NeedEnergy

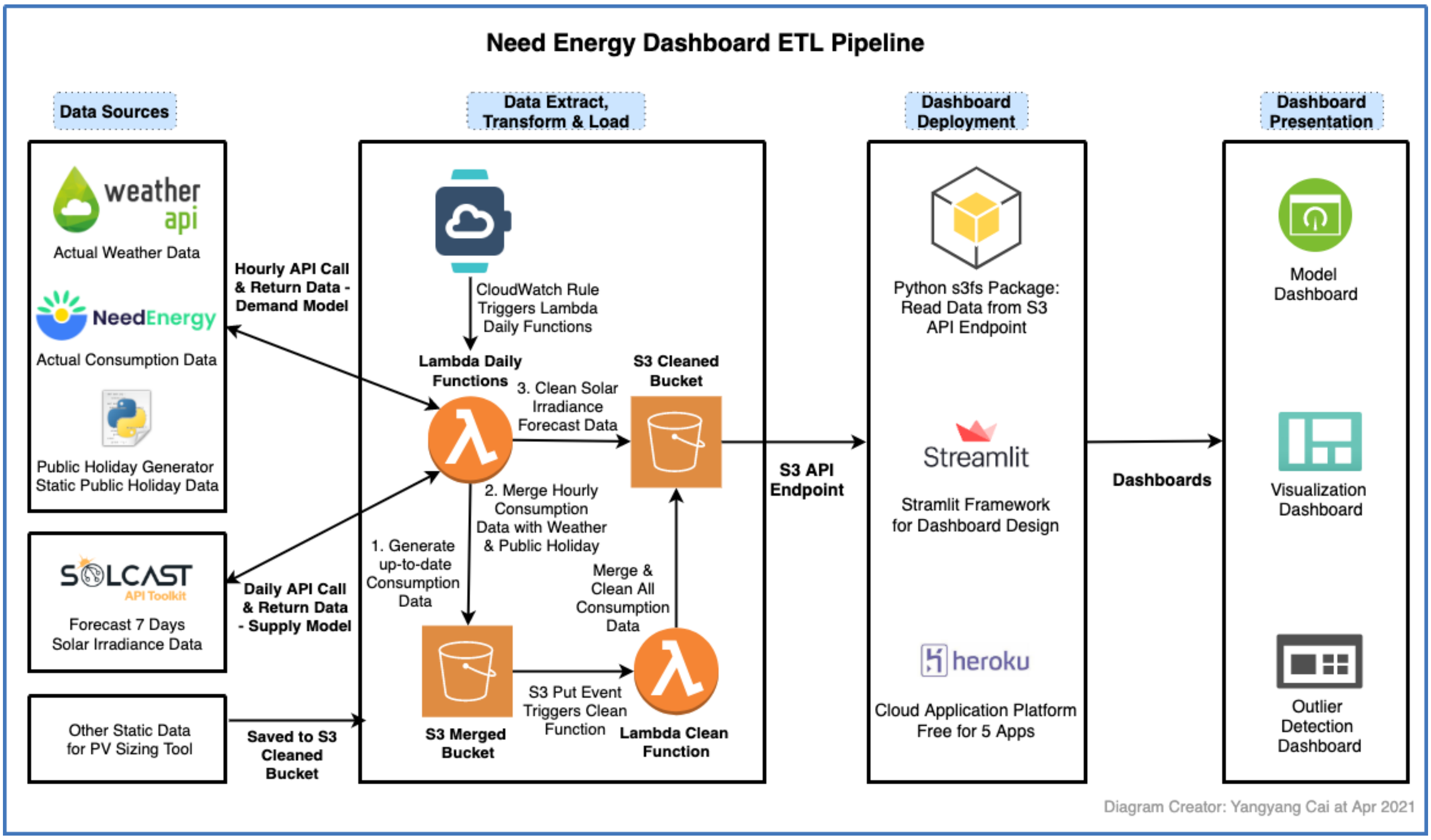
Dashboard ETL Pipeline Guideline

Author: Yangyang Cai

Date: Apr 2021

Dashboard ETL Pipeline Overview

Diagram Overview



Need Energy Dashboard ETL Pipeline
Data Sources
Data Extract, Transform & Load
Dashboard Deployment
Dashboard Presentation
weather api
Actual Weather Data
NeedEnergy
Actual Consumption Data
Public Holiday Generator Static Public Holiday Data
SOLCAST API Toolkit
Forecast 7 Days Solar Irradiance Data
Other Static Data for PV Sizing Tool
Hourly API Call & Return Data - Demand Model
Daily API Call & Return Data - Supply Model
Saved to S3 Cleaned Bucket
CloudWatch Rule Triggers Lambda Daily Functions
Lambda Daily Functions
3. Clean Solar Irradiance Forecast Data
S3 Cleaned Bucket
2. Merge Hourly Consumption Data with Weather & Public Holiday
1. Generate up-to-date Consumption Data
Merge & Clean All Consumption Data
S3 Merged Bucket
S3 Put Event Triggers Clean Function
Lambda Clean Function
S3 API Endpoint
Python s3fs Package: Read Data from S3 API Endpoint
Streamlit
Stramlit Framework for Dashboard Design
heroku
Cloud Application Platform Free for 5 Apps
Dashboards
Model Dashboard
Visualization Dashboard
Outlier Detection Dashboard
Diagram Creator: Yangyang Cai at Apr 2021

Stages Overview

- **Data Sources**
  - OpenWeather API: One Call API used to get hourly actual weather data, free service could get 1000 calls per day.
  - **Need Energy API**: Need Energy API used to get hourly actual consumption data
  - **Public Holiday Generator**: Python Script used to generate public holidays information
  - Solcast API: Solcast API used to get 7 days forecast solar irradiance data, apply *pip install solcast* to install this package. Free service could get 10 calls per day.
  - **Static Data**: Static Data sources used for PV Sizing tool

- **Data Extract, Transform & Load**
  - Scheduled by CloudWacth, managed by Lambda Function, data folders are S3 Bucket.
  - **Consumption Data**: Consumption merge function runs hourly at XX:10 to get previous hour's data, saves hourly merged data to S3 Merged Bucket, which is S3 Put Event. And this S3 Put Event at S3 Merged Bucket triggers another

    lambda function to merge and clean all consumption data, saves final cleaned data to S3 Cleaned Bucket.
  - **Solar Irradiance Data**: Daily supply lambda function run daily at 00:05 UTC+2, saves 7 days forecast solar irradiance data to S3 Cleansed Bucket

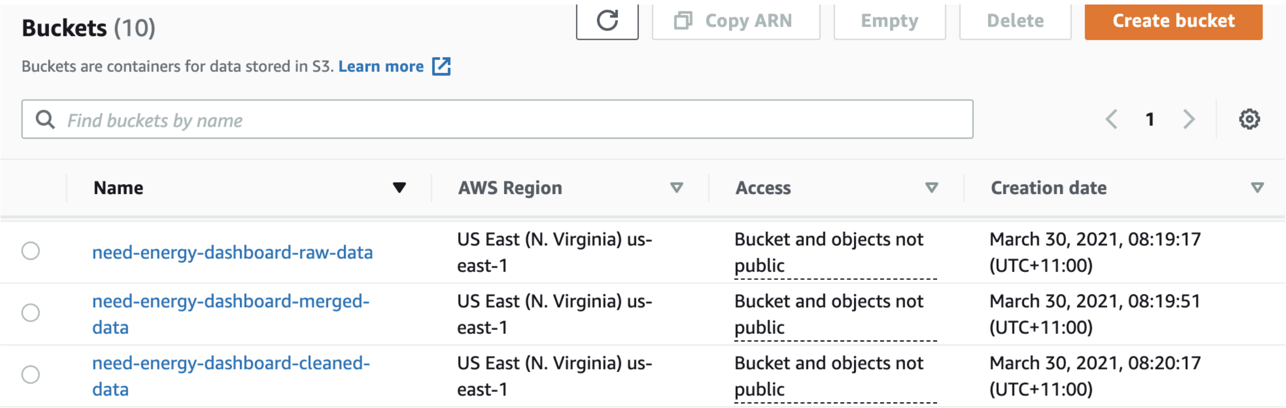
- **Dashboard Deployment**
  - **Python s3fs Package**: S3FS builds on aiobotocore to provide a convenient Python filesystem interface for S3. Details in S3 API EndPoint for Dashboard Section.
  - **Streamlit:** Streamlit turns data scripts into shareable web apps in minutes, all in Python, all for free and no front-end experience requried.
  - **Heroku:** Cloud Application Platform, free for 5 Apps

- **Dashboard Presentation**
  - **Model Dashboard:** Dashboard to plot demand model and supply model, showing alert information based on forecast consumptionl and supply information.
  - **Visualization Dashboard:** Dashboard to show data exploration for all datasets provided by Need Energy Company.
  - **Outlier Detection Dashboard**: Dashboard to show outlier dection for all dastasets provide by Need Energy Company.

Dashboard ETL Pipeline Setup

S3 Bucket Setup

1. Create S3 Bucket



Buckets (10)

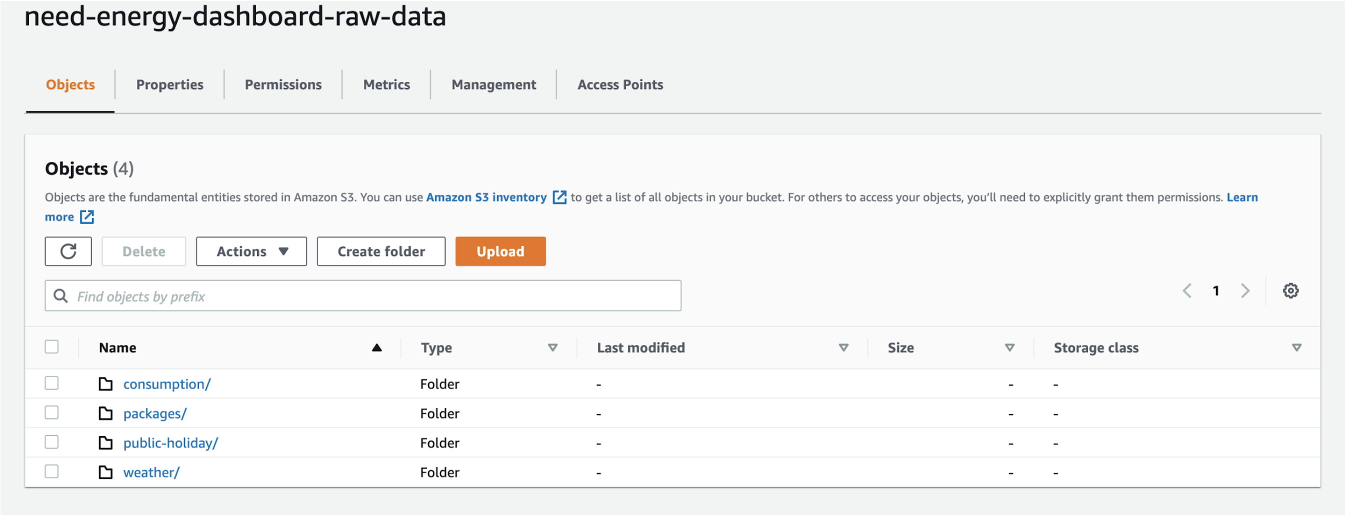
Buckets are containers for data stored in S3. Learn more

	Name	AWS Region	Access	Creation date
○	need-energy-dashboard-raw-data	US East (N. Virginia) us-east-1	Bucket and objects not public	March 30, 2021, 08:19:17 (UTC+11:00)
○	need-energy-dashboard-merged-data	US East (N. Virginia) us-east-1	Bucket and objects not public	March 30, 2021, 08:19:51 (UTC+11:00)
○	need-energy-dashboard-cleaned-data	US East (N. Virginia) us-east-1	Bucket and objects not public	March 30, 2021, 08:20:17 (UTC+11:00)

The S3 Buckets Details:

- **need-energy-dashboard-raw-data:** Stored daily API data and public holiday data
- **need-energy-dashboard-merged-data:** Stored daily merged data
- **need-energy-dashboard-cleaned-data:** Stored historical cleaned data for modelling

2. Create folders under S3 Bucket



need-energy-dashboard-raw-data

Objects (4)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

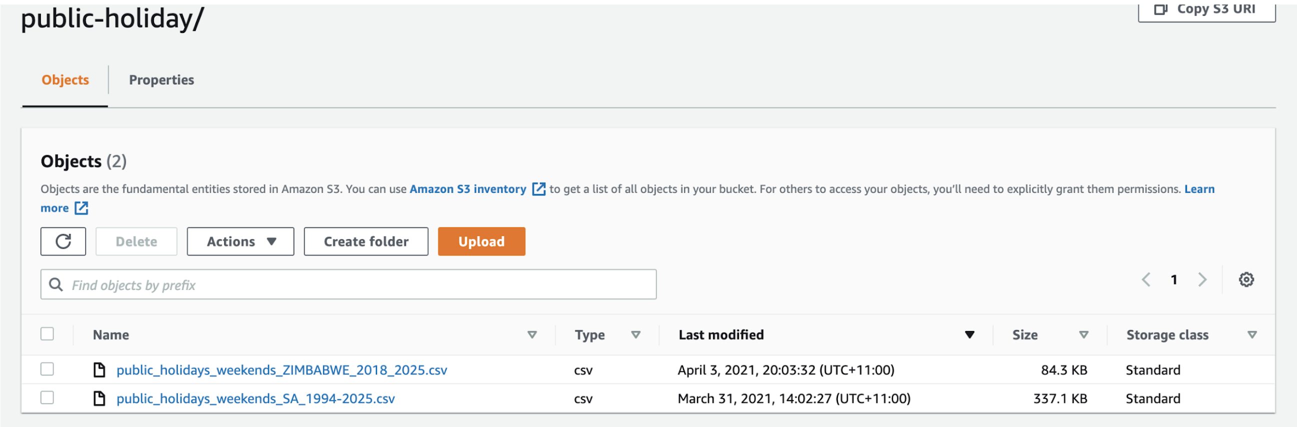
□	Name	Type	Last modified	Size	Storage class
□	consumption/	Folder	-	-	-
□	packages/	Folder	-	-	-
□	public-holiday/	Folder	-	-	-
□	weather/	Folder	-	-	-

The S3 Folders Details:

- **consumption:** This folder used to store hourly consumption raw data
- **packages:** This folder used to store the package used for lambda function
- **public-holiday:** This folder used to store public holiday information
- **weather:** This folder used to store hourly weather raw data

3. Upload statistic data & historical data to S3 Bucket

- Upload the static public holiday to S3 Raw Data Bucket
  Currently the SA data covers the period from 1994 to 2025 and Zimbabwe covers the period from 2018 to 2025. The script is summaried in Appenidx A and you could rerun it to generate future data



