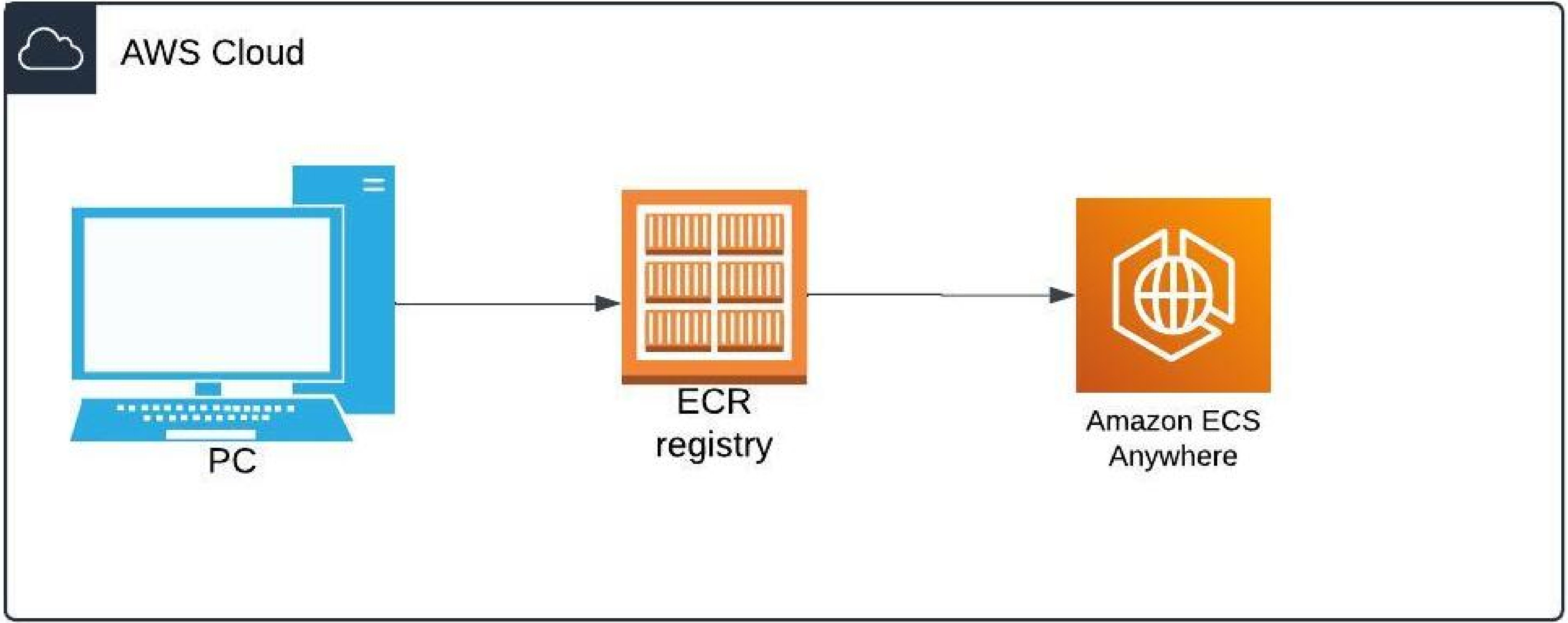
Event-Driven Data Validation and Processing Pipeline Using S3, Step Functions, and ECS

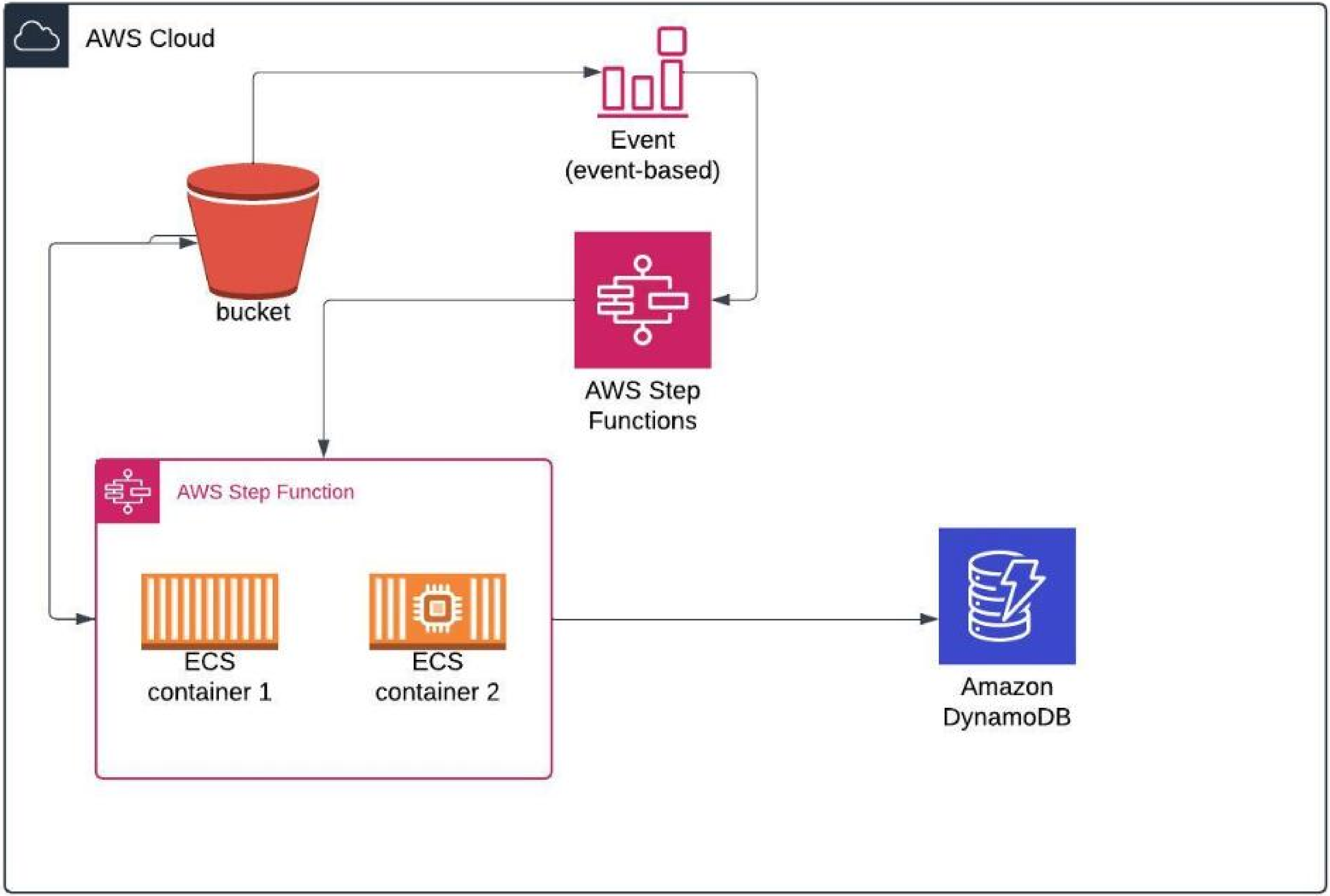
Objective

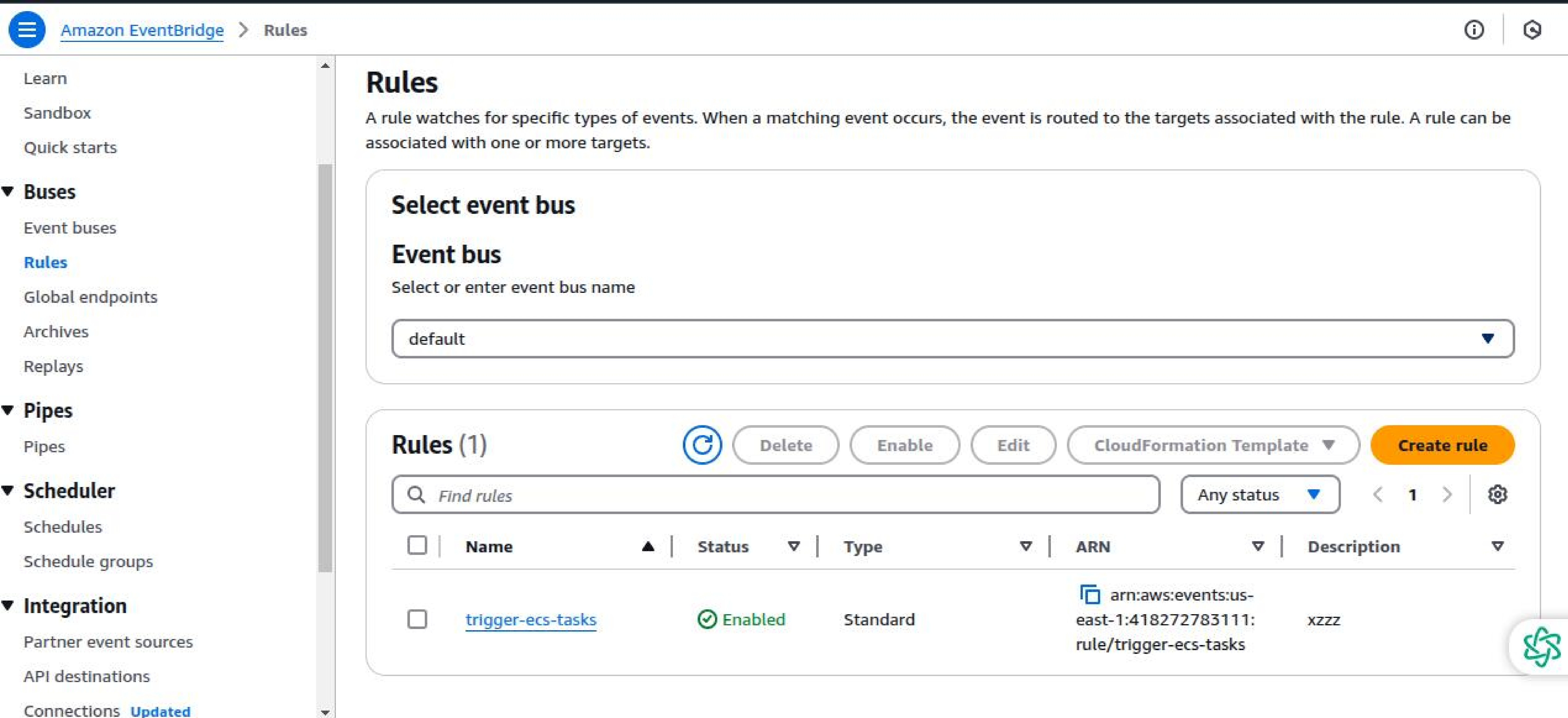
- Trigger: Use Amazon EventBridge to monitor S3 for newly created objects and trigger an AWS Step Function.
- Validate: Perform data validation using Python scripts with Pandas, ensuring data integrity.
- Move: Transfer validated data to a designated S3 location.
- Condition Check: Evaluate the output of the first validation job to determine readiness for further processing.
- Process: Trigger the second task to perform additional data transformations using Python.
- Run Tasks: Execute both Python scripts as containerized tasks on AWS Fargate with images stored in Amazon ECR.
- Track: Record the status and timestamps of the pipeline’s execution in Amazon DynamoDB.

Tools Used

- Amazon S3: As the source and destination for raw and validated data.
- Amazon EventBridge: For detecting object creation events in S3 and triggering the pipeline.
- AWS Step Functions: For orchestrating the pipeline's execution, including validation and condition checks.
- Python: For writing data validation and processing logic using Pandas.
- Amazon ECS (Fargate): For running Python scripts in a serverless, containerized environment.
- Amazon ECR: For storing container images used in ECS tasks.
- Amazon DynamoDB: For logging pipeline status and tracking execution timestamps.
- Pandas: For performing data validation and transformation operations.
- CloudWatch: For monitoring the execution flow and capturing logs.


AWS Cloud
PC
ECR registry
Amazon ECS Anywhere


AWS Cloud
Event
(event-based)
bucket
AWS Step
Functions
AWS Step Function
ECS
container 1
ECS
container 2
Amazon
DynamoDB



Rules

A rule watches for specific types of events. When a matching event occurs, the event is routed to the targets associated with the rule. A rule can be associated with one or more targets.

Select event bus

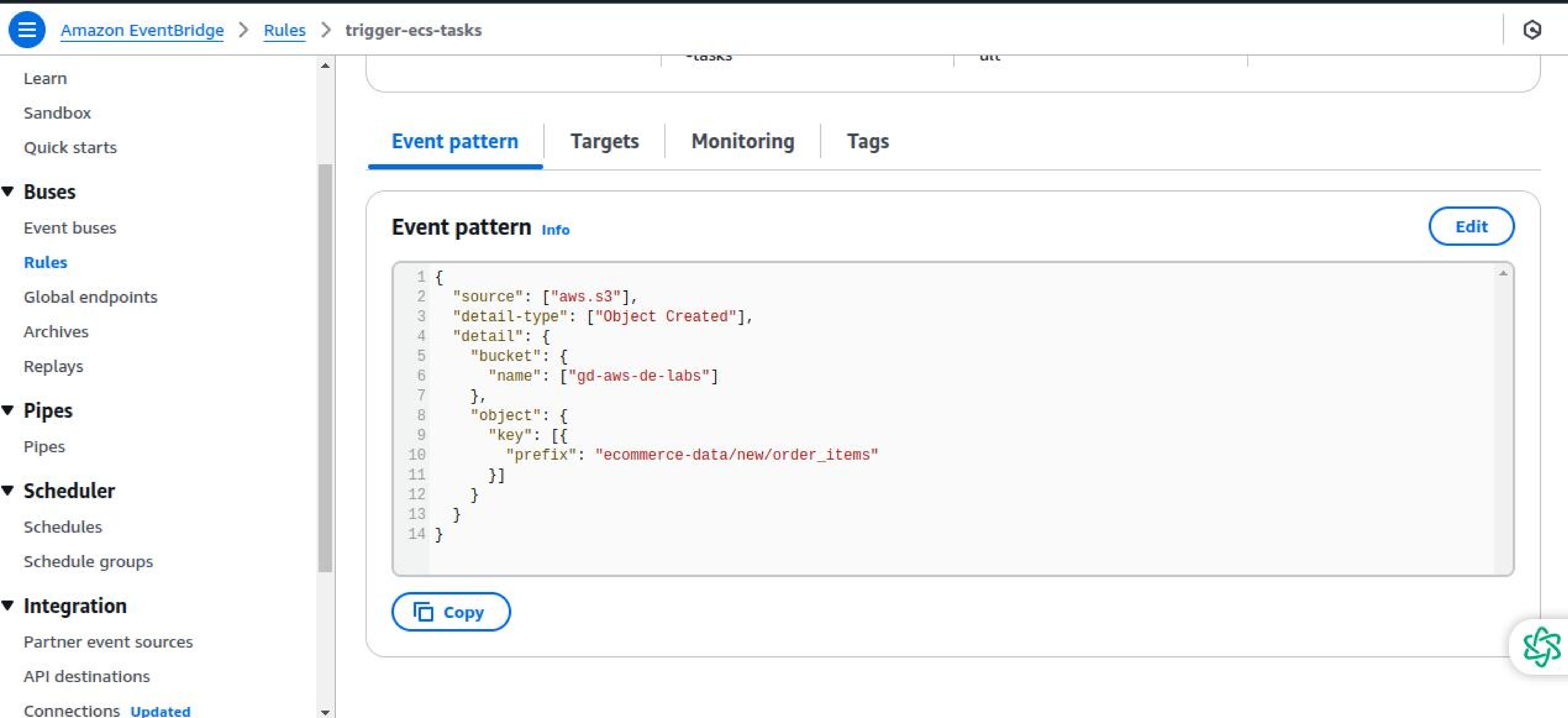
Event bus

Select or enter event bus name

default ▼

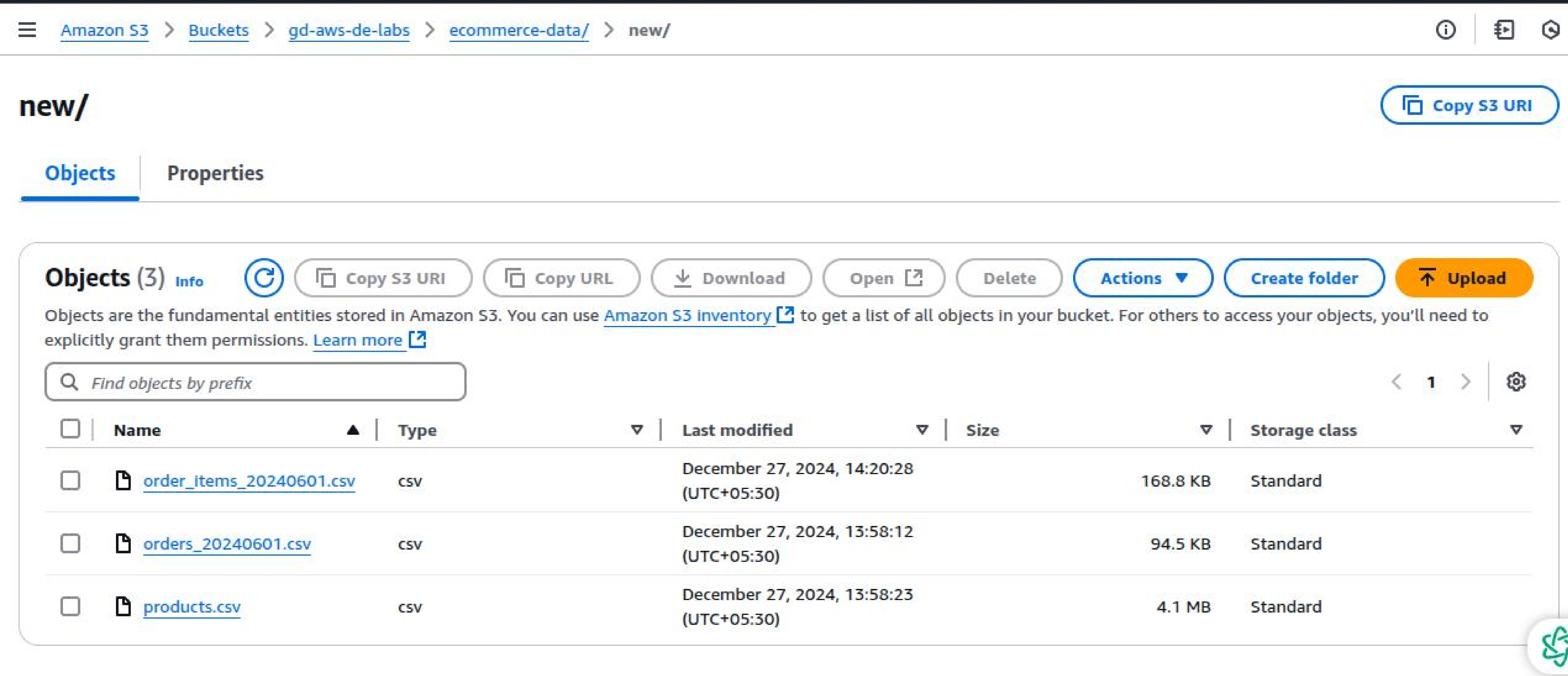
Rules (1)

□	Name ▲	Status	Type	ARN	Description
□	trigger-ecs-tasks	✓ Enabled	Standard	arn:aws:events:us-east-1:418272783111:rule/trigger-ecs-tasks	xzzz



Event pattern

```
{
  "source": ["aws.s3"],
  "detail-type": ["Object Created"],
  "detail": {
    "bucket": {
      "name": ["gd-aws-de-labs"]
    },
    "object": {
      "key": [{
        "prefix": "ecommerce-data/new/order_items"
      }]
    }
  }
}
```

