Snowpark for Python Meets Streamlit

Train, Deploy, and Run a ML model using Python, Snowpark and Streamlit

Meet Dash

Dash Desai, Snowflake

Senior Developer Advocate

Technical Evangelist – Snowpark

dash.desai@snowflake.com

Follow me: @iamontheinet

DISCLAIMER

Other than statements of historical fact, all information contained in the presentations and accompanying oral commentary made available as part of this event (collectively, the "Materials"), including statements regarding (i) Snowflake's business strategy and plans, (ii) Snowflake's new or enhanced products, services, and technology offerings, including those that are under development, (iii) market size and growth, trends, and competitive considerations, and (iv) the integration, interoperability, and availability of our products with and on third-party platforms, are forward-looking statements. These forward-looking statements are subject to a number of risks, uncertainties and assumptions, including those described under the heading "Risk Factors" and elsewhere in the Quarterly Reports on Form 10-Q and Annual Reports on Form 10-K that Snowflake files with the Securities and Exchange Commission. In light of these risks, uncertainties, and assumptions, the future events and trends discussed in the Materials may not occur, and actual results could differ materially and adversely from those anticipated or implied in the forward-looking statements. As a result, you should not rely on any forwarding-looking statements as predictions of future events.

Any future product or roadmap information (collectively, the "Roadmap") is intended to outline general product direction; is not a commitment, promise, or legal obligation for Snowflake to deliver any future products, features, or functionality; and is not intended to be, and shall not be deemed to be, incorporated into any contract. The actual timing of any product, feature, or functionality that is ultimately made available may be different from what is presented in the Roadmap. The Roadmap information should not be used when making a purchasing decision. Further, note that Snowflake has made no determination as to whether separate fees will be charged for any future products, features, and/or functionality which may ultimately be made available.

The Materials may contain information provided by third-parties, including those participating in this event. Snowflake has not independently verified this information, and usage of this information does not mean or imply that Snowflake has adopted this information as its own or independently verified its accuracy.

AGENDA

- Snowpark and Snowpark For Python
  - Intro to Snowpark
  - Snowpark DataFrame API
  - Snowpark Python Functions
  - Snowflake + Anaconda
  - Train & deploy ML model with Snowpark For Python
  - Turn ML into actionable insights with Streamlit
- Demo
- DIY: Hands-On Workshop
- [Advanced] Snowpark For Python: Under the Hood

