

Building a Real-Time Weather Data Pipeline for Weather Analytics

1. Create S3 Bucket using Terraform

IAM User with Access code to create S3 bucket from Terraform

The screenshot shows the AWS IAM User details page for 'neha_iam1'. The user was created on June 20, 2025, at 06:10 UTC-04:00. It has 'Console access Enabled with MFA'. Two access keys are listed: 'Access key 1' (AKIAITASPUUSUOLMF7Z255 - Active, used 23 hours ago, 31 days old) and 'Access key 2' (Create access key). The 'Permissions' tab is selected, showing two policies attached: 'AdministratorAccess' (AWS managed - job function, Directly) and 'AmazonS3FullAccess' (AWS managed, Directly). A search bar and filter by type dropdown are also visible.

Create a Terraform Configuration File and run it:

The screenshot shows the Terraform IDE interface. In the center, the code editor displays the `main.tf` file:

```

resource "aws_s3_object" "bronze_folder"
  terraform {
    required_providers {
      aws = {
        source  = "hashicorp/aws"
        version = ">= 5.0"
      }
    }
  }

# Configure AWS Provider
provider "aws" {
  region = "us-east-1"      # Change to your preferred region
  access_key = "AKIATAPUSJULMF7ZSS5"
  secret_key = "BbxK9RKAT9lpHT90FrdHvrekVX/TdGws1SeL5"
}

# Create the S3 bucket
resource "aws_s3_bucket" "data_lake" {
  bucket = "neha-bc-project-003" # must be globally unique
  tags = {
    Name     = "DataLakeBucket"
    Environment = "Dev"
  }
}

```

Below the code editor, the terminal window shows the output of the `terraform apply` command:

```

aws_s3_object.gold_folder: Refreshing state... [id=gold/]
aws_s3_object.silver_folder: Refreshing state... [id=silver/]
aws_s3_object.bronze_folder: Refreshing state... [id=bronze/]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

```

The status bar at the bottom indicates the current profile is `AWS: profile:default`.

Verify whether the bucket is created as expected:

The screenshot shows the AWS S3 console. On the left, the navigation menu includes options like General purpose buckets, Directory buckets, Table buckets, Vector buckets, Access Grants, Access Points for general purpose buckets, Access Points for directory buckets, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, and IAM Access Analyzer for S3. Under Storage Lens, there are links for Dashboards, Storage Lens groups, and AWS Organizations settings.

The main content area displays the "General purpose buckets" section. It shows a table with one item:

Name	AWS Region	Creation date
neha-bc-project-003	US East (N. Virginia) us-east-1	July 21, 2025, 22:03:12 (UTC-04:00)

On the right, there are two informational boxes: "Account snapshot" and "External access summary - new".

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with 'Amazon S3' navigation and various settings like 'General purpose buckets' and 'Storage Lens'. The main area is titled 'neha-bc-project-003' and shows a table of objects. The table has columns for Name, Type, Last modified, Size, and Storage class. It lists three entries: 'bronze/' (Folder), 'gold/' (Folder), and 'silver/' (Folder). There are also buttons for Actions, Create folder, and Upload.

Bronze Layer

1. Create Kinesis Data Stream

The screenshot shows the AWS Kinesis service landing page. It features a large central banner with the text 'Amazon Kinesis services' and 'Collect, process, and analyze data streams in real time.' To the right, there's a 'Get started' section with three options: 'Kinesis Data Streams' (selected), 'Amazon Data Firehose - new (Formerly Kinesis Data Firehose)', and 'Managed Apache Flink (Formerly Kinesis Data Analytics)'. Below this is a 'Pricing (United States (N. Virginia))' section showing 'Amazon Kinesis Data Streams' at '\$0.015 per Hour'. At the bottom, there's a 'How it works' section with tabs for 'Kinesis Data Streams', 'Amazon Data Firehose', and 'Managed Apache Flink'.

Screenshot of the AWS Kinesis Data Streams 'Create data stream' configuration page.

Data stream configuration

Data stream name: weather-stream

Capacity mode: Provisioned (selected)

Provisioned shards: 1 (Shard estimator)

Total data stream capacity: 1 MiB/second and 1,000 records/second

Write capacity: 1 MiB/second and 1,000 records/second

Read capacity: 2 MiB/second

Data stream settings:

Setting	Value	Editable after creation
Capacity mode	Provisioned	Yes
Provisioned shards	1	Yes
Data retention period	1 day	Yes
Server-side encryption	Disabled	Yes
Monitoring enhanced metrics	Disabled	Yes
Data stream sharing policy	No policy	Yes

Tags - optional: No tags associated with the Kinesis Data Stream.

Screenshot of the AWS Kinesis Data Streams 'Create data stream' configuration page, showing the throughput settings.

Capacity mode: Provisioned mode has a fixed-throughput pricing model. See [Kinesis pricing for Provisioned mode](#).

Throughput settings:

Setting	Value
1 MiB/second and 1,000 records/second	2 MiB/second

Data stream settings:

Setting	Value	Editable after creation
Capacity mode	Provisioned	Yes
Provisioned shards	1	Yes
Data retention period	1 day	Yes
Server-side encryption	Disabled	Yes
Monitoring enhanced metrics	Disabled	Yes
Data stream sharing policy	No policy	Yes

Tags - optional: No tags associated with the Kinesis Data Stream.

neha-bc-project-003 - S3 buck: X Kinesis | us-east-1 + Private browsing Search [Alt+S] United States (N. Virginia) neha_iam1 @ 2073-9921-8460

Amazon Kinesis > Data streams > Create data stream

Data Stream Settings

You can edit the settings after the data stream has been created and is in the active status.

Setting	Value	Editable after creation
Capacity mode	Provisioned	Yes
Provisioned shards	1	Yes
Data retention period	1 day	Yes
Server-side encryption	Disabled	Yes
Monitoring enhanced metrics	Disabled	Yes
Data stream sharing policy	No policy	Yes

Tags - optional Info

You can add tags to organize your AWS resources, track costs, and control access.

No tags associated with the Kinesis Data Stream.

Add new tag

You can add up to 50 more tags.

Cancel Create data stream

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Kinesis | us-east-1 us-east-1.console.aws.amazon.com/kinesis/home?region=us-east-1#streams/details/weather-stream/details Private browsing United States (N. Virginia) neha_iam1 @ 2073-9921-8460

Amazon Kinesis > Data streams > weather-stream

Creating data stream weather-stream.
Creating a new data stream can take up to 10 minutes. Learn more

weather-stream Info Delete

Data stream summary

Status	Creating	Capacity mode	Provisioned	
		Data retention period	1 day	
		ARN	arn:aws:kinesis:us-east-1:207399218460:stream/weather-stream	
			Creation time	July 21, 2025 at 22:09 EDT

Applications Monitoring Configuration Enhanced fan-out (0) Data viewer Data analytics - new Data stream shari

Producers Info

Producers put records into Kinesis Data Streams.

Amazon Kinesis Agent
Use a stand-alone Java software application to send data to the stream. [Learn more](#)

AWS SDK
Use AWS SDK for Java to develop producers. [Learn more](#)

Amazon Kinesis Producer Library (KPL)
Use KPL to develop producers. [Learn more](#)

[View in GitHub](#) [View in GitHub](#) [View in GitHub](#)

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the AWS Kinesis Data Stream creation page. A green success message at the top right says "Data stream weather-stream successfully created." The main section displays the "weather-stream" details:

Status	Capacity mode	ARN	Creation time
Active	Provisioned	arn:aws:kinesis:us-east-1:207399218460:stream/weather-stream	July 21, 2025 at 22:09 EDT
Data retention period		1 day	

Below this, tabs for Applications, Monitoring, Configuration, Enhanced fan-out (0), Data viewer, Data analytics - new, and Data stream sharing are visible. The Applications tab is selected.

The screenshot shows the AWS Kinesis Data Streams list page. A green success message at the top right says "Data stream weather-stream successfully created." The main section displays the "Data streams (1)" table:

Name	Status	Capacity mode	Provisioned shards	Sharing policy	Data retention period	Encryption	Consumers with enhanced fan-out
weather-stream	Active	Provisioned	1	No	1 day	Disabled	0

At the top of the table, there are buttons for "Process data in real time", "Create a Firehose stream", "Actions", and "Create data stream".

