Content Page

1.1 About Report

Project Name: Analysis Data Pipeline for Global Fisheries
Module: Data Engineering in the Cloud
By: Javen Lai Le Yu (2202934B)
Class: P03

1.2 Background

I am tasked to create an infrastructure to host fishing data so that Data Analysts can create reports about fishing impacts in the open seas. My solution is a batch Data Pipeline using AWS cloud services that provides analysts with the data they require to perform analytics.

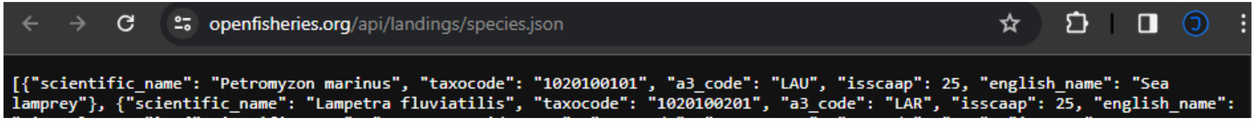
1.3 Problem Statement

Data analysts need to analyze catch performance for species and countries, for the years from 1950 to 2018.

1.4 Data Source

The datasets about fishes will be pulled from OpenFisheries.org's publicly available RESTful API, utilizing the following endpoints:

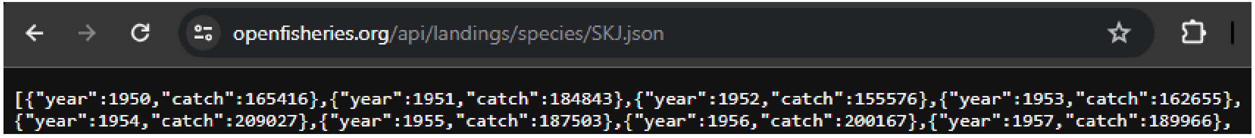
1. **/species.json**



```
[{"scientific_name": "Petromyzon marinus", "taxocode": "1020100101", "a3_code": "LAU", "isscaap": 25, "english_name": "Sea
lamprey"}, {"scientific_name": "Lampetra fluviatilis", "taxocode": "1020100201", "a3_code": "LAR", "isscaap": 25, "english_name":
```

Provides the list of all species of fish that exist and their respective information.

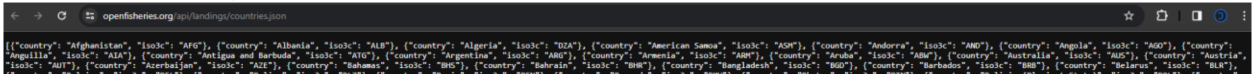
2. **/species/{a3_code}.json**



```
[{"year":1950,"catch":165416},{"year":1951,"catch":184843},{"year":1952,"catch":155576},{"year":1953,"catch":162655},
{"year":1954,"catch":209027},{"year":1955,"catch":187503},{"year":1956,"catch":200167},{"year":1957,"catch":189966},
```

Provides how many tonnes of a fish species were caught each year, accessed using the a3_code of the fish species.

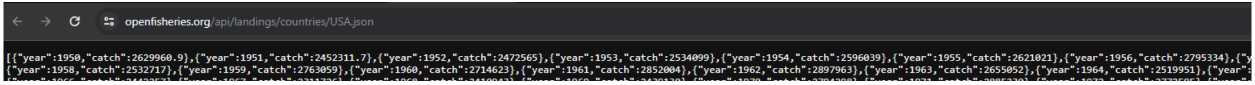
3. **/countries.json**



```
[{"country": "Afghanistan", "iso3c": "AFG"}, {"country": "Albania", "iso3c": "ALB"}, {"country": "Algeria", "iso3c": "DZA"}, {"country": "American Samoa", "iso3c": "ASM"}, {"country": "Andorra", "iso3c": "AND"}, {"country": "Angola", "iso3c": "AGO"}, {"country":
"Anguilla", "iso3c": "AIA"}, {"country": "Antigua and Barbuda", "iso3c": "ATG"}, {"country": "Argentina", "iso3c": "ARG"}, {"country": "Armenia", "iso3c": "ARM"}, {"country": "Aruba", "iso3c": "ABW"}, {"country": "Australia", "iso3c": "AUS"}, {"country": "Austria",
"iso3c": "AUT"}, {"country": "Azerbaijan", "iso3c": "AZE"}, {"country": "Bahamas", "iso3c": "BHS"}, {"country": "Bahrain", "iso3c": "BHR"}, {"country": "Bangladesh", "iso3c": "BGD"}, {"country": "Barbados", "iso3c": "BRB"}, {"country": "Belarus", "iso3c": "BLR"},
```

Provides the list of countries available and their iso3c.

4. **countries/{iso3c}.json**



```
[{"year":1950,"catch":2629960.9},{"year":1951,"catch":2452311.7},{"year":1952,"catch":2472565},{"year":1953,"catch":2534099},{"year":1954,"catch":2596039},{"year":1955,"catch":2621021},{"year":1956,"catch":2795334},{"y
{"year":1958,"catch":2532717},{"year":1959,"catch":2763059},{"year":1960,"catch":2714623},{"year":1961,"catch":2852004},{"year":1962,"catch":2897963},{"year":1963,"catch":2655052},{"year":1964,"catch":2519951},{"year":
```

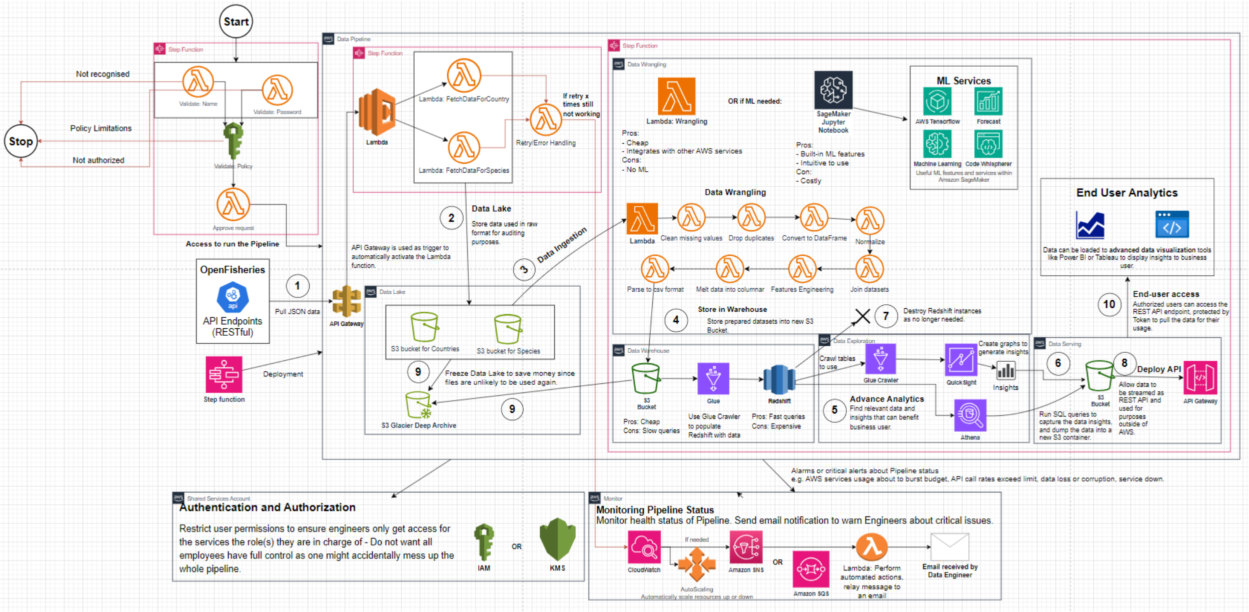
Provides how many tonnes of fish were caught each year for the specified country.

1.5 Objectives of the project

1. Implement a viable prototype of the designed Data Pipeline using AWS services as Proof of Concept (POC).
2. Perform basic Business Intelligence by creating visualizations to answer the problem statement using the prototype to prove the pipeline's usability and judge its feasibility for actual deployment using larger datasets.

2. Data Pipeline Architecture

The Data Pipeline architecture has been designed under the Group Project section.


Start
Stop
Not recognised
Policy Limitations
Not authorized
Step Function
Validate: Name
Validate: Password
Validate: Policy
Approve request
Access to run the Pipeline
OpenFisheries
api
API Endpoints (RESTful)
1
Pull JSON data
API Gateway
Step function
Deployment
Data Pipeline
Step Function
Lambda
Lambda: FetchDataForCountry
Lambda: FetchDataForSpecies
Retry/Error Handling
If retry x times still not working
API Gateway is used as trigger to automatically activate the Lambda function.
2
Data Lake
Store data used in raw format for auditing purposes.
3
Data Ingestion
Data Lake
S3 bucket for Countries
S3 bucket for Species
9
Freeze Data Lake to save money since files are unlikely to be used again.
S3 Glacier Deep Archive
9
Step Function
Data Wrangling
Lambda: Wrangling
Pros:
- Cheap
- Integrates with other AWS services
Cons:
- No ML
OR if ML needed:
SageMaker Jupyter Notebook
Pros:
- Built-in ML features
- Intuitive to use
Con:
- Costly
ML Services
AWS Tensorflow
Forecast
Machine Learning
Code Whisperer
Useful ML features and services within Amazon SageMaker
Data Wrangling
Lambda
Clean missing values
Drop duplicates
Convert to DataFrame
Normalize
Join datasets
Features Engineering
Melt data into columnar
Parse to csv format
4
Store in Warehouse
Store prepared datasets into new S3 Bucket.
7
Destroy Redshift instances as no longer needed.
Data Warehouse
S3 Bucket
Pros: Cheap
Cons: Slow queries
Glue
Use Glue Crawler to populate Redshift with data
Redshift
Pros: Fast queries
Cons: Expensive
Data Exploration
Crawl tables to use
Glue Crawler
QuickSight
Create graphs to generate insights
Insights
5
Advance Analytics
Find relevant data and insights that can benefit business user.
Athena
Data Serving
6
S3 Bucket
Run SQL queries to capture the data insights, and dump the data into a new S3 container.
8
Deploy API
Allow data to be streamed as REST API and used for purposes outside of AWS.
API Gateway
End User Analytics
Data can be loaded to advanced data visualization tools like Power BI or Tableau to display insights to business user.
10
End-user access
Authorized users can access the REST API endpoint, protected by Token to pull the data for their usage.
Alarms or critical alerts about Pipeline status
e.g. AWS services usage about to burst budget, API call rates exceed limit, data loss or corruption, service down.
Shared Services Account
Authentication and Authorization
Restrict user permissions to ensure engineers only get access for the services the role(s) they are in charge of - Do not want all employees have full control as one might accidentally mess up the whole pipeline.
IAM
OR
KMS
Monitor
Monitoring Pipeline Status
Monitor health status of Pipeline. Send email notification to warn Engineers about critical issues.
CloudWatch
If needed
AutoScaling
Automatically scale resources up or down
Amazon SNS
OR
Amazon SQS
Lambda: Perform automated actions, relay message to an email
Email received by Data Engineer

Limitations of Implementation:

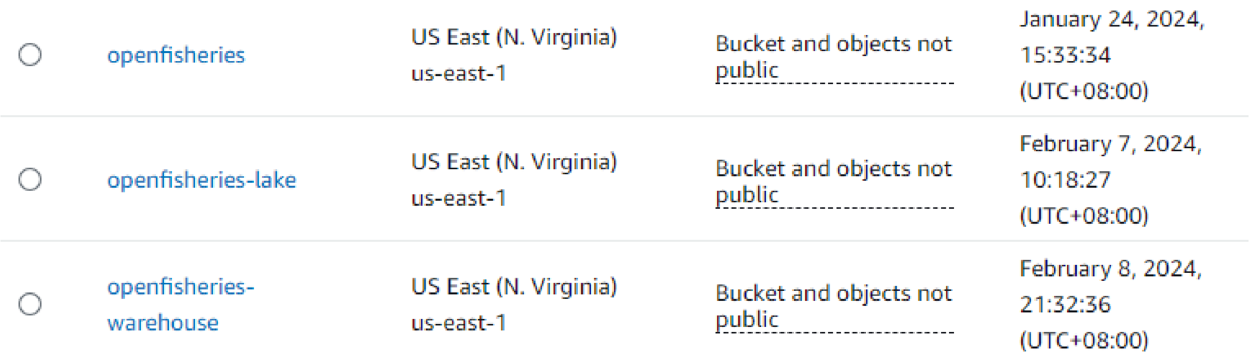
1. Restricted IAM access as I have to work with a controlled AWS environment where I cannot create new roles. Hence, unable to implement authentication and authorization measures.
2. Budget of $100.