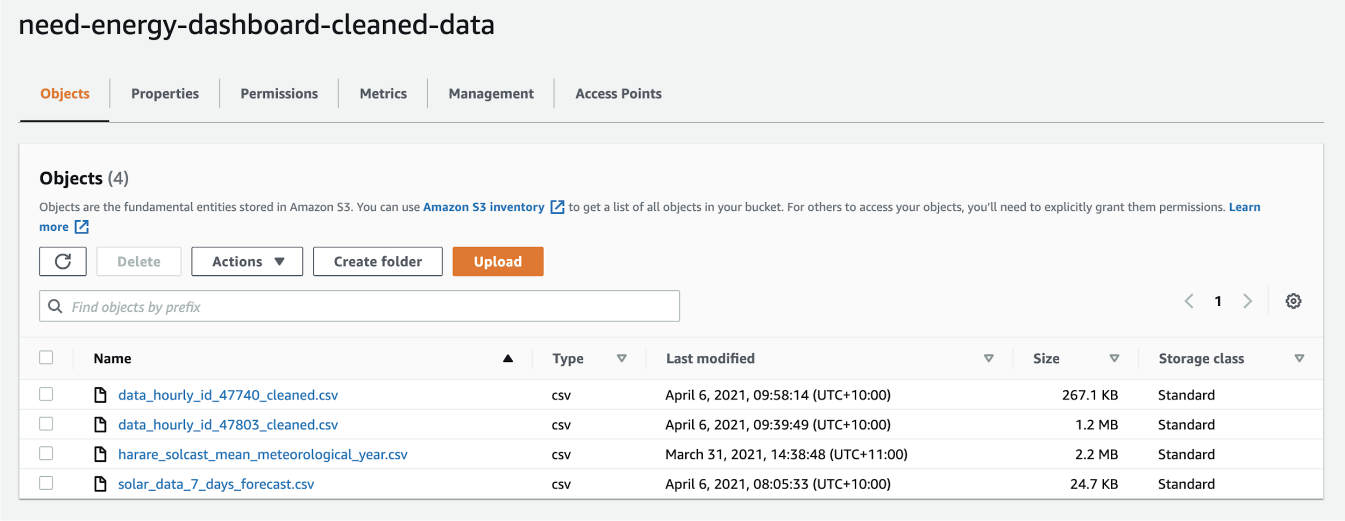
public-holiday/

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

	Name	Type	Last modified	Size	Storage class
□	public_holidays_weekends_ZIMBABWE_2018_2025.csv	csv	April 3, 2021, 20:03:32 (UTC+11:00)	84.3 KB	Standard
□	public_holidays_weekends_SA_1994-2025.csv	csv	March 31, 2021, 14:02:27 (UTC+11:00)	337.1 KB	Standard

- Upload the recent actual consumption data & weather data to S3 Merged Bucket

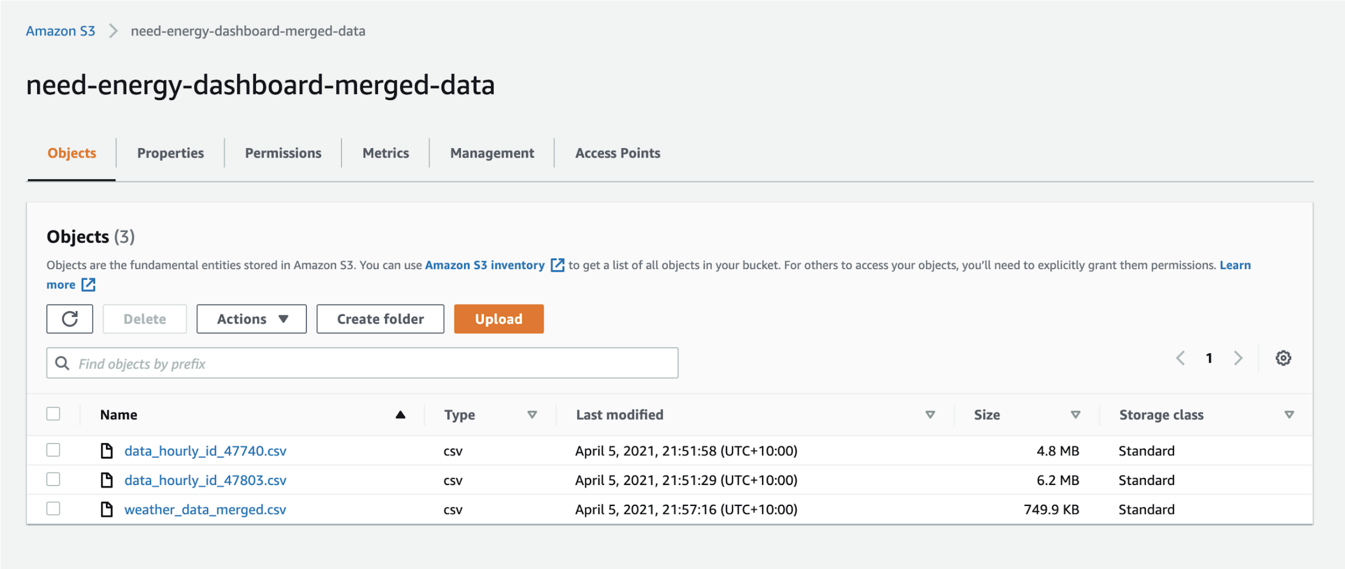


need-energy-dashboard-cleaned-data

Objects (4)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

	Name	Type	Last modified	Size	Storage class
□	data_hourly_id_47740_cleaned.csv	csv	April 6, 2021, 09:58:14 (UTC+10:00)	267.1 KB	Standard
□	data_hourly_id_47803_cleaned.csv	csv	April 6, 2021, 09:39:49 (UTC+10:00)	1.2 MB	Standard
□	harare_solcast_mean_meteorological_year.csv	csv	March 31, 2021, 14:38:48 (UTC+11:00)	2.2 MB	Standard
□	solar_data_7_days_forecast.csv	csv	April 6, 2021, 08:05:33 (UTC+10:00)	24.7 KB	Standard



need-energy-dashboard-merged-data

Objects (3)

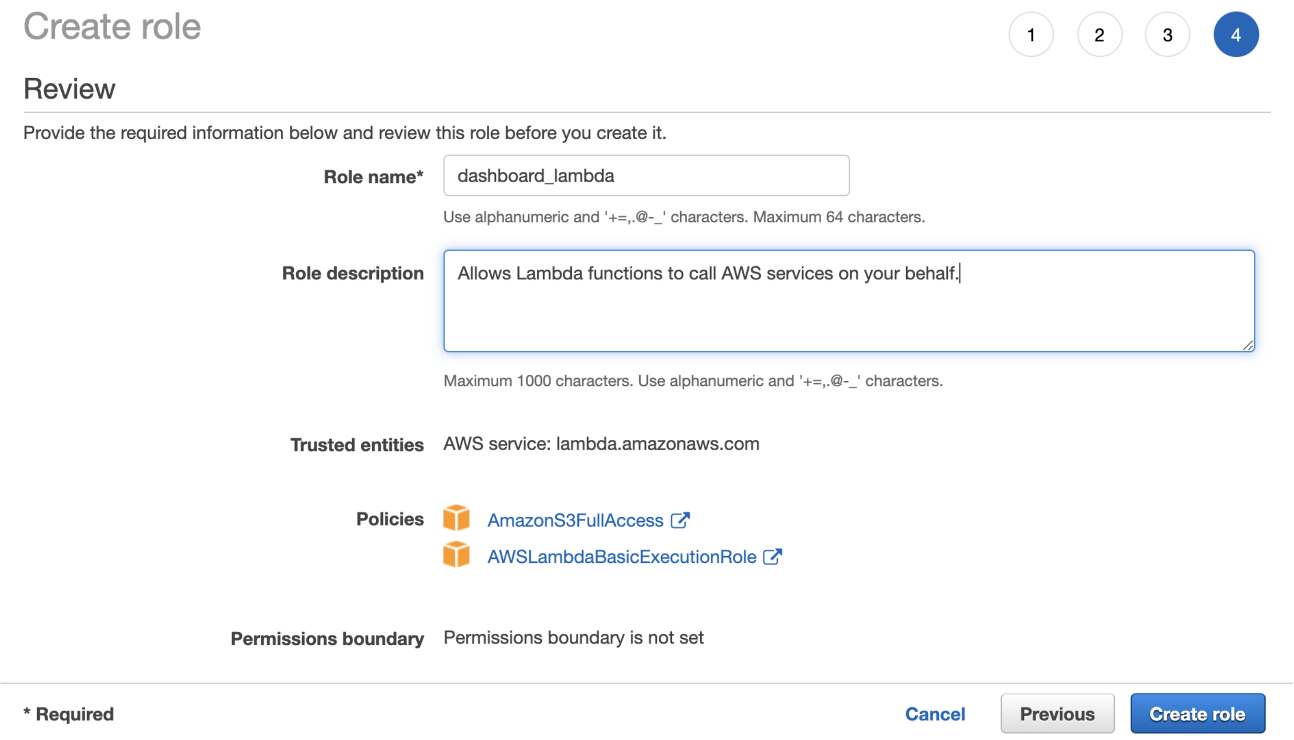
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

	Name	Type	Last modified	Size	Storage class
□	data_hourly_id_47740.csv	csv	April 5, 2021, 21:51:58 (UTC+10:00)	4.8 MB	Standard
□	data_hourly_id_47803.csv	csv	April 5, 2021, 21:51:29 (UTC+10:00)	6.2 MB	Standard
□	weather_data_merged.csv	csv	April 5, 2021, 21:57:16 (UTC+10:00)	749.9 KB	Standard

- Upload the recent merged and cleaned data to S3 Cleaned Bucket
- Upload the static data sources for PV Sizing Tool

IAM Role Setup

- Select Lambda Use cases
- Create role, make sure setup correct permission



Create role

Review

Provide the required information below and review this role before you create it.

Role name* dashboard_lambda

Use alphanumeric and '+=,.@-_' characters. Maximum 64 characters.

Role description Allows Lambda functions to call AWS services on your behalf.

Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.

Trusted entities AWS service: lambda.amazonaws.com

Policies AmazonS3FullAccess
AWSLambdaBasicExecutionRole

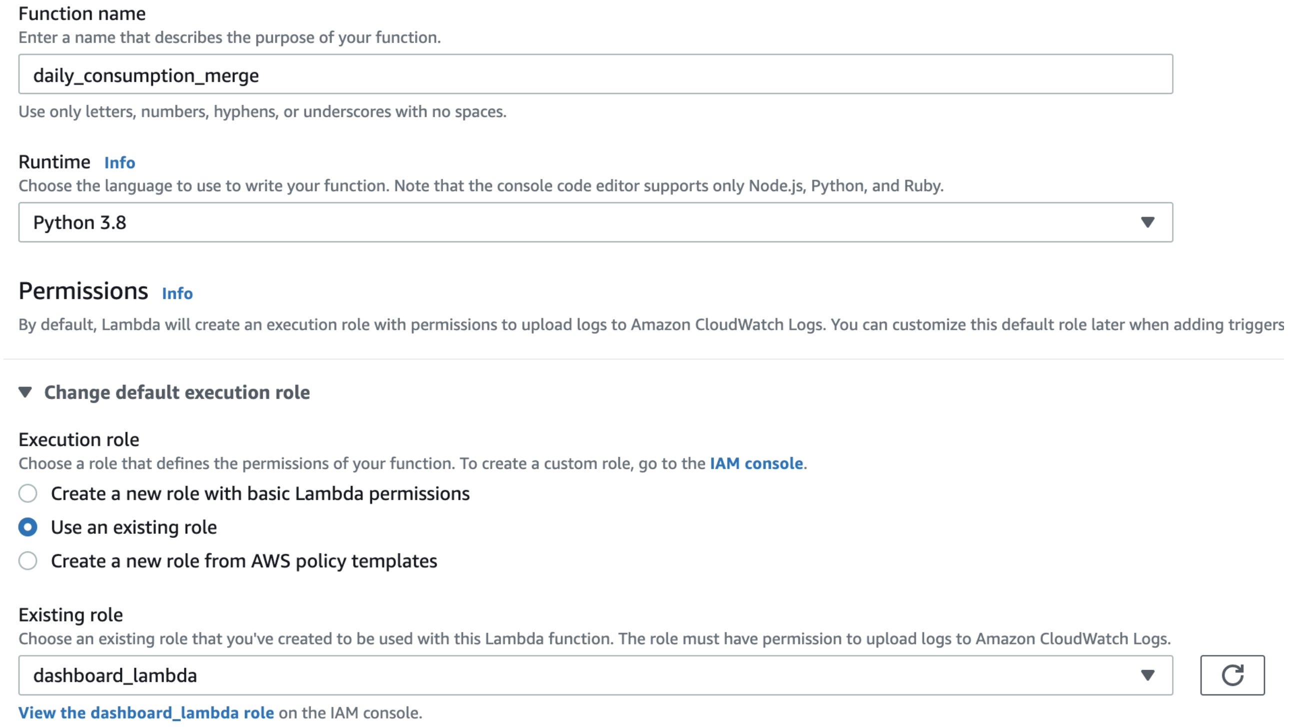
Permissions boundary Permissions boundary is not set

* Required

Lambda Function Setup

1. Create Consumption Merge Function

- Select Author from scratch
- Setup our first lambda function



Function name
Enter a name that describes the purpose of your function.
daily_consumption_merge
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.8 ▼

Permissions Info
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

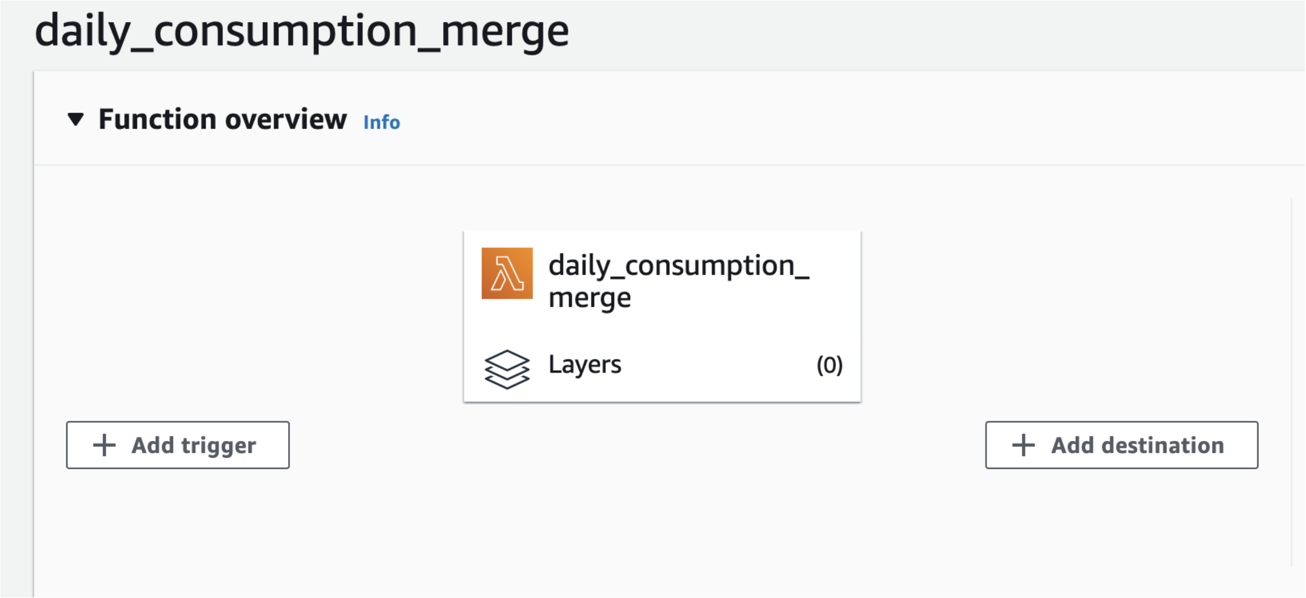
▼ Change default execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

○ Create a new role with basic Lambda permissions
● Use an existing role
○ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
dashboard_lambda ▼
View the dashboard_lambda role on the IAM console.

