aws

개발자 가이드

Amazon Elastic Container Service

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Elastic Container Service: 개발자 가이드

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

상표 및 브랜드 디자인은 타사 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

Amazon ECS란 무엇입니까? 1
용어 및 구성 요소 1
Features 3
프로비저닝 3
요금 4
관련 서비스 4
시작 5
설정 5
AWS Management Console 5
AWS 계정에 등록 6
관리자 액세스 권한이 있는 사용자 생성 6
Virtual Private Cloud 생성 7
보안 그룹 생성 8
EC2 인스턴스에 연결하기 위한 자격 증명 생성 11
AWS CLI 설치 12
Amazon ECS를 사용하기 위한 다음 단계 12
컨테이너 이미지 생성 13
사전 조건 13
Docker 이미지 생성 15
Amazon Elastic 컨테이너 레지스트리에 이미지 푸시 17
정리 19
다음 단계 19
Fargate 시작 유형에 대한 작업을 생성하는 방법 알아보기 19
사전 조건 20
1단계: 클러스터 생성 20
2단계: 태스크 정의 생성 21
3단계: 서비스 생성 22
4단계: 서비스 보기 23
5단계: 정리 23
Fargate 시작 유형에 대한 Windows 작업을 생성하는 방법 알아보기 24
사전 조건 24
1단계: 클러스터 생성 25
2단계: Windows 태스크 정의 등록 25
3단계: 태스크 정의를 사용하여 서비스 생성 27

4단계: 서비스 보기 27
5단계: 정리 28
EC2 시작 유형에 대한 Windows 작업을 생성하는 방법 알아보기 28
사전 조건 29
1단계: 클러스터 생성 29
2단계: 태스크 정의 등록 31
3단계: 서비스 생성 32
4단계: 서비스 보기 33
5단계: 정리 33
AWS CDK 사용 34
1단계: AWS CDK 프로젝트 설정 35
2단계: AWS CDK를 사용하여 Fargate에 컨테이너화된 웹 서버 정의 37
3단계: 웹 서비스 테스트 45
4단계: 정리 45
다음 단계 45
AWS Copilot CLI를 사용하여 리소스 생성 46
AWS Copilot CLI 설치 47
AWS Copilot CLI를 사용하여 샘플 Amazon ECS 애플리케이션 배포 55
AWS CloudFormation을 사용하는 Amazon ECS 58
템플릿 AWS CloudFormation 예제 59
태스크 정의 59
클러스터 71
서비스 84
Amazon ECS에 대한 IAM 역할 141
AWS CloudFormation 콘솔을 사용하여 Amazon ECS 리소스 생성 144
사전 조건 144
1단계: 스택 템플릿 생성 144
2단계: Amazon ECS 리소스를 위한 스택 생성 150
3단계: 확인 151
4단계: 리소스 정리 151
AWS CLI용 AWS CloudFormation 명령을 사용하여 Amazon ECS 리소스 생성 152
사전 조건 152
1단계: 스택 생성 152
2단계: 리소스 생성 확인 158
3단계: 정리 159
모범 사례 160

AWS Fargate 163
연습 163
용량 공급자 164
태스크 정의 164
플랫폼 버전 164
서비스 로드 밸런싱 165
사용량 지표 165
Fargate 시작 유형 사용 시기에 대한 보안 고려 사항 165
Fargate 보안 모범 사례 166
AWS KMS를 사용하여 Fargate를 위한 임시 스토리지 암호화 166
Fargate를 사용한 커널 시스템 호출 추적을 위한 SYS_PTRACE 기능 166
Fargate Runtime Monitoring과 함께 Amazon GuardDuty 사용 167
Fargate 보안 고려 사항 167
Fargate 플랫폼 버전 168
..... 168
Linux 플랫폼 버전 1.4.0으로 마이그레이션 169
Linux 플랫폼 버전 변경 로그 170
Linux 플랫폼 버전 사용 중단 173
Windows 플랫폼 버전 변경 로그 175
Amazon ECS에 대한 Fargate의 Windows 컨테이너 고려 사항 175
Fargate 작업 임시 스토리지 177
Fargate Linux 컨테이너 플랫폼 버전 177
Fargate Windows 컨테이너 플랫폼 버전 178
AWS Fargate 임시 스토리지에 대한 고객 관리형 키 178
태스크 사용 중지 및 유지 관리 191
태스크 사용 중지 공지 개요 192
태스크 사용 중지를 옵트아웃할 수 있나요? 195
다른 AWS 서비스를 통해 작업 사용 중지 알림을 받을 수 있나요? 196
작업 사용 중지가 예약된 이후에 작업 사용 중지를 변경할 수 있나요? 196
Amazon ECS는 서비스의 일부인 작업을 어떻게 처리하나요? 196
Amazon ECS에서 독립 실행형 작업을 자동으로 처리할 수 있나요? 196
Amazon ECS에서 AWS Fargate 태스크 사용 중지 준비 196
AWS Fargate 리전 199
AWS Fargate의 Linux 컨테이너 199
AWS Fargate의 Windows 컨테이너 201
Amazon ECS를 위한 솔루션 설계 204

Capacity 205
서비스 엔드포인트 205
네트워킹 206
기능 액세스 207
IAM 역할 208
로깅 208
시작 유형 208
Fargate 시작 유형 209
EC2 시작 유형 212
외부 시작 유형 215
듀얼 스택 엔드포인트 사용 216
듀얼 스택 엔드포인트 사용(AWS CLI) 217
듀얼 스택 엔드포인트 사용(AWS SDK) 217
REST API의 듀얼 스택 엔드포인트 사용 218
공유 서브넷, 로컬 영역, Wavelength Zone의 애플리케이션 219
공유 서브넷 219
로컬 영역 220
Wavelength Zone 221
AWS Outposts의 Amazon Elastic Container Service 221
고려 사항 222
사전 조건 222
AWS Outposts에서 클러스터 생성 개요 222
네트워킹 모범 사례 225
애플리케이션을 인터넷에 연결 225
Amazon ECS로의 인바운드 연결을 수신하는 모범 사례 229
AWS 서비스 연결 모범 사례 234
서비스 연결 모범 사례 236
AWS 계정 및 VPC 간의 서비스 네트워킹을 위한 모범 사례 241
네트워킹 문제 해결을 위한 AWS 서비스 241
태스크 시작 시간 최적화 243
대규모 운영 244
Amazon ECS 서비스 할당량 및 API 스로틀링 제한 244
제한 문제 처리 249
오토 스케일링 및 용량 관리 모범 사례 252
태스크 크기 결정 252
서비스 오토 스케일링 최적화 254

용량 및 가용성 ... 259
클러스터 용량 ... 263
Fargate 태스크 크기 선택 ... 263
클러스터 오토 스케일링 속도 향상 ... 264
계정 설정을 사용하여 기능에 액세스 ... 265
Amazon 리소스 이름(ARN) 및 ID ... 266
ARN 및 리소스 ID 형식 타임라인 ... 269
Container Insights ... 269
AWS Fargate Federal Information Processing Standard(FIPS-140) 규정 준수 ... 271
태그 지정 권한 부여 ... 272
태그 지정 권한 부여 타임라인 ... 273
AWS Fargate 작업 사용 중지 대기 시간 ... 274
Linux 컨테이너 인스턴스 네트워크 인터페이스 증가 ... 275
Runtime Monitoring(Amazon GuardDuty 통합) ... 275
듀얼스택 IPv6 VPC ... 276
기본 로그 드라이버 모드 ... 277
콘솔을 사용하여 계정 설정 보기 ... 277
계정 설정 수정 ... 278
기본 계정 설정으로 되돌리기 ... 279
AWS CLI를 사용하여 계정 설정 관리 ... 279
Amazon ECS에 대한 IAM 역할 ... 281
태스크 정의 ... 285
작업 정의 상태 ... 286
삭제를 차단할 수 있는 Amazon ECS 리소스 ... 287
애플리케이션 설계 ... 288
컨테이너 이미지 모범 사례 ... 289
작업 크기 모범 사례 ... 291
EC2 시작 유형에 대한 작업 네트워킹 ... 292
Fargate 시작 유형에 대한 작업 네트워킹 ... 303
작업의 스토리지 옵션 ... 306
컨테이너 스왑 메모리 공간 관리 ... 388
Fargate 시작 유형에 대한 작업 정의 차이 ... 390
Windows를 실행하는 EC2 인스턴스에 대한 작업 정의 차이 ... 398
콘솔을 사용하여 작업 정의 생성 ... 399
JSON 검사기 ... 400
AWS CloudFormation 스택 ... 400

절차 400
콘솔을 사용하여 작업 정의 업데이트 426
JSON 검사기 427
절차 427
콘솔을 사용하여 작업 정의 개정판 등록 취소 428
AWS CloudFormation 스택 400
절차 429
콘솔을 사용하여 작업 정의 개정판 삭제 429
삭제를 차단할 수 있는 Amazon ECS 리소스 287
절차 430
작업 정의 사용 사례 431
GPU 워크로드에 대한 작업 정의 431
비디오 트랜스코딩 워크로드에 대한 작업 정의 441
AWS Neuron 기계 학습 워크로드에 대한 작업 정의 455
딥 러닝 인스턴스에 대한 작업 정의 463
64비트 ARM 워크로드에 대한 작업 정의 465
CloudWatch에 로그 전송 467
로그를 AWS 서비스 또는 AWS Partner로 전송 471
AWS 컨테이너가 아닌 이미지 사용 484
작업의 개별 컨테이너 다시 시작 487
컨테이너로 민감한 데이터 전달 489
Fargate 시작 유형에 대한 태스크 정의 파라미터 511
Family 511
시작 유형 512
태스크 역할 512
태스크 실행 역할 512
네트워크 모드 513
런타임 플랫폼 513
태스크 크기 514
컨테이너 정의 517
Elastic Inference 액셀러레이터 이름 560
프록시 구성 561
볼륨 563
Tags 570
기타 태스크 정의 파라미터 570
EC2 시작 유형에 대한 태스크 정의 파라미터 573

Family ... 573
시작 유형 ... 573
태스크 역할 ... 574
태스크 실행 역할 ... 574
네트워크 모드 ... 575
런타임 플랫폼 ... 576
태스크 크기 ... 577
컨테이너 정의 ... 578
Elastic Inference 액셀러레이터 이름 ... 626
작업 배치 제약 ... 626
프록시 구성 ... 627
볼륨 ... 629
Tags ... 636
기타 태스크 정의 파라미터 ... 637
태스크 정의 템플릿 ... 639
태스크 정의 예제 ... 649
웹 서버 ... 650
splunk 로그 드라이버 ... 652
fluentd 로그 드라이버 ... 653
gelf 로그 드라이버 ... 654
외부 인스턴스의 워크로드 ... 654
Amazon ECR 이미지 및 작업 정의 IAM 역할 ... 656
명령의 진입점 ... 656
컨테이너 종속성 ... 657
작업 정의의 볼륨 ... 659
Windows 샘플 태스크 정의 ... 659
클러스터 ... 661
용량 공급자 ... 662
Fargate용 클러스터 ... 663
Fargate Spot 종료 알림 ... 664
Fargate 시작 유형에 대한 클러스터 생성 ... 666
EC2 시작 유형을 위한 용량 공급자 ... 668
EC2 컨테이너 인스턴스 보안 ... 670
Amazon EC2 시작 유형에 대한 클러스터 생성 ... 671
클러스터 Auto Scaling ... 676
Amazon EC2 컨테이너 인스턴스 ... 708

외부 시작 유형을 위한 클러스터 839
지원되는 운영 체제 및 시스템 아키텍처 839
고려 사항 840
외부 시작 유형에 대한 클러스터 생성 843
Amazon ECS 클러스터에 외부 인스턴스 등록 845
외부 인스턴스 등록 취소 851
AWS Systems Manager 에이전트 및 Amazon ECS 컨테이너 에이전트 업데이트 856
클러스터 업데이트 861
클러스터 삭제 862
컨테이너 인스턴스 등록 취소 863
절차 864
컨테이너 인스턴스 드레이닝 865
서비스에 대한 드레이닝 동작 865
독립 실행형 태스크에 대한 드레이닝 동작 866
절차 866
Amazon EC2 컨테이너 인스턴스 867
수명 주기 868
Docker 지원 869
Amazon ECS 최적화 AMI 869
추가 정보 869
컨테이너 에이전트 구성 870
Amazon ECS 컨테이너 에이전트 설치 873
컨테이너 에이전트 로그 구성 파라미터 878
프라이빗 Docker 이미지를 위한 컨테이너 인스턴스 구성 880
태스크 및 이미지 정리 885
컨테이너 예약 887
컴퓨팅 옵션 889
작업 수명 주기 889
수명 주기 상태 890
Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법 891
EC2 시작 유형 892
Fargate 시작 유형 893
전략을 사용하여 작업 배치 정의 893
그룹 관련 작업 898
작업에 사용되는 컨테이너 인스턴스 정의 899
독립 실행형 작업 909

작업 워크플로 909
애플리케이션을 태스크로 실행 910
Amazon EventBridge Scheduler를 사용하여 태스크 예약 920
작업 중지 926
서비스 927
인프라 컴퓨팅 옵션 928
서비스 Auto Scaling 929
서비스 로드 밸런싱 929
상호 연결 서비스 930
서비스 배포 컨트롤러 및 전략 931
서비스 배포 1068
서비스 개정 1078
가용 영역 재조정 1086
로드 밸런싱을 사용하여 서비스 트래픽 분산 1093
서비스 Auto Scaling 1106
서비스 상호 연결 1142
작업 축소 보호 1198
Amazon ECS 및 Fargate를 사용한 결함 주입 1207
Amazon ECS 서비스 파라미터 업데이트 1215
서비스 삭제 1240
짧은 서비스 ARN 마이그레이션 1241
서비스 스로틀 로직 1247
서비스 정의 파라미터 1248
서비스 정의 템플릿 1274
리소스에 태그 지정 1282
리소스 태그 지정 방법 1282
생성 시 리소스 태그 지정 1285
제한 사항 1286
Amazon ECS 관리형 태그 1286
결제에 태그 사용 1287
리소스에 태그 추가 1288
컨테이너 인스턴스에 태그 추가 1290
외부 컨테이너 인스턴스 1291
사용 보고서 1292
작업 수준 비용 및 사용 1293
모니터링 1295

Amazon ECS 모니터링 모범 사례 1295
모니터링 도구 1296
자동 도구 1296
수동 도구 1298
CloudWatch를 사용하여 Amazon ECS 모니터링 1299
고려 사항 1299
권장 지표 1300
Amazon ECS 지표 보기 1300
Amazon ECS CloudWatch 지표 1302
AWS Fargate 사용량 지표 1325
Amazon ECS 클러스터 예약 지표 1326
Amazon ECS 클러스터 사용률 지표 1328
Amazon ECS 서비스 사용률 지표 1330
EventBridge를 사용하여 Amazon ECS 오류에 대한 응답 자동화 1333
Amazon ECS 이벤트 1334
이벤트 처리 1358
관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링 1361
컨테이너 상태 확인을 사용하여 작업 상태 확인 1362
태스크 상태 결정 방법 1364
상태 확인 및 에이전트 연결 끊김 1365
컨테이너 상태 확인 1365
Amazon ECS 컨테이너 인스턴스 상태 모니터링 1365
컨테이너 인스턴스 상태 문제 1366
애플리케이션 추적 데이터를 사용하여 Amazon ECS 최적화 기회 식별 1367
AWS Distro for OpenTelemetry와 AWS X-Ray 통합에 필요한 IAM 권한 1367
태스크 정의에서 AWS X-Ray 통합을 위한 AWS Distro for OpenTelemetry 사이드카 지정 .. 1369
애플리케이션 지표를 사용하여 Amazon ECS 애플리케이션 성능 상호 연관 1370
Amazon CloudWatch로 애플리케이션 지표 내보내기 1371
Amazon Managed Service for Prometheus로 애플리케이션 지표 내보내기 1374
AWS CloudTrail을 사용하여 Amazon ECS API 직접 호출 로깅 1378
CloudTrail의 Amazon ECS 관리 이벤트 1380
Amazon ECS 이벤트 예제 1380
Amazon ECS 서비스 및 태스크에 대한 CloudWatch Logs Live Tail 보기 1382
필수 권한 1382
절차 1383
AWS CLI 참조 1384

메타데이터를 사용하여 워크로드 모니터링 1384
환경 변수 1385
컨테이너 메타데이터 파일 1386
EC2의 Amazon ECS 작업에 대해 사용 가능한 작업 메타데이터 1392
Fargate의 작업에 사용할 수 있는 작업 메타데이터 1434
컨테이너 내부 검사 1457
Runtime Monitoring을 사용하여 무단 동작 식별 1460
Amazon ECS에서 Runtime Monitoring이 작동하는 방식 1460
고려 사항 1461
리소스 사용률 1462
Fargate 워크로드에 대한 Runtime Monitoring 1462
EC2 워크로드에 대한 Runtime Monitoring 1466
Runtime Monitoring 문제 해결 1470
ECS Exec를 사용하여 Amazon ECS 컨테이너 모니터링 1473
고려 사항 1474
사전 조건 1476
아키텍처 1476
ECS Exec 사용 1477
ECS Exec을 사용한 로깅 1479
IAM 정책을 사용하여 ECS Exec에 대한 액세스 제한 1484
Compute Optimizer 권장 사항 1487
Fargate 작업 크기에 대한 권장 사항 1488
문제 해결 1489
중지된 작업 오류 해결 1492
중지된 작업 오류 메시지 업데이트 1492
중지된 작업 오류 보기 1497
중지된 작업 오류 메시지 1498
작업 연결 확인 1517
IAM 역할 요청 보기 1522
서비스 이벤트 메시지 보기 1523
Amazon ECS 서비스 이벤트 메시지 1524
Amazon ECS 서비스 비정상 이벤트 메시지 1534
Amazon ECS 가용 영역 서비스 리밸런싱 서비스 이벤트 메시지 1535
Amazon ECS의 서비스 로드 밸런서 문제 해결 1537
Amazon ECS에서 서비스 Auto Scaling 문제 해결 1539
작업 정의 잘못된 CPU 또는 메모리 오류 문제 해결 1539

컨테이너 에이전트 로그 보기 1541
Amazon ECS 로그 수집기를 사용하여 컨테이너 로그 수집 1542
에이전트 내부 검사 1545
Amazon ECS의 Docker 진단 1547
Amazon ECS에 Docker 컨테이너 나열 1547
Amazon ECS에서 Docker 로그 보기 1548
Amazon ECS에서 Docker 컨테이너 검사 1549
Amazon ECS에서 Docker 대몬의 자세한 출력 구성 1550
Amazon ECS의 Docker `API error (500): devmapper` 문제 해결 1552
ECS Exec 문제를 해결합니다. 1553
Exec Checker를 사용하여 확인 1553
`execute-command` 호출 중 오류 발생 1553
Amazon ECS Anywhere 문제 해결 1554
외부 인스턴스 등록 문제 1554
외부 인스턴스 네트워크 문제 1555
작업 실행 문제 1555
AWS Fargate 제한 할당량 1556
Fargate에서 `RunTask` API 제한 1556
Fargate에서 요금 할당량 조정 1557
API 실패 이유 1557
보안 1566
ID 및 액세스 관리 1567
고객 1567
ID를 통한 인증 1568
정책을 사용하여 액세스 관리 1571
Amazon Elastic Container Service가 IAM과 작동하는 방식 1573
자격 증명 기반 정책 예제 1584
Amazon ECS에 대한 AWS 관리형 정책 1596
서비스 연결 역할 사용 1608
Amazon ECS에 대한 IAM 역할 1612
Amazon ECS 콘솔에 필요한 권한 1672
Amazon ECS 블루/그린 배포에서 Lambda 함수에 필요한 권한 1681
Amazon ECS 서비스 Auto Scaling에 필요한 IAM 권한 1682
생성 시 리소스에 태그 지정 1683
문제 해결 1688
IAM 모범 사례 1690

로깅 및 모니터링 1692
규정 준수 확인 1694
규정 준수 및 보안 모범 사례 1695
AWS Fargate FIPS-140 규정 준수 1697
AWS Fargate FIPS-140 고려 사항 1697
Fargate에서 FIPS 사용 1697
Fargate FIPS-140 감사에 CloudTrail 사용 1698
인프라 보안 1699
인터페이스 VPC 엔드포인트(AWS PrivateLink) 1700
공동 책임 모델 1706
Fargate 시작 유형 1707
EC2 시작 유형 1708
네트워크 보안 모범 사례 1708
전송 중 암호화 1709
태스크 네트워킹 1710
AWS PrivateLink 및 Amazon ECS 1710
컨테이너 에이전트 설정 1711
네트워크 보안 권장 사항 1712
태스크 및 컨테이너 보안 모범 사례 1713
최소로 생성하거나 distroless 이미지 사용 1713
이미지를 스캔하여 취약성 확인 1714
이미지에서 특수 권한 제거 1715
큐레이팅한 이미지 세트 생성 1716
애플리케이션 패키지 및 라이브러리에서 취약성 스캔 1714
정적 코드 분석 수행 1716
루트가 아닌 사용자로 컨테이너 실행 1717
읽기 전용 루트 파일 시스템 사용 1717
CPU 및 메모리 제한과 함께 작업 구성(Amazon EC2) 1717
Amazon ECR에서 변경 불가능한 태그 사용 1718
컨테이너를 privileged로 실행하지 않음(Amazon EC2) 1718
컨테이너에서 불필요한 Linux 기능 제거 1718
고객 관리형 키(CMK)를 사용하여 Amazon ECR로 푸시된 이미지 암호화 1719
자습서 1720
AWS CLI를 사용하여 Fargate 시작 유형에 대한 Linux 작업 생성 1722
사전 조건 1723
1단계: 클러스터 생성 1724

2단계: Linux 태스크 정의 등록 1725
3단계: 작업 정의 나열 1726
4단계: 서비스 생성 1726
5단계: 서비스 나열 1727
6단계: 실행 서비스 설명 1727
7단계: 테스트 1730
8단계: 정리 1733
AWS CLI를 사용하여 Fargate 시작 유형에 대한 Windows 작업 생성 1734
사전 조건 1734
1단계: 클러스터 생성 1735
2단계: Windows 태스크 정의 등록 1736
3단계: 태스크 정의 나열 1737
4단계: 서비스 생성 1737
5단계: 서비스 나열 1738
6단계: 실행 서비스 설명 1738
7단계: 정리 1741
AWS CLI를 사용하여 EC2 시작 유형에 대한 작업 생성 1741
사전 조건 1742
클러스터 생성 1742
Amazon ECS AMI로 컨테이너 인스턴스 시작 1743
컨테이너 인스턴스 나열 1746
컨테이너 인스턴스 설명 1746
태스크 정의 등록 1749
태스크 정의 나열 1751
서비스 생성 1751
서비스 나열 1752
서비스 설명 1752
실행 중인 태스크 설명 1753
웹 서버 테스트 1754
리소스 정리 1755
CloudWatch Events 이벤트를 수신 대기하도록 Amazon ECS 구성 1757
필수 조건: 테스트 클러스터 설정 1757
1단계: Lambda 함수 생성 1757
2단계: 이벤트 규칙 등록 1758
3단계: 태스크 정의 생성 1759
4단계: 규칙 테스트 1760

작업 중지 이벤트에 대한 Amazon Simple Notification Service 알림 보내기 1761
필수 조건: 테스트 클러스터 설정 1761
전제 조건: Amazon SNS 대한 권한 구성 1761
1단계: Amazon SNS 주제 생성 및 구독 1762
2단계: 이벤트 규칙 등록 1762
3단계: 규칙 테스트 1763
여러 줄 또는 스택 추적 로그 메시지 연결 1765
필수 IAM 권한 1765
여러 줄 로그 설정을 사용할 시기 지정 1766
구문 분석 및 연결 옵션 1768
Windows 컨테이너에서 Fluent Bit 배포 1787
사전 조건 1789
1단계: IAM 액세스 역할 생성 1790
2단계: Amazon ECS Windows 컨테이너 인스턴스 생성 1791
3단계: Fluent Bit 구성 1792
4단계: 로그를 CloudWatch로 라우팅하는 Windows Fluent Bit 작업 정의 등록 1794
5단계: 대몬(daemon) 스케줄링 전략을 사용하여 ecs-windows-fluent-bit 작업 정의를 Amazon ECS 서비스로 실행 1796
6단계: 로그를 생성하는 Windows 작업 정의 등록 1797
7단계: windows-app-task 작업 정의 실행 1798
8단계: CloudWatch에서 로그 확인 1799
9단계: 정리 1800
EC2 Linux 컨테이너에 대해 gMSA 사용 1801
고려 사항 1801
사전 조건 1802
설정 1803
CredSpec 파일 1810
Fargate의 Linux 컨테이너에서 gMSA 사용 1811
고려 사항 1811
사전 조건 1812
설정 1812
CredSpec 파일 1815
AWS CLI를 사용하여 도메인이 없는 gMSA로 Windows 컨테이너 사용 1817
사전 조건 1818
1단계: Active Directory 도메인 서비스(AD DS)에서 gMSA 계정 생성 및 구성 1819
2단계: Secrets Manager로 보안 인증 업로드 1821

3단계: 도메인이 없는 gMSA 정보를 포함하도록 CredSpec JSON 수정 1822
4단계: Amazon S3에 CredSpec 업로드 1823
5단계: (선택 사항) Amazon ECS 클러스터 생성 1824
6단계: 컨테이너 인스턴스에 대한 IAM 역할 생성 1824
7단계: 사용자 지정 작업 실행 역할 생성 1824
8단계: Amazon ECS Exec에 대한 작업 역할 생성 1826
9단계: 작업 정의 등록 1827
10단계: Windows 컨테이너 인스턴스 등록 1829
11단계: 컨테이너 인스턴스 확인 1829
12단계: Windows 작업 실행 1830
13단계: 컨테이너에 gMSA 보안 인증이 있는지 확인 1831
14단계: 정리 1832
디버깅 1833
EC2 Windows 컨테이너에 대해 gMSA를 사용하는 방법을 알아봅니다. 1834
고려 사항 1834
사전 조건 1835
설정 1836
Image Builder를 사용하여 사용자 지정된 Amazon ECS 최적화 AMI 구축 1842
코드형 인프라(IaC)에서 이미지 ARN 사용 1843
AWS CloudFormation에서 이미지 ARN 사용 1846
Terraform에서 이미지 ARN 사용 1847
AWS Deep Learning Containers 사용 1847
Service quotas 1848
AWS Management Console에서 서비스 할당량 관리 1848
서비스 할당량 및 API 제한 한도 처리 1850
Elastic Load Balancing 245
탄력적 네트워크 인터페이스 247
AWS Cloud Map 248
Amazon ECS API 참조 1855
문서 기록 1856

Amazon Elastic Container Service란 무엇입니까?

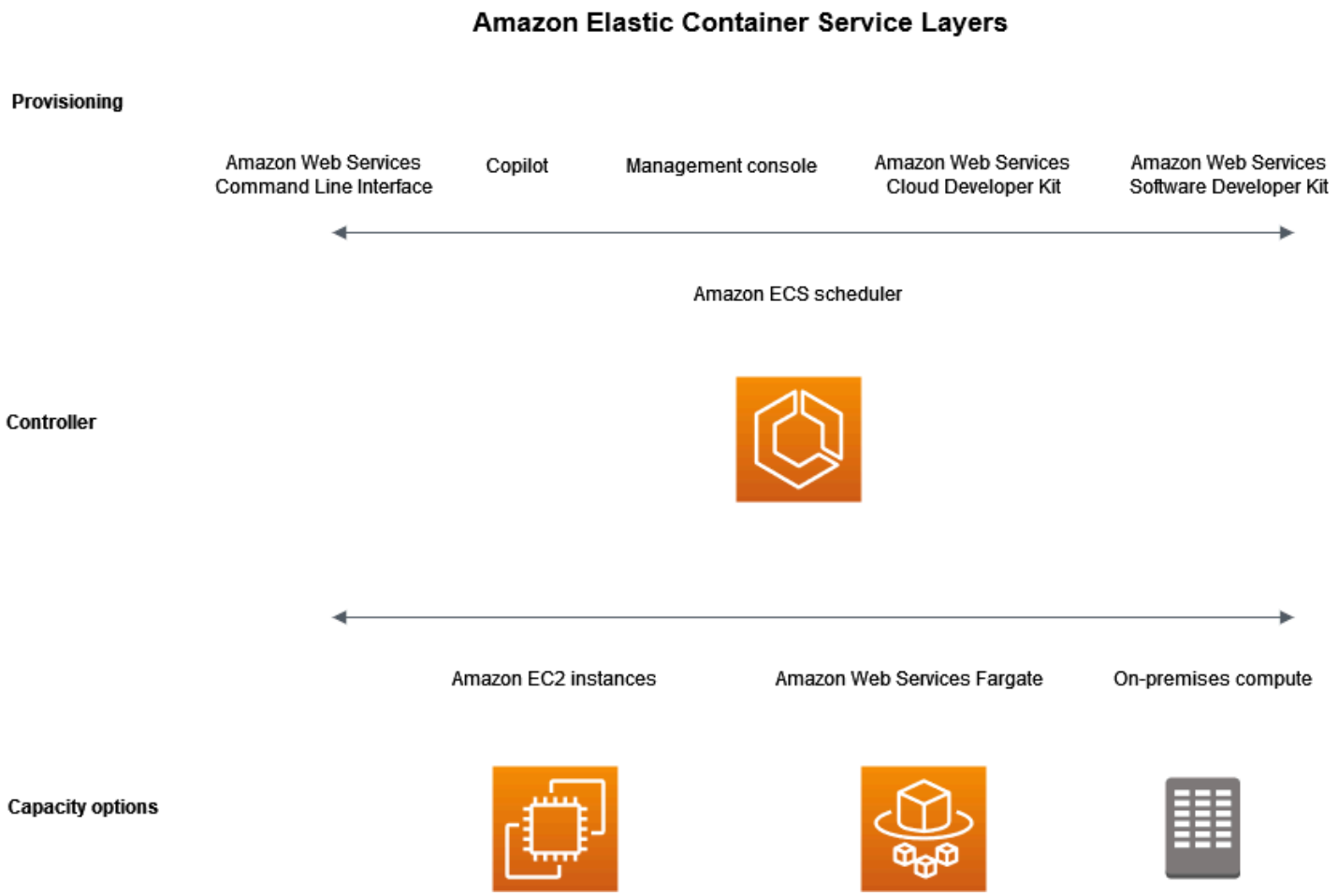
Amazon Elastic Container Service(Amazon ECS)는 컨테이너 애플리케이션을 쉽게 배포, 관리 및 확대할 수 있도록 도와주는 완전 관리형 컨테이너 오케스트레이션 서비스입니다. 완전 관리형 서비스인 Amazon ECS에는 AWS 구성과 운영 모범 사례가 내장되어 있습니다. Amazon Elastic Container Registry와 같은 AWS 도구 및 Docker와 같은 서드파티 도구와 통합됩니다. 이러한 통합을 통해 환경이 아닌 애플리케이션 구축에 더욱 집중할 수 있습니다. 컨트롤 플레인을 관리하는 복잡한 과정 없이 클라우드의 AWS 리전와 온프레미스에서 컨테이너 워크로드를 실행하고 확장할 수 있습니다.

용어 및 구성 요소

Amazon ECS에는 세 가지 계층이 있습니다.

- 용량 - 컨테이너가 실행되는 인프라입니다.
- 컨트롤러 - 컨테이너에서 실행되는 애플리케이션을 배포하고 관리합니다.
- 프로비저닝 - 스케줄러와 함께 애플리케이션 및 컨테이너를 배포 및 관리하는 데 사용할 수 있는 도구입니다.

다음 다이어그램은 Amazon ECS 계층을 나타냅니다.



용량은 컨테이너가 실행되는 인프라입니다. 다음은 용량 옵션에 대한 개요입니다.

- AWS 클라우드의 Amazon EC2 인스턴스

  인스턴스 유형, 인스턴스 수를 선택하고 용량을 관리합니다.

- AWS 클라우드의 서버리스(AWS Fargate)

  Fargate는 서버리스 종량제 컴퓨팅 엔진입니다. Fargate를 사용하면 서버를 관리하거나, 용량 계획을 처리하거나, 보안을 위해 컨테이너 워크로드를 격리할 필요가 없습니다.

- 온프레미스 가상 머신(VM) 또는 서버

  Amazon ECS Anywhere는 외부 인스턴스(예: 온프레미스 서버 또는 가상 머신(VM))을 Amazon ECS 클러스터에 등록하도록 지원합니다.

Amazon ECS 스케줄러는 애플리케이션을 관리하는 소프트웨어입니다.

Features

Amazon ECS는 다음의 대략적인 기능을 제공합니다.

태스크 정의

애플리케이션의 블루프린트입니다.

클러스터

애플리케이션이 실행되는 인프라입니다.

Task

작업을 수행한 다음 중지하는 배치 작업 같은 애플리케이션입니다.

Service

장기 실행되는 상태 비저장 애플리케이션입니다.

계정 설정

기능에 대한 액세스를 허용합니다.

클러스터 Auto Scaling

Amazon ECS는 클러스터에 등록된 Amazon EC2 인스턴스의 규모 조정을 관리합니다.

서비스 오토 스케일링

Amazon ECS는 서비스에서 원하는 태스크 수를 자동으로 늘리거나 줄입니다.

프로비저닝

Amazon ECS를 프로비저닝하기 위한 여러 가지 옵션이 있습니다.

- AWS Management Console - Amazon ECS 리소스 액세스에 사용할 수 있는 웹 인터페이스를 제공합니다.
- AWS Command Line Interface(AWS CLI) - Amazon ECS를 포함하여 다양한 AWS 서비스에 대한 명령을 제공합니다. Windows, Mac, Linux에서 지원됩니다. 자세한 정보는 AWS Command Line Interface을 참조하세요.
- AWS SDK - 언어별 API를 제공하고 많은 연결 세부 정보를 처리합니다. 서명 계산, 요청 재시도 처리 및 오류 처리가 여기에 포함됩니다. 자세한 내용은 AWS SDK를 참조하십시오.

- AWS CDK - 익숙한 프로그래밍 언어를 사용하여 클라우드 애플리케이션 리소스를 모델링하고 프로비저닝하는 데 사용할 수 있는 오픈 소스 소프트웨어 개발 프레임워크를 제공합니다. AWS CDK는 AWS CloudFormation을 통해 안전하고 반복 가능한 방식으로 리소스를 프로비저닝합니다.

요금

Amazon ECS 요금은 컨테이너에 대해 선택한 용량 옵션에 따라 달라집니다.

- Amazon ECS 요금 - Amazon ECS의 요금 정보
- AWS Fargate 요금 - Fargate의 요금 정보

관련 서비스

Amazon ECS와 함께 사용할 서비스

다른 AWS 서비스를 사용하여 Amazon ECS에 태스크 및 서비스를 배포할 수 있습니다.

Amazon EC2 Auto Scaling

애플리케이션의 로드를 처리할 수 있는 정확한 수의 Amazon EC2 인스턴스를 유지하는 데 도움이 됩니다.

Amazon CloudWatch

서비스 및 태스크를 모니터링합니다.

Amazon Elastic 컨테이너 레지스트리

컨테이너 이미지를 저장하고 관리합니다.

Elastic Load Balancing

수신 서비스 트래픽을 자동으로 분산합니다.

Amazon GuardDuty

컨테이너 인스턴스 및 워크로드의 잠재적 무단 사용 또는 악의적 사용을 탐지합니다.

Amazon ECS 리소스 생성 및 사용 방법 알아보기

다음 가이드에서는 Amazon ECS에 액세스할 수 있는 도구에 대한 소개와 컨테이너를 실행하기 위한 입문자용 절차를 제공합니다. Docker 기본 사항에서는 Docker 컨테이너 이미지를 생성하고 Amazon ECR 프라이빗 리포지토리에 업로드하는 기본 단계를 안내합니다. 시작 가이드에서는 AWS Copilot 명령줄 인터페이스 및 AWS Management Console을 사용하여 Amazon ECS 및 AWS Fargate에서 컨테이너를 실행하는 일반적인 태스크를 완료하는 방법을 안내합니다.

내용

- Amazon ECS 사용 설정
- Amazon ECS에서 사용할 컨테이너 이미지 생성
- Fargate 시작 유형에 대한 Amazon ECS Linux 작업을 생성하는 방법 알아보기
- Fargate 시작 유형에 대한 Amazon ECS Windows 작업을 생성하는 방법 알아보기
- EC2 시작 유형에 대한 Amazon ECS Windows 작업을 생성하는 방법 알아보기
- AWS CDK를 사용하여 Amazon ECS 리소스 생성
- AWS Copilot 명령줄 인터페이스를 사용하여 Amazon ECS 리소스 생성

Amazon ECS 사용 설정

Amazon Web Services(AWS)에 이미 가입했고 Amazon Elastic Compute Cloud (Amazon EC2)를 사용하고 있는 경우, Amazon ECS를 곧 사용할 수 있습니다. 두 서비스의 설정 프로세스는 유사합니다. 다음 가이드에서는 첫 번째 Amazon ECS 클러스터를 시작할 준비를 합니다.

Amazon ECS를 설정하려면 다음 태스크를 수행합니다.

AWS Management Console

AWS Management Console은 Amazon ECS 리소스를 관리하기 위한 브라우저 기반 인터페이스입니다. 콘솔은 서비스에 대한 시각적 개요를 제공하여 추가적인 도구 없이도 Amazon ECS의 기능을 쉽게 탐색할 수 있습니다.ls. 콘솔 사용 방법을 안내하는 여러 가지 관련 자습서와 시연이 준비되어 있습니다.

콘솔 사용법을 안내하는 자습서는 Amazon ECS 리소스 생성 및 사용 방법 알아보기 섹션을 참조하세요.

많은 고객이 처음 시작할 때는 콘솔을 사용하는 것을 선호합니다. 작업이 성공하는지 시각적 피드백을 통해 즉시 확인할 수 있기 때문입니다. AWS Management Console에 친숙한 AWS 고객은 로드 밸런서, Amazon EC2 인스턴스와 같은 관련 리소스를 손쉽게 관리할 수 있습니다.

AWS Management Console로 시작하세요.

AWS 계정에 등록

AWS 계정이 없는 경우 다음 절차에 따라 계정을 생성합니다.

AWS 계정에 가입

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   AWS 계정 루트 사용자에 가입하면 AWS 계정 루트 사용자가 만들어집니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS는 가입 절차 완료된 후 사용자에게 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자에 보안 조치를 한 다음, AWS IAM Identity Center을 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

1. 루트 사용자를 선택하고 AWS 계정이메일 주소를 입력하여 AWS Management Console에 계정 소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자용 가상 MFA 디바이스 활성화(콘솔)를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center 설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 기본 IAM Identity Center 디렉터리로 사용자 액세스 구성을 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자로 로그인하는 데 도움이 필요한 경우 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

Virtual Private Cloud 생성

Amazon Virtual Private Cloud(Amazon VPC)를 사용하여 사용자가 정의한 가상 네트워크로 AWS 리소스를 시작할 수 있습니다. VPC에서 컨테이너 인스턴스를 시작하는 것이 좋습니다.

기본 VPC가 있는 경우 이 섹션을 건너뛰고 보안 그룹 생성 작업으로 이동합니다. 기본 VPC가 있는지 결정하려면 Amazon VPC 사용 설명서의 기본 VPC와 기본 서브넷 작업을 참조하세요. 그렇지 않으면 아래 단계에 따라 계정에서 기본이 아닌 VPC를 생성할 수 있습니다.

VPC를 생성하는 방법에 대한 자세한 내용은 Amazon VPC 사용 설명서의 VPC 생성을 참조하고 다음 표를 사용하여 선택할 옵션을 결정하세요.

옵션	값
생성할 리소스	VPC 전용
명칭	선택적으로 VPC의 이름을 입력합니다.
IPv4 CIDR block	IPv4 CIDR 수동 입력 CIDR 블록 크기는 /16과 /28 사이여야 합니다.
IPv6 CIDR block	No IPv6 CIDR 블록
Tenancy	Default

Amazon VPC에 대한 자세한 내용은 Amazon VPC 사용 설명서의 Amazon VPC란 무엇인가? 섹션을 참조하세요.

보안 그룹 생성

보안 그룹은 연결된 컨테이너 인스턴스에 대한 방화벽 역할을 하여 컨테이너 인스턴스 수준에서 인바운드 트래픽과 아웃바운드 트래픽을 모두 제어합니다. 해당 IP 주소에서 SSH를 사용하여 컨테이너 인스턴스에 연결할 수 있게 하는 규칙을 보안 그룹에 추가할 수 있습니다. 어디서나 인바운드 및 아웃바운드 HTTP/HTTPS 액세스를 허용하는 규칙을 추가할 수도 있습니다. 작업에 필요한 포트를 여는 규칙을 추가합니다. 컨테이너 인스턴스는 Amazon ECS 서비스 엔드포인트와 통신하기 위해 외부 네트워크에 액세스해야 합니다.

여러 리전에서 컨테이너 인스턴스를 시작하려면 각 리전에서 보안 그룹을 생성해야 합니다. 자세한 내용은 Amazon EC2 사용 설명서에서 리전 및 가용 영역을 참조하세요.

> **Tip**
>
> 로컬 컴퓨터의 퍼블릭 IP 주소가 필요하며, 서비스를 사용해 확인할 수 있습니다. 예를 들면, Amazon에서는 http://checkip.amazonaws.com/ 또는 https://checkip.amazonaws.com/ 서비스를 제공합니다. IP 주소를 제공하는 다른 서비스를 찾으려면 "what is my IP address"로 검색합니다. 고정 IP 주소가 없는 방화벽 뒤나 ISP(인터넷 서비스 제공업체)를 통해 연결되어 있는 경우 클라이언트 컴퓨터가 사용하는 IP 주소의 범위를 찾아야 합니다.

보안 그룹을 생성하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스에 대한 보안 그룹 생성을 참조하고 다음 표를 사용하여 선택할 옵션을 결정하세요.

옵션	값
리전	키 페어를 생성한 리전과 동일.
명칭	기억하기 쉬운 이름(예: ecs-instances-default-cluster).
VPC	기본 VPC('(기본값)'으로 표시). **Note** 계정에서 Amazon EC2 Classic을 지원하는 경우 이전 작업에서 생성한 VPC를 선택합니다.

사용 사례에 추가할 아웃바운드 규칙에 대한 자세한 내용은 Amazon EC2 사용 설명서의 다양한 사용 사례에 대한 보안 그룹 규칙을 참조하세요.

Amazon ECS 컨테이너 인스턴스는 인바운드 포트를 열어야 할 필요가 없습니다. 하지만 컨테이너 인스턴스에 로그인하고 Docker 명령을 사용하여 태스크를 검사할 수 있도록 SSH 규칙을 추가할 수 있습니다. 또한 컨테이너 인스턴스에서 웹 서버를 실행하는 태스크를 호스팅하게 하려는 경우 HTTP 및 HTTPS에 대한 규칙을 추가할 수도 있습니다. 컨테이너 인스턴스는 Amazon ECS 서비스 엔드포인트와 통신하기 위해 외부 네트워크에 액세스해야 합니다. 이러한 보안 그룹 규칙을 필요에 따라 추가하려면 다음 절차를 수행합니다.

보안 그룹에 다음 세 가지 인바운드 규칙을 추가합니다. 보안 그룹을 생성하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 보안 그룹 규칙 구성을 참조하세요.

옵션	값	
HTTP 규칙	유형: HTTP 출처: 모든 소스(`0.0.0.0/0`) 이 옵션은 0.0.0.0/0 IPv4 CIDR 블록을 자동으로 소스로 추가합니다. 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 특정 IP 주소나 주소 범위만 인스턴스에 액세스하도록 허용하세요.	
HTTPS 규칙	유형: HTTPS 출처: 모든 소스(`0.0.0.0/0`) 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 특정 IP 주소나 주소 범위만 인스턴스에 액세스하도록 허용하세요.	
SSH 규칙	유형: SSH 소스(Source)에서 사용자 지정(Custom)을 선택하고 컴퓨터 또는 네트워크의 퍼블릭 IP 주소를 CIDR 표기법으로 지정합니다. 개별 IP 주소를 CIDR 표기법으로 지정하려면 라우팅 접두사 `/32`를 추	

옵션	값	
	가합니다. 예를 들어, IP 주소가 `203.0.113.25` 인 경우 `203.0.113.25/32` 를 지정합니다. 회사에서 주소를 범위로 할당하는 경우 전체 범위(예: `203.0.113.0/24`)를 지정합니다. **Important** 테스트를 위해 짧은 시간 동안만 허용하는 경우를 제외하고는 해당 인스턴스에 대한 모든 IP 주소(`0.0.0.0/0`) 로부터의 SSH 액세스를 허용하지 않는 것이 보안을 위해 좋습니다.	

EC2 인스턴스에 연결하기 위한 자격 증명 생성

Amazon ECS의 경우, 키 페어는 EC2 시작 유형을 사용할 때만 필요합니다.

AWS에서는 퍼블릭 키 암호화를 사용하여 인스턴스에 대한 로그인 정보를 보호합니다. Amazon ECS 컨테이너 인스턴스 같은 Linux 인스턴스는 SSH 액세스에 사용할 암호가 없습니다. 키 페어를 사용하면 인스턴스에 안전하게 로그인할 수 있습니다. 컨테이너 인스턴스를 시작할 때 키 페어의 이름을 지정한 다음 SSH를 사용하여 로그인할 때 프라이빗 키를 제공합니다.

키 페어를 아직 생성하지 않은 경우 Amazon EC2 콘솔을 사용하여 생성할 수 있습니다. 여러 리전에서 인스턴스를 시작하려면 각 리전에서 키 페어를 생성해야 합니다. 리전에 대한 자세한 내용은 Amazon EC2 사용 설명서의 리전 및 가용 영역을 참조하세요.

키 페어 생성

- Amazon EC2 콘솔을 사용하여 키 페어를 생성합니다. 키 페어 생성에 대한 자세한 내용은 Amazon EC2 사용 설명서의 키 페어 생성을 참조하세요.

인스턴스에 연결하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Linux 인스턴스에 연결을 참조하세요.

AWS CLI 설치

AWS Management Console은 Amazon ECS로 모든 태스크를 수동 관리하는 데 사용할 수 있습니다. 그러나 로컬 데스크톱이나 개발자 상자에 AWS CLI를 설치하면 Amazon ECS에서 일반적인 관리 작업을 자동화하는 스크립트를 빌드할 수도 있습니다.

Amazon ECS에 AWS CLI를 사용하려면 최신 AWS CLI 버전을 설치합니다. AWS CLI 설치 또는 최신 버전으로의 업그레이드 방법에 대한 자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS CLI 최신 버전 설치 또는 업데이트를 참조하세요.

AWS Command Line Interface(AWS CLI)는 AWS 서비스를 관리하는 데 사용하는 통합 도구입니다. 이 도구 하나만으로 여러 AWS 서비스를 제어하고 스크립트를 통해 서비스를 자동화할 수 있습니다. AWS CLI의 Amazon ECS 명령은 Amazon ECS API를 그대로 반영합니다.

AWS CLI는 명령줄 도구로 스크립트를 작성하고 연결하는 것을 선호하고 이에 익숙하면서도 Amazon ECS 리소스에서 어떤 태스크를 할지 명확히 알고 있는 고객에게 적합합니다. AWS CLI는 Amazon ECS API에 익숙해지고자 하는 고객에게도 유용합니다. 고객은 AWS CLI를 사용하여 명령줄 인터페이스에서 직접 생성, 읽기, 업데이트, 삭제 등을 비롯한 여러 가지 Amazon ECS 리소스 태스크를 수행할 수 있습니다.

Amazon ECS API와 해당 CLI 명령에 익숙하거나 익숙해지고 싶고, 자동 스크립트를 작성해 Amazon ECS 리소스에 특정한 태스크를 수행하고 싶다면 AWS CLI를 사용하세요.

AWS는 명령줄 도구 AWS Tools for Windows PowerShell도 제공합니다. 자세한 내용은 AWS Tools for Windows PowerShell 사용 설명서를 참조하세요.

Amazon ECS를 사용하기 위한 다음 단계

AWS CLI를 설치한 후 Amazon ECS를 계속 사용하면서 활용할 수 있는 다양한 도구가 있습니다. 다음 링크는 이러한 도구 중 일부에 대해 설명하고 Amazon ECS에서 이 도구를 사용하는 방법에 대한 예를 제공합니다.

- Docker로 첫 번째 컨테이너 이미지를 생성하고 Amazon ECS 작업 정의에 사용할 수 있도록 Amazon ECR로 푸시합니다.
- Fargate 시작 유형에 대한 Amazon ECS Linux 작업을 생성하는 방법 알아보기.
- Fargate 시작 유형에 대한 Amazon ECS Windows 작업을 생성하는 방법 알아보기.
- EC2 시작 유형에 대한 Amazon ECS Windows 작업을 생성하는 방법 알아보기.
- 원하는 프로그래밍 언어를 사용하여 AWS CDK를 사용하여 Amazon ECS 리소스 생성로 인프라 또는 아키텍처를 코드로 정의합니다.
- Amazon ECS에서 AWS CloudFormation 사용를 사용한 자동 배포로 환경 내 모든 AWS 리소스를 정의하고 관리합니다.
- 완전한 AWS Copilot 명령줄 인터페이스를 사용하여 Amazon ECS 리소스 생성 전체 개발자 워크플로로 인프라의 AWS 모범 사례를 준수하는 컨테이너 애플리케이션을 생성, 릴리스 및 운영할 수 있습니다.

Amazon ECS에서 사용할 컨테이너 이미지 생성

Amazon ECS는 작업 정의에 Docker 이미지를 사용하여 컨테이너를 시작합니다. Docker는 사용자가 컨테이너에서 분산 애플리케이션을 구축, 실행, 테스트 및 배포할 수 있는 도구를 제공합니다.

Amazon ECS는 컨테이너식 애플리케이션을 컨테이너 인스턴스 또는 AWS Fargate에 예약합니다. 컨테이너식 애플리케이션은 컨테이너 이미지로 패키징됩니다. 이 예제에서는 웹 서버의 컨테이너 이미지를 생성합니다.

첫 번째 Docker 이미지를 생성하고 해당 이미지를 Amazon ECS 태스크 정의에 사용하기 위해 컨테이너 레지스트리인 Amazon ECR에 푸시할 수 있습니다. 이 과정에서는 여러분이 Docker가 무엇인지 및 작동 방식에 대해 기본적인 이해를 하고 있다고 가정합니다. Docker에 대한 자세한 내용은 Docker란 무엇입니까?와 Docker 설명서를 참조하세요.

사전 조건

시작하기 전에 다음 사전 요구 사항을 충족하는지 확인합니다.

- Amazon ECR 설정 단계를 완료했는지 확인합니다.. 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 Amazon ECR에서의 수명주기를 따라 이미지 이동하기를 참조하세요.
- 사용자에게 Amazon ECR 서비스에 액세스하고 사용하는 데 필요한 IAM 권한이 있습니다. 자세한 내용은 Amazon ECR 관리형 정책을 참조하세요.

- 도커가 설치되어 있습니다. Amazon Linux 2023의 Docker 설치 단계에 대한 자세한 내용은 AL2023에 Docker 설치 섹션을 참조하세요. 다른 모든 운영 체제의 경우 Docker Desktop 개요에서 Docker 설명서를 참조하세요.
- AWS CLI를 설치하고 구성합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS CLI 최신 버전의 설치 또는 업데이트를 참조하세요.

로컬 개발 환경이 없거나 필요하지 않고 Amazon EC2 인스턴스를 사용하여 Docker를 사용하려는 경우 다음 단계를 수행하여 Amazon Linux 2023을 사용하는 Amazon EC2 인스턴스를 시작하고 Docker Engine과 Docker CLI를 설치합니다.

AL2023에 Docker 설치

Docker는 최신 Linux 배포 버전(Ubuntu 등)을 비롯하여 MacOS 및 Windows 등 다양한 운영 체제에서 사용할 수 있습니다. 특정 운영 체제에 Docker를 설치하는 방법에 대한 자세한 내용은 Docker 설치 안내서를 참조하십시오.

Docker를 사용하기 위해 로컬 개발 시스템이 필요하지 않습니다. Amazon EC2를 이미 사용 중인 경우 Amazon Linux 2023 인스턴스를 시작하고 Docker를 설치하여 시작할 수 있습니다.

이미 Docker가 설치되어 있으면 Docker 이미지 생성 단계로 건너뜁니다.

Amazon Linux 2023 AMI를 사용하는 Amazon EC2 인스턴스에 Docker를 설치하려면

1. 최신 Amazon Linux 2023 AMI에서 인스턴스를 시작합니다. 자세한 내용은 Amazon EC2 사용 설명서의 콘솔에서 인스턴스 시작 마법사를 사용하여 EC2 인스턴스 시작을 참조하세요.
2. 인스턴스에 연결합니다. 자세한 내용은 Amazon EC2 사용 설명서의 EC2 인스턴스에 연결을 참조하세요.
3. 인스턴스에 설치한 패키지 및 패키지 캐시를 업데이트합니다.

   ```
   sudo yum update -y
   ```

4. 최신 Docker Community Edition 패키지를 설치합니다.

   ```
   sudo yum install docker
   ```

5. Docker 서비스를 시작합니다.

   ```
   sudo service docker start
   ```

6. sudo를 사용하지 않고도 Docker 명령을 실행할 수 있도록 docker 그룹에 ec2-user를 추가합니다.

```
sudo usermod -a -G docker ec2-user
```

7. 로그아웃하고 다시 로그인해서 새 docker 그룹 권한을 선택합니다. 이를 위해 현재 SSH 터미널 창을 닫고 새 창에서 인스턴스를 다시 연결할 수 있습니다. 새 SSH 세션은 해당되는 docker 그룹 권한을 갖게 됩니다.
8. sudo 없이도 ec2-user가 Docker 명령을 실행할 수 있는지 확인합니다.

```
docker info
```

> **Note**
>
> 경우에 따라서는 ec2-user가 Docker 대몬에 액세스할 수 있는 권한을 제공하기 위해 인스턴스를 재부팅해야 할 수도 있습니다. 다음 오류가 표시될 경우 인스턴스를 재부팅합니다.
>
> ```
> Cannot connect to the Docker daemon. Is the docker daemon running on this
>  host?
> ```

Docker 이미지 생성

Amazon ECS 태스크 정의는 컨테이너 이미지를 사용하여 클러스터의 컨테이너 인스턴스에서 컨테이너를 시작합니다. 이 섹션에서는 간단한 웹 애플리케이션의 Docker 이미지를 생성하여 이를 로컬 시스템이나 Amazon EC2 인스턴스에서 테스트한 다음, Amazon ECR 컨테이너 레지스트리에 푸시하여 Amazon ECS 작업 정의에서 사용할 수 있도록 합니다.

간단한 웹 애플리케이션의 Docker 이미지를 생성하려면

1. Dockerfile이라는 파일을 생성합니다. Dockerfile은 Docker 이미지에 사용할 기본 이미지 및 이를 설치하고 실행할 항목을 설명하는 매니페스트입니다. Dockerfile에 대한 자세한 내용은 Dockerfile 참조를 참조하세요.

```
touch Dockerfile
```

2. 방금 만든 Dockerfile을 수정하고 다음 내용을 추가합니다.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest

# Update installed packages and install Apache
RUN yum update -y && \
 yum install -y httpd

# Write hello world message
RUN echo 'Hello World!' > /var/www/html/index.html

# Configure Apache
RUN echo 'mkdir -p /var/run/httpd' >> /root/run_apache.sh && \
 echo 'mkdir -p /var/lock/httpd' >> /root/run_apache.sh && \
 echo '/usr/sbin/httpd -D FOREGROUND' >> /root/run_apache.sh && \
 chmod 755 /root/run_apache.sh

EXPOSE 80

CMD /root/run_apache.sh
```

이 Dockerfile은 Amazon ECR 퍼블릭에서 호스팅되는 퍼블릭 Amazon Linux 2023 이미지를 사용합니다. RUN 지침은 패키지 캐시를 업데이트하고, 웹 서버의 일부 소프트웨어 패키지를 설치하고, 'Hello World!'를 작성합니다. 콘텐츠를 웹 서버 문서 루트에 작성합니다. EXPOSE 명령은 컨테이너의 포트 80에서 수신 대기 중임을 나타내며, CMD 명령은 웹 서버를 시작합니다.

3. Dockerfile에서 Docker 이미지를 빌드합니다.

> **Note**
>
> 아래의 명령에서 Docker의 일부 버전에서는 아래 보이는 상대 경로 대신에 Dockerfile의 전체 경로가 필요할 수 있습니다.
> Apple Silicon과 같은 ARM 기반 시스템에서 명령을 실행하는 경우 --platform 옵션인 '--platform linux/amd64'를 사용하세요.

```
docker build -t hello-world .
```

4. 컨테이너 이미지를 나열합니다.

```
docker images --filter reference=hello-world
```

출력:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
 SIZE
hello-world         latest              e9ffedc8c286        4 minutes ago
 194MB
```

5. 새로 빌드된 이미지를 실행합니다. `-p 80:80` 옵션은 컨테이너에 있는 노출된 포트 80을 호스트 시스템에 있는 포트 80에 매핑합니다.

```
docker run -t -i -p 80:80 hello-world
```

> **Note**
>
> Apache 웹 서버로부터의 출력이 터미널 창에 표시됩니다. "`Could not reliably determine the fully qualified domain name`" 메시지는 무시해도 됩니다.

6. 브라우저를 열고 Docker를 실행하고 컨테이너를 호스팅하고 있는 서버를 가리킵니다.

   - EC2 인스턴스를 사용하고 있는 경우 서버의 Public DNS 값이며, 이는 SSH로 인스턴스에 연결할 때 사용하는 주소와 동일합니다. 인스턴스의 보안 그룹에서 포트 80에 인바운드 트래픽을 허용해야 합니다.
   - Docker를 로컬에서 실행하고 있는 경우, 브라우저에서 http://localhost/를 가리킵니다.

   "Hello, World!" 문이 있는 웹 페이지가 표시됩니다.

7. Ctrl + c를 입력하여 Docker 컨테이너를 중지합니다.

Amazon Elastic 컨테이너 레지스트리에 이미지 푸시

Amazon ECR은 AWS 관리형 이미지 레지스트리 서비스입니다. Docker CLI를 사용하여 Amazon ECR 리포지토리에서 이미지를 푸시, 풀링 및 관리할 수 있습니다. Amazon ECR 제품 세부 정보, 주요 고객 사례 연구 및 FAQ에 대해서는 Amazon Elastic 컨테이너 레지스트리 제품 세부 정보 페이지를 참조하세요.

이미지에 태그를 지정하고 Amazon ECR에 푸시하려면

1. `hello-world` 이미지를 저장할 Amazon ECR 리포지토리를 생성합니다. 출력의 `repositoryUri`를 참고합니다.

   `region`을 AWS 리전(예: `us-east-1`)으로 바꿉니다.

   ```
   aws ecr create-repository --repository-name hello-repository --region region
   ```

   출력:

   ```
   {
       "repository": {
           "registryId": "aws_account_id",
           "repositoryName": "hello-repository",
           "repositoryArn": "arn:aws:ecr:region:aws_account_id:repository/hello-
   repository",
           "createdAt": 1505337806.0,
           "repositoryUri": "aws_account_id.dkr.ecr.region.amazonaws.com/hello-
   repository"
       }
   }
   ```

2. `hello-world` 이미지를 이전 단계의 `repositoryUri` 값으로 태그 지정합니다.

   ```
   docker tag hello-world aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository
   ```

3. aws ecr get-login-password 명령을 실행합니다. 인증할 레지스트리 URI를 지정합니다. 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 레지스트리 권한을 참조하세요.

   ```
   aws ecr get-login-password --region region | docker login --username AWS --
   password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
   ```

   출력:

   ```
   Login Succeeded
   ```

> **Important**
>
> 오류가 발생하면 최신 버전의 AWS CLI를 설치하거나 업그레이드합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS CLI 최신 버전의 설치 또는 업데이트를 참조하세요.

4. 이전 단계의 `repositoryUri` 값을 사용하여 Amazon ECR로 이미지를 푸시합니다.

```
docker push aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository
```

정리

Amazon ECS 작업 정의를 생성하고 컨테이너 이미지로 작업을 시작하려면 다음 단계(으)로 넘어가세요. Amazon ECR 이미지 실험이 끝나면 이미지 저장 비용을 물지 않도록 리포지토리를 삭제할 수 있습니다.

```
aws ecr delete-repository --repository-name hello-repository --region region --force
```

다음 단계

작업 정의에는 작업 실행 역할이 필요합니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

컨테이너 이미지를 생성하고 Amazon ECR로 푸시한 후 작업 정의에서 해당 이미지를 사용할 수 있습니다. 자세한 내용은 다음 중 하나를 참조하십시오.

- the section called “Fargate 시작 유형에 대한 작업을 생성하는 방법 알아보기”
- the section called “Fargate 시작 유형에 대한 Windows 작업을 생성하는 방법 알아보기”
- AWS CLI를 사용하여 Fargate 시작 유형에 대한 Amazon ECS Linux 작업 생성

Fargate 시작 유형에 대한 Amazon ECS Linux 작업을 생성하는 방법 알아보기

Amazon Elastic Container Service(Amazon ECS)는 컨테이너를 손쉽게 실행, 중지 및 관리할 수 있게 하는 컨테이너 관리 서비스로서 확장성과 속도가 뛰어납니다. AWS Fargate에서 서비스 또는 태스

크를 시작하여 Amazon ECS에서 관리하는 서버리스 인프라에서 컨테이너를 호스팅할 수 있습니다. Fargate에 대한 자세한 내용은 Amazon ECS용 AWS Fargate 섹션을 참조하세요.

Amazon ECS를 지원하는 리전에서 테스크에 대한 AWS Fargate 시작 유형을 사용하여 AWS Fargate에서의 Amazon ECS를 시작합니다.

AWS Fargate에서 Amazon ECS를 시작하려면 다음 단계를 완료합니다.

사전 조건

시작하기 전에 Amazon ECS 사용 설정의 단계를 완료하고 IAM 사용자에게 `AdministratorAccess` IAM 정책 예제에서 지정된 권한이 있는지 확인합니다.

콘솔에서는 Fargate 작업에 필요한 작업 실행 IAM 역할을 자동으로 생성하려고 합니다. 콘솔에서 이 IAM 역할을 생성할 수 있으려면 다음 중 하나가 충족되어야 합니다.

- 사용자에게 관리자 권한이 있습니다. 자세한 정보는 Amazon ECS 사용 설정을 참조하세요.
- 사용자에게 서비스 역할을 생성할 수 있는 IAM 권한이 있습니다. 자세한 내용은 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하세요.
- 관리자 권한이 있는 사용자가 작업 실행 역할을 수동으로 생성하였기 때문에 사용할 계정에서 사용할 수 있습니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

> **Important**
>
> 작업 정의를 사용하여 서비스를 생성할 때 선택하는 보안 그룹에는 인바운드 트래픽을 위해 포트 80이 열려 있어야 합니다. 다음 인바운드 규칙을 보안 그룹에 추가합니다. 보안 그룹을 생성하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스에 대한 보안 그룹 생성을 참조하세요.
>
> - 유형: HTTP
> - 프로토콜: TCP
> - 포트 범위: 80
> - 출처: 모든 소스(`0.0.0.0/0`)

1단계: 클러스터 생성

기본 VPC를 사용하는 클러스터를 생성합니다.

시작하기 전에 적절한 IAM 권한을 할당합니다. 자세한 내용은 the section called “Amazon ECS 클러스터 예제” 섹션을 참조하세요.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터(Clusters) 페이지에서 클러스터 생성(Create cluster)을 선택합니다.
5. 클러스터 구성(Cluster configuration) 아래의 클러스터 이름(Cluster name)에 고유한 이름을 입력합니다.

   이름은 최대 255자(대/소문자), 숫자 및 하이픈을 포함할 수 있습니다.
6. (선택 사항) Container Insights를 설정하려면 모니터링(Monitoring)을 확장한 다음 Container Insights 사용(Use Container Insights)을 설정합니다.
7. (선택 사항) 클러스터를 식별하려면 태그(Tags)를 펼친 다음, 태그를 구성합니다.

   [태그 추가] 태그 추가(Add tag)를 선택하고 다음을 수행합니다.

   - 키(Key)에 키 이름을 입력합니다.
   - 값에 키 값을 입력합니다.

   [태그 제거] 태그의 키와 값 오른쪽에 있는 제거를 선택합니다.
8. 생성(Create)을 선택합니다.

2단계: 태스크 정의 생성

태스크 정의는 애플리케이션에 대한 청사진과 같습니다. Amazon ECS에서 태스크를 시작할 때마다 태스크 정의를 지정합니다. 그래야만 컨테이너에 사용할 도커 이미지, 작업에서 사용할 컨테이너 수, 각 컨테이너의 리소스 할당을 서비스가 알 수 있습니다.

1. 탐색 창에서 태스크 정의를 선택합니다.
2. 새 태스크 정의 생성(Create new Task Definition), JSON으로 새 수정 생성(Create new revision with JSON)을 선택합니다.
3. 다음 예제 태스크 정의를 복사하여 상자에 붙여 넣은 다음 저장(Save)을 선택합니다.

```
{
    "family": "sample-fargate",
```

```
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "fargate-app",
            "image": "public.ecr.aws/docker/library/httpd:latest",
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                "sh",
  "-c"
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "256",
    "memory": "512"
}
```

4. 생성(Create)을 선택합니다.

3단계: 서비스 생성

태스크 정의를 사용하여 태스크를 생성합니다.

1. 탐색 창에서 클러스터를 선택하고 1단계: 클러스터 생성에서 생성한 클러스터를 선택합니다.
2. Services(서비스) 탭에서 Create(생성)를 선택합니다.

3. 배포 구성(Deployment configuration) 아래에서 애플리케이션 배포 방법을 지정합니다.

   a. 태스크 정의(Task definition)는 2단계: 태스크 정의 생성에서 생성한 태스크 정의를 선택합니다.

   b. 서비스 이름(Service name)에 서비스의 이름을 입력합니다.

   c. 원하는 태스크(Desired tasks)에 1을 입력합니다.

4. 네트워킹에서 작업을 위해 새 보안 그룹을 생성하거나 기존 보안 그룹을 선택할 수 있습니다. 사용하는 보안 그룹에 사전 조건에 나열된 인바운드 규칙이 있는지 확인합니다.

5. 생성(Create)을 선택합니다.

4단계: 서비스 보기

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 서비스를 실행한 클러스터를 선택합니다.
4. 서비스 탭의 서비스 이름에서 3단계: 서비스 생성에서 생성한 서비스를 선택합니다.
5. 작업 탭을 선택한 다음 서비스의 작업을 선택합니다.
6. 작업 페이지의 구성 섹션에 있는 퍼블릭 IP에서 공개 주소를 선택합니다.

5단계: 정리

Amazon ECS 클러스터 사용을 완료를 한 후에 사용하지 않는 리소스에 대해 요금이 발생하는 것을 방지하기 위해 연결된 리소스를 정리해야 합니다.

작업, 서비스, 클러스터, 컨테이너 인스턴스 같은 일부 Amazon ECS 서비스는 Amazon ECS 콘솔을 사용하여 정리됩니다. Amazon EC2 인스턴스, Elastic Load Balancing 로드 밸런서, Auto Scaling 그룹 같은 일부 리소스는 Amazon EC2 콘솔에서 수동으로 정리하거나 리소스를 만든 AWS CloudFormation 스택을 삭제하여 정리해야 합니다.

1. 탐색 창에서 클러스터를 선택합니다.
2. 클러스터 페이지에서 이 자습서용으로 생성한 클러스터를 선택합니다.
3. 서비스 탭을 선택합니다.
4. 서비스를 선택한 다음 삭제를 선택합니다.

5. 확인 프롬프트에서 delete를 입력한 다음, 삭제(Delete)를 선택합니다. 또는 `Force delete` 옵션을 사용하여 삭제하기 전에 Amazon ECS에서 사용자 대신 서비스 규모를 조정할 수 있습니다.

   서비스가 삭제될 때까지 기다립니다.

6. 클러스터 삭제(Delete Cluster)를 선택합니다. 확인 프롬프트에서 delete ***`cluster-name`***을 입력한 다음 삭제(Delete)를 선택합니다. 클러스터를 삭제하면 Auto Scaling 그룹, VPC 또는 로드 밸런서를 포함하여 클러스터로 생성된 관련 리소스가 정리됩니다.

Fargate 시작 유형에 대한 Amazon ECS Windows 작업을 생성하는 방법 알아보기

Amazon ECS를 지원하는 리전에서 테스크에 대한 AWS Fargate 시작 유형을 사용하여 AWS Fargate에서의 Amazon ECS를 시작합니다.

AWS Fargate에서 Amazon ECS를 시작하려면 다음 단계를 완료합니다.

사전 조건

시작하기 전에 Amazon ECS 사용 설정의 단계를 완료하고 IAM 사용자에게 `AdministratorAccess` IAM 정책 예제에서 지정된 권한이 있는지 확인합니다.

콘솔에서는 Fargate 작업에 필요한 작업 실행 IAM 역할을 자동으로 생성하려고 합니다. 콘솔에서 이 IAM 역할을 생성할 수 있으려면 다음 중 하나가 충족되어야 합니다.

- 사용자에게 관리자 권한이 있습니다. 자세한 정보는 Amazon ECS 사용 설정을 참조하세요.
- 사용자에게 서비스 역할을 생성할 수 있는 IAM 권한이 있습니다. 자세한 내용은 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하세요.
- 관리자 권한이 있는 사용자가 작업 실행 역할을 수동으로 생성하였기 때문에 사용할 계정에서 사용할 수 있습니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

> **Important**
>
> 작업 정의를 사용하여 서비스를 생성할 때 선택하는 보안 그룹에는 인바운드 트래픽을 위해 포트 80이 열려 있어야 합니다. 다음 인바운드 규칙을 보안 그룹에 추가합니다. 보안 그룹을 생성하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스에 대한 보안 그룹 생성을 참조하세요.

- 유형: HTTP
- 프로토콜: TCP
- 포트 범위: 80
- 출처: 모든 소스(`0.0.0.0/0`)

1단계: 클러스터 생성

기본 VPC를 사용하는 windows라는 이름의 새 클러스터를 생성할 수 있습니다.

AWS Management Console를 사용하여 클러스터를 생성하려면

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터(Clusters) 페이지에서 클러스터 생성(Create cluster)을 선택합니다.
5. 클러스터 구성(Cluster configuration)의 클러스터 이름(Cluster name)에 windows를 입력합니다.
6. (선택 사항) Container Insights를 설정하려면 모니터링(Monitoring)을 확장한 다음 Container Insights 사용(Use Container Insights)을 설정합니다.
7. (선택 사항) 클러스터를 식별하려면 태그(Tags)를 펼친 다음, 태그를 구성합니다.

   [태그 추가] 태그 추가(Add tag)를 선택하고 다음을 수행합니다.

   - 키(Key)에 키 이름을 입력합니다.
   - 값에 키 값을 입력합니다.

   [태그 제거] 태그의 키와 값 오른쪽에 있는 제거를 선택합니다.
8. 생성(Create)을 선택합니다.

2단계: Windows 태스크 정의 등록

Amazon ECS 클러스터에서 Windows 컨테이너를 실행하려면 먼저 태스크 정의를 등록해야 합니다. 다음 태스크 정의 예시는 `mcr.microsoft.com/windows/servercore/iis` 컨테이너 이미지가 포함된 컨테이너 인스턴스의 포트 8080에 있는 간단한 웹페이지를 보여 줍니다.

AWS Management Console을 사용하여 샘플 태스크 정의를 등록하려면

1. 탐색 창에서 태스크 정의를 선택합니다.
2. 새 태스크 정의 생성(Create new task definition), JSON으로 새 태스크 정의 생성(Create new task definition with JSON)을 선택합니다.
3. 다음 예제 태스크 정의를 복사하여 상자에 붙여 넣은 다음 저장(Save)을 선택합니다.

```
{
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html
 -Type file -Value '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
 <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p>'; C:\\ServiceMonitor.exe w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "networkMode": "awsvpc",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["FARGATE"]
```

```
}
```

4. 정보를 확인한 후 생성(Create)을 선택합니다.

3단계: 태스크 정의를 사용하여 서비스 생성

태스크 정의를 등록한 후 태스크 정의를 사용하여 클러스터에 태스크를 배치할 수 있습니다. 다음 절차에서는 작업 정의를 사용하여 서비스를 생성하고 작업 하나를 클러스터에 배치합니다.

콘솔을 사용하여 태스크 정의에서 서비스를 생성하려면

1. 탐색 창에서 클러스터를 선택하고 1단계: 클러스터 생성에서 생성한 클러스터를 선택합니다.
2. Services(서비스) 탭에서 Create(생성)를 선택합니다.
3. 배포 구성(Deployment configuration) 아래에서 애플리케이션 배포 방법을 지정합니다.
   a. 태스크 정의(Task definition)는 2단계: Windows 태스크 정의 등록에서 생성한 태스크 정의를 선택합니다.
   b. 서비스 이름(Service name)에 서비스의 이름을 입력합니다.
   c. 원하는 태스크(Desired tasks)에 1을 입력합니다.
4. 네트워킹에서 새 보안 그룹을 생성하거나 기존 그룹을 선택할 수 있습니다. 사용하는 보안 그룹에 사전 조건에 나열된 인바운드 규칙이 있는지 확인합니다.
5. 생성(Create)을 선택합니다.

4단계: 서비스 보기

서비스가 태스크를 클러스터로 시작한 후 브라우저에서 서비스를 보고 IIS 테스트 페이지를 열어 컨테이너가 실행 중인지 확인할 수 있습니다.

> **Note**
>
> 컨테이너 인스턴스가 Windows 컨테이너 기본 계층을 다운로드하고 추출하는 데는 최대 15분 정도 걸릴 수 있습니다.

서비스를 보려면

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.

2. 탐색 창에서 클러스터를 선택합니다.
3. 서비스를 실행한 클러스터를 선택합니다.
4. 서비스 탭의 서비스 이름에서 3단계: 태스크 정의를 사용하여 서비스 생성에서 생성한 서비스를 선택합니다.
5. 작업 탭을 선택한 다음 서비스의 작업을 선택합니다.
6. 작업 페이지의 구성 섹션에 있는 퍼블릭 IP에서 공개 주소를 선택합니다.

5단계: 정리

Amazon ECS 클러스터 사용을 완료를 한 후에 사용하지 않는 리소스에 대해 요금이 발생하는 것을 방지하기 위해 연결된 리소스를 정리해야 합니다.

작업, 서비스, 클러스터, 컨테이너 인스턴스 같은 일부 Amazon ECS 서비스는 Amazon ECS 콘솔을 사용하여 정리됩니다. Amazon EC2 인스턴스, Elastic Load Balancing 로드 밸런서, Auto Scaling 그룹 같은 일부 리소스는 Amazon EC2 콘솔에서 수동으로 정리하거나 리소스를 만든 AWS CloudFormation 스택을 삭제하여 정리해야 합니다.

1. 탐색 창에서 클러스터를 선택합니다.
2. 클러스터 페이지에서 이 자습서용으로 생성한 클러스터를 선택합니다.
3. 서비스 탭을 선택합니다.
4. 서비스를 선택한 다음 삭제를 선택합니다.
5. 확인 프롬프트에서 delete를 입력한 다음, 삭제(Delete)를 선택합니다.

   서비스가 삭제될 때까지 기다립니다.
6. 클러스터 삭제(Delete Cluster)를 선택합니다. 확인 프롬프트에서 delete ***cluster-name***을 입력한 다음 삭제(Delete)를 선택합니다. 클러스터를 삭제하면 Auto Scaling 그룹, VPC 또는 로드 밸런서를 포함하여 클러스터로 생성된 관련 리소스가 정리됩니다.

EC2 시작 유형에 대한 Amazon ECS Windows 작업을 생성하는 방법 알아보기

작업 정의를 등록하고 클러스터를 생성하며 콘솔에서 서비스를 생성하여 EC2 시작 유형으로 Amazon ECS를 시작합니다.

EC2 시작 유형을 사용하여 Amazon ECS를 시작하려면 다음 단계를 완료합니다.

사전 조건

시작하기 전에 Amazon ECS 사용 설정의 단계를 완료하고 IAM 사용자에게 `AdministratorAccess` IAM 정책 예제에서 지정된 권한이 있는지 확인합니다.

콘솔에서는 Fargate 작업에 필요한 작업 실행 IAM 역할을 자동으로 생성하려고 합니다. 콘솔에서 이 IAM 역할을 생성할 수 있으려면 다음 중 하나가 충족되어야 합니다.

- 사용자에게 관리자 권한이 있습니다. 자세한 정보는 Amazon ECS 사용 설정을 참조하세요.
- 사용자에게 서비스 역할을 생성할 수 있는 IAM 권한이 있습니다. 자세한 내용은 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하세요.
- 관리자 권한이 있는 사용자가 작업 실행 역할을 수동으로 생성하였기 때문에 사용할 계정에서 사용할 수 있습니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

Important

작업 정의를 사용하여 서비스를 생성할 때 선택하는 보안 그룹에는 인바운드 트래픽을 위해 포트 80이 열려 있어야 합니다. 다음 인바운드 규칙을 보안 그룹에 추가합니다. 보안 그룹을 생성하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스에 대한 보안 그룹 생성을 참조하세요.

- 유형: HTTP
- 프로토콜: TCP
- 포트 범위: 80
- 출처: 모든 소스(`0.0.0.0/0`)

1단계: 클러스터 생성

Amazon ECS 클러스터는 작업, 서비스 및 컨테이너 인스턴스의 논리적 그룹입니다.

다음 단계에서는 하나의 Amazon EC2 인스턴스가 등록되어 있는 클러스터를 생성하는 절차를 단계별로 살펴봅니다. 생성된 클러스터에서 태스크를 실행할 수 있습니다. 특정 필드가 언급되지 않은 경우 콘솔 기본값은 그대로 둡니다.

새 클러스터 생성(Amazon ECS 콘솔)

시작하기 전에 적절한 IAM 권한을 할당합니다. 자세한 내용은 the section called “Amazon ECS 클러스터 예제” 섹션을 참조하세요.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터(Clusters) 페이지에서 클러스터 생성(Create cluster)을 선택합니다.
5. 클러스터 구성(Cluster configuration) 아래의 클러스터 이름(Cluster name)에 고유한 이름을 입력합니다.

   이름은 최대 255자(대/소문자), 숫자 및 하이픈을 포함할 수 있습니다.
6. (선택 사항) 태스크와 서비스가 시작되는 VPC와 서브넷을 변경하려면 네트워킹(Networking)에서 다음 태스크 중 하나를 수행합니다.

   - 서브넷을 제거하려면서브넷(Subnets)에서 제거하려는 각 서브넷에 대해 X를 선택합니다.
   - 기본(default) VPC가 아닌 다른 VPC로 변경하려면 VPC에서 기존 VPC를 선택한 다음 서브넷(Subnets)에서 각 서브넷을 선택합니다.
7. Amazon EC2 인스턴스를 클러스터에 추가하려면 인프라를 확장한 다음 Amazon EC2 인스턴스를 선택합니다. 다음으로 용량 공급자 역할을 하는 Auto Scaling 그룹을 구성합니다.

   a. 기존 Auto Scaling 그룹을 사용하려면 Auto Scaling 그룹(ASG)(Auto Scaling group (ASG))에서 해당 그룹을 선택합니다.

   b. Auto Scaling 그룹을 생성하려면 Auto Scaling 그룹(ASG)(Auto Scaling group (ASG))에서 새 그룹 생성(Create new group)을 선택한 후 그룹에 대한 다음 세부 정보를 제공합니다.

      - 운영 체제/아키텍처(Operating system/Architecture)에서 Auto Scaling 그룹 인스턴스에 대한 Amazon ECS 최적화 AMI를 선택합니다.
      - EC2 인스턴스 유형(EC2 instance type)에서 워크로드의 인스턴스 유형을 선택합니다. 지원되는 인스턴스 유형에 대한 자세한 내용은 Amazon EC2 인스턴스 유형을 참조하세요.

        관리형 확장은 Auto Scaling 그룹에서 동일하거나 유사한 인스턴스 유형을 사용해야 가장 효과적입니다.
      - SSH 키 페어(SSH key pair)에서 인스턴스에 연결할 때 자격 증명을 증명하는 페어를 선택합니다.

- 용량(Capacity)에 Auto Scaling 그룹에서 시작할 최소 인스턴스 수와 최대 인스턴스 수를 입력합니다. Amazon EC2 인스턴스가 AWS 리소스에 존재하는 동안에는 비용이 발생합니다. 자세한 내용은 Amazon EC2 요금을 참조하세요.

8. (선택 사항) Container Insights를 설정하려면 모니터링(Monitoring)을 확장한 다음 Container Insights 사용(Use Container Insights)을 설정합니다.
9. (선택 사항) 클러스터 태그를 관리하려면 태그(Tags)를 확장하고 다음 작업 중 하나를 수행합니다.

   [태그 추가] 태그 추가(Add tag)를 선택하고 다음을 수행합니다.

   - 키(Key)에 키 이름을 입력합니다.
   - 값에 키 값을 입력합니다.

   [태그 제거] 태그의 키와 값 오른쪽에 있는 제거를 선택합니다.
10. 생성(Create)을 선택합니다.

2단계: 태스크 정의 등록

AWS Management Console을 사용하여 샘플 태스크 정의를 등록하려면

1. 탐색 창에서 태스크 정의를 선택합니다.
2. 새 태스크 정의 생성(Create new task definition), JSON으로 새 태스크 정의 생성(Create new task definition with JSON)을 선택합니다.
3. 다음 예제 작업 정의를 복사하여 상자에 붙여 넣은 다음 저장을 선택합니다.

```
{
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html
 -Type file -Value '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
 <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p>'; C:\\ServiceMonitor.exe w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
```

```
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 443,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["EC2"]
}
```

4. 정보를 확인한 후 생성(Create)을 선택합니다.

3단계: 서비스 생성

Amazon ECS 서비스를 사용하면 Amazon ECS 클러스터에서 지정된 수의 태스크 정의 인스턴스를 동시에 실행하고 관리할 수 있습니다. 어떤 이유로 작업이 실패 또는 중지되는 경우 Amazon ECS 서비스 스케줄러가 태스크 정의의 다른 인스턴스를 시작하여 이를 대체하고 서비스의 원하는 작업 수를 유지합니다. 서비스에 대한 자세한 내용은 Amazon ECS 서비스 섹션을 참조하세요.

서비스 생성

1. 탐색 창에서 클러스터를 선택합니다.
2. 1단계: 클러스터 생성에서 생성한 클러스터를 선택합니다.
3. 서비스(Services) 탭에서 생성(Create)을 선택합니다.
4. 환경(Environment) 섹션에서 다음 작업을 수행합니다.
   a. 컴퓨팅 옵션(Compute options)에서 시작 유형(Launch type)을 선택합니다.
   b. 시작 유형(Launch type)에서 EC2를 선택합니다.

5. 배포 구성(Deployment configuration) 섹션에서 다음을 수행합니다.
   a. 패밀리(Family)는 2단계: 태스크 정의 등록에서 생성한 태스크 정의를 선택합니다.
   b. 서비스 이름(Service name)에 서비스의 이름을 입력합니다.
   c. 원하는 태스크(Desired tasks)에 1을 입력합니다.
6. 옵션을 검토하고 Create(생성)를 선택합니다.
7. 서비스 보기(View service)를 선택하여 서비스를 검토합니다.

4단계: 서비스 보기

이 서비스는 웹 기반 애플리케이션이므로 웹 브라우저에서 컨테이너를 볼 수 있습니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 서비스를 실행한 클러스터를 선택합니다.
4. 서비스 탭의 서비스 이름에서 3단계: 서비스 생성에서 생성한 서비스를 선택합니다.
5. 작업 탭을 선택한 다음 서비스의 작업을 선택합니다.
6. 작업 페이지의 구성 섹션에 있는 퍼블릭 IP에서 공개 주소를 선택합니다. 아래 스크린샷은 예상 출력입니다.

5단계: 정리

Amazon ECS 클러스터 사용을 완료를 한 후에 사용하지 않는 리소스에 대해 요금이 발생하는 것을 방지하기 위해 연결된 리소스를 정리해야 합니다.

작업, 서비스, 클러스터, 컨테이너 인스턴스 같은 일부 Amazon ECS 서비스는 Amazon ECS 콘솔을 사용하여 정리됩니다. Amazon EC2 인스턴스, Elastic Load Balancing 로드 밸런서, Auto Scaling 그룹 같은 일부 리소스는 Amazon EC2 콘솔에서 수동으로 정리하거나 리소스를 만든 AWS CloudFormation 스택을 삭제하여 정리해야 합니다.

1. 탐색 창에서 클러스터를 선택합니다.
2. 클러스터 페이지에서 이 자습서용으로 생성한 클러스터를 선택합니다.
3. 서비스 탭을 선택합니다.
4. 서비스를 선택한 다음 삭제를 선택합니다.
5. 확인 프롬프트에서 delete를 입력한 다음, 삭제(Delete)를 선택합니다.

   서비스가 삭제될 때까지 기다립니다.
6. 클러스터 삭제(Delete Cluster)를 선택합니다. 확인 프롬프트에서 delete ***cluster-name***을 입력한 다음 삭제(Delete)를 선택합니다. 클러스터를 삭제하면 Auto Scaling 그룹, VPC 또는 로드 밸런서를 포함하여 클러스터로 생성된 관련 리소스가 정리됩니다.

AWS CDK를 사용하여 Amazon ECS 리소스 생성

AWS 클라우드 개발 키트 (AWS CDK)는 코드형 인프라(IAC) 프레임워크로 선택한 프로그래밍 언어를 사용하여 AWS 클라우드 인프라를 정의할 수 있습니다. 자체 클라우드 인프라를 정의하려면 먼저 하나 이상의 스택을 포함하는 앱(CDK 지원 언어 중 하나로)을 작성해야 합니다. 그런 다음 합성하여 AWS CloudFormation 템플릿을 만들고 리소스를 AWS 계정에 배포합니다. 이 주제의 절차를 따라서 Amazon Elastic Container Service(Amazon ECS)와 AWS CDK를 사용하여 컨테이너식 웹 서버를 Fargate에 배포할 수 있습니다.

CDK에 포함된 AWS Construct Library는 AWS 서비스가 제공하는 리소스를 모델링하는 데 사용할 수 있는 모듈을 제공합니다. 많이 사용되는 서비스의 라이브러리는 스마트 기본값과 모범 사례를 통해 큐레이트된 구조를 제공합니다. 특히 이 모듈 중 하나인 `aws-ecs-patterns`는 컨테이너화된 서비스와 필요한 모든 지원 리소스를 몇 줄의 코드로 정의하는 데 사용할 수 있는 상위 수준의 추상화를 제공합니다.

이 주제에서는 `ApplicationLoadBalancedFargateService` 구조를 사용합니다. 이 구조는 애플리케이션 로드 밸런서 뒤의 Fargate에 Amazon ECS 서비스를 배포합니다. 또한 `aws-ecs-patterns` 모듈에는 네트워크 로드 밸런서를 사용하고 Amazon EC2를 실행하는 구조도 포함되어 있습니다.

이 작업을 시작하기 전에 AWS CDK 개발 환경을 구축하고 다음 명령을 실행하여 AWS CDK를 설치합니다. AWS CDK 개발 환경 설정 방법에 대한 지침은 AWS CDK로 시작하기 - 사전 조건을 참조하세요.

```
npm install -g aws-cdk
```

> **Note**
>
> 이 지침에서는 AWS CDK v2를 사용하고 있다고 가정합니다.

주제

- 1단계: AWS CDK 프로젝트 설정
- 2단계: AWS CDK를 사용하여 Fargate에 컨테이너화된 웹 서버 정의
- 3단계: 웹 서비스 테스트
- 4단계: 정리
- 다음 단계

1단계: AWS CDK 프로젝트 설정

새 AWS CDK 앱에 대한 디렉터리를 생성하고 프로젝트를 초기화합니다.

TypeScript

```
mkdir hello-ecs
cd hello-ecs
cdk init --language typescript
```

JavaScript

```
mkdir hello-ecs
cd hello-ecs
cdk init --language javascript
```

Python

```
mkdir hello-ecs
cd hello-ecs
cdk init --language python
```

프로젝트가 시작되면 프로젝트의 가상 환경을 활성화하고 AWS CDK의 기준선 종속성을 설치합니다.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
mkdir hello-ecs
cd hello-ecs
cdk init --language java
```

이 Maven 프로젝트를 Java IDE로 가져옵니다. 예를 들어 Eclipse에서 파일 > 가져오기 > Maven > 기존 Maven 프로젝트를 사용합니다.

C#

```
mkdir hello-ecs
cd hello-ecs
cdk init --language csharp
```

Go

```
mkdir hello-ecs
cd hello-ecs
cdk init --language go
```

> **Note**
>
> AWS CDK 애플리케이션 템플릿은 프로젝트 디렉토리의 이름을 사용하여 소스 파일 및 클래스의 이름을 생성합니다. 이 예에서 디렉터리 이름은 `hello-ecs`입니다. 다른 프로젝트 디렉터리 이름을 사용하는 경우 앱이 이 지침과 일치하지 않습니다.

AWS CDK v2에는 `aws-cdk-lib`라는 단일 패키지에 모든 AWS 서비스에 대한 안정적인 구성이 포함되어 있습니다. 이 패키지는 프로젝트를 초기화하거나 종속 항목으로 설치됩니다. 특정 프로그래밍 언어로 작업하는 경우 프로젝트를 처음 빌드할 때 패키지가 설치됩니다. 이 주제에서는 Amazon ECS 작업을 위한 높은 수준의 추상화를 제공하는 Amazon ECS 패턴 구문을 사용하는 방법을 설명합니다. 이

모듈은 Amazon ECS 구조 및 다른 구조를 사용하여 Amazon ECS 애플리케이션에 필요한 리소스를 프로비저닝합니다.

이러한 라이브러리를 CDK 애플리케이션으로 가져오는 데 사용하는 이름은 사용하는 프로그래밍 언어에 따라 약간 다릅니다. 참고로 다음은 지원되는 각 CDK 프로그래밍 언어에서 사용되는 이름입니다.

TypeScript

```
aws-cdk-lib/aws-ecs
aws-cdk-lib/aws-ecs-patterns
```

JavaScript

```
aws-cdk-lib/aws-ecs
aws-cdk-lib/aws-ecs-patterns
```

Python

```
aws_cdk.aws_ecs
aws_cdk.aws_ecs_patterns
```

Java

```
software.amazon.awscdk.services.ecs
software.amazon.awscdk.services.ecs.patterns
```

C#

```
Amazon.CDK.AWS.ECS
Amazon.CDK.AWS.ECS.Patterns
```

Go

```
github.com/aws/aws-cdk-go/awscdk/v2/awsecs
github.com/aws/aws-cdk-go/awscdk/v2/awsecspatterns
```

2단계: AWS CDK를 사용하여 Fargate에 컨테이너화된 웹 서버 정의

컨테이너 이미지 amazon-ecs-sample를 사용합니다. 이 이미지에는 Ngingx에서 실행되는 PHP 웹 앱이 포함되어 있습니다.

생성한 AWS CDK 프로젝트의 경우 스택 정의가 포함된 파일을 다음 예제 중 하나와 비슷하게 편집합니다.

> **Note**
>
> 스택은 배포 단위입니다. 모든 리소스는 스택에 있어야 하며 스택의 모든 리소스는 동시에 배포됩니다. 리소스를 배포하지 못하면 이미 배포된 다른 모든 리소스가 롤백됩니다. AWS CDK 앱은 여러 스택을 포함할 수 있으며 한 스택의 리소스는 다른 스택의 리소스를 참조할 수 있습니다.

TypeScript

다음과 유사하도록 `lib/hello-ecs-stack.ts`을(를) 업데이트합니다.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as ecs from 'aws-cdk-lib/aws-ecs';
import * as ecsp from 'aws-cdk-lib/aws-ecs-patterns';

export class HelloEcsStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    new ecsp.ApplicationLoadBalancedFargateService(this, 'MyWebServer', {
      taskImageOptions: {
        image: ecs.ContainerImage.fromRegistry('public.ecr.aws/ecs-sample-image/
amazon-ecs-sample:latest'),
      },
      publicLoadBalancer: true
    });
  }
}
```

JavaScript

다음과 유사하도록 `lib/hello-ecs-stack.js`을(를) 업데이트합니다.

```
const cdk = require('aws-cdk-lib');
const { Construct } = require('constructs');
const ecs = require('aws-cdk-lib/aws-ecs');
const ecsp = require('aws-cdk-lib/aws-ecs-patterns');
```

```
class HelloEcsStack extends cdk.Stack {
  constructor(scope = Construct, id = string, props = cdk.StackProps) {
    super(scope, id, props);

    new ecsp.ApplicationLoadBalancedFargateService(this, 'MyWebServer', {
      taskImageOptions: {
        image: ecs.ContainerImage.fromRegistry('amazon/amazon-ecs-sample'),
      },
      publicLoadBalancer: true
    });
  }
}

module.exports = { HelloEcsStack }
```

Python

다음과 유사하도록 `hello-ecs/hello_ecs_stack.py`을(를) 업데이트합니다.

```
import aws_cdk as cdk
from constructs import Construct

import aws_cdk.aws_ecs as ecs
import aws_cdk.aws_ecs_patterns as ecsp

class HelloEcsStack(cdk.Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        ecsp.ApplicationLoadBalancedFargateService(self, "MyWebServer",
            task_image_options=ecsp.ApplicationLoadBalancedTaskImageOptions(
                image=ecs.ContainerImage.from_registry("amazon/amazon-ecs-sample")),
            public_load_balancer=True
        )
```

Java

다음과 유사하도록 `src/main/java/com.myorg/HelloEcsStack.java`을(를) 업데이트합니다.

```
package com.myorg;
```

```
import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;

import software.amazon.awscdk.services.ecs.ContainerImage;
import
 software.amazon.awscdk.services.ecs.patterns.ApplicationLoadBalancedFargateService;
import
 software.amazon.awscdk.services.ecs.patterns.ApplicationLoadBalancedTaskImageOptions;

public class HelloEcsStack extends Stack {
    public HelloEcsStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloEcsStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        ApplicationLoadBalancedFargateService.Builder.create(this, "MyWebServer")
         .taskImageOptions(ApplicationLoadBalancedTaskImageOptions.builder()
           .image(ContainerImage.fromRegistry("amazon/amazon-ecs-sample"))
           .build())
         .publicLoadBalancer(true)
         .build();
    }
}
```

C#

다음과 유사하도록 `src/HelloEcs/HelloEcsStack.cs`을(를) 업데이트합니다.

```
using Amazon.CDK;
using Constructs;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
namespace HelloEcs
{
    public class HelloEcsStack : Stack
    {
        internal HelloEcsStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
```

```
            new ApplicationLoadBalancedFargateService(this, "MyWebServer",
                new ApplicationLoadBalancedFargateServiceProps
                {
                    TaskImageOptions = new ApplicationLoadBalancedTaskImageOptions
                    {
                        Image = ContainerImage.FromRegistry("amazon/amazon-ecs-
sample")
                    },
                    PublicLoadBalancer = true
                });
        }
    }
}
```

Go

다음과 유사하도록 hello-ecs.go을(를) 업데이트합니다.

```
package main

import (
 "github.com/aws/aws-cdk-go/awscdk/v2"
 // "github.com/aws/aws-cdk-go/awscdk/v2/awssqs"
 "github.com/aws/aws-cdk-go/awscdk/v2/awsecs"
 "github.com/aws/aws-cdk-go/awscdk/v2/awsecspatterns"
 "github.com/aws/constructs-go/constructs/v10"
 "github.com/aws/jsii-runtime-go"
)

type HelloEcsStackProps struct {
 awscdk.StackProps
}

func NewHelloEcsStack(scope constructs.Construct, id string, props
 *HelloEcsStackProps) awscdk.Stack {
 var sprops awscdk.StackProps
 if props != nil {
  sprops = props.StackProps
 }
 stack := awscdk.NewStack(scope, &id, &sprops)

 // The code that defines your stack goes here

 // example resource
```

```
 // queue := awssqs.NewQueue(stack, jsii.String("HelloEcsQueue"),
 &awssqs.QueueProps{
 //  VisibilityTimeout: awscdk.Duration_Seconds(jsii.Number(300)),
 // })
 res := awsecspatterns.NewApplicationLoadBalancedFargateService(stack,
 jsii.String("MyWebServer"),
  &awsecspatterns.ApplicationLoadBalancedFargateServiceProps{
   TaskImageOptions: &awsecspatterns.ApplicationLoadBalancedTaskImageOptions{
    Image: awsecs.ContainerImage_FromRegistry(jsii.String("amazon/amazon-ecs-
sample"), &awsecs.RepositoryImageProps{}),
   },
  },
 )
 awscdk.NewCfnOutput(stack, jsii.String("LoadBalancerDNS"),
 &awscdk.CfnOutputProps{Value: res.LoadBalancer().LoadBalancerDnsName()})

 return stack
}

func main() {
 defer jsii.Close()

 app := awscdk.NewApp(nil)

 NewHelloEcsStack(app, "HelloEcsStack", &HelloEcsStackProps{
  awscdk.StackProps{
   Env: env(),
  },
 })

 app.Synth(nil)
}

// env determines the AWS environment (account+region) in which our stack is to
// be deployed. For more information see: https://docs.aws.amazon.com/cdk/latest/
guide/environments.html
func env() *awscdk.Environment {
 // If unspecified, this stack will be "environment-agnostic".
 // Account/Region-dependent features and context lookups will not work, but a
 // single synthesized template can be deployed anywhere.
 //---------------------------------------------------------------------------
 return nil

 // Uncomment if you know exactly what account and region you want to deploy
```

```
 // the stack to. This is the recommendation for production stacks.
 //---------------------------------------------------------------------------
 // return &awscdk.Environment{
 //  Account: jsii.String("123456789012"),
 //  Region:  jsii.String("us-east-1"),
 // }

 // Uncomment to specialize this stack for the AWS Account and Region that are
 // implied by the current CLI configuration. This is recommended for dev
 // stacks.
 //---------------------------------------------------------------------------
 // return &awscdk.Environment{
 //  Account: jsii.String(os.Getenv("CDK_DEFAULT_ACCOUNT")),
 //  Region:  jsii.String(os.Getenv("CDK_DEFAULT_REGION")),
 // }
}
```

위의 짧은 스니펫에는 다음이 포함됩니다.

- 서비스의 논리적 이름: `MyWebServer`.
- Amazon ECR 퍼블릭 갤러리에서 가져온 컨테이너 이미지: `amazon/amazon-ecs-sample`.
- 로드 밸런서에 퍼블릭 주소가 있으므로 인터넷에서 액세스할 수 있다는 사실 등 기타 관련 정보.

AWS CDK에서는 다음 리소스를 포함하여 웹 서버를 배포하는 데 필요한 모든 리소스를 만듭니다. 이 예제에서는 이러한 리소스가 생략되었습니다.

- Amazon ECS 클러스터
- Amazon VPC 및 Amazon EC2 인스턴스
- Auto Scaling 그룹
- Application Load Balancer
- IAM 역할 및 정책

자동 프로비저닝된 일부 리소스는 스택에 정의된 모든 Amazon ECS 서비스에서 공유됩니다.

소스 파일을 저장한 다음 애플리케이션의 기본 디렉터리에서 `cdk synth` 명령을 실행합니다. AWS CDK가 앱을 실행하고 앱에서 AWS CloudFormation 템플릿을 합성한 다음 템플릿을 표시합니다. 템플릿은 약 600줄의 YAML 파일입니다. 파일의 시작 부분이 여기에 표시됩니다. 템플릿이 이 예제와 다를 수 있습니다.

```
Resources:
  MyWebServerLB3B5FD3AB:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      LoadBalancerAttributes:
        - Key: deletion_protection.enabled
          Value: "false"
      Scheme: internet-facing
      SecurityGroups:
        - Fn::GetAtt:
            - MyWebServerLBSecurityGroup01B285AA
            - GroupId
      Subnets:
        - Ref: EcsDefaultClusterMnL3mNNYNVpcPublicSubnet1Subnet3C273B99
        - Ref: EcsDefaultClusterMnL3mNNYNVpcPublicSubnet2Subnet95FF715A
      Type: application
    DependsOn:
      - EcsDefaultClusterMnL3mNNYNVpcPublicSubnet1DefaultRouteFF4E2178
      - EcsDefaultClusterMnL3mNNYNVpcPublicSubnet2DefaultRouteB1375520
    Metadata:
      aws:cdk:path: HelloEcsStack/MyWebServer/LB/Resource
  MyWebServerLBSecurityGroup01B285AA:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Automatically created Security Group for ELB
 HelloEcsStackMyWebServerLB06757F57
      SecurityGroupIngress:
        - CidrIp: 0.0.0.0/0
          Description: Allow from anyone on port 80
          FromPort: 80
          IpProtocol: tcp
          ToPort: 80
      VpcId:
        Ref: EcsDefaultClusterMnL3mNNYNVpc7788A521
    Metadata:
      aws:cdk:path: HelloEcsStack/MyWebServer/LB/SecurityGroup/Resource
# and so on for another few hundred lines
```

AWS 계정의 서비스를 배포하려면 `cdk deploy` 애플리케이션의 기본 디렉터리에 있는 명령을 실행합니다. AWS CDK가 생성한 IAM 정책을 승인하라는 메시지가 표시됩니다.

배포에는 AWS CDK가 여러 리소스를 생성하면서 몇 분이 소요됩니다. 배포에서 출력되는 마지막 몇 줄에는 로드 밸런서의 퍼블릭 호스트 이름과 새 웹 서버의 URL이 포함됩니다. 내용은 다음과 같습니다.

```
Outputs:
HelloEcsStack.MyWebServerLoadBalancerDNSXXXXXXX = Hello-MyWeb-ZZZZZZZZZZZZZ-
ZZZZZZZZZZ.us-west-2.elb.amazonaws.com
HelloEcsStack.MyWebServerServiceURLYYYYYYYY = http://Hello-MyWeb-ZZZZZZZZZZZZZ-
ZZZZZZZZZZ.us-west-2.elb.amazonaws.com
```

3단계: 웹 서비스 테스트

배포 출력에서 URL을 복사해 웹 브라우저에 붙여 넣습니다. 웹 서버의 다음 환영 메시지가 표시됩니다.

4단계: 정리

웹 서버 사용을 마친 후 애플리케이션의 기본 디렉터리에 있는 `cdk destroy` 명령을 실행하여 CDK를 사용하여 서비스를 종료합니다. 이렇게 하면 향후 의도하지 않은 요금이 발생하는 것을 방지할 수 있습니다.

다음 단계

AWS CDK를 사용하여 AWS 인프라를 개발하는 방법에 대해 자세히 알아보려면 AWS CDK 개발자 안내서를 참조하세요.

선택한 언어로 AWS CDK 앱을 작성하는 방법에 대한 내용은 다음을 참조하세요.

TypeScript

TypeScript에서 AWS CDK 작업

JavaScript

JavaScript에서 AWS CDK 작업

Python

Python에서 AWS CDK 작업

Java

Java에서 AWS CDK 작업

C#

C#에서 AWS CDK 작업

Go

Go에서 AWS CDK 작업

이 주제에서 사용한 AWS 구성 라이브러리에 대한 자세한 내용은 아래 AWS CDK API 참조 개요를 참조하세요.

- aws-ecs
- aws-ecs-patterns

AWS Copilot 명령줄 인터페이스를 사용하여 Amazon ECS 리소스 생성

AWS Copilot 명령줄 인터페이스(CLI)는 로컬 개발 환경에서 Amazon ECS를 기반으로 프로덕션 지원 컨테이너화 애플리케이션을 간단하게 구축, 릴리스, 운영할 수 있는 명령을 제공합니다. AWS Copilot CLI는 최신 모애플리케이션 모범 사례를 지원하는 개발자 워크플로와 일치합니다. 코드형 인프라에서 사용자 대신 프로비저닝한 CI/CD 파이프라인을 생성하는 것까지 모두 가능합니다. 일상적인 개발 및 테스트 사이클에서 AWS Management Console 대신 AWS Copilot CLI를 사용합니다.

현재 AWS Copilot은 Linux, macOS 및 Windows 시스템을 지원합니다. 최신 버전의 AWS Copilot CLI에 대한 자세한 내용은 릴리스를 참조하세요.

> **Note**
>
> AWS Copilot CLI의 소스 코드는 GitHub에서 얻을 수 있습니다. 포함하고 싶은 변경 사항에 대해서는 문제 및 풀 요청을 제출할 것을 권장합니다. 하지만 Amazon Web Services는 현재 AWS Copilot 코드의 변경된 사본을 실행하도록 지원하지 않습니다. Gitter 또는 GitHub에서 문의를 통해 AWS Copilot에 대한 문제를 보고하세요. 여기에서 문제를 신고하고, 피드백을 제공하고, 버그를 보고할 수 있습니다.
> Copilot 개발에 대한 자세한 내용은 What's the future of Copilot?을 참조하세요.

주제

- AWS Copilot CLI 설치
- AWS Copilot CLI를 사용하여 샘플 Amazon ECS 애플리케이션 배포

AWS Copilot CLI에 대한 추가 문서는 AWS Copilot 웹 사이트에서 확인할 수 있습니다.

AWS Copilot CLI 설치

Homebrew를 사용하거나 다음 단계에 따라 바이너리를 수동으로 다운로드하여 AWS Copilot CLI를 설치할 수 있습니다.

Homebrew 사용

다음 명령은 Homebrew를 사용하여 macOS 또는 Linux 시스템에 AWS Copilot CLI를 설치하는 데 사용합니다. 설치하기 전에 미리 Homebrew를 설치해야 합니다. 자세한 내용은 Homebrew를 참조하세요.

```
brew install aws/tap/copilot-cli
```

바이너리 다운로드

Homebrew 대신 macOS, Windows 또는 Linux 시스템에 AWS Copilot CLI를 수동으로 설치할 수 있습니다. 바이너리를 다운로드하려면 운영 체제에 따라 다음 명령을 사용하세요. macOS 및 Linux 예제에는 또한 바이너리에 실행 권한을 적용하는 명령이 포함되어 있으며 설치가 되는지 확인하는 도움말 메뉴를 나열합니다.

macOS

macOS의 경우:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-darwin \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

macOS ARM 시스템의 경우:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-darwin-arm64 \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

Linux

Linux x86 (64-bit) 시스템:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-linux \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

Linux ARM 시스템:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-linux-arm64 \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

Windows

PowerShell을 사용하여 다음의 명령을 실행합니다.

```
New-Item -Path 'C:\copilot' -ItemType directory; `
  Invoke-WebRequest -OutFile 'C:\copilot\copilot.exe' https://github.com/aws/
copilot-cli/releases/latest/download/copilot-windows.exe
```

(선택 사항) PGP 서명을 사용하여 수동으로 설치한 AWS Copilot CLI 확인

AWS Copilot CLI 실행 파일은 PGP 서명을 사용하여 암호화 서명됩니다. PGP 서명은 AWS Copilot CLI 실행 파일의 유효성을 확인하는 데 사용할 수 있습니다. 다음 단계에 따라 GnuPG 도구를 사용하여 서명을 확인할 수 있습니다.

1. GnuPG를 다운로드하고 설치합니다. 자세한 내용은 GnuPG 웹 사이트를 참조하세요.

   macOS

   Homebrew를 사용하는 것이 좋습니다. 웹 사이트의 지침에 따라 Homebrew를 설치합니다. 자세한 내용은 Homebrew를 참조하세요. Homebrew 설치 이후 macOS 터미널에서 다음 명령을 실행합니다.

   ```
   brew install gnupg
   ```

   Linux

   원하는 Linux 패키지 관리자를 사용하여 gpg를 설치합니다.

   Windows

   GnuPG 웹 사이트에서 Windows Simple Installer를 다운로드하고 관리자로 설치합니다. GnuPG를 설치한 후 Administrator PowerShell을 닫았다가 다시 엽니다.

   자세한 내용은 GnuPG Download를 참조하세요.
2. GnuPG 경로가 환경 경로에 추가되었는지 확인합니다.

   macOS

   ```
   echo $PATH
   ```

   출력에 GnuPG 경로가 보이지 않는 경우 다음 명령을 실행하여 경로에 추가하세요.

   ```
   PATH=$PATH:<path to GnuPG executable files>
   ```

   Linux

   ```
   echo $PATH
   ```

출력에 GnuPG 경로가 보이지 않는 경우 다음 명령을 실행하여 경로에 추가하세요.

```
export PATH=$PATH:<path to GnuPG executable files>
```

Windows

```
Write-Output $Env:PATH
```

출력에 GnuPG 경로가 보이지 않는 경우 다음 명령을 실행하여 경로에 추가하세요.

```
$Env:PATH += ";<path to GnuPG executable files>"
```

3. 로컬 일반 텍스트 파일을 생성합니다.

macOS

터미널에서 다음을 입력합니다.

```
touch <public_key_filename.txt>
```

TextEdit로 파일을 엽니다.

Linux

gedit와 같은 텍스트 편집기에서 텍스트 파일을 생성합니다. `public_key_filename.txt`로 저장합니다.

Windows

메모장과 같은 텍스트 편집기에서 텍스트 파일을 생성합니다. `public_key_filename.txt`로 저장합니다.

4. Amazon ECS PGP 퍼블릭 키의 다음 내용을 추가하고 파일을 저장합니다.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2

mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRMO2fl74lmavr4Vg
7K/KH8VHlq2uRw32/B94XLEgRbGTMdWFdKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu
BoQAhjBQ+bEm0kNy0LjNgjNlnL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx
N+6pHBJrKIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uycHkeliCrw4kj0+skizBgO
```

```
2K7oVX8Oc3j5+ZilhL/qDLXmUCb2az5cMM1mOoF8EKX5HaNuq1KfwJxqXE6NNIcO
lFTrT7QwD5fMNld3FanLgv/ZnIrsSaqJOL6zRSq8O4LN1OWBVbndExk2Kr+5kFxn
5lBPgfPgRj5hQ+KTHMa9Y8Z7yUc64BJiN6F9Nl7FJuSsfqbdkvRLsQRbcBG9qxX3
rJAEhieJzVMEUNl+EgeCkxj5xuSkNU7zw2c3hQZqEcrADLV+hvFJktOz9Gm6xzbq
lTnWWCz4xrIWtuEBA2qE+MlDheVd78a3gIsEaSTfQq0osYXaQbvlnSWOoc1y/5Zb
zizHTJIhLtUyls9WisP2s0emeHZicVMfW61EgPrJAiupgc7kyZvFt4YwfwARAQAB
tCRBbWF6b24gRUNTIDxlY3Mtc2VjdXJpdHlAYW1hem9uLmNvbT6JAhwEEAECAAYF
AlrjL0YACgkQHivRXs0TaQrg1g/+JppwPqHnlVPmv7lessB8I5UqZeD6p6uVpHd7
Bs3pcPp8BV7BdRbs3sPLt5bV1+rkqOlw+0gZ4Q/ue/YbWtOAt4qY0OcEo0HgcnaX
lsB827QIfZIVtGWMhuh94xzm/SJkvngml6KB3YJNnWP61A9qJ37/VbVVLzvcmazA
McWB4HUMNrhd0JgBCo0gIpqCbpJEvUc02Bjn23eEJsS9kC7OUAHyQkVnx4d9UzXF
4OoISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJJX2CSqt7tWJ8gk1n3H3Y
SReRXJRnv7DsDDBwFgT6r5Q2HW1TBUvaoZy5hF6maD09nHcNnvBjqADzeT8Tr/Qu
bBCLzkNSYqqkpgtwv7seoD2P4n1giRvDAOEfMZpVkUr+C252IaH1HZFEz+TvBVQM
Y8OWWxmIJW+J6evjo3N1eO19UHv71jvoF8zljbI4bsL2c+QTJmOv7nRqzDQgCWyp
Id/v2dUVVTk1j9omuLBBwNJzQCB+72LcIzJhYmaP1HC4LcKQG+/f41exuItenatK
lEJQhYtyVXcBlh6Yn/wzNg2NWOwb3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz
N2HqkTSQh77Z8KPKmyGopsmN/reMuilPdINb249nA0dzoN+nj+tTFOYCIaLaFyjs
Z0r1QAOJAjkEEwECACMFAlq1SasCGwMHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIX
gAAKCRC86dmkLVF4T9iFEACEnkm1dNXsWUx34R3c0vamHrPxvfkyI1FlEUen8D1h
uX9xy6jCEROHWEp0rjGK4QDPgM93sWJ+s1UAKg214QRVzft0y9/DdR+twApA0fzy
uavIthGd6+03jAAo6udYDE+cZC3P7XBbDiYEWk4XAF9I1JjB8hTZUgvXBL046JhG
eM17+crgUyQeetkiOQemLbsbXQ40Bd9V7zf7XJraFd8VrwNUwNb+9KFtgAsc9rk+
YIT/PEf+YOPysgcxI4sTWghtyCulVnuGoskgDv4v73PALU0ieUrvvQVqWMRvhVx1
0X90J7cC1KOyhlEQQ1aFTgmQjmXexVTwIBm8LvysFK6YXM41KjOrlz3+6xBIm/qe
bFyLUnf4WoiuOplAaJhK9pRY+XEnGNxdtN4D26Kd0F+PLkm3Tr3Hy3b1Ok34FlGr
KVHUq1TZD7cvMnnNKEELTUcKX+1mV3an16nmAg/my1JSUt6BNK2rJpY1s/kkSGSE
XQ4zuF2IGCpvBFhYAlt5Un5zwqkwwQR3/n2kwAoDzonJcehDw/C/cGos5D0aIU7I
K2X2aTD3+pA7Mx3IMe2hqmYqRt9X42yF1PIEVRneBRJ3HDezAgJrNh0GQWRQkhIx
gz6/cTR+ekr5TptVszS9few2GpI5bCgBKBisZIssT89aw7mAKWut0Gcm4qM9/yK6
1bkCDQRatUmrARAAxNPvVwreJ2yAiFcUpdRlVhsuOgnxvs1QgsIw3H7+Pacr9Hpe
8uftYZqdC82KeSKhpHq7c8gMTMucIINtH25x9BCc73E33EjCL9Lqov1TL7+QkgHe
T+JIhZwdD8Mx2K+LVVVu/aWkNrfMuNwyDUciSI4D5QHa8T+F8fgN4OTpwYjirzel
5yoICMr9hVcbzDNv/ozKCxjx+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viQKJ646s+
psiqXRYtVvYInEhLVrJ0aV6zHFoigE/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxrAg7rOvyRN9cAXfeSMf77I+XTifigNna8x
t/MOdjXr1fjF4pThEi5u6WsuRdFwjY2azEv3vevodTi4HoJReH6dFRa6y8c+UDgl
2iHiOKIpQqLbHEfQmHcDd2fix+AaJKMnPGNku9qCFEMbgSRJpXz6BfwnY1QuKE+I
R6jA0frUNt2jhiGG/F8RceXzohaaC/Cx7LUCUFWc0n7z32C9/Dtj7I1PMOacdZzz
bjJzRKO/ZDv+UN/c9dwAkllzAyPMwGBkUaY68EBstnIliW34aWm6IiHhxioVPKSp
VJfyiXPO0EXqujtHLAeChfjcns3I12YshT1dv2PafG53fp33ZdzeUgsBo+EAEQEA
AYkCHwQYAQIACQUCWrVJqwIbDAAKCRC86dmkLVF4T+ZdD/9x/8APzgNJF3o3STrF
jvnV1ycyhWYGAeBJiu7wjsNWwzMFOv15tLjB7AqeVxZn+WKDD/mIOQ45OZvnYZuy
X7DR0JszaH9wrYTxZLVruAu+t6UL0y/XQ4L1GZ9QR6+r+7t1Mvbfy7BlHbvX/gYt
```

```
Rwe/uwdibI0CagEzyX+2D3kTOlHO5XThbXaNf8AN8zha91Jt2Q2UR2X5T6JcwtMz
FBvZnl3LSmZyE0EQehS2iUurU4uWOpGppuqVnbi0jbCvCHKgDGrqZ0smKNAQng54
F365W3g8AfY48s8XQwzmcliowYX9bT8PZiEi0J4QmQh0aXkpqZyFefuWeOL2R94S
XKzr+gRh3BAULoqF+qK+IUMxTip9KTPNvYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe
TXiKQ8DBWDhBPVPrruLIaenTtZEOsPc4I85yt5U9RoPTStcOr34s3w5yEaJagt6S
Gc5r9ysjkfH6+6rbi1ujxMgROSqtqr+RyB+V9A5/OgtNZc8llK6u4UoOCde8jUUW
vqWKvjJB/Kz3u4zaeNu2ZyyHaOqOuH+TETcW+jsY9IhbEzqN5yQYGi4pVmDkY5vu
lXbJnbqPKpRXgM9BecV9AMbPgbDq/5LnHJJXg+G8YQOgp4lR/hC1TEFdIp5wM8AK
CWsENyt2o1rjgMXiZOMF8A5oBLkCDQRatUuSARAAr77kj7j2QR2SZeOSlFBvV7oS
mFeSNnz9xZssqrsm6bTwSHM6YLDwc7Sdf2esDdyzONETwqrVCg+FxgL8hmo9hS4c
rR6tmrP0mOmptr+xLLsKcaP7ogIXsyZnrEAEsvW8PnfayoiPCdc3cMCR/lTnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLKvaxl7PNelaHGJQY/xo+m
V0bndxf9IY+4oFJ4blD32WqvyxESo7vW6WBh7oqv3Zbm0yQrr8a6mDBpqLkvWwNI
3kpJR974tg5o5LfDu1BeeyHWPSGm4U/G4JB+JIG1ADy+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmuOmhGyTssoG+3OOcGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLmZJXikAOlvE3/wgMqCXscbycbLjLD/bXIuFWo3rzoezeXjgi/DJx
jKBAyBTYO5nMcth1O9oaFd9d0HbsOUDkIMnsgGBE766Piro6MHo0T0rXl07Tp4pI
rwuSOsc6XzCzdImj0Wc6axS/HeUKRXWdXJwno5awTwXKRJMXGfhCvSvbcbc2Wx+L
IKvmB7EB4K3fmjFFE67yolmiw2qRcUBfygtH3eL5XZU28MiCpue8Y8GKJoBAUyvf
KeM1rO8Jm3iRAc5a/D0AEQEAAYkEPgQYAQIACQUCWrVLkgIbAgIpCRC86dmkLVF4
T8FdIAQZAQIABgUCWrVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bXO1dQR4tKhhQ
P0LRqy6zlBY9ILCLowNdGZdqorogUiUymgn3VhEhVtxTOoHcN7qOuM01PNsRnOeS
EYjf8Xrb1clzkD6xULwmOclTb9bBxnBc/4PFvHAbZW3QzusaZniNgkuxt6BTfloS
Of4inq71kjmGK+TlzQ6mUMQUg228NUQC+a84EPqYyAeY1sgvgB7hJBhYL0QAxhcW
6m20Rd8iEc6HyzJ3yCOCsKip/nRWAbf0OvfHfRBp0+m0ZwnJM8cPRFjOqqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNV7giSRIkk0eDSko+bFy6VbMzKUMkUJK3
D3eHFAMkujmbfJmSMTJOPGn5SB1HyjCZNx6bhIIbQyEUB9gKCmUFaqXKwKpF6rj0
iQXAJxLR/shZ5Rk96VxzOphUl7T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmrOqX9zBCVXh0mdWYLrWvmzQFWzG7AoE55fkf8nAEPsalrCdtaNUBHRXA0OQxG
AHMOdJQQvBsmqMvuAdjkDWpFu5y0My5ddU+hiUzUyQLjL5Hhd5LOUDdewlZgIw1j
xrEAUzDKetnemM8GkHxDgg8koev5frmShJuce7vSjKpCNg3EIJSgqMOPFjJuLWtZ
vjHeDNbJy6uNL65ckJy6WhGjEADS2WAW1D6Tfekkc21SsIXk/LqEpLMR/0g5OUif
wcEN1rS9IJXBwIy8MelN9qr5KcKQLmfdfBNEyyceBhyVl0MDyHOKC+7PofMtkGBq
13QieRHv5GJ8LB3fclqHV8pwTTo3Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279ioVTrwpECse0XkiRyKToTjwOb73CGkBZZpJyqux/rmCV/fp4ALdSW8zbz
FJVORaivhoWwzjpfQKhwcU9lABXi2UvVm14v0AfeI7oiJPSU1zM4fEny4oiIBXlR
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhqUIMii+mWra23EwjChaxpvjjcUH
5ilLc5Zq781aCYRygYQw+hu5nFkOH1R+Z50Ubxjd/aqUfnGIAX7kPMD3Lof4KldD
Q8ppQriUvxVo+4nPV6rpTy/PyqCLWDjkguHpJsEFsMkwajrAz0QNSAU5CJ0G2Zu4
yxvYlumHCEl7nbFrm0vIiA75Sa8KnywTDsyZsu3XcOcf3g+g1xWTpjJqy2bYXlqz
9uDOWtArWHOis6bq8l9RE6xr1RBVXS6uqgQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMBdA64e9Rq7bFR7Tvt8gasCZYlNr3lydh+dFHIEkH53HzQe6l88HEic
+0jVnLkCDQRa55wJARAAyLya2Lx6gyoWoJN1a6740q3o8e9d4KggQOfGMTCflmeq
ivuzgN+3DZHN+9ty2KxXMtn0mhHBerZdbNJyjMNT1gAgrhPNB4HtXBXum2wS57WK
```

```
DNmade914L7FWTPAWBG2Wn448OEHTqsClICXXWy9IICgclAEyIq0Yq5mAdTEgRJS
Z8t4GpwtDL9gNQyFXaWQmDmkAsCygQMvhAlmu9xOIzQG5CxSnZFk7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1civ1OwuUidgfPCSvOUW1JojsdCQA
L+RZJcoXq7lfOFj/eNjeOSstCTDPfTCL+kThE6E5neDtbQHBYkEX1BRiTedsV4+M
ucgiTrdQFWKf89G72xdv8ut9AYYQ2BbEYU+JAYhUH8rYYui2dHKJIgjNvJscuUWb
+QEqJIRleJRhrO+/CHgMs4fZAkWF1VFhKBkcKmEjLn1f7EJJUUW84ZhKXjO/AUPX
1CHsNjziRceuJCJYox1cwsoq6jTE50GiNzcIxTn9xUc0UMKFeggNAFys1K+TDTm3
Bzo8H5ucjCUEmUm9lhkGwqTZgOlRX5eqPX+JBoSaObqhgqCa5IPinKRa6MgoFPHK
6sYKqroYwBGgZm6Js5chpNchvJMs/3WXNOEVg0J3z3vP0DMhxqWm+r+n9zlW8qsA
EQEAAYkEPgQYAQgACQUCWuecCQIbAgIpCRC86dmkLVF4T8FdIAQZAQgABgUCWuec
CQAKCRBQ3szEcQ5hr+ykD/4tOLRHFHXuKUcxgGaubUcVtsFrwBKma1cYjqaPms8u
6Sk0wfGRI32G/GhOrp0Ts/MOkbObq6VLTh8N5Yc/53MEl8zQFw9Y5AmRoW4PZXER
ujs5s7p4oR7xHMihMjCCBn1bvrR+34YPfgzTcgLiOEFHYT8UTxwnGmXOvNkMM7md
xD3CV5q6VAte8WKBo/220II3fcQlc9r/oWX4kXXkb0v9hoGwKbDJ1tzqTPrp/xFt
yohqnvImpnlz+Q9zXmbrWYL9/g8VCmW/NN2gju2G3Lu/TlFUWIT4v/5OPK6TdeNb
VKJO4+S8bTayqSG9CML1S57KSgCo5HUhQWeSNHI+fpe5oX6FALPT9JLDce8OZz1i
cZZ0MELP37mOOQun0AlmHm/hVzf0f311PtbzcqWaE51tJvgUR/nZFo6Ta3O5Ezhs
3VlEJNQ1Ijf/6DH87SxvAoRIARCuZd0qxBcDK0avpFzUtbJd24lRA3WJpkEiMqKv
RDVZkE4b6TW61f0o+LaVfK6E8oLpixegS4fiqC16mFrOdyRk+RJJfIUyz0WTDVmt
g0U1CO1ezokMSqkJ7724pyjr2xf/r9/sC6aOJwB/lKgZkJfC6NqL7TlxVA31dUga
LEOvEJTTE4gl+tYtfsCDvALCtqL0jduSkUo+RXcBItmXhA+tShW0pbS2Rtx/ixua
KohVD/0R4QxiSwQmICNtm9mw9ydIl1yjYXX5a9x4wMJracNY/LBybJPFnZnT4dYR
z4XjqysDwvvYZByaWoIe3QxjX84V6MlI2IdAT/xImu8gbaCI8tmyfpIrLnPKiR9D
VFYfGBXuAX7+HgPPSFtrHQONCALxxzlbNpS+zxt9r0MiLgcLyspWxSdmoYGZ6nQP
RO5Nm/ZVS+u2imPCRzNUZEMa+dlE6kHx0rS0dPiuJ4O7NtPeYDKkoQtNagspsDvh
cK7CSqAiKMq06UBTxqlTSRkm62eOCtcs3p3OeHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mC7X+45lcCfmcVt94TFNL5HwEUVJpmOgmzILCI8yoDTWzloo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQXGmNJ3bCLuc/jq7ysGq69xiKmTlUeXFm+aojcRO5i
zyShIRJZ0GZfuzDYFDbMV9amA/YQGygLw//zP5ju5SW26dNxlf3MdFQE5JJ86rn9
MgZ4gcpazHEVUsbZsgkLizRp9imUiH8ymLqAXnfRGlU/LpNSefnvDFTtEIRcpOHc
bhayG0bk51Bd4mioOXnIsKy4j63nJXA27x5EVVHQ1sYRN8Ny4Fdr2tMAmj2O+X+J
qX2yy/UX5nSPU492e2CdZ1UhoU0SRFY3bxKHKB7SDbVeav+K5g==
=Gi5D
-----END PGP PUBLIC KEY BLOCK-----
```

참조용 Amazon ECS PGP 퍼블릭 키에 대한 세부 정보:

```
Key ID: BCE9D9A42D51784F
Type: RSA
Size: 4096/4096
Expires: Never
User ID: Amazon ECS
Key fingerprint: F34C 3DDA E729 26B0 79BE AEC6 BCE9 D9A4 2D51 784F
```

5. 터미널에서 다음 명령을 사용하여 Amazon ECS PGP 퍼블릭 키로 파일을 가져옵니다.

```
gpg --import <public_key_filename.txt>
```

6. AWS Copilot CLI 서명을 다운로드합니다. 서명은 `.asc` 확장자 파일에 저장된 ASCII 분리 PGP 서명입니다. 서명 파일은 해당 실행 파일과 동일한 이름에 `.asc`가 추가되어 있습니다.

   macOS

   macOS 시스템의 경우 다음 명령을 사용합니다.

```
sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/download/copilot-darwin.asc
```

   Linux

   Linux x86(64비트) 시스템은 다음의 명령을 실행합니다.

```
sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/download/copilot-linux.asc
```

   Linux ARM 시스템은 다음의 명령을 실행합니다.

```
sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/download/copilot-linux-arm64.asc
```

   Windows

   PowerShell을 사용하여 다음의 명령을 실행합니다.

```
Invoke-WebRequest -OutFile 'C:\copilot\copilot.asc' https://github.com/aws/copilot-cli/releases/latest/download/copilot-windows.exe.asc
```

7. 다음 명령으로 서명을 확인합니다.

   - macOS 및 Linux 시스템의 경우:

```
gpg --verify copilot.asc /usr/local/bin/copilot
```

   - Windows 시스템의 경우:

```
gpg --verify 'C:\copilot\copilot.asc' 'C:\copilot\copilot.exe'
```

예상 결과:

```
gpg: Signature made Tue Apr  3 13:29:30 2018 PDT
gpg:                using RSA key DE3CBD61ADAF8B8E
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
     Subkey fingerprint: EB3D F841 E2C9 212A 2BD4  2232 DE3C BD61 ADAF 8B8E
```

> **Important**
>
> 결과에서 경고가 예상되지만 문제가 되지는 않습니다. 개인 PGP 키(보유한 경우)와 Amazon ECS PGP 키 사이에 신뢰 체인이 없기 때문에 발생한 것입니다. 자세한 내용은 Web of trust를 참조하세요.

8. Windows 설치의 경우 Powershell에서 다음 명령을 실행하여 AWS Copilot 디렉터리를 경로에 추가합니다.

```
$Env:PATH += ";<path to Copilot executable files>"
```

AWS Copilot CLI를 사용하여 샘플 Amazon ECS 애플리케이션 배포

AWSCopilot CLI를 설치한 후 다음 단계에 따라 샘플 앱을 배포하고, 배포를 확인하고, 리소스를 정리할 수 있습니다.

사전 조건

시작하기 전에 다음 사전 조건을 충족하는지 확인합니다.

- AWS CLI를 설치하고 구성합니다. 자세한 내용은 AWS 명령줄 인터페이스를 참조하세요.
- `aws configure`를 실행하여 애플리케이션 및 서비스를 관리하기 위해 AWS Copilot CLI에서 사용할 기본 프로필을 설정합니다.
- Docker를 설치 및 실행합니다. 자세한 내용은 Docker 시작하기를 참조하세요.

단일 명령을 사용하여 샘플 Amazon ECS 애플리케이션 배포

1. 다음 명령을 사용하여 GitHub 리포지토리에서 복제된 샘플 웹 애플리케이션을 배포하세요. AWS Copilot `init` 및 해당 플래그에 대한 자세한 내용은 AWS Copilot 설명서를 참조하세요.

```
git clone https://github.com/aws-samples/aws-copilot-sample-service.git demo-app &&
 \
cd demo-app &&                                \
copilot init --app demo                       \
  --name api                                  \
  --type 'Load Balanced Web Service'          \
  --dockerfile './Dockerfile'                 \
  --port 80                                   \
  --tag  latest                               \
  --deploy
```

2. 배포가 완료되면 AWS Copilot CLI는 배포를 확인하는 데 사용할 수 있는 URL을 반환합니다. 다음 명령을 사용하여 앱의 상태를 확인할 수도 있습니다.

   - AWS Copilot 애플리케이션을 모두 나열합니다.

     ```
     copilot app ls
     ```

   - 애플리케이션의 환경 및 서비스에 대한 정보를 표시합니다.

     ```
     copilot app show
     ```

   - 환경에 대한 정보를 표시합니다.

     ```
     copilot env ls
     ```

   - 엔드포인트, 용량 및 관련 리소스를 포함하여 서비스에 대한 정보를 표시합니다.

     ```
     copilot svc show
     ```

   - 애플리케이션의 모든 서비스 목록입니다.

     ```
     copilot svc ls
     ```

   - 배포된 서비스의 로그를 표시합니다.

```
copilot svc logs
```

- 서비스 상태를 표시합니다.

```
copilot svc status
```

3. 이 데모를 완료하면 다음 명령을 실행하여 관련 리소스를 정리하고 사용하지 않은 리소스에 대한 요금이 부과되지 않도록 하세요.

```
copilot app delete
```

Amazon ECS에서 AWS CloudFormation 사용

Amazon ECS는 사용자가 정의한 템플릿을 사용하여 AWS 리소스를 모델링 및 설정하는 데 사용할 수 있는 서비스인 AWS CloudFormation과 통합됩니다. AWS CloudFormation은 YAML 또는 JSON 형식의 텍스트 파일인 템플릿을 사용합니다. 템플릿은 생성하려는 AWS 리소스의 블루프린트와 같습니다. 템플릿을 생성하고 제출하면 AWS CloudFormation이 스택을 생성합니다. 템플릿에서 정의한 리소스를 스택을 통해 관리합니다. 리소스를 생성, 업데이트 또는 삭제하려는 경우 해당 리소스에서 생성된 스택을 생성, 업데이트 또는 삭제합니다. 스택을 업데이트할 때는 먼저 변경 세트를 생성해야 합니다. 변경 세트는 변경을 적용하기 전에 그 영향을 받는 항목을 보여줍니다. 그러면 예를 들어 데이터베이스 이름을 변경하여 실수로 데이터베이스를 삭제하는 경우를 방지할 수 있습니다. 템플릿, 스택 및 변경 세트에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS CloudFormation 작동 방식을 참조하세요.

AWS CloudFormation을 사용하여 리소스 및 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있습니다. Amazon ECS 클러스터, 태스크 정의, 서비스 등 원하는 모든 AWS 리소스를 설명하는 템플릿을 생성할 수 있습니다. 그런 다음 AWS CloudFormation은 해당 리소스를 프로비저닝하고 구성합니다.

또한 AWS CloudFormation을 사용하면 템플릿을 재사용하여 Amazon ECS 리소스를 일관되고 반복적으로 설정할 수 있습니다. 리소스를 한 번 설명한 후 여러 AWS 계정 및 AWS 리전 간에 동일한 리소스를 다시 프로비저닝할 수 있습니다.

AWS CloudFormation 템플릿을 AWS Management Console 또는 AWS Command Line Interface와 함께 사용하여 리소스를 생성할 수 있습니다.

AWS CloudFormation에 대한 자세한 내용은 다음 리소스를 참조하세요.

- AWS CloudFormation
- AWS CloudFormation 사용 설명서
- AWS CloudFormation 명령줄 인터페이스 사용 설명서

주제

- Amazon ECS용 AWS CloudFormation 템플릿 예제
- AWS CloudFormation 콘솔을 사용하여 Amazon ECS 리소스 생성
- AWS CLI용 AWS CloudFormation 명령을 사용하여 Amazon ECS 리소스 생성

Amazon ECS용 AWS CloudFormation 템플릿 예제

Amazon ECS 및 관련 서비스에 대한 리소스를 프로비저닝하고 구성하려면 AWS CloudFormation 템플릿에 익숙해야 합니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS CloudFormation 템플릿을 참조하세요. AWS CloudFormation 템플릿은 AWS CloudFormation 스택에서 프로비저닝하려는 리소스를 설명하는 JSON 또는 YAML 형식의 텍스트 파일입니다. JSON 또는 YAML 형식에 익숙하지 않은 경우 AWS 인프라 컴포저를 사용하여 AWS CloudFormation 템플릿을 시작할 수 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 Infrastructure Composer를 사용하여 시각적으로 템플릿 생성을 참조하세요.

AWS CloudFormation를 사용하여 Amazon ECS 클러스터, 태스크 정의 및 서비스를 생성할 수 있습니다. 다음 주제에는 다양한 구성으로 리소스를 생성하는 방법을 보여주는 템플릿이 포함되어 있습니다. AWS CloudFormation 콘솔 또는 AWS CLI에서 이러한 템플릿을 사용하여 이러한 리소스를 생성할 수 있습니다. AWS CloudFormation 콘솔을 사용하여 리소스를 생성하는 방법에 대한 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하세요.

다음 주제에서는 Amazon ECS 태스크 정의, 클러스터 및 서비스용 예제 템플릿을 제공합니다.

주제

- 태스크 정의
- 클러스터
- 서비스
- Amazon ECS에 대한 IAM 역할

태스크 정의

태스크 정의는 파라미터를 설명하는 애플리케이션의 블루프린트와 애플리케이션을 구성하는 하나 이상의 컨테이너입니다. 다음은 Amazon ECS 태스크 정의용 예제 AWS CloudFormation 템플릿입니다. Amazon ECS 태스크 정의에 대한 자세한 내용은 Amazon ECS 작업 정의 섹션을 참조하세요.

Fargate Linux 태스크 정의

다음 템플릿을 사용하여 샘플 Fargate Linux 태스크를 생성할 수 있습니다.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
```

```
  "Description": "ECS Task Definition with parameterized values",
  "Parameters": {
    "ContainerImage": {
      "Type": "String",
      "Default": "public.ecr.aws/docker/library/httpd:2.4",
      "Description": "The container image to use for the task"
    },
    "ContainerCpu": {
      "Type": "Number",
      "Default": 256,
      "Description": "The number of CPU units to reserve for the container",
      "AllowedValues": [256, 512, 1024, 2048, 4096]
    },
    "ContainerMemory": {
      "Type": "Number",
      "Default": 512,
      "Description": "The amount of memory (in MiB) to reserve for the container",
      "AllowedValues": [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192]
    },
    "TaskFamily": {
      "Type": "String",
      "Default": "task-definition-cfn",
      "Description": "The name of the task definition family"
    },
    "ContainerName": {
      "Type": "String",
      "Default": "sample-fargate-app",
      "Description": "The name of the container"
    },
    "ContainerPort": {
      "Type": "Number",
      "Default": 80,
      "Description": "The port number on the container"
    },
    "HostPort": {
      "Type": "Number",
      "Default": 80,
      "Description": "The port number on the host"
    },
    "ExecutionRoleArn": {
      "Type": "String",
      "Default": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole",
      "Description": "The ARN of the task execution role"
    },
```

```
    "LogGroup": {
      "Type": "String",
      "Default": "/ecs/fargate-task-definition",
      "Description": "The CloudWatch log group for container logs"
    },
    "NetworkMode": {
      "Type": "String",
      "Default": "awsvpc",
      "Description": "The Docker networking mode to use",
      "AllowedValues": ["awsvpc", "bridge", "host", "none"]
    },
    "OperatingSystemFamily": {
      "Type": "String",
      "Default": "LINUX",
      "Description": "The operating system for the task",
      "AllowedValues": ["LINUX", "WINDOWS_SERVER_2019_FULL",
 "WINDOWS_SERVER_2019_CORE", "WINDOWS_SERVER_2022_FULL", "WINDOWS_SERVER_2022_CORE"]
    }
  },
  "Resources": {
    "ECSTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ContainerDefinitions": [
          {
            "Command": [
              "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</
title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && -
foreground\""
            ],
            "EntryPoint": [
              "sh",
              "-c"
            ],
            "Essential": true,
            "Image": {"Ref": "ContainerImage"},
            "LogConfiguration": {
              "LogDriver": "awslogs",
              "Options": {
                "mode": "non-blocking",
                "max-buffer-size": "25m",
```

```
                "awslogs-create-group": "true",
                "awslogs-group": {"Ref": "LogGroup"},
                "awslogs-region": {"Ref": "AWS::Region"},
                "awslogs-stream-prefix": "ecs"
              }
            },
            "Name": {"Ref": "ContainerName"},
            "PortMappings": [
              {
                "ContainerPort": {"Ref": "ContainerPort"},
                "HostPort": {"Ref": "HostPort"},
                "Protocol": "tcp"
              }
            ]
          }
        ],
        "Cpu": {"Ref": "ContainerCpu"},
        "ExecutionRoleArn": {"Ref": "ExecutionRoleArn"},
        "Family": {"Ref": "TaskFamily"},
        "Memory": {"Ref": "ContainerMemory"},
        "NetworkMode": {"Ref": "NetworkMode"},
        "RequiresCompatibilities": [
          "FARGATE"
        ],
        "RuntimePlatform": {
          "OperatingSystemFamily": {"Ref": "OperatingSystemFamily"}
        }
      }
    }
  },
  "Outputs": {
    "TaskDefinitionArn": {
      "Description": "The ARN of the created task definition",
      "Value": {"Ref": "ECSTaskDefinition"}
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: 'ECS Task Definition to deploy a sample app'
Parameters:
```

```
  ContainerImage:
    Type: String
    Default: 'public.ecr.aws/docker/library/httpd:2.4'
    Description: The container image to use for the task
  ContainerCpu:
    Type: Number
    Default: 256
    Description: The number of CPU units to reserve for the container
    AllowedValues: [256, 512, 1024, 2048, 4096]
  ContainerMemory:
    Type: Number
    Default: 512
    Description: The amount of memory (in MiB) to reserve for the container
    AllowedValues: [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192]
  TaskFamily:
    Type: String
    Default: 'task-definition-cfn'
    Description: The name of the task definition family
  ContainerName:
    Type: String
    Default: 'sample-fargate-app'
    Description: The name of the container
  ContainerPort:
    Type: Number
    Default: 80
    Description: The port number on the container
  HostPort:
    Type: Number
    Default: 80
    Description: The port number on the host
  ExecutionRoleArn:
    Type: String
    Default: 'arn:aws:iam::111122223333:role/ecsTaskExecutionRole'
    Description: The ARN of the task execution role
  LogGroup:
    Type: String
    Default: '/ecs/fargate-task-definition'
    Description: The CloudWatch log group for container logs
  NetworkMode:
    Type: String
    Default: 'awsvpc'
    Description: The Docker networking mode to use
    AllowedValues: ['awsvpc', 'bridge', 'host', 'none']
  OperatingSystemFamily:
```

```
    Type: String
    Default: 'LINUX'
    Description: The operating system for the task
    AllowedValues: ['LINUX', 'WINDOWS_SERVER_2019_FULL', 'WINDOWS_SERVER_2019_CORE',
 'WINDOWS_SERVER_2022_FULL', 'WINDOWS_SERVER_2022_CORE']
Resources:
  ECSTaskDefinition:
    Type: 'AWS::ECS::TaskDefinition'
    Properties:
      ContainerDefinitions:
        - Command:
            - >-
              /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
              App</title> <style>body {margin-top: 40px; background-color:
              #333;} </style> </head><body> <div
              style=color:white;text-align:center> <h1>Amazon ECS Sample
              App</h1> <h2>Congratulations!</h2> <p>Your application is now
              running on a container in Amazon ECS.</p> </div></body></html>' >
              /usr/local/apache2/htdocs/index.html && httpd-foreground"
          EntryPoint:
            - sh
            - '-c'
          Essential: true
          Image: !Ref ContainerImage
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-create-group: 'true'
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: ecs
          Name: !Ref ContainerName
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              HostPort: !Ref HostPort
              Protocol: tcp
      Cpu: !Ref ContainerCpu
      ExecutionRoleArn: !Ref ExecutionRoleArn
      Family: !Ref TaskFamily
      Memory: !Ref ContainerMemory
      NetworkMode: !Ref NetworkMode
      RequiresCompatibilities:
```

```
        - FARGATE
      RuntimePlatform:
        OperatingSystemFamily: !Ref OperatingSystemFamily
Outputs:
  TaskDefinitionArn:
    Description: The ARN of the created task definition
    Value: !Ref ECSTaskDefinition
```

Amazon ECS 태스크 정의

다음 템플릿을 사용하여 생성한 Amazon EFS 파일 시스템을 사용하는 태스크를 생성할 수 있습니다. Amazon ECS에서 Amazon EBS 볼륨을 사용하는 방법에 대한 자세한 내용은 Amazon ECS에서 Amazon EFS 볼륨 사용 섹션을 참조하세요.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Create a task definition for a web server with parameterized
 values.",
  "Parameters": {
    "ExecutionRoleArn": {
      "Type": "String",
      "Default": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
      "Description": "The ARN of the task execution role"
    },
    "NetworkMode": {
      "Type": "String",
      "Default": "awsvpc",
      "Description": "The Docker networking mode to use",
      "AllowedValues": ["awsvpc", "bridge", "host", "none"]
    },
    "TaskFamily": {
      "Type": "String",
      "Default": "my-ecs-task",
      "Description": "The name of the task definition family"
    },
    "ContainerCpu": {
      "Type": "String",
      "Default": "256",
      "Description": "The number of CPU units to reserve for the container",
      "AllowedValues": ["256", "512", "1024", "2048", "4096"]
```

```
    },
    "ContainerMemory": {
      "Type": "String",
      "Default": "512",
      "Description": "The amount of memory (in MiB) to reserve for the container",
      "AllowedValues": ["512", "1024", "2048", "3072", "4096", "5120", "6144",
"7168", "8192"]
    },
    "ContainerName": {
      "Type": "String",
      "Default": "nginx",
      "Description": "The name of the container"
    },
    "ContainerImage": {
      "Type": "String",
      "Default": "public.ecr.aws/nginx/nginx:latest",
      "Description": "The container image to use for the task"
    },
    "ContainerPort": {
      "Type": "Number",
      "Default": 80,
      "Description": "The port number on the container"
    },
    "InitProcessEnabled": {
      "Type": "String",
      "Default": "true",
      "Description": "Whether to enable the init process inside the container",
      "AllowedValues": ["true", "false"]
    },
    "EfsVolumeName": {
      "Type": "String",
      "Default": "efs-volume",
      "Description": "The name of the EFS volume"
    },
    "EfsContainerPath": {
      "Type": "String",
      "Default": "/usr/share/nginx/html",
      "Description": "The path in the container where the EFS volume will be
mounted"
    },
    "LogGroup": {
      "Type": "String",
      "Default": "LogGroup",
      "Description": "The CloudWatch log group for container logs"
```

```
    },
    "LogStreamPrefix": {
      "Type": "String",
      "Default": "efs-task",
      "Description": "The prefix for the log stream"
    },
    "EfsFilesystemId": {
      "Type": "String",
      "Default": "fs-1234567890abcdef0",
      "Description": "The ID of the EFS filesystem"
    },
    "EfsRootDirectory": {
      "Type": "String",
      "Default": "/",
      "Description": "The root directory in the EFS filesystem"
    },
    "EfsTransitEncryption": {
      "Type": "String",
      "Default": "ENABLED",
      "Description": "Whether to enable transit encryption for EFS",
      "AllowedValues": ["ENABLED", "DISABLED"]
    }
  },
  "Resources": {
    "ECSTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ExecutionRoleArn": {"Ref": "ExecutionRoleArn"},
        "NetworkMode": {"Ref": "NetworkMode"},
        "RequiresCompatibilities": ["FARGATE"],
        "Family": {"Ref": "TaskFamily"},
        "Cpu": {"Ref": "ContainerCpu"},
        "Memory": {"Ref": "ContainerMemory"},
        "ContainerDefinitions": [
          {
            "Name": {"Ref": "ContainerName"},
            "Image": {"Ref": "ContainerImage"},
            "Essential": true,
            "PortMappings": [
              {
                "ContainerPort": {"Ref": "ContainerPort"},
                "Protocol": "tcp"
              }
            ],
```

```
            "LinuxParameters": {
              "InitProcessEnabled": {"Ref": "InitProcessEnabled"}
            },
            "MountPoints": [
              {
                "SourceVolume": {"Ref": "EfsVolumeName"},
                "ContainerPath": {"Ref": "EfsContainerPath"}
              }
            ],
            "LogConfiguration": {
              "LogDriver": "awslogs",
              "Options": {
                "mode": "non-blocking",
                "max-buffer-size": "25m",
                "awslogs-group": {"Ref": "LogGroup"},
                "awslogs-region": {"Ref": "AWS::Region"},
                "awslogs-create-group": "true",
                "awslogs-stream-prefix": {"Ref": "LogStreamPrefix"}
              }
            }
          }
        ],
        "Volumes": [
          {
            "Name": {"Ref": "EfsVolumeName"},
            "EFSVolumeConfiguration": {
              "FilesystemId": {"Ref": "EfsFilesystemId"},
              "RootDirectory": {"Ref": "EfsRootDirectory"},
              "TransitEncryption": {"Ref": "EfsTransitEncryption"}
            }
          }
        ]
      }
    }
  },
  "Outputs": {
    "TaskDefinitionArn": {
      "Description": "The ARN of the created task definition",
      "Value": {"Ref": "ECSTaskDefinition"}
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: Create a task definition for a web server with parameterized values.
Parameters:
  ExecutionRoleArn:
    Type: String
    Default: arn:aws:iam::123456789012:role/ecsTaskExecutionRole
    Description: The ARN of the task execution role
  NetworkMode:
    Type: String
    Default: awsvpc
    Description: The Docker networking mode to use
    AllowedValues: [awsvpc, bridge, host, none]
  TaskFamily:
    Type: String
    Default: my-ecs-task
    Description: The name of the task definition family
  ContainerCpu:
    Type: String
    Default: "256"
    Description: The number of CPU units to reserve for the container
    AllowedValues: ["256", "512", "1024", "2048", "4096"]
  ContainerMemory:
    Type: String
    Default: "512"
    Description: The amount of memory (in MiB) to reserve for the container
    AllowedValues: ["512", "1024", "2048", "3072", "4096", "5120", "6144", "7168",
 "8192"]
  ContainerName:
    Type: String
    Default: nginx
    Description: The name of the container
  ContainerImage:
    Type: String
    Default: public.ecr.aws/nginx/nginx:latest
    Description: The container image to use for the task
  ContainerPort:
    Type: Number
    Default: 80
    Description: The port number on the container
  InitProcessEnabled:
    Type: String
    Default: "true"
```

```
    Description: Whether to enable the init process inside the container
    AllowedValues: ["true", "false"]
  EfsVolumeName:
    Type: String
    Default: efs-volume
    Description: The name of the EFS volume
  EfsContainerPath:
    Type: String
    Default: /usr/share/nginx/html
    Description: The path in the container where the EFS volume will be mounted
  LogGroup:
    Type: String
    Default: LogGroup
    Description: The CloudWatch log group for container logs
  LogStreamPrefix:
    Type: String
    Default: efs-task
    Description: The prefix for the log stream
  EfsFilesystemId:
    Type: String
    Default: fs-1234567890abcdef0
    Description: The ID of the EFS filesystem
  EfsRootDirectory:
    Type: String
    Default: /
    Description: The root directory in the EFS filesystem
  EfsTransitEncryption:
    Type: String
    Default: ENABLED
    Description: Whether to enable transit encryption for EFS
    AllowedValues: [ENABLED, DISABLED]
Resources:
  ECSTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ExecutionRoleArn: !Ref ExecutionRoleArn
      NetworkMode: !Ref NetworkMode
      RequiresCompatibilities:
        - FARGATE
      Family: !Ref TaskFamily
      Cpu: !Ref ContainerCpu
      Memory: !Ref ContainerMemory
      ContainerDefinitions:
        - Name: !Ref ContainerName
```

```
          Image: !Ref ContainerImage
          Essential: true
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              Protocol: tcp
          LinuxParameters:
            InitProcessEnabled: !Ref InitProcessEnabled
          MountPoints:
            - SourceVolume: !Ref EfsVolumeName
              ContainerPath: !Ref EfsContainerPath
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-create-group: "true"
              awslogs-stream-prefix: !Ref LogStreamPrefix
      Volumes:
        - Name: !Ref EfsVolumeName
          EFSVolumeConfiguration:
            FilesystemId: !Ref EfsFilesystemId
            RootDirectory: !Ref EfsRootDirectory
            TransitEncryption: !Ref EfsTransitEncryption
Outputs:
  TaskDefinitionArn:
    Description: The ARN of the created task definition
    Value: !Ref ECSTaskDefinition
```

클러스터

Amazon ECS 클러스터는 태스크 또는 서비스의 논리적 그룹입니다. 다음 템플릿을 사용하여 다양한 구성의 클러스터를 생성할 수 있습니다. Amazon ECS 클러스터에 대한 자세한 내용은 Amazon ECS 클러스터 섹션을 참조하세요.

기본 설정으로 빈 클러스터 생성

다음 템플릿을 사용하여 기본 설정으로 빈 클러스터를 생성할 수 있습니다.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": "MyEmptyCluster"
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  ECSCluster:
    Type: 'AWS::ECS::Cluster'
    Properties:
      ClusterName: MyEmptyCluster
```

관리형 스토리지 암호화 및 향상된 Container Insights를 사용하여 빈 클러스터 생성

다음 템플릿을 사용하여 클러스터 수준 관리형 스토리지 및 향상된 Container Insights가 활성화된 클러스터를 생성할 수 있습니다. 클러스터 수준 암호화는 Amazon EBS 볼륨과 같은 Amazon ECS 관리형 데이터 볼륨에 적용됩니다. Amazon EBS 암호화에 대한 자세한 내용은 Amazon ECS 태스크에 연결된 Amazon EBS 볼륨에 저장된 데이터 암호화 섹션을 참조하세요. 향상된 관찰성과 함께 Container Insights를 사용하는 방법에 대한 자세한 내용은 관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
   "Resources": {
        "Cluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
```

```
                "ClusterName": "EncryptedEnhancedCluster",
                "ClusterSettings": [
                    {
                        "Name": "containerInsights",
                        "Value": "enhanced"
                    }
                ],
                "Configuration": {
                    "ManagedStorageConfiguration": {
                        "KmsKeyId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
                    }
                }
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Cluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: EncryptedEnhancedCluster
      ClusterSettings:
        - Name: containerInsights
          Value: enhanced
      Configuration:
        ManagedStorageConfiguration:
          KmsKeyId: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

AL2023 Amazon ECS-Optimized-AMI를 사용하여 클러스터 생성

다음 템플릿을 사용하여 Amazon EC2에서 AL2023 인스턴스를 실행하는 용량 공급자를 사용하는 클러스터를 생성할 수 있습니다.

> **Important**
>
> 최신 AMI ID는 Amazon Elastic Container Service 개발자 안내서의 Amazon ECS 최적화 AMI를 참조하세요.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "EC2 ECS cluster that starts out empty, with no EC2 instances
 yet. An ECS capacity provider automatically launches more EC2 instances as required
 on the fly when you request ECS to launch services or standalone tasks.",
    "Parameters": {
        "InstanceType": {
            "Type": "String",
            "Description": "EC2 instance type",
            "Default": "t2.medium",
            "AllowedValues": [
                "t1.micro",
                "t2.2xlarge",
                "t2.large",
                "t2.medium",
                "t2.micro",
                "t2.nano",
                "t2.small",
                "t2.xlarge",
                "t3.2xlarge",
                "t3.large",
                "t3.medium",
                "t3.micro",
                "t3.nano",
                "t3.small",
                "t3.xlarge"
            ]
        },
        "DesiredCapacity": {
            "Type": "Number",
            "Default": "0",
            "Description": "Number of EC2 instances to launch in your ECS cluster."
        },
        "MaxSize": {
            "Type": "Number",
            "Default": "100",
            "Description": "Maximum number of EC2 instances that can be launched in
 your ECS cluster."
        },
        "ECSAMI": {
            "Description": "The Amazon Machine Image ID used for the cluster",
            "Type": "AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>",
```

```
            "Default": "/aws/service/ecs/optimized-ami/amazon-linux-2023/
recommended/image_id"
        },
        "VpcId": {
            "Type": "AWS::EC2::VPC::Id",
            "Description": "VPC ID where the ECS cluster is launched",
            "Default": "vpc-1234567890abcdef0"
        },
        "SubnetIds": {
            "Type": "List<AWS::EC2::Subnet::Id>",
            "Description": "List of subnet IDs where the EC2 instances will be
 launched",
            "Default": "subnet-021345abcdef67890"
        }
    },
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterSettings": [
                    {
                        "Name": "containerInsights",
                        "Value": "enabled"
                    }
                ]
            }
        },
        "ECSAutoScalingGroup": {
            "Type": "AWS::AutoScaling::AutoScalingGroup",
            "DependsOn": [
                "ECSCluster",
                "EC2Role"
            ],
            "Properties": {
                "VPCZoneIdentifier": {
                    "Ref": "SubnetIds"
                },
                "LaunchTemplate": {
                    "LaunchTemplateId": {
                        "Ref": "ContainerInstances"
                    },
                    "Version": {
                        "Fn::GetAtt": [
                            "ContainerInstances",
```

```
                            "LatestVersionNumber"
                        ]
                    }
                },
                "MinSize": 0,
                "MaxSize": {
                    "Ref": "MaxSize"
                },
                "DesiredCapacity": {
                    "Ref": "DesiredCapacity"
                },
                "NewInstancesProtectedFromScaleIn": true
            },
            "UpdatePolicy": {
                "AutoScalingReplacingUpdate": {
                    "WillReplace": "true"
                }
            }
        },
        "ContainerInstances": {
            "Type": "AWS::EC2::LaunchTemplate",
            "Properties": {
                "LaunchTemplateName": "asg-launch-template-2",
                "LaunchTemplateData": {
                    "ImageId": {
                        "Ref": "ECSAMI"
                    },
                    "InstanceType": {
                        "Ref": "InstanceType"
                    },
                    "IamInstanceProfile": {
                        "Name": {
                            "Ref": "EC2InstanceProfile"
                        }
                    },
                    "SecurityGroupIds": [
                        {
                            "Ref": "ContainerHostSecurityGroup"
                        }
                    ],
                    "UserData": {
                        "Fn::Base64": {
                            "Fn::Sub": "#!/bin/bash -xe\n echo ECS_CLUSTER=
${ECSCluster} >> /etc/ecs/ecs.config\n yum install -y aws-cfn-bootstrap\n /opt/aws/
```

```
bin/cfn-init -v --stack ${AWS::StackId} --resource ContainerInstances --configsets
 full_install --region ${AWS::Region} &\n"
                        }
                    },
                    "MetadataOptions": {
                        "HttpEndpoint": "enabled",
                        "HttpTokens": "required"
                    }
                }
            }
        },
        "EC2InstanceProfile": {
            "Type": "AWS::IAM::InstanceProfile",
            "Properties": {
                "Path": "/",
                "Roles": [
                    {
                        "Ref": "EC2Role"
                    }
                ]
            }
        },
        "CapacityProvider": {
            "Type": "AWS::ECS::CapacityProvider",
            "Properties": {
                "AutoScalingGroupProvider": {
                    "AutoScalingGroupArn": {
                        "Ref": "ECSAutoScalingGroup"
                    },
                    "ManagedScaling": {
                        "InstanceWarmupPeriod": 60,
                        "MinimumScalingStepSize": 1,
                        "MaximumScalingStepSize": 100,
                        "Status": "ENABLED",
                        "TargetCapacity": 100
                    },
                    "ManagedTerminationProtection": "ENABLED"
                }
            }
        },
        "CapacityProviderAssociation": {
            "Type": "AWS::ECS::ClusterCapacityProviderAssociations",
            "Properties": {
                "CapacityProviders": [
```

```
                {
                    "Ref": "CapacityProvider"
                }
            ],
            "Cluster": {
                "Ref": "ECSCluster"
            },
            "DefaultCapacityProviderStrategy": [
                {
                    "Base": 0,
                    "CapacityProvider": {
                        "Ref": "CapacityProvider"
                    },
                    "Weight": 1
                }
            ]
        }
    },
    "ContainerHostSecurityGroup": {
        "Type": "AWS::EC2::SecurityGroup",
        "Properties": {
            "GroupDescription": "Access to the EC2 hosts that run containers",
            "VpcId": {
                "Ref": "VpcId"
            }
        }
    },
    "EC2Role": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "ec2.amazonaws.com"
                            ]
                        },
                        "Action": [
                            "sts:AssumeRole"
                        ]
                    }
                ]
```

```
                },
                "Path": "/",
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role",
                    "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
                ]
            }
        },
        "ECSTaskExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "ecs-tasks.amazonaws.com"
                                ]
                            },
                            "Action": [
                                "sts:AssumeRole"
                            ],
                            "Condition": {
                                "ArnLike": {
                                    "aws:SourceArn": {
                                        "Fn::Sub": "arn:${AWS::Partition}:ecs:
${AWS::Region}:${AWS::AccountId}:*"
                                    }
                                },
                                "StringEquals": {
                                    "aws:SourceAccount": {
                                        "Fn::Sub": "${AWS::AccountId}"
                                    }
                                }
                            }
                        }
                    ]
                },
                "Path": "/",
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy"
```

```
                ]
            }
        }
    },
    "Outputs": {
        "ClusterName": {
            "Description": "The ECS cluster into which to launch resources",
            "Value": "ECSCluster"
        },
        "ECSTaskExecutionRole": {
            "Description": "The role used to start up a task",
            "Value": "ECSTaskExecutionRole"
        },
        "CapacityProvider": {
            "Description": "The cluster capacity provider that the service should
 use to request capacity when it wants to start up a task",
            "Value": "CapacityProvider"
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: EC2 ECS cluster that starts out empty, with no EC2 instances yet. An
 ECS capacity provider automatically launches more EC2 instances as required on the
 fly when you request ECS to launch services or standalone tasks.
Parameters:
  InstanceType:
    Type: String
    Description: EC2 instance type
    Default: t2.medium
    AllowedValues:
      - t1.micro
      - t2.2xlarge
      - t2.large
      - t2.medium
      - t2.micro
      - t2.nano
      - t2.small
      - t2.xlarge
      - t3.2xlarge
      - t3.large
```

```
      - t3.medium
      - t3.micro
      - t3.nano
      - t3.small
      - t3.xlarge
  DesiredCapacity:
    Type: Number
    Default: '0'
    Description: Number of EC2 instances to launch in your ECS cluster.
  MaxSize:
    Type: Number
    Default: '100'
    Description: Maximum number of EC2 instances that can be launched in your ECS
 cluster.
  ECSAMI:
    Description: The Amazon Machine Image ID used for the cluster
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ecs/optimized-ami/amazon-linux-2023/recommended/image_id
  VpcId:
    Type: AWS::EC2::VPC::Id
    Description: VPC ID where the ECS cluster is launched
    Default: vpc-1234567890abcdef0
  SubnetIds:
    Type: List<AWS::EC2::Subnet::Id>
    Description: List of subnet IDs where the EC2 instances will be launched
    Default: subnet-021345abcdef67890
Resources:
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterSettings:
        - Name: containerInsights
          Value: enabled
  ECSAutoScalingGroup:
    Type: AWS::AutoScaling::AutoScalingGroup
    DependsOn:
      - ECSCluster
      - EC2Role
    Properties:
      VPCZoneIdentifier: !Ref SubnetIds
      LaunchTemplate:
        LaunchTemplateId: !Ref ContainerInstances
        Version: !GetAtt ContainerInstances.LatestVersionNumber
      MinSize: 0
```

```
      MaxSize: !Ref MaxSize
      DesiredCapacity: !Ref DesiredCapacity
      NewInstancesProtectedFromScaleIn: true
    UpdatePolicy:
      AutoScalingReplacingUpdate:
        WillReplace: 'true'
  ContainerInstances:
    Type: AWS::EC2::LaunchTemplate
    Properties:
      LaunchTemplateName: asg-launch-template-2
      LaunchTemplateData:
        ImageId: !Ref ECSAMI
        InstanceType: !Ref InstanceType
        IamInstanceProfile:
          Name: !Ref EC2InstanceProfile
        SecurityGroupIds:
          - !Ref ContainerHostSecurityGroup
        UserData: !Base64
          Fn::Sub: |
            #!/bin/bash -xe
            echo ECS_CLUSTER=${ECSCluster} >> /etc/ecs/ecs.config
            yum install -y aws-cfn-bootstrap
            /opt/aws/bin/cfn-init -v --stack ${AWS::StackId} --resource
ContainerInstances --configsets full_install --region ${AWS::Region} &
        MetadataOptions:
          HttpEndpoint: enabled
          HttpTokens: required
  EC2InstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Path: /
      Roles:
        - !Ref EC2Role
  CapacityProvider:
    Type: AWS::ECS::CapacityProvider
    Properties:
      AutoScalingGroupProvider:
        AutoScalingGroupArn: !Ref ECSAutoScalingGroup
        ManagedScaling:
          InstanceWarmupPeriod: 60
          MinimumScalingStepSize: 1
          MaximumScalingStepSize: 100
          Status: ENABLED
          TargetCapacity: 100
```

```
      ManagedTerminationProtection: ENABLED
  CapacityProviderAssociation:
    Type: AWS::ECS::ClusterCapacityProviderAssociations
    Properties:
      CapacityProviders:
        - !Ref CapacityProvider
      Cluster: !Ref ECSCluster
      DefaultCapacityProviderStrategy:
        - Base: 0
          CapacityProvider: !Ref CapacityProvider
          Weight: 1
  ContainerHostSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the EC2 hosts that run containers
      VpcId: !Ref VpcId
  EC2Role:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - ec2.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role
        - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - ecs-tasks.amazonaws.com
            Action:
              - sts:AssumeRole
            Condition:
              ArnLike:
```

```
                aws:SourceArn: !Sub arn:${AWS::Partition}:ecs:${AWS::Region}:
${AWS::AccountId}:*
              StringEquals:
                aws:SourceAccount: !Sub ${AWS::AccountId}
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
Outputs:
  ClusterName:
    Description: The ECS cluster into which to launch resources
    Value: ECSCluster
  ECSTaskExecutionRole:
    Description: The role used to start up a task
    Value: ECSTaskExecutionRole
  CapacityProvider:
    Description: The cluster capacity provider that the service should use to
 request capacity when it wants to start up a task
    Value: CapacityProvider
```

서비스

Amazon ECS 서비스를 사용하면 Amazon ECS 클러스터에서 지정된 수의 태스크 정의 인스턴스를 동시에 실행하고 유지 관리할 수 있습니다. 태스크가 실패하거나 중지되면 Amazon ECS 서비스 스케줄러가 태스크 정의의 다른 인스턴스를 시작하여 해당 태스크를 대체합니다. 이렇게 하면 서비스에서 원하는 수의 태스크를 유지 관리하는 데 도움이 됩니다. 다음 템플릿을 사용하여 서비스를 배포할 수 있습니다. Amazon ECS 서비스에 대한 자세한 내용은 Amazon ECS 서비스 섹션을 참조하세요.

애플리케이션 배포

다음 템플릿은 태스크 정의에서 정의된 태스크를 시작 및 유지 관리하는 `cfn-service`로 이름이 지정된 Amazon ECS 서비스를 생성합니다. 템플릿은 `CFNCluster`로 이름이 지정된 클러스터를 생성합니다. 클러스터에는 웹 서버를 설정하는 Linux Fargate 작업 정의가 포함되어 있습니다. 또한 템플릿은 필요한 네트워킹 및 로깅 리소스를 생성합니다. Amazon ECS 태스크 실행 역할도 생성합니다. 태스크 실행 역할에 대한 자세한 정보는 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "A template that deploys an application that is built on an
 Apache web server Docker image by creating an Amazon ECS cluster, task definition,
```

```
and service. The template also creates networking and logging resources, and an
Amazon ECS task execution role.",
   "Parameters": {
       "ClusterName": {
           "Type": "String",
           "Default": "CFNCluster",
           "Description": "Name of the ECS Cluster"
       },
       "TaskFamily": {
           "Type": "String",
           "Default": "task-definition-cfn",
           "Description": "Family name for the Task Definition"
       },
       "ServiceName": {
           "Type": "String",
           "Default": "cfn-service",
           "Description": "Name of the ECS Service"
       },
       "ContainerImage": {
           "Type": "String",
           "Default": "public.ecr.aws/docker/library/httpd:2.4",
           "Description": "Container image to use for the task"
       },
       "TaskCpu": {
           "Type": "Number",
           "Default": 256,
           "AllowedValues": [
               256,
               512,
               1024,
               2048,
               4096
           ],
           "Description": "CPU units for the task"
       },
       "TaskMemory": {
           "Type": "Number",
           "Default": 512,
           "AllowedValues": [
               512,
               1024,
               2048,
               4096,
               8192,
```

```
                16384
            ],
            "Description": "Memory (in MiB) for the task"
        },
        "DesiredCount": {
            "Type": "Number",
            "Default": 1,
            "Description": "Desired number of tasks to run"
        },
        "LogGroupName": {
            "Type": "String",
            "Default": "/ecs/fargate-task-definition",
            "Description": "CloudWatch Log Group name"
        },
        "VpcCidr": {
            "Type": "String",
            "Default": "10.0.0.0/16",
            "Description": "CIDR block for the VPC"
        },
        "PublicSubnet1Cidr": {
            "Type": "String",
            "Default": "10.0.0.0/24",
            "Description": "CIDR block for public subnet 1"
        },
        "PublicSubnet2Cidr": {
            "Type": "String",
            "Default": "10.0.1.0/24",
            "Description": "CIDR block for public subnet 2"
        }
    },
    "Resources": {
        "VPC": {
            "Type": "AWS::EC2::VPC",
            "Properties": {
                "CidrBlock": {
                    "Ref": "VpcCidr"
                },
                "EnableDnsSupport": true,
                "EnableDnsHostnames": true,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-VPC"
```

```
                    }
                }
            ]
        }
    },
    "InternetGateway": {
        "Type": "AWS::EC2::InternetGateway",
        "Properties": {
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-IGW"
                    }
                }
            ]
        }
    },
    "InternetGatewayAttachment": {
        "Type": "AWS::EC2::VPCGatewayAttachment",
        "Properties": {
            "InternetGatewayId": {
                "Ref": "InternetGateway"
            },
            "VpcId": {
                "Ref": "VPC"
            }
        }
    },
    "PublicSubnet1": {
        "Type": "AWS::EC2::Subnet",
        "Properties": {
            "VpcId": {
                "Ref": "VPC"
            },
            "AvailabilityZone": {
                "Fn::Select": [
                    0,
                    {
                        "Fn::GetAZs": ""
                    }
                ]
            },
            "CidrBlock": {
```

```
                    "Ref": "PublicSubnet1Cidr"
                },
                "MapPublicIpOnLaunch": true,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-PublicSubnet1"
                        }
                    }
                ]
            }
        },
        "PublicSubnet2": {
            "Type": "AWS::EC2::Subnet",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "AvailabilityZone": {
                    "Fn::Select": [
                        1,
                        {
                            "Fn::GetAZs": ""
                        }
                    ]
                },
                "CidrBlock": {
                    "Ref": "PublicSubnet2Cidr"
                },
                "MapPublicIpOnLaunch": true,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-PublicSubnet2"
                        }
                    }
                ]
            }
        },
        "PublicRouteTable": {
            "Type": "AWS::EC2::RouteTable",
            "Properties": {
```

```
                "VpcId": {
                    "Ref": "VPC"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-PublicRouteTable"
                        }
                    }
                ]
            }
        },
        "DefaultPublicRoute": {
            "Type": "AWS::EC2::Route",
            "DependsOn": "InternetGatewayAttachment",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PublicRouteTable"
                },
                "DestinationCidrBlock": "0.0.0.0/0",
                "GatewayId": {
                    "Ref": "InternetGateway"
                }
            }
        },
        "PublicSubnet1RouteTableAssociation": {
            "Type": "AWS::EC2::SubnetRouteTableAssociation",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PublicRouteTable"
                },
                "SubnetId": {
                    "Ref": "PublicSubnet1"
                }
            }
        },
        "PublicSubnet2RouteTableAssociation": {
            "Type": "AWS::EC2::SubnetRouteTableAssociation",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PublicRouteTable"
                },
                "SubnetId": {
```

```
                "Ref": "PublicSubnet2"
            }
        }
    },
    "ECSSecurityGroup": {
        "Type": "AWS::EC2::SecurityGroup",
        "Properties": {
            "GroupDescription": "Security group for ECS tasks",
            "VpcId": {
                "Ref": "VPC"
            },
            "SecurityGroupIngress": [
                {
                    "IpProtocol": "tcp",
                    "FromPort": 80,
                    "ToPort": 80,
                    "CidrIp": "0.0.0.0/0"
                },
                {
                    "IpProtocol": "tcp",
                    "FromPort": 443,
                    "ToPort": 443,
                    "CidrIp": "0.0.0.0/0"
                }
            ]
        }
    },
    "ECSTaskExecutionRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "ecs-tasks.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "ManagedPolicyArns": [
```

```
                    "arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy"
                ]
            }
        },
        "TaskLogGroup": {
            "Type": "AWS::Logs::LogGroup",
            "DeletionPolicy": "Retain",
            "UpdateReplacePolicy": "Retain",
            "Properties": {
                "LogGroupName": {
                    "Ref": "LogGroupName"
                },
                "RetentionInDays": 30
            }
        },
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": {
                    "Ref": "ClusterName"
                }
            }
        },
        "ECSTaskDefinition": {
            "Type": "AWS::ECS::TaskDefinition",
            "Properties": {
                "ContainerDefinitions": [
                    {
                        "Command": [
                        "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
                        ],
                        "EntryPoint": [
                            "sh",
                            "-c"
                        ],
                        "Essential": true,
                        "Image": {
                            "Ref": "ContainerImage"
```

```
                },
                "LogConfiguration": {
                    "LogDriver": "awslogs",
                    "Options": {
                        "mode": "non-blocking",
                        "max-buffer-size": "25m",
                        "awslogs-create-group": "true",
                        "awslogs-group": {
                            "Ref": "LogGroupName"
                        },
                        "awslogs-region": {
                            "Ref": "AWS::Region"
                        },
                        "awslogs-stream-prefix": "ecs"
                    }
                },
                "Name": "sample-fargate-app",
                "PortMappings": [
                    {
                        "ContainerPort": 80,
                        "HostPort": 80,
                        "Protocol": "tcp"
                    }
                ]
            }
        ],
        "Cpu": {
            "Ref": "TaskCpu"
        },
        "ExecutionRoleArn": {
            "Fn::GetAtt": [
                "ECSTaskExecutionRole",
                "Arn"
            ]
        },
        "Family": {
            "Ref": "TaskFamily"
        },
        "Memory": {
            "Ref": "TaskMemory"
        },
        "NetworkMode": "awsvpc",
        "RequiresCompatibilities": [
            "FARGATE"
```

```
                ],
                "RuntimePlatform": {
                    "OperatingSystemFamily": "LINUX"
                }
            }
        },
        "ECSService": {
            "Type": "AWS::ECS::Service",
            "DependsOn": [
                "PublicSubnet1RouteTableAssociation",
                "PublicSubnet2RouteTableAssociation"
            ],
            "Properties": {
                "ServiceName": {
                    "Ref": "ServiceName"
                },
                "Cluster": {
                    "Ref": "ECSCluster"
                },
                "DesiredCount": {
                    "Ref": "DesiredCount"
                },
                "LaunchType": "FARGATE",
                "NetworkConfiguration": {
                    "AwsvpcConfiguration": {
                        "AssignPublicIp": "ENABLED",
                        "SecurityGroups": [
                            {
                                "Ref": "ECSSecurityGroup"
                            }
                        ],
                        "Subnets": [
                            {
                                "Ref": "PublicSubnet1"
                            },
                            {
                                "Ref": "PublicSubnet2"
                            }
                        ]
                    }
                },
                "TaskDefinition": {
                    "Ref": "ECSTaskDefinition"
                }
```

```
            }
        }
    },
    "Outputs": {
        "ClusterName": {
            "Description": "The name of the ECS cluster",
            "Value": {
                "Ref": "ECSCluster"
            }
        },
        "TaskDefinitionArn": {
            "Description": "The ARN of the task definition",
            "Value": {
                "Ref": "ECSTaskDefinition"
            }
        },
        "ServiceName": {
            "Description": "The name of the ECS service",
            "Value": {
                "Ref": "ECSService"
            }
        },
        "VpcId": {
            "Description": "The ID of the VPC",
            "Value": {
                "Ref": "VPC"
            }
        },
        "PublicSubnet1": {
            "Description": "The ID of public subnet 1",
            "Value": {
                "Ref": "PublicSubnet1"
            }
        },
        "PublicSubnet2": {
            "Description": "The ID of public subnet 2",
            "Value": {
                "Ref": "PublicSubnet2"
            }
        },
        "SecurityGroup": {
            "Description": "The ID of the security group",
            "Value": {
                "Ref": "ECSSecurityGroup"
```

```
            }
        },
        "ExecutionRoleArn": {
            "Description": "The ARN of the task execution role",
            "Value": {
                "Fn::GetAtt": [
                    "ECSTaskExecutionRole",
                    "Arn"
                ]
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: A template that deploys an application that is built on an Apache
 web server Docker image by creating an Amazon ECS cluster, task definition, and
 service. The template also creates networking and logging resources, and an Amazon
 ECS task execution role.
Parameters:
  ClusterName:
    Type: String
    Default: CFNCluster
    Description: Name of the ECS Cluster
  TaskFamily:
    Type: String
    Default: task-definition-cfn
    Description: Family name for the Task Definition
  ServiceName:
    Type: String
    Default: cfn-service
    Description: Name of the ECS Service
  ContainerImage:
    Type: String
    Default: public.ecr.aws/docker/library/httpd:2.4
    Description: Container image to use for the task
  TaskCpu:
    Type: Number
    Default: 256
    AllowedValues: [256, 512, 1024, 2048, 4096]
    Description: CPU units for the task
```

```
  TaskMemory:
    Type: Number
    Default: 512
    AllowedValues: [512, 1024, 2048, 4096, 8192, 16384]
    Description: Memory (in MiB) for the task
  DesiredCount:
    Type: Number
    Default: 1
    Description: Desired number of tasks to run
  LogGroupName:
    Type: String
    Default: /ecs/fargate-task-definition
    Description: CloudWatch Log Group name
  VpcCidr:
    Type: String
    Default: 10.0.0.0/16
    Description: CIDR block for the VPC
  PublicSubnet1Cidr:
    Type: String
    Default: 10.0.0.0/24
    Description: CIDR block for public subnet 1
  PublicSubnet2Cidr:
    Type: String
    Default: 10.0.1.0/24
    Description: CIDR block for public subnet 2
Resources:
  # VPC and Networking Resources
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCidr
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-VPC
  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-IGW
  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
```

```
  Properties:
    InternetGatewayId: !Ref InternetGateway
    VpcId: !Ref VPC
PublicSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [0, !GetAZs '']
    CidrBlock: !Ref PublicSubnet1Cidr
    MapPublicIpOnLaunch: true
    Tags:
      - Key: Name
        Value: !Sub ${AWS::StackName}-PublicSubnet1
PublicSubnet2:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [1, !GetAZs '']
    CidrBlock: !Ref PublicSubnet2Cidr
    MapPublicIpOnLaunch: true
    Tags:
      - Key: Name
        Value: !Sub ${AWS::StackName}-PublicSubnet2
PublicRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref VPC
    Tags:
      - Key: Name
        Value: !Sub ${AWS::StackName}-PublicRouteTable
DefaultPublicRoute:
  Type: AWS::EC2::Route
  DependsOn: InternetGatewayAttachment
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway
PublicSubnet1RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: !Ref PublicRouteTable
    SubnetId: !Ref PublicSubnet1
PublicSubnet2RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
```

```
  Properties:
    RouteTableId: !Ref PublicRouteTable
    SubnetId: !Ref PublicSubnet2
# Security Group
ECSSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Security group for ECS tasks
    VpcId: !Ref VPC
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: 0.0.0.0/0

# IAM Roles
ECSTaskExecutionRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: ecs-tasks.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
# CloudWatch Logs
TaskLogGroup:
  Type: AWS::Logs::LogGroup
  DeletionPolicy: Retain
  UpdateReplacePolicy: Retain
  Properties:
    LogGroupName: !Ref LogGroupName
    RetentionInDays: 30
# ECS Resources
ECSCluster:
  Type: AWS::ECS::Cluster
  Properties:
```

```
      ClusterName: !Ref ClusterName
  ECSTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ContainerDefinitions:
        - Command:
            - >-
              /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
              App</title> <style>body {margin-top: 40px; background-color:
              #333;} </style> </head><body> <div
              style=color:white;text-align:center> <h1>Amazon ECS Sample
              App</h1> <h2>Congratulations!</h2> <p>Your application is now
              running on a container in Amazon ECS.</p> </div></body></html>' >
              /usr/local/apache2/htdocs/index.html && httpd-foreground"s
          EntryPoint:
            - sh
            - '-c'
          Essential: true
          Image: !Ref ContainerImage
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-create-group: 'true'
              awslogs-group: !Ref LogGroupName
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: ecs
          Name: sample-fargate-app
          PortMappings:
            - ContainerPort: 80
              HostPort: 80
              Protocol: tcp
      Cpu: !Ref TaskCpu
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
      Family: !Ref TaskFamily
      Memory: !Ref TaskMemory
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      RuntimePlatform:
        OperatingSystemFamily: LINUX
  ECSService:
    Type: AWS::ECS::Service
```

```
    DependsOn:
      - PublicSubnet1RouteTableAssociation
      - PublicSubnet2RouteTableAssociation
    Properties:
      ServiceName: !Ref ServiceName
      Cluster: !Ref ECSCluster
      DesiredCount: !Ref DesiredCount
      LaunchType: FARGATE
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups:
            - !Ref ECSSecurityGroup
          Subnets:
            - !Ref PublicSubnet1
            - !Ref PublicSubnet2
      TaskDefinition: !Ref ECSTaskDefinition
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref ECSCluster
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref ECSTaskDefinition
  ServiceName:
    Description: The name of the ECS service
    Value: !Ref ECSService
  VpcId:
    Description: The ID of the VPC
    Value: !Ref VPC
  PublicSubnet1:
    Description: The ID of public subnet 1
    Value: !Ref PublicSubnet1
  PublicSubnet2:
    Description: The ID of public subnet 2
    Value: !Ref PublicSubnet2
  SecurityGroup:
    Description: The ID of the security group
    Value: !Ref ECSSecurityGroup
  ExecutionRoleArn:
    Description: The ARN of the task execution role
    Value: !GetAtt ECSTaskExecutionRole.Arn
```

ECS Exec이 활성화된 서비스 배포

다음 템플릿을 사용하여 ECS Exec이 활성화된 서비스를 배포할 수 있습니다. 이 서비스는 ECS Exec 세션을 암호화하기 위한 KMS 키와 실행 명령 세션 로그를 Amazon S3 버킷으로 리디렉션하기 위한 로그 구성을 사용하여 클러스터에서 실행됩니다. 자세한 내용은 ECS Exec를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "ECS Cluster with Fargate Service and Task Definition and ECS
 Exec enabled.",
    "Parameters": {
        "ClusterName": {
            "Type": "String",
            "Default": "CFNCluster",
            "Description": "Name of the ECS Cluster"
        },
        "S3BucketName": {
            "Type": "String",
            "Default": "amzn-s3-demo-bucket",
            "Description": "S3 bucket for ECS execute command logs"
        },
        "KmsKeyId": {
            "Type": "String",
            "Default": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "Description": "KMS Key ID for ECS execute command encryption"
        },
        "ContainerImage": {
            "Type": "String",
            "Default": "public.ecr.aws/docker/library/httpd:2.4",
            "Description": "Container image to use for the task"
        },
        "ContainerCpu": {
            "Type": "Number",
            "Default": 256,
            "AllowedValues": [256, 512, 1024, 2048, 4096],
            "Description": "CPU units for the container (256 = 0.25 vCPU)"
        },
        "ContainerMemory": {
            "Type": "Number",
            "Default": 512,
```

```
            "AllowedValues": [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192],
            "Description": "Memory for the container (in MiB)"
        },
        "DesiredCount": {
            "Type": "Number",
            "Default": 1,
            "Description": "Desired count of tasks in the service"
        },
       "SecurityGroups": {
 "Type": "List<AWS::EC2::SecurityGroup::Id>",
 "Description": "Security Group IDs for the ECS Service"
},
"Subnets": {
 "Type": "List<AWS::EC2::Subnet::Id>",
 "Description": "Subnet IDs for the ECS Service"
},
        "ServiceName": {
            "Type": "String",
            "Default": "cfn-service",
            "Description": "Name of the ECS service"
        },
        "TaskFamily": {
            "Type": "String",
            "Default": "task-definition-cfn",
            "Description": "Family name for the task definition"
        },
        "TaskExecutionRoleArn": {
            "Type": "String",
            "Description": "ARN of an existing IAM role for ECS task execution",
            "Default": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole"
        },
        "TaskRoleArn": {
            "Type": "String",
            "Description": "ARN of an existing IAM role for ECS tasks",
            "Default": "arn:aws:iam::111122223333:role/execTaskRole"
        }
    },
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": {"Ref": "ClusterName"},
                "Configuration": {
                    "ExecuteCommandConfiguration": {
```

```
                        "Logging": "OVERRIDE",
                        "LogConfiguration": {
                            "S3BucketName": {"Ref": "S3BucketName"}
                        },
                        "KmsKeyId": {"Ref": "KmsKeyId"}
                    }
                },
                "Tags": [
                    {
                        "Key": "Environment",
                        "Value": {"Ref": "AWS::StackName"}
                    }
                ]
            }
        },
        "ECSTaskDefinition": {
            "Type": "AWS::ECS::TaskDefinition",
            "Properties": {
                "ContainerDefinitions": [
                    {
                        "Command": [
                            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS
 Sample App</title> <style>body {margin-top: 40px; background-color: #333;} </style>
 </head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
                        ],
                        "EntryPoint": [
                            "sh",
                            "-c"
                        ],
                        "Essential": true,
                        "Image": {"Ref": "ContainerImage"},
                        "LogConfiguration": {
                            "LogDriver": "awslogs",
                            "Options": {
                                "mode": "non-blocking",
                                "max-buffer-size": "25m",
                                "awslogs-create-group": "true",
                                "awslogs-group": {"Fn::Sub": "/ecs/
${AWS::StackName}"},
                                "awslogs-region": {"Ref": "AWS::Region"},
                                "awslogs-stream-prefix": "ecs"
```

```
                        }
                    },
                    "Name": "sample-fargate-app",
                    "PortMappings": [
                        {
                            "ContainerPort": 80,
                            "HostPort": 80,
                            "Protocol": "tcp"
                        }
                    ]
                }
            ],
            "Cpu": {"Ref": "ContainerCpu"},
            "ExecutionRoleArn": {"Ref": "TaskExecutionRoleArn"},
            "TaskRoleArn": {"Ref": "TaskRoleArn"},
            "Family": {"Ref": "TaskFamily"},
            "Memory": {"Ref": "ContainerMemory"},
            "NetworkMode": "awsvpc",
            "RequiresCompatibilities": [
                "FARGATE"
            ],
            "RuntimePlatform": {
                "OperatingSystemFamily": "LINUX"
            },
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {"Fn::Sub": "${AWS::StackName}-TaskDefinition"}
                }
            ]
        }
    },
    "ECSService": {
        "Type": "AWS::ECS::Service",
        "Properties": {
            "ServiceName": {"Ref": "ServiceName"},
            "Cluster": {"Ref": "ECSCluster"},
            "DesiredCount": {"Ref": "DesiredCount"},
            "EnableExecuteCommand": true,
            "LaunchType": "FARGATE",
            "NetworkConfiguration": {
                "AwsvpcConfiguration": {
                    "AssignPublicIp": "ENABLED",
                    "SecurityGroups": {"Ref": "SecurityGroups"},
```

```
                        "Subnets": {"Ref": "Subnets"}
                    }
                },
                "TaskDefinition": {"Ref": "ECSTaskDefinition"},
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {"Fn::Sub": "${AWS::StackName}-Service"}
                    }
                ]
            }
        }
    },
    "Outputs": {
        "ClusterName": {
            "Description": "The name of the ECS cluster",
            "Value": {"Ref": "ECSCluster"}
        },
        "ServiceName": {
            "Description": "The name of the ECS service",
            "Value": {"Ref": "ServiceName"}
        },
        "TaskDefinitionArn": {
            "Description": "The ARN of the task definition",
            "Value": {"Ref": "ECSTaskDefinition"}
        },
        "ClusterArn": {
            "Description": "The ARN of the ECS cluster",
            "Value": {"Fn::GetAtt": ["ECSCluster", "Arn"]}
        },
        "StackName": {
            "Description": "The name of this stack",
            "Value": {"Ref": "AWS::StackName"}
        },
        "StackId": {
            "Description": "The unique identifier for this stack",
            "Value": {"Ref": "AWS::StackId"}
        },
        "Region": {
            "Description": "The AWS Region where the stack is deployed",
            "Value": {"Ref": "AWS::Region"}
        },
        "AccountId": {
            "Description": "The AWS Account ID",
```

```
            "Value": {"Ref": "AWS::AccountId"}
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: ECS Cluster with Fargate Service and Task Definition and ECS Exec
 enabled.
Parameters:
  ClusterName:
    Type: String
    Default: CFNCluster
    Description: Name of the ECS Cluster
  S3BucketName:
    Type: String
    Default: amzn-s3-demo-bucket
    Description: S3 bucket for ECS execute command logs
  KmsKeyId:
    Type: String
    Default: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
    Description: KMS Key ID for ECS execute command encryption
  ContainerImage:
    Type: String
    Default: public.ecr.aws/docker/library/httpd:2.4
    Description: Container image to use for the task
  ContainerCpu:
    Type: Number
    Default: 256
    AllowedValues: [256, 512, 1024, 2048, 4096]
    Description: CPU units for the container (256 = 0.25 vCPU)
  ContainerMemory:
    Type: Number
    Default: 512
    AllowedValues: [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192]
    Description: Memory for the container (in MiB)
  DesiredCount:
    Type: Number
    Default: 1
    Description: Desired count of tasks in the service
 SecurityGroups:
    Type: List<AWS::EC2::SecurityGroup::Id>
```

```
    Description: Security Group IDs for the ECS Service
  Subnets:
    Type: List<AWS::EC2::Subnet::Id>
    Description: Subnet IDs for the ECS Service
  ServiceName:
    Type: String
    Default: cfn-service
    Description: Name of the ECS service
  TaskFamily:
    Type: String
    Default: task-definition-cfn
    Description: Family name for the task definition
  TaskExecutionRoleArn:
    Type: String
    Description: ARN of an existing IAM role for ECS task execution
    Default: 'arn:aws:iam::111122223333:role/ecsTaskExecutionRole'
  TaskRoleArn:
    Type: String
    Description: ARN of an existing IAM role for ECS tasks
    Default: 'arn:aws:iam::111122223333:role/execTaskRole'
Resources:
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Ref ClusterName
      Configuration:
        ExecuteCommandConfiguration:
          Logging: OVERRIDE
          LogConfiguration:
            S3BucketName: !Ref S3BucketName
          KmsKeyId: !Ref KmsKeyId
      Tags:
        - Key: Environment
          Value: !Ref AWS::StackName
  ECSTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ContainerDefinitions:
        - Command:
            - >-
             /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
              App</title> <style>body {margin-top: 40px; background-color:
              #333;} </style> </head><body> <div
              style=color:white;text-align:center> <h1>Amazon ECS Sample
```

```
            App</h1> <h2>Congratulations!</h2> <p>Your application is now
            running on a container in Amazon ECS.</p> </div></body></html>' >
            /usr/local/apache2/htdocs/index.html && httpd-foreground"
        EntryPoint:
          - sh
          - '-c'
        Essential: true
        Image: !Ref ContainerImage
        LogConfiguration:
          LogDriver: awslogs
          Options:
            mode: non-blocking
            max-buffer-size: 25m
            awslogs-create-group: 'true'
            awslogs-group: !Sub /ecs/${AWS::StackName}
            awslogs-region: !Ref AWS::Region
            awslogs-stream-prefix: ecs
        Name: sample-fargate-app
        PortMappings:
          - ContainerPort: 80
            HostPort: 80
            Protocol: tcp
    Cpu: !Ref ContainerCpu
    ExecutionRoleArn: !Ref TaskExecutionRoleArn
    TaskRoleArn: !Ref TaskRoleArn
    Family: !Ref TaskFamily
    Memory: !Ref ContainerMemory
    NetworkMode: awsvpc
    RequiresCompatibilities:
      - FARGATE
    RuntimePlatform:
      OperatingSystemFamily: LINUX
    Tags:
      - Key: Name
        Value: !Sub ${AWS::StackName}-TaskDefinition
ECSService:
  Type: AWS::ECS::Service
  Properties:
    ServiceName: !Ref ServiceName
    Cluster: !Ref ECSCluster
    DesiredCount: !Ref DesiredCount
    EnableExecuteCommand: true
    LaunchType: FARGATE
    NetworkConfiguration:
```

```
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups: !Ref SecurityGroups
          Subnets: !Ref Subnets
      TaskDefinition: !Ref ECSTaskDefinition
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-Service
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref ECSCluster
  ServiceName:
    Description: The name of the ECS service
    Value: !Ref ServiceName
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref ECSTaskDefinition
  ClusterArn:
    Description: The ARN of the ECS cluster
    Value: !GetAtt ECSCluster.Arn
  StackName:
    Description: The name of this stack
    Value: !Ref AWS::StackName
  StackId:
    Description: The unique identifier for this stack
    Value: !Ref AWS::StackId
  Region:
    Description: The AWS Region where the stack is deployed
    Value: !Ref AWS::Region
  AccountId:
    Description: The AWS Account ID
    Value: !Ref AWS::AccountId
```

Amazon VPC Lattice를 사용하는 서비스 배포

다음 템플릿을 사용하여 VPC Lattice를 사용하는 Amazon ECS 서비스의 생성을 시작할 수 있습니다. VPC Lattice를 설정하려면 추가로 다음 단계를 완료해야 할 수 있습니다.

- 인바운드 규칙 `vpc-lattice` 접두사를 허용하고 포트 80에서 트래픽을 허용하도록 보안 그룹의 VPC Lattice에 대한 인바운드 규칙을 업데이트합니다.
- 서비스의 VPC를 VPC Lattice 서비스 네트워크에 연결합니다.

- Amazon Route 53을 사용하여 프라이빗 또는 퍼블릭 호스팅 영역을 구성합니다.
- VPC Lattice 서비스에서 리스너 및 리스너 규칙을 구성합니다.
- 대상 그룹의 상태 확인 구성을 확인합니다.

Amazon VPC에서 VPC Lattice를 사용하는 방법에 대한 자세한 내용은 Amazon VPC Lattice를 사용하여 Amazon ECS 서비스 연결, 관찰 및 보안 유지 섹션을 참조하세요.

JSON

```
{
 "AWSTemplateFormatVersion": "2010-09-09",
 "Description": "The template used to create an ECS Service with VPC Lattice.",
 "Parameters": {
  "ECSClusterName": {
   "Type": "String",
   "Default": "vpc-lattice-cluster"
  },
  "ECSServiceName": {
   "Type": "String",
   "Default": "vpc-lattice-service"
  },
  "SecurityGroupIDs": {
   "Type": "List<AWS::EC2::SecurityGroup::Id>",
   "Description": "Security Group IDs for the ECS Service"
  },
  "SubnetIDs": {
   "Type": "List<AWS::EC2::Subnet::Id>",
   "Description": "Subnet IDs for the ECS Service"
  },
  "VpcID": {
   "Type": "AWS::EC2::VPC::Id",
   "Description": "VPC ID for the resources"
  },
  "ContainerImage": {
   "Type": "String",
   "Default": "public.ecr.aws/docker/library/httpd:2.4",
   "Description": "Container image to use for the task"
  },
  "TaskCpu": {
   "Type": "Number",
   "Default": 256,
   "AllowedValues": [256, 512, 1024, 2048, 4096],
```

```
    "Description": "CPU units for the task"
   },
   "TaskMemory": {
    "Type": "Number",
    "Default": 512,
    "AllowedValues": [512, 1024, 2048, 4096, 8192, 16384],
    "Description": "Memory (in MiB) for the task"
   },
   "LogGroupName": {
    "Type": "String",
    "Default": "/ecs/vpc-lattice-task",
    "Description": "CloudWatch Log Group name"
   },
   "EnableContainerInsights": {
    "Type": "String",
    "Default": "enabled",
    "AllowedValues": ["enabled", "disabled"],
    "Description": "Enable or disable CloudWatch Container Insights for the cluster"
   }
  },
  "Resources": {
   "ECSCluster": {
    "Type": "AWS::ECS::Cluster",
    "Properties": {
     "ClusterName": {"Ref": "ECSClusterName"},
     "ClusterSettings": [
      {
       "Name": "containerInsights",
       "Value": {"Ref": "EnableContainerInsights"}
      }
     ],
     "Tags": [
      {
       "Key": "Name",
       "Value": {"Ref": "ECSClusterName"}
      }
     ]
    }
   },
   "ECSTaskExecutionRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
     "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
```

```
    "Statement": [
     {
      "Effect": "Allow",
      "Principal": {
       "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
     }
    ]
   },
   "ManagedPolicyArns": [
    "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
   ]
  }
 },
 "TaskLogGroup": {
  "Type": "AWS::Logs::LogGroup",
  "DeletionPolicy": "Retain",
  "UpdateReplacePolicy": "Retain",
  "Properties": {
   "LogGroupName": {"Ref": "LogGroupName"},
   "RetentionInDays": 30
  }
 },
 "VpcLatticeTaskDefinition": {
  "Type": "AWS::ECS::TaskDefinition",
  "Properties": {
   "ContainerDefinitions": [
    {
     "Command": [
      "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
<style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
<div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
<h2>Congratulations!</h2> <p>Your application is now running on a container in
Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
httpd-foreground\""
     ],
     "EntryPoint": [
      "sh",
      "-c"
     ],
     "Essential": true,
     "Image": {"Ref": "ContainerImage"},
     "LogConfiguration": {
```

```
        "LogDriver": "awslogs",
        "Options": {
         "mode": "non-blocking",
         "max-buffer-size": "25m",
         "awslogs-create-group": "true",
         "awslogs-group": {"Ref": "LogGroupName"},
         "awslogs-region": {"Ref": "AWS::Region"},
         "awslogs-stream-prefix": "ecs"
        }
       },
       "Name": "vpc-lattice-container",
       "PortMappings": [
        {
         "ContainerPort": 80,
         "HostPort": 80,
         "Protocol": "tcp",
         "Name": "vpc-lattice-port"
        }
       ]
      }
     ],
     "Cpu": {"Ref": "TaskCpu"},
     "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
     "Family": "vpc-lattice-task-definition",
     "Memory": {"Ref": "TaskMemory"},
     "NetworkMode": "awsvpc",
     "RequiresCompatibilities": [
      "FARGATE"
     ],
     "RuntimePlatform": {
      "OperatingSystemFamily": "LINUX"
     }
    }
   },
   "ECSService": {
    "Type": "AWS::ECS::Service",
    "Properties": {
     "Cluster": {"Ref": "ECSCluster"},
     "TaskDefinition": {"Ref": "VpcLatticeTaskDefinition"},
     "LaunchType": "FARGATE",
     "ServiceName": {"Ref": "ECSServiceName"},
     "SchedulingStrategy": "REPLICA",
     "DesiredCount": 2,
     "AvailabilityZoneRebalancing": "ENABLED",
```

```
    "NetworkConfiguration": {
     "AwsvpcConfiguration": {
      "AssignPublicIp": "ENABLED",
      "SecurityGroups": {
       "Ref": "SecurityGroupIDs"
      },
      "Subnets": {
       "Ref": "SubnetIDs"
      }
     }
    },
    "PlatformVersion": "LATEST",
    "VpcLatticeConfigurations": [
     {
      "RoleArn": "arn:aws:iam::111122223333:role/ecsInfrastructureRole",
      "PortName": "vpc-lattice-port",
      "TargetGroupArn": {
       "Ref": "TargetGroup1"
      }
     }
    ],
    "DeploymentConfiguration": {
     "DeploymentCircuitBreaker": {
      "Enable": true,
      "Rollback": true
     },
     "MaximumPercent": 200,
     "MinimumHealthyPercent": 100
    },
    "DeploymentController": {
     "Type": "ECS"
    },
    "ServiceConnectConfiguration": {
     "Enabled": false
    },
    "Tags": [],
    "EnableECSManagedTags": true
   }
  },
  "TargetGroup1": {
   "Type": "AWS::VpcLattice::TargetGroup",
   "Properties": {
    "Type": "IP",
    "Name": "first-target-group",
```

```
    "Config": {
     "Port": 80,
     "Protocol": "HTTP",
     "VpcIdentifier": {"Ref": "VpcID"},
     "HealthCheck": {
      "Enabled": true,
      "Path": "/"
     }
    },
    "Tags": [
     {
      "Key": "ecs-application-networking/ServiceName",
      "Value": {"Ref": "ECSServiceName"}
     },
     {
      "Key": "ecs-application-networking/ClusterName",
      "Value": {"Ref": "ECSClusterName"}
     },
     {
      "Key": "ecs-application-networking/TaskDefinition",
      "Value": {"Ref": "VpcLatticeTaskDefinition"}
     },
     {
      "Key": "ecs-application-networking/VpcId",
      "Value": {"Ref": "VpcID"}
     }
    ]
   }
  }
 },
"Outputs": {
 "ClusterName": {
  "Description": "The cluster used to create the service.",
  "Value": {
   "Ref": "ECSCluster"
  }
 },
 "ClusterArn": {
  "Description": "The ARN of the ECS cluster",
  "Value": {
   "Fn::GetAtt": ["ECSCluster", "Arn"]
  }
 },
 "ECSService": {
```

```
   "Description": "The created service.",
   "Value": {
    "Ref": "ECSService"
   }
  },
  "TaskDefinitionArn": {
   "Description": "The ARN of the task definition",
   "Value": {
    "Ref": "VpcLatticeTaskDefinition"
   }
  }
 }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: The template used to create an ECS Service with VPC Lattice.

