Whitepaper

Benchmarking Snowpark for Python in Keboola

A study of performance and costs for small and big data engineering tasks
Executive Summary

We compared Snowpark's performance across three data engineering use cases: a basic query in SQL, string manipulations, and time-series predictions. We varied the dataset sizes and compared Snowpark's performance against several other technologies, including Python, SQL, and managed Spark.

The results indicate that Snowpark is highly proficient in managing substantial volumes of data with no detrimental effect on performance. It surpassed Spark in 7 out of 8 implementations, demonstrating both superior execution speed and cost efficiency. The sole exception was pure Python, which proved faster and more cost-effective for simple data wrangling tasks on smaller datasets.

In summary, based on its performance and cost-effectiveness, we recommend Snowpark as the go-to solution for (big) data engineering.
Table of Contents

Executive Summary 2

1. The motivation for benchmarking Snowpark for Python 4
   1.1 What is Snowpark for Python? 4
   1.2 What are the advantages of Snowpark? 5
   1.3 What is the benchmarking goal? 5

2. Benchmarking methodology 6
   2.1 Three dimensions of benchmarking 6
   2.2 Dataset (size) 6
   2.3 Baseline technology 8
      2.3.1 Keboola infrastructure specifications 8
      2.3.2 Databricks infrastructure specifications 8
   2.4 Tested use cases 9
      2.4.1 Use Case #1: Simple data engineering 9
      2.4.2 Use Case #2: String manipulation 9
      2.4.3 Use Case #3: Time-series predictions using Prophet 9
   2.5 Benchmark success metric: Runtime 10

3. Benchmark results 10
   3.1 Snowpark benchmark against Python and Spark on different use cases and dataset sizes 13
   3.2 Snowpark benchmark against Spark: Processing time 15
   3.3 Cost comparison 16

4. Overall observation 18

5. Considerations for benchmarking generalizations 18

6. Keboola with Snowflake and Streamlit 19
1. The motivation for benchmarking Snowpark for Python

Snowflake started by disrupting data analytics, then collaboration, and now data applications. With its fully-managed, endlessly scalable, and performant platform, Snowflake disrupted the market and broke down silos by creating the Data Cloud.

Snowflake began as a predominantly SQL-based platform - the standard language of data professionals. However, a significant contingent of professionals favor working with data in programming languages other than SQL. These include data scientists and engineers who depend on Python, Java, and Scala for their data operations.

These programming languages offer the advantage of an active data community, libraries that take care of the heavy lifting for analytics and engineering, and frameworks like Streamlit that effortlessly productize data into apps.

This is where Snowpark comes into the picture. Snowpark for Python became generally available within Snowflake in 2022.

Snowpark is the set of libraries and runtimes that securely deploy and process non-SQL code in Snowflake, including Python, Java, and Scala. It allows data engineers, data scientists, and data developers to code in their familiar way with their language of choice, and execute pipelines, ML workflows, and data apps faster and more securely, all within Snowflake.

Snowpark takes care of the background work to make coding languages such as Python, Scala, or Java, run on Snowflake data seamlessly and in a fully optimized manner. There is no additional effort required from the user.

1.1 What is Snowpark for Python?

Snowpark for Python (and hereafter referred to as only “Snowpark”) includes a set of libraries and runtime contracts that securely deploy and process Python code in Snowflake. Snowpark includes:

1. **Snowpark DataFrame API**: This client-side library enables Python developers to write code natively in Snowflake using familiar DataFrames.

2. **Snowpark ML API**: Snowpark ML is a client-side library for faster and more intuitive end-to-end ML development in Snowflake. Snowpark ML has 2 APIs: Snowpark ML Modeling (public preview) for model development and Snowpark ML operations (private preview) for model deployment.

3. **Snowpark User Defined Functions (UDFs)**: Server-side runtime contract that executes custom Python code in Snowflake, including business logic or trained
machine learning models. Leverage the embedded Anaconda repository for effortless access to thousands of open-source libraries.

