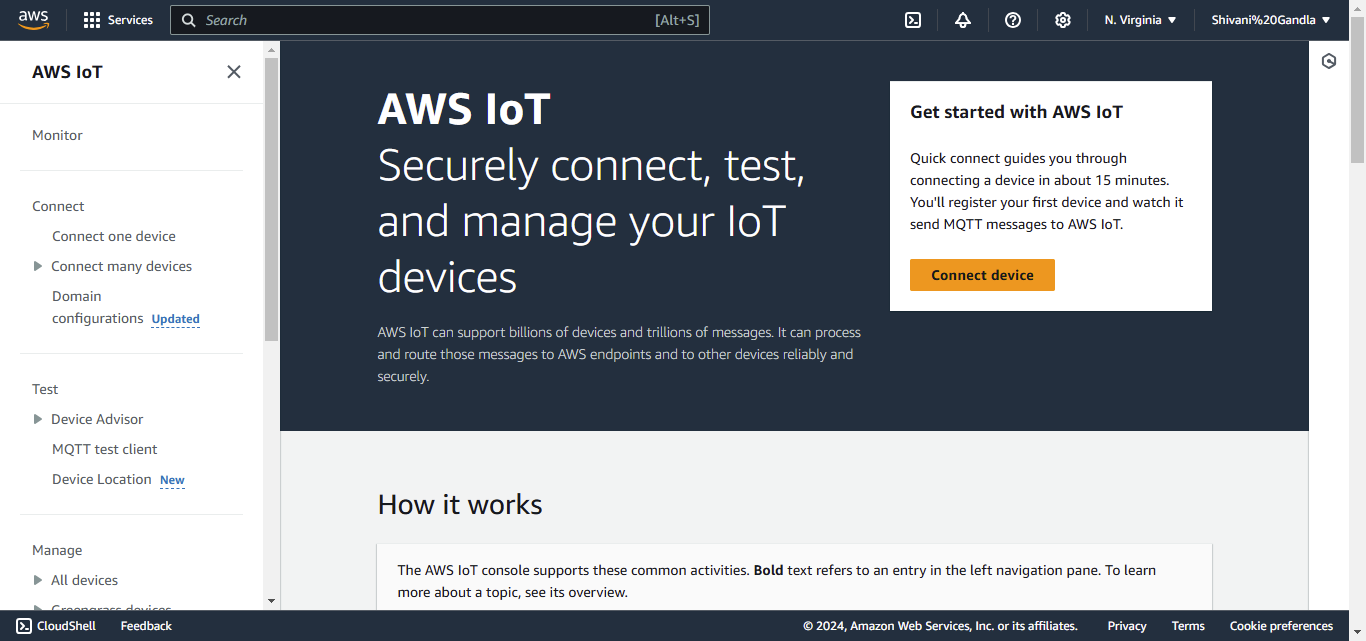
**Set Up IoT Devices in AWS IoT Core**

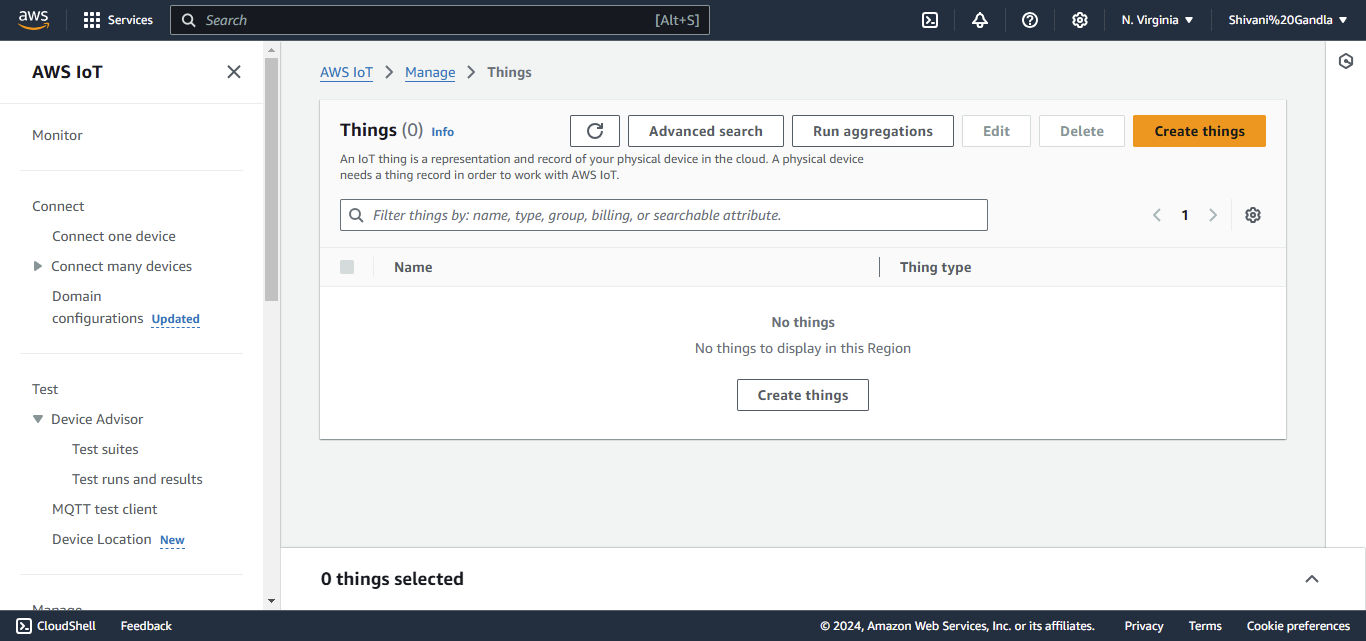
**Step 1: Set Up IoT Devices in AWS IoT Core**

**1. Register Devices**

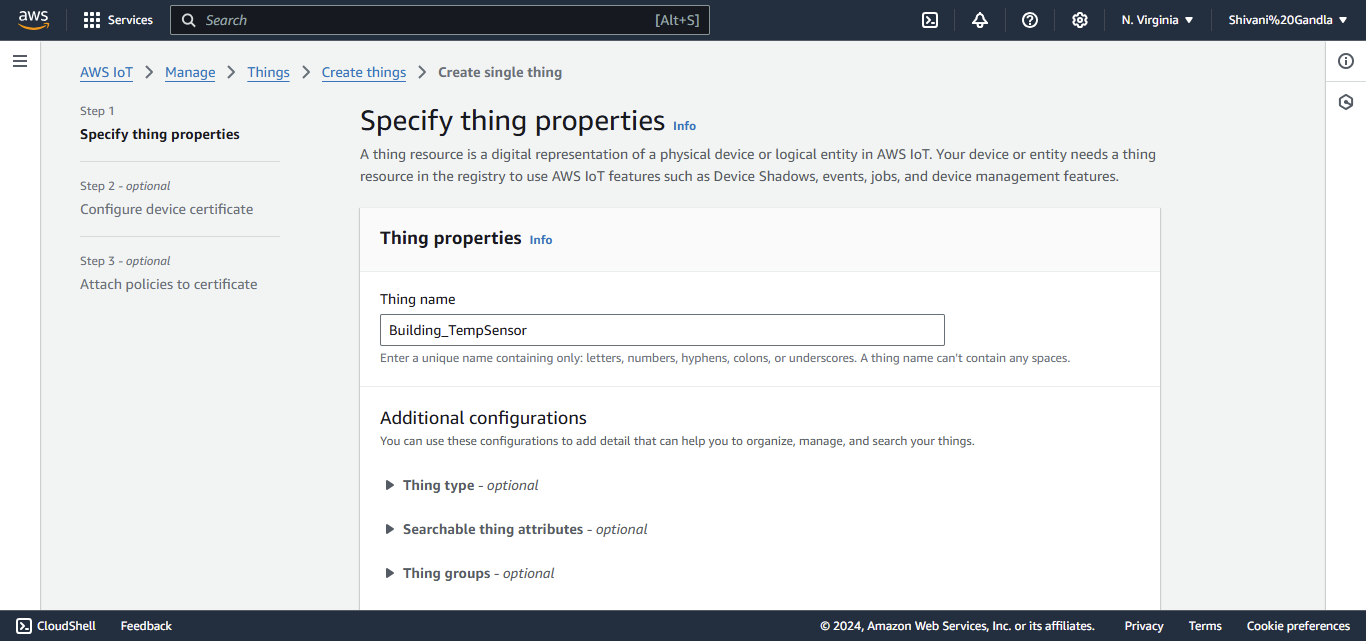
1. Go to the [AWS Management Console](https://aws.amazon.com/console/).
2. Navigate to **IoT Core**.
3. Under **Manage > Things**, click **Create Thing**:

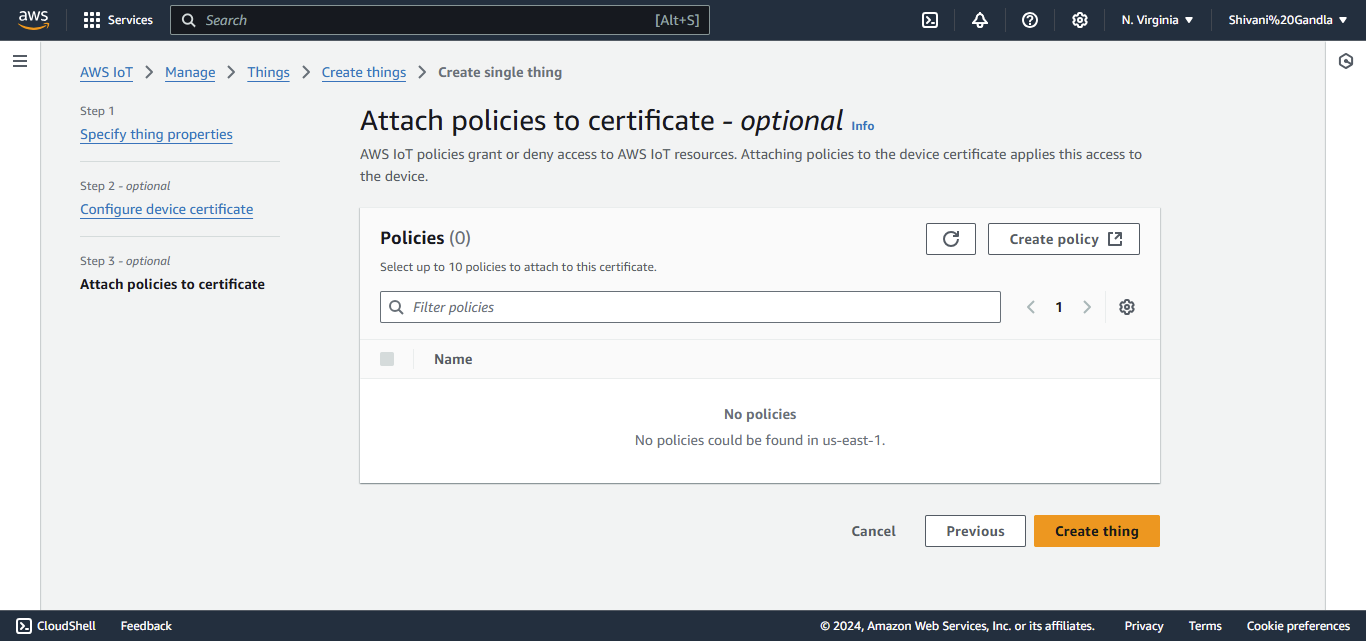
****

* + Choose "Create a single thing."

****

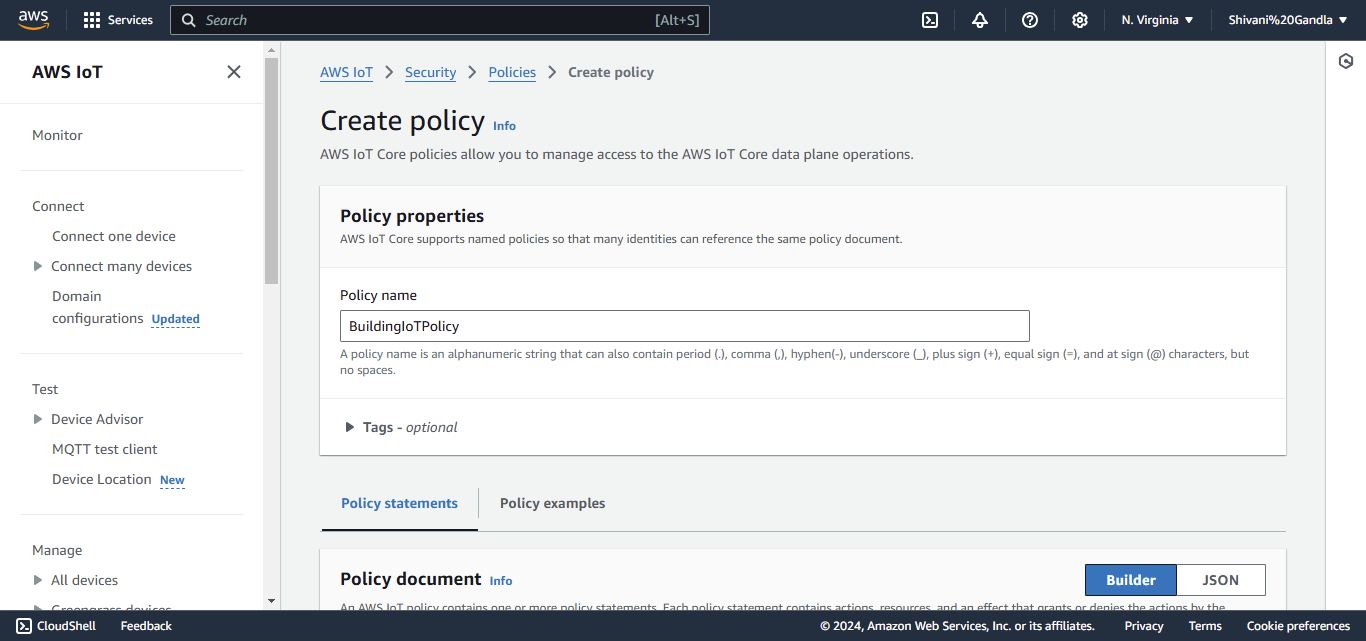
* Provide a **name** (e.g., Building\_TempSensor).

****

****

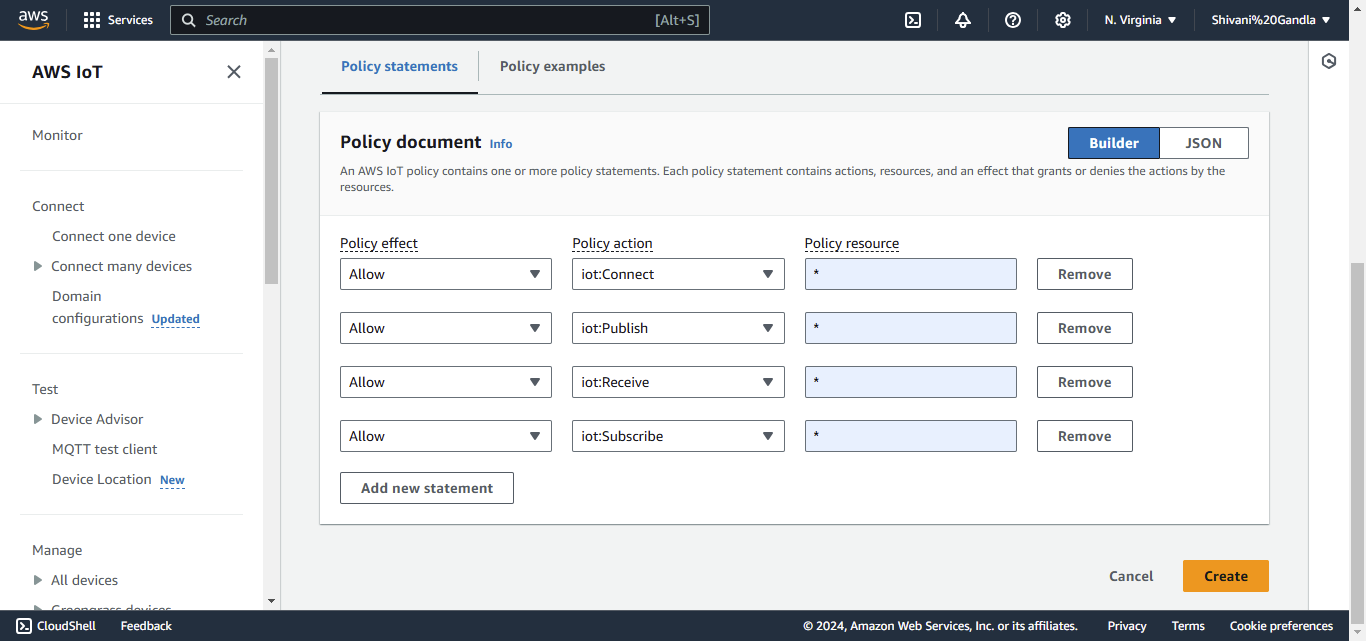
**Create a New Policy:**

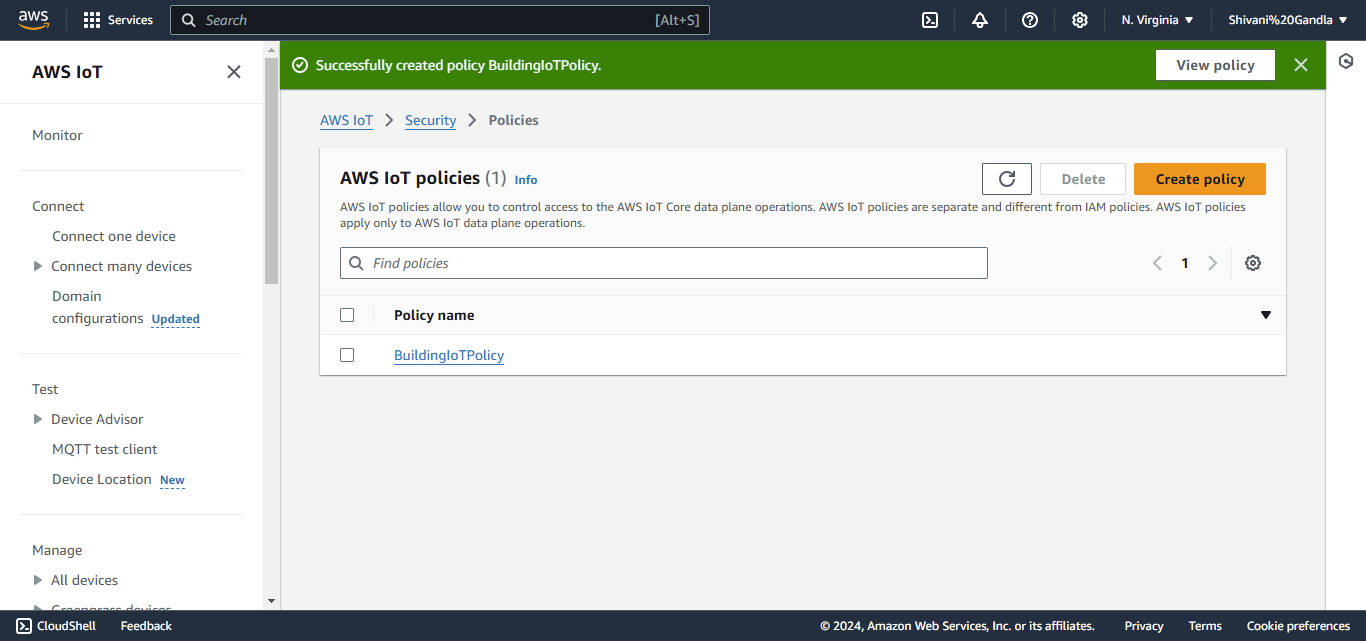
* Click on **Create Policy**.
* Enter a name for the policy (e.g., BuildingIoTPolicy).



Add policy statements:

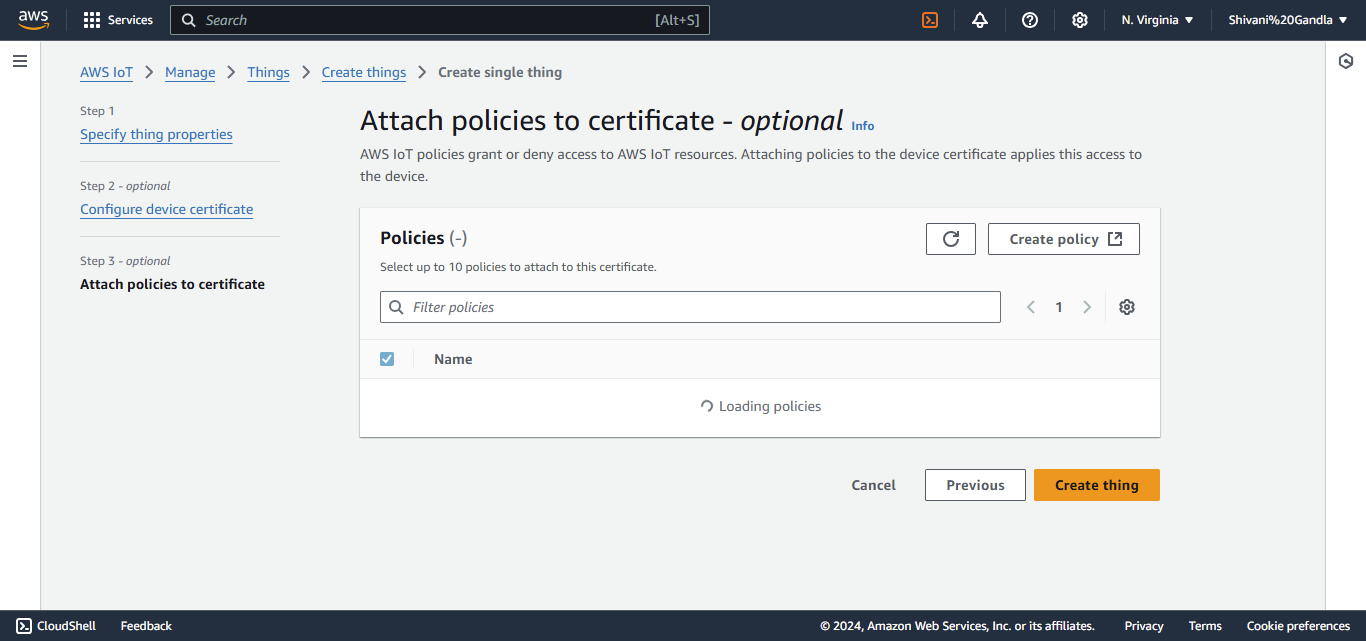
* Action: iot:Connect, iot:Publish, iot:Subscribe, iot:Receive.

****

****

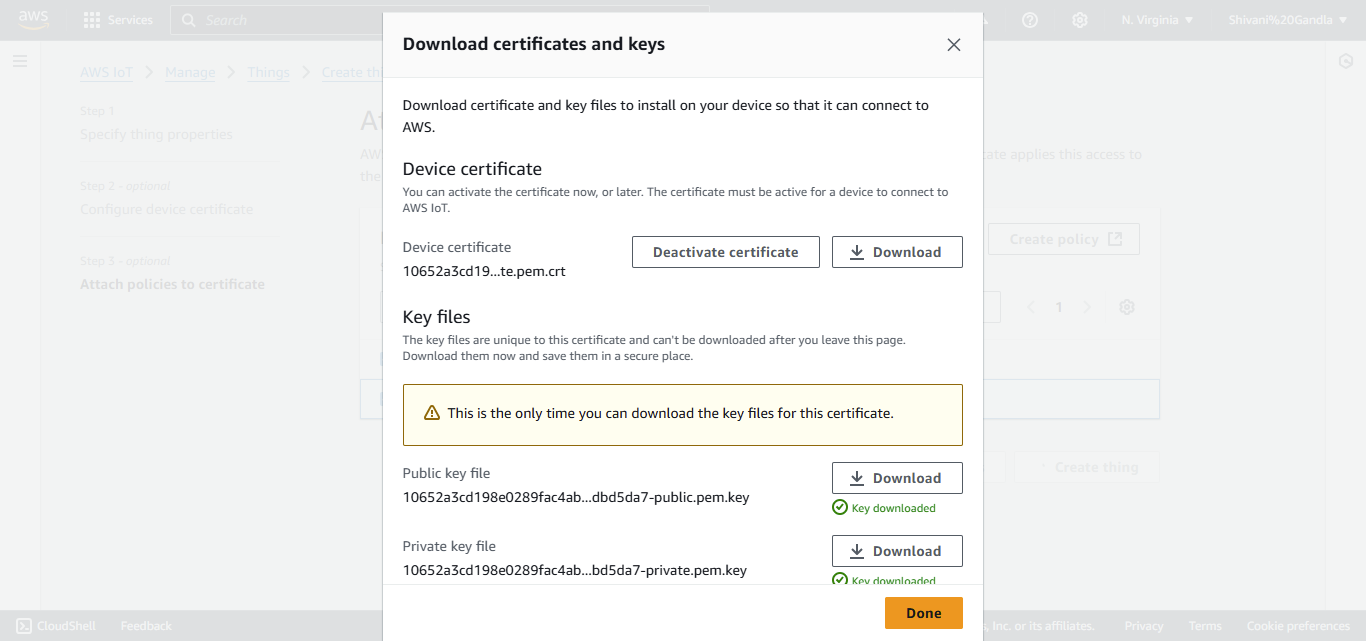
**Attach a Policy:**

* Attach an IoT policy to the certificate:
  + If a policy doesn’t exist, you can create one under **Secure > Policies**

****

Download the following files when prompted:

* X.509 certificate (for the IoT device).
* Private key.
* Public key.
* Root CA certificate.

