
📘 Snowpark – Full Guide (Beginner to Advanced)

📌 Table of Contents

1️⃣ What is Snowpark?

Snowpark is a developer framework that lets you use **Python, Java, or Scala** to process data **directly inside Snowflake**, eliminating data movement while enhancing scalability, performance, and security.

2️⃣ Architecture Overview

- **Client APIs** in Python/Java/Scala
- **Code pushed to Snowflake Virtual Warehouse**
- **Transformations executed within the data warehouse**

This enables:

- Minimal latency
- Reduced data transfer costs
- Full leverage of Snowflake's compute power

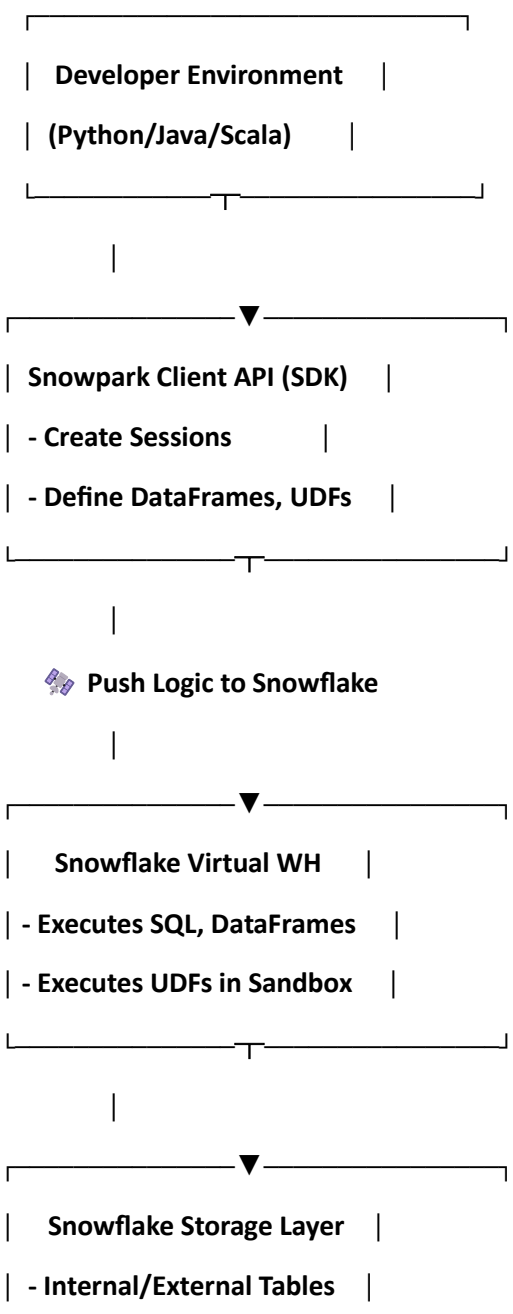
📐 Snowpark Architecture (Detailed)

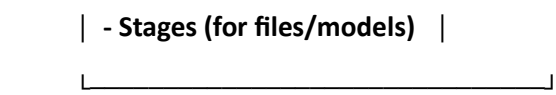
🎯 Goal of Snowpark Architecture

Enable developers to write and execute data processing and ML logic in languages like Python, Scala, and Java, while ensuring:

- **No data movement**
- **In-database execution**
- **Scalability & security**

📊 High-Level Architecture





🔄 Data Movement (Or Lack Thereof)

- ❌ Do not pull data to the client like in Pandas/Spark
- ✅ Push your code to Snowflake via Snowpark
- ✅ Logic is optimized and executed *within* the warehouse

🔐 Where Execution Happens

Component	Executed in
DataFrame logic	Virtual Warehouse
SQL queries	Query Optimizer + Engine
Python UDFs	Isolated sandbox (Python 3.8)
ML Model inference	In UDF, inside warehouse

⚙️ Compute + Storage Separation

Snowflake handles:

- Storage: Managed, scalable, compressed
- Compute: Virtual warehouses, elastic
- Metadata: Immutable, audit-ready

🧱 Core Concepts in Snowpark (Detailed)

1️⃣ Session

Establishes connection to Snowflake and maintains context for:

- **DataFrame creation**
- **UDF registration**
- **Stage/file access**

```
from snowflake.snowpark import Session


session = Session.builder.configs({...}).create()
```

2️⃣ DataFrames

Like Pandas/Spark, but executed in Snowflake.

📌 Properties:

- **Immutable**
- **Lazy evaluation**
- **Composable**

💡 Example:

```
df = session.create_dataframe([(1, 'A'), (2, 'B')], schema=["id", "label"])

filtered_df = df.filter(df["id"] > 1)
```

3️⃣ Functions

Use SQL-style and Snowpark-specific functions:

```
from snowflake.snowpark.functions import col, upper


df.select(upper(col("label"))).show()
```

4️⃣ UDFs (User Defined Functions)

Encapsulate logic like a Python function and run securely within Snowflake.

```
from snowflake.snowpark.functions import udf


@udf
def multiply(x: int) -> int:
    return x * 2
```

Can also load ML models:

```
@udf
def predict(area: float) -> float:
    import joblib
    model = joblib.load("/tmp/model.pkl")
    return model.predict([[area]])[0]
```

5️⃣ Stored Procedures

Use for reusable multi-step logic, similar to SQL stored procedures.

```
from snowflake.snowpark.stored_procedure import stored_procedure


@stored_procedure(name="increment_value")
def increment(session: Session, x: int) -> int:
    return x + 1
```

6️⃣ Stages

Used for loading/saving:

- **CSVs**
- **Model files**
- **Scripts**

```
snowsql -q "PUT file://model.pkl @my_stage"
session.file.get("@my_stage/model.pkl", "./")
```

