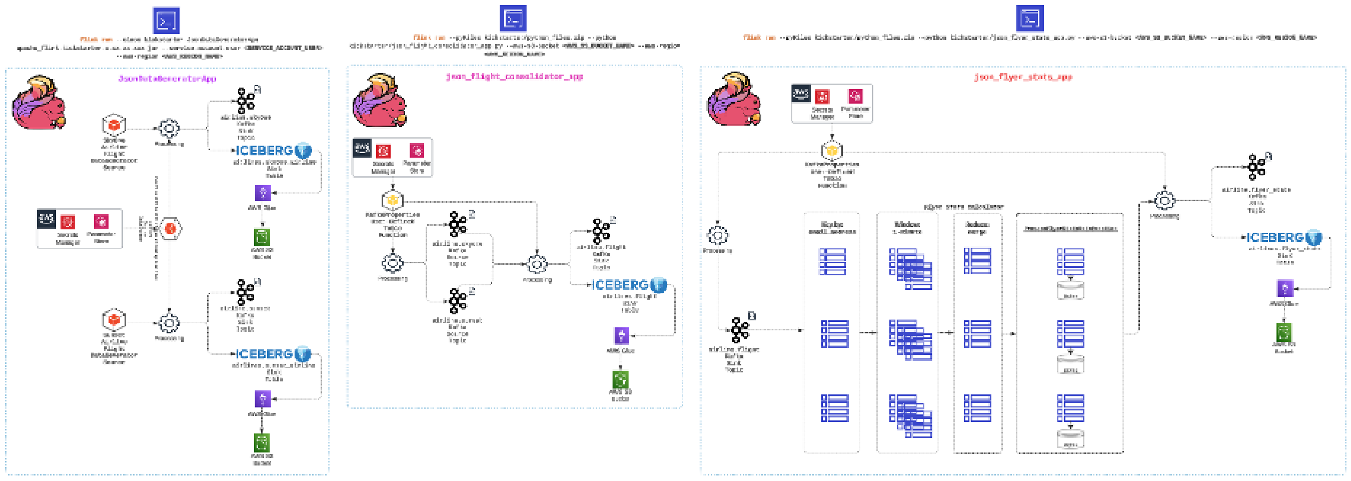
Apache Flink Kickstarter

This project showcases the exceptional capabilities of Apache Flink, known for its high performance, scalability, and advanced stream processing features, which is a core component of the signalRoom technology stack. Staying true to our mission of empowering developers by sharing practical knowledge, we demonstrate the use of Flink through real-world examples based on the blog series "Building Apache Flink Applications in Java."



Flink App	Description
`Data Generator App`	This Flink App generates realistic flight data for the fictional airlines Sunset Air and Sky One Airlines, seamlessly integrating modern data streaming technologies. Flight events are published to dedicated Kafka topics (`airline.sunset` and `airline.skyone`), enabling real-time processing and analysis. Simultaneously, the synthetic data flows into the `apache_kickstarter.airlines.skyone_airline` and `apache_kickstarter.airlines.sunset_airline` Apache Iceberg Tables, respectively, stored in an AWS S3 bucket, providing a robust, scalable foundation for historical data analytics. The entire solution is implemented in Java, overcoming limitations in **PyFlink**, which currently lacks a Python-based Apache Flink data generator source. This design ensures a powerful, flexible data pipeline, ideal for both real-time and batch use cases.
`Flight Consolidator App`	Imports flight data from `airline.sunset` and `airline.skyone` Kafka topics and standardizes them into a unified `airline.flight` Kafka topic, and the `apache_kickstarter.airlines.flight` Apache Iceberg Table. Implemented in both Java and Python.
`Flyer Stats App`	Processes data from the `airline.flight` Kafka topic to aggregate flyer statistics into the `airline.flyer_stats` Kafka topic, and the `apache_kickstarter.airlines.flyer_stats` Apache Iceberg Table. Implemented in both Java and Python.