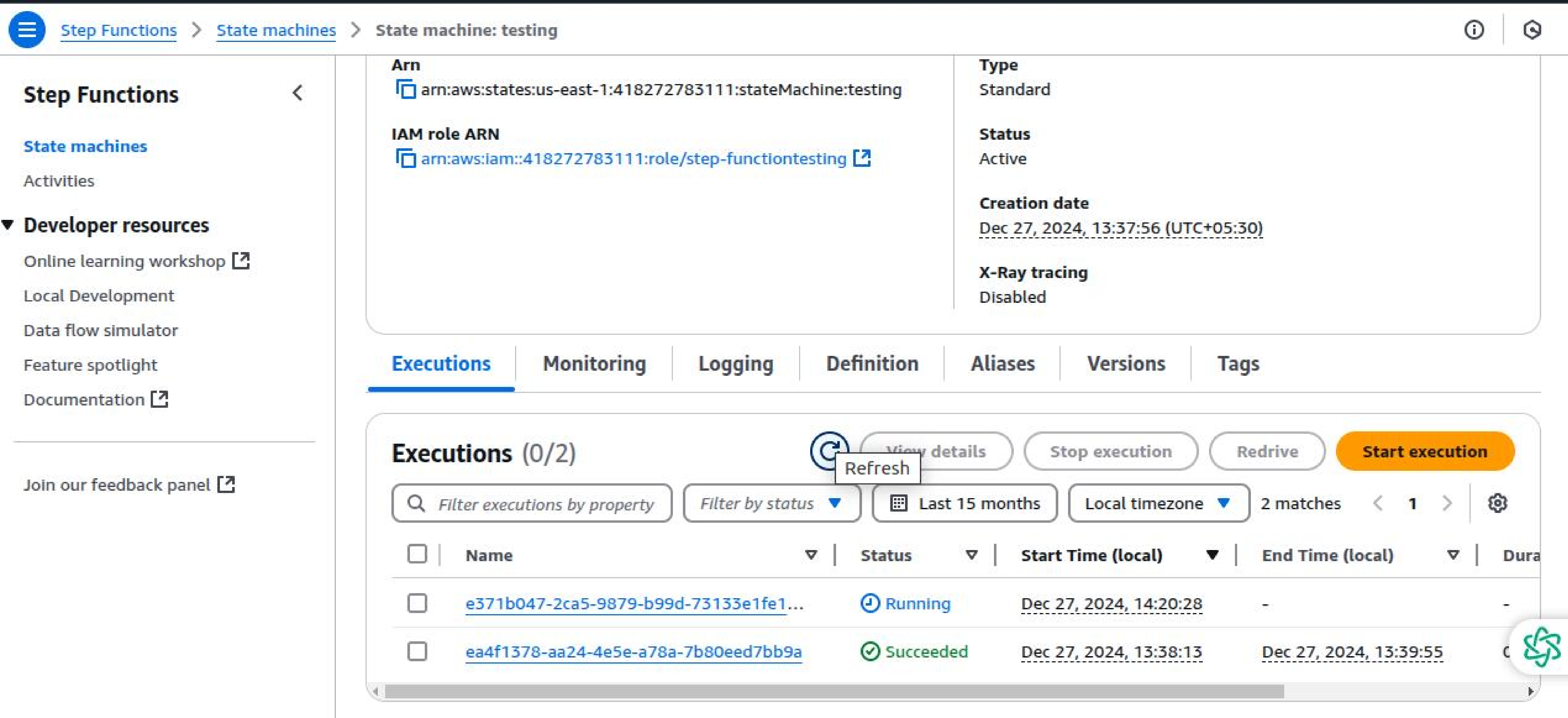


new/

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
□	order_items_20240601.csv	csv	December 27, 2024, 14:20:28 (UTC+05:30)	168.8 KB	Standard
□	orders_20240601.csv	csv	December 27, 2024, 13:58:12 (UTC+05:30)	94.5 KB	Standard
□	products.csv	csv	December 27, 2024, 13:58:23 (UTC+05:30)	4.1 MB	Standard



Arn
arn:aws:states:us-east-1:418272783111:stateMachine:testing

IAM role ARN
arn:aws:iam::418272783111:role/step-functiontesting

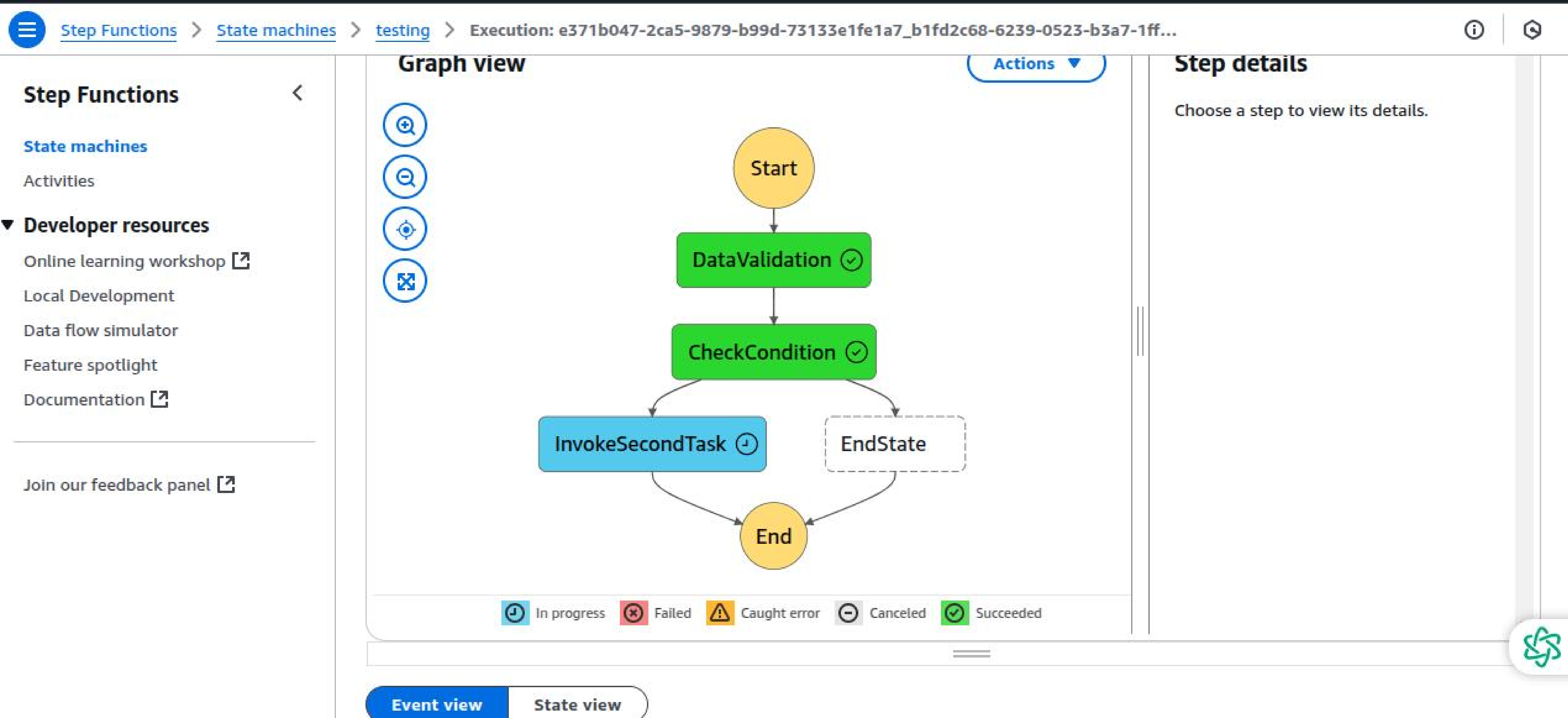
Type
Standard

Status
Active

Creation date
Dec 27, 2024, 13:37:56 (UTC+05:30)

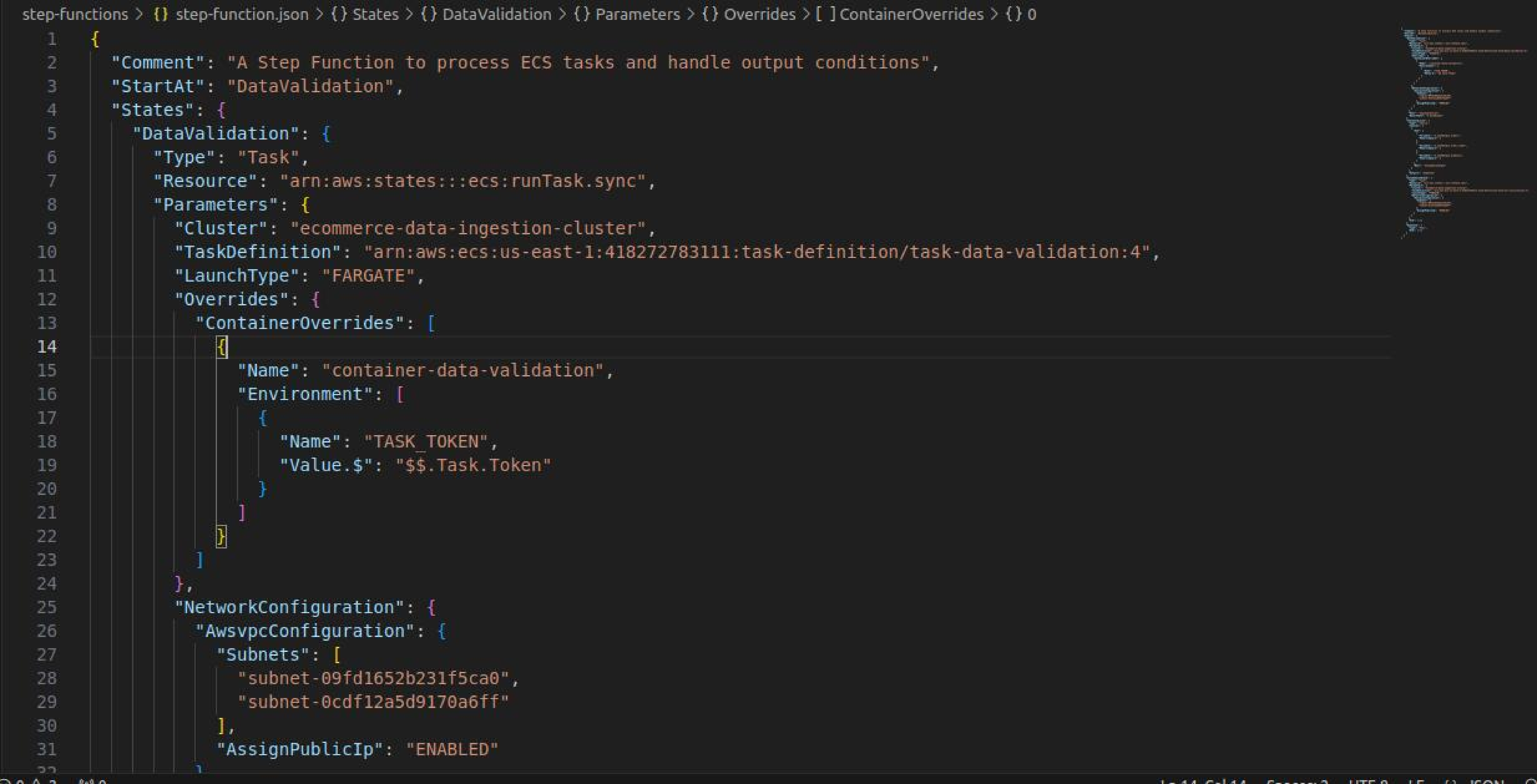
X-Ray tracing
Disabled

□	Name	Status	Start Time (local)	End Time (local)	Dura
□	e371b047-2ca5-9879-b99d-73133e1fe1...	Running	Dec 27, 2024, 14:20:28	-	-
□	ea4f1378-aa24-4e5e-a78a-7b80eed7bb9a	Succeeded	Dec 27, 2024, 13:38:13	Dec 27, 2024, 13:39:55	C

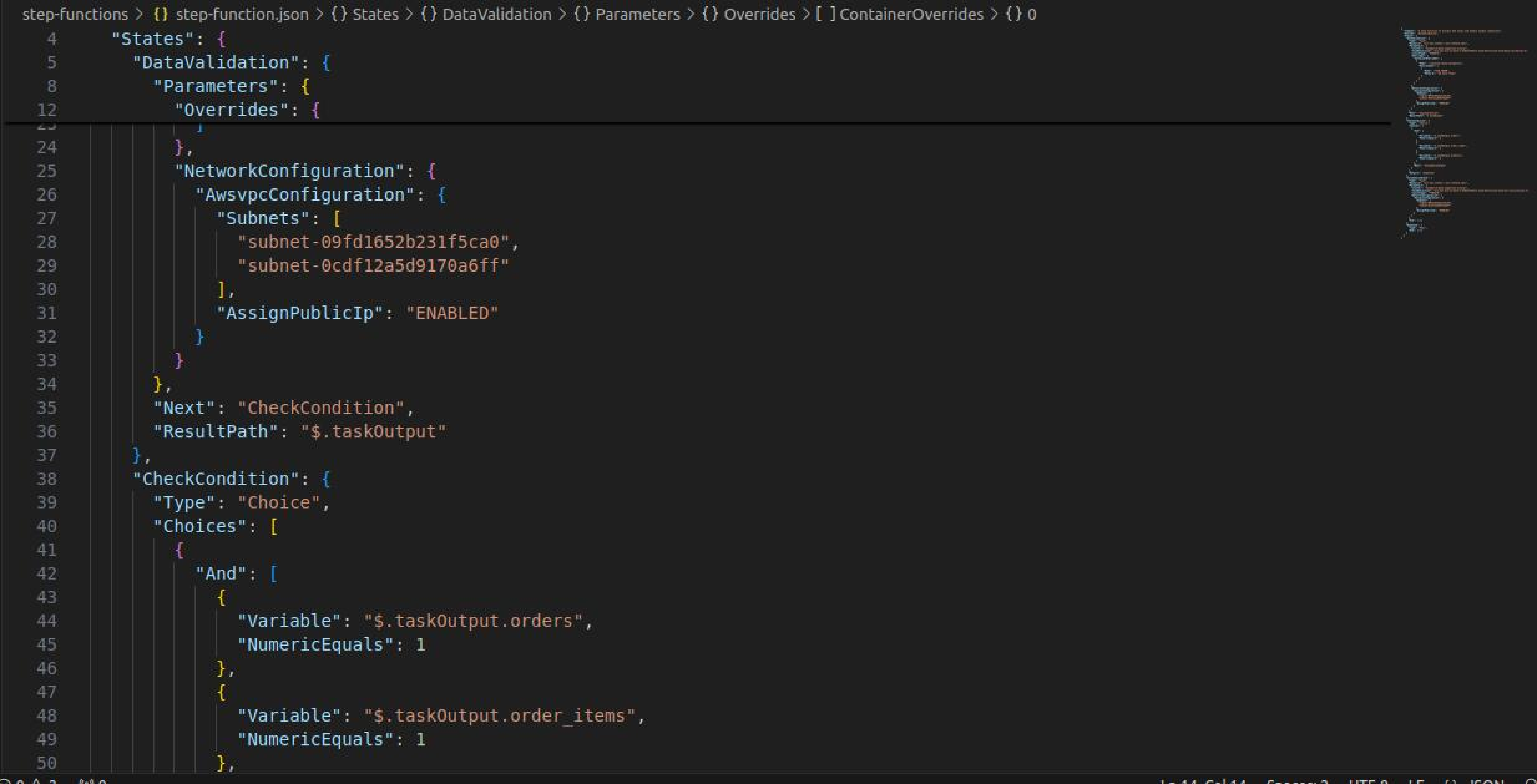


Step Functions > State machines > testing > Execution: e371b047-2ca5-9879-b99d-73133e1fe1a7_b1fd2c68-6239-0523-b3a7-1ff...

Choose a step to view its details.

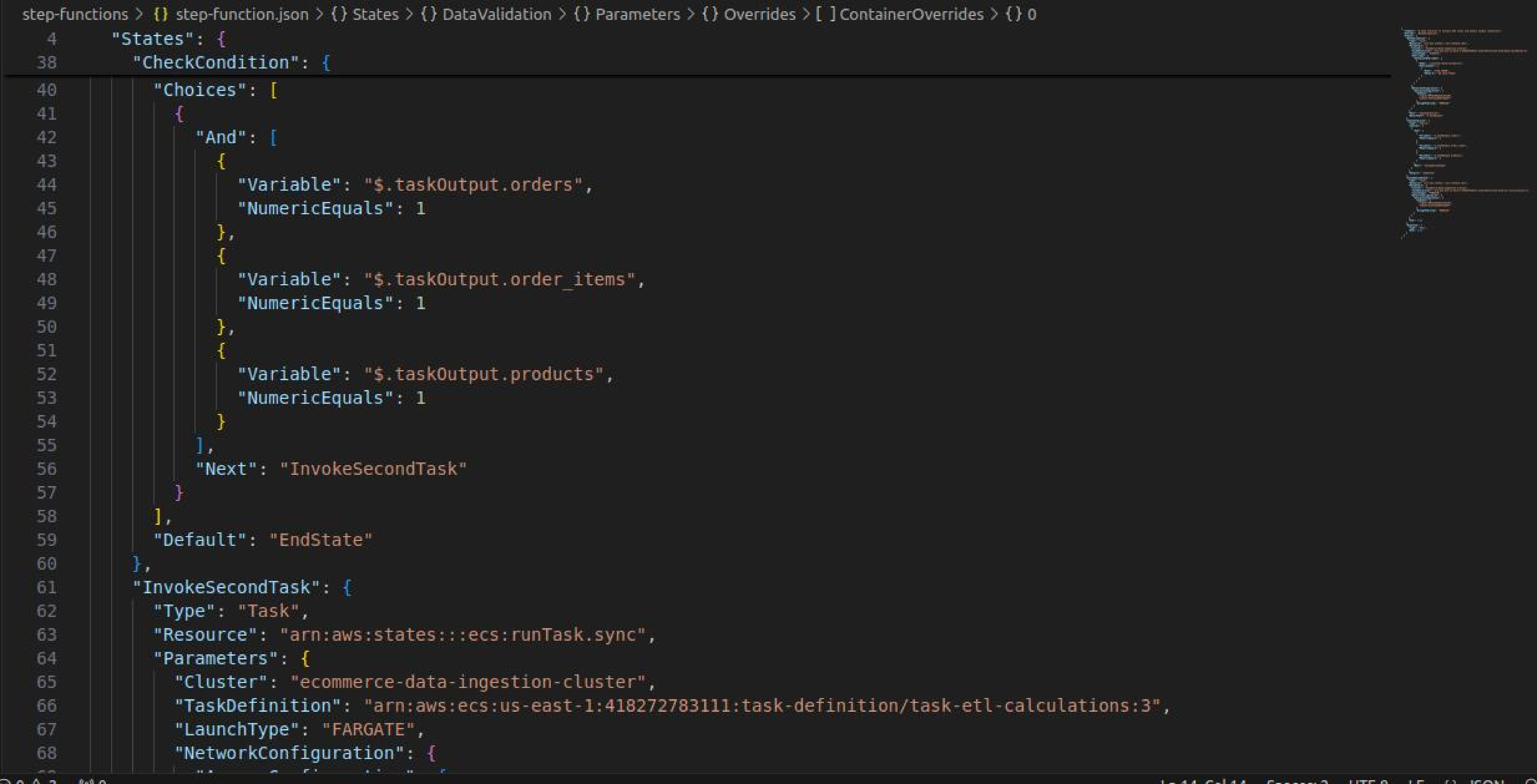

step-functions > {} step-function.json > {} States > {} DataValidation > {} Parameters > {} Overrides > [ ] ContainerOverrides > {} 0

```json
{
  "Comment": "A Step Function to process ECS tasks and handle output conditions",
  "StartAt": "DataValidation",
  "States": {
    "DataValidation": {
      "Type": "Task",
      "Resource": "arn:aws:states:::ecs:runTask.sync",
      "Parameters": {
        "Cluster": "ecommerce-data-ingestion-cluster",
        "TaskDefinition": "arn:aws:ecs:us-east-1:418272783111:task-definition/task-data-validation:4",
        "LaunchType": "FARGATE",
        "Overrides": {
          "ContainerOverrides": [
            {
              "Name": "container-data-validation",
              "Environment": [
                {
                  "Name": "TASK_TOKEN",
                  "Value.$": "$$.Task.Token"
                }
              ]
            }
          ]
        },
        "NetworkConfiguration": {
          "AwsvpcConfiguration": {
            "Subnets": [
              "subnet-09fd1652b231f5ca0",
              "subnet-0cdf12a5d9170a6ff"
            ],
            "AssignPublicIp": "ENABLED"
```