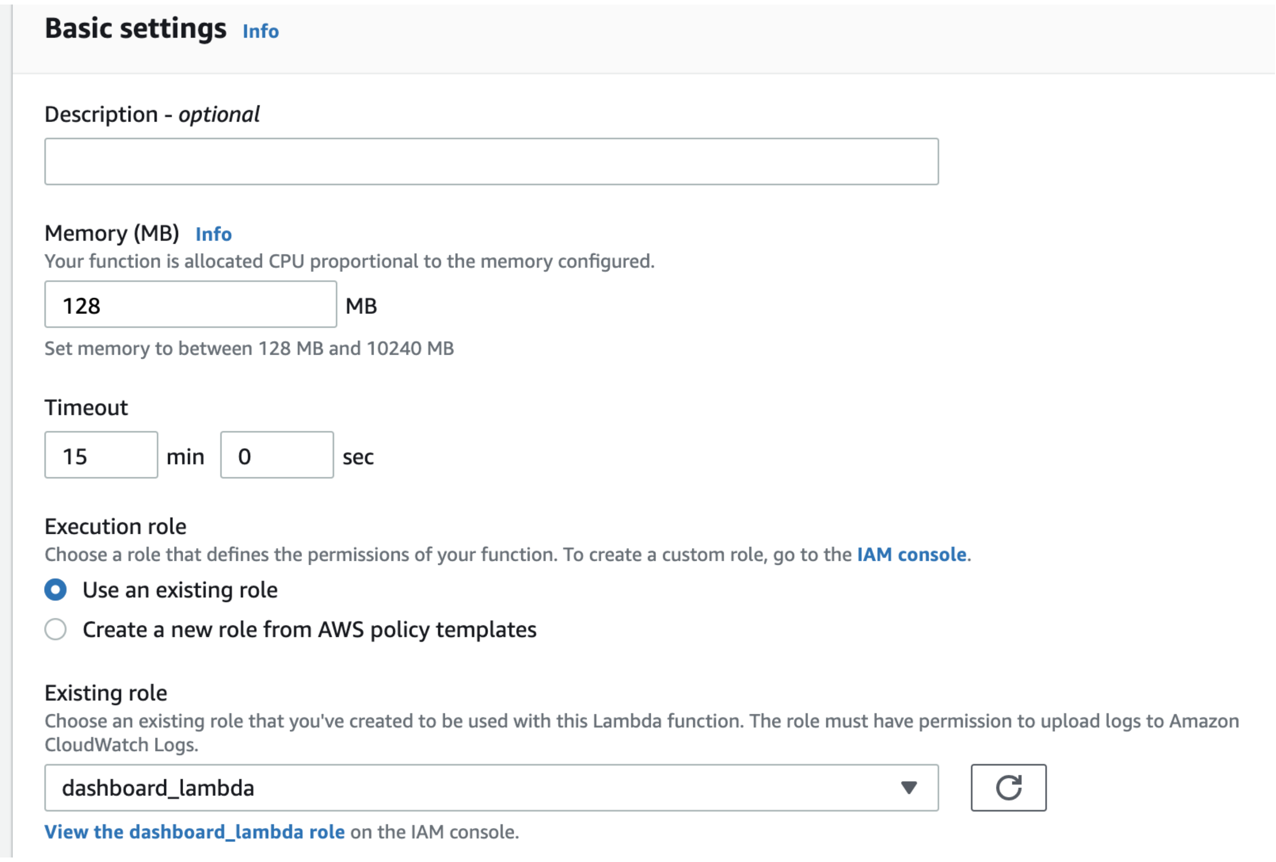
- Add lambda trigger (Cloud Watch Section): This function has been changed to be hourly consumption



daily_consumption_merge

▼ Function overview Info

- Add lambda function Scripts (Appendix B)
- Setup configuration: run out time to be 15 minutes



Basic settings

Description - *optional*

Memory (MB)

Your function is allocated CPU proportional to the memory configured.

128 MB

Set memory to between 128 MB and 10240 MB

Timeout

15 min 0 sec

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

● Use an existing role

○ Create a new role from AWS policy templates

Existing role

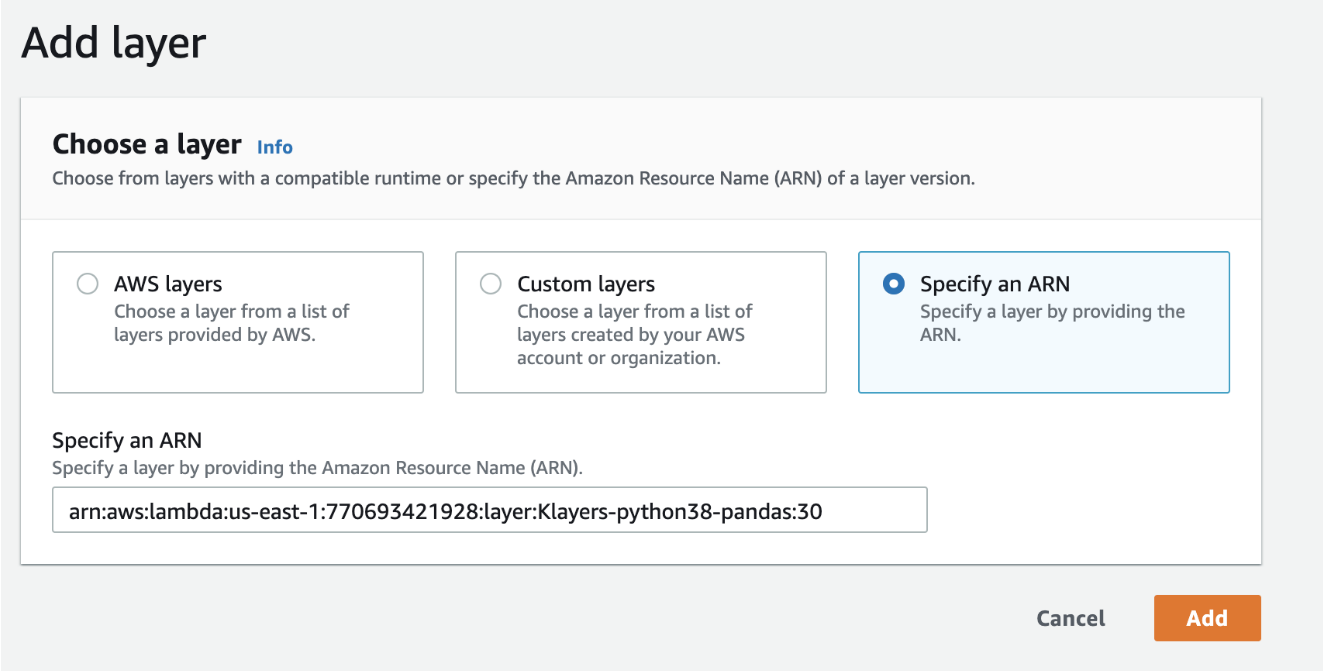
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

dashboard_lambda

View the dashboard_lambda role on the IAM console.

- Setup ARN Layers **-** Because lambda does not have requests and pandas python library, check this article https://melissa-bain.medium.com/how-to-import-python-packages-in-aws-lambda-pandas-scipy-numpy-bb2c98c974e9

  get ARN from here: (Make sure the region is right): https://github.com/keithrozario/Klayers/blob/master/deployments/python3.8/arns/us-east-1.csv



Add layer

Choose a layer Info

Choose from layers with a compatible runtime or specify the Amazon Resource Name (ARN) of a layer version.

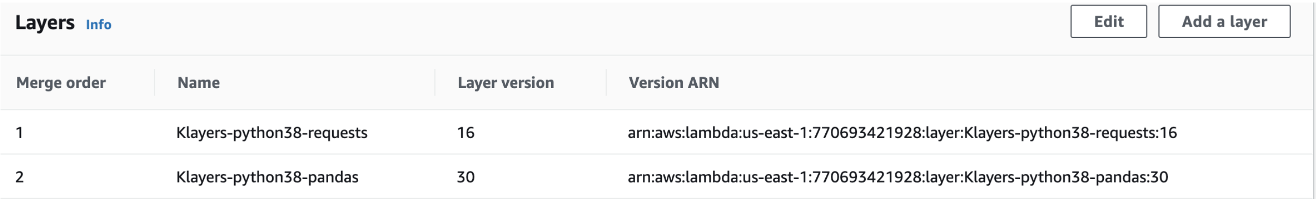
○ AWS layers
Choose a layer from a list of layers provided by AWS.

○ Custom layers
Choose a layer from a list of layers created by your AWS account or organization.

● Specify an ARN
Specify a layer by providing the ARN.

Specify an ARN
Specify a layer by providing the Amazon Resource Name (ARN).

arn:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-pandas:30

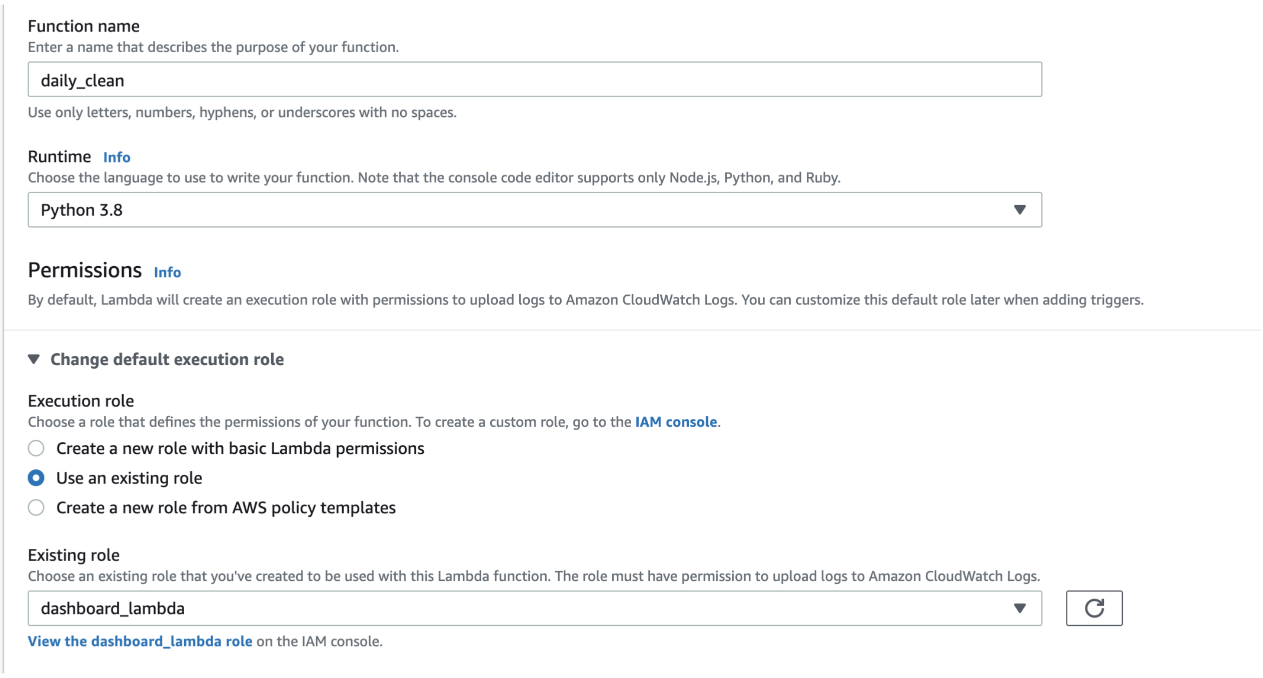


Layers Info

Merge order	Name	Layer version	Version ARN
1	Klayers-python38-requests	16	arn:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-requests:16
2	Klayers-python38-pandas	30	arn:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-pandas:30

2. Create Consumption Clean Function

- Create Lambda Function



Function name

Enter a name that describes the purpose of your function.

daily_clean

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.8

Permissions Info

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.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

○ Create a new role with basic Lambda permissions

● Use an existing role

○ Create a new role from AWS policy templates

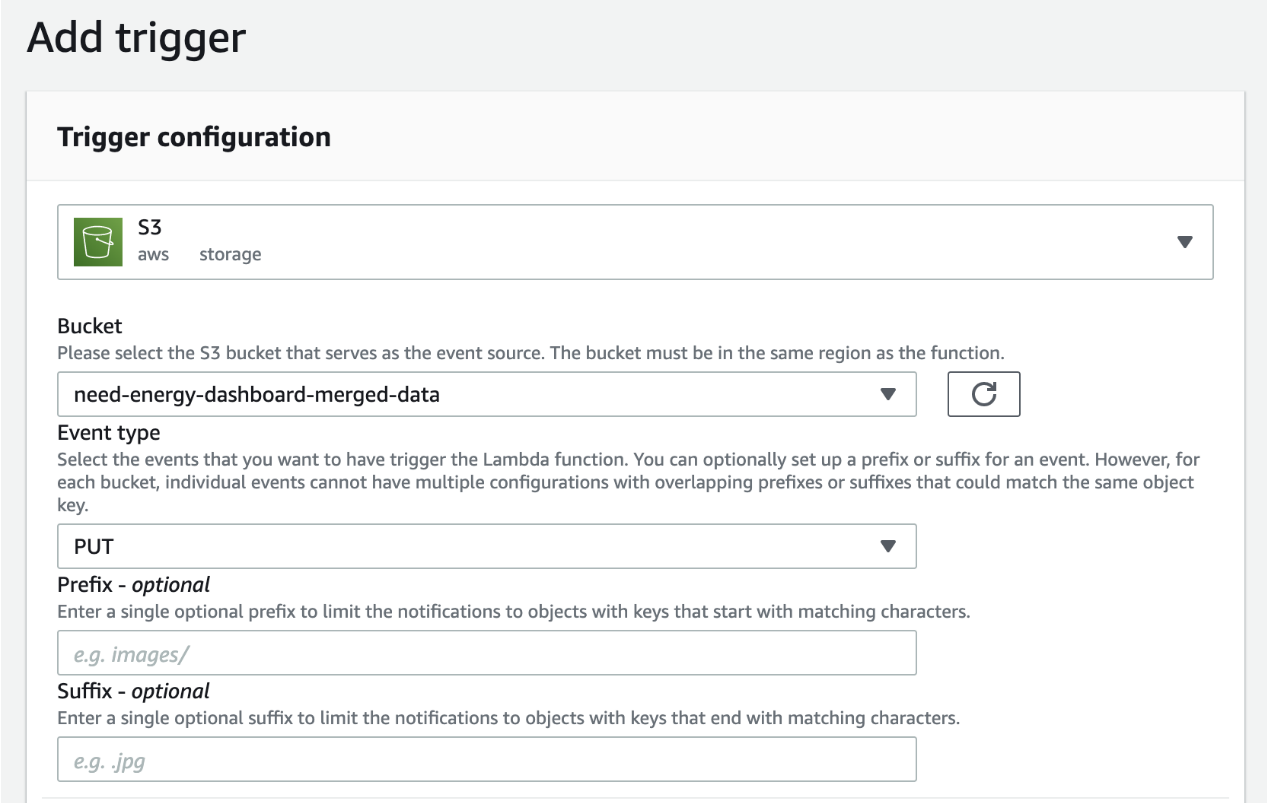
Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

dashboard_lambda

View the dashboard_lambda role on the IAM console.

