aws

Panduan Migrasi

Amazon Managed Workflows for Apache Airflow

Amazon Managed Workflows for Apache Airflow: Panduan Migrasi

Table of Contents

Apa itu panduan migrasi Amazon MWAA?

Alur Kerja Terkelola Amazon untuk Apache Airflow adalah layanan orkestrasi terkelola untuk Apache Airflow yang memungkinkan Anda mengoperasikan pipeline data di cloud dalam skala besar. Amazon MWAA mengelola penyediaan dan pemeliharaan berkelanjutan Apache Airflow sehingga Anda tidak perlu lagi khawatir tentang menambal, menskalakan, atau mengamankan instance.

Amazon MWAA secara otomatis menskalakan sumber daya komputasi yang menjalankan tugas untuk memberikan kinerja yang konsisten sesuai permintaan. Amazon MWAA mengamankan data Anda secara default. Beban kerja Anda berjalan di lingkungan cloud Anda sendiri yang terisolasi dan aman menggunakan Amazon Virtual Private Cloud. Ini memastikan bahwa data dienkripsi secara otomatis menggunakan. AWS Key Management Service

Gunakan panduan ini untuk memigrasikan alur kerja Apache Airflow yang dikelola sendiri ke Amazon MWAA, atau memutakhirkan lingkungan MWAA Amazon yang ada ke versi Apache Airflow yang baru. Tutorial migrasi menjelaskan cara membuat, atau mengkloning lingkungan Amazon MWAA baru, memigrasikan sumber daya alur kerja, dan mentransfer metadata alur kerja dan log ke lingkungan baru Anda.

Sebelum Anda mencoba tutorial migrasi, kami sarankan untuk meninjau topik berikut.

- Arsitektur jaringan
- Pertimbangan utama

Jelajahi arsitektur jaringan Amazon MWAA

Bagian berikut menjelaskan komponen utama yang membentuk lingkungan Amazon MWAA, dan kumpulan AWS layanan yang diintegrasikan oleh setiap lingkungan untuk mengelola sumber dayanya, menjaga keamanan data Anda, dan menyediakan pemantauan dan visibilitas untuk alur kerja Anda.

Topik

Komponen Amazon MWAA

Lingkungan Amazon MWAA terdiri dari empat komponen utama berikut:

1. Scheduler - Mem-parsing dan memantau semua DAG Anda, dan mengantri tugas untuk dieksekusi ketika dependensi DAG terpenuhi. Amazon MWAA menyebarkan penjadwal sebagai AWS Fargate cluster dengan minimal 2 penjadwal. Anda dapat meningkatkan jumlah penjadwal hingga lima, tergantung pada beban kerja Anda. Untuk informasi selengkapnya tentang kelas lingkungan Amazon MWAA, lihat kelas lingkungan Amazon MWAA.
2. Pekerja — Satu atau lebih tugas Fargate yang menjalankan tugas terjadwal Anda. Jumlah pekerja untuk lingkungan Anda ditentukan oleh rentang antara jumlah minimum dan maksimum yang Anda tentukan. Amazon MWAA memulai auto-scaling worker ketika jumlah tugas yang diantrian dan berjalan lebih dari yang dapat ditangani oleh pekerja Anda yang ada. Saat menjalankan dan mengantri tugas berjumlah nol selama lebih dari dua menit, Amazon MWAA mengurangi jumlah pekerja ke minimum. Untuk informasi selengkapnya tentang cara Amazon MWAA menangani pekerja auto-scaling, lihat penskalaan otomatis Amazon MWAA.
3. Server web - Menjalankan UI web Apache Airflow. Anda dapat mengkonfigurasi server web dengan akses jaringan pribadi atau publik. Dalam kedua kasus tersebut, akses ke pengguna Apache Airflow Anda dikendalikan oleh kebijakan kontrol akses yang Anda tentukan AWS Identity and Access Management (IAM). Untuk informasi selengkapnya tentang mengonfigurasi kebijakan akses IAM untuk lingkungan Anda, lihat Mengakses lingkungan Amazon MWAA.
4. Database - Menyimpan metadata tentang lingkungan Apache Airflow dan alur kerja Anda, termasuk riwayat run DAG. Basis data adalah database Aurora PostgreSQL penyewa tunggal

yang dikelola oleh, AWS dan dapat diakses oleh penjadwal dan kontainer Fargate pekerja melalui titik akhir Amazon VPC yang diamankan secara pribadi.

Setiap lingkungan Amazon MWAA juga berinteraksi dengan serangkaian AWS layanan untuk menangani berbagai tugas, termasuk menyimpan dan mengakses DAG dan dependensi tugas, mengamankan data Anda saat istirahat, dan mencatat dan memantau lingkungan Anda. Diagram berikut menunjukkan komponen yang berbeda dari lingkungan Amazon MWAA.

> **Note**
>
> Layanan Amazon VPC bukan VPC bersama. Amazon MWAA menciptakan AWS VPC milik untuk setiap lingkungan yang Anda buat.

