aws

마이그레이션 가이드

Amazon Managed Workflows for Apache Airflow

Amazon Managed Workflows for Apache Airflow: 마이그레이션 가이드

Table of Contents

Amazon MWAA 마이그레이션 가이드란 무엇입니까?

Amazon Managed Workflows for Apache Airflow는 Apache Airflow에 대한 관리형 오케스트레이션 서비스로 클라우드에서 데이터 파이프라인을 대규모로 쉽게 운영할 수 있습니다. Amazon MWAA가 Apache Airflow의 프로비저닝 및 지속적인 유지 관리를 관리하므로 사용자는 더 이상 인스턴스의 패치, 확장 또는 보안에 대해 걱정할 필요가 없습니다.

Amazon MWAA는 작업을 실행하는 컴퓨팅 리소스의 규모를 자동으로 조정하여 온디맨드 방식으로 일관된 성능을 제공합니다. Amazon MWAA는 기본적으로 데이터를 보호합니다. Amazon Virtual Private Cloud를 사용하여 격리되고 안전한 자체 클라우드 환경에서 워크로드를 실행할 수 있습니다. 이렇게 하면 AWS Key Management Service을(를) 사용하여 데이터가 자동으로 암호화됩니다.

이 가이드를 활용하여 자체 관리형 Apache Airflow 워크플로를 Amazon MWAA로 마이그레이션하거나 기존 Amazon MWAA 환경을 새 Apache Airflow 버전으로 업그레이드합니다. 마이그레이션 튜토리얼에서는 새 Amazon MWAA 환경을 생성 또는 복제하고, 워크플로 리소스를 마이그레이션하고, 워크플로우 메타데이터와 로그를 새 환경으로 전송하는 방법을 설명합니다.

마이그레이션 튜토리얼을 시도하기 전에 다음 주제를 검토하시기 바랍니다.

- 네트워크 아키텍처
- 주요 고려 사항

Amazon MWAA 네트워크 아키텍처 살펴보기

다음 섹션에서는 Amazon MWAA 환경을 구성하는 주요 구성 요소와 각 환경이 리소스를 관리하고, 데이터를 안전하게 유지하고, 워크플로에 대한 모니터링 및 가시성을 제공하기 위해 통합되는 AWS 서비스 세트에 대해 설명합니다.

주제

Amazon MWAA 구성 요소

Amazon MWAA 환경은 다음과 같은 네 가지 주요 구성 요소로 구성되어 있습니다.

1. 스케줄러 — 모든 DAG를 구문 분석 및 모니터링하고 DAG의 종속성이 충족되면 실행을 위해 작업을 대기열에 넣습니다. Amazon MWAA는 스케줄러를 최소 2개의 스케줄러가 있는 AWS Fargate 클러스터로 배포합니다. 워크로드에 따라 스케줄러 수를 최대 5개까지 늘릴 수 있습니다. Amazon MWAA 환경 클래스에 대한 자세한 내용은 Amazon MWAA 환경 클래스를 참조하세요.
2. 작업자 — 예약된 작업을 실행하는 하나 이상의 Fargate 작업. 사용자 환경의 작업자 수는 지정한 최소 및 최대 수 사이의 범위에 따라 결정됩니다. Amazon MWAA는 대기하고 실행 중인 작업의 수가 기존 작업자가 처리할 수 있는 양보다 많을 때 작업자 자동 규모 조정을 시작합니다. 실행 중인 작업과 대기 중인 작업의 합계가 2분 이상 0이 되면 Amazon MWAA는 작업자 수를 최소로 축소합니다. Amazon MWAA가 자동 규모 조정 작업자를 처리하는 방법에 대한 자세한 내용은 Amazon MWAA 자동 규모 조정을 참조하세요.
3. 웹 서버 — Apache Airflow 웹 UI를 실행합니다. 프라이빗 또는 퍼블릭 네트워크 액세스로 웹 서버를 구성할 수 있습니다. 두 경우 모두 Apache Airflow 사용자의 액세스는 AWS Identity and Access Management(IAM)에서 정의한 액세스 제어 정책 및 SSO에 의해 제어됩니다. 환경에 대한 IAM 액세스 정책을 구성하는 방법에 대한 자세한 내용은 Amazon MWAA 환경 액세스를 참조하세요.
4. 데이터베이스 — DAG 실행 기록을 포함하여 Apache Airflow 환경 및 워크플로에 대한 메타데이터를 저장합니다. 데이터베이스는 AWS에서 관리하는 단일 테넌트 Aurora PostgreSQL 데이터베이스이며 비공개로 보호된 Amazon VPC 엔드포인트를 통해 스케줄러 및 작업자 Fargate 컨테이너에 액세스할 수 있습니다.