4. **Snowpark Stored procedures**: Server-side runtime contract that operationalizes and orchestrates DataFrame operations and custom Python code to enable users to build and run data pipelines within Snowflake.

### 1.2 What are the advantages of Snowpark?

Snowpark promises many advantages:

- **Language of choice.** Enable all data users to bring their work to a single platform with native Python, Java, and Scala support to develop data pipelines, machine learning models, applications, and more.
- **Improved security and governance.** Moving data can cause security issues. Snowpark mitigates this issue by allowing code to run within Snowflake, eliminating the need to move data.
- **Price and performance.** Customers see a median of 3.5x faster performance and 34% cost savings with Snowpark over managed Spark for data engineering and ML workloads.
- **Custom app development.** Software teams can create custom applications using the data stored in Snowflake and the programming logic of their preferred language.
- **Extensibility.** Python, Java, and Scala offer much more flexibility than SQL and enable the creation of robust open-source software. Snowpark enables the execution of these libraries on Snowflake, allowing developers to efficiently process both internal and external data within the Snowflake environment.
- **Democratization and collaboration.** Snowpark allows the same analytical queries to be written in either Snowflake's SQL or Python (or Java or Scala). This inclusivity invites a broader range of professionals to partake in data engineering, science, and analytics projects (regardless of their technical background), thus promoting collaboration among data professionals.
- **Simplified DevOps.** With Snowpark, the DevOps CI/CD pipelines and tests can be written and run within Snowflake. This reduces the complexity of the tech stack and simplifies operations.
- **Lower storage costs.** Snowpark's ability to work with compressed data can result in lower storage costs compared to uncompressed data formats like CSV/Parquet.
- **Inherited Snowflake’s benefits.** Snowpark comes with all the performance, elasticity, scalability, and security benefits that Snowflake offers, which are critical for production workloads.

Given these potential benefits, data teams may question whether or not they should transition from their current technology stack to Snowflake for machine learning and data engineering tasks.

This question becomes especially relevant when considering the memory limitations of Python and the significant overhead involved in making Spark operational.
The best way to answer this question is to assess Snowpark against a benchmark.

1.3 What is the benchmarking goal?

The purpose of this whitepaper is to conduct a benchmarking analysis of Snowpark for Python, comparing it to other technologies in the space of data engineering and machine learning.

This analysis aims to assist data teams in determining whether Snowpark's offering outweighs those of alternative data stacks.

The whitepaper aims to answer the main research question: “Can Snowpark for Python outperform Python (standard library and other Python frameworks) and Spark when building secure and scalable data pipelines and machine learning (ML) workflows directly within Snowflake?”

To provide an answer, the Keboola team designed and executed a series of benchmarking tests to compare the performance and cost-effectiveness of running data pipelines in Snowpark in comparison to other available technologies.

2. Benchmarking methodology

2.1 Three dimensions of benchmarking

The benchmarks were developed on three orthogonal dimensions to evaluate the performance and costs of Snowpark for Python based on:

1. **Dataset size**: How does a change in dataset size (by order of magnitude) impact the performance and cost of Snowpark?
2. **Baseline technology**: How do Snowpark's performance and costs compare with other technological solutions in this domain?
3. **Use case**: How do Snowpark's performance and costs vary across different data engineering scenarios?

We will now explore each in detail.

2.2 Dataset (size)

The benchmarking research used the TPC-H database. From the TPC Benchmark™ H (TPC-H) specification:

“TPC-H is a decision support benchmark. It consists of a suite of business-oriented ad hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark
illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions."

The database offers a familiar business schema:

**Figure 2: The TPC-H Schema**

![Diagram of TPC-H Schema]

**Legend:**
- The parentheses following each table name contain the prefix of the column names for that table;
- The arrows point in the direction of the one-to-many relationships between tables;
- The number/formula below each table name represents the cardinality (number of rows) of the table. Some are factored by SF, the Scale Factor, to obtain the chosen database size. The cardinality for the LINEITEM table is approximate (see Clause 4.2.5).