- Amazon S3 - Amazon MWAA menyimpan semua sumber daya alur kerja Anda, seperti DAG, persyaratan, dan file plugin di bucket Amazon S3. Untuk informasi selengkapnya tentang membuat bucket sebagai bagian dari pembuatan lingkungan, dan mengunggah sumber daya Amazon MWAA Anda, lihat Membuat bucket Amazon S3 untuk Amazon MWAA di Panduan Pengguna Amazon MWAA.
- Amazon SQS — Amazon MWAA menggunakan Amazon SQS untuk mengantri tugas alur kerja Anda dengan pelaksana Seledri.
- Amazon ECR - Amazon ECR menampung semua gambar Apache Airflow. Amazon MWAA hanya mendukung gambar AWS Apache Airflow yang dikelola.
- AWS KMS— Amazon MWAA menggunakan AWS KMS untuk memastikan data Anda aman saat istirahat. Secara default, Amazon MWAA menggunakan AWS KMS kunci yang AWS dikelola, tetapi Anda dapat mengonfigurasi lingkungan untuk menggunakan kunci yang dikelola pelanggan Anda sendiri. AWS KMS Untuk informasi selengkapnya tentang menggunakan AWS KMS kunci yang dikelola pelanggan Anda sendiri, lihat Customer-managed kunci untuk Enkripsi Data di Panduan Pengguna Amazon MWAA.
- CloudWatch- Amazon MWAA terintegrasi dengan CloudWatch dan mengirimkan log Apache Airflow dan metrik lingkungan CloudWatch, memungkinkan Anda memantau sumber daya Amazon MWAA Anda dan memecahkan masalah.

Konektivitas

Lingkungan Amazon MWAA Anda membutuhkan akses ke semua AWS layanan yang terintegrasi dengannya. Peran eksekusi Amazon MWAA mengontrol bagaimana akses diberikan ke Amazon MWAA untuk terhubung ke AWS layanan lain atas nama Anda. Untuk konektivitas jaringan, Anda dapat menyediakan akses internet publik ke VPC Amazon atau membuat titik akhir VPC Amazon. Untuk informasi selengkapnya tentang mengonfigurasi titik akhir VPC Amazon AWS PrivateLink() untuk lingkungan Anda, lihat Mengelola akses ke titik akhir VPC di Amazon MWAA di Panduan Pengguna Amazon MWAA.

Amazon MWAA menginstal persyaratan pada penjadwal dan pekerja. Jika kebutuhan Anda bersumber dari PyPirepositori publik, lingkungan Anda memerlukan konektivitas ke internet untuk mengunduh pustaka yang diperlukan. Untuk lingkungan pribadi, Anda dapat menggunakan PyPi repositori pribadi, atau menggabungkan pustaka dalam `.whl`file sebagai plugin khusus untuk lingkungan Anda.

Saat Anda mengonfigurasi Apache Airflow dalam mode pribadi, UI Apache Airflow hanya dapat diakses oleh VPC Amazon Anda melalui titik akhir Amazon VPC.

Untuk informasi selengkapnya tentang jaringan, lihat Jaringan di Panduan Pengguna Amazon MWAA.

Pertimbangan utama untuk bermigrasi ke lingkungan MWAA baru

Pelajari lebih lanjut tentang pertimbangan utama, seperti otentikasi dan peran eksekusi Amazon MWAA, saat Anda berencana untuk memigrasikan beban kerja Apache Airflow ke Amazon MWAA.

Topik

- Autentikasi
- Peran eksekusi

Autentikasi

Amazon MWAA menggunakan AWS Identity and Access Management (IAM) untuk mengontrol akses ke Apache Airflow UI. Anda harus membuat dan mengelola kebijakan IAM yang memberikan izin kepada pengguna Apache Airflow Anda untuk mengakses server web dan mengelola. DAGs Anda dapat mengelola otentikasi dan otorisasi untuk peran default Apache Airflow menggunakan IAM di berbagai akun.

Anda dapat lebih lanjut mengelola dan membatasi pengguna Apache Airflow untuk mengakses hanya sebagian dari alur DAGs kerja Anda dengan membuat peran Airflow kustom dan memetakannya ke prinsipal IAM Anda. Untuk informasi lebih lanjut dan step-by-step tutorial, lihat Tutorial: Membatasi akses pengguna Amazon MWAA ke subset. DAGs

Anda juga dapat mengonfigurasi identitas federasi untuk mengakses Amazon MWAA. Untuk informasi lebih lanjut lihat yang berikut ini.

- Lingkungan Amazon MWAA dengan akses publik - Menggunakan Okta sebagai penyedia identitas dengan Amazon MWAA di Blog Komputasi.AWS
- Lingkungan Amazon MWAA dengan akses pribadi - Mengakses lingkungan MWAA Amazon pribadi menggunakan identitas federasi.

Peran eksekusi

Amazon MWAA menggunakan peran eksekusi yang memberikan izin ke lingkungan Anda untuk mengakses layanan lain. AWS Anda dapat memberikan alur kerja Anda akses ke AWS layanan

dengan menambahkan izin yang relevan ke peran. Jika Anda memilih opsi default untuk membuat peran eksekusi baru saat pertama kali membuat lingkungan, Amazon MWAA melampirkan izin minimal yang diperlukan untuk peran tersebut, kecuali dalam kasus Log yang Amazon MWAA CloudWatch menambahkan semua grup log secara otomatis.

