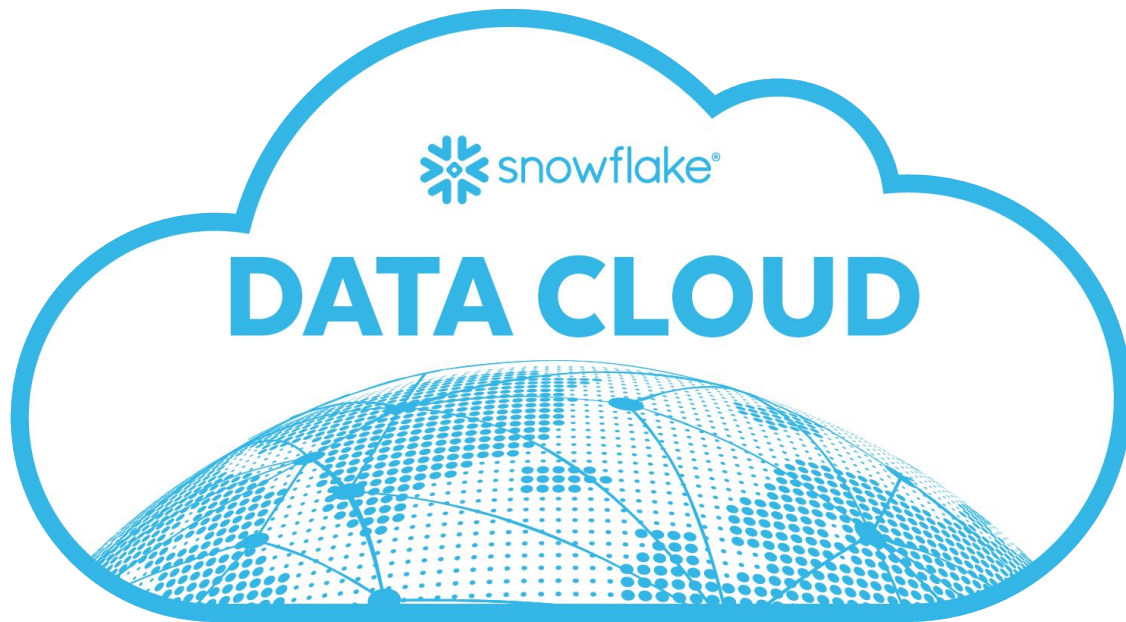


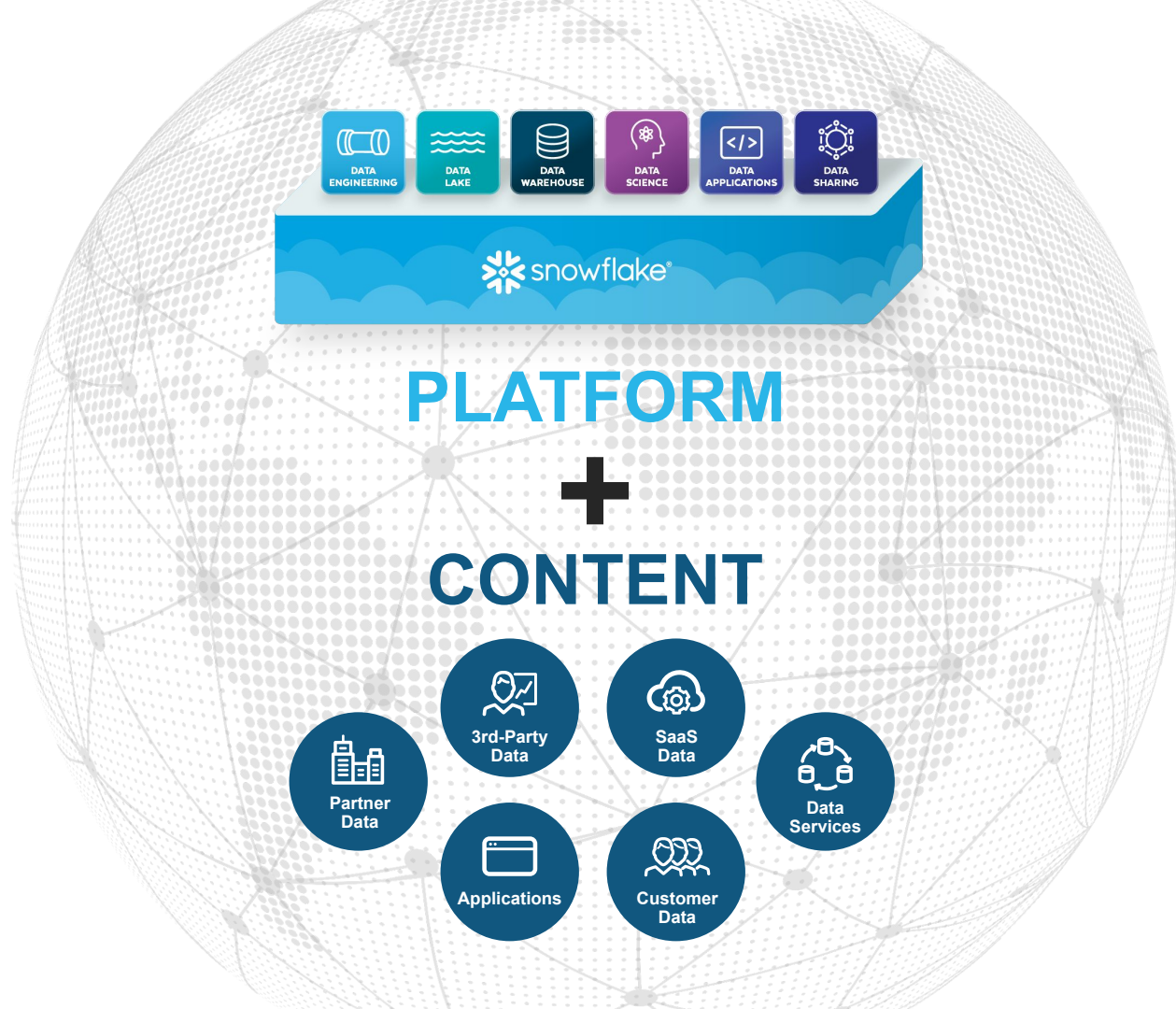
THE DATA CLOUD IS A GLOBAL NETWORK



One global, unified system connecting companies and data providers to the most relevant data for their business



ELEMENTS OF THE DATA CLOUD



THE SNOWFLAKE PLATFORM

Customers, for example:

- *Create & manage users*
- *Load data*
- *Execute commands*
- *Export data*



Built-in security & governance features protect the data you load and use in Snowflake

- Snowflake uses the industry-standard “shared responsibility” model
- Snowflake personnel do not have unauthorized access to customer data
- Snowflake personnel do not *collect, delete, update, disclose, or use* customer data

Snowflake, for example:

- *Processes requests*
- *Maintains security*
- *Manages capacity*

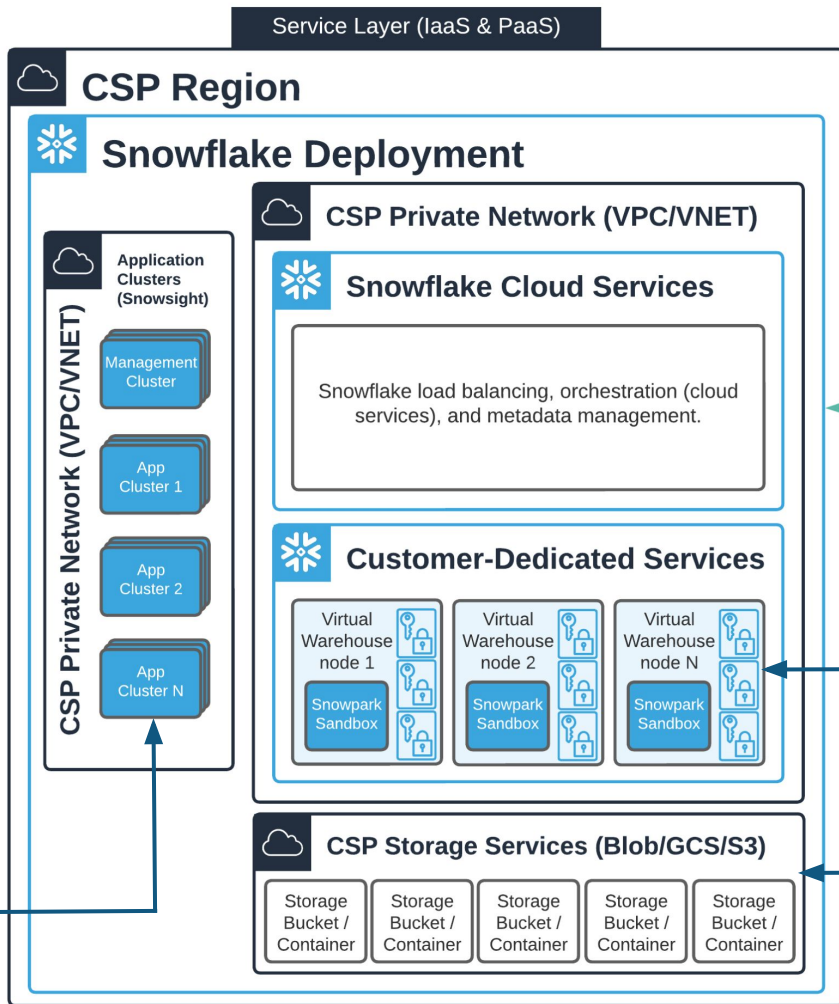
- Orchestrated security built in to the fabric of the platform
 - Automated controls in place for all functions
 - Constant monitoring
 - Analysis to detect and mitigate threats quickly

Snowflake uses sophisticated mechanisms to keep the platform safe and stable



- Customers never have direct access (e.g., "SSH") to the Snowflake VPC/VNET
- All access to customer data is through the Snowflake Service application layer
- Customer data is decrypted in memory on dedicated Virtual Warehouse VMs:
 - Only the data **required to process the command is decrypted**
 - Virtual Warehouses are **ephemeral**, meaning they run only when needed

Snowsight app clusters are customer-dedicated services



SNOWFLAKE ARCHITECTURE at a Glance



Virtual Warehouses only process data for a single customer.

Snowpark code runs in a [multi-layered sandbox](#).

Customer data is encrypted at rest using **dedicated encryption keys**.

Storage is governed by dedicated IaaS user principals.