또한 모든 Amazon MWAA 환경은 DAG 및 작업 종속성 저장 및 액세스, 저장된 데이터 보호, 환경 로깅 및 모니터링을 비롯한 다양한 태스크를 처리하기 위한 일련의 AWS 서비스와 상호 작용합니다. 다음 다이어그램은 Amazon MWAA 환경의 다양한 구성 요소를 보여줍니다.

> **Note**
>
> 서비스 Amazon VPC는 공유 VPC가 아닙니다. Amazon MWAA는 사용자가 생성하는 모든 환경에 대해 AWS 소유의 VPC를 생성합니다.

- Amazon S3 — Amazon MWAA는 DAG, 요구 사항 및 플러그인 파일과 같은 모든 워크플로 리소스를 Amazon S3 버킷에 저장합니다. 환경 생성의 일환으로 버킷을 생성하고 Amazon MWAA 리소스를 업로드하는 방법에 대한 자세한 내용은 Amazon MWAA 사용 설명서의 Amazon MWAA용 Amazon S3 버킷 생성을 참조하세요.
- Amazon SQS — Amazon MWAA는 Amazon SQS를 사용하여 Celery 실행자로 워크플로 작업을 대기열에 추가합니다/
- Amazon ECR — Amazon ECR은 모든 Apache Airflow 이미지를 호스팅합니다. Amazon MWAA는 AWS 관리형 Apache Airflow 이미지만 지원합니다.
- AWS KMS — Amazon MWAA는 저장된 데이터를 안전하게 보호하기 위해 AWS KMS를 사용합니다. 기본적으로 Amazon MWAA는 AWS관리형 AWS KMS 키를 사용하지만 고객이 관리형 AWS KMS 키를 사용하도록 환경을 구성할 수 있습니다. 자체 고객 관리형 AWS KMS 키 사용에 대한 자세한 내용은 Amazon MWAA 사용 설명서의 데이터 암호화를 위한 고객 관리형 키를 참조하세요.
- CloudWatch — Amazon MWAA는 CloudWatch와 통합되어 Apache Airflow 로그 및 환경 지표를 CloudWatch에 제공하므로, 사용자는 Amazon MWAA 리소스를 모니터링하고 문제를 해결할 수 있습니다.

연결

Amazon MWAA 환경은 통합되는 모든 AWS 서비스에 액세스할 수 있어야 합니다. Amazon MWAA 실행 역할은 사용자를 대신하여 다른 AWS 서비스에 연결할 수 있도록 Amazon MWAA에 액세스 권한을 부여하는 방법을 제어합니다. 네트워크 연결의 경우 Amazon VPC에 대한 퍼블릭 인터넷 액세스를 제공하거나 Amazon VPC 엔드포인트를 생성할 수 있습니다. 환경에 맞게 Amazon VPC 엔드포인트(AWS PrivateLink)를 구성하는 방법에 대한 자세한 내용은 Amazon MWAA 사용 설명서의 Amazon MWAA의 VPC 엔드포인트에 대한 액세스 관리를 참조하세요.

Amazon MWAA는 스케줄러 및 작업자에 요구 사항을 설치합니다. 요구 사항이 공개 PyPI 리포지토리에서 제공되는 경우 필요한 라이브러리를 다운로드하려면 환경에 인터넷에 연결되어 있어야 합니다. 프라이빗 환경의 경우 프라이빗 PyPI 리포지토리를 사용하거나 라이브러리를 사용자 환경에 맞는 사용자 지정 플러그인으로 `.whl` 파일에 번들로 묶을 수 있습니다.

Apache Airflow를 프라이빗 모드로 구성하면 Amazon VPC 엔드포인트를 통해서만 Amazon VPC가 Apache Airflow UI에 액세스할 수 있습니다.

네트워킹에 대한 자세한 내용은 Amazon MWAA 사용 설명서의 네트워킹을 참조하세요.

새 MWAA 환경으로 마이그레이션하기 위한 주요 고려 사항

Apache Airflow 워크로드를 Amazon MWAA로 마이그레이션하려는 경우 인증 및 Amazon MWAA 실행 역할과 같은 주요 고려 사항에 대해 자세히 알아봅니다.