Setelah peran eksekusi dibuat, Amazon MWAA tidak dapat mengelola kebijakan izinnya atas nama Anda. Untuk memperbarui peran eksekusi, Anda harus mengedit kebijakan untuk menambah dan menghapus izin sesuai kebutuhan. Misalnya, Anda dapat mengintegrasikan lingkungan Amazon MWAA Anda AWS Secrets Manager sebagai backend untuk menyimpan rahasia dan string koneksi dengan aman untuk digunakan dalam alur kerja Apache Airflow Anda. Untuk melakukannya, lampirkan kebijakan izin berikut ke peran eksekusi lingkungan Anda.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-east-1:111122223333:secret:*"
        },
        {
            "Effect": "Allow",
            "Action": "secretsmanager:ListSecrets",
            "Resource": "*"
        }
    ]
}
```

Mengintegrasikan dengan AWS layanan lain mengikuti pola yang sama: Anda menambahkan kebijakan izin yang relevan ke peran eksekusi Amazon MWAA Anda, memberikan izin kepada Amazon MWAA untuk mengakses layanan. Untuk informasi selengkapnya tentang mengelola peran

eksekusi Amazon MWAA, dan untuk melihat contoh tambahan, kunjungi peran eksekusi Amazon MWAA di Panduan Pengguna Amazon MWAA.

Bermigrasi ke lingkungan Amazon MWAA baru

Jelajahi langkah-langkah berikut untuk memigrasikan beban kerja Apache Airflow Anda yang ada ke lingkungan Amazon MWAA baru. Anda dapat menggunakan langkah-langkah ini untuk bermigrasi dari Amazon MWAA versi lama ke rilis versi baru, atau memigrasikan penyebaran Apache Airflow yang dikelola sendiri ke Amazon MWAA. Tutorial ini mengasumsikan Anda bermigrasi dari Apache Airflow v1.10.12 yang ada ke Amazon MWAA baru yang menjalankan Apache Airflow v2.5.1, tetapi Anda dapat menggunakan prosedur yang sama untuk bermigrasi dari, atau ke versi Apache Airflow yang berbeda.

Topik

Prasyarat

Untuk dapat menyelesaikan langkah-langkah dan memigrasikan lingkungan Anda, Anda memerlukan yang berikut:

- Penyebaran Apache Airflow. Ini bisa berupa lingkungan MWAA Amazon yang dikelola sendiri atau yang ada.
- Docker diinstal untuk sistem operasi lokal Anda.
- AWS Command Line Interface versi 2 diinstal.

Langkah satu: Buat lingkungan Amazon MWAA baru yang menjalankan versi Apache Airflow terbaru yang didukung

Anda dapat membuat lingkungan menggunakan langkah-langkah terperinci dalam Memulai Amazon MWAA di Panduan Pengguna Amazon MWAA, atau dengan menggunakan templat. CloudFormation

Jika Anda bermigrasi dari lingkungan Amazon MWAA yang ada, dan menggunakan CloudFormation templat untuk membuat lingkungan lama, Anda dapat mengubah `AirflowVersion` properti untuk menentukan versi baru.

```
MwaaEnvironment:
  Type: AWS::MWAA::Environment
  DependsOn: MwaaExecutionPolicy
  Properties:
    Name: !Sub "${AWS::StackName}-MwaaEnvironment"
    SourceBucketArn: !GetAtt EnvironmentBucket.Arn
    ExecutionRoleArn: !GetAtt MwaaExecutionRole.Arn
    AirflowVersion: 2.5.1
    DagS3Path: dags
  NetworkConfiguration:
    SecurityGroupIds:
      - !GetAtt SecurityGroup.GroupId
    SubnetIds:
      - !Ref PrivateSubnet1
      - !Ref PrivateSubnet2
  WebserverAccessMode: PUBLIC_ONLY
  MaxWorkers: !Ref MaxWorkerNodes
  LoggingConfiguration:
    DagProcessingLogs:
      LogLevel: !Ref DagProcessingLogs
      Enabled: true
    SchedulerLogs:
      LogLevel: !Ref SchedulerLogsLevel
      Enabled: true
    TaskLogs:
      LogLevel: !Ref TaskLogsLevel
      Enabled: true
    WorkerLogs:
      LogLevel: !Ref WorkerLogsLevel
      Enabled: true
    WebserverLogs:
      LogLevel: !Ref WebserverLogsLevel
      Enabled: true
```

Atau, jika bermigrasi dari lingkungan Amazon MWAA yang ada, Anda dapat menyalin skrip Python berikut yang menggunakan AWS SDK for Python (Boto3) untuk mengkloning lingkungan Anda. Anda juga dapat mengunduh skrip.

Skrip Python

```
# This Python file uses the following encoding: utf-8
'''
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
SPDX-License-Identifier: MIT-0

Permission is hereby granted, free of charge, to any person obtaining a copy of this
software and associated documentation files (the "Software"), to deal in the Software
without restriction, including without limitation the rights to use, copy, modify,
merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
'''
from __future__ import print_function
import argparse
import json
import socket
import time
import re
import sys
from datetime import timedelta
from datetime import datetime
import boto3
from botocore.exceptions import ClientError, ProfileNotFound
from boto3.session import Session
ENV_NAME = ""
REGION = ""

def verify_boto3(boto3_current_version):
    '''
    check if boto3 version is valid, must be 1.17.80 and up
    return true if all dependenceis are valid, false otherwise
    '''
    valid_starting_version = '1.17.80'
    if boto3_current_version == valid_starting_version:
        return True
```

```
    ver1 = boto3_current_version.split('.')
    ver2 = valid_starting_version.split('.')
    for i in range(max(len(ver1), len(ver2))):
        num1 = int(ver1[i]) if i < len(ver1) else 0
        num2 = int(ver2[i]) if i < len(ver2) else 0
        if num1 > num2:
            return True
        elif num1 < num2:
            return False
    return False


def get_account_id(env_info):
    '''
    Given the environment metadata, fetch the account id from the
    environment ARN
    '''
    return env_info['Arn'].split(":")[4]


def validate_envname(env_name):
    '''
    verify environment name doesn't have path to files or unexpected input
    '''
    if re.match(r"^[a-zA-Z][0-9a-zA-Z-_]*$", env_name):
        return env_name
    raise argparse.ArgumentTypeError("%s is an invalid environment name value" %
 env_name)


def validation_region(input_region):
    '''
    verify environment name doesn't have path to files or unexpected input
    REGION: example is us-east-1
    '''
    session = Session()
    mwaa_regions = session.get_available_regions('mwaa')
    if input_region in mwaa_regions:
        return input_region
    raise argparse.ArgumentTypeError("%s is an invalid REGION value" % input_region)


