (i.e. the metadata) must be exported and imported into the separate environment.

With Snowflake, the process of creating environments and ensuring DR is complete is simpler by an order of management with Snowflake due to Zero Copy Clone.

Annual Environment Management Costs:

Org Size	Snowflake Hrs	Snowflake $'s	Databricks Hours	Databricks $'s
Small	160	$19,200	1,500	$180,000
Medium	320	$38,400	4,000	$480,000
Large	500	$60,000	10,000	$1,200,000

Functional – Security Management

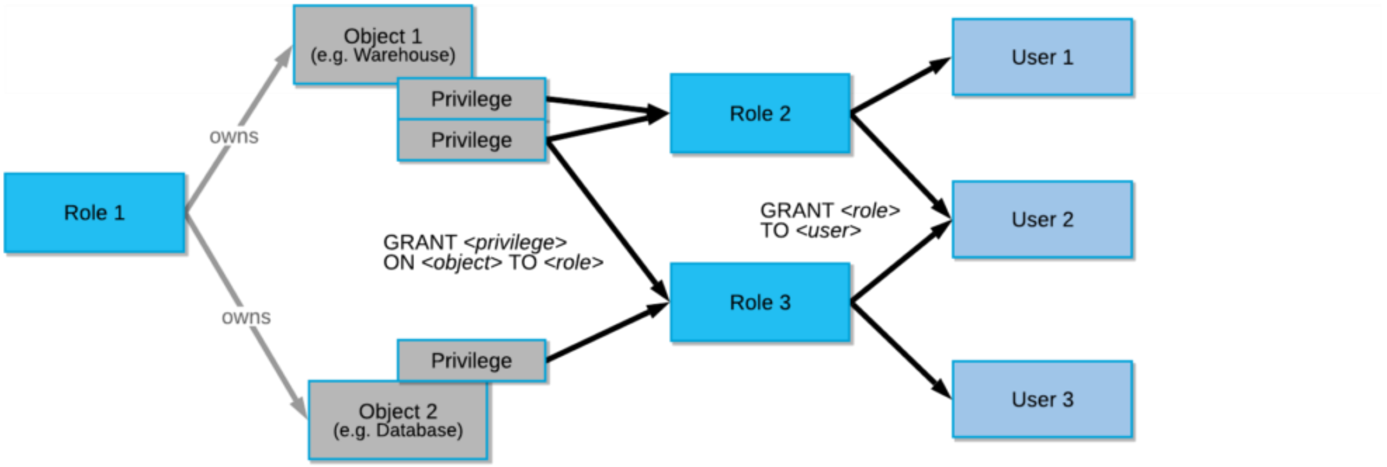
Security Management is an area where the two platforms do not offer the same set of capabilities. Snowflake was built with security from the ground up. Databricks has added security capabilities over time.

The capabilities are as of November 2023.

