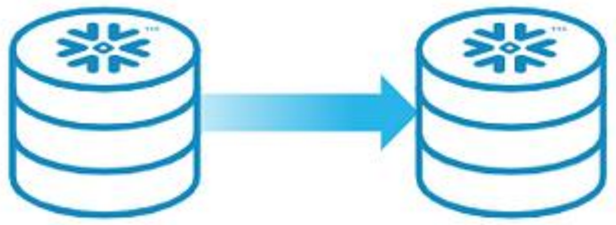
Data replication and failover

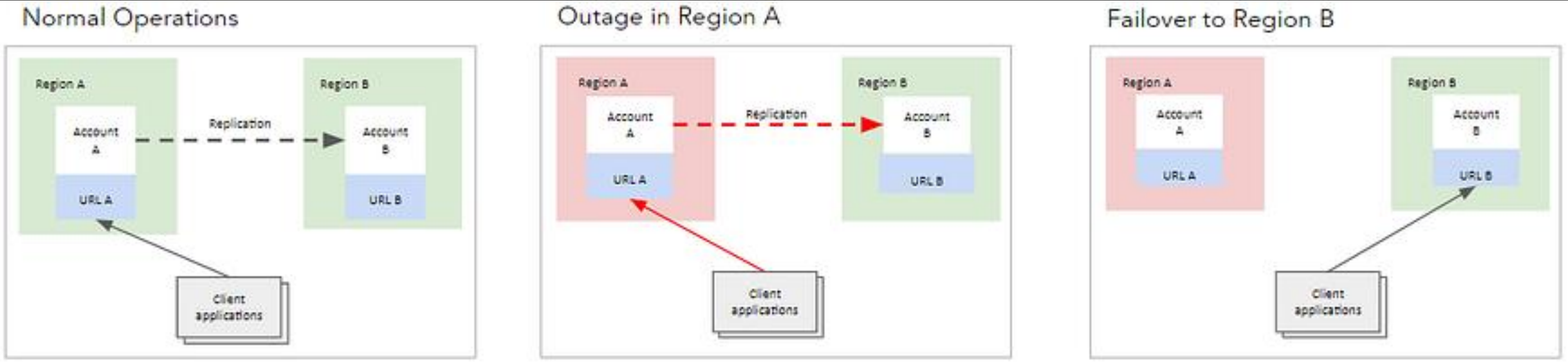
Database Replication



- Serverless replication of data from one account to another
- One command and runs instantaneously
- Executed through a Task

Database Failover

When one Snowflake account is unavailable, another can be promoted to be primary. This can happen across cloud providers or regions.



A: Prepare for an outage

Setup Snowflake account(s)
Replicate databases to that account

B: Monitor Snowflake's availability

Use Snowflake status page - subscribe to updates

C: Recover from an outage

Readable secondary databases
Failover databases (read-write)

1. Data Replication Concepts

Replication is the process of syncing database objects from a **Primary** account to one or more **Secondary** accounts in different regions or even different cloud providers (e.g., from AWS US-East to Azure West Europe).

Key Objects Replicated

It's not just the raw data. Snowflake replicates:

- **Permanent and Transient Tables** (Temporary tables are *not* replicated).
- **Metadata:** Roles, Users, Privileges, and Shares.
- **Integrations:** API integrations, Storage integrations, and Network policies.

The Replication Process

1. **Primary Database:** The "source of truth" where Read/Write operations happen.
2. **Secondary Database:** A read-only replica. To keep it current, you must "refresh" it.
3. **Refresh Mechanism:** Incremental data transfers are used to minimize costs.

2. Failover and Failback

While replication moves the data, **Failover** is the action of switching operations to the secondary site during an outage.

Business Continuity Features

- **Failover Groups:** You group databases and account objects (like users/roles) together. If a region goes down, you fail over the entire group to ensure the environment remains functional.
- **Client Redirection:** Snowflake provides a **Connection URL**. Instead of changing your application code, you simply point the URL to the new primary site.
- **Promotion:** When you fail over, the **Secondary** database is "promoted" to **Primary**, becoming Read/Write.