(source: [TPC Benchmark H Standard Specification](https://www.tpc.org/benchmarks/h/))
Critically, the database offers many sizes to test technology against volumes of data that increase in orders of magnitude:

- **TPCH_SF1**: Consists of the base row size (several million elements).
- **TPCH_SF10**: Consists of the base row size x 10.
- **TPCH_SF100**: Consists of the base row size x 100 (several hundred million elements).
- **TPCH_SF1000**: Consists of the base row size x 1000 (several billion elements).

### 2.3 Baseline technology

Different teams use different technologies for their (big) data engineering transformations.

Accordingly, the benchmark compares the performance of Snowpark for Python with other alternatives for data wrangling:

- **Pure Python (standard library)**
- **Pandas**
- **SQL (Snowflake)**
- **Spark**

The performance and costs of each technology are tightly coupled with the infrastructure on which they run. From allocated memory to parallelized distributed workers, the infrastructure has a crucial effect on the outcome of the tests.

For this reason, the benchmarking tests were performed on two infrastructures in two phases:

1. Keboola
2. Databricks

#### 2.3.1 Keboola infrastructure specifications

The transformation scripts were run on different Keboola backends ([technical specifications](#)). The principle was to use the smallest backend first and **dynamically scale** it based on the needs of the tested use case.

In Keboola, the backend size directly correlates with costs and performance. A larger backend consumes more Keboola credits, but it also delivers enhanced performance.

### 2.4 Tested use cases

Three engineering use cases were used to assess the performance and costs of Snowpark on different task complexities.
2.4.1 Use Case #1: Simple data engineering

The instructions for building the use case were: “For all ORDERS in AUTOMOBILE segment with status = ‘F’, calculate the number of distinct line items, the total quantity of line items, the average discount per line item, and the total price of the order.”

2.4.2 Use Case #2: String manipulation

The instructions for building the use case were: “Split the column TPCH_SFNC.LINEITEM.COMMENT by space and capitalize each word.”

String manipulations can be challenging to perform in SQL, often necessitating the chaining of multiple functions.

Python offers a more straightforward approach to string manipulations. However, when working with large datasets, the following tradeoffs need to be considered:

- Must be processed row by row with pure Python
- Pandas can run out of memory or need to be customized for batch processing
- Snowpark and Spark require a UDF implementation.

2.4.3 Use Case #3: Time-series predictions using Prophet

The instructions for building the use case were: “Use the Prophet library to predict the number of daily orders 360 days into the future.”

Predicting with Prophet is a fairly common use case, as it provides a simple and satisfactory way to make time-series predictions. It is also something that requires Python and has no easy alternative to pure SQL. This use case also allows us to evaluate the extensibility of Snowpark in comparison to other options.

2.5 Benchmark success metric: Processing Time

Processing time - the time it takes in seconds to execute a transformation from start to end - is an excellent benchmarking metric. It allows for easy performance comparisons across various technologies, frameworks, use cases, and data sizes.

Additionally, the processing time is directly linked to cost. Platforms such as Keboola, Databricks, Snowflake, and other modern data engineering solutions use processing time as their pricing unit. The more processing (memory, CPU, TPU, etc.) you consume, the more you are charged.
# 3. Benchmark results

The tests comparing Snowpark’s performance across different use cases and different technologies are summarized in the table below.

