Not your Father's Data Lakehouse

Building a Data Lakehouse with Trino and Iceberg

Table of Contents

Lab 1: Introduction and setup

Learning objectives
- Describe the lab scenario and goals.
- Demonstrate how to set up a free Starburst Galaxy account.
- Understand how to continue using Starburst Galaxy after the end of the lab.

Activities
1. Lab overview
2. Create a Starburst Galaxy account
3. Housekeeping items

Part 1: Lab overview

You are a data engineer at Burst Bank. There's a new potential fraudulent scheme that is rising in popularity. Your team is tasked with identifying web logon actions in the month of March that may qualify as suspicious activity for the financial crimes team to investigate. You need to help by discovering, transforming, and cleaning the data from multiple sources. To do so, you will create a data pipeline in your data lakehouse and produce a curated data product.

Step 1 - Data information

Pretend the date is March 6th, 2024. This lab contains generated web logon actions that are ingested into AWS S3 for the first five days of March. Each dataset contains information about web logon actions that existed during the day to the Burst Bank website, including cust_key, login_timestamp, ip_address, and more.

```
{"custkey":1000597,"login_timestamp":"2024-03-01
21:33:06","ip_address":"55.248.111.217","device_type":"mobile","login_success":false,"
login_attempts":3,"login_date":"2024-03-01"}
```

The ingested data is partitioned in AWS S3 by day.

Importantly, you do not have any information about the specific customers, which is necessary for the final output. This is contained in separate lookup tables.

Step 2 - Description of activities

To make sense of data from multiple sources, you will need to federate across Postgres and AWS S3. To do this, you will begin by creating a data lakehouse using Trino and Iceberg. You will use Starburst Galaxy to read the data in the land layer via schema discovery, then clean and optimize that data into more performance Parquet files and Iceberg table format in the structure layer. In the last step, you will join the Customer information from Postgres with the web logon action information in AWS S3 into a single table that is cleaned and ready to be utilized by our teams. After completing the discovery, location, governance, and query stages, you will end the lab by creating a data product, which package the dataset in a curated way for easy consumption.
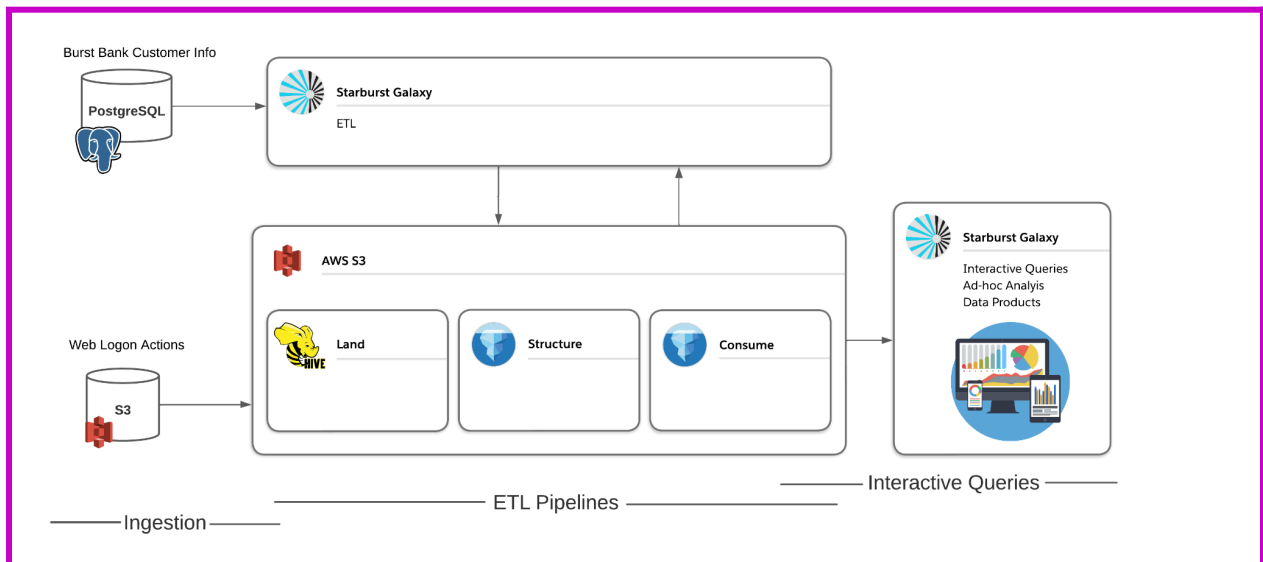
Part 2: Create a Starburst Galaxy account

Now it's time to create a Starburst Galaxy account. If you already have one, you can use that. If not, follow the steps below.

Step 1 - Create a Starburst Galaxy domain

Navigate to the Starburst Galaxy homepage. Enter your First Name, Last Name, and Email address. Click the **Create Account** button.

First Name *:

Last Name *:

Business Email *:

Create Account

By clicking **Create Account**, you agree to Starburst Galaxy's terms of service and privacy policy.

Step 2 - Enter your confirmation code

Go to your email inbox and get the code you were just sent from the subject line of the email.

Confirm your email address

Your confirmation code is below. Enter it in your open browser window and we'll help you get signed in. This code will expire in 24 hours.

129-102

If you didn't request this email, there's nothing to worry about. You can safely ignore it.

Go back to the Starburst Galaxy login, and enter the code. As soon as you type the last digit the verification process will start.

Step 3 - Choose a domain name

Select a meaningful domain name according to the guidelines provided. Click the **Create account** button.

Step 4 - **Create a password**

Enter a memorable and meaningful password. Click the **Create account** button.

Step 5 - **Answer the entrance survey**

Read through the introductory onboarding information. Provide answers accordingly. If you do not know what to answer for a sign-up reason, put **Starburst training or certification**.