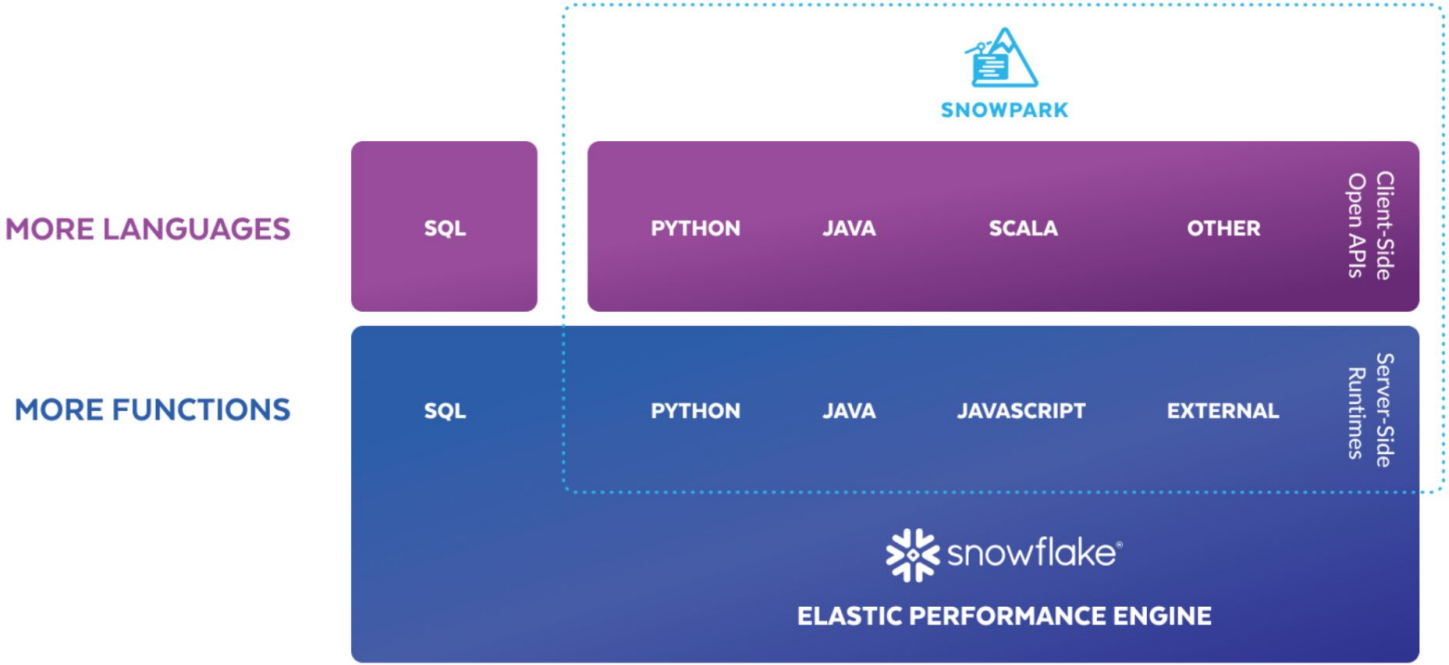
DIY: HANDS-ON WORKSHOP

Train, Deploy, and Run a ML model using Python, Snowpark and Streamlit

Step-by-Step Guide On GitHub

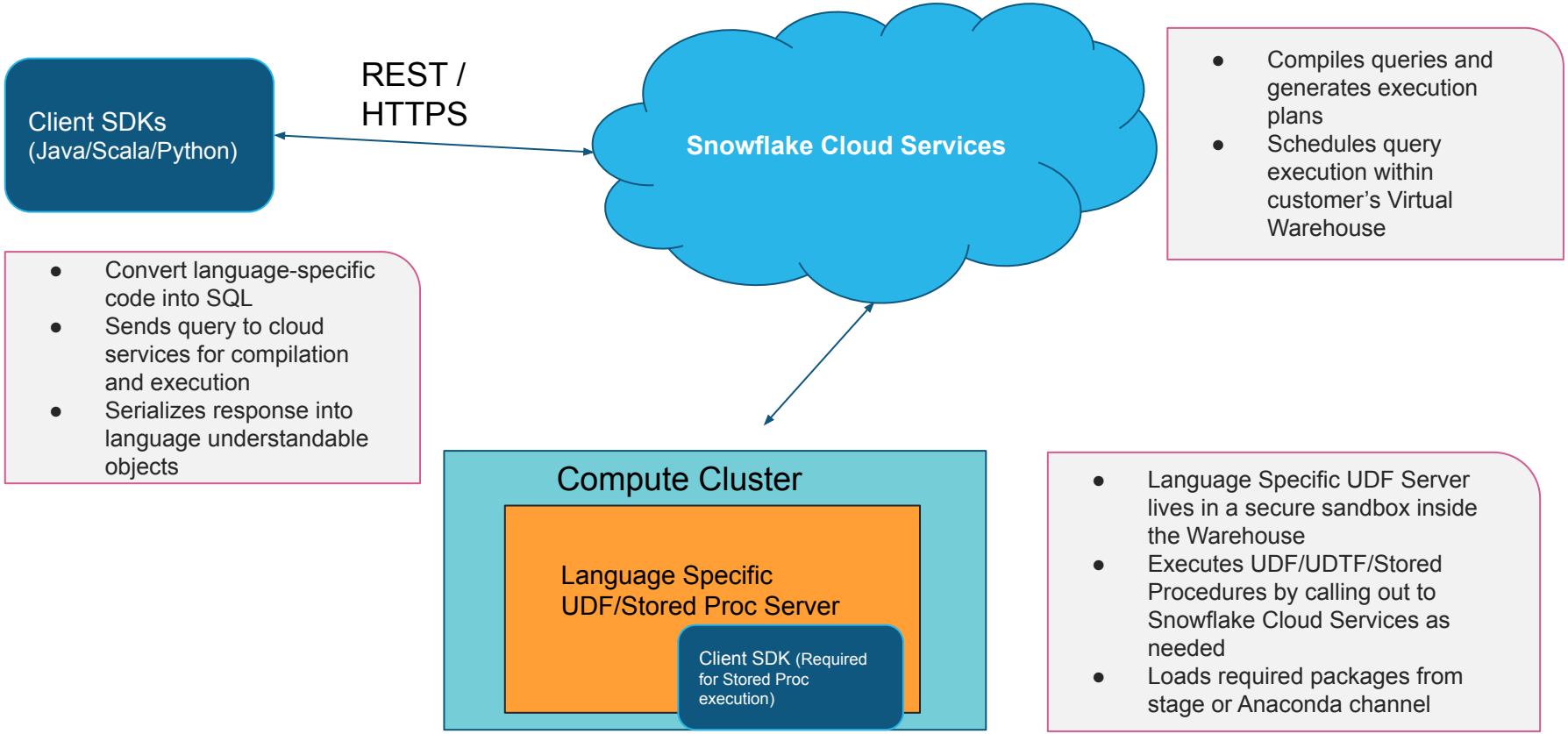
https://bit.ly/DashSnowparkPython

CODE THE SAME WAY, EXECUTE FASTER WITH SNOWPARK



* Snowpark for Python in Public Preview

SNOWPARK COMPONENTS



- Compiles queries and generates execution plans
- Schedules query execution within customer's Virtual Warehouse

- Convert language-specific code into SQL
- Sends query to cloud services for compilation and execution
- Serializes response into language understandable objects

- Language Specific UDF Server lives in a secure sandbox inside the Warehouse
- Executes UDF/UDTF/Stored Procedures by calling out to Snowflake Cloud Services as needed
- Loads required packages from stage or Anaconda channel

SNOWPARK CLIENT SDKs

- Client libraries developed by Snowflake
- Exposes APIs to build data applications, data pipelines and workflows in Java, Scala and Python* (*currently in Public Preview*)
- Applications can be run anywhere with connectivity to Snowflake

**Snowpark Python Client is Open Source – contributions are welcome :)*

SNOWPARK FOR PYTHON

Familiar programming constructs

Use familiar syntax with DataFrame abstraction – distributed computation using Snowflake's elastic engine

Rich ecosystem

Easy access to thousands of packages pre-installed with automated dependency management

Secure processing

100% pushdown in a highly secure, sandboxed environment

* Snowpark for Python in Public Preview

HOW DOES IT ALL WORK

DataFrames

- Are internally represented by a SQL query, or a sequence of SQL queries and statements
- As DataFrames are composed, new queries are generated

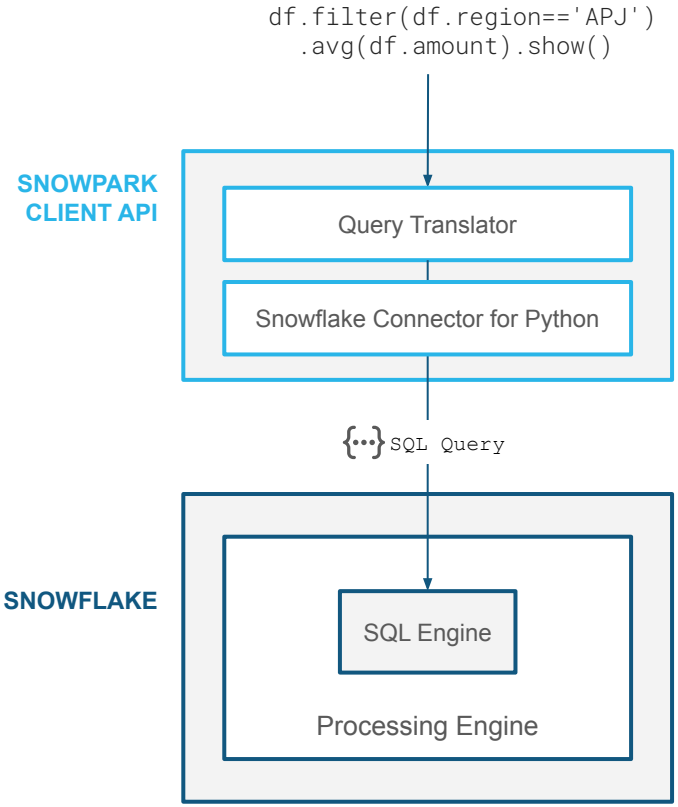
When an action is invoked

- Generated SQL statements are sent to your Snowflake warehouse for execution
- Results are returned

Functions

- Functions and lambda closures are serialized on the client
- Uploaded to a temporary Snowflake stage
- Specify dependencies available in Anaconda Snowflake Channel
- A Python UDF is created from the serialized code and dependencies
- Generated UDF is used in the SQL corresponding to a DataFrame

