How to query MOGREPS NetCDF4 Files with

Athena on AWS

Author: Margaret Cheng
Date: 09/27/2018

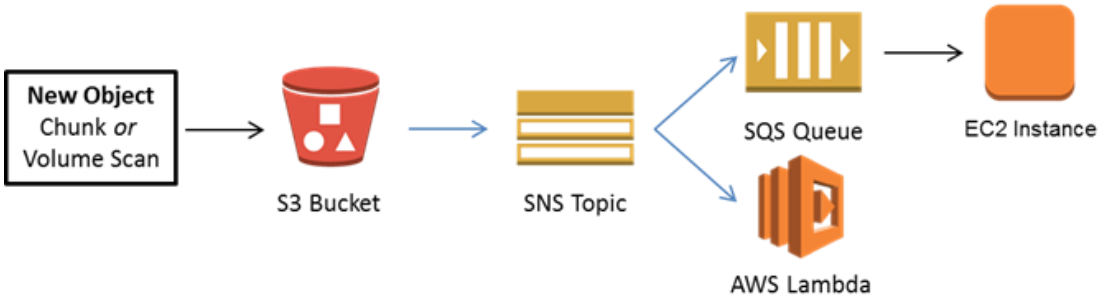
This article provides a secure and cost-effective solution to query MOGREPS high-resolution weather forecast data on AWS Athena that make the data processing highly available, highly responsive, and timely available.

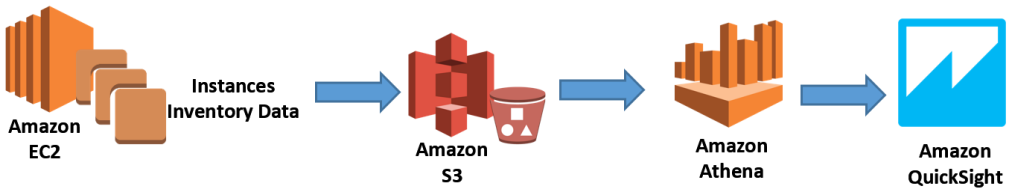
The first section describes the basic steps to set up Python and dependent software packages in order to ingest and transform MOBREPS-UK weather forecast data, which are in the netCDF format, on AWS S3 bucket.

The second section demonstrates how to get the latest netCDF files in AWS S3, how to ingest unstructured netCDF data, transform it to another file format (e.g. txt, csv, json) and prepare it for AWS Athena query. In this document, JSON file format is used as an example to demonstrate source dataset query in Athena.

To enhance the real-time data processing, Amazon Simple Notification Service (SNS) can be used to get new data as soon as it is available. The third section demonstrates how to use SNS to get new netCDF file as soon as it is available and how to use Amazon Simple Queue Service (SQS) to decouple incoming jobs from pipeline processes. In this scenario, messages are pushed to multiple subscribers, which eliminates the need to periodically check or poll for updates and enables parallel asynchronous processing of data by the subscribers.

Application Architecture





Prerequisites

This document assumes that the user has basic knowledge on how to launch an Amazon EC2 instance and access a Linux instance such as:

1) How to set up an Amazon Machine Image (AMI) via the Amazon Web Services (AWS) Management Console
2) How to create appropriate security credentials (i.e. Amazon EC2 Key Pairs) in order to establish a secure SSH connection to a created instance.
3) Having in record the AWS Access Key ID with the Secret Access Key.

You also need to create a S3 Bucket for storing the transformed data.

Notes:
a) AWS provides a step-by-step user guide that walks through the processes from signing up for AWS account to launching an EC2 instance.
b) By default the AWS user guide selects an 64-bit Amazon Linux AMI (Amazon Machine Image) to create the EC2 instance. In this example, Amazon Linux Ubuntu 16.04 is used.

Basic Prerequisites:
1) An Amazon AMI with Ubuntu Server 16.04 (Long Term Storage) LTS 64-bit with a t2.micro instance type.
2) An Amazon S3 bucket for source data (In this case, MOGREPS-UK S3 bucket located at s3://mogreps-uk in eu-west-2 region)
5) An SSH client (e.g. "putty" in windows/unix, Git Bash tool, or "terminal" in mac) to connect to your EC2 instance.
6) A basic knowledge about Linux commands.

