Lab 13: Building Your First Real-Time AWS Pipeline with Managed Flink

**Goal:** This lab provides a beginner-friendly, hands-on introduction to building a serverless, batch data pipeline on AWS. You will learn how to process data with a Python script in Amazon Managed Service for Apache Flink and store the results in Amazon S3.

### **INDEX**

Purpose of this Lab

Prerequisites

Part 1: Setting Up Your AWS Infrastructure

Part 2: Developing the Flink Python Processing Logic

Part 3: Deploying the Managed Flink Application

Part 4: Testing and Verification

Part 5: Cleanup

Part 6: Next Steps

### **Purpose of this Lab**

This lab serves as your first step into the world of serverless data processing on the cloud. Instead of managing servers, you will use a fully managed AWS service to build an end-to-end pipeline. This approach allows you to focus purely on the application logic and data flow.

By completing this lab, you will:

* **Provision Core AWS Services:** Configure S3 for data storage.
* **Understand IAM Permissions:** Create a dedicated IAM role that grants the Flink service secure access to other AWS resources.
* **Use Managed Flink with Python:** Deploy and run a PyFlink script using Amazon's managed service.
* **Implement a Full Data Flow:** Process an in-memory dataset and see the results appear in your data lake (S3).
* **Gain Foundational Cloud Skills:** Build a simple, scalable architecture that is the basis for more complex real-time and batch systems.

### **Prerequisites**

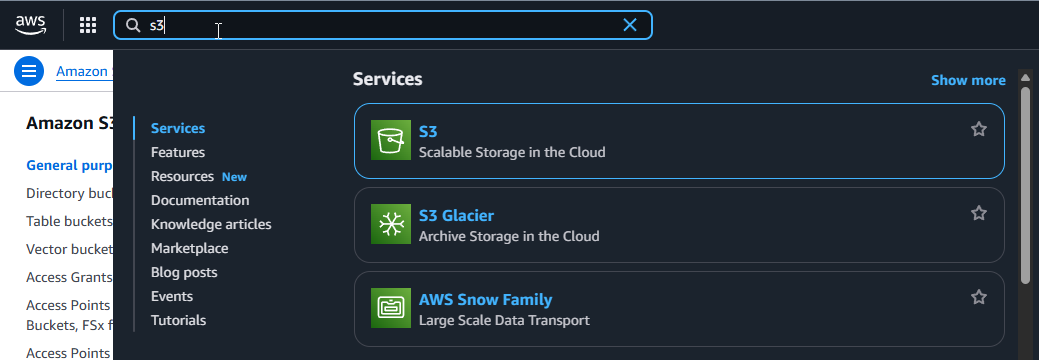
* An active AWS account with permissions to create IAM roles, S3 buckets, and Managed Flink applications.

### **Part 1: Setting Up Your AWS Infrastructure**

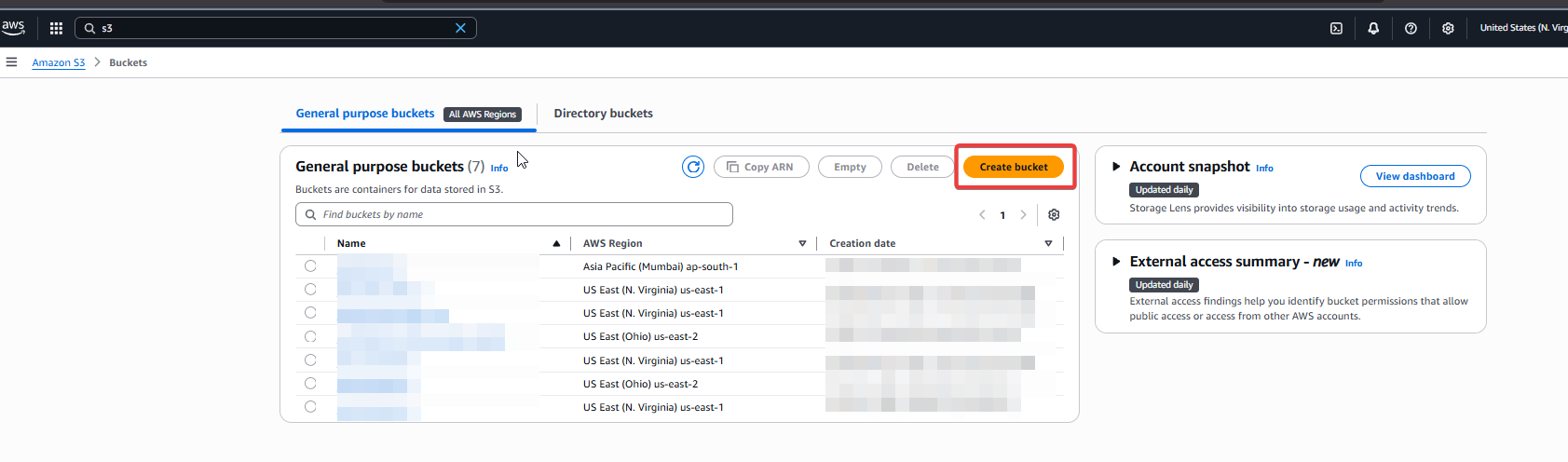
#### **Step 1: Create an S3 Bucket**

This bucket will be the destination, or "sink," for the data processed by Flink.

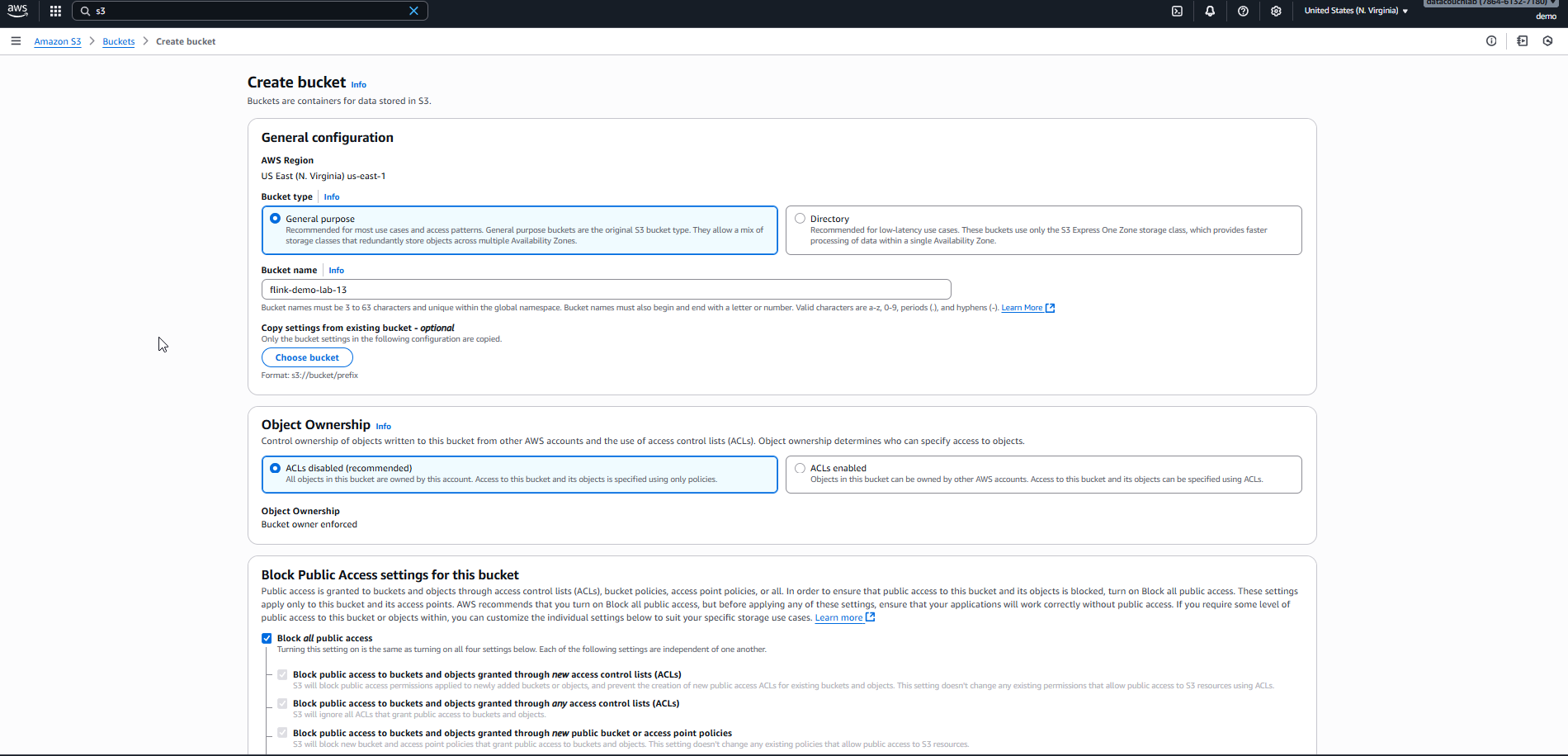
1. Navigate to the **S3** service in the AWS Management Console.



