Airflow setup on ECS

Table of Contents

[Overview 1](#_Toc184718251)

[End to end setup 1](#_Toc184718252)

[Dependencies 1](#_Toc184718253)

[DockerFile 1](#_Toc184718254)

[Entrypoint File 3](#_Toc184718255)

[Airflow configuration 3](#_Toc184718256)

[Elastic Container Service 4](#_Toc184718257)

[Components 4](#_Toc184718258)

[Service running in ECS 5](#_Toc184718259)

[GIT Integration for CI/CD 6](#_Toc184718260)

[Steps to setup GIT 6](#_Toc184718261)

[Steps to Setup Airflow on ECS 8](#_Toc184718262)

# Overview

* Apache Airflow is Deployed with Containerization using docker to enhance scalability and availability of the service to orchestrate and schedule DataEngineering pipelines to run in batch process.
* The Airflow is tightly integrated with GIT for CI/CD of DAG migration from Azure Git repository. This supports seamless deployment and versioning of code on GIT.
* The Airflow database is backed by Amazon RDS Postgresql to enable the capabilities of DAG, connection management and parallel execution of DAGs in Prod environment.
* Leveraged AWS ECS with Fargate to bring up the serverless service to minimize operational efforts needed to manage the infrastructure.

## End to end setup

A computer screen shot of a computer

Description automatically generated

# Dependencies

Create all the necessary files needed to setup ECS on AWS.

## DockerFile

The Dockerfile below is composed of all the dependencies to enable Airflow 2.9.2 service backed by Python3.9 to orchestrate batch jobs.

* Apache Airflow 2.9.2
* Python3.9
* Psycopg2 for Postgres connection
* Pem files to connect to EC2 servers as needed
* GIT
* A DAG to sync the Airflow DAGs from GIT to container location

# Use the official airflow image

FROM apache/airflow:2.9.2-python3.9

# Install psycopg2

RUN pip install psycopg2-binary

#Copy airflow config file to container

COPY airflow.cfg /opt/airflow/airflow.cfg

#Copy pem files of ec2 servers where the Data Engineering pipelines reside

COPY linux.pem /opt/airflow/linux.pem

COPY Oregon.pem /opt/airflow/Oregon.pem

# Switch to root user to install Git and enable container to sync DAGs on GIT repo

USER root

# Install Git

RUN apt-get update && apt-get install -y git openssh-client

# Copy the SSH keys into the container. These are needed for Azure git authentication

COPY id\_rsa /root/.ssh/id\_rsa

COPY id\_rsa.pub /root/.ssh/id\_rsa.pub

# Set permissions for the SSH keys and pem files

RUN chmod 600 /root/.ssh/id\_rsa && chmod 644 /root/.ssh/id\_rsa.pub

RUN chmod 600 /opt/airflow/linux.pem

RUN chmod 600 /opt/airflow/Oregon.pem

# Add Azure DevOps SSH host to known\_hosts to avoid host verification issues

RUN mkdir -p /root/.ssh && ssh-keyscan ssh.dev.azure.com >> /root/.ssh/known\_hosts

# Set permissions for SSH directory and known\_hosts

RUN chmod 700 /root/.ssh && chmod 644 /root/.ssh/known\_hosts

#Execute only if we want to force refresh dags from git

RUN rm -rf /opt/airflow/dags/\* && git clone git@ssh.dev.azure.com:v3/devops/DEA/dag-composer /opt/airflow/dags

#Copy dag sync file to airflow

COPY prod\_dag\_sync.py /opt/airflow/dags/

# Copy the entrypoint script into the image

COPY entrypoint.sh /usr/local/bin/entrypoint.sh

# Set the entrypoint to the script

ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]

#Expose the webserver port

EXPOSE 8080

## Entrypoint File

A shell script called **Entrypoint.sh** file will be executed on the container to initiate the Airflow service.

#!/bin/bash

# Initialize the Airflow database

airflow db init

# Start the Airflow webserver in the background

airflow webserver &

# Start the Airflow scheduler

airflow scheduler

# Wait for all background jobs to finish

wait

## Airflow configuration

The configurations are defined in the airflow.cfg that contains following sections:

Core – DAG Path and Executory type

Database – Connections pointing to database (Define postgres connection in this case)

Email and Webserver -email server and webserver server details with IP and port information

[core]

dags\_folder = /opt/airflow/dags

executor = LocalExecutor

load\_examples = False

[database]

sql\_alchemy\_conn =

[email]

email\_backend = airflow.utils.email.send\_email\_smtp

email\_conn\_id = smtp\_default

default\_email\_on\_retry = True

default\_email\_on\_failure = True

[smtp]

smtp\_host =

smtp\_starttls = True

smtp\_ssl = False

smtp\_port =

smtp\_mail\_from = data\_business\_intelligence@company.com

smtp\_timeout = 30

smtp\_retry\_limit = 2

[webserver]

rbac = True

web\_server\_host = 0.0.0.0

web\_server\_port = 8080

# Elastic Container Service

The AWS ECS setup is needed to run the docker image as a ‘Service’ on ECS cluster.

