aws

移行ガイド

Amazon Managed Workflows for Apache Airflow

Amazon Managed Workflows for Apache Airflow: 移行ガイド

Table of Contents

Amazon MWAA 移行ガイドとは何ですか？

Amazon Managed Workflows for Apache Airflowは、Apache Airflow 用のマネージドオーケストレーション・サービスで、クラウド上でデータ・パイプラインを大規模に運用することができます。Amazon MWAA が Apache Airflow のプロビジョニングと継続的なメンテナンスを管理するため、インスタンスのパッチ適用、スケーリング、セキュリティ保護について心配する必要がありません。

Amazon MWAA は、タスクを実行するコンピューティングリソースを自動的にスケールして、オンデマンドで一貫したパフォーマンスを提供します。Amazon MWAA はデフォルトでデータを保護します。ワークロードは、Amazon Virtual Private Cloud を使用して、お客様独自の隔離された安全なクラウド環境で実行されます。これにより、データが AWS Key Management Service で自動的に暗号化されます。

このガイドを使用して、自己管理型の Apache Airflow ワークフローを Amazon MWAA に移行するか、既存の Amazon MWAA 環境を新しい Apache Airflow バージョンにアップグレードしてください。移行チュートリアルでは、新しい Amazon MWAA 環境を作成または複製し、ワークフローリソースを移行し、ワークフローのメタデータとログを新しい環境に転送する方法について説明します。

移行チュートリアルを開始する前に、以下のトピックを確認することをお勧めします。

- ネットワークアーキテクチャ
- 主な考慮事項

Amazon MWAA ネットワークアーキテクチャを試す

次のセクションでは、Amazon MWAA 環境を構成する主要なコンポーネント及び、各環境がそのリソースを管理し、データの安全性の確保、ワークフローに対する監視と可視性を提供するために統合される AWS のサービスセットについて説明します。

トピック

Amazon MWAA コンポーネント

Amazon MWAA 環境は、次の 4 つの主要コンポーネントで構成されます。

1. スケジューラー — すべての DAG を解析及び監視し、DAG の依存関係が満たされた場合に実行するタスクをキューに入れます。Amazon MWAA は、スケジューラーを少なくとも 2 つのスケジューラーを持つ AWS Fargate クラスターとしてデプロイします。ワークロードに応じて、スケジューラーの数を最大 5 つまで増やすことができます。Amazon MWAA 環境クラスの詳細については、Amazon MWAA 環境クラス を参照してください。
2. ワーカー — スケジュールされたタスクを実行する 1 つ以上の Fargate タスク。環境のワーカー数は、指定した 最小 と 最大 数の間の範囲によって決まります。Amazon MWAA は、キューに入れられたタスクと実行中のタスクの数が既存のワーカーが処理できる数を超えると、自動スケーリングワーカーを開始します。実行中のタスクとキューに入れられたタスクの合計が 2 分を超えてゼロになると、Amazon MWAA はワーカーの数を最小値まで縮小します。Amazon MWAA が自動スケーリングワーカーを処理する方法の詳細については、Amazon MWAA 自動スケーリング を参照してください。
3. ウェブサーバー — Apache Airflow ウェブ UI を実行します。プライベートまたはパブリックネットワークアクセスを使用してウェブサーバーを設定できます。いずれの場合も、Apache Airflow ユーザーへのアクセスは AWS Identity and Access Management (IAM) で定義したアクセス制御ポリシーによって制御されます。環境に合わせた IAM アクセスポリシーの設定の詳細については、Amazon MWAA 環境へのアクセス を参照してください。
4. データベース — DAG 実行履歴を含む、Apache Airflow 環境とワークフローに関するメタデータを保存します。データベースは、AWS によって管理されるシングルテナントのAurora PostgreSQLデータベースであり、プライベートで保護されたAmazon VPC エンドポイントを介してスケジューラーとワーカーの Fargate コンテナにアクセスできます。

すべての Amazon MWAA 環境は、DAG とタスクの依存関係の保存とアクセス、保存中のデータの保護、環境のログ記録とモニタリングなど、さまざまなタスクを処理する一連の AWS サービスと相互作用します。次の図は、Amazon MWAA 環境のさまざまなコンポーネントを示しています。