1. Click **Create bucket**.



1. Enter a **globally unique bucket name** (e.g., piramal-flink-lab-output-YOURNAME). Keep the region as is.

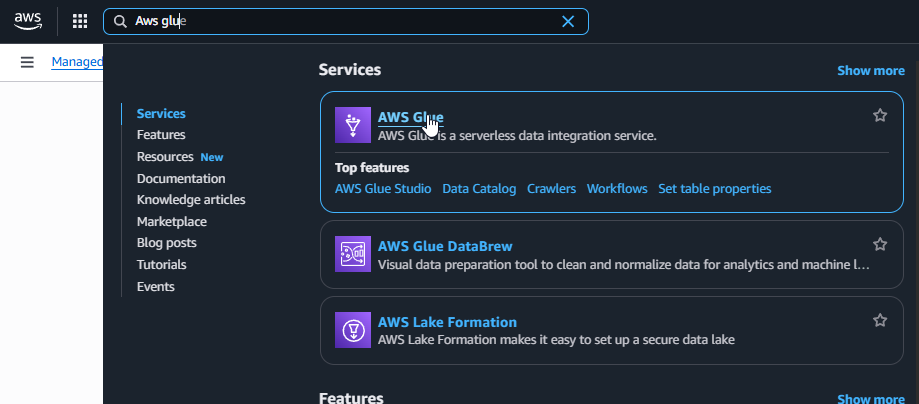


1. Leave all other settings as default and click **Create bucket**. Note this bucket name for later.

#### **Step 2: Create an AWS Glue Database**

The Flink application setup now requires a Glue Data Catalog database to be associated with it.

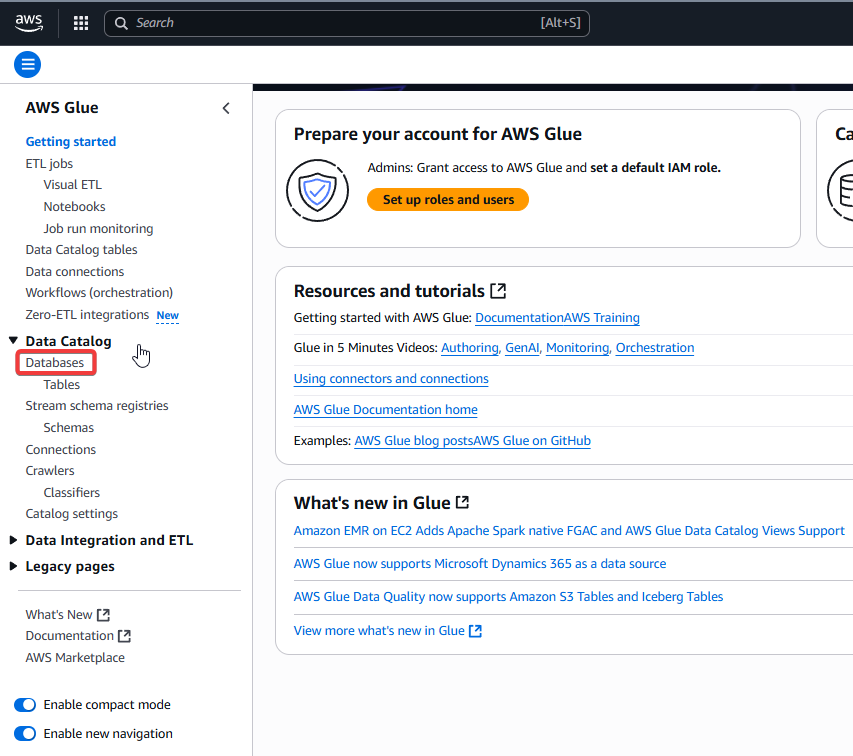
1. Navigate to the **AWS Glue** service.



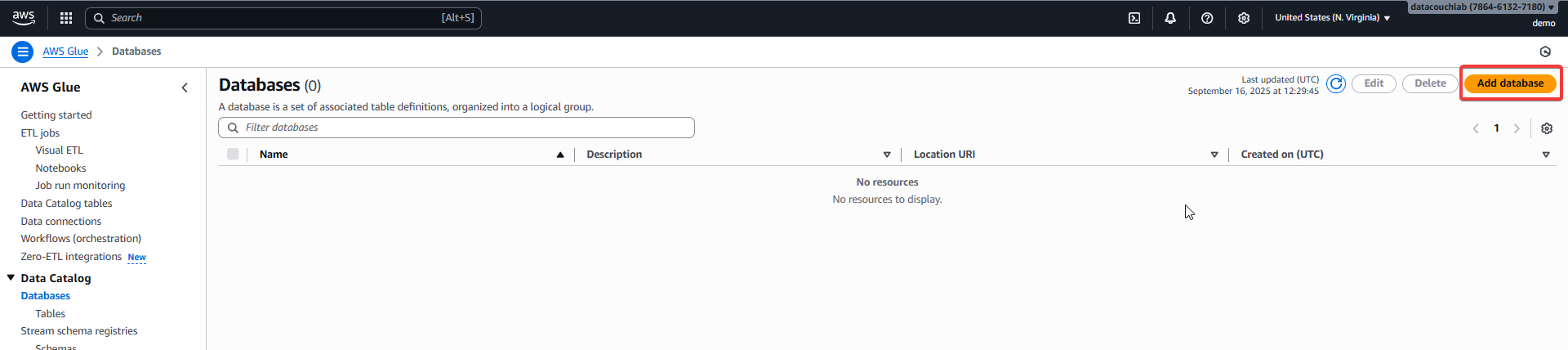
1. In the left navigation pane, click to expand the **Data Catalog** section.



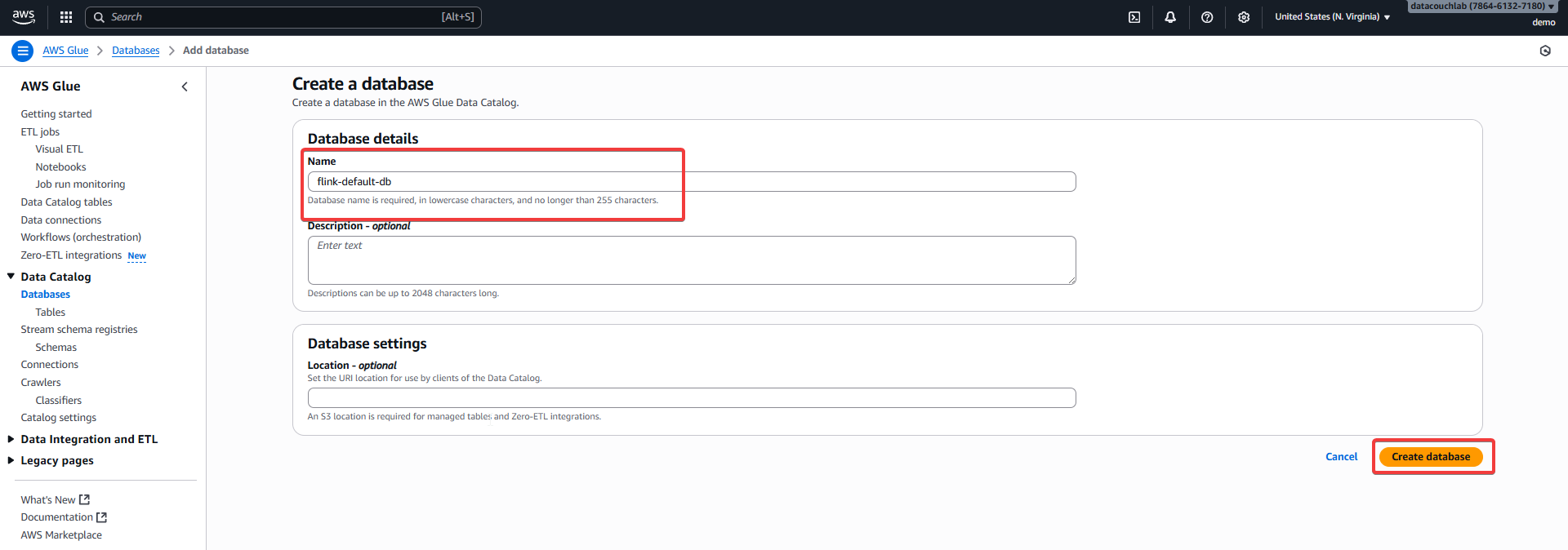
1. Under **Data Catalog**, click **Databases**.



