MASTERCLASS: USING PYTHON IN SNOWFLAKE VIA SNOWPARK

John DaCosta
Sales Engineer

Joe Burns
Sales Engineer

Daniel Lacouture
Sales Engineer

Agenda

1. Snowflake recap in < 3 mins
2. Why did we create Snowpark?
3. Snowpark Architecture
4. Snowpark Demo

Want to follow along with us?
https://github.com/NickAkincilar/Sample_Snowpark_Demos

WHY SNOWFLAKE?

FAST FOR ANY WORKLOAD

Run any number or type of job across all users and data volumes quickly and reliably.

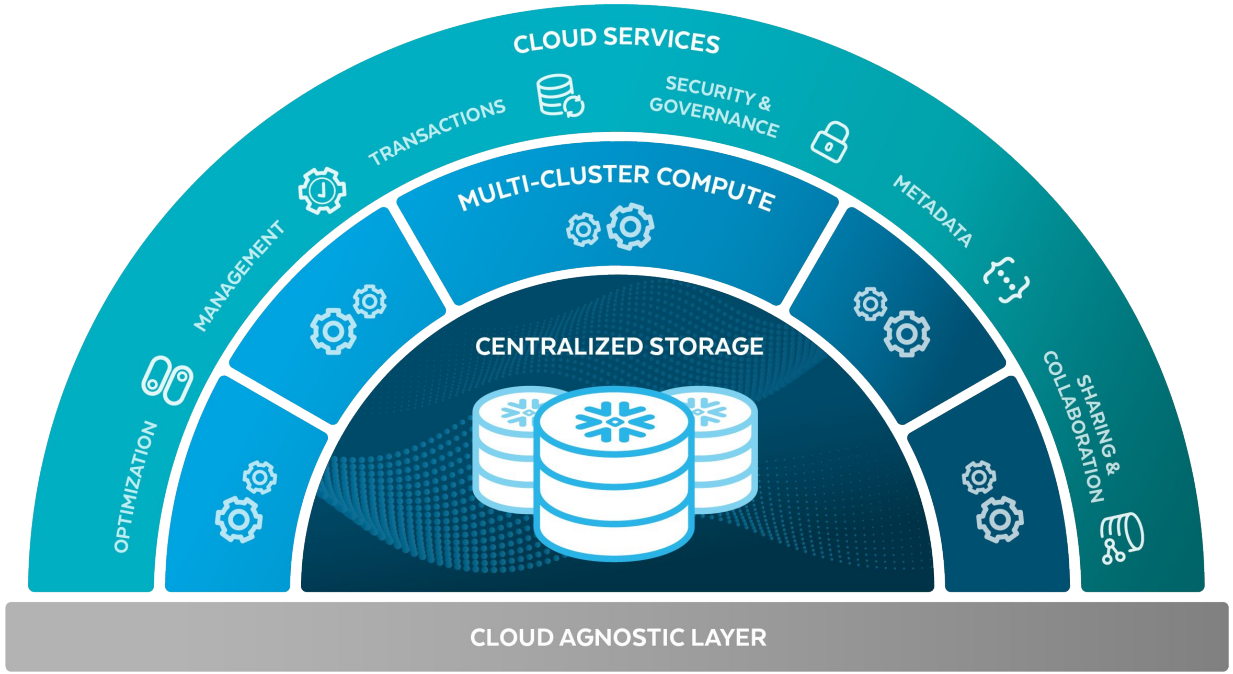
IT JUST WORKS

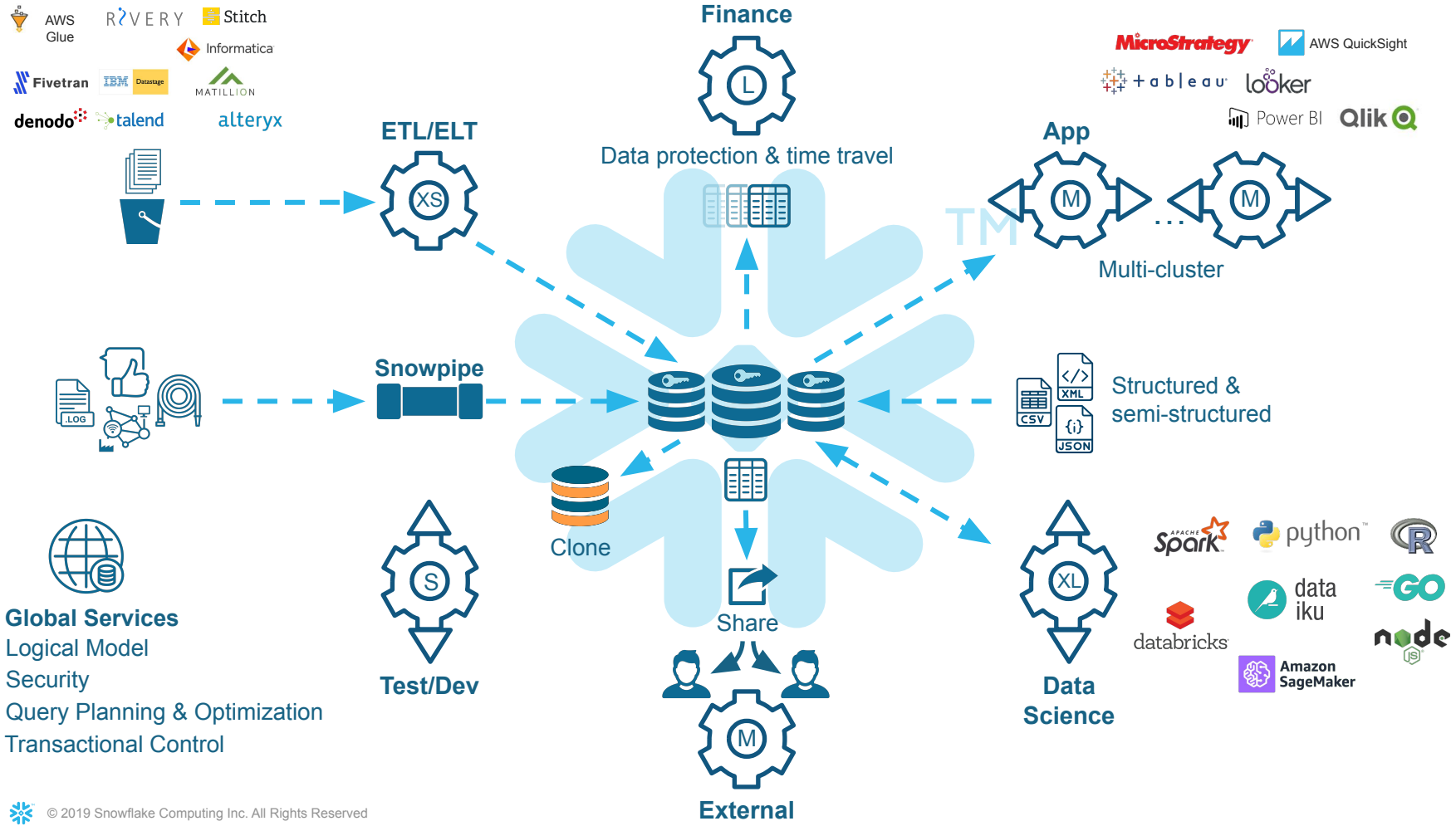
Replace manual with automated to operate at scale, optimize costs, and minimize downtime.

