Reference Architecture: Confluent and Snowflake

Snowflake and Confluent partner to provide their customers a better overall experience combining their capabilities. Confluent provides distributed, scalable, and secure event delivery that can scale to handle trillions of events a day with Confluent Platform and Confluent Cloud. Billed as the "first analytic database built for the cloud", Snowflake offers a popular data warehouse as a service that runs on the most popular cloud providers, emphasizing speed, ease of use and flexibility over traditional data warehouse solutions. The Snowflake Kafka Connector bridges the two, providing a well supported path for data in Apache Kafka to be available in Snowflake. This document provides an overview of Confluent and Snowflake's combined offering, a detailed tutorial for getting started with the integration, and unique considerations to keep in mind when working with these two technologies.

Use Cases

The Snowflake Kafka Connector can support any use case supported by Snowflake, including but not limited to:

- Data warehouse modernization
- Ad hoc, accelerated data analytics
- Data lake insights
- Data exchange
- Developing embedded applications
- Data science and engineering

In each of these use cases, Snowflake needs access to data ingested from various kinds of data sources, including both real-time and on-prem. No ingestion framework to Snowflake is complete without Apache Kafka support.

Technical Brief

The recommended Snowflake integration with Confluent is via the Snowflake Kafka Connector supported by Snowflake. It's a sink connector only. Additionally, Kafka and Snowflake are seen integrated via indirect integrations with Snowpipe, using shared intermediate storage such as S3. The most commonly deployed integration is to dump the data from Kafka into a cloud store using the S3 connector, and then to use Snowpipe to ingest the batched files into Snowflake. Both the connector and the most commonly deployed integration route the data through the Snowpipe service, which is a thin wrapper to S3.

Partner Product Overview

Snowflake is the "first analytic database built for the cloud". Snowflake offers a data warehouse as a service that runs on the most popular cloud providers. Their benefits include speed, ease of use and flexibility over traditional data warehouse solutions. Snowflake users only need to create databases, tables, and virtual warehouses, load data, and execute queries.

Snowflake Concepts

Snowflake's data warehouse is a true SaaS offering. More specifically:

- There is no hardware (virtual or physical) for users to select, install, configure, or manage.
- There is no software for users to install, configure, or manage.
- Ongoing maintenance, management, and tuning is handled by Snowflake.

Snowflake runs completely on cloud infrastructure. All components of Snowflake's service (other than an optional command line client), run in a public cloud infrastructure. Snowflake uses virtual compute instances for its compute needs and a storage service for persistent storage of data. Snowflake cannot be run on private cloud infrastructures (on-premises or hosted). Snowflake is not a packaged software offering that can be installed by a user. Snowflake manages all aspects of software installation and updates.

Features provided by Snowflake include:

- Zero copy cloning: Giving an easy way to take a snapshot of any schema, table or database. This creates a derived copy of that object which initially shares the underlying storage without any cost.
- Time travel: Enabling access to historical data which may have been changed or deleted.
- Data sharing: Enabling account-to-account sharing of data through tables, secured views and UDFs.

In addition to data ingest via Snowpipe, Snowflake offers a variety of connectivity options for applications, including:

- JDBC driver
- ODBC driver
- A query CLI called SnowQL
- a Web UI

It may be possible to use other integrations such as the JDBC sink connector with Kafka, but the native connector is preferred.

Product Components

It's insufficient to describe Snowflake as simply a "cloud data warehouse" without appreciating the architecture built for a scalable multi-tenant service engineered ground up to run in a SaaS offering. Snowflake layers include:

Storage Layer - Databases: Comprises the storage layer. Databases are logical groupings of objects consisting primarily of tables and views organized into schemas. Data is stored in a columnar format and encrypted using AES-256.

Compute Layer - Virtual Warehouses: Constitutes the compute layer of Snowflake. They're sized bundles of compute resources provisioned from a cloud provider and made available to end users for query execution. They can be dynamically allocated and destroyed and elastically resized. Each virtual data warehouse represents an independent compute cluster that can access the same shared data concurrently without contention.

Services Layer: Coordinates and manages the entire snowflake system. It authenticates users, manages sessions, compiles queries, performs query optimization and access coordination.

A high-level illustration of this architecture (from the Snowflake Concepts video) follows:

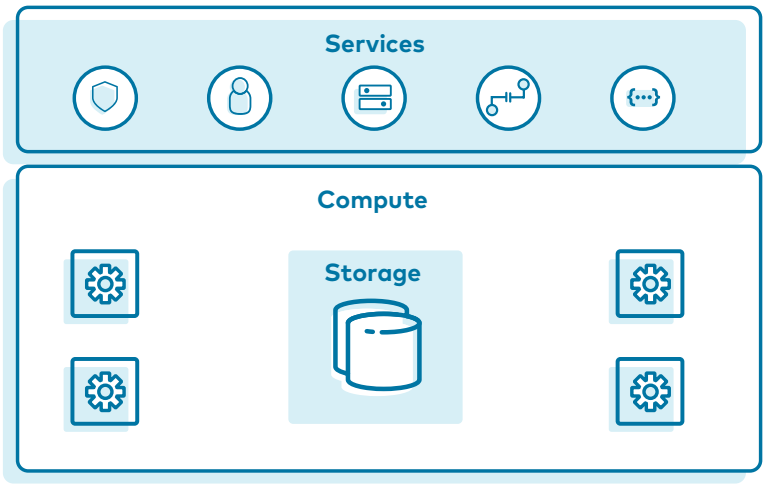


Figure 1: Snowflake Architecture at a high-level

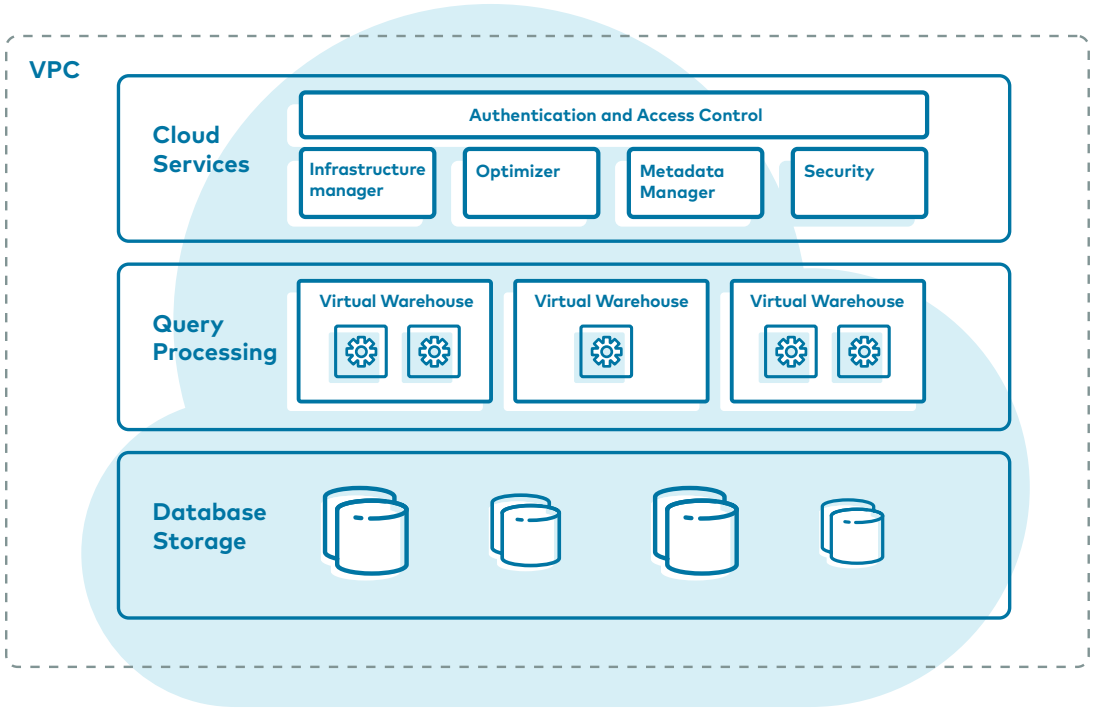


Figure 2: Snowflake Architecture — detailed

Integrations

Automatic Data Loading via Snowpipe and S3

The most commonly deployed integration is to dump the data from Kafka into a cloud store using the S3 connector as illustrated below, and then to use Snowpipe to ingest the batched files into Snowflake. This implementation pattern is illustrated below:

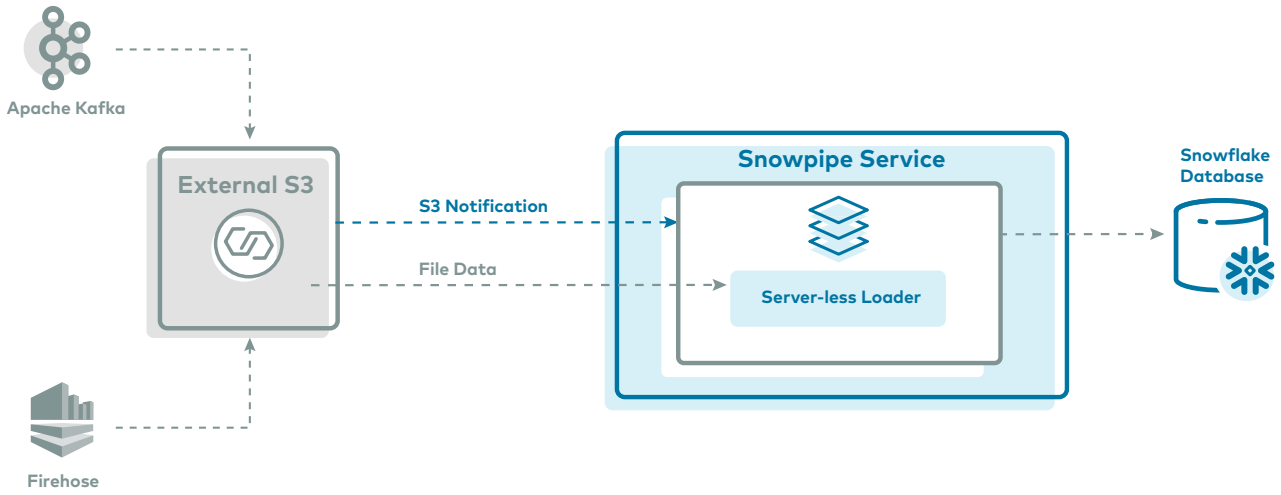


Figure 3: Common integration pattern with Kafka and Snowflake without a Kafka Connector

Connector

The connector automates that ingestion process, batching data from Kafka into Snowflake via the Snowpipe service.