Created by Wade Waldron, Staff Software Practice Lead at Confluent Inc., these applications are designed to handle enterprise-scale use cases to demonstrate Flink's potential for real-world applications. This will start with securely fetching Kafka Cluster and Schema Registry Cluster API keys via ***AWS Secrets Manager***, and retrieving essential Kafka client configuration properties through the ***AWS Systems Manager Parameter***.

Beyond simply transforming and enriching data, we stream it into Kafka topics and ***Apache Iceberg Tables***, designed for high-performance, large-scale analytics with support for ACID transactions, offering highly scalable, real-time data processing and durable storage.

Our journey with Flink doesn’t end with Java; Python-based Flink applications also shine in this project, leveraging Flink SQL, Table API, and DataFrame API, each offering unique advantages: Flink SQL for declarative queries, Table API for a mix of SQL-like syntax and programmatic control, and DataFrame API for efficient data manipulation similar to pandas or R. This provides a seamless, flexible approach to solving complex streaming challenges, ultimately unlocking an efficient and scalable architecture for modern data needs.

Explore the repo, run the Docker containers (compatible with both Mac M chips and non-Mac M chips), and see how Apache Flink can be utilized to build robust, enterprise-grade streaming applications.

Table of Contents

1.0 Are these examples better described as Flink Jobs or Flink Applications-—and why does the distinction matter?

> *"What's in a name? That which we call a rose by any other name would smell just as sweet."*
>
> *-- William Shakespeare*

Did you know Flink jobs are often called Flink applications? It’s not just a fancy name upgrade—there’s a good reason behind it! Unlike a single task or computation, a Flink job is like a mini software ecosystem. Calling it an 'application' captures its true essence: a complex, integrated workflow that solves big data challenges, just like any full-fledged software app.

By using 'Flink applications,' we're emphasizing their depth and sophistication. It’s not just about crunching numbers; it’s about building something comprehensive, dynamic, and powerful—just like any great app you use every day. (Curious for more details? Check out the rationale here!)

2.0 Let's get started!

Ready to supercharge your data streaming skills? As of October 4, 2024, Apache Flink on Confluent Cloud brings incredible possibilities and a few challenges! Currently, it doesn't support the DataStream API, and its Table API is limited (click here for a list of what is supported) to creating Java or Python-based Flink applications. Don't worry, though—we've got you covered!

In this repo, you'll find an example code where we use the Java DataStream API to create a ***custom source*** beyond Flink's typical capabilities. Imagine making external calls to AWS Secrets Manager to securely retrieve Kafka Cluster API Key secrets or pulling Kafka consumer and producer client configuration properties from the AWS Systems Manager Parameter Store—all directly integrated into Flink!

Prefer Python? We've got you there too. With the Table API, you can use a ***User-Defined Table Function*** to do the same. Plus, we don't just sink data into Kafka topics; this repo showcases how to store it in ***Apache Iceberg tables***, unlocking even more power for your data.

To help you start quickly, the repo comes with ***Docker containers*** for Mac M chip and non-Mac M chip machines, letting you run an Apache Flink 1.20.0 cluster locally. For the Kafka and Schema Registry resources, we're fully leveraging Confluent Cloud. Dive in and explore—the future of data streaming is here!

These are the steps

1. Take care of the cloud and local environment prequisities listed below:

    > You need to have the following cloud accounts:
    >
    > - AWS Account *with SSO configured*
    > - Confluent Cloud Account
    > - Docker Account
    > - GitHub Account *with OIDC configured for AWS*
    > - Snowflake Account
    > - Terraform Cloud Account

    > You need to have the following installed on your local machine:
    >
    > - AWS CLI version 2
    > - Confluent CLI version 3 or higher
    > - Docker Desktop
    > - Java JDK (Java Development Kit) 17
    > - Python 3.11
    > - Terraform CLI version 1.9.3 or higher

2. Clone the repo:

    ```
    git clone https://github.com/j3-signalroom/apache_flink-
    kickstarter.git
    ```