> **Note**
>
> Amazon VPC サービスは共有 VPC ではありません。Amazon MWAAは、作成する環境ごとにAWSの専有VPCを作成します。

- Amazon S3 — Amazon MWAA は、DAG、要件、プラグインファイルなど、すべてのワークフローリソースを Amazon S3 バケットに保存します。環境作成の一環としてバケットを作成し、Amazon MWAA リソースをアップロードする方法の詳細については、Amazon MWAA ユーザーガイド の Amazon MWAA 用の Amazon S3 バケットの作成 を参照してください。
- Amazon SQS — Amazon MWAA は Amazon SQS を使用して、Celery エグゼキューターを用いワークフロータスクをキューイングします。
- Amazon ECR — Amazon ECR はすべてのApache Airflow イメージをホストします。Amazon MWAA は、AWS の管理された Apache Airflow イメージのみをサポートしています。
- AWS KMS — Amazon MWAA は、AWS KMS を使用してデータが安全に静的に保管されることを確認します。デフォルトでは、Amazon MWAAは、 AWSで管理された AWS KMS 鍵 を使用しますが、環境を構成して独自の カスタマーマネージド AWS KMSキーを使用することもできます。独自のカスタマーマネージド AWS KMS キーの使用に関する詳細については、Amazon MWAA ユーザーガイド の データ暗号化用のカスタマーマネージドキー を参照してください。
- CloudWatch — Amazon MWAA は CloudWatch と統合され、Apache Airflow ログと環境メトリクスをCloudWatch に送信します。これにより、Amazon MWAA リソースを監視し、問題をトラブルシューティングできるようになります。

接続

Amazon MWAA 環境は、統合されているすべてのAWSサービスにアクセスする必要があります。Amazon MWAA 実行ロール は、ユーザーに代わって他の AWS サービスに接続するためのアクセス権限を Amazon MWAA に付与する方法を制御します。ネットワーク接続については、Amazon VPC へのパブリックインターネットアクセスを提供するか、Amazon VPC エンドポイントを作成できます。環境に合わせた Amazon VPC エンドポイント (AWS PrivateLink) の設定の詳細について

は、Amazon MWAA ユーザーガイド の Amazon MWAA での VPC エンドポイントへのアクセスの管理 を参照してください。

Amazon MWAA はスケジューラーとワーカーに要件をインストールします。要件がパブリック PyPi リポジトリからのものである場合、必要なライブラリをダウンロードするには、環境がインターネットに接続する必要があります。プライベート環境では、プライベート PyPI リポジトリを使用するか、ライブラリを環境のカスタムプラグインとして `.whl`ファイル にバンドルできます。

Apache Airflow を プライベートモード で設定すると、Apache Airflow UI には Amazon VPC エンドポイントを介してのみ Amazon VPC からアクセスできます。

ネットワークの詳細については、Amazon MWAA ユーザーガイド の ネットワーク を参照してください。

新しい MWAA 環境への移行に関する主な考慮事項

Apache Airflow ワークロードを Amazon MWAA に移行する計画を立てる際に、認証や Amazon MWAA 実行ロールなどの重要な考慮事項について詳しく学んでください。

トピック

認証

Amazon MWAA は AWS Identity and Access Management (IAM) を使用して Apache Airflow UI へのアクセスを制御します。ウェブサーバーにアクセスして DAG を管理する権限を Apache Airflow ユーザーに付与するための IAM ポリシーを作成して管理する必要があります。異なるアカウントで IAM を使用して、Apache Airflow の デフォルトロール の認証と認可の両方を管理できます。

カスタム Airflow ロールを作成して IAM プリンシパルにマッピングすることで、Apache Airflow ユーザーがワークフロー DAG のサブセットのみにアクセスするようにさらに管理および制限できます。詳細とステップごとのチュートリアルについては、チュートリアル: DAG のサブセットへの Amazon MWAA ユーザーのアクセスを制限する を参照してください。

Amazon MWAA にアクセスするようにフェデレーテッドアイデンティティを設定することもできます。詳細については、以下を参照してください。

- パブリックアクセスが可能なAmazon MWAA 環境 — AWS コンピュートブログ の Amazon MWAA で ID プロバイダーとして Okta を使用 します。
- プライベートアクセスによる Amazon MWAA 環境— フェデレーテッドアイデンティティを使用してプライベート Amazon MWAA 環境にアクセスします。