- Add trigger as S3 Megered Bucket (Put Object Event)
  Updated: Put merged_47803 to be prefix:



Add trigger

Trigger configuration

S3
aws storage

Bucket

Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.

need-energy-dashboard-merged-data

Event type

Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

PUT

Prefix - *optional*

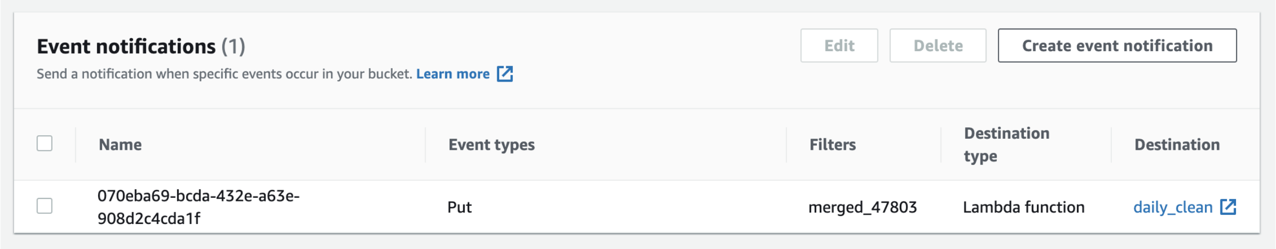
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.

e.g. images/

Suffix - *optional*

Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.

e.g. .jpg

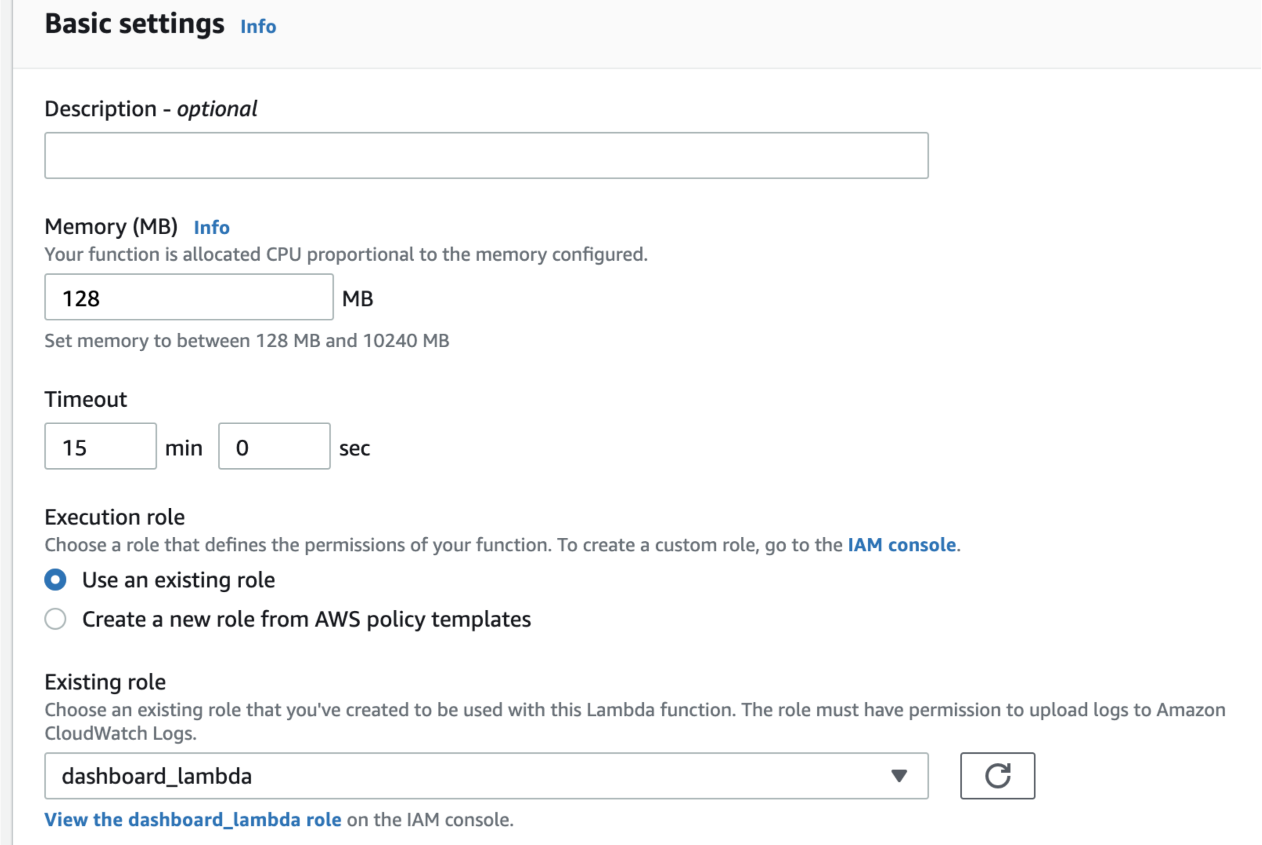


Event notifications (1)

Send a notification when specific events occur in your bucket. Learn more

□	Name	Event types	Filters	Destination type	Destination
□	070eba69-bcda-432e-a63e-908d2c4cda1f	Put	merged_47803	Lambda function	daily_clean

- Create Clean Function Scripts (Appendix C)
- Setup configuration: run out time to be 15 minutes



Basic settings Info

Description - *optional*

Memory (MB) Info

Your function is allocated CPU proportional to the memory configured.

128 MB

Set memory to between 128 MB and 10240 MB

Timeout

15 min 0 sec

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the **IAM console**.

● Use an existing role

○ Create a new role from AWS policy templates

Existing role

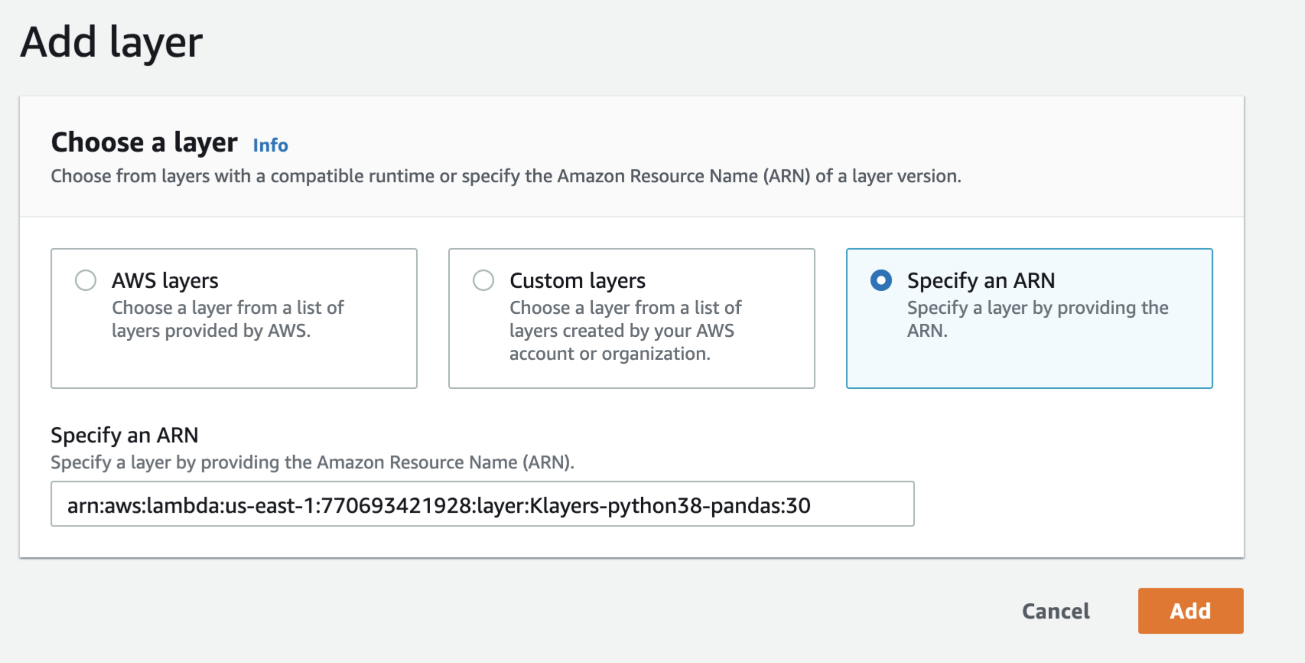
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

dashboard_lambda

View the dashboard_lambda role on the IAM console.

- Setup ARN layers - Because lambda does not have requests and pandas python library
  Check this article
  https://melissa-bain.medium.com/how-to-import-python-packages-in-aws-lambda-pandas-scipy-numpy-bb2c98c974e9

  Get ARN from here: (Make sure the region is right)
  https://github.com/keithrozario/Klayers/blob/master/deployments/python3.8/arns/us-east-1.csv



Add layer

Choose a layer Info

Choose from layers with a compatible runtime or specify the Amazon Resource Name (ARN) of a layer version.

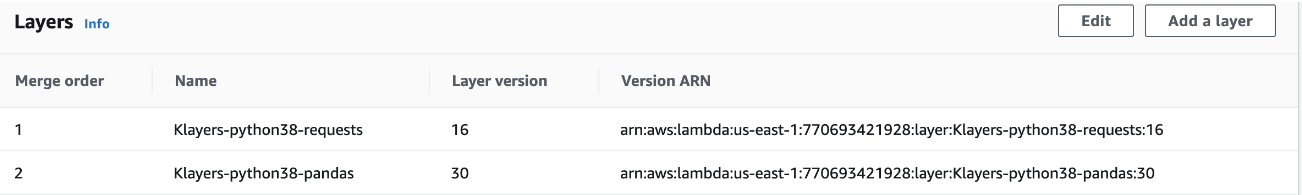
○ AWS layers
Choose a layer from a list of layers provided by AWS.

○ Custom layers
Choose a layer from a list of layers created by your AWS account or organization.

● Specify an ARN
Specify a layer by providing the ARN.

Specify an ARN
Specify a layer by providing the Amazon Resource Name (ARN).

arn:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-pandas:30

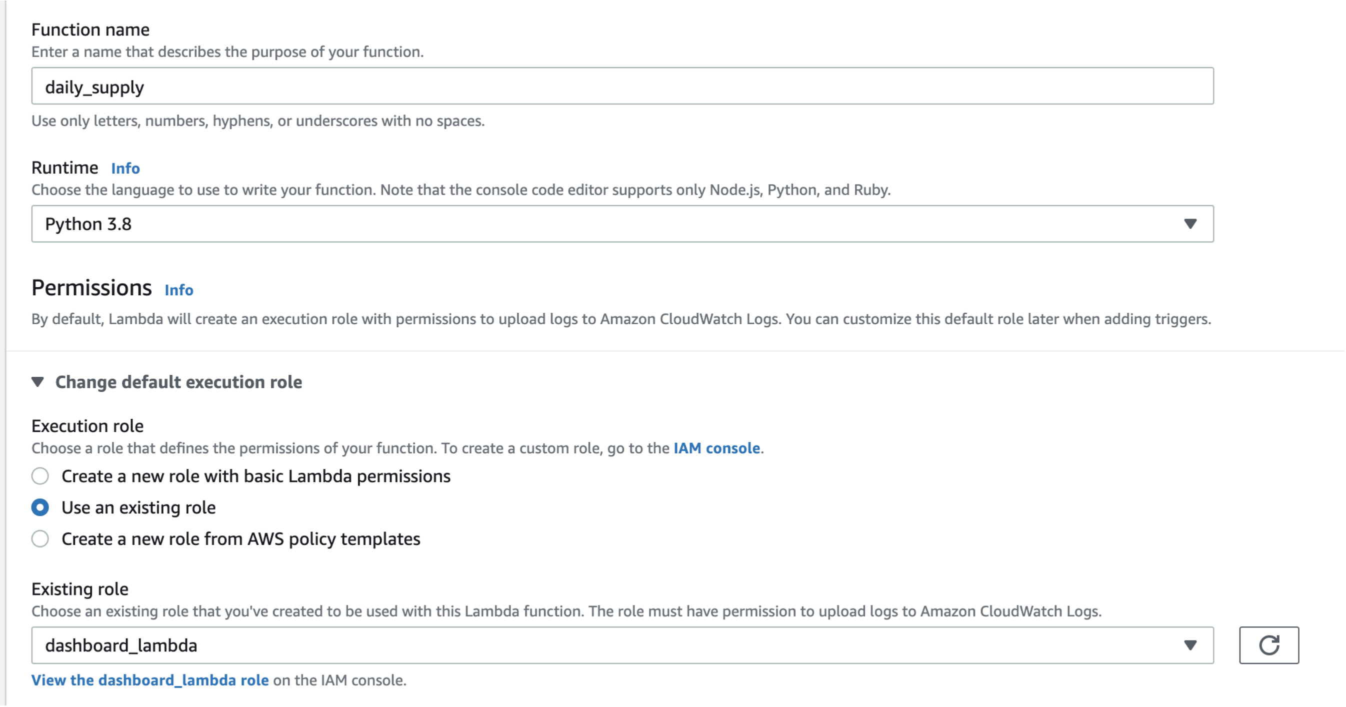


Layers Info

Merge order	Name	Layer version	Version ARN
1	Klayers-python38-requests	16	arn:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-requests:16
2	Klayers-python38-pandas	30	arn:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-pandas:30

3. Create Daily Supply Function

- Create Lambda Function



Function name

Enter a name that describes the purpose of your function.

daily_supply

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.8 ▼

Permissions Info

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.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the **IAM console**.

○ Create a new role with basic Lambda permissions

● Use an existing role

○ Create a new role from AWS policy templates

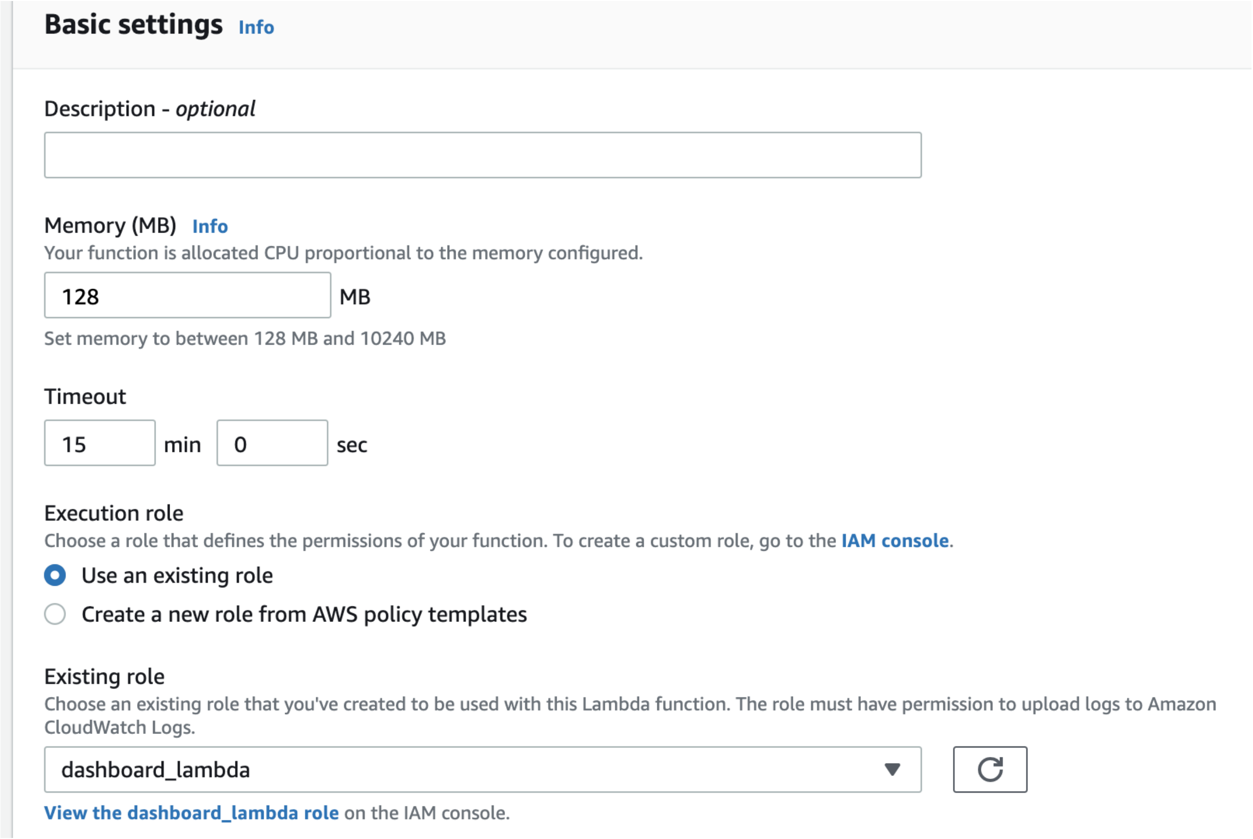
Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

dashboard_lambda ▼

View the dashboard_lambda role on the IAM console.

- Setup Lambda Trigger (Cloud Watch Section)
- Create Daily Supply Function Scripts (Appendix D)
- Setup configuration: run out time to be 15 minutes



Basic settings Info

Description - *optional*

Memory (MB) Info

Your function is allocated CPU proportional to the memory configured.