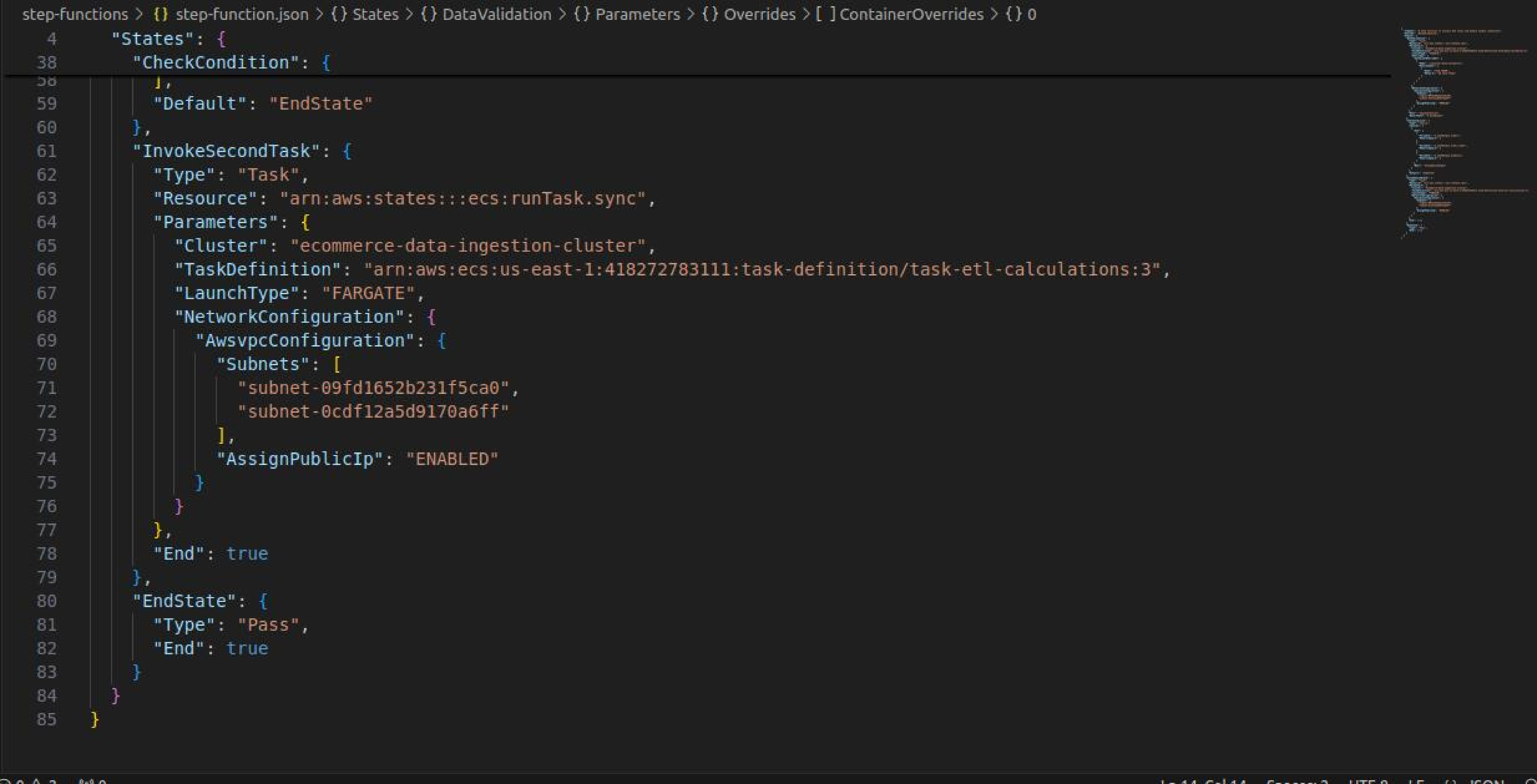


step-functions > {} step-function.json > {} States > {} DataValidation > {} Parameters > {} Overrides > [ ] ContainerOverrides > {} 0

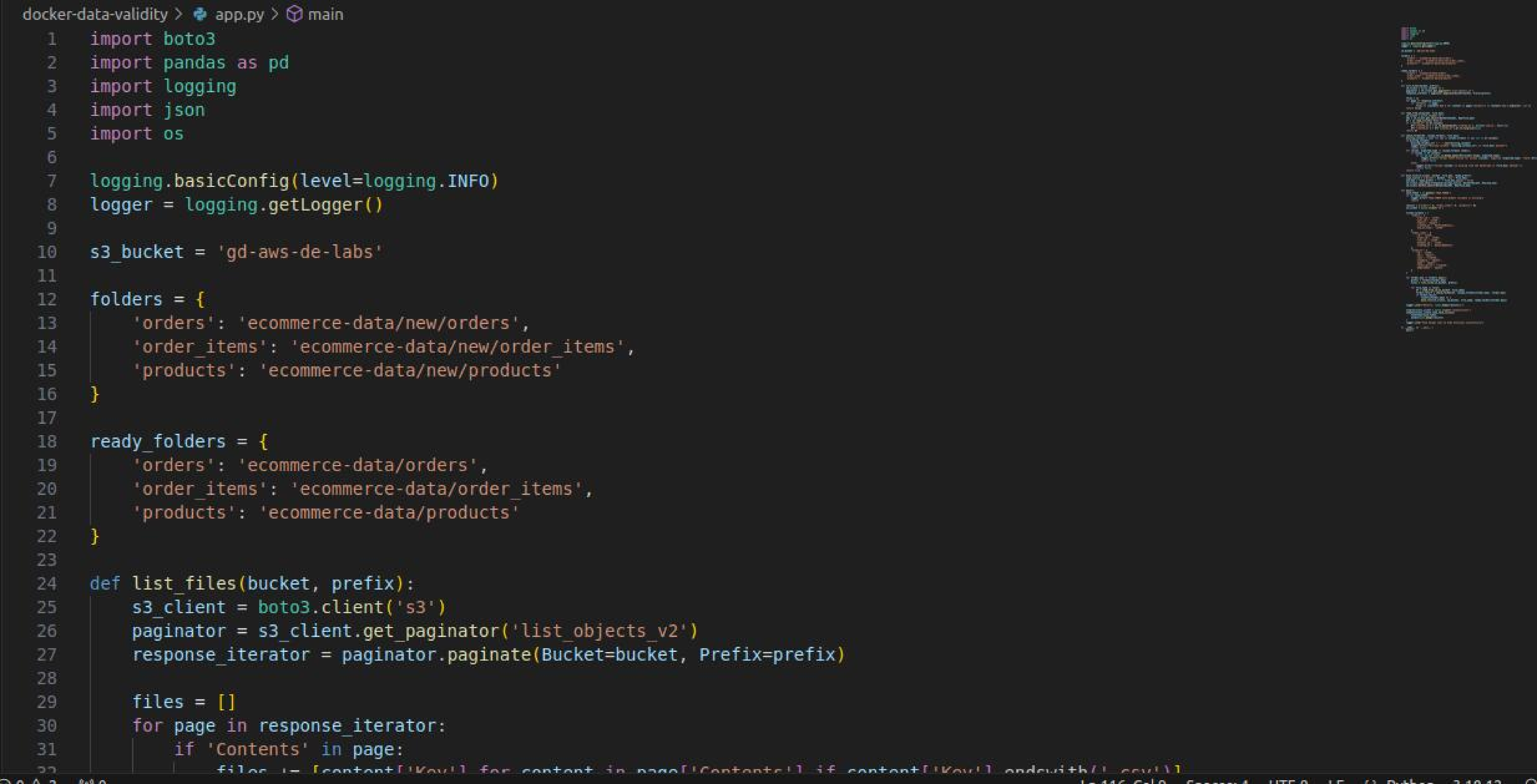
```
    "States": {
      "DataValidation": {
        "Parameters": {
          "Overrides": {
            ]
          },
          "NetworkConfiguration": {
            "AwsvpcConfiguration": {
              "Subnets": [
                "subnet-09fd1652b231f5ca0",
                "subnet-0cdf12a5d9170a6ff"
              ],
              "AssignPublicIp": "ENABLED"
            }
          }
        },
        "Next": "CheckCondition",
        "ResultPath": "$.taskOutput"
      },
      "CheckCondition": {
        "Type": "Choice",
        "Choices": [
          {
            "And": [
              {
                "Variable": "$.taskOutput.orders",
                "NumericEquals": 1
              },
              {
                "Variable": "$.taskOutput.order_items",
                "NumericEquals": 1
              },
```


step-functions > {} step-function.json > {} States > {} DataValidation > {} Parameters > {} Overrides > [ ] ContainerOverrides > {} 0

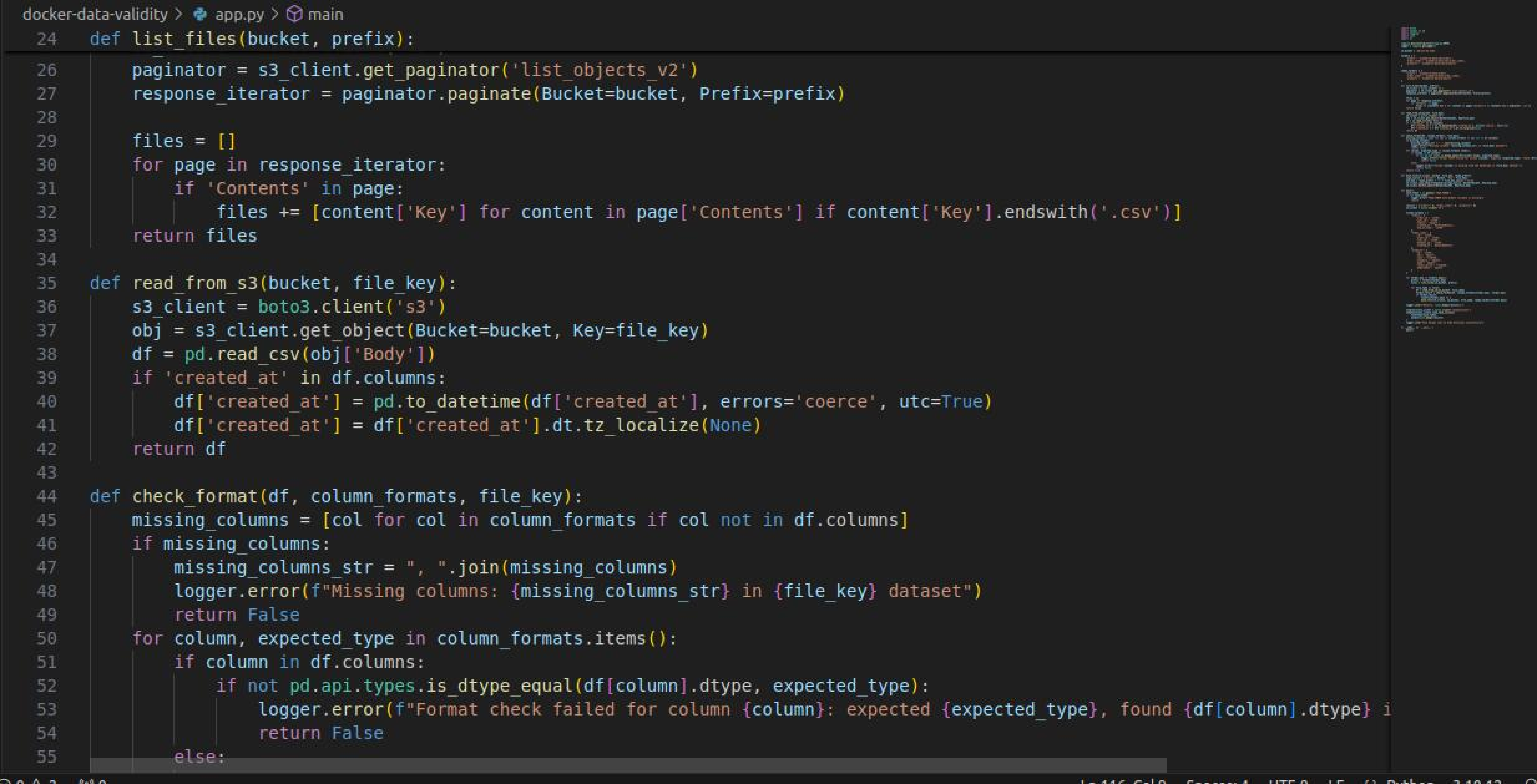
```
    "States": {
      "CheckCondition": {
        "Choices": [
          {
            "And": [
              {
                "Variable": "$.taskOutput.orders",
                "NumericEquals": 1
              },
              {
                "Variable": "$.taskOutput.order_items",
                "NumericEquals": 1
              },
              {
                "Variable": "$.taskOutput.products",
                "NumericEquals": 1
              }
            ],
            "Next": "InvokeSecondTask"
          }
        ],
        "Default": "EndState"
      },
      "InvokeSecondTask": {
        "Type": "Task",
        "Resource": "arn:aws:states:::ecs:runTask.sync",
        "Parameters": {
          "Cluster": "ecommerce-data-ingestion-cluster",
          "TaskDefinition": "arn:aws:ecs:us-east-1:418272783111:task-definition/task-etl-calculations:3",
          "LaunchType": "FARGATE",
          "NetworkConfiguration": {
```


step-functions > {} step-function.json > {} States > {} DataValidation > {} Parameters > {} Overrides > [ ] ContainerOverrides > {} 0

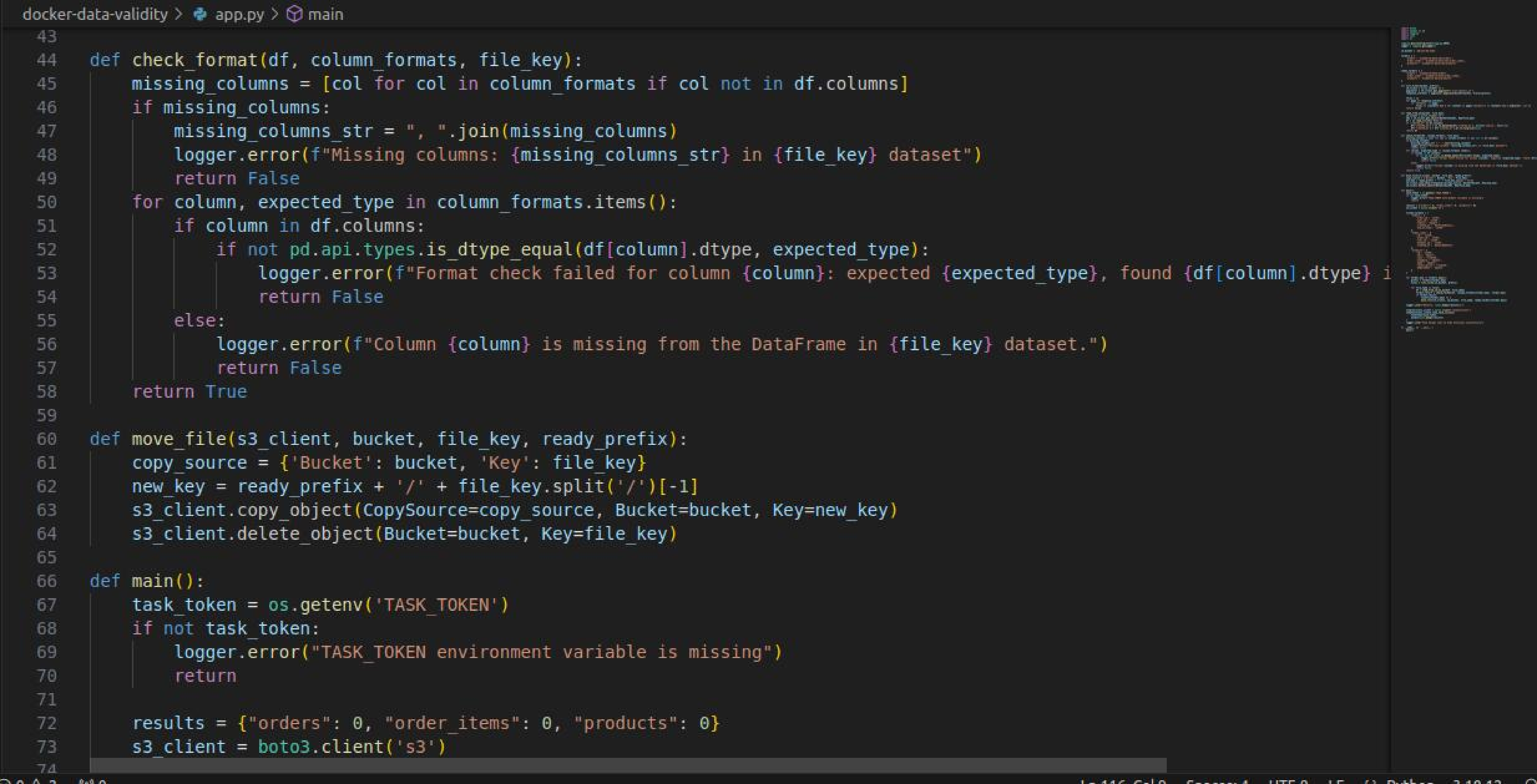
```
  "States": {
    "CheckCondition": {
      ],
      "Default": "EndState"
    },
    "InvokeSecondTask": {
      "Type": "Task",
      "Resource": "arn:aws:states:::ecs:runTask.sync",
      "Parameters": {
        "Cluster": "ecommerce-data-ingestion-cluster",
        "TaskDefinition": "arn:aws:ecs:us-east-1:418272783111:task-definition/task-etl-calculations:3",
        "LaunchType": "FARGATE",
        "NetworkConfiguration": {
          "AwsvpcConfiguration": {
            "Subnets": [
              "subnet-09fd1652b231f5ca0",
              "subnet-0cdf12a5d9170a6ff"
            ],
            "AssignPublicIp": "ENABLED"
          }
        }
      },
      "End": true
    },
    "EndState": {
      "Type": "Pass",
      "End": true
    }
  }
}
```



```python
import boto3
import pandas as pd
import logging
import json
import os

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger()

s3_bucket = 'gd-aws-de-labs'

folders = {
    'orders': 'ecommerce-data/new/orders',
    'order_items': 'ecommerce-data/new/order_items',
    'products': 'ecommerce-data/new/products'
}

ready_folders = {
    'orders': 'ecommerce-data/orders',
    'order_items': 'ecommerce-data/order_items',
    'products': 'ecommerce-data/products'
}

def list_files(bucket, prefix):
    s3_client = boto3.client('s3')
    paginator = s3_client.get_paginator('list_objects_v2')
    response_iterator = paginator.paginate(Bucket=bucket, Prefix=prefix)

    files = []
    for page in response_iterator:
        if 'Contents' in page:
            files += [content['Key'] for content in page['Contents'] if content['Key'].endswith('.csv')]
```

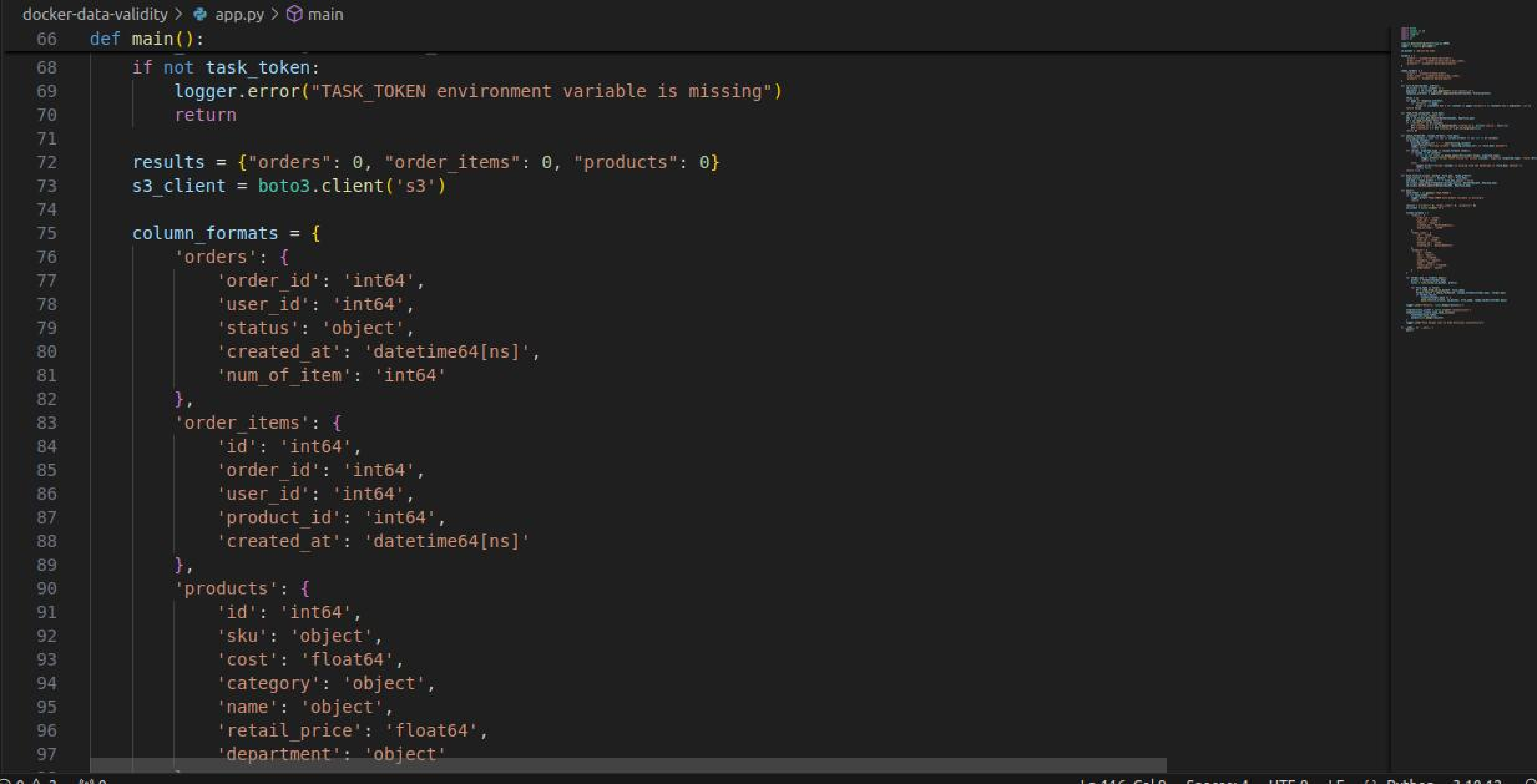


```python
def list_files(bucket, prefix):
    paginator = s3_client.get_paginator('list_objects_v2')
    response_iterator = paginator.paginate(Bucket=bucket, Prefix=prefix)

    files = []
    for page in response_iterator:
        if 'Contents' in page:
            files += [content['Key'] for content in page['Contents'] if content['Key'].endswith('.csv')]
    return files

def read_from_s3(bucket, file_key):
    s3_client = boto3.client('s3')
    obj = s3_client.get_object(Bucket=bucket, Key=file_key)
    df = pd.read_csv(obj['Body'])
    if 'created_at' in df.columns:
        df['created_at'] = pd.to_datetime(df['created_at'], errors='coerce', utc=True)
        df['created_at'] = df['created_at'].dt.tz_localize(None)
    return df

def check_format(df, column_formats, file_key):
    missing_columns = [col for col in column_formats if col not in df.columns]
    if missing_columns:
        missing_columns_str = ", ".join(missing_columns)
        logger.error(f"Missing columns: {missing_columns_str} in {file_key} dataset")
        return False
    for column, expected_type in column_formats.items():
        if column in df.columns:
            if not pd.api.types.is_dtype_equal(df[column].dtype, expected_type):
                logger.error(f"Format check failed for column {column}: expected {expected_type}, found {df[column].dtype} i
                return False
        else:
```