## Components

* ECS Cluster ‘[DataEngineering\_Dev](https://us-west-2.console.aws.amazon.com/ecs/v2/clusters/DataEngineering_Dev?region=us-west-2" \o "https://us-west-2.console.aws.amazon.com/ecs/v2/clusters/DataEngineering_Dev?region=us-west-2)’
* ECS Task Definition :
  + Name: airflow\_task
  + Launch Type: Fargate
  + Task Size - 2 vCPU, 8 GB Memory
  + Container Details:
    - Provide image URI
    - Port Mapping
      * Container port : 8080
      * Protocol: TCP
    - Specify Container level CPU and Memory
  + Specify Ephemeral Storage needed for container
  + Specify other properties like log location, health metrics, Environment variables
* Create Service:
  + Launch Type: Fargate
  + Application Type: Service
  + Family: airflow\_task
  + Service Name: [Airflow\_service\_v2](https://us-west-2.console.aws.amazon.com/ecs/v2/clusters/DataEngineering_Dev/services/Airflow_service_v2/health?region=us-west-2)
  + Desired Tasks: 2
  + Deployment Type: Rolling update
  + [Optional]Load Balancing: Choose ALB and the Target Group where the ALB is created
  + Networking: Choose VPC (Boomi - Prod/Dev - TFS) and Security Groups

## Service running in ECS A screenshot of a computer Description automatically generated

# GIT Integration for CI/CD

The Airflow setup is tightly integrated with Azure GIT repository ‘dag-composer’ to sync all the DAGS.

## Steps to setup GIT

* Generate keys on the local machine: ssh-keygen -t rsa
* Navigate to Azure GIT → User Settings → SSH Public keys → New Key → Copy paste the public key generated from the above step
* Copy the public and private key files to the container under /root/.ssh
* Assign right permissions to the keys as defined in Dockerfile
* Copy the dag files from git repo to /opt/airflow/dags/ executing ‘git clone’ on the container which will execute when the image is setup on ECS service for the first time or when updated again.

RUN rm -rf /opt/airflow/dags/\* && git clone git@ssh.dev.azure.com:v3/devops/DEA/dag-composer /opt/airflow/dags

* Created DAG ‘**prod\_dag\_sync**’ to sync the airflow dags from GIT to the airflow dag location on hourly basis.
* This will keep track of new changes added to GIT repository and pull them into Airflow automatically.

from airflow import DAG

from airflow.operators.bash\_operator import BashOperator

from datetime import datetime, timedelta

args = {

"owner": "airflow",

"provide\_context": True,

'email': [

'myemail@mycompany.com'

],

'email\_on\_failure': True,

'email\_on\_retry': False,

'retries': 0,

'retry\_delay': timedelta(seconds=30)

}

with DAG(

dag\_id='prod\_dag\_sync',

schedule\_interval='@hourly',

start\_date=datetime(2024, 10, 15),

default\_args=args,

catchup=False,

tags=["PROD|GIT\_DAG\_SYNC"]

) as dag:

git\_clone = BashOperator(

task\_id="git\_clone",

bash\_command="""

git config --global --add safe.directory /opt/airflow/dags

if [ ! -d /opt/airflow/dags/.git ]; then

echo "Files not available in dag path"

else

echo "Pulling latest changes"

cd /opt/airflow/dags && git pull origin main;

fi

"""

)

git\_clone

# Steps to Setup Airflow on ECS

1. Setup Docker on your machine
2. Create the below prereq files:
   1. airflow.cfg with customized configuration
   2. entrypoint.sh used to initiate Airflow on the container
   3. Public and private SSH keys for GIT authentication with Azure repo.
3. Create Airflow folder on your local machine and navigate to it from your terminal
4. Create DockerFile as defined in DockerFile section
5. Build the docker image from the DockerFile using ‘docker build’ command
6. Run the docker image locally using ‘docker run’ to do the following.
   1. Validate the service is installed successfully
   2. Set up admin login to connect to Airflow
      1. docker exec -it airflow /bin/bash
      2. airflow users create --username admin --firstname xxxx --lastname xxxx --role Admin --email xxxx[@xxxx.com](mailto:balasubramanyam.vakkanthamramamurthy@intapp.com) --password xxxx
      3. exit
   3. Test the service by connecting to Airflow UI using ‘localhost:8080’
7. Stop the service on local
8. Create necessary AWS services to run Airflow on ECS
   1. Create the Elastic Container Registry on AWS where the docker images can be stored and provide necessary authentication for the users to push the docker images from local repository to [ECR](https://aws.amazon.com/ecr/getting-started/)
   2. Create [Elastic Container Service](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/get-set-up-for-amazon-ecs.html) where the docker images run based on the configs defined
   3. [optiopnal] Set up Application Load blancer on AWS with Target Groups to have a permanent url – (Ex: <https://de-airflow.company.net/> )
9. Install [AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html) and configure it with AWS keys
10. Push the docker image to ECR using the commands defined in AWS ECR (ECR → Select Repository → View push commands) .
11. Navigate to ECS and create a Task Definition by specifying the URI of the image and the capacity needed.
12. Navigate to ECS cluster and create service and provide the task definition and define the configuration to execute Fargate ECS service.