1. Click **Add database**.



1. Enter a **Database name** (e.g., flink-default-db).

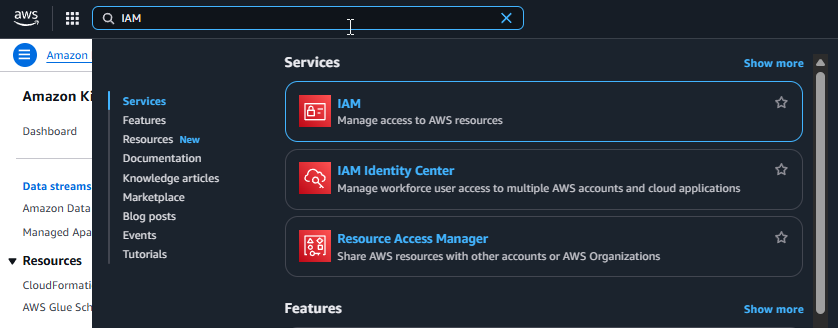


1. Click **Create database**.

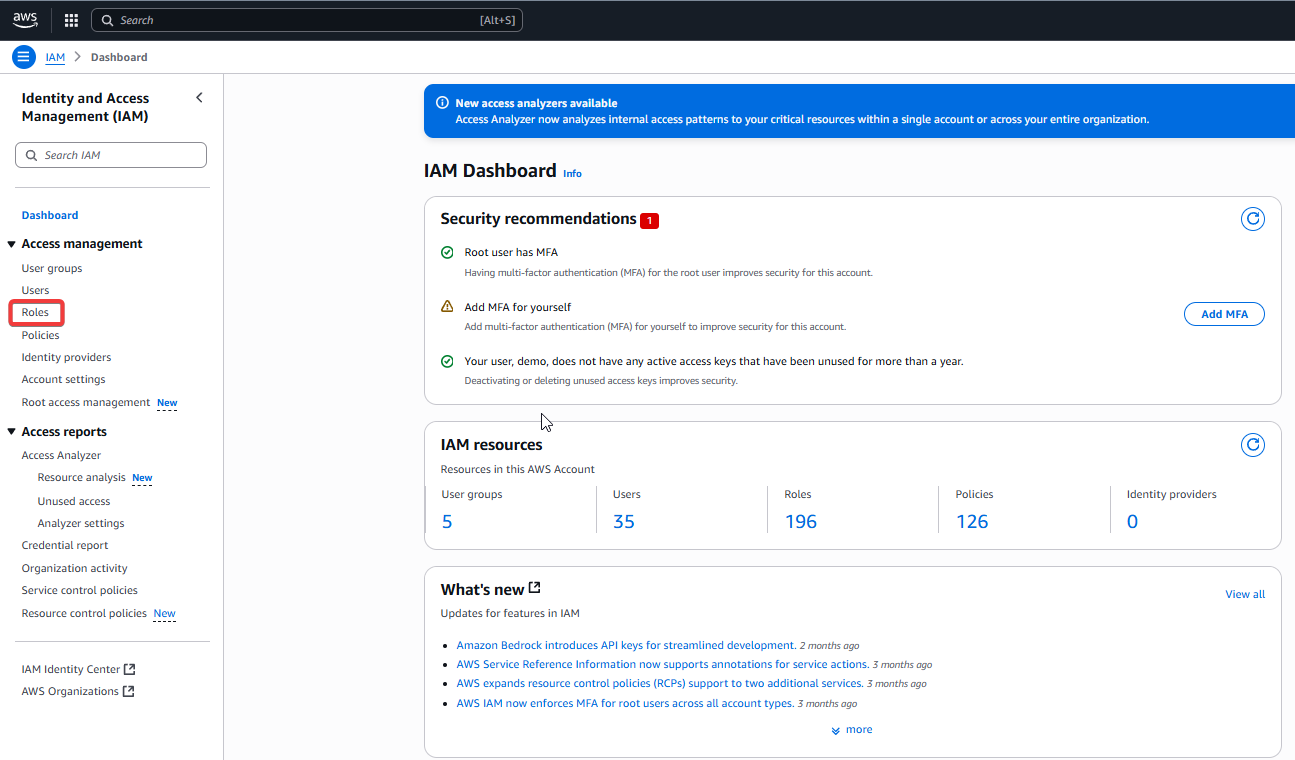
#### **Step 4: Create an IAM Role for Flink**

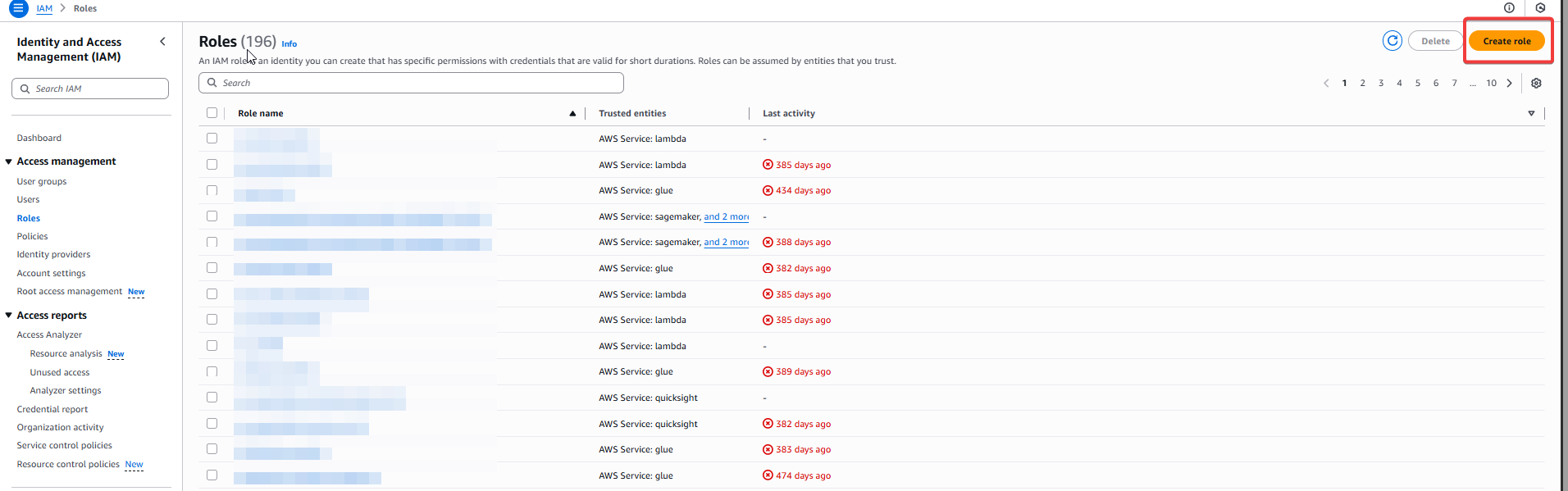
Flink needs explicit permissions to read from Kinesis and write to S3.