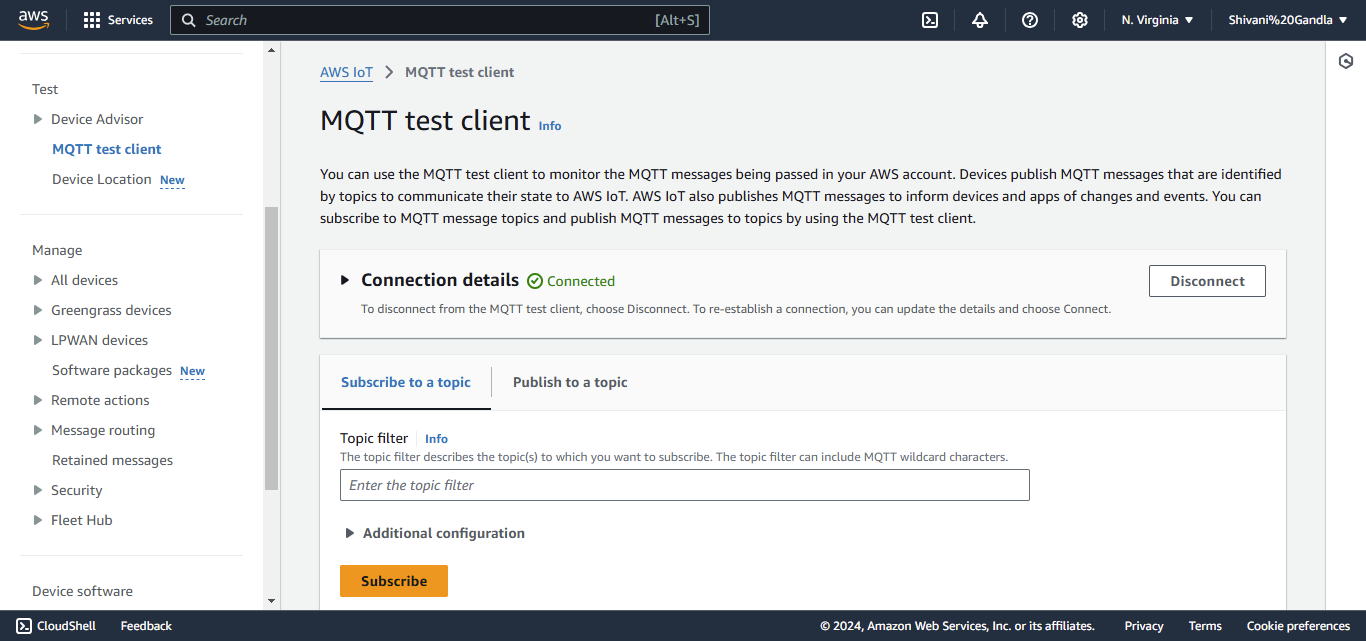
****

****

**2. Configure MQTT Topics**

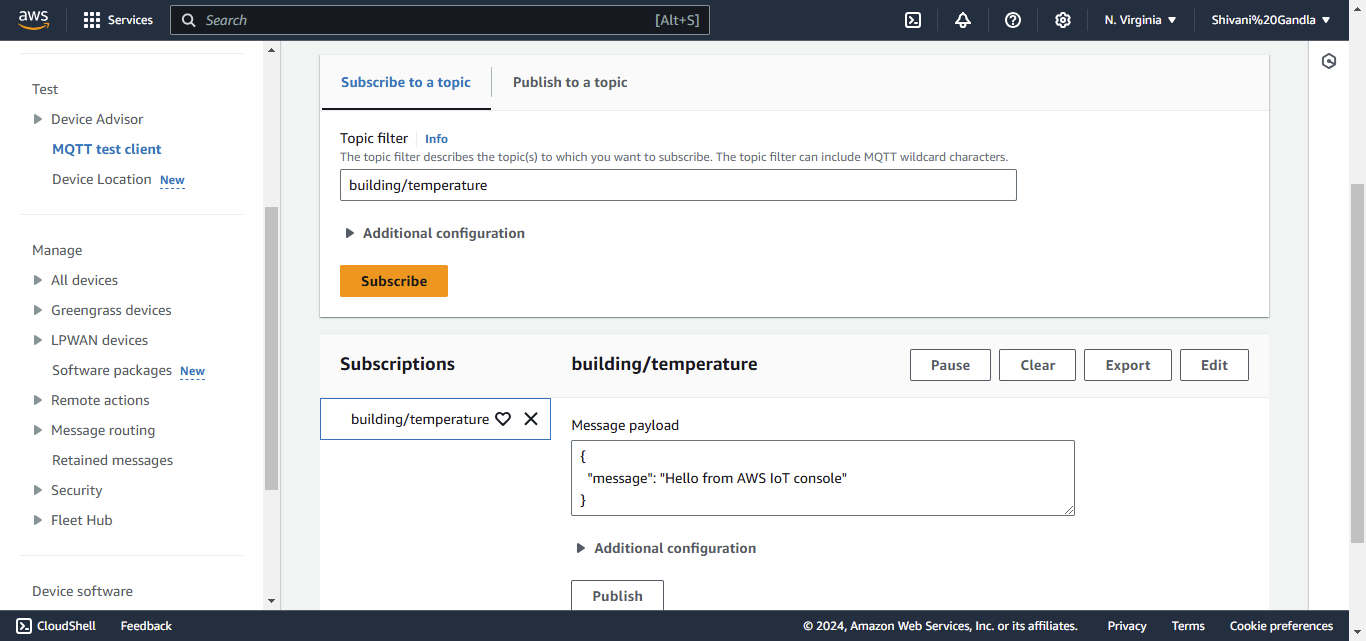
1. Go to **Test > MQTT Test Client** in IoT Core

* In the left-hand menu, go to **Test > MQTT Test Client**.

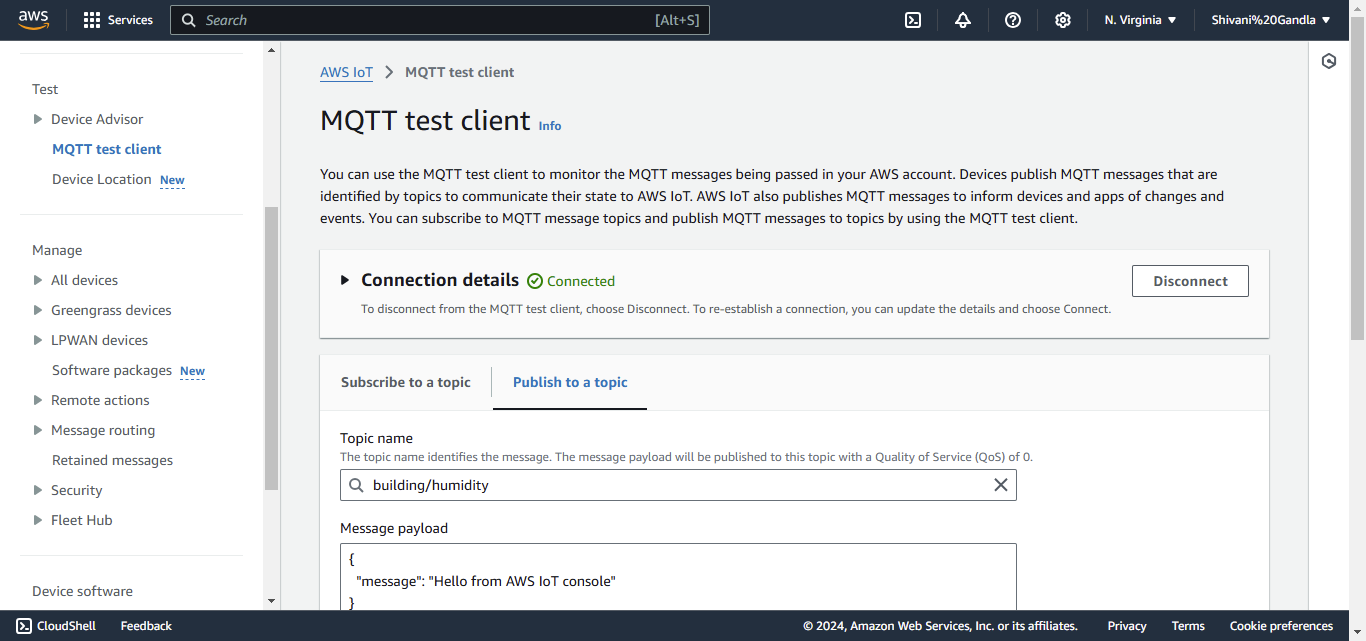
****

Under the MQTT Test Client, you can subscribe to topics to test communication. Examples:

* Subscribe to a topic like building/temperature to receive temperature data.



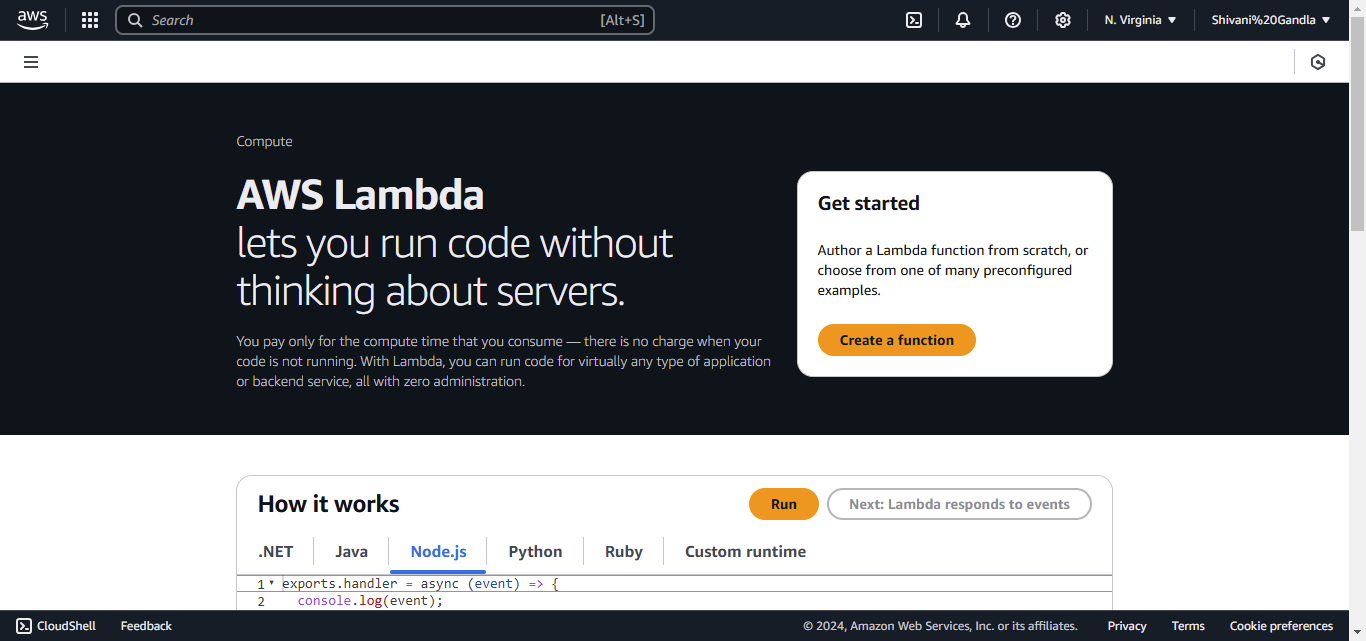
* Publish data to a topic like building/humidity to send humidity information.

****

### ****Step 2: Process IoT Data Using AWS Lambda****

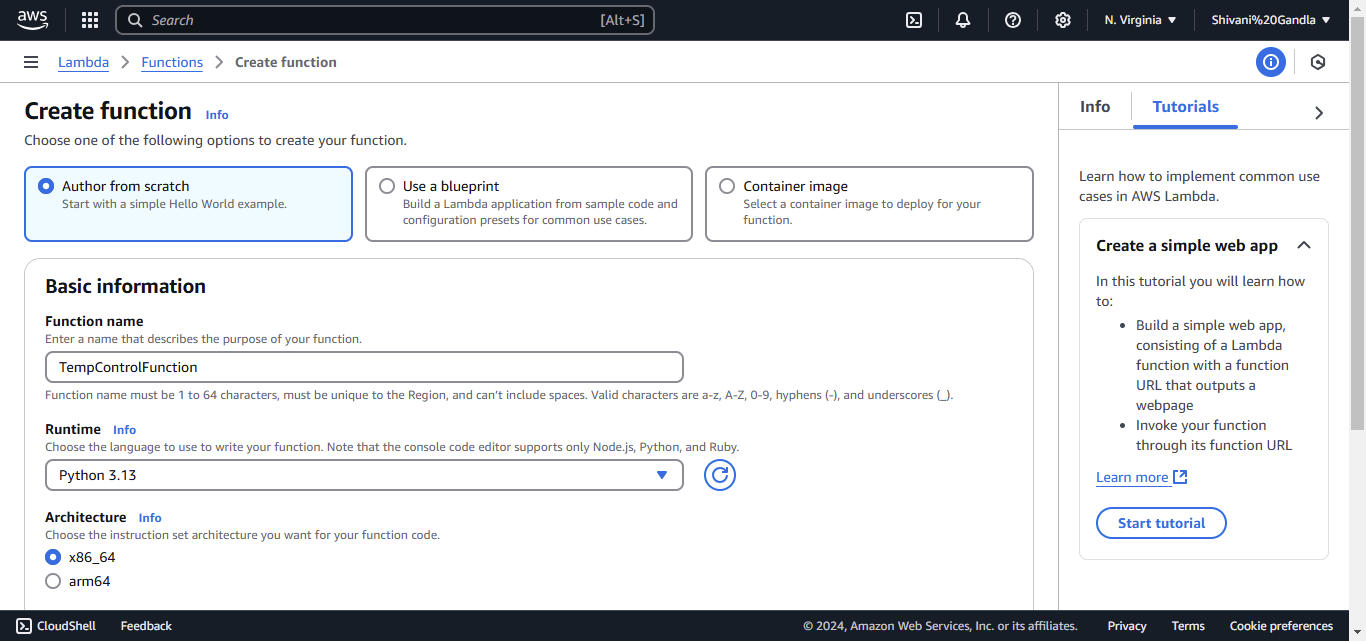
#### ****Create a Lambda Function****

1. Navigate to **AWS Lambda** and click **Create Function**

****

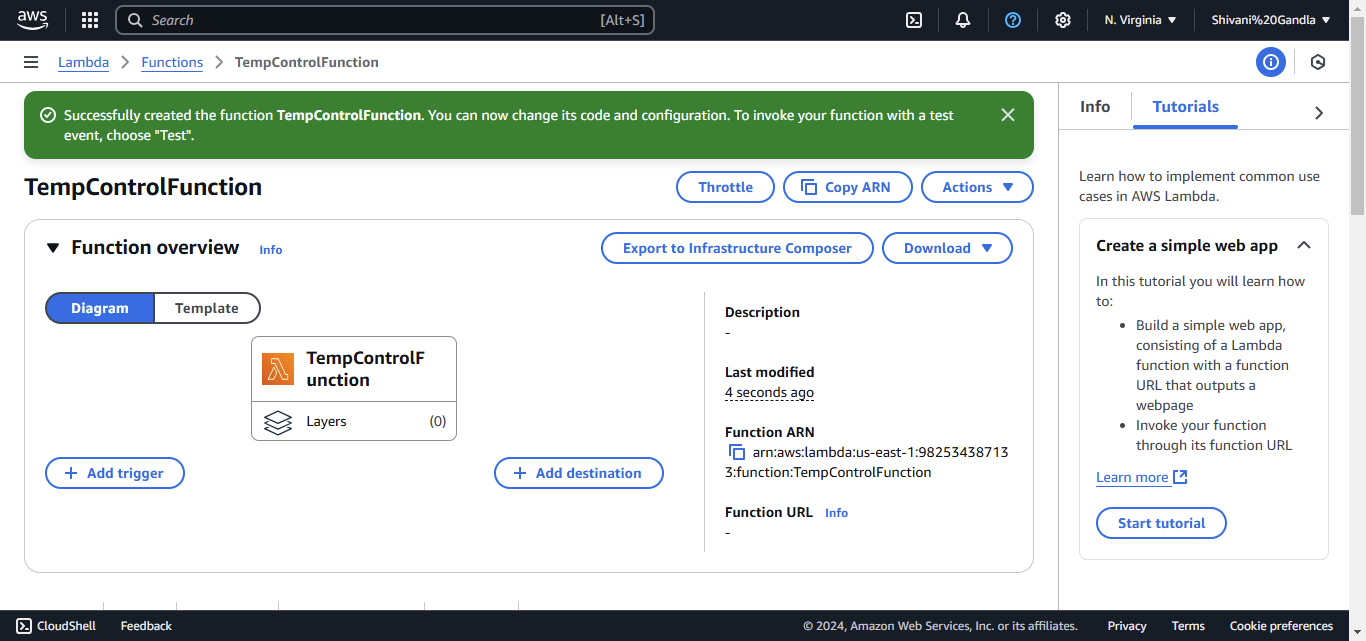
 Choose "Author from scratch."

 Provide a **name** (e.g., TempControlFunction).

****

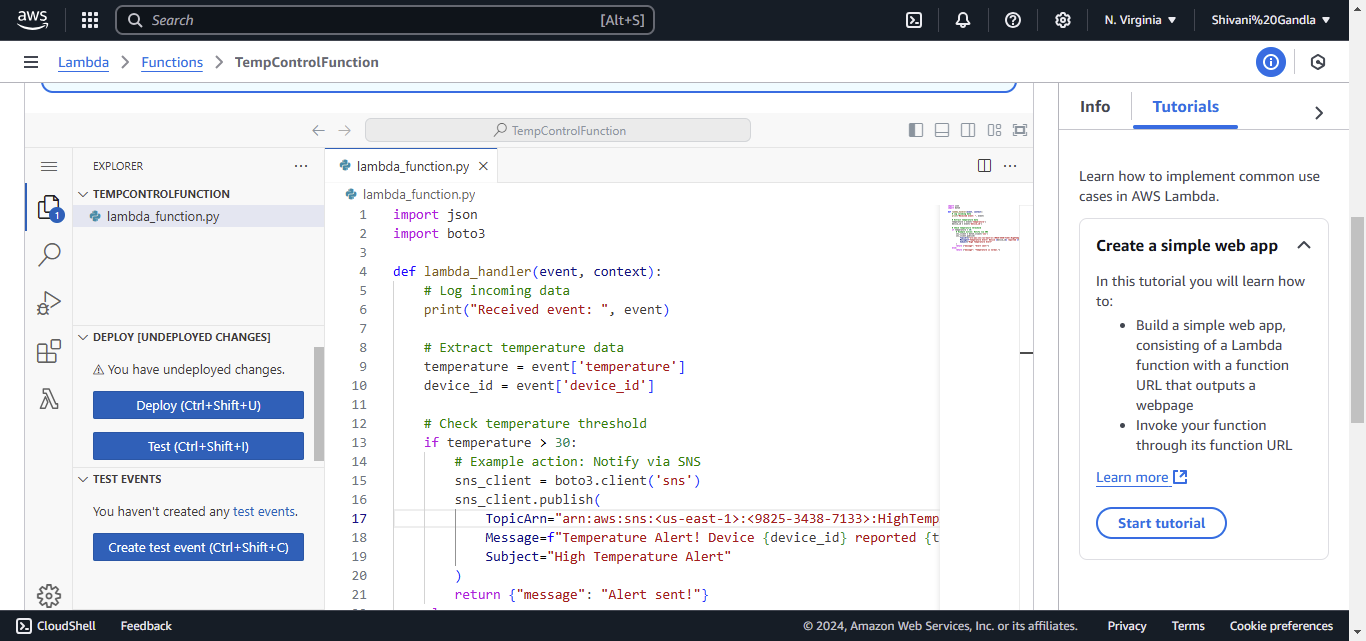
 Leave the **Execution role** option as **Create a new role with basic Lambda permissions** (or choose an existing role if it has required permissions).

 Click **Create function**.

**** ****

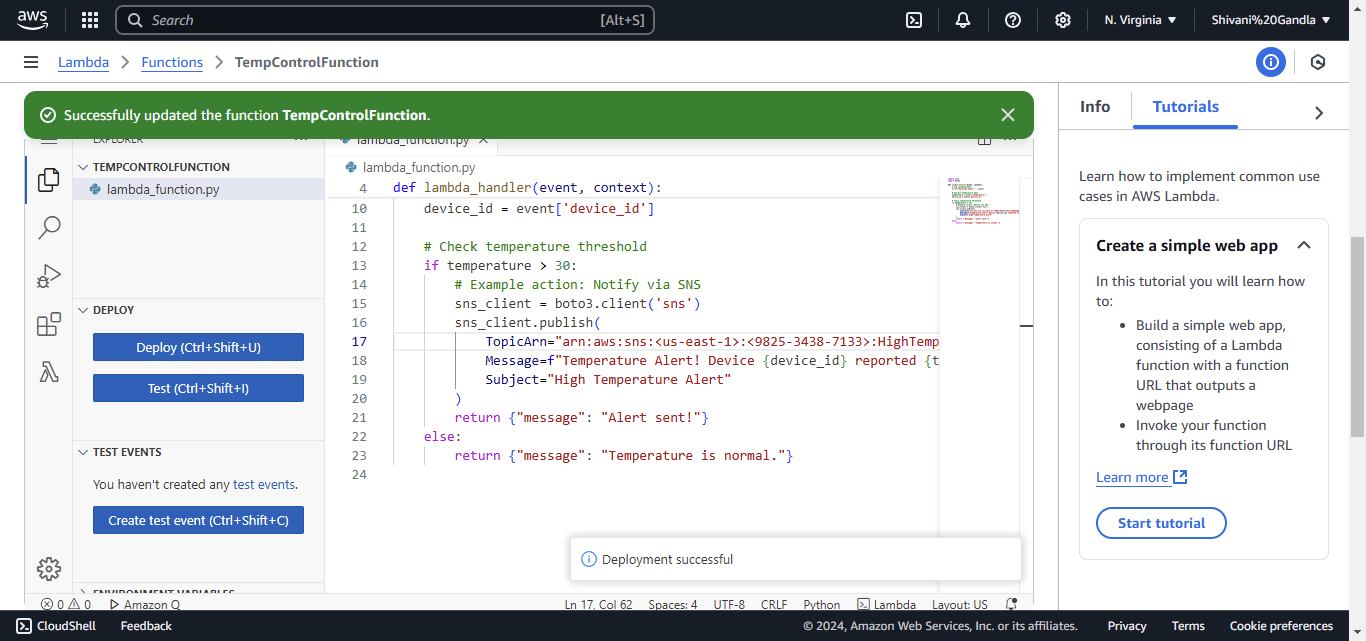
### ****Write the Function Code****

1. **Edit Function Code:**
   * Scroll down to the **Code source** section.



**Deploy the Code:**

* Click **Deploy** to save and deploy your function.