実行ロール

Amazon MWAA は、他の AWS サービスにアクセスするためのアクセス許可を環境に付与する実行ロールを使用します。関連するアクセス許可をロールに追加することで、ワークフローに AWS サービスへのアクセスを提供できます。初めて環境を作成するときに、新しい実行ロールを作成するデフォルトオプションを選択した場合、Amazon MWAA はロールに必要な最小限のアクセス許可を

ロールにアタッチします。ただし、Amazon MWAA がすべてのロググループを自動的に追加する CloudWatch Logs の場合を除きます。

実行ロールが作成されると、Amazon MWAA はユーザーに代わってそのアクセス権限ポリシーを管理できなくなります。実行ロールを更新するには、ポリシーを編集して必要に応じてアクセス許可を追加し、削除する必要があります。例えば、Amazon MWAA 環境をバックエンドとして AWS Secrets Manager を統合して、Apache Airflow ワークフローで使用するシークレットと接続文字列を安全に保存します。そのためには、以下のアクセス権限ポリシーを環境の実行ロールにアタッチします。

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": "*"
        }
    ]
}
```

他の AWS サービスとの統合も同様のパターンに従います。関連するアクセス許可ポリシーを Amazon MWAA 実行ロールに追加し、サービスにアクセスするためのアクセス許可を Amazon MWAA に付与します。Amazon MWAA 実行ロールの管理に関する詳細およびその他の例については、Amazon MWAA ユーザーガイドの Amazon MWAA 実行ロール を参照してください。

新しい Amazon MWAA 環境へ移行する

以下のトピックでは、既存の Apache Airflow ワークロードを新しい Amazon MWAA 環境に移行する手順を試します。これらの手順を使用して、古いバージョンの Amazon MWAA から新しいバージョンのリリースに移行したり、自己管理型の Apache Airflow デプロイを Amazon MWAA に移行したりできます。このチュートリアルでは、既存の Apache Airflow v1.10.12 から新しい Amazon MWAA に移行して Apache Airflow v2.5.1 を実行することを前提としていますが、同じ手順を使用して別の Apache Airflow バージョンから移行したり、あるいは別の Apache Airflow バージョンへ移行することもできます。

トピック

前提条件

手順を完了して環境を移行するには、以下が必要です。

- Apache Airflow デプロイ。これは自己管理型でも既存の Amazon MWAA 環境である可能性があります。
- ローカルシステムに Docker がインストールされています。
- AWS Command Line Interface バージョン 2 がインストールされました。

ステップ 1: サポートされている最新バージョンの Apache Airflow を実行する新しい Amazon MWAA 環境を作成します。

環境を作成するには、「Amazon MWAA ユーザーガイド」の「Amazon MWAA の開始方法」の詳細なステップを使用するか、 CloudFormation テンプレートを使用します。 既存の Amazon

MWAA 環境から移行し、 CloudFormation テンプレートを使用して古い環境を作成している場合は、 `AirflowVersion`プロパティを変更して新しいバージョンを指定できます。

```
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
```

または、既存の Amazon MWAA 環境から移行する場合、次の Python スクリプトをコピーして、AWS SDK の Python (Boto3) を使用して環境を複製できます。スクリプトをダウンロードすることもできます。

Python スクリプト