SNOWFLAKE SECURITY & GOVERNANCE AT A GLANCE

1 Network Controls

- [All communication secured](#) using TLS 1.2 for all client communications, and controlled by [Network Policies](#) (IP Allowlisting)
- CSP Private Networking Integration
 - [GCP Private Service Connect](#)
 - [AWS Privatelink](#)
 - [AWS VPC ID S3 policies](#)
 - [Azure Private Link](#)
 - [Azure subnet rules for Blob access](#)

2 Identity & Access

- [User Provisioning with SCIM](#)
- [Native Snowflake credentials](#)
 - Multi-Factor Authentication
 - Key Pair Authentication
- [Federated Identity](#)
 - SAML 2.0-based SSO
 - OAuth 2.0 delegated authorization
- [Password & Session Policies](#)
- [Service / Functional Accounts](#)

3 Data Governance

- [Built-in Features & partner integrations](#)
- [RBAC & DAC, Views & UDFs](#)
- Column-Level Security
 - [Dynamic data masking](#)
 - [External tokenization](#)
- [Row access policy](#)
- [Conditional policy](#)
- [Tagging](#)
- [Tag-based policy](#)
- [Classification](#)
- [Object dependencies](#)
- [Access history](#)

4 Encryption

- [Customer data always encrypted in flight](#)
- Data-at-rest always encrypted using a hierarchical key model
 - [Rooted in the CSP's HSM](#)
 - [Automated key rotation & re-keying](#)
 - [BYOK with "Tri-Secret Secure"](#)

5 Data Protection

- Account, region, cloud, and data-level recovery & failover
 - [Fail-Safe](#)
 - [Time Travel](#)
 - [Cross-cloud & region replication & failover](#)
 - [AWS, Azure, GCP redundancy](#)

6 Auditing

- [Comprehensive audit trail for all activities by all users from login](#)
 - Stored for 365 days in customer tamper-proof area of account
- [Access History for every object](#)
- [Pull or push models for export](#)

7 Compliance & Legal



SOC 2 Type II
12 Month Coverage Period

SOC 1 Type II
6 Month Coverage Period



Moderate and High on SnowGov regions:
AWS, Azure

- [Full compliance report list](#)
- [Snowflake Security Addendum](#)
- [Snowflake's own Security Data Lake](#)
- <https://www.snowflake.com/legal/> for DPA (GDPR), acceptable use, support, and more – [about GDPR](#)



INFOSEC & COMPLIANCE

All reports, attestations, documentation, and certifications

Third-Party Reports & Certifications

- Snowflake SOC 1 and 2 Type II Report
- Snowflake PCI-DSS-AOC-Final Report
- HITRUST Certification w/ HIPAA scoping factor (with Shared Responsibility Matrix)
- Snowflake's ISO 27001 Certificate (includes ISO/IEC 27017 & ISO/IEC 27018)
- ISO 9001
- CSA STAR Level I
- BSI C5 (Germany)
- [FedRAMP](#) (Package on OMB MAX)
 - FedRAMP [High](#) (AWS), [Moderate](#) (Azure)
 - DoD CC SRG IL4
- StateRAMP & TX-RAMP
- IRAP Protected
- Cyber Essentials Plus
- CyberGRX Report
- Penetration Test Results



ISO/IEC 27001
ISO/IEC 27017
ISO/IEC 27018



SOC 2 Type II
12 Month Coverage Period
SOC 1 Type II
6 Month Coverage Period



DoD CC SRG IL4:
AWS GovCloud



PCI-DSS



Moderate and High on
SnowGov regions:
AWS, Azure



800-171



HM Government
G-Cloud13
Supplier

Snowflake's Policy & Legal Documentation

- [Snowflake Legal Page](#)
- [Snowflake Security Addendum](#)
- [Snowflake Data Privacy Framework Notice](#)

Snowflake Internal Controls & Testing

- DRP, BCP, and Information System Contingency Plans
- Security Incident Process
- Staff Training, Onboarding, and Access Policies

Snowflake Self-Assessment Reports

- CAIQ
- SIG Lite
- Red Team Pen Tests

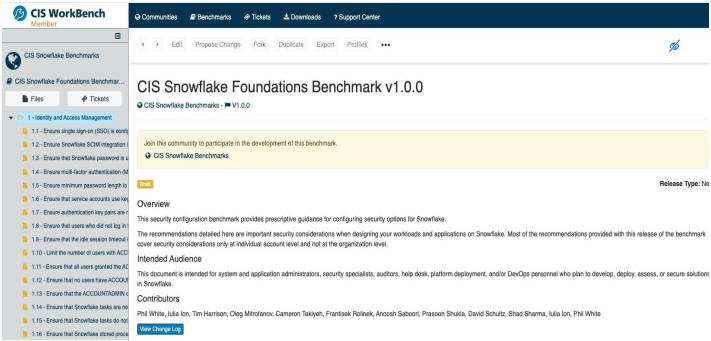


SNOWFLAKE SHARED RESPONSIBILITY MODEL & CIS BENCHMARKS



Security Best Practices for the Customer

- Snowflake collaborates with the [Center for Internet Security \(CIS\)](#) and the security community to create and publish the [CIS Snowflake Foundations Benchmark](#). (Create a free CIS account to access it.)
- The Snowflake [shared responsibility model](#) establishes a clear understanding of who is responsible for which security controls.
- These two initiatives play a critical role in helping secure Snowflake accounts for all Snowflake customers.
- Snowflake avoids account compromises by following the security best practices in the CIS benchmark.

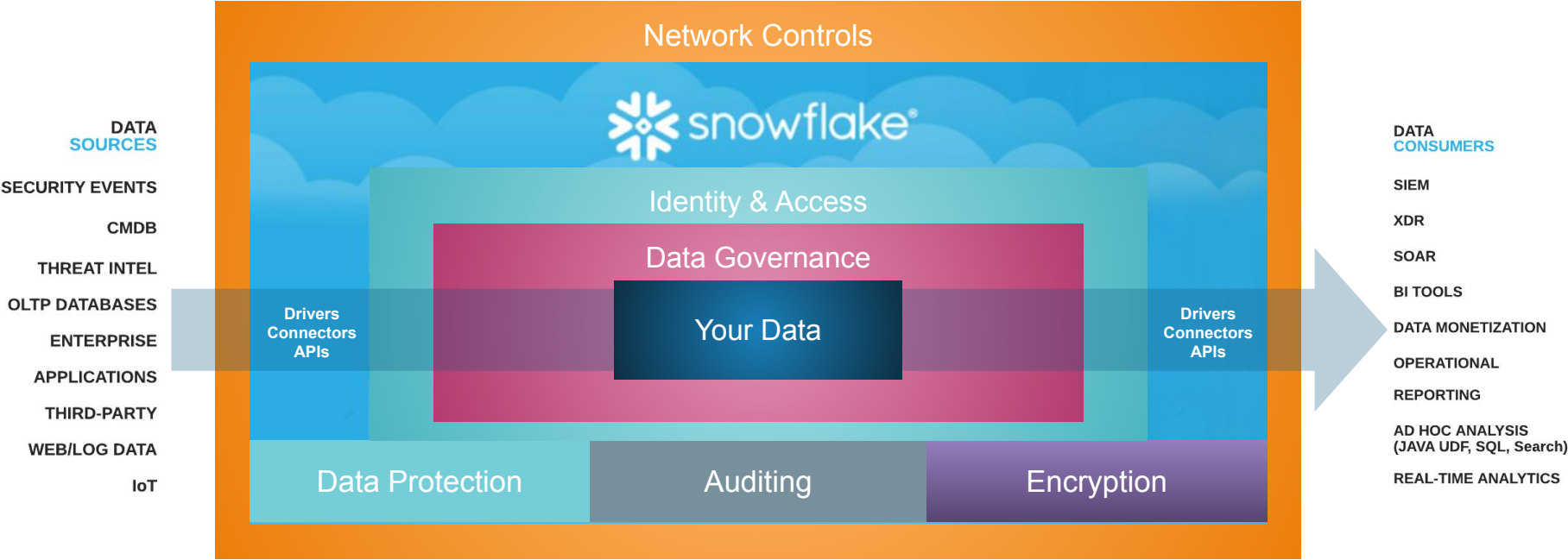
Snowflake CIS Benchmarks



Shared Responsibility Model

Area	Snowflake	Customer	Cloud Provider
 Platform Security	<p>Perform vulnerability patching for Kernel and host OS used by Snowflake services.</p> <p>Provide secure networking, identity and access, vulnerability management (not testing), "third party" security audits, bug bounty programs, communications with cloud providers, and threat and abuse detection program, vulnerability testing and audits, incident management, and transparent communication with customers.</p>	<p>Not applicable.</p>	<p>Ensure data center security, hardware trust and firmware updates, integrity of the OS, VMs, and other resources to make the platform secure.</p>
 Network Security	<p>Secure all VPC and VNETs and networking between Snowflake components and between Snowflake components and customer data (when hosted by Snowflake).</p>	<p>Follow best practices when configuring networks, such as allowlists and least privileged principle. Detect and respond to Snowflake account compromise due to factors within customer control (e.g., data compromised due to S3 bucket opened to internet).</p>	<p>Secure the networking layer.</p>

SNOWFLAKE SECURITY & GOVERNANCE AT A GLANCE



Achieve: Compliance benchmarks, Privacy goals

Snowflake Drivers & Connectors

- C / C++
- Golang
- Hive
- JDBC
- Kafka
- .NET
- Node.js
- ODBC
- PHP PDO
- Python
- Snowsight
- SnowSQL
- Spark
- SQLAlchemy
- Web UI
- SQL API

Common Connection Pattern

HTTPS/TLS



Common Access Control



Snowflake
VPC/VNet



NETWORK CONTROLS – SECURE COMMUNICATION

Common Connection Pattern for Drivers & Connectors

- Every [driver & connector](#) connects the same way
- All communication encrypted end-to-end
 - All customer data flows solely over HTTPS
 - Connections encrypted [using TLS 1.2](#) from client through to the Snowflake Service
 - HSTS in use for client communications
- Data encrypted at rest

Common Access Control for all Sessions

- IP allowlisting available to restrict client communication to specific IP addresses using [customer-configured Network Policies](#)
- Authentication required for all connections



NETWORK CONTROLS – NETWORK POLICIES

- Network Policies encapsulate an IP allowlist and an IP blocklist
- [Customer-configured Network Policy](#)

Network policies can be applied at three levels



1. Snowflake Account

- All traffic will use this policy, unless there is a more specific one.
- Control is applied at authentication time.



2. Outside Integration

- Applies to traffic at the integration endpoint only. For example: SCIM or OAuth security integrations.



3. User Specific

- Applies to the specific user only.
- Best practice for users used as service accounts.

The most specific policy always wins.

Edit network policy

MULTIVERSE as ACCOUNTADMIN

Enter a valid IPv4 address and optional CIDR or multiple addresses in a comma-separated list.