2. Prepare and Run Script to Stream Weather Data

First, create IAM Role with lambda as use case:

Screenshot of the AWS IAM 'Create role' wizard - Step 1: Name, review, and create.

Role details

Role name: LambdaRoleProject3

Description: Allows Lambda functions to call AWS services on your behalf.

Step 1: Select trusted entities

Trust policy:

```
1+ {
2+   "Version": "2012-10-17",
3+   "Statement": [
4+     {
5+       "Effect": "Allow",
6+       "Action": [
7+         "sts::AssumeRole"

```

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS IAM 'Create role' wizard - Step 2: Add permissions.

Permissions policy summary

Policy name	Type	Attached as
AmazonAthenaFullAccess	AWS managed	Permissions policy
AmazonKinesisFullAccess	AWS managed	Permissions policy
AmazonS3FullAccess	AWS managed	Permissions policy
AmazonS3ObjectLambdaExecutionRolePolicy	AWS managed	Permissions policy
AWSLambda_FullAccess	AWS managed	Permissions policy
CloudWatchLogsFullAccess	AWS managed	Permissions policy

Step 3: Add tags

Add tags - optional Info

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

[Add new tag](#)

You can add up to 50 more tags.

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the AWS IAM console with the LambdaRoleProject3 role selected. The left sidebar shows navigation options like Identity and Access Management (IAM), Access management, Roles, and Access reports. The main panel displays the role's summary, including its ARN (arn:aws:iam::207399218460:role/LambdaRoleProject3) and creation date (July 21, 2025, 22:15 (UTC-04:00)). The Permissions tab is active, showing six attached policies: AmazonAthenaFullAccess and AmazonKinesisFullAccess, both of which are AWS managed policies.

To run in CloudShell

1. Click the terminal icon on the AWS Console (bottom nav bar).

2. To Open the file:

a. nano weather_stream-project-3.py

3. Paste the streaming script into a file:

4. To save and exit nano:

a. Press Ctrl + O to write (save) the file

b. Press Enter to confirm

c. Press Ctrl + X to exit

5. Make it executable:

a. chmod +x weather_stream-project-3.py

6. Run it:

a. Python3 weather_stream-project-3.py

The screenshot shows the AWS CloudShell interface in a browser window. The title bar includes tabs for 'neha-bc-project-003 - S3 buck:', 'Kinesis | us-east-1', 'CloudShell | us-east-1' (which is active), and 'LambdaRoleProject3 | IAM | Gi...'. The main area displays a Python script named 'weather_stream-project-3.py'. The script imports modules like boto3, json, time, and requests, and creates a Kinesis client. It defines an API key and a list of cities (New York, Los Angeles, Mexico City, Toronto, Chicago, Houston, Miami, Dallas-Fort Worth, Montreal). A loop sends weather data from the OpenWeatherMap API to a Kinesis stream every 60 seconds. The code is annotated with comments explaining its purpose. The interface has a dark theme with light-colored text and standard Linux-style keyboard shortcuts at the bottom.

```

GNU nano 8.3
weather_stream-project-3.py
Modified

# Create a Kinesis client
kinesis = boto3.client('kinesis', region_name='us-east-1')

# API key and list of cities
API_KEY = "f2ffdd03d1f74ac5b09fa933e24e8876"
CITIES = ["New York", "Los Angeles", "Mexico City", "Toronto", "Chicago", "Houston", "Miami", "Dallas-Fort Worth", "Montreal"]

# Start an infinite loop to send weather data every 60 seconds
while True:
    for city in CITIES:
        url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}"
        response = requests.get(url.json())

        # Send data to Kinesis stream
        kinesis.put_record(
            StreamName="weather_stream",
            Data=json.dumps(response),
            PartitionKey=city
        )
        print(response)
        time.sleep(60)
    time.sleep(60)

```

Script:

```

import boto3

import json

import time

import requests


# Create a Kinesis client

kinesis = boto3.client("kinesis", region_name="us-east-1")

```

```

# API key and list of cities

API_KEY = "f2ffdd03d1f74ac5b09fa933e24e8876"

CITIES = [
    "New York", "Los Angeles", "Mexico City",
    "Toronto", "Chicago", "Houston",
    "Miami", "Dallas-Fort Worth", "Montreal"
]

```

```

# Infinite loop to send weather data every 60 seconds

while True:

    for city in CITIES:

        url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}"

        response = requests.get(url).json()

        # Send data to Kinesis stream

        kinesis.put_record(
            StreamName="weather-stream",
            Data=json.dumps(response),
            PartitionKey=city
        )

        print(f"Sent weather data for {city}")

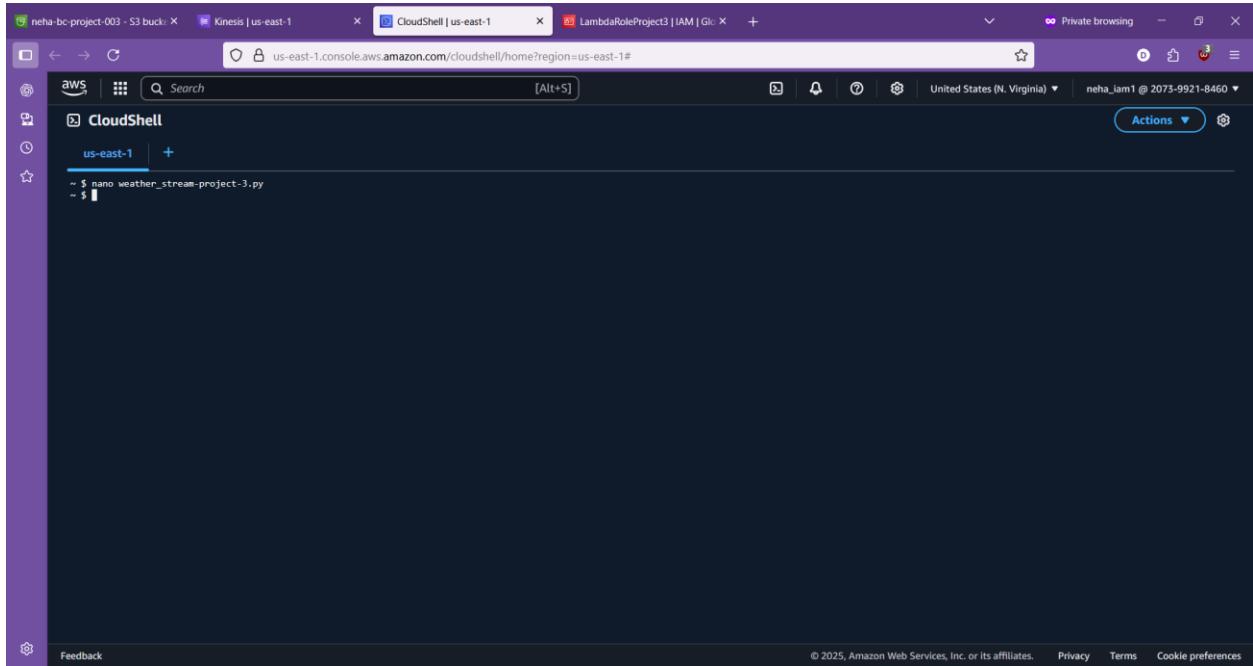
```

```

# Wait 60 seconds before the next batch

time.sleep(60)

```



A screenshot of the AWS CloudShell interface in a browser window. The title bar shows tabs for 'neha-bc-project-003 - S3 buck...', 'Kinesis | us-east-1', 'CloudShell | us-east-1' (which is active), and 'LambdaRoleProject3 | IAM | Gl...'. The main content area displays a terminal session:

```
~ $ nano weather_stream-project-3.py
~ $ chmod +x weather_stream-project-3.py
~ $ ./weather_stream-project-3.py
-bash: Python3: command not found
~ $ python3 weather_stream-project-3.py
  File "/home/cloudshell-user/weather_stream-project-3.py", line 26
    time.sleep(60)
IndentationError: unexpected indent
~ $ nano weather_stream-project-3.py
~ $ chmod +x weather_stream-project-3.py
~ $ python3 weather_stream-project-3.py
```