```
# 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)
```

ステップ 2: ワークフローリソースの移行

Apache Airflow v2 はメジャーバージョンのリリースです。Apache Airflow v1 から移行する場合は、ワークフローリソースを準備し、DAG、要件、プラグインに加えた変更を確認する必要があります。そのためには、Docker と Amazon MWAA ローカルランナー を使用して、ローカルオペレーティングシステムで Apache Airflow の ブリッジ バージョンを設定することをお勧めします。Amazon MWAA ローカルランナーには、Amazon MWAA 環境をローカルに複製するコマンドラインインターフェイス (CLI) ユーティリティが用意されています。

Apache Airflow のバージョンを変更するときは必ず、`--constraint` 内の正しい `requirements.txt` URL お客様するようにしてください。

ワークフローリソースを移行するには

1. aws-mwaa-local-runner リポジトリのフォークを作成し、Amazon MWAA ローカルランナーのコピーをクローンしてください。
2. aws-mwaa-local-runner リポジトリの `v1.10.15` ブランチをチェックアウトしてください。Apache Airflow は、Apache Airflow v2 への移行を支援するために、ブリッジリリースとして v1.10.15 をリリースしました。Amazon MWAA は v1.10.15 をサポートしていませんが、Amazon MWAA ローカルランナーを使用してリソースをテストできます。

3. Amazon MWAA ローカルランナー CLI ツールを使用して Docker イメージをビルドし、Apache Airflow をローカルで実行します。詳細については、GitHub リポジトリのローカル・ランナー README を参照してください。
4. ローカルで実行されている Apache Airflow を使用して、Apache Airflow ドキュメンテーションウェブサイトの 1.10 から 2 へのアップグレード で説明されている手順に従ってください。

    a. `requirements.txt` を更新するには、Amazon MWAA ユーザーガイド の Python 依存関係の管理 で推奨されているベストプラクティスに従ってください。

    b. 既存の Apache Airflow v1.10.12 環境のプラグインにカスタムオペレータとセンサーをバンドルしている場合は、それらを DAG フォルダに移動してください。Apache Airflow v2+ のモジュール管理のベストプラクティスについての詳細な情報については、Apache Airflow のドキュメンテーションウェブサイトの モジュール管理 を参照してください。
5. ワークフローリソースに必要な変更を加えたら、aws-mwaa-local-runner リポジトリの `v2.5.1` ブランチをチェックアウトし、更新したワークフロー DAG、要件、カスタムプラグインをローカルでテストしてください。別の Apache Airflow バージョンに移行する場合は、代わりにそのバージョンに適したローカルランナーブランチを使用できます。
6. ワークフローリソースのテストに成功したら、DAG、`requirements.txt`、およびプラグインを、新しい Amazon MWAA 環境で設定した Amazon S3 バケットにコピーします。

ステップ 3: 既存の環境からメタデータをエクスポートする

、などの Apache Airflow メタデータテーブル `dag` は `dag_tag`、`dag_code` 更新された DAG ファイルを環境の Amazon S3 バケットにコピーし、スケジューラーがそれらを解析すると自動的に入力されます。権限関連のテーブルも、IAM 実行ロールの権限に基づいて自動的に入力されます。移行する必要はありません。

必要であれば、DAG 履歴、`variable`、`slot_pool`、`sla_miss`、などのデータを移行できます。`xcom`、 `job`、および関連する `log` テーブル。タスクインスタンスのログは、CloudWatch Logs の `airflow-`*`{environment_name}`* ロググループの下に格納されています。古い実行のタスクインスタンスログを表示したい場合は、それらのログを新しい環境ロググループにコピーする必要があります。関連するコストを削減するために、数日分のログだけを移動することをおすすめします。

既存の Amazon MWAA 環境から移行する場合、メタデータデータベースには直接アクセスできません。既存のAmazon MWAA 環境からメタデータを Amazon S3 バケットにエクスポートするには、DAG を実行する必要があります。自己管理環境から移行する場合は、以下の手順を使用して Apache Airflow メタデータをエクスポートすることもできます。

データをエクスポートすると、新しい環境で DAG を実行し、データをインポートできます。エクスポートとインポートの処理中、他のすべての DAG は一時停止されます。

既存の環境からメタデータをエクスポートするには

1. を使用して Amazon S3 バケットを作成し AWS CLI 、エクスポートしたデータを保存します。UUID と `region` をお客様の情報に置き換えます。

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

> **Note**
>
> 変数に保存する接続などの機密データを移行する場合は、Amazon S3 バケットの デフォルト暗号化を有効にする ことをお勧めします。

2.

> **Note**
>
> 自己管理型の環境からの移行には適用されません。

既存の環境の実行ロールを変更し、次のポリシーを追加して、ステップ 1 で作成したバケットへの書き込みアクセスを許可します。

JSON

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