To ace the SnowPro Core, remember these specific rules:

- **Database Objects:** Only Permanent and Transient databases can be replicated.
- **Credit Consumption:** Replication uses **Compute credits** (for the refresh process) and **Cloud Services credits**. You are also charged for **Data Transfer** fees across regions/clouds.
- **Database Shares:** You can replicate databases that are being shared, but there are specific steps to ensure the "Share" object exists in the secondary region.
- **External Tables:** Standard replication does not include the physical files in your external stages (S3/Azure Blob); you must manage that underlying storage replication yourself.

Steps to Create Replication in snowflake

Step 1: Enable Replication for the Account

```
ALTER ACCOUNT SET REPLICATION_ALLOWED_TO_ACCOUNTS = <org_name>.<target_account_name>;
```

Step 2: Create a Secondary Database (The Replica)

```
CREATE DATABASE <db_name> AS REPLICA OF <org_name>.<source_account_name>.<db_name>;
```

Step 3: Perform the Initial Refresh

```
ALTER DATABASE <db_name> REFRESH;
```

Step 4: Schedule Recurring Refreshes (Optional but Recommended)

Step 5: Configure a Failover Group (For Business Critical Edition)

If you want to be able to "failover" (promote the secondary to primary), you should use Failover Groups. This bundles multiple databases and account objects (Users, Roles) together.

Secure Data Sharing

Secure Data Sharing enables sharing selected objects in a database in your account with other Snowflake accounts. The following Snowflake database objects can be shared:

•Tables

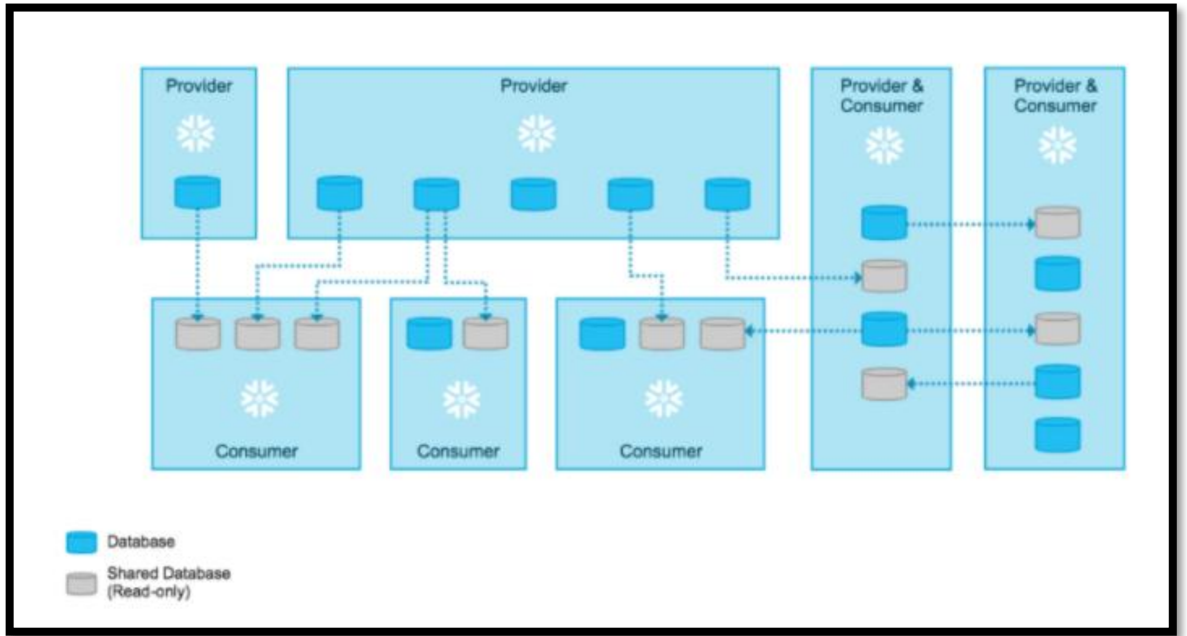
•External tables

•Secure views

•Secure materialized views

•Secure UDFs

With Secure Data Sharing, ***no*** actual data is copied or transferred between accounts. All sharing is accomplished through Snowflake's services layer and metadata store. The shared data does not take up any storage in a consumer account and, therefore, does not contribute to the consumer's monthly data storage charges. The ***only*** charges to consumers are for the compute resources (i.e. virtual warehouses) used to query the shared data.