CONNECTED TO WHAT MATTERS

Extend access and collaboration across teams, workloads, clouds, and data, seamlessly and securely.

SNOWFLAKE ARCHITECTURE





WHY DID WE CREATE SNOWPARK?

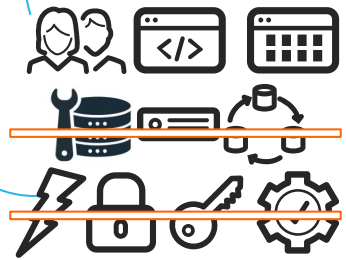
SPARK
2014 - 2023

BIG DATA
2010 - 2023

snowflake®
2012 - 2015 - 2023
$10B in FY'29

Data Engineers

Python | Scala | Java



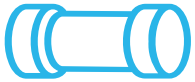
SQL

Managed Platform

BENEFITS OF MIGRATING TO SNOWPARK

Language of Choice

Enable all data users to bring their work to a single platform with **native python, java, scala and SQL support**



No Governance Trade-offs

Non-SQL workloads now run within consistent controls **trusted by over 500 of the Forbes Global 2000**

Faster & Cheaper Pipelines

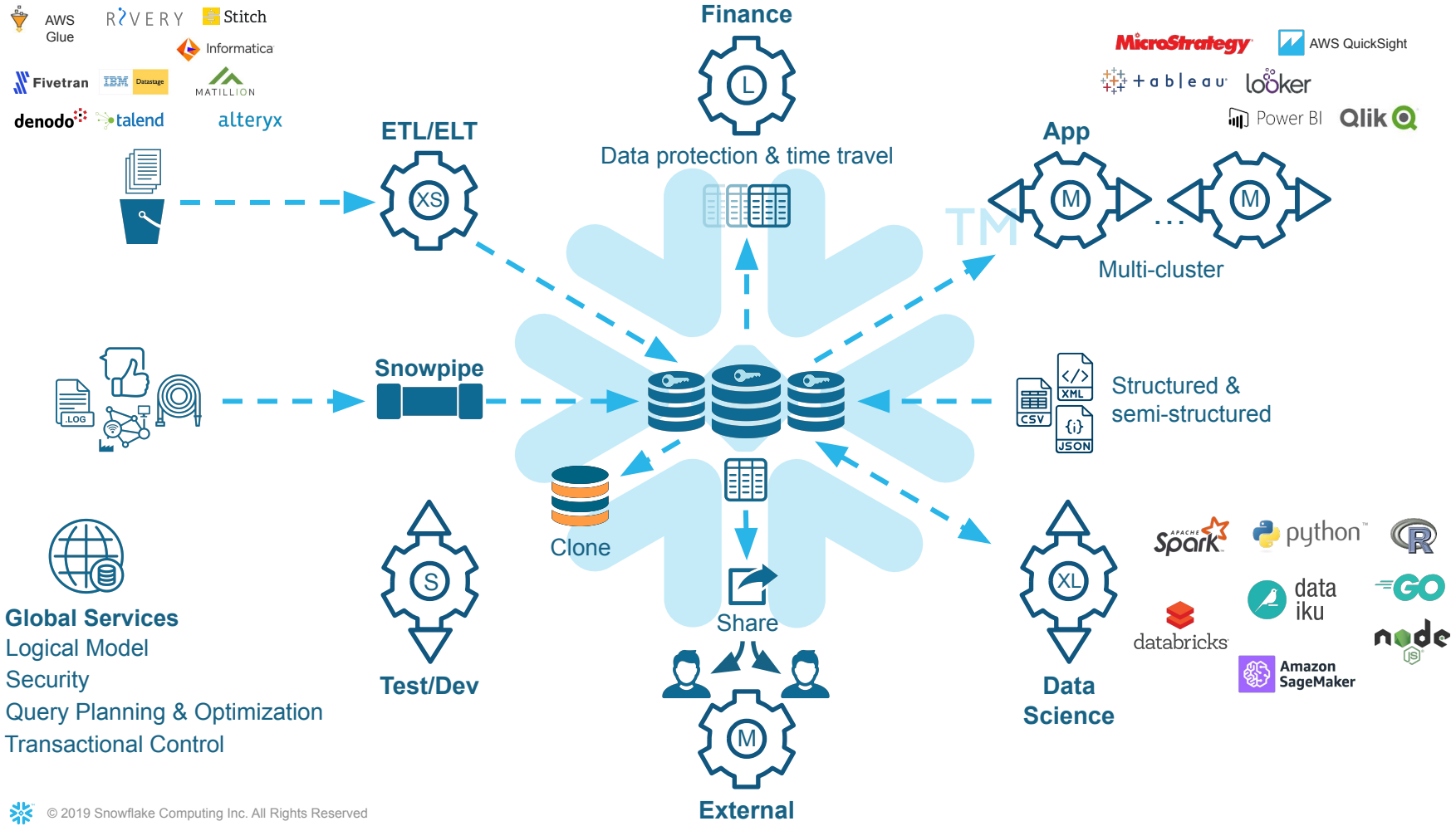
Migrate existing Spark pipelines with **minimal code change**, **better price-performance**, transparent cost and less ops overhead