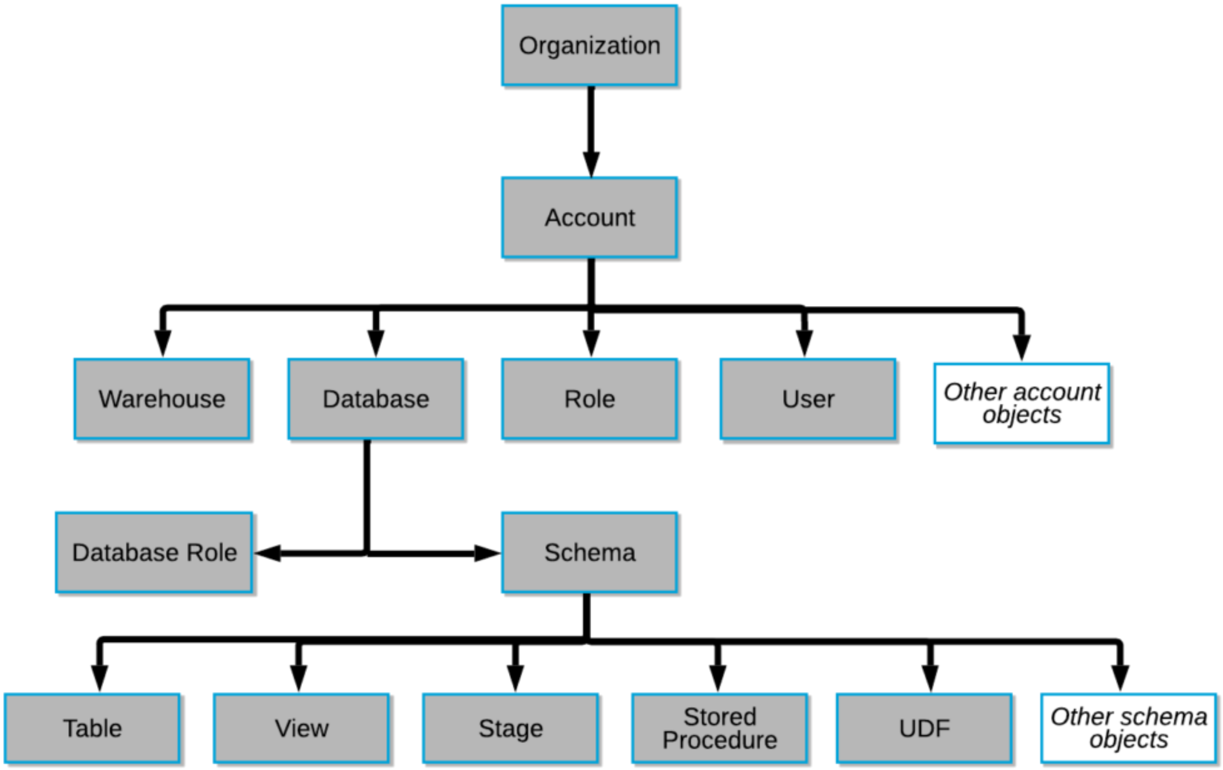
This role could go in either platform admin or data engineering; however, since security must be adhered strictly by the data engineering on the Databricks side, we have placed this with Data Engineering.

How Snowflake does Policy Management

With Snowflake you create policies based on resources or data classification. Here is a handy url to quickly get you spun up: Access control Overview. Within this, you see the traditional GRANT that allows privileges to be granted to roles (which are in turn other roles, or users) for a given resource.



Resources can be any tables, view, udf, etc.:



Snowflake also took it a step further, and they allow data to be classified and tagged, with policies created on the data classification (at the column level) that allows that data to be allowed/denied/masked. The column level distinction is extremely important – it is very common for PII data to mixed in tables with non-PII data. They also have a row level filtering capability. This is very important as well, as you often find yourselves not wanting to grant access to all users for all rows on a given fact table.

Here is the data governance primer url to quickly get you spun up: Data Governance Primer

There are several key points here:

1. Data Classification (url here), the classifier, which is called by function EXTERNAL_SEMANTIC_CATEGORIES , derives from the data columns which may be PII data. It is driven by a probabilistic algorithm that looks for the existence of such data.

2. Object Tagging (url here) – you the data engineer tag the data by type. You create your own tags and you may have multiple sensitive types (CPNI, HIPAA, PCI, PII, etc.) depending on your business and you may also want to have non-sensitive types. Tagging allows you to search and set policy based on data classification type.

3. Set your column and row level policies (url here). Now that we have the data tagged, we can set policies based on those tags. I can say for all my CPNI data, only allow access for users with CPNI role.

How Databricks does Policy Management

In Databricks SQL, users are granted access to a resource using the grant command (url here).

The syntax is:

```
GRANT privilege_types ON securable_object TO principal;
```

For example:

```
GRANT CREATE ON SCHEMA myschema TO `alf@melmak.et`;
GRANT ALL PRIVILEGES ON TABLE forecasts TO finance;
GRANT SELECT ON TABLE sample_data TO USERS;
```

This is not much different than Snowflake resource based granting, the exception being with Databricks you are granting to users and groups and not to roles. Roles do give you more flexibility, but the bigger gap is the lack of column masking and row filtering, which is noticeably absent..