Shares

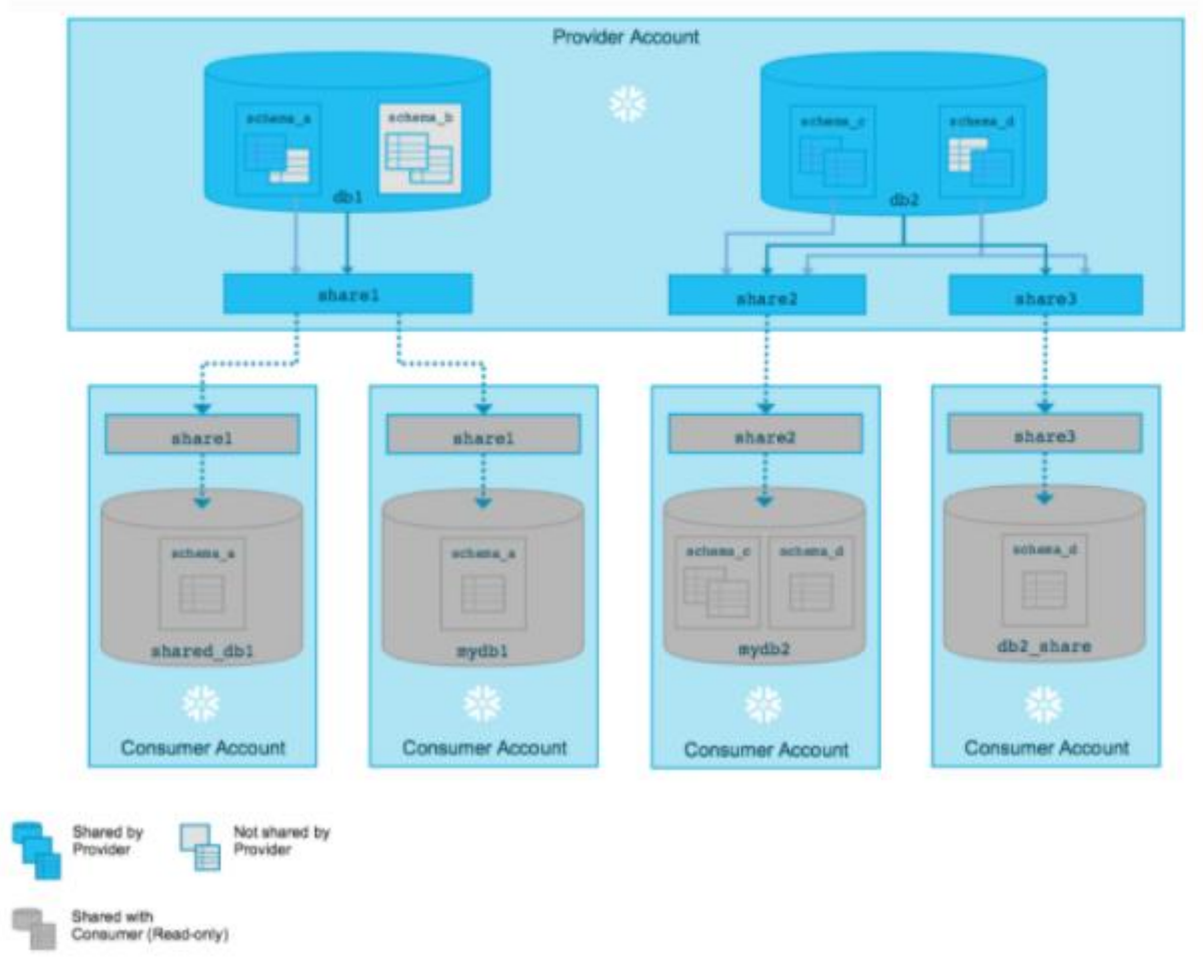
Shares are named Snowflake objects that encapsulate all of the information required to share a database. Each share consists of:

•The privileges that grant access to the database(s) and the schema containing the objects to share.

•The privileges that grant access to the specific objects in the database.

•The consumer accounts with which the database and its objects are shared.

Once a database is created (in a consumer account) from a share, all the shared objects are accessible to users in the consumer account.



Shares are secure, configurable, and controlled 100% by the provider account:

•New objects added to a share become immediately available to all consumers, providing real-time access to shared data.