A screenshot of the AWS CloudShell interface in a browser window. The title bar shows tabs for 'neha-bc-project-003 - S3 buck...', 'Kinesis | us-east-1', 'CloudShell | us-east-1' (which is active), and 'LambdaRoleProject3 | IAM | Gl...'. The main content area displays a terminal session:

```
~ $ nano weather_stream-project-3.py
~ $ chmod +x weather_stream-project-3.py
~ $ ./weather_stream-project-3.py
-bash: Python3: command not found
~ $ python3 weather_stream-project-3.py
  File "/home/cloudshell-user/weather_stream-project-3.py", line 26
    time.sleep(60)
IndentationError: unexpected indent
~ $ nano weather_stream-project-3.py
~ $ chmod +x weather_stream-project-3.py
~ $ python3 weather_stream-project-3.py
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/site-packages/botocore/client.py", line 314, in _make_api_call
    return self._make_api_call(operation_name, kwargs)
  File "/usr/local/lib/python3.9/site-packages/botocore/context.py", line 124, in wrapper
    return func(*args, **kwargs)
  File "/usr/local/lib/python3.9/site-packages/botocore/client.py", line 1031, in _make_api_call
    raise error_class(parsed_response, operation_name)
botocore.errorfactory.ResourceNotFoundException: An error occurred (ResourceNotFoundException) when calling the PutRecord operation: Stream weather_stream under account 207399218460 not found.
```

```

neha-bc-project-003 - S3 buck: X Kinesis | us-east-1 X CloudShell | us-east-1 X neha.jam1 | IAM | Global X + Private browsing X
aws CloudShell us-east-1 Search [Alt+S] Actions United States (N. Virginia) neha_jam1 @ 2073-9921-8460
CloudShell us-east-1 +
e pyenv exec aws kinesis --region=us-east-1#
Traceback (most recent call last):
  File "/home/cloudshell-user/weather_stream-project-3.py", line 24, in <module>
    kinesis.put_record()
  File "/usr/local/lib/python3.9/site-packages/botocore/client.py", line 570, in _api_call
    return self._make_api_call(operation_name, kwargs)
  File "/usr/local/lib/python3.9/site-packages/botocore/context.py", line 124, in wrapper
    return func(*args, **kwargs)
  File "/usr/local/lib/python3.9/site-packages/botocore/client.py", line 1031, in _make_api_call
    raise error_class(parsed_response, operation_name)
botocore.errorfactory.ResourceNotFoundException: An error occurred (ResourceNotFoundException) when calling the PutRecord operation: Stream weather_stream under account 207399218460 not found.
~ $ aws kinesis list-streams --region=us-east-1
{
  "StreamNames": [
    "weather-stream"
  ],
  "StreamSummaries": [
    {
      "StreamName": "weather-stream",
      "ARN": "arn:aws:kinesis:us-east-1:207399218460:stream/weather-stream",
      "StreamStatus": "ACTIVE",
      "StreamLastModified": "2025-07-22T02:09:55+00:00",
      "StreamMode": "PROVISIONED"
    },
    "StreamCreationTimestamp": "2025-07-22T02:09:55+00:00"
  ]
}
~ $ nano weather_stream-project-3.py
~ $ chmod +x weather_stream-project-3.py
~ $ python3 weather_stream-project-3.py
Sent weather data for New York
Sent weather data for Los Angeles
Sent weather data for Mexico City
Sent weather data for Toronto
Sent weather data for Chicago
Sent weather data for Washington
Sent weather data for Miami
Sent weather data for Dallas-Fort Worth
Sent weather data for Montreal
[...]

```

Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Silver Layer

1. Create Lambda function to handle the weather data

AWS Lambda lets you run code without thinking about servers.

You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

Create a function

How it works

.NET Java Node.js Python Ruby Custom runtime

```

1 * exports.handler = async (event) => {
2   console.log(event);
3   return 'Hello from Lambda!';
4 };

```

Run Next: Lambda responds to events

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS Lambda 'Create function' wizard step 1: Basic information.

Basic information

Function name: weatherStreamFunction

Runtime: Python 3.13

Architecture: x86_64

Permissions: By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Author from scratch (selected) vs **Use a blueprint** vs **Container image**

Screenshot of the AWS Lambda 'Create function' wizard step 2: Additional configurations.

Change default execution role

Execution role: Use an existing role (selected). Role: LambdaRoleProject3

Existing role: LambdaRoleProject3

Additional configurations: Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Create function button

The screenshot shows the AWS Lambda Functions console. A green success message at the top states: "Successfully created the function weatherStreamFunction. You can now change its code and configuration. To invoke your function with a test event, choose 'Test'." Below this, the function name "weatherStreamFunction" is displayed. The "Function overview" section shows a diagram with a Kinesis stream trigger. The "Configuration" tab is currently selected. On the right side, there is a "Description" panel with details like "Last modified 2 seconds ago", "Function ARN arn:aws:lambda:us-east-1:207399218460:function:weatherStreamFunction", and a "Function URL". Buttons for "Export to Infrastructure Composer", "Copy ARN", and "Actions" are also present.

2. Add the following Triggers

Kinesis - to fetch the real time data from kinesis

S3 - to store the raw (Bronze layer) data and Cleaned/processed (Siler layer) data in S3

The screenshot shows the AWS Lambda Functions console under the "Configuration" tab. A message indicates that a Kinesis trigger was successfully added but is in a disabled state. The "General configuration" section is visible, showing fields for "Lambda function name", "Memory (MB)", "Timeout (seconds)", and "Execution role". A detailed description of triggers and destinations is provided on the right side of the page.

Screenshot of the AWS Lambda 'Add triggers' configuration page.

Trigger configuration

Bucket: s3/neha-bc-project-003

Event types: All object create events

Prefix - optional: e.g. images/

Suffix - optional: e.g. jpg

Triggers

A trigger is a service or resource that invokes your function. You can add triggers for existing resources only. If the resource doesn't exist yet, create it in the corresponding service console and then add the trigger to your function. When you add a trigger, the console configures the resources and permissions required to use it.

Most services invoke your function directly with an event structure that is specific per service. For some services, AWS Lambda reads service data and invokes the function with a resource called an event source mapping. An event source mapping is a Lambda resource that reads from an event source and invokes a Lambda function. For a list of services, see [Lambda event source mappings](#).

Was this content helpful?

Screenshot of the AWS Lambda 'Add triggers' configuration page, showing the 'Recursive invocation' section.

Use: s3/neha-bc-project-003

Event types: All object create events

Prefix - optional: e.g. images/

Suffix - optional: e.g. jpg

Recursive invocation: I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

Triggers

A trigger is a service or resource that invokes your function. You can add triggers for existing resources only. If the resource doesn't exist yet, create it in the corresponding service console and then add the trigger to your function. When you add a trigger, the console configures the resources and permissions required to use it.

Most services invoke your function directly with an event structure that is specific per service. For some services, AWS Lambda reads service data and invokes the function with a resource called an event source mapping. An event source mapping is a Lambda resource that reads from an event source and invokes a Lambda function. For a list of services, see [Lambda event source mappings](#).

Was this content helpful?

The screenshot shows the AWS Lambda console interface. In the center, there's a 'Function overview' section with a diagram showing 'weatherStreamFunction' triggering 'Kinesis' and pointing to 'S3'. Below this, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration' (which is selected), 'Aliases', and 'Versions'. To the right, there's a detailed sidebar for 'Function configuration' with sections for 'Description', 'Last modified', 'Function ARN', 'Function URL', and 'Layers'. A note at the top says: 'The trigger neha-bc-project-003 was successfully added to function weatherStreamFunction. The function is now receiving events from the trigger.'