Tools to be installed on EC2 for data ingestion and transformation using built-in Python 2.7 are as below:

Main tools:

- HDF5 1.10.1
- NetCDF4 4.6.1
- NetCDF4-Python Interface 1.4.1

Other Python libraries:

- Setuptools
- Wget
- Cftime
- Cython
- Pandas
- Numpy
- matplotlib
- Pathlib
- Boto3

Install Tools to Read MOGREPS-UK NetCDF files with Python on AWS

This section provides steps to install HDF5 and NetCDF4 libraries for data processing in Python.

Prerequisites:

Install updates and required libraries before installing HDF5 and netCDF4. Use SSH client to connect to your EC2 instance. Once connected to EC2, type below commands to install libraries.

Step 1: Using any of your preferred SSH client, login to your Ubuntu Amazon EC2 instance (use ubuntu as username) using your private key and Public DNS address. e.g.

```
$ ssh -i AWS.pem ubuntu@ec2-XX-YYY-ZZZ-UUU.us-west-2.compute.amazonaws.com
```

Step 2: To keep your sudo privileges active throughout the session, type

$ sudo -i

At this point, you may want to update the local package index with the latest changes made in the repositories by typing the following:

$ apt-get update

Step 3: Install required packages and related dependencies.

$ apt-get install make gcc g++ curl libxml2 libxml2-dev libssl-dev libcurl3 libcurl4-gnutls-dev openssl pkg-config

$apt-get update

$apt-get install m4

$apt install python-pip

$pip install pandas

$pip install matplotlib

$python -m pip install --upgrade pip setuptools wheel

$apt-get install python-numpy

$pip install pathlib

$pip install boto3

$pip install cftime

$ apt-get install zlib1g-dev

$ apt-get install curl

$apt-get install cython

Main Steps:

Step 1: Download and Install the HDF5 library

$ sudo -i

$ cd /usr/local/src/

$ wget http://www.hdfgroup.org/ftp/HDF5/current/src/hdf5-1.10.1.tar.gz

$ tar -xzvf hdf5-1.10.1.tar.gz

$ cd hdf5-1.10.1

$ ./configure --prefix=/usr/localapt

$ make

$ make install

Notes: To view the full list of the options, use "./configure --help".

Step 2: Install the NetCDF4 library

The NetCDF4 library can be installed in a similar way.

```
$ sudo -i
$ cd /usr/local/src/
$ wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4.6.1.tar.gz
$ tar -xzvf netcdf-4.6.1.tar.gz
$ cd netcdf-4.6.1

$ ./configure    --prefix=/usr/local
$ make
$ make install
```

Step 3: Install the NetCDF4-Python Interface

```
$ sudo -i
$ cd /usr/local/src/
$ wget
https://files.pythonhosted.org/packages/d9/c0/653b79fcea4efc9a79ce3ae95a31c1669f312ab0c53b3d45037c4e419c2e/netCDF4-1.4.1.tar.gz
$ tar -xzvf netCDF4-1.4.1.tar.gz
$ cd netCDF4-cd 1.4.1
$ python setup.py build
$ python setup.py install
$ cd test
$ python run_all.py
```

After running the run_all.py script, the console shows below tools have been installed.

```
netcdf4-python version: 1.4.1
HDF5 lib version:          1.10.1
netcdf lib version:        4.6.1
numpy version                1.11.0
cython version               0.23.4
```

Prepare Data for Querying as soon as it arrives in S3 Bucket.

Boto3 and Lambda can be used to check for the latest netCDF files in MOGREPS S3 bucket. The process involves uploading your code to AWS Lambda, set up your code to trigger from MOGREPS S3 bucket, then have Lambda runs your code only when triggered. This section provides sample code to show how to create an automated process in Python using Lambda and Boto3 to check for latest netCDF4 files in S3 and download it to local EC2 directory for processing.

Locate AWS credentials for Boto3:

Create a credentials file in ~/.aws/ directory. The shared credentials file (~/.aws/credentials) can have multiple profiles defined as below:

```
[default]

aws_access_key_id=foo

aws_secret_access_key=bar

[dev]

aws_access_key_id=foo2

aws_secret_access_key=bar2

[prod]

aws_access_key_id=foo3

aws_secret_access_key=bar3
```

For more details, see Boto3 credentials document.