•Access to a share (or any of the objects in a share) can be revoked at any time.

Key Secure Data Sharing Features

Zero Data Movement: Data is never copied or transferred; consumers access the provider's data directly in real-time.

Live, Read-Only Access: Shared data is always up-to-date, and consumers cannot modify or delete the source data.

Secure Views & Materialized Views: Providers can use secure views to hide sensitive data (columns/rows) while still sharing relevant insights.

Granular Access Control: Uses standard RBAC (Role-Based Access Control) to manage permissions on specific databases, schemas, and tables.

Secure Data Exchange/Marketplace: Enables sharing within a private, curated exchange or public marketplace for broader collaboration.

Reader Accounts: Providers can create temporary, lightweight accounts for non-Snowflake customers to consume data, with the provider bearing all costs.

End-to-End Encryption: All data is encrypted in transit and at rest.

Metadata-Based: Sharing happens through the services layer, meaning no storage costs for the consumer

Cloning

Zero-Copy Cloning is a powerful feature in Snowflake that allows you to create a clone of a database, schema, or table without duplicating the underlying data. This means that the clone is created almost instantly and doesn’t consume additional storage space initially, making it an efficient way to manage and experiment with data.

Key Features of Zero-Copy Cloning

1.**Instant Cloning**: Clones are created quickly, as they reference the same data as the original object. This allows for near-instantaneous creation of clones, even for large datasets.
2. **Storage Efficiency**: Initially, the cloned object does not take up additional storage space, as it points to the same underlying data. Storage is only consumed as changes are made to either the original object or the clone.
3. **Independent Changes**: After a clone is created, it can be modified independently of the original object. Changes made to the clone do not affect the original object and vice versa. This is useful for testing or development without risking the integrity of the original data.
4. **Point-in-Time Cloning**: You can create a clone of a table, schema, or database as it existed at a specific point in time using Time Travel features. This allows you to recover or analyze data from a previous state easily.

How to Use Zero-Copy Cloning

To create a clone in Snowflake, you use the CLONE command. Here’s an example:

```
CREATE TABLE|DATABASE|SCHEMA my_cloned_table CLONE <my_value>;
```

Time Travel

Time Travel in Snowflake lets you access data that has been modified or deleted by querying previous versions of a table or a database at any point within a retention period (up to 90 days, depending on your Snowflake edition). The retention period defines how long Snowflake keeps the historical data, enabling you to use this feature for things like:

•**Data recovery**: If you accidentally delete or overwrite important data, Time Travel allows you to recover it.
•**Auditing**: Review how your data has changed over time, which can be crucial for auditing purposes.
•**Debugging**: If you encounter an issue in your data pipeline, you can query data as it was before certain transformations or processes.

Now, let's dive into three methods you can use to query data from a specific time in the past using Time Travel.

1. Querying Data Using Time Travel with an Offset

```
SELECT * FROM OUR_FIRST_DB.public.test AT (OFFSET => -60 * 1.5);
```

2. Querying Data Using a Specific Timestamp

```
SELECT * FROM OUR_FIRST_DB.public.test BEFORE (TIMESTAMP => '2025-01-02 14:21:50.581'::timestamp);
```

3. Querying Data Using a Query ID

```
SELECT * FROM OUR_FIRST_DB.public.test BEFORE (STATEMENT => '019b9ee5-0500-8473-0043-4d8300073062');
```

Time Travel Use Cases

Time Travel provides invaluable benefits in a variety of real-world scenarios:

1. **Recovering Data**: If you've accidentally deleted or updated data, Time Travel can allow you to recover its previous state without relying on backups.
2. **Audit and Compliance**: For businesses in regulated industries, Time Travel helps maintain historical data snapshots to comply with legal requirements and auditing processes.
3. **Data Correction**: If your ETL pipeline introduces erroneous data, you can query the data as it existed before the mistake and fix the issue.
4. **Business Intelligence & Analytics**: Analysts can use Time Travel to review historical versions of data, which may reveal valuable insights into past trends, anomalies, or changes in behavior.