Modifications to Pipeline design:

- Unable to create users or roles to emulate several data engineers due to IAM restrictions.
- I will have a central S3 Bucket as a data lake to store species and countries, separated by folders.
- Use Glue Catalog as the database instead of Redshift due to tight budget.
- Skip Steps 8 and 10. Do not need to release data as API when I'm the only analyst using the data.

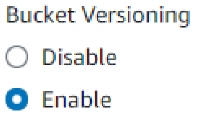
3. Creating Pipeline

3.1 Preparing S3 Buckets



○	openfisheries	US East (N. Virginia) us-east-1	Bucket and objects not public	January 24, 2024, 15:33:34 (UTC+08:00)
○	openfisheries-lake	US East (N. Virginia) us-east-1	Bucket and objects not public	February 7, 2024, 10:18:27 (UTC+08:00)
○	openfisheries-warehouse	US East (N. Virginia) us-east-1	Bucket and objects not public	February 8, 2024, 21:32:36 (UTC+08:00)

Created all 3 S3 Buckets that will be used. Default configurations were used. However, for openfisheries, I enabled versioning since this bucket stores the main data files that will be used for analysis, hence keeping historical versions would be good to track deletes or changes.



Bucket Versioning

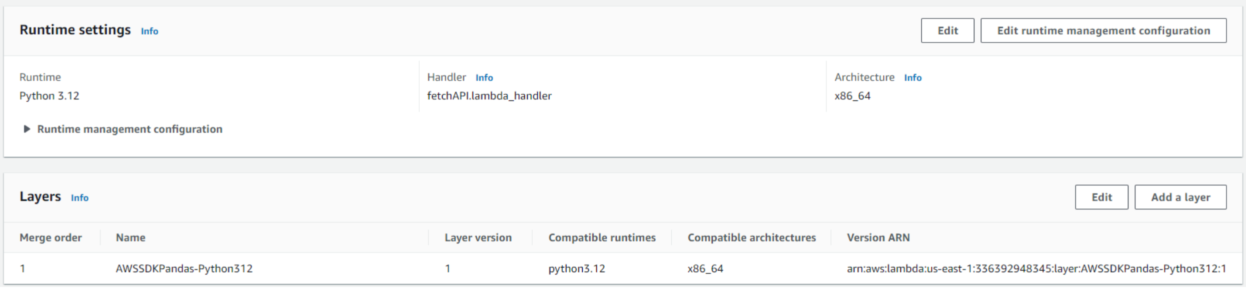
○ Disable

● Enable

3.2 Data Lake

Goal: Pull 30 different datasets from OpenFisheries REST API and dump them into S3 Data Lake.

3.2.1 Configurations of Lambda function, fetchAPI.py:



Runtime settings Info

Edit
Edit runtime management configuration

Runtime	Handler Info	Architecture Info
Python 3.12	fetchAPI.lambda_handler	x86_64

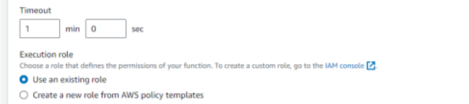
▸ Runtime management configuration

Layers Info

Merge order	Name	Layer version	Compatible runtimes	Compatible architectures	Version ARN
1	AWSSDKPandas-Python312	1	python3.12	x86_64	arn:aws:lambda:us-east-1:336392948345:layer:AWSSDKPandas-Python312:1

3.2.2 Modified Timeout to 1 min to ensure sufficient timing is given for code to run before timing out.

3.2.3 Execute Function under LabRole permissions.



Timeout

1 min 0 sec

Execution role

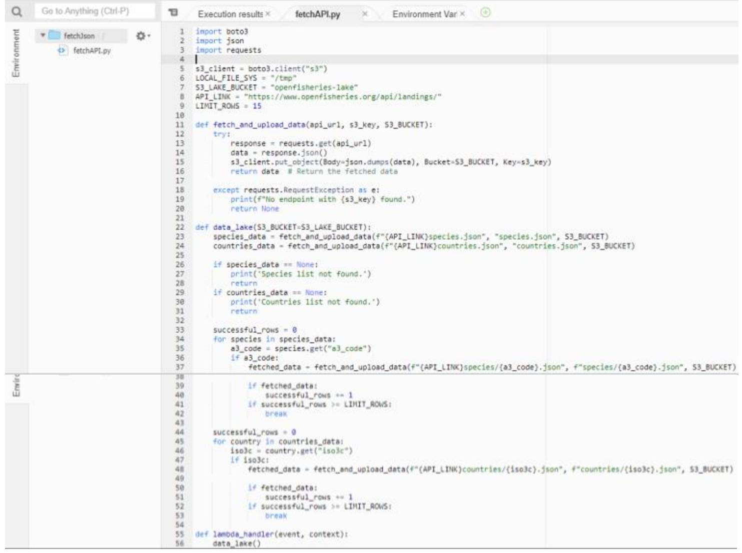
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

● Use an existing role

○ Create a new role from AWS policy templates

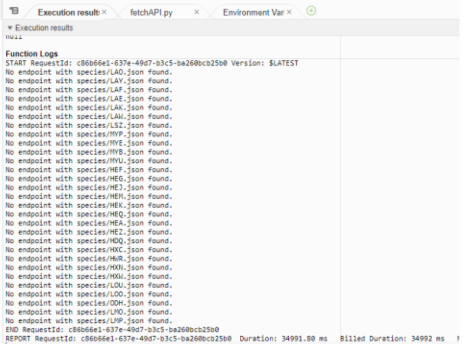
3.2.4 **Data Ingestion:** Lambda code to pull data from API Endpoints

No retry mechanism because unreachable endpoints do not exist. For error-tolerance, the code moves on to the next country/species in the list. It runs until 1 min timeout OR if 30 (15-15) datasets are loaded.



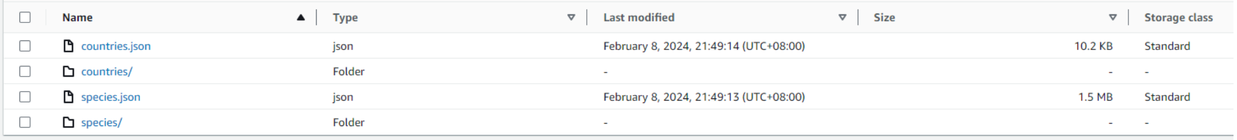
```python
import boto3
import json
import requests

s3_client = boto3.client("s3")
LOCAL_FILE_SYS = "/tmp"
S3_LAKE_BUCKET = "openfisheries-lake"
API_LINK = "https://www.openfisheries.org/api/landings/"
LIMIT_ROWS = 15

def fetch_and_upload_data(api_url, s3_key, S3_BUCKET):
    try:
        response = requests.get(api_url)
        data = response.json()
        s3_client.put_object(Body=json.dumps(data), Bucket=S3_BUCKET, Key=s3_key)
        return data  # Return the fetched data

    except requests.RequestException as e:
        print(f"No endpoint with {s3_key} found.")
        return None

def data_lake(S3_BUCKET=S3_LAKE_BUCKET):
    species_data = fetch_and_upload_data(f"{API_LINK}species.json", "species.json", S3_BUCKET)
    countries_data = fetch_and_upload_data(f"{API_LINK}countries.json", "countries.json", S3_BUCKET)

    if species_data == None:
        print('Species list not found.')
        return
    if countries_data == None:
        print('Countries list not found.')
        return

    successful_rows = 0
    for species in species_data:
        a3_code = species.get("a3_code")
        if a3_code:
            fetched_data = fetch_and_upload_data(f"{API_LINK}species/{a3_code}.json", f"species/{a3_code}.json", S3_BUCKET)

            if fetched_data:
                successful_rows += 1
            if successful_rows >= LIMIT_ROWS:
                break

    successful_rows = 0
    for country in countries_data:
        iso3c = country.get("iso3c")
        if iso3c:
            fetched_data = fetch_and_upload_data(f"{API_LINK}countries/{iso3c}.json", f"countries/{iso3c}.json", S3_BUCKET)

            if fetched_data:
                successful_rows += 1
            if successful_rows >= LIMIT_ROWS:
                break

def lambda_handler(event, context):
    data_lake()
```

After running Python code that **pulls the first 15 species and 15 countries** in the lists available, an interesting issue was found: not all species specified on the species list have an accessible endpoint.



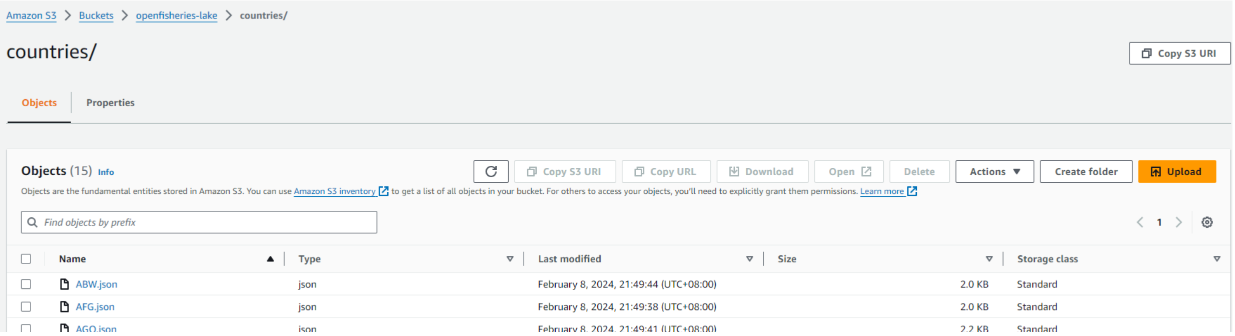
```
null

Function Logs
START RequestId: c86b66e1-637e-49d7-b3c5-ba260bcb25b0 Version: $LATEST
No endpoint with species/LAO.json found.
No endpoint with species/LAY.json found.
No endpoint with species/LAF.json found.
No endpoint with species/LAE.json found.
No endpoint with species/LAK.json found.
No endpoint with species/LAW.json found.
No endpoint with species/LSZ.json found.
No endpoint with species/MYP.json found.
No endpoint with species/MYE.json found.
No endpoint with species/MYB.json found.
No endpoint with species/MYU.json found.
No endpoint with species/HEF.json found.
No endpoint with species/HEG.json found.
No endpoint with species/HEJ.json found.
No endpoint with species/HEM.json found.
No endpoint with species/HEK.json found.
No endpoint with species/HEQ.json found.
No endpoint with species/HEA.json found.
No endpoint with species/HEZ.json found.
No endpoint with species/HDQ.json found.
No endpoint with species/HXC.json found.
No endpoint with species/HWR.json found.
No endpoint with species/HXN.json found.
No endpoint with species/HXW.json found.
No endpoint with species/LOU.json found.
No endpoint with species/LOO.json found.
No endpoint with species/ODH.json found.
No endpoint with species/LMO.json found.
No endpoint with species/LMP.json found.
END RequestId: c86b66e1-637e-49d7-b3c5-ba260bcb25b0
REPORT RequestId: c86b66e1-637e-49d7-b3c5-ba260bcb25b0  Duration: 34991.80 ms   Billed Duration: 34992 ms
```

3.2.5 S3 Bucket results:



□	Name ▲	Type ▽	Last modified ▽	Size ▽	Storage class
□	countries.json	json	February 8, 2024, 21:49:14 (UTC+08:00)	10.2 KB	Standard
□	countries/	Folder	-	-	-
□	species.json	json	February 8, 2024, 21:49:13 (UTC+08:00)	1.5 MB	Standard
□	species/	Folder	-	-	-

3.2.6 Proof that data were successfully dumped into Data Lake



countries/

Objects (15) Info

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

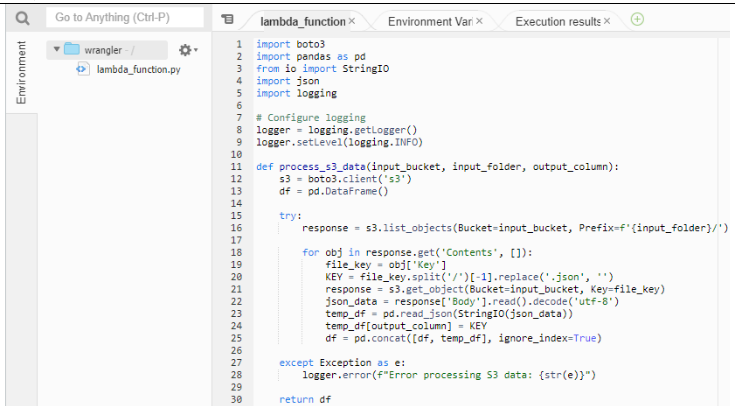
□	Name ▲	Type ▽	Last modified ▽	Size ▽	Storage class ▽
□	ABW.json	json	February 8, 2024, 21:49:44 (UTC+08:00)	2.0 KB	Standard
□	AFG.json	json	February 8, 2024, 21:49:38 (UTC+08:00)	2.0 KB	Standard
□	AGO.json	json	February 8, 2024, 21:49:41 (UTC+08:00)	2.2 KB	Standard

3.3 Data Wrangling

Extract files from Data Lake into Lambda Function that performs cleaning and transformation to prepare the data the be stored in Data Warehouse schema and usable for analytics. The same configurations for the Lambda function are used (Runtime + Layer to enable pandas libraries to be used).