1. Navigate to the **IAM** service in the AWS Console.

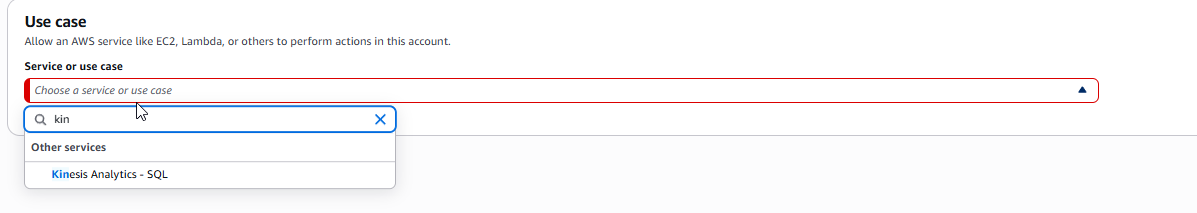


1. Go to **Roles** and click **Create role**.



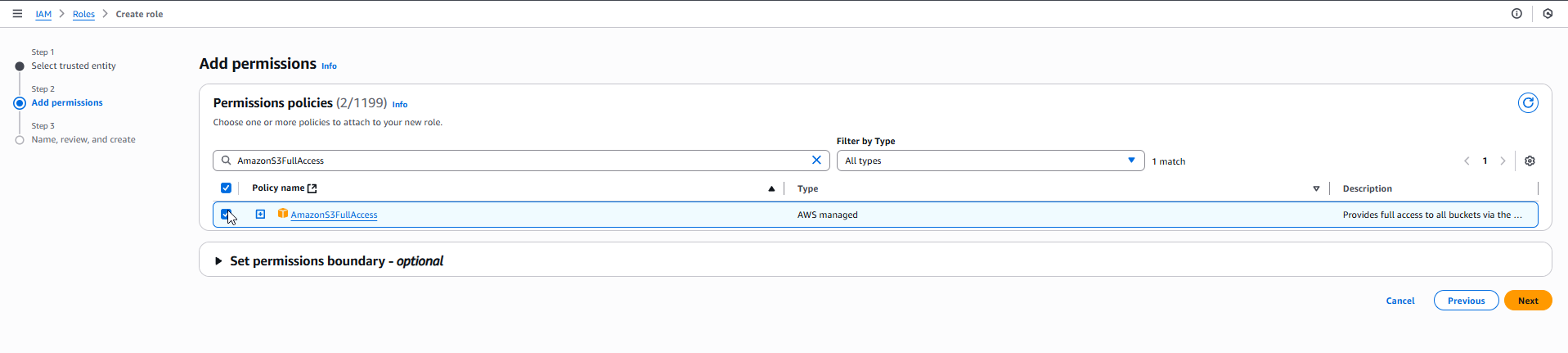


1. For the trusted entity, select **AWS service**. For the use case, choose **Kinesis** from the dropdown and then select **Kinesis Data Analytics** underneath it. Click **Next**.



1. On the "Add permissions" page, search for and add the following two policies:
   * AmazonS3FullAccess
   * AWSGlueServiceRole

**(Note: In a production environment, you would create a custom, least-privilege policy instead of using full access).**

****

****

1. Click **Next**.
2. Enter a **Role name** (e.g., Flink-Kinesis-S3-Role) and click **Create role**.

### **Part 2: Developing the Flink SQL Processing Logic**

The following SQL script defines a source table that reads from our Kinesis stream, a sink table that writes to our S3 bucket, and an INSERT statement that filters the data. Our goal is to only store sensor readings where the temperature is above a "critical" threshold of 30 degrees.

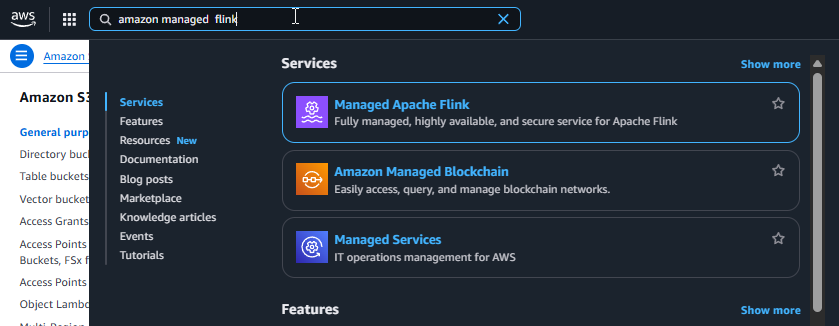
| %flink.pyflink  # Step 1: Import necessary classes  from pyflink.table import DataTypes, TableEnvironment, EnvironmentSettings  from pyflink.table.expressions import col  # Step 2: Create a BATCH Table Environment  # We explicitly create a batch environment because our data source (a Python list)  # is finite. This tells Flink to process all the data and then finish.  settings = EnvironmentSettings.new\_instance().in\_batch\_mode().build()  table\_env = TableEnvironment.create(settings)  # Step 3: Define your raw data  sensor\_data = [  (101, 25.5),  (102, 32.8),  (103, 29.0),  (104, 35.1),  (105, 22.4)  ]  # Step 4: Define the schema for the data  schema = DataTypes.ROW([  DataTypes.FIELD("sensor\_id", DataTypes.INT()),  DataTypes.FIELD("temperature", DataTypes.FLOAT())  ])  # Step 5: Create the source Flink Table from the in-memory data  sensor\_table = table\_env.from\_elements(sensor\_data, schema)  # Step 6: Process the data to filter for critical alerts  critical\_alerts = sensor\_table.where(col('temperature') > 30.0)  # Step 7: Define the S3 Sink Table using SQL  # A "sink" is a destination for data. This SQL command creates a table definition  # that tells Flink how and where to write the data.  # Make sure to replace 'your-s3-bucket-name' with the actual name of your S3 bucket.  s3\_bucket\_name = 'piramal-flink-lab-output-YOURNAME'  table\_env.execute\_sql(f"""  CREATE TABLE s3\_sink (  sensor\_id INT,  temperature FLOAT  ) WITH (  'connector' = 'filesystem',  'path' = 's3a://{s3\_bucket\_name}/critical-alerts/',  'format' = 'csv'  )  """)  # Step 8: Execute the job to insert the filtered data into the S3 sink table.  # This command takes the data from our 'critical\_alerts' table and inserts it  # into the 's3\_sink' table we just defined. The .wait() call ensures the script  # waits for the job to complete.  critical\_alerts.execute\_insert('s3\_sink').wait()  print("Job finished. Data written to S3.") |
| --- |

You will copy and paste this code later. Remember to replace flink-demo-lab-13-YOURNAME with your specific S3 bucket name.

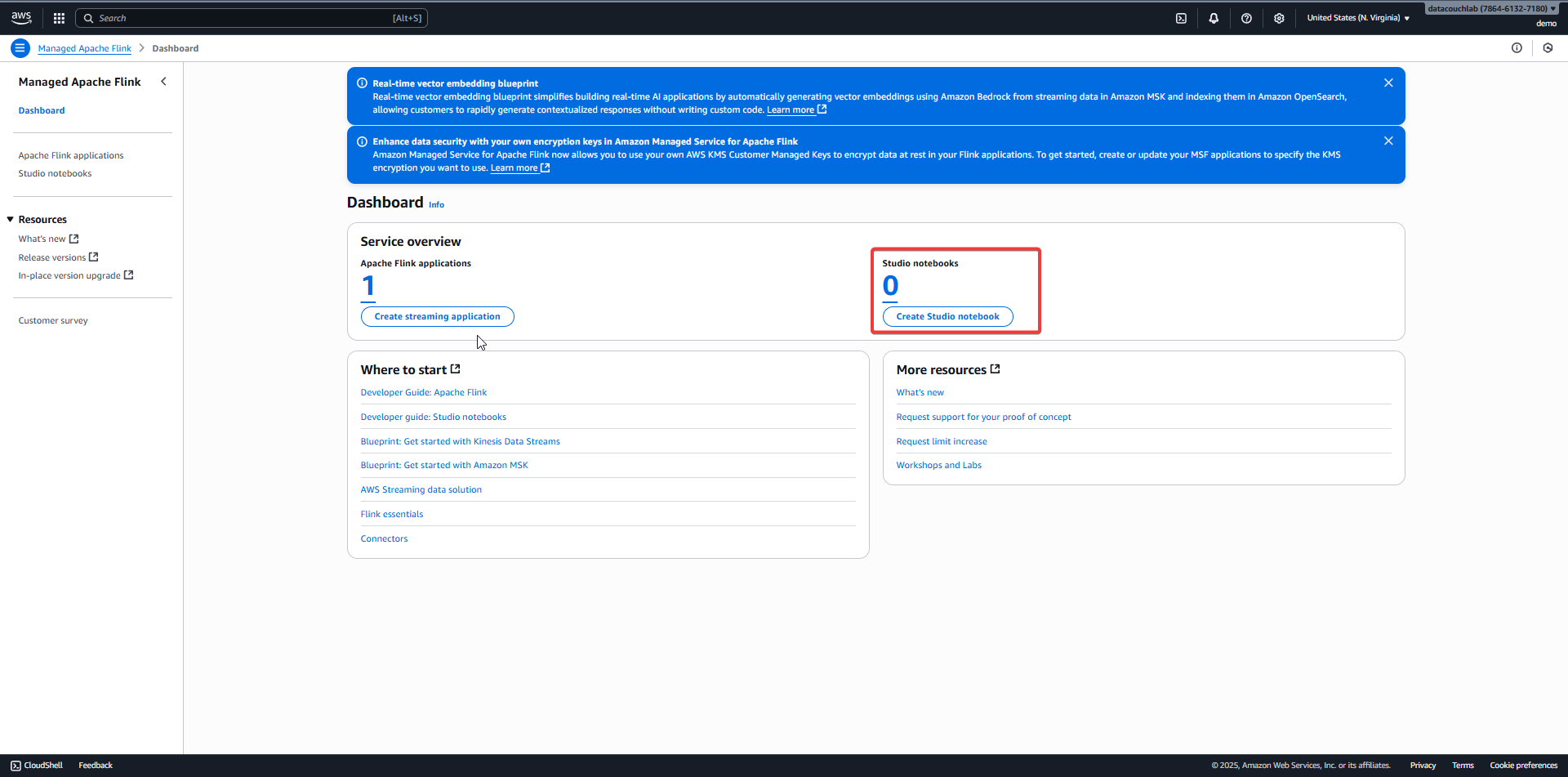
### **Part 3: Deploying the Managed Flink Application**