Important Considerations for Time Travel

- **Retention Period**: Snowflake retains historical data for a specific duration, typically up to **90 days**. However, the exact retention period can vary based on your Snowflake edition.
- **Cost**: While Time Travel is a powerful feature, it does incur storage costs for maintaining historical data versions. Be mindful of how much historical data you're keeping and how long.
- **Data Modification and Time Travel**: Time Travel allows you to access **data as it was at any point in time**, but it won't be able to recover dropped objects like tables or schemas after the retention period has passed.

Fail-safe

Snowflake provides a Fail-safe feature to ensure data protection. Fail-safe is part of the continuous data protection lifecycle provided by Snowflake. It provides non-configurable 7 days further storage of historical data after the time travel period has ended.

So, if we update or delete the “n” number of rows from a table then these updated rows will be retained first in the time travel history for the configured retention period, which can be up to 90 days. Once the time travel history ends, Snowflake still keeps the data for a further 7 days as a backup. It is, however, important to note that the time travel data can be directly queried by the customer but the fail-safe data can only be recovered by requesting Snowflake support. Also, time travel and fail-safe both contribute to the storage cost.

Why Fail-safe?

Data is backed up at regular intervals, there can be situations between the two back-ups during which the data has been changed accidentally or deleted due to some mistake. So the fail-safe provides a quick and efficient way to recover such data with minimal cost and minimal effort.

Key Features of Fail-Safe

•The amount of storage for fail-safe can be of 0 or 7 days and it is not configurable.

•For permanent tables, the fail-safe is 7 days.

•For transient and temporary tables, the fail-safe is 0 days.

•Users are not allowed to query on fail-safe.

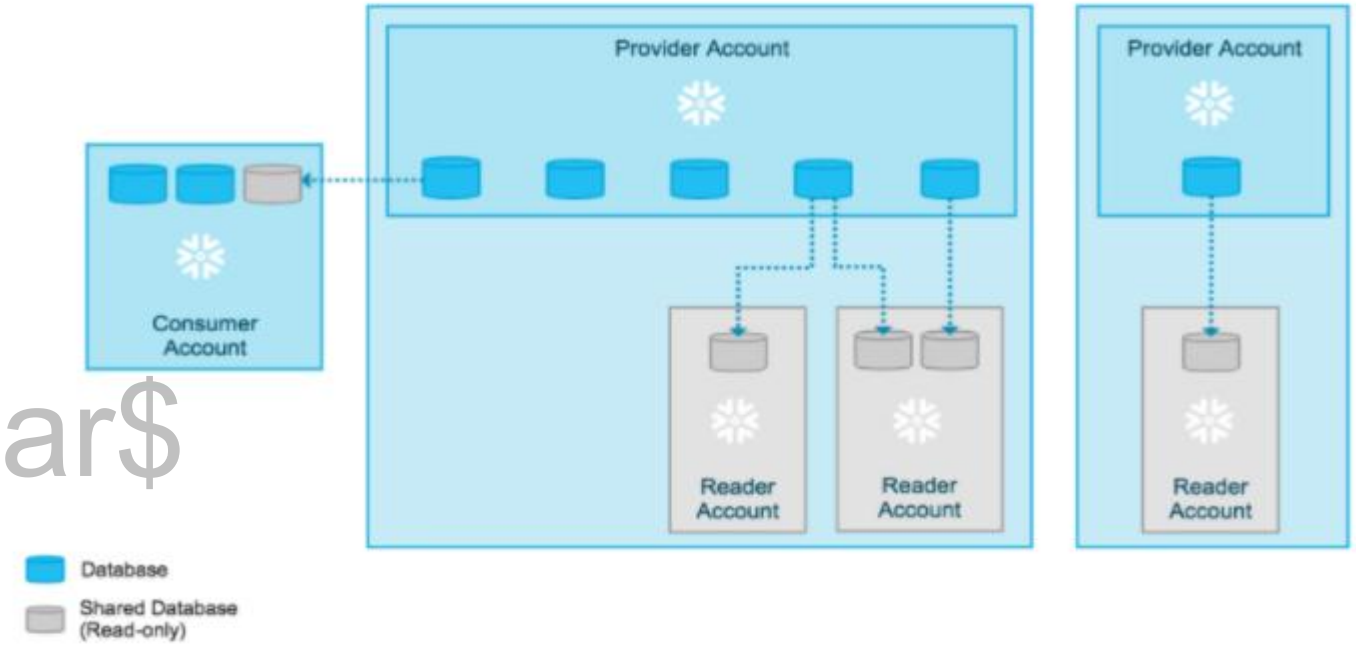
•Data can only be recovered through Snowflake support.

