snowflake®

データエンジニア向け
PYTHON FOR SNOWFLAKEガイド

CHAMPION GUIDES

Eブック

TABLE OF CONTENTS

はじめに

Pythonは、最も人気のあるプログラミング言語として常時トップ3にランクインしており、スタックオーバーフローの年次デベロッパー調査によると、調査対象となった全開発者の68%がPythonでの作業が「好き」だと回答しています。[1]

しかし、長年にわたってデータエンジニアは、Pythonと他のプログラミング言語で、データ変換のためのツールを使い分けることを余儀なくされていました。それらの言語の知識があったとしても、それぞれに別のコンピュート環境を設定して管理するのは面倒であり、時間もかかります。

Snowparkは、Snowflakeのデベロッパーフレームワークであり、Python、SQL、Java、Scalaのネイティブサポートにより、すべてのデータユーザーが自分の作業をSnowflakeデータクラウドに取り込むことができます。Snowparkを利用すると、データエンジニアは、自分の好みの言語を使用して、単一のプラットフォームでMLモデルやアプリケーションに供給するパイプラインをより迅速かつ安全に実行できます。

本書では、Snowparkについて紹介し、さらにSnowflakeデータクラウド内でPythonを使用するためのベストプラクティスについて説明しています。具体的な内容は次のとおりです。

- Snowparkを使用してSnowflakeがデータエンジニアリングをサポートする方法と、その主なメリットとユースケース
- SnowparkがPython、その他のプログラミング言語、およびSQLをサポートする方法
- データエンジニアがPythonを効果的に利用し Snowflakeプラットフォーム内でインパクトを与える方法
- Snowparkをより大きなSnowflakeデータエンジニアリングエコシステムに適合させる方法

さらに、データエンジニアがSnowflakeとSnowparkの使用を開始する際に役立つリソースも紹介します。

SNOWPARKによるデータエンジニアリング

Snowflakeの最新のデベロッパーフレームワークであるSnowparkを使用すると、データエンジニアは好みのプログラミング言語で、管理されたシンプルな高速パイプラインを構築できます。Snowparkは、データエンジニアに多くのメリットを提供します。

- **SQL、Python、Java、Scalaなど複数の言語をサポートする単一のプラットフォーム**
- **ガバナンスを損なうことなくすべてのワークロードでの一貫したセキュリティを実現**
- **より迅速で安価、かつレジリエントなパイプライン**

単一のプラットフォーム

複数の処理エンジンにわたってさまざまなチームが複数の言語を使用する場合、アーキテクチャは大幅に複雑化します。Snowparkは、個別の処理エンジンを必要とせず、選択したプログラミング言語をネイティブにサポートすることによって、アーキテクチャを合理化します。図1を参照してください。Snowparkを利用すると、単一のプラットフォームであるSnowflakeで、すべてのチームが同じデータ上でコラボレーションすることができます。

注目のお客様事例

HyperFinity

小売業・消費財向けノーコード型意思決定インテリジェンスプラットフォームであるHyperFinityは、Snowflakeのお客様ですが、同社ではMLおよびAIのイニシアチブにSQLとPythonを利用しています。Snowparkを採用することで、HyperFinityはこの両方の言語に対応する単一のプラットフォームを持つこととなり、煩雑なデータの移動や、各種サービス間でのデータの移動を維持するために開発されたコードを排除できるようになりました。その結果、HyperFinityではよりシームレスな動作が可能となり、PythonとSQLを1つの環境で開発、テスト、展開できるようになったため、全体としてよりアジャイルな運用が実現しました。