Use Boto3 to Sort S3 bucket by last modified

Below is the sample code to sort files in S3 bucket by last modified.

```
import boto3
# Define a lambda to get the last modified time
```

```
get_last_modified = lambda obj: int(obj['LastModified'].strftime('%s'))
#Get the latest .nc files in MOGREPS S3 bucket using Boto3
s3 = boto3.resource('s3', use_ssl=False, region_name="eu-west-2")
bucket = s3.Bucket('mogreps-uk')
unsorted = []
for file in bucket.objects.filter(Prefix='prods_op_mogreps-uk_20140101'):
    unsorted.append(file)

files = [obj.key for obj in sorted(unsorted, key=get_last_modified,
      reverse=True)][0:9]
# download the latest .nc file to EC2
for filename in files:
    file_in = '/mnt/' + filename
    print "Copy .nc file to EC2 " , file_in
    bucket.download_file(filename, file_in)
```

Notes: As a security best practice, consider using rotating access keys to regularly change the access keys for IAM users in your account.

Read and Transform the MOGREPS-UK netCDF File in Python.

Once a new netCDF file in MOGREPS S3 bucket has been detected, it can start processing and transforming netCDF file. This sample code shows how to process netCDF (.nc) file and output interesting variables to JSON file and store it in a S3 bucket for querying. Note that you need to create a S3 bucket (In this case, s3://mogreps-uk-json) before processing the .nc file. The transformed data (.json) will be stored in this S3 bucket.

```
import boto3
import json
import os
import sys
import netCDF4
from itertools import groupby
```

```
from operator import attrgetter
from pathlib import Path
from dateutil.parser import parse
from datetime import datetime
import uuid
import xarray as xr

filename = 'prods_op_mogreps-uk_20130101_03_00_006.nc'
source_bucket = 'mogreps-uk'
local_dir = '/mnt/'
output_bucket = 'mogreps-uk-json'
file_in = local_dir + filename

# Open file in a netCDF reader
nc = netCDF4.Dataset(file_in, 'r')

#Look at the variables
#print(nc.variables.keys())

#Look at the dimensions
#print(nc.dimensions)

#Look at interesting variables
interesting_variables = {vname: v for vname, v in nc.variables.items() if
'grid_mapping' in v.ncattrs()}
#for name, var in interesting_variables.items():
#        print(name)
#        print(var)

groupby(interesting_variables.values(), attrgetter('dimensions'))

groups = []
uniquekeys = []
data = interesting_variables.values()
keyfunc = attrgetter('dimensions')
data = sorted(data, key=keyfunc)
for k, g in groupby(data, keyfunc):
    groups.append(list(g))          # Store group iterator as a list
```

```
    uniquekeys.append(k)

#print(uniquekeys)
#print(groups)

group_names = [[var.name for var in group] for group in groups]
list(zip(uniquekeys, group_names))

def generate_products(nco, dims, variables):
    prod_name = '_'.join(v.name for v in variables)
    return {
        'name': prod_name,
        'description': prod_name,
        'metadata_type': 'eo',
        'metadata': {
            'platform': {'code': 'mogreps'},
            'product_type': prod_name,
            'format': {
                'name': 'NETCDF'
            }
        },
        'measurements': [
            {'name': v.name,
             'dtype': str(v.dtype),
             'units': str(v.units),
             'nodata': v._FillValue}
            for v in variables
        ]
    }

prods = [generate_products(nc, keys, variables)
            for keys, variables in zip(uniquekeys, groups)]
#print(prods)


def find_bounds(filename, dims):
    bounds = {}
    with xr.open_dataset(filename) as ds:
```

```
            for dim in dims:
                coord = ds[dim]
                if coord.axis == 'X':
                    bounds['left'] = float(coord.min())
                    bounds['right'] = float(coord.max())
                elif coord.axis == 'Y':
                    bounds['top'] = float(coord.max())
                    bounds['bottom'] = float(coord.min())
                elif coord.axis == 'T':
                    bounds['start'] = coord.min().data
                    bounds['end'] = coord.max().data
        return bounds

bounds = find_bounds(file_in, ('time', 'grid_latitude', 'grid_longitude'))

#print(bounds)

def make_dataset(filename, dims, variables):
        bounds = find_bounds(filename, dims)

        p = Path(filename)

        mtime = datetime.fromtimestamp(p.stat().st_mtime)
        return {
            'id': str(uuid.uuid4()),
            'processing_level': 'modelled',
            'product_type': 'gamma_ray',
            'creation_dt': mtime.isoformat(),
            'extent': {
                'coord': {
                    'ul': {'lon': bounds['left'], 'lat': bounds['top']},
                    'ur': {'lon': bounds['right'], 'lat': bounds['top']},
                    'll': {'lon': bounds['left'], 'lat': bounds['bottom']},
                    'lr': {'lon': bounds['right'], 'lat': bounds['bottom']},
                },
                'from_dt': str(bounds['start']),
                'to_dt': str(bounds['end']),
            },
```

```
            'format': {'name': 'NETCDF'},
            'image': {
                  'bands': {
                        vname: {
                              'path': filename,
                              'layername': vname,
                        } for vname in variables
                  }
            },
            'lineage': {'source_datasets': {}},
      }

odc_ds = make_dataset(file_in, uniquekeys[1], group_names[1])

#serialize and write to json file to S3
output_key = filename.replace('.nc', '.json')
output_path = "s3://" + output_bucket + '/' + output_key
print "Write transformed data to ", output_path

#write data to an S3 object using boto3
obj = s3.Object(output_bucket,output_key)
obj.put(Body=json.dumps(odc_ds))
```