Step 6 - Navigate around the UI

Close the in-app product tour as you will go through this throughout the lab.
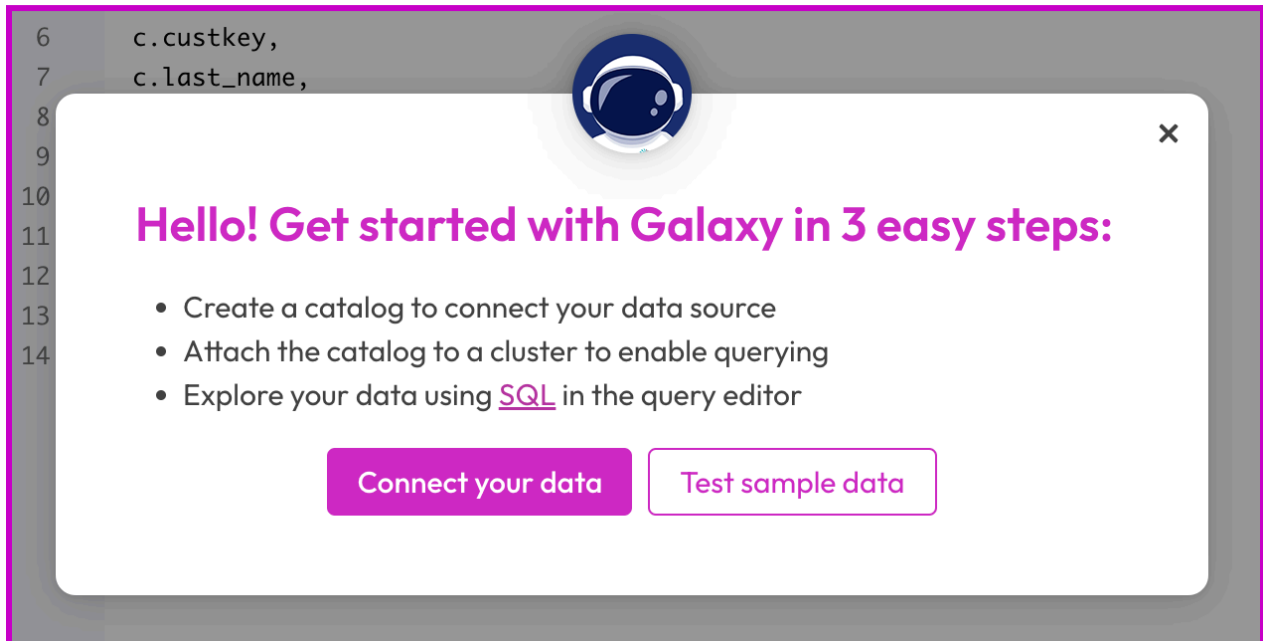
You will land on the query editor - let the lab take it from here.

Part 3: Housekeeping

As part of this workshop, the credentials for Amazon S3 will be available for one week. It is critical to understand that after April 2nd, 2024, **YOU WILL BE UNABLE TO RUN ANY QUERIES AGAINST THE TWO CATALOGS CREATED IN THIS LAB**.

If you want to continue exploring Starburst Galaxy, here are some other free projects and helpful links you can utilize with your Starburst Galaxy account:
- Federate multiple data sources tutorial
- Starburst Academy
  - Starburst Galaxy courses
  - Data foundations courses
  - Learn SQL courses
  - Starburst foundations
- Starburst Galaxy documentation
- Near Real-Time Ingestion tutorial

For questions about the lab material, please reach out to monica.miller@starburstdata.com.

END OF LAB EXERCISE

Lab 2: Create your Amazon S3 catalog

Objective

You're going to begin by setting up an AWS S3 catalog in Starburst Galaxy and connect to the Burst Bank web logon action data.

Step 1 - Sign in and verify your role

Sign in to Starburst Galaxy. Use the account credentials you previously created. In the upper right corner of the screen, confirm that your role is set as `accountadmin`.

Step 2 - Create Amazon S3 catalog

Navigate to the **Catalogs** pane on the left-hand side of Starburst Galaxy. Select the **Create catalog** button to create your first catalog.
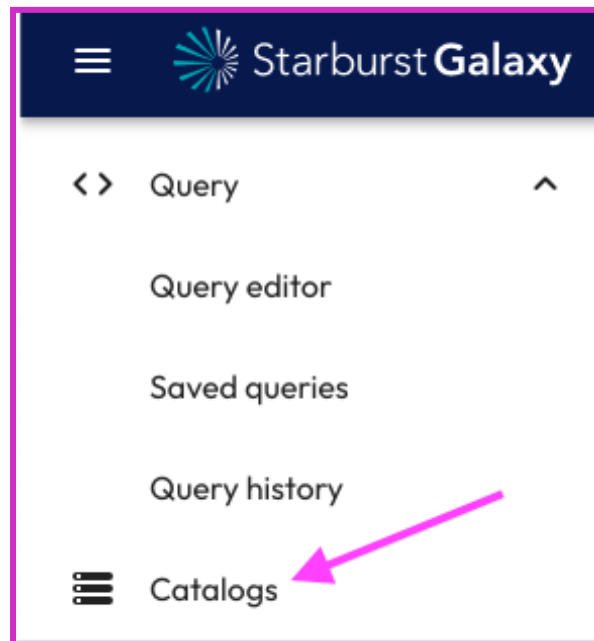
Select **Amazon S3** as your first data source.

Use the information below to configure your catalog. It will query objects in Amazon S3, specifically the NYC rideshare data in your data lake. Provide the necessary credentials to authenticate the connection.

Catalog Name: web_logon_actions
Description: Login action attempts for March 2024
Authentication with: AWS ACCESS KEY (select the radio button AWS access key)
      Access Key: AKIAYUW62MUV33KXHT6V
      Secret Key: VZFckL0tE2vi68JPefQUdjuw+N+9Cmqw5elKpgXB
Metastore type: Starburst Galaxy
Default S3 bucket name: starburst101-handsonlab
Default directory name: <mark>your-name</mark> (ex: monica-miller)
      Allow creating external tables - YES
      Allow writing to external tables - YES
Default table format: Iceberg