Accounts

Providers: A data provider is any Snowflake account that creates shares and makes them available to other Snowflake accounts to consume. As a data provider, you share a database with one or more Snowflake accounts. For each database you share, Snowflake supports using grants to provide granular access control to selected objects in the database

Consumers: A data consumer is any account that chooses to create a database from a share made available by a data provider. As a data consumer, once you add a shared database to your account, you can access and query the objects in the database just as you would with any other database in your account.

Reader Accounts: Each reader account belongs to the provider account that created it. Similar to standard consumer accounts, the provider account uses ***shares*** to share databases with reader accounts; however, a reader account can only consume data from the provider account that created it: Users in a reader account can query data that has been shared with it, but cannot perform any of the DML tasks that are allowed in a full account (data loading, insert, update, etc.).



Resharing & Direct Shares

Resharing requires using Snowflake Data Exchange or direct sharing to a third party from the consumer side

Resharing Limitations: While a consumer cannot directly "re-share" a shared database, they can create a new view or table from the shared data and then share that new object with another account, effectively creating a daisy-chain.

Direct Shares

Snowflake **Direct Shares** provide a secure, real-time, and simple method for sharing specific database objects (like tables and secure views) between a data provider's Snowflake account and one or more consumer Snowflake accounts within the **same Snowflake region**. A key feature is that no data is actually copied or moved, eliminating data duplication and ensuring consumers always access live, up-to-date data.

Key Characteristics of Direct Shares

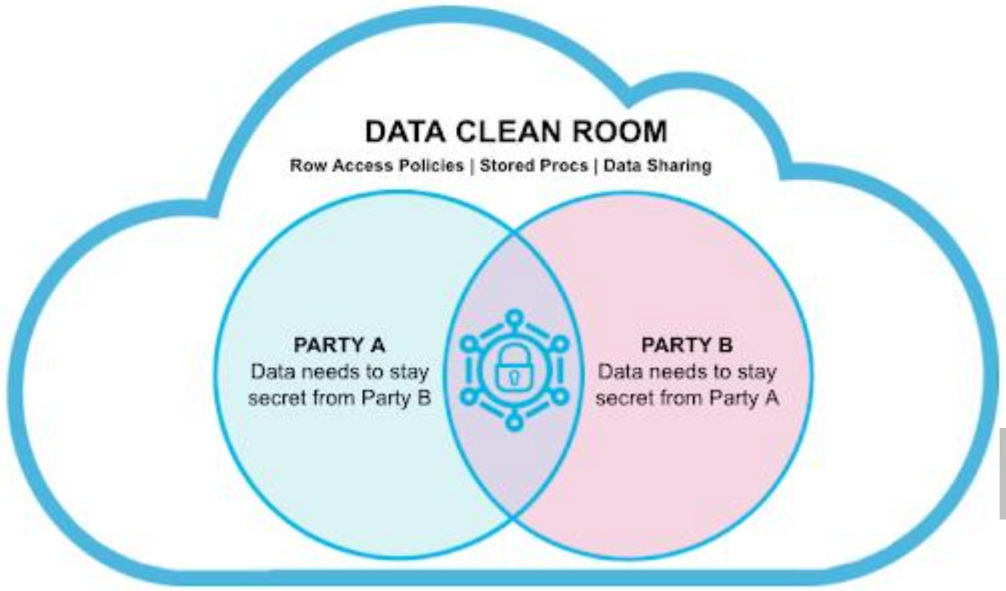
Zero Copy: Data is not copied or transferred; access is granted via Snowflake's metadata store.
Real-time Access: Any updates to the provider's data are immediately available to the consumer.
Cost Allocation: The data provider incurs storage costs, while the consumer pays for the compute resources (virtual warehouses) used to query the shared data.
Access Control: The provider maintains complete control over the data and can revoke access at any time.
Region-Specific: Direct shares are restricted to sharing data within the same Snowflake region. For cross-region or cross-cloud sharing, a private listing or data replication is required.
Consumer Types: Sharing can occur with another existing Snowflake account or with a non-Snowflake user via a special, provider-managed reader account.