주제

Authentication

Amazon MWAA는 AWS Identity and Access Management (IAM)을 사용하여 Apache Airflow UI에 대한 액세스를 제어합니다. Apache Airflow 사용자에게 웹 서버에 액세스하고 DAG를 관리할 수 있는 권한을 부여하는 IAM 정책을 만들고 관리해야 합니다. 여러 계정에서 IAM을 사용하여 Apache Airflow의 기본 역할에 대한 인증과 권한 부여를 모두 관리할 수 있습니다.

사용자 지정 Airflow 역할을 생성하고 이를 IAM 보안 주체에 매핑하여 Apache Airflow 사용자가 워크플로 DAG의 일부에만 액세스하도록 관리하고 제한할 수 있습니다. 자세한 내용과 단계별 튜토리얼은 튜토리얼: DAG의 하위 집합에 대한 Amazon MWAA 사용자 액세스 제한을 참조하세요.

Amazon MWAA에 액세스하도록 페더레이션된 ID를 구성할 수도 있습니다. 자세한 내용은 다음을 참조하세요.

- 퍼블릭 액세스가 가능한 Amazon MWAA 환경 — AWS Compute 블로그에서 Amazon MWAA와 함께 Okta를 ID 공급자로 사용합니다.
- 프라이빗 액세스가 가능한 Amazon MWAA 환경 — 페더레이션 ID를 사용하여 프라이빗 Amazon MWAA 환경에 액세스합니다.

실행 역할

Amazon MWAA는 환경에 다른 AWS 서비스에 액세스할 수 있는 권한을 부여하는 실행 역할을 사용합니다. 역할에 관련 권한을 추가하여 워크플로에 AWS 서비스에 대한 액세스 권한을 제공할 수 있습니다. 환경을 처음 생성할 때 새 실행 역할을 생성하는 기본 옵션을 선택하면 Amazon MWAA가 역할에 필요한 최소 권한을 연결합니다. 단, Amazon MWAA가 모든 로그 그룹을 자동으로 추가하는 CloudWatch Logs의 경우는 예외입니다.

실행 역할이 생성되면 Amazon MWAA는 사용자를 대신하여 권한 정책을 관리할 수 없습니다. 실행 역할을 업데이트하려면 정책을 편집하여 필요에 따라 권한을 추가 및 제거해야 합니다. 예를 들어 Amazon MWAA 환경과 AWS Secrets Manager를 백엔드로 통합하여 Apache Airflow 워크플로우에 사용할 암호와 연결 문자열을 안전하게 저장할 수 있습니다. 이렇게 하려면 환경의 실행 역할에 다음 권한 정책을 추가합니다.

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": "*"
        }
    ]
}
```

다른 AWS 서비스와의 통합은 유사한 패턴을 따릅니다. Amazon MWAA 실행 역할에 관련 권한 정책을 추가하여 Amazon MWAA에 서비스에 액세스할 수 있는 권한을 부여합니다. Amazon MWAA 실행 역할 관리에 대한 자세한 내용과 추가 예를 보려면 Amazon MWAA 사용 설명서의 Amazon MWAA 실행 역할을 참조하세요.

새로운 Amazon MWAA 환경으로 마이그레이션

기존 Apache Airflow 워크로드를 새로운 Amazon MWAA 환경으로 마이그레이션하려면 다음 단계를 알아보세요. 이러한 단계를 사용하여 이전 버전의 Amazon MWAA에서 새 버전 릴리스로 마이그레이션하거나 자체 관리형 Apache Airflow 배포를 Amazon MWAA로 마이그레이션할 수 있습니다. 이 튜토리얼에서는 기존 Apache Airflow v1.10.12에서 Apache Airflow v2.5.1을 실행하는 새 Amazon MWAA로 마이그레이션한다고 가정하지만, 동일한 절차를 사용하여 다른 Apache Airflow 버전에서 또는 다른 Apache Airflow 버전으로 마이그레이션할 수 있습니다.

주제

사전 조건

단계를 완료하고 환경을 마이그레이션하려면 다음이 필요합니다.

- Apache Airflow 배포. 이는 자체 관리형 또는 기존 Amazon MWAA 환경일 수 있습니다.
- 로컬 운영 시스템에 설치한 도커.
- AWS Command Line Interface 버전 2가 설치되었습니다.

1단계: 지원되는 최신 Apache Airflow 버전을 실행하는 새 Amazon MWAA 환경 생성

Amazon MWAA 사용 설명서의 Amazon MWAA 시작하기의 세부 단계를 사용하거나 CloudFormation 템플릿을 사용하여 환경을 생성할 수 있습니다. 기존 Amazon MWAA 환경에서 마이그레이션하고 CloudFormation 템플릿을 사용하여 이전 환경을 생성하는 경우 `AirflowVersion` 속성을 변경하여 새 버전을 지정할 수 있습니다.