### ****Attach IoT Rule to Lambda****

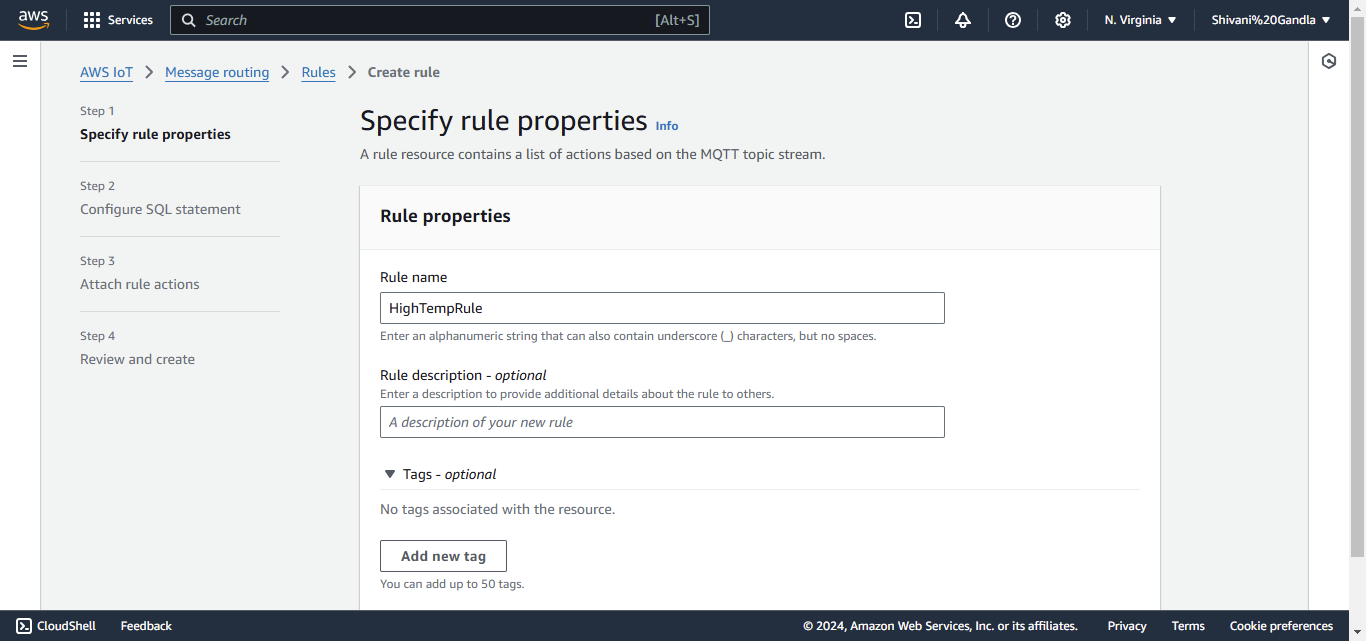
1. **Access IoT Core Rules:**
   * Navigate to **IoT Core** from the list of AWS services.



*  In the left-hand menu, click **Act > Rules**.

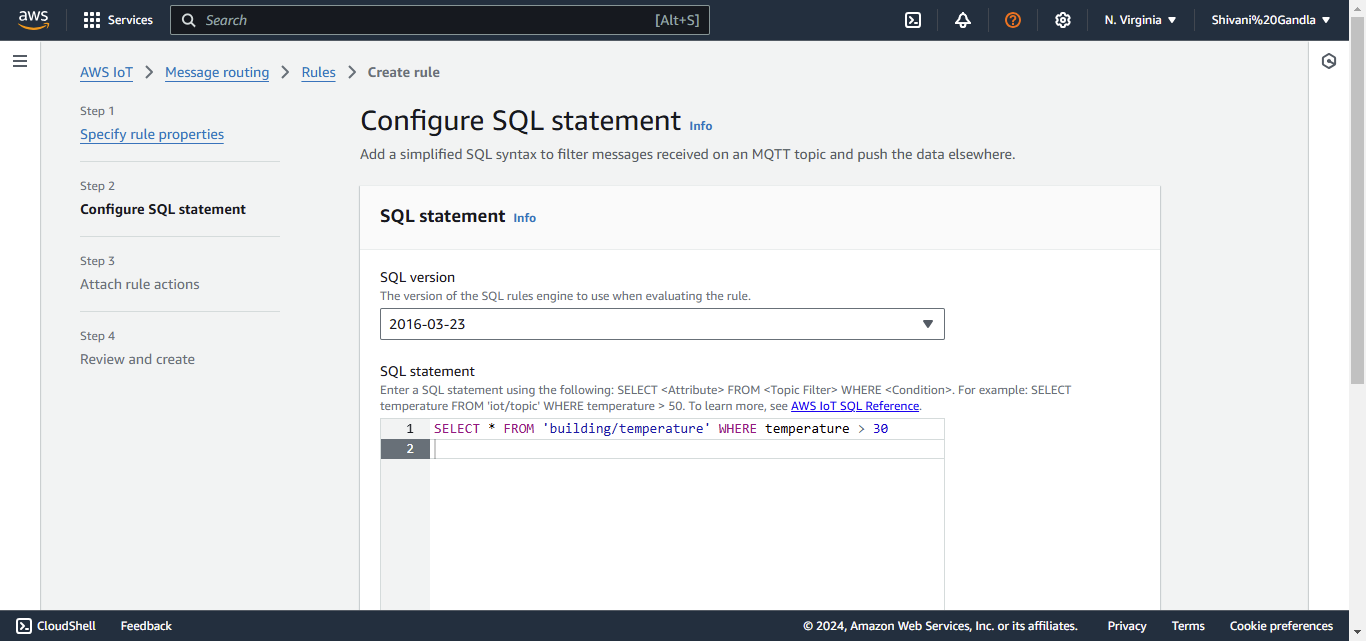
 **Create a New Rule:**

* Click **Create a rule**.
* Enter a **Rule name** (e.g., HighTempRule).



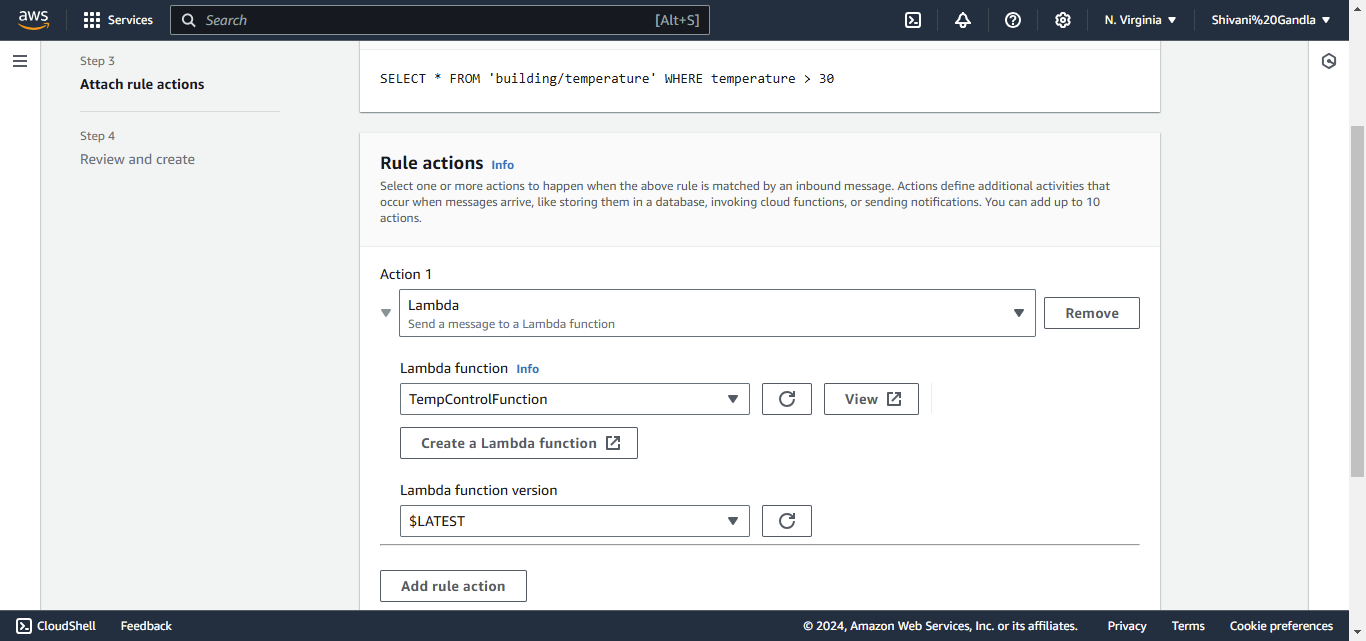
**Define Rule Query:**

* In the **Rule query statement** field, enter the SQL query:



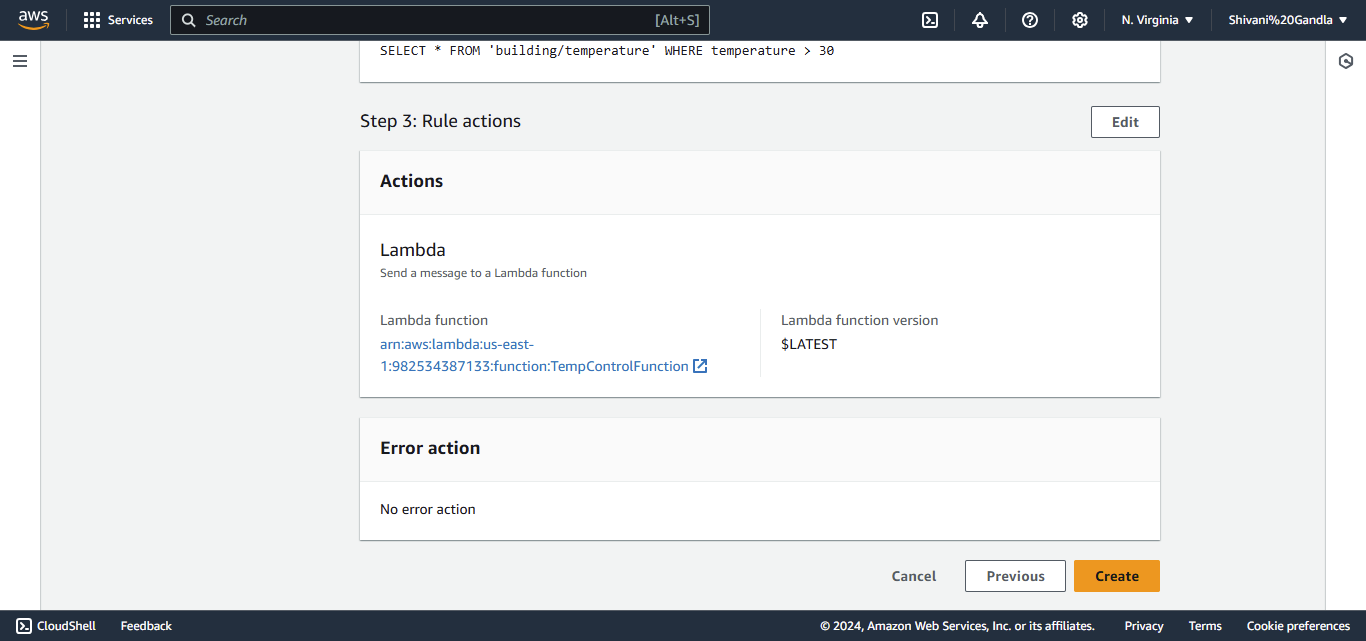
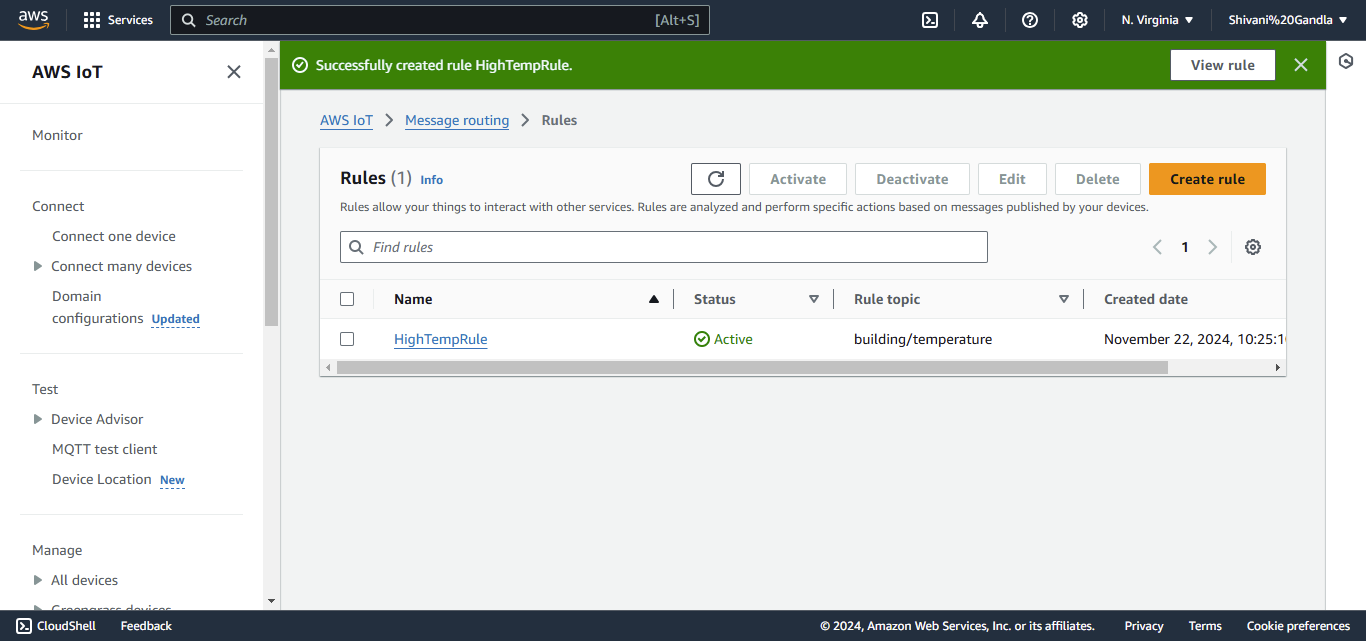
**Add an Action to Invoke Lambda:**

* Under **Set one or more actions**, click **Add action**.
* Select **Send a message to a Lambda function**.



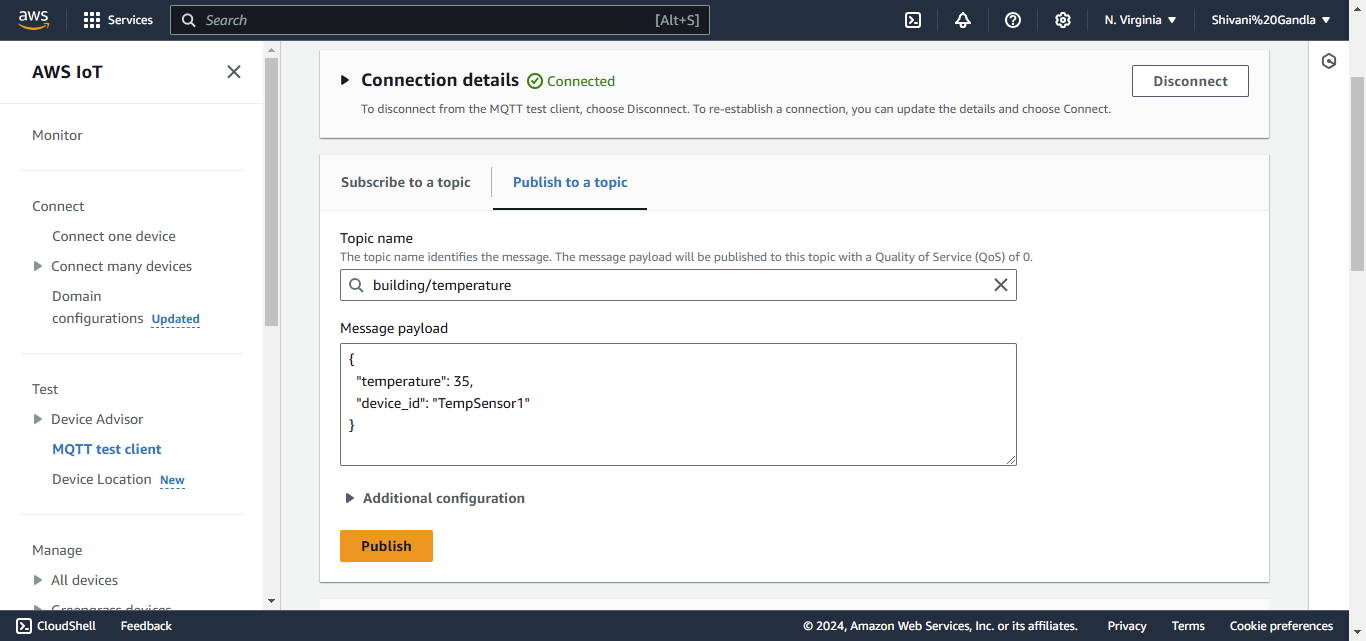
**Complete Rule Creation:**

* Review your rule and actions.
* Click **Create rule** to save it.

### ****Test the Setup****

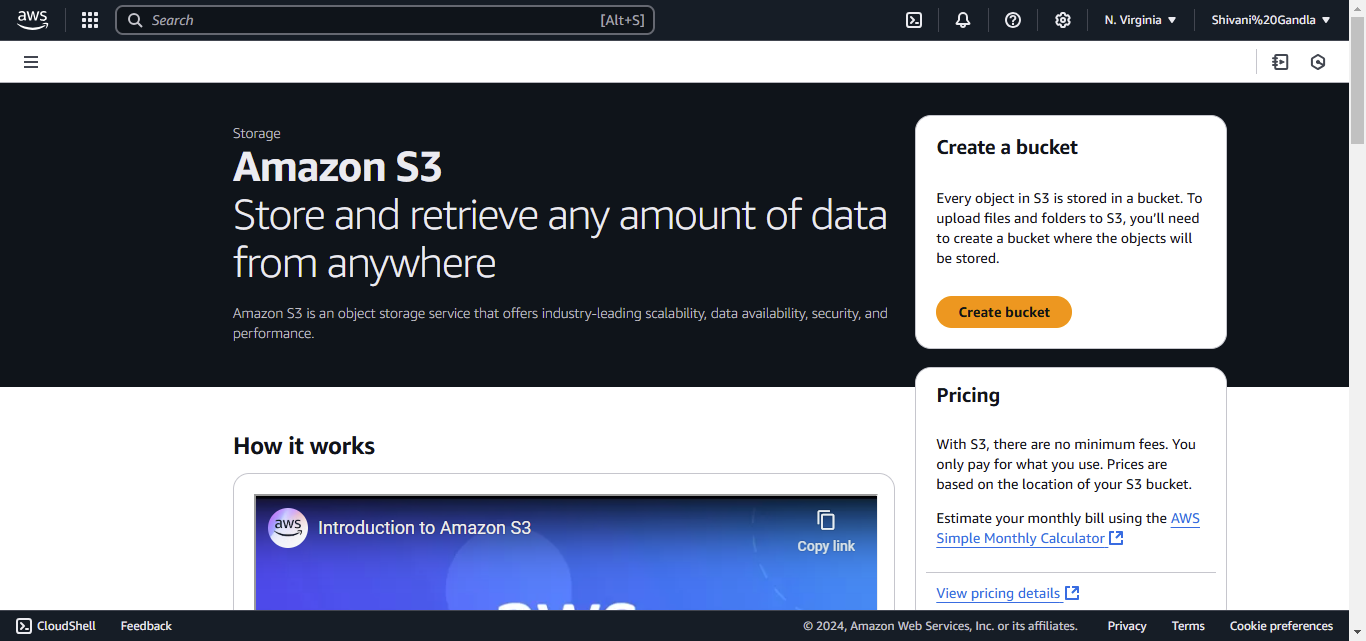
1. **Simulate Data in MQTT Test Client:**
   * Navigate to **Test > MQTT Test Client** in IoT Core.