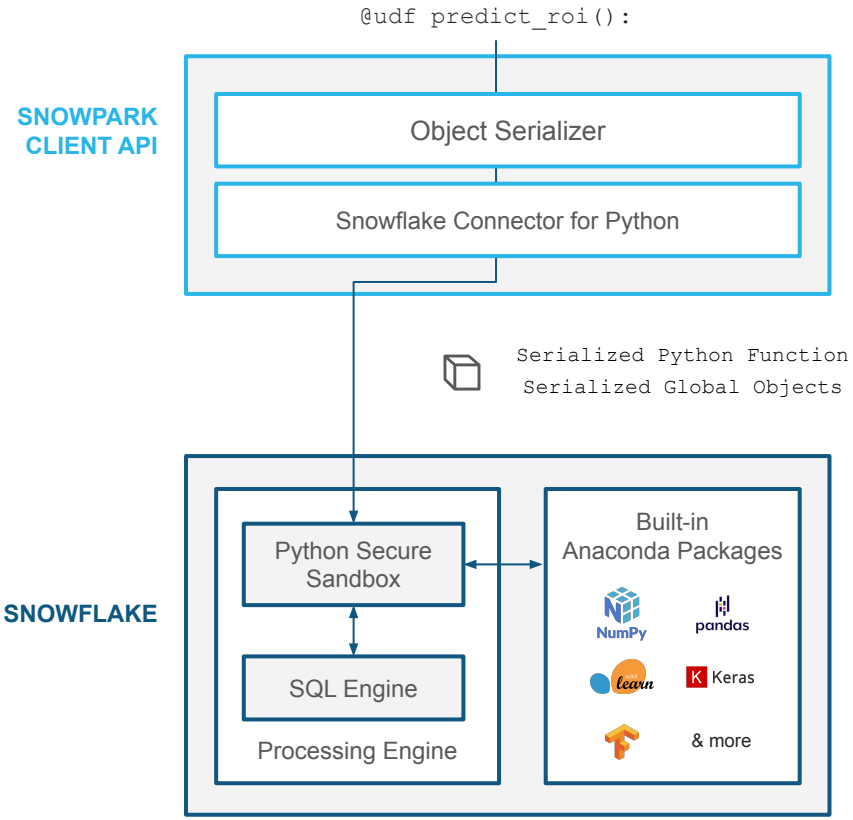
DataFrame API Query



DataFrame API

- Query Snowflake data with Python
- Familiar DataFrame API
- 100% push-down to Snowflake
- Native Snowflake performance and scale

Python Functions



Python Functions

- Bring custom Python code to Snowflake as User Defined Functions (UDFs)
- Code is serialized and pushed down to run in a secure sandboxed environment
- Seamlessly access third-party packages with Anaconda integration

FROM SNOWPARK TO SQL

The SQL generated from Snowpark can be found in the Query History section of Snowsight

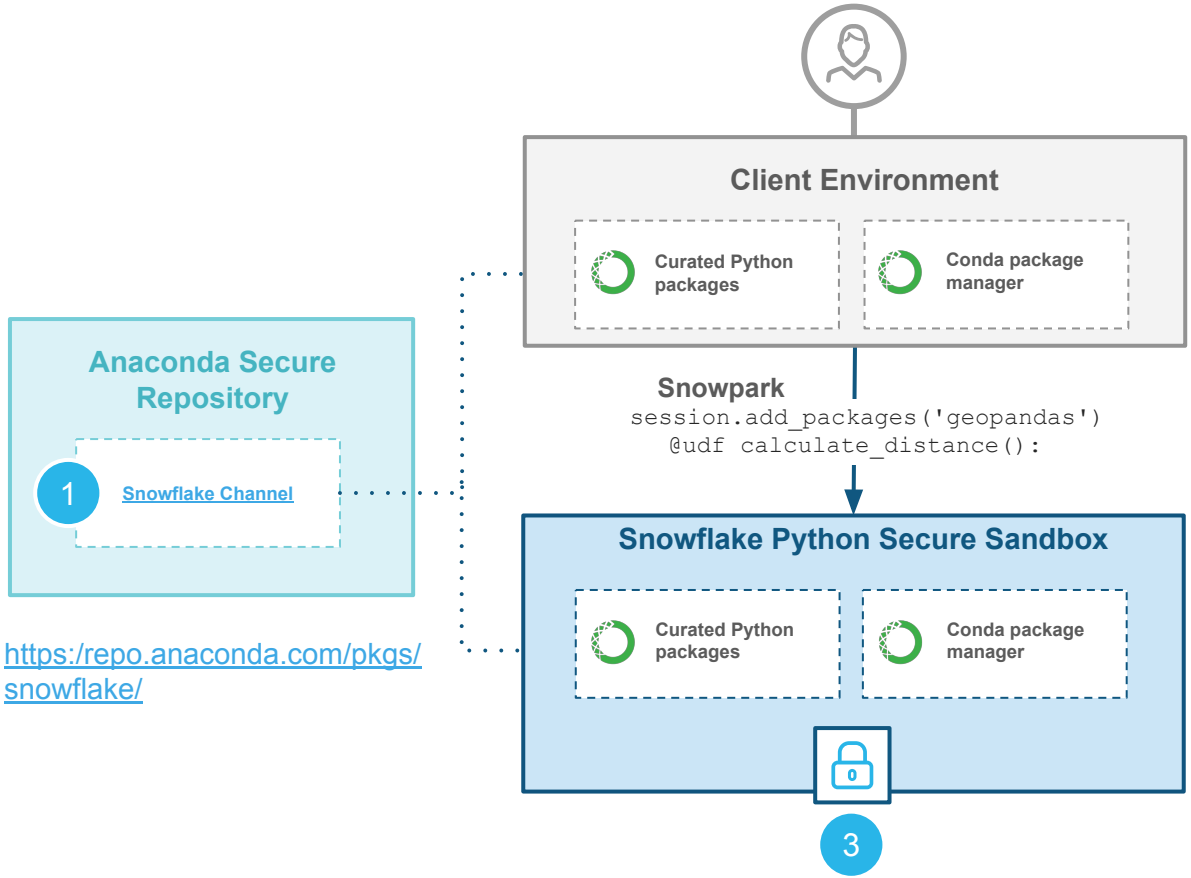
Snowpark Code	Generated SQL
`df = session.table("sample_product_data")` `df.show() # Triggers Evaluation`	`SELECT * FROM ( SELECT * FROM (sample_product_data))` `LIMIT 10`

`df = session.table("sample_product_data")\`
`.filter(col("category_id").startswith("hello"));`
`df.collect() # Triggers evaluation`
	`SELECT * FROM ( SELECT * FROM ( SELECT * FROM`
	`(sample_product_data)) WHERE startswith("CATEGORY_ID",`
	`'hello'))`