Parameters:
  ECSClusterName:
    Type: String
    Default: vpc-lattice-cluster
  ECSServiceName:
    Type: String
    Default: vpc-lattice-service
  SecurityGroupIDs:
    Type: List<AWS::EC2::SecurityGroup::Id>
    Description: Security Group IDs for the ECS Service
  SubnetIDs:
    Type: List<AWS::EC2::Subnet::Id>
    Description: Subnet IDs for the ECS Service
  VpcID:
    Type: AWS::EC2::VPC::Id
    Description: VPC ID for the resources
  ContainerImage:
    Type: String
    Default: public.ecr.aws/docker/library/httpd:2.4
    Description: Container image to use for the task
  TaskCpu:
    Type: Number
    Default: 256
    AllowedValues: [256, 512, 1024, 2048, 4096]
```

```
    Description: CPU units for the task
  TaskMemory:
    Type: Number
    Default: 512
    AllowedValues: [512, 1024, 2048, 4096, 8192, 16384]
    Description: Memory (in MiB) for the task
  LogGroupName:
    Type: String
    Default: /ecs/vpc-lattice-task
    Description: CloudWatch Log Group name
  EnableContainerInsights:
    Type: String
    Default: 'enhanced'
    AllowedValues: ['enabled', 'disabled', 'enhanced']
    Description: Enable or disable CloudWatch Container Insights for the cluster

Resources:
  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Ref ECSClusterName
      ClusterSettings:
        - Name: containerInsights
          Value: !Ref EnableContainerInsights
      Tags:
        - Key: Name
          Value: !Ref ECSClusterName

  # IAM Roles
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy

  # CloudWatch Logs
```

```
  TaskLogGroup:
    Type: AWS::Logs::LogGroup
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      LogGroupName: !Ref LogGroupName
      RetentionInDays: 30

  # Task Definition
  VpcLatticeTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ContainerDefinitions:
        - Command:
            - >-
              /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
              App</title> <style>body {margin-top: 40px; background-color:
              #333;} </style> </head><body> <div
              style=color:white;text-align:center> <h1>Amazon ECS Sample
              App</h1> <h2>Congratulations!</h2> <p>Your application is now
              running on a container in Amazon ECS.</p> </div></body></html>' >
              /usr/local/apache2/htdocs/index.html && httpd-foreground"
          EntryPoint:
            - sh
            - '-c'
          Essential: true
          Image: !Ref ContainerImage
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-create-group: 'true'
              awslogs-group: !Ref LogGroupName
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: ecs
          Name: vpc-lattice-container
          PortMappings:
            - ContainerPort: 80
              HostPort: 80
              Protocol: tcp
              Name: vpc-lattice-port
      Cpu: !Ref TaskCpu
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
```

```
      Family: vpc-lattice-task-definition
      Memory: !Ref TaskMemory
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      RuntimePlatform:
        OperatingSystemFamily: LINUX

  ECSService:
    Type: AWS::ECS::Service
    Properties:
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref VpcLatticeTaskDefinition
      LaunchType: FARGATE
      ServiceName: !Ref ECSServiceName
      SchedulingStrategy: REPLICA
      DesiredCount: 2
      AvailabilityZoneRebalancing: ENABLED
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups: !Ref SecurityGroupIDs
          Subnets: !Ref SubnetIDs
      PlatformVersion: LATEST
      VpcLatticeConfigurations:
        - RoleArn: arn:aws:iam::111122223333:role/ecsInfrastructureRole
          PortName: vpc-lattice-port
          TargetGroupArn: !Ref TargetGroup1
      DeploymentConfiguration:
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
        MaximumPercent: 200
        MinimumHealthyPercent: 100
      DeploymentController:
        Type: ECS
      ServiceConnectConfiguration:
        Enabled: false
      Tags: []
      EnableECSManagedTags: true

  TargetGroup1:
    Type: AWS::VpcLattice::TargetGroup
    Properties:
```

```
      Type: IP
      Name: first-target-group
      Config:
        Port: 80
        Protocol: HTTP
        VpcIdentifier: !Ref VpcID
        HealthCheck:
          Enabled: true
          Path: /
      Tags:
        - Key: ecs-application-networking/ServiceName
          Value: !Ref ECSServiceName
        - Key: ecs-application-networking/ClusterName
          Value: !Ref ECSClusterName
        - Key: ecs-application-networking/TaskDefinition
          Value: !Ref VpcLatticeTaskDefinition
        - Key: ecs-application-networking/VpcId
          Value: !Ref VpcID

Outputs:
  ClusterName:
    Description: The cluster used to create the service.
    Value: !Ref ECSCluster
  ClusterArn:
    Description: The ARN of the ECS cluster
    Value: !GetAtt ECSCluster.Arn
  ECSService:
    Description: The created service.
    Value: !Ref ECSService
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref VpcLatticeTaskDefinition
```

볼륨 구성을 사용하여 서비스 배포

다음 템플릿은 서비스 정의에 볼륨 구성을 포함합니다. Amazon ECS는 시작 시 볼륨 구성을 사용하여 Amazon EBS 볼륨과 같은 데이터 볼륨의 구성을 지원합니다. Amazon EBS 볼륨에 대한 자세한 내용은 Amazon ECS에서 Amazon EBS 볼륨 사용 섹션을 참조하세요.

JSON

```
{
```

```
 "AWSTemplateFormatVersion": "2010-09-09",
 "Description": "The template used to create an ECS Service that includes a
volume configuration. The configuration is used to create Amazon EBS volumes for
attachment to the tasks. One volume is attached per task.",
 "Parameters": {
   "ECSClusterName": {
     "Type": "String",
     "Default": "volume-config-cluster",
     "Description": "Name of the ECS cluster"
   },
   "SecurityGroupIDs": {
     "Type": "List<AWS::EC2::SecurityGroup::Id>",
     "Description": "Security Group IDs for the ECS Service"
   },
   "SubnetIDs": {
     "Type": "List<AWS::EC2::Subnet::Id>",
     "Description": "Subnet IDs for the ECS Service"
   },
   "InfrastructureRoleArn": {
     "Type": "String",
     "Description": "ARN of the IAM role that ECS will use to manage EBS volumes"
   },
   "ContainerImage": {
     "Type": "String",
     "Default": "public.ecr.aws/nginx/nginx:latest",
     "Description": "Container image to use for the task"
   },
   "TaskCpu": {
     "Type": "String",
     "Default": "2048",
     "Description": "CPU units for the task"
   },
   "TaskMemory": {
     "Type": "String",
     "Default": "4096",
     "Description": "Memory (in MiB) for the task"
   },
   "VolumeSize": {
     "Type": "String",
     "Default": "10",
     "Description": "Size of the EBS volume in GiB"
   },
   "VolumeType": {
     "Type": "String",
```

```
      "Default": "gp3",
      "AllowedValues": ["gp2", "gp3", "io1", "io2", "st1", "sc1", "standard"],
      "Description": "EBS volume type"
    },
    "VolumeIops": {
      "Type": "String",
      "Default": "3000",
      "Description": "IOPS for the EBS volume (required for io1, io2, and gp3)"
    },
    "VolumeThroughput": {
      "Type": "String",
      "Default": "125",
      "Description": "Throughput for the EBS volume (only for gp3)"
    },
    "FilesystemType": {
      "Type": "String",
      "Default": "xfs",
      "AllowedValues": ["xfs", "ext4"],
      "Description": "Filesystem type for the EBS volume"
    },
    "EnableContainerInsights": {
      "Type": "String",
      "Default": "enhanced",
      "AllowedValues": ["enabled", "disabled", "enhanced"],
      "Description": "Enable or disable CloudWatch Container Insights for the
cluster"
    }
  },
  "Resources": {
    "ECSCluster": {
      "Type": "AWS::ECS::Cluster",
      "Properties": {
        "ClusterName": {"Ref": "ECSClusterName"},
        "ClusterSettings": [
          {
            "Name": "containerInsights",
            "Value": {"Ref": "EnableContainerInsights"}
          }
        ],
        "Tags": [
          {
            "Key": "Name",
            "Value": {"Ref": "ECSClusterName"}
          }
```

```
        ]
      }
    },
    "ECSTaskExecutionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
        ]
      }
    },
    "EBSTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "Family": "ebs-task-attach-task-def",
        "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
        "NetworkMode": "awsvpc",
        "RequiresCompatibilities": [
          "EC2",
          "FARGATE"
        ],
        "Cpu": {"Ref": "TaskCpu"},
        "Memory": {"Ref": "TaskMemory"},
        "ContainerDefinitions": [
          {
            "Name": "nginx",
            "Image": {"Ref": "ContainerImage"},
            "Essential": true,
            "PortMappings": [
              {
                "Name": "nginx-80-tcp",
                "ContainerPort": 80,
```

```
                "HostPort": 80,
                "Protocol": "tcp",
                "AppProtocol": "http"
              }
            ],
            "MountPoints": [
              {
                "SourceVolume": "ebs-vol",
                "ContainerPath": "/foo-container-path",
                "ReadOnly": false
              }
            ]
          }
        ],
        "Volumes": [
          {
            "Name": "ebs-vol",
            "ConfiguredAtLaunch": true
          }
        ]
      }
    },
    "ECSService": {
      "Type": "AWS::ECS::Service",
      "Properties": {
        "Cluster": {"Ref": "ECSCluster"},
        "TaskDefinition": {"Ref": "EBSTaskDefinition"},
        "LaunchType": "FARGATE",
        "ServiceName": "ebs",
        "SchedulingStrategy": "REPLICA",
        "DesiredCount": 1,
        "NetworkConfiguration": {
          "AwsvpcConfiguration": {
            "AssignPublicIp": "ENABLED",
            "SecurityGroups": {"Ref": "SecurityGroupIDs"},
            "Subnets": {"Ref": "SubnetIDs"}
          }
        },
        "PlatformVersion": "LATEST",
        "DeploymentConfiguration": {
          "MaximumPercent": 200,
          "MinimumHealthyPercent": 100,
          "DeploymentCircuitBreaker": {
            "Enable": true,
```

```
            "Rollback": true
          }
        },
        "DeploymentController": {
          "Type": "ECS"
        },
        "Tags": [],
        "EnableECSManagedTags": true,
        "VolumeConfigurations": [
          {
            "Name": "ebs-vol",
            "ManagedEBSVolume": {
              "RoleArn": {"Ref": "InfrastructureRoleArn"},
              "VolumeType": {"Ref": "VolumeType"},
              "Iops": {"Ref": "VolumeIops"},
              "Throughput": {"Ref": "VolumeThroughput"},
              "SizeInGiB": {"Ref": "VolumeSize"},
              "FilesystemType": {"Ref": "FilesystemType"},
              "TagSpecifications": [
                {
                  "ResourceType": "volume",
                  "PropagateTags": "TASK_DEFINITION"
                }
              ]
            }
          }
        ]
      }
    }
  },
  "Outputs": {
    "ClusterName": {
      "Description": "The cluster used to create the service.",
      "Value": {"Ref": "ECSCluster"}
    },
    "ClusterArn": {
      "Description": "The ARN of the ECS cluster",
      "Value": {"Fn::GetAtt": ["ECSCluster", "Arn"]}
    },
    "ECSService": {
      "Description": "The created service.",
      "Value": {"Ref": "ECSService"}
    },
    "TaskDefinitionArn": {
```

```
      "Description": "The ARN of the task definition",
      "Value": {"Ref": "EBSTaskDefinition"}
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: The template used to create an ECS Service that includes a volume
 configuration. The configuration is used to create Amazon EBS volumes for
 attachment to the tasks. One volume is attached per task.
Parameters:
  ECSClusterName:
    Type: String
    Default: volume-config-cluster
    Description: Name of the ECS cluster

  SecurityGroupIDs:
    Type: List<AWS::EC2::SecurityGroup::Id>
    Description: Security Group IDs for the ECS Service

  SubnetIDs:
    Type: List<AWS::EC2::Subnet::Id>
    Description: Subnet IDs for the ECS Service

  InfrastructureRoleArn:
    Type: String
    Description: ARN of the IAM role that ECS will use to manage EBS volumes

  ContainerImage:
    Type: String
    Default: public.ecr.aws/nginx/nginx:latest
    Description: Container image to use for the task

  TaskCpu:
    Type: String
    Default: "2048"
    Description: CPU units for the task

  TaskMemory:
    Type: String
    Default: "4096"
```

```
    Description: Memory (in MiB) for the task

  VolumeSize:
    Type: String
    Default: "10"
    Description: Size of the EBS volume in GiB

  VolumeType:
    Type: String
    Default: gp3
    AllowedValues: [gp2, gp3, io1, io2, st1, sc1, standard]
    Description: EBS volume type

  VolumeIops:
    Type: String
    Default: "3000"
    Description: IOPS for the EBS volume (required for io1, io2, and gp3)

  VolumeThroughput:
    Type: String
    Default: "125"
    Description: Throughput for the EBS volume (only for gp3)

  FilesystemType:
    Type: String
    Default: xfs
    AllowedValues: [xfs, ext4]
    Description: Filesystem type for the EBS volume

  EnableContainerInsights:
    Type: String
    Default: 'enhanced'
    AllowedValues: ['enabled', 'disabled', 'enhanced']
    Description: Enable or disable CloudWatch Container Insights for the cluster

Resources:
  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Ref ECSClusterName
      ClusterSettings:
        - Name: containerInsights
          Value: !Ref EnableContainerInsights
```

```
    Tags:
      - Key: Name
        Value: !Ref ECSClusterName

# IAM Role for Task Execution
ECSTaskExecutionRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: ecs-tasks.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy

# Task Definition
EBSTaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    Family: ebs-task-attach-task-def
    ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
    NetworkMode: awsvpc
    RequiresCompatibilities:
      - EC2
      - FARGATE
    Cpu: !Ref TaskCpu
    Memory: !Ref TaskMemory
    ContainerDefinitions:
      - Name: nginx
        Image: !Ref ContainerImage
        Essential: true
        PortMappings:
          - Name: nginx-80-tcp
            ContainerPort: 80
            HostPort: 80
            Protocol: tcp
            AppProtocol: http
        MountPoints:
          - SourceVolume: ebs-vol
            ContainerPath: /foo-container-path
            ReadOnly: false
```

```
      Volumes:
        - Name: ebs-vol
          ConfiguredAtLaunch: true

  ECSService:
    Type: AWS::ECS::Service
    Properties:
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref EBSTaskDefinition
      LaunchType: FARGATE
      ServiceName: ebs
      SchedulingStrategy: REPLICA
      DesiredCount: 1
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups: !Ref SecurityGroupIDs
          Subnets: !Ref SubnetIDs
      PlatformVersion: LATEST
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 100
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
      DeploymentController:
        Type: ECS
      Tags: []
      EnableECSManagedTags: true
      VolumeConfigurations:
        - Name: ebs-vol
          ManagedEBSVolume:
            RoleArn: !Ref InfrastructureRoleArn
            VolumeType: !Ref VolumeType
            Iops: !Ref VolumeIops
            Throughput: !Ref VolumeThroughput
            SizeInGiB: !Ref VolumeSize
            FilesystemType: !Ref FilesystemType
            TagSpecifications:
              - ResourceType: volume
                PropagateTags: TASK_DEFINITION

Outputs:
  ClusterName:
```

```
    Description: The cluster used to create the service.
    Value: !Ref ECSCluster
  ClusterArn:
    Description: The ARN of the ECS cluster
    Value: !GetAtt ECSCluster.Arn
  ECSService:
    Description: The created service.
    Value: !Ref ECSService
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref EBSTaskDefinition
```

용량 공급자를 사용하여 서비스 배포

다음 템플릿은 용량 공급자를 통해 실행할 AL2023 용량을 요청하는 서비스를 정의합니다. 컨테이너가 온라인 상태가 되면 AL2023 인스턴스에서 실행됩니다.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "An example service that deploys in AWS VPC networking mode on
 EC2 capacity. Service uses a capacity provider to request EC2 instances to run
 on. Service runs with networking in private subnets, but still accessible to the
 internet via a load balancer hosted in public subnets.",
  "Parameters": {
      "VpcId": {
          "Type": "String",
          "Description": "The VPC that the service is running inside of"
      },
      "PublicSubnetIds": {
          "Type": "List<AWS::EC2::Subnet::Id>",
          "Description": "List of public subnet ID's to put the load balancer in"
      },
      "PrivateSubnetIds": {
          "Type": "List<AWS::EC2::Subnet::Id>",
          "Description": "List of private subnet ID's that the AWS VPC tasks are in"
      },
      "ClusterName": {
          "Type": "String",
          "Description": "The name of the ECS cluster into which to launch
 capacity."
```

```
    },
    "ECSTaskExecutionRole": {
        "Type": "String",
        "Description": "The role used to start up an ECS task"
    },
    "CapacityProvider": {
        "Type": "String",
        "Description": "The cluster capacity provider that the service should use
to request capacity when it wants to start up a task"
    },
    "ServiceName": {
        "Type": "String",
        "Default": "web",
        "Description": "A name for the service"
    },
    "ImageUrl": {
        "Type": "String",
        "Default": "public.ecr.aws/docker/library/nginx:latest",
        "Description": "The url of a docker image that contains the application
process that will handle the traffic for this service"
    },
    "ContainerCpu": {
        "Type": "Number",
        "Default": 256,
        "Description": "How much CPU to give the container. 1024 is 1 CPU"
    },
    "ContainerMemory": {
        "Type": "Number",
        "Default": 512,
        "Description": "How much memory in megabytes to give the container"
    },
    "ContainerPort": {
        "Type": "Number",
        "Default": 80,
        "Description": "What port that the application expects traffic on"
    },
    "DesiredCount": {
        "Type": "Number",
        "Default": 2,
        "Description": "How many copies of the service task to run"
    }
},
"Resources": {
    "TaskDefinition": {
```

```
            "Type": "AWS::ECS::TaskDefinition",
            "Properties": {
                "Family": {
                    "Ref": "ServiceName"
                },
                "Cpu": {
                    "Ref": "ContainerCpu"
                },
                "Memory": {
                    "Ref": "ContainerMemory"
                },
                "NetworkMode": "awsvpc",
                "RequiresCompatibilities": [
                    "EC2"
                ],
                "ExecutionRoleArn": {
                    "Ref": "ECSTaskExecutionRole"
                },
                "ContainerDefinitions": [
                    {
                        "Name": {
                            "Ref": "ServiceName"
                        },
                        "Cpu": {
                            "Ref": "ContainerCpu"
                        },
                        "Memory": {
                            "Ref": "ContainerMemory"
                        },
                        "Image": {
                            "Ref": "ImageUrl"
                        },
                        "PortMappings": [
                            {
                                "ContainerPort": {
                                    "Ref": "ContainerPort"
                                },
                                "HostPort": {
                                    "Ref": "ContainerPort"
                                }
                            }
                        ],
                        "LogConfiguration": {
                            "LogDriver": "awslogs",
```

```
                        "Options": {
                            "mode": "non-blocking",
                            "max-buffer-size": "25m",
                            "awslogs-group": {
                                "Ref": "LogGroup"
                            },
                            "awslogs-region": {
                                "Ref": "AWS::Region"
                            },
                            "awslogs-stream-prefix": {
                                "Ref": "ServiceName"
                            }
                        }
                    }
                }
            ]
        }
    },
    "Service": {
        "Type": "AWS::ECS::Service",
        "DependsOn": "PublicLoadBalancerListener",
        "Properties": {
            "ServiceName": {
                "Ref": "ServiceName"
            },
            "Cluster": {
                "Ref": "ClusterName"
            },
            "PlacementStrategies": [
                {
                    "Field": "attribute:ecs.availability-zone",
                    "Type": "spread"
                },
                {
                    "Field": "cpu",
                    "Type": "binpack"
                }
            ],
            "CapacityProviderStrategy": [
                {
                    "Base": 0,
                    "CapacityProvider": {
                        "Ref": "CapacityProvider"
                    },
```

```
                    "Weight": 1
                }
            ],
            "NetworkConfiguration": {
                "AwsvpcConfiguration": {
                    "SecurityGroups": [
                        {
                            "Ref": "ServiceSecurityGroup"
                        }
                    ],
                    "Subnets": {
                        "Ref": "PrivateSubnetIds"
                    }
                }
            },
            "DeploymentConfiguration": {
                "MaximumPercent": 200,
                "MinimumHealthyPercent": 75
            },
            "DesiredCount": {
                "Ref": "DesiredCount"
            },
            "TaskDefinition": {
                "Ref": "TaskDefinition"
            },
            "LoadBalancers": [
                {
                    "ContainerName": {
                        "Ref": "ServiceName"
                    },
                    "ContainerPort": {
                        "Ref": "ContainerPort"
                    },
                    "TargetGroupArn": {
                        "Ref": "ServiceTargetGroup"
                    }
                }
            ]
        }
    },
    "ServiceSecurityGroup": {
        "Type": "AWS::EC2::SecurityGroup",
        "Properties": {
            "GroupDescription": "Security group for service",
```

```
            "VpcId": {
                "Ref": "VpcId"
            }
        }
    },
    "ServiceTargetGroup": {
        "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
        "Properties": {
            "HealthCheckIntervalSeconds": 6,
            "HealthCheckPath": "/",
            "HealthCheckProtocol": "HTTP",
            "HealthCheckTimeoutSeconds": 5,
            "HealthyThresholdCount": 2,
            "TargetType": "ip",
            "Port": {
                "Ref": "ContainerPort"
            },
            "Protocol": "HTTP",
            "UnhealthyThresholdCount": 10,
            "VpcId": {
                "Ref": "VpcId"
            },
            "TargetGroupAttributes": [
                {
                    "Key": "deregistration_delay.timeout_seconds",
                    "Value": 0
                }
            ]
        }
    },
    "PublicLoadBalancerSG": {
        "Type": "AWS::EC2::SecurityGroup",
        "Properties": {
            "GroupDescription": "Access to the public facing load balancer",
            "VpcId": {
                "Ref": "VpcId"
            },
            "SecurityGroupIngress": [
                {
                    "CidrIp": "0.0.0.0/0",
                    "IpProtocol": -1
                }
            ]
        }
```

```
        },
        "PublicLoadBalancer": {
            "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
            "Properties": {
                "Scheme": "internet-facing",
                "LoadBalancerAttributes": [
                    {
                        "Key": "idle_timeout.timeout_seconds",
                        "Value": "30"
                    }
                ],
                "Subnets": {
                    "Ref": "PublicSubnetIds"
                },
                "SecurityGroups": [
                    {
                        "Ref": "PublicLoadBalancerSG"
                    }
                ]
            }
        },
        "PublicLoadBalancerListener": {
            "Type": "AWS::ElasticLoadBalancingV2::Listener",
            "Properties": {
                "DefaultActions": [
                    {
                        "Type": "forward",
                        "ForwardConfig": {
                            "TargetGroups": [
                                {
                                    "TargetGroupArn": {
                                        "Ref": "ServiceTargetGroup"
                                    },
                                    "Weight": 100
                                }
                            ]
                        }
                    }
                ],
                "LoadBalancerArn": {
                    "Ref": "PublicLoadBalancer"
                },
                "Port": 80,
                "Protocol": "HTTP"
```

```
            }
        },
        "ServiceIngressfromLoadBalancer": {
            "Type": "AWS::EC2::SecurityGroupIngress",
            "Properties": {
                "Description": "Ingress from the public ALB",
                "GroupId": {
                    "Ref": "ServiceSecurityGroup"
                },
                "IpProtocol": -1,
                "SourceSecurityGroupId": {
                    "Ref": "PublicLoadBalancerSG"
                }
            }
        },
        "LogGroup": {
            "Type": "AWS::Logs::LogGroup"
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: >-
  An example service that deploys in AWS VPC networking mode on EC2 capacity.
  Service uses a capacity provider to request EC2 instances to run on. Service
  runs with networking in private subnets, but still accessible to the internet
  via a load balancer hosted in public subnets.
Parameters:
  VpcId:
    Type: String
    Description: The VPC that the service is running inside of
  PublicSubnetIds:
    Type: 'List<AWS::EC2::Subnet::Id>'
    Description: List of public subnet ID's to put the load balancer in
  PrivateSubnetIds:
    Type: 'List<AWS::EC2::Subnet::Id>'
    Description: List of private subnet ID's that the AWS VPC tasks are in
  ClusterName:
    Type: String
    Description: The name of the ECS cluster into which to launch capacity.
  ECSTaskExecutionRole:
```

```
    Type: String
    Description: The role used to start up an ECS task
  CapacityProvider:
    Type: String
    Description: >-
      The cluster capacity provider that the service should use to request
      capacity when it wants to start up a task
  ServiceName:
    Type: String
    Default: web
    Description: A name for the service
  ImageUrl:
    Type: String
    Default: 'public.ecr.aws/docker/library/nginx:latest'
    Description: >-
      The url of a docker image that contains the application process that will
      handle the traffic for this service
  ContainerCpu:
    Type: Number
    Default: 256
    Description: How much CPU to give the container. 1024 is 1 CPU
  ContainerMemory:
    Type: Number
    Default: 512
    Description: How much memory in megabytes to give the container
  ContainerPort:
    Type: Number
    Default: 80
    Description: What port that the application expects traffic on
  DesiredCount:
    Type: Number
    Default: 2
    Description: How many copies of the service task to run
Resources:
  TaskDefinition:
    Type: 'AWS::ECS::TaskDefinition'
    Properties:
      Family: !Ref ServiceName
      Cpu: !Ref ContainerCpu
      Memory: !Ref ContainerMemory
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - EC2
      ExecutionRoleArn: !Ref ECSTaskExecutionRole
```

```
      ContainerDefinitions:
        - Name: !Ref ServiceName
          Cpu: !Ref ContainerCpu
          Memory: !Ref ContainerMemory
          Image: !Ref ImageUrl
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              HostPort: !Ref ContainerPort
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: !Ref ServiceName
  Service:
    Type: AWS::ECS::Service
    DependsOn: PublicLoadBalancerListener
    Properties:
      ServiceName: !Ref ServiceName
      Cluster: !Ref ClusterName
      PlacementStrategies:
        - Field: 'attribute:ecs.availability-zone'
          Type: spread
        - Field: cpu
          Type: binpack
      CapacityProviderStrategy:
        - Base: 0
          CapacityProvider: !Ref CapacityProvider
          Weight: 1
      NetworkConfiguration:
        AwsvpcConfiguration:
          SecurityGroups:
            - !Ref ServiceSecurityGroup
          Subnets: !Ref PrivateSubnetIds
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 75
      DesiredCount: !Ref DesiredCount
      TaskDefinition: !Ref TaskDefinition
      LoadBalancers:
        - ContainerName: !Ref ServiceName
          ContainerPort: !Ref ContainerPort
```

```
        TargetGroupArn: !Ref ServiceTargetGroup
ServiceSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: Security group for service
    VpcId: !Ref VpcId
ServiceTargetGroup:
  Type: 'AWS::ElasticLoadBalancingV2::TargetGroup'
  Properties:
    HealthCheckIntervalSeconds: 6
    HealthCheckPath: /
    HealthCheckProtocol: HTTP
    HealthCheckTimeoutSeconds: 5
    HealthyThresholdCount: 2
    TargetType: ip
    Port: !Ref ContainerPort
    Protocol: HTTP
    UnhealthyThresholdCount: 10
    VpcId: !Ref VpcId
    TargetGroupAttributes:
      - Key: deregistration_delay.timeout_seconds
        Value: 0
PublicLoadBalancerSG:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: Access to the public facing load balancer
    VpcId: !Ref VpcId
    SecurityGroupIngress:
      - CidrIp: 0.0.0.0/0
        IpProtocol: -1
PublicLoadBalancer:
  Type: 'AWS::ElasticLoadBalancingV2::LoadBalancer'
  Properties:
    Scheme: internet-facing
    LoadBalancerAttributes:
      - Key: idle_timeout.timeout_seconds
        Value: '30'
    Subnets: !Ref PublicSubnetIds
    SecurityGroups:
      - !Ref PublicLoadBalancerSG
PublicLoadBalancerListener:
  Type: 'AWS::ElasticLoadBalancingV2::Listener'
  Properties:
    DefaultActions:
```

```
        - Type: forward
          ForwardConfig:
            TargetGroups:
              - TargetGroupArn: !Ref ServiceTargetGroup
                Weight: 100
      LoadBalancerArn: !Ref PublicLoadBalancer
      Port: 80
      Protocol: HTTP
  ServiceIngressfromLoadBalancer:
    Type: 'AWS::EC2::SecurityGroupIngress'
    Properties:
      Description: Ingress from the public ALB
      GroupId: !Ref ServiceSecurityGroup
      IpProtocol: -1
      SourceSecurityGroupId: !Ref PublicLoadBalancerSG
  LogGroup:
    Type: 'AWS::Logs::LogGroup'
```

Amazon ECS에 대한 IAM 역할

AWS CloudFormation 템플릿을 사용하여 Amazon ECS에서 사용할 IAM 역할을 생성할 수 있습니다. Amazon ECS용 IAM 역할에 대한 자세한 내용은 Amazon ECS에 대한 IAM 역할 섹션을 참조하세요.

Amazon ECS 작업 실행 역할

태스크 실행 IAM 역할은 Amazon ECS 컨테이너 에이전트 및 Fargate 에이전트에 사용자를 대신하여 AWS API 호출을 수행할 권한을 부여합니다. 태스크의 요구 사항에 따라 이 역할이 필요합니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

다음 템플릿을 사용하여 AmazonECSTaskExecutionRolePolicy 관리형 정책을 사용하는 간단한 태스크 실행 역할을 생성할 수 있습니다.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "CloudFormation template for ECS Task Execution Role",
  "Resources": {
    "ECSTaskExecutionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
```

```
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": ["ecs-tasks.amazonaws.com"]
              },
              "Action": ["sts:AssumeRole"],
              "Condition": {
                "ArnLike": {
                  "aws:SourceArn": {
                    "Fn::Sub": "arn:aws:ecs:${AWS::Region}:${AWS::AccountId}:*"
                  }
                },
                "StringEquals": {
                  "aws:SourceAccount": {
                    "Ref": "AWS::AccountId"
                  }
                }
              }
            }
          ]
        },
        "Path": "/",
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
        ]
      }
    }
  },
  "Outputs": {
    "ECSTaskExecutionRoleARN": {
      "Description": "ARN of the ECS Task Execution Role",
      "Value": {
        "Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]
      },
      "Export": {
        "Name": {
          "Fn::Sub": "${AWS::StackName}-ECSTaskExecutionRoleARN"
        }
      }
    },
    "ECSTaskExecutionRoleName": {
      "Description": "Name of the ECS Task Execution Role",
      "Value": {
```

```
        "Ref": "ECSTaskExecutionRole"
      },
      "Export": {
        "Name": {
          "Fn::Sub": "${AWS::StackName}-ECSTaskExecutionRoleName"
        }
      }
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'CloudFormation template for ECS Task Execution Role'
Resources:
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: [ecs-tasks.amazonaws.com]
            Action: ['sts:AssumeRole']
            Condition:
              ArnLike:
                aws:SourceArn: !Sub arn:aws:ecs:${AWS::Region}:${AWS::AccountId}:*
              StringEquals:
                aws:SourceAccount: !Ref AWS::AccountId
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
Outputs:
  ECSTaskExecutionRoleARN:
    Description: ARN of the ECS Task Execution Role
    Value: !GetAtt ECSTaskExecutionRole.Arn
    Export:
      Name: !Sub "${AWS::StackName}-ECSTaskExecutionRoleARN"
  ECSTaskExecutionRoleName:
    Description: Name of the ECS Task Execution Role
    Value: !Ref ECSTaskExecutionRole
    Export:
```

```
      Name: !Sub "${AWS::StackName}-ECSTaskExecutionRoleName"
```

AWS CloudFormation 콘솔을 사용하여 Amazon ECS 리소스 생성

AWS CloudFormation에서 Amazon ECS를 사용하는 한 가지 방법은 AWS Management Console을 사용하는 것입니다. 여기에서 태스크 정의, 클러스터, 서비스와 같은 Amazon ECS 구성 요소를 위한 AWS CloudFormation 스택을 생성하고 콘솔에서 직접 배포할 수 있습니다. 다음 자습서에서는 AWS CloudFormation 콘솔을 사용하여 Amazon ECS 서비스, 태스크 정의 및 클러스터를 생성하는 방법을 보여줍니다.

사전 조건

이 자습서에서는 다음 사전 조건이 충족되었다고 가정합니다.

- Amazon ECS 사용 설정의 단계가 완료되었습니다.
- IAM 사용자는 AmazonECS_FullAccess IAM 정책 예제에 지정된 필수 권한을 가집니다.

1단계: 스택 템플릿 생성

다음 단계를 사용하여 Amazon ECS 서비스 및 기타 관련 리소스를 위한 AWS CloudFormation 스택 템플릿을 생성합니다.

1. 원하는 텍스트 편집기를 사용하여 `ecs-tutorial-template.yaml`이라는 파일을 생성합니다.
2. `ecs-tutorial-template.yaml` 파일에 다음 템플릿을 붙여 넣고 변경 사항을 저장합니다.

```
AWSTemplateFormatVersion: 2010-09-09
Description: A template that deploys an application that is built on an Apache
 web server Docker image by creating an Amazon ECS cluster, task definition, and
 service. The template also creates networking and logging resources, and an Amazon
 ECS task execution role.
Parameters:
  ClusterName:
    Type: String
    Default: CFNCluster
    Description: Name of the ECS Cluster
  TaskFamily:
    Type: String
    Default: task-definition-cfn
    Description: Family name for the Task Definition
```

```
  ServiceName:
    Type: String
    Default: cfn-service
    Description: Name of the ECS Service
  ContainerImage:
    Type: String
    Default: public.ecr.aws/docker/library/httpd:2.4
    Description: Container image to use for the task
  TaskCpu:
    Type: Number
    Default: 256
    AllowedValues: [256, 512, 1024, 2048, 4096]
    Description: CPU units for the task
  TaskMemory:
    Type: Number
    Default: 512
    AllowedValues: [512, 1024, 2048, 4096, 8192, 16384]
    Description: Memory (in MiB) for the task
  DesiredCount:
    Type: Number
    Default: 1
    Description: Desired number of tasks to run
  LogGroupName:
    Type: String
    Default: /ecs/fargate-task-definition
    Description: CloudWatch Log Group name
  VpcCidr:
    Type: String
    Default: 10.0.0.0/16
    Description: CIDR block for the VPC
  PublicSubnet1Cidr:
    Type: String
    Default: 10.0.0.0/24
    Description: CIDR block for public subnet 1
  PublicSubnet2Cidr:
    Type: String
    Default: 10.0.1.0/24
    Description: CIDR block for public subnet 2
Resources:
  # VPC and Networking Resources
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCidr
```

```
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-VPC
  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-IGW
  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC
  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PublicSubnet1Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-PublicSubnet1
  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PublicSubnet2Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-PublicSubnet2
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-PublicRouteTable
  DefaultPublicRoute:
```

```
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway
  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1
  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2
  # Security Group
  ECSSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Security group for ECS tasks
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 443
          ToPort: 443
          CidrIp: 0.0.0.0/0
  # IAM Roles
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
```

```
# CloudWatch Logs
TaskLogGroup:
  Type: AWS::Logs::LogGroup
  DeletionPolicy: Retain
  UpdateReplacePolicy: Retain
  Properties:
    LogGroupName: !Ref LogGroupName
    RetentionInDays: 30
# ECS Resources
ECSCluster:
  Type: AWS::ECS::Cluster
  Properties:
    ClusterName: !Ref ClusterName
ECSTaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    ContainerDefinitions:
      - Command:
          - >-
            /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
            App</title> <style>body {margin-top: 40px; background-color:
            #333;} </style> </head><body> <div
            style=color:white;text-align:center> <h1>Amazon ECS Sample
            App</h1> <h2>Congratulations!</h2> <p>Your application is now
            running on a container in Amazon ECS.</p> </div></body></html>' >
            /usr/local/apache2/htdocs/index.html && httpd-foreground"s
        EntryPoint:
          - sh
          - '-c'
        Essential: true
        Image: !Ref ContainerImage
        LogConfiguration:
          LogDriver: awslogs
          Options:
            mode: non-blocking
            max-buffer-size: 25m
            awslogs-create-group: 'true'
            awslogs-group: !Ref LogGroupName
            awslogs-region: !Ref 'AWS::Region'
            awslogs-stream-prefix: ecs
        Name: sample-fargate-app
        PortMappings:
          - ContainerPort: 80
            HostPort: 80
```

```
              Protocol: tcp
      Cpu: !Ref TaskCpu
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
      Family: !Ref TaskFamily
      Memory: !Ref TaskMemory
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      RuntimePlatform:
        OperatingSystemFamily: LINUX
  ECSService:
    Type: AWS::ECS::Service
    DependsOn:
      - PublicSubnet1RouteTableAssociation
      - PublicSubnet2RouteTableAssociation
    Properties:
      ServiceName: !Ref ServiceName
      Cluster: !Ref ECSCluster
      DesiredCount: !Ref DesiredCount
      LaunchType: FARGATE
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups:
            - !Ref ECSSecurityGroup
          Subnets:
            - !Ref PublicSubnet1
            - !Ref PublicSubnet2
      TaskDefinition: !Ref ECSTaskDefinition
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref ECSCluster
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref ECSTaskDefinition
  ServiceName:
    Description: The name of the ECS service
    Value: !Ref ECSService
  VpcId:
    Description: The ID of the VPC
    Value: !Ref VPC
  PublicSubnet1:
    Description: The ID of public subnet 1
```

```
    Value: !Ref PublicSubnet1
  PublicSubnet2:
    Description: The ID of public subnet 2
    Value: !Ref PublicSubnet2
  SecurityGroup:
    Description: The ID of the security group
    Value: !Ref ECSSecurityGroup
  ExecutionRoleArn:
    Description: The ARN of the task execution role
    Value: !GetAtt ECSTaskExecutionRole.Arn
```

2단계: Amazon ECS 리소스를 위한 스택 생성

템플릿 파일을 생성한 후 AWS CloudFormation 콘솔을 사용하여 다음 단계에 따라 템플릿으로 스택을 생성할 수 있습니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/cloudformation에서 AWS CloudFormation 콘솔을 엽니다.
2. 스택 페이지의 오른쪽 상단에서 스택 생성을 선택하고 새 리소스 사용(표준)을 선택합니다.
3. 기존 템플릿 선택을 선택합니다.
4. 템플릿 파일 업로드을 선택한 다음 파일 선택을 선택하여 `ecs-tutorial-template` 파일을 선택합니다.

   파일이 Amazon S3 버킷에 업로드된 후 Infrastructure Composer에서 보기를 선택하여 Infrastructure Composer에서 템플릿을 시각화할 수 있습니다. AWS CloudFormation 템플릿 및 Infrastructure Composer에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 Infrastructure Composer를 사용하여 시각적으로 템플릿 생성을 참조하세요.
5. 다음을 선택합니다.
6. 스택 세부 정보 지정 페이지의 스택 이름에서 스택의 이름을 입력합니다. `ecs-tutorial-stack`. 파라미터 아래의 모든 파라미터 값을 기본값 그대로 두고 다음을 선택합니다.
7. 스택 옵션 구성 페이지의 기능에서 AWS CloudFormation이 IAM 리소스를 생성하도록 승인하는 확인란을 선택합니다. 이 승인은 템플릿에 정의된 대로 Amazon ECS 태스크 실행 역할을 생성하는 데 필요합니다. 다른 모든 설정은 기본값 그대로 두고 다음을 선택합니다.
8. 검토 및 생성 페이지에서 스택 세부 정보를 검토한 다음 제출을 선택하여 스택 생성을 시작합니다.

3단계: 확인

다음 단계에 따라 제공된 템플릿을 사용한 Amazon ECS 리소스 생성을 확인합니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/cloudformation에서 AWS CloudFormation 콘솔을 엽니다.
2. 스택 페이지에서 ecs-tutorial-stack을 선택합니다.
3. 이벤트 탭을 선택합니다. 이벤트 상태가 `CREATE_IN_PROGRESS`로 표시되면 생성이 완료되고 상태가 `CREATE_COMPLETE`로 변경될 때까지 기다립니다.
4. 이벤트 상태가 `CREATE_COMPLETE`로 전환되면 리소스 탭을 선택합니다. 논리적 ID가 `ECSCluster`, , `ECSTaskDefinition` 및 `ECSService`인 리소스가 각각 표시됩니다.
5. Amazon ECS 클러스터 생성을 확인하려면 `ECSCluster`와 연결된 물리적 ID를 선택합니다. 그러면 생성된 `CFNCluster`라는 클러스터를 볼 수 있는 Amazon ECS 콘솔로 리디렉션됩니다.
6. Amazon ECS 서비스 생성을 확인하려면 `ECSService`와 연결된 물리적 ID를 선택합니다. 그러면 클러스터 `cfnCluster`에서 생성된 `cfn-service`라는 서비스를 볼 수 있는 Amazon ECS 콘솔로 리디렉션됩니다.
7. Amazon ECS 태스크 정의 생성을 확인하려면 `ECSTaskDefinition`과 연결된 물리적 ID를 선택합니다. 그러면 이름이 `task-definition-cfn`인 태스크 정의 개정을 볼 수 있는 Amazon ECS 콘솔로 리디렉션됩니다.

4단계: 리소스 정리

추가 비용이 발생하지 않도록 리소스를 정리하려면 다음 단계를 수행합니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/cloudformation에서 AWS CloudFormation 콘솔을 엽니다.
2. 스택 페이지에서 ecs-tutorial-stack을 선택합니다.
3. 삭제를 선택합니다.
4. 확인 메시지가 나타나면 다시 삭제를 선택합니다.
5. 이벤트 탭을 선택합니다. 상태가 DELETE_IN_PROGRESS로 변경된 다음, 리소스가 삭제되거나 등록 취소되면 DELETE_COMPLETE로 `ecs-tutorial-stack` 변경됩니다. 삭제하는 데 몇 분 정도 소요됩니다.
6. 리소스 탭을 선택합니다. 이제 상태가 DELETE_COMPLETE로 업데이트된 논리적 ID 목록이 표시됩니다.

AWS CLI용 AWS CloudFormation 명령을 사용하여 Amazon ECS 리소스 생성

다음 자습서에서는 AWS CLI를 사용하여 AWS CloudFormation 템플릿을 통해 Amazon ECS 리소스를 생성하는 방법을 보여줍니다.

사전 조건

- Amazon ECS 사용 설정의 단계가 완료되었습니다.
- IAM 사용자는 AmazonECS_FullAccess IAM 정책 예제에 지정된 필수 권한을 가집니다.

1단계: 스택 생성

AWS CLI를 사용하여 스택을 생성하려면 다음 단계를 수행합니다.

1. 원하는 텍스트 편집기를 사용하여 `ecs-tutorial-template.yaml`이라는 파일을 생성합니다.
2. `ecs-tutorial-template.yaml` 파일에 다음 템플릿을 붙여 넣고 변경 사항을 저장합니다.

```
AWSTemplateFormatVersion: 2010-09-09
Description: A template that deploys an application that is built on an Apache
 web server Docker image by creating an Amazon ECS cluster, task definition, and
 service. The template also creates networking and logging resources, and an Amazon
 ECS task execution role.
Parameters:
  ClusterName:
    Type: String
    Default: CFNCluster
    Description: Name of the ECS Cluster
  TaskFamily:
    Type: String
    Default: task-definition-cfn
    Description: Family name for the Task Definition
  ServiceName:
    Type: String
    Default: cfn-service
    Description: Name of the ECS Service
  ContainerImage:
    Type: String
    Default: public.ecr.aws/docker/library/httpd:2.4
    Description: Container image to use for the task
```

```
  TaskCpu:
    Type: Number
    Default: 256
    AllowedValues: [256, 512, 1024, 2048, 4096]
    Description: CPU units for the task
  TaskMemory:
    Type: Number
    Default: 512
    AllowedValues: [512, 1024, 2048, 4096, 8192, 16384]
    Description: Memory (in MiB) for the task
  DesiredCount:
    Type: Number
    Default: 1
    Description: Desired number of tasks to run
  LogGroupName:
    Type: String
    Default: /ecs/fargate-task-definition
    Description: CloudWatch Log Group name
  VpcCidr:
    Type: String
    Default: 10.0.0.0/16
    Description: CIDR block for the VPC
  PublicSubnet1Cidr:
    Type: String
    Default: 10.0.0.0/24
    Description: CIDR block for public subnet 1
  PublicSubnet2Cidr:
    Type: String
    Default: 10.0.1.0/24
    Description: CIDR block for public subnet 2
Resources:
  # VPC and Networking Resources
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCidr
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-VPC
  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
```

```
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-IGW
  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC
  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PublicSubnet1Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-PublicSubnet1
  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PublicSubnet2Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-PublicSubnet2
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-PublicRouteTable
  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway
  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
```

```
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1
  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2
  # Security Group
  ECSSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Security group for ECS tasks
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 443
          ToPort: 443
          CidrIp: 0.0.0.0/0
  # IAM Roles
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
  # CloudWatch Logs
  TaskLogGroup:
    Type: AWS::Logs::LogGroup
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      LogGroupName: !Ref LogGroupName
      RetentionInDays: 30
```

```
# ECS Resources
ECSCluster:
  Type: AWS::ECS::Cluster
  Properties:
    ClusterName: !Ref ClusterName
ECSTaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    ContainerDefinitions:
      - Command:
          - >-
            /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
            App</title> <style>body {margin-top: 40px; background-color:
            #333;} </style> </head><body> <div
            style=color:white;text-align:center> <h1>Amazon ECS Sample
            App</h1> <h2>Congratulations!</h2> <p>Your application is now
            running on a container in Amazon ECS.</p> </div></body></html>' >
            /usr/local/apache2/htdocs/index.html && httpd-foreground"s
        EntryPoint:
          - sh
          - '-c'
        Essential: true
        Image: !Ref ContainerImage
        LogConfiguration:
          LogDriver: awslogs
          Options:
            mode: non-blocking
            max-buffer-size: 25m
            awslogs-create-group: 'true'
            awslogs-group: !Ref LogGroupName
            awslogs-region: !Ref 'AWS::Region'
            awslogs-stream-prefix: ecs
        Name: sample-fargate-app
        PortMappings:
          - ContainerPort: 80
            HostPort: 80
            Protocol: tcp
    Cpu: !Ref TaskCpu
    ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
    Family: !Ref TaskFamily
    Memory: !Ref TaskMemory
    NetworkMode: awsvpc
    RequiresCompatibilities:
      - FARGATE
```

```
      RuntimePlatform:
        OperatingSystemFamily: LINUX
  ECSService:
    Type: AWS::ECS::Service
    DependsOn:
      - PublicSubnet1RouteTableAssociation
      - PublicSubnet2RouteTableAssociation
    Properties:
      ServiceName: !Ref ServiceName
      Cluster: !Ref ECSCluster
      DesiredCount: !Ref DesiredCount
      LaunchType: FARGATE
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups:
            - !Ref ECSSecurityGroup
          Subnets:
            - !Ref PublicSubnet1
            - !Ref PublicSubnet2
      TaskDefinition: !Ref ECSTaskDefinition
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref ECSCluster
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref ECSTaskDefinition
  ServiceName:
    Description: The name of the ECS service
    Value: !Ref ECSService
  VpcId:
    Description: The ID of the VPC
    Value: !Ref VPC
  PublicSubnet1:
    Description: The ID of public subnet 1
    Value: !Ref PublicSubnet1
  PublicSubnet2:
    Description: The ID of public subnet 2
    Value: !Ref PublicSubnet2
  SecurityGroup:
    Description: The ID of the security group
    Value: !Ref ECSSecurityGroup
  ExecutionRoleArn:
```

```
    Description: The ARN of the task execution role
    Value: !GetAtt ECSTaskExecutionRole.Arn
```

3. 템플릿 파일을 생성한 후 다음 명령을 사용하여 스택을 생성합니다. `--capabilities` 플래그는 템플릿에 지정된 대로 Amazon ECS 태스크 실행 역할을 생성하는 데 필요합니다. `--parameters` 플래그를 지정하여 템플릿 파라미터를 사용자 지정할 수도 있습니다.

```
aws cloudformation create-stack \
      --stack-name ecs-stack \
      --template-body file://ecs-tutorial-template.yaml \
      --capabilities CAPABILITY_IAM
```

2단계: 리소스 생성 확인

리소스가 올바르게 생성되었는지 확인하려면 다음 단계를 수행합니다. Amazon ECS 콘솔에서 확인할 수도 있습니다.

1. 다음 명령을 실행하여 AWS 리전의 모든 태스크 정의를 나열합니다.

```
aws ecs list-task-definitions
```

이 명령은 태스크 정의 Amazon 리소스 이름(ARN) 목록을 반환합니다. 템플릿을 사용하여 생성한 태스크 정의의 ARN은 다음 형식으로 표시됩니다.

```
{
    "taskDefinitionArns": [
     .....
        "arn:aws:ecs:aws-region:111122223333:task-definition/task-definition-
cfn:1",
     .....
    ]
}
```

2. 다음 명령을 실행하여 AWS 리전의 모든 클러스터를 나열합니다.

```
aws ecs list-clusters
```

이 명령은 클러스터 ARN 목록을 반환합니다. 템플릿을 사용하여 생성한 클러스터의 ARN은 다음 형식으로 표시됩니다.

```
{
    "clusterArns": [
        .....
        "arn:aws:ecs:aws-region:111122223333:cluster/CFNCluster",
        .....
    ]
}
```

3. 다음 명령을 실행하여 클러스터 CFNCluster의 모든 서비스를 나열합니다.

```
aws ecs list-services \
      --cluster CFNCluster
```

이 명령은 서비스 ARN 목록을 반환합니다. 템플릿을 사용하여 생성한 서비스의 ARN은 다음 형식으로 표시됩니다.

```
{
    "serviceArns": [
        "arn:aws:ecs:aws-region:111122223333:service/CFNCluster/cfn-service"
    ]
}
```

3단계: 정리

생성한 리소스를 정리하려면 다음 명령을 실행합니다.

```
aws cloudformation delete-stack \
      --stack-name ecs-stack
```

delete-stack 명령은 이 자습서에서 생성된 AWS CloudFormation 스택의 삭제를 시작하여 스택의 모든 리소스를 삭제합니다. 삭제를 확인하려면 2단계: 리소스 생성 확인의 절차를 반복하면 됩니다. 출력의 ARN 목록에는 더 이상 task-definition-cfn이라는 태스크 정의 또는 CFNCLuster라는 클러스터가 포함되지 않습니다. list-services 호출이 실패합니다.

Amazon ECS 모범 사례

다음 페이지에서 Amazon ECS 네트워킹에 대한 가장 중요한 운영 모범 사례를 알아볼 수 있습니다.

모범 사례 개요	자세히 알아보기
애플리케이션을 인터넷에 연결	Amazon ECS 애플리케이션을 인터넷에 연결
인터넷에서 Amazon ECS로의 인바운드 연결을 수신합니다.	인터넷에서 Amazon ECS로의 인바운드 연결을 수신하는 모범 사례
VPC 내의 다른 AWS 서비스에 Amazon ECS 연결	VPC 내에서 Amazon ECS를 AWS 서비스에 연결하는 모범 사례
AWS 계정 및 VPC 전반의 네트워크 서비스	AWS 계정 및 VPC 간의 Amazon ECS 서비스 네트워킹을 위한 모범 사례
네트워크 문제 해결	Amazon ECS 네트워킹 문제 해결을 위한 AWS 서비스

다음 페이지에서 Amazon ECS의 Fargate에 대한 가장 중요한 운영 모범 사례를 알아볼 수 있습니다.

모범 사례 개요	자세히 알아보기
Fargate 보안	Amazon ECS의 Fargate 보안 모범 사례
Fargate 보안 고려 사항	Amazon ECS에 대한 Fargate 보안 고려 사항
Fargate 컨테이너 이미지 가져오기 동작의 Linux 컨테이너	Amazon ECS에 대한 Fargate 컨테이너 이미지 가져오기 동작에서 Linux 컨테이너

모범 사례 개요	자세히 알아보기	
Fargate 컨테이너 이미지 가져오기 동작의 Windows 컨테이너	Amazon ECS에 대한 Fargate 컨테이너 이미지 가져오기 동작에서 Windows 컨테이너	
Fargate 태스크 사용 중지	Amazon ECS에서 AWS Fargate에 대한 태스크 사용 중지 및 유지 관리	

다음 페이지에서 태스크 정의에 대한 가장 중요한 운영 모범 사례를 알아볼 수 있습니다.

모범 사례 개요	자세히 알아보기	
컨테이너 이미지	Amazon ECS 컨테이너 이미지 모범 사례	
태스크 크기	Amazon ECS 작업 크기 모범 사례	
볼륨 모범 사례	Amazon ECS 작업에 대한 스토리지 옵션	

다음 페이지에서 클러스터와 용량에 대한 가장 중요한 운영 모범 사례를 알아볼 수 있습니다.

모범 사례	자세히 알아보기	
Fargate 보안	Amazon ECS의 Fargate 보안 모범 사례	
EC2 컨테이너 인스턴스 보안 고려 사항	Amazon ECS에 대한 Amazon EC2 컨테이너 인스턴스 보안 고려 사항	
클러스터 Auto Scaling	Amazon ECS 클러스터 Auto Scaling 최적화	

모범 사례	자세히 알아보기
대규모 운영	대규모 Amazon ECS 운영
오토 스케일링 및 용량 관리	Amazon ECS 오토 스케일링 및 용량 관리 모범 사례

다음 페이지에서 태스크와 서비스에 대한 가장 중요한 운영 모범 사례를 알아볼 수 있습니다.

모범 사례 개요	자세히 알아보기
태스크 시작 시간 최적화	Amazon ECS 태스크 시작 시간 최적화
서비스 파라미터	Amazon ECS 서비스 파라미터의 모범 사례
로드 밸런서 상태 확인 파라미터 최적화	Amazon ECS에 대한 로드 밸런서 상태 확인 파라미터 최적화
로드 밸런서 Connection Draining 파라미터 최적화	Amazon ECS에 대한 로드 밸런서 Connection Draining 파라미터 최적화
서비스 Auto Scaling 최적화	Amazon ECS 서비스 오토 스케일링 최적화

다음 페이지에서 보안에 대한 가장 중요한 운영 모범 사례를 알아볼 수 있습니다.

모범 사례 개요	자세히 알아보기
네트워크 보안	Amazon ECS에 대한 네트워크 보안 모범 사례
작업 및 컨테이너 보안	Amazon ECS 태스크 및 컨테이너 보안 모범 사례

Amazon ECS용 AWS Fargate

AWS Fargate는 Amazon EC2 인스턴스의 서버나 클러스터를 관리할 필요 없이 컨테이너를 실행하기 위해 Amazon ECS에 사용할 수 있는 기술입니다. AWS Fargate를 사용하면 더 이상 컨테이너를 실행하기 위해 가상 머신의 클러스터를 프로비저닝, 구성 또는 조정할 필요가 없습니다. 따라서 서버 유형을 선택하거나, 클러스터를 조정할 시점을 결정하거나, 클러스터 패킹을 최적화할 필요가 없습니다.

Fargate 시작 유형을 사용하여 태스크와 서비스를 실행할 때는 애플리케이션을 컨테이너에 패키징하고, CPU 및 메모리 요구 사항을 지정한 다음, 네트워킹 및 IAM 정책을 정의하고, 애플리케이션을 시작합니다. 각 Fargate 태스크에는 자체 격리 경계가 있으며 다른 태스크와 기본 커널, CPU 리소스, 메모리 리소스 또는 탄력적 네트워크 인터페이스를 공유하지 않습니다. `requiresCompatibilities` 작업 정의 파라미터를 `FARGATE`로 설정하여 Fargate에 대한 작업 정의를 구성합니다. 자세한 내용은 시작 유형 섹션을 참조하세요.

Fargate는 Amazon Linux 2(플랫폼 버전 1.3.0), Bottlerocket 운영 체제(플랫폼 버전 1.4.0) 및 Microsoft Windows 2019 Server Full and Core 에디션용 플랫폼 버전을 제공합니다. 달리 지정되지 않는 한이 페이지의 정보는 모든 Fargate 플랫폼에 적용됩니다.

이 주제에서는 Fargate 태스크와 서비스의 다양한 구성 요소를 설명하고 Amazon ECS에 Fargate를 사용하기 위한 특수한 고려 사항을 살펴봅니다.

Fargate에서 Linux 컨테이너가 지원되는 리전에 대한 자세한 정보는 the section called “AWS Fargate의 Linux 컨테이너” 섹션을 참조하세요.

Fargate에서 Windows 컨테이너가 지원되는 리전에 대한 자세한 정보는 the section called “AWS Fargate의 Windows 컨테이너” 섹션을 참조하세요.

연습

콘솔을 사용하여 시작하는 방법은 다음을 참조하세요.

- Fargate 시작 유형에 대한 Amazon ECS Linux 작업을 생성하는 방법 알아보기
- Fargate 시작 유형에 대한 Amazon ECS Windows 작업을 생성하는 방법 알아보기

AWS CLI를 사용하여 시작하는 방법은 다음을 참조하세요.

- AWS CLI를 사용하여 Fargate 시작 유형에 대한 Amazon ECS Linux 작업 생성

- AWS CLI를 사용하여 Fargate 시작 유형에 대한 Amazon ECS Windows 작업 생성

용량 공급자

다음과 같은 용량 공급자를 사용할 수 있습니다.

- Fargate
- Fargate 스팟 - AWS Fargate 가격 대비 할인된 요금으로 중단 방지 Amazon ECS 작업을 실행합니다. Fargate 스팟은 여분의 컴퓨팅 용량에 대한 태스크를 실행합니다. AWS에 용량이 다시 필요한 경우 2분간 경고한 후에 태스크가 중단됩니다. 자세한 내용은 Fargate용 Amazon ECS 클러스터 섹션을 참조하세요.

태스크 정의

Fargate 시작 유형을 사용하는 태스크는 사용 가능한 Amazon ECS 태스크 정의 파라미터 중 일부를 지원하지 않을 수 있습니다. 전혀 지원되지 않는 파라미터도 있고 Fargate 작업에 다르게 작동하는 파라미터도 있습니다. 자세한 내용은 태스크 CPU 및 메모리 섹션을 참조하세요.

플랫폼 버전

AWS Fargate 플랫폼 버전은 Fargate 태스크 인프라를 위한 특정 실행 시간 환경을 참조하는 데 사용합니다. 이것은 커널 버전과 컨테이너 실행 시간 버전의 조합입니다. 작업을 실행하거나 여러 개의 동일한 작업을 유지 관리하는 서비스를 생성할 플랫폼 버전을 선택합니다.

플랫폼 버전의 새 개정판은 커널 또는 운영 체제 업데이트, 새로운 기능, 버그 수정 또는 보안 업데이트처럼 런타임 환경이 개선됨에 따라 릴리스됩니다. Fargate 플랫폼 버전은 새 플랫폼 버전 개정판을 통해 업데이트됩니다. 각 작업은 해당 수명 주기 동안 하나의 플랫폼 버전 개정판에서 실행됩니다. 최신 플랫폼 버전 개정판을 사용하려면 새 작업을 시작해야 합니다. Fargate에서 실행되는 새 작업은 항상 플랫폼 버전의 최신 개정판에서 실행되므로 항상 안전하고 패치가 적용된 인프라에서 작업을 시작할 수 있습니다.

기존 플랫폼 버전에 영향을 미치는 보안 문제가 발견되면 AWS는 플랫폼 버전에 패치를 적용한 새 개정판을 만들고 취약한 개정판에서 실행 중인 작업을 중지합니다. 사용자에게 Fargate 작업이 만료 예정이라는 알림이 전송되는 경우도 있습니다. 자세한 내용은 Amazon ECS에서 AWS Fargate에 대한 태스크 사용 중지 및 유지 관리 섹션을 참조하세요.

자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.

서비스 로드 밸런싱

Elastic Load Balancing을 사용하여 서비스의 태스크 간에 트래픽을 고르게 분산하도록 AWS Fargate의 Amazon ECS 서비스를 구성할 수도 있습니다.

AWS Fargate의 Amazon ECS 서비스는 Application Load Balancer 및 Network Load Balancer 로드 밸런서 유형을 지원합니다. Application Load Balancer는 HTTP/HTTPS(또는 계층 7) 트래픽을 라우팅하는 데 사용합니다. Network Load Balancer는 TCP 또는 UDP(또는 계층 4) 트래픽을 라우팅하는 데 사용합니다. 자세한 내용은 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산 섹션을 참조하세요.

이러한 서비스에 대한 대상 그룹을 생성할 때 대상 유형을 `instance`가 아닌 `ip`로 선택해야 합니다. 이는 `awsvpc` 네트워크 모드를 사용하는 작업이 Amazon EC2 인스턴스가 아닌 탄력적 네트워크 인터페이스와 연결되기 때문입니다. 자세한 내용은 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산 섹션을 참조하세요.

Network Load Balancer를 사용하여 AWS Fargate 태스크의 Amazon ECS로 UDP 트래픽을 라우팅하는 것은 플랫폼 버전 1.4 이상을 사용할 때만 지원됩니다.

사용량 지표

CloudWatch 사용량 지표를 사용하여 계정의 리소스 사용량을 확인할 수 있습니다. 이러한 지표를 사용하여 CloudWatch 그래프 및 대시보드에서 현재 서비스 사용량을 시각화합니다.

AWS Fargate 사용량 지표는 AWS 서비스 할당량에 해당합니다. 사용량이 서비스 할당량에 가까워지면 경고하는 경보를 구성할 수 있습니다. AWS Fargate 서비스 할당량에 대한 자세한 내용은 Amazon Web Services 일반 참조의 Amazon ECS 엔드포인트 및 할당량을 참조하세요.

AWS Fargate 사용량 지표에 대한 자세한 내용은 AWS Fargate사용량 지표를 참조하세요.

Fargate 시작 유형 사용 시기에 대한 Amazon ECS 보안 고려 사항

엄격한 작업 격리를 원하는 고객은 Fargate를 사용하는 것이 좋습니다. Fargate는 하드웨어 가상화 환경에서 각 작업을 실행합니다. 이러한 컨테이너화된 워크로드는 네트워크 인터페이스, Fargate 임시 스토리지, CPU 또는 메모리를 다른 작업과 공유하지 않습니다. 자세한 내용은 Security Overview of AWS Fargate를 참조하세요.

Amazon ECS의 Fargate 보안 모범 사례

AWS Fargate를 사용할 때 다음 모범 사례를 고려하는 것이 좋습니다. 추가 지침은 Security overview of AWS Fargate를 참조하세요.

AWS KMS를 사용하여 Fargate를 위한 임시 스토리지 암호화

AWS KMS 또는 고객 관리형 키로 임시 스토리지를 암호화해야 합니다. 기본적으로 플랫폼 버전 `1.4.0` 이상을 사용하여 Fargate에서 호스팅되는 작업은 각각 최소 20GiB 이상의 임시 스토리지를 받습니다. 자세한 내용은 고객 관리형 키(CMK)를 참조하세요. 임시 스토리지의 총량은 작업 정의에 `ephemeralStorage` 파라미터를 지정하여 최대 200GiB까지 늘릴 수 있습니다. 2020년 5월 28일 이후 시작된 작업의 경우, Fargate에서 관리하는 암호화 키를 사용하는 AES-256 암호화 알고리즘으로 임시 스토리지가 암호화됩니다.

자세한 내용은 Amazon ECS 작업에 대한 스토리지 옵션을 참조하세요.

예: 임시 스토리지 암호화를 사용하여 Fargate 플랫폼 버전 1.4.0에서 작업 시작

다음 명령은 Fargate 플랫폼 버전 1.4에서 작업을 시작합니다. 이 작업은 클러스터의 일부로 시작되므로 자동으로 암호화되는 20GiB의 임시 스토리지를 사용합니다.

```
aws ecs run-task --cluster clustername \
  --task-definition taskdefinition:version \
  --count 1
  --launch-type "FARGATE" \
  --platform-version 1.4.0 \
  --network-configuration
 "awsvpcConfiguration={subnets=[subnetid],securityGroups=[securitygroupid]}" \
  --region region
```

Fargate를 사용한 커널 시스템 호출 추적을 위한 SYS_PTRACE 기능

컨테이너에서 추가되거나 삭제되는 Linux 기능의 기본 구성은 Docker에서 제공됩니다.

Fargate에서 시작된 태스크는 SYS_PTRACE 커널 기능 추가만 지원합니다.

다음 비디오는 Sysdig Falco 프로젝트를 통해 이 기능을 사용하는 방법을 보여줍니다.

#ContainersFromTheCouch - Troubleshooting your Fargate Task using SYS_PTRACE capability

앞의 비디오에서 설명한 코드는 여기 GitHub에서 찾을 수 있습니다.

Fargate Runtime Monitoring과 함께 Amazon GuardDuty 사용

Amazon GuardDuty는 AWS 환경 내 계정, 컨테이너, 워크로드 및 데이터를 보호하는 데 도움이 되는 위협 감지 서비스입니다. GuardDuty는 기계 학습(ML) 모델, 이상 및 위협 감지 기능을 통해 다양한 로그 소스와 런타임 활동을 지속적으로 모니터링하여 사용자 환경의 잠재적 보안 위험과 악의적 활동을 식별하고 우선순위를 지정합니다.

GuardDuty의 Runtime Monitoring은 AWS 로그 및 네트워킹 활동을 지속적으로 모니터링하여 악의적 동작 또는 무단 동작을 식별함으로써 Fargate에서 실행되는 워크로드를 보호합니다. Runtime Monitoring은 파일 액세스, 프로세스 실행 및 네트워크 연결과 같은 호스트 내 동작을 분석하는 경량의 완전관리형 GuardDuty 보안 에이전트를 사용합니다. 여기에는 권한 에스컬레이션, 노출된 자격 증명 사용 또는 악의적인 IP 주소 및 도메인과의 통신, 그리고 Amazon EC2 인스턴스 및 컨테이너 워크로드에 존재하는 맬웨어 같은 문제가 포함됩니다. 자세한 내용은 GuardDuty 사용 설명서의 GuardDuty Runtime Monitoring을 참조하세요.

Amazon ECS에 대한 Fargate 보안 고려 사항

Fargate는 각 워크로드를 격리된 가상 환경에서 실행하므로 작업마다 전용 인프라 용량이 있습니다. Fargate에서 실행되는 워크로드는 네트워크 인터페이스, 임시 스토리지, CPU 또는 메모리를 다른 작업과 공유하지 않습니다. 작업 내에서 애플리케이션 컨테이너와 사이드카 컨테이너 또는 단순히 사이드카를 비롯한 여러 컨테이너를 실행할 수 있습니다. 사이드카는 Amazon ECS 작업에서 애플리케이션 컨테이너와 함께 실행되는 컨테이너입니다. 애플리케이션 컨테이너는 코어 애플리케이션 코드를 실행하는 반면 사이드카에서 실행되는 프로세스는 애플리케이션을 확대할 수 있습니다. 사이드카를 사용하면 애플리케이션 기능을 전용 컨테이너로 분리하여 애플리케이션의 부분을 더 쉽게 업데이트할 수 있습니다.

동일한 작업에 속하는 컨테이너는 항상 동일한 호스트에서 실행되고 컴퓨팅 리소스를 공유하므로 Fargate 시작 유형의 리소스를 공유합니다. 이러한 컨테이너는 Fargate에서 제공하는 임시 스토리지도 공유합니다. 작업의 Linux 컨테이너는 IP 주소 및 네트워크 포트를 비롯한 네트워크 네임스페이스를 공유합니다. 작업 내에서 작업에 속한 컨테이너는 로컬 호스트를 통해 상호 통신할 수 있습니다.

Fargate의 런타임 환경에서는 EC2 인스턴스에서 지원되는 특정 컨트롤러 기능을 사용하지 못합니다. Fargate에서 실행되는 워크로드를 설계할 때 다음 사항을 고려하세요.

- 권한 있는 컨테이너 또는 액세스 없음 - 권한 있는 컨테이너 또는 액세스와 같은 기능은 현재 Fargate에서 사용할 수 없습니다. 이는 Docker에서 Docker 실행과 같은 사용 사례에 영향을 미칩니다.
- Linux 기능에 대한 제한된 액세스 - 컨테이너가 Fargate에서 실행되는 환경은 잠겨 있습니다. CAP_SYS_ADMIN 및 CAP_NET_ADMIN과 같은 추가 Linux 기능은 권한 에스컬레이션을 방지하기

위해 제한됩니다. Fargate는 작업에 CAP_SYS_PTRACE Linux 기능을 추가하여 작업 내에 배포된 관찰성 및 보안 도구에서 컨테이너식 애플리케이션을 모니터링할 수 있도록 지원합니다.

- 기본 호스트에 액세스할 수 없음 - 고객과 AWS 운영자 모두 고객 워크로드를 실행하는 호스트에 연결할 수 없습니다. ECS exec을 사용하여 명령을 실행하거나 Fargate에서 실행되는 컨테이너로 셸을 가져올 수 있습니다. ECS exec을 사용하여 디버깅을 위한 진단 정보를 수집할 수 있습니다. 또한 Fargate는 컨테이너가 파일 시스템, 디바이스, 네트워킹 및 컨테이너 런타임과 같은 기본 호스트의 리소스에 액세스하지 못하게 합니다.
- 네트워킹 - 보안 그룹 및 네트워크 ACL을 사용하여 인바운드 및 아웃바운드 트래픽을 제어할 수 있습니다. Fargate 작업은 VPC에 구성된 서브넷에서 IP 주소를 받습니다.

Amazon ECS에 대한 Fargate 플랫폼 버전

AWS Fargate 플랫폼 버전은 Fargate 태스크 인프라를 위한 특정 실행 시간 환경을 참조하는 데 사용합니다. 이것은 커널 버전과 컨테이너 실행 시간 버전의 조합입니다. 작업을 실행하거나 여러 개의 동일한 작업을 유지 관리하는 서비스를 생성할 플랫폼 버전을 선택합니다.

플랫폼 버전의 새 개정판은 커널 또는 운영 체제 업데이트, 새로운 기능, 버그 수정 또는 보안 업데이트처럼 런타임 환경이 개선됨에 따라 릴리스됩니다. Fargate 플랫폼 버전은 새 플랫폼 버전 개정판을 통해 업데이트됩니다. 각 작업은 해당 수명 주기 동안 하나의 플랫폼 버전 개정판에서 실행됩니다. 최신 플랫폼 버전 개정판을 사용하려면 새 작업을 시작해야 합니다. Fargate에서 실행되는 새 작업은 항상 플랫폼 버전의 최신 개정판에서 실행되므로 항상 안전하고 패치가 적용된 인프라에서 작업을 시작할 수 있습니다.

기존 플랫폼 버전에 영향을 미치는 보안 문제가 발견되면 AWS는 플랫폼 버전에 패치를 적용한 새 개정판을 만들고 취약한 개정판에서 실행 중인 작업을 중지합니다. 사용자에게 Fargate 작업이 만료 예정이라는 알림이 전송되는 경우도 있습니다. 자세한 내용은 Amazon ECS에서 AWS Fargate에 대한 태스크 사용 중지 및 유지 관리 섹션을 참조하세요.

태스크 정의를 생성하거나 서비스를 배포할 때 플랫폼 버전을 지정합니다.

플랫폼 버전을 지정할 때 다음 사항을 고려해야 합니다.

- `1.4.0`, LATEST 등과 같은 특정 버전 번호를 지정할 수 있습니다.

  LATEST Linux 플랫폼 버전은 `1.4.0`입니다.

  LATEST Windows 플랫폼 버전은 `1.0.0`입니다.

- 서비스의 플랫폼 버전을 업데이트하려면 배포를 생성하세요. 예를 들어 Linux 플랫폼 버전 `1.3.0`에서 작업을 실행하는 서비스가 있다고 가정합니다. Linux 플랫폼 버전 `1.4.0`에서 작업을 실행하도록 서비스를 변경하려면 서비스를 업데이트하고 새 플랫폼 버전을 지정합니다. 작업은 최신 플랫폼 버전과 최신 플랫폼 버전 개정판으로 재배포됩니다. 배포에 대한 자세한 내용은 Amazon ECS 서비스 섹션을 참조하세요.
- 플랫폼 버전을 업데이트하지 않고 서비스를 확장하면 해당 태스크는 서비스의 현재 배포에 지정된 플랫폼 버전을 수신합니다. 예를 들어 Linux 플랫폼 버전 `1.3.0`에서 작업을 실행하는 서비스가 있다고 가정합니다. 원하는 만큼 서비스 수를 늘리면 서비스 스케줄러는 플랫폼 버전 `1.3.0`의 최신 개정판을 사용하여 새 작업을 시작합니다.
- 새 작업은 항상 플랫폼 버전의 최신 개정판에서 실행됩니다. 따라서 항상 보안이 유지되고 패치가 적용된 인프라에서 작업을 시작할 수 있습니다.
- Fargate에서 Linux 컨테이너와 Windows 컨테이너의 플랫폼 버전 번호는 독립적입니다. 예를 들어 Fargate의 Windows 컨테이너용 플랫폼 버전 `1.0.0` 및 Fargate의 Linux 컨테이너용 플랫폼 버전 `1.0.0`에서 사용되는 동작, 기능 및 소프트웨어는 서로 비교할 수 없습니다.
- 다음은 Fargate Windows 플랫폼 버전에 적용됩니다.

  Microsoft Windows Server 컨테이너 이미지는 특정 버전의 Windows Server에서 생성되어야 합니다. 작업을 실행하거나 서비스를 생성할 때 `platformFamily`에서 해당 Windows Server 컨테이너 이미지와 일치하는 동일한 버전의 Windows Server를 선택해야 합니다. 또한 작업 정의에 일치하는 `operatingSystemFamily`를 제공하여 작업이 잘못된 Windows 버전에서 실행되지 않도록 할 수 있습니다. 자세한 내용은 Microsoft Learn 웹 사이트에서 Matching container host version with container image versions를 참조하세요.

Amazon ECS에서 Linux 플랫폼 버전 1.4.0으로 마이그레이션

Fargate의 Amazon ECS 태스크를 플랫폼 버전 `1.0.0`, `1.1.0`, `1.2.0` 또는 `1.3.0`에서 플랫폼 버전 `1.4.0`으로 마이그레이션할 때 고려해야 할 사항은 다음과 같습니다. 태스크를 마이그레이션하기 전에 플랫폼 버전 `1.4.0`에서 태스크가 올바르게 작동하는지 확인하는 것이 좋습니다.

- 태스크를 주고받은 네트워크 트래픽 동작이 업데이트되었습니다. 플랫폼 버전 1.4.0부터 모든 Fargate의 Amazon ECS 태스크는 단일 탄력적 네트워크 인터페이스(태스크 ENI라고 함)를 받고, 모든 네트워크 트래픽은 VPC 내의 해당 ENI를 통해 흐르게 됩니다. 사용자는 VPC 흐름 로그를 통해 이를 볼 수 있습니다. 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션을 참조하세요.
- 인터페이스 VPC 엔드포인트를 사용하는 경우 다음을 고려해야 합니다.

- Amazon ECR로 호스팅되는 컨테이너 이미지의 경우 다음 엔드포인트가 필요합니다. 자세한 정보는 Amazon Elastic Container Registry 사용 설명서의 Amazon ECR 인터페이스 VPC 엔드포인트 (AWS PrivateLink)를 참조하세요.
  - com.amazonaws.***region***.ecr.dkr Amazon ECR VPC 엔드포인트
  - com.amazonaws.***region***.ecr.api Amazon ECR VPC 엔드포인트
  - Amazon S3 게이트웨이 엔드포인트
- 태스크 정의에서 Secret Manager 보안 암호를 참조하여 컨테이너에 대한 중요한 데이터를 검색하는 경우 Secret Manager용 인터페이스 VPC 엔드포인트를 생성해야 합니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 VPC 엔드포인트와 함께 Secrets Manager 사용을 참조하세요.
- 태스크 정의에서 Systems Manager Parameter Store 파라미터를 참조하여 컨테이너에 대한 중요한 데이터를 검색하는 경우, System Manager용 인터페이스 VPC 엔드포인트를 생성해야 합니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 Systems Manager용 VPC 엔드포인트를 사용하여 EC2 인스턴스의 보안 개선을 참조하세요.
- 태스크와 연결된 탄력적 네트워크 인터페이스(ENI)의 보안 그룹에는 해당 태스크와 VPC 엔드포인트 간의 트래픽을 허용하도록 만들어진 보안 그룹 규칙이 있어야 합니다.

Fargate Linux 플랫폼 버전 변경 로그

사용 가능한 Linux 플랫폼 버전은 다음과 같습니다. 플랫폼 버전 사용 중단에 대한 자세한 정보는 AWS Fargate Linux 플랫폼 버전 사용 중단 섹션을 참조하세요.

1.4.0

플랫폼 버전 `1.4.0`의 변경 사항은 다음과 같습니다.

- 2020년 11월 5일부터 플랫폼 버전 `1.4.0`을 사용하여 Fargate에서 시작한 새로운 Amazon ECS 태스크는 다음의 기능을 사용할 수 있습니다.
  - Secrets Manager를 사용하여 민감한 데이터를 저장하는 경우, 특정 JSON 키 또는 특정 암호 버전을 환경 변수로 삽입하거나, 로그 구성에 삽입할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.
  - `environmentFiles` 컨테이너 정의 파라미터를 사용하여 환경 변수를 일괄 지정합니다. 자세한 정보는 개별 환경 변수를 Amazon ECS 컨테이너로 전달 섹션을 참조하세요.
  - VPC에서 실행되는 태스크와 IPv6에 활성화된 서브넷은 프라이빗 IPv4 주소와 IPv6 주소가 모두 할당됩니다. 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션을 참조하세요.

  - 태스크 메타데이터 엔드포인트 버전 4는 태스크 시작 유형, 컨테이너의 Amazon 리소스 이름(ARN), 로그 드라이버, 사용한 로그 드라이버 옵션 등을 포함한 태스크와 컨테이너 관련 추가 메타데이터를 제공합니다. `/stats` 엔드포인트를 쿼리하면 컨테이너의 네트워크 속도 통계도 수신합니다. 자세한 내용은 작업 메타데이터 엔드포인트 버전 4를 참조하세요.
- 2020년 7월 30일부터 플랫폼 버전 `1.4.0`을 사용하여 Fargate에서 시작한 새로운 Amazon ECS 태스크는 Network Load Balancer를 사용하여 Fargate의 Amazon ECS 태스크로 UDP 트래픽을 라우팅할 수 있습니다. 자세한 정보는 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산 섹션을 참조하세요.
- 2020년 5월 28일부터 플랫폼 버전 `1.4.0`을 사용하여 Fargate에서 시작되는 새로운 Amazon ECS는 AWS에서 소유한 암호화 키를 사용하여 AES-256 암호화 알고리즘으로 휘발성 스토리지를 암호화합니다. 자세한 내용은 Amazon ECS에 대한 Fargate 작업 임시 스토리지 및 Amazon ECS 작업에 대한 스토리지 옵션(을)를 참조하세요.
- 영구 태스크 스토리지를 위한 Amazon EFS 파일 시스템 볼륨 사용에 대한 지원이 추가되었습니다. 자세한 정보는 Amazon ECS에서 Amazon EFS 볼륨 사용 섹션을 참조하세요.
- 각 태스크의 휘발성 태스크 스토리지가 최소 20GB로 증가했습니다. 자세한 내용은 Amazon ECS에 대한 Fargate 작업 임시 스토리지 섹션을 참조하세요.
- 태스크를 주고받은 네트워크 트래픽 동작이 업데이트되었습니다. 플랫폼 버전 1.4.0부터 모든 Fargate 태스크는 단일 탄력적 네트워크 인터페이스(태스크 ENI라고 함)를 받고, 모든 네트워크 트래픽은 VPC 내에서 해당 ENI를 통해 흐르게 됩니다. 사용자는 VPC 흐름 로그를 통해 이를 볼 수 있습니다. Amazon EC2 시작 유형의 네트워킹에 대한 자세한 내용은 EC2 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션을 참조하세요. Fargate 시작 유형의 네트워킹에 대한 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션 섹션을 참조하세요.
- 태스크 ENI는 점보 프레임에 대한 지원을 추가합니다. 네트워크 인터페이스는 단일 프레임 내에 맞는 가장 큰 페이로드 크기인 최대 전송 단위(MTU)로 구성됩니다. MTU가 클수록 단일 프레임 내에 더 많은 애플리케이션 페이로드가 들어갈 수 있으므로 프레임당 오버헤드가 줄어들고 효율성이 향상됩니다. 점보 프레임을 지원하면 VPC 내에 남아 있는 모든 트래픽과 같이 태스크와 대상 간의 네트워크 경로가 점보 프레임을 지원할 때 오버헤드가 줄어듭니다.
- CloudWatch Container Insights가 Fargate 태스크에 대한 네트워크 성능 지표를 포함합니다. 자세한 정보는 관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.
- 태스크에 대한 네트워크 통계 및 태스크를 실행 중인 가용 영역 등 Fargate 태스크에 대한 추가 정보를 제공하는 태스크 메타데이터 엔드포인트 버전 4에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 작업 메타데이터 엔드포인트 버전 4 및 Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 엔드포인트 버전 4(을)를 참조하세요.

- 컨테이너 정의에 SYS_PTRACE Linux 파라미터에 대한 지원이 추가되었습니다. 자세한 정보는 Linux 파라미터 섹션을 참조하세요.
- Fargate 컨테이너 에이전트가 모든 Fargate 태스크에 대해 Amazon ECS 컨테이너 에이전트의 사용을 대체합니다. 일반적으로 이 변경 사항은 태스크 실행 방식에 영향을 주지 않습니다.
- 컨테이너 런타임은 이제 Docker 대신 Containerd를 사용합니다. 이 변경 사항은 태스크 실행 방식에 영향을 주지 않을 가능성이 큽니다. 컨테이너 런타임에 시작되는 일부 오류 메시지는 Docker를 언급하는 것에서 보다 일반적인 오류로 변경됩니다. 자세한 내용은 Amazon ECS 중지된 작업 오류 메시지를 참조하세요.
- Amazon Linux 2를 기반으로 합니다.

1.3.0

플랫폼 버전 `1.3.0`의 변경 사항은 다음과 같습니다.

- 2019년 9월 30일부터 시작되는 새로운 Fargate 태스크는 `awsfirelens` 로그 드라이버를 지원합니다. 작업 정의 파라미터를 사용하여 로그를 저장하고 분석하기 위해 로그를 AWS 서비스 또는 AWS 파트너 네트워크(APN)를 대상으로 라우팅하도록 FireLens for Amazon ECS를 구성합니다. 자세한 내용은 Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송 섹션을 참조하세요.
- Fargate 태스크의 태스크 재생이 추가되었습니다. 이 기능은 Amazon ECS 서비스에 속하는 태스크를 새로 고치는 프로세스입니다. 자세한 내용은 Amazon ECS에서AWS Fargate에 대한 태스크 사용 중지 및 유지 관리를 참조하세요.
- 2019년 3월 27일부터 새로운 Fargate 태스크는 프록시 구성, 컨테이너 시작 및 종료는 물론, 컨테이너별 시작 및 중지 제한 시간 값에 대한 종속성을 정의하는 데 사용하는 태스크 정의 파라미터를 추가로 출시했습니다. 자세한 정보는 프록시 구성, 컨테이너 종속성, 컨테이너 제한 시간 섹션을 참조하세요.
- 2019년 4월 2일부터 새로운 Fargate 태스크는 AWS Secrets Manager 암호 또는 AWS Systems Manager 파라미터 스토어 파라미터에 중요한 데이터를 저장한 다음 컨테이너 정의에서 참조하여 중요한 데이터를 컨테이너에 주입할 수 있습니다. 자세한 정보는 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.
- 2019년 5월 1일부터 시작되는 새로운 Fargate 태스크는 `secretOptions` 컨테이너 정의 파라미터를 사용하여 컨테이너의 로그 구성에서 민감한 데이터를 참조하는 것을 지원합니다. 자세한 정보는 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.
- 2019년 5월 1일부터 시작되는 새로운 Fargate 태스크는 `awslogs` 로그 드라이버 외에 `splunk` 로그 드라이버도 지원합니다. 자세한 정보는 스토리지 및 로깅 섹션을 참조하세요.

- 2019년 7월 9일부터 시작되는 모든 새로운 Fargate 태스크는 CloudWatch Container Insights를 지원합니다. 자세한 정보는 관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.
- 2019년 12월 3일부터 Fargate 스팟 용량 공급자가 지원됩니다. 자세한 정보는 Fargate용 Amazon ECS 클러스터 섹션을 참조하세요.
- Amazon Linux 2를 기반으로 합니다.

AWS Fargate Linux 플랫폼 버전 사용 중단

이 페이지에는 AWS Fargate에서 사용 중단하거나 사용 중단 예정인 Linux 플랫폼 버전이 나와 있습니다. 이 플랫폼 버전은 공지된 사용 중단 날짜까지 사용할 수 있습니다.

사용 중단 예정인 각 플랫폼 버전에 강제 업데이트 날짜가 제공됩니다. 강제 업데이트 날짜에 사용 중단 예정인 플랫폼 버전을 가리키는 LATEST 플랫폼 버전을 사용하는 모든 서비스는 강제 신규 배포 옵션을 사용하여 업데이트됩니다. 강제 신규 배포 옵션을 사용하여 서비스를 업데이트하면 사용 중단 예정인 플랫폼 버전에서 실행되는 모든 태스크가 중단되고 해당 시점에 LATEST 태그가 가리키는 플랫폼 버전을 사용하여 새로운 태스크가 시작됩니다. 명시적 플랫폼 버전이 설정된 독립적 태스크나 서비스는 강제 업데이트에 영향을 받지 않습니다.

독립적 태스크가 가장 최신 플랫폼 버전을 사용하도록 서비스를 업데이트하는 것이 좋습니다. 가장 최신 플랫폼 버전으로 마이그레이션하는 방법에 대한 자세한 정보는 Amazon ECS에서 Linux 플랫폼 버전 1.4.0으로 마이그레이션을 참조하세요.

플랫폼 버전이 사용 중단 날짜에 도달하면 해당 플랫폼 버전은 새로운 태스크나 서비스에 사용할 수 없게 됩니다. 사용 중단된 플랫폼 버전을 명시적으로 사용하는 독립적 태스크나 서비스는 태스크가 중단될 때까지 해당 플랫폼 버전을 계속 사용합니다. 사용 중단 날짜 이후에는 사용 중단된 플랫폼 버전은 더 이상 보안 업데이트나 버그 수정을 받지 못합니다.

플랫폼 버전	강제 업데이트 날짜	사용 중단 날짜
1.0.0	2020년 10월 26일	2020년 12월 14일
1.1.0	2020년 10월 26일	2020년 12월 14일
1.2.0	2020년 10월 26일	2020년 12월 14일

현재 플랫폼 버전에 대한 자세한 정보는 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.

사용 중단된 Fargate Linux 버전 변경 로그

1.2.0

플랫폼 버전 `1.2.0`의 변경 사항은 다음과 같습니다.

> **Note**
>
> 플랫폼 버전 `1.2.0`을 더 이상 사용할 수 없습니다. 플랫폼 버전 사용 중단에 대한 자세한 정보는 AWS Fargate Linux 플랫폼 버전 사용 중단 섹션을 참조하세요.

- AWS Secrets Manager을 사용하는 프라이빗 레지스트리 인증 지원이 추가되었습니다. 자세한 정보는 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용 섹션을 참조하세요.

1.1.0

플랫폼 버전 `1.1.0`의 변경 사항은 다음과 같습니다.

> **Note**
>
> 플랫폼 버전 `1.1.0`을 더 이상 사용할 수 없습니다. 플랫폼 버전 사용 중단에 대한 자세한 정보는 AWS Fargate Linux 플랫폼 버전 사용 중단 섹션을 참조하세요.

- Amazon ECS 태스크 메타데이터 엔드포인트에 대한 지원이 추가되었습니다. 자세한 내용은 Fargate의 작업에 대해 사용 가능한 Amazon ECS 작업 메타데이터 섹션을 참조하세요.
- 컨테이너 정의에서 Docker 상태 확인에 대한 지원이 추가되었습니다. 자세한 정보는 상태 확인 섹션을 참조하세요.
- Amazon ECS 서비스 검색에 대한 지원이 추가되었습니다. 자세한 정보는 서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결 섹션을 참조하세요.

1.0.0

플랫폼 버전 `1.0.0`의 변경 사항은 다음과 같습니다.

> **Note**
>
> 플랫폼 버전 `1.0.0`을 더 이상 사용할 수 없습니다. 플랫폼 버전 사용 중단에 대한 자세한 정보는 AWS Fargate Linux 플랫폼 버전 사용 중단 섹션을 참조하세요.

- Amazon Linux 2017.09를 기반으로 합니다.
- 최초 릴리스.

Fargate Windows 플랫폼 버전 변경 로그

Windows 컨테이너에 사용 가능한 플랫폼 버전은 다음과 같습니다.

1.0.0

플랫폼 버전 `1.0.0`의 변경 사항은 다음과 같습니다.

- 다음 Microsoft Windows Server 운영 체제 지원을 위한 초기 릴리스:
  - Windows Server 2019 Full
  - Windows Server 2019 Core
  - Windows Server 2022 Full
  - Windows Server 2022 Core

Amazon ECS에 대한 Fargate의 Windows 컨테이너 고려 사항

다음은 AWS Fargate에서 Windows 컨테이너를 실행할 때 알아야 할 차이점과 고려 사항입니다.

Linux 및 Windows 컨테이너에서 작업을 실행해야 하는 경우 각 운영 체제에 대해 별도의 작업 정의를 생성해야 합니다.

AWS에서 운영 체제 라이선스 관리를 처리하므로 추가 Microsoft Windows Server 라이선스가 필요하지 않습니다.

AWS Fargate의 Windows 컨테이너는 다음 운영 체제를 지원합니다.

- Windows Server 2019 Full
- Windows Server 2019 Core

- Windows Server 2022 Full
- Windows Server 2022 Core

AWS Fargate의 Windows 컨테이너는 awslogs 드라이버를 지원합니다. 자세한 내용은 the section called “CloudWatch에 로그 전송 ” 섹션을 참조하세요.

다음 기능은 Fargate의 Windows 컨테이너에서 지원되지 않습니다.

- Amazon FSx
- ENI 트렁킹
- Windows 컨테이너용 gMSA
- 태스크를 위한 App Mesh 서비스 및 프록시 통합
- 태스크를 위한 FireLens 로그 라우터 통합
- EFS 볼륨
- EBS 볼륨
- 다음 작업 정의 파라미터:
  - `maxSwap`
  - `swappiness`
  - `environmentFiles`
- Fargate 스팟 용량 공급자
- 이미지 볼륨

  Dockerfile `volume` 옵션은 무시됩니다. 대신 태스크 정의에서 바인드 탑재를 사용합니다. 자세한 내용은 Amazon ECS에서 바인드 탑재 사용 섹션을 참조하세요.
- Windows 컨테이너에 대해서는 태스크 레벨 CPU와 메모리 파라미터가 무시됩니다. Windows 컨테이너에 대해서는 컨테이너 레벨 리소스를 지정할 것을 권장합니다.
- 태스크용 메모리
- 컨테이너의 mermoryReservation
- 컨테이너에서 정책 다시 시작

Amazon ECS에 대한 Fargate 작업 임시 스토리지

프로비저닝되면 AWS Fargate의 Linux 컨테이너에서 호스팅되는 각 Amazon ECS 태스크는 바인드 탑재를 위해 다음 임시 스토리지를 받게 됩니다. 작업 정의에서 `volumes`, `mountPoints` 및 `volumesFrom` 파라미터를 사용하여 이 볼륨을 탑재하고 컨테이너 간에 공유할 수 있습니다. AWS Fargate의 Windows 컨테이너에서는 지원되지 않습니다.

Fargate Linux 컨테이너 플랫폼 버전

버전 1.4.0 이상

기본적으로 플랫폼 버전 `1.4.0` 이상을 사용하여 Fargate에서 호스팅되는 Amazon ECS 태스크는 최소 20GiB 이상의 임시 스토리지를 받습니다. 임시 스토리지의 총량은 최대 200GiB까지 높일 수 있습니다. 태스크 정의에서 `ephemeralStorage` 파라미터를 지정하여 태스크를 수행할 수 있습니다.

태스크의 풀링, 압축 및 비압축 컨테이너 이미지는 임시 스토리지에 저장됩니다. 태스크가 사용해야 하는 임시 스토리지의 총량을 확인하려면 컨테이너 이미지가 사용하는 스토리지 용량을 태스크에 할당된 임시 스토리지의 총 용량에서 빼세요.

2020년 5월 28일 이후 시작된 플랫폼 버전 `1.4.0` 이상을 사용하는 태스크의 경우, AES-256 암호화 알고리즘으로 임시 스토리지가 암호화됩니다. 이 알고리즘은 AWS에서 소유한 암호화 키를 사용합니다. 고객 관리형 키를 생성할 수도 있습니다. 자세한 내용은 AWS Fargate 임시 스토리지에 대한 고객 관리형 키를 참조하세요.

2022년 11월 18일 이후 시작된 플랫폼 버전 `1.4.0` 이상을 사용하는 작업의 경우 작업 메타데이터 엔드포인트를 통해 임시 스토리지 사용량이 보고됩니다. 작업의 애플리케이션은 작업 메타데이터 엔드포인트 버전 4를 쿼리하여 임시 스토리지 예약 크기 및 사용량을 가져올 수 있습니다.

또한 Amazon CloudWatch Container Insights를 켜면 임시 스토리지 예약 크기와 사용된 양이 Amazon CloudWatch Container Insights로 전송됩니다.

> **Note**
>
> Fargate는 디스크 공간을 예약합니다. Fargate에서만 사용됩니다. 이에 대한 요금은 청구되지 않습니다. 이러한 지표에는 표시되지 않습니다. 그러나 `df` 등의 다른 도구에서는 이 추가 스토리지를 볼 수 있습니다.

버전 1.3.0 이하

플랫폼 버전 `1.3.0` 이전 버전을 사용하는 Fargate의 Amazon ECS 태스크는 각각 다음과 같은 임시 스토리지를 받습니다.

- 10GB의 Docker 계층 스토리지

> **Note**
>
> 이 용량에는 압축 및 비압축 컨테이너 이미지 아티팩트가 모두 포함됩니다.

- 볼륨 마운트를 위한 추가 4GB입니다. 작업 정의에서 `volumes`, `mountPoints` 및 `volumesFrom` 파라미터를 사용하여 이 볼륨을 탑재하고 컨테이너 간에 공유할 수 있습니다.

Fargate Windows 컨테이너 플랫폼 버전

버전 1.0.0 이상

기본적으로 플랫폼 버전 `1.0.0` 이상을 사용하여 Fargate에서 호스팅되는 Amazon ECS 태스크는 최소 20GiB 이상의 임시 스토리지를 받습니다. 임시 스토리지의 총량은 최대 200GiB까지 높일 수 있습니다. 태스크 정의에서 `ephemeralStorage` 파라미터를 지정하여 태스크를 수행할 수 있습니다.

태스크의 풀링, 압축 및 비압축 컨테이너 이미지는 임시 스토리지에 저장됩니다. 태스크가 사용해야 하는 임시 스토리지의 총량을 확인하려면 컨테이너 이미지가 사용하는 스토리지 용량을 태스크에 할당된 임시 스토리지의 총 용량에서 빼세요.

자세한 내용은 Amazon ECS에서 바인드 탑재 사용 섹션을 참조하세요.

Amazon ECS용 AWS Fargate 임시 스토리지에 대한 고객 관리형 키

AWS Fargate에서는 임시 스토리지에 저장된 Amazon ECS 작업의 데이터를 암호화하는 데 고객 관리형 키를 지원하므로 규제에 민감한 고객이 내부 보안 정책을 준수할 수 있도록 지원합니다. 고객은 여전히 Fargate의 서버리스 이점을 누리면서 규정 준수 감사자에게 자체 관리형 스토리지 암호화에 대한 향상된 가시성을 제공합니다. Fargate는 기본적으로 Fargate에서 관리하는 임시 스토리지 암호화를 지원하지만 고객은 금융 또는 의료 관련 정보와 같은 민감한 데이터를 암호화할 때 자체 관리형 키를 사용할 수도 있습니다.

자체 키를 AWS KMS에 가져오거나 AWS KMS에서 키를 새로 생성할 수 있습니다. 이러한 자체 관리형 키는 AWS KMS에 저장되며 교체, 비활성화, 삭제와 같은 표준 AWS KMS 수명 주기 작업을 수행합니다. CloudTrail 로그에서 키 액세스 및 사용을 감사할 수 있습니다.

기본적으로 KMS 키는 키당 50,000개의 권한 부여를 지원합니다. Fargate는 고객 관리형 키 작업당 단일 AWS KMS 권한 부여를 사용하므로 키에 대해 최대 50,000개의 동시 작업을 지원합니다. 이 수를 늘리려면 한도 증가를 요청할 수 있으며, 해당 요청은 사례별로 승인됩니다.

Fargate는 고객 관리형 키를 사용하는 데 추가 비용을 청구하지 않습니다. 스토리지 및 API 요청에 AWS KMS 키를 사용하는 경우에만 표준 요금이 청구됩니다.

주제

- Amazon ECS용 Fargate 임시 스토리지에 대한 암호화 키 생성
- Amazon ECS용 Fargate 임시 스토리지에 대한 AWS KMS 키 관리

Amazon ECS용 Fargate 임시 스토리지에 대한 암호화 키 생성

고객 관리형 키를 생성하여 Fargate 임시 스토리지에 저장된 데이터를 암호화합니다.

> **Note**
>
> 고객 관리형 키를 사용하는 Fargate 임시 스토리지 암호화는 Windows 작업 클러스터에서 사용할 수 없습니다.
> 고객 관리형 키를 사용하는 Fargate 임시 스토리지 암호화는 `1.4.0` 이전의 `platformVersions`에서 사용할 수 없습니다.
> Fargate는 Fargate에서만 사용하는 임시 스토리지에 공간을 예약하며, 이 공간에 대해서는 요금이 청구되지 않습니다. 할당은 고객 관리형 키가 아닌 경우와 다를 수 있지만, 전체 공간은 동일합니다. `df`와 같은 도구에서 어떻게 다른지 확인할 수 있습니다.
> Fargate 임시 스토리지에 대해서는 다중 리전 키가 지원되지 않습니다.
> Fargate 임시 스토리지에 대해서는 KMS 키 별칭이 지원되지 않습니다.

AWS KMS에서 Fargate의 임시 스토리지를 암호화하기 위해 고객 관리형 키(CMK)를 생성하려면 다음 단계를 수행합니다.

1. https://console.aws.amazon.com/kms로 이동합니다.
2. 관련 지침은 AWS Key Management Service 개발자 안내서의 Creating Keys를 참조하세요.
3. AWS KMS 키를 생성할 때는 키 정책에 Fargate 서비스 관련 AWS KMS 작업 권한을 제공해야 합니다. Amazon ECS 리소스에서 고객 관리형 키를 사용하려면 정책에서 다음 API 작업을 허용해야 합니다.

- `kms:GenerateDataKeyWithoutPlainText` - 제공된 AWS KMS 키에서 암호화된 데이터 키를 생성하려면 `GenerateDataKeyWithoutPlainText`를 직접 호출합니다.
- `kms:CreateGrant` - 고객 관리형 키에 권한 부여를 추가합니다. 권한 부여는 지정된 AWS KMS 키에 대한 액세스를 제어하며, 여기서 Amazon ECS Fargate에 필요한 권한 부여 작업에 대한 액세스를 허용합니다. 권한 부여 사용에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서를 참조하십시오. 이를 통해 Amazon ECS Fargate에서 다음을 수행할 수 있습니다.
  - 임시 스토리지 데이터를 복호화하는 데 사용할 암호화 키를 가져오기 위해 AWS KMS로 `Decrypt`를 직접 호출합니다.
  - 서비스가 `RetireGrant`를 사용할 수 있도록 은퇴하는 보안 주체를 설정하세요.
- `kms:DescribeKey` - Amazon ECS에서 키가 대칭이고 활성화되었는지 확인할 수 있도록 고객 관리형 키 세부 정보를 제공합니다.

다음 예제에서는 암호화를 위해 대상 키에 적용할 AWS KMS 키 정책을 보여줍니다. 다음 예시 정책 명령을 사용하려면 *### ## ## ###*를 실제 정보로 바꿉니다. 항상 그렇듯이 필요한 권한만 구성하되, 오류를 방지하려면 최소한 한 명의 사용자에게 권한이 있는 AWS KMS를 제공해야 합니다.

```
{
      "Sid": "Allow generate data key access for Fargate tasks.",
      "Effect": "Allow",
      "Principal": { "Service":"fargate.amazonaws.com" },
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext"
      ],
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:aws:ecs:clusterAccount": [
            "customerAccountId"
          ],
          "kms:EncryptionContext:aws:ecs:clusterName": [
             "clusterName"
          ]
        }
      },
      "Resource": "*"
    },
    {
```

```
      "Sid": "Allow grant creation permission for Fargate tasks.",
      "Effect": "Allow",
      "Principal": { "Service":"fargate.amazonaws.com" },
      "Action": [
        "kms:CreateGrant"
      ],
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:aws:ecs:clusterAccount": [
            "customerAccountId"
          ],
          "kms:EncryptionContext:aws:ecs:clusterName": [
             "clusterName"
          ]
        },
       "ForAllValues:StringEquals": {
          "kms:GrantOperations": [
             "Decrypt"
          ]
       }
      },
      "Resource": "*"
    },
    {
      "Sid": "Allow describe key permission for cluster operator - CreateCluster
 and UpdateCluster.",
      "Effect": "Allow",
      "Principal": { "AWS":"arn:aws:iam::customerAccountId:role/customer-chosen-
role" },
      "Action": [
        "kms:DescribeKey"
      ],
      "Resource": "*"
    }
```

Fargate 작업은 키를 사용한 암호화 작업에 `aws:ecs:clusterAccount` 및 `aws:ecs:clusterName` 암호화 컨텍스트 키를 사용합니다. 특정 계정 및/또는 클러스터에 대한 액세스를 제한하려면 고객이 이러한 권한을 추가해야 합니다. 클러스터를 지정할 때 ARN이 아닌 클러스터 이름을 사용하세요.

자세한 내용은 AWS KMS 개발자 가이드에서 암호화 컨텍스트를 참조하세요.

클러스터를 생성하거나 업데이트할 때 조건 키 `fargateEphemeralStorageKmsKeyId`를 사용할 수 있습니다. 이 조건 키를 사용하면 고객이 IAM 정책을 보다 세밀하게 제어할 수 있습니다. `fargateEphemeralStorageKmsKeyId` 구성 업데이트는 새 서비스 배포에만 적용됩니다.

다음은 고객이 승인된 특정 AWS KMS 키 집합에만 권한을 부여하도록 허용하는 예제입니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:UpdateCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "ecs:fargate-ephemeral-storage-kms-key": "arn:aws:kms:us-
west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        }
      }
    }
  ]
}
```

다음은 클러스터와 이미 연결된 AWS KMS 키를 제거하려는 시도를 거부하는 예제입니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": [
        "ecs:CreateCluster",
        "ecs:UpdateCluster"
```

```
    ],
    "Resource": "*",
    "Condition": {
      "Null": {
        "ecs:fargate-ephemeral-storage-kms-key": "true"
      }
    }
  }
}
```

고객은 AWS CLI `describe-tasks`, `describe-cluster` 또는 `describe-services` 명령을 통해 비관리형 작업이나 서비스 작업이 키를 사용하여 암호화되는지 확인할 수 있습니다.

자세한 내용은 AWS KMS 개발자 안내서의 AWS KMS에 사용되는 조건 키를 참조하세요.

AWS Management Console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 왼쪽 탐색에서 클러스터를 선택하고 오른쪽 상단에서 클러스터 생성 선택하거나 기존 클러스터를 선택합니다. 기존 클러스터의 경우 오른쪽 상단에서 클러스터 업데이트를 선택합니다.
3. 워크플로의 암호화 섹션 아래 관리형 스토리지 및 Fargate 임시 스토리지에서 AWS KMS 키를 선택할 수 있습니다. 여기에서 AWS KMS 키 생성을 선택할 수도 있습니다.
4. 새 클러스터 생성을 완료한 후 생성을 선택하거나 기존 클러스터를 업데이트한 경우 업데이트를 선택합니다.

AWS CLI

다음은 AWS CLI를 사용하여 클러스터를 생성하고 Fargate 임시 스토리지를 구성하는 예제입니다 (*###* 값을 자체 값으로 바꿈).

```
aws ecs create-cluster --cluster clusterName \
--configuration '{"managedStorageConfiguration":
{"fargateEphemeralStorageKmsKeyId":"arn:aws:kms:us-
west-2:012345678901:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"}}'
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-west-2:012345678901:cluster/clusterName",
        "clusterName": "clusterName",
        "configuration": {
```

```
            "managedStorageConfiguration": {
                "fargateEphemeralStorageKmsKeyId": "arn:aws:kms:us-
west-2:012345678901:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
            }
        },
        "status": "ACTIVE",
        "registeredContainerInstancesCount": 0,
        "runningTasksCount": 0,
        "pendingTasksCount": 0,
        "activeServicesCount": 0,
        "statistics": [],
        "tags": [],
        "settings": [],
        "capacityProviders": [],
        "defaultCapacityProviderStrategy": []
    },
    "clusterCount": 5
}
```

AWS CloudFormation

다음은 AWS CloudFormation를 사용하여 클러스터를 생성하고 Fargate 임시 스토리지를 구성하는 예제 템플릿입니다(*###* 값을 자체 값으로 바꿈).

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  MyCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: "clusterName"
      Configuration:
        ManagedStorageConfiguration:
          FargateEphemeralStorageKmsKeyId: "arn:aws:kms:us-
west-2:012345678901:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Amazon ECS용 Fargate 임시 스토리지에 대한 AWS KMS 키 관리

Fargate 임시 스토리지를 암호화하기 위해 AWS KMS 키를 생성하거나 가져온 후에는 다른 AWS KMS 키와 동일한 방식으로 관리합니다.

AWS KMS 키 자동 교체

자동 키 교체를 활성화하거나 수동으로 교체할 수 있습니다. 자동 키 교체는 키에 대한 새로운 암호화 자료를 생성하여 키를 매년 자동으로 교체합니다. 또한 AWS KMS에서는 이전 버전의 모든 암호화 자료를 저장하므로 이전 키 버전을 사용한 모든 데이터를 복호화할 수 있습니다. 교체된 자료는 키를 삭제할 때까지 AWS KMS에서 삭제되지 않습니다.

자동 키 교체는 선택 사항이며, 언제라도 활서화하거나 비활성화할 수 있습니다.

AWS KMS 키 비활성화 또는 취소

AWS KMS에서 고객 관리형 키를 비활성화해도 실행 중인 작업에는 영향을 주지 않으며 수명 주기 동안 계속 작동합니다. 새 작업에서 비활성화되거나 취소된 키를 사용하는 경우 키에 액세스할 수 없으므로 작업이 실패합니다. 이미 암호화된 데이터를 복호화하는 데 비활성화된 키가 필요하지 않도록 하려면 CloudWatch 경보 또는 유사한 기능을 설정해야 합니다.

AWS KMS 키 삭제

키 삭제는 항상 최후의 수단이어야 하며, 삭제된 키가 다시 필요하지 않다고 확신하는 경우에만 삭제해야 합니다. 삭제된 키를 사용하려는 새 작업의 경우 해당 작업에서 키에 액세스할 수 없기 때문에 작업이 실패합니다. AWS KMS에서는 키를 삭제하는 대신, 키를 활성화할 것을 권장합니다. 키를 삭제해야 한다고 생각되면 먼저 키를 비활성화하고 CloudWatch 경보를 설정하여 키가 필요하지 않은지 확인하는 것이 좋습니다. 키를 삭제한 경우 AWS KMS에서는 설정을 복구할 7일 이상의 유예 기간을 제공합니다.

AWS KMS 액세스 키 감사

CloudTrail 로그를 사용하여 AWS KMS 키에 대한 액세스를 감사할 수 있습니다. AWS KMS 작업 `CreateGrant`, `GenerateDataKeyWithoutPlaintext` 및 `Decrypt`를 확인할 수 있습니다. 또한 이러한 작업에서는 CloudTrail에서 로그인한 `EncryptionContext`의 일부로 `aws:ecs:clusterAccount` 및 `aws:ecs:clusterName`도 표시합니다.

다음은 `GenerateDataKeyWithoutPlaintext`, `GenerateDataKeyWithoutPlaintext (DryRun)`, `CreateGrant`, `CreateGrant (DryRun)` 및 `RetireGrant`에 대한 예제 CloudTrail 이벤트입니다(### 값을 자체 값으로 바꿈).

GenerateDataKeyWithoutPlaintext

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ec2-frontend-api.amazonaws.com"
```

```
    },
    "eventTime": "2024-04-23T18:08:13Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyWithoutPlaintext",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ec2-frontend-api.amazonaws.com",
    "userAgent": "ec2-frontend-api.amazonaws.com",
    "requestParameters": {
        "numberOfBytes": 64,
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "encryptionContext": {
            "aws:ecs:clusterAccount": "account-id",
            "aws:ebs:id": "vol-xxxxxxx",
            "aws:ecs:clusterName": "cluster-name"
        }
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

GenerateDataKeyWithoutPlaintext (DryRun)

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "fargate.amazonaws.com"
```

```
    },
    "eventTime": "2024-04-23T18:08:11Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyWithoutPlaintext",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "fargate.amazonaws.com",
    "userAgent": "fargate.amazonaws.com",
    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is
 set.",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "dryRun": true,
        "numberOfBytes": 64,
        "encryptionContext": {
            "aws:ecs:clusterAccount": "account-id",
            "aws:ecs:clusterName": "cluster-name"
        }
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```


CreateGrant


```
{
    "eventVersion": "1.08",
```

```
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ec2-frontend-api.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:08:13Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ec2-frontend-api.amazonaws.com",
    "userAgent": "ec2-frontend-api.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "granteePrincipal": "fargate.us-west-2.amazonaws.com",
        "operations": [
            "Decrypt"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:ecs:clusterAccount": "account-id",
                "aws:ebs:id": "vol-xxxx",
                "aws:ecs:clusterName": "cluster-name"
            }
        },
        "retiringPrincipal": "ec2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
```

```
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

CreateGrant (DryRun)

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "fargate.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:08:11Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "fargate.amazonaws.com",
    "userAgent": "fargate.amazonaws.com",
    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is
 set.",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "granteePrincipal": "fargate.us-west-2.amazonaws.com",
        "dryRun": true,
        "operations": [
            "Decrypt"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:ecs:clusterAccount": "account-id",
                "aws:ecs:clusterName": "cluster-name"
            }
        }
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
```

```
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

RetireGrant

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2024-04-20T18:37:38Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "RetireGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
    },
    "additionalEventData": {
        "grantId":
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
```

```
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

Amazon ECS에서 AWS Fargate에 대한 태스크 사용 중지 및 유지 관리

AWS에서는 AWS Fargate의 기본 인프라를 유지 관리합니다. AWS에서는 플랫폼 버전 개정을 인프라의 새 개정으로 교체해야 하는 시기를 결정합니다. 이를 작업 사용 중지라고 합니다. AWS에서는 플랫폼 버전 개정이 사용 중지되면 작업 사용 중지 알림을 보냅니다. 지원되는 플랫폼 버전을 정기적으로 업데이트하여 Fargate 런타임 소프트웨어 및 운영 체제 및 컨테이너 런타임과 같은 기본 종속성에 대한 업데이트를 포함하는 새 개정을 도입합니다. 최신 개정이 사용 가능해지면 모든 고객 워크로드가 Fargate 플랫폼 버전의 최신 개정에서 실행되도록 하기 위해 이전 개정을 사용 중지합니다. 수정 버전이 사용 중지되면 해당 수정 버전에서 실행 중인 모든 작업은 중지됩니다.

Amazon ECS 작업은 서비스 작업 또는 독립 실행형 작업으로 분류할 수 있습니다. 서비스 작업은 서비스의 일부로 배포되고 Amazon ECS 일정에서 제어합니다. 자세한 내용은 Amazon ECS 서비스 섹션을 참조하세요. 독립 실행형 작업은 예약된 작업(Amazon EventBridge에서 시작), AWS Batch 또는 AWS Step Functions와 같은 외부 스케줄러에서 시작하거나 직접 Amazon ECS RunTask API에서 시작하는 작업입니다. Amazon ECS 스케줄러가 자동으로 태스크를 대체하므로 서비스 태스크에 대한 태스크 사용 중지에 대응하여 어떤 조치도 취할 필요가 없습니다.

독립 실행형 태스크의 경우 태스크 사용 중지에 대응하여 추가적인 처리를 수행해야 할 수도 있습니다. 자세한 내용은 Amazon ECS에서 독립 실행형 작업을 자동으로 처리할 수 있나요? 섹션을 참조하세요.

서비스 태스크의 경우 AWS 전에 먼저 이러한 태스크를 대체하지 않는 한, 태스크 사용 중지에 대한 어떤 조치도 취할 필요가 없습니다. Amazon ECS 스케줄러에서 태스크를 중지하는 경우 `maximumPercent`를 사용하며, 원하는 서비스 개수를 유지하기 위해 새 태스크를 시작합니다. 모범 사례를 따라 태스크 사용 중지의 영향을 최소화합니다. REPLICA 서비스 스케줄러를 사용하는 서비스의 기본 `maximumPercent` 값은 200%입니다. 따라서 AWS Fargate가 태스크 사용 중지를 시작하면

Amazon ECS는 먼저 새 태스크를 예약한 다음 해당 태스크가 실행될 때까지 기다렸다가 이전 태스크를 사용 중지합니다. `maximumPercent` 값을 100%로 설정하면 Amazon ECS에서는 태스크를 먼저 중지한 다음에 바꿉니다.

독립 실행형 작업 사용 중지의 경우 AWS에서 작업 사용 중지 날짜 또는 이후에 작업을 중지합니다. Amazon ECS는 태스크가 중지될 때 대체 태스크를 시작하지 않습니다. 이러한 작업을 계속 실행해야 하는 경우 알림에 표시된 시간 이전에 실행 중인 작업을 중지하고 대체 작업을 시작해야 합니다. 따라서 고객은 독립 실행형 작업의 상태를 모니터링하고 필요한 경우 중지된 작업을 대체하는 로직을 구현하는 것이 좋습니다.

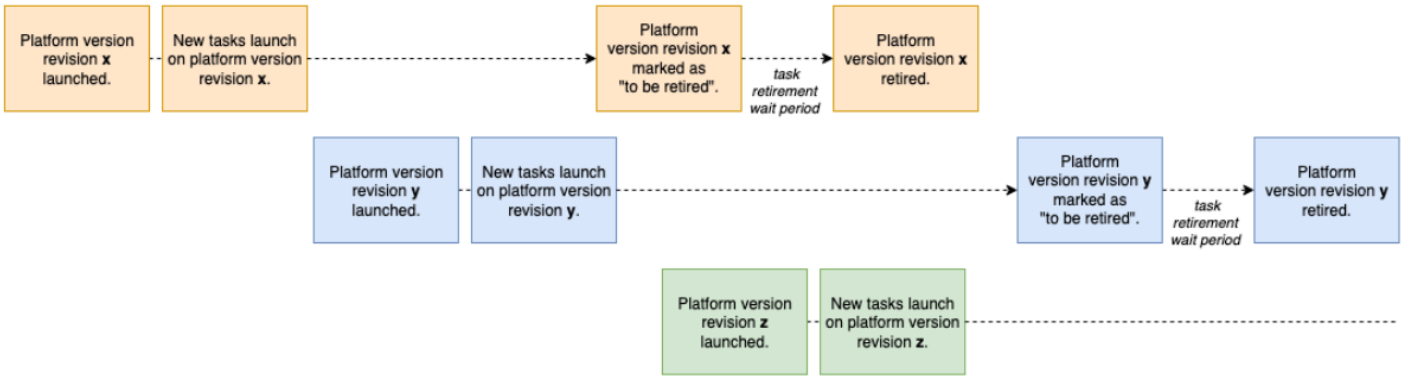
어떤 시나리오에서든 작업이 중지되면 `describe-tasks`를 실행할 수 있습니다. 응답의 `stoppedReason`은 `ECS is performing maintenance on the underlying infrastructure hosting the task`입니다.

새 플랫폼 버전 개정을 새 개정으로 교체해야 하는 경우 작업 유지 관리가 적용됩니다. 기본 Fargate 호스트에 문제가 있는 경우 Amazon ECS는 작업 사용 중지 통지 없이 호스트를 교체합니다.

태스크 사용 중지 공지 개요

AWS에서 플랫폼 버전 개정을 사용 중지해야 한다고 표시하면 모든 리전에서 해당 플랫폼 버전 개정에서 실행 중인 모든 태스크를 식별합니다. 그런 다음 각 리전별 계정당 하나의 알림을 보내 영향을 받는 태스크나 서비스와 사용 중지가 시작되는 날짜를 강조 표시합니다.

다음 그림은 새로운 개정 출시부터 플랫폼 개정 사용 중지까지 Fargate 플랫폼 버전 개정의 수명 주기를 보여줍니다.



다음 정보는 세부 사항을 제공합니다.

- 새 플랫폼 버전 개정이 시작되면 모든 새 태스크는 이 개정에 따라 예약됩니다.
- 예약되어 실행 중인 기존 태스크는 태스크 기간 동안 원래 적용되었던 개정에 그대로 유지되며 새 개정으로 마이그레이션되지 않습니다.

- 예를 들어 서비스 업데이트 또는 Fargate 태스크 사용 중지와 같은 새로운 태스크는 출시 시점에 사용 가능한 최신 플랫폼 버전 개정에 따라 배치됩니다.

태스크 사용 중지 알림은 AWS Health 대시보드 및 등록된 이메일 주소로의 이메일을 통해 전송되며, 여기에는 다음 정보가 포함됩니다.

- 작업 사용 중지 날짜 - 이 날짜 또는 이후에 작업이 중지됩니다.
- 독립 실행형 작업의 경우 작업의 ID
- 서비스 작업의 경우 서비스가 실행되는 클러스터의 ID와 서비스의 ID
- 수행해야 하는 다음 단계

일반적으로 각 AWS 리전에서 서비스 및 독립 실행형 작업에 대해 각각 하나의 알림을 보냅니다. 하지만 경우에 따라 각 작업 유형에 대해 두 개 이상의 이벤트를 수신할 수 있습니다. 예를 들어 사용 중지할 작업이 너무 많아 알림 메커니즘의 한도를 초과하는 경우가 이에 해당합니다.

사용 중지가 예약된 작업은 다음과 같은 방법으로 식별할 수 있습니다.

- AWS Health Dashboard은

  AWS Health 알림은 Amazon Storage Service와 같은 아카이브 스토리지에 대한 Amazon EventBridge, AWS Lambda 함수 실행과 같은 자동화된 작업 수행 또는 Amazon Simple Notification Service와 같은 기타 알림 시스템을 통해 전송될 수 있습니다. 자세한 내용을 알아보려면 Amazon EventBridge를 사용하여 AWS Health 이벤트 모니터링을 참조하세요. Amazon Chime, Slack 또는 Microsoft Teams에 알림을 보내기 위한 샘플 구성은 GitHub의 AWS Health Aware 리포지토리를 참조하세요.

  다음은 샘플 EventBridge 이벤트입니다.

```
{
    "version": "0",
    "id": "3c268027-f43c-0171-7425-1d799EXAMPLE",
    "detail-type": "AWS Health Event",
    "source": "aws.health",
    "account": "123456789012",
    "time": "2023-08-16T23:18:51Z",
    "region": "us-east-1",
    "resources": [
        "cluster|service",
        "cluster|service"
```

```
    ],
    "detail": {
        "eventArn": "arn:aws:health:us-east-1::event/ECS/
AWS_ECS_TASK_PATCHING_RETIREMENT/AWS_ECS_TASK_PATCHING_RETIREMENT_test1",
        "service": "ECS",
        "eventScopeCode": "ACCOUNT_SPECIFIC",
        "communicationId":
 "7988399e2e6fb0b905ddc88e0e2de1fd17e4c9fa60349577446d95a18EXAMPLE",
        "lastUpdatedTime": "Wed, 16 Aug 2023 23:18:52 GMT",
        "eventRegion": "us-east-1",
        "eventTypeCode": "AWS_ECS_TASK_PATCHING_RETIREMENT",
        "eventTypeCategory": "scheduledChange",
        "startTime": "Wed, 16 Aug 2023 23:18:51 GMT",
        "endTime": "Fri, 18 Aug 2023 23:18:51 GMT",
        "eventDescription": [
            {
                "language": "en_US",
                "latestDescription": "\\nA software update has been deployed to
 Fargate which includes CVE patches or other critical patches. No action is required
 on your part. All new tasks launched automatically uses the latest software
 version. For existing tasks, your tasks need to be restarted in order for these
 updates to apply. Your tasks running as part of the following ECS Services will
 be automatically updated beginning Wed, 16 Aug 2023 23:18:51 GMT.\\n\\nAfter Wed,
 16 Aug 2023 23:18:51 GMT, the ECS scheduler will gradually replace these tasks,
 respecting the deployment settings for your service. Typically, services should
 see little to no interruption during the update and no action is required. When AWS
 stops tasks, AWS uses the minimum healthy percent (1) and launches a new task in
 an attempt to maintain the desired count for the service. By default, the minimum
 healthy percent of a service is 100 percent, so a new task is started first before
 a task is stopped. Service tasks are routinely replaced in the same way when
 you scale the service or deploy configuration changes or deploy task definition
 revisions. If you would like to control the timing of this restart you can update
 the service before Wed, 16 Aug 2023 23:18:51 GMT, by running the update-service
 command from the ECS command-line interface specifying force-new-deployment for
 services using Rolling update deployment type. For example:\\n\\n$ aws ecs update-
service -service service_name \\\n--cluster cluster_name -force-new-deployment\
\n\\nFor services using Blue/Green deployment type with AWS CodeDeploy:\\nPlease
 refer to create-deployment document (2) and create new deployment using same task
 definition revision.\\n\\nFor further details on ECS deployment types, please
 refer to ECS Deployment Developer Guide (1).\\nFor further details on Fargate's
 update process, please refer to the AWS Fargate User Guide (3).\\nIf you have
 any questions or concerns, please contact AWS Support (4).\\n\\n(1) https://
docs.aws.amazon.com/AmazonECS/latest/developerguide/deployment-types.html\\n(2)
 https://docs.aws.amazon.com/cli/latest/reference/deploy/create-deployment.html\\n(3)
```

```
 https://docs.aws.amazon.com/AmazonECS/latest/userguide/task-maintenance.html\\n(4)
 https://aws.amazon.com/support\\n\\nA list of your affected resources(s) can be
 found in the 'Affected resources' tab in the 'Cluster/ Service' format in the AWS
 Health Dashboard. \\n\\n"
            }
        ],
      "affectedEntities": [
                {
                    "entityValue": "arn:aws:ecs:eu-west-1:111222333444:task/
examplecluster/00805ce1d81940b5a37398e5a2c23333"
                },
                {
                    "entityValue": "arn:aws:ecs:eu-west-1:111222333444:task/
examplecluster/00805ce1d81940b5a37398e5a2c25555"
                }
            }
        ]
    }
}
```

- 이메일

  AWS 계정 ID에 대해 등록된 이메일로 이메일이 전송됩니다.

태스크 사용 중지를 준비하는 방법에 대한 자세한 내용은 Amazon ECS에서 AWS Fargate 태스크 사용 중지 준비 섹션을 참조하세요.

태스크 사용 중지를 옵트아웃할 수 있나요?

아니요. AWS 공동 책임 모델의 일부로, AWS는 AWS Fargate의 기본 인프라를 관리하고 유지 보수할 책임이 있습니다. 여기에는 보안과 안정성을 보장하기 위한 주기적인 플랫폼 업데이트 수행이 포함됩니다. 이러한 업데이트는 AWS에 의해 자동으로 적용되며 고객이 옵트아웃할 수 없습니다. 이는 EC2 인스턴스에서 워크로드를 실행할 때와 비교하여 AWS에서 기본 플랫폼 유지 관리를 처리한다는 점에서 AWS Fargate를 사용할 때의 주요 이점입니다. 이 모델을 사용하면 인프라 유지 관리가 아닌 애플리케이션에 집중할 수 있습니다. AWS는 이러한 플랫폼 업데이트를 자동으로 적용함으로써 고객이 별도의 조치를 취하지 않아도 Fargate 환경을 최신 상태로 안전하게 유지할 수 있습니다. 따라서 Fargate에서 워크로드를 실행하기 위한 안정적이고 안전한 컨테이너화된 환경을 제공하는 데 도움이 됩니다.

다른 AWS 서비스를 통해 작업 사용 중지 알림을 받을 수 있나요?

AWS에서는 AWS Health Dashboard 및 AWS 계정의 기본 이메일 연락처로 작업 사용 중지 알림을 보냅니다. AWS Health Dashboard에서는 EventBridge를 비롯한 다른 AWS 서비스와의 다양한 통합을 제공합니다. EventBridge를 사용하여 통지 가시성을 자동화할 수 있습니다(예: 메시지를 ChatOps 도구로 전달). 자세한 내용은 Solution overview: Capturing task retirement notifications를 참조하세요.

작업 사용 중지가 예약된 이후에 작업 사용 중지를 변경할 수 있나요?

아니요. 일정은 작업 사용 중지 대기 시간(기본 7일)을 기준으로 합니다. 시간이 더 필요한 경우 대기 시간을 14일로 구성할 수 있습니다. 자세한 내용은 2단계: 태스크 사용 중지 알림을 캡처하여 팀에 알리고 조치 취하기 섹션을 참조하세요. 이 구성의 변경 내용은 향후 예약된 사용 중지에 적용됩니다. 현재 예약된 사용 중지는 영향을 받지 않습니다. 추가 문의 사항이 있는 경우 지원에 문의하세요.

Amazon ECS는 서비스의 일부인 작업을 어떻게 처리하나요?

서비스 태스크의 경우 AWS 전에 먼저 이러한 태스크를 대체하지 않는 한, 태스크 사용 중지에 대응하여 어떤 조치도 취할 필요가 없습니다. Amazon ECS 스케줄러에서 작업을 중지하는 경우 최소 정상 상태 백분율을 사용하며, 원하는 서비스 개수를 유지하기 위해 새 작업을 시작합니다. Fargate 사용 중지의 영향을 최소화하려면 Amazon ECS 모범 사례에 따라 워크로드를 배포해야 합니다. 예를 들어, 상태 비저장 애플리케이션을 웹 또는 API 서버와 같은 Amazon ECS 서비스로 배포하는 경우, 고객은 여러 개의 태스크 복제본을 배포하고 minimumHealthyPercent를 100%로 설정해야 합니다. 기본적으로 서비스의 최소 상태 백분율은 100%입니다. 따라서 Fargate가 태스크 사용 중지를 시작하면 Amazon ECS는 먼저 새 태스크를 예약한 다음 해당 태스크가 실행될 때까지 기다렸다가 이전 태스크를 사용 중지합니다. 태스크 사용 중지의 일환으로 서비스 태스크는 서비스 규모를 조정하거나 구성 변경 내용을 배포하거나 태스크 정의 개정을 배포할 때 동일한 방식으로 정기적으로 대체됩니다. 태스크 사용 중지 프로세스를 준비하려면 Amazon ECS에서 AWS Fargate 태스크 사용 중지 준비 섹션을 참조하세요.

Amazon ECS에서 독립 실행형 작업을 자동으로 처리할 수 있나요?

아니요. AWS에서는 `RunTask`, 예약된 작업(예: EventBridge 스케줄러를 통해), AWS Batch 또는 AWS Step Functions에 의해 시작된 독립 실행형 작업에 대한 대체 작업을 생성할 수 없습니다. Amazon ECS는 서비스의 일부인 작업만 관리합니다.

Amazon ECS에서 AWS Fargate 태스크 사용 중지 준비

태스크 사용 중지를 준비하려면 다음 작업을 수행하세요.

1. 태스크 사용 중지 대기 기간을 설정합니다.

2. 태스크 사용 중지 알림을 캡처하여 팀원에게 알릴 수 있습니다.
3. 태스크 사용 중지 타이밍을 정확히 제어할 수는 없지만 force-deployment 옵션으로 서비스를 업데이트하여 태스크 대체를 제어할 수 있습니다.

1단계: 태스크 사용 중지 대기 시간 설정

Fargate에서 작업 사용 중지를 시작하는 시간을 구성할 수 있습니다. 업데이트를 즉시 적용해야 하는 워크로드의 경우 즉시 설정(0)을 선택합니다. 더 세부적으로 제어해야 하는 경우, 예를 들어 특정 기간에만 작업을 중지할 수 있는 경우 7일(7) 또는 14일(14) 옵션을 구성합니다.

최신 플랫폼 수정 버전을 더 빨리 받으려면 짧은 대기 기간을 선택하는 것이 좋습니다.

`put-account-setting-default` 또는 `put-account-setting`을 루트 사용자 또는 관리 사용자로 실행하여 대기 기간을 구성합니다. `name`에는 `fargateTaskRetirementWaitPeriod` 옵션을 사용하고 `value` 옵션을 다음 값 중 하나로 설정합니다.

- `0` - AWS는 알림을 보내고 영향을 받은 작업을 즉시 사용 중지합니다.
- 7 - AWS는 알림을 보내고 7일 기다린 후 영향을 받은 작업을 사용 중지합니다.
- 14 - AWS는 알림을 보내고 14일 기다린 후 영향을 받은 작업을 사용 중지합니다.

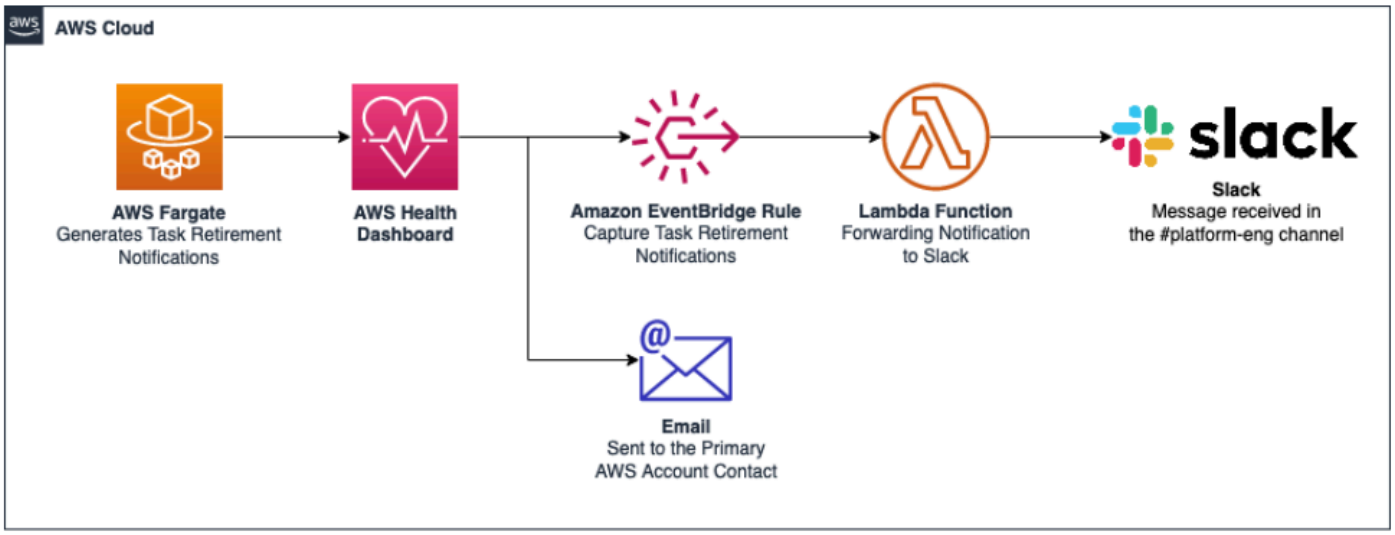
기본값은 7일입니다.

자세한 정보는 Amazon Elastic Container Service API Reference의 put-account-setting-default 및 put-account-setting을 참조하세요.

2단계: 태스크 사용 중지 알림을 캡처하여 팀에 알리고 조치 취하기

태스크 사용 중지가 예정되어 있으면 AWS는 AWS Health Dashboard와 AWS 계정의 기본 이메일 연락처로 태스크 사용 중지 알림을 보냅니다. AWS Health Dashboard에서는 Amazon EventBridge를 비롯한 다른 AWS 서비스와의 다양한 통합을 제공합니다. EventBridge를 사용하여 태스크 사용 중지 알림에서 자동화를 구축할 수 있습니다. 예를 들어, 메시지를 ChatOps 도구로 전달하여 예정된 사용 중지의 가시성을 높입니다. AWS Health 인식은 AWS Health Dashboard의 기능과 조직 전체에 알림을 배포할 수 있는 방법을 보여주는 리소스입니다. 태스크 사용 중지 알림을 Slack과 같은 채팅 애플리케이션으로 전달할 수 있습니다.

다음 그림은 솔루션 개요를 보여줍니다.



다음 정보는 세부 사항을 제공합니다.

- Fargate는 AWS Health Dashboard에 태스크 사용 중지 알림을 보냅니다.
- AWS Health Dashboard는 AWS 계정의 기본 이메일 연락처로 메일을 전송하고 EventBridge에 알립니다.
- EventBridge에는 사용 중지 알림을 캡처하는 규칙이 있습니다.

  이벤트 세부 정보 유형: `"AWS Health Event"` and the Event Detail Type Code: `"AWS_ECS_TASK_PATCHING_RETIREMENT"`의 이벤트를 찾는 규칙
- 규칙은 Slack 수신 Webhook를 사용하여 정보를 Slack으로 전달하는 Lambda 함수를 트리거합니다. 자세한 내용은 수신 웹후크를 참조하세요.

코드 예제는 Github의 Capturing AWS Fargate Task Retirement Notifications를 참조하세요.

3단계: 태스크 대체 제어

태스크 사용 중지의 정확한 타이밍을 제어할 수는 없지만 대기 시간을 정의할 수는 있습니다. 원하는 일정에 따라 태스크를 대체하는 것을 제어하려면 태스크 사용 중지 공지를 캡처하여 태스크 사용 중지 날짜를 먼저 파악하면 됩니다. 그런 다음 서비스를 다시 배포하여 대체 태스크를 시작하고 독립 실행형 태스크도 마찬가지로 대체할 수 있습니다. 롤링 배포를 사용하는 서비스의 경우 사용 중지 시작 시간 전에 `force-deployment` 옵션과 함께 `update-service`를 사용하여 서비스를 업데이트합니다.

다음 `update-service` 예제에서는 `force-deployment` 옵션을 사용합니다.

```
aws ecs update-service —-service service_name \
    --cluster cluster_name \
     --force-new-deployment
```

블루/그린 배포를 사용하는 서비스의 경우 AWS CodeDeploy에서 새 배포를 생성해야 합니다. 배포를 생성하는 방법에 대한 자세한 내용은 AWS Command Line Interface 참조의 create-deployment를 참조하세요.

AWS Fargate의 Amazon ECS에 대해 지원되는 리전

다음 테이블을 사용하여 AWS Fargate의 Linux 컨테이너 및 AWS Fargate의 Windows 컨테이너에 대한 리전 지원을 확인할 수 있습니다.

AWS Fargate의 Linux 컨테이너

AWS Fargate의 Amazon ECS Linux 컨테이너는 다음 AWS 리전에서 지원됩니다. 지원되는 가용 영역 ID는 해당되는 경우 명시되어 있습니다.

리전 이름	리전
미국 동부(오하이오)	us-east-2
미국 동부(버지니아 북부)	us-east-1
미국 서부(캘리포니아 북부)	us-west-1(usw1-az1 및 usw1-az3만 해당)
미국 서부(오레곤)	us-west-2
캐나다(중부)	ca-central-1
캐나다 서부(캘거리)	ca-west-1
멕시코(중부)	mx-central-1
아프리카(케이프타운)	af-south-1
아시아 태평양(홍콩)	ap-east-1
아시아 태평양(뭄바이)	ap-south-1

리전 이름	리전
아시아 태평양(도쿄)	ap-northeast-1(apne1-az1 , apne1-az2 , apne1-az4 만 해당)
아시아 태평양(서울)	ap-northeast-2
아시아 태평양(오사카)	ap-northeast-3
아시아 태평양(하이데라바드)	ap-south-2
아시아 태평양(싱가포르)	ap-southeast-1
아시아 태평양(시드니)	ap-southeast-2
아시아 태평양(태국)	ap-southeast-7
아시아 태평양(자카르타)	ap-southeast-3
아시아 태평양(멜버른)	ap-southeast-4
아시아 태평양(말레이시아)	ap-southeast-5
캐나다(중부)	ca-central-1
캐나다 서부(캘거리)	ca-west-1
중국(베이징)	cn-north-1(cnn1-az1 및 cnn1-az2만 해당)
중국(닝샤)	cn-northwest-1
유럽(프랑크푸르트)	eu-central-1
유럽(취리히)	eu-central-2
유럽(아일랜드)	eu-west-1
유럽(런던)	eu-west-2
유럽(파리)	eu-west-3
유럽(밀라노)	eu-south-1

리전 이름	리전
유럽(스페인)	eu-south-2
유럽(스톡홀름)	eu-north-1
남아메리카(상파울루)	sa-east-1
이스라엘(텔아비브)	il-central-1
중동(바레인)	me-south-1
중동(UAE)	me-central-1
AWS GovCloud(미국 동부)	us-gov-east-1
AWS GovCloud(미국 서부)	us-gov-west-1

AWS Fargate의 Windows 컨테이너

AWS Fargate의 Amazon ECS Windows 컨테이너는 다음 AWS 리전에서 지원됩니다. 지원되는 가용 영역 ID는 해당되는 경우 명시되어 있습니다.

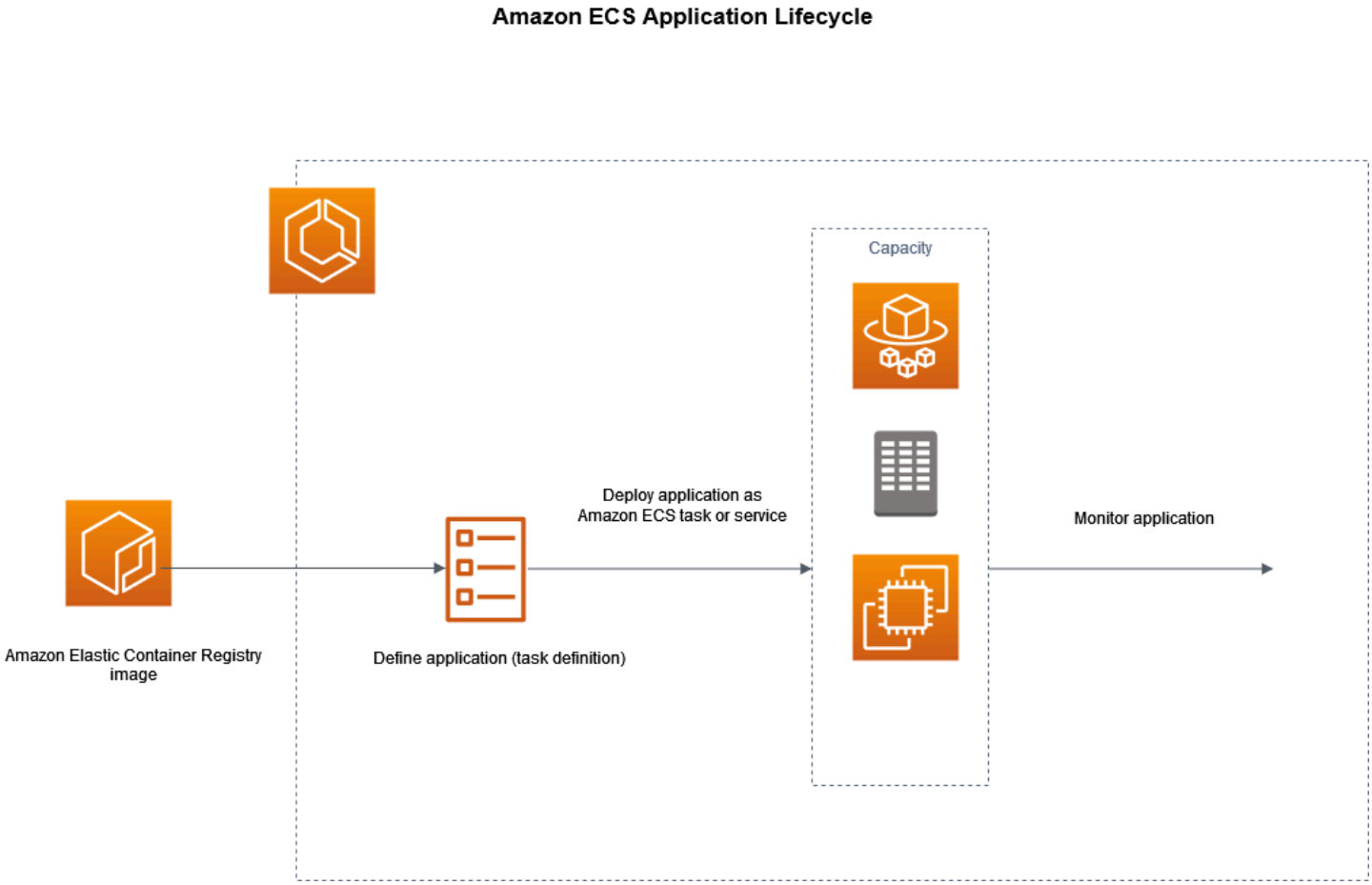
리전 이름	리전
미국 동부(오하이오)	us-east-2
미국 동부(버지니아 북부)	us-east-1(`use1-az1`, `use1-az2`, `use1-az4`, `use1-az5` 및 `use1-az6`만 해당)
미국 서부(캘리포니아 북부)	us-west-1(`usw1-az1` 및 `usw1-az3`만 해당)
미국 서부(오레곤)	us-west-2
아프리카(케이프타운)	af-south-1
아시아 태평양(홍콩)	ap-east-1
아시아 태평양(뭄바이)	ap-south-1

리전 이름	리전
아시아 태평양(하이데라바드)	ap-south-2
아시아 태평양(오사카)	ap-northeast-3
아시아 태평양(서울)	ap-northeast-2
아시아 태평양(싱가포르)	ap-southeast-1
아시아 태평양(시드니)	ap-southeast-2
아시아 태평양(멜버른)	ap-southeast-4
아시아 태평양(말레이시아)	ap-southeast-5
아시아 태평양(도쿄)	ap-northeast-1(apne1-az1 , apne1-az2 , apne1-az4 만 해당)
캐나다(중부)	ca-central-1(cac1-az1 및 cac1-az2만 해당)
캐나다 서부(캘거리)	ca-west-1
중국(베이징)	cn-north-1(cnn1-az1 및 cnn1-az2만 해당)
중국(닝샤)	cn-northwest-1
유럽(프랑크푸르트)	eu-central-1
유럽(취리히)	eu-central-2
유럽(아일랜드)	eu-west-1
유럽(런던)	eu-west-2
유럽(파리)	eu-west-3
유럽(밀라노)	eu-south-1
유럽(스페인)	eu-south-2
유럽(스톡홀름)	eu-north-1

리전 이름	리전
남아메리카(상파울루)	sa-east-1
이스라엘(텔아비브)	il-central-1
중동(UAE)	me-central-1
중동(바레인)	me-south-1

Amazon ECS를 위한 솔루션 설계

다음 다이어그램은 애플리케이션 수명 주기와 Amazon ECS 구성 요소와의 작동 방식을 보여줍니다.



컨테이너에서 실행할 수 있도록 애플리케이션을 설계해야 합니다. 컨테이너는 소프트웨어 애플리케이션을 실행하는 데 필요한 모든 것을 포함하는 소프트웨어 개발의 표준화된 단위입니다. 여기에는 관련 코드, 런타임, 시스템 도구 및 시스템 라이브러리가 포함됩니다. 컨테이너는 이미지라고 하는 읽기 전용 템플릿에서 생성됩니다. 이미지는 일반적으로 Dockerfile에서 구축됩니다. Dockerfile은 컨테이너 빌드 지침을 포함하는 일반 텍스트 파일입니다. 이러한 이미지는 빌드된 후 다운로드할 수 있는 Amazon ECR과 같은 레지스트리에 저장됩니다.

이미지를 생성하고 저장한 후 Amazon ECS 작업 정의를 생성합니다. 작업 정의는 애플리케이션에 대한 청사진과 같습니다. 작업 정의는 애플리케이션을 구성하는 파라미터 및 하나 이상의 컨테이너를 설명하는 JSON 형식의 텍스트 파일입니다. 예를 들어, 이를 사용하여 운영 체제에 대한 이미지 및 파라미터, 사용할 컨테이너, 애플리케이션에 대해 개방할 포트, 작업의 컨테이너와 함께 사용할 데이터 볼륨을 지정할 수 있습니다. 태스크 정의에 사용할 수 있는 특정 파라미터는 특정 애플리케이션의 필요에 따라 달라집니다.

작업 정의를 정의한 후에는 클러스터에 서비스 또는 작업으로 배포합니다. 클러스터는 클러스터에 등록된 용량 인프라에서 실행되는 작업 또는 서비스의 논리적 그룹입니다.

태스크는 클러스터 내 태스크 정의를 인스턴스화하는 것입니다. 독립 실행형 태스크를 실행하거나 서비스의 일부로 태스크를 실행할 수 있습니다. Amazon ECS 서비스를 사용하여 Amazon ECS 클러스터에서 원하는 수의 태스크를 동시에 실행하고 유지할 수 있습니다. 태스크가 어떤 이유로든 실패하거나 중지하면 Amazon ECS 서비스 스케줄러가 태스크 정의에 따라 다른 인스턴스를 시작하는 방식으로 작동합니다. 이로써 이를 대체하여 서비스에서 원하는 수의 태스크를 유지할 수 있습니다.

컨테이너 에이전트는 Amazon ECS 클러스터 내의 각 컨테이너 인스턴스에서 실행됩니다. 에이전트는 현재 실행 중인 태스크와 컨테이너의 리소스 사용률에 대한 정보를 Amazon ECS로 전송합니다. Amazon ECS로부터 요청을 수신할 때마다 태스크를 시작 및 중지합니다.

작업 또는 서비스를 배포한 후 다음 도구 중 하나를 사용하여 배포 및 애플리케이션을 모니터링할 수 있습니다.

- CloudWatch
- 런타임 모니터링

Capacity

용량은 컨테이너가 실행되는 인프라입니다. 다음과 같은 옵션이 있습니다.

- Amazon EC2 인스턴스
- 서버리스(AWS Fargate)
- 온프레미스 가상 머신(VM) 또는 서버

클러스터를 생성할 때 인프라를 지정합니다. 또한 작업 정의를 등록할 때도 인프라 유형을 지정합니다. 작업 정의에서는 인프라를 '시작 유형'이라고 합니다. 독립 실행형 작업을 실행하거나 서비스를 배포할 때도 시작 유형을 사용합니다. 시작 유형 옵션에 대한 자세한 내용은 Amazon ECS 시작 유형 섹션을 참조하세요.

서비스 엔드포인트

서비스 엔드포인트는 인터넷 프로토콜 버전 4(IPv4) 또는 인터넷 프로토콜 버전 6(IPv6)을 사용하여 프로그래밍 방식으로 서비스에 연결하는 데 사용하는 Amazon ECS의 진입점 URL입니다. 기본적으로 Amazon ECS에 프로그래밍 방식으로 연결하기 위해 실행하는 요청은 IPv4 요청만 지원하는 서비스

엔드포인트를 사용합니다. IPv4 또는 IPv6 요청을 사용하여 Amazon ECS와 프로그래밍 방식으로 연결하려면 듀얼 스택 엔드포인트를 사용할 수 있습니다. 듀얼 스택 엔드포인트에 대한 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

네트워킹

AWS 리소스는 서브넷에서 생성됩니다. EC2 인스턴스를 사용하는 경우 Amazon ECS는 클러스터를 생성할 때 지정한 서브넷에서 인스턴스를 시작합니다. 작업은 인스턴스 서브넷에서 실행됩니다. Fargate 또는 온프레미스 가상 머신의 경우 작업을 실행하거나 서비스를 생성할 때 서브넷을 지정합니다.

애플리케이션에 따라 서브넷은 프라이빗 또는 퍼블릭 서브넷일 수 있으며 서브넷은 다음 AWS 리소스 중 하나에 속할 수 있습니다.

- 가용 영역
- 로컬 영역
- Wavelength Zone
- AWS 리전
- AWS Outposts

자세한 내용은 공유 서브넷, 로컬 영역 및 Wavelength Zone의 Amazon ECS 애플리케이션 또는 AWS Outposts의 Amazon Elastic Container Service을 참조하세요.

다음 방법 중 하나를 사용하여 애플리케이션을 인터넷에 연결할 수 있습니다.

- 인터넷 게이트웨이를 사용하는 퍼블릭 서브넷

  대용량의 대역폭이나 최소 지연 시간이 필요한 퍼블릭 애플리케이션이 있는 경우 퍼블릭 서브넷을 사용합니다. 적용 가능한 시나리오로, 비디오 스트리밍 및 게임 서비스가 포함됩니다.
- NAT 게이트웨이를 사용하는 퍼블릭 서브넷

  컨테이너를 외부 직접 액세스로부터 보호하려면 프라이빗 서브넷을 사용합니다. 적용 가능한 시나리오로, 사용자 데이터와 암호를 저장하는 결제 처리 시스템 또는 컨테이너가 포함됩니다.
- AWS PrivateLink

  AWS PrivateLink를 사용하면 트래픽을 퍼블릭 인터넷에 노출하지 않고 VPC, AWS 서비스, 온프레미스 네트워크 간에서 프라이빗 연결을 확보할 수 있습니다.

기능 액세스

Amazon ECS 계정 설정을 사용하여 다음 기능에 액세스할 수 있습니다.

- Container Insights

  CloudWatch Container Insights는 컨테이너 애플리케이션 및 마이크로서비스의 지표 및 로그를 수집하고, 종합하며, 요약합니다. 이 지표에는 CPU, 메모리, 디스크, 네트워크 같은 리소스 사용률이 포함되어 있습니다.

- `awsvpc` 트렁킹

  특정 EC2 인스턴스 유형의 경우 새로 시작한 컨테이너 인스턴스에서 추가 네트워크 인터페이스(ENI)를 사용할 수 있습니다.

- 태그 지정 권한 부여

  사용자는 리소스를 생성하는 작업을 위한 권한(예: `ecsCreateCluster`)이 있어야 합니다. 리소스 생성 작업에서 태그가 지정되면 AWS은 `ecs:TagResource` 작업에서 추가 권한 부여를 수행해 사용자 또는 역할에 태그를 생성할 권한이 있는지 확인합니다.

- Fargate FIPS-140 규정 준수

  Fargate에서는 민감한 정보를 보호하는 암호화 모듈의 보안 요구 사항을 규정하는 Federal Information Processing Standard(FIPS-140)를 지원합니다. 이는 현재 미국 및 캐나다 정부 표준이며 연방 정보 보안 관리법(FISMA) 또는 연방 위험 및 권한 부여 관리 프로그램(FedRAMP)을 준수해야 하는 시스템에 적용됩니다.

- Fargate 작업 사용 중지 시간 변경

  패치 적용을 위해 Fargate 작업을 사용 중지하기 전 대기 기간을 구성할 수 있습니다.

- 듀얼 스택 VPC

  작업에서 IPv4, IPv6 또는 모두를 통해 통신할 수 있도록 허용합니다.

- Amazon 리소스 이름(ARN) 형식

  태그 지정 권한 부여와 같은 특정 기능을 사용하려면 새로운 Amazon 리소스 이름(ARN) 형식이 필요합니다.

자세한 내용은 계정 설정을 사용하여 Amazon ECS 기능에 액세스 섹션을 참조하세요.

IAM 역할

IAM 역할은 계정에 생성할 수 있는, 특정 권한을 지닌 IAM 자격 증명입니다. Amazon ECS에서는 역할을 생성하여 컨테이너 또는 서비스와 같은 Amazon ECS 리소스에 권한을 부여할 수 있습니다.

일부 Amazon ECS 기능에는 역할이 필요합니다. 자세한 내용은 Amazon ECS에 대한 IAM 역할 섹션을 참조하세요.

로깅

로깅 및 모니터링은 Amazon ECS 워크로드의 안정성, 가용성 및 성능을 유지 관리하는 데 중요한 부분입니다. 다음과 같은 옵션을 사용할 수 있습니다.

- Amazon CloudWatch 로그 - Amazon CloudWatch로 로그를 라우팅합니다.
- FireLens for Amazon ECS - 로그 저장 및 분석을 위해 AWS 서비스 또는 AWS Partner Network 대상으로 로그를 라우팅합니다. AWS Partner Network는 프로그램, 전문 지식 및 리소스를 활용하여 고객 제품을 구축, 마케팅 및 판매하는 글로벌 파트너 커뮤니티입니다.

Amazon ECS 시작 유형

작업 정의 시작 유형은 작업을 실행할 수 있는 용량을 정의합니다(예: AWS Fargate).

시작 유형을 선택한 후 Amazon ECS는 사용자가 구성한 작업 정의 파라미터가 시작 유형과 호환되는지 확인합니다.

다음 테이블에서는 시작 유형을 설명합니다.

시작 유형		자세히 알아보기
Fargate	Fargate는 서버를 관리하지 않고 애플리케이션 구축에만 집중할 수 있게 해주는 종량제 서버리스 컴퓨팅 엔진입니다.	Amazon ECS의 Fargate 시작 유형
EC2	인스턴스 유형, 인스턴스 수를 선택하고 용량을 관리합니다.	Amazon ECS의 EC2 시작 유형
온프레미스 가상 머신(VM) 또는 서버	Amazon ECS Anywhere는 외부 인스턴스(예: 온프레미스 서	Amazon ECS의 외부(Amazon ECS Anywhere) 시작 유형

시작 유형		자세히 알아보기
	버 또는 가상 머신(VM))을 클러스터에 등록하도록 지원합니다.	

Amazon ECS의 Fargate 시작 유형

Fargate는 서버를 관리하지 않고 애플리케이션 구축에만 집중할 수 있게 해주는 종량제 서버리스 컴퓨팅 엔진입니다. Fargate를 선택하면 EC2 인프라를 관리할 필요가 없습니다. 컨테이너 이미지를 구축하고 애플리케이션을 실행할 클러스터를 정의하기만 하면 됩니다. Fargate는 다음을 포함한 AWS 서비스와 기본적으로 통합되어 있습니다.

- Amazon VPC
- 오토 스케일링
- Elastic Load Balancing
- IAM
- Secrets Manager

Fargate는 애플리케이션에 필요한 정확한 CPU와 메모리를 선택하므로 EC2보다 더 효과적으로 제어할 수 있습니다. Fargate에서 용량 스케일 아웃을 처리하므로 트래픽 급증에 대해 걱정할 필요가 없습니다. 즉, Fargate를 사용하면 운영 부담이 줄어듭니다.

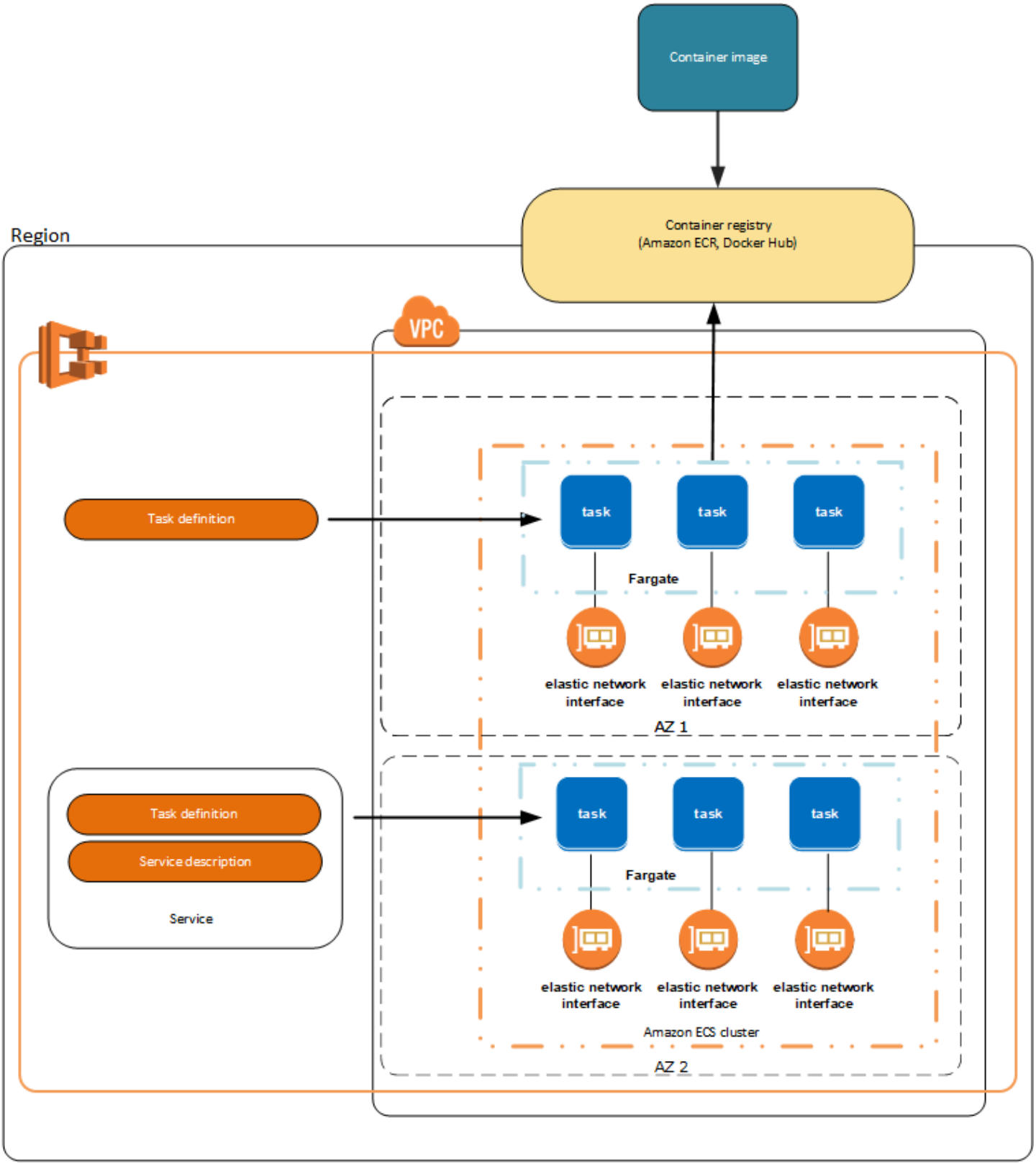
Fargate는 PCI, FIPS 140-3, FedRAMP 및 HIPAA를 포함한 규정 준수 프로그램의 표준을 충족합니다. 자세한 내용은 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하세요.

Fargate는 다음 워크로드에 적합합니다.

- 낮은 운영 오버헤드가 필요한 대규모 워크로드
- 가끔 버스트가 발생하는 소규모 워크로드
- 작은 워크로드
- 배치 워크로드

Fargate가 지원되는 리전에 대한 자세한 정보는 the section called “AWS Fargate 리전” 섹션을 참조하세요.

다음의 다이어그램은 일반 아키텍처를 나타냅니다.



Fargate의 Amazon ECS에 대한 자세한 정보는 Amazon ECS용 AWS Fargate를 참조하세요.

Amazon ECS에 대한 Fargate 컨테이너 이미지 가져오기 동작에서 Linux 컨테이너

모든 Fargate 태스크는 자체 일회용 단일 테넌트 인스턴스에서 실행됩니다. Fargate에서 Linux 컨테이너를 실행하면 컨테이너 이미지 또는 컨테이너 이미지 레이어가 인스턴스에 캐싱되지 않습니다. 따라서 태스크에 정의된 각 컨테이너 이미지에 대해 각 Fargate 태스크의 컨테이너 이미지 레지스트리에서 전체 컨테이너 이미지를 가져와야 합니다. 이미지를 가져오는 데 걸리는 시간은 Fargate 태스크를 시작하는 데 걸리는 시간과 직접적인 상관관계가 있습니다.

이미지 가져오기 시간을 최적화하려면 다음 사항을 고려하세요.

컨테이너 이미지 근접성

컨테이너 이미지를 다운로드하는 데 걸리는 시간을 줄이려면 데이터를 최대한 컴퓨팅에 가까운 곳에 두세요. 인터넷을 통해 또는 AWS 리전 간에 컨테이너 이미지를 가져오면 다운로드 시간에 영향을 미칠 수 있습니다. 태스크가 실행될 동일한 리전에 컨테이너 이미지를 저장하는 것이 좋습니다. Amazon ECR에 컨테이너 이미지를 저장하는 경우 VPC 인터페이스 엔드포인트를 사용하여 이미지 가져오기 시간을 더욱 단축하세요. 자세한 내용은 Amazon ECR 사용 설명서의 Amazon ECR 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

컨테이너 이미지 크기 줄이기

컨테이너 이미지의 크기는 다운로드 시간에 직접적으로 영향을 미칩니다. 컨테이너 이미지의 크기 또는 컨테이너 이미지 레이어 수를 줄이면 이미지를 다운로드하는 데 걸리는 시간을 줄일 수 있습니다. 가벼운 기본 이미지(예: 최소 Amazon Linux 2023 컨테이너 이미지)는 기존 운영 체제 기본 이미지에 기반을 둔 이미지보다 훨씬 작을 수 있습니다. 최소 이미지에 대한 자세한 내용은 Amazon Linux 2023 사용 설명서의 AL2023 최소 컨테이너 이미지를 참조하세요.

대체 압축 알고리즘

컨테이너 이미지 레이어는 컨테이너 이미지 레지스트리로 푸시할 때 압축되는 경우가 많습니다. 컨테이너 이미지 레이어를 압축하면 네트워크를 통해 전송되고 컨테이너 이미지 레지스트리에 저장해야 하는 데이터의 양이 줄어듭니다. 컨테이너 런타임에 의해 컨테이너 이미지 레이어가 인스턴스로 다운로드되면 해당 레이어의 압축이 해제됩니다. 사용된 압축 알고리즘과 런타임에 사용 가능한 vCPU의 양은 컨테이너 이미지 압축을 푸는 데 걸리는 시간에 영향을 미칩니다. Fargate에서는 태스크의 크기를 늘리거나 더 성능이 뛰어난 zstd 압축 알고리즘을 활용하여 압축 해제에 걸리는 시간을 줄일 수 있습니다. 자세한 내용은 GitHub의 zstd를 참조하세요. Fargate를 위한 이미지를 구현하는 방법에 대한 자세한 내용은 zstd 압축 컨테이너 이미지를 사용하여 AWS Fargate 시작 시간 단축을 참조하세요.

컨테이너 이미지 지연 로딩

큰 컨테이너 이미지(> 250mb)의 경우 모든 컨테이너 이미지를 다운로드하는 것보다 컨테이너 이미지를 지연 로드하는 것이 최적일 수 있습니다. Fargate에서는 Seekable OCI(SOCI)를 사용하여 컨테이너 이미지 레지스트리에서 컨테이너 이미지를 지연 로드할 수 있습니다. 자세한 내용은 GitHub의 soci-snapshotter와 Seekable OCI(SOCI)를 사용하여 컨테이너 이미지 지연 로딩을 참조하세요.