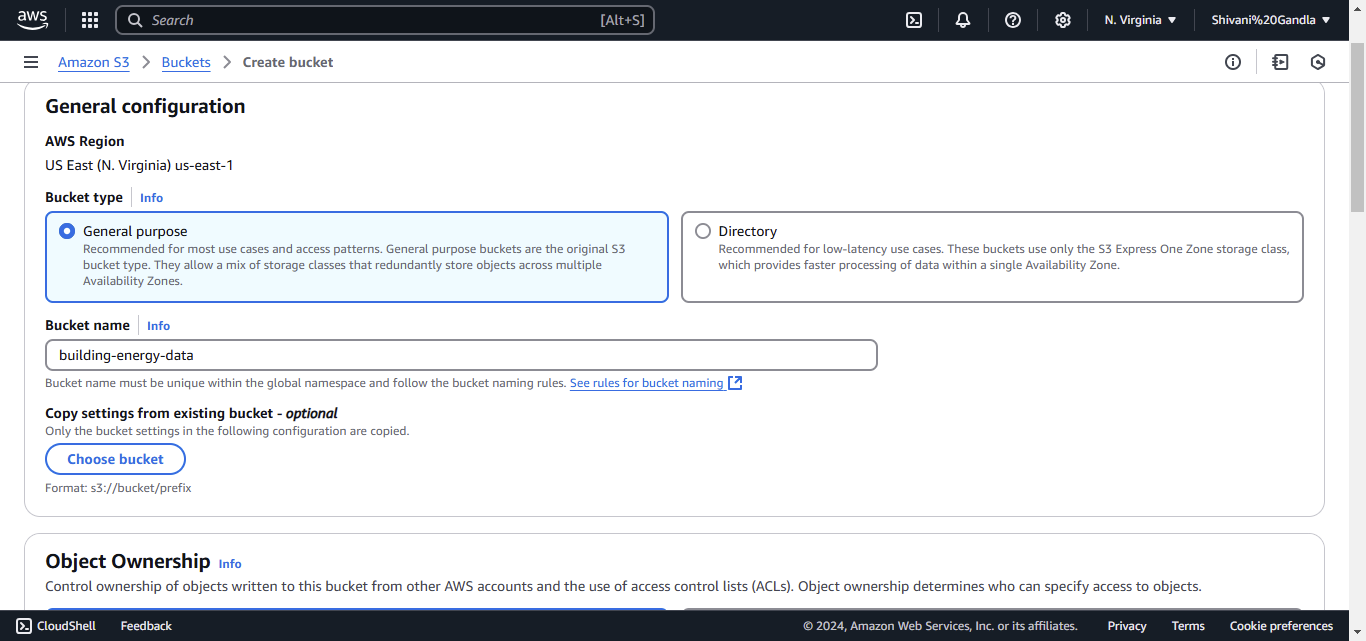


### ****Step 3: Store and Analyze Data****

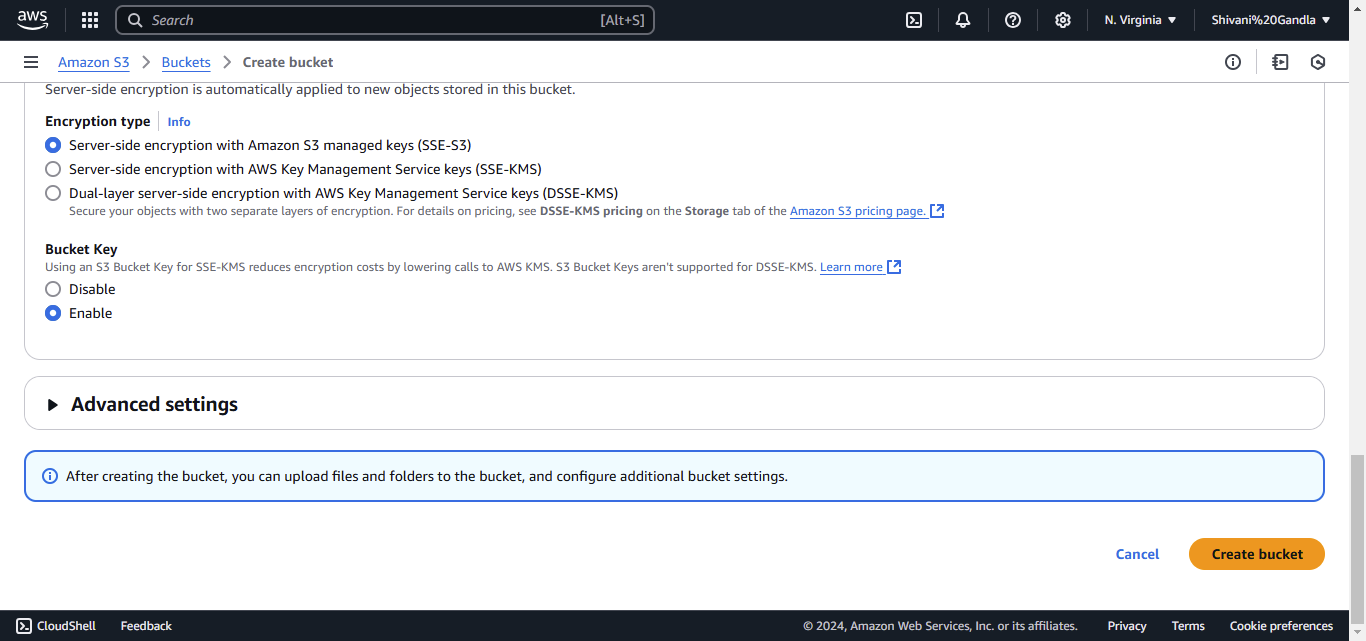
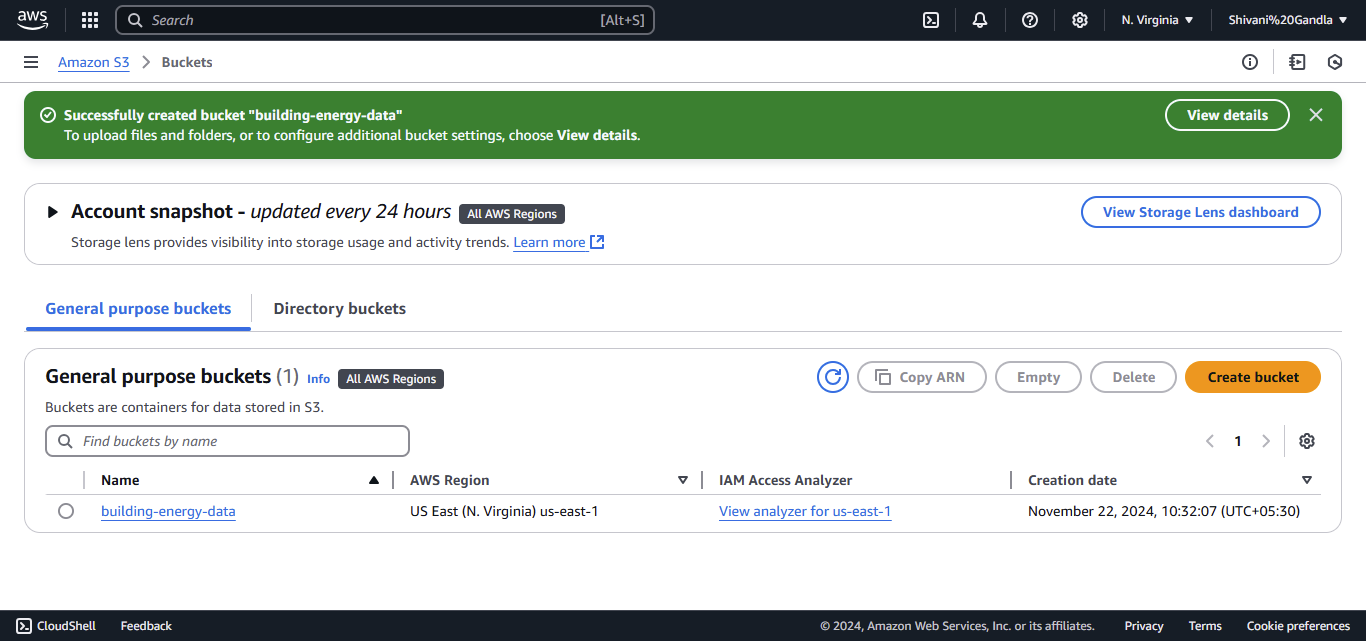
**Create a New Bucket:**

* Click **Create bucket**.
* Enter a **Bucket name** (e.g., building-energy-data).





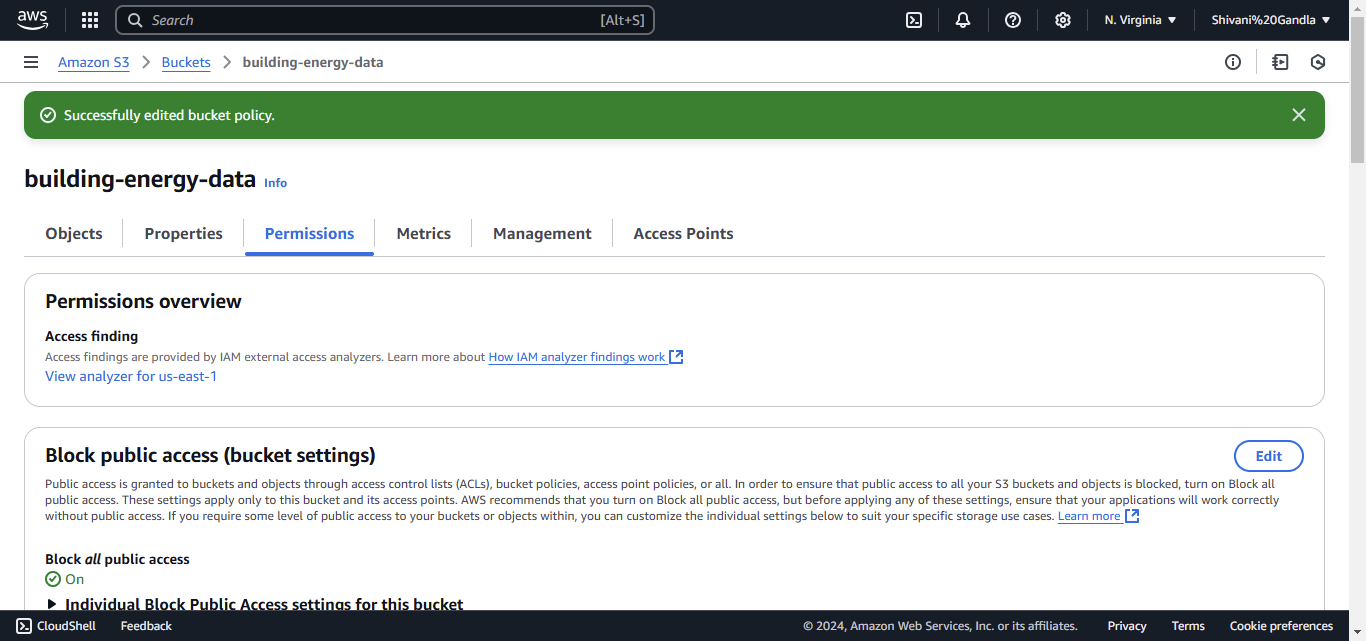
Click on create S3 bucket

**Set Bucket Permissions:**

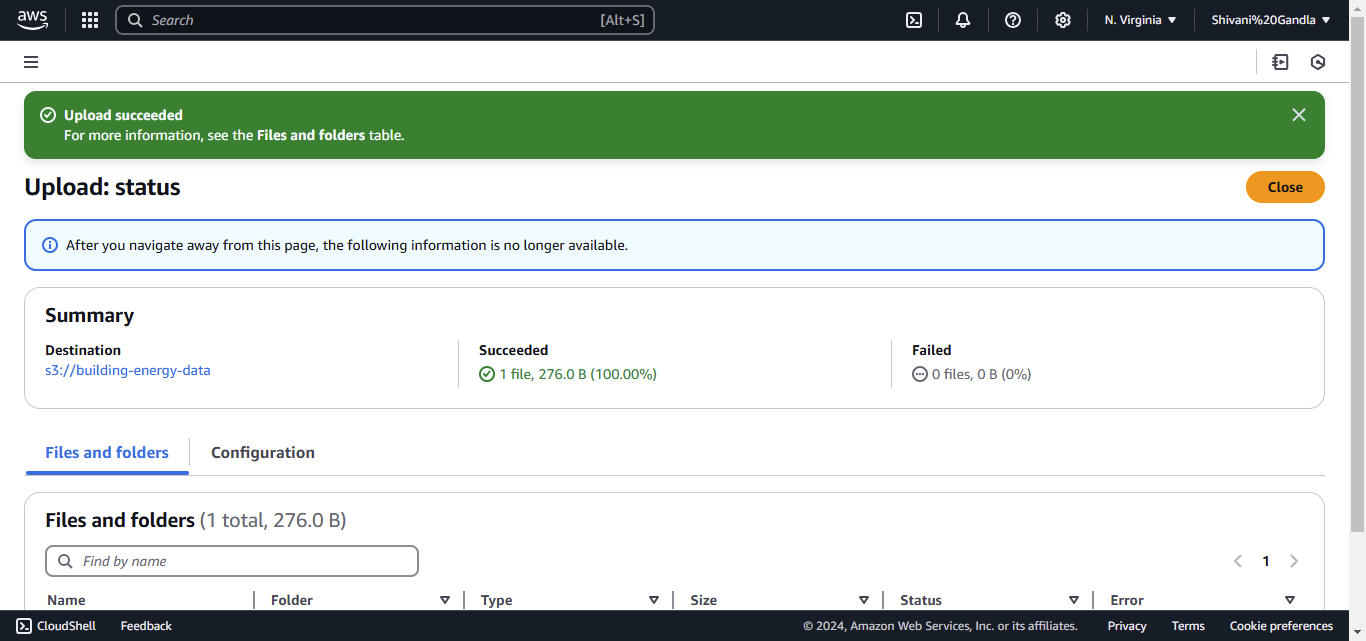
* In the **Bucket settings for Block Public Access**, ensure public access is blocked (default setting).
* Under **Bucket Policy** (after bucket creation):
  + Go to your bucket, click **Permissions > Bucket Policy**, and add a policy to allow access from AWS Lambda or IoT (if required). An example policy:



Click **Save changes.** 

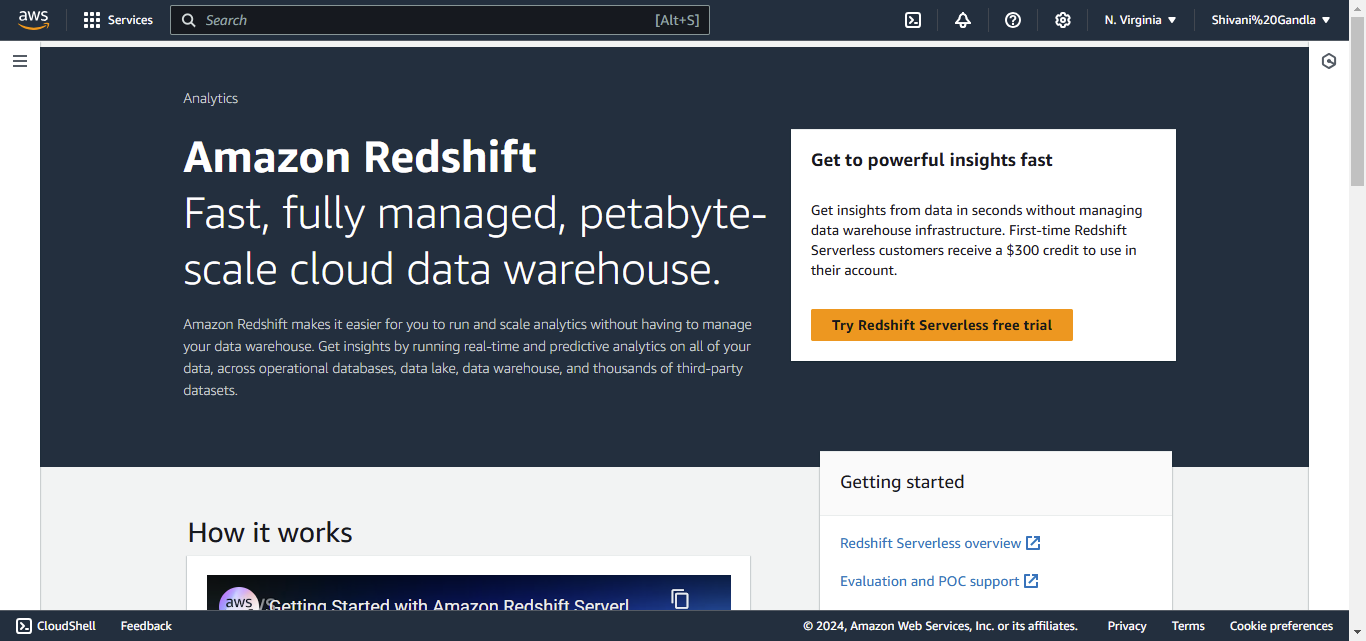
**Upload Test Data:**

* Navigate to your bucket.
* Click **Upload** and upload a sample file (e.g., device\_data.csv).



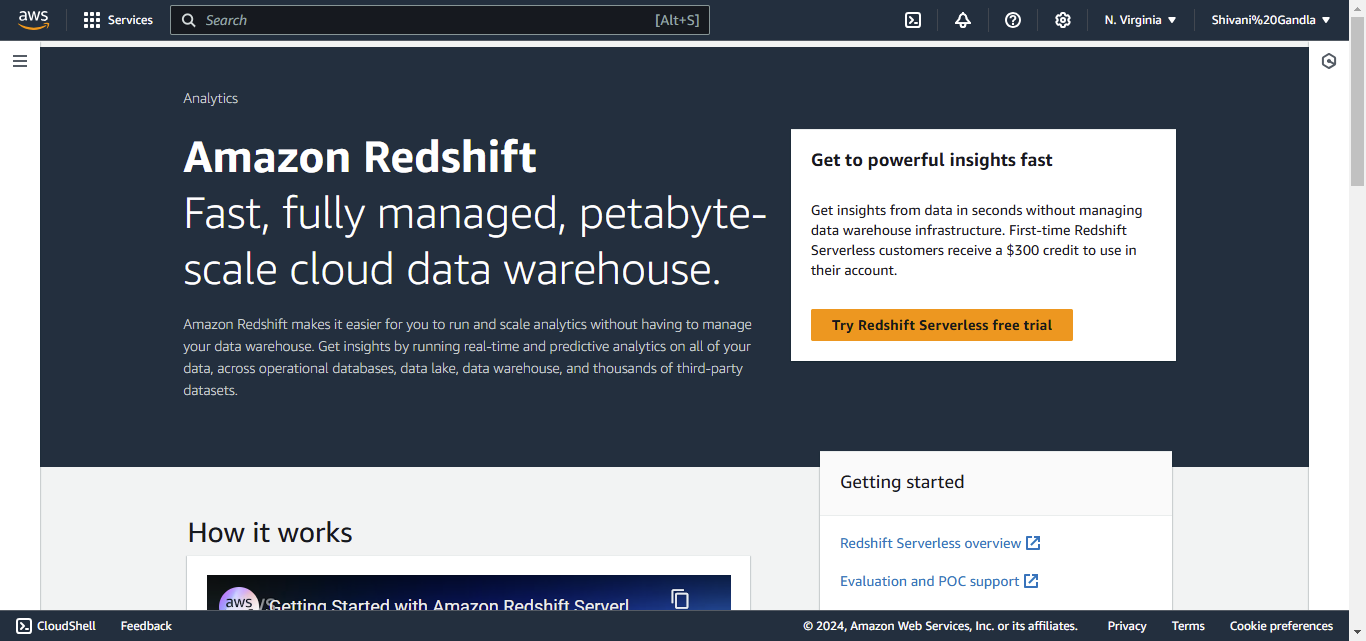
### ****Set Up Amazon Redshift****

* **Access Amazon Redshift:**
  + In the AWS Management Console, search for **"Redshift"** and select it.

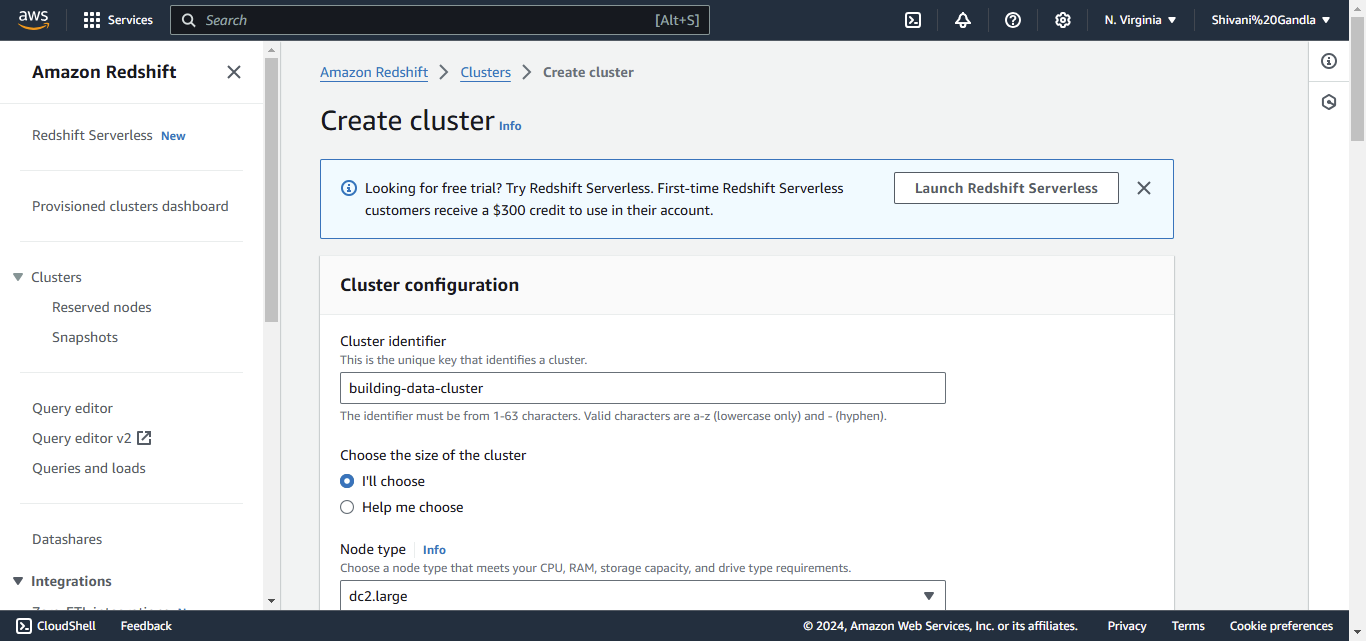


**Create a New Cluster:**

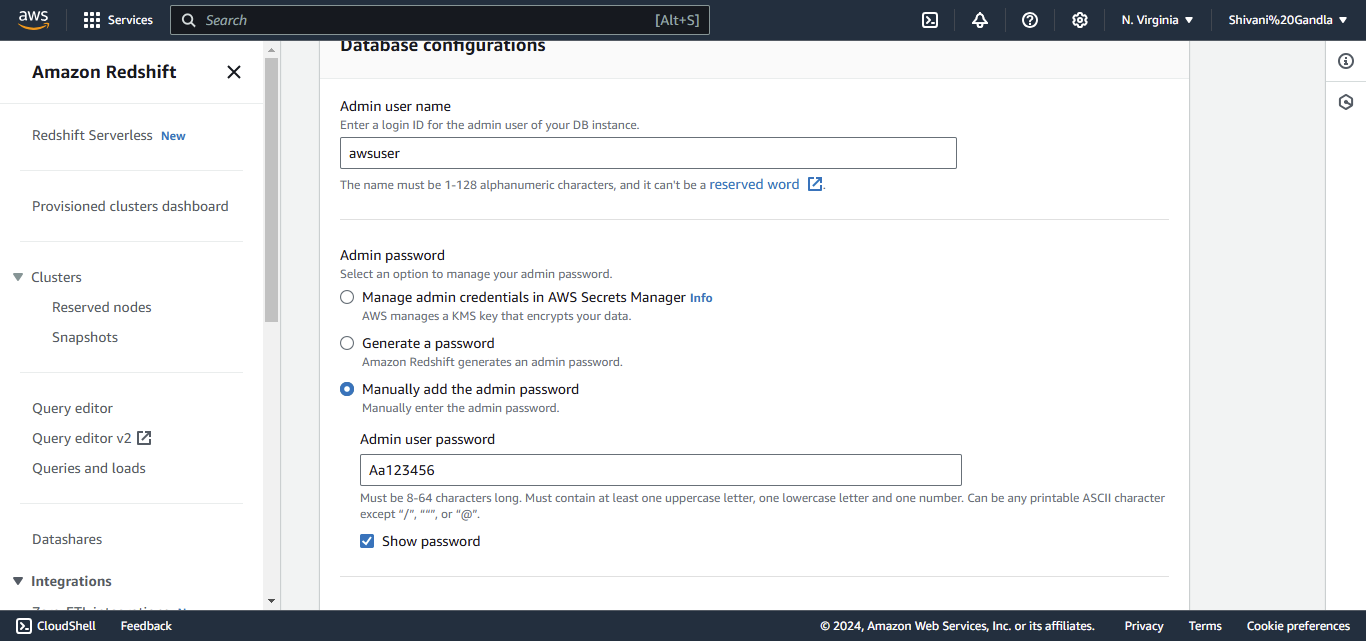
* Click **Create cluster**.



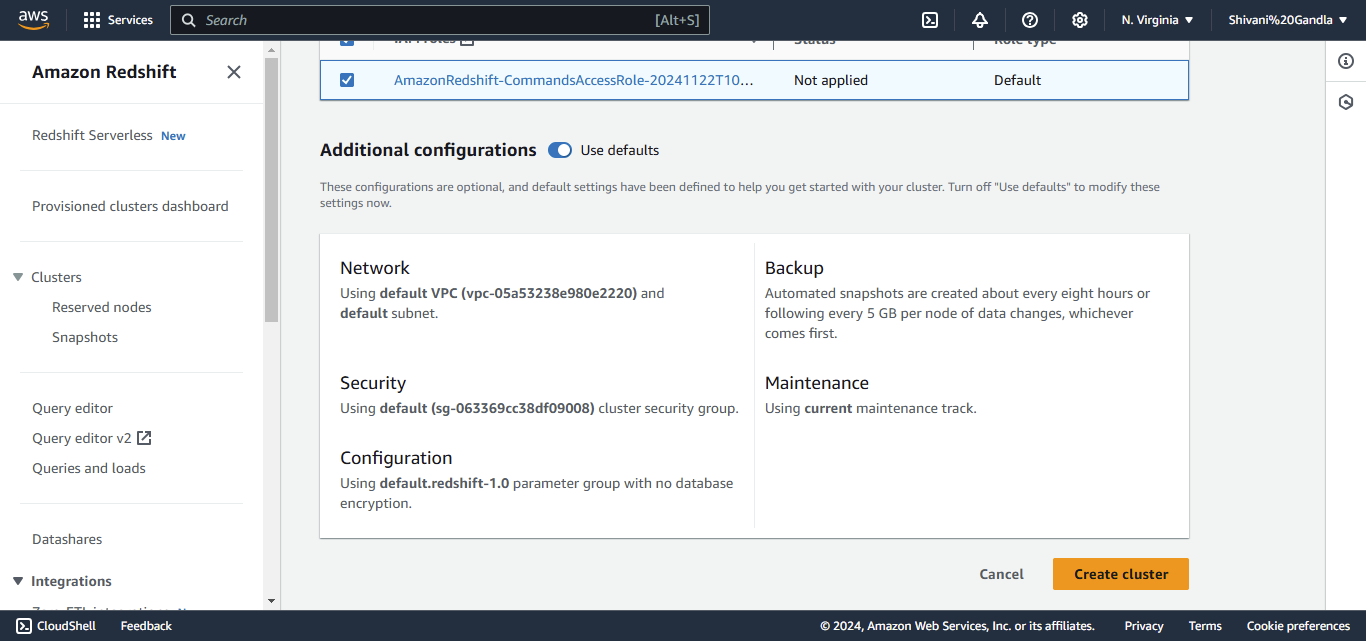
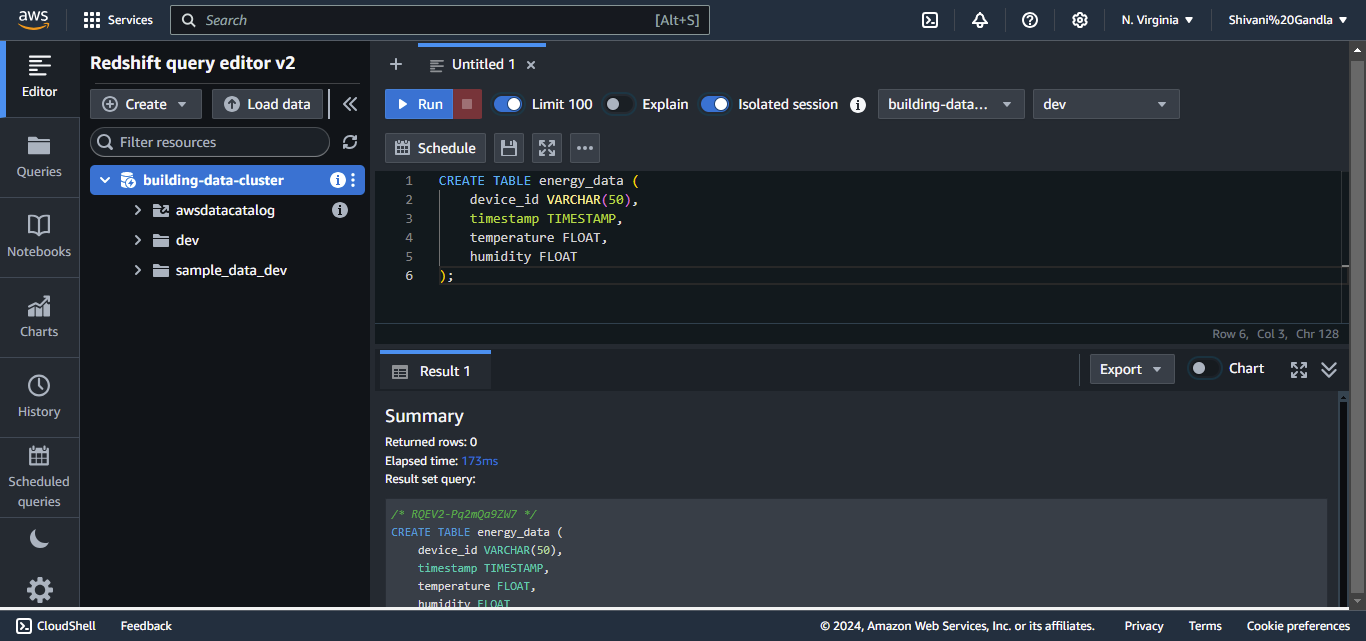
 **Cluster identifier:** Enter a name (e.g., building-data-cluster).