Paste the lambda function code in the “code” tab and Click “deploy”

The screenshot shows the AWS Lambda function editor. On the left, there's an 'EXPLORER' panel with 'WEATHERSTREAMFUNCTION' selected, showing 'lambda_function.py'. The main area contains the Python code for the function:

```

lambda_function.py
def handle_kinesis_event(event):
    try:
        print(f" Uploaded raw file: s3://[bucket]/[bronze_prefix][filename]")
        return {"statusCode": 200, "body": f"Uploaded {filename}"}
    except Exception as e:
        print(f" Error in handle_kinesis_event: {e}")
        return {"statusCode": 500, "body": str(e)}

def handle_s3_event(event):
    """Triggered by S3 - reads Bronze JSON + cleans + saves to Silver"""
    try:
        for record in event['Records']:
            source_bucket = record['s3']['bucket']['name']
            file_key = record['s3']['object']['key']

            # Download Bronze JSON
            response = s3.get_object(Bucket=source_bucket, Key=file_key)
            file_data = response['Body'].read()
            raw_json = json.loads(file_data)

            # Flatten weather JSON
            df = pd.json_normalize(
                raw_json,
                record_path=['records'],
                meta=['file_key'],
                sep='.'
            )
            df.to_parquet(
                f"s3://[bronze_prefix][file_key].parquet",
                compression='gzip'
            )
    except Exception as e:
        print(f" Error in handle_s3_event: {e}")
        return {"statusCode": 500, "body": str(e)}

```

Below the code, there are buttons for 'Deploy (Ctrl+Shift+U)' and 'Test (Ctrl+Shift+I)'. The status bar at the bottom indicates 'Successfully updated the function weatherStreamFunction'.

Code:

```

import json

import boto3

import base64

from datetime import datetime

```

```
import pandas as pd
import io

# AWS clients
s3 = boto3.client('s3')

# bucket & prefixes
bucket = "neha-bc-project-003"
bronze_prefix = "bronze/weather_data/"
silver_prefix = "silver/weather_data/"

def lambda_handler(event, context):
    print("Received event:", json.dumps(event))

    # Detect Kinesis event
    if 'Records' in event and 'kinesis' in event['Records'][0]:
        return handle_kinesis_event(event)

    # Detect S3 event
    elif 'Records' in event and 's3' in event['Records'][0]:
        return handle_s3_event(event)

    else:
        return {
            "statusCode": 400,
            "body": "Unknown event type"
        }

def handle_kinesis_event(event):
```

```

"""Triggered by Kinesis : stores raw JSON in Bronze layer (handles multiple records)"""

try:
    # Decode all records in the batch
    records = [
        json.loads(base64.b64decode(rec['kinesis']['data']).decode('utf-8'))
        for rec in event['Records']
    ]

    now = datetime.utcnow().strftime("%Y-%m-%d-%H-%M")
    filename = f"weather_data_{now}.json"

    # Save as a JSON array
    body = json.dumps(records, indent=2)

    # Upload to Bronze layer
    s3.put_object(
        Bucket=bucket,
        Key=f"{bronze_prefix}{filename}",
        Body=body,
        ContentType='application/json'
    )

    print(f"Uploaded {len(records)} records to s3://{bucket}/{bronze_prefix}{filename}")

    return {"statusCode": 200, "body": f"Uploaded {len(records)} records"}

except Exception as e:
    print(f"Error in handle_kinesis_event: {e}")

    return {"statusCode": 500, "body": str(e)}

```

```
def handle_s3_event(event):
    """Triggered by S3 : reads Bronze JSON → cleans → saves to Silver"""

    try:
        for record in event['Records']:
            source_bucket = record['s3']['bucket']['name']
            file_key = record['s3']['object']['key']

            # Download Bronze JSON
            response = s3.get_object(Bucket=source_bucket, Key=file_key)
            file_data = response['Body'].read().decode('utf-8').strip()

            # Skip if Bronze file is empty
            if not file_data:
                print(f"Skipping empty Bronze file: {file_key}")
                continue

            # Try parsing JSON, skip if invalid
            try:
                raw_json = json.loads(file_data)
            except json.JSONDecodeError as e:
                print(f"Skipping invalid JSON in {file_key} -> {e}")
                continue

            # If it's a batch JSON array, flatten each weather object
            if isinstance(raw_json, list):
                combined_df = pd.DataFrame()
                for weather_entry in raw_json:
                    df = flatten_weather_json(weather_entry)
                    if not df.empty:
                        combined_df = pd.concat([combined_df, df], ignore_index=True)

                # Save to Silver
                df.to_csv(f"/tmp/{file_key}.csv", index=False)
```

```
combined_df = pd.concat([combined_df, df], ignore_index=True)

else:

    df = flatten_weather_json(raw_json)

    combined_df = df if not df.empty else pd.DataFrame()

# If no valid records, skip saving

if combined_df.empty:

    print(f"No valid weather data found in {file_key}. Skipping upload.")

    continue

# Convert to CSV in-memory

out_buffer = io.StringIO()

combined_df.to_csv(out_buffer, index=False)

# Generate Silver-layer filename

cleaned_filename = file_key.split('/')[-1].replace('.json', '.csv')

cleaned_key = f"{silver_prefix}{cleaned}_{cleaned_filename}"

# Upload cleaned CSV to Silver

s3.put_object(

    Bucket=bucket,

    Key=cleaned_key,

    Body=out_buffer.getvalue(),

    ContentType='text/csv'

)

print(f"Cleaned file uploaded to: s3://{bucket}/{cleaned_key}")

print(combined_df.head())
```

```
return {  
    'statusCode': 200,  
    'body': json.dumps('File cleaned and uploaded successfully!')  
}  
  
except Exception as e:  
    print(f"Error in handle_s3_event: {e}")  
    return {"statusCode": 500, "body": str(e)}  
  
def flatten_weather_json(raw_json):  
    """Helper: Flattens a single weather JSON record into a DataFrame"""  
  
    # Skip if record does not have weather data  
    if not isinstance(raw_json, dict) or 'weather' not in raw_json:  
        print("Skipping record without 'weather' key:", raw_json)  
        return pd.DataFrame()  
  
    df = pd.json_normalize(  
        raw_json,  
        record_path=['weather'],  
        meta=[  
            ['coord', 'lon'],  
            ['coord', 'lat'],  
            ['main', 'temp'],  
            ['main', 'pressure'],  
            ['main', 'humidity'],  
            ['wind', 'speed'],  
            ['sys', 'country'],  
            ['sys', 'sunrise'],
```

```
        ['sys', 'sunset'],
        'name',
        'dt'
    ],
    errors='ignore'
)

# Rename columns
df.rename(columns={
    'coord.lon': 'lon',
    'coord.lat': 'lat',
    'main': 'weather',
    'description': 'weather_description',
    'main.temp': 'temp',
    'main.pressure': 'pressure',
    'main.humidity': 'humidity',
    'wind.speed': 'wind_speed',
    'sys.country': 'country_code',
    'sys.sunrise': 'sunrise',
    'sys.sunset': 'sunset',
    'name': 'city'
}, inplace=True)

if df.empty:
    return df

# Temperature in Celsius (OpenWeather returns Kelvin)
df['temp_celsius'] = df['temp'] - 273.15
```

```
# Convert sunrise/sunset to UTC datetime
df['sunrise_utc'] = pd.to_datetime(df['sunrise'], unit='s')
df['sunset_utc'] = pd.to_datetime(df['sunset'], unit='s')

# Map only US/Canada/Mexico
df['country'] = df['country_code'].map(
    lambda x: 'United States' if x == 'US' else
    'Canada' if x == 'CA' else
    'Mexico' if x == 'MX' else x
)

# Reorder columns
df = df[['lon', 'lat', 'weather', 'weather_description',
         'temp', 'temp_celsius', 'pressure', 'humidity',
         'wind_speed', 'country', 'country_code',
         'sunrise', 'sunset', 'city', 'dt']]

return df
```

3. Check in S3 Bronze and Silver Layer folder, files are created.

The screenshot shows the AWS S3 console with the path `Amazon S3 > Buckets > neha-bc-project-003 > bronze/ > weather_data/`. The 'Objects' tab is selected, displaying three JSON files: `weather_data_2025-07-22-03-22.json`, `weather_data_2025-07-22-03-27.json`, and `weather_data_2025-07-22-03-28.json`. The objects are listed in a table with columns for Name, Type, Last modified, Size, and Storage class.