The Gap

Here is a list of the native capabilities of each platform:

Item	Snowflake	Databricks
Principals can be users or groups	✓	✓
AD and MFA Integration	✓	✓
Resource Based Policies	✓	✓
Tagged Based Policies	✓	X
Column Level Masking*	✓	X
Row Level Filtering	✓	X

* In November 2023, Databricks went public preview with a masked column function. (https://docs.databricks.com/en/sql/language-manual/sql-ref-syntax-ddl-column-mask.html). We have not tested this new feature, though it looks like you have to call the function for each masked column you wish to create and make sure it is used in the create table function (not ideal, manual, extra work).

With Databricks, users must employ workarounds. Below are the examples of work-arounds necessary:

Workaround #1 – Column Masking

There are two options available to Databricks users to hide sensitive columns. If you are a Databricks Data Engineer, make sure you use this for all sensitive data (PII, CPNI, PCI, HIPAA, etc.).

1. UDF – create a user defined function (https://www.databricks.com/blog/2020/11/20/enforcing-column-level-encryption-and-avoiding-data-duplication-with-pii.html). Here a UDF is utilized to encrypt the data prior to inserting to table. A view with a decryption is available for privileged users to decrypt.

2. Beginning with DBR 12.2, you can create a table that stores the actual value and a view that the regular users will use. In the view, users select against a mask. Ensure users only select against the view, not the actual table.
create table customer (acctid int, fname varchar(20), lname varchar(20), ssn varchar(11));
create view customer_vw as select acctid, fname, lname, mask(ssn) from customer;

Pay attention not to miss any sensitive columns!

Workaround #2 – Row Filtering
You have to change your data model to separate the sensitive rows (and join if you want both) or use views. Here is the views way to do it.

Take the telco example. Some users can see all data. Other users should only see non-government. You have to split into two tables and users who see all will need to join the two tables or create a view, only allow users access to that view and ensure the view does the filter so government can not be seen:

Phone	AccountType	Lname	Fname	CallDate	CallTime	NumberCalled
913-555-1212	Consumer	Doe	Jane	2023-05-09	11:04:05	913-555-1215
913-555-1213	Consumer	Doe	Jim	2023-05-09	11:04:06	913-555-1215
913-555-1214	Business	Doe	John	2023-05-09	11:04:07	913-555-1215
913-555-1222	Government	Biden	Joe	2023-05-09	11:04:07	913-555-1215
913-555-1223	Government	Trump	Donald	2023-05-09	11:04:07	913-555-1215

create view as select * from cdr where accounttype != “Government”

Make sure non-privileged users can only hit the view!

Make sure all sensitive data comes in that matches where clause. E.g. Government, not government. Alternatively, catch more things in the where clause:

create view as select * from cdr where accounttype != “Government” or accounttype != “government” or accounttype != “Govt” accounttype != “govt”

Workaround #3 – No Tagging
There is no work-around here. You are going to need a 3rd party tool or create your own set of metadata tables.

Security Cost

The differentiator between the cost of Snowflake and Databricks from a security perspective is the cost of creating and maintaining the workarounds. There is a security cost (risk) to the organization that is very real, but estimating that is beyond the scope of this. We are just putting the labor cost of maintaining the workarounds. The cost varies based upon the amount of private data you have, the industry you are in, the regulations surrounding your data and what data you choose to put in your data lake. For the purposes of this document, we are just going to use an industry average of 12% of columns containing private data.

Security Management Annual Costs:

Org Size	Snowflake Hrs	Snowflake $'s	Databricks Hours	Databricks $'s
Small	1,250	$100,000	2,250	$180,000
Medium	2,500	$200,000	4,500	$360,000
Large	5,000	$400,000	9,000	$720,000

Functional – Job Management / Optimization

Snowflake is a managed service – no optimizations required.

Databricks is a SaaS offering – a number of items must be looked at to get optimal performance and price/performance.