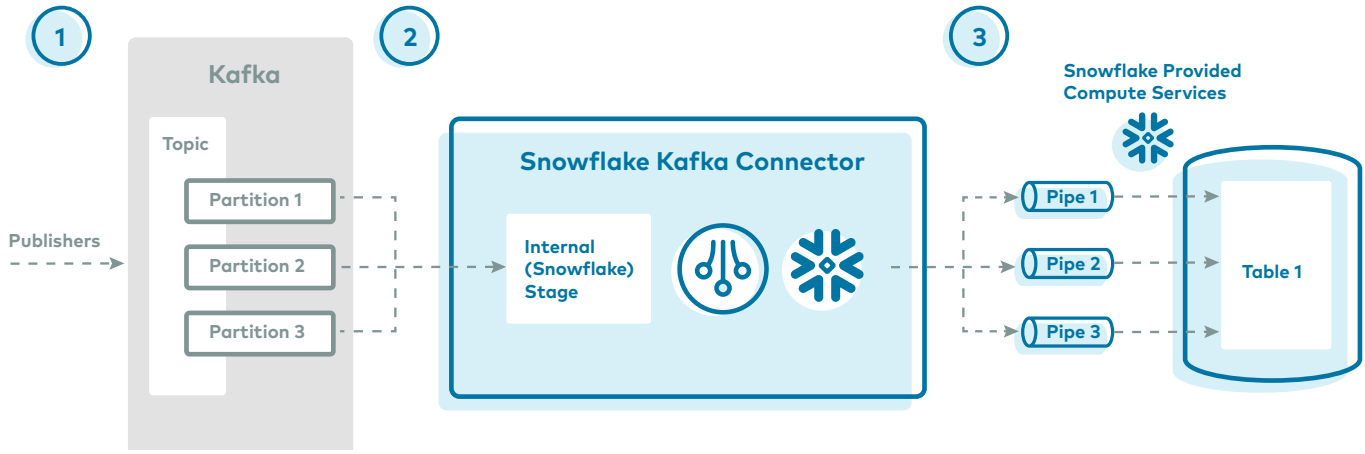


Figure 4: Operations of the Kafka Connector

1. One or more applications publish JSON or Avro records to a Kafka cluster. The records are split into one or more topic partitions.

2. The Kafka connector buffers messages from the Kafka topics. When a time (buffer.flush.time defaulting to 30 seconds), memory (buffer.size.bytes defaulting to 5MB) or message count (buffer.count.records defaulting to 10000 records) threshold is reached, the connector writes the messages to a temporary file in the internal stage. This triggers Snowpipe to ingest the temporary file. Snowpipe copies a pointer to the data file into a queue. While the default settings are a good start, the optimal tuning for these thresholds is going to depend on the workload and the environment.

3. A Snowflake-provided virtual warehouse loads data from the staged file into the target table (i.e. the table specified in the configuration file for the topic) via the pipe created for the Kafka topic partition.

4. Not shown: The connector monitors Snowpipe and deletes each file in the internal stage after confirming that the file data was loaded into the table. If a failure occurred loading the data, the connector moves the file into the table stage and produces an error message logged in the connector log file.

5. The connector repeats steps 2-4.

Verification

Snowflake has verified multiple versions of the connector with Confluent, including versions 0.3, 1.0, 1.1 and 1.2. The verification followed the guidelines of the Verified Integration Program, the results of which are listed below. More information on the verification is available from Confluent. For details on checklist omissions and negative results, see the table below.

OVERALL	
Implement the Kafka Connect API*	✓
Source or Sink*	Sink
Fully supported*	✓
Technical and Business Contacts*	✓
License*	✓
FUCTIONALITY	
Usable with Avro or JSON converters	✓
Uses or supports Single Message Transforms	✗
Exactly-once support	✓
Configurable*	✓
INTERNALS AND SUPPORTABILITY	
Error handling*	N/a
Logging*	✓
Metrics	✓
Graceful back-off	✓
Packaging*	✓

TESTING*	
Unit tests	✓
System tests	✓
Confluent Platform integration tests	✓
Performance tests	✓
DOCUMENTATION	
Connect API version used*	✓
Confluent platform supported versions*	✓
Partner product supported versions*	✓
Supported data types*	✓
Schema evolution compatibility policies	✓
Configurations*	✓
Quickstart guide / Tutorial	✓

Deployment

Hybrid on-prem / in-Cloud

Snowflake is a fully-managed cloud data warehouse operated as a service. Snowflake handles all the complexity of deploying, managing, and operation on the cloud service provider of your choice.

The canonical joint-deployment of this integration involves a local / on-prem deployment of Confluent Platform, including Kafka Connect running the Snowflake Kafka Connector.