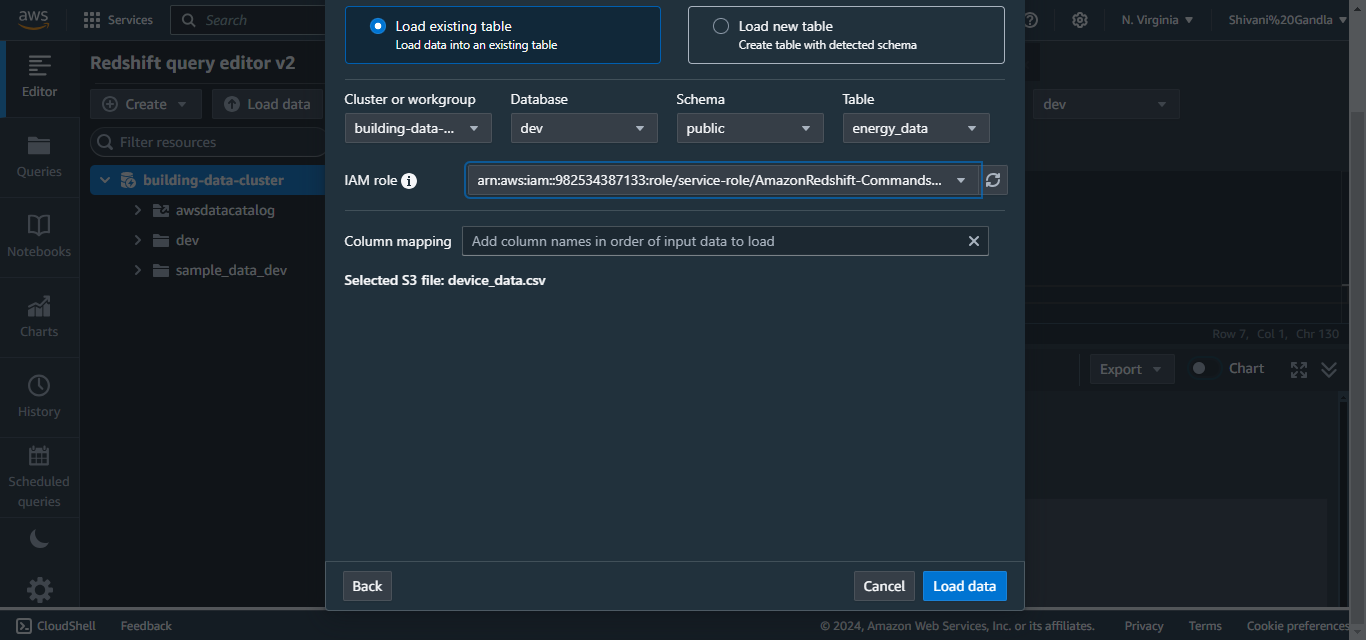


* **Master user credentials:** Set a username and password.



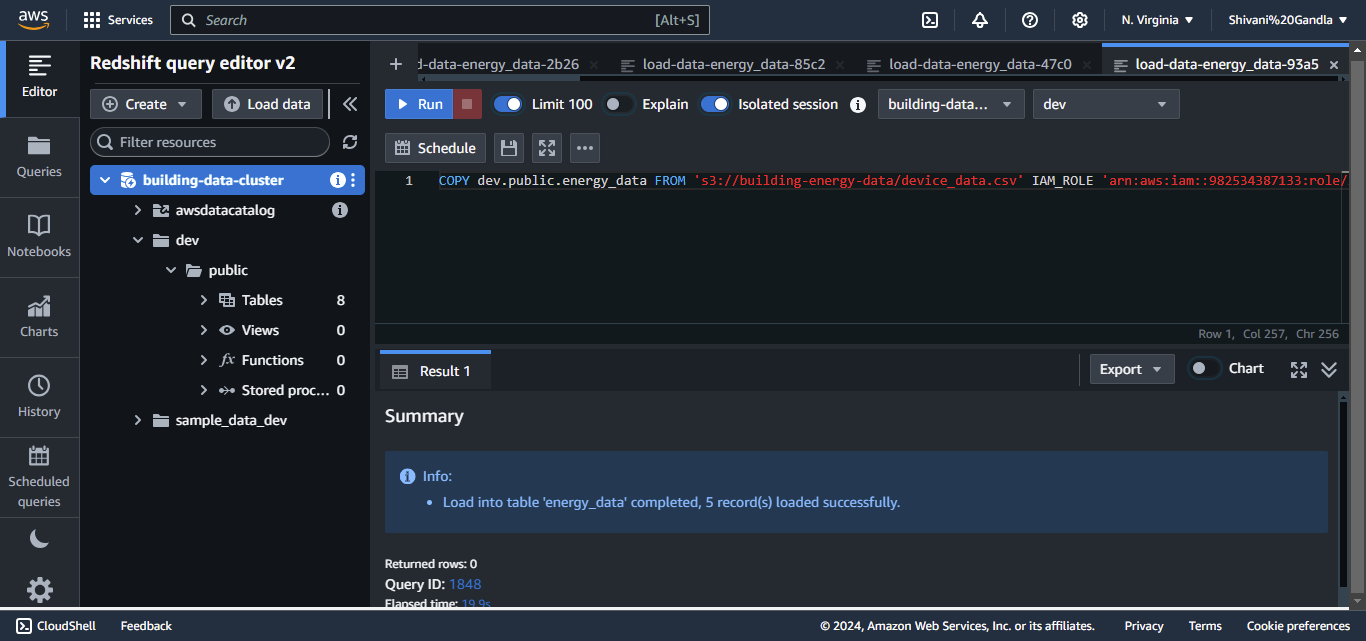
* Click **Create cluster** and wait for the cluster to be provisioned.



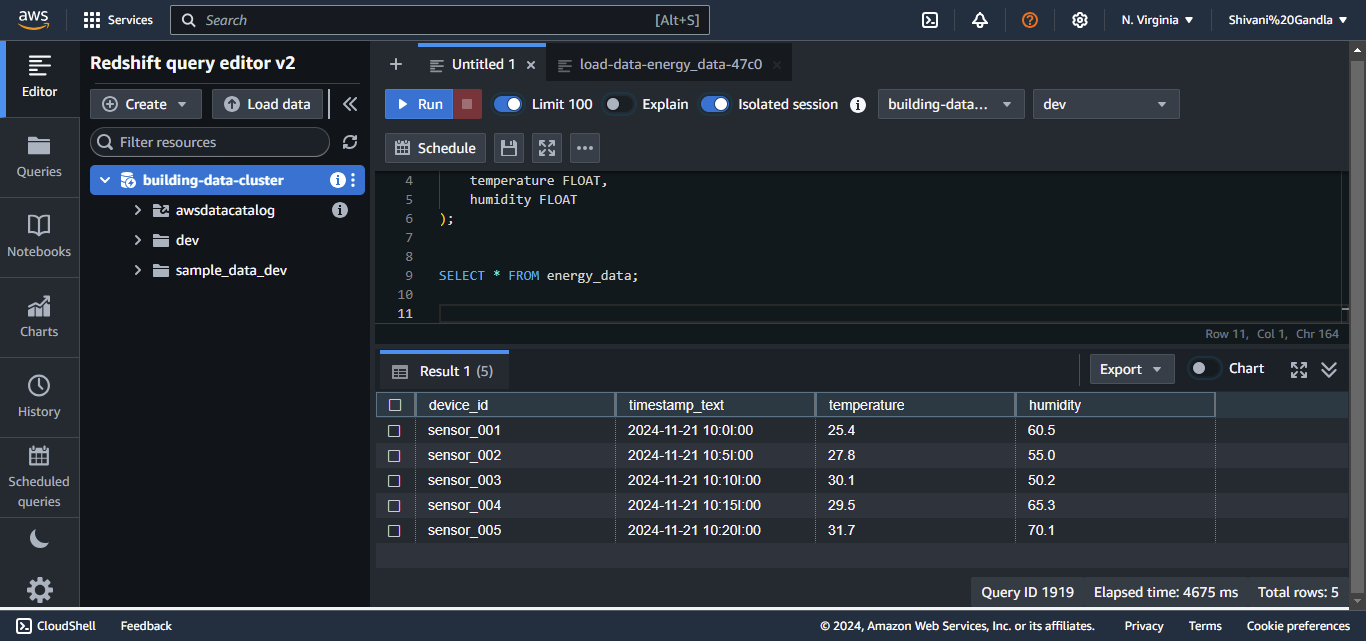
**Run the COPY Command in Redshift:**

* Go back to the Query Editor in Redshift.



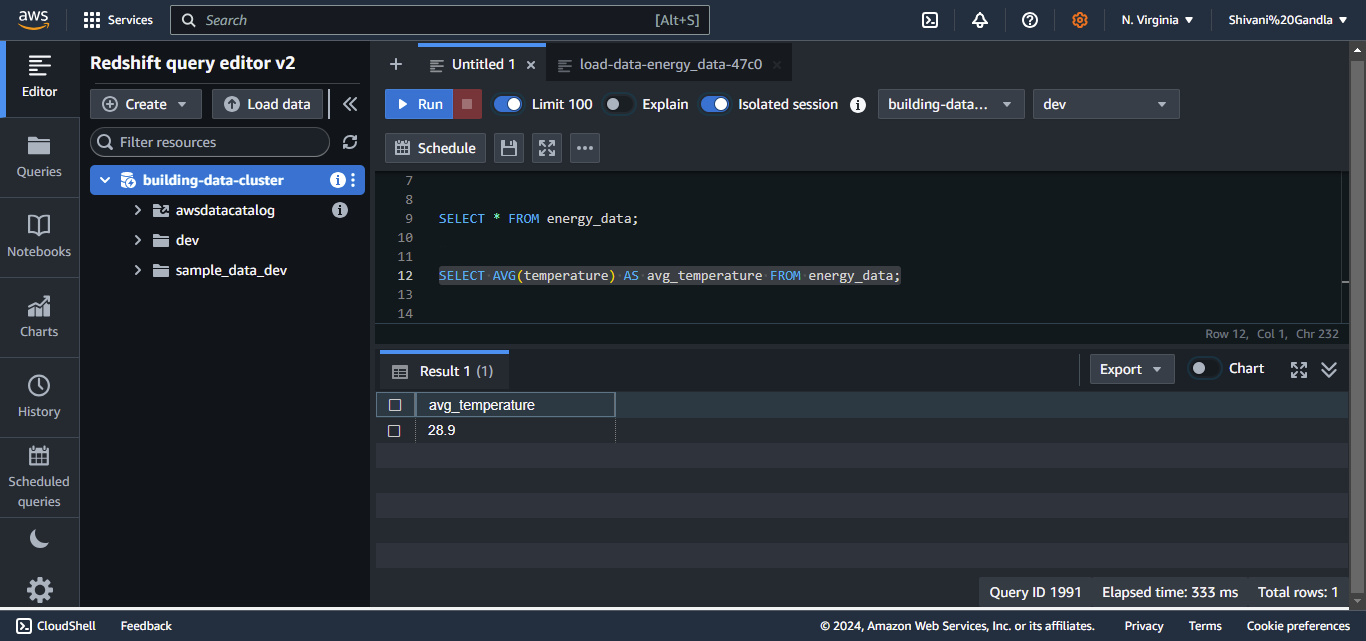
**Verify the Data Load:**

* Run the following query in Redshift to verify the data:

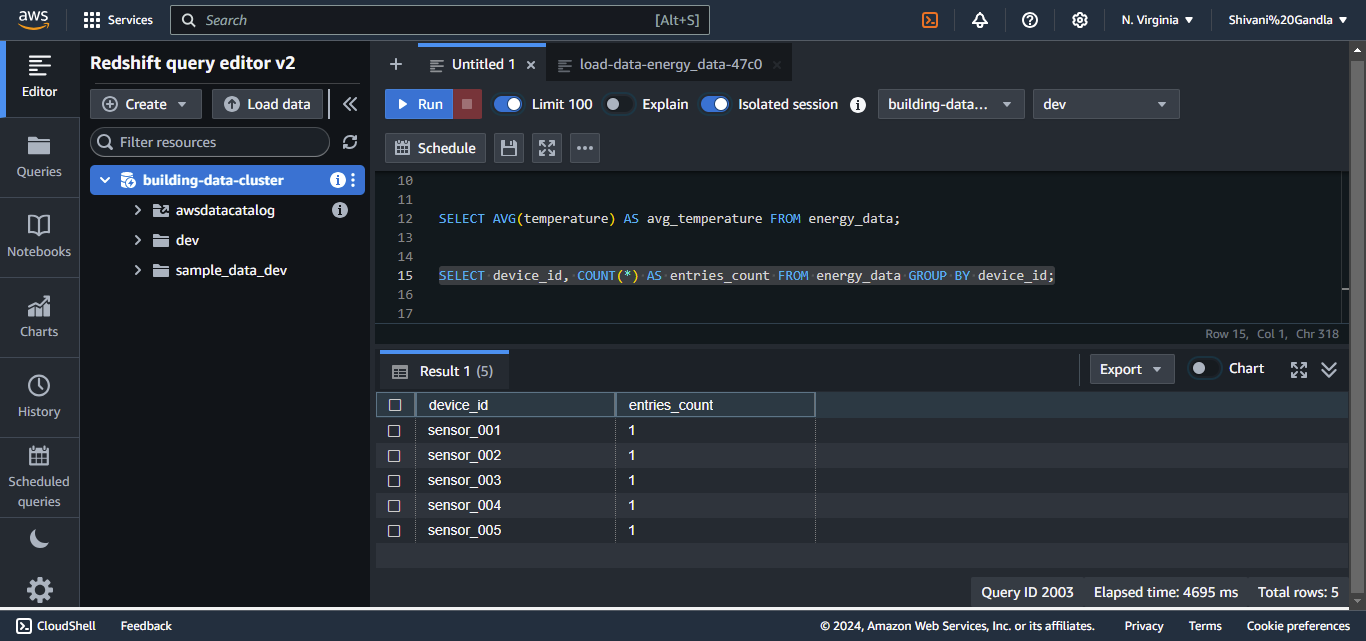


### ****Analyze Data in Redshift****

1. **Run Queries:**
   * Example: Get average temperature:



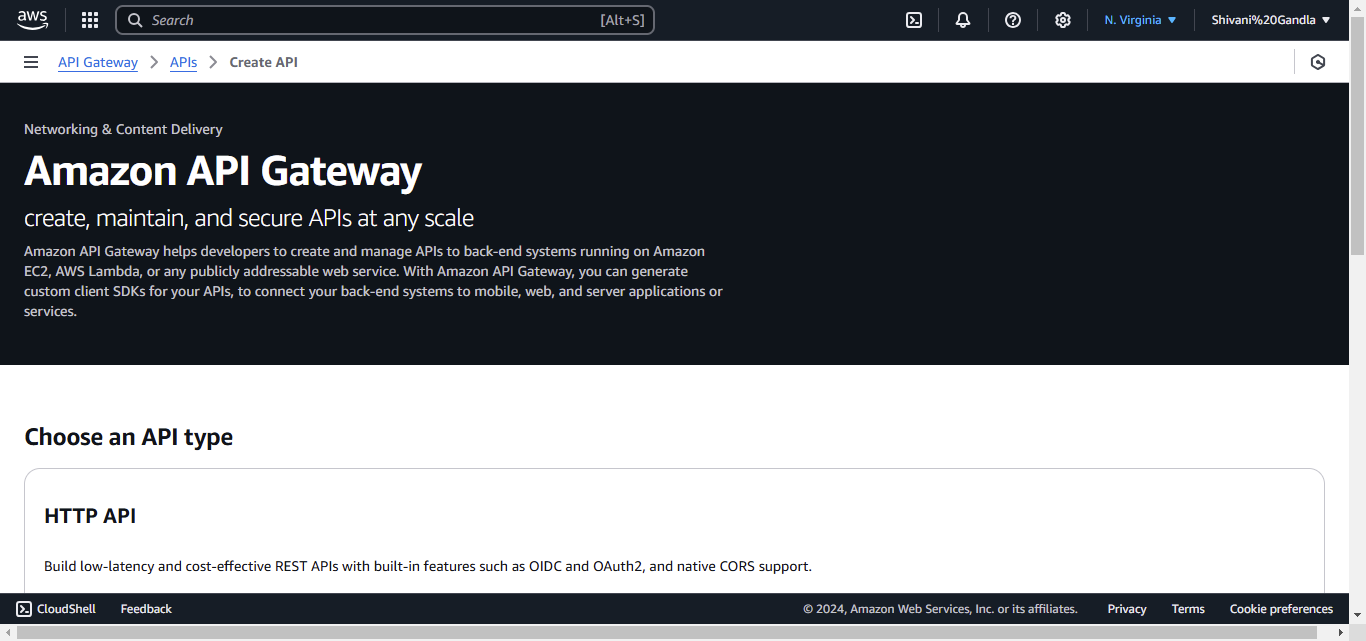
* Example: Count entries by device:



**Step 4: Create a Web-Based Dashboard**

#### ****1. Set Up API Gateway****

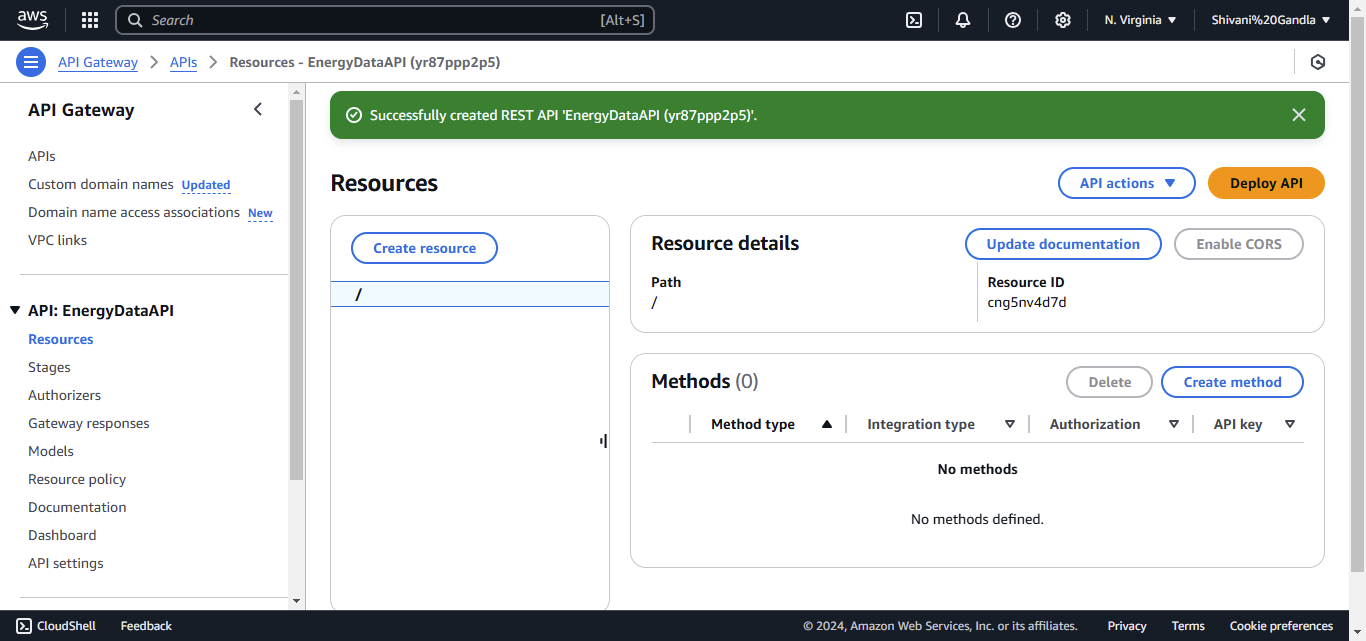
1. Navigate to **API Gateway** and create a REST API.



 Navigate to **API Gateway** from the list of services.

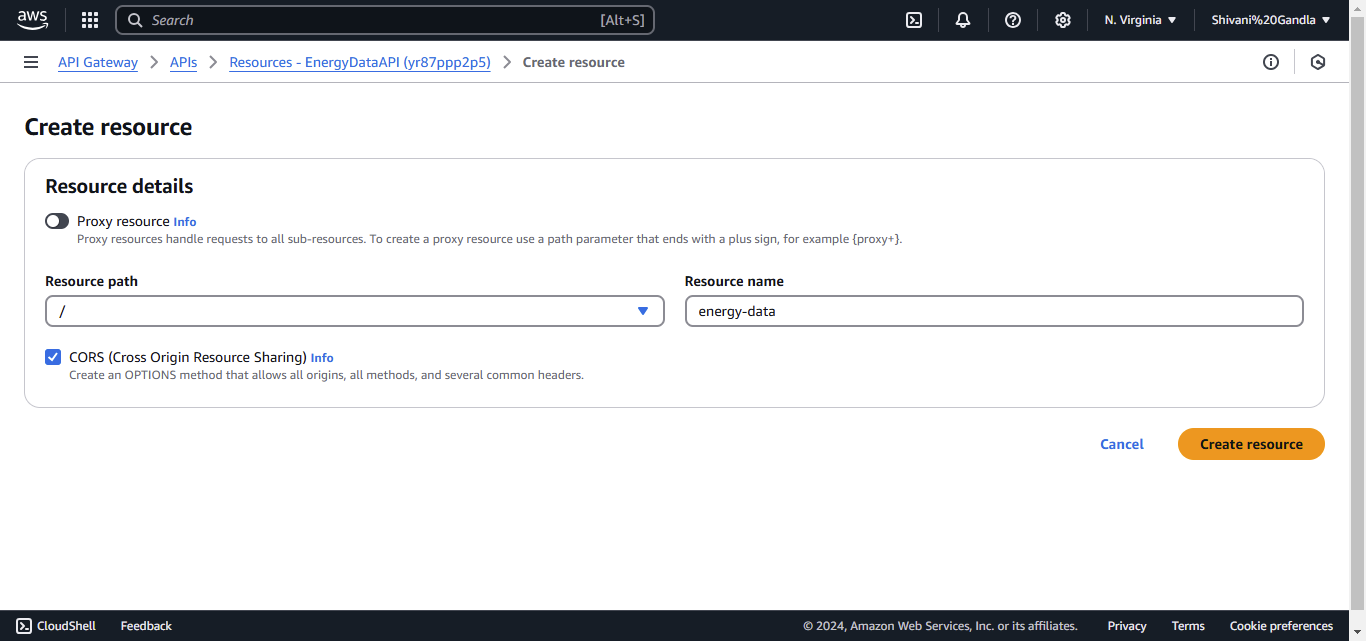
 Click **Create API**, then select **REST API** (Public or Private as per your need).