Snowflake	Databricks
Pick the correct warehouse	1. Get the right cluster size. 2. Get the right vm's 3. Identifying broadcast join vs. sort merge join opportunities. Make sure you get autoBroadcast correct size and make sure driver memory sufficient. 4. Identifying where to switch from java serializer to kyro serializer. 5. Identifying where to apply schema upfront - everything is 20 bytes in java (including inferred schemas) 6. Identifying troublesome UDF's and converting to Dataframe Operations (usually involves a lot of code) 7. Getting the partitioning right 8. Making sure you coalesce or repartition on small datasets. 9. Generating config for schema - otherwise when schema changes your ddl within your pySpark code has to change (you either do chunk of work upfront or chunk of work going through pySpark code when schema changes). 10. Have to update stats manually. 11. Tweak executor core/mem, driver core/mem

Benchmarks have shown that a fully optimized Databricks cluster is similar in performance to an out of the box Snowflake cluster.

Squadron Data has performed multiple Spark Optimizations where we have delivered 60-70% efficiency gains for a $250k SOW in a mid-sized organization. Organizations will incur greater costs if they are educating developers and having developers perform individually for each of their jobs.

Annual Job Management / Optimization Cost:

Org Size	Snowflake Hrs	Snowflake $'s	Databricks Hours	Databricks $'s
Small	80	$6,400	1,000	$80,000
Medium	160	$12,800	2,500	$200,000
Large	320	$25,600	6,000	$480,000

Functional – ETL Development

Here we are comparing the actual ETL Development between the two use cases. We are not comparing migration of existing legacy data platforms (hadoop, Teradata, Exadata, etc.). For these use cases, Snowflake is a non-brainer – and the migration costs will be half (or less) to Snowflake than to Databricks.

Here we considered the ongoing ETL Development costs once the basic platform is stood up.

Both platforms support a wide range of file formats (parquet, delimited, json, xml, etc.) and both support a wide range of input methods (api, object store ingest, rdbms with or without cdc, noSQL etc.). Snowflake has a wider range of integrations with external systems (e.g. salesforce).

Many customers use one or more tools to perform this ETL development (e.g. dbt, Matillion). The level of effort from within these tools is the same. For validation outside the ETL tool, Snowflake is widely considered easier (sql vs. spark).

For those customers choosing to use native tools, there is not much difference. Both platforms offer a rich UI. Databricks offers this through a Jupyter-like notebook experience. Snowflake has its own UI. For those Snowflake users who prefer the notebook experience, Hex notebooks are as robust or more as Databricks and are very cheap. Snowflake also has its own notebook in private preview. There really isn’t much difference from the UI perspective.

Overall, organizations will find both platforms to be very similar from an ETL development perspective. The key differentiators are the testing validation efforts and obtaining the necessary Spark proficiency on staff, both of which are pluses for Snowflake and minuses for Databricks.

Annual ETL Development Cost:

Org Size	**Snowflake Hrs**	**Snowflake $'s**	**Databricks Hours**	**Databricks $'s**
Small	6,460	$775,200	6,800	$816,000
Medium	16,150	$1,938,000	17,000	$2,040,000
Large	48,450	$5,814,000	51,000	$6,120,000

Functional – BI / Reporting

The BI and Reporting team spends most of their time in a tool. Common tools include Power BI, Tableau, Looker, DOMO and others.

70% of their work function is spent within these tools. The tools connect to the backing data warehouse via jdbc/odbc. There is no difference between Databricks and Snowflake for this portion of their time.

It is the 30% of the job done outside the BI tool where the traditional BI developer will find much more ease with Snowflake than with Databricks, for a number of reasons:

1. <u>SQL Implementation and Familiarity:</u> Snowflake's SQL is very close to standard ANSI SQL, which makes it familiar to anyone who has experience with traditional SQL databases. This familiarity can make it easier for users to transition to or start using Snowflake without a steep learning curve.
2. <u>Ease of Querying:</u> In Snowflake, querying data is straightforward due to its relational database structure and SQL-friendly nature. Users can write standard SQL queries without needing to consider the underlying complexity of data formats or distributed data processing, which is often abstracted away.
3. <u>Less Need for Optimization</u>: Snowflake automatically handles many of the performance optimizations, such as query optimization, data partitioning, and clustering. This means that users often do not need to write complex SQL for performance reasons, which can be a requirement in some Databricks scenarios, especially when dealing with large-scale data processing.
4. <u>Simplified Data Warehouse Operations</u>: For typical data warehouse operations like aggregations, joins, and reporting, Snowflake's SQL operations are usually more straightforward and require less customization or tuning compared to Databricks, which is built on Apache Spark and may require more Spark-specific optimizations and considerations.
5. <u>Integrated Data Warehouse Features</u>: Snowflake includes features like zero-copy cloning, time travel, and automatic scaling, which are accessible via simple SQL commands. These features can simplify data management tasks without the need for complex SQL scripting.
6. <u>User Interface and Tooling</u>: The Snowflake web interface provides an easy-to-use environment for writing and executing SQL queries, along with helpful features like query history, query plan visualization, and result caching, which can simplify the SQL development process.
7. <u>Data Sharing Capabilities</u>: Snowflake's data sharing capabilities allow users to easily share data across different accounts or even with external entities using SQL, without complex ETL processes or data duplication.

In contrast, SQL in Databricks, while also powerful and flexible, is often used in the context of larger data engineering and data science workflows. Databricks is built on top of Apache Spark, and leveraging its full capabilities often requires an understanding of Spark's distributed computing model. This can involve considerations like data partitioning, optimizing data formats for Spark, and sometimes integrating with other languages like Python or Scala for more complex operations.

Ignoring the fact that you can use a less expensive resources if using Snowflake, the cost difference between the two is as follows:

Org Size	Snowflake Hrs	Snowflake $'s	Databricks Hours	Databricks $'s
Small	5,824	$465,920	6,400	$512,000
Medium	14,560	$1,164,800	16,000	$1,280,000
Large	43,680	$3,494,400	48,000	$3,840,000

Functional – Analysts / Power Users

The following tables compares the list of tools and technologies that a citizen developer business analyst needs expertise in to perform their daily job.

% of Job	Activity	Snowflake	Databricks
50%	Analyzing trends in existing data.	SQL, Snowpark	Apache Spark, Spark SQL, Data Frames and Spark API
20%	Quickly pulling in data to join against existing data.	Snowflake command COPY INTO, data sharing	DB Utils or DB Delta or DB Notebook jobs
15%	Sharing data with internal users and external partners.	Snowflake data sharing feature to share securely between Snowflake accounts.	DBFS, DB Utils and Delta sharing is possible through unity catalog.
15%	Integrating data with other systems (to/from).	Snowflake commands, SQL	DB Utils, Spark data frames

While Databricks offers similar functionality to citizen developers, it does come with a steeper learning curve relative to Snowflake. The tools used to do so in Databricks require more time to do so than in Snowflake (e.g. to source data, in Snowflake you use a COPY INTO, in Databricks you create a spark program to read the delimited data and saveAs dataframe). To operate effectively in Databricks, the customer will need to invest time and effort in gaining proficiency in Apache Spark and related technologies and allocate more time per initiative to achieve the desired result.

For citizen developers, Snowflake would be a significantly easier tool to use compared to Databricks for a number of reasons:

Ease of Use:

Snowflake: Designed for ease of use with a simple SQL interface and point-and-click features. No coding required for basic tasks.
Databricks: Requires coding knowledge, primarily in Python, Scala, or R, and executed via the spark framework which can be a barrier for citizen developers.

Learning Curve:

Snowflake: Offers a shallower learning curve, with readily available documentation and tutorials.

Databricks: Requires significant effort to learn Spark and associated libraries, making it more challenging for beginners.

Pre-built Functionality:

Snowflake: Provides pre-built dashboards and BI tools that citizen developers can readily use without needing to develop complex queries.

Databricks: Requires more development effort to build similar functionality, as it primarily offers a platform for building custom solutions.

Customization:

Snowflake: Limited customizability, which is often not a concern for citizen developers who primarily need to perform analysis.