Amazon ECS에 대한 Fargate 컨테이너 이미지 가져오기 동작에서 Windows 컨테이너

Fargate Windows는 Microsoft에서 제공한 가장 최근 달과 이전 달의 servercore 기본 이미지를 캐싱합니다. 이 이미지는 매주 패치 화요일에 업데이트되는 KB/빌드 번호 패치와 일치합니다. 예를 들면, 2024년 4월 9일에 Microsoft에서 Windows Server 2019용 KB5036896(17763.5696)을 릴리스했습니다. 2024년 3월 12일의 이전 달 KB는 KB5035849(17763.5576)였습니다. 따라서 플랫폼 `WINDOWS_SERVER_2019_CORE` 및 `WINDOWS_SERVER_2019_FULL`의 경우 다음과 같은 컨테이너 이미지가 캐싱됩니다.

- `mcr.microsoft.com/windows/servercore:ltsc2019`
- `mcr.microsoft.com/windows/servercore:10.0.17763.5696`
- `mcr.microsoft.com/windows/servercore:10.0.17763.5576`

또한 2024년 4월 9일에 Microsoft에서 Windows Server 2022용 KB5036909(20348.2402)를 릴리스했습니다. 2024년 3월 12일의 이전 달 KB는 KB5035857(20348.2340)였습니다. 따라서 플랫폼 `WINDOWS_SERVER_2022_CORE` 및 `WINDOWS_SERVER_2022_FULL`의 경우 다음과 같은 컨테이너 이미지가 캐싱됩니다.

- `mcr.microsoft.com/windows/servercore:ltsc2022`
- `mcr.microsoft.com/windows/servercore:10.0.20348.2402`
- `mcr.microsoft.com/windows/servercore:10.0.20348.2340`

Amazon ECS의 EC2 시작 유형

EC2 시작 유형은 가격에 최적화되어야 하는 대규모 워크로드에 적합합니다.

EC2 시작 유형을 사용하여 태스크 정의 및 서비스를 모델링하는 방법을 고려할 때 어떤 프로세스를 함께 실행해야 하는지와 각 구성 요소의 규모를 어떻게 조정할지를 고려하는 것이 좋습니다.

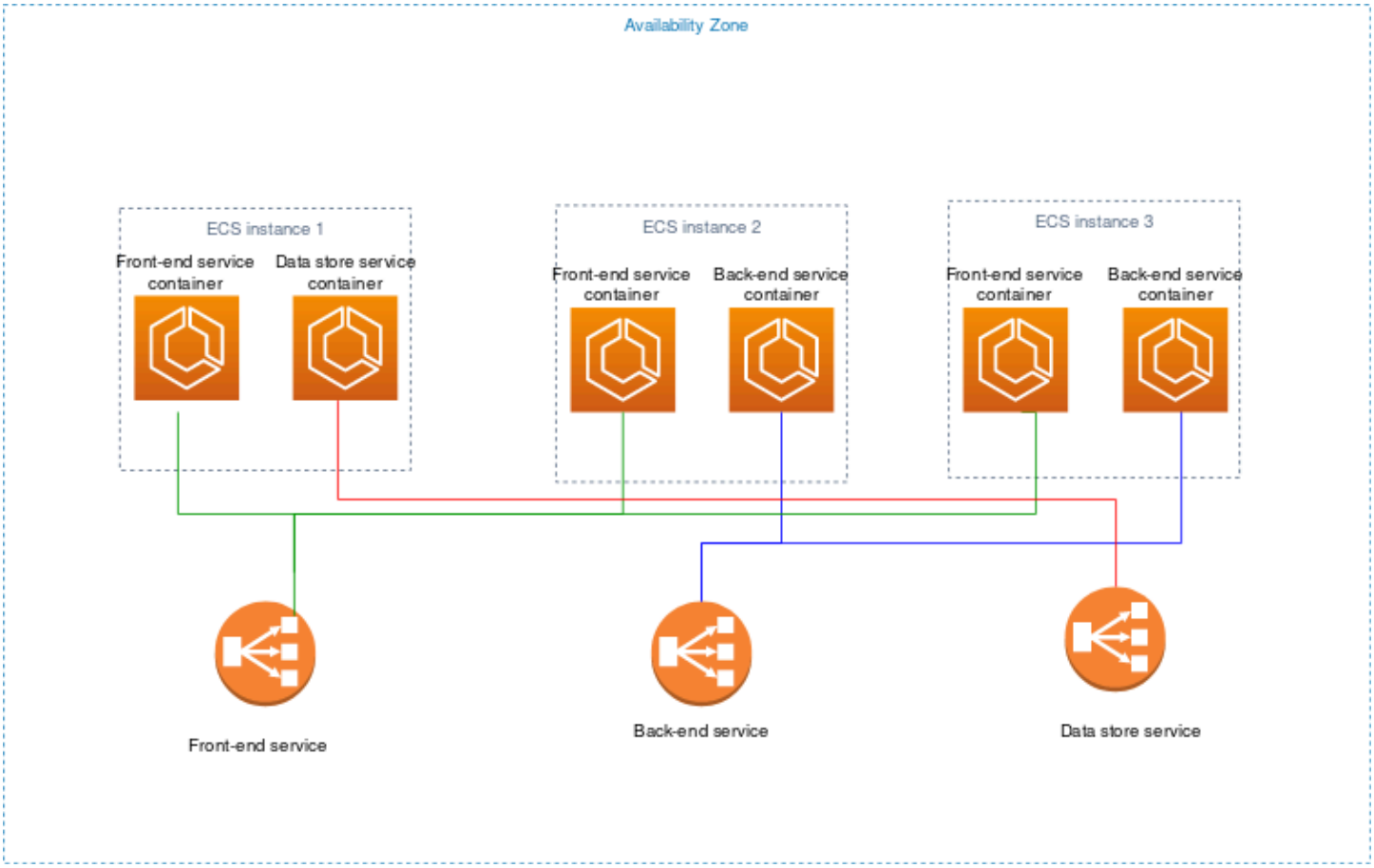
예를 들어 다음 구성 요소로 이루어진 애플리케이션을 가정해 보겠습니다.

- 웹 페이지에 정보를 표시하는 프런트엔드 서비스
- 프런트엔드 서비스에 API를 제공하는 백엔드 서비스
- 데이터 스토어

이 예에서는 공통 용도로 사용되는 컨테이너를 그룹화하는 태스크 정의를 만듭니다. 서로 다른 구성 요소를 여러 개의 개별 작업 정의로 구분합니다. 다음 예제 클러스터에는 3개의 프런트 엔드 서비스 컨테이너, 2개의 백엔드 서비스 컨테이너 및 1개의 데이터 스토어 서비스 컨테이너를 실행하는 3개의 컨테이너 인스턴스가 있습니다.

동일한 태스크 정의의 관련된 컨테이너를 그룹화할 수 있습니다(예: 반드시 함께 실행해야 하는 연결된 컨테이너). 예를 들어 로그 스트리밍 컨테이너를 프런트 엔드 서비스에 추가하고 동일한 태스크 정의에 추가합니다.

태스크 정의로부터 서비스를 생성하여 원하는 태스크의 가용성을 유지할 수 있습니다. 자세한 내용은 Amazon ECS 롤링 업데이트 배포 생성 섹션을 참조하세요. 서비스에서 컨테이너를 Elastic Load Balancing 로드 밸런서와 연결할 수 있습니다. 자세한 정보는 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산을 참조하세요. 애플리케이션 요구 사항이 변경될 경우 서비스를 업데이트하여 원하는 태스크의 개수를 조정할 수 있습니다. 또는 태스크에 새 버전의 컨테이너를 배포하도록 서비스를 업데이트할 수 있습니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.



Amazon ECS의 EC2 및 외부 시작 유형에 대한 컨테이너 이미지 가져오기 동작

컨테이너를 시작하는 데 걸리는 시간은 기본 컨테이너 이미지에 따라 다릅니다. 예를 들어, 더 큰 이미지(Debian, Ubuntu, Amazon1/2의 전체 버전)는 시작 시간이 더 오래 걸릴 수 있습니다. 상대적으로 슬림한 버전(Debian-slim, Ubuntu-slim, Amazon-slim) 또는 더 작은 기본 이미지(Alpine)에 비해 컨테이너에서 실행되는 서비스가 더 많기 때문입니다.

Amazon ECS 에이전트에서 작업을 시작하면 원격 레지스트리에서 Docker 이미지를 가져온 다음, 로컬 복사본을 캐싱합니다. 애플리케이션의 각 릴리스에 새 이미지 태그를 사용하는 경우 이 동작은 필요하지 않습니다.

`ECS_IMAGE_PULL_BEHAVIOR` 에이전트 파라미터는 이미지 가져오기 동작을 결정합니다. 다음과 같은 옵션을 사용할 수 있습니다.

- `ECS_IMAGE_PULL_BEHAVIOR: default`

  이미지는 원격에서 가져옵니다. 가져오지 못하면 인스턴스의 캐싱된 이미지가 사용됩니다.
- `ECS_IMAGE_PULL_BEHAVIOR: always`

이미지는 원격에서 가져옵니다. 가져오기에 실패하면 작업에 실패합니다.

배포 속도를 높이려면 Amazon ECS 에이전트 파라미터를 다음 값 중 하나로 설정합니다.

- `ECS_IMAGE_PULL_BEHAVIOR: once`

  이전에 동일 컨테이너 인스턴스의 작업에서 가져온 적이 없거나 이미지가 자동 이미지 정리 프로세스에 의해 제거된 경우에만 이미지를 원격에서 가져옵니다. 그렇지 않으면 인스턴스에 캐시된 이미지가 사용됩니다. 이 옵션은 불필요한 이미지 가져오기 시도를 방지합니다.

- `ECS_IMAGE_PULL_BEHAVIOR: prefer-cached`

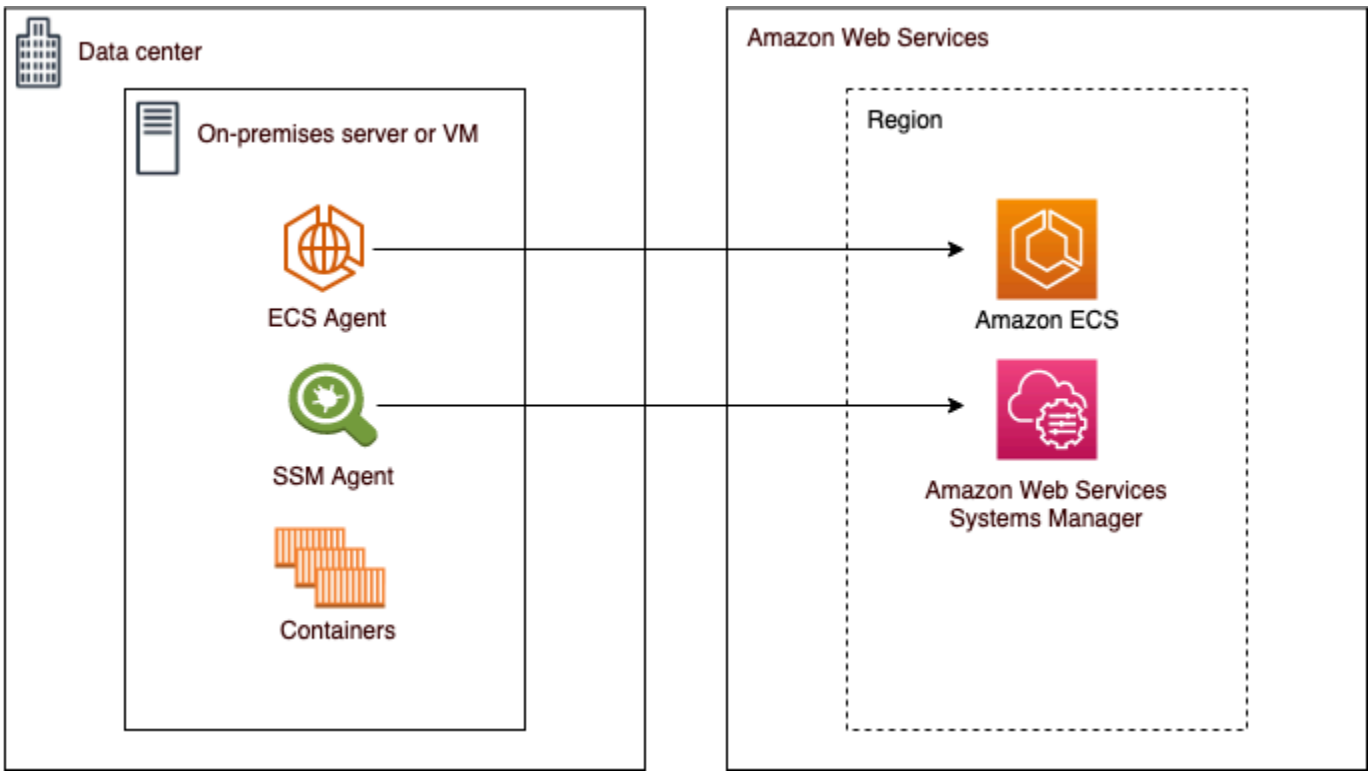
  캐시된 이미지가 없으면 이미지를 원격에서 가져옵니다. 그렇지 않으면 인스턴스에 캐시된 이미지가 사용됩니다. 캐시된 이미지가 제거되지 않도록 컨테이너에서 자동 이미지 정리는 꺼져 있습니다.

`ECS_IMAGE_PULL_BEHAVIOR` 파라미터를 위의 값 중 하나로 설정하면 Amazon ECS 에이전트가 다운로드한 기존 이미지를 사용하기 때문에 시간을 절약할 수 있습니다. 더 큰 Docker 이미지의 경우 네트워크를 통해 가져오는 데 다운로드 시간이 10~20초가 걸릴 수 있습니다.

Amazon ECS의 외부(Amazon ECS Anywhere) 시작 유형

Amazon ECS Anywhere는 외부 인스턴스(예: 온프레미스 서버 또는 가상 머신(VM))을 Amazon ECS 클러스터에 등록하도록 지원합니다. 외부 인스턴스는 아웃바운드 트래픽 또는 프로세스 데이터를 생성하는 애플리케이션 실행에 최적화되어 있습니다. 애플리케이션에 인바운드 트래픽이 필요한 경우 Elastic Load Balancing 지원이 없기 때문에 이러한 워크로드를 실행하는 데 효율성이 떨어집니다. Amazon ECS는 외부 인스턴스에서 서비스를 생성하거나 태스크를 실행하는 데 사용할 수 있는 새로운 `EXTERNAL` 시작 유형을 추가했습니다.

다음은 Amazon ECS Anywhere에 대한 상위 수준의 시스템 아키텍처 개요를 제공합니다. 온프레미스 서버에는 Amazon ECS 에이전트와 SSM 에이전트가 모두 설치되어 있습니다.



자세한 내용은 외부 시작 유형을 위한 Amazon ECS 클러스터 섹션을 참조하세요.

Amazon ECS 듀얼 스택 엔드포인트 사용

Amazon ECS 듀얼 스택 엔드포인트는 인터넷 프로토콜 버전 4(IPv4) 및 인터넷 프로토콜 버전 6(IPv6)을 통해 Amazon ECS에 대한 요청을 지원합니다. 모든 Amazon ECS 엔드포인트 목록은 AWS 일반 참조의 Amazon ECS 엔드포인트 및 할당량을 참조하세요.

REST API를 사용하는 경우, 엔드포인트 이름(URI)을 사용하여 Amazon ECS 엔드포인트에 직접 액세스합니다. Amazon ECS는 리전 듀얼 스택 엔드포인트 이름만 지원합니다. 이는 이름의 일부로 리전을 지정해야 함을 뜻합니다.

듀얼 스택 엔드포인트 이름에는 다음 명명 규칙 `ecs.region.api.aws`를 사용합니다.

AWS Command Line Interface(AWS CLI) 및 AWS SDK를 사용하는 경우, 파라미터 또는 플래그를 사용하여 듀얼 스택 엔드포인트를 변경할 수 있습니다. 구성 파일의 Amazon ECS 엔드포인트를 재정의하여 듀얼 스택 엔드포인트를 직접 지정할 수도 있습니다.

다음 섹션에서는 AWS CLI, AWS SDK 및 REST API의 듀얼 스택 엔드포인트를 사용하는 방법을 설명합니다.

주제

- AWS CLI의 듀얼 스택 엔드포인트 사용
- AWS SDK의 듀얼 스택 엔드포인트 사용
- REST API의 듀얼 스택 엔드포인트 사용

AWS CLI의 듀얼 스택 엔드포인트 사용

이 단원에서는 듀얼 스택 엔드포인트에 요청하는 데 사용되는 AWS CLI 명령의 예를 보여 줍니다. AWS CLI 설치 또는 최신 버전으로의 업그레이드 방법에 대한 자세한 내용은 AWS Command Line Interface 버전 2 사용 설명서의 AWS CLI 최신 버전 설치 또는 업데이트를 참조하세요.

듀얼 스택 엔드포인트를 사용하려면 AWS CLI의 `config` 파일에서 구성 값 `use_dualstack_endpoint`를 `true`로 설정하여 `ecs` AWS CLI 명령의 모든 Amazon ECS 요청을 지정된 리전의 듀얼 스택 엔드포인트로 보냅니다. `--region` 옵션을 사용하여 `config` 파일 또는 명령에서 리전을 지정할 수 있습니다. AWS CLI의 구성 파일에 대한 자세한 내용은 AWS Command Line Interface 버전 2 사용 설명서의 AWS CLI의 구성 및 자격 증명 파일 설정을 참조하세요.

특정 AWS CLI 명령에 대해서만 듀얼 스택 엔드포인트를 사용하려는 경우, 다음 방법 중 하나를 사용할 수 있습니다.

- `ecs` 명령에 대해 `--endpoint-url` 파라미터를 `https://ecs.`*`aws-region`*`.api.aws` 또는 `http://ecs.`*`aws-region`*`.api.aws`로 설정하여 명령별로 듀얼 스택 엔드포인트를 사용할 수 있습니다.

  다음 예제 명령은 사용 가능한 모든 클러스터를 나열하고 해당 요청에 듀얼 스택 엔드포인트를 사용합니다.

  ```
  $ aws ecs list-clusters --endpoint-url https://ecs.aws-region.api.aws
  ```

- AWS Config 파일에 별도의 프로파일을 설정할 수 있습니다. 예를 들어, `use_dualstack_endpoint`를 `true`로 설정하는 프로파일을 하나 작성하고, `use_dualstack_endpoint`를 설정하지 않는 프로파일을 하나 작성합니다. 명령을 실행할 때 듀얼 스택 엔드포인트를 사용할지 여부에 따라 사용할 프로파일을 지정합니다.

AWS SDK의 듀얼 스택 엔드포인트 사용

이 섹션에서는 AWS SDK를 사용하여 듀얼 스택 엔드포인트에 액세스하는 방법의 예제를 보여줍니다.

AWS SDK for Java 2.x

다음 예제는 AWS SDK for Java 2.x를 사용하여 us-east-1 리전의 듀얼 스택 엔드포인트를 지정하는 방법을 보여줍니다.

```
Region region = Region.US_EAST_1
EcsClient client =
 EcsClient.builder().region(region).dualstackEnabled(true).build();
```

AWS SDK for Go

다음 예제는 AWS SDK for Go를 사용하여 us-east-1 리전의 듀얼 스택 엔드포인트를 지정하는 방법을 보여줍니다.

```
sess := session.Must(session.NewSession())
svc := ecs.New(sess, &aws.Config{
    Region: aws.String(endpoints.UsEast1RegionID),
    Endpoint: aws.String("https://ecs.us-east-1.api.aws")
})
```

자세한 내용은 AWS SDK 및 도구 참조 가이드의 듀얼 스택 및 FIPS 엔드포인트를 참조하세요.

REST API의 듀얼 스택 엔드포인트 사용

REST API를 사용하는 경우 요청에서 지정하여 듀얼 스택 엔드포인트에 직접 액세스할 수 있습니다. 다음 예제에서는 듀얼 스택 엔드포인트를 사용하여 us-east-1 리전의 모든 Amazon ECS 클러스터를 나열합니다.

```
POST / HTTP/1.1
Host: ecs.us-east-1.api.aws
Accept-Encoding: identity
Content-Length: 2
X-Amz-Target: AmazonEC2ContainerServiceV20141113.ListClusters
X-Amz-Date: 20150429T170621Z
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{}
```

공유 서브넷, 로컬 영역 및 Wavelength Zone의 Amazon ECS 애플리케이션

Amazon ECS는 짧은 지연 시간 또는 로컬 데이터 처리가 필요한 경우 Local Zone, Wavelength Zone 및 AWS Outposts를 사용하는 워크로드를 지원합니다.

- 로컬 영역을 AWS 리전의 확장으로 사용하여 최종 사용자와 가까운 여러 위치에 리소스를 배치할 수 있습니다.
- Wavelength Zone을 사용하여 5G 디바이스 및 최종 사용자에게 매우 짧은 지연 시간을 제공하는 애플리케이션을 빌드할 수 있습니다. Wavelength는 표준 AWS 컴퓨팅 및 스토리지 서비스를 통신 사업자의 5G 네트워크 엣지에 배포합니다.
- AWS Outposts는 네이티브 AWS 서비스, 인프라 및 운영 모델을 사실상 모든 데이터 센터, 코로케이션 공간 또는 온프레미스 시설로 옮길 수 있습니다.

> **Important**
>
> 현재 AWS Fargate 워크로드의 Amazon ECS는 Local Zone, Wavelength Zone 또는 AWS Outposts에서 지원되지 않습니다.

Local Zone, Wavelength Zone, AWS Outposts 사이의 차이에 대한 자세한 정보는 AWS Wavelength FAQ에서 대기 시간이 짧거나 로컬 데이터 처리가 필요한 애플리케이션에서 AWS Wavelength, AWS Local Zone, AWS Outposts 중 무엇을 사용할지 결정하는 방법을 참조하세요.

공유 서브넷

VPC 공유를 사용하여 서브넷을 동일한 AWS Organizations 내의 다른 AWS 계정과 공유할 수 있습니다.

다음 사항을 고려하며 EC2 시작 유형에 공유 VPC를 사용할 수 있습니다.

- 참여자 계정에서 Amazon ECS 리소스에 대해 서브넷을 사용하려면 먼저 VPC 서브넷의 소유자가 참여자 계정과 해당 서브넷을 공유해야 합니다.
- 컨테이너 인스턴스에 대해 VPC의 기본 보안 그룹을 사용할 수 없습니다. 소유자에게 속해 있기 때문입니다. 또한 참여자는 소유자 또는 다른 참여자가 소유한 보안 그룹을 사용하여 인스턴스를 시작할 수 없습니다.

- 공유 서브넷에서는 참여자와 소유자가 각 계정 내의 보안 그룹을 별도로 제어합니다. 서브넷 소유자는 참여자가 생성한 보안 그룹을 볼 수 있지만 이 보안 그룹에 대해 조치를 취할 수는 없습니다. 서브넷 소유자가 보안 그룹을 제거하거나 수정하고자 하는 경우 보안 그룹을 생성한 참가자가 조치를 취해야 합니다.
- 공유 VPC 소유자는 참여자가 공유 서브넷에서 생성한 클러스터의 보기, 업데이트 또는 삭제가 불가능합니다. 여기에 더해 계정마다 액세스 권한이 다른 VPC 리소스가 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 소유자 및 참여자에 대한 책임 및 권한을 참조하세요.

다음 사항을 고려하며 Fargate 시작 유형에 공유 VPC를 사용할 수 있습니다.

- 참여자 계정에서 Amazon ECS 리소스에 대해 서브넷을 사용하려면 먼저 VPC 서브넷의 소유자가 참여자 계정과 해당 서브넷을 공유해야 합니다.
- VPC의 기본 보안 그룹을 사용하여 서비스를 생성하거나 작업을 실행할 수 없습니다. 소유자에게 속해 있기 때문입니다. 또한 참여자는 소유자 또는 다른 참여자가 소유한 보안 그룹을 사용하여 서비스를 생성하거나 작업을 실행할 수 없습니다.
- 공유 서브넷에서는 참여자와 소유자가 각 계정 내의 보안 그룹을 별도로 제어합니다. 서브넷 소유자는 참여자가 생성한 보안 그룹을 볼 수 있지만 이 보안 그룹에 대해 조치를 취할 수는 없습니다. 서브넷 소유자가 보안 그룹을 제거하거나 수정하고자 하는 경우 보안 그룹을 생성한 참가자가 조치를 취해야 합니다.
- 공유 VPC 소유자는 참여자가 공유 서브넷에서 생성한 클러스터의 보기, 업데이트 또는 삭제가 불가능합니다. 여기에 더해 계정마다 액세스 권한이 다른 VPC 리소스가 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 소유자 및 참여자에 대한 책임 및 권한을 참조하세요.

VPC 서브넷 공유에 관한 자세한 내용은 Amazon VPC 사용 설명서의 다른 계정과 VPC 공유를 참조하세요.

로컬 영역

로컬 영역은 사용자와 지리적으로 근접한 AWS 리전의 확장입니다. Local Zones는 인터넷에 대한 자체 연결을 가지고 있으며 AWS Direct Connect를 지원합니다. Local Zones에서 생성된 리소스는 대기 시간이 매우 짧은 통신으로 로컬 사용자에게 제공될 수 있습니다. 자세한 내용은 AWS 로컬 영역을 참조하세요.

로컬 영역은 리전 코드 다음에 위치를 나타내는 식별자로 표시됩니다(예: `us-west-2-lax-1a`).

로컬 영역을 사용하려면 영역에 옵트인해야 합니다. 옵트인하고 나면 로컬 영역에 Amazon VPC 및 서브넷을 생성해야 합니다.

Amazon EC2 인스턴스, Amazon FSx 파일 서버 및 Application Load Balancer를 시작하여 Amazon ECS 클러스터와 작업에 사용할 수 있습니다.

자세한 내용은 AWS Local Zones 사용 설명서의 What is AWS Local Zones?를 참조하세요.

Wavelength Zone

AWS Wavelength를 사용하면 개발자는 모바일 디바이스 및 최종 사용자에게 매우 짧은 지연 시간을 제공하는 애플리케이션을 빌드할 수 있습니다. Wavelength는 표준 AWS 컴퓨팅 및 스토리지 서비스를 통신 사업자의 5G 네트워크 엣지에 배포합니다. Amazon Virtual Private Cloud를 하나 이상의 Wavelength Zone으로 확장할 수 있습니다. 그런 다음 Amazon EC2 인스턴스와 같은 AWS 리소스를 사용하여 리전의 AWS 서비스에 대한 극도로 짧은 지연 시간 및 연결을 필요로 하는 애플리케이션을 실행할 수 있습니다.

Wavelength Zone은 Wavelength 인프라가 배포된 통신 사업자 위치의 격리된 영역입니다. Wavelength Zone은 AWS 리전에 연결되어 있습니다. Wavelength Zone은 리전의 논리적 확장이며, 리전의 제어 플레인에 의해 관리됩니다.

리전 코드 뒤에 Wavelength Zone을 나타내는 식별자를 붙여 Wavelength Zone을 나타냅니다(예: `us-east-1-wl1-bos-wlz-1`).

Wavelength Zone을 사용하려면 Wavelength Zone을 옵트인해야 합니다. 옵트인하고 나면 Wavelength Zone에 Amazon VPC와 서브넷을 생성해야 합니다. 그런 다음 영역에서 Amazon EC2 인스턴스를 실행하여 Amazon EC2 클러스터와 작업에 사용할 수 있습니다.

자세한 정보는 AWS Wavelength개발자 가이드에서 AWS Wavelength 시작하기를 참조하세요.

Wavelength Zones를 모든 AWS 리전에서 사용할 수 있는 것은 아닙니다. Wavelength Zone을 지원하는 리전에 대한 자세한 내용은 AWS Wavelength 개발자 안내서의 사용 가능한 Wavelength Zone을 참조하세요.

AWS Outposts의 Amazon Elastic Container Service

AWS Outposts는 온프레미스 시설의 기본 AWS 서비스, 인프라 및 운영 모델을 지원합니다. AWS Outposts 환경에서는 AWS 클라우드에서 사용하는 것과 동일한 AWS API, 도구, 인프라를 사용할 수 있습니다.

AWS Outposts의 Amazon ECS는 온프레미스 데이터 및 애플리케이션과 매우 가까운 거리에서 실행해야 하는 대기 시간이 짧은 워크로드에 이상적입니다.

AWS Outposts에 대한 자세한 내용은 AWS Outposts 사용 설명서를 참조하세요.

고려 사항

다음은 AWS Outposts에서 Amazon ECS를 사용할 때 고려해야 할 사항입니다.

- Amazon Elastic Container Registry, AWS Identity and Access Management 및 Network Load Balancer는 AWS Outposts가 아닌 AWS 리전에서 실행됩니다. 이렇게 하면 이러한 서비스와 컨테이너 간의 지연 시간이 증가합니다.
- AWS Fargate는 AWS Outposts에서 사용할 수 없습니다.

다음은 AWS Outposts에 대한 네트워크 연결 고려 사항입니다.

- AWS Outposts와 AWS 리전 간의 네트워크 연결이 끊어지면 클러스터는 계속 실행됩니다. 하지만 연결이 복원될 때까지 새 클러스터를 생성하거나 기존 클러스터에 대해 새 태스크를 수행할 수 없습니다. 인스턴스에 장애가 발생한 경우 인스턴스는 자동으로 교체되지 않습니다. CloudWatch Logs 에이전트가 로그 및 이벤트 데이터를 업데이트할 수 없습니다.
- AWS Outposts와 AWS 리전 간에 안정적이고 가용성이 높으며 지연 시간이 짧은 연결을 제공하는 것이 좋습니다.

사전 조건

다음은 AWS Outposts에서 Amazon ECS를 사용하기 위한 사전 조건입니다.

- 온프레미스 데이터 센터에 Outpost가 설치 및 구성되어 있어야 합니다.
- Outpost와 AWS 리전 간에 안정적인 네트워크 연결이 있어야 합니다.

AWS Outposts에서 클러스터 생성 개요

다음은 구성 개요입니다.

1. AWS Outposts에 대한 권한으로 역할 및 정책을 생성합니다.
2. AWS Outposts에 대한 권한이 있는 IAM 인스턴스 프로필을 생성합니다.
3. VPC를 생성하거나 AWS Outposts와 동일한 리전에 있는 기존 VPC를 사용합니다.
4. 서브넷을 생성하거나 AWS Outposts와 연결된 기존 서브넷을 사용합니다.

   이는 컨테이너 인스턴스가 실행되는 서브넷입니다.
5. 클러스터의 컨테이너 인스턴스에 대한 보안 그룹을 생성합니다.

6. Amazon ECS 클러스터를 생성합니다.
7. 인스턴스를 클러스터로 시작하기 위해 Amazon ECS 컨테이너 에이전트 환경 변수를 정의합니다.
8. 컨테이너를 실행합니다.

Amazon ECS와 AWS Outposts의 통합 방법에 대한 자세한 내용은 Extend Amazon ECS across two AWS Outposts racks를 참조하세요.

다음 예제에서는 AWS Outposts에서 Amazon ECS 클러스터를 생성합니다.

1. AWS Outposts에 대한 권한으로 역할 및 정책을 생성합니다.

   `role-policy.json` 파일은 리소스에 대한 효과 및 작업이 포함된 정책 문서입니다. 파일 형식에 대한 자세한 내용은 IAM API Reference(IAM API 참조)의 PutRolePolicy를 참조하세요.

   ```
   aws iam create-role --role-name ecsRole \
       --assume-role-policy-document file://ecs-policy.json
   aws iam put-role-policy --role-name ecsRole --policy-name ecsRolePolicy \
       --policy-document file://role-policy.json
   ```

2. AWS Outposts에 대한 권한이 있는 IAM 인스턴스 프로필을 생성합니다.

   ```
   aws iam create-instance-profile --instance-profile-name outpost
   aws iam add-role-to-instance-profile --instance-profile-name outpost \
       --role-name ecsRole
   ```

3. VPC를 생성합니다.

   ```
   aws ec2 create-vpc --cidr-block 10.0.0.0/16
   ```

4. AWS Outposts와 연결된 서브넷을 생성합니다.

   ```
   aws ec2 create-subnet \
       --cidr-block 10.0.3.0/24 \
       --vpc-id vpc-xxxxxxxx \
       --outpost-arn arn:aws:outposts:us-west-2:123456789012:outpost/op-
   xxxxxxxxxxxxxxxxx \
       --availability-zone-id usw2-az1
   ```

5. 컨테이너 인스턴스에 대한 보안 그룹을 생성하여 AWS Outposts를 위한 적절한 CIDR 범위를 지정합니다. (이 단계는 AWS Outposts에 대해 다릅니다.)

```
aws ec2 create-security-group --group-name MyOutpostSG
aws ec2 authorize-security-group-ingress --group-name MyOutpostSG --protocol tcp \
    --port 22 --cidr 10.0.3.0/24
aws ec2 authorize-security-group-ingress --group-name MyOutpostSG --protocol tcp \
    --port 80 --cidr 10.0.3.0/24
```

6. 클러스터를 생성합니다.
7. Amazon ECS 컨테이너 에이전트 환경 변수를 정의하여 이전 단계에서 생성한 클러스터로 인스턴스를 시작하고 클러스터를 식별하는 데 도움이 되게 추가하려는 태그를 정의합니다(예: 클러스터가 Outpost용임을 나타내기 위해 `Outpost`).

```
#! /bin/bash
cat << 'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_IMAGE_PULL_BEHAVIOR=prefer-cached
ECS_CONTAINER_INSTANCE_TAGS={"environment": "Outpost"}
EOF
```

> **Note**
>
> 해당 리전의 Amazon ECR에서 컨테이너 이미지를 가져와 지연이 발생하는 것을 방지하려면 이미지 캐시를 사용합니다. 이렇게 하려면 작업이 실행될 때마다 `ECS_IMAGE_PULL_BEHAVIOR`를 `prefer-cached`로 설정하여 인스턴스 자체에서 캐시된 이미지를 사용하도록 Amazon ECS 에이전트를 구성합니다.

8. 이 인스턴스가 실행되어야 하는 AWS Outposts의 VPC와 서브넷과 AWS Outposts에서 사용할 수 있는 인스턴스 유형을 지정하여 컨테이너 인스턴스를 생성합니다. (이 단계는 AWS Outposts에 대해 다릅니다.)

   `userdata.txt` 파일에는 인스턴스가 일반적인 자동 구성 태스크를 수행하고 인스턴스가 시작된 후 스크립트를 실행하는 데 사용할 수 있는 사용자 데이터가 포함되어 있습니다. API 직접 호출용 파일에 대한 자세한 내용은 Amazon EC2 사용 설명서의 시작 시 Linux 인스턴스에서 명령 실행을 참조하세요.

```
aws ec2 run-instances --count 1 --image-id ami-xxxxxxxx --instance-type c5.large \
    --key-name aws-outpost-key --subnet-id subnet-xxxxxxxxxxxxxxxxx \
    --iam-instance-profile Name outpost --security-group-id sg-xxxxxx \
    --associate-public-ip-address --user-data file://userdata.txt
```

> **Note**
>
> 이 명령은 클러스터에 인스턴스를 추가할 때도 사용됩니다. 클러스터에 배포된 모든 컨테이너는 해당 특정 AWS Outposts에 배치됩니다.

Amazon ECS 네트워킹 모범 사례

현대적 애플리케이션은 일반적으로 서로 통신하는 여러 분산 구성 요소로 구축됩니다. 예를 들어, 모바일 또는 웹 애플리케이션은 API 엔드포인트와 통신할 수 있으며, API는 인터넷을 통해 통신하는 여러 마이크로서비스를 통해 구동될 수 있습니다.

애플리케이션을 인터넷에 연결하는 모범 사례에 대한 자세한 내용은 Amazon ECS 애플리케이션을 인터넷에 연결 섹션을 참조하세요.

인터넷에서 Amazon ECS로 인바운드 연결을 수신하는 모범 사례에 대한 자세한 내용은 인터넷에서 Amazon ECS로의 인바운드 연결을 수신하는 모범 사례 섹션을 참조하세요.

Amazon ECS를 다른 AWS 서비스에 연결하는 모범 사례에 대한 자세한 내용은 VPC 내에서 Amazon ECS를 AWS 서비스에 연결하는 모범 사례 섹션을 참조하세요.

VPC 내 서비스에 연결하는 모범 사례에 대한 자세한 내용은 VPC에서 Amazon ECS 서비스를 연결하는 모범 사례 섹션을 참조하세요.

AWS 계정 및 VPC 간 네트워킹 서비스의 모범 사례에 대한 자세한 내용은 AWS 계정 및 VPC 간의 Amazon ECS 서비스 네트워킹을 위한 모범 사례 섹션을 참조하세요.

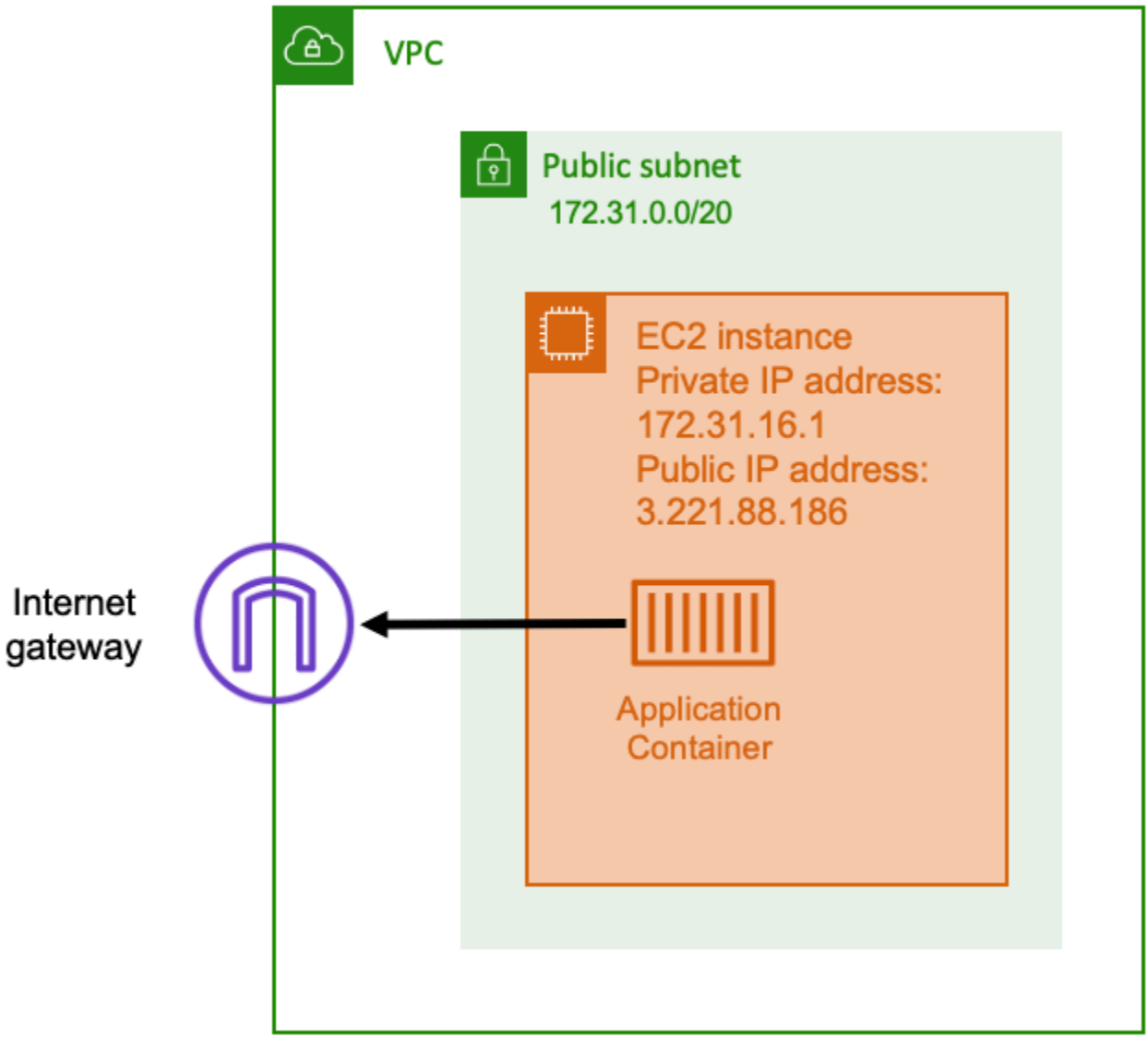
네트워킹 문제 해결을 위한 서비스의 모범 사례에 대한 자세한 내용은 Amazon ECS 네트워킹 문제 해결을 위한 AWS 서비스 섹션을 참조하세요.

Amazon ECS 애플리케이션을 인터넷에 연결

대부분의 컨테이너화된 애플리케이션에는 인터넷에 대한 아웃바운드 액세스가 필요한 구성 요소가 최소한 몇 개 있습니다. 예를 들어 모바일 앱의 백엔드에는 푸시 알림에 대해 아웃바운드 액세스가 필요합니다.

Amazon Virtual Private Cloud에는 VPC와 인터넷 간의 통신을 용이하게 하는 두 가지 기본 방법이 있습니다.

퍼블릭 서브넷 및 인터넷 게이트웨이



인터넷 게이트웨이로 라우팅되는 퍼블릭 서브넷을 사용하는 경우 컨테이너화된 애플리케이션을 퍼블릭 서브넷의 VPC 내부 호스트에서 실행할 수 있습니다. 컨테이너를 실행하는 호스트에는 퍼블릭 IP 주소가 할당됩니다. 이 퍼블릭 IP 주소는 인터넷에서 라우팅할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터넷 게이트웨이를 참조하세요.

이 네트워크 아키텍처는 애플리케이션을 실행하는 호스트와 인터넷상의 다른 호스트 간 직접 통신을 용이하게 합니다. 통신은 양방향으로 이루어집니다. 즉, 인터넷상의 다른 호스트에 아웃바운드 연결을 설정할 수 있을 뿐만 아니라 인터넷상의 다른 호스트가 해당 호스트에 연결을 시도할 수도 있습니다. 따라서 보안 그룹 및 방화벽 규칙에 주의해야 합니다. 그러면 인터넷상의 다른 호스트는 사용자가 열고 싶지 않은 연결을 열 수 없습니다.

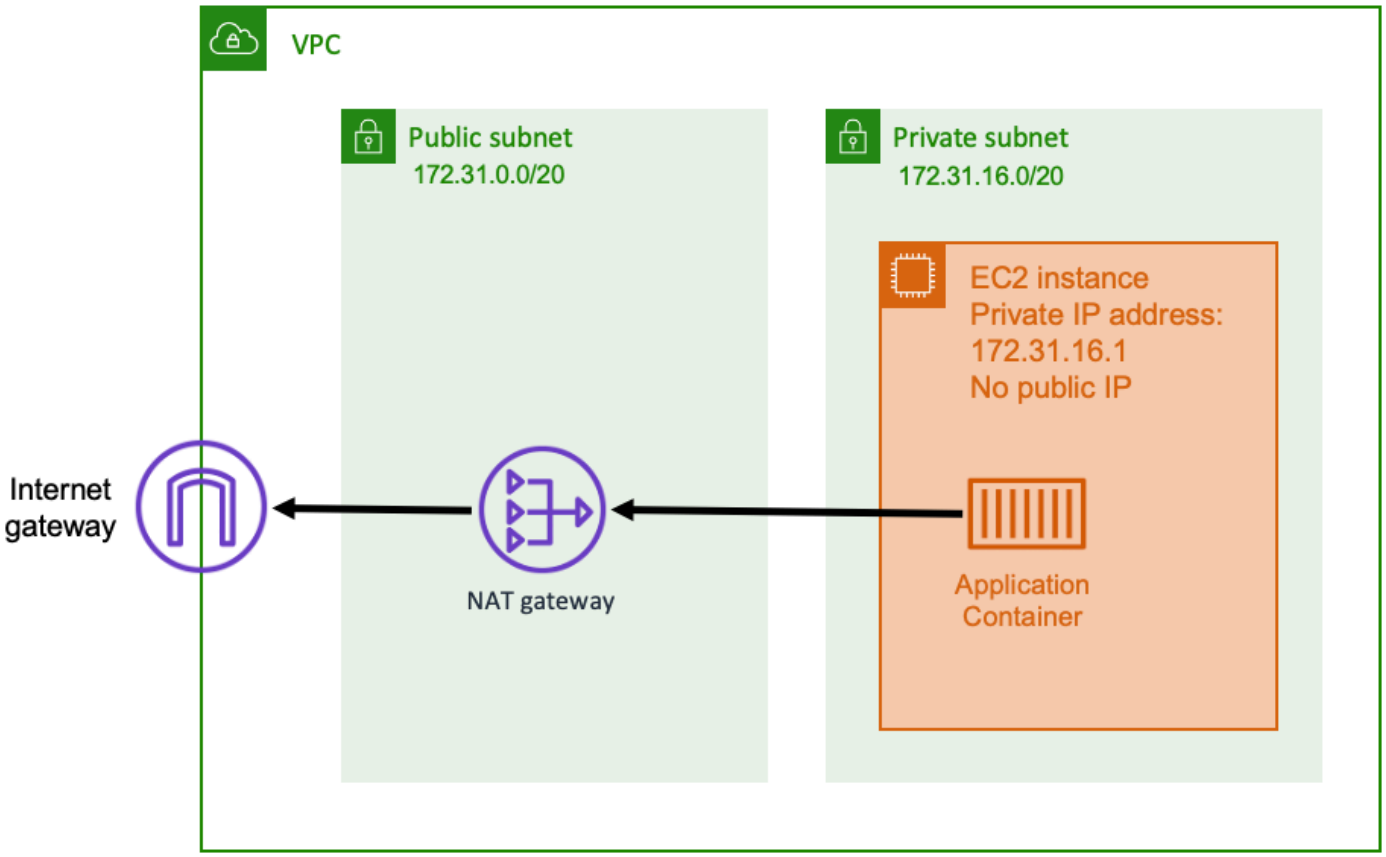
예를 들어, 애플리케이션이 Amazon EC2에서 실행되는 경우 SSH 액세스를 위한 포트 22가 열려 있지 않아야 합니다. 그렇지 않으면 인스턴스가 인터넷의 악성 봇으로부터 지속적인 SSH 연결 시도를 받을 수 있습니다. 이 봇은 퍼블릭 IP 주소를 트롤링합니다. 악성 봇이 열린 SSH 포트를 찾으면 암호를 무차별 입력하여 인스턴스에 액세스하려고 시도합니다. 이 때문에 많은 조직에서는 퍼블릭 서브넷의 사용을 제한하고 전부는 아니더라도 대부분의 리소스를 프라이빗 서브넷 내에 두는 방식을 선호합니다.

네트워킹에 퍼블릭 서브넷을 사용하는 방식은 많은 양의 대역폭이나 최소 지연 시간이 필요한 퍼블릭 애플리케이션에 적합합니다. 적용 가능한 사용 사례로, 비디오 스트리밍 및 게임 서비스가 포함됩니다.

이 네트워킹 접근 방식은 Amazon EC2에서 Amazon ECS를 사용하는 경우 및 AWS Fargate에서 사용하는 경우 모두 지원됩니다.

- Amazon EC2 - 퍼블릭 서브넷에서 EC2 인스턴스를 시작할 수 있습니다. Amazon ECS는 이러한 EC2 인스턴스를 클러스터 용량으로 사용하며, 인스턴스에서 실행되는 모든 컨테이너는 아웃바운드 네트워킹을 위해 호스트의 기본 퍼블릭 IP 주소를 사용할 수 있습니다. `host` 및 `bridge` 네트워크 모드 모두에 적용됩니다. 그러나 `awsvpc` 네트워크 모드는 퍼블릭 IP 주소를 사용하는 작업 ENI를 제공하지 않습니다. 따라서 인터넷 게이트웨이를 직접 사용할 수 없습니다.
- Fargate - Amazon ECS 서비스를 생성할 때 서비스의 네트워킹 구성에 대한 퍼블릭 서브넷을 지정하고 퍼블릭 IP 주소 할당 옵션을 사용합니다. 각 Fargate 작업은 퍼블릭 서브넷에서 네트워크로 연결되며 인터넷과 직접 통신하기 위한 자체 퍼블릭 IP 주소를 가지고 있습니다.

프라이빗 서브넷 및 NAT 게이트웨이



프라이빗 서브넷 및 NAT 게이트웨이를 사용하는 경우 프라이빗 서브넷에 있는 호스트에서 컨테이너화된 애플리케이션을 실행할 수 있습니다. 따라서 이 호스트에는 VPC 내에서 라우팅할 수 있지만 인터넷에서 라우팅할 수 없는 프라이빗 IP 주소가 있습니다. 즉, VPC 내의 다른 호스트는 프라이빗 IP 주소를 사용하여 호스트에 연결할 수 있지만 인터넷상의 다른 호스트는 호스트와 인바운드 통신을 할 수 없습니다.

프라이빗 서브넷에서는 Network Address Translation(NAT) 게이트웨이를 사용하여 프라이빗 서브넷 내부의 호스트를 인터넷에 연결할 수 있습니다. 인터넷상의 호스트는 퍼블릭 서브넷 내에 있는 NAT 게이트웨이의 퍼블릭 IP 주소에서 시작하는 인바운드 연결을 수신합니다. NAT 게이트웨이는 인터넷과 프라이빗 서비스 사이에서 브리지 역할을 합니다. 이 구성은 인터넷에서 공격자가 직접 액세스하지 못하도록 VPC를 보호하므로 보안상의 이유로 종종 선호됩니다. 자세한 정보는 Amazon VPC 사용 설명서의 NAT 게이트웨이 섹션을 참조하세요.

이 프라이빗 네트워킹 접근 방식은 컨테이너를 외부 직접 액세스로부터 보호하려는 시나리오에 적합합니다. 적용 가능한 시나리오로, 사용자 데이터와 암호를 저장하는 결제 처리 시스템 또는 컨테이너가 포함됩니다. 계정에서 NAT 게이트웨이 생성 및 사용에 대한 요금이 청구됩니다. NAT 게이트웨이 시간

당 사용 요금 및 데이터 처리 요금도 적용됩니다. 이중화를 위해 각 가용 영역에 하나의 NAT 게이트웨이가 있어야 합니다. 이렇게 하면 단일 가용 영역의 가용성이 손실되어도 아웃바운드 연결이 손상되지 않습니다. 따라서 워크로드가 적은 경우 프라이빗 서브넷 및 NAT 게이트웨이를 사용하는 것이 더 비용 효율적일 수 있습니다.

이 네트워킹 접근 방식은 Amazon EC2에서 Amazon ECS를 사용하는 경우 및 AWS Fargate에서 사용하는 경우 모두 지원됩니다.

- Amazon EC2 - 프라이빗 서브넷에서 EC2 인스턴스를 시작할 수 있습니다. 이러한 EC2 호스트에서 실행되는 컨테이너는 기본 호스트 네트워킹을 사용하며 아웃바운드 요청은 NAT 게이트웨이를 통과합니다.
- Fargate - Amazon ECS 서비스를 생성할 때 서비스의 네트워킹 구성에 대한 프라이빗 서브넷을 지정하고 퍼블릭 IP 주소 할당 옵션을 사용하지 않습니다. 각 Fargate 작업은 프라이빗 서브넷에서 호스팅됩니다. 아웃바운드 트래픽은 해당 프라이빗 서브넷에 연결한 모든 NAT 게이트웨이를 통해 라우팅됩니다.

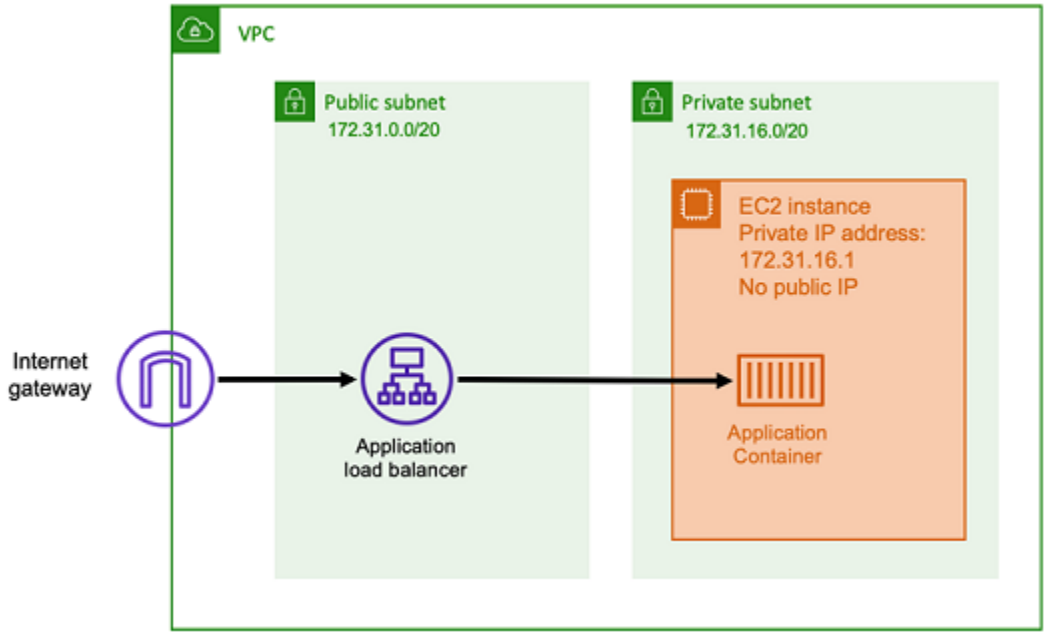
인터넷에서 Amazon ECS로의 인바운드 연결을 수신하는 모범 사례

퍼블릭 서비스를 실행하는 경우 인터넷의 인바운드 트래픽을 수락해야 합니다. 예를 들어 퍼블릭 웹 사이트는 브라우저의 인바운드 HTTP 요청을 수락해야 합니다. 이 경우 인터넷의 다른 호스트도 애플리케이션 호스트에 대한 인바운드 연결을 시작해야 합니다.

이 문제에 대한 한 가지 접근 방식은 퍼블릭 IP 주소를 사용하는 퍼블릭 서브넷에 있는 호스트에서 컨테이너를 시작하는 것입니다. 그러나 대규모 애플리케이션에는 권장되지 않습니다. 이러한 경우에는 인터넷 및 애플리케이션 사이에 확장 가능한 입력 레이어를 배치하는 것이 더 좋습니다. 이 접근 방식에서는 이 섹션에 나열된 모든 AWS 서비스를 입력으로 사용할 수 있습니다.

Application Load Balancer

Application Load Balancer는 애플리케이션 계층에서 작동합니다. 이는 개방형 시스템 상호 연결(OSI) 모델의 일곱 번째 계층입니다. 따라서 Application Load Balancer는 퍼블릭 HTTP 서비스에 적합합니다. 웹 사이트 또는 HTTP REST API가 있는 경우 Application Load Balancer는 이 워크로드에 적합한 로드 밸런서입니다. 자세한 내용은 Application Load Balancer 사용 설명서의 What is an Application Load Balancer?를 참조하세요.



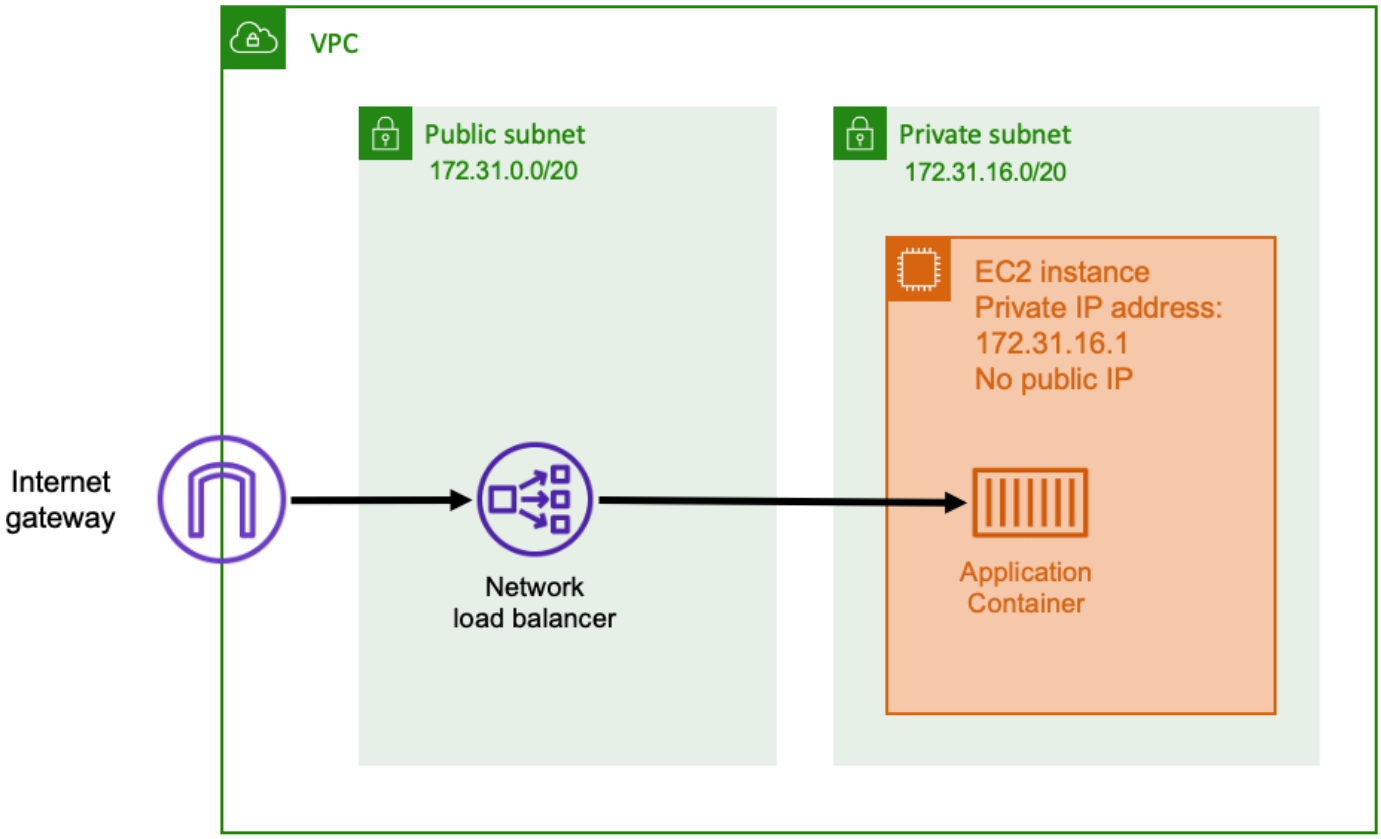
이 아키텍처에서는 퍼블릭 서브넷에 Application Load Balancer를 생성하여 퍼블릭 IP 주소를 사용하고 인터넷의 인바운드 연결을 수신할 수 있습니다. Application Load Balancer에서 인바운드 연결 또는 특히 HTTP 요청을 수신하면 프라이빗 IP 주소를 사용하여 애플리케이션에 대한 연결을 엽니다. 그런 다음, 내부 연결을 통해 요청을 전달합니다.

Application Load Balancer의 장점은 다음과 같습니다.

- SSL/TLS 종료 - Application Load Balancer는 클라이언트와의 통신을 위한 보안 HTTPS 통신 및 인증서를 유지할 수 있습니다. 선택적으로 로드 밸런서 수준에서 SSL 연결을 종료할 수 있으므로 애플리케이션에서 인증서를 처리할 필요가 없습니다.
- 고급 라우팅 - Application Load Balancer는 여러 DNS 호스트 이름을 보유할 수 있습니다. 또한 호스트 이름이나 요청 경로와 같은 지표를 기반으로 수신 HTTP 요청을 다른 대상으로 전송하는 고급 라우팅 기능도 지원합니다. 즉, 단일 Application Load Balancer를 여러 내부 서비스 또는 REST API의 여러 경로에 있는 마이크로서비스의 입력으로도 사용할 수 있습니다.
- gRPC 지원 및 웹 소켓 - Application Load Balancer는 HTTP 외에도 다양한 기능을 처리할 수 있습니다. 또한 HTTP/2 지원을 통해 gRPC 및 웹 소켓 기반 서비스를 로드 밸런싱할 수 있습니다.
- 보안 - Application Load Balancer는 악성 트래픽으로부터 애플리케이션을 보호하는 데 도움이 됩니다. 여기에는 HTTP 동기화 해제 완화와 같은 기능이 포함되어 있으며, AWS 웹 애플리케이션 방화벽(AWS WAF)과 통합됩니다. AWS WAF에서는 SQL 명령어 삽입 또는 크로스 사이트 스크립팅과 같은 공격 패턴을 포함할 수 있는 악성 트래픽도 필터링할 수 있습니다.

Network Load Balancer

Network Load Balancer는 오픈 시스템 상호 연결(OSI) 모델의 네 번째 계층에서 작동합니다. 비 HTTP 프로토콜 또는 종단 간 암호화가 필요한 시나리오에 적합하지만, Application Load Balancer에서 제공하는 동일한 HTTP 특정 기능을 제공하지는 않습니다. 따라서 Network Load Balancer는 HTTP를 사용하지 않는 애플리케이션에 가장 적합합니다. 자세한 내용은 Network Load Balancer 사용 설명서의 Network Load Balancer란?을 참조하세요.



Network Load Balancer가 입력으로 사용되는 경우 Application Load Balancer와 유사하게 작동합니다. 퍼블릭 서브넷에서 생성되고 인터넷에서 액세스할 수 있는 퍼블릭 IP 주소를 사용하기 때문입니다. 그러면 Network Load Balancer는 컨테이너를 실행하는 호스트의 프라이빗 IP 주소에 대한 연결을 열고 퍼블릭 측에서 프라이빗 측으로 패킷을 전송합니다.

Network Load Balancer 기능

Network Load Balancer는 네트워킹 스택의 하위 수준에서 작동하기 때문에 Application Load Balancer와 동일한 기능 세트를 제공하지 않습니다. 하지만 다음과 같은 중요한 기능이 있습니다.

- 종단 간 암호화 - Network Load Balancer는 OSI 모델의 네 번째 계층에서 작동하므로 패킷 콘텐츠를 읽지 않습니다. 따라서 종단 간 암호화가 필요한 로드 밸런싱 통신에 적합합니다.
- TLS 암호화 - Network Load Balancer는 종단 간 암호화 외에도 TLS 연결을 종료할 수 있습니다. 따라서 백엔드 애플리케이션에서 자체 TLS를 구현할 필요가 없습니다.
- UDP 지원 - Network Load Balancer는 OSI 모델의 네 번째 계층에서 작동하므로 TCP 이외의 비 HTTP 워크로드 및 프로토콜에 적합합니다.

연결 닫기

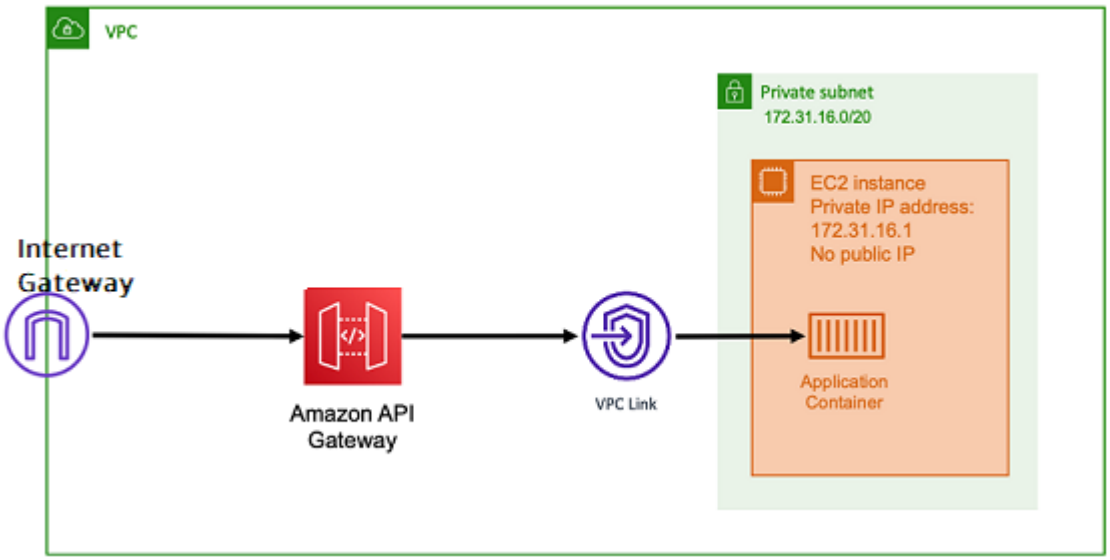
Network Load Balancer는 OSI 모델의 상위 계층에서 애플리케이션 프로토콜을 관찰하지 않으므로 해당 프로토콜에서 클라이언트에 닫기 메시지를 보낼 수 없습니다. Application Load Balancer와 달리, 이러한 연결은 애플리케이션이 닫아야 합니다. 아니면 작업이 중지되거나 교체될 때 네 번째 계층 연결을 닫도록 Network Load Balancer를 구성할 수 있습니다. Network Load Balancer 설명서의 Network Load Balancer 대상 그룹에 대한 연결 종료 설정을 참조하세요.

Network Load Balancer에서 네 번째 계층의 연결을 닫으면 클라이언트가 원하지 않는 오류 메시지를 처리하지 않는 경우 클라이언트에 해당 메시지가 표시될 수 있습니다. 권장 클라이언트 구성에 대한 자세한 정보는 Builders' Library를 참조하세요.

연결을 종료하는 방법은 애플리케이션마다 다르지만, Network Load Balancer 대상 등록 취소 지연이 클라이언트 연결 제한 시간보다 길도록 보장하는 한 가지 방법이 있습니다. 이전 작업이 모든 클라이언트를 천천히 드레이닝하는 동안, 클라이언트에서 먼저 제한 시간이 초과되고 Network Load Balancer를 통해 다음 작업에 정상적으로 다시 연결합니다. Network Load Balancer 대상 등록 취소 지연에 대한 자세한 내용은 Network Load Balancer 설명서를 참조하세요.

Amazon API Gateway HTTP API

Amazon API Gateway는 요청 볼륨이 갑자기 증가하거나 요청 볼륨이 적은 HTTP 애플리케이션에 적합합니다. 자세한 내용은 API Gateway 개발자 안내서의 What is Amazon API Gateway?를 참조하세요.



Application Load Balancer 및 Network Load Balancer의 요금 모델에는 로드 밸런서에서 수신 연결을 항상 수락하도록 유지하는 시간당 요금이 포함되어 있습니다. 반면, API Gateway는 각 요청에 대해 개별적으로 요금을 청구합니다. 따라서 요청이 들어오지 않으면 요금이 부과되지 않습니다. 트래픽 부하가 높은 경우 Application Load Balancer 또는 Network Load Balancer는 API Gateway보다 저렴한 요청당 요금으로 더 많은 양의 요청을 처리할 수 있습니다. 하지만 전반적으로 요청 수가 적거나 일정 기간 트래픽이 적은 경우 사용률이 낮은 로드 밸런서를 유지 관리하기 위해 시간당 요금을 지불하는 것보다 API Gateway를 사용하는 누적 요금이 더 비용 효율적입니다. 또한 API Gateway는 API 응답을 캐싱할 수 있으므로 백엔드 요청 비율이 낮아질 수 있습니다.

API Gateway는 VPC 링크를 사용하여 작동합니다. 이를 통해 AWS 관리형 서비스에서 프라이빗 IP 주소를 사용해 VPC의 프라이빗 서브넷 내에 있는 호스트에 연결할 수 있습니다. Amazon ECS 서비스 검색에서 관리하는 AWS Cloud Map 서비스 검색 레코드를 살펴봄으로써 이러한 프라이빗 IP 주소를 감지할 수 있습니다.

API Gateway에서 지원하는 기능은 다음과 같습니다.

- API Gateway 작업은 로드 밸런서와 비슷하지만, API 관리와 관련된 고유한 추가 기능이 있습니다.
- API Gateway에서는 클라이언트 권한 부여, 사용 계층, 요청 및 응답 수정과 관련된 추가 기능을 제공합니다. 자세한 내용은 Amazon API Gateway 기능을 참조하세요.
- API Gateway는 엣지, 리전 및 프라이빗 API 게이트웨이 엔드포인트를 지원할 수 있습니다. 엣지 엔드포인트는 관리형 CloudFront 배포를 통해 사용할 수 있습니다. 리전 엔드포인트 및 프라이빗 엔드포인트는 모두 리전에 로컬로 존재합니다.
- SSL/TLS 종료
- 다양한 HTTP 경로를 다양한 백엔드 마이크로서비스로 라우팅

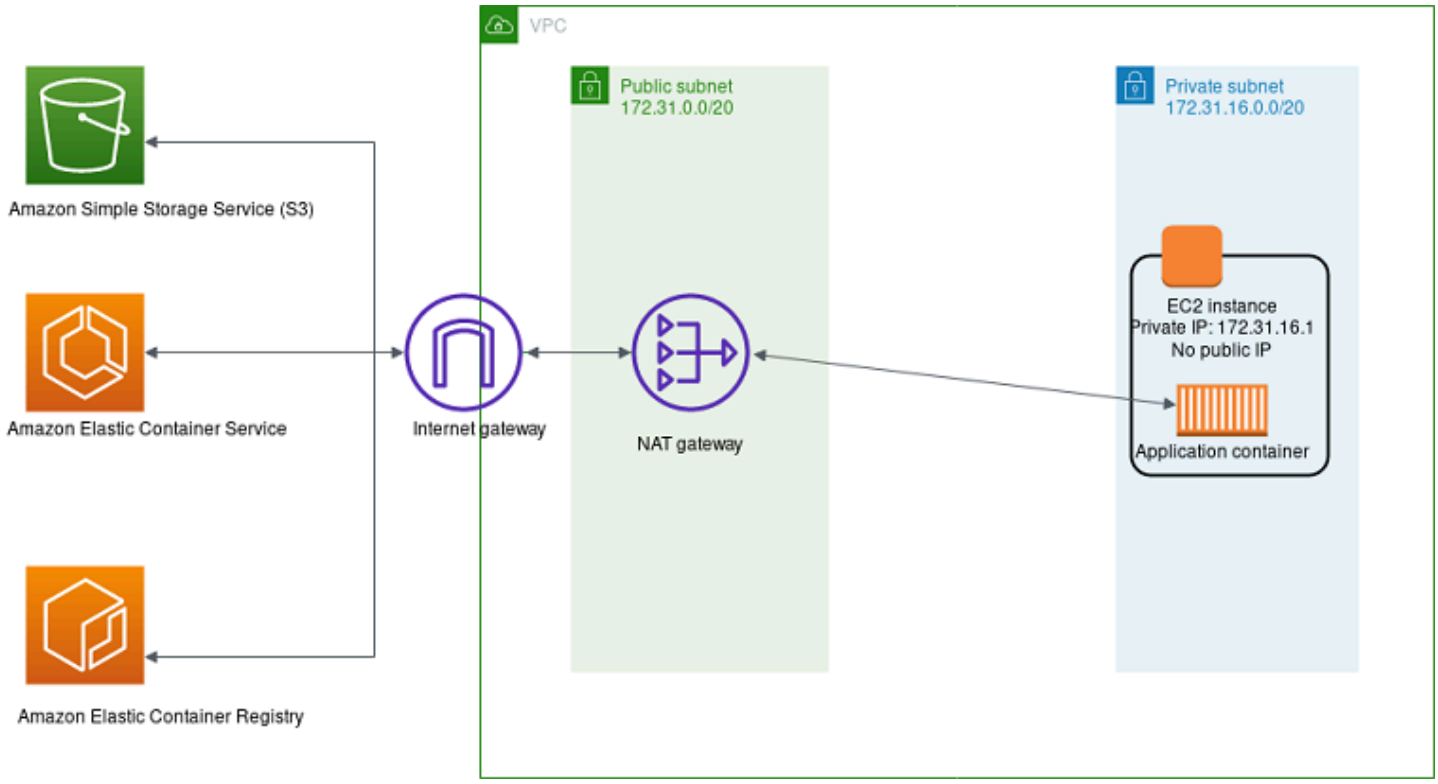
위의 기능 외에도 API Gateway는 API를 무단 사용으로부터 보호하기 위해 사용할 수 있는 사용자 지정 Lambda 권한 부여자 사용을 지원합니다. 자세한 내용은 Field Notes: Serverless Container-based APIs with Amazon ECS and Amazon API Gateway를 참조하세요.

VPC 내에서 Amazon ECS를 AWS 서비스에 연결하는 모범 사례

Amazon ECS가 올바르게 동작하려면 각 호스트에서 실행되는 Amazon ECS 컨테이너 에이전트가 Amazon ECS 컨트롤 플레인과 통신해야 합니다. Amazon ECR에 컨테이너 이미지를 저장하는 경우 Amazon EC2 호스트는 Amazon ECR 서비스 엔드포인트 및 이미지 레이어가 저장된 Amazon S3와 통신해야 합니다. DynamoDB에 저장된 데이터를 유지하는 등 컨테이너화된 애플리케이션에 다른 AWS 서비스를 사용하는 경우 이러한 서비스에도 필요한 네트워킹 지원이 있는지 다시 한 번 확인하세요.

NAT 게이트웨이

Amazon ECS 태스크가 다른 AWS 서비스에 액세스할 수 있도록 보장하는 가장 쉬운 방법은 NAT 게이트웨이를 사용하는 것입니다. 이 방법에 대한 자세한 내용은 프라이빗 서브넷 및 NAT 게이트웨이 섹션을 참조하세요.



이 접근 방식을 사용하면 다음과 같은 단점이 있습니다.

- NAT 게이트웨이가 통신할 수 있는 대상을 제한할 수 없습니다. 또한 백엔드 계층이 VPC의 모든 아웃바운드 통신을 중단하지 않으면서 통신할 수 있는 대상을 제한할 수 없습니다.

- NAT 게이트웨이는 통과하는 데이터 GB당 요금을 부과합니다. 다음 작업에 NAT 게이트웨이를 사용하는 경우 대역폭 GB당 요금이 부과됩니다.
  - Amazon S3에서 대용량 파일 다운로드
  - DynamoDB에서 대량의 데이터베이스 쿼리 수행
  - Amazon ECR에서 이미지 가져오기

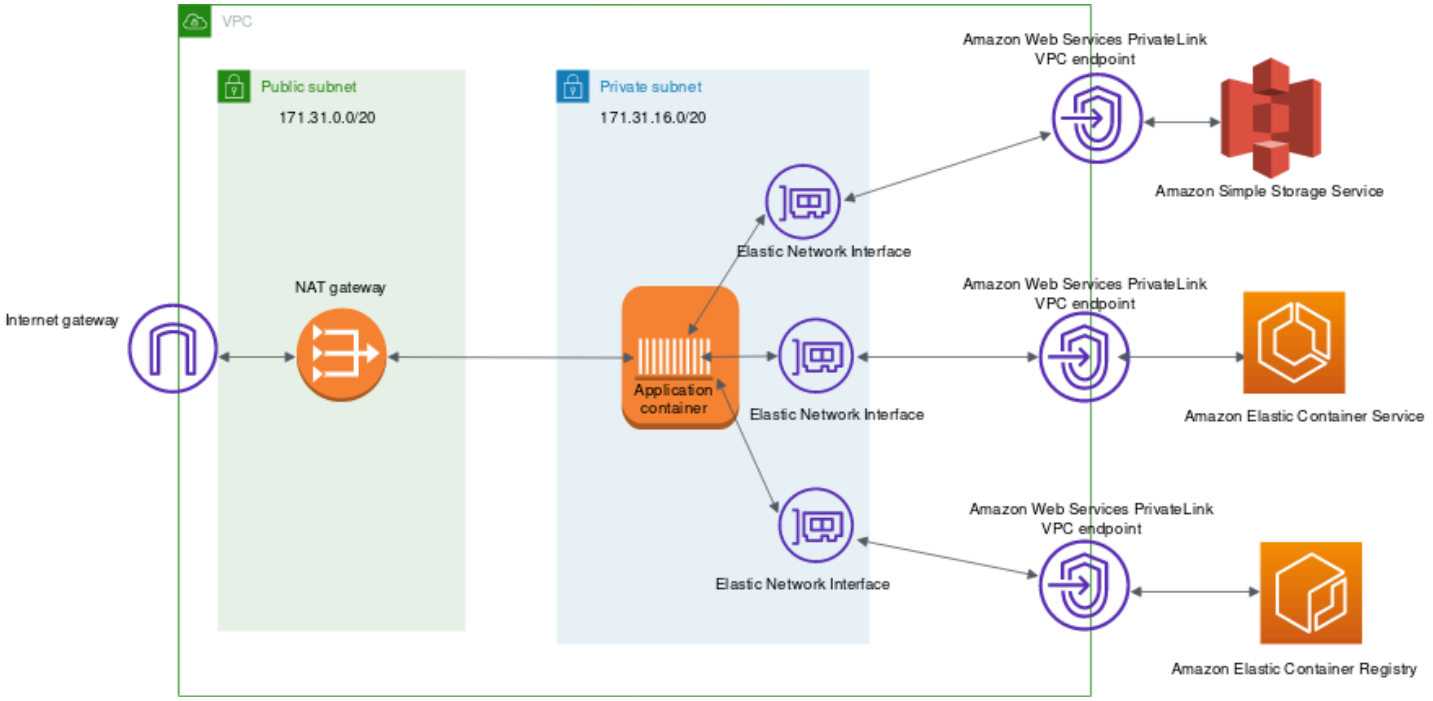
  또한 NAT 게이트웨이는 5Gbps의 대역폭을 지원하며 최대 45Gbps까지 자동 확장합니다. 단일 NAT 게이트웨이를 통해 라우팅하는 경우 매우 높은 대역폭 연결이 필요한 애플리케이션에 네트워킹 제약 조건이 발생할 수 있습니다. 해결 방법은 워크로드를 여러 서브넷으로 나누고 각 서브넷에 고유한 NAT 게이트웨이를 제공하는 것입니다.

AWS PrivateLink

AWS PrivateLink는 트래픽을 퍼블릭 인터넷에 노출시키지 않고 VPC, AWS 서비스, 온프레미스 네트워크 간에서 프라이빗 연결을 제공합니다.

VPC 엔드포인트에서는 VPC와 지원되는 AWS 서비스 및 VPC 엔드포인트 서비스 간에 프라이빗 연결을 사용할 수 있습니다. VPC와 기타 서비스 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다. VPC 엔드포인트에는 인터넷 게이트웨이, 가상 프라이빗 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결이 필요하지 않습니다. VPC의 Amazon EC2 인스턴스는 서비스의 리소스와 통신하는 데 퍼블릭 IP 주소를 필요로 하지 않습니다.

다음 다이어그램에서는 인터넷 게이트웨이 대신 VPC 엔드포인트를 사용할 때 AWS 서비스와의 통신이 작동하는 방식을 보여줍니다. AWS PrivateLink는 서브넷 내부에 탄력적 네트워크 인터페이스(ENI)를 프로비저닝하고, VPC 라우팅 규칙을 사용하여 ENI를 통해 서비스 호스트 이름에 대한 모든 통신을 대상 AWS 서비스로 직접 전송합니다. 이 트래픽은 더 이상 NAT 게이트웨이 또는 인터넷 게이트웨이를 사용할 필요가 없습니다.



다음은 Amazon ECS 서비스와 함께 사용되는 몇 가지 일반적인 VPC 엔드포인트의 예입니다.

- Amazon S3에 대한 게이트웨이 엔드포인트
- DynamoDB VPC 엔드포인트
- Amazon ECS VPC 엔드포인트
- Amazon ECR VPC 엔드포인트

다른 많은 AWS 서비스가 VPC 엔드포인트를 지원합니다. 사용량이 매우 많은 AWS 서비스가 있는 경우 해당 서비스와 관련된 설명서와 해당 트래픽에 대한 VPC 엔드포인트를 생성하는 방법을 살펴봐야 합니다.

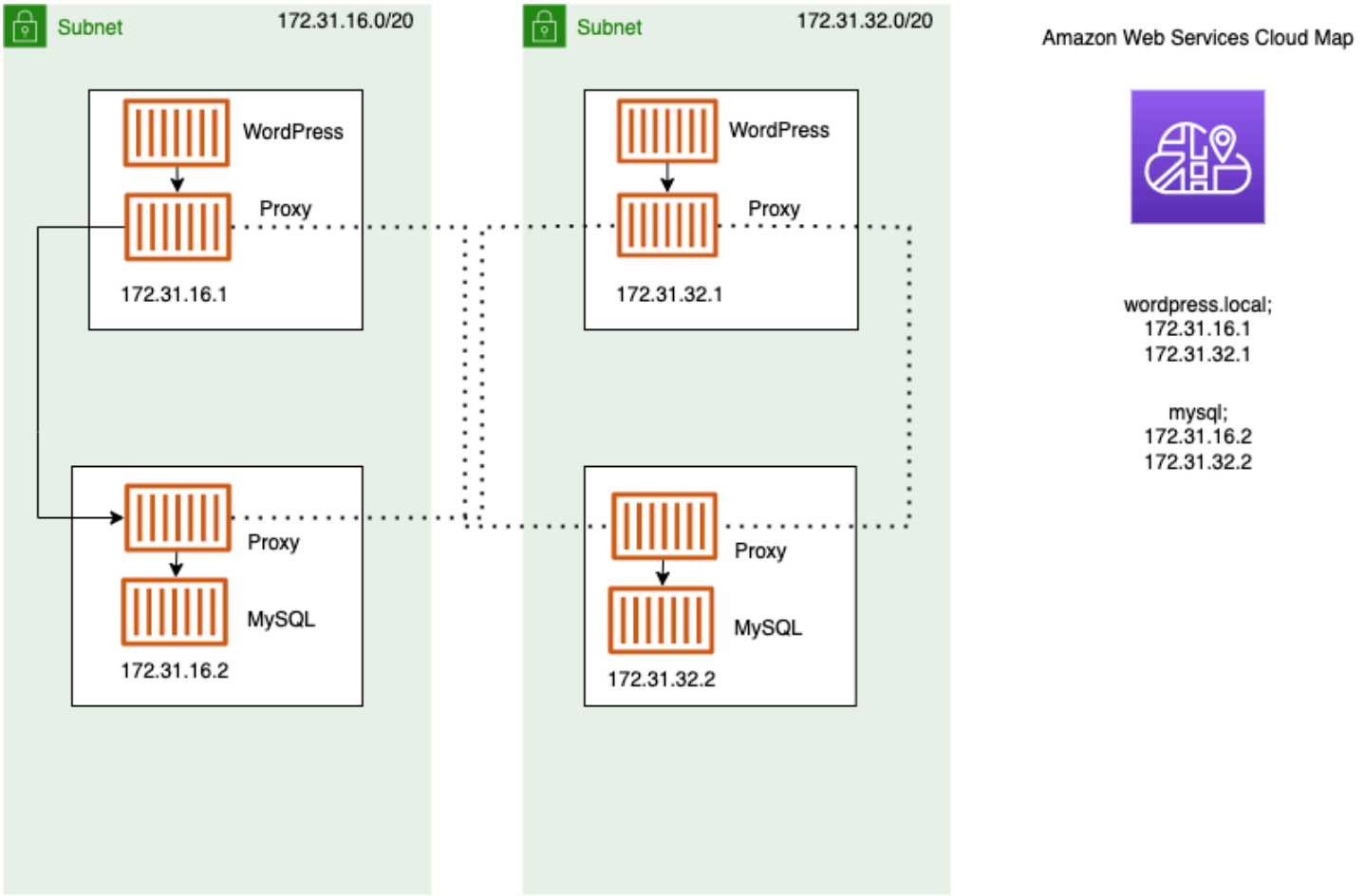
VPC에서 Amazon ECS 서비스를 연결하는 모범 사례

VPC에서 Amazon ECS 태스크를 사용할 경우 모놀리식 애플리케이션을 별도의 부분으로 분할하여 안전한 환경에서 독립적으로 배포하고 확장할 수 있습니다. 이 아키텍처를 서비스 지향 아키텍처(SOA) 또는 마이크로서비스라고 합니다. 하지만 VPC 내부와 외부 모두에서 이러한 모든 부분이 서로 통신할 수 있는지 확인하는 것이 어려울 수 있습니다. 원활한 통신을 보장하는 방법에는 여러 가지가 있지만 각각 장단점이 있습니다.

Service Connect 사용

서비스 검색, 연결 및 트래픽 모니터링을 위한 Amazon ECS 구성을 제공하는 Service Connect를 사용하는 것이 좋습니다. Service Connect를 사용하면 애플리케이션에서 짧은 이름과 표준 포트를 사용하여 동일한 리전에 있는 여러 VCP에 있는 항목을 포함하여 동일한 클러스터, 다른 클러스터의 서비스에 연결할 수 있습니다. 자세한 내용은 Amazon ECS Service Connect를 참조하세요.

Service Connect를 사용하면 Amazon ECS가 서비스 검색의 모든 부분을 관리합니다. 즉, 검색할 수 있는 이름을 생성하고, 작업이 시작 및 중지될 때 각 작업에 대한 항목을 동적으로 관리하며, 이름을 검색하도록 구성된 각 작업에서 에이전트를 실행할 수 있습니다. 애플리케이션은 DNS 이름에 대한 표준 기능을 사용하고 연결을 설정하여 이름을 조회할 수 있습니다. 애플리케이션이 이미 이 작업을 수행하는 경우 Service Connect를 사용하기 위해 애플리케이션을 수정할 필요가 없습니다.



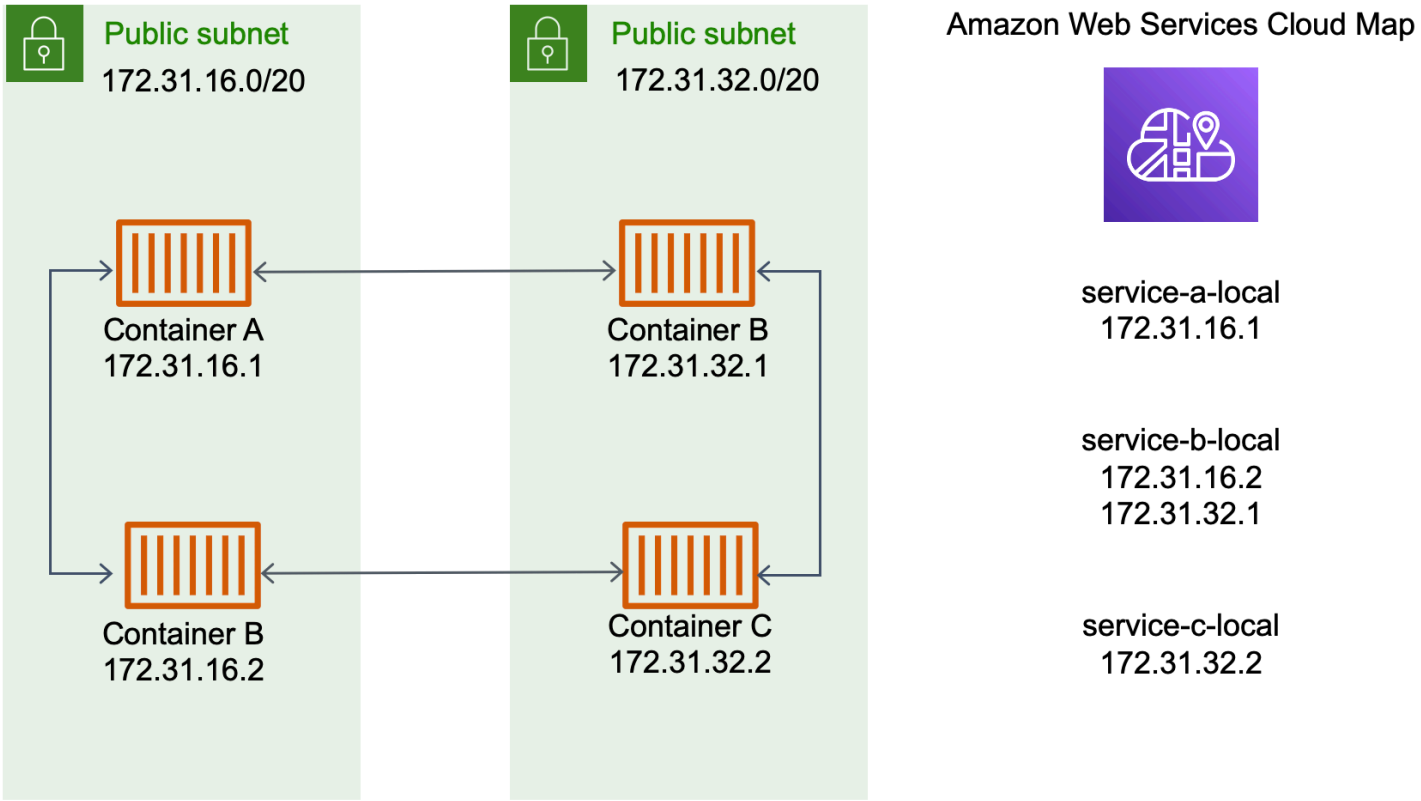
변경 사항은 배포 중에만 발생

각 서비스 및 작업 정의 내에 전체 구성을 제공합니다. Amazon ECS는 각 서비스 배포에서 이 구성의 변경 내용을 관리하여 배포의 모든 작업이 동일한 방식으로 작동하도록 합니다. 예를 들어, 서비스 검색으로 DNS에서 흔히 발생하는 문제는 마이그레이션 제어와 관련됩니다. 새 교체 IP 주소를 가리키도

록 DNS 이름을 변경하는 경우 모든 클라이언트가 새 서비스를 사용하기 시작하는 데 최대 TTL 시간이 걸릴 수 있습니다. Service Connect를 사용하면 클라이언트 배포에서 클라이언트 작업을 교체하여 구성을 업데이트합니다. 다른 배포와 동일한 방식으로 Service Connect 변경 내용에 영향을 주도록 배포 회로 차단기 및 기타 배포 구성을 구성할 수 있습니다.

서비스 검색 사용

서비스 간 통신의 또 다른 접근 방식으로, 서비스 검색을 사용하는 직접 통신이 있습니다. 이 접근 방식에서는 Amazon ECS와의 AWS Cloud Map 서비스 검색 통합을 사용할 수 있습니다. Amazon ECS는 서비스 검색을 사용하여 시작된 작업 목록을 AWS Cloud Map과 동기화합니다. 이 제품에서는 해당 특정 서비스에서 하나 이상 작업에 대한 내부 IP 주소로 확인되는 DNS 호스트 이름을 유지 관리합니다. Amazon VPC의 다른 서비스는 이 DNS 호스트 이름을 사용하여 내부 IP 주소를 통해 다른 컨테이너로 직접 트래픽을 보낼 수 있습니다. 자세한 내용은 서비스 검색을 참조하세요.



이전 다이어그램에는 세 가지 서비스가 있습니다. `service-a-local`에는 컨테이너가 하나 있고 컨테이너가 두 개 있는 `service-b-local`과 통신합니다. `service-b-local`도 컨테이너가 하나 있는 `service-c-local`과 통신해야 합니다. 세 서비스 모두에서 각 컨테이너는 AWS Cloud Map의 내부 DNS 이름을 사용하여 통신이 필요한 다운스트림 서비스의 컨테이너 내부 IP 주소를 찾을 수 있습니다.

서비스 간 통신에 대한 이러한 접근 방식은 지연 시간을 줄여줍니다. 컨테이너 사이에 추가 구성 요소가 없기 때문에 언뜻 보기에도 간단합니다. 트래픽은 한 컨테이너에서 다른 컨테이너로 직접 이동합니다.

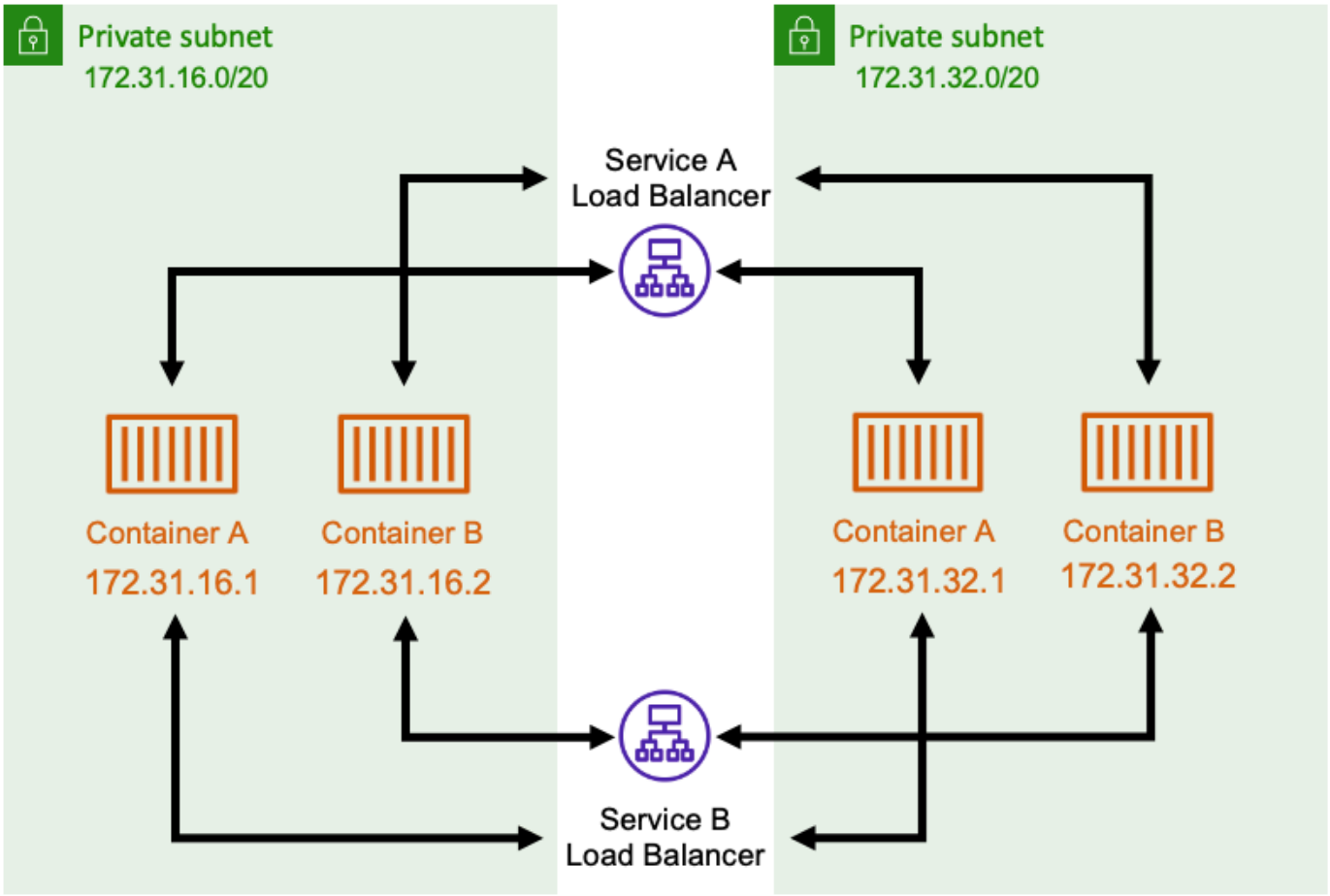
이 접근 방식은 각 작업의 IP 주소가 고유한 awsvpc 네트워크 모드를 사용할 때 적합합니다. 대부분의 소프트웨어는 IP 주소로 직접 확인되는 DNS A 레코드 사용만 지원합니다. awsvpc 네트워크 모드를 사용하는 경우 각 작업의 IP 주소는 A 레코드입니다. 하지만 bridge 네트워크 모드를 사용하는 경우 여러 컨테이너가 동일한 IP 주소를 공유할 수 있습니다. 또한 동적 포트 매핑으로 해당 단일 IP 주소에서 컨테이너에 포트 번호가 무작위로 할당됩니다. 이 경우 A 레코드는 더 이상 서비스 검색에 충분하지 않습니다. 또한 SRV 레코드를 사용해야 합니다. 이 유형의 레코드는 IP 주소와 포트 번호를 모두 추적할 수 있지만 애플리케이션을 적절하게 구성해야 합니다. 사용하는 일부 사전 구축된 애플리케이션은 SRV 레코드를 지원하지 않을 수 있습니다.

awsvpc 네트워크 모드의 또 다른 이점은 각 서비스에 대해 고유한 보안 그룹이 보유한다는 점입니다. 해당 서비스와 통신해야 하는 특정 업스트림 서비스의 수신 연결만 허용하도록 이 보안 그룹을 구성할 수 있습니다.

서비스 검색을 사용하는 서비스 간 직접 통신의 주된 단점은 재시도 및 연결 실패 처리를 위해 추가 로직을 구현해야 한다는 점입니다. DNS 레코드에는 캐싱되는 시간을 제어하는 TTL(Time To Live) 기간이 있습니다. 애플리케이션이 최신 버전의 DNS 레코드를 선택할 수 있도록 DNS 레코드를 업데이트하고 캐시가 만료되기까지 어느 정도 시간이 걸립니다. 따라서 애플리케이션이 DNS 레코드를 확인할 때 더 이상 존재하지 않는 다른 컨테이너를 가리킬 수 있습니다. 애플리케이션은 재시도를 처리하고 잘못된 백엔드를 무시할 수 있는 로직을 포함해야 합니다.

내부 로드 밸런서 사용

서비스 간 통신에 대한 또 다른 접근 방식은 내부 로드 밸런서를 사용하는 것입니다. 내부 로드 밸런서는 VPC 내부에만 존재하며 VPC 내부의 서비스에만 액세스할 수 있습니다.



로드 밸런서는 각 서브넷에 중복 리소스를 배포하여 고가용성을 유지합니다. serviceA의 컨테이너가 serviceB의 컨테이너와 통신해야 하는 경우 로드 밸런서에 대한 연결을 엽니다. 그러면 로드 밸런서가 service B의 컨테이너에 대한 연결을 엽니다. 로드 밸런서는 각 서비스 간의 모든 연결을 관리하는 중앙 위치의 역할을 합니다.

serviceB의 컨테이너가 중지되면 로드 밸런서가 해당 컨테이너를 풀에서 제거할 수 있습니다. 또한 로드 밸런서는 풀의 각 다운스트림 대상에 대해 상태 확인을 수행하여 잘못된 대상이 다시 정상 상태가 될 때까지 풀에서 자동으로 제거할 수 있습니다. 애플리케이션은 더 이상 다운스트림 컨테이너가 몇 개 있는지 알 필요가 없습니다. 단순히 로드 밸런서에 대한 연결을 열면 됩니다.

이 접근 방식은 모든 네트워크 모드에서 이점이 있습니다. 로드 밸런서는 awsvpc 네트워크 모드를 사용할 때는 태스크 IP 주소를 추적하고 bridge 네트워크 모드를 사용할 때는 더 복잡한 IP 주소 및 포트 조합을 추적할 수 있습니다. 로드 밸런서는 모든 IP 주소 및 포트 조합 간에 트래픽을 균등하게 분배합니다. 이는 여러 컨테이너가 실제로 동일한 Amazon EC2 인스턴스에서 호스팅되고 포트만 다른 경우에도 마찬가지입니다.

이 접근 방식의 한 가지 단점은 비용입니다. 고가용성을 보장하려면 로드 밸런서의 각 가용 영역에 리소스가 있어야 합니다. 따라서 로드 밸런서에 대한 비용과 로드 밸런서를 통과하는 트래픽 양에 대한 비용을 계산하는 오버헤드 때문에 추가 비용이 발생합니다.

하지만 여러 서비스가 로드 밸런서를 공유하도록 하면 오버헤드 비용을 줄일 수 있습니다. 특히 Application Load Balancer를 사용하는 REST 서비스에 적합합니다. 서로 다른 서비스로 트래픽을 라우팅하는 경로 기반 라우팅 규칙을 생성할 수 있습니다. 예를 들어, `/api/user/*` 규칙은 `user` 서비스의 일부인 컨테이너로 라우팅하고 `/api/order/*` 규칙은 연관된 `order` 서비스로 라우팅할 수 있습니다. 이 접근 방식을 사용하면 Application Load Balancer 하나에 대해서만 비용을 지불하고 API에 대해 하나의 일관된 URL을 갖게 됩니다. 하지만 백엔드의 다양한 마이크로서비스로 트래픽을 분산시킬 수 있습니다.

AWS 계정 및 VPC 간의 Amazon ECS 서비스 네트워킹을 위한 모범 사례

여러 팀과 부서가 있는 조직에 속한 경우 공유 AWS 계정 내의 개별 VPC나 여러 개인 AWS 계정과 연결된 VPC에 서비스를 독립적으로 배포할 가능성이 높습니다. 어떤 방식으로 서비스를 배포하든 VPC 간에 트래픽을 라우팅할 수 있도록 네트워킹 구성 요소를 보완하는 것이 좋습니다. 이를 위해 여러 AWS 서비스를 사용하여 기존 네트워킹 구성 요소를 보완할 수 있습니다.

- AWS Transit Gateway - 먼저 이 네트워킹 서비스를 고려해야 합니다. 이 서비스는 Amazon VPC, AWS 계정 및 온프레미스 네트워크 간의 연결을 라우팅하기 위한 중앙 허브 역할을 합니다. 자세한 내용은 Amazon VPC Transit Gateways 설명서의 전송 게이트웨이란 무엇입니까?를 참조하세요.
- Amazon VPC 및 VPN 지원 - 이 서비스를 사용하여 사이트 간 VPN 연결을 생성하여 온프레미스 네트워크를 VPC에 연결할 수 있습니다. 자세한 내용은 AWS Site-to-Site VPN 사용 설명서의 AWS Site-to-Site VPN이란 무엇입니까? 섹션을 참조하세요.
- Amazon VPC - Amazon VPC 피어링을 사용하면 동일한 계정 내에서 또는 여러 계정 간에서 여러 VPC를 연결할 수 있습니다. 자세한 내용은 Amazon VPC 피어링 가이드의 VPC 피어링이란? 섹션을 참조하세요.
- 공유 VPC - 여러 AWS 계정 간에서 VPC 및 VPC 서브넷을 사용할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 공유 VPC 작업을 참조하세요.

Amazon ECS 네트워킹 문제 해결을 위한 AWS 서비스

다음과 같은 서비스와 기능을 사용하면 네트워크 및 서비스 구성에 대한 통찰력을 얻을 수 있습니다. 이 정보를 사용하여 네트워킹 문제를 해결하고 서비스 최적화를 개선할 수 있습니다.

CloudWatch Container Insights

CloudWatch Container Insights는 컨테이너 애플리케이션 및 마이크로서비스의 지표 및 로그를 수집하고 종합하며 요약합니다. 이러한 지표에는 CPU, 메모리, 디스크, 네트워크 같은 리소스 사용률이 포함됩니다. 지표는 CloudWatch 자동 대시보드에서 사용할 수 있습니다. 자세한 정보는 Amazon CloudWatch 사용 설명서의 Amazon ECS에서 Container Insights 설정을 참조하세요.

AWS X-Ray

AWS X-Ray는 애플리케이션이 수행하는 네트워크 요청에 대한 정보를 수집하는 데 사용할 수 있는 추적 서비스입니다. SDK를 사용하여 애플리케이션을 계측하고 서비스 간, 서비스와 AWS 서비스 엔드포인트 간 트래픽의 타이밍과 응답 코드를 캡처할 수 있습니다. 자세한 내용은 AWS X-Ray 개발자 가이드의 AWS X-Ray이란 무엇입니까?를 참조하세요.

또한 서비스가 다른 서비스와 네트워킹을 구성하는 방식을 AWS X-Ray 그래프로 살펴볼 수 있습니다. 이 기능을 사용하여 각 서비스 간 링크가 작동하는 방식에 대한 집계 통계를 탐색할 수도 있습니다. 마지막으로, 특정 트랜잭션을 자세히 살펴보고 네트워크 호출을 나타내는 세그먼트가 해당 특정 트랜잭션과 어떻게 연결되어 있는지 확인할 수 있습니다.

이러한 기능을 사용하여 네트워킹 병목 현상이 있는지 또는 네트워크 내의 특정 서비스가 예상과 다르게 작동하는지 파악할 수 있습니다.

VPC 흐름 로그

Amazon VPC 흐름 로그를 사용하여 네트워크 성능을 분석하고 연결 문제를 디버깅할 수 있습니다. VPC 흐름 로그를 활성화하면 VPC 내의 모든 연결 로그를 캡처할 수 있습니다. 여기에는 Elastic Load Balancing, Amazon RDS, NAT 게이트웨이 및 사용 중인 기타 주요 AWS 서비스와 관련된 네트워킹 인터페이스 연결이 포함됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 흐름 로그를 참조하세요.

네트워크 튜닝 팁

네트워킹을 개선하기 위해 세밀하게 튜닝할 수 있는 몇 가지 설정이 있습니다.

nofile ulimit

애플리케이션에서 트래픽이 많고 다수의 동시 연결을 처리해야 한다고 예상되는 경우 허용된 파일 수에 대한 시스템 할당량을 고려해야 합니다. 열려 있는 네트워크 소켓이 많은 경우 각 소켓을 파일 설명자로 표시해야 합니다. 파일 설명자 할당량이 너무 낮으면 네트워크 소켓이 제한됩니다. 이로 인해 연결에 실패하거나 오류가 발생합니다. Amazon ECS 태스크 정의에서 파일 수에 대한 컨테이너별 할

당량을 업데이트할 수 있습니다. Amazon EC2(AWS Fargate 대신)에서 실행 중인 경우 기본 Amazon EC2 인스턴스에서 이러한 할당량을 조정해야 할 수도 있습니다.

sysctl net

튜닝 가능한 설정의 또 다른 범주는 `sysctl` net 설정입니다. 구체적인 설정은 사용 중인 Linux 배포판을 참조해야 합니다. 이러한 설정 중 많은 수가 읽기 및 쓰기 버퍼의 크기를 조정합니다. 많은 수의 컨테이너를 포함하는 대규모 Amazon EC2 인스턴스를 실행하는 일부 상황에서 이 기능이 유용할 수 있습니다.

Amazon ECS 태스크 시작 시간 최적화

작업 시작 속도를 높이려면 다음 권장 사항을 고려합니다.

- 컨테이너 이미지 및 binpack 인스턴스 캐싱

  EC2 시작 유형을 사용하는 경우 Amazon ECS 컨테이너 에이전트 가져오기 동작을 `ECS_IMAGE_PULL_BEHAVIOR: prefer-cached`로 구성할 수 있습니다. 캐시된 이미지가 없으면 이미지를 원격에서 가져옵니다. 그렇지 않으면 인스턴스에 캐시된 이미지가 사용됩니다. 캐시된 이미지가 제거되지 않도록 컨테이너에서 자동 이미지 정리는 꺼져 있습니다. 그러면 후속 시작 시 이미지 가져오기 시간이 줄어듭니다. `binpack` 배치 전략을 사용하여 구성할 수 있는 컨테이너 인스턴스의 작업 밀도가 높으면 캐싱의 효과가 더욱 커집니다. 컨테이너 이미지 캐싱은 일반적으로 컨테이너 이미지 크기가 큰 Windows 기반 워크로드(수십 단위의 GB)에 특히 유용합니다. `binpack` 배치 전략을 사용할 때는 각 컨테이너 인스턴스에 `awsvpc` 네트워크 모드로 더 많은 작업을 배치하기 위해 탄력적 네트워크 인터페이스(ENI) 트렁킹을 사용하는 방법도 고려할 수 있습니다. ENI 트렁킹은 `awsvpc` 모드에서 실행할 수 있는 작업 수를 늘립니다. 예를 들어 2개의 작업 실행만 지원하는 c5.large 인스턴스는 ENI 트렁킹으로 최대 10개의 작업을 실행할 수 있습니다.
- 최적의 네트워크 모드 선택

  많은 인스턴스에서 `awsvpc` 네트워크 모드가 이상적이지만 이 네트워크 모드는 작업 시작 지연 시간을 본질적으로 증가시킬 수 있습니다. `awsvpc` 모드에서 각 작업에 대해 Amazon ECS 워크플로는 Amazon EC2 API를 간접 호출하여 ENI를 프로비저닝하고 연결해야 하기 때문에 작업 시작 시 몇 초의 오버헤드가 추가되기 때문입니다. 반면 `awsvpc` 네트워크 모드를 사용할 때 주요 이점은 각 작업에 트래픽을 허용하거나 거부하는 보안 그룹이 있다는 점입니다. 즉, 작업과 서비스 간 통신을 보다 세분화된 수준에서 제어할 수 있는 유연성이 향상됩니다. 배포 속도가 우선이라면 작업 시작 속도를 높이기 위해 `bridge` 모드를 사용하는 방법을 고려할 수 있습니다. 자세한 내용은 the section called “AWSVPC 네트워크 모드” 섹션을 참조하세요.
- 작업 시작 수명 주기를 추적하여 최적화 기회를 모색합니다.

애플리케이션을 시작하는 데 걸리는 시간을 파악하기란 쉽지 않습니다. 애플리케이션 시작 중에 컨테이너 이미지를 시작하고, 시작 스크립트를 실행하며, 기타 구성을 실행하는 데 매우 많은 시간이 걸릴 수 있습니다. 태스크 메타데이터 엔드포인트를 사용하여 지표를 게시하여 컨테이너 메타데이터 응답의 `StartedAt`에서 태스크 또는 서비스의 `StartedAt` 시간까지 애플리케이션 시작 시간을 추적할 수 있습니다. 이 데이터를 통해 애플리케이션이 총 시작 시간에 미치는 영향을 파악하고, 불필요한 애플리케이션별 오버헤드를 줄이며 컨테이너 이미지를 최적화할 수 있는 영역을 찾을 수 있습니다. 자세한 내용은 Amazon ECS 오토 스케일링 및 용량 관리 모범 사례 섹션을 참조하세요.

- 최적의 인스턴스 유형 선택(EC2 시작 유형의 경우)

  올바른 인스턴스 유형 선택은 작업에서 구성한 리소스 예약(예: CPU, 메모리)을 기반으로 합니다. 따라서 인스턴스 규모를 조정할 때 단일 인스턴스에 배치할 수 있는 작업 수를 계산할 수 있습니다. 잘 배치된 작업의 간단한 예제로, m5.large 인스턴스(2 vCPU 및 8GB 메모리 지원)에서 0.5 vCPU와 2GB의 메모리 예약이 필요한 4개의 작업을 호스팅할 수 있습니다. 이 작업 정의를 예약하면 인스턴스의 리소스를 최대한 활용할 수 있습니다.

대규모 Amazon ECS 운영

Amazon ECS를 대규모로 운영하기 시작하는 경우 Amazon ECS 및 Amazon ECS와 통합되는 AWS 서비스에 대한 서비스 할당량 및 API 스로틀이 사용자에게 미치는 영향을 고려하세요.

주제

- Amazon ECS 서비스 할당량 및 API 스로틀링 제한
- Amazon ECS 제한 문제 처리

Amazon ECS 서비스 할당량 및 API 스로틀링 제한

Amazon ECS는 Elastic Load Balancing, AWS Cloud Map, Amazon EC2를 비롯한 여러 AWS 서비스에 통합됩니다. Amazon ECS는 이러한 긴밀한 통합을 통해 서비스 로드 밸런싱, 서비스 검색, 태스크 네트워킹, 클러스터 오토 스케일링 등의 여러 기능을 포함합니다. Amazon ECS 및 통합되는 기타 AWS 서비스는 모두 서비스 할당량 및 API 속도 제한을 유지 관리하여 일관된 성능 및 사용률을 보장합니다. 또한 이러한 서비스 할당량은 필요한 수준을 초과하여 많은 리소스가 실수로 프로비저닝되는 것을 방지하고 청구 비용을 늘릴 수 있는 악의적인 행동으로부터 보호합니다.

서비스 할당량 및 AWS API 속도 제한을 숙지하면 예상치 못한 성능 저하에 대한 걱정 없이 워크로드 규모 조정을 계획할 수 있습니다. 자세한 내용은 Amazon ECS 서비스 할당량 및 Amazon ECS API에 대한 요청 스로틀링을 참조하세요.

Amazon ECS에서 워크로드 규모를 조정할 때 다음과 같은 서비스 할당량을 고려하세요. 서비스 할당량 증가를 요청하는 방법에 대한 지침은 AWS Management Console에서 Amazon ECS 및 AWS Fargate 서비스 할당량 관리를 참조하세요.

- AWS Fargate에는 각 AWS 리전에서 동시에 실행되는 작업 수를 제한하는 할당량이 있습니다. Amazon ECS에는 온디맨드 작업 및 Fargate 스팟 작업 모두에 대한 할당량이 있습니다. 각 서비스 할당량에는 Fargate에서 실행하는 모든 Amazon EKS 포드도 포함됩니다. Fargate 할당량에 대한 자세한 내용은 AWS Fargate용 Amazon Elastic Container Service 사용 설명서의 AWS Fargate 서비스 할당량을 참조하세요.
- Amazon EC2 인스턴스에서 실행되는 작업의 경우 각 클러스터에 등록할 수 있는 Amazon EC2 인스턴스의 최대 개수는 5,000개입니다. Auto Scaling 그룹 용량 공급자와 함께 Amazon ECS 클러스터 Auto Scaling을 사용하거나 클러스터의 Amazon EC2 인스턴스를 자체적으로 관리하는 경우 이 할당량으로 인해 배포 병목 현상이 발생할 수 있습니다. 더 많은 용량이 필요한 경우 클러스터를 생성하거나 서비스 할당량 증가를 요청할 수 있습니다.
- Auto Scaling 그룹 용량 공급자와 함께 Amazon ECS 클러스터 Auto Scaling을 사용하는 경우 서비스 규모를 확장할 때 `Tasks in the PROVISIONING state per cluster` 할당량을 고려합니다. 이 할당량은 각 클러스터에서 용량 공급자가 용량을 늘릴 수 있는 `PROVISIONING` 상태의 최대 작업 수입니다. 동시에 많은 작업을 시작하면 이 할당량을 쉽게 충족할 수 있습니다. 한 가지 예로, 각각 수백 개의 작업을 포함하는 수십 개의 서비스를 동시에 배포하는 경우가 있습니다. 이 경우 용량 공급자는 클러스터의 용량이 부족할 때 새 컨테이너 인스턴스를 시작하여 작업을 배치해야 합니다. 용량 공급자가 추가 Amazon EC2 인스턴스를 시작하는 동안에도 Amazon ECS 서비스 스케줄러는 작업을 계속 병렬로 시작할 수 있습니다. 하지만 클러스터 용량이 충분하지 않아 이 활동이 제한될 수 있습니다. Amazon ECS 서비스 스케줄러는 새 컨테이너 인스턴스가 시작될 때 작업 배치를 재시도하기 위한 백오프 및 지수 제한 전략을 구현합니다. 따라서 배포 또는 스케일 아웃 시간이 느려질 수 있습니다. 이러한 상황을 방지하려면 다음 방식 중 하나로 서비스 배포를 계획할 수 있습니다. 클러스터 용량을 늘릴 필요 없이 많은 태스크를 배포하거나 새 태스크를 시작할 수 있도록 예비 클러스터 용량을 확보합니다.

워크로드 규모를 조정할 때 Amazon ECS 서비스 할당량을 고려하는 것 외에도 Amazon ECS와 통합된 다른 AWS 서비스 서비스의 서비스 할당량도 고려합니다. 다음 섹션에서는 각 서비스의 주요 속도 제한을 자세히 설명하고 잠재적인 스로틀링 문제를 해결하기 위한 권장 사항을 제공합니다.

Elastic Load Balancing

Elastic Load Balancing을 사용하여 작업 전체에 균일하게 트래픽 분산하도록 Amazon ECS 서비스를 구성할 수 있습니다. 로드 밸런서를 선택하는 방법에 대한 자세한 내용은 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산을 참조하세요.

Elastic Load Balancing 서비스 할당량

워크로드 규모를 조정할 때는 다음과 같은 Elastic Load Balancing 서비스 할당량을 고려하세요. 대부분의 Elastic Load Balancing 서비스 할당량은 조정 가능하며 Service Quotas 콘솔에서 증가를 요청할 수 있습니다.

Application Load Balancer

Application Load Balancer를 사용할 때는 사용 사례에 따라 다음에 대한 할당량 증가를 요청해야 할 수 있습니다.

- `Targets per Application Load Balancer` 할당량: Application Load Balancer 뒤에 있는 대상의 수입니다.
- `Targets per Target Group per Region` 할당량: 대상 그룹 뒤에 있는 대상의 수입니다.

자세한 내용은 Application Load Balancer에 대한 할당량을 참조하세요.

Network Load Balancer

Network Load Balancer에 등록할 수 있는 대상 수에는 더 엄격한 제한이 적용됩니다. Network Load Balancer를 사용할 때 종종 교차 영역 지원을 활성화하려고 합니다. 이 경우 각 Network Load Balancer의 가용 영역당 최대 대상 수와 함께 `Targets per Availability Zone Per Network Load Balancer`의 추가 조정 제한이 적용됩니다. 자세한 내용은 Network Load Balancer 할당량을 참조하세요.

Elastic Load Balancing API 제한

Amazon ECS 서비스가 로드 밸런서를 사용하도록 구성하는 경우 대상 그룹 상태 확인을 통과해야 서비스가 정상으로 간주됩니다. 이러한 상태 확인을 수행하기 위해 Amazon ECS는 사용자를 대신하여 Elastic Load Balancing API 작업을 간접 호출합니다. 계정에 로드 밸런서로 구성된 서비스가 많으면 특히 `RegisterTarget`, `DeregisterTarget`, `DescribeTargetHealth` Elastic Load Balancing API 작업에서 스로틀링이 발생할 수 있으므로 서비스 배포 속도가 느려질 수 있습니다. 제한이 발생하면 Amazon ECS 서비스 이벤트 메시지에서 제한 오류가 발생합니다.

AWS Cloud Map API 제한이 발생하는 경우 지원에 AWS Cloud Map API 제한 한도 상향에 지침을 문의할 수 있습니다. 스로틀링 오류와 같은 문제의 모니터링 및 문제 해결에 대한 자세한 내용은 스로틀링 문제 처리를 참조하세요.

탄력적 네트워크 인터페이스

태스크에서 awsvpc 네트워크 모드를 사용하는 경우, Amazon ECS는 각 태스크에 대해 고유한 탄력적 네트워크 인터페이스(ENI)를 프로비저닝합니다. Amazon ECS 서비스가 Elastic Load Balancing 로드 밸런서를 사용하는 경우 이러한 네트워크 인터페이스도 서비스에 정의된 적절한 대상 그룹에 대한 대상으로 등록됩니다.

탄력적 네트워크 인터페이스 서비스 할당량

awsvpc 네트워크 모드를 사용하는 작업을 실행하면 고유한 탄력적 네트워크 인터페이스가 각 작업에 연결됩니다. 인터넷을 통해 이러한 작업에 도달해야 할 경우에는 해당 작업의 탄력적 네트워크 인터페이스에 퍼블릭 IP 주소를 할당합니다. Amazon ECS 워크로드 규모를 조정할 때는 다음과 같은 두 가지 중요한 할당량을 고려합니다.

- `Network interfaces per Region` 할당량: 계정의 AWS 리전에 있는 최대 네트워크 인터페이스 수입니다.
- `Elastic IP addresses per Region` 할당량: AWS 리전에 있는 최대 탄력적 IP 주소 수입니다.

이 두 서비스 할당량은 모두 조정 가능하며, Service Quotas 콘솔에서 할당량 증가를 요청할 수 있습니다. 자세한 내용은 Amazon VPC 서비스 할당량을 참조하세요.

Amazon EC2 인스턴스에 호스팅되는 Amazon ECS 워크로드의 경우 awsvpc 네트워크 모드를 사용하는 작업을 실행할 때는 각 Amazon EC2 인스턴스의 최대 네트워크 인스턴스 수에 해당하는 `Maximum network interfaces` 서비스 할당량을 고려합니다. 이 할당량은 인스턴스에 배치할 수 있는 작업 수를 제한합니다. 할당량은 조정할 수 없으며 Service Quotas 콘솔에서 사용할 수 없습니다. 자세한 내용은 Amazon EC2 사용 설명서의 IP addresses per network interface per instance type를 참조하세요.

Amazon EC2 인스턴스에 연결할 수 있는 네트워크 인터페이스 수를 변경할 수는 없지만 탄력적 네트워크 인터페이스 트렁킹 기능을 사용하여 사용 가능한 네트워크 인터페이스 수를 늘릴 수 있습니다. 예를 들어 `c5.large` 인스턴스는 최대 3개의 네트워크 인터페이스를 보유할 수 있습니다. 인스턴스에 대한 기본 네트워크 인터페이스는 한 개로 계산됩니다. 따라서 인스턴스에 네트워크 인터페이스를 2개 더 연결할 수 있습니다. awsvpc 네트워크 모드를 사용하는 각 작업에 네트워크 인터페이스가 필요하므로 대개 이 인스턴스 유형에서는 이러한 작업을 2개만 실행할 수 있습니다. 이로 인해 클러스터 용량 사용률이 낮아질 수 있습니다. 탄력적 네트워크 인터페이스 트렁킹을 활성화하면 네트워크 인터페이스 밀도를 높여 각 인스턴스에 더 많은 작업을 배치할 수 있습니다. 트렁킹을 켜면 `c5.large` 인스턴스는 최대 12개의 네트워크 인터페이스를 보유할 수 있습니다. 인스턴스에는 기본 네트워크 인터페이스가 있으며, Amazon ECS는 '트렁크' 네트워크 인터페이스를 생성하고 컨테이너 인스턴스에 연결합

니다. 따라서 이 구성을 사용하면 인스턴스에서 기본 2개의 작업 대신 10개의 작업을 실행할 수 있습니다. 자세한 내용은 탄력적 네트워크 인터페이스를 참조하세요.

탄력적 네트워크 인터페이스 API 제한

awsvpc 네트워크 모드를 사용하는 작업을 실행할 때 Amazon ECS는 다음 Amazon EC2 API에 의존합니다. 이러한 API마다 API 제한이 서로 다릅니다. 자세한 내용은 Request throttling for the Amazon EC2 API를 참조하세요.

- CreateNetworkInterface
- AttachNetworkInterface
- DetachNetworkInterface
- DeleteNetworkInterface
- DescribeNetworkInterfaces
- DescribeVpcs
- DescribeSubnets
- DescribeSecurityGroups
- DescribeInstances

탄력적 네트워크 인터페이스 프로비저닝 워크플로 중에 Amazon EC2 API 직접 호출이 제한되는 경우 Amazon ECS 서비스 스케줄러는 지수 백오프를 통해 자동으로 재시도합니다. 이러한 자동 사용 중지로 인해 때때로 태스크 시작이 지연되어 배포 속도가 느려질 수 있습니다. API 제한이 발생하면 서비스 이벤트 메시지에 `Operations are being throttled. Will try again later.` 메시지가 표시됩니다. Amazon EC2 API 제한을 일관되게 충족하는 경우 API 제한 한도 상향 방법에 대한 지침을 지원에 문의할 수 있습니다. 제한 오류의 모니터링 및 문제 해결에 대한 자세한 내용은 제한 문제 처리를 참조하세요.

AWS Cloud Map

Amazon ECS 서비스 검색은 AWS Cloud Map API를 사용하여 Amazon ECS 서비스에 대한 네임스페이스를 관리합니다. 서비스에 작업 수가 많은 경우 다음 권장 사항을 고려하세요. 자세한 내용은 Amazon ECS 서비스 검색 고려 사항을 참조하세요.

AWS Cloud Map service quotas

Amazon ECS 서비스가 서비스 검색을 사용하도록 구성된 경우 서비스의 최대 태스크 수에 해당하는 `Tasks per service` 할당량은 해당 서비스의 최대 인스턴스 수인 AWS Cloud Map `Instances`

per service 서비스 할당량의 영향을 받습니다. 특히 AWS Cloud Map 서비스 할당량은 실행할 수 있는 최대 태스크 수를 서비스당 1,000개 태스크로 줄입니다. AWS Cloud Map 할당량은 변경할 수 없습니다. 자세한 내용은 AWS Cloud Map 서비스 할당량을 참조하세요.

AWS Cloud Map API 조절

Amazon ECS는 사용자를 대신하여 ListInstances, GetInstancesHealthStatus, RegisterInstance 및 DeregisterInstance AWS Cloud Map API를 직접 호출합니다. 이를 통해 서비스 검색을 지원하고 작업을 시작할 때 상태 확인을 수행할 수 있습니다. 많은 작업에서 서비스 검색을 사용하는 여러 서비스를 동시에 배포하는 경우, 이로 인해 AWS Cloud Map API 제한 한도가 초과될 수 있습니다. 이 경우 Amazon ECS 서비스 이벤트 메시지에 Operations are being throttled. Will try again later 메시지가 표시되고, 배포 및 작업 시작 속도가 느려집니다. AWS Cloud Map에서는 이러한 API의 제한 한도를 문서화하지 않습니다. 이로 인해 제한이 발생하는 경우 지원에 API 제한 한도 상향에 지침을 문의할 수 있습니다. 스로틀링 오류와 같은 문제의 모니터링 및 문제 해결에 대한 자세한 권장 사항은 스로틀링 문제 처리를 참조하세요.

Amazon ECS 제한 문제 처리

제한 오류는 크게 동기 제한 및 비동기 제한이라는 두 가지 주요 범주로 분류됩니다.

동기 제한

동기 제한이 수행되면 Amazon ECS에서 즉시 오류 응답을 수신합니다. 이 범주는 일반적으로 작업을 실행하거나 서비스를 생성하는 동안 Amazon ECS API를 호출할 때 발생합니다. 관련된 제한 및 관련 제한 한도에 대한 자세한 내용은 Amazon ECS API에 대한 요청 제한을 참조하세요.

애플리케이션에서 API 요청을 시작하면(예를 들어, AWS CLI 또는 AWS SDK를 사용함) API 제한을 해결할 수 있습니다. 오류를 처리하도록 애플리케이션을 설계하거나 API 직접 호출에 대한 재시도 로직이 포함된 지수 백오프 및 지터 전략을 구현하여 이를 수행할 수 있습니다. 자세한 내용은 시간 제한, 재시도 및 지터를 사용한 백오프를 참조하세요.

AWS SDK를 사용하는 경우 자동 재시도 로직이 기본 제공되며 구성 가능합니다.

비동기 제한

비동기 제한은 Amazon ECS 또는 AWS CloudFormation에서 사용자를 대신하여 API를 직접 호출해 리소스를 프로비저닝하는 비동기 워크플로로 인해 발생합니다. Amazon ECS가 사용자를 대신하여 어떤 AWS API를 직접 호출하는지 파악하는 것이 중요합니다. 예를 들어 awsvpc 네트워크 모드를 사용하는 작업에 대해 CreateNetworkInterface API가 간접 호출되고 로드 밸런서에 등록된 작업의 상태 확인을 수행할 때 DescribeTargetHealth API가 간접 호출됩니다.

워크로드가 상당한 규모에 도달하면 이러한 API 작업이 제한될 수 있습니다. 즉, Amazon ECS에서 시행하는 제한 또는 호출되는 AWS 서비스를 위반할 수준으로 제한이 발생할 수 있습니다. 예를 들어, 각각 `awsvpc` 네트워크 모드를 사용하는 수백 개의 작업을 동시에 보유하는 수백 개의 서비스를 배포하는 경우 Amazon ECS는 Amazon EC2 API 작업(예: `CreateNetworkInterface`) 및 Elastic Load Balancing API 작업(예: `RegisterTarget` 또는 `DescribeTargetHealth`)을 간접 호출하여 각각 탄력적 네트워크 인터페이스와 로드 밸런서를 등록합니다. 이러한 API 직접 호출은 API 제한을 초과할 수 있으며 이 경우 제한 오류가 발생할 수 있습니다. 다음은 서비스 이벤트 메시지에 포함된 Elastic Load Balancing 제한 오류의 예제입니다.

```
{
   "userIdentity":{
      "arn":"arn:aws:sts::111122223333:assumed-role/AWSServiceRoleForECS/ecs-service-
scheduler",
      "eventTime":"2022-03-21T08:11:24Z",
      "eventSource":"elasticloadbalancing.amazonaws.com",
      "eventName":" DescribeTargetHealth ",
      "awsRegion":"us-east-1",
      "sourceIPAddress":"ecs.amazonaws.com",
      "userAgent":"ecs.amazonaws.com",
      "errorCode":"ThrottlingException",
      "errorMessage":"Rate exceeded",
      "eventID":"0aeb38fc-229b-4912-8b0d-2e8315193e9c"
   }
}
```

이러한 API 직접 호출이 계정의 다른 API 트래픽과 제한을 공유하는 경우 서비스 이벤트로 생성되더라도 모니터링이 어려울 수 있습니다.

제한 모니터링

제한되는 API 요청과 이러한 요청의 발행자를 식별하는 것이 중요합니다. 제한을 모니터링하고 CloudWatch, Amazon Athena, Amazon EventBridge와 통합되는 AWS CloudTrail을 사용할 수 있습니다. CloudWatch Logs로 특정 이벤트를 전송하도록 CloudTrail을 구성할 수 있습니다. CloudWatch Logs 로그 인사이트는 이벤트를 구문 분석하고 분석합니다. 이를 통해 직접 호출을 수행한 사용자 또는 IAM 역할, 수행된 API 직접 호출 수와 같은 제한 이벤트의 세부 정보를 식별합니다. 자세한 내용은 Monitoring CloudTrail log files with CloudWatch Logs를 참조하세요.

CloudWatch Logs Insights에 대한 자세한 내용 및 로그 파일을 쿼리하는 방법에 대한 지침은 Analyzing log data with CloudWatch Logs Insights를 참조하세요.

Amazon Athena를 사용하면 표준 SQL을 사용하여 쿼리를 생성하고 데이터를 분석할 수 있습니다. 예를 들어, Athena 테이블을 생성하여 CloudTrail 이벤트를 구문 분석할 수 있습니다. 자세한 내용은 Using the CloudTrail console to create an Athena table for CloudTrail logs를 참조하세요.

Athena 테이블을 생성한 후 다음과 같은 SQL 쿼리를 사용하여 `ThrottlingException` 오류를 조사할 수 있습니다.

모든 *### ##*을 사용자의 값으로 바꿉니다.

```
select eventname, errorcode,eventsource,awsregion, useragent,COUNT(*) count
FROM cloudtrail_table-name
where errorcode = 'ThrottlingException'
AND eventtime between '2024-09-24T00:00:08Z' and '2024-09-23T23:15:08Z'
group by errorcode, awsregion, eventsource, useragent, eventname
order by count desc;
```

또한, Amazon ECS는 Amazon EventBridge로 이벤트 알림을 내보냅니다. 리소스 상태 변경 이벤트 및 서비스 작업 이벤트가 있습니다. 여기에는 `ECS_OPERATION_THROTTLED` 및 `SERVICE_DISCOVERY_OPERATION_THROTTLED`와 같은 API 제한 이벤트가 포함됩니다. 자세한 내용은 Amazon ECS 서비스 작업 이벤트 섹션을 참조하세요.

이에 응답하여 AWS Lambda와 같은 서비스에서 작업을 수행하기 위해 이러한 이벤트를 사용할 수 있습니다. 자세한 내용은 Amazon ECS 이벤트 처리 섹션을 참조하세요.

독립 실행형 작업을 실행하는 경우 일부 API 작업(예: `RunTask`)은 비동기식이며, 재시도 작업은 자동으로 수행되지 않습니다. 이 경우 EventBridge 통합에서 AWS Step Functions와 같은 서비스를 사용하여 제한되거나 실패한 작업을 재시도할 수 있습니다. 자세한 내용은 Manage a container task (Amazon ECS, Amazon SNS)를 참조하세요.

CloudWatch를 사용하여 제한 모니터링

CloudWatch는 AWS 리소스별 아래 `Usage` 네임스페이스에서 API 사용 모니터링 기능을 제공합니다. 이러한 지표는 API 유형과 지표 이름 CallCount로로 기록됩니다. 이러한 지표가 특정 임계값에 도달할 때마다 시작할 경보를 생성할 수 있습니다. 자세한 내용은 서비스 할당량 시각화 및 경보 설정을 참조하세요.

또한 CloudWatch는 이상 탐지 기능을 제공합니다. 이 기능은 기계 학습을 사용하여 활성화한 지표의 특정 동작을 기반으로 기준을 분석하고 설정합니다. 비정상적인 API 활동이 있는 경우 이 기능을 CloudWatch 경보와 함께 사용할 수 있습니다. 자세한 내용은 CloudWatch 이상 탐지 사용을 참조하세요.

제한 오류를 사전에 모니터링하여 관련 제한 한도를 높이고 고유한 애플리케이션 요구 사항에 대한 지침을 받을 수 있도록 지원에 문의할 수 있습니다.

Amazon ECS 오토 스케일링 및 용량 관리 모범 사례

Amazon ECS에서 모든 크기의 컨테이너화된 애플리케이션 워크로드를 실행할 수 있습니다. 여기에는 최소한의 테스트 환경과 글로벌 규모로 운영되는 대규모 프로덕션 환경이 포함됩니다.

다른 모든 AWS 서비스와 같이 Amazon ECS에서는 사용한 만큼만 비용을 지불합니다. 애플리케이션을 적절하게 설계하면 필요할 때 필요한 리소스만 소모하여 비용을 절감할 수 있습니다.

다음 권장 사항은 Amazon ECS 워크로드를 실행하여 서비스 수준 목표를 충족하는 동시에 비용 효율적으로 운영하는 방법을 보여줍니다.

주제

- Amazon ECS에 대한 태스크 크기 결정
- Amazon ECS 서비스 오토 스케일링 최적화
- Amazon ECS 용량 및 가용성
- Amazon ECS 클러스터 용량
- Amazon ECS에 대한 Fargate 태스크 크기 선택
- Amazon EC2의 용량 공급자를 사용하여 Amazon ECS 클러스터 용량 프로비저닝 속도 향상

Amazon ECS에 대한 태스크 크기 결정

Amazon ECS에서 컨테이너를 배포할 때 가장 중요한 선택 사항 중 하나는 컨테이너와 태스크 크기입니다. 컨테이너와 태스크 크기 모두 규모 조정 및 용량 계획에 필수적입니다.

Amazon ECS는 CPU 및 메모리와 같이 용량에 대해 두 가지 리소스 지표를 사용합니다. Amazon ECS는 CPU를 전체 vCPU의 1/1,024 단위로 측정합니다(여기서 1,024 단위는 전체 vCPU 1개와 같음). Amazon ECS는 메모리를 메가바이트 단위로 측정합니다.

작업 정의에서 리소스 예약 및 제한을 선언할 수 있습니다.

예약을 선언하는 경우 태스크에 필요한 최소 리소스 크기를 선언합니다. 태스크는 적어도 요청한 리소스 크기를 수신합니다. 애플리케이션은 선언한 예약보다 더 많은 CPU 또는 메모리를 사용할 수 있습니다. 하지만 이 경우 함께 선언한 제한이 적용됩니다.

예약 크기를 초과하여 사용하는 것을 버스팅이라고 합니다. 버스팅은 애플리케이션이 예약한 것보다 더 많은 리소스를 사용하지만 선언된 제한 내에서 유지됨을 의미합니다. Amazon ECS는 예약을 보장합니다. 예를 들어 Amazon EC2 인스턴스를 사용하여 용량을 제공하는 경우 Amazon ECS는 예약을 충족할 수 없는 인스턴스에 태스크를 배치하지 않습니다.

제한은 컨테이너 또는 작업에서 사용할 수 있는 CPU 단위 또는 메모리의 최대 크기입니다. 컨테이너에서 이 제한보다 많은 CPU를 사용하려고 하면 Amazon ECS가 이를 스로틀링합니다. 컨테이너에서 이 제한보다 많은 메모리를 사용하려고 하면 Amazon ECS가 컨테이너를 중지합니다.

이 값을 선택하는 것은 어려울 수 있습니다. 애플리케이션에 가장 적합한 값은 애플리케이션의 리소스 요구 사항에 따라 크게 달라집니다.

성공적인 리소스 요구 사항 계획을 수립하려면 애플리케이션 부하 테스트가 매우 중요합니다. 부하 테스트는 애플리케이션의 요구 사항을 더 잘 이해하는 데 도움이 됩니다.

상태 비저장 애플리케이션

로드 밸런서 이면의 애플리케이션과 같이 수평적으로 규모를 조정하는 상태 비저장 애플리케이션의 경우 먼저 애플리케이션이 요청을 지원할 때 소비하는 메모리 크기를 결정하는 것이 좋습니다.

이를 위해 `ps` 또는 `top`과 같은 기존 도구를 사용할 수 있습니다. CloudWatch Container Insights와 같은 모니터링 솔루션을 사용할 수도 있습니다.

CPU 예약을 결정할 때 비즈니스 요구 사항에 맞게 애플리케이션 규모를 조정하는 방법을 고려하세요.

256개의 CPU 단위(또는 1/4 vCPU)와 같은 더 작은 CPU 예약을 사용하여 비용을 최소화하는 세분화된 방식으로 스케일 아웃할 수 있습니다. 하지만 급증하는 수요를 감당할 만큼 빠르게 규모가 조정되지 않을 수도 있습니다.

더 큰 CPU 예약을 사용하면 보다 빠르게 스케일 인 및 스케일 아웃할 수 있습니다. 그러면 수요 급증에 더 빠르게 대응할 수 있습니다. 하지만 CPU 예약이 커지면 비용이 증가합니다.

기타 애플리케이션

싱글톤 작업자 또는 데이터베이스 서버와 같이 수평적으로 규모가 조정되지 않는 애플리케이션의 경우 사용 가능한 용량과 비용이 가장 중요한 고려 사항입니다.

서비스 수준 목표를 달성하기 위해 트래픽을 처리하려면 부하 테스트에서 표시하는 필요한 수준에 따라 메모리와 CPU 크기를 선택합니다. Amazon ECS는 적절한 용량을 갖춘 호스트에 애플리케이션을 배치하도록 보장합니다.

Amazon ECS 서비스 오토 스케일링 최적화

Amazon ECS 서비스는 관리형 작업 모음입니다. 각 서비스에는 관련 작업 정의, 원하는 작업 수 및 선택적 배치 전략이 있습니다.

Amazon ECS 서비스 오토 스케일링은 Application Auto Scaling 서비스를 통해 구현됩니다. Application Auto Scaling은 CloudWatch 지표를 지표 조정 소스로 사용합니다. 또한 CloudWatch 경보를 사용하여 서비스를 스케일 인 또는 스케일 아웃할 시기에 대한 임곗값을 설정합니다.

규모 조정에 대한 임계치를 제공합니다. 대상 추적 규모 조정이라고 하는 지표 대상을 설정할 수 있습니다. 단계 규모 조정이라고 하는 임계치를 지정할 수도 있습니다.

Application Auto Scaling을 구성한 후에는 서비스에 적절한 원하는 태스크 수를 지속적으로 계산합니다. 또한 스케일 인 또는 스케일 아웃하여 원하는 작업 수가 변경될 경우 이를 Amazon ECS에 알립니다.

서비스 Auto Scaling을 효과적으로 사용하려면 적절한 조정 지표를 선택해야 합니다. 다음 섹션에서 지표를 선택하는 방법을 설명합니다.

애플리케이션 특성 지정

애플리케이션 규모를 적절하게 조정하려면 애플리케이션을 스케일 인해야 하는 조건과 스케일 아웃해야 하는 시기를 알아야 합니다.

기본적으로 수요가 용량을 초과할 것으로 예상되는 경우 애플리케이션을 스케일 아웃해야 합니다. 반대로, 리소스가 수요를 초과할 경우 비용을 절약하기 위해 애플리케이션을 스케일 인할 수 있습니다.

사용률 지표 식별

효과적으로 규모를 조정하려면 사용률 또는 포화도를 나타내는 지표를 식별해야 합니다. 이 지표는 조정에 유용하도록 다음 속성을 보여주어야 합니다.

- 지표는 수요와 상관되어야 합니다. 리소스가 일정하게 유지되지만 수요가 변경되면 지표 값도 변경되어야 합니다. 수요가 증가하거나 감소하면 지표가 증가하거나 감소해야 합니다.
- 지표 값은 용량에 비례하여 스케일 인되어야 합니다. 수요가 일정할 때 리소스를 더 추가하면 지표 값이 비례적으로 변해야 합니다. 따라서 작업 수를 두 배로 늘리면 지표가 50% 감소해야 합니다.

사용률 지표를 식별하는 가장 좋은 방법은 스테이징 환경과 같은 사전 프로덕션 환경에서 부하 테스트를 수행하는 것입니다. 상용 및 오픈 소스 부하 테스트 솔루션이 널리 사용됩니다. 이러한 솔루션은 일반적으로 합성 부하를 생성하거나 실제 사용자 트래픽을 시뮬레이션할 수 있습니다.

부하 테스트 프로세스를 시작하려면 먼저 애플리케이션의 사용률 지표에 대한 대시보드를 빌드해야 합니다. 이러한 지표로, CPU 사용률, 메모리 사용률, I/O 작업, I/O 대기열 길이, 네트워크 처리량이 포함됩니다. CloudWatch Container Insights와 같은 서비스를 통해 이러한 지표를 수집할 수 있습니다. Amazon Managed Service for Prometheus를 Amazon Managed Grafana와 함께 사용하여 수집할 수도 있습니다. 이 프로세스 중에 애플리케이션의 응답 시간이나 작업 완료율에 대한 지표를 수집하고 도표를 작성해야 합니다.

부하 테스트를 수행할 때 작은 요청이나 느린 작업 삽입 속도로 시작합니다. 애플리케이션이 워밍업될 수 있도록 몇 분 동안 이 속도를 일정하게 유지합니다. 그런 다음, 속도를 천천히 올리고 몇 분 동안 일정하게 유지합니다. 이 주기를 반복하고 애플리케이션의 응답 또는 완료 시간이 너무 느려 서비스 수준 목표(SLO)를 충족하지 못하면 매번 비율을 높입니다.

부하 테스트를 수행하는 동안 각 사용률 지표를 살펴봅니다. 부하와 함께 증가하는 지표는 가장 적합한 사용률 지표로 사용할 수 있는 가장 좋은 후보입니다.

다음으로, 포화 상태에 도달한 리소스를 식별합니다. 동시에 사용률 지표를 살펴보고 어떤 지표가 높은 수준에서 먼저 평준화되는지 확인합니다. 또는 피크에 도달한 후 애플리케이션과 먼저 충돌하는 지표를 살펴봅니다. 예를 들어 부하를 추가할 때 CPU 사용률이 0%에서 70%~80%로 증가하는 경우 부하를 더 추가한 후에도 이 수준을 유지한다면 CPU는 포화 상태일 수 있습니다. CPU 아키텍처에 따라 100%에 도달하지 못할 수도 있습니다. 예를 들어 부하를 추가할 때 메모리 사용률이 증가하다가 애플리케이션이 작업 또는 Amazon EC2 인스턴스 메모리 제한에 도달했을 때 갑자기 애플리케이션 충돌이 발생한다고 가정합니다. 이 상황에서는 메모리가 완전히 소모되었을 가능성이 큽니다. 애플리케이션에서 여러 리소스를 사용할 수 있습니다. 따라서 가장 먼저 고갈되는 리소스를 나타내는 지표를 선택합니다.

마지막으로 태스크 또는 Amazon EC2 인스턴스 수를 두 배로 늘린 후 부하 테스트를 다시 시도합니다. 주요 지표가 이전보다 절반 정도 증가하거나 감소한다고 가정합니다. 이 경우 지표는 용량에 비례합니다. 이는 Auto Scaling에 적합한 사용률 지표입니다.

이제 이 가상 시나리오를 고려합니다. 애플리케이션의 부하 테스트를 수행한 결과, 초당 요청 100개에서 CPU 사용률이 결국 80%에 도달한다고 가정합니다. 부하를 더 추가해도 CPU 사용률은 더 이상 증가하지 않습니다. 하지만 이렇게 하면 애플리케이션 응답 속도가 느려집니다. 그런 다음, 부하 테스트를 다시 실행하여 작업 수를 두 배로 늘리되 속도를 이전 피크 값으로 유지합니다. 평균 CPU 사용률이 약 40%로 떨어지면 평균 CPU 사용률은 조정 지표로 사용하기에 적합합니다. 반면, 작업 수를 늘린 후에도 CPU 사용률이 80%로 유지되면 평균 CPU 사용률은 적절한 조정 지표가 아닙니다. 이 경우 적절한 지표를 찾으려면 더 많은 연구가 필요합니다.

공통 애플리케이션 모델 및 조정 속성

모든 종류의 소프트웨어를 AWS에서 실행할 수 있습니다. 자체 개발되는 워크로드도 많지만, 널리 사용되는 오픈 소스 소프트웨어에 기반하는 워크로드도 있습니다. 어디에서 생성되었든 서비스의 몇 가지 일반적인 디자인 패턴을 확인할 수 있습니다. 효과적인 규모 조정 방법은 대부분 패턴에 따라 달라집니다.

효율적인 CPU 의존 서버

효율적인 CPU 의존 서버는 CPU 및 네트워크 처리량 이외의 리소스를 거의 사용하지 않습니다. 각 요청은 애플리케이션에서만 처리할 수 있습니다. 요청은 데이터베이스와 같은 다른 서비스에 의존하지 않습니다. 애플리케이션은 수십만 개의 동시 요청을 처리할 수 있으며, 이를 위해 여러 CPU를 효율적으로 활용할 수 있습니다. 각 요청은 메모리 오버헤드가 적은 전용 스레드에서 처리되거나 각 CPU에서 요청을 지원하는 비동기 이벤트 루프가 실행되기도 합니다. 애플리케이션의 각 복제본은 요청을 균등하게 처리할 수 있습니다. CPU 이전에 고갈될 수 있는 유일한 리소스는 네트워크 대역폭입니다. CPU 의존 서비스에서는 피크 처리량에서도 메모리 사용률이 사용 가능한 리소스의 일부에 불과합니다.

이 유형의 애플리케이션에 대해 CPU 기반 오토 스케일링을 사용할 수 있습니다. 애플리케이션은 조정 측면에서 최대한의 유연성을 제공합니다. 여기에 더 큰 Amazon EC2 인스턴스 또는 Fargate vCPU를 제공하여 수직적으로 규모를 조정할 수 있습니다. 또한 복제본을 더 추가하여 수평적으로 규모를 조정할 수도 있습니다. 복제본을 더 추가하거나 인스턴스 크기를 두 배로 늘리면 용량 대비 평균 CPU 사용률이 절반으로 줄어듭니다.

이 애플리케이션에 Amazon EC2 용량을 사용하는 경우 `c5` 또는 `c6g` 패밀리와 같은 컴퓨팅 최적화 인스턴스에 배치하는 방법을 고려합니다.

효율적인 메모리 의존 서버

효율적인 메모리 의존 서버는 요청당 상당한 양의 메모리를 할당합니다. 최대 동시성에서(처리량이 반드시 필요하지 않음) CPU 리소스가 고갈되기 전에 메모리가 고갈됩니다. 요청이 종료되면 요청에 연결된 메모리를 비웁니다. 사용 가능한 메모리가 있는 한, 추가 요청을 수락할 수 있습니다.

이 유형의 애플리케이션에 대해 메모리 기반 오토 스케일링을 사용할 수 있습니다. 애플리케이션은 조정 측면에서 최대한의 유연성을 제공합니다. 여기에 더 큰 Amazon EC2 또는 Fargate 메모리를 제공하여 모두 수직적으로 규모를 조정할 수 있습니다. 또한 복제본을 더 추가하여 수평적으로 규모를 조정할 수도 있습니다. 복제본을 더 추가하거나 인스턴스 크기를 두 배로 늘리면 용량 대비 평균 메모리 사용률이 절반으로 줄어들 수 있습니다.

이 애플리케이션에 Amazon EC2 용량을 사용하는 경우 `r5` 또는 `r6g` 패밀리와 같은 메모리 최적화 인스턴스에 배치하는 방법을 고려합니다.

일부 메모리 의존 애플리케이션은 요청이 종료될 때 요청에 연결된 메모리를 비우지 않으므로 동시성이 줄어도 사용되는 메모리가 줄지 않습니다. 이 경우 메모리 기반 규모 조정을 사용하지 않는 것이 좋습니다.

작업자 기반 서버

작업자 기반 서버는 각 개별 작업자 스레드에 대해 하나의 요청을 차례로 처리합니다. 작업자 스레드는 POSIX 스레드와 같은 경량 스레드일 수 있습니다. UNIX 프로세스처럼 가중치가 더 큰 스레드일 수도 있습니다. 스레드가 무엇이든 애플리케이션이 지원할 수 있는 최대 동시성은 항상 존재합니다. 일반적으로 동시성 제한은 사용 가능한 메모리 리소스에 비례하여 설정됩니다. 동시성 제한에 도달하면 애플리케이션은 추가 요청을 백로그 대기열에 배치합니다. 백로그 대기열에서 오버플로가 발생하면 애플리케이션은 추가 수신 요청을 즉시 거부합니다. 이 패턴에 맞는 일반적인 애플리케이션으로, Apache 웹 서버와 Gunicorn이 있습니다.

일반적으로 요청 동시성은 이 애플리케이션 규모를 조정하는 데 가장 적합한 지표입니다. 각 복제본에는 동시성 제한이 있으므로 평균 제한에 도달하기 전에 스케일 아웃하는 것이 중요합니다.

요청 동시성 지표를 확보하는 가장 좋은 방법은 애플리케이션에서 해당 지표를 CloudWatch에 보고하도록 하는 것입니다. 애플리케이션의 각 복제본은 동시 요청 수를 사용자 지정 지표로 매우 자주 게시할 수 있습니다. 빈도는 최소 1분에 한 번으로 설정하는 것이 좋습니다. 여러 보고서가 수집된 후 평균 동시성을 조정 지표로 사용할 수 있습니다. 이 지표는 전체 동시성을 가져와 이 값을 복제본 수로 나누어 계산합니다. 예를 들어, 총 동시성이 1,000이고 복제본 수가 10개인 경우 평균 동시 실행성은 100입니다.

애플리케이션이 Application Load Balancer 뒤에 있는 경우 로드 밸런서의 `ActiveConnectionCount` 지표를 조정 지표 요소로 사용할 수도 있습니다. 평균값을 구하려면 `ActiveConnectionCount` 지표를 복제본 수로 나누어야 합니다. 원시 개수 값과는 반대로, 규모 조정에 평균값을 사용해야 합니다.

이 설계가 가장 잘 작동하려면 낮은 요청 빈도에서 응답 지연 시간의 표준 편차가 작아야 합니다. 수요가 적은 기간에는 대부분의 요청에 짧은 시간으로 응답하고 평균 응답 시간보다 훨씬 오래 걸리는 요청은 많지 않은 것이 좋습니다. 평균 응답 시간은 95 백분위수 응답 시간에 가까워야 합니다. 그렇지 않으면 결과적으로 대기열 오버플로가 발생할 수 있습니다. 이로 인해 오류가 발생합니다. 오버플로 위험을 줄이려면 필요한 경우 추가 복제본을 제공하는 것이 좋습니다.

대기 중인 서버

대기 중인 서버는 각 요청에 대해 일부 처리를 수행하지만 작동을 위해 하나 이상의 다운스트림 서비스에 크게 의존합니다. 컨테이너 애플리케이션은 종종 데이터베이스 및 기타 API 서비스와 같은 다운스트림 서비스를 많이 사용합니다. 특히 용량이 크거나 동시성이 높은 시나리오에서는 이러한 서비스가

응답하는 데 다소 시간이 걸릴 수 있습니다. 이러한 애플리케이션은 CPU 리소스를 거의 사용하지 않고 가용 메모리 측면에서 최대 동시성을 활용하는 경향이 있기 때문입니다.

대기 중인 서비스는 애플리케이션의 설계 방식에 따라 메모리 의존 서버 패턴 또는 작업자 의존 서버 패턴 중 하나에 적합합니다. 애플리케이션의 동시성이 메모리로만 제한되는 경우 평균 메모리 사용률을 조정 지표로 사용해야 합니다. 애플리케이션의 동시성이 작업자 제한을 기반으로 하는 경우 평균 동시성을 조정 지표로 사용해야 합니다.

Java 기반 서버

Java 기반 서버가 CPU에 의존하고 CPU 리소스에 비례하여 조정되는 경우 효율적인 CPU 의존 서버 패턴에 적합할 수 있습니다. 이 경우 평균 CPU 사용률이 조정 지표로 적절할 수 있습니다. 하지만 많은 Java 애플리케이션은 CPU에 의존하지 않으므로 조정하기 어렵습니다.

최고의 성능을 위해 Java Virtual Machine(JVM) 힙에 메모리를 최대한 많이 할당하는 것이 좋습니다. Java 8 업데이트 191 이상을 포함한 최신 버전의 JVM에서는 컨테이너에 맞도록 힙 크기를 최대한 크게 자동으로 설정합니다. 즉, Java에서 메모리 사용률이 애플리케이션 사용률에 비례하는 경우는 거의 없습니다. 요청 빈도와 동시성이 증가해도 메모리 사용률은 일정하게 유지됩니다. 따라서 메모리 사용률에 따라 Java 기반 서버를 확장하지 않는 것이 좋습니다. 대신 일반적으로 CPU 사용률을 기준으로 조정하는 것이 좋습니다.

Java 기반 서버에서 CPU를 모두 소모하기 전에 힙이 고갈되는 경우도 있습니다. 애플리케이션이 동시성이 높아 힙이 고갈되기 쉬운 경우 평균 연결이 가장 좋은 조정 지표입니다. 애플리케이션이 높은 처리량에서 힙이 고갈되기 쉬운 경우 평균 요청 속도가 가장 좋은 조정 지표입니다.

기타 가비지 수집 런타임을 사용하는 서버

많은 서버 애플리케이션은 .NET 및 Ruby와 같은 가비지 수집을 수행하는 런타임을 기반으로 합니다. 이러한 서버 애플리케이션은 앞서 설명한 패턴 중 하나에 적합할 수 있습니다. 그러나 Java와 마찬가지로 메모리를 기반으로 이러한 애플리케이션은 조정하지 않는 것이 좋습니다. 관찰된 평균 메모리 사용률이 처리량이나 동시성과 관련되지 않은 경우가 많기 때문입니다.

이러한 애플리케이션의 경우 애플리케이션이 CPU에 의존하는 경우 CPU 사용률을 높이는 것이 좋습니다. 그렇지 않으면 부하 테스트 결과에 따라 평균 처리량이나 평균 동시성을 기준으로 조정하는 것이 좋습니다.

작업 프로세서

많은 워크로드에는 비동기 작업 처리가 포함됩니다. 여기에는 요청을 실시간으로 수신하지 않고 대신 작업 대기열을 구독하여 작업을 수신하는 애플리케이션이 포함됩니다. 이러한 유형의 애플리케이션에

서 적절한 조정 지표는 거의 항상 대기열 깊이입니다. 대기열 증가는 보류 중인 작업이 처리 용량을 초과한다는 의미이고, 빈 대기열은 수행할 작업보다 용량이 더 많다는 의미입니다.

Amazon SQS 및 Amazon Kinesis Data Streams와 같은 AWS 메시징 서비스에서는 조정에 사용할 수 있는 CloudWatch 지표를 제공합니다. Amazon SQS의 경우 `ApproximateNumberOfMessagesVisible`이 가장 좋은 지표입니다. Kinesis Data Streams의 경우 Kinesis Client Library(KCL)에서 게시한 `MillisBehindLatest` 지표를 사용하는 방법을 고려합니다. 이 지표를 조정에 사용하기 전에 모든 소비자를 대상으로 이 지표의 평균을 구해야 합니다.

Amazon ECS 용량 및 가용성

애플리케이션 가용성은 오류 없는 환경을 제공하고 애플리케이션 지연 시간을 최소화하는 데 매우 중요합니다. 가용성은 액세스할 수 있고 수요를 충족할 수 있는 충분한 용량의 리소스를 확보하는 데 달려 있습니다. AWS는 가용성을 관리하기 위한 몇 가지 메커니즘을 제공합니다. Amazon ECS에 호스팅하는 애플리케이션의 경우 여기에 오토 스케일링 및 가용 영역(AZ)이 포함됩니다. Auto Scaling은 정의한 지표를 기반으로 태스크 또는 인스턴스 수를 관리하는 반면, 가용 영역을 사용하면 격리되어 있지만 지리적으로 가까운 위치에서 애플리케이션을 호스팅할 수 있습니다.

태스크 크기와 마찬가지로 용량과 가용성에는 반드시 고려해야 할 몇 가지 단점이 있습니다. 이상적으로는 용량이 수요에 완벽하게 일치하는 것이 좋습니다. 낮은 지연 시간과 오류율 등 서비스 수준 목표(SLO)를 충족하기 위해 요청을 처리하고 작업을 처리할 수 있는 충분한 용량을 언제나 확보할 수 있습니다. 용량이 너무 높아서 과도한 비용이 발생하거나, 너무 낮아서 지연 시간과 오류율이 높아져서는 안 됩니다.

Auto Scaling은 잠재적인 프로세스입니다. 먼저 CloudWatch는 실시간 지표를 수신해야 합니다. 그런 다음 CloudWatch는 분석을 위해 지표를 집계해야 하며 지표의 세분성에 따라 최대 몇 분이 걸릴 수 있습니다. CloudWatch는 지표를 경보 임계값과 비교하여 리소스 부족 또는 초과를 식별합니다. 불안정성을 방지하려면 설정된 임계치를 넘고 몇 분 후에 경보가 울리도록 구성해야 합니다. 또한 새 태스크를 프로비저닝하고 더 이상 필요하지 않은 태스크를 종료하는 데에도 시간이 걸립니다.

시스템의 이러한 잠재적인 지연 때문에 오버프로비저닝을 통해 여유 공간을 유지해야 합니다. 오버프로비저닝은 단기 수요 버스트를 수용하는 데 도움이 될 수 있습니다. 또한 애플리케이션이 포화 상태에 도달하지 않고 추가 요청을 처리하는 데 도움이 됩니다. 일반적으로 규모 조정 목표치를 사용률의 60~80% 사이로 설정하는 것을 권장합니다. 이렇게 하면 여전히 추가 용량을 프로비저닝하는 동안 애플리케이션이 급증하는 추가 수요를 더 잘 처리할 수 있습니다.

오버프로비저닝을 권장하는 또 다른 이유는 가용 영역 장애에 신속하게 대응하기 위해서입니다. AWS에서는 프로덕션 워크로드를 여러 가용 영역에서 제공할 것을 권장합니다. 이는 가용 영역 장애가 발생하더라도 나머지 가용 영역에서 실행 중인 태스크가 계속해서 수요를 충족할 수 있기 때문입니다. 애플

리케이션이 두 개의 가용 영역에서 실행되는 경우 일반 태스크 수를 두 배로 늘려야 합니다. 이는 잠재적인 장애 발생 시 즉시 용량을 제공할 수 있도록 하기 위한 것입니다. 애플리케이션이 3개의 가용 영역에서 실행되는 경우 일반 태스크 수의 1.5배를 실행하는 것이 좋습니다. 즉, 일반적인 제공에 필요한 태스크 2개당 3개의 태스크를 실행합니다.

규모 조정 속도 극대화

Auto Scaling은 적용되는 데 시간이 걸리는 반응형 프로세스입니다. 그러나 스케일 아웃하는 데 필요한 시간을 최소화하는 데 도움이 되는 몇 가지 방법이 있습니다.

이미지 크기를 최소화하세요. 이미지가 클수록 이미지 저장소에서 다운로드하고 압축을 푸는 데 시간이 오래 걸립니다. 따라서 이미지 크기를 더 작게 유지하면 컨테이너를 시작하는 데 필요한 시간이 줄어듭니다. 이미지 크기를 줄이려면 다음 구체적인 권장 사항을 따르세요.

- 정적 바이너리를 빌드하거나 Golang을 사용할 수 있는 경우 이미지 `FROM` 스크래치를 빌드하고 결과 이미지에 바이너리 애플리케이션만 포함하세요.
- Amazon Linux 또는 Ubuntu와 같은 업스트림 배포판 공급업체의 최소화된 기본 이미지를 사용하세요.
- 최종 이미지에 빌드 아티팩트를 포함하지 마세요. 다단계 빌드를 사용하는 것이 도움이 될 수 있습니다.
- 가능하다면 `RUN` 단계를 간소화합니다. 각 `RUN` 단계는 새 이미지 레이어를 생성하므로 레이어를 다운로드하는 데 추가 왕복 시간이 소요됩니다. 여러 명령이 `&&`로 결합된 단일 `RUN` 단계는 여러 `RUN` 단계가 있는 경우보다 레이어 수가 적습니다.
- ML 추론 데이터와 같은 데이터를 최종 이미지에 포함하려면 시작하고 트래픽 제공을 시작하는 데 필요한 데이터만 포함하세요. 서비스에 영향을 주지 않고 Amazon S3 또는 기타 스토리지에서 온디맨드 방식으로 데이터를 가져오는 경우 해당 위치에 데이터를 저장하세요.

이미지를 가까이 두세요. 네트워크 지연 시간이 길수록 이미지를 다운로드하는 데 시간이 더 오래 걸립니다. 워크로드가 있는 것과 동일한 AWS 리전의 리포지토리에 이미지를 호스팅하세요. Amazon ECR은 Amazon ECS를 사용할 수 있는 모든 리전에서 사용할 수 있는 고성능 이미지 리포지토리입니다. 컨테이너 이미지를 다운로드하기 위해 인터넷이나 VPN 링크를 거치지 마세요. 동일한 리전에서 이미지를 호스팅하면 전반적인 신뢰성이 향상됩니다. 다른 리전의 네트워크 연결 문제 및 가용성 문제의 위험을 완화합니다. 또는 Amazon ECR 리전 간 복제를 구현하여 이를 지원할 수도 있습니다.

로드 밸런서 상태 확인 임계값을 줄입니다. 로드 밸런서는 애플리케이션에 트래픽을 보내기 전에 상태 확인을 수행합니다. 대상 그룹에 대한 기본 상태 확인 구성은 90초 이상 걸릴 수 있습니다. 이 과정에서

로드 밸런서는 상태를 확인하고 요청을 수신합니다. 상태 확인 간격과 임계값 수를 낮추면 애플리케이션이 트래픽을 더 빠르게 수용하고 다른 태스크에 대한 로드를 줄일 수 있습니다.

콜드 스타트 성능을 고려하세요. Java와 같은 일부 애플리케이션은 JIT(Just-In-Time) 컴파일을 수행하는 런타임을 사용합니다. 컴파일 프로세스가 시작될 때 애플리케이션 성능을 확인할 수 있습니다. 해결 방법은 워크로드에서 지연 시간이 중요한 부분을 콜드 스타트 성능 저하를 부과하지 않는 언어로 다시 작성하는 것입니다.

타겟 추적 규모 조정 정책이 아닌 단계 규모 조정 정책을 사용하세요. Amazon ECS 작업을 위한 여러 Application Auto Scaling 옵션이 있습니다. 대상 추적이 가장 사용하기 쉬운 모드입니다. 이 옵션을 사용하는 경우 CPU 평균 사용률과 같은 지표의 목표 값을 설정하기만 하면 됩니다. 그러면 자동 스케일러가 해당 값을 달성하는 데 필요한 작업 수를 자동으로 관리합니다. 단계 조정을 사용하면 조정 지표의 특정 임계값과 임계값을 초과했을 때 추가 또는 제거할 작업 수를 정의하므로 수요 변화에 더 빠르게 대응할 수 있습니다. 더욱이 임계값 경보를 위반하는 시간을 최소화하여 수요 변화에 매우 빠르게 대응할 수 있습니다. 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 서비스 Auto Scaling을 참조하세요.

클러스터 용량을 제공하기 위해 Amazon EC2 인스턴스를 사용하는 경우 다음 권장 사항을 고려하세요.

더 큰 Amazon EC2 인스턴스와 더 빠른 Amazon EBS 볼륨을 사용하세요. 더 큰 Amazon EC2 인스턴스와 더 빠른 Amazon EBS 볼륨을 사용하면 이미지 다운로드 및 준비 속도를 향상할 수 있습니다. 주어진 Amazon EC2 인스턴스 패밀리 내에서 네트워크 및 Amazon EBS 최대 처리량은 인스턴스 크기가 증가함에 따라 증가합니다(예: `m5.xlarge`에서 `m5.2xlarge`로). 또한 Amazon EBS 볼륨을 사용자 지정하여 처리량과 IOPS를 늘릴 수도 있습니다. 예를 들어 `gp2` 볼륨을 사용하는 경우 기준 처리량을 더 많이 제공하는 더 큰 볼륨을 사용하세요. `gp3` 볼륨을 사용하는 경우 볼륨을 생성할 때 처리량과 IOPS를 지정하세요.

Amazon EC2 인스턴스에서 실행되는 태스크에는 브리지 네트워크 모드를 사용하세요. Amazon EC2에서 `bridge` 네트워크 모드를 사용하는 태스크는 `awsvpc` 네트워크 모드를 사용하는 태스크보다 빠르게 시작됩니다. `awsvpc` 네트워크 모드를 사용하는 경우 Amazon ECS는 작업을 시작하기 전에 탄력적 네트워크 인터페이스(ENI)를 인스턴스에 연결합니다. 이로 인해 지연 시간이 추가됩니다. 그러나 브리지 네트워킹을 사용하는 데에는 몇 가지 상충 관계가 있습니다. 이러한 태스크에는 자체 보안 그룹이 없으며 로드 밸런싱에 몇 가지 영향이 있습니다. 자세한 내용은 탄력적 로드 밸런싱 사용 설명서의 로드 밸런서 대상 그룹을 참조하세요.

수요 충격 처리

일부 애플리케이션은 갑자기 큰 수요 충격을 받기도 합니다. 이는 뉴스 이벤트, 대규모 세일, 미디어 이벤트, 입소문을 타고 매우 짧은 시간 내에 트래픽이 빠르고 크게 증가하는 기타 이벤트 등 다양한 이유로 발생합니다. 계획하지 않으면 수요가 가용 리소스를 빠르게 초과할 수 있습니다.

수요 충격을 처리하는 가장 좋은 방법은 수요 충격을 예상하고 그에 따라 계획을 세우는 것입니다. Auto Scaling에는 시간이 걸릴 수 있으므로 수요 충격이 시작되기 전에 애플리케이션을 스케일 아웃하는 것이 좋습니다. 최상의 결과를 얻으려면 공유 캘린더를 사용하는 팀 간의 긴밀한 협업을 포함하는 비즈니스 계획을 세우는 것이 좋습니다. 이벤트를 기획하는 팀은 사전에 애플리케이션 담당 팀과 긴밀하게 협력해야 합니다. 이를 통해 해당 팀은 명확한 일정 계획을 세울 수 있는 충분한 시간을 확보할 수 있습니다. 이벤트 전에 용량을 스케일 아웃하고 이벤트 후에 용량을 스케일 인하도록 예약할 수 있습니다. 자세한 내용은 Application Auto Scaling 사용 설명서의 예약된 크기 조정을 참조하세요.

Enterprise Support 플랜을 사용하는 경우에는 기술 계정 관리자(TAM)와도 협력하세요. TAM은 서비스 할당량을 확인하고 이벤트가 시작되기 전에 필요한 할당량을 늘리도록 할 수 있습니다. 이렇게 하면 실수로 서비스 할당량을 초과하지 않습니다. 또한 로드 밸런서와 같은 서비스를 미리 준비하여 이벤트가 원활하게 진행될 수 있도록 지원할 수 있습니다.

예상하지 못한 수요 충격을 처리하는 것은 더 어려운 문제입니다. 예상하지 못한 충격의 진폭이 충분히 크면 수요가 빠르게 용량을 초과할 수 있습니다. 또한 Auto Scaling이 반응하는 기능을 능가할 수도 있습니다. 예상하지 못한 충격에 대비하는 가장 좋은 방법은 리소스를 오버프로비저닝하는 것입니다. 언제든지 예상되는 최대 트래픽 수요를 처리할 수 있는 충분한 리소스를 확보해야 합니다.

예상하지 못한 수요 충격에 대비해 최대 용량을 유지하는 것은 비용이 많이 들 수 있습니다. 비용 영향을 완화하려면 대규모 수요 충격이 임박했음을 예측할 수 있는 선행 지표나 이벤트를 찾아보세요. 지표 또는 이벤트가 중요한 사전 알림을 안정적으로 제공하는 경우 이벤트가 발생하거나 지표가 설정한 특정 임계값을 초과하는 즉시 스케일 아웃 프로세스를 시작합니다.

애플리케이션이 예상하지 못한 갑작스러운 수요 충격을 받기 쉬운 경우에는 중요하지 않은 기능을 희생하면서도 고객을 위해 중요한 기능은 유지하는 고성능 모드를 애플리케이션에 추가하는 것을 고려하세요. 예를 들어 애플리케이션이 비용이 많이 드는 사용자 지정 응답 생성에서 정적 응답 페이지 제공으로 전환할 수 있다고 가정합니다. 이 시나리오에서는 애플리케이션의 규모를 전혀 조정하지 않고도 처리량을 크게 늘릴 수 있습니다.

마지막으로 수요 충격에 더 잘 대처하기 위해 모놀리식 서비스를 분리하는 것을 고려할 수 있습니다. 애플리케이션이 실행 비용이 많이 들고 규모 조정 속도가 느린 모놀리식 서비스인 경우 성능에 중요한 부분을 추출하거나 재작성하여 별도의 서비스로 실행할 수 있습니다. 그러면 이러한 새로운 서비스는 덜 중요한 구성 요소와 독립적으로 규모를 조정할 수 있습니다. 애플리케이션의 다른 부분과 별도로 성

능에 중요한 기능을 유연하게 스케일 아웃할 수 있으면 용량을 추가하는 데 걸리는 시간을 줄이고 비용을 절약할 수 있습니다.

Amazon ECS 클러스터 용량

여러 방법으로 Amazon ECS 클러스터에 용량을 제공할 수 있습니다. 예를 들어 Amazon EC2 인스턴스를 시작하고 Amazon ECS 컨테이너 에이전트를 사용하여 시작 시 클러스터에 등록할 수 있습니다. 그러나 조정을 직접 관리해야 하므로 이 방법은 어려울 수 있습니다. 따라서 Amazon ECS 용량 공급자를 사용하는 것이 좋습니다. 용량 공급자는 사용자를 대신해 리소스 조정을 관리합니다. Amazon EC2, Fargate, Fargate 스팟과 같이 세 가지 유형의 용량 공급자가 있습니다. Fargate 용량 공급자에 대한 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 클러스터 워크로드를 참조하고 EC2 시작 유형의 경우 EC2 시작 유형에 대한 Amazon ECS 클러스터 워크로드를 참조하세요.

Fargate 및 Fargate 스팟 용량 공급자는 사용자를 대신해 Fargate 태스크의 수명 주기를 처리합니다. Fargate는 온디맨드 용량을 제공하고 Fargate 스팟은 스팟 용량을 제공합니다. 태스크를 시작하면 Amazon ECS가 사용자를 대신해 Fargate 리소스를 프로비저닝합니다. 이 Fargate 리소스는 태스크 정의에서 선언한 태스크 수준 제한에 직접 해당하는 메모리 및 CPU 단위와 함께 제공됩니다. 각 태스크는 자체 Fargate 리소스를 수신하여 태스크와 컴퓨팅 리소스 간에 1:1 관계를 형성합니다.

Fargate 스팟에서 실행되는 태스크는 중단될 수 있습니다. 중단은 2분 경고 후에 발생합니다. 수요가 많은 기간에 발생합니다. Fargate 스팟은 배치 작업, 개발 또는 스테이징 환경과 같은 중단 방지 워크로드에 가장 적합합니다. 또한 고가용성과 짧은 지연 시간이 필요하지 않은 다른 시나리오에도 적합합니다.

Fargate 온디맨드 태스크와 함께 Fargate 스팟 태스크를 실행할 수 있습니다. 이를 함께 사용하면 저렴한 비용으로 프로비저닝 '버스트' 용량을 받을 수 있습니다.

Amazon ECS는 태스크에 대한 Amazon EC2 인스턴스 용량도 관리할 수 있습니다. 각 Amazon EC2 용량 공급자는 사용자가 지정한 Amazon EC2 Auto Scaling 그룹과 연결됩니다. Amazon EC2 용량 공급자를 사용하는 경우 클러스터 오토 스케일링은 예약된 모든 태스크를 배치할 수 있도록 Amazon EC2 Auto Scaling 그룹의 크기를 유지 관리합니다.

Amazon ECS에 대한 Fargate 태스크 크기 선택

AWS Fargate 태스크 정의의 경우 태스크 수준에서 CPU와 메모리를 지정해야 하며 오버헤드를 고려하지 않아도 됩니다. Fargate 태스크에 대해 컨테이너 레벨에서 CPU와 메모리를 지정할 수도 있습니다. 하지만 필수는 아닙니다. 리소스 제한은 사용자가 선언한 예약 수준 이상이어야 합니다. 대부분의 경우 태스크 정의에 선언된 각 컨테이너의 예약 합계로 설정할 수 있습니다. 그런 다음 숫자를 가장 가까운 Fargate 크기로 반올림합니다. 사용 가능한 크기에 대한 자세한 내용은 태스크 CPU 및 메모리를 참조하세요.

Amazon EC2의 용량 공급자를 사용하여 Amazon ECS 클러스터 용량 프로비저닝 속도 향상

Amazon EC2에서 Amazon ECS를 실행하는 고객은 Amazon ECS 클러스터 오토 스케일링(CAS)을 활용하여 Amazon EC2 Auto Scaling(ASG) 그룹의 규모 조정을 관리할 수 있습니다. CAS를 사용하면 ASG 규모를 자동으로 조정하도록 Amazon ECS를 구성하고 태스크 실행에만 집중할 수 있습니다. ECS는 추가 개입 없이 필요에 따라 ASG를 스케일 인 및 스케일 아웃하도록 보장합니다. Amazon ECS 용량 공급자는 애플리케이션의 요구 사항을 충족하기에 충분한 컨테이너 인스턴스가 있는지 확인하여 클러스터의 인프라를 관리하는 데 사용됩니다. Amazon ECS CAS의 작동 방식을 알아보려면 Deep Dive on Amazon ECS Cluster Auto Scaling을 참조하세요.

CAS는 클러스터 용량 조정을 위해 ASG와의 CloudWatch 기반 통합에 의존하므로 CloudWatch 지표 게시와 관련된 고유한 지연 시간, 지표 `CapacityProviderReservation`이 CloudWatch 경보(높음 및 낮음 모두)를 위반하는 데 걸리는 시간, 새로 시작한 Amazon EC2 인스턴스가 워밍업하는 데 걸리는 시간이 존재합니다. 더 빠른 배포를 위해 CAS의 응답성을 높이려면 다음 작업을 수행할 수 있습니다.

용량 제공자 단계 규모 조정 크기

Amazon ECS 용량 공급자는 결국 애플리케이션의 요구 사항을 충족하기 위해 컨테이너 인스턴스를 늘리거나 줄일 것입니다. Amazon ECS가 시작할 최소 인스턴스 수는 기본적으로 1로 설정됩니다. 보류 중인 작업을 배치하는 데 여러 인스턴스가 필요한 경우 배포 시간이 추가될 수 있습니다. Amazon ECS API를 사용해 `minimumScalingStepSize`를 늘려 Amazon ECS가 한 번에 스케일 인하거나 스케일 아웃하는 최소 인스턴스 수를 늘릴 수 있습니다. `maximumScalingStepSize`가 너무 낮으면 한 번에 스케일 인 또는 스케일 아웃하는 컨테이너 인스턴스 수를 제한하여 배포 속도가 느려질 수 있습니다.

> **Note**
>
> 이 구성은 현재 `CreateCapacityProvider` 또는 `UpdateCapacityProvider` API를 통해서만 사용할 수 있습니다.

인스턴스 워밍업 기간

인스턴스 워밍업 기간은 새로 시작된 Amazon EC2 인스턴스가 Auto Scaling 그룹의 CloudWatch 지표에 기여할 수 있는 기간입니다. 지정된 워밍업 기간이 만료되면 인스턴스는 ASG의 집계된 지표로 계산되고 CAS는 다음 계산 반복을 진행하여 필요한 인스턴스 수를 추정합니다.

instanceWarmupPeriod의 기본값은 300초이며 보다 반응성이 뛰어난 조정을 위해 CreateCapacityProvider 또는 UpdateCapacityProvider API를 사용하여 더 낮은 값으로 구성할 수 있습니다.

예비 용량

용량 공급자가 작업 배치에 사용할 수 있는 컨테이너 인스턴스가 없는 경우 Amazon EC2 인스턴스를 즉시 시작하여 클러스터 용량을 늘리고(스케일 아웃) 컨테이너를 시작하기 전에 해당 인스턴스가 부팅될 때까지 기다려야 합니다. 이로 인해 작업 시작 속도가 크게 느려질 수 있습니다. 여기에서는 두 가지 옵션이 있습니다.

이 경우 예비 Amazon EC2 용량이 이미 시작되어 태스크를 실행할 준비가 되어 있으면 효과적인 작업 시작 속도가 빨라집니다. `Target Capacity` 구성을 사용하여 클러스터의 예비 용량을 유지하도록 지정할 수 있습니다. 예를 들어 `Target Capacity`를 80%로 설정하면 클러스터에 항상 20%의 예비 용량이 필요함을 나타냅니다. 이 여유 용량을 통해 모든 독립 실행형 태스크를 즉시 시작할 수 있으므로 태스크 시작이 제한되지 않습니다. 이 접근 방식의 단점은 예비 클러스터 용량을 유지하는 데 드는 비용이 증가할 수 있다는 것입니다.

고려할 수 있는 또 다른 접근 방식은 용량 공급자가 아닌 서비스에 헤드룸을 추가하는 것입니다. 즉, 예비 용량을 시작하기 위한 `Target Capacity` 구성을 줄이는 대신 대상 추적 조정 지표 또는 서비스 Auto Scaling의 단계 규모 조정 임계값을 수정하여 서비스의 복제본 수를 늘릴 수 있습니다. 이 접근 방식은 워크로드가 급증하는 경우에만 유용하며 새 서비스를 배포하거나 처음으로 0개에서 N개의 태스크로 전환하는 경우에는 효과가 없다는 점에 유의하세요. 관련 조정 정책에 대한 자세한 내용은 대상 추적 조정 정책 또는 단계 조정 정책을 참조하세요.

계정 설정을 사용하여 Amazon ECS 기능에 액세스

Amazon ECS 계정 설정으로 이동하여 특정 기능을 옵트인하거나 옵트아웃할 수 있습니다. 각 AWS 리전에서는 계정 수준에서 또는 특정 사용자/역할에 대해 각 계정 설정을 옵트인 또는 옵트아웃할 수 있습니다.

다음의 경우에 해당한다면 특정 기능을 옵트인하거나 옵트아웃하는 것이 좋습니다.

- 사용자 또는 역할은 각 계정에 대해 특정 계정 설정을 옵트인하거나 옵트아웃할 수 있습니다.
- 사용자 또는 역할은 계정 내 모든 사용자에 대해 기본 옵트인 또는 옵트아웃 설정을 지정할 수 있습니다.
- 루트 사용자 또는 관리 권한이 있는 사용자는 계정의 특정 역할이나 사용자를 옵트인하거나 옵트아웃할 수 있습니다. 루트 사용자의 계정 설정이 변경되면 개별 계정 설정이 선택되지 않은 모든 사용자 및 역할에 대한 기본값이 설정됩니다.

> **Note**
>
> 페더레이션 사용자는 루트 사용자로 계정이 설정되고 자신에 대한 별도의 명시적 계정 설정은 없습니다.

다음 계정 설정을 사용할 수 있습니다. 각 계정 설정에 대해 개별적으로 옵트인 및 옵트아웃해야 합니다.

리소스 이름	자세히 알아보기
`containerInsights`	Container Insights

`serviceLongArnFormat`
`taskLongArnFormat`
`containerInstanceLongArnFormat`
	Amazon 리소스 이름(ARN) 및 ID

`tagResourceAuthorization`	태그 지정 권한 부여
`fargateFIPSMode`	AWS Fargate Federal Information Processing Standard(FIPS-140) 규정 준수
`fargateTaskRetirementWaitPeriod`	AWS Fargate 작업 사용 중지 대기 시간
`guardDutyActivate`	Runtime Monitoring(Amazon GuardDuty 통합)
`dualStackIPv6`	듀얼스택 IPv6 VPC
`awsvpcTrunking`	Linux 컨테이너 인스턴스 네트워크 인터페이스 증가
`defaultLogDriverMode`	기본 로그 드라이버 모드

Amazon 리소스 이름(ARN) 및 ID

Amazon ECS 리소스가 생성되면 각 리소스에 고유한 Amazon 리소스 이름(ARN) 및 리소스 식별자(ID)를 할당합니다. 명령줄 도구 또는 Amazon ECS API를 사용하여 Amazon ECS로 작업할 경우 특정

명령에 리소스 ARN 또는 ID가 필요합니다. 예를 들어, stop-task AWS CLI명령을 사용하여 태스크를 중지할 경우, 작업 ARN 또는 ID를 명령에 지정해야 합니다.

Amazon ECS에는 Amazon ECS 서비스, 태스크, 컨테이너 인스턴스에 대한 리소스 ID 및 Amazon 리소스 이름(ARN)에 대한 새 형식이 도입됩니다. 각 리소스 유형의 옵트인 상태에 따라 리소스가 사용하는 Amazon 리소스 이름(ARN) 형식이 달라집니다. 해당 리소스 유형에 대한 리소스 태그 지정과 같은 기능을 사용하려면 새 ARN 형식을 사용해야 합니다.

새 Amazon 리소스 이름(ARN) 및 리소스 ID 형식은 리전별로 옵트인하거나 옵트아웃할 수 있습니다. 현재 새로 생성된 모든 계정은 기본적으로 옵트인되어 있습니다.