The above Python code output the transformed data in JSON format to S3 bucket (s3://mogreps-uk-json). Below is the sample output JSON file.


```
{
  "lineage": {
    "source_datasets": {

    }
  },
  "product_type": "gamma_ray",
  "extent": {
    "from_dt": "2013-01-01T06:04:59.999982080",
    "to_dt": "2013-01-01T09:00:00.000000000",
    "coord": {
      "ul": {
```

```
        "lat": 7.1600494384765625,
        "lon": 354.9107360839844
      },
      "ll": {
        "lat": -3.779949903488159,
        "lon": 354.9107360839844
      },
      "lr": {
        "lat": -3.779949903488159,
        "lon": 363.31072998046875
      },
      "ur": {
        "lat": 7.1600494384765625,
        "lon": 363.31072998046875
      }
    }
  },
  "format": {
    "name": "NETCDF"
  },
  "image": {
    "bands": {
      "stratiform_snowfall_rate": {
        "path":
"/mnt/s3-mogreps-uk/prods_op_mogreps-uk_20130101_03_00_006.nc",
        "layername": "stratiform_snowfall_rate"
      },
      "stratiform_rainfall_rate": {
        "path":
"/mnt/s3-mogreps-uk/prods_op_mogreps-uk_20130101_03_00_006.nc",
        "layername": "stratiform_rainfall_rate"
      }
    }
  },
  "id": "611d36cd-1d2a-4a1a-9fd8-9c051832e7d7",
  "processing_level": "modelled",
  "creation_dt": "2017-06-21T14:36:06"
}
```

Query MOGREPS-UK Data with AWS Athena

What is Athena?

Athena is an interactive query service that makes it easy to analyze data directly in Amazon Simple Storage Service (Amazon S3) using standard SQL. With a few actions in the AWS Management Console, you can point Athena at your data stored in Amazon S3 and begin using standard SQL to run ad-hoc queries and get results in seconds.

This makes it perfect for a variety of standard data formats, including CSV, JSON, ORC, and Parquet. You now need to supply Athena with information about your data and define the schema for your logs with a Hive-compliant DDL statement. Athena uses Presto, a distributed SQL engine, to run queries. It also uses Apache Hive DDL syntax to create, drop, and alter tables and partitions. Athena uses an approach known as schema-on-read, which allows you to use this schema at the time you execute the query. Essentially, you are going to be creating a mapping for each field in the log to a corresponding column in your results.

Why Athena?

Amazon Athena is the best tool to query geospatial data like MOGREPS weather forecast. It is highly interactive, highly secure and highly available.

Amazon Athena is highly secure because it uses IAM policies to restrict access to Athena operations. Encryption options enable you to encrypt query result files in Amazon S3 and query data encrypted in Amazon S3. Users must have the appropriate permissions to access the Amazon S3 locations and decrypt files.

Athena has robust functionality to query geospatial data. To use geospatial functions in Athena, input your data in the WKT format, or use the Hive JSON SerDe. You can also use the geometry data types supported in Athena.