`@udf(name="mod5_udf")`
`def mod5(value: int) -> int:`
`  return value % 5`
	`CREATE TEMPORARY FUNCTION mod5_udf(arg1 BIGINT)`
	`RETURNS BIGINT LANGUAGE PYTHON RUNTIME_VERSION=3.8`
	`PACKAGES=('cloudpickle==2.0.0') HANDLER='compute'`
	`AS $$`
	`import pickle`
	`func = pickle.loads(bytes.fromhex('<byte code>'))`
	`$$`

`df = session.create_dataframe([6, 7, 8],`
`schema=["v"])`
`df.select(mod5(df.v)).collect()`
	`SELECT mod5_udf("V") FROM ( SELECT "V" FROM ( SELECT *`
	`FROM ( VALUES (6 :: bigint), (7 :: bigint), (8 :: bigint)`
	`AS SNOWPARK_TEMP_TABLE_MUOQERZJ7X("V"))))`

SNOWFLAKE + ANACONDA



1 **Easy Access**
Curated packages pre-installed in Snowflake also available for local development

2 **No Dependency Hell**
Conda package manager integrated in Snowflake secure sandbox

3 **Scalable and Secure**
Process with secure sandbox integrated into Snowflake processing engine

All of this with no additional charges beyond warehouse usage

SEARCHING ANACONDA PACKAGES

- Query the INFORMATION_SCHEMA.PACKAGES view to see available packages
- For example, to see the latest* available versions of numpy, pandas, and xgboost:

```
SELECT package_name, MAX(version) AS latest_version
FROM INFORMATION_SCHEMA.PACKAGES
WHERE language = 'python' AND
  package_name IN ('numpy', 'pandas', 'xgboost')
GROUP BY package_name
ORDER BY 1;
```

Row	PACKAGE_NAME	VERSION
1	numpy	1.21.5
2	pandas	1.4.1
3	xgboost	1.5.0

*As of the time of this writing.

DEPENDENCY MANAGEMENT

Use packages in Anaconda

Use session.add_packages(), or packages=[...].

UDF server will install packages to the Python runtime in the sandbox.

You can request new packages to Anaconda Snowflake Channel.

```
session.add_packages(["numpy==1.2"])

session.udf.register(add_nums,
packages=["numpy==1.2"])
```

Bring your own Python files

Use session.add_import(), or imports=[...].

Python files and directories are zipped and sent to a stage.

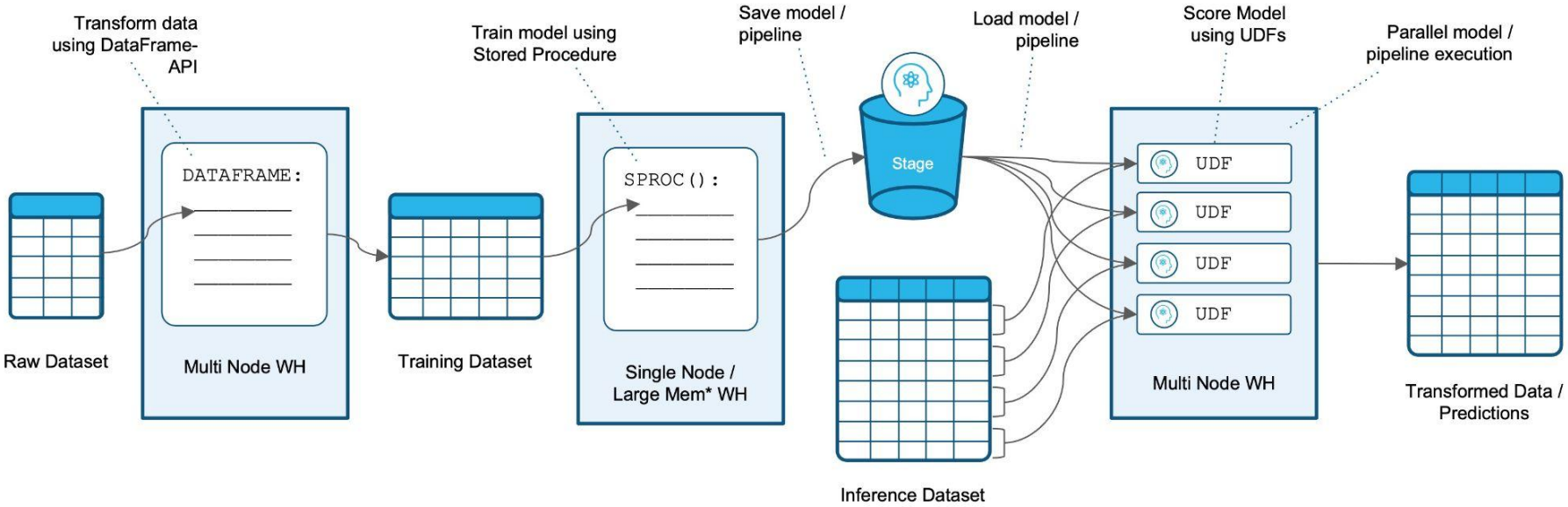
Other files are sent as is to a stage.

UDF server will use zipimport to load the Python files and directories.

```
session.add_import("./a_python_file.py")

session.udf.register(add nums,
imports=["./a_python_file.py"])
```

END-TO-END ML USING SNOWPARK



*Currently in Private Preview

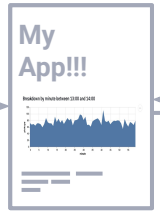
GETTING FROM DATA TO ACTION IS HARD

Step 1

Explore in a Jupyter notebook.

Step 2

Copy-paste into a Python script.



Step 3

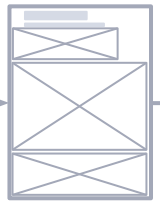
Hack a Flask app. HTML, HTTP, JS...

Step 4

Uh oh. Need features. Code is unintelligible.

Step 5

Call in the tools team. They get requirements.



Step 6

Lay out wireframe and components.

Step 7

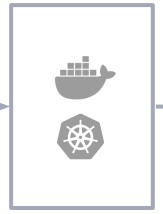
Code up the app in CSS, React, JS, etc.

Step 8

Talk to IT and get them to provision you a server

Step 9

Have DevOps configure it with the right permissions



Step 10

Get the Networking team to open a port for your server

Step 11

Figure out how to securely store and pass db credentials to your app

Step 12

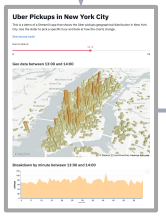
Set up SSO and other ACLs on the app so that only certain users can access

Step 13

Get IT to restart the app every time it crashes from a bug in the ML library you're using

Step 14

App finally released 2 months after you had the idea



Step ∞

Call IT every time app needs to update, scale...