```
    ]
}
```

3. amazon-mwaa-examples リポジトリをクローンし、移行シナリオの metadata-migration サブディレクトリに移動してください。

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

4. export_data.py で、S3_BUCKET の文字列の値を、エクスポートしたメタデータを保存するために作成した Amazon S3 バケットに置き換えます。

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

5. requirements.txt ファイルを metadata-migration ディレクトリに配置してください。既存の環境の要件ファイルが既にある場合は、requirements.txt で指定されている追加の要件をファイルに追加してください。既存の要件ファイルがない場合は、metadata-migration ディレクトリにあるものを使用してください。
6. 既存の環境に関連付けられている Amazon S3 バケットの DAG ディレクトリに export_data.py をコピーします。自己管理環境から移行する場合は、/dags フォルダに export_data.py をコピーします。
7. 更新した requirements.txt を、既存の環境に関連付けられている Amazon S3 バケットにコピーし、環境を編集して新しい requirements.txt バージョンを指定します。
8. 環境が更新されたら、Apache Airflow UI にアクセスし、db_export DAG の一時停止を解除し、ワークフローをトリガーして実行してください。
9. メタデータが mwaa-migration-*{UUID}* Amazon S3 バケットの data/migration/*existing-version*_to_*new-version*/export/ にエクスポートされ、各テーブルが専用のファイルにあることを確認します。

ステップ 4: メタデータを新しい環境にインポートする

メタデータを新しい環境にインポートするには

1. import_data.py で、以下の文字列値を自分の情報に置き換えてください。
   - 既存の Amazon MWAA 環境からの移行の場合:

```
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_DAYS は、ワークフローが新しい環境にコピーするログファイルの日数を制御します。

- セルフマネージド環境からの移行の場合:

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

2. (オプション) import_data.py は、失敗したタスクログのみをコピーします。すべてのタスクログをコピーする場合は、以下のコードスニペットに示すように、getDagTasks 関数を変更して ti.state = 'failed' を削除してください。

```
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. 新しい環境の実行ロールを変更し、次のポリシーを追加します。アクセス権限ポリシーは、Apache Airflow メタデータをエクスポートした Amazon S3 バケットから Amazon MWAA が読み取り、既存のロググループからタスクインスタンスログをコピーできるようにします。すべてのプレースホルダーを自分の情報に置き換えます。

> **Note**
>
> 自己管理環境から移行する場合は、ポリシーから CloudWatch Logs 関連のアクセス許可を削除する必要があります。

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. 新しい環境に関連付けられた Amazon S3 バケットの`import_data.py` DAG ディレクトリにコピーし、Apache Airflow UI にアクセスして `db_import` DAG の一時停止を解除し、ワークフローをトリガーします。新しい DAG は、数分後に Apache Airflow UI に表示されます。
5. DAG の実行が完了したら、個々の DAG にアクセスして DAG の実行履歴がコピーされていることを確認します。

次の手順

- Amazon MWAA 環境クラスと機能の詳細については、Amazon MWAA ユーザーガイド の Amazon MWAA 環境クラス を参照してください。
- Amazon MWAA が自動スケーリングワーカーを処理する方法の詳細については、Amazon MWAA ユーザーガイド の Amazon MWAA 自動スケーリング を参照してください。
- Amazon MWAA REST API の詳細については、Amazon MWAA REST API を参照してください。

ワークロード AWS Data Pipeline を から Amazon MWAA に移行する

AWS は 2012 年に AWS Data Pipeline サービスを開始しました。当時、顧客は、さまざまなコンピューティングオプションを使用して、異なるデータソース間でデータを移動できるサービスを求めていました。データ転送のニーズは時間とともに変化するため、そのニーズに対応するソリューションも変化します。現在は、ビジネス要件に最も近いソリューションを選択できるようになりました。ワークロードは、次のいずれかのサービスに移行できます AWS 。

- Amazon Managed Workflows for Apache Airflow (Amazon MWAA) を使用して、Apache Airflow のワークフローオーケストレーションを管理します。
- Step Functionsを使用して、複数の AWS のサービス間でワークフローを組織化します。
- AWS Glue を使用して Apache Spark アプリケーションを実行およびオーケストレーションします。

選択するオプションは、 AWS Data Pipelineの現在のワークロードによって異なります。このトピックでは、 から Amazon MWAA AWS Data Pipeline に移行する方法について説明します。

トピック

Amazon MWAA を選択する

Amazon Managed Workflows for Apache Airflow (Amazon MWAA) は、Apache Airflow向けのマネージド・オーケストレーション・サービスで、クラウド上でエンドツーエンドのデータ・パイプラインを大規模にセットアップし、運用することができます。Apache Airflow は、ワークフロー と呼ばれる一連のプロセスやタスクをプログラムで作成、スケジュール設定、監視するために使用されるオープンソースのツールです。Amazon MWAAを使用すると、スケーラビリティ、可用性、セキュリティのための基盤を管理する必要なく、Apache Airflow と Python プログラミング言語を使用して

ワークフローを作成できます。Amazon MWAA は、ニーズに合わせてワークフロー容量を自動的にスケーリングし、 AWS セキュリティサービスと統合して、データへの迅速かつ安全なアクセスを提供します。

から Amazon MWAA に移行する利点のいくつか AWS Data Pipeline を以下に示します。

- スケーラビリティとパフォーマンスの強化 — Amazon MWAA は、ワークフローを定義して実行するための柔軟でスケーラブルなフレームワークを提供します。これにより、ユーザーは大規模で複雑なワークフローを簡単に処理でき、動的タスクスケジューリング、データ駆動型ワークフロー、並列処理などの機能を活用できます。
- モニタリングとロギングの向上 — Amazon MWAA は Amazon CloudWatch と統合され、ワークフローのモニタリングとロギングを強化します。Amazon MWAA は、システムメトリックスとログを CloudWatch に自動的に送信します。つまり、ワークフローの進行状況とパフォーマンスをリアルタイムで追跡し、発生した問題を特定できるということです。
- AWS サービスやサードパーティーソフトウェアとの統合の向上 – Amazon MWAA は、Amazon S3、Amazon Redshift AWS Glue、DBT、Snowflake、Databricks などのサードパーティーソフトウェアなどのさまざまな AWS サービスと統合されます。これにより、さまざまな環境やサービス間でデータを処理し、転送することができます。
- オープンソースのデータパイプラインツール — Amazon MWAA は、使い慣れたオープンソースの Apache Airflow 製品を活用しています。Apache Airflow は、取り込み、処理、転送、整合性テスト、品質チェック、データリネージの確認など、データパイプライン管理のあらゆる側面を処理するように設計された専用ツールです。
- モダンで柔軟なアーキテクチャ — Amazon MWAA はコンテナ化とクラウドネイティブなサーバーレステクノロジーを活用しています。つまり、柔軟性と移植性が向上し、ワークフロー環境のデプロイと管理が容易になります。

アーキテクチャとコンセプトのマッピング

AWS Data Pipeline と Amazon MWAA には異なるアーキテクチャとコンポーネントがあり、移行プロセスやワークフローの定義と実行方法に影響を与える可能性があります。このセクションでは、両方のサービスのアーキテクチャとコンポーネントの概要を説明し、主な相違点をいくつか強調します。