4 allowed IP addresses

3		2	×
7		26	×
1		.17	×
1		31	×

0 blocked IP addresses

No blocked addresses

Add addresses that will be blocked from accessing your Snowflake account

Comment (optional)

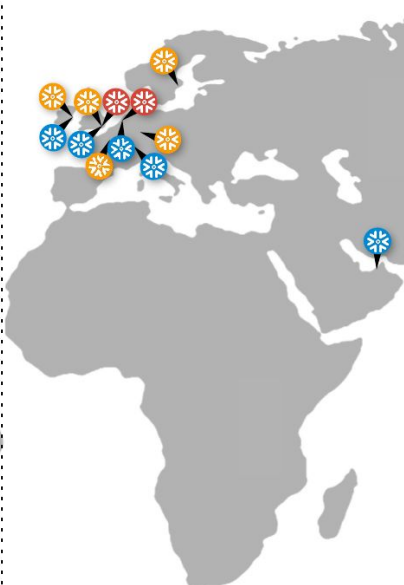


NETWORK CONTROLS – ANY SUPPORTED REGION

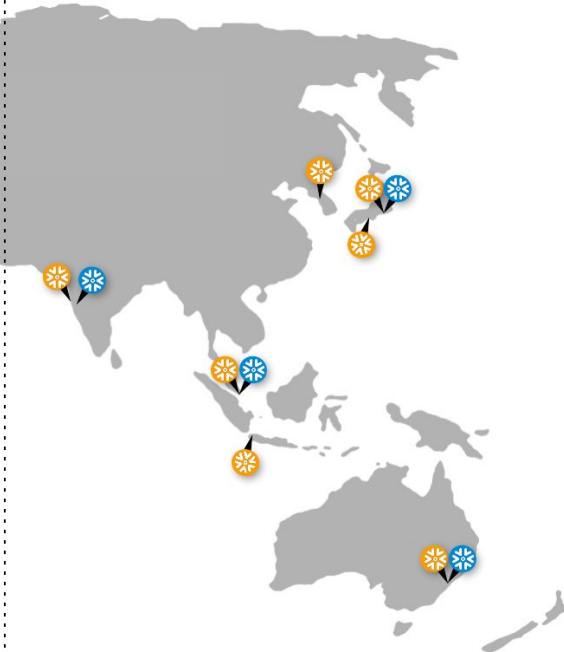
North & South America



Europe & Middle East



Asia Pacific



[Most up-to-date list of Snowflake regions](#)



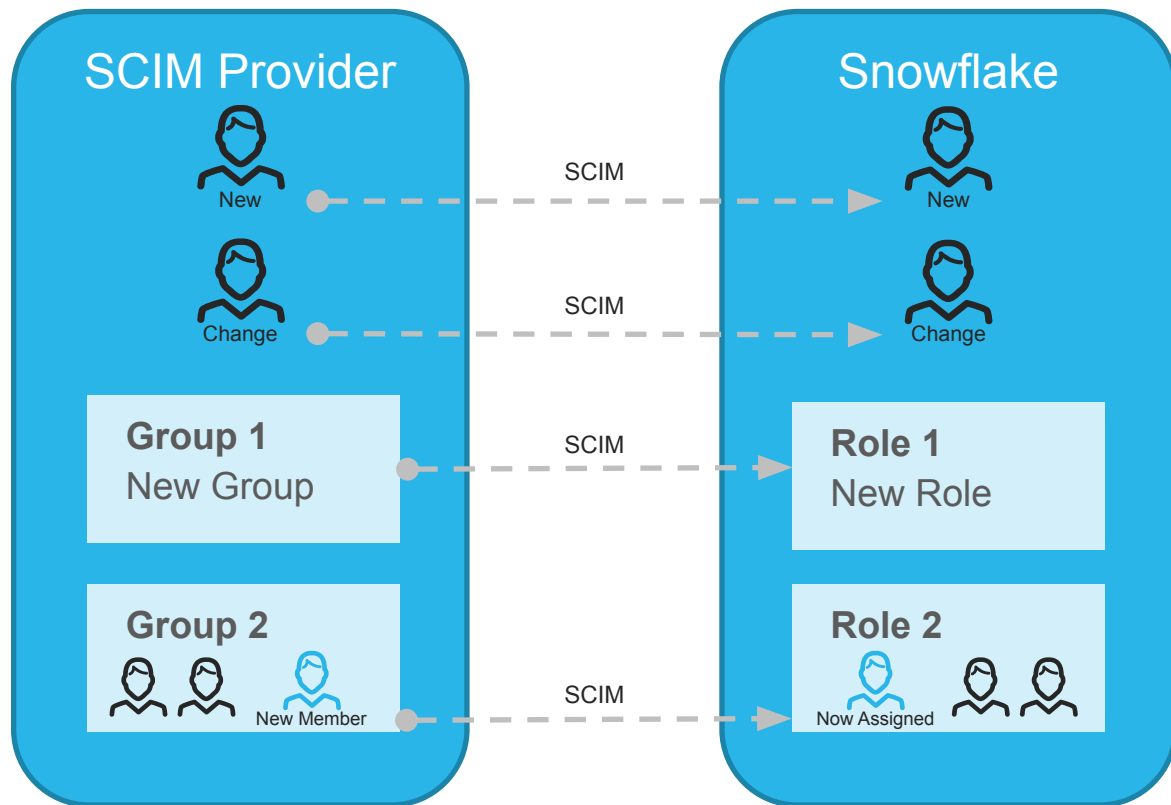
SNOWFLAKE USER PROVISIONING

Users & Roles Must Exist in Snowflake Before Authentication

- User objects are created before they authenticate
- User object attributes determine authentication (e.g. LOGIN_NAME will be the user visible name at login)

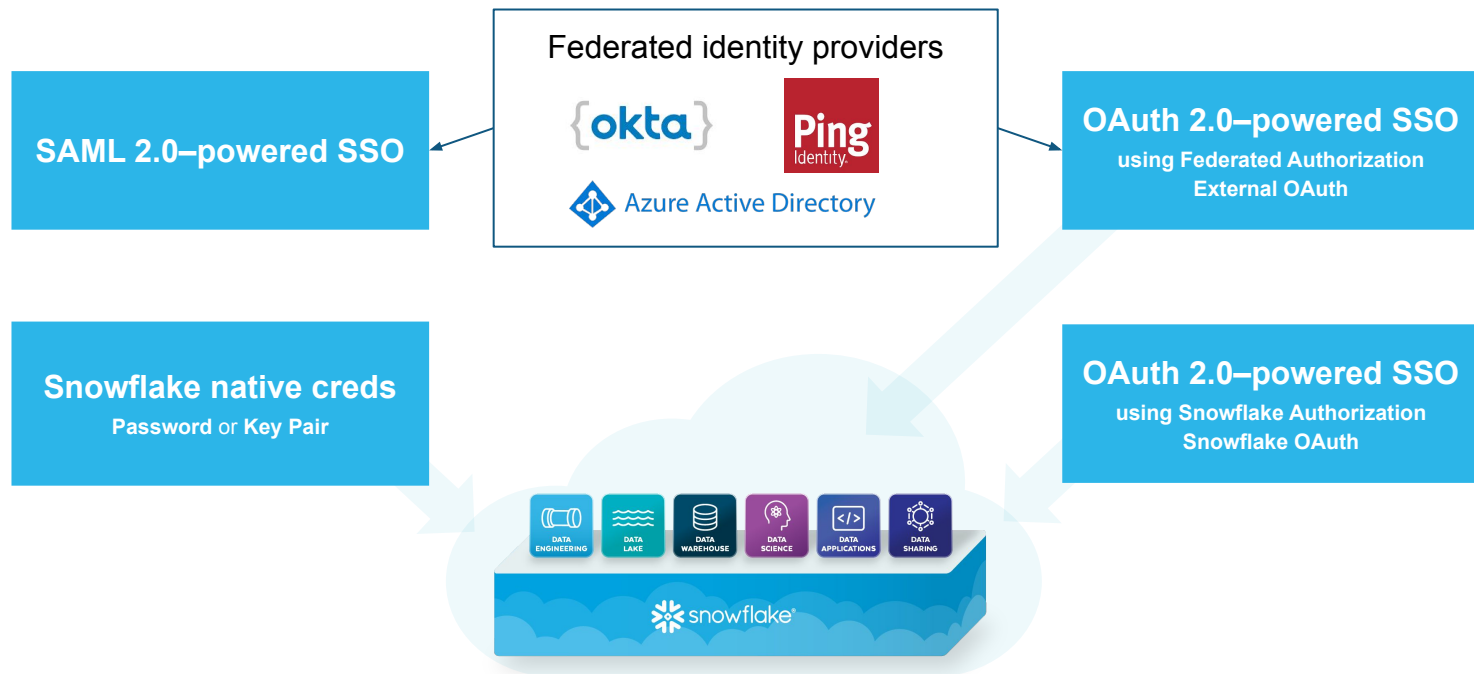
Put Your Authoritative Directory in Control with SCIM

- User creation, changes, & deletes
- Roles driven by group membership
- Use Okta, Azure AD, or any system that speaks [SCIM](#), or...
- Any system that can [use SQL](#)



SNOWFLAKE AUTHENTICATION

How to do Authentication & Delegated Authorization for Snowflake



- Three different federated concepts:
 - [SCIM](#)
 - [SAML](#)
 - [OAuth](#)
- Users may have multiple authN & authZ methods all configured at once
- [Browser-based SSO](#) (aka “[External Browser](#)” mode) can bring SSO to desktop apps that do not natively support it

Please note: Partners shown are a sample list and not the full list of supported platforms.



CONTROLLING PASSWORDS & SESSIONS THROUGH POLICIES

Password Policies

- [Control min/max length, character requirements \(upper, lower, number, special\), age, retries, and lockout times](#)
- Apply at account or user levels; user-level overrides account-level
- Works alongside other Snowflake user level settings
 - Can still reset individual user passwords
 - No effects on federated user properties

Session Policies

- [Control idle time for sessions](#)
- Apply at account or user levels; user-level overrides account-level
- Consider the interaction with client side behaviors and settings (e.g. [CLIENT_SESSION_KEEP_ALIVE](#))

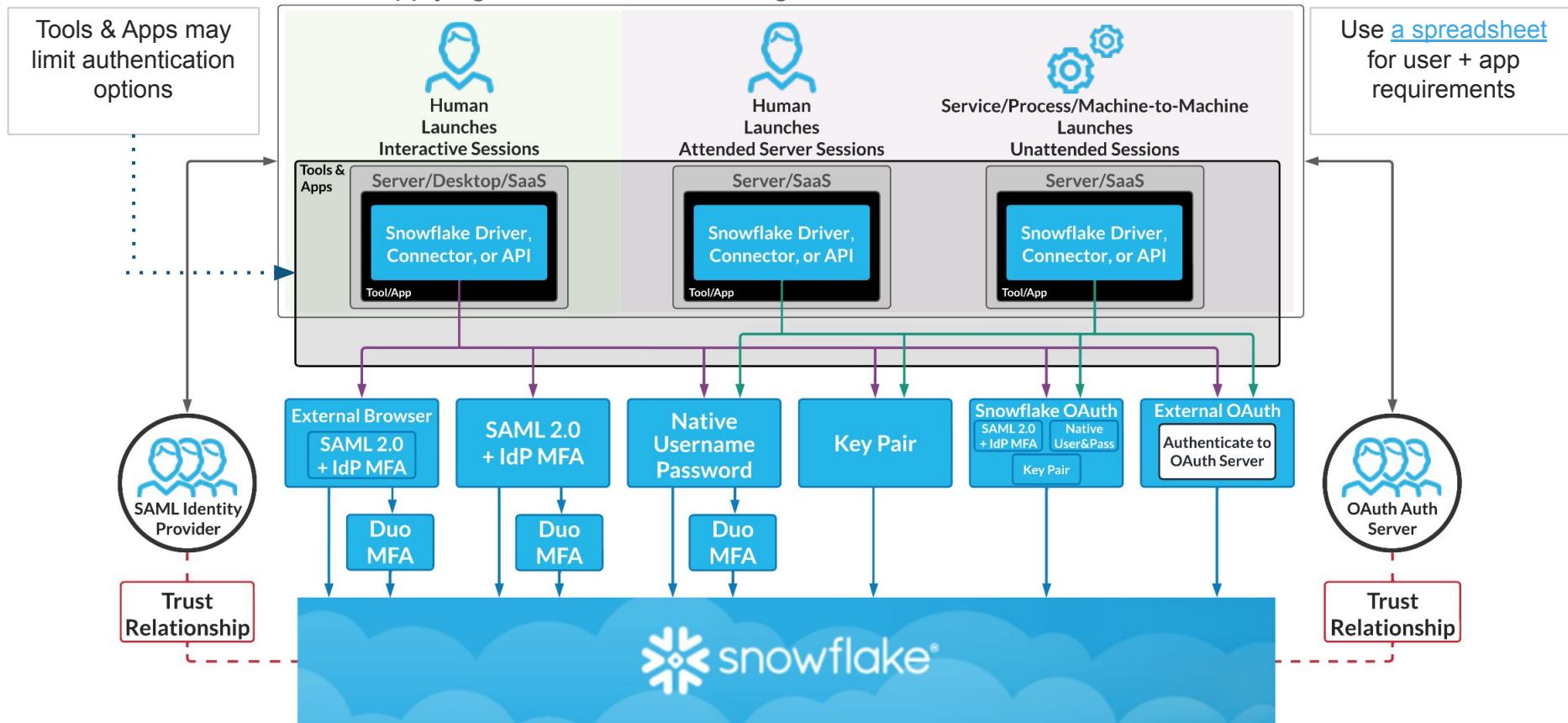
Policies mean ease of management

- Policies are objects (like tables or views) and can be centrally managed
- Apply a single policy to multiple objects
- Built-in separation of duty: policy admins assign and users are subject to policy controls
- All application and use is fully audited