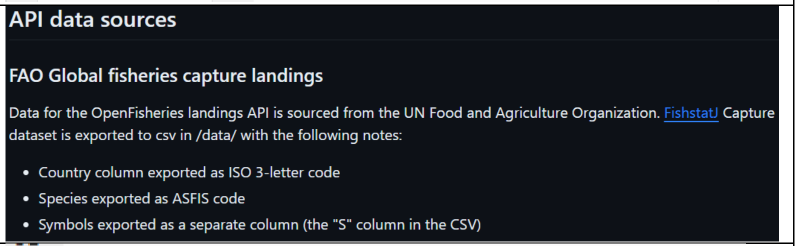
Added a logger to log errors that occurred in data cleaning.

This function appends a new column to store the file's name as a value (Species/Country name). Then, it unpivots the data from separate files into a single dataset by stacking them in rows and transforming the data into columnar format (to be used for analysis).



```python
import boto3
import pandas as pd
from io import StringIO
import json
import logging

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def process_s3_data(input_bucket, input_folder, output_column):
    s3 = boto3.client('s3')
    df = pd.DataFrame()

    try:
        response = s3.list_objects(Bucket=input_bucket, Prefix=f'{input_folder}/')

        for obj in response.get('Contents', []):
            file_key = obj['Key']
            KEY = file_key.split('/')[-1].replace('.json', '')
            response = s3.get_object(Bucket=input_bucket, Key=file_key)
            json_data = response['Body'].read().decode('utf-8')
            temp_df = pd.read_json(StringIO(json_data))
            temp_df[output_column] = KEY
            df = pd.concat([df, temp_df], ignore_index=True)

    except Exception as e:
        logger.error(f"Error processing S3 data: {str(e)}")

    return df
```

Based on OpenFisheries API GitHub documentation, the data in the API is already prepared as it was originally taken from another data source, suggesting that the data quality should be relatively good and clean.



API data sources

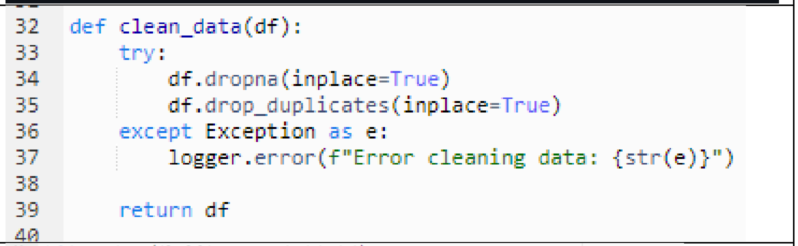
FAO Global fisheries capture landings

Data for the OpenFisheries landings API is sourced from the UN Food and Agriculture Organization. FishstatJ Capture dataset is exported to csv in /data/ with the following notes:

- Country column exported as ISO 3-letter code
- Species exported as ASFIS code
- Symbols exported as a separate column (the "S" column in the CSV)

Basic data cleaning practices still applied despite trust in the API just as precautionary measures.

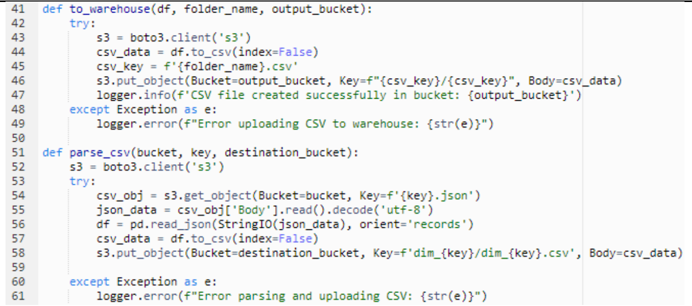
This function performs basic data cleaning like dropping missing values and duplicated rows.



```python
def clean_data(df):
    try:
        df.dropna(inplace=True)
        df.drop_duplicates(inplace=True)
    except Exception as e:
        logger.error(f"Error cleaning data: {str(e)}")

    return df
```

1st Function stores each file as CSV into a file dir using its name before storing the file because each file dir can only contain 1 file for Glue Crawler to Athena to work.

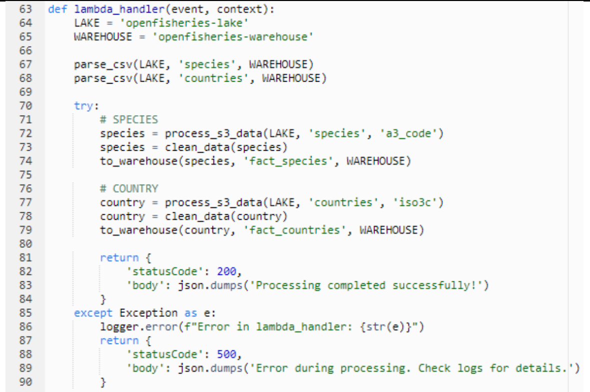
2nd Function to parse countries and species lists into CSV before storing them in S3.



```python
def to_warehouse(df, folder_name, output_bucket):
    try:
        s3 = boto3.client('s3')
        csv_data = df.to_csv(index=False)
        csv_key = f'{folder_name}.csv'
        s3.put_object(Bucket=output_bucket, Key=f"{csv_key}/{csv_key}", Body=csv_data)
        logger.info(f'CSV file created successfully in bucket: {output_bucket}')
    except Exception as e:
        logger.error(f"Error uploading CSV to warehouse: {str(e)}")

def parse_csv(bucket, key, destination_bucket):
    s3 = boto3.client('s3')
    try:
        csv_obj = s3.get_object(Bucket=bucket, Key=f'{key}.json')
        json_data = csv_obj['Body'].read().decode('utf-8')
        df = pd.read_json(StringIO(json_data), orient='records')
        csv_data = df.to_csv(index=False)
        s3.put_object(Bucket=destination_bucket, Key=f'dim_{key}/dim_{key}.csv', Body=csv_data)

    except Exception as e:
        logger.error(f"Error parsing and uploading CSV: {str(e)}")
```

Handler is the function that gets run when this Lambda function is run. Hence, this acts as a main() that activates the data wrangling process.

The countries and species list doesn't need cleaning as it's used as a look-up table (dictionary to find endpoint) and not actual data rows.



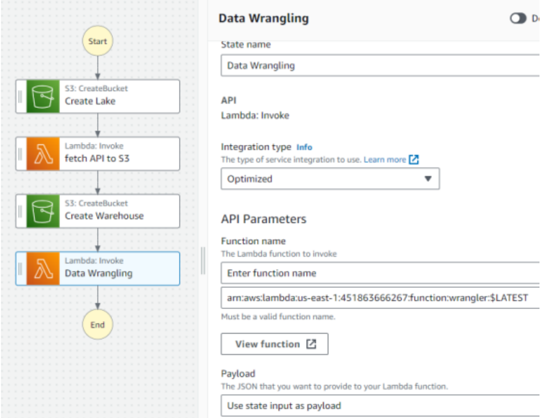
```python
def lambda_handler(event, context):
    LAKE = 'openfisheries-lake'
    WAREHOUSE = 'openfisheries-warehouse'

    parse_csv(LAKE, 'species', WAREHOUSE)
    parse_csv(LAKE, 'countries', WAREHOUSE)

    try:
        # SPECIES
        species = process_s3_data(LAKE, 'species', 'a3_code')
        species = clean_data(species)
        to_warehouse(species, 'fact_species', WAREHOUSE)

        # COUNTRY
        country = process_s3_data(LAKE, 'countries', 'iso3c')
        country = clean_data(country)
        to_warehouse(country, 'fact_countries', WAREHOUSE)

        return {
            'statusCode': 200,
            'body': json.dumps('Processing completed successfully!')
        }
    except Exception as e:
        logger.error(f"Error in lambda_handler: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps('Error during processing. Check logs for details.')
        }
```

3.4 Automating using Step Functions

Goal: Create a function that runs all pre-processing work with 1 click to ensure easy Pipeline reusability.

Lake = openfisheries-lake,
Fetch API = fetchAPI,
Warehouse = openfisheries-warehouse,
Data Wrangling = wrangler

This workflow was manually created using drag and drop.



Data Wrangling

State name: Data Wrangling

API: Lambda: Invoke

Integration type: Optimized
The type of service integration to use.

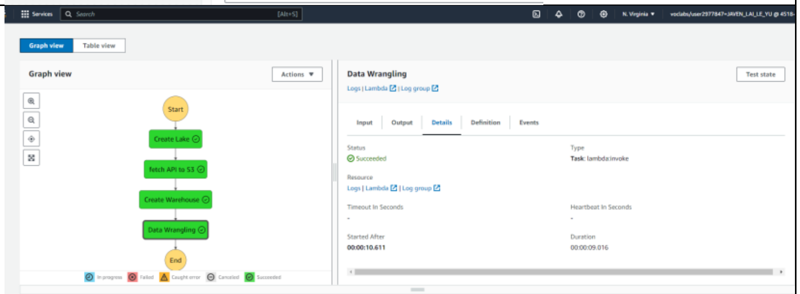
API Parameters

Function name
The Lambda function to invoke
Enter function name
arn:aws:lambda:us-east-1:451863666267:function:wrangler:$LATEST
Must be a valid function name.

Payload
The JSON that you want to provide to your Lambda function.
Use state input as payload

This workflow was able to successfully run without encountering any errors.

This is validated as I cleared the S3 Buckets, ran this Step function, and the S3 buckets were populated with data.



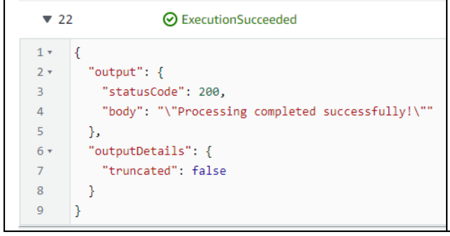
Graph view

Data Wrangling
Logs | Lambda | Log group

Status: Succeeded
Type: Task: lambda:invoke
Resource: Logs | Lambda | Log group
Timeout In Seconds: -
Heartbeat In Seconds: -
Started After: 00:00:10.611
Duration: 00:00:09.016

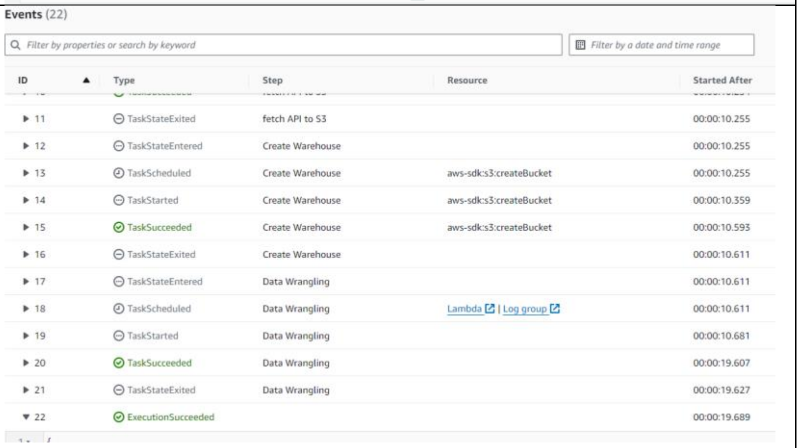
This is the list of events triggered by the Step function.

As seen, all the steps were successfully run and the functions were executed accordingly.



22 ExecutionSucceeded

```
{
  "output": {
    "statusCode": 200,
    "body": "\"Processing completed successfully!\""
  },
  "outputDetails": {
    "truncated": false
  }
}
```


Filter by properties or search by keyword
Filter by a date and time range

Events (22)

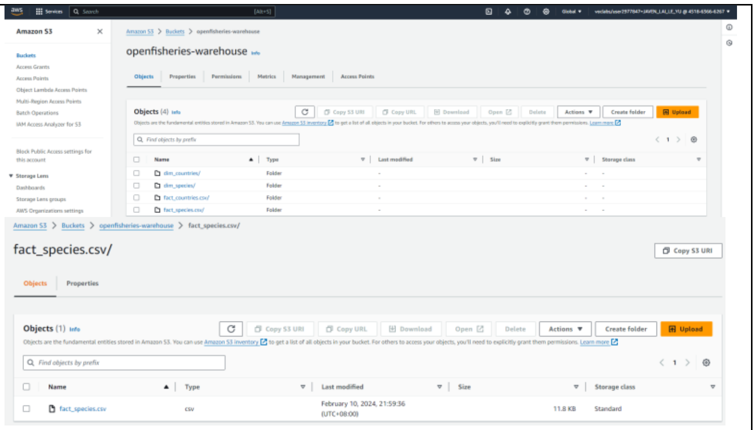
ID	Type	Step	Resource	Started After
11	TaskStateExited	fetch API to S3		00:00:10.255
12	TaskStateEntered	Create Warehouse		00:00:10.255
13	TaskScheduled	Create Warehouse	aws-sdk:s3:createBucket	00:00:10.255
14	TaskStarted	Create Warehouse	aws-sdk:s3:createBucket	00:00:10.359
15	TaskSucceeded	Create Warehouse	aws-sdk:s3:createBucket	00:00:10.593
16	TaskStateExited	Create Warehouse		00:00:10.611
17	TaskStateEntered	Data Wrangling		00:00:10.611
18	TaskScheduled	Data Wrangling	Lambda | Log group	00:00:10.611
19	TaskStarted	Data Wrangling		00:00:10.681
20	TaskSucceeded	Data Wrangling		00:00:19.607
21	TaskStateExited	Data Wrangling		00:00:19.627
22	ExecutionSucceeded			00:00:19.689

3.5 Data Warehouse

All the prepared datasets for the data warehouse are stored in a S3 Bucket.