 Name your API (e.g., EnergyDataAPI) and click **Create API**.

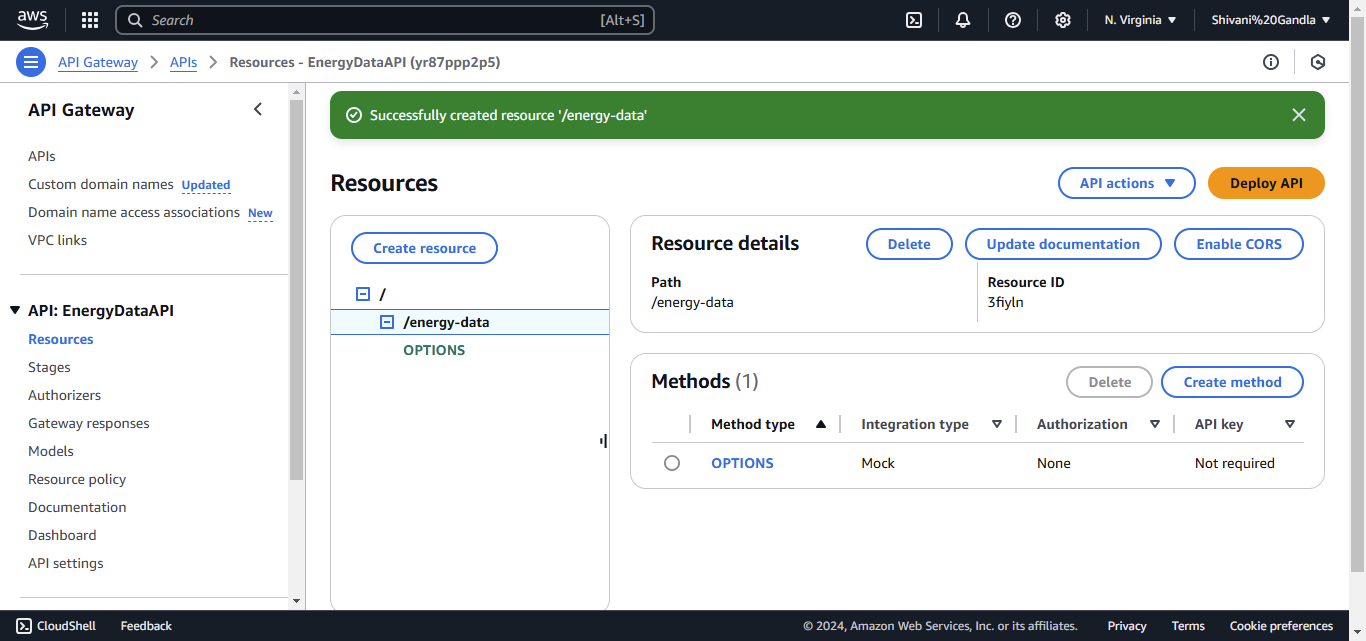


#### ****Add a Resource****

* In the **Resources** section of your API:
* Click **Actions > Create Resource**.

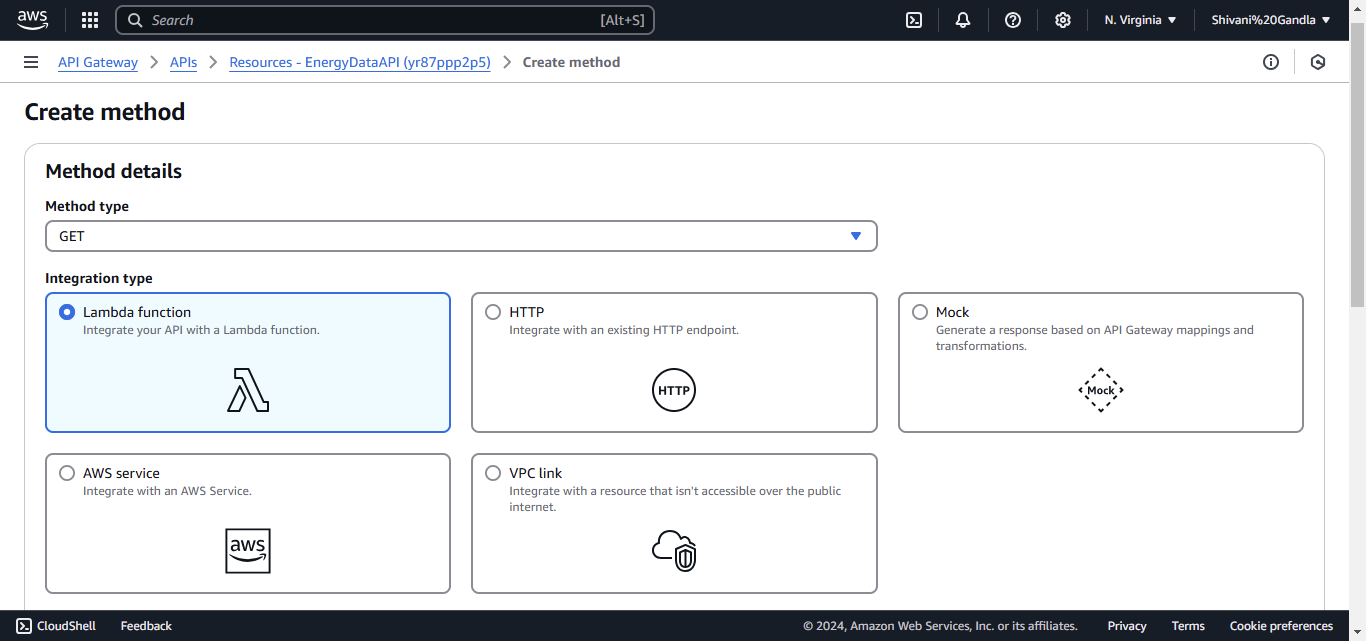


Enable **CORS (Cross-Origin Resource Sharing)** if your frontend and backend are hosted on different domains.



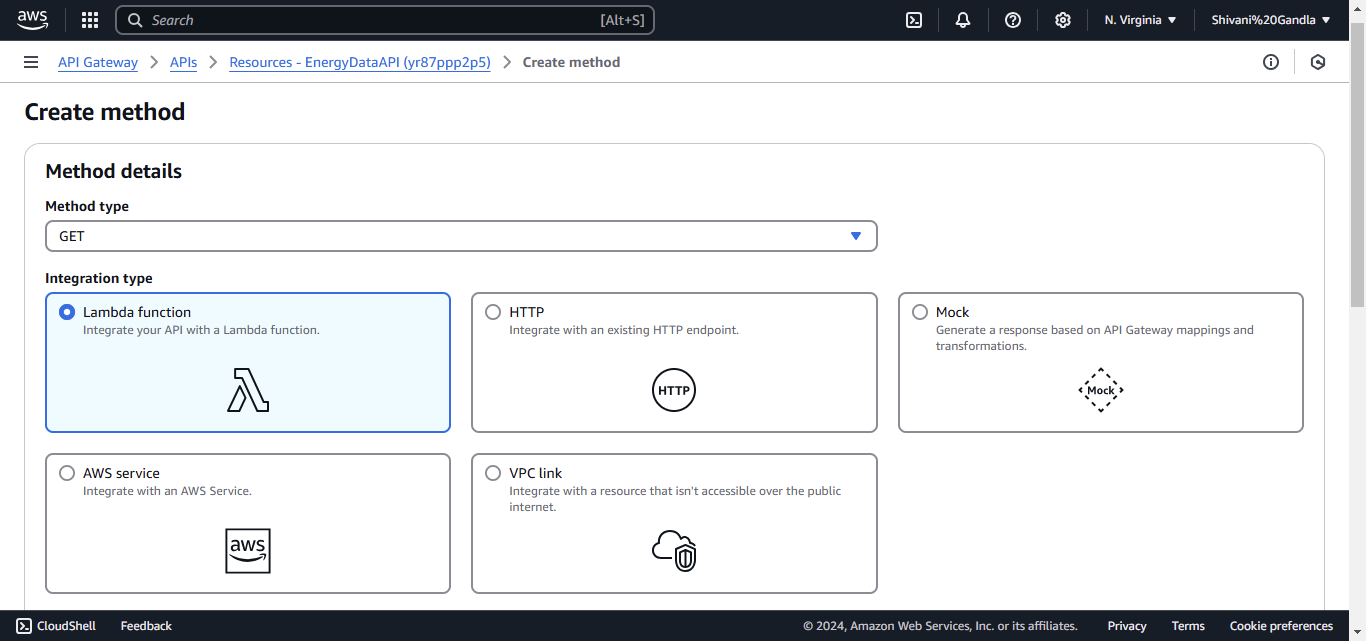
#### ****Add a GET Method****

* Select the newly created resource (/energy-data).
* Click **Actions > Create Method** and choose **GET**.



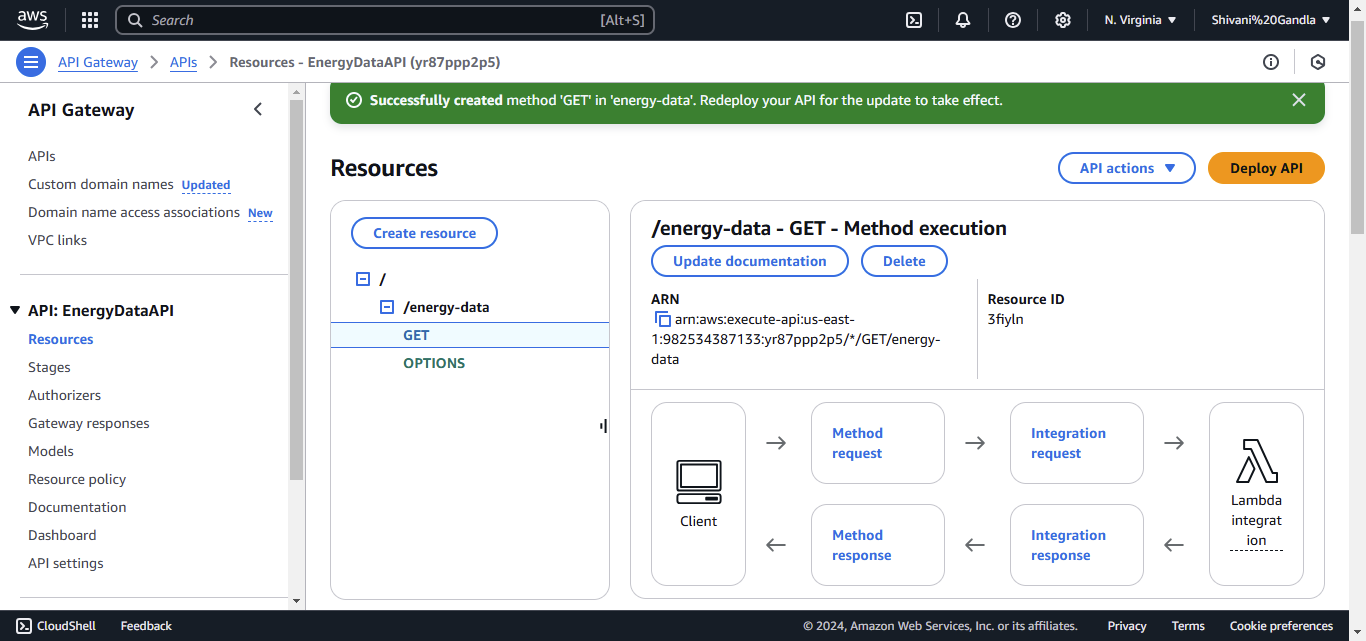
For the **Integration Type**, select one of the following:

* **Lambda Function**: If you want to fetch data via AWS Lambda.
* **HTTP**: If the data is available from a different REST API.
* **AWS Service**: If you are directly fetching data from Amazon Redshift.



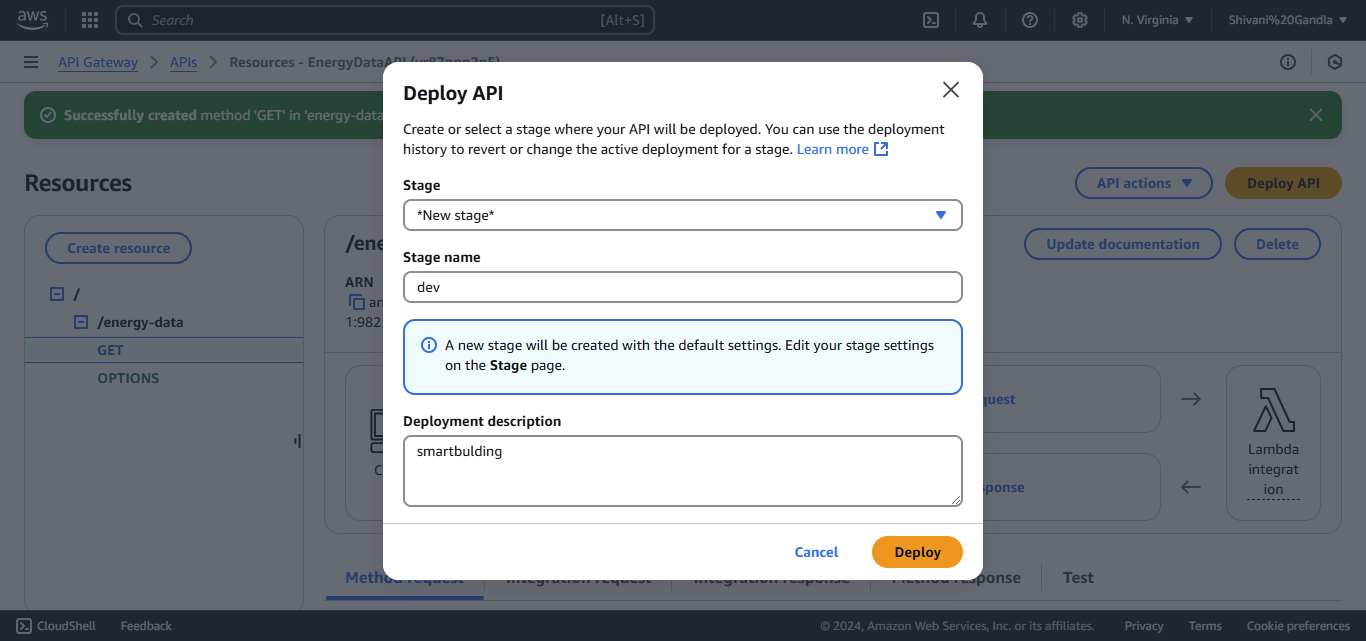
#### ****Lambda Integration Example****

* Select **Lambda Function**, then enter the name of your Lambda function (e.g., FetchEnergyData).



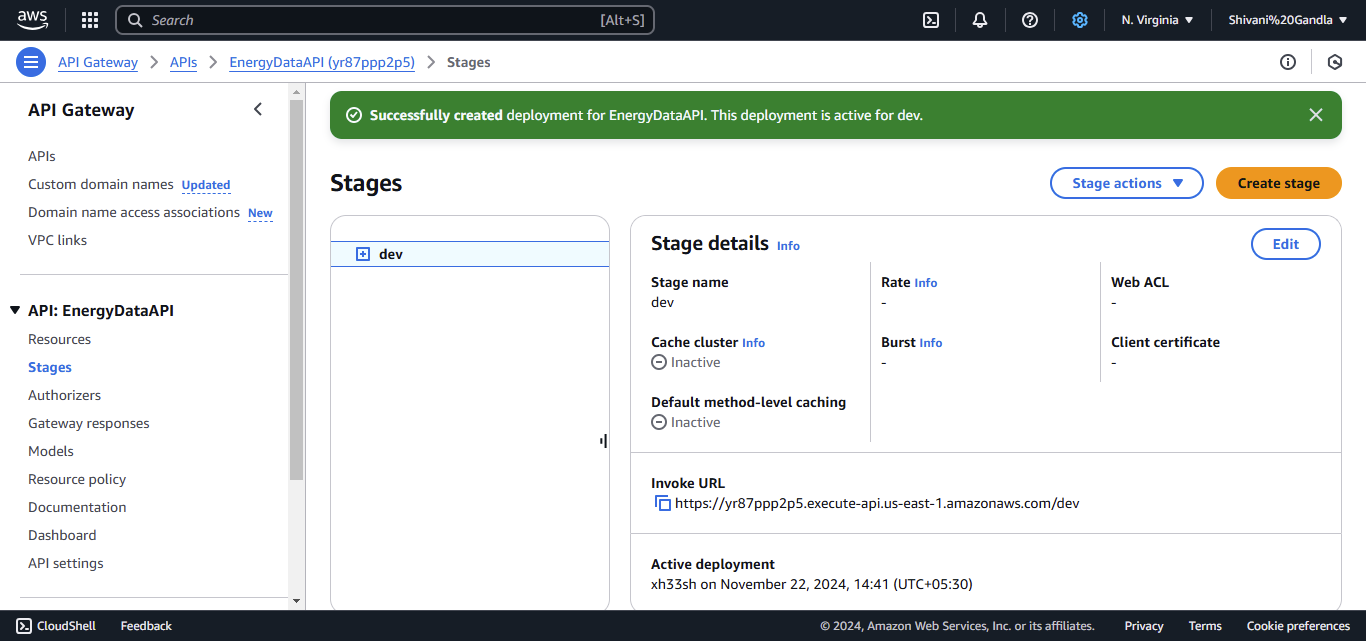
#### ****Deploy the API****

* Click **Actions > Deploy API**.



 Create a new stage (e.g., dev ) and click **Deploy**.

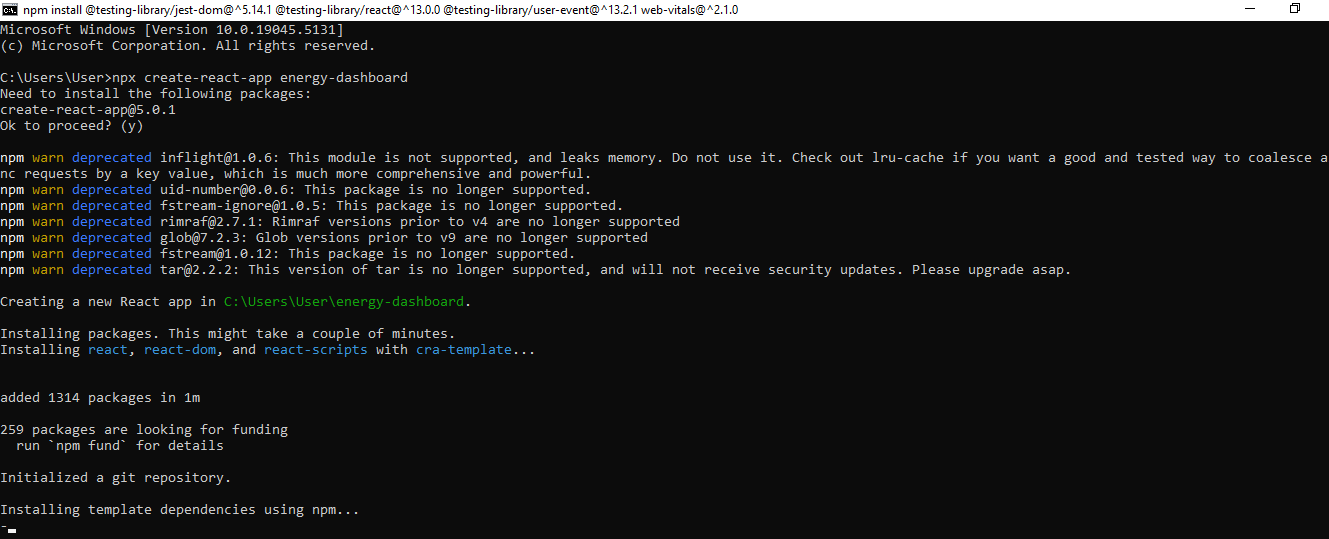
<https://yr87ppp2p5.execute-api.us-east-1.amazonaws.com/dev>

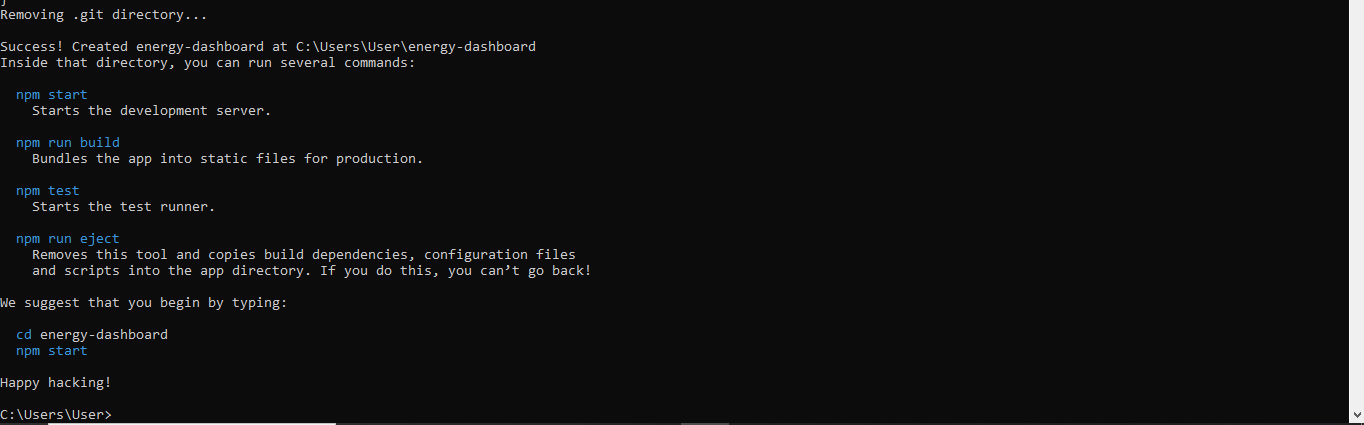


### ****Part 2: Develop the Frontend****

#### ****Set Up React****

* Install Node.js and npm if not already installed.
  + Download from [Node.js Official Website](https://nodejs.org/).
* Create a React application:





Navigate to the project directory:



Start the development server to test:



#### ****Build the Dashboard Component****

* Open the src folder in your project directory.
* Replace the content of App.js with the following code:

import React, { useState, useEffect } from "react";

function Dashboard() {

const [data, setData] = useState([]);

// Replace <api-gateway-endpoint> with the actual Invoke URL

const apiEndpoint = "https://<api-gateway-endpoint>/energy-data";

useEffect(() => {

fetch(apiEndpoint)

.then((res) => res.json())

.then((data) => setData(data))

.catch((err) => console.error("Error fetching data:", err));

}, []);

return (

<div style={{ padding: "20px", fontFamily: "Arial, sans-serif" }}>

<h1 style={{ textAlign: "center" }}>Energy Dashboard</h1>

<table style={{ width: "100%", borderCollapse: "collapse", margin: "20px auto" }}>

<thead>

<tr style={{ backgroundColor: "#f4f4f4", textAlign: "left" }}>

<th style={{ border: "1px solid #ddd", padding: "8px" }}>Device ID</th>

<th style={{ border: "1px solid #ddd", padding: "8px" }}>Temperature (°C)</th>

<th style={{ border: "1px solid #ddd", padding: "8px" }}>Timestamp</th>

</tr>

</thead>

<tbody>

{data.map((item, index) => (

<tr key={index} style={{ borderBottom: "1px solid #ddd" }}>

<td style={{ border: "1px solid #ddd", padding: "8px" }}>{item.device\_id}</td>

<td style={{ border: "1px solid #ddd", padding: "8px" }}>{item.temperature}</td>

<td style={{ border: "1px solid #ddd", padding: "8px" }}>{item.timestamp}</td>

</tr>

))}

</tbody>

</table>

</div>

);