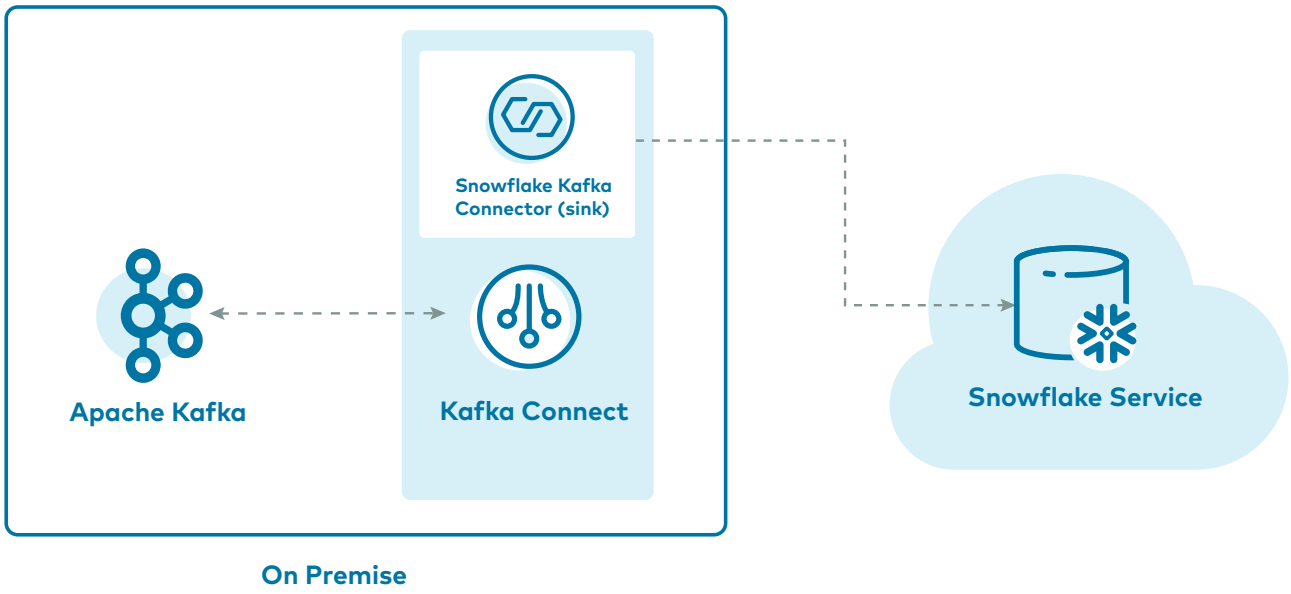


Figure 5: Hybrid cloud: Self-managed Confluent Platform (including Kafka and Kafka Connect) connecting to Snowflake

Assuming you have Confluent Platform already running pre-installed, connecting to Snowflake involves these high level steps:

1. Set up Kafka Connect
2. Create a private / public keypair
3. Set up authentication and authorization in Snowflake
4. Deploy and configure the connector
5. Use the connector

See the tutorial on page 10 for more information.

Managed-Cloud

By mid-2020, the Snowflake Connector for Kafka will be hosted in Confluent Cloud as a preview and configured to connect with Snowflake when the two services are running in the same cloud and region.

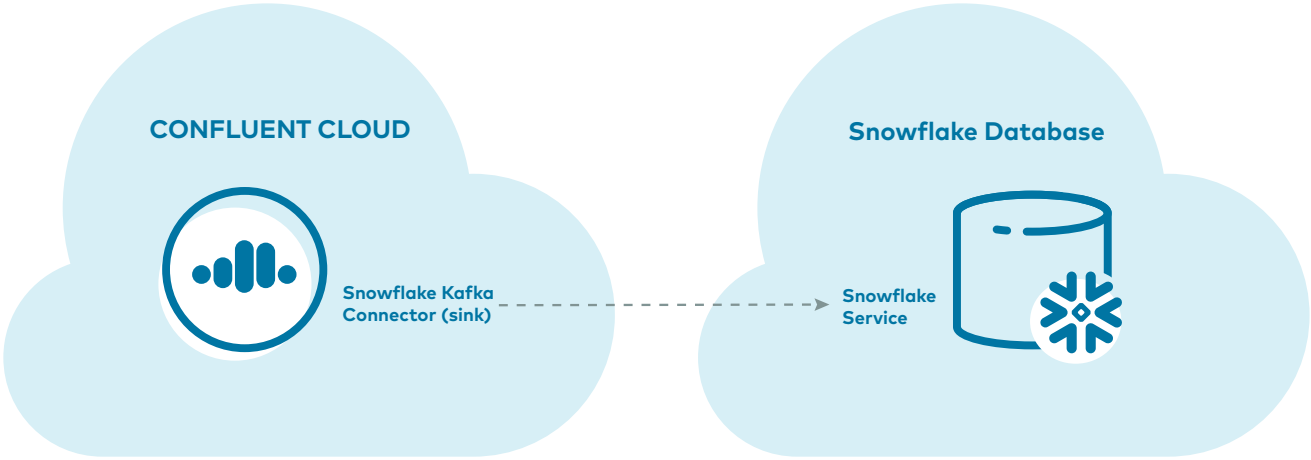


Figure 6: Managed Cloud

Considerations

Kafka Connect for Cloud

The Snowflake Connector is planned to be made available in Confluent Cloud in the Second half of 2020 as a preview. However, until this is the case, Kafka Connect will need to be self-managed. The connector will also need to be manually configured and deployed. Documentation and examples for connecting Kafka Connect to Confluent Cloud is found in the resources section.

Order of Insert

Although Snowflake Connector guarantees exactly-once delivery, and although Kafka guarantees strong ordering of data in a topic, the connector does not guarantee that rows are inserted in the order that they were originally published in Kafka.

This is because Snowpipe doesn't offer the same ordering guarantees provided by Kafka. Records are batched in parallel to an eventually consistent datastore such as s3, losing the ordering. While this doesn't alter the data, it could mean that data arrives in Snowflake out of order. Therefore, if you're doing real-time transformations, aggregations etc. in which strictly following event order is important, the recommendation is to do so in Kafka, KSQL or Kafka Streams. This will ensure that the strict ordering guarantees provided by Kafka are in place for the transformations.

Single Message Transforms

Snowflake provides its own converters and Single Message Transforms are not supported. The Snowflake converter provides a customized object to pass broken records from the converter to SinkTask, allowing the broken records to propagate through to the database where they are handled without having to refer back to the Dead Letter Queue. Currently Kafka Connect provides two error handling options: "none" and "all". "None" configures the connector to fail fast and "all" ignores broken records. Snowflake proposed a third option, "continue" which allows broken records to pass through to SinkTask where they are handled by the SinkTask. This proposal can be found in [#KAFKA-9740] Add a "continue" option for Kafka Connect error handling - ASF JIRA. SMTs and standard converters will not be supported until this issue is resolved or worked around.