The intention of doing so is to ensure dataset files are easily and conveniently accessible when needed, as the relational database that stores these data can be destroyed to cut costs.

Due to a tight budget, Glue Catalog is used as a database to store the data to be queryable using Athena.



Amazon S3 > Buckets > openfisheries-warehouse

openfisheries-warehouse

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.

Name	Type	Last modified	Size	Storage class
dim_countries/	Folder	-	-	-
dim_species/	Folder	-	-	-
fact_countries.csv/	Folder	-	-	-
fact_species.csv/	Folder	-	-	-

Amazon S3 > Buckets > openfisheries-warehouse > fact_species.csv/

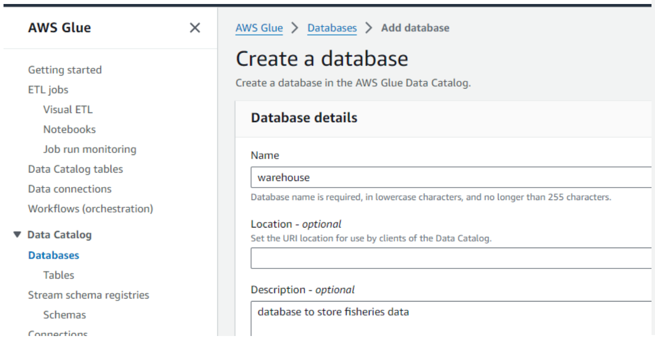
fact_species.csv/

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.

Name	Type	Last modified	Size	Storage class
fact_species.csv	csv	February 10, 2024, 21:59:36 (UTC+08:00)	11.8 KB	Standard

3.5.1 Creating Database at Glue Data Catalog



Create a database

Create a database in the AWS Glue Data Catalog.

Database details

Name

warehouse

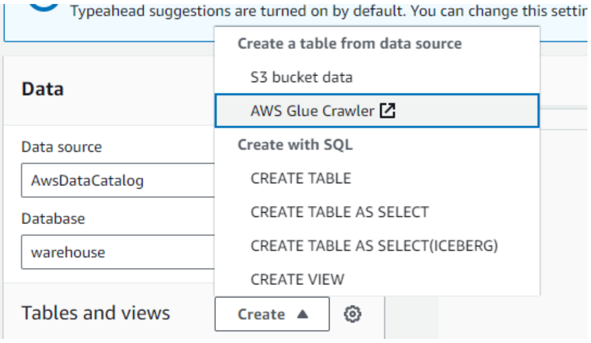
Database name is required, in lowercase characters, and no longer than 255 characters.

Location - *optional*

Set the URI location for use by clients of the Data Catalog.

Description - *optional*

database to store fisheries data


Typeahead suggestions are turned on by default. You can change this settir
Create a table from data source
S3 bucket data
AWS Glue Crawler
Create with SQL
CREATE TABLE
CREATE TABLE AS SELECT
CREATE TABLE AS SELECT(ICEBERG)
CREATE VIEW
Create ▲

Data

Data source

AwsDataCatalog

Database

warehouse

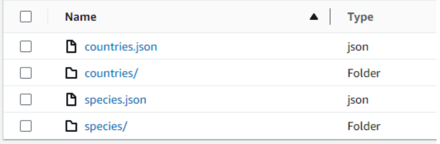
Tables and views

3.5.2 Glue Crawler

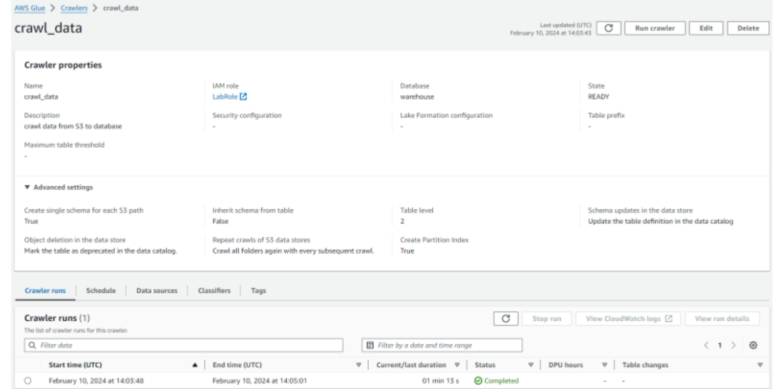
To effortlessly load the datasets from the S3 warehouse into the database, a Crawler is used to infer and automate the creation of tables based on the dataset files and load the data into the tables.

Some additional configurations:
Firstly, I declared that each S3 file directory path has its own schema.

Next, I set the Table level to 2 to ensure that the files nested within the respective sub-folders get crawled:



□	Name ▲	Type
□	countries.json	json
□	countries/	Folder
□	species.json	json
□	species/	Folder



crawl_data

Last updated (UTC)
February 10, 2024 at 14:03:43

Crawler properties

Name	IAM role	Database	State
crawl_data	LabRole	warehouse	READY

Description	Security configuration	Lake Formation configuration	Table prefix
crawl data from S3 to database	-	-	-

Maximum table threshold

-

▼ Advanced settings

Create single schema for each S3 path	Inherit schema from table	Table level	Schema updates in the data store
True	False	2	Update the table definition in the data catalog

Object deletion in the data store	Repeat crawls of S3 data stores	Create Partition Index
Mark the table as deprecated in the data catalog.	Crawl all folders again with every subsequent crawl.	True

Crawler runs (1)

The list of crawler runs for this crawler.

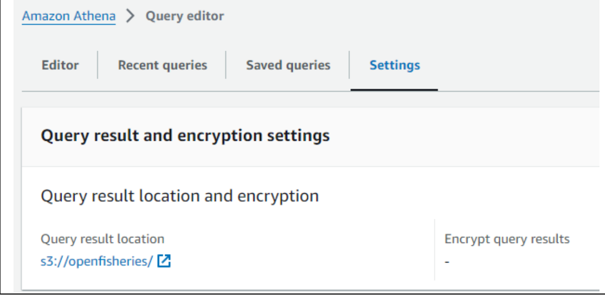
	Start time (UTC)	End time (UTC)	Current/last duration	Status	DPU hours	Table changes
○	February 10, 2024 at 14:03:48	February 10, 2024 at 14:05:01	01 min 13 s	✓ Completed	-	-

3.6 Exploratory Data Analysis (EDA) + Preparing Data for Analytics

3.6.0 Configurations for Athena

Configured Athena to dump results of all queries run into the openfisheries bucket meant for storing the insights from SQL queries.

When an SQL query is run from Editor console, the results will be logged and sent to 'openfisheries' S3 Bucket with versioning.



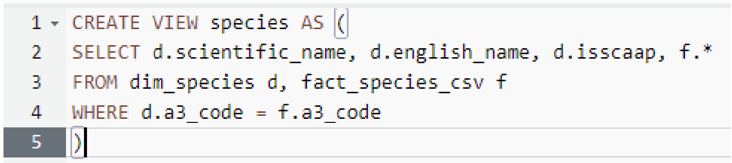
Query result and encryption settings

Query result location and encryption

Query result location	Encrypt query results
s3://openfisheries/	-

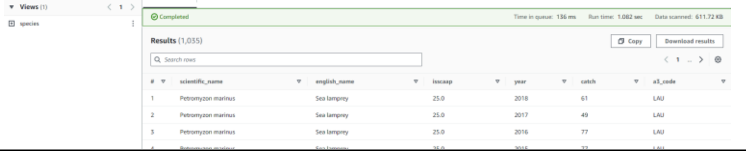
3.6.1 Species:

To prepare the data to be used for EDA, I created a View which essentially acts as a table that only has the relevant columns I need to use.



```
CREATE VIEW species AS (
SELECT d.scientific_name, d.english_name, d.isscaap, f.*
FROM dim_species d, fact_species_csv f
WHERE d.a3_code = f.a3_code
)
```

SELECT * FROM species to check the data inside the View table. There are 1035 observations from 2018 to 1950 for the 15 selected species.



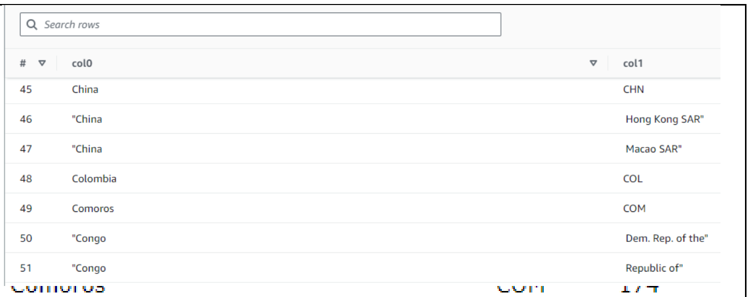
✓ Completed Time in queue: 136 ms Run time: 1.082 sec Data scanned: 611.72 KB

Results (1,035)

#	scientific_name	english_name	isscaap	year	catch	a3_code
1	Petromyzon marinus	Sea lamprey	25.0	2018	61	LAU
2	Petromyzon marinus	Sea lamprey	25.0	2017	49	LAU
3	Petromyzon marinus	Sea lamprey	25.0	2016	77	LAU

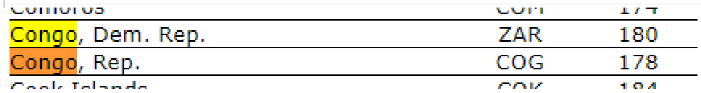
Found that the iso3c for certain countries was wrong. Then, I noticed that the country name was duplicated for these records. Another issue with the countries list dataset was that the column headers were not configured properly.

However, these issues should not be a concern as they do not affect the main data and can be easily fixed with SQL.



Search rows

#	col0	col1
45	China	CHN
46	"China	Hong Kong SAR"
47	"China	Macao SAR"
48	Colombia	COL
49	Comoros	COM
50	"Congo	Dem. Rep. of the"
51	"Congo	Republic of"



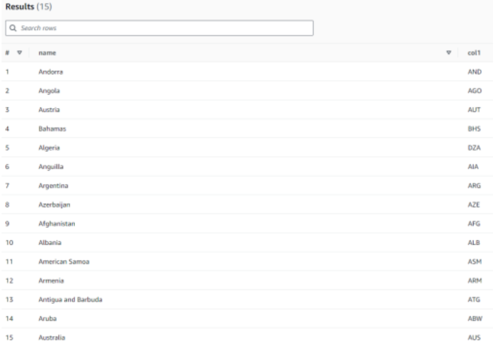
Comoros	COM	174
Congo, Dem. Rep.	ZAR	180
Congo, Rep.	COG	178
Cook Islands	COK	184

https://wits.worldbank.org/WITS/wits/WITSHELP/Content/Codes/Country_Codes.htm

3.6.2 Countries:

The 15 files I sampled can NEVER contain a country with faulty iso3c as an endpoint with wrong iso3c does not exist.

A possible fix to this issue would be to source online for a flawless country and iso3c list with no data issues instead of using the OpenFisheries countries list if all countries' fishery data needs to be extracted and used for analysis.

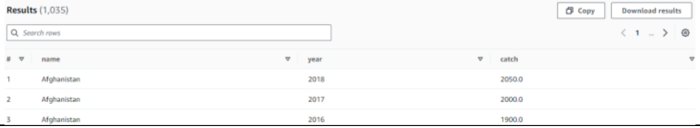


Results (15)

Search rows

#	name	col1
1	Andorra	AND
2	Angola	AGO
3	Austria	AUT
4	Bahamas	BHS
5	Algeria	DZA
6	Anguilla	AIA
7	Argentina	ARG
8	Azerbaijan	AZE
9	Afghanistan	AFG
10	Albania	ALB
11	American Samoa	ASM
12	Armenia	ARM
13	Antigua and Barbuda	ATG
14	Aruba	ABW
15	Australia	AUS

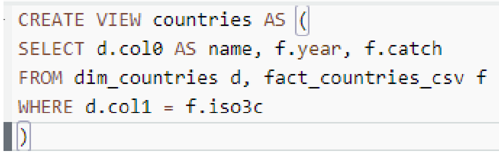
Similar to species, I created a View for countries that only contains the relevant columns needed for analysis. Additionally, column headers are fixed to be an accurate representation of what the values that the column holds.