128 MB

Set memory to between 128 MB and 10240 MB

Timeout

15 min 0 sec

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

● Use an existing role

○ Create a new role from AWS policy templates

Existing role

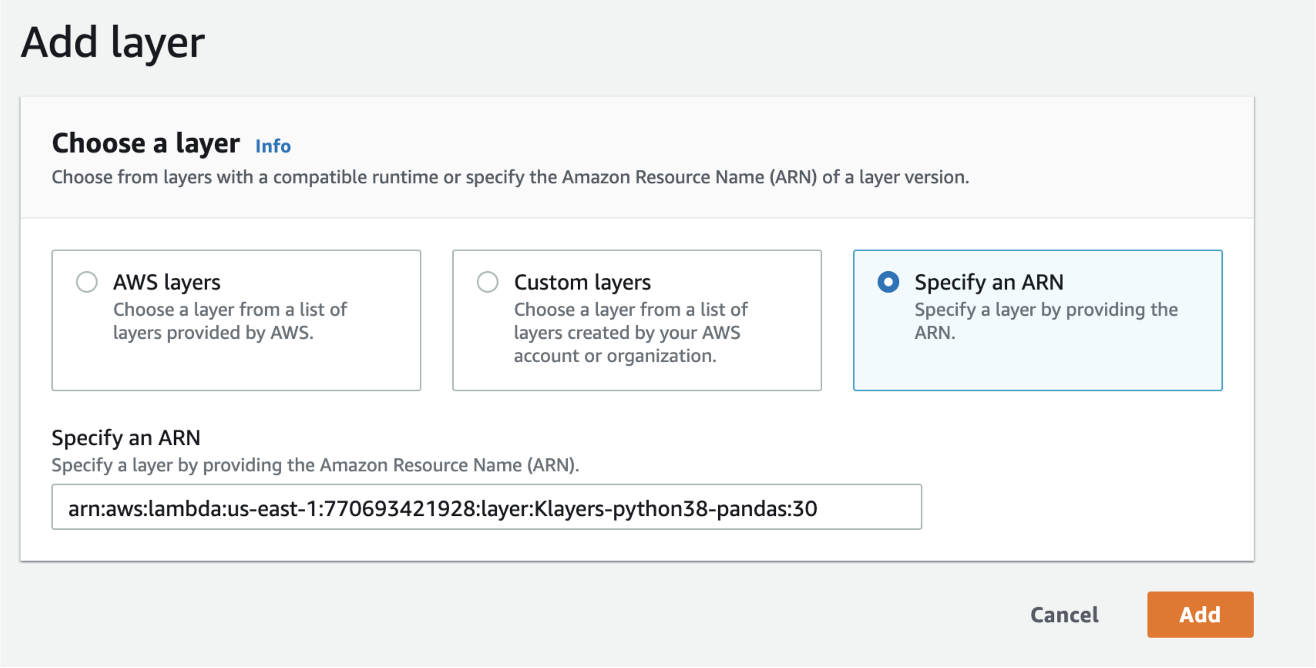
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

dashboard_lambda

View the dashboard_lambda role on the IAM console.

- Setup ARN layers **-** Because lambda does not have requests and pandas python library
  Check this article
  https://melissa-bain.medium.com/how-to-import-python-packages-in-aws-lambda-pandas-scipy-numpy-bb2c98c974e9

  Get ARN from here: (Make sure the region is right)
  https://github.com/keithrozario/Klayers/blob/master/deployments/python3.8/arns/us-east-1.csv



Add layer

Choose a layer Info

Choose from layers with a compatible runtime or specify the Amazon Resource Name (ARN) of a layer version.

○ AWS layers
Choose a layer from a list of layers provided by AWS.

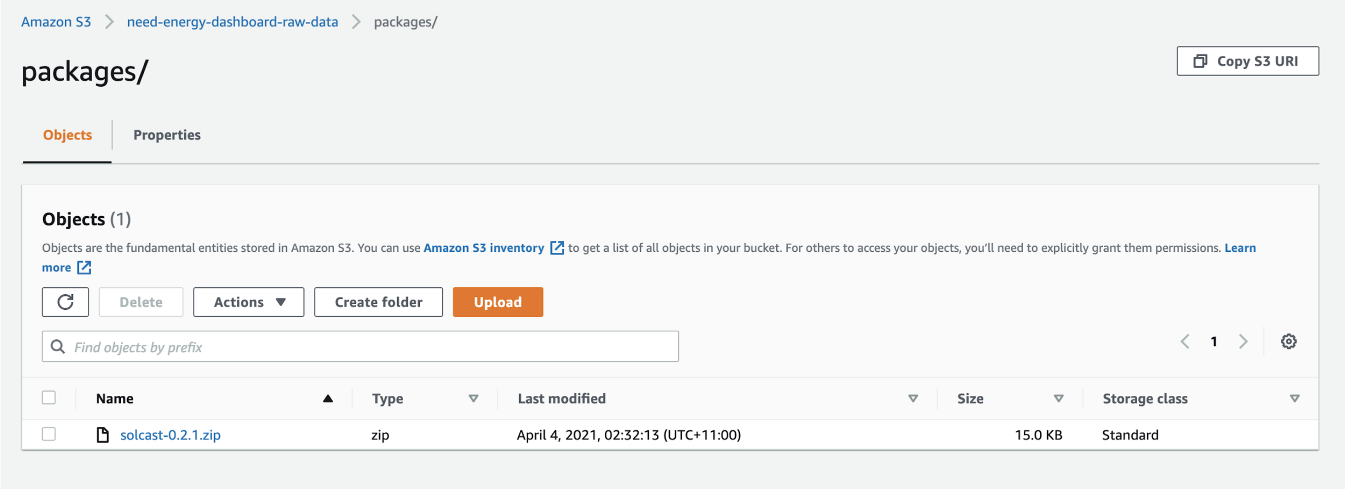
○ Custom layers
Choose a layer from a list of layers created by your AWS account or organization.

● Specify an ARN
Specify a layer by providing the ARN.

Specify an ARN
Specify a layer by providing the Amazon Resource Name (ARN).

arn:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-pandas:30

- Create Customer Layer (Because could not get ARN from above article
  - Add Package to S3 Bucket - > For files larger than 10 MB, consider uploading using Amazon S3.
  - s3://need-energy-dashboard-raw-data/packages/solcast-0.2.1.zip
  - Check the AWS configuration doc
  - Appendix E for creating zip layer



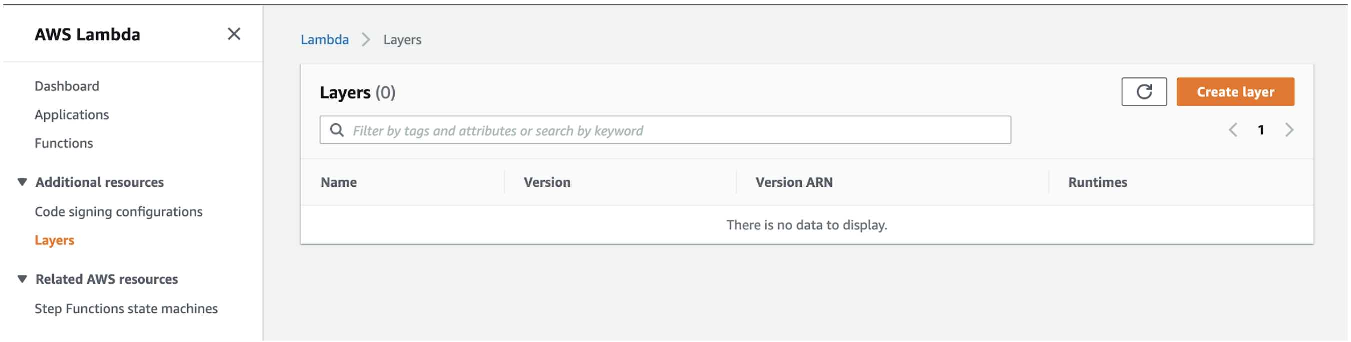
Amazon S3 > need-energy-dashboard-raw-data > packages/

packages/

Objects (1)

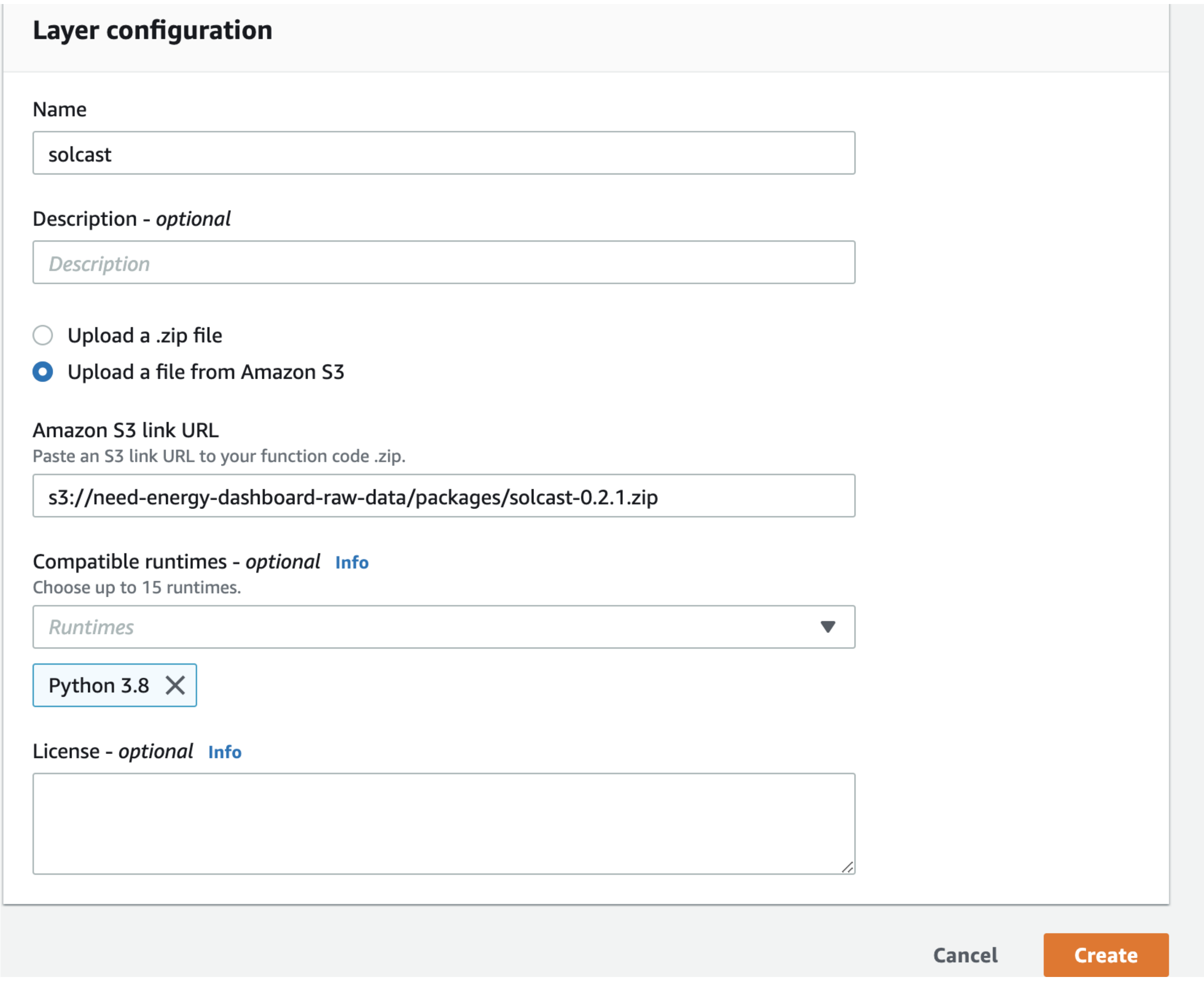
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name	Type	Last modified	Size	Storage class
□	solcast-0.2.1.zip	zip	April 4, 2021, 02:32:13 (UTC+11:00)	15.0 KB	Standard



Layers (0)

Name	Version	Version ARN	Runtimes
There is no data to display.			



Layer configuration

Name

solcast

Description - *optional*

Description

○ Upload a .zip file

● Upload a file from Amazon S3

Amazon S3 link URL

Paste an S3 link URL to your function code .zip.

s3://need-energy-dashboard-raw-data/packages/solcast-0.2.1.zip

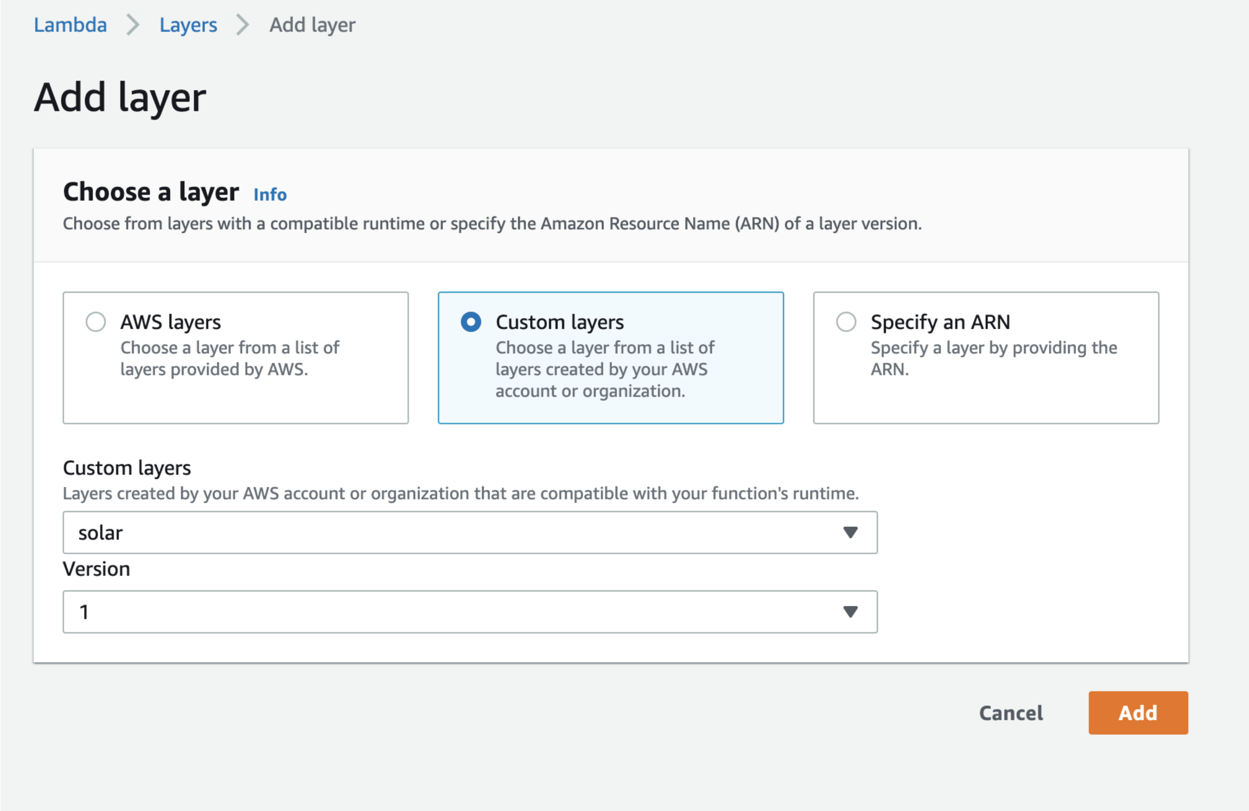
Compatible runtimes - *optional* Info

Choose up to 15 runtimes.

Runtimes

Python 3.8

License - *optional* Info



Lambda > Layers > Add layer

Add layer

Choose a layer Info

Choose from layers with a compatible runtime or specify the Amazon Resource Name (ARN) of a layer version.

○ AWS layers
Choose a layer from a list of layers provided by AWS.

● Custom layers
Choose a layer from a list of layers created by your AWS account or organization.

○ Specify an ARN
Specify a layer by providing the ARN.