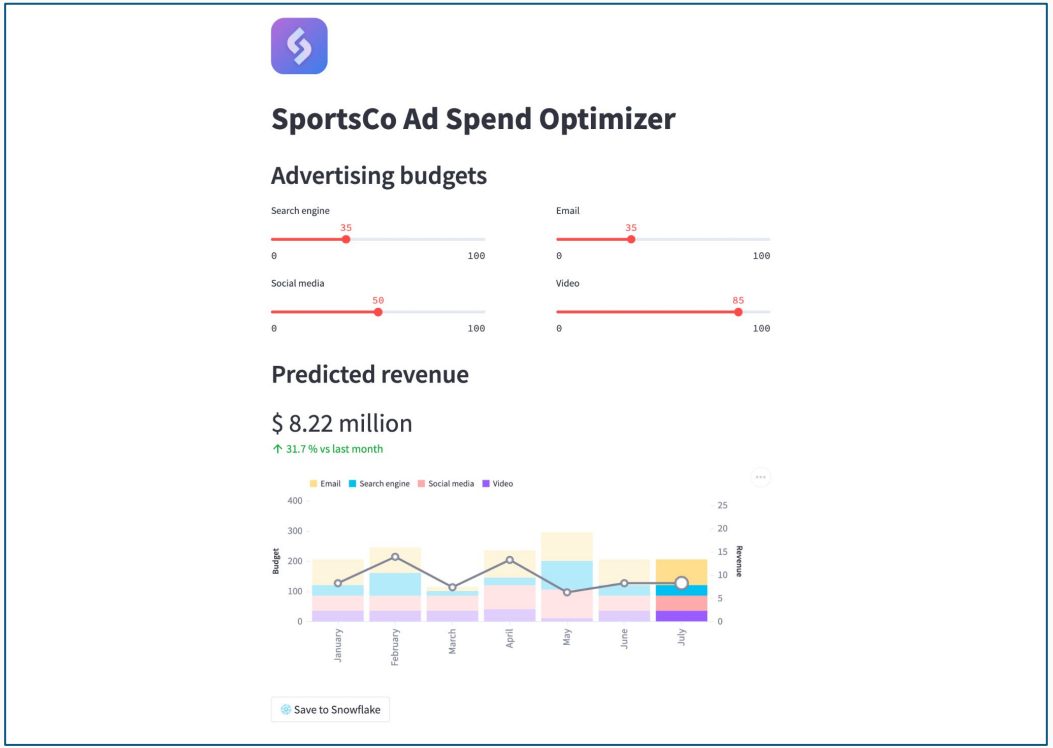
Step ∞+1

Call IT every time there's a bug, kernel update, etc...

Step ∞+n

And repeat for every new app you make!

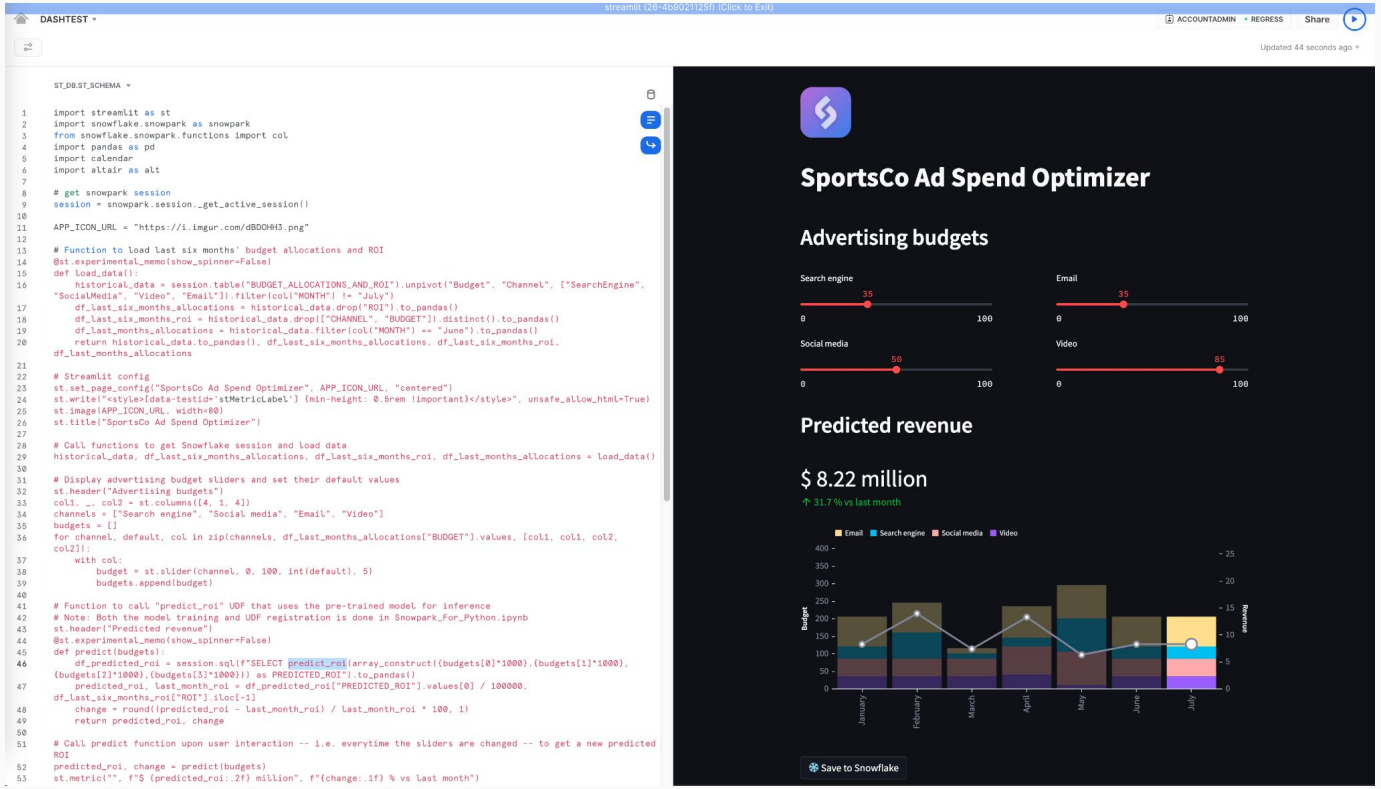
WHAT IF BUILDING APPS WERE AS EASY AS WRITING PYTHON SCRIPTS? Meet Streamlit!