def validation_profile(profile_name):
    '''
```

```
    verify profile name doesn't have path to files or unexpected input
    '''
    if re.match(r"^[a-zA-Z0-9]*$", profile_name):
        return profile_name
    raise argparse.ArgumentTypeError("%s is an invalid profile name value" %
 profile_name)

def validation_version(version_name):
    '''
    verify profile name doesn't have path to files or unexpected input
    '''
    if re.match(r"[1-2].\d.\d", version_name):
        return version_name
    raise argparse.ArgumentTypeError("%s is an invalid version name value" %
 version_name)

def validation_execution_role(execution_role_arn):
    '''
    verify profile name doesn't have path to files or unexpected input
    '''
    if re.match(r'(?i)\b((?:[a-z][\w-]+:(?:/{1,3}|[a-z0-9%])|www\d{0,3}[.]|[a-z0-9.
\-]+[.][a-z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|
(\([^\s()<>]+\)))*\)|[^\s`!()\[\]{};:\'".,<>?«»“”‘’]))', execution_role_arn):
        return execution_role_arn
    raise argparse.ArgumentTypeError("%s is an invalid execution role ARN" %
 execution_role_arn)

def create_new_env(env):
    '''
    method to duplicate env
    '''
    mwaa = boto3.client('mwaa', region_name=REGION)

    print('Source Environment')
    print(env)
    if (env['AirflowVersion']=="1.10.12") and (VERSION=="2.2.2"):
        if env['AirflowConfigurationOptions']
['secrets.backend']=='airflow.contrib.secrets.aws_secrets_manager.SecretsManagerBackend':
            print('swapping',env['AirflowConfigurationOptions']['secrets.backend'])
            env['AirflowConfigurationOptions']
['secrets.backend']='airflow.providers.amazon.aws.secrets.secrets_manager.SecretsManagerBackend
    env['LoggingConfiguration']['DagProcessingLogs'].pop('CloudWatchLogGroupArn')
    env['LoggingConfiguration']['SchedulerLogs'].pop('CloudWatchLogGroupArn')
    env['LoggingConfiguration']['TaskLogs'].pop('CloudWatchLogGroupArn')
```

```
    env['LoggingConfiguration']['WebserverLogs'].pop('CloudWatchLogGroupArn')
    env['LoggingConfiguration']['WorkerLogs'].pop('CloudWatchLogGroupArn')
    env['AirflowVersion']=VERSION
    env['ExecutionRoleArn']=EXECUTION_ROLE_ARN
    env['Name']=ENV_NAME_NEW
    env.pop('Arn')
    env.pop('CreatedAt')
    env.pop('LastUpdate')
    env.pop('ServiceRoleArn')
    env.pop('Status')
    env.pop('WebserverUrl')
    if not env['Tags']:
        env.pop('Tags')
    print('Destination Environment')
    print(env)

    return mwaa.create_environment(**env)

def get_mwaa_env(input_env_name):

    # https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
mwaa.html#MWAA.Client.get_environment
    mwaa = boto3.client('mwaa', region_name=REGION)
    environment = mwaa.get_environment(
        Name=input_env_name
    )['Environment']

    return environment

def print_err_msg(c_err):
    '''short method to handle printing an error message if there is one'''
    print('Error Message: {}'.format(c_err.response['Error']['Message']))
    print('Request ID: {}'.format(c_err.response['ResponseMetadata']['RequestId']))
    print('Http code: {}'.format(c_err.response['ResponseMetadata']['HTTPStatusCode']))

#
# Main
#
# Usage:
# python3 clone_environment.py --envname MySourceEnv --envnamenew MyDestEnv --region
 us-west-2 --execution_role AmazonMWAA-MyDestEnv-ExecutionRole --version 2.2.2
#
# based on https://github.com/awslabs/aws-support-tools/blob/master/MWAA/verify_env/
verify_env.py
```

```
#

if __name__ == '__main__':
    if sys.version_info[0] < 3:
        print("python2 detected, please use python3. Will try to run anyway")
    if not verify_boto3(boto3.__version__):
        print("boto3 version ", boto3.__version__, "is not valid for this script. Need
 1.17.80 or higher")
        print("please run pip install boto3 --upgrade --user")
        sys.exit(1)
    parser = argparse.ArgumentParser()
    parser.add_argument('--envname', type=validate_envname, required=True, help="name
 of the source MWAA environment")
    parser.add_argument('--region', type=validation_region,
 default=boto3.session.Session().region_name,
                        required=False, help="region, Ex: us-east-1")
    parser.add_argument('--profile', type=validation_profile, default=None,
                        required=False, help="AWS CLI profile, Ex: dev")
    parser.add_argument('--version', type=validation_version, default="2.2.2",
                        required=False, help="Airflow destination version, Ex: 2.2.2")
    parser.add_argument('--execution_role', type=validation_execution_role,
 default=None,
                        required=True, help="New environment execution role ARN, Ex:
 arn:aws:iam::112233445566:role/service-role/AmazonMWAA-MyEnvironment-ExecutionRole")
    parser.add_argument('--envnamenew', type=validate_envname, required=True,
 help="name of the destination MWAA environment")

    args, _ = parser.parse_known_args()
    ENV_NAME = args.envname
    REGION = args.region
    PROFILE = args.profile
    VERSION = args.version
    EXECUTION_ROLE_ARN = args.execution_role
    ENV_NAME_NEW = args.envnamenew

    try:
        print("PROFILE",PROFILE)
        if PROFILE:
            boto3.setup_default_session(profile_name=PROFILE)
        env = get_mwaa_env(ENV_NAME)
        response = create_new_env(env)
        print(response)
    except ClientError as client_error:
        if client_error.response['Error']['Code'] == 'LimitExceededException':
```