Name	Type	Last modified	Size	Storage class
<code>weather_data_2025-07-22-03-22.json</code>	json	July 21, 2025, 23:22:50 (UTC-04:00)	591.0 B	Standard
<code>weather_data_2025-07-22-03-27.json</code>	json	July 21, 2025, 23:27:28 (UTC-04:00)	6.7 KB	Standard
<code>weather_data_2025-07-22-03-28.json</code>	json	July 21, 2025, 23:28:26 (UTC-04:00)	6.7 KB	Standard

The screenshot shows the AWS S3 console with the path `Amazon S3 > Buckets > neha-bc-project-003 > silver/ > weather_data/`. The 'Objects' tab is selected, displaying four CSV files: `cleaned_weather_data_2025-07-22-03-31.csv`, `cleaned_weather_data_2025-07-22-03-32.csv`, `cleaned_weather_data_2025-07-22-03-33.csv`, and `cleaned_weather_data_2025-07-22-03-34.csv`. The objects are listed in a table with columns for Name, Type, Last modified, Size, and Storage class.

Name	Type	Last modified	Size	Storage class
<code>cleaned_weather_data_2025-07-22-03-31.csv</code>	csv	July 21, 2025, 23:31:33 (UTC-04:00)	1.2 KB	Standard
<code>cleaned_weather_data_2025-07-22-03-32.csv</code>	csv	July 21, 2025, 23:32:28 (UTC-04:00)	1.2 KB	Standard
<code>cleaned_weather_data_2025-07-22-03-33.csv</code>	csv	July 21, 2025, 23:33:29 (UTC-04:00)	1.2 KB	Standard
<code>cleaned_weather_data_2025-07-22-03-34.csv</code>	csv	July 21, 2025, 23:34:30 (UTC-04:00)	1.2 KB	Standard

4. Connect to Redshift database

IAM role

The screenshot shows the AWS IAM Roles page. On the left, there's a sidebar with navigation links like Dashboard, Access management, Access reports, and CloudShell. The main content area displays a table of managed policies attached to the role. The table has columns for Policy name, Type, and Attached entities. Policies listed include AdministratorAccess, AmazonRedshiftAllCommandsFullAccess, AmazonRedshiftFullAccess, AmazonS3FullAccess, AWSGlueConsoleFullAccess, AWSGlueServiceRole, and SecretsManagerReadWrite.

Policy name	Type	Attached entities
AdministratorAccess	AWS managed - job function	3
AmazonRedshiftAllCommandsFullAccess	AWS managed	1
AmazonRedshiftFullAccess	AWS managed	1
AmazonS3FullAccess	AWS managed	3
AWSGlueConsoleFullAccess	AWS managed	1
AWSGlueServiceRole	AWS managed	1
SecretsManagerReadWrite	AWS managed	1

5. Create VPC

The screenshot shows the AWS VPC Details page for a VPC with ID vpc-01482044d67551604. The left sidebar includes links for EC2 Global View, Virtual private cloud (Your VPCs, Subnets, Route tables, Internet gateways, Egress-only internet gateways, Carrier gateways, DHCP option sets, Elastic IPs, Managed prefix lists, NAT gateways, Peering connections, Route servers), Security (Network ACLs, Security groups), and CloudShell/Feedback. The main content area displays the VPC details, including its ID, state (Available), and various configuration settings like Block Public Access (Off), DHCP option set (dopt-0d59d505885ee4a1c), and IPv4 CIDR (10.0.0.0/16). Below the details, there are tabs for Resource map, CIDRs, Flow logs, Tags, and Integrations. The Resource map section shows the VPC, Subnets (6), Route tables (6), and Network interface (igw) components.

The screenshot shows the AWS VPC Resource Map interface. On the left, there's a sidebar with 'Virtual private cloud' and 'Security' sections. The main area has tabs for 'Resource map', 'CIDRs', 'Flow logs', 'Tags', and 'Integrations'. Under 'Resource map', there are three main sections: 'Subnets (6)', 'Route tables (6)', and 'Network interface connections (0)'. The 'Subnets' section lists subnets grouped by AZ: 'us-east-1a' (subnet-public1-us-east-1a, subnet-private3-us-east-1a, subnet-private1-us-east-1a) and 'us-east-1b' (subnet-public2-us-east-1b, subnet-private2-us-east-1b, subnet-private4-us-east-1b). The 'Route tables' section lists route tables: rtb-private1-us-east-1a, rtb-private4-us-east-1b, rtb-private2-us-east-1b, rtb-0e610da3bd4ef3cc8, rtb-public, and rtb-private3-us-east-1a. A network interface connection 'igw' is shown connecting to 'vpce-s3'.

6. Configure Security group

The screenshot shows the AWS Security Groups page. The sidebar includes 'Virtual private cloud' and 'Security' sections. The main area displays 'Security Groups (1/1)' with a table showing one entry: 'sg-05326575fad7c8a87' with 'Name' and 'Owner' both set to '-'. An 'Actions' dropdown menu is open over this entry, showing options like 'Edit inbound rules', 'Edit outbound rules', 'Manage tags', etc. Below the table, a detailed view for 'sg-05326575fad7c8a87 - default' is shown with tabs for 'Details', 'Inbound rules', 'Outbound rules', 'Sharing - new', 'VPC associations - new', and 'Tags'. The 'Details' tab shows the security group name is 'default', ID is 'sg-05326575fad7c8a87', owner is '207399218460', inbound rules count is 1, and outbound rules count is 1. The 'Description' field is 'default VPC security group'. The 'VPC ID' is 'vpc-01482044d67551604'.

The screenshot shows the AWS VPC Security Groups console. The URL is <https://us-east-1.console.aws.amazon.com/vpcconsole/home?region=us-east-1#SecurityGroup:groupId=sg-05326575fad7c8a87>. The page title is "sg-05326575fad7c8a87 - default". The left sidebar shows "VPC dashboard", "Virtual private cloud" (Your VPCs, Subnets, Route tables, Internet gateways, Egress-only internet gateways, Carrier gateways, DHCP option sets, Elastic IPs, Managed prefix lists, NAT gateways, Peering connections, Route servers), and "Security" (Network ACLs, Security groups). The main content area displays the "Details" for the security group "sg-05326575fad7c8a87". It includes fields for Security group name (default), Security group ID (sg-05326575fad7c8a87), Description (default VPC security group), Owner (207399218460), Inbound rules count (2 Permission entries), and Outbound rules count (1 Permission entry). Below this, there are tabs for "Inbound rules" (selected), "Outbound rules", "Sharing - new", "VPC associations - new", and "Tags". The "Inbound rules" section shows two entries:

Name	Security group rule ID	Type	Protocol	Port range
sgr-07a3fa36cb5b5e132	-	All traffic	All	All
sgr-0cebcb1e3cab922a4a	-	Redshift	TCP	5439

7. Create Redshift workgroup configuration

The screenshot shows the Amazon Redshift Serverless dashboard. The URL is <https://us-east-1.console.aws.amazon.com/redshiftv2/2/home?region=us-east-1#/serverless-dashboard>. The page title is "Amazon Redshift Serverless". The left sidebar shows "Serverless dashboard" (info) and "Namespaces / Workgroups" (info). The main content area includes sections for "Namespace overview" (info), "Namespaces / Workgroups" (info), and "Queries metrics". The "Namespace overview" section shows 0 total snapshots, 0 data shares in my account, 0 data shares requiring authorization, 0 data shares from other accounts, and 0 data shares requiring association. The "Namespaces / Workgroups" section shows 0 namespaces or workgroups created. The "Total compute usage - new" section shows 0 consumed RPU hours. A note states: "Retrieving a workgroup's total compute usage uses the workgroup's compute capacity, and might affect performance." The bottom of the page includes "CloudShell" and "Feedback" buttons.

The screenshot shows the 'Create workgroup' step in the Amazon Redshift Serverless console. The page title is 'Create workgroup'. On the left, there's a sidebar with navigation steps: Step 1 (Create workgroup), Step 2 (Choose namespace), and Step 3 (Review and create). The main content area is titled 'Workgroup' with a sub-section 'Performance and cost controls'. In the 'Workgroup' section, the 'Workgroup name' field contains 'weather-stream-wg'. Below it, a note says: 'The name must be from 3-64 characters. Valid characters are a-z (lowercase only), 0-9 (numbers), and - (hyphen)'. In the 'Performance and cost controls' section, the 'Base capacity' radio button is selected, with a note: 'Set the base capacity in Redshift processing units (RPUs) that Amazon Redshift can use to run queries. Alternatively, set price-performance target to optimize resources. Amazon Redshift uses AI-driven scaling and optimization to automatically adjust your resources when running queries.' At the bottom right of the page, there are links for 'Privacy', 'Terms', and 'Cookie preferences'.