#### **Step 1: Create a Managed Flink Application**

1. Navigate to **Amazon Managed Service for Apache Flink**.



1. Click **Create Studio notebook**.

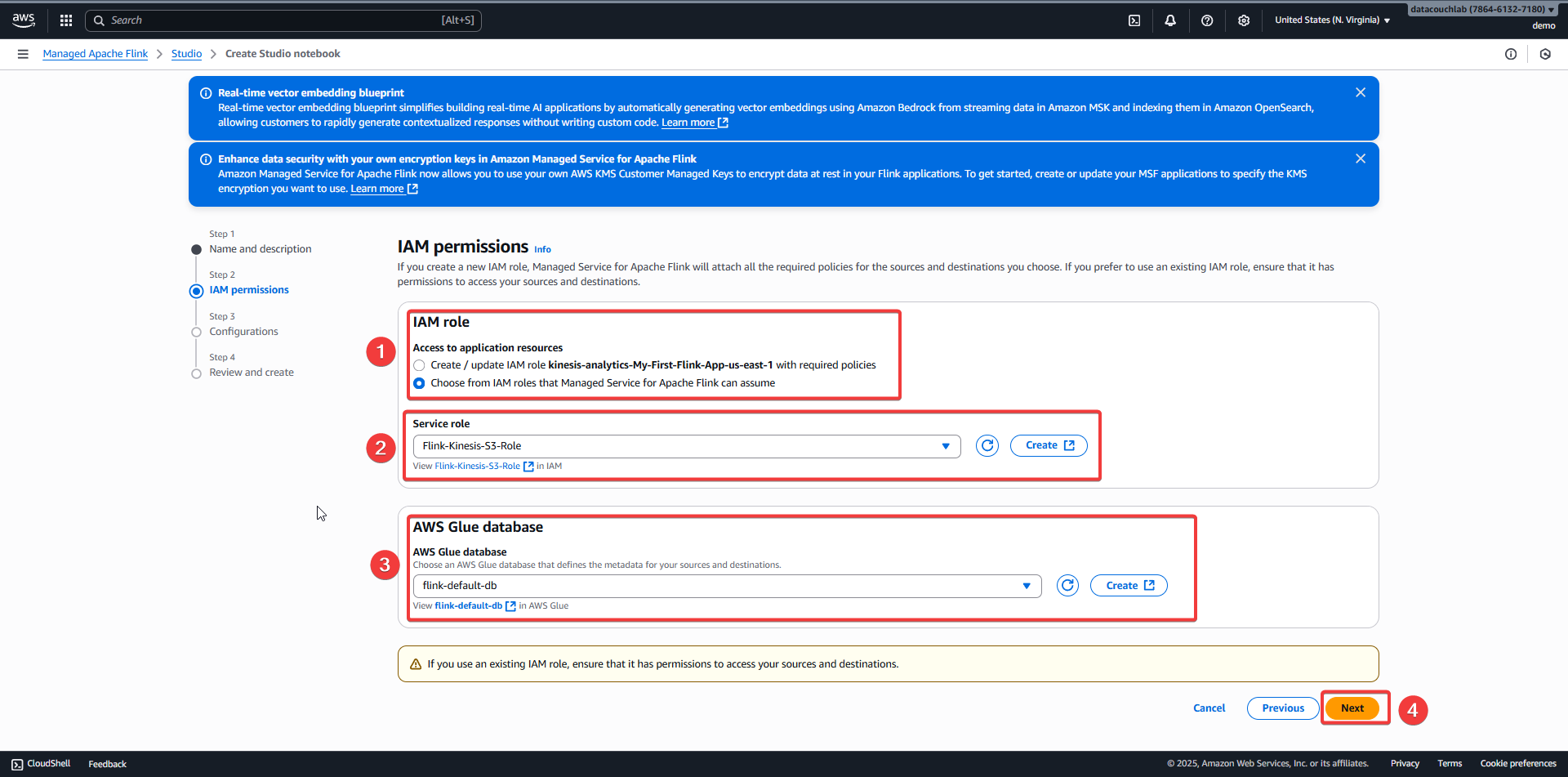


1. On the **Name and description** page, under **Creation method**, select **"Create with custom settings"**.
2. Enter an **Application name** (e.g., My-First-Flink-App) and click **Next**.

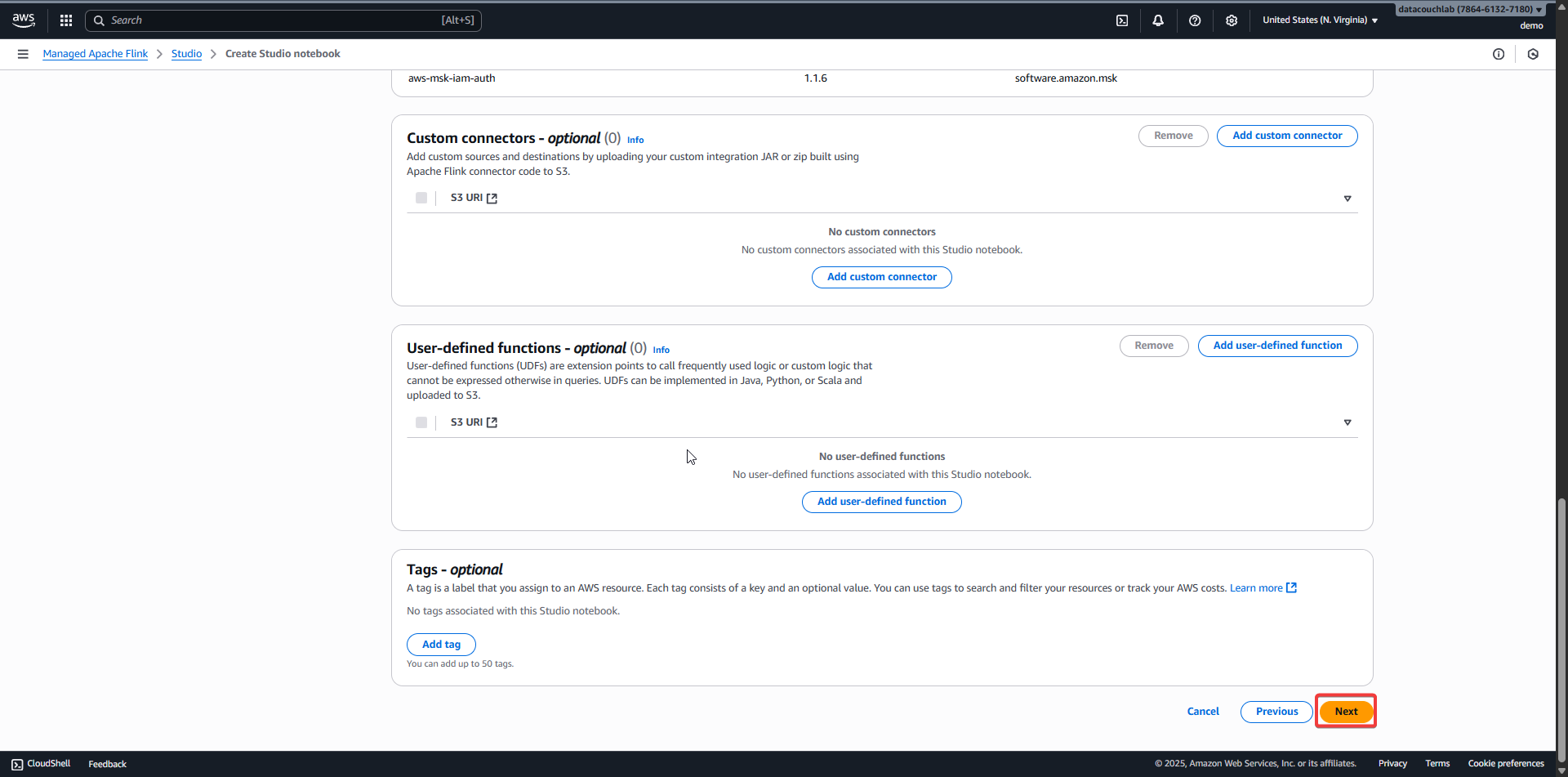


1. On the **IAM permissions** page:

* Click the dropdown under **Service role** and select the Flink-Kinesis-S3-Role you created earlier.
* Under **AWS Glue database**, select the flink-default-db you created.
* Click **Next**.