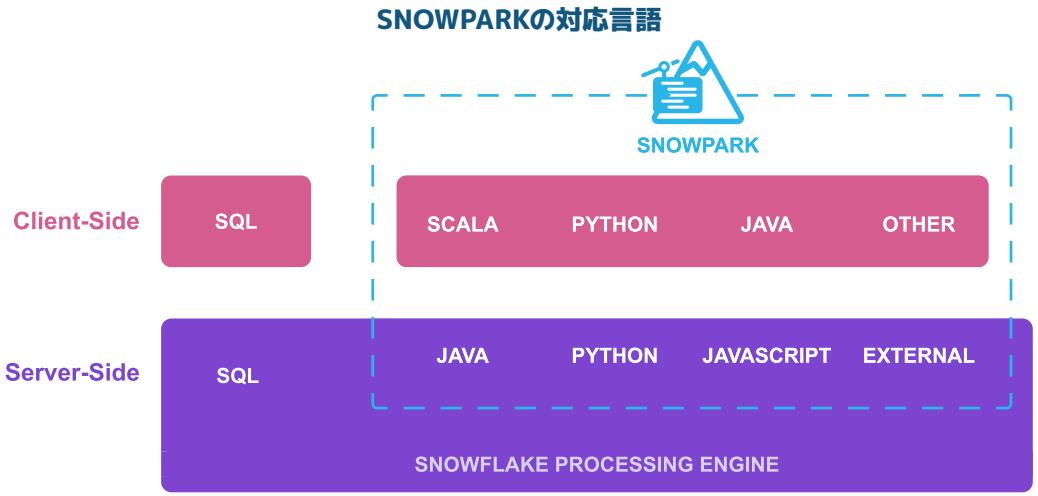

図1：Snowparkを利用すると、複数言語を扱う開発者がSnowflakeのパワーを活用できるようになります

ガバナンスを犠牲にしない

Snowflakeには、エンタープライズグレードのガバナンスコントロールとセキュリティが組み込まれています。たとえば、Snowparkは、データを分離して悪意のあるワークロードからネットワークとホストを保護し、開発者が実行するライブラリを管理者がコントロールできるようにすることで、安全を確保しています。データセキュリティとコンプライアンス対策に一貫性があり、それらがビルトインされているという点で、開発者は構築に自信を持つことができます。

注目のお客様事例

英国の一般家庭や企業向けにガスおよびゼロカーボン電力を供給するEDFは、データアプリケーションの展開にSnowparkを採用しました。Snowflake内で作業することで、このプロジェクトでは、データのアクセシビリティを承認するためのサインオフやミーティングの追加が不要となりました。その代わりにSnowflakeが提供するセキュリティルールとプロジェクトに適用されるセキュリティルールの中で作業することで、EDFチームはシームレスに拡張できます。

Snowparkをデータエンジニアリング業務に統合して以来、EDFでは、顧客向けのMLドリブン型プログラムの制作期間が数か月からわずか3〜4週間に短縮され、生産量は4倍に増加しました。

高速で低コストのパイプライン

Snowparkは、Snowflake独自のマルチクラスター共有データアーキテクチャのおかげで、より優れた価格性能、コストの透明性の向上、運用オーバーヘッドの少ないパイプラインを実現します。Snowflakeは、現代の組織が必要とする性能、規模、伸縮性、同時実行性を実現する、単一の統合プラットフォームです。

注目のお客様事例

ライフサイエンス業界における分析、テクノロジーソリューション、臨床研究サービスのリーディングプロバイダーであるIQVIAでも、Snowflakeの導入することでこれらのメリットを利用できるようになりました。同社では、処理すべき構造化データ、半構造化データ、非構造化データの量が増加しており、ビジネスの拡張に伴い複雑化する状況を管理する必要に迫られていました。

SnowflakeにSnowparkを導入して以来、同社は行レベルアクセス、データマスキング、データと処理の近接化といった、一貫したエンタープライズレベルのガバナンス機能により、より迅速かつ容易にデータエンジニアリングパイプラインならびにインテリジェントアプリを開発できるようになりました。大量のデータを処理するパイプラインの構築にSnowparkを活用することで、IQVIAのコストはそれまでのパイプラインプロセスと比較して3分の1に削減されました。

SNOWFLAKE（SNOWPARK）によるデータエンジニアリング

Snowparkは、データエンジニアリングのための強力なデベロッパーフレームワークです。Snowparkで作業するデータエンジニアの重要なユースケースには、次のようなものがあります。

- **ETL/ELT：**データチームは、Snowparkを使用することで、JSON、Parquet、XMLなどのタイプを問わず、未加工データをモデリング済みの形式に変換できます。その後、すべてのデータ変換をSnowparkストアドプロシージャとしてパッケージ化し、Snowflakeタスクやその他のオーケストレーションツールでジョブを操作してスケジュールできます。
- **カスタムロジック：**ユーザーは、Snowparkのユーザー定義関数（UDF）を活用して、SQLクエリや変換を実行する同じプラットフォームで、PythonまたはJavaで記述された複雑なデータ処理とカスタムビジネスロジックを備えたアーキテクチャを合理化できます。管理、拡張、運用のための個別のクラスターは必要ありません。
- **データサイエンスとMLパイプライン：**データチームは、統合されたAnacondaレポジトリとパッケージマネージャーを使用して、MLデータパイプラインを実稼働環境に導入する際に共同作業を行うことができます。トレーニングされたMLモデルはUDFとしてパッケージ化され、データの近くでモデル推論を実行できるため、モデル開発から実稼働までのパスを短縮できます。

SNOWFLAKE FOR PYTHON

Snowpark for Pythonを利用することで、データエンジニアは、Snowflakeエンジンのスケール、セキュリティ、パフォーマンスの恩恵を受けながら、使い慣れたツールとプログラミング言語を活用することができます。すべての処理はデータのすぐ横にある安全なPythonサンドボックスで実行されるため、使用言語に関係なく、ビルトインのガバナンスを備えたより高速かつスケーラブルなパイプラインが実現します。図2は、SnowparkクライアントAPIとSnowflakeサーバーサイドランタイムの概要です。

この図は、DataFrame、UDF、ストアドプロシージャで構成されるSnowpark for Pythonアーキテクチャを表しています。これは、どのクライアントIDEまたはノートブックからでも開発することができます。実行されるとすべてSnowflakeにプッシュダウンされ、Snowflake処理エンジンのパフォーマンス、伸縮性、ガバナンスのメリットを享受することができます。