Validate the connection by hitting **Test connection.** Your catalog should return the same message indicating that you can now add the catalog.
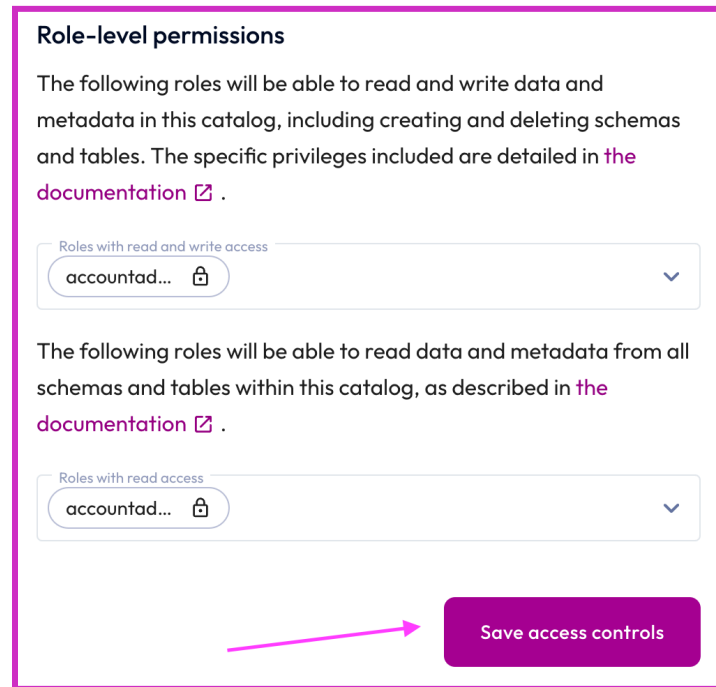
Select **Connect catalog**. This will save the credentials for your Amazon S3 catalog.

Step 3 - Set permissions

Next, accept the default permissions for your catalog by selecting the button **Save access controls**.

Role-level permissions

The following roles will be able to read and write data and metadata in this catalog, including creating and deleting schemas and tables. The specific privileges included are detailed in the documentation ⧉ .

Roles with read and write access
> accountad... 🔒   ⌄

The following roles will be able to read data and metadata from all schemas and tables within this catalog, as described in the documentation ⧉ .
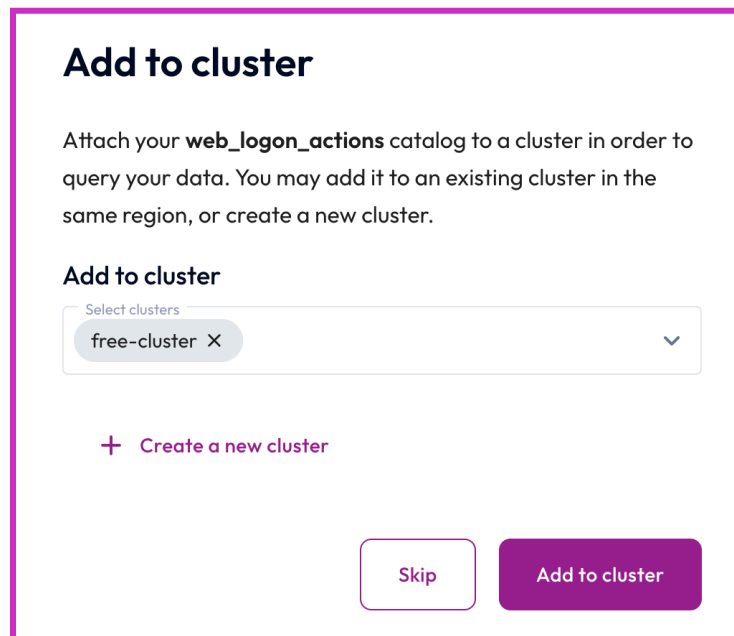
Roles with read access
> accountad... 🔒   ⌄

Save access controls

Step 4 - Select Add to cluster

Finally, you will be asked if you want to add your newly created catalog to a cluster. Select the **free-cluster**. Click **Add to cluster**.

Add to cluster

Attach your **web_logon_actions** catalog to a cluster in order to query your data. You may add it to an existing cluster in the same region, or create a new cluster.

Add to cluster

Select clusters
> free-cluster ✕   ⌄

＋ Create a new cluster

Skip   **Add to cluster**

After your catalog has been added, you will see this pop up. Go ahead and select Run discovery.
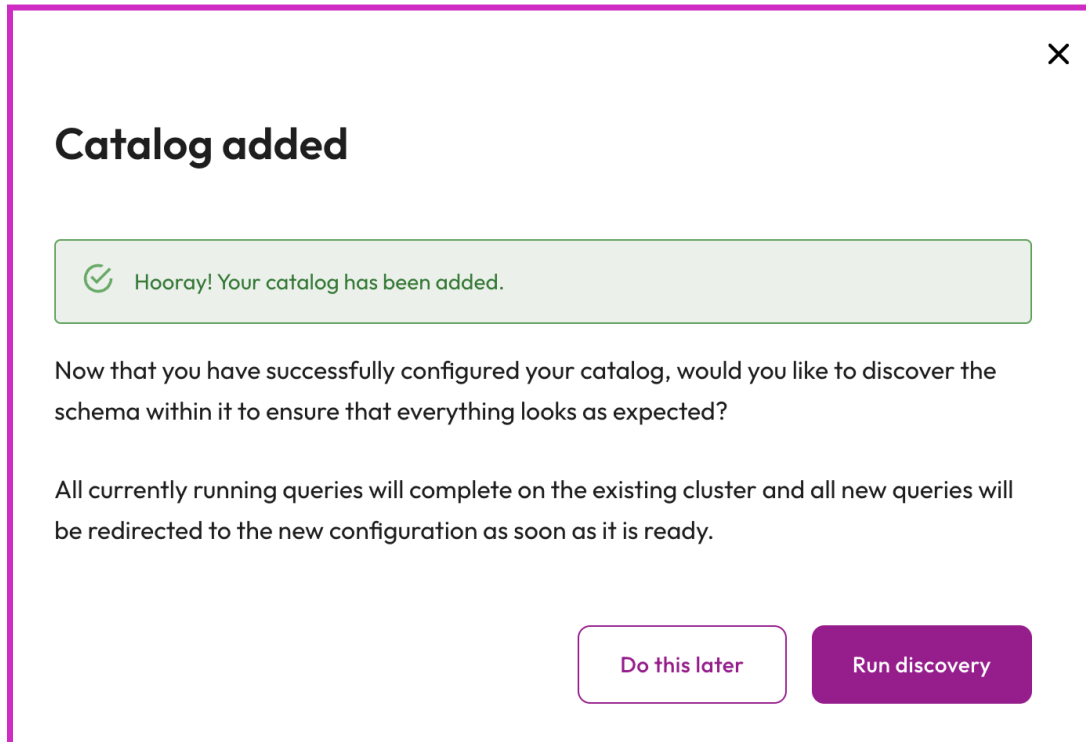
Lab 2: Use Schema Discovery to Ingest your Raw Data

Objective

One of the most difficult parts of data engineering is discovering where your data lives, or convincing someone to let you view the data so you can accurately build the data pipeline requested by another team. Starburst Galaxy utilizes schema discovery so that you can automatically discover and access the data located in your data lake.