SNOWFLAKE AUTHENTICATION PATTERNS

Applying Authentication & Delegated Authorization for Snowflake



SERVICE / FUNCTIONAL ACCOUNT USERS

- There are no special user principles in Snowflake
 - A service account will be the same kind of principle as everything else
 - All policies and attributes will behave the same
- There should be stricter rules & more process for these user objects
 - Good candidates for [user-level Network Policy](#)
 - Consider how Password Policies, Session Policies & Authentication Policies should be applied
 - Use strict RBAC (least privilege)
 - Consider integrating with privileged identity / account management solutions
- The activity of these users should be regular and predictable in most cases
 - Monitor all SQL closely and have a low threshold for suspicious activities
- When possible, use individual users versus shared / service / functional accounts for best auditing

SNOWFLAKE SERVICE ACCOUNT SECURITY GUIDANCE

- Part 1: [Defining the threats](#)
- Part 2: [Understanding the mitigations for the threats](#)
- Part 3: [Using Hashicorp Vault credential rotation for Service Accounts](#) (See [Hashicorp's new, updated Snowflake support](#) with more options and key pair features)



SNOWFLAKE GOVERNANCE



SNOWFLAKE GOVERNANCE CAPABILITIES



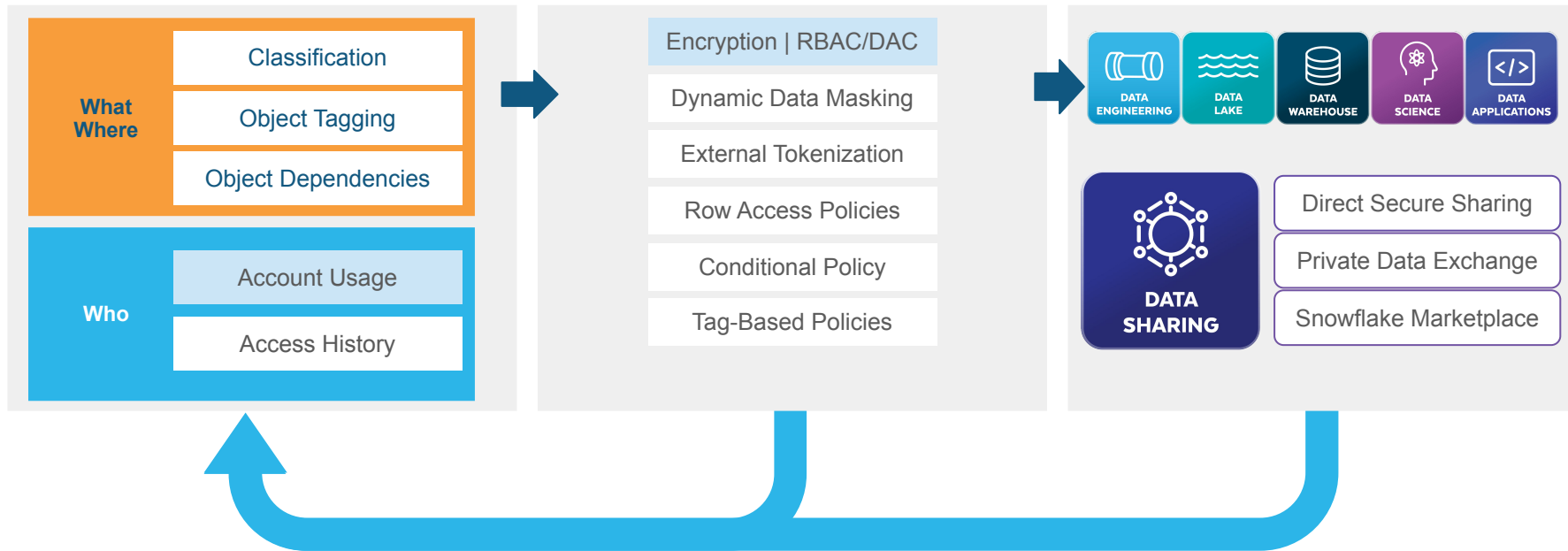
Know Your Data



Protect Your Data



Unlock Your Data



FAMILYNAME	GENDER	AGE	ZIPCODE	PHONE	OPTIN
Romero	F	56	31675	4282478462	Y
Polk	N	52	16971	7565697078	U
Gaines	F	61	28877	4210779144	Y
Gordon	M	36	85115	6474995638	N
Hammer	F	63	50587	5947720813	Y

We start with a table

- The table has been instantiated from the encrypted, at-rest files (micro-partitions)
- The information in the table is opaque to Snowflake
- This table contains data “in the clear,” but you might load data that’s been modified in some way to protect it as well

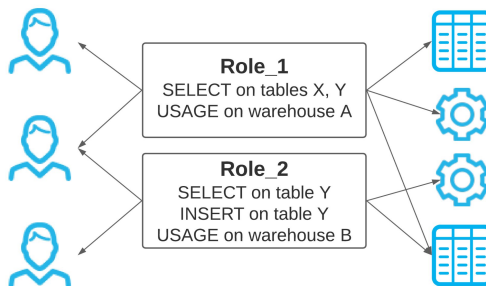


RBAC & DAC

Views & UDFs

name	age	zip_code
Polk	5-	169**
Gaines	6-	288**
Gordon	3-	851**
Hammer	6-	505**

FAMILYNAME	GENDER	AGE	ZIPCODE	PHONE	OPTIN
Romero	F	56	31675	4282478462	Y
Polk	N	52	16971	7565697078	U
Gaines	F	61	28877	4210779144	Y
Gordon	M	36	85115	6474995638	N
Hammer	F	63	50587	5947720813	Y

**RBAC & DAC protect the table**

- Every object in Snowflake is subject to these controls, and they are at the whole-object level
- RBAC inheritance and other RBAC features apply
- The customer controls RBAC completely
- DAC (Discretionary Access Control) applies to the role that owns the object, unless the object is subject to [Managed Access Schema](#)

You may also create [Views](#) & [UDFs](#)

- These are mostly used to redact or transform rows, columns, or even cells, and create a new object
- The new object has RBAC and DAC controls



RBAC & DAC

Views & UDFs

Column-Level Security

Row Access Policy

Conditional Policy

We can use Policy controls for Columns, Rows, and Fields

- Prevent View/UDF explosion
- Table/View owners and privileged users (such as ACCOUNTADMIN) unauthorized to data by default
- Ensure controls are applied in any context where the object's data is used

We get more ease of management

- Centrally manage policies
- Apply a single policy to multiple tables
- Built-in separation of duty: policy admins assign, and users are subject to policy controls
- All application and use is fully audited

FAMILYNAME	GENDER	AGE	ZIPCODE	PHONE	OPTIN
Romero	F	56	31675	4282478462	Y
Polk	N	52	16971	7565697078	U
Gaines	F	61	28877	4210779144	Y
Gordon	M	36	85115	6474995638	N
Hammer	F	63	50587	5947720813	Y



Dynamic Data Masking

RBAC & DAC

Views & UDFs

Column-Level Security

Dynamic Data Masking

We can leverage Column-level Security to dynamically mask data at query time

- No change to the stored data
- Mask or partially mask using constant value, hash, and custom functions
- Unmask for authorized users only

FAMILYNAME	GENDER	AGE	ZIPCODE	PHONE	OPTIN
Romero	F	56	31675	4282478462	Y
Polk	N	52	16971	7565697078	U
Gaines	F	61	28877	4210779144	Y
Gordon	M	36	85115	6474995638	N
Hammer	F	63	50587	5947720813	Y

Query results

phone	name
_-5534	** masked **
_-3564	** masked **
_-9787	** masked **



Alex
(Unauthorized)

Query results

phone	name
408-123-5534	** masked **
510-335-3564	** masked **
214-553-9787	** masked **



Morgan
(Partially Authorized)

```
create or replace masking policy F00
as (val string) returns string ->
case
  when is_granted_to_invoker_role('SEECLEAR')
  then val
  when current_role('ONLYPART')
  then regexp_replace(val, '[0-9]', '*', 7)
  when is_role_in_session('CRYPTO')
  then decrypt_raw(val, keyval, IV, ...)
  when is_role_in_session('BESPOKE')
  then user_defined_Func(val, baz, ...)
  else '** masked **'
end;
```



RBAC & DAC

Views & UDFs

Column-Level Security

External Tokenization

FAMILYNAME	GENDER	AGE	ZIPCODE	PHONE	OPTIN
Romero	F	56	31675	awiufyaf873fg	Y
Polk	N	52	16971	78ybhsbbcvzd	U
Gaines	F	61	28877	984iuwrfgjffss	Y
Gordon	M	36	85115	ciudsjhciasudg	N
Hammer	F	63	50587	8347ryfgvgshf	Y

Example using policy:

```
create or replace masking policy BAR as (val
string) returns string ->
case
  when is_granted_to_invoker_role('SEETOKENS')
  then val
  when current_role('GETREAL')
  then detok_ext_func(val, current_user(),...)
  else '** masked **'
end;
```

Example using SQL outside policy:

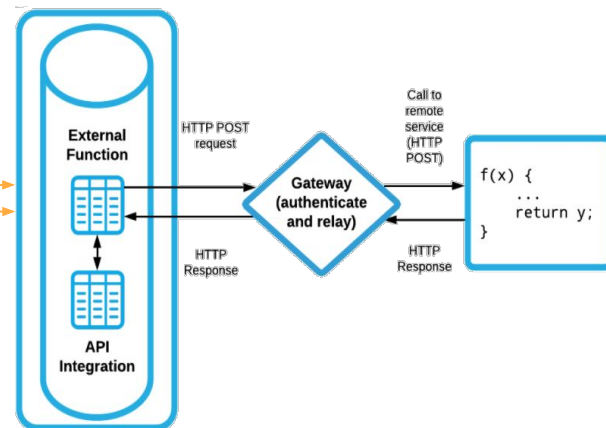
```
SELECT detok_ext_func(T1.phone) AS REAL_PHONE
, T1.GENDER
, T2.ZIP
FROM T1
JOIN T2
ON T2.PHONE = T1.PHONE
```

Ingest protected (PII/PHI) data as Externally Tokenized

- Using tokenization provider functionality upstream from Snowflake

De-tokenize for authorized users at query time

- The tokenization provider is called using a Snowflake External Function to de-tokenize data
- For unauthorized users, third-party service is not called
- Can be used in policy or outside