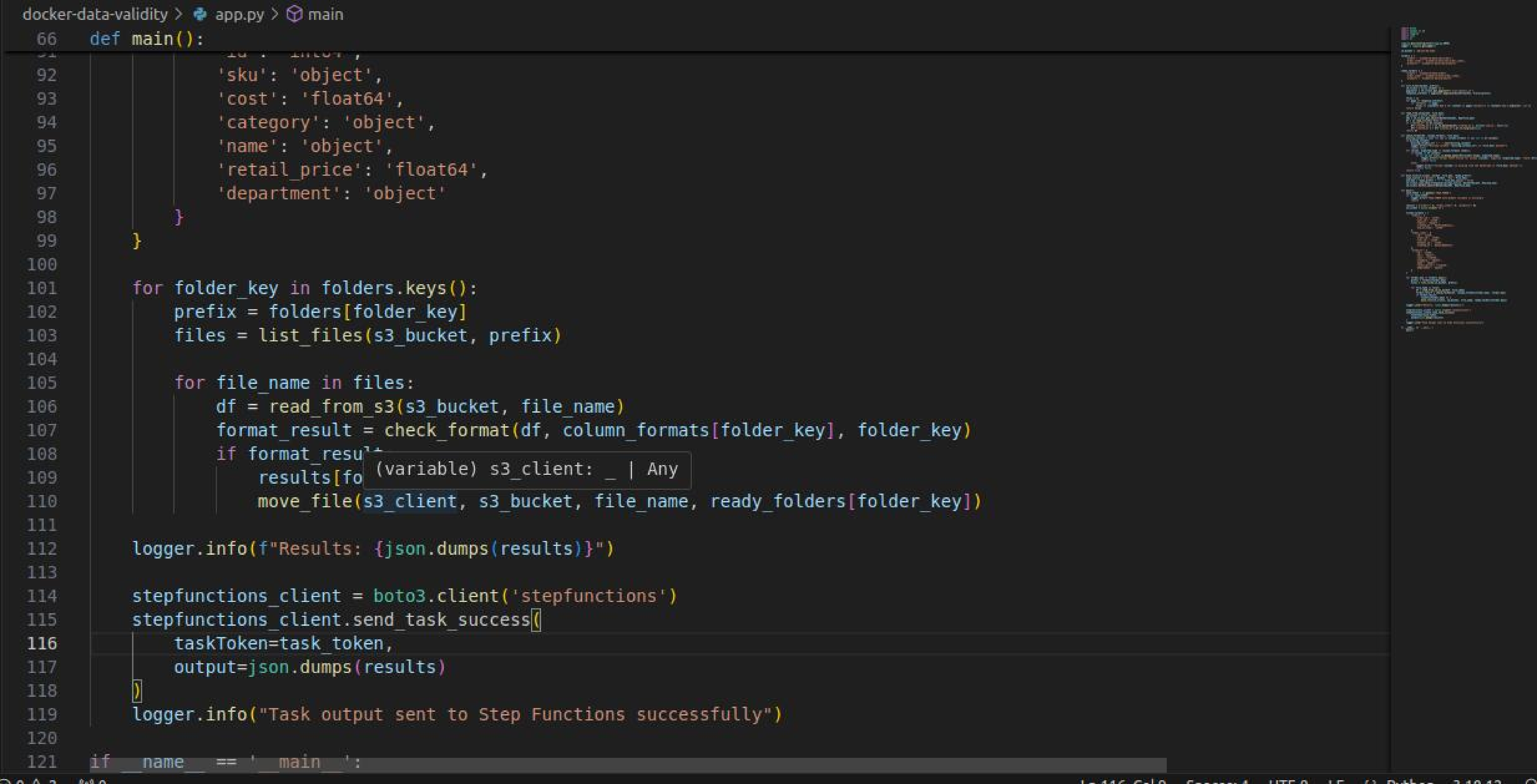


docker-data-validity > app.py > main

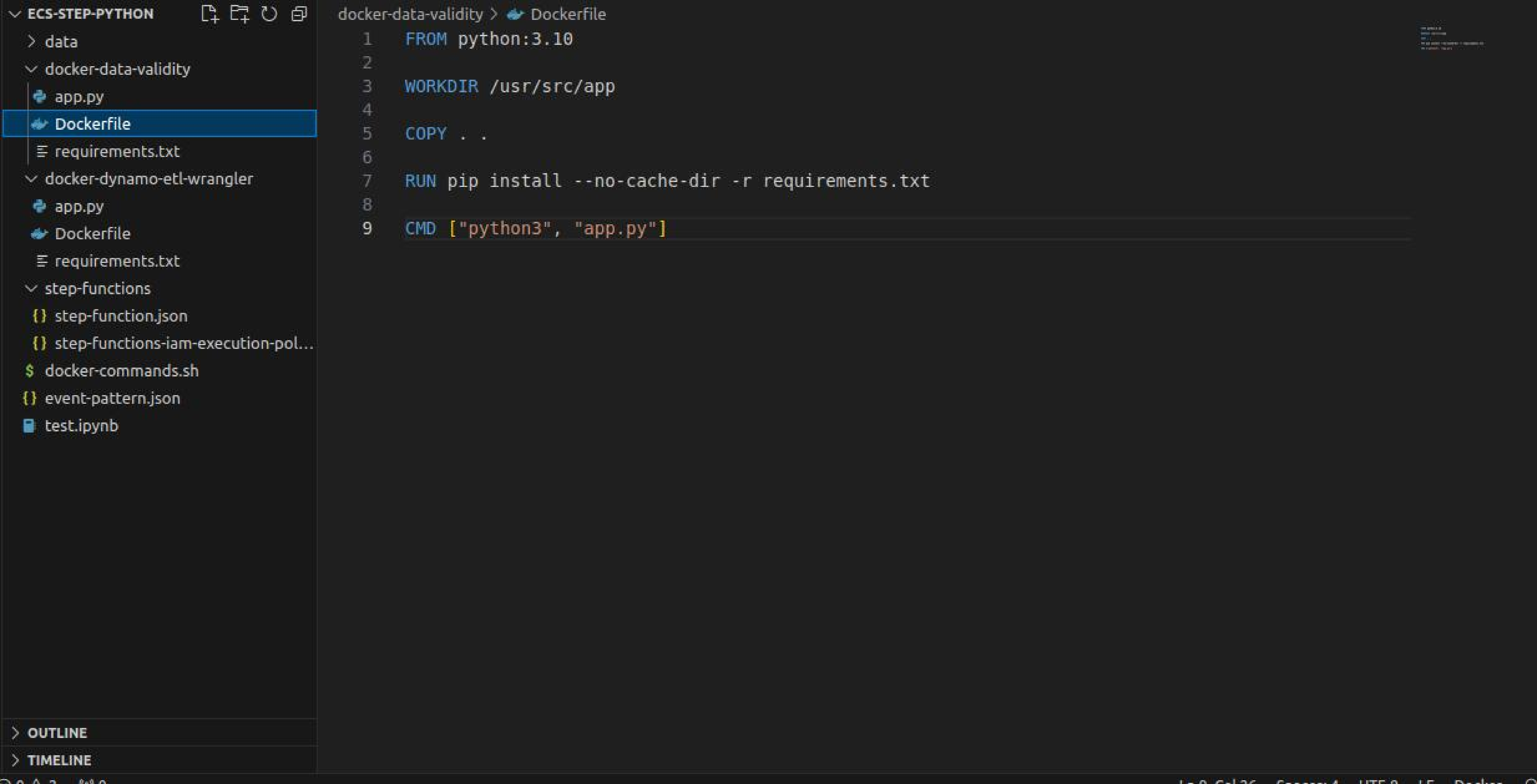
```python

def check_format(df, column_formats, file_key):
    missing_columns = [col for col in column_formats if col not in df.columns]
    if missing_columns:
        missing_columns_str = ", ".join(missing_columns)
        logger.error(f"Missing columns: {missing_columns_str} in {file_key} dataset")
        return False
    for column, expected_type in column_formats.items():
        if column in df.columns:
            if not pd.api.types.is_dtype_equal(df[column].dtype, expected_type):
                logger.error(f"Format check failed for column {column}: expected {expected_type}, found {df[column].dtype} i
                return False
        else:
            logger.error(f"Column {column} is missing from the DataFrame in {file_key} dataset.")
            return False
    return True

def move_file(s3_client, bucket, file_key, ready_prefix):
    copy_source = {'Bucket': bucket, 'Key': file_key}
    new_key = ready_prefix + '/' + file_key.split('/')[-1]
    s3_client.copy_object(CopySource=copy_source, Bucket=bucket, Key=new_key)
    s3_client.delete_object(Bucket=bucket, Key=file_key)

def main():
    task_token = os.getenv('TASK_TOKEN')
    if not task_token:
        logger.error("TASK_TOKEN environment variable is missing")
        return

    results = {"orders": 0, "order_items": 0, "products": 0}
    s3_client = boto3.client('s3')
```



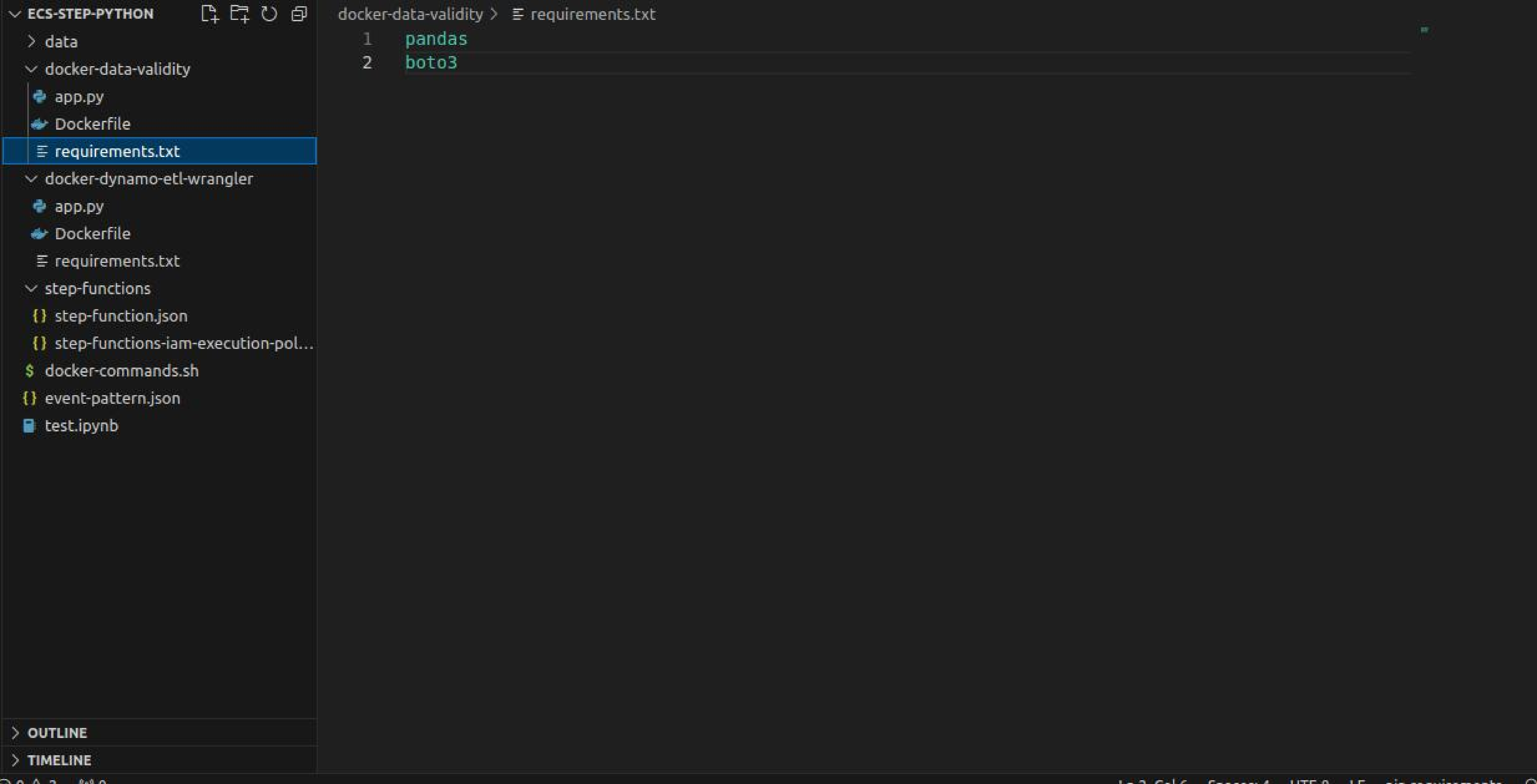
```python
def main():
    if not task_token:
        logger.error("TASK_TOKEN environment variable is missing")
        return

    results = {"orders": 0, "order_items": 0, "products": 0}
    s3_client = boto3.client('s3')

    column_formats = {
        'orders': {
            'order_id': 'int64',
            'user_id': 'int64',
            'status': 'object',
            'created_at': 'datetime64[ns]',
            'num_of_item': 'int64'
        },
        'order_items': {
            'id': 'int64',
            'order_id': 'int64',
            'user_id': 'int64',
            'product_id': 'int64',
            'created_at': 'datetime64[ns]'
        },
        'products': {
            'id': 'int64',
            'sku': 'object',
            'cost': 'float64',
            'category': 'object',
            'name': 'object',
            'retail_price': 'float64',
            'department': 'object'
```



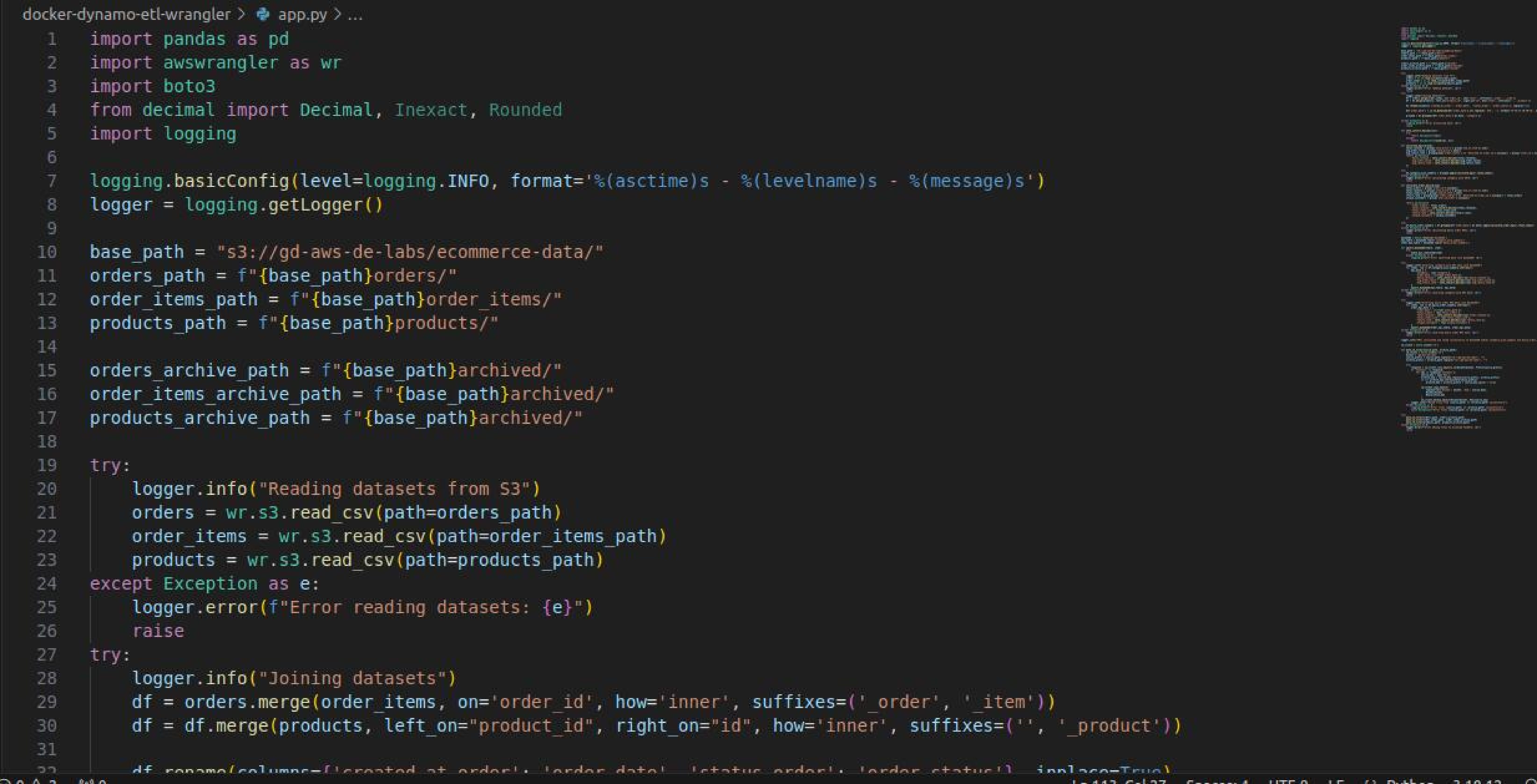
```python
def main():
            'sku': 'object',
            'cost': 'float64',
            'category': 'object',
            'name': 'object',
            'retail_price': 'float64',
            'department': 'object'
        }
    }

    for folder_key in folders.keys():
        prefix = folders[folder_key]
        files = list_files(s3_bucket, prefix)

        for file_name in files:
            df = read_from_s3(s3_bucket, file_name)
            format_result = check_format(df, column_formats[folder_key], folder_key)
            if format_resul
                results[fo
                move_file(s3_client, s3_bucket, file_name, ready_folders[folder_key])

    logger.info(f"Results: {json.dumps(results)}")

    stepfunctions_client = boto3.client('stepfunctions')
    stepfunctions_client.send_task_success(
        taskToken=task_token,
        output=json.dumps(results)
    )
    logger.info("Task output sent to Step Functions successfully")

if __name__ == '__main__':
```



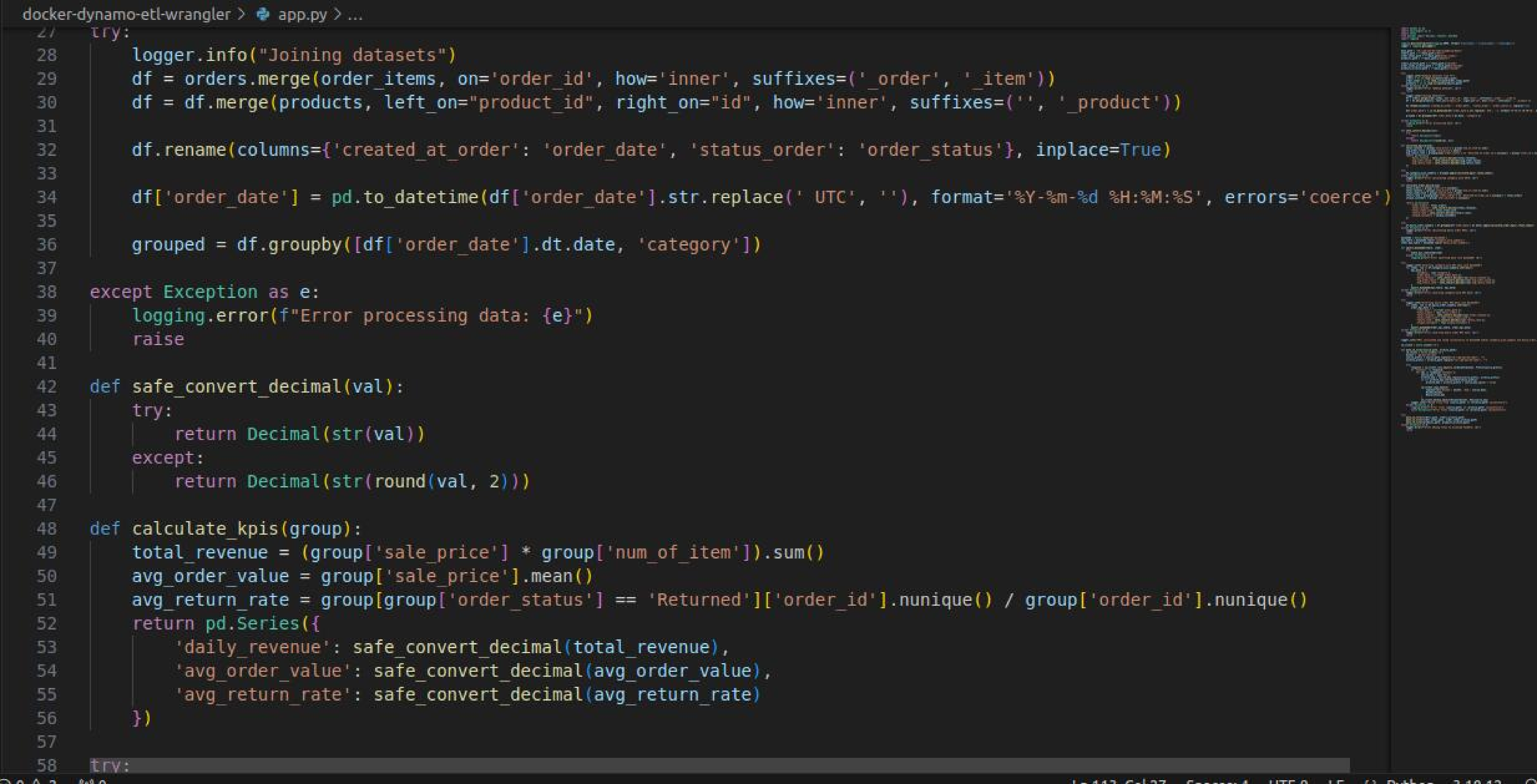
```
FROM python:3.10

WORKDIR /usr/src/app

COPY . .

RUN pip install --no-cache-dir -r requirements.txt

CMD ["python3", "app.py"]
```