언제든지 새로운 Amazon 리소스 이름(ARN) 및 리소스 ID 형식을 옵트인 또는 옵트아웃할 수 있습니다. 옵트인한 후 사용자가 생성하는 모든 새 리소스가 새 형식을 사용합니다.

> **Note**
>
> 리소스 ID는 생성 이후 변경되지 않습니다. 따라서 새 형식을 옵트인하거나 옵트아웃해도 기존 리소스 ID에는 영향을 주지 않습니다.

다음 섹션에서는 ARN 및 리소스 ID 형식이 어떻게 변화하고 있는지를 설명합니다. 새 형식으로의 전환에 대한 자세한 내용은 Amazon Elastic Container Service FAQ를 참조하세요.

Amazon 리소스 이름(ARN) 형식

일부 리소스에는 사용자에게 친숙한 이름이 있습니다(예: `production`이라는 이름의 서비스). 그러나 Amazon 리소스 이름(ARN) 형식을 사용하여 리소스를 지정해야 하는 경우도 있습니다. Amazon ECS 태스크, 서비스 및 컨테이너 인스턴스에 대한 새 ARN 형식에는 클러스터 이름이 포함됩니다. 새 ARN 형식을 옵트인하는 방법에 대한 자세한 내용은 Amazon ECS 계정 설정 수정 섹션을 참조하세요.

다음 표에서는 각 리소스 유형에 대한 현재 형식과 새 형식을 모두 보여 줍니다.

리소스 유형	ARN
컨테이너 인스턴스	현재: `arn:aws:ecs:` *`region`*`:`*`aws_account_id`* `:container-instance/` *`container-instance-id`* 새 형식: `arn:aws:ecs:` *`region`*`:`*`aws_account_id`* `:container-instance/` *`cluster-name`* `/`*`container-instance-id`*

리소스 유형	ARN
Amazon ECS 서비스	현재: arn:aws:ecs: *region*:*aws_account_id* :service/ *service-name* 새 형식: arn:aws:ecs: *region*:*aws_account_id* :service/ *cluster-name* /*service-name*
Amazon ECS 태스크	현재: arn:aws:ecs: *region*:*aws_account_id* :task/*task-id* 새 형식: arn:aws:ecs: *region*:*aws_account_id* :task/*cluster-name* /*task-id*

리소스 ID 길이

리소스 ID는 문자와 숫자의 고유한 조합 형식을 취합니다. 새 리소스 ID 형식에는 Amazon ECS 태스크 및 컨테이너 인스턴스에 대한 더 짧은 ID가 포함됩니다. 현재 리소스 ID 형식의 길이는 36자입니다. 새 ID는 하이픈을 포함하지 않는 32자 형식입니다. 새 리소스 ID 형식을 옵트인하는 방법에 대한 자세한 내용은 Amazon ECS 계정 설정 수정 섹션을 참조하세요.

기본값은 enabled입니다.

옵트인 후 시작된 리소스만 새 ARN 및 리소스 ID 형식을 받습니다. 기존의 모든 리소스가 영향을 받는 것은 아닙니다. Amazon ECS 서비스 및 태스크를 새 ARN 및 리소스 ID 형식으로 전환하려면 해당 서비스 또는 태스크를 다시 생성해야 합니다. 컨테이너 인스턴스를 새 ARN과 리소스 ID 형식으로 전환하려면 컨테이너 인스턴스를 드레이닝해야 하며 새 컨테이너 인스턴스가 시작되어 클러스터에 등록되어 있어야 합니다.

> **Note**
>
> Amazon ECS 서비스에 의해 시작된 작업은 해당 서비스가 2018년 11월 16일 또는 그 이후에 생성되었고 서비스를 생성한 사용자가 작업에 대한 새 형식을 옵트인한 경우에만 새 ARN 및 리소스 ID 형식을 받을 수 있습니다.

ARN 및 리소스 ID 형식 타임라인

Amazon ECS 리소스에 대한 새로운 Amazon 리소스 이름(ARN) 및 리소스 ID 형식에 대한 옵트인 및 옵트아웃 기간의 타임라인이 2021년 4월 1일에 종료되었습니다. 기본적으로 모든 새 계정은 새 형식으로 옵트인됩니다. 새로 생성되는 모든 리소스는 새로운 형식을 받고 더 이상 옵트아웃할 수 없습니다.

Container Insights

2024년 12월 2일 AWS에서는 Amazon ECS의 향상된 관찰성을 갖춘 Container Insights를 출시했습니다. 이 버전은 Amazon EC2 및 Fargate 시작 유형을 사용하여 Amazon ECS 클러스터의 관찰성을 향상할 수 있도록 지원합니다. Amazon ECS에서 향상된 관찰성으로 Container Insights를 구성하면, Container Insights는 클러스터 수준에서 환경의 컨테이너 수준까지 상세한 인프라 원격 측정을 자동으로 수집하고, 대시보드에서 다양한 지표와 차원을 보여주는 데이터를 표시합니다. 그런 다음 Container Insights 콘솔에서 이러한 기본 대시보드를 사용하여 컨테이너 상태와 성능을 더 잘 이해하고 이상 징후를 식별하여 문제를 더 빠르게 완화할 수 있습니다.

Container Insights 대신 관찰성이 향상된 Container Insights를 사용하는 것이 좋습니다. 컨테이너 환경에서 세부적인 가시성을 제공하여 평균 해결 시간을 단축하기 때문입니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 향상된 관찰성 지표를 갖춘 Amazon ECS Container Insights를 참조하세요.

`containerInsights` 계정 설정의 기본값은 `disabled`입니다.

관찰성이 향상된 Container Insights

다음 명령을 사용하여 향상된 관찰성을 갖춘 Container Insights를 켭니다.

`containerInsights` 계정 설정을 `enhanced`로 설정합니다.

```
aws ecs put-account-setting --name containerInsights --value enhanced
```

출력 예시

```
{
    "setting": {
        "name": "containerInsights",
        "value": "enhanced",
        "principalArn": "arn:aws:iam::123456789012:johndoe",
         "type": user
```

```
    }
}
```

이 계정 설정을 설정하면 모든 새 클러스터가 향상된 관찰성을 갖춘 Container Insights를 자동으로 사용합니다. `update-cluster-settings` 명령을 사용하여 기존 클러스터에 관찰성이 향상된 Container Insights를 추가하거나, Container Insights를 사용하는 클러스터를 관찰성이 향상된 Container Insights로 업그레이드합니다.

```
aws ecs update-cluster-settings --cluster cluster-name --settings
 name=containerInsights,value=enhanced
```

콘솔을 사용하여 관찰성이 향상된 Container Insights를 구성할 수도 있습니다. 자세한 내용은 Amazon ECS 계정 설정 수정 섹션을 참조하세요.

Container Insights

`containerInsights` 계정 설정을 `enabled`로 설정할 경우, 기본적으로 모든 새 클러스터에 Container Insights가 활성화됩니다. `update-cluster-settings`를 사용하여 기존 클러스터를 수정할 수 있습니다.

Container Insights를 사용하려면 `containerInsights` 계정 설정을 `enabled`로 설정합니다. 다음 명령 중 하나를 사용하여 Container Insights를 켭니다.

```
aws ecs put-account-setting --name containerInsights --value enabled
```

출력 예시

```
{
    "setting": {
        "name": "containerInsights",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:johndoe",
         "type": user
    }
}
```

`containerInsights` 계정 설정을 `enabled`로 설정할 경우, 기본적으로 모든 새 클러스터에 Container Insights가 활성화됩니다. `update-cluster-settings` 명령을 사용하여 기존 클러스터에 Container Insights를 추가합니다.

```
aws ecs update-cluster-settings --cluster cluster-name --settings
 name=containerInsights,value=enabled
```

콘솔을 사용하여 Container Insights를 구성할 수도 있습니다. 자세한 내용은 Amazon ECS 계정 설정 수정 섹션을 참조하세요.

AWS Fargate Federal Information Processing Standard(FIPS-140) 규정 준수

Fargate에서는 민감한 정보를 보호하는 암호화 모듈의 보안 요구 사항을 규정하는 Federal Information Processing Standard(FIPS-140)를 지원합니다. 이는 현재 미국 및 캐나다 정부 표준이며 연방 정보 보안 관리법(FISMA) 또는 연방 위험 및 권한 부여 관리 프로그램(FedRAMP)을 준수해야 하는 시스템에 적용됩니다.

리소스 이름은 `fargateFIPSMode`입니다.

기본값은 `disabled`입니다.

Fargate에서 Federal Information Processing Standard(FIPS-140) 규정 준수를 켜야 합니다. 자세한 내용은 the section called “AWS Fargate FIPS-140 규정 준수” 섹션을 참조하세요.

> **Important**
>
> `fargateFIPSMode` 계정 설정은 Amazon ECS API 또는 AWS CLI를 사용해야 변경할 수 있습니다. 자세한 내용은 Amazon ECS 계정 설정 수정 섹션을 참조하세요.

`fargateFIPSMode` 옵션을 `enabled`로 설정하여 `put-account-setting-default`를 생성합니다. 자세한 정보는 Amazon Elastic Container Service API Reference(Amazon Elastic Container Service API 참조)의 put-account-setting-default를 참조하세요.

- 다음 명령을 사용하여 FIPS-140 규정 준수를 켤 수 있습니다.

```
aws ecs put-account-setting-default --name fargateFIPSMode --value enabled
```

출력 예시

```
{
    "setting": {
```

```
        "name": "fargateFIPSMode",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:root",
         "type": user
    }
}
```

`list-account-settings`를 실행하여 현재 FIPS-140 규정 준수 상태를 볼 수 있습니다. `effective-settings` 옵션을 사용하면 계정 수준 설정을 볼 수 있습니다.

```
aws ecs list-account-settings --effective-settings
```

태그 지정 권한 부여

Amazon ECS에서 리소스 생성을 위한 태그 지정 권한 부여를 도입합니다. 사용자에게 리소스를 생성하는 작업에 대한 태그 지정 권한(예: `ecsCreateCluster`)이 있어야 합니다. 리소스를 생성하고 해당 리소스에 대한 태그를 지정하면 AWS는 추가 권한 부여를 수행하여 태그를 생성할 권한이 있는지 확인합니다. 따라서 `ecs:TagResource` 작업을 사용할 수 있는 명시적 권한을 부여해야 합니다. 자세한 내용은 the section called “생성 시 리소스에 태그 지정” 섹션을 참조하세요.

태그 지정 권한 부여를 옵트인하려면 `tagResourceAuthorization` 옵션을 `put-account-setting-default`로 설정한 상태로 `enable`을 실행합니다. 자세한 정보는 Amazon Elastic Container Service API Reference(Amazon Elastic Container Service API 참조)의 put-account-setting-default를 참조하세요. `list-account-settings`를 실행하여 현재 태그 지정 권한 부여 상태를 볼 수 있습니다.

- 다음 명령을 사용하여 태그 지정 권한 부여를 활성화할 수 있습니다.

```
aws ecs put-account-setting-default --name tagResourceAuthorization --value on --
region region
```

출력 예시

```
{
    "setting": {
        "name": "tagResourceAuthorization",
        "value": "on",
        "principalArn": "arn:aws:iam::123456789012:root",
        "type": user
```

```
    }
}
```

태그 지정 권한 부여를 활성화한 후 사용자가 리소스 생성 시 리소스에 태그를 지정할 수 있도록 적절한 권한을 구성해야 합니다. 자세한 내용은 the section called “생성 시 리소스에 태그 지정” 섹션을 참조하세요.

`list-account-settings`를 실행하여 현재 태그 지정 권한 부여 상태를 볼 수 있습니다. `effective-settings` 옵션을 사용하면 계정 수준 설정을 볼 수 있습니다.

```
aws ecs list-account-settings --effective-settings
```

태그 지정 권한 부여 타임라인

태그 지정 권한 부여가 활성 상태인지 확인하려면 `list-account-settings`를 실행하여 `tagResourceAuthorization` 값을 보면 됩니다. 값이 `on`인 경우 태그 지정 권한 부여가 사용 중인 것입니다. 자세한 정보는 Amazon Elastic Container Service API Reference(Amazon Elastic Container Service API 참조)의 list-account-settings를 참조하세요.

다음은 태그 지정 권한 부여와 관련된 중요한 날짜입니다.

- 2023년 4월 18일 - 태그 지정 권한 부여가 도입되었습니다. 이 기능을 사용하려면 모든 신규 및 기존 계정이 옵트인해야 합니다. 태그 지정 권한 부여 사용을 시작하도록 옵트인할 수 있습니다. 옵트인을 통해 적절한 권한을 부여해야 합니다.
- 2024년 2월 9일 - 2024년 3월 6일 - 모든 신규 계정 및 영향을 받지 않는 기존 계정에는 기본적으로 태그 지정 권한 부여가 켜져 있습니다. IAM 정책을 확인하기 위해 `tagResourceAuthorization` 계정 설정을 활성화 또는 비활성활 수 있습니다.

  AWS에서 영향을 받는 계정에 알립니다.

  이 기능을 비활성화하려면 `tagResourceAuthorization` 옵션을 `off`로 설정한 상태로 `put-account-setting-default`를 실행합니다.
- 2024년 3월 7일 - 태그 지정 권한 부여를 활성화한 경우 더 이상 계정 설정을 비활성화할 수 없습니다.

  이 날짜 이전에 IAM 정책 테스트를 완료하는 것이 좋습니다.
- 2024년 3월 29일 - 모든 계정이 태그 지정 권한 부여를 사용합니다. Amazon ECS 콘솔 또는 AWS CLI에서 더 이상 계정 수준 설정을 사용할 수 없습니다.

AWS Fargate 작업 사용 중지 대기 시간

플랫폼 수정 버전에서 실행 중인 Fargate 작업이 사용 중지로 표시된 경우 AWS에서 알림을 보냅니다. 자세한 내용은 Amazon ECS에서 AWS Fargate에 대한 태스크 사용 중지 및 유지 관리 섹션을 참조하세요.

AWS는 AWS Fargate의 기본 인프라 패치와 유지 관리를 담당합니다. Fargate에서 호스팅되는 Amazon ECS 작업에 보안 또는 인프라 업데이트가 필요하다고 AWS에서 판단한 경우, 작업을 중지하고 이를 대체할 새 작업을 시작해야 합니다. 패치 적용을 위해 작업이 사용 중지되기 전 대기 기간을 구성할 수 있습니다. 작업을 즉시 사용 중지하거나, 7일을 기다리거나, 14일을 기다릴 수 있습니다.

이 설정은 계정 수준에서 적용됩니다.

Fargate에서 작업 사용 중지를 시작하는 시간을 구성할 수 있습니다. 업데이트를 즉시 적용해야 하는 워크로드의 경우 즉시 설정(0)을 선택합니다. 더 세부적으로 제어해야 하는 경우, 예를 들어 특정 기간에만 작업을 중지할 수 있는 경우 7일(7) 또는 14일(14) 옵션을 구성합니다.

최신 플랫폼 수정 버전을 더 빨리 받으려면 짧은 대기 기간을 선택하는 것이 좋습니다.

`put-account-setting-default` 또는 `put-account-setting`을 루트 사용자 또는 관리 사용자로 실행하여 대기 기간을 구성합니다. `name`에는 `fargateTaskRetirementWaitPeriod` 옵션을 사용하고 `value` 옵션을 다음 값 중 하나로 설정합니다.

- `0` - AWS는 알림을 보내고 영향을 받은 작업을 즉시 사용 중지합니다.
- `7` - AWS는 알림을 보내고 7일 기다린 후 영향을 받은 작업을 사용 중지합니다.
- `14` - AWS는 알림을 보내고 14일 기다린 후 영향을 받은 작업을 사용 중지합니다.

기본값은 7일입니다.

자세한 정보는 Amazon Elastic Container Service API Reference의 put-account-setting-default 및 put-account-setting을 참조하세요.

다음 명령을 실행하여 대기 기간을 14일로 설정할 수 있습니다.

```
aws ecs put-account-setting-default --name fargateTaskRetirementWaitPeriod --value 14
```

출력 예시

```
{
    "setting": {
```

```
        "name": "fargateTaskRetirementWaitPeriod",
        "value": "14",
        "principalArn": "arn:aws:iam::123456789012:root",
        "type: user"
    }
}
```

`list-account-settings`를 실행하여 현재 Fargate 작업 사용 중지 대기 시간을 볼 수 있습니다. `effective-settings` 옵션을 사용합니다.

```
aws ecs list-account-settings --effective-settings
```

Linux 컨테이너 인스턴스 네트워크 인터페이스 증가

`awsvpc` 네트워크 모드를 사용하는 각 Amazon ECS 작업은 고유한 탄력적 네트워크 인터페이스(ENI)를 수신합니다. 이 인터페이스는 이를 호스팅하는 컨테이너 인스턴스에 연결되어 있습니다. Amazon EC2 인스턴스에 연결할 수 있는 네트워크 인터페이스 수에 대한 기본 제한이 있으며 기본 네트워크 인터페이스는 한 개로 계산됩니다. 예를 들어 기본적으로 `c5.large` 인스턴스에는 ENI를 3개까지 연결할 수 있습니다. 인스턴스에 대한 기본 네트워크 인터페이스는 한 개로 계산되므로 인스턴스에 ENI를 2개 더 연결할 수 있습니다. `awsvpc` 네트워크 모드를 사용하는 각 작업에는 ENI가 필요하므로 대개 이 인스턴스 유형에서는 이러한 작업을 2개만 실행할 수 있습니다.

Amazon ECS에서는 지원되는 Amazon EC2 인스턴스 유형을 사용하여 늘어난 ENI 밀도와 함께 컨테이너 인스턴스를 시작할 수 있습니다. 이러한 인스턴스 유형을 사용하고 `awsvpcTrunking` 계정 설정을 활성화하면 새로 시작된 컨테이너 인스턴스에서 추가 ENI를 사용할 수 있습니다. 이 구성을 통해 각 컨테이너 인스턴스에 추가 작업을 배치할 수 있습니다.

예를 들어 `awsvpcTrunking`이 있는 `c5.large` 인스턴스의 경우 ENI 제한이 12로 증가합니다. 컨테이너 인스턴스는 기본 네트워크 인터페이스를 가지며 Amazon ECS는 “트렁크” 네트워크 인터페이스를 생성하여 컨테이너 인스턴스에 연결합니다. 따라서 이 구성을 통해 컨테이너 인스턴스에서 현재 두 개의 작업 대신 10개의 태스크를 시작할 수 있습니다.

Runtime Monitoring(Amazon GuardDuty 통합)

Runtime Monitoring은 AWS 로그 및 네트워킹 활동을 지속적으로 모니터링하여 악의적 동작 또는 무단 동작을 식별함으로써 Fargate 및 EC2 컨테이너 인스턴스에서 실행되는 워크로드를 보호하는 지능형 위협 감지 서비스입니다.

`guardDutyActivate` 파라미터는 Amazon ECS에서 읽기 전용이며, Amazon ECS 계정의 보안 관리자가 Runtime Monitoring을 켜는지 또는 끄는지 여부를 나타냅니다. GuardDuty는 사용자를 대신하여

이 계정 설정을 제어합니다. 자세한 내용은 Runtime Monitoring으로 Amazon ECS 워크로드 보호를 참조하세요.

`list-account-settings`를 실행하여 현재 GuardDuty 통합 설정을 볼 수 있습니다.

```
aws ecs list-account-settings
```

출력 예시

```
{
    "setting": {
        "name": "guardDutyActivate",
        "value": "on",
        "principalArn": "arn:aws:iam::123456789012:doej",
        "type": aws-managed"
    }
}
```

듀얼스택 IPv6 VPC

Amazon ECS는 기본 프라이빗 IPv4 주소 외에 IPv6 주소로도 태스크 제공을 지원합니다.

태스크가 IPv6 주소를 받으려면 해당 태스크는 `awsvpc` 네트워크 모드를 사용하고, 듀얼스택 모드에 대해 구성된 VPC로 시작되어야 하며, `dualStackIPv6` 계정 설정을 활성화해야 합니다. 기타 요구 사항에 대한 자세한 내용은 EC2 시작 유형의 경우 듀얼 스택 모드로 VPC 사용하기 섹션, Fargate 시작 유형의 경우 듀얼 스택 모드로 VPC 사용하기 섹션을 참조하세요.

> **Important**
>
> `dualStackIPv6` 계정 설정은 Amazon ECS API 또는 AWS CLI를 사용해야 변경할 수 있습니다. 자세한 정보는 Amazon ECS 계정 설정 수정을 참조하세요.

2020년 10월 1일과 11월 2일 사이에 IPv6가 활성화된 서브넷에서 `awsvpc` 네트워크 모드를 사용하여 실행 중인 태스크가 있을 경우, 태스크가 실행된 리전의 기본 `dualStackIPv6` 계정 설정은 `disabled`입니다. 해당 조건이 충족되지 않을 경우 리전의 기본 `dualStackIPv6` 설정은 `enabled`입니다.

기본값은 `disabled`입니다.

기본 로그 드라이버 모드

Amazon ECS는 컨테이너에서 선택한 로그 드라이버로 로그 메시지의 기본 전송 모드 설정을 지원합니다. 전송 모드는 컨테이너에서 로그 드라이버로의 로그 흐름이 중단될 때 애플리케이션 안정성에 영향을 줍니다.

`defaultLogDriverMode` 설정은 `blocking` 및 `non-blocking`의 두 가지 값을 지원합니다. 이러한 전송 모드에 대한 자세한 내용은 Amazon Elastic Container Service API 참조의 LogConfiguration을 참조하세요.

컨테이너 정의의 `logConfiguration`에서 전송 모드를 지정하지 않으면 이 계정 설정을 사용하여 지정하는 모드가 기본값으로 사용됩니다.

기본 전송 모드는 `non-blocking`입니다.

> **Note**
>
> 2025년 6월 25일 Amazon ECS는 태스크 가용성을 로깅보다 우선시하기 위해 기본 로그 드라이버 모드를 `blocking`에서 `non-blocking`으로 변경했습니다. 이 변경 후에도 `blocking` 모드를 계속 사용하려면 다음 중 하나를 수행합니다.
>
> - 컨테이너 정의의 `logConfiguration`에서 `mode` 옵션을 `blocking`로 설정합니다.
> - `defaultLogDriverMode` 계정 설정을 `blocking`로 설정합니다.

기본 로그 드라이버 모드를 `blocking`으로 설정하려면 다음 명령을 실행할수 있습니다.

```
aws ecs put-account-setting-default --name defaultLogDriverMode --value "blocking"
```

콘솔을 사용하여 Amazon ECS 계정 설정 보기

콘솔에서 계정 설정을 확인하여 어떤 기능에 액세스할 수 있는지 확인합니다.

> **Important**
>
> `dualStackIPv6`, `fargateFIPSMode` 및 `fargateTaskRetirementWaitPeriod` 계정 설정은 AWS CLI를 사용해야 확인하거나 변경할 수 있습니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.

2. 상단의 탐색 모음에서 계정 설정을 보려는 리전을 선택합니다.
3. 탐색 페이지에서 Account Settings(계정 설정)를 선택합니다.

Amazon ECS 계정 설정 수정

계정 설정을 수정하여 Amazon ECS 기능에 액세스합니다.

`guardDutyActivate` 파라미터는 Amazon ECS에서 읽기 전용이며, Amazon ECS 계정의 보안 관리자가 Runtime Monitoring을 켜는지 또는 끄는지 여부를 나타냅니다. GuardDuty는 사용자를 대신하여 이 계정 설정을 제어합니다. 자세한 내용은 Runtime Monitoring으로 Amazon ECS 워크로드 보호를 참조하세요.

> **Important**
>
> `dualStackIPv6`, `fargateFIPSMode` 및 `fargateTaskRetirementWaitPeriod` 계정 설정은 AWS CLI를 사용해야 확인하거나 변경할 수 있습니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 상단의 탐색 모음에서 계정 설정을 보려는 리전을 선택합니다.
3. 탐색 페이지에서 Account Settings(계정 설정)를 선택합니다.
4. 업데이트를 선택합니다.
5. (선택 사항) 각 EC2 인스턴스에 대해 awsvpc 네트워크 모드에서 실행할 수 있는 태스크 수를 늘리거나 줄이려면 AWSVPC 트렁킹에서 AWSVPC 트렁킹을 선택합니다.
6. (선택 사항) 클러스터에 대해 CloudWatch Container Insights를 기본적으로 사용하거나 사용을 중지하려면 CloudWatch Container Insights 관찰성에서 다음 옵션 중 하나를 선택합니다.
   - 향상된 관찰성을 갖춘 Container Insights를 사용하려면 Container Insights with enhanced observability를 선택합니다.
   - Container Insights를 사용하려면 Container Insights를 선택합니다.
   - Container Insights 사용을 중지하려면 꺼짐을 선택합니다.
7. (선택 사항) 태그 지정 권한 부여를 활성화하거나 비활성화하려면 리소스 태그 지정 권한 부여에서 리소스 태그 지정 권한 부여를 선택하거나 선택 취소합니다.
8. (선택 사항) 컨테이너의 `logConfiguration`에 로그 전송 모드가 정의되지 않은 경우에 기본 로그 드라이버 모드를 구성하려면 기본 로그 드라이버 모드에서 다음 옵션 중 하나를 선택합니다.

- 기본 로그 드라이버 모드를 `blocking`으로 설정하려면 차단을 선택합니다.
- 기본 로그 드라이버 모드를 `non-blocking`으로 설정하려면 비차단을 선택합니다.

9. 변경 사항 저장을 선택합니다.
10. 확인 화면에서 확인(Confirm)을 선택해 해당 선택을 저장합니다.

다음 단계

관찰성이 향상된 Container Insights 또는 Container Insights를 켜면 기능을 사용하도록 선택적으로 기존 클러스터를 업데이트할 수 있습니다. 자세한 내용은 Amazon ECS 클러스터 업데이트 섹션을 참조하세요.

기본 Amazon ECS 계정 설정으로 되돌리기

AWS Management Console을 사용하여 Amazon ECS 계정 설정을 기본값으로 되돌릴 수 있습니다.

Revert to account default(계정 기본값으로 되돌리기) 옵션은 계정 설정이 더 이상 기본 설정이 아닌 경우에만 사용할 수 있습니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 상단의 탐색 모음에서 계정 설정을 보려는 리전을 선택합니다.
3. 탐색 페이지에서 Account Settings(계정 설정)를 선택합니다.
4. 업데이트를 선택합니다.
5. Revert to account default(계정 기본값으로 되돌리기)를 선택합니다.
6. 확인 화면에서 확인(Confirm)을 선택해 해당 선택을 저장합니다.

AWS CLI를 사용하여 Amazon ECS 계정 설정 관리

Amazon ECS API, AWS CLI 또는 SDK를 사용하여 계정 설정을 관리할 수 있습니다. `dualStackIPv6`, `fargateFIPSMode` 및 `fargateTaskRetirementWaitPeriod` 계정 설정을 확인하거나 변경하려면 이 도구를 사용해야 합니다.

> **Note**
>
> 듀얼 스택 서비스 엔드포인트를 사용하면 AWS CLI, SDK 및 Amazon ECS API에서 IPv4 및 IPv6 모두를 통해 Amazon ECS와 상호 작용할 수 있습니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

작업 정의에 사용할 수 있는 API 작업에 대한 자세한 내용은 Amazon Elastic Container Service API 참조의 Account setting actions를 참조하세요.

다음 명령 중 하나를 사용하여 모든 사용자 또는 역할에 대한 기본 계정 설정을 수정합니다. 사용자 또는 역할이 이러한 설정을 명시적으로 재정의하는 경우를 제외하고 이러한 변경 사항이 전체 AWS 계정에 적용됩니다.

- put-account-setting-default(AWS CLI)

```
aws ecs put-account-setting-default --name serviceLongArnFormat --value enabled --
region us-east-2
```

  이 명령을 사용하여 다른 계정 설정을 변경할 수도 있습니다. 이를 위해서 name 파라미터를 해당 계정 설정으로 바꿉니다.

- Write-ECSAccountSetting(AWS Tools for Windows PowerShell)

```
Write-ECSAccountSettingDefault -Name serviceLongArnFormat -Value enabled -Region us-
east-1 -Force
```

사용자 계정에 대한 계정 설정 변경 방법(AWS CLI)

다음 명령 중 하나를 사용하여 사용자에 대한 계정 설정을 수정합니다. 루트 사용자로서 이런 명령을 사용 중인 경우에는 사용자 또는 역할이 스스로 이러한 설정을 명시적으로 재정의하지 않는 한 이러한 변경 사항이 전체 AWS 계정에 적용됩니다.

- put-account-setting(AWS CLI)

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --region us-
east-1
```

이 명령을 사용하여 다른 계정 설정을 변경할 수도 있습니다. 이를 위해서 name 파라미터를 해당 계정 설정으로 바꿉니다.

- Write-ECSAccountSetting(AWS Tools for Windows PowerShell)

```
Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -Force
```

특정 사용자 또는 역할에 대한 계정 설정 수정(AWS CLI)

다음 명령 중 하나를 사용하고 요청 내에서 사용자, 역할 또는 루트 사용자의 ARN을 지정하여 특정 사용자 또는 역할에 대한 계정 설정을 수정합니다.

- put-account-setting(AWS CLI)

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --principal-
arn arn:aws:iam::aws_account_id:user/principalName --region us-east-1
```

이 명령을 사용하여 다른 계정 설정을 변경할 수도 있습니다. 이를 위해서 name 파라미터를 해당 계정 설정으로 바꿉니다.

- Write-ECSAccountSetting(AWS Tools for Windows PowerShell)

```
Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -PrincipalArn
 arn:aws:iam::aws_account_id:user/principalName -Region us-east-1 -Force
```

Amazon ECS에 대한 IAM 역할

IAM 역할은 계정에 생성할 수 있는, 특정 권한을 지닌 IAM 자격 증명입니다. Amazon ECS에서는 역할을 생성하여 컨테이너 또는 서비스와 같은 Amazon ECS 리소스에 권한을 부여할 수 있습니다.

Amazon ECS에 필요한 역할은 작업 정의 시작 유형 및 사용하는 기능에 따라 다릅니다. 다음 테이블을 사용하여 Amazon ECS에 필요한 IAM 역할을 결정합니다.

역할	정의	필요한 경우	추가 정보
태스크 실행 역할	이 역할을 통해 Amazon ECS가 사용자를 대신해 다른	작업은 AWS Fargate 또는 외부 인스턴스에 호스팅됩니다. 그리고,	Amazon ECS 태스크 실행 IAM 역할

역할	정의	필요한 경우	추가 정보
	AWS 서비스를 사용할 수 있습니다.	• Amazon ECR 프라이빗 리포지토리에서 컨테이너 이미지를 가져옵니다. • 작업을 실행하는 계정과 다른 계정의 Amazon ECR 프라이빗 리포지토리에서 컨테이너 이미지를 가져옵니다. • `awslogs` 로그 드라이버를 사용해 CloudWatch Logs에 컨테이너 로그를 보냅니다. 작업은 AWS Fargate 또는 Amazon EC2 인스턴스에 호스팅됩니다. 그리고, • 프라이빗 레지스트리 인증을 사용합니다. • Runtime Monitoring을 사용합니다. • 작업 정의에서 Secrets Manager 비밀 또는 AWS Systems Manager Parameter Store 파라미터를 사용하여 민감한 데이터를 참조합니다.	

역할	정의	필요한 경우	추가 정보
태스크 역할	이 역할을 통해 컨테이너의 애플리케이션 코드가 다른 AWS 서비스를 사용할 수 있습니다.	애플리케이션은 Amazon S3와 같은 다른 AWS 서비스에 액세스합니다.	Amazon ECS 작업 IAM 역할
컨테이너 인스턴스 역할	이 역할을 통해 EC2 인스턴스 또는 외부 인스턴스를 클러스터에 등록할 수 있습니다.	작업은 Amazon EC2 인스턴스 또는 외부 인스턴스에 호스팅됩니다.	Amazon ECS 컨테이너 인스턴스 IAM 역할
Amazon ECS Anywhere 역할	이 역할을 통해 외부 인스턴스에서 AWS API에 액세스할 수 있습니다.	작업은 외부 인스턴스에 호스팅됩니다.	Amazon ECS Anywhere IAM 역할
로드 밸런서에 대한 Amazon ECS 인프라 역할	이 역할을 통해 Amazon ECS는 블루/그린 배포에서 사용자를 대신해 클러스터의 로드 밸런서 리소스를 관리할 수 있습니다.	또는 Amazon RDS 블루/그린 배포를 사용하길 원합니다.	로드 밸런서에 대한 Amazon ECS 인프라 IAM 역할
Amazon ECS CodeDeploy 역할	이 역할을 통해 CodeDeploy에서 서비스를 업데이트할 수 있습니다.	CodeDeploy 블루/그린 배포 유형을 사용하여 서비스를 배포합니다.	Amazon ECS CodeDeploy IAM 역할
Amazon ECS EventBridge 역할	이 역할을 통해 EventBridge에서 서비스를 업데이트할 수 있습니다.	EventBridge 규칙 및 대상을 사용하여 작업을 예약합니다.	Amazon ECS EventBridge IAM 역할

역할	정의	필요한 경우	추가 정보
Amazon ECS 인프라 역할	이 역할을 통해 Amazon ECS에서 클러스터의 인프라 리소스를 관리할 수 있습니다.	• Amazon EBS 볼륨을 Fargate 또는 EC2 시작 유형의 Amazon ECS 작업에 연결하려고 합니다. 인프라 역할을 통해 Amazon ECS에서 작업에 사용할 Amazon EBS 볼륨을 관리할 수 있습니다. • 전송 계층 보안(TLS)을 사용하여 Amazon ECS Service Connect 서비스 간 트래픽을 암호화하려고 합니다. • VPC Lattice 대상 그룹을 생성하려는 참입니다.	Amazon ECS 인프라 IAM 역할

Amazon ECS 작업 정의

작업 정의는 애플리케이션에 대한 청사진과 같습니다. 작업 정의는 애플리케이션을 구성하는 파라미터 및 하나 이상의 컨테이너를 설명하는 JSON 형식의 텍스트 파일입니다.

태스크 정의에서 지정할 수 있는 일부 파라미터는 다음과 같습니다.

- 사용할 시작 유형으로서 해당 태스크가 호스팅되는 인프라 결정
- 태스크의 각 컨테이너에 사용할 Docker 이미지
- 각 태스크 또는 태스크 내 각 컨테이너에서 사용할 CPU 및 메모리의 양
- 메모리 및 CPU 요구 사항
- 작업이 실행되는 컨테이너의 운영 체제
- 태스크의 컨테이너에 사용할 Docker 네트워킹 모드
- 태스크에 사용할 로깅 구성
- 컨테이너가 종료 또는 실패하더라도 태스크를 계속 실행할지 여부
- 컨테이너 시작 시 컨테이너가 실행할 명령
- 태스크의 컨테이너에서 사용할 데이터 볼륨
- 태스크에서 사용해야 하는 IAM 역할

작업 정의 파라미터의 전체 목록은 Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터를 참조하세요.

태스크 정의를 만든 후 태스크 정의를 태스크 또는 서비스로 실행할 수 있습니다.

- 태스크는 클러스터 내 태스크 정의를 인스턴스화하는 것입니다. Amazon ECS에서 애플리케이션에 대한 태스크 정의를 생성하면 클러스터에서 실행할 태스크 수를 지정할 수 있습니다.
- Amazon ECS 서비스는 Amazon ECS 클러스터에서 원하는 수의 태스크를 동시에 실행하고 유지할 수 있습니다. 태스크가 어떤 이유로든 실패하거나 중지하면 Amazon ECS 서비스 스케줄러가 태스크 정의에 따라 다른 인스턴스를 시작하는 방식으로 작동합니다. 이로써 이를 대체하여 서비스에서 원하는 수의 태스크를 유지할 수 있습니다.

주제

- Amazon ECS 작업 정의 상태

- Amazon ECS에 대한 애플리케이션 설계
- 콘솔을 사용하여 Amazon ECS 작업 정의 생성
- 콘솔을 사용하여 Amazon ECS 작업 정의 업데이트
- 콘솔을 사용하여 Amazon ECS 작업 정의 개정 등록 취소
- 콘솔을 사용하여 Amazon ECS 작업 정의 개정 삭제
- Amazon ECS 작업 정의 사용 사례
- Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터
- EC2 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터
- Amazon ECS 태스크 정의 템플릿
- Amazon ECS 작업 정의 예제

Amazon ECS 작업 정의 상태

작업을 생성, 등록 취소 또는 삭제하면 작업 정의 상태가 변경됩니다. 콘솔에서 또는 `DescribeTaskDefinition`을 사용하여 작업 정의 상태를 볼 수 있습니다.

다음은 작업 정의에 대해 가능한 상태입니다.

ACTIVE

작업 정의는 Amazon ECS에 등록된 후 `ACTIVE` 상태가 됩니다. `ACTIVE` 상태의 작업 정의를 사용하여 작업을 실행하거나 서비스를 생성할 수 있습니다.

INACTIVE

작업 정의를 등록 취소하면 작업 정의가 `ACTIVE` 상태에서 `INACTIVE` 상태로 전환됩니다. `DescribeTaskDefinition`을 호출하여 `INACTIVE` 작업 정의를 검색할 수 있습니다. `INACTIVE` 상태의 작업 정의를 사용하여 새 작업을 실행하거나 새 서비스를 만들 수 없습니다. 기존 서비스 또는 작업에는 영향을 미치지 않습니다.

DELETE_IN_PROGRESS

삭제를 위해 작업 정의를 제출하면 작업 정의가 `INACTIVE` 상태에서 `DELETE_IN_PROGRESS` 상태로 전환됩니다. 작업 정의가 `DELETE_IN_PROGRESS` 상태가 되면 Amazon ECS는 대상 작업 정의가 활성 작업이나 배포에서 참조되고 있지 않은지 주기적으로 확인한 다음 작업 정의를 영구적으로 삭제합니다. `DELETE_IN_PROGRESS` 상태의 작업 정의를 사용하여 새 작업을 실행하거나 새 서비

스를 만들 수 없습니다. 작업 정의는 기존 작업 및 서비스에 영향을 미치지 않으며 언제든지 삭제를 위해 제출할 수 있습니다.

`DELETE_IN_PROGRESS` 상태인 작업 정의는 콘솔에서 볼 수 있으며 `DescribeTaskDefinition`을 호출하여 작업 정의를 검색할 수 있습니다.

모든 `INACTIVE` 작업 정의 개정을 삭제하면 작업 정의 이름이 콘솔에 표시되지 않고 API에도 반환되지 않습니다. 작업 정의 개정이 `DELETE_IN_PROGRESS` 상태인 경우 작업 정의 이름이 콘솔에 표시되고 API에서 반환됩니다. 작업 정의 이름은 Amazon ECS에서 유지되며 다음에 해당 이름으로 작업 정의를 생성하면 개정 번호가 증가합니다.

작업 정의를 관리하는 데 AWS Config를 사용하는 경우 모든 작업 정의 등록에 대해 AWS Config에서 요금이 부과됩니다. 최신 `ACTIVE` 작업 정의의 등록을 취소하는 경우에만 요금이 부과됩니다. 작업 정의를 삭제하는 데는 요금이 부과되지 않습니다. 요금에 대한 자세한 내용은 AWS Config 요금 부분을 참조하세요.

삭제를 차단할 수 있는 Amazon ECS 리소스

작업 정의 개정을 사용하는 Amazon ECS 리소스가 있는 경우 작업 정의 삭제 요청이 완료되지 않습니다. 다음 리소스는 작업 정의가 삭제되지 않도록 할 수 있습니다.

- Amazon ECS 단독 작업 - 작업을 정상으로 유지하려면 작업 정의가 필요합니다.
- Amazon ECS 서비스 작업 - 작업을 정상으로 유지하려면 작업 정의가 필요합니다.
- Amazon ECS 서비스 배포 및 작업 세트 - Amazon ECS 배포 또는 작업 세트에 대한 조정 이벤트가 시작될 때 작업 정의가 필요합니다.

작업 정의가 `DELETE_IN_PROGRESS` 상태를 유지하는 경우 콘솔 또는 AWS CLI를 사용하여 작업 정의 삭제를 차단하는 리소스를 식별한 다음 중지할 수 있습니다.

차단된 리소스가 제거된 후 작업 정의 삭제

작업 정의 삭제를 차단하는 리소스를 제거하면 다음 규칙이 적용됩니다.

- Amazon ECS 작업 - 작업이 중지된 후 작업 정의 삭제를 완료하는 데 최대 1시간이 걸릴 수 있습니다.
- Amazon ECS 서비스 배포 및 작업 세트 - 배포 또는 작업 세트를 삭제한 후 작업 정의 삭제를 완료하는 데 최대 24시간이 걸릴 수 있습니다.

Amazon ECS에 대한 애플리케이션 설계

애플리케이션에 대한 작업 정의를 생성하여 애플리케이션을 설계합니다. 작업 정의에는 다음을 포함하여 애플리케이션에 대한 정보를 정의하는 파라미터가 포함됩니다.

- 사용할 시작 유형으로, 해당 작업이 호스팅된 인프라를 결정합니다.

  EC2 시작 유형을 사용할 때 인스턴스 유형도 선택합니다. GPU와 같은 일부 인스턴스 유형의 경우 추가 파라미터를 설정해야 합니다. 자세한 내용은 Amazon ECS 작업 정의 사용 사례 섹션을 참조하세요.
- 애플리케이션 코드와 애플리케이션 코드를 실행하는 데 필요한 모든 종속성이 들어 있는 컨테이너 이미지.
- 태스크의 컨테이너에서 사용할 네트워킹 모드.

  네트워킹 모드는 작업이 네트워크를 통해 통신하는 방식을 결정합니다.

  EC2 인스턴스에서 실행되는 태스크의 경우 여러 옵션이 있지만 awsvpc 네트워크 모드를 사용하는 것이 좋습니다. awsvpc 네트워크 모드는 애플리케이션이 서로 통신하는 방식과 VPC 내 다른 서비스와 통신하는 방식을 더 세부적으로 제어하여 컨테이너 네트워킹을 간소화합니다.

  Fargate에서 실행되는 태스크의 경우 awsvpc 네트워크 모드만 사용해야 합니다.
- 작업에 사용할 로깅 구성.
- 작업의 컨테이너에서 사용할 데이터 볼륨.

작업 정의 파라미터의 전체 목록은 Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터를 참조하세요.

작업 정의를 생성할 때는 다음 지침을 따르세요.

- 각 작업 정의 패밀리는 하나의 비즈니스 용도로만 사용합니다.

  여러 유형의 애플리케이션 컨테이너를 동일한 태스크 정의로 그룹화하면 해당 컨테이너를 독립적으로 조정할 수 없습니다. 예를 들어 웹 사이트와 API에는 일반적으로 서로 다른 조정 패턴이 필요합니다. 트래픽이 증가하면 필요한 웹 컨테이너 수가 API 컨테이너 수와 달라질 수 있습니다. 이 두 컨테이너를 동일한 태스크 정의에 배포하는 경우 모든 태스크에서 동일한 수의 웹 컨테이너와 API 컨테이너가 실행됩니다.
- 각 애플리케이션 버전을 작업 정의 패밀리 내의 작업 정의 개정과 일치시킵니다.

태스크 정의 패밀리 내에서 각 태스크 정의 개정은 특정 컨테이너 이미지 설정의 특정 시점 스냅샷을 나타냅니다. 컨테이너가 특정 버전의 애플리케이션 코드를 실행하는 데 필요한 모든 구성 요소의 스냅샷인 것과 비슷합니다.

애플리케이션 코드 버전, 컨테이너 이미지 태그, 태스크 정의 개정 간에 일대일 매핑을 생성합니다. 일반적인 릴리스 프로세스에는 git 커밋 SHA로 태그가 지정된 컨테이너 이미지로 변환되는 git 커밋이 포함됩니다. 그러면 해당 컨테이너 이미지 태그에 고유한 Amazon ECS 작업 정의 개정이 적용됩니다. 마지막으로 Amazon ECS 서비스가 새 태스크 정의 개정을 배포하도록 업데이트됩니다.

- 각 작업 정의 패밀리에 대해 서로 다른 IAM 역할을 사용합니다.

  고유한 IAM 역할로 각 작업 정의를 정의합니다. 각 비즈니스 구성 요소에 고유한 태스크 정의 패밀리를 제공하며 이 사례를 구현합니다. 이 두 가지 모범 사례를 모두 구현하면 AWS 계정 내 리소스에 대한 각 서비스의 액세스 권한을 제한할 수 있습니다. 예를 들어 인증 서비스에 비밀번호 데이터베이스에 연결할 수 있는 액세스 권한을 부여할 수 있습니다. 이와 동시에 주문 서비스만 신용 카드 결제 정보에 액세스할 수 있도록 할 수 있습니다.

Amazon ECS 컨테이너 이미지 모범 사례

컨테이너 이미지는 컨테이너 빌드 방법에 대한 일련의 지침입니다. 컨테이너 이미지에는 애플리케이션 코드와 애플리케이션 코드를 실행하는 데 필요한 모든 종속성이 들어 있습니다. 애플리케이션 종속성에는 애플리케이션 코드가 사용하는 소스 코드 패키지, 해석되는 언어를 위한 언어 런타임, 동적으로 연결된 코드가 사용하는 바이너리 패키지 등이 포함됩니다.

컨테이너 이미지를 디자인하고 빌드할 때는 다음 지침을 따르세요.

- 모든 애플리케이션 종속성을 컨테이너 이미지 내에 정적 파일로 저장하여 완전한 컨테이너 이미지를 만듭니다.

  컨테이너 이미지에서 변경할 사항이 있으면 변경 사항이 적용된 새 컨테이너 이미지를 구축합니다.
- 컨테이너 내에서 단일 애플리케이션 프로세스를 실행합니다.

  컨테이너 수명은 애플리케이션 프로세스가 실행되는 동안입니다. Amazon ECS는 장애가 발생한 프로세스를 교체하고 교체 프로세스를 시작할 위치를 결정합니다. 전체 이미지를 지원하면 전체 배포의 복원력이 향상됩니다.
- 애플리케이션에서 `SIGTERM`을 처리하게 합니다.

Amazon ECS에서 작업을 중지하면 먼저 SIGTERM 신호를 작업에 보내 애플리케이션에 해당 작업을 완료 및 종료해야 함을 알립니다. 그러면 Amazon ECS에서 SIGKILL 메시지를 보냅니다. 애플리케이션에서 SIGTERM을 무시하는 경우 Amazon ECS 서비스는 프로세스를 종료하기 위한 SIGKILL 신호 전송을 기다려야 합니다.

애플리케이션이 작업을 완료하는 데 걸리는 시간을 파악하고 애플리케이션이 SIGTERM 신호를 처리하는지 확인해야 합니다. 애플리케이션의 신호 처리에서는 애플리케이션이 새 작업을 받지 못하도록 하고 진행 중인 작업을 완료해야 합니다. 또는 완료하는 데 시간이 너무 오래 걸릴 경우 완료되지 않은 작업을 작업 외부의 스토리지에 저장해야 합니다.

- stdout 및 stderr에 로그를 기록하도록 컨테이너화된 애플리케이션을 구성합니다.

  애플리케이션 코드에서 로그 처리를 분리하면 인프라 수준에서 로그 처리를 더욱 유연하게 조정할 수 있습니다. 한 가지 예로, 로깅 시스템을 변경할 수 있습니다. 서비스를 수정하고 새 컨테이너 이미지를 구축 및 배포하는 대신 설정을 조정할 수 있습니다.

- 태그를 사용하여 컨테이너 이미지의 버전을 관리합니다.

  컨테이너 이미지는 컨테이너 레지스트리에 저장됩니다. 레지스트리의 각 이미지는 태그로 식별됩니다. latest라는 태그가 있습니다. 이 태그는 git 리포지토리의 HEAD와 마찬가지로 최신 버전의 애플리케이션 컨테이너 이미지를 가리키는 포인터 역할을 합니다. latest 태그는 테스트 용도로만 사용하는 것이 좋습니다. 가장 좋은 방법은 각 빌드의 고유한 태그로 컨테이너 이미지에 태그를 지정하는 것입니다. 이미지를 빌드하는 데 사용된 git 커밋에 대한 git SHA를 사용하여 이미지에 태그를 지정하는 것이 좋습니다.

  모든 커밋에 대해 컨테이너 이미지를 빌드할 필요는 없습니다. 하지만 특정 코드 커밋을 프로덕션 환경에 릴리스할 때마다 새 컨테이너 이미지를 빌드하는 것이 좋습니다. 또한 이미지 내에 있는 코드의 git 커밋에 해당하는 태그로 이미지에 태그를 지정하는 것이 좋습니다. 이미지에 git 커밋으로 태그를 지정한 경우 이미지가 실행 중인 코드 버전을 더 빨리 찾을 수 있습니다.

  또한 Amazon Elastic Container Registry에서 변경할 수 없는 이미지 태그를 켜는 것이 좋습니다. 이 설정을 사용하면 태그가 가리키는 컨테이너 이미지를 변경할 수 없습니다. 대신 Amazon ECR은 새 이미지를 새 태그로 업로드해야 합니다. 자세한 내용은 Amazon ECR 사용 설명서의 이미지 태그 변경 가능성을 참조하세요.

AWS Fargate에 실행하도록 애플리케이션을 설계하는 경우 동일 작업 정의에 여러 컨테이너를 배포하는 방식과 여러 작업 정의에 따로 컨테이너를 배포하는 방식 중에서 결정해야 합니다. 다음의 조건이 필요한 경우, 단일 태스크 정의에 여러 컨테이너를 배포하는 것이 좋습니다.

- 컨테이너가 공통 수명 주기를 공유하는 경우(즉, 함께 시작하고 종료).
- 컨테이너가 동일한 기본 호스트에서 실행되어야 하는 경우(즉, 로컬호스트 포트에서 한 컨테이너가 다른 컨테이너를 참조함)
- 컨테이너가 리소스를 공유합니다.
- 컨테이너가 데이터 볼륨을 공유하는 경우

다음의 조건이 필요한 경우, 여러 태스크 정의에 따로 컨테이너를 배포하는 것이 좋습니다. 그러면 컨테이너를 개별적으로 규모 조정, 프로비저닝 및 프로비저닝 해제할 수 있습니다.

Amazon ECS 작업 크기 모범 사례

컨테이너와 작업 크기 모두 규모 조정 및 용량 계획에 필수적입니다. Amazon ECS에서 CPU 및 메모리는 용량에 사용되는 두 가지 리소스 지표입니다. CPU는 전체 vCPU의 1/1,024 단위로 측정됩니다(여기서 1,024 단위는 전체 vCPU 1개와 같음). 메모리는 메가바이트 단위로 측정됩니다. 태스크 정의에서 리소스 예약 및 제한을 구성할 수 있습니다.

예약을 구성하는 경우 작업에 필요한 최소 리소스 크기를 설정하는 것입니다. 작업은 적어도 요청된 리소스 크기를 수신합니다. 애플리케이션은 선언한 예약보다 더 많은 CPU 또는 메모리를 사용할 수 있습니다. 하지만 이 경우 함께 선언한 모든 제한에 종속됩니다. 예약 크기를 초과하여 사용하는 것을 버스팅이라고 합니다. Amazon ECS에서는 예약이 보장됩니다. 예를 들어 Amazon EC2 인스턴스를 사용하여 용량을 제공하는 경우 Amazon ECS는 예약을 이행할 수 없는 인스턴스에 작업을 배치하지 않습니다.

제한은 컨테이너 또는 작업에서 사용할 수 있는 CPU 단위 또는 메모리의 최대 크기입니다. 이 제한보다 많은 CPU를 사용하려고 하면 제한이 발생합니다. 이때 메모리를 더 사용하려고 하면 컨테이너가 중지됩니다.

이 값을 선택하는 것은 어려울 수 있습니다. 애플리케이션에 가장 적합한 값은 애플리케이션의 리소스 요구 사항에 따라 크게 달라지기 때문입니다. 성공적인 리소스 요구 사항 계획을 수립하고 애플리케이션의 요구 사항을 더 잘 이해하기 위해서는 애플리케이션 부하 테스트가 매우 중요합니다.

상태 비저장 애플리케이션

로드 밸런서 이면의 애플리케이션과 같이 수평적으로 규모를 조정하는 상태 비저장 애플리케이션의 경우 먼저 애플리케이션이 요청을 처리할 때 소비하는 메모리 크기를 결정하는 것이 좋습니다. 이를 위해CloudWatch Container Insights와 같은 모니터링 솔루션 또는 기존 도구(ps 또는 top)를 사용할 수 있습니다.

CPU 예약을 결정할 때는 비즈니스 요구 사항에 맞게 애플리케이션 규모를 조정하는 방법을 고려합니다. 256개의 CPU 단위(또는 1/4 vCPU)와 같은 더 작은 CPU 예약을 사용하여 비용을 최소화하는 세분화된 방식으로 스케일 아웃할 수 있습니다. 하지만 급증하는 수요를 감당할 만큼 빠르게 규모가 조정되지 않을 수도 있습니다. 더 큰 CPU 예약을 사용하면 보다 빠르게 스케일 인 및 스케일 아웃할 수 있으므로 수요 급증에 더 빠르게 대응할 수 있습니다. 하지만 CPU 예약이 커지면 비용이 증가합니다.

기타 애플리케이션

싱글톤 작업자 또는 데이터베이스 서버와 같이 수평적으로 규모가 조정되지 않는 애플리케이션의 경우 사용 가능한 용량과 비용이 가장 중요한 고려 사항입니다. 서비스 수준 목표를 달성하기 위해 트래픽을 처리하려면 부하 테스트에서 표시하는 필요한 수준에 따라 메모리와 CPU 크기를 선택해야 합니다. Amazon ECS는 적절한 용량을 갖춘 호스트에 애플리케이션을 배치하도록 보장합니다.

EC2 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션

Amazon EC2 인스턴스에서 호스팅되는 Amazon ECS 태스크의 네트워킹 동작은 태스크 정의에 정의된 네트워크 모드에 따라 다릅니다. 다른 네트워크 모드를 사용해야 할 특별한 필요가 있지 않는 한 `awsvpc` 네트워크 모드를 사용하는 것이 좋습니다.

사용 가능한 네트워크 모드는 다음과 같습니다.

네트워크 모드	EC2에서의 Linux 컨테이너	EC2에서의 Windows 컨테이너	설명
`awsvpc`	예	예	태스크에 고유한 탄력적 네트워크 인터페이스(ENI)와 기본 프라이빗 IPv4 주소가 할당됩니다. 이렇게 하면 태스크에 Amazon EC2 인스턴스와 동일한 네트워킹 속성이 적용됩니다.
`bridge`	예	아니요	태스크는 태스크를 호스팅하는 각 Amazon EC2 인스턴스 내부에서 실행되는 Docker의 기본 가상 네트워크를 이용합니다. Linux의 기본 가상 네트워크는 `bridge` Docker 네트워크 드라이버를 사용합니다. 이는 작업 정의에 네트워크 모드가 지정되지 않았을 때의 Linux의 기본 네트워크 모드입니다.

네트워크 모드	EC2에서의 Linux 컨테이너	EC2에서의 Windows 컨테이너	설명
`host`	예	아니요	태스크는 해당 태스크를 호스팅하는 Amazon EC2 인스턴스의 ENI에 컨테이너 포트를 직접 매핑하여 Docker의 기본 가상 네트워크를 우회하는 호스트 네트워크를 사용합니다. 이 네트워크 모드에서는 동적 포트 매핑을 사용할 수 없습니다. 이 모드를 사용하는 작업 정의의 컨테이너는 특정`hostPort` 숫자를 지정해야 합니다. 호스트의 포트 번호는 여러 작업에서 사용할 수 없습니다. 따라서 단일 Amazon EC2 인스턴스에서 같은 태스크 정의를 가진 여러 태스크를 실행할 수 없습니다.
`none`	예	아니요	태스크에 외부 네트워크 연결이 없습니다.
`default`	아니요	예	태스크는 태스크를 호스팅하는 각 Amazon EC2 인스턴스 내부에서 실행되는 Windows에서의 Docker 기본 가상 네트워크를 이용합니다. Windows의 기본 가상 네트워크는`nat` Docker 네트워크 드라이버를 사용합니다. 이는 작업 정의에서 네트워크 모드가 지정되지 않았을 때의 Windows의 기본 네트워크 모드입니다.

Linux에서의 Docker 네트워킹에 대한 자세한 정보는 Docker 설명서에서 네트워킹 개요를 참조하세요.

Windows에서의 Docker 네트워킹에 대한 자세한 내용은 Windows 설명서의 Microsoft 컨테이너에서 Windows 컨테이너 네트워킹 부문을 참조하세요.

Amazon ECS 작업에 대한 네트워크 인터페이스 할당

`awsvpc` 네트워크 모드에서 제공하는 태스크 네트워킹 기능은 Amazon ECS 태스크에 Amazon EC2 인스턴스와 동일한 네트워킹 속성을 제공합니다. `awsvpc` 네트워크 모드를 사용하면 컨테이너 네트워킹을 간소화합니다. 애플리케이션이 서로 통신하는 방식과 VPC 내 다른 서비스와 통신하는 방식을 더 세부적으로 제어할 수 있기 때문입니다. 또한 `awsvpc` 네트워크 모드는 태스크 내에서 더 세부적으로 보안 그룹 및 네트워크 모니터링 도구를 사용 가능하게 만들어 컨테이너의 보안을 강화합니다. VPC 흐

름 로그와 같은 기타 Amazon EC2 네트워킹 기능을 이용하여 작업에서 주고받는 트래픽을 모니터링할 수도 있습니다. 또한 같은 태스크에 속한 컨테이너는 `localhost` 인터페이스로 통신할 수 있습니다.

작업 탄력적 네트워크 인터페이스(ENI)는 Amazon ECS의 완전관리형 기능입니다. Amazon ECS는 ENI를 생성하고 지정된 보안 그룹이 있는 호스트 Amazon EC2 인스턴스에 이를 연결합니다. 태스크는 Amazon EC2 인스턴스가 기본 네트워크 인터페이스에서 하는 것과 동일한 방식으로 ENI에서 네트워크 트래픽을 보내고 받습니다. 각 태스크 ENI는 기본적으로 프라이빗 IPv4 주소가 할당됩니다. VPC가 듀얼 스택 모드를 사용하도록 설정되어 있고 IPv6 CIDR 블록과 함께 서브넷을 사용하는 경우 태스크 ENI도 IPv6 주소를 수신합니다. 각 태스크는 ENI를 하나만 가질 수 있습니다.

이 ENI는 계정의 Amazon EC2 콘솔에서 볼 수 있습니다. 계정은 ENI를 분리하거나 수정할 수 없습니다. 이는 실행 중인 태스크와 연결된 ENI가 우발적으로 삭제되는 일을 방지하기 위한 것입니다. Amazon ECS 콘솔 또는 DescribeTasks API 태스크를 통해 태스크에 대한 ENI 연결 정보를 볼 수 있습니다. 태스크가 중지되거나 서비스의 규모가 작아진 경우에는 태스크 ENI가 분리되고 삭제됩니다.

ENI 밀도를 높여야 하는 경우 `awsvpcTrunking` 계정 설정을 사용합니다. 또한 Amazon ECS는 컨테이너 인스턴스에 대한 '트렁크' 네트워크 인터페이스를 생성하고 연결합니다. 트렁크 네트워크는 Amazon ECS에서 완전히 관리됩니다. 트렁크 ENI는 Amazon ECS 클러스터에서 컨테이너 인스턴스를 종료하거나 등록 취소할 때 삭제됩니다. `awsvpcTrunking` 계정 설정에 대한 자세한 내용은 사전 조건 섹션을 참조하세요.

작업 정의의 `networkMode` 파라미터에서 `awsvpc`를 지정합니다. 자세한 내용은 네트워크 모드 섹션을 참조하세요.

그 다음에 작업을 실행하거나 서비스를 생성할 때 작업을 배치할 하나 이상의 서브넷과 ENI에 연결할 하나 이상의 보안 그룹이 포함된 `networkConfiguration` 파라미터를 사용합니다. 자세한 내용은 네트워크 구성 섹션을 참조하세요. 이 태스크는 해당 서브넷과 동일한 가용 영역에서 호환되는 Amazon EC2 인스턴스에 배치되고 지정된 보안 그룹은 이 태스크에 대해 프로비저닝된 ENI와 연결됩니다.

Linux 고려 사항

Linux 운영 체제를 사용할 때는 다음 사항을 고려하세요.

- `awsvpc` 모드에서 p5.48xlarge 인스턴스를 사용하는 경우 인스턴스에서 1개를 초과하는 태스크를 실행할 수 없습니다.
- `awsvpc` 네트워크 모드를 사용하는 태스크 및 서비스는 사용자를 대신하여 Amazon ECS에 다른 Amazon ECS 서비스를 호출할 권한을 부여하는 AWS 서비스 연결 역할이 필요합니다. 이 역할은 클러스터를 생성하거나 AWS Management Console에서 서비스를 생성 또는 업데이트할 때 자동으로

생성됩니다. 자세한 정보는 Amazon ECS에 대해 서비스 연결 역할 사용을 참조하세요. 다음 AWS CLI 명령을 사용하여 서비스 연결 역할을 생성할 수도 있습니다.

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Amazon EC2 Linux 인스턴스는 awsvpc 네트워크 모드를 사용하는 태스크를 실행하기 위해 버전 1.15.0 이상의 컨테이너 에이전트가 필요합니다. Amazon ECS 최적화 AMI를 사용하는 경우 해당 인스턴스에는 1.15.0-4 이상 버전의 ecs-init 패키지가 필요합니다.
- Amazon ECS는 enableDnsHostnames 및 enableDnsSupport 옵션이 모두 VPC에서 활성화된 경우 Amazon에서 제공한 (내부) DNS 호스트 이름을 함께 사용하여 태스크의 호스트 이름을 채웁니다. 이러한 옵션이 사용 설정되지 않으면 태스크의 DNS 호스트 이름이 임의의 호스트 이름이 설정됩니다. VPC DNS 설정에 대한 자세한 정보는 Amazon VPC 사용 설명서의 VPC와 함께 DNS 사용을 참조하세요.
- awsvpc 네트워크 모드를 사용하는 각 Amazon ECS 태스크는 고유한 탄력적 네트워크 인터페이스(ENI)를 받습니다. 이 ENI는 이를 호스팅하는 Amazon EC2 인스턴스에 연결되어 있습니다. Amazon EC2 Linux 인스턴스에 연결할 수 있는 네트워크 인터페이스 수에 대한 기본 할당량이 있습니다. 기본 네트워크 인터페이스는 해당 할당량에 대해 하나로 계산됩니다. 예를 들어 기본적으로 c5.large 인스턴스에는 ENI를 3개까지만 연결할 수 있습니다. 인스턴스에 대한 기본 네트워크 인터페이스는 한 개로 계산됩니다. 인스턴스에 ENI를 2개 더 연결할 수 있습니다. awsvpc 네트워크 모드를 사용하는 각 태스크에는 ENI가 필요하므로 대개 이 인스턴스 유형에서는 이러한 태스크를 2개까지만 실행할 수 있습니다. 각 인스턴스 유형의 기본 ENI 제한에 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 유형별 네트워크 인터페이스당 IP 주소를 참조하세요.
- Amazon ECS에서는 지원되는 인스턴스 유형을 사용하여 늘어난 ENI 밀도와 함께 Amazon EC2 Linux 인스턴스를 시작할 수 있습니다. awsvpcTrunking 계정 설정에 옵트인하고 이러한 인스턴스 유형을 사용하여 Amazon EC2 Linux 인스턴스를 클러스터에 등록하는 경우 이러한 인스턴스의 ENI 할당량이 더 높습니다. 이러한 인스턴스를 이 정도 높은 할당량으로 사용하면 각 Amazon EC2 Linux 인스턴스에 더 많은 태스크를 배치할 수 있습니다. 트렁킹 기능과 함께 늘어난 ENI 밀도를 활용하려면 Amazon EC2 인스턴스에 버전 1.28.1 이상의 컨테이너 에이전트가 필요합니다. Amazon ECS 최적화 AMI를 사용하는 경우 해당 인스턴스에는 1.28.1-2 이상 버전의 ecs-init 패키지가 필요합니다. awsvpcTrunking 계정 설정 옵트인에 대한 자세한 정보는 계정 설정을 사용하여 Amazon ECS 기능에 액세스 섹션을 참조하세요. ENI 트렁킹에 대한 자세한 정보는 Amazon ECS Linux 컨테이너 인스턴스 네트워크 인터페이스 증가 섹션을 참조하세요.
- Amazon EC2 Linux 인스턴스에서 awsvpc 네트워크 모드를 사용하는 태스크를 호스팅할 때 작업 ENI에 퍼블릭 IP 주소가 부여되지 않습니다. 인터넷에 액세스하려면 NAT 게이트웨이를 사용하도록 구성된 프라이빗 서브넷에서 시작되어야 합니다. 자세한 정보는 Amazon VPC 사용 설명서의 NAT 게이트웨이 섹션을 참조하세요. 인바운드 네트워크 액세스는 프라이빗 IP 주소를 사용하는 VPC에

서 시도하거나 VPC 내에서 로드 밸런서를 통해 라우팅되어야 합니다. 퍼블릭 서브넷 내에서 시작된 태스크는 인터넷에 액세스할 수 없습니다.

- Amazon ECS는 Amazon EC2 Linux 인스턴스에 연결된 ENI만 인식합니다. 인스턴스에 ENI를 수동으로 연결한 경우, Amazon ECS는 네트워크 어댑터가 충분하지 않은 인스턴스에 태스크를 추가하려고 시도할 수 있습니다. 이로 인해 태스크 시간이 초과되고 프로비저닝 해제 상태가 된 다음 중지 상태가 될 수 있습니다. 따라서 ENI를 인스턴스에 수동으로 연결하지 않는 것이 좋습니다.
- Amazon EC2 Linux 인스턴스는 `awsvpc` 네트워크 모드로 태스크를 배치하는 것을 감안하여 `ecs.capability.task-eni` 용량으로 등록해야 합니다. `1.15.0-4` 이상 버전의 `ecs-init`을 실행 중인 인스턴스는 이 속성으로 자동 등록됩니다.
- Amazon EC2 Linux 인스턴스에서 생성 및 연결하는 ENI는 계정을 통해 수동으로 분리하거나 수정할 수 없습니다. 이는 실행 중인 태스크와 연결된 ENI가 우발적으로 삭제되는 일을 방지하기 위한 것입니다. 태스크에 대한 ENI를 해제하려면 태스크를 중지합니다.
- `awsVpcConfiguration` 네트워크 모드를 사용하는 태스크를 실행하거나 서비스를 생성할 때 `awsvpc`에서 서브넷 16개와 보안 그룹 5개만 지정할 수 있다는 제한이 있습니다. 자세한 정보는 Amazon Elastic Container Service API 참조의 AwsVpcConfiguration을 참조하세요.
- `awsvpc` 네트워크 모드에서 태스크가 시작되면 Amazon ECS 컨테이너 에이전트는 태스크 정의에 있는 컨테이너를 시작하기 전에 각 태스크에 대한 추가 `pause` 컨테이너를 생성합니다. 그런 다음 amazon-ecs-cni-plugins CNI 플러그인을 실행하여 `pause` 컨테이너의 네트워크 네임스페이스를 구성합니다. 그런 다음 에이전트는 태스크의 나머지 컨테이너를 시작하며 이러한 컨테이너는 `pause` 컨테이너의 네트워크 스택을 공유합니다. 따라서 태스크의 모든 컨테이너가 ENI의 IP 주소로 주소를 지정할 수 있으며 `localhost` 인터페이스를 통해 서로 통신할 수 있습니다.
- `awsvpc` 네트워크 모드만을 사용하는 태스크가 포함된 서비스는 Application Load Balancer 및 Network Load Balancer를 지원합니다. 이러한 서비스에 대한 대상 그룹을 생성할 때 대상 유형을 `ip`로 선택해야 합니다. `instance`는 사용하지 않습니다. 이는 `awsvpc` 네트워크 모드를 사용하는 태스크가 Amazon EC2 Linux 인스턴스가 아닌 ENI와 연결되기 때문입니다. 자세한 정보는 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산을 참조하세요.
- VPC가 사용하는 DHCP 옵션 세트를 변경하도록 업데이트된 경우, 이러한 변경 사항을 기존 작업에 적용할 수 없습니다. 이러한 변경 사항이 적용된 새 작업을 시작하고 올바르게 작동하는지 확인한 다음 기존 작업을 중지하여 네트워크 구성을 안전하게 변경합니다.

Windows 고려 사항

Windows 운영 체제를 사용할 때 고려할 사항을 다음과 같습니다.

- Amazon ECS 최적화 Windows 서버 2016 AMI를 사용하는 컨테이너 인스턴스는 awsvpc 네트워크 모드를 사용하는 태스크를 호스팅할 수 없습니다. Amazon ECS 최적화 Windows Server 2016 AMI와 awsvpc 네트워크 모드를 지원하는 Windows AMI가 포함된 클러스터가 있는 경우, awsvpc 네트워크 모드를 사용하는 태스크는 Windows 2016 Server 인스턴스에서 시작되지 않습니다. 대신, 지원하는 awsvpc네트워크 모드 인스턴스에서 시작됩니다.
- awsvpc 네트워크 모드를 사용하는 Windows 컨테이너에 CloudWatch 지표를 사용하려면 Amazon EC2 Windows 인스턴스에 버전 1.57.1 이상의 컨테이너 에이전트가 필요합니다.
- awsvpc 네트워크 모드를 사용하는 태스크 및 서비스는 사용자를 대신하여 Amazon ECS에 다른 Amazon ECS 서비스를 호출할 권한을 부여하는 AWS 서비스 연결 역할이 필요합니다. 이 역할은 클러스터를 생성하거나 AWS Management Console에서 서비스를 생성 또는 업데이트할 때 자동으로 생성됩니다. 자세한 정보는 Amazon ECS에 대해 서비스 연결 역할 사용을 참조하세요. 다음 AWS CLI 명령을 사용하여 서비스 연결 역할을 생성할 수도 있습니다.

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Amazon EC2 Windows 인스턴스는 awsvpc 네트워크 모드를 사용하는 태스크를 실행하기 위해 버전 1.54.0 이상의 컨테이너 에이전트가 필요합니다. 인스턴스를 부트스트랩할 때 awsvpc 네트워크 모드에 필요한 옵션을 구성해야 합니다. 자세한 정보는 the section called “컨테이너 인스턴스 부트스트래핑”을 참조하세요.
- Amazon ECS는 enableDnsHostnames 및 enableDnsSupport 옵션이 모두 VPC에서 사용 설정된 경우 Amazon에서 제공한 (내부) DNS 호스트 이름을 함께 사용하여 태스크의 호스트 이름을 채웁니다. 이러한 옵션이 사용 설정되지 않으면 태스크의 DNS 호스트 이름이 임의의 호스트 이름입니다. VPC DNS 설정에 대한 자세한 정보는 Amazon VPC 사용 설명서의 VPC와 함께 DNS 사용을 참조하세요.
- awsvpc 네트워크 모드를 사용하는 각 Amazon ECS 태스크는 고유한 탄력적 네트워크 인터페이스(ENI)를 받습니다. 이 ENI는 이를 호스팅하는 Amazon EC2 Windows 인스턴스에 연결되어 있습니다. Amazon EC2 Windows 인스턴스에 연결할 수 있는 네트워크 인터페이스 수에 대한 기본 할당량이 있습니다. 기본 네트워크 인터페이스는 해당 할당량에 대해 하나로 계산됩니다. 예를 들어 기본적으로 c5.large 인스턴스에는 ENI를 3개까지만 연결할 수 있습니다. 인스턴스에 대한 기본 네트워크 인터페이스는 그 중 한 개로 계산됩니다. 인스턴스에 ENI를 2개 더 연결할 수 있습니다. awsvpc 네트워크 모드를 사용하는 각 태스크에는 ENI가 필요하므로 대개 이 인스턴스 유형에서는 이러한 태스크를 2개까지만 실행할 수 있습니다. 각 인스턴스 유형의 기본 ENI 제한에 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 유형별 네트워크 인터페이스당 IP 주소를 참조하세요.
- Amazon EC2 Windows 인스턴스에서 awsvpc 네트워크 모드를 사용하는 태스크를 호스팅할 때 태스크 ENI에 퍼블릭 IP 주소가 부여되지 않습니다. 인터넷에 액세스하려면 NAT 게이트웨이를 사용하도록 구성된 프라이빗 서브넷에서 태스크를 시작합니다. 자세한 정보는 Amazon VPC 사용 설명

서의 NAT 게이트웨이 섹션을 참조하세요. 인바운드 네트워크 액세스는 프라이빗 IP 주소를 사용하는 VPC에서 시도하거나 VPC 내에서 로드 밸런서를 통해 라우팅되어야 합니다. 퍼블릭 서브넷 내에서 시작된 태스크는 인터넷에 액세스할 수 없습니다.

- Amazon ECS는 Amazon EC2 Windows 인스턴스에 연결된 ENI만 인식합니다. 인스턴스에 ENI를 수동으로 연결한 경우, Amazon ECS는 네트워크 어댑터가 충분하지 않은 인스턴스에 태스크를 추가하려고 시도할 수 있습니다. 이로 인해 태스크 시간이 초과되고 프로비저닝 해제 상태가 된 다음 중지 상태가 될 수 있습니다. 따라서 ENI를 인스턴스에 수동으로 연결하지 않는 것이 좋습니다.
- Amazon EC2 Windows 인스턴스는 `awsvpc` 네트워크 모드로 태스크를 배치하는 것을 감안하여 `ecs.capability.task-eni` 용량으로 등록해야 합니다.
- Amazon EC2 Windows 인스턴스에서 생성 및 연결하는 ENI는 수동으로 분리하거나 수정할 수 없습니다. 이는 실행 중인 태스크와 연결된 ENI가 우발적으로 삭제되는 일을 방지하기 위한 것입니다. 태스크에 대한 ENI를 해제하려면 태스크를 중지합니다.
- 태스크를 실행하거나 `awsvpc` 네트워크 모드를 사용하는 서비스를 생성할 때, `awsVpcConfiguration`에는 최대 16개의 서브넷과 5개의 보안 그룹만 지정할 수 있습니다. 자세한 정보는 Amazon Elastic Container Service API 참조의 AwsVpcConfiguration을 참조하세요.
- `awsvpc` 네트워크 모드에서 태스크가 시작되면 Amazon ECS 컨테이너 에이전트는 태스크 정의에 있는 컨테이너를 시작하기 전에 각 태스크에 대한 추가 `pause` 컨테이너를 생성합니다. 그런 다음 amazon-ecs-cni-plugins CNI 플러그인을 실행하여 `pause` 컨테이너의 네트워크 네임스페이스를 구성합니다. 그런 다음 에이전트는 태스크의 나머지 컨테이너를 시작하며 이러한 컨테이너는 `pause` 컨테이너의 네트워크 스택을 공유합니다. 따라서 태스크의 모든 컨테이너가 ENI의 IP 주소로 주소를 지정할 수 있으며 `localhost` 인터페이스를 통해 서로 통신할 수 있습니다.
- `awsvpc` 네트워크 모드만을 사용하는 태스크가 포함된 서비스는 Application Load Balancer 및 Network Load Balancer를 지원합니다. 이러한 서비스에 대한 대상 그룹을 생성할 때 대상 유형을 `instance`가 아닌 `ip`로 선택해야 합니다. 이는 `awsvpc` 네트워크 모드를 사용하는 태스크가 Amazon EC2 Windows 인스턴스가 아닌 ENI와 연결되기 때문입니다. 자세한 정보는 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산을 참조하세요.
- VPC가 사용하는 DHCP 옵션 세트를 변경하도록 업데이트된 경우, 이러한 변경 사항을 기존 작업에 적용할 수 없습니다. 이러한 변경 사항이 적용된 새 작업을 시작하고 올바르게 작동하는지 확인한 다음 기존 작업을 중지하여 네트워크 구성을 안전하게 변경합니다.
- 다음은 EC2 Windows 구성에서 `awsvpc` 네트워크 모드를 사용할 때 지원되지 않는 항목입니다.
  - 듀얼 스택 구성
  - IPv6
  - ENI 트렁킹

듀얼 스택 모드로 VPC 사용하기

VPC를 듀얼 스택 모드로 사용하는 경우 작업은 IPv4, IPv6 또는 둘 다를 통해 통신할 수 있습니다. IPv4 및 IPv6 주소는 서로 독립적입니다. 따라서 IPv4 및 IPv6에 대해 별도로 VPC에서 라우팅 및 보안을 구성해야 합니다. VPC를 듀얼 스택 모드로 구성하는 방법에 대한 자세한 정보는 Amazon VPC 사용 설명서의 IPv6로 마이그레이션하기를 참조하세요.

인터넷 게이트웨이 또는 아웃바운드 전용 인터넷 게이트웨이로 VPC를 구성한 경우, VPC 듀얼 스택 모드로 사용할 수 있습니다. 이렇게 하면 IPv6 주소에 할당되는 태스크가 인터넷 게이트웨이 또는 송신 전용 인터넷 게이트웨이를 통해 인터넷에 액세스할 수 있습니다. NAT 게이트웨이는 선택 사항입니다. 자세한 정보는 Amazon VPC 사용 설명서의 인터넷 게이트웨이 및 송신 전용 인터넷 게이트를 참조하세요.

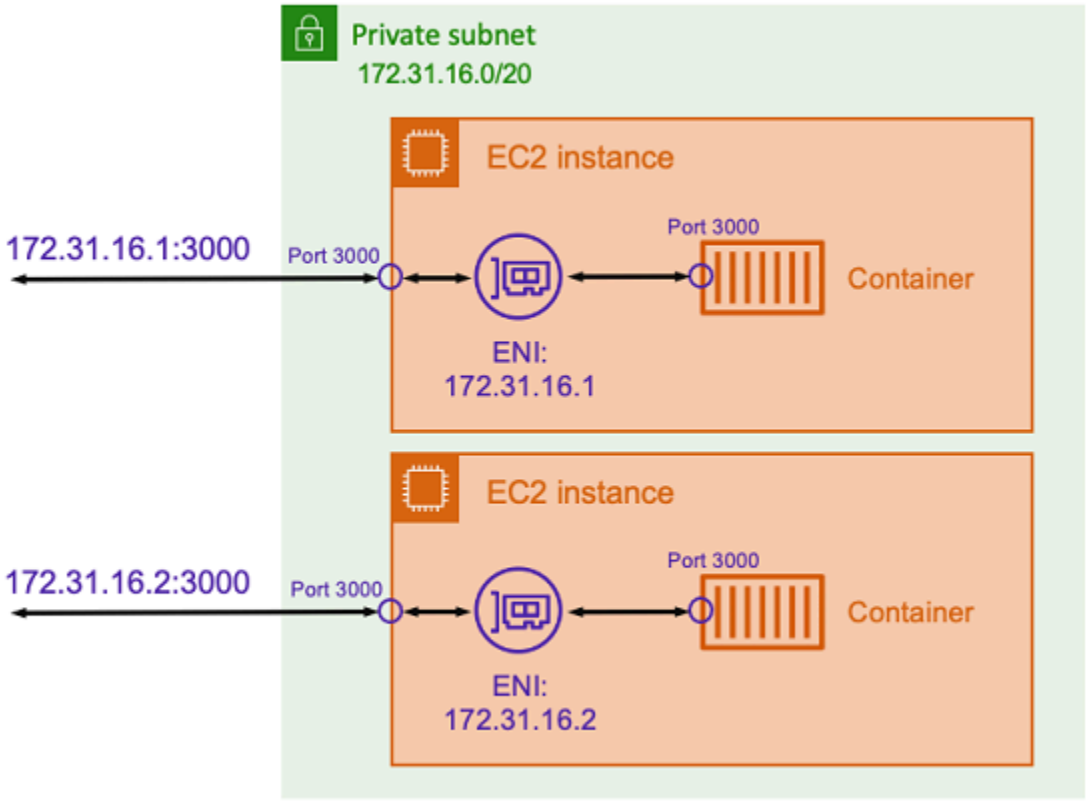
다음 조건이 충족되면 Amazon ECS 태스크에 IPv6 주소가 할당됩니다.

- 태스크를 호스팅하는 Amazon EC2 Linux 인스턴스가 버전 `1.45.0` 이상의 컨테이너 에이전트를 사용하고 있습니다. 인스턴스에서 사용 중인 에이전트 버전을 확인하고 필요한 경우 업데이트하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.
- `dualStackIPv6` 계정 설정이 활성화되어 있습니다. 자세한 정보는 계정 설정을 사용하여 Amazon ECS 기능에 액세스 섹션을 참조하세요.
- 태스크가 `awsvpc` 네트워크 모드를 사용하고 있습니다.
- VPC 및 서브넷은 IPv6으로 구성됩니다. 구성에는 지정된 서브넷에서 생성된 네트워크 인터페이스가 포함됩니다. 이중 스택 모드용 VPC 구성 방법에 대한 자세한 정보는 Amazon VPC 사용 설명서의 IPv6로 마이그레이션하기와 서브넷의 IPv6 주소 지정 속성 수정을 참조하세요.

EC2 인스턴스 네트워크 인터페이스에 Amazon ECS 컨테이너 포트 매핑

`host` 네트워크 모드는 Amazon EC2 인스턴스에서 호스팅되는 Amazon ECS 작업에만 지원됩니다. Fargate 기반 Amazon ECS를 사용하는 경우 지원되지 않습니다.

`host` 네트워크 모드는 Amazon ECS에서 지원되는 가장 기본적인 네트워크 모드입니다. 호스트 모드를 사용하면 컨테이너의 네트워킹이 컨테이너를 실행 중인 기본 호스트에 직접 연결됩니다.



위 다이어그램에 표시된 것과 비슷한 포트 3000에서 수신 대기하는 Express 애플리케이션을 사용하여 Node.js 컨테이너를 실행한다고 가정해 보겠습니다. host 네트워크 모드를 사용하는 경우 컨테이너는 기본 호스트 Amazon EC2 인스턴스의 IP 주소를 사용하여 포트 3000에서 트래픽을 수신합니다. 이 모드 사용은 권장하지 않습니다.

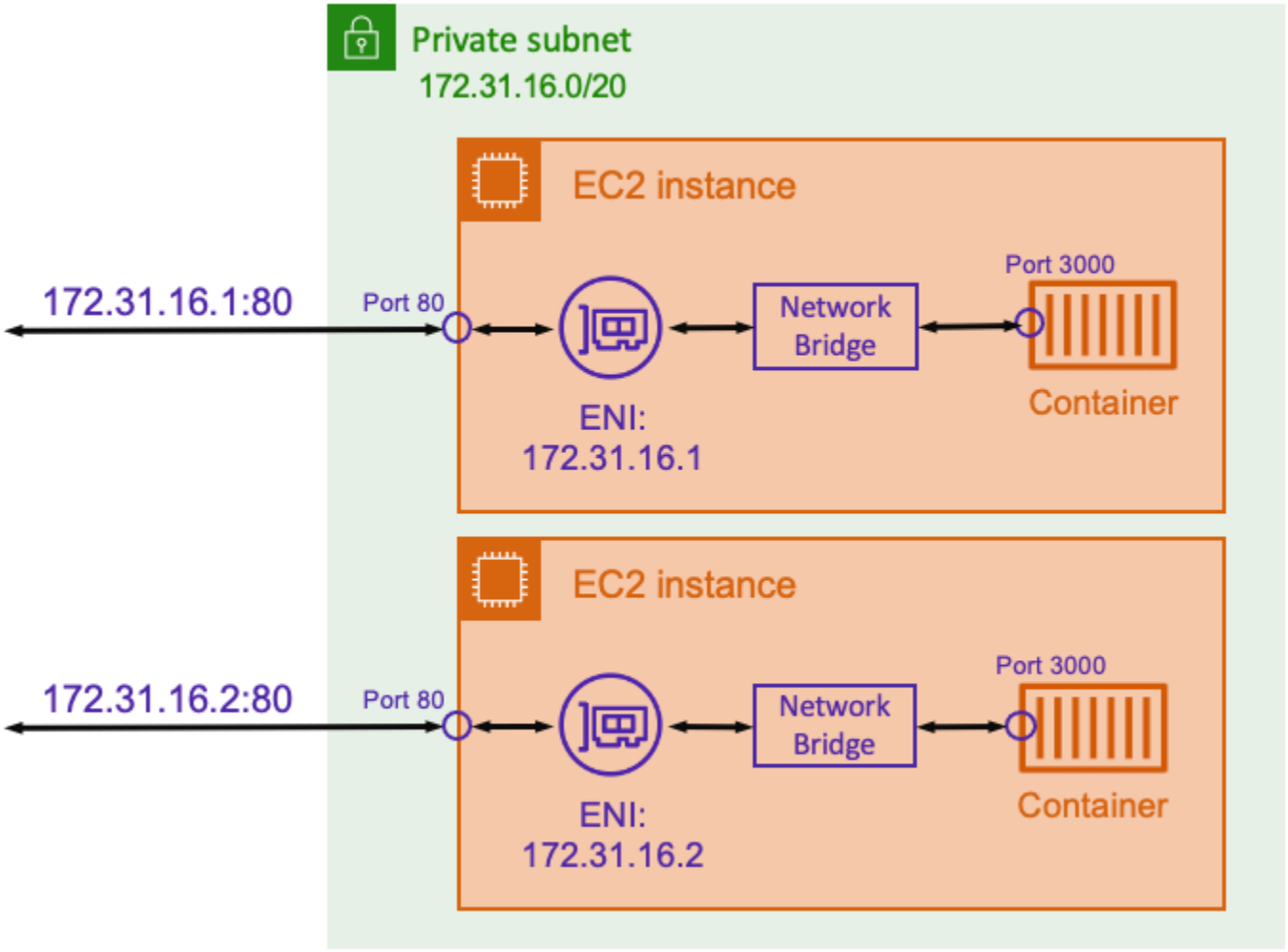
이 네트워크 모드를 사용하는 데에는 상당한 단점이 있습니다. 각 호스트에서 작업의 인스턴스화를 두 번 이상 실행할 수 없습니다. 첫 번째 작업만 Amazon EC2 인스턴스의 필요한 포트에 바인딩할 수 있기 때문입니다. host 네트워크 모드를 사용할 때는 컨테이너 포트를 다시 매핑할 방법도 없습니다. 예를 들어 애플리케이션이 특정 포트 번호를 수신해야 하는 경우 포트 번호를 직접 다시 매핑할 수 없습니다. 대신 애플리케이션 구성을 변경하여 모든 포트 충돌을 관리해야 합니다.

host 네트워크 모드를 사용할 경우 보안에도 영향을 미칠 수 있습니다. 이 모드를 사용하면 컨테이너가 호스트를 가장하고 호스트의 프라이빗 루프백 네트워크 서비스에 연결할 수 있습니다.

Amazon ECS Linux 작업에 대해 Docker의 가상 네트워크 사용

bridge 네트워크 모드는 Amazon EC2 인스턴스에서 호스팅되는 Amazon ECS 작업에만 지원됩니다.

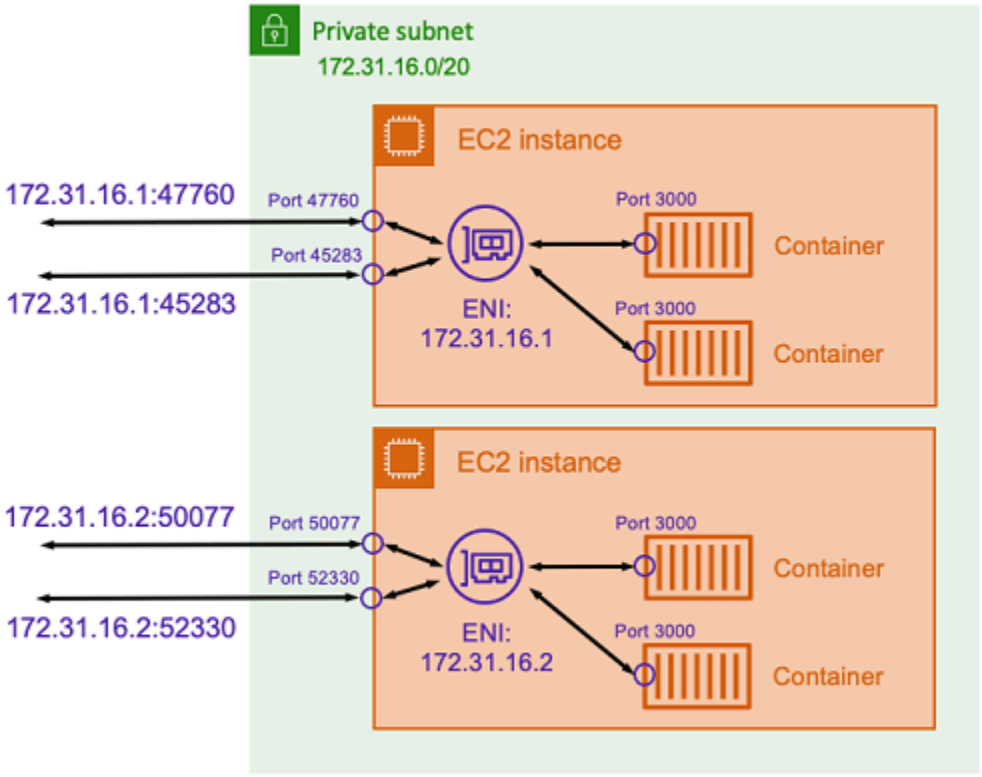
`bridge` 모드를 사용하면 가상 네트워크 브리지를 사용하여 호스트와 컨테이너 네트워킹 사이에 계층을 생성합니다. 이렇게 하면 호스트 포트를 컨테이너 포트에 다시 매핑하는 포트 매핑을 생성할 수 있습니다. 매핑은 정적이거나 동적일 수 있습니다.



정적 포트 매핑을 사용하면 컨테이너 포트에 매핑하려는 호스트 포트를 명시적으로 정의할 수 있습니다. 위 예제를 사용하면 호스트의 포트 `80`가 컨테이너의 포트 `3000`에 매핑됩니다. 컨테이너화된 애플리케이션과 통신하려면 Amazon EC2 인스턴스의 IP 주소로 포트 `80`을 통해 트래픽을 전송합니다. 컨테이너화된 애플리케이션의 관점에서 보면 인바운드 트래픽이 포트 `3000`을 통해 전달된다고 볼 수 있습니다.

트래픽 포트만 변경하려는 경우에는 정적 포트 매핑이 적합합니다. 하지만 여전히 `host` 네트워크 모드를 사용할 때와 같은 단점이 있습니다. 각 호스트에서 작업의 인스턴스화를 두 번 이상 실행할 수 없습니다. 정적 포트 매핑에서는 단일 컨테이너만 포트 80에 매핑할 수 있기 때문입니다.

이 문제를 해결하려면 다음 다이어그램과 같은 동적 포트 매핑과 함께 `bridge` 네트워크 모드를 사용하는 것이 좋습니다.



포트 매핑에서 호스트 포트를 지정하지 않으면 Docker가 임시 포트 범위에서 사용되지 않는 임의의 포트를 선택하여 컨테이너의 공용 호스트 포트로 할당하도록 할 수 있습니다. 예를 들어 컨테이너의 포트 `3000`에서 수신 대기하는 Node.js 애플리케이션에는 Amazon EC2 호스트의 `47760`과 같이 높은 번호의 임의 포트가 할당될 수 있습니다. 이렇게 하면 호스트에서 해당 컨테이너의 여러 복사본을 실행할 수 있습니다. 또한 호스트에서 각 컨테이너에 자체 포트를 할당할 수 있습니다. 컨테이너의 각 복사본은 포트 `3000`에서 트래픽을 수신합니다. 하지만 이러한 컨테이너로 트래픽을 보내는 클라이언트는 임의로 할당된 호스트 포트를 사용합니다.

Amazon ECS를 사용하면 각 작업에 임의로 할당된 포트를 추적할 수 있습니다. 이렇게 하려면 작업 IP 주소 및 포트 목록을 포함하도록 로드 밸런서 대상 그룹 및 AWS Cloud Map 서비스 검색을 자동으로 업데이트합니다. 따라서 동적 포트가 있는 `bridge` 모드를 사용하여 운영되는 서비스를 더 쉽게 사용할 수 있습니다.

그러나 `bridge` 네트워크 모드를 사용할 때의 한 가지 단점은 서비스 간 통신을 차단하기가 어렵다는 것입니다. 서비스는 사용되지 않는 임의의 포트에 할당될 수 있으므로 호스트 간에 넓은 포트 범위를 열어야 합니다. 하지만 특정 서비스가 다른 특정 서비스 하나와만 통신할 수 있도록 특정 규칙을 만드는 것은 쉽지 않습니다. 서비스에는 보안 그룹 네트워킹 규칙에 사용할 특정 포트가 없습니다.

Fargate 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션

기본적으로 Fargate의 모든 Amazon ECS 태스크에는 기본 프라이빗 IP 주소와 함께 탄력적 네트워크 인터페이스(ENI)가 제공됩니다. 퍼블릭 서브넷을 사용할 때 필요에 따라 태스크의 ENI에 퍼블릭 IP 주소를 할당할 수도 있습니다. VPC가 듀얼 스택 모드를 사용하도록 구성되어 있고 IPv6 CIDR 블록과 함께 서브넷을 사용하는 경우 작업의 ENI도 IPv6 주소를 수신합니다. 한 태스크에는 한 번에 하나의 ENI만 연결할 수 있습니다. 또한 같은 태스크에 속한 컨테이너는 `localhost` 인터페이스로 통신할 수 있습니다. VPC와 서브넷에 대한 자세한 내용은 Amazon VPC 사용 설명서의 Amazon VPC 작동 방식을 참조하세요.

Fargate의 태스크가 컨테이너 이미지를 풀링하려면 해당 태스크에 인터넷으로 연결되는 경로가 있어야 합니다. 태스크에 인터넷이 연결되는 경로가 있는지 확인하는 방법은 다음과 같습니다.

- 퍼블릭 서브넷을 사용할 때 태스크의 ENI에 퍼블릭 IP 주소를 할당할 수 있습니다.
- 프라이빗 서브넷을 사용할 때 서브넷은 NAT 게이트웨이를 연결할 수 있습니다.
- Amazon ECR에서 호스팅되는 컨테이너 이미지를 사용하는 경우 인터페이스 VPC 엔드포인트를 사용하도록 Amazon ECR을 구성할 수 있습니다. 그러면 태스크의 프라이빗 IPv4 주소를 통해 이미지 가져오기가 실행됩니다. 자세한 정보는 Amazon Elastic Container Registry 사용 설명서의 Amazon ECR 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

각 태스크는 고유한 ENI를 받으므로 VPC 흐름 로그와 같은 네트워킹 기능을 활용하여 태스크에서 주고받는 트래픽을 모니터링할 수도 있습니다. 자세한 정보는 Amazon VPC 사용 설명서의 VPC 흐름 로그를 참조하세요.

AWS PrivateLink를 이용할 수도 있습니다. 프라이빗 IP 주소를 통해 Amazon ECS API에 액세스할 수 있도록 VPC 인터페이스 엔드포인트를 구성할 수 있습니다. AWS PrivateLink는 VPC와 Amazon ECS 간의 모든 네트워크 트래픽을 Amazon 네트워크로 제한합니다. 인터넷 게이트웨이, NAT 디바이스 또는 가상 프라이빗 게이트웨이가 필요 없습니다. 자세한 내용은 Amazon ECS 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

AWS CloudFormation에서 `NetworkConfiguration` 리소스를 사용하는 방법의 예는 the section called “템플릿 AWS CloudFormation 예제”를 참조하세요.

생성된 ENI는 AWS Fargate로 완전 관리됩니다. 또한 Fargate에 대한 권한을 부여하는 데 사용되는 관련 IAM 정책이 있습니다. Fargate 플랫폼 버전 `1.4.0` 이상을 사용하는 태스크의 경우 태스크는 단일 ENI(태스크 ENI라고 함)를 수신하고 모든 네트워크 트래픽은 VPC 내의 해당 ENI를 통해 흐릅니다. 이 트래픽은 VPC 흐름 로그에 기록됩니다. 버전이 `1.3.0` 이전인 Fargate 플랫폼을 사용하는 태스크의 경우 태스크 ENI 외에도 VPC 흐름 로그에 표시되지 않는 일부 네트워크 트래픽에 사용되는 별도의

Fargate 소유 ENI를 수신합니다. 다음 표는 네트워크 트래픽 동작과 각 플랫폼 버전에 필요한 IAM 정책에 대해 설명합니다.

작업	Linux 플랫폼 버전 `1.3.0` 이하의 트래픽 흐름	Linux 플랫폼 버전 `1.4.0`의 트래픽 흐름	Windows 플랫폼 버전 `1.0.0`의 트래픽 흐름	IAM 권한
Amazon ECR 로그인 자격 증명 검색	Fargate 소유 ENI	태스크 ENI	태스크 ENI	태스크 실행 IAM 역할
이미지 가져오기	태스크 ENI	태스크 ENI	태스크 ENI	태스크 실행 IAM 역할
로그 드라이버를 통해 로그 전송	태스크 ENI	태스크 ENI	태스크 ENI	태스크 실행 IAM 역할
Amazon ECS용 FireLens를 통해 로그 전송	태스크 ENI	태스크 ENI	태스크 ENI	태스크 IAM 역할
Secrets Manager 또는 Systems Manager에서 암호 검색	Fargate 소유 ENI	태스크 ENI	태스크 ENI	태스크 실행 IAM 역할
Amazon EFS 파일 시스템 트래픽	사용할 수 없음	태스크 ENI	태스크 ENI	태스크 IAM 역할
애플리케이션 트래픽	태스크 ENI	태스크 ENI	태스크 ENI	태스크 IAM 역할

고려 사항

태스크 네트워킹을 사용할 때는 다음 항목을 고려하세요.

- Amazon ECS 서비스 연결 역할은 Amazon ECS에 사용자를 대신해서 다른 AWS 서비스를 호출할 수 있는 권한을 제공해야 합니다. 이 역할은 클러스터를 생성하거나 AWS Management Console에

서 서비스를 생성 또는 업데이트할 때 생성됩니다. 자세한 정보는 Amazon ECS에 대해 서비스 연결 역할 사용을 참조하세요. 다음 AWS CLI 명령을 사용하여 서비스 연결 역할을 생성할 수도 있습니다.

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Amazon ECS는 `enableDnsHostnames` 및 `enableDnsSupport` 옵션이 모두 VPC에서 사용 설정된 경우 Amazon에서 제공한 DNS 호스트 이름을 함께 사용하여 태스크의 호스트 이름을 채웁니다. 이러한 옵션이 사용 설정되지 않으면 태스크의 DNS 호스트 이름이 임의의 호스트 이름이 설정됩니다. VPC DNS 설정에 대한 자세한 정보는 Amazon VPC 사용 설명서의 VPC와 함께 DNS 사용을 참조하세요.
- `awsVpcConfiguration`에는 최대 16개의 서브넷과 5개의 보안 그룹만 지정할 수 있습니다. 자세한 정보는 Amazon Elastic Container Service API 참조의 AwsVpcConfiguration을 참조하세요.
- Fargate에서 생성 및 연결하는 ENI는 수동으로 분리하거나 수정할 수 없습니다. 이는 실행 중인 태스크와 연결된 ENI가 우발적으로 삭제되는 일을 방지하기 위한 것입니다. 태스크에 대한 ENI를 해제하려면 태스크를 중지합니다.
- VPC 서브넷이 사용하는 DHCP 옵션 세트를 변경하도록 업데이트된 경우, VPC를 사용하는 기존 작업에도 이러한 변경 사항을 적용할 수 없습니다. 새 작업을 시작하면, 새 변경 사항을 테스트하는 동안 원활하게 마이그레이션할 수 있는 새 설정을 수신한 다음 롤백이 필요하지 않은 경우 이전 작업을 중지합니다.
- 다음은 Linux의 경우 Fargate 플랫폼 버전 `1.4.0` 이상, Windows의 경우 Fargate 플랫폼 버전 `1.0.0`에서 실행되는 태스크에 적용됩니다. 듀얼 스택 서브넷에서 시작된 태스크는 IPv4 주소와 IPv6 주소를 받습니다.
- 플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0`(Windows)을 사용하는 태스크의 경우 태스크 ENI가 점보 프레임을 지원합니다. 네트워크 인터페이스는 단일 프레임 내에 맞는 가장 큰 페이로드 크기인 최대 전송 단위(MTU)로 구성됩니다. MTU가 클수록 단일 프레임 내에 더 많은 애플리케이션 페이로드가 들어갈 수 있으므로 프레임당 오버헤드가 줄어들고 효율성이 향상됩니다. 점보 프레임을 지원하면 태스크와 대상 간의 네트워크 경로가 점보 프레임을 지원할 때 오버헤드가 줄어듭니다.
- Fargate 시작 유형을 사용하는 태스크의 서비스는 Application Load Balancer 및 Network Load Balancer만 지원합니다. Classic Load Balancer는 지원되지 않습니다. 대상 그룹을 생성할 때 대상 유형을 `instance`가 아닌 `ip`로 선택해야 합니다. 자세한 정보는 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산을 참조하세요.

듀얼 스택 모드로 VPC 사용하기

VPC를 듀얼 스택 모드로 사용하는 경우 태스크는 IPv4나 IPv6, 또는 둘 다를 통해 통신할 수 있습니다. IPv4 및 IPv6 주소는 서로 독립적이며 IPv4 및 IPv6에 대해 별도로 VPC의 라우팅 및 보안을 구성해야

합니다. VPC를 듀얼 스택 모드로 구성하는 방법에 대한 자세한 정보는 Amazon VPC 사용 설명서의 IPv6로 마이그레이션하기를 참조하세요.

다음 조건이 충족되면 Fargate의 Amazon ECS 태스크에 IPv6 주소가 할당됩니다.

- 작업을 시작하려는 리전에서 IAM 보안 주체가 작업을 시작할 수 있도록 Amazon ECS `dualStackIPv6` 계정 설정이 켜져 있습니다(`enabled`). 이 설정은 API 또는 AWS CLI를 사용해서만 수정할 수 있습니다. 계정 기본 설정을 설정하여 계정의 특정 IAM 보안 주체에 대해 이 설정을 켜거나 계정 전체에서 이 설정을 켤 수 있습니다. 자세한 내용은 계정 설정을 사용하여 Amazon ECS 기능에 액세스 섹션을 참조하세요.
- VPC와 서브넷이 IPv6에 대해 활성화되어 있습니다. VPC를 듀얼 스택 모드로 구성하는 방법에 대한 자세한 정보는 Amazon VPC 사용 설명서의 IPv6로 마이그레이션하기를 참조하세요.
- 서브넷에서 IPv6 주소를 자동 할당할 수 있습니다. 서브넷을 구성하는 방법에 대한 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 서브넷의 IPv6 주소 지정 속성 수정을 참조하세요.
- 작업 또는 서비스가 Fargate 플랫폼 버전 `1.4.0` 이상(Linux)을 사용합니다.

인터넷 게이트웨이 또는 아웃바운드 전용 인터넷 게이트웨이로 VPC를 구성하면 IPv6 주소가 할당된 Fargate의 Amazon ECS 작업이 인터넷에 액세스할 수 있습니다. NAT 게이트웨이는 필요하지 않습니다. 자세한 정보는 Amazon VPC 사용 설명서의 인터넷 게이트웨이 및 송신 전용 인터넷 게이트를 참조하세요.

Amazon ECS 작업에 대한 스토리지 옵션

Amazon ECS는 요구 사항에 따라 유연하고 비용 대비 효율적이며 사용이 쉬운 데이터 스토리지 옵션을 제공합니다. Amazon ECS는 컨테이너에 대해 다음과 같은 데이터 볼륨 옵션을 지원합니다.

데이터 볼륨	지원되는 시작 유형	지원되는 운영 체제	스토리지 지속성	사용 사례
Amazon Elastic Block Store(Amazon EBS)	Fargate, Amazon EC2	Linux	독립 실행형 작업에 연결된 경우 지속될 수 있습니다. 서비스에서 유지 관리하는 작업에 연결된 경우 임시입니다.	Amazon EBS 볼륨은 데이터 집약적인 컨테이너화된 워크로드에 대해 비용 효율적이며 내구성이 뛰어난 고성능 블록 스토리지를 제

데이터 볼륨	지원되는 시작 유형	지원되는 운영 체제	스토리지 지속성	사용 사례
				공합니다. 일반적인 사용 사례로, 데이터베이스, 가상 데스크톱, 루트 볼륨과 같은 트랜잭션 워크로드와 로그 처리 및 ETL 워크로드와 같은 처리량 집약적 워크로드가 있습니다. 자세한 내용은 Amazon ECS에서 Amazon EBS 볼륨 사용 섹션을 참조하세요.

데이터 볼륨	지원되는 시작 유형	지원되는 운영 체제	스토리지 지속성	사용 사례
Amazon Elastic File System(Amazon EFS)	Fargate, Amazon EC2	Linux	지속적	Amazon EFS 볼륨은 파일을 추가하고 제거할 때 자동으로 확장 및 축소되는, Amazon ECS 작업에 사용할 수 있는 간단하고 확장 가능하며 영구적인 공유 파일 스토리지를 제공합니다. Amazon EFS 볼륨은 동시성을 지원하며, 수평적으로 규모가 조정되고 짧은 지연 시간, 높은 처리량, 읽기 후 쓰기 일관성과 같은 스토리지 기능이 필요한 컨테이너화된 애플리케이션에 유용합니다. 일반적인 사용 사례로, 데이터 분석, 미디어 처리, 콘텐츠 관리, 웹 지원과 같은 워크로드가 있습니다. 자세한 내용은 Amazon ECS에서 Amazon EFS

데이터 볼륨	지원되는 시작 유형	지원되는 운영 체제	스토리지 지속성	사용 사례
				볼륨 사용 섹션을 참조하세요.

데이터 볼륨	지원되는 시작 유형	지원되는 운영 체제	스토리지 지속성	사용 사례
Amazon FSx for Windows File Server	Amazon EC2	Windows	지속적	FSx for Windows File Server 볼륨은 영구적이고 분산 및 공유된 고정 파일 스토리지가 필요한 Windows 작업을 프로비저닝하는 데 사용할 수 있는 완전관리형 Windows 파일 서버를 제공합니다. 일반적인 사용 사례로, 애플리케이션 출력을 저장하기 위한 영구 스토리지로 로컬 폴더가 필요할 수 있는.NET 애플리케이션이 있습니다. Amazon FSx for Windows File Server는 컨테이너에서 로컬 폴더를 제공하므로 SMB 공유로 지원되는 동일한 파일 시스템에서 여러 컨테이너가 읽고 쓸 수 있습니다. 자세한 내용은 Amazon ECS에서 FSx

데이터 볼륨	지원되는 시작 유형	지원되는 운영 체제	스토리지 지속성	사용 사례
				for Windows File Server 볼륨 사용 섹션을 참조하세요.

데이터 볼륨	지원되는 시작 유형	지원되는 운영 체제	스토리지 지속성	사용 사례
Docker 볼륨	Amazon EC2	Windows, Linux	지속적	Docker 볼륨은 Docker 컨테이너 런타임의 기능으로, 호스트의 파일 시스템에서 디렉터리를 탑재하여 컨테이너가 데이터를 유지할 수 있도록 합니다. Docker 볼륨 드라이버(플러그인이라고도 함)는 컨테이너 볼륨을 외부 스토리지 시스템과 통합하는 데 사용됩니다. Docker 볼륨은 타사 드라이버 또는 기본 제공 `local` 드라이버로 관리할 수 있습니다. Docker 볼륨의 일반적인 사용 사례로, 동일한 컨테이너 인스턴스의 서로 다른 컨테이너에 있는 여러 위치에서 볼륨 공유 또는 영구 데이터 볼륨 공유가 있습니다. 자세한 내용은 Amazon ECS

데이터 볼륨	지원되는 시작 유형	지원되는 운영 체제	스토리지 지속성	사용 사례
				에서 Docker 볼륨 사용 섹션을 참조하세요.
바인드 탑재	Fargate, Amazon EC2	Windows, Linux	임시	바인드 탑재는 컨테이너에 탑재된 호스트의 파일 또는 디렉터리(예: Amazon EC2 인스턴스 또는 AWS Fargate)로 구성됩니다. 바인드 탑재의 일반적인 사용 사례로, 같은 작업에서 다른 컨테이너와 소스 컨테이너의 볼륨 공유 또는 하나 이상의 컨테이너에서 호스트 볼륨 또는 빈 볼륨 탑재가 있습니다. 자세한 내용은 Amazon ECS에서 바인드 탑재 사용 섹션을 참조하세요.

Amazon ECS에서 Amazon EBS 볼륨 사용

Amazon Elastic Block Store(Amazon EBS) 볼륨은 데이터 집약적인 워크로드를 위한 가용성, 비용 효율성 및 내구성이 뛰어난 고성능 블록 스토리지를 제공합니다. Amazon EBS 볼륨은 처리량이 많고 트랜잭션 집약적인 애플리케이션을 위해 Amazon ECS 작업과 함께 사용할 수 있습니다.

독립 실행형 작업 실행 중에 작업에 EBS 볼륨 하나를 연결하는 데 사용할 구성을 제공할 수 있습니다. 서비스 생성 또는 업데이트 중에 Amazon ECS 서비스에서 관리하는 각 작업에 작업당 하나의 EBS 볼륨을 연결하는 데 사용할 구성을 제공할 수 있습니다.

작업 정의 대신 시작 시 볼륨 구성을 제공함으로써 특정 데이터 볼륨 유형이나 특정 EBS 볼륨 설정으로 제한되지 않는 작업 정의를 생성할 수 있습니다. 그런 다음, 다양한 런타임 환경에서 작업 정의를 재사용할 수 있습니다. 예를 들어, 사전 프로덕션 환경보다 프로덕션 워크로드에 대해 배포하는 동안 더 많은 처리량을 제공할 수 있습니다.

Amazon ECS 작업에 연결된 Amazon EBS 볼륨은 사용자를 대신하여 Amazon ECS에서 관리합니다. 볼륨은 데이터를 보호하기 위해 AWS Key Management Service(AWS KMS) 키로 암호화할 수 있습니다. 첨부할 빈 새 볼륨을 구성하거나 스냅샷을 사용하여 기존 볼륨에서 데이터를 로드할 수 있습니다. 스냅샷을 사용하여 볼륨을 구성할 때 `volumeInitializationRate`를 MiB/s 단위로 지정하면 예측 가능한 시간 내에 스냅샷에서 데이터를 가져와서 완전히 초기화된 볼륨을 생성할 수 있습니다. 볼륨 초기화에 대한 자세한 내용은 Amazon EBS 사용 설명서의 Amazon EBS 볼륨 초기화를 참조하세요. Amazon EBS 볼륨 구성에 대한 자세한 내용은 Amazon ECS 작업 정의에서 볼륨 구성을 시작 시간으로 연기 및 Amazon ECS 배포 시 Amazon EBS 볼륨 구성 지정 섹션을 참조하세요.

볼륨 성능을 모니터링하기 위해 Amazon CloudWatch 지표를 사용할 수도 있습니다. Amazon EBS 볼륨의 Amazon ECS 지표에 대한 자세한 내용은 Amazon ECS CloudWatch 지표 및 Amazon ECS Container Insights 지표를 참조하세요.

Amazon ECS를 지원하는 모든 상업용 및 중국 AWS 리전에서 Amazon EBS 볼륨을 태스크에 연결하는 기능이 지원됩니다.

Amazon EBS 볼륨에 대한 자세한 내용은 Amazon EBS 사용 설명서의 Amazon EBS 볼륨을 참조하세요.

지원되는 운영 체제 및 시작 유형

다음 표는 지원되는 운영 체제 및 시작 유형 구성을 제공합니다.

시작 유형	Linux	Windows
Fargate	Amazon EBS 볼륨은 플랫폼 버전 `1.4.0` 이상(Linux)에서 지원됩니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.	지원되지 않음

시작 유형	Linux	Windows
EC2	Amazon ECS 최적화 Amazon Machine Image(AMI)가 있는 Nitro 기반 Linux 인스턴스에서 호스팅되는 태스크. 인스턴스 유형에 대한 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 유형을 참조하세요. Amazon EBS 볼륨은 ECS 최적화 AMI `20231219` 이상에서 지원됩니다. 자세한 내용은 Amazon ECS 최적화 AMI 메타데이터 검색을 참조하세요.	Amazon ECS 최적화 Amazon Machine Image(AMI)가 있는 Nitro 기반 Linux 인스턴스에서 호스팅되는 태스크. 인스턴스 유형에 대한 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 유형을 참조하세요. Amazon EBS 볼륨은 ECS 최적화 AMI `20241017` 이상에서 지원됩니다. 자세한 내용은 Amazon ECS 최적화 Windows AMI 메타데이터 검색을 참조하세요.

고려 사항

Amazon EBS 볼륨을 사용할 때 다음을 고려합니다.

- `use1-az3` 가용 영역에서 Fargate 시작 유형 Amazon ECS 태스크에 연결하도록 Amazon EBS 볼륨을 구성할 수 없습니다.
- 마그네틱(`standard`) Amazon EBS 볼륨 유형은 Fargate에 호스팅된 작업에서 지원되지 않습니다. Amazon EBS 볼륨에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EBS volumes를 참조하세요.
- Amazon ECS 인프라 IAM 역할은 배포 시 볼륨을 구성하는 서비스 또는 독립 실행형 작업을 생성할 때 필요합니다. AWS 관리형 `AmazonECSInfrastructureRolePolicyForVolumes` IAM 정책을 역할에 연결하거나 관리형 정책을 지침으로 사용하여 특정 요구 사항에 맞는 권한을 보유한 자체 정책을 생성하고 연결할 수 있습니다. 자세한 내용은 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.
- Amazon EBS 볼륨을 최대 한 개만 각 Amazon ECS 작업에 연결할 수 있으며, 새 볼륨이어야 합니다. 기존 Amazon EBS 볼륨은 작업에 연결할 수 없습니다. 하지만 배포 시 기존 볼륨의 스냅샷을 사용하여 새 Amazon EBS 볼륨을 구성할 수 있습니다.
- Amazon ECS 서비스에서 Amazon EBS 볼륨을 사용하려면 배포 컨트롤러가 'ECS'여야 합니다. 이 배포 컨트롤러를 사용하는 경우 롤링 및 블루/그린 배포 전략이 모두 지원됩니다.

- 작업의 컨테이너에서 탑재된 Amazon EBS 볼륨에 쓰려면 컨테이너를 루트 사용자로 실행해야 합니다.
- Amazon ECS는 예약된 태그 `AmazonECSCreated` 및 `AmazonECSManaged`를 연결된 볼륨에 자동으로 추가합니다. 볼륨에서 이러한 태그를 제거하면 Amazon ECS에서 사용자를 대신하여 볼륨을 관리할 수 없습니다. Amazon EBS 볼륨 태그 지정에 대한 자세한 내용은 Amazon EBS 볼륨 태그 지정을 참조하세요. Amazon ECS 리소스 태그 지정에 대한 자세한 내용은 Amazon ECS 리소스 태그 지정을 참조하세요.
- 파티션이 포함된 Amazon EBS 볼륨 스냅샷에서 볼륨을 프로비저닝하는 방식은 지원되지 않습니다.
- 서비스에서 관리하는 작업에 연결된 볼륨은 보존되지 않으며 작업 종료 시 항상 삭제됩니다.
- AWS Outposts에서 실행 중인 Amazon ECS 작업에 연결하도록 Amazon EBS 볼륨을 구성할 수 없습니다.

Amazon ECS 작업 정의에서 볼륨 구성을 시작 시간으로 연기

작업에 연결하도록 Amazon EBS 볼륨을 구성하려면 작업 정의에 탑재 지점 구성을 지정하고 볼륨 이름을 지정해야 합니다. 또한 작업 정의에서 연결하도록 Amazon EBS 볼륨을 구성할 수 없으므로 `configuredAtLaunch`를 `true`로 설정해야 합니다. 대신 Amazon EBS 볼륨은 배포 중에 연결하도록 구성됩니다.

AWS Command Line Interface(AWS CLI)를 사용하여 작업 정의를 등록하려면 템플릿을 JSON 파일로 저장하고, 파일을 `register-task-definition` 명령의 입력으로 전달합니다.

AWS Management Console을 사용하여 작업 정의를 생성하고 등록하려면 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

다음 작업 정의는 작업 정의에서 `mountPoints` 및 `volumes` 객체의 구문을 표시합니다. 작업 정의 파라미터에 대한 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터 섹션을 참조하세요. 이 예제를 사용하려면 *`user input placeholders`*를 사용자의 정보로 대체합니다.

Linux

```
{
    "family": "mytaskdef",
    "containerDefinitions": [
        {
            "name": "nginx",
            "image": "public.ecr.aws/nginx/nginx:latest",
            "networkMode": "awsvpc",
           "portMappings": [
```

```
                {
                    "name": "nginx-80-tcp",
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp",
                    "appProtocol": "http"
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "myEBSVolume",
                    "containerPath": "/mount/ebs",
                    "readOnly": true
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "myEBSVolume",
            "configuredAtLaunch": true
        }
    ],
    "requiresCompatibilities": [
        "FARGATE", "EC2"
    ],
    "cpu": "1024",
    "memory": "3072",
    "networkMode": "awsvpc"
}
```

Windows

```
{
    "family": "mytaskdef",
     "memory": "4096",
     "cpu": "2048",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["EC2"]
    "containerDefinitions": [
        {
```

```
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 443,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "myEBSVolume",
                    "containerPath": "drive:\ebs",
                    "readOnly": true
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "myEBSVolume",
            "configuredAtLaunch": true
        }
    ],
    "requiresCompatibilities": [
        "FARGATE", "EC2"
    ],
    "cpu": "1024",
    "memory": "3072",
    "networkMode": "awsvpc"
```

```
}
```

`mountPoints`

유형: 객체 배열

필수 여부: 아니요

컨테이너에서 데이터 볼륨의 탑재 지점입니다. 이 파라미터는 create-container Docker API의 `Volumes` 및 docker run에 대한 `--volume` 옵션에 매핑됩니다.

Windows 컨테이너는 전체 디렉터리를 동일한 드라이브에 `$env:ProgramData`로 마운트할 수 있습니다. Windows 컨테이너는 디렉터리를 다른 드라이브에 탑재할 수 없으며, 탑재 지점은 여러 드라이브에 걸쳐 사용할 수 없습니다. Amazon EBS 볼륨을 Amazon ECS 작업에 직접 연결하려면 탑재 지점을 지정해야 합니다.

`sourceVolume`

유형: 문자열

필수 항목 여부: 예(`mountPoints` 사용 시)

탑재할 볼륨의 이름입니다.

`containerPath`

유형: 문자열

필수 항목 여부: 예(`mountPoints` 사용 시)

볼륨을 탑재할 컨테이너의 경로입니다.

`readOnly`

유형: 부울

필수 여부: 아니요

이 값이 `true`일 경우 컨테이너에는 볼륨에 대한 읽기 전용 액세스가 부여됩니다. 이 값이 `false`일 경우 컨테이너는 볼륨에 쓸 수 있습니다. 기본값은 `false`입니다.

Windows 운영 체제를 실행하는 EC2 인스턴스에서 실행되는 태스크의 경우 값을 기본값인 `false`로 둡니다.

`name`

유형: 문자열

필수 여부: 아니요

볼륨의 이름입니다. 최대 255자의 문자(대문자 및 소문자), 숫자, 하이(-) 및 밑줄(_)이 허용됩니다. 이 이름은 컨테이너 정의 `mountPoints` 객체의 `sourceVolume` 파라미터에서 참조됩니다.

`configuredAtLaunch`

유형: 부울

필수: 예. EBS 볼륨을 작업에 직접 연결하려는 경우 필수입니다.

시작 시 볼륨을 구성할 수 있는지 여부를 지정합니다. `true`로 설정하면 독립 실행형 작업을 실행하거나 서비스를 생성 또는 업데이트할 때 볼륨을 구성할 수 있습니다. `false`로 설정하면 작업 정의에서 다른 볼륨 구성을 제공할 수 없습니다. Amazon EBS 볼륨을 작업에 연결하도록 구성하려면 이 파라미터를 제공하고 `true`로 설정해야 합니다.

Amazon ECS 태스크에 연결된 Amazon EBS 볼륨에 저장된 데이터 암호화

AWS Key Management Service(AWS KMS)를 사용하여 데이터를 보호하는 암호화 키를 생성하고 관리할 수 있습니다. Amazon EBS 볼륨은 AWS KMS keys를 사용하여 저장 중에 암호화됩니다. 다음 유형의 데이터가 암호화됩니다.

- 볼륨에 저장된 데이터
- 디스크 I/O
- 볼륨에서 생성된 스냅샷
- 암호화된 스냅샷에서 생성된 새 볼륨

태스크에 연결된 Amazon EBS 볼륨은 별칭이 `alias/aws/ebs`인 기본 AWS 관리형 키 또는 볼륨 구성에 지정된 대칭 고객 관리형 키를 사용하여 암호화할 수 있습니다. 기본 AWS 관리형 키는 AWS 리전당 AWS 계정마다 고유하며 자동으로 생성됩니다. 대칭 고객 관리형 키를 생성하려면 AWS KMS 개발자 안내서의 대칭 암호화 KMS 키 생성의 단계를 따르세요.

특정 AWS 리전에서 생성 후 태스크에 연결된 모든 새 볼륨이 계정에 구성된 KMS 키를 사용하여 암호화되도록 기본적으로 Amazon EBS 암호화를 구성할 수 있습니다. Amazon EBS 암호화 및 암호화 기본 제공에 대한 자세한 내용은 Amazon EBS 사용 설명서의 Amazon EBS 암호화를 참조하세요.

또한 클러스터를 생성하거나 업데이트할 때 Amazon ECS 관리형 스토리지에 대한 Amazon ECS 클러스터 수준 암호화를 설정할 수 있습니다. 클러스터 수준 암호화는 특정 클러스터에서 실행되는 태스크에 연결된 모든 Amazon EBS 볼륨을 클러스터 수준에서 지정된 KMS 키를 사용하여 암호화하는 데

사용할 수 있습니다. 클러스터 수준에서 암호화를 구성하는 방법에 대한 자세한 내용은 Amazon ECS API 참조의 ManagedStorageConfiguration을 참조하세요.

해당 키 조합을 구성할 수 있습니다. KMS 키의 우선순위는 다음과 같습니다.

1. 볼륨 구성에 지정된 KMS 키. 볼륨 구성에서 KMS 키를 지정하면 Amazon EBS 기본값 및 클러스터 수준에서 지정된 모든 KMS 키가 재정의됩니다.
2. 클러스터 수준에서 지정된 KMS 키. Amazon ECS 관리형 스토리지의 클러스터 수준 암호화를 위한 KMS 키를 지정하면 Amazon EBS 기본 암호화는 재정의하지만 볼륨 구성에 지정된 KMS 키는 재정의하지 않습니다.
3. Amazon EBS 기본 암호화. 기본 암호화는 볼륨 구성에서 클러스터 수준 KMS 키 또는 키를 지정하지 않은 경우에 적용됩니다. 기본적으로 Amazon EBS 암호화를 활성화하는 경우 기본값은 기본적으로 암호화에 지정하는 KMS 키입니다. 그렇지 않으면 기본값은 별칭이 `alias/aws/ebs`인 AWS 관리형 키입니다.

> **Note**
>
> 볼륨 구성에서 `encrypted`를 `false`로 설정하고 클러스터 수준 KMS 키를 지정하지 않은 채 기본적으로 Amazon EBS 암호화를 활성화하면 볼륨은 기본적으로 Amazon EBS 암호화에 지정된 키로 계속 암호화됩니다.

고객 관리형 KMS 키 정책

고객 관리형 키를 사용하여 태스크에 연결된 EBS 볼륨을 암호화하려면 볼륨 구성에 사용하는 IAM 역할이 키를 사용하는 데 필요한 권한을 보유하도록 KMS 키 정책을 구성해야 합니다. 키 정책에는 `kms:CreateGrant` 및 `kms:GenerateDataKey*` 권한이 포함되어야 합니다. `kms:ReEncryptTo` 및 `kms:ReEncryptFrom` 권한은 스냅샷을 사용하여 생성된 볼륨을 암호화하는 데 필요합니다. 연결을 위해 새로운 빈 볼륨만 구성하고 암호화하려는 경우 `kms:ReEncryptTo` 및 `kms:ReEncryptFrom` 권한을 제외할 수 있습니다.

다음 JSON 코드 조각은 KMS 키 정책에 연결할 수 있는 키 정책 명령을 보여줍니다. 이한 명령을 사용하면 EBS 볼륨을 암호화하려는 경우 Amazon ECS에서 키를 사용할 수 있는 액세스 권한이 제공됩니다. 정책 명령 예제를 사용하려면 *`user input placeholders`*를 사용자 정보로 바꿉니다. 항상 그렇듯이 필요한 권한만 구성합니다.

```
{
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/ecsInfrastructureRole" },
```

```
      "Action": "kms:DescribeKey",
      "Resource":"*"
    },
    {
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/ecsInfrastructureRole" },
      "Action": [
      "kms:GenerateDataKey*",
      "kms:ReEncryptTo",
      "kms:ReEncryptFrom"
      ],
      "Resource":"*",
      "Condition": {
        "StringEquals": {
          "kms:CallerAccount": "aws_account_id",
          "kms:ViaService": "ec2.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
          "kms:EncryptionContextKeys": "aws:ebs:id"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/ecsInfrastructureRole" },
      "Action": "kms:CreateGrant",
      "Resource":"*",
      "Condition": {
        "StringEquals": {
          "kms:CallerAccount": "aws_account_id",
          "kms:ViaService": "ec2.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
          "kms:EncryptionContextKeys": "aws:ebs:id"
        },
        "Bool": {
          "kms:GrantIsForAWSResource": true
        }
      }
    }
```

키 정책 및 권한에 대한 자세한 내용은 AWS KMS 개발자 안내서의 Key policies in AWS KMS 및 AWS KMS permissions를 참조하세요. 키 권한과 관련된 EBS 볼륨 연결 문제 해결은 Amazon ECS 작업에 Amazon EBS 볼륨 연결 관련 문제 해결 섹션을 참조하세요.

Amazon ECS 배포 시 Amazon EBS 볼륨 구성 지정

`configuredAtLaunch` 파라미터가 `true`로 설정된 작업 정의를 등록한 후 독립 실행형 작업을 실행하거나 서비스를 생성 또는 업데이트할 때 배포 시 Amazon EBS 볼륨을 구성할 수 있습니다.

볼륨을 구성하려면 Amazon ECS API를 사용하거나 다음 AWS CLI 명령의 입력으로 JSON 파일을 전달할 수 있습니다.

- `run-task` - 독립 실행형 ECS 작업을 실행합니다.
- `start-task` -특정 컨테이너 인스턴스에서 독립 실행형 ECS 작업을 실행합니다. 이 명령은 Fargate 시작 유형 작업에는 적용할 수 없습니다.
- `create-service` - 새 ECS 서비스를 생성합니다.
- `update-service` - 기존 서비스를 업데이트합니다.

> **Note**
>
> 작업의 컨테이너에서 탑재된 Amazon EBS 볼륨에 쓰려면 컨테이너를 루트 사용자로 실행해야 합니다.

또한 AWS Management Console을 사용하여 Amazon EBS 볼륨을 구성할 수도 있습니다. 자세한 내용은 애플리케이션을 Amazon ECS 태스크로 실행, Amazon ECS 롤링 업데이트 배포 생성, Amazon ECS 서비스 업데이트 섹션을 참조하세요.

다음 JSON 코드 조각은 배포 시 구성할 수 있는 Amazon EBS 볼륨의 모든 파라미터를 보여줍니다. 볼륨 구성에 이러한 파라미터를 사용하려면 *user input placeholders*를 사용자 정보로 바꿉니다. 이러한 파라미터에 대한 자세한 내용은 볼륨 구성을 참조하세요.

```
"volumeConfigurations": [
        {
            "name": "ebs-volume",
            "managedEBSVolume": {
                "encrypted": true,
                "kmsKeyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
```

```
                "volumeType": "gp3",
                "sizeInGiB": 10,
                "snapshotId": "snap-12345",
                "volumeInitializationRate":100,
                "iops": 3000,
                "throughput": 125,
                "tagSpecifications": [
                    {
                        "resourceType": "volume",
                        "tags": [
                            {
                                "key": "key1",
                                "value": "value1"
                            }
                        ],
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn": "arn:aws:iam::1111222333:role/ecsInfrastructureRole",
                 "terminationPolicy": {
                    "deleteOnTermination": true//can't be configured for service-
managed tasks, always true
                },
                "filesystemType": "ext4"
            }
        }
    ]
```

Important

구성에서 지정하는 volumeName이 작업 정의에서 지정하는 volumeName과 같은지 확인합니다.

볼륨 연결 상태를 확인하는 방법에 대한 자세한 내용은 Amazon ECS 작업에 Amazon EBS 볼륨 연결 관련 문제 해결 섹션을 참조하세요. EBS 볼륨 연결에 필요한 Amazon ECS 인프라 AWS Identity and Access Management(IAM) 역할에 대한 자세한 내용은 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.

다음은 Amazon EBS 볼륨의 구성을 보여주는 JSON 코드 조각 예제입니다. 코드 조각을 JSON 파일에 저장하고 `--cli-input-json file://filename` 파라미터를 사용하여 파일을 AWS CLI 명령의

파라미터로 전달하여 이 예제를 사용할 수 있습니다. *user input placeholders*를 사용자의 정보로 대체합니다.

독립 실행형 작업에 대한 볼륨 구성

다음 코드 조각은 Amazon EBS 볼륨을 독립 실행형 작업에 연결하도록 구성하는 구문을 보여줍니다. 다음 JSON 코드 조각은 volumeType, sizeInGiB, encrypted, kmsKeyId 설정을 구성하는 구문을 보여줍니다. JSON 파일에 지정된 구성은 EBS 볼륨을 생성하여 독립 실행형 작업에 연결하는 데 사용됩니다.

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "volumeConfigurations": [
        {
            "name": "datadir",
            "managedEBSVolume": {
                "volumeType": "gp3",
                "sizeInGiB": 100,
                "roleArn":"arn:aws:iam::1111222333:role/ecsInfrastructureRole",
                "encrypted": true,
                "kmsKeyId":
 "arn:aws:kms:region:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            }
        }
   ]
}
```

서비스 생성 시 볼륨 구성

다음 코드 조각은 서비스에서 관리하는 작업에 연결하도록 Amazon EBS 볼륨을 구성하는 구문을 보여줍니다. 볼륨은 snapshotId 파라미터를 사용하여 지정된 스냅샷에서 200MiB/s의 속도로 소싱됩니다. JSON 파일에 지정된 구성은 EBS 볼륨을 생성하여 서비스에서 관리하는 각 작업에 연결하는 데 사용됩니다.

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "serviceName": "mysvc",
   "desiredCount": 2,
   "volumeConfigurations": [
        {
```

```
            "name": "myEbsVolume",
            "managedEBSVolume": {
              "roleArn":"arn:aws:iam::1111222333:role/ecsInfrastructureRole",
              "snapshotId": "snap-12345",
              "volumeInitializationRate": 200
            }
        }
    ]
}
```

서비스 업데이트 시 볼륨 구성

다음 JSON 코드 조각은 이전에 작업에 연결하도록 Amazon EBS 볼륨을 구성하지 않았던 서비스를 업데이트하는 구문을 보여줍니다. `configuredAtLaunch`가 `true`로 설정된 작업 정의 개정의 ARN을 제공해야 합니다. 다음 JSON 코드 조각은 `volumeType`, `sizeInGiB`, `throughput`, `iops`, `filesystemType` 설정을 구성하는 구문을 보여줍니다. 이 구성은 EBS 볼륨을 생성하여 서비스에서 관리하는 각 작업에 연결하는 데 사용됩니다.

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "service": "mysvc",
   "desiredCount": 2,
   "volumeConfigurations": [
        {
            "name": "myEbsVolume",
            "managedEBSVolume": {
              "roleArn":"arn:aws:iam::1111222333:role/ecsInfrastructureRole",
               "volumeType": "gp3",
                "sizeInGiB": 100,
                 "iops": 3000,
                "throughput": 125,
                "filesystemType": "ext4"
            }
        }
   ]
}
```

Amazon EBS 볼륨을 더 이상 사용하지 않도록 서비스 구성

다음 JSON 코드 조각은 Amazon EBS 볼륨을 더 이상 사용하지 않도록 서비스를 업데이트하는 구문을 보여줍니다. `configuredAtLaunch`가 `false`로 설정된 작업 정의 또는 `configuredAtLaunch` 파

라미터가 없는 작업 정의의 ARN을 제공해야 합니다. 또한 빈 volumeConfigurations 객체를 제공해야 합니다.

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "service": "mysvc",
   "desiredCount": 2,
   "volumeConfigurations": []
}
```

Amazon EBS 볼륨에 대한 종료 정책

Amazon ECS 작업이 종료되면 Amazon ECS는 deleteOnTermination 값을 사용하여 종료된 작업에 연결된 Amazon EBS 볼륨을 삭제할지 여부를 결정합니다. 기본적으로 작업에 연결된 EBS 볼륨은 작업이 종료될 때 삭제됩니다. 독립 실행형 작업의 경우 이 설정을 변경하여 대신 작업 종료 시 볼륨을 보존할 수 있습니다.

> **Note**
>
> 서비스에서 관리하는 작업에 연결된 볼륨은 보존되지 않으며 작업 종료 시 항상 삭제됩니다.

Amazon EBS 볼륨 태그 지정

tagSpecifications 객체를 사용하여 Amazon EBS 볼륨에 태그를 지정할 수 있습니다. 객체를 사용하여 볼륨이 독립 실행형 작업에 연결되었는지 또는 서비스 내 작업에 연결되었는지에 따라 자체 태그를 제공하고 작업 정의 또는 서비스에서 태그 전파를 설정할 수 있습니다. 볼륨에 연결할 수 있는 최대 태그 수는 50개입니다.

> **Important**
>
> Amazon ECS는 Amazon EBS 볼륨에 예약된 태그 AmazonECSCreated 및 AmazonECSManaged를 자동으로 연결합니다. 즉, 볼륨에 최대 48개의 추가 태그 연결을 제어할 수 있습니다. 이러한 추가 태그는 사용자 정의 태그, ECS 관리형 태그 또는 전파된 태그일 수 있습니다.

Amazon ECS 관리형 태그를 볼륨에 추가하려면 UpdateService, CreateService, RunTask 또는 StartTask 직접 호출에서 enableECSManagedTags를 true로 설정해야 합니다. Amazon

ECS 관리형 태그를 켜면 Amazon ECS에서 클러스터 및 서비스 정보(aws:ecs:*clusterName* 및 aws:ecs:*serviceName*)로 볼륨에 태그를 자동 지정합니다. Amazon ECS 리소스 태그 지정에 대한 자세한 내용은 Amazon ECS 리소스 태그 지정을 참조하세요.

다음 JSON 코드 조각은 사용자 정의 태그를 사용하여 서비스의 각 작업에 연결된 각 Amazon EBS 볼륨에 사용자 정의 태그를 지정하는 구문을 보여줍니다. 서비스를 생성하는 데 이 예제를 사용하려면 *user input placeholders*를 사용자 정보로 바꿉니다.

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "serviceName": "mysvc",
   "desiredCount": 2,
   "enableECSManagedTags": true,
   "volumeConfigurations": [
        {
            "name": "datadir",
            "managedEBSVolume": {
                "volumeType": "gp3",
                "sizeInGiB": 100,
                 "tagSpecifications": [
                    {
                        "resourceType": "volume",
                        "tags": [
                            {
                                "key": "key1",
                                "value": "value1"
                            }
                        ],
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn":"arn:aws:iam:1111222333:role/ecsInfrastructureRole",
                "encrypted": true,
                "kmsKeyId":
 "arn:aws:kms:region:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            }
        }
   ]
}
```

> **Important**
>
> Amazon EBS 볼륨에 태그를 지정하려면 `volume` 리소스 유형을 지정해야 합니다.

Fargate 온디맨드 태스크의 Amazon EBS 볼륨 성능

Fargate 온디맨드 작업에 사용할 수 있는 기준 Amazon EBS 볼륨 IOPS 및 처리량은 작업에 요청한 총 CPU 단위에 따라 달라집니다. Fargate 작업에 0.25, 0.5 또는 1의 가상 CPU 단위(vCPU)를 요청하는 경우 범용 SSD 볼륨(gp2 또는 gp3)이나 하드 디스크 드라이브(HDD) 볼륨(`st1` 또는`sc1`)을 구성하는 것이 좋습니다. Fargate 작업에 2 이상의 vCPU를 요청하는 경우 작업에 연결된 Amazon EBS 볼륨에 다음과 같은 기준 성능 제한이 적용됩니다. 일시적으로 다음 제한보다 높은 EBS 성능을 얻을 수 있습니다. 그러나 이러한 제한에 따라 워크로드를 계획하는 것이 좋습니다.

요청된 CPU 단위(vCPU 단위)	기준 Amazon EBS IOPS(16KiB I/O)	기준 Amazon EBS 처리량(MBPS 단위, 128KiB I/O)	기준 대역폭(Mbps 단위)
2	3,000	75	360
4	5,000	120	1,150
8	10,000개	250	2,300
16	15,000	500	4,500

> **Note**
>
> Amazon EBS 볼륨을 Fargate 작업에 연결하도록 구성하면 Fargate 작업에 대한 Amazon EBS 성능 제한이 작업의 임시 스토리지와 연결된 볼륨 사이에서 공유됩니다.

EC2 태스크의 Amazon EBS 볼륨 성능

Amazon EBS는 성능 특성 및 가격이 다른 볼륨 유형을 제공하므로 애플리케이션의 필요에 맞게 스토리지 성능과 비용을 조정할 수 있습니다. 볼륨당 IOPS 및 볼륨당 처리량을 포함한 성능에 대한 자세한 내용은 Amazon Elastic Block Store 사용 설명서의 Amazon EBS 볼륨 유형을 참조하세요.

Amazon ECS 작업에 Amazon EBS 볼륨 연결 관련 문제 해결

Amazon EBS 볼륨을 Amazon ECS 작업에 연결하는 작업과 관련된 문제를 확인하거나 해결해야 할 수 있습니다.

볼륨 연결 상태 확인

AWS Management Console을 사용하여 Amazon ECS 작업에 대한 Amazon EBS 볼륨 연결 상태를 볼 수 있습니다. 작업이 시작되고 연결이 실패하는 경우 문제 해결에 사용할 수 있는 상태 이유도 표시됩니다. 생성된 볼륨이 삭제되고 작업이 중지됩니다. 상태 이유에 대한 자세한 내용은 Amazon ECS 태스크에 대한 Amazon EBS 볼륨 연결의 상태 이유 섹션을 참조하세요.

콘솔을 사용하여 볼륨의 연결 상태 및 상태 이유를 보는 방법

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터 페이지에서 작업이 실행 중인 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 나타납니다.
3. 클러스터 세부 정보 페이지에서 작업을 선택합니다.
4. 볼륨 연결 상태를 보려는 작업을 선택합니다. 검사하려는 작업이 중지된 경우 원하는 상태 필터링을 사용하고 중지됨을 선택해야 할 수도 있습니다.
5. 작업의 세부 정보 페이지에서 볼륨 탭을 선택합니다. 연결 상태에서 Amazon EBS 볼륨의 연결 상태를 볼 수 있습니다. 볼륨이 태스크에 연결되지 않으면 연결 상태 아래에서 상태를 선택하여 장애 원인을 표시할 수 있습니다.

또한 DescribeTasks API를 사용하여 작업의 볼륨 연결 상태 및 관련 상태 이유를 볼 수도 있습니다.

서비스 및 작업 실패

Amazon EBS 볼륨에만 국한되지 않는 서비스 또는 작업 실패가 발생할 수 있습니다. 이 경우 볼륨 연결에 영향을 미칠 수 있습니다. 자세한 내용은 단원을 참조하세요.

- 서비스 이벤트 메시지
- 중지된 작업 오류 코드
- API 실패 이유

Amazon ECS 태스크에 대한 Amazon EBS 볼륨 연결의 상태 이유

Amazon ECS 작업에 연결하기 위해 Amazon EBS 볼륨을 구성할 때 AWS Management Console에서 상태 이유의 형태로 발생할 수 있는 문제를 해결하기 위해 다음 참조를 사용합니다. 콘솔에서 이러한 상태 이유를 찾는 방법에 대한 자세한 내용은 볼륨 연결 상태 확인 섹션을 참조하세요.

ECS는 구성된 ECS 인프라 역할 'arn:aws:iam::*111122223333*:role/*ecsInfrastructureRole*'을 수임할 수 없습니다. 전달되는 역할이 Amazon ECS와 적절한 신뢰 관계를 보유하는지 확인합니다.

이 상태 이유는 다음 시나리오에 나타납니다.

- 필요한 신뢰 정책을 연결하지 않고 IAM 역할을 제공합니다. Amazon ECS는 역할에 필요한 신뢰 정책이 없는 경우 제공한 Amazon ECS 인프라 IAM 역할에 액세스할 수 없습니다. 작업이 `DEPROVISIONING` 상태에서 멈출 수 있습니다. 필요한 신뢰 정책에 대한 자세한 내용은 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.
- IAM 사용자에게 Amazon ECS 인프라 역할을 Amazon ECS에 전달할 수 있는 권한이 없습니다. 작업이 `DEPROVISIONING` 상태에서 멈출 수 있습니다. 이 문제를 방지하려면 사용자에게 `PassRole` 권한을 연결할 수 있습니다. 자세한 내용은 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.
- IAM 역할에 Amazon EBS 볼륨 연결에 필요한 권한이 없습니다. 작업이 `DEPROVISIONING` 상태에서 멈출 수 있습니다. Amazon EBS 볼륨을 작업에 연결하는 데 필요한 특정 권한에 대한 자세한 내용은 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.

> **Note**
>
> 역할 전파 지연으로 인해 이 오류 메시지가 표시될 수도 있습니다. 몇 분을 기다린 후 역할 사용을 다시 시도해도 문제가 해결되지 않으면 역할에 대한 신뢰 정책을 잘못 구성했을 수 있습니다.

ECS가 EBS 볼륨을 설정하지 못했습니다. Encountered IdempotentParameterMismatch', '제공한 클라이언트 토큰이 이미 삭제된 리소스에 연결되어 있습니다. 다른 클라이언트 토큰을 사용하세요.'

다음 AWS KMS 주요 시나리오에서 `IdempotentParameterMismatch` 메시지가 표시될 수 있습니다.

- 유효하지 않은 KMS 키 ARN, ID 또는 별칭을 지정합니다. 이 시나리오에서 작업이 성공적으로 시작된 것처럼 보이지만 AWS에서 KMS 키를 비동기적으로 인증하기 때문에 실제로 작업은 실패합니다. 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EBS 암호화를 참조하세요.

- Amazon ECS 인프라 IAM 역할에서 암호화에 키를 사용하도록 허용하는 권한이 없는 고객 관리형 키를 제공합니다. 키 정책 권한 문제를 방지하려면 Amazon EBS 볼륨의 데이터 암호화에서 예제 AWS KMS 키 정책을 참조하세요.

Amazon EBS 볼륨 이벤트 및 Amazon ECS 작업 상태 변경 이벤트를 Amazon CloudWatch 그룹과 같은 대상으로 전송하도록 Amazon EventBridge를 설정할 수 있습니다. 그런 다음, 이러한 이벤트를 사용하여 볼륨 연결에 영향을 미치는 특정 고객 관리형 키 관련 문제를 식별할 수 있습니다. 자세한 내용은 단원을 참조하세요.

- AWS re:Post의 EventBridge 규칙의 대상으로 사용할 CloudWatch 로그 그룹을 생성하려면 어떻게 해야 합니까?
- 작업 상태 변경 이벤트.
- Amazon EBS 사용 설명서의 Amazon EBS를 위한 Amazon EventBridge 이벤트를 참조하세요.

작업에 EBS 볼륨 연결을 구성하는 동안 ECS 제한 시간이 초과되었습니다.

다음 파일 시스템 형식 시나리오에서 이 메시지가 표시됩니다.

- 구성 중에 지정하는 파일 시스템 형식이 작업의 운영 체제와 호환되지 않습니다.
- 스냅샷에서 Amazon EBS 볼륨을 생성하도록 구성했는데 스냅샷의 파일 시스템 형식이 작업의 운영 체제와 호환되지 않습니다. 스냅샷에서 생성된 볼륨의 경우 스냅샷을 생성할 때 볼륨에서 사용하던 것과 동일한 파일 시스템 유형을 지정해야 합니다.

Amazon ECS 컨테이너 에이전트 로그를 활용하여 Amazon EC2 시작 유형 작업에 대한 이 메시지 문제를 해결할 수 있습니다. 자세한 내용은 Amazon ECS 로그 파일 위치 및 Amazon ECS 로그 수집기를 참조하세요.

Amazon ECS에서 Amazon EFS 볼륨 사용

Amazon Elastic File System(Amazon EFS)은 Amazon ECS 태스크에 간단하고 확장 가능한 파일 스토리지를 제공합니다. Amazon EFS를 사용하면 스토리지 용량이 탄력적입니다. 파일을 추가 및 제거하면 스토리지 용량이 자동으로 확장 및 축소됩니다. 애플리케이션에서 스토리지가 필요할 때 필요한 만큼 확보할 수 있습니다.

Amazon ECS에서 Amazon EFS 파일 시스템을 사용하여 컨테이너 인스턴스 집합 간에 파일 시스템 데이터를 내보낼 수 있습니다. 이렇게 하면 태스크는 해당 태스크가 차지한 인스턴스와 상관없이 동일한 영구 스토리지에 액세스할 수 있습니다. 태스크 정의는 컨테이너 인스턴스에서 볼륨 마운트를 참조하여 파일 시스템을 사용해야 합니다.

자습서는 콘솔을 사용하여 Amazon ECS에 대한 Amazon EFS 파일 시스템 구성을 참조하세요.

고려 사항

Amazon EFS 볼륨을 사용할 때는 다음 사항을 고려해야 합니다.

- EC2 시작 유형을 사용하는 태스크의 경우, Amazon EFS 파일 시스템 지원이 컨테이너 에이전트 버전이 1.35.0인 Amazon ECS 최적화 AMI 버전 `20191212`의 공개 미리보기로 추가되었습니다. 그러나 Amazon EFS 파일 시스템 지원은 Amazon EFS 액세스 포인트 및 IAM 권한 부여 기능이 포함된 컨테이너 에이전트 버전이 1.38.0인 Amazon ECS 최적화 AMI 버전 `20200319`의 정식 출시를 시작했습니다. 이러한 기능을 사용하려면 Amazon ECS 최적화 AMI 버전 `20200319` 이상을 사용하는 것이 좋습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.

  > **Note**
  >
  > 자체 AMI를 생성하는 경우 컨테이너 에이전트 1.38.0 이상, `ecs-init` 버전 1.38.0-1 이상을 사용하고 Amazon EC2 인스턴스에서 다음 명령을 실행하여 Amazon ECS 볼륨 플러그인을 활성화해야 합니다. 이 명령은 기본 이미지로 Amazon Linux 2 또는 Amazon Linux를 사용하는지에 따라 달라집니다.
  >
  > Amazon Linux 2
  >
  > ```
  > yum install amazon-efs-utils
  > systemctl enable --now amazon-ecs-volume-plugin
  > ```
  >
  > Amazon Linux
  >
  > ```
  > yum install amazon-efs-utils
  > sudo shutdown -r now
  > ```

- Fargate에서 호스팅되는 태스크의 경우 Amazon EFS 파일 시스템이 플랫폼 버전 1.4.0 이상(Linux)에서 지원됩니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.
- Fargate에서 호스팅되는 태스크에 Amazon EFS 볼륨을 사용할 경우, Fargate는 Amazon EFS 볼륨을 관리하는 감독자 컨테이너를 생성합니다. 감독자 컨테이너는 태스크 메모리와 CPU를 소량 사용합니다. 감독자 컨테이너는 작업 메타데이터 버전 4 엔드포인트를 쿼리할 때 표시됩니다. 또한 CloudWatch Container Insights에서 컨테이너 이름 `aws-fargate-supervisor`로 표시됩니다. Amazon EC2 시작 유형을 사용하는 경우 자세한 내용은 Amazon ECS 작업 메타데이터 엔드포인트 버전 4 섹션을 참조하세요. Fargate 시작 유형을 사용하는 방법에 대한 자세한 내용은 Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 엔드포인트 버전 4 섹션을 참조하세요.

- Amazon EFS 볼륨 사용이나 `EFSVolumeConfiguration` 지정은 외부 인스턴스에서 지원되지 않습니다.
- 에이전트 구성 파일의 `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` 파라미터를 기본값보다 작은 값(약 1시간)으로 설정하는 것이 좋습니다. 이 변경을 통해 EFS 마운트 보안 인증 만료를 방지하고 사용하지 않는 마운트를 정리할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 단원을 참조하세요.

Amazon EFS 액세스 포인트 사용

Amazon EFS 액세스 포인트는 EFS 파일 시스템에 대한 애플리케이션별 진입점으로, 공유 데이터 세트에 대한 애플리케이션 액세스를 관리할 수 있도록 합니다. Amazon EFS 액세스 포인트와 액세스 제어 방법에 대한 자세한 정보는 Amazon Elastic File System 사용 설명서의 Amazon EFS 액세스 포인트 사용하기를 참조하세요.

액세스 포인트는 액세스 포인트를 통해 이루어지는 모든 파일 시스템 요청에 대해 사용자의 POSIX 그룹을 포함한 사용자 자격 증명을 적용할 수 있습니다. 또한 파일 시스템에 대해 다른 루트 디렉터리를 적용할 수 있습니다. 이는 클라이언트가 지정된 디렉터리 또는 하위 디렉터리의 데이터에만 액세스할 수 있도록 합니다.

> **Note**
>
> EFS 액세스 포인트를 생성할 때 파일 시스템에서 루트 디렉터리 역할을 하는 경로를 지정합니다. Amazon ECS 태스크 정의의 액세스 포인트 ID로 EFS 파일 시스템을 참조할 때 루트 디렉터리는 생략하거나 `/`로 설정할 수 있습니다. 그러면 EFS 액세스 포인트에 설정된 경로가 적용됩니다.

Amazon ECS 태스크 IAM 역할을 사용하여 특정 애플리케이션에서 특정 액세스 포인트를 사용하도록 적용할 수 있습니다. IAM 정책을 액세스 포인트와 결합하면 애플리케이션의 특정 데이터 세트에 안전하게 액세스할 수 있습니다. IAM 역할을 사용하는 방법에 대한 자세한 정보는 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.

Amazon ECS에서 Amazon EFS 볼륨 사용 모범 사례

Amazon ECS에서 Amazon EFS를 사용할 때 다음 모범 사례 권장 사항을 참고하세요.

Amazon EFS 볼륨의 보안 및 액세스 제어

Amazon EFS는 Amazon EFS 파일 시스템에 저장된 데이터를 보안하고 해당 데이터가 필요한 애플리케이션에서만 데이터에 액세스할 수 있도록 보장하는 데 사용할 수 있는 액세스 제어 기능을 제공합니다. 저장 중 데이터 및 전송 중 데이터 암호화를 활성화하여 데이터를 보안할 수 있습니다. 자세한 내용은 Amazon Elastic File System 사용 설명서에서 Amazon EFS의 데이터 암호화를 참조하세요.

데이터 암호화 외에도 Amazon EFS를 사용하여 파일 시스템에 대한 액세스를 제한할 수 있습니다. EFS에서 액세스 제어를 구현하는 세 가지 방법이 있습니다.

- 보안 그룹 - Amazon EFS 탑재 대상을 사용하면 네트워크 트래픽을 허용 및 거부하는 데 사용되는 보안 그룹을 구성할 수 있습니다. Amazon EFS에 연결된 보안 그룹이 Amazon ECS 인스턴스에 연결된 보안 그룹의 NFS 트래픽(포트 2049)을 허용하도록 구성하거나 `awsvpc` 네트워크 모드를 사용하는 경우 Amazon ECS 작업을 허용하도록 구성할 수 있습니다.
- IAM - IAM을 사용하여 Amazon EFS 파일 시스템에 대한 액세스를 제한할 수 있습니다. 이 기능이 구성된 경우 Amazon ECS 작업에는 EFS 파일 시스템을 탑재하도록 파일 시스템 액세스를 위한 IAM 역할이 필요합니다. 자세한 내용은 Amazon Elastic 파일 시스템 사용 설명서의 IAM을 사용하여 파일 시스템 데이터 액세스 제어를 참조하세요.

  IAM 정책은 Amazon EFS 파일 시스템에 연결할 때 클라이언트가 TLS를 사용하도록 요구하는 등의 사전 정의된 조건을 적용할 수도 있습니다. 자세한 내용은 Amazon Elastic File System 사용 설명서의 클라이언트에 대한 Amazon EFS 조건 키를 참조하세요.
- Amazon EFS 액세스 포인트 - Amazon EFS 액세스 포인트는 Amazon EFS 파일 시스템에 대한 애플리케이션별 진입점입니다. 액세스 포인트를 사용하여 액세스 포인트를 통해 이루어지는 모든 파일 시스템 요청에 대해 사용자의 POSIX 그룹을 포함한 사용자 자격 증명을 적용할 수 있습니다. 또한 파일 시스템에 대해 다른 루트 디렉터리를 적용할 수 있습니다. 이는 클라이언트가 지정된 디렉터리 또는 하위 디렉터리의 데이터에만 액세스할 수 있도록 합니다.

IAM 정책

IAM 정책을 사용하여 Amazon EFS 파일 시스템에 대한 액세스를 제어할 수 있습니다.

파일 시스템 정책을 사용하여 클라이언트의 파일 시스템 액세스에 대해 다음 작업을 지정할 수 있습니다.

작업	설명
elasticfilesystem:ClientMount	파일 시스템에 대한 읽기 전용 액세스를 제공합니다.
elasticfilesystem:ClientWrite	파일 시스템에 대한 쓰기 권한을 제공합니다.
elasticfilesystem:ClientRootAccess	파일 시스템에 액세스할 때 루트 사용자를 사용할 수 있는 기능을 제공합니다.

정책의 각 작업을 지정해야 합니다. 정책은 다음과 같은 방법으로 정의할 수 있습니다.

- 클라이언트 기반 - 정책을 작업 역할에 연결합니다.

  태스크 정의를 생성할 때 IAM 권한 부여 옵션을 설정합니다.
- 리소스 기반 - Amazon EFS 파일 시스템에 정책을 연결합니다.

  리소스 기반 정책이 없는 경우 파일 시스템 생성 시 기본적으로 모든 보안 주체(*)에게 액세스 권한이 부여됩니다.

IAM 권한 부여 옵션을 설정할 때 작업 역할과 관련된 정책과 Amazon EFS 리소스 기반 정책을 병합합니다. IAM 권한 부여 옵션은 정책과 함께 작업 ID(작업 역할)를 Amazon EFS로 전달합니다. 이를 통해 Amazon EFS 리소스 기반 정책이 정책에 지정된 IAM 사용자 또는 역할에 대한 컨텍스트를 사용할 수 있습니다. 이 옵션을 설정하지 않으면 Amazon EFS 리소스 수준 정책이 IAM 사용자를 "익명"으로 식별합니다.

보안을 극대화하려면 Amazon EFS 파일 시스템에 세 가지 액세스 제어를 모두 구현하는 방법을 고려합니다. 예를 들어, Amazon EFS 탑재 지점에 연결된 보안 그룹이 컨테이너너 인스턴스 또는 Amazon ECS 작업과 연결된 보안 그룹의 수신 NFS 트래픽만 허용하도록 구성할 수 있습니다. 또한 허용된 보안 그룹에서 시작된 연결이더라도 파일 시스템에 액세스하는 데 IAM 역할을 요구하도록 Amazon EFS를 구성할 수 있습니다. 마지막으로 Amazon EFS 액세스 포인트를 사용하여 POSIX 사용자 권한을 적용하고 애플리케이션의 루트 디렉터리를 지정할 수 있습니다.

다음 작업 정의 코드 조각은 액세스 포인트를 사용하여 Amazon EFS 파일 시스템을 탑재하는 방법을 보여줍니다.

```
"volumes": [
    {
```

```
      "efsVolumeConfiguration": {
        "fileSystemId": "fs-1234",
        "authorizationConfig": {
          "accessPointId": "fsap-1234",
          "iam": "ENABLED"
        },
        "transitEncryption": "ENABLED",
        "rootDirectory": ""
      },
      "name": "my-filesystem"
    }
]
```

Amazon EFS 볼륨 성능

Amazon EFS는 범용 및 최대 I/O라는 두 가지 성능 모드를 제공합니다. 범용은 콘텐츠 관리 시스템 및 CI/CD 도구와 같이 지연 시간에 민감한 애플리케이션에 적합합니다. 반면 최대 I/O 파일 시스템은 데이터 분석, 미디어 처리, 기계 학습과 같은 워크로드에 적합합니다. 이러한 워크로드는 수백 또는 수천 개의 컨테이너에서 병렬 작업을 수행해야 하며 가능한 가장 높은 총 처리량과 IOPS를 요구합니다. 자세한 내용은 Amazon Elastic File System 사용 설명서의 Amazon EFS performance modes를 참조하세요.

지연 시간에 민감한 일부 워크로드는 최대 I/O 성능 모드에서 제공하는 더 높은 I/O 수준과 범용 성능 모드에서 제공하는 더 낮은 지연 시간이 필요합니다. 이러한 유형의 워크로드에는 여러 범용 성능 모드 파일 시스템을 생성하는 것이 좋습니다. 이 경우 워크로드 및 애플리케이션이 지원할 수 있는 한, 모든 파일 시스템에 애플리케이션 워크로드를 분산시키는 것이 좋습니다.

Amazon EFS 볼륨 처리량

모든 Amazon EFS 파일 시스템에는 측정된 처리량이 연결되어 있습니다. 이 처리량은 프로비저닝된 처리량을 사용하는 파일 시스템의 프로비저닝된 처리량의 크기 또는 버스팅 처리량을 사용하는 파일 시스템의 EFS Standard 또는 One Zone 스토리지 클래스에 저장된 데이터의 크기에 따라 결정됩니다. 자세한 내용은 Amazon Elastic File System 사용 설명서의 Understanding metered throughput을 참조하세요.

Amazon EFS 파일 시스템의 기본 처리량 모드는 버스팅 모드입니다. 버스팅 모드를 사용하면 파일 시스템이 확장됨에 따라 파일 시스템에서 사용할 수 있는 처리량이 스케일 인 또는 스케일 아웃합니다. 파일 기반 워크로드는 일반적으로 급증하기 때문에 일정 기간 높은 수준의 처리량이 필요하고 나머지 시간에는 낮은 수준의 처리량을 지원하면 되므로 Amazon EFS에는 일정 기간 높은 처리량 수준을 허용하도록 버스트 기능이 설계되었습니다. 또한 많은 워크로드에서 읽기 작업이 많기 때문에 읽기 작업은 다른 NFS 작업(예: 쓰기)에 비해 1:3 비율로 측정됩니다.

모든 Amazon EFS 파일 시스템은 Amazon EFS Standard 또는 Amazon EFS One Zone 스토리지의 각 TB에 대해 50MB/s의 일관된 기준 성능을 제공합니다. 크기에 관계없이 모든 파일 시스템은 100MB/s까지 버스트할 수 있습니다. EFS Standard 또는 EFS One Zone 스토리지에서 1TB를 초과하는 파일 시스템은 TB당 100MB/s까지 버스트할 수 있습니다. 읽기 작업은 1:3 비율로 측정되므로 읽기 처리량의 각 TiB에 대해 최대 300MiBS/s까지 지원할 수 있습니다. 파일 시스템에 데이터를 추가하면 파일 시스템에서 사용할 수 있는 최대 처리량 규모가 Amazon EFS Standard 스토리지 클래스의 스토리지에서 선형적으로 자동 조정됩니다. 저장된 데이터 크기보다 많은 처리량이 필요한 경우 워크로드에 필요한 특정 크기만큼 프로비저닝된 처리량을 구성할 수 있습니다.

파일 시스템 처리량은 파일 시스템에 연결된 모든 Amazon EC2 인스턴스에서 공유됩니다. 예를 들어, 100MB/s의 처리량까지 버스트할 수 있는 1TB 파일 시스템은 단일 Amazon EC2 인스턴스에서 100MB/s를 지원할 수 있습니다(각각 10MB/s 지원 가능). 자세한 내용은 Amazon Elastic 파일 시스템 사용 설명서의 Amazon EFS 성능을 참조하세요.

Amazon EFS 볼륨에 대한 비용 최적화

Amazon EFS는 스토리지 규모 조정을 단순화합니다. Amazon EFS 파일 시스템은 데이터를 추가하면 자동으로 확장됩니다. 특히 Amazon EFS 버스팅 처리량 모드에서는 Standard 스토리지 클래스의 파일 시스템 크기가 커짐에 따라 처리량 규모가 조정됩니다. EFS 파일 시스템의 프로비저닝된 처리량에 대해 추가 비용을 지불하지 않고도 처리량을 향상시키기 위해 Amazon EFS 파일 시스템을 여러 애플리케이션과 공유할 수 있습니다. Amazon EFS 액세스 포인트를 사용하면 공유 Amazon EFS 파일 시스템에서 스토리지 격리를 구현할 수 있습니다. 이렇게 하면 애플리케이션이 여전히 동일한 파일 시스템을 공유하더라도 사용자가 권한을 부여하지 않으면 데이터에 액세스할 수 없습니다.

데이터가 증가하면 Amazon EFS를 통해 자주 액세스하지 않는 파일을 하위 스토리지 클래스로 자동으로 이동할 수 있습니다. Amazon EFS Standard-Infrequent Access(IA) 스토리지 클래스는 매일 액세스하지 않는 파일의 스토리지 비용을 줄여줍니다. 또한 Amazon EFS가 제공하는 고가용성, 높은 내구성, 탄력성 및 POSIX 파일 시스템 액세스에 손상을 주지 않습니다. 자세한 내용은 Amazon Elastic File System 사용 설명서의 EFS 스토리지 클래스를 참조하세요.

Amazon EFS 수명 주기 정책을 사용하여 자주 액세스하지 않는 파일을 Amazon EFS IA 스토리지로 이동해 비용을 자동으로 절약하는 방법을 고려합니다. 자세한 내용을 알아보려면 Amazon Elastic File System User Guide(Amazon Elastic File System 사용 설명서)의 Amazon EFS lifecycle management(EFS 수명 주기 관리)를 참조하세요.

Amazon EFS 파일 시스템을 생성할 때 Amazon EFS가 여러 가용 영역(Standard)에 데이터를 복제할지 또는 단일 가용 영역 내에 데이터를 중복 저장할지 선택할 수 있습니다. Amazon EFS One Zone 스토리지 클래스는 Amazon EFS Standard 스토리지 클래스에 비해 스토리지 비용을 크게 절감할 수 있습니다. 다중 AZ 복원력이 필요하지 않은 워크로드에는 Amazon EFS One Zone 스토리지 클래스

를 사용하는 방법을 고려합니다. 자주 액세스하지 않는 파일을 Amazon EFS One Zone-Infrequent Access로 이동하여 Amazon EFS One Zone 스토리지 비용을 더욱 절감할 수 있습니다. 자세한 내용은 Amazon EFS Infrequent Access를 참조하세요.

Amazon EFS 볼륨 데이터 보호

Amazon EFS는 Standard 스토리지 클래스를 사용하는 파일 시스템에서 여러 가용 영역에 걸쳐 데이터를 저장합니다. Amazon EFS One Zone 스토리지 클래스를 선택하면 단일 가용 영역 내에 데이터를 중복 저장합니다. 또한 Amazon EFS는 지정된 1년 동안 99.99999999999%의 내구성을 제공할 수 있도록 설계되었습니다.

모든 환경과 마찬가지로 백업을 보유하고 실수로 인한 삭제를 방지하는 보호 장치를 구축하는 것이 모범 사례입니다. Amazon EFS 데이터의 경우 모범 사례로, AWS Backup을 사용하여 정기적으로 테스트되고 올바르게 작동하는 백업이 포함됩니다. Amazon EFS One Zone 스토리지 클래스를 사용하는 파일 시스템은 사용자가 이 기능을 비활성화하지 않는 한, 파일 시스템 생성 시 기본적으로 파일을 자동으로 백업하도록 구성됩니다. 자세한 내용은 Amazon Elastic File System 사용 설명서의 EFS 파일 시스템 백업을 참조하세요.

Amazon ECS 작업 정의에서 Amazon EFS 파일 시스템 지정

컨테이너에 Amazon EFS 파일 시스템 볼륨을 사용하려면 태스크 정의에 볼륨 및 마운트 지점 구성을 지정해야 합니다. 다음 태스크 정의 JSON 코드 조각은 컨테이너에 사용할 `volumes` 및 `mountPoints` 객체의 구문을 나타냅니다.

```
{
    "containerDefinitions": [
        {
            "name": "container-using-efs",
            "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": [
                "ls -la /mount/efs"
            ],
            "mountPoints": [
                {
                    "sourceVolume": "myEfsVolume",
                    "containerPath": "/mount/efs",
                    "readOnly": true
                }
```

```
            ]
        }
    ],
    "volumes": [
        {
            "name": "myEfsVolume",
            "efsVolumeConfiguration": {
                "fileSystemId": "fs-1234",
                "rootDirectory": "/path/to/my/data",
                "transitEncryption": "ENABLED",
                "transitEncryptionPort": integer,
                "authorizationConfig": {
                    "accessPointId": "fsap-1234",
                    "iam": "ENABLED"
                }
            }
        }
    ]
}
```

`efsVolumeConfiguration`

유형: 객체

필수 여부: 아니요

이 파라미터는 Amazon EFS 볼륨을 사용할 때 지정됩니다.

`fileSystemId`

유형: 문자열

필수 항목 여부: 예

사용할 Amazon EFS 파일 시스템 ID입니다.

`rootDirectory`

유형: 문자열

필수 여부: 아니요

호스트 내의 루트 디렉터리로 탑재할 Amazon EFS 파일 시스템 내 디렉터리입니다. 이 파라미터가 생략되면 Amazon EFS 볼륨의 루트가 사용됩니다. `/`를 지정하면 이 파라미터를 생략하는 것과 동일한 효과가 있습니다.

> **Important**
>
> `authorizationConfig`에서 EFS 액세스 포인트를 지정하는 경우 루트 디렉터리 파라미터를 생략하거나 `/`로 설정하여 EFS 액세스 포인트에 설정된 경로를 적용해야 합니다.

`transitEncryption`

유형: 문자열

유효한 값: `ENABLED` | `DISABLED`

필수 여부: 아니요

Amazon ECS 호스트와 Amazon EFS 서버 간에 전송 중인 Amazon EFS 데이터에 대해 암호화를 사용 설정할지 여부를 지정합니다. Amazon EFS IAM 권한 부여를 사용하는 경우 전송 중 데이터 암호화를 사용 설정해야 합니다. 이 파라미터가 누락되면 `DISABLED`의 기본값이 사용됩니다. 자세한 정보는 Amazon Elastic File System 사용 설명서의 전송 중 데이터 암호화(Encrypting Data in Transit)를 참조하세요.

`transitEncryptionPort`

유형: 정수

필수 여부: 아니요

Amazon ECS 호스트와 Amazon EFS 서버 간에 암호화된 데이터를 전송할 때 사용할 포트입니다. 전송 중 데이터 암호화 포트를 지정하지 않으면 Amazon EFS 탑재 헬퍼가 사용하는 포트 선택 전략이 사용됩니다. 자세한 정보는 Amazon Elastic File System 사용 설명서의 EFS 탑재 헬퍼를 참조하세요.

`authorizationConfig`

유형: 객체

필수 여부: 아니요

Amazon EFS 파일 시스템에 대한 권한 부여 구성 세부 정보입니다.

`accessPointId`

유형: 문자열

필수 여부: 아니요

사용할 액세스 포인트 ID입니다. 액세스 포인트를 지정하는 경우 `efsVolumeConfiguration`의 루트 디렉터리 값을 생략하거나 `/`로 설정하여 EFS 액세스 포인트에 설정된 경로를 적용해야 합니다. 액세스 포인트를 사용하는 경우 `EFSVolumeConfiguration`에서 전송 중 데이터 암호화를 활성화해야 합니다. 자세한 정보는 Amazon Elastic File System 사용자 설명서의 Amazon EFS 액세스 포인트 태스크를 참조세요.

`iam`

유형: 문자열

유효한 값: `ENABLED` | `DISABLED`

필수 여부: 아니요

Amazon EFS 파일 시스템을 탑재할 때 태스크 정의에 정의된 Amazon ECS 태스크 IAM 역할을 사용할지 여부를 지정합니다. 활성화된 경우 `EFSVolumeConfiguration`에서 전송 중 데이터 암호화를 활성화해야 합니다. 이 파라미터가 누락되면 `DISABLED`의 기본값이 사용됩니다. 자세한 정보는 태스크의 IAM 역할을 참조하세요.

콘솔을 사용하여 Amazon ECS에 대한 Amazon EFS 파일 시스템 구성

Amazon ECS에서 Amazon Elastic File System(Amazon EFS) 파일 시스템을 사용하는 방법을 알아보세요.

1단계: Amazon ECS 클러스터 생성

다음 단계에 따라 Amazon ECS 클러스터를 생성합니다.

새 클러스터 생성(Amazon ECS 콘솔)

시작하기 전에 적절한 IAM 권한을 할당합니다. 자세한 내용은 the section called “Amazon ECS 클러스터 예제” 섹션을 참조하세요.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터(Clusters) 페이지에서 클러스터 생성(Create cluster)을 선택합니다.
5. Cluster configuration(클러스터 구성) 아래의 Cluster name(클러스터 이름)에 클러스터 이름으로 `EFS-tutorial`을 입력합니다.

6. (선택 사항) 태스크와 서비스가 시작되는 VPC와 서브넷을 변경하려면 네트워킹(Networking)에서 다음 태스크 중 하나를 수행합니다.

   - 서브넷을 제거하려면서브넷(Subnets)에서 제거하려는 각 서브넷에 대해 X를 선택합니다.
   - 기본(default) VPC가 아닌 다른 VPC로 변경하려면 VPC에서 기존 VPC를 선택한 다음 서브넷(Subnets)에서 각 서브넷을 선택합니다.
7. Amazon EC2 인스턴스를 클러스터에 추가하려면 인프라를 확장한 다음 Amazon EC2 인스턴스를 선택합니다. 다음으로 용량 공급자 역할을 하는 Auto Scaling 그룹을 구성합니다.

   - Auto Scaling 그룹을 생성하려면 Auto Scaling 그룹(ASG)(Auto Scaling group (ASG))에서 새 그룹 생성(Create new group)을 선택한 후 그룹에 대한 다음 세부 정보를 제공합니다.
     - Operating system/Architecture(운영 체제/아키텍처)에서 Amazon Linux 2를 선택합니다.
     - EC2 인스턴스 유형(EC2 instance type)에 `t2.micro`를 선택합니다.

       SSH 키 페어(SSH key pair)에서 인스턴스에 연결할 때 자격 증명을 증명하는 페어를 선택합니다.
     - 용량에 1을 입력합니다.
8. 생성(Create)을 선택합니다.

2단계: Amazon EC2 인스턴스 및 Amazon EFS 파일 시스템의 보안 그룹 생성

이 단계에서는 포트 80에서 인바운드 네트워크 트래픽을 허용하는 Amazon EC2 인스턴스 및 컨테이터 인스턴스에서 인바운드 액세스를 허용하는 Amazon EFS 파일 시스템의 보안 그룹을 생성합니다.

다음 옵션을 사용하여 Amazon EC2 인스턴스에 대한 보안 그룹을 생성합니다.

- 보안 그룹 이름 - 보안 그룹의 고유한 이름입니다.
- VPC - 클러스터에 대해 앞서 확인한 VPC입니다.
- 인바운드 규칙
  - 유형 - HTTP
  - 소스 - 0.0.0.0/0.

다음 옵션을 사용하여 Amazon EFS 파일 시스템에 대한 보안 그룹을 생성합니다.

- 보안 그룹 이름 - 보안 그룹의 고유한 이름입니다. 예를 들어 `EFS-access-for-sg-`*`dc025fa2`*입니다.

- VPC - 클러스터에 대해 앞서 확인한 VPC입니다.
- 인바운드 규칙
  - 유형 - NFS
  - 소스 - 인스턴스용으로 생성한 보안 그룹의 ID를 사용하여 사용자 지정합니다.

보안 그룹을 생성하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스에 대한 보안 그룹 생성을 참조하세요.

3단계: Amazon EFS 파일 시스템 생성

이 단계에서는 Amazon EFS 파일 시스템을 생성합니다.

Amazon ECS 작업에 대한 Amazon EFS 파일 시스템 생성하기

1. Amazon Elastic File System 콘솔(https://console.aws.amazon.com/efs/)을 엽니다.
2. 파일 시스템 생성(Create file system)을 선택합니다.
3. 파일 시스템의 이름을 입력한 다음 컨테이너 인스턴스가 호스팅되는 VPC를 선택합니다. 기본적으로 지정된 VPC의 각 서브넷은 해당 VPC의 기본 보안 그룹을 사용하는 탑재 대상을 수신합니다. 그런 다음 사용자 지정을 선택합니다.

   > **Note**
   >
   > 이 자습서에서는 Amazon EFS 파일 시스템, Amazon ECS 클러스터, 컨테이너 인스턴스 및 작업이 동일한 VPC에 있다고 가정합니다. 다른 VPC에서 파일 시스템을 탑재하는 방법에 대한 자세한 내용은 Amazon EFS 사용 설명서의 Walkthrough: Mount a file system from a different VPC를 참조하세요.

4. 파일 시스템 설정 페이지에서 선택적 설정을 구성한 다음 성능 설정에서 파일 시스템의 버스트 처리량 모드를 선택합니다. 설정을 구성한 후 다음을 선택합니다.

   a. (선택 사항) 파일 시스템에 대한 태그를 추가합니다. 예를 들어, 이름(Name) 키 옆에 있는 값(Value) 열에 해당 이름을 입력하여 파일 시스템에 대한 고유한 이름을 지정할 수 있습니다.

   b. (선택 사항) 수명 주기 관리를 통해 자주 액세스하지 않는 스토리지에 드는 비용을 절감할 수 있습니다. 자세한 정보는 Amazon Elastic File System User Guide의 EFS 수명 주기 관리를 참조하세요.

   c. (선택 사항) 암호화를 활성화합니다. 유휴 상태인 Amazon EFS 파일 시스템의 암호화를 활성화하려면 확인란을 선택합니다.

5. 네트워크 액세스 페이지의 탑재 대상에서 모든 가용 영역의 기존 보안 그룹 구성을 2단계: Amazon EC2 인스턴스 및 Amazon EFS 파일 시스템의 보안 그룹 생성에서 파일 시스템용으로 생성한 보안 그룹으로 바꾸고 다음을 선택합니다.
6. 이 자습서에서는 파일 시스템 정책을 구성할 필요가 없으므로 다음을 선택하여 이 섹션을 건너뛰어도 됩니다.
7. 파일 시스템 옵션을 검토하고 생성을 선택하여 프로세스를 완료합니다.
8. 파일 시스템 화면에서 파일 시스템 ID를 기록합니다. 다음 단계에서는 Amazon ECS 태스크 정의에서 이 값을 참조합니다.

4단계: Amazon EFS 파일 시스템에 콘텐츠 추가

이 단계에서는 Amazon EFS 파일 시스템을 Amazon EC2 인스턴스에 탑재하고 콘텐츠를 추가합니다. 이는 데이터의 지속적인 특성을 설명하기 위해 이 자습서에서 테스트 목적으로 사용됩니다. 이 기능을 사용하면 일반적으로 애플리케이션이나 Amazon EFS 파일 시스템에 데이터를 쓰는 다른 방법이 있습니다.

Amazon EC2 인스턴스를 생성하고 Amazon EFS 파일 시스템을 탑재하려면

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 인스턴스 시작을 선택합니다.
3. 애플리케이션 및 OS 이미지(Amazon Machine Image)에서 Amazon Linux 2 AMI(HVM)를 선택합니다.
4. 인스턴스 유형에서 기본 인스턴스 유형 `t2.micro`를 유지합니다.
5. 키 쌍(로그인)에서 인스턴스에 대한 SSH 액세스를 위한 키 쌍을 선택합니다.
6. 네트워크 설정에서 Amazon EFS 파일 시스템 및 Amazon ECS 클러스터에 대해 지정한 VPC를 선택합니다. 서브넷과 2단계: Amazon EC2 인스턴스 및 Amazon EFS 파일 시스템의 보안 그룹 생성에서 생성한 인스턴스 보안 그룹을 선택합니다. 인스턴스의 보안 그룹을 구성합니다. 퍼블릭 IP 자동 할당이 활성화되어 있는지 확인합니다.
7. 스토리지 구성에서 파일 시스템의 편집 버튼을 선택한 다음 EFS를 선택합니다. 3단계: Amazon EFS 파일 시스템 생성에서 생성한 파일 시스템을 선택합니다. 선택적으로 탑재 지점을 변경하거나 기본값을 그대로 둘 수 있습니다.

> **Important**
>
> 서브넷을 선택해야만 인스턴스에 파일 시스템을 추가할 수 있습니다.

8. 보안 그룹 자동 생성 및 연결을 선택 취소합니다. 다른 확인란은 선택된 상태로 둡니다. 공유 파일 시스템 추가(Add shared file system)를 선택합니다.
9. 고급 세부 정보(Advanced Details)에서 Amazon EFS 파일 시스템 탑재 단계를 사용하여 사용자 데이터 스크립트가 자동으로 채워져 있는지 확인합니다.
10. 요약에서 인스턴스 수가 1인지 확인합니다. 인스턴스 시작을 선택합니다.
11. 인스턴스 시작 페이지에서 모든 인스턴스 보기를 선택하여 인스턴스의 상태를 확인합니다. 처음에는 인스턴스 상태가 `PENDING`입니다. 상태가 `RUNNING`으로 변경되고 인스턴스가 모든 상태 확인을 통과하면 인스턴스를 사용할 준비가 된 것입니다.

이제 Amazon EC2 인스턴스에 연결하고 Amazon EFS 파일 시스템에 콘텐츠를 추가합니다.

Amazon EC2 인스턴스에 연결하고 Amazon EFS 파일 시스템에 콘텐츠를 추가하려면

1. SSH를 사용하여 생성된 Amazon EC2 인스턴스에 연결합니다. 자세한 내용은 Amazon EC2 사용 설명서의 SSH를 사용하여 Linux 인스턴스에 연결을 참조하세요.
2. 터미널 창에서 df -T 명령을 실행하여 Amazon EFS 파일 시스템이 탑재되었는지 확인합니다. 다음 출력에서 Amazon EFS 파일 시스템 탑재를 강조했습니다.

```
$ df -T
Filesystem     Type            1K-blocks    Used        Available Use% Mounted on
devtmpfs       devtmpfs           485468       0           485468   0% /dev
tmpfs          tmpfs              503480       0           503480   0% /dev/shm
tmpfs          tmpfs              503480     424           503056   1% /run
tmpfs          tmpfs              503480       0           503480   0% /sys/fs/
cgroup
/dev/xvda1     xfs               8376300 1310952          7065348  16% /
127.0.0.1:/    nfs4     9007199254739968       0 9007199254739968   0% /mnt/efs/fs1
tmpfs          tmpfs              100700       0           100700   0% /run/
user/1000
```

3. Amazon EFS 파일 시스템이 탑재된 디렉터리로 이동합니다. 위 예제에서는 `/mnt/efs/fs1`이 여기에 해당합니다.
4. 다음 콘텐츠가 포함된 `index.html`이라는 파일을 생성합니다.

```
<html>
    <body>
        <h1>It Works!</h1>
        <p>You are using an Amazon EFS file system for persistent container
 storage.</p>
```

```
    </body>
</html>
```

5단계: 태스크 정의 생성

다음 태스크 정의는 `efs-html`이라는 데이터 볼륨을 생성합니다. `nginx` 컨테이너는 NGINX 루트 `/usr/share/nginx/html`에서 호스트 데이터 볼륨을 탑재합니다.

Amazon ECS 콘솔을 사용하여 새 작업 정의를 생성하는 방법

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 작업 정의를 선택합니다.
3. 새 태스크 정의 생성(Create new task definition), JSON으로 새 태스크 정의 생성(Create new task definition with JSON)을 선택합니다.
4. JSON 편집기 상자에서 다음 JSON 텍스트를 복사하여 붙여넣고, `fileSystemId`를 Amazon EFS 파일 시스템의 ID로 바꿉니다.