<table>
<thead>
<tr>
<th>Processing Backend</th>
<th>Use Case</th>
<th>Data Set</th>
<th>DataSet Size (Compressed SNFLK GB)</th>
<th>Storage Costs (USD/month)</th>
<th>Processing Time (s)</th>
<th>Processing Costs SMALL</th>
<th>TCO Monthly (Weekly processing)</th>
<th>TCO Monthly (Daily processing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>Data engineering</td>
<td>SF10 (uncompressed csv)</td>
<td>6.125</td>
<td>$0.25</td>
<td>152</td>
<td>0.024</td>
<td>$0.34</td>
<td>$0.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SF100 (uncompressed csv)</td>
<td>73.5</td>
<td>$2.94</td>
<td>not applicable</td>
<td>not applicable</td>
<td>not applicable</td>
<td>not applicable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SF1000 (uncompressed csv)</td>
<td>766.5</td>
<td>$30.66</td>
<td>not applicable</td>
<td>not applicable</td>
<td>not applicable</td>
<td>not applicable</td>
</tr>
<tr>
<td></td>
<td>String manipulation</td>
<td>SF10 (uncompressed csv)</td>
<td>5.6</td>
<td>$0.22</td>
<td>420</td>
<td>0.065</td>
<td>$0.49</td>
<td>$2.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SF100 (uncompressed csv)</td>
<td>52.5</td>
<td>$2.10</td>
<td>5040</td>
<td>0.784</td>
<td>$5.24</td>
<td>$25.62</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SF1000 (with pandas chunks)</td>
<td>52.5</td>
<td>$2.10</td>
<td>7560</td>
<td>1.176</td>
<td>$6.80</td>
<td>$37.38</td>
</tr>
<tr>
<td></td>
<td>Predictions using Prophet</td>
<td>SF10 (uncompressed csv)</td>
<td>14</td>
<td>$0.56</td>
<td>210</td>
<td>0.033</td>
<td>$0.69</td>
<td>$1.54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SF1000 (uncompressed csv)</td>
<td>171.5</td>
<td>$6.86</td>
<td>not applicable</td>
<td>not applicable</td>
<td>not applicable</td>
<td>not applicable</td>
</tr>
<tr>
<td>Snowpark</td>
<td>Data engineering</td>
<td>SF10 (SNFLK)</td>
<td>1.75</td>
<td>$0.07</td>
<td>7</td>
<td>$0.010</td>
<td>$0.11</td>
<td>$0.37</td>
</tr>
<tr>
<td></td>
<td>compresse d)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SF100 (SNFLK compresse d)</td>
<td>21</td>
<td>$ 0.84</td>
<td>17</td>
<td>$ 0.024</td>
<td>$ 0.94</td>
<td>$ 1.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SF1000 (SNFLK compresse d)</td>
<td>219</td>
<td>$ 8.76</td>
<td>140</td>
<td>$ 0.201</td>
<td>$ 9.56</td>
<td>$ 14.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String manipulation</td>
<td>SF10 (SNFLK compresse d)</td>
<td>1.6</td>
<td>$ 0.06</td>
<td>71</td>
<td>$ 0.102</td>
<td>$ 0.47</td>
<td>$ 3.12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SF100 (SNFLK compresse d)</td>
<td>15</td>
<td>$ 0.60</td>
<td>382</td>
<td>$ 0.548</td>
<td>$ 2.79</td>
<td>$ 17.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SF1000 (SNFLK compresse d)</td>
<td>159</td>
<td>$ 6.36</td>
<td>3600</td>
<td>$ 5.162</td>
<td>$ 27.01</td>
<td>$ 161.23</td>
<td></td>
</tr>
<tr>
<td>Predictions using Prophet</td>
<td>SF100 (SNFLK compresse d)</td>
<td>4</td>
<td>$ 0.16</td>
<td>27</td>
<td>$ 0.039</td>
<td>$ 0.31</td>
<td>$ 1.32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SF1000 (SNFLK compresse d)</td>
<td>49</td>
<td>$ 1.96</td>
<td>34</td>
<td>$ 0.049</td>
<td>$ 2.16</td>
<td>$ 3.42</td>
<td></td>
</tr>
</tbody>
</table>