3. Set up your Terraform Cloud environment locally or leverage GitHub Actions to create the complete setup effortlessly. Here's what you can expect:
    - A Confluent Cloud environment featuring a Kafka Cluster, fully equipped with pre-configured example Kafka topics—ready to power your data streaming needs.
    - AWS Secrets Manager securely storing API Key Secrets for the Kafka Cluster, along with AWS Systems Manager Parameter Store containing Kafka Consumer and Producer properties for easy integration.
    - An AWS S3 bucket with a dedicated `warehouse` folder, serving as the landing zone for Apache Iceberg Tables populated by two Python-based Flink apps, bringing your data streaming architecture to life.
4. Run Apache Flink locally on your Mac, or use the provided Docker containers from the project to launch Apache Flink and Apache Iceberg seamlessly on your machine.

With these steps, you'll have everything set up to run enterprise-grade data streaming applications in no time!

2.1 DevOps in Action: Running Terraform Locally

Install the Terraform CLI on your local machine, and make sure you have an HCP Terraform account to run the Terraform configuration. Learn how to set up Terraform Cloud for local use by clicking here.

2.1.1 Run locally

```
scripts/deploy-terraform.sh <create | delete> --profile=<SSO_PROFILE_NAME>
                                                   --confluent-api-key=
<CONFLUENT_API_KEY>
                                                   --confluent-api-secret=
<CONFLUENT_API_SECRET>
                                                   --snowflake-warehouse=
<SNOWFLAKE_WAREHOUSE>
                                                   --service-account-user=
<SERVICE_ACCOUNT_USER>
                                                   --day-count=<DAY_COUNT>
                                                   --auto-offset-reset=
<earliest | latest>
                                                   --number-of-api-keys-
to-retain=<NUMBER_OF_API_KEYS_TO_RETAIN>
                                                   --admin-service-user-
secrets-root-path=<ADMIN_SERVICE_USER_SECRETS_ROOT_PATH>
```

Argument placeholder	Replace with

Argument placeholder	Replace with
`<SSO_PROFILE_NAME>`	your AWS SSO profile name for your AWS infrastructue that host your AWS Secrets Manager.
`<CONFLUENT_API_KEY>`	your organization's Confluent Cloud API Key (also referred as Cloud API ID).
`<CONFLUENT_API_SECRET>`	your organization's Confluent Cloud API Secret.
`<SNOWFLAKE_WAREHOUSE>`	the Snowflake warehouse (or "virtual warehouse") you choose to run the resources in Snowflake.
`<SERVICE_ACCOUNT_USER>`	the Snowflake service account user who is to be assigned the RSA key pairs for its authentication.
`<DAY_COUNT>`	how many day(s) should the API Key be rotated for.
`<AUTO_OFFSET_RESET>`	Use `earliest`, when you want to read the first event in a Kafka topic. Otherwise, specify `latest`.
`<NUMBER_OF_API_KEYS_TO_RETAIN>`	Specifies the number of API keys to create and retain.
`<ADMIN_SERVICE_USER_SECRETS_ROOT_PATH>`	the root path in AWS Secrets Manager for admin service user secrets.

To learn more about this script, click here.

2.2 DevOps in Action: Running Terraform in the cloud

In order to run the Terraform configuration from GitHub, the Terraform Cloud API token and Confluent Cloud API Key are required as GitHub Secret variables. Learn how to do to get the Terraform Cloud API token and Confluent Cloud API key here.

2.2.1 Run from the cloud

Follow these steps:

a. **Deploy the Repository**: Ensure that you have cloned or forked the repository to your GitHub account.

b. **Set Required Secrets and Variables**: Before running any of the GitHub workflows provided in the repository, you must define at least the `AWS_DEV_ACCOUNT_ID` variable (which should contain your AWS Account ID for your development environment). To do this:

- Go to the **Settings** of your cloned or forked repository in GitHub.