Snowflake Identifiers

The connector name must be a valid Snowflake identifier. See this issue on the Snowflake connector repo.

Limit Connector to One Instance

Instances of the Kafka connector do not communicate with each other. If you start multiple instances of the connector on the same topics or partitions, then multiple copies of the same row might be inserted into the table. This is not recommended; each topic should be processed by only one instance of the connector.

Tutorial

Robin Moffat, Developer Advocate at Confluent, wrote the following tutorial for getting started with the Snowflake connector. It presumes a local version of Confluent Platform installed already. See his blog for more information.

We'll revise this tutorial for a simpler setup when the managed connector is available in Confluent Cloud.

Set up Snowflake account and key pair

Generate the keys:

```
# Create Private key - keep this safe, do not share!
openssl genrsa -out snowflake_key.pem 2048
# Generate public key from private key. You can share your public key.
openssl rsa -in snowflake_key.pem  -pubout -out snowflake_key.pub
```

You should now have two files:

```
$ ls -l snowflake_key*
-rw-r--r--  1 rmoff  staff  1679 21 Nov 09:28 snowflake_key.pem
-rw-r--r--  1 rmoff  staff   451 21 Nov 09:28 snowflake_key.pub
```

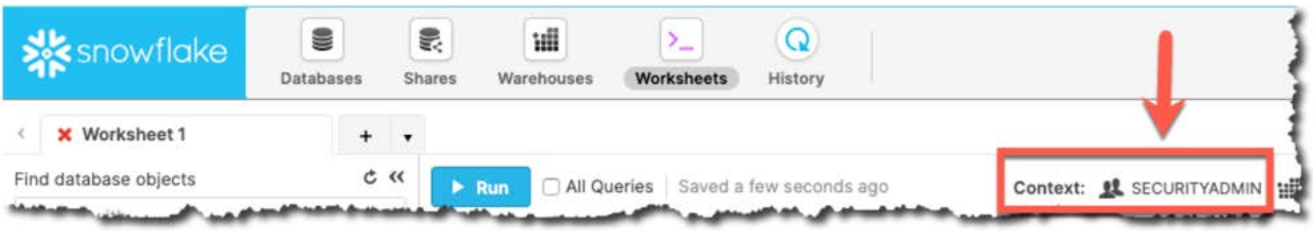
Obtain the public key:

```
$ cat snowflake_key.pub
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAya/BRlyhsfdlJQnPqoRn
lJfxKxujoyionNBPIDFpVpGZ9C1ZE7Q1kGIrEoZfq1t2p6lT8cX6gIZkMDF10I/8
yqHGiCdSEQBuMYXwWpnl3C1sttFHNfxbsjiKSZDlMTbEmzwU5s5LpMt8YvFWp8Iu
3ilHK9Vwy0wbsMDCjDcrC6xCS6qp1n4oso+V24aaxKd/mUtpPy9toAx2NC5GMoDb
tehlbTyPkk/9qFl7GUsf46HbQMEGoGkRrY9VFm+3Z8wCwsFNpURIvLEBcrTFdnmn
IgDBa96+dKgaN8qV6RW3ZMheQOJH1tP3M0qXsLNbR00E7yAlCYjNQD3hXjGKL3Oc
5wIDAQAB
-----END PUBLIC KEY-----
```

But minus the header and footer and joined over a single line. You can do this manually, or by script:

```
$ grep -v "BEGIN PUBLIC" snowflake_key.pub | grep -v "END PUBLIC"|tr -d \n
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAya/BRlyhsfdlJQnPqoRnlJfxKxujoyionNBPIDFpVpGZ9C1ZE
7Q1kGIrEoZfq1t2p6lT8cX6gIZkMDF10I/8yqHGiCdSEQBuMYXwWpnl3C1sttFHNfxbsjiKSZDlMTbEmzwU5s5LpMt8Yv
FWp8Iu3ilHK9Vwy0wbsMDCjDcrC6xCS6qp1n4oso+V24aaxKd/mUtpPy9toAx2NC5GMoDbtehlbTyPkk/9qFl7GUsf46H
bQMEGoGkRrY9VFm+3Z8wCwsFNpURIvLEBcrTFdnmnIgDBa96+dKgaN8qV6RW3ZMheQOJH1tP3M0qXsLNbR00E7yAlCYjN
QD3hXjGKL3Oc5wIDAQAB
```

Create a snowflake user. Switch to security admin role:

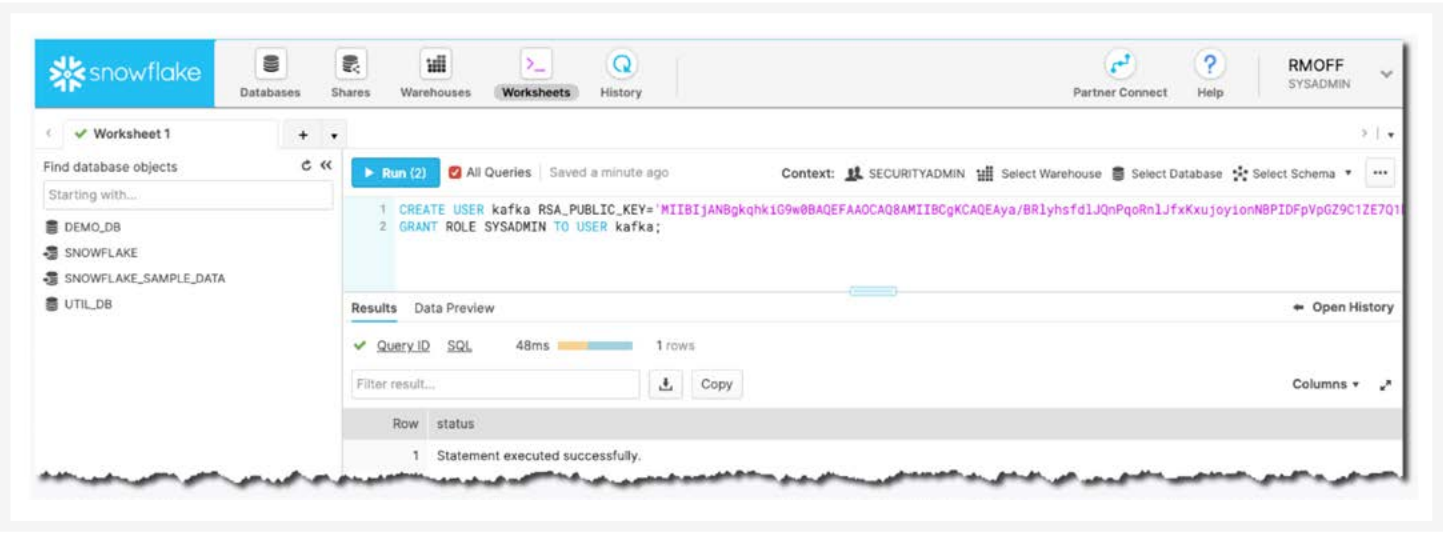


NOTE Make sure you do this in the `Context` section of the worksheet, not the top-right dropdown (otherwise you'll get `SQL access control error: Insufficient privileges to operate on account 'xyz'`).

Now create a user for Kafka. Here we also grant Kafka full access for simplicity.

```
CREATE USER kafka
RSA_PUBLIC_KEY='MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAya/BRlyhsfdlJQnPqoRnlJfxKxujoyion
NBPIDFpVpGZ9C1ZE7Q1kGIrEoZfq1t2p6lT8cX6gIZkMDF10I/8yqHGiCdSEQBuMYXwWpnl3C1sttFHNfxbsjiKSZDlMT
bEmzwU5s5LpMt8YvFWp8Iu3ilHK9Vwy0wbsMDCjDcrC6xCS6qp1n4oso+V24aaxKd/mUtpPy9toAx2NC5GMoDbtehlbTy
Pkk/9qFl7GUsf46HbQMEGoGkRrY9VFm+3Z8wCwsFNpURIvLEBcrTFdnmnIgDBa96+dKgaN8qV6RW3ZMheQOJH1tP3M0qX
sLNbR00E7yAlCYjNQD3hXjGKL3Oc5wIDAQAB'
```

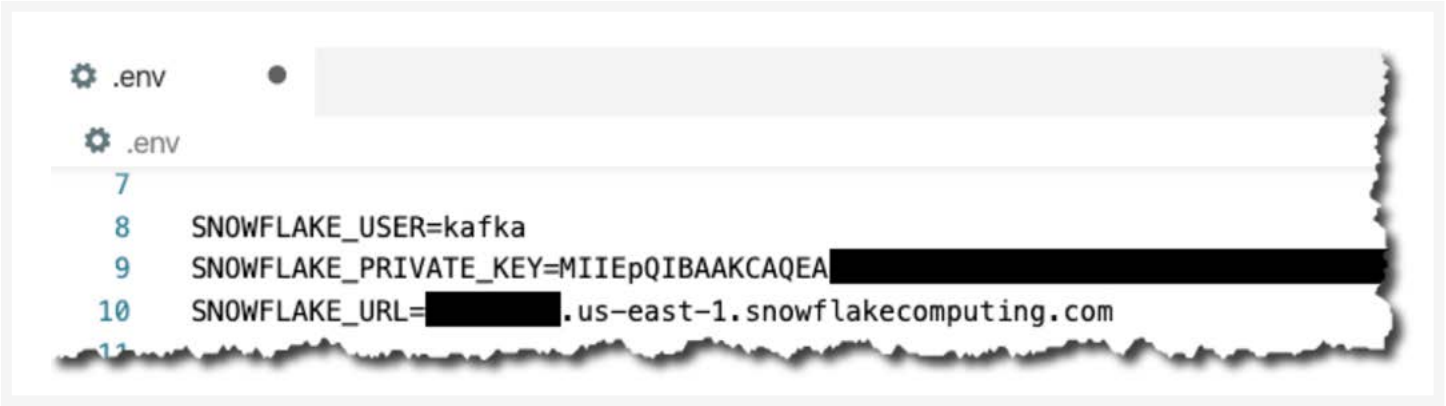
```
GRANT ROLE SYSADMIN TO USER kafka;
```



Provide the private key:

NOTE Your private key is private — don't share it with anyone who shouldn't have access to the account, and definitely don't put it in a reference architecture document!

Put this value, along with the URL of your Snowflake environment and the user that we created (kafka) in the .env file.



Make this .env file available to Kafka Connect. In the example below, it's in /data/credentials.properties.