開発内容に応じて、Snowflakeでのコードの実行方法は異なります。まず挙げられるのがDataFrame操作です。これは、フィルター、集計、結合、およびその他類似する操作など、データに対する変換や操作のようなものです。これらのDataFrame操作はSQLに変換され、Snowflakeの定評のある性能を活用することで、当該データの処理を分散し拡張します。

SNOWPARK FOR PYTHONのアーキテクチャ

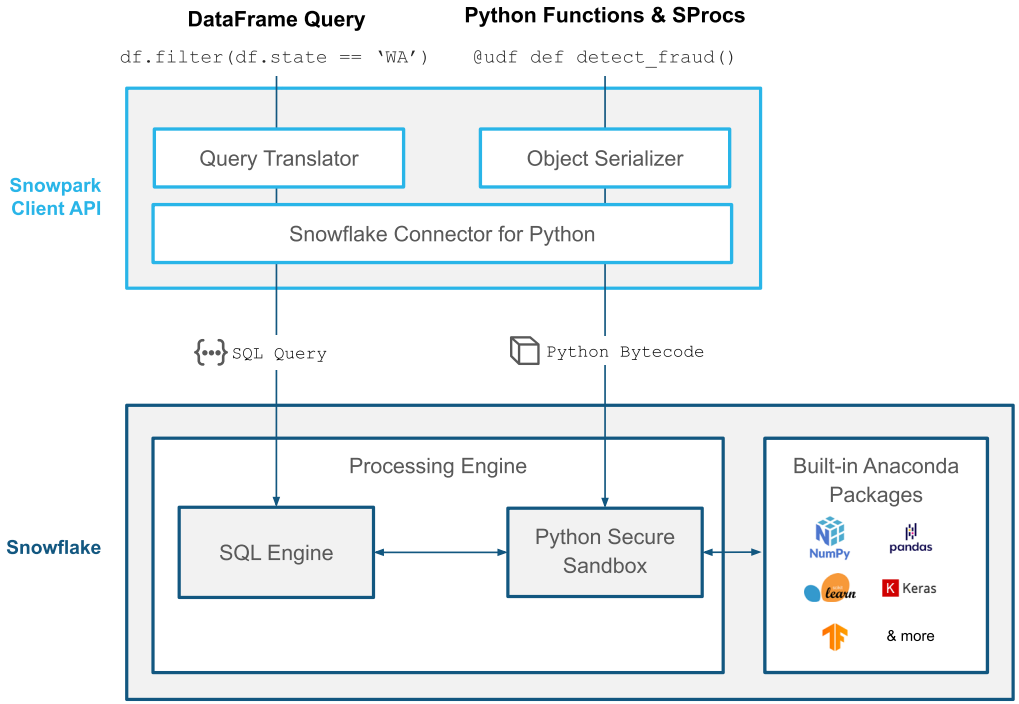

図2：Snowparkデータフレームと Pythonの関数をシームレスに併用可能

PythonまたはJavaのカスタムコードの場合、SQLへの変換はありません。そうではなく、コードがシリアル化されてSnowflakeに送信され、JavaまたはPythonのセキュアなサンドボックス内で処理されます。Pythonの場合、統合されたAnacondaパッケージレポジトリで利用可能なサードパーティのオープンソースライブラリがカスタムコードに含まれていれば、パッケージマネージャーを使うことで、複雑な環境管理をせずコードを確実に実行することができます。

SNOWPARKクライアントAPI

SnowparkクライアントAPIはオープンソースであり、どのPython環境でも動作します。データエンジニアは、Pythonのコード上でDataFrameを使ったクエリを構築できます。この際、SQL文字列の作成や受け渡しは不要です。

SNOWPARKのデータフレーム

Snowparkは、データエンジニアが好んで使用する言語に、深く統合されたDataFrameスタイルのプログラミングを提供します。データエンジニアは、選択したIDEまたは開発ツールを使用し、PythonでのDataFrameスタイルのプログラミングを用いてSnowpark内でクエリを構築できます。バックグラウンドでは、すべてのDataFrame操作がトランスペアレントにSQLクエリに変換され、スケーラブルなSnowflake処理エンジンにプッシュダウンされます。DataFramesではファーストクラスの言語コンストラクトが使用されているため、エンジニアは、開発環境での型のチェック、IntelliSense、エラーレポートなどのサポートを利用することもできます。