Databricks: Highly customizable but requires advanced coding skills to leverage this effectively.

Overall, Snowflake's user-friendly interface, pre-built functionality, and lower learning curve make it a better fit for citizen developers who need to perform basic geo location analysis without significant technical expertise.

Efficiency gains will vary from 20-40%, with 40% witnessed in the large-scale Telco domain. To be conservative, we will place the efficiency gain at 25%:

Org Size	Snowflake Hrs	Snowflake $'s	Databricks Hours	Databricks $'s
Small	9,000	$720,000	12,000	$960,000
Medium	22,500	$1,800,000	30,000	$2,400,000
Large	67,500	$5,400,000	90,000	$7,200,000

Functional – Traditional Machine Learning

The following are standard duties of a Data Scientist. Let's compare Snowflake and Databricks on what is required for Data Scientist to perform those duties regularly.

Snowflake

A data scientist can create an ML model in Snowflake using several methods, each with its own strengths and weaknesses. Here are some of the most common approaches:

1. Snowpark Python UDFs:
This method allows data scientists to write custom ML models in Python and use them directly within Snowflake. This provides flexibility and control but requires a strong understanding of Python and Snowflake UDFs.
Steps:

- Develop your ML model in Python: Choose an appropriate library like scikit-learn, TensorFlow, or PyTorch.
- Create a Snowpark UDF: Wrap your Python model into a UDF using the Snowpark Python API.
- Execute the UDF in Snowflake: Use the UDF to apply your model to the data stored in Snowflake tables.

2. Snowflake External Functions:
This method allows data scientists to leverage pre-trained ML models from various providers like AI21 Labs, Reka, and NVIDIA. This is easier to use than developing a custom model but offers less flexibility and control.
Steps:

- Identify a suitable pre-trained model: Choose a model that aligns with your specific task and data type.
- Access the model through Snowflake: Use the Snowflake External Functions feature to access and run the model on your Snowflake data.

3. Snowpark Container Services (Private Preview):
With this method, data scientists can run and fine-tune LLMs directly in Snowflake using containerized data applications. This offers high performance and scalability but requires expertise in Docker and container technologies.
Steps:

- Prepare your containerized LLM application: Package your LLM model and dependencies into a Docker container image.
- Deploy the container image to Snowflake: Use the Snowpark Container Services API to deploy your image to a Snowflake cluster.
- Run and interact with the LLM application: Use Snowpark APIs to send data to the container and receive predictions.

4. Integration with Third-Party Tools:
Snowflake integrates with various third-party ML platforms like Databricks, Amazon SageMaker, and Google Cloud Vertex AI. This allows data scientists to leverage the benefits of these platforms within the Snowflake environment.

Databricks

Databricks offers an end-to-end data science framework and set of tools for the data scientist to perform their job:

1. Setting Up:
   - Cluster configuration: Choose a cluster with appropriate resources (CPUs, RAM, storage) based on your data size and model complexity.
   - Library installation: Install necessary libraries like TensorFlow, PyTorch, Spark MLlib, or others specific to your model.
   - Data access: Configure access to your data stored in various sources like S3, ADLS gen2, etc.
2. Data Preparation:
   - Data loading: Load your data into Databricks using the appropriate connector.
   - Data exploration: Analyze data statistics, identify missing values, and understand data distribution.
   - Data pre-processing: Cleanse the data, handle missing values, and format it for training.
   - Feature engineering: Create relevant features based on domain knowledge and data analysis.
   - Data splitting: Split your data into training, validation, and testing sets for model training and evaluation.
3. Model Training:
   - Model selection: Choose an appropriate ML algorithm based on your task and data type (e.g., classification, regression, clustering).
   - Model training: Train the model on the training data.
   - Hyperparameter tuning: Optimize model performance by adjusting hyperparameters through grid search, random search, or Bayesian optimization.
   - Model evaluation: Evaluate the model's performance on the validation set.
4. Model Deployment:
   - Model packaging: Save the trained model in a format suitable for deployment (e.g., MLflow model, PMML).
   - Model deployment: Deploy the model as a REST API, batch scoring process, or integrate it into applications.
   - Model monitoring: Track model performance in production, detect performance degradation, and trigger retraining if necessary.
5. Additional Tools and Features:
   - Databricks AutoML: Automates the process of model selection, hyperparameter tuning, and model training for faster model development.
   - Databricks Feature Store: Centralized repository for managing and sharing features across different ML projects.
   - Databricks Model Registry: Provides a central platform for storing, versioning, and managing ML models.
   - MLflow: Open-source platform for managing the ML lifecycle, including model tracking, experiment comparison, and reproducibility.