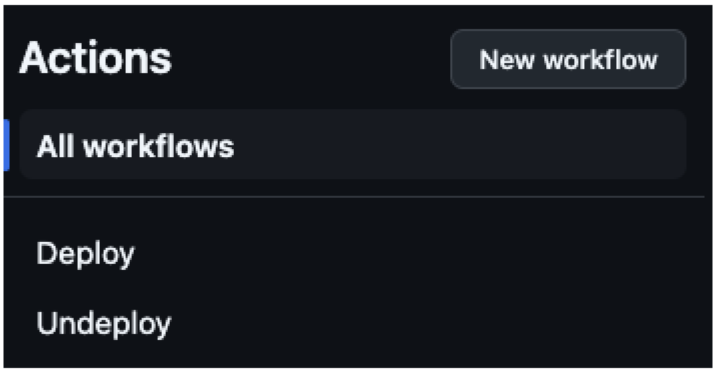
- Navigate to **Secrets and Variables** > **Actions**.
- Add the `AWS_DEV_ACCOUNT_ID` and any other required variables or secrets.

c. **Navigate to the Actions Page**:

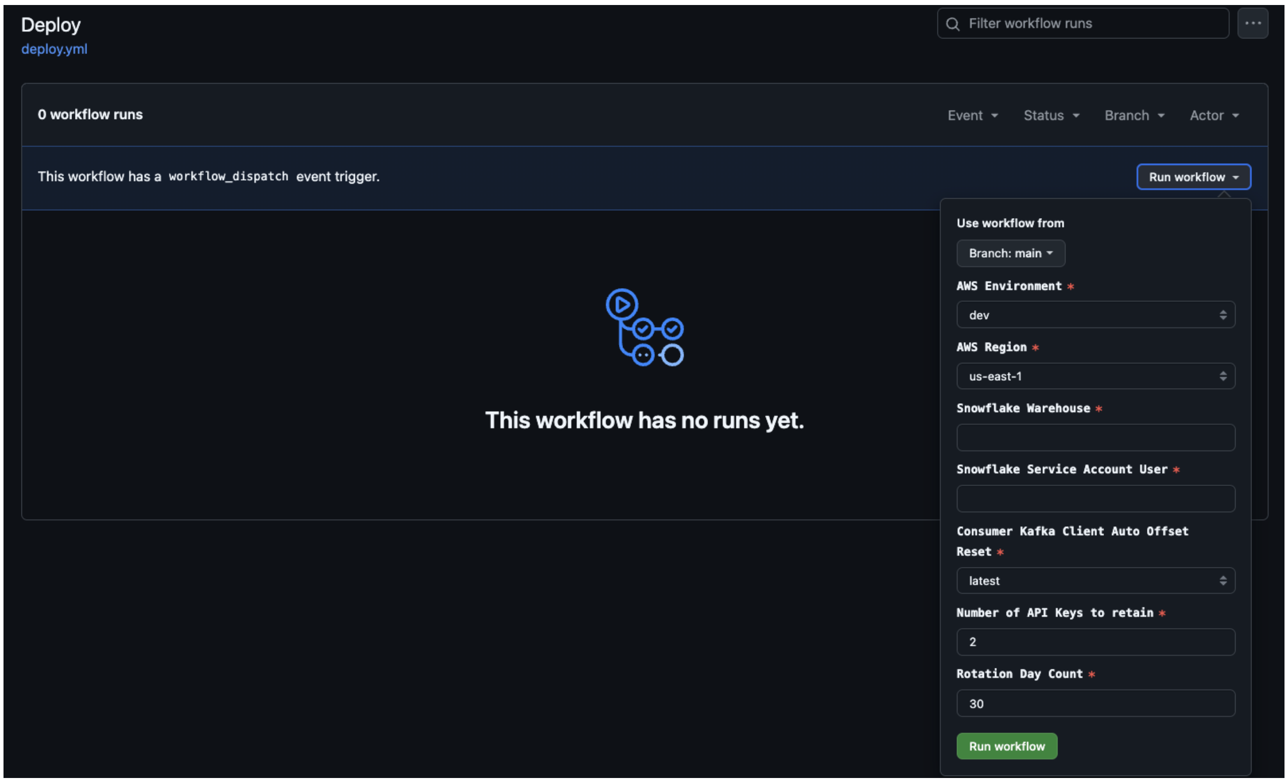
- From the cloned or forked repository on GitHub, click on the **Actions** tab.

d. **Select and Run the Deploy Workflow**:

- Find the **Deploy workflow** link on the left side of the Actions page and click on it.