1. On the **Configurations** page, leave all settings as their default values and click **Next**.



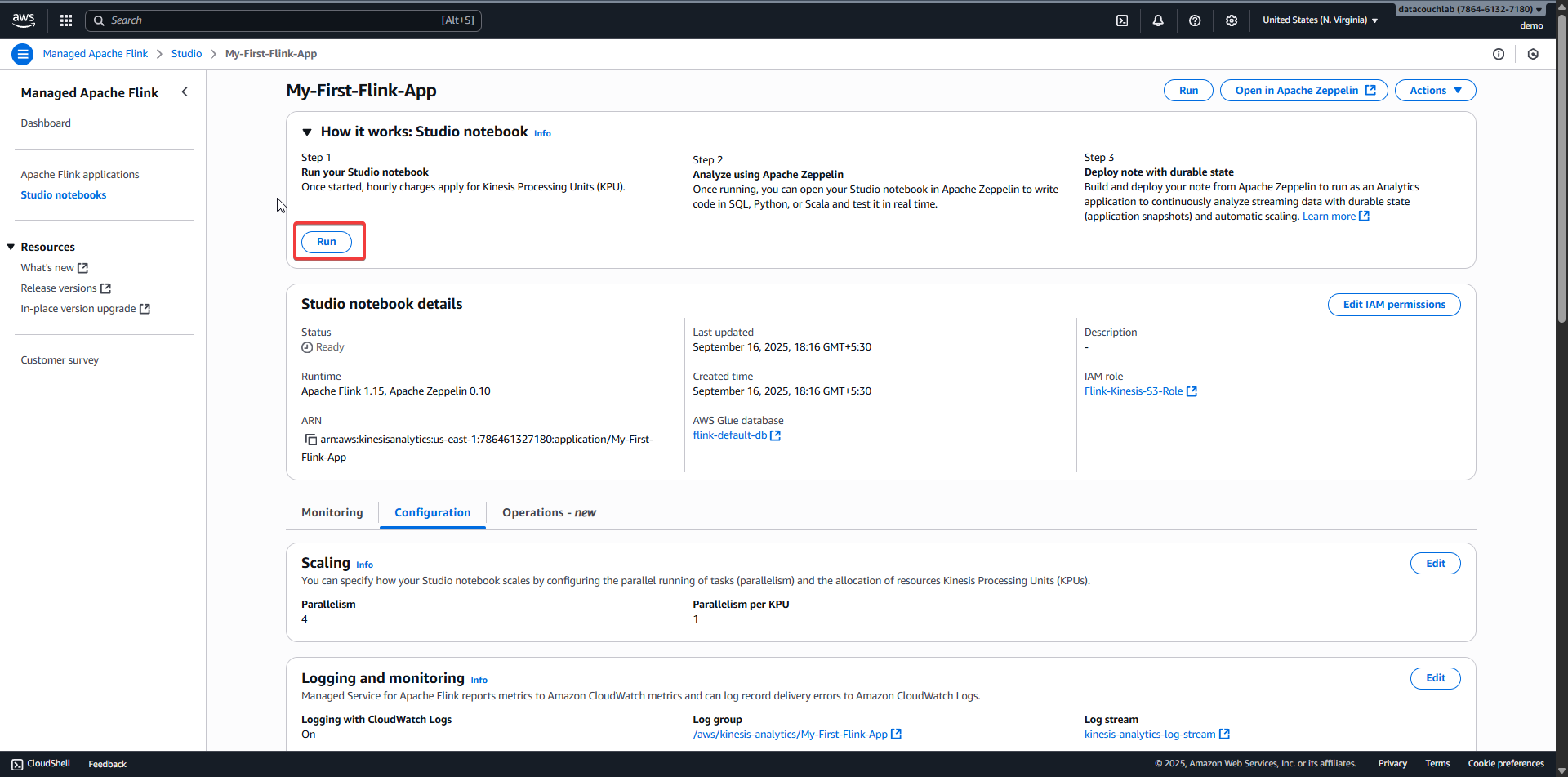
1. On the **Review and create** page, click **Create Studio notebook**. It will take several minutes to provision.

#### **Step 2: Configure and Run the Application**

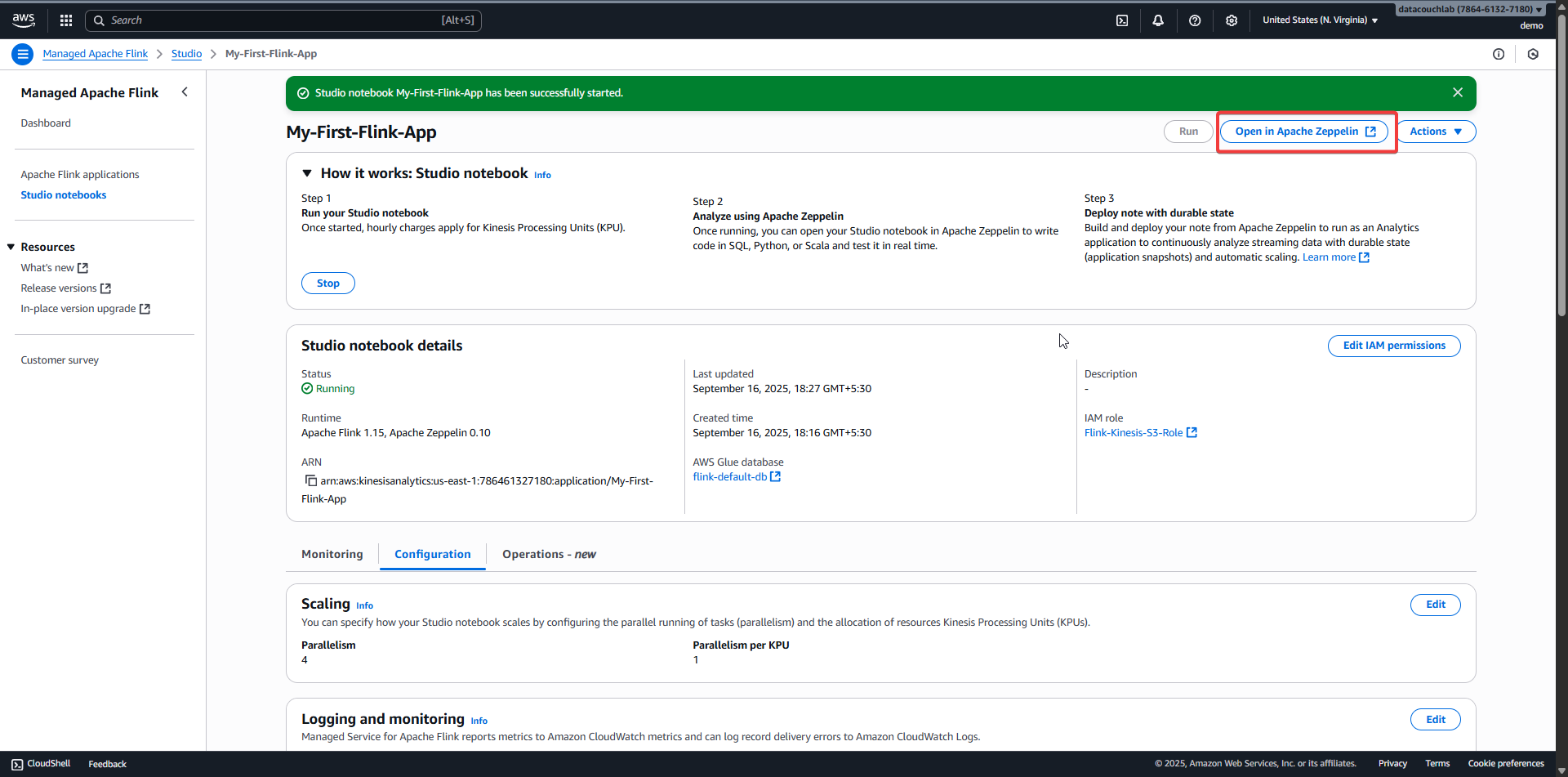
Once the application status is **Ready**, you must start it. Click the blue **Run** button.