In this section, you will use schema discovery S3 object storage location so that you can execute queries within the data lake. Schema discovery will act as the land layer, our raw data layer that is the first level in our data lake reporting structure.

Step 1 - Click Run schema discovery

Within the catalog navigation, hit the **Run schema** discovery button.
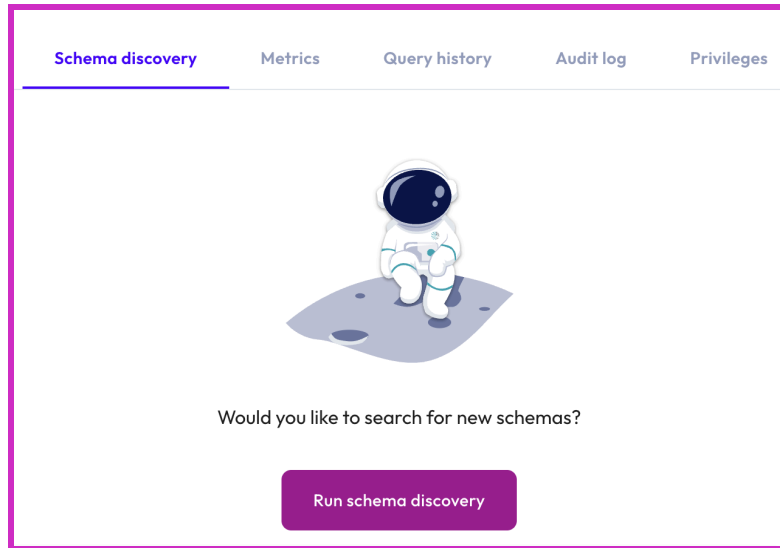
Enter the following information, including the catalog location URI, the default schema name, the type of discovery.

Catalog location URI:
`s3://starburst101-handsonlab-nyc-uber-rides/0324_logon_actions/`
Check the Add location privilege checkbox? `YES`
Default Schema: `discovered_schema`
Type of discovery: `Full discovery`

Hit **Run discovery.** Starburst will start scanning for you. Then, it will return code to create your desired schema and table. You'll return a singular table with 5 partitions (for March 1st - March 5th, 2024) - we are pretending these are the only dates in March thus far. Hit **Create all tables.**

Schema discovery has done the heavy lifting so you don't have to spend time trying to investigate what columns exist, or bother your AWS administrator to give you details about the file.

Click on the Query text to view the full queries. Your first query created the desired schema, and the second query created a Hive table. After you are done, Hit **Close**.

You will see a record of schema discovery in the catalog metadata.

Step 3 - Validate schema discovery

Navigate to the **Query editor** on the left hand side navigation pane.
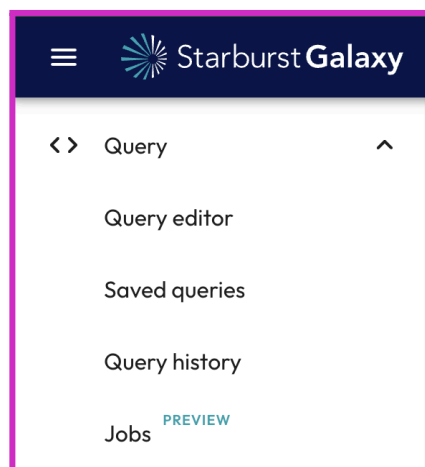
If you already have queries, add a new tab ✚ using the fuschia plus sign. Change the location drop-downs in the top left hand corner to match the cluster and catalog previously created.

Cluster: free-cluster
Catalog: web_logon_actions

Run the following query to validate the table you created using schema discovery.

```
SELECT * FROM "web_logon_actions"."discovered_schema"."0324_logon_actions" LIMIT 10;
```

Run a command to view the CREATE TABLE statement:

```
SHOW CREATE TABLE "web_logon_actions"."discovered_schema"."0324_logon_actions";
```

You should return the same code as run with Schema discovery. Notice that the columns are already utilizing proper data types. Also notice that the table format is HIVE. You will update this as you build your reporting structure in your data lake using Great Lakes connectivity. Recall that the default table format you set for this catalog was Iceberg. However, you are still able to create a table with the Hive table format due to Great Lakes connectivity in Starburst Galaxy. Great Lakes connectivity abstracts the details of using different table formats and file types when using certain write access statements for object storage systems.

END OF LAB EXERCISE

If you are unable to create your schema using schema discovery - refer to appendix section A.

Lab 3: Build within your data lake

Now it's time to use both your data sources and create a reporting structure in S3.

- **Land layer** - This is the raw data you were ingesting that's landing in S3. Thanks to schema discovery, this layer is already created and all you will need to do is validate.
- **Structure layer** - This is the enriched, cleaned, and cleansed data.
- **Consume layer** - This is the data that is ready to be queried and utilized by an end consumer.

Starburst is special because it allows you to build this reporting structure not just with data that already exists in your data lake, but also with data that exists in other data sources in your orbit - like our Postgres Bank data. Because the land layer is already created, we will start on the Structure layer. This is where we can utilize Iceberg.

Part 1: Create your schema

Objective

You're going to begin by creating your schema. Schemas control the structure of the data inside them. You will use this to ensure that the logon data is structured appropriately, plus this is where you will keep this separate since the data in here is using the Iceberg table format.

Step 1 - Select the rideshare cluster

Navigate to the **Query editor**. Add a new tab using the fuschia plus sign ➕. Validate the location drop-downs in the top left hand corner match the cluster and catalog previously created.