One platform for all workloads

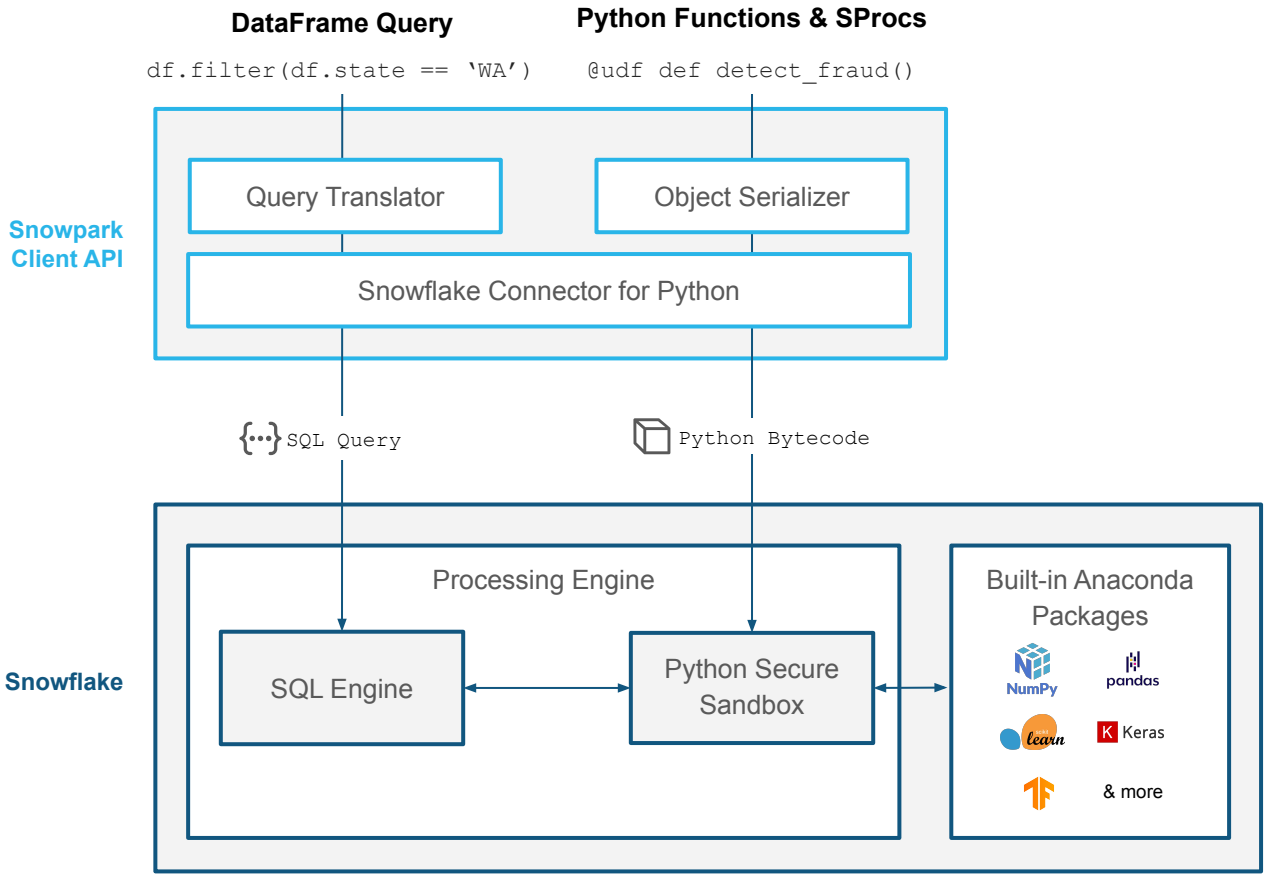
Business units **scale to meet their own needs without impacting** or being impacted by other business units

Customers transitioning from Spark-based pipelines are reporting that Snowpark is up to 2-3x faster at 30-50% of the cost.