1. **Pure-Python Web Framework**
2. **No HTML**
3. **No CSS**
4. **No JavaScript**

Live Demo

WHAT IF YOU COULD BUILD STREAMLIT APPS IN SNOWFLAKE? Meet Streamlit in Snowflake!



SNOWPARK: What's New

High Memory Warehouses*

- New type of Snowflake virtual warehouse
- Can be used for workloads that require a large amount of memory
- Not constrained by CPU requirements
- Typical use case
  - Train a ML model using custom code on a single node
  - Guidelines: https://docs.snowflake.com/en/LIMITEDACCESS/high-memory-warehouses.html#guidelines

**Currently in Private Preview*

SNOWPARK: A Look Ahead

- Streamlit in Snowflake
- Support for multiple Python versions
- User-Defined Aggregate Functions
- External Access

DIY: Hands-On Workshop

DIY: HANDS-ON WORKSHOP

Train, Deploy, and Run a ML model using Python, Snowpark and Streamlit

Step-by-Step Guide On GitHub

https://bit.ly/DashSnowparkPython

OTHER RESOURCES

- Quick Start Guides
  - Getting Started With Snowpark for Python and Streamlit
  - Getting Started With Snowpark Python
  - Machine Learning with Snowpark Python
- Videos: Snowpark | A Look Under The Hood
  - Snowpark API
  - Snowpark User-Defined Functions (UDFs)
- Blogs on Medium
  - Deploy Custom UDFs Using GitHub Actions
  - Snowpark For Python Open Source: How I Contributed And So Can You
- Demos on GitHub
- Developer Guide

THANK YOU!

Dash Desai, Snowflake

Senior Developer Advocate

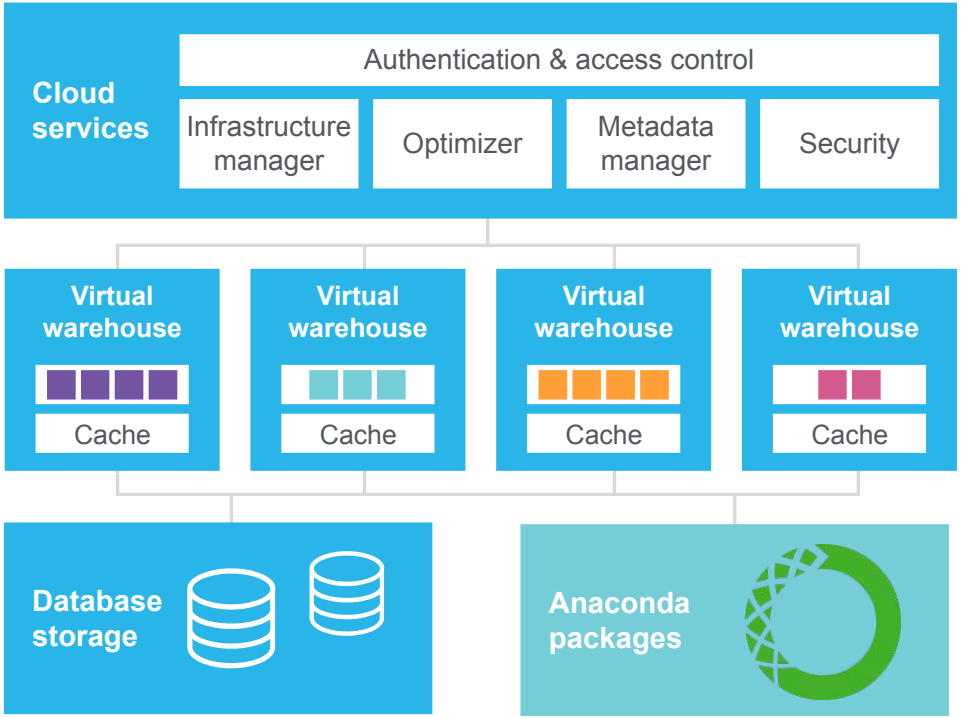
Technical Evangelist – Snowpark

dash.desai@snowflake.com

Follow me: @iamontheinet

Snowpark Python: Under The Hood

PYTHON FUNCTIONS



Python functions also leverage the existing Snowflake warehouse model for processing.

Python functions may use packages from the Snowflake Anaconda channel, which is updated regularly.

During function creation, Snowflake “solves” for the specified packages to determine which to install prior to query execution.

Packages are cached just like tables and function imports.

SOLVED PACKAGES ARE FROZEN

- The specific versions of packages are determined when functions are created.
- You can use GET_DDL to retrieve the original DDL statement:

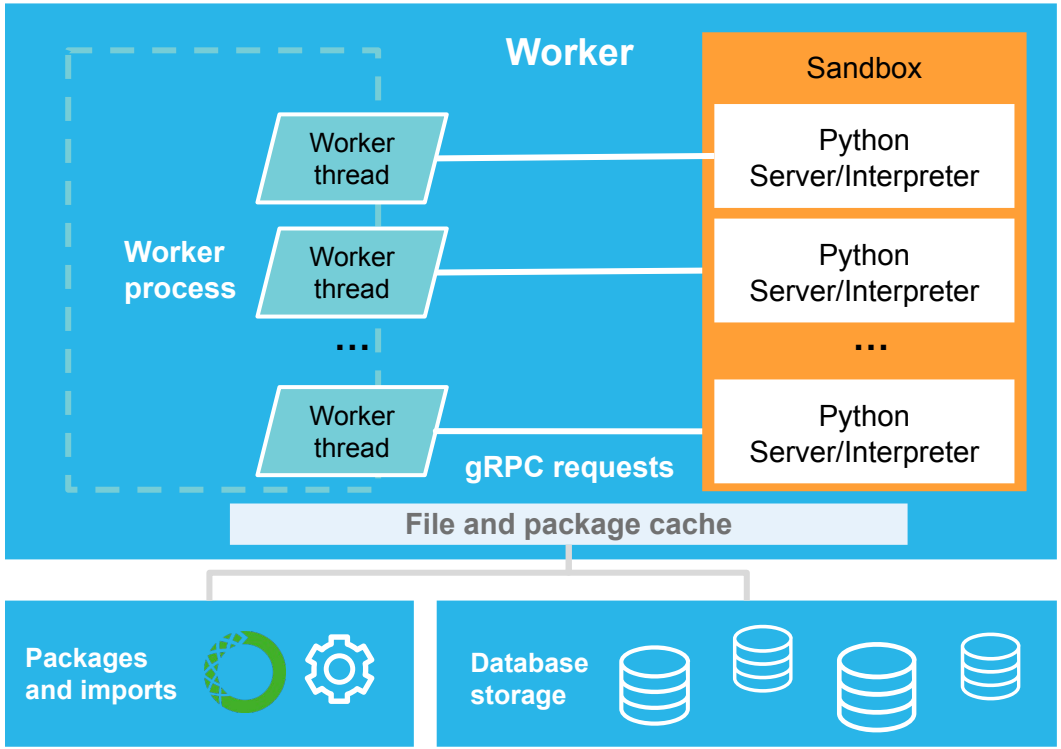
```
SELECT GET_DDL('function', 'py_udf()');
```