Cluster: free-cluster
Catalog: web_logon_actions

Step 2 - Create lab schema

In the Query editor pane, open a new tab and create a new schema with the following command. Highlight and run the command below. Replace <yourname> with your actual name, or another identifier that you prefer.
Note: Only use lowercase characters and numbers; special characters or spaces are not allowed, so remember to replace the < > characters in the example below. You'll use this throughout the lab, so choose something easy to remember and type.

```
CREATE SCHEMA web_logon_actions.<yourname>;
```

Step 3 - Verify schema in catalog

Once this query is **Finished**, in the catalog explorer, expand the `web_logon_actions` catalog to verify your new schema is present and is empty. You should have 3 total schemas listed in your AWS data lake.

Note: If you named your schema as <yourname>, create the schema again with the instructions above.
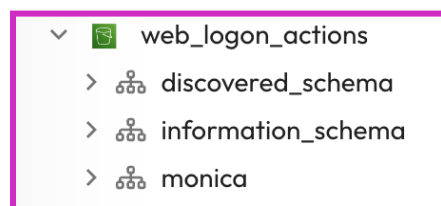
Step 4 - Select schema after creation

In the top left hand corner, select your schema. Validate the location drop-downs in the top left hand corner match the cluster and catalog previously created.

Cluster: free-cluster
Catalog: web_logon_actions
Schema: <<mark>yourname</mark>> (ex: monica_miller)
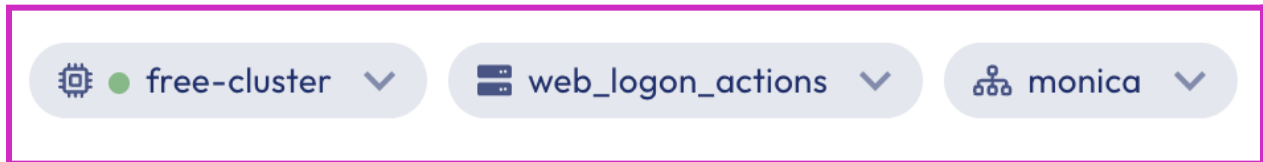
Part 2: Build in the structure layer

Objective

Now it's time to build the structure layer. This is the second of the three data lake reporting structure layers and holds data once it has been transformed.

Step 1 - Create the structure table

The table you just created is not yet fully optimized and transformed. For example, the CSV file format uses the varchar data type as a default for every field, even when it is not accurate. You must build a structure layer, creating a new table and casting the necessary values into more accurate data types with the Parquet file format and Iceberg table format.

Run the following command to create the structure layer table. Make sure you are in the schema you just created.

```sql
CREATE TABLE logon_actions_structure

WITH
 (
   type = 'iceberg',
   format = 'parquet',
   partitioning = ARRAY['login_date']
 ) AS SELECT * FROM
 web_logon_actions.discovered_schema."0324_logon_actions";
```

Let's validate the table creation. Run a basic select statement.

```sql
SELECT * from logon_actions_structure LIMIT 10;
```

You can see the data appears the same, but the makeup of the table is different. Run a show create table statement to view the table.

```
SHOW CREATE TABLE logon_actions_structure;
```

```
CREATE TABLE web_logon_actions.monica.logon_actions_structure (
  custkey integer,
  login_timestamp timestamp(6),
  ip_address varchar,
  device_type varchar,
  login_success boolean,
  login_attempts integer,
  login_date date
)
WITH (
  format = 'PARQUET',
  format_version = 2,
  location = 's3://starburst101-handsonlab/monica-miller/monica/logon_actions_structure-f9eeee068ade4b7696b14f30956a3ec9',
  partitioning = ARRAY['login_date'],
  type = 'ICEBERG'
)
```

Step 2 - Understand where metadata is stored

Iceberg stores table metadata (schema, partitioning details, statistics, versioning information, etc) on the data lake alongside the actual data files. The following diagram presents the richness of this architectural approach.
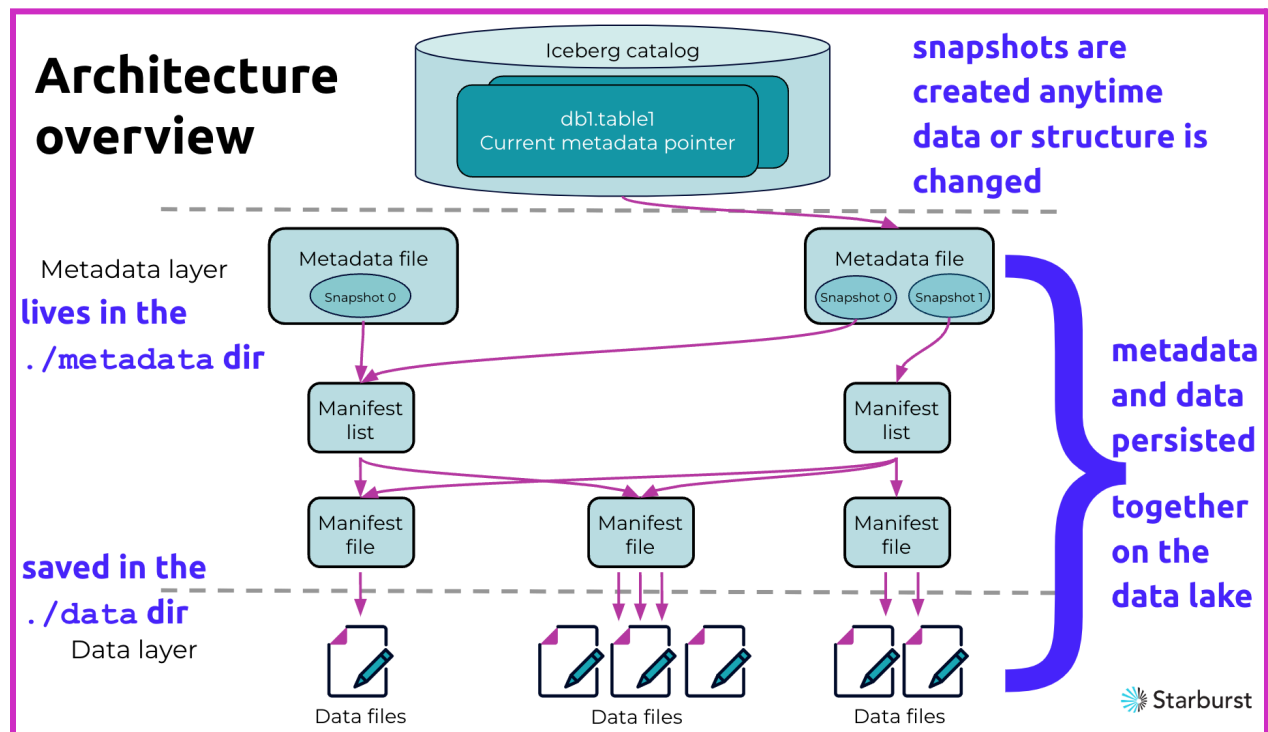
The previous diagram will not be fully described in this exercise, but the following excerpt was taken from the Apache Iceberg Specification and may aid your understanding.