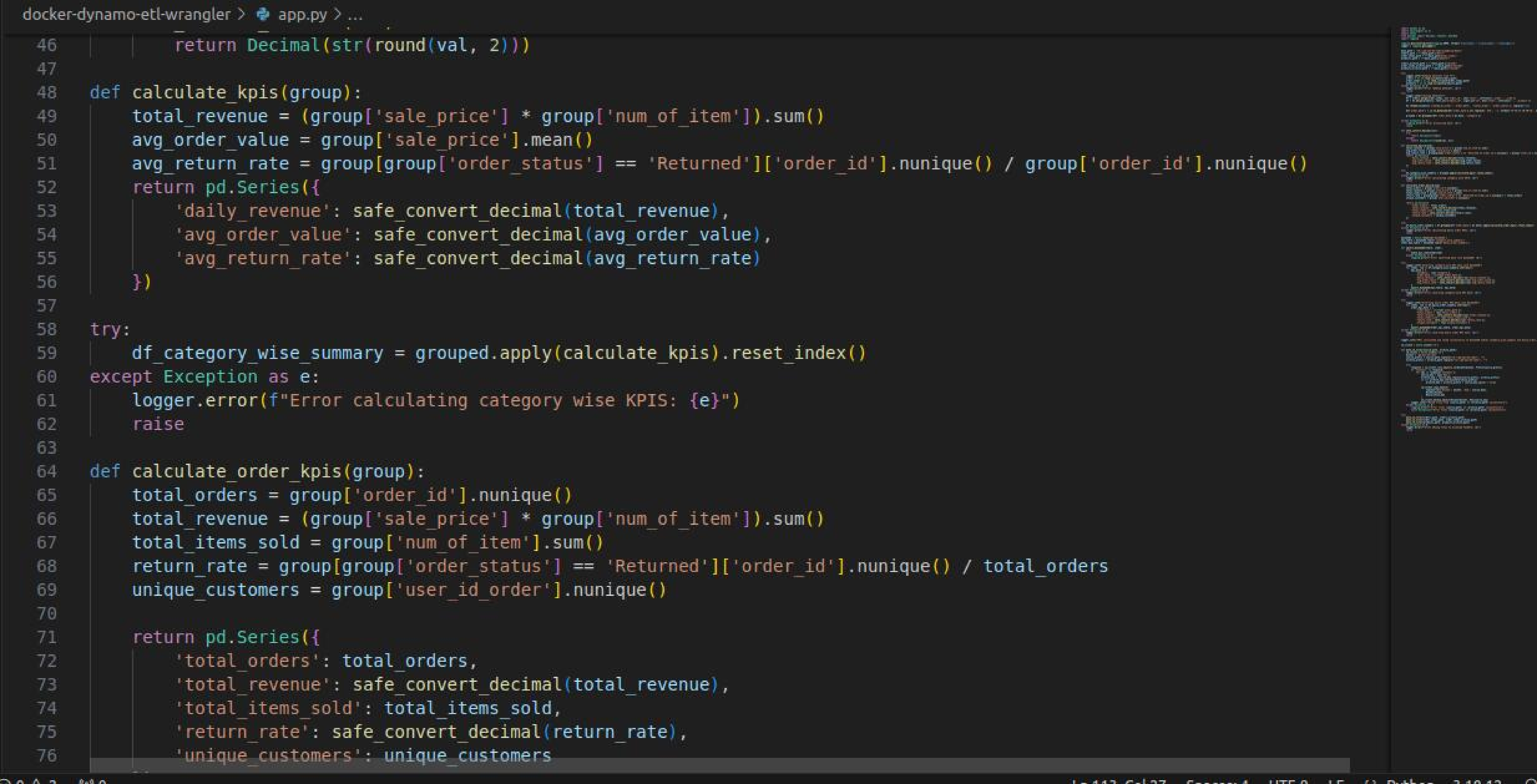
```
pandas
boto3
```



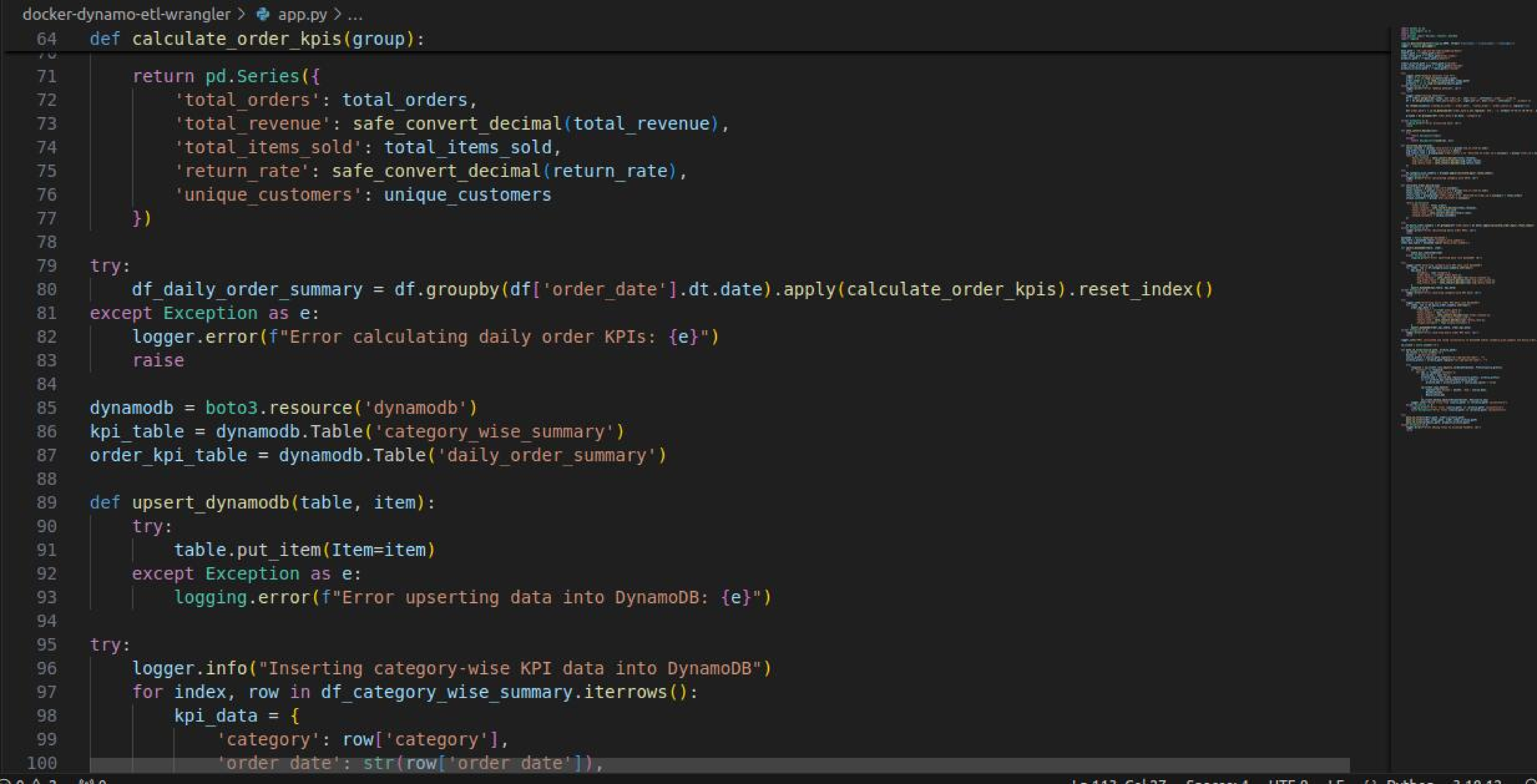
```python
import pandas as pd
import awswrangler as wr
import boto3
from decimal import Decimal, Inexact, Rounded
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger()

base_path = "s3://gd-aws-de-labs/ecommerce-data/"
orders_path = f"{base_path}orders/"
order_items_path = f"{base_path}order_items/"
products_path = f"{base_path}products/"

orders_archive_path = f"{base_path}archived/"
order_items_archive_path = f"{base_path}archived/"
products_archive_path = f"{base_path}archived/"

try:
    logger.info("Reading datasets from S3")
    orders = wr.s3.read_csv(path=orders_path)
    order_items = wr.s3.read_csv(path=order_items_path)
    products = wr.s3.read_csv(path=products_path)
except Exception as e:
    logger.error(f"Error reading datasets: {e}")
    raise
try:
    logger.info("Joining datasets")
    df = orders.merge(order_items, on='order_id', how='inner', suffixes=('_order', '_item'))
    df = df.merge(products, left_on="product_id", right_on="id", how='inner', suffixes=('', '_product'))

    df.rename(columns={'created_at_order': 'order_date', 'status_order': 'order_status'}, inplace=True)
```



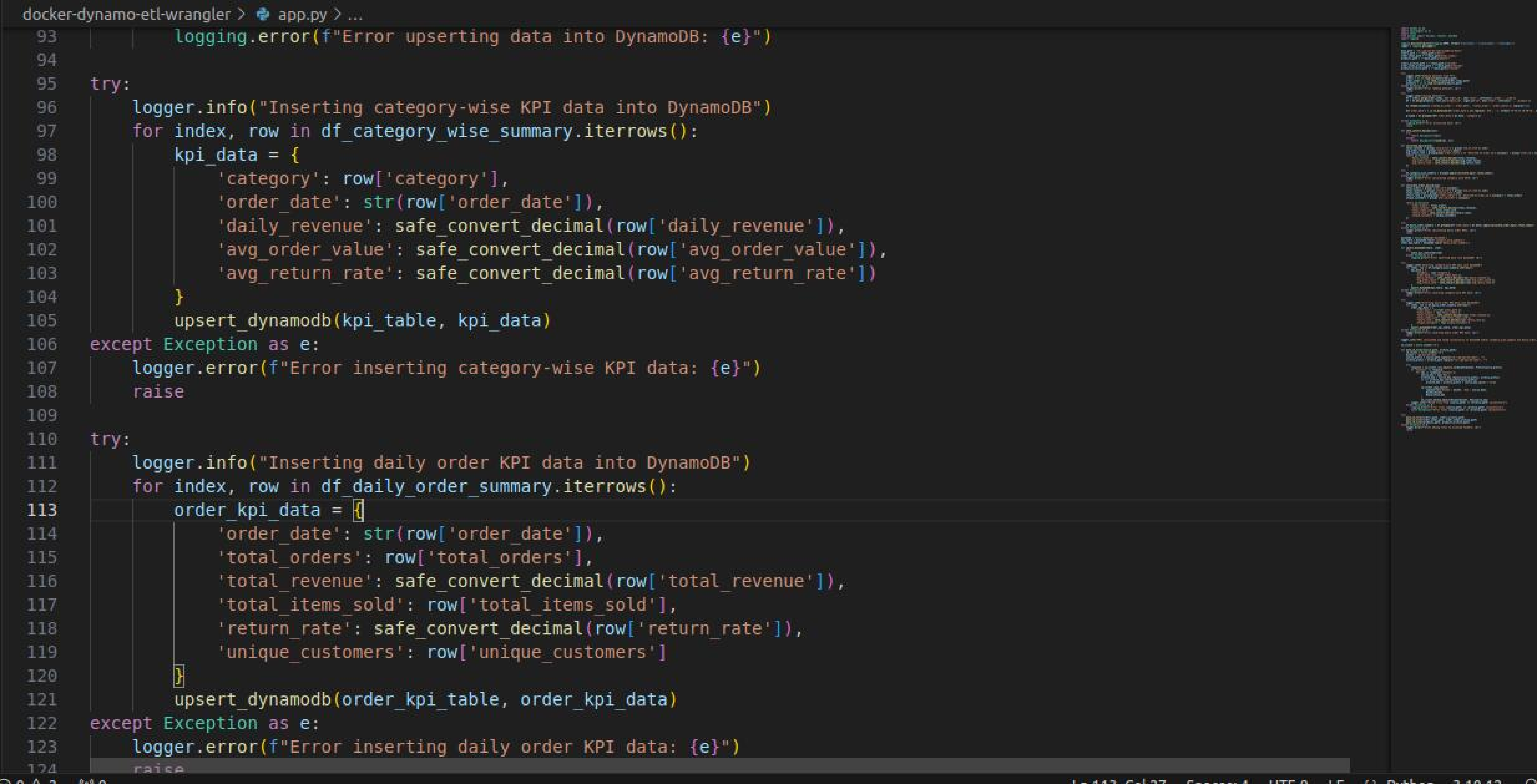
```python
try:
    logger.info("Joining datasets")
    df = orders.merge(order_items, on='order_id', how='inner', suffixes=('_order', '_item'))
    df = df.merge(products, left_on="product_id", right_on="id", how='inner', suffixes=('', '_product'))

    df.rename(columns={'created_at_order': 'order_date', 'status_order': 'order_status'}, inplace=True)

    df['order_date'] = pd.to_datetime(df['order_date'].str.replace(' UTC', ''), format='%Y-%m-%d %H:%M:%S', errors='coerce')

    grouped = df.groupby([df['order_date'].dt.date, 'category'])

except Exception as e:
    logging.error(f"Error processing data: {e}")
    raise

def safe_convert_decimal(val):
    try:
        return Decimal(str(val))
    except:
        return Decimal(str(round(val, 2)))

def calculate_kpis(group):
    total_revenue = (group['sale_price'] * group['num_of_item']).sum()
    avg_order_value = group['sale_price'].mean()
    avg_return_rate = group[group['order_status'] == 'Returned']['order_id'].nunique() / group['order_id'].nunique()
    return pd.Series({
        'daily_revenue': safe_convert_decimal(total_revenue),
        'avg_order_value': safe_convert_decimal(avg_order_value),
        'avg_return_rate': safe_convert_decimal(avg_return_rate)
    })

try:
```



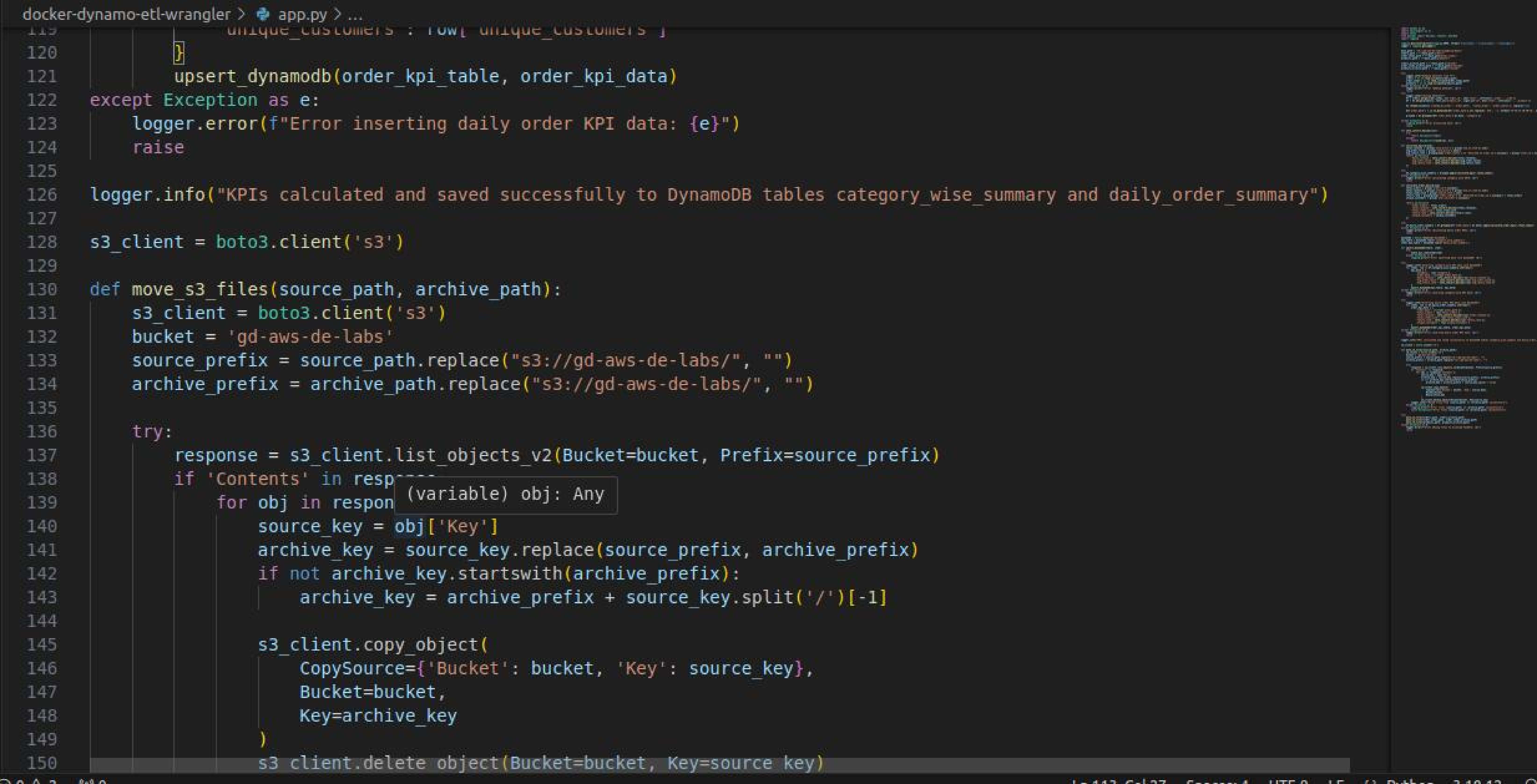
```python
        return Decimal(str(round(val, 2)))

def calculate_kpis(group):
    total_revenue = (group['sale_price'] * group['num_of_item']).sum()
    avg_order_value = group['sale_price'].mean()
    avg_return_rate = group[group['order_status'] == 'Returned']['order_id'].nunique() / group['order_id'].nunique()
    return pd.Series({
        'daily_revenue': safe_convert_decimal(total_revenue),
        'avg_order_value': safe_convert_decimal(avg_order_value),
        'avg_return_rate': safe_convert_decimal(avg_return_rate)
    })

try:
    df_category_wise_summary = grouped.apply(calculate_kpis).reset_index()
except Exception as e:
    logger.error(f"Error calculating category wise KPIS: {e}")
    raise

def calculate_order_kpis(group):
    total_orders = group['order_id'].nunique()
    total_revenue = (group['sale_price'] * group['num_of_item']).sum()
    total_items_sold = group['num_of_item'].sum()
    return_rate = group[group['order_status'] == 'Returned']['order_id'].nunique() / total_orders
    unique_customers = group['user_id_order'].nunique()

    return pd.Series({
        'total_orders': total_orders,
        'total_revenue': safe_convert_decimal(total_revenue),
        'total_items_sold': total_items_sold,
        'return_rate': safe_convert_decimal(return_rate),
        'unique_customers': unique_customers
```