```
            print_err_msg(client_error)
            print('please retry the script')
        elif client_error.response['Error']['Code'] in ['AccessDeniedException',
'NotAuthorized']:
            print_err_msg(client_error)
            print('please verify permissions used have permissions documented in
readme')
        elif client_error.response['Error']['Code'] == 'InternalFailure':
            print_err_msg(client_error)
            print('please retry the script')
        else:
            print_err_msg(client_error)
    except ProfileNotFound as profile_not_found:
        print('profile', PROFILE, 'does not exist; check the profile name')
    except IndexError as error:
        print("Error:", error)
```

Langkah kedua: Migrasikan sumber daya alur kerja Anda

Apache Airflow v2 adalah rilis versi utama. Jika Anda bermigrasi dari Apache Airflow v1, Anda harus menyiapkan sumber daya alur kerja Anda dan memverifikasi perubahan yang Anda buat pada, persyaratan, dan plugin Anda. DAGs Untuk melakukannya, kami sarankan untuk mengonfigurasi versi jembatan Apache Airflow di sistem operasi lokal Anda menggunakan Docker dan pelari lokal Amazon MWAA. Pelari lokal Amazon MWAA menyediakan utilitas antarmuka baris perintah (CLI) yang mereplikasi lingkungan Amazon MWAA secara lokal.

Setiap kali Anda mengubah versi Apache Airflow, pastikan Anda mereferensikan URL yang `--constraint` benar di situs Anda. `requirements.txt`

Untuk memigrasikan sumber daya alur kerja

1. Buat fork aws-mwaa-local-runnerrepositori, dan kloning salinan pelari lokal Amazon MWAA.
2. Periksa `v1.10.15` cabang aws-mwaa-local-runner repositori. Apache Airflow merilis v1.10.15 sebagai rilis jembatan untuk membantu migrasi ke Apache Airflow v2, dan meskipun Amazon MWAA tidak mendukung v1.10.15, Anda dapat menggunakan pelari lokal Amazon MWAA untuk menguji sumber daya Anda.
3. Gunakan alat CLI runner lokal Amazon MWAA untuk membuat image Docker dan menjalankan Apache Airflow secara lokal. Untuk informasi selengkapnya, lihat README runner lokal di repositori. GitHub

4. Menggunakan Apache Airflow berjalan secara lokal, ikuti langkah-langkah yang dijelaskan dalam Upgrade dari 1.10 ke 2 di situs dokumentasi Apache Airflow.

    a. Untuk memperbarui`requirements.txt`, ikuti praktik terbaik yang kami rekomendasikan dalam Mengelola dependensi Python, di Panduan Pengguna Amazon MWAA.

    b. Jika Anda telah menggabungkan operator dan sensor khusus Anda dengan plugin Anda untuk lingkungan Apache Airflow v1.10.12 yang ada, pindahkan ke folder DAG Anda. Untuk informasi lebih lanjut tentang praktik terbaik manajemen modul untuk Apache Airflow v2+, lihat Manajemen Modul di situs web dokumentasi Apache Airflow.

5. Setelah Anda membuat perubahan yang diperlukan pada sumber daya alur kerja Anda, periksa `v2.5.1` cabang aws-mwaa-local-runner repositori, dan uji alur kerja, persyaratan DAGs, dan plugin kustom Anda yang diperbarui secara lokal. Jika Anda bermigrasi ke versi Apache Airflow yang berbeda, Anda dapat menggunakan cabang runner lokal yang sesuai untuk versi Anda.

6. Setelah berhasil menguji sumber daya alur kerja, salin DAGs`requirements.txt`, dan plugin ke bucket Amazon S3 yang dikonfigurasi dengan lingkungan Amazon MWAA baru.

Langkah ketiga: Mengekspor metadata dari lingkungan Anda yang ada

Tabel metadata Apache Airflow `dag` seperti`dag_tag`,, `dag_code` dan secara otomatis terisi saat Anda menyalin file DAG yang diperbarui ke bucket Amazon S3 lingkungan Anda dan penjadwal menguraikannya. Tabel terkait izin juga terisi secara otomatis berdasarkan izin peran eksekusi IAM Anda. Anda tidak perlu memigrasikan mereka.

Anda dapat memigrasikan data yang terkait dengan riwayat DAG`variable`,`slot_pool`,`sla_miss`,, dan jika diperlukan, `xcom`job, dan `log` tabel. Log contoh tugas disimpan dalam CloudWatch Log di bawah grup `airflow-`*`{environment_name}`* log. Jika Anda ingin melihat log instance tugas untuk proses yang lebih lama, log tersebut harus disalin ke grup log lingkungan baru. Kami menyarankan Anda memindahkan log hanya beberapa hari untuk mengurangi biaya terkait.

Jika Anda bermigrasi dari lingkungan Amazon MWAA yang ada, tidak ada akses langsung ke database metadata. Anda harus menjalankan DAG untuk mengekspor metadata dari lingkungan Amazon MWAA yang ada ke bucket Amazon S3 pilihan Anda. Langkah-langkah berikut juga dapat digunakan untuk mengekspor metadata Apache Airflow jika Anda bermigrasi dari lingkungan yang dikelola sendiri.

Setelah data diekspor, Anda kemudian dapat menjalankan DAG di lingkungan baru Anda untuk mengimpor data. Selama proses ekspor dan impor, semua lainnya DAGs dijeda.

Untuk mengekspor metadata dari lingkungan yang ada

1. Buat bucket Amazon S3 menggunakan AWS CLI untuk menyimpan data yang diekspor. Ganti `UUID` dan `region` dengan informasi Anda.