> *This table format tracks individual data files in a table instead of directories. This allows writers to create data files in-place and only adds files to the table in an explicit commit.*
>
> *Table state is maintained in metadata files. All changes to table state create a new metadata file and replace the old metadata with an atomic swap. The table metadata file tracks the table schema, partitioning config, custom properties, and snapshots of the table contents. A snapshot represents the state of a table at some time and is used to access the complete set of data files in the table.*
>
> *Data files in snapshots are tracked by one or more manifest files that contain a row for each data file in the table, the file's partition data, and its metrics. The data in a snapshot is the union of all files in its manifests. Manifest files are reused across snapshots to avoid rewriting metadata that is slow-changing. Manifests can track data files with any subset of a table and are not associated with partitions.*
>
> *The manifests that make up a snapshot are stored in a manifest list file. Each manifest list stores metadata about manifests, including partition stats and data file counts. These stats are used to avoid reading manifests that are not required for an operation.*

The key points to understand are:
1. Data and metadata live together inside the table's data lake table location
    a. The metadata subdirectory contains JSON and AVRO files to represent multiple versions of a data
    b. The data subdirectory contains the actual data files (supports ORC, Parquet, and AVRO) that span all the various versions identified in the metadata
2. Snapshots (i.e. versions) are created anytime the structure or the data changes

Step 3 - Manipulate the table using Iceberg

Now that our table is in Iceberg format, let's explore some of Iceberg's unique capabilities to our table before creating our consume layer.

Verify that partitions in your newly created table. The `$partitions` metadata table is able to help. Run the following command.

```
SELECT * FROM "logon_actions_structure$partitions";
```

You will see the five partitions available.

	partition	record_cou...	file_c...	total_...	data
Finished	{ login_date = 2024-03-02 }	2000	1	29605	{ custkey = { min = 1000001, max = 1000999, null_...
Avg. read speed 6.8 rows/s	{ login_date = 2024-03-01 }	2000	1	29588	{ custkey = { min = 1000003, max = 1001000, null_...
Elapsed time 0.73s	{ login_date = 2024-03-05 }	3000	1	42931	{ custkey = { min = 1000001, max = 1001000, null_...
Rows 5	{ login_date = 2024-03-04 }	3000	1	42894	{ custkey = { min = 1000001, max = 1001000, null_...
Results from cache No	{ login_date = 2024-03-03 }	3000	1	42725	{ custkey = { min = 1000002, max = 1001000, null_...

Verify that a snapshot was created for the table creation. The `$snapshots` table provides a more detailed view.

```
SELECT * FROM "logon_actions_structure$snapshots";
```

	committed_at	snapshot_id	parent_id	operation	manifest_list
Finished	2024-03-26 09:41:21...	8953389557387882...	NULL	append	s3://starburst101-han...
Avg. read speed 2 rows/s					
Elapsed time 0.50s					
Rows 1					
Results from cache No					

As we make alterations to the table, new snapshots will appear. The financial crimes team lets you know they are not concerned about true logon actions that have attempted login twice or less. Let's run a statement to delete the data they don't need.

```
DELETE FROM logon_actions_structure
WHERE login_success = TRUE and login_attempts <= 2;
```

You will delete 4330 rows. Let's rerun the snapshot command to see what happened.

```
SELECT * FROM "logon_actions_structure$snapshots";
```

You will see a new snapshot has been created. While you perform a delete operation, the operation says Overwrite. This is because the [Iceberg default](#) is to overwrite for delete files that match an expression on data rows from the table.

committed_at	snapshot_id	parent_id	operation	manifest_list
2024-03-26 09:41:21...	8953389557387882...	NULL	append	s3://starburst101-han...
2024-03-26 10:00:2...	4740980285783307...	8953389557387882...	overwrite	s3://starburst101-han...

Finished | Avg. read speed 4.6 rows/s | Elapsed time 0.44s | Rows 2 | Results from cache No

Try running the same delete command again and see what happens.

```
DELETE FROM logon_actions_structure
WHERE login_success = TRUE and login_attempts <= 2;
```

You will see 0 rows are returned. This validates the command is idempotent. Run a Select statement to validate the delete worked.

```
SELECT * FROM logon_actions_structure
WHERE login_success = TRUE and login_attempts <= 2;
```

Yay! Delete operation was successful.

Now, let's update the null IP addresses to -1. Run the following command.

```
UPDATE logon_actions_structure
SET ip_address = '-1'
WHERE ip_address is NULL;
```

Check the following information to view if the update was successful.

```
SELECT * FROM logon_actions_structure WHERE ip_address is NULL;
```

Now, view the updated snapshots table. You will see 3 entries.

committed_at	snapshot_id	parent_id	operation	manifest_list
2024-03-26 09:41:21...	8953389557387882...	NULL	append	s3://starburst101-han...
2024-03-26 10:00:2...	4740980285783307...	8953389557387882...	overwrite	s3://starburst101-han...
2024-03-26 10:05:5...	484399817596708654	4740980285783307...	overwrite	s3://starburst101-han...

Finished — Avg. read speed 5.5 rows/s — Elapsed time 0.55s — Rows 3 — Results from cache No

Uh oh! Your data consumers changed their mind. They don't want any of the IP addresses updated. So, what can you do with snapshot files? One useful feature is called "time travel". This allows you to query prior versions of the table via the `snapshot_id`. The second row in the results below relates to the second snapshot. The first snapshot was at the empty table creation. The second was the initial `INSERT` statement.