Databricks offers the more complete package of tooling for the Data Scientist. This may change with Snowflake Containerized Services, but for now, we can assume a 20% efficiency gain with Databricks over Snowflake:

Org Size	Snowflake Hrs	Snowflake $'s	Databricks Hours	Databricks $'s
Small	3,840	$460,800	3,200	$384,000
Medium	9,600	$1,152,000	8,000	$960,000
Large	28,800	$3,456,000	24,000	$2,880,000

Common Use Case - Comparative Price Performance

Note: The purpose of this case study is about organizational impact – not price performance. A separate case study on price performance will be delivered 1Q24. We include here a single use case, just for reference.

Experience and industry benchmarks show a >30% increase in price performance and performance of Snowflake over Databricks.

We ran a common use case using SPY Options ticker data. For Databricks, we ran in a few scenarios to get the most optimal time. The following are the results.

	Snowflake		Databricks xlarge		Databricks xlarge2		Databricks xlarge4	
Type	**Ingest Time**	**$ per Query**	**Ingest Time**	**$ per Query**	**Ingest Time**	**$ per Query**	**Ingest Time**	**$ per Query**
Incoming to Bronze	55 sec	$.046	3 min 11 sec	$.042	1 min 23 sec	$.039	40 sec	$.039
Bronze to Gold	23 sec	$.019	2 min 52 sec	$.035	1 min 58 sec	$.047	1 min 41 sec	$.086
Total	1 min 18 sec	**$.065**	6 min 2 sec	**$.077**	3 min 21 sec	**$.086**	2 min 21 sec	**$.125**

For Snowflake, we used x-small.

For Databricks we used gen purpose interactive for incoming to bronze and SQL Warehouse Pro for Bronze to Gold. However, from a price-performance perspective, the SQL Warehouse Pro although faster, was worse from a performance perspective. So, we then switched the bronze to gold to use general purpose. We ran with different sizes to show linear price performance.

If better performance is desired, for Databricks, one should move to SQL Warehouse Pro though the price performance is not there. Different queries will have different characteristics in terms of where and how they should be run on Databricks.

Further tuning can get the performance and price-performance numbers closer to Snowflake.

For Snowflake we are using extra-small.
For Databricks, we are using DBX 13.3 with i3.xlarge AWS instance:

There is a huge difference in the spin-up and auto-suspend/terminate and spin-up times and costs:
Snowflake: Best practice is to auto-suspend at 2-3 minutes.
Databricks: Lowest achievable terminate is 10 minutes.

Note: With Databricks you pay for compute. While it's not known what percentage of costs are incurred during startup, for this document we assume $0 (though in practice there is some cost).

Pricing Calcs:

		Incoming->Bronze	Bronze->Gold
Snowflake	**Duration**	55	23
	Warehouse type	x-small	x-small
	Credits	0.01528	0.00639
	$'s/credit price	$3	$3
	$'s	**$0.046**	**$0.019**
	Total	**$0.065**	

		Incoming->Bronze	Bronze->Gold
dbx (gen purpose / gen purpose)	**Duration**	191	172
	Cluster type	Gen purpose	Gen purpose
	DBU/hr rate	1	1
	$/DBU price	$0.40	$0.40
	$/hr ec2 price (i3.xlarge)	$0.312	$0.312
	$/hr ec2 price (i3.2xlarge)	$0.624	$0.624
	$/hr ec2 price (i3.4xlarge)	$1.248	$1.248
	# ec2 instances (i3.xlarge)	1	1
	# ec2 instances (i3.2xlarge)	0	0
	# ec2 instances (i3.4xlarge)	0	0
	AWS $'s (i3.xlarge)	$0.021	$0.016
	DBX $'s	$0.021	$0.019
	$'s	**$0.042**	**$0.035**
		$0.077	