- Run the CREATE OR REPLACE FUNCTION statement to update the solved packages, such as to pick up a new version of ***numpy***

PYTHON FUNCTION EXECUTION

1

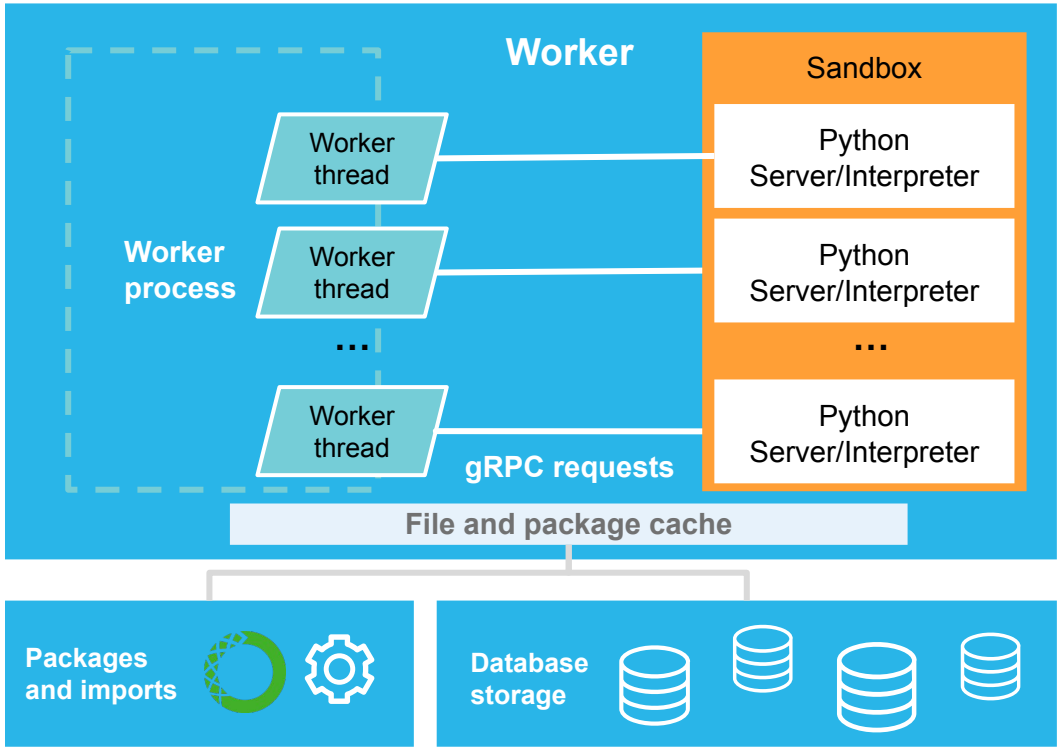
- At query startup, packages are installed on the warehouse nodes.
- If another query on the warehouse recently used the same packages, they will be cached and do not need to be installed again.
- Files from the IMPORTS list, including .py, .zip, and other imports, are either downloaded or pulled from the cache.



PYTHON FUNCTION EXECUTION

2

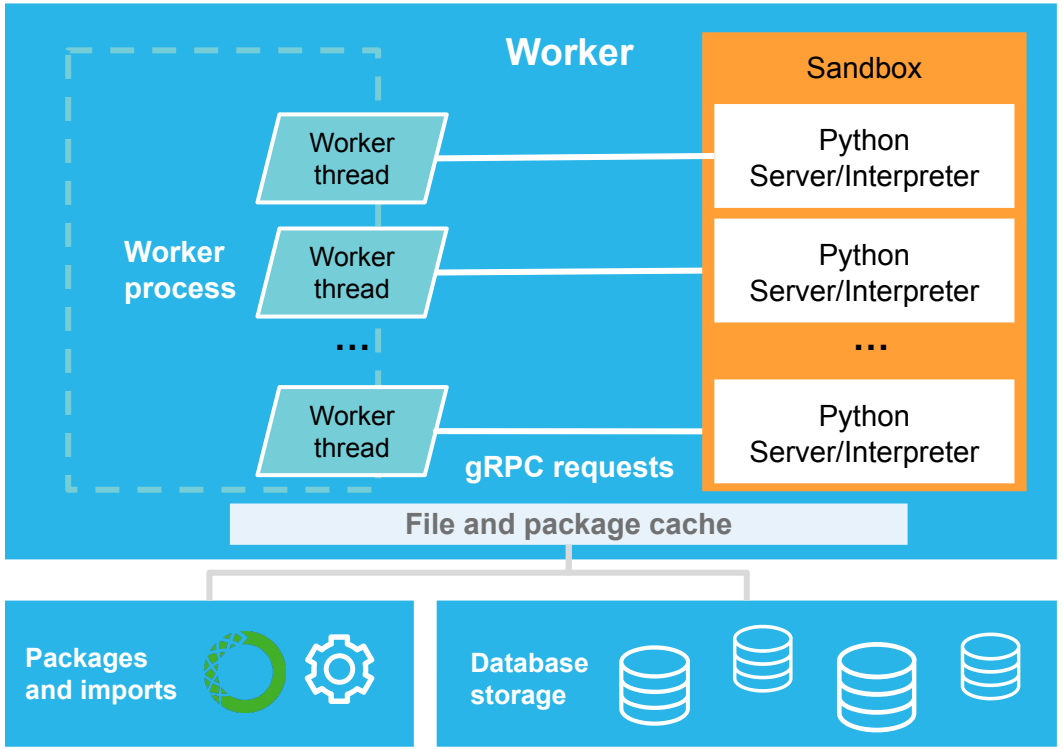
- Since Python has a global interpreter lock, Snowflake creates many Python interpreter processes for each function in the query.
- Snowflake initializes the Python interpreter before forking additional processes to reduce initialization time.