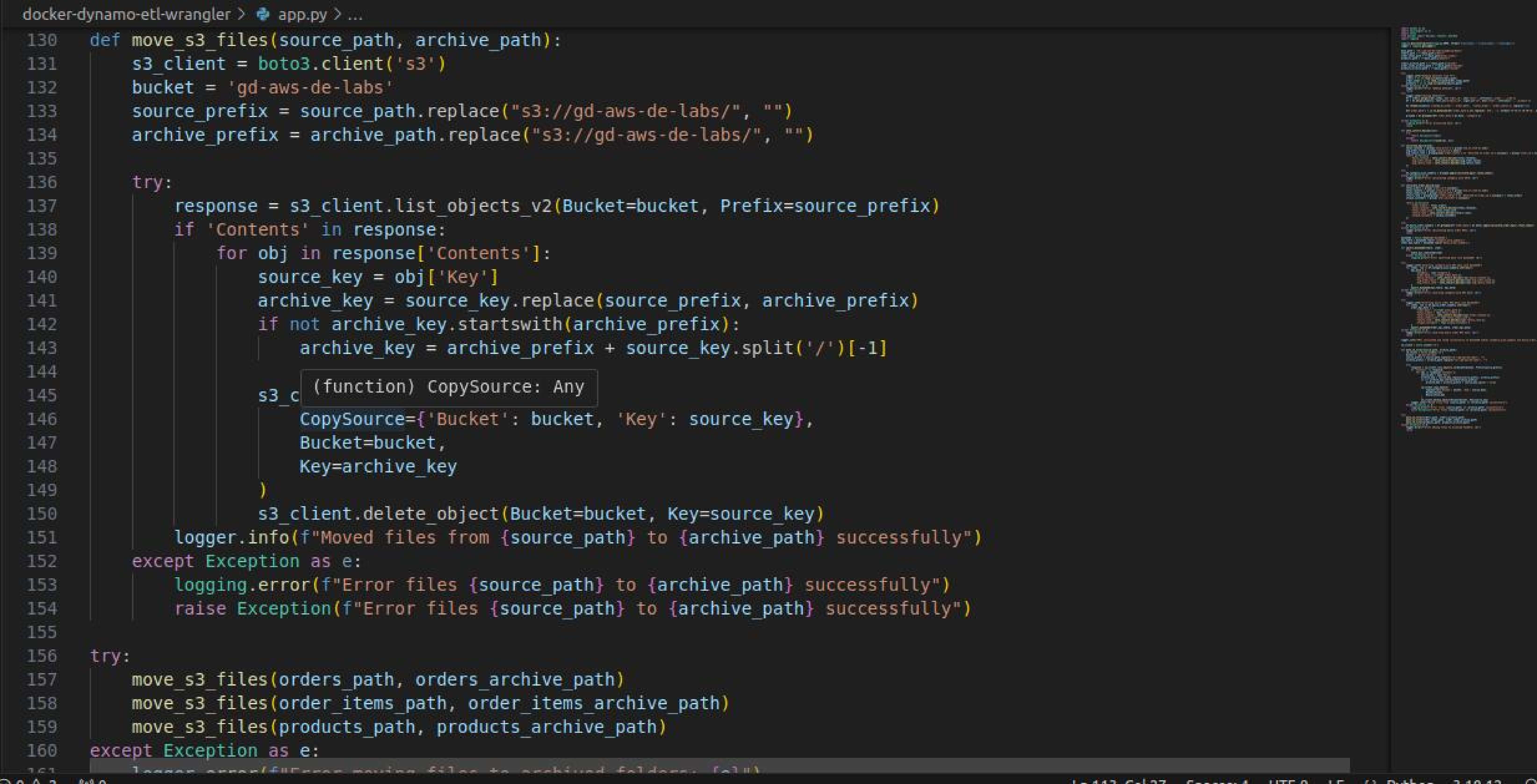
```python
def calculate_order_kpis(group):
    return pd.Series({
        'total_orders': total_orders,
        'total_revenue': safe_convert_decimal(total_revenue),
        'total_items_sold': total_items_sold,
        'return_rate': safe_convert_decimal(return_rate),
        'unique_customers': unique_customers
    })

try:
    df_daily_order_summary = df.groupby(df['order_date'].dt.date).apply(calculate_order_kpis).reset_index()
except Exception as e:
    logger.error(f"Error calculating daily order KPIs: {e}")
    raise

dynamodb = boto3.resource('dynamodb')
kpi_table = dynamodb.Table('category_wise_summary')
order_kpi_table = dynamodb.Table('daily_order_summary')

def upsert_dynamodb(table, item):
    try:
        table.put_item(Item=item)
    except Exception as e:
        logging.error(f"Error upserting data into DynamoDB: {e}")

try:
    logger.info("Inserting category-wise KPI data into DynamoDB")
    for index, row in df_category_wise_summary.iterrows():
        kpi_data = {
            'category': row['category'],
            'order_date': str(row['order_date']),
```



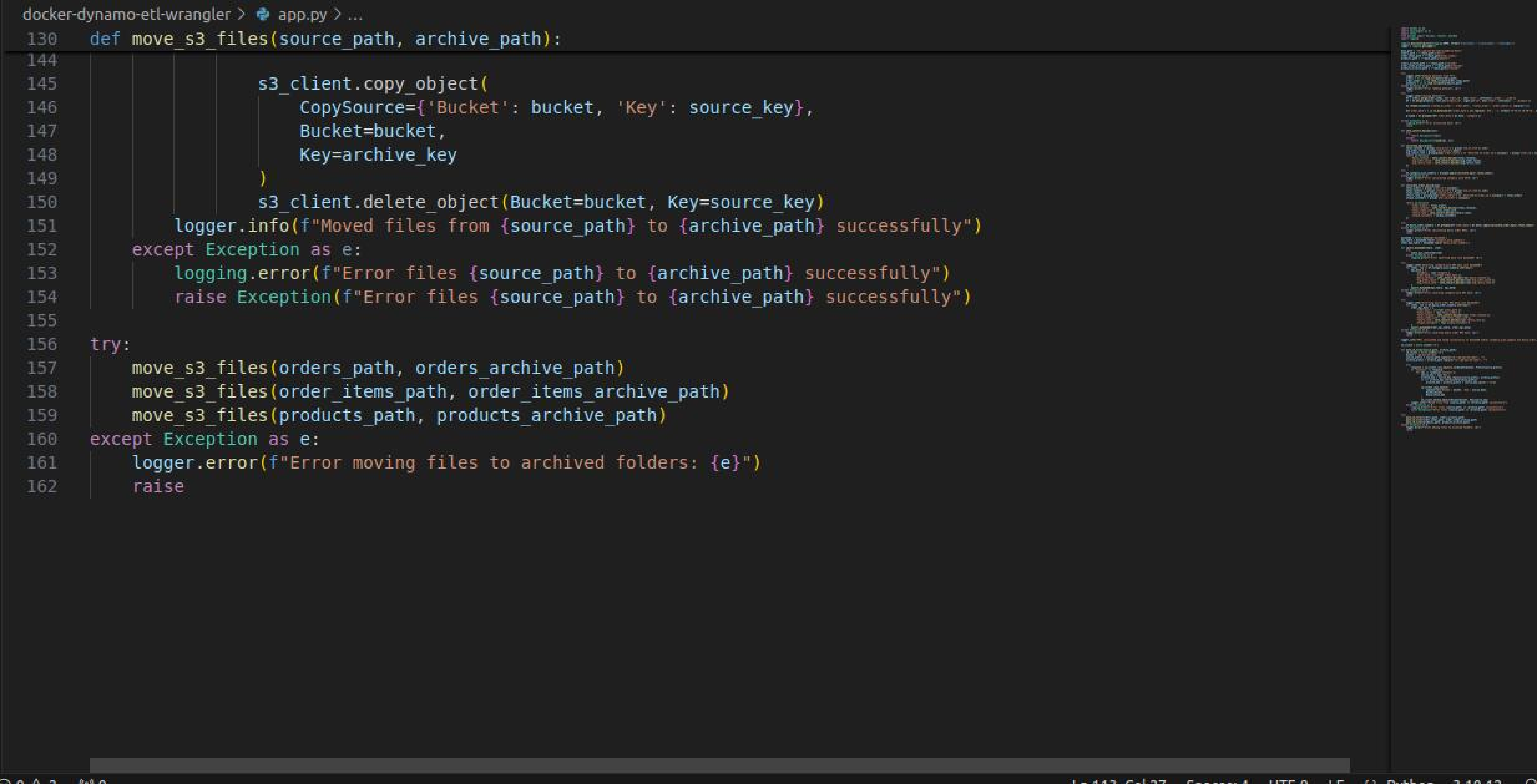
```python
        logging.error(f"Error upserting data into DynamoDB: {e}")

try:
    logger.info("Inserting category-wise KPI data into DynamoDB")
    for index, row in df_category_wise_summary.iterrows():
        kpi_data = {
            'category': row['category'],
            'order_date': str(row['order_date']),
            'daily_revenue': safe_convert_decimal(row['daily_revenue']),
            'avg_order_value': safe_convert_decimal(row['avg_order_value']),
            'avg_return_rate': safe_convert_decimal(row['avg_return_rate'])
        }
        upsert_dynamodb(kpi_table, kpi_data)
except Exception as e:
    logger.error(f"Error inserting category-wise KPI data: {e}")
    raise

try:
    logger.info("Inserting daily order KPI data into DynamoDB")
    for index, row in df_daily_order_summary.iterrows():
        order_kpi_data = {
            'order_date': str(row['order_date']),
            'total_orders': row['total_orders'],
            'total_revenue': safe_convert_decimal(row['total_revenue']),
            'total_items_sold': row['total_items_sold'],
            'return_rate': safe_convert_decimal(row['return_rate']),
            'unique_customers': row['unique_customers']
        }
        upsert_dynamodb(order_kpi_table, order_kpi_data)
except Exception as e:
    logger.error(f"Error inserting daily order KPI data: {e}")
    raise
```



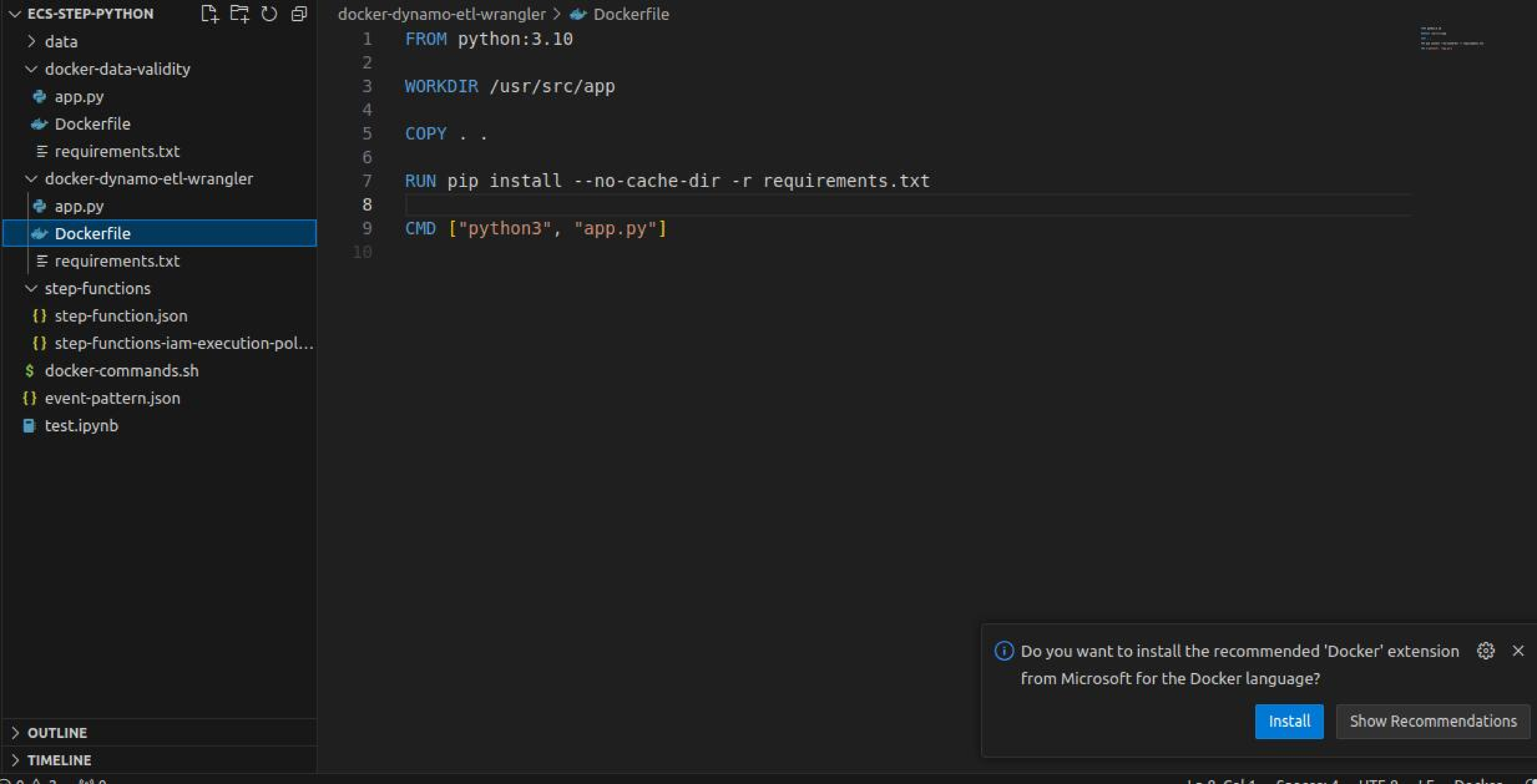
```
            'unique_customers': row['unique_customers']
        }
        upsert_dynamodb(order_kpi_table, order_kpi_data)
except Exception as e:
    logger.error(f"Error inserting daily order KPI data: {e}")
    raise

logger.info("KPIs calculated and saved successfully to DynamoDB tables category_wise_summary and daily_order_summary")

s3_client = boto3.client('s3')

def move_s3_files(source_path, archive_path):
    s3_client = boto3.client('s3')
    bucket = 'gd-aws-de-labs'
    source_prefix = source_path.replace("s3://gd-aws-de-labs/", "")
    archive_prefix = archive_path.replace("s3://gd-aws-de-labs/", "")

    try:
        response = s3_client.list_objects_v2(Bucket=bucket, Prefix=source_prefix)
        if 'Contents' in resp
            for obj in respon
                source_key = obj['Key']
                archive_key = source_key.replace(source_prefix, archive_prefix)
                if not archive_key.startswith(archive_prefix):
                    archive_key = archive_prefix + source_key.split('/')[-1]

                s3_client.copy_object(
                    CopySource={'Bucket': bucket, 'Key': source_key},
                    Bucket=bucket,
                    Key=archive_key
                )
                s3_client.delete_object(Bucket=bucket, Key=source_key)
```


docker-dynamo-etl-wrangler > app.py > ...
(function) CopySource: Any

```python
def move_s3_files(source_path, archive_path):
    s3_client = boto3.client('s3')
    bucket = 'gd-aws-de-labs'
    source_prefix = source_path.replace("s3://gd-aws-de-labs/", "")
    archive_prefix = archive_path.replace("s3://gd-aws-de-labs/", "")

    try:
        response = s3_client.list_objects_v2(Bucket=bucket, Prefix=source_prefix)
        if 'Contents' in response:
            for obj in response['Contents']:
                source_key = obj['Key']
                archive_key = source_key.replace(source_prefix, archive_prefix)
                if not archive_key.startswith(archive_prefix):
                    archive_key = archive_prefix + source_key.split('/')[-1]

                s3_c
                    CopySource={'Bucket': bucket, 'Key': source_key},
                    Bucket=bucket,
                    Key=archive_key
                )
                s3_client.delete_object(Bucket=bucket, Key=source_key)
        logger.info(f"Moved files from {source_path} to {archive_path} successfully")
    except Exception as e:
        logging.error(f"Error files {source_path} to {archive_path} successfully")
        raise Exception(f"Error files {source_path} to {archive_path} successfully")

try:
    move_s3_files(orders_path, orders_archive_path)
    move_s3_files(order_items_path, order_items_archive_path)
    move_s3_files(products_path, products_archive_path)
except Exception as e:
    logger.error(f"Error moving files to archived folders: {e}")
```

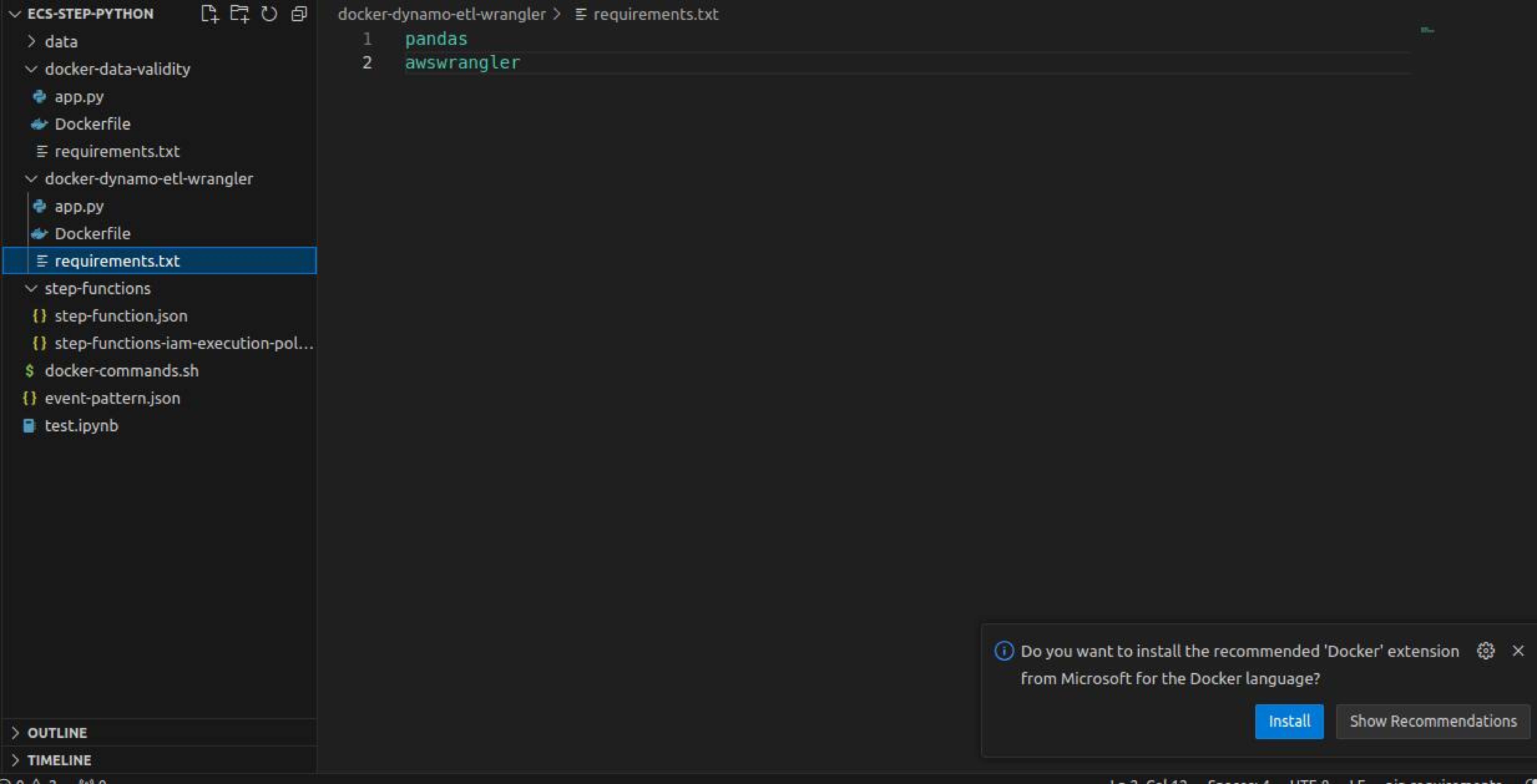


```python
def move_s3_files(source_path, archive_path):

                s3_client.copy_object(
                    CopySource={'Bucket': bucket, 'Key': source_key},
                    Bucket=bucket,
                    Key=archive_key
                )
                s3_client.delete_object(Bucket=bucket, Key=source_key)
        logger.info(f"Moved files from {source_path} to {archive_path} successfully")
    except Exception as e:
        logging.error(f"Error files {source_path} to {archive_path} successfully")
        raise Exception(f"Error files {source_path} to {archive_path} successfully")

try:
    move_s3_files(orders_path, orders_archive_path)
    move_s3_files(order_items_path, order_items_archive_path)
    move_s3_files(products_path, products_archive_path)
except Exception as e:
    logger.error(f"Error moving files to archived folders: {e}")
    raise
```