<p>|                      | SF10 (uncompre ssed csv) | 6.125   | $ 0.12  | 33      | $ 0.009 | $ 0.16  | $ 0.39  |
|                      | SF100 (uncompre ssed csv) | 73.5    | $ 1.47  | 380     | $ 0.104 | $ 1.89  | $ 4.60  |
|                      | SF1000 (uncompre ssed csv) | 766.5   | $ 15.33 | 4680    | $ 1.284 | $ 20.47 | $ 53.86 |
| Spark                | SF10 (uncompre ssed csv) | 5.6     | $ 0.11  | 160     | $ 0.044 | $ 0.29  | $ 1.43  |
| String manipulation  | SF100 (uncompre ssed csv) | 52.5    | $ 1.05  | 1534    | $ 0.421 | $ 2.73  | $ 13.68 |</p>
<table>
<thead>
<tr>
<th></th>
<th>(uncompressed csv)</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Predictions using Prophet</strong></td>
<td>SF100 (uncompressed csv)</td>
<td>556.5</td>
<td>$11.13</td>
<td><strong>14976</strong></td>
<td>$4.110</td>
<td>$27.57</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>SF100 (uncompressed csv)</td>
<td>14</td>
<td>$0.28</td>
<td><strong>16</strong></td>
<td>$0.004</td>
<td>$0.30</td>
</tr>
<tr>
<td></td>
<td>SF1000 (uncompressed csv)</td>
<td>171.5</td>
<td>$3.43</td>
<td><strong>131</strong></td>
<td>$0.036</td>
<td>$3.57</td>
</tr>
<tr>
<td><strong>Data engineering</strong></td>
<td>SF1000 (uncompressed csv)</td>
<td>219</td>
<td>$8.76</td>
<td><strong>140</strong></td>
<td>0.201</td>
<td>$9.56</td>
</tr>
</tbody>
</table>
3.1 Snowpark benchmark against Python and managed Spark on different use cases and dataset sizes

Graph - Runtime Use Case 1

Data set 1
- Snowpark: 7
- Spark: 33
- SQL: 152
- Python: N/A

Data set 2
- Snowpark: 17
- Spark: 380
- SQL: N/A
- Python: N/A

Data set 3
- Snowpark: 140
- Spark: 140
- SQL: N/A
- Python: N/A

Graph - Runtime Use Case 2
Highlights:

- Snowpark successfully executed all transformations on a small backend. In contrast, traditional data engineering technologies like Pandas and pure Python failed on the largest Keboola backends due to memory issues. To make these traditional technologies function, additional overhead is required, such as scripting to process data row-wise or in batches.
- While Pandas showcased faster performance than Snowpark for small datasets, Snowpark consistently outperformed Python and SQL for larger datasets and across various use cases.
- Even when data size increased by orders of magnitude, Snowpark maintained consistent performance. This characteristic positions Snowpark as an excellent choice for scalability.
3.2 Snowpark benchmark against managed Spark: Processing time

When comparing Snowpark and Spark, Snowpark is the clear winner.

<table>
<thead>
<tr>
<th>Dataset (size)</th>
<th>Snowpark (sec)</th>
<th>Spark (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case #1 - Simple Data Engineering</td>
<td>SF10</td>
<td>7</td>
</tr>
<tr>
<td>SF100</td>
<td>17</td>
<td>380</td>
</tr>
<tr>
<td>SF1000</td>
<td>140</td>
<td>4680</td>
</tr>
<tr>
<td>Use Case #2 - String Manipulations</td>
<td>SF10</td>
<td>71</td>
</tr>
<tr>
<td>SF100</td>
<td>382</td>
<td>1534</td>
</tr>
<tr>
<td>SF1000</td>
<td>3600</td>
<td>14976</td>
</tr>
<tr>
<td>Use Case #3 - Prophet Predictions</td>
<td>SF100</td>
<td>27</td>
</tr>
<tr>
<td>SF1000</td>
<td>34</td>
<td>131</td>
</tr>
</tbody>
</table>

Key benchmarking results:

<table>
<thead>
<tr>
<th>Snowpark use cases result in comparison to Spark</th>
<th>Use cases outperformed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snowpark</td>
<td>Spark</td>
</tr>
<tr>
<td>Simple Data Engineering</td>
<td>5-33x faster</td>
</tr>
<tr>
<td>String Manipulations</td>
<td>2-4x faster</td>
</tr>
<tr>
<td>Predictions using Prophet</td>
<td>1,5x slower / 4x faster</td>
</tr>
<tr>
<td>Total score</td>
<td>7</td>
</tr>
</tbody>
</table>

- **Use case #1 (Basic operations in SQL):** Snowpark outperformed Spark across all tested infrastructures and dataset sizes, demonstrating 5-33 times faster performance.
- **Use case #2 (String manipulation):** Snowpark showcased faster performance than Spark, though the difference was smaller compared to the first use case. It outperformed Spark by 2-4 times in terms of speed.
- **Use case #3 (Time-series predictions with Prophet):** Snowpark was 1-5 times slower on smaller datasets but 4 times faster on larger datasets.
Looking at the direct results, it is clear that Snowpark outperforms Spark in most scenarios – it came out on top in 7 out of 8 use cases, including those involving machine learning tasks. The only instance where Spark led the charge was in one particular machine learning use case.

In addition, the benchmarking tests solely focused on runtime performance. Customers using managed Spark should consider that its running costs are likely to be higher than Snowpark’s. This is because the tests did not factor in other DataOps expenses.

### 3.3 Cost comparison

We will now delve into the costs associated with running jobs on each framework. While speed is essential, budgetary constraints must also be considered. Here's an overview of the cost comparison:

- For simple data engineering tasks, Snowpark proves to be the most cost-effective option regardless of the dataset size.
- In the case of string manipulation, Snowpark offers faster processing but it tends to be slightly more expensive.
- When it comes to machine learning tasks, such as prediction using Prophet, Snowpark delivers faster execution, making managed Spark the pricier alternative.

Considering the total cost of ownership, setting up a server infrastructure for Python (either through a cloud vendor or locally) can become cost-prohibitive for larger datasets that require expensive high-RAM machines.

Managed Spark users would also need a competent team to prepare Spark scripts and develop within Databricks or other environments, which would incur additional costs (see graphs below).

On the other hand, Snowpark offers a streamlined approach. Users can easily convert a Python script to a Snowpark script, register a UDF, and use it in existing SQL or Python scripts without the need to set up additional infrastructure.

However, it is important to note that a Snowflake database is required to leverage Snowpark’s capabilities.
Important takeaways:

- Using Databricks on top of the data in Snowflake incurs separate costs from Snowflake’s compute expenses.
- There is additional engineering overhead associated with setting up and developing within the Databricks-specific environment.
- A competent team is needed to prepare Spark scripts and develop within Databricks environment.
4. Overall observation

Our recommendation to users is to choose Snowpark for both small and large datasets. Not only is it likely to result in similar or even reduced costs, but it can also simplify your infrastructure and expedite pipeline processing.

Based on the results and tests above, we can draw the following conclusions:

- Snowpark is an exceptional choice for both small and large datasets. In particular, it excels with larger datasets that often cannot be processed in memory during Python transformations. For smaller datasets, pure Python transformations may be cost-equivalent or slightly cheaper, but significantly slower. This was observed across various scenarios, including simple engineering operations that are easily performed in SQL or Python, as well as operations exclusive to Python.
- In comparison to Spark, Snowpark demonstrated faster performance in 7 out of 8 use cases, while maintaining a similar cost profile. Snowpark tends to be more cost-effective for smaller datasets but may be slightly more expensive for larger ones.
- Regarding cost, small transformations are 10 times cheaper to perform in pure Python compared to Snowpark, as illustrated by the SF1 comparison in Use Case 1. The inexpensive cost is attributed to the low-priced machine running a Python transformation in Keboola, which is billed at a rate 6 times lower than a small SQL transformation.
- We also tested operations that are still possible to carry out in SQL but are much simpler in Python (such as string manipulation tested in Use Case 2). We can only confirm the above - Snowpark is superior (and cheaper) for large datasets, while pure Python seems to be cheaper (though slower) for small datasets.