Results (1,035)

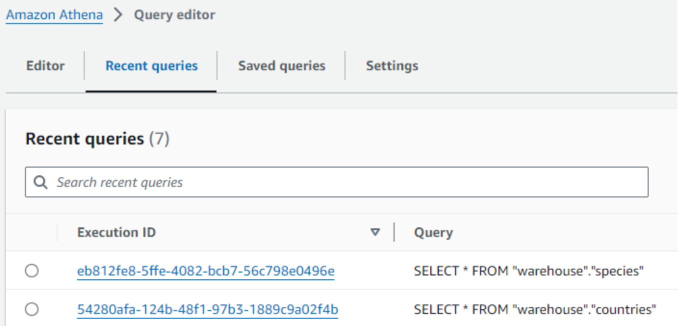
#	name	year	catch
1	Afghanistan	2018	2050.0
2	Afghanistan	2017	2000.0
3	Afghanistan	2016	1900.0

Code used to create View:



```
CREATE VIEW countries AS (
SELECT d.col0 AS name, f.year, f.catch
FROM dim_countries d, fact_countries_csv f
WHERE d.col1 = f.iso3c
)
```

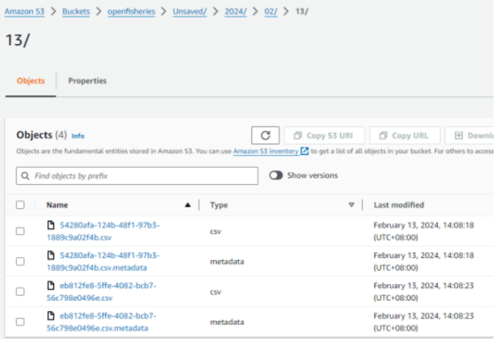
These 2 main datasets are the final product of this Pipeline, and are stored in the openfisheries S3 Bucket for indefinite usage (until Openfisheries updates again).



Recent queries (7)

	Execution ID	Query
○	eb812fe8-5ffe-4082-bcb7-56c798e0496e	SELECT * FROM "warehouse"."species"
○	54280afa-124b-48f1-97b3-1889c9a02f4b	SELECT * FROM "warehouse"."countries"

How openfisheries S3 bucket looks:



Amazon S3 > Buckets > openfisheries > Unsaved/ > 2024/ > 02/ > 13/
13/
Objects
Properties
Objects (4) Info
Copy S3 URI
Copy URL
Downl

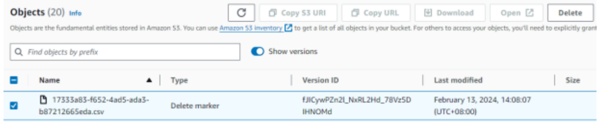
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access

	Name	Type	Last modified
□	54280afa-124b-48f1-97b3-1889c9a02f4b.csv	csv	February 13, 2024, 14:08:18 (UTC+08:00)
□	54280afa-124b-48f1-97b3-1889c9a02f4b.csv.metadata	metadata	February 13, 2024, 14:08:18 (UTC+08:00)
□	eb812fe8-5ffe-4082-bcb7-56c798e0496e.csv	csv	February 13, 2024, 14:08:23 (UTC+08:00)
□	eb812fe8-5ffe-4082-bcb7-56c798e0496e.csv.metadata	metadata	February 13, 2024, 14:08:23 (UTC+08:00)

The CSV files are the main products.

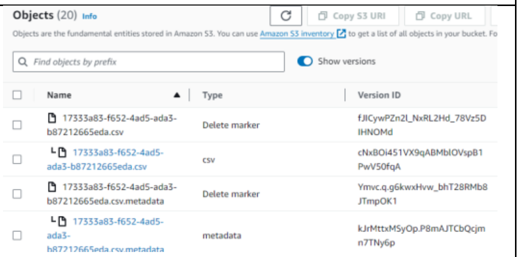
To show that the version control for 'openfisheries' S3 bucket works, the photo shows that all the previous files that were deleted are still recorded and traceable in the bucket when 'Show Versions' is checked.

To delete the trace of the file permanently:



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

	Name	Type	Version ID	Last modified	Size
☑	17333a83-f652-4ad5-ada3-b87212665eda.csv	Delete marker	fJICywPZn2I_NxRL2Hd_78Vz5DIHNOMd	February 13, 2024, 14:08:07 (UTC+08:00)	



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. Fo

	Name	Type	Version ID
□	17333a83-f652-4ad5-ada3-b87212665eda.csv	Delete marker	fJICywPZn2I_NxRL2Hd_78Vz5DIHNOMd
□	17333a83-f652-4ad5-ada3-b87212665eda.csv	csv	cNxBOi451VX9qABMblOVspB1PwV50fqA
□	17333a83-f652-4ad5-ada3-b87212665eda.csv.metadata	Delete marker	Ymvc.q.g6kwxHvw_bhT28RMb8JTmpOK1
□	17333a83-f652-4ad5-ada3-b87212665eda.csv.metadata	metadata	kJrMttxMSyOp.P8mAJTCbQcjmn7TNy6p

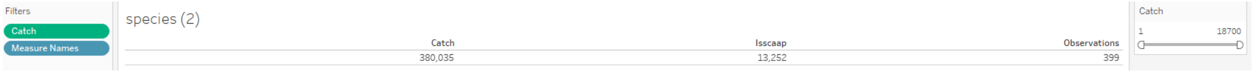
Deleted files were unnecessary queries

3.7 Advanced Analytics using Tableau

The files were downloaded from the insights S3 Bucket and loaded onto Tableau to conduct professional Data Analytics to report any insights or findings regarding fishes… and to answer the problem statement.

3.7.1 Some steps taken to build visualization

1. Filter Catch=0 *for the KPI graph* as 0 is almost the same as having no fish caught that year since the sum is less than 1 tonne. These rows are retained in data because a country that has nearly 0 catches for a year is still considered an insight.



Catch	Isscaap	Observations
380,035	13,252	399

2. Created a Dashboard for both Species and Countries to conduct analysis. Designed the dashboard to be interactive by allowing the data to be filtered using each graph in the dashboard.

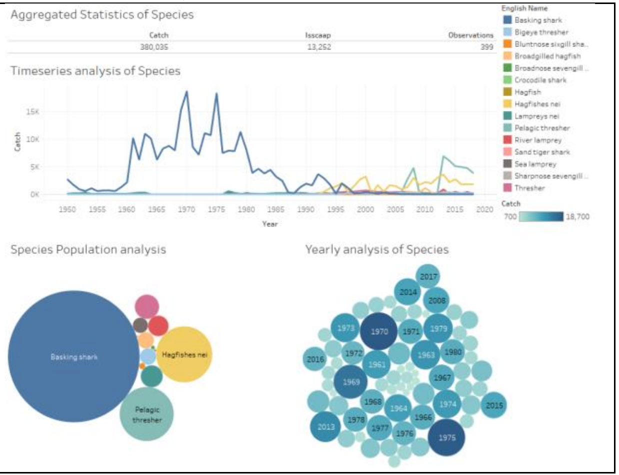
3.7.2 Species Dashboard to analyze Species performance across the year.

1. The basking shark was an extremely popular fish during 1960-1980, but the catch quantity drastically fell after. This suggests that this species is either going extinct as it was overfished or that people lost interest in catching this species.

2. The most popular fish in the modern 21st century is the Bigeye thresher, which especially gained popularity from 2012 onwards.

3. The Hagfish Nei was only caught in 1993, suggesting that its existence was found in 1993.

4. Most of the fish were caught during 1975, 1970, and 1969.


Aggregated Statistics of Species

Catch	Isscaap	Observations
380,035	13,252	399

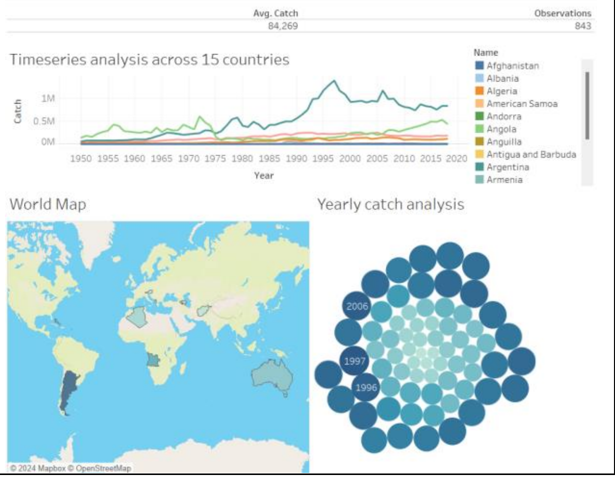
3.7.3 Countries Dashboard to analyze countries' catch performance across the year.

1. Argentina is the heaviest fishing country out of the 15 countries ever since 1975 onwards.

2. Angola's fishing intensity dropped sharply after 1975, but they picked up their pace again from 2010 onwards.

3. Countries surrounding the open ocean waters tend to have higher catches in tonnes.

4. There seem to be no peak years for catching fish for these 15 countries since there are ample years with almost the same size and color. However, the top 3 years are 2006, 1997, and 1996.



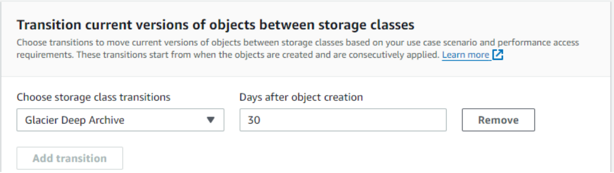
Avg. Catch	Observations
84,269	843

Link to video explanation of how Dashboard was interpreted:
https://youtu.be/o_mECUFbNJg

3.8 Archiving S3 Buckets (End of Pipeline procedure)

Since Data Lake and Warehouse S3 Buckets will no longer be used by Pipeline, they will be archived.

Configured my Lake and Warehouse S3 buckets to transition the files into Glacier Deep Archive S3 Buckets after 30 days, since these files will not be used anymore but is good to keep in case they are needed for future auditing and checking purposes.



Transition current versions of objects between storage classes

Choose transitions to move current versions of objects between storage classes based on your use case scenario and performance access requirements. These transitions start from when the objects are created and are consecutively applied. Learn more

Choose storage class transitions: Glacier Deep Archive ▼

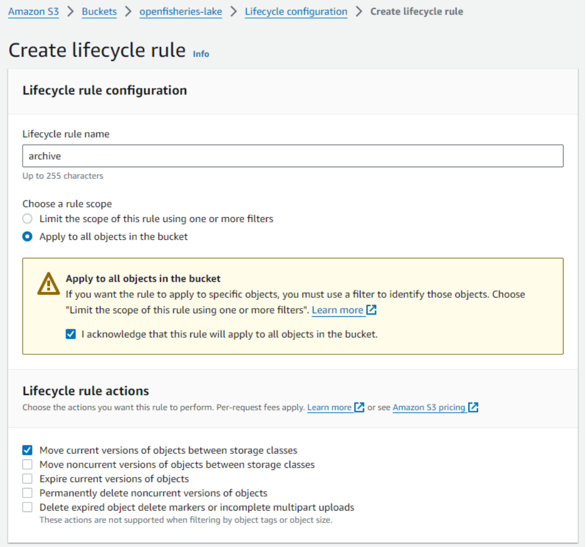
Days after object creation: 30

Remove

Add transition

Using Glacier instead of S3 helps optimize cloud expenses by cutting down on costs as Glacier is cheaper than S3.

Deep Archive is used as it's the cheapest option.



Amazon S3 > Buckets > openfisheries-lake > Lifecycle configuration > Create lifecycle rule

Create lifecycle rule Info

Lifecycle rule configuration

Lifecycle rule name: archive

Up to 255 characters

Choose a rule scope

○ Limit the scope of this rule using one or more filters

● Apply to all objects in the bucket

Apply to all objects in the bucket

If you want the rule to apply to specific objects, you must use a filter to identify those objects. Choose "Limit the scope of this rule using one or more filters". Learn more

☑ I acknowledge that this rule will apply to all objects in the bucket.