```
{
    "containerDefinitions": [
        {
            "memory": 128,
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "mountPoints": [
                {
                    "containerPath": "/usr/share/nginx/html",
                    "sourceVolume": "efs-html"
                }
            ],
            "name": "nginx",
            "image": "public.ecr.aws/docker/library/nginx:latest"
        }
    ],
    "volumes": [
        {
```

```
            "name": "efs-html",
            "efsVolumeConfiguration": {
                "fileSystemId": "fs-1324abcd",
                "transitEncryption": "ENABLED"
            }
        }
    ],
    "family": "efs-tutorial",
    "executionRoleArn":"arn:aws:iam::111122223333:role/ecsTaskExecutionRole"
}
```

> **Note**
>
> Amazon ECS 태스크 실행 IAM 역할에는 Amazon EFS 파일 시스템을 탑재하는 데 특정 Amazon EFS 관련 권한이 필요하지 않습니다. Amazon EFS 리소스 기반 정책이 없는 경우 파일 시스템 생성 시 기본적으로 모든 위탁자(*)에게 액세스 권한이 부여됩니다. Amazon ECS 태스크 정의에서 'EFS IAM 권한 부여'가 활성화된 경우에만 Amazon ECS 태스크 역할이 필요합니다. 활성화된 경우 Amazon EFS 리소스 기반 정책에서 태스크 역할 자격 증명에 대해 Amazon EFS 파일 시스템에 대한 액세스가 허용되어야 하며 익명 액세스는 비활성화되어야 합니다.

5. 생성(Create)을 선택합니다.

6단계: 작업 실행 및 결과 보기

이제 Amazon EFS 파일 시스템이 생성되고 NGINX 컨테이너에 사용할 웹 콘텐츠가 있으므로 생성된 태스크 정의를 사용하여 태스크를 실행할 수 있습니다. NGINX 웹 서버는 간단한 HTML 페이지를 제공합니다. Amazon EFS 파일 시스템에서 콘텐츠를 업데이트한 경우 변경 사항이 해당 파일 시스템도 탑재한 컨테이너에 전파됩니다.

작업은 클러스터에 정의한 서브넷에서 실행됩니다.

콘솔을 사용하여 작업을 실행하고 결과를 보려면 다음을 수행하세요.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터 페이지에서 독립 실행형 태스크를 실행하는 클러스터를 선택합니다.

   서비스를 시작할 리소스를 결정합니다.

서비스 시작	단계
클러스터	a. 클러스터 페이지에서 서비스를 생성할 클러스터를 선택합니다. b. 작업 탭에서 새 작업 실행을 선택합니다.
시작 유형	a. Task(작업) 페이지에서 작업 정의를 선택합니다. b. 개정이 둘 이상 있는 경우 개정을 선택합니다. c. Create(생성), Run task(작업 실행)를 선택합니다.

3. (선택 사항) 예약된 태스크가 클러스터 인프라에 배포되는 방식을 선택합니다. 컴퓨팅 구성(Compute configuration)을 열고 다음을 수행합니다.

배포 방법	단계
시작 유형	a. 컴퓨팅 옵션(Compute options) 섹션에서 시작 유형(Launch type)을 선택합니다. b. Launch Type(시작 유형)을 EC2로 선택합니다.

4. 애플리케이션 유형(Application type)에서 작업(Task)을 선택합니다.
5. 작업 정의의 경우 앞서 생성한 `efs-tutorial` 작업 정의를 선택합니다.
6. Desired tasks(원하는 작업)에 1을 입력합니다.
7. 생성(Create)을 선택합니다.
8. Cluster(클러스터) 페이지에서 Infrastructure(인프라)를 선택합니다.
9. 컨테이너 인스턴스에서 연결할 컨테이너 인스턴스를 선택합니다.
10. 컨테이너 인스턴스페이지에서 네트워킹 아래에서 인스턴스의 퍼블릭 IP를 기록합니다.

11. 브라우저를 열고 퍼블릭 IP 주소를 입력합니다. 다음과 메시지가 표시됩니다.

```
It works!
You are using an Amazon EFS file system for persistent container storage.
```

> **Note**
>
> 메시지가 표시되지 않는 경우 컨테이너 인스턴스의 보안 그룹이 포트 80에서 인바운드 네트워크 트래픽을 허용하고 파일 시스템의 보안 그룹이 컨테이너 인스턴스에서 인바인드 액세스를 허용하는지 확인합니다.

Amazon ECS에서 FSx for Windows File Server 볼륨 사용

FSx for Windows File Server는 Windows 파일 시스템이 지원하는 완전관리형 Windows 파일 서버를 제공합니다. FSx for Windows File Server와 ECS를 함께 사용하면 영구적이고 분산, 공유된 고정 파일 스토리지로 Windows 태스크를 프로비저닝할 수 있습니다. 자세한 정보는 FSx for Windows File Server란 무엇인가요?를 참조하세요.

> **Note**
>
> Amazon ECS 최적화 Windows Server 2016 Full AMI을 사용하는 EC2 인스턴스는 FSx for Windows File Server ECS 태스크 볼륨을 지원하지 않습니다.
> Fargate 구성의 Windows 컨테이너에서 Windows 파일 서버 볼륨용 FSx를 사용할 수 없습니다. 대신 시작 시 컨테이너를 탑재하도록 수정할 수 있습니다.

FSx for Windows File Server를 사용하여 공유된 외부 스토리지, 고가용성 리전 스토리지, 고성능 스토리지에 액세스해야 하는 Windows 워크로드를 배포할 수 있습니다. 하나 이상의 FSx for Windows File Server 파일 시스템 볼륨을 Amazon ECS Windows 인스턴스에 실행되는 Amazon ECS 컨테이너에 탑재할 수 있습니다. 단일 Amazon ECS 작업 내 여러 Amazon ECS 컨테이너 사이에서 FSx for Windows File Server 파일 시스템 볼륨을 공유할 수 있습니다.

ECS로 FSx for Windows File Server를 사용하도록 지원하려면 FSx for Windows File Server 파일 시스템 ID와 관련 정보를 태스크 정의에 포함해야 합니다. 다음은 태스크 정의 JSON 코드 조각의 예제입니다. 태스크 정의를 생성, 실행하기 전에 다음이 필요합니다.

- 유효한 도메인에 가입된 ECS Windows EC2 인스턴스. Amazon EC2의 온프레미스 Active Directory 또는 자체 Active Directory인 AWS Directory Service for Microsoft Active Directory에서 호스팅할 수 있습니다.
- Active Directory 도메인을 결합하고 FSx for Windows File Server 파일 시스템을 연결하는 데 사용하는 자격 증명이 포함된 AWS Secrets Manager 보안 암호 또는 Systems Manager 파라미터. 자격 증명 값은 Active Directory를 생성할 때 입력하는 이름과 암호 자격 증명입니다.

관련 자습서는 Amazon ECS에 대한 FSx for Windows File Server 파일 시스템을 구성하는 방법 알아보기 섹션을 참조하세요.

고려 사항

FSx for Windows File Server 볼륨을 사용할 때 다음을 고려하세요.

- FSx for Windows File Server와 Amazon ECS는 Windows Amazon EC2 인스턴스만 지원합니다. Linux Amazon EC2 인스턴스는 지원하지 않습니다.
- FSx for Windows File Server와 Amazon ECS는 AWS Fargate를 지원하지 않습니다.
- `awsvpc` 네트워크 모드의 FSx for Windows File Server와 Amazon ECS는 컨테이너 에이전트의 버전 `1.54.0` 이상이 필요합니다.
- Amazon ECS 태스크에 사용할 수 있는 드라이브 문자의 최대 개수는 23개입니다. FSx for Windows File Server 볼륨이 포함된 각 태스크는 드라이브 문자가 할당됩니다.
- 기본적으로 태스크 리소스 정리 시간은 작업이 끝나고 3시간 후입니다. 태스크에서 생성한 파일 매핑은 사용하는 태스크가 없어도 3시간 동안 유지됩니다. 기본 정리 시간은 Amazon ECS 환경 변수 `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION`을 사용하여 구성할 수 있습니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.
- 일반적으로 태스크는 FSx for Windows File Server 파일 시스템과 동일한 VPC에서 실행됩니다. 그러나 Amazon ECS 클러스터 VPC와 FSx for Windows File Server 파일 시스템 사이에 VPC 피어링을 통한 네트워크 연결이 설정되어 있다면 VPC 간 지원이 가능합니다.
- VPC 보안 그룹을 구성하면 네트워크 수준에서 FSx for Windows File Server 파일 시스템에 대한 액세스를 제어할 수 있습니다. Active Directory 보안 그룹이 올바르게 구성된 Active Directory 도메인과 결합된 EC2 인스턴스에서 호스팅되는 작업만 FSx for Windows File Server 파일 공유에 액세스할 수 있습니다. 보안 그룹이 잘못 구성되면 Amazon ECS는 “`unable to mount file system` *`fs-id`*” 오류 메시지와 함께 태스크를 시작하는 데 실패합니다.
- FSx for Windows File Server는 AWS Identity and Access Management(IAM)과 통합되어 IAM 사용자와 그룹이 특정 FSx for Windows File Server 리소스에 취할 수 있는 태스크를 제어합니다. 고객은 클라이언트 권한 부여로 특정 FSx for Windows File Server 파일 시스템에 대한 액세스를 허용하

거나 거부하는 IAM 역할을 정의할 수 있고, 선택적으로 읽기 전용 액세스를 요구하고 클라이언트에서 파일 시스템에 대한 루트 액세스를 허용 또는 거부할 수 있습니다. 자세한 정보는 Amazon FSx Windows 사용 설명서의 보안을 참조하세요.

Amazon ECS에서 FSx for Windows File Server 사용 모범 사례

Amazon ECS에서 FSx for Windows File Server를 사용할 때 이 모범 사례 권장 사항을 참고하세요.

FSx for Windows File Server의 보안 및 액세스 제어

FSx for Windows File Server는 FSx for Windows File Server 파일 시스템에 저장된 데이터를 보안하고 해당 데이터가 필요한 애플리케이션에서만 데이터에 액세스할 수 있도록 보장하는 데 사용할 수 있는 액세스 제어 기능을 제공합니다.

FSx for Windows File Server 볼륨의 데이터 암호화

FSx for Windows File Server는 파일 시스템에 대해 두 가지 암호화 양식을 지원합니다. 저장 중 데이터 및 전송 중 데이터 암호화 기능이 이에 해당합니다. 전송 중 데이터의 암호화는 SMB 프로토콜 3.0 이상을 지원하는 컨테이너 인스턴스에 매핑된 파일 공유에서 지원됩니다. Amazon FSx 파일 시스템을 생성할 때 저장 데이터 암호화가 자동으로 활성화됩니다. Amazon FSx는 애플리케이션을 수정할 필요 없이 파일 시스템에 액세스할 때 SMB 암호화를 사용하여 전송 중 데이터를 자동으로 암호화합니다. 자세한 내용은 Amazon FSx for Windows File Server 사용 설명서의 Data encryption in Amazon FSx를 참조하세요.

폴더 수준 액세스 제어를 위해 Windows ACL 사용

Windows Amazon EC2 인스턴스는 Active Directory 자격 증명을 사용하여 Amazon FSx 파일 공유에 액세스합니다. 파일 및 폴더 수준 및 세분화된 수준의 액세스 제어를 위해 표준 Windows 액세스 제어 목록(ACL)을 사용합니다. 공유 내 특정 폴더에 대해 각각 여러 자격 증명을 생성할 수 있으며, 이 자격 증명은 특정 작업에 매핑됩니다.

다음 예제에서 작업은 Secrets Manager에 저장된 자격 증명을 사용하여 App01 폴더에 액세스할 수 있습니다. 해당 Amazon 리소스 이름(ARN)은 1234입니다.

```
"rootDirectory": "\\path\\to\\my\\data\App01",
"credentialsParameter": "arn-1234",
"domain": "corp.fullyqualified.com",
```

또 다른 예제로, 작업은 Secrets Manager에 저장된 자격 증명을 사용하여 App02 폴더에 액세스할 수 있습니다. ARN은 6789입니다.

```
"rootDirectory": "\\path\\to\\my\\data\App02",
"credentialsParameter": "arn-6789",
"domain": "corp.fullyqualified.com",
```

Amazon ECS 작업 정의에서 FSx for Windows File Server 파일 시스템 지정

컨테이너에 FSx for Windows File Server 파일 시스템 볼륨을 사용하려면 태스크 정의에 볼륨 및 마운트 지점 구성을 지정합니다. 다음 태스크 정의 JSON 코드 조각은 컨테이너에 사용할 `volumes` 및 `mountPoints` 객체의 구문을 나타냅니다.

```
{
    "containerDefinitions": [
        {
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "portMappings": [],
            "command": ["New-Item -Path C:\\fsx-windows-dir\\index.html -ItemType file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>It Works!</h2> <p>You are using Amazon
 FSx for Windows File Server file system for persistent container storage.</p>' -
Force"],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": false,
            "name": "container1",
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
                }
            ]
        },
        {
            "entryPoint": [
                "powershell",
                "-Command"
            ],
```

```
            "portMappings": [
                {
                    "hostPort": 443,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": ["Remove-Item -Recurse C:\\inetpub\\wwwroot\\* -Force; Start-
Sleep -Seconds 120; Move-Item -Path C:\\fsx-windows-dir\\index.html -Destination C:\
\inetpub\\wwwroot\\index.html -Force; C:\\ServiceMonitor.exe w3svc"],
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
                }
            ],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": true,
            "name": "container2"
        }
    ],
    "family": "fsx-windows",
    "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
    "volumes": [
        {
            "name": "fsx-windows-dir",
            "fsxWindowsFileServerVolumeConfiguration": {
                "fileSystemId": "fs-0eeb5730b2EXAMPLE",
                "authorizationConfig": {
                    "domain": "example.com",
                    "credentialsParameter": "arn:arn-1234"
                },
                "rootDirectory": "share"
            }
        }
    ]
}
```

`FSxWindowsFileServerVolumeConfiguration`

유형: 객체

필수 여부: 아니요

이 파라미터는 태스크 저장에 FSx for Windows File Server 파일 시스템을 사용할 때 지정됩니다.

`fileSystemId`

유형: 문자열

필수 항목 여부: 예

사용할 FSx for Windows File Server 파일 시스템 ID입니다.

`rootDirectory`

유형: 문자열

필수 항목 여부: 예

호스트 내의 루트 디렉터리로 탑재할 FSx for Windows File Server 파일 시스템 내 디렉터리입니다.

`authorizationConfig`

`credentialsParameter`

유형: 문자열

필수 항목 여부: 예

권한 부여 자격 증명 옵션:

- Secrets Manager 암호의 Amazon 리소스 이름(ARN).
- Systems Manager 파라미터의 Amazon 리소스 이름(ARN).

`domain`

유형: 문자열

필수 항목 여부: 예

셀프 호스팅된 EC2 Active Directory 또는 AWS Directory Service for Microsoft Active Directory(AWS Managed Microsoft AD) 디렉터리에서 호스팅하는 정규화된 도메인 이름입니다.

FSx for Windows File Server 볼륨 자격 증명을 저장하는 방법

자격 증명 파라미터와 함께 사용할 자격 증명을 저장하는 방법에는 두 가지가 있습니다.

- AWS Secrets Manager 암호

  이 자격 증명은 다른 암호 유형 카테고리를 사용하여 AWS Secrets Manager 콘솔에서 생성할 수 있습니다. 각 키/값 쌍, 사용자 이름/관리자 및 암호/##에 대한 열을 추가합니다.

- Systems Manager 파라미터

  이 자격 증명은 다음 예제 코드 조각에 표시된 형식으로 텍스트를 입력하여 Systems Manager 파라미터 콘솔에서 생성될 수 있습니다.

```
{
  "username": "admin",
  "password": "password"
}
```

태스크 정의 `FSxWindowsFileServerVolumeConfiguration` 파라미터의 `credentialsParameter`는 암호 ARN 또는 Systems Manager 파라미터 ARN을 보유합니다. 자세한 정보는 Secrets Manager 사용 설명서의 AWS Secrets Manager란 무엇인가요? 및 Systems Manager 사용 설명서의 Systems Manager 파라미터 스토어를 참조하세요.

Amazon ECS에 대한 FSx for Windows File Server 파일 시스템을 구성하는 방법 알아보기

FSx for Windows File Server 파일 시스템과 파일 시스템에 액세스할 수 있는 컨테이너를 호스팅하는 Amazon ECS 최적화 Windows 인스턴스를 시작하는 방법에 대해 알아보세요. 이렇게 하려면 먼저 AWS Directory Service AWS 관리형 Microsoft Active Directory를 생성합니다. 그런 다음 Amazon EC2 인스턴스 및 태스크 정의를 사용하여 FSx for Windows File Server 파일 시스템 및 클러스터를 생성합니다. FSx for Windows File Server 파일 시스템을 사용하도록 컨테이너에 대한 태스크 정의를 구성합니다. 마지막으로 파일 시스템을 테스트합니다.

이 태스크는 Active Directory 또는 FSx for Windows File Server 파일 시스템을 시작하거나 삭제할 때마다 20~45분 정도 걸립니다. 자습서를 완료하거나 몇 세션 동안 자습서를 완료하기 위해 90분 이상 예약할 준비를 합니다.

자습서의 사전 조건

- 관리 사용자입니다. Amazon ECS 사용 설정을(를) 참조하세요.

- (선택 사항) RDP 액세스를 통해 EC2 Windows 인스턴스에 연결하기 위한 PEM 키 페어. 키 페어를 생성하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 키 페어 및 Amazon 인스턴스를 참조하세요.
- 하나 이상의 퍼블릭 서브넷과 프라이빗 서브넷, 보안 그룹이 있는 VPC. 기본 VPC를 사용할 수 있습니다. NAT 게이트웨이 또는 장치가 필요하지 않습니다. AWS Directory Service는 Active Directory에서 NAT(네트워크 주소 변환)를 지원하지 않습니다. 이 작업을 수행하려면 VPC 내에 Active Directory, FSx for Windows File Server 파일 시스템, ECS 클러스터 및 EC2 인스턴스가 있어야 합니다. VPC와 Active Directory에 대한 자세한 내용은 VPC 생성 및 AWS 관리형 Microsoft AD 생성을 위한 사전 조건을 참조하세요.
- IAM ecsInstanceRole 및 ecsTaskExecutionRole 권한은 계정과 연결됩니다. 이러한 서비스 연결 역할을 통해 서비스가 API 호출을 수행하고 사용자를 대신하여 컨테이너, 보안 암호, 디렉터리 및 파일 서버에 액세스할 수 있습니다.

1단계: IAM 액세스 역할 생성

AWS Management Console을 사용하여 클러스터를 생성합니다.

1. ecsInstanceRole이 있는지 확인하고 없는 경우 생성 방법을 확인하려면 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.
2. 실제 프로덕션 환경에서는 최소 사용 권한에 맞게 역할 정책을 사용자 지정하는 것이 좋습니다. 이 자습서를 통해 작업하기 위해 다음 AWS 관리형 정책이 ecsInstanceRole에 연결되었는지 확인합니다. 아직 연결되어 있지 않은 경우 정책을 연결합니다.

   - AmazonEC2ContainerServiceforEC2Role
   - AmazonSSMManagedInstanceCore
   - AmazonSSMDirectoryServiceAccess

   AWS 관리형 정책을 연결하려면

   a. IAM 콘솔(IAM console)을 엽니다.
   b. 탐색 창에서 역할를 선택합니다.
   c. AWS 관리형 역할을 선택합니다.
   d. 권한, 정책 연결을 선택합니다.
   e. 연결에 사용할 수 있는 정책의 범위를 좁히려면 필터(Filter)를 사용합니다.
   f. 적절한 정책을 선택하고 정책 연결(Attach policy)을 선택합니다.

3. ecsTaskExecutionRole이 있는지 확인하고 없는 경우 생성 방법을 확인하려면 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

   실제 프로덕션 환경에서는 최소 사용 권한에 맞게 역할 정책을 사용자 지정하는 것이 좋습니다. 이 자습서를 통해 작업하기 위해 다음 AWS 관리형 정책이 ecsTaskExecutionRole에 연결되었는지 확인합니다. 정책이 아직 연결되어 있지 않은 경우 정책을 연결합니다. 이전 섹션에 제시한 절차를 통해 AWS 관리형 정책에 연결합니다.

   - SecretsManagerReadWrite
   - AmazonFSxReadOnlyAccess
   - AmazonSSMReadOnlyAccess
   - AmazonECSTaskExecutionRolePolicy

2단계: Windows Active Directory(AD) 생성

1. AWS Directory Service 관리 안내서의 AWS 관리형 Microsoft AD 생성에 설명된 단계를 따릅니다. 이 자습서에서 지정한 VPC를 사용합니다. AWS 관리형 Microsoft AD 생성의 3단계에서 다음 단계에 사용할 사용자 이름과 관리자 암호를 저장합니다. 또한 향후 단계를 위해 정규화된 디렉터리 DNS 이름을 적어 둡니다. Active Directory를 생성하는 동안 다음 단계를 완료할 수 있습니다.
2. 다음 단계에서 사용할 AWS Secrets Manager 보안 암호를 생성합니다. AWS Secrets Manager 사용 설명서의 Secrets Manager 시작하기를 참조하세요.

   a. Secrets Manager 콘솔을 엽니다.
   b. 새 보안 암호 저장(Store a new secret)을 클릭합니다.
   c. 다른 유형의 보안 암호(Other type of secrets)를 선택합니다.
   d. 첫 번째 행의 보안 암호 키/값(Secret key/value)에서 값이 **admin**인 키 **username**를 생성합니다. +행 추가(+ Add row)를 클릭합니다.
   e. 새 행에서 키 **password**를 생성합니다. 값에 AWS 관리형 AD 디렉터리 생성의 3단계에서 입력한 암호를 입력합니다.
   f. 다음(Next) 버튼을 클릭합니다.
   g. 보안 암호 이름 및 설명을 입력합니다. 다음(Next)을 클릭합니다.
   h. 다음을 클릭합니다. 저장(Store)을 클릭합니다.
   i. 보안 암호 페이지에서 방금 생성한 보안 암호를 클릭합니다.
   j. 다음 단계에서 사용할 새 보안 암호의 ARN을 저장합니다.
   k. Active Directory를 생성하는 동안 다음 단계로 진행할 수 있습니다.

3단계: 보안 그룹 확인 및 업데이트

이 단계에서는 사용 중인 보안 그룹에 대한 규칙을 확인하고 업데이트합니다. 이를 위해 VPC에 대해 생성한 기본 보안 그룹을 사용할 수 있습니다.

보안 그룹을 확인하고 업데이트합니다.

포트 간에 데이터를 전송하려면 보안 그룹을 생성하거나 편집해야 합니다. 자세한 정보는 FSx for Windows File Server 사용 설명서의 Amazon VPC 보안 그룹에서 설명합니다. 이 태스크는 다음 인바운드 규칙 표의 첫 번째 행에 표시된 보안 그룹 인바운드 규칙을 생성하여 수행할 수 있습니다. 이 규칙은 보안 그룹에 할당된 네트워크 인터페이스(및 연결된 인스턴스)로부터의 인바운드 트래픽을 허용합니다. 생성한 모든 클라우드 리소스는 동일한 VPC 내에 있으며 동일한 보안 그룹에 연결됩니다. 따라서 이 규칙을 사용하면 필요에 따라 FSx for Windows File Server 파일 시스템, Active Directory 및 ECS 인스턴스에서 트래픽을 주고받을 수 있습니다. 다른 인바운드 규칙은 트래픽이 웹사이트 및 ECS 인스턴스에 연결하기 위한 RDP 액세스를 제공하도록 허용합니다.

다음 표에서는 이 자습서에 필요한 보안 그룹 인바운드 규칙을 보여 줍니다.

유형	프로토콜	포트 범위	소스
모든 트래픽	모두	모두	*## ##(SG)*
HTTPS	TCP	443	0.0.0.0/0
RDP	TCP	3389	랩톱 IP 주소

다음 표에서는 이 자습서에 필요한 보안 그룹 아웃바운드 규칙을 보여 줍니다.

유형	프로토콜	포트 범위	대상
모든 트래픽	모두	모두	0.0.0.0/0

1. EC2 콘솔을 열고 왼쪽 메뉴에서 보안 그룹(Security Groups)을 선택합니다.
2. 이제 표시된 보안 그룹 목록에서 이 자습서에서 사용하는 보안 그룹의 왼쪽에 있는 확인란을 선택합니다.

보안 그룹 정보가 표시됩니다.

3. 인바운드 규칙(Inbound rules) 또는 아웃바운드 규칙(Outbound rules) 탭을 클릭하고 인바운드 규칙 편집(Edit inbound rules) 또는 아웃바운드 규칙 편집(Edit outbound rules) 버튼을 선택하여 인바운드 및 아웃바운드 규칙을 편집합니다. 이전 표에 표시된 규칙과 일치하도록 규칙을 편집합니다. 이 자습서의 뒷부분에서 EC2 인스턴스를 생성한 후, Amazon EC2 사용 설명서의 RDP를 사용하여 Windows 인스턴스에 연결에서 설명한 것처럼 EC2 인스턴스의 퍼블릭 IP 주소로 인바운드 규칙 RDP 소스를 편집합니다.

4단계: Amazon FSx for Windows File Server 파일 시스템 생성

보안 그룹을 확인 및 업데이트하고 Active Directory를 생성하여 활성 상태가 되면 Active Directory와 동일한 VPC에 FSx for Windows File Server 파일 시스템을 생성합니다. 다음 단계에 따라 Windows File Server 파일 시스템을 Windows 작업에 사용할 FSx for Windows File Server 파일 시스템을 생성합니다.

첫 파일 시스템을 생성합니다.

1. Amazon FSx 콘솔을 엽니다.
2. 대시보드에서 파일 시스템 생성(Create file system)을 선택해 파일 시스템 생성 마법사를 시작합니다.
3. 파일 시스템 유형 선택 페이지에서 FSx for Windows File Server를 선택한 다음 다음을 선택합니다. 파일 시스템 생성 페이지가 표시됩니다.
4. 파일 시스템 정보(File system details) 섹션에서 파일 시스템의 이름을 입력합니다. 파일 시스템의 이름을 지정하면 파일 시스템을 보다 쉽게 찾고 관리할 수 있습니다. 최대 256자의 유니코드 문자를 사용할 수 있습니다. 허용되는 문자는 문자, 숫자, 공백과 특수 문자 +기호(+), 빼기 기호(-), 등호(=), 마침표(.), 밑줄(_), 콜론(:) 및 슬래시(/)입니다.
5. 배포 유형(Deployment type)에서 단일 AZ(Single-AZ)를 사용하여 단일 가용 영역에 배포되는 파일 시스템을 배포합니다. 단일 AZ 2는 단일 가용 영역 파일 시스템의 최신 세대이며 SSD 및 HDD 스토리지를 지원합니다.
6. 스토리지 유형(Storage type)에서 HDD를 선택합니다.
7. 스토리지 용량(Storage capacity)에 최소 스토리지 용량을 입력합니다.
8. 용량 처리량(Throughput capacity)을 기본 설정으로 유지합니다.
9. 네트워크 및 보안(Network & security) 섹션에서 AWS Directory Service 디렉터리에 선택한 것과 동일한 Amazon VPC를 선택합니다.

10. VPC 보안 그룹(VPC Security Groups)에서 3단계: 보안 그룹 확인 및 업데이트에서 확인한 보안 그룹을 선택합니다.
11. Windows 인증(Windows authentication)에서 AWS 관리형 Microsoft Active Directory를 선택한 다음 목록에서 AWS Directory Service 디렉터리를 선택합니다.
12. 암호화(Encryption)에서 기본 암호화 키(Encryption key) 설정을 aws/fsx(기본값)로 그대로 사용합니다.
13. 유지 관리 기본 설정(Maintenance preferences)의 기본 설정을 유지합니다.
14. 다음(Next) 버튼을 클릭합니다.
15. 파일 시스템 생성 페이지에 표시된 파일 시스템 구성을 검토합니다. 참조할 수 있도록 파일 시스템을 생성한 후 수정할 수 있는 파일 시스템 설정을 확인합니다. 파일 시스템 생성을 선택합니다.
16. 파일 시스템 ID를 기록합니다. 이후 단계에서 사용해야 합니다.

    FSx for Windows File Server 시스템이 생성되는 동안 다음 단계로 이동하여 클러스터 및 EC2 인스턴스를 생성할 수 있습니다.

5단계: Amazon ECS 클러스터 생성

Amazon ECS 콘솔을 사용하여 클러스터 생성

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터(Clusters) 페이지에서 클러스터 생성(Create cluster)을 선택합니다.
5. 클러스터 구성의 클러스터 이름에 windows-fsx-cluster를 입력합니다.
6. 인프라에서 AWS Fargate(서버리스)를 선택 취소한 다음 Amazon EC2 인스턴스를 선택합니다.
   - Auto Scaling 그룹을 생성하려면 Auto Scaling 그룹(ASG)(Auto Scaling group (ASG))에서 새 그룹 생성(Create new group)을 선택한 후 그룹에 대한 다음 세부 정보를 제공합니다.
     - 운영 체제/아키텍처의에서 Windows Server 2019 Core를 선택합니다.
     - EC2 인스턴스 유형에서 t2.medium 또는 t2.micro를 선택합니다.
7. 생성(Create)을 선택합니다.

6단계: Amazon ECS 최적화 Amazon EC2 인스턴스 생성

Amazon ECS Windows 컨테이너 인스턴스를 생성합니다.

Amazon ECS 인스턴스 생성

1. `aws ssm get-parameters` 명령을 사용하여 VPC를 호스팅하는 리전에 대한 AMI 이름을 검색합니다. 자세한 내용은 Amazon ECS 최적화 AMI 메타데이터 검색을 참조하세요.
2. Amazon EC2 콘솔을 사용하여 인스턴스를 시작합니다.
   a. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
   b. 탐색 모음에서 사용할 리전을 선택합니다.
   c. EC2 대시보드에서 인스턴스 시작(Launch Instance)을 선택합니다.
   d. Name(이름)에 고유한 이름을 입력합니다.
   e. 애플리케이션 및 OS 이미지(Amazon Machine Image)에서 검색 필드에 첫 번째 단계에서 검색한 AMI 이름을 입력합니다.
   f. 인스턴스 유형에서 t2.medium 또는 t2.micro를 선택합니다.
   g. Key pair (login)(키 페어(로그인))에서 키 페어를 선택합니다. 키 페어를 지정하지 않은 경우
   h. 네트워크 설정에서 VPC 및 서브넷에 대해 VPC 및 퍼블릭 서브넷을 선택합니다.
   i. Network settings(네트워크 설정)의 Security group(보안 그룹)에서 기존 보안 그룹을 선택하거나 새 보안 그룹을 생성합니다. 선택한 보안 그룹에 자습서의 사전 조건에 정의된 인바운드 및 아웃바운드 규칙이 있는지 확인합니다.
   j. Network settings(네트워크 설정)의 Auto-assign Public IP(퍼블릭 IP 자동 할당)에서 Enable(활성화)을 선택합니다.
   k. 고급 세부 정보를 확장하고 도메인 조인 디렉터리에서 생성한 Active Directory의 ID를 선택합니다. 이 옵션 도메인은 EC2 인스턴스가 시작될 때 AD에 조인합니다.
   l. Advanced details(고급 세부 정보)의 IAM instance profile(IAM 인스턴스 프로파일)에서 ecsInstanceRole을 선택합니다.
   m. 다음 사용자 데이터로 Amazon ECS 컨테이너 인스턴스를 구성합니다. Advanced Details(고급 세부 정보)에서 다음 스크립트를 User data(사용자 데이터) 필드에 붙여 넣고 *`cluster_name`*을 클러스터의 이름으로 바꿉니다.

```
<powershell>
Initialize-ECSAgent -Cluster windows-fsx-cluster -EnableTaskIAMRole
</powershell>
```

n. 준비가 되었으면 승인 필드를 선택한 다음 인스턴스 시작(Launch Instances)을 선택합니다.

o. 확인 페이지에서 인스턴스가 실행 중인지 확인할 수 있습니다. 인스턴스 보기를 선택하여 확인 페이지를 닫고 콘솔로 돌아갑니다.

3. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
4. 탐색 창에서 클러스터를 선택한 다음 windows-fsx-cluster를 선택합니다.
5. 인프라 탭을 선택하고 인스턴스가 windows-fsx-cluster 클러스터에 등록되었는지 확인합니다.

7단계: Windows 태스크 정의 등록

Amazon ECS 클러스터에서 Windows 컨테이너를 실행하려면 먼저 태스크 정의를 등록해야 합니다. 다음 작업 정의 예제는 간단한 웹페이지를 보여 줍니다. 이 작업으로 FSx 파일 시스템에 액세스할 수 있는 두 개의 컨테이너가 시작됩니다. 첫 번째 컨테이너는 HTML 파일을 파일 시스템에 작성합니다. 두 번째 컨테이너는 파일 시스템에서 HTML 파일을 다운로드하고 웹페이지를 제공합니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 작업 정의를 선택합니다.
3. 새 태스크 정의 생성(Create new task definition), JSON으로 새 태스크 정의 생성(Create new task definition with JSON)을 선택합니다.
4. JSON 편집기 상자에서 작업 실행 역할의 값과 FSx 파일 시스템에 대한 세부 정보를 바꾼 다음 저장을 선택합니다.

```
{
    "containerDefinitions": [
        {
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "portMappings": [],
            "command": ["New-Item -Path C:\\fsx-windows-dir\\index.html -ItemType
 file -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body
 {margin-top: 40px; background-color: #333;} </style> </head><body> <div
 style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>It
 Works!</h2> <p>You are using Amazon FSx for Windows File Server file system for
 persistent container storage.</p>' -Force"],
            "cpu": 512,
            "memory": 256,
```

```
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": false,
            "name": "container1",
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
                }
            ]
        },
        {
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "portMappings": [
                {
                    "hostPort": 443,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": ["Remove-Item -Recurse C:\\inetpub\\wwwroot\\* -Force;
 Start-Sleep -Seconds 120; Move-Item -Path C:\\fsx-windows-dir\\index.html -
Destination C:\\inetpub\\wwwroot\\index.html -Force; C:\\ServiceMonitor.exe
 w3svc"],
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
                }
            ],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": true,
            "name": "container2"
        }
    ],
    "family": "fsx-windows",
```

```
    "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
    "volumes": [
        {
            "name": "fsx-windows-dir",
            "fsxWindowsFileServerVolumeConfiguration": {
                "fileSystemId": "fs-0eeb5730b2EXAMPLE",
                "authorizationConfig": {
                    "domain": "example.com",
                    "credentialsParameter": "arn:arn-1234"
                },
                "rootDirectory": "share"
            }
        }
    ]
}
```

8단계: 작업 실행 및 결과 보기

태스크를 실행하기 전에 FSx for Windows File Server 파일 시스템의 상태가 사용 가능(Available)인지 확인합니다. 사용 가능하면 생성한 태스크 정의를 사용하여 태스크를 실행할 수 있습니다. 파일 시스템을 사용하여 HTML 파일을 섞는 컨테이너를 생성하여 태스크를 시작합니다. 셔플 후, 웹 서버는 간단한 HTML 페이지를 제공합니다.

> **Note**
>
> VPN 내에서 웹사이트에 연결하지 못할 수도 있습니다.

Amazon ECS 콘솔을 사용하여 작업을 실행하고 결과를 확인합니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택한 다음 windows-fsx-cluster를 선택합니다.
3. 작업 탭을 선택하고 새 작업 실행을 선택합니다.
4. 시작 유형(Launch Type)을 EC2로 선택합니다.
5. 배포 구성에서 작업 정의에 대해 fsx-windows를 선택한 다음 생성을 선택합니다.
6. 작업 상태가 RUNNING이면 작업 ID를 선택합니다.
7. 컨테이너에서 container1 상태가 STOPPED이면 container2를 선택하여 컨테이너의 세부 정보를 확인합니다.

8. container2의 컨테이너 세부 정보에서 네트워크 바인딩을 선택한 다음 컨테이너와 연결된 외부 IP 주소를 클릭합니다. 브라우저가 열리고 다음 메시지가 표시됩니다.

```
Amazon ECS Sample App
It Works!
You are using Amazon FSx for Windows File Server file system for persistent
 container storage.
```

> **Note**
>
> 메시지가 표시되는 데 몇 분 정도 걸릴 수 있습니다. 몇 분 후 메시지가 표시되지 않는 경우 VPN에서 실행되고 있지 않은지 확인하고 컨테이너 인스턴스의 보안 그룹이 포트 443에서 인바운드 네트워크 HTTP 트래픽을 허용하는지 확인합니다.

9단계: 정리

> **Note**
>
> FSx for Windows File Server 파일 시스템 또는 AD를 삭제하는 데 20~45분 정도 걸립니다. AD 삭제 태스크를 시작하기 전에 FSx for Windows File Server 파일 시스템의 삭제 작업이 완료될 때까지 기다려야 합니다.

FSx for Windows File Server 파일 시스템을 삭제합니다.

1. Amazon FSx 콘솔을 엽니다.
2. 방금 생성한 FSx for Windows File Server 파일 시스템의 왼쪽에 있는 라디오 버튼을 선택합니다.
3. 작업을 선택합니다.
4. 파일 시스템 삭제(Delete file system)를 선택합니다.

AD를 삭제합니다.

1. AWS Directory Service 콘솔을 엽니다.
2. 방금 생성한 AD의 왼쪽에 있는 라디오 버튼을 선택합니다.
3. 작업을 선택합니다.
4. 디렉터리 삭제(Delete directory)를 선택합니다.

클러스터를 삭제합니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택한 다음 windows-fsx-cluster를 선택합니다.
3. 클러스터 삭제를 선택합니다.
4. 상자에 구문을 입력한 다음, 삭제를 선택합니다.

EC2 인스턴스를 종료합니다.

1. Amazon EC2 콘솔을 엽니다.
2. 왼쪽 메뉴에서 인스턴스(Instances)를 선택합니다.
3. 생성한 EC2 인스턴스의 왼쪽에 있는 상자를 선택합니다.
4. 인스턴스 상태를 클릭하고 인스턴스 종료를 클릭합니다.

보안 암호를 삭제합니다.

1. Secrets Manager 콘솔을 엽니다.
2. 이 시연을 위해 생성한 보안 암호를 선택합니다.
3. 작업(Actions)을 클릭합니다.
4. 보안 암호 삭제(Delete secret)를 선택합니다.

Amazon ECS에서 Docker 볼륨 사용

Docker 볼륨을 사용할 때는 기본적으로 제공되는 `local` 드라이버 또는 타사 볼륨 드라이버를 사용할 수 있습니다. Docker 볼륨은 Docker에서 관리되며, 디렉터리는 볼륨 데이터가 포함되는 컨테이너 인스턴스의 `/var/lib/docker/volumes`에 생성됩니다.

도커 볼륨을 사용하려면 태스크 정의에서 `dockerVolumeConfiguration`을 지정합니다. 자세한 내용은 Docker 설명서 센터의 볼륨을 참조하세요.

Docker 볼륨의 몇 가지 일반 사용 사례는 다음과 같습니다.

- 컨테이너에 사용할 영구 데이터 볼륨 제공
- 동일한 컨테이너 인스턴스의 컨테이너마다 다른 위치에서 정의된 데이터 볼륨 공유
- 비어있는 비영구 데이터 볼륨을 정의한 후 동일 태스크에 속하는 다수의 컨테이너에 탑재

- 타사 드라이버에서 관리하는 태스크에 데이터 볼륨을 제공하는 방법

Docker 볼륨 사용 시 고려 사항

Docker 볼륨을 사용할 때는 다음 사항을 고려해야 합니다.

- EC2 시작 유형 또는 외부 인스턴스를 사용하는 경우에는 Docker 볼륨만 지원됩니다.
- Windows 컨테이너는 `local` 드라이버의 사용만 지원합니다.
- 타사 드라이버를 사용하는 경우에는 컨테이너 에이전트를 시작하기 전에 해당 드라이버가 컨테이너 인스턴스에 설치되어 있고 활성 상태여야 합니다. 에이전트를 시작하기 전에 타사 드라이버가 활성 상태가 아니면 다음 명령 중 하나를 사용하여 컨테이너 에이전트를 다시 시작할 수 있습니다.
  - Amazon ECS 최적화 Amazon Linux 2 AMI의 경우:

    ```
    sudo systemctl restart ecs
    ```

  - Amazon ECS 최적화 Amazon Linux AMI의 경우:

    ```
    sudo stop ecs && sudo start ecs
    ```

작업 정의에서 Docker 볼륨을 지정하는 방법에 대한 자세한 내용은 Amazon ECS 작업 정의에서 Docker 볼륨 지정 섹션을 참조하세요.

Amazon ECS 작업 정의에서 Docker 볼륨 지정

컨테이너가 데이터 볼륨을 사용할 수 있으려면 태스크 정의에서 볼륨 및 탑재 지점 구성을 지정해야 합니다. 이번 섹션에서는 컨테이너의 볼륨 구성에 대해 설명합니다. Docker 볼륨을 사용하는 태스크의 경우 `dockerVolumeConfiguration`을 지정합니다. 바인드 탑재 호스트 볼륨을 사용하는 태스크의 경우, `host`와 `sourcePath`(옵션)를 지정합니다.

다음 태스크 정의 JSON은 컨테이너에 사용할 `volumes` 및 `mountPoints` 객체의 구문을 나타냅니다.

```
{
    "containerDefinitions": [
        {
            "mountPoints": [
                {
                    "sourceVolume": "string",
                    "containerPath": "/path/to/mount_volume",
                    "readOnly": boolean
```

```
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "string",
            "dockerVolumeConfiguration": {
                "scope": "string",
                "autoprovision": boolean,
                "driver": "string",
                "driverOpts": {
                    "key": "value"
                },
                "labels": {
                    "key": "value"
                }
            }
        }
    ]
}
```

name

유형: 문자열

필수 여부: 아니요

볼륨의 이름입니다. 최대 255자의 문자(대문자 및 소문자), 숫자, 하이(-) 및 밑줄(_)이 허용됩니다. 이 이름은 컨테이너 정의 mountPoints 객체의 sourceVolume 파라미터에서 참조됩니다.

dockerVolumeConfiguration

유형: DockerVolumeConfiguration 객체

필수 여부: 아니요

이 파라미터는 Docker 볼륨을 사용할 때 지정됩니다. Docker 볼륨은 EC2 인스턴스에서 작업을 실행하는 경우에만 지원됩니다. Windows 컨테이너는 local 드라이버의 사용만 지원합니다. 바인드 탑재를 사용하려면 대신에 host를 지정하세요.

scope

유형: 문자열

유효한 값: `task` | `shared`

필수 여부: 아니요

수명 주기를 결정하는 Docker 볼륨의 범위입니다. 범위가 `task`로 지정된 Docker 볼륨은 태스크가 시작될 때 자동으로 프로비저닝되고, 태스크가 중단되면 삭제됩니다. 범위가 `shared`로 지정된 Docker 볼륨은 태스크 중단 후에도 유지됩니다.

`autoprovision`

유형: 부울

기본 값: `false`

필수 여부: 아니요

이 값이 `true`인 경우 도커 볼륨이 아직 없으면 도커 볼륨이 생성됩니다. 이 필드는 `scope`가 `shared`인 경우에만 사용됩니다. `scope`가 `task`인 경우 이 파라미터를 생략해야 합니다.

`driver`

유형: 문자열

필수 여부: 아니요

사용할 Docker 볼륨 드라이버입니다. Docker에서 제공하는 드라이버 이름이 작업 배치에 사용되므로 드라이버 값과 이 이름이 일치해야 합니다. Docker 플러그인 CLI를 사용하여 드라이버를 설치했다면 `docker plugin ls`를 사용하여 컨테이너 인스턴스에서 드라이버 이름을 검색합니다. 다른 방법을 사용하여 드라이버를 설치했다면 Docker 플러그인 검색을 사용하여 드라이버 이름을 검색합니다.

`driverOpts`

유형: 문자열

필수 여부: 아니요

전달할 Docker 드라이버에 특정한 옵션 맵입니다. 이 파라미터는 Docker의 볼륨 생성의 `DriverOpts`에 매핑됩니다.

`labels`

유형: 문자열

필수 여부: 아니요

Docker 볼륨에 추가할 사용자 지정 메타데이터입니다.

`mountPoints`

유형: 객체 배열

필수 여부: 아니요

컨테이너에서 데이터 볼륨의 탑재 지점입니다. 이 파라미터는 create-container Docker API의 `Volumes` 및 docker run에 대한 `--volume` 옵션에 매핑됩니다.

Windows 컨테이너는 전체 디렉터리를 동일한 드라이브에 `$env:ProgramData`로 마운트할 수 있습니다. Windows 컨테이너는 디렉터리를 다른 드라이브에 탑재할 수 없으며, 탑재 지점은 여러 드라이브에 걸쳐 사용할 수 없습니다. Amazon EBS 볼륨을 Amazon ECS 작업에 직접 연결하려면 탑재 지점을 지정해야 합니다.

`sourceVolume`

유형: 문자열

필수 항목 여부: 예(`mountPoints` 사용 시)

탑재할 볼륨의 이름입니다.

`containerPath`

유형: 문자열

필수 항목 여부: 예(`mountPoints` 사용 시)

볼륨을 탑재할 컨테이너의 경로입니다.

`readOnly`

유형: 부울

필수 여부: 아니요

이 값이 `true`일 경우 컨테이너에는 볼륨에 대한 읽기 전용 액세스가 부여됩니다. 이 값이 `false`일 경우 컨테이너는 볼륨에 쓸 수 있습니다. 기본값은 `false`입니다.

Windows 운영 체제를 실행하는 EC2 인스턴스에서 실행되는 태스크의 경우 값을 기본값인 `false`로 둡니다.

Amazon ECS용 Docker 볼륨 예제

다음 예제에서는 컨테이너에 임시 스토리지를 제공하는 방법과 여러 컨테이너에 공유 볼륨을 제공하는 방법, 컨테이너에 NFS 영구 스토리지를 제공하는 방법을 보여줍니다.

Docker 볼륨을 사용하여 컨테이너에 임시구 스토리지를 제공하는 방법

이 예에서 컨테이너는 태스크가 완료된 후 폐기되는 빈 데이터 볼륨을 사용합니다. 사용 사례 중 한 예로, 태스크 도중 Scratch 파일 스토리지 위치에 액세스해야 하는 컨테이너가 있을 수 있습니다. Docker 볼륨을 사용하면 이러한 태스크를 수행할 수 있습니다.

1. 태스크 정의 `volumes` 섹션에서 `name` 및 `DockerVolumeConfiguration` 값을 사용하여 데이터 볼륨을 정의합니다. 이 예제에서는 태스크가 중단된 후 볼륨이 삭제되도록 범위를 `task`로 지정하고 기본 제공 `local` 드라이버를 사용합니다.

```
"volumes": [
    {
        "name": "scratch",
        "dockerVolumeConfiguration" : {
            "scope": "task",
            "driver": "local",
            "labels": {
                "scratch": "space"
            }
        }
    }
]
```

2. `containerDefinitions` 섹션에서 정의된 볼륨 이름과 컨테이너에서 볼륨을 탑재할 `mountPoints` 값을 참조하는 `containerPath` 값을 사용하여 컨테이너를 정의합니다.

```
"containerDefinitions": [
    {
        "name": "container-1",
        "mountPoints": [
            {
              "sourceVolume": "scratch",
              "containerPath": "/var/scratch"
            }
        ]
    }
]
```

Docker 볼륨을 사용하여 여러 컨테이너에 영구 스토리지 제공

이번 예제에서는 다수의 컨테이너에서 공유 볼륨을 사용하고, 이 볼륨을 사용하는 단일 태스크가 중단된 후에도 볼륨을 유지하려고 합니다. 내장 `local` 드라이버를 사용 중입니다. 또한 볼륨이 컨테이너 인스턴스의 수명 주기에 계속해서 연결되도록 합니다.

1. 태스크 정의 `volumes` 섹션에서 `name` 및 `DockerVolumeConfiguration` 값을 사용하여 데이터 볼륨을 정의합니다. 이 예제에서는 `shared` 범위를 지정하여 볼륨이 유지되도록 자동 프로비저닝을 `true`로 설정합니다. 이는 사용하기 위해 볼륨이 생성되도록 하기 위함입니다. 그런 다음 내장 `local` 드라이버도 사용합니다.

```
"volumes": [
    {
        "name": "database",
        "dockerVolumeConfiguration" : {
            "scope": "shared",
            "autoprovision": true,
            "driver": "local",
            "labels": {
                "database": "database_name"
            }
        }
    }
]
```

2. `containerDefinitions` 섹션에서 정의된 볼륨 이름과 컨테이너에서 볼륨을 탑재할 `mountPoints` 값을 참조하는 `containerPath` 값을 사용하여 컨테이너를 정의합니다.

```
"containerDefinitions": [
    {
        "name": "container-1",
        "mountPoints": [
        {
          "sourceVolume": "database",
          "containerPath": "/var/database"
        }
      ]
    },
    {
      "name": "container-2",
      "mountPoints": [
        {
```

```
            "sourceVolume": "database",
            "containerPath": "/var/database"
          }
        ]
      }
    ]
```

도커 볼륨을 사용하여 컨테이너에 NFS 영구 스토리지 제공

이 예에서 컨테이너는 작업이 시작될 때 자동으로 탑재되고 작업이 중지될 때 탑재 해제되는 NFS 데이터 볼륨을 사용합니다. 도커 내장`local` 드라이버가 사용됩니다. 사용 사례 중 한 예로, ECS Anywhere 작업에서 해당 스토리지에 액세스해야 하는 경우가 있습니다. NFS 드라이버 옵션이 있는 도커 볼륨을 사용하면 이러한 작업을 수행할 수 있습니다.

1. 태스크 정의 `volumes` 섹션에서 `name` 및 `DockerVolumeConfiguration` 값을 사용하여 데이터 볼륨을 정의합니다. 이 예제에서는 작업이 중지된 후 볼륨이 탑재 해제되도록 `task` 범위를 지정합니다. `local` 드라이버를 사용하고 그에 따라 `type`, `device` 및 `o` 옵션으로 `driverOpts`를 구성합니다. `NFS_SERVER`를 NFS 서버 엔드포인트로 바꿉니다.

   ```
   "volumes": [
          {
              "name": "NFS",
              "dockerVolumeConfiguration" : {
                  "scope": "task",
                  "driver": "local",
                  "driverOpts": {
                      "type": "nfs",
                      "device": "$NFS_SERVER:/mnt/nfs",
                      "o": "addr=$NFS_SERVER"
                  }
              }
          }
     ]
   ```

2. `containerDefinitions` 섹션에서 정의된 볼륨 이름과 컨테이너에서 볼륨을 탑재할 `mountPoints` 값을 참조하는 `containerPath` 값을 사용하여 컨테이너를 정의합니다.

   ```
   "containerDefinitions": [
          {
              "name": "container-1",
              "mountPoints": [
   ```

```
            {
              "sourceVolume": "NFS",
              "containerPath": "/var/nfsmount"
            }
        ]
    }
]
```

Amazon ECS에서 바인드 탑재 사용

바인드 탑재로 호스트의 파일 또는 디렉터리(예: Amazon EC2 인스턴스)가 컨테이너에 탑재됩니다. 바인드 탑재는 Fargate와 Amazon EC2 인스턴스 모두에서 호스팅되는 태스크에 지원됩니다. 바인드 탑재는 이를 사용하는 컨테이너의 수명 주기에 연결됩니다. 바인드 탑재를 사용하는 모든 컨테이너가 중지되면(예: 태스크가 중지될 때) 데이터가 제거됩니다. Amazon EC2 인스턴스에서 호스팅되는 작업의 경우 작업 정의에 `host` 및 `sourcePath` 값(선택 사항)을 지정하여 호스트 Amazon EC2 인스턴스의 수명 주기에 데이터를 연결할 수 있습니다. 자세한 내용은 Docker 설명서에서 바인드 탑재를 참조하세요.

바인드 탑재의 일반 사용 사례는 다음과 같습니다.

- 하나 이상의 컨테이너에 마운트할 빈 데이터 볼륨을 제공합니다.
- 하나 이상의 컨테이너에 호스트 데이터 볼륨을 마운트합니다.
- 동일한 태스크의 다른 컨테이너에 소스 컨테이너의 데이터 볼륨을 공유합니다.
- Dockerfile의 경로와 콘텐츠를 하나 이상의 컨테이너에 노출합니다.

바인드 탑재 사용 시 고려 사항

바인드 탑재를 사용할 때는 다음 사항을 고려해야 합니다.

- 플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0` 이상(Windows)을 사용하여 AWS Fargate에 호스팅되는 작업의 경우 기본적으로 바인드 탑재를 위해 최소 20GiB 이상의 임시 스토리지를 수신합니다. 임시 스토리지의 총량은 작업 정의에 `ephemeralStorage` 파라미터를 지정하여 최대 200GiB까지 늘릴 수 있습니다.
- 태스크가 실행될 때 Dockerfile에서 데이터 볼륨으로 파일을 노출하기 위해 Amazon ECS 데이터 영역은 `VOLUME` 명령을 찾습니다. `VOLUME` 명령에 지정된 절대 경로가 태스크 정의에 지정된 `containerPath`와 동일한 경우 `VOLUME` 명령 경로의 데이터가 데이터 볼륨에 복사됩니다. 다음 Dockerfile 예제에서 `/var/log/exported` 디렉터리에 있는 `examplefile`이라는 파일이 호스트에 기록된 다음 컨테이너 내부에 마운트됩니다.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN mkdir -p /var/log/exported
RUN touch /var/log/exported/examplefile
VOLUME ["/var/log/exported"]
```

기본적으로 볼륨 권한은 0755로 설정되고 소유자는 root로 설정됩니다. 이러한 권한은 Dockerfile에서 사용자 지정할 수 있습니다. 다음 예에서는 디렉터리의 소유자를 node로 정의합니다.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install -y shadow-utils && yum clean all
RUN useradd node
RUN mkdir -p /var/log/exported && chown node:node /var/log/exported
RUN touch /var/log/exported/examplefile
USER node
VOLUME ["/var/log/exported"]
```

- Amazon EC2 인스턴스에서 호스팅되는 태스크의 경우 host 및 sourcePath 값이 지정되지 않으면 Docker 대몬이 바인드 탑재를 관리합니다. 컨테이너가 이 바인드 탑재를 참조하지 않으면, Amazon ECS 컨테이너 에이전트 태스크 정리 서비스가 결국 이를 삭제합니다. 기본적으로 이 작업은 컨테이너가 종료되고 3시간 후에 발생합니다. 하지만 이 기간을 ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION 에이전트 변수로 구성할 수 있습니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성을 참조하세요. 컨테이너의 수명 주기가 끝나더라도 이 데이터를 유지하려면 바인드 탑재에 sourcePath 값을 지정합니다.

Amazon ECS 작업 정의에서 바인드 탑재 지정

Fargate 또는 Amazon EC2 인스턴스에 호스팅되는 Amazon ECS 작업의 경우 다음 작업 정의 JSON 코드 조각은 작업 정의에 대한 volumes, mountPoints, ephemeralStorage 객체 구문을 보여줍니다.

```
{
   "family": "",
   ...
   "containerDefinitions" : [
      {
         "mountPoints" : [
            {
               "containerPath" : "/path/to/mount_volume",
               "sourceVolume" : "string"
```

```
            }
         ],
         "name" : "string"
      }
   ],
   ...
   "volumes" : [
      {
         "name" : "string"
      }
   ],
   "ephemeralStorage": {
   "sizeInGiB": integer
   }
}
```

Amazon EC2 인스턴스에서 호스팅되는 Amazon ECS 태스크의 경우, 태스크 볼륨 세부 정보를 지정할 때 host 파라미터 및 sourcePath를 선택 항목으로 사용할 수 있습니다. 지정된 경우, 바인드 탑재를 컨테이너가 아닌 태스크의 수명주기에 연결합니다.

```
"volumes" : [
    {
        "host" : {
            "sourcePath" : "string"
        },
        "name" : "string"
    }
]
```

다음은 각 태스크 정의 파라미터에 대해 자세한 설명입니다.

name

유형: 문자열

필수 여부: 아니요

볼륨의 이름입니다. 최대 255자의 문자(대문자 및 소문자), 숫자, 하이(-) 및 밑줄(_)이 허용됩니다. 이 이름은 컨테이너 정의의 mountPoints 객체의 sourceVolume 파라미터에서 참조됩니다.

host

필수 여부: 아니요

host 파라미터는 바인드 탑재의 수명 주기를 태스크가 아니라 호스트 Amazon EC2 인스턴스와 연결하는 데 사용합니다. host 파라미터가 비어 있으면 Docker 대몬이 데이터 볼륨의 호스트 경로를 할당하지만 해당 볼륨과 연결된 컨테이너가 실행을 중지한 후 데이터 유지가 보장되지 않습니다.

Windows 컨테이너는 전체 디렉터리를 동일한 드라이브에 `$env:ProgramData`로 마운트할 수 있습니다.

> **Note**
>
> sourcePath 파라미터는 Amazon EC2 인스턴스에 호스팅된 작업을 사용하는 경우에만 지원됩니다.

`sourcePath`

유형: 문자열

필수 여부: 아니요

host 파라미터가 사용되는 경우 sourcePath를 지정하여 컨테이너에 제시되는 호스트 Amazon EC2 인스턴스 상의 경로를 선언합니다. 이 파라미터가 비어 있으면 Docker 대몬이 사용자 대신 호스트 경로를 할당합니다. host 파라미터에 sourcePath 파일 위치가 들어 있으면, 사용자가 수동으로 삭제하지 않는 한 데이터 볼륨이 호스트 Amazon EC2 인스턴스 상에 지정된 위치를 유지합니다. sourcePath 값이 호스트 Amazon EC2 인스턴스에 없을 경우 Docker 대몬이 해당 경로를 생성합니다. 해당 위치가 있을 경우 소스 경로 폴더의 콘텐츠를 내보냅니다.

`mountPoints`

유형: 객체 배열

필수 여부: 아니요

컨테이너에서 데이터 볼륨의 탑재 지점입니다. 이 파라미터는 create-container Docker API의 Volumes 및 docker run에 대한 --volume 옵션에 매핑됩니다.

Windows 컨테이너는 전체 디렉터리를 동일한 드라이브에 `$env:ProgramData`로 마운트할 수 있습니다. Windows 컨테이너는 디렉터리를 다른 드라이브에 탑재할 수 없으며, 탑재 지점은 여러 드라이브에 걸쳐 사용할 수 없습니다. Amazon EBS 볼륨을 Amazon ECS 작업에 직접 연결하려면 탑재 지점을 지정해야 합니다.

`sourceVolume`

유형: 문자열

필수 항목 여부: 예(`mountPoints` 사용 시)

탑재할 볼륨의 이름입니다.

`containerPath`

유형: 문자열

필수 항목 여부: 예(`mountPoints` 사용 시)

볼륨을 탑재할 컨테이너의 경로입니다.

`readOnly`

유형: 부울

필수 여부: 아니요

이 값이 `true`일 경우 컨테이너에는 볼륨에 대한 읽기 전용 액세스가 부여됩니다. 이 값이 `false`일 경우 컨테이너는 볼륨에 쓸 수 있습니다. 기본값은 `false`입니다.

Windows 운영 체제를 실행하는 EC2 인스턴스에서 실행되는 태스크의 경우 값을 기본값인 `false`로 둡니다.

`ephemeralStorage`

유형: 객체

필수 여부: 아니요

태스크에 할당되는 임시 스토리지 용량(GB)입니다. 이 파라미터는 플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0` 이상(Windows)을 사용하는 AWS Fargate에서 호스팅되는 태스크에 대해 제공되는 임시 스토리지 총량을 기본 용량 이상으로 확장할 때 사용합니다.

Copilot CLI, CloudFormation,AWS SDK 또는 CLI를 사용하여 바인드 탑재에 대한 임시 스토리지를 지정할 수 있습니다.

Amazon ECS에 대한 바인드 탑재 예제

다음 예제에서는 컨테이너에 바인드 탑재를 사용하는 일반적인 사용 사례를 다룹니다.

Fargate 태스크를 위한 임시 스토리지 공간의 증가된 양을 할당하는 방법

플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0` 이상(Windows)을 사용하여 Fargate에서 호스팅되는 Amazon ECS 태스크의 경우 태스크의 컨테이너에 사용할 임시 스토리지 공간의 기본 양 이상을 할당할 수 있습니다. 이 예제는 다른 예제에 포함하여 Fargate 태스크에 더 많은 임시 스토리지를 할당할 수 있습니다.

- 태스크 정의에서 `ephemeralStorage` 객체를 정의합니다. `sizeInGiB`는 `21`과 `200` 값 사이의 정수여야 하며 GiB로 표현합니다.

```
"ephemeralStorage": {
    "sizeInGiB": integer
}
```

하나 이상의 컨테이너에 빈 데이터 볼륨을 제공하는 방법

경우에 따라 태스크의 컨테이너에 스크래치 공간을 제공합니다. 예를 들어 태스크 도중 동일한 스크래치 파일 스토리지 위치에 액세스해야 하는 2개의 데이터베이스 컨테이너가 있을 수 있습니다. 이 태스크는 바인드 탑재를 사용하여 수행할 수 있습니다.

1. 태스크 정의 `volumes` 섹션에서 이름이 `database_scratch`인 바인드 탑재를 정의합니다.

```
  "volumes": [
    {
      "name": "database_scratch"
    }
  ]
```

2. `containerDefinitions` 섹션에서 데이터베이스 컨테이너 정의를 생성합니다. 이는 볼륨을 탑재하기 위함입니다.

```
"containerDefinitions": [
    {
      "name": "database1",
      "image": "my-repo/database",
      "cpu": 100,
      "memory": 100,
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "database_scratch",
```

```
          "containerPath": "/var/scratch"
        }
      ]
    },
    {
      "name": "database2",
      "image": "my-repo/database",
      "cpu": 100,
      "memory": 100,
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "database_scratch",
          "containerPath": "/var/scratch"
        }
      ]
    }
  ]
```

Dockerfile의 경로와 콘텐츠를 컨테이너에 노출하는 방법

이 예제에서는 컨테이너 내부에 탑재할 데이터를 쓰는 Dockerfile이 있습니다. 이 예제는 Fargate 또는 Amazon EC2 .인스턴스에서 호스팅되는 태스크에 적용됩니다.

1. Dockerfile을 생성합니다. 다음 예제에서는 퍼블릭 Amazon Linux 2 컨테이너 이미지를 사용하고 `/var/log/exported` 디렉토리에서 컨테이너 내부에 탑재하려는 `examplefile`이라는 파일을 생성합니다. `VOLUME` 명령은 절대 경로를 지정해야 합니다.

   ```
   FROM public.ecr.aws/amazonlinux/amazonlinux:latest
   RUN mkdir -p /var/log/exported
   RUN touch /var/log/exported/examplefile
   VOLUME ["/var/log/exported"]
   ```

   기본적으로 볼륨 권한은 `0755`로 설정되고 소유자는 `root`로 설정됩니다. 이러한 권한은 Dockerfile에서 변경할 수 있습니다. 다음 예에서는 `/var/log/exported` 디렉터리의 소유자를 `node`로 정의합니다.

   ```
   FROM public.ecr.aws/amazonlinux/amazonlinux:latest
   RUN yum install -y shadow-utils && yum clean all
   RUN useradd node
   RUN mkdir -p /var/log/exported && chown node:node /var/log/exported
   ```

```
USER node
RUN touch /var/log/exported/examplefile
VOLUME ["/var/log/exported"]
```

2. 태스크 정의 volumes 섹션에서 이름이 application_logs인 볼륨을 정의합니다.

```
"volumes": [
  {
    "name": "application_logs"
  }
]
```

3. containerDefinitions 섹션에서 애플리케이션 컨테이너 정의를 생성합니다. 이에 따라 스토리지를 탑재합니다. containerPath 값은 Dockerfile의 VOLUME 명령에서 지정한 절대 경로와 일치해야 합니다.

```
"containerDefinitions": [
  {
    "name": "application1",
    "image": "my-repo/application",
    "cpu": 100,
    "memory": 100,
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "application_logs",
        "containerPath": "/var/log/exported"
      }
    ]
  },
  {
    "name": "application2",
    "image": "my-repo/application",
    "cpu": 100,
    "memory": 100,
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "application_logs",
        "containerPath": "/var/log/exported"
      }
    ]
  }
```

```
]
```

호스트 Amazon EC2 인스턴스의 수명 주기에 연결된 컨테이너에 빈 데이터 볼륨을 제공하는 방법

Amazon EC2 인스턴스에서 호스팅되는 태스크의 경우 바인드 탑재를 사용하여 호스트 Amazon EC2 인스턴스의 수명 주기에 데이터를 연결할 수 있습니다. 이 태스크는 `host` 파라미터를 사용하고 `sourcePath` 값을 지정하여 수행할 수 있습니다. `sourcePath`에 있는 모든 파일의 `containerPath` 값이 컨테이너에 표시됩니다. `containerPath` 값으로 기록된 모든 파일은 호스트 Amazon EC2 인스턴스에서 `sourcePath` 값으로 기록됩니다.

> **Important**
>
> Amazon ECS는 Amazon EC2 인스턴스에 스토리지를 동기화하지 않습니다. 영구 스토리지를 사용하는 태스크는 클러스터에서 가용 용량이 있는 어떤 Amazon EC2 인스턴스에도 배치할 수 있습니다. 태스크 중단 및 재시작 후 영구 스토리지가 필요한 경우에는 항상 AWS CLI start-task 명령을 사용하여 태스크 시작 시 동일한 Amazon EC2 인스턴스를 지정합니다. 영구 스토리지용 Amazon EFS 볼륨을 사용할 수도 있습니다. 자세한 정보는 Amazon ECS에서 Amazon EFS 볼륨 사용 섹션을 참조하세요.

1. 태스크 정의 `volumes` 섹션에서 `name` 및 `sourcePath` 값을 사용하여 바인드 탑재를 정의합니다. 다음 예에서 호스트 Amazon EC2 인스턴스에는 컨테이너 내부에 탑재하려는 `/ecs/webdata`의 데이터가 포함됩니다.

   ```
   "volumes": [
     {
       "name": "webdata",
       "host": {
         "sourcePath": "/ecs/webdata"
       }
     }
   ]
   ```

2. `containerDefinitions` 섹션에서 바인드 탑재 이름과 컨테이너에서 바인드 탑재를 탑재할 `mountPoints` 값을 참조하는 `containerPath` 값을 사용하여 컨테이너를 정의합니다.

   ```
   "containerDefinitions": [
     {
       "name": "web",
   ```

```
      "image": "public.ecr.aws/docker/library/nginx:latest",
      "cpu": 99,
      "memory": 100,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80
        }
      ],
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "webdata",
          "containerPath": "/usr/share/nginx/html"
        }
      ]
    }
  ]
```

정의된 볼륨을 여러 위치에 있는 여러 컨테이너에 탑재하는 방법

태스크 정의에서 데이터 볼륨을 정의하여 해당 볼륨을 여러 컨테이너의 여러 위치에 탑재할 수 있습니다. 예를 들어, 호스트 컨테이너의 /data/webroot에 웹 사이트 데이터 폴더가 있습니다. 문서 루트가 서로 다른 두 웹 서버에 해당 데이터 볼륨을 읽기 전용으로 탑재할 수 있습니다.

1. 태스크 정의 volumes 섹션에서 이름 webroot 및 소스 경로 /data/webroot를 사용하여 데이터 볼륨을 정의합니다.

   ```
   "volumes": [
     {
       "name": "webroot",
       "host": {
         "sourcePath": "/data/webroot"
       }
     }
   ]
   ```

2. containerDefinitions 섹션에서 mountPoints 볼륨을 해당 컨테이너의 문서 루트를 가리키는 webroot 값과 연결하는 containerPath 값을 사용하여 각 웹 서버의 컨테이너를 정의합니다.

```
"containerDefinitions": [
  {
    "name": "web-server-1",
    "image": "my-repo/ubuntu-apache",
    "cpu": 100,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webroot",
        "containerPath": "/var/www/html",
        "readOnly": true
      }
    ]
  },
  {
    "name": "web-server-2",
    "image": "my-repo/sles11-apache",
    "cpu": 100,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 8080,
        "hostPort": 8080
      }
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webroot",
        "containerPath": "/srv/www/htdocs",
        "readOnly": true
      }
    ]
  }
]
```

volumesFrom을 사용하여 다른 컨테이너로부터 볼륨을 탑재하는 방법

Amazon EC2 인스턴스에서 호스팅되는 태스크의 경우 컨테이너에 하나 이상의 볼륨을 정의한 후 다른 컨테이너 정의(동일한 태스크 내)의 volumesFrom 파라미터를 사용하여 원래 정의된 탑재 지점에서 sourceContainer로부터 모든 볼륨을 탑재할 수 있습니다. volumesFrom 파라미터는 태스크 정의에 정의된 볼륨과 Dockerfile을 사용하여 이미지에 내장된 볼륨에 적용됩니다.

1. (선택 사항) 이미지에 내장된 볼륨을 공유하려면 Dockerfile의 명령인 VOLUME을 사용합니다. 다음 예제 Dockerfile은 httpd 이미지를 사용한 후 볼륨을 추가하고 Apache 문서 루트에서 dockerfile_volume에 탑재합니다. 이 폴더는 httpd 웹 서버에서 사용하는 폴더입니다.

   ```
   FROM httpd
   VOLUME ["/usr/local/apache2/htdocs/dockerfile_volume"]
   ```

   이 Dockerfile을 사용하여 이미지를 만들어 Docker Hub와 같은 리포지토리에 푸시한 후 태스크 정의에서 사용할 수 있습니다. 다음 단계에서 사용된 my-repo/httpd_dockerfile_volume 이미지 예제는 위의 Dockerfile을 사용하여 구축되었습니다.

2. 컨테이너에 다른 볼륨 및 탑재 지점을 정의하는 태스크 정의를 생성합니다. 이 예제 volumes 섹션에서는 empty라는 빈 볼륨을 생성하며, Docker 대몬이 이 볼륨을 관리합니다. host_etc라는 호스트 볼륨도 정의되어 있습니다. 호스트 컨테이너 인스턴스에서 /etc 폴더를 내보냅니다.

   ```
   {
     "family": "test-volumes-from",
     "volumes": [
       {
         "name": "empty",
         "host": {}
       },
       {
         "name": "host_etc",
         "host": {
           "sourcePath": "/etc"
         }
       }
     ],
   ```

   컨테이너 정의 섹션에서 앞서 정의한 볼륨을 탑재하는 컨테이너를 생성합니다. 이 예제에서 web 컨테이너는 empty 및 host_etc 볼륨을 탑재합니다. 이는 Dockerfile에서 볼륨으로 빌드된 이미지를 사용하는 컨테이너입니다.

```
"containerDefinitions": [
    {
      "name": "web",
      "image": "my-repo/httpd_dockerfile_volume",
      "cpu": 100,
      "memory": 500,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80
        }
      ],
      "mountPoints": [
        {
          "sourceVolume": "empty",
          "containerPath": "/usr/local/apache2/htdocs/empty_volume"
        },
        {
          "sourceVolume": "host_etc",
          "containerPath": "/usr/local/apache2/htdocs/host_etc"
        }
      ],
      "essential": true
    },
```

volumesFrom을 사용하여 web 컨테이너와 연결된 모든 볼륨을 탑재하는 다른 컨테이너를 생성합니다. web 컨테이너에 있는 모든 볼륨도 마찬가지로 busybox 컨테이너에 탑재됩니다. 여기에는 my-repo/httpd_dockerfile_volume 이미지를 빌드하는 데 사용된 Dockerfile에 지정된 볼륨이 포함됩니다.

```
    {
      "name": "busybox",
      "image": "busybox",
      "volumesFrom": [
        {
          "sourceContainer": "web"
        }
      ],
      "cpu": 100,
      "memory": 500,
      "entryPoint": [
```

```
        "sh",
        "-c"
      ],
      "command": [
        "echo $(date) > /usr/local/apache2/htdocs/empty_volume/date && echo $(date)
 > /usr/local/apache2/htdocs/host_etc/date && echo $(date) > /usr/local/apache2/
htdocs/dockerfile_volume/date"
      ],
      "essential": false
    }
  ]
}
```

이 태스크가 실행되면 두 컨테이너가 볼륨을 탑재하며, busybox 컨테이너의 command가 파일에 날짜 및 시간을 씁니다. 이 파일은 각 볼륨 폴더에서 date로 호출됩니다. 그러면 web 컨테이너가 표시하는 웹 사이트에 이 폴더가 표시됩니다.

> **Note**
>
> busybox 컨테이너는 빠른 명령을 실행한 후 종료하므로 컨테이너 정의에서 "essential": false를 설정해야 합니다. 그렇지 않으면 컨테이너 종료 후 전체 태스크가 중지됩니다.

Amazon ECS에서 컨테이너 스왑 메모리 공간 관리

Amazon ECS를 사용하면 컨테이너 수준에서 Linux 기반 Amazon EC2 인스턴스의 스왑 메모리 공간 사용을 제어할 수 있습니다. 컨테이너별 스왑 구성을 사용하면 태스크 정의 내의 각 컨테이너에서 스왑을 사용 설정 또는 사용 중지할 수 있습니다. 이 기능을 사용 설정한 사용자의 경우, 사용되는 최대 스왑 공간을 제한할 수 있습니다. 예를 들어, 대기 시간이 중요한 컨테이너는 스왑을 사용 중지할 수 있습니다. 반면, 일시적인 메모리 요구가 높은 컨테이너는 스왑을 켜서 컨테이너가 로드될 때 메모리 부족 오류가 발생할 가능성을 줄일 수 있습니다.

컨테이너의 스왑 구성은 다음 컨테이너 정의 파라미터로 관리됩니다.

`maxSwap`

컨테이너가 사용할 수 있는 총 스왑 메모리 양(MiB) 이 파라미터는 docker run에 대한 `--memory-swap` 옵션으로 변환되며 컨테이너 메모리의 합계에 `maxSwap` 값을 더한 값이 됩니다.

0의 maxSwap 값이 지정되면 컨테이너는 스왑을 사용하지 않습니다. 허용되는 값은 0 또는 양수입니다. maxSwap 파라미터를 생략하면 컨테이너는 실행 중인 컨테이너 인스턴스에 대한 스왑 구성을 사용합니다. swappiness 매개 변수를 사용하려면 maxSwap 값을 설정해야 합니다.

swappiness

이를 통해 컨테이너의 메모리 스왑 동작을 조정할 수 있습니다. 필요한 경우가 아니면 swappiness 값이 0이 되어 스와핑이 발생하지 않도록 합니다. 100의 swappiness 값은 페이지가 적극적으로 스와핑되도록 합니다. 허용되는 값은 0과 100 사이의 숫자입니다. swappiness 파라미터를 지정하지 않으면 60의 기본값이 사용됩니다. maxSwap 값이 지정되지 않은 경우 이 파라미터는 무시됩니다. 이 파라미터는 docker run에 대한 --memory-swappiness 옵션에 매핑됩니다.

다음 예에서는 JSON 구문이 제공됩니다.

```
"containerDefinitions": [{
        ...
        "linuxParameters": {
            "maxSwap": integer,
            "swappiness": integer
        },
        ...
}]
```

고려 사항

컨테이너별 스왑 구성을 사용하는 경우 다음을 고려하세요.

- 컨테이너를 사용하려면 작업을 호스팅하는 Amazon EC2 인스턴스에서 스왑 공간을 사용 설정하고 할당해야 합니다. Amazon ECS에 최적화된 AMI에는 기본적으로 스왑이 사용 설정되어 있지 않습니다. 이 기능을 사용하려면 인스턴스에서 스왑을 활성화해야합니다. 자세한 내용은 Amazon EC2사용자 안내서의 인스턴스 스토어 스왑 볼륨 또는 Amazon EC2 인스턴스에서 스왑 스페이스로 작동하도록 메모리를 할당하려면 어떻게 해야 하나요?를 참조하세요.
- 스왑 공간 컨테이너 정의 파라미터는 EC2 시작 유형을 지정하는 작업 정의에 대해서만 지원됩니다. 이러한 파라미터는 Fargate의 Amazon ECS 사용 전용 작업 정의에는 지원되지 않습니다.
- 이 기능은 Linux 컨테이너에서만 지원됩니다. 현재 Windows 컨테이너는 지원되지 않습니다.
- 만약maxSwap과 swappiness컨테이너 정의 파라미터가 태스크 정의에서 생략되는 경우, 각 컨테이너는 swappiness의 기본값으로 60을 갖습니다. 또한 총 스왑 사용량은 컨테이너 메모리의 두 배로 제한됩니다.

- Amazon Linux 2023에서 작업을 사용하는 경우에는 `swappiness` 파라미터가 지원되지 않습니다.

Fargate 시작 유형에 대한 Amazon ECS 작업 정의 차이

Fargate를 사용하려면 Fargate 시작 유형을 사용하도록 태스크 정의를 구성해야 합니다. Fargate를 사용할 때 추가 고려 사항이 있습니다.

태스크 정의 파라미터

Fargate 시작 유형을 사용하는 태스크는 사용 가능한 Amazon ECS 태스크 정의 파라미터 중 일부를 지원하지 않을 수 있습니다. 전혀 지원되지 않는 파라미터도 있고 Fargate 작업에 다르게 작동하는 파라미터도 있습니다.

다음 태스크 정의 파라미터는 Fargate 태스크에서 유효하지 않습니다.

- `disableNetworking`
- `dnsSearchDomains`
- `dnsServers`
- `dockerSecurityOptions`
- `extraHosts`
- `gpu`
- `ipcMode`
- `links`
- `placementConstraints`
- `privileged`
- `maxSwap`
- `swappiness`

다음 태스크 정의 파라미터는 Fargate 태스크에서 유효하지만 유의해야 할 제한이 있습니다.

- `linuxParameters` – 컨테이너에 적용되는 Linux 전용 옵션을 지정할 때 `capabilities`에 추가할 수 있는 기능은 `CAP_SYS_PTRACE`뿐입니다. `devices`, `sharedMemorySize` 및 `tmpfs` 파라미터는 지원되지 않습니다. 자세한 내용은 Linux 파라미터 섹션을 참조하세요.

- `volumes` – Fargate 태스크는 바인드 탑재 호스트 볼륨만 지원하므로 `dockerVolumeConfiguration` 파라미터는 지원되지 않습니다. 자세한 내용은 볼륨 섹션을 참조하세요.
- `cpu` - AWS Fargate의 Windows 컨테이너의 경우 값은 1 vCPU보다 작을 수 없습니다.
- `networkConfiguration` - Fargate 태스크에서는 항상 `awsvpc` 네트워크 모드가 사용됩니다.

태스크 정의가 Fargate으로 사용하는 데 유효한지 확인하려면 태스크 정의를 등록할 때 다음을 지정할 수 있습니다.

- AWS Management Console에서 기능 필요 필드에 `FARGATE`를 지정합니다.
- AWS CLI에서 `--requires-compatibilities` 옵션을 지정합니다.
- Amazon ECS API에서 `requiresCompatibilities` 플래그를 지정합니다.

운영 체제 및 아키텍처

AWS Fargate에 대한 태스크 및 컨테이너 정의를 구성할 때 컨테이너가 실행하는 운영 체제를 지정해야 합니다. AWS Fargate에서 지원되는 운영 체제는 다음과 같습니다.

- Amazon Linux 2

> **Note**
>
> Linux 컨테이너는 호스트 운영 체제의 커널 및 커널 구성만 사용합니다. 예를 들어 커널 구성에 `sysctl` 시스템 컨트롤이 포함됩니다. Linux 컨테이너 이미지는 모든 Linux 배포판의 파일 및 프로그램을 포함하는 기본 이미지로 만들 수 있습니다. CPU 아키텍처가 일치하면 모든 운영 체제의 모든 Linux 컨테이너 이미지에서 컨테이너를 실행할 수 있습니다.

- Windows Server 2019 Full
- Windows Server 2019 Core
- Windows Server 2022 Full
- Windows Server 2022 Core

AWS Fargate에서 Windows 컨테이너를 실행하는 경우 X86_64 CPU 아키텍처가 있어야 합니다.

AWS Fargate에서 Linux 컨테이너를 실행할 때 X86_64 CPU 아키텍처 또는 ARM 기반 애플리케이션용 ARM64 아키텍처를 사용할 수 있습니다. 자세한 내용은 the section called “64비트 ARM 워크로드에 대한 작업 정의” 섹션을 참조하세요.

태스크 CPU 및 메모리

AWS Fargate용 Amazon ECS 태스크 정의를 사용하려면 태스크 수준에서 CPU와 메모리를 지정해야 합니다. 대부분의 사용 사례에서는 태스크 레벨에서 이러한 리소스를 지정해야 합니다. 유효한 태스크 레벨 CPU 및 메모리 조합은 아래 표와 같습니다. 작업 정의에서 메모리 값을 문자열(MiB 또는 GB 단위)로 지정할 수 있습니다. 예를 들어 메모리 값을 `3072`(MiB) 또는 `3 GB`(GB)로 지정할 수 있습니다. JSON 파일에서 CPU 값을 문자열(CPU 단위 또는 가상 CPU(vCPU))로 지정할 수 있습니다. 예를 들어 CPU 값을 `1024`(CPU 단위) 또는 `1 vCPU`(vCPU)로 지정할 수 있습니다.

CPU 값	메모리 값	AWS Fargate에 지원되는 운영 체제
256(.25 vCPU)	512MiB, 1GB, 2GB	Linux
512(.5 vCPU)	1GB, 2GB, 3GB, 4GB	Linux
1024(1 vCPU)	2GB, 3GB, 4GB, 5GB, 6GB, 7GB, 8GB	Linux, Windows
2048(2 vCPU)	4~16GB(1GB 증분)	Linux, Windows
4096(4 vCPU)	8~30GB(1GB 증분)	Linux, Windows
8192 (8 vCPU) **Note** 이 옵션은 Linux 플랫폼 `1.4.0` 이상이 필요합니다.	16~60GB(4GB 증분)	Linux
16384 (16vCPU)	32~120GB(8GB 증분)	Linux

CPU 값	메모리 값	AWS Fargate에 지원되는 운영 체제
Note 이 옵션은 Linux 플랫폼 1.4.0 이상이 필요합니다.		

태스크 네트워킹

AWS Fargate용 Amazon ECS 태스크에는 탄력적 네트워크 인터페이스를 각 태스크에 제공하는 awsvpc 네트워크 모드가 필요합니다. 이 네트워크 모드를 사용하여 태스크를 실행하거나 서비스를 생성할 때는 네트워크 인터페이스를 연결할 하나 이상의 서브넷과 네트워크 인터페이스에 적용할 하나 이상의 보안 그룹을 지정해야 합니다.

퍼블릭 서브넷을 사용하는 경우에는 네트워크 인터페이스에 퍼블릭 IP 주소를 제공할지를 결정합니다. 퍼블릭 서브넷의 Fargate 태스크가 컨테이너 이미지를 풀링하려면 인터넷에 대한 경로 또는 인터넷으로 요청을 라우팅할 수 있는 NAT 게이트웨이를 사용하여 태스크의 탄력적 네트워크 인터페이스에 퍼블릭 IP 주소를 할당해야 합니다. 프라이빗 서브넷의 Fargate 태스크가 컨테이너 이미지를 풀링하려면 인터넷으로 요청을 라우팅할 수 있는 NAT 게이트웨이가 서브넷에 있어야 합니다. Amazon ECR에서 컨테이너 이미지를 호스팅하면 인터페이스 VPC 엔드포인트를 사용하도록 Amazon ECR을 구성할 수 있습니다. 이 경우 태스크의 프라이빗 IPv4 주소는 이미지 풀링에 사용됩니다. Amazon ECR 인터페이스 엔드포인트에 대한 자세한 정보는 Amazon Elastic Container Registry 사용자 설명서의 Amazon ECR 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

다음은 Fargate 서비스에 대한 networkConfiguration 섹션 예제입니다.

```
"networkConfiguration": {
   "awsvpcConfiguration": {
      "assignPublicIp": "ENABLED",
      "securityGroups": [ "sg-12345678" ],
      "subnets": [ "subnet-12345678" ]
   }
}
```

태스크 리소스 제한

AWS Fargate의 Linux 컨테이너에 대한 Amazon ECS 태스크 정의는 `ulimits` 파라미터를 지원하여 컨테이너에 대해 설정할 리소스 제한을 정의합니다.

AWS Fargate의 Windows용 Amazon ECS 태스크 정의는 컨테이너에 대해 설정할 리소스 제한을 정의하는 `ulimits` 파라미터를 지원하지 않습니다.

Fargate에 호스팅되는 Amazon ECS 작업은 `nofile` 리소스 제한 파라미터를 제외하고 운영 체제에서 설정한 기본 리소스 제한 값을 사용합니다. `nofile` 리소스 제한은 컨테이너가 사용할 수 있는 열린 파일 수에 대한 제한을 설정합니다. Fargate에서 기본 `nofile` 소프트 제한은 `65535`이고 하드 제한은 `65535`입니다. 두 제한의 값을 모두 최대 `1048576`으로 설정할 수 있습니다.

두 배로 늘어난 사용자 지정 `nofile` 제한을 정의하는 방법을 나타내는 예제 태스크 정의 코드 조각은 다음과 같습니다.

```
"ulimits": [
    {
       "name": "nofile",
       "softLimit": 2048,
       "hardLimit": 8192
    }
]
```

조정할 수 있는 다른 리소스 제한에 대한 자세한 정보는 리소스 제한 섹션을 참조하세요.

로깅

이벤트 로깅

Amazon ECS는 수행한 작업을 EventBridge에 기록합니다. EventBridge용 Amazon ECS 이벤트를 사용하여 Amazon ECS 클러스터, 서비스 및 작업의 현재 상태에 관해 실시간에 가까운 알림을 받을 수 있습니다. 또한 이러한 이벤트에 응답하도록 작업을 자동화할 수 있습니다. 자세한 내용은 EventBridge를 사용하여 Amazon ECS 오류에 대한 응답 자동화 섹션을 참조하세요.

작업 수명 주기 로깅

Fargate에서 실행되는 작업은 타임스탬프를 게시하여 작업 수명 주기의 상태를 통해 작업을 추적합니다. AWS Management Console의 작업 세부 정보 및 AWS CLI 및 SDK의 작업 설명에서 타임스탬프를 확인할 수 있습니다. 예를 들어 타임스탬프를 사용하여 컨테이너 이미지를 다운로드하는 데 소요된 시간을 평가하고 컨테이너 이미지 크기를 최적화할지 아니면 Seekable OCI 인덱스를 사용할지 결정할

수 있습니다. 컨테이너 이미지 사례에 대한 자세한 내용은 Amazon ECS 컨테이너 이미지 모범 사례를 참조하세요.

애플리케이션 로깅

AWS Fargate용 Amazon ECS 태스크 정의는 로그 구성을 위해 `awslogs`, `splunk` 및 `awsfirelens` 로그 드라이버를 지원합니다.

`awslogs` 로그 드라이버는 Amazon CloudWatch Logs로 로그 정보를 보낼 Fargate 태스크를 구성합니다. 다음은 `awslogs` 로그 드라이버가 구성되는 태스크 정의 조각을 보여줍니다.

```
"logConfiguration": {
   "logDriver": "awslogs",
   "options": {
      "awslogs-group" : "/ecs/fargate-task-definition",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "ecs"
   }
}
```

태스크 정의에서 `awslogs` 로그 드라이버를 사용하여 컨테이너 로그를 CloudWatch Logs로 보내는 방법에 대한 자세한 정보는 Amazon ECS 로그를 CloudWatch로 전송 섹션을 참조하세요.

태스크 정의의 `awsfirelens` 로그 드라이버에 대한 자세한 정보는 Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송 섹션을 참조하세요.

태스크 정의에서 `splunk` 로그 드라이버를 사용하는 방법에 대한 자세한 정보는 `splunk` 로그 드라이버 섹션을 참조하세요.

태스크 스토리지

Fargate에서 호스팅되는 Amazon ECS 태스크의 경우 다음과 같은 스토리지 유형이 지원됩니다.

- Amazon EBS 볼륨은 데이터 집약적인 컨테이너화된 워크로드에 대해 비용 효율적이며 내구성이 뛰어난 고성능 블록 스토리지를 제공합니다. 자세한 내용은 Amazon ECS에서 Amazon EBS 볼륨 사용 섹션을 참조하세요.
- 영구 스토리지용 Amazon EFS 볼륨. 자세한 내용은 Amazon ECS에서 Amazon EFS 볼륨 사용 섹션을 참조하세요.
- 휘발성 스토리지용 바인드 마운트. 자세한 내용은 Amazon ECS에서 바인드 탑재 사용 섹션을 참조하세요.

Seekable OCI(SOCI)를 사용하여 컨테이너 이미지 지연 로딩

Linux 플랫폼 `1.4.0` 버전을 사용하는 Fargate 기반 Amazon ECS 작업은 Seekable OCI(SOCI)를 사용하여 작업을 더 빠르게 시작할 수 있습니다. SOCI를 사용하면 컨테이너가 이미지 풀에 몇 초만 소비하고 시작할 수 있으므로, 이미지가 백그라운드에서 다운로드되는 동안 환경 설정 및 애플리케이션 인스턴스화에 필요한 시간을 얻을 수 있습니다. 이를 지연 로딩이라고 합니다. Fargate가 Amazon ECS 작업을 시작하면 Fargate는 작업의 이미지에 대한 SOCI 인덱스가 있는지 자동으로 감지하여 전체 이미지가 다운로드될 때까지 기다리지 않고 컨테이너를 시작합니다.

SOCI 인덱스 없이 실행되는 컨테이너의 경우 컨테이너 이미지가 완전히 다운로드된 이후에 컨테이너가 시작됩니다. 이 동작은 Fargate의 다른 모든 플랫폼 버전과 Amazon EC2 인스턴스의 Amazon ECS 최적화 AMI에서 동일합니다.

Seekable OCI(SOCI)는 컨테이너 이미지를 느리게 로드하여 컨테이너를 더 빠르게 시작할 수 있는 AWS에서 개발된 오픈 소스 기술입니다. SOCI는 기존 컨테이너 이미지 내에 파일의 인덱스(SOCI 인덱스)를 생성하는 방식으로 작동합니다. 이 인덱스를 사용하면 전체 이미지를 다운로드하기 전에 컨테이너 이미지에서 개별 파일을 추출할 수 있으므로 컨테이너를 더 빠르게 시작할 수 있습니다. SOCI 인덱스는 컨테이너 레지스트리 내의 이미지와 동일한 리포지토리에 아티팩트로 저장되어야 합니다. 인덱스는 이미지 내용의 신뢰할 수 있는 출처이므로 신뢰할 수 있는 출처의 SOCI 인덱스만 사용해야 합니다. 자세한 내용은 컨테이너 이미지 지연 로딩을 위한 Seekable OCI 소개를 참조하세요.

SOCI를 사용하려는 고객은 SOCI 인덱스 매니페스트 v2만 사용할 수 있습니다. 이전에 Fargate에서 SOCI를 사용한 기존 고객은 SOCI 인덱스 매니페스트 v1을 계속 사용할 수 있지만, SOCI 인덱스 매니페스트 v2로 마이그레이션하는 것이 좋습니다. SOCI 인덱스 매니페스트 v2는 컨테이너 이미지와 해당 SOCI 인덱스 간에 명시적 관계를 생성하여 일관된 배포를 보장합니다.

고려 사항

Fargate가 SOCI 인덱스를 사용하여 작업에서 컨테이너 이미지를 느리게 로드하도록 하려면 다음 사항을 고려합니다.

- Linux 플랫폼 버전 `1.4.0`에서 실행되는 작업만 SOCI 인덱스를 사용할 수 있습니다. Fargate 기반 Windows 컨테이너를 실행하는 작업은 지원되지 않습니다.
- X86_64 또는 ARM64 CPU 아키텍처에서 실행되는 작업은 지원됩니다.
- 작업 정의의 컨테이너 이미지는 호환 가능한 이미지 레지스트리에 저장되어야 합니다. 호환 가능한 레지스트리 목록은 다음과 같습니다.
  - Amazon ECR 프라이빗 레지스트리
- gzip 압축을 사용하거나 압축되지 않은 컨테이너 이미지만 지원됩니다. zstd 압축을 사용하는 컨테이너 이미지는 지원되지 않습니다.

- SOCI 인덱스 매니페스트 v2의 경우 SOCI 인덱스 매니페스트를 생성하면 SOCI 인덱스에 대한 주석을 추가할 때 컨테이너 이미지 매니페스트가 수정됩니다. 이를 통해 새 컨테이너 이미지 다이제스트가 생성됩니다. 컨테이너 이미지 파일 시스템 계층의 내용은 변경되지 않습니다.
- SOCI 인덱스 매니페스트 v2의 경우 컨테이너 이미지가 컨테이너 이미지 리포지토리에 이미 저장된 경우 SOCI 인덱스가 생성된 후 컨테이너 이미지를 다시 푸시해야 합니다. 컨테이너 이미지를 다시 푸시하면 파일 시스템 계층을 복제하여 스토리지 비용이 증가하지 않고 새 매니페스트 파일만 업로드됩니다.
- 250 MiB 압축된 크기보다 큰 컨테이너 이미지를 사용하여 지연 로딩을 시도하는 것이 좋습니다. 크기가 작은 이미지를 로드하는 데 걸리는 시간이 줄어들 가능성은 거의 없습니다.
- 지연 로딩으로 인해 작업 시작 시간이 달라질 수 있으므로 Elastic Load Balancing의 상태 확인 유예 기간과 같은 다양한 제한 시간을 변경해야 할 수 있습니다.
- 컨테이너 이미지가 지연 로드되지 않도록 하려면 SOCI 인덱스가 연결되지 않은 상태에서 컨테이너 이미지를 다시 푸시해야 합니다.

Seekable OCI 인덱스 생성

컨테이너 이미지 로드를 지연하려는 경우 SOCI 인덱스(메타데이터 파일)를 생성하여 컨테이너 이미지와 함께 컨테이너 이미지 리포지토리에 저장해야 합니다. SOCI 인덱스를 생성하고 푸시하기 위해 GitHub의 오픈 소스 soci-snapshotter CLI 도구를 사용할 수 있습니다. 또는 CloudFormation AWS SOCI Index Builder를 배포할 수 있습니다. 컨테이너 이미지가 Amazon ECR로 푸시될 때 SOCI 인덱스를 자동으로 생성하여 푸시하는 서버리스 솔루션입니다. 솔루션 및 설치 단계에 대한 자세한 내용은 GitHub의 CFN AWS SOCI Index Builder를 참조하세요. CloudFormation AWS SOCI Index Builder는 SOCI 시작을 자동화하는 방법이며, 오픈 소스 soci 도구를 사용하면 인덱스 생성과 관련된 유연성이 향상되고 지속적 통합 및 지속적 배포(CI/CD) 파이프라인에 인덱스 생성을 통합할 수 있습니다.

> **Note**
>
> 이미지에 대한 SOCI 인덱스를 생성하려면 `soci-snapshotter`를 실행 중인 컴퓨터의 containerd 이미지 저장소에 이미지가 있어야 합니다. 이미지가 Docker 이미지 저장소에 있는 경우 이미지를 찾을 수 없습니다.

작업에서 지연 로딩을 사용했는지 확인

SOCI를 사용하여 작업이 지연 로드되었는지 확인하려면 작업 내에서 작업 메타데이터 엔드포인트를 확인합니다. 작업 메타데이터 엔드포인트 버전 4를 쿼리하면 쿼리하는 컨테이너의 기본 경로에 `Snapshotter` 필드가 있습니다. 또한 `/task` 경로에 각 컨테이너에 대한 `Snapshotter` 필드가 있

습니다. 이 필드의 기본값은 overlayfs이며, SOCI를 사용하는 경우 이 필드는 soci로 설정됩니다. 컨테이너 이미지에 SOCI 인덱스 매니페스트 v2가 연결되어 있는지 확인하려면 AWS CLI를 사용하여 Amazon ECR에서 이미지 인덱스를 검색하면 됩니다.

```
IMAGE_REPOSITORY=r
IMAGE_TAG=latest

aws ecr batch-get-image \
    --repository-name=$IMAGE_REPOSITORY \
    --image-ids imageTag=$IMAGE_TAG \
    --query 'images[0].imageManifest' --output text | jq -r '.manifests[] |
 select(.artifactType=="application/vnd.amazon.soci.index.v2+json")'
```

컨테이너 이미지에 SOCI 인덱스 매니페스트 v1이 연결되어 있는지 확인하려면 OCI 참조 API를 사용하면 됩니다.

```
ACCOUNT_ID=111222333444
AWS_REGION=us-east-1
IMAGE_REPOSITORY=nginx-demo
IMAGE_TAG=latest
IMAGE_DIGEST=$(aws ecr describe-images --repository-name $IMAGE_REPOSITORY --image-ids
 imageTag=$IMAGE_TAG --query 'imageDetails[0].imageDigest' --output text)
ECR_PASSWORD=$(aws ecr get-login-password)

curl \
    --silent \
    --user AWS:$ECR_PASSWORD \
    https://$ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/v2/$IMAGE_REPOSITORY/
referrers/$IMAGE_DIGEST?artifactType=application%2Fvnd.amazon.soci.index.v1%2Bjson | jq
 -r '.'
```

Windows를 실행하는 EC2 인스턴스에 대한 Amazon ECS 작업 정의 차이

EC2 Windows 인스턴스에서 실행되는 작업은 사용 가능한 모든 Amazon ECS 작업 정의 파라미터를 지원하지는 않습니다. 전혀 지원되지 않는 파라미터도 있고 일부는 다르게 작동합니다.

다음 작업 정의 파라미터는 Amazon EC2 Windows 작업 정의에서 지원되지 않습니다.

- containerDefinitions

  - disableNetworking
  - dnsServers
  - dnsSearchDomains
  - extraHosts
  - links
  - linuxParameters
  - privileged
  - readonlyRootFilesystem
  - user
  - ulimits
- volumes
  - dockerVolumeConfiguration
- cpu

  Windows 컨테이너에 대해서는 컨테이너 레벨 CPU를 지정할 것을 권장합니다.
- memory

  Windows 컨테이너에 대해서는 컨테이너 레벨 메모리를 지정할 것을 권장합니다.
- proxyConfiguration
- ipcMode
- pidMode
- taskRoleArn

  EC2 Windows 인스턴스의 작업에 대한 IAM 역할에는 추가 구성이 필요하지만 이러한 구성의 대부분은 Linux 컨테이너 인스턴스에서의 작업에 대한 IAM 역할 구성과 비슷합니다. 자세한 내용은 the section called “ Amazon EC2 Windows 인스턴스 추가 구성”을 참조하세요.

콘솔을 사용하여 Amazon ECS 작업 정의 생성

태스크 또는 서비스로 실행하는 애플리케이션을 정의할 수 있도록 태스크 정의를 생성합니다.

외부 시작 유형에 대한 작업 정의를 생성할 때 JSON 편집기를 사용하여 태스크 정의를 생성하고 `requireCapabilities` 파라미터를 `EXTERNAL`로 설정해야 합니다.

콘솔 환경을 사용하거나 JSON 파일을 지정하여 태스크 정의를 생성할 수 있습니다.

JSON 검사기

Amazon ECS 콘솔 JSON 편집기는 JSON 파일에서 다음 사항을 검사합니다.

- 파일은 유효한 JSON 파일입니다.
- 파일에 관련 없는 키는 없습니다.
- 파일에 `familyName` 파라미터가 있습니다.
- `containerDefinitions` 아래 하나 이상의 항목이 있습니다.

AWS CloudFormation 스택

다음 동작은 2023년 1월 12일 이전에 새 Amazon ECS 콘솔에서 생성된 작업 정의에 적용됩니다.

작업 정의를 생성하면 Amazon ECS 콘솔은 이름이 `ECS-Console-V2-TaskDefinition-`으로 시작하는 CloudFormation 스택을 자동으로 생성합니다. AWS CLI 또는 AWS SDK를 사용하여 작업 정의를 등록 취소한 경우 작업 정의 스택을 수동으로 삭제해야 합니다. 자세한 정보는 AWS CloudFormation 사용 설명서의 스택 삭제를 참조하세요.

2023년 1월 12일 이후에 생성된 작업 정의에서는 CloudFormation 스택이 자동으로 생성되지 않습니다.

절차

Amazon ECS console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 작업 정의를 선택합니다.
3. 새 작업 정의 생성 메뉴에서 새 작업 정의 생성을 선택합니다.
4. 태스크 정의 패밀리(Task definition family)에서 태스크 정의에 대해 고유한 이름을 지정합니다.
5. 시작 유형에서 애플리케이션 환경을 선택합니다. 콘솔 기본값은 AWS Fargate(서버리스)입니다. Amazon ECS는 이 값으로 작업 정의 파라미터가 인프라 유형에 유효한지 검증을 수행합니다.
6. 운영 체제/아키텍처(Operating system/Architecture)에서 태스크에 사용할 운영 체제와 CPU 아키텍처를 선택합니다.

64비트 ARM 아키텍처에서 작업을 실행하려면 Linux/ARM64를 선택합니다. 자세한 내용은 the section called “런타임 플랫폼” 섹션을 참조하세요.

Windows 컨테이너에서 AWS Fargate 작업을 실행하려면 지원되는 Windows 운영 체제를 선택합니다. 자세한 내용은 the section called “운영 체제 및 아키텍처” 섹션을 참조하세요.

7. 태스크 크기(Task size)에서 태스크용으로 예약할 CPU 및 메모리 값을 선정합니다. CPU 값은 vCPU로 지정되고 메모리는 GB로 지정됩니다.

   다음 표에서는 Fargate에서 호스팅되는 태스크에 대해 유효한 CPU와 메모리 조합을 보여줍니다.

CPU 값	메모리 값	AWS Fargate에 지원되는 운영 체제
256(.25 vCPU)	512MiB, 1GB, 2GB	Linux
512(.5 vCPU)	1GB, 2GB, 3GB, 4GB	Linux
1024(1 vCPU)	2GB, 3GB, 4GB, 5GB, 6GB, 7GB, 8GB	Linux, Windows
2048(2 vCPU)	4~16GB(1GB 증분)	Linux, Windows
4096(4 vCPU)	8~30GB(1GB 증분)	Linux, Windows
8192 (8 vCPU) **Note** 이 옵션은 Linux 플랫폼 `1.4.0` 이상이 필요합니다.	16~60GB(4GB 증분)	Linux
16384 (16vCPU)	32~120GB(8GB 증분)	Linux

CPU 값	메모리 값	AWS Fargate에 지원되는 운영 체제
Note 이 옵션은 Linux 플랫폼 1.4.0 이상이 필요합니다.		

EC2 또는 외부 시작 유형을 사용하는 태스크의 경우 지원되는 태스크 CPU 값은 128 CPU 유닛(0.125 vCPU)에서 196,608 CPU 유닛(192 vCPU) 사이입니다.

메모리 값을 GB 단위로 지정하려면 값 뒤에 GB를 입력합니다. 예를 들어 메모리 값을 3GB로 설정하려면 3GB를 입력합니다.

> **Note**
>
> Windows 컨테이너에 대해서는 태스크 레벨 CPU와 메모리 파라미터가 무시됩니다.

8. 네트워크 모드(Network mode)에서 사용할 네트워크 모드를 선택합니다. 기본값은 awsvpc 모드입니다. 자세한 내용은 Amazon ECS 작업 배치를 참조하세요.

   브리지를 선택하는 경우 포트 매핑 아래 호스트 포트에 컨테이너용으로 예약할 컨테이너 인스턴스의 포트 번호를 입력합니다.

9. (선택 사항) 작업 역할 섹션을 확장하여 작업에 대한 AWS Identity and Access Management(IAM) 역할을 구성합니다.

   a. 태스크 역할(Task role)에서 태스크에 할당할 IAM 역할을 선택합니다. 작업 IAM 역할은 작업의 컨테이너에서 AWS API 작업을 직접 호출할 수 있는 권한을 제공합니다.

   b. 작업 실행 역할에서 역할을 선택합니다.

      작업 실행 역할을 사용하는 경우에 대한 자세한 정보는 the section called “태스크 실행 IAM 역할”을 참조하세요. 역할이 필요하지 않은 경우 없음을 선택합니다.

10. (선택 사항) 작업 배치 섹션을 확장하여 배치 제약 조건을 추가합니다. 작업 배치 제약 조건으로 기본 제공 또는 사용자 지정 속성을 사용하여 작업 배치에 사용되는 컨테이너 인스턴스를 필터링할 수 있습니다.

11. (선택 사항) 결함 주입 섹션을 확장하여 결함 주입을 활성화합니다. 결함 주입을 사용하면 애플리케이션이 특정 장애 시나리오에 어떻게 대응하는지 테스트할 수 있습니다.
12. 태스크 정의에서 정의할 각 컨테이너에 대해 다음 단계를 수행합니다.

    a. 이름(Name)에 컨테이너 이름을 입력합니다.

    b. 이미지 URI(Image URI)에서 컨테이너를 시작하는 데 사용할 이미지를 지정합니다. Amazon ECR 퍼블릭 갤러리 레지스트리의 이미지는 Amazon ECR 퍼블릭 레지스트리 이름만 사용하여 지정할 수 있습니다. 예를 들어 `public.ecr.aws/ecs/amazon-ecs-agent:latest`가 지정되면 Amazon ECR 퍼블릭 갤러리에 호스팅되는 Amazon Linux 컨테이너가 사용됩니다. 다른 모든 리포지토리의 경우 `repository-url/image:tag` 또는 `repository-url/image@digest` 형식을 사용하여 리포지토리를 지정합니다.

    c. 이미지가 Amazon ECR 외부의 프라이빗 레지스트리에 있는 경우 프라이빗 레지스트리에서 에서 프라이빗 레지스트리 인증을 켭니다. 그런 다음 Secrets Manager ARN 또는 이름에 보안 암호의 Amazon 리소스 이름(ARN)을 입력합니다.

    d. 필수 컨테이너에서 작업 정의에 두 개 이상의 컨테이너가 정의되어 있으면 해당 컨테이너를 필수로 간주할지를 지정할 수 있습니다. 컨테이너가 필수로 표시된 경우 해당 컨테이너가 중지되면 작업도 중지됩니다. 각 태스크 정의에는 하나 이상의 필수 컨테이너가 포함되어야 합니다.

    e. 포트 매핑을 통해 컨테이너는 호스트의 포트에 액세스하여 트래픽을 송수신할 수 있습니다. 포트 매핑(Port mappings)에서 다음 중 하나를 수행합니다.

       - awsvpc 네트워크 모드를 사용하는 경우 컨테이너 포트(Container port)와 프로토콜(Protocol)에서 컨테이너에 사용할 포트 매핑을 선택합니다.
       - bridge 네트워크 모드를 사용하는 경우 컨테이너 포트(Container port)와 프로토콜(Protocol)에서 컨테이너에 사용할 포트 매핑을 선택합니다.

       포트 매핑 추가(Add more port mappings)를 선택하여 추가 컨테이너 포트 매핑을 더 지정합니다.

    f. 컨테이너에 루트 파일 시스템에 대한 읽기 전용 액세스를 부여하려면 읽기 전용 루트 파일 시스템에서 읽기 전용을 선택합니다.

    g. (선택 사항) 리소스 할당 제한의 작업 수준 값과 다른 컨테이너 수준 CPU, GPU 및 메모리 제한을 정의하려면 다음을 수행합니다.

       - CPU에 Amazon ECS 컨테이너 에이전트가 컨테이너에 대해 예약하는 CPU 단위 수를 입력합니다.

- GPU에 컨테이너 인스턴스의 GPU 단위 수를 입력합니다.

  GPU를 지원하는 Amazon EC2 인스턴스에는 GPU당 1개의 GPU 단위가 있습니다. 자세한 내용은 the section called “GPU 워크로드에 대한 작업 정의” 섹션을 참조하세요.
- 메모리 하드 제한에 컨테이너에 표시할 메모리 크기를 GB 단위로 입력합니다. 컨테이너가 하드 제한 을 초과하려 하면 해당 컨테이너는 중지됩니다.
- Docker 20.10.0 이상의 대몬은 컨테이너용으로 최소 6MiB의 메모리를 예약합니다. 그러므로 컨테이너에 대해 6MiB 미만의 메모리를 지정하지 마세요.

  Docker 19.03.13-ce 이하의 대몬은 컨테이너용으로 최소 4MiB의 메모리를 예약합니다. 그러므로 컨테이너에 대해 4MiB 미만의 메모리를 지정하지 마세요.
- 메모리 소프트 제한에 컨테이너용으로 예약할 메모리의 소프트 제한(GB)을 입력합니다.

  시스템 메모리가 경합하는 경우 Docker는 컨테이너 메모리를 이 소프트 제한으로 유지하려고 합니다. 작업 수준 메모리를 지정하지 않은 경우, 메모리 하드 제한 및 메모리 소프트 제한 중 하나 또는 모두에 0이 아닌 정수를 지정해야 합니다. 둘 다 지정하는 경우 메모리 하드 제한은 메모리 소프트 제한보다 커야 합니다.

  이 기능은 Windows 컨테이너에서 지원되지 않습니다.

h. (선택 사항)환경 변수 섹션을 확장하여 컨테이너에 삽입할 환경 변수를 지정합니다. 키-값 페어를 사용하여 개별적으로 환경 변수를 지정하거나 Amazon S3 버킷에서 호스팅되는 환경 변수 파일을 지정하여 배치 단위로 환경 변수를 지정할 수 있습니다. 환경 변수 파일의 형식을 지정하는 방법에 대한 자세한 내용은 개별 환경 변수를 Amazon ECS 컨테이너로 전달 섹션을 참조하세요.

   비밀 스토리지에 대한 환경 변수를 지정할 때 키에 보안 암호 이름을 입력합니다. 그런 다음 ValueFrom에 Systems Manager Parameter Store 보안 암호 또는 Secrets Manager 보안 암호의 전체 ARN을 입력합니다.

i. (선택 사항) 로그 수집 사용(Use log collection) 옵션을 선택하여 로그 구성을 지정합니다. 사용 가능한 로그 드라이버마다 지정할 로그 드라이버 옵션이 있습니다. 기본 옵션에서는 Amazon CloudWatch Logs로 컨테이너 로그를 전송합니다. 다른 로그 드라이버 옵션은 AWS FireLens를 사용하여 구성됩니다. 자세한 내용은 Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송 섹션을 참조하세요.

   다음은 각 컨테이너 로그 대상에 대한 자세한 설명입니다.

- Amazon CloudWatch - CloudWatch Logs로 컨테이너 로그를 전송하도록 작업을 구성합니다. 사용자를 대신하여 CloudWatch 로그 그룹을 생성하는 기본 로그 드라이버 옵션이 제공됩니다. 다른 로그 그룹 이름을 지정하려면 드라이버 옵션 값을 변경합니다.
- Splunk로 로그 내보내기 - 컨테이너 로그를 원격 서비스로 보내는 Splunk 드라이버로 전송하도록 작업을 구성합니다. Splunk 웹 서비스에 대한 URL을 입력해야 합니다. Splunk 토큰은 민감한 데이터로 취급될 수 있으므로 보안 암호 옵션으로 지정됩니다.
- Amazon Data Firehose로 로그 내보내기 - Firehose로 컨테이너 로그를 전송하도록 작업을 구성합니다. Firehose 전송 스트림으로 로그를 전송하는 기본 로그 드라이버 옵션이 제공됩니다. 다른 전송 스트림 이름을 지정하려면 드라이버 옵션 값을 변경합니다.
- Amazon Kinesis Data Streams로 로그 내보내기 - Kinesis Data Streams로 컨테이너 로그를 전송하도록 작업을 구성합니다. Kinesis Data Streams 스트림으로 로그를 전송하는 기본 로그 드라이버 옵션이 제공됩니다. 다른 스트림 이름을 지정하려면 드라이버 옵션 값을 변경합니다.
- Amazon OpenSearch Service로 로그 내보내기 - OpenSearch Service 도메인으로 컨테이너 로그를 전송하도록 작업을 구성합니다. 로그 드라이버 옵션이 제공되어야 합니다.
- Amazon S3로 로그 내보내기 - Amazon S3 버킷으로 컨테이너 로그를 전송하도록 작업을 구성합니다. 기본 로그 드라이버 옵션이 제공되지만 유효한 Amazon S3 버킷 이름을 지정해야 합니다.

j. (선택 사항) 추가 컨테이너 파라미터를 구성합니다.

이 옵션을 구성하는 방법	조치	
재시작 정책 이 옵션은 컨테이너가 종료될 때 컨테이너를 다시 시작하는 재시작 정책을 정의합니다.	재시작 정책을 확장하고 다음 항목을 구성합니다. • 컨테이너에 대한 재시작 정책을 활성화하려면 재시작 정책 활성화를 켜십시오. • 무시된 종료 코드의 경우 쉼표로 구분된 정수 컨테이너 종료 코드 목록을 지정하십시오. 컨테이너가 지정된 종료 코드 중 하나라도 있는 상태로 종료되는 경우 Amazon ECS는 컨테이너 재시작을 시도하지 않습니다. 아무것도 지정하지 않으면 Amazon ECS는 종료 코드를 무시하지 않습니다. • 재설정 시도 기간의 경우 종료 시 재시작을 시도하기 전에 컨테이너를 실행해야 하는 정수 기간(초)을 지정합니다. Amazon ECS는 재설정 시도 기간(초) 마다 한 번만 컨테이너 재시작을 시도할 수 있습니다. 아무것도 지정하지 않은 경우 컨테이너를 300초 동안 실행해야	

이 옵션을 구성하는 방법	조치	
	재시작을 시도할 수 있습니다.	

이 옵션을 구성하는 방법	조치	
HealthCheck 다음은 컨테이너의 정상 상태를 결정하는 명령입니다. 자세한 내용은 컨테이너 상태 확인을 사용하여 Amazon ECS 작업 상태 확인 섹션을 참조하세요.	HealthCheck를 확장한 후 다음 항목을 구성합니다. • Command(명령)에 쉼표로 구분된 명령 목록을 입력합니다. CMD에서 명령을 시작하여 명령 인수를 직접 실행하거나 CMD-SHELL 에서 명령을 시작하여 컨테이너의 기본 쉘에서 명령을 실행할 수 있습니다. 둘 다 지정하지 않으면 CMD가 사용됩니다. • Interval(간격)에 각 상태 확인 사이의 시간(초)을 입력합니다. 유효한 값은 5~30입니다. • Timeout(제한 시간)에 실패로 간주되기 전에 상태 확인이 성공하기까지의 대기 시간(초)입니다. 유효한 값은 2~60입니다. • Start period(시작 기간)에 상태 확인 명령이 실행되기 전에 컨테이너가 부트스트랩하기까지의 대기 시간(초)을 입력합니다. 유효한 값은 0~300입니다. •	

이 옵션을 구성하는 방법	조치	
	Retries(재시도)에 실패 발생 시 상태 확인 명령을 재시도할 횟수를 입력합니다. 유효한 값은 1~10입니다.	
시작 종속성 순서 이 옵션은 컨테이너 시작 및 종료에 대한 종속성을 정의합니다. 컨테이너는 여러 종속성을 포함할 수 있습니다.	시작 종속성 순서를 확장한 후 다음을 구성합니다. a. 컨테이너 종속성 추가를 선택합니다. b. 컨테이너에서 컨테이너를 선택합니다. c. 조건에서 시작 종속성 조건을 선택합니다. 종속성을 더 추가하려면 컨테이너 종속성 추가를 선택합니다.	

이 옵션을 구성하는 방법	조치	
컨테이너 제한 시간 이 옵션은 컨테이너 시작 및 중지 시기를 결정합니다.	컨테이너 시간 제한을 확장한 후 다음을 구성합니다. • 컨테이너에 대한 종속성 해결을 포기하기 전에 대기할 시간을 구성하려면 시작 제한 시간에 시간(초)을 입력합니다. • 컨테이너가 자체적으로 정상 종료하지 않을 경우 중지될 때까지 대기할 시간을 구성하려면 중지 제한 시간에 시간(초)을 입력합니다.	

이 옵션을 구성하는 방법	조치	
컨테이너 네트워크 설정 이 옵션은 컨테이너 내에서 네트워킹을 사용할지 여부를 결정합니다.	컨테이너 네트워크 설정을 확장한 후 다음을 구성합니다. • 컨테이너 네트워킹을 비활성화하려면 네트워킹 끄기를 선택합니다. • 컨테이너에 표시되는 DNS 서버 IP 주소를 구성하려면 DNS 서버에서 각 서버의 IP 주소를 별도의 줄에 입력합니다. • 컨테이너에 제공된 정규화되지 않은 호스트 이름을 검색하도록 DNS 도메인을 구성하려면 DNS 검색 도메인에서 각 도메인을 별도의 줄에 입력합니다. 패턴은 **^[a-zA-Z0-9-.]{0,253}[a-zA-Z0-9]$** 입니다. • 컨테이너 호스트 이름을 구성하려면 호스트 이름에 컨테이너 goat 이름을 입력합니다. • 컨테이너의 `/etc/hosts` 파일에 추가된 호스트 이름 및 IP 주소 매핑을 추가하려면 호스트 추	

이 옵션을 구성하는 방법	조치	
	가를 선택한 다음 호스트 이름 및 IP 주소에 호스트 이름과 IP 주소를 입력합니다.	

이 옵션을 구성하는 방법	조치	
Docker 구성 이는 Dockerfile의 값을 재정의합니다.	Docker 구성을 확장하고 다음 항목을 구성합니다. • 명령에 컨테이너에 대한 실행 가능한 명령을 입력합니다. 이 파라미터는 Docker 원격 API의 컨테이너 생성 섹션에 있는 Cmd에 매핑되며 COMMAND 옵션은 docker run 에 매핑됩니다. 이 파라미터는 Dockerfile의 CMD 명령을 재정의합니다. • 진입점에 컨테이너로 전달되는 Docker 진입점을 입력합니다. 이 파라미터는 Docker 원격 API의 컨테이너 생성 섹션에 있는 Entrypoint 에 매핑되며 --entrypoint 옵션은 docker run에 매핑됩니다. 이 파라미터는 Dockerfile의 ENTRYPOINT 명령을 재정의합니다. • Working Directory(작업 디렉터리)에 컨테이너가 실행할 디렉터리와 제공	

이 옵션을 구성하는 방법	조치	
	된 진입점 및 명령 지침을 입력합니다. 이 파라미터는 Docker 원격 API의 컨테이너 생성 섹션에 있는 `WorkingDir` 에 매핑되며 `--workdir` 옵션은 `docker run`에 매핑됩니다. 이 파라미터는 Dockerfile의 `WORKDIR` 명령을 재정의합니다.	
리소스 제한 (Ulimits) 이 값은 운영 체제에 설정된 기본 리소스 할당량 설정을 재정의합니다. 이 파라미터는 Docker 원격 API(Docker Remote API)의 컨테이너 생성(Create a container) 섹션에 있는 `Ulimits`(와)과 docker run에 대한 `--ulimit` 옵션에 매핑됩니다.	리소스 제한(ulimits)을 확장한 하고 ulimit 추가를 선택합니다. 제한 이름에서 제한을 선택합니다. 그런 다음 소프트 제한 및 하드 제한에 값을 입력합니다. ulimits를 더 추가하려면 ulimit 추가를 선택합니다.	

이 옵션을 구성하는 방법	조치	
Docker 레이블 이 옵션은 컨테이너에 메타데이터를 추가합니다. 이 파라미터는 Docker 원격 API(Docker Remote API)의 컨테이너 생성(Create a container) 섹션에 있는 `Labels`(와)과 docker run에 대한 `--label` 옵션에 매핑됩니다.	Docker 레이블을 확장하고 키 값 쌍 추가를 선택한 다음, 키 및 값을 입력합니다. Docker 레이블을 더 추가하려면 키 값 쌍 추가를 선택합니다.	

k. (선택 사항) 컨테이너 추가(Add more containers)를 선택하여 컨테이너를 태스크 정의에 더 추가합니다.

13. (선택 사항) 스토리지(Storage) 섹션은 Fargate에 호스팅되는 작업에 대한 임시 스토리지 크기를 확장하는 데 사용됩니다. 이 섹션을 사용하여 작업에 대한 데이터 볼륨 구성을 추가할 수도 있습니다.

    - Fargate 태스크에대한 20 gibibytes (GiB)의 기본값인 이상으로 사용 가능한 임시 스토리지를 확장하려면 양(Amount)에서 값을 최대 200 GiB으로 지정합니다.

14. (선택 사항) 작업 정의에 대한 데이터 볼륨 구성을 추가하려면 볼륨 추가를 선택하고 다음 단계를 수행합니다.

    a. 볼륨 이름(Volume name)에서 데이터 볼륨의 이름을 입력합니다. 데이터 볼륨 이름은 컨테이너 탑재 지점을 생성할 때 사용됩니다.

    b. 볼륨 구성의 경우 작업 정의를 생성할 때 볼륨을 구성할지 또는 배포 중에 볼륨을 구성할지 여부를 선택합니다.

> **Note**
>
> 작업 정의를 생성할 때 구성할 수 있는 볼륨으로, 바인드 탑재, Docker, Amazon EFS 및 Amazon FSx for Windows File Server가 있습니다. 배포 시, 작업을 실행할 때 또는 서비스를 생성하거나 업데이트할 때 구성할 수 있는 볼륨으로, Amazon EBS가 있습니다.

c. 볼륨 유형에서 선택한 구성 유형과 호환되는 볼륨 유형을 선택하고 볼륨 유형을 구성합니다.

볼륨 유형	단계	
바인드 탑재	a. Add mount point(탑재 지점 추가)를 선택하고 다음을 구성합니다. • 컨테이너(Container)에서 탑재 지점의 컨테이너를 선택합니다. • 소스 볼륨(Source volume)에서 컨테이너에 탑재할 데이터 볼륨을 선택합니다. • 컨테이너 경로(Container path)에 볼륨을 탑재할 컨테이너 경로를 입력합니다. • 읽기 전용에서 컨테이너에 볼륨에 대한 읽기 전용 액세스가 부여되는지 선택합니다. b. 탑재 지점을 추가하려면 Add mount point(탑재 지점 추가)를 선택합니다.	

볼륨 유형	단계	
EFS	a. File system ID(파일 시스템 ID)에서 Amazon EFS 파일 시스템 ID를 선택합니다. b. (선택 사항) Root directory(루트 디렉터리)에 호스트 내의 루트 디렉터리로 탑재할 Amazon EFS 파일 시스템 내 디렉터리입니다. 이 파라미터가 생략되면 Amazon EFS 볼륨의 루트가 사용됩니다. EFS 액세스 포인트를 사용할 계획이면 이 필드를 비워 둡니다. c. (선택 사항) Access point(액세스 포인트)에서 사용할 액세스 포인트 ID를 선택합니다. d. (선택 사항) Amazon EFS 파일 시스템과 Amazon ECS 호스트 간의 데이터를 암호화하거나 볼륨을 탑재할 때 작업 실행 역할을 사용하려면 Advanced configurations(고급 구성)를 선택하고 다음을 구성합니다. •	

볼륨 유형	단계	
	Amazon EFS 파일 시스템과 Amazon ECS 호스트 간에 데이터를 암호화하려면 Transit encryption(전송 암호화)을 선택한 후 Port(포트)에 Amazon ECS 호스트와 Amazon EFS 서버 간에 암호화된 데이터를 전송할 때 사용할 포트를 입력합니다. 전송 중 데이터 암호화 포트를 지정하지 않으면 Amazon EFS 탑재 헬퍼가 사용하는 포트 선택 전략이 사용됩니다. 자세한 정보는 Amazon Elastic File System 사용 설명서의 EFS 탑재 헬퍼를 참조하세요. • Amazon EFS 파일 시스템을 탑재할 때 작업 정의에 정의된 Amazon ECS 작업 IAM 역할을 사용하려면 IAM authorization(IAM 권한 부여)을 선택합니다. e. Add mount point(탑재 지점 추가)를 선택하고 다음을 구성합니다. •	

볼륨 유형	단계	
	컨테이너(Container)에서 탑재 지점의 컨테이너를 선택합니다. • 소스 볼륨(Source volume)에서 컨테이너에 탑재할 데이터 볼륨을 선택합니다. • 컨테이너 경로(Container path)에 볼륨을 탑재할 컨테이너 경로를 입력합니다. • 읽기 전용에서 컨테이너에 볼륨에 대한 읽기 전용 액세스가 부여되는지 선택합니다. f. 탑재 지점을 추가하려면 Add mount point(탑재 지점 추가)를 선택합니다.	

볼륨 유형	단계	
Docker	a. 드라이버에 Docker 볼륨 구성을 입력합니다. Windows 컨테이너는 로컬 드라이버의 사용만 지원합니다. 바인드 탑재를 사용하려면 host를 지정합니다. b. Scope(범위)에서 볼륨 수명 주기를 선택합니다. • 작업이 시작되고 중지될 때 수명 주기가 지속되도록 하려면 Task(작업)를 선택합니다. • 작업이 중지된 후에도 볼륨이 유지되게 하려면 Shared(공유)를 선택합니다. c. Add mount point(탑재 지점 추가)를 선택하고 다음을 구성합니다. • 컨테이너(Container)에서 탑재 지점의 컨테이너를 선택합니다. • 소스 볼륨(Source volume)에서 컨테이너에 탑재할 데이터 볼륨을 선택합니다. •	

볼륨 유형	단계	
	컨테이너 경로(Container path)에 볼륨을 탑재할 컨테이너 경로를 입력합니다. • 읽기 전용에서 컨테이너에 볼륨에 대한 읽기 전용 액세스가 부여되는지 선택합니다. d. 탑재 지점을 추가하려면 Add mount point(탑재 지점 추가)를 선택합니다.	

볼륨 유형	단계	
FSx for Windows File Server	a. 파일 시스템 ID에서 FSx for Windows File Server 파일 시스템 ID를 선택합니다. b. 루트 디렉터리에 호스트 내 루트 디렉터리로 탑재할 FSx for Windows File Server 파일 시스템 내 디렉터리를 입력합니다. c. Credential parameter(보안 인증 파라미터)에서 자격 증명 저장 방법을 선택합니다. • AWS Secrets Manager를 사용하려면 Secrets Manager 보안 암호의 Amazon 리소스 이름(ARN)을 입력합니다. • AWS Systems Manager를 사용하려면 Systems Manager 파라미터의 Amazon 리소스 이름(ARN)을 입력합니다. d. 도메인에 셀프 호스팅된 EC2 Active Directory 또는 AWS Directory Service for Microsoft Active Directory(AWS	

볼륨 유형	단계	
	Managed Microsoft AD) 디렉터리에서 호스팅하는 정규화된 도메인 이름을 입력합니다. e. Add mount point(탑재 지점 추가)를 선택하고 다음을 구성합니다. • 컨테이너(Container)에서 탑재 지점의 컨테이너를 선택합니다. • 소스 볼륨(Source volume)에서 컨테이너에 탑재할 데이터 볼륨을 선택합니다. • 컨테이너 경로(Container path)에 볼륨을 탑재할 컨테이너 경로를 입력합니다. • 읽기 전용에서 컨테이너에 볼륨에 대한 읽기 전용 액세스가 부여되는지 선택합니다. f. 탑재 지점을 추가하려면 Add mount point(탑재 지점 추가)를 선택합니다.	

볼륨 유형	단계	
Amazon EBS	a. Add mount point(탑재 지점 추가)를 선택하고 다음을 구성합니다. • 컨테이너(Container)에서 탑재 지점의 컨테이너를 선택합니다. • 소스 볼륨(Source volume)에서 컨테이너에 탑재할 데이터 볼륨을 선택합니다. • 컨테이너 경로(Container path)에 볼륨을 탑재할 컨테이너 경로를 입력합니다. • 읽기 전용에서 컨테이너에 볼륨에 대한 읽기 전용 액세스가 부여되는지 선택합니다. b. 탑재 지점을 추가하려면 Add mount point(탑재 지점 추가)를 선택합니다.	

15. 다른 컨테이너의 볼륨을 추가하려면 다음 위치에서 볼륨 추가를 선택하고 다음을 구성합니다.
    - 컨테이너에서 컨테이너를 선택합니다.
    - 소스에서 마운트하려는 볼륨이 있는 컨테이너를 선택합니다.
    - 읽기 전용에서 컨테이너에 볼륨에 대한 읽기 전용 액세스가 부여되는지 선택합니다.

16. (선택 사항) AWS Distro for OpenTelemetry 통합을 사용하여 애플리케이션 추적 및 지표 수집 설정을 구성하려면 모니터링을 확장하고 지표 수집 사용을 선택하여 작업에 대한 지표를 수집하고 Amazon CloudWatch 또는 Amazon Managed Service for Prometheus로 전송합니다. 이 옵션을 선택하면 Amazon ECS는 애플리케이션 지표를 전송하도록 미리 구성된 AWS Distro for OpenTelemetry 컨테이너 사이드카를 생성합니다. 자세한 내용은 애플리케이션 지표를 사용하여 Amazon ECS 애플리케이션 성능 상호 연관 섹션을 참조하세요.

    a. Amazon CloudWatch를 선택하면 사용자 정의 애플리케이션 지표가 CloudWatch에 사용자 정의 지표로 라우팅됩니다. 자세한 정보는 Amazon CloudWatch로 애플리케이션 지표 내보내기을 참조하세요.

    > **Important**
    >
    > Amazon CloudWatch로 애플리케이션 지표를 내보낼 때 필요한 권한이 있는 태스크 IAM 역할이 태스크 정의에 필요합니다. 자세한 정보는 AWS Distro for OpenTelemetry와 Amazon CloudWatch 통합에 필요한 IAM 권한을 참조하세요.

    b. Amazon Managed Service for Prometheus(Prometheus 라이브러리 계측)(Amazon Managed Service for Prometheus (Prometheus libraries instrumentation))를 선택하면 태스크 수준 CPU, 메모리, 네트워크 및 스토리지 지표와 사용자 정의 애플리케이션 지표가 Amazon Managed Service for Prometheus로 라우팅됩니다. 작업 공간 원격 쓰기 엔드포인트에서 Prometheus 작업 영역에 대한 원격 쓰기 엔드포인트 URL을 입력합니다. 스크레이핑 대상에서 AWS Distro for OpenTelemetry 수집기가 지표 데이터를 스크레이핑하는 데 사용할 수 있는 호스트와 포트를 입력합니다. 자세한 내용은 Amazon Managed Service for Prometheus로 애플리케이션 지표 내보내기 섹션을 참조하세요.

    > **Important**
    >
    > Amazon Managed Service for Prometheus로 애플리케이션 지표를 내보낼 때 필요한 권한이 있는 태스크 IAM 역할이 태스크 정의에 필요합니다. 자세한 내용은 AWS Distro for OpenTelemetry와 Amazon Managed Service for Prometheus 통합에 필요한 IAM 권한 섹션을 참조하세요.

    c. Amazon Managed Service for Prometheus(OpenTelemetry 계측)를 선택하면 작업 수준 CPU, 메모리, 네트워크 및 스토리지 지표와 사용자 정의 애플리케이션 지표가 Amazon Managed Service for Prometheus로 라우팅됩니다. 작업 공간 원격 쓰기 엔드포인트에서 Prometheus 작업 영역에 대한 원격 쓰기 엔드포인트 URL을 입력합니다. 자세한 내용은

Amazon Managed Service for Prometheus로 애플리케이션 지표 내보내기 섹션을 참조하세요.

> **Important**
>
> Amazon Managed Service for Prometheus로 애플리케이션 지표를 내보낼 때 필요한 권한이 있는 태스크 IAM 역할이 태스크 정의에 필요합니다. 자세한 정보는 AWS Distro for OpenTelemetry와 Amazon Managed Service for Prometheus 통합에 필요한 IAM 권한을 참조하세요.

17. (선택 사항) 태그(Tags) 섹션을 확장하여 태그를 키-값 페어로 태스크 정의에 추가합니다.
    - [태그 추가] 태그 추가를 선택하고 다음을 수행합니다.
      - 키에서 키 이름을 입력합니다.
      - 값에 키 값을 입력합니다.
    - [태그 제거] 태그 옆에 있는 태그 제거를 선택합니다.
18. 생성을 선택하여 작업 정의를 등록합니다.

Amazon ECS console JSON editor

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 작업 정의를 선택합니다.
3. 새 작업 정의 생성 메뉴에서 JSON으로 새 작업 정의 생성을 선택합니다.
4. JSON 편집기 상자에서 JSON 파일을 편집합니다.

   JSON은 에 지정된 유효성 검사를 통과해야the section called “JSON 검사기” 합니다.
5. 생성(Create)을 선택합니다.

콘솔을 사용하여 Amazon ECS 작업 정의 업데이트

태스크 정의 개정은 새 파라미터 값이 기존 파라미터 값을 대체하는 현재 태스크 정의의 복사본입니다. 수정하지 않는 모든 파라미터는 새 개정에 있습니다.

태스크 정의를 업데이트하려면 태스크 정의 개정을 생성합니다. 서비스에서 태스크 정의가 사용되는 경우 업데이트된 태스크 정의를 사용하도록 서비스를 업데이트해야 합니다.

개정을 생성할 때 다음 컨테이너 속성과 환경 속성을 수정할 수 있습니다.

- 컨테이너 이미지 URI
- 포트 매핑
- 환경 변수
- 태스크 크기
- 컨테이너 크기
- 태스크 역할
- 태스크 실행 역할
- 볼륨 및 컨테이너 탑재 포인트
- 프라이빗 레지스트리

JSON 검사기

Amazon ECS 콘솔 JSON 편집기는 JSON 파일에서 다음 사항을 검사합니다.

- 파일은 유효한 JSON 파일입니다.
- 파일에 관련 없는 키가 없습니다.
- 파일에 `familyName` 파라미터가 있습니다.
- `containerDefinitions`에 하나 이상의 항목이 있습니다.

절차

Amazon ECS console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 태스크 정의가 들어 있는 리전을 선택합니다.
3. 탐색 창에서 태스크 정의를 선택합니다.
4. 작업 정의를 선택합니다.
5. 태스크 정의 개정을 선택한 다음 새 개정 생성, 새 개정 생성을 선택합니다.
6. 새 태스크 정의 개정 생성(Create new task definition revision) 페이지에서 변경합니다. 예를 들어 기존 컨테이너 정의(컨테이너 이미지, 메모리 한계 또는 포트 매핑 등)를 변경하려면 컨테이너를 선택하고 변경합니다.
7. 정보를 확인하고 업데이트를 선택합니다.

8. 만약 태스크 정의가 서비스에서 사용되는 경우, 업데이트된 태스크 정의 서비스를 업데이트합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.

Amazon ECS console JSON editor

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 작업 정의를 선택합니다.
3. 새 개정 생성(Create new revision), JSON으로 새 개정 생성(Create new revision with JSON)을 선택합니다.
4. JSON 편집기 상자에서 JSON 파일을 편집합니다.

   JSON은 에 지정된 유효성 검사를 통과해야the section called “JSON 검사기” 합니다.
5. 생성(Create)을 선택합니다.

콘솔을 사용하여 Amazon ECS 작업 정의 개정 등록 취소

작업을 실행하거나 서비스를 업데이트할 때 `ListTaskDefinition` API 호출 또는 콘솔에 더 이상 표시되지 않도록 태스크 정의 개정을 등록 취소할 수 있습니다.

태스크 정의 개정을 등록 취소하면 즉시 `INACTIVE`로 표시됩니다. `INACTIVE` 태스크 정의 개정을 참조하는 기존 태스크 및 서비스는 중단 없이 계속 실행됩니다. `INACTIVE` 태스크 정의 개정을 참조하는 기존 서비스는 서비스의 원하는 개수를 수정하여 확장하거나 수정할 수 있습니다.

`INACTIVE` 태스크 정의 개정을 사용하여 새 태스크를 실행하거나 새 서비스를 생성할 수 없습니다. 또한 `INACTIVE` 태스크 정의 개정을 참조하도록 기존 서비스를 업데이트할 수 없습니다(이러한 제한이 적용되기까지 등록 취소 후 최대 10분의 유예 기간이 있을 수 있음).

> **Note**
>
> 작업 패밀리에서 모든 개정의 등록을 취소하면 작업 정의 패밀리가 `INACTIVE` 목록으로 이동합니다. `INACTIVE` 태스크 정의의 새로운 개정을 추가하면 태스크 정의 패밀리가 `ACTIVE` 리스트로 다시 돌아갑니다.
>
> 현재, `INACTIVE` 태스크 정의 개정은 계정에서 무기한 검색 가능한 상태로 유지됩니다. 그러나 이 동작은 향후 변경될 수 있습니다. 따라서 관련 태스크 및 서비스의 수명 주기 이상으로 지속되는 `INACTIVE` 태스크 정의 수정 버전에 의존해서는 안 됩니다.

AWS CloudFormation 스택

다음 동작은 2023년 1월 12일 이전에 새 Amazon ECS 콘솔에서 생성된 작업 정의에 적용됩니다.

작업 정의를 생성하면 Amazon ECS 콘솔은 이름이 `ECS-Console-V2-TaskDefinition-`으로 시작하는 CloudFormation 스택을 자동으로 생성합니다. AWS CLI 또는 AWS SDK를 사용하여 작업 정의를 등록 취소한 경우 작업 정의 스택을 수동으로 삭제해야 합니다. 자세한 정보는 AWS CloudFormation 사용 설명서의 스택 삭제를 참조하세요.

2023년 1월 12일 이후에 생성된 작업 정의에서는 CloudFormation 스택이 자동으로 생성되지 않습니다.

절차

새 작업 정의 등록 취소 방법(Amazon ECS 콘솔)

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 태스크 정의가 들어 있는 리전을 선택합니다.
3. 탐색 창에서 태스크 정의를 선택합니다.
4. Task definitions(작업 정의) 페이지에서 등록 취소하려는 하나 이상의 개정을 포함하는 작업 정의 패밀리를 선택합니다.
5. 작업 정의 이름 페이지에서 삭제할 수정 사항을 선택한 다음 작업, 등록 취소를 선택합니다.
6. 등록 취소(Deregister) 창에서 정보를 확인하고 등록 취소(Deregister)를 선택하여 완료합니다.

콘솔을 사용하여 Amazon ECS 작업 정의 개정 삭제

Amazon ECS에서 특정 작업 정의 개정이 더 이상 필요하지 않은 경우 작업 정의 개정을 삭제할 수 있습니다.

작업 정의 개정판을 삭제하면 즉시 `INACTIVE`에서 `DELETE_IN_PROGRESS`로 전환됩니다. `DELETE_IN_PROGRESS` 작업 정의 개정판을 참조하는 기존 작업 및 서비스는 중단 없이 계속 실행됩니다.

`DELETE_IN_PROGRESS` 작업 정의 개정판을 사용하여 새 작업을 실행하거나 새 서비스를 생성할 수 없습니다. 또한 `DELETE_IN_PROGRESS` 작업 정의 개정판을 참조하도록 기존 서비스를 업데이트할 수 없습니다.

모든 INACTIVE 작업 정의 개정을 삭제하면 작업 정의 이름이 콘솔에 표시되지 않고 API에도 반환되지 않습니다. 작업 정의 개정이 DELETE_IN_PROGRESS 상태인 경우 작업 정의 이름이 콘솔에 표시되고 API에서 반환됩니다. 작업 정의 이름은 Amazon ECS에서 유지되며 다음에 해당 이름으로 작업 정의를 생성하면 개정 번호가 증가합니다.

삭제를 차단할 수 있는 Amazon ECS 리소스

작업 정의 개정을 사용하는 Amazon ECS 리소스가 있는 경우 작업 정의 삭제 요청이 완료되지 않습니다. 다음 리소스는 작업 정의가 삭제되지 않도록 할 수 있습니다.

- Amazon ECS 단독 작업 - 작업을 정상으로 유지하려면 작업 정의가 필요합니다.
- Amazon ECS 서비스 작업 - 작업을 정상으로 유지하려면 작업 정의가 필요합니다.
- Amazon ECS 서비스 배포 및 작업 세트 - Amazon ECS 배포 또는 작업 세트에 대한 조정 이벤트가 시작될 때 작업 정의가 필요합니다.

작업 정의가 DELETE_IN_PROGRESS 상태를 유지하는 경우 콘솔 또는 AWS CLI를 사용하여 작업 정의 삭제를 차단하는 리소스를 식별한 다음 중지할 수 있습니다.

차단된 리소스가 제거된 후 작업 정의 삭제

작업 정의 삭제를 차단하는 리소스를 제거하면 다음 규칙이 적용됩니다.

- Amazon ECS 작업 - 작업이 중지된 후 작업 정의 삭제를 완료하는 데 최대 1시간이 걸릴 수 있습니다.
- Amazon ECS 서비스 배포 및 작업 세트 - 배포 또는 작업 세트를 삭제한 후 작업 정의 삭제를 완료하는 데 최대 24시간이 걸릴 수 있습니다.

절차

작업 정의를 등록 취소하는 방법(Amazon ECS 콘솔).

작업 정의 개정판을 삭제하기 전에 작업 정의 개정판을 등록 취소해야 합니다. 자세한 내용은 the section called “콘솔을 사용하여 작업 정의 개정판 등록 취소” 섹션을 참조하세요.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 태스크 정의가 들어 있는 리전을 선택합니다.
3. 탐색 창에서 태스크 정의를 선택합니다.

4. 작업 정의 페이지에서 삭제하려는 하나 이상의 개정판을 포함하는 작업 정의 패밀리를 선택합니다.
5. 작업 정의 이름 페이지에서 삭제할 개정을 선택하고 작업, 삭제를 선택합니다.

   삭제를 사용할 수 없는 경우 작업 정의 등록을 취소해야 합니다.
6. 삭제 확인 창에서 정보를 확인하고 삭제를 선택하여 완료합니다.

Amazon ECS 작업 정의 사용 사례

다양한 AWS 서비스 및 기능에 대한 작업 정의를 작성하는 방법에 대해 자세히 알아봅니다.

워크로드에 따라 설정해야 하는 특정 작업 정의 파라미터가 있습니다. 또한 EC2 시작 유형의 경우 워크로드에 맞게 설계된 특정 인스턴스를 선택해야 합니다.

주제

- GPU 워크로드에 대한 Amazon ECS 작업 정의
- 비디오 트랜스코딩 워크로드에 대한 Amazon ECS 태스크 정의
- AWS Neuron 기계 학습 워크로드에 대한 Amazon ECS 작업 정의
- 딥 러닝 인스턴스에 대한 Amazon ECS 작업 정의
- 64비트 ARM 워크로드에 대한 Amazon ECS 작업 정의
- Amazon ECS 로그를 CloudWatch로 전송
- Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송
- Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용
- 컨테이너 재시작 정책이 있는 Amazon ECS 작업의 개별 컨테이너 재시작
- Amazon ECS 컨테이너로 민감한 데이터 전달

GPU 워크로드에 대한 Amazon ECS 작업 정의

GPU를 지원하는 컨테이너 인스턴스로 클러스터를 생성하는 경우 Amazon ECS는 GPU를 사용하는 워크로드를 지원합니다. p2, p3, p5, g3, g4 및 g5 인스턴스 유형을 사용하는 Amazon EC2 GPU 기반 컨테이너 인스턴스는 NVIDIA GPU에 대한 액세스를 제공합니다. 자세한 내용은 Amazon EC2 인스턴스 유형 가이드의 Linux 가속 컴퓨팅 인스턴스를 참조하세요.

Amazon ECS는 NVIDIA 커널 드라이버와 Docker GPU 실행 시간이 구성되어 있어 바로 사용할 수 있는 GPU 최적화 AMI를 제공합니다. 자세한 정보는 Amazon ECS 최적화 Linux AMI을 참조하세요.

컨테이너 수준에서 태스크 할당을 고려할 때 태스크 정의 내 GPU 개수를 지정할 수 있습니다. Amazon ECS는 GPU를 지원하는 사용 가능한 컨테이너 인스턴스로 예약하고 최적의 성능을 위해 물리적 GPU를 적절한 컨테이너에 고정합니다.

다음과 같은 Amazon EC2 GPU 기반 인스턴스 유형이 지원됩니다. 자세한 내용은 Amazon EC2 P2 인스턴스, Amazon EC2 P3 인스턴스, Amazon EC2 P4d 인스턴스, Amazon EC2 P5 인스턴스, Amazon EC2 G3 인스턴스, Amazon EC2 G4 인스턴스, Amazon EC2 G5 인스턴스, Amazon EC2 G6 인스턴스, Amazon EC2 G6e 인스턴스를 참조하세요.

인스턴스 유형	GPU	GPU 메모리 (GiB)	vCPU	메모리(GiB)
p3.2xlarge	1	16	8	61
p3.8xlarge	4	64	32	244
p3.16xlarge	8	128	64	488
p3dn.24xlarge	8	256	96	768
p4d.24xlarge	8	320	96	1152
p5.48xlarge	8	640	192	2048
g3s.xlarge	1	8	4	30.5
g3.4xlarge	1	8	16	122
g3.8xlarge	2	16	32	244
g3.16xlarge	4	32	64	488
g4dn.xlarge	1	16	4	16
g4dn.2xlarge	1	16	8	32
g4dn.4xlarge	1	16	16	64
g4dn.8xlarge	1	16	32	128
g4dn.12xlarge	4	64	48	192

인스턴스 유형	GPU	GPU 메모리 (GiB)	vCPU	메모리(GiB)
g4dn.16xlarge	1	16	64	256
g5.xlarge	1	24	4	16
g5.2xlarge	1	24	8	32
g5.4xlarge	1	24	16	64
g5.8xlarge	1	24	32	128
g5.16xlarge	1	24	64	256
g5.12xlarge	4	96	48	192
g5.24xlarge	4	96	96	384
g5.48xlarge	8	192	192	768
g6.xlarge	1	24	4	16
g6.2xlarge	1	24	8	32
g6.4xlarge	1	24	16	64
g6.8xlarge	1	24	32	128
g6.16.xlarge	1	24	64	256
g6.12xlarge	4	96	48	192
g6.24xlarge	4	96	96	384
g6.48xlarge	8	192	192	768
g6.metal	8	192	192	768
gr6.4xlarge	1	24	16	128
g6e.xlarge	1	48	4	32

인스턴스 유형	GPU	GPU 메모리 (GiB)	vCPU	메모리(GiB)
g6e.2xlarge	1	48	8	64
g6e.4xlarge	1	48	16	128
g6e.8xlarge	1	48	32	256
g6e16.xlarge	1	48	64	512
g6e12.xlarge	4	192	48	384
g6e24.xlarge	4	192	96	768
g6e48.xlarge	8	384	192	1536
gr6.8xlarge	1	24	32	256

AWS Systems Manager Parameter Store API를 쿼리하여 Amazon ECS 최적화 AMI의 Amazon Machine Image(AMI) ID를 검색할 수 있습니다. 이 파라미터를 사용하면 Amazon ECS 최적화 AMI ID를 수동으로 조회할 필요가 없습니다. Systems Manager 파라미터 스토어 API에 대한 자세한 내용은 GetParameter 섹션을 참조하세요. Amazon ECS 최적화 AMI 메타데이터를 검색하려면 사용자에 `ssm:GetParameter` IAM 권한이 있어야 합니다.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended --region us-east-1
```

고려 사항

> **Note**
>
> g2 인스턴스 패밀리 유형에 대한 지원은 더 이상 사용되지 않습니다.
> p2 인스턴스 유형 패밀리는 `20230912` 이전 버전의 Amazon ECS GPU 최적화 AMI에서만 지원됩니다. p2 인스턴스를 계속 사용해야 하는 경우 P2 인스턴스가 필요한 경우 수행할 작업를 참조하세요.
> 이 두 인스턴스 패밀리 유형에서 NVIDIA/CUDA 드라이버를 현재 위치 업데이트하면 잠재적인 GPU 워크로드 실패가 발생합니다.

Amazon ECS에서 GPU 작업을 시작하기 전에 다음 사항을 고려하는 것이 좋습니다.

- 클러스터에 GPU와 비-GPU 컨테이너 인스턴스가 혼재되어 있을 수 있습니다.
- 외부 인스턴스에서 GPU 워크로드를 실행할 수 있습니다. 클러스터에 외부 인스턴스를 등록할 때 `--enable-gpu` 플래그가 설치 스크립트에 포함되어 있는지 확인합니다. 자세한 정보는 Amazon ECS 클러스터에 외부 인스턴스 등록을 참조하세요.
- 에이전트 구성 파일에서 `ECS_ENABLE_GPU_SUPPORT`를 `true`로 설정해야 합니다. 자세한 정보는 the section called “컨테이너 에이전트 구성”을 참조하세요.
- 태스크를 실행하거나 서비스를 생성할 경우 작업 배치 제약 조건을 구성할 때 인스턴스 유형 속성을 사용하여 태스크가 시작될 컨테이너 인스턴스를 결정할 수 있습니다. 이렇게 하면 리소스를 더욱 효과적으로 사용할 수 있습니다. 자세한 정보는 Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법 섹션을 참조하세요.

  다음 예제는 기본 클러스터의 `g4dn.xlarge` 컨테이너 인스턴스에서 태스크를 시작합니다.

  ```
  aws ecs run-task --cluster default --task-definition ecs-gpu-task-def \
       --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
    g4dn.xlarge" --region us-east-2
  ```

- 컨테이너 정의에 지정된 GPU 리소스 요구 사항이 있는 각 컨테이너에 대해 Amazon ECS는 컨테이너 런타임을 NVIDIA 컨테이너 런타임으로 설정합니다.
- NVIDIA 컨테이너 런타임이 제대로 작동하려면 컨테이너에 일부 환경 변수를 설정해야 합니다. 이러한 환경 변수 목록은 Specialized Configurations with Docker를 참조하세요. Amazon ECS는 Amazon ECS가 컨테이너에 할당하는 GPU 디바이스 ID의 목록이 될 `NVIDIA_VISIBLE_DEVICES` 환경 변수 값을 설정합니다. 다른 필수 환경 변수의 경우, Amazon ECS가 이러한 변수를 설정하지 않습니다. 따라서 컨테이너 이미지가 해당 변수를 설정하거나 컨테이너 정의에 설정되어 있어야 합니다.
- p5 인스턴스 유형 패밀리는 `20230929` 이상 버전의 Amazon ECS GPU 최적화 AMI에서 지원됩니다.
- g4 인스턴스 유형 패밀리는 `20230913` 이상 버전의 Amazon ECS GPU 최적화 AMI에서 지원됩니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요. Amazon ECS 콘솔의 클러스터 생성 워크플로에서는 지원되지 않습니다. 이러한 인스턴스 유형을 사용하려면 Amazon EC2 콘솔, AWS CLI 또는 API를 사용하고 클러스터에 인스턴스를 수동으로 등록해야 합니다.
- p4d.24xlarge 인스턴스 유형은 CUDA 11 이상에서만 작동합니다.
- Amazon ECS GPU 최적화 AMI는 IPv6가 활성화되어 있어서 yum을 사용할 때 문제가 발생합니다. 다음 명령으로 IPv4를 사용하도록 yum을 구성하면 문제가 해결됩니다.

```
echo "ip_resolve=4" >> /etc/yum.conf
```

- NVIDIA/CUDA 기본 이미지를 사용하지 않는 컨테이너 이미지를 구축할 때는 `NVIDIA_DRIVER_CAPABILITIES` 컨테이너 런타임 변수를 다음 값 중 하나로 설정해야 합니다.
  - `utility,compute`
  - `all`

  변수를 설정하는 방법에 대한 자세한 정보는 NVIDIA 웹사이트에서 NVIDIA Container Runtime 제어를 참조하세요.
- GPU는 Windows 컨테이너에서 지원되지 않습니다.

Amazon ECS에 대한 GPU 컨테이너 인스턴스 시작

Amazon ECS에서 GPU 인스턴스를 사용하려면 시작 템플릿과 사용자 데이터 파일을 생성하고 인스턴스를 시작해야 합니다.

그런 다음, GPU용으로 구성된 작업 정의를 사용하는 작업을 실행할 수 있습니다.

시작 템플릿 사용

시작 템플릿을 생성할 수 있습니다.

- AMI에 대해 Amazon ECS 최적화 GPU AMI ID를 사용하는 시작 템플릿을 생성합니다. 시작 템플릿을 생성하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 정의한 파라미터를 사용하여 새 시작 템플릿 생성을 참조하세요.

  Amazon 머신 이미지에 대해 이전 단계의 AMI ID를 사용합니다. Systems Manager 파라미터를 사용하여 AMI ID를 지정하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 시작 템플릿에 Systems Manager 파라미터 지정을 참조하세요.

  시작 템플릿의 사용자 데이터에 다음을 추가합니다. *`cluster-name`*을 해당 클러스터의 이름으로 바꿉니다.

  ```
  #!/bin/bash
  echo ECS_CLUSTER=cluster-name >> /etc/ecs/ecs.config;
  echo ECS_ENABLE_GPU_SUPPORT=true >> /etc/ecs/ecs.config
  ```

AWS CLI 사용

AWS CLI를 사용하여 컨테이너 인스턴스를 시작할 수 있습니다.

1. `userdata.toml`이라는 파일을 생성합니다. 이 파일은 인스턴스 사용자 데이터로 사용됩니다. *cluster-name*을 해당 클러스터의 이름으로 바꿉니다.

```
#!/bin/bash
echo ECS_CLUSTER=cluster-name >> /etc/ecs/ecs.config;
echo ECS_ENABLE_GPU_SUPPORT=true >> /etc/ecs/ecs.config
```

2. 다음 명령을 실행하여 GPU AMI ID를 가져옵니다. 다음 단계에서 이 정보를 사용합니다.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended --region us-east-1
```

3. 다음 명령을 실행하여 GPU 인스턴스를 시작합니다. 다음 파라미터를 대체해야 합니다.

   - *subnet*을 인스턴스가 시작될 프라이빗 또는 퍼블릭 서브넷의 ID로 바꿉니다.
   - *gpu_ami*를 이전 단계의 AMI ID로 바꿉니다.
   - *t3.large*를 사용하려는 인스턴스 유형으로 바꿉니다.
   - *region*을 리전 코드로 바꿉니다.

```
aws ec2 run-instances --key-name ecs-gpu-example \
   --subnet-id subnet \
   --image-id gpu_ami \
   --instance-type t3.large \
   --region region \
   --tag-specifications 'ResourceType=instance,Tags=[{Key=GPU,Value=example}]' \
   --user-data file://userdata.toml \
   --iam-instance-profile Name=ecsInstanceRole
```

4. 다음 명령을 실행하여 컨테이너 인스턴스가 클러스터에 등록되었는지 확인합니다. 이 명령을 실행할 때 다음 파라미터를 바꿔야 합니다.

   - *cluster*를 클러스터 이름으로 바꿉니다.
   - *region*을 리전 코드로 바꿉니다.

```
aws ecs list-container-instances --cluster cluster-name --region region
```

Amazon ECS 작업 정의에서 GPU 지정

Docker GPU 실행 시간과 컨테이너 인스턴스의 GPU를 이용하려면 태스크 정의 시 컨테이너에 필요한 GPU 수를 지정해야 합니다. GPU를 지원하는 컨테이너가 배치되면 Amazon ECS 컨테이너 에이전트가 원하는 수의 물리적 GPU를 적절한 컨테이너에 고정합니다. 태스크의 모든 컨테이너에 예약된 GPU 수가 태스크가 실행되는 컨테이너 인스턴스에서 사용할 수 있는 GPU 수를 초과할 수 없습니다. 자세한 정보는 콘솔을 사용하여 Amazon ECS 작업 정의 생성을 참조하세요.

> **Important**
>
> 태스크 정의 때 GPU 조건을 지정하지 않으면 해당 태스크에서 기본 Docker 실행 시간이 사용됩니다.

다음은 태스크 정의 시 GPU 조건의 JSON 형식입니다.

```
{
  "containerDefinitions": [
     {
        ...
        "resourceRequirements" : [
            {
               "type" : "GPU",
               "value" : "2"
            }
        ],
     },
...
}
```

다음 예제는 GPU 조건을 지정하는 Docker 컨테이너용 구문을 보여줍니다. 이 컨테이너는 GPU 2개를 사용하고, `nvidia-smi` 유틸리티를 실행한 후 종료합니다.

```
{
  "containerDefinitions": [
    {
```

```
      "memory": 80,
      "essential": true,
      "name": "gpu",
      "image": "nvidia/cuda:11.0.3-base",
      "resourceRequirements": [
         {
           "type":"GPU",
           "value": "2"
         }
      ],
      "command": [
        "sh",
        "-c",
        "nvidia-smi"
      ],
      "cpu": 100
    }
  ],
  "family": "example-ecs-gpu"
}
```

GPU 공유

GPU를 공유하려면 다음을 구성해야 합니다.

1. 공유되어야 하는 GPU를 Amazon ECS가 예약하지 않도록 태스크 정의에서 GPU 리소스 요구 사항을 제거합니다.
2. GPU를 공유하려면 다음과 같은 사용자 데이터를 인스턴스에 추가합니다. 이렇게 하면 nvidia가 컨테이너 인스턴스의 기본 Docker 컨테이너 런타임이 되므로 모든 Amazon ECS 컨테이너가 GPU를 사용할 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 사용자 데이터 입력을 사용하여 EC2 인스턴스를 시작할 때 명령 실행을 참조하세요.

   ```
   const userData = ec2.UserData.forLinux();
    userData.addCommands(
    'sudo rm /etc/sysconfig/docker',
    'echo DAEMON_MAXFILES=1048576 | sudo tee -a /etc/sysconfig/docker',
    'echo OPTIONS="--default-ulimit nofile=32768:65536 --default-runtime nvidia" |
    sudo tee -a /etc/sysconfig/docker',
    'echo DAEMON_PIDFILE_TIMEOUT=10 | sudo tee -a /etc/sysconfig/docker',
    'sudo systemctl restart docker',
   );
   ```

3. 컨테이너에서 NVIDIA_VISIBLE_DEVICES 환경 변수를 설정합니다. 태스크 정의에서 환경 변수를 지정하면 됩니다. 유효한 값에 대한 자세한 내용은 NVIDIA 설명서 사이트의 GPU Enumeration을 참조하세요.

P2 인스턴스가 필요한 경우 수행할 작업

P2 인스턴스를 사용해야 하는 경우 다음 옵션 중 하나를 사용하여 인스턴스를 계속 사용할 수 있습니다.

두 옵션 모두 인스턴스 사용자 데이터를 수정해야 합니다. 자세한 내용은 Amazon EC2 사용 설명서의 사용자 데이터 입력을 사용하여 EC2 인스턴스를 시작할 때 명령 실행을 참조하세요.

마지막으로 지원되는 GPU 최적화 AMI 사용

GPU 최적화 AMI 20230906 버전을 사용하고 인스턴스 사용자 데이터에 다음을 추가할 수 있습니다.

cluster-name을 해당 클러스터의 이름으로 바꿉니다.

```
#!/bin/bash
echo "exclude=*nvidia* *cuda*" >> /etc/yum.conf
echo "ECS_CLUSTER=cluster-name" >> /etc/ecs/ecs.config
```

최신 GPU 최적화 AMI 사용 및 사용자 데이터 업데이트

인스턴스 사용자 데이터에 다음을 추가할 수 있습니다. 이렇게 하면 Nvidia 535/Cuda12.2 드라이버가 제거되고 Nvidia 470/Cuda11.4 드라이버가 설치되고 버전이 수정됩니다.

```
#!/bin/bash
yum remove -y cuda-toolkit* nvidia-driver-latest-dkms*
tmpfile=$(mktemp)
cat >$tmpfile <<EOF
[amzn2-nvidia]
name=Amazon Linux 2 Nvidia repository
mirrorlist=\$awsproto://\$amazonlinux.\$awsregion.\$awsdomain/\$releasever/amzn2-
nvidia/latest/\$basearch/mirror.list
priority=20
gpgcheck=1
gpgkey=https://developer.download.nvidia.com/compute/cuda/repos/rhel7/
x86_64/7fa2af80.pub
enabled=1
exclude=libglvnd-*
EOF
```

```
mv $tmpfile /etc/yum.repos.d/amzn2-nvidia-tmp.repo
yum install -y system-release-nvidia cuda-toolkit-11-4 nvidia-driver-latest-
dkms-470.182.03
yum install -y libnvidia-container-1.4.0 libnvidia-container-tools-1.4.0 nvidia-
container-runtime-hook-1.4.0 docker-runtime-nvidia-1

echo "exclude=*nvidia* *cuda*" >> /etc/yum.conf
nvidia-smi
```

자체 P2 호환 GPU 최적화 AMI 생성

P2 인스턴스와 호환되는 사용자 지정 Amazon ECS GPU 최적화 AMI를 생성한 다음 AMI를 사용하여 P2 인스턴스를 시작할 수 있습니다.

1. 다음 명령을 실행하여 `amazon-ecs-ami repo`를 복제합니다.

   ```
   git clone https://github.com/aws/amazon-ecs-ami
   ```

2. `release.auto.pkrvars.hcl` 또는 `overrides.auto.pkrvars.hcl`에서 필요한 Amazon ECS 에이전트 및 소스 Amazon Linux AMI 버전을 설정합니다.
3. 다음 명령을 실행하여 프라이빗 P2 호환 EC2 AMI를 빌드합니다.

   region을 인스턴스 리전이 리전으로 바꿉니다.

   ```
   REGION=region make al2keplergpu
   ```

4. AMI에서 다음 인스턴스 사용자 데이터를 사용하여 Amazon ECS 클러스터에 연결합니다.

   cluster-name을 해당 클러스터의 이름으로 바꿉니다.

   ```
   #!/bin/bash
   echo "ECS_CLUSTER=cluster-name" >> /etc/ecs/ecs.config
   ```

비디오 트랜스코딩 워크로드에 대한 Amazon ECS 태스크 정의

Amazon ECS에서 비디오 트랜스코딩 워크로드를 사용하려면 Amazon EC2 VT1 인스턴스를 등록합니다. 이러한 인스턴스를 등록한 후, 라이브 및 사전 렌더링된 비디오 트랜스코딩 워크로드를 Amazon ECS에서 태스크로 실행할 수 있습니다. Amazon EC2 VT1 인스턴스는 Xilinx U30 미디어 트랜스코딩 카드를 사용하여 라이브 및 사전 렌더링된 비디오 트랜스코딩 워크로드 속도를 향상합니다.

> **Note**
>
> Amazon ECS가 아닌 컨테이너에서 비디오 트랜스코딩 워크로드를 실행하는 방법에 대한 지침은 Xilinx 설명서를 참조하세요.

고려 사항

Amazon ECS에 VT1 배포를 시작하기 전에 다음에 주의하세요.

- 클러스터에 VT1과 VT1이 아닌 인스턴스가 혼재되어 있을 수 있습니다.
- 가속 AVC(H.264) 및 HEVC(H.265) 코덱으로 Xilinx U30 미디어 트랜스코딩 카드를 사용하는 Linux 애플리케이션이 필요합니다.

  > **Important**
  >
  > 다른 코덱을 사용하는 애플리케이션은 VT1 인스턴스에서 성능이 향상되지 않을 수 있습니다.

- U30 카드에서 트랜스코딩 태스크는 하나만 실행할 수 있습니다. 각 카드에는 연결된 두 개의 디바이스가 있습니다. 각 VT1 인스턴스 카드만큼 트랜스코딩 태스크를 실행할 수 있습니다.
- 서비스를 생성하거나 독립적 태스크를 실행할 경우 작업 배치 제약 조건을 구성할 때 인스턴스 유형 속성을 사용할 수 있습니다. 이렇게 하면 지정한 컨테이너 인스턴스에서 태스크가 시작됩니다. 또한, 리소스를 효과적으로 사용하고 비디오 트랜스코딩 워크로드에 대한 태스크가 VT1 인스턴스에 있는지 확인할 수 있습니다. 자세한 정보는 Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법을 참조하세요.

  다음의 예제에서는 `default` 클러스터에 있는 `vt1.3xlarge` 인스턴스에서 태스크를 실행합니다.

  ```
  aws ecs run-task \
       --cluster default \
       --task-definition vt1-3xlarge-xffmpeg-processor \
       --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
   vt1.3xlarge"
  ```

- 호스트 컨테이너 인스턴스에서 특정 U30 카드를 사용하도록 컨테이너를 구성합니다. `linuxParameters` 파라미터 및 지정한 디바이스 세부 정보를 사용하여 이 작업을 수행할 수 있습니다. 자세한 정보는 태스크 정의 요구 사항을 참조하세요.

VT1 AMI 사용

Amazon ECS 컨테이너 인스턴스용 Amazon EC2에서 AMI를 실행하는 두 가지 옵션이 있습니다. 첫 번째 옵션은 AWS Marketplace에서 Xilinx 공식 AMI를 사용하는 것입니다. 두 번째 옵션은 샘플 저장소에서 자체 AMI를 빌드하는 것입니다.

- Xilinx는 AWS Marketplace에서 AMI를 제공합니다.
- Amazon ECS는 비디오 트랜스코딩 워크로드에 대한 AMI를 빌드하는 데 사용할 수 있는 샘플 리포지토리를 제공합니다. 이 AMI는 Xilinx U30 드라이버와 함께 제공됩니다. GitHub에서 Packer 스크립트가 포함된 리포지토리를 찾을 수 있습니다. Packer에 대한 자세한 정보는 Packer 설명서를 참조하세요.

태스크 정의 요구 사항

Amazon ECS에서 비디오 트랜스코딩 컨테이너를 실행하려면, 태스크 정의에 가속화된 H.264/AVC 및 H.265/HEVC 코덱을 사용하는 비디오 트랜스코딩 애플리케이션이 포함되어야 합니다. 컨테이너 이미지를 빌드하려면 Xilinx GitHub에서 다음 단계를 수행합니다.

태스크 정의는 인스턴스 유형에 고유해야 합니다. 인스턴스 유형은 3xlarge, 6xlarge 및 24xlarge입니다. 호스트 컨테이너 인스턴스에서 특정 Xilinx U30 디바이스를 사용하도록 컨테이너를 구성해야 합니다. `linuxParameters` 파라미터를 사용해 이 작업을 수행할 수 있습니다. 다음 표에는 각 인스턴스 유형에 해당하는 카드 및 디바이스 SoC가 자세히 나와 있습니다.

인스턴스 유형	vCPU	RAM(GiB)	U30 액셀러레이터 카드	주소 지정 가능한 XCU30 SoC 디바이스	디바이스 경로
vt1.3xlarge	12	24	1	2	`/dev/dri/renderD128 ,/dev/dri/renderD129`
vt1.6xlarge	24	48	2	4	`/dev/dri/renderD128 ,/dev/`

인스턴스 유형	vCPU	RAM(GiB)	U30 액셀러레이터 카드	주소 지정 가능한 XCU30 SoC 디바이스	디바이스 경로
					dri/renderD129 ,/dev/dri/renderD130 ,/dev/dri/renderD131

인스턴스 유형	vCPU	RAM(GiB)	U30 액셀러레이터 카드	주소 지정 가능한 XCU30 SoC 디바이스	디바이스 경로
vt1.24xlarge	96	182	8	16	/dev/dri/renderD128 ,/dev/dri/renderD129 ,/dev/dri/renderD130 ,/dev/dri/renderD131 ,/dev/dri/renderD132 ,/dev/dri/renderD133 ,/dev/dri/renderD134 ,/dev/dri/renderD135 ,/dev/dri/renderD136 ,/dev/dri/renderD137 ,/dev/dri/

인스턴스 유형	vCPU	RAM(GiB)	U30 액셀러레이터 카드	주소 지정 가능한 XCU30 SoC 디바이스	디바이스 경로
					renderD138 ,/dev/dri/renderD139 ,/dev/dri/renderD140 ,/dev/dri/renderD141 ,/dev/dri/renderD142 ,/dev/dri/renderD143

Important

작업 정의에 EC2 인스턴스에 없는 디바이스가 나열되면 태스크가 실행되지 않습니다. 태스크가 실패하면 다음과 같이 오류 메시지가 표시됩니다. `stoppedReason: CannotStartContainerError: Error response from daemon: error gathering device information while adding custom device "/dev/dri/renderD130": no such file or directory`

Amazon ECS 작업 정의에서 비디오 트랜스코딩 지정

다음 예에서는 Amazon EC2에서 Linux 컨테이너의 태스크 정의에 사용되는 구문이 제공됩니다. 이 태스크 정의는 Xilinx 설명서에서 제공하는 절차에 따라 빌드된 컨테이너 이미지용입니다. 이 예제를 사용

하는 경우, 자체 이미지를 사용하여 image를 대체하고 비디오 파일을 /home/ec2-user 디렉터리의 인스턴스로 복사합니다.

vt1.3xlarge

1. 다음 콘텐츠가 포함된 vt1-3xlarge-ffmpeg-linux.json이라는 텍스트 파일을 생성합니다.

```
{
    "family": "vt1-3xlarge-xffmpeg-processor",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == vt1.3xlarge"
        }
    ],
    "containerDefinitions": [
        {
            "entryPoint": [
                "/bin/bash",
                "-c"
            ],
            "command": ["/video/ecs_ffmpeg_wrapper.sh"],
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/dri/renderD128",
                        "hostPath": "/dev/dri/renderD128",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD129",
                        "hostPath": "/dev/dri/renderD129",
                        "permissions": [
                            "read",
```

```
                            "write"
                        ]
                    }
                ]
            },
            "mountPoints": [
                {
                    "containerPath": "/video",
                    "sourceVolume": "video_file"
                }
            ],
            "cpu": 0,
            "memory": 12000,
            "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-
xffmpeg",
            "essential": true,
            "name": "xilinix-xffmpeg"
        }
    ],
    "volumes": [
        {
            "name": "video_file",
            "host": {"sourcePath": "/home/ec2-user"}
        }
    ]
}
```

2. 작업 정의를 등록합니다.

```
aws ecs register-task-definition --family vt1-3xlarge-xffmpeg-processor --cli-
input-json file://vt1-3xlarge-xffmpeg-linux.json --region us-east-1
```

vt1.6xlarge

1. 다음 콘텐츠가 포함된 `vt1-6xlarge-ffmpeg-linux.json`이라는 텍스트 파일을 생성합니다.

```
{
    "family": "vt1-6xlarge-xffmpeg-processor",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
```

```
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == vt1.6xlarge"
        }
    ],
    "containerDefinitions": [
        {
            "entryPoint": [
                "/bin/bash",
                "-c"
            ],
            "command": ["/video/ecs_ffmpeg_wrapper.sh"],
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/dri/renderD128",
                        "hostPath": "/dev/dri/renderD128",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD129",
                        "hostPath": "/dev/dri/renderD129",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD130",
                        "hostPath": "/dev/dri/renderD130",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD131",
                        "hostPath": "/dev/dri/renderD131",
```

```
                        "permissions": [
                            "read",
                            "write"
                        ]
                    }
                ]
            },
            "mountPoints": [
                {
                    "containerPath": "/video",
                    "sourceVolume": "video_file"
                }
            ],
            "cpu": 0,
            "memory": 12000,
            "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-
xffmpeg",
            "essential": true,
            "name": "xilinix-xffmpeg"
        }
    ],
    "volumes": [
        {
            "name": "video_file",
            "host": {"sourcePath": "/home/ec2-user"}
        }
    ]
}
```

2. 작업 정의를 등록합니다.

```
aws ecs register-task-definition --family vt1-6xlarge-xffmpeg-processor --cli-
input-json file://vt1-6xlarge-xffmpeg-linux.json --region us-east-1
```

vt1.24xlarge

1. 다음 콘텐츠가 포함된 `vt1-24xlarge-ffmpeg-linux.json`이라는 텍스트 파일을 생성합니다.

```
{
    "family": "vt1-24xlarge-xffmpeg-processor",
    "requiresCompatibilities": ["EC2"],
```

```
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == vt1.24xlarge"
        }
    ],
    "containerDefinitions": [
        {
            "entryPoint": [
                "/bin/bash",
                "-c"
            ],
            "command": ["/video/ecs_ffmpeg_wrapper.sh"],
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/dri/renderD128",
                        "hostPath": "/dev/dri/renderD128",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD129",
                        "hostPath": "/dev/dri/renderD129",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD130",
                        "hostPath": "/dev/dri/renderD130",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
```

```
                    "containerPath": "/dev/dri/renderD131",
                    "hostPath": "/dev/dri/renderD131",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD132",
                    "hostPath": "/dev/dri/renderD132",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD133",
                    "hostPath": "/dev/dri/renderD133",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD134",
                    "hostPath": "/dev/dri/renderD134",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD135",
                    "hostPath": "/dev/dri/renderD135",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD136",
                    "hostPath": "/dev/dri/renderD136",
                    "permissions": [
                        "read",
```

```
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD137",
                    "hostPath": "/dev/dri/renderD137",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD138",
                    "hostPath": "/dev/dri/renderD138",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD139",
                    "hostPath": "/dev/dri/renderD139",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD140",
                    "hostPath": "/dev/dri/renderD140",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
                    "containerPath": "/dev/dri/renderD141",
                    "hostPath": "/dev/dri/renderD141",
                    "permissions": [
                        "read",
                        "write"
                    ]
                },
                {
```

```
                        "containerPath": "/dev/dri/renderD142",
                        "hostPath": "/dev/dri/renderD142",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD143",
                        "hostPath": "/dev/dri/renderD143",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    }
                ]
            },
            "mountPoints": [
                {
                    "containerPath": "/video",
                    "sourceVolume": "video_file"
                }
            ],
            "cpu": 0,
            "memory": 12000,
            "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-
xffmpeg",
            "essential": true,
            "name": "xilinix-xffmpeg"
        }
    ],
    "volumes": [
        {
            "name": "video_file",
            "host": {"sourcePath": "/home/ec2-user"}
        }
    ]
}
```

2. 작업 정의를 등록합니다.

```
aws ecs register-task-definition --family vt1-24xlarge-xffmpeg-processor --cli-
input-json file://vt1-24xlarge-xffmpeg-linux.json --region us-east-1
```

AWS Neuron 기계 학습 워크로드에 대한 Amazon ECS 작업 정의

기계 학습 워크로드를 위해 Amazon EC2 Trn1, Amazon EC2 Inf1 및 Amazon EC2 Inf2 인스턴스를 클러스터에 등록할 수 있습니다.

Amazon EC2 Trn1 인스턴스는 AWS Trainium 칩으로 구동됩니다. 이러한 인스턴스는 클라우드에서의 기계 학습을 위한 고성능 및 저비용 교육을 제공합니다. Trn1 인스턴스에서 AWS Neuron과 함께 기계 학습 프레임워크를 사용하여 기계 학습 추론 모델을 교육시킬 수 있습니다. 그런 다음 Inf1 인스턴스 또는 Inf2 인스턴스에서 모델을 실행하여 AWS Inferentia 칩의 가속을 사용할 수 있습니다.

Amazon EC2 Inf1 인스턴스 및 Inf2 인스턴스는 AWS Inferentia 칩으로 구동됩니다. 이 칩은 클라우드에서 고성능의 가장 저렴한 추론을 제공합니다.

기계 학습 모델은 특화된 소프트웨어 개발 키트(SDK) AWS Neuron을 사용하여 컨테이너에 배포됩니다. SDK는 AWS 기계 학습 칩의 추론 성능을 최적화하는 컴파일러, 런타임 및 프로파일링 도구로 구성됩니다. AWS Neuron은 TensorFlow, PyTorch, Apache MXNet과 같은 인기 있는 기계 학습 프레임워크를 지원합니다.

고려 사항

Amazon ECS에 Neuron 배포를 시작하기 전에 다음에 주의하세요.

- 클러스터에 Trn1, Inf1, Inf2 및 그 외 인스턴스가 혼재되어 있을 수 있습니다.
- AWS Neuron을 지원하는 기계 학습 프레임워크를 사용하는 컨테이너에 Linux 애플리케이션이 필요합니다.

  > **Important**
  >
  > 다른 프레임워크를 사용하는 애플리케이션은 Trn1, Inf1 및 Inf2 인스턴스에서 성능이 향상되지 않을 수 있습니다.

- 각 AWSTrainium 또는 AWSInferentia 칩에서 단 하나의 추론 또는 추론 교육 태스크만 실행할 수 있습니다. Inf1의 경우 각 칩에는 4개의 NeuronCore가 있습니다. Trn1과 Inf2의 경우 각 칩에는 2개의 NeuronCore가 있습니다. 각 Trn1, Inf1 및 Inf2 인스턴스의 칩이 존재하는 만큼 많은 작업을 실행할 수 있습니다.
- 서비스를 생성하거나 독립적 태스크를 실행할 경우 작업 배치 제약 조건을 구성할 때 인스턴스 유형 속성을 사용할 수 있습니다. 이렇게 하면 지정한 컨테이너 인스턴스에서 태스크가 시작됩니다. 이렇게 하면 전체 리소스 사용률을 최적화하고 추론 워크로드에 대한 작업이 Trn1, Inf1 및 Inf2 인스턴스

에 있는지 확인할 수 있습니다. 자세한 내용은 Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법 섹션을 참조하세요.

다음의 예제에서는 `default` 클러스터에 있는 `Inf1.xlarge` 인스턴스에서 태스크를 실행합니다.

```
aws ecs run-task \
     --cluster default \
     --task-definition ecs-inference-task-def \
     --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
 Inf1.xlarge"
```

- Neuron 리소스 요구 사항은 태스크 정의에서 정의할 수 없습니다. 그 대신에 호스트 컨테이너 인스턴스에서 특정 AWS Trainium 또는 AWS Inferentia 칩을 사용하도록 컨테이너를 구성합니다. `linuxParameters` 파라미터 및 지정한 디바이스 세부 정보를 사용하여 이 작업을 수행할 수 있습니다. 자세한 내용은 태스크 정의 요구 사항 섹션을 참조하세요.

Amazon ECS 최적화 Amazon Linux 2023(Neuron) AMI 사용

Amazon ECS는 AWS Trainium 및 AWS Inferentia 워크로드용 Amazon Linux 2023을 기반으로 하는 Amazon ECS에 최적화된 AMI를 제공합니다. 이는 Docker용 AWS Neuron 드라이버 및 런타임과 함께 제공됩니다. 이 AMI를 사용하면 Amazon ECS에서 더욱 쉽게 기계 학습 인퍼런스 워크로드를 실행할 수 있습니다.

Amazon EC2 Trn1, Inf1, Inf2 인스턴스를 시작할 때 Amazon ECS 최적화 Amazon Linux 2023(Neuron) AMI를 사용하는 것을 권장합니다.

다음 명령으로 AWS CLI를 사용하여 현재 Amazon ECS에 최적화된 Amazon Linux 2023(Neuron) AMI를 검색할 수 있습니다.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/neuron/
recommended
```

Amazon ECS에 최적화된 Amazon Linux 2023(Neuron) AMI는 다음 리전에서 지원됩니다.

- 미국 동부(버지니아 북부)
- 미국 동부(오하이오)
- 미국 서부(캘리포니아 북부)
- 미국 서부(오리건)
- 아시아 태평양(뭄바이)

- 아시아 태평양(오사카)
- 아시아 태평양(서울)
- 아시아 태평양(도쿄)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 캐나다(중부)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)
- 유럽(파리)
- 유럽(스톡홀름)
- 남아메리카(상파울루)

Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI 사용

Amazon ECS는AWS Trainium 및 AWS Inferentia 워크로드용 Amazon Linux 2를 기반으로 하는 Amazon ECS에 최적화된 AMI를 제공합니다. 이는 Docker용 AWS Neuron 드라이버 및 런타임과 함께 제공됩니다. 이 AMI를 사용하면 Amazon ECS에서 더욱 쉽게 기계 학습 인퍼런스 워크로드를 실행할 수 있습니다.

Amazon EC2 Trn1, Inf1 및 Inf2 인스턴스를 시작할 때 Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI를 사용하는 것이 좋습니다.

최신 Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI는 다음 명령으로 AWS CLI에서 검색할 수 있습니다.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/inf/
recommended
```

Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI는 다음 리전에서 지원됩니다.

- 미국 동부(버지니아 북부)
- 미국 동부(오하이오)
- 미국 서부(캘리포니아 북부)
- 미국 서부(오리건)

- 아시아 태평양(뭄바이)
- 아시아 태평양(오사카)
- 아시아 태평양(서울)
- 아시아 태평양(도쿄)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 캐나다(중부)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)
- 유럽(파리)
- 유럽(스톡홀름)
- 남아메리카(상파울루)

태스크 정의 요구 사항

Amazon ECS에 Neuron을 배포하려면 태스크 정의에 TensorFlow용 추론 모델을 제공하는 사전 구축된 컨테이너에 대한 컨테이너 정의를 포함해야 합니다. AWS 딥 러닝 컨테이너에 의해 제공됩니다. 이 컨테이너에는 AWS Neuron 런타임과 TensorFlow Serving 애플리케이션이 포함됩니다. 시작 시 이 컨테이너는 Amazon S3에서 모델을 가져오고, 저장된 모델로 Neuron TensorFlow Serving을 시작하고, 예측 요청을 기다립니다. 다음 예의 컨테이너 이미지에는 TensorFlow 1.15와 Ubuntu 18.04가 있습니다. Neuron에 최적화된 사전 구축된 Deep Learning Containers의 전체 목록은 GitHub에서 관리됩니다. 자세한 내용은 Using AWS Neuron TensorFlow Serving을 참조하세요.

```
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-neuron:1.15.4-neuron-py37-ubuntu18.04
```

또는, 자체 Neuron 사이드카 컨테이너 이미지를 구축할 수 있습니다. 자세한 내용은 AWS Deep Learning AMIs 개발자 안내서의 Tutorial: Neuron TensorFlow Serving을 참조하세요.

작업 정의는 단일 인스턴스 유형에만 고유해야 합니다. 호스트 컨테이너 인스턴스에서 특정 AWS Trainium 또는 AWS Inferentia 디바이스를 사용하도록 컨테이너를 구성해야 합니다. `linuxParameters` 파라미터를 사용해 이 작업을 수행할 수 있습니다. 다음 표에는 각 인스턴스 유형에 해당하는 칩이 자세히 나와 있습니다.

인스턴스 유형	vCPU	RAM(GiB)	AWS ML 액셀러레이터 칩	디바이스 경로
trn1.2xlarge	8	32	1	/dev/neuron0
trn1.32xlarge	128	512	16	/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3 ,/dev/neuron4 ,/dev/neuron5 ,/dev/neuron6 ,/dev/neuron7 ,/dev/neuron8 ,/dev/neuron9 ,/dev/neuron10 ,/dev/neuron11 ,/dev/neuron12 ,/dev/neuron13 ,/dev/neuron14 ,/dev/neuron15
inf1.xlarge	4	8	1	/dev/neuron0
inf1.2xlarge	8	16	1	/dev/neuron0

인스턴스 유형	vCPU	RAM(GiB)	AWS ML 액셀러레이터 칩	디바이스 경로
inf1.6xlarge	24	48	4	`/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3`
inf1.24xlarge	96	192	16	`/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3 ,/dev/neuron4 ,/dev/neuron5 ,/dev/neuron6 ,/dev/neuron7 ,/dev/neuron8 ,/dev/neuron9 ,/dev/neuron10 ,/dev/neuron11 ,/dev/neuron12 ,/dev/neuron13 ,/dev/neuron14 ,/dev/neuron15`
inf2.xlarge	8	16	1	`/dev/neuron0`

인스턴스 유형	vCPU	RAM(GiB)	AWS ML 액셀러레이터 칩	디바이스 경로
inf2.8xlarge	32	64	1	`/dev/neuron0`
inf2.24xlarge	96	384	6	`/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3 ,/dev/neuron4 ,/dev/neuron5 ,`
inf2.48xlarge	192	768	12	`/dev/neuron0 ,/dev/neuron1 ,/dev/neuron2 ,/dev/neuron3 ,/dev/neuron4 ,/dev/neuron5 ,/dev/neuron6 ,/dev/neuron7 ,/dev/neuron8 ,/dev/neuron9 ,/dev/neuron10 ,/dev/neuron11`

Amazon ECS 작업 정의에서 AWS Neuron 기계 학습 지정

다음은 사용할 구문을 표시한 `inf1.xlarge`에 대한 Linux 태스크 정의의 예입니다.