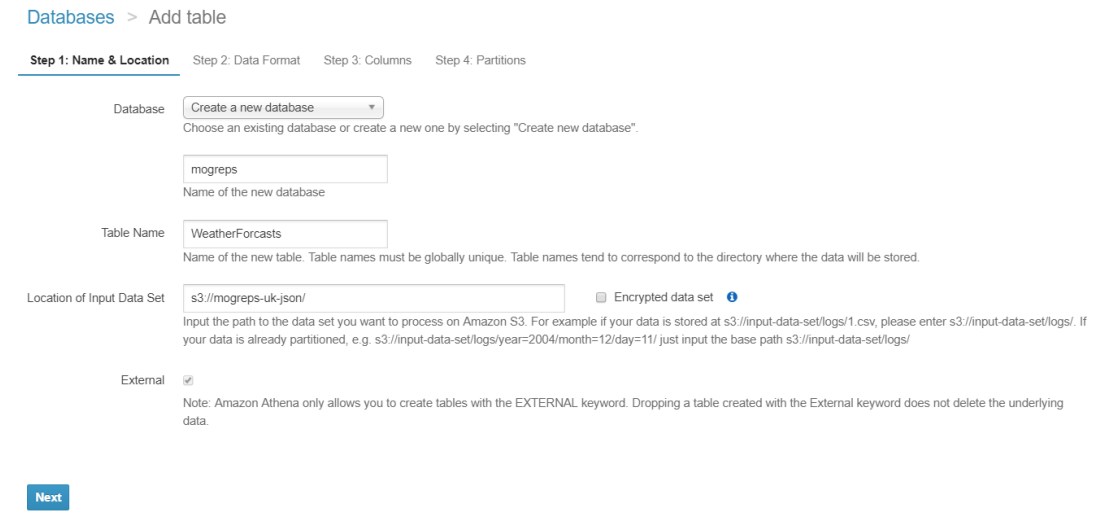
Query data in AWS Athena

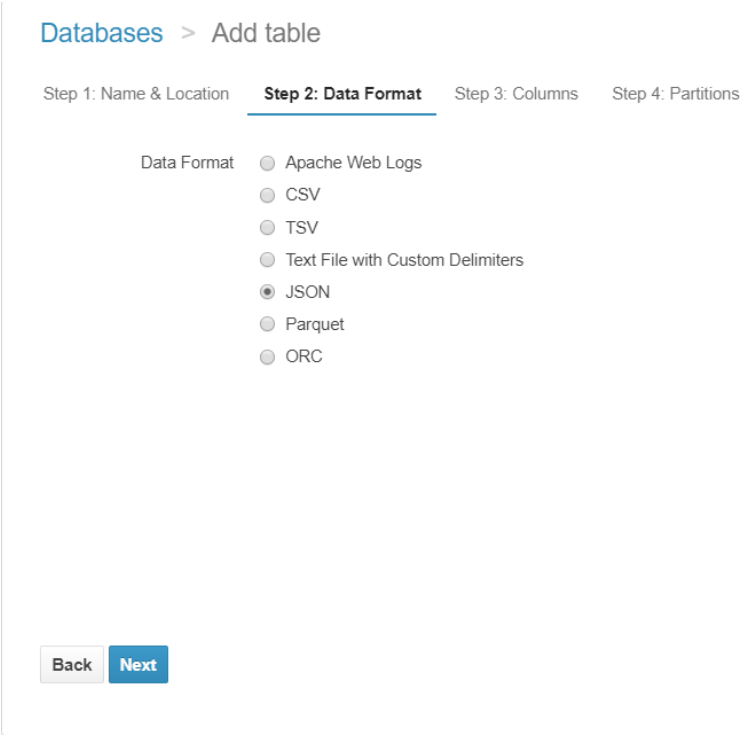
Step1: Open the Athena console in AWS, create a database.