The screenshot shows the 'Virtual private cloud (VPC)' configuration step in the Amazon Redshift Serverless console. The page title is 'Create serverless workgroup'. The main content area includes sections for 'Virtual private cloud (VPC)', 'VPC security groups', 'Subnet', and 'SSL'. In the 'VPC' section, a note says: 'Your resources can communicate only over the IPv4 addressing protocol.' In the 'VPC security groups' section, a note says: 'Your resources can communicate over IPv4, IPv6, or both. For dual-stack mode, associate an IPv6 CIDR block with a subnet in your VPC using Amazon EC2.' The 'Subnet' section shows three subnet IDs: 'subnet-05a0376ca31b04cd4', 'subnet-05414590a75394baf', and 'subnet-010a017d3d25d31ff'. A note at the bottom of this section says: 'SSL is now enforced by default for new serverless workgroups.' In the 'SSL' section, a note says: 'SSL is enabled by default to help you get started with your workgroup. You can change the SSL settings after you finish creating your workgroup.' The 'Enable' radio button is selected, with a note: 'Enable SSL for enhanced security'. At the bottom right of the page, there are links for 'Privacy', 'Terms', and 'Cookie preferences'.

The screenshot shows the AWS Redshift console interface for creating a new serverless workgroup. In the center, there is a list of subnet IDs under the heading "Choose three or more subnet IDs". Three subnets are selected: "subnet-private2-us-east-1b", "subnet-private3-us-east-1a", and "subnet-private1-us-east-1a". Below this list, there is a note: "SSL is now enforced by default for new serverless workgroups." At the bottom, there is an "SSL" section with a "Enable" button.

The screenshot shows the "Create workgroup" step of the AWS Redshift Serverless wizard. The current step is "Step 2 Choose namespace". The "Namespace" section is active, showing the option "Create a new namespace" selected. A text input field contains the namespace name "weather-stream-ns". The "Database name and password" section is partially visible below it.

Database name and password

Database name
The name of the first database in the Amazon Redshift Serverless environment.

The name must be 1-64 alphanumeric characters (lowercase only), and it can't be a reserved word.

Admin user credentials
IAM credentials provided as your default admin user credentials. To add a new admin username and password, customize admin user credentials.
 Customize admin user credentials
To use the default IAM credentials, clear this option.

Admin user name
The administrator's user name for the first database.

The name must be 1-128 alphanumeric characters, and it can't be a reserved word.

Admin password
Select an option to manage your admin password.
 Manage admin credentials in AWS Secrets Manager Info
AWS manages a KMS key that encrypts your data.
 Generate a password
Amazon Redshift generates an admin password.
 Manually add the admin password
Manually enter the admin password.
Admin user password
The password of the admin user.

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

able to run these SQL commands without an IAM role attached to your namespace.

The IAM role [AmazonRedshift-CommandsAccessRole-20250722T000821](#) was successfully created and set as the default.

Associated IAM roles (1)

Create IAM role

Create, associate, or remove an IAM role. You can associate up to 150 IAM roles. You can also choose an IAM role and set it as the default.

	IAM roles	Status	Role type
<input type="checkbox"/>	AmazonRedshift-CommandsAccessRole-20250722T000821	Not applied	Default

Encryption and security

Your data is encrypted by default with an AWS owned key. To choose a different key, customize your encryption settings.

console.aws.amazon.com/iam/home?region=us-east-1#roles/AmazonRedshift-CommandsAccessRole-20250722T000821

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS Redshift Serverless console showing the creation of a new workgroup named "weather-stream-wg".

Database name and password

Database name	dev
Admin user credentials	admin

Permissions

Default IAM role arn:aws:iam::207399218460:role/service-role/ AmazonRedshift- CommandsAccessRole-20250722T000821	IAM roles arn:aws:iam::207399218460:role/service-role/ AmazonRedshift- CommandsAccessRole-20250722T000821
---	--

Encryption and security

AWS KMS encryption AWS owned key	Audit logging Off
-------------------------------------	----------------------

Cancel Previous Create

Screenshot of the AWS Redshift Serverless console showing the creation progress of a new workgroup.

Creation in progress
Amazon Redshift Serverless is creating weather-stream-wg.

Serverless dashboard

Namespace overview

Total snapshots	Datashares in my account	Datashares requiring authorization	Datashares from other accounts	Datashares requiring association
0	0	0	0	0

Namespaces / Workgroups

Namespace	Status	Workgroup	Status	Average query duration	Average number
weather-stream-ns	Available	weather-stream-wg	Creating	--	--

Total compute usage - new

Choose a workgroup Last hour To visualize the costs of your total compute usage, go to AWS Cost Explorer

Total consumed RPU hours --

You have successfully created weather-stream-wg and attached it to weather-stream-ns

Amazon Redshift Serverless

Serverless dashboard [Info](#)

Namespace overview [Info](#)

Namespace data from your account

Total snapshots	Dashboards in my account	Dashboards requiring authorization	Dashboards from other accounts	Dashboards requiring association
0	0	0	0	0

Namespaces / Workgroups [Info](#)

Namespace	Status	Workgroup	Status	Average query duration	Average number
weather-stream-ns	Available	weather-stream-wg	Available	--	--

Total compute usage - new [Info](#)

Choose a workgroup [▼](#) Last hour [▼](#)

To visualize the costs of your total compute usage, go to [AWS Cost Explorer](#)

Total consumed RPU hours
--

8. Query Data

```
CREATE TABLE weather_data_query.weather_data (lon DOUBLE PRECISION, lat DOUBLE PRECISION, weather VARCHAR(100), weather_description VARCHAR(200), temp DOUBLE PRECISION, temp_celsius DOUBLE PRECISION, pressure DOUBLE PRECISION, humidity INT, wind_speed DOUBLE PRECISION, country VARCHAR(100), country_code VARCHAR(100), sunrise BIGINT, sunset BIGINT, city VARCHAR(100), dt BIGINT);
```

Redshift query editor v2

Serverless: weather-stream-ns [?](#)

native databases (2)

- dev
- public
- weather_data...
- sample_data_dev
- external databases (1)

```
Row 17, Col 1, Chr 488
CREATE TABLE weather_data_query.weather_data (
    lon DOUBLE PRECISION,
    lat DOUBLE PRECISION,
    weather VARCHAR(100),
    weather_description VARCHAR(200),
    temp DOUBLE PRECISION,
    temp_celsius DOUBLE PRECISION,
    pressure DOUBLE PRECISION,
    humidity INT,
    wind_speed DOUBLE PRECISION,
    country VARCHAR(100),
    country_code VARCHAR(100),
    sunrise BIGINT, sunset BIGINT,
    city VARCHAR(100), dt BIGINT);
```

Summary

Returned rows: 0
Elapsed time: 372ms
Result set query:

```
/* RDEV2-wrE2D5MwTH */
CREATE TABLE weather_data_query.weather_data (
    lon DOUBLE PRECISION,
    lat DOUBLE PRECISION,
    weather VARCHAR(100),
    weather_description VARCHAR(200),
    temp DOUBLE PRECISION,
    temp_celsius DOUBLE PRECISION,
```

The screenshot shows the AWS Redshift query editor interface. On the left, the sidebar includes 'Editor', 'Queries', 'Notebooks', 'Charts', 'History', 'Scheduled queries', and navigation icons. The main area displays a query for a database named 'weather-stream-wg'. The query is:

```

1 lat DOUBLE PRECISION,
2 weather VARCHAR(100),
3 weather_description VARCHAR(200),
4 temp DOUBLE PRECISION,
5 temp_celsius DOUBLE PRECISION,
6 pressure DOUBLE PRECISION,
7 humidity INT,
8 wind_speed DOUBLE PRECISION,
9 country VARCHAR(100),
10 country_code VARCHAR(100),
11 sunrise BIGINT, sunset BIGINT,
12
13
14
15

```

The results section shows 'Result 1' with a summary: 'Returned rows: 0', 'Elapsed time: 372ms', and 'Result set query'.

