Simple ETL: NASA API TO AWS S3

Oamen Modupe

Project Overview

This project extracts astronomical data from NASA's API, specifically the Astronomy Picture of the Day (APOD) and Near-Earth Object (NEO) datasets. The extracted data is stored in an AWS S3 bucket and orchestrated using Managed Workflows for Apache Airflow (MWAA).

Workflow Steps

1. **Project Setup**
    - Initialize the project using Poetry for dependency management.
    - Set up version control with Git and GitHub.
2. **Pipeline Development**
    - Implement Python scripts to extract data from NASA's APOD and NEO APIs.
    - Create utility functions for handling API requests and data processing.
    - Design Airflow DAGs to automate the data extraction and loading processes.
    - Define necessary variables and configurations in Airflow.
3. **Secrets Management**
    - Create a .env file to store API keys and other environment variables.
    - Load environment variables into scripts.
    - Store sensitive credentials in GitHub Secrets for security.
4. **Infrastructure Setup with AWS CDK**
    - Create raw and DAG S3 buckets using AWS CDK to store extracted data and DAG files.
    - Upload the necessary folders to S3 using Boto3.
    - Deploy MWAA using AWS CDK to orchestrate the data pipeline.
5. **CI/CD and Deployment**
    - Implement automated testing and deployment using GitHub Actions.
    - Ensure pipeline reliability with unit tests and workflow monitoring.

Key Technologies Used

- **Poetry**: Dependency and package management.
- **Airflow (MWAA)**: Orchestrating ETL processes.
- **AWS S3**: Storage for raw and processed data.
- **AWS CDK**: Infrastructure as Code for resource provisioning.
- **GitHub Actions**: Automating deployment and testing.
- **Boto3**: AWS SDK for Python to interact with S3.
- **NASA API**: Data source for astronomical insights.

1. **Project Setup**

- Initialize the project using Poetry for dependency management.

- Install Poetry (if not already installed):
  `pip install poetry`
- Create a new project:
  `poetry new nasa-data-pipeline`
- Navigate to the project directory:
  `cd nasa-data-pipeline`
- Initialize Poetry in an existing project (if applicable):
  `poetry init`
- Install dependencies (example with requests and boto3):
  `poetry add requests boto3 pylint pytest aws-cdk-lib`

- Setting Up Version Control with Git and GitHub

a. Initialize a Git repository:
   `git init`
b. Create a .gitignore file to exclude unnecessary files (e.g., virtual environments, cache, and credentials)
c. Create development branch: `git checkout -b dev`
d. Add project files to Git:
   `git add .`
e. Commit the changes:
   `git commit -m "Initial project setup with Poetry and Git"`
f. Create a new GitHub repository (manually or via CLI) and link it to the local project:
   `git remote add origin https://github.com/your-username/nasa-data-pipeline.git`
g. Push the project to GitHub: `git push origin dev`

2. **Pipeline Development**

- **Implement Python scripts to extract data from NASA's APOD and NEO APIs and load them to S3**

  a. Neo_pipeline.py

  This script extracts data from NASA's Near-Earth Object (NEO) API, processes the data, and loads it into an AWS S3 bucket.

  Key Steps:

  a. Environment Setup:

        - Loads environment variables from a .env file (including the NASA API key).
        - Sets up paths for importing modules and defining project directories.
    b. Extract Data (extract_data function):
        - Calls the NASA NEO API for a specified date range (3 days by default) to fetch information on near-Earth objects
        - Uses the make_api_call function from general_functions to retrieve data
    c. Transform Data (transform function):
        - Processes the data by iterating through each day's asteroid information
        - Extracts relevant fields (e.g., asteroid name, size, approach data, etc.) and stores the data in a Pandas DataFrame
        - Saves the transformed data locally as a CSV file.
    d. Load Data (neo_pipeline function):
        - Calls the transform function to process the extracted data and save it as a CSV
        - Loads the transformed data into the destination system (S3) using the load function from general_functions.

b. apod_pipeline.py

This script extracts data from NASA's Astronomy Picture of the Day (APOD) API, processes the data, and loads it into an AWS S3 bucket.

Key Steps:

a. Environment Setup:
    - Loads environment variables from a .env file (specifically the NASA API key)
    - Defines project directory paths and adds them to sys.path for module imports.
b. Extract Data (extract function):
    - Calls the NASA APOD API with parameters including the current date to retrieve the Astronomy Picture of the Day
    - It uses the make_api_call function from general_functions to handle API requests and fetch data.
c. Transform Data (transform function)
    - Transforms the raw data returned by the API into a Pandas DataFrame for easier manipulation
    - Concatenates multiple data entries into a single DataFrame
    - Saves the transformed DataFrame as a CSV file locally with a date-specific filename.
d. Load Data (apod_pipeline function):
    - Executes the transform function to process the data

        - Loads the transformed data into the destination system (e.g., AWS S3) using the load function from general_functions

- **Create utility functions for handling API requests and data processing.**
    a. general_functions.py
    This module contains utility functions that are globally reusable across the project. These functions handle tasks such as API requests, data processing, and interaction with AWS S3.