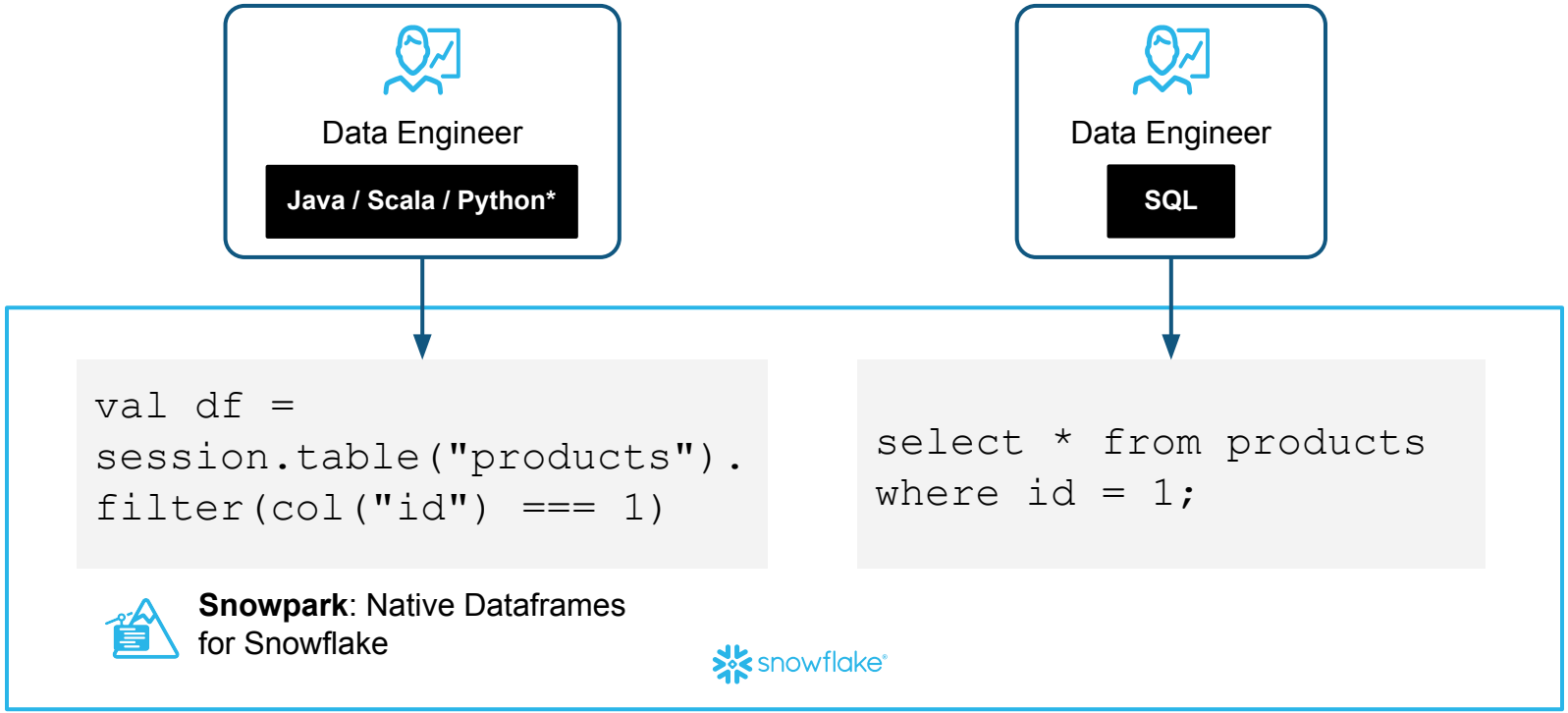
TELL ME MORE ABOUT SNOWPARK



Snowpark for Python



How it Works



How This Works Inside a Function

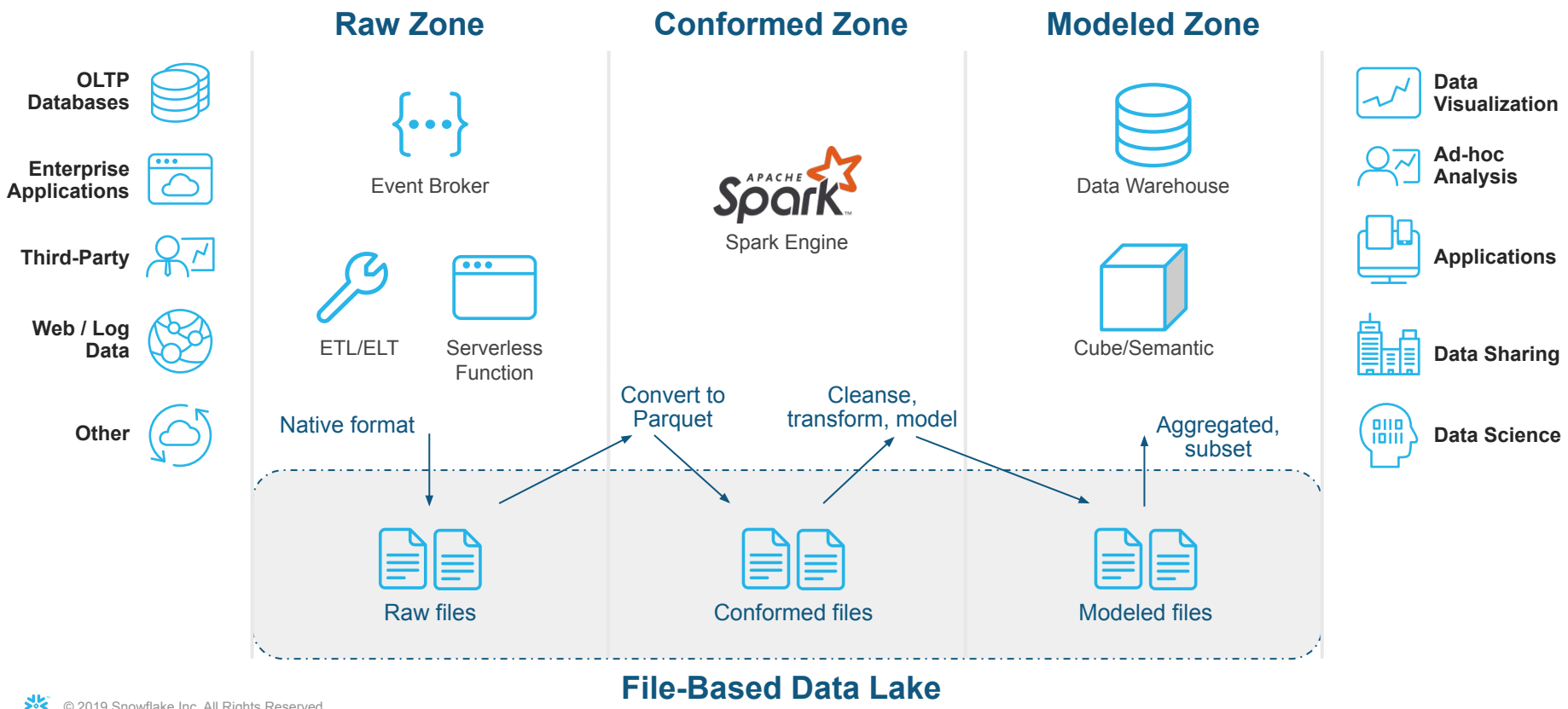
"SQL"