```
{
    "family": "ecs-neuron",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == inf1.xlarge"
        }
    ],
    "executionRoleArn": "${YOUR_EXECUTION_ROLE}",
    "containerDefinitions": [
        {
            "entryPoint": [
                "/usr/local/bin/entrypoint.sh",
                "--port=8500",
                "--rest_api_port=9000",
                "--model_name=resnet50_neuron",
                "--model_base_path=s3://amzn-s3-demo-bucket/resnet50_neuron/"
            ],
            "portMappings": [
                {
                    "hostPort": 8500,
                    "protocol": "tcp",
                    "containerPort": 8500
                },
                {
                    "hostPort": 8501,
                    "protocol": "tcp",
                    "containerPort": 8501
                },
                {
                    "hostPort": 0,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "linuxParameters": {
                "devices": [
                    {
```

```
                        "containerPath": "/dev/neuron0",
                        "hostPath": "/dev/neuron0",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    }
                ],
                "capabilities": {
                    "add": [
                        "IPC_LOCK"
                    ]
                }
            },
            "cpu": 0,
            "memoryReservation": 1000,
            "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference-neuron:1.15.4-neuron-py37-ubuntu18.04",
            "essential": true,
            "name": "resnet50"
        }
    ]
}
```

딥 러닝 인스턴스에 대한 Amazon ECS 작업 정의

Amazon ECS에서 딥 러닝 워크로드를 사용하려면 Amazon EC2 DL1 인스턴스를 클러스터에 등록합니다. Amazon EC2 DL1 인스턴스는 Habana Labs(Intel 회사)의 Gaudi 액셀러레이터로 구동됩니다. Habana SynapseAI SDK를 사용하여 Habana Gaudi 액셀러레이터에 연결합니다. SDK는 TensorFlow, PyTorch와 같은 인기 있는 기계 학습 프레임워크를 지원합니다.

고려 사항

Amazon ECS에 DL1 배포를 시작하기 전에 다음에 주의하세요.

- 클러스터에 DL1과 DL1이 아닌 인스턴스가 혼재되어 있을 수 있습니다.
- 서비스를 생성하거나 독립적 태스크를 실행할 경우 작업 배치 제약 조건을 구성할 때 인스턴스 유형 속성을 사용하여 태스크가 시작되는 지정 컨테이너 인스턴스를 확인할 수 있습니다. 또한, 리소스를 효과적으로 사용하고 딥 러닝 워크로드에 대한 태스크가 DL1 인스턴스에 있는지 확인할 수 있습니다. 자세한 정보는 Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법을 참조하세요.

  다음의 예제에서는 `default` 클러스터에 있는 `dl1.24xlarge` 인스턴스에서 태스크를 실행합니다.

```
aws ecs run-task \
     --cluster default \
     --task-definition ecs-dl1-task-def \
     --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
 dl1.24xlarge"
```

DL1 AMI 사용

Amazon ECS용 Amazon EC2 DL1 인스턴스에서 AMI를 실행하는 세 가지 옵션이 있습니다.

- Habana에서 제공하는 AWS Marketplace AMI를 사용합니다(링크).
- Amazon Web Services에서 제공하는 Habana 딥 러닝 AMI를 사용합니다. 이 AMI는 포함되어 있지 않으므로 Amazon ECS 컨테이너 에이전트를 별도로 설치해야 합니다.
- Packer를 사용하여 GitHub 리포지토리에서 제공하는 사용자 지정 AMI를 빌드합니다. 자세한 정보는 Packer 설명서를 참조하세요.

Amazon ECS 작업 정의에서 딥 러닝 지정

Amazon ECS에서 Habana Gaudi 가속 딥 러닝 컨테이너를 실행하려면, 태스크 정의는 AWS 딥 러닝 컨테이너에서 제공하는 Habana SynapseAI를 사용하여 TensorFlow 또는 PyTorch용 딥 러닝 모델을 제공하는 사전 구축 컨테이너에 대한 컨테이너 정의를 포함해야 합니다.

다음의 컨테이너 이미지에는 TensorFlow 2.7.0과 Ubuntu 20.04가 있습니다. Habana Gaudi 엑셀러레이터에 최적화된 사전 구축된 Deep Learning Containers의 전체 목록은 GitHub에서 관리됩니다. 자세한 정보는 abana Training Containers를 참조하세요.

```
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training-habana:2.7.0-hpu-py38-
synapseai1.2.0-ubuntu20.04
```

다음은 사용할 구문을 표시하는 Amazon EC2의 Linux 컨테이너용 태스크 정의 예입니다. 이 예제에서는 여기(`vault.habana.ai/gaudi-docker/1.1.0/ubuntu20.04/habanalabs/tensorflow-installer-tf-cpu-2.6.0:1.1.0-614`)서 찾은 Habana Labs System Management Interface Tool(HL-SMI)을 포함하는 이미지를 사용합니다.

```
{
    "family": "dl-test",
```

```
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == dl1.24xlarge"
        }
    ],
    "networkMode": "host",
    "cpu": "10240",
    "memory": "1024",
    "containerDefinitions": [
        {
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": ["hl-smi"],
            "cpu": 8192,
            "environment": [
                {
                    "name": "HABANA_VISIBLE_DEVICES",
                    "value": "all"
                }
            ],
            "image": "vault.habana.ai/gaudi-docker/1.1.0/ubuntu20.04/habanalabs/
tensorflow-installer-tf-cpu-2.6.0:1.1.0-614",
            "essential": true,
            "name": "tensorflow-installer-tf-hpu"
        }
    ]
}
```

64비트 ARM 워크로드에 대한 Amazon ECS 작업 정의

Amazon ECS는 64비트 ARM 애플리케이션 사용을 지원합니다. AWS Graviton 프로세서로 구동되는 플랫폼에서 애플리케이션을 실행할 수 있습니다. 다양한 워크로드에 적합합니다. 여기에는 애플리케이션 서버, 마이크로 서비스, 고성능 컴퓨팅, CPU 기반 기계 학습 추론, 비디오 인코딩, 전자 설계 자동화, 게임, 오픈 소스 데이터베이스, 인 메모리 캐시 등의 워크로드가 포함됩니다.

고려 사항

64비트 ARM 아키텍처를 사용하는 태스크 정의 배포를 시작하기 전에 다음에 유의합니다.

- 애플리케이션에서 Fargate 또는 EC2 시작 유형을 사용할 수 있습니다.
- 애플리케이션에서 Linux 운영 체제만 사용할 수 있습니다.
- Fargate 유형의 경우 애플리케이션은 Fargate 플랫폼 버전 `1.4.0` 이상을 사용해야 합니다.
- 애플리케이션은 모니터링을 위해 Fluent Bit 또는 CloudWatch를 사용할 수 있습니다.
- Fargate 시작 유형의 경우 다음 AWS 리전는 64비트 ARM 워크로드를 지원하지 않습니다.
  - 미국 동부(버지니아 북부), `use1-az3` 가용 영역
- Amazon EC2 시작 유형의 경우 다음을 참조하여 사용하려는 인스턴스 유형을 리전에서 지원하는지 확인합니다.
  - Amazon EC2 M6g 인스턴스
  - Amazon EC2 T4g 인스턴스
  - Amazon EC2 C6g 인스턴스
  - Amazon EC2 R6gd 인스턴스
  - Amazon EC2 X2gd 인스턴스

  필터와 함께 Amazon EC2 `describe-instance-type-offerings` 명령을 사용하여 해당 리전에 대한 인스턴스 상품을 볼 수도 있습니다.

```
aws ec2 describe-instance-type-offerings --filters Name=instance-
type,Values=instance-type --region region
```

  다음 예에서는 미국 동부(버지니아 북부)(us-east-1) 리전의 M6 인스턴스 유형 가용성을 확인합니다.

```
aws ec2 describe-instance-type-offerings --filters "Name=instance-type,Values=m6*" --
region us-east-1
```

  자세한 정보는 Amazon EC2 Command Line Reference(Amazon EC2 명령줄 레퍼런스)의 describe-instance-type-offerings를 참조하세요.

Amazon ECS 작업 정의에서 ARM 아키텍처 지정

ARM 아키텍처를 활용하려면 `cpuArchitecture` 태스크 정의 파라미터에 `ARM64`를 지정합니다.

다음 예에서, ARM 아키텍처는 태스크 정의에 지정됩니다. JSON 형식입니다.

```
{
    "runtimePlatform": {
        "operatingSystemFamily": "LINUX",
        "cpuArchitecture": "ARM64"
    },
...
}
```

다음 예는 "hello world"를 표시하는 ARM 아키텍처에 대한 태스크 정의입니다.

```
{
 "family": "arm64-testapp",
 "networkMode": "awsvpc",
 "containerDefinitions": [
    {
        "name": "arm-container",
        "image": "public.ecr.aws/docker/library/busybox:latest",
        "cpu": 100,
        "memory": 100,
        "essential": true,
        "command": [ "echo hello world" ],
        "entryPoint": [ "sh", "-c" ]
    }
 ],
 "requiresCompatibilities": [ "EC2" ],
 "cpu": "256",
 "memory": "512",
 "runtimePlatform": {
        "operatingSystemFamily": "LINUX",
        "cpuArchitecture": "ARM64"
  },
 "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole"
}
```

Amazon ECS 로그를 CloudWatch로 전송

CloudWatch Logs로 로그 정보를 전송하도록 태스크의 컨테이너를 구성할 수 있습니다. 태스크의 Fargate 시작 유형을 사용하는 경우, 컨테이너에서 로그를 볼 수 있습니다. EC2 시작 유형을 사용하고 있다면 한 곳의 편리한 위치에서 컨테이너의 다양한 로그를 볼 수 있으며 컨테이너 로그가 컨테이너 인스턴스에서 디스크 공간을 차지하지 못하도록 방지합니다.

> **Note**
>
> 태스크의 컨테이너에서 기록되는 정보 유형은 대부분 ENTRYPOINT 명령에 따라 결정됩니다. 기본적으로 수집되는 로그는 컨테이너를 로컬에서 실행했을 때 일반적으로 대화식 터미널에 표시되는 명령 출력(STDOUT 및 STDERR I/O 스트림)을 나타냅니다. awslogs 로그 드라이버는 이러한 로그를 Docker에서 CloudWatch Logs로 전달하는 역할만 합니다. 다른 파일 데이터 또는 스트림을 수집할 수 있는 대체 방법을 포함해 Docker 로그가 처리되는 방식에 대한 자세한 정보는 Docker 설명서에서 컨테이너 또는 서비스 로그 보기 섹션을 참조하세요.

Amazon ECS 컨테이너 인스턴스에서 CloudWatch Logs로 시스템 로그를 보내려면 Amazon CloudWatch Logs 사용 설명서의 로그 파일 모니터링 및 CloudWatch Logs 할당량을 참조하세요.

Fargate 시작 유형

작업에 Fargate 시작 유형을 사용하는 경우 작업 정의에 필요한 logConfiguration 파라미터를 추가하여 awslogs 로그 드라이버를 켜야 합니다. 자세한 내용은 Amazon ECS 태스크 정의 예제: 로그를 CloudWatch로 라우팅 섹션을 참조하세요.

Fargate의 Windows 컨테이너의 경우 태스크 정의 파라미터에 특수 문자(예: & \ < > ^ |)가 있는 경우 다음 옵션 중 하나를 수행합니다.

- 전체 파라미터 문자열 주위에 큰따옴표가 포함된 이스케이프(\) 추가

  예시

  ```
  "awslogs-multiline-pattern": "\"^[|DEBUG|INFO|WARNING|ERROR\"",
  ```

- 각 특수 문자 주위에 이스케이프(^) 문자 추가

  예시

  ```
  "awslogs-multiline-pattern": "^^[^|DEBUG^|INFO^|WARNING^|ERROR",
  ```

EC2 시작 유형

해당 태스크에 EC2 시작 유형을 사용하는 경우 awslogs 로그 드라이버를 설정하려면 Amazon ECS 컨테이너 인스턴스에 버전 1.9.0 이상의 컨테이너 에이전트가 필요합니다. 에이전트 버전을 확인하고

최신 버전으로 업데이트하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

> **Note**
>
> Amazon ECS 최적화 AMI 또는 `ecs-init` 패키지의 최소 버전 `1.9.0-1`이 포함된 사용자 지정 AMI를 사용해야 합니다. 사용자 지정 AMI를 사용하는 경우 docker run 문 또는 환경 변수 파일에서 다음 환경 변수를 사용하여 에이전트를 시작할 때 Amazon EC2 인스턴스에서 `awslogs` 로깅 드라이버를 사용할 수 있도록 지정해야 합니다.
>
> ```
> ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
> ```

또한 Amazon ECS 컨테이너 인스턴스는 컨테이너 인스턴스를 시작할 때 사용하는 IAM 역할에 `logs:CreateLogStream` 및 `logs:PutLogEvents` 권한도 요구합니다. Amazon ECS에서 `awslogs` 로그 드라이버를 사용 설정하기 전에 Amazon ECS 컨테이너 인스턴스 역할을 생성한 경우 이 권한을 추가해야 할 수 있습니다. `ecsTaskExecutionRole`은 태스크에 할당되고 올바른 권한이 포함되어 있을 때 사용됩니다. 작업 실행 역할에 대한 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요. 컨테이너 인스턴스가 컨테이너 인스턴스에 대해 관리형 IAM 정책을 사용하는 경우 컨테이너 인스턴스에 올바른 권한이 부여될 수 있습니다. 컨테이너 인스턴스의 관리형 IAM 정책에 대한 자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.

Amazon ECS 태스크 정의 예제: 로그를 CloudWatch로 라우팅

컨테이너가 `awslogs`로 로그를 전송할 수 있으려면 태스크 정의에서 컨테이너에 대한 CloudWatch 로그 드라이버를 지정해야 합니다. 로그 파라미터에 대한 자세한 내용은 스토리지 및 로깅 섹션을 참조하세요.

다음의 작업 정의 JSON에는 각 컨테이너에 대해 지정된 `logConfiguration` 개체가 있습니다. 하나는 `awslogs-wordpress`라는 로그 그룹에 로그를 보내는 WordPress 컨테이너용입니다. 다른 하나는 `awslogs-mysql`이라는 로그 그룹에 로그를 보내는 MySQL 컨테이너용입니다. 두 컨테이너 모두 `awslogs-example` 로그 스트림 접두사를 사용합니다.

```
{
    "containerDefinitions": [
        {
            "name": "wordpress",
            "links": [
                "mysql"
```

```
            ],
            "image": "public.ecr.aws/docker/library/wordpress:latest",
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "awslogs-wordpress",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "awslogs-example"
                }
            },
            "memory": 500,
            "cpu": 10
        },
        {
            "environment": [
                {
                    "name": "MYSQL_ROOT_PASSWORD",
                    "value": "password"
                }
            ],
            "name": "mysql",
            "image": "public.ecr.aws/docker/library/mysql:latest",
            "cpu": 10,
            "memory": 500,
            "essential": true,
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "awslogs-mysql",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "awslogs-example",
                    "mode": "non-blocking",
                    "max-buffer-size": "25m"
                }
            }
```

```
        }
    ],
    "family": "awslogs-example"
}
```

다음 단계

- CloudWatch AWS CLI 또는 API를 사용하여 로그 그룹에 대한 보존 정책을 선택적으로 설정할 수 있습니다. 자세한 내용은 AWS Command Line Interface 참조의 put-retention-policy를 참조하세요.
- 컨테이너 정의 로그 구성에서 `awslogs` 로그 드라이버로 태스크 정의를 등록한 후 사용하여 태스크를 실행하거나 해당 태스크 정의를 사용하여 CloudWatch Logs로 로그 전송을 시작할 수 있습니다. 자세한 내용은 애플리케이션을 Amazon ECS 태스크로 실행 및 Amazon ECS 롤링 업데이트 배포 생성(을)를 참조하세요.

Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송

Amazon ECS용 FireLens를 사용하면 작업 정의 파라미터를 사용하여 로그 스토리지 및 분석을 위해 AWS 서비스 또는 AWS Partner Network(APN) 대상으로 로그를 라우팅할 수 있습니다. AWS Partner Network는 프로그램, 전문 지식 및 리소스를 활용하여 고객 제품을 구축, 마케팅 및 판매하는 글로벌 파트너 커뮤니티입니다. 자세한 내용은 AWS Partner를 참조하세요. FireLens는 Fluentd 및 Fluent Bit와 함께 작동합니다. AWS for Fluent Bit 이미지가 제공되거나 자체 Fluentd 또는 Fluent Bit 이미지를 사용할 수 있습니다.

기본적으로 Amazon ECS는 FireLens 컨테이너가 해당 컨테이너를 사용하는 다른 컨테이너보다 먼저 시작되도록 컨테이너 종속성을 구성합니다. 또한 FireLens 컨테이너는 해당 컨테이너를 사용하는 모든 컨테이너가 중지된 후에 중지됩니다.

Amazon ECS용 FireLens를 사용할 때는 다음 사항을 고려해야 합니다.

- 콘솔에서 컨테이너 이름을 쉽게 구분할 수 있도록 로그 컨테이너 이름에 `my_service_`를 추가하는 것이 좋습니다.
- Amazon ECS는 기본적으로 애플리케이션 컨테이너와 FireLens 컨테이너 사이에서 시작 컨테이너 순서 종속성을 추가합니다. 애플리케이션 컨테이너와 FireLens 컨테이너 사이에서 컨테이너 순서를 지정하면 기본 시작 컨테이너 순서가 재정의됩니다.
- Amazon ECS용 FireLens는 Linux의 AWS Fargate와 Linux의 Amazon EC2 모두에서 호스팅되는 작업에 대해 지원됩니다. Windows 컨테이너는 FireLens를 지원하지 않습니다.

Windows 컨테이너에 대한 중앙 집중식 로깅을 구성하는 방법에 대한 자세한 내용은 Centralized logging for Windows containers on Amazon ECS using Fluent Bit(Fluent Bit를 사용하여 Amazon ECS에서 Windows 컨테이너에 대한 중앙 집중식 로깅)를 참조하세요.

- AWS CloudFormation 템플릿을 사용하여 Amazon ECS를 위한 FireLens를 구성할 수 있습니다. 자세한 정보는 AWS CloudFormation 사용 설명서의 AWS::ECS::TaskDefinition FirelensConfiguration을 참조하세요.
- FireLens의 수신 포트는 24224이므로 FireLens 로그 라우터가 태스크 범위를 벗어나지 않도록 하기 위해, 태스크에서 사용하는 보안 그룹의 포트 24224에서 인바운드 트래픽을 허용해서는 안 됩니다. `awsvpc` 네트워크 모드를 사용하는 작업의 경우 이것은 태스크와 연결된 보안 그룹입니다. `host` 네트워크 모드를 사용하는 태스크의 경우 이것은 태스크를 호스팅하는 Amazon EC2 인스턴스와 연결된 보안 그룹입니다. `bridge` 네트워크 모드를 사용하는 태스크의 경우 포트 24224를 사용하는 포트 매핑을 생성하지 마세요.
- `bridge` 네트워크 모드를 사용하는 작업의 경우 FireLens 구성이 포함된 컨테이너는 해당 컨테이너를 사용하는 모든 애플리케이션 컨테이너가 시작되기 전에 시작해야 합니다. 컨테이너의 시작 순서를 제어하려면 태스크 정의에서 종속성 조건을 사용하세요. 자세한 정보는 컨테이너 종속성 섹션을 참조하세요.

> **Note**
>
> FireLens 구성과 함께 컨테이너 정의에서 종속성 조건 파라미터를 사용하는 경우 각 컨테이너에 `START` 또는 `HEALTHY` 조건 요구 사항이 있는지 확인하세요.

- 기본적으로 FireLens는 클러스터 및 태스크 정의 이름을 추가하고 클러스터의 Amazon 리소스 이름(ARN)을 stdout/stderr 컨테이너 로그에 메타데이터 키로 추가합니다. 다음은 메타데이터 형식의 예입니다.

```
"ecs_cluster": "cluster-name",
"ecs_task_arn": "arn:aws:ecs:region:111122223333:task/cluster-
name/f2ad7dba413f45ddb4EXAMPLE",
"ecs_task_definition": "task-def-name:revision",
```

로그에 메타데이터를 포함시키지 않으려면 태스크 정의의 `firelensConfiguration` 섹션에서 `enable-ecs-log-metadata`를 `false`로 설정합니다.

```
"firelensConfiguration":{
   "type":"fluentbit",
   "options":{
```

```
        "enable-ecs-log-metadata":"false",
        "config-file-type":"file",
        "config-file-value":"/extra.conf"
}
```

이 기능을 사용하려면 태스크에 필요한 AWS 서비스 사용에 필수적인 권한을 제공하는 태스크에 IAM 역할을 생성해야 합니다. 예를 들어, 컨테이너가 로그를 Firehose로 라우팅하는 경우 작업에 `firehose:PutRecordBatch` API를 직접 호출할 수 있는 권한이 필요합니다. 자세한 정보는 IAM 사용 설명서의 IAM 자격 증명 권한 추가 및 제거 섹션을 참조하세요.

다음 조건에서는 태스크 수행 시 Amazon ECS 태스크 실행 역할이 필요할 수도 있습니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

- 태스크가 Fargate에 호스팅되고 Amazon ECR에서 컨테이너 이미지를 가져오거나 로그 구성의 AWS Secrets Manager에서 민감한 데이터를 참조하는 경우, 태스크 실행 IAM 역할을 포함하고 있어야 합니다.
- Amazon S3에서 호스팅되는 사용자 지정 구성 파일을 사용하는 경우에는 작업 실행 IAM 역할에 `s3:GetObject` 권한이 포함되어야 합니다.

호스팅하는 파일 또는 Amazon S3에 있는 파일을 포함하여 Amazon ECS에서 여러 구성 파일을 사용하는 방법에 대한 자세한 내용은 Init process for Fluent Bit on ECS, multi-config support를 참조하세요.

예제 구성에 대한 자세한 내용은 Amazon ECS 태스크 정의 예제: 로그를 FireLens로 라우팅 섹션을 참조하세요.

높은 처리량을 위한 Amazon ECS 로그 구성

작업 정의를 생성할 때 `log-driver-buffer-limit`에 값을 지정하여 메모리에 버퍼링되는 로그 줄 수를 지정할 수 있습니다. 자세한 정보는 Docker 설명서의 Fluentd logging driver(Fluentd 로깅 드라이버)를 참조하세요.

Docker에서 버퍼 메모리가 부족하여 새 메시지를 추가할 수 있도록 버퍼 메시지를 삭제할 수 있으므로 처리량이 많을 때 이 옵션을 사용합니다.

버퍼 제한 옵션과 함께 Amazon ECS용 FireLens를 사용할 때 다음 사항을 고려해야 합니다.

- 이 옵션은 플랫폼 버전 `1.4.0` 이상의 Amazon EC2 시작 유형 및 Fargate 시작 유형에서 지원됩니다.
- 이 옵션은 `logDriver`가 `awsfirelens`로 설정된 경우에만 유효합니다.

- 기본 버퍼 제한은 1048576개의 로그 줄입니다.
- 버퍼 제한은 로그 라인 0줄 이상, 536870912줄 미만이어야 합니다.
- 이 버퍼에 사용되는 최대 메모리 양은 각 로그 줄의 크기와 버퍼 크기를 곱한 값입니다. 예를 들어 애플리케이션의 로그 줄이 평균 2KiB인 경우 4096의 버퍼 제한은 최대 8MiB를 사용합니다. 작업 수준에서 할당된 총 메모리 양은 로그 드라이버 메모리 버퍼 외에도 모든 컨테이너에 할당된 메모리 양보다 커야 합니다.

awsfirelens 로그 드라이버가 태스크 정의에 지정되어 있으면 Amazon ECS 컨테이너 에이전트는 다음 환경 변수를 컨테이너에 주입합니다.

FLUENT_HOST

FireLens 컨테이너에 할당된 IP 주소입니다.

> **Note**
>
> bridge 네트워크 모드에서 EC2 시작 유형을 사용하는 경우 FireLens 로그 라우터 컨테이너(컨테이너 정의에 firelensConfiguration 객체가 있는 컨테이너)를 다시 시작한 후 애플리케이션 컨테이너의 FLUENT_HOST 환경 변수가 부정확해질 수 있습니다. 이는 FLUENT_HOST가 동적 IP 주소이며 재시작 후 변경될 수 있기 때문입니다. 애플리케이션 컨테이너에서 FLUENT_HOST IP 주소로 직접 로깅하면 주소가 변경된 후 실패하기 시작할 수 있습니다. 개별 컨테이너를 재시작하는 방법에 대한 자세한 내용은 컨테이너 재시작 정책이 있는 Amazon ECS 작업의 개별 컨테이너 재시작 섹션을 참조하세요.

FLUENT_PORT

Fluent Forward 프로토콜이 수신 대기 중인 포트입니다.

FLUENT_HOST 및 FLUENT_PORT 환경 변수를 사용하면 stdout을 거치지 않고 코드에서 로그 라우터로 직접 로깅할 수 있습니다. 자세한 정보는 GitHub에서 fluent-logger-golang을 참조하세요.

다음은 log-driver-buffer-limit를 지정하는 구문입니다. my_service_를 사용자 서비스의 이름으로 바꿉니다.

```
{
    "containerDefinitions": [
        {
```

```
            "name": "my_service_log_router",
            "image": "public.ecr.aws/aws-observability/aws-for-fluent-bit:stable",
            "cpu": 0,
            "memoryReservation": 51,
            "portMappings": [],
            "essential": true,
            "environment": [],
            "mountPoints": [],
            "volumesFrom": [],
            "user": "0",
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/ecs-aws-firelens-sidecar-container",
                    "mode": "non-blocking",
                    "awslogs-create-group": "true",
                    "max-buffer-size": "25m",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "firelens"
                },
                "secretOptions": []
            },
            "systemControls": [],
            "firelensConfiguration": {
                "type": "fluentbit"
            }
        },
        {
            "essential": true,
            "image": "public.ecr.aws/docker/library/httpd:latest",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "firehose",
                    "region": "us-west-2",
                    "delivery_stream": "my-stream",
                    "log-driver-buffer-limit": "51200"
                }
            },
            "dependsOn": [
                {
                    "containerName": "log_router",
                    "condition": "START"
```

```
                }
            ],
            "memoryReservation": 100
        }
    ]
}
```

Amazon ECS용 Fluent Bit 이미지 리포지토리에 대한 AWS

AWS는 CloudWatch Logs 및 Firehose 모두에 대해 플러그인과 함께 Fluent Bit 이미지를 제공합니다. Fluent Bit가 Fluentd보다 리소스 사용률이 낮으므로 Fluent Bit를 로그 라우터로 사용하는 것이 좋습니다. 자세한 정보는 Fluent Bit용 CloudWatch Logs 및 Fluent Bit용 Amazon Kinesis Firehose를 참조하세요.

AWS for Fluent Bit 이미지는 고가용성을 위해 대부분 AWS 리전의 Amazon ECR 퍼블릭 갤러리와 Amazon ECR 리포지토리에 있는 Amazon ECR에 사용할 수 있습니다.

Amazon ECR 퍼블릭 갤러리

AWS for Fluent Bit 이미지는 Amazon ECR 퍼블릭 갤러리에서 사용할 수 있습니다. Amazon ECR 퍼블릭 갤러리는 퍼블릭 리포지토리이며 모든 AWS 리전에서 사용할 수 있으므로 여기에 AWS for Fluent Bit 이미지를 다운로드하는 것이 좋습니다. 자세한 정보는 Amazon ECR 퍼블릭 갤러리의 aws-for-fluent-bit를 참조하세요.

Linux

Amazon ECR 퍼블릭 갤러리의 AWS for Fluent Bit 이미지는 `ARM 64` 또는 `x86-64` 아키텍처가 있는 Amazon Linux 운영 체제를 지원합니다.

원하는 이미지 태그로 리포지토리 URL을 지정하여 Amazon ECR 퍼블릭 갤러리에서 AWS for Fluent Bit 이미지를 가져올 수 있습니다. 사용 가능한 이미지 태그는 Amazon ECR 퍼블릭 갤러리의 이미지 태그(Image tags) 탭에서 확인할 수 있습니다.

Docker CLI에 사용할 구문은 다음과 같습니다.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:tag
```

예를 들어 다음 Docker CLI 명령을 사용하여 안정적인 최신 AWS for Fluent Bit 이미지를 가져올 수 있습니다.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:stable
```

> **Note**
>
> 인증되지 않은 풀은 허용되지만 인증된 가져오기보다 속도 제한이 낮습니다. 가져오기 전 AWS 계정을 인증하려면 다음 명령을 사용합니다.
>
> ```
> aws ecr-public get-login-password --region us-east-1 | docker login --username
>  AWS --password-stdin public.ecr.aws
> ```

Windows

Amazon ECR 퍼블릭 갤러리의 AWS for Fluent Bit 이미지는 다음 운영 체제를 사용하는 AMD64 아키텍처를 지원합니다.

- Windows Server 2022 Full
- Windows Server 2022 Core
- Windows Server 2019 Full
- Windows Server 2019 Core

AWS Fargate에 있는 Windows 컨테이너는 FireLens를 지원하지 않습니다.

원하는 이미지 태그로 리포지토리 URL을 지정하여 Amazon ECR 퍼블릭 갤러리에서 AWS for Fluent Bit 이미지를 가져올 수 있습니다. 사용 가능한 이미지 태그는 Amazon ECR 퍼블릭 갤러리의 이미지 태그(Image tags) 탭에서 확인할 수 있습니다.

Docker CLI에 사용할 구문은 다음과 같습니다.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:tag
```

예를 들어 이 Docker CLI 명령을 사용하여 안정적인 최신 AWS for Fluent Bit 이미지를 가져올 수 있습니다.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:windowsservercore-
stable
```

> **Note**
>
> 인증되지 않은 풀은 허용되지만 인증된 가져오기보다 속도 제한이 낮습니다. 가져오기 전 AWS 계정을 인증하려면 다음 명령을 사용합니다.
>
> ```
> aws ecr-public get-login-password --region us-east-1 | docker login --username
>  AWS --password-stdin public.ecr.aws
> ```

Amazon ECR

Fluent Bit용 AWS 이미지는 고가용성을 위해 Amazon ECR에서 사용할 수 있습니다. 이 이미지는 AWS GovCloud (US)를 포함한 대부분의 AWS 리전에서 사용할 수 있습니다.

Linux

다음 명령을 사용하여 안정적인 AWS for Fluent Bit 이미지 URI를 검색할 수 있습니다.

```
aws ssm get-parameters \
      --names /aws/service/aws-for-fluent-bit/stable \
      --region us-east-1
```

다음 명령을 사용하여 모든 버전의 Fluent Bit용 AWS 이미지를 나열하고 Systems Manager 파라미터 스토어 파라미터를 쿼리할 수 있습니다.

```
aws ssm get-parameters-by-path \
      --path /aws/service/aws-for-fluent-bit \
      --region us-east-1
```

안정적인 최신 AWS for Fluent Bit 이미지는 Systems Manager Parameter Store 이름을 참조하여 AWS CloudFormation 템플릿에서 참조할 수 있습니다. 다음은 예제입니다.

```
Parameters:
  FireLensImage:
    Description: Fluent Bit image for the FireLens Container
    Type: AWS::SSM::Parameter::Value<String>
    Default: /aws/service/aws-for-fluent-bit/stable
```

Windows

다음 명령을 사용하여 안정적인 AWS for Fluent Bit 이미지 URI를 검색할 수 있습니다.

```
aws ssm get-parameters \
      --names /aws/service/aws-for-fluent-bit/windowsservercore-stable \
      --region us-east-1
```

다음 명령을 사용하여 모든 버전의 Fluent Bit용 AWS 이미지를 나열하고 Systems Manager 파라미터 스토어 파라미터를 쿼리할 수 있습니다.

```
aws ssm get-parameters-by-path \
      --path /aws/service/aws-for-fluent-bit/windowsservercore \
      --region us-east-1
```

안정적인 최신 AWS for Fluent Bit 이미지는 Systems Manager 파라미터 스토어 이름을 참조하여 AWS CloudFormation 템플릿에서 참조할 수 있습니다. 다음은 예제입니다.

```
Parameters:
  FireLensImage:
    Description: Fluent Bit image for the FireLens Container
    Type: AWS::SSM::Parameter::Value<String>
    Default: /aws/service/aws-for-fluent-bit/windowsservercore-stable
```

Amazon ECS 태스크 정의 예제: 로그를 FireLens로 라우팅

FireLens에서 사용자 정의 로그 라우팅을 사용하려면 태스크 정의에 다음 사항을 지정해야 합니다.

- FireLens 구성을 포함하는 로그 라우터 컨테이너. 컨테이너는 `essential`로 표시하는 것이 좋습니다.
- `awsfirelens` 로그 드라이버를 지정하는 로그 구성이 포함된 하나 이상의 애플리케이션 컨테이너.
- 태스크가 로그를 라우팅하는 데 필요한 권한이 포함된 태스크 IAM 역할 Amazon 리소스 이름(ARN).

AWS Management Console을 사용하여 새로운 태스크 정의를 생성할 때 로그 라우터 컨테이너를 쉽게 추가할 수 있는 FireLens 통합 섹션이 있습니다. 자세한 정보는 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

Amazon ECS는 로그 구성을 변환하고 Fluentd 또는 Fluent Bit 출력 구성을 생성합니다. 출력 구성은 Fluent Bit의 경우 `/fluent-bit/etc/fluent-bit.conf`, Fluentd의 경우 `/fluentd/etc/fluent.conf`에서 로그 라우팅 컨테이너에 마운트됩니다.

> **Important**
>
> FireLens의 수신 포트는 24224입니다. 그러므로 FireLens 로그 라우터가 태스크 범위를 벗어나지 않도록 태스크에서 사용하는 보안 그룹의 포트 24224에서 수신 트래픽을 허용해서는 안 됩니다. `awsvpc` 네트워크 모드를 사용하는 작업의 경우 이것은 태스크와 연결된 보안 그룹입니다. `host` 네트워크 모드를 사용하는 태스크의 경우 이것은 태스크를 호스팅하는 Amazon EC2 인스턴스와 연결된 보안 그룹입니다. `bridge` 네트워크 모드를 사용하는 태스크의 경우 포트 24224를 사용하는 포트 매핑을 생성하지 마세요.

기본적으로 Amazon ECS는 로그 원본을 식별하는 데 도움이 되는 추가 필드를 로그 항목에 추가합니다.

- `ecs_cluster` - 태스크가 속한 클러스터의 이름입니다.
- `ecs_task_arn` – 컨테이너가 속한 태스크의 전체 Amazon 리소스 이름(ARN)
- `ecs_task_definition` - 태스크에서 사용 중인 태스크 정의 이름 및 개정입니다.
- `ec2_instance_id` – 컨테이너가 호스팅되는 Amazon EC2 인스턴스 ID입니다. 이 필드는 EC2 시작 유형을 사용하는 태스크에서만 유효합니다.

메타데이터를 원하지 않는 경우 `enable-ecs-log-metadata`를 `false`로 설정할 수 있습니다.

다음 태스크 정의 예에서는 Fluent Bit를 사용하여 로그를 CloudWatch Logs로 라우팅하는 로그 라우터 컨테이너를 정의합니다. 또한 로그 구성을 사용하여 Amazon Data Firehose로 로그를 라우팅하고 이벤트를 버퍼링하는 데 사용되는 메모리를 2MiB로 설정하는 애플리케이션 컨테이너를 정의합니다.

> **Note**
>
> 자세한 작업 정의 예제는 GitHub의 Amazon ECS FireLens examples를 참조하세요.

```
{
  "family": "firelens-example-firehose",
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
  "containerDefinitions": [
```

```
    {
            "name": "log_router",
            "image": "public.ecr.aws/aws-observability/aws-for-fluent-bit:stable",
            "cpu": 0,
            "memoryReservation": 51,
            "portMappings": [],
            "essential": true,
            "environment": [],
            "mountPoints": [],
            "volumesFrom": [],
            "user": "0",
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/ecs-aws-firelens-sidecar-container",
                    "mode": "non-blocking",
                    "awslogs-create-group": "true",
                    "max-buffer-size": "25m",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "firelens"
                },
                "secretOptions": []
            },
            "systemControls": [],
            "firelensConfiguration": {
                "type": "fluentbit"
            }
        },
    {
      "essential": true,
      "image": "public.ecr.aws/docker/library/httpd:latest",
      "name": "app",
      "logConfiguration": {
        "logDriver": "awsfirelens",
        "options": {
          "Name": "firehose",
          "region": "us-west-2",
          "delivery_stream": "my-stream",
          "log-driver-buffer-limit": "2097152"
        }
      },
      "memoryReservation": 100
    }
  ]
```

```
}
```

`logConfiguration` 객체에 옵션으로 지정된 키 값 페어는 Fluentd 또는 Fluent Bit 출력 구성을 생성하는 데 사용됩니다. 다음은 Fluent Bit 출력 정의의 코드 예시입니다.

```
[OUTPUT]
    Name   firehose
    Match  app-firelens*
    region us-west-2
    delivery_stream my-stream
```

> **Note**
>
> FireLens는 `match` 구성을 관리합니다. 태스크 정의에서 `match` 구성을 지정하지 않습니다.

사용자 지정 구성 파일 사용

사용자 지정 구성 파일을 지정할 수 있습니다. 구성 파일 형식은 사용 중인 로그 라우터의 기본 형식입니다. 자세한 정보는 Fluentd Config 파일 구문과 YAML 구성을 참조하세요.

사용자 정의 구성 파일에서 `bridge` 또는 `awsvpc` 네트워크 모드를 사용하는 태스크의 경우 FireLens가 입력 구성에 Fluentd나 Fluent Bit를 추가하기 때문에 TCP를 통해 Fluentd 또는 Fluent Bit 전달 입력을 설정하지 않습니다.

사용자 정의 구성 파일을 지정하려면 FireLens 구성에 다음 옵션이 포함되어야 합니다.

`config-file-type`

사용자 정의 구성 파일의 원본 위치입니다. 사용할 수 있는 옵션은 `s3` 또는 `file`입니다.

> **Note**
>
> AWS Fargate에 호스팅된 태스크는 `file` 구성 파일 유형만을 지원합니다.

`config-file-value`

사용자 정의 구성 파일의 원본입니다. `s3` 구성 파일 유형을 사용하는 경우 구성 파일 값은 Amazon S3 버킷 및 파일의 전체 ARN입니다. `file` 구성 파일 유형을 사용하는 경우 구성 파일 값은 컨테이너 이미지 또는 컨테이너에 마운트된 볼륨에 있는 구성 파일의 전체 경로입니다.

> **Important**
>
> 사용자 정의 구성 파일을 사용할 때는 FireLens가 사용하는 경로가 아닌 다른 경로를 지정해야 합니다. Amazon ECS는 Fluent Bit의 경우 `/fluent-bit/etc/fluent-bit.conf` 파일 경로를, Fluentd의 경우 `/fluentd/etc/fluent.conf` 파일 경로를 예약합니다.

다음 예에는 사용자 정의 구성을 지정할 때 필요한 구문이 나와 있습니다.

> **Important**
>
> Amazon S3에서 호스팅되는 사용자 정의 구성 파일을 지정하려면 적절한 권한이 있는 태스크 실행 IAM 역할을 생성했는지 확인하세요.

다음은 사용자 정의 구성을 지정할 때 필요한 구문입니다.

```
{
  "containerDefinitions": [
    {
      "essential": true,
      "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-
bit:stable",
      "name": "log_router",
      "firelensConfiguration": {
        "type": "fluentbit",
        "options": {
          "config-file-type": "s3 | file",
          "config-file-value": "arn:aws:s3:::amzn-s3-demo-bucket/fluent.conf
 | filepath"
        }
      }
    }
  ]
}
```

> **Note**
>
> AWS Fargate에 호스팅된 태스크는 `file` 구성 파일 유형만을 지원합니다.

Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용

프라이빗 레지스트리를 사용하여 AWS Secrets Manager에 자격 증명을 저장한 후 작업 정의에서 참조합니다. 이는 태스크 정의에서 인증이 필요한 AWS 외부의 프라이빗 레지스트리에 있는 컨테이너 이미지를 참조하는 방법을 제공합니다. 이 기능은 Fargate, Amazon EC2 인스턴스 및 Amazon ECS Anywhere를 사용하는 외부 인스턴스에서 호스팅되는 태스크에서 지원됩니다.

Important

태스크 정의가 Amazon ECR에 저장된 이미지를 참조하는 경우 이 주제가 적용되지 않습니다. 자세한 정보는 Amazon Elastic Container Registry 사용 설명서의 Amazon ECS에서 Amazon ECR 이미지 사용을 참조하세요.

Amazon EC2 인스턴스에서 호스팅되는 태스크의 경우 이 기능을 사용하려면 버전 `1.19.0` 이상의 컨테이너 에이전트가 있어야 합니다. 그러나 최신 버전의 컨테이너 에이전트를 사용하는 것이 좋습니다. 에이전트 버전을 확인하고 최신 버전으로 업데이트하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

Fargate에서 호스팅되는 태스크의 경우 이 기능을 사용하려면 플랫폼 버전 `1.2.0` 이상이 필요합니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.

컨테이너 정의에서 자신이 생성한 암호의 세부 정보와 함께 `repositoryCredentials` 객체를 지정합니다. 참조하는 보안 암호는 다른 AWS 리전 또는 이를 사용하는 태스크와 다른 계정에서 가져온 것일 수 있습니다.

Note

Amazon ECS API, AWS CLI 또는 AWS SDK를 사용할 때 시작하는 태스크와 같은 AWS 리전에 암호가 존재할 경우, 암호의 전체 ARN 또는 이름을 사용할 수 있습니다. 암호가 다른 계정에 있는 경우 암호의 전체 ARN을 지정해야 합니다. AWS Management Console을 사용할 때는 항상 암호의 전체 ARN을 지정해야 합니다.

다음은 필요한 파라미터를 나타낸 태스크 정의의 예제 조각입니다.

다음 파라미터를 대체합니다.

- *`private-repo`*를 프라이빗 리포지토리 호스트 이름으로 대체

- *private-image*를 이미지 이름으로 대체
- *arn:aws:secretsmanager:region:aws_account_id:secret:secret_name*을 보안 암호의 Amazon 리소스 이름(ARN)으로 대체

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
    }
]
```

> **Note**
>
> 프라이빗 레지스트리 인증을 활성화할 수 있는 또 한 가지 방법은 Amazon ECS 컨테이너 에이전트 환경 변수를 사용해 프라이빗 레지스트리를 인증하는 것입니다. 이 방법은 Amazon EC2 인스턴스에서 호스팅되는 태스크에만 지원됩니다. 자세한 내용은 프라이빗 Docker 이미지를 위한 Amazon ECS 컨테이너 인스턴스 구성 섹션을 참조하세요.

프라이빗 레지스트리를 사용하는 방법

1. 태스크 정의에는 태스크 실행 역할이 있어야 합니다. 컨테이너 에이전트는 이 기능을 통해 컨테이너 이미지를 가져올 수 있습니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

   프라이빗 레지스트리 인증을 사용하면 Amazon ECS 태스크가 인증 자격 증명이 필요한 AWS 외부의 프라이빗 레지스트리(예: Docker Hub, Quay.io 또는 자체 프라이빗 레지스트리)에서 컨테이너 이미지를 가져올 수 있습니다. 이 기능은 Secrets Manager를 사용하여 레지스트리 자격 증명을 안전하게 저장한 다음, `repositoryCredentials` 파라미터를 사용하여 태스크 정의에서 참조합니다.

   프라이빗 레지스트리 인증 구성에 대한 자세한 내용은 Amazon ECS에서 AWS 외의 컨테이너 이미지 사용을 참조하세요.

프라이빗 레지스트리 자격 증명이 포함된 암호에 액세스 권한을 부여하려면 다음 권한을 인라인 정책으로 태스크 실행 역할에 추가합니다. 자세한 정보는 IAM 정책 추가 및 제거 섹션을 참조하세요.

- `secretsmanager:GetSecretValue` - Secrets Manager에서 프라이빗 레지스트리 자격 증명을 검색하는 데 필요합니다.
- `kms:Decrypt` – 암호가 사용자 지정 KMS 키를 사용하고 기본 키를 사용하지 않는 경우에 필요합니다. 사용자 지정 키의 Amazon 리소스 이름(ARN)을 리소스로 추가해야 합니다.

다음 예제에서는 인라인 정책이 권한을 추가합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [

 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:secret_name",
                "arn:aws:kms:<region>:<aws_account_id>:key/key_id"
            ]
        }
    ]
}
```

2. AWS Secrets Manager에서 프라이빗 레지스트리 자격 증명에 사용할 암호를 생성합니다. 보안 암호 생성 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 Create an AWS Secrets Manager secret을 참조하세요.

   다음 형식을 사용하여 프라이빗 레지스트리 자격 증명을 입력합니다.

```
{
  "username" : "privateRegistryUsername",
  "password" : "privateRegistryPassword"
}
```

3. 태스크 정의 등록. 자세한 내용은 the section called “콘솔을 사용하여 작업 정의 생성” 섹션을 참조하세요.

컨테이너 재시작 정책이 있는 Amazon ECS 작업의 개별 컨테이너 재시작

태스크 정의에 정의된 각 필수 및 비필수 컨테이너에 대한 재시작 정책을 사용하여 일시적 장애를 더 빠르게 극복하고 태스크 가용성을 유지할 수 있습니다. 컨테이너의 재시작 정책을 활성화하면 Amazon ECS에서 컨테이너가 종료되면 작업을 교체할 필요 없이 컨테이너를 다시 시작할 수 있습니다.

컨테이너에는 기본적으로 재시작 정책이 활성화되어 있지 않습니다. 컨테이너에 대한 재시작 정책을 활성화하면 컨테이너를 다시 시작하지 않을 종료 코드를 지정할 수 있습니다. 이는 성공을 나타내는 종료 코드일 수 있으며(예: 종료 코드 `0`), 이는 다시 시작할 필요가 없습니다. 컨테이너를 성공적으로 실행해야 재시작을 시도할 수 있는 기간도 지정할 수 있습니다. 이런 파라미터에 대한 자세한 내용은 재시작 정책 섹션을 참조하세요. 이러한 값을 지정하는 작업 정의의 예는 Amazon ECS 작업 정의에서 컨테이너 재시작 정책 지정을 참조하십시오.

Amazon ECS 작업 메타데이터 엔드포인트 또는 CloudWatch 컨테이너 인사이트를 사용하여 컨테이너가 재시작된 횟수를 모니터링할 수 있습니다. 태스크 메타데이터 엔드포인트에 관한 자세한 내용은 Amazon ECS 작업 메타데이터 엔드포인트 버전 4 및 Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 엔드포인트 버전 4 섹션을 참조하세요. Amazon ECS의 Container Insights 지표에 관한 자세한 내용은 Amazon CloudWatch 사용자 설명서의 Amazon ECS Container Insights 지표를 참조하세요.

컨테이너 재시작 정책은 Fargate, Amazon EC2 인스턴스 및 Amazon ECS Anywhere를 사용하는 외부 인스턴스에서 호스팅되는 태스크에서 지원됩니다.

고려 사항

컨테이너에 대해 재시작 정책을 활성화하기 전에 다음 사항을 고려하십시오.

- Fargate에서 Windows 컨테이너에는 재시작 정책이 지원되지 않습니다.
- Amazon EC2 인스턴스에서 호스팅되는 태스크의 경우 이 기능을 사용하려면 버전 `1.86.0` 이상의 컨테이너 에이전트가 있어야 합니다. 그러나 최신 버전의 컨테이너 에이전트를 사용하는 것이 좋습니다. 에이전트 버전을 확인하고 최신 버전으로 업데이트하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.
- Fargate에서 호스팅되는 태스크의 경우 이 기능을 사용하려면 플랫폼 버전 `1.4.0` 이상이 필요합니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.
- `bridge` 네트워크 모드에서 EC2 시작 유형을 사용하는 경우 FireLens 로그 라우터 컨테이너(컨테이너 정의에 `firelensConfiguration` 객체가 있는 컨테이너)를 다시 시작한 후 애플리케이션 컨테

이너의 FLUENT_HOST 환경 변수가 부정확해질 수 있습니다. 이는 FLUENT_HOST가 동적 IP 주소이며 재시작 후 변경될 수 있기 때문입니다. 애플리케이션 컨테이너에서 FLUENT_HOST IP 주소로 직접 로깅하면 주소가 변경된 후 실패하기 시작할 수 있습니다. FLUENT_HOST에 대한 자세한 정보는 높은 처리량을 위한 Amazon ECS 로그 구성 섹션을 참조하세요.

- Amazon ECS 에이전트는 컨테이너 재시작 정책을 처리합니다. 예상치 못한 이유로 Amazon ECS 에이전트가 실패하거나 더 이상 실행되지 않는 경우 컨테이너가 재시작되지 않습니다.
- 정책에 정의된 재시작 시도 기간에 따라 Amazon ECS가 컨테이너를 재시작하기 전에 컨테이너가 실행되어야 하는 시간(초)이 결정됩니다.

Amazon ECS 작업 정의에서 컨테이너 재시작 정책 지정

작업 정의에서 컨테이너의 재시작 정책을 지정하려면 컨테이너 정의 내에서 `restartPolicy` 객체를 지정하십시오. `restartPolicy` 객체에 대한 자세한 정보는 재시작 정책 단원을 참조하세요.

다음은 웹 서버를 설정하는 Fargate 시작 유형에서 Linux 컨테이너를 사용하는 태스크 정의입니다. 컨테이너 정의에는 `restartPolicy` 객체가 포함되며, 컨테이너의 재시작 정책을 활성화하려면 true로 `enabled`가 설정됩니다. 컨테이너를 180초 동안 실행해야 다시 시작할 수 있으며, 성공했음을 나타내는 종료 코드 `0`과 함께 종료되면 다시 시작되지 않습니다.

```
{
  "containerDefinitions": [
    {
      "command": [
        "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
 <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
      ],
      "entryPoint": ["sh", "-c"],
      "essential": true,
      "image": "public.ecr.aws/docker/library/httpd:2.4",
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/fargate-task-definition",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "ecs"
        }
```

```
      },
      "name": "sample-fargate-app",
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "tcp"
        }
      ],
      "restartPolicy": {
        "enabled": true,
        "ignoredExitCodes": [0],
        "restartAttemptPeriod": 180
      }
    }
  ],
  "cpu": "256",
  "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
  "family": "fargate-task-definition",
  "memory": "512",
  "networkMode": "awsvpc",
  "runtimePlatform": {
    "operatingSystemFamily": "LINUX"
  },
  "requiresCompatibilities": ["FARGATE"]
}
```

컨테이너 정의에서 `restartPolicy` 객체로 작업 정의를 등록한 후 해당 작업 정의를 사용하여 작업을 실행하거나 서비스를 생성할 수 있습니다. 자세한 내용은 애플리케이션을 Amazon ECS 태스크로 실행 및 Amazon ECS 롤링 업데이트 배포 생성(을)를 참조하세요.

Amazon ECS 컨테이너로 민감한 데이터 전달

데이터베이스의 보안 인증과 같은 중요 데이터를 컨테이너로 안전하게 전달할 수 있습니다.

API 키 및 데이터베이스 보안 인증과 같은 보안 암호는 애플리케이션이 다른 시스템에 액세스하는 데 자주 사용됩니다. 일반적으로 보안 암호는 사용자 이름과 암호, 인증서 또는 API 키로 구성됩니다. 이러한 보안 암호는 IAM을 사용하는 특정 IAM 보안 주체로 액세스를 제한하고 런타임 시 컨테이너에 주입해야 합니다.

AWS Secrets Manager 및 Amazon EC2 Systems Manager Parameter Store에서 컨테이너에 보안 암호를 원활하게 주입할 수 있습니다. 이러한 보안 암호는 작업에서 다음 중 하나로 참조할 수 있습니다.

1. `secrets` 컨테이너 정의 파라미터를 사용하는 환경 변수로 참조합니다.
2. 로깅 플랫폼에 인증이 필요한 경우 `secretOptions`로 참조합니다. 자세한 내용은 logging configuration options를 참조하세요.
3. 컨테이너가 풀되는 레지스트리에 인증이 필요한 경우 `repositoryCredentials` 컨테이너 정의 파라미터를 사용하여 이미지에 의해 풀되는 보안 암호로 참조합니다. Amazon ECR 퍼블릭 갤러리에서 이미지를 풀할 때 이 방법을 사용합니다. 자세한 내용은 Private registry authentication for tasks를 참조하세요.

보안 암호 관리를 설정할 때 다음을 수행하는 것이 좋습니다.

AWS Secrets Manager 또는 AWS Systems Manager Parameter Store를 사용하여 보안 암호 자료 저장

API 키, 데이터베이스 자격 증명 및 기타 보안 암호 자료를 안전하게 Secrets Manager에 저장하거나 Systems Manager Parameter Store에 암호화된 파라미터로 저장해야 합니다. 두 서비스 모두 민감한 데이터를 암호화하는 AWS KMS를 사용하는 관리형 키-값 저장소라는 점에서 유사합니다. 하지만 Secrets Manager는 보안 암호를 자동으로 교체하고, 보안 암호를 무작위로 생성하고, 계정 간에 보안 암호를 공유하는 기능도 포함합니다. 이러한 기능을 활용하려면 Secrets Manager를 사용합니다. 그렇지 않으면 Systems Manager Parameter Store에서 암호화된 파라미터를 사용합니다.

> **Important**
>
> 보안 암호가 변경되면 새 배포를 강제로 적용하고 새 작업을 시작하여 최신 보안 암호 값을 검색해야 합니다. 자세한 정보는 다음의 주제를 참조하세요.
>
> - 태스크 - 태스크를 중지했다가 시작합니다. 자세한 내용은 Amazon ECS 태스크 중지 및 애플리케이션을 Amazon ECS 태스크로 실행(을)를 참조하세요.
> - 서비스 - 서비스를 업데이트하고 새 배포 강제 적용 옵션을 사용합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.

암호화된 Amazon S3 버킷에서 데이터 검색

보안 암호를 암호화된 Amazon S3 버킷에 저장하고 작업 역할을 사용하여 이러한 보안 암호에 대한 액세스를 제한해야 합니다. 이렇게 하면 환경 변수 값이 실수로 로그에 누출되어 `docker inspect`를 실행할 때 표시되는 문제를 방지할 수 있습니다. 이렇게 하는 경우, Amazon S3 버킷에서 보안 암호를

읽도록 애플리케이션을 작성해야 합니다. 자세한 내용은 Amazon S3 버킷에 대한 기본 서버 측 암호화 동작 설정을 참조하세요.

사이드카 컨테이너를 사용하여 보안 암호를 볼륨에 마운트

환경 변수를 사용하면 데이터 누출 위험이 높아지므로 보안 암호를 AWS Secrets Manager에서 읽고 공유 볼륨에 쓰는 사이드카 컨테이너를 실행해야 합니다. Amazon ECS 컨테이너 순서를 사용하면 이 컨테이너를 애플리케이션 컨테이너보다 먼저 실행하고 종료할 수 있습니다. 이렇게 하면 이후 애플리케이션 컨테이너는 보안 암호가 기록된 볼륨을 마운트합니다. Amazon S3 버킷 방법과 마찬가지로 공유 볼륨에서 보안 암호를 읽도록 애플리케이션을 작성해야 합니다. 이 볼륨은 범위가 작업으로 지정되므로 작업이 중지되면 자동으로 삭제됩니다. 예제는 task-def.json 프로젝트를 참조하세요.

Amazon EC2에서는 보안 암호가 기록되는 볼륨을 AWS KMS 고객 관리형 키로 암호화할 수 있습니다. AWS Fargate에서는 서비스 관리형 키를 사용하여 볼륨 스토리지를 자동으로 암호화합니다.

개별 환경 변수를 Amazon ECS 컨테이너로 전달

> **Important**
>
> 민감한 데이터는 AWS Secrets Manager 암호 또는 AWS Systems Manager 파라미터 스토어 파라미터에 저장하는 것이 좋습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.
>
> 작업 정의에 지정된 환경 변수는 작업 정의에 대해 `DescribeTaskDefinition` 작업이 허용된 모든 사용자 및 역할에서 읽을 수 있습니다.

환경 변수는 다음과 같은 방법으로 컨테이너에 전달할 수 있습니다.

- `environment` 컨테이너 정의 파라미터를 개별적으로 사용합니다. 이것은 docker container run에 대한 `--env` 옵션에 매핑됩니다.
- 대량으로 `environmentFiles` 컨테이너 정의 파라미터를 사용하여 환경 변수를 포함하는 하나 이상의 파일을 나열합니다. 파일은 Amazon S3에서 호스팅되어야 합니다. 이것은 docker run에 대한 `--env-file` 옵션에 매핑됩니다.

다음은 개별 환경 변수를 지정하는 방법을 보여주는 태스크 정의의 조각입니다.

```
{
    "family": "",
```

```
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            ...
            "environment": [
                {
                    "name": "variable",
                    "value": "value"
                }
            ],
            ...
        }
    ],
    ...
}
```

환경 변수를 Amazon ECS 컨테이너로 전달

> **Important**
>
> 민감한 데이터는 AWS Secrets Manager 암호 또는 AWS Systems Manager 파라미터 스토어 파라미터에 저장하는 것이 좋습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.
> 환경 변수 파일은 Amazon S3의 객체이며 모든 Amazon S3 보안 고려 사항이 적용됩니다.
> Windows 컨테이너와 Fargate의 Windows 컨테이너에서는 `environmentFiles` 파라미터를 사용할 수 없습니다.

환경 변수 파일을 생성하고 Amazon S3에 저장하여 컨테이너로 환경 변수를 전달할 수 있습니다.

파일에 환경 변수를 지정하여 환경 변수를 대량으로 주입할 수 있습니다. 컨테이너 정의 내에서 환경 변수 파일이 포함된 Amazon S3 버킷 목록을 이용해 `environmentFiles` 객체를 지정합니다.

Amazon ECS는 환경 변수에 크기 제한을 적용하지 않지만 용량이 큰 환경 변수 파일로 인해 디스크 공간이 가득 찰 수도 있습니다. 환경 변수 파일을 사용하는 각 태스크는 해당 파일 사본을 디스크에 다운로드합니다. Amazon ECS에서는 작업 정리의 일부로 파일을 제거합니다.

지원되는 환경 변수에 대한 자세한 내용은 Advanced container definition parameters- Environment를 참조하세요.

컨테이너 정의에서 환경 변수 파일을 지정할 때는 다음 사항을 고려합니다.

- Amazon EC2에 있는 Amazon ECS 태스크의 경우 이 기능을 사용하기 위해 컨테이너 인스턴스에 버전 `1.39.0` 이상의 컨테이너 에이전트가 필요합니다. 에이전트 버전을 확인하고 최신 버전으로 업데이트하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.
- AWS Fargate에 있는 Amazon ECS 태스크의 경우 이 기능을 사용하기 위해 플랫폼 버전 `1.4.0` 이상(Linux)을 사용해야 합니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.

  변수가 운영 체제 플랫폼에 대해 지원되는지 확인합니다. 자세한 내용은 the section called “컨테이너 정의” 및 the section called “기타 태스크 정의 파라미터” 단원을 참조하세요.
- 파일은 `.env` 파일 확장명과 UTF-8 인코딩을 사용해야 합니다.
- 태스크 실행 역할은 Amazon S3에 대한 추가 권한과 함께 이 기능을 사용해야 합니다. 이렇게 하면 컨테이너 에이전트가 Amazon S3에서 환경 변수 파일을 가져올 수 있습니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.
- 작업 정의당 파일 수는 10개로 제한됩니다.
- 환경 파일의 각 줄에는 `VARIABLE=VALUE` 형식의 환경 변수를 포함합니다. 공백이나 인용 부호는 Amazon ECS 파일 값의 일부로 포함됩니다. #으로 시작하는 줄은 주석으로 처리되며 무시됩니다. 환경 변수 파일 구문에 대한 자세한 내용은 Docker 설명서의 환경 변수 설정(-e, --env, --env-file)을 참조하세요.

  다음은 적절한 구문입니다.

  ```
  #This is a comment and will be ignored
  VARIABLE=VALUE
  ENVIRONMENT=PRODUCTION
  ```
- 컨테이너 정의에서 `environment` 파라미터를 사용하여 지정된 환경 변수가 있는 경우 환경 파일 내에 포함된 변수보다 우선합니다.
- 여러 환경 파일이 지정되고 동일한 변수를 포함하는 경우 입력 순서대로 처리됩니다. 즉, 변수의 첫 번째 값이 사용되고 중복 변수의 후속 값은 무시됩니다. 고유한 변수 이름을 사용하는 것이 좋습니다.
- 환경 파일이 컨테이너 재정의로 지정된 경우 해당 파일이 사용됩니다. 또한 컨테이너 정의에 지정된 다른 환경 파일은 무시됩니다.
- Fargate 시작 유형에는 다음 규칙이 적용됩니다.

- 파일은 네이티브 Docker env 파일과 유사하게 처리됩니다.
- 비어 있는 상태로 Amazon S3에 저장된 환경 변수를 참조하는 컨테이너 정의는 컨테이너에 표시되지 않습니다.
- 셸 이스케이프 처리는 지원되지 않습니다.
- 컨테이너 진입점은 VARIABLE 값을 해석합니다.

예시

다음은 환경 변수 파일을 지정하는 방법을 보여주는 태스크 정의의 조각입니다.

```
{
    "family": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            ...
            "environmentFiles": [
                {
                    "value": "arn:aws:s3:::amzn-s3-demo-
bucket/envfile_object_name.env",
                    "type": "s3"
                }
            ],
            ...
        }
    ],
    ...
}
```

Amazon ECS에서 프로그래밍 방식으로 Secrets Manager 보안 암호 전달

애플리케이션에서 민감한 정보를 일반 텍스트로 하드 코딩하는 대신, Secrets Manager를 사용하여 민감한 데이터를 저장할 수 있습니다.

중요 데이터를 검색할 때 이 방법을 권장하는 이유는 이후에 Secrets Manager 보안 암호가 업데이트되면 애플리케이션이 자동으로 최신 버전의 보안 암호를 검색하기 때문입니다.

Secrets Manager에서 보안 암호를 생성합니다. Secrets Manager 보안 암호를 생성한 후 애플리케이션 코드를 업데이트하여 보안 암호를 검색하세요.

Secrets Manager에서 중요 데이터를 보호하기 전에 다음 고려 사항을 검토하세요.

- CreateSecret API의 `SecretString` 파라미터로 생성된 암호이며 텍스트 데이터를 저장하는 암호만이 지원됩니다. 이진 데이터를 저장하는 보안 암호(CreateSecret API의 `SecretBinary` 파라미터로 생성된 보안 암호임)는 지원되지 않습니다.
- 인터페이스 VPC 엔드포인트를 사용하여 보안 제어를 강화합니다. Secrets Manager용 인터페이스 VPC 엔드포인트를 생성해야 합니다. VPC 엔드포인트에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 VPC 종단점 생성을 참조하세요.
- 작업에서 사용하는 VPC는 DNS 확인을 사용해야 합니다.
- 태스크 정의에서 Secrets Manager에 대한 추가 권한을 보유한 태스크 역할을 사용해야 합니다. 자세한 내용은 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.

Secrets Manager 보안 암호 생성

Secrets Manager 콘솔을 사용하여 민감한 데이터에 대한 암호를 생성할 수 있습니다. 보안 암호 생성 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager 보안 암호 생성을 참조하세요.

애플리케이션을 업데이트하여 프로그래밍 방식으로 Secrets Manager 보안 암호 검색

애플리케이션에서 직접 Secrets Manager API를 호출하여 보안 암호를 검색할 수 있습니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 Retrieve secrets from AWS Secrets Manager를 참조하세요.

AWS Secrets Manager에 저장된 민감한 데이터를 검색하려면 AWS SDK 코드 예제 코드 라이브러리의 Code examples for AWS Secrets Manager using AWS SDKs를 참조하세요.

Amazon ECS에서 프로그래밍 방식으로 Systems Manager Parameter Store 보안 암호 전달

Systems Manager Parameter Store는 보안 저장 및 보안 암호 관리 기능을 제공합니다. 애플리케이션에 이 정보를 하드 코딩하는 대신 암호, 데이터베이스 문자열, EC2 인스턴스 ID, AMI ID, 라이선스 코드와 같은 데이터를 파라미터 값으로 저장할 수 있습니다. 값을 일반 텍스트 또는 암호화된 데이터로 저장할 수 있습니다.

민감한 데이터를 검색할 때 이 방법을 권장하는 이유는 이후에 Secrets Manager Parameter Store 파라미터가 업데이트되면 애플리케이션이 자동으로 최신 버전을 검색하기 때문입니다.

Systems Manager Parameter Store에서 중요 데이터를 보호하기 전에 다음 고려 사항을 검토하세요.

- 텍스트 데이터를 저장하는 보안 암호만 지원됩니다. 이진 데이터를 저장하는 보안 암호는 지원되지 않습니다.
- 인터페이스 VPC 엔드포인트를 사용하여 보안 제어를 강화합니다.
- 작업에서 사용하는 VPC는 DNS 확인을 사용해야 합니다.
- EC2 시작 유형을 사용하는 태스크의 경우, 이 기능을 사용하려면 Amazon ECS 에이전트 구성 변수인 `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true`를 사용해야 합니다. 컨테이너 인스턴스를 생성할 때 이를 `/etc/ecs/ecs.config` 파일에 추가하거나 기존 인스턴스에 추가한 다음 ECS 에이전트를 다시 시작할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.
- 태스크 정의에서 Systems Manager Parameter Store에 대한 추가 권한을 보유한 태스크 역할을 사용해야 합니다. 자세한 내용은 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.

파라미터 생성

Systems Manager 콘솔을 사용하면 중요 데이터에 대한 Systems Manager Parameter Store 파라미터를 생성할 수 있습니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 Systems Manager 파라미터 생성(콘솔) 또는 Systems Manager 파라미터 생성(AWS CLI)을 참조하세요.

애플리케이션을 업데이트하여 프로그래밍 방식으로 Systems Manager Parameter Store 보안 암호 검색

Systems Manager Parameter Store 파라미터에 저장된 민감한 데이터를 검색하려면 AWS SDK 코드 예제 코드 라이브러리의 Code examples for Systems Manager using AWS SDKs를 참조하세요.

Amazon ECS 환경 변수를 통해 Secrets Manager 보안 암호 전달

암호를 환경 변수로 주입하는 경우 암호의 전체 내용, 암호 내 특정 JSON 키를 지정할 수 있습니다. 이는 컨테이너에 노출되는 민감한 데이터를 제어하는 데 도움이 됩니다. 암호 버전 관리에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 Secrets Manager에는 무엇이 있습니까?를 참조하세요.

환경 변수를 사용하여 컨테이너에 Secrets Manager 보안 암호를 삽입할 때 다음을 고려해야 합니다.

- 컨테이너가 처음 시작될 때 해당 컨테이너에 중요한 정보가 주입됩니다. 암호가 이후에 업데이트되거나 교체되면 컨테이너가 업데이트된 값을 자동으로 받지 않습니다. 새 태스크를 시작해야 하거나 작업이 서비스의 일부인 경우 서비스를 업데이트하고 새 배포 강제 적용을 사용하여 서비스에서 새 태스크를 시작하도록 강제로 지정할 수 있습니다.
- 컨테이너에서 실행되는 애플리케이션, 컨테이너 로그 및 디버깅 도구는 환경 변수에 액세스할 수 있습니다.

- AWS Fargate에서의 Amazon ECS 태스크의 경우 다음 사항을 고려합니다.
  - 암호의 전체 내용을 환경 변수 또는 로그 구성으로 주입하려면 플랫폼 버전 `1.3.0` 이상을 사용해야 합니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.
  - 암호의 JSON 키 또는 버전을 환경 변수 또는 로그 구성으로 주입하려면 플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0`(Windows)을 사용해야 합니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.
- EC2에서의 Amazon ECS 태스크의 경우 다음 사항을 고려해야 합니다.
  - 암호의 특정 JSON 키 또는 버전을 사용하여 암호를 주입하려면 컨테이너 인스턴스에 컨테이너 에이전트 버전 `1.37.0` 이상이 있어야 합니다. 그러나 최신 버전의 컨테이너 에이전트를 사용하는 것이 좋습니다. 에이전트 버전을 확인하고 최신 버전으로 업데이트하는 방법에 대한 자세한 내용은 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

    암호의 전체 내용을 환경 변수로 주입하거나 로그 구성에 암호를 주입하려면 컨테이너 인스턴스에 컨테이너 에이전트 버전 `1.22.0` 이상이 있어야 합니다.
- 인터페이스 VPC 엔드포인트를 사용하여 보안 제어를 향상하고 프라이빗 서브넷을 통해 Secrets Manager에 연결합니다. Secrets Manager용 인터페이스 VPC 엔드포인트를 생성해야 합니다. VPC 엔드포인트에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 VPC 종단점 생성을 참조하세요. Secrets Manager 및 Amazon VPC 사용에 대한 자세한 내용은 How to connect to Secrets Manager service within a Amazon VPC를 참조하세요.
- `awslogs` 로깅 드라이버를 사용하도록 구성된 Windows 태스크의 경우 컨테이너 인스턴스에 `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` 환경 변수도 설정해야 합니다. 다음 구문을 사용합니다.

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE",
 $TRUE, "Machine")
Initialize-ECSAgent -Cluster <cluster name> -EnableTaskIAMRole -LoggingDrivers
 '["json-file","awslogs"]'
</powershell>
```

- 태스크 정의에서 Secrets Manager에 대한 추가 권한을 보유한 작업 실행 역할을 사용해야 합니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

AWS Secrets Manager 보안 암호 생성

Secrets Manager 콘솔을 사용하여 민감한 데이터에 대한 암호를 생성할 수 있습니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager 보안 암호 생성을 참조하세요.

컨테이너 정의에 환경 변수 추가

컨테이너 정의 내에서 다음을 지정할 수 있습니다.

- 컨테이너에 설정할 환경 변수의 이름을 포함하는 `secrets` 객체
- Secrets Manager 암호의 Amazon 리소스 이름(ARN)
- 컨테이너에 제공할 중요한 데이터를 포함하는 추가 파라미터

다음 예제에서는 Secrets Manager 암호에 대해 지정해야 하는 전체 구문을 보여 줍니다.

```
arn:aws:secretsmanager:region:aws_account_id:secret:secret-name:json-key:version-stage:version-id
```

다음 섹션에서는 추가 파라미터에 대해 설명합니다. 이러한 파라미터는 선택 사항이지만 사용하지 않는 경우 기본값을 사용하려면 콜론(:)을 포함시켜야 합니다. 추가 컨텍스트에 대한 예제가 아래에 나와 있습니다.

`json-key`

환경 변수 값으로 설정할 값과 함께 키-값 쌍의 키 이름을 지정합니다. JSON 형식의 값만 지원됩니다. JSON 키를 지정하지 않으면 암호의 전체 내용이 사용됩니다.

`version-stage`

사용할 암호 버전의 스테이징 레이블을 지정합니다. 버전 스테이징 레이블이 지정된 경우 버전 ID를 지정할 수 없습니다. 버전 단계가 지정되지 않은 경우 기본 동작은 `AWSCURRENT` 스테이징 레이블을 사용하여 암호를 검색하는 것입니다.

스테이징 레이블은 암호가 업데이트되거나 교체되는 경우 암호의 여러 버전을 추적하는 데 사용됩니다. 암호의 각 버전에는 하나 이상의 스테이징 레이블과 ID가 있습니다.

`version-id`

사용하고자 하는 암호 버전의 고유 식별자를 지정합니다. 버전 ID가 지정된 경우 버전 스테이징 레이블을 지정할 수 없습니다. 버전 ID가 지정되지 않은 경우 기본 동작은 `AWSCURRENT` 스테이징 레이블을 사용하여 암호를 검색하는 것입니다.

버전 ID는 암호가 업데이트되거나 교체되는 경우 암호의 여러 버전을 추적하는 데 사용됩니다. 암호의 각 버전에는 ID가 있습니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager의 주요 개념 및 용어를 참조하세요.

컨테이너 정의 예제

다음 예제에서는 컨테이너 정의에서 Secretes Manager 암호를 참조할 수 있는 방법을 보여 줍니다.

Example 전체 암호 참조

다음은 Secret Manager 암호의 전체 텍스트를 참조할 때 형식을 나타내는 태스크 정의의 조각입니다.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-
AbCdEf"
    }]
  }]
}
```

컨테이너 내에서 이 보안 암호의 값에 액세스하려면 `$environment_variable_name` 변수를 호출해야 합니다.

Example 전체 시크릿 참조

다음은 여러 Secret Manager 시크릿의 전체 텍스트를 참조할 때 형식을 나타내는 태스크 정의의 조각입니다.

```
{
  "containerDefinitions": [{
     "secrets": [
      {
        "name": "environment_variable_name1",
         "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-
AbCdEf"
      },
      {
        "name": "environment_variable_name2",
         "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-
abcdef"
      },
      {
        "name": "environment_variable_name3",
        "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-
ABCDEF"
```

```
      }
    ]
  }]
}
```

컨테이너 내에서 이 시크릿의 값에 액세스하려면 $environment_variable_name1, $environment_variable_name2, $environment_variable_name3을 직접적으로 호출해야 합니다.

Example 암호 내에서 특정 키 참조

다음은 암호의 내용을 관련 버전 스테이징 레이블 및 버전 ID와 함께 표시하는 get-secret-value 명령의 출력 예를 보여 줍니다.

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "VersionId": "871d9eca-18aa-46a9-8785-981ddEXAMPLE",
    "SecretString": "{\"username1\":\"password1\",\"username2\":\"password2\",
\"username3\":\"password3\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": 1581968848.921
}
```

ARN 끝에 키 이름을 지정하여 컨테이너 정의에서 이전 출력의 특정 키를 참조합니다.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::"
    }]
  }]
}
```

Example 특정 비밀 버전 참조

다음은 암호의 암호화되지 않은 내용을 모든 버전의 암호에 대한 메타데이터와 함께 표시하는 describe-secret 명령의 출력 예입니다.

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "Description": "Example of a secret containing application authorization data.",
    "RotationEnabled": false,
    "LastChangedDate": 1581968848.926,
    "LastAccessedDate": 1581897600.0,
    "Tags": [],
    "VersionIdsToStages": {
        "871d9eca-18aa-46a9-8785-981ddEXAMPLE": [
            "AWSCURRENT"
        ],
        "9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE": [
            "AWSPREVIOUS"
        ]
    }
}
```

ARN 끝에 키 이름을 지정하여 컨테이너 정의에서 이전 출력의 특정 버전 스테이징 레이블을 참조합니다.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf::AWSPREVIOUS:"
    }]
  }]
}
```

ARN 끝에 키 이름을 지정하여 컨테이너 정의에서 이전 출력의 특정 버전 ID를 참조합니다.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf:::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"
    }]
  }]
}
```

Example 암호의 특정 키 및 버전 스테이징 레이블 참조

다음은 암호 내 특정 키와 특정 버전 스테이징 레이블을 모두 참조하는 방법을 보여줍니다.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1:AWSPREVIOUS:"
    }]
  }]
}
```

특정 키 및 버전 ID를 지정하려면 다음 구문을 사용합니다.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"
    }]
  }]
}
```

환경 변수에 지정된 보안 암호를 사용하여 태스크 정의를 생성하는 방법에 대한 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

Amazon ECS 환경 변수를 통해 Systems Manager 파라미터 전달

Amazon ECS를 사용하면 AWS Systems Manager Parameter Store 파라미터에 민감한 데이터를 저장한 후 컨테이너 정의에서 참조하여 컨테이너에 민감한 데이터를 주입할 수 있습니다.

환경 변수를 사용하여 Systems Manager 보안 암호를 컨테이너에 주입할 때 다음을 고려하세요.

- 컨테이너가 처음 시작될 때 해당 컨테이너에 중요한 정보가 주입됩니다. 암호가 이후에 업데이트되거나 교체되면 컨테이너가 업데이트된 값을 자동으로 받지 않습니다. 새 태스크를 시작해야 하거나 작업이 서비스의 일부인 경우 서비스를 업데이트하고 새 배포 강제 적용을 사용하여 서비스에서 새 태스크를 시작하도록 강제로 지정할 수 있습니다.
- AWS Fargate에서의 Amazon ECS 태스크의 경우 다음 사항을 고려해야 합니다.

  - 암호의 전체 내용을 환경 변수 또는 로그 구성으로 주입하려면 플랫폼 버전 `1.3.0` 이상을 사용해야 합니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.
  - 암호의 JSON 키 또는 버전을 환경 변수 또는 로그 구성으로 주입하려면 플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0`(Windows)을 사용해야 합니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.
- EC2에서의 Amazon ECS 태스크의 경우 다음 사항을 고려해야 합니다.
  - 암호의 특정 JSON 키 또는 버전을 사용하여 암호를 주입하려면 컨테이너 인스턴스에 컨테이너 에이전트 버전 `1.37.0` 이상이 있어야 합니다. 그러나 최신 버전의 컨테이너 에이전트를 사용하는 것이 좋습니다. 에이전트 버전을 확인하고 최신 버전으로 업데이트하는 방법에 대한 자세한 내용은 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

    암호의 전체 내용을 환경 변수로 주입하거나 로그 구성에 암호를 주입하려면 컨테이너 인스턴스에 컨테이너 에이전트 버전 `1.22.0` 이상이 있어야 합니다.
- 인터페이스 VPC 엔드포인트를 사용하여 보안 제어를 강화합니다. Systems Manager용 인터페이스 VPC 엔드포인트를 생성해야 합니다. VPC 엔드포인트에 대한 자세한 내용은 AWS Systems Manager 사용 설명서의 Systems Manager용 VPC 엔드포인트를 사용하여 EC2 인스턴스의 보안 개선을 참조하세요.
- 태스크 정의에서는 Systems Manager Parameter Store에 대한 추가 권한을 보유한 태스크 실행 역할을 사용해야 합니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.
- `awslogs` 로깅 드라이버를 사용하도록 구성된 Windows 태스크의 경우 컨테이너 인스턴스에 `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` 환경 변수도 설정해야 합니다. 다음 구문을 사용합니다.

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE",
 $TRUE, "Machine")
Initialize-ECSAgent -Cluster <cluster name> -EnableTaskIAMRole -LoggingDrivers
 '["json-file","awslogs"]'
</powershell>
```

Systems Manager 파라미터 생성

Systems Manager 콘솔을 사용하면 중요 데이터에 대한 Systems Manager Parameter Store 파라미터를 생성할 수 있습니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 Systems Manager 파라미터 생성(콘솔) 또는 Systems Manager 파라미터 생성(AWS CLI)을 참조하세요.

컨테이너 정의에 환경 변수 추가

태스크 정의의 컨테이너 정의 내에서 컨테이너에 설정할 환경 변수의 이름으로 `secrets`를 지정하여 컨테이너에 제공할 민감한 데이터가 들어있는 Systems Manager Parameter Store 파라미터의 전체 ARN을 지정합니다. 자세한 내용은 `secrets` 섹션을 참조하세요.

다음은 Systems Manager 파라미터 스토어 파라미터를 참조할 때 형식을 나타내는 태스크 정의의 조각입니다. Systems Manager 파라미터 스토어 파라미터가 현재 실행 중인 태스크와 동일한 리전에 있는 경우, 파라미터의 전체 ARN 또는 이름을 사용할 수 있습니다. 파라미터가 다른 리전에 있다면 전체 ARN을 지정합니다.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }]
  }]
}
```

환경 변수에 지정된 보안 암호를 사용하여 태스크 정의를 생성하는 방법에 대한 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

애플리케이션을 업데이트하여 프로그래밍 방식으로 Systems Manager Parameter Store 보안 암호 검색

Systems Manager Parameter Store 파라미터에 저장된 민감한 데이터를 검색하려면 AWS SDK 코드 예제 코드 라이브러리의 Code examples for Systems Manager using AWS SDKs를 참조하세요.

Amazon ECS 로깅 구성에 대한 보안 암호 전달

`logConfiguration`에서 `secretOptions` 파라미터를 사용하여 로깅에 사용되는 민감한 데이터를 전달할 수 있습니다.

Secrets Manager 또는 Systems Manager에 보안 암호를 저장할 수 있습니다.

Secrets Manager 사용

컨테이너 정의 내에서, `logConfiguration`을 지정할 때 컨테이너에 설정할 로그 드라이버 옵션의 이름과 컨테이너에 제공할 민감한 데이터가 들어있는 Secret Manager 암호의 전체 ARN을 사용하여

`secretOptions`를 지정할 수 있습니다. 시크릿 생성에 대한 자세한 내용은 Create an AWS Secrets Manager를 참조하세요.

다음은 Secrets Manager 암호를 참조할 때 형식을 나타내는 태스크 태스크의 조각입니다.

```
{
  "containerDefinitions": [{
    "logConfiguration": [{
      "logDriver": "splunk",
      "options": {
        "splunk-url": "https://your_splunk_instance:8088"
      },
      "secretOptions": [{
        "name": "splunk-token",
        "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-
AbCdEf"
      }]
    }]
  }]
}
```

컨테이너 정의에 환경 변수 추가

컨테이너 정의 내에서 컨테이너에 설정할 환경 변수의 이름으로 `secrets`(을)를 지정하여 컨테이너에 제공할 민감한 데이터가 들어있는 Systems Manager 파라미터 스토어 파라미터의 전체 ARN을 지정합니다. 자세한 내용은 `secrets` 섹션을 참조하세요.

다음은 Systems Manager 파라미터 스토어 파라미터를 참조할 때 형식을 나타내는 태스크 정의의 조각입니다. Systems Manager 파라미터 스토어 파라미터가 현재 실행 중인 태스크와 동일한 리전에 있는 경우, 파라미터의 전체 ARN 또는 이름을 사용할 수 있습니다. 파라미터가 다른 리전에 있다면 전체 ARN을 지정합니다.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }]
  }]
}
```

환경 변수에 지정된 보안 암호를 사용하여 태스크 정의를 생성하는 방법에 대한 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

Systems Manager 사용

로그 구성에 중요 데이터를 삽입할 수 있습니다. 컨테이너 정의 내에서, `logConfiguration`을 정의할 때 컨테이너에 설정할 로그 드라이버 옵션의 이름과 컨테이너에 제공할 민감한 데이터가 들어있는 Systems Manager 파라미터 스토어 파라미터의 전체 ARN을 사용하여 `secretOptions`를 지정할 수 있습니다.

> **Important**
>
> Systems Manager 파라미터 스토어 파라미터가 현재 실행 중인 태스크와 동일한 리전에 있는 경우, 파라미터의 전체 ARN 또는 이름을 사용할 수 있습니다. 파라미터가 다른 리전에 있다면 전체 ARN을 지정합니다.

다음은 Systems Manager 파라미터 스토어 파라미터를 참조할 때 형식을 나타내는 태스크 정의의 조각입니다.

```
{
  "containerDefinitions": [{
    "logConfiguration": [{
      "logDriver": "fluentd",
      "options": {
        "tag": "fluentd demo"
      },
      "secretOptions": [{
        "name": "fluentd-address",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter:/parameter_name"
      }]
    }]
  }]
}
```

Amazon ECS에서 Secrets Manager 암호를 사용하여 민감한 데이터 지정

Amazon ECS에서는 AWS Secrets Manager 암호에 민감한 데이터를 저장하고 컨테이너 정의에서 참조하는 방식으로 컨테이너에 민감한 데이터를 삽입할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.

Secrets Manager 보안 암호를 생성하고, Amazon ECS 태스크 정의에서 보안 암호를 참조한 다음, 보안 암호 내용을 표시하는 컨테이너 내부의 환경 변수를 쿼리하여 작동하는지 확인하는 방법을 알아보세요.

사전 조건

이 자습서에서는 다음 사전 조건이 충족되었다고 가정합니다.

- Amazon ECS 사용 설정의 단계가 완료되었습니다.
- 사용자에게는 Secrets Manager 및 Amazon ECS 리소스를 생성하는 데 필요한 IAM 권한이 있습니다.

1단계: Secrets Manager 보안 암호 생성

Secrets Manager 콘솔을 사용하여 민감한 데이터에 대한 암호를 생성할 수 있습니다. 이 자습서에서는 이후 컨테이너에서 참조할 기본 사용자 이름 및 암호를 저장하는 기본 보안을 생성합니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager 보안 암호 생성을 참조하세요.

이 보안 암호에 저장되는 키/값 페어는 자습서 마지막에 있는 컨테이너의 환경 변수 값입니다.

Secret ARN(보안 암호 ARN)을 저장하여 이후 단계의 작업 실행 IAM 정책 및 작업 정의에 참조합니다.

2단계: 태스크 실행 역할에 보안 암호 권한 추가

Amazon ECS가 Secrets Manager 보안 암호에서 민감한 데이터를 가져오려면 작업 실행 역할에 대한 보안 암호 권한을 보유해야 합니다. 자세한 내용은 Secrets Manager 또는 Systems Manager 권한 섹션을 참조하세요.

3단계: 태스크 정의 생성

Amazon ECS 콘솔을 사용하면 Secrets Manager 암호를 참조하는 태스크 정의를 생성할 수 있습니다.

암호를 지정하는 태스크 정의를 생성하려면

IAM 콘솔을 사용해 작업 실행 역할에 필요한 권한을 업데이트합니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 작업 정의를 선택합니다.
3. 새 태스크 정의 생성(Create new task definition), JSON으로 새 태스크 정의 생성(Create new task definition with JSON)을 선택합니다.

4. JSON 편집기 상자에 다음 태스크 정의 JSON 문자열을 입력하여 1단계에서 생성한 Secrets Manager 보안 암호의 전체 ARN과 2단계에서 업데이트한 작업 실행 역할을 지정했는지 확인합니다. 저장을 선택합니다.

5.

```
{
    "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "entryPoint": [
                "sh",
                "-c"
            ],
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
            ],
            "cpu": 10,
            "secrets": [
                {
                    "valueFrom":
 "arn:aws:secretsmanager:region:aws_account_id:secret:username_value",
                    "name": "username_value"
                }
            ],
            "memory": 300,
            "image": "public.ecr.aws/docker/library/httpd:2.4",
            "essential": true,
            "name": "ecs-secrets-container"
        }
    ],
    "family": "ecs-secrets-tutorial"
}
```

6. 생성(Create)을 선택합니다.

4단계: 클러스터 생성

Amazon ECS 콘솔을 사용해 태스크를 실행할 컨테이너 인스턴스가 있는 클러스터를 생성합니다. 기존 클러스터에 자습서를 위한 태스크 정의 인스턴스를 실행할 수 있는 가용 리소스가 있는 컨테이너 인스턴스가 적어도 하나 등록된 경우에는 다음 단계로 이동합니다.

이 자습서에서는 Amazon ECS 최적화 Amazon Linux 2 AMI를 사용하여 하나의 `t2.micro` 컨테이너 인스턴스로 클러스터를 생성합니다.

EC2 시작 유형의 클러스터를 생성하는 방법에 대한 자세한 내용은 the section called “Amazon EC2 시작 유형에 대한 클러스터 생성” 섹션을 참조하세요.

5단계: 태스크 실행

Amazon ECS 콘솔을 사용해 생성한 태스크 정의로 태스크를 실행할 수 있습니다. 이 자습서에서는 이전 단계에서 생성한 클러스터에서 EC2 시작 유형을 사용해 태스크를 실행합니다.

작업을 실행하는 방법에 대한 정보는 the section called “애플리케이션을 태스크로 실행” 섹션을 참조하세요.

6단계: 확인

모든 단계가 성공적으로 완료되었으며 다음 단계를 따라 컨테이너 내에서 생성한 환경 변수가 잘 생성되었는지 확인합니다.

생성한 환경 변수 확인

1. 컨테이너 인스턴스의 퍼블릭 IP 또는 DNS 주소를 찾습니다.
   a. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
   b. 탐색 창에서 클러스터를 선택한 다음, 생성한 클러스터를 선택합니다.
   c. Infrastructure(인프라)를 선택한 다음 컨테이너 인스턴스를 선택합니다.
   d. 인스턴스로 인스턴스의 Public IP(퍼블릭 IP) 또는 Public DNS(퍼블릭 DNS)를 기록합니다.
2. Mac OS 또는 Linux 컴퓨터를 사용 중인 경우, 다음 명령을 사용하여 인스턴스에 연결해 프라이빗 키 경로와 인스턴스의 퍼블릭 주소를 대체합니다.

```
$ ssh -i /path/to/my-key-pair.pem ec2-user@ec2-198-51-100-1.compute-1.amazonaws.com
```

Windows 컴퓨터 사용에 대한 자세한 내용은 Amazon EC2 사용 설명서의 PuTTY를 사용하여 Linux 인스턴스에 연결을 참조하세요.

> **Important**
>
> 인스턴스 연결 문제에 대한 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스에 연결 문제 해결을 참조하세요.

3. 인스턴스에서 실행되는 컨테이너 목록을 표시합니다. `ecs-secrets-tutorial` 컨테이너의 컨테이너 ID를 기록합니다.

```
docker ps
```

4. 이전 단계의 결과로 얻은 컨테이너 ID를 사용해 `ecs-secrets-tutorial` 컨테이너에 연결합니다.

```
docker exec -it container_ID /bin/bash
```

5. `echo` 명령을 사용해 환경 변수 값을 프린트합니다.

```
echo $username_value
```

자습서가 성공했다면 다음 결과가 표시되어야 합니다.

```
password_value
```

> **Note**
>
> 또한 `env`(또는 `printenv`) 명령을 사용해 컨테이너 내의 모든 환경 변수 목록을 표시할 수 있습니다.

7단계: 정리

이 자습서로 완료를 한 후에 사용하지 않는 리소스에 대해 요금이 발생하는 것을 방지하기 위해 연결된 리소스를 정리해야 합니다.

리소스 정리

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
4. 클러스터 삭제(Delete Cluster)를 선택합니다.
5. 확인 상자에 delete ***cluster name***을 입력한 다음 삭제를 선택합니다.
6. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
7. 탐색 창에서 역할을 선택합니다.
8. 역할 목록에서 `ecsTaskExecutionRole`을 검색하고 선택합니다.
9. Permissions(권한)를 선택한 다음 ECSSecretsTutorial 옆에 있는 X를 선택합니다. 제거를 선택합니다.
10. https://console.aws.amazon.com/secretsmanager/에서 Secrets Manager 콘솔을 엽니다.
11. 생성한 username_value 보안 암호를 선택하고 작업(Actions), 보안 암호 삭제(Delete secret)를 선택합니다.

Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터

태스크 정의는 태스크 패밀리, AWS Identity and Access Management(IAM) 태스크 역할, 네트워크 모드, 컨테이너 정의, 볼륨, 시작 유형 등의 개별 부분으로 분할됩니다. 태스크 정의에는 패밀리 및 컨테이너 정의가 필요합니다. 하지만 태스크 역할, 네트워크 모드, 볼륨 및 시작 유형은 선택 사항입니다.

JSON 파일에서 이러한 파라미터를 사용하여 태스크 정의를 구성할 수 있습니다.

다음은 Fargate 시작 유형의 각 태스크 정의 파라미터에 대한 자세한 설명입니다.

Family

`family`

유형: 문자열

필수 항목 여부: 예

태스크 정의를 등록할 때 패밀리를 지정합니다. 패밀리는 개정 번호를 사용하여 지정된 태스크 정의의 여러 버전에 대한 이름과 비슷합니다. 특정 패밀리에 등록된 첫 번째 태스크 정의에는 개정 번호 1이 부여되고 그 이후 등록된 태스크 정의에는 순차적으로 개정 번호가 부여됩니다.

시작 유형

태스크 정의를 등록할 때 Amazon ECS가 태스크 정의의 유효성을 검사해야 하는 시작 유형을 지정할 수 있습니다. 태스크 정의가 유효성을 지정한 호환성에 대해 검사하지 않으면 클라이언트 예외가 반환됩니다. 자세한 정보는 Amazon ECS 시작 유형을 참조하세요.

태스크 정의에서 다음 파라미터가 허용됩니다.

`requiresCompatibilities`

유형: 문자열 배열

필수 여부: 아니요

유효 값: `FARGATE`

태스크 정의의 유효성을 검사하기 위한 시작 유형입니다. 이를 통해 작업 정의에 사용되는 모든 파라미터가 시작 유형의 요구 사항을 충족하는지 확인할 수 있습니다.

태스크 역할

`taskRoleArn`

유형: 문자열

필수 여부: 아니요

태스크 정의를 등록할 때 태스크 권한의 컨테이너가 사용자 대신 연결된 정책에 지정된 AWS API를 호출하도록 허용하는 IAM 역할에 태스크 역할을 제공할 수 있습니다. 자세한 내용은 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.

태스크 실행 역할

`executionRoleArn`

유형: 문자열

필수 항목 여부: 조건부

Amazon ECS 컨테이너 에이전트에 사용자를 대신하여 AWS API를 호출할 권한을 부여하는 태스크 실행 역할의 Amazon 리소스 이름(ARN)입니다.

> **Note**
>
> 태스크의 요구 사항에 따라 태스크 실행 IAM 역할이 필요합니다. 자세한 정보는 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

네트워크 모드

`networkMode`

유형: 문자열

필수 항목 여부: 예

태스크의 컨테이너에 사용할 Docker 네트워킹 모드. Fargate에서 호스팅되는 Amazon ECS 작업의 경우 `awsvpc` 네트워크 모드가 필요합니다.

런타임 플랫폼

`operatingSystemFamily`

유형: 문자열

필수 항목 여부: 조건부

기본값: LINUX

이 파라미터는 Fargate에서 호스팅되는 Amazon ECS 태스크에 필요합니다.

태스크 정의를 등록할 때 운영 체제 제품군을 지정합니다.

유효한 값은 `LINUX`, `WINDOWS_SERVER_2025_FULL`, `WINDOWS_SERVER_2025_CORE`, `WINDOWS_SERVER_2022_FULL`, `WINDOWS_SERVER_2022_CORE`, `WINDOWS_SERVER_2019_FULL`, `WINDOWS_SERVER_2019_CORE`입니다.

서비스에서 사용되는 모든 태스크 정의는 이 파라미터에 대해 동일한 값을 가져야 합니다.

태스크 정의가 서비스의 일부인 경우 이 값은 서비스 `platformFamily` 값과 일치해야 합니다.

`cpuArchitecture`

유형: 문자열

필수 항목 여부: 조건부

기본값: X86_64

이 파라미터를 null로 두면 Fargate에 호스팅된 작업을 시작할 때 기본값이 자동으로 할당됩니다.

태스크 정의를 등록할 때 CPU 아키텍처를 지정합니다. 유효 값은 X86_64 및 ARM64입니다.

서비스에서 사용되는 모든 태스크 정의는 이 파라미터에 대해 동일한 값을 가져야 합니다.

Linux 태스크가 있는 경우 값을 ARM64로 설정할 수 있습니다. 자세한 내용은 the section called “64 비트 ARM 워크로드에 대한 작업 정의” 섹션을 참조하세요.

태스크 크기

태스크 정의를 등록할 때 태스크에 대해 사용할 CPU 및 메모리 총량을 지정할 수 있습니다. 이것은 컨테이너 정의 수준의 cpu 및 memory 값과는 구분됩니다. Fargate(Linux와 Windows 모두)에서 호스팅되는 태스크의 경우 이 필드는 필수이고 지원되는 cpu와 memory 모두에 대한 특정 값이 있습니다.

태스크 정의에서 다음 파라미터가 허용됩니다.

cpu

유형: 문자열

필수 항목 여부: 예

> **Note**
>
> 태스크 수준 CPU 및 메모리 파라미터는 필수이며 태스크가 실행되는 인스턴스 유형과 크기를 결정하는 데 사용됩니다. Windows 태스크의 경우 Windows에는 컨테이너 그룹에 집합 리소스 제한을 쉽게 적용할 수 있는 기본 메커니즘이 없으므로 런타임 시 이러한 값이 적용되지 않습니다. 리소스 제한을 적용하려면 Windows 컨테이너에 대한 컨테이너 수준 리소스를 사용하는 것이 좋습니다.

태스크에 대해 표시되는 CPU 단위의 하드 제한입니다. JSON 파일에서 CPU 값을 문자열(CPU 단위 또는 가상 CPU(vCPU))로 지정할 수 있습니다. 예를 들어 CPU 값을 1024(CPU 단위) 또는 1 vCPU(vCPU)로 지정할 수 있습니다. 태스크 정의를 등록할 때는 vCPU 값이 CPU 단위를 나타내는 정수로 변환됩니다.

이 필드는 필수이며 다음 값 중 하나를 사용해야 합니다. 이 필드에 따라 `memory` 파라미터에 지원되는 값의 범위가 결정됩니다. 유효한 태스크 레벨 CPU 및 메모리 조합은 아래 표와 같습니다.

CPU 값	메모리 값	AWS Fargate에 지원되는 운영 체제
256(.25 vCPU)	512MiB, 1GB, 2GB	Linux
512(.5 vCPU)	1GB, 2GB, 3GB, 4GB	Linux
1024(1 vCPU)	2GB, 3GB, 4GB, 5GB, 6GB, 7GB, 8GB	Linux, Windows
2048(2 vCPU)	4~16GB(1GB 증분)	Linux, Windows
4096(4 vCPU)	8~30GB(1GB 증분)	Linux, Windows
8192 (8 vCPU) **Note** 이 옵션은 Linux 플랫폼 `1.4.0` 이상이 필요합니다.	16~60GB(4GB 증분)	Linux
16384 (16vCPU) **Note** 이 옵션은 Linux 플랫폼 `1.4.0` 이상이 필요합니다.	32~120GB(8GB 증분)	Linux

`memory`

유형: 문자열

필수 항목 여부: 예

> **Note**
>
> 태스크 수준 CPU 및 메모리 파라미터는 필수이며 태스크가 실행되는 인스턴스 유형과 크기를 결정하는 데 사용됩니다. Windows 태스크의 경우 Windows에는 컨테이너 그룹에 집합 리소스 제한을 쉽게 적용할 수 있는 기본 메커니즘이 없으므로 런타임 시 이러한 값이 적용되지 않습니다. 리소스 제한을 적용하려면 Windows 컨테이너에 대한 컨테이너 수준 리소스를 사용하는 것이 좋습니다.

작업에 표시할 메모리의 하드 제한. 작업 정의에서 메모리 값을 문자열(메비바이트(MiB) 또는 기가바이트(GB) 단위)로 지정할 수 있습니다. 예를 들어 메모리 값을 3072(MiB) 또는 3 GB(GB)로 지정할 수 있습니다. 태스크 정의를 등록할 때는 GB 값이 MiB를 나타내는 정수로 변환됩니다.

이 필드는 필수이며 다음 값 중 하나를 사용해야 합니다. 이 필드에 따라 cpu 파라미터에 지원되는 값의 범위가 결정됩니다.

메모리 값(MiB 단위, GB와 대략적으로 동등한 값)	CPU 값	Fargate에 지원되는 운영 체제
512(0.5GB), 1,024(1GB), 2,048(2GB)	256(.25 vCPU)	Linux
1,024(1GB), 2,048(2GB), 3,072(3GB), 4,096(4GB)	512(.5 vCPU)	Linux
2,048(2GB), 3,072(3GB), 4,096(4GB), 5,120(5GB), 6,144(6GB), 7,168(7GB), 8,192(8GB)	1024(1 vCPU)	Linux, Windows
4,096(4GB)~16,384(16GB) (1,024(1GB) 증분)	2048(2 vCPU)	Linux, Windows
8,192(8GB)~30,720(30GB) (1,024(1GB) 증분)	4096(4 vCPU)	Linux, Windows
16~60GB(4GB 증분)	8192 (8 vCPU)	Linux

메모리 값(MiB 단위, GB와 대략적으로 동등한 값)	CPU 값	Fargate에 지원되는 운영 체제
Note 이 옵션은 Linux 플랫폼 1.4.0 이상이 필요합니다.		
32~120GB(8GB 증분) Note 이 옵션은 Linux 플랫폼 1.4.0 이상이 필요합니다.	16384 (16vCPU)	Linux

컨테이너 정의

작업 정의를 등록할 때 컨테이너 인스턴스의 Docker 대몬(daemon)으로 전달되는 컨테이너 정의의 목록을 지정해야 합니다. 컨테이너 정의에서 다음 파라미터가 허용됩니다.

주제

- 스탠다드 컨테이너 정의 파라미터
- 고급 컨테이너 정의 파라미터
- 기타 컨테이너 정의 파라미터

스탠다드 컨테이너 정의 파라미터

다음 태스크 정의 파라미터는 대부분의 컨테이너 정의에서 필요하거나 사용됩니다.

주제

- 명칭
- 이미지
- 메모리

- 포트 매핑
- 프라이빗 리포지토리 보안 인증

명칭

`name`

유형: 문자열

필수 항목 여부: 예

컨테이너의 이름. 최대 255개의 문자(대문자 및 소문자), 숫자, 하이픈 및 밑줄이 허용됩니다. 여러 컨테이너를 한 태스크 정의에 연결하는 경우, 한 컨테이너의 `name`을 다른 컨테이너의 `links`에 입력할 수 있습니다. 이는 컨테이너를 연결하기 위한 것입니다.

이미지

`image`

유형: 문자열

필수 항목 여부: 예

컨테이너를 시작하는 데 사용되는 이미지입니다. 이 문자열은 Docker 대몬(daemon)으로 직접 전달됩니다. 기본적으로 Docker Hub 레지스트리 내 이미지는 사용 가능합니다. *repository-url*/*image*:*tag* 또는 *repository-url*/*image*@*digest*를 사용하여 다른 리포지토리를 지정할 수도 있습니다. 최대 255개의 문자(대문자 및 소문자), 숫자, 하이픈, 밑줄, 콜론, 마침표, 슬래시 및 부호가 허용됩니다. 이 파라미터는 Docker 컨테이너 생성 명령에 있는 `Image` 및 docker run 명령의 `IMAGE` 파라미터에 매핑됩니다.

- 새 태스크가 시작될 때 Amazon ECS 컨테이너 에이전트가 컨테이너에서 사용할 지정 이미지와 태그의 최신 버전을 끌어옵니다. 그러나 이미 실행 중인 태스크에는 추후 리포지토리 이미지 업데이트가 전파되지 않습니다.
- 태스크 정의의 이미지 경로에 태그 또는 다이제스트를 지정하지 않으면 Amazon ECS 컨테이너 에이전트가 지정된 이미지의 최신 버전을 가져옵니다.
- 그러나 이미 실행 중인 태스크에는 추후 리포지토리 이미지 업데이트가 전파되지 않습니다.
- 프라이빗 리포지토리의 이미지는 지원되지 않습니다. 자세한 정보는 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용을 참조하세요.

- Amazon ECR 리포지토리의 이미지는 전체 `registry/repository:tag` 또는 `registry/repository@digest` 명명 규칙(예: *aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*:*latest* 또는 *aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*@*sha256:94afd1f2e64d908bc90dbca0035a5b567EXAMPLE*)을 사용하여 지정할 수 있습니다.
- Docker Hub 공식 리포지토리 안의 이미지는 단일 이름을 사용합니다(예: ubuntu 또는 mongo).
- Docker Hub 다른 리포지토리에 저장된 이미지는 조직 이름으로 한정됩니다(예: amazon/amazon-ecs-agent).
- 다른 온라인 리포지토리 안의 이미지는 도메인 이름을 사용하여 추가로 한정됩니다(예: quay.io/assemblyline/ubuntu).

`versionConsistency`

유형: 문자열

유효한 값: `enabled|disabled`

필수 여부: 아니요

컨테이너 정의에 제공된 컨테이너 이미지 태그가 Amazon ECS에서 이미지 다이제스트로 확인되는지 여부를 지정합니다. 기본적으로 이 동작은 `enabled`입니다. 컨테이너의 값을 `disabled`로 설정하면 Amazon ECS에서는 컨테이너 이미지 태그가 다이제스트로 확인되지 않고 컨테이너 정의에 지정된 원본 이미지 URI가 배포에 사용됩니다. 컨테이너 이미지 확인에 대한 자세한 내용은 컨테이너 이미지 확인 단원을 참조하세요.

메모리

`memory`

유형: 정수

필수 여부: 아니요

컨테이너에 표시할 메모리의 양(MiB)입니다. 컨테이너가 여기서 지정된 메모리를 초과하려 하면 해당 컨테이너가 중지됩니다. 태스크 내 모든 컨테이너에 대해 예약된 총 메모리 양은 태스크 `memory` 값(지정된 경우)보다 작아야 합니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `Memory` 및 docker run에 대한 `--memory` 옵션에 매핑됩니다.

Docker 20.10.0 이상의 대몬(daemon)은 컨테이너용으로 최소 6MiB의 메모리를 남겨둡니다. 따라서 컨테이너에 6MiB 미만의 메모리를 지정하지 마세요.

Docker 19.03.13-ce 이하의 대몬(daemon)은 컨테이너용으로 최소 4MiB의 메모리를 남겨둡니다. 따라서 컨테이너에 4MiB 미만의 메모리를 지정하지 마세요.

> **Note**
>
> 특정 인스턴스 유형에 대해 태스크에 가능한 한 많은 메모리를 제공하여 리소스 사용률을 최대화하려는 경우 Amazon ECS Linux 컨테이너 인스턴스 메모리 예약 섹션을 참조하세요.

`memoryReservation`

유형: 정수

필수 여부: 아니요

컨테이너용으로 예약할 메모리의 소프트 제한(MiB)입니다. 시스템 메모리가 경합하는 경우 Docker는 컨테이너 메모리를 이 소프트 제한으로 유지하려고 합니다. 하지만 필요한 경우 컨테이너에서 더 많은 메모리를 사용할 수 있습니다. 컨테이너는 `memory` 파라미터에 지정된 하드 한도까지(해당하는 경우) 또는 컨테이너 인스턴스의 모든 가용 메모리(둘 중 먼저 발생하는 것)를 사용할 수 있습니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `MemoryReservation` 및 docker run에 대한 `--memory-reservation` 옵션에 매핑됩니다.

태스크 레벨 메모리 값이 지정되지 않은 경우 컨테이너 정의의 `memory` 또는 `memoryReservation`에 대해 하나 또는 둘 다에 0이 아닌 정수를 지정해야 합니다. 둘 모두 지정하는 경우 `memory`가 `memoryReservation`보다 커야 합니다. `memoryReservation`을 지정하는 경우 컨테이너가 배치된 컨테이너 인스턴스의 가용 메모리 리소스에서 해당 값이 차감됩니다. 그렇지 않으면 `memory` 값이 사용됩니다.

예를 들어 컨테이너가 통상적으로 128MiB의 메모리를 사용하지만 경우에 따라 잠깐 동안 사용량이 256MiB까지 급증한다고 가정합니다. 이 경우 `memoryReservation`을 128MiB로 설정하고 `memory` 하드 제한을 300MiB로 설정할 수 있습니다. 이 구성은 컨테이너가 컨테이너 인스턴스의 잔여 리소스 중 128MiB의 메모리만 예약하도록 허용합니다. 동시에, 이 구성을 사용하여 필요할 때 컨테이너가 더 많은 메모리 리소스를 사용하도록 허용할 수도 있습니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

Docker 20.10.0 이상의 대몬(daemon)은 컨테이너용으로 최소 6MiB의 메모리를 남겨둡니다. 따라서 컨테이너에 6MiB 미만의 메모리를 지정하지 마세요.

Docker 19.03.13-ce 이하의 대몬(daemon)은 컨테이너용으로 최소 4MiB의 메모리를 남겨둡니다. 따라서 컨테이너에 4MiB 미만의 메모리를 지정하지 마세요.

> **Note**
>
> 특정 인스턴스 유형에 대해 태스크에 가능한 한 많은 메모리를 제공하여 리소스 사용률을 최대화하려는 경우 Amazon ECS Linux 컨테이너 인스턴스 메모리 예약 섹션을 참조하세요.

포트 매핑

`portMappings`

유형: 객체 배열

필수 여부: 아니요

포트 매핑은 컨테이너의 네트워크 포트를 외부에 노출시켜 클라이언트가 애플리케이션에 액세스할 수 있도록 합니다. 이 파라미터는 동일한 태스크 내에서 컨테이너 간 통신에도 사용됩니다.

`awsvpc` 네트워크 모드를 사용하는 태스크 정의의 경우 `containerPort`만 지정합니다. `hostPort`는 항상 무시되며 컨테이너 포트는 호스트에서 임의의 높은 번호의 포트에 자동으로 매핑됩니다.

Windows에서의 포트 매핑은 `localhost`가 아니라 `NetNAT` 게이트웨이 주소를 사용합니다. Windows에서의 포트 매핑에는 루프백이 없으므로 호스트 자체에서 컨테이너의 매핑된 포트에 액세스할 수 없습니다.

이 파라미터의 필드 대부분(`containerPort`, `hostPort`, `protocol` 포함)은 Docker 컨테이너 생성 명령에 있는 `PortBindings`와 docker run에 대한 `--publish` 옵션에 매핑됩니다. 태스크 정의

의 네트워크 모드가 host로 설정되어 있으면 호스트 포트가 정의되지 않은 상태이거나 호스트 포트가 포트 매핑의 컨테이너 포트와 일치해야 합니다.

> **Note**
>
> 태스크가 RUNNING 상태에 도달한 후에는 다음 위치에서 수동 및 자동 호스트/컨테이너 포트 할당을 확인할 수 있습니다.
>
> - 콘솔: 선택한 태스크에 대한 컨테이너 설명의 네트워크 바인딩 섹션.
> - AWS CLI: describe-tasks 명령 출력의 networkBindings 섹션.
> - API: DescribeTasks 응답.
> - 메타데이터: 작업 메타데이터 엔드포인트.

`appProtocol`

유형: 문자열

필수 여부: 아니요

포트 매핑에 사용되는 애플리케이션 프로토콜입니다. 이 파라미터는 Service Connect에만 적용됩니다. 애플리케이션에서 사용하는 프로토콜과 일치하도록 이 파라미터를 설정하는 것이 좋습니다. 이 파라미터를 설정하면 Amazon ECS가 서비스 연결 프록시에 프로토콜별 연결 처리를 추가합니다. 이 파라미터를 설정하면 Amazon ECS가 Amazon ECS 콘솔과 CloudWatch에 프로토콜별 원격 분석을 추가합니다.

이 파라미터의 값을 설정하지 않으면 TCP가 사용됩니다. 그러나 Amazon ECS는 TCP에 대한 프로토콜별 원격 측정을 추가하지 않습니다.

자세한 내용은 the section called “ Service Connect” 섹션을 참조하세요.

유효한 프로토콜 값: `"http" | "http2" | "grpc"`

`containerPort`

유형: 정수

필수 항목 여부: 예(portMappings 사용 시)

사용자 지정 또는 자동 할당된 호스트 포트에 바인딩되는 컨테이너 포트 번호입니다.

awsvpc 네트워크 모드를 사용하는 태스크의 경우 containerPort를 사용하여 노출된 포트를 지정합니다.

Fargate의 Windows 컨테이너의 경우 포트 3150은 `containerPort`에 사용할 수 없습니다. 이 포트는 예약되어 있기 때문입니다.

`containerPortRange`

유형: 문자열

필수 여부: 아니요

동적으로 매핑된 호스트 포트 범위에 바인딩된 컨테이너의 포트 번호 범위입니다.

이 파라미터는 `register-task-definition` API를 사용해서만 설정할 수 있습니다. 옵션은 `portMappings` 파라미터에서 사용할 수 있습니다. 자세한 정보는 AWS Command Line Interface 참조의 register-task-definition을 참조하세요.

`containerPortRange` 지정 시 다음 규칙이 적용됩니다.

- `awsvpc` 네트워크 모드를 사용해야 합니다.
- 이 파라미터는 Linux 및 Windows 운영 체제 모두에서 사용할 수 있습니다.
- 컨테이너 인스턴스의 컨테이너 에이전트 버전은 1.67.0 이상이어야 하고 `ecs-init` 패키지의 버전은 1.67.0-1 이상이어야 합니다.
- 컨테이너당 최대 100개의 포트 범위를 지정할 수 있습니다.
- `hostPortRange`는 지정하지 않습니다. `hostPortRange`의 값은 다음과 같이 설정됩니다.
  - `awsvpc` 네트워크 모드를 사용하는 작업에 있는 컨테이너의 경우 `hostPort`는 `containerPort`와 동일한 값으로 설정됩니다. 이는 정적 매핑 전략입니다.
- `containerPortRange`의 유효한 값은 1~65,535입니다.
- 포트는 컨테이너당 하나의 포트 매핑에만 포함될 수 있습니다.
- 중첩되는 포트 범위는 지정할 수 없습니다.
- 범위의 첫 번째 포트는 범위의 마지막 포트보다 작아야 합니다.
- Docker에서는 포트 수가 많을 때 Docker 대몬(daemon) 구성 파일에서 docker-proxy를 끌 것을 권장합니다.

  자세한 내용은 GitHub의 Issue #11185를 참조하세요.

  Docker 대몬(daemon) 구성 파일에서 docker-proxy를 끄는 방법에 대한 자세한 내용은 Amazon ECS 개발자 안내서의 Docker daemon을 참조하세요.

DescribeTasks를 직접적으로 호출하여 컨테이너 포트에 바인딩된 호스트 포트인 `hostPortRange`를 볼 수 있습니다.

포트 범위는 EventBridge로 전송되는 Amazon ECS 작업 이벤트에 포함되지 않습니다. 자세한 내용은 the section called “EventBridge를 사용하여 Amazon ECS 오류에 대한 응답 자동화” 섹션을 참조하세요.

`hostPortRange`

유형: 문자열

필수 여부: 아니요

네트워크 바인딩에 사용되는 호스트의 포트 번호 범위입니다. 이 값은 Docker가 할당하고 Amazon ECS 에이전트가 전달합니다.

`hostPort`

유형: 정수

필수 여부: 아니요

컨테이너용으로 예약할 컨테이너 인스턴스 포트 번호입니다.

`hostPort`는 비워 두거나 `containerPort`와 같은 값을 지정할 수 있습니다.

기본 임시 포트 범위 Docker 1.6.0 버전 이상은 인스턴스의 `/proc/sys/net/ipv4/ip_local_port_range`에 나열됩니다. 이 커널 파라미터를 사용할 수 없을 경우 기본 휘발성 포트 범위 `49153-65535`가 사용됩니다. 휘발성 포트 범위에서 호스트 포트를 지정하지 마세요. 자동 할당을 위해 예약되어 있기 때문입니다. 일반적으로 32768 미만의 포트는 임시 포트 범위를 벗어납니다.

기본 예약 포트는 SSH용 22, Docker 포트 2375 및 2376, Amazon ECS 컨테이너 에이전트 포트 `51678-51680`입니다. 실행 중인 작업에 대해 기존에 사용자 지정된 호스트 포트도 해당 작업이 실행되는 동안 예약됩니다. 작업이 중지하면 호스트 포트는 해제됩니다. 현재 예약된 포트는 describe-container-instances 출력의 `remainingResources`에 표시됩니다. 하나의 컨테이너 인스턴스는 한 번에 최대 100개의 예약 포트(기본 예약 포트 포함)를 가질 수 있습니다. 자동 할당된 포트는 예약 포트 할당량 100개에 포함되지 않습니다.

`name`

유형: 문자열

필수: 아니요, 서비스에서 Service Connect와 VPC Lattice를 구성하는 데 필요

포트 매핑에 사용되는 이름입니다. 이 파라미터는 Service Connect와 VPC Lattice에만 적용됩니다. 이 파라미터는 서비스의 Service Connect와 VPC Lattice 구성에서 사용하는 이름입니다.

자세한 내용은 Service Connect를 사용하여 짧은 이름으로 Amazon ECS 서비스 연결 섹션을 참조하세요.

다음 예제에서는 Service Connect와 VPC Lattice의 필수 필드를 모두 사용합니다.

```
"portMappings": [
    {
        "name": string,
        "containerPort": integer
    }
]
```

`protocol`

유형: 문자열

필수 여부: 아니요

포트 매핑에 사용되는 프로토콜입니다. 유효 값은 `tcp`및 `udp`입니다. 기본값은 `tcp`입니다.

> **Important**
>
> Service Connect에는 `tcp`만 지원됩니다. 이 필드가 설정되지 않은 경우 `tcp`로 암시됩니다.

호스트 포트를 지정하는 경우 다음 구문을 사용합니다.

```
"portMappings": [
    {
        "containerPort": integer,
        "hostPort": integer
    }
    ...
]
```

호스트 포트가 자동 할당되도록 하려면 다음 구문을 사용합니다.

```
"portMappings": [
    {
```

```
        "containerPort": integer
    }
    ...
]
```

프라이빗 리포지토리 보안 인증

`repositoryCredentials`

유형: RepositoryCredentials 객체

필수 여부: 아니요

프라이빗 레지스트리 인증을 위한 리포지토리 보안 인증입니다.

자세한 내용은 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용 섹션을 참조하세요.

`credentialsParameter`

유형: 문자열

필수 항목 여부: 예(`repositoryCredentials` 사용 시)

프라이빗 리포지토리 자격 증명이 포함된 암호의 Amazon 리소스 이름(ARN)입니다.

자세한 내용은 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용 섹션을 참조하세요.

> **Note**
>
> Amazon ECS API, AWS CLI 또는 AWS SDK를 사용할 때 시작하는 작업과 같은 리전에 보안 암호가 있는 경우 보안 암호의 전체 ARN 또는 이름을 사용할 수 있습니다. AWS Management Console을 사용하는 경우 보안 암호의 전체 ARN을 지정해야 합니다.

다음은 필요한 파라미터를 나타낸 태스크 정의의 예제 조각입니다.

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
```

```
        }
    }
]
```

고급 컨테이너 정의 파라미터

다음의 고급 컨테이너 정의 파라미터는 Amazon ECS 컨테이너 인스턴스에서 컨테이너를 시작하는 데 사용된 docker run 명령에 확장된 기능을 제공합니다.

주제

- 재시작 정책
- 상태 확인
- 환경
- 네트워크 설정
- 스토리지 및 로깅
- 보안
- 리소스 제한
- Docker 레이블

재시작 정책

`restartPolicy`

컨테이너 재시작 정책 및 관련 구성 파라미터. 컨테이너의 재시작 정책을 설정하면 작업을 교체할 필요 없이 Amazon ECS에서 컨테이너를 다시 시작할 수 있습니다. 자세한 내용은 컨테이너 재시작 정책이 있는 Amazon ECS 작업의 개별 컨테이너 재시작 섹션을 참조하세요.

`enabled`

유형: 부울

필수 여부: 예

재시작 정책이 컨테이너에 대해 활성화되는지 여부를 지정합니다.

`ignoredExitCodes`

유형: 정수 배열

필수 여부: 아니요

Amazon ECS가 재시작을 시도하지 않고 무시하는 종료 코드 목록입니다. 최대 50개의 컨테이너 종료 코드를 지정할 수 있습니다. 기본적으로 Amazon ECS는 종료 코드를 무시하지 않습니다.

`restartAttemptPeriod`

유형: 정수

필수 여부: 아니요

재시작을 시도하기 전에 컨테이너를 실행해야 하는 시간(초)입니다. 컨테이너는 `restartAttemptPeriod`초 당 한 번만 재시작할 수 있습니다. 이 기간 동안 컨테이너를 실행할 수 없고 일찍 종료되면 다시 시작되지 않습니다. 최소 60초의 `restartAttemptPeriod`와 최대 1,800초의 `restartAttemptPeriod`를 설정할 수 있습니다. 기본적으로 컨테이너는 300초 동안 실행되어야 재시작할 수 있습니다.

상태 확인

`healthCheck`

해당 컨테이너에 대한 컨테이너 상태 확인 명령 및 연결된 구성 파라미터. 자세한 내용은 컨테이너 상태 확인을 사용하여 Amazon ECS 작업 상태 확인 섹션을 참조하세요.

`command`

상태가 정상인지를 확인하기 위해 컨테이너가 실행하는 명령을 나타내는 문자열 배열입니다. 명령 인수를 직접 실행하려면 문자열 어레이가 `CMD`으로 시작하면 되고, 컨테이너의 기본 셸에서 명령을 실행하려면 `CMD-SHELL`로 시작하면 됩니다. 둘 다 지정하지 않으면 `CMD`가 사용됩니다.

AWS Management Console에서 작업 정의를 등록할 때는 쉼표로 구분된 명령 목록을 사용합니다. 이러한 명령은 작업 정의가 생성된 후 문자열로 변환됩니다. 다음은 상태 확인에 대한 입력 예제입니다.

```
CMD-SHELL, curl -f http://localhost/ || exit 1
```

AWS Management Console JSON 패널, AWS CLI 또는 API를 사용하여 작업 정의를 등록할 때는 명령 목록을 괄호로 묶어야 합니다. 다음은 상태 확인에 대한 입력 예제입니다.

```
[ "CMD-SHELL", "curl -f http://localhost/ || exit 1" ]
```

stderr 출력이 없는 종료 코드 0은 성공을 나타내고, 0이 아닌 종료 코드는 실패를 나타냅니다.

`interval`

각 상태 확인 간의 시간(초). 5초에서 300초 사이를 지정할 수 있습니다. 기본값은 30초입니다.

`timeout`

실패로 간주되기 전에 상태 확인이 성공하기까지의 대기 시간(초)입니다. 2초에서 60초 사이를 지정할 수 있습니다. 기본 값은 5초입니다.

`retries`

컨테이너 상태가 비정상이라고 간주되기 전에 실패한 상태 확인을 재시도하는 횟수입니다. 1에서 10 사이의 재시도 횟수를 지정할 수 있습니다. 기본값은 3회 재시도입니다.

`startPeriod`

실패한 상태 확인이 최대 재시도 횟수에 포함되기 전에 컨테이너에 부트스트랩 시간이 제공되는 유예 기간 옵션입니다. 0~300초 범위의 값을 지정할 수 있습니다. 기본적으로 `startPeriod`는 비활성화되어 있습니다.

상태 검사가 `startPeriod` 내에 성공한 경우 컨테이너는 정상으로 간주되고 모든 후속 실패는 최대 재시도 횟수에 포함됩니다.

환경

`cpu`

유형: 정수

필수 여부: 아니요

Amazon ECS 컨테이너 에이전트가 컨테이너에 대해 예약하는 cpu 단위입니다. Linux에서 이 파라미터는 컨테이너 만들기 섹션의 `CpuShares`에 매핑됩니다.

이 필드는 Fargate 시작 유형을 사용하는 태스크의 경우 선택 사항입니다. 태스크 내 모든 컨테이너에 대해 예약된 총 CPU 양은 태스크 수준 cpu 값보다 작아야 합니다.

Linux 컨테이너는 할당된 양과 동일한 비율로, 컨테이너 인스턴스의 다른 컨테이너와 할당되지 않은 CPU 단위를 공유합니다. 예를 들어 싱글 코어 인스턴스 유형에서 단일 컨테이너 작업을 컨테이너에 512개 CPU 단위를 지정하고 실행한다고 가정해 보겠습니다. 또한 이 작업은 컨테이너 인스턴

스에서 실행되는 유일한 작업입니다. 이 예에서 컨테이너는 언제라도 1,024 CPU 단위 전부를 사용할 수 있습니다. 하지만, 해당 컨테이너 인스턴스에서 동일한 태스크의 다른 사본을 시작했다고 가정합니다. 각 작업에는 필요한 경우 최소 512개의 CPU 단위가 보장됩니다. 마찬가지로 다른 컨테이너가 나머지 CPU를 사용하지 않을 경우 각 컨테이너의 CPU 사용량이 높아질 수 있습니다. 하지만 두 작업이 항상 100% 활성 상태인 경우 CPU 단위는 512개로 제한됩니다.

Linux 컨테이너 인스턴스에서, 컨테이너 인스턴스의 Docker 대몬(daemon)은 CPU 값을 사용하여 실행 컨테이너의 상대적인 CPU 공유 비율을 계산합니다. Linux 커널이 허용하는 최소 유효 CPU 공유 값은 2이고, Linux 커널이 허용하는 최대 유효 CPU 공유 값은 262144입니다. 하지만 CPU 파라미터는 필수 항목이 아니며 컨테이너 정의에서 2 미만이나 262144를 초과하는 CPU 값을 사용할 수 있습니다. CPU 값이 2 미만(null 포함)이거나 262144를 초과하는 경우 동작은 Amazon ECS 컨테이너 에이전트 버전에 따라 달라집니다.

Windows 컨테이너 인스턴스에서 CPU 할당량은 절대 할당량으로 적용됩니다. Windows 컨테이너는 작업 정의에 정의된 지정된 양의 CPU에만 액세스할 수 있습니다. Null 또는 0의 CPU 값이 Docker에 `0`으로 전달되고, Windows는 이 값을 1개 CPU의 1%로 해석합니다.

추가 예제는 How Amazon ECS manages CPU and memory resources를 참조하세요.

`gpu`

이 파라미터는 Fargate에서 호스팅되는 컨테이너에 대해서는 지원되지 않습니다.

유형: ResourceRequirement 객체

필수 여부: 아니요

Amazon ECS 컨테이너 에이전트가 컨테이너에 대해 예약하는 실제 GPUs 수입니다. 태스크의 모든 컨테이너에 예약된 GPU 수가 태스크가 실행되는 컨테이너 인스턴스에서 사용할 수 있는 GPU 수를 초과하면 안 됩니다. 자세한 내용은 GPU 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요.

`Elastic Inference accelerator`

이 파라미터는 Fargate에서 호스팅되는 컨테이너에 대해서는 지원되지 않습니다.

유형: ResourceRequirement 객체

필수 여부: 아니요

`InferenceAccelerator` 유형의 경우 `value`는 태스크 정의에 지정된 `InferenceAccelerator`의 `deviceName`과 일치합니다. 자세한 내용은 the section called "Elastic Inference 액셀러레이터 이름" 섹션을 참조하세요.

`essential`

유형: 부울

필수 여부: 아니요

컨테이너의 `essential` 파라미터가 `true`로 표시되어 있고 해당 컨테이너가 어떤 이유로든 실패 또는 중지하는 경우를 가정해 보겠습니다. 이 경우 작업에 포함된 다른 모든 컨테이너도 중지합니다. 컨테이너의 `essential` 파라미터가 `false`로 표시된 경우에는 해당 컨테이너의 실패가 태스크의 나머지 컨테이너에 영향을 주지 않습니다. 이 파라미터가 생략된 경우 컨테이너가 필수로 간주됩니다.

모든 태스크에는 하나 이상의 필수 컨테이너가 있어야 합니다. 여러 컨테이너로 구성된 애플리케이션이 있다고 가정해 보겠습니다. 이 경우, 공통 용도로 사용되는 컨테이너를 구성 요소로 그룹화하고, 다른 구성 요소를 여러 작업 정의로 분리합니다. 자세한 내용은 Amazon ECS에 대한 애플리케이션 설계 섹션을 참조하세요.

```
"essential": true|false
```

`entryPoint`

> **Important**
>
> 초기 버전의 Amazon ECS 컨테이너 에이전트는 `entryPoint` 파라미터를 적절히 처리하지 못합니다. `entryPoint`를 사용하는 데 문제가 있을 경우 컨테이너 에이전트를 업데이트하거나 명령 및 인수를 `command` 배열 항목으로 입력하세요.

유형: 문자열 배열

필수 여부: 아니요

컨테이너로 전달되는 진입점입니다.

```
"entryPoint": ["string", ...]
```

`command`

유형: 문자열 배열

필수 여부: 아니요

컨테이너로 전달되는 명령입니다. 이 파라미터는 컨테이너 생성 명령의 Cmd 및 docker run에 대한 COMMAND 파라미터에 매핑됩니다. 여러 인수가 있는 경우 각 인수는 배열에서 각각 분리된 문자열이어야 합니다.

```
"command": ["string", ...]
```

`workingDirectory`

유형: 문자열

필수 여부: 아니요

컨테이너에서 명령을 실행할 태스크 디렉터리입니다. 이 파라미터는 Docker 원격 API(Docker Remote API)의 컨테이너 생성(Create a container) 섹션에 있는 `WorkingDir`(와)과 docker run에 대한 `--workdir` 옵션에 매핑됩니다.

```
"workingDirectory": "string"
```

`environmentFiles`

Fargate의 Windows 컨테이너에서는 사용할 수 없습니다.

유형: 객체 배열

필수 여부: 아니요

컨테이너로 전달할 환경 변수를 포함하는 파일 목록입니다. 이 파라미터는 docker run 명령에 대한 `--env-file` 옵션에 매핑됩니다.

최대 10개의 환경 파일을 지정할 수 있습니다. 파일의 확장명은 `.env`여야 합니다. 환경 파일의 각 줄에는 `VARIABLE=VALUE` 형식의 환경 변수를 포함합니다. #으로 시작하는 줄은 주석으로 처리되며 무시됩니다.

컨테이너 정의에 지정된 개별 환경 변수가 있는 경우 환경 파일 내에 포함된 변수보다 우선합니다. 동일한 변수를 포함하는 여러 환경 파일이 지정된 경우 위에서 아래로 처리됩니다. 고유한 변수 이름을 사용하는 것이 좋습니다. 자세한 정보는 개별 환경 변수를 Amazon ECS 컨테이너로 전달을 참조하세요.

`value`

유형: 문자열

필수 항목 여부: 예

환경 변수 파일을 포함하는 Amazon S3 객체의 Amazon 리소스 이름(ARN)입니다.

`type`

유형: 문자열

필수 항목 여부: 예

사용할 파일 유형입니다. 지원되는 유일한 값은 `s3`입니다.

`environment`

유형: 객체 배열

필수 여부: 아니요

컨테이너로 전달할 환경 변수입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `Env` 및 docker run 명령에 대한 `--env` 옵션에 매핑됩니다.

> **Important**
> 자격 증명 데이터와 같은 민감한 정보에 대해서는 일반 텍스트 환경 변수를 사용하지 않는 것이 좋습니다.

`name`

유형: 문자열

필수 항목 여부: `environment` 사용 시, 예

환경 변수의 이름입니다.

`value`

유형: 문자열

필수 항목 여부: `environment` 사용 시, 예

환경 변수의 값입니다.

```
"environment" : [
    { "name" : "string", "value" : "string" },
```

```
    { "name" : "string", "value" : "string" }
]
```

`secrets`

유형: 객체 배열

필수 여부: 아니요

컨테이너에 공개할 보안 암호를 나타내는 객체입니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.

`name`

유형: 문자열

필수 항목 여부: 예

컨테이너에서 환경 변수로 설정할 값입니다.

`valueFrom`

유형: 문자열

필수 항목 여부: 예

컨테이너에 노출될 암호입니다. 지원되는 값은 AWS Secrets Manager 암호의 전체 ARN이거나 혹은 AWS Systems Manager 파라미터 스토어 내 파라미터의 전체 Amazon 리소스 이름(ARN)입니다.

> **Note**
>
> Systems Manager Parameter Store 파라미터 또는 Secrets Manager 파라미터가 현재 실행 중인 작업과 동일한 AWS 리전에 있을 경우 해당 암호의 전체 ARN 또는 이름을 사용할 수 있습니다. 파라미터가 다른 리전에 있다면 전체 ARN을 지정해야 합니다.

```
"secrets": [
    {
        "name": "environment_variable_name",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }
]
```

네트워크 설정

`disableNetworking`

Fargate에서 실행되는 태스크에는 이 파라미터가 지원되지 않습니다.

유형: 부울

필수 여부: 아니요

이 파라미터가 true일 경우 컨테이너 내에서 네트워킹이 해제됩니다.

기본값은 `false`입니다.

```
"disableNetworking": true|false
```

`links`

이 파라미터는 `awsvpc` 네트워크 모드를 사용하는 작업에 대해서는 지원되지 않습니다.

유형: 문자열 배열

필수 여부: 아니요

`link` 파라미터는 포트 매핑 필요 없이 컨테이너가 서로 통신하도록 허용합니다. 이 파라미터는 작업 정의의 네트워크 모드가 `bridge`로 설정된 경우에만 지원됩니다. `name:internalName` 구성은 Docker 링크의 `name:alias`와 유사합니다. 최대 255개의 문자(대문자 및 소문자), 숫자, 하이픈 및 밑줄이 허용됩니다.

> **Important**
>
> 동일한 컨테이너 인스턴스에 배치된 컨테이너는 링크 또는 호스트 포트 매핑 없이도 서로 통신할 수 있습니다. 컨테이너 인스턴스에서 네트워크 격리는 보안 그룹 및 VPC 설정에 의해 제어됩니다.

```
"links": ["name:internalName", ...]
```

`hostname`

유형: 문자열

필수 여부: 아니요

컨테이너에 사용할 호스트 이름입니다. 이 파라미터는 Docker 컨테이너 생성의 Hostname 및 docker run에 대한 --hostname 옵션에 매핑됩니다.

> **Note**
>
> awsvpc 네트워크 모드를 사용 중인 경우에는 hostname 파라미터가 지원되지 않습니다.

```
"hostname": "string"
```

dnsServers

Fargate에서 실행되는 태스크에는 지원되지 않습니다.

유형: 문자열 배열

필수 여부: 아니요

컨테이너에 제공되는 DNS 서버의 목록입니다.

```
"dnsServers": ["string", ...]
```

extraHosts

이 파라미터는 awsvpc 네트워크 모드를 사용하는 작업에 대해서는 지원되지 않습니다.

유형: 객체 배열

필수 여부: 아니요

컨테이너의 /etc/hosts 파일에 추가할 호스트 이름 및 IP 주소 매핑의 목록입니다.

이 파라미터는 Docker 컨테이너 생성 명령의 ExtraHosts 및 docker run에 대한 --add-host 옵션에 매핑됩니다.

```
"extraHosts": [
      {
        "hostname": "string",
```

```
        "ipAddress": "string"
      }
      ...
    ]
```

`hostname`

유형: 문자열

필수 항목 여부: 예(`extraHosts` 사용 시)

`/etc/hosts` 항목에 사용할 호스트 이름입니다.

`ipAddress`

유형: 문자열

필수 항목 여부: 예(`extraHosts` 사용 시)

`/etc/hosts` 항목에 사용할 IP 주소입니다.

스토리지 및 로깅

`readonlyRootFilesystem`

유형: 부울

필수 여부: 아니요

이 파라미터가 true일 경우 컨테이너에는 루트 파일 시스템에 대한 읽기 전용 액세스가 부여됩니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `ReadonlyRootfs` 및 docker run에 대한 `--read-only` 옵션에 매핑됩니다.

> **Note**
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

기본값은 `false`입니다.

```
"readonlyRootFilesystem": true|false
```

`mountPoints`

유형: 객체 배열

필수 여부: 아니요

컨테이너에서 데이터 볼륨의 탑재 지점입니다. 이 파라미터는 create-container Docker API의 `Volumes` 및 docker run에 대한 `--volume` 옵션에 매핑됩니다.

Windows 컨테이너는 전체 디렉터리를 동일한 드라이브에 `$env:ProgramData`로 마운트할 수 있습니다. Windows 컨테이너는 디렉터리를 다른 드라이브에 탑재할 수 없으며, 탑재 지점은 여러 드라이브에 걸쳐 사용할 수 없습니다. Amazon EBS 볼륨을 Amazon ECS 작업에 직접 연결하려면 탑재 지점을 지정해야 합니다.

`sourceVolume`

유형: 문자열

필수 항목 여부: 예(`mountPoints` 사용 시)

탑재할 볼륨의 이름입니다.

`containerPath`

유형: 문자열

필수 항목 여부: 예(`mountPoints` 사용 시)

볼륨을 탑재할 컨테이너의 경로입니다.

`readOnly`

유형: 부울

필수 여부: 아니요

이 값이 `true`일 경우 컨테이너에는 볼륨에 대한 읽기 전용 액세스가 부여됩니다. 이 값이 `false`일 경우 컨테이너는 볼륨에 쓸 수 있습니다. 기본값은 `false`입니다.

Windows 운영 체제를 실행하는 EC2 인스턴스에서 실행되는 태스크의 경우 값을 기본값인 `false`로 둡니다.

`volumesFrom`

유형: 객체 배열

필수 여부: 아니요

다른 컨테이너로부터 탑재할 데이터 볼륨입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `VolumesFrom` 및 docker run에 대한 `--volumes-from` 옵션에 매핑됩니다.

`sourceContainer`

유형: 문자열

필수 항목 여부: `volumesFrom` 사용 시, 예

탑재할 볼륨이 위치한 컨테이너의 이름입니다.

`readOnly`

유형: 부울

필수 여부: 아니요

이 값이 `true`일 경우 컨테이너에는 볼륨에 대한 읽기 전용 액세스가 부여됩니다. 이 값이 `false`일 경우 컨테이너는 볼륨에 쓸 수 있습니다. 기본값은 `false`입니다.

```
"volumesFrom": [
                {
                  "sourceContainer": "string",
                  "readOnly": true|false
                }
              ]
```

`logConfiguration`

유형: LogConfiguration 객체

필수 여부: 아니요

컨테이너의 로그 구성 사양입니다.

로그 구성을 사용하는 태스크 정의에 대한 자세한 정보는 Amazon ECS 작업 정의 예제 섹션을 참조하세요.

이 파라미터는 Docker 컨테이너 생성 명령의 `LogConfig` 및 docker run에 대한 `--log-driver` 옵션에 매핑됩니다. 기본적으로 컨테이너는 Docker 대몬(daemon)이 사용하는 것과 동일한 로깅 드라이버를 사용합니다. 하지만 컨테이너는 이 파라미터를 사용하여 컨테이너 정의에 로그 드라이버를 지정함으로써 Docker 대몬(daemon)과 다른 로깅 드라이버를 사용할 수 있습니다. 컨테이너에

다른 로깅 드라이버를 사용하려면 컨테이너 인스턴스에서(또는 원격 로깅 옵션의 다른 로그 서버에서) 로그 시스템이 올바르게 구성되어야 합니다.

컨테이너에 대한 로그 구성을 지정할 때 다음 사항에 유의해야 합니다.

- Amazon ECS는 Docker 대몬(daemon)에 사용 가능한 로깅 드라이버의 하위 집합을 지원합니다. 향후의 Amazon ECS 컨테이너 에이전트 릴리스에서 로그 드라이버가 추가될 예정입니다.
- 이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker Remote API 버전 1.18 이상을 사용해야 합니다.
- 태스크 외부에서 추가 소프트웨어를 설치해야 합니다. Fluentd 출력 집계자 또는 Gelf 로그를 보내기 위해 Logstash를 실행 중인 원격 호스트를 예로 들 수 있습니다.

```
"logConfiguration": {
      "logDriver": "awslogs","fluentd","gelf","json-
file","journald","splunk","syslog","awsfirelens",
      "options": {"string": "string"
        ...},
 "secretOptions": [{
  "name": "string",
  "valueFrom": "string"
 }]
}
```

`logDriver`

타입: 문자열

유효값: `"awslogs","fluentd","gelf","json-file","journald","splunk","syslog","awsfirelens"`

필수 항목 여부: `logConfiguration` 사용 시, 예

컨테이너에 사용할 로드 드라이버입니다. 앞에 나열된 유효한 값은 Amazon ECS 컨테이너 에이전트가 기본적으로 통신할 수 있는 로그 드라이버입니다.

지원되는 로그 드라이버는 `awslogs`, `splunk` 및 `awsfirelens`입니다.

태스크 정의에서 `awslogs` 로그 드라이버를 사용하여 컨테이너 로그를 CloudWatch Logs로 보내는 방법에 대한 자세한 정보는 Amazon ECS 로그를 CloudWatch로 전송 섹션을 참조하세요.

`awsfirelens` 로그 드라이버 사용에 대한 자세한 정보는 사용자 지정 로그 라우팅을 참조하세요.

> **Note**
>
> 목록에 포함되지 않은 사용자 지정 드라이버가 있는 경우, GitHub에서 사용 가능한 Amazon ECS 컨테이너 에이전트 프로젝트를 포킹하여 해당 드라이버와 함께 작동하도록 사용자 지정할 수 있습니다. 포함하고 싶은 변경에 대해서는 풀 요청을 제출할 것을 권장합니다. 단, 현재 이 소프트웨어의 수정된 사본 실행은 지원되지 않습니다.

이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker 원격 API 버전 1.18 이상을 사용해야 합니다.

`options`

유형: 문자열 대 문자열 맵

필수 여부: 아니요

로그 드라이버로 보낼 구성 옵션의 키/값 맵입니다.

지정할 수 있는 옵션은 로그 드라이버에 따라 달라집니다. 다음은 `awslogs` 라우터를 사용하여 로그를 Amazon CloudWatch로 라우팅할 때 지정할 수 있는 몇 가지 옵션입니다.

`awslogs-create-group`

필수 여부: 아니요

로그 그룹을 자동으로 생성할지를 지정합니다. 이 옵션이 지정되지 않은 경우 기본적으로 `false`로 설정됩니다.

> **Note**
>
> `awslogs-create-group`을 사용하기 전에 IAM 정책이 `logs:CreateLogGroup` 권한을 포함해야 합니다.

`awslogs-region`

필수 여부: 예

`awslogs` 로그 드라이버가 Docker 로그를 보낼 AWS 리전을 지정합니다. 여러 리전의 클러스터로부터 모든 로그를 CloudWatch Logs 내 단일 리전으로 전송하도록 선택할 수 있습니다. 이는 로그를 모두 한 곳에서 볼 수 있도록 하기 위함입니다. 또는, 리전별로 구분하여 세

분화할 수 있습니다. 지정한 로그 그룹이 이 옵션으로 지정한 리전에 위치하는지 확인하세요.

`awslogs-group`

필수 여부: 예

awslogs 로그 드라이버가 로그 스트림을 전송할 로그 그룹을 지정해야 합니다.

`awslogs-stream-prefix`

필수 여부: 예

awslogs-stream-prefix 옵션을 사용하여 로그 스트림을 지정한 접두사, 컨테이너 이름 및 컨테이너가 속하는 Amazon ECS 태스크의 ID와 연결할 수 있습니다. 이 옵션을 사용하여 접두사를 지정하는 경우 로그 스트림은 다음 형식을 취합니다.

```
prefix-name/container-name/ecs-task-id
```

이 옵션을 사용하여 접두사를 지정하지 않는 경우 로그 스트림 이름이 컨테이너 인스턴스의 Docker 대몬에 의해 할당된 컨테이너 ID를 따서 지정됩니다. Docker 컨테이너 ID(컨테이너 인스턴스에서만 사용 가능)만으로는 로그를 전송한 컨테이너로 역추적하기 어려우므로 이 옵션에서 접두사를 지정하는 것이 좋습니다.

Amazon ECS 서비스의 경우 서비스 이름을 접두사로 사용할 수 있습니다. 이렇게 하면 로그 스트림을 컨테이너가 속하는 서비스, 로그를 전송한 컨테이너의 이름, 컨테이너가 속하는 작업의 ID로 추적할 수 있습니다.

Amazon ECS 콘솔을 사용할 때 로그 창에 로그가 표시되도록 하기 위해서는 해당 로그에 스트림 접두사를 지정해야 합니다.

`awslogs-datetime-format`

필수 여부: 아니요

이 옵션은 Python `strftime` 형식의 여러 줄 시작 패턴을 정의합니다. 로그 메시지는 패턴과 일치하는 하나의 줄과 패턴과 일치하지 않는 나머지 줄들로 이루어져 있습니다. 일치하는 줄은 로그 메시지 간의 구분 기호입니다.

이 형식을 사용하는 사용 사례의 한 예는 스택 덤프와 같은 출력을 구문 분석하는 것이며, 그렇지 않으면 여러 항목에 기록될 수 있습니다. 올바른 패턴을 통해 단일 항목으로 캡처할 수 있습니다.

자세한 내용은 awslogs-datetime-format을 참조하세요.

awslogs-datetime-format 및 awslogs-multiline-pattern 옵션을 모두 구성할 수는 없습니다.

> **Note**
>
> 여러 줄 로깅은 모든 로그 메시지의 정규식 구문 분석 및 일치 태스크를 수행합니다. 이는 로깅 성능에 부정적인 영향을 줄 수 있습니다.

awslogs-multiline-pattern

필수 여부: 아니요

이 옵션은 정규식을 사용하여 여러 줄 시작 패턴을 정의합니다. 로그 메시지는 패턴과 일치하는 하나의 줄과 패턴과 일치하지 않는 나머지 줄들로 이루어져 있습니다. 일치하는 줄은 로그 메시지 간의 구분 기호입니다.

자세한 정보는 awslogs-multiline-pattern을 참조하세요.

이 옵션은 awslogs-datetime-format을 구성하는 경우에 무시됩니다.

awslogs-datetime-format 및 awslogs-multiline-pattern 옵션을 모두 구성할 수는 없습니다.

> **Note**
>
> 여러 줄 로깅은 모든 로그 메시지의 정규식 구문 분석 및 일치 태스크를 수행합니다. 이는 로깅 성능에 부정적인 영향을 줄 수 있습니다.

다음 옵션은 지원되는 모든 로그 드라이버에 적용됩니다.

mode

필수 여부: 아니요

유효한 값: non-blocking | blocking

이 옵션은 컨테이너에서 logDriver를 사용하여 지정된 로그 드라이버로 로그 메시지를 전송하는 모드를 정의합니다. 선택한 전송 모드는 컨테이너에서 나오는 로그 흐름이 중단될 때 애플리케이션 가용성에 영향을 줍니다.

blocking 모드를 사용하는 경우 로그 흐름이 중단되면 stdout 및 stderr 스트림에 쓰기 위한 컨테이너 코드의 호출이 차단됩니다. 결과적으로 애플리케이션의 로깅 스레드가 차단됩니다. 이로 인해 애플리케이션이 응답하지 않고 컨테이너 상태 확인에 실패할 수 있습니다.

non-blocking 모드를 사용하는 경우 컨테이너의 로그는 max-buffer-size 옵션으로 구성된 인 메모리 중간 버퍼에 대신 저장됩니다. 이렇게 하면 로그를 전송할 수 없을 때 애플리케이션이 응답하지 않게 되는 것을 방지할 수 있습니다. 서비스 가용성을 보장하고 일부 로그 손실이 괜찮다면 이 모드를 사용하는 것이 좋습니다. 자세한 내용은 Preventing log loss with non-blocking mode in the awslogs container log driver를 참조하세요.

defaultLogDriverMode 계정 설정을 사용하여 특정 AWS 리전의 모든 컨테이너에 대해 기본 mode를 설정할 수 있습니다. logConfiguration에서 mode 옵션을 지정하지 않거나 계정 설정을 구성하지 않으면 Amazon ECS는 기본적으로 non-blocking 모드로 설정됩니다. 계정 설정에 대한 자세한 내용은 기본 로그 드라이버 모드 섹션을 참조하세요.

non-blocking 모드를 사용하면 max-buffer-size 로그 옵션은 중간 메시지 스토리지에 사용되는 버퍼의 크기를 제어합니다. 애플리케이션에 따라 적절한 버퍼 크기를 지정해야 합니다. 작업 수준에서 할당된 총 메모리 양은 로그 드라이버 메모리 버퍼 외에도 모든 컨테이너에 할당된 메모리 양보다 커야 합니다.

> **Note**
>
> 2025년 6월 25일 Amazon ECS는 태스크 가용성을 로깅보다 우선시하기 위해 기본 로그 드라이버 모드를 blocking에서 non-blocking으로 변경했습니다. 이 변경 후에도 blocking 모드를 계속 사용하려면 다음 중 하나를 수행합니다.
>
> - 컨테이너 정의의 logConfiguration에서 mode 옵션을 blocking로 설정합니다.
> - defaultLogDriverMode 계정 설정을 blocking로 설정합니다.

max-buffer-size

필수 여부: 아니요

기본 값: 10m

non-blocking 모드를 사용하면 max-buffer-size 로그 옵션은 중간 메시지 스토리지에 사용되는 버퍼의 크기를 제어합니다. 애플리케이션에 따라 적절한 버퍼 크기를 지정해야 합

니다. 버퍼가 가득 차면 로그를 더 이상 저장할 수 없습니다. 저장할 수 없는 로그는 손실됩니다.

`splunk` 로그 라우터를 사용하여 로그를 라우팅하려면 `splunk-token`과 `splunk-url`을 지정해야 합니다.

`awsfirelens` 로그 라우터를 사용하여 로그 스토리지 및 분석을 위해 AWS Partner Network 또는 AWS 서비스 대상으로 로그를 라우팅하는 경우 `log-driver-buffer-limit` 옵션을 설정하여 메모리에 버퍼링되는 이벤트 수를 제한한 후 로그 라우터 컨테이너로 전송할 수 있습니다. 처리량이 많으면 Docker 내부 버퍼의 메모리가 부족해질 수 있으므로 잠재적인 로그 손실 문제를 해결하는 데 도움이 될 수 있습니다. 자세한 내용은 the section called “높은 처리량을 위한 로그 구성” 섹션을 참조하세요.

`awsfirelens`를 사용하여 로그를 라우팅할 때 지정할 수 있는 다른 옵션은 대상에 따라 다릅니다. 로그를 Amazon Data Firehose로 내보낼 때 AWS 리전으로 `region`을 지정하고 `delivery_stream`으로 로그 스트림의 이름을 지정할 수 있습니다.

로그를 Amazon Kinesis Data Streams로 내보낼 때 `region`으로 AWS 리전을 지정하고 `stream`으로 데이터 스트림 이름을 지정할 수 있습니다.

로그를 Amazon OpenSearch Service로 내보낼 때 `Name`, `Host`(프로토콜이 없는 OpenSearch Service 엔드포인트), `Port`, `Index`, `Type`, `Aws_auth`, `Aws_region`, `Suppress_Type_Name`, `tls` 등의 옵션을 지정할 수 있습니다.

로그를 Amazon S3로 내보내는 경우 `bucket` 옵션을 사용하여 버킷을 지정할 수 있습니다. `region`, `total_file_size`, `upload_timeout` 및 `use_put_object`도 옵션으로 지정할 수 있습니다.

이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker 원격 API 버전 1.19 이상을 사용해야 합니다.

`secretOptions`

유형: 객체 배열

필수 여부: 아니요

로그 구성에 전달할 암호를 나타내는 객체입니다. 로그 구성에 사용되는 보안 암호에는 인증 토큰, 인증서 또는 암호화 키가 포함될 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.

`name`

유형: 문자열

필수 항목 여부: 예

컨테이너에서 환경 변수로 설정할 값입니다.

`valueFrom`

유형: 문자열

필수 항목 여부: 예

컨테이너의 로그 구성에 노출할 암호.

```
"logConfiguration": {
 "logDriver": "splunk",
 "options": {
  "splunk-url": "https://cloud.splunk.com:8080",
  "splunk-token": "...",
  "tag": "...",
  ...
 },
 "secretOptions": [{
  "name": "splunk-token",
  "valueFrom": "/ecs/logconfig/splunkcred"
 }]
}
```

`firelensConfiguration`

유형: FirelensConfiguration 객체

필수 여부: 아니요

컨테이너의 FireLens 구성입니다. 컨테이너 로그에 대한 로그 라우터를 지정하고 구성하는 데 사용됩니다. 자세한 정보는 Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송 섹션을 참조하세요.

```
{
    "firelensConfiguration": {
        "type": "fluentd",
```

```
        "options": {
            "KeyName": ""
        }
    }
}
```

`options`

유형: 문자열 대 문자열 맵

필수 여부: 아니요

로그 라우터를 구성할 때 사용할 옵션의 키/값 맵입니다. 이 필드는 선택 사항이며, 사용자 지정 구성 파일을 지정하거나 태스크, 태스크 정의, 클러스터 및 컨테이너 인스턴스 세부 정보와 같은 추가 메타데이터를 로그 이벤트에 추가하는 데 사용할 수 있습니다. 이 필드를 지정할 경우 사용할 구문은 `"options":{"enable-ecs-log-metadata":"true|false","config-file-type:"s3|file","config-file-value":"arn:aws:s3:::amzn-s3-demo-bucket/fluent.conf|filepath"}`입니다. 자세한 정보는 Amazon ECS 태스크 정의 예제: 로그를 FireLens로 라우팅 섹션을 참조하세요.

`type`

유형: 문자열

필수 항목 여부: 예

사용할 로그 라우터입니다. 유효한 값은 `fluentd` 또는 `fluentbit`입니다.

보안

컨테이너 보안에 대한 자세한 내용은 Amazon ECS 태스크 및 컨테이너 보안 모범 사례를 참조하세요.

`credentialSpecs`

유형: 문자열 배열

필수 여부: 아니요

Active Directory 인증을 위한 컨테이너를 구성하는 보안 인증 사양(`CredSpec`) 파일에 대한 SSM 또는 Amazon S3의 ARN 목록입니다. 이 파라미터를 `dockerSecurityOptions` 대신 사용하는 것이 좋습니다. 최대 ARN 수는 1개입니다.

각 ARN의 형식은 두 가지입니다.

credentialspecdomainless:MyARN

Secrets Manager에서 보안 암호에 대한 추가 섹션에 `credentialspecdomainless:MyARN`을 사용하여 `CredSpec`를 제공합니다. 보안 암호에 도메인에 대한 로그인 보안 인증을 제공합니다.

컨테이너 인스턴스에서 실행되는 각 작업을 서로 다른 도메인에 조인할 수 있습니다.

컨테이너 인스턴스를 도메인에 조인하지 않고도 이 형식을 사용할 수 있습니다.

credentialspec:MyARN

`credentialspec:MyARN`을 사용하여 단일 도메인에 대한 `CredSpec`를 제공합니다.

이 작업 정의를 사용하는 작업을 시작하려면 먼저 컨테이너 인스턴스를 도메인에 조인해야 합니다.

두 형식 모두에서 `MyARN`을 SSM 또는 Amazon S3의 ARN으로 바꿉니다.

`credspec`에서는 사용자 이름, 암호 및 연결할 도메인을 포함하는 보안 암호에 대한 Secrets Manager의 ARN을 제공해야 합니다. 보안을 강화하기 위해 도메인 없는 인증의 경우 인스턴스가 도메인에 조인되지 않습니다. 인스턴스의 다른 애플리케이션은 도메인 없는 보안 인증을 사용할 수 없습니다. 작업을 다른 도메인에 조인해야 하는 경우에도 이 파라미터를 사용하여 동일한 인스턴스에서 작업을 실행할 수 있습니다. 자세한 내용은 Using gMSAs for Windows Containers 및 Using gMSAs for Linux Containers를 참조하세요.

`user`

유형: 문자열

필수 여부: 아니요

컨테이너 내부에서 사용할 사용자입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `User` 및 docker run에 대한 `--user` 옵션에 매핑됩니다.

> **Important**
>
> `host` 네트워크 모드를 사용하는 작업을 실행할 때는 루트 사용자(UID 0)를 사용하여 컨테이너를 실행해서는 안 됩니다. 보안상 항상 루트가 아닌 사용자를 사용합니다.

다음 형식을 사용하여 user를 지정할 수 있습니다. UID 또는 GID를 지정하는 경우 이를 양의 정수로 지정해야 합니다.

- user
- user:group
- uid
- uid:gid
- user:gid
- uid:group

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

```
"user": "string"
```

리소스 제한

ulimits

유형: 객체 배열

필수 여부: 아니요

컨테이너에 정의할 ulimit 값 목록. 이 값은 운영 체제에 설정된 기본 리소스 할당량 설정을 다시 정의합니다. 이 파라미터는 Docker 컨테이너 생성 명령의 Ulimits 및 docker run에 대한 --ulimit 옵션에 매핑됩니다.

Fargate에 호스팅되는 Amazon ECS 작업은 nofile 리소스 제한 파라미터를 제외하고 운영 체제에서 설정한 기본 리소스 제한 값을 사용합니다. nofile 리소스 제한은 컨테이너가 사용할 수 있는 열린 파일 수에 대한 제한을 설정합니다. Fargate에서 기본 nofile 소프트 제한은 65535이고 하드 제한은 65535입니다. 두 제한의 값을 모두 최대 1048576으로 설정할 수 있습니다. 자세한 내용은 태스크 리소스 제한 섹션을 참조하세요.

이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker 원격 API 버전 1.18 이상을 사용해야 합니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

```
"ulimits": [
      {
        "name":
 "core"|"cpu"|"data"|"fsize"|"locks"|"memlock"|"msgqueue"|"nice"|"nofile"|"nproc"|"rss"|"rtp
        "softLimit": integer,
        "hardLimit": integer
      }
      ...
    ]
```

`name`

타입: 문자열

유효값: `"core" | "cpu" | "data" | "fsize" | "locks" | "memlock" | "msgqueue" | "nice" | "nofile" | "nproc" | "rss" | "rtprio" | "rttime" | "sigpending" | "stack"`

필수 항목 여부: 예(`ulimits` 사용 시)

`ulimit`의 `type`입니다.

`hardLimit`

유형: 정수

필수 항목 여부: 예(`ulimits` 사용 시)

`ulimit` 유형의 하드 제한입니다. 값은 `ulimit`의 `type`에 따라 바이트, 초 또는 개수로 지정할 수 있습니다.

`softLimit`

유형: 정수

필수 항목 여부: 예(`ulimits` 사용 시)

`ulimit` 유형의 소프트 제한입니다. 값은 `ulimit`의 `type`에 따라 바이트, 초 또는 개수로 지정할 수 있습니다.

Docker 레이블

`dockerLabels`

유형: 문자열 대 문자열 맵

필수 여부: 아니요

컨테이너에 추가할 레이블의 키/값 맵입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `Labels` 및 docker run에 대한 `--label` 옵션에 매핑됩니다.

이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker 원격 API 버전 1.18 이상을 사용해야 합니다.

```
"dockerLabels": {"string": "string"
      ...}
```

기타 컨테이너 정의 파라미터

다음 컨테이너 정의 파라미터는 JSON을 통해 구성(Configure via JSON) 옵션을 사용하여 Amazon ECS 콘솔에서 태스크 정의를 등록할 때 사용할 수 있습니다. 자세한 정보는 콘솔을 사용하여 Amazon ECS 작업 정의 생성을 참조하세요.

주제

- Linux 파라미터
- 컨테이너 종속성
- 컨테이너 제한 시간
- System Controls
- 대화형
- 의사 터미널

Linux 파라미터

`linuxParameters`

유형: LinuxParameters 객체

필수 여부: 아니요

컨테이너에 적용되는 Linux 관련 옵션(예: KernelCapabilities)

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서는 지원되지 않습니다.

```
"linuxParameters": {
      "capabilities": {
        "add": ["string", ...],
        "drop": ["string", ...]
        }
      }
```

`capabilities`

유형: KernelCapabilities 객체

필수 여부: 아니요

컨테이너에 대해 Docker에서 제공하는 기본 구성에서 제거할 Linux 기능입니다. 이러한 Linux 기능에 대한 자세한 정보는 capabilities(7) Linux 매뉴얼 페이지를 참조하세요.

`add`

유형: 문자열 배열

유효값: `"SYS_PTRACE"`

필수 여부: 아니요

컨테이너에 대해 Docker에서 제공하는 기본 구성에 추가할 Linux 기능입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `CapAdd` 및 docker run에 대한 `--cap-add` 옵션에 매핑됩니다.

`drop`

유형: 문자열 배열

유효값: `"ALL" | "AUDIT_CONTROL" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER"`

```
| "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" |
"LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" |
"NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW"
| "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" |
"SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT"
| "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" |
"SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"
```

필수 여부: 아니요

컨테이너에 대해 Docker에서 제공하는 기본 구성에서 제거할 Linux 기능입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `CapDrop` 및 docker run에 대한 `--cap-drop` 옵션에 매핑됩니다.

`devices`

컨테이너에 노출될 모든 호스트 디바이스. 이 파라미터는 Docker 컨테이너 생성 명령의 `Devices` 및 docker run에 대한 `--device` 옵션에 매핑됩니다.

> **Note**
>
> 이 `devices` 파라미터는 Fargate 시작 유형을 사용하는 경우 지원되지 않습니다.

유형: Device 객체 배열

필수 여부: 아니요

`hostPath`

호스트 컨테이너 인스턴스의 디바이스 경로.

유형: 문자열

필수 항목 여부: 예

`containerPath`

호스트 디바이스를 노출할 컨테이너 내 경로.

유형: 문자열

필수 여부: 아니요

`permissions`

디바이스를 위해 컨테이너에 제공할 명시적 권한. 컨테이너는 기본적으로 디바이스에서 `read`, `write` 및 `mknod`에 대한 권한이 있습니다.

유형: 문자열 배열

유효한 값: `read` | `write` | `mknod`

`initProcessEnabled`

신호를 전달하고 프로세스의 결과를 받아들이는 컨테이너 내에서 `init` 프로세스를 실행합니다. 이 파라미터는 docker run에 대한 `--init` 옵션에 매핑됩니다.

이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker 원격 API 버전 1.25 이상을 사용해야 합니다.

`maxSwap`

Fargate에서 실행되는 태스크에는 지원되지 않습니다.

컨테이너가 사용할 수 있는 총 스왑 메모리 양(MiB) 이 파라미터는 docker run에 대한 `--memory-swap` 옵션으로 변환되며 컨테이너 메모리의 합계에 `maxSwap` 값을 더한 값이 됩니다.

`0`의 `maxSwap` 값이 지정되면 컨테이너는 스왑을 사용하지 않습니다. 허용되는 값은 `0` 또는 양수입니다. `maxSwap` 파라미터를 생략하면 컨테이너는 실행 중인 컨테이너 인스턴스에 대한 스왑 구성을 사용합니다. `swappiness` 매개 변수를 사용하려면 `maxSwap` 값을 설정해야 합니다.

`sharedMemorySize`

`/dev/shm` 볼륨의 크기 값(MiB)입니다. 이 파라미터는 docker run에 대한 `--shm-size` 옵션에 매핑됩니다.

> **Note**
>
> Fargate 시작 유형을 사용하는 태스크를 사용하는 경우에는 `sharedMemorySize` 파라미터가 지원되지 않습니다.

유형: 정수

`tmpfs`

컨테이너 경로, 마운트 옵션 및 tmpfs 마운트의 최대 크기(MiB)입니다. 이 파라미터는 docker run에 대한 `--tmpfs` 옵션에 매핑됩니다.

> **Note**
>
> Fargate 시작 유형을 사용하는 태스크를 사용하는 경우에는 `tmpfs` 파라미터가 지원되지 않습니다.

유형: Tmpfs 객체 배열

필수 여부: 아니요

`containerPath`

tmpfs 볼륨을 마운트할 절대 파일 경로입니다.

유형: 문자열

필수 항목 여부: 예

`mountOptions`

tmpfs 볼륨 마운트 옵션의 목록입니다.

유형: 문자열 배열

필수 여부: 아니요

유효 값: `"defaults" | "ro" | "rw" | "suid" | "nosuid" | "dev" | "nodev" | "exec" | "noexec" | "sync" | "async" | "dirsync" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" | "private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" | "relatime" | "norelatime" | "strictatime" | "nostrictatime" | "mode" | "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"`

`size`

tmpfs 볼륨의 최대 크기(MiB) 입니다.

유형: 정수

필수 여부: 예

컨테이너 종속성

dependsOn

유형: ContainerDependency 객체의 배열

필수 여부: 아니요

컨테이너 시작 및 종료에 대해 정의된 종속성입니다. 컨테이너는 여러 종속성을 포함할 수 있습니다. 컨테이너가 시작될 때 종속성이 정의되면 컨테이너를 종료할 때 종속성이 취소됩니다. 문제 해결 예는 컨테이너 종속성 섹션을 참조하세요.

> **Note**
>
> 컨테이너가 종속성 제약을 충족하지 못하거나 제약을 충족하기 전에 시간이 초과되면 Amazon ECS는 종속된 컨테이너를 다음 상태로 진행시키지 못합니다.

이 파라미터를 사용하려면 태스크 또는 서비스에서 플랫폼 버전 `1.3.0` 이상(Linux) 또는 `1.0.0`(Windows)을 사용해야 합니다.

```
"dependsOn": [
    {
        "containerName": "string",
        "condition": "string"
    }
]
```

containerName

유형: 문자열

필수 항목 여부: 예

지정된 조건을 충족해야 하는 컨테이너 이름입니다.

condition

유형: 문자열

필수 항목 여부: 예

컨테이너의 종속성 조건입니다. 다음은 사용 가능한 조건과 그 동작입니다.

- START - 이 조건은 오늘의 링크와 볼륨의 동작을 에뮬레이션합니다. 이 조건은 종속 컨테이너가 시작되었는지 확인한 다음에 다른 컨테이너 시작을 허용합니다.
- COMPLETE - 이 조건은 다른 컨테이너를 시작하기 전에 종속 컨테이너 실행이 완료(종료)되었는지 확인합니다. 이는 스크립트를 실행한 후 종료하는 필수적이지 않은 컨테이너에 유용할 수 있습니다. 필수 컨테이너에는 이 조건을 설정할 수 없습니다.
- SUCCESS - 이 조건은 COMPLETE와 동일하지만 컨테이너가 zero 상태로 종료되어야 합니다. 필수 컨테이너에는 이 조건을 설정할 수 없습니다.
- HEALTHY - 이 조건은 다른 컨테이너를 시작하기 전에 종속 컨테이너가 컨테이너 상태 확인을 통과하는지 확인합니다. 이렇게 하려면 태스크 정의에서 종속 컨테이너에 상태 확인이 구성되어 있어야 합니다. 이 조건은 태스크 시작 시에만 확인됩니다.

컨테이너 제한 시간

`startTimeout`

유형: 정수

필수 여부: 아니요

예제 값: `120`

컨테이너에 대한 종속성 해결을 포기하기 전에 대기할 시간(초)입니다.

예를 들어 containerA에 대한 태스크 정의에서 containerB에 대한 종속성이 COMPLETE, SUCCESS 또는 HEALTHY 상태인 두 개의 컨테이너를 지정합니다. containerB에 대해 startTimeout 값이 지정되고 그 시간 내에 원하는 상태에 도달하지 않으면 containerA는 시작하지 않습니다.

> **Note**
>
> 컨테이너가 종속성 제약을 충족하지 못하거나 제약을 충족하기 전에 시간이 초과되면 Amazon ECS는 종속된 컨테이너를 다음 상태로 진행시키지 못합니다.

이 파라미터를 사용하려면 태스크 또는 서비스에서 플랫폼 버전 `1.3.0` 이상(Linux)을 사용해야 합니다. 최대값은 120초입니다.

`stopTimeout`

유형: 정수

필수 여부: 아니요

예제 값: `120`

자체적으로 정상적으로 종료하지 않을 경우 컨테이너를 강제 종료하기까지 대기할 시간(초)입니다.

이 파라미터를 사용하려면 태스크 또는 서비스에서 플랫폼 버전 `1.3.0` 이상(Linux)을 사용해야 합니다. 파라미터를 지정하지 않으면 기본값인 30초가 사용됩니다. 최대값은 120초입니다.

System Controls

`systemControls`

유형: SystemControl 객체

필수 여부: 아니요

컨테이너에서 설정할 네임스페이스 커널 파라미터의 목록입니다. 이 파라미터는 Docker 컨테이너 생성 명령에 있는 `Sysctls` 및 docker run에 대한 `--sysctl` 옵션에 매핑됩니다. 예를 들어 `net.ipv4.tcp_keepalive_time` 설정을 구성하여 연결을 더 오래 유지할 수 있습니다.

`awsvpc` 또는 `host` 네트워크 모드도 사용하는 단일 작업에서 여러 컨테이너에 네트워크 관련 `systemControls` 파라미터를 지정하지 않는 것이 좋습니다. 이렇게 하면 다음과 같은 단점이 있습니다.

- 컨테이너에 `systemControls`를 설정할 경우 태스크의 모든 컨테이너에 적용됩니다. 단일 작업의 여러 컨테이너에 서로 다른 `systemControls`를 설정할 경우 마지막으로 시작되는 컨테이너에 따라 적용될 `systemControls`가 결정됩니다.

태스크 내 컨테이너에 사용할 IPC 리소스 네임스페이스를 설정하는 경우, 시스템 제어에 다음 사항이 적용됩니다. 자세한 정보는 IPC 모드을 참조하세요.

- `host` IPC 모드를 사용하는 태스크의 경우, IPC 네임스페이스 `systemControls`는 지원되지 않습니다.
- `task` IPC 모드를 사용하는 작업의 경우, IPC 네임스페이스 `systemControls` 값은 작업 내 모든 컨테이너에 적용됩니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

> **Note**
>
> 이 파라미터는 플랫폼 버전 `1.4.0` 이상(Linux)을 사용 중인 경우 AWS Fargate에서 호스팅되는 작업에 대해서만 지원됩니다. Fargate의 Windows 컨테이너에서는 지원되지 않습니다.

```
"systemControls": [
    {
         "namespace":"string",
         "value":"string"
    }
]
```

`namespace`

유형: 문자열

필수 여부: 아니요

`value`를 설정할 네임스페이스 커널 파라미터입니다.

유효한 IPC 네임스페이스 값:`"kernel.msgmax" | "kernel.msgmnb" | "kernel.msgmni" | "kernel.sem" | "kernel.shmall" | "kernel.shmmax" | "kernel.shmmni" | "kernel.shm_rmid_forced"` 및 `"fs.mqueue.*"`로 시작하는 Sysctls

유효한 네트워크 네임스페이스 값: `"net.*"`로 시작하는 Sysctls. Fargate에서는 컨테이너 내에 있는 네임스페이스 Sysctls만 허용됩니다.

이러한 값은 모두 Fargate에서 지원됩니다.

`value`

유형: 문자열

필수 여부: 아니요

namespace에 지정된 네임스페이스 커널 파라미터의 값입니다.

대화형

interactive

유형: 부울

필수 여부: 아니요

이 파라미터가 true일 경우 이것을 사용하여 stdin 또는 tty를 할당해야 하는 컨테이너화된 애플리케이션을 배포할 수 있습니다. 이 파라미터는 Docker 컨테이너 생성 명령의 OpenStdin 및 docker run에 대한 --interactive 옵션에 매핑됩니다.

기본값은 false입니다.

의사 터미널

pseudoTerminal

유형: 부울

필수 여부: 아니요

이 파라미터가 true일 경우 TTY가 할당됩니다. 이 파라미터는 Docker 컨테이너 생성 명령의 Tty 및 docker run에 대한 --tty 옵션에 매핑됩니다.

기본값은 false입니다.

Elastic Inference 액셀러레이터 이름

태스크 정의에 대한 Elastic Inference 액셀러레이터 리소스 요구 사항.

> **Note**
>
> Amazon Elastic Inference(EI)는 더 이상 고객에게 제공되지 않습니다.

태스크 정의에는 다음 파라미터가 허용됩니다.

`deviceName`

유형: 문자열

필수 항목 여부: 예

Amazon Elastic Inference 액셀러레이터 디바이스 이름입니다. `deviceName`은 컨테이너 정의에서도 참조해야 합니다. `Elastic Inference accelerator`를 참조하세요.

`deviceType`

유형: 문자열

필수 항목 여부: 예

사용할 Elastic Inference 액셀러레이터입니다.

프록시 구성

`proxyConfiguration`

유형: ProxyConfiguration 객체

필수 여부: 아니요

App Mesh 프록시에 대한 구성 세부 정보입니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

```
"proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "string",
    "properties": [
        {
           "name": "string",
           "value": "string"
        }
    ]
}
```

`type`

유형: 문자열

유효한 값: `APPMESH`

필수 여부: 아니요

프록시 유형입니다. 지원되는 유일한 값은 `APPMESH`입니다.

`containerName`

유형: 문자열

필수 항목 여부: 예

App Mesh 프록시로 사용할 컨테이너의 이름입니다.

`properties`

형식: KeyValuePair 객체 배열

필수 여부: 아니요

키-값 쌍으로 지정된 CNI(Container Network Interface) 플러그인을 제공하는 네트워크 구성 파라미터 세트입니다.

- `IgnoredUID` - (필수) 컨테이너 정의의 `user` 파라미터에 정의된 프록시 컨테이너의 사용자 ID(UID)입니다. 프록시가 트래픽을 무시하는 데 사용됩니다. `IgnoredGID`를 지정한 경우 이 필드를 비워둘 수 있습니다.
- `IgnoredGID` - (필수) 컨테이너 정의의 `user` 파라미터에 정의된 프록시 컨테이너의 그룹 ID(GID)입니다. 프록시가 트래픽을 무시하는 데 사용됩니다. `IgnoredUID`를 지정한 경우 이 필드를 비워둘 수 있습니다.
- `AppPorts` - (필수) 애플리케이션이 사용하는 포트 목록입니다. 이러한 포트에 대한 네트워크 트래픽은 `ProxyIngressPort` 및 `ProxyEgressPort`로 전달됩니다.
- `ProxyIngressPort` - (필수) `AppPorts`로 들어오는 트래픽을 보낼 포트를 지정합니다.
- `ProxyEgressPort` - (필수) `AppPorts`에서 나가는 트래픽을 보낼 포트를 지정합니다.
- `EgressIgnoredPorts` - (필수) 지정된 포트로 가는 아웃바운드 트래픽은 무시되고 `ProxyEgressPort`로 리디렉션되지 않습니다. 값은 빈 목록일 수 있습니다.
- `EgressIgnoredIPs` - (필수) 지정된 IP 주소로 가는 아웃바운드 트래픽은 무시되고 `ProxyEgressPort`로 리디렉션되지 않습니다. 값은 빈 목록일 수 있습니다.

`name`

유형: 문자열

필수 여부: 아니요

키-값 쌍의 이름입니다.

`value`

유형: 문자열

필수 여부: 아니요

키-값 쌍의 값입니다.

볼륨

작업 정의를 등록할 때 선택적으로 볼륨 목록이 컨테이너 인스턴스의 Docker 대몬에 전달되도록 지정할 수 있습니다. 그러면 동일한 컨테이너 인스턴스에 속하는 다른 컨테이너가 액세스하는 데 전달된 볼륨 목록을 사용할 수 있습니다.

사용할 수 있는 데이터 볼륨의 유형은 다음과 같습니다.

- Amazon EBS 볼륨 - 데이터 집약적인 컨테이너화된 워크로드에 대해 비용 효율적이며 내구성이 뛰어난 고성능 블록 스토리지를 제공합니다. 독립 실행형 작업을 실행하거나 서비스를 생성 또는 업데이트할 때 Amazon ECS 작업당 1개의 Amazon EBS 볼륨을 연결할 수 있습니다. Amazon EBS 볼륨은 Fargate에서 호스팅되는 Linux 태스크에 지원됩니다. 자세한 내용은 Amazon ECS에서 Amazon EBS 볼륨 사용 섹션을 참조하세요.
- Amazon EFS 볼륨 — Amazon ECS 태스크에 사용할 수 있는 간단하고 영구적인 확장형 파일 스토리지를 제공합니다. Amazon EFS를 사용하면 스토리지 용량이 탄력적입니다. 파일을 추가 및 제거하면 스토리지 용량이 자동으로 확장 및 축소됩니다. 애플리케이션에서 스토리지가 필요할 때 필요한 만큼 확보할 수 있습니다. Amazon EFS 볼륨은 Fargate에서 호스팅되는 태스크에 지원됩니다. 자세한 내용은 Amazon ECS에서 Amazon EFS 볼륨 사용 섹션을 참조하세요.
- FSx for Windows File Server 볼륨 - 완전 관리형 Microsoft Windows 파일 서버를 제공합니다. 이러한 파일 서버는 Windows 파일 시스템으로 지원됩니다. FSx for Windows File Server와 Amazon ECS를 함께 사용하면 영구적이고 분산, 공유된 고정 파일 스토리지로 Windows 태스크를 프로비저닝할 수 있습니다. 자세한 정보는 Amazon ECS에서 FSx for Windows File Server 볼륨 사용을 참조하세요.

Fargate의 Windows 컨테이너는 이 옵션을 지원하지 않습니다.

- 바인드 탑재 – 컨테이너에 탑재된 호스트 컴퓨터의 파일 또는 디렉터리입니다. 바인드 탑재 호스트 볼륨은 태스크를 실행할 때 지원됩니다. 바인드 탑재 호스트 볼륨을 사용하려면 태스크 정의에서 `host`와 `sourcePath`(옵션) 값을 지정합니다.

자세한 내용은 Amazon ECS 작업에 대한 스토리지 옵션 섹션을 참조하세요.

컨테이너 정의에서 다음 파라미터가 허용됩니다.

`name`

유형: 문자열

필수 여부: 아니요

볼륨의 이름입니다. 최대 255자의 문자(대문자 및 소문자), 숫자, 하이(-) 및 밑줄(_)이 허용됩니다. 이 이름은 컨테이너 정의 `mountPoints` 객체의 `sourceVolume` 파라미터에서 참조됩니다.

`host`

필수 여부: 아니요

`host` 파라미터는 바인드 탑재의 수명 주기를 태스크가 아니라 호스트 Amazon EC2 인스턴스와 연결하는 데 사용합니다. `host` 파라미터가 비어 있으면 Docker 대몬이 데이터 볼륨의 호스트 경로를 할당하지만 해당 볼륨과 연결된 컨테이너가 실행을 중지한 후 데이터 유지가 보장되지 않습니다.

Windows 컨테이너는 전체 디렉터리를 동일한 드라이브에 `$env:ProgramData`로 마운트할 수 있습니다.

> **Note**
>
> `sourcePath` 파라미터는 Amazon EC2 인스턴스에 호스팅된 작업을 사용하는 경우에만 지원됩니다.

`sourcePath`

유형: 문자열

필수 여부: 아니요

host 파라미터가 사용되는 경우 sourcePath를 지정하여 컨테이너에 제시되는 호스트 Amazon EC2 인스턴스 상의 경로를 선언합니다. 이 파라미터가 비어 있으면 Docker 대몬이 사용자 대신 호스트 경로를 할당합니다. host 파라미터에 sourcePath 파일 위치가 들어 있으면, 사용자가 수동으로 삭제하지 않는 한 데이터 볼륨이 호스트 Amazon EC2 인스턴스 상에 지정된 위치를 유지합니다. sourcePath 값이 호스트 Amazon EC2 인스턴스에 없을 경우 Docker 대몬이 해당 경로를 생성합니다. 해당 위치가 있을 경우 소스 경로 폴더의 콘텐츠를 내보냅니다.

configuredAtLaunch

유형: 부울

필수 여부: 아니요

시작 시 볼륨을 구성할 수 있는지 여부를 지정합니다. true로 설정하면 독립 실행형 작업을 실행하거나 서비스를 생성 또는 업데이트할 때 볼륨을 구성할 수 있습니다. true로 설정하면 작업 정의에서 다른 볼륨 구성을 제공할 수 없습니다. 작업에 연결하도록 Amazon EBS 볼륨을 구성하려면 이 파라미터를 true로 설정해야 합니다. configuredAtLaunch를 true로 설정하고 볼륨 구성을 시작 단계로 연기하면 볼륨 유형이나 특정 볼륨 설정으로 제한되지 않는 작업 정의를 생성할 수 있습니다. 그러면 여러 실행 환경에서 작업 정의를 재사용할 수 있습니다. 자세한 내용은 Amazon EBS volumes를 참조하세요.

dockerVolumeConfiguration

유형: DockerVolumeConfiguration 객체

필수 여부: 아니요

이 파라미터는 Docker 볼륨을 사용할 때 지정됩니다. Docker 볼륨은 EC2 인스턴스에서 작업을 실행하는 경우에만 지원됩니다. Windows 컨테이너는 local 드라이버의 사용만 지원합니다. 바인드 탑재를 사용하려면 대신에 host를 지정하세요.

scope

유형: 문자열

유효한 값: task | shared

필수 여부: 아니요

수명 주기를 결정하는 Docker 볼륨의 범위입니다. 범위가 task로 지정된 Docker 볼륨은 태스크가 시작될 때 자동으로 프로비저닝되고, 태스크가 중단되면 삭제됩니다. 범위가 shared로 지정된 Docker 볼륨은 태스크 중단 후에도 유지됩니다.

`autoprovision`

유형: 부울

기본 값: `false`

필수 여부: 아니요

이 값이 `true`인 경우 도커 볼륨이 아직 없으면 도커 볼륨이 생성됩니다. 이 필드는 `scope`가 `shared`인 경우에만 사용됩니다. `scope`가 `task`인 경우 이 파라미터를 생략해야 합니다.

`driver`

유형: 문자열

필수 여부: 아니요

사용할 Docker 볼륨 드라이버입니다. Docker에서 제공하는 드라이버 이름이 작업 배치에 사용되므로 드라이버 값과 이 이름이 일치해야 합니다. Docker 플러그인 CLI를 사용하여 드라이버를 설치했다면 `docker plugin ls`를 사용하여 컨테이너 인스턴스에서 드라이버 이름을 검색합니다. 다른 방법을 사용하여 드라이버를 설치했다면 Docker 플러그인 검색을 사용하여 드라이버 이름을 검색합니다.