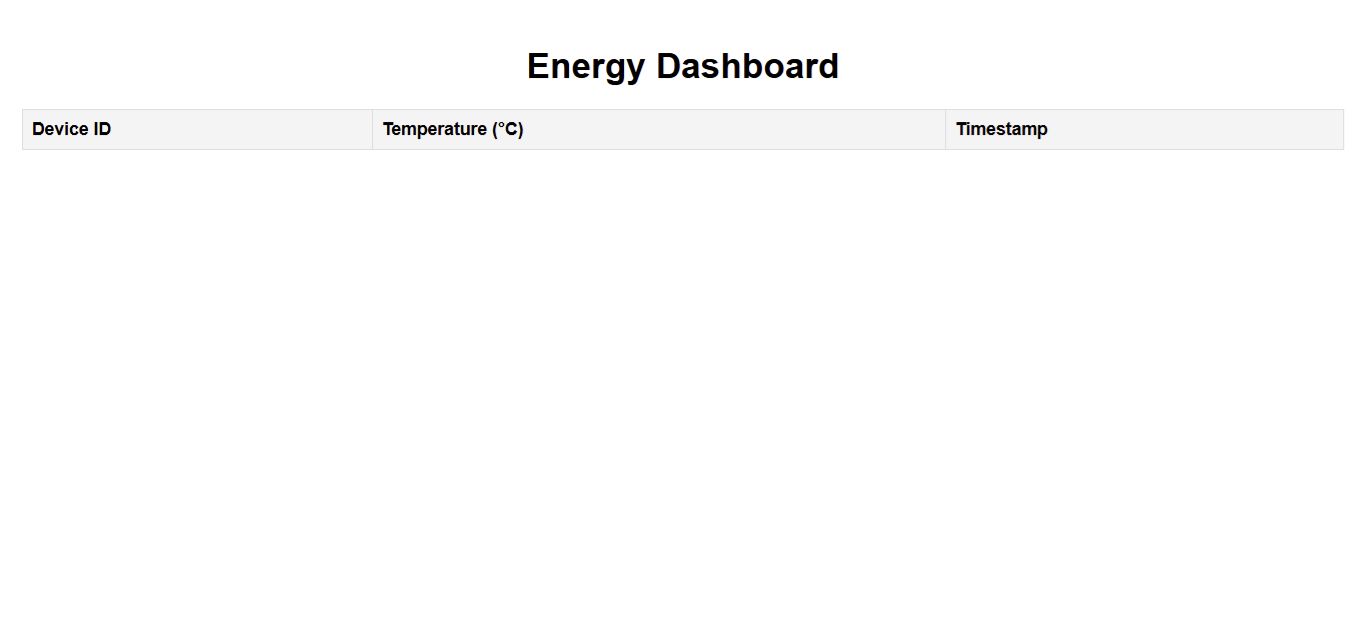
}

export default Dashboard;

#### ****Test the Dashboard****

* Run the development server:

npm start

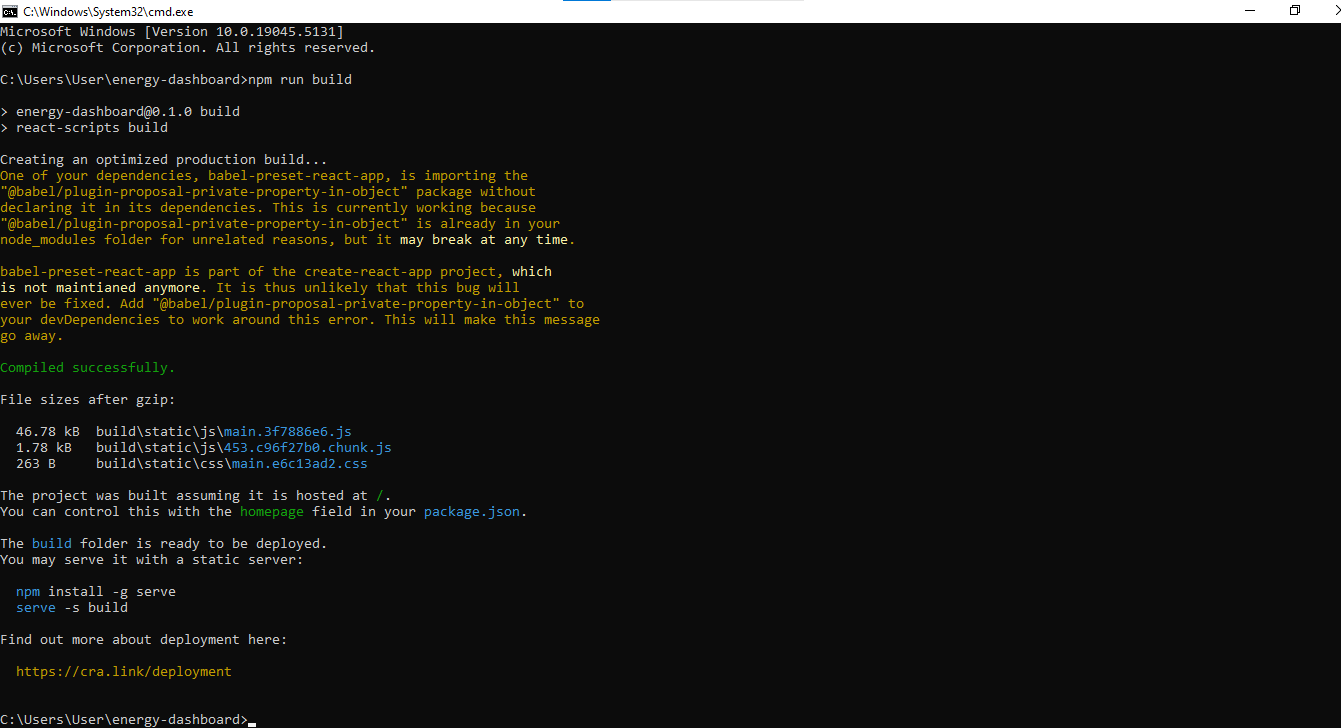


### ****Part 3: Deploy the Dashboard****

### ****Build for Production****

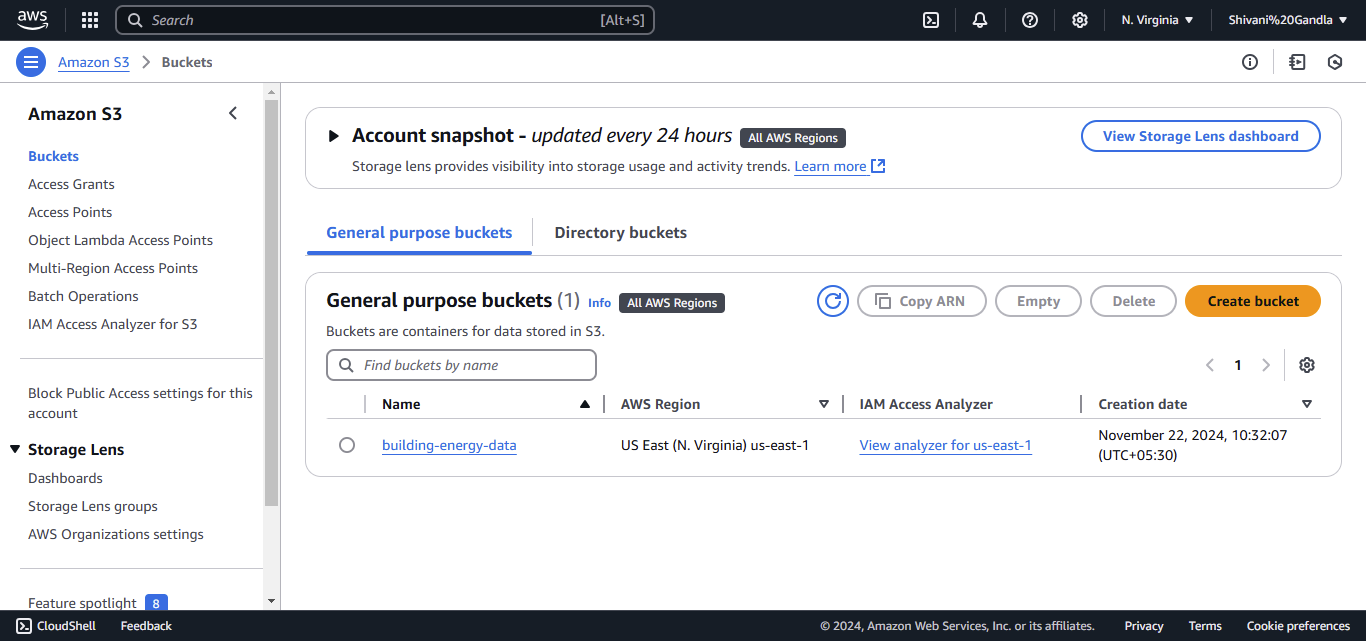
* Open your terminal and navigate to the project directory.
* Run the following command to generate a production-ready build of your React app

npm run build

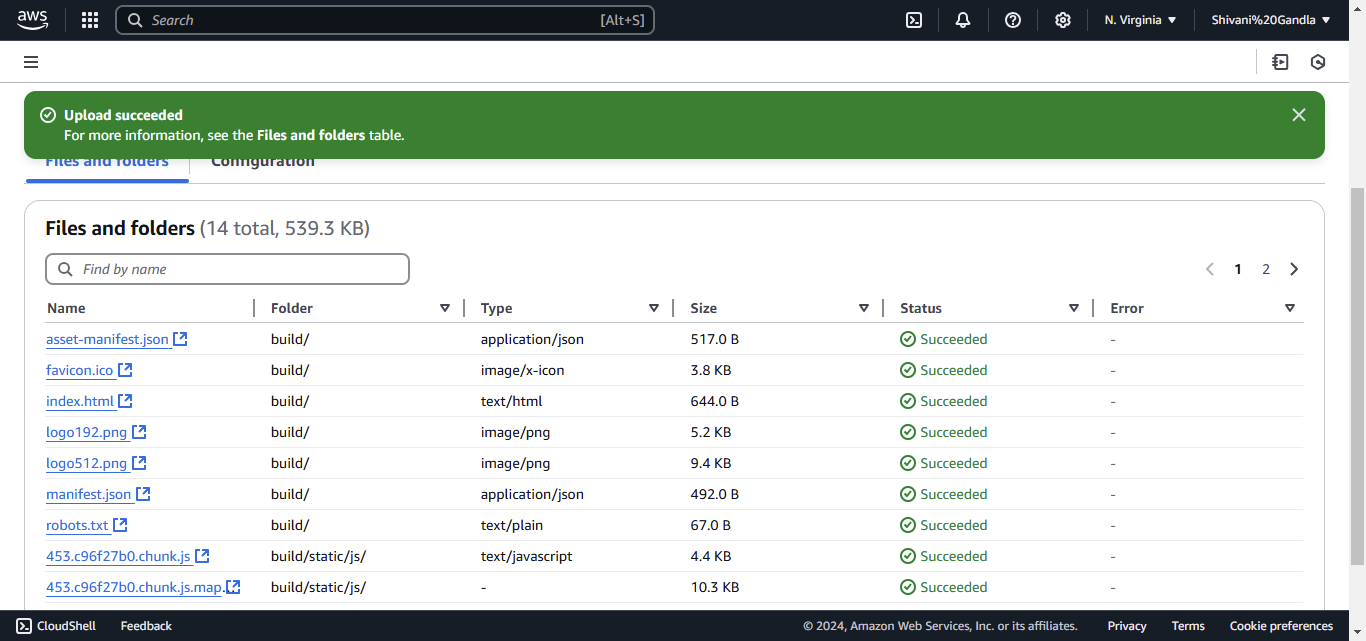


#### ****Deploy to AWS S3****

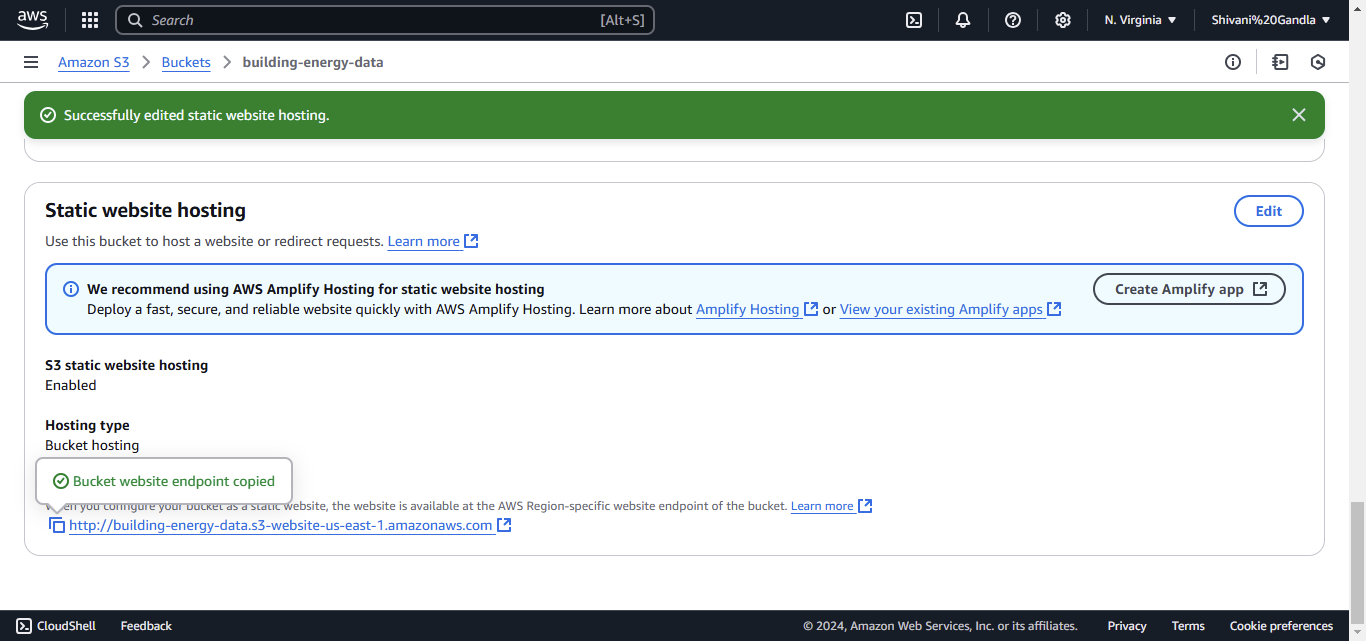
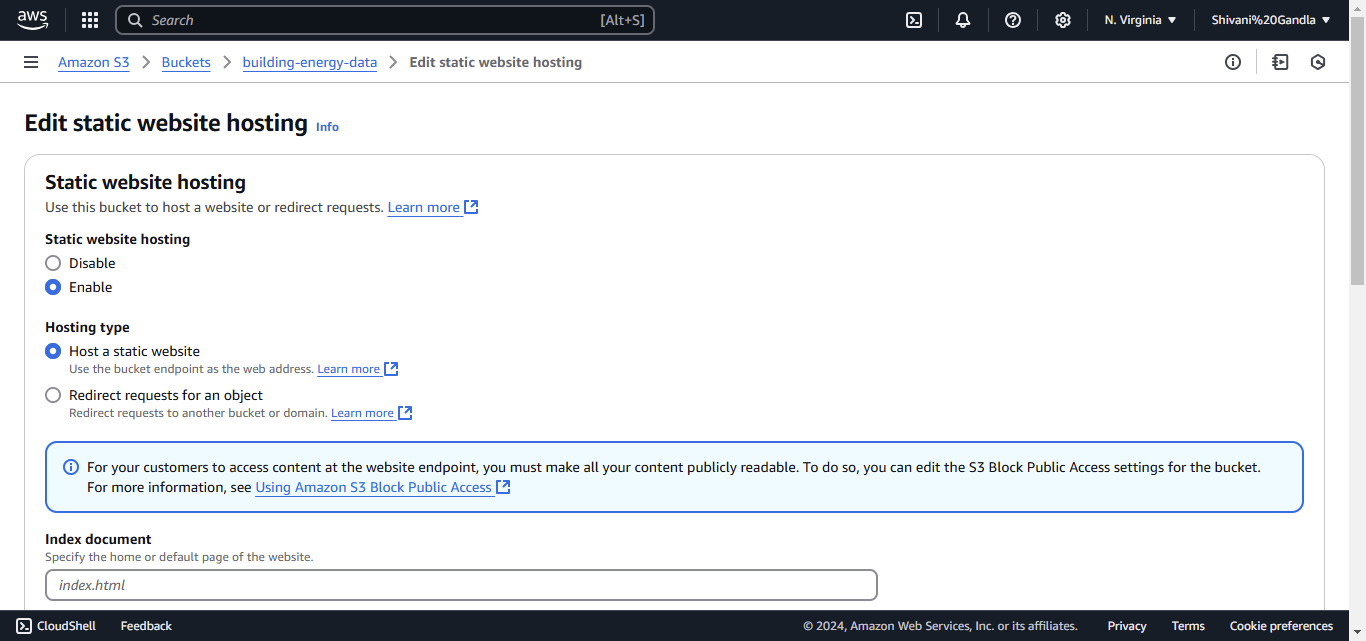
* Navigate to the **Amazon S3** console.



* Upload the contents of the build folder to the bucket.



* Enable **Static Website Hosting** under bucket properties.



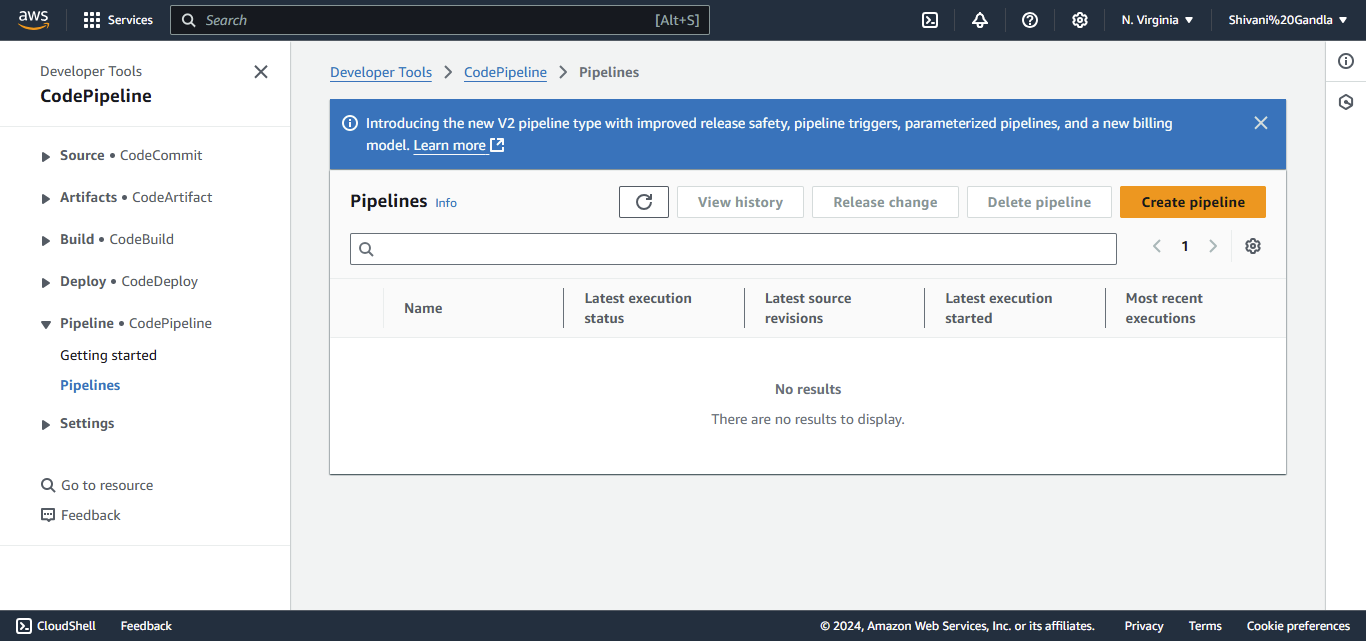
<http://building-energy-data.s3-website-us-east-1.amazonaws.com>

**Step 5: Deployment**

**CI/CD Pipeline**

#### ****1. CI/CD Pipeline****

* **Go to the CodePipeline Console**:
* Navigate to **AWS CodePipeline** from the AWS Management Console.



* **Create a Pipeline**:
* Click **Create Pipeline**.

**Step 5: Monitor with AWS CloudWatch**

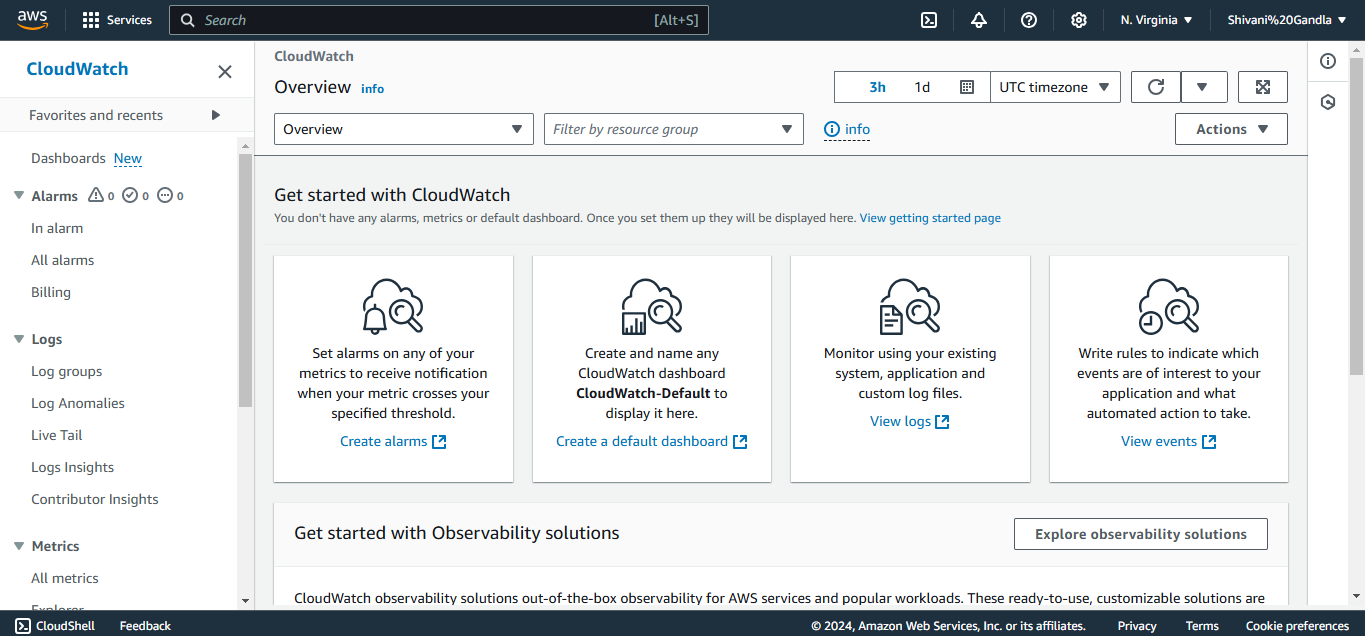
Monitoring your system is crucial for tracking the performance of devices, Lambda functions, and API requests.

#### ****1. Set Up Metrics in CloudWatch****

CloudWatch metrics track the performance of various AWS resources like Lambda functions, IoT devices, and Redshift clusters.