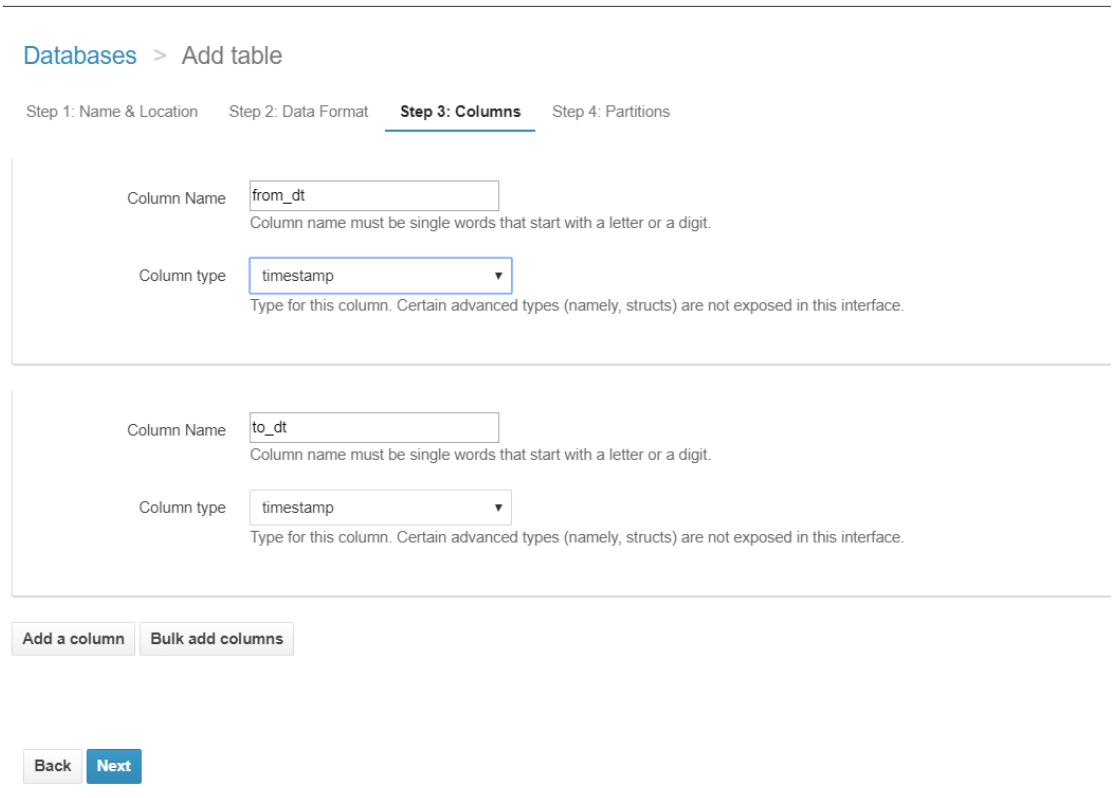
```
CREATE DATABASE mydatabase
```

Step2: Create a Table

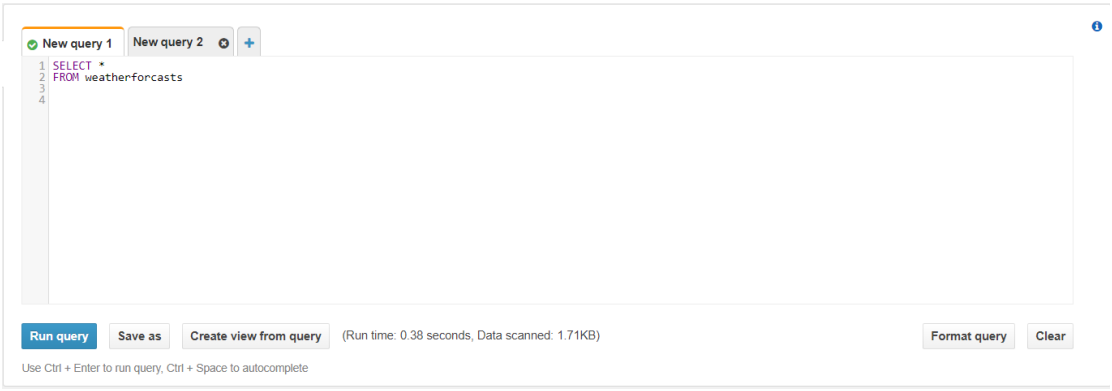
Select the directory with JSON output file (In this case, s3://mogreps-uk-json) from previous section as Dataset.







Step3: Query Data



AWS provides lots of samples on coding and querying in Athena. For more information, see Athena user guide and code samples.

Subscribing to MOGREPS Data Notification in Amazon SNS

To enhance the real-time processing of data, you can setup Amazon SNS to create a notification for every new file added to the MOGREPS S3 bucket and archive buckets for MOGREPS on AWS. Simply subscribing to these notifications using Amazon SQS and AWS Lambda, you can automatically add new real-time and near-real-time MOGREPS data into a queue or trigger event-based processing if the data meets certain criteria such as geographic location.

Amazon Simple Notification Service (SNS) is a flexible, fully managed pub/sub messaging and mobile notifications service for coordinating the delivery of messages to subscribing endpoints and clients. With SNS you can fan-out messages to a large number of subscribers, including distributed systems and services, and mobile devices. It is easy to set up, operate, and reliably send notifications to all your endpoints – at any scale. You can get started using SNS in a matter of minutes using the AWS Management Console, AWS Command Line Interface, or using the AWS SDK with just three simple APIs. SNS eliminates the complexity and overhead associated with managing and operating dedicated messaging software and infrastructure. For details, see Getting Started with Amazon SNS.