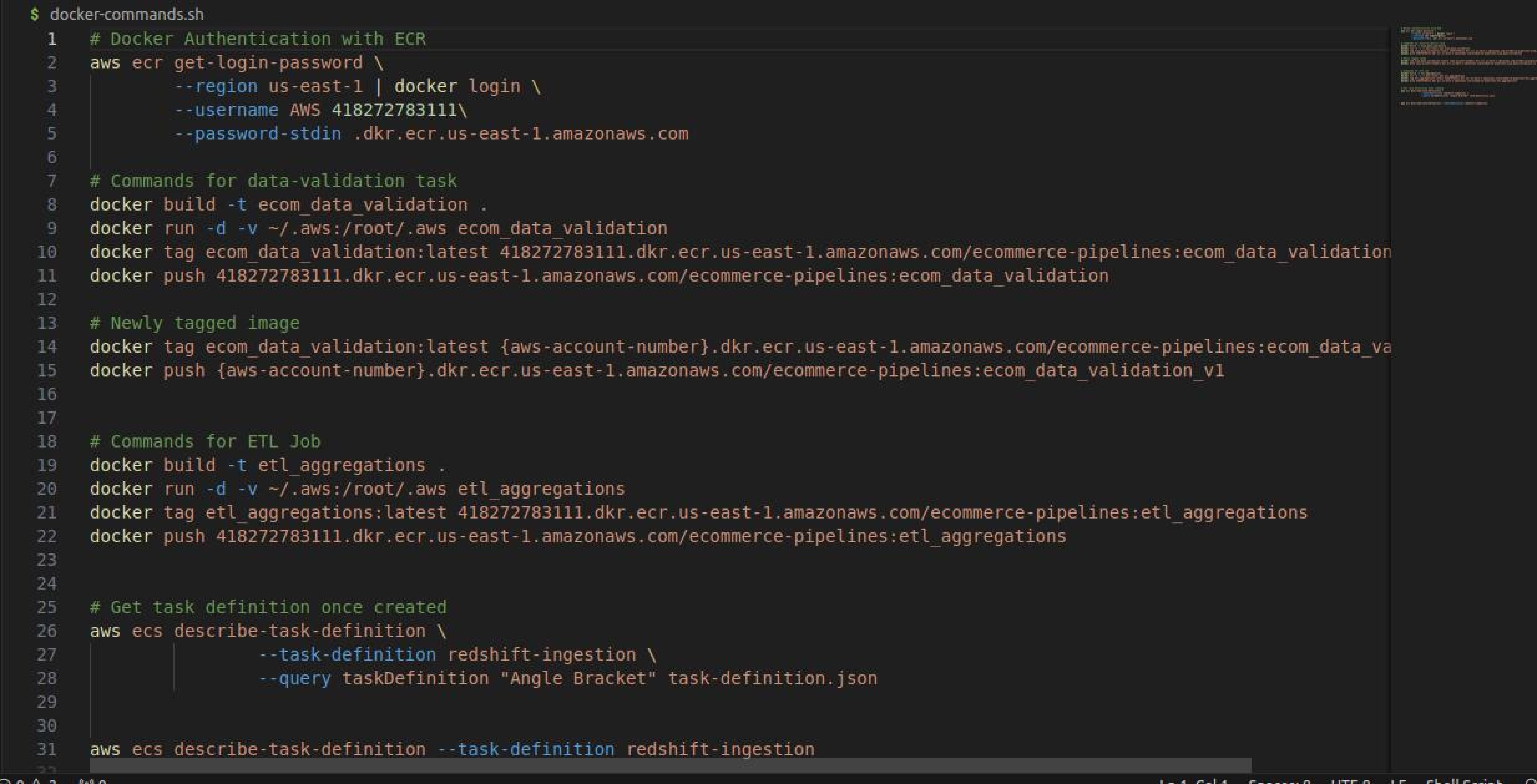
```
FROM python:3.10

WORKDIR /usr/src/app

COPY . .

RUN pip install --no-cache-dir -r requirements.txt

CMD ["python3", "app.py"]
```

Do you want to install the recommended 'Docker' extension from Microsoft for the Docker language?



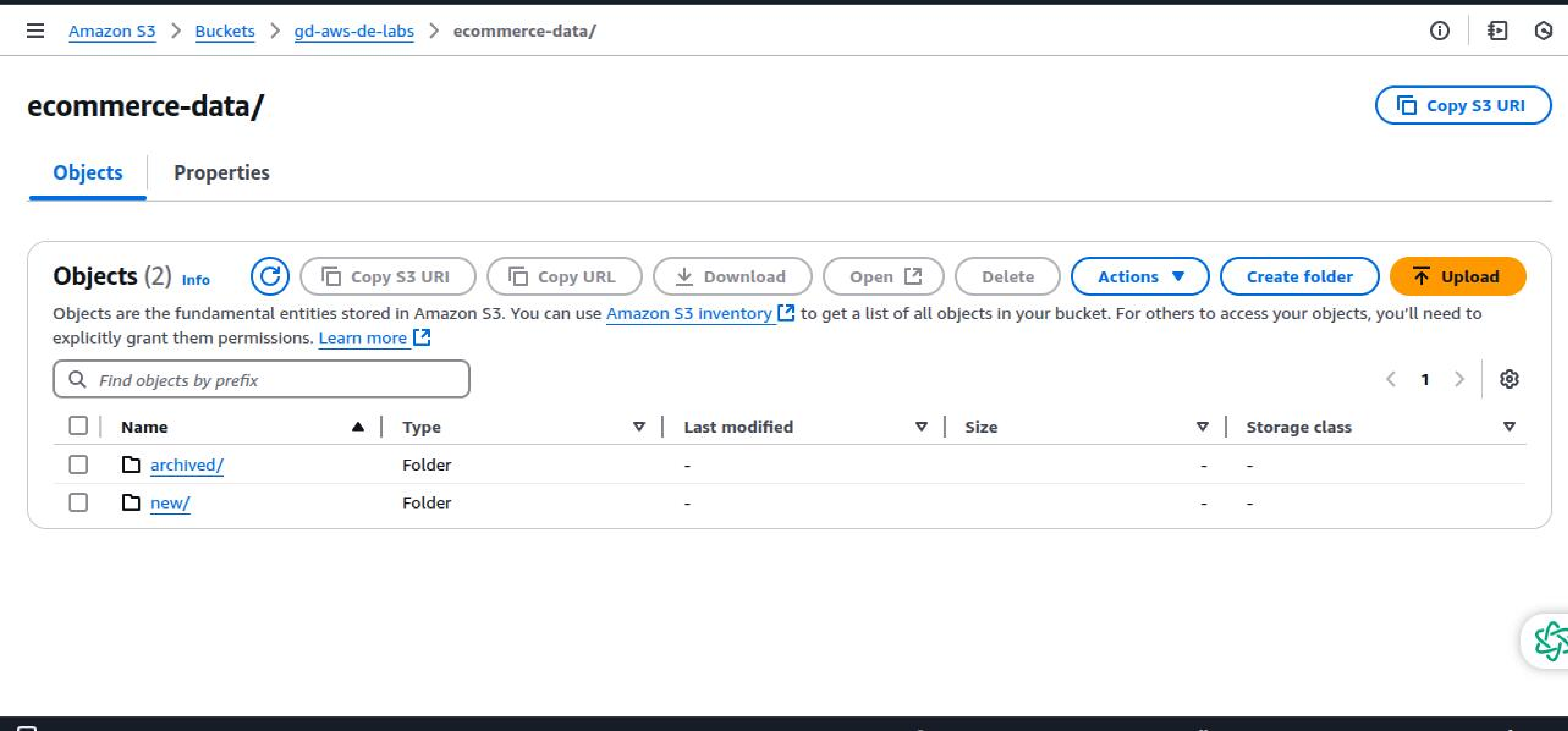
```
pandas
awswrangler
```

Do you want to install the recommended 'Docker' extension from Microsoft for the Docker language?



docker-commands.sh

```
# Docker Authentication with ECR
aws ecr get-login-password \
        --region us-east-1 | docker login \
        --username AWS 418272783111\
        --password-stdin .dkr.ecr.us-east-1.amazonaws.com

# Commands for data-validation task
docker build -t ecom_data_validation .
docker run -d -v ~/.aws:/root/.aws ecom_data_validation
docker tag ecom_data_validation:latest 418272783111.dkr.ecr.us-east-1.amazonaws.com/ecommerce-pipelines:ecom_data_validation
docker push 418272783111.dkr.ecr.us-east-1.amazonaws.com/ecommerce-pipelines:ecom_data_validation

# Newly tagged image
docker tag ecom_data_validation:latest {aws-account-number}.dkr.ecr.us-east-1.amazonaws.com/ecommerce-pipelines:ecom_data_va
docker push {aws-account-number}.dkr.ecr.us-east-1.amazonaws.com/ecommerce-pipelines:ecom_data_validation_v1


# Commands for ETL Job
docker build -t etl_aggregations .
docker run -d -v ~/.aws:/root/.aws etl_aggregations
docker tag etl_aggregations:latest 418272783111.dkr.ecr.us-east-1.amazonaws.com/ecommerce-pipelines:etl_aggregations
docker push 418272783111.dkr.ecr.us-east-1.amazonaws.com/ecommerce-pipelines:etl_aggregations


# Get task definition once created
aws ecs describe-task-definition \
                --task-definition redshift-ingestion \
                --query taskDefinition "Angle Bracket" task-definition.json


aws ecs describe-task-definition --task-definition redshift-ingestion
```



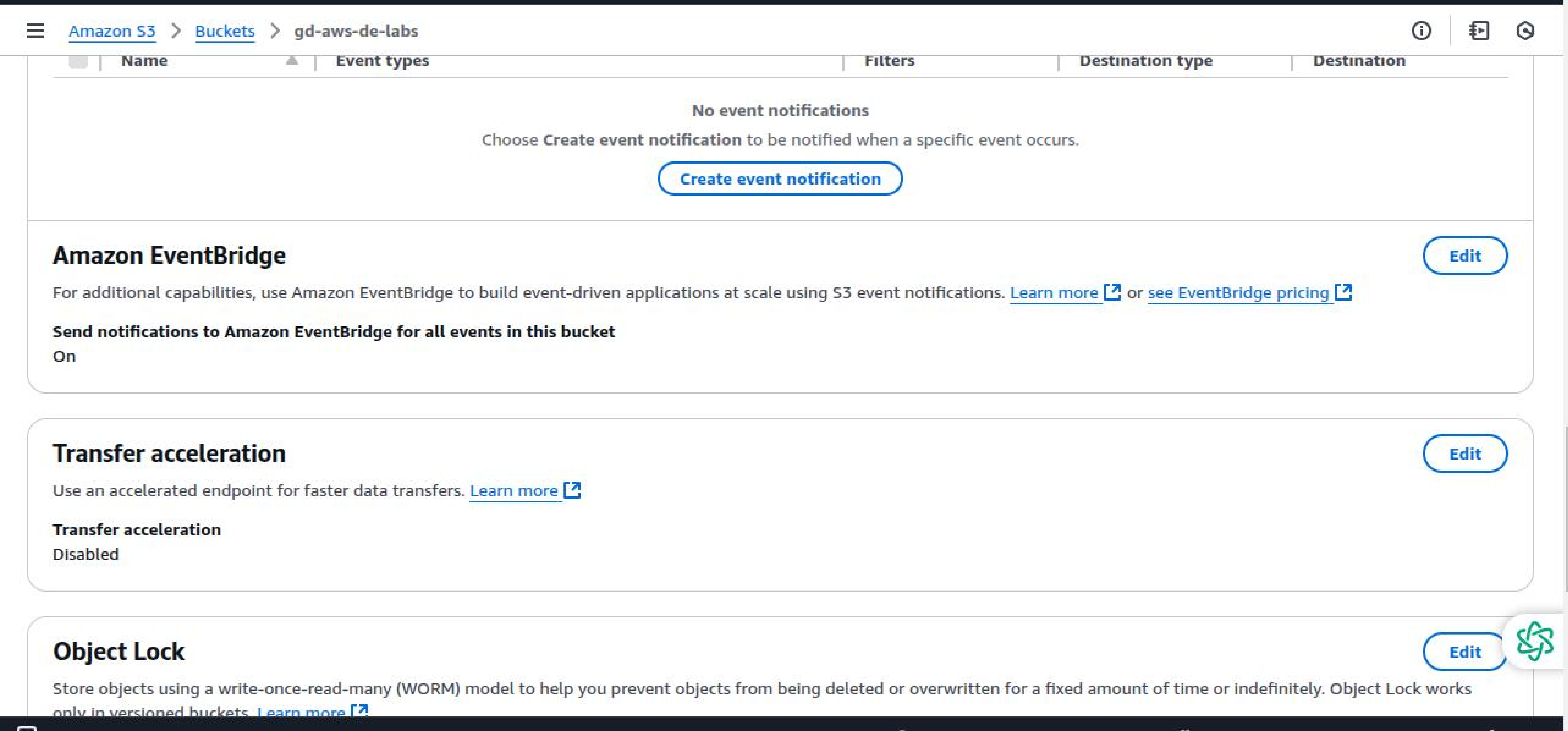
ecommerce-data/

Objects (2) 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

Find objects by prefix

□	Name ▲	Type	Last modified	Size	Storage class
□	archived/	Folder	-	-	-
□	new/	Folder	-	-	-



Amazon S3 > Buckets > gd-aws-de-labs

Name	Event types	Filters	Destination type	Destination

No event notifications

Choose **Create event notification** to be notified when a specific event occurs.

Amazon EventBridge

For additional capabilities, use Amazon EventBridge to build event-driven applications at scale using S3 event notifications. Learn more or see EventBridge pricing

Send notifications to Amazon EventBridge for all events in this bucket

On

Transfer acceleration

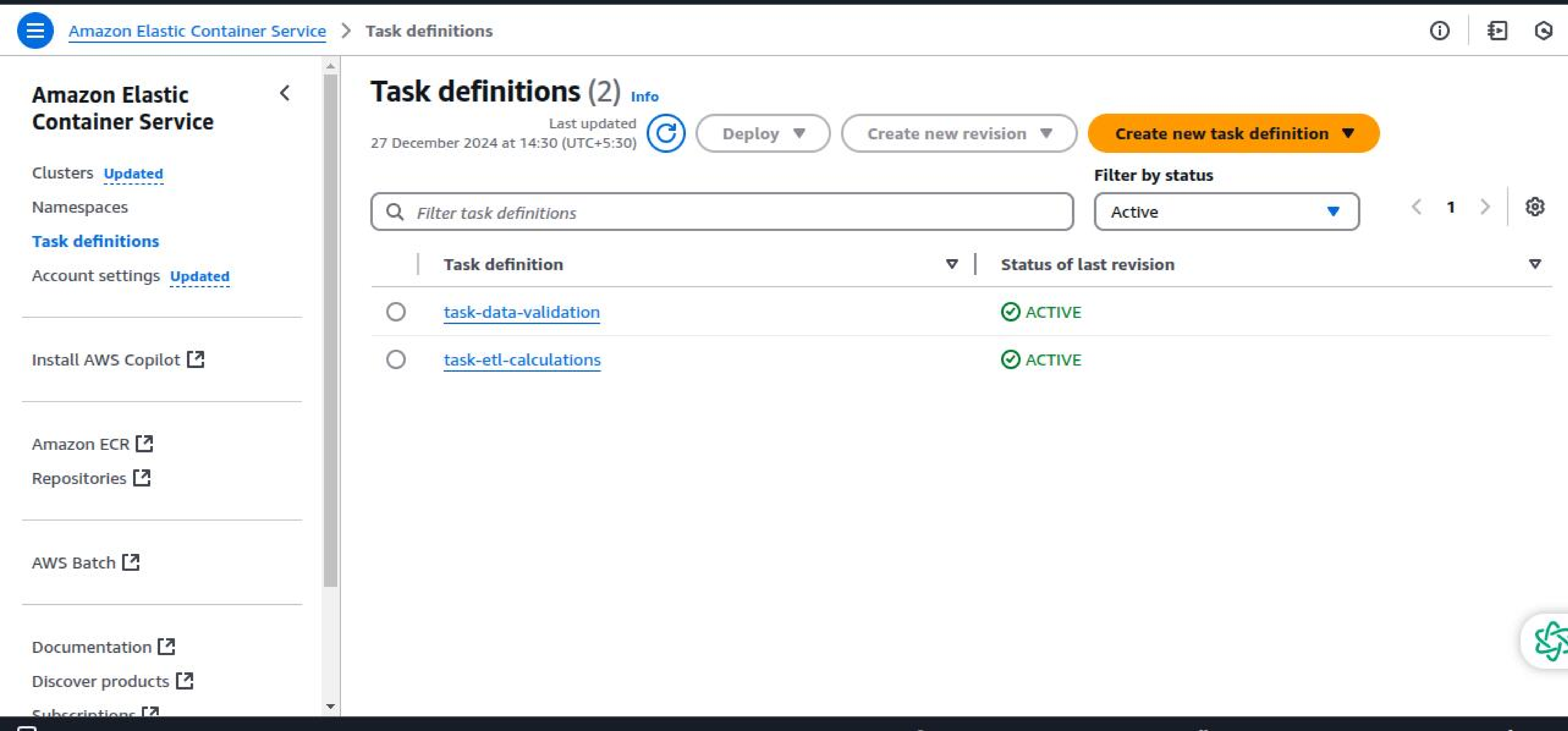
Use an accelerated endpoint for faster data transfers. Learn more

Transfer acceleration

Disabled

Object Lock

Store objects using a write-once-read-many (WORM) model to help you prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. Object Lock works only in versioned buckets. Learn more



Amazon Elastic Container Service > Task definitions

Amazon Elastic Container Service

Clusters Updated

Namespaces

Task definitions

Account settings Updated

Install AWS Copilot

Amazon ECR

Repositories

AWS Batch

Documentation

Discover products

Subscriptions

Task definitions (2) Info

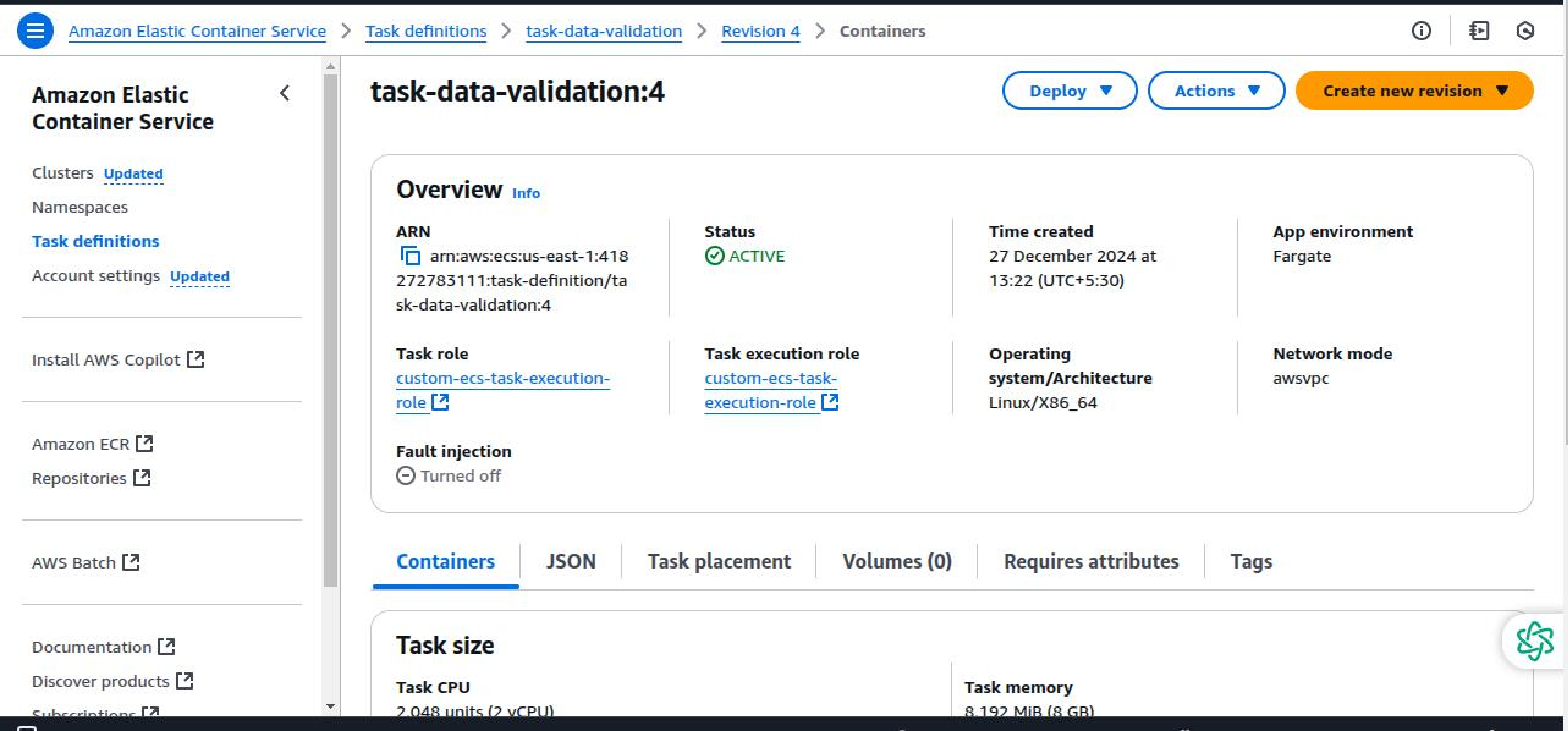
Last updated
27 December 2024 at 14:30 (UTC+5:30)

Deploy ▼ | Create new revision ▼ | Create new task definition ▼

Filter task definitions

Filter by status: Active

	Task definition	Status of last revision
○	task-data-validation	✓ ACTIVE
○	task-etl-calculations	✓ ACTIVE