5. Considerations for benchmarking generalizations

Before drawing general conclusions from the benchmarking performance, several factors need to be considered:

1. **Infrastructural nuances**: The configuration of the infrastructure had a significant impact on the benchmarking results. For example, auto-scaling in Databricks has a huge impact on performance (and costs) but it also makes it difficult to understand and estimate the $/hour for individual clusters. Therefore, the speed of execution is more important when evaluating Snowpark vs. Databricks. Additionally, different Databricks clusters exhibited similar performance despite potential cost differences. This may be attributed to our limited experience and understanding of cluster selection.
2. **Technical issues preventing planned comparisons**: We ran into issues when trying to write the data to a new Snowflake object using Spark's .write() function. This
operation took longer than expected, which has been reported by other users as well and is probably due to our Databricks configuration. As a workaround, we exported datasets to the Delta format in Hive Metastore to execute the Spark script directly on it, as this would have been the most likely approach for Databricks users to take. Finally, we decided to execute a simple SQL directly on Hive, too. The operations for Use Case #1 are likely more difficult for Spark to compute as the tables were “just stored” in the Metastore without any optimization (e.g., keys, partitioning), leaving little for the engine to utilize when performing the scans for joins.

6. Keboola with Snowflake and Streamlit

Are you ready to turn data into insights and take app development to the next level? The combination of Snowflake, Streamlit, and Keboola has created an integration that will blow your mind. Let’s dive into why this integration is so amazing for users:

**Streamlit: Creating Interactive Apps in a Snap**

Streamlit is an open-source Python library for quickly building interactive apps from data and Python code. It's widely adopted by large companies and startups for its simplicity and versatility.

**Snowflake: Seamless Integration and Powerful Computing**

Snowflake acquired Streamlit because it recognized the immense potential of combining its capabilities. Soon, you'll be able to run Streamlit applications directly on Snowflake's flexible and elastic compute environment, with no data movement.

**Keboola: Simplified Development, Deployment, and Governance**

Keboola is an end-to-end Data Platform as a Service that takes the development experience to new heights and simplifies the deployment and management of Streamlit apps. Here's what Keboola brings to the table:

- **Managed development environment:** Keboola offers managed Jupyter-based Python Workspaces automatically integrated with Snowflake databases. Users don’t need to fuss over credentials and integration as the workspace is directly connected to Snowflake, allowing easy session initiation.
- **Effortless UDF development:** Within the Keboola workspace, users can interact with Snowflake, develop, test, and register UDFs for pipeline usage. It’s possible to
dynamically set the processing backend size for each job to handle diverse datasets and complex data transformations.

- **Management of your Snowpark scripts**: Keboola simplifies the deployment process of your developed scripts as Transformations, which can be seamlessly integrated into your new or existing pipelines or Flows.

- **Deployment and management of Streamlit apps**: Keboola takes away the hassle of setting up and maintaining separate infrastructure. Users can either use their Git repository or directly input code.

- **Governance**: While any user can develop and register UDFs for future SQL or Python scripts, there's no mechanism for UDF versioning or permission management. Keboola is working towards a managed UDF catalog for an overview of available functions and to facilitate sharing among users.

- **Authorization**: With Streamlit DataApps, Keboola provides the infrastructure necessary for the deployment, hosting, and access management of Streamlit apps. In addition, we offer a variety of 3rd party authorization options, which can be especially useful for large organizations to manage access to specific apps and provide seamless data access management within the Keboola platform.

With the combined power of Snowflake, Streamlit, and Keboola, you have everything you need to unlock the full potential of your data and create exceptional applications. [Click here](#) to see this powerful trio in action.