Data Clean Rooms

Data clean rooms are configurable, isolated Snowflake environments where collaborators can import data, specify what queries can be run against that data, and configure data protection settings such as differential privacy and specifying joinable and projectable columns. Access to a clean room is by invitation only.

Key Concepts

- Secure Collaboration
- Provider and Consumer Roles
- Access Control & Policies
- No-Code UI & Developer APIs


DATA CLEAN ROOM
Row Access Policies | Stored Procs | Data Sharing
PARTY A
Data needs to stay secret from Party B
PARTY B
Data needs to stay secret from Party A

Common Use Cases

Snowflake Data Clean Rooms are used across various industries, especially for scenarios requiring sensitive data analysis:

Advertising & Marketing: Measuring campaign performance (e.g., ad reach and frequency, last-touch attribution), optimizing ad spend, and building look-alike audiences without sharing user-level data.

Healthcare: Accelerating drug research by enabling secure data analysis between labs and healthcare facilities without exposing patient information.

Financial Services: Improving fraud detection models and credit scoring while safeguarding customer data.

Listings

In Snowflake, Listings are an enhanced method of Secure Data Sharing that allow providers to share data, Native Apps, or services with consumers. Listings can be categorized by their visibility and accessibility as either **Private or Public** (Marketplace).

Private Listings (Targeted Sharing)

Private listings are designed for secure, direct, and controlled sharing with specific, known Snowflake accounts.

Visibility: Not visible to the public. Only the specified consumer accounts can see and access them.
Access Requirement: The provider must know the specific Organization and Account Name of the consumer.
Use Cases: Sharing with specific partners, clients, or internal departments.
Capabilities: Supports free and paid models (including custom or offline payment agreements).
Cross-Region: Supports Cross-Cloud Auto-Fulfillment to replicate data to the consumer's region automatically.

Public Listings (Snowflake Marketplace)

Public listings are published on the Snowflake Marketplace, making them discoverable and accessible to a wider audience.

Visibility: Visible to all Snowflake users, allowing for broader market reach.
Access Requirement: Any Snowflake user can find and initiate the "Get" process for the listing.
Use Cases: Data monetization, publishing public data, and promoting data products to the entire Snowflake Data Cloud.
Capabilities: Supports free, limited trial (1-90 days), and paid listings.
Approval Process: Requires approval from Snowflake to be published on the Marketplace

Native Apps

The Snowflake Native App Framework enables developers to build, distribute, and monetize data applications that run entirely within a consumer's Snowflake account. This approach brings the application logic to where the data already resides, eliminating the need for data movement and enhancing security and governance.