SNOWFLAKEプラットフォームアーキテクチャ

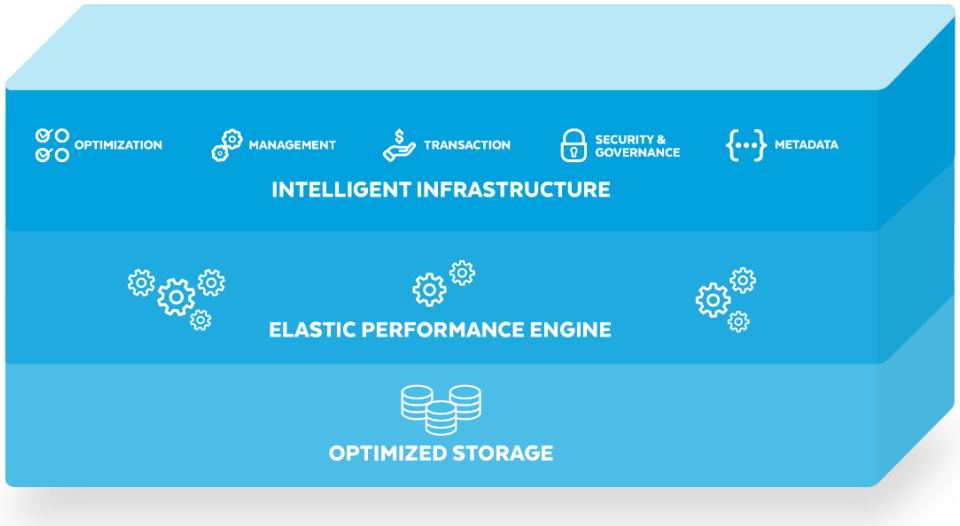

図3：Snowflakeの単一のプラットフォームでは、3つの独自のアーキテクチャレイヤーが統合されている

SNOWFLAKEサーバーサイドランタイム

Snowflakeは、クラウドベースのプラットフォームです。このプラットフォームは、ストレージとコンピュートがアーキテクチャ的には分離されつつ、論理的には統合されており、これらのリソースをほぼ無制限に使用できるよう最適化されています。また、Snowflakeのアーキテクチャは、伸縮性のある拡張、多言語処理、統一されたガバナンスによっても支えられています。

インテリジェントなインフラストラクチャーによって、あらゆるものの適切な機能が実現します。コンピュートクラスタは、自動的またはオンザフライで開始、停止、サイズ変更することができるため、いつでもコンピュートリソースの増減によりニーズに対応することができます。Snowflakeでは、柔軟性と合わせてスピードも優先しており、ワークロードごとに専用のコンピュートクラスタにほぼ瞬時にアクセスできるため、ユーザーは性能を低下させることなく、ほぼ無制限の同時実行性を得ることができます。Snowflakeの単一のプラットフォーム内で統合される3つのアーキテクチャレイヤーを図3に示します。

Snowpark Pythonサーバーサイドランタイムでは、SnowflakeのセキュアなPythonサンドボックスに展開されるPython UDFとストアドプロシージャを記述することができます。UDFとストアドプロシージャの2つもSnowparkのキーコンポーネントであり、これを利用するとデータエンジニアはSnowparkにプリインストールされているオープンソースパッケージを活用しながら、Snowflakeのコンピュートエンジンにカスタム Pythonロジックを持ち込むことができます。

SNOWPARKユーザー定義関数（UDF）

Pythonで書かれたカスタムロジックは、UDFを使用してSnowflakeで直接実行されます。関数はスタンドアロンの場合もあれば、DataFrame操作の一部として呼び出され、データを処理される場合もあります。Snowparkはカスタムコードを Pythonのバイトコードにシリアル化する処理を実行し、すべてのロジックをSnowflakeにプッシュすることで、データに隣接してそれらを実行します。コードをホストする目的で、Snowparkは、Snowflakeエンジンに組み込まれたセキュアでサンドボックス化されたPythonランタイムが有しています。Python UDFは、基盤となるPythonコードに関連する処理をスケールアウトします。これは、すべてのスレッドおよびノードで並行して発生し、関数が実行されている仮想ウェアハウスを構成するものです。

データエンジニアがSnowparkで使用できるUDFには、次のような種類があります。

- **Scalar UDF：**各行に対して単独で演算を行い、単一の結果を生成するもの
- **ベクトル化されたUDF：**入力行のバッチをPandas DataFramesとして受け取り、結果のバッチをPandas配列またはシリーズとして返すもの
- **ユーザー定義テーブル関数：**入力行ごとに複数の行を返す、行グループに対して1つの結果を返す、あるいは複数の行にまたがって状態を維持するもの

流通センターと出荷先の距離を計算するために使用されるSnowpark UDFの例を右に示します。

```
#Given geo-coordinates, UDF to calculate distance between
distribution center and shipping locations


from snowflake.snowpark.functions import udf
import geopandas as gpd
from shapely.geometry import Point


@udf(packages=['geopandas'])
def calculate_distance(lat1: float, long1: float, lat2: float, long2:
float)-> float:
   points_df = gpd.GeoDataFrame({'geometry': [Point(long1, lat1),
Point(long2, lat2)]}, crs='EPSG:4326').to_crs('EPSG:3310')
  return points_df.distance(points_df.shift()).iloc[1]


# Call function on dataframe containing location coordinates
distance_df = loc_df.select(loc_df.sale_id, loc_df.distribution_
center_address, loc_df.shipping_address, \
    calculate_distance(loc_df.distribution_center_lat, loc_
df.distribution_center_lng, loc_df.shipping_lat, loc_df.shipping_
lng) \
   .alias('distribution_center_to_shipping_distance'))
```