First, run a select to view the logon actions of the specific customer with the custkey equal to 100643.

```
SELECT * FROM logon_actions_structure
WHERE custkey = 1000643;
```

custkey	login_timestamp	ip_address	device_type	login_success
1000643	2024-03-01 19:24:49....	108.227.79.148	mobile	false
1000643	2024-03-03 15:19:36....	-1	desktop	false
1000643	2024-03-01 20:42:39....	-1	desktop	false
1000643	2024-03-02 10:29:43....	65.36.78.155	desktop	false
1000643	2024-03-03 23:14:09....	254.251.203.31	laptop	false
1000643	2024-03-05 20:35:30....	133.93.255.13	laptop	false

Finished — Avg. read speed 9.2K rows/s — Elapsed time 0.95s — Rows 6 — Results from cache No

Replace 12345678890 in the following query with **_your_** second `snapshot_id` value to see what the table looked like back at that point.

```
SELECT * FROM logon_actions_structure
FOR VERSION AS OF 12345678890
WHERE custkey = 1000643;
```

You can see that the IP Addresses in the previous snapshot are NULL instead of -1.

✓ Finished	Avg. read speed 12.2K rows/s	Elapsed time 0.71s	Rows 6	Results from cache No			
custkey	**login_timestamp**	**ip_address**	**device_type**	**login_success**
1000643	2024-03-05 20:35:30...	133.93.255.13	laptop	false
1000643	2024-03-01 19:24:49....	108.227.79.148	mobile	false
1000643	2024-03-01 20:42:39....	NULL	desktop	false
1000643	2024-03-03 15:19:36....	NULL	desktop	false
1000643	2024-03-03 23:14:09....	254.251.203.31	laptop	false
1000643	2024-03-02 10:29:43....	65.36.78.155	desktop	false

Nevermind, they changed their mind again. Good thing you have the snapshots just in case!

Part 4: Build the consume layer

Objective

Now it's time to construct the last of the three layers of the data lake reporting structure, the consume layer. This will ready the layer for final consumption by data consumers.

Step 1 - Federate your data

Before we jump in and create the final table, run a single select statement which returns data from both your Postgres and S3 sources in one query.

```sql
SELECT
    a.custkey,
    a.login_success,
    a.login_attempts,
    a.login_date,
    b.first_name,
    b.last_name,
    b.fico,
    c.risk_appetite,
    c.customer_segment
FROM logon_actions_structure a
INNER JOIN sample.burstbank.customer b on cast(a.custkey as varchar)
= b.custkey
INNER JOIN sample.burstbank.customer_profile c ON b.custkey =
c.custkey;
```

Step 2 - Create the consume table

Run the following query to create the consume table, This constructs a new table from two separate data sources–Amazon S3 and Postgres.

```
CREATE TABLE logon_actions_consume AS
SELECT
    a.custkey,
    a.login_success,
    a.login_attempts,
    a.login_date,
    b.fico,
    b.first_name,
    b.last_name,
    c.risk_appetite,
    c.customer_segment
FROM logon_actions_structure a
INNER JOIN sample.burstbank.customer b on cast(a.custkey as varchar)
= b.custkey
INNER JOIN sample.burstbank.customer_profile c on b.custkey =
c.custkey
WHERE a.custkey in (select custkey from logon_actions_structure group
by custkey having count(custkey) > 5);
```

The results should show 8114 rows. Run a `SELECT` statement to validate that the table's values are similar to the image below.

```
SELECT * from logon_actions_consume LIMIT 10;
```

custkey	logi...	login...	login_date	fico	first_name	last_name	risk_appetite	customer_segment
1000002	false	3	2024-03-02	570	Ryan	Moon	wild_west	gold
1000002	false	2	2024-03-05	570	Ryan	Moon	wild_west	gold
1000002	false	3	2024-03-03	570	Ryan	Moon	wild_west	gold
1000002	false	2	2024-03-03	570	Ryan	Moon	wild_west	gold
1000002	false	3	2024-03-05	570	Ryan	Moon	wild_west	gold

The final table has all the information for the financial crimes analysts to investigate suspicious activity. Next, create a data product for them to begin.
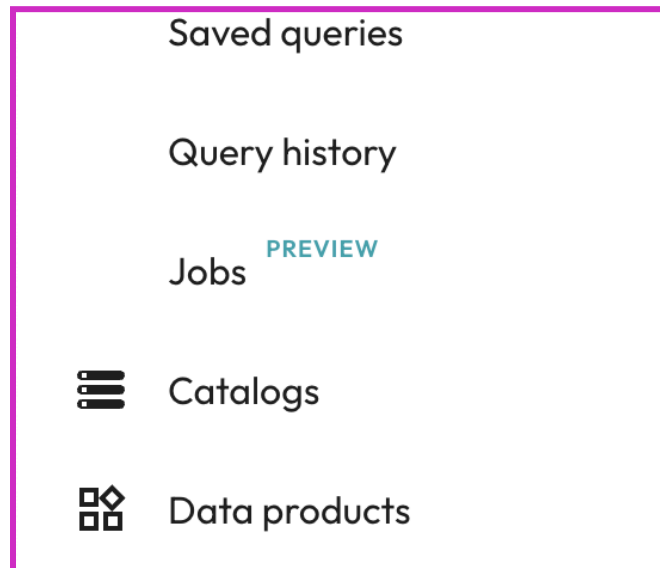
Lab 4: Create data products

Objective

Now it's time to create a data product using your datasets. Data products curate data in a way that makes it more accessible and useful, and can be shared across teams.

Step 1 - Navigate to your web_logon_actions catalog

Go to the catalogs tab on your left hand pane.

Click on the name of the web_logon_actions catalog.

Click within your schema that you have created (ex: monica).

Step 2 - Enter additional information

Make sure you are in your schema as mentioned in the previous step. Before creating a data product, enter some information into your catalog.

Select **Show details** on the right hand side.