```
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
```

또는, 기존 Amazon MWAA 환경에서 마이그레이션하는 경우 AWS SDK for Python(Boto3)을 사용하는 다음 Python 스크립트를 복사하여 환경을 복제할 수 있습니다. 스크립트를 다운로드할 수 있습니다.

Python 스크립트

```
# 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)
```

2단계: 워크플로 리소스 마이그레이션

Apache Airflow v2는 메이저 버전 릴리스입니다. Apache Airflow v1에서 마이그레이션하는 경우 워크플로 리소스를 준비하고 DAG, 요구 사항 및 플러그인에 대한 변경 사항을 확인해야 합니다. 이렇게 하려면 Docker와 Amazon MWAA 로컬 러너를 사용하여 로컬 운영 체제에서 브리지 버전의 Apache Airflow를 구성하는 것이 좋습니다. Amazon MWAA 로컬 러너는 Amazon MWAA 환경을 로컬로 복제하는 명령줄 인터페이스(CLI) 유틸리티를 제공합니다.

Apache Airflow 버전을 변경할 때마다 `requirements.txt`에서 올바른 `--constraint` URL을 참조하는지 확인합니다.

워크플로 리소스를 마이그레이션하려면

1. aws-mwaa-local-runner 리포지토리의 포크를 생성하고 Amazon MWAA 로컬 러너의 사본을 복제합니다.
2. aws-mwaa-local-runner 리포지토리의 `v1.10.15` 브랜치를 확인합니다. Apache Airflow는 Apache Airflow v2로 마이그레이션하는 데 도움이 되는 브리지 릴리스로 v1.10.15를 릴리스했으며, Amazon MWAA가 v1.10.15를 지원하지 않지만 Amazon MWAA 로컬 러너를 사용하여 리소스를 테스트할 수 있습니다.
3. Amazon MWAA 로컬 러너 CLI 도구를 사용하여 도커 이미지를 빌드하고 로컬에서 Apache Airflow를 실행할 수 있습니다. 자세한 내용은 GitHub 리포지토리에서 로컬 러너 README를 참조하세요.
4. Apache Airflow를 사용해 로컬에서 실행하려면 Apache Airflow 설명서 웹 사이트의 1.10에서 2로 업그레이드에 설명된 단계를 따릅니다.
    a. `requirements.txt`을(를) 업데이트하려면 Amazon MWAA 사용 설명서의 Python 종속성 관리에서 권장하는 모범 사례를 따르세요.

b. 사용자 지정 운영자 및 센서를 기존 Apache Airflow v1.10.12 환경용 플러그인과 함께 번들링한 경우 이를 DAG 폴더로 옮깁니다. Apache Airflow v2+의 모듈 관리 모범 사례에 대한 자세한 내용은 Apache Airflow 설명서 웹 사이트의 모듈 관리를 참조하세요.

5. 워크플로 리소스에 대한 필수 변경 사항을 실시한 후에는 aws-mwaa-local-runner 리포지토리의 `v2.5.1` 브랜치를 체크아웃하고 업데이트된 워크플로우 DAG, 요구 사항 및 사용자 지정 플러그인을 로컬에서 테스트합니다. 다른 Apache Airflow 버전으로 마이그레이션하는 경우 해당 버전에 적합한 로컬 러너 브랜치를 대신 사용할 수 있습니다.
6. 워크플로 리소스를 성공적으로 테스트한 후 DAG, `requirements.txt` 및 플러그인을 새 Amazon MWAA 환경으로 구성한 Amazon S3 버킷에 복사합니다.

3단계: 기존 환경에서 메타데이터 내보내기

업데이트된 DAG 파일을 환경의 Amazon S3 버킷에 복사하고 스케줄러가 이를 구문 분석하면 `dag`, `dag_tag` 및 `dag_code`와 같은 Apache Airflow 메타데이터 테이블이 자동으로 채워집니다. 권한 관련 테이블도 사용자의 IAM 실행 역할 권한에 따라 자동으로 채워집니다. 이를 마이그레이션할 필요는 없습니다.

DAG 기록, `variable`, `slot_pool`, `sla_miss`와 관련된 데이터 및 필요한 경우 `xcom`, `job` 및 `log` 테이블과 관련된 데이터를 마이그레이션할 수 있습니다. 작업 인스턴스 로그는 `airflow-`*`{environment_name}`* 로그 그룹 아래의 CloudWatch Logs에 저장됩니다. 이전 실행의 작업 인스턴스 로그를 보려면 해당 로그를 새 환경 로그 그룹에 복사해야 합니다. 관련 비용을 줄이기 위해 며칠 분량의 로그만 옮기는 것이 좋습니다.

기존 Amazon MWAA 환경에서 마이그레이션하는 경우 메타데이터 데이터베이스에 직접 액세스할 수 없습니다. DAG를 실행하여 기존 Amazon MWAA 환경에서 선택한 Amazon S3 버킷으로 메타데이터를 내보내야 합니다. 자체 관리형 환경에서 마이그레이션하는 경우 다음 단계를 사용하여 Apache Airflow 메타데이터를 내보낼 수도 있습니다.

데이터를 내보낸 후에는 새 환경에서 DAG를 실행하여 데이터를 가져올 수 있습니다. 내보내기 및 가져오기 프로세스 중에 다른 모든 DAG는 일시 중지됩니다.

기존 환경에서 메타데이터를 내보내려면

1. 를 사용하여 Amazon S3 버킷을 생성 AWS CLI 하여 내보낸 데이터를 저장합니다. `UUID` 및 `region`를 사용자의 정보로 교체합니다.

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