ストアドプロシージャ

Snowparkストアドプロシージャは、データエンジニアがPythonコードを運用し、パイプラインの実行、オーケストレーション、スケジューリングを可能にします。ストアドプロシージャは、一度作成しておけば、オーケストレーションあるいは自動化のツールで簡単なCALL文を使うことで、何度でも実行できます。Snowflakeは、SQL、Python、Java、Javascript、Scalaのストアドプロシージャに対応しているため、データエンジニアはポリグロットのパイプラインを簡単に作成することができます。

ストアドプロシージャを使用するために、開発者はクライアントAPI内でSnowparkのsproc()関数を使用してPython関数をバンドルし、Snowparkにサーバーサイドでこれを展開させることができます。Snowparkは、Pythonのコードと依存関係をバイトコードにシリアル化し、それらを自動的にSnowflakeステージに格納します。Snowflakeでは、仮の（セッションレベルでの）あるいは恒久的なオブジェクトとして作成されます。

ストアドプロシージャは単一ノードです。つまり、ストアドプロシージャ内で大規模なデータの変換や分析を行う場合は、クライアントAPIや展開されたその他のUDFを活用して、コンピュートクラスタのすべてのノードにわたってコンピュートを拡張する必要があるということになります。

Snowpark for Pythonのパイプラインを運用し、企業の営業ボーナスを日次ベースで計算する方法を、右の簡単な例に示します。

```
-- Create python stored procedure to host and run the snowpark pipeline
to calculate and apply bonuses
create or replace procedure apply_bonuses(sales_table string, bonus_table
string)
  returns string
  language python
  runtime_version = '3.8'
  packages = ('snowflake-snowpark-python')
  handler = 'apply_bonuses'
AS
$$
from snowflake.snowpark.functions import udf, col
from snowflake.snowpark.types import *

def apply_bonuses(session, sales_table, bonus_table):
   session.table(sales_table).select(col("rep_id"), col("sales_amount")*0.1).
write.save_as_table(bonus_table)
  return "SUCCESS"
$$;

--Call stored procedure to apply bonuses
call apply_bonuses('wholesale_sales','bonuses');

— Query bonuses table to see newly applied bonuses
select * from bonuses;

— Create a task to run the pipeline on a daily basis
create or replace task bonus_task
warehouse = 'xs'
schedule = '1440 minute'
as
call apply_bonuses('wholesale_sales','bonuses');
```

SNOWFLAKE、ANACONDA、オープンソースエコシステム

Pythonのメリットの1つは、オープンソースパッケージやライブラリの豊富なエコシステムにあります。近年、オープンソースパッケージは、データエンジニアリングの高速化および簡便化を支える大きな要因の1つとなっています。オープンソースのイノベーションを活用するために、Snowparkは Anacondaと提携し、ウェアハウスの使用状況を上回る追加コストやユーザーへのライセンス提供なしの製品統合を実現しました。

Snowflakeのデータエンジニアは、Anacondaが提供するシームレスな依存関係管理とキュレーション済みの一連の包括的なオープンソースパッケージを活用することにより、データの移動やコピーを必要とすることなく、Pythonベースのパイプラインを加速できるようになりました。すべてのSnowparkユーザーは、文字列マッチングのためのfuzzy wuzzy、地理空間分析のためのh3、機械学習と予測データ分析のためのscikit-learnなど、Anacondaレポジトリからプリインストールされている何千もの人気のパッケージを活用することができます。さらに、SnowparkはCondaパッケージマネージャーと統合されているため、依存関係の欠如によりPython環境に不具合が生じるといったことも避けられます。

Snowflakeでのオープンソースパッケージの利用は、次に挙げるコードのようにシンプルです。これは、ユーザーがSnowparkからNumPy、XGBoost、Pandasなどのパッケージを直接呼び出す方法を示すものです。

Snowparkは、昨今データ変換のための最も一般的なソリューションの1つであるdbtにも完全に対応しています。dbtは、SQLを優先とした変換ワークフローをサポートしていますが、2022年にPythonのサポートを導入しました。dbtはSQLとPythonの両方をサポートしているため、ユーザーは最も使い慣れた、そして目的に合った言語で変換を記述できます。また、Snowparkのdbtでは、データエンジニアリングやデータサイエンス用の最先端のパッケージなど、オープンソースのPythonエコシステムで利用できるツールを使った分析が可能です。SQLを優先としたワークフローをサポートしていますが、2022年にはdbtにおいて内部でSnowparkを実行する第2言語としてPythonが導入され、オープンソースのPythonエコシステムで利用可能なツールを使用して分析を実行できるようになりました。

```
-- Returns an array of the package versions of NumPy, Pandas, and XGboost
create or replace function py_udf()
returns array
language python
runtime_version = 3.8
packages = ('numpy','pandas==1.4.*','xgboost==1.5.0')
handler = 'udf'
as $$
import numpy as np
import pandas as pd
import xgboost as xgb
def udf():
  return [np.__version__, pd.__version__, xgb.__version__]
$$;
```