PYTHON FUNCTION EXECUTION

3

- When the query is done, packages and imports remain in a local cache, but the sandbox and Python interpreters are cleaned up.
- Re-running the same query, or another query that uses some of the same packages or imports, will be faster due to caching.

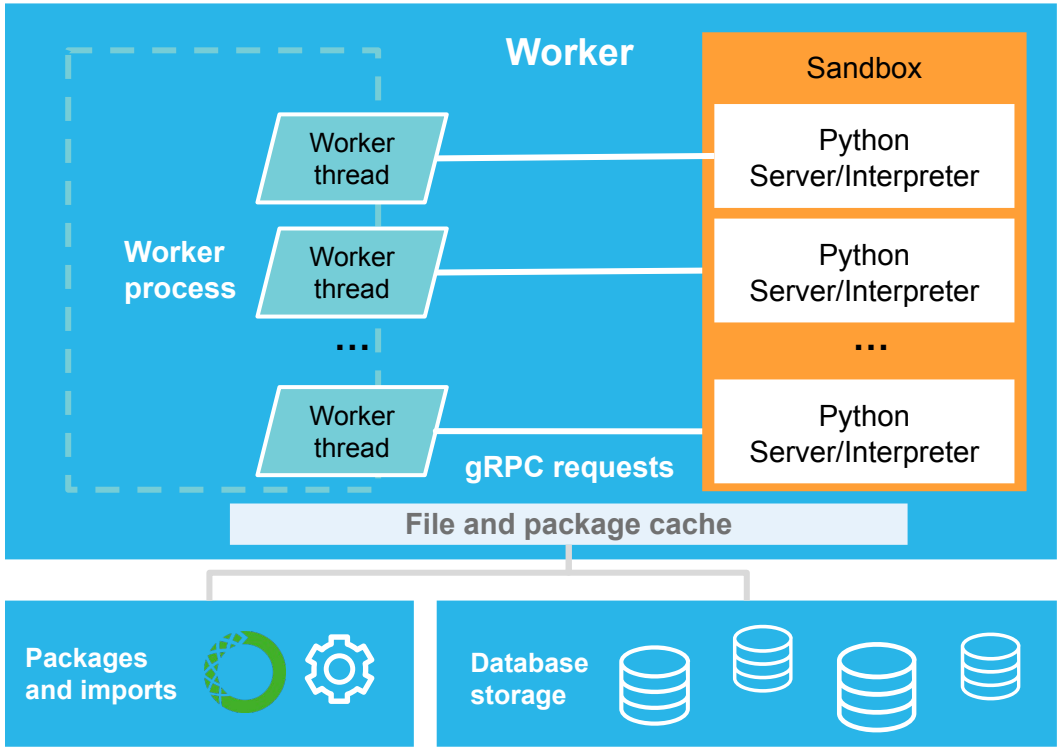


PYTHON FUNCTION SANDBOXING

4

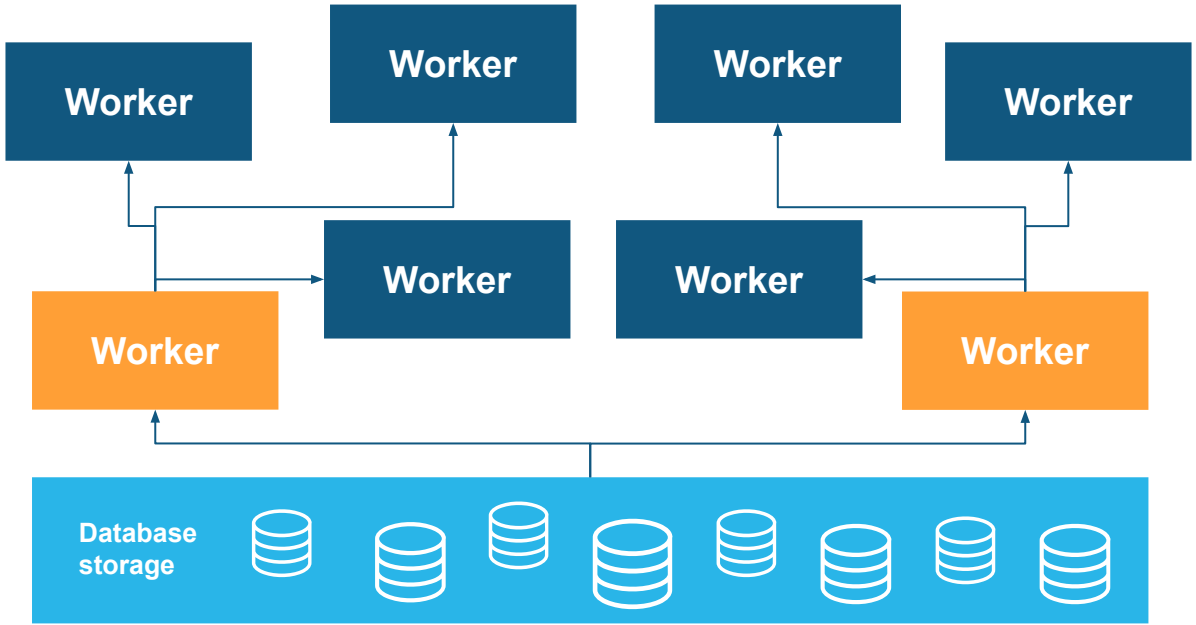
- The Python function sandbox prevents network and local file system access.
- Anaconda shares common vulnerabilities and exposures (CVEs*) through their website.
- The sandbox mitigates CVEs that could lead to data exfiltration or attacks on the host system.

*CVE = Common Vulnerabilities and Exposures



PYTHON PARALLELIZATION

- Snowflake attempts to use the full power of your warehouse when running a query with a Python UDF or UDTF.
- Rows are redistributed between nodes in the warehouse to parallelize expensive computations.
- Snowflake may adapt the query plan based on historical execution statistics.
- Using LIMIT or a heavily skewed GROUP BY, PARTITION BY, or JOIN may prevent effective parallelization.



OTHER RESOURCES

- Quick Start Guides
  - Getting Started With Snowpark for Python and Streamlit
  - Getting Started With Snowpark Python
  - Machine Learning with Snowpark Python
- Videos: Snowpark | A Look Under The Hood
  - Snowpark API
  - Snowpark User-Defined Functions (UDFs)
- Blogs on Medium
  - Deploy Custom UDFs Using GitHub Actions
  - Snowpark For Python Open Source: How I Contributed And So Can You
- Demos on GitHub
- Developer Guide

THANK YOU!

Dash Desai, Snowflake

Senior Developer Advocate

Technical Evangelist – Snowpark

dash.desai@snowflake.com

Follow me: @iamontheinet