Row Access Policy

RBAC & DAC

Views & UDFs

Column-Level Security

Row Access Policy

FAMILYNAME	GENDER	AGE	ZIPCODE	PHONE	OPTIN
Romero	F	56	31675	4282478462	Y
Polk	N	52	16971	7565697078	U
Gaines	F	61	28877	4210779144	Y
Gordon	M	36	85115	6474995638	N
Hammer	F	63	50587	5947720813	Y

Query results

zip_code	name
31675	**masked**



Alex
(Only 31675)

Query results

zip_code	name
28877	**masked**



Morgan
(Only 28877)

Filter rows at query time based on user role and lookup table

- Policy contains condition(s) to allow or filter out rows
- Policy is applied to one or more table, view, or external table in an account
- Dynamically generated predicate filters out rows the user is not authorized to see at query time
- Can be combined with other controls

```
create or replace row access policy F00
as (this_zip varchar) returns boolean ->
  'all_seeing_role' = current_role()
or
  exists (
    select 1 from zip_mapping_table
    where info_reader = current_role()
    and zip_code = this_zip);
```

Data Policies Performance Boost (Memoizable Functions)

Boost performance of complex policies with many mapping table lookups

```
create or replace row function
  allowed_sales_regions() memoizable
returns array as $$ ...
```



Conditional Policy

RBAC & DAC

Views & UDFs

Column-Level Security

Row Access Policy

Conditional Policy

Query results



All Users

Conditional policy is a type of Column-level Security policy to dynamically mask data at query time in *one column* based on *the value of another*

- Row-level policy for column access
- No change to the stored data
- Same properties as other column-level policies

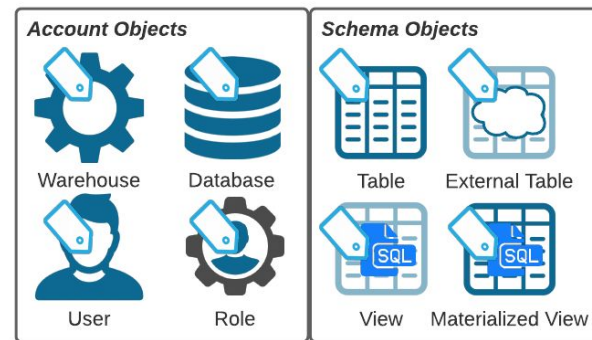
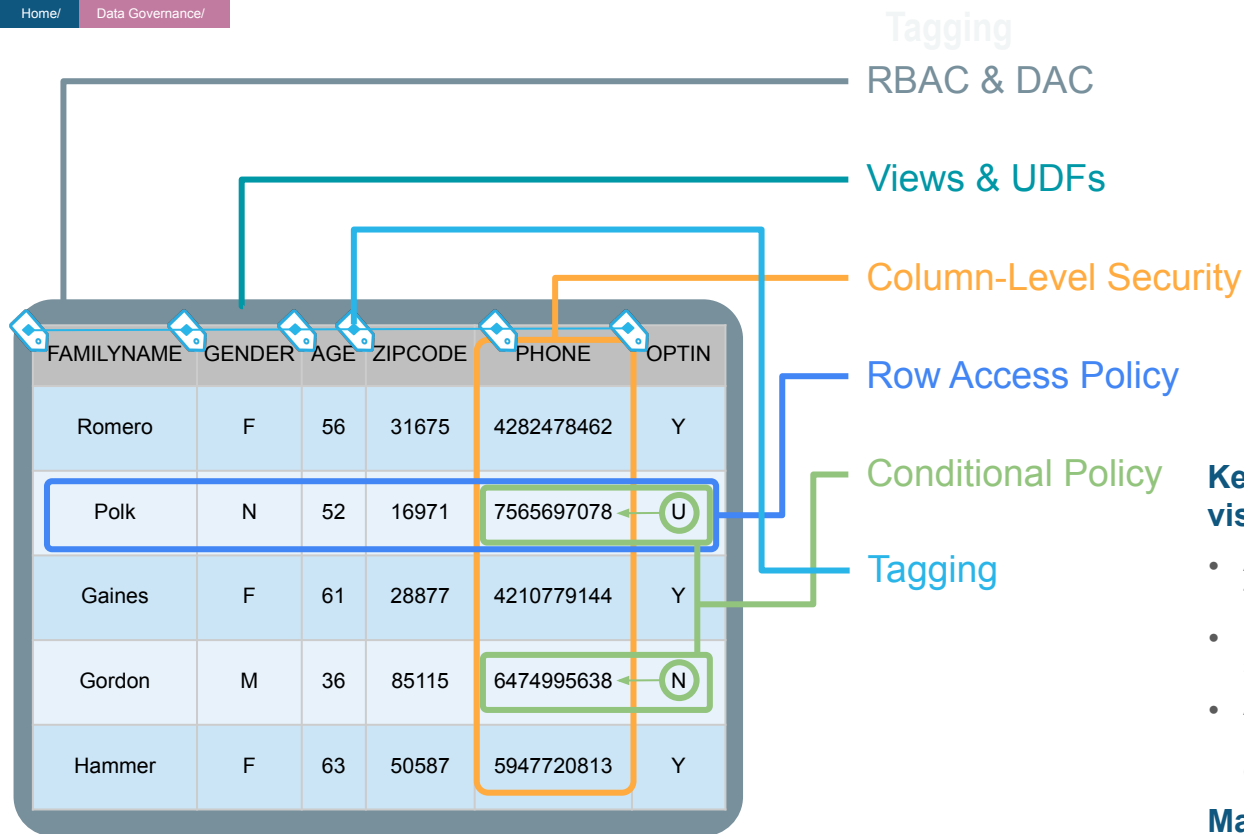
```
create or replace masking policy BAZ
as (val string, optin string) returns string ->
case
  when optin = 'Y' then val
  else '***MASKED***'
end;
```

```
alter table start_with_a_table modify column PHONE set
masking policy BAZ using (PHONE, OPTIN);
```

FAMILYNAME	GENDER	AGE	ZIPCODE	PHONE	OPTIN
Romero	F	56	31675	4282478462	Y
Polk	N	52	16971	7565697078	U
Gaines	F	61	28877	4210779144	Y
Gordon	M	36	85115	6474995638	N
Hammer	F	63	50587	5947720813	Y

phone	name
MASKED	Polk
4210779144	Gaines
MASKED	Gordon





Keep track of sensitive data for visibility and compliance

- Assign tags to sensitive columns, tables, external tables, or views
- Easily audit sensitive objects without appropriate security policies
- Assign tags to virtual warehouses, Snowpipe, materialized views, and clustered tables to keep track of resource usage for cost visibility and attribution

Manage tags with flexible administration models

- Centralized tag creation and assignment for centrally managed governance
- Decentralized tag assignment controlled by privileges for object owner-supplied tag value



Tag Based Policy

RBAC & DAC

Views & UDFs

Column-Level Security

Row Access Policy

Conditional Policy

Tagging

Query results



Note that the directly applied Conditional policy overrides the Tag-based policy

phone	age	optin	name
MASKED	88,888,888	U	Polk
4210779144	88,888,888	Y	Gaines
MASKED	88,888,888	N	Gordon

Policy1

Policy2



Column1

Column2

Tag-based policy allows you to assign a policy to a tag instead of a specific object.

- Assign policy based on the metadata tracked in tags
- Assign policy to untagged columns to mask until scanned and curated
- Most specifically-assigned policy for any column wins (column policy overrides tag policy)

```
create tag MASKME;
```

```
create masking policy string1 as (val string) returns string ->
case when current_role() in ('YUP') then val
else '** masked **' end;
```

```
create masking policy numb1 as (val int) returns int ->
case when current_role() in ('YUP') then val
else 88888888 end;
```

```
alter tag MASKME set masking policy string1,
masking policy numb1;
```

```
alter table start_with_a_table modify column PHONE set tag
MASKME = 'YUP';
```

```
alter table start_with_a_table modify column AGE set tag
MASKME = 'YUP';
```



Classification

RBAC & DAC

Views & UDFs

Column-Level Security

Row Access Policy

Conditional Policy

Tagging

Classification



FAMILYNAME	GENDER	AGE	ZIPCODE	PHONE	OPTIN
Romero	F	56	31675	4282478462	Y
Polk	N	52	16971	7565697078	U
Gaines	F	61	28877	4210779144	Y
Gordon	M	36	85115	6474995638	N
Hammer	F	63	50587	5947720813	Y

Classify columns containing personal data

- Automatically detect columns with personal data
- Apply Snowflake-defined semantic and privacy category system tags
- Assign data access policies to columns based on system tags
- Use system tags to audit personal data among millions of columns
- Seamlessly integrates across Snowflake's governance capabilities and Data Sharing
- Populate and manipulate tags using third-party GRC, MDM, and other classification solutions