AWS Data Pipeline と Amazon MWAA はどちらもフルマネージドサービスです。ワークロードを Amazon MWAA に移行する場合、Apache Airflow を使用して既存のワークフローをモデル化するための新しい概念を学ぶ必要があるかもしれません。ただし、インフラストラクチャを管理したり、

ワーカーにパッチを適用したり、オペレーティングシステムの更新を管理したりする必要はありません。

次の表は、 の主要な概念 AWS Data Pipeline を Amazon MWAA の主要な概念と関連付けています。この情報を基にして移行計画を設計してください。

概念	AWS Data Pipeline	Amazon MWAA
パイプライン定義	AWS Data Pipeline は、ワークフローを定義する JSON ベースの設定ファイルを使用します。	Amazon MWAA は、ワークフローを定義する Python ベースの有向非循環グラフ (DAGs) を使用します。
パイプライン実行環境	ワークフローは Amazon EC2 instances. AWS Data Pipeline provisions で実行され、ユーザーに代わってこれらのインスタンスを管理します。	Amazon MWAA は Amazon ECS コンテナ環境を使用してタスクを実行します。
パイプラインコンポーネント	アクティビティ とは、ワークフローの一部として実行されるタスクを処理することです。	オペレータ (タスク) はワークフローの基本的な処理単位です。
	前提条件 には、アクティビティが実行される前に正でなければならない条件文が含まれます。	センサー (タスク) は、リソースまたはタスクが完了するのを待ってから実行できる条件ステートメントです。
	のリソース AWS Data Pipeline は、パイプラインアクティビティが指定する作業を実行する AWS コンピューティングリソースを指します。Amazon EC2 と Amazon EMR は、2 つのリソースで利用できます。	DAG 内のタスクを使用すると、Amazon ECS、Amazon EMR、Amazon EKS など、さまざまなコンピューティングリソースを定義できます。Amazon MWAA は、Amazon ECS 上で実行されるワーカーに対して Python

概念	AWS Data Pipeline	Amazon MWAA
		オペレーションを実行します。
パイプラインの実行	AWS Data Pipeline では、通常のレートベースおよび cron ベースのパターンで実行をスケジュールできます。	Amazon MWAA は、cron の式やプリセット、カスタムタイムテーブルによるスケジューリングをサポートしています。
	インスタンス とは、パイプラインの各実行を指します。	DAG 実行とは、Apache Airflow ワークフローの各実行を指します。
	試行 とは、失敗した操作を再試行することです。	Amazon MWAA は、DAG レベルまたはタスクレベルで定義した再試行をサポートします。

実装例

多くの場合、Amazon MWAA に移行 AWS Data Pipeline した後、現在オーケストレーションしているリソースを再利用できます。次のリストには、最も一般的な AWS Data Pipeline ユースケースで Amazon MWAA を使用する実装例が含まれています。

- Amazon EMR ジョブの実行 (AWS ワークショップ)
- Apache Hive と Hadoop 用のカスタムプラグインの作成(Amazon MWAA ユーザーガイド)
- S3 から Redshift へのデータのコピー (AWS workshop)
- リモート Amazon ECS インスタンスでのシェルスクリプトの実行 (Amazon MWAA ユーザーガイド)
- ハイブリッド (オンプレミス) ワークフローの調整 (ブログ記事)

その他のチュートリアルと例については、以下を参照してください。

- Amazon MWAA チュートリアル

- Amazon MWAA コード例

価格比較

の料金は AWS Data Pipeline 、パイプラインの数と各パイプラインの使用量に基づきます。1 日に 2 回以上 (高頻度) 実行するアクティビティには、1 アクティビティにつき 1 か月あたり 1 USD の費用がかかります。1 日 1 回以下 (低頻度) に実行するアクティビティには、1 回のアクティビティにつき 1 か月あたり 0.60 USD の費用がかかります。非アクティブなパイプラインの料金は、パイプライン 1 つにつき 1 USD です。詳細については、AWS Data Pipeline 料金表 ページを参照してください。

Amazon MWAA の料金は、マネージド Apache Airflow 環境の存続期間と、より多くのワーカーやスケジューラーの容量を提供するために必要な追加の自動スケーリングに基づいています。Amazon MWAA 環境の使用量に対しては、時間単位 (1 秒単位で請求) で支払いますが、料金は環境の規模によって異なります。Amazon MWAA は、環境設定に基づいてワーカー数を自動スケーリングします。 AWS は追加のワーカーのコストは個別に計算します。さまざまな Amazon MWAA 環境サイズを使用する場合の 1 時間あたりのコストの詳細については、Amazon MWAA 料金表 ページを参照してください。

関連リソース

Amazon MWAA の使用に関する詳細情報とベストプラクティスについては、次のリソースを参照してください。

- Amazon MWAA API リファレンス
- Amazon MWAA のモニタリングダッシュボードとアラーム
- Amazon MWAA での Apache Airflow のパフォーマンス調整

Amazon MWAA のドキュメント履歴

次の表は、2022 年 3 月以降に追加された Amazon MWAA 移行ガイドの重要な更新点について説明しています。

変更	説明	日付
AWS Data Pipeline から Amazon MWAA へのワークロードの移行に関する新しいトピック	AWS Data Pipeline から Amazon MWAA への既存のワークロードの移行に関する新しい情報とガイダンスが追加されました。この情報を移行計画の設計に役立ててください。 • ワークロード AWS Data Pipeline を から Amazon MWAA に移行する	2023 年 4 月 14 日
Amazon MWAA 移行ガイドの開始	Amazon MWAA では、新しい Amazon MWAA 環境への移行に関する詳細なガイダンスを提供するようになりました。Amazon MWAA 移行ガイドに記載されている手順は、既存の Amazon MWAA 環境、または自己管理型の Apache Airflow デプロイからの移行に適用されます。 • Amazon MWAA 移行ガイドについて	2022 年 3 月 7 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。