Python

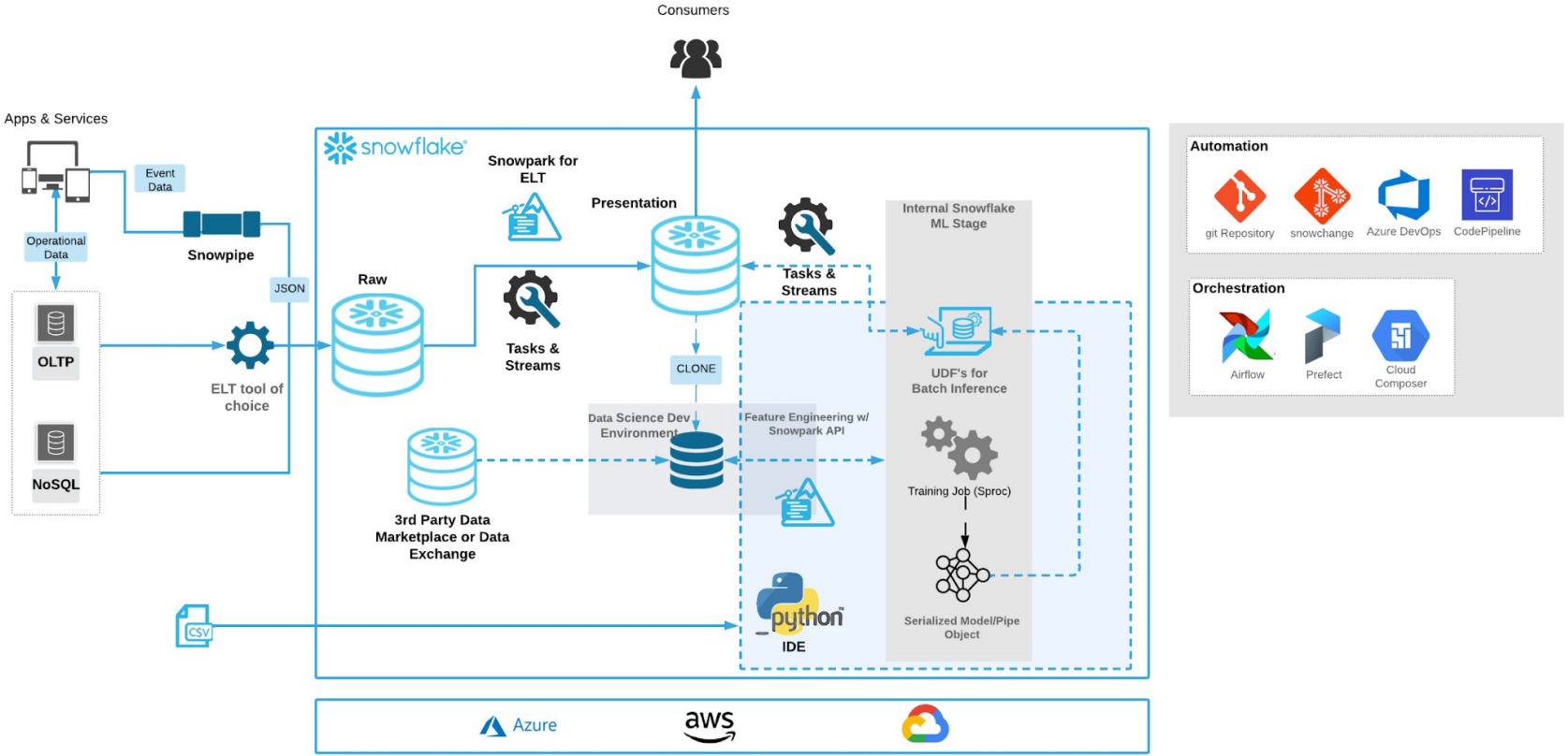
"SQL"

```
snow_df_spend = session.table('campaign_spend')

snow_df_spend_per_channel = snow_df_spend.group_by(year('DATE'), month('DATE'),'CHANNEL').agg(sum('TOTAL_COST').as_('TOTAL_COST')).\
    with_column_renamed('"YEAR(DATE)"',"YEAR").with_column_renamed('"MONTH(DATE)"',"MONTH").sort('YEAR','MONTH')

snow_df_spend_per_month = snow_df_spend_per_channel.pivot('CHANNEL',['search_engine','social_media','video','email']).sum('TOTAL_COST').sort('YEAR','MONTH')

snow_df_revenue = session.table('monthly_revenue')
snow_df_revenue_per_month = snow_df_revenue.group_by('YEAR','MONTH').agg(sum('REVENUE')).sort('YEAR','MONTH').with_column_renamed('SUM(REVENUE)','REVENUE')

snow_df_spend_and_revenue_per_month = snow_df_spend_per_month.join(snow_df_revenue_per_month, ["YEAR","MONTH"])

snow_df_spend_and_revenue_per_month_clean = snow_df_spend_and_revenue_per_month.dropna()

snow_df_spend_and_revenue_per_month_clean = snow_df_spend_and_revenue_per_month_clean.drop(['YEAR','MONTH'])

# Load features
df = snow_df_spend_and_revenue_per_month_clean.to_pandas()

# Preprocess the Numeric columns
# We apply PolynomialFeatures and StandardScaler preprocessing steps to the numeric columns
# NOTE: High degrees can cause overfitting.
numeric_features = ['SEARCH_ENGINE','SOCIAL_MEDIA','VIDEO','EMAIL']
numeric_transformer = Pipeline(steps=[('poly',PolynomialFeatures(degree = polynomial_features_degrees)),('scaler', StandardScaler())])

# Combine the preprocessed step together using the Column Transformer module
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features)])

# The next step is the integrate the features we just preprocessed with our Machine Learning algorithm to enable us to build a model
pipeline = Pipeline(steps=[('preprocessor', preprocessor),('classifier', LinearRegression())])
parameteres = {}

X = df.drop('REVENUE', axis = 1)
y = df['REVENUE']

# Split dataset into training and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 42)

# Use GridSearch to find the best fitting model based on number_of_folds folds
model = GridSearchCV(pipeline, param_grid=parameteres, cv=number_of_folds)

model.fit(X_train, y_train)
train_r2_score = model.score(X_train, y_train)
test_r2_score = model.score(X_test, y_test)

model_output_dir = '/tmp'
model_file = os.path.join(model_output_dir, 'model.joblib')
dump(model, model_file)
session.file.put(model_file, "@demo_models",overwrite=True)

# Save features into a Snowflake table call MARKETING_BUDGETS_FEATURES
snow_df_spend_and_revenue_per_month_clean.write.mode('overwrite').save_as_table('MARKETING_BUDGETS_FEATURES')
```