Gold Layer

1. Create AWS Glue Connection

The screenshot shows the AWS Glue Connectors page. The left sidebar lists 'AWS Glue' (Getting started, ETL jobs, Data Catalog tables, Data connections, Data Catalog), 'Data Catalog', 'Data Integration and ETL', and 'Legacy pages'. The main area has sections for 'Connectors (0)' and 'Connections (0)'. Both sections show a message: 'No connectors to display. You can create a custom connector or get a Marketplace connector.' There are buttons for 'Create custom connector' and 'Create connection'.

The screenshot shows the AWS Glue 'Create connection' wizard. The left sidebar lists various AWS services like ETL jobs, Data Catalog, and Data Integration and ETL. The main area is titled 'Choose data source' with a step-by-step guide: Step 1 (selected), Step 2, Step 3, and Step 4. A search bar at the top right shows 'jdbc'. Below it, a 'JDBC' option is highlighted with a blue border, showing a database icon and the text 'Connect to a data system using JDBC.' There are 'Learn more' and 'Next' buttons at the bottom right.

The screenshot shows the Amazon Redshift Serverless 'Workgroup configuration' page for 'weather-stream-wg'. At the top, a message says 'This workgroup is associated with a namespace' and provides a link to 'weather-stream-ns'. The main section is titled 'weather-stream-wg' with an 'info' link. Under 'General information', there are several details:

Workgroup	Date created	Endpoint
weather-stream-wg	July 22, 2025, 00:09 (UTC-04:00)	weather-stream-wg.207399218460.us-east-1.redshift-serverless.amazonaws.com:5439/dev
Namespace	Status	JDBC URL
weather-stream-ns	Available	jdbc:redshift://weather-stream-wg.207399218460.us-east-1.redshift-serverless.amazonaws.com:5439/dev
Workgroup ARN	Base capacity	copy jdbc:redshift://weather-stream-wg.207399218460.us-east-1.redshift-serverless.amazonaws.com:5439/dev
arn:aws:redshift-serverless:us-east-1:207399218460:workgroup/cec534f7-47ae-4d03-bfe7-32ea1c5dc94	128 RPUs	Driver=(Amazon Redshift (x64)); Server=weather-stream-wg.207399218460.us-east-1.redshift-serverless.amazonaws.com; Database=dev
Workgroup version	Custom domain name	
1.0.117458	-	
Patch version	Track - new	

At the bottom, there are 'Actions' and 'Query data' buttons.

Screenshot of the AWS Glue 'Create connection' wizard Step 2: Configure connection.

The left sidebar shows the AWS Glue navigation menu with 'Data connections' selected. The main panel displays the 'Configure connection' step, which is the second step in the process.

Connection details

JDBC URL: stream-wg.207399218460.us-east-1.redshift-serverless.amazonaws.com:5439/dev

JDBC Driver Class name - optional: (empty input field)

JDBC Driver S3 Path - optional: s3://bucket/prefix/object

Credential type: Username and password (selected radio button)

Username: admin

Password: (redacted)

Step 1: Choose data source

Step 3: Set properties

Step 4: Review and create

Screenshot of the AWS Glue 'Create connection' wizard Step 3: Set properties.

The left sidebar shows the AWS Glue navigation menu with 'Data connections' selected. The main panel displays the 'Set properties' step, which is the third step in the process.

Connection Properties

Name: Jdbc connection

Require SSL connection: (unchecked checkbox)

Description - optional: (empty input field)

Tags: (empty input field)

Step 1: Choose data source

Step 2: Configure connection

Step 4: Review and create

Cancel Previous Next

The screenshot shows the AWS Glue Create connection wizard. On the left sidebar, under the 'Data connections' section, 'Zero-ETL integrations' is highlighted. The main panel displays 'Step 1: Choose data source' with a sub-section 'Data source' showing a JDBC icon and the name 'JDBC'. A vertical navigation bar on the left lists steps: Step 1 (Choose data source), Step 2 (Configure connection), Step 3 (Set properties), and Step 4 (Review and create). Step 4 is currently selected.

The screenshot shows the 'Step 2: Configure connection' screen. It includes a 'Connection details' section with a JDBC URL: 'jdbc:redshift://weather-stream-wg.207399218460.us-east-1.redshift-serverless.amazonaws.com:5439/dev'. To the right, it specifies a 'Secret type' as 'Username and password'. Below this is a 'Step 3: Set properties' section with a 'Connection properties' table showing a single entry: Name 'Jdbc connection' and Description '-'. At the bottom, there is a 'Tags' section with a table for adding key-value pairs, currently showing 'No tags'.

Sneha-bc-pro Jdbc conn: X SecurityGroup Redshift - Ser Editor | Redshift Kinesis | us-e weatherStream CloudWatch CloudShell | Roles | IAM | + Private browsing

aws Search [Alt+S] United States (N. Virginia) neha_iam1 @ 2073-9921-8460

AWS Glue > Connectors > Jdbc connection

Jdbc connection

Connection details Info

Connector type: JDBC

Driver class name: -

Username: admin

Subnet: -

Description: -

Last modified: 2025-07-22 00:22:50.784000

Connection URL: jdbc:redshift://weather-stream-wg.207399218460.us-east-1.redshift-serverless.amazonaws.com:5439/dev

Driver path: -

Require SSL connection: -

Security groups: -

Created on: 2025-07-22 00:22:50.784000

Class name: -

Tags (0) Manage tags

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

"Jdbc connection" connection successfully created. To begin using your connection you must create a job.

Edit Delete Create job

Sneha-bc-pro Connector X SecurityGroup Redshift - Ser Editor | Redshift Kinesis | us-e weatherStream CloudWatch CloudShell | Roles | IAM | + Private browsing

aws Search [Alt+S] United States (N. Virginia) neha_iam1 @ 2073-9921-8460

AWS Glue > Connectors

Marketplace connectors

Subscribe to connectors from AWS partners to expand your data sources. Go to AWS Marketplace

Custom connectors

Provide your own connector to expand your data sources. Creating custom connectors Create custom connector

Connectors (0) Info

You can manage your connectors or use them to create connections. Filter connections by property

Name	Status	Type	Last modified
No connectors to display. You can create a custom connector or get a Marketplace connector. Create custom connector			

Actions ▾

Connections (1) Info

You can manage your connections or use a connection in a job. Filter connections by property

Name	Status	Type	Last modified	Actions	Create connection	Create job	
Jdbc connection	Ready	JDBC	Jul 22, 2025	View details	Delete	Edit	Test connection

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the AWS Glue Connectors page. On the left, there's a sidebar with options like Getting started, ETL jobs, Data Catalog, Data Integration and ETL, and Legacy pages. The main area has two sections: Marketplace connectors (with a 'Go to AWS Marketplace' button) and Custom connectors (with a 'Create custom connector' button). A central modal window titled 'Test Connection' is open, showing an IAM role dropdown set to 'glueRoleProject3' and a 'View' button. Below it, a note says 'Ensure that this role has permission to access your data store.' with a 'Create IAM role' link. At the bottom of the modal are 'Cancel' and 'Confirm' buttons. In the background, there's a 'Connections (1)' table with one entry: 'Jdbc connection' (Status: Ready, Type: JDBC, Last modified: Jul 22, 2025, Version: 1). The status bar at the bottom right indicates '© 2025, Amazon Web Services, Inc. or its affiliates.'

This screenshot is nearly identical to the first one, showing the AWS Glue Connectors page. The 'Test Connection' modal is open again, but this time it displays an error message: 'InvalidInputException: Unable to resolve any valid connection'. There is also a 'Troubleshoot' link next to the error message. The background 'Connections (1)' table and the status bar at the bottom right are identical to the first screenshot.

The screenshot shows the AWS Glue Connectors page. On the left, there's a sidebar with options like 'AWS Glue', 'Data Catalog', 'Data Integration and ETL', and 'Legacy pages'. The main area has two sections: 'Marketplace connectors' (with a 'Go to AWS Marketplace' button) and 'Custom connectors' (with a 'Create custom connector' button). A central modal window titled 'Test Connection' displays a green success message: 'Successfully connected to the data store with connection Jdbc connection.' Below this, there's a 'Connections (1)' section with a table showing one entry: 'Jdbc connection' (Status: Ready, Type: JDBC, Last modified: Jul 22, 2025, Version: 1).