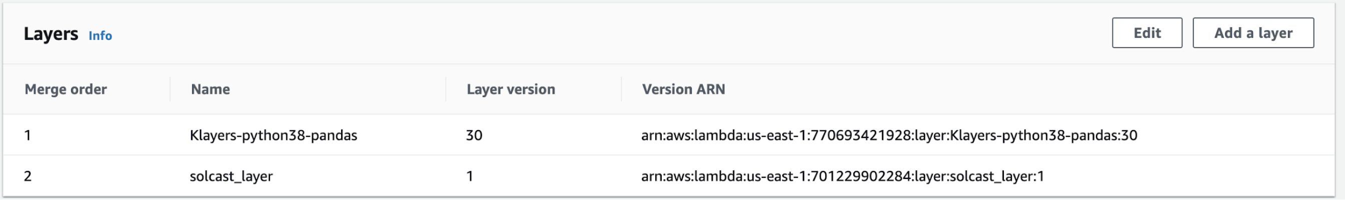
Custom layers
Layers created by your AWS account or organization that are compatible with your function's runtime.

solar

Version

1

Cancel Add



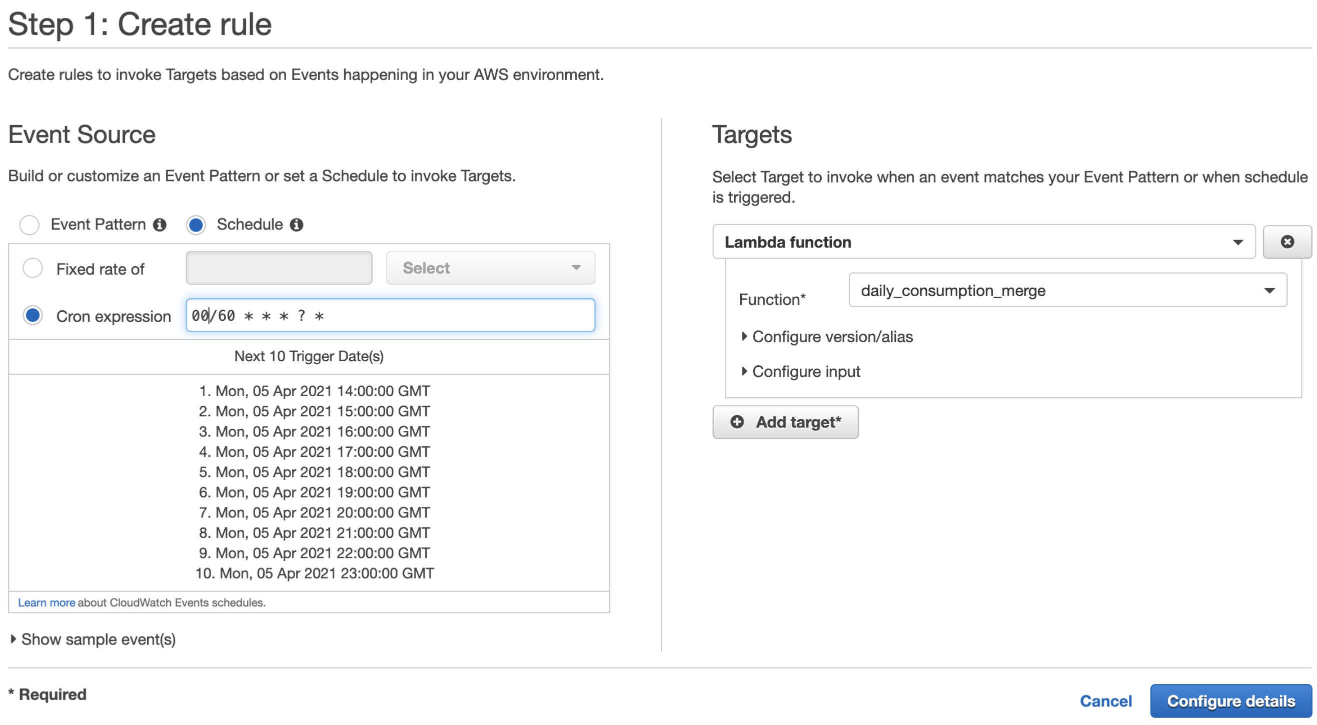
Layers Info

Edit Add a layer

Merge order	Name	Layer version	Version ARN
1	Klayers-python38-pandas	30	arn:aws:lambda:us-east-1:770693421928:layer:Klayers-python38-pandas:30
2	solcast_layer	1	arn:aws:lambda:us-east-1:701229902284:layer:solcast_layer:1

CloudWatch Setup

Daily_Consumption_Merge Lambda Function (Runs Hourly)



Step 1: Create rule

Create rules to invoke Targets based on Events happening in your AWS environment.

Event Source

Build or customize an Event Pattern or set a Schedule to invoke Targets.

○ Event Pattern ● Schedule

○ Fixed rate of [ ] Select

● Cron expression `00/60 * * * ? *`

Next 10 Trigger Date(s)

1. Mon, 05 Apr 2021 14:00:00 GMT
2. Mon, 05 Apr 2021 15:00:00 GMT
3. Mon, 05 Apr 2021 16:00:00 GMT
4. Mon, 05 Apr 2021 17:00:00 GMT
5. Mon, 05 Apr 2021 18:00:00 GMT
6. Mon, 05 Apr 2021 19:00:00 GMT
7. Mon, 05 Apr 2021 20:00:00 GMT
8. Mon, 05 Apr 2021 21:00:00 GMT
9. Mon, 05 Apr 2021 22:00:00 GMT
10. Mon, 05 Apr 2021 23:00:00 GMT

Learn more about CloudWatch Events schedules.

▸ Show sample event(s)

Targets

Select Target to invoke when an event matches your Event Pattern or when schedule is triggered.

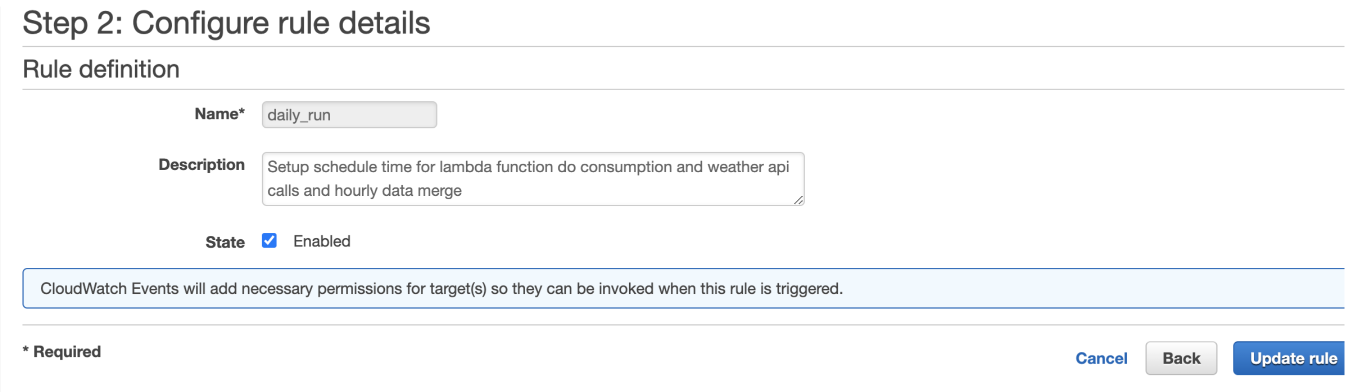
Lambda function

Function* daily_consumption_merge

▸ Configure version/alias

▸ Configure input

* Required



Step 2: Configure rule details

Rule definition

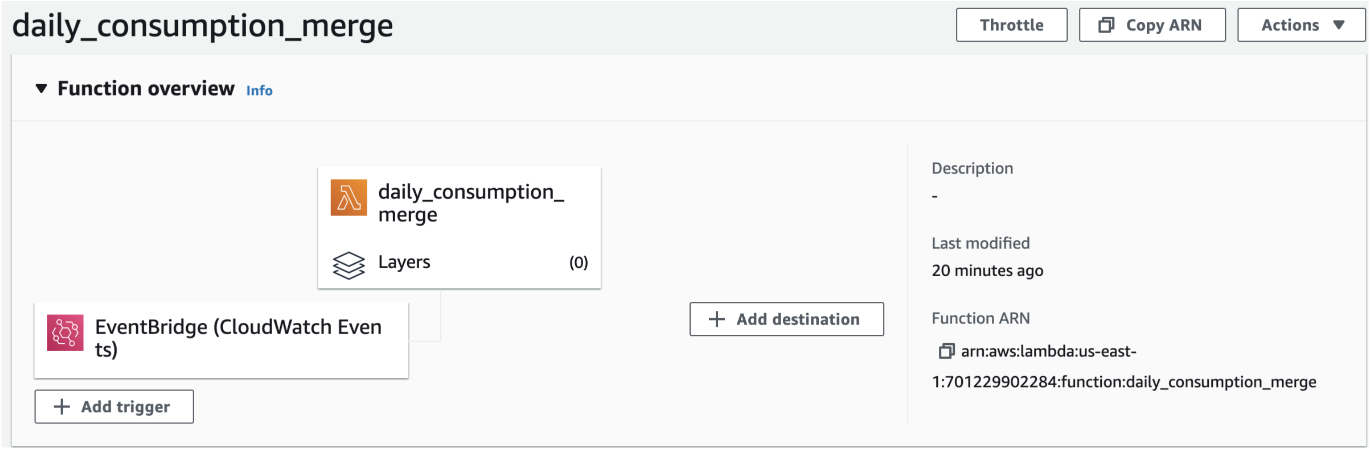
Name* daily_run

Description: Setup schedule time for lambda function do consumption and weather api calls and hourly data merge

State ☑ Enabled

CloudWatch Events will add necessary permissions for target(s) so they can be invoked when this rule is triggered.

* Required



daily_consumption_merge

Description

-

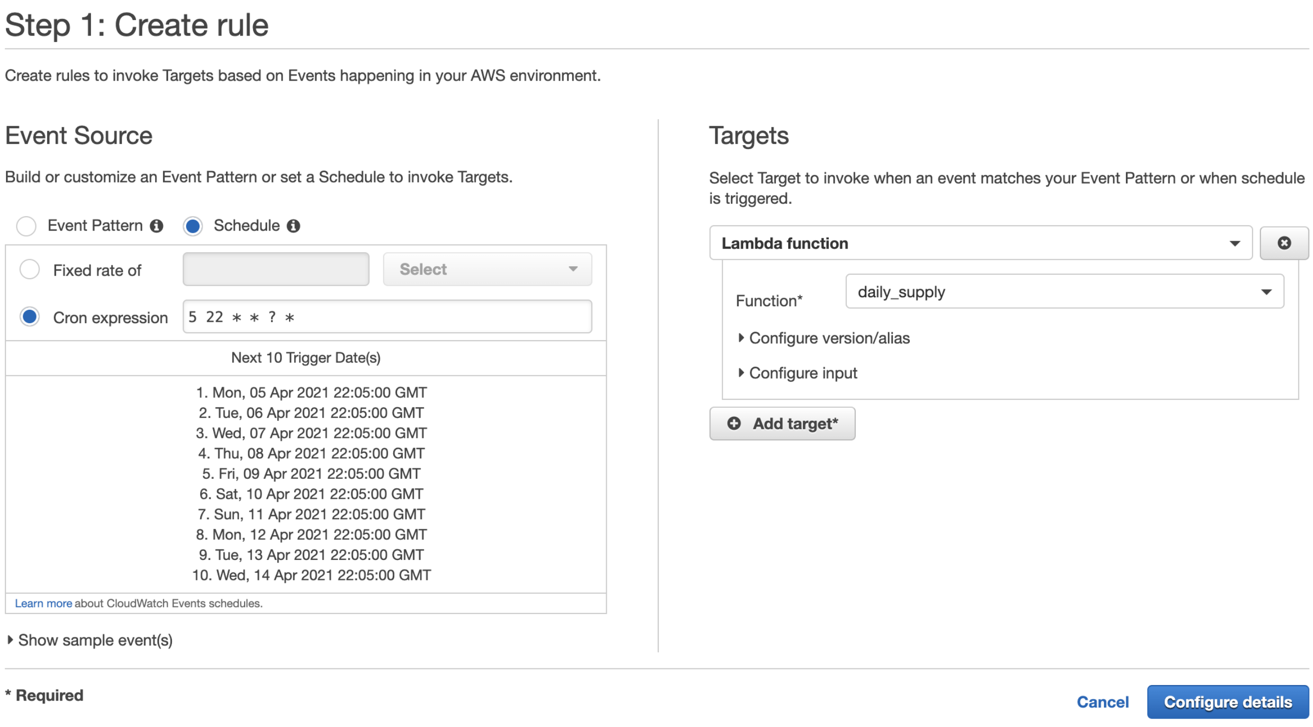
Last modified

20 minutes ago

Function ARN

arn:aws:lambda:us-east-1:701229902284:function:daily_consumption_merge

Daily_Supply Lambda Function (Runs Daily)



Step 1: Create rule

Create rules to invoke Targets based on Events happening in your AWS environment.

Event Source

Build or customize an Event Pattern or set a Schedule to invoke Targets.

○ Event Pattern ● Schedule

○ Fixed rate of [ ] Select

● Cron expression: `5 22 * * ? *`

Next 10 Trigger Date(s)

1. Mon, 05 Apr 2021 22:05:00 GMT
2. Tue, 06 Apr 2021 22:05:00 GMT
3. Wed, 07 Apr 2021 22:05:00 GMT
4. Thu, 08 Apr 2021 22:05:00 GMT
5. Fri, 09 Apr 2021 22:05:00 GMT
6. Sat, 10 Apr 2021 22:05:00 GMT
7. Sun, 11 Apr 2021 22:05:00 GMT
8. Mon, 12 Apr 2021 22:05:00 GMT
9. Tue, 13 Apr 2021 22:05:00 GMT
10. Wed, 14 Apr 2021 22:05:00 GMT

Learn more about CloudWatch Events schedules.

▸ Show sample event(s)

Targets

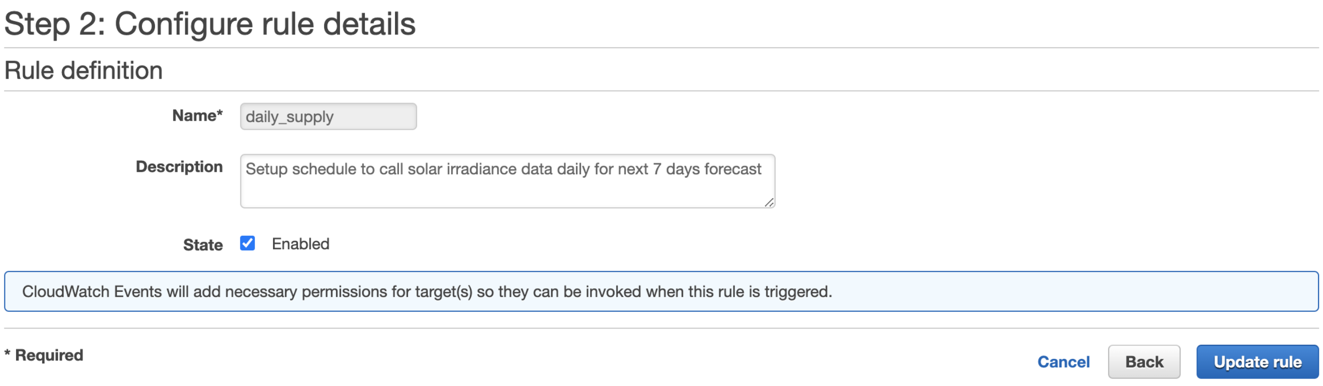
Select Target to invoke when an event matches your Event Pattern or when schedule is triggered.