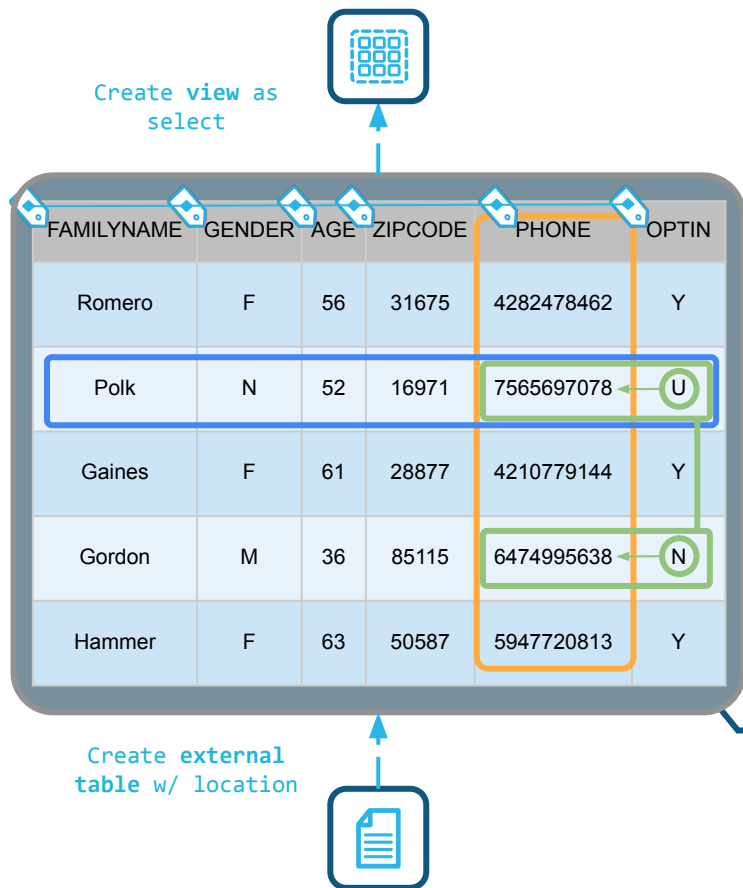


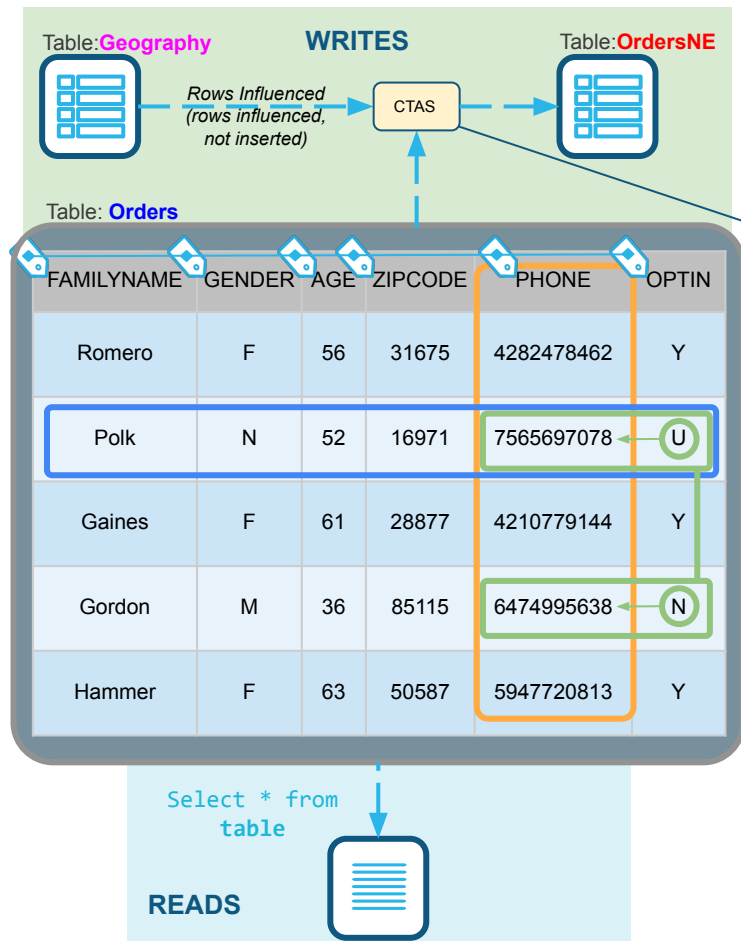
Impact analysis as objects are built. Determine data provenance.

- Identify all upstream source tables of a desired view to satisfy regulatory compliance, such as GDPR, CCPA, HIPAA, PCI.
- Proactively identify impacted downstream views before making metadata changes referenced in views and tables.
- Identify source tables and views of desired view to verify trustworthiness and popularity of the source.

Object Dependencies answers difficult questions:

- Where did the data come from?
- What data is impacted by other objects?
- Where is the data located?
- What does the data produce?
- What does the data consume?
- What associations are there between data elements?
- What data sources should we use to develop new customer experience initiatives?
- What data in my enterprise needs to be brought into compliance with industry regulations?
- Where do we have risk exposure that needs to be mitigated?
- Who or what business process changed the data?
- How can data scientists improve confidence in the data they need for advanced analytics?
- What tool or technology made the change?

**Object Dependencies**



User	ROWS INFLUENCED		COLUMN LINEAGE	
	OBJECTS_ACCESSED	OBJECTS_MODIFIED	SOURCE_COLUMN_NAME	TARGET_COLUMN_NAME
Taylor	Orders <ul style="list-style-type: none"> FAMILYNAME GENDER Geography <ul style="list-style-type: none"> Region 	OrdersNE <ul style="list-style-type: none"> FAMILYNAME GEN 	Orders .FAMILYNAME Orders .GENDER	OrdersNE .FAMILYNAME OrdersNE .GEN

Create table **OrdersNE** (FAMILYNAME, GEN) as select FAMILYNAME, GENDER from **Orders** JOIN geography ON **Geography**.region = **Orders**.zipcode where **Geography**...

Access & Lineage as it happens.

- Deep understanding of the use and origins of sensitive information.
- Determine popular and unused objects to [the column level](#).

Access History gives answers in two ways:

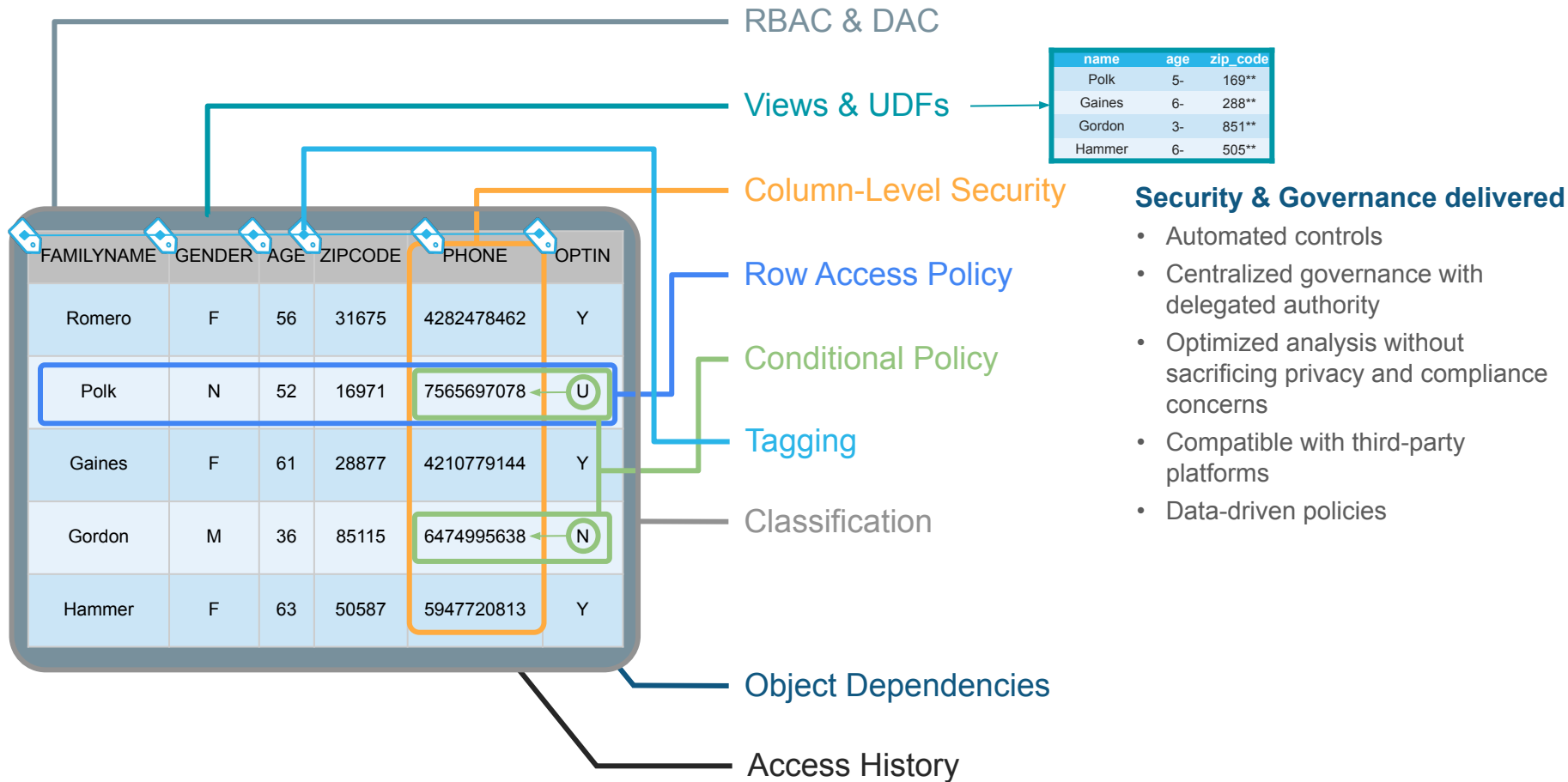
(READS) SELECT QUERIES

- What tables are queried the most?
- When & who last queried a table?
- What tables users accessed the most?
- What tables are commonly used together?
- What columns are used the most?
- What columns are not used at all?
- What tables have the overall slowest queries?
- Who saw results from what column, what table, when and where?

(WRITES) MANIPULATION QUERIES

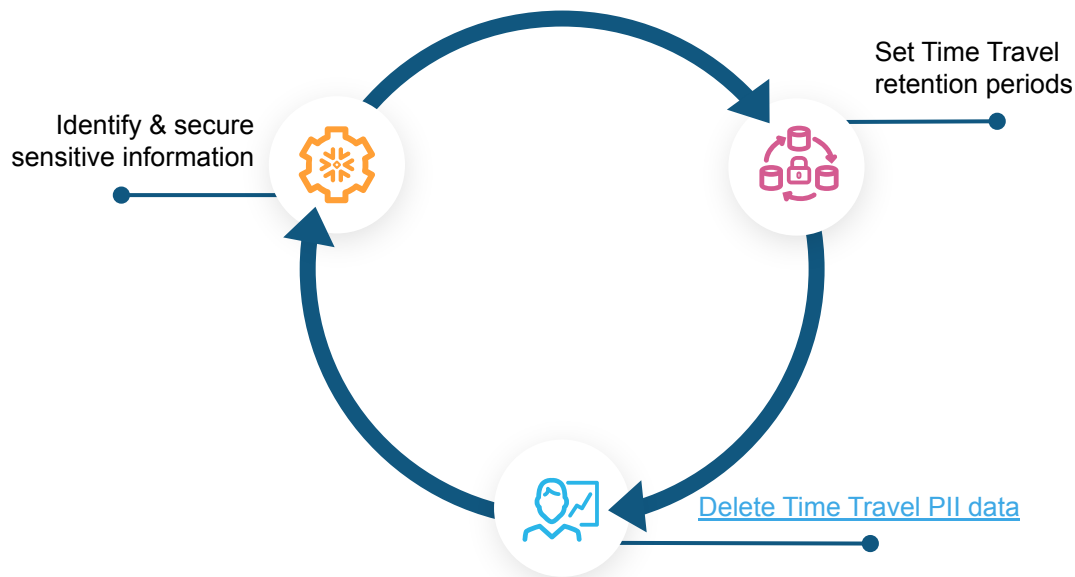
- What tables act as origins for other tables
- What are the most common columns used in the WHERE clause?
- What are the most common columns used in the JOIN clause for a table?
- Are these predicates in line with the table clustering keys?
- When were tables created, when were tables altered, how were tables altered?





TIME TRAVEL & FAIL-SAFE

GDPR, CPRA, and other emerging regulations allow individuals to request the deletion of their personal information, unless exceptions apply. This means your Snowflake recovery policies must properly align to your organization's compliance policies.



Snowflake provides product features for customers to meet the demands of data privacy regulations.