```
--region {region}
```

> **Note**
>
> 변수에 저장하는 연결과 같이 민감한 데이터를 마이그레이션하는 경우 Amazon S3 버킷의 기본 암호화를 활성화하는 것이 좋습니다.

2.

> **Note**
>
> 자체 관리형 환경에서 마이그레이션하는 경우에는 적용되지 않습니다.

기존 환경의 실행 역할을 수정하고 다음 정책을 추가하여 1단계에서 생성한 버킷에 쓰기 액세스 권한을 부여합니다.

JSON

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

3. amazon-mwaa-examples 리포지토리를 복제하고 마이그레이션 시나리오의 `metadata-migration` 하위 디렉터리로 이동합니다.

```
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`에서 내보낸 메타데이터를 저장하기 위해 `S3_BUCKET`에 대한 문자열 값을 생성한 Amazon S3 버킷으로 교체합니다.

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

5. metadata-migration 디렉터리에서 requirements.txt 파일을 찾습니다. 기존 환경에 대한 요구 사항 파일이 이미 있는 경우 requirements.txt에 지정된 추가 요구 사항을 사용자 파일에 추가합니다. 기존 요구 사항 파일이 없는 경우 metadata-migration 디렉터리에 제공된 파일을 사용하면 됩니다.
6. 기존 환경에 연결된 Amazon S3 버킷의 DAG 디렉터리에 export_data.py을 복사합니다. 자체 관리형 환경에서 마이그레이션하는 경우 사용자 /dags 폴더에 export_data.py을 복사합니다.
7. 업데이트된 requirements.txt을 기존 환경과 연결된 Amazon S3 버킷으로 복사한 다음 환경을 편집하여 새 requirements.txt 버전을 지정합니다.
8. 환경이 업데이트된 후 Apache Airflow UI에 액세스하여 db_export DAG의 일시 중지를 해제하고 워크플로가 실행되도록 트리거합니다.
9. 메타데이터가 mwaa-migration-*{UUID}* Amazon S3 버킷의 data/migration/*existing-version*_to_*new-version*/export/로 내보내졌고 각 테이블이 전용 파일에 포함되어 있는지 확인합니다.

4단계: 새 환경으로 메타데이터 가져오기

새 환경으로 메타데이터를 가져오려면

1. import_data.py에서 다음에 대한 문자열 값을 사용자의 정보로 교체합니다.
   - 기존 Amazon MWAA 환경에서 마이그레이션하는 경우:

     ```
     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_DAYS는 워크플로가 새 환경에 복사하는 로그 파일의 일수를 제어합니다.
   - 자체 관리형 환경에서 마이그레이션하는 경우