Confirm the action. The application status will change to **Starting**. Wait for it to become **Running**.

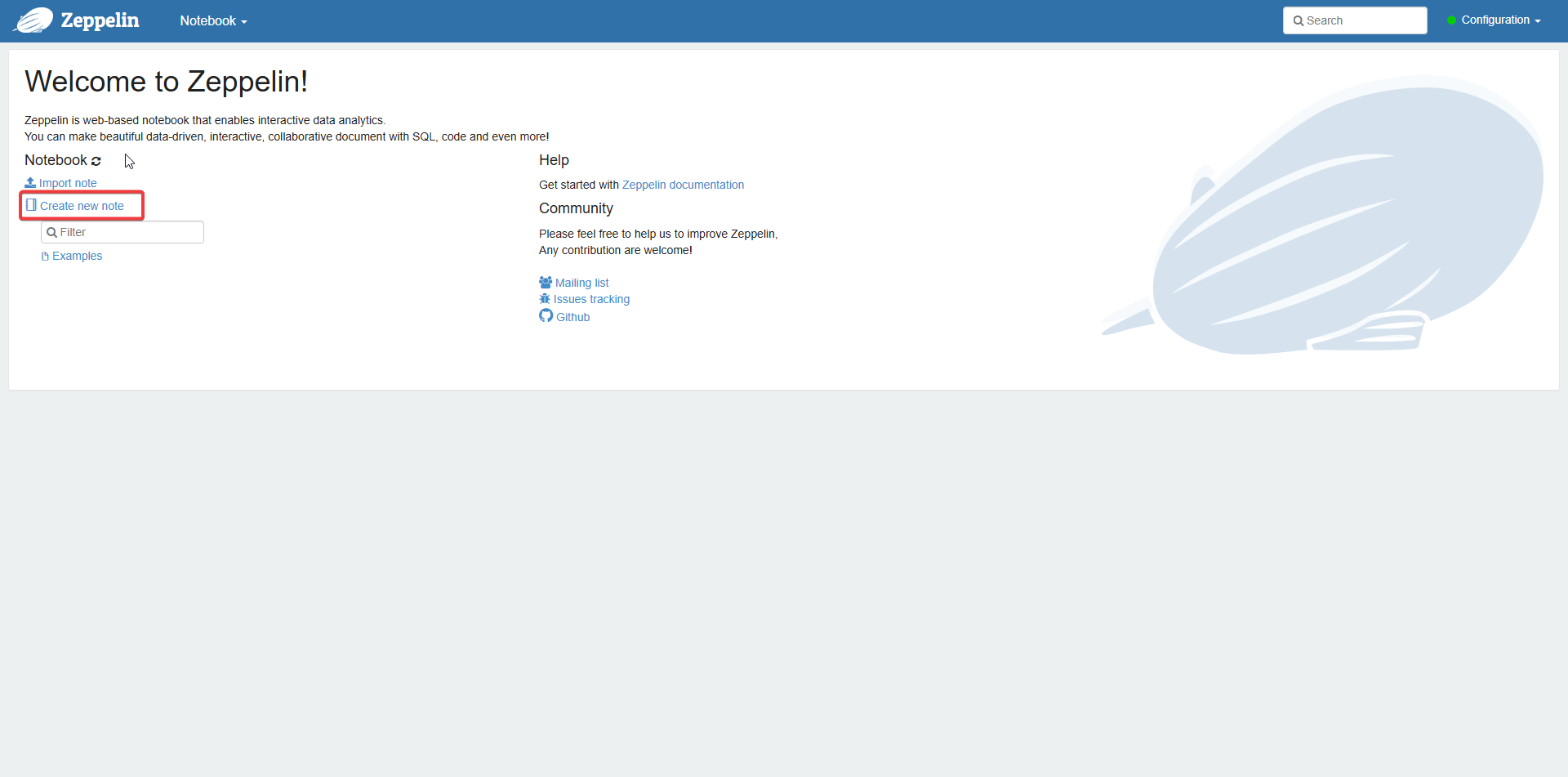
1. Once the application status is **Ready**, you must start it. Click the blue **Run** button.

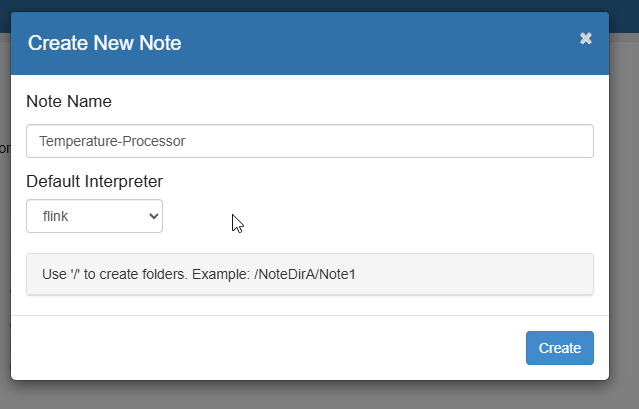


1. Confirm the action. The application status will change to **Starting**. Wait for it to become **Running**.
2. Once the application status is **Running**, click **Open in Apache Zeppelin**.