Lifecycle rule actions

Choose the actions you want this rule to perform. Per-request fees apply. Learn more or see Amazon S3 pricing

☑ Move current versions of objects between storage classes

☐ Move noncurrent versions of objects between storage classes

☐ Expire current versions of objects

☐ Permanently delete noncurrent versions of objects

☐ Delete expired object delete markers or incomplete multipart uploads

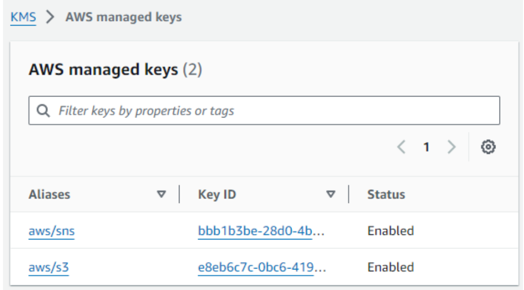
These actions are not supported when filtering by object tags or object size.

3.9 Security

Since I am restricted from IAM, I decided to configure KMS to ensure the pipeline is secured and protected.

For each S3 bucket, enable default server-side encryption using SSE-KMS that protects S3 objects when no other key is defined.

KMS key to protect SNS data when no other key is defined.



KMS > AWS managed keys

AWS managed keys (2)

Filter keys by properties or tags

Aliases	Key ID	Status
aws/sns	bbb1b3be-28d0-4b...	Enabled
aws/s3	e8eb6c7c-0bc6-419...	Enabled

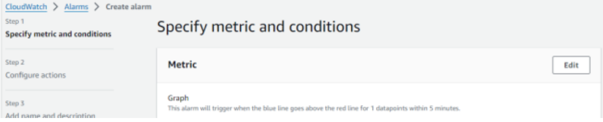
4 Monitoring (CloudWatch)

To monitor the health status of the Pipeline, I created an Alarm to alert potential issues that could arise from Glue usage and Errors that occur when trying to fetch data from API.

The alarm will trigger when the blue line goes beyond the threshold within 5 minutes (to alert engineers about any sudden sharp rise in spending).

To alert issues with fetching data from OpenFisheries API, any error that occurs will be flagged as an error occurring that would cause the error count to exceed the threshold of 0 error.

To monitor Glue usage, the Billing for Glue is used, and the bill for Glue cannot exceed the maximum threshold set in USD. The threshold is $2. If Glue usage deviates by $2 from normal expected usage, this anomalous activity will be notified to the engineers.



CloudWatch > Alarms > Create alarm

Step 1
Specify metric and conditions

Step 2
Configure actions

Step 3
Add name and description

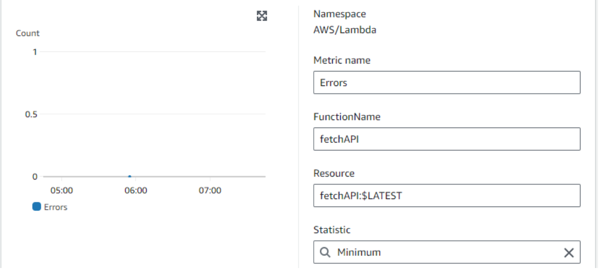
Specify metric and conditions

Metric Edit

Graph

This alarm will trigger when the blue line goes above the red line for 1 datapoints within 5 minutes.

Unresolved error when fetching data from API:



Namespace: AWS/Lambda

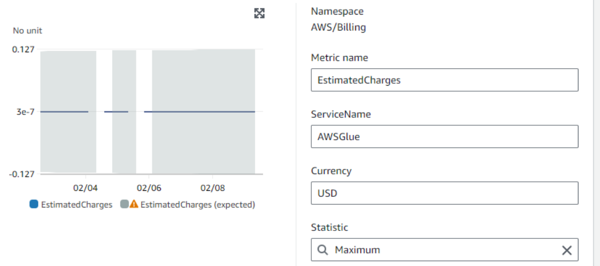
Metric name: Errors

FunctionName: fetchAPI

Resource: fetchAPI:$LATEST

Statistic: Minimum

Glue's Billing to ensure no exceed budget:



Namespace: AWS/Billing

Metric name: EstimatedCharges

ServiceName: AWSGlue

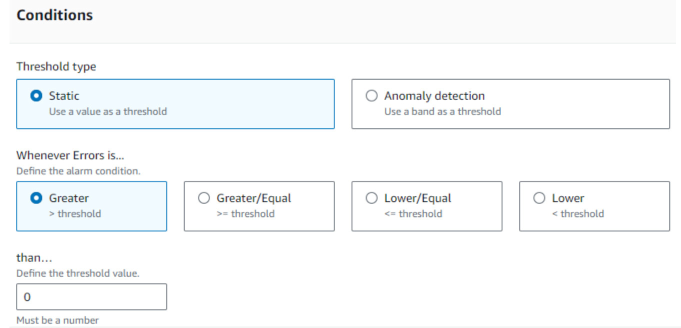
Currency: USD

Statistic: Maximum

4.1 Critical error (unable to be resolved) in fetching data from Openfisheries API

When an error occurs, the blue line WILL exceed the 0 threshold, hence engineers will receive the notification of the failure.

The alarm will send a notification to the fetching_API topic, and the email endpoint is set to the engineer's email address.



Conditions

Threshold type

● Static
Use a value as a threshold

○ Anomaly detection
Use a band as a threshold

Whenever Errors is...
Define the alarm condition.

● Greater
> threshold

○ Greater/Equal
>= threshold

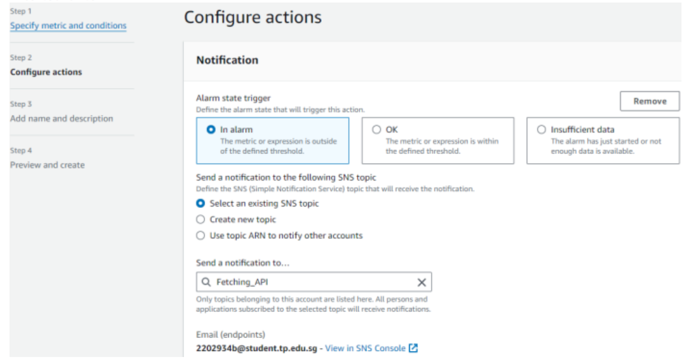
○ Lower/Equal
<= threshold

○ Lower
< threshold

than...
Define the threshold value.

0

Must be a number



Configure actions

Notification

Alarm state trigger
Define the alarm state that will trigger this action.

● In alarm
The metric or expression is outside of the defined threshold.

○ OK
The metric or expression is within the defined threshold.

○ Insufficient data
The alarm has just started or not enough data is available.

Send a notification to the following SNS topic
Define the SNS (Simple Notification Service) topic that will receive the notification.

● Select an existing SNS topic
○ Create new topic
○ Use topic ARN to notify other accounts

Send a notification to...

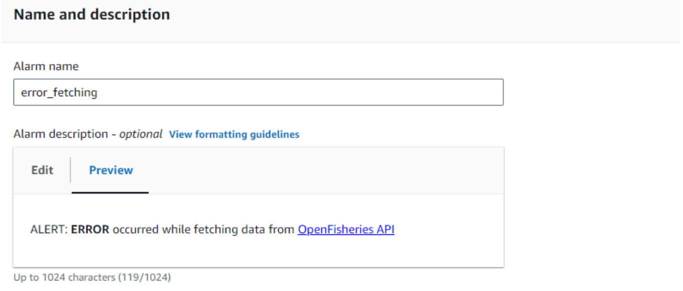
Fetching_API

Only topics belonging to this account are listed here. All persons and applications subscribed to the selected topic will receive notifications.

Email (endpoints)
2202934b@student.tp.edu.sg - View in SNS Console

This is the message that will appear in the email message if the alarm is triggered.

The engineer can recognize the email using the distinct alarm name, to know that the notification alarm is regarding an error with fetching the REST API from OpenFisheries.



Name and description

Alarm name

error_fetching

Alarm description - *optional* View formatting guidelines

ALERT: **ERROR** occurred while fetching data from OpenFisheries API

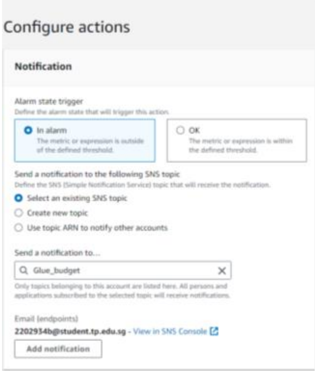
Up to 1024 characters (119/1024)

4.2 Exceeding Glue Budget

If Glue usage exceeds the designated budget threshold, the Engineer's email will receive a message to warn him/her.

Mail will alert the engineer to assess the situation and disable service (if needed) before usage spirals out of control.

Do not auto-disable service as approval from an engineer is crucial before any major change that disrupts the pipeline.



Configure actions

Notification

Alarm state trigger
Define the alarm state that will trigger this action.

● In alarm
The metric or expression is outside of the defined threshold.

○ OK
The metric or expression is within the defined threshold.

Send a notification to the following SNS topic
Define the SNS (Simple Notification Service) topic that will receive the notification.

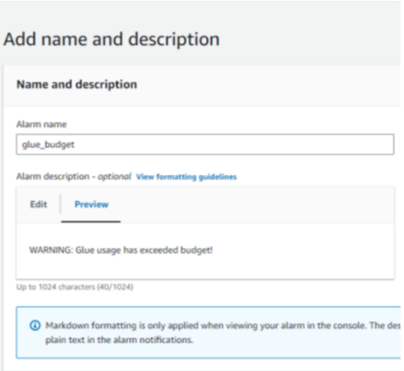
● Select an existing SNS topic
○ Create new topic
○ Use topic ARN to notify other accounts

Send a notification to...

Glue_budget

Only topics belonging to this account are listed here. All persons and applications subscribed to the selected topic will receive notifications.

Email (endpoints)
2202934b@student.tp.edu.sg - View in SNS Console



Add name and description

Name and description

Alarm name

glue_budget

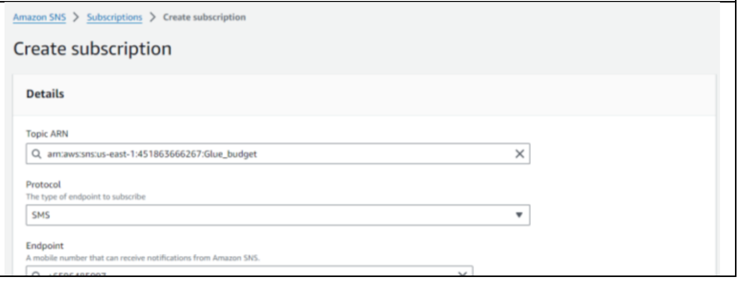
Alarm description - *optional* View formatting guidelines

WARNING: Glue usage has exceeded budget!

Up to 1024 characters (40/1024)

Markdown formatting is only applied when viewing your alarm in the console. The des plain text in the alarm notifications.

Since exceeding budget is a SERIOUS concern that demands immediate action as the consequences could be dire, a subscription was also added to send an SMS to the engineer's number (because he/she may not check email in time to assess the problem) whereas he should have a phone with him at all times.



Amazon SNS > Subscriptions > Create subscription

Create subscription

Details

Topic ARN

arn:aws:sns:us-east-1:451863666267:Glue_budget

Protocol
The type of endpoint to subscribe

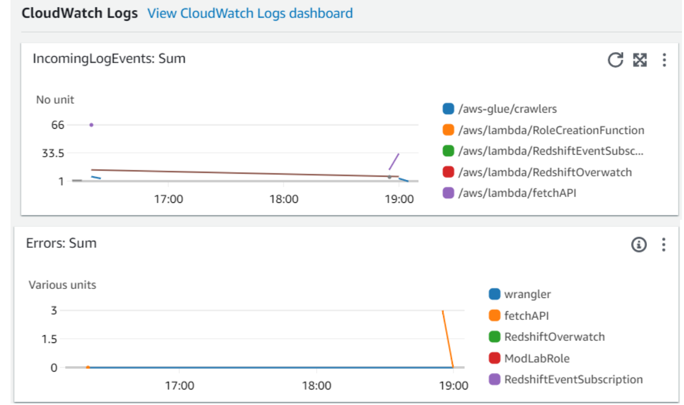
SMS

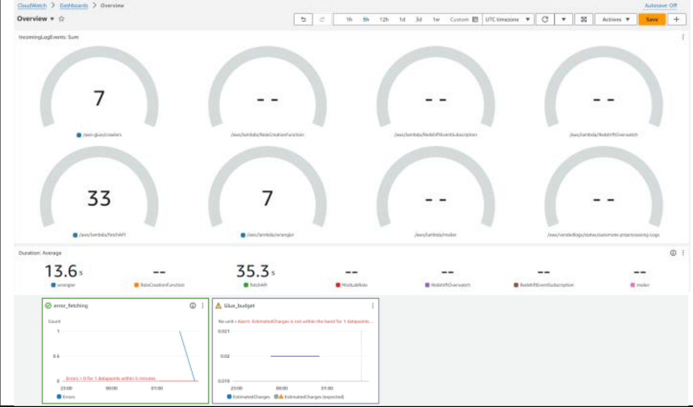
Endpoint
A mobile number that can receive notifications from Amazon SNS.