- On the **Deploy workflow** page, click the **Run workflow** button.
- A workflow dialog box will appear. Fill in the necessary details and click **Run workflow** to initiate the Terraform deployment process.



This workflow has a `workflow_dispatch` event trigger.

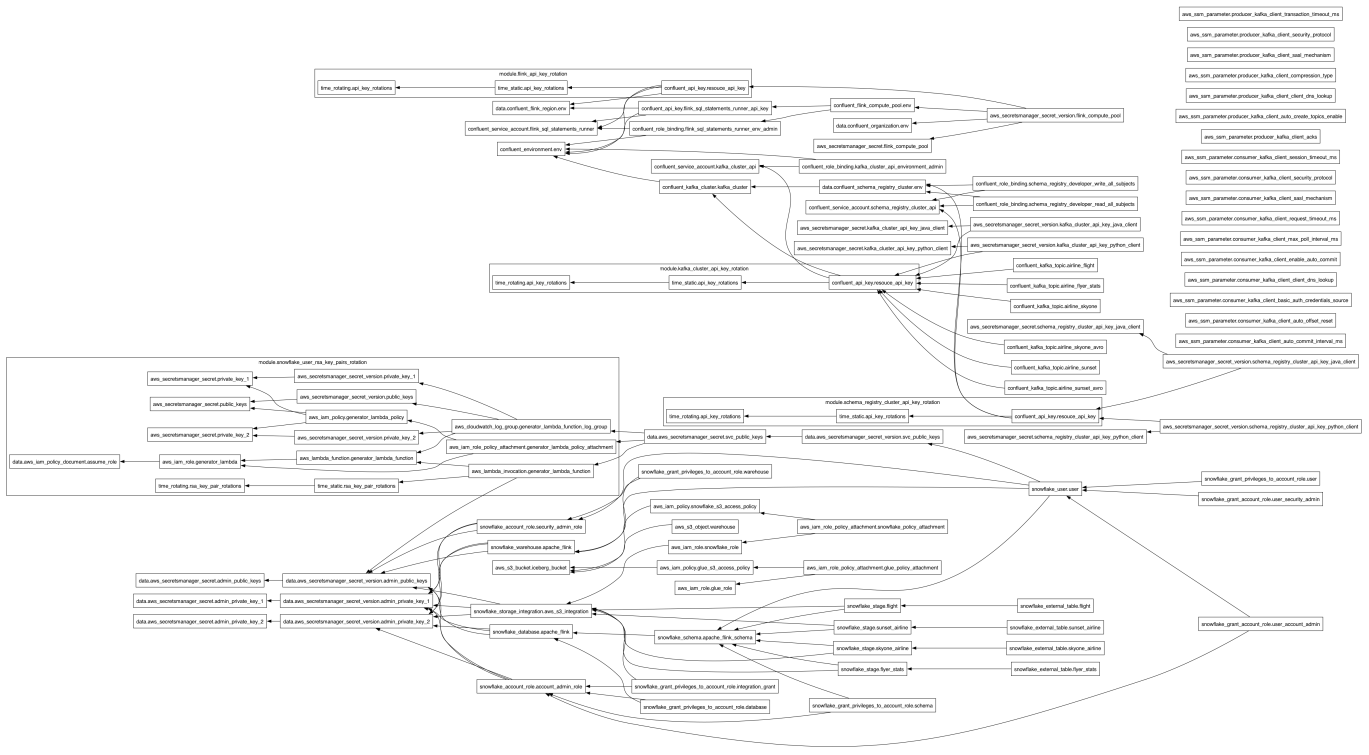
This workflow has no runs yet.

Use workflow from: Branch: main
AWS Environment *: dev
AWS Region *: us-east-1
Snowflake Warehouse *:
Snowflake Service Account User *:
Consumer Kafka Client Auto Offset Reset *: latest
Number of API Keys to retain *: 2
Rotation Day Count *: 30

By following these steps, you will run the Terraform configuration directly from GitHub, leveraging GitHub Actions for automation and deployment.