The screenshot shows the AWS CloudWatch Log groups page. The sidebar includes 'CloudWatch' with 'Logs' selected, 'Metrics', 'Application Signals (APM)', 'Network Monitoring', and 'Insights'. The main content shows a table of log events for the path '/aws-glue/testconnection/output/Jdbc_connection'. The events are timestamped from 2025-07-22T05:00:57.036Z to 2025-07-22T05:00:58.037Z. Key logs include:

- Driver properties = com.amazonaws.glue.jobexecutor.commands.jdbc.JdbcDriverProperties@2de5444f
- Driver com.amazon.redshift.jdbc41.Driver@3fcdf, Version 1.2 ---
- Starting connector. driver com.amazon.redshift.jdbc41.Driver@3fcdf
- Attempting to connect with SSL host matching: jdbc:redshift://weather-stream-wg.207399218460.us-east-1.redshift-serverless.amazonaws.com:5439/...
- JDBC Connection successful to jdbc:redshift://weather-stream-wg.207399218460.us-east-1.redshift-serverless.amazonaws.com:5439/dev

At the bottom, it says 'No newer events at this moment. Auto retry paused. Resume'.

Create Glue Visual ETL job

glueJobVisual

Data source properties - S3

Name: Amazon S3

S3 source type: S3 location

S3 URL: data_2025-07-22-03-44.csv

Recursive:

Data format: CSV

Delimiter: Comma (,)

Escape character - optional: \

glueJobVisual

Last modified on 2025-07-22, 1:28:30 a.m.

Job runs (1/1)

Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity (DPUs)	Worker type	Glue version
Succeeded	0	07/22/2025 01:28:34	07/22/2025 01:30:41	1 m 49 s	2 DPUs	G.1X	5.0

Run details

Job name	Start time (Local)	Glue version	Last modified on (Local)
glueJobVisual	07/22/2025 01:28:34	5.0	07/22/2025 01:30:41
Id	End time (Local)	Worker type	Log group name
jr_e1864ecdde85b29eb78109cc70643bb7ff8c7786e1707/22/2025 01:30:41 9309ebae1881764006efd		G.1X	/aws-glue/jobs
Run status	Start up-time	Max capacity	Number of workers
Succeeded	17 seconds	2 DPUs	2
Retry attempt number	Execution time	Execution class	Timeout

2. Check in Redshift table

Editor | Redshift query editor v2

aws Search

You successfully updated Serverless: weather-stream-wg. To analyze data, connect to Serverless: weather-stream-wg.

Redshift query editor v2

Editor Queries Notebooks Charts History Scheduled queries CloudShell Feedback

Serverless: weather-stream-wg (native databases (2), external databases (1))

```

2 CREATE TABLE weather_data_query.weather_data (
3     lon DOUBLE PRECISION,
4     lat DOUBLE PRECISION,
5     weather VARCHAR(100),
6     weather_description VARCHAR(200),
7     temp DOUBLE PRECISION,
8     temp_celsius DOUBLE PRECISION,
9     pressure DOUBLE PRECISION,
10    humidity INT,
11    wind_speed DOUBLE PRECISION,
12    country VARCHAR(100),
13    country_code VARCHAR(100),
14    sunrise BIGINT,sunset BIGINT,
15    city VARCHAR(100),
16    dt BIGINT);
17
18
19 SELECT * from weather_data_query.weather_data;

```

Result 1 (9)

	lon	lat	weather	weather_description	temp	temp_celsius
1	-74.006	40.7143	Clear	clear sky	297.38	24.23C
2	-118.2437	34.0522	Clear	clear sky	292.42	19.27C
3	-99.1277	19.4285	Thunderstorm	thunderstorm	290.9	17.75
4	60.1977	40.7105	Rain	rain	290.0	17.70

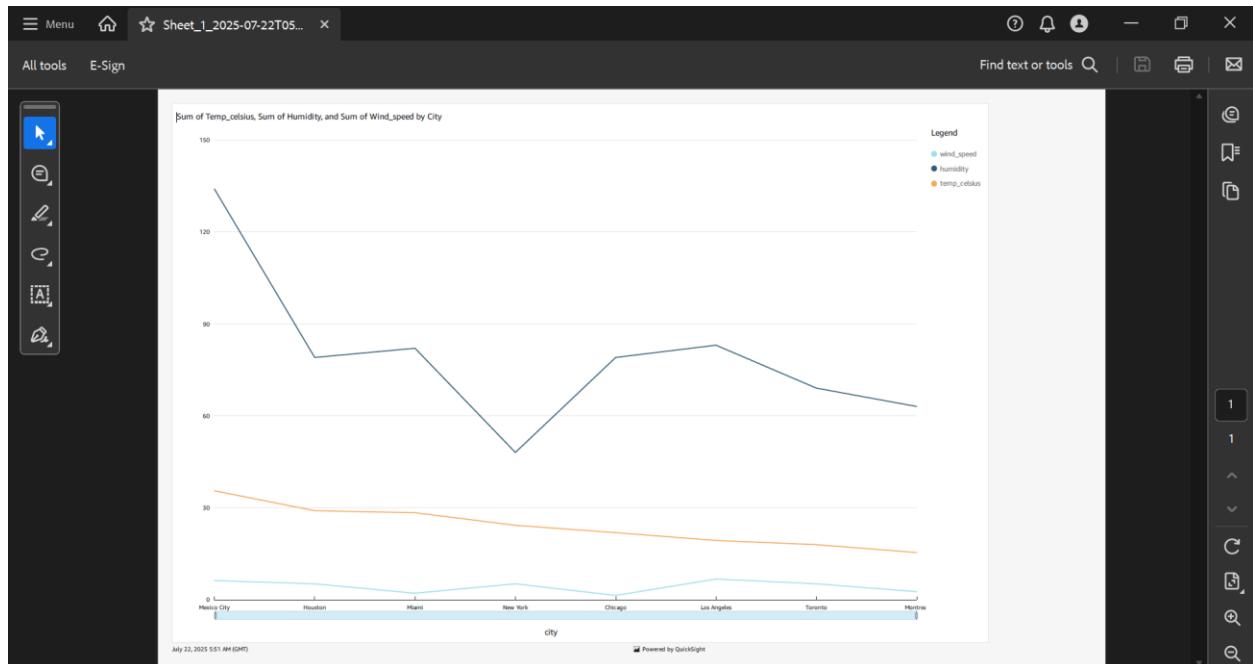
Query ID 5150 Elapsed time: 3193 ms Total rows: 9

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Visualization



Sheet_1_2025-07-22T
05_51_41.pdf



Advance analytics

SNS topics:

The screenshot shows the AWS SNS console with a successful subscription message. The message states: "Subscription to bc003 created successfully. The ARN of the subscription is arn:aws:sns:us-east-1:207399218460:bc003:2d1c1b57-89e2-4b3a-a0b6-6d581ce07717." Below this, there is a detailed view of the subscription "Subscription: 2d1c1b57-89e2-4b3a-a0b6-6d581ce07717". The details include:

- ARN:** arn:aws:sns:us-east-1:207399218460:bc003:2d1c1b57-89e2-4b3a-a0b6-6d581ce07717
- Endpoint:** ehaan.scruggs@mailmagnet.co
- Topic:** bc003
- Subscription Principal:** arn:aws:iam::207399218460:user/neha_jam1
- Status:** Pending confirmation
- Protocol:** EMAIL

The screenshot shows the AWS SNS confirmation page with the message: "Subscription confirmed! You have successfully subscribed. Your subscription's id is: arn:aws:sns:us-east-1:207399218460:bc003:2d1c1b57-89e2-4b3a-a0b6-6d581ce07717. If it was not your intention to subscribe, [click here to unsubscribe](#)." The page also includes the AWS logo and the Simple Notification Service header.

Databricks:

<https://community.cloud.databricks.com/editor/notebooks/3950310733661324?o=1049562611814646#command/3950310733661325>