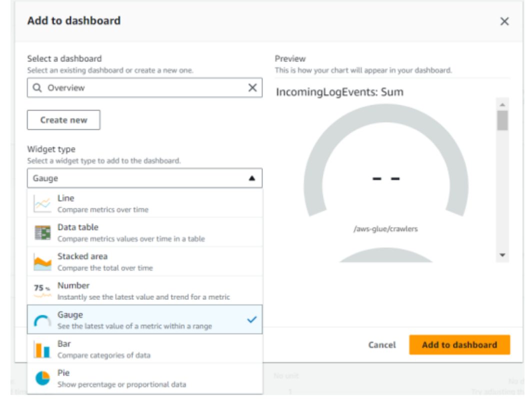
4.3 Logs for all services are enabled:

Confirm that logging is enabled for all relevant AWS services. This ensures that comprehensive logs are available for analysis in case of issues or for routine monitoring.

If more types of unique charts and/or a Dashboard with only relevant metrics is required by the Engineer to monitor the Pipeline, a Dashboard can be created. An example of CloudWatch metrics dashboard is shown below:







Add to dashboard ✕

Select a dashboard
Select an existing dashboard or create a new one.

Overview ✕

Create new

Widget type
Select a widget type to add to the dashboard.

Gauge ▲

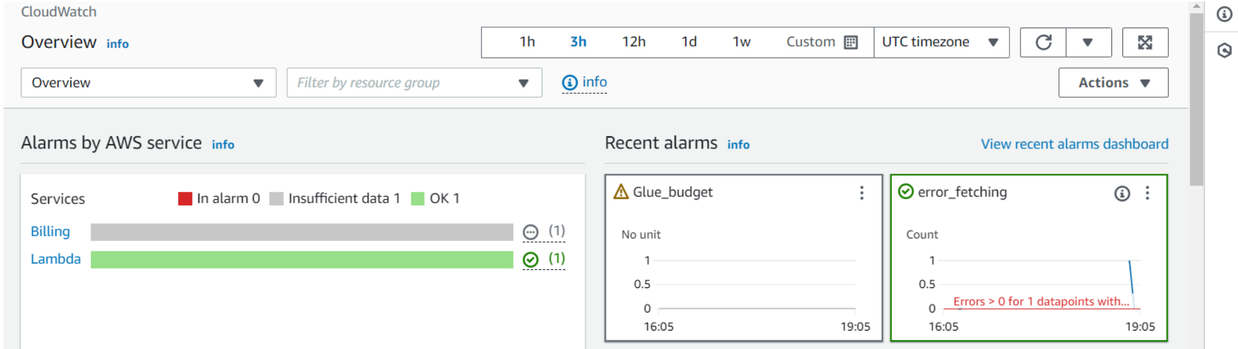
- Line — Compare metrics over time
- Data table — Compare metrics values over time in a table
- Stacked area — Compare the total over time
- Number — Instantly see the latest value and trend for a metric
- Gauge — See the latest value of a metric within a range ✓
- Bar — Compare categories of data
- Pie — Show percentage or proportional data

Preview
This is how your chart will appear in your dashboard.

Cancel Add to dashboard

4.4 Alarms enabled:

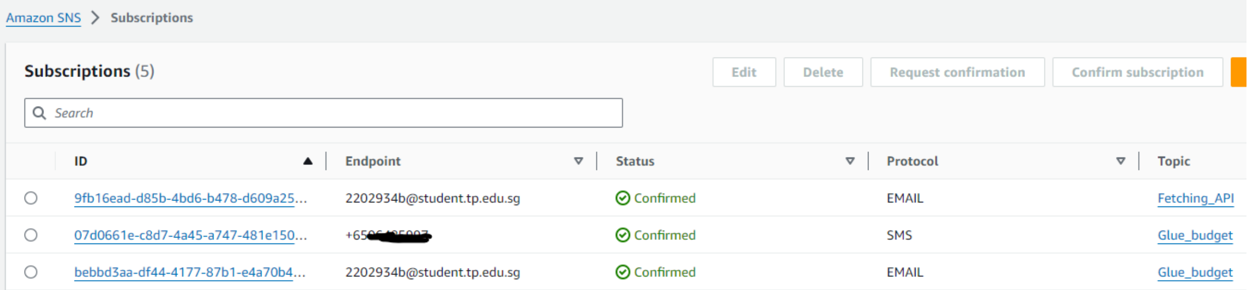
When the issue is fixed (after re-running the Step function and successful run), the Service alarm status will change back to OK.



Services	🟥 In alarm 0	Insufficient data 1	🟩 OK 1
Billing		(1)	
Lambda			(1)

4.5 SNS Subscription:

Confirmed and activated, ready to send message to engineer when needed.



Amazon SNS > Subscriptions

Subscriptions (5)

Edit Delete Request confirmation Confirm subscription

Search

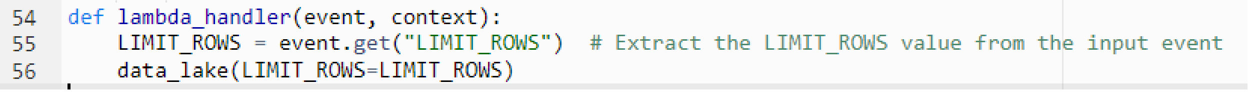
	ID	Endpoint	Status	Protocol	Topic
○	9fb16ead-d85b-4bd6-b478-d609a25...	2202934b@student.tp.edu.sg	✓ Confirmed	EMAIL	Fetching_API
○	07d0661e-c8d7-4a45-a747-481e150...	+65[redacted]	✓ Confirmed	SMS	Glue_budget
○	bebbd3aa-df44-4177-87b1-e4a70b4...	2202934b@student.tp.edu.sg	✓ Confirmed	EMAIL	Glue_budget

[*To see an SNS Mail in action, refer to section 6*]

5. Deployment (Automating Pipeline)

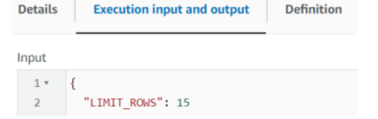
1. Added feature to dynamically declare how many datasets to be used.

Modified fetchAPI.py Lambda function to take LIMIT_ROWS from Step input to allow analysts to choose how many datasets they want to use. By default 15 datasets for countries and species are used to align with the project purpose of analyzing 30 datasets.



```
def lambda_handler(event, context):
    LIMIT_ROWS = event.get("LIMIT_ROWS")  # Extract the LIMIT_ROWS value from the input event
    data_lake(LIMIT_ROWS=LIMIT_ROWS)
```

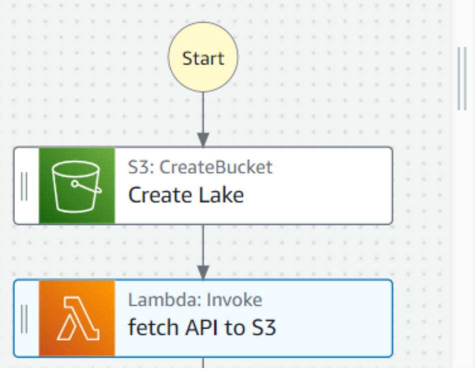
Analysts using the pipeline can conveniently use this Step function and can choose how many species and countries they want for their analysis by stating it in the input of the Step function.

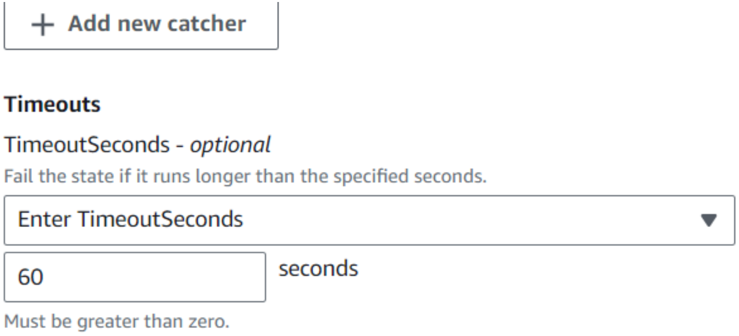


Input

```
{
 "LIMIT_ROWS": 15
```

2. Added a timeout to stop fetching API if it takes too long.





Timeouts

TimeoutSeconds - *optional*

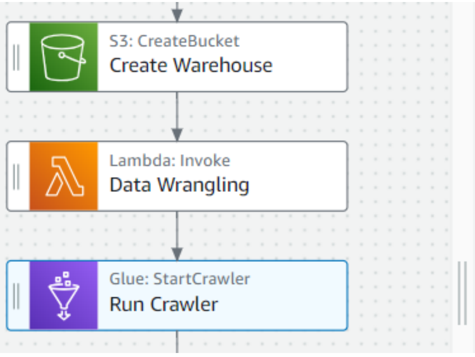
Fail the state if it runs longer than the specified seconds.

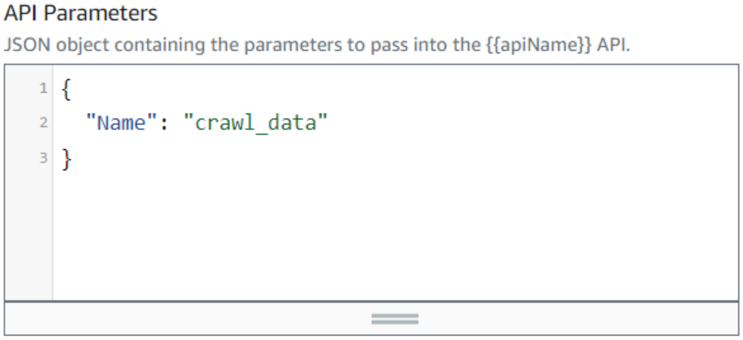
Enter TimeoutSeconds ▼

60 seconds

Must be greater than zero.

3. Run the crawler to populate (or update the existing) the Data Catalog database with data.



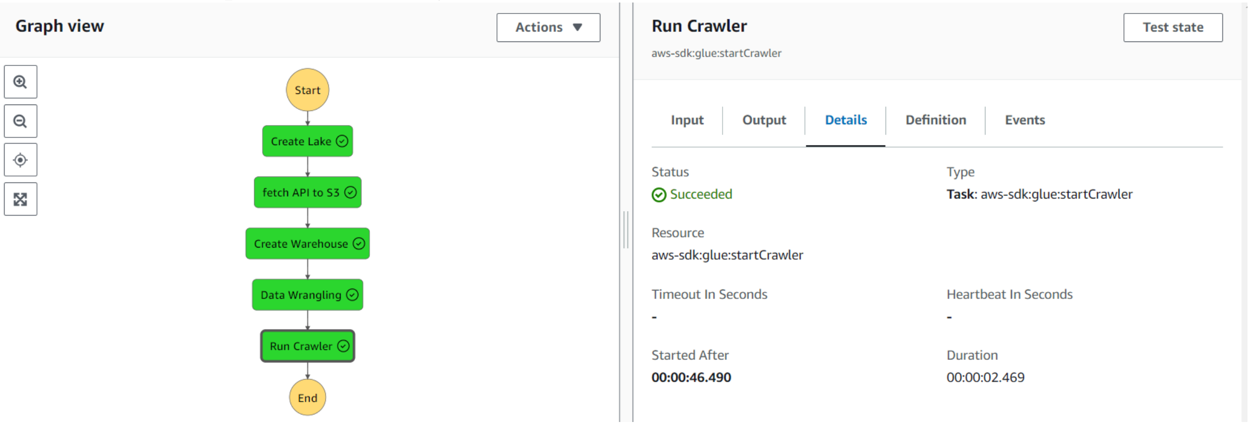


API Parameters

JSON object containing the parameters to pass into the {{apiName}} API.

```
{
  "Name": "crawl_data"
}
```

4. Prove of that Step workflow is fully functional and works:



Run Crawler

aws-sdk:glue:startCrawler

Status: ✓ Succeeded

Type: **Task**: aws-sdk:glue:startCrawler

Resource: aws-sdk:glue:startCrawler

Timeout In Seconds: -

Heartbeat In Seconds: -

Started After: **00:00:46.490**

Duration: 00:00:02.469

This Step function fully prepares the data for analyst's use. Analyst only has to use Athena > Query editor to query the data they need to build their visualizations and report their findings and insights.