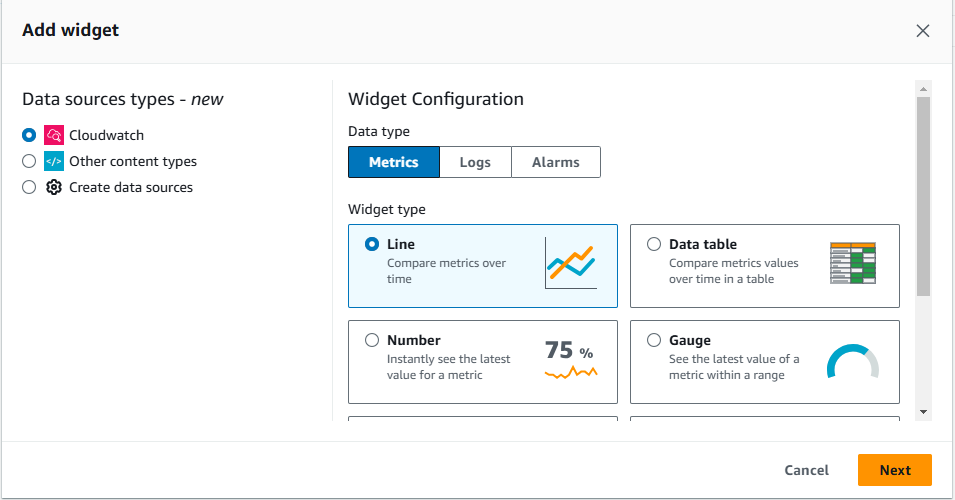
##### **Step 1: Set Up Metrics for IoT Devices and Lambda Functions**

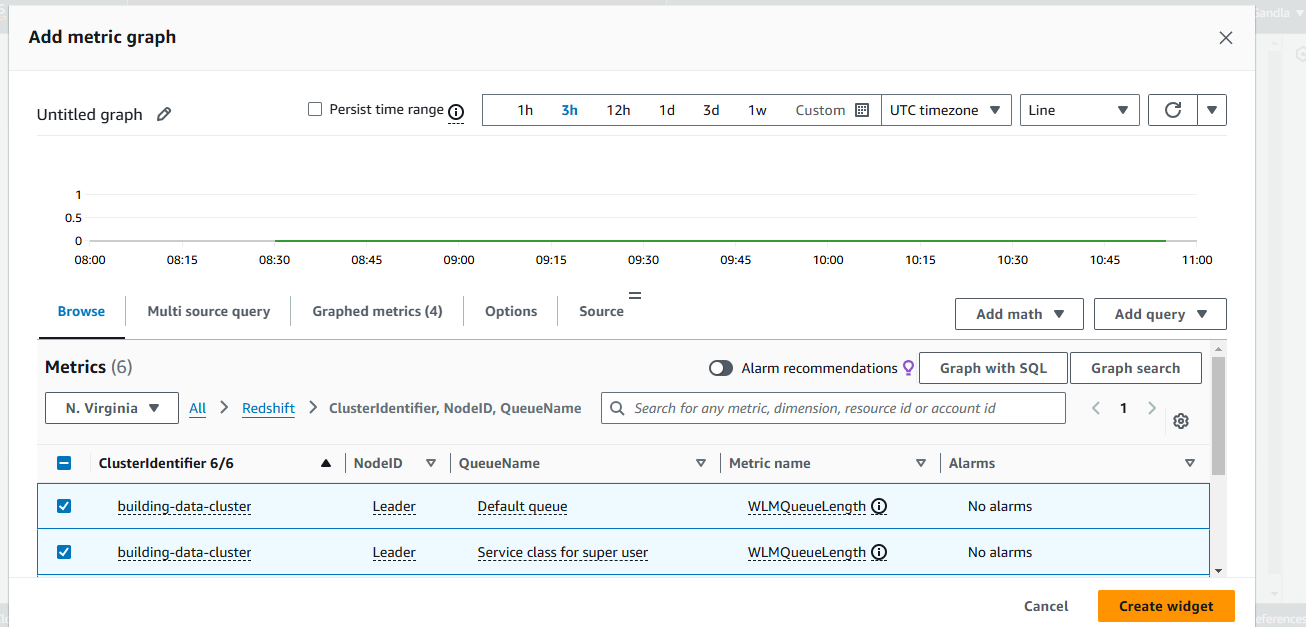
1. **Go to CloudWatch Console**:
   * In the AWS Management Console, search for **CloudWatch** and click to open.

****

##### **Step 1: Set Up Dashboards**

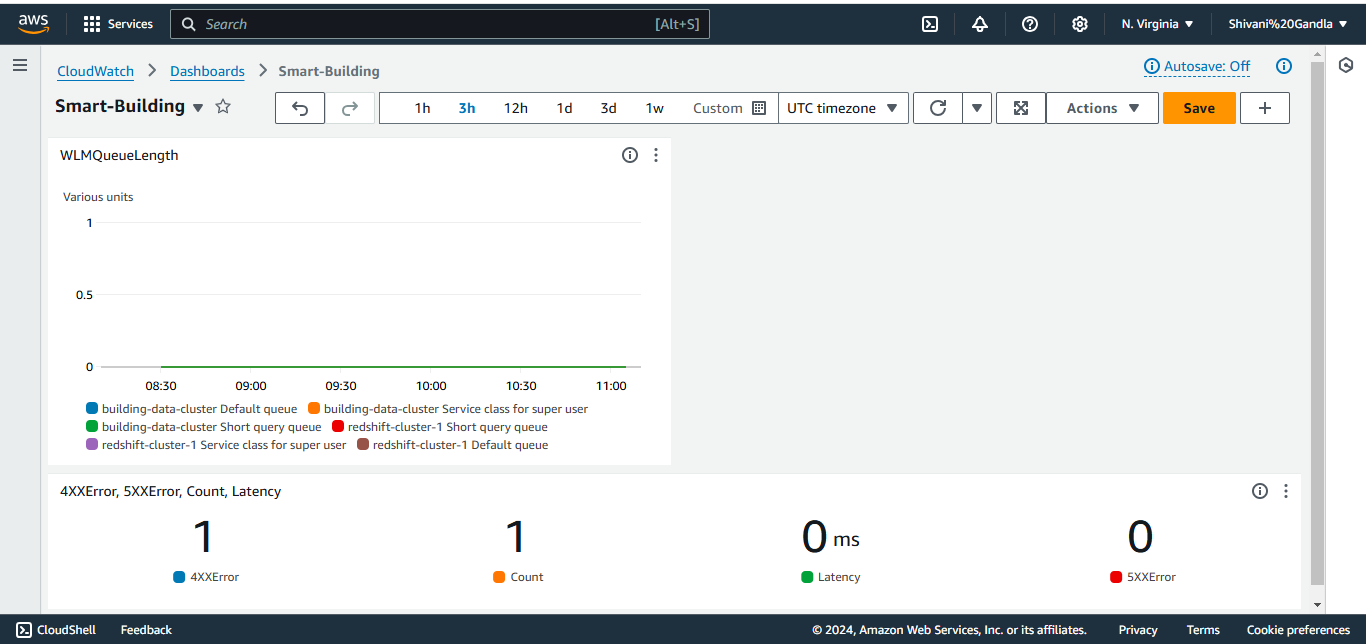
1. In the **CloudWatch Console**, go to **Dashboards** and click **Create Dashboard**.
2. Name your dashboard (e.g., Smart-Building).
3. **Add Widgets**:
   * Click **Add Widget** to include graphs for Lambda invocations, error rates, IoT device temperature readings, and more.
   * Choose **Line Graph**, **Number**, or **Text** to visualize key metrics.

****

****

Customize the dashboard to show relevant metrics like:

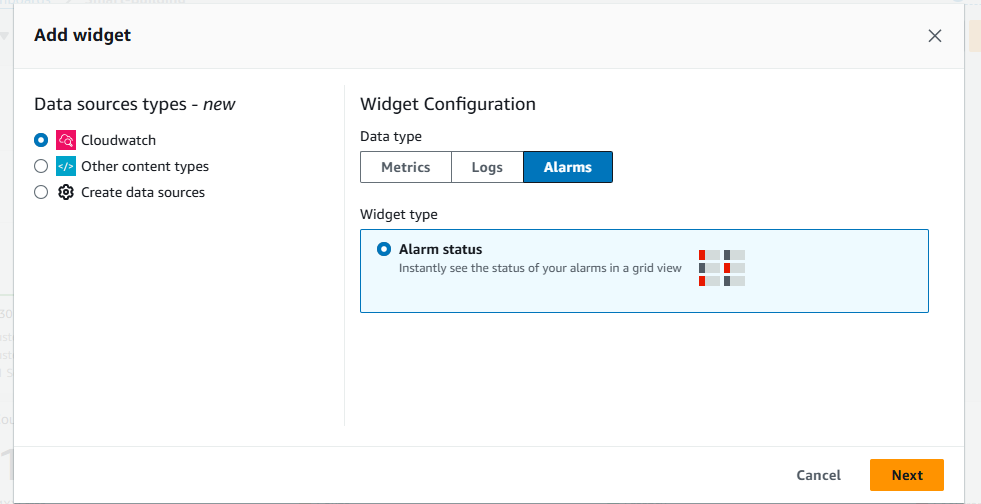
* Temperature over time.
* Lambda function error counts.
* Redshift cluster health.

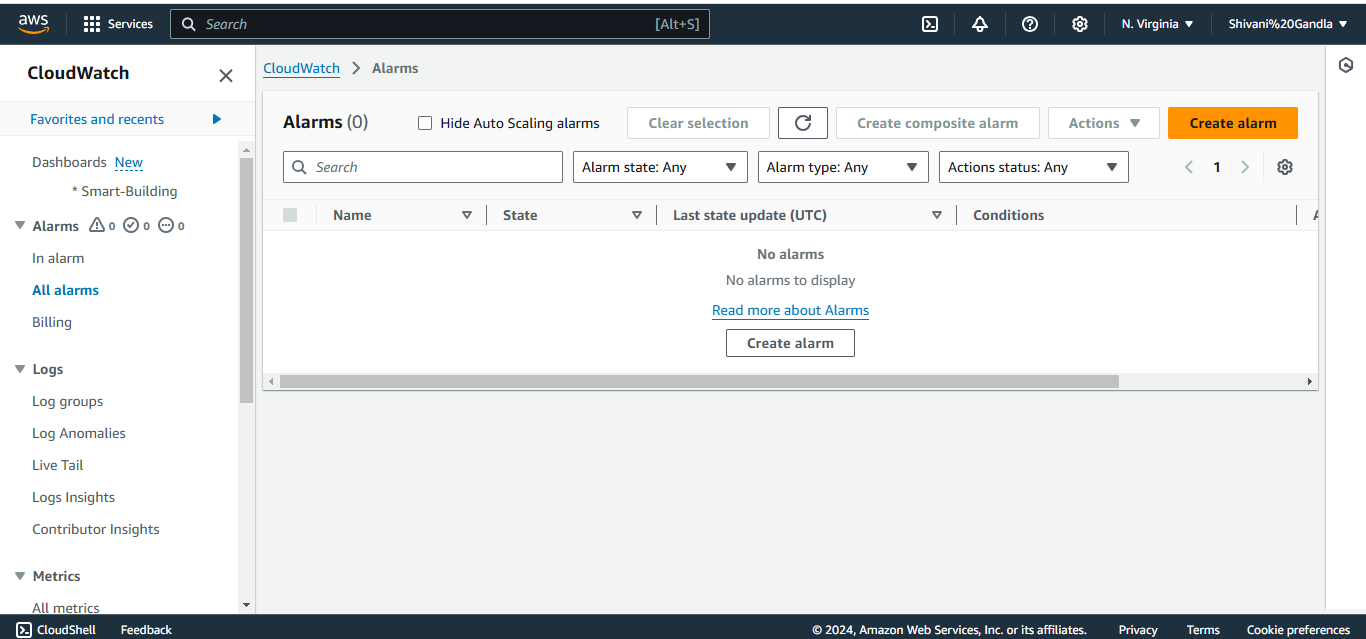
****

#### ****2. Create CloudWatch Alarms****

Alarms help you track abnormal events and trigger actions like notifications.

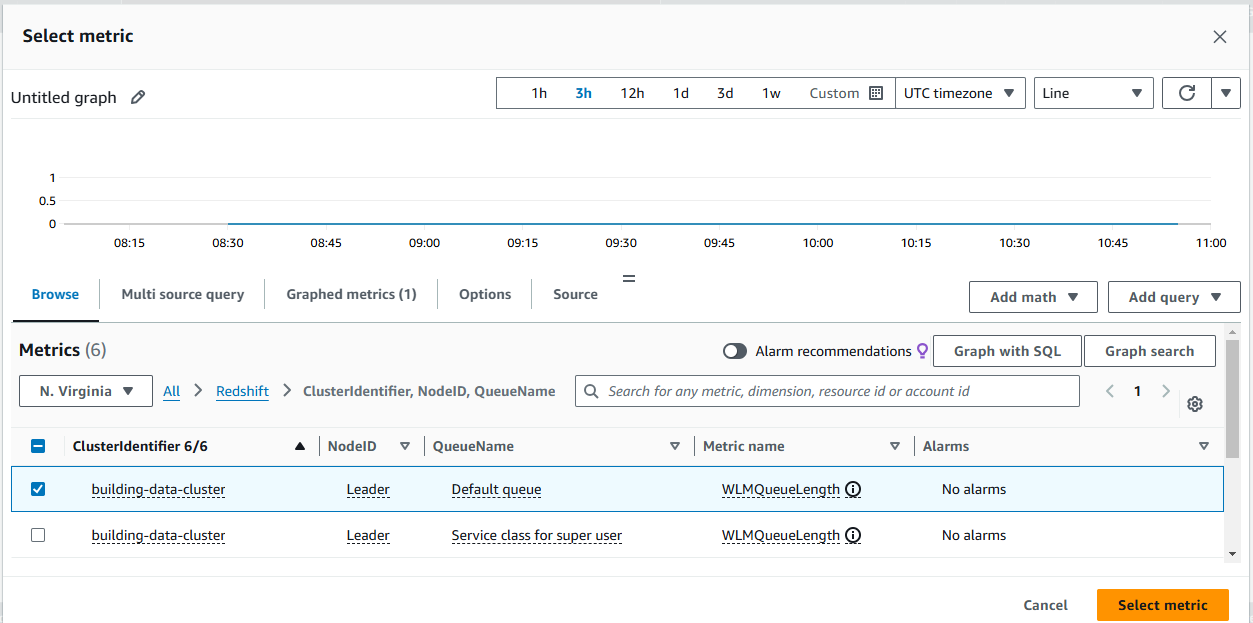
##### **Step 1: Create Alarms for Your Metrics**

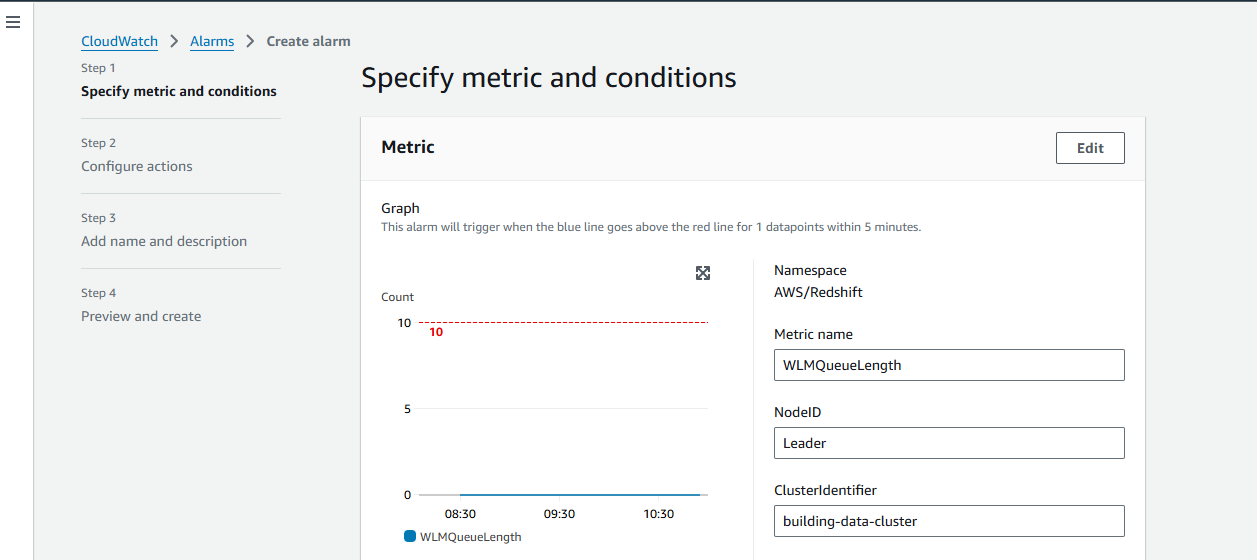
****

****

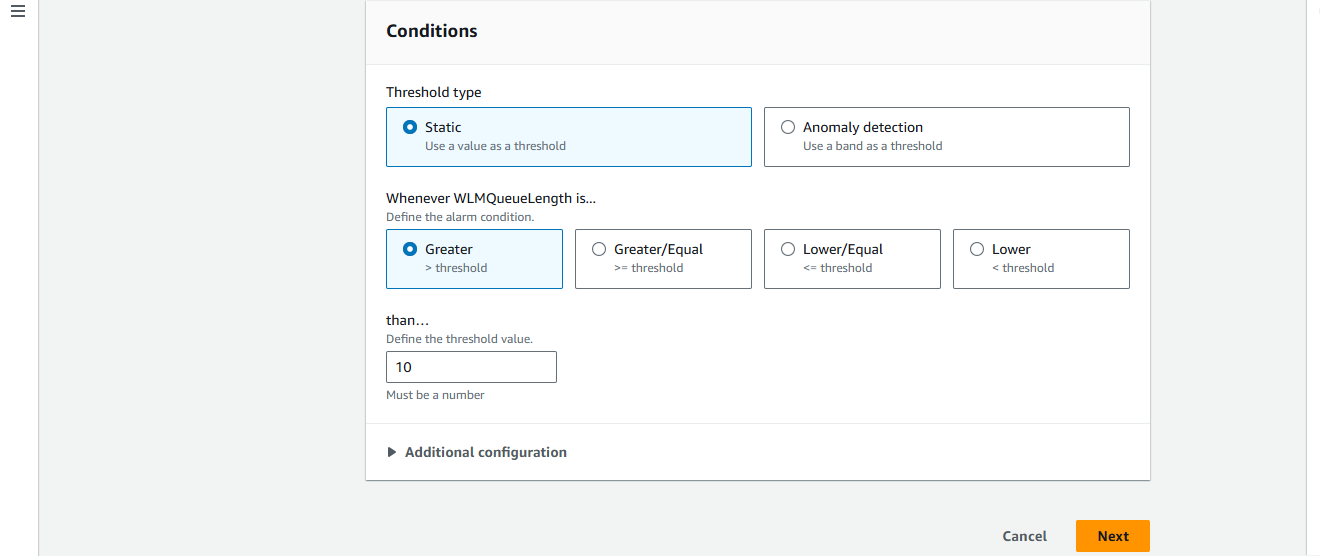
 **Create an Alarm for Redshift**:

* Select the metric (e.g., building-data-cluster).
* Set the **Notification** action (for example, to send an email via SNS).

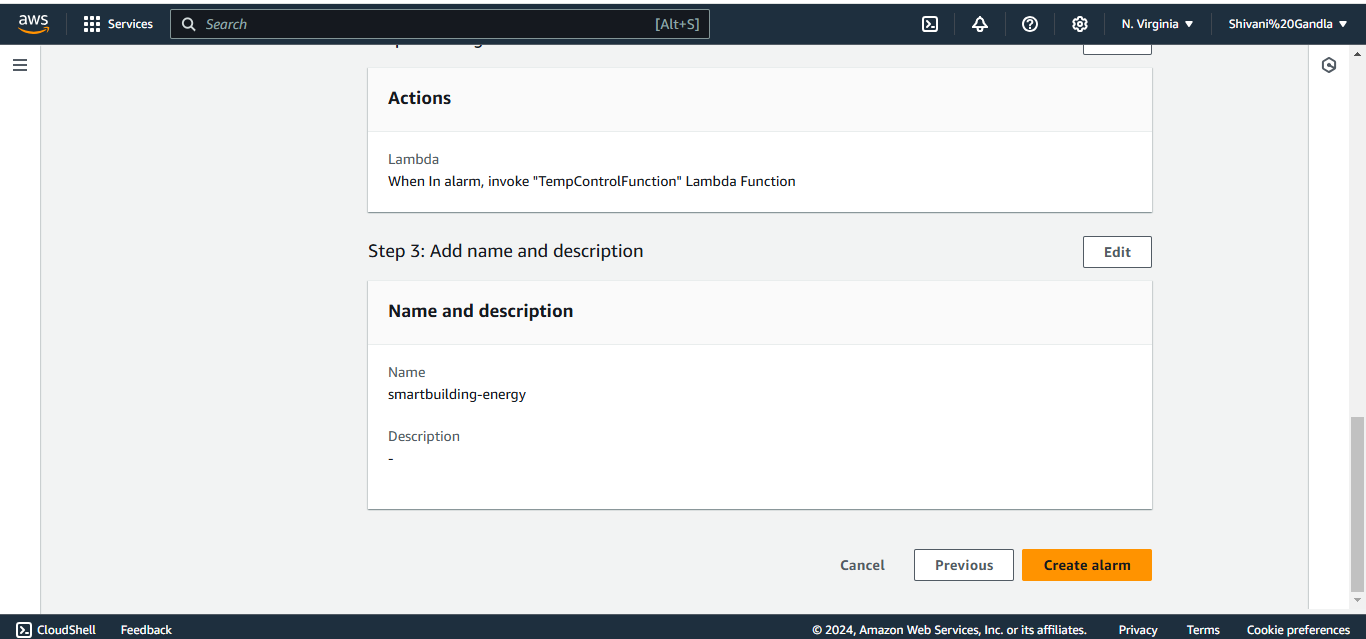
****

****

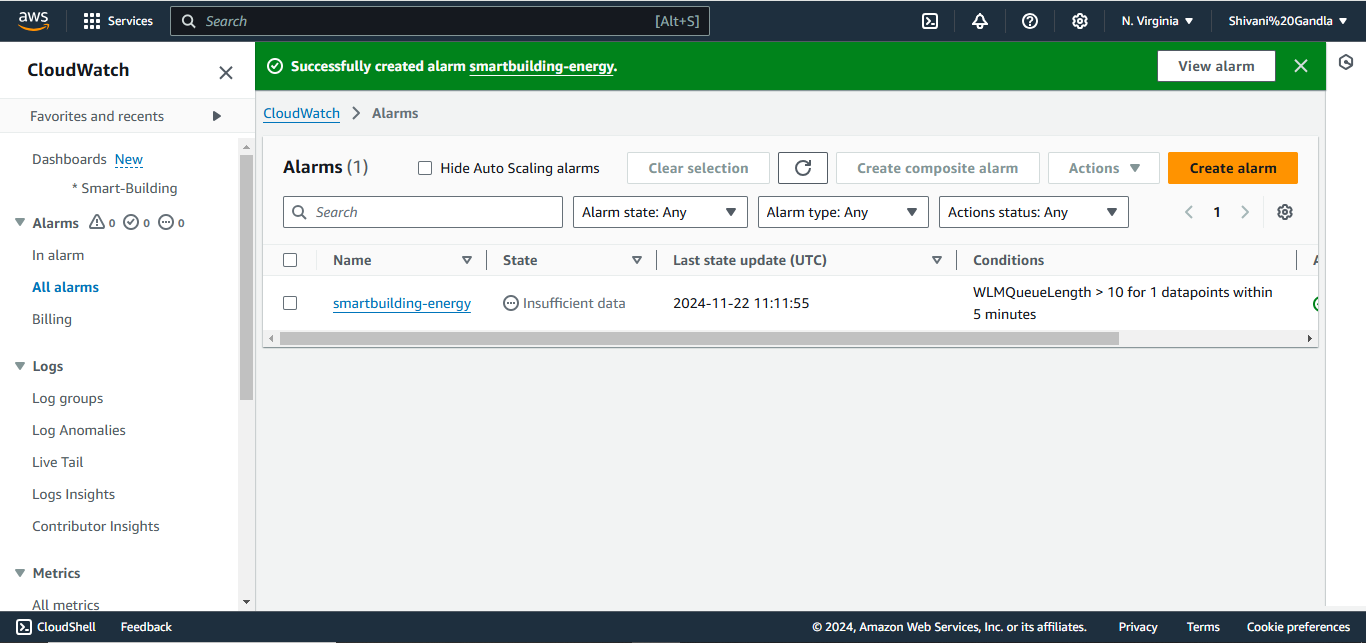
**Click “Next”**

****

**Now click “create alarm”.**

****

**The Alarm was created “Successfully”.**

****

**Here is the final CloudWatch Dashboard.**