ベストプラクティス：PYTHONを使用したSNOWPARKでのデータエンジニアリング

Snowpark for Pythonの開発者コミュニティが急成長する中で、データエンジニアらは自分たちの作業の指針となる「ベストプラクティス」を求めています。Snowpark DataFrame、UDF、ストアドプロシージャの連携の仕組みを把握すると、データエンジニアのSnowflakeでの作業がより効率的かつ安全なものとなります。そこで、SnowparkでPythonを扱うデータエンジニアのためのベストプラクティスを簡単にまとめました。

1. **開発にはSnowparkクライアントを、安全な実行のためにはSnowflakeエンジンを最大限に活用。**

   Snowparkは、好みのIDEや開発およびデバッグツールで使用することができるほか、透明性を保ちながら実行をSnowflakeにプッシュダウンすることもできます。すべてのデータをメモリに取り込むSnowparkクライアントのto_pandas()の使用を念頭に置きつつ、このユーティリティを最大限に活用してください。また、Cachetoolsは、指定した期間にわたって、限られた数のアイテムを保存するための一連のキャッシュアルゴリズムを提供するPythonのライブラリです。読み込みが反復される場合にロジックがメモリにキャッシュされるようにすることで、UDFやストアドプロシージャを高速化するために使用できます。

2. **Anacondaとの統合で開発から本番までのフローを加速。**

   クライアントサイドとサーバーサイドの操作の互換性を確保するため、ローカル開発にはSnowflake Anacondaチャネルを使用することをお勧めします。サードパーティ製パッケージの安定した最新バージョンを使用してコードを構築する場合、Condaパッケージマネージャーによって依存関係が指定されるため、ユーザー側でそれらを指定する必要はなく、非常に安心して作業を行うことができます。ご希望のパッケージがSnowflakeでご利用いただけない場合は、Snowflakeコミュニティを通じてフィードバックをお寄せください。Snowflakeのチームが統合を促進するよう取り組ませていただきます。純粋なPythonパッケージであれば、ブロックを解除し、ステージ経由でパッケージを持ち込むことができます。

3. **特徴量変換やMLスコアリング用にベクトル化されたUDFを使用。**

   Batch APIを使用してベクトル化されたUDFは、scalar UDFを一括して実行できます。サードパーティ製のPythonパッケージを利用し、行ごとに独立して変換を実行し、行を一括して処理することで効率的に処理をスケールアウトできる場合は、Python UDF Batch APIを使用します。これは、サードパーティ製Pythonパッケージを使用し、特徴量エンジニアリングの一環としてデータに対する機械学習特有の変換を実行する場合や、MLのバッチ推論を実行する場合に一般的なシナリオです。

4. **メモリ集約型のワークロードには、Snowparkに最適化されたウェアハウスを使用。**

   Snowparkに最適化されたウェアハウスは、大規模なデータセットを扱うデータエンジニアにとって重要です。開発中に次のエラーが発生したら、Snowparkに最適化されたウェアハウスの使用を検討してください。`100357 (P0000):UDF available memory exhausted`（UDFの使用可能メモリの枯渇）。Snowparkに最適化されたウェアハウスを必要とするワークロードと他のワークロードを混在させないようにしてください。混在させる必要がある場合は、`session.use_warehouse()`メソッドを呼び出し、標準のウェアハウスに戻すことを検討してください。

SNOWPARKの先へ：SNOWFLAKEデータエンジニアリングエコシステムにおけるその他の機能

Snowparkのほかにも、Snowflakeは多くのデータエンジニアリング機能を備えており、好みの言語によるシンプルで信頼性が高いデータパイプラインに総合的に対応できる、高速かつ柔軟なプラットフォームを提供しています。

図4は、データエンジニアリングをシンプル化する取り込み、変換、配信のためのSnowflakeの高度な機能の概要を示すものです。

Snowflakeを利用すると、データエンジニアリングチームは、ストリーミングやバッチ、構造化データ、半構造化データ、非構造化データなど、あらゆる種類のデータを単一のプラットフォームを使用して取り込むことができます。JSON、XML、Avro、Parquet、ORC、Icebergなどのデータ形式に対応しています。現在パブリックプレビュー中のSnowpipeストリーミングでは、Apache Kafkaトピックからのストリームを含むストリーミングデータをSnowflakeテーブルに直接取り込めるようになっています。データクラウドのおかげで、これらのデータはすべて、Snowflakeマーケットプレイスを介して、プロバイダー間や、社内チーム、顧客、パートナー、その他のデータコンシューマー間でアクセスしたり共有したりすることができます。

データは、Snowparkを使用してデータエンジニアが選択した言語を使って変換できます。タスクをテーブルストリームと組み合わせて、連続してELTワークフローを実行し、最近変更されたテーブル行を処理できます。タスクを簡単にチェーン化して連続で実行し、より複雑な定期処理に対応することができます。これらすべてを高速で実行できるほか、複雑なプロジェクトでユーザー数、データ数、ジョブ数が発展しても、それに対応できるよう拡張することができます。Snowflakeは常に機能強化を続けています。