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

2. (선택 사항) `import_data.py`은 실패한 작업 로그만 복사합니다. 모든 작업 로그를 복사하려면 `getDagTasks` 함수를 수정하고 다음 코드 조각에 표시된 대로 `ti.state = 'failed'`를 삭제합니다.

```
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. 새 환경의 실행 역할을 수정하고 다음 정책을 추가합니다. 권한 정책을 통해 Amazon MWAA는 Apache Airflow 메타데이터를 내보낸 Amazon S3 버킷에서 읽고 기존 로그 그룹에서 작업 인스턴스 로그를 복사할 수 있습니다. 모든 자리 표시자를 사용자 정보로 교체합니다.

> **Note**
>
> 자체 관리형 환경에서 마이그레이션하는 경우 정책에서 CloudWatch Logs 관련 권한을 삭제해야 합니다.

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. 새 환경과 연결된 Amazon S3 버킷의 DAG 디렉터리에 `import_data.py`을 복사한 다음 Apache Airflow UI에 액세스하여 `db_import` DAG를 일시 중지 해제하고 워크플로를 트리거합니다. 몇 분 후에 Apache Airflow UI에 새 DAG가 나타납니다.
5. DAG 실행이 완료되면 각 개별 DAG에 액세스하여 DAG 실행 기록이 복사되었는지 확인합니다.

다음 단계

- 사용 가능한 Amazon MWAA 환경 클래스 및 기능에 대한 자세한 내용은 Amazon MWAA 사용 설명서의 Amazon MWAA 환경 클래스를 참조하세요.
- Amazon MWAA가 자동 크기 조정 작업자를 처리하는 방법에 대한 자세한 내용은 Amazon MWAA 사용 설명서의 Amazon MWAA 오토 스케일링을 참조하세요.
- Amazon MWAA REST API에 대한 자세한 내용은 Amazon MWAA REST API를 참조하세요.

에서 Amazon MWAA AWS Data Pipeline 로 워크로드 마이그레이션

AWS 는 2012년에 AWS Data Pipeline 서비스를 시작했습니다. 당시 고객은 다양한 컴퓨팅 옵션을 사용하여 서로 다른 데이터 소스 간에 데이터를 이동할 수 있는 서비스를 원했습니다. 시간이 경과하면서 데이터 전송 요구 사항이 변함에 따라 이러한 요구 사항에 대한 솔루션도 변했습니다. 이제 비즈니스 요구 사항에 가장 부합하는 솔루션을 선택할 수 있습니다. 워크로드를 다음 AWS 서비스 중 하나로 마이그레이션할 수 있습니다.

- Amazon Managed Workflows for Apache Airflow(Amazon MWAA)를 사용하여 Apache Airflow에 대한 워크플로 오케스트레이션을 관리합니다.
- Step Functions를 사용하여 여러 AWS 서비스간의 워크플로를 오케스트레이션합니다.
- AWS Glue 를 사용하여 Apache Spark 애플리케이션을 실행하고 오케스트레이션합니다.

선택한 옵션은 현재 AWS Data Pipeline의 워크로드에 따라 다릅니다. 이 주제에서는에서 Amazon MWAA AWS Data Pipeline 로 마이그레이션하는 방법을 설명합니다.

주제

Amazon MWAA 선택

Amazon Managed Workflows for Apache Airflow(Amazon MWAA)는 Apache Airflow에 대한 관리형 오케스트레이션 서비스로 클라우드에서 엔드투엔드 데이터 파이프라인을 대규모로 쉽게 설정하고 운영할 수 있습니다. Apache Airflow는 워크플로라고 하는 프로세스 및 작업 시퀀스를 프로그래밍 방식으로 작성, 예약 및 모니터링하는 데 사용되는 오픈 소스 도구입니다. Amazon MWAA를 사용하면 확장성, 가용성 및 보안을 위해 기본 인프라를 관리할 필요 없이 Apache Airflow와 Python 프로그래밍 언어를 사용하여 워크플로를 생성할 수 있습니다. Amazon MWAA는 필요에 맞게 워크플로 용량을 자동으로 확장하고 AWS 보안 서비스와 통합되어 데이터에 대한 빠르고 안전한 액세스를 제공합니다.