- Time Travel & Fail-safe have data retention implications
- Up to 90 days (Time Travel)
- 7 days (Fail-safe)
- For PII erasure requests, you must consider Time Travel and its setting

[More on Time Travel & Fail-safe](#)



DATABASE REPLICATION & FAILOVER

1 Cross-Cloud & Cross-Region Replication

- Business continuity & disaster recovery
- Secure data sharing across regions/clouds
- Data portability for account migrations

2 Zero Performance Impact on Primary

- Asynchronous replication

3 Reduced Data Loss

- Incremental refreshes

4 Instant Recovery

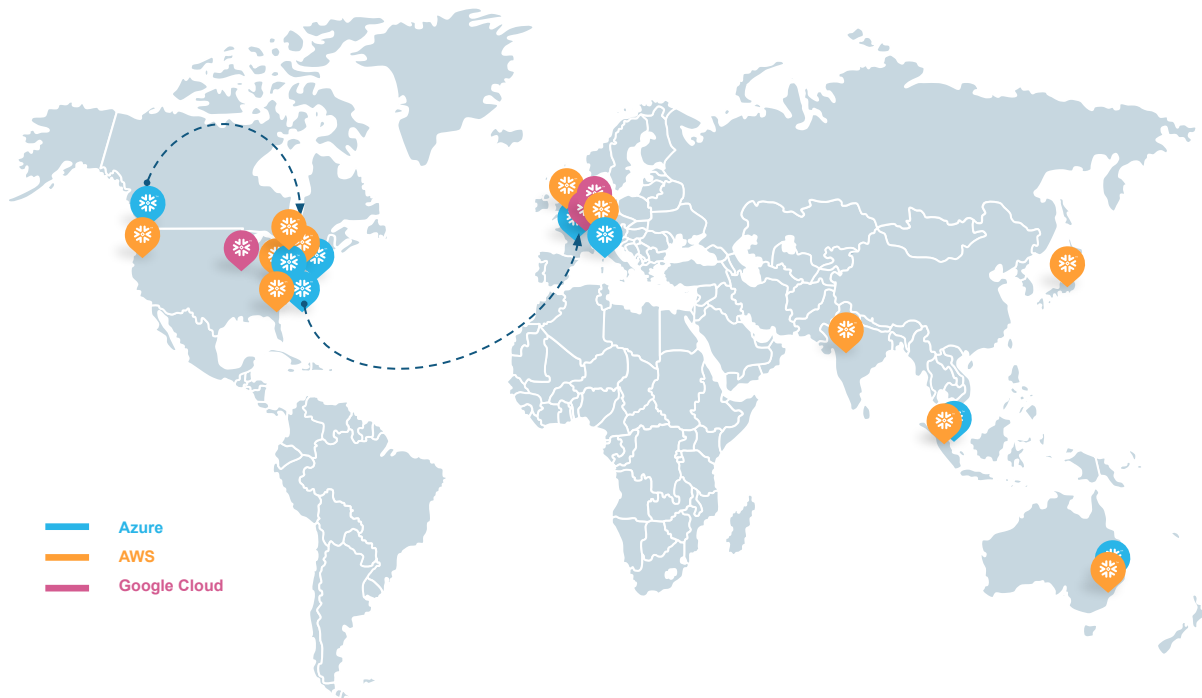
- Read: Readable secondary databases
- Write: Database failover

5 Secure

- Data encrypted at-rest & in-transit
- Tri-Secret Secure compatible

6 Cost Effective

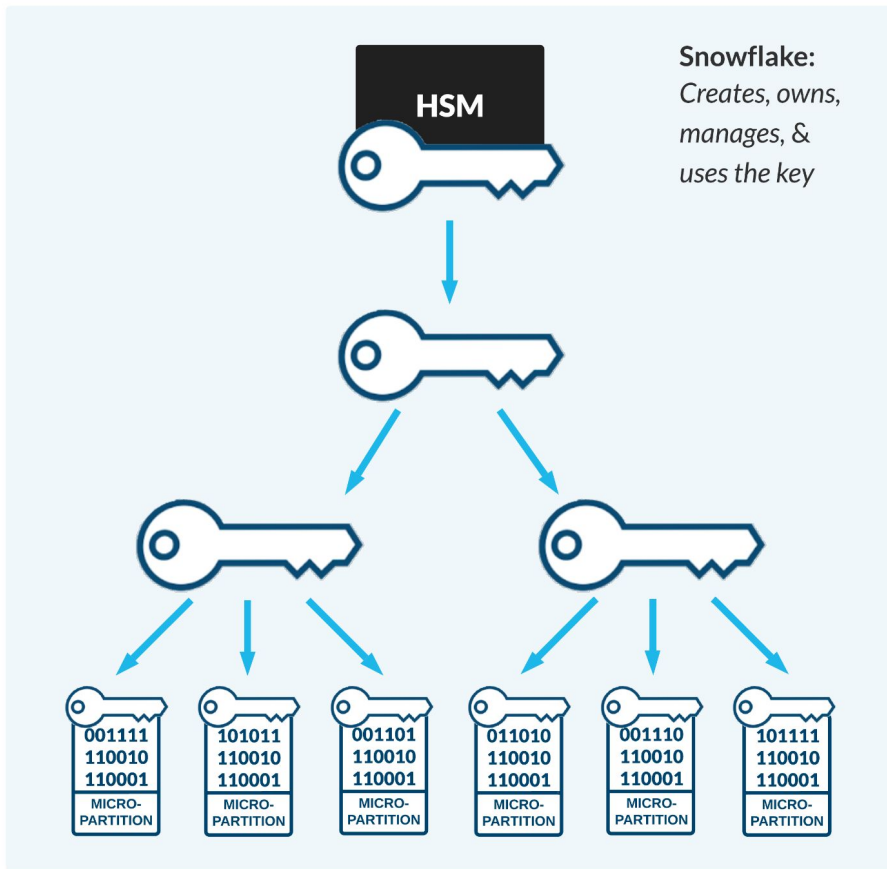
- Replication costs: Data transfer & compute (serverless)
- Control which databases to replicate



[More about Database Replication & Failover](#)



HIERARCHICAL KEY MODEL

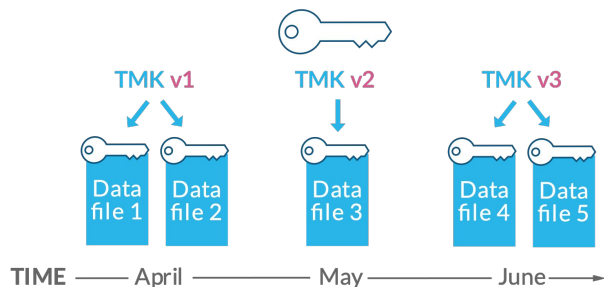


Hierarchical Key Model

- Hierarchical key model rooted in the CSP's HSM
 - GCP: [Cloud HSM](#)
 - AWS: [Cloud HSM](#)
 - Azure: [Dedicated HSM](#)
- All data at rest is encrypted by default, with no configuration required

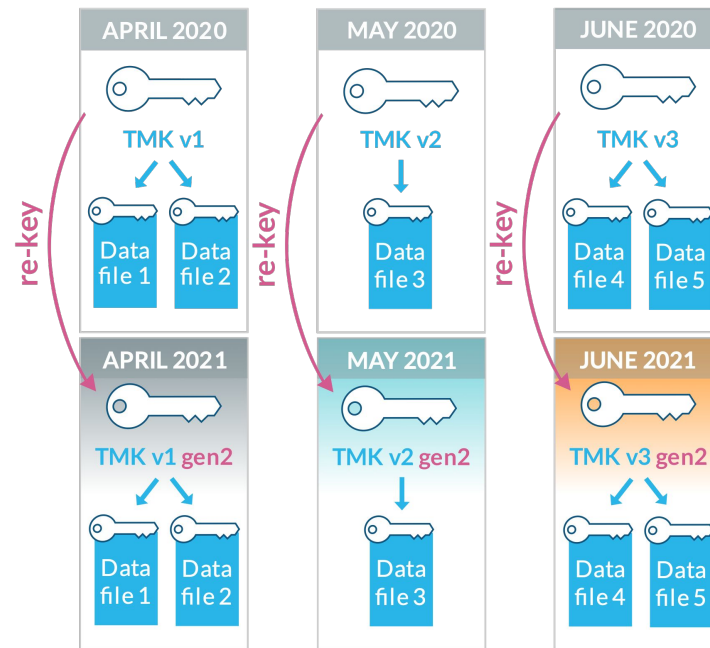


KEY ROTATION & RE-KEYING



Key Rotation

- Snowflake rotates keys every 30 days
- Process is transparent to customer and queries

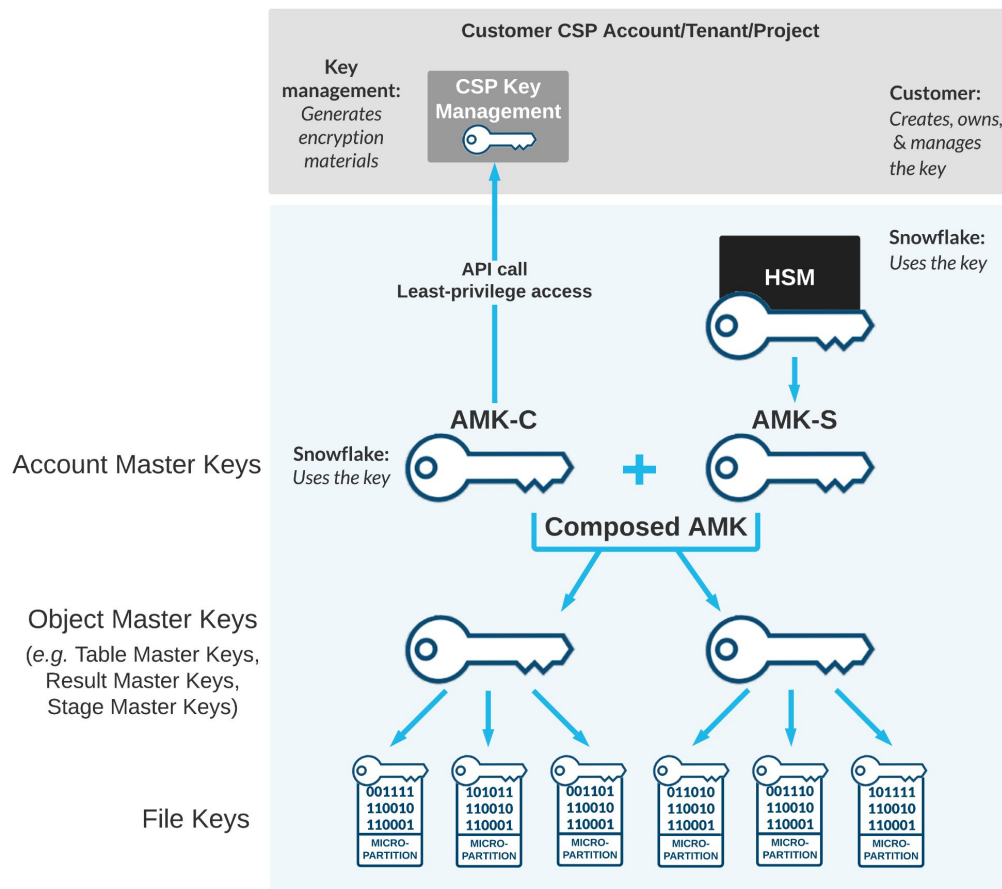


Key Re-Keying

- Yearly re-keying re-encrypts data on the key's birthday
- Re-keying requires Enterprise Edition or better
- Process is transparent to customer and queries



TRI-SECRET SECURE KEY MODEL



Hierarchical Key Model using Tri-Secret Secure

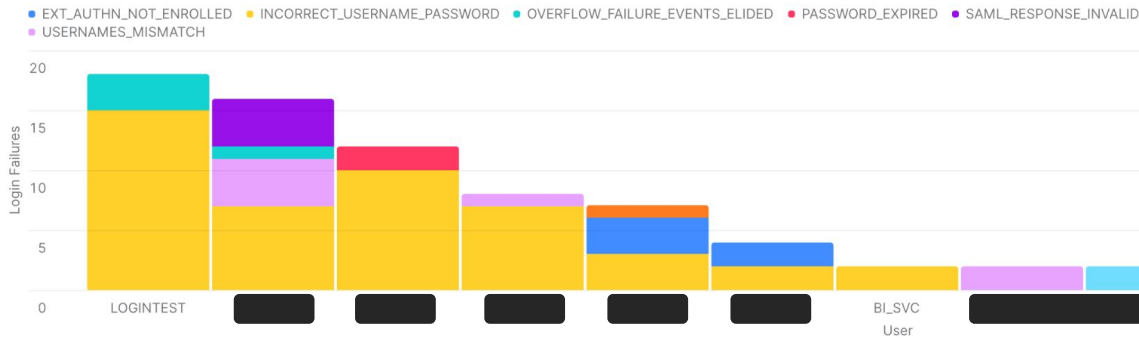
- Hierarchical key model adds a hybrid HYOK & BYOK model to give the customer control
- Customer holds key in their CSP Key Management and brings key materials to Snowflake to be part of the key-encrypting key (the Account Master Key or AMK)
- CSP-supported key managers:
 - GCP: [Cloud KMS](#)
 - AWS: [AWS KMS](#)
 - Azure: [Key Vault](#)