```
aws s3api create-bucket \
--bucket mwaa-migration-{UUID}\
--region {region}
```

> **Note**
>
> Jika Anda memigrasikan data sensitif, seperti koneksi yang disimpan dalam variabel, sebaiknya aktifkan enkripsi default untuk bucket Amazon S3.

2.

> **Note**
>
> Tidak berlaku untuk migrasi dari lingkungan yang dikelola sendiri.

Ubah peran eksekusi lingkungan yang ada dan tambahkan kebijakan berikut untuk memberikan akses tulis ke bucket yang Anda buat di langkah pertama.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*"
            ],
            "Resource": [
                "arn:aws:s3:::mwaa-migration-{UUID}/*"
            ]
        }
```

```
    ]
}
```

3. Kloning amazon-mwaa-examplesrepositori, dan arahkan ke `metadata-migration` subdirektori untuk skenario migrasi Anda.

```
git clone https://github.com/aws-samples/amazon-mwaa-examples.git
cd amazon-mwaa-examples/usecases/metadata-migration/existing-version-new-version/
```

4. Di`export_data.py`, ganti nilai string S3_BUCKET dengan bucket Amazon S3 yang Anda buat untuk menyimpan metadata yang diekspor.

```
S3_BUCKET = 'mwaa-migration-{UUID}'
```

5. Temukan `requirements.txt` file di `metadata-migration` direktori. Jika Anda sudah memiliki file persyaratan untuk lingkungan yang ada, tambahkan persyaratan tambahan yang ditentukan `requirements.txt` ke file Anda. Jika Anda tidak memiliki file persyaratan yang ada, Anda cukup menggunakan yang disediakan di `metadata-migration` direktori.
6. Salin `export_data.py` ke direktori DAG bucket Amazon S3 yang terkait dengan lingkungan Anda yang ada. Jika bermigrasi dari lingkungan yang dikelola sendiri, salin `export_data.py` ke folder Anda`/dags`.
7. Salin pembaruan Anda `requirements.txt` ke bucket Amazon S3 yang terkait dengan lingkungan yang ada, lalu edit lingkungan untuk menentukan versi baru`requirements.txt`.
8. Setelah lingkungan diperbarui, akses UI Apache Airflow, unpause DAG, dan picu alur `db_export` kerja untuk berjalan.
9. Verifikasi bahwa metadata diekspor ke `data/migration/`*`existing-version`*`_to_`*`new-version`*`/export/` bucket `mwaa-migration-`*`{UUID}`* Amazon S3, dengan setiap tabel dalam file khusus itu sendiri.

Langkah empat: Mengimpor metadata ke lingkungan baru Anda

Untuk mengimpor metadata ke lingkungan baru Anda

1. Di`import_data.py`, ganti nilai string untuk yang berikut ini dengan informasi Anda.
   - Untuk migrasi dari lingkungan Amazon MWAA yang ada:

```
S3_BUCKET = 'mwaa-migration-{UUID}'
```

```
        OLD_ENV_NAME='{old_environment_name}'
        NEW_ENV_NAME='{new_environment_name}'
        TI_LOG_MAX_DAYS = {number_of_days}
```

MAX_DAYSmengontrol berapa hari file log yang disalin alur kerja ke lingkungan baru.

- Untuk migrasi dari lingkungan yang dikelola sendiri:

```
S3_BUCKET = 'mwaa-migration-{UUID}'
        NEW_ENV_NAME='{new_environment_name}'
```

2. (Opsional) hanya `import_data.py` menyalin log tugas yang gagal. Jika Anda ingin menyalin semua log tugas, memodifikasi `getDagTasks` fungsi, dan menghapus `ti.state = 'failed'` seperti yang ditunjukkan dalam cuplikan kode berikut.

```
def getDagTasks():
     session = settings.Session()
     dagTasks = session.execute(f"select distinct ti.dag_id, ti.task_id,
 date(r.execution_date) as ed \
     from task_instance ti, dag_run r where r.execution_date > current_date -
 {TI_LOG_MAX_DAYS} and \
     ti.dag_id=r.dag_id and ti.run_id = r.run_id order by ti.dag_id,
 date(r.execution_date);").fetchall()
     return dagTasks
```

3. Ubah peran eksekusi lingkungan baru Anda dan tambahkan kebijakan berikut. Kebijakan izin memungkinkan Amazon MWAA membaca dari bucket Amazon S3 tempat Anda mengekspor metadata Apache Airflow, dan menyalin log instance tugas dari grup log yang ada. Ganti semua placeholder dengan informasi Anda.