		Incoming->Bronze	Bronze->Gold
dbx (gen purpose 2xl / gen purpose 2xl)	**Duration**	83	118
	Cluster type	Gen purpose	Gen purpose
	DBU/hr rate	2	2
	$/DBU price	$0.40	$0.40
	$/hr ec2 price (i3.xlarge)	$0.312	$0.312
	$/hr ec2 price (i3.2xlarge)	$0.624	$0.624
	$/hr ec2 price (i3.4xlarge)	$1.248	$1.248
	# ec2 instances (i3.xlarge)	0	0
	# ec2 instances (i3.2xlarge)	1	1
	# ec2 instances (i3.4xlarge)	0	0
	AWS $'s (i3.xlarge)	$0.021	$0.021
	DBX $'s	$0.018	$0.026
	$'s	**$0.039**	**$0.047**
		$0.086	

		Incoming->Bronze	Bronze->Gold
	Duration	40	101

dbx (gen purpose 4xl / gen purpose 4xl)	**Cluster type**	Gen purpose	Gen purpose
	DBU/hr rate	4	4
	$/DBU price	$0.40	$0.40
	$/hr ec2 price (i3.xlarge)	$0.312	$0.312
	$/hr ec2 price (i3.2xlarge)	$0.624	$0.624
	$/hr ec2 price (i3.4xlarge)	$1.248	$1.248
	# ec2 instances (i3.xlarge)	0	0
	# ec2 instances (i3.2xlarge)	0	0
	# ec2 instances (i3.4xlarge)	1	1
	AWS $'s (i3.xlarge)	$0.021	$0.042
	DBX $'s	$0.018	$0.045
	$'s	**$0.039**	**$0.086**
		$0.125	

Again, a separate case study with a variety of common use cases will be delivered 1Q24. This is a single common use case just for reference

Appendix - Migration to Unity Catalog

Migration from hms to unity catalog in Databricks is an area where a lot of Databricks customers are struggling with. If you are in this situation, you may find it easier to move to another platform. If you do wish to attempt to move hms's to Unity Catalog, here are the steps you will need to take per HMS.

There are specific steps if you want to migrate from Hive Metastore (which most people use still) to Unity Catalog (released in 2021):

1. Prepare your environment:
    a. Ensure you have a storage credential with an IAM role that authorizes Unity Catalog to access the tables' location path.
    b. Create an external location that references the storage credential and the path to the data on your cloud tenant.
    c. Grant CREATE EXTERNAL TABLE permission on the external locations of the tables to be upgraded.
2. Upgrade tables and views:
    a. Open the Catalog Explorer in the Databricks workspace.
    b. Select the hive_metastore catalog and choose the schema (database) containing the tables you want to upgrade.
    c. Click the "Upgrade" button at the top right of the schema detail view.
    d. Select the tables you want to upgrade and click "Next."
    e. Specify the destination catalog, schema (database), and owner for each table.
    f. Click "Next" and then "Upgrade" to start the upgrade process.
3. Verify upgraded tables and views:
    a. Once the upgrade is complete, verify that the tables and views are now registered in Unity Catalog.
    b. You can check this by querying the tables or views using their fully qualified names in the context of the Unity Catalog catalog and schema.
4. Update access permissions:
    a. Review and update access permissions for the newly upgraded tables and views in Unity Catalog.
    b. Grant appropriate permissions to principals who need access to these data assets.
5. Decommission Hive Metastore:
    a. Once you have confirmed that all tables and views have been successfully upgraded and access permissions are set correctly, you can decommission the Hive Metastore.
    b. This involves removing the Hive Metastore configuration and ensuring that no applications are still using it.