7️⃣ File Formats & Data Loading

Define formats and use them in reads:

```
session.sql("CREATE FILE FORMAT my_csv TYPE = 'CSV'
FIELD_OPTIONALLY_ENCLOSED_BY='\"'").collect()


df = session.read.option("FORMAT_NAME", "my_csv").csv("@my_stage/myfile.csv")
```

🧠 Model Inference with Snowpark

📍 **Workflow:**

1. **Train model externally (e.g., scikit-learn)**
2. **Save with joblib**
3. **Upload to stage**
4. **Load and use in UDF**

```
@udf(name="predict_price")
def predict_price(area: float) -> float:
    import joblib
    model = joblib.load("/tmp/house_model.pkl")
    return float(model.predict([[area]])[0])
```

Call:

```
SELECT area, predict_price(area) FROM houses;
```

🔄 Lazy Evaluation

- **Code like df.filter(...).select(...) doesn't run immediately**
- **Runs only on .collect(), .show(), or .write**

This enables query plan optimization by Snowflake's engine.

🛡️ Execution Security

- **UDFs run in secure Python sandbox**
- **Can't access network**
- **Can only access uploaded files in stages**
- **Resource quotas managed by warehouse sizing**

4️⃣ Working with Snowpark – Basics

⚙️ Setup

```
pip install snowflake-snowpark-python
```

🔦 Connect

```
from snowflake.snowpark import Session


session = Session.builder.configs({
  "account": "xyz",
  "user": "abc",
  "password": "...",
  "role": "SYSADMIN",
  "warehouse": "COMPUTE_WH",
  "database": "DEMO_DB",
  "schema": "PUBLIC"
}).create()
```

📋 Create DataFrame

```
data = [("Alice", 80), ("Bob", 90)]
df = session.create_dataframe(data, schema=["Name", "Score"])
df.show()
```

🔎 Filter, Sort, Aggregate

```
df.filter(df["Score"] > 85).sort("Score", ascending=False).show()
```

5️⃣ Snowpark – Advanced Concepts

🧠 UDFs (User Defined Functions)

```
from snowflake.snowpark.functions import udf


@udf
def square(x: int) -> int:
```

```
    return x * x
```

Apply UDF:

```
df.with_column("Squared", square(df["Score"])).show()
```

⚙️ **Stored Procedures**

```
from snowflake.snowpark.stored_procedure import stored_procedure


@stored_procedure
def increment_score(session: Session, score: int) -> int:
    return score + 1
```

📦 **Working with Stages (File Upload)**

```
session.file.put("iris.csv", "@my_stage", auto_compress=False)
df = session.read.option("header", True).csv("@my_stage/iris.csv")
df.show()
```

🧠 **Machine Learning with Snowpark**

✅ **6.1. Use Case: Predicting House Prices with Snowpark & scikit-learn**

Step 1: Train ML Model (outside Snowflake)

```
import joblib
from sklearn.linear_model import LinearRegression
import pandas as pd


df = pd.read_csv("houses.csv")
model = LinearRegression().fit(df[["area"]], df["price"])
joblib.dump(model, "house_model.pkl")
```

Step 2: Upload Model to Snowflake Stage

```
snowsql -q "PUT file://house_model.pkl @ml_stage"
```

Step 3: Define UDF for Prediction

```
import joblib
import pandas as pd
```

```
def predict_price(area: float) -> float:
    model = joblib.load("/tmp/model/house_model.pkl")
    return model.predict([[area]])[0]
```

Register UDF in Snowflake:

```
from snowflake.snowpark.functions import udf

@udf(name="predict_price", replace=True)
def predict_udf(area: float) -> float:
    return predict_price(area)
```

Use UDF:

```
SELECT area, predict_price(area) AS predicted_price FROM houses;
```

✅ 6.2. ML Tools You Can Integrate

Tool	Purpose
scikit-learn	Model training/scoring
XGBoost	Advanced ML
ONNX / TensorFlow	Model deployment inside Snowflake
Snowflake ML (Beta)	In-database ML training

✅ 6.3. Model Registry with Stages

- Save model as .pkl, .joblib, or .onnx
- Use PUT to upload to internal/external stage
- Load model during UDF execution

7️⃣ Deployment Strategies

Strategy	Description
UDF + Snowpark	Lightweight predictions
Stored Procedure	Model orchestration or batch scoring

Strategy	Description
Streamlit + Snowflake	Real-time dashboard interface
Airflow + Snowflake	ML pipeline automation

8️⃣ Security & Governance

- Role-Based Access Control (RBAC)
- Access Control on Stages
- Masking Policies
- Data Retention + Lineage
- Execution monitoring via Query History

9️⃣ Real-World Scenarios

Scenario	Snowpark Role
Churn Prediction	Train outside, predict inside with UDF
Credit Risk Analysis	Complex data transformation, scoring via Snowpark
Retail Forecasting	ML pipeline using Python & Snowpark
Healthcare Analytics	Anonymized in-place processing using masking + UDF

🔟 Best Practices

- ✅ Use DataFrames instead of raw SQL for logic separation
- ✅ Minimize UDF logic – use native functions when possible
- ✅ Test locally, deploy remotely
- ✅ Use external stages for model storage
- ✅ Always validate model accuracy before in-database scoring

---

🔗 Resources

- Official Snowpark Docs (Python)
- Snowflake Quickstarts
- Snowflake Blog

- Snowpark for ML – YouTube

✅ **Summary**

Feature	Use
Snowpark	Run logic inside Snowflake
DataFrame	Transform data securely
UDFs	Custom logic in Python/Java
ML Integration	Predictive analytics without moving data
Deployment	Store models, create ML pipelines, secure inference

---