`driverOpts`

유형: 문자열

필수 여부: 아니요

전달할 Docker 드라이버에 특정한 옵션 맵입니다. 이 파라미터는 Docker의 볼륨 생성의 `DriverOpts`에 매핑됩니다.

`labels`

유형: 문자열

필수 여부: 아니요

Docker 볼륨에 추가할 사용자 지정 메타데이터입니다.

`efsVolumeConfiguration`

유형: EFSVolumeConfiguration 객체

필수 여부: 아니요

이 파라미터는 Amazon EFS 볼륨을 사용할 때 지정됩니다.

`fileSystemId`

유형: 문자열

필수 항목 여부: 예

사용할 Amazon EFS 파일 시스템 ID입니다.

`rootDirectory`

유형: 문자열

필수 여부: 아니요

호스트 내의 루트 디렉터리로 탑재할 Amazon EFS 파일 시스템 내 디렉터리입니다. 이 파라미터가 생략되면 Amazon EFS 볼륨의 루트가 사용됩니다. `/`를 지정하면 이 파라미터를 생략하는 것과 동일한 효과가 있습니다.

> **Important**
>
> `authorizationConfig`에 EFS 액세스 포인트가 지정된 경우 루트 디렉터리 파라미터를 생략하거나 `/`로 설정해야 합니다. 그러면 EFS 액세스 포인트에 설정된 경로가 적용됩니다.

`transitEncryption`

유형: 문자열

유효한 값: `ENABLED` | `DISABLED`

필수 여부: 아니요

Amazon ECS 호스트와 Amazon EFS 서버 간에 전송 중인 Amazon EFS 데이터에 대해 암호화를 사용 설정할지 여부를 지정합니다. Amazon EFS IAM 권한 부여를 사용하는 경우 전송 중 데이터 암호화를 사용 설정해야 합니다. 이 파라미터가 누락되면 `DISABLED`의 기본값이 사용됩니다. 자세한 정보는 Amazon Elastic File System 사용 설명서의 전송 중 데이터 암호화 (Encrypting Data in Transit)를 참조하세요.

`transitEncryptionPort`

유형: 정수

필수 여부: 아니요

Amazon ECS 호스트와 Amazon EFS 서버 간에 암호화된 데이터를 전송할 때 사용할 포트입니다. 전송 중 데이터 암호화 포트를 지정하지 않으면 작업에서는 Amazon EFS 탑재 헬퍼가 사용하는 포트 선택 전략을 사용합니다. 자세한 정보는 Amazon Elastic File System 사용자 설명서의 EFS 탑재 헬퍼를 참조하세요.

`authorizationConfig`

유형: EFSAuthorizationConfig 객체

필수 여부: 아니요

Amazon EFS 파일 시스템에 대한 권한 부여 구성 세부 정보입니다.

`accessPointId`

유형: 문자열

필수 여부: 아니요

사용할 액세스 포인트 ID입니다. 액세스 포인트가 지정된 경우 `efsVolumeConfiguration`에서 루트 디렉터리 값을 생략하거나 `/`로 설정해야 합니다. 그러면 EFS 액세스 포인트에 설정된 경로가 적용됩니다. 액세스 포인트를 사용하는 경우 `EFSVolumeConfiguration`에서 전송 중 데이터 암호화를 활성화해야 합니다. 자세한 정보는 Amazon Elastic File System 사용자 설명서의 Amazon EFS 액세스 포인트 태스크를 참조세요.

`iam`

유형: 문자열

유효한 값: `ENABLED` | `DISABLED`

필수 여부: 아니요

Amazon EFS 파일 시스템을 탑재할 때 작업 정의에 정의된 Amazon ECS 작업 IAM 역할을 사용할지 여부를 지정합니다. 활성화된 경우 `EFSVolumeConfiguration`에서 전송 중 데이터 암호화를 활성화해야 합니다. 이 파라미터가 누락되면 `DISABLED`의 기본값이 사용됩니다. 자세한 정보는 태스크의 IAM 역할을 참조하세요.

`FSxWindowsFileServerVolumeConfiguration`

유형: FSxWindowsFileServerVolumeConfiguration 객체

필수 여부: 예

이 파라미터는 작업을 저장할 Amazon FSx for Windows File Server 파일 시스템을 사용할 때 지정됩니다.

`fileSystemId`

유형: 문자열

필수 항목 여부: 예

사용할 FSx for Windows File Server 파일 시스템 ID입니다.

`rootDirectory`

유형: 문자열

필수 항목 여부: 예

호스트 내의 루트 디렉터리로 탑재할 FSx for Windows File Server 파일 시스템 내 디렉터리입니다.

`authorizationConfig`

`credentialsParameter`

유형: 문자열

필수 항목 여부: 예

권한 부여 자격 증명 옵션입니다.

옵션:

- AWS Secrets Manager 보안 암호의 Amazon 리소스 이름(ARN)입니다.
- AWS Systems Manager 파라미터의 ARN입니다.

`domain`

유형: 문자열

필수 항목 여부: 예

셀프 호스팅된 EC2 Active Directory 또는 AWS Directory Service for Microsoft Active Directory(AWS Managed Microsoft AD) 디렉터리에서 호스팅하는 정규화된 도메인 이름입니다.

Tags

태스크 정의를 등록할 때 태스크 정의에 적용되는 메타데이터 태그를 선택적으로 지정할 수 있습니다. 태그는 태스크 정의를 분류하고 정리하는 데 도움을 줍니다. 각 태그는 키와 값(선택사항)으로 구성됩니다. 두 가지를 모두 정의합니다. 자세한 정보는 Amazon ECS 리소스 태그 지정을 참조하세요.

> **Important**
>
> 개인 식별 정보나 기타 기밀 정보 또는 민감한 정보를 태그에 추가하지 않습니다. 청구를 비롯한 여러 AWS 서비스에서 태그에 액세스할 수 있습니다. 태그는 개인 데이터나 민감한 데이터에 사용하기 위한 것이 아닙니다.

태그 개체에서 허용되는 파라미터는 다음과 같습니다.

`key`

유형: 문자열

필수 여부: 아니요

하나의 태그를 구성하는 키-값 쌍의 일부분입니다. 키는 더 구체적인 태그 값에 대해 범주와 같은 역할을 하는 일반적인 레이블입니다.

`value`

유형: 문자열

필수 여부: 아니요

하나의 태그를 구성하는 키-값 쌍의 선택적 부분입니다. 하나의 값은 태그 범주(키) 내에서 서술자 역할을 수행합니다.

기타 태스크 정의 파라미터

다음의 태스크 정의 파라미터는 JSON을 통한 구성(Configure via JSON) 옵션을 사용하여 Amazon ECS 콘솔에 태스크 정의를 등록할 때 사용할 수 있습니다. 자세한 정보는 콘솔을 사용하여 Amazon ECS 작업 정의 생성을 참조하세요.

주제

- 임시 스토리지

- IPC 모드
- PID 모드
- 결함 주입

임시 스토리지

`ephemeralStorage`

유형: EphemeralStorage 객체

필수 여부: 아니요

태스크에 할당되는 임시 스토리지 용량(GB)입니다. 이 파라미터는 AWS Fargate에서 호스팅되는 태스크에 대해 제공되는 임시 스토리지 총량을 기본 용량 이상으로 확장할 때 사용합니다. 자세한 내용은 the section called “바인드 탑재” 섹션을 참조하세요.

> **Note**
>
> 이 파라미터는 플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0` 이상(Windows)에서만 지원됩니다.

IPC 모드

`ipcMode`

Fargate에서 실행되는 태스크에는 지원되지 않습니다.

유형: 문자열

필수 여부: 아니요

해당 태스크의 컨테이너에 사용할 IPC 리소스 네임스페이스입니다. 유효한 값은 `host`, `task` 또는 `none`입니다. `host`를 지정하면 동일한 컨테이너 인스턴스에서 `host` IPC 모드를 지정한 태스크 내 모든 컨테이너가 동일한 IPC 리소스를 호스트 Amazon EC2 인스턴스와 공유합니다. `task`를 지정하면 지정된 태스크 내 모든 컨테이너가 동일한 IPC 리소스를 공유합니다. `none`이 지정된 경우, 태스크 컨테이너 내에 있는 IPC 리소스는 프라이빗이며, 태스크 또는 컨테이너 인스턴스의 다른 컨테이너와 공유되지 않습니다. 값을 지정하지 않을 경우, IPC 리소스 네임스페이스 공유는 컨테이너 인스턴스의 Docker 데몬 설정에 따라 달라집니다.

host IPC 모드를 사용하는 경우, 원치 않는 IPC 네임스페이스 노출이 발생할 위험이 커집니다.

해당 태스크 내 컨테이너에 대해 systemControls를 사용하여 네임스페이스가 있는 커널 파라미터를 설정하는 경우, IPC 리소스 네임스페이스에 다음 사항이 적용됩니다.

- host IPC 모드를 사용하는 태스크의 경우, IPC 네임스페이스 관련 systemControls는 지원되지 않습니다.
- task IPC 모드를 사용하는 태스크의 경우, IPC 네임스페이스 관련 systemControls가 태스크 내 모든 컨테이너에 적용됩니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너 또는 Fargate 시작 유형을 사용하는 태스크에 대해서는 지원되지 않습니다.

PID 모드

`pidMode`

유형: 문자열

유효한 값: host | task

필수 여부: 아니요

해당 태스크의 컨테이너에 사용할 프로세스 네임스페이스입니다. 유효한 값은 host 또는 task입니다. Linux 컨테이너에서 유효한 값은 task뿐입니다. 예를 들어 사이드카 모니터링에서는 동일한 작업에서 실행 중인 다른 컨테이너에 대한 정보에 액세스하기 위해 pidMode가 필요할 수 있습니다.

task을 지정하면 지정된 태스크 내 모든 컨테이너가 동일한 프로세스 네임스페이스를 공유합니다.

값을 지정하지 않을 경우, 기본값은 각 컨테이너의 프라이빗 네임스페이스입니다.

> **Note**
>
> 이 파라미터는 플랫폼 버전 1.4.0 이상(Linux)을 사용 중인 경우 AWS Fargate에서 호스팅되는 작업에 대해서만 지원됩니다. Fargate의 Windows 컨테이너에서는 지원되지 않습니다.

결함 주입

`enableFaultInjection`

유형: 부울

유효한 값: `true` | `false`

필수 여부: 아니요

이 파라미터가 `true`로 설정된 경우 작업의 페이로드에서 Amazon ECS와 Fargate는 작업 컨테이너의 결함 주입 요청을 수락합니다. 이 파라미터는 기본적으로 `false`로 설정되어 있습니다.

EC2 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터

작업 정의는 작업 패밀리, AWS Identity and Access Management(IAM) 작업 역할, 네트워크 모드, 컨테이너 정의, 볼륨, 작업 배치 제약 조건, 시작 유형 등의 별도 부분으로 분할됩니다. 태스크 정의에는 패밀리 및 컨테이너 정의가 필요합니다. 하지만 태스크 역할, 네트워크 모드, 볼륨, 작업 배치 제약 및 시작 유형은 선택 사항입니다.

JSON 파일에서 이러한 파라미터를 사용하여 태스크 정의를 구성할 수 있습니다.

다음은 EC2 시작 유형의 각 태스크 정의 파라미터에 대한 자세한 설명입니다.

Family

`family`

유형: 문자열

필수 항목 여부: 예

태스크 정의를 등록할 때 패밀리를 지정합니다. 패밀리는 개정 번호를 사용하여 지정된 태스크 정의의 여러 버전에 대한 이름과 비슷합니다. 특정 패밀리에 등록된 첫 번째 태스크 정의에는 개정 번호 1이 부여되고 그 이후 등록된 태스크 정의에는 순차적으로 개정 번호가 부여됩니다.

시작 유형

태스크 정의를 등록할 때 Amazon ECS가 태스크 정의의 유효성을 검사해야 하는 시작 유형을 지정할 수 있습니다. 태스크 정의가 유효성을 지정한 호환성에 대해 검사하지 않으면 클라이언트 예외가 반환됩니다. 자세한 정보는 Amazon ECS 시작 유형을 참조하세요.

태스크 정의에서 다음 파라미터가 허용됩니다.

`requiresCompatibilities`

유형: 문자열 배열

필수 여부: 아니요

유효 값: `EC2`

태스크 정의의 유효성을 검사하기 위한 시작 유형입니다. 이를 통해 작업 정의에 사용되는 모든 파라미터가 시작 유형의 요구 사항을 충족하는지 확인할 수 있습니다.

태스크 역할

`taskRoleArn`

유형: 문자열

필수 여부: 아니요

태스크 정의를 등록할 때 태스크 권한의 컨테이너가 사용자 대신 연결된 정책에 지정된 AWS API를 호출하도록 허용하는 IAM 역할에 태스크 역할을 제공할 수 있습니다. 자세한 정보는 Amazon ECS 작업 IAM 역할을 참조하세요.

Amazon ECS 최적화 Windows Server AMI를 시작하려면 Windows에서 태스크를 위한 IAM 역할에 `-EnableTaskIAMRole` 옵션을 설정해야 합니다. 이 기능을 활용하려면 컨테이너가 일부 구성 코드도 실행해야 합니다. 자세한 정보는 Amazon EC2 Windows 인스턴스 추가 구성을 참조하세요.

태스크 실행 역할

`executionRoleArn`

유형: 문자열

필수 항목 여부: 조건부

Amazon ECS 컨테이너 에이전트에 사용자를 대신하여 AWS API를 호출할 권한을 부여하는 태스크 실행 역할의 Amazon 리소스 이름(ARN)입니다.

> **Note**
>
> 태스크의 요구 사항에 따라 태스크 실행 IAM 역할이 필요합니다. 자세한 정보는 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

네트워크 모드

`networkMode`

유형: 문자열

필수 여부: 아니요

태스크의 컨테이너에 사용할 Docker 네트워킹 모드. Amazon EC2 Linux 인스턴스에서 호스팅되는 Amazon ECS의 경우, 유효한 값은 `none`, `bridge`, `awsvpc` 및 `host`입니다. 네트워크 모드를 지정하지 않으면 기본 네트워크 모드는 `bridge`입니다. Amazon EC2 Windows 인스턴스에서 호스팅되는 Amazon ECS 태스크의 경우, 유효한 값은 `default` 및 `awsvpc`입니다. 네트워크 모드가 지정되지 않으면 `default` 네트워크 모드를 사용합니다.

네트워크 모드가 `none`으로 설정된 경우에는 태스크의 컨테이너가 외부 네트워크와 연결되지 않고 컨테이너 정의에서 포트 매핑을 지정할 수 없습니다.

네트워크 모드가 `bridge`인 경우, 태스크는 태스크를 호스팅하는 각 Amazon EC2 인스턴스 내부에서 실행되는 Linux에서의 Docker 기본 가상 네트워크를 사용합니다. Linux의 기본 가상 네트워크는 `bridge` Docker 네트워크 드라이버를 사용합니다.

네트워크 모드가 `host`인 경우, 태스크는 해당 태스크를 호스팅하는 Amazon EC2 인스턴스의 ENI에 컨테이너 포트를 직접 매핑하여 Docker의 기본 가상 네트워크를 우회하는 호스트 네트워크를 사용합니다. 이 네트워크 모드에서는 동적 포트 매핑을 사용할 수 없습니다. 이 모드를 사용하는 작업 정의의 컨테이너는 특정`hostPort` 숫자를 지정해야 합니다. 호스트의 포트 번호는 여러 작업에서 사용할 수 없습니다. 따라서 단일 Amazon EC2 인스턴스에서 같은 태스크 정의를 가진 여러 태스크를 실행할 수 없습니다.

> **Important**
>
> `host` 네트워크 모드를 사용하여 태스크를 실행할 때는 더 나은 보안을 위해 루트 사용자(UID 0)를 사용하여 컨테이너를 실행해서는 안 됩니다. 보안상 항상 루트가 아닌 사용자를 사용합니다.

네트워크 모드가 awsvpc인 경우 해당 태스크에 탄력적 네트워크 인터페이스가 할당되므로 서비스를 생성하거나 태스크 정의로 태스크를 실행하려면 NetworkConfiguration을 지정해야 합니다. 자세한 내용은 EC2 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션 섹션을 참조하세요.

네트워크 모드가 default인 경우, 태스크는 태스크를 호스팅하는 각 Amazon EC2 인스턴스 내부에서 실행되는 Windows에서의 Docker 기본 가상 네트워크를 이용합니다. Windows의 기본 가상 네트워크는nat Docker 네트워크 드라이버를 사용합니다.

host 및 awsvpc 네트워크 모드는 Amazon EC2 네트워크 스택을 사용하는 컨테이너에 가장 높은 수준의 네트워킹 성능을 제공합니다. host 및 awsvpc 네트워크 모드에서는 노출된 컨테이너 포트가 해당 호스트 포트(host 네트워크 모드의 경우) 또는 연결된 탄력적 네트워크 인터페이스 포트(awsvpc 네트워크 의 경우)에 직접 매핑됩니다. 이 때문에 동적 호스트 포트 매핑을 사용할 수 없습니다.

기본 EC2 인스턴스의 운영 체제에 따라 네트워크 모드가 허용됩니다. 운영 체제가 Linux일 경우에는 모든 네트워크 모드를 사용할 수 있습니다. Windows의 경우, default 및 awsvpc 모드를 사용할 수 있습니다.

런타임 플랫폼

operatingSystemFamily

유형: 문자열

필수 항목 여부: 조건부

기본값: LINUX

태스크 정의를 등록할 때 운영 체제 제품군을 지정합니다.

유효한 값은LINUX, WINDOWS_SERVER_2025_FULL, WINDOWS_SERVER_2025_CORE, WINDOWS_SERVER_2022_CORE, WINDOWS_SERVER_2022_FULL, WINDOWS_SERVER_2019_FULL, WINDOWS_SERVER_2019_CORE, WINDOWS_SERVER_2016_FULL, WINDOWS_SERVER_2004_CORE, WINDOWS_SERVER_20H2_CORE입니다.

서비스에서 사용되는 모든 태스크 정의는 이 파라미터에 대해 동일한 값을 가져야 합니다.

태스크 정의가 서비스의 일부인 경우 이 값은 서비스 platformFamily 값과 일치해야 합니다.

`cpuArchitecture`

유형: 문자열

필수 항목 여부: 조건부

기본값: X86_64

태스크 정의를 등록할 때 CPU 아키텍처를 지정합니다. 유효 값은 `X86_64` 및 `ARM64`입니다.

서비스에서 사용되는 모든 태스크 정의는 이 파라미터에 대해 동일한 값을 가져야 합니다.

Linux 태스크가 있는 경우 값을 `ARM64`로 설정할 수 있습니다. 자세한 내용은 the section called “64비트 ARM 워크로드에 대한 작업 정의” 섹션을 참조하세요.

태스크 크기

태스크 정의를 등록할 때 태스크에 대해 사용할 CPU 및 메모리 총량을 지정할 수 있습니다. 이것은 컨테이너 정의 수준의 `cpu` 및 `memory` 값과는 구분됩니다. Amazon EC2 인스턴스에서 호스팅되는 태스크의 경우, 이 필드는 선택 사항입니다.

> **Note**
>
> Windows 컨테이너에 대해서는 태스크 레벨 CPU와 메모리 파라미터가 무시됩니다. Windows 컨테이너에 대해서는 컨테이너 레벨 리소스를 지정할 것을 권장합니다.

`cpu`

유형: 문자열

필수 항목 여부: 조건부

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

태스크에 대해 표시되는 CPU 단위의 하드 제한입니다. JSON 파일에서 CPU 값을 문자열(CPU 단위 또는 가상 CPU(vCPU))로 지정할 수 있습니다. 예를 들어 CPU 값을 `1024`(CPU 단위) 또는 `1`

vCPU(vCPU)로 지정할 수 있습니다. 태스크 정의를 등록할 때는 vCPU 값이 CPU 단위를 나타내는 정수로 변환됩니다.

이 필드는 선택 사항입니다. 클러스터에 요청된 CPU 단위가 사용 가능한 등록된 컨테이너 인스턴스가 없는 경우 태스크가 실패합니다. 지원되는 값은 `0.125` vCPU에서 192 vCPU 사이입니다.

`memory`

유형: 문자열

필수 항목 여부: 조건부

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

작업에 표시할 메모리의 하드 제한. 작업 정의에서 메모리 값을 문자열(메비바이트(MiB) 또는 기가바이트(GB) 단위)로 지정할 수 있습니다. 예를 들어 메모리 값을 `3072`(MiB) 또는 `3 GB`(GB)로 지정할 수 있습니다. 태스크 정의를 등록할 때는 GB 값이 MiB를 나타내는 정수로 변환됩니다.

이 필드는 선택 사항이며 아무 값이나 사용할 수 있습니다. 태스크 수준 메모리 값이 지정된 경우 컨테이너 수준 메모리 값은 선택 사항입니다. 클러스터에 요청된 메모리에 사용 가능한 등록된 컨테이너 인스턴스가 없는 경우 태스크가 실패합니다. 특정 인스턴스 유형에 대해 태스크에 가능한 한 많은 메모리를 제공하여 리소스 사용률을 최대화할 수 있습니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 메모리 예약 섹션을 참조하세요.

컨테이너 정의

작업 정의를 등록할 때 컨테이너 인스턴스의 Docker 대몬(daemon)으로 전달되는 컨테이너 정의의 목록을 지정해야 합니다. 컨테이너 정의에서 다음 파라미터가 허용됩니다.

주제

- 스탠다드 컨테이너 정의 파라미터
- 고급 컨테이너 정의 파라미터
- 기타 컨테이너 정의 파라미터

스탠다드 컨테이너 정의 파라미터

다음 태스크 정의 파라미터는 대부분의 컨테이너 정의에서 필요하거나 사용됩니다.

주제

- 명칭
- 이미지
- 메모리
- 포트 매핑
- 프라이빗 리포지토리 보안 인증

명칭

name

유형: 문자열

필수 항목 여부: 예

컨테이너의 이름. 최대 255개의 문자(대문자 및 소문자), 숫자, 하이픈 및 밑줄이 허용됩니다. 여러 컨테이너를 한 태스크 정의에 연결하는 경우, 한 컨테이너의 name을 다른 컨테이너의 links에 입력할 수 있습니다. 이는 컨테이너를 연결하기 위한 것입니다.

이미지

image

유형: 문자열

필수 항목 여부: 예

컨테이너를 시작하는 데 사용되는 이미지입니다. 이 문자열은 Docker 대몬(daemon)으로 직접 전달됩니다. 기본적으로 Docker Hub 레지스트리 내 이미지는 사용 가능합니다. *repository-url*/*image*:*tag* 또는 *repository-url*/*image*@*digest*를 사용하여 다른 리포지토리를 지정할 수도 있습니다. 최대 255개의 문자(대문자 및 소문자), 숫자, 하이픈, 밑줄, 콜론, 마침표, 슬래시 및 부호가 허용됩니다. 이 파라미터는 Docker 컨테이너 생성 명령에 있는 Image 및 docker run 명령의 IMAGE 파라미터에 매핑됩니다.

- 새 태스크가 시작될 때 Amazon ECS 컨테이너 에이전트가 컨테이너에서 사용할 지정 이미지와 태그의 최신 버전을 끌어옵니다. 그러나 이미 실행 중인 태스크에는 추후 리포지토리 이미지 업데이트가 전파되지 않습니다.
- 태스크 정의의 이미지 경로에 태그 또는 다이제스트를 지정하지 않으면 Amazon ECS 컨테이너 에이전트가 지정된 이미지의 최신 버전을 가져옵니다.
- 그러나 이미 실행 중인 태스크에는 추후 리포지토리 이미지 업데이트가 전파되지 않습니다.
- 프라이빗 리포지토리의 이미지는 지원되지 않습니다. 자세한 정보는 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용을 참조하세요.
- Amazon ECR 리포지토리의 이미지는 전체 `registry/repository:tag` 또는 `registry/repository@digest` 명명 규칙(예: *aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*:*latest* 또는 *aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*@*sha256:94afd1f2e64d908bc90dbca0035a5b567EXAMPLE*)을 사용하여 지정할 수 있습니다.
- Docker Hub 공식 리포지토리 안의 이미지는 단일 이름을 사용합니다(예: `ubuntu` 또는 `mongo`).
- Docker Hub 다른 리포지토리에 저장된 이미지는 조직 이름으로 한정됩니다(예: `amazon/amazon-ecs-agent`).
- 다른 온라인 리포지토리 안의 이미지는 도메인 이름을 사용하여 추가로 한정됩니다(예: `quay.io/assemblyline/ubuntu`).

`versionConsistency`

유형: 문자열

유효한 값: `enabled|disabled`

필수 여부: 아니요

컨테이너 정의에 제공된 컨테이너 이미지 태그가 Amazon ECS에서 이미지 다이제스트로 확인되는지 여부를 지정합니다. 기본적으로 이 동작은 `enabled`입니다. 컨테이너의 값을 `disabled`로 설정하면 Amazon ECS에서는 컨테이너 이미지 태그가 다이제스트로 확인되지 않고 컨테이너 정의에 지정된 원본 이미지 URI가 배포에 사용됩니다. 컨테이너 이미지 확인에 대한 자세한 내용은 컨테이너 이미지 확인 단원을 참조하세요.

메모리

`memory`

유형: 정수

필수 여부: 아니요

컨테이너에 표시할 메모리의 양(MiB)입니다. 컨테이너가 여기서 지정된 메모리를 초과하려 하면 해당 컨테이너가 중지됩니다. 태스크 내 모든 컨테이너에 대해 예약된 총 메모리 양은 태스크 `memory` 값(지정된 경우)보다 작아야 합니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `Memory` 및 docker run에 대한 `--memory` 옵션에 매핑됩니다.

작업 레벨 메모리 값 또는 컨테이너 레벨 메모리 값을 지정해야 합니다. 컨테이너 수준 `memory` 및 `memoryReservation` 값을 모두 지정하는 경우 `memory` 값이 `memoryReservation` 값보다 커야 합니다. `memoryReservation`을 지정하는 경우 컨테이너가 배치된 컨테이너 인스턴스의 가용 메모리 리소스에서 해당 값이 차감됩니다. 그렇지 않으면 `memory` 값이 사용됩니다.

Docker 20.10.0 이상의 대몬(daemon)은 컨테이너용으로 최소 6MiB의 메모리를 남겨둡니다. 따라서 컨테이너에 6MiB 미만의 메모리를 지정하지 마세요.

Docker 19.03.13-ce 이하의 대몬(daemon)은 컨테이너용으로 최소 4MiB의 메모리를 남겨둡니다. 따라서 컨테이너에 4MiB 미만의 메모리를 지정하지 마세요.

> **Note**
>
> 특정 인스턴스 유형에 대해 태스크에 가능한 한 많은 메모리를 제공하여 리소스 사용률을 최대화하려는 경우 Amazon ECS Linux 컨테이너 인스턴스 메모리 예약 섹션을 참조하세요.

`memoryReservation`

유형: 정수

필수 여부: 아니요

컨테이너용으로 예약할 메모리의 소프트 제한(MiB)입니다. 시스템 메모리가 경합하는 경우 Docker는 컨테이너 메모리를 이 소프트 제한으로 유지하려고 합니다. 하지만 필요한 경우 컨테이너에서 더 많은 메모리를 사용할 수 있습니다. 컨테이너는 `memory` 파라미터에 지정된 하드 한도까지(해

당하는 경우) 또는 컨테이너 인스턴스의 모든 가용 메모리(둘 중 먼저 발생하는 것)를 사용할 수 있습니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `MemoryReservation` 및 docker run에 대한 `--memory-reservation` 옵션에 매핑됩니다.

태스크 레벨 메모리 값이 지정되지 않은 경우 컨테이너 정의의 `memory` 또는 `memoryReservation`에 대해 하나 또는 둘 다에 0이 아닌 정수를 지정해야 합니다. 둘 모두 지정하는 경우 `memory`가 `memoryReservation`보다 커야 합니다. `memoryReservation`을 지정하는 경우 컨테이너가 배치된 컨테이너 인스턴스의 가용 메모리 리소스에서 해당 값이 차감됩니다. 그렇지 않으면 `memory` 값이 사용됩니다.

예를 들어 컨테이너가 통상적으로 128MiB의 메모리를 사용하지만 경우에 따라 잠깐 동안 사용량이 256MiB까지 급증한다고 가정합니다. 이 경우 `memoryReservation`을 128MiB로 설정하고 `memory` 하드 제한을 300MiB로 설정할 수 있습니다. 이 구성은 컨테이너가 컨테이너 인스턴스의 잔여 리소스 중 128MiB의 메모리만 예약하도록 허용합니다. 동시에, 이 구성을 사용하여 필요할 때 컨테이너가 더 많은 메모리 리소스를 사용하도록 허용할 수도 있습니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

Docker 20.10.0 이상의 대몬(daemon)은 컨테이너용으로 최소 6MiB의 메모리를 남겨둡니다. 따라서 컨테이너에 6MiB 미만의 메모리를 지정하지 마세요.

Docker 19.03.13-ce 이하의 대몬(daemon)은 컨테이너용으로 최소 4MiB의 메모리를 남겨둡니다. 따라서 컨테이너에 4MiB 미만의 메모리를 지정하지 마세요.

> **Note**
>
> 특정 인스턴스 유형에 대해 태스크에 가능한 한 많은 메모리를 제공하여 리소스 사용률을 최대화하려는 경우 Amazon ECS Linux 컨테이너 인스턴스 메모리 예약 섹션을 참조하세요.

포트 매핑

`portMappings`

유형: 객체 배열

필수 여부: 아니요

포트 매핑은 컨테이너의 네트워크 포트를 외부에 노출시켜 클라이언트가 애플리케이션에 액세스할 수 있도록 합니다. 이 파라미터는 동일한 태스크 내에서 컨테이너 간 통신에도 사용됩니다.

awsvpc 네트워크 모드를 사용하는 태스크 정의의 경우 containerPort만 지정합니다. hostPort는 항상 무시되며 컨테이너 포트는 호스트에서 임의의 높은 번호의 포트에 자동으로 매핑됩니다.

Windows에서의 포트 매핑은 localhost가 아니라 NetNAT 게이트웨이 주소를 사용합니다. Windows에서의 포트 매핑에는 루프백이 없으므로 호스트 자체에서 컨테이너의 매핑된 포트에 액세스할 수 없습니다.

이 파라미터의 필드 대부분(containerPort, hostPort, protocol 포함)은 Docker 컨테이너 생성 명령에 있는 PortBindings와 docker run에 대한 --publish 옵션에 매핑됩니다. 태스크 정의의 네트워크 모드가 host로 설정되어 있으면 호스트 포트가 정의되지 않은 상태이거나 호스트 포트가 포트 매핑의 컨테이너 포트와 일치해야 합니다.

> **Note**
>
> 태스크가 RUNNING 상태에 도달한 후에는 다음 위치에서 수동 및 자동 호스트/컨테이너 포트 할당을 확인할 수 있습니다.
>
> - 콘솔: 선택한 태스크에 대한 컨테이너 설명의 네트워크 바인딩 섹션.
> - AWS CLI: describe-tasks 명령 출력의 networkBindings 섹션.
> - API: DescribeTasks 응답.
> - 메타데이터: 작업 메타데이터 엔드포인트.

appProtocol

유형: 문자열

필수 여부: 아니요

포트 매핑에 사용되는 애플리케이션 프로토콜입니다. 이 파라미터는 Service Connect에만 적용됩니다. 애플리케이션에서 사용하는 프로토콜과 일치하도록 이 파라미터를 설정하는 것이 좋습니다. 이 파라미터를 설정하면 Amazon ECS가 서비스 연결 프록시에 프로토콜별 연결 처리를 추가합니다. 이 파라미터를 설정하면 Amazon ECS가 Amazon ECS 콘솔과 CloudWatch에 프로토콜별 원격 분석을 추가합니다.

이 파라미터의 값을 설정하지 않으면 TCP가 사용됩니다. 그러나 Amazon ECS는 TCP에 대한 프로토콜별 원격 측정을 추가하지 않습니다.

자세한 내용은 the section called “ Service Connect” 섹션을 참조하세요.

유효한 프로토콜 값: `"http" | "http2" | "grpc"`

`containerPort`

유형: 정수

필수 항목 여부: 예(`portMappings` 사용 시)

사용자 지정 또는 자동 할당된 호스트 포트에 바인딩되는 컨테이너 포트 번호입니다.

`awsvpc` 네트워크 모드를 사용하는 태스크의 경우 `containerPort`를 사용하여 노출된 포트를 지정합니다.

EC2 시작 유형의 작업에서 컨테이너를 사용하고 컨테이너 포트는 지정하고 호스트 포트는 지정하지 않는다고 가정해 보겠습니다. 그러면 컨테이너는 자동으로 임시 포트 범위의 호스트 포트를 수신합니다. 자세한 내용은 `hostPort` 섹션을 참조하세요. 이렇게 자동 할당된 포트 매핑은 컨테이너 인스턴스의 예약 포트 할당량 100개에 포함되지 않습니다.

`containerPortRange`

유형: 문자열

필수 여부: 아니요

동적으로 매핑된 호스트 포트 범위에 바인딩된 컨테이너의 포트 번호 범위입니다.

이 파라미터는 `register-task-definition` API를 사용해서만 설정할 수 있습니다. 옵션은 `portMappings` 파라미터에서 사용할 수 있습니다. 자세한 정보는 AWS Command Line Interface 참조의 register-task-definition을 참조하세요.

`containerPortRange` 지정 시 다음 규칙이 적용됩니다.

- `bridge` 네트워크 모드 또는 `awsvpc` 네트워크 모드를 사용해야 합니다.
- 이 파라미터는 Linux 및 Windows 운영 체제 모두에서 사용할 수 있습니다.
- 컨테이너 인스턴스의 컨테이너 에이전트 버전은 1.67.0 이상이어야 하고 `ecs-init` 패키지의 버전은 1.67.0-1 이상이어야 합니다.
- 컨테이너당 최대 100개의 포트 범위를 지정할 수 있습니다.
- `hostPortRange`는 지정하지 않습니다. `hostPortRange`의 값은 다음과 같이 설정됩니다.

  - `awsvpc` 네트워크 모드를 사용하는 작업에 있는 컨테이너의 경우 `hostPort`는 `containerPort`와 동일한 값으로 설정됩니다. 이는 정적 매핑 전략입니다.
  - `bridge` 네트워크 모드를 사용하는 작업에 있는 컨테이너의 경우 Amazon ECS 에이전트는 기본 임시 범위에서 열린 호스트 포트를 찾고 이를 도커로 전달하여 컨테이너 포트에 바인딩합니다.
- `containerPortRange`의 유효한 값은 1~65,535입니다.
- 포트는 컨테이너당 하나의 포트 매핑에만 포함될 수 있습니다.
- 중첩되는 포트 범위는 지정할 수 없습니다.
- 범위의 첫 번째 포트는 범위의 마지막 포트보다 작아야 합니다.
- Docker에서는 포트 수가 많을 때 Docker 대몬(daemon) 구성 파일에서 docker-proxy를 끌 것을 권장합니다.

  자세한 내용은 GitHub의 Issue #11185를 참조하세요.

  Docker 대몬(daemon) 구성 파일에서 docker-proxy를 끄는 방법에 대한 자세한 내용은 Amazon ECS 개발자 안내서의 Docker daemon을 참조하세요.

DescribeTasks를 직접적으로 호출하여 컨테이너 포트에 바인딩된 호스트 포트인 `hostPortRange`를 볼 수 있습니다.

포트 범위는 EventBridge로 전송되는 Amazon ECS 작업 이벤트에 포함되지 않습니다. 자세한 내용은 the section called “EventBridge를 사용하여 Amazon ECS 오류에 대한 응답 자동화” 섹션을 참조하세요.

`hostPortRange`

유형: 문자열

필수 여부: 아니요

네트워크 바인딩에 사용되는 호스트의 포트 번호 범위입니다. 이 값은 Docker가 할당하고 Amazon ECS 에이전트가 전달합니다.

`hostPort`

유형: 정수

필수 여부: 아니요

컨테이너용으로 예약할 컨테이너 인스턴스 포트 번호입니다.

이 경우 컨테이너 포트 매핑을 위해 예약되지 않은 호스트 포트를 지정할 수 있습니다. 이를 정적 호스트 포트 매핑이라고 합니다. 또는 `hostPort`는 생략(또는 `0`으로 설정)하고 `containerPort`는 지정할 수 있습니다. 컨테이너에서는 컨테이너 인스턴스 운영 체제 및 Docker 버전에 대한 임시 포트 범위의 포트를 자동으로 수신합니다. 이를 동적 호스트 포트 매핑이라고 합니다.

기본 임시 포트 범위 Docker 1.6.0 버전 이상은 인스턴스의 `/proc/sys/net/ipv4/ip_local_port_range`에 나열됩니다. 이 커널 파라미터를 사용할 수 없을 경우 기본 휘발성 포트 범위 `49153-65535`가 사용됩니다. 휘발성 포트 범위에서 호스트 포트를 지정하지 마세요. 자동 할당을 위해 예약되어 있기 때문입니다. 일반적으로 32768 미만의 포트는 임시 포트 범위를 벗어납니다.

기본 예약 포트는 SSH용 22, Docker 포트 2375 및 2376, Amazon ECS 컨테이너 에이전트 포트 `51678-51680`입니다. 실행 중인 작업에 대해 기존에 사용자 지정된 호스트 포트도 해당 작업이 실행되는 동안 예약됩니다. 작업이 중지하면 호스트 포트는 해제됩니다. 현재 예약된 포트는 describe-container-instances 출력의 `remainingResources`에 표시됩니다. 하나의 컨테이너 인스턴스는 한 번에 최대 100개의 예약 포트(기본 예약 포트 포함)를 가질 수 있습니다. 자동 할당된 포트는 예약 포트 할당량 100개에 포함되지 않습니다.

`name`

유형: 문자열

필수: 아니요, 서비스에서 Service Connect와 VPC Lattice를 구성하는 데 필요

포트 매핑에 사용되는 이름입니다. 이 파라미터는 Service Connect와 VPC Lattice에만 적용됩니다. 이 파라미터는 서비스의 Service Connect와 VPC Lattice 구성에서 사용하는 이름입니다.

자세한 내용은 Service Connect를 사용하여 짧은 이름으로 Amazon ECS 서비스 연결 섹션을 참조하세요.

다음 예제에서는 Service Connect와 VPC Lattice의 필수 필드를 모두 사용합니다.

```
"portMappings": [
    {
        "name": string,
        "containerPort": integer
    }
]
```

`protocol`

유형: 문자열

필수 여부: 아니요

포트 매핑에 사용되는 프로토콜입니다. 유효 값은 `tcp`및 `udp`입니다. 기본값은 `tcp`입니다.

> **Important**
>
> Service Connect에는 `tcp`만 지원됩니다. 이 필드가 설정되지 않은 경우 `tcp`로 암시됩니다.

> **Important**
>
> UDP 지원은 버전 1.2.0 이상의 Amazon ECS 컨테이너 에이전트(예: `amzn-ami-2015.03.c-amazon-ecs-optimized` AMI) 또는 버전 1.3.0 이상으로 업데이트된 컨테이너 에이전트를 사용하여 시작된 컨테이너 인스턴스에서만 가능합니다. 컨테이너 에이전트를 최신 버전으로 업데이트하려면 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

호스트 포트를 지정하는 경우 다음 구문을 사용합니다.

```
"portMappings": [
    {
        "containerPort": integer,
        "hostPort": integer
    }
    ...
]
```

호스트 포트가 자동 할당되도록 하려면 다음 구문을 사용합니다.

```
"portMappings": [
    {
        "containerPort": integer
    }
    ...
```

```
]
```

프라이빗 리포지토리 보안 인증

`repositoryCredentials`

유형: RepositoryCredentials 객체

필수 여부: 아니요

프라이빗 레지스트리 인증을 위한 리포지토리 보안 인증입니다.

자세한 내용은 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용 섹션을 참조하세요.

`credentialsParameter`

유형: 문자열

필수 항목 여부: 예(`repositoryCredentials` 사용 시)

프라이빗 리포지토리 자격 증명이 포함된 암호의 Amazon 리소스 이름(ARN)입니다.

자세한 내용은 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용 섹션을 참조하세요.

> **Note**
>
> Amazon ECS API, AWS CLI 또는 AWS SDK를 사용할 때 시작하는 작업과 같은 리전에 보안 암호가 있는 경우 보안 암호의 전체 ARN 또는 이름을 사용할 수 있습니다. AWS Management Console을 사용하는 경우 보안 암호의 전체 ARN을 지정해야 합니다.

다음은 필요한 파라미터를 나타낸 태스크 정의의 예제 조각입니다.

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
```

```
    }
]
```

고급 컨테이너 정의 파라미터

다음의 고급 컨테이너 정의 파라미터는 Amazon ECS 컨테이너 인스턴스에서 컨테이너를 시작하는 데 사용된 docker run 명령에 확장된 기능을 제공합니다.

주제

- 재시작 정책
- 상태 확인
- 환경
- 네트워크 설정
- 스토리지 및 로깅
- 보안
- 리소스 제한
- Docker 레이블

재시작 정책

`restartPolicy`

컨테이너 재시작 정책 및 관련 구성 파라미터. 컨테이너의 재시작 정책을 설정하면 작업을 교체할 필요 없이 Amazon ECS에서 컨테이너를 다시 시작할 수 있습니다. 자세한 내용은 컨테이너 재시작 정책이 있는 Amazon ECS 작업의 개별 컨테이너 재시작 섹션을 참조하세요.

`enabled`

유형: 부울

필수 여부: 예

재시작 정책이 컨테이너에 대해 활성화되는지 여부를 지정합니다.

`ignoredExitCodes`

유형: 정수 배열

필수 여부: 아니요

Amazon ECS가 재시작을 시도하지 않고 무시하는 종료 코드 목록입니다. 최대 50개의 컨테이너 종료 코드를 지정할 수 있습니다. 기본적으로 Amazon ECS는 종료 코드를 무시하지 않습니다.

`restartAttemptPeriod`

유형: 정수

필수 여부: 아니요

재시작을 시도하기 전에 컨테이너를 실행해야 하는 시간(초)입니다. 컨테이너는 `restartAttemptPeriod`초 당 한 번만 재시작할 수 있습니다. 이 기간 동안 컨테이너를 실행할 수 없고 일찍 종료되면 다시 시작되지 않습니다. 최소 60초의 `restartAttemptPeriod`와 최대 1,800초의 `restartAttemptPeriod`를 설정할 수 있습니다. 기본적으로 컨테이너는 300초 동안 실행되어야 재시작할 수 있습니다.

상태 확인

`healthCheck`

해당 컨테이너에 대한 컨테이너 상태 확인 명령 및 연결된 구성 파라미터. 자세한 내용은 컨테이너 상태 확인을 사용하여 Amazon ECS 작업 상태 확인 섹션을 참조하세요.

`command`

상태가 정상인지를 확인하기 위해 컨테이너가 실행하는 명령을 나타내는 문자열 배열입니다. 명령 인수를 직접 실행하려면 문자열 어레이가 `CMD`으로 시작하면 되고, 컨테이너의 기본 셸에서 명령을 실행하려면 `CMD-SHELL`로 시작하면 됩니다. 둘 다 지정하지 않으면 `CMD`가 사용됩니다.

AWS Management Console에서 작업 정의를 등록할 때는 쉼표로 구분된 명령 목록을 사용합니다. 이러한 명령은 작업 정의가 생성된 후 문자열로 변환됩니다. 다음은 상태 확인에 대한 입력 예제입니다.

```
CMD-SHELL, curl -f http://localhost/ || exit 1
```

AWS Management Console JSON 패널, AWS CLI 또는 API를 사용하여 작업 정의를 등록할 때는 명령 목록을 괄호로 묶어야 합니다. 다음은 상태 확인에 대한 입력 예제입니다.

```
[ "CMD-SHELL", "curl -f http://localhost/ || exit 1" ]
```

stderr 출력이 없는 종료 코드 0은 성공을 나타내고, 0이 아닌 종료 코드는 실패를 나타냅니다.

`interval`

각 상태 확인 간의 시간(초). 5초에서 300초 사이를 지정할 수 있습니다. 기본값은 30초입니다.

`timeout`

실패로 간주되기 전에 상태 확인이 성공하기까지의 대기 시간(초)입니다. 2초에서 60초 사이를 지정할 수 있습니다. 기본 값은 5초입니다.

`retries`

컨테이너 상태가 비정상이라고 간주되기 전에 실패한 상태 확인을 재시도하는 횟수입니다. 1에서 10 사이의 재시도 횟수를 지정할 수 있습니다. 기본값은 3회 재시도입니다.

`startPeriod`

실패한 상태 확인이 최대 재시도 횟수에 포함되기 전에 컨테이너에 부트스트랩 시간이 제공되는 유예 기간 옵션입니다. 0초부터 300초까지 지정할 수 있습니다. 기본적으로 `startPeriod`는 비활성화되어 있습니다.

상태 검사가 `startPeriod` 내에 성공한 경우 컨테이너는 정상으로 간주되고 모든 후속 실패는 최대 재시도 횟수에 포함됩니다.

환경

`cpu`

유형: 정수

필수 여부: 아니요

Amazon ECS 컨테이너 에이전트가 컨테이너에 대해 예약하는 cpu 단위입니다. Linux에서 이 파라미터는 컨테이너 만들기 섹션의 `CpuShares`에 매핑됩니다.

> **Note**
>
> 각 Amazon EC2 인스턴스 유형에 사용 가능한 CPU 단위 수를 결정할 수 있습니다. 이렇게 하려면 Amazon EC2 인스턴스 세부 정보 페이지에서 해당 인스턴스 유형에 대해 나열된 vCPU 수에 1,024를 곱합니다.

Linux 컨테이너는 할당된 양과 동일한 비율로, 컨테이너 인스턴스의 다른 컨테이너와 할당되지 않은 CPU 단위를 공유합니다. 예를 들어 싱글 코어 인스턴스 유형에서 단일 컨테이너 작업을 컨테이너에 512개 CPU 단위를 지정하고 실행한다고 가정해 보겠습니다. 또한 이 작업은 컨테이너 인스턴스에서 실행되는 유일한 작업입니다. 이 예에서 컨테이너는 언제라도 1,024 CPU 단위 전부를 사용할 수 있습니다. 하지만, 해당 컨테이너 인스턴스에서 동일한 태스크의 다른 사본을 시작했다고 가정합니다. 각 작업에는 필요한 경우 최소 512개의 CPU 단위가 보장됩니다. 마찬가지로 다른 컨테이너가 나머지 CPU를 사용하지 않을 경우 각 컨테이너의 CPU 사용량이 높아질 수 있습니다. 하지만 두 작업이 항상 100% 활성 상태인 경우 CPU 단위는 512개로 제한됩니다.

Linux 컨테이너 인스턴스에서, 컨테이너 인스턴스의 Docker 대몬(daemon)은 CPU 값을 사용하여 실행 컨테이너의 상대적인 CPU 공유 비율을 계산합니다. Linux 커널이 허용하는 최소 유효 CPU 공유 값은 2이고, Linux 커널이 허용하는 최대 유효 CPU 공유 값은 262144입니다. 하지만 CPU 파라미터는 필수 항목이 아니며 컨테이너 정의에서 2 미만이나 262144를 초과하는 CPU 값을 사용할 수 있습니다. CPU 값이 2 미만(null 포함)이거나 262144를 초과하는 경우 동작은 Amazon ECS 컨테이너 에이전트 버전에 따라 달라집니다.

- 에이전트 버전 <= 1.1.0: Null 및 0의 CPU 값이 Docker에 0으로 전달되고, Docker는 이를 1,024개 CPU 공유로 변환합니다. 1의 CPU 값이 Docker에 1로 전달되고, Linux 커널이 이를 2개 CPU 공유로 변환합니다.
- 에이전트 버전 >= 1.2.0: Null, 0 및 1의 CPU 값이 Docker에 2개 CPU 공유로 전달됩니다.
- 에이전트 버전 >= 1.84.0: 256 vCPU보다 큰 CPU 값은 Docker에 256으로 전달되며, 이는 262144개의 CPU 공유에 해당합니다.

Windows 컨테이너 인스턴스에서 CPU 할당량은 절대 할당량으로 적용됩니다. Windows 컨테이너는 작업 정의에 정의된 지정된 양의 CPU에만 액세스할 수 있습니다. Null 또는 0의 CPU 값이 Docker에 0으로 전달되고, Windows는 이 값을 1개 CPU의 1%로 해석합니다.

추가 예제는 How Amazon ECS manages CPU and memory resources를 참조하세요.

gpu

유형: ResourceRequirement 객체

필수 여부: 아니요

Amazon ECS 컨테이너 에이전트가 컨테이너에 대해 예약하는 실제 GPUs 수입니다. 태스크의 모든 컨테이너에 예약된 GPU 수가 태스크가 실행되는 컨테이너 인스턴스에서 사용할 수 있는 GPU 수를 초과하면 안 됩니다. 자세한 내용은 GPU 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서는 지원되지 않습니다.

`Elastic Inference accelerator`

유형: ResourceRequirement 객체

필수 여부: 아니요

`InferenceAccelerator` 유형의 경우 `value`는 태스크 정의에 지정된 `InferenceAccelerator`의 `deviceName`과 일치합니다. 자세한 내용은 the section called "Elastic Inference 액셀러레이터 이름" 섹션을 참조하세요.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서는 지원되지 않습니다.

`essential`

유형: 부울

필수 여부: 아니요

컨테이너의 `essential` 파라미터가 `true`로 표시되어 있고 해당 컨테이너가 어떤 이유로든 실패 또는 중지하는 경우를 가정해 보겠습니다. 이 경우 작업에 포함된 다른 모든 컨테이너도 중지합니다. 컨테이너의 `essential` 파라미터가 `false`로 표시된 경우에는 해당 컨테이너의 실패가 태스크의 나머지 컨테이너에 영향을 주지 않습니다. 이 파라미터가 생략된 경우 컨테이너가 필수로 간주됩니다.

모든 태스크에는 하나 이상의 필수 컨테이너가 있어야 합니다. 여러 컨테이너로 구성된 애플리케이션이 있다고 가정해 보겠습니다. 이 경우, 공통 용도로 사용되는 컨테이너를 구성 요소로 그룹화하고, 다른 구성 요소를 여러 작업 정의로 분리합니다. 자세한 내용은 Amazon ECS에 대한 애플리케이션 설계 섹션을 참조하세요.

```
"essential": true|false
```

`entryPoint`

> **Important**
>
> 초기 버전의 Amazon ECS 컨테이너 에이전트는 `entryPoint` 파라미터를 적절히 처리하지 못합니다. `entryPoint`를 사용하는 데 문제가 있을 경우 컨테이너 에이전트를 업데이트하거나 명령 및 인수를 `command` 배열 항목으로 입력하세요.

유형: 문자열 배열

필수 여부: 아니요

컨테이너로 전달되는 진입점입니다.

```
"entryPoint": ["string", ...]
```

`command`

유형: 문자열 배열

필수 여부: 아니요

컨테이너로 전달되는 명령입니다. 이 파라미터는 컨테이너 생성 명령의 `Cmd` 및 docker run에 대한 `COMMAND` 파라미터에 매핑됩니다. 여러 인수가 있는 경우 각 인수는 배열에서 각각 분리된 문자열이어야 합니다.

```
"command": ["string", ...]
```

`workingDirectory`

유형: 문자열

필수 여부: 아니요

컨테이너에서 명령을 실행할 태스크 디렉터리입니다. 이 파라미터는 Docker 원격 API의 컨테이너 생성 섹션에 있는 `WorkingDir`와 docker run에 대한 `--workdir` 옵션에 매핑됩니다.

```
"workingDirectory": "string"
```

`environmentFiles`

유형: 객체 배열

필수 여부: 아니요

컨테이너로 전달할 환경 변수를 포함하는 파일 목록입니다. 이 파라미터는 docker run 명령에 대한 `--env-file` 옵션에 매핑됩니다.

FIPS가 활성화된 경우 마침표(.)가 포함된 버킷 이름(예: amzn-s3-demo-bucket1.name.example)은 지원되지 않습니다. 버킷 이름에 마침표(.)가 있으면 에이전트가 Amazon S3에서 환경 변수 파일을 가져올 수 없기 때문에 태스크가 시작되지 않습니다.

Windows 컨테이너에는 이 파라미터를 사용할 수 없습니다.

최대 10개의 환경 파일을 지정할 수 있습니다. 파일의 확장명은 `.env`여야 합니다. 환경 파일의 각 줄에는 `VARIABLE=VALUE` 형식의 환경 변수를 포함합니다. #으로 시작하는 줄은 주석으로 처리되며 무시됩니다.

컨테이너 정의에 지정된 개별 환경 변수가 있는 경우 환경 파일 내에 포함된 변수보다 우선합니다. 동일한 변수를 포함하는 여러 환경 파일이 지정된 경우 위에서 아래로 처리됩니다. 고유한 변수 이름을 사용하는 것이 좋습니다. 자세한 정보는 개별 환경 변수를 Amazon ECS 컨테이너로 전달을 참조하세요.

`value`

유형: 문자열

필수 항목 여부: 예

환경 변수 파일을 포함하는 Amazon S3 객체의 Amazon 리소스 이름(ARN)입니다.

`type`

유형: 문자열

필수 항목 여부: 예

사용할 파일 유형입니다. 지원되는 유일한 값은 `s3`입니다.

`environment`

유형: 객체 배열

필수 여부: 아니요

컨테이너로 전달할 환경 변수입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 Env 및 docker run 명령에 대한 `--env` 옵션에 매핑됩니다.

> **Important**
>
> 자격 증명 데이터와 같은 민감한 정보에 대해서는 일반 텍스트 환경 변수를 사용하지 않는 것이 좋습니다.

`name`

유형: 문자열

필수 항목 여부: `environment` 사용 시, 예

환경 변수의 이름입니다.

`value`

유형: 문자열

필수 항목 여부: `environment` 사용 시, 예

환경 변수의 값입니다.

```
"environment" : [
    { "name" : "string", "value" : "string" },
    { "name" : "string", "value" : "string" }
]
```

`secrets`

유형: 객체 배열

필수 여부: 아니요

컨테이너에 공개할 보안 암호를 나타내는 객체입니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.

`name`

유형: 문자열

필수 항목 여부: 예

컨테이너에서 환경 변수로 설정할 값입니다.

`valueFrom`

유형: 문자열

필수 항목 여부: 예

컨테이너에 노출될 암호입니다. 지원되는 값은 AWS Secrets Manager 암호의 전체 ARN이거나 혹은 AWS Systems Manager 파라미터 스토어 내 파라미터의 전체 Amazon 리소스 이름(ARN)입니다.

> **Note**
>
> Systems Manager Parameter Store 파라미터 또는 Secrets Manager 파라미터가 현재 실행 중인 작업과 동일한 AWS 리전에 있을 경우 해당 암호의 전체 ARN 또는 이름을 사용할 수 있습니다. 파라미터가 다른 리전에 있다면 전체 ARN을 지정해야 합니다.

```
"secrets": [
    {
        "name": "environment_variable_name",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }
]
```

네트워크 설정

`disableNetworking`

유형: 부울

필수 여부: 아니요

이 파라미터가 true일 경우 컨테이너 내에서 네트워킹이 해제됩니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너 또는 awsvpc 네트워크 모드를 사용하는 작업에 대해서는 지원되지 않습니다.

기본값은 false입니다.

```
"disableNetworking": true|false
```

links

유형: 문자열 배열

필수 여부: 아니요

link 파라미터는 포트 매핑 필요 없이 컨테이너가 서로 통신하도록 허용합니다. 이 파라미터는 작업 정의의 네트워크 모드가 bridge로 설정된 경우에만 지원됩니다. name:internalName 구성은 Docker 링크의 name:alias와 유사합니다. 최대 255개의 문자(대문자 및 소문자), 숫자, 하이픈 및 밑줄이 허용됩니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너 또는 awsvpc 네트워크 모드를 사용하는 작업에 대해서는 지원되지 않습니다.

> **Important**
>
> 동일한 컨테이너 인스턴스에 배치된 컨테이너는 링크 또는 호스트 포트 매핑 없이도 서로 통신할 수 있습니다. 컨테이너 인스턴스에서 네트워크 격리는 보안 그룹 및 VPC 설정에 의해 제어됩니다.

```
"links": ["name:internalName", ...]
```

hostname

유형: 문자열

필수 여부: 아니요

컨테이너에 사용할 호스트 이름입니다. 이 파라미터는 Docker 컨테이너 생성의 Hostname 및 docker run에 대한 --hostname 옵션에 매핑됩니다.

> **Note**
>
> awsvpc 네트워크 모드를 사용 중인 경우에는 hostname 파라미터가 지원되지 않습니다.

```
"hostname": "string"
```

dnsServers

유형: 문자열 배열

필수 여부: 아니요

컨테이너에 제공되는 DNS 서버의 목록입니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너 또는 awsvpc 네트워크 모드를 사용하는 작업에 대해서는 지원되지 않습니다.

```
"dnsServers": ["string", ...]
```

dnsSearchDomains

유형: 문자열 배열

필수 여부: 아니요

패턴: ^[a-zA-Z0-9-.]{0,253}[a-zA-Z0-9]$

컨테이너에 제공되는 DNS 검색 도메인의 목록입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 DnsSearch 및 docker run에 대한 --dns-search 옵션에 매핑됩니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너 또는 awsvpc 네트워크 모드를 사용하는 태스크에 대해서는 지원되지 않습니다.

```
"dnsSearchDomains": ["string", ...]
```

`extraHosts`

유형: 객체 배열

필수 여부: 아니요

컨테이너의 `/etc/hosts` 파일에 추가할 호스트 이름 및 IP 주소 매핑의 목록입니다.

이 파라미터는 Docker 컨테이너 생성 명령의 `ExtraHosts` 및 docker run에 대한 `--add-host` 옵션에 매핑됩니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너 또는 `awsvpc` 네트워크 모드를 사용하는 태스크에 대해서는 지원되지 않습니다.

```
"extraHosts": [
      {
        "hostname": "string",
        "ipAddress": "string"
      }
      ...
    ]
```

`hostname`

유형: 문자열

필수 항목 여부: 예(`extraHosts` 사용 시)

`/etc/hosts` 항목에 사용할 호스트 이름입니다.

`ipAddress`

유형: 문자열

필수 항목 여부: 예(`extraHosts` 사용 시)

`/etc/hosts` 항목에 사용할 IP 주소입니다.

스토리지 및 로깅

`readonlyRootFilesystem`

유형: 부울

필수 여부: 아니요

이 파라미터가 true일 경우 컨테이너에는 루트 파일 시스템에 대한 읽기 전용 액세스가 부여됩니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `ReadonlyRootfs` 및 docker run에 대한 `--read-only` 옵션에 매핑됩니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

기본값은 `false`입니다.

```
"readonlyRootFilesystem": true|false
```

`mountPoints`

유형: 객체 배열

필수 여부: 아니요

컨테이너에서 데이터 볼륨의 탑재 지점입니다. 이 파라미터는 create-container Docker API의 `Volumes` 및 docker run에 대한 `--volume` 옵션에 매핑됩니다.

Windows 컨테이너는 전체 디렉터리를 동일한 드라이브에 `$env:ProgramData`로 마운트할 수 있습니다. Windows 컨테이너는 디렉터리를 다른 드라이브에 탑재할 수 없으며, 탑재 지점은 여러 드라이브에 걸쳐 사용할 수 없습니다. Amazon EBS 볼륨을 Amazon ECS 작업에 직접 연결하려면 탑재 지점을 지정해야 합니다.

`sourceVolume`

유형: 문자열

필수 항목 여부: 예(`mountPoints` 사용 시)

탑재할 볼륨의 이름입니다.

containerPath

유형: 문자열

필수 항목 여부: 예(mountPoints 사용 시)

볼륨을 탑재할 컨테이너의 경로입니다.

readOnly

유형: 부울

필수 여부: 아니요

이 값이 true일 경우 컨테이너에는 볼륨에 대한 읽기 전용 액세스가 부여됩니다. 이 값이 false일 경우 컨테이너는 볼륨에 쓸 수 있습니다. 기본값은 false입니다.

Windows 운영 체제를 실행하는 태스크의 경우 값을 기본값인 false로 둡니다.

volumesFrom

유형: 객체 배열

필수 여부: 아니요

다른 컨테이너로부터 탑재할 데이터 볼륨입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 VolumesFrom 및 docker run에 대한 --volumes-from 옵션에 매핑됩니다.

sourceContainer

유형: 문자열

필수 항목 여부: volumesFrom 사용 시, 예

탑재할 볼륨이 위치한 컨테이너의 이름입니다.

readOnly

유형: 부울

필수 여부: 아니요

이 값이 true일 경우 컨테이너에는 볼륨에 대한 읽기 전용 액세스가 부여됩니다. 이 값이 false일 경우 컨테이너는 볼륨에 쓸 수 있습니다. 기본값은 false입니다.

```
"volumesFrom": [
```

```
            {
              "sourceContainer": "string",
              "readOnly": true|false
            }
          ]
```

`logConfiguration`

유형: LogConfiguration 객체

필수 여부: 아니요

컨테이너의 로그 구성 사양입니다.

로그 구성을 사용하는 태스크 정의에 대한 자세한 정보는 Amazon ECS 작업 정의 예제 섹션을 참조하세요.

이 파라미터는 Docker 컨테이너 생성 명령의 `LogConfig` 및 docker run에 대한 `--log-driver` 옵션에 매핑됩니다. 기본적으로 컨테이너는 Docker 대몬(daemon)이 사용하는 것과 동일한 로깅 드라이버를 사용합니다. 하지만 컨테이너는 이 파라미터를 사용하여 컨테이너 정의에 로그 드라이버를 지정함으로써 Docker 대몬(daemon)과 다른 로깅 드라이버를 사용할 수 있습니다. 컨테이너에 다른 로깅 드라이버를 사용하려면 컨테이너 인스턴스에서(또는 원격 로깅 옵션의 다른 로그 서버에서) 로그 시스템이 올바르게 구성되어야 합니다.

컨테이너에 대한 로그 구성을 지정할 때 다음 사항에 유의해야 합니다.

- Amazon ECS는 Docker 대몬(daemon)에 사용 가능한 로깅 드라이버의 하위 집합을 지원합니다. 향후의 Amazon ECS 컨테이너 에이전트 릴리스에서 로그 드라이버가 추가될 예정입니다.
- 이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker Remote API 버전 1.18 이상을 사용해야 합니다.
- 컨테이너 인스턴스에서 실행되는 Amazon ECS 컨테이너 에이전트는 `ECS_AVAILABLE_LOGGING_DRIVERS` 환경 변수를 사용하여 해당 인스턴스에서 사용 가능한 로깅 드라이버를 등록해야 해당 인스턴스에 배치된 컨테이너가 이들 로그 구성 옵션을 사용할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

```
"logConfiguration": {
      "logDriver": "awslogs","fluentd","gelf","json-
file","journald","splunk","syslog","awsfirelens",
      "options": {"string": "string"
        ...},
 "secretOptions": [{
```

```
  "name": "string",
  "valueFrom": "string"
 }]
}
```

`logDriver`

타입: 문자열

유효값: `"awslogs","fluentd","gelf","json-file","journald","splunk","syslog","awsfirelens"`

필수 항목 여부: `logConfiguration` 사용 시, 예

컨테이너에 사용할 로드 드라이버입니다. 앞에 나열된 유효한 값은 Amazon ECS 컨테이너 에이전트가 기본적으로 통신할 수 있는 로그 드라이버입니다.

지원되는 로그 드라이버는 `awslogs`, `fluentd`, `gelf`, `json-file`, `journald`, `syslog`, `splunk`, `awsfirelens`입니다.

태스크 정의에서 `awslogs` 로그 드라이버를 사용하여 컨테이너 로그를 CloudWatch Logs로 보내는 방법에 대한 자세한 정보는 Amazon ECS 로그를 CloudWatch로 전송 섹션을 참조하세요.

`awsfirelens` 로그 드라이버 사용에 대한 자세한 정보는 사용자 지정 로그 라우팅을 참조하세요.

> **Note**
>
> 목록에 포함되지 않은 사용자 지정 드라이버가 있는 경우, GitHub에서 사용 가능한 Amazon ECS 컨테이너 에이전트 프로젝트를 포킹하여 해당 드라이버와 함께 작동하도록 사용자 지정할 수 있습니다. 포함하고 싶은 변경에 대해서는 풀 요청을 제출할 것을 권장합니다. 단, 현재 이 소프트웨어의 수정된 사본 실행은 지원되지 않습니다.

이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker 원격 API 버전 1.18 이상을 사용해야 합니다.

`options`

유형: 문자열 대 문자열 맵

필수 여부: 아니요

로그 드라이버로 보낼 구성 옵션의 키/값 맵입니다.

지정할 수 있는 옵션은 로그 드라이버에 따라 달라집니다. 다음은 awslogs 라우터를 사용하여 로그를 Amazon CloudWatch로 라우팅할 때 지정할 수 있는 몇 가지 옵션입니다.

awslogs-create-group

필수 여부: 아니요

로그 그룹을 자동으로 생성할지를 지정합니다. 이 옵션이 지정되지 않은 경우 기본적으로 false로 설정됩니다.

> **Note**
>
> awslogs-create-group을 사용하기 전에 IAM 정책이 logs:CreateLogGroup 권한을 포함해야 합니다.

awslogs-region

필수 여부: 예

awslogs 로그 드라이버가 Docker 로그를 보낼 AWS 리전을 지정합니다. 여러 리전의 클러스터로부터 모든 로그를 CloudWatch Logs 내 단일 리전으로 전송하도록 선택할 수 있습니다. 이는 로그를 모두 한 곳에서 볼 수 있도록 하기 위함입니다. 또는, 리전별로 구분하여 세분화할 수 있습니다. 지정한 로그 그룹이 이 옵션으로 지정한 리전에 위치하는지 확인하세요.

awslogs-group

필수 여부: 예

awslogs 로그 드라이버가 로그 스트림을 전송할 로그 그룹을 지정해야 합니다.

awslogs-stream-prefix

필수 항목 여부: 선택 사항

awslogs-stream-prefix 옵션을 사용하여 로그 스트림을 지정한 접두사, 컨테이너 이름 및 컨테이너가 속하는 Amazon ECS 태스크의 ID와 연결할 수 있습니다. 이 옵션을 사용하여 접두사를 지정하는 경우 로그 스트림은 다음 형식을 취합니다.

```
prefix-name/container-name/ecs-task-id
```

이 옵션을 사용하여 접두사를 지정하지 않는 경우 로그 스트림 이름이 컨테이너 인스턴스의 Docker 대몬에 의해 할당된 컨테이너 ID를 따서 지정됩니다. Docker 컨테이너 ID(컨테이너 인스턴스에서만 사용 가능)만으로는 로그를 전송한 컨테이너로 역추적하기 어려우므로 이 옵션에서 접두사를 지정하는 것이 좋습니다.

Amazon ECS 서비스의 경우 서비스 이름을 접두사로 사용할 수 있습니다. 이렇게 하면 로그 스트림을 컨테이너가 속하는 서비스, 로그를 전송한 컨테이너의 이름, 컨테이너가 속하는 작업의 ID로 추적할 수 있습니다.

Amazon ECS 콘솔을 사용할 때 로그 창에 로그가 표시되도록 하기 위해서는 해당 로그에 스트림 접두사를 지정해야 합니다.

`awslogs-datetime-format`

필수 여부: 아니요

이 옵션은 Python `strftime` 형식의 여러 줄 시작 패턴을 정의합니다. 로그 메시지는 패턴과 일치하는 하나의 줄과 패턴과 일치하지 않는 나머지 줄들로 이루어져 있습니다. 일치하는 줄은 로그 메시지 간의 구분 기호입니다.

이 형식을 사용하는 사용 사례의 한 예는 스택 덤프와 같은 출력을 구문 분석하는 것이며, 그렇지 않으면 여러 항목에 기록될 수 있습니다. 올바른 패턴을 통해 단일 항목으로 캡처할 수 있습니다.

자세한 내용은 awslogs-datetime-format을 참조하세요.

`awslogs-datetime-format` 및 `awslogs-multiline-pattern` 옵션을 모두 구성할 수는 없습니다.

> **Note**
>
> 여러 줄 로깅은 모든 로그 메시지의 정규식 구문 분석 및 일치 태스크를 수행합니다. 이는 로깅 성능에 부정적인 영향을 줄 수 있습니다.

`awslogs-multiline-pattern`

필수 여부: 아니요

이 옵션은 정규식을 사용하여 여러 줄 시작 패턴을 정의합니다. 로그 메시지는 패턴과 일치하는 하나의 줄과 패턴과 일치하지 않는 나머지 줄들로 이루어져 있습니다. 일치하는 줄은 로그 메시지 간의 구분 기호입니다.

자세한 정보는 awslogs-multiline-pattern을 참조하세요.

이 옵션은 awslogs-datetime-format을 구성하는 경우에 무시됩니다.

awslogs-datetime-format 및 awslogs-multiline-pattern 옵션을 모두 구성할 수는 없습니다.

> **Note**
>
> 여러 줄 로깅은 모든 로그 메시지의 정규식 구문 분석 및 일치 태스크를 수행합니다. 이는 로깅 성능에 부정적인 영향을 줄 수 있습니다.

다음 옵션은 지원되는 모든 로그 드라이버에 적용됩니다.

mode

필수 여부: 아니요

유효한 값: non-blocking | blocking

이 옵션은 컨테이너에서 logDriver를 사용하여 지정된 로그 드라이버로 로그 메시지를 전송하는 모드를 정의합니다. 선택한 전송 모드는 컨테이너에서 나오는 로그 흐름이 중단될 때 애플리케이션 가용성에 영향을 줍니다.

blocking 모드를 사용하는 경우 로그 흐름이 중단되면 stdout 및 stderr 스트림에 쓰기 위한 컨테이너 코드의 호출이 차단됩니다. 결과적으로 애플리케이션의 로깅 스레드가 차단됩니다. 이로 인해 애플리케이션이 응답하지 않고 컨테이너 상태 확인에 실패할 수 있습니다.

non-blocking 모드를 사용하는 경우 컨테이너의 로그는 max-buffer-size 옵션으로 구성된 인 메모리 중간 버퍼에 대신 저장됩니다. 이렇게 하면 로그를 전송할 수 없을 때 애플리케이션이 응답하지 않게 되는 것을 방지할 수 있습니다. 서비스 가용성을 보장하고 일부 로그 손실이 괜찮다면 이 모드를 사용하는 것이 좋습니다. 자세한 내용은 Preventing log loss with non-blocking mode in the awslogs container log driver를 참조하세요.

defaultLogDriverMode 계정 설정을 사용하여 특정 AWS 리전의 모든 컨테이너에 대해 기본 mode를 설정할 수 있습니다. logConfiguration에서 mode 옵션을 지정하지 않거나 계정 설정을 구성하지 않으면 Amazon ECS는 기본적으로 non-blocking 모드로 설정됩니다. 계정 설정에 대한 자세한 내용은 기본 로그 드라이버 모드 섹션을 참조하세요.

non-blocking 모드를 사용하면 max-buffer-size 로그 옵션은 중간 메시지 스토리지에 사용되는 버퍼의 크기를 제어합니다. 애플리케이션에 따라 적절한 버퍼 크기를 지정해야 합니다. 작업 수준에서 할당된 총 메모리 양은 로그 드라이버 메모리 버퍼 외에도 모든 컨테이너에 할당된 메모리 양보다 커야 합니다.

> **Note**
>
> 2025년 6월 25일 Amazon ECS는 태스크 가용성을 로깅보다 우선시하기 위해 기본 로그 드라이버 모드를 blocking에서 non-blocking으로 변경했습니다. 이 변경 후에도 blocking 모드를 계속 사용하려면 다음 중 하나를 수행합니다.
>
> - 컨테이너 정의의 logConfiguration에서 mode 옵션을 blocking로 설정합니다.
> - defaultLogDriverMode 계정 설정을 blocking로 설정합니다.

max-buffer-size

필수 여부: 아니요

기본 값: 10 m

non-blocking 모드를 사용하면 max-buffer-size 로그 옵션은 중간 메시지 스토리지에 사용되는 버퍼의 크기를 제어합니다. 애플리케이션에 따라 적절한 버퍼 크기를 지정해야 합니다. 버퍼가 가득 차면 로그를 더 이상 저장할 수 없습니다. 저장할 수 없는 로그는 손실됩니다.

splunk 로그 라우터를 사용하여 로그를 라우팅하려면 splunk-token과 splunk-url을 지정해야 합니다.

awsfirelens 로그 라우터를 사용하여 로그 스토리지 및 분석을 위해 AWS Partner Network 또는 AWS 서비스 대상으로 로그를 라우팅하는 경우 log-driver-buffer-limit 옵션을 설정하여 메모리에 버퍼링되는 이벤트 수를 제한한 후 로그 라우터 컨테이너로 전송할 수 있습니다. 처리량이 많으면 Docker 내부 버퍼의 메모리가 부족해질 수 있으므로 잠재적인 로그 손실 문제를 해결하는 데 도움이 될 수 있습니다. 자세한 내용은 the section called “높은 처리량을 위한 로그 구성” 섹션을 참조하세요.

awsfirelens를 사용하여 로그를 라우팅할 때 지정할 수 있는 다른 옵션은 대상에 따라 다릅니다. 로그를 Amazon Data Firehose로 내보낼 때 AWS 리전으로 region을 지정하고 delivery_stream으로 로그 스트림의 이름을 지정할 수 있습니다.

로그를 Amazon Kinesis Data Streams로 내보낼 때 `region`으로 AWS 리전을 지정하고 `stream`으로 데이터 스트림 이름을 지정할 수 있습니다.

로그를 Amazon OpenSearch Service로 내보낼 때 `Name`, `Host`(프로토콜이 없는 OpenSearch Service 엔드포인트), `Port`, `Index`, `Type`, `Aws_auth`, `Aws_region`, `Suppress_Type_Name`, `tls` 등의 옵션을 지정할 수 있습니다.

로그를 Amazon S3로 내보내는 경우 `bucket` 옵션을 사용하여 버킷을 지정할 수 있습니다. `region`, `total_file_size`, `upload_timeout` 및 `use_put_object`도 옵션으로 지정할 수 있습니다.

이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker 원격 API 버전 1.19 이상을 사용해야 합니다.

`secretOptions`

유형: 객체 배열

필수 여부: 아니요

로그 구성에 전달할 암호를 나타내는 객체입니다. 로그 구성에 사용되는 보안 암호에는 인증 토큰, 인증서 또는 암호화 키가 포함될 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.

`name`

유형: 문자열

필수 항목 여부: 예

컨테이너에서 환경 변수로 설정할 값입니다.

`valueFrom`

유형: 문자열

필수 항목 여부: 예

컨테이너의 로그 구성에 노출할 암호.

```
"logConfiguration": {
 "logDriver": "splunk",
```

```
 "options": {
  "splunk-url": "https://cloud.splunk.com:8080",
  "splunk-token": "...",
  "tag": "...",
  ...
 },
 "secretOptions": [{
  "name": "splunk-token",
  "valueFrom": "/ecs/logconfig/splunkcred"
 }]
}
```

`firelensConfiguration`

유형: FirelensConfiguration 객체

필수 여부: 아니요

컨테이너의 FireLens 구성입니다. 컨테이너 로그에 대한 로그 라우터를 지정하고 구성하는 데 사용됩니다. 자세한 정보는 Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송 섹션을 참조하세요.

```
{
    "firelensConfiguration": {
        "type": "fluentd",
        "options": {
            "KeyName": ""
        }
    }
}
```

`options`

유형: 문자열 대 문자열 맵

필수 여부: 아니요

로그 라우터를 구성할 때 사용할 옵션의 키/값 맵입니다. 이 필드는 선택 사항이며, 사용자 지정 구성 파일을 지정하거나 태스크, 태스크 정의, 클러스터 및 컨테이너 인스턴스 세부 정보와 같은 추가 메타데이터를 로그 이벤트에 추가하는 데 사용할 수 있습니다. 이 필드를 지정할 경우 사용할 구문은 `"options":{"enable-ecs-log-metadata":"true|false","config-file-type:"s3|file","config-file-value":"arn:aws:s3:::amzn-s3-demo-`

bucket/fluent.conf|filepath"}입니다. 자세한 정보는 Amazon ECS 태스크 정의 예제: 로그를 FireLens로 라우팅 섹션을 참조하세요.

type

유형: 문자열

필수 항목 여부: 예

사용할 로그 라우터입니다. 유효한 값은 fluentd 또는 fluentbit입니다.

보안

컨테이너 보안에 대한 자세한 내용은 Amazon ECS 태스크 및 컨테이너 보안 모범 사례를 참조하세요.

credentialSpecs

유형: 문자열 배열

필수 여부: 아니요

Active Directory 인증을 위한 컨테이너를 구성하는 보안 인증 사양(CredSpec) 파일에 대한 SSM 또는 Amazon S3의 ARN 목록입니다. 이 파라미터를 dockerSecurityOptions 대신 사용하는 것이 좋습니다. 최대 ARN 수는 1개입니다.

각 ARN의 형식은 두 가지입니다.

credentialspecdomainless:MyARN

Secrets Manager에서 보안 암호에 대한 추가 섹션에 credentialspecdomainless:MyARN을 사용하여 CredSpec를 제공합니다. 보안 암호에 도메인에 대한 로그인 보안 인증을 제공합니다.

컨테이너 인스턴스에서 실행되는 각 작업을 서로 다른 도메인에 조인할 수 있습니다.

컨테이너 인스턴스를 도메인에 조인하지 않고도 이 형식을 사용할 수 있습니다.

credentialspec:MyARN

credentialspec:MyARN을 사용하여 단일 도메인에 대한 CredSpec를 제공합니다.

이 작업 정의를 사용하는 작업을 시작하려면 먼저 컨테이너 인스턴스를 도메인에 조인해야 합니다.

두 형식 모두에서 MyARN을 SSM 또는 Amazon S3의 ARN으로 바꿉니다.

credspec에서는 사용자 이름, 암호 및 연결할 도메인을 포함하는 보안 암호에 대한 Secrets Manager의 ARN을 제공해야 합니다. 보안을 강화하기 위해 도메인 없는 인증의 경우 인스턴스가 도메인에 조인되지 않습니다. 인스턴스의 다른 애플리케이션은 도메인 없는 보안 인증을 사용할 수 없습니다. 작업을 다른 도메인에 조인해야 하는 경우에도 이 파라미터를 사용하여 동일한 인스턴스에서 작업을 실행할 수 있습니다. 자세한 내용은 Using gMSAs for Windows Containers 및 Using gMSAs for Linux Containers를 참조하세요.

`privileged`

유형: 부울

필수 여부: 아니요

이 파라미터가 true일 경우 컨테이너에는 호스트 컨테이너 인스턴스에 대한 승격된 권한을 부여받습니다(root 사용자와 비슷함). privileged로 컨테이너를 실행하는 것이 좋습니다. 대부분의 경우, privileged를 사용하는 대신 특정 파라미터를 사용하여 정확히 필요한 권한을 지정할 수 있습니다.

이 파라미터는 Docker 컨테이너 생성 명령의 Privileged 및 docker run에 대한 --privileged 옵션에 매핑됩니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너 또는 Fargate 시작 유형을 사용하는 태스크에 대해서는 지원되지 않습니다.

기본값은 false입니다.

```
"privileged": true|false
```

`user`

유형: 문자열

필수 여부: 아니요

컨테이너 내부에서 사용할 사용자입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 User 및 docker run에 대한 --user 옵션에 매핑됩니다.

> **Important**
>
> host 네트워크 모드를 사용하는 작업을 실행할 때는 루트 사용자(UID 0)를 사용하여 컨테이너를 실행해서는 안 됩니다. 보안상 항상 루트가 아닌 사용자를 사용합니다.

다음 형식을 사용하여 user를 지정할 수 있습니다. UID 또는 GID를 지정하는 경우 이를 양의 정수로 지정해야 합니다.

- `user`
- `user:group`
- `uid`
- `uid:gid`
- `user:gid`
- `uid:group`

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

```
"user": "string"
```

`dockerSecurityOptions`

유형: 문자열 배열

유효한 값: "no-new-privileges" | "apparmor:PROFILE" | "label:*value*" | "credentialspec:*CredentialSpecFilePath*"

필수 여부: 아니요

여러 보안 시스템에 대한 사용자 지정 구성을 제공하는 문자열 목록입니다.

Linux 태스크의 경우 이 파라미터를 사용하여 SELinux 및 AppArmor 다단계 보안 시스템의 사용자 지정 레이블을 참조할 수 있습니다.

이 파라미터를 사용하여 Active Directory 인증을 위한 컨테이너를 구성하는 보안 인증 사양 파일을 참조할 수 있습니다. 자세한 내용은 Amazon ECS의 EC2 Windows 컨테이너에 대해 gMSA를 사용

하는 방법 알아보기 및 Amazon ECS에서 EC2 Linux 컨테이너에 대해 gMSA 사용(을)를 참조하세요.

이 파라미터는 Docker 컨테이너 생성 명령의 `SecurityOpt` 및 docker run에 대한 `--security-opt` 옵션에 매핑됩니다.

```
"dockerSecurityOptions": ["string", ...]
```

> **Note**
>
> 컨테이너 인스턴스에서 실행되는 Amazon ECS 컨테이너 에이전트는 `ECS_SELINUX_CAPABLE=true` 또는 `ECS_APPARMOR_CAPABLE=true` 환경 변수를 사용하여 해당 인스턴스에서 사용 가능한 로깅 드라이버를 등록해야 해당 인스턴스에 배치된 컨테이너가 이들 보안 옵션을 사용할 수 있습니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성을 참조하세요.

리소스 제한

`ulimits`

유형: 객체 배열

필수 여부: 아니요

컨테이너에 정의할 `ulimit` 값 목록. 이 값은 운영 체제에 설정된 기본 리소스 할당량 설정을 다시 정의합니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `Ulimits` 및 docker run에 대한 `--ulimit` 옵션에 매핑됩니다.

이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker 원격 API 버전 1.18 이상을 사용해야 합니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

```
"ulimits": [
      {
```

```
      "name":
"core"|"cpu"|"data"|"fsize"|"locks"|"memlock"|"msgqueue"|"nice"|"nofile"|"nproc"|"rss"|"rtp
      "softLimit": integer,
      "hardLimit": integer
    }
    ...
  ]
```

name

타입: 문자열

유효값: "core" | "cpu" | "data" | "fsize" | "locks" | "memlock" | "msgqueue" | "nice" | "nofile" | "nproc" | "rss" | "rtprio" | "rttime" | "sigpending" | "stack"

필수 항목 여부: 예(ulimits 사용 시)

ulimit의 type입니다.

hardLimit

유형: 정수

필수 항목 여부: 예(ulimits 사용 시)

ulimit 유형의 하드 제한입니다. 값은 ulimit의 type에 따라 바이트, 초 또는 개수로 지정할 수 있습니다.

softLimit

유형: 정수

필수 항목 여부: 예(ulimits 사용 시)

ulimit 유형의 소프트 제한입니다. 값은 ulimit의 type에 따라 바이트, 초 또는 개수로 지정할 수 있습니다.

Docker 레이블

dockerLabels

유형: 문자열 대 문자열 맵

필수 여부: 아니요

컨테이너에 추가할 레이블의 키/값 맵입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 `Labels` 및 docker run에 대한 `--label` 옵션에 매핑됩니다.

이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker 원격 API 버전 1.18 이상을 사용해야 합니다.

```
"dockerLabels": {"string": "string"
      ...}
```

기타 컨테이너 정의 파라미터

다음 컨테이너 정의 파라미터는 JSON을 통해 구성(Configure via JSON) 옵션을 사용하여 Amazon ECS 콘솔에서 태스크 정의를 등록할 때 사용할 수 있습니다. 자세한 정보는 콘솔을 사용하여 Amazon ECS 작업 정의 생성을 참조하세요.

주제

- Linux 파라미터
- 컨테이너 종속성
- 컨테이너 제한 시간
- System Controls
- 대화형
- 의사 터미널

Linux 파라미터

`linuxParameters`

유형: LinuxParameters 객체

필수 여부: 아니요

컨테이너에 적용되는 Linux 관련 옵션(예: KernelCapabilities)

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서는 지원되지 않습니다.

```
"linuxParameters": {
      "capabilities": {
        "add": ["string", ...],
        "drop": ["string", ...]
        }
      }
```

`capabilities`

유형: KernelCapabilities 객체

필수 여부: 아니요

컨테이너에 대해 Docker에서 제공하는 기본 구성에 추가하거나 제거할 Linux 기능입니다. 이러한 Linux 기능에 대한 자세한 정보는 capabilities(7) Linux 매뉴얼 페이지를 참조하세요.

`add`

유형: 문자열 배열

유효값: `"ALL" | "AUDIT_CONTROL" | "AUDIT_READ" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER" | "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" | "LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" | "NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW" | "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" | "SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT" | "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" | "SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"`

필수 여부: 아니요

컨테이너에 대해 Docker에서 제공하는 기본 구성에 추가할 Linux 기능입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 CapAdd 및 docker run에 대한 `--cap-add` 옵션에 매핑됩니다.

`add`

유형: 문자열 배열

유효값: `"SYS_PTRACE"`

필수 여부: 아니요

컨테이너에 대해 Docker에서 제공하는 기본 구성에 추가할 Linux 기능입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 CapAdd 및 docker run에 대한 --cap-add 옵션에 매핑됩니다.

drop

유형: 문자열 배열

유효값: "ALL" | "AUDIT_CONTROL" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER" | "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" | "LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" | "NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW" | "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" | "SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT" | "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" | "SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"

필수 여부: 아니요

컨테이너에 대해 Docker에서 제공하는 기본 구성에서 제거할 Linux 기능입니다. 이 파라미터는 Docker 컨테이너 생성 명령의 CapDrop 및 docker run에 대한 --cap-drop 옵션에 매핑됩니다.

devices

컨테이너에 노출될 모든 호스트 디바이스. 이 파라미터는 Docker 컨테이너 생성 명령의 Devices 및 docker run에 대한 --device 옵션에 매핑됩니다.

유형: Device 객체 배열

필수 여부: 아니요

hostPath

호스트 컨테이너 인스턴스의 디바이스 경로.

유형: 문자열

필수 항목 여부: 예

containerPath

호스트 디바이스를 노출할 컨테이너 내 경로.

유형: 문자열

필수 여부: 아니요

permissions

디바이스를 위해 컨테이너에 제공할 명시적 권한. 컨테이너는 기본적으로 디바이스에서 read, write 및 mknod에 대한 권한이 있습니다.

유형: 문자열 배열

유효한 값: read | write | mknod

initProcessEnabled

신호를 전달하고 프로세스의 결과를 받아들이는 컨테이너 내에서 init 프로세스를 실행합니다. 이 파라미터는 docker run에 대한 --init 옵션에 매핑됩니다.

이 파라미터를 사용하려면 컨테이너 인스턴스에서 Docker 원격 API 버전 1.25 이상을 사용해야 합니다.

maxSwap

컨테이너가 사용할 수 있는 총 스왑 메모리 양(MiB) 이 파라미터는 docker run에 대한 --memory-swap 옵션으로 변환되며 컨테이너 메모리의 합계에 maxSwap 값을 더한 값이 됩니다.

0의 maxSwap 값이 지정되면 컨테이너는 스왑을 사용하지 않습니다. 허용되는 값은 0 또는 양수입니다. maxSwap 파라미터를 생략하면 컨테이너는 실행 중인 컨테이너 인스턴스에 대한 스왑 구성을 사용합니다. swappiness 매개 변수를 사용하려면 maxSwap 값을 설정해야 합니다.

sharedMemorySize

/dev/shm 볼륨의 크기 값(MiB)입니다. 이 파라미터는 docker run에 대한 --shm-size 옵션에 매핑됩니다.

유형: 정수

swappiness

이 파라미터를 통해 컨테이너의 메모리 스왑 동작을 조정할 수 있습니다. swappiness 값이 0이면 필요하지 않은 경우 스와핑이 발생하지 않도록 합니다. swappiness 값이 100이면 페이

지가 자주 스와핑되도록 합니다. 허용되는 값은 0과 100 사이의 숫자입니다. 값을 지정하지 않으면 기본값 60이 사용됩니다. 또한, maxSwap 값이 지정되지 않은 경우 이 파라미터는 무시됩니다. 이 파라미터는 docker run에 대한 --memory-swappiness 옵션에 매핑됩니다.

> **Note**
>
> Amazon Linux 2023에서 작업을 사용하는 경우에는 swappiness 파라미터가 지원되지 않습니다.

`tmpfs`

컨테이너 경로, 마운트 옵션 및 tmpfs 마운트의 최대 크기(MiB)입니다. 이 파라미터는 docker run에 대한 --tmpfs 옵션에 매핑됩니다.

유형: Tmpfs 객체 배열

필수 여부: 아니요

`containerPath`

tmpfs 볼륨을 마운트할 절대 파일 경로입니다.

유형: 문자열

필수 항목 여부: 예

`mountOptions`

tmpfs 볼륨 마운트 옵션의 목록입니다.

유형: 문자열 배열

필수 여부: 아니요

유효 값: `"defaults" | "ro" | "rw" | "suid" | "nosuid" | "dev" | "nodev" | "exec" | "noexec" | "sync" | "async" | "dirsync" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" | "private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" | "relatime" | "norelatime" | "strictatime" | "nostrictatime" | "mode" | "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"`

`size`

tmpfs 볼륨의 최대 크기(MiB) 입니다.

유형: 정수

필수 여부: 예

컨테이너 종속성

`dependsOn`

유형: ContainerDependency 객체의 배열

필수 여부: 아니요

컨테이너 시작 및 종료에 대해 정의된 종속성입니다. 컨테이너는 여러 종속성을 포함할 수 있습니다. 컨테이너가 시작될 때 종속성이 정의되면 컨테이너를 종료할 때 종속성이 취소됩니다. 문제 해결 예는 컨테이너 종속성 섹션을 참조하세요.

> **Note**
> 컨테이너가 종속성 제약을 충족하지 못하거나 제약을 충족하기 전에 시간이 초과되면 Amazon ECS는 종속된 컨테이너를 다음 상태로 진행시키지 못합니다.

컨테이너 종속성을 사용하려면 컨테이너 인스턴스에 버전 `1.26.0` 이상의 컨테이너 에이전트가 필요합니다. 그러나 최신 버전의 컨테이너 에이전트를 사용하는 것이 좋습니다. 에이전트 버전을 확인하고 최신 버전으로 업데이트하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요. Amazon ECS 최적화 Amazon Linux AMI를 사용하는 경우 해당 인스턴스에는 `1.26.0-1` 이상 버전의 `ecs-init` 패키지가 필요합니다. 컨테이너 인스턴스가 버전 `20190301` 이상에서 시작된 경우 필요한 버전의 컨테이너 에이전트 및 `ecs-init`이 포함되어 있습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.

```
"dependsOn": [
    {
        "containerName": "string",
        "condition": "string"
    }
]
```

containerName

유형: 문자열

필수 항목 여부: 예

지정된 조건을 충족해야 하는 컨테이너 이름입니다.

condition

유형: 문자열

필수 항목 여부: 예

컨테이너의 종속성 조건입니다. 다음은 사용 가능한 조건과 그 동작입니다.

- START - 이 조건은 오늘의 링크와 볼륨의 동작을 에뮬레이션합니다. 이 조건은 종속 컨테이너가 시작되었는지 확인한 다음에 다른 컨테이너 시작을 허용합니다.
- COMPLETE - 이 조건은 다른 컨테이너를 시작하기 전에 종속 컨테이너 실행이 완료(종료)되었는지 확인합니다. 이는 스크립트를 실행한 후 종료하는 필수적이지 않은 컨테이너에 유용할 수 있습니다. 필수 컨테이너에는 이 조건을 설정할 수 없습니다.
- SUCCESS - 이 조건은 COMPLETE와 동일하지만 컨테이너가 zero 상태로 종료되어야 합니다. 필수 컨테이너에는 이 조건을 설정할 수 없습니다.
- HEALTHY - 이 조건은 다른 컨테이너를 시작하기 전에 종속 컨테이너가 컨테이너 상태 확인을 통과하는지 확인합니다. 이렇게 하려면 태스크 정의에서 종속 컨테이너에 상태 확인이 구성되어 있어야 합니다. 이 조건은 태스크 시작 시에만 확인됩니다.

컨테이너 제한 시간

startTimeout

유형: 정수

필수 여부: 아니요

예제 값: 120

컨테이너에 대한 종속성 해결을 포기하기 전에 대기할 시간(초)입니다.

예를 들어 containerA에 대한 태스크 정의에서 containerB에 대한 종속성이 COMPLETE, SUCCESS 또는 HEALTHY 상태인 두 개의 컨테이너를 지정합니다. containerB에 대해

`startTimeout` 값이 지정되고 그 시간 내에 원하는 상태에 도달하지 않으면 `containerA`는 시작하지 않습니다.

> **Note**
>
> 컨테이너가 종속성 제약을 충족하지 못하거나 제약을 충족하기 전에 시간이 초과되면 Amazon ECS는 종속된 컨테이너를 다음 상태로 진행시키지 못합니다.

최대값은 120초입니다.

`stopTimeout`

유형: 정수

필수 여부: 아니요

예제 값: `120`

자체적으로 정상적으로 종료하지 않을 경우 컨테이너를 강제 종료하기까지 대기할 시간(초)입니다.

`stopTimeout` 파라미터를 지정하지 않으면 Amazon ECS 컨테이너 에이전트 구성 변수 `ECS_CONTAINER_STOP_TIMEOUT`에 대해 설정된 값이 사용됩니다. `stopTimeout` 파라미터 또는 `ECS_CONTAINER_STOP_TIMEOUT` 에이전트 구성 변수를 모두 설정하지 않은 경우 Linux 컨테이너의 경우 30초(기본값)가 사용되고 Windows 컨테이너의 경우 30초(기본값)가 사용됩니다. 컨테이너 중지 제한 시간 값을 활성화하려면 컨테이너 인스턴스에 버전 1.26.0 이상의 컨테이너 에이전트가 필요합니다. 그러나 최신 버전의 컨테이너 에이전트를 사용하는 것이 좋습니다. 에이전트 버전을 확인하고 최신 버전으로 업데이트하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요. Amazon ECS 최적화 Amazon Linux AMI를 사용하는 경우 해당 인스턴스에는 1.26.0-1 이상 버전의 `ecs-init` 패키지가 필요합니다. 컨테이너 인스턴스가 버전 `20190301` 이상에서 시작된 경우 필요한 버전의 컨테이너 에이전트 및 `ecs-init`이 포함되어 있습니다. 자세한 정보는 Amazon ECS 최적화 Linux AMI을 참조하세요.

System Controls

`systemControls`

유형: SystemControl 객체

필수 여부: 아니요

컨테이너에서 설정할 네임스페이스 커널 파라미터의 목록입니다. 이 파라미터는 Docker 컨테이너 생성 명령에 있는 Sysctls 및 docker run에 대한 --sysctl 옵션에 매핑됩니다. 예를 들어 net.ipv4.tcp_keepalive_time 설정을 구성하여 연결을 더 오래 유지할 수 있습니다.

awsvpc 또는 host 네트워크 모드도 사용하는 단일 작업에서 여러 컨테이너에 네트워크 관련 systemControls 파라미터를 지정하지 않는 것이 좋습니다. 이렇게 하면 다음과 같은 단점이 있습니다.

- awsvpc 네트워크 모드를 사용하는 작업의 경우 컨테이너에 systemControls을 설정할 경우 작업의 모든 컨테이너에 적용됩니다. 단일 작업의 여러 컨테이너에 서로 다른 systemControls를 설정할 경우 마지막으로 시작되는 컨테이너에 따라 적용될 systemControls가 결정됩니다.
- host 네트워크 모드를 사용하는 태스크의 경우, 네트워크 네임스페이스 systemControls는 지원되지 않습니다.

태스크 내 컨테이너에 사용할 IPC 리소스 네임스페이스를 설정하는 경우, 시스템 제어에 다음 사항이 적용됩니다. 자세한 정보는 IPC 모드을 참조하세요.

- host IPC 모드를 사용하는 태스크의 경우, IPC 네임스페이스 systemControls는 지원되지 않습니다.
- task IPC 모드를 사용하는 작업의 경우, IPC 네임스페이스 systemControls 값은 작업 내 모든 컨테이너에 적용됩니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

```
"systemControls": [
    {
         "namespace":"string",
         "value":"string"
    }
]
```

namespace

유형: 문자열

필수 여부: 아니요

value를 설정할 네임스페이스 커널 파라미터입니다.

유효한 IPC 네임스페이스 값:"kernel.msgmax" | "kernel.msgmnb" | "kernel.msgmni" | "kernel.sem" | "kernel.shmall" | "kernel.shmmax" | "kernel.shmmni" | "kernel.shm_rmid_forced" 및 "fs.mqueue.*"로 시작하는 Sysctls

유효한 네트워크 네임스페이스 값: "net.*"로 시작하는 Sysctls

value

유형: 문자열

필수 여부: 아니요

namespace에 지정된 네임스페이스 커널 파라미터의 값입니다.

대화형

interactive

유형: 부울

필수 여부: 아니요

이 파라미터가 true일 경우 이것을 사용하여 stdin 또는 tty를 할당해야 하는 컨테이너화된 애플리케이션을 배포할 수 있습니다. 이 파라미터는 Docker 컨테이너 생성 명령의 OpenStdin 및 docker run에 대한 --interactive 옵션에 매핑됩니다.

기본값은 false입니다.

의사 터미널

pseudoTerminal

유형: 부울

필수 여부: 아니요

이 파라미터가 true일 경우 TTY가 할당됩니다. 이 파라미터는 Docker 컨테이너 생성 명령의 Tty 및 docker run에 대한 --tty 옵션에 매핑됩니다.

기본값은 false입니다.

Elastic Inference 액셀러레이터 이름

태스크 정의에 대한 Elastic Inference 액셀러레이터 리소스 요구 사항.

> **Note**
>
> Amazon Elastic Inference(EI)는 더 이상 고객에게 제공되지 않습니다.

태스크 정의에는 다음 파라미터가 허용됩니다.

`deviceName`

유형: 문자열

필수 항목 여부: 예

Amazon Elastic Inference 액셀러레이터 디바이스 이름입니다. `deviceName`은 컨테이너 정의에서도 참조해야 합니다. `Elastic Inference accelerator`를 참조하세요.

`deviceType`

유형: 문자열

필수 항목 여부: 예

사용할 Elastic Inference 액셀러레이터입니다.

작업 배치 제약

태스크 정의를 등록할 때 Amazon ECS가 태스크를 배치하는 방식을 사용자 지정하는 작업 배치 제약을 제공할 수 있습니다.

제약을 사용하여 가용 영역, 인스턴스 유형 또는 사용자 지정 속성에 따라 작업을 배치할 수 있습니다. 자세한 내용은 Amazon ECS가 작업에 사용하는 컨테이너 인스턴스 정의 섹션을 참조하세요.

컨테이너 정의에서 다음 파라미터가 허용됩니다.

`expression`

유형: 문자열

필수 여부: 아니요

제약에 적용할 클러스터 쿼리 언어 표현식입니다. 자세한 정보는 Amazon ECS 작업에 대한 컨테이너 인스턴스를 정의하도록 표현식 생성 섹션을 참조하세요.

`type`

유형: 문자열

필수 항목 여부: 예

제약의 유형입니다. `memberOf`를 사용하여 선택을 유효한 후보 그룹으로 제한합니다.

프록시 구성

`proxyConfiguration`

유형: ProxyConfiguration 객체

필수 여부: 아니요

App Mesh 프록시에 대한 구성 세부 정보입니다.

EC2 시작 유형을 사용하는 태스크의 경우 프록시 구성을 사용하려면 컨테이너 인스턴스에 컨테이너 에이전트 버전 1.26.0 이상과 `ecs-init` 패키지의 버전 1.26.0-1 이상이 필요합니다. 컨테이너 인스턴스가 Amazon ECS 최적화 AMI 버전 `20190301` 이상에서 시작된 경우 필요한 버전의 컨테이너 에이전트 및 `ecs-init`가 포함되어 있습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

```
"proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "string",
    "properties": [
        {
           "name": "string",
           "value": "string"
        }
    ]
}
```

`type`

유형: 문자열

유효한 값: `APPMESH`

필수 여부: 아니요

프록시 유형입니다. 지원되는 유일한 값은 `APPMESH`입니다.

`containerName`

유형: 문자열

필수 항목 여부: 예

App Mesh 프록시로 사용할 컨테이너의 이름입니다.

`properties`

형식: KeyValuePair 객체 배열

필수 여부: 아니요

키-값 쌍으로 지정된 CNI(Container Network Interface) 플러그인을 제공하는 네트워크 구성 파라미터 세트입니다.

- `IgnoredUID` - (필수) 컨테이너 정의의 `user` 파라미터에 정의된 프록시 컨테이너의 사용자 ID(UID)입니다. 프록시가 트래픽을 무시하는 데 사용됩니다. `IgnoredGID`를 지정한 경우 이 필드를 비워둘 수 있습니다.
- `IgnoredGID` - (필수) 컨테이너 정의의 `user` 파라미터에 정의된 프록시 컨테이너의 그룹 ID(GID)입니다. 프록시가 트래픽을 무시하는 데 사용됩니다. `IgnoredUID`를 지정한 경우 이 필드를 비워둘 수 있습니다.
- `AppPorts` - (필수) 애플리케이션이 사용하는 포트 목록입니다. 이러한 포트에 대한 네트워크 트래픽은 `ProxyIngressPort` 및 `ProxyEgressPort`로 전달됩니다.
- `ProxyIngressPort` - (필수) `AppPorts`로 들어오는 트래픽을 보낼 포트를 지정합니다.
- `ProxyEgressPort` - (필수) `AppPorts`에서 나가는 트래픽을 보낼 포트를 지정합니다.
- `EgressIgnoredPorts` - (필수) 지정된 포트로 가는 아웃바운드 트래픽은 무시되고 `ProxyEgressPort`로 리디렉션되지 않습니다. 값은 빈 목록일 수 있습니다.
- `EgressIgnoredIPs` - (필수) 지정된 IP 주소로 가는 아웃바운드 트래픽은 무시되고 `ProxyEgressPort`로 리디렉션되지 않습니다. 값은 빈 목록일 수 있습니다.

`name`

유형: 문자열

필수 여부: 아니요

키-값 쌍의 이름입니다.

`value`

유형: 문자열

필수 여부: 아니요

키-값 쌍의 값입니다.

볼륨

작업 정의를 등록할 때 선택적으로 볼륨 목록이 컨테이너 인스턴스의 Docker 대몬에 전달되도록 지정할 수 있습니다. 그러면 동일한 컨테이너 인스턴스에 속하는 다른 컨테이너가 액세스하는 데 전달된 볼륨 목록을 사용할 수 있습니다.

사용할 수 있는 데이터 볼륨의 유형은 다음과 같습니다.

- Amazon EBS 볼륨 - 데이터 집약적인 컨테이너화된 워크로드에 대해 비용 효율적이며 내구성이 뛰어난 고성능 블록 스토리지를 제공합니다. 독립 실행형 작업을 실행하거나 서비스를 생성 또는 업데이트할 때 Amazon ECS 작업당 1개의 Amazon EBS 볼륨을 연결할 수 있습니다. Amazon EBS 볼륨은 Linux 태스크에 지원됩니다. 자세한 내용은 Amazon ECS에서 Amazon EBS 볼륨 사용 섹션을 참조하세요.
- Amazon EFS 볼륨 — Amazon ECS 태스크에 사용할 수 있는 간단하고 영구적인 확장형 파일 스토리지를 제공합니다. Amazon EFS를 사용하면 스토리지 용량이 탄력적입니다. 파일을 추가 및 제거하면 스토리지 용량이 자동으로 확장 및 축소됩니다. 애플리케이션에서 스토리지가 필요할 때 필요한 만큼 확보할 수 있습니다. Amazon EFS 볼륨이 지원됩니다. 자세한 내용은 Amazon ECS에서 Amazon EFS 볼륨 사용 섹션을 참조하세요.
- FSx for Windows File Server 볼륨 - 완전 관리형 Microsoft Windows 파일 서버를 제공합니다. 이러한 파일 서버는 Windows 파일 시스템으로 지원됩니다. FSx for Windows File Server와 Amazon ECS를 함께 사용하면 영구적이고 분산, 공유된 고정 파일 스토리지로 Windows 태스크를 프로비저닝할 수 있습니다. 자세한 정보는 Amazon ECS에서 FSx for Windows File Server 볼륨 사용을 참조하세요.

Fargate의 Windows 컨테이너는 이 옵션을 지원하지 않습니다.

- Docker 볼륨 – 호스트 Amazon EC2 인스턴스의 `/var/lib/docker/volumes` 아래에 생성되는 Docker 관리형 볼륨입니다. Docker 볼륨 드라이버(플러그인이라고도 함)는 볼륨을 Amazon EBS와 같은 외부 스토리지 시스템과 통합하는 데 사용됩니다. 내장형 `local` 볼륨 드라이버 또는 타사 볼륨 드라이버를 사용할 수 있습니다. Docker 볼륨은 Amazon EC2 인스턴스에서 작업을 실행하는 경우에만 지원됩니다. Windows 컨테이너는 `local` 드라이버의 사용만 지원합니다. Docker 볼륨을 사용하려면 태스크 정의에서 `dockerVolumeConfiguration`을 지정합니다.
- 바인드 탑재 – 컨테이너에 탑재된 호스트 컴퓨터의 파일 또는 디렉터리입니다. 바인드 탑재 호스트 볼륨이 지원됩니다. 바인드 탑재 호스트 볼륨을 사용하려면 태스크 정의에서 `host`와 `sourcePath`(옵션) 값을 지정합니다.

자세한 내용은 Amazon ECS 작업에 대한 스토리지 옵션 섹션을 참조하세요.

컨테이너 정의에서 다음 파라미터가 허용됩니다.

`name`

유형: 문자열

필수 여부: 아니요

볼륨의 이름입니다. 최대 255자의 문자(대문자 및 소문자), 숫자, 하이(-) 및 밑줄(_)이 허용됩니다. 이 이름은 컨테이너 정의 `mountPoints` 객체의 `sourceVolume` 파라미터에서 참조됩니다.

`host`

필수 여부: 아니요

`host` 파라미터는 바인드 탑재의 수명 주기를 태스크가 아니라 호스트 Amazon EC2 인스턴스와 연결하는 데 사용합니다. `host` 파라미터가 비어 있으면 Docker 대몬이 데이터 볼륨의 호스트 경로를 할당하지만 해당 볼륨과 연결된 컨테이너가 실행을 중지한 후 데이터 유지가 보장되지 않습니다.

Windows 컨테이너는 전체 디렉터리를 동일한 드라이브에 `$env:ProgramData`로 마운트할 수 있습니다.

> **Note**
>
> `sourcePath` 파라미터는 Amazon EC2 인스턴스에 호스팅된 작업을 사용하는 경우에만 지원됩니다.

sourcePath

유형: 문자열

필수 여부: 아니요

host 파라미터가 사용되는 경우 sourcePath를 지정하여 컨테이너에 제시되는 호스트 Amazon EC2 인스턴스 상의 경로를 선언합니다. 이 파라미터가 비어 있으면 Docker 대몬이 사용자 대신 호스트 경로를 할당합니다. host 파라미터에 sourcePath 파일 위치가 들어 있으면, 사용자가 수동으로 삭제하지 않는 한 데이터 볼륨이 호스트 Amazon EC2 인스턴스 상에 지정된 위치를 유지합니다. sourcePath 값이 호스트 Amazon EC2 인스턴스에 없을 경우 Docker 대몬이 해당 경로를 생성합니다. 해당 위치가 있을 경우 소스 경로 폴더의 콘텐츠를 내보냅니다.

configuredAtLaunch

유형: 부울

필수 여부: 아니요

시작 시 볼륨을 구성할 수 있는지 여부를 지정합니다. true로 설정하면 독립 실행형 작업을 실행하거나 서비스를 생성 또는 업데이트할 때 볼륨을 구성할 수 있습니다. true로 설정하면 작업 정의에서 다른 볼륨 구성을 제공할 수 없습니다. 작업에 연결하도록 Amazon EBS 볼륨을 구성하려면 이 파라미터를 true로 설정해야 합니다. configuredAtLaunch를 true로 설정하고 볼륨 구성을 시작 단계로 연기하면 볼륨 유형이나 특정 볼륨 설정으로 제한되지 않는 작업 정의를 생성할 수 있습니다. 그러면 여러 실행 환경에서 작업 정의를 재사용할 수 있습니다. 자세한 내용은 Amazon EBS volumes를 참조하세요.

dockerVolumeConfiguration

유형: DockerVolumeConfiguration 객체

필수 여부: 아니요

이 파라미터는 Docker 볼륨을 사용할 때 지정됩니다. Docker 볼륨은 EC2 인스턴스에서 작업을 실행하는 경우에만 지원됩니다. Windows 컨테이너는 local 드라이버의 사용만 지원합니다. 바인드 탑재를 사용하려면 대신에 host를 지정하세요.

scope

유형: 문자열

유효한 값: task | shared

필수 여부: 아니요

수명 주기를 결정하는 Docker 볼륨의 범위입니다. 범위가 `task`로 지정된 Docker 볼륨은 태스크가 시작될 때 자동으로 프로비저닝되고, 태스크가 중단되면 삭제됩니다. 범위가 `shared`로 지정된 Docker 볼륨은 태스크 중단 후에도 유지됩니다.

`autoprovision`

유형: 부울

기본 값: `false`

필수 여부: 아니요

이 값이 `true`인 경우 도커 볼륨이 아직 없으면 도커 볼륨이 생성됩니다. 이 필드는 `scope`가 `shared`인 경우에만 사용됩니다. `scope`가 `task`인 경우 이 파라미터를 생략해야 합니다.

`driver`

유형: 문자열

필수 여부: 아니요

사용할 Docker 볼륨 드라이버입니다. Docker에서 제공하는 드라이버 이름이 작업 배치에 사용되므로 드라이버 값과 이 이름이 일치해야 합니다. Docker 플러그인 CLI를 사용하여 드라이버를 설치했다면 `docker plugin ls`를 사용하여 컨테이너 인스턴스에서 드라이버 이름을 검색합니다. 다른 방법을 사용하여 드라이버를 설치했다면 Docker 플러그인 검색을 사용하여 드라이버 이름을 검색합니다.

`driverOpts`

유형: 문자열

필수 여부: 아니요

전달할 Docker 드라이버에 특정한 옵션 맵입니다. 이 파라미터는 Docker의 볼륨 생성의 `DriverOpts`에 매핑됩니다.

`labels`

유형: 문자열

필수 여부: 아니요

Docker 볼륨에 추가할 사용자 지정 메타데이터입니다.

`efsVolumeConfiguration`

유형: EFSVolumeConfiguration 객체

필수 여부: 아니요

이 파라미터는 Amazon EFS 볼륨을 사용할 때 지정됩니다.

`fileSystemId`

유형: 문자열

필수 항목 여부: 예

사용할 Amazon EFS 파일 시스템 ID입니다.

`rootDirectory`

유형: 문자열

필수 여부: 아니요

호스트 내의 루트 디렉터리로 탑재할 Amazon EFS 파일 시스템 내 디렉터리입니다. 이 파라미터가 생략되면 Amazon EFS 볼륨의 루트가 사용됩니다. `/`를 지정하면 이 파라미터를 생략하는 것과 동일한 효과가 있습니다.

> **Important**
>
> `authorizationConfig`에 EFS 액세스 포인트가 지정된 경우 루트 디렉터리 파라미터를 생략하거나 `/`로 설정해야 합니다. 그러면 EFS 액세스 포인트에 설정된 경로가 적용됩니다.

`transitEncryption`

유형: 문자열

유효한 값: `ENABLED` | `DISABLED`

필수 여부: 아니요

Amazon ECS 호스트와 Amazon EFS 서버 간에 전송 중인 Amazon EFS 데이터에 대해 암호화를 사용 설정할지 여부를 지정합니다. Amazon EFS IAM 권한 부여를 사용하는 경우 전송

중 데이터 암호화를 사용 설정해야 합니다. 이 파라미터가 누락되면 DISABLED의 기본값이 사용됩니다. 자세한 정보는 Amazon Elastic File System 사용 설명서의 전송 중 데이터 암호화 (Encrypting Data in Transit)를 참조하세요.

`transitEncryptionPort`

유형: 정수

필수 여부: 아니요

Amazon ECS 호스트와 Amazon EFS 서버 간에 암호화된 데이터를 전송할 때 사용할 포트입니다. 전송 중 데이터 암호화 포트를 지정하지 않으면 작업에서는 Amazon EFS 탑재 헬퍼가 사용하는 포트 선택 전략을 사용합니다. 자세한 정보는 Amazon Elastic File System 사용자 설명서의 EFS 탑재 헬퍼를 참조하세요.

`authorizationConfig`

유형: EFSAuthorizationConfig 객체

필수 여부: 아니요

Amazon EFS 파일 시스템에 대한 권한 부여 구성 세부 정보입니다.

`accessPointId`

유형: 문자열

필수 여부: 아니요

사용할 액세스 포인트 ID입니다. 액세스 포인트가 지정된 경우 `efsVolumeConfiguration`에서 루트 디렉터리 값을 생략하거나 `/`로 설정해야 합니다. 그러면 EFS 액세스 포인트에 설정된 경로가 적용됩니다. 액세스 포인트를 사용하는 경우 `EFSVolumeConfiguration`에서 전송 중 데이터 암호화를 활성화해야 합니다. 자세한 정보는 Amazon Elastic File System 사용자 설명서의 Amazon EFS 액세스 포인트 태스크를 참조세요.

`iam`

유형: 문자열

유효한 값: `ENABLED` | `DISABLED`

필수 여부: 아니요

Amazon EFS 파일 시스템을 탑재할 때 작업 정의에 정의된 Amazon ECS 작업 IAM 역할을 사용할지 여부를 지정합니다. 활성화된 경우 `EFSVolumeConfiguration`에서 전송 중 데이터 암호화를 활성화해야 합니다. 이 파라미터가 누락되면 `DISABLED`의 기본값이 사용됩니다. 자세한 정보는 태스크의 IAM 역할을 참조하세요.

`FSxWindowsFileServerVolumeConfiguration`

유형: FSxWindowsFileServerVolumeConfiguration 객체

필수 여부: 예

이 파라미터는 작업을 저장할 Amazon FSx for Windows File Server 파일 시스템을 사용할 때 지정됩니다.

`fileSystemId`

유형: 문자열

필수 항목 여부: 예

사용할 FSx for Windows File Server 파일 시스템 ID입니다.

`rootDirectory`

유형: 문자열

필수 항목 여부: 예

호스트 내의 루트 디렉터리로 탑재할 FSx for Windows File Server 파일 시스템 내 디렉터리입니다.

`authorizationConfig`

`credentialsParameter`

유형: 문자열

필수 항목 여부: 예

권한 부여 자격 증명 옵션입니다.

옵션:

- AWS Secrets Manager 보안 암호의 Amazon 리소스 이름(ARN)입니다.

- AWS Systems Manager 파라미터의 ARN입니다.

`domain`

유형: 문자열

필수 항목 여부: 예

셀프 호스팅된 EC2 Active Directory 또는 AWS Directory Service for Microsoft Active Directory(AWS Managed Microsoft AD) 디렉터리에서 호스팅하는 정규화된 도메인 이름입니다.

Tags

태스크 정의를 등록할 때 태스크 정의에 적용되는 메타데이터 태그를 선택적으로 지정할 수 있습니다. 태그는 태스크 정의를 분류하고 정리하는 데 도움을 줍니다. 각 태그는 키와 값(선택사항)으로 구성됩니다. 두 가지를 모두 정의합니다. 자세한 정보는 Amazon ECS 리소스 태그 지정을 참조하세요.

> **Important**
> 개인 식별 정보나 기타 기밀 정보 또는 민감한 정보를 태그에 추가하지 않습니다. 청구를 비롯한 여러 AWS 서비스에서 태그에 액세스할 수 있습니다. 태그는 개인 데이터나 민감한 데이터에 사용하기 위한 것이 아닙니다.

태그 개체에서 허용되는 파라미터는 다음과 같습니다.

`key`

유형: 문자열

필수 여부: 아니요

하나의 태그를 구성하는 키-값 쌍의 일부분입니다. 키는 더 구체적인 태그 값에 대해 범주와 같은 역할을 하는 일반적인 레이블입니다.

`value`

유형: 문자열

필수 여부: 아니요

하나의 태그를 구성하는 키-값 쌍의 선택적 부분입니다. 하나의 값은 태그 범주(키) 내에서 서술자 역할을 수행합니다.

기타 태스크 정의 파라미터

다음의 태스크 정의 파라미터는 JSON을 통한 구성(Configure via JSON) 옵션을 사용하여 Amazon ECS 콘솔에 태스크 정의를 등록할 때 사용할 수 있습니다. 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

주제

- IPC 모드
- PID 모드
- 결함 주입

IPC 모드

`ipcMode`

유형: 문자열

필수 여부: 아니요

해당 태스크의 컨테이너에 사용할 IPC 리소스 네임스페이스입니다. 유효한 값은 `host`, `task` 또는 `none`입니다. `host`를 지정하면 동일한 컨테이너 인스턴스에서 `host` IPC 모드를 지정한 태스크 내 모든 컨테이너가 동일한 IPC 리소스를 호스트 Amazon EC2 인스턴스와 공유합니다. `task`를 지정하면 지정된 태스크 내 모든 컨테이너가 동일한 IPC 리소스를 공유합니다. `none`이 지정된 경우, 태스크 컨테이너 내에 있는 IPC 리소스는 프라이빗이며, 태스크 또는 컨테이너 인스턴스의 다른 컨테이너와 공유되지 않습니다. 값을 지정하지 않을 경우, IPC 리소스 네임스페이스 공유는 컨테이너 인스턴스의 Docker 데몬 설정에 따라 달라집니다.

`host` IPC 모드를 사용하는 경우, 원치 않는 IPC 네임스페이스 노출이 발생할 위험이 커집니다.

해당 태스크 내 컨테이너에 대해 `systemControls`를 사용하여 네임스페이스가 있는 커널 파라미터를 설정하는 경우, IPC 리소스 네임스페이스에 다음 사항이 적용됩니다.

- `host` IPC 모드를 사용하는 태스크의 경우, IPC 네임스페이스 관련 `systemControls`는 지원되지 않습니다.
- `task` IPC 모드를 사용하는 태스크의 경우, IPC 네임스페이스 관련 `systemControls`가 태스크 내 모든 컨테이너에 적용됩니다.

PID 모드

`pidMode`

유형: 문자열

유효한 값: `host` | `task`

필수 여부: 아니요

해당 태스크의 컨테이너에 사용할 프로세스 네임스페이스입니다. 유효한 값은 `host` 또는 `task`입니다. 예를 들어 사이드카 모니터링에서는 동일한 작업에서 실행 중인 다른 컨테이너에 대한 정보에 액세스하기 위해 `pidMode`가 필요할 수 있습니다.

`host`를 지정하면 동일한 컨테이너 인스턴스에서 `host` PID 모드를 지정한 태스크 내 모든 컨테이너가 동일한 프로세스 네임스페이스를 호스트 Amazon EC2 인스턴스와 공유합니다.

`task`을 지정하면 지정된 태스크 내 모든 컨테이너가 동일한 프로세스 네임스페이스를 공유합니다.

값을 지정하지 않을 경우, 기본값은 각 컨테이너의 프라이빗 네임스페이스입니다.

`host` PID 모드를 사용하는 경우, 원치 않는 프로세스 네임스페이스 노출이 발생할 위험이 커집니다.

> **Note**
>
> 이 파라미터는 Windows 컨테이너에서 지원되지 않습니다.

결함 주입

`enableFaultInjection`

유형: 부울

유효한 값: `true` | `false`

필수 여부: 아니요

이 파라미터가 `true`로 설정된 경우 작업의 페이로드에서 Amazon ECS는 작업 컨테이너의 결함 주입 요청을 수락합니다. 이 파라미터는 기본적으로 `false`로 설정되어 있습니다.

Amazon ECS 태스크 정의 템플릿

빈 태스크 정의 템플릿은 아래와 같습니다. 이 템플릿을 사용하여 태스크 정의를 생성한 다음 AWS CLI `--cli-input-json` 옵션을 사용하여 콘솔 JSON 입력 영역으로 붙여 넣거나 파일로 저장하고 사용할 수 있습니다. 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터 섹션을 참조하세요.

Amazon EC2 시작 유형 템플릿

```
{
  "family": "",
  "taskRoleArn": "",
  "executionRoleArn": "",
  "networkMode": "none",
  "containerDefinitions": [
    {
      "name": "",
      "image": "",
      "repositoryCredentials": {
        "credentialsParameter": ""
      },
      "cpu": 0,
      "memory": 0,
      "memoryReservation": 0,
      "links": [""],
      "portMappings": [
        {
          "containerPort": 0,
          "hostPort": 0,
          "protocol": "tcp"
        }
      ],
      "restartPolicy": {
        "enabled": true,
        "ignoredExitCodes": [0],
        "restartAttemptPeriod": 180
      },
      "essential": true,
      "entryPoint": [""],
      "command": [""],
      "environment": [
        {
          "name": "",
```

```
          "value": ""
        }
      ],
      "environmentFiles": [
        {
          "value": "",
          "type": "s3"
        }
      ],
      "mountPoints": [
        {
          "sourceVolume": "",
          "containerPath": "",
          "readOnly": true
        }
      ],
      "volumesFrom": [
        {
          "sourceContainer": "",
          "readOnly": true
        }
      ],
      "linuxParameters": {
        "capabilities": {
          "add": [""],
          "drop": [""]
        },
        "devices": [
          {
            "hostPath": "",
            "containerPath": "",
            "permissions": ["read"]
          }
        ],
        "initProcessEnabled": true,
        "sharedMemorySize": 0,
        "tmpfs": [
          {
            "containerPath": "",
            "size": 0,
            "mountOptions": [""]
          }
        ],
        "maxSwap": 0,
```

```
        "swappiness": 0
      },
      "secrets": [
        {
          "name": "",
          "valueFrom": ""
        }
      ],
      "dependsOn": [
        {
          "containerName": "",
          "condition": "COMPLETE"
        }
      ],
      "startTimeout": 0,
      "stopTimeout": 0,
      "hostname": "",
      "user": "",
      "workingDirectory": "",
      "disableNetworking": true,
      "privileged": true,
      "readonlyRootFilesystem": true,
      "dnsServers": [""],
      "dnsSearchDomains": [""],
      "extraHosts": [
        {
          "hostname": "",
          "ipAddress": ""
        }
      ],
      "dockerSecurityOptions": [""],
      "interactive": true,
      "pseudoTerminal": true,
      "dockerLabels": {
        "KeyName": ""
      },
      "ulimits": [
        {
          "name": "nofile",
          "softLimit": 0,
          "hardLimit": 0
        }
      ],
      "logConfiguration": {
```

```
        "logDriver": "splunk",
        "options": {
          "KeyName": ""
        },
        "secretOptions": [
          {
            "name": "",
            "valueFrom": ""
          }
        ]
      },
      "healthCheck": {
        "command": [""],
        "interval": 0,
        "timeout": 0,
        "retries": 0,
        "startPeriod": 0
      },
      "systemControls": [
        {
          "namespace": "",
          "value": ""
        }
      ],
      "resourceRequirements": [
        {
          "value": "",
          "type": "InferenceAccelerator"
        }
      ],
      "firelensConfiguration": {
        "type": "fluentbit",
        "options": {
          "KeyName": ""
        }
      }
    }
  ],
  "volumes": [
    {
      "name": "",
      "host": {
        "sourcePath": ""
      },
```

```
      "configuredAtLaunch": true,
      "dockerVolumeConfiguration": {
        "scope": "shared",
        "autoprovision": true,
        "driver": "",
        "driverOpts": {
          "KeyName": ""
        },
        "labels": {
          "KeyName": ""
        }
      },
      "efsVolumeConfiguration": {
        "fileSystemId": "",
        "rootDirectory": "",
        "transitEncryption": "DISABLED",
        "transitEncryptionPort": 0,
        "authorizationConfig": {
          "accessPointId": "",
          "iam": "ENABLED"
        }
      },
      "fsxWindowsFileServerVolumeConfiguration": {
        "fileSystemId": "",
        "rootDirectory": "",
        "authorizationConfig": {
          "credentialsParameter": "",
          "domain": ""
        }
      }
    }
  ],
  "placementConstraints": [
    {
      "type": "memberOf",
      "expression": ""
    }
  ],
  "requiresCompatibilities": ["EC2"],
  "cpu": "",
  "memory": "",
  "tags": [
    {
      "key": "",
```

```
      "value": ""
    }
  ],
  "pidMode": "task",
  "ipcMode": "task",
  "proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "",
    "properties": [
      {
        "name": "",
        "value": ""
      }
    ]
  },
  "inferenceAccelerators": [
    {
      "deviceName": "",
      "deviceType": ""
    }
  ],
  "ephemeralStorage": {
    "sizeInGiB": 0
  },
  "runtimePlatform": {
    "cpuArchitecture": "X86_64",
    "operatingSystemFamily": "WINDOWS_SERVER_20H2_CORE"
  }
}
```

Fargate 시작 유형 템플릿

Important

Fargate 시작 유형의 경우 다음 값 중 하나와 함께 `operatingSystemFamily` 파라미터를 포함해야 합니다.

- `LINUX`
- `WINDOWS_SERVER_2019_FULL`
- `WINDOWS_SERVER_2019_CORE`

- WINDOWS_SERVER_2022_FULL
- WINDOWS_SERVER_2022_CORE

```
{
    "family": "",
    "runtimePlatform": {"operatingSystemFamily": ""},
    "taskRoleArn": "",
    "executionRoleArn": "",
    "networkMode": "awsvpc",
    "platformFamily": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            "repositoryCredentials": {"credentialsParameter": ""},
            "cpu": 0,
            "memory": 0,
            "memoryReservation": 0,
            "links": [""],
            "portMappings": [
                {
                    "containerPort": 0,
                    "hostPort": 0,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [""],
            "command": [""],
            "environment": [
                {
                    "name": "",
                    "value": ""
                }
            ],
            "environmentFiles": [
                {
                    "value": "",
                    "type": "s3"
                }
            ],
```

```
            "mountPoints": [
                {
                    "sourceVolume": "",
                    "containerPath": "",
                    "readOnly": true
                }
            ],
            "volumesFrom": [
                {
                    "sourceContainer": "",
                    "readOnly": true
                }
            ],
            "linuxParameters": {
                "capabilities": {
                    "add": [""],
                    "drop": [""]
                },
                "devices": [
                    {
                        "hostPath": "",
                        "containerPath": "",
                        "permissions": ["read"]
                    }
                ],
                "initProcessEnabled": true,
                "sharedMemorySize": 0,
                "tmpfs": [
                    {
                        "containerPath": "",
                        "size": 0,
                        "mountOptions": [""]
                    }
                ],
                "maxSwap": 0,
                "swappiness": 0
            },
            "secrets": [
                {
                    "name": "",
                    "valueFrom": ""
                }
            ],
            "dependsOn": [
```

```
            {
                "containerName": "",
                "condition": "HEALTHY"
            }
        ],
        "startTimeout": 0,
        "stopTimeout": 0,
        "hostname": "",
        "user": "",
        "workingDirectory": "",
        "disableNetworking": true,
        "privileged": true,
        "readonlyRootFilesystem": true,
        "dnsServers": [""],
        "dnsSearchDomains": [""],
        "extraHosts": [
            {
                "hostname": "",
                "ipAddress": ""
            }
        ],
        "dockerSecurityOptions": [""],
        "interactive": true,
        "pseudoTerminal": true,
        "dockerLabels": {"KeyName": ""},
        "ulimits": [
            {
                "name": "msgqueue",
                "softLimit": 0,
                "hardLimit": 0
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {"KeyName": ""},
            "secretOptions": [
                {
                    "name": "",
                    "valueFrom": ""
                }
            ]
        },
        "healthCheck": {
            "command": [""],
```

```
                "interval": 0,
                "timeout": 0,
                "retries": 0,
                "startPeriod": 0
            },
            "systemControls": [
                {
                    "namespace": "",
                    "value": ""
                }
            ],
            "resourceRequirements": [
                {
                    "value": "",
                    "type": "GPU"
                }
            ],
            "firelensConfiguration": {
                "type": "fluentd",
                "options": {"KeyName": ""}
            }
        }
    ],
    "volumes": [
        {
            "name": "",
            "host": {"sourcePath": ""},
            "configuredAtLaunch":true,
            "dockerVolumeConfiguration": {
                "scope": "task",
                "autoprovision": true,
                "driver": "",
                "driverOpts": {"KeyName": ""},
                "labels": {"KeyName": ""}
            },
            "efsVolumeConfiguration": {
                "fileSystemId": "",
                "rootDirectory": "",
                "transitEncryption": "ENABLED",
                "transitEncryptionPort": 0,
                "authorizationConfig": {
                    "accessPointId": "",
                    "iam": "ENABLED"
                }
```

```
            }
        }
    ],
    "requiresCompatibilities": ["FARGATE"],
    "cpu": "",
    "memory": "",
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "ephemeralStorage": {"sizeInGiB": 0},
    "pidMode": "task",
    "ipcMode": "none",
    "proxyConfiguration": {
        "type": "APPMESH",
        "containerName": "",
        "properties": [
            {
                "name": "",
                "value": ""
            }
        ]
    },
    "inferenceAccelerators": [
        {
            "deviceName": "",
            "deviceType": ""
        }
    ]
}
```

다음 AWS CLI 명령을 사용하여 이 태스크 정의 템플릿을 생성할 수 있습니다.

```
aws ecs register-task-definition --generate-cli-skeleton
```

Amazon ECS 작업 정의 예제

예제와 코드 조각을 복사하여 고유한 작업 정의를 생성할 수 있습니다.

예제를 복사한 다음 클래식 콘솔에서 JSON을 통한 구성 옵션을 사용할 때 붙여넣을 수 있습니다. 계정 ID 사용과 같은 예제를 사용자 지정해야 합니다. 이 코드 조각을 작업 정의 JSON에 포함할 수 있습니다. 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 및 Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터(을)를 참조하세요.

태스크 정의에 대한 추가 예제는 GitHub의 AWS 샘플 태스크 정의를 참조하세요.

주제

- 웹 서버
- splunk 로그 드라이버
- fluentd 로그 드라이버
- gelf 로그 드라이버
- 외부 인스턴스의 워크로드
- Amazon ECR 이미지 및 작업 정의 IAM 역할
- 명령의 진입점
- 컨테이너 종속성
- 작업 정의의 볼륨
- Windows 샘플 태스크 정의

웹 서버

다음은 웹 서버를 설정하는 Fargate 시작 유형에서 Linux 컨테이너를 사용하는 태스크 정의의 예입니다.

```
{
   "containerDefinitions": [
      {
         "command": [
            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</
title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
         ],
         "entryPoint": [
            "sh",
```

```
            "-c"
         ],
         "essential": true,
         "image": "public.ecr.aws/docker/library/httpd:2.4",
         "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
               "awslogs-group" : "/ecs/fargate-task-definition",
               "awslogs-region": "us-east-1",
               "awslogs-stream-prefix": "ecs"
            }
         },
         "name": "sample-fargate-app",
         "portMappings": [
            {
               "containerPort": 80,
               "hostPort": 80,
               "protocol": "tcp"
            }
         ]
      }
   ],
   "cpu": "256",
   "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family": "fargate-task-definition",
   "memory": "512",
   "networkMode": "awsvpc",
   "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
    },
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

다음은 웹 서버를 설정하는 Fargate 시작 유형에서 Windows 컨테이너를 사용하는 태스크 정의의 예입니다.

```
{
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
```

```
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "networkMode": "awsvpc",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["FARGATE"]
}
```

`splunk` 로그 드라이버

다음 코드 조각에서는 로그를 원격 서비스로 보내는 작업 정의에서 `splunk` 로그 드라이버를 사용하는 방법을 보여줍니다. Splunk 토큰 파라미터는 민감한 데이터로 취급될 수 있으므로 암호 옵션으로 지정됩니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.

```
"containerDefinitions": [{
  "logConfiguration": {
   "logDriver": "splunk",
   "options": {
    "splunk-url": "https://cloud.splunk.com:8080",
```

```
    "tag": "tag_name",
   },
   "secretOptions": [{
    "name": "splunk-token",
    "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:splunk-token-
KnrBkD"
}],
```

`fluentd` 로그 드라이버

다음 코드 조각에서는 로그를 원격 서비스로 보내는 작업 정의에서 `fluentd` 로그 드라이버를 사용하는 방법을 보여줍니다. `fluentd-address` 값은 민감한 데이터로 취급될 수 있으므로 암호 옵션으로 지정됩니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.

```
"containerDefinitions": [{
 "logConfiguration": {
  "logDriver": "fluentd",
  "options": {
   "tag": "fluentd demo"
  },
  "secretOptions": [{
   "name": "fluentd-address",
   "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:fluentd-address-
KnrBkD"
  }]
 },
 "entryPoint": [],
 "portMappings": [{
             "hostPort": 80,
             "protocol": "tcp",
             "containerPort": 80
             },
             {
  "hostPort": 24224,
  "protocol": "tcp",
  "containerPort": 24224
 }]
}],
```

`gelf` 로그 드라이버

다음 코드 조각에서는 Gelf 로그를 입력으로 사용해 Logstash를 실행 중인 원격 호스트로 로그를 보내는 작업 정의에서 `gelf` 로그 드라이버를 사용하는 방법을 보여줍니다. 자세한 내용은 `logConfiguration` 섹션을 참조하세요.

```
"containerDefinitions": [{
 "logConfiguration": {
  "logDriver": "gelf",
  "options": {
   "gelf-address": "udp://logstash-service-address:5000",
   "tag": "gelf task demo"
  }
 },
 "entryPoint": [],
 "portMappings": [{
   "hostPort": 5000,
   "protocol": "udp",
   "containerPort": 5000
  },
  {
   "hostPort": 5000,
   "protocol": "tcp",
   "containerPort": 5000
  }
 ]
}],
```

외부 인스턴스의 워크로드

Amazon ECS 태스크 정의를 등록할 때는 `requiresCompatibilities` 파라미터를 사용하고 태스크 정의가 호환되는지 확인하는 `EXTERNAL`을 지정하여 외부 인스턴스에서 Amazon ECS 워크로드를 실행할 때 사용합니다. 콘솔을 사용하여 작업 정의를 등록하는 경우 JSON 편집기를 사용해야 합니다. 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

> **Important**
>
> 태스크에 태스크 실행 IAM 역할이 필요한 경우 해당 역할이 태스크 정의에 지정되어 있는지 확인합니다.

워크로드를 배포할 때 서비스를 생성하거나 독립 실행형 태스크를 실행하는 경우 EXTERNAL 시작 유형을 사용합니다.

다음은 예제 태스크 정의입니다.

Linux

```
{
 "requiresCompatibilities": [
  "EXTERNAL"
 ],
 "containerDefinitions": [{
  "name": "nginx",
  "image": "public.ecr.aws/nginx/nginx:latest",
  "memory": 256,
  "cpu": 256,
  "essential": true,
  "portMappings": [{
   "containerPort": 80,
   "hostPort": 8080,
   "protocol": "tcp"
  }]
 }],
 "networkMode": "bridge",
 "family": "nginx"
}
```

Windows

```
{
 "requiresCompatibilities": [
  "EXTERNAL"
 ],
 "containerDefinitions": [{
  "name": "windows-container",
  "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
  "memory": 256,
  "cpu": 512,
  "essential": true,
  "portMappings": [{
   "containerPort": 80,
   "hostPort": 8080,
   "protocol": "tcp"
```

```
  }]
 }],
 "networkMode": "bridge",
 "family": "windows-container"
}
```

Amazon ECR 이미지 및 작업 정의 IAM 역할

다음 코드 조각에서는 123456789012.dkr.ecr.us-west-2.amazonaws.com 레지스트리의 v1 태그를 포함하는 aws-nodejs-sample이라는 Amazon ECR 이미지를 사용합니다. 이 태스크의 컨테이너는 IAM 역할로부터 arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole 권한을 상속합니다. 자세한 내용은 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.

```
{
    "containerDefinitions": [
        {
            "name": "sample-app",
            "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/aws-nodejs-
sample:v1",
            "memory": 200,
            "cpu": 10,
            "essential": true
        }
    ],
    "family": "example_task_3",
    "taskRoleArn": "arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole"
}
```

명령의 진입점

다음 코드 조각에서는 진입점과 명령 인수를 사용하는 Docker 컨테이너용 구문을 보여줍니다. 이 컨테이너는 example.com을 네 번 ping한 후 종료합니다.

```
{
    "containerDefinitions": [
        {
            "memory": 32,
            "essential": true,
            "entryPoint": ["ping"],
            "name": "alpine_ping",
            "readonlyRootFilesystem": true,
```

```
            "image": "alpine:3.4",
            "command": [
                "-c",
                "4",
                "example.com"
            ],
            "cpu": 16
        }
    ],
    "family": "example_task_2"
}
```

컨테이너 종속성

이 코드 조각에서는 컨테이너 종속성이 지정된 여러 컨테이너가 있는 작업 정의의 구문을 보여줍니다. 다음 태스크 정의에서 envoy 컨테이너를 시작하기 전에 app 컨테이너가 필요한 컨테이너 상태 확인 파라미터에 의해 정상 상태에 도달해야 합니다. 자세한 내용은 컨테이너 종속성 섹션을 참조하세요.

```
{
  "family": "appmesh-gateway",
  "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
  },
  "proxyConfiguration":{
      "type": "APPMESH",
      "containerName": "envoy",
      "properties": [
          {
              "name": "IgnoredUID",
              "value": "1337"
          },
          {
              "name": "ProxyIngressPort",
              "value": "15000"
          },
          {
              "name": "ProxyEgressPort",
              "value": "15001"
          },
          {
              "name": "AppPorts",
              "value": "9080"
          },
```

```
            {
                "name": "EgressIgnoredIPs",
                "value": "169.254.170.2,169.254.169.254"
            }
        ]
    },
    "containerDefinitions": [
      {
        "name": "app",
        "image": "application_image",
        "portMappings": [
          {
            "containerPort": 9080,
            "hostPort": 9080,
            "protocol": "tcp"
          }
        ],
        "essential": true,
        "dependsOn": [
          {
            "containerName": "envoy",
            "condition": "HEALTHY"
          }
        ]
      },
      {
        "name": "envoy",
        "image": "840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-
envoy:v1.15.1.0-prod",
        "essential": true,
        "environment": [
          {
            "name": "APPMESH_VIRTUAL_NODE_NAME",
            "value": "mesh/meshName/virtualNode/virtualNodeName"
          },
          {
            "name": "ENVOY_LOG_LEVEL",
            "value": "info"
          }
        ],
        "healthCheck": {
          "command": [
            "CMD-SHELL",
            "echo hello"
```

```
        ],
        "interval": 5,
        "timeout": 2,
        "retries": 3
      }
    }
  ],
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}
```

작업 정의의 볼륨

다음을 사용하여 작업에서 볼륨을 지정하는 방법을 이해합니다.

- Amazon EBS 볼륨을 구성하는 방법에 대한 자세한 내용은 Amazon ECS 배포 시 Amazon EBS 볼륨 구성 지정 섹션을 참조하세요.
- Amazon EFS 볼륨을 구성하는 방법에 대한 자세한 내용은 콘솔을 사용하여 Amazon ECS에 대한 Amazon EFS 파일 시스템 구성 섹션을 참조하세요.
- FSx for Windows File Server 볼륨을 구성하는 방법에 대한 자세한 내용은 Amazon ECS에 대한 FSx for Windows File Server 파일 시스템을 구성하는 방법 알아보기 섹션을 참조하세요.
- Docker 볼륨을 구성하는 방법에 대한 자세한 내용은 Amazon ECS용 Docker 볼륨 예제 섹션을 참조하세요.
- 바인드 탑재를 구성하는 방법에 대한 자세한 내용은 Amazon ECS에 대한 바인드 탑재 예제 섹션을 참조하세요.

Windows 샘플 태스크 정의

Amazon ECS에서 Windows 컨테이너를 시작하는 데 도움을 줄 샘플 태스크 정의는 아래와 같습니다.

Example Windows용 Amazon ECS 콘솔 샘플 애플리케이션

다음 태스크 정의는 Amazon ECS 처음 실행 마법사에서 생성되는 Amazon ECS 콘솔 샘플 애플리케이션으로서 `microsoft/iis` Windows 컨테이너 이미지를 사용하도록 이식되었습니다.

```
{
  "family": "windows-simple-iis",
  "containerDefinitions": [
    {
```

```
      "name": "windows_sample_app",
      "image": "mcr.microsoft.com/windows/servercore/iis",
      "cpu": 1024,
      "entryPoint":["powershell", "-Command"],
      "command":["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -
Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80
        }
      ],
      "memory": 1024,
      "essential": true
    }
  ],
  "networkMode": "awsvpc",
  "memory": "1024",
  "cpu": "1024"
}
```

Amazon ECS 클러스터

Amazon ECS 클러스터는 작업 또는 서비스의 논리적 그룹입니다. 클러스터는 작업 및 서비스 외에도 다음과 같은 리소스로 구성됩니다.

- 인프라 용량은 다음의 조합일 수 있습니다.
  - AWS 클라우드의 Amazon EC2 인스턴스
  - AWS 클라우드의 서버리스(AWS Fargate)
  - 온프레미스 가상 머신(VM) 또는 서버
- 작업과 서비스가 실행되는 네트워크(VPC 및 서브넷)

  용량으로 Amazon EC2 인스턴스를 사용하는 경우 서브넷은 가용 영역, 로컬 영역, Wavelength 영역 또는 AWS Outposts에 있을 수 있습니다.
- 선택적 네임스페이스

  네임스페이스는 서비스 연결과의 서비스 간 통신에 사용됩니다.
- 모니터링 옵션

  CloudWatch Container Insights는 추가 비용이 발생하며 완전 관리형 서비스입니다. 이 서비스는 Amazon ECS 지표 및 로그를 자동으로 수집, 집계 및 요약합니다.

다음은 Amazon ECS 클러스터에 관한 일반 개념입니다.

- 클러스터를 생성하여 리소스를 분리합니다.
- 클러스터는 AWS 리전별로 고유합니다.
- 클러스터는 다음 상태 중 하나일 수 있습니다.

  ACTIVE

  클러스터는 해당하는 태스크에 적용될 준비가 되었으며, 클러스터로 컨테이너 인스턴스를 등록할 수 있습니다.

  PROVISIONING

  클러스터에 연결된 용량 공급자가 있으며 용량 공급자에게 필요한 리소스가 생성 중입니다.

  DEPROVISIONING

  클러스터에 연결된 용량 공급자가 있으며 용량 공급자에게 필요한 리소스를 삭제 중입니다.

FAILED

클러스터에 연결된 용량 공급자가 있으며 용량 공급자에게 필요한 리소스를 생성하는 데 실패했습니다.

INACTIVE

클러스터가 삭제되었습니다. `INACTIVE` 상태인 클러스터는 일정 기간 동안 계정에서 검색 가능한 상태로 유지될 수 있습니다. 이 동작은 향후 변경될 수 있으므로 `INACTIVE` 클러스터가 지속되는 상태에 의존해서는 안 됩니다.

- EC2 시작 유형 또는 Auto Scaling 그룹 용량 공급자에 대해 다른 인스턴스 유형을 사용할 수 있습니다. 인스턴스는 한 번에 하나의 클러스터에만 등록할 수 있습니다.
- 사용자 지정 IAM 정책을 생성하여 클러스터에 대한 액세스를 제한할 수 있습니다. 자세한 정보는 Amazon Elastic Container Service의 자격 증명 기반 정책 예의 Amazon ECS 클러스터 예제 섹션을 참조하세요.
- Service Auto Scaling을 사용하여 Fargate 태스크의 규모를 조정할 수 있습니다. 자세한 내용은 Amazon ECS 서비스 자동 조정 섹션을 참조하세요.
- 클러스터에 대한 기본 Service Connect 네임스페이스를 구성할 수 있습니다. 기본 Service Connect 네임스페이스를 설정한 후 Service Connect를 켜서 네임스페이스의 클라이언트 서비스로 클러스터에 생성된 모든 새 서비스를 추가할 수 있습니다. 추가 구성은 필요하지 않습니다. 자세한 내용은 Service Connect를 사용하여 짧은 이름으로 Amazon ECS 서비스 연결 섹션을 참조하세요.

용량 공급자

Amazon ECS 용량 공급자는 클러스터의 작업에 대한 인프라의 조정을 관리할 수 있습니다. 각 클러스터에는 하나 이상의 용량 공급자와 하나의 선택적 용량 공급자 전략이 있을 수 있습니다. 클러스터에 기본 용량 공급자 전략을 할당할 수 있습니다. 용량 공급자 전략은 태스크가 클러스터의 용량 공급자에 분배되는 방식을 결정합니다. 독립 실행형 태스크를 실행하거나 서비스를 생성할 때 클러스터의 기본 용량 공급자 전략 또는 기본 용량 공급자 전략을 재정의하는 용량 공급자 전략을 사용합니다. 클러스터의 기본 용량 공급자 전략은 시작 유형 또는 태스크 또는 서비스의 용량 공급자 전략을 지정하지 않은 경우에만 적용됩니다. 이러한 파라미터 중 하나를 제공하면 기본 전략이 사용되지 않습니다.

Fargate의 경우 용량을 생성하거나 관리할 필요가 없습니다. 다음의 사전 정의된 용량 공급자를 클러스터와 연결하기만 하면 됩니다.

- Fargate
- Fargate 스팟

용량으로 EC2 인스턴스를 사용하는 경우 Auto Scaling 그룹을 사용하여 EC2 인스턴스를 관리합니다. 오토 스케일링은 애플리케이션 로드를 처리하는 데 사용할 수 있는 EC2 인스턴스의 정확한 수를 확보하는 데 도움이 됩니다.

클러스터는 AWS Fargate, Amazon EC2 인스턴스 또는 외부 인스턴스에서 호스팅되는 작업이 섞여 있을 수 있습니다. 작업은 시작 유형 또는 용량 공급자 전략으로 Fargate 또는 EC2 인프라에서 실행할 수 있습니다. EC2를 시작 유형으로 사용하는 경우 Amazon ECS는 Amazon EC2 Auto Scaling 그룹의 용량을 추적하거나 규모를 조정하지 않습니다. 시작 유형에 대한 자세한 내용은 Amazon ECS 시작 유형 섹션을 참조하세요.

클러스터에 Auto Scaling 용량 공급자와 Fargate 용량 공급자가 혼합되어 있을 수 있습니다. 용량 공급자 전략에는 Auto Scaling 그룹 용량 공급자 또는 Fargate 용량 공급자만 포함될 수 있습니다.

Fargate용 Amazon ECS 클러스터

Amazon ECS 용량 공급자는 클러스터의 작업에 대한 인프라의 조정을 관리할 수 있습니다. 각 클러스터에는 하나 이상의 용량 공급자와 하나의 선택적 용량 공급자 전략이 있을 수 있습니다. 용량 공급자 전략은 태스크가 클러스터의 용량 공급자에 분배되는 방식을 결정합니다. 표준 실행 작업을 실행하거나 서비스를 생성할 때 클러스터의 기본 용량 공급자 전략 또는 기본 전략을 재정의하는 용량 공급자 전략을 사용할 수 있습니다.

AWS Fargate에서 작업을 실행할 때는 용량을 생성하거나 관리할 필요가 없습니다. 다음의 사전 정의된 용량 공급자를 클러스터와 연결하기만 하면 됩니다.

- Fargate
- Fargate 스팟

AWS Fargate 용량 공급자의 Amazon ECS를 사용하면 Amazon ECS 작업에서 Fargate 및 Fargate Spot 용량을 모두 사용할 수 있습니다.

Fargate Spot을 사용하면 Fargate 가격 대비 할인된 요금으로 중단 방지 Amazon ECS 작업을 실행할 수 있습니다. Fargate 스팟은 여분의 컴퓨팅 용량에 대한 태스크를 실행합니다. AWS에 용량이 다시 필요한 경우 2분간 경고한 후에 작업이 중단됩니다.

Fargate 및 Fargate Spot 용량 공급자를 사용하는 작업이 중지되면 작업 상태 변경 이벤트가 Amazon EventBridge로 전송됩니다. 중지된 이유는 원인을 설명합니다. 자세한 내용은 Amazon ECS 작업 상태 변경 이벤트 섹션을 참조하세요.

클러스터에 Fargate 용량 공급자와 Auto Scaling 그룹 용량 공급자가 혼합되어 있을 수 있습니다. 그러나 용량 공급자 전략에 Fargate 또는 Auto Scaling 그룹 용량 공급자만 포함될 수 있으며 둘 다 포함될 수는 없습니다. 자세한 내용은 Auto Scaling 그룹 용량 공급자를 참조하세요.

용량 공급자를 사용할 때는 다음 사항을 고려해야 합니다.

- 용량 공급자를 클러스터와 연결하려면 먼저 클러스터를 용량 공급자 전략과 연결해야 합니다.
- 용량 공급자 전략에 최대 20개의 용량 공급자를 지정할 수 있습니다.
- Fargate 용량 공급자를 사용하기 위해 Auto Scaling 그룹 용량 공급자를 사용하여 서비스를 업데이트할 수 없습니다. 반대의 경우도 마찬가지입니다.
- 용량 공급자 전략에서 콘솔의 용량 공급자에 대해 `weight` 값이 지정되지 않으면 기본값 `1`을 사용합니다. API 또는 AWS CLI를 사용할 경우에 기본값은 `0`입니다.
- 용량 공급자 전략 내에서 여러 용량 공급자가 지정된 경우 하나 이상의 용량 공급자가 0보다 큰 가중치 값을 가져야 합니다. 가중치가 0인 용량 공급자는 태스크를 배치하는 데 사용되지 않습니다. 전략에 가중치가 모두 0으로 동일한 여러 개의 용량 공급자를 지정하는 경우 용량 공급자 전략을 사용하는 `RunTask` 또는 `CreateService` 작업이 실패합니다.
- 용량 공급자 전략에서는 하나의 용량 공급자만 정의된 base 값을 가질 수 있습니다. base 값을 지정하지 않으면 기본값 0이 사용됩니다.
- 클러스터에 Auto Scaling 그룹 용량 공급자와 Fargate 용량 공급자가 혼합되어 있을 수 있습니다. 그러나 용량 공급자 전략에 Auto Scaling 그룹 또는 Fargate 용량 공급자만 포함될 수 있으며 둘 다 포함될 수는 없습니다.
- 클러스터에는 용량 공급자와 시작 유형을 모두 사용하는 서비스와 독립 실행형 작업이 혼합되어 포함될 수 있습니다. 시작 유형이 아닌 용량 공급자 전략을 사용하도록 서비스를 업데이트할 수 있습니다. 하지만 이 경우 새 배포를 강제로 실행해야 합니다.

Fargate Spot 종료 알림

수요가 매우 많은 기간에는 Fargate 스팟 용량을 사용할 수 없을 수 있습니다. 이로 인해 Fargate 스팟 작업이 지연될 수 있습니다. 이러한 경우 필요한 용량을 사용할 수 있을 때까지 Amazon ECS 서비스에서 태스크 시작을 다시 시도합니다. Fargate는 스팟 용량을 온디맨드 용량으로 대체하지 않습니다.

스팟 중단으로 인해 Fargate Spot 용량을 사용하는 태스크가 중지되면 태스크가 중단되기 전에 2분 간 경고가 전송됩니다. 경고는 작업 상태 변경 이벤트로 Amazon EventBridge에 전송되고 실행 중인 작업에 SIGTERM 신호로 전송됩니다. Fargate Spot을 서비스의 일부로 사용할 경우 이 시나리오에서 서비스 스케줄러는 중단 신호를 수신하고 사용 가능한 용량이 있는 경우 Fargate Spot에 대한 추가 작업의

시작을 시도합니다. 작업이 하나뿐인 서비스는 용량을 사용할 수 있을 때까지 중단됩니다. 정상 종료에 대한 자세한 내용은 Graceful shutdowns with ECS를 참조하세요.

작업이 중지되기 전에 컨테이너가 정상적으로 종료되도록 하려면 다음을 구성할 수 있습니다.

- 태스크가 사용 중인 컨테이너 정의에서 `stopTimeout` 초 이하의 `120` 값을 지정할 수 있습니다. 기본 `stopTimeout` 값은 30초입니다. `stopTimeout` 값을 더 길게 지정하면 작업 상태 변경 이벤트가 수신되는 순간부터 컨테이너가 강제로 중지되는 시점 사이의 시간이 더 많이 확보됩니다. 자세한 내용은 컨테이너 제한 시간 섹션을 참조하세요.
- SIGTERM 신호를 컨테이너 내에서 수신해야 정리 태스크를 수행할 수 있습니다. 이 신호를 처리하는 데 실패하면 작업은 구성된 `stopTimeout` 이후에 SIGKILL 신호를 받는 데 실패하여 데이터가 손실되거나 오류가 발생할 수 있습니다.

다음은 작업 상태 변경 이벤트의 코드 조각입니다. 이 코드 조각에는 Fargate Spot 중단에 대한 중지 사유와 중지 코드가 표시됩니다.

```
{
  "version": "0",
  "id": "9bcdac79-b31f-4d3d-9410-fbd727c29fab",
  "detail-type": "ECS Task State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "resources": [
    "arn:aws:ecs:us-east-1:111122223333:task/b99d40b3-5176-4f71-9a52-9dbd6f1cebef"
  ],
  "detail": {
    "clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",
    "createdAt": "2016-12-06T16:41:05.702Z",
    "desiredStatus": "STOPPED",
    "lastStatus": "RUNNING",
    "stoppedReason": "Your Spot Task was interrupted.",
    "stopCode": "SpotInterruption",
    "taskArn": "arn:aws:ecs:us-east-1:111122223333:task/
b99d40b3-5176-4f71-9a52-9dbd6fEXAMPLE",
    ...
  }
}
```

다음은 Amazon ECS 작업 상태 변경 이벤트에 대한 EventBridge 규칙을 생성하는 데 사용되는 이벤트 패턴입니다. 필요한 경우 `detail` 필드에 클러스터를 지정할 수 있습니다. 이렇게 하면 해당 클러

스터에 대한 작업 상태 변경 이벤트를 수신하게 됩니다. EventBridge 규칙 생성에 대한 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge 시작하기를 참조하세요.

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Task State Change"
    ],
    "detail": {
        "clusterArn": [
            "arn:aws:ecs:us-west-2:111122223333:cluster/default"
        ]
    }
}
```

Fargate 시작 유형에 대한 Amazon ECS 클러스터 생성

클러스터를 생성하여 태스크와 서비스가 실행되는 인프라를 정의합니다.

시작하기 전에 먼저 Amazon ECS 사용 설정 단계를 완료하고 적절한 IAM 권한을 할당했는지 확인합니다. 자세한 내용은 the section called “Amazon ECS 클러스터 예제” 섹션을 참조하세요. Amazon ECS 콘솔은 AWS CloudFormation 스택을 생성하여 Amazon ECS 클러스터에 필요한 리소스를 생성합니다.

콘솔은 Fargate 및 Fargate Spot 용량 공급자를 클러스터와 자동으로 연결합니다.

다음 옵션을 수정할 수 있습니다.

- 클러스터에 네임스페이스를 추가합니다.

  네임스페이스를 사용하면 클러스터에서 생성한 서비스를 추가 구성 없이 네임스페이스의 다른 서비스에 연결할 수 있습니다. 자세한 내용은 Amazon ECS 서비스 상호 연결 섹션을 참조하세요.
- 관찰성이 향상된 Container Insights 또는 Container Insights를 켭니다.

  CloudWatch Container Insights는 컨테이너 애플리케이션 및 마이크로서비스의 지표 및 로그를 수집하고, 종합하며, 요약합니다. 또한 Container Insights는 컨테이너 재시작 오류 등의 진단 정보를 제공합니다. 이 정보를 사용하여 문제를 격리하고 신속하게 해결할 수 있습니다. 자세한 내용은 the section called “관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링” 섹션을 참조하세요.

2024년 12월 2일 AWS에서는 Amazon ECS의 향상된 관찰성을 갖춘 Container Insights를 출시했습니다. 이 버전은 Amazon EC2 및 Fargate 시작 유형을 사용하여 Amazon ECS 클러스터의 관찰성을 향상할 수 있도록 지원합니다. Amazon ECS에서 향상된 관찰성으로 Container Insights를 구성하면, Container Insights는 클러스터 수준에서 환경의 컨테이너 수준까지 상세한 인프라 원격 측정을 자동으로 수집하고, 대시보드에서 다양한 지표와 차원을 보여주는 데이터를 표시합니다. 그런 다음 Container Insights 콘솔에서 이러한 기본 대시보드를 사용하여 컨테이너 상태와 성능을 더 잘 이해하고 이상 징후를 식별하여 문제를 더 빠르게 완화할 수 있습니다.

Container Insights 대신 관찰성이 향상된 Container Insights를 사용하는 것이 좋습니다. 컨테이너 환경에서 세부적인 가시성을 제공하여 평균 해결 시간을 단축하기 때문입니다.

- 클러스터를 식별하는 데 도움이 되는 태그를 추가합니다.
- 관리형 스토리지에 AWS KMS 키를 할당합니다. 키 생성 방법에 대한 자세한 내용은 AWS Key Management Service 사용 설명서의 KMS 키 생성을 참조하세요.
- Fargate 임시 스토리지에 AWS KMS 키를 할당합니다. 키 생성 방법에 대한 자세한 내용은 AWS Key Management Service 사용 설명서의 KMS 키 생성을 참조하세요.

절차

새 클러스터 생성(Amazon ECS 콘솔)

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터(Clusters) 페이지에서 클러스터 생성(Create cluster)을 선택합니다.
5. 클러스터 구성에서 다음을 구성합니다.
   - 클러스터 이름에 고유한 이름을 입력합니다.

     이름은 최대 255자(대/소문자), 숫자 및 하이픈을 포함할 수 있습니다.
   - (선택 사항) 서비스 연결에 사용되는 네임스페이스를 클러스터 이름과 다르게 하려면 네임스페이스에 고유한 이름을 입력합니다.
6. (선택 사항) Container Insights를 사용하고 모니터링을 확장한 후에 다음 옵션 중 하나를 선택합니다.
   - 권장되는 관찰성이 향상된 Container Insights를 사용하려면 관찰성이 향상된 Container Insights를 선택합니다.

- Container Insights를 사용하려면 Container Insights를 선택합니다.

7. (선택 사항) 암호화에서 다음을 구성할 수 있습니다.

   - Fargate 임시 스토리지에서 데이터를 암호화합니다. 암호화에서 Fargate 임시 스토리지에 Fargate 임시 스토리지 데이터를 암호화하는 데 사용할 AWS KMS 키의 ARN을 입력합니다.
   - 관리형 스토리지의 데이터를 암호화합니다. 암호화에서 관리형 스토리지에 관리형 스토리지 데이터를 암호화하는 데 사용할 AWS KMS 키의 ARN을 입력합니다.

8. (선택 사항) 클러스터를 식별하려면 태그(Tags)를 펼친 다음, 태그를 구성합니다.

   [태그 추가] 태그 추가(Add tag)를 선택하고 다음을 수행합니다.

   - 키(Key)에 키 이름을 입력합니다.
   - 값에 키 값을 입력합니다.

   [태그 제거] 태그의 키와 값 오른쪽에 있는 제거를 선택합니다.

9. 생성(Create)을 선택합니다.

다음 단계

클러스터를 생성한 후 애플리케이션에 대한 작업 정의를 생성한 다음 독립 실행형 작업으로 실행하거나 서비스의 일부로 실행할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- Amazon ECS 작업 정의
- 애플리케이션을 Amazon ECS 태스크로 실행
- Amazon ECS 롤링 업데이트 배포 생성

EC2 시작 유형을 위한 Amazon ECS 용량 공급자

용량으로 Amazon EC2 인스턴스를 사용하는 경우 Auto Scaling 그룹을 사용하여 클러스터에 등록된 Amazon EC2 인스턴스를 관리합니다. Auto Scaling은 애플리케이션 로드를 처리하는 데 사용할 수 있는 Amazon EC2 인스턴스의 정확한 수를 확보하는 데 도움이 됩니다.

Managed Scaling 기능을 사용하여 Auto Scaling 그룹의 스케일 인 및 스케일 아웃 작업을 Amazon ECS가 관리하도록 하거나 직접 규모 조정 작업을 관리할 수 있습니다. 자세한 내용은 클러스터 Auto Scaling을 통해 Amazon ECS 용량 자동 관리 섹션을 참조하세요.

새로운 빈 Auto Scaling 그룹을 생성하는 것이 좋습니다. 기존 Auto Scaling 그룹을 사용하는 경우 용량 공급자를 생성하는 데 사용되는 Auto Scaling 그룹에 앞서 이미 실행 중이고 Amazon ECS 클러스터에 등록된 그룹과 연결된 Amazon EC2 인스턴스가 용량 공급자에 제대로 등록되지 않을 수 있습니다. 이럴 경우 용량 공급자 전략에서 용량 공급자를 사용할 때 문제가 발생할 수 있습니다. `DescribeContainerInstances`를 사용하면 컨테이너 인스턴스가 용량 공급자와 연결되어 있는지를 확인할 수 있습니다.

> **Note**
>
> 빈 Auto Scaling 그룹을 생성하려면 원하는 카운트를 0으로 설정합니다. 용량 공급자를 생성하고 클러스터와 연결한 후에 스케일 아웃할 수 있습니다.
> Amazon ECS 콘솔을 사용하는 경우 Amazon ECS는 AWS CloudFormation 스택의 일부로 사용자를 대신하여 Amazon EC2 시작 템플릿 및 Auto Scaling 그룹을 생성합니다. `EC2ContainerService-<ClusterName>` 접두사가 붙습니다. Auto Scaling 그룹을 해당 클러스터의 용량 공급자로 사용할 수 있습니다.

워크로드에 지장을 주지 않으면서 Amazon EC2 인스턴스를 정상적으로 종료할 수 있도록 관리형 인스턴스 드레이닝을 사용하는 것이 좋습니다. 이 기능은 기본적으로 켜져 있습니다. 자세한 내용은 EC2 인스턴스에서 실행되는 Amazon ECS 워크로드를 안전하게 중지 섹션을 참조하세요.

클래식 콘솔에서 Auto Scaling 그룹 용량 공급자를 사용할 경우 다음 사항을 고려합니다.

- Auto Scaling 그룹이 스케일 아웃하려면 `MaxSize` 가 0보다 커야 합니다.
- Auto Scaling 그룹에는 인스턴스 가중치 설정이 있을 수 없습니다.
- Auto Scaling 그룹이 실행되는 작업 수에 맞게 스케일 아웃할 수 없는 경우 작업이 `PROVISIONING` 상태를 지나 전환되지 못합니다.
- 용량 공급자가 관리하는 Auto Scaling 그룹과 연결된 조정 정책 리소스를 수정하지 마세요.
- 용량 공급자를 생성하 때 관리형 조정이 켜진 경우 Auto Scaling의 원하는 수를 `0`으로 설정할 수 있습니다. 관리형 조정이 켜지면 Amazon ECS에서 Auto Scaling 그룹의 스케일 인 및 스케일 아웃 작업을 관리합니다.
- 용량 공급자를 클러스터와 연결하려면 먼저 클러스터를 용량 공급자 전략과 연결해야 합니다.
- 용량 공급자 전략에 최대 20개의 용량 공급자를 지정할 수 있습니다.
- Fargate 용량 공급자를 사용하기 위해 Auto Scaling 그룹 용량 공급자를 사용하여 서비스를 업데이트할 수 없습니다. 반대의 경우도 마찬가지입니다.

- 용량 공급자 전략에서 콘솔의 용량 공급자에 대해 `weight` 값이 지정되지 않으면 기본값 `1`을 사용합니다. API 또는 AWS CLI를 사용할 경우에 기본값은 `0`입니다.
- 용량 공급자 전략 내에서 여러 용량 공급자가 지정된 경우 하나 이상의 용량 공급자가 0보다 큰 가중치 값을 가져야 합니다. 가중치가 0인 용량 공급자는 태스크를 배치하는 데 사용되지 않습니다. 전략에 가중치가 모두 0으로 동일한 여러 개의 용량 공급자를 지정하는 경우 용량 공급자 전략을 사용하는 `RunTask` 또는 `CreateService` 작업이 실패합니다.
- 용량 공급자 전략에서는 하나의 용량 공급자만 정의된 base 값을 가질 수 있습니다. base 값을 지정하지 않으면 기본값 0이 사용됩니다.
- 클러스터에 Auto Scaling 그룹 용량 공급자와 Fargate 용량 공급자가 혼합되어 있을 수 있습니다. 그러나 용량 공급자 전략에 Auto Scaling 그룹 또는 Fargate 용량 공급자만 포함될 수 있으며 둘 다 포함될 수는 없습니다.
- 클러스터에는 용량 공급자와 시작 유형을 모두 사용하는 서비스와 독립 실행형 작업이 혼합되어 포함될 수 있습니다. 시작 유형이 아닌 용량 공급자 전략을 사용하도록 서비스를 업데이트할 수 있습니다. 하지만 이 경우 새 배포를 강제로 실행해야 합니다.
- Amazon ECS에서 Amazon EC2 Auto Scaling 웜 풀을 지원합니다. 웜 풀은 서비스에 배치할 준비가 되어 사전 초기화된 Amazon EC2 인스턴스의 그룹입니다. 애플리케이션을 스케일 아웃해야 할 때마다 Amazon EC2 Auto Scaling은 콜드 인스턴스를 실행하는 대신 웜 풀에서 미리 초기화된 인스턴스를 사용합니다. 이렇게 하면 인스턴스가 서비스에 배치되기 전에 최종 초기화 프로세스를 실행할 수 있습니다. 자세한 내용은 Amazon ECS Auto Scaling 그룹에 대해 사전 초기화된 인스턴스 구성 섹션을 참조하세요.

Amazon EC2 Auto Scaling 시작 템플릿 생성에 대한 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 Auto Scaling 시작 템플릿을 참조하세요. Amazon EC2 Auto Scaling 그룹 생성에 대한 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 오토 스케일링을 참조하세요.

Amazon ECS에 대한 Amazon EC2 컨테이너 인스턴스 보안 고려 사항

위협 모델 내에서 단일 컨테이너 인스턴스와 이 인스턴스의 액세스를 고려해야 합니다. 예를 들어 영향을 받는 단일 작업이 동일한 인스턴스에서 감염되지 않은 작업의 IAM 권한을 활용할 수 있습니다.

이러한 문제를 방지하려면 다음을 사용하는 것이 좋습니다.

- 작업을 실행할 때 관리자 권한을 사용하지 마세요.
- 작업에 대한 최소 액세스 권한을 갖는 작업 역할을 할당합니다.

  컨테이너 에이전트는 Amazon ECS 리소스에 액세스하는 데 사용되는 고유한 보안 인증 ID가 있는 토큰을 자동으로 생성합니다.

- awsvpc 네트워크 모드를 사용하는 작업에서 실행하는 컨테이너가 Amazon EC2 인스턴스 프로파일에 제공된 보안 인증 정보에 액세스하지 못하게 방지하려면(단, 작업 역할에 제공된 권한은 허용) 에이전트 구성 파일에서 ECS_AWSVPC_BLOCK_IMDS 에이전트 구성 변수를 true로 설정하고 에이전트를 다시 시작합니다.
- Amazon GuardDuty Runtime Monitoring을 사용하여 AWS 환경 내 클러스터 및 컨테이너에 대한 위협을 감지합니다. Runtime Monitoring은 파일 액세스, 프로세스 실행 및 네트워크 연결과 같은 개별 Amazon ECS 워크로드에 대한 런타임 가시성을 추가하는 GuardDuty 보안 에이전트를 사용합니다. 자세한 내용은 GuardDuty 사용 설명서의 GuardDuty Runtime Monitoring을 참조하세요.

Amazon EC2 시작 유형에 대한 Amazon ECS 클러스터 생성

클러스터를 생성하여 태스크와 서비스가 실행되는 인프라를 정의합니다.

시작하기 전에 먼저 Amazon ECS 사용 설정 단계를 완료하고 적절한 IAM 권한을 할당했는지 확인합니다. 자세한 내용은 the section called “Amazon ECS 클러스터 예제” 섹션을 참조하세요. Amazon ECS 콘솔은 AWS CloudFormation 스택을 생성하여 Amazon ECS 클러스터에 필요한 리소스를 생성하는 간단한 방법을 제공합니다.

클러스터 생성 프로세스를 가능한 한 쉽게 만들기 위해 콘솔에는 아래에서 설명하는 여러 선택 항목에 대한 기본 선택 항목이 있습니다. 추가 컨텍스트를 제공하는 콘솔의 대부분의 섹션에서 사용할 수 있는 도움말 패널도 있습니다.

클러스터를 생성할 때 Amazon EC2 인스턴스를 등록하거나 클러스터를 생성한 후 클러스터에 추가 인스턴스를 등록할 수 있습니다.

다음 기본 옵션을 수정할 수 있습니다.

- 인스턴스가 시작되는 서브넷을 변경합니다.
- 컨테이너 인스턴스에 대한 트래픽을 제어하는 데 사용되는 보안 그룹을 변경합니다.
- 클러스터에 네임스페이스를 추가합니다.

  네임스페이스를 사용하면 클러스터에서 생성한 서비스를 추가 구성 없이 네임스페이스의 다른 서비스에 연결할 수 있습니다. 자세한 내용은 Amazon ECS 서비스 상호 연결 섹션을 참조하세요.
- 관찰성이 향상된 Container Insights 또는 Container Insights를 켭니다.

  CloudWatch Container Insights는 컨테이너 애플리케이션 및 마이크로서비스의 지표 및 로그를 수집하고, 종합하며, 요약합니다. 또한 Container Insights는 컨테이너 재시작 오류 등의 진단 정보를 제공합니다. 이 정보를 사용하여 문제를 격리하고 신속하게 해결할 수 있습니다. 자세한 내용은 the

section called “관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링” 섹션을 참조하세요.

2024년 12월 2일 AWS에서는 Amazon ECS의 향상된 관찰성을 갖춘 Container Insights를 출시했습니다. 이 버전은 Amazon EC2 및 Fargate 시작 유형을 사용하여 Amazon ECS 클러스터의 관찰성을 향상할 수 있도록 지원합니다. Amazon ECS에서 향상된 관찰성으로 Container Insights를 구성하면, Container Insights는 클러스터 수준에서 환경의 컨테이너 수준까지 상세한 인프라 원격 측정을 자동으로 수집하고, 대시보드에서 다양한 지표와 차원을 보여주는 데이터를 표시합니다. 그런 다음 Container Insights 콘솔에서 이러한 기본 대시보드를 사용하여 컨테이너 상태와 성능을 더 잘 이해하고 이상 징후를 식별하여 문제를 더 빠르게 완화할 수 있습니다.

Container Insights 대신 관찰성이 향상된 Container Insights를 사용하는 것이 좋습니다. 컨테이너 환경에서 세부적인 가시성을 제공하여 평균 해결 시간을 단축하기 때문입니다.

- 관리형 스토리지에 AWS KMS 키를 할당합니다. 키 생성 방법에 대한 자세한 내용은 AWS Key Management Service 사용 설명서의 KMS 키 생성을 참조하세요.
- Fargate 임시 스토리지에 AWS KMS 키를 할당합니다. 키 생성 방법에 대한 자세한 내용은 AWS Key Management Service 사용 설명서의 KMS 키 생성을 참조하세요.
- 클러스터를 식별하는 데 도움이 되는 태그를 추가합니다.

Auto Scaling 그룹 옵션

Amazon EC2 인스턴스를 사용할 때 태스크와 서비스가 실행되는 인프라를 관리하려면 Auto Scaling 그룹을 지정해야 합니다.

새 Auto Scaling 그룹을 생성하도록 선택하면 다음 동작에 대해 자동으로 구성됩니다.

- Amazon ECS에서 Auto Scaling 그룹의 축소 및 확장 작업을 관리합니다.
- Amazon ECS는 태스크를 포함하고 Auto Scaling 그룹에 있는 Amazon EC2 인스턴스가 축소 작업 중에 종료되는 것을 방지하지 않습니다. 자세한 정보는 AWS Auto Scaling 사용 설명서의 인스턴스 보호를 참조하세요.

그룹에 대해 시작할 인스턴스의 유형과 수를 결정하는 다음 Auto Scaling 그룹 속성을 구성합니다.

- Amazon ECS 최적화 AMI
- 인스턴스 유형.
- 인스턴스에 연결할 때 자격 증명을 증명하는 SSH 키 페어입니다. SSH 키를 생성하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 키 페어 및 Linux 인스턴스를 참조하세요.

- Auto Scaling 그룹에 대해 시작할 최소 인스턴스 수
- Auto Scaling 그룹에 대해 시작되는 최대 인스턴스 수입니다.

  그룹을 확장하려면 최댓값이 0보다 커야 합니다.

Amazon ECS는 AWS CloudFormation 스택의 일부로 사용자를 대신하여 Amazon EC2 Auto Scaling 시작 템플릿과 Auto Scaling 그룹을 생성합니다. AMI, 인스턴스 유형, SSH 키 페어에 대해 지정한 값은 시작 템플릿에 있습니다. 쉽게 식별할 수 있도록 템플릿에 접두사 `EC2ContainerService-<ClusterName>`이 붙습니다. Auto Scaling 그룹의 접두사는 `<ClusterName>-ECS-Infra-ECSAutoScalingGroup`입니다.

Auto Scaling 그룹에 대해 시작된 인스턴스는 시작 템플릿을 사용합니다.

네트워킹 옵션

기본적으로 인스턴스는 리전의 기본 서브넷으로 시작됩니다. 컨테이너 인스턴스에 대한 트래픽을 제어하고 현재 서브넷과 연결되어 있는 보안 그룹이 사용됩니다. 인스턴스에 대한 서브넷과 보안 그룹을 변경할 수 있습니다.

기존 서브넷을 선택할 수 있습니다. 기존 보안 그룹을 사용하거나 새 보안 그룹을 생성할 수 있습니다. 새 보안 그룹을 생성할 때는 인바운드 규칙을 하나 이상 지정해야 합니다.

인바운드 규칙은 컨테이너 인스턴스에 도달할 수 있는 트래픽을 결정하며 다음 속성을 포함합니다.

- 허용할 프로토콜
- 허용할 포트의 범위
- 인바운드 트래픽(소스)

특정 주소 또는 CIDR 블록의 인바운드 트래픽을 허용하려면 허용되는 CIDR과 함께 소스에 사용자 지정을 사용합니다.

모든 대상의 인바운드 트래픽을 허용하려면 소스에 위치 무관을 사용합니다. 그러면 0.0.0.0/0 IPv4 CIDR 블록 및 ::/0 IPv6 CIDR 블록이 자동으로 추가됩니다.

로컬 컴퓨터의 인바운드 트래픽을 허용하려면 소스에 소스 그룹을 사용합니다. 그러면 로컬 컴퓨터의 IP 주소가 허용되는 소스로 추가됩니다.

새 클러스터 생성(Amazon ECS 콘솔)

시작하기 전에 적절한 IAM 권한을 할당합니다. 자세한 내용은 the section called “Amazon ECS 클러스터 예제” 섹션을 참조하세요.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터(Clusters) 페이지에서 클러스터 생성(Create cluster)을 선택합니다.
5. 클러스터 구성에서 다음을 구성합니다.
   - 클러스터 이름에 고유한 이름을 입력합니다.

     이름은 최대 255자(대/소문자), 숫자 및 하이픈을 포함할 수 있습니다.
   - (선택 사항) 서비스 연결에 사용되는 네임스페이스를 클러스터 이름과 다르게 하려면 네임스페이스에 고유한 이름을 입력합니다.
6. Amazon EC2 인스턴스를 클러스터에 추가하고 인프라를 확장한 다음 Amazon EC2 인스턴스를 선택합니다.

   다음으로 용량 공급자 역할을 하는 Auto Scaling 그룹을 구성합니다.

   a. 기존 Auto Scaling 그룹을 사용하려면 Auto Scaling 그룹(ASG)(Auto Scaling group (ASG))에서 해당 그룹을 선택합니다.

   b. Auto Scaling 그룹을 생성하려면 Auto Scaling 그룹(ASG)(Auto Scaling group (ASG))에서 새 그룹 생성(Create new group)을 선택한 후 그룹에 대한 다음 세부 정보를 제공합니다.

      - 프로비저닝 모델에서 온디맨드 인스턴스 또는 스팟 인스턴스 중 사용할 인스턴스를 선택합니다.
      - 스팟 인스턴스를 사용하려는 경우 할당 전략에서 인스턴스에 사용할 스팟 용량 풀(인스턴스 유형, 가용 영역)을 선택합니다.

        대부분의 워크로드에서 가격 용량 최적화를 선택할 수 있습니다.

        자세한 내용은 Amazon EC2 사용 설명서의 스팟 인스턴스를 위한 할당 전략을 참조하세요.
      - 컨테이너 인스턴스 Amazon Machine Image(AMI)는 Auto Scaling 그룹 인스턴스에 대한 Amazon ECS 최적화 AMI를 선택합니다.
      - EC2 인스턴스 유형(EC2 instance type)에서 워크로드의 인스턴스 유형을 선택합니다.

관리형 확장은 Auto Scaling 그룹에서 동일하거나 유사한 인스턴스 유형을 사용해야 가장 효과적입니다.

- EC2 인스턴스 역할의 경우 기존 컨테이너 인스턴스 역할을 선택하거나 새로 생성할 수 있습니다.

  자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.
- 용량(Capacity)에 Auto Scaling 그룹에서 시작할 최소 인스턴스 수와 최대 인스턴스 수를 입력합니다.
- SSH 키 페어(SSH key pair)에서 인스턴스에 연결할 때 자격 증명을 증명하는 페어를 선택합니다.
- 더 큰 이미지와 스토리지를 허용하려면 루트 EBS 볼륨 크기에 값(GIB)을 입력합니다.

7. (선택 사항) VPC와 서브넷을 변경하려면 Amazon EC2 인스턴스에 대한 네트워킹에서 다음 작업 중 하나를 수행합니다.

   - 서브넷을 제거하려면서브넷(Subnets)에서 제거하려는 각 서브넷에 대해 X를 선택합니다.
   - 기본 VPC가 아닌 다른 VPC로 변경하려면 VPC에서 기존 VPC를 선택한 다음 서브넷에서 서브넷을 선택합니다.
   - 보안 그룹을 선택합니다. 보안 그룹에서 다음 옵션 중 하나를 선택합니다.
     - 기존 보안 그룹을 선택하려면 기존 보안 그룹 선택을 선택한 다음 보안 그룹을 선택합니다.
     - 보안 그룹을 생성하려면 새 보안 그룹 생성을 선택합니다. 그런 다음 각 인바운드 규칙에 대해 규칙 추가를 선택합니다.

       인바운드 규칙에 대한 자세한 내용은 네트워킹 옵션을 참조하세요.
   - Amazon EC2 컨테이너 인스턴스에 퍼블릭 IP 주소를 자동으로 할당하려면 퍼블릭 IP 자동 할당에서 다음 옵션 중 하나를 선택합니다.
     - 서브넷 설정 사용 - 인스턴스가 시작되는 서브넷이 퍼블릭 서브넷인 경우 인스턴스에 퍼블릭 IP 주소를 할당합니다.
     - 켜기 - 인스턴스에 퍼블릭 IP 주소를 할당합니다.

8. (선택 사항) Container Insights를 사용하고 모니터링을 확장한 후에 다음 옵션 중 하나를 선택합니다.

   - 권장되는 관찰성이 향상된 Container Insights를 사용하려면 관찰성이 향상된 Container Insights를 선택합니다.
   - Container Insights를 사용하려면 Container Insights를 선택합니다.

9. (선택 사항)

   수동 옵션으로 Runtime Monitoring을 사용하고 GuardDuty에서 이 클러스터를 모니터링하도록 하려면 태그 추가를 선택하고 다음을 수행합니다.

   - 키에 **guardDutyRuntimeMonitoringManaged**를 입력합니다.
   - 값에 **true**를 입력합니다.
10. (선택 사항) 관리형 스토리지의 데이터를 암호화합니다. 암호화에서 관리형 스토리지에 관리형 스토리지 데이터를 암호화하는 데 사용할 AWS KMS 키의 ARN을 입력합니다.
11. (선택 사항) 클러스터 태그를 관리하려면 태그(Tags)를 확장하고 다음 작업 중 하나를 수행합니다.

    [태그 추가] 태그 추가(Add tag)를 선택하고 다음을 수행합니다.

    - 키(Key)에 키 이름을 입력합니다.
    - 값에 키 값을 입력합니다.

    [태그 제거] 태그의 키와 값 오른쪽에 있는 제거를 선택합니다.
12. 생성(Create)을 선택합니다.

다음 단계

클러스터를 생성한 후 애플리케이션에 대한 작업 정의를 생성한 다음 독립 실행형 작업으로 실행하거나 서비스의 일부로 실행할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- Amazon ECS 작업 정의
- 애플리케이션을 Amazon ECS 태스크로 실행
- Amazon ECS 롤링 업데이트 배포 생성

클러스터 Auto Scaling을 통해 Amazon ECS 용량 자동 관리

Amazon ECS는 클러스터에 등록된 Amazon EC2 인스턴스의 크기 조정을 관리할 수 있습니다. 이를 Amazon ECS 클러스터 Auto Scaling이라고 합니다. Amazon ECS Auto Scaling 그룹 용량 공급자를 생성할 때 Managed Scaling을 켭니다. 그런 다음 이 Auto Scaling 그룹에서 인스턴스 사용률에 대한 목표 백분율(`targetCapacity`)을 설정합니다. Amazon ECS는 Auto Scaling 그룹에 대한 두 가지 사용자 지정 CloudWatch 지표와 대상 추적 조정 정책을 생성합니다. 그런 다음 Amazon ECS는 태스크에서 사용하는 리소스 사용률에 따라 스케일 인 및 스케일 아웃 작업을 관리합니다.

클러스터와 연결된 각 Auto Scaling 그룹 용량 공급자에 대해 Amazon ECS는 다음 리소스를 생성하고 관리합니다.

- 지표 값이 낮은 CloudWatch 경보
- 지표 값이 높은 CloudWatch 경보
- 대상 추적 조정 정책입니다.

> **Note**
>
> Amazon ECS에서 대상 추적 조정 정책을 생성하여 Auto Scaling 그룹에 연결합니다. 대상 추적 조정 정책을 업데이트하려면 조정 정책을 직접 업데이트하는 대신 용량 공급자 관리형 조정 설정을 업데이트해야 합니다.

관리형 조정을 끄거나 클러스터에서 용량 공급자 연결을 해제하면 Amazon ECS가 CloudWatch 지표와 대상 추적 조정 정책 리소스를 제거합니다.

Amazon ECS는 다음 지표를 사용하여 수행할 작업을 결정합니다.

`CapacityProviderReservation`

특정 용량 공급자에 대해 사용 중인 클러스터 컨테이너 인스턴스의 백분율. Amazon ECS는 이 지표를 생성합니다.

Amazon ECS는 `CapacityProviderReservation` 값을 0에서 100 사이의 숫자로 설정합니다. Amazon ECS는 다음 수식을 사용하여 Auto Scaling 그룹에 남아 있는 용량의 비율을 나타냅니다. Amazon ECS는 CloudWatch에 지표를 게시합니다. 지표를 계산하는 방법에 대한 자세한 내용은 Deep Dive on Amazon ECS Cluster Auto Scaling을 참조하세요.