    Key Functions:

    a. **Save_data_locally**: Saves a Pandas DataFrame as a CSV file locally and returns the local file path where the CSV file is saved.
    b. **upload_objects_to_s3**: Uploads a file or object to an S3 bucket.
    c. **make_api_call**: Makes a GET request to an API and yields the JSON response if data is available.
    d. **load**: Loads data to an S3 bucket by uploading the local file.

- **Design Airflow DAGs to automate the data extraction and loading processes.**
    a. Etl_dag.py
    This Airflow DAG orchestrates the ETL (Extract, Transform, Load) processes for extracting data from NASA's Astronomy Picture of the Day (APOD) and Near-Earth Object (NEO) APIs, transforming the data, and loading it into an S3 bucket.

    Key Components:

    a. **DAG Configuration (create_dag function)**: Defines the Airflow DAG with the required schedule, tasks, and dependencies for the ETL pipeline.
    It returns the defined Airflow DAG instance.
    This function sets up the DAG to run on a daily basis, with retries configured for failures. It also sets the start date to 2025-03-27 and includes tags for easier identification.
    b. **Task Definitions**:
        - **task_1**: A placeholder start task (EmptyOperator) to signify the beginning of the DAG execution.
        - **task_2**: A PythonOperator that calls the apod_pipeline function, responsible for handling the ETL process for the APOD data.
        - **task_3**: A PythonOperator that calls the neo_pipeline function, responsible for handling the ETL process for the NEO data.
        task_4: An EmptyOperator that marks the end of the DAG execution.

c. **Task Dependencies**:
The tasks are organized in a sequence where task_1 (start task) runs first, then task_2 (APOD ETL) and task_3 (NEO ETL) run in parallel, and finally, task_4 (end task) runs after the completion of both ETL tasks.

3. Secrets Management

- **Create a .env file to store API keys and other environment variables**

  The .env file is used to store sensitive information like API keys and other environment variables locally.

  Example Content:

  ```
  NASA_API_KEY=your_api_key_here

  DATA_BUCKET=your_s3_bucket_name

  DATA_FOLDER_NAME=your_folder_name
  ```

- **Loaded environment variables into scripts**

  The .env file is loaded using the dotenv package, which allows the application to read and access the stored secrets.

  Example Code:

  ```
  from dotenv import load_dotenv

  load_dotenv()

  NASA_API_KEY = os.getenv('NASA_API_KEY')
  ```

  This method ensures that the sensitive credentials are not hard-coded into the source code and can be easily accessed across various scripts (e.g., API requests, S3 operations).

- **Store sensitive credentials in GitHub Secrets for security.**

  For added security in the CI/CD pipeline, sensitive credentials like API keys and access tokens are stored in GitHub Secrets rather than being included directly in the repository. GitHub Secrets are automatically injected into the GitHub Actions environment during deployment, allowing the application to securely access the secrets without exposing them in the code.

Steps:

- Navigate to the GitHub repository
- Go to the **Settings** tab.
- Under **Secrets and variables**, select **Actions**.
- Add new secrets by providing a name (e.g., NASA_API_KEY) and value (the actual API key or secret).

4. Infrastructure Setup with AWS CDK

- **Create S3 buckets using AWS CDK to store extracted data and DAG files, and deploy MWAA**
  a. Setup_infrastructure_stack.py

     This script sets up the infrastructure for storing the extracted data and Airflow DAG files in Amazon S3. It uses AWS CDK (Cloud Development Kit) to define and create two S3 buckets: one for raw data and another for storing the DAG files.

     Key Components:

     a. **Imports**:
        - **aws_cdk**: The core AWS CDK module used to define infrastructure resources.
        - **aws_s3**: The S3 service module to create and configure S3 buckets
        - **RemovalPolicy**: A CDK construct used to define the behavior of the resource when it is removed (e.g., whether to delete the resource)
        - **Construct**: The base class for defining constructs (i.e., AWS resources).
        - **variables.py**: Imports bucket names (DAGS_BUCKET and DATA_BUCKET) from a separate variables module.

     b. **Class SetupInfrastructureStack**:
        This class defines a stack that creates two S3 buckets using AWS CDK.
        The class inherits from Stack, and inside its constructor, it creates two S3 buckets

  b. Aws_mwaa_stack.py

     This script defines an AWS CDK stack to set up a managed Airflow environment using Amazon MWAA (Managed Workflows for Apache Airflow). It creates the necessary IAM roles, network configurations, and associates the MWAA environment with an existing S3 bucket for storing Airflow DAG files.