> **Note**
>
> Jika Anda bermigrasi dari lingkungan yang dikelola sendiri, Anda harus menghapus izin terkait CloudWatch Log dari kebijakan.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
             "arn:aws:logs:us-east-1:111122223333:log-
group:airflow-{old_environment_name}*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::mwaa-migration-{UUID}",
                "arn:aws:s3:::mwaa-migration-{UUID}/*"
            ]
        }
    ]
}
```

4. Salin `import_data.py` ke direktori DAG bucket Amazon S3 yang terkait dengan lingkungan baru Anda, lalu akses UI Apache Airflow untuk menghentikan jeda DAG dan memicu alur kerja`db_import`. DAG baru akan muncul di Apache Airflow UI dalam beberapa menit.
5. Setelah DAG run selesai, verifikasi bahwa riwayat lari DAG Anda disalin dengan mengakses setiap DAG individu.

Langkah selanjutnya

- Untuk informasi selengkapnya tentang kelas dan kemampuan lingkungan Amazon MWAA yang tersedia, lihat kelas lingkungan Amazon MWAA di Panduan Pengguna Amazon MWAA.
- Untuk informasi selengkapnya tentang cara Amazon MWAA menangani pekerja penskalaan otomatis, lihat penskalaan otomatis Amazon MWAA di Panduan Pengguna Amazon MWAA.
- Untuk informasi selengkapnya tentang Amazon MWAA REST API, lihat Amazon MWAA REST API.

Migrasikan beban kerja dari ke AWS Data Pipeline Amazon MWAA

AWS meluncurkan AWS Data Pipeline layanan pada tahun 2012. Pada saat itu, pelanggan menginginkan layanan yang memungkinkan mereka menggunakan berbagai opsi komputasi untuk memindahkan data antara sumber data yang berbeda. Karena kebutuhan transfer data berubah dari waktu ke waktu, begitu juga solusi untuk kebutuhan tersebut. Anda sekarang memiliki opsi untuk memilih solusi yang paling sesuai dengan kebutuhan bisnis Anda. Anda dapat memigrasikan beban kerja Anda ke salah satu layanan berikut: AWS

- Gunakan Alur Kerja Terkelola Amazon untuk Apache Airflow (Amazon MWAA) untuk mengelola orkestrasi alur kerja untuk Apache Airflow.
- Gunakan Step Functions untuk mengatur alur kerja antara beberapa. Layanan AWS
- Gunakan AWS Glue untuk menjalankan dan mengatur aplikasi Apache Spark.

Opsi yang Anda pilih tergantung pada beban kerja Anda saat ini. AWS Data Pipeline Topik ini menjelaskan cara bermigrasi dari AWS Data Pipeline ke Amazon MWAA.

Topik

Memilih Amazon MWAA

Amazon Managed Workflows for Apache Airflow (Amazon MWAA) adalah layanan orkestrasi terkelola untuk Apache Airflow yang memungkinkan Anda mengatur dan mengoperasikan pipeline data di cloud dalam skala besar. end-to-end Apache Airflow adalah alat sumber terbuka yang digunakan untuk secara terprogram membuat, menjadwalkan, dan memantau urutan proses dan tugas yang disebut sebagai alur kerja. Dengan Amazon MWAA, Anda dapat menggunakan Apache Airflow dan bahasa pemrograman Python untuk membuat alur kerja tanpa harus mengelola

infrastruktur dasar untuk skalabilitas, ketersediaan, dan keamanan. Amazon MWAA secara otomatis menskalakan kapasitas alur kerjanya untuk memenuhi kebutuhan Anda, dan terintegrasi dengan layanan AWS keamanan untuk membantu memberi Anda akses cepat dan aman ke data Anda.

Berikut ini menyoroti beberapa manfaat bermigrasi dari AWS Data Pipeline ke Amazon MWAA:

- Peningkatan skalabilitas dan kinerja - Amazon MWAA menyediakan kerangka kerja yang fleksibel dan dapat diskalakan untuk menentukan dan mengeksekusi alur kerja. Hal ini memungkinkan pengguna untuk menangani alur kerja yang besar dan kompleks dengan mudah, dan memanfaatkan fitur seperti penjadwalan tugas dinamis, alur kerja berbasis data, dan paralelisme.
- Pemantauan dan pencatatan yang ditingkatkan - Amazon MWAA terintegrasi dengan Amazon CloudWatch untuk meningkatkan pemantauan dan pencatatan alur kerja Anda. Amazon MWAA secara otomatis mengirimkan metrik dan log sistem ke. CloudWatch Ini berarti Anda dapat melacak kemajuan dan kinerja alur kerja Anda secara real-time, dan mengidentifikasi masalah apa pun yang muncul.
- Integrasi yang lebih baik dengan AWS layanan dan perangkat lunak pihak ketiga — Amazon MWAA terintegrasi dengan berbagai AWS layanan lain, seperti Amazon S3,, dan AWS Glue Amazon Redshift, serta perangkat lunak pihak ketiga seperti DBT, Snowflake, dan Databricks. Ini memungkinkan Anda memproses, dan mentransfer, data di berbagai lingkungan dan layanan.
- Alat pipa data sumber terbuka — Amazon MWAA memanfaatkan produk Apache Airflow sumber terbuka yang sama dengan yang Anda kenal. Apache Airflow adalah alat yang dibuat khusus yang dirancang untuk menangani semua aspek manajemen pipa data, termasuk konsumsi, pemrosesan, transfer, pengujian integritas, pemeriksaan kualitas, dan memastikan garis keturunan data.
- Arsitektur modern dan fleksibel — Amazon MWAA memanfaatkan kontainerisasi dan teknologi cloud-native, tanpa server. Ini berarti lebih banyak fleksibilitas dan portabilitas, serta penyebaran dan pengelolaan lingkungan alur kerja Anda yang lebih mudah.

Arsitektur dan pemetaan konsep

AWS Data Pipeline dan Amazon MWAA memiliki arsitektur dan komponen yang berbeda, yang dapat memengaruhi proses migrasi dan cara alur kerja didefinisikan dan dijalankan. Bagian ini meninjau arsitektur dan komponen untuk kedua layanan, dan menyoroti beberapa perbedaan utama.