2.3 Visualizing the Terraform Configuration

Below is the Terraform visualization of the Terraform configuration. It shows the resources and their dependencies, making the infrastructure setup easier to understand.


module.flink_api_key_rotation
time_rotating.api_key_rotations
time_static.api_key_rotations
confluent_api_key.resouce_api_key
data.confluent_flink_region.env
confluent_api_key.flink_sql_statements_runner_api_key
confluent_service_account.flink_sql_statements_runner
confluent_role_binding.flink_sql_statements_runner_env_admin
confluent_environment.env
confluent_flink_compute_pool.env
data.confluent_organization.env
aws_secretsmanager_secret_version.flink_compute_pool
aws_secretsmanager_secret.flink_compute_pool
confluent_service_account.kafka_cluster_api
confluent_role_binding.kafka_cluster_api_environment_admin
confluent_kafka_cluster.kafka_cluster
data.confluent_schema_registry_cluster.env
confluent_service_account.schema_registry_cluster_api
confluent_role_binding.schema_registry_developer_write_all_subjects
confluent_role_binding.schema_registry_developer_read_all_subjects
aws_secretsmanager_secret.kafka_cluster_api_key_java_client
aws_secretsmanager_secret.kafka_cluster_api_key_python_client
aws_secretsmanager_secret_version.kafka_cluster_api_key_java_client
aws_secretsmanager_secret_version.kafka_cluster_api_key_python_client
module.kafka_cluster_api_key_rotation
confluent_kafka_topic.airline_flight
confluent_kafka_topic.airline_flyer_stats
confluent_kafka_topic.airline_skyone
aws_secretsmanager_secret.schema_registry_cluster_api_key_java_client
confluent_kafka_topic.airline_skyone_avro
confluent_kafka_topic.airline_sunset
confluent_kafka_topic.airline_sunset_avro
aws_secretsmanager_secret_version.schema_registry_cluster_api_key_java_client
module.schema_registry_cluster_api_key_rotation
aws_secretsmanager_secret_version.schema_registry_cluster_api_key_python_client
aws_secretsmanager_secret.schema_registry_cluster_api_key_python_client
aws_ssm_parameter.producer_kafka_client_transaction_timeout_ms
aws_ssm_parameter.producer_kafka_client_security_protocol
aws_ssm_parameter.producer_kafka_client_sasl_mechanism
aws_ssm_parameter.producer_kafka_client_compression_type
aws_ssm_parameter.producer_kafka_client_client_dns_lookup
aws_ssm_parameter.producer_kafka_client_auto_create_topics_enable
aws_ssm_parameter.producer_kafka_client_acks
aws_ssm_parameter.consumer_kafka_client_session_timeout_ms
aws_ssm_parameter.consumer_kafka_client_security_protocol
aws_ssm_parameter.consumer_kafka_client_sasl_mechanism
aws_ssm_parameter.consumer_kafka_client_request_timeout_ms
aws_ssm_parameter.consumer_kafka_client_max_poll_interval_ms
aws_ssm_parameter.consumer_kafka_client_enable_auto_commit
aws_ssm_parameter.consumer_kafka_client_client_dns_lookup
aws_ssm_parameter.consumer_kafka_client_basic_auth_credentials_source
aws_ssm_parameter.consumer_kafka_client_auto_offset_reset
aws_ssm_parameter.consumer_kafka_client_auto_commit_interval_ms
module.snowflake_user_rsa_key_pairs_rotation
aws_secretsmanager_secret.private_key_1
aws_secretsmanager_secret_version.private_key_1
aws_secretsmanager_secret.public_keys
aws_secretsmanager_secret_version.public_keys
aws_iam_policy.generator_lambda_policy
aws_cloudwatch_log_group.generator_lambda_function_log_group
aws_secretsmanager_secret.private_key_2
aws_secretsmanager_secret_version.private_key_2
aws_iam_role_policy_attachment.generator_lambda_policy_attachment
data.aws_iam_policy_document.assume_role
aws_iam_role.generator_lambda
aws_lambda_function.generator_lambda_function
aws_lambda_invocation.generator_lambda_function
time_rotating.rsa_key_pair_rotations
time_static.rsa_key_pair_rotations
data.aws_secretsmanager_secret.svc_public_keys
data.aws_secretsmanager_secret_version.svc_public_keys
aws_secretsmanager_secret.schema_registry_cluster_api_key_python_client
snowflake_grant_privileges_to_account_role.warehouse
snowflake_user.user
snowflake_grant_privileges_to_account_role.user
snowflake_grant_account_role.user_security_admin
aws_iam_policy.snowflake_s3_access_policy
snowflake_account_role.security_admin_role
aws_s3_object.warehouse
aws_iam_role_policy_attachment.snowflake_policy_attachment
snowflake_warehouse.apache_flink
aws_iam_role.snowflake_role
aws_s3_bucket.iceberg_bucket
aws_iam_policy.glue_s3_access_policy
aws_iam_role_policy_attachment.glue_policy_attachment
aws_iam_role.glue_role
data.aws_secretsmanager_secret.admin_public_keys
data.aws_secretsmanager_secret_version.admin_public_keys
data.aws_secretsmanager_secret.admin_private_key_1
data.aws_secretsmanager_secret_version.admin_private_key_1
data.aws_secretsmanager_secret.admin_private_key_2
data.aws_secretsmanager_secret_version.admin_private_key_2
snowflake_storage_integration.aws_s3_integration
snowflake_stage.flight
snowflake_external_table.flight
snowflake_database.apache_flink
snowflake_stage.sunset_airline
snowflake_external_table.sunset_airline
snowflake_schema.apache_flink_schema
snowflake_stage.skyone_airline
snowflake_external_table.skyone_airline
snowflake_stage.flyer_stats
snowflake_external_table.flyer_stats
snowflake_grant_account_role.user_account_admin
snowflake_account_role.account_admin_role
snowflake_grant_privileges_to_account_role.integration_grant
snowflake_grant_privileges_to_account_role.database
snowflake_grant_privileges_to_account_role.schema