Key Features and Concepts

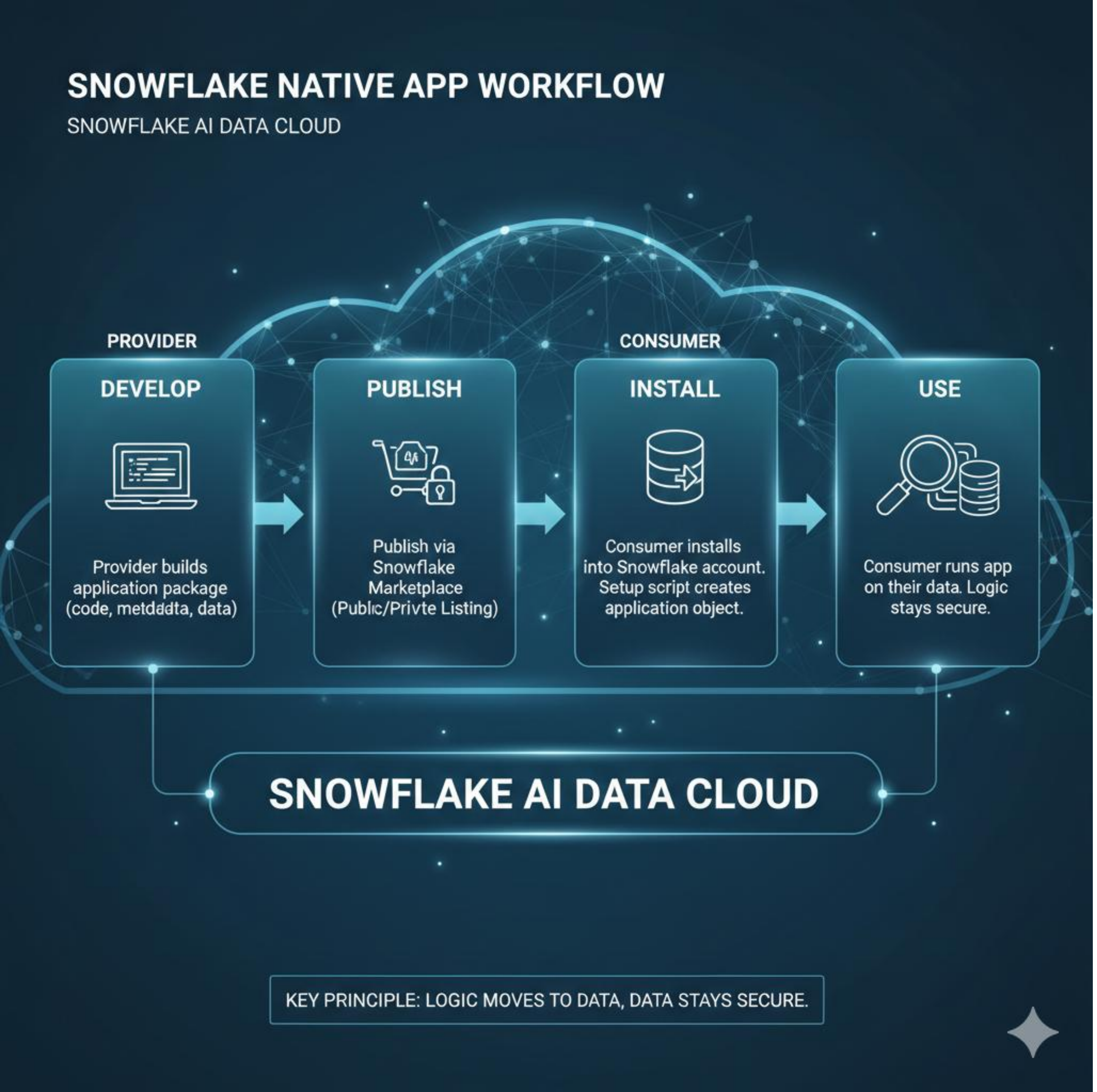
In-Account Execution: Native Apps are installed and run directly in the consumer's Snowflake account, so sensitive data never leaves their secure environment.

Monetization & Distribution: Providers can list their applications on the Snowflake Marketplace using various licensing and pricing models (free, trial, or paid subscriptions).

Development Tools: Applications can be built using familiar languages and tools within the Snowflake ecosystem, including SQL, Python via Snowpark API, and user interfaces via Streamlit in Snowflake.

Intellectual Property Protection: The framework is designed to protect the provider's underlying code and business logic, even while it executes within the consumer's account.

Snowpark Container Services (SPCS): For more complex applications, Native Apps can leverage SPCS to run containerized workloads, allowing for the integration of external software, sophisticated AI/ML models, and third-party services.


SNOWFLAKE NATIVE APP WORKFLOW
SNOWFLAKE AI DATA CLOUD
PROVIDER
CONSUMER
DEVELOP
Provider builds application package (code, metddata, data)
PUBLISH
Publish via Snowflake Marketplace (Public/Privte Listing)
INSTALL
Consumer installs into Snowflake account. Setup script creates application object.
USE
Consumer runs app on their data. Logic stays secure.
SNOWFLAKE AI DATA CLOUD
KEY PRINCIPLE: LOGIC MOVES TO DATA, DATA STAYS SECURE.

Example Use Cases

AI & ML Solutions: Companies can distribute proprietary AI/ML models (e.g., for credit risk prediction or computer vision) as Native Apps, allowing customers to run predictions against their private data securely.

Data Analytics: Providers offer ready-to-use analytics tools (e.g., sales or finance analytics) that integrate seamlessly with a consumer's existing data, offering immediate insights without requiring a separate data warehouse build.

Data Connectors: Developers can build native data ingestion connectors for various platforms (e.g., GitHub, Kafka) that run inside Snowflake to streamline data pipelines.