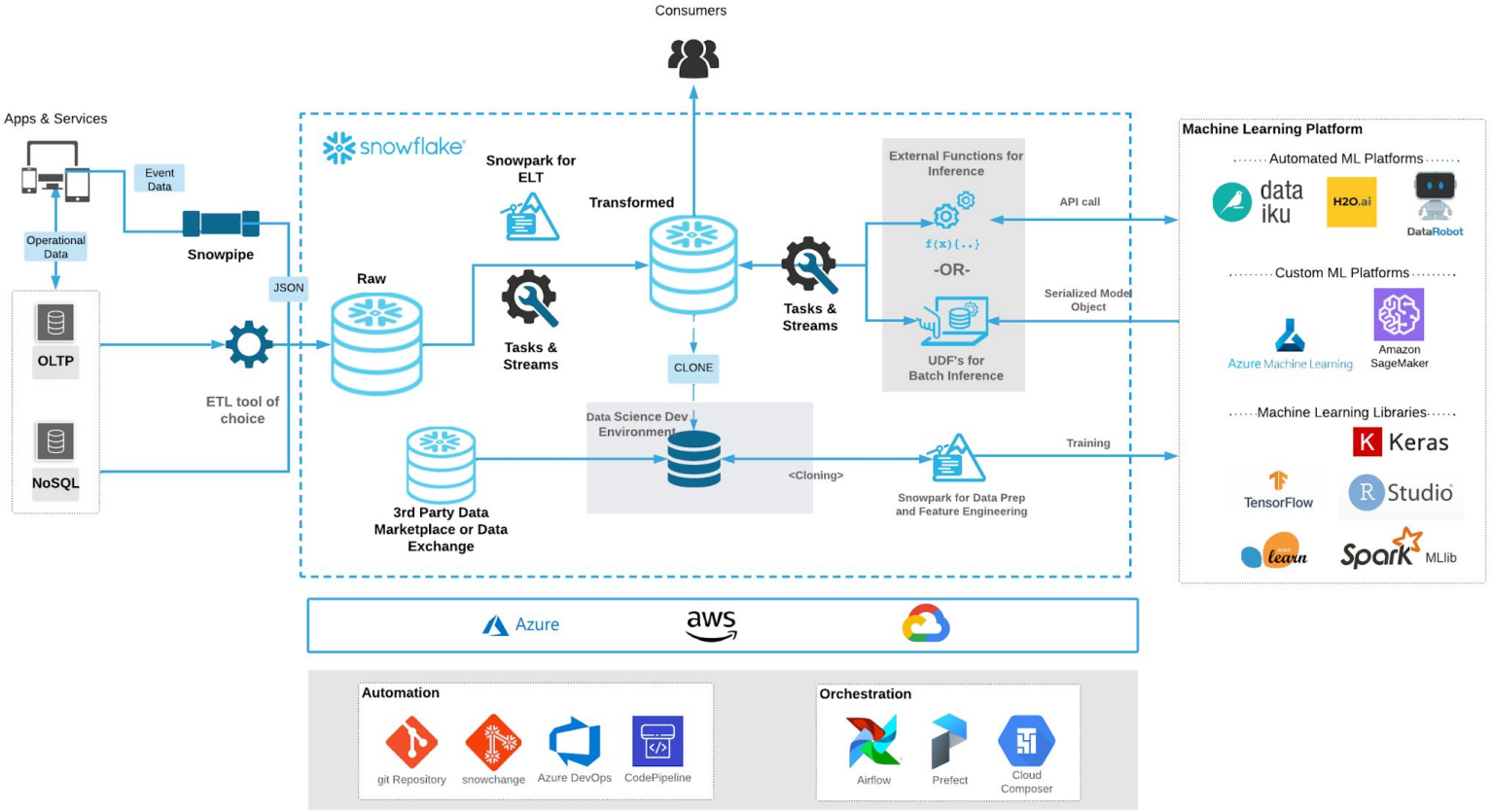
“MODERN” ARCHITECTURE (BEFORE)