Lambda function

Function*: daily_supply

▸ Configure version/alias

▸ Configure input



Step 2: Configure rule details

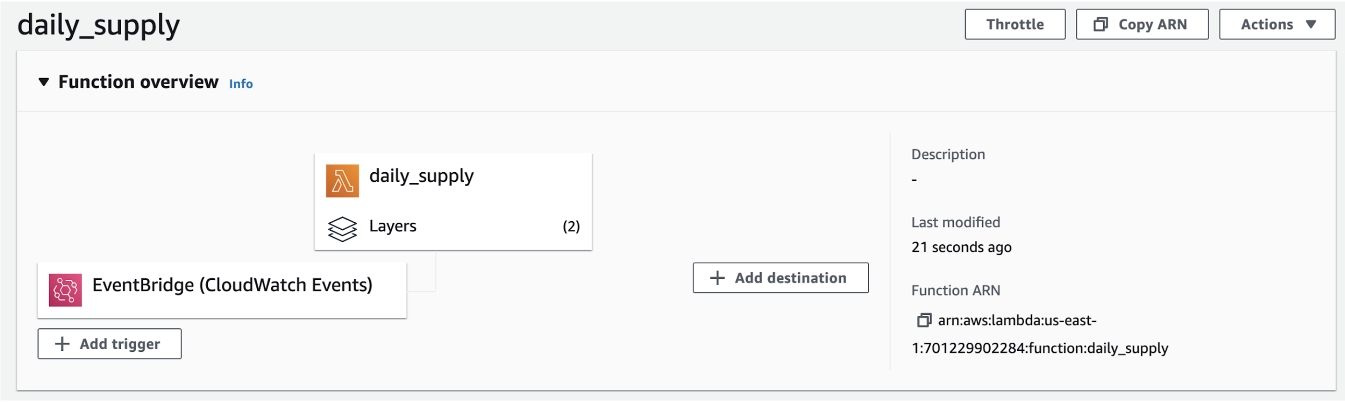
Rule definition

Name*: daily_supply

Description: Setup schedule to call solar irradiance data daily for next 7 days forecast

State: ☑ Enabled

CloudWatch Events will add necessary permissions for target(s) so they can be invoked when this rule is triggered.



daily_supply

▾ Function overview Info

Description: -

Last modified: 21 seconds ago

Function ARN: arn:aws:lambda:us-east-1:701229902284:function:daily_supply

S3 API Endpoint for Dashboard

Package Required

Python s3fs Package: S3FS builds on aiobotocore to provide a convenient Python filesystem interface for S3.

S3 API Endpoints

Consumption Data (Actual Data - Update Hourly)

- s3://need-energy-dashboard-merged-data/data_hourly_id_47740.csv
- s3://need-energy-dashboard-merged-data/data_hourly_id_47803.csv

Consumption Data (Static Data due to no API Access)

- s3://need-energy-dashboard-merged-data/MiniSub1_5min_clean.csv
  Period from 2021-01-18 18:30:00 - 2021-02-15 03:20:00
- s3://need-energy-dashboard-merged-data/Croydon_5min_clean.csv
  Period from 2021-03-17 15:00:00 - 2021-04-08 03:15:00
- s3://need-energy-dashboard-merged-data/Feltex_5min_clean.csv
- Period from 2021-03-17 15:00:00 - 2021-04-08 03:20:00

Consumption Data (Cleaned Data - Update Hourly)

- s3://need-energy-dashboard-cleaned-data/data_hourly_id_47740_cleaned.csv
  Static data for modelling due to lots of missing data has been dropped
- s3://need-energy-dashboard-cleaned-data/data_hourly_id_47803_cleaned.csv
  Update hourly

Supply Data (Forecast Data - Update Daily)

- s3://need-energy-dashboard-cleaned-data/solar_data_7_days_forecast.csv

Static Data (PV Sizing Tool)

- s3://need-energy-dashboard-cleaned-data/harare_solcast_mean_meteorological_year.csv

Appendix

A. Public Holiday Generators

1. Public_holiday_generator_SA.py

```
from datetime import date, datetime
import pandas as pd
import holidays
def get_holidays(startYear = 2018, endYear = 2025, countryCode = 'ZA'):
    """
    Takes in a start and end date, and start and end year.

    Produces a dataframe with a daily date and columns:
    holiday - 'Y' for holiday
    holidayName - name of the holiday if holiday is 'Y'

    Returns a dataframe
    """

    holidayDict = {}
    for i in range(startYear, endYear):
        for date, name in sorted(holidays.CountryHoliday(countryCode,years=[i]).items()):
            holidayDict[date] = name
        holiday_df = pd.DataFrame(list(holidayDict.items()),columns = ['day','holidayName'])
        holiday_df['day'] = pd.to_datetime(holiday_df['day']).dt.date
    return holiday_df

def get_days(start = '1/1/2018',startYear = 2018, end = '31/12/2025',endYear = 2025,
countryCode = 'ZA'):
    """
    Takes in a start and end date, and start and end year.

    Produces a dataframe with a daily date and columns:
    weekend - 'Y' for weendend and 'N' for workday
    dayOfweek - numerical day of the week identifier 0 for monday
    weekNum - numerical number of the week
    holiday - 'Y' for holiday
    holidayName - name of the holiday if holiday is 'Y'

    Returns a dataframe
    """

    #generate the range of daily dates
    dates = pd.date_range(start = start, end = end)
    date_df = pd.DataFrame(dates, columns = ['day'])
    date_df['day'] = pd.to_datetime(date_df['day'])
    country_holidays = get_holidays(startYear = startYear, endYear = endYear, countryCode =
countryCode)

    date_df['dayName'] = pd.DatetimeIndex(date_df['day']).day_name()
    date_df['dayOfWeek'] = date_df['day'].dt.dayofweek
    date_df['weekend'] = date_df['dayOfWeek'].apply(lambda x: 'Y' if x>4 else 'N')
    date_df['weekNum'] = date_df['day'].dt.week
    date_df['holiday'] = date_df['day'].apply(lambda x: 'Y' if x in country_holidays['day'].values
else 'N')
    date_df['day'] = date_df['day'].dt.date
    date_df = date_df.merge(country_holidays, on='day', how='left', indicator=False)

    date_df.to_csv(f'../public_holidays_weekends.csv', index=False)
    return date_df

start = '1/1/1994'
startYear = 1994
end = '31/12/2021'
endYear = 2021
countryCode = 'ZA'
get_days(start = start, startYear = startYear, end = end, endYear = endYear, countryCode =
countryCode)
```

2. Public_holiday_generator_ZIMBABWE.py

```
from datetime import date, datetime
import pandas as pd

def get_holidays(startYear = 2018, endYear = 2025):
    """
    Takes in a start and end date, and start, end year
    holidays package does not contain ZIMBABWE, which need manually setup

    Produces a dataframe with a daily date and columns:
    holiday - 'Y' for holiday
    holidayName - name of the holiday if holiday is 'Y

    Returns a dataframe
    """

    holidayDict = {u'2018-01-01': 'New Year''s Day',
 u'2018-02-21': 'Robert Mugabe National Youth Day',
 u'2018-03-30': 'Good Friday',
 u'2018-03-31': 'Easter Saturday',
 u'2018-04-01': 'Easter Sunday',
 u'2018-04-02': 'Easter Monday',
 u'2018-04-18': 'Independence Day',
 u'2018-05-01': 'Workder''s Day',
 u'2018-05-25': 'Africa Day',
 u'2018-06-30': 'Polling Day',
 u'2018-08-13': 'Heroes'' Day',
 u'2018-08-14': 'Defense Forces Day',
 u'2018-12-22': 'National Unity Day',
 u'2018-12-25': 'Christmas Day',
 u'2018-12-26': 'Boxing Day',

 u'2019-01-01': 'New Year''s Day',
 u'2019-02-21': 'Robert Mugabe National Youth Day',
 u'2019-04-18': 'Good Friday',
 u'2019-04-19': 'Easter Saturday',
 u'2019-04-20': 'Easter Sunday',
 u'2019-04-21': 'Easter Monday',
 u'2019-04-22': 'Independence Day',
 u'2019-05-01': 'Workder''s Day',
 u'2019-05-25': 'Africa Day',
 u'2019-08-12': 'Heroes'' Day',
 u'2019-08-13': 'Defense Forces Day',
 u'2019-10-25': 'Solidarity Day Against Sanctions',
 u'2019-12-23': 'National Unity Day',
 u'2019-12-25': 'Christmas Day',
 u'2019-12-26': 'Boxing Day',

 u'2020-01-01': 'New Year''s Day',
 u'2020-02-21': 'Robert Mugabe National Youth Day',
 u'2020-04-10': 'Good Friday',
 u'2020-04-11': 'Easter Saturday',
 u'2020-04-12': 'Easter Sunday',
 u'2020-04-13': 'Easter Monday',
 u'2020-04-18': 'Independence Day',
 u'2020-05-01': 'Workder''s Day',
 u'2020-05-25': 'Africa Day',
 u'2020-08-10': 'Heroes'' Day',
 u'2020-08-11': 'Defense Forces Day',
 u'2020-12-22': 'National Unity Day',
 u'2020-12-25': 'Christmas Day',
 u'2020-12-26': 'Boxing Day',

 u'2021-01-01': 'New Year''s Day',
 u'2021-02-21': 'Robert Mugabe National Youth Day',
 u'2021-04-02': 'Good Friday',
 u'2021-04-03': 'Easter Saturday',
 u'2021-04-04': 'Easter Sunday',
 u'2021-04-05': 'Easter Monday',
 u'2021-04-18': 'Independence Day',
 u'2021-05-01': 'Workder''s Day',
 u'2021-05-25': 'Africa Day',
 u'2021-08-09': 'Heroes'' Day',
 u'2021-08-10': 'Defense Forces Day',
 u'2021-12-22': 'National Unity Day',
 u'2021-12-25': 'Christmas Day',
 u'2021-12-26': 'Boxing Day',

 u'2022-01-01': 'New Year''s Day',
 u'2022-02-21': 'Robert Mugabe National Youth Day',
 u'2022-04-15': 'Good Friday',
 u'2022-04-16': 'Easter Saturday',
 u'2022-04-17': 'Easter Sunday',
 u'2022-04-18': 'Easter Monday',
 u'2022-05-01': 'Workder''s Day',
 u'2022-05-25': 'Africa Day',
 u'2022-08-08': 'Heroes'' Day',
 u'2022-08-09': 'Defense Forces Day',
 u'2022-12-22': 'National Unity Day',
 u'2022-12-26': 'Boxing Day',
 u'2022-12-27': 'Christmas Day',

 u'2023-01-02': 'New Year''s Day',
 u'2023-02-21': 'Robert Mugabe National Youth Day',
 u'2023-04-07': 'Good Friday',
 u'2023-04-08': 'Easter Saturday',
 u'2023-04-09': 'Easter Sunday',
 u'2023-04-10': 'Easter Monday',
 u'2023-04-18': 'Independence Day',
 u'2023-05-01': 'Workder''s Day',
 u'2023-05-25': 'Africa Day',
 u'2023-08-14': 'Heroes'' Day',
 u'2023-08-15': 'Defense Forces Day',
 u'2023-12-22': 'National Unity Day',
 u'2023-12-25': 'Christmas Day',
 u'2023-12-26': 'Boxing Day',

 u'2024-01-01': 'New Year''s Day',
 u'2024-02-21': 'Robert Mugabe National Youth Day',
 u'2024-03-29': 'Good Friday',
 u'2024-03-30': 'Easter Saturday',
 u'2024-03-31': 'Easter Sunday',
 u'2024-04-01': 'Easter Monday',
 u'2024-04-18': 'Independence Day',
 u'2024-05-01': 'Workder''s Day',
 u'2024-05-25': 'Africa Day',
 u'2024-08-12': 'Heroes'' Day',
 u'2024-08-13': 'Defense Forces Day',
 u'2024-12-23': 'National Unity Day',
 u'2024-12-25': 'Christmas Day',
 u'2024-12-26': 'Boxing Day',
 u'2025-01-01': 'New Year''s Day',
 u'2025-02-21': 'Robert Mugabe National Youth Day',
 u'2025-04-18': 'Good Friday',
 u'2025-04-19': 'Easter Saturday',
 u'2025-04-20': 'Easter Sunday',
 u'2025-04-21': 'Easter Monday',
 u'2025-05-01': 'Workder''s Day',
 u'2025-05-26': 'Africa Day',
 u'2025-08-11': 'Heroes'' Day',
 u'2025-08-12': 'Defense Forces Day',
 u'2025-12-22': 'National Unity Day',
 u'2025-12-25': 'Christmas Day',
 u'2025-12-26': 'Boxing Day'}
    holiday_df = pd.DataFrame(holidayDict.items(), columns = ['day','holidayName'])
    holiday_df['day'] = pd.to_datetime(holiday_df['day'])

    return holiday_df

def get_days(start = '1/1/2018',startYear = 2018, end = '31/12/2025',endYear = 2025):
    """
    Takes in a start and end date, and start and end year.

    Produces a dataframe with a daily date and columns:
    weekend - 'Y' for weendend and 'N' for workday
    dayOfweek - numerical day of the week identifier 0 for monday
    weekNum - numerical number of the week
    holiday - 'Y' for holiday
    holidayName - name of the holiday if holiday is 'Y'

    Returns a dataframe
    """

    #generate the range of daily dates
    dates = pd.date_range(start = start, end = end)
    date_df = pd.DataFrame(dates, columns = ['day'])
    date_df['day'] = pd.to_datetime(date_df['day'])
    country_holidays = get_holidays(startYear = startYear, endYear = endYear)

    date_df['dayName'] = pd.DatetimeIndex(date_df['day']).day_name()
    date_df['dayOfWeek'] = date_df['day'].dt.dayofweek
    date_df['weekend'] = date_df['dayOfWeek'].apply(lambda x: 'Y' if x>4 else 'N')
    date_df['weekNum'] = date_df['day'].dt.week
    date_df['holiday'] = date_df['day'].apply(lambda x: 'Y' if x in list(country_holidays['day'])
else 'N')

    date_df = date_df.merge(country_holidays, on='day', how='left', indicator=False)

    date_df.to_csv(f'../public_holidays_weekends_ZIMBABWE.csv', index=False)
    return date_df
```

```
start = '1/1/2018'
startYear = 2018
end = '31/12/2025'
endYear = 2025
get_days(start = start, startYear = startYear, end = end, endYear = endYear)
```