宣言型パイプラインを構築する方法を提供するダイナミックテーブルは、現在プライベートレビュー中ですが、ここでは、Snowpark for PythonのUDFやストアドプロシージャとは異なるアプローチでパイプラインを構築することが可能です。これらのツールは、データエンジニアリングのワークロードをシンプル化するために、データの変更に応じて自動的にデータを段階的に処理していくよう設計されています。Snowflakeは、データベースオブジェクトとデータ操作言語の管理をすべて自動化し、データエンジニアが拡張可能で高性能、かつコスト効率の高いデータパイプラインを簡単に構築できるようにしています。

その結果、データパイプラインにはインテリジェントなインフラストラクチャー、パイプラインオートメーション、データプログラマビリティが搭載されることになりました。Snowflakeのシンプル化されたパイプラインは、分析、アプリケーション、MLモデルを強化しつつ、管理対象のデータのコピーは1つのみとなるため、メンテナンスはほぼ不要となります。また、セキュアデータシェアリング機能を使用して社内チーム、顧客、パートナーと直接データのアクセスや共有を実現できるほか、Snowflakeマーケットプレイスを通じてより多くのデータプロバイダーやコンシューマーとのデータアクセスや共有も可能となります。Snowflakeの最新のデータシェアリングテクノロジーでは、データは移動されません。その代わり、データプロバイダーはデータコンシューマーに、読み取り専用のライブデータコピーにほぼ瞬時のアクセスを許可します。このアプローチにより、レイテンシーが軽減され、古くなったデータをコピーして移動する必要がなくなり、同じデータの複数のコピーを管理することによるガバナンスの問題点が大幅に減少します。

Snowflakeは、データエンジニアがすべてのデータに素早くアクセスし、性能と信頼性を高め、根本的にシンプルなデータパイプラインを構築できるように設計されています。革新的なパイプラインオートメーションとデータプログラマビリティにより、データエンジニアはワークフローをシンプル化して不要なものを排除することで、自分が最も大きなインパクトを与えられる仕事に労力を集中させることができます。