다음은에서 Amazon MWAA AWS Data Pipeline 로 마이그레이션할 때 얻을 수 있는 몇 가지 이점을 강조합니다.

- 확장성 및 성능 향상 – Amazon MWAA는 워크플로를 정의하고 실행할 수 있는 유연하고 확장 가능한 프레임워크를 제공합니다. 이를 통해 사용자는 크고 복잡한 워크플로를 쉽게 처리하고 동적 작업 예약, 데이터 기반 워크플로 및 병렬화와 같은 기능을 활용할 수 있습니다.
- 모니터링 및 로깅 개선 – Amazon MWAA는 Amazon CloudWatch와 통합되어 워크플로의 모니터링 및 로깅을 개선합니다. Amazon MWAA는 자동으로 시스템 지표와 로그를 CloudWatch에 전송합니다. 즉, 워크플로의 진행 상황과 성능을 실시간으로 추적하고 발생하는 문제를 식별할 수 있습니다.
- AWS 서비스 및 타사 소프트웨어와의 통합 개선 - Amazon MWAA는 Amazon S3, Amazon Redshift와 같은 다양한 기타 AWS 서비스뿐만 아니라 DBT, Snowflake, Databricks와 같은 타사 소프트웨어와도 통합됩니다. AWS Glue이를 통해 다양한 환경 및 서비스 전반에서 데이터를 처리하고 전송할 수 있습니다.
- 오픈 소스 데이터 파이프라인 도구 – Amazon MWAA는 사용자에게 익숙한 것과 동일한 오픈 소스 Apache Airflow 제품을 활용합니다. Apache Airflow는 수집, 처리, 전송, 무결성 테스트, 품질 검사, 데이터 계보 보장 등 데이터 파이프라인 관리의 모든 측면을 처리하도록 설계된 특수 목적의 도구입니다.
- 현대적이고 유연한 아키텍처 – Amazon MWAA는 컨테이너화와 클라우드 네이티브, 서버리스 기술을 활용합니다. 따라서 유연성과 이동성이 향상되고 워크플로 환경을 더 쉽게 배포 및 관리할 수 있습니다.

아키텍처 및 개념 매핑

AWS Data Pipeline 및 Amazon MWAA에는 마이그레이션 프로세스와 워크플로 정의 및 실행 방식에 영향을 미칠 수 있는 다양한 아키텍처와 구성 요소가 있습니다. 이 섹션에서는 두 서비스의 아키텍처와 구성 요소를 개괄적으로 살펴보고 몇 가지 주요 차이점을 중점적으로 설명합니다.

AWS Data Pipeline 및 Amazon MWAA는 모두 완전 관리형 서비스입니다. 워크로드를 Amazon MWAA로 마이그레이션할 때 Apache Airflow를 사용하여 기존 워크플로를 모델링하는 새로운 개념을 배워야 할 수 있습니다. 하지만 인프라를 관리하고, 작업자에게 패치를 적용하고, 운영 체제 업데이트를 관리할 필요는 없습니다.

다음 표에서는의 주요 개념을 Amazon MWAA의 주요 개념 AWS Data Pipeline 과 연결합니다. 이 정보를 출발점으로 삼아 마이그레이션 계획을 설계합니다.

개념	AWS Data Pipeline	Amazon MWAA
파이프라인 정의	AWS Data Pipeline 는 워크플로를 정의하는 JSON 기반 구성 파일을 사용합니다.	Amazon MWAA는 워크플로를 정의하는 Python 기반 방향성 비순환 그래프(DAG)를 사용합니다.
파이프라인 실행 환경	워크플로는 Amazon EC2 인스턴스에서 실행됩니다.는 사용자를 대신하여 이러한 인스턴스를 AWS Data Pipeline 프로비저닝하고 관리합니다.	Amazon MWAA는 Amazon ECS 컨테이너식 환경을 사용하여 작업을 실행합니다.
파이프라인 구성요소	활동은 워크플로의 일부로 실행되는 작업을 처리하는 것입니다.	연산자(작업)는 워크플로의 기본 처리 단위입니다.
	사전 조건은 활동 실행 전에 충족되어야 할 조건문이 포함되어 있습니다.	센서(작업)는 리소스 또는 작업이 완료될 때까지 기다렸다가 실행할 수 있는 조건문을 나타냅니다.
	의 리소스는 파이프라인 활동이 지정하는 작업을 수행하는 AWS 컴퓨팅 리소스를 AWS Data Pipeline 나타냅니다. Amazon EC2와 Amazon EMR은 사용 가능한 두 가지 리소스입니다.	DAG에서 작업을 사용하면 Amazon ECS, Amazon EMR 및 Amazon EKS를 비롯한 다양한 컴퓨팅 리소스를 정의할 수 있습니다. Amazon MWAA는 Amazon ECS에서 실행되는 작업자에서 Python 작업을 실행합니다.