Keduanya AWS Data Pipeline dan Amazon MWAA adalah layanan yang dikelola sepenuhnya. Saat memigrasikan beban kerja ke Amazon MWAA, Anda mungkin perlu mempelajari konsep baru untuk

memodelkan alur kerja yang ada menggunakan Apache Airflow. Namun, Anda tidak perlu mengelola infrastruktur, menambal pekerja, dan mengelola pembaruan sistem operasi.

Tabel berikut mengaitkan konsep-konsep kunci AWS Data Pipeline dengan yang ada di Amazon MWAA. Gunakan informasi ini sebagai titik awal untuk merancang rencana migrasi.

Konsep	AWS Data Pipeline	Amazon MWAA
Definisi pipa	AWS Data Pipeline menggunakan file konfigurasi berbasis JSON yang mendefinisikan alur kerja.	Amazon MWAA menggunakan Directed Acyclic Graphs () berbasis Python yang menentukan alur kerja. DAGs
Lingkungan eksekusi pipa	Alur kerja berjalan di instans Amazon EC2. AWS Data Pipeline menyediakan dan mengelola instans ini atas nama Anda.	Amazon MWAA menggunakan lingkungan kontainer Amazon ECS untuk menjalankan tugas.
Komponen pipa	Aktivitas adalah memproses tugas yang berjalan sebagai bagian dari alur kerja.	Operator (Tugas) adalah unit pemrosesan dasar dari alur kerja.
	Prasyarat berisi pernyataan kondisional yang harus benar sebelum suatu aktivitas dapat dijalankan.	Sensor (Tugas) mewakili pernyataan bersyarat yang dapat menunggu sumber daya atau tugas diselesaikan sebelum dijalankan.
	Sumber daya AWS Data Pipeline mengacu pada sumber daya AWS komputasi yang melakukan pekerjaan yang ditentukan oleh aktivitas pipeline. Amazon EC2 dan Amazon EMR adalah dua sumber daya yang tersedia.	Menggunakan tugas dalam DAG, Anda dapat menentukan berbagai sumber daya komputasi, termasuk Amazon ECS, Amazon EMR, dan Amazon EKS. Amazon MWAA menjalankan operasi Python pada pekerja yang berjalan di Amazon ECS.

Konsep	AWS Data Pipeline	Amazon MWAA
Eksekusi alur	AWS Data Pipeline mendukung penjadwalan berjalan dengan pola berbasis tarif reguler, dan berbasis cron.	Amazon MWAA mendukung penjadwalan dengan ekspresi cron dan preset, serta jadwal khusus.
	Sebuah instance mengacu pada setiap proses pipa.	DAG run mengacu pada setiap proses alur kerja Apache Airflow.
	Upaya mengacu pada percobaan ulang operasi yang gagal.	Amazon MWAA mendukung percobaan ulang yang Anda tentukan baik di level DAG, atau di tingkat tugas.

Contoh implementasi

Dalam banyak kasus, Anda akan dapat menggunakan kembali sumber daya yang saat ini Anda atur setelah AWS Data Pipeline bermigrasi ke Amazon MWAA. Daftar berikut berisi contoh implementasi menggunakan Amazon MWAA untuk kasus penggunaan yang paling umum AWS Data Pipeline .

- Menjalankan pekerjaan EMR Amazon (lokakarya)AWS
- Membuat plugin khusus untuk Apache Hive dan Hadoop (Panduan Pengguna Amazon MWAA)
- Menyalin data dari S3 ke Redshift (bengkel)AWS
- Menjalankan skrip shell pada instance Amazon ECS jarak jauh (Panduan Pengguna Amazon MWAA)
- Mengatur alur kerja hybrid (on-prem) (Posting blog)

Untuk tutorial dan contoh tambahan, lihat yang berikut ini:

- Tutorial Amazon MWAA
- Contoh kode Amazon MWAA

Perbandingan harga

Harga untuk AWS Data Pipeline didasarkan pada jumlah pipa, serta seberapa banyak Anda menggunakan setiap pipa. Aktivitas yang Anda jalankan lebih dari sekali sehari (frekuensi tinggi) berharga $1 per bulan per aktivitas. Aktivitas yang Anda jalankan sekali sehari atau kurang (frekuensi rendah) berharga $0,60 per bulan per aktivitas. Pipa Tidak Aktif dihargai $1 per pipa. Untuk informasi lebih lanjut, lihat halaman AWS Data Pipeline harga.

Harga untuk Amazon MWAA didasarkan pada durasi waktu lingkungan Apache Airflow terkelola Anda, dan penskalaan otomatis tambahan apa pun yang diperlukan untuk menyediakan lebih banyak pekerja, atau kapasitas penjadwal. Anda membayar untuk penggunaan lingkungan Amazon MWAA Anda setiap jam (ditagih pada resolusi satu detik), dengan biaya yang bervariasi tergantung pada ukuran lingkungan. Amazon MWAA secara otomatis menskalakan jumlah pekerja berdasarkan konfigurasi lingkungan Anda. AWS menghitung biaya pekerja tambahan secara terpisah. Untuk informasi lebih lanjut tentang biaya per jam menggunakan berbagai ukuran lingkungan Amazon MWAA, lihat halaman harga Amazon MWAA.

Sumber daya terkait

Untuk informasi selengkapnya dan praktik terbaik untuk menggunakan Amazon MWAA, lihat sumber daya berikut:

- Referensi API Amazon MWAA
- Memantau dasbor dan alarm di Amazon MWAA
- Penyetelan kinerja untuk Apache Airflow di Amazon MWAA

Riwayat Dokumen Amazon MWAA

Tabel berikut menjelaskan penambahan penting pada panduan migrasi Amazon MWAA, mulai Maret 2022.

Perubahan	Deskripsi	Tanggal
Topik baru tentang migrasi beban kerja dari ke AWS Data Pipeline Amazon MWAA	Menambahkan informasi dan panduan baru tentang migrasi beban kerja yang ada dari AWS Data Pipeline Amazon MWAA. Gunakan informasi ini untuk membantu Anda merancang rencana migrasi. • Migrasikan beban kerja dari ke AWS Data Pipeline Amazon MWAA	April 14, 2023
Peluncuran Panduan Migrasi Amazon MWAA	Amazon MWAA sekarang menawarkan panduan terperinci tentang migrasi ke lingkungan Amazon MWAA baru. Langkah-langkah yang dijelaskan dalam Panduan Migrasi MWAA Amazon berlaku untuk mgirating dari lingkungan Amazon MWAA yang ada, atau dari penerapan Apache Airflow yang dikelola sendiri. • Tentang panduan migrasi Amazon MWAA	7 Maret 2022

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.