End to End SQL & Non-SQL Workloads

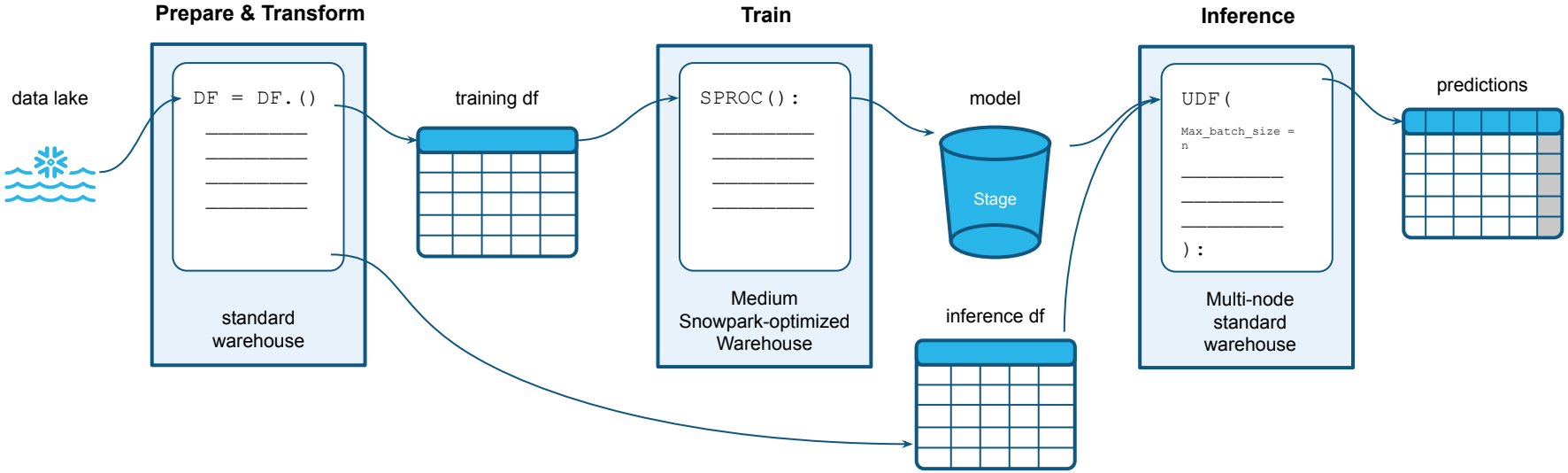


Integration with Existing ML Stack



Example End-to-End Machine Learning

Effortless, scalable and secure processing without data movement across compute environments



ML Training Tip:
Use UDTFs to train multiple models in parallel using M-4XL Snowpark-optimized WH

ML Inference Tip:
Speed up inference using Vectorized UDFs to process rows in batches & cachetools to cache model load from stage

Snowpark for Python

Client API

DataFrame queries / transforms and submit UDFs / Stored Procs for execution.

UDFs

Execute custom Python code, **including OS packages**, in Snowflake secure Python sandbox.

Vectorized UDFs

Pandas dataframe batch processing of vectorized functions (e.g. model inference).

UDTFs: Table Functions

Non 1:1 transformations with custom partitioning guaranteeing contiguous batches.

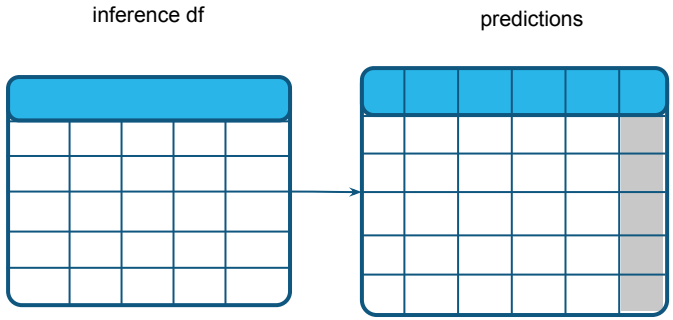
Stored Procs

Host and operationalize Python code and/or Snowpark API calls. Single node bounded.

UDF: User Defined Function

A Deeper Dive

- Takes each row as an array
- Returns a scalar value
- Stacks scalar values into a result column