파이프라인 실행	AWS Data Pipeline 는 정기적인 속도 기반 및 cron 기반 패턴으로 실행 예약을 지원합니다.	Amazon MWAA는 사용자 지정 시간표뿐만 아니라 cron 표현식 및 사전 설정을 통한 일정 관리를 지원합니다.

개념	AWS Data Pipeline	Amazon MWAA
	인스턴스는 파이프라인의 각 실행을 참조합니다.	DAG 실행은 Apache Airflow 워크플로의 각 실행을 의미합니다.
	시도란 실패한 작업을 다시 시도하는 것을 말합니다.	Amazon MWAA는 DAG 수준 또는 작업 수준에서 사용자가 정의하는 재시도를 지원합니다.

구현 예제

대부분의 경우 Amazon MWAA로 마이그레이션한 AWS Data Pipeline 후 현재 오케스트레이션 중인 리소스를 재사용할 수 있습니다. 다음 목록에는 가장 일반적인 AWS Data Pipeline 사용 사례에 Amazon MWAA를 사용한 예제 구현이 포함되어 있습니다.

- Amazon EMR 작업 실행(AWS 워크숍)
- Apache Hive 및 Hadoop용 사용자 지정 플러그인 생성(Amazon MWAA 사용 설명서)
- S3에서 Redshift로 데이터 복사(AWS 워크숍)
- 원격 Amazon ECS 인스턴스에서 쉘 스크립트 실행(Amazon MWAA 사용 설명서)
- 하이브리드(온프레미스) 워크플로 오케스트레이션(블로그 게시물)

추가 튜토리얼 및 예제는 다음을 참조하세요.

- Amazon MWAA 튜토리얼
- Amazon MWAA 코드 예제

요금 비교

요금은 파이프라인 수와 각 파이프라인을 사용하는 양을 기준으로 AWS Data Pipeline 합니다. 하루에 두 번 이상 실행하는 활동(빈도가 높음)에는 활동당 월 1 USD의 요금이 부과됩니다. 하루에 한 번 또는 그 이하(낮은 빈도)를 실행하는 활동은 활동당 월 0.60 USD의 요금이 부과됩니다. 비활성 파이프라인의 요금은 파이프라인당 1 USD입니다. 자세한 내용은 AWS Data Pipeline 요금 페이지를 참조하세요.

Amazon MWAA 요금은 관리형 Apache Airflow 환경이 존재하는 기간, 더 많은 작업자 또는 스케줄러 용량을 제공하는 데 필요한 추가 Auto Scaling을 기반으로 합니다. Amazon MWAA 환경 사용에 대한 요금은 시간당(1초 단위로 청구)이며, 환경 규모에 따라 요금이 달라집니다. Amazon MWAA는 환경 구성을 기반으로 작업자 수를 자동으로 조정합니다. AWS 은 추가 작업자의 비용을 별도로 계산합니다. 다양한 Amazon MWAA 환경 크기를 사용하는 데 드는 시간당 비용에 대한 자세한 내용은 Amazon MWAA 요금 페이지를 참조하세요.

관련 리소스

Amazon MWAA 사용에 대한 자세한 내용 및 모범 사례는 다음 리소스를 참조하세요.

- Amazon MWAA API 참조
- Amazon MWAA의 대시보드 및 알람 모니터링
- Amazon MWAA의 Apache Airflow 성능 튜닝

Amazon MWAA 문서 기록

다음 표에는 2022년 5월 이후 Amazon MWAA 마이그레이션 가이드에 대한 중요 추가 사항이 설명되어 있습니다.

변경 사항	설명	날짜
워크로드를 AWS Data Pipeline에서 Amazon MWAA로 마이그레이션하는 것에 대한 새 주제	기존 워크로드를 AWS Data Pipeline에서 Amazon MWAA로 마이그레이션하는 것에 대한 새로운 정보와 지침 추가. 이 정보를 사용하면 마이그레이션 계획을 설계하는 데 도움이 됩니다. • 에서 Amazon MWAA AWS Data Pipeline 로 워크로드 마이그레이션	2023년 4월 14일
Amazon MWAA 마이그레이션 가이드 출시	Amazon MWAA는 이제 새로운 Amazon MWAA 환경으로 마이그레이션하는 방법에 대한 자세한 지침을 제공합니다. Amazon MWAA 마이그레이션 가이드에 설명된 단계는 기존 Amazon MWAA 환경 또는 자체 관리형 Apache Airflow 배포에서 마이그레이션하는 경우에 적용됩니다. • Amazon MWAA 마이그레이션 가이드에 대한 정보	2022년 3월 7일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.