**Snowflake and OpenWeather API Integration**

Contents

[1. Project Overview 1](#_Toc193622017)

[1.1 Objective 1](#_Toc193622018)

[1.2 Benefits 1](#_Toc193622019)

[2. Solution Architecture 1](#_Toc193622020)

[2.1 Architecture Components 1](#_Toc193622021)

[2.2 Data Flow 2](#_Toc193622022)

[3. Implementation Details 2](#_Toc193622023)

[3.1 AWS API Gateway Setup 2](#_Toc193622024)

[3.2 AWS Lambda Function 2](#_Toc193622025)

[3.3 IAM Role & Policy 2](#_Toc193622026)

[3.4 Terraform Deployment 3](#_Toc193622027)

[3.5 Snowflake SQL Configuration 3](#_Toc193622028)

[4. Testing & Validation 4](#_Toc193622029)

[4.1 API Gateway Validation 4](#_Toc193622030)

[4.2 Snowflake Query Validation 5](#_Toc193622031)

[5. Security & Compliance 5](#_Toc193622032)

[6. Deployment & Maintenance 5](#_Toc193622033)

[6.1 Deployment Steps 5](#_Toc193622034)

[6.2 Ongoing Maintenance 5](#_Toc193622035)

[7. Conclusion 6](#_Toc193622036)

# 1. Project Overview

## 1.1 Objective

This project aims to integrate Snowflake with the OpenWeatherMap API using AWS API Gateway and Lambda. The integration allows Snowflake users to fetch real-time weather data via SQL queries, leveraging an external function that securely calls the OpenWeatherMap API through AWS API Gateway.

## 1.2 Benefits

* **Serverless & Scalable**: API Gateway and Lambda handle requests efficiently without requiring dedicated servers.
* **Secure Integration**: Uses IAM roles and API Gateway authentication to protect API access.
* **Seamless SQL Queries**: Enables Snowflake users to fetch weather data using standard SQL queries.
* **Automated Deployment**: Terraform provisions AWS resources for quick setup and reproducibility.

# 2. Solution Architecture

## 2.1 Architecture Components

* **Snowflake**: Executes SQL queries using an external function.
* **AWS API Gateway**: Acts as an intermediary to forward requests to OpenWeatherMap.
* **AWS Lambda**: Processes API calls and formats responses.
* **IAM Roles**: Grants Snowflake permission to invoke API Gateway securely.
* **Terraform**: Automates the provisioning of AWS API Gateway, Lambda, and IAM policies.
* **OpenWeatherMap API**: Provides real-time weather data.

## 2.2 Data Flow

1. A user runs SELECT get\_weather('San Diego'); in Snowflake.
2. Snowflake calls the External Function linked to AWS API Gateway.
3. API Gateway forwards the request to AWS Lambda.
4. Lambda fetches weather data from OpenWeatherMap.
5. OpenWeatherMap responds with weather data.
6. Snowflake returns the API response to the user.

# 3. Implementation Details

## 3.1 AWS API Gateway Setup

* Deploys a **REST API Gateway** named openweather\_api.
* Defines /weather as an endpoint supporting GET and POST requests.
* Integrates API Gateway with AWS Lambda using **AWS\_PROXY** mode.
* Deploys the API Gateway in the prod stage.
* Grants API Gateway permission to invoke Lambda.

## 3.2 AWS Lambda Function

* The Lambda function (openweather\_lambda) is deployed using Python 3.11.
* It processes requests from API Gateway and fetches weather data using the OpenWeatherMap API.
* Uses an environment variable for the OpenWeatherMap API key.
* Returns formatted weather data, including city name, temperature, weather description, and wind speed.

## 3.3 IAM Role & Policy

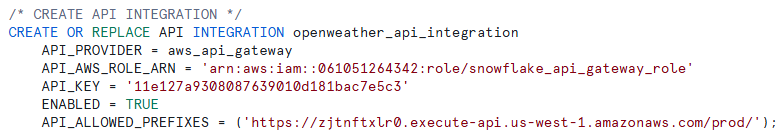
* **IAM Role for Lambda**: Grants basic execution permissions for CloudWatch logging.
* **IAM Role for Snowflake**: Allows Snowflake to invoke API Gateway securely.
* **IAM Role for CloudWatch Logs**: Enables API Gateway logging.
* **IAM Policy Attachments**:
  + Lambda has an **AWSLambdaBasicExecutionRole**.
  + Snowflake has execute-api:Invoke permissions for API Gateway.
  + API Gateway can push logs to CloudWatch.