B. Hourly Consumption Merge Lambda Function

1. hourly_consumption.py

```
from datetime import datetime
import requests
import pandas as pd
import json
from datetime import datetime,timedelta
from dateutil import tz

HERE_TZ = tz.tzlocal()
ZIMBABWE_TZ = tz.gettz('UTC+2')
COLUMNS = ['date', 'timestamp', 'consumption', 'solar', 'alwaysOn', 'gridImport',
       'gridExport', 'selfConsumption', 'selfSufficiency', 'active',
       'reactive', 'voltages', 'phaseVoltages', 'currentHarmonics',
       'voltageHarmonics']
AGGREGATION_CODES = {"five_min":"1", "hourly":"2", "daily":"3",
                     "monthly":"4", "quarterly":"5", "ten_min":"6",
                     "fifteen_min":"7", "twenty_min":"8", "thirty":"9"}


def collect_data(aggregation_type, service_location_id,new_tz):
    # Setting up variables
    aggregation_code = AGGREGATION_CODES[aggregation_type]

    from_ = int((datetime.now() - timedelta(hours=1)).replace(minute=0,
second=0).timestamp()) * 1000
    to_ = int((datetime.now()).replace(minute=0, second=0).timestamp()) * 1000

    # Making request
    URL =
"https://app1pub.smappee.net/dev/v3/servicelocation/{}/consumption?aggregation={}&from={
}&to={}"
    url = URL.format(service_location_id, aggregation_code, from_, to_)
    payload={}
    headers = {
      'Authorization': 'Bearer ab299760-c43d-320c-8be0-1bb74a643a8b'
    }
    response = requests.request("GET", url, headers=headers, data=payload)
    # Json Serializing
    data = json.loads(response.text)

    # Converting to pandas and saving .csv file
    try:
        data_df = pd.DataFrame(data['consumptions'])
        data_df['date'] = data_df['timestamp'].apply(lambda x: datetime.fromtimestamp(x/1000)
                                                .astimezone(new_tz)
                                                .replace(tzinfo=None))
        data_df = data_df[COLUMNS]
        file_name = f'data_{aggregation_type}_id_{service_location_id}_timestamp_{from_}.csv'
        # data_df.to_csv(file_name, index=False)
    except KeyError as e:
        columns =
['date','timestamp','consumption','solar','alwaysOn','gridImport','gridExport','selfConsumption','
selfSufficiency','active','reactive','voltages','phaseVoltages','currentHarmonics','voltageHarmo
nics']
        data_df = pd.DataFrame(columns = columns)
        file_name = f'data_{aggregation_type}_id_{service_location_id}_timestamp_{from_}.csv'
        print('I got a KeyError - reason "%s"' % str(e))
    return file_name, data_df

def main():
    # aggregation_types = ['five_min', 'hourly', 'daily', 'monthly']
    aggregation_types = ['hourly']
    service_location_ids = [
    '47740',# Puma Rhodesville,
    '47803'# Puma HQ# #
    ]
    new_tz = tz.gettz('UTC+2')

    file_names = []
    data_dfs = []
    for service_location_id in service_location_ids:
        for aggregation_type in aggregation_types:
            file_name, data_df = collect_data(aggregation_type=aggregation_type,
service_location_id=service_location_id,new_tz=new_tz)
            file_names.append(file_name)
            data_dfs.append(data_df)
    return service_location_ids, file_names, data_dfs

if __name__ == "__main__":
    main()
```

2. hourly_weather.py

```
# import and install packages
import requests
from datetime import datetime
from dateutil import tz
import pandas as pd

def currentday(base_url, lat, lon, units, api_key, new_tz):
    weather_df = pd.DataFrame()
    today = datetime.now()
    new_time = str(int((today).timestamp()))
    api_call = base_url + "lat=" + lat + "&lon=" + lon + "&dt=" + new_time + "&units=" + units +
"&appid=" + api_key

    # get method of requests module
    # return response object
    response = requests.get(api_call)

    # json method of response object
    # convert json format data into
    # python format data
    weather_json = response.json()

    # load data to dataframe and do timezone conversion
    hourly_df = pd.DataFrame(weather_json['hourly'])
    hourly_df['datetime'] = hourly_df['dt'].apply(lambda x: datetime.fromtimestamp(x)
                                                  .astimezone(new_tz)
                                                  .replace(tzinfo=None))
    weather_df = weather_df.append(hourly_df)

    weather_df.sort_values(by='datetime', inplace=True)
    file_name = f'weather_hourly_SA_timestamp_{new_time}.csv'
    # weather_df.to_csv(file_name, index=False)

    return file_name,weather_df

def main():
    # setup parameters check this https://openweathermap.org/api/one-call-api#data
    api_key = '7f0c1694586e146cd38d4d1863743fbd'
    base_url = 'https://api.openweathermap.org/data/2.5/onecall/timemachine?'

    # Harare, South Africa
    lat = str(-17.82772)
    lon = str(31.05337)
    new_tz = tz.gettz('UTC+2')

    # Temperature is available in Fahrenheit, Celsius and Kelvin units.
    # Wind speed is available in miles/hour and meter/sec.
    # For temperature in Fahrenheit and wind speed in miles/hour, use units=imperial
    # For temperature in Celsius and wind speed in meter/sec, use units=metric
    # Temperature in Kelvin and wind speed in meter/sec is used by default
    units = 'metric'

    file_name, weather_df = currentday(base_url, lat, lon, units, api_key, new_tz)

    return file_name, weather_df[-2:-1]

if __name__ == "__main__":
    main()
```

3. hourly_merge.py

```
import pandas as pd
import numpy as np
from datetime import datetime

def main(consumption_data, weather_data, public_holidays_data, service_location_id):
    # Process consumption data
    df = consumption_data.astype({'date':'datetime64[ns]'}).rename(columns={'date':'datetime'}).set_index('datetime')
    df = pd.DataFrame(df['consumption'])
    df = df.asfreq('1H')

    # Convert consumption column to kWH (its a more common metric than Wh)
    df['consumption'] = df['consumption']/1000
    df.rename(columns={'consumption':'consumption_kWh'}, inplace=True)

    # Add season column
    df['date'] = df.index.strftime('%Y-%m-%d')
    df['year'] = df.index.year
    df['dayOfYear'] = df.index.dayofyear
    df['month'] = df.index.month
    df['monthName'] = df.index.month_name()
    df['week'] = df.index.isocalendar().week
    df['day'] = df.index.day
    df['dayName'] = df.index.day_name()
    df['hour'] = df.index.hour
    df['minute'] = df.index.minute
    df['dayOfWeek'] = df.index.dayofweek
    df['weekend'] = df['dayOfWeek'].apply(lambda x: 1 if x >= 5 else 0)
    df['time'] = df.index.time
    df['dayOfMonth'] = df.index.strftime('%m-%d')
    df['hourMinute'] = df.index.strftime('%H:%M')

    bins = [0,4,8,12,16,20,24]
    #labels = ['Late Night', 'Early Morning','Morning','Noon','Eve','Night']
```

```
labels = [1, 2,3,4,5,6]
df['session'] = pd.cut(df['hour'], bins=bins, labels=labels, include_lowest=True)

def season_df(df):
    if df['month'] == 12 | df['month'] == 1 | df['month'] == 2:
        return 2 #'Summer'
    elif df['month'] == 3 | df['month'] == 4 | df['month'] == 5:
        return 3 #'Autumn'
    elif df['month'] == 6 | df['month'] == 7 | df['month'] == 8:
        return 4 #'Winter'
    else:
        return 1 #'Spring'

df['season'] = df.apply(season_df, axis = 1)

# Process weather data
weather_df = weather_data.astype({'datetime':'datetime64[ns]'})
weather_df = weather_df[['temp', 'humidity', 'clouds','datetime']].set_index('datetime')
weather_df = weather_df.asfreq('1H')

# Rename and divide by 100 to make it more ML friendly
weather_df['clouds'] = weather_df['clouds']/100
weather_df.rename(columns={'clouds':'cloud_cover'}, inplace=True)

# Temperature in degrees C, rename with units
weather_df.rename(columns={'temp':'temp_degreeC'}, inplace=True)

# Humidity is relative humidity as a %
# Rename and divide by 100 to make it more ML friendly
weather_df['humidity'] = weather_df['humidity']/100
weather_df.rename(columns={'humidity':'rel_humidity'}, inplace=True)

# Process holiday data
holiday_df = public_holidays_data
holiday_df = holiday_df[['day','holiday','holidayName']]
holiday_df.rename(columns = {'day':'date'},inplace=True)

# Merge all datasets
combined_df = df.join(weather_df)

combined_df['date'] = pd.to_datetime(combined_df['date'], utc = False)
holiday_df['date'] = pd.to_datetime(holiday_df['date'], utc = False)
combined_df = pd.merge(combined_df.reset_index(), holiday_df)
combined_df = combined_df.rename(columns={'index':'datetime'}).set_index('datetime')

# Replace Holiday 'Y' with 1
# Replace Holiday NaN with 0
combined_df['holiday'] = np.where(combined_df['holiday']=='Y',1,0)
# Add workingday or non-working day column
```

```
    combined_df['workingDay'] = np.where(np.logical_and(combined_df['weekend']==0,
combined_df['holiday']==0),1,0)

    today = datetime.now()
    new_time = str(int((today).timestamp()))
    file_name = f'merged_{service_location_id}_timestamp_{new_time}.csv'
    return file_name, combined_df
```

4. lambda_function.py

```
import boto3
import pandas as pd
import hourly_consumption as dc
import hourly_weather as dw
import hourly_merge as dm
from io import StringIO # python3; python2: BytesIO

def lambda_handler(event, context):
    s3_r = boto3.resource("s3")  # pointer to AWS S3 service
    s3_c = boto3.client('s3')

    bucket_name = "need-energy-dashboard-raw-data" # bucket for saving our data
    bucket_to = "need-energy-dashboard-merged-data"

    # run daily api to generate daily consumption and save historical merged consumption
data to merged
    service_location_ids, consumption_files, consumption_dfs = dc.main()
    for i in range(0,2):
        consumption_file = consumption_files[i]
        consumption_df = consumption_dfs[i]
        service_location_id =  service_location_ids[i]

        # consumption_buffer = StringIO()
        # consumption_df.to_csv(consumption_buffer, date_format="%Y-%m-%d %H:%M:%S")
        # write object to bucket
        # s3_r.Object(bucket_name,
f'consumption/{consumption_file}').put(Body=consumption_buffer.getvalue())

        histical_file_name = f'data_hourly_id_{service_location_id}.csv'
        histical_file_obj = s3_c.get_object(Bucket=bucket_to, Key=histical_file_name)
        histical_file_df = pd.read_csv(histical_file_obj['Body'])
        histical_file_df = histical_file_df.append(consumption_df)
        cols_to_keep =
['date','timestamp','consumption','solar','alwaysOn','gridImport','gridExport','selfConsumption','
selfSufficiency','active','reactive','voltages','phaseVoltages','currentHarmonics','voltageHarmo
nics']
        histical_buffer = StringIO()
        histical_file_df.loc[:, cols_to_keep].to_csv(histical_buffer, date_format="%Y-%m-%d
%H:%M:%S")
        # write object to bucket
        s3_r.Object(bucket_to, f'{histical_file_name}').put(Body=histical_buffer.getvalue())

    # run daily api to generate daily weather
    weather_buffer = StringIO()
    weather_file, weather_df = dw.main()
    weather_df.to_csv(weather_buffer,date_format="%Y-%m-%d %H:%M:%S")
    # write object to bucket
    s3_r.Object(bucket_name, f'weather/{weather_file}').put(Body=weather_buffer.getvalue())

    # get public holiday object and file (key) from bucket
    SA_holiday_file_name = 'public-holiday/public_holidays_weekends_SA_1994-2025.csv'
    SA_public_holidays_obj = s3_c.get_object(Bucket=bucket_name,
Key=SA_holiday_file_name)
    SA_public_holidays_df = pd.read_csv(SA_public_holidays_obj['Body'])

    ZA_holiday_file_name =
'public-holiday/public_holidays_weekends_ZIMBABWE_2018_2025.csv'
    ZA_public_holidays_obj = s3_c.get_object(Bucket=bucket_name,
Key=ZA_holiday_file_name)
    ZA_public_holidays_df = pd.read_csv(ZA_public_holidays_obj['Body'])

    # run merge function to merge all datasets
    for i in range(0,2):
        consumption_file = consumption_files[i]
        consumption_df = consumption_dfs[i]
        service_location_id =  service_location_ids[i]
        merged_buffer = StringIO()
        if service_location_id == '47803':
            merged_file, merged_df = dm.main(consumption_df,
weather_df,SA_public_holidays_df, service_location_id)
            merged_df.to_csv(merged_buffer,date_format="%Y-%m-%d %H:%M:%S")
            s3_r.Object(bucket_to, f'{merged_file}').put(Body=merged_buffer.getvalue())

    # print results
    return f"Succeed!"
```

C. Hourly Consumption Clean Lambda Function

1. hourly_clean_HQ.py

```
import pandas as pd
import numpy as np

def main(merged_data):
    df = merged_data.set_index('datetime')

    # Drop duplication
    df = df[~df.index.duplicated(keep = 'first')]

    # Impulate missing consumption data using average values for weekend, month and hour
    profile_df = df.groupby([df['weekend'], df['monthName'], df['hour']]).mean()
    for month in df[df['consumption_kWh']==0]['monthName'].unique():
        weekday_avg = profile_df.loc[0, month]['consumption_kWh']
        weekend_avg = profile_df.loc[1, month]['consumption_kWh']
        df[(df['monthName'] == month) & (df['weekend'] == 0) & (df['consumption_kWh'] == 0)] =
weekday_avg.values
        df[(df['monthName'] == month) & (df['weekend'] == 1) & (df['consumption_kWh'] == 0)] =
weekend_avg.values

    # impulate missing weather data using backward fill
    weather_features = ['temp_degreeC', 'rel_humidity', 'cloud_cover']
    for feature in weather_features:
        df[feature] = df[feature].fillna(method = 'bfill')

    # Use simple forward fill to replace negative cloud cover values
    df['cloud_cover']= df['cloud_cover'].mask(df['cloud_cover']<0).ffill(downcast='infer')

    return df
```

2. lambda_function.py

```
import json
import hourly_clean_HQ as dcHQ
from io import StringIO # python3; python2: BytesIO
import boto3
import pandas as pd

def lambda_handler(event, context):
    s3_r = boto3.resource("s3")  # pointer to AWS S3 service
    s3_c = boto3.client('s3')
    bucket_name = "need-energy-dashboard-cleaned-data"

    csv_buffer = StringIO()
    # get daily merged file
    source_bucket = event['Records'][0]['s3']['bucket']['name']
    source_key = event['Records'][0]['s3']['object']['key']

    merged_obj = s3_c.get_object(Bucket=source_bucket, Key=source_key)
    merged_df = pd.read_csv(merged_obj['Body'])

    # get historical data
    cleaned_file_name = 'data_hourly_id_47803_cleaned.csv'
    cleaned_obj = s3_c.get_object(Bucket=bucket_name, Key=cleaned_file_name)
    cleaned_df = pd.read_csv(cleaned_obj['Body'])

    # merge daily and historical data
    combined_df = cleaned_df.append(merged_df)

    # run clean function to do data wrangling for modelling
    new_cleaned_df = dcHQ.main(combined_df)
    new_cleaned_df.to_csv(csv_buffer)

    # write object to bucket
    s3_r.Object(bucket_name, f'{cleaned_file_name}').put(Body=csv_buffer.getvalue())

    # print result
    return f"Succeed! Find your new {f'{cleaned_file_name}'} file in {bucket_name} bucket.)"
```

D. Daily Supply Lambda Function

1. daily_supply.py

```
import pandas as pd
import solcast
from dateutil import tz
from datetime import datetime

API_KEY = '8bQiJkGE--ukTUA2l20--y1S3QYa1wEg'
latitude = -17.82772
longitude = 31.05337
new_tz = tz.gettz('UTC+2')

def main():
    today = datetime.now()
    new_time = str(int((today).timestamp()))

    data = solcast.get_radiation_forecasts(latitude, longitude, API_KEY)
    seven_day_forecast = data.forecasts
    data_df = pd.DataFrame(seven_day_forecast)
    data_df['period_end'] = data_df['period_end'].apply(lambda x:
x.astimezone(new_tz).replace(tzinfo=None))

    file_name = f'solar_data_7_days_forecast.csv'
    return file_name, data_df

if __name__ == "__main__":
    main()
```

2. lambda_function.py

```
import json
import daily_supply as ds
import boto3
from io import StringIO # python3; python2: BytesIO

def lambda_handler(event, context):
    s3 = boto3.resource("s3")  # pointer to AWS S3 service
    bucket_name = "need-energy-dashboard-cleaned-data" # bucket for saving our data

    # run api daily to generate 7 days forecast solar irradiance data
    supply_buffer = StringIO()
    supply_file, supply_df = ds.main()
    supply_df.to_csv(supply_buffer)
    # write object to bucket
    s3.Object(bucket_name, f'{supply_file}').put(Body=supply_buffer.getvalue())

    # print results
    return f"Succeed! Find your new {f'{supply_file}'} file in {bucket_name} bucket."
```

E. Zip Solcast Locally for Lambda Custom Layer

```
$ mkdir python
$ cd python
$ pip3 install solcast -t .
$ tree -L 1
$ cd ..
$ zip -r solcast_layer.zip python
```