Setting up the Snowflake connector

On your local system, cluster, or in the same cloud provider and region as the target Snowflake DB:

Install the connector:

```
confluent-hub install --no-prompt snowflakeinc/snowflake-kafka-connector:0.5.5
```

Restart the Kafka Connect connector and check that it's been loaded:

```
$ curl -s localhost:8083/connector-plugins|jq '.[].class'|grep snowflake
"com.snowflake.kafka.connector.SnowflakeSinkConnector"
```

Now set up your connector configuration. A few important settings of note:

- **topics** — A comma separated list of one or more topics that are to be streamed to Snowflake. You can optionally map topics to table names with snowflake.topic2table.map but this is not mandatory.
- **value.converter** — Snowflake provide their own converters (see considerations above). Use either:
  - com.snowflake.kafka.connector.records.SnowflakeAvroConverter
  - com.snowflake.kafka.connector.records.SnowflakeJsonConverter
- **Authentication / sensitive information** is referred to in a separate file that's loaded by the connector directly:
  - snowflake.url.name
  - snowflake.user.name — we created the user kafka for this above
  - snowflake.private.key — this is the key that we extracted in the step above

Create the connector:

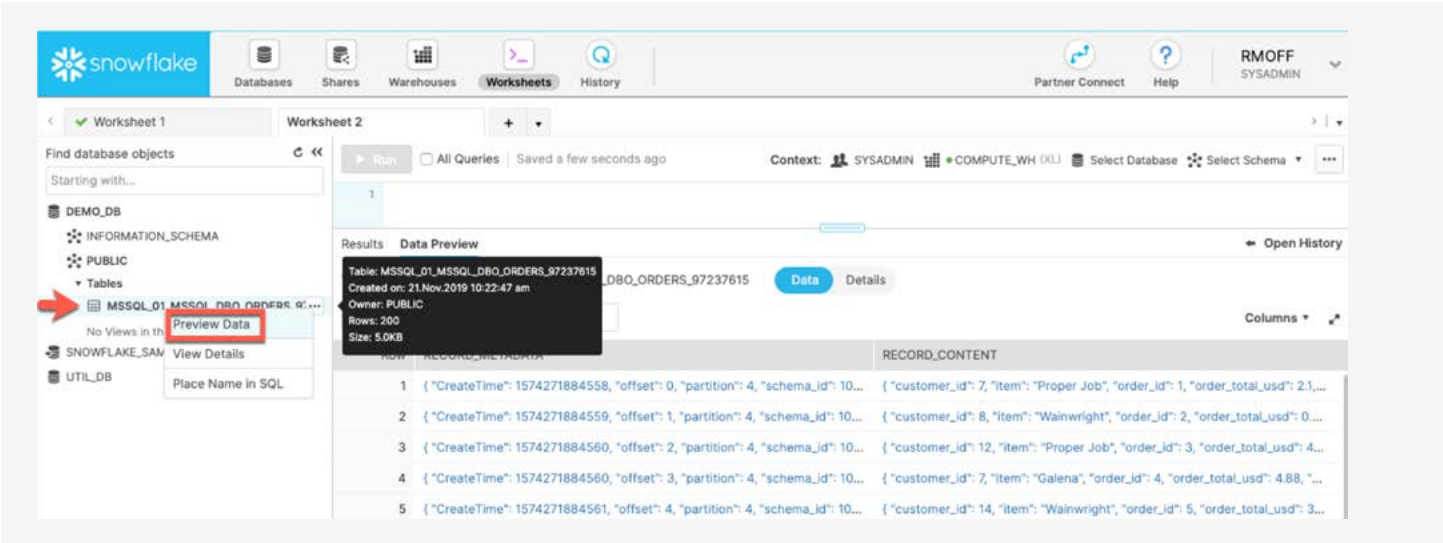
```
curl -i -X PUT -H  "Content-Type:application/json" \
    http://localhost:8083/connectors/sink_snowflake_01/config \
    -d '{
```

```
        "connector.class":"com.snowflake.kafka.connector.SnowflakeSinkConnector",
        "tasks.max":1,
        "topics":"mssql-01-mssql.dbo.ORDERS",
        "snowflake.url.name":"${file:/data/credentials.properties:SNOWFLAKE_HOST}",
        "snowflake.user.name":"${file:/data/credentials.properties:SNOWFLAKE_USER}",
        "snowflake.user.role":"SYSADMIN",
        "snowflake.private.key":"${file:/data/credentials.properties:SNOWFLAKE_PRIVATE_KEY}",
        "snowflake.database.name":"DEMO_DB",
        "snowflake.schema.name":"PUBLIC",
        "key.converter":"org.apache.kafka.connect.storage.StringConverter",
        "value.converter":"com.snowflake.kafka.connector.records.SnowflakeAvroConverter",
        "value.converter.schema.registry.url":"https://${file:/data/credentials.properties:CCLOUD_
SCHEMA_REGISTRY_HOST}",
        "value.converter.basic.auth.credentials.source":"USER_INFO",
        "value.converter.basic.auth.user.info":"${file:/data/credentials.properties:CCLOUD_SCHEMA_REG-
ISTRY_API_KEY}:${file:/data/credentials.properties:CCLOUD_SCHEMA_REGISTRY_API_SECRET}"
    }'
```

Check that it's running:

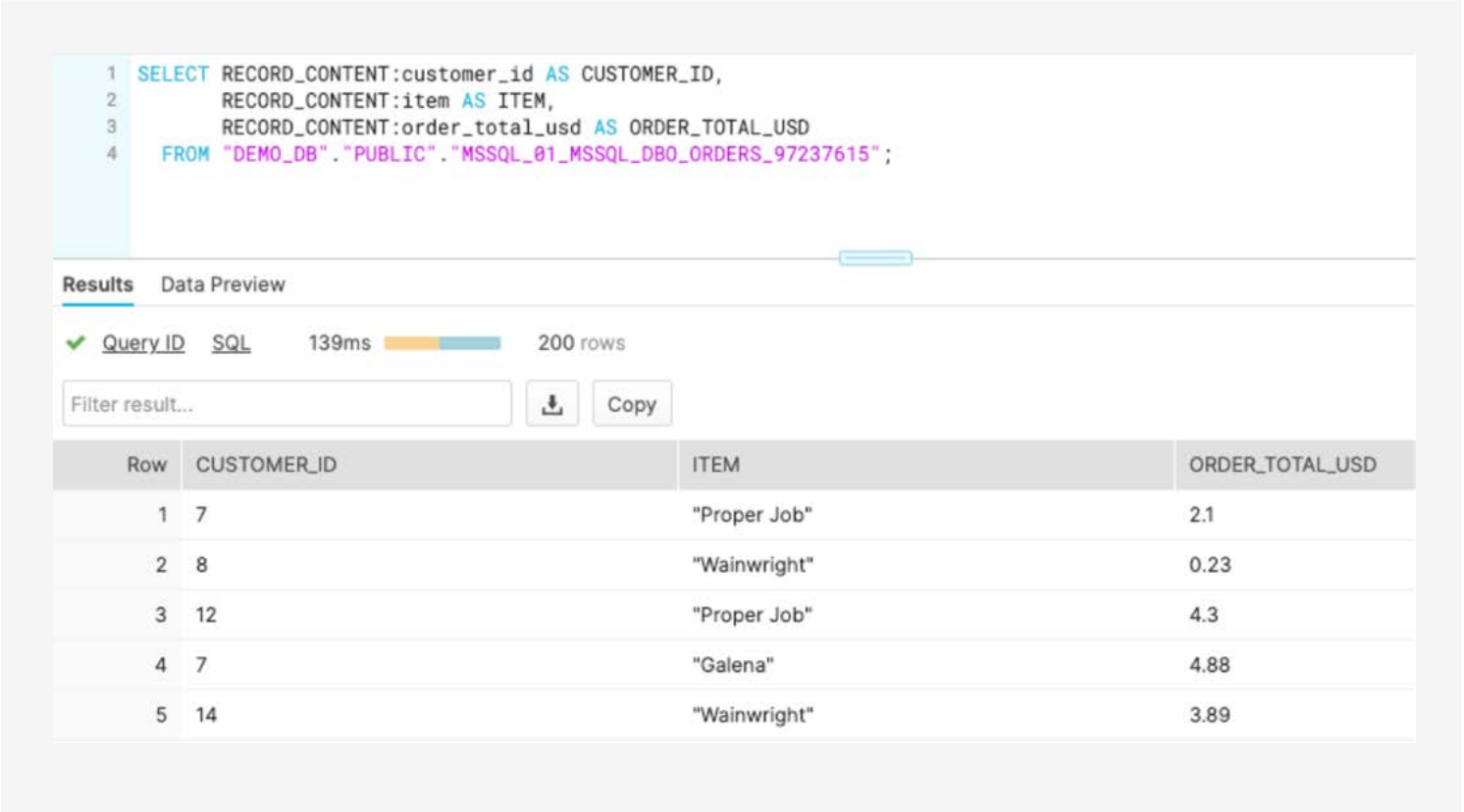
```
$ curl -s "http://localhost:8083/connectors?expand=info&expand=status" | \
           jq '. | to_entries[] | [ .value.info.type, .key, .value.status.connector.state,.value.
status.tasks[].state,.value.info.config."connector.class"]|join(":|:")' | \
           column -s : -t| sed 's/\"//g'| sort
sink    |  sink_snowflake_01          |  RUNNING  |  RUNNING  |
com.snowflake.kafka.connector.SnowflakeSinkConnector
```

Check Snowflake:



The connector writes the Kafka message payload to the RECORD_CONTENT field and its metadata (partition, offset, etc) to RECORD_METADATA. You can access the nested values using the colon as a seperator, e.g.:

```
SELECT RECORD_CONTENT:customer_id AS CUSTOMER_ID,
       RECORD_CONTENT:item AS ITEM,
       RECORD_CONTENT:order_total_usd AS ORDER_TOTAL_USD
  FROM "DEMO_DB"."PUBLIC"."MSSQL_01_MSSQL_DBO_ORDERS_97237615";
```



Row	CUSTOMER_ID	ITEM	ORDER_TOTAL_USD
1	7	"Proper Job"	2.1
2	8	"Wainwright"	0.23
3	12	"Proper Job"	4.3
4	7	"Galena"	4.88
5	14	"Wainwright"	3.89

Additional Resources

- Connector on Confluent Hub https://www.confluent.io/hub/snowflakeinc/snowflake-kafka-connector
- Source code: https://github.com/snowflakedb/snowflake-kafka-connector

Concepts, Documentation and Training

- Snowpipe: Load data fast, analyze even faster (Snowpipe intro)
- Getting Started — Architecture & Key Concepts (Snowflake basics)
- Snowflake Connector for Kafka (Snowflake Kafka Connector user's guide)
- Connecting to Snowflake (documentation on connecting to Snowflake as a source)
- Introduction to Snowpipe (Snowpipe user's guide)
- Logs & Offsets: (Near) Real Time ELT with Apache Kafka + Snowflake (Example without the connector)
- Streaming From Kafka to Snowflake : Part 1 — Kafka to S3 (Example without the connector)
- Technology Partners: Tableau, Alteryx, Amazon & More (Snowflake's partner ecosystem excluding Confluent)
- Revolutionary features of Snowflake that sets it apart — A Deep dive (A deeper dive on Snowflake features)

Tutorials and Quickstarts

Pipeline to the Cloud — On-Premises Data Streaming for Cloud Analytics (Confluent blog including Snowflake)

Streaming data from SQL Server to Kafka to Snowflake with Kafka Connect (Robin Moffat's personal blog on using the Kafka Connector in tandem with SQLServer)

Confluent Demo Scene — Pipeline to the Cloud (Confluent github repo including snowflake)