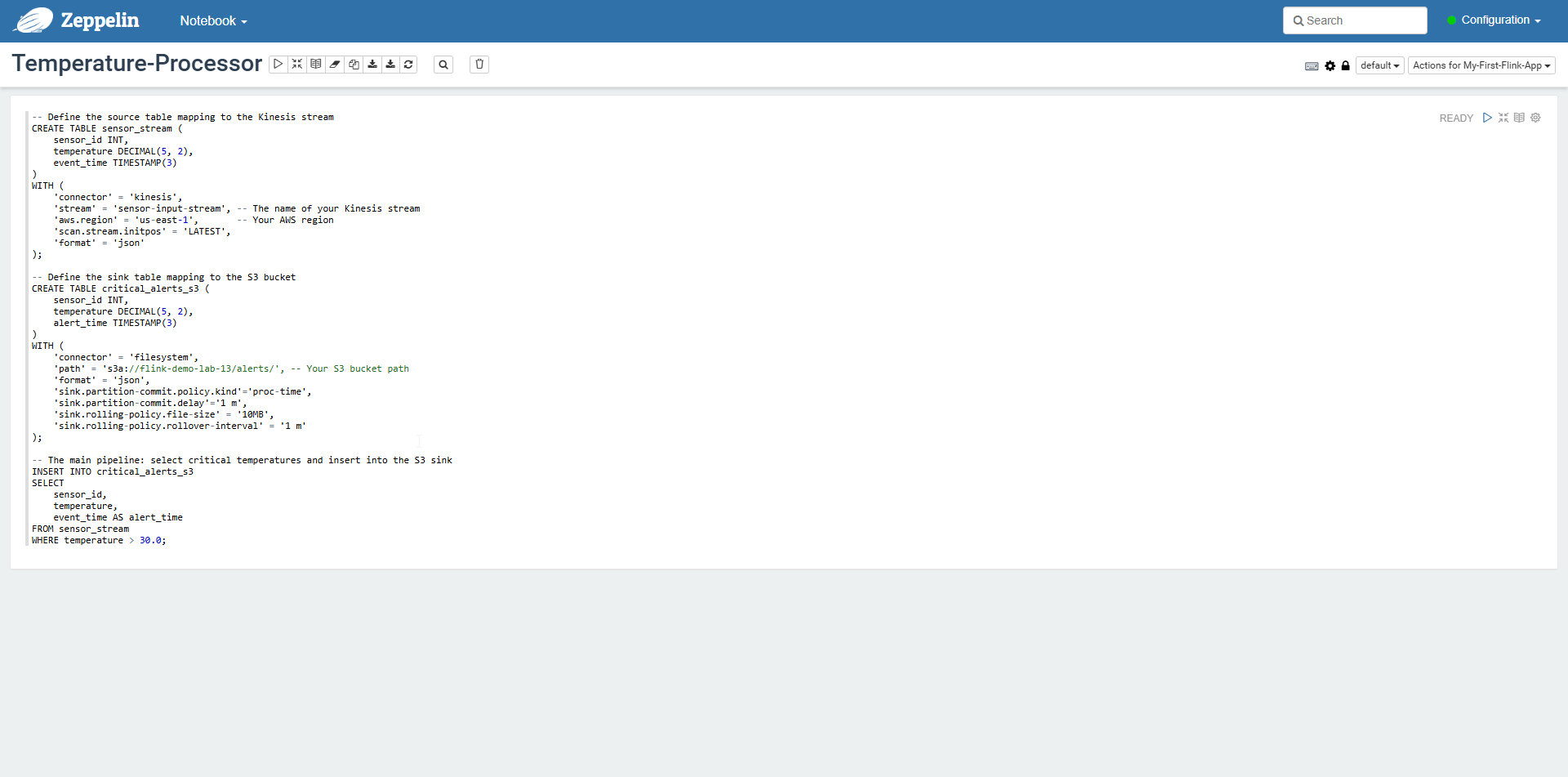


1. A new Zeppelin notebook will open. Click **Create a new note** and name it Temperature-Processor.

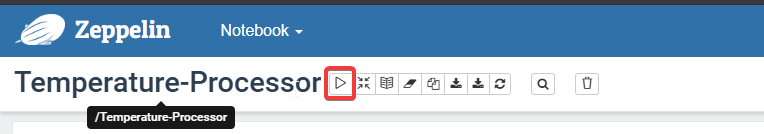


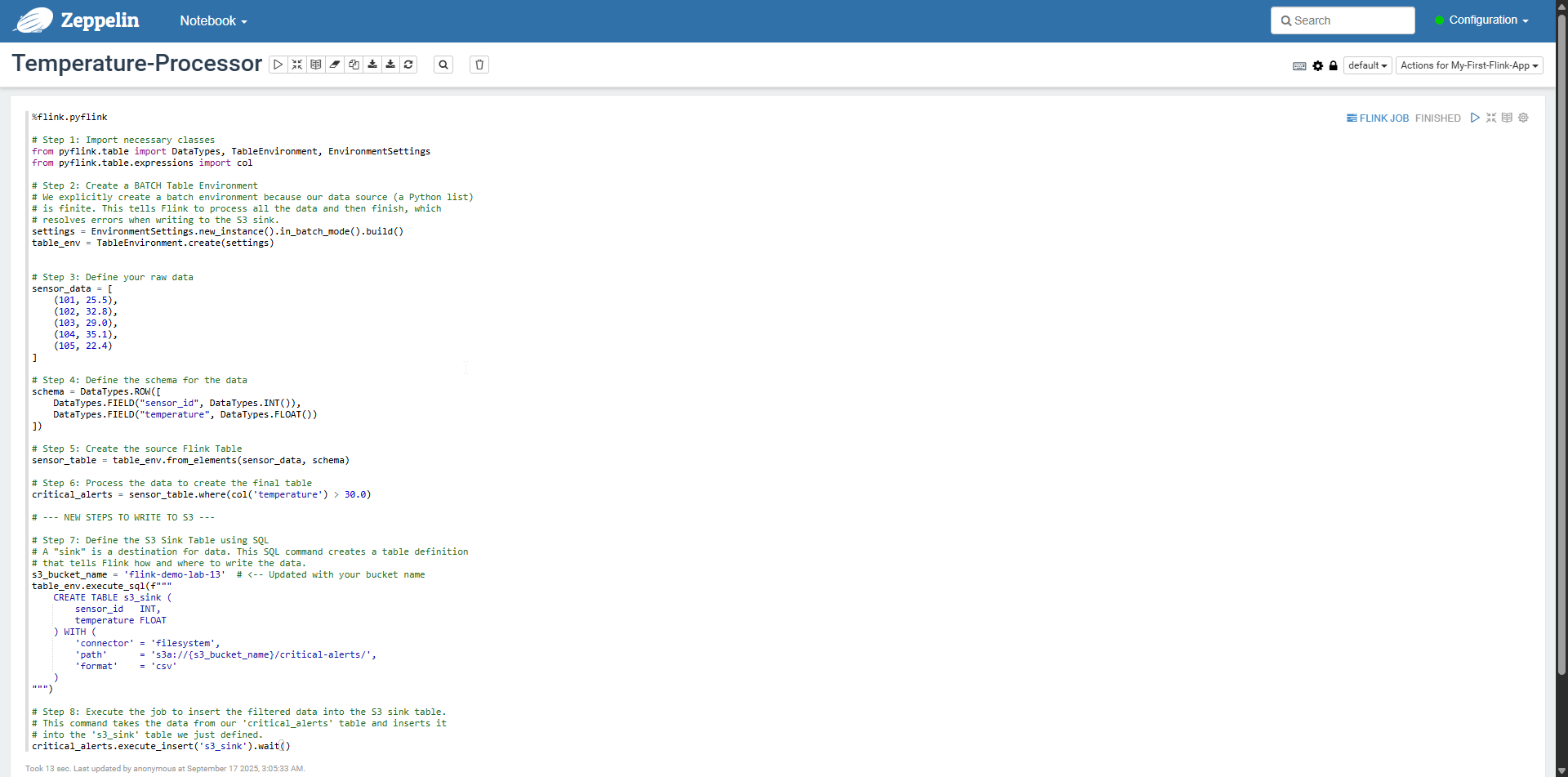


1. In the first notebook cell, paste the entire PyFlink script from Part 2. **Remember to update the S3 bucket name in the script.**



1. Click the "Run" (play) button to execute the cell. Zeppelin will interpret the script and deploy it as a Flink batch job. Since this is a batch job, it will run to completion and then stop.





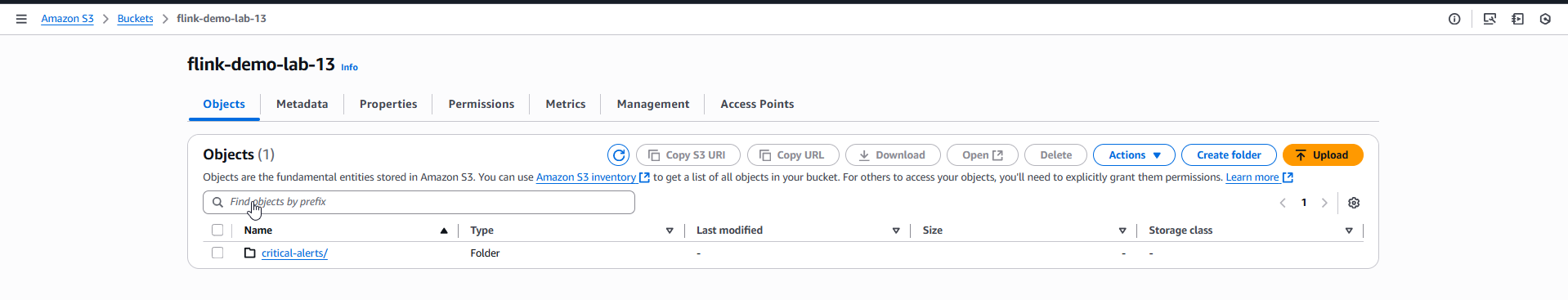
### **Part 4: Testing and Verification**

#### **Step 1: Verify the Output in S3**

If you have never configured the AWS CLI before, you need to provide it with security credentials to access your AWS account.

1. Once the Zeppelin notebook finishes running, navigate to your S3 bucket in the AWS Console.

### You should see a new folder named critical-alerts/.

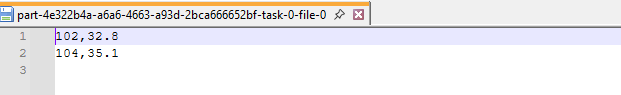


1. You should see a new folder named critical-alerts/.



1. Open the downloaded file with a text editor. The contents should be the records for the critical temperature alerts:

| 102,32.8 104,35.1 |
| --- |



1. This confirms your end-to-end pipeline is working correctly!

### **Part 5: Cleanup**

To avoid ongoing charges, it is crucial to delete all the resources you created.

1. **Delete the Flink Application:** In the Managed Flink console, select your application and choose **Actions -> Delete**. Confirm the deletion.
2. **Empty and Delete the S3 Bucket:** In the S3 console, navigate to your bucket, select all files and folders, and delete them. Then, delete the bucket itself.
3. **Delete the IAM Role:** In the IAM console, find and delete the Flink-S3-Role.
4. **Delete the Glue Database:** In the Glue console, find and delete the flink-default-db database.

### **Part 6: Next Steps**

* **Modify the Python:** Try more complex data transformations using the PyFlink Table API, such as aggregations (group\_by, select) or adding new columns.
* **Change the Source:** Instead of an in-memory list, modify the script to read data from a source file (like a CSV or Parquet file) already stored in another S3 bucket.
* **Change the Sink:** Explore writing the output to a different format (like 'json' or 'parquet') or to a different sink connector supported by Flink.