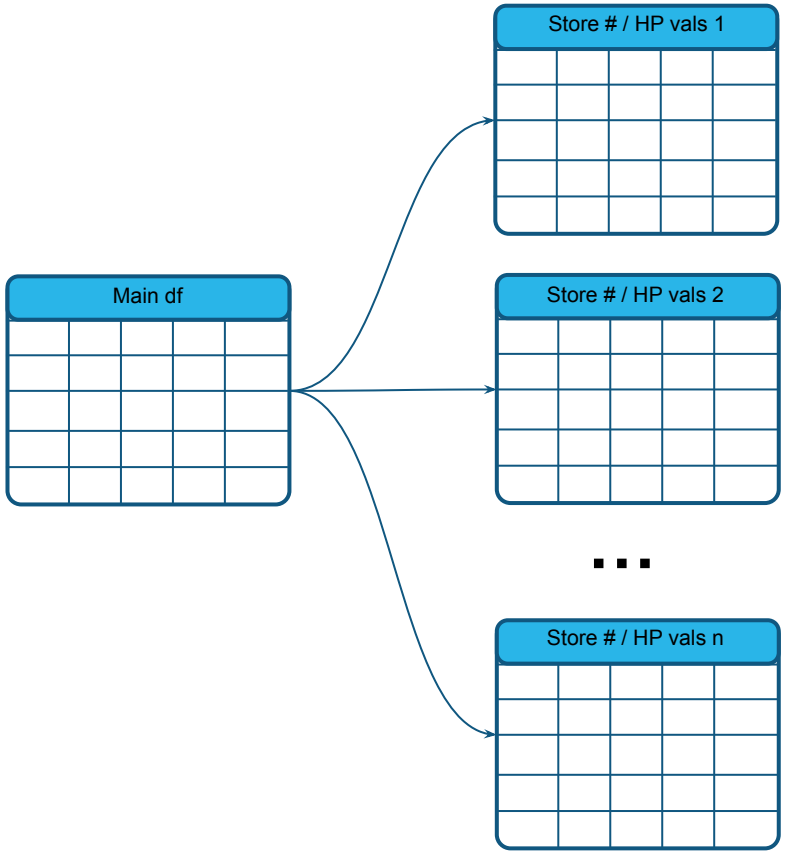


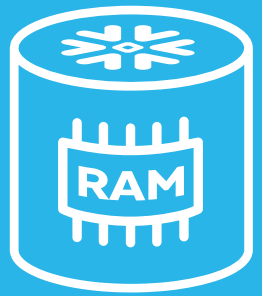

```
# defining featurelist
features = ['C_BIRTH_YEAR', 'CA_ZIP', 'CD_GENDER', 'CD_MARITAL_STATUS', 'CD_CREDIT_RATING', 'CD_EDUCATION_STATUS', 'CD_DEP_COUNT']

# creating the udf on snowflake. The @ symbol will take whatever function follows and attribute it to the udf.
@F.pandas_udf(session=session, max_batch_size=10000, is_permanent=True, stage_location='@ml_models', name="clv_xgboost_udf", replace=True)

# naming and defining the UDF as predict
def predict(df:  T.PandasDataFrame[int, str, str, str, str, str, int]) -> T.PandasSeries[float]:
        m = read_file('model.joblib.gz')
        # m = read_file("@mlmodels/model.joblib.gz")

        df.columns = features

        return m.predict(df)
```

UDTF: User Defined Table Function

A Deeper Dive

- Partition the data across nodes for parallelization
- A stored procedure can be run in parallel on each node
- Parallel HPT, Time-series forecasting, etc





Snowpark-optimized warehouses

16X
memory

10X
cache

Effortless execution of memory-intensive operations

Bring ML training, in-memory analytics (e.g. correlations) or other memory-intensive operations inside Snowflake's secure Python/Java sandbox.

Accelerate subsequent run execution

Provide speedup when cached artifacts (Python packages, intermediate results, JARs, etc) are reused on subsequent runs

Plus all the benefits of standard virtual warehouses

Pricing

	Virtual Warehouses Credits per Hour									
Warehouse Types	**XS**	**S**	**M**	**L**	**XL**	**2XL**	**3XL**	**4XL**	**5XL**	**6XL**
Standard Warehouse	1	2	4	8	16	32	64	128	256	512
Snowpark-Optimized WH	N/A	N/A	**6**	**12**	**24**	**48**	**96**	**192**	**384**	**786**

SNOWPARK DEMO OVERVIEW

SNOWPARK DEMO

WHAT

Snowpark for Data Engineering
DataFrames pushdown to Snowflake

Snowpark for Data Science
Running 3rd party libraries on Snowflake

WHY

Overall: Security, Speed, and Simplicity

1. Simplify tech stack
2. Reduce extra data hops
3. Jumpstart Snowpark projects

HOW

Public Notebook available on Github

Data Engineering
Uses sample TPC Data

Data Science Sentiment Analysis
Uses public Amazon review data

SNOWPARK DEMO

SNOWPARK DEMO RECAP

WHAT

Snowpark for Data Engineering
DataFrames pushdown to Snowflake

Snowpark for Data Science
Running 3rd party libraries on Snowflake

WHY

Overall: Security, Speed, and Simplicity

1. Simplify tech stack
2. Reduce extra data hops
3. Jumpstart Snowpark projects

HOW

Public Notebook available on Github

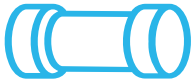
Data Engineering
Uses sample TPC Data

Data Science Sentiment Analysis
Uses public Amazon review data

BENEFITS OF MIGRATING TO SNOWPARK

Language of Choice

Enable all data users to bring their work to a single platform with **native python, java, scala and SQL support**



No Governance Trade-offs

Non-SQL workloads now run within consistent controls **trusted by over 500 of the Forbes Global 2000**

Faster & Cheaper Pipelines

Migrate existing Spark pipelines with **minimal code change**, **better price-performance**, transparent cost and less ops overhead

One platform for all workloads

Business units **scale to meet their own needs without impacting** or being impacted by other business units

Customers transitioning from Spark-based pipelines are reporting that Snowpark is up to 2-3x faster at 30-50% of the cost.

SNOWFLAKE

FAST FOR ANY WORKLOAD

Run any number or type of job across all users and data volumes quickly and reliably.

IT JUST WORKS

Replace manual with automated to operate at scale, optimize costs, and minimize downtime.

CONNECTED TO WHAT MATTERS

Extend access and collaboration across teams, workloads, clouds, and data, seamlessly and securely.

SQL **PYTHON** **SCALA** **JAVA**

Next Steps

- Code Analysis and Assessment
- POC Scoping and Plan
- Test Snowpark
  - Performance
  - Total Cost of Ownership
  - Ease of Use
  - Simplicity and Efficiency
  - Governance & Security
- Migration Readiness Assessment
  - Migration Discovery Workshop
  - Project Plan
  - Level of Effort estimates and scoping
- Hands on Quickstarts
  - Getting Started With Snowpark for Python and Streamlit
  - Data Engineering Pipelines with Snowpark Python
- Best Practices Webinar
  - https://www.snowflake.com/webinar/thought-leadership/best-practices-for-python-data-transformations-in-snowflake-2023-05-23/