Amazon Elastic Container Service > Task definitions > task-data-validation > Revision 4 > Containers

task-data-validation:4

Overview Info

ARN
arn:aws:ecs:us-east-1:418272783111:task-definition/task-data-validation:4

Status
✓ ACTIVE

Time created
27 December 2024 at 13:22 (UTC+5:30)

App environment
Fargate

Task role
custom-ecs-task-execution-role

Task execution role
custom-ecs-task-execution-role

Operating system/Architecture
Linux/X86_64

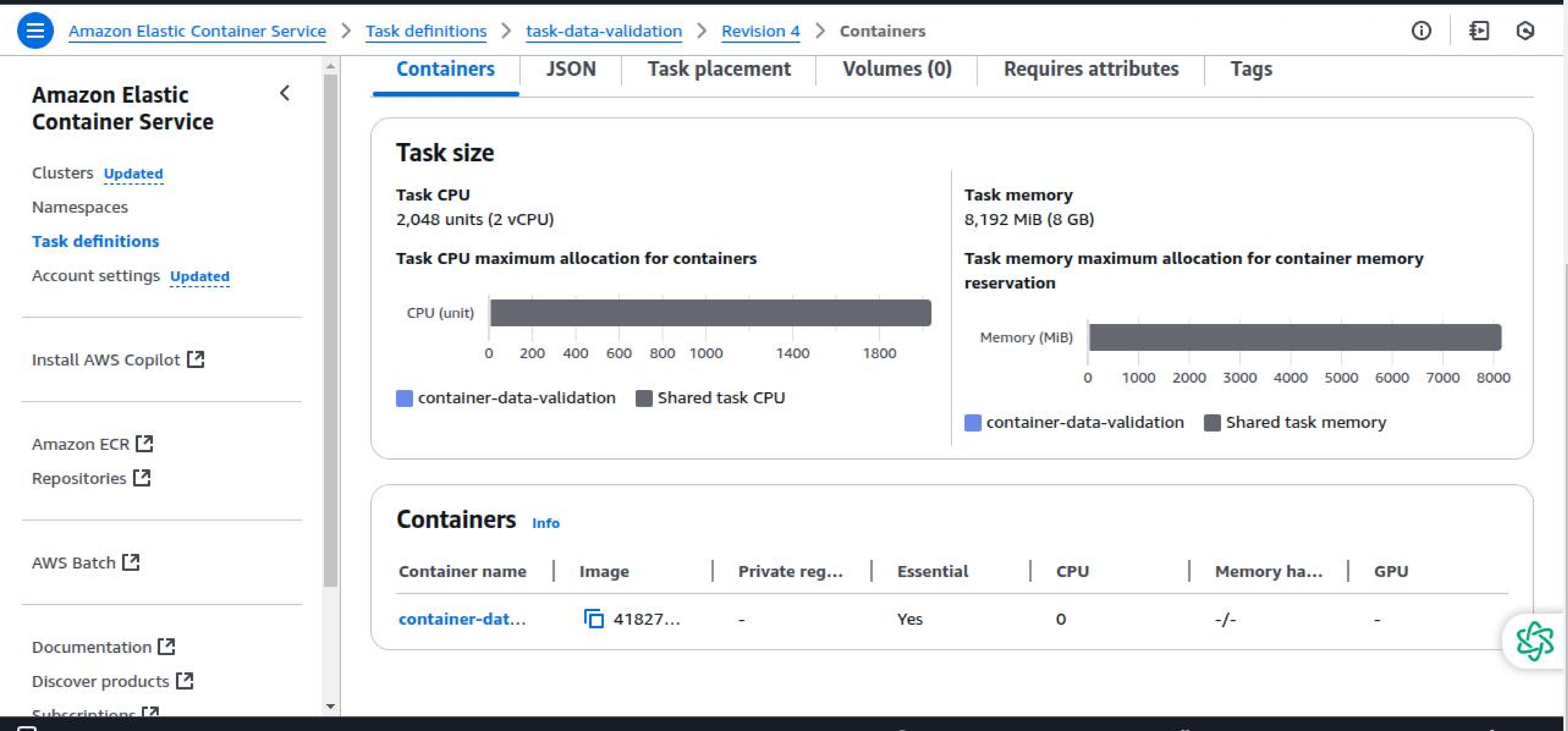
Network mode
awsvpc

Fault injection
Turned off

Task size

Task CPU
2,048 units (2 vCPU)

Task memory
8,192 MiB (8 GB)



Task size

Task CPU

2,048 units (2 vCPU)

Task CPU maximum allocation for containers

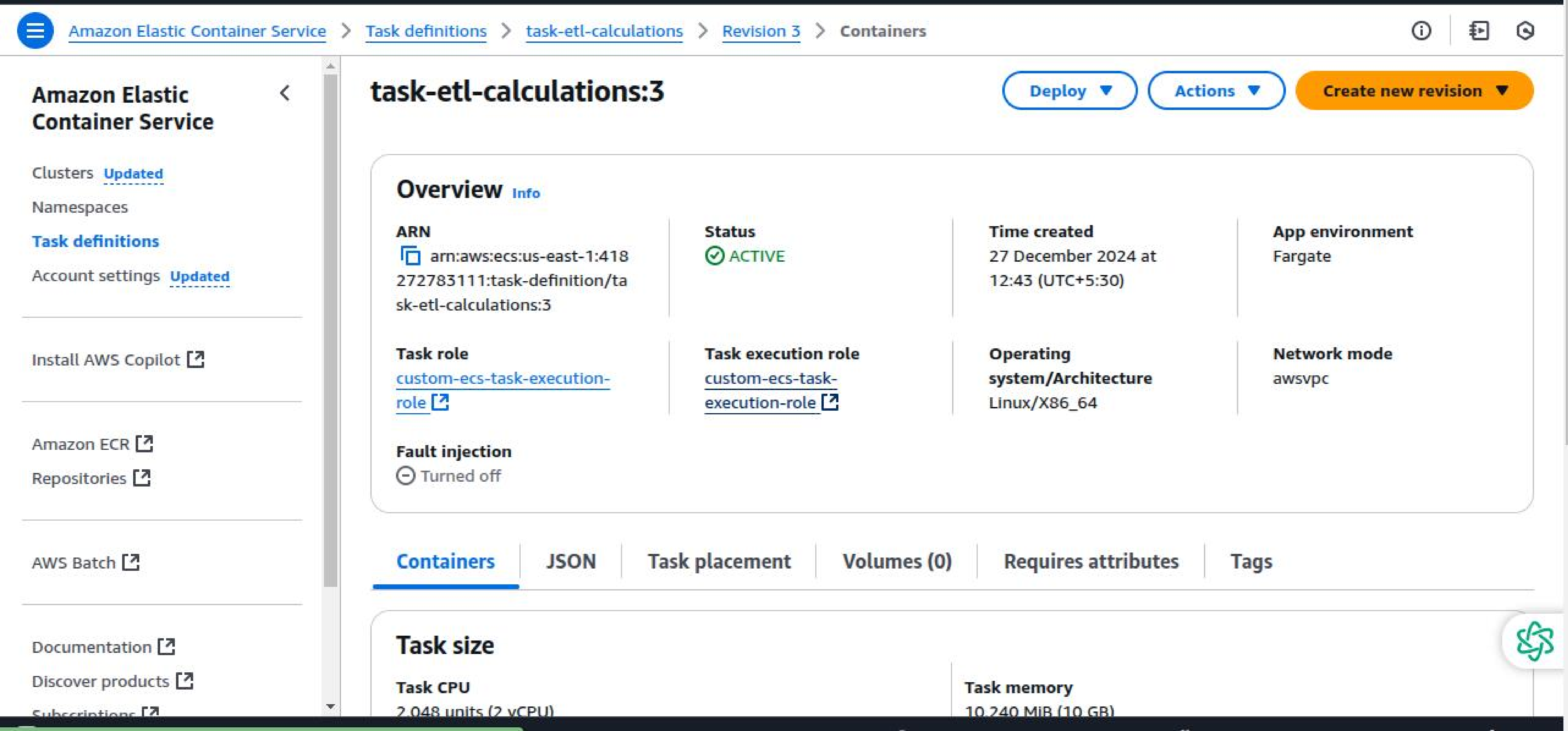
Task memory

8,192 MiB (8 GB)

Task memory maximum allocation for container memory reservation

Containers Info

Container name	Image	Private reg...	Essential	CPU	Memory ha...	GPU
container-dat...	41827...	-	Yes	0	-/-	-


Amazon Elastic Container Service > Task definitions > task-etl-calculations > Revision 3 > Containers

task-etl-calculations:3

Overview Info

ARN
arn:aws:ecs:us-east-1:418272783111:task-definition/task-etl-calculations:3

Status
ACTIVE

Time created
27 December 2024 at 12:43 (UTC+5:30)

App environment
Fargate

Task role
custom-ecs-task-execution-role

Task execution role
custom-ecs-task-execution-role

Operating system/Architecture
Linux/X86_64

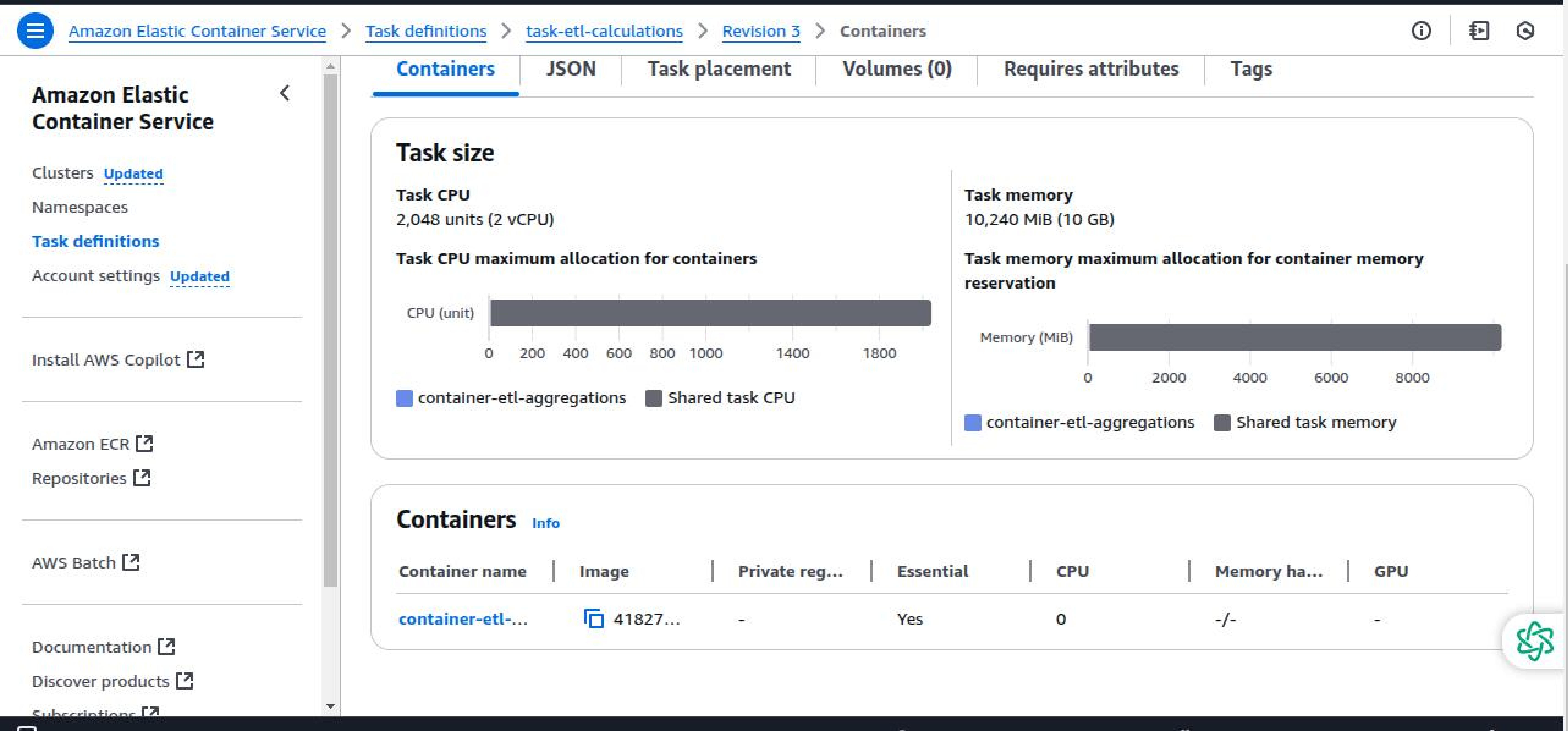
Network mode
awsvpc

Fault injection
Turned off

Task size

Task CPU
2,048 units (2 vCPU)

Task memory
10,240 MiB (10 GB)



Amazon Elastic Container Service > Task definitions > task-etl-calculations > Revision 3 > Containers

Amazon Elastic Container Service

Clusters Updated

Namespaces

Task definitions

Account settings Updated

Install AWS Copilot

Amazon ECR

Repositories

AWS Batch

Documentation

Discover products

Subscriptions

Containers | JSON | Task placement | Volumes (0) | Requires attributes | Tags

Task size

Task CPU
2,048 units (2 vCPU)

Task CPU maximum allocation for containers

🟦 container-etl-aggregations ⬛ Shared task CPU

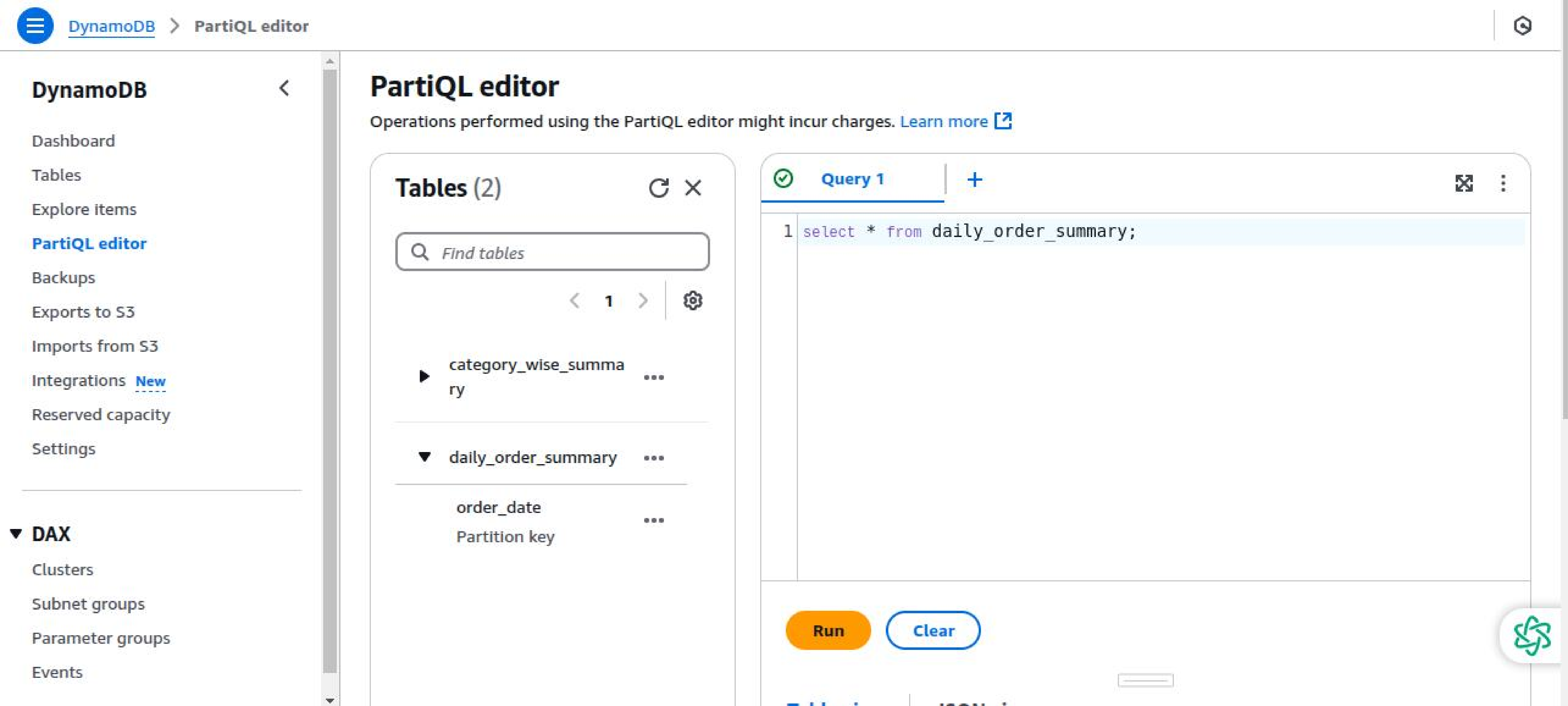
Task memory
10,240 MiB (10 GB)

Task memory maximum allocation for container memory reservation

🟦 container-etl-aggregations ⬛ Shared task memory

Containers Info

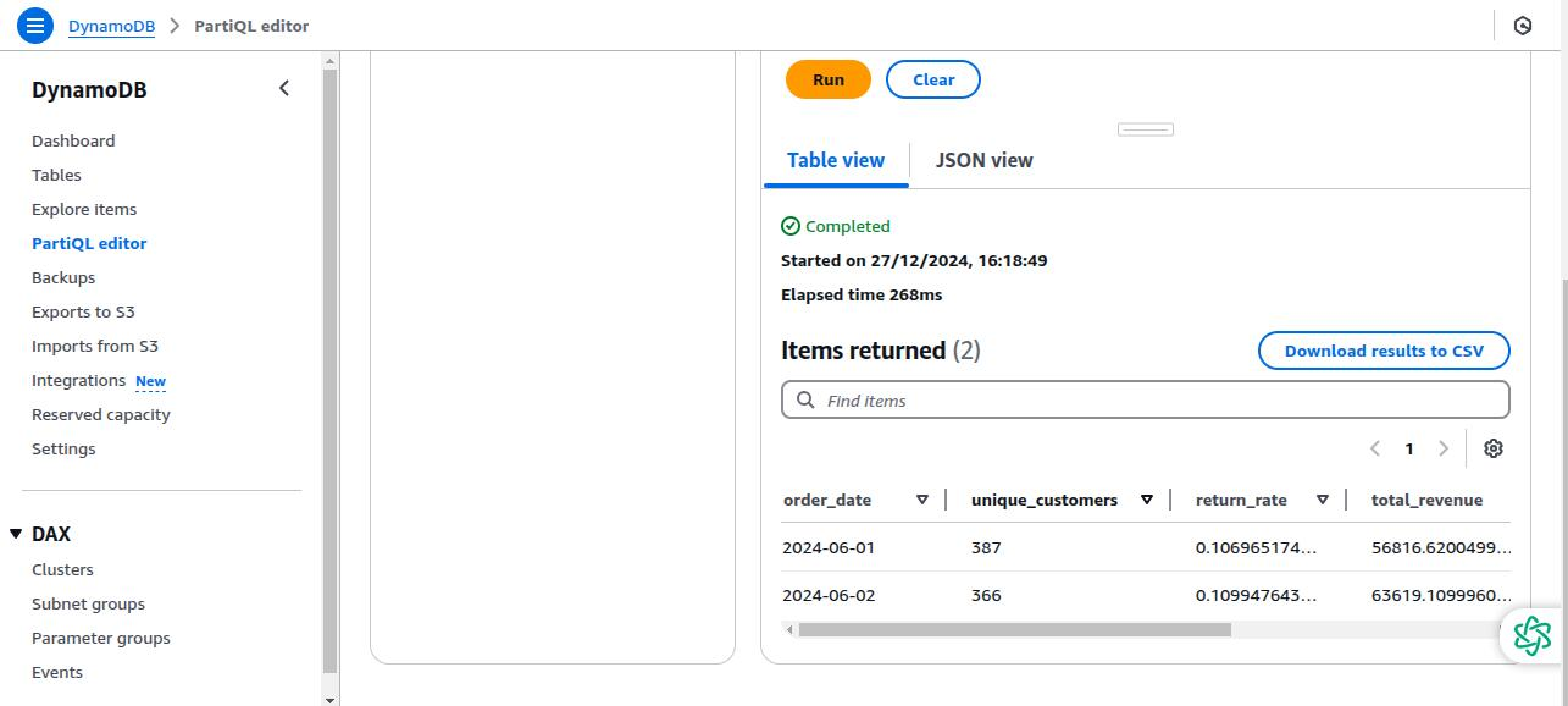
Container name	Image	Private reg...	Essential	CPU	Memory ha...	GPU
container-etl-...	41827...	-	Yes	0	-/-	-



PartiQL editor

Operations performed using the PartiQL editor might incur charges. Learn more

```
select * from daily_order_summary;
```



Completed

Started on 27/12/2024, 16:18:49

Elapsed time 268ms

Items returned (2)

order_date	unique_customers	return_rate	total_revenue
2024-06-01	387	0.106965174…	56816.6200499…
2024-06-02	366	0.109947643…	63619.1099960…