Edit the following information:

Description: Suspicious fraudulent activity for the month of March 2024.
Links: Frauds and scams
      Link URL: https://www.consumerfinance.gov/consumer-tools/fraud/
Contacts: yourname

DESCRIPTION		TAGS	LINKS	CONTACTS	OWNER
Suspicious fraudulent activity for the month of Ma...		🏷 0	🔗 1	👤 1	🪪 accountadmin

DESCRIPTION	Suspicious fraudulent activity for the month of March 2024.	✏
TAGS	No tags added yet.	✏
LINKS	🔗 Frauds and scams	✏
CONTACTS	👤 monica.miller@starburstdata.com	✏
OWNER	🪪 accountadmin	✏

Step 3 - Promote your data product

Select **Promote to data product**.

monica_miller — Promote to data product ⋮

Show details

DESCRIPTION	TAGS	LINKS	CONTACTS	OWNER
This is the da...	🏷 0	🔗 1	👤 1	🪪 accountadmin

Import all the information you've already added to the schema. Select **Import**.

Add a descriptive name like **FIN-CRIMES-03-24**. The summary has already been populated based on the information you added to the schema.

Add the following description: This is the data that will help Burst Bank stomp out the new fraudulent trend in web logon transactions!

Select the **free-cluster** cluster as the default cluster. Your contacts and supporting information have been automatically populated.

Select **Promote to data product**.
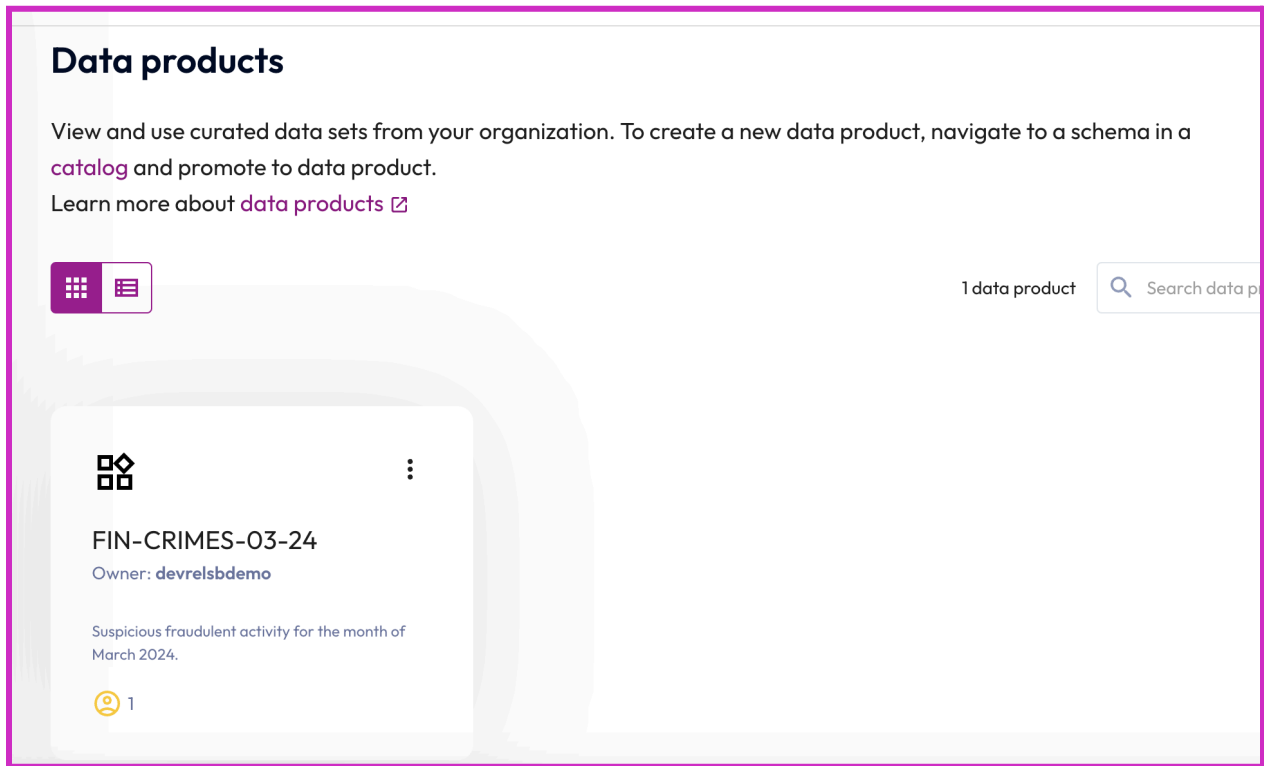
Congratulations! You have created and promoted your first data product. Navigate to the Data products tab to view your work.

Appendix: Section A

If you are unable to create your land layer with schema discovery, utilize the code below. This will create the `discovered_schema` and the `year_month` table manually.

Step 1 - Select the rideshare cluster

Navigate to the **Query editor**. Add a new tab using the fuschia plus sign **+**. Validate the location drop-downs in the top left hand corner match the cluster and catalog previously created.

Cluster: nyc-ride-info
Catalog: nyc_uber_rides

Step 2 - Create lab schema
In the Query editor pane, open a new tab and create a new schema with the following command.

```
CREATE SCHEMA web_logon_actions.discovered_schema;
```

Step 3 - Select schema after creation
In the top left hand corner, select your schema. Validate the location drop-downs in the top left hand corner match the cluster and catalog previously created.

Cluster: free-cluster
Catalog: web_logon_actions
Schema: `discovered_schema`

Step 2 - Create the external table

Run the following command to create an external table from the logon data stored in the data lake. This table is partitioned by day.

```
CREATE TABLE web_logon_actions.discovered_schema."0324_logon_actions" (
  custkey integer,
  login_timestamp timestamp(3),
  ip_address varchar,
  device_type varchar,
  login_success boolean,
  login_attempts integer,
  login_date date
)
WITH (
  external_location = 's3://starburst101-handsonlab-nyc-uber-rides/0324_logon_actions/',
  format = 'JSON',
  partitioned_by = ARRAY['login_date'],
  type = 'HIVE'
);
```

Recall that the default table format you set for this catalog was Iceberg. However, you are still able to create a table with the Hive table format due to [Great Lakes connectivity](#) in Starburst Galaxy. Great Lakes connectivity abstracts the details of using different table formats and file types when using certain write access statements for object storage systems.

Since you are using Starburst Galaxy's metastore, run the following command to sync the metadata.

```
call system.sync_partition_metadata('discovered_schema',
'web_logon_actions', 'ADD');
```