Key Components:

a. **Imports**:
   - **aws_cdk**: The core AWS CDK module for defining AWS resources.
   - **aws_mwaa**: The AWS MWAA module for managing Airflow environments.
   - **aws_s3**: The S3 service module for managing S3 buckets.
   - **aws_iam**: The IAM service module for managing roles and policies.
   - **aws_ec2**: The EC2 module for managing VPCs and subnets.
   - **constructs**: The base class for defining AWS constructs (i.e., resources).

b. **Class MwaaStack**:
   This class defines a stack to set up an Amazon MWAA environment, including the required IAM role, network configurations, and S3 integration.

c. **IAM Role for MWAA**:
   Defines an IAM role (MwaaExecutionRole) for the MWAA environment to interact with other AWS services (such as S3 and Lambda).

   Policies:

   - **AWSLambdaBasicExecutionRole**: Grants basic Lambda execution permissions.
   - **AmazonS3FullAccess**: Grants full access to S3 (you can adjust this policy based on security needs

d. **VPC Configuration**:

   Private Subnets: Extracts the private subnets from the VPC for added security. The MWAA environment will run in these private subnets.

e. **MWAA Environment**
   - **Airflow Version**: Specifies the version of Airflow to use (e.g., "2.10.1").
   - **Execution Role ARN**: The IAM role ARN used by MWAA for permissions
   - **Source Bucket**: The S3 bucket containing the DAGs (existing_s3_bucket).
   - **DAG Path**: The S3 path where DAGs are stored (dag_s3_path).
   - **Environment Class**: Defines the compute resources for the MWAA environment (mw1.micro in this case).
   - **Network Configuration**: Configures the network settings for MWAA, using private subnets and a security group.
   - **Webserver Access Mode**: Configures access to the Airflow webserver. This example uses "PUBLIC_ONLY", but it could be adjusted for private access.
   - **Tags**: Adds tags to the MWAA environment for easier identification (e.g., "Project": "Airflow").

c. Vpc_stack.py

   This script defines an AWS CDK stack to create a Virtual Private Cloud (VPC) with both public and private subnets. The VPC is designed to support the deployment of resources such as Amazon MWAA in a secure and scalable network configuration.

   Key Components:

   a. **Imports**:
      - **aws_cdk**: The core AWS CDK module for defining AWS resources
      - **aws_ec2**: The EC2 module for managing networking resources, such as VPCs and subnets.
      - **constructs**: The base class for defining AWS constructs (i.e., resources).

   b. **Class VpcStack**:
      This class defines a stack that creates a VPC with both public and private subnets, suitable for running AWS services like MWAA in a secure network.

   c. **VPC Configuration**:
      - **max_azs=2**: This parameter specifies that two availability zones (AZs) should be used for the VPC. This ensures high availability across multiple AZs.

   d. **Subnets**:
      - **Public Subnet**: A subnet that is exposed to the internet (e.g., for web servers or other public-facing services).
      - **Private Subnet**: A subnet that is not directly exposed to the internet but can access the internet via a NAT gateway or similar setup (PRIVATE_WITH_EGRESS).

d. app.py

   This script defines the main entry point for the AWS CDK application. It synthesizes the necessary CloudFormation templates by initializing the stacks required for the infrastructure setup, including the creation of S3 buckets, VPC, and MWAA environment

   - **Upload the necessary folders to S3 using Boto3**
      - create_folders.py

        This script uploads specified folders to the respective Amazon S3 buckets. It uses a reusable function (upload_objects_to_s3) to upload folders to S3.

6. CI/CD and Deployment

- **Implement automated testing and deployment using GitHub Actions.**

- **Ensure pipeline reliability with unit tests and workflow monitoring.**
    a. Deploy.yml

       This GitHub Actions pipeline automates the deployment of files to an S3 bucket whenever changes are pushed to the dev branch. It is specifically designed to upload Airflow DAGs to an S3 bucket.

    b. tests.yml

       This GitHub Actions pipeline automates the testing process of your codebase, running linting and unit tests whenever changes are made. It is triggered by both pull requests targeting the main or master branches and pushes to the dev branch.

Resources

https://api.nasa.gov/index.html
https://jsoneditoronline.org/#left=local.huqobo&right=local.wadaki
https://github.com/NeonStone7/Basecone_data_pipeline/actions
https://dzone.com/articles/deploying-managed-workflows-for-apache-airflow-mwa