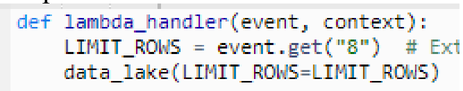
6. Evidence that Services work as intended

Footage of configuring Pipeline:

https://youtu.be/qmR2lkdPQWs (Had to redo as my Main Lab got terminated)

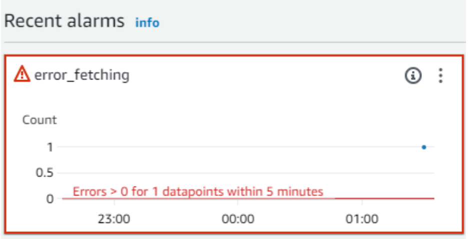
Prove that CloudWatch Alert powered by SNS works:

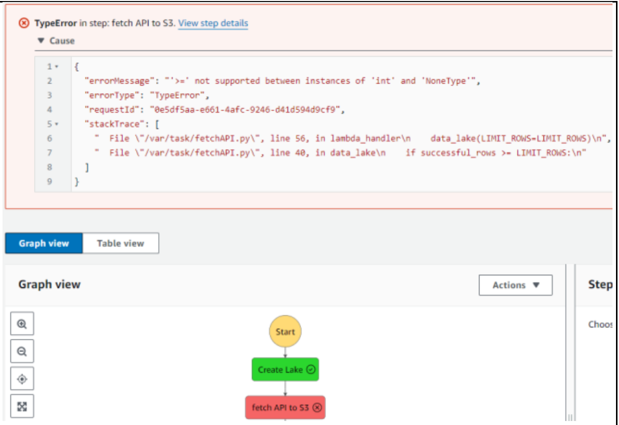
To trigger the Error alert, I broke fetchAPI.py script such that it will encounter an error:



```
def lambda_handler(event, context):
    LIMIT_ROWS = event.get("8")  # Ext
    data_lake(LIMIT_ROWS=LIMIT_ROWS)
```

After Step execution failed, this log turned red:

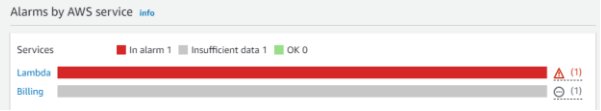


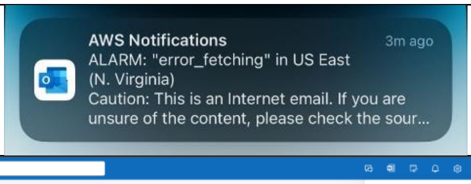


TypeError in step: fetch API to S3. View step details

▼ Cause

```
{
  "errorMessage": "'>=' not supported between instances of 'int' and 'NoneType'",
  "errorType": "TypeError",
  "requestId": "0e5df5aa-e661-4afc-9246-d41d594d9cf9",
  "stackTrace": [
    "  File \"/var/task/fetchAPI.py\", line 56, in lambda_handler\n    data_lake(LIMIT_ROWS=LIMIT_ROWS)\n",
    "  File \"/var/task/fetchAPI.py\", line 40, in data_lake\n    if successful_rows >= LIMIT_ROWS:\n"
  ]
}
```





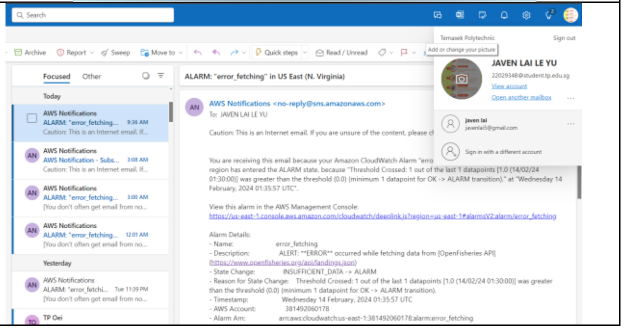
AWS Notifications 3m ago

ALARM: "error_fetching" in US East (N. Virginia)

Caution: This is an Internet email. If you are unsure of the content, please check the sour...

As shown above, when Lambda code has unresolved issues when run, an alarm is triggered to send an email to the engineer to alert them about the issue. The email sent notifies the issue that went wrong.

After testing that CloudWatch Alarm + SNS works properly. The broken fetchAPI function is then reverted to its original working version.



AWS Notifications	ALARM: "error_fetching...	9:36 AM
	Caution: This is an Internet email. If...	
AWS Notifications	AWS Notification - Subs...	3:08 AM
	Caution: This is an Internet email. If...	
AWS Notifications	ALARM: "error_fetching...	3:00 AM
	[You don't often get email from no...	
AWS Notifications	ALARM: "error_fetching...	12:01 AM
	[You don't often get email from no...	
AWS Notifications	ALARM: "error_fetchi...	Tue 11:39 PM
	[You don't often get email from no...	
TP Oei		

JAVEN LAI LE YU
22029348@student.tp.edu.sg

javen lai
javenlai5@gmail.com

ALARM: "error_fetching" in US East (N. Virginia)

AWS Notifications <no-reply@sns.amazonaws.com>
To: JAVEN LAI LE YU

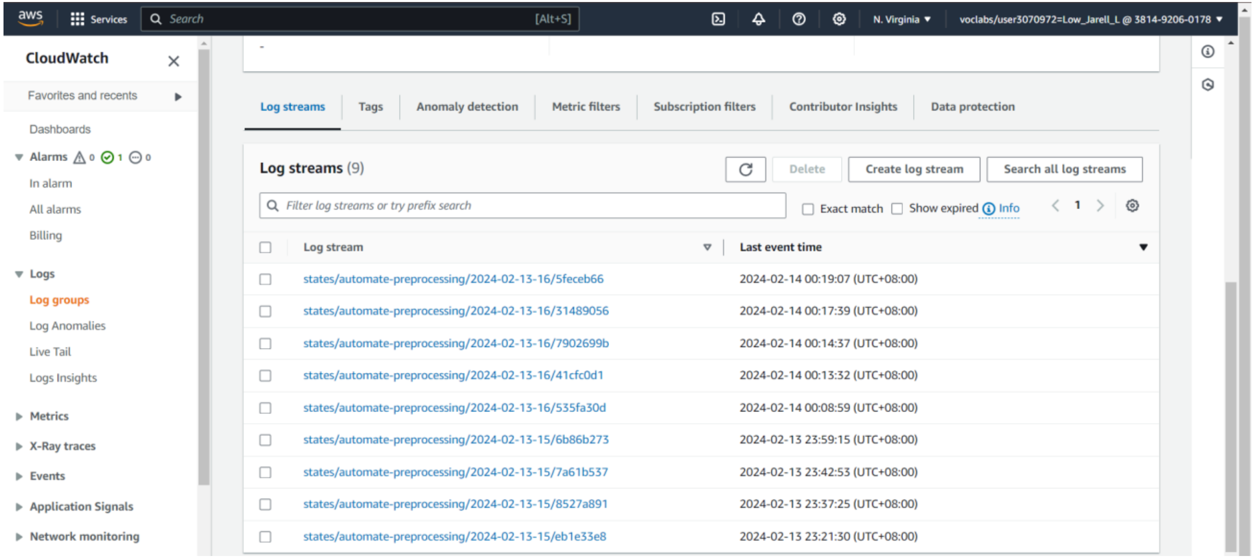
Caution: This is an Internet email. If you are unsure of the content, please ch

You are receiving this email because your Amazon CloudWatch Alarm "erro region has entered the ALARM state, because "Threshold Crossed: 1 out of the last 1 datapoints [1.0 (14/02/24 01:30:00)] was greater than the threshold (0.0) (minimum 1 datapoint for OK -> ALARM transition)." at "Wednesday 14 February, 2024 01:35:57 UTC".

View this alarm in the AWS Management Console:
https://us-east-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-east-1#alarmsV2:alarm/error_fetching

Alarm Details:
- Name: error_fetching
- Description: ALERT: **ERROR** occurred while fetching data from [OpenFisheries API](https://www.openfisheries.org/api/landings.json)
- State Change: INSUFFICIENT_DATA -> ALARM
- Reason for State Change: Threshold Crossed: 1 out of the last 1 datapoints [1.0 (14/02/24 01:30:00)] was greater than the threshold (0.0) (minimum 1 datapoint for OK -> ALARM transition).
- Timestamp: Wednesday 14 February, 2024 01:35:57 UTC
- AWS Account: 381492060178
- Alarm Arn: arn:aws:cloudwatch:us-east-1:381492060178:alarm:error_fetching

Proof that CloudWatch Logs for services are enabled:



Log streams (9)

Log stream	Last event time
states/automate-preprocessing/2024-02-13-16/5feceb66	2024-02-14 00:19:07 (UTC+08:00)
states/automate-preprocessing/2024-02-13-16/31489056	2024-02-14 00:17:39 (UTC+08:00)
states/automate-preprocessing/2024-02-13-16/7902699b	2024-02-14 00:14:37 (UTC+08:00)
states/automate-preprocessing/2024-02-13-16/41cfc0d1	2024-02-14 00:13:32 (UTC+08:00)
states/automate-preprocessing/2024-02-13-16/535fa30d	2024-02-14 00:08:59 (UTC+08:00)
states/automate-preprocessing/2024-02-13-15/6b86b273	2024-02-13 23:59:15 (UTC+08:00)
states/automate-preprocessing/2024-02-13-15/7a61b537	2024-02-13 23:42:53 (UTC+08:00)
states/automate-preprocessing/2024-02-13-15/8527a891	2024-02-13 23:37:25 (UTC+08:00)
states/automate-preprocessing/2024-02-13-15/eb1e33e8	2024-02-13 23:21:30 (UTC+08:00)

Checklist (configurations of services)

Logging and Monitoring Mechanism for Compliance

Lambda Functions:

- Enable CloudWatch Logs for Lambda functions. (Y)
- Implemented structured logging of unsuccessful operations compliance and auditing. (Y)
- Configure CloudWatch Alarms for critical Lambda metrics: Errors + Exceeding Budget. (Y)

S3 Buckets:

- Enable CloudWatch Metrics for S3 buckets. (Y)
- AWS CloudTrail for S3 bucket access monitoring. Not needed as API not implemented. (N)
- Configure S3 bucket access logging for compliance. (N)
- Enable version control to track any changes made. Only applied to insights bucket (Y)

AWS Glue:

- Enable CloudWatch Logs for AWS Glue ETL jobs and CloudWatch Alarms for Glue job metrics. (Y)

Athena:

- Enable CloudWatch Metrics for Athena. (Y)
- Monitor query execution times and errors. (Y)
- Set up CloudWatch Alarms for Athena query metrics. (Y)

Amazon QuickSight – Visualization tool replaced with Tableau due to AWS budget. (N)

Access Control Management (IAM, Users, Roles, and Policies) – Restrict access from school (N)

Ideal Configurations for Each Component

S3 Bucket:

- Block public access to the S3 bucket. (Y)
- Implement versioning and logging for S3 bucket. – Track all activity in insights bucket (Y)
- Configure lifecycle policies for data retention. – Archive Lake and Warehouse after usage (Y)

AWS Glue:

- Set up AWS Glue crawlers to automatically infer the structure of the data. (Y)
- Utilize Glue triggers for job scheduling. – Used Step function to trigger Crawler (-)
- Optimize Glue job configurations for resource utilization. – Not applicable (N)

Amazon Athena:

- Organized data into Views for usage for analytics (Y)
- Implement query execution time limits for resource optimization. – No need since dataset is small (N)
- Define query result location for efficient retrieval. – Created Views to store in insights S3 (Y)

Tableau:

Plan-Perform-Monitor-Reflect (PPMR)

https://tasks.office.com/tp.edu.sg/en-US/Home/Planner/#/plantaskboard?groupId=cb0614be-0efc-4c92-a96a-975d80323615&planId=dM_PKRaeNkqXQGMZPLXNssgACuh1

Additional Info

My original Lab, Javen_Lai was terminated and Low_Jarell is used in the second half where I had to redo everything as Javen_Lai progress was completely gone.

I skipped API Gateway as a trigger for Pipeline considering how OpenFisheries was last updated in 2018, it wouldn't be efficient to have a live hook to detect changes in API and update accordingly. Instead, I made the Pipeline into a Step function, so analysts can manually update the batch data if needed.

I skipped QuickSight since Tableau is used for visualizations, hence redundant to do visualizations twice which is why I only did advanced analytics to gather all insights in 1 round of EDA.

References:

- https://tplms.polite.edu.sg/d2l/le/enhancedSequenceViewer/372754?url=https%3A%2F%2F425c3724-b655-4262-9c13-4699f01f20f3.sequences.api.brightspace.com%2F372754%2Factivity%2F7215919%3FfilterOnDatesAndDepth%3D1 [week 14 workshop to pull data from API]
- https://tplms.polite.edu.sg/d2l/le/enhancedSequenceViewer/372754?url=https%3A%2F%2F425c3724-b655-4262-9c13-4699f01f20f3.sequences.api.brightspace.com%2F372754%2Factivity%2F7839269%3FfilterOnDatesAndDepth%3D1 [week 18 workshop to enable SNS mail]
- https://youtu.be/qmR2lkdPQWs [Speed running AWS Setup]
- https://youtu.be/o_mECUFbNJg [Showcase of Dashboard + How to use and interpret]