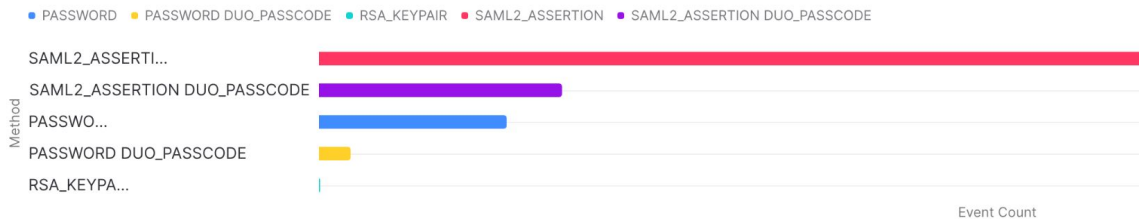
AUDIT LOGGING – ACCOUNT USAGE

- [Auditing tracks every user's activity at all times in full detail](#)
 - Access control [using built-in database roles](#)
- Kept in a customer tamper-proof area of your account for 365 days
- [All drivers and connectors also have extended logging](#)
- We have materials to help
 - *Security Features Checklist Parts 2 & 3*
 - [Security Dashboards \(i.e. project "Sentry"\)](#)

Authentication: Failures, by User and Reason



Authentication: Breakdown by Method



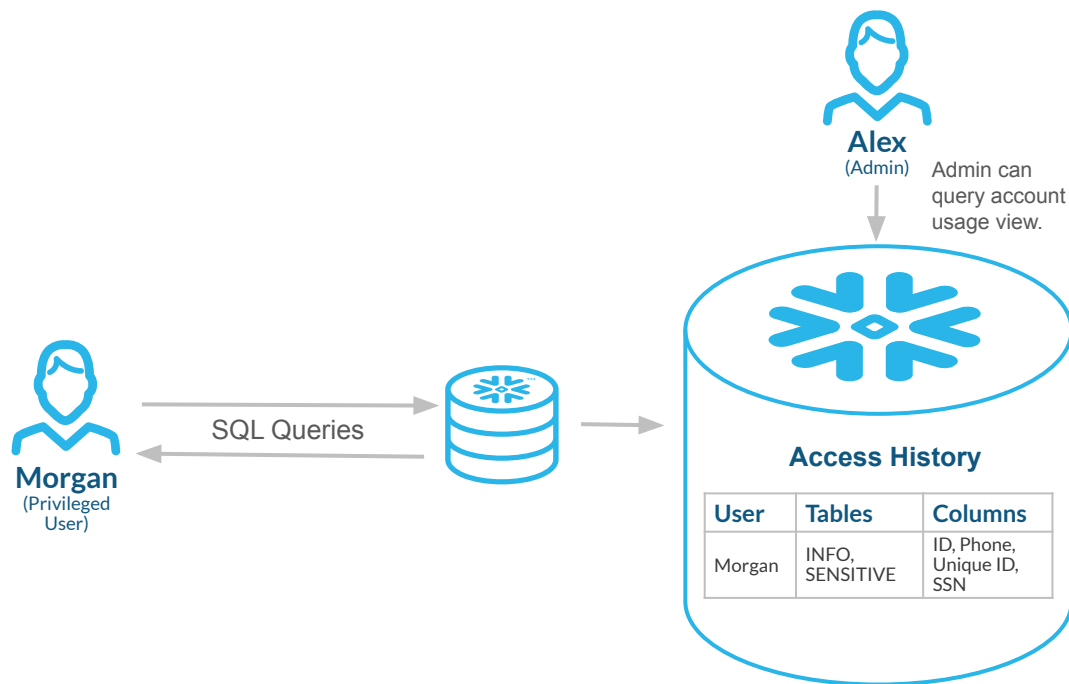
AUDIT LOGGING – ACCESS HISTORY

Access History supplies audit data access to comply with regulatory requirements and data governance initiatives.

- Access log of the tables, views, and columns each query accesses
- Includes base objects (e.g. table serving a view) indirectly accessed by the query
- Data access history available for easy reporting as a Account Usage View

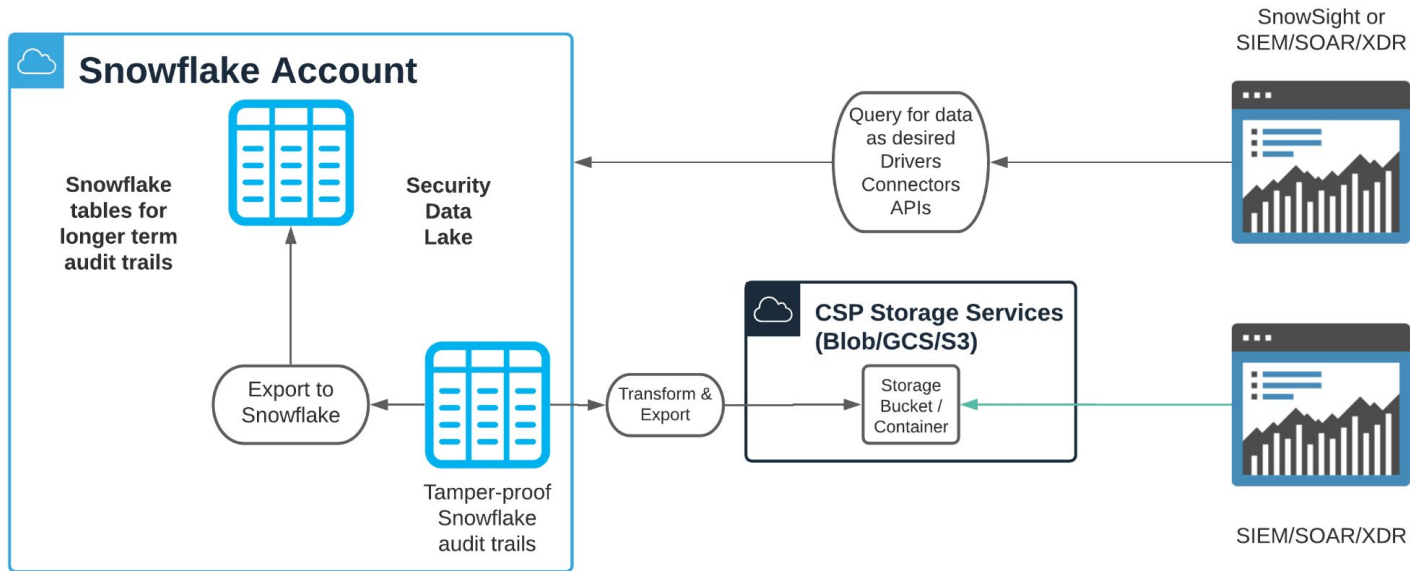
Discover unused data to determine whether to archive or delete the data.

Notify users prior to altering a table, view, or column.



AUDIT LOGGING – EXPORTING AUDIT LOGS

- Results can be further filtered using SQL predicates
- Export through JDBC or as JSON for use in SIEM



INFRASTRUCTURE SECURITY & MONITORING

How is the Snowflake Infosec Team monitoring the service?

Snowflake's internal Critical Security Controls dashboard provides real-time risk visibility

- Access Control, Security Assessment & Authorization, Configuration Management, Security Awareness, *etc.* all represented on a single Dashboard
- Real-time monitoring of data loaded into Snowflake from internal and other relevant data sources

Snowflake uses CIS benchmark templates for configuration hardening

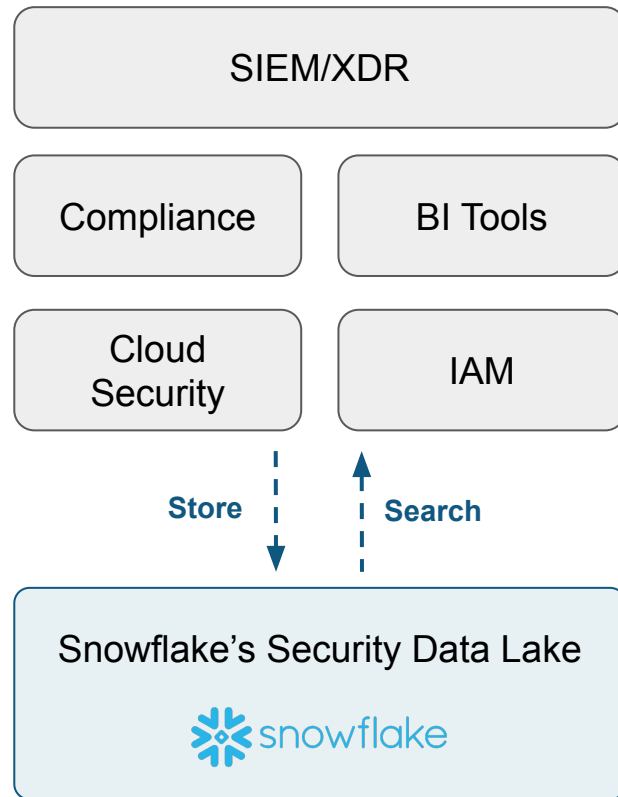
- Service configuration information is collected centrally in Snowflake
- Continuously and automatically tracked—unplanned changes cause alerts
- Part of the Snowflake Security Compliance Team's dashboard

Snowflake undergoes many 3rd-party penetration tests each year

- Comprehensive Web Application Penetration Test – Annually
- Internal Network Penetration Test – Annually
- Major Functionality Penetration Tests – As major functionality is released as part of the SDLC

Snowflake performs daily vulnerability scans on infrastructure

- Vulnerabilities are remediated per Security Policy
- Remediation trends tracked using Snowflake



GDPR – GENERAL DATA PROTECTION REGULATION

What is it?

- GDPR is an EU regulation that went into effect on May 25, 2018
- Governs the protection and processing of EU personal data

What does it mean in the context of Snowflake?

Different requirements apply to different types of entities:

- **Controller** – Snowflake customers are responsible for complying with GDPR independently from Snowflake
- **Processor** – Snowflake is responsible for the following:
 - Putting data processing addendums in place with our customers and our vendors
 - Only using our customers' EU personal data to provide our service to them
 - Being transparent about how we handle and process our customers' EU personal data on their behalf and keeping accurate records
 - Securing customers' EU personal data in our service
 - Facilitating our customers' compliance with data subject requests
 - Notifying customers about changes to our list of subcontractors

41

Snowflake responsibilities are documented in a [Data Processing Addendum](#) (DPA on snowflake.com/legal).

Available for signature now



USEFUL LINKS



- [Snowflake Security Overview and Best Practices](#)
- [Snowflake Security Product Documentation](#)
- [Managing Governance in Snowflake](#)





THANK YOU