SNOWFLAKEを利用したデータエンジニアリング

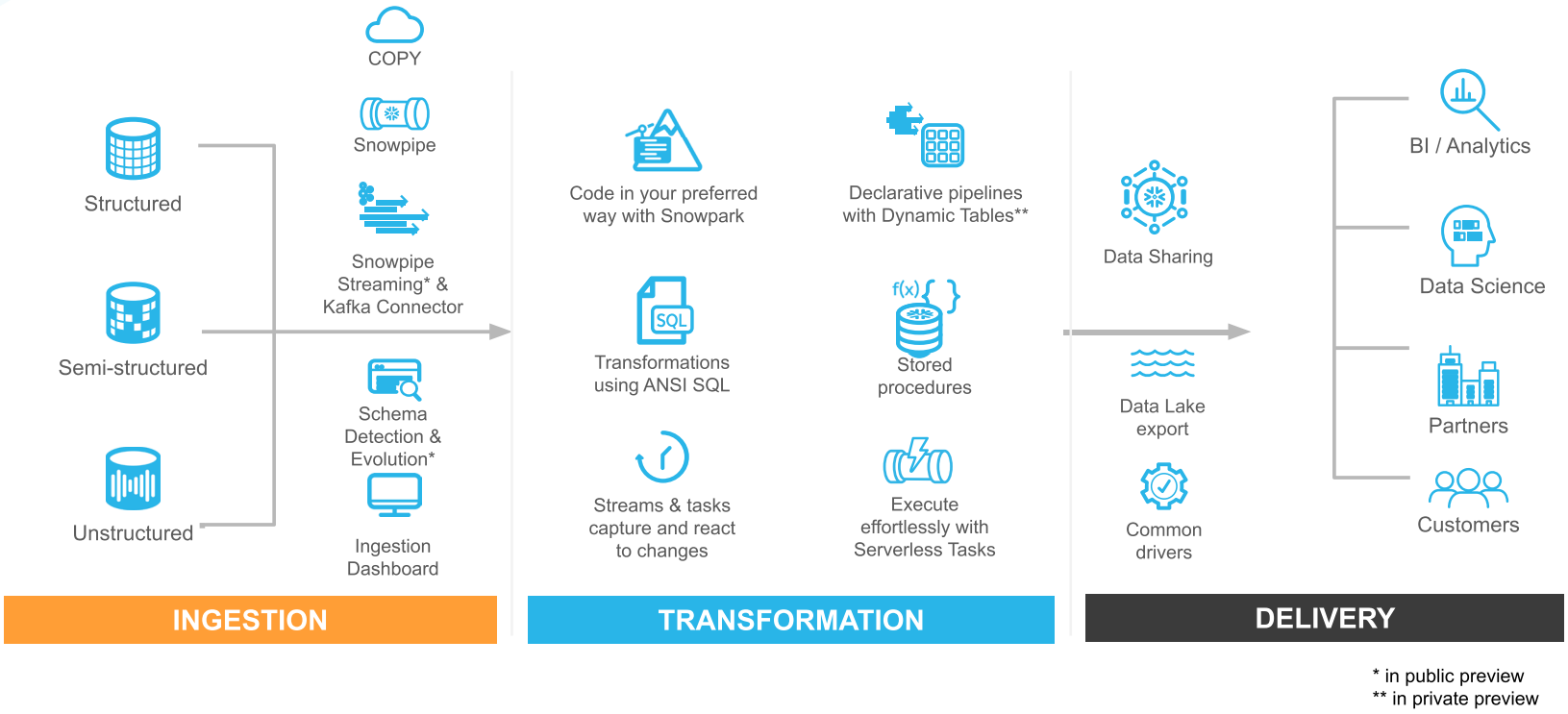

図4：Snowflakeは、非構造化データ、半構造化データ、構造化データの取り込みに対応しており、自動ワークフローによって変換と配信を促進

SNOWPARK 入門

Snowparkでコードを開発し展開するために、開発者は常に自分が気に入っている統合開発環境（IDE）やノートブックから柔軟に作業できるようになっています。

データエンジニアは、Pythonカーネルを実行できる場所であれば、どこでも、簡単にSnowparkでの開発に着手できます。データエンジニアはSnowparkクライアントAPIをインストールし、Snowflakeアカウントへの接続を確立するだけでよく、新しいツールは必要ないため、学習曲線が最小化されます。

Snowparkは開発者に柔軟性を提供することを目指しており、次のような多くの開発インターフェイスに対応しています。

- **コードエディタおよびIDE：**多くのデータエンジニアは、構築にコードエディタやIDEを利用しています。ここでは、ローカルデバッグ、オートコンプリート、ソースコントロールとの統合などの機能を利用できます。Snowparkは、VSコード、IntelliJ、PyCharmなどのツールで問題なく機能します。VSコードは、エディタ内でノートブックエクスペリエンスを提供するJupyter拡張機能と連携し、ノートブックエクスペリエンスにブレークポイントやデバッグを取り入れることができます。この際、Jupyterコンテナやランタイムを別途管理する必要はありません。コードエディタやIDEは、パイプライン構築のための豊富な開発およびテストのための最適な選択肢です。
- **Snowsightワークシート：**Snowsightは、SQLとPython（現在パブリックプレビュー中）に対応し、統一された使いやすいエクスペリエンスを提供するSnowflakeのウェブインターフェイスです。これらのワークシートは、Snowparkセッションのオートコンプリートを提供し、ストアドプロシージャとしてブラウザから直接実行できます。Snowsightは、Snowparkを記述して実行するインストール不要のエディタを探しているチームに最適なオプションです。ここではコードが、自動化されたパイプラインの一部としてオーケストレーションできるストアドプロシージャに素早く変換されます。
- **オープンソースのノートブックソリューション：**Snowparkでパイプラインを構築するための一般的な選択肢の1つが、ノートブックを活用することです。ノートブックでは、セルを使った迅速な実験が可能です。Snowparkでは、Jupyter Notebooksなど、さまざまなノートブックソリューションを実行できます。これは、Snowflakeにセキュアに接続しながらローカルで動作するものであり、データオペレーションの実行が可能です。コンテナやPythonが動作するマシンであれば、Snowparkパイプラインを構築し、実行することができます。Apache Zeppelinなどその他のノートブックソリューションでSnowparkを扱う場合にも、同様のアプローチが可能です。オープンソースのノートブックソリューションは、データ調査のための素晴らしい選択肢です。
- **パートナー統合ソリューション：**Snowpark Acceleratedのパートナーの多くは、ホスト型のオープンソースノートブックや独自の統合エクスペリエンスを提供しています。それらのソリューションには、すぐに使えるSnowpark APIがプリインストールされており、安全にデータ接続できます。これらの密接な統合により、パイプライン、モデル、アプリの構築と展開が高速化されます。パートナー統合の詳細については、Snowpark Accelerated ページをご覧ください。

リソース

データエンジニアリングのためのSnowparkでぜひSnowflakeのパワーを活用してください。開始に際しては、以下のリソースをご参照ください。

- **無料トライアル**
- **クイックスタート**
- **開発者向けドキュメント**
- **メディアブログ**
- **SNOWFLAKEフォーラム**

SNOWFLAKEについて

Snowflakeは、Snowflakeのデータクラウドを用い、あらゆる組織が自らのデータを活用できるようにします。顧客企業はデータクラウドを利用してサイロ化されたデータを統合し、データを検索して安全に共有しながら、さまざまな分析ワークロードを実行しています。データやユーザーがどこに存在するかに関係なく、Snowflakeは複数のクラウドと地域にまたがり単一のデータ体験を提供します。多くの業界から何千ものお客様（2023年4月30日時点で、2022年のForbes Global 2000社（G2K）のうち590社を含む）が、Snowflakeデータクラウドを全社で幅広いビジネスに活用しています。

詳しくは、**snowflake.com**をご覧ください

引用元

[1] https://insights.stackoverflow.com/survey