## 3.4 Terraform Deployment

* **Provider Configuration**: AWS resources are deployed in us-west-1.
* **Variables**: Terraform uses variable-based configuration for flexibility.
* **Resources Created**:
  + API Gateway
  + Lambda Function
  + IAM Roles & Policies
  + CloudWatch Logging
* **Outputs**:
  + API Gateway Invoke URL
  + Snowflake IAM Role ARN
  + CloudWatch IAM Role ARN

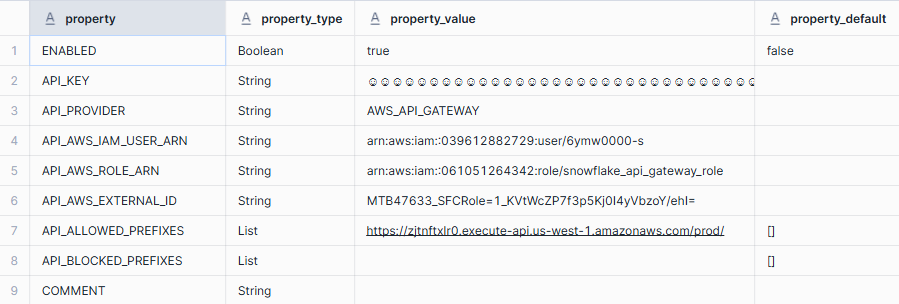
## 3.5 Snowflake SQL Configuration

**Create API Integration**

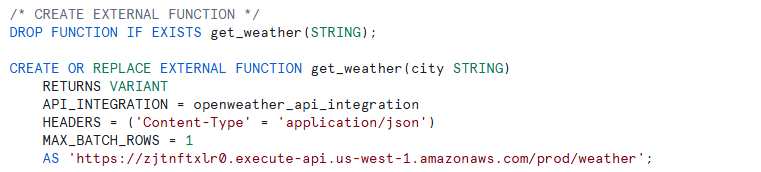


**Pull properties for IAM Snowflake role**



****

**Create External Function**



# 4. Testing & Validation

## 4.1 API Gateway Validation

* Call API Gateway endpoint:

curl -X POST "https://zjtnftxlr0.execute-api.us-west-1.amazonaws.com/prod/weather" -H "Content-Type: application/json" -d '{"city": "San Diego"}'

* Validate response format and data accuracy.

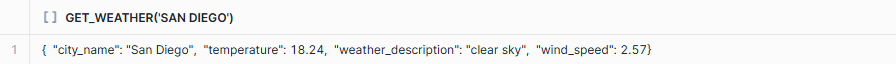
## 4.2 Snowflake Query Validation

**Test the function**

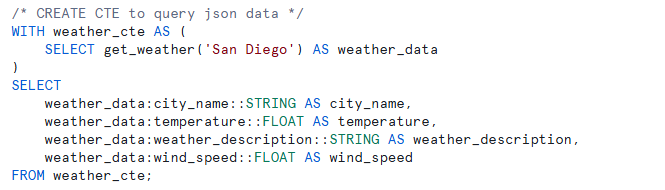
* Run test queries

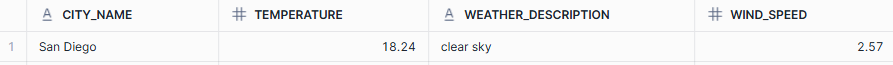


* Confirm JSON output with temperature, humidity, and weather conditions.



**Create CTE to query JSON data**





# 5. Security & Compliance

* **IAM Role Restriction**: Ensures only Snowflake can invoke API Gateway.
* **API Key Management**: OpenWeatherMap API key is stored securely.
* **Rate Limiting**: Configure OpenWeatherMap limits to avoid excessive usage.
* **CORS & Authentication**: Restrict API access to authorized sources only.

# 6. Deployment & Maintenance

## 6.1 Deployment Steps

1. Run terraform apply to provision AWS resources.
2. Configure Snowflake API Integration and External Function.
3. Test API Gateway and Snowflake SQL queries.
4. Deploy additional logging or monitoring if needed.

## 6.2 Ongoing Maintenance

* **Monitor API Usage**: Track API calls and optimize as needed.
* **Update IAM Policies**: Adjust permissions if new integrations are required.
* **Rotate API Keys**: Secure OpenWeatherMap API key and update Snowflake function.

# 7. Conclusion

This solution provides a secure, scalable, and efficient method for Snowflake users to access live weather data using SQL queries. The integration of AWS API Gateway ensures smooth and controlled API access, while Terraform enables quick and repeatable deployment.

## Next Steps

* Deploy Terraform and validate the Snowflake API function.
* Implement additional API integrations for broader datasets.
* Optimize query performance and caching for frequent API calls.