> **To fully view the image, open it in another tab on your browser to zoom in.**

When you update the Terraform Configuration, to update the Terraform visualization, use the `terraform graph` command with Graphviz to generate a visual representation of the resources and their dependencies. To do this, run the following command:

```
terraform graph | dot -Tpng > .blog/images/terraform-visualization.png
```

3.0 Hands-On Kickoff: Practical Examples for Rapid Learning

3.1 Flink Applications Powered by Java on a locally running Apache Flink Cluster in Docker

Let's go!

3.2 Flink Applications Powered by Python on a locally running Apache Flink Cluster in Docker

Let's go!

3.3 Flink Applications Powered by Python on Confluent Cloud for Apache Flink

Let's go!

4.0 Powering Analytics with Apache Iceberg tables in Snowflake

After running the Flink applications, you can now leverage the data stored in the Apache Iceberg tables in Snowflake for analytics. Click here to learn more about using **Non-Managed Apache Iceberg tables** in

Snowflake. This blog post will help you understand the trade-offs, operational characteristics, and integration patterns of using Apache Iceberg tables in Snowflake. These terraform configurations:

- aws-iam-snowflake-setup - creates the necessary Snowflake IAM roles and policies in AWS to get started.
- snowflake-non-managed-apache-iceberg-table-privileges-setup - grants the necessary privileges to the Snowflake resources to get started.
- snowflake-non-managed-apache-iceberg-table-setup - creates the necessary resources in Snowflake to get started.

5.0 Resources

Apache Flink's Core is Dataflow Programming

What is Apache Flink? — Architecture

Apache Flink Use Cases

Building Apache Flink Applications in Java

J3's techStack Lexicon

Unlocking Schema Registry Access: Granting Confluent Service Account Permissions with Terraform

6.0 Important Note(s)

Known Issue(s)