```
CapacityProviderReservation = (number of instances needed) / (number of running
 instances) x 100
```

`DesiredCapacity`

Auto Scaling 그룹에 대한 용량의 양입니다. 이 지표는 CloudWatch에 게시되지 않습니다.

Amazon ECS는 `CapacityProviderReservation` 지표를 CloudWatch에 `AWS/ECS/ManagedScaling` 네임스페이스로 게시합니다. `CapacityProviderReservation` 지표를 사용하면 다음 작업 중 하나가 발생합니다.

`CapacityProviderReservation` 값이 `targetCapacity`와 같음

Auto Scaling 그룹을 스케일 인 또는 스케일 아웃할 필요가 없습니다. 목표 사용률에 도달했습니다.

`CapacityProviderReservation` 값이 `targetCapacity`보다 큼

`targetCapacity` 백분율보다 높은 비율의 용량을 사용하는 작업이 많습니다. `CapacityProviderReservation` 지표 값이 증가하면 관련 CloudWatch 경보가 작동합니다. 이 경보는 Auto Scaling 그룹에 대한 `DesiredCapacity` 값을 업데이트합니다. Auto Scaling 그룹은 이 값을 사용하여 EC2 인스턴스를 시작한 다음 클러스터에 등록합니다.

`targetCapacity`가 기본값인 100%인 경우 작업을 실행할 인스턴스에 사용할 수 있는 용량이 없기 때문에 스케일 아웃 중 새 작업은 `PENDING` 상태가 됩니다. 새 인스턴스가 ECS에 등록되면 이러한 작업이 새 인스턴스에서 시작됩니다.

`CapacityProviderReservation` 값이 `targetCapacity`보다 작음

`targetCapacity` 백분율보다 낮은 비율의 용량을 사용하는 작업이 적고 종료할 수 있는 인스턴스가 하나 이상 있습니다. `CapacityProviderReservation` 지표 값이 감소하면 관련 CloudWatch 경보가 작동합니다. 이 경보는 Auto Scaling 그룹에 대한 `DesiredCapacity` 값을 업데이트합니다. Auto Scaling 그룹은 이 값을 사용하여 EC2 컨테이너 인스턴스를 종료한 다음 클러스터에서 등록을 취소합니다.

Auto Scaling 그룹은 그룹 종료 정책에 따라 스케일 인 이벤트 중 처음 종료할 인스턴스를 결정합니다. 또한 인스턴스 스케일 인 보호 설정이 켜진 인스턴스는 피합니다. 관리형 종료 보호를 켜는 경우 클러스터 Auto Scaling에서 인스턴스 스케일 인 보호 설정이 지정된 인스턴스를 관리할 수 있습니다. 관리형 종료 보호에 대한 자세한 내용은 Amazon ECS가 종료하는 인스턴스 제어을 참조하세요. Auto Scaling 그룹에서 인스턴스를 종료하는 방법에 대한 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 Control which Auto Scaling instances terminate during scale in을 참조하세요.

클러스터 Auto Scaling을 사용할 때는 다음 사항을 고려해야 합니다.

- Amazon ECS가 관리하는 것 이외의 조정 정책이 있는 용량 공급자와 연결된 Auto Scaling 그룹에 대해 원하는 용량을 변경하거나 관리하지 마세요.
- Amazon ECS가 인스턴스 0개에서 스케일 아웃되면 인스턴스 2개가 자동으로 시작됩니다.
- Amazon ECS는 사용자를 대신하여 다른 AWS Auto Scaling를 호출하는 데 필요한 권한에 대해 `AWSServiceRoleForECS` 서비스 연결 IAM 역할을 사용합니다. 자세한 내용은 Amazon ECS에 대해 서비스 연결 역할 사용 섹션을 참조하세요.

- Auto Scaling 그룹에 용량 공급자를 사용하는 경우 용량 공급자를 생성하는 사용자, 그룹 또는 역할에는 `autoscaling:CreateOrUpdateTags` 권한이 필요합니다. 이는 용량 공급자와 연결할 때 Amazon ECS가 Auto Scaling 그룹에 태그를 추가하기 때문입니다.

  > **Important**
  >
  > 사용하는 도구가 Auto Scaling 그룹에서 `AmazonECSManaged` 태그를 제거하지 않는지 확인합니다. 이 태그가 제거되면 Amazon ECS에서는 규모 조정을 관리할 수 없습니다.

- 클러스터 Auto Scaling은 그룹에 대해 MinimumCapacity 또는 MaximumCapacity를 수정하지 않습니다. 그룹을 스케일 아웃하려면 MaximumCapacity 값이 0보다 커야 합니다.
- Auto Scaling(관리형 조정)이 켜져 있을 때 한 번에 하나의 클러스터에만 용량 공급자를 연결할 수 있습니다. 용량 공급자가 관리형 크기 조정을 해제한 경우 이를 여러 클러스터와 연결할 수 있습니다.
- 관리형 조정이 꺼져 있으면 용량 공급자가 스케일 인 또는 스케일 아웃하지 않습니다. 용량 공급자 전략을 사용하여 용량 공급자 간에 작업의 균형을 조정할 수 있습니다.
- 이 `binpack` 전략은 용량 측면에서 가장 효율적인 전략입니다.
- 목표 용량이 100% 미만인 경우 배치 전략은 `spread` 전략보다 순위가 더 높을 수밖에 없는 `binpack` 전략을 사용해야 합니다. 이렇게 하면 각 작업에 전용 인스턴스가 있거나 한도에 도달할 때까지 용량 공급자가 스케일 아웃할 수 없습니다.

클러스터 Auto Scaling 켜기

콘솔 또는 AWS CLI를 사용하여 클러스터 오토 스케일링을 활성화할 수 있습니다.

콘솔을 사용하여 EC2 시작 유형에 대한 클러스터를 생성할 때 Amazon ECS에서는 사용자를 대신하여 Auto Scaling 그룹을 생성하고 대상 용량을 설정합니다. 자세한 내용은 Amazon EC2 시작 유형에 대한 Amazon ECS 클러스터 생성 섹션을 참조하세요.

Auto Scaling 그룹을 생성한 다음에 클러스터에 할당할 수도 있습니다. 자세한 내용은 Amazon ECS 용량 공급자 업데이트 섹션을 참조하세요.

AWS CLI를 사용하는 경우 클러스터 생성 후

1. 용량 공급자를 생성하기 전에 Auto Scaling 그룹을 생성해야 합니다. 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 오토 스케일링을 참조하세요.
2. `put-cluster-capacity-providers`를 사용하여 클러스터 용량 공급자를 수정합니다. 자세한 내용은 Amazon ECS 클러스터 Auto Scaling 켜기 섹션을 참조하세요.

Amazon ECS 클러스터 Auto Scaling 최적화

Amazon EC2에서 Amazon ECS를 실행하는 고객은 클러스터 Auto Scaling을 활용하여 Amazon EC2 Auto Scaling 그룹의 규모 조정을 관리할 수 있습니다. 클러스터 Auto Scaling을 사용하면 Auto Scaling 그룹의 규모를 자동으로 조정하고 태스크 실행에만 집중하도록 Amazon ECS를 구성할 수 있습니다. Amazon ECS는 추가 개입 없이 Auto Scaling 그룹이 필요에 따라 스케일 아웃 및 스케일 인하도록 보장합니다. Amazon ECS 용량 공급자는 애플리케이션의 요구 사항을 충족하기에 충분한 컨테이너 인스턴스가 있는지 확인하여 클러스터의 인프라를 관리하는 데 사용됩니다. 클러스터 Auto Scaling이 내부적으로 어떻게 작동하는지 알아보려면 Amazon ECS 클러스터 Auto Scaling에 대한 심층 분석을 참조하세요.

클러스터 Auto Scaling은 클러스터 용량 조정을 위한 Auto Scaling 그룹과의 CloudWatch 기반 통합에 의존합니다. 따라서

- CloudWatch 지표 게시와 관련된 고유한 지연 시간,
- 지표 `CapacityProviderReservation`이 CloudWatch 경보(높음 및 낮음 모두)를 위반하는 데 걸리는 시간,
- 새로 시작한 Amazon EC2 인스턴스가 워밍업하는 데 걸리는 시간이 있습니다. 더 빠른 배포를 위해 클러스터 Auto Scaling의 응답성을 높이려면 다음 작업을 수행할 수 있습니다.

용량 제공자 단계 규모 조정 크기

Amazon ECS 용량 공급자는 애플리케이션의 요구 사항을 충족하기 위해 컨테이너 인스턴스를 늘리거나 줄일 것입니다. Amazon ECS가 시작할 최소 인스턴스 수는 기본적으로 1로 설정됩니다. 보류 중인 작업을 배치하는 데 여러 인스턴스가 필요한 경우 배포 시간이 추가될 수 있습니다. Amazon ECS API를 통해 `minimumScalingStepSize`를 늘려 Amazon ECS가 한 번에 스케일 인하거나 스케일 아웃하는 최소 인스턴스 수를 늘릴 수 있습니다. `maximumScalingStepSize`가 너무 낮으면 한 번에 스케일 인 또는 스케일 아웃하는 컨테이너 인스턴스 수를 제한하여 배포 속도가 느려질 수 있습니다.

> **Note**
>
> 이 구성은 현재 `CreateCapacityProvider` 또는 `UpdateCapacityProvider` API를 통해서만 사용할 수 있습니다.

인스턴스 워밍업 기간

인스턴스 워밍업 기간은 새로 시작된 Amazon EC2 인스턴스가 Auto Scaling 그룹의 CloudWatch 지표에 기여할 수 있는 기간입니다. 지정된 워밍업 기간이 만료되면 인스턴스는 Auto Scaling 그룹의 집계된 지표로 계산되고 클러스터 Auto Scaling은 다음 계산 반복을 진행하여 필요한 인스턴스 수를 추정합니다.

`instanceWarmupPeriod`의 기본값은 300초이며 보다 반응성이 뛰어난 조정을 위해 `CreateCapacityProvider` 또는 `UpdateCapacityProvider` API를 통해 더 낮은 값으로 구성할 수 있습니다. 과다 프로비저닝을 방지하려면 값을 60초 이상으로 설정하는 것이 좋습니다.

예비 용량

용량 공급자가 작업 배치에 사용할 수 있는 컨테이너 인스턴스가 없는 경우 Amazon EC2 인스턴스를 즉시 시작하여 클러스터 용량을 늘리고(스케일 아웃) 컨테이너를 시작하기 전에 해당 인스턴스가 부팅될 때까지 기다려야 합니다. 이로 인해 작업 시작 속도가 크게 느려질 수 있습니다. 여기에서는 두 가지 옵션이 있습니다.

이 경우 예비 Amazon EC2 용량이 이미 시작되어 태스크를 실행할 준비가 되어 있으면 효과적인 작업 시작 속도가 빨라집니다. `Target Capacity` 구성을 사용하여 클러스터의 예비 용량을 유지하도록 지정할 수 있습니다. 예를 들어 `Target Capacity`를 80%로 설정하면 클러스터에 항상 20%의 예비 용량이 필요함을 나타냅니다. 이 여유 용량을 통해 모든 독립 실행형 태스크를 즉시 시작할 수 있으므로 태스크 시작이 제한되지 않습니다. 이 접근 방식의 단점은 예비 클러스터 용량을 유지하는 데 드는 비용이 증가할 수 있다는 것입니다.

고려할 수 있는 또 다른 접근 방식은 용량 공급자가 아닌 서비스에 헤드룸을 추가하는 것입니다. 즉, 예비 용량을 시작하기 위한 `Target Capacity` 구성을 줄이는 대신 대상 추적 조정 지표 또는 서비스 Auto Scaling의 단계 규모 조정 임계값을 수정하여 서비스의 복제본 수를 늘릴 수 있습니다. 이 접근 방식은 워크로드가 급증하는 경우에만 유용하며 새 서비스를 배포하거나 처음으로 0개에서 N개의 태스크로 전환하는 경우에는 효과가 없다는 점에 유의하세요. 관련 규모 조정 정책에 대한 자세한 내용은 Amazon Elastic Container Service 개발자 안내서에서 대상 추적 조정 정책 또는 단계 조정 정책을 참조하세요.

Amazon ECS 관리형 규모 조정 동작

관리형 규모 조정을 사용하는 Auto Scaling 그룹 용량 공급자가 있는 경우 Amazon ECS에서는 클러스터에 추가할 인스턴스의 최적 개수를 예측하고 이 값을 사용하여 요청하거나 릴리스할 인스턴스 수를 결정합니다.

관리형 확장 동작

Amazon ECS는 서비스, 독립 실행형 태스크 또는 클러스터 기본값의 용량 공급자 전략에 따라 각 태스크에 대한 용량 공급자를 선택합니다. 단일 용량 공급자의 경우 Amazon ECS는 이러한 단계의 나머지 내용을 따릅니다.

용량 공급자 전략이 없는 작업은 용량 공급자가 무시합니다. 용량 공급자 전략이 없는 보류 중인 작업의 경우 용량 공급자가 스케일 아웃하지 않습니다. 시작 유형을 설정하는 작업 또는 서비스에서는 용량 공급자 전략을 설정할 수 없습니다.

관리형 확장에 대한 자세한 설명은 아래와 같습니다.

- 각 그룹에 똑같은 리소스 요구 사항이 적용되도록 이 용량 공급자의 모든 프로비저닝 작업을 그룹화합니다.
- Auto Scaling 그룹에서 여러 인스턴스 유형을 사용하는 경우 Auto Scaling 그룹의 인스턴스 유형은 파라미터를 기준으로 정렬됩니다. 이러한 파라미터에는 vCPU, 메모리, 탄력적 네트워크 인터페이스(ENI), 포트, GPU 등이 있습니다. 각 파라미터의 가장 작은 인스턴스 유형과 가장 큰 인스턴스 유형이 선택됩니다. 인스턴스 유형을 선택하는 방법에 대한 자세한 내용은 Amazon ECS에 대한 Amazon EC2 컨테이너 인스턴스 섹션을 참조하세요.

  > **Important**
  >
  > 작업 그룹의 리소스 요구 사항이 Auto Scaling 그룹의 가장 작은 인스턴스 유형보다 큰 경우 해당 작업 그룹은 이 용량 공급자로 실행할 수 없습니다. 용량 공급자는 Auto Scaling 그룹의 규모를 조정하지 않습니다. 작업은 `PROVISIONING` 상태로 유지됩니다.
  > 작업이 `PROVISIONING` 상태로 유지되지 않도록 하려면 여러 최소 리소스 요구 사항에 대해 Auto Scaling 그룹과 용량 공급자를 별도로 생성하는 것이 좋습니다. 작업을 실행하거나 서비스를 생성할 때 Auto Scaling 그룹의 가장 작은 인스턴스 유형에서 작업을 실행할 수 있는 용량 공급자 전략에만 용량 공급자를 추가하세요. 다른 파라미터에는 배치 제약 조건을 사용할 수 있습니다.

- 각 작업 그룹에 대해 Amazon ECS는 배치되지 않은 작업을 실행하는 데 필요한 인스턴스 개수를 계산합니다. 이 계산에서는 `binpack` 전략을 사용합니다. 이 전략은 작업의 vCPU, 메모리, 탄력적 네트워크 인터페이스(ENI), 포트, GPU 요구 사항을 고려합니다. 또한 Amazon EC2 인스턴스의 리소스 가용성을 고려합니다. 가장 큰 인스턴스 유형의 값은 계산된 최대 인스턴스 개수로 처리됩니다. 가장 작은 인스턴스 유형에 대한 값은 보호로 사용됩니다. 가장 작은 인스턴스 유형이 작업의 인스턴스를 하나 이상 실행할 수 없는 경우 계산에서 작업을 호환되지 않는 것으로 간주합니다. 따라서 스케일 아웃 계산에서 작업이 제외됩니다. 모든 작업이 가장 작은 인스턴스 유형과 호환되지 않으면 클러

스터 Auto Scaling이 중지되고 `CapacityProviderReservation` 값은 `targetCapacity` 값으로 유지됩니다.

- Amazon ECS는 다음 중 하나에 해당하는 경우 `minimumScalingStepSize`를 기준으로 `CapacityProviderReservation` 지표를 CloudWatch에 게시합니다.
  - 계산된 최대 인스턴스 수가 최소 조정 단계 크기보다 작습니다.
  - `maximumScalingStepSize` 또는 계산된 최대 인스턴스 수 중 낮은 값입니다.
- CloudWatch 경보는 용량 공급자에 `CapacityProviderReservation` 지표를 사용합니다. `CapacityProviderReservation` 지표가 `targetCapacity` 값보다 크면 경보로 인해 Auto Scaling 그룹의 `DesiredCapacity`가 증가합니다. `targetCapacity` 값은 클러스터 Auto Scaling 활성화 단계에서 CloudWatch 경보로 전송되는 용량 공급자 설정입니다.

  `targetCapacity`의 기본값은 100%입니다.
- Auto Scaling 그룹은 추가 EC2 인스턴스를 시작합니다. 오버프로비저닝을 방지하기 위해, Auto Scaling은 최근에 시작된 EC2 인스턴스 용량이 새 인스턴스를 시작하기 전에 안정화되었는지 확인합니다. Auto Scaling이 모든 기존 인스턴스가 `instanceWarmupPeriod`를 통과했는지 검사합니다(현재에서 인스턴스 시작 시간을 뺀 값). `instanceWarmupPeriod` 내에 있는 인스턴스에 대해 스케일 아웃이 차단됩니다.

  새로 시작된 인스턴스의 기본 워밍업 시간은 300초입니다.

자세한 내용은 Deep dive on Amazon ECS cluster auto scaling(Amazon ECS 클러스터 Auto Scaling 심층 분석)을 참조하세요.

확장 고려 사항

확장 프로세스에 대해 다음 사항을 고려하세요.

- 여러 배치 제약이 있지만 `distinctInstance` 작업 배치 제약만 사용하는 것을 권장합니다. 이는 샘플링된 인스턴스와 호환되지 않는 배치 제약 조건을 사용하기 때문에 스케일 아웃 프로세스가 중지되지 않습니다.
- 관리형 확장은 Auto Scaling 그룹에서 동일하거나 유사한 인스턴스 유형을 사용해야 가장 효과적입니다.
- 스케일 아웃 프로세스가 필요하고 현재 실행 중인 컨테이너 인스턴스가 없으면 Amazon ECS는 항상 처음에는 인스턴스 2개로 스케일 아웃한 다음 추가 스케일 아웃 또는 스케일 인 프로세스를 수행합니다. 인스턴스 워밍업 기간에는 추가 스케일 아웃이 대기합니다. 스케일 인 프로세스의 경우 Amazon ECS는 항상 스케일 아웃 프로세스 후 15분을 기다린 다음 스케일 인 프로세스를 시작합니다.

- 두 번째 스케일 아웃 단계는 전체 스케일링 제한에 영향을 미칠 수 있는 `instanceWarmupPeriod`가 만료될 때까지 기다려야 합니다. 이 시간을 줄여야 하는 경우 EC2 인스턴스가 Amazon ECS 에이전트(오버프로비저닝 방지)를 시작하고 시작할 수 있을 만큼 `instanceWarmupPeriod`가 충분히 커야 합니다.
- 클러스터 Auto Scaling은 용량 공급자 Auto Scaling에서 시작 구성, 시작 템플릿 및 여러 인스턴스 유형을 지원합니다. 또한 여러 인스턴스 유형 없이 속성 기반 인스턴스 유형 선택을 사용할 수도 있습니다.
- Auto Scaling 그룹과 온디맨드 인스턴스, 여러 인스턴스 유형 또는 스팟 인스턴스를 사용할 경우 우선순위 목록에서 용량이 큰 인스턴스 유형을 높은 순위에 올리고 가중치를 지정하지 않습니다. 현재는 가중치 지정을 지원하지 않습니다. 자세한 정보는 AWS Auto Scaling 사용 설명서의 여러 인스턴스 유형을 제공하는 Auto Scaling 그룹을 참조하세요.
- 그러면 Amazon ECS가 계산된 인스턴스 최댓값이 최소 조정 단계 크기보다 작을 경우 `minimumScalingStepSize`를 시작하고, 그렇지 않으면 `maximumScalingStepSize` 또는 계산된 인스턴스 개수의 최댓값 중 작은 개수를 시작합니다.
- Amazon ECS 서비스 또는 `run-task`가 작업을 시작하고 용량 공급자 컨테이너 인스턴스에 작업을 시작하기에 충분한 리소스가 없는 경우 Amazon ECS는 각 클러스터에 대해 이 상태의 작업 수를 제한하고 이 제한을 초과하여 작업이 실행되지 않도록 합니다. 자세한 내용은 Service quotas 섹션을 참조하세요.

관리형 축소 동작

Amazon ECS는 클러스터 내의 각 용량 공급자에 대한 컨테이너 인스턴스를 모니터링합니다. 컨테이너 인스턴스에 실행 중인 작업이 없으면 컨테이너 인스턴스는 비어 있는 것으로 간주되고 Amazon ECS는 스케일 인 프로세스를 시작합니다.

CloudWatch 축소 경보에는 Auto Scaling 그룹의 축소 프로세스가 시작되기 전에 15데이터 포인트(15분)가 필요합니다. Amazon ECS가 등록된 컨테이너 인스턴스의 수를 줄여야 할 때까지 축소 프로세스가 시작된 후, Auto Scaling 그룹은 `DesireCapacity` 값을 인스턴스 1개보다 크고 매분 50% 미만으로 설정합니다.

스케일 인 프로세스가 진행 중인 동안 Amazon ECS가 스케일 아웃을 요청하는 경우 (`CapacityProviderReservation`이 100보다 큼) 스케일 인 프로세스가 중지되고 필요한 경우 처음부터 시작됩니다.

다음은 스케일 인 동작에 대해 자세히 설명합니다.

1. Amazon ECS는 비어 있는 컨테이너 인스턴스의 수를 계산합니다. 대몬(daemon) 작업이 실행 중인 경우에도 컨테이너 인스턴스는 비어 있는 것으로 간주됩니다.
2. Amazon ECS는 다음 수식을 사용하여 Auto Scaling 그룹의 실제 크기 대비 필요한 크기의 비율을 백분율로 나타내는 0~100 사이의 숫자로 `CapacityProviderReservation` 값을 설정합니다. Amazon ECS는 CloudWatch에 지표를 게시합니다. 지표를 계산하는 방법에 대한 자세한 내용은 Deep Dive on Amazon ECS Cluster Auto Scaling을 참조하세요.

```
CapacityProviderReservation = (number of instances needed) / (number of running
 instances) x 100
```

3. `CapacityProviderReservation` 지표는 CloudWatch 경보를 생성합니다. 이 경보는 Auto Scaling 그룹에 대한 `DesiredCapacity` 값을 업데이트합니다. 그러면 다음 작업 중 하나가 수행됩니다.
   - 용량 공급자 관리형 종료를 사용하지 않는 경우 Auto Scaling 그룹은 Auto Scaling 그룹 종료 정책을 사용하여 EC2 인스턴스를 선택하고 EC2 인스턴스 수가 `DesiredCapacity`에 도달할 때까지 인스턴스를 종료합니다. 그런 다음 컨테이너 인스턴스가 클러스터에서 등록 취소됩니다.
   - 모든 컨테이너 인스턴스가 관리형 종료 보호를 사용하는 경우 Amazon ECS는 비어 있는 컨테이너 인스턴스에서 스케일 인 보호를 제거합니다. 그러면 Auto Scaling 그룹에서 EC2 인스턴스를 종료할 수 있습니다. 그런 다음 컨테이너 인스턴스가 클러스터에서 등록 취소됩니다.

Amazon ECS가 종료하는 인스턴스 제어

> **Important**
>
> 클러스터 Auto Scaling의 관리형 종료 보호 기능을 사용하려면 Auto Scaling 그룹에서 Auto Scaling 인스턴스 스케일 인 보호를 켜야 합니다.

관리형 종료 보호 기능을 사용하면 클러스터 Auto Scaling을 통해 종료되는 인스턴스를 제어할 수 있습니다. 관리형 종료 보호 기능을 사용한 경우 Amazon ECS는 실행 중인 Amazon ECS 태스크가 없는 EC2 인스턴스만 종료합니다. `DAEMON` 일정 전략을 사용하는 서비스에서 실행하는 작업은 무시되며 인스턴스에서 이러한 작업을 실행 중인 경우에도 클러스터 Auto Scaling에서 인스턴스를 종료될 수 있습니다. 클러스터의 모든 인스턴스가 이러한 작업을 실행하고 있기 때문입니다.

Amazon ECS는 먼저 Auto Scaling 그룹에서 EC2 인스턴스에 대한 인스턴스 스케일 인 보호 옵션을 켭니다. 그런 다음 Amazon ECS는 인스턴스에 작업을 배치합니다. 인스턴스에서 모든 대몬이 아닌 작업

이 중지되면 Amazon ECS는 축소 프로세스를 시작하고 EC2 인스턴스에 대한 축소 보호를 해제합니다. 그러면 Auto Scaling 그룹에서 인스턴스를 종료할 수 있습니다.

Auto Scaling 인스턴스 스케일 인 보호는 Auto Scaling에서 종료할 수 있는 EC2 인스턴스를 제어합니다. 스케일 인 기능이 켜진 인스턴스는 스케일 인 프로세스 중에 종료할 수 없습니다. Auto Scaling 인스턴스 스케일 인 보호에 대한 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 Using instance scale-in protection을 참조하세요.

`targetCapacity` 비율을 설정하여 여유 용량을 확보할 수 있습니다. 이러한 방식으로 Auto Scaling 그룹이 더 많은 인스턴스를 시작할 때까지 기다리지 않고 향후 작업을 더 빠르게 시작할 수 있습니다. Amazon ECS는 목표 용량 값을 사용하여 서비스가 생성하는 CloudWatch 지표를 관리합니다. Amazon ECS는 CloudWatch 지표를 관리합니다. Auto Scaling 그룹은 안정 상태로 처리되어 규모 조정 작업이 필요하지 않습니다. 값은 0~100%가 될 수 있습니다. 예를 들어, Amazon ECS 작업에서 사용하는 용량 외에 10%의 여유 용량을 유지하도록 Amazon ECS를 구성하려면 대상 용량 값을 90%로 설정합니다. 용량 공급자에 대한 `targetCapacity` 값을 설정할 때는 다음 사항을 고려해야 합니다.

- 100% 미만의 `targetCapacity` 값은 클러스터에 있어야 하는 사용 가능한 용량(Amazon EC2 인스턴스)의 양을 나타냅니다. 여유 용량은 실행 중인 작업이 없음을 의미합니다.
- 추가 `binpack`이 없는 가용 영역과 같은 배치 제약 조건은 Amazon ECS가 최종적으로 인스턴스당 하나의 작업을 실행하도록 하며, 이는 바람직한 동작이 아닐 수 있습니다.

관리형 종료 보호를 사용하려면 Auto Scaling 그룹에서 Auto Scaling 인스턴스 스케일 인 보호를 켜야 합니다. 스케일 인 보호를 켜지 않고 관리형 종료 보호를 켜는 경우 원치 않는 동작이 발생할 수 있습니다. 예를 들어 인스턴스가 드레이닝 상태에서 멈출 수 있습니다. 자세한 정보는 Amazon EC2 Auto Scaling 사용 설명서의 인스턴스 스케일 인 방지 사용을 참조하세요.

용량 공급자와 함께 종료 보호를 사용하는 경우에는 용량 공급자와 연결된 Auto Scaling 그룹에서 인스턴스 분리와 같은 수동 작업을 수행하지 마세요. 수동 작업으로 인해 용량 공급자의 스케일 인 작업이 중단될 수 있습니다. Auto Scaling 그룹에서 인스턴스를 분리하는 경우 Amazon ECS 클러스터에서 분리된 인스턴스를 등록 취소해야 합니다.

Amazon ECS 용량 공급자의 관리형 종료 방지 기능 업데이트

관리형 종료 방지 기능을 사용하는 경우 기존 용량 공급자의 설정을 업데이트해야 합니다.

콘솔

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.

3. 클러스터 페이지에서 인프라 탭을 선택하세요.
4. 용량 공급자를 선택하세요.
5. 업데이트를 선택하여 용량 공급자 설정을 수정합니다.
6. Auto Scaling 그룹 설정에서 관리형 종료 방지 기능을 전환하여 기능을 활성화하거나 비활성화합니다.
7. 업데이트를 선택합니다.

AWS CLI

다음 `update-capacity-provider` 명령을 사용하여 용량 공급자의 관리형 종료 방지 기능 설정을 업데이트할 수 있습니다.

관리형 종료 방지 기능을 활성화하려면

```
aws ecs update-capacity-provider \
  --capacity-provider CapacityProviderName \
  --auto-scaling-group-provider
 "managedScaling={status=ENABLED,targetCapacity=70,minimumScalingStepSize=1,maximumScalingStepS
```

관리형 종료 방지 기능을 비활성화하려면

```
aws ecs update-capacity-provider \
  --capacity-provider CapacityProviderName \
  --auto-scaling-group-provider
 "managedScaling={status=ENABLED,targetCapacity=70,minimumScalingStepSize=1,maximumScalingStepS
```

> **Note**
>
> 클러스터 전체에 변경 사항이 적용되려면 몇 분 정도 걸릴 수 있습니다. 관리형 종료 방지 기능을 활성화하면 이미 태스크를 실행 중인 인스턴스가 스케일 인 이벤트로부터 보호됩니다. 관리형 종료 방지 기능을 비활성화하면 다음 ECS 용량 공급자 관리 주기 동안 인스턴스에서 보호 플래그가 제거됩니다.

태스크 실행을 위한 콘솔

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.

2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
3. 클러스터 페이지에서 태스크 탭을 선택하세요.
4. 작업을 선택합니다.
5. 구성에서 관리형 종료 방지 기능을 전환하여 기능을 활성화하거나 비활성화하세요.
6. 태스크 스케일 인 보호 구성을 선택하세요.

   태스크 스케일 인 보호 구성 대화 상자가 표시됩니다.

   a. 태스크 스케일 인 보호에서 켜기로 전환하세요.
   b. 만료(분)에 태스크 스케일 인 보호가 종료되기 전의 시간(분)을 입력하세요.
   c. Update(업데이트)를 선택합니다.

Amazon ECS 클러스터 Auto Scaling 켜기

클러스터 오토 스케일링을 사용 설정하면 Amazon ECS가 클러스터에 등록된 Amazon EC2 인스턴스의 규모 조정을 관리합니다.

콘솔을 사용하여 클러스터 오토 스케일링을 활성화하려면 Amazon ECS를 위한 용량 공급자 생성 단원을 참조하세요.

시작하기 전에 Auto Scaling 그룹과 용량 공급자를 생성합니다. 자세한 내용은 the section called “EC2 시작 유형을 위한 용량 공급자” 섹션을 참조하세요.

클러스터 Auto Scaling을 켜려면 용량 공급자를 클러스터와 연결한 다음 클러스터 Auto Scaling을 켭니다.

1. `put-cluster-capacity-providers` 명령을 사용하여 하나 이상의 용량 공급자를 클러스터와 연결합니다.

   AWS Fargate 용량 공급자를 추가하려면 요청에 `FARGATE` 및 `FARGATE_SPOT` 용량 공급자를 포함합니다. 자세한 정보는 AWS CLI 명령 참조의 `put-cluster-capacity-providers` 섹션을 참조하세요.

```
aws ecs put-cluster-capacity-providers \
  --cluster ClusterName \
  --capacity-providers CapacityProviderName FARGATE FARGATE_SPOT \
  --default-capacity-provider-strategy capacityProvider=CapacityProvider,weight=1
```

EC2 시작 유형에서 Auto Scaling 그룹을 추가하려면 요청에 Auto Scaling 그룹 이름을 포함합니다. 자세한 정보는 AWS CLI 명령 참조의 put-cluster-capacity-providers 섹션을 참조하세요.

```
aws ecs put-cluster-capacity-providers \
  --cluster ClusterName \
  --capacity-providers CapacityProviderName \
  --default-capacity-provider-strategy capacityProvider=CapacityProvider,weight=1
```

2. describe-clusters 명령을 사용하여 연결에 성공했는지 확인합니다. 자세한 정보는 AWS CLI 명령 참조의 describe-clusters 섹션을 참조하세요.

```
aws ecs describe-clusters \
  --cluster ClusterName \
  --include ATTACHMENTS
```

3. update-capacity-provider 명령을 사용하여 용량 공급자에 대해 관리형 Auto Scaling을 사용 설정합니다. 자세한 정보는 AWS CLI 명령 참조의 update-capacity-provider 섹션을 참조하세요.

```
aws ecs update-capacity-provider \
  --capacity-providers CapacityProviderName \
  --auto-scaling-group-provider managedScaling=ENABLED
```

Amazon ECS 클러스터 Auto Scaling 끄기

클러스터에 등록된 EC2 인스턴스를 보다 세밀하게 제어해야 하는 경우 클러스터 오토 스케일링을 끕니다.

클러스터에 대한 클러스터 Auto Scaling을 해제하려면 용량 제공자를 클러스터에서 설정한 Managed Scaling과의 연결을 해제하거나 용량 공급자를 업데이트하여 Managed Scaling을 해제할 수 있습니다.

용량 공급자 연결 해제

용량 공급자를 클러스터와 연결 해제하려면 다음 단계를 수행합니다.

1. put-cluster-capacity-providers 명령을 사용하여 Auto Scaling 그룹 용량 공급자를 클러스터와 연결 해제합니다. 클러스터는 AWS Fargate 용량 공급자와의 연결을 유지할 수 있습니다.

자세한 정보는 AWS CLI 명령 참조의 put-cluster-capacity-providers 섹션을 참조하세요.

```
aws ecs put-cluster-capacity-providers \
  --cluster ClusterName \
  --capacity-providers FARGATE FARGATE_SPOT \
  --default-capacity-provider-strategy '[]'
```

put-cluster-capacity-providers 명령을 사용하여 Auto Scaling 그룹 용량 공급자를 클러스터와 연결 해제합니다. 자세한 정보는 AWS CLI 명령 참조의 put-cluster-capacity-providers 섹션을 참조하세요.

```
aws ecs put-cluster-capacity-providers \
  --cluster ClusterName \
  --capacity-providers [] \
  --default-capacity-provider-strategy '[]'
```

2. describe-clusters 명령을 사용하여 연결 해제에 성공했는지 확인합니다. 자세한 정보는 AWS CLI 명령 참조의 describe-clusters 섹션을 참조하세요.

```
aws ecs describe-clusters \
  --cluster ClusterName \
  --include ATTACHMENTS
```

용량 공급자에 대해 관리형 조정 끄기

용량 공급자에 대해 관리형 조정을 해제하려면 다음 단계를 수행합니다.

- update-capacity-provider 명령을 사용하여 용량 공급자에 대해 관리형 Auto Scaling을 사용 해제합니다. 자세한 정보는 AWS CLI 명령 참조의 update-capacity-provider 섹션을 참조하세요.

```
aws ecs update-capacity-provider \
  --capacity-providers CapacityProviderName \
  --auto-scaling-group-provider managedScaling=DISABLED
```

Amazon ECS를 위한 용량 공급자 생성

클러스터 생성이 완료된 후 EC2 시작 유형에 대한 새 용량 공급자(Auto Scaling 그룹)를 생성할 수 있습니다. 용량 공급자는 애플리케이션의 인프라 관리 및 확장에 도움이 됩니다.

용량 공급자를 생성하기 전에 Auto Scaling 그룹을 생성해야 합니다. 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 오토 스케일링을 참조하세요.

클러스터에 대한 용량 공급자 생성(Amazon ECS 콘솔)

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
4. Cluster : ***name***(클러스터: 이름) 페이지에서 Infrastructure(인프라)를 선택한 다음 Create(생성)를 선택합니다.
5. Create capacity providers(용량 공급자 생성) 페이지에서 다음 옵션을 구성합니다.
   a. Basic details(기본 세부 정보)의 Capacity provider name(용량 공급자 이름)에 고유한 용량 공급자 이름을 입력합니다.
   b. Auto Scaling group(오토 스케일링) 아래에서 Use an existing Auto Scaling group(기존 오토 스케일링 사용)에 대해 오토 스케일링을 선택합니다.
   c. (선택 사항) 조정 정책을 구성하려면 Scaling policies(조정 정책)에서 다음 옵션을 구성합니다.
      - Amazon ECS에서 스케일 인 및 스케일 아웃 작업을 관리하도록 하려면 Turn on managed scaling(관리형 크기 조정 켜기)을 선택합니다.
      - 실행 중인 Amazon ECS 작업이 있는 EC2 인스턴스가 종료되지 않도록 하려면 Turn on scaling protection(크기 조정 보호 켜기)을 선택합니다.
      - Set target capacity(대상 용량 설정)에서 Amazon ECS 관리형 대상 추적 조정 정책에서 사용되는 CloudWatch 지표에 대한 대상 값을 입력합니다.
6. 생성(Create)을 선택합니다.

Amazon ECS 용량 공급자 업데이트

Auto Scaling 그룹을 용량 공급자로 사용하는 경우 그룹의 조정 정책을 수정할 수 있습니다.

클러스터의 용량 공급자를 업데이트하는 방법(Amazon ECS 콘솔)

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
4. Cluster : ***name***(클러스터: 이름) 페이지에서 Infrastructure(인프라)를 선택한 다음 Update(업데이트)를 선택합니다.
5. Create capacity providers(용량 공급자 생성) 페이지에서 다음 옵션을 구성합니다.
   - Auto Scaling group(오토 스케일링)의 Scaling policies(스케일링 정책)에서 다음 옵션을 구성합니다.
     - Amazon ECS에서 스케일 인 및 스케일 아웃 작업을 관리하도록 하려면 Turn on managed scaling(관리형 크기 조정 켜기)을 선택합니다.
     - 실행 중인 Amazon ECS 작업이 있는 EC2 인스턴스가 종료되지 않도록 하려면 크기 조정 보호 켜기를 선택합니다.
     - Set target capacity(대상 용량 설정)에서 Amazon ECS 관리형 대상 추적 조정 정책에서 사용되는 CloudWatch 지표에 대한 대상 값을 입력합니다.
6. 업데이트를 선택합니다.

Amazon ECS 용량 공급자 삭제

사용을 마친 Auto Scaling 그룹 용량 공급자는 삭제할 수 있습니다. 그룹이 삭제된 후 Auto Scaling 그룹 용량 공급자는 `INACTIVE` 상태로 전환됩니다. `INACTIVE` 상태인 용량 공급자는 일정 기간 동안 계정에서 검색 가능한 상태로 유지될 수 있습니다. 하지만 이 동작은 향후 변경될 수 있으므로 `INACTIVE` 용량 공급자가 지속되는 상태에 의존해서는 안 됩니다. Auto Scaling 그룹 용량 공급자를 삭제하기 전에 모든 서비스의 용량 공급자 전략에서 해당 용량 공급자를 제거해야 합니다. Amazon ECS 콘솔의 `UpdateService` API 또는 업데이트 서비스 워크플로를 사용하여 서비스의 용량 공급자 전략에서 용량 공급자를 제거할 수 있습니다. 새 배포 강제 적용 옵션을 사용하면 용량 공급자가 제공한 Amazon EC2 인스턴스 용량을 사용하는 모든 작업이 나머지 용량 공급자의 용량을 사용하도록 전환될 수 있습니다.

클러스터에 대한 용량 공급자 삭제 방법(Amazon ECS 콘솔)

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

3. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
4. Cluster : ***name***(클러스터: name) 페이지에서 Infrastructure(인프라)를 선택하고 Auto Scaling을 선택한 다음 Delete(삭제)를 선택합니다.
5. 확인란에 delete ***Auto Scaling group name***(오토 스케일링 이름 삭제)을 입력합니다.
6. 삭제를 선택합니다.

EC2 인스턴스에서 실행되는 Amazon ECS 워크로드를 안전하게 중지

관리형 인스턴스 드레이닝은 Amazon EC2 인스턴스의 정상적인 종료를 지원합니다. 이를 통해 워크로드를 안전하게 중지하고 비종료 인스턴스로 일정을 다시 예약할 수 있습니다. 워크로드 중단에 대한 걱정 없이 인프라 유지 관리 및 업데이트가 수행됩니다. 관리형 인스턴스 드레이닝을 사용하면 Amazon EC2 인스턴스를 교체해야 하는 인프라 관리 워크플로를 간소화하는 동시에, 애플리케이션의 복원력과 가용성을 보장할 수 있습니다.

Amazon ECS 관리형 인스턴스 드레이닝은 Auto Scaling 그룹 인스턴스 교체와 함께 작동합니다. 인스턴스 새로 고침 및 최대 인스턴스 수명을 기준으로 고객은 용량에 대한 최신 OS 및 보안 요구 사항을 계속 준수해나갈 수 있습니다.

관리형 인스턴스 드레이닝은 Amazon ECS 용량 공급자와만 사용할 수 있습니다. Amazon ECS 콘솔, AWS CLI 또는 SDK를 사용하여 Auto Scaling 그룹 용량 공급자를 생성하거나 업데이트할 때 관리형 인스턴스 드레이닝을 켤 수 있습니다.

다음 이벤트는 Amazon ECS 관리형 인스턴스 드레이닝에 포함됩니다.

- Auto Scaling 그룹 인스턴스 새로 고침 - Auto Scaling 그룹의 Amazon EC2 인스턴스를 배치 단위로 수동 교체하는 대신 인스턴스 새로 고침을 사용하여 롤링 교체를 수행합니다. 이 방법은 많은 인스턴스를 교체해야 하는 경우에 유용합니다. 인스턴스 새로 고침은 Amazon EC2 콘솔 또는 `StartInstanceRefresh` API를 통해 시작됩니다. 관리형 종료 보호를 사용하는 경우 `StartInstanceRefresh` 직접 호출 시 스케일 인 보호를 위해 `Replace`를 선택해야 합니다.
- 최대 인스턴스 수명 - Auto Scaling 그룹 인스턴스를 교체할 때 최대 수명을 정의할 수 있습니다. 이는 내부 보안 정책 또는 규정 준수를 기반으로 교체 인스턴스를 예약하는 데 유용합니다.
- Auto Scaling 그룹 스케일 인 - 규모 조정 정책 및 예약된 조정 작업에 따라 Auto Scaling 그룹은 인스턴스의 자동 규모 조정을 지원합니다. Auto Scaling 그룹을 Amazon ECS 용량 공급자로 사용하면 실행 중인 태스크가 없을 때 Auto Scaling 그룹 인스턴스를 스케일 인할 수 있습니다.
- Auto Scaling 그룹 확인 - Auto Scaling 그룹은 비정상 인스턴스의 종료를 관리하기 위한 많은 상태 확인을 지원합니다.

- AWS CloudFormation 스택 업데이트 – AWS CloudFormation 스택에 `UpdatePolicy` 속성을 추가하여 그룹이 변경될 때 롤링 업데이트를 수행할 수 있습니다.
- 스팟 용량 재조정 - Auto Scaling 그룹은 Amazon EC2 용량 재조정 알림에 따라 중단 위험이 더 높은 스팟 인스턴스를 사전에 교체하려고 합니다. Auto Scaling 그룹은 교체가 시작되고 정상 상태이면 이전 인스턴스를 종료합니다. Amazon ECS 관리형 인스턴스 드레이닝은 스팟이 아닌 인스턴스를 드레이닝하는 것과 동일한 방식으로 스팟 인스턴스를 드레이닝합니다.
- 스팟 중단 - 스팟 인스턴스는 2분 알림과 함께 종료됩니다. Amazon ECS 관리형 인스턴스 드레이닝은 이에 대응하여 인스턴스를 드레이닝 상태로 전환합니다.

Amazon EC2 Auto Scaling 수명 주기 후크((관리형 인스턴스 드레이닝 포함)

Auto Scaling 그룹 수명 주기 후크를 사용하면 고객이 인스턴스 수명 주기의 특정 이벤트에 의해 트리거되는 솔루션을 생성하고 해당 특정 이벤트가 발생할 때 사용자 지정 작업을 수행할 수 있습니다. Auto Scaling 그룹은 최대 50개의 후크를 허용합니다. 종료 후크는 여러 개 존재할 수 있고 병렬로 수행되며 Auto Scaling 그룹은 모든 후크가 완료될 때까지 기다렸다가 인스턴스를 종료합니다.

Amazon ECS 관리형 후크 종료 외에도, 고유한 수명 주기 종료 후크를 구성할 수 있습니다. 수명 주기 후크에는 `default action`이 있으며 Amazon ECS 관리형 후크와 같은 다른 후크가 사용자 지정 후크로 인한 오류의 영향을 받지 않도록 기본값으로 `continue`를 설정하는 것을 권장합니다.

Auto Scaling 그룹 종료 수명 주기 후크를 이미 구성하고 Amazon ECS 관리형 인스턴스 드레이닝도 활성화한 경우 두 수명 주기 후크가 모두 수행됩니다. 하지만 상대적 타이밍은 보장되지 않습니다. 수명 주기 후크에는 제한 시간이 경과했을 때 수행할 작업을 지정하는 `default action` 설정이 있습니다. 실패할 경우 사용자 지정 후크의 기본 결과로 `continue`를 사용하는 것이 좋습니다. 이렇게 하면 다른 후크, 특히 Amazon ECS 관리형 후크가 사용자 지정 수명 주기 후크에서 발생한 오류의 영향을 받지 않습니다. `abandon`의 대체 결과로 인해 다른 모든 후크를 건너뛰므로, 이는 사용하지 않는 것이 좋습니다. Auto Scaling 그룹 수명 주기 후크에 대한 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 Amazon EC2 Auto Scaling 수명 주기 후크를 참조하세요.

작업 및 관리형 인스턴스 드레이닝

Amazon ECS 관리형 인스턴스 드레이닝은 컨테이너 인스턴스에 있는 기존 드레이닝 기능을 사용합니다. 컨테이너 인스턴스 드레이닝 기능은 Amazon ECS 서비스에 속하는 복제 작업에서 교체를 수행하고 중지합니다. `RunTask`에 의해 간접적으로 호출되는 것과 같은 `PENDING` 또는 `RUNNING` 상태의 독립 실행형 태스크는 영향을 받지 않습니다. 태스크가 수동으로 완료되거나 중지될 때까지 기다려야 합니다. 컨테이너 인스턴스는 모든 작업이 중지되거나 48시간이 경과할 때까지 `DRAINING` 상태로 남아 있습니다. 모든 복제 작업이 중지된 후 대몬 작업이 마지막으로 중지됩니다.

관리형 인스턴스 드레이닝 및 관리형 종료 보호

관리형 인스턴스 드레이닝은 관리형 종료가 비활성화된 경우에도 작동합니다. 관리형 종료 보호 기능에 대한 자세한 내용은 Amazon ECS가 종료하는 인스턴스 제어 섹션을 참조하세요.

다음 테이블에는 관리형 종료와 관리형 드레이닝의 다양한 조합에 대한 동작이 요약되어 있습니다.

관리형 종료	관리형 드레이닝	결과
활성화됨	활성화됨	Amazon ECS는 작업을 실행 중인 Amazon EC2 인스턴스가 스케일 인 이벤트로 인해 종료되지 않도록 보호합니다. 종료 보호가 설정되지 않거나 스팟 중단이 수신되었거나 인스턴스 새로 고침으로 강제 적용되는 인스턴스와 같이, 종료 중인 모든 인스턴스는 정상적으로

관리형 종료	관리형 드레이닝	결과
		드레이닝됩니다.
비활성	활성화됨	Amazon ECS는 태스크를 실행하는 Amazon EC2 인스턴스의 스케일 인을 차단하지 않습니다. 하지만 종료되는 모든 인스턴스는 정상적으로 드레이닝됩니다.

관리형 종료	관리형 드레이닝	결과
활성화됨	비활성	Amazon ECS는 작업을 실행 중인 Amazon EC2 인스턴스가 스케일 인 이벤트로 인해 종료되지 않도록 보호합니다. 하지만 스팟 중단 또는 강제 인스턴스 새로 고침으로 인해 또는 실행 중인 작업이 없는 경우에는 여전히 인스턴스가 종료될 수 있습니다. Amazon ECS는 이러한 인스턴스에 대해 정상적인 드레이닝을 수행하지 않으

관리형 종료	관리형 드레이닝	결과
		며, 중지된 후 교체 서비스 작업을 시작합니다.
비활성	비활성	Amazon EC2 인스턴스는 Amazon ECS 작업을 실행하는 경우에도 언제든지 스케일 인 또는 종료될 수 있습니다. Amazon ECS는 중지된 후 교체 서비스 작업을 시작합니다.

관리형 인스턴스 드레이닝 및 스팟 인스턴스 드레이닝

스팟 인스턴스 드레이닝을 사용하면 Amazon ECS 에이전트에 `ECS_ENABLE_SPOT_INSTANCE_DRAINING` 환경 변수를 설정하여 2분간의 스팟 중단에 대응하여 Amazon ECS에서 인스턴스를 드레이닝 상태로 설정할 수 있습니다. Amazon ECS 관리형 인스턴스 드레이닝을 사용하면 스팟 중단뿐 아니라 여러 가지 이유로 종료되는 Amazon EC2 인스턴스의 정상적인 종료를 지원할 수 있습니다. 예를 들어 Amazon EC2 Auto Scaling 용량 재조정을 사용하여 중단 위험이 높은 스팟 인스턴스를 사전에 교체할 수 있으며 관리형 인스턴스 드레이닝은 교체되는 스팟 인스턴스를 정상적으로 종료합니다. 관리형 인스턴스 드레이닝을 사용하는 경우 스팟 인스턴스 드레이닝을 별도로 활성화할 필요가 없으므로 Auto Scaling 그룹 사용자 데이터의

ECS_ENABLE_SPOT_INSTANCE_DRAINING이 중복됩니다. 스팟 인스턴스 드레이닝에 대한 자세한 내용은 스팟 인스턴스 섹션을 참조하세요.

EventBridge에서 관리형 인스턴스 드레이닝이 작동하는 방식

Amazon ECS 관리형 인스턴스 드레이닝 이벤트는 Amazon EventBridge에 게시되며, Amazon ECS는 계정의 기본 버스에서 EventBridge 관리형 규칙을 생성하여 관리형 인스턴스 드레이닝을 지원합니다. Lambda, Amazon SNS 및 Amazon SQS 등의 다른 AWS 서비스로 이러한 이벤트를 필터링하고 문제를 해결할 수 있습니다.

- Amazon EC2 Auto Scaling은 수명 주기 후크를 간접적으로 호출할 때 EventBridge에 이벤트를 전송합니다.
- 스팟 중단 알림은 EventBridge에 게시됩니다.
- Amazon ECS는 Amazon ECS 콘솔과 API를 통해 검색할 수 있는 오류 메시지를 생성합니다.
- EventBridge에서는 일시적 장애를 완화하기 위한 재시도 메커니즘이 기본 제공됩니다.

인스턴스를 안전하게 종료하도록 Amazon ECS 용량 공급자 구성

Amazon ECS 콘솔과 AWS CLI를 사용하여 Auto Scaling 그룹 용량 공급자를 생성하거나 업데이트할 때 관리형 인스턴스 드레이닝을 켤 수 있습니다.

> **Note**
>
> 용량 공급자를 생성하면 관리형 인스턴스 드레이닝이 기본적으로 켜져 있습니다.

다음은 AWS CLI를 사용하여 관리형 인스턴스 드레이닝이 활성화된 상태에서 용량 공급자를 생성하고 클러스터의 기존 용량 공급자에 대해 관리형 인스턴스 드레이닝을 활성화하는 예제입니다.

관리형 인스턴스 드레이닝 기능이 활성화된 상태에서 용량 공급자 생성

관리형 인스턴스 드레이닝이 활성화된 상태로 용량 공급자를 생성하려면 create-capacity-provider 명령을 사용합니다. managedDraining 파라미터를 ENABLED로 설정합니다.

```
aws ecs create-capacity-provider \
--name capacity-provider \
--auto-scaling-group-provider '{
  "autoScalingGroupArn": "asg-arn",
```

```
  "managedScaling": {
    "status": "ENABLED",
    "targetCapacity": 100,
    "minimumScalingStepSize": 1,
    "maximumScalingStepSize": 1
  },
  "managedDraining": "ENABLED",
  "managedTerminationProtection": "ENABLED",
}'
```

응답:

```
{
    "capacityProvider": {
        "capacityProviderArn": "capacity-provider-arn",
        "name": "capacity-provider",
        "status": "ACTIVE",
        "autoScalingGroupProvider": {
            "autoScalingGroupArn": "asg-arn",
            "managedScaling": {
                "status": "ENABLED",
                "targetCapacity": 100,
                "minimumScalingStepSize": 1,
                "maximumScalingStepSize": 1
            },
            "managedTerminationProtection": "ENABLED"
            "managedDraining": "ENABLED"
        }
    }
}
```

클러스터의 기존 용량 공급자에 대해 관리형 인스턴스 드레이닝 활성화

클러스터의 기존 용량 공급자에 대해 관리형 인스턴스 드레이닝을 활성화하려면 update-capacity-provider 명령을 사용합니다. managedDraining에서는 현재 DISABLED로 표시되고, updateStatus에서는 UPDATE_IN_PROGRESS로 표시됩니다.

```
aws ecs update-capacity-provider \
--name cp-draining \
--auto-scaling-group-provider '{
  "managedDraining": "ENABLED"
}
```

응답:

```
{
    "capacityProvider": {
        "capacityProviderArn": "cp-draining-arn",
        "name": "cp-draining",
        "status": "ACTIVE",
        "autoScalingGroupProvider": {
            "autoScalingGroupArn": "asg-draining-arn",
            "managedScaling": {
                "status": "ENABLED",
                "targetCapacity": 100,
                "minimumScalingStepSize": 1,
                "maximumScalingStepSize": 1,
                "instanceWarmupPeriod": 300
            },
            "managedTerminationProtection": "DISABLED",
            "managedDraining": "DISABLED" // before update
        },
        "updateStatus": "UPDATE_IN_PROGRESS", // in progress and need describe again to
 find out the result
        "tags": [
        ]
    }
}
```

describe-clusters 명령을 사용하고 ATTACHMENTS를 포함합니다. 관리형 인스턴스의 드레이닝 연결의 status는 PRECREATED이고 전체 attachmentsStatus는 UPDATING입니다.

```
aws ecs describe-clusters --clusters cluster-name --include ATTACHMENTS
```

응답:

```
{
    "clusters": [
        {
            ...

            "capacityProviders": [
                "cp-draining"
            ],
```

```
            "defaultCapacityProviderStrategy": [],
            "attachments": [
                # new precreated managed draining attachment
                {
                    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
                    "type": "managed_draining",
                    "status": "PRECREATED",
                    "details": [
                        {
                            "name": "capacityProviderName",
                            "value": "cp-draining"
                        },
                        {
                            "name": "autoScalingLifecycleHookName",
                            "value": "ecs-managed-draining-termination-hook"
                        }
                    ]
                },

                ...

            ],
            "attachmentsStatus": "UPDATING"
        }
    ],
    "failures": []
}
```

업데이트가 끝나면 `describe-capacity-providers`를 사용합니다. 그러면 `managedDraining`은 이제 `ENABLED`입니다.

```
aws ecs describe-capacity-providers --capacity-providers cp-draining
```

응답:

```
{
    "capacityProviders": [
        {
            "capacityProviderArn": "cp-draining-arn",
            "name": "cp-draining",
            "status": "ACTIVE",
            "autoScalingGroupProvider": {
                "autoScalingGroupArn": "asg-draning-arn",
```

```
                "managedScaling": {
                    "status": "ENABLED",
                    "targetCapacity": 100,
                    "minimumScalingStepSize": 1,
                    "maximumScalingStepSize": 1,
                    "instanceWarmupPeriod": 300
                },
                "managedTerminationProtection": "DISABLED",
                "managedDraining": "ENABLED" // successfully update
            },
            "updateStatus": "UPDATE_COMPLETE",
            "tags": []
        }
    ]
}
```

Amazon ECS 관리형 인스턴스 드레이닝 문제 해결

관리형 인스턴스 드레이닝 문제를 해결해야 할 수 있습니다. 다음은 사용 중 발생할 수 있는 문제와 그 해결 방법의 예입니다.

Auto Scaling을 사용하는 경우 최대 인스턴스 수명을 초과한 후에도 인스턴스가 종료되지 않습니다.

Auto Scaling 그룹을 사용하는 동안 최대 인스턴스 수명에 도달하거나 초과한 후에도 인스턴스가 종료되지 않으면 스케일 인으로부터 보호되기 때문일 수 있습니다. 관리형 종료를 끄고 관리형 드레이닝을 허용하여 인스턴스 재활용을 처리할 수 있습니다.

AWS Management Console을 사용하여 Amazon ECS 클러스터 Auto Scaling에 대한 리소스 생성

AWS Management Console을 사용하여 클러스터 Auto Scaling을 위한 리소스를 생성하는 방법을 알아보세요. 리소스에 이름이 필요한 경우 접두사 `ConsoleTutorial`을 사용하여 모든 리소스가 고유한 이름을 가지도록 하고 쉽게 찾을 수 있도록 합니다.

주제

- 사전 조건
- 1단계: Amazon ECS 클러스터 생성
- 2단계: 태스크 정의 등록
- 3단계: 작업 실행
- 4단계: 확인

- 5단계: 정리

사전 조건

이 자습서에서는 다음 사전 조건이 충족되었다고 가정합니다.

- Amazon ECS 사용 설정의 단계가 완료되었습니다.
- IAM 사용자는 AmazonECS_FullAccess IAM 정책 예제에 지정된 필수 권한을 가집니다.
- Amazon ECS 컨테이너 인스턴스 IAM 역할이 생성됩니다. 자세한 정보는 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.
- Amazon ECS 서비스 연결 IAM 역할이 생성됩니다. 자세한 정보는 Amazon ECS에 대해 서비스 연결 역할 사용 섹션을 참조하세요.
- Auto Scaling 서비스 연결 IAM 역할이 생성됩니다. 자세한 정보는 Amazon EC2 Auto Scaling 사용 설명서의 Amazon EC2 Auto Scaling 서비스 연결 역할을 참조하세요.
- 사용할 VPC 및 보안 그룹이 생성되었습니다. 자세한 내용은 the section called “Virtual Private Cloud 생성” 섹션을 참조하세요.

1단계: Amazon ECS 클러스터 생성

다음 단계에 따라 Amazon ECS 클러스터를 생성합니다.

Amazon ECS는 AWS CloudFormation 스택의 일부로 사용자를 대신하여 Amazon EC2 Auto Scaling 시작 템플릿과 Auto Scaling 그룹을 생성합니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택한 다음 클러스터 생성을 선택합니다.
3. Cluster configuration(클러스터 구성)의 Cluster name(클러스터 이름)에 `ConsoleTutorial-cluster`를 입력합니다.
4. 인프라에서 AWS Fargate(서버리스)를 선택 취소한 다음 Amazon EC2 인스턴스를 선택합니다. 다음으로 용량 공급자 역할을 하는 Auto Scaling 그룹을 구성합니다.
    - Auto Scaling 그룹(ASG)에서 다음을 수행합니다. 새 ASG 생성을 선택한 후 그룹에 대한 다음 세부 정보를 입력합니다.
        - Operating system/Architecture(운영 체제/아키텍처)에서 Amazon Linux 2를 선택합니다.
        - EC2 instance type(EC2 인스턴스 유형)에서 t3.nano를 선택합니다.

- 용량(Capacity)에 Auto Scaling 그룹에서 시작할 최소 인스턴스 수와 최대 인스턴스 수를 입력합니다.

5. (선택 사항) 클러스터 태그를 관리하려면 태그(Tags)를 확장하고 다음 작업 중 하나를 수행합니다.

   [태그 추가] 태그 추가(Add tag)를 선택하고 다음을 수행합니다.

   - 키(Key)에 키 이름을 입력합니다.
   - 값에 키 값을 입력합니다.

   [태그 제거] 태그의 키와 값 오른쪽에 있는 제거를 선택합니다.

6. 생성(Create)을 선택합니다.

2단계: 태스크 정의 등록

클러스터에서 태스크를 실행하려면 먼저 태스크 정의를 등록해야 합니다. 태스크 정의는 그룹화된 컨테이너의 목록입니다. 다음 예제는 Docker Hub의 `amazonlinux` 이미지를 사용하고 대기하는 간단한 태스크 정의입니다. 사용 가능한 태스크 정의 파라미터에 대한 자세한 정보는 Amazon ECS 작업 정의 섹션을 참조하세요.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 작업 정의를 선택합니다.
3. 새 태스크 정의 생성(Create new task definition), JSON으로 새 태스크 정의 생성(Create new task definition with JSON)을 선택합니다.
4. JSON 편집기 상자에서 다음 내용을 복사하여 붙여 넣습니다.

```
{
    "family": "ConsoleTutorial-taskdef",
    "containerDefinitions": [
        {
            "name": "sleep",
            "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
            "memory": 20,
            "essential": true,
            "command": [
                "sh",
                "-c",
                "sleep infinity"
            ]
```

```
        }
    ],
    "requiresCompatibilities": [
        "EC2"
    ]
}
```

5. 생성(Create)을 선택합니다.

3단계: 작업 실행

계정에 대한 태스크 정의를 등록한 후 클러스터에서 태스크를 실행할 수 있습니다. 이 자습서에서는 `ConsoleTutorial-taskdef` 클러스터에서 `ConsoleTutorial-cluster` 태스크 정의의 인스턴스 5개를 실행합니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터 페이지에서 ConsoleTutorial-cluster를 선택합니다.
3. 작업에서 새 작업 실행을 선택합니다.
4. 환경 섹션의 컴퓨팅 옵션에서 용량 공급자 전략을 선택합니다.
5. 배포 구성에서 애플리케이션 유형으로 작업을 선택합니다.
6. 패밀리 드롭다운 목록에서 ConsoleTutorial-taskdef를 선택합니다.
7. 원하는 작업에 5를 입력합니다.
8. 생성(Create)을 선택합니다.

4단계: 확인

자습서의 이 지점에서, 5개의 작업이 실행되는 클러스터와 용량 공급자가 있는 Auto Scaling 그룹이 있어야 합니다. 용량 공급자가 Amazon ECS 관리형 조정을 활성화했습니다.

CloudWatch 지표, Auto Scaling 그룹 설정 및 마지막으로 Amazon ECS 클러스터 작업 수를 확인하여 모든 것이 제대로 작동하는지 확인할 수 있습니다.

클러스터에 대한 CloudWatch 지표를 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 화면 상단의 탐색 모음에서 리전을 선택합니다.
3. 탐색 창의 지표에서 모든 지표를 선택합니다.

4. 모든 지표 페이지의 찾아보기 탭에서 AWS/ECS/ManagedScaling을 선택합니다.
5. CapacityProviderName, ClusterName을 선택합니다.
6. ConsoleTutorial-cluster ClusterName에 해당하는 확인란을 선택합니다.
7. 그래프로 표시된 지표 탭에서 기간을 30초로 변경하고 통계를 최대로 변경합니다.

   그래프에 표시된 값은 용량 공급자에 대한 목표 용량 값을 보여 줍니다. 이것은 우리가 설정한 목표 용량 퍼센트인 100에서 시작되어야 합니다. 대상 추적 조정 정책에 대한 경보를 트리거하는 값인 200으로 확장됩니다. 그런 다음 경보가 Auto Scaling 그룹을 확장하도록 트리거합니다.

다음 단계에 따라 Auto Scaling 그룹 세부 정보를 보고 확장 작업이 발생했는지 확인합니다.

Auto Scaling 그룹이 확장되었는지 확인하려면

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 화면 상단의 탐색 모음에서 리전을 선택합니다.
3. 탐색 창의 Auto Scaling에서 Auto Scaling 그룹을 선택합니다.
4. 이 자습서에서 생성한 ConsoleTutorial-cluster Auto Scaling 그룹을 선택합니다. 원하는 용량에서 값을 확인하고 인스턴스 관리 탭에서 인스턴스를 확인하여 그룹이 인스턴스 2개로 스케일 아웃되었는지 확인합니다.

다음 단계를 사용하여 Amazon ECS 클러스터를 확인하여 Amazon EC2 인스턴스가 클러스터에 등록되었고 작업이 RUNNING 상태로 전환되었는지 확인합니다.

Auto Scaling 그룹에서 인스턴스 확인

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터(Clusters) 페이지에서 ConsoleTutorial-cluster 클러스터를 선택합니다.
4. 작업 탭에서 RUNNING 상태의 5개 작업이 표시되는지 확인합니다.

5단계: 정리

이 자습서를 완료한 후에는 사용하지 않는 리소스에 요금이 발생하지 않도록 연결된 리소스를 정리합니다. 용량 공급자 및 태스크 정의의 삭제는 지원되지 않지만 이러한 리소스와 관련된 비용은 없습니다.

자습서 리소스를 정리하려면

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 페이지에서 ConsoleTutorial-cluster를 선택합니다.
4. ConsoleTutorial-cluster 페이지에서 작업 탭을 선택한 다음 중지, 모두 중지를 선택합니다.
5. 탐색 창에서 클러스터를 선택합니다.
6. 클러스터 페이지에서 ConsoleTutorial-cluster를 선택합니다.
7. 페이지 오른쪽 상단에서 클러스터 삭제를 선택합니다.
8. 확인 상자에 ConsoleTutorial-cluster 삭제를 입력하고 삭제를 선택합니다.
9. 다음 단계에 따라 Auto Scaling 그룹을 삭제합니다.

    a. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
    b. 화면 상단의 탐색 모음에서 리전을 선택합니다.
    c. 탐색 창의 Auto Scaling에서 Auto Scaling 그룹을 선택합니다.
    d. `ConsoleTutorial-cluster` Auto Scaling 그룹을 선택한 다음 작업을 선택합니다.
    e. 작업 메뉴에서 삭제를 선택합니다. 확인 상자에 delete를 입력한 다음 삭제를 선택합니다.

Amazon ECS에 대한 Amazon EC2 컨테이너 인스턴스

Amazon ECS 컨테이너 인스턴스는 Amazon ECS 컨테이너 에이전트를 실행하고 클러스터에 등록되는 Amazon EC2 인스턴스입니다. Amazon ECS에서 EC2 시작 유형, 외부 시작 유형 또는 Auto Scaling 그룹 용량 공급자를 사용하여 작업을 실행하면 작업이 활성 컨테이너 인스턴스에 배치됩니다. 컨테이너 인스턴스 관리 및 유지 관리는 사용자의 책임입니다.

Amazon ECS에서 컨테이너식 워크로드를 실행하는 데 필요한 기본 사양을 충족하도록 자체 Amazon EC2 인스턴스 AMI를 만들 수도 있지만 Amazon ECS 최적화 AMI는 AWS 엔지니어들에 의해 Amazon ECS에서 사전 구성 및 테스트됩니다. AWS에서 신속하게 시작하고 컨테이너를 실행할 수 있는 가장 간단한 방법입니다.

콘솔을 사용하여 클러스터를 생성하면 Amazon ECS는 선택한 운영 체제와 연결된 최신 AMI를 사용하여 인스턴스에 대한 시작 템플릿을 생성합니다.

AWS CloudFormation을 사용하여 클러스터를 생성할 때 SSM 파라미터는 Auto Scaling 그룹 인스턴스용 Amazon EC2 시작 템플릿의 일부입니다. 동적 Systems Manager 파라미터를 사용하여 배포할

Amazon ECS 최적화 AMI를 결정하도록 템플릿을 구성할 수 있습니다. 이 파라미터를 사용하면 스택을 배포할 때마다 EC2 인스턴스에 적용해야 하는 업데이트가 있는지 확인할 수 있습니다. Systems Manager 파라미터를 사용하는 방법에 대한 예제는 AWS CloudFormation 사용 설명서의 Create an Amazon ECS cluster with the Amazon ECS-optimized Amazon Linux 2023 AMI를 참조하세요.

- Amazon ECS 최적화 Linux AMI 메타데이터 검색
- Amazon ECS 최적화 Bottlerocket AMI 메타데이터 검색
- Amazon ECS 최적화 Windows AMI 메타데이터 검색

애플리케이션과 호환되는 인스턴스 유형 중에서 선택할 수 있습니다. 인스턴스가 크면 동시에 더 많은 작업을 시작할 수 있습니다. 인스턴스가 작으면 더 세분화된 방식으로 스케일 아웃하여 비용을 절감할 수 있습니다. 클러스터의 모든 애플리케이션에 맞는 단일 Amazon EC2 인스턴스 유형을 선택할 필요는 없습니다. 대신 각 그룹의 인스턴스 유형이 서로 다른 여러 Auto Scaling 그룹을 생성할 수 있습니다. 그런 다음 이러한 그룹 각각에 대해 Amazon EC2 용량 공급자를 생성할 수 있습니다.

사용할 인스턴스 패밀리 유형과 인스턴스 유형을 결정하려면 다음 지침을 따르세요.

- 애플리케이션의 특정 요구 사항을 충족하지 않는 인스턴스 유형 또는 인스턴스 패밀리를 제거합니다. 예를 들어 애플리케이션에 GPU가 필요한 경우 GPU가 없는 모든 인스턴스 유형을 제외할 수 있습니다.
- 네트워크 처리량과 스토리지를 포함한 요구 사항을 고려하세요.
- CPU와 메모리를 고려하세요. 일반적으로 CPU 및 메모리는 실행하려는 작업의 복제본을 하나 이상 수용할 수 있을 만큼 커야 합니다.

스팟 인스턴스

스팟 용량은 온디맨드 인스턴스에 비해 상당한 비용 절감 효과를 줄 수 있습니다. 스팟 용량이란 초과 용량으로, 온디맨드 또는 예약 용량보다 상당히 저렴합니다. 스팟 용량은 배치 처리 및 기계 학습 워크로드와 개발 및 스테이징 환경에 적합합니다. 더 일반적으로는, 일시적인 가동 중지가 허용되는 모든 워크로드에 적합합니다.

스팟 용량을 항상 사용할 수 있는 것은 아니므로 다음과 같은 결과가 초래될 수 있습니다.

- 수요가 매우 많은 기간에는 스팟 용량을 사용할 수 없을 수 있습니다. 이로 인해 Amazon EC2 스팟 인스턴스 시작이 지연될 수 있습니다. 이러한 경우 Amazon ECS 서비스는 작업 시작을 다시 시도하며, Amazon EC2 Auto Scaling 그룹도 필요한 용량을 사용할 수 있을 때까지 인스턴스 시작을 다시 시도합니다. Amazon EC2는 스팟 용량을 온디맨드 용량으로 대체하지 않습니다.

- 전체 용량 수요가 증가하면 2분 경고만 남기고 스팟 인스턴스와 작업이 종료될 수 있습니다. 경고가 전송된 후 인스턴스가 완전히 종료되기 전에 필요한 경우 작업에서 순서대로 종료를 시작해야 합니다. 이렇게 하면 오류 가능성을 최소화하는 데 도움이 됩니다. 정상 종료에 대한 자세한 내용은 Graceful shutdowns with ECS를 참조하세요.

스팟 용량 부족을 최소화하려면 다음 권장 사항을 고려하세요.

- 여러 리전 및 가용 영역 사용 - 스팟 용량은 리전 및 가용 영역에 따라 다릅니다. 여러 리전 및 가용 영역에서 워크로드를 실행하여 스팟 가용성을 개선할 수 있습니다. 가능하면 작업 및 인스턴스를 실행하는 리전의 모든 가용 영역에 있는 서브넷을 지정합니다.
- 여러 Amazon EC2 인스턴스 유형 사용 - Amazon EC2 Auto Scaling과 함께 혼합 인스턴스 정책을 사용하면 여러 인스턴스 유형이 Auto Scaling 그룹으로 시작됩니다. 이렇게 하면 필요할 때 스팟 용량 요청을 충족할 수 있습니다. 안정성을 극대화하고 복잡성을 최소화하려면 혼합 인스턴스 정책에서 CPU 및 메모리 양이 거의 같은 인스턴스 유형을 사용합니다. 이러한 인스턴스는 세대가 다른 인스턴스이거나 동일한 기본 인스턴스 유형의 변형일 수 있습니다. 이 경우 필요하지 않을 수 있는 추가 기능이 함께 제공될 수 있다는 점에 유의하세요. 이러한 목록의 예로는 m4.large, m5.large, m5a.large, m5d.large, m5n.large, m5dn.large 및 m5ad.large 등이 있습니다. 자세한 설명은 Amazon EC2 Auto Scaling 사용자 가이드의 여러 인스턴스 유형 및 구매 옵션이 포함된 Auto Scaling 그룹을 참조하세요.
- 용량 최적화 스팟 할당 전략 사용 - Amazon EC2 스팟을 사용할 때 용량 최적화 및 비용 최적화 할당 전략 중에서 선택할 수 있습니다. 새 인스턴스를 시작할 때 용량 최적화 전략을 선택하면 Amazon EC2 스팟은 선택한 가용 영역에서 가용성이 가장 높은 인스턴스 유형을 선택합니다. 이렇게 하면 인스턴스가 시작되는 즉시 종료될 가능성을 줄일 수 있습니다.

컨테이너 인스턴스에서 스팟 종료 알림을 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- 스팟 인스턴스 알림을 수신하도록 Amazon ECS Linux 컨테이너 인스턴스 구성
- 스팟 인스턴스 알림을 수신하도록 Amazon ECS Windows 컨테이너 인스턴스 구성

Amazon ECS 최적화 Linux AMI

> **Important**
>
> Amazon ECS 최적화 Amazon Linux 2 AMI는 업스트림 Amazon Linux 2 운영 체제의 동일한 EOL 날짜를 반영하여 2026년 6월 30일에 수명 종료에 도달합니다(자세한 내용은 Amazon Linux 2 FAQ 참조). 고객은 2028년까지 장기 지원을 포함하는 Amazon Linux 2023을 사용하도

록 애플리케이션을 업그레이드하는 것이 좋습니다. Amazon Linux 2에서 Amazon Linux 2023으로 마이그레이션하는 방법에 대한 자세한 내용은 Amazon Linux 2 Amazon ECS 최적화 AMI에서 Amazon Linux 2023 Amazon ECS 최적화 AMI로 마이그레이션을 참조하세요.

Amazon ECS는 이러한 요구 사항 및 권장 사항으로 미리 구성된 Amazon ECS 최적화 AMI를 제공하여 컨테이너 워크로드를 실행합니다. Amazon EC2 인스턴스에 대해 Amazon ECS 최적화 Amazon Linux 2023 AMI를 사용하는 것이 좋습니다. 최신 Amazon ECS 최적화 AMI에서 컨테이너 인스턴스를 시작하면 최신 보안 업데이트 및 컨테이너 에이전트 버전을 받을 수 있습니다. 인스턴스를 시작하는 방법에 대한 자세한 정보는 Amazon ECS Linux 컨테이너 인스턴스 시작를 참조하세요.

콘솔을 사용하여 클러스터를 생성하면 Amazon ECS는 선택한 운영 체제와 연결된 최신 AMI를 사용하여 인스턴스에 대한 시작 템플릿을 생성합니다.

AWS CloudFormation을 사용하여 클러스터를 생성할 때 SSM 파라미터는 Auto Scaling 그룹 인스턴스용 Amazon EC2 시작 템플릿의 일부입니다. 동적 Systems Manager 파라미터를 사용하여 배포할 Amazon ECS 최적화 AMI를 결정하도록 템플릿을 구성할 수 있습니다. 이 파라미터를 사용하면 스택을 배포할 때마다 EC2 인스턴스에 적용해야 하는 업데이트가 있는지 확인할 수 있습니다. Systems Manager 파라미터를 사용하는 방법에 대한 예제는 AWS CloudFormation 사용 설명서의 Create an Amazon ECS cluster with the Amazon ECS-optimized Amazon Linux 2023 AMI를 참조하세요.

Amazon ECS에 최적화된 AMI를 사용자 지정해야 하는 경우 GitHub의 Amazon ECS Optimized AMI Build Recipes를 참조하세요.

Amazon Linux 2023 운영 시스템으로 Amazon EC2 인스턴스에 사용할 수 있는 Amazon ECS 최적화 AMI의 변형은 다음과 같습니다.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2023	Amazon ECS 최적화 Amazon Linux 2023 AMI	Amazon Linux 2023은 AWS의 차세대 Amazon Linux입니다. 대부분의 경우 Amazon ECS 워크로드에 대해 Amazon EC2 인스턴스를 시작하는 데 권장됩니다. 자세한 정보는	기본적으로 Amazon ECS 최적화 Amazon Linux 2023 AMI에는 총 30GiB 루트 볼륨이 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있

운영 체제	AMI	설명	스토리지 구성
		Amazon Linux 2023 사용 설명서의 What is Amazon Linux 2023을 참조하세요.	습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS 최적화 Amazon Linux 2023 AMI의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2023(arm64)	Amazon ECS 최적화 Amazon Linux 2023(arm64) AMI	Amazon Linux 2023을 기반으로 하는 이 AMI는 Amazon ECS 워크로드에 대해 Arm 기반 AWS Graviton/Graviton 2/Graviton 3/Graviton 4 프로세서로 구동되는 Amazon EC2 인스턴스를 시작할 때 사용하는 것이 좋습니다. 자세한 내용은 Amazon EC2 인스턴스 유형 안내서의 Amazon EC2 범용 인스턴스를 위한 사양을 참조하세요.	기본적으로 Amazon ECS 최적화 Amazon Linux 2023 AMI에는 총 30GiB 루트 볼륨이 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS 최적화 Amazon Linux 2023 AMI의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2023(Neuron)	Amazon ECS 최적화 Amazon Linux 2023 AMI	Amazon Linux 2023을 기반으로 하는 이 AMI는 Amazon EC2 Inf1, Trn1 또는 Inf2 인스턴스를 위한 것입니다. AWS Inferentia 및 AWS Trainium 드라이버와 Docker용 AWS Neuron 런타임이 사전 구성되어 있어 Amazon ECS에서 기계 학습 추론 워크로드를 쉽게 실행할 수 있습니다. 자세한 내용은 AWS Neuron 기계 학습 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요. Amazon ECS에 최적화된 Amazon Linux 2023(Neuron) AMI는 AWS CLI가 사전 설치되어 제공되지 않습니다.	기본적으로 Amazon ECS 최적화 Amazon Linux 2023 AMI에는 총 30GiB 루트 볼륨이 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS 최적화 Amazon Linux 2023 AMI의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2023 GPU	Amazon ECS 최적화 Amazon Linux 2023 GPU AMI	Amazon Linux 2023을 기반으로 하는 이 AMI는 Amazon ECS 워크로드의 Amazon EC2 GPU 기반 인스턴스를 시작할 때 사용하는 것이 좋습니다. NVIDIA 커널 드라이버와 Docker GPU 런타임으로 사전 구성되어 있어 Amazon ECS에서 GPU를 활용하는 워크로드를 실행합니다. 자세한 내용은 GPU 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요.	기본적으로 Amazon ECS 최적화 Amazon Linux 2023 AMI에는 총 30GiB 루트 볼륨이 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS 최적화 Amazon Linux 2023 AMI의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

Amazon Linux 2 운영 시스템으로 Amazon EC2 인스턴스에 사용할 수 있는 Amazon ECS 최적화 AMI의 변형은 다음과 같습니다.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2	Amazon ECS 최적화 Amazon Linux 2 커널 5.10 AMI	Amazon Linux 2를 기반으로 하는 이 AMI는 Amazon ECS 워크	기본적으로 Amazon Linux 2 기반 Amazon ECS 최적화

운영 체제	AMI	설명	스토리지 구성
		로드에 Amazon EC2 인스턴스를 시작하고 Linux 커널 4.14 대신 커널 5.10을 사용하려는 경우에 사용하기 위한 것입니다. Amazon ECS 최적화 Amazon Linux 2 커널 5.10 AMI에서 AWS CLI가 사전 설치되지 않은 상태입니다.	AMI(Amazon ECS 최적화 Amazon Linux 2 AMI, Amazon ECS 최적화 Amazon Linux 2(arm64) AMI 및 Amazon ECS GPU 최적화 AMI)는 단일 30GiB 루트 볼륨과 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS에 최적화 Amazon Linux 2 AMI 의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2	Amazon ECS 최적화 Amazon Linux 2 AMI	Amazon ECS 워크로드를 위한 것입니다. Amazon ECS 최적화 Amazon Linux 2 AMI에서 AWS CLI가 사전 설치되지 않은 상태입니다.	기본적으로 Amazon Linux 2 기반 Amazon ECS 최적화 AMI(Amazon ECS 최적화 Amazon Linux 2 AMI, Amazon ECS 최적화 Amazon Linux 2(arm64) AMI 및 Amazon ECS GPU 최적화 AMI)는 단일 30GiB 루트 볼륨과 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS에 최적화 Amazon Linux 2 AMI 의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2(arm64)	Amazon ECS 최적화 Amazon Linux 2 커널 5.10(arm64) AMI	Amazon Linux 2를 기반으로 하는 이 AMI는 Arm 기반 AWS Graviton/Graviton 2/Graviton 3/Graviton 4 프로세서로 구동되는 Amazon EC2 인스턴스를 위한 것이며, Amazon ECS 워크로드에 Linux 커널 4.14 대신 Linux 커널 5.10을 사용하려는 경우 사용할 수 있습니다. 자세한 내용은 Amazon EC2 인스턴스 유형 안내서의 Amazon EC2 범용 인스턴스를 위한 사양을 참조하세요. Amazon ECS 최적화 Amazon Linux 2(arm64) AMI에서 AWS CLI가 사전 설치되지 않은 상태입니다.	기본적으로 Amazon Linux 2 기반 Amazon ECS 최적화 AMI(Amazon ECS 최적화 Amazon Linux 2 AMI, Amazon ECS 최적화 Amazon Linux 2(arm64) AMI 및 Amazon ECS GPU 최적화 AMI)는 단일 30GiB 루트 볼륨과 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS에 최적화 Amazon Linux 2 AMI 의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2(arm64)	Amazon ECS 최적화 Amazon Linux 2(arm64) AMI	Amazon Linux 2를 기반으로 하는 이 AMI는 Amazon ECS 워크로드에 대해 Arm 기반 AWS Graviton/Graviton 2/Graviton 3/Graviton 4 프로세서로 구동되는 Amazon EC2 인스턴스를 시작할 때 사용하기 위한 것입니다. Amazon ECS 최적화 Amazon Linux 2(arm64) AMI에서 AWS CLI가 사전 설치되지 않은 상태입니다.	기본적으로 Amazon Linux 2 기반 Amazon ECS 최적화 AMI(Amazon ECS 최적화 Amazon Linux 2 AMI, Amazon ECS 최적화 Amazon Linux 2(arm64) AMI 및 Amazon ECS GPU 최적화 AMI)는 단일 30GiB 루트 볼륨과 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS에 최적화 Amazon Linux 2 AMI 의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2(GPU)	Amazon ECS GPU 최적화 커널 5.10 AMI	Amazon Linux 2를 기반으로 하는 이 AMI는 Amazon ECS 워크로드에 대해 Linux 커널 5.10에서 Amazon EC2 GPU 기반 인스턴스를 시작할 때 사용하는 것이 좋습니다. NVIDIA 커널 드라이버와 Docker GPU 런타임으로 사전 구성되어 있어 Amazon ECS에서 GPU를 활용하는 워크로드를 실행합니다. 자세한 내용은 GPU 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요.	기본적으로 Amazon Linux 2 기반 Amazon ECS 최적화 AMI(Amazon ECS 최적화 Amazon Linux 2 AMI, Amazon ECS 최적화 Amazon Linux 2(arm64) AMI 및 Amazon ECS GPU 최적화 AMI)는 단일 30GiB 루트 볼륨과 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS에 최적화 Amazon Linux 2 AMI 의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2(GPU)	Amazon ECS GPU 최적화 AMI	Amazon Linux 2를 기반으로 하는 이 AMI는 Amazon ECS 워크로드에 대해 Linux 커널 4.14에서 Amazon EC2 GPU 기반 인스턴스를 시작할 때 사용하는 것이 좋습니다. NVIDIA 커널 드라이버와 Docker GPU 런타임으로 사전 구성되어 있어 Amazon ECS에서 GPU를 활용하는 워크로드를 실행합니다. 자세한 내용은 GPU 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요.	기본적으로 Amazon Linux 2 기반 Amazon ECS 최적화 AMI(Amazon ECS 최적화 Amazon Linux 2 AMI, Amazon ECS 최적화 Amazon Linux 2(arm64) AMI 및 Amazon ECS GPU 최적화 AMI)는 단일 30GiB 루트 볼륨과 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS에 최적화 Amazon Linux 2 AMI 의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2(Neuron)	Amazon ECS 최적화 Amazon Linux 2(Neuron) 커널 5.10 AMI	Amazon Linux 2를 기반으로 하는 이 AMI는 Amazon EC2 Inf1, Trn1 또는 Inf2 인스턴스를 위한 것입니다. Linux 커널 5.10에서 AWS Inferentia 및 AWS Trainium 드라이버와 Docker용 AWS Neuron 런타임이 사전 구성되어 있어 Amazon ECS에서 기계 학습 추론 워크로드를 쉽게 실행할 수 있습니다. 자세한 내용은 AWS Neuron 기계 학습 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요. Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI는 AWS CLI가 사전 설치되지 않은 상태입니다.	기본적으로 Amazon Linux 2 기반 Amazon ECS 최적화 AMI(Amazon ECS 최적화 Amazon Linux 2 AMI, Amazon ECS 최적화 Amazon Linux 2(arm64) AMI 및 Amazon ECS GPU 최적화 AMI)는 단일 30GiB 루트 볼륨과 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS에 최적화 Amazon Linux 2 AMI 의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

운영 체제	AMI	설명	스토리지 구성
Amazon Linux 2(Neuron)	Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI	Amazon Linux 2를 기반으로 하는 이 AMI는 Amazon EC2 Inf1, Trn1 또는 Inf2 인스턴스를 위한 것입니다. AWS Inferentia 및 AWS Trainium 드라이버와 Docker용 AWS Neuron 런타임이 사전 구성되어 있어 Amazon ECS에서 기계 학습 추론 워크로드를 쉽게 실행할 수 있습니다. 자세한 내용은 AWS Neuron 기계 학습 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요. Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI는 AWS CLI가 사전 설치되지 않은 상태입니다.	기본적으로 Amazon Linux 2 기반 Amazon ECS 최적화 AMI(Amazon ECS 최적화 Amazon Linux 2 AMI, Amazon ECS 최적화 Amazon Linux 2(arm64) AMI 및 Amazon ECS GPU 최적화 AMI)는 단일 30GiB 루트 볼륨과 함께 제공됩니다. 시작 시 30GiB 루트 볼륨 크기를 수정하여 컨테이너 인스턴스에서 가용 스토리지를 늘릴 수 있습니다. 이 스토리지는 운영 체제와 Docker 이미지 및 메타데이터에 사용됩니다. Amazon ECS에 최적화 Amazon Linux 2 AMI 의 기본 파일 시스템은 `xfs`이며 Docker는`overlay2` 스토리지 드라이버를 사용합니다. 자세한 정보는 Docker 설명서의 OverlayFS 스토리지 드라이버 사용을 참조하세요.

Amazon ECS는 GitHub에서 Amazon ECS 최적화 AMI의 Linux 변형에 대한 변경 로그를 제공합니다. 자세한 정보는 Changelog를 참조하세요.

Amazon ECS 최적화 AMI의 Linux 변형은 Amazon Linux 2 AMI 또는 Amazon Linux 2023 AMI를 기반으로 사용합니다. Systems Manager Parameter Store API를 쿼리하여 각 변형에 대한 AMI 이름을 검색할 수 있습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 메타데이터 검색 섹션을 참조하세요. Amazon Linux 2 AMI 릴리스 정보도 제공됩니다. 자세한 정보는 Amazon Linux 2 릴리스 정보를 참조하세요. Amazon Linux 2023 릴리스 정보도 제공됩니다. 자세한 정보는 Amazon Linux 2023 release notes를 참조하세요.

다음 페이지에서는 변경 사항에 대한 추가 정보를 제공합니다.

- GitHub의 Source AMI release notes
- Docker 설명서의 Docker Engine 릴리스 정보
- NVIDIA 설명서의 NVIDIA 드라이버 설명서
- GitHub의 Amazon ECS Agent changelog

  `ecs-init` 애플리케이션의 소스 코드 및 에이전트 패키징을 위한 스크립트 및 구성은 이제 에이전트 리포지토리의 일부입니다. `ecs-init`의 이전 버전 및 패키징에 대해서는 GitHub의 Amazon ecs-init changelog를 참조하세요.

Amazon ECS 최적화 AMI에 보안 업데이트 적용

Amazon Linux를 기반으로 하는 Amazon ECS 최적화 AMI에는 cloud-init의 사용자 지정 버전이 포함되어 있습니다. Cloud-init는 클라우드 컴퓨팅 환경에서 Linux 이미지 부트스트랩을 수행하고 인스턴스를 시작할 때 원하는 작업을 수행하는 데 사용됩니다. 기본적으로 2024년 6월 12일 이전에 출시된 Amazon Linux 기반의 모든 Amazon ECS 최적화 AMI에는 인스턴스 시작 시 모든 '심각' 및 '중요' 보안 업데이트가 적용됩니다.

Amazon Linux 2를 기반으로 하는 Amazon ECS 최적화 AMI의 2024년 6월 12일 릴리스부터 기본 동작에 더 이상 시작 시 패키지 업데이트가 포함되지 않습니다. 대신 릴리스가 출시되면 새로운 Amazon ECS 최적화 AMI로 업데이트하는 것이 좋습니다. Amazon ECS 최적화 AMI는 사용 가능한 보안 업데이트 또는 기본 AMI 변경 내용이 있을 때 릴리스됩니다. 이렇게 하면 최신 패키지 버전과 보안 업데이트를 받고 인스턴스 시작을 통해 패키지 버전을 변경할 수 없게 됩니다. 최신 Amazon ECS 최적화 AMI 검색에 대한 자세한 내용은 Amazon ECS 최적화 Linux AMI 메타데이터 검색 섹션을 참조하세요.

새 AMI가 출시되면 새 AMI로 업데이트하도록 환경을 자동화하는 것이 좋습니다. 사용 가능한 옵션에 대한 자세한 내용은 Amazon ECS enables easier EC2 capacity management, with managed instance draining을 참조하세요.

AMI 버전에서 '심각' 및 '중요' 보안 업데이트를 수동으로 계속 적용하려면 Amazon EC2 인스턴스에서 다음 명령을 실행할 수 있습니다.

```
yum update --security
```

시작 시 보안 업데이트를 다시 활성화하려면 Amazon EC2 인스턴스를 시작할 때 cloud-init 사용자 데이터의 `#cloud-config` 섹션에 다음 줄을 추가할 수 있습니다. 자세한 내용은 Amazon Linux 사용 설명서의 Using cloud-init on Amazon Linux 2를 참조하세요.

```
#cloud-config
repo_upgrade: security
```

Amazon ECS 최적화 Linux AMI 메타데이터 검색

Amazon ECS 최적화 AMI 메타데이터를 프로그래밍 방식으로 검색할 수 있습니다. 메타데이터에는 AMI 이름, Amazon ECS 컨테이너 에이전트 버전, Docker 버전이 포함된 Amazon ECS 런타임 버전이 포함됩니다.

콘솔을 사용하여 클러스터를 생성하면 Amazon ECS는 선택한 운영 체제와 연결된 최신 AMI를 사용하여 인스턴스에 대한 시작 템플릿을 생성합니다.

AWS CloudFormation을 사용하여 클러스터를 생성할 때 SSM 파라미터는 Auto Scaling 그룹 인스턴스용 Amazon EC2 시작 템플릿의 일부입니다. 동적 Systems Manager 파라미터를 사용하여 배포할 Amazon ECS 최적화 AMI를 결정하도록 템플릿을 구성할 수 있습니다. 이 파라미터를 사용하면 스택을 배포할 때마다 EC2 인스턴스에 적용해야 하는 업데이트가 있는지 확인할 수 있습니다. Systems Manager 파라미터를 사용하는 방법에 대한 예제는 AWS CloudFormation 사용 설명서의 Create an Amazon ECS cluster with the Amazon ECS-optimized Amazon Linux 2023 AMI를 참조하세요.

Amazon ECS 최적화 AMI의 각 변형에 대한 AMI ID, 이미지 이름, 운영 체제, 컨테이너 에이전트 버전, 소스 이미지 이름 및 런타임 버전은 Systems Manager Parameter Store API를 쿼리하여 프로그래밍 방식으로 검색할 수 있습니다. Systems Manager Parameter Store API에 대한 자세한 정보는 GetParameters 및 GetParametersByPath를 참조하세요.

> **Note**
>
> Amazon ECS 최적화 AMI 메타데이터를 검색하려면 관리 사용자에게 다음과 같은 IAM 권한이 있어야 합니다. 이러한 권한은 `AmazonECS_FullAccess` IAM 정책에 추가되었습니다.
>
> - ssm:GetParameters
> - ssm:GetParameter

- ssm:GetParametersByPath

Systems Manager Parameter Store 파라미터 형식

다음은 각 Amazon ECS 최적화 AMI 변형에 대한 파라미터 이름의 형식입니다.

Linux Amazon ECS 최적화 AMI

- Amazon Linux 2023 AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/<version>
```

- Amazon Linux 2023(arm64) AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/arm64/<version>
```

- Amazon Linux 2023(Neuron) AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/neuron/<version>
```

- Amazon Linux 2023(GPU) AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/gpu/<version>
```

  Amazon Linux 2 AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/<version>
```

- Amazon Linux 2 커널 5.10 AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/<version>
```

- Amazon Linux 2(arm64) AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/arm64/<version>
```

- Amazon Linux 2 커널 5.10(arm64) AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/arm64/<version>
```

- Amazon ECS GPU 최적화 커널 5.10 AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/gpu/<version>
```

- Amazon Linux 2(GPU) AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/<version>
```

- Amazon ECS 최적화 Amazon Linux 2(Neuron) 커널 5.10 AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/inf/<version>
```

- Amazon Linux 2(Neuron) AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/inf/<version>
```

다음과 같은 파라미터 이름 형식은 하위 파라미터 `image_id`를 사용하여 최신 권장 Amazon ECS 최적화 Amazon Linux 2 AMI의 이미지 ID를 검색합니다.

```
/aws/service/ecs/optimized-ami/amazon-linux-2/recommended/image_id
```

다음 파라미터 이름 형식은 AMI 이름을 지정함으로써 특정 Amazon ECS 최적화 AMI 버전의 메타데이터를 가져옵니다.

- Amazon ECS 최적화 Amazon Linux 2 AMI 메타데이터:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/amzn2-ami-ecs-hvm-2.0.20181112-x86_64-ebs
```

> **Note**
>
> Amazon ECS 최적화 Amazon Linux 2 AMI의 버전을 모두 검색할 수 있습니다. Amazon ECS 최적화 AMI 버전 `amzn-ami-2017.09.l-amazon-ecs-optimized`(Linux) 이후 버전만 가져올 수 있습니다.

예시

다음 예에서는 Amazon ECS 최적화 AMI 변형에 대한 메타데이터를 검색할 수 있는 방법을 보여 줍니다.

최신 권장 Amazon ECS 최적화 AMI의 메타데이터 검색

다음과 같은 AWS CLI 명령과 AWS CLI를 함께 사용하여 최신 권장 Amazon ECS 최적화 AMI를 검색할 수 있습니다.

Linux Amazon ECS 최적화 AMI

- Amazon ECS 최적화 Amazon Linux 2023 AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
recommended --region us-east-1
```

- Amazon ECS 최적화 Amazon Linux 2023(arm64) AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
arm64/recommended --region us-east-1
```

- Amazon ECS 최적화 Amazon Linux 2023(Neuron) AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
neuron/recommended --region us-east-1
```

- Amazon ECS 최적화 Amazon Linux 2023 GPU AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/gpu/
recommended --region us-east-1
```

- Amazon ECS 최적화 Amazon Linux 2 커널 5.10 AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/recommended --region us-east-1
```

- Amazon ECS 최적화 Amazon Linux 2 AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
recommended --region us-east-1
```

- Amazon ECS 최적화 Amazon Linux 2 커널 5.10(arm64) AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/arm64/recommended --region us-east-1
```

- Amazon ECS 최적화 Amazon Linux 2(arm64) AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/arm64/
recommended --region us-east-1
```

- Amazon ECS GPU 최적화 커널 5.10 AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/gpu/recommended --region us-east-1
```

- Amazon ECS GPU 최적화 AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended --region us-east-1
```

- Amazon ECS 최적화 Amazon Linux 2(Neuron) 커널 5.10 AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/inf/recommended --region us-east-1
```

- Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/inf/
recommended --region us-east-1
```

권장되는 최신 Amazon ECS 최적화 Amazon Linux 2023 AMI의 이미지 ID 검색

하위 파라미터 `image_id`를 사용하여 권장되는 최신 Amazon ECS 최적화 Amazon Linux 2023 AMI ID의 이미지 ID를 검색할 수 있습니다.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-
linux-2023/recommended/image_id --region us-east-1
```

`image_id` 값만 검색하기 위해, 다음 예와 같이 특정 파라미터 값을 쿼리할 수 있습니다.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

특정 Amazon ECS 최적화 Amazon Linux 2 AMI 버전의 메타데이터 검색

AWS CLI와 AWS CLI 명령을 사용하여 다음 특정 Amazon ECS 최적화 Amazon Linux AMI 버전의 메타데이터를 가져옵니다. AMI 이름을 검색할 Amazon ECS 최적화 Amazon Linux AMI의 이름으로 바꿉니다.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/amzn2-ami-
ecs-hvm-2.0.20200928-x86_64-ebs --region us-east-1
```

Systems Manager GetParametersByPath API를 사용한 Amazon ECS 최적화 Amazon Linux 2 커널 5.10 AMI 메타데이터 검색

다음 명령과 AWS CLI를 사용한 Systems Manager GetParametersByPath API로 Amazon ECS 최적화 Amazon Linux 2 AMI 메타데이터 검색

```
aws ssm get-parameters-by-path --path /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/ --region us-east-1
```

권장되는 최신 Amazon ECS 최적화 Amazon Linux 2 커널 5.10 AMI의 이미지 ID 검색

하위 파라미터 `image_id`를 사용하여 권장되는 최신 Amazon ECS 최적화 Amazon Linux 2 커널 5.10 AMI ID의 이미지 ID를 검색할 수 있습니다.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/recommended/image_id --region us-east-1
```

`image_id` 값만 검색하기 위해, 다음 예와 같이 특정 파라미터 값을 쿼리할 수 있습니다.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

AWS CloudFormation 템플릿에서 최신 권장 Amazon ECS 최적화 AMI 사용

Systems Manager 파라미터 스토어 이름을 참조하여 AWS CloudFormation 템플릿에서 최신 Amazon ECS 최적화 AMI를 참조할 수 있습니다.

Linux 예

```
Parameters:kernel-5.10
  LatestECSOptimizedAMI:
    Description: AMI ID
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/recommended/
image_id
```

Amazon Linux 2에서 Amazon Linux 2023 Amazon ECS 최적화 AMI로 마이그레이션

Amazon Linux 이후 Amazon ECS는 2026년 6월 30일에 Amazon Linux 2 Amazon ECS 최적화 AMI에 대한 표준 지원을 종료합니다. 이 날짜 이후에는 Amazon ECS 에이전트 버전이 고정되고 소스 Amazon Linux 2 AMI가 업데이트된 경우에만 새 Amazon Linux 2 Amazon ECS 최적화 AMI가 게시됩니다. 전체 수명 종료(EOL)는 2026년 6월 30일에 적용되며, 그 이후에는 소스 AMI가 업데이트되더라도 Amazon ECS 최적화 Amazon Linux 2 AMI는 더 이상 게시되지 않습니다.

Amazon Linux 2023은 사전 구성된 보안 정책, 허용 모드의 SELinux, 기본적으로 활성화된 IMDSv2 전용 모드, 최적화된 부팅 시간, 향상된 보안 및 성능을 위한 개선된 패키지 관리를 통해 기본적으로 안전한 접근 방식을 제공합니다.

Amazon Linux 2와 Amazon Linux 2023 Amazon ECS 최적화 AMI 사이의 호환성은 매우 높은 수준이며, 대부분의 고객은 두 운영 체제 간 워크로드의 변경은 거의 없거나 최소 수준으로 유지됩니다.

자세한 내용은 Amazon Linux 2023 사용 설명서의 Comparing Amazon Linux 2 and Amazon Linux 2023 및 AL2023 FAQ를 참조하세요.

호환성 고려 사항

패키지 관리 및 OS 업데이트

이전 Amazon Linux 버전과 달리, Amazon ECS 최적화 Amazon Linux 2023 AMI는 특정 Amazon Linux 리포지토리 버전에서만 사용할 수 있습니다. 이를 통해 사용자가 실수로 패키지를 업데이트하여 원치 않거나 호환되지 않는 변경 내용이 발생하는 것을 방지할 수 있습니다. 자세한 내용은 Amazon Linux 2023 사용 설명서의 Managing repositories and OS updates in Amazon Linux 2023을 참조하세요.

Linux 커널 버전

Amazon Linux 2 AMI는 Linux 커널 4.14 및 5.10에 기반하지만, Amazon Linux 2023은 Linux 커널 6.1 및 6.12를 사용합니다. 자세한 내용은 Amazon Linux 2023 사용 설명서의 Comparing Amazon Linux 2 and Amazon Linux 2023 kernels을 참조하세요.

패키지 가용성 변경 내용

다음은 Amazon Linux 2023의 주목할 만한 패키지 변경 내용입니다.

- Amazon Linux 2의 일부 소스 바이너리 패키지는 Amazon Linux 2023에서 더 이상 사용할 수 없습니다. 자세한 내용은 Amazon Linux 2023 릴리스 정보의 Packages removed from Amazon Linux 2023을 참조하세요.
- Amazon Linux가 다양한 버전의 패키지를 지원하는 방법의 변경 내용. Amazon Linux 2에 사용되는 `amazon-linux-extras` 시스템이 Amazon Linux 2023에 존재하지 않습니다. 모든 패키지는 '코어' 리포지토리에서 사용할 수 있습니다.
- Extra Packages for Enterprise Linux(EPEL)가 Amazon Linux 2023에서 지원되지 않습니다. 자세한 내용은 Amazon Linux 2023 사용 설명서의 EPEL compatibility in Amazon Linux 2023을 참조하세요.
- 32비트 애플리케이션은 Amazon Linux 2023에서 지원되지 않습니다. 자세한 내용은 Amazon Linux 2023 사용 설명서의 Deprecated features from Amazon Linux 2를 참조하세요

제어 그룹(cgroups) 변경 내용

제어 그룹(cgroup)은 프로세스를 계층적으로 구성하고 프로세스 간에 시스템 리소스를 배포하는 Linux 커널 기능입니다. 제어 그룹은 `systemd`로 컨테이너 런타임을 광범위하게 구현할 수 있습니다.

Amazon ECS 에이전트, Docker 및 컨테이너는 모두 cgroupv1과 cgroupv2 둘 다 지원합니다. Amazon ECS 에이전트와 컨테이너 런타임은 사용자를 대신해 cgroup을 관리하므로 Amazon ECS 고객은이 기본 cgroup 업그레이드를 변경하지 않아도 됩니다.

cgroupv2에 대한 자세한 내용은 Amazon Linux 2023 사용 설명서의 Control groups v2 in Amazon Linux 2023을 참조하세요.

인스턴스 메타데이터 서비스(IMDS) 변경 내용

Amazon Linux 2023에는 기본적으로 인스턴스 메타데이터 서비스 버전 2(IMDSv2)가 필요합니다. IMDSv2는 보안 태세를 개선하는 데 도움이 되는 몇 가지 이점을 제공합니다. 세션을 시작하려면 간단한 HTTP PUT 요청으로 비밀 토큰을 생성해야 하는 세션 지향 인증 방법을 사용합니다. 세션의 토큰은 1초에서 6시간 사이로 유효할 수 있습니다.

IMDSv1에서 IMDSv2로의 전환에 대한 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 메타데이터 서비스 버전 2 사용으로 전환을 참조하세요.

IMDSv1을 사용하려는 경우 인스턴스 메타데이터 옵션 시작 속성을 사용하여 설정을 수동으로 재정의하면 계속 사용할 수 있습니다.

메모리 스왑 변경 내용

Amazon Linux 2023 및 cgroups v2에서는 컨테이너별 메모리 스왑이 지원되지 않습니다. 자세한 내용은 Amazon ECS에서 컨테이너 스왑 메모리 공간 관리 섹션을 참조하세요.

FIPS 검증 변경 내용

Amazon Linux 2는 FIPS 140-2 인증을 획득했고 Amazon Linux 2023은 FIPS 140-3 인증을 획득했습니다.

Amazon Linux 2023에서 FIPS 모드를 활성화하려면 Amazon EC2 인스턴스에 필요한 패키지를 설치하고 Amazon Linux 2023 사용 설명서의 Enable FIPS Mode on Amazon Linux 2023에 나온 지침에 따라 구성 단계를 수행합니다.

가속화된 인스턴스 지원

Amazon ECS 최적화 Amazon Linux 2023 AMI는 Neuron 및 GPU 가속 인스턴스 유형을 모두 지원합니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.

사용자 지정 AMI 빌드

공식적으로 지원되고 게시된 Amazon Linux 2023용 Amazon ECS 최적화 AMI로 전환하는 것이 좋지만, Amazon ECS 최적화 AMIs의 Linux 변형을 빌드하는 데 사용되는 오픈 소스 빌드 스크립트를 사용하여 사용자 지정 Amazon Linux 2 Amazon ECS 최적화 AMI를 계속 빌드할 수 있습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 구축 스크립트 섹션을 참조하세요.

마이그레이션 전략

철저한 애플리케이션 테스트를 포함하는 마이그레이션 계획을 생성하고 구현하는 것이 좋습니다. 다음 섹션에서는 Amazon ECS 인프라를 관리하는 방법에 따라 다양한 마이그레이션 전략을 간략하게 설명합니다.

Amazon ECS 용량 공급자로 마이그레이션

1. 새 시작 템플릿으로 새 용량 공급자를 생성하세요. 기존 항목과 와 유사한 시작 템플릿을 포함하는 Auto Scaling 그룹을 참조해야 하지만 Amazon Linux 2 Amazon ECS 최적화 AMI 대신 Amazon Linux 2023 변형 중 하나를 지정해야 합니다. 이 새 용량 공급자를 기존 Amazon ECS 클러스터에 추가하세요.
2. 기존 Amazon Linux 2 용량 공급자와 새로운 Amazon Linux 2023 용량 공급자를 모두 포함하도록 클러스터의 기본 용량 공급자 전략을 업데이트합니다. 먼저 Amazon Linux 2 공급자에서 높은 가

중치, Amazon Linux 2023 공급자에서 낮은 가중치로 시작하세요(예: Amazon Linux 2: 가중치 80, Amazon Linux 2023: 가중치 20). 그러면 새 태스크가 예약될 때 Amazon ECS가 Amazon Linux 2023 인스턴스 프로비저닝을 시작합니다. 인스턴스가 올바르게 등록되고 새 인스턴스에서 태스크가 성공적으로 실행될 수 있는지 확인하세요.

3. 클러스터의 기본 전략에서 용량 공급자 가중치를 점진적으로 조정하여 Amazon Linux 2023 공급자의 가중치를 늘리는 동시에 시간 경과에 따라 Amazon Linux 2 공급자 가중치를 줄이세요(예: 60/40, 40/60, 20/80의 순서로 조정). Amazon Linux 2023 인스턴스의 우선순위를 지정하도록 개별 서비스 용량 공급자 전략을 업데이트할 수도 있습니다. 태스크 배치를 모니터링하여 Amazon Linux 2023 인스턴스에서 성공적으로 실행되고 있는지 확인하세요.
4. 선택적으로 Amazon Linux 2 컨테이너 인스턴스를 드레이닝하여 태스크 마이그레이션을 가속하세요. Amazon Linux 2023 대체 용량이 충분한 경우 Amazon ECS 콘솔 또는 AWS CLI를 통해 Amazon Linux 2 컨테이너 인스턴스를 수동으로 드레이닝하여 Amazon Linux 2에서 Amazon Linux 2023으로 태스크 전환을 가속화할 수 있습니다. 마이그레이션이 완료되면 클러스터에서 Amazon Linux 2 용량 공급자를 제거하고 연결된 Auto Scaling 그룹을 삭제하세요.

Amazon EC2 Auto Scaling 그룹으로 마이그레이션

1. 새 시작 템플릿을 사용하여 새 Amazon EC2 Auto Scaling 그룹을 생성하세요. 기존 시작 템플릿과 유사하지만 Amazon Linux 2 Amazon ECS 최적화 AMI 대신 Amazon Linux 2023 변형 중 하나를 지정해야 합니다. 이 새로운 Auto Scaling 그룹은 기존 클러스터에서 인스턴스를 시작할 수 있습니다.
2. 먼저 Amazon Linux 2023 인스턴스가 클러스터에 등록되도록 Auto Scaling 그룹을 스케일 업하세요. 인스턴스가 올바르게 등록되고 새 인스턴스에서 태스크가 성공적으로 실행될 수 있는지 확인하세요.
3. Amazon Linux 2023에서 태스크가 작동하는지 확인한 후 모든 Amazon Linux 2 인스턴스를 완전히 교체할 때까지 Amazon Linux 2 Auto Scaling 그룹을 점진적으로 스케일 다운하면서 Amazon Linux 2023 Auto Scaling 그룹을 스케일 업하세요.
4. Amazon Linux 2023 대체 용량이 충분한 경우 컨테이너 인스턴스를 명시적으로 드레이닝하여 Amazon Linux 2에서 Amazon Linux 2023으로 태스크 전환을 가속화할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 드레이닝 섹션을 참조하세요.

수동 관리형 인스턴스로 마이그레이션

1. Amazon Linux 2 대신 Amazon ECS 최적화 Amazon Linux 2023 AMI를 사용하여 새 Amazon EC2 인스턴스를 수동으로 시작하거나 이를 시작하는 스크립트를 조정하세요. 이러한 인스턴스가 기존

Amazon Linux 2 인스턴스와 동일한 보안 그룹, 서브넷, IAM 역할 및 클러스터 구성을 사용하는지 확인하세요. 인스턴스는 시작 시 기존 Amazon ECS 클러스터에 자동으로 등록되어야 합니다.

2. 새 Amazon Linux 2023 인스턴스가 Amazon ECS 클러스터에 성공적으로 등록되고 `ACTIVE` 상태인지 확인하세요. 자연적인 태스크 배치를 기다리거나 일부 태스크를 수동으로 중지 또는 시작하여 일정 조정을 트리거함으로써 이러한 새 인스턴스에서 태스크를 예약하고 제대로 실행할 수 있는지 테스트하세요.
3. 필요한 경우 추가 Amazon Linux 2023 인스턴스를 시작한 다음 Amazon Linux 2 인스턴스를 하나씩 수동으로 드레이닝하고 종료하여 Amazon Linux 2 인스턴스를 점진적으로 교체하세요. 인스턴스를 `DRAINING` 상태로 설정하여 Amazon ECS 콘솔을 통해 인스턴스를 드레이닝할 수 있습니다. 그러면 인스턴스에 새 태스크가 배치되지 않고 기존 태스크가 완료되거나 다른 곳에서 다시 예약됩니다.

Amazon ECS 최적화 Linux AMI 구축 스크립트

Amazon ECS에서 Amazon ECS 최적화 AMI의 Linux 변형을 구축하는 데 사용되는 구축 스크립트를 오픈 소스로 제공합니다. 이러한 빌드 스크립트를 이제 GitHub에서 사용할 수 있습니다. 자세한 정보는 GitHub의 amazon-ecs-ami를 참조하세요.

Amazon ECS에 최적화된 AMI를 사용자 지정해야 하는 경우 GitHub에서 Amazon ECS Optimized AMI 빌드 레시피를 참조하세요.

구축 스크립트 리포지토리에는 Amazon ECS 최적화 AMI의 각 Linux 변형을 생성하기 위한 HashiCorp 패커 템플릿과 구축 스크립트가 포함되어 있습니다. 이러한 스크립트는 Amazon ECS 최적화 AMI 구축의 단일 출처이므로 GitHub 리포지토리를 따라 AMI에 대한 변경을 모니터링할 수 있습니다. 예를 들어 사용자가 Amazon ECS 팀이 공식 AMI에 사용하는 것과 동일한 버전의 Docker를 자체 AMI에 사용하기를 원할 수 있습니다.

자세한 정보는 GitHub의 aws/amazon-ecs-ami에서 Amazon ECS AMI 리포지토리를 참조하세요.

Amazon ECS 최적화 Linux AMI 구축

1. `aws/amazon-ecs-ami` GitHub 리포지토리를 복제합니다.

```
git clone https://github.com/aws/amazon-ecs-ami.git
```

2. AMI를 생성할 때 사용할 AWS 리전에 대한 환경 변수를 추가합니다. `us-west-2` 값을 사용할 리전으로 바꿉니다.

```
export REGION=us-west-2
```

3. AMI 구축을 위해 Makefile이 제공됩니다. 복제된 리포지토리의 루트 디렉터리에서 구축하려는 Amazon ECS 최적화 AMI의 Linux 변형에 해당하는 다음 명령 중 하나를 사용합니다.

   - Amazon ECS 최적화 Amazon Linux 2 AMI

     ```
     make al2
     ```

   - Amazon ECS 최적화 Amazon Linux 2(arm64) AMI

     ```
     make al2arm
     ```

   - Amazon ECS GPU 최적화 AMI

     ```
     make al2gpu
     ```

   - Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI

     ```
     make al2inf
     ```

   - Amazon ECS 최적화 Amazon Linux 2023 AMI

     ```
     make al2023
     ```

   - Amazon ECS 최적화 Amazon Linux 2023(arm64) AMI

     ```
     make al2023arm
     ```

   - Amazon ECS 최적화 Amazon Linux 2023(Neuron) AMI

     ```
     make al2023neu
     ```

Amazon ECS 최적화 Bottlerocket AMI

Bottlerocket은 Linux 기반 오픈 소스 운영 체제로, AWS에서 가상 머신 또는 베어 메탈 호스트에서 컨테이너를 실행하기 위해 특별히 개발했습니다. Amazon ECS 최적화 Bottlerocket AMI는 안전하며 컨테이너를 실행하는 데 필요한 최소 수의 패키지만 포함합니다. 이를 통해 리소스 사용량을 향상하고, 보안 공격 표면을 줄이고, 관리 오버헤드를 낮출 수 있습니다. 또한 Bottlerocket AMI는 Amazon ECS

와 통합되어 클러스터의 컨테이너 인스턴스 업데이트와 관련된 운영 오버헤드를 줄이는 데에도 도움이 됩니다.

Bottlerocket은 다음과 같은 점에서 Amazon Linux와 다릅니다.

- Bottlerocket은 패키지 관리자를 포함하지 않으며, 소프트웨어를 컨테이너로만 실행할 수 있습니다. Bottlerocket에 대한 업데이트는 한 단계로 적용하고 롤백할 수 있으므로, 업데이트 오류가 발생할 가능성이 줄어듭니다.
- Bottlerocket 호스트를 관리하는 기본 메커니즘은 컨테이너 스케줄러를 사용하는 것입니다. Amazon Linux와 달리 개별 Bottlerocket 인스턴스에 로그인하는 것은 고급 디버깅 및 문제 해결 목적으로만 드물게 사용됩니다.

Bottlerocket에 대한 자세한 내용은 GitHub의 설명서 및 릴리스를 참조하세요.

커널 6.1 및 커널 5.10용 Amazon ECS 최적화 Bottlerocket AMI 변형으로 여러 가지가 있습니다.

다음 변형은 커널 6.1을 사용합니다.

- `aws-ecs-2`
- `aws-ecs-2-nvidia`

다음 변형은 커널 5.10을 사용합니다.

- `aws-ecs-1`
- `aws-ecs-1-nvidia`

  `aws-ecs-1-nvidia` 변형에 대한 자세한 정보는 Announcing NVIDIA GPU support for Bottlerocket on Amazon ECS를 참조하세요.

고려 사항

Amazon ECS에서 Bottlerocket AMI를 사용할 때는 다음 사항을 고려해야 합니다.

- Bottlerocket은 `x86_64` 및 `arm64` 프로세서가 있는 Amazon EC2 인스턴스를 지원합니다. Bottlerocket AMI를 Inferentia 칩이 있는 Amazon EC2 인스턴스와 함께 사용하는 것은 권장되지 않습니다.

- Bottlerocket 이미지에는 SSH 서버 또는 쉘이 포함되지 않습니다. 하지만 대역 외 관리 도구를 사용하여 SSH 관리자 액세스 권한을 얻고 부트스트래핑을 수행할 수 있습니다. 자세한 내용은 GitHub의 bottlerocket README.md에 있는 관련 부분을 참조하세요.
  - 탐색
  - 관리자 컨테이너
- 기본적으로 Bottlerocket에는 사용 설정된 제어 컨테이너가 있습니다. 이 컨테이너는 Amazon EC2 Bottlerocket 인스턴스에서 명령을 실행하거나 셸 세션을 시작하는 데 사용할 수 있는 AWS Systems Manager 에이전트를 실행합니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 세션 관리자 설정을 참조하세요.

- Bottlerocket은 컨테이너 워크로드에 최적화되어 있으며 보안에 중점을 둡니다. Bottlerocket은 패키지 관리자를 포함하지 않으며 변경 불가능합니다. 보안 기능 및 지침에 대한 자세한 정보는 GitHub 웹사이트의 Security Features 및 Security Guidance를 참조하세요.
- `awsvpc` 네트워크 모드는 Bottlerocket AMI 버전 `1.1.0` 이상에 대해 지원됩니다.
- 작업 정의의 App Mesh는 Bottlerocket AMI 버전 `1.15.0` 이상에 대해 지원됩니다.
- `initProcessEnabled` 작업 정의 파라미터는 Bottlerocket AMI 버전 `1.19.0` 이상에서 지원됩니다.
- Bottlerocket AMI는 다음 서비스 및 기능도 지원하지 않습니다.
  - ECS Anywhere
  - Service Connect
  - 암호화 모드의 Amazon EFS
  - `awsvpc` 네트워크 모드의 Amazon EFS
  - Amazon EBS 볼륨을 탑재할 수 없음
  - Elastic Inference 액셀러레이터

Amazon ECS 최적화 Bottlerocket AMI 메타데이터 검색

AWS Systems Manager Parameter Store API를 쿼리하여 Amazon ECS 최적화 AMI의 Amazon Machine Image(AMI) ID를 검색할 수 있습니다. 이 파라미터를 사용하면 Amazon ECS 최적화 AMI ID를 수동으로 조회할 필요가 없습니다. Systems Manager 파라미터 스토어 API에 대한 자세한 내용은 GetParameter 섹션을 참조하세요. Amazon ECS 최적화 AMI 메타데이터를 검색하려면 사용자에 `ssm:GetParameter` IAM 권한이 있어야 합니다.

aws-ecs-2 Bottlerocket AMI 변형

AWS CLI 또는 AWS Management Console을 사용하여 AWS 리전 및 아키텍처별로 안정적인 최신 `aws-ecs-2` Bottlerocket AMI 변형을 검색할 수 있습니다.

- AWS CLI - 다음 AWS CLI 명령에서 하위 파라미터 `image_id`를 사용하여 권장되는 최신 Amazon ECS 최적화 Bottlerocket AMI의 이미지 ID를 검색할 수 있습니다. *region*은 필요한 AMI ID의 리전 코드로 바꿉니다. 지원되는 AWS 리전에 대한 자세한 내용은 GitHub에서 Finding an AMI를 참조하세요. 최신 버전이 아닌 다른 버전을 검색하려면 `latest` 항목을 버전 번호로 바꿉니다.
  - 64비트(x86_64) 아키텍처:

    ```
    aws ssm get-parameter --region us-east-2 --name "/aws/service/bottlerocket/aws-ecs-2/x86_64/latest/image_id" --query Parameter.Value --output text
    ```

  - 64비트 Arm(`arm64`) 아키텍처의 경우:

    ```
    aws ssm get-parameter --region us-east-2 --name "/aws/service/bottlerocket/aws-ecs-2/arm64/latest/image_id" --query Parameter.Value --output text
    ```

- AWS Management Console - AWS Management Console에서 URL을 사용하여 권장되는 Amazon ECS 최적화 AMI ID를 쿼리할 수 있습니다. URL을 사용하면 파라미터의 ID 값이 포함된 Amazon EC2 Systems Manager 콘솔이 열립니다. 다음 URL에서 *region*은 필요한 AMI ID의 리전 코드로 바꿉니다. 지원되는 AWS 리전에 대한 자세한 내용은 GitHub에서 Finding an AMI를 참조하세요.
  - 64비트(x86_64) 아키텍처:

    ```
    https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-ecs-2/x86_64/latest/image_id/description?region=region#
    ```

  - 64비트 Arm(`arm64`) 아키텍처의 경우:

    ```
    https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-ecs-2/arm64/latest/image_id/description?region=region#
    ```

aws-ecs-2-nvidia Bottlerocket AMI 변형

AWS CLI 또는 AWS Management Console을 사용하여 리전 및 아키텍처별로 안정적인 최신 `aws-ecs-2-nvdia` Bottlerocket AMI 변형을 검색할 수 있습니다.

- AWS CLI - 다음 AWS CLI 명령에서 하위 파라미터 `image_id`를 사용하여 권장되는 최신 Amazon ECS 최적화 Bottlerocket AMI의 이미지 ID를 검색할 수 있습니다. *`region`*은 필요한 AMI ID의 리전 코드로 바꿉니다. 지원되는 AWS 리전에 대한 자세한 내용은 GitHub에서 Finding an AMI를 참조하세요. 최신 버전이 아닌 다른 버전을 검색하려면 `latest` 항목을 버전 번호로 바꿉니다.
  - 64비트(x86_64) 아키텍처:

    ```
    aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-ecs-2-nvidia/x86_64/latest/image_id" --query Parameter.Value --output text
    ```

  - 64비트 Arm(`arm64`) 아키텍처의 경우:

    ```
    aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-ecs-2-nvidia/arm64/latest/image_id" --query Parameter.Value --output text
    ```

- AWS Management Console - AWS Management Console에서 URL을 사용하여 권장되는 Amazon ECS 최적화 AMI ID를 쿼리할 수 있습니다. URL을 사용하면 파라미터의 ID 값이 포함된 Amazon EC2 Systems Manager 콘솔이 열립니다. 다음 URL에서 *`region`*은 필요한 AMI ID의 리전 코드로 바꿉니다. 지원되는 AWS 리전에 대한 자세한 내용은 GitHub에서 Finding an AMI를 참조하세요.
  - 64비트(x86_64) 아키텍처의 경우:

    ```
    https://regionconsole.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-ecs-2-nvidia/x86_64/latest/image_id/description?region=region#
    ```

  - 64비트 Arm(`arm64`) 아키텍처의 경우:

    ```
    https://regionconsole.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-ecs-2-nvidia/arm64/latest/image_id/description?region=region#
    ```

`aws-ecs-1` Bottlerocket AMI 변형

AWS CLI 또는 AWS Management Console을 사용하여 AWS 리전 및 아키텍처별로 안정적인 최신 `aws-ecs-1` Bottlerocket AMI 변형을 검색할 수 있습니다.

- AWS CLI - 다음 AWS CLI 명령에서 하위 파라미터 `image_id`를 사용하여 권장되는 최신 Amazon ECS 최적화 Bottlerocket AMI의 이미지 ID를 검색할 수 있습니다. *`region`*은 필요한 AMI ID의 리전 코드로 바꿉니다. 지원되는 AWS 리전에 대한 자세한 내용은 GitHub에서 Finding an AMI를 참조하세요. 최신 버전이 아닌 다른 버전을 검색하려면 `latest` 항목을 버전 번호로 바꿉니다.
  - 64비트(x86_64) 아키텍처:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-ecs-1/x86_64/latest/image_id" --query Parameter.Value --output text
```

- 64비트 Arm(arm64) 아키텍처의 경우:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-ecs-1/arm64/latest/image_id" --query Parameter.Value --output text
```

- AWS Management Console - AWS Management Console에서 URL을 사용하여 권장되는 Amazon ECS 최적화 AMI ID를 쿼리할 수 있습니다. URL을 사용하면 파라미터의 ID 값이 포함된 Amazon EC2 Systems Manager 콘솔이 열립니다. 다음 URL에서 *region*은 필요한 AMI ID의 리전 코드로 바꿉니다. 지원되는 AWS 리전에 대한 자세한 내용은 GitHub에서 Finding an AMI를 참조하세요.
  - 64비트(x86_64) 아키텍처:

```
https://region.console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-ecs-1/x86_64/latest/image_id/description
```

  - 64비트 Arm(arm64) 아키텍처의 경우:

```
https://region.console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-ecs-1/arm64/latest/image_id/description
```

aws-ecs-1-nvidia Bottlerocket AMI 변형

AWS CLI 또는 AWS Management Console을 사용하여 리전 및 아키텍처별로 안정적인 최신 aws-ecs-1-nvdia Bottlerocket AMI 변형을 검색할 수 있습니다.

- AWS CLI - 다음 AWS CLI 명령에서 하위 파라미터 image_id를 사용하여 권장되는 최신 Amazon ECS 최적화 Bottlerocket AMI의 이미지 ID를 검색할 수 있습니다. *region*은 필요한 AMI ID의 리전 코드로 바꿉니다. 지원되는 AWS 리전에 대한 자세한 내용은 GitHub에서 Finding an AMI를 참조하세요. 최신 버전이 아닌 다른 버전을 검색하려면 latest 항목을 버전 번호로 바꿉니다.
  - 64비트(x86_64) 아키텍처:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-ecs-1-nvidia/x86_64/latest/image_id" --query Parameter.Value --output text
```

  - 64비트 Arm(arm64) 아키텍처의 경우:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-1-nvidia/arm64/latest/image_id" --query Parameter.Value --output text
```

- AWS Management Console - AWS Management Console에서 URL을 사용하여 권장되는 Amazon ECS 최적화 AMI ID를 쿼리할 수 있습니다. URL을 사용하면 파라미터의 ID 값이 포함된 Amazon EC2 Systems Manager 콘솔이 열립니다. 다음 URL에서 *region*은 필요한 AMI ID의 리전 코드로 바꿉니다. 지원되는 AWS 리전에 대한 자세한 내용은 GitHub에서 Finding an AMI를 참조하세요.
  - 64비트(x86_64) 아키텍처의 경우:

    ```
    https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/
    aws-ecs-1-nvidia/x86_64/latest/image_id/description?region=region#
    ```

  - 64비트 Arm(arm64) 아키텍처의 경우:

    ```
    https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/
    aws-ecs-1-nvidia/arm64/latest/image_id/description?region=region#
    ```

다음 단계

Amazon ECS에서 Bottlerocket 운영 체제를 시작하는 방법에 대한 자세한 자습서는 GitHub에서 Using a Bottlerocket AMI with Amazon ECS 및 AWS 블로그 사이트에서 Bottlerocket 및 Amazon ECS 시작하기를 참조하세요.

Bottlerocket 인스턴스를 시작하는 방법에 대한 자세한 내용은 Amazon ECS에 대한 Bottlerocket 인스턴스 시작 섹션을 참조하세요.

Amazon ECS에 대한 Bottlerocket 인스턴스 시작

컨테이너 워크로드를 실행할 수 있도록 Bottlerocket 인스턴스를 시작할 수 있습니다.

AWS CLI를 사용하여 Bottlerocket 인스턴스를 시작할 수 있습니다.

1. `userdata.toml`이라는 파일을 생성합니다. 이 파일은 인스턴스 사용자 데이터로 사용됩니다. *cluster-name*을 해당 클러스터의 이름으로 바꿉니다.

   ```
   [settings.ecs]
   cluster = "cluster-name"
   ```

2. the section called “Amazon ECS 최적화 Bottlerocket AMI 메타데이터 검색”에 포함된 명령 중 하나를 사용하여 Bottlerocket AMI ID를 가져옵니다. 다음 단계에서 이 정보를 사용합니다.
3. 다음 명령을 실행하여 Bottlerocket 인스턴스를 시작합니다. 다음 파라미터를 대체해야 합니다.

   - *subnet*을 인스턴스가 시작될 프라이빗 또는 퍼블릭 서브넷의 ID로 바꿉니다.
   - *bottlerocket_ami*를 이전 단계의 AMI ID로 바꿉니다.
   - *t3.large*를 사용하려는 인스턴스 유형으로 바꿉니다.
   - *region*을 리전 코드로 바꿉니다.

   ```
   aws ec2 run-instances --key-name ecs-bottlerocket-example \
      --subnet-id subnet \
      --image-id bottlerocket_ami \
      --instance-type t3.large \
      --region region \
      --tag-specifications
    'ResourceType=instance,Tags=[{Key=bottlerocket,Value=example}]' \
      --user-data file://userdata.toml \
      --iam-instance-profile Name=ecsInstanceRole
   ```
4. 다음 명령을 실행하여 컨테이너 인스턴스가 클러스터에 등록되었는지 확인합니다. 이 명령을 실행할 때 다음 파라미터를 바꿔야 합니다.

   - *cluster*를 클러스터 이름으로 바꿉니다.
   - *region*을 리전 코드로 바꿉니다.

   ```
   aws ecs list-container-instances --cluster cluster-name --region region
   ```

Amazon ECS에서 Bottlerocket 운영 체제를 시작하는 방법에 대한 자세한 연습은 GitHub의 Using a Bottlerocket AMI with Amazon ECS 및 AWS 블로그 사이트의 Getting started with Bottlerocket and Amazon ECS를 참조하세요.

Amazon ECS Linux 컨테이너 인스턴스 관리

Amazon ECS 워크로드에 EC2 인스턴스를 사용하는 경우 인스턴스를 유지 관리할 책임은 사용자에게 있습니다.

관리 절차

- Amazon ECS Linux 컨테이너 인스턴스 시작
- 데이터 전달을 위한 Amazon ECS Linux 컨테이너 인스턴스 부트스트래핑
- 스팟 인스턴스 알림을 수신하도록 Amazon ECS Linux 컨테이너 인스턴스 구성
- Amazon ECS Linux 컨테이너 인스턴스를 시작할 때 스크립트 실행
- Amazon ECS Linux 컨테이너 인스턴스 네트워크 인터페이스 증가
- Amazon ECS Linux 컨테이너 인스턴스 메모리 예약
- AWS Systems Manager를 사용한 원격으로 Amazon ECS 컨테이너 인스턴스 관리
- Amazon ECS Linux 컨테이너 인스턴스에 HTTP 프록시 사용
- Amazon ECS Auto Scaling 그룹에 대해 사전 초기화된 인스턴스 구성
- Amazon ECS 컨테이너 에이전트 업데이트

각 Amazon ECS 컨테이너 에이전트 버전은 서로 다른 기능 세트를 지원하며 이전 버전에 대한 버그 수정을 제공합니다. 기능한 한 최신 버전의 Amazon ECS 컨테이너 에이전트를 사용할 것이 좋습니다. 컨테이너 에이전트를 최신 버전으로 업데이트하려면 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

각 에이전트 릴리스에 포함된 기능 및 기능 향상을 보려면 https://github.com/aws/amazon-ecs-agent/releases를 참조하세요.

> **Important**
>
> 신뢰할 수 있는 지표를 위한 최소 Docker 버전은 Docker 버전 `v20.10.13` 이상이며, Amazon ECS 최적화 AMI `20220607` 이상 버전에 포함되어 있습니다.
> Amazon ECS 에이전트 버전 `1.20.0` 이상에서는 `18.01.0` 이전 Docker 버전에 대한 지원이 중단되었습니다.

Amazon ECS Linux 컨테이너 인스턴스 시작

Amazon EC2 콘솔을 사용하여 Amazon ECS 컨테이너 인스턴스를 생성할 수 있습니다.

Amazon EC2 콘솔, AWS CLI 및 SDK을 비롯한 다양한 방법으로 인스턴스를 시작할 수 있습니다. 인스턴스를 시작하는 다른 방법에 대한 자세한 정보는 Amazon EC2 사용 설명서의 인스턴스 시작을 참조하세요.

시작 마법사에 대한 자세한 내용은 Amazon EC2 사용 설명서의 새 인스턴스 시작 마법사를 사용하여 인스턴스 시작을 참조하세요.

시작하기 전에 Amazon ECS 사용 설정의 단계를 완료해야 합니다.

새 Amazon EC2 마법사를 사용하여 인스턴스를 시작할 수 있습니다. 인스턴스 시작 마법사는 인스턴스를 시작하는 데 필요한 시작 파라미터를 지정합니다.

인스턴스 구성을 위한 파라미터

- 절차
- 이름 및 태그
- 애플리케이션 및 OS 이미지(Amazon Machine Image)
- 인스턴스 유형
- 키 페어(로그인)
- 네트워크 설정
- 스토리지 구성
- 고급 세부 정보

절차

시작하기 전에 Amazon ECS 사용 설정의 단계를 완료해야 합니다.

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 화면 상단의 탐색 모음에는 현재 AWS 리전이 표시됩니다(예: 미국 동부(오하이오)). 인스턴스를 시작할 리전을 선택합니다.
3. Amazon EC2 콘솔 대시보드에서 인스턴스 시작을 선택합니다.

이름 및 태그

인스턴스 이름은 태그이며, 여기서 키는 이름이고 값은 사용자가 지정하는 이름입니다. 인스턴스, 볼륨 및 Elastic Graphics에 태그를 지정할 수 있습니다. 스팟 인스턴스의 경우 스팟 인스턴스 요청만 태깅할 수 있습니다.

인스턴스 이름과 추가 태그를 지정하는 것은 선택 사항입니다.

- 이름(Name)에 인스턴스를 설명하는 이름을 입력합니다. 이름을 지정하지 않으면 인스턴스를 시작할 때 자동으로 생성되는 ID로 인스턴스를 식별할 수 있습니다.

- 태그를 추가하려면 추가 태그 추가(Add additional tags)를 선택합니다. 태그 추가(Add tag)를 선택한 다음 키와 값을 입력하고 태그를 지정할 리소스 유형을 선택합니다. 추가할 각 추가 태그에 대해 태그 추가(Add tag)를 다시 선택합니다.

애플리케이션 및 OS 이미지(Amazon Machine Image)

Amazon Machine Image(AMI)에는 인스턴스를 생성하는 데 필요한 정보가 포함되어 있습니다. 예를 들어 AMI에는 웹 서버 역할을 수행하는 데 필요한 소프트웨어가 포함될 수 있습니다(예: Apache, 사용자의 웹 사이트).

검색창을 사용하여 AWS에서 공개한 적절한 Amazon ECS 최적화 AMI를 찾습니다.

1. 다음 용어 중 하나를 검색창에 입력합니다.
   - **`ami-ecs`**
   - Amazon ECS 최적화 AMI의 값.

     최신 Amazon ECS 최적화 AMI 및 이들의 값은 Linux Amazon ECS 최적화 AMI를 참조하세요.
2. Enter를 누릅니다.
3. Amazon Machine Image(AMI) 선택 페이지에서 AWSMarketplace AMIs 탭을 선택합니다.
4. 왼쪽의 결과 구체화(Refine results) 창에서 Amazon Web Services를 게시자(Publisher)로 선택합니다.
5. 사용하려는 AMI 행에서 선택(Select)을 선택합니다.

   또는, AMI를 선택하지 않고 시작 인스턴스 마법사로 돌아가려면 취소(Cancel)(오른쪽 상단)를 선택합니다. 기본 AMI가 선택됩니다. AMI가 Amazon ECS 최적화 Linux AMI에 설명된 요구 사항을 충족하는지 확인하세요.

인스턴스 유형

인스턴스 유형은 인스턴스의 하드웨어 구성과 크기를 정의합니다. 대형 인스턴스는 CPU와 메모리가 더 높습니다. 자세한 정보는 인스턴스 유형을 참조하세요.

- 인스턴스 유형(Instance type)에서 인스턴스에 대한 인스턴스 유형을 선택합니다.

  여기서 선택하는 인스턴스 유형은 실행할 작업에 사용 가능한 리소스를 결정합니다.

키 페어(로그인)

키 페어 이름(Key pair name)에서 기존 키 페어를 선택하거나 새로운 키 페어 생성(Create new key pair)을 선택하여 새로 생성합니다.

> **Important**
>
> 키 페어 없이 진행(Proceed without key pair)(권장하지 않음) 옵션을 선택할 경우 사용자가 다른 방법으로 로그인할 수 있도록 구성된 AMI를 선택해야만 인스턴스에 연결할 수 있습니다.

네트워크 설정

필요에 따라 네트워크 설정을 구성합니다.

- 네트워킹 플랫폼(Networking platform): Virtual Private Cloud (VPC)를 선택한 다음 네트워킹 인터페이스(Network interfaces) 섹션에서 서브넷을 지정합니다.
- VPC: 보안 그룹을 생성할 기존 VPC를 선택합니다.
- 서브넷(Subnet): 가용 영역, 로컬 영역, Wavelength Zone 또는 Outposts와 연결된 서브넷에서 인스턴스를 시작할 수 있습니다.

  가용 영역에서 인스턴스를 시작하려면 인스턴스를 시작할 서브넷을 선택합니다. 새 서브넷을 생성하려면 새 서브넷 생성을 선택하여 Amazon VPC 콘솔로 이동합니다. 마친 후에 인스턴스 시작 마법사로 돌아와 새로 고침 아이콘을 선택하면 해당 서브넷이 목록에 로딩됩니다.

  로컬 영역에서 인스턴스를 시작하려면 로컬 영역에 생성된 서브넷을 선택합니다.

  Outposts에서 인스턴스를 시작하려면 Outposts와 연결된 VPC의 서브넷을 선택합니다.
- 퍼블릭 IP 자동 할당(Auto-assign Public IP): 인터넷에서 인스턴스에 액세스할 수 있어야 한다면 퍼블릭 IP 자동 할당(Auto-assign Public IP) 필드가 사용 설정(Enable)으로 설정되어 있는지 확인합니다. 이렇게 설정이 되어 있지 않으면 이 필드를 비활성화(Disable)로 설정합니다.

> **Note**
>
> 컨테이너 인스턴스는 Amazon ECS 서비스 엔드포인트와 통신하기 위한 액세스 권한이 필요합니다. 액세스 권한은 인터페이스 VPC 엔드포인트를 통하거나 퍼블릭 IP 주소가 있는 컨테이너 인스턴스를 통해 부여할 수 있습니다.
> 인터페이스 VPC 엔드포인트에 대한 자세한 정보는 Amazon ECS 및 인터페이스 VPC 엔드포인트(AWS PrivateLink) 섹션을 참조하세요.

인터페이스 VPC 엔드포인트가 구성되어 있지 않고 컨테이너 인스턴스 컴퓨팅 리소스에 퍼블릭 IP 주소가 없는 경우, NAT(Network Address Translation)를 사용하여 이 액세스 권한을 제공해야 합니다. 자세한 정보는 Amazon VPC 사용 설명서의 NAT 게이트웨이 및 이 가이드의 Amazon ECS Linux 컨테이너 인스턴스에 HTTP 프록시 사용 섹션을 참조하세요.

- 방화벽(보안 그룹)(Firewall (security groups)): 보안 그룹을 사용하여 컨테이너 인스턴스의 방화벽 규칙을 정의합니다. 이 규칙은 컨테이너 인스턴스에 전달되는 수신 네트워크 트래픽을 지정합니다. 다른 모든 트래픽은 무시됩니다.
  - 기존 보안 그룹을 선택하려면 기존 보안 그룹 선택(Select an existing security group)을 선택하고 Amazon ECS 사용 설정에서 생성한 보안 그룹을 선택합니다.

스토리지 구성

선택한 AMI에는 루트 볼륨을 포함한 하나 이상의 스토리지 볼륨이 있습니다. 인스턴스에 연결할 추가 볼륨을 지정할 수 있습니다.

간단(Simple) 보기를 사용할 수 있습니다.

- 스토리지 유형(Storage type): 컨테이너 인스턴스의 스토리지를 구성합니다.

  Amazon ECS 최적화 Amazon Linux 2 AMI를 사용하는 경우, 운영 체제와 Docker 간에 공유하는 단일 30GiB 볼륨이 인스턴스에 구성됩니다.

  Amazon ECS 최적화 AMI를 사용하는 경우, 인스턴스에는 2개의 볼륨이 구성되어 있습니다. Root 볼륨은 운영 체제가 사용하고, 두 번째 Amazon EBS 볼륨(`/dev/xvdcz`에 연결)은 Docker가 사용합니다.

  애플리케이션 요구에 맞춰 인스턴스의 볼륨 크기를 선택적으로 늘리거나 줄일 수 있습니다.

고급 세부 정보

고급 세부 정보에서 필드를 볼 수 있도록 섹션을 확장하고 인스턴스를 위한 추가 파라미터를 지정합니다.

- 구매 옵션(Purchasing option): 스팟 인스턴스를 요청하려면 스팟 인스턴스 요청(Request Spot Instances)을 선택합니다. 스팟 인스턴스와 관련된 다른 필드도 설정해야 합니다. 자세한 정보는 스팟 인스턴스 요청을 참조하세요.

> **Note**
>
> 스팟 인스턴스를 사용하는 경우 `Not available` 메시지가 표시되면 다른 인스턴스 유형을 선택해야 할 수 있습니다.

.

- IAM 인스턴스 프로파일(IAM instance profile): 컨테이너 인스턴스 IAM 역할을 선택합니다. 이는 일반적으로 `ecsInstanceRole`로 이름이 지정됩니다.

> **Important**
>
> 적절한 IAM 권한을 사용하여 컨테이너 인스턴스를 시작하지 않으면 Amazon ECS 에이전트가 클러스터에 연결할 수 없습니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.

- 사용자 데이터: 사용자 데이터(예: Amazon ECS 컨테이너 에이전트 구성의 에이전트 환경 변수)로 Amazon ECS 컨테이너 인스턴스를 구성합니다. Amazon EC2 사용자 데이터 스크립트는 인스턴스가 처음 시작될 때 한 번만 실행됩니다. 다음은 사용자 데이터의 일반적인 용례입니다.
  - 기본적으로 컨테이너 인스턴스는 기본 클러스터로 시작됩니다. 기본이 아닌 클러스터로 시작하려면 고급 세부 정보(Advanced Details) 목록을 선택합니다. 그런 다음 사용자 데이터(User data) 필드에 다음 스크립트를 붙여 넣고 *your_cluster_name*을 클러스터 이름으로 대체합니다.

```
#!/bin/bash
echo ECS_CLUSTER=your_cluster_name >> /etc/ecs/ecs.config
```

  - Amazon S3에 `ecs.config` 파일이 있고 컨테이너 인스턴스 역할에 대한 Amazon S3 읽기 전용 액세스가 활성화된 경우, 고급 세부 정보(Advanced Details) 목록을 선택합니다. 그런 다음 사용자 데이터(User data) 필드에 다음 스크립트를 붙여 넣고 *your_bucket_name*을 AWS CLI를 설치할 버킷 이름으로 대체하고 시작 시 구성 파일을 작성합니다.

> **Note**
>
> 이 구성에 대한 자세한 정보는 Amazon S3에 Amazon ECS 컨테이너 인스턴스 구성 저장 섹션을 참조하세요.

```
#!/bin/bash
yum install -y aws-cli
aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
```

- ECS_CONTAINER_INSTANCE_TAGS 구성 파라미터를 사용하여 컨테이너 인스턴스의 태그를 지정합니다. 이렇게 하면 오로지 Amazon ECS과 연결된 태그가 생성되며, 이러한 태그는 Amazon EC2 API를 사용하여 볼 수 없습니다.

  > **Important**
  >
  > Amazon EC2 Auto Scaling 그룹을 사용하여 컨테이너 인스턴스를 시작한 경우 ECS_CONTAINER_INSTANCE_TAGS 에이전트 구성 파라미터를 사용하여 태그를 추가해야 합니다. 이는 Auto Scaling을 사용하여 시작된 Amazon EC2 인스턴스에 태그가 추가되는 방식 때문입니다.

  ```
  #!/bin/bash
  cat <<'EOF' >> /etc/ecs/ecs.config
  ECS_CLUSTER=your_cluster_name
  ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
  EOF
  ```

- 컨테이너 인스턴스의 태그를 지정한 다음, ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM 구성 파라미터를 사용하여 Amazon EC2에서 Amazon ECS로 태그를 전파합니다.

  컨테이너 인스턴스와 연결된 태그를 전파하고 your_cluster_name이라고 하는 클러스터에 컨테이너 인스턴스를 등록하는 사용자 데이터 스크립트의 예는 다음과 같습니다.

  ```
  #!/bin/bash
  cat <<'EOF' >> /etc/ecs/ecs.config
  ECS_CLUSTER=your_cluster_name
  ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
  EOF
  ```

자세한 내용은 데이터 전달을 위한 Amazon ECS Linux 컨테이너 인스턴스 부트스트래핑 섹션을 참조하세요.

데이터 전달을 위한 Amazon ECS Linux 컨테이너 인스턴스 부트스트래핑

Amazon EC2 인스턴스를 시작하면 사용자 데이터를 EC2 인스턴스에 전달할 수 있습니다. 이 데이터는 일반적인 구성 태스크를 자동으로 수행하는 데 사용할 수 있고, 인스턴스가 부팅될 때 스크립트를 실행하는 데 사용할 수도 있습니다. Amazon ECS의 경우 사용자 데이터의 가장 일반적인 사용 사례는 구성 정보를 Docker 대몬 및 Amazon ECS 컨테이너 에이전트에 전달하는 것입니다.

클라우드 boothook, 셸 스크립트, `cloud-init` 명령을 비롯하여 여러 유형의 사용자 데이터를 Amazon EC2에 전달할 수 있습니다. 이러한 유형 및 다른 형식 유형에 대한 자세한 정보는 Cloud-Init 설명서를 참조하세요.

Amazon EC2 시작 마법사를 사용할 때 사용자 데이터를 전달하려면 Amazon ECS Linux 컨테이너 인스턴스 시작 섹션을 참조하세요.

컨테이너 에이전트 구성 또는 Docker 대몬 구성의 데이터를 전달하도록 컨테이너 인스턴스를 구성할 수 있습니다.

Amazon ECS 컨테이너 에이전트

Linux 변형의 Amazon ECS 최적화 AMI는 컨테이너 에이전트가 시작될 때 `/etc/ecs/ecs.config` 파일에서 에이전트 구성 데이터를 찾습니다. 시작 시 Amazon EC2 사용자 데이터를 사용하여 이 구성 데이터를 지정할 수 있습니다. 사용 가능한 Amazon ECS 컨테이너 에이전트 구성 변수에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

클러스터 이름과 같이 단일 에이전트 구성 변수만 설정하려면 echo를 사용하여 구성 파일에 변수를 복사합니다.

```
#!/bin/bash
echo "ECS_CLUSTER=MyCluster" >> /etc/ecs/ecs.config
```

`/etc/ecs/ecs.config`에 작성할 여러 변수가 있는 경우 다음 heredoc 형식을 사용합니다. 이 형식은 cat으로 시작하는 라인과 `EOF` 사이의 모든 항목을 구성 파일에 작성합니다.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username":"my_name","password":"my_password","email":"email@example.com"}}
ECS_LOGLEVEL=debug
```

```
ECS_WARM_POOLS_CHECK=true
EOF
```

사용자 지정 인스턴스 속성을 설정하려면 ECS_INSTANCE_ATTRIBUTES 환경 변수를 설정합니다.

```
#!/bin/bash
cat <<'EOF' >> ecs.config
ECS_INSTANCE_ATTRIBUTES={"envtype":"prod"}
EOF
```

Docker 대몬

Amazon EC2 사용자 데이터를 사용하여 Docker 대몬 구성 정보를 지정할 수 있습니다. 이 구성 옵션에 대한 자세한 정보는 Docker 대몬 설명서를 참조하세요.

> **Note**
>
> 사용자 지정 Docker 구성은 경우에 따라 경고 없이 향후 Amazon ECS 변경 사항이나 기능과 충돌할 수 있으므로 AWS에서 지원하지 않습니다.

아래 예에서 사용자 정의 옵션이 Docker 대몬 구성 파일 /etc/docker/daemon.json에 추가되며, 파일은 인스턴스가 시작될 때 사용자 데이터에 지정됩니다.

```
#!/bin/bash
cat <<EOF >/etc/docker/daemon.json
{"debug": true}
EOF
systemctl restart docker --no-block
```

아래 예에서 사용자 정의 옵션이 Docker 대몬 구성 파일 /etc/docker/daemon.json에 추가되며, 파일은 인스턴스가 시작될 때 사용자 데이터에 지정됩니다. 이 예제에서는 Docker 대몬(daemon) 구성 파일에서 docker-proxy를 비활성화하는 방법을 보여줍니다.

```
#!/bin/bash
cat <<EOF >/etc/docker/daemon.json
{"userland-proxy": false}
EOF
systemctl restart docker --no-block
```

스팟 인스턴스 알림을 수신하도록 Amazon ECS Linux 컨테이너 인스턴스 구성

스팟 가격이 요청의 최고가를 초과하거나 용량이 더 이상 제공되지 않는 경우 Amazon EC2는 스팟 인스턴스를 종료, 중지 또는 최대 절전 모드로 전환합니다. Amazon EC2는 종료 및 중지 작업에 대해 스팟 인스턴스 2분 중단 알림을 제공합니다. 최대 절전 모드 작업에 대한 2분 알림을 제공하지 않습니다. 인스턴스에서 Amazon ECS 스팟 인스턴스 드레이닝이 활성화된 경우 Amazon ECS는 스팟 인스턴스 중단 알림을 수신하고 인스턴스를 `DRAINING` 상태로 둡니다.

> **Important**
>
> Amazon ECS는 Auto Scaling 용량 재분배에 의해 인스턴스가 제거될 때 Amazon EC2로부터 알림을 받지 않습니다. 자세한 정보는 Amazon EC2 Auto Scaling 용량 재분배를 참조하세요.

컨테이너 인스턴스를 `DRAINING`으로 설정할 경우 Amazon ECS는 새 작업이 컨테이너 인스턴스에서 배치를 위해 예약되지 않도록 합니다. 드레이닝 컨테이너 인스턴스에서 `PENDING` 상태인 서비스 작업이 즉시 중지됩니다. 클러스터에 사용 가능한 컨테이너 인스턴스가 있는 경우 거기서 대체 서비스 작업이 시작됩니다.

스팟 인스턴스 드레이닝은 기본적으로 꺼져 있습니다.

인스턴스를 시작할 때 스팟 인스턴스 드레인을 켤 수 있습니다. 사용자 데이터 필드에 다음 스크립트를 추가합니다. *`MyCluster`*를 컨테이너 인스턴스를 등록할 클러스터의 이름으로 바꿉니다.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
EOF
```

자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 시작 섹션을 참조하세요.

기존 컨테이너 인스턴스에 스팟 인스턴스 드레이닝 사용 설정

1. SSH를 통해 스팟 인스턴스에 연결합니다.
2. `/etc/ecs/ecs.config` 파일을 편집하고 다음을 추가합니다.

   ```
   ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
   ```

3. `ecs` 서비스를 다시 시작합니다.

- Amazon ECS 최적화 Amazon Linux 2 AMI의 경우:

```
sudo systemctl restart ecs
```

4. (선택 사항) 에이전트 내부 검사 API 태스크를 쿼리하여 에이전트가 실행 중인지 확인하고 새 컨테이너 인스턴스에 대한 일부 정보를 볼 수 있습니다. 자세한 내용은 the section called “컨테이너 내부 검사” 섹션을 참조하세요.

```
curl http://localhost:51678/v1/metadata
```

Amazon ECS Linux 컨테이너 인스턴스를 시작할 때 스크립트 실행

모니터링, 보안, 지표, 서비스 검색 또는 로깅과 같은 작업 또는 보안 문제를 처리하기 위해 모든 컨테이너 인스턴스에서 특정 컨테이너를 실행해야 할 수도 있습니다.

이렇게 하려면 시작 시 사용자 데이터 스크립트와 함께 docker run 명령(또는 일부 init 시스템에서는 Upstart나 systemd 등)을 호출하도록 컨테이너 인스턴스를 구성하는 것입니다. 이 방법은 효과적이지만 몇 가지 단점도 있는데, Amazon ECS는 컨테이너에 대한 지식이 없고 사용되는 CPU, 메모리, 포트 또는 기타 리소스를 모니터링할 수 없기 때문입니다. Amazon ECS가 모든 작업 리소스를 적절히 고려하도록 하려면 컨테이너 인스턴스에서 실행할 컨테이너의 태스크 정의를 생성합니다. 그런 다음 Amazon ECS를 사용하여 Amazon EC2 사용자 데이터와 함께 시작 시간에 태스크를 배치합니다.

다음 절차의 Amazon EC2 사용자 데이터 스크립트는 Amazon ECS 내부 검사 API를 사용하여 컨테이너 인스턴스를 식별한 다음 AWS CLI와 start-task 명령을 사용하여 시작 도중 지정된 태스크를 자체에서 실행합니다.

컨테이너 인스턴스 시작 시간에 태스크를 시작하려면

1. `ecsInstanceRole` IAM 역할을 수정하여 `StartTask` API 작업에 대한 권한을 추가합니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 역할 권한 업데이트를 참조하세요.
2. Amazon ECS 최적화 Amazon Linux 2 AMI를 사용하여 하나 이상의 컨테이너 인스턴스를 시작합니다. 새 컨테이너 인스턴스를 시작하고 EC2 사용자 데이터에서 다음 예제 스크립트를 사용합니다. *`your_cluster_name`*을 등록할 컨테이너 인스턴스의 클러스터로, *`my_task_def`*를 시작 시 인스턴스에서 실행할 태스크 정의로 바꿉니다.

   자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 시작 섹션을 참조하세요.

> **Note**
>
> 아래의 MIME 멀티파트 콘텐츠는 셸 스크립트를 사용하여 구성 값을 설정하고 패키지를 설치합니다. 또한 ecs 서비스가 실행되고 내부 검사 API를 사용할 수 있게 된 후 systemd 태스크를 사용하여 태스크를 시작합니다.

```
Content-Type: multipart/mixed; boundary="==BOUNDARY=="
MIME-Version: 1.0

--==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
# Specify the cluster that the container instance should register into
cluster=your_cluster_name

# Write the cluster configuration variable to the ecs.config file
# (add any other configuration variables here also)
echo ECS_CLUSTER=$cluster >> /etc/ecs/ecs.config

START_TASK_SCRIPT_FILE="/etc/ecs/ecs-start-task.sh"
cat <<- 'EOF' > ${START_TASK_SCRIPT_FILE}
 exec 2>>/var/log/ecs/ecs-start-task.log
 set -x

 # Install prerequisite tools
 yum install -y jq aws-cli

 # Wait for the ECS service to be responsive
 until curl -s http://localhost:51678/v1/metadata
 do
  sleep 1
 done

 # Grab the container instance ARN and AWS Region from instance metadata
 instance_arn=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .ContainerInstanceArn' | awk -F/ '{print $NF}' )
 cluster=$(curl -s http://localhost:51678/v1/metadata | jq -r '. | .Cluster' | awk
 -F/ '{print $NF}' )
```

```
 region=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .ContainerInstanceArn' | awk -F: '{print $4}')

 # Specify the task definition to run at launch
 task_definition=my_task_def

 # Run the AWS CLI start-task command to start your task on this container instance
 aws ecs start-task --cluster $cluster --task-definition $task_definition --
container-instances $instance_arn --started-by $instance_arn --region $region
EOF

# Write systemd unit file
UNIT="ecs-start-task.service"
cat <<- EOF > /etc/systemd/system/${UNIT}
      [Unit]
      Description=ECS Start Task
      Requires=ecs.service
      After=ecs.service

      [Service]
      Restart=on-failure
      RestartSec=30
      ExecStart=/usr/bin/bash ${START_TASK_SCRIPT_FILE}

      [Install]
      WantedBy=default.target
EOF

# Enable our ecs.service dependent service with `--no-block` to prevent systemd
 deadlock
# See https://github.com/aws/amazon-ecs-agent/issues/1707
systemctl enable --now --no-block "${UNIT}"
--==BOUNDARY==--
```

3. 컨테이너 인스턴스가 올바른 클러스터로 시작되고 태스크가 시작되었는지 확인합니다.

   a. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.

   b. 탐색 모음에서 클러스터가 속한 리전을 선택합니다.

   c. 탐색 창에서 클러스터를 선택하고 컨테이너 인스턴스를 호스팅하는 클러스터를 선택합니다.

   d. Cluster(클러스터) 페이지에서 Tasks(작업)를 선택한 다음 작업을 선택합니다.

      실행한 각 컨테이너 인스턴스에서 작업이 실행 중이어야 합니다.

작업이 보이지 않는 경우, SSH를 사용하여 컨테이너 인스턴스에 로그인하고 `/var/log/ecs/ecs-start-task.log` 파일에서 디버깅 정보를 확인할 수 있습니다.

Amazon ECS Linux 컨테이너 인스턴스 네트워크 인터페이스 증가

> **Note**
>
> 이 기능은 Fargate에서 사용할 수 없습니다.

`awsvpc` 네트워크 모드를 사용하는 각 작업은 고유한 탄력적 네트워크 인터페이스(ENI)를 수신합니다. 이 인터페이스는 이를 호스팅하는 컨테이너 인스턴스에 연결되어 있습니다. Amazon EC2 인스턴스에 연결할 수 있는 네트워크 인터페이스 수에 대한 기본 제한이 있으며 기본 네트워크 인터페이스는 한 개로 계산됩니다. 예를 들어 기본적으로 `c5.large` 인스턴스에는 ENI를 3개까지 연결할 수 있습니다. 인스턴스에 대한 기본 네트워크 인터페이스는 한 개로 계산되므로 인스턴스에 ENI를 2개 더 연결할 수 있습니다. `awsvpc` 네트워크 모드를 사용하는 각 작업에는 ENI가 필요하므로 대개 이 인스턴스 유형에서는 이러한 작업을 2개만 실행할 수 있습니다.

Amazon ECS에서는 지원되는 Amazon EC2 인스턴스 유형을 사용하여 늘어난 ENI 밀도와 함께 컨테이너 인스턴스를 시작할 수 있습니다. 이러한 인스턴스 유형을 사용하고 `awsvpcTrunking` 계정 설정을 활성화하면 새로 시작된 컨테이너 인스턴스에서 추가 ENI를 사용할 수 있습니다. 이 구성을 통해 각 컨테이너 인스턴스에 추가 작업을 배치할 수 있습니다. 콘솔을 사용하여 기능을 켜려면 Amazon ECS 계정 설정 수정 섹션을 참조하세요. AWS CLI를 사용하여 기능을 켜려면 AWS CLI를 사용하여 Amazon ECS 계정 설정 관리 섹션을 참조하세요.

예를 들어 `awsvpcTrunking`이 있는 `c5.large` 인스턴스의 경우 ENI 제한이 12로 증가합니다. 컨테이너 인스턴스는 기본 네트워크 인터페이스를 가지며 Amazon ECS는 "트렁크" 네트워크 인터페이스를 생성하여 컨테이너 인스턴스에 연결합니다. 따라서 이 구성을 통해 컨테이너 인스턴스에서 현재 두 개의 작업 대신 10개의 태스크를 시작할 수 있습니다.

트렁크 네트워크 인터페이스는 Amazon ECS에 의해 완전히 관리되며 클러스터에서 컨테이너 인스턴스를 종료하거나 등록 취소할 때 삭제됩니다. 자세한 내용은 EC2 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션 섹션을 참조하세요.

고려 사항

ENI 트렁킹 기능을 사용하는 경우에는 다음을 고려하세요.

- Amazon ECS 최적화 AMI의 Linux 변형 또는 버전 `1.28.1` 이상의 컨테이너 에이전트 및 버전 `1.28.1-2` 이상의 ecs-init 패키지를 포함하는 Amazon Linux 변형에서만 늘어난 ENI 제한을 지원합니다. Amazon ECS 최적화 AMI의 최신 Linux 변형을 사용하는 경우 이러한 요구 사항이 충족됩니다. Windows 컨테이너는 현재 지원되지 않습니다.
- `awsvpcTrunking`을 활성화한 후 시작된 새 Amazon EC2 인스턴스만 늘어난 ENI 제한 및 트렁크 네트워크 인터페이스를 수신합니다. 이전에 시작한 인스턴스는 수행된 작업에 상관없이 이러한 기능을 받지 않습니다.
- Amazon EC2 인스턴스에는 리소스 기반 IPv4 DNS 요청이 꺼져 있어야 합니다. 이 옵션을 비활성화하려면 Amazon EC2 콘솔에서 새 인스턴스를 생성할 때 리소스 기반 IPV4(A 레코드) DNS 요청 활성화 옵션을 삭제합니다. AWS CLI을(를) 사용하여 이 옵션을 비활성화하려면 다음 명령을 사용합니다.

```
aws ec2 modify-private-dns-name-options --instance-id i-xxxxxxx --no-enable-resource-name-dns-a-record --no-dry-run
```

- 공유 서브넷의 Amazon EC2 인스턴스는 지원되지 않습니다. 이 인스턴스를 사용하는 경우 클러스터에 등록하지 못합니다.
- 작업은 `awsvpc` 네트워크 모드 및 EC2 시작 유형을 사용해야 합니다. Fargate 시작 유형을 사용하는 작업은 시작한 인스턴스의 개수에 상관없이 항상 전용 ENI를 수신하기 때문에 이 기능이 필요하지 않습니다.
- 작업은 컨테이너 인스턴스와 동일한 Amazon VPC에서 시작해야 합니다. 작업이 동일한 VPC 내에 있지 않으면 속성 오류로 인해 태스크를 시작하지 못할 수 있습니다.
- 새 컨테이너 인스턴스를 시작할 때 인스턴스가 `REGISTERING` 상태로 전환되며 해당 인스턴스에 대해 트렁크 탄력적 네트워크 인터페이스가 프로비저닝됩니다. 등록이 실패하면 인스턴스가 `REGISTRATION_FAILED` 상태로 전환됩니다. 실패 이유를 설명하는 `statusReason` 필드를 확인하기 위해 컨테이너 인스턴스를 설명하여 실패한 등록 문제를 해결할 수 있습니다. 입니다. 그런 다음 컨테이너 인스턴스를 수동으로 등록 취소하거나 종료할 수 있습니다. 컨테이너 인스턴스가 성공적으로 등록 취소되거나 종료되면 Amazon ECS가 트렁크 ENI를 삭제합니다.

> **Note**
>
> Amazon ECS는 컨테이너 인스턴스 상태 변경 이벤트를 발생시켜 사용자가 `REGISTRATION_FAILED` 상태로 전환하는 인스턴스를 모니터링할 수 있습니다. 자세한 정보는 Amazon ECS 컨테이너 인스턴스 상태 변경 이벤트 섹션을 참조하세요.

- 컨테이너 인스턴스가 종료되면 인스턴스가 `DEREGISTERING` 상태로 전환되며 트렁크 탄력적 네트워크 인터페이스가 프로비저닝 해제됩니다. 그런 다음 인스턴스가 `INACTIVE` 상태로 전환됩니다.

- 늘어난 ENI 제한이 있는 퍼블릭 서브넷의 컨테이너 인스턴스가 중지된 후 다시 시작되면 인스턴스의 해당 퍼블릭 IP 주소가 없어지고 컨테이너 에이전트의 연결이 끊어집니다.
- `awsvpcTrunking`을 활성화하면 컨테이너 인스턴스는 VPC의 기본 보안 그룹을 사용하는 추가 ENI를 수신하며 Amazon ECS에서 관리됩니다.

  기본 VPC는 각 가용 영역의 퍼블릭 서브넷, 인터넷 게이트웨이 및 DNS 확인 활성화 설정과 함께 제공됩니다. 기본 라우팅 테이블은 인터넷으로 대상 주소가 정해진 서브넷의 트래픽을 인터넷 게이트웨이로 전송하기 때문에 서브넷은 퍼블릭 서브넷입니다. 대상 주소 0.0.0.0/0에서 인터넷 게이트웨이로의 라우팅을 제거함으로써 기본 서브넷을 프라이빗 서브넷으로 만들 수 있습니다. 하지만 이렇게 하면 해당 서브넷에서 실행하는 컨테이너 인스턴스는 인터넷에 액세스할 수 없습니다. 보안 그룹 규칙을 추가하거나 삭제하여 서브넷으로 들어오고 나가는 트래픽을 제어할 수 있습니다. 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 보안 그룹 규칙을 참조하세요.

사전 조건

늘어난 ENI 제한이 있는 컨테이너 인스턴스를 시작하기 전에 다음 필수 조건을 완료해야 합니다.

- Amazon ECS에 대한 서비스 연결 역할을 생성해야 합니다. Amazon ECS 서비스 연결 역할은 Amazon ECS에 사용자를 대신해서 다른 AWS 서비스를 호출할 수 있는 권한을 제공합니다. 이 역할은 클러스터를 생성하거나 AWS Management Console에서 서비스를 생성 또는 업데이트할 때 자동으로 생성됩니다. 자세한 정보는 Amazon ECS에 대해 서비스 연결 역할 사용 섹션을 참조하세요. 다음 AWS CLI 명령을 사용하여 서비스 연결 역할을 생성할 수도 있습니다.

  ```
  aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
  ```

- 계정 또는 컨테이너 인스턴스 IAM 역할이 `awsvpcTrunking` 계정 설정으로 활성화해야 합니다. 2개의 컨테이너 인스턴스 역할(`ecsInstanceRole`)을 생성하는 것이 좋습니다. 그런 다음, 역할 하나에 `awsvpcTrunking` 계정 설정을 활성화하고 ENI 트렁킹이 필요한 작업에 해당 역할을 사용할 수 있습니다. 컨테이너 인스턴스 역할에 대한 자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.

필수 조건이 충족되면 지원되는 Amazon EC2 인스턴스 유형 중 하나를 사용하여 새 컨테이너 인스턴스를 시작할 수 있으며, 해당 인스턴스에 늘어난 ENI 제한이 포함됩니다. 지원되는 인스턴스 유형의 목록은 증가한 Amazon ECS 컨테이너 네트워크 인터페이스에 대해 지원되는 인스턴스 섹션을 참조하세요. 컨테이너 인스턴스에는 버전 `1.28.1` 이상의 컨테이너 에이전트 및 버전 `1.28.1-2` 이상의 ecs-init 패키지가 있어야 합니다. Amazon ECS 최적화 AMI의 최신 Linux 변형을 사용하는 경우 이러한 요구 사항이 충족됩니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 시작 섹션을 참조하세요.

> **Important**
>
> Amazon EC2 인스턴스에는 리소스 기반 IPv4 DNS 요청이 꺼져 있어야 합니다. 이 옵션을 비활성화하려면 리소스 기반 IPV4(A 레코드) DNS 요청 활성화 옵션이 Amazon EC2 콘솔을 사용하여 새 인스턴스를 생성할 때 선택 취소되었는지 확인합니다. AWS CLI을(를) 사용하여 이 옵션을 비활성화하려면 다음 명령을 사용합니다.
>
> ```
> aws ec2 modify-private-dns-name-options --instance-id i-xxxxxxx --no-enable-resource-name-dns-a-record --no-dry-run
> ```

AWS CLI를 사용하여 늘어난 ENI 제한이 있는 컨테이너 인스턴스를 보는 방법

각 컨테이너 인스턴스에는 트렁크 네트워크 인터페이스라고 하는 기본 네트워크 인터페이스가 있습니다. 다음 명령을 통해 `ecs.awsvpc-trunk-id` 속성을 쿼리하여 늘어난 ENI 제한이 있는 컨테이너 인스턴스를 나열합니다. 이는 컨테이너 인스턴스에 트렁크 네트워크 인터페이스가 있음을 나타냅니다.

- list-attributes(AWS CLI)

```
aws ecs list-attributes \
      --target-type container-instance \
      --attribute-name ecs.awsvpc-trunk-id \
      --cluster cluster_name \
      --region us-east-1
```

- Get-ECSAttributeList(AWS Tools for Windows PowerShell)

```
Get-ECSAttributeList -TargetType container-instance -AttributeName ecs.awsvpc-trunk-id -Region us-east-1
```

증가한 Amazon ECS 컨테이너 네트워크 인터페이스에 대해 지원되는 인스턴스

다음은 지원되는 Amazon EC2 인스턴스 유형 및 `awsvpcTrunking` 계정 설정을 활성화하기 전과 후 각 인스턴스 유형에서 `awsvpc` 네트워크 모드를 사용하는 여러 작업을 시작할 수 있는 방법을 보여줍니다.

> **Important**
>
> 다른 인스턴스 유형이 동일한 인스턴스 패밀리에서 지원되지만 `a1.metal`, `c5.metal`, `c5a.8xlarge`, `c5ad.8xlarge`, `c5d.metal`, `m5.metal`, `p3dn.24xlarge`, `r5.metal`, `r5.8xlarge` 및 `r5d.metal` 인스턴스 유형은 지원되지 않습니다.
> `c5n`, `d3`, `d3en`, `g3`, `g3s`, `g4dn`, `i3`, `i3en`, `inf1`, `m5dn`, `m5n`, `m5zn`, `mac1`, `r5b`, `r5n`, `r5dn`, `u-12tb1`, `u-6tb1`, `u-9tb1` 및 `z1d` 인스턴스 패밀리는 지원되지 않습니다.

주제

- 범용
- 컴퓨팅 최적화
- 메모리 최적화
- 스토리지 최적화
- 액셀러레이티드 컴퓨팅
- 고성능 컴퓨팅

범용

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
a1.medium	1	10
a1.large	2	10
a1.xlarge	3	20
a1.2xlarge	3	40
a1.4xlarge	7	60
m5.large	2	10
m5.xlarge	3	20
m5.2xlarge	3	40

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
m5.4xlarge	7	60
m5.8xlarge	7	60
m5.12xlarge	7	60
m5.16xlarge	14	120
m5.24xlarge	14	120
m5a.large	2	10
m5a.xlarge	3	20
m5a.2xlarge	3	40
m5a.4xlarge	7	60
m5a.8xlarge	7	60
m5a.12xlarge	7	60
m5a.16xlarge	14	120
m5a.24xlarge	14	120
m5ad.large	2	10
m5ad.xlarge	3	20
m5ad.2xlarge	3	40
m5ad.4xlarge	7	60
m5ad.8xlarge	7	60
m5ad.12xlarge	7	60
m5ad.16xlarge	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
m5ad.24xlarge	14	120
m5d.large	2	10
m5d.xlarge	3	20
m5d.2xlarge	3	40
m5d.4xlarge	7	60
m5d.8xlarge	7	60
m5d.12xlarge	7	60
m5d.16xlarge	14	120
m5d.24xlarge	14	120
m5d.metal	14	120
m6a.large	2	10
m6a.xlarge	3	20
m6a.2xlarge	3	40
m6a.4xlarge	7	60
m6a.8xlarge	7	90
m6a.12xlarge	7	120
m6a.16xlarge	14	120
m6a.24xlarge	14	120
m6a.32xlarge	14	120
m6a.48xlarge	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
m6a.metal	14	120
m6g.medium	1	4
m6g.large	2	10
m6g.xlarge	3	20
m6g.2xlarge	3	40
m6g.4xlarge	7	60
m6g.8xlarge	7	60
m6g.12xlarge	7	60
m6g.16xlarge	14	120
m6g.metal	14	120
m6gd.medium	1	4
m6gd.large	2	10
m6gd.xlarge	3	20
m6gd.2xlarge	3	40
m6gd.4xlarge	7	60
m6gd.8xlarge	7	60
m6gd.12xlarge	7	60
m6gd.16xlarge	14	120
m6gd.metal	14	120
m6i.large	2	10

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
m6i.xlarge	3	20
m6i.2xlarge	3	40
m6i.4xlarge	7	60
m6i.8xlarge	7	90
m6i.12xlarge	7	120
m6i.16xlarge	14	120
m6i.24xlarge	14	120
m6i.32xlarge	14	120
m6i.metal	14	120
m6id.large	2	10
m6id.xlarge	3	20
m6id.2xlarge	3	40
m6id.4xlarge	7	60
m6id.8xlarge	7	90
m6id.12xlarge	7	120
m6id.16xlarge	14	120
m6id.24xlarge	14	120
m6id.32xlarge	14	120
m6id.metal	14	120
m6idn.large	2	10

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
m6idn.xlarge	3	20
m6idn.2xlarge	3	40
m6idn.4xlarge	7	60
m6idn.8xlarge	7	90
m6idn.12xlarge	7	120
m6idn.16xlarge	14	120
m6idn.24xlarge	14	120
m6idn.32xlarge	15	120
m6idn.metal	15	120
m6in.large	2	10
m6in.xlarge	3	20
m6in.2xlarge	3	40
m6in.4xlarge	7	60
m6in.8xlarge	7	90
m6in.12xlarge	7	120
m6in.16xlarge	14	120
m6in.24xlarge	14	120
m6in.32xlarge	15	120
m6in.metal	15	120
m7a.medium	1	4

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
m7a.large	2	10
m7a.xlarge	3	20
m7a.2xlarge	3	40
m7a.4xlarge	7	60
m7a.8xlarge	7	90
m7a.12xlarge	7	120
m7a.16xlarge	14	120
m7a.24xlarge	14	120
m7a.32xlarge	14	120
m7a.48xlarge	14	120
m7a.metal-48xl	14	120
m7g.medium	1	4
m7g.large	2	10
m7g.xlarge	3	20
m7g.2xlarge	3	40
m7g.4xlarge	7	60
m7g.8xlarge	7	60
m7g.12xlarge	7	60
m7g.16xlarge	14	120
m7g.metal	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
m7gd.medium	1	4
m7gd.large	2	10
m7gd.xlarge	3	20
m7gd.2xlarge	3	40
m7gd.4xlarge	7	60
m7gd.8xlarge	7	60
m7gd.12xlarge	7	60
m7gd.16xlarge	14	120
m7gd.metal	14	120
m7i.large	2	10
m7i.xlarge	3	20
m7i.2xlarge	3	40
m7i.4xlarge	7	60
m7i.8xlarge	7	90
m7i.12xlarge	7	120
m7i.16xlarge	14	120
m7i.24xlarge	14	120
m7i.48xlarge	14	120
m7i.metal-24xl	14	120
m7i.metal-48xl	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
m7i-flex.large	2	4
m7i-flex.xlarge	3	10
m7i-flex.2xlarge	3	20
m7i-flex.4xlarge	7	40
m7i-flex.8xlarge	7	60
m7i-flex.12xlarge	7	120
m7i-flex.16xlarge	14	120
m8g.medium	1	4
m8g.large	2	10
m8g.xLarge	3	20
m8g.2xlarge	3	40
m8g.4xlarge	7	60
m8g.8xlarge	7	60
m8g.12xlarge	7	60
m8g.16xlarge	14	120
m8g.24xlarge	14	120
m8g.48xlarge	14	120
m8g.metal-24xl	14	120
m8g.metal-48xl	14	120
m8gd.medium	1	4

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
m8gd.large	2	10
m8gd.xlarge	3	20
m8gd.2xlarge	3	40
m8gd.4xlarge	7	60
m8gd.8xlarge	7	60
m8gd.12xlarge	7	60
m8gd.16xlarge	14	120
m8gd.24xlarge	14	120
m8gd.48xlarge	14	120
m8gd.metal-24xl	14	120
m8gd.metal-48xl	14	120
mac2.metal	7	12
mac2-m1ultra.metal	7	12
mac2-m2.metal	7	12
mac2-m2pro.metal	7	12

컴퓨팅 최적화

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c5.large	2	10

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c5.xlarge	3	20
c5.2xlarge	3	40
c5.4xlarge	7	60
c5.9xlarge	7	60
c5.12xlarge	7	60
c5.18xlarge	14	120
c5.24xlarge	14	120
c5a.large	2	10
c5a.xlarge	3	20
c5a.2xlarge	3	40
c5a.4xlarge	7	60
c5a.12xlarge	7	60
c5a.16xlarge	14	120
c5a.24xlarge	14	120
c5ad.large	2	10
c5ad.xlarge	3	20
c5ad.2xlarge	3	40
c5ad.4xlarge	7	60
c5ad.12xlarge	7	60
c5ad.16xlarge	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c5ad.24xlarge	14	120
c5d.large	2	10
c5d.xlarge	3	20
c5d.2xlarge	3	40
c5d.4xlarge	7	60
c5d.9xlarge	7	60
c5d.12xlarge	7	60
c5d.18xlarge	14	120
c5d.24xlarge	14	120
c6a.large	2	10
c6a.xlarge	3	20
c6a.2xlarge	3	40
c6a.4xlarge	7	60
c6a.8xlarge	7	90
c6a.12xlarge	7	120
c6a.16xlarge	14	120
c6a.24xlarge	14	120
c6a.32xlarge	14	120
c6a.48xlarge	14	120
c6a.metal	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c6g.medium	1	4
c6g.large	2	10
c6g.xlarge	3	20
c6g.2xlarge	3	40
c6g.4xlarge	7	60
c6g.8xlarge	7	60
c6g.12xlarge	7	60
c6g.16xlarge	14	120
c6g.metal	14	120
c6gd.medium	1	4
c6gd.large	2	10
c6gd.xlarge	3	20
c6gd.2xlarge	3	40
c6gd.4xlarge	7	60
c6gd.8xlarge	7	60
c6gd.12xlarge	7	60
c6gd.16xlarge	14	120
c6gd.metal	14	120
c6gn.medium	1	4
c6gn.large	2	10

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c6gn.xlarge	3	20
c6gn.2xlarge	3	40
c6gn.4xlarge	7	60
c6gn.8xlarge	7	60
c6gn.12xlarge	7	60
c6gn.16xlarge	14	120
c6i.large	2	10
c6i.xlarge	3	20
c6i.2xlarge	3	40
c6i.4xlarge	7	60
c6i.8xlarge	7	90
c6i.12xlarge	7	120
c6i.16xlarge	14	120
c6i.24xlarge	14	120
c6i.32xlarge	14	120
c6i.metal	14	120
c6id.large	2	10
c6id.xlarge	3	20
c6id.2xlarge	3	40
c6id.4xlarge	7	60

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c6id.8xlarge	7	90
c6id.12xlarge	7	120
c6id.16xlarge	14	120
c6id.24xlarge	14	120
c6id.32xlarge	14	120
c6id.metal	14	120
c6in.large	2	10
c6in.xlarge	3	20
c6in.2xlarge	3	40
c6in.4xlarge	7	60
c6in.8xlarge	7	90
c6in.12xlarge	7	120
c6in.16xlarge	14	120
c6in.24xlarge	14	120
c6in.32xlarge	15	120
c6in.metal	15	120
c7a.medium	1	4
c7a.large	2	10
c7a.xlarge	3	20
c7a.2xlarge	3	40

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c7a.4xlarge	7	60
c7a.8xlarge	7	90
c7a.12xlarge	7	120
c7a.16xlarge	14	120
c7a.24xlarge	14	120
c7a.32xlarge	14	120
c7a.48xlarge	14	120
c7a.metal-48xl	14	120
c7g.medium	1	4
c7g.large	2	10
c7g.xlarge	3	20
c7g.2xlarge	3	40
c7g.4xlarge	7	60
c7g.8xlarge	7	60
c7g.12xlarge	7	60
c7g.16xlarge	14	120
c7g.metal	14	120
c7gd.medium	1	4
c7gd.large	2	10
c7gd.xlarge	3	20

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c7gd.2xlarge	3	40
c7gd.4xlarge	7	60
c7gd.8xlarge	7	60
c7gd.12xlarge	7	60
c7gd.16xlarge	14	120
c7gd.metal	14	120
c7gn.medium	1	4
c7gn.large	2	10
c7gn.xlarge	3	20
c7gn.2xlarge	3	40
c7gn.4xlarge	7	60
c7gn.8xlarge	7	60
c7gn.12xlarge	7	60
c7gn.16xlarge	14	120
c7gn.metal	14	120
c7i.large	2	10
c7i.xlarge	3	20
c7i.2xlarge	3	40
c7i.4xlarge	7	60
c7i.8xlarge	7	90

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c7i.12xlarge	7	120
c7i.16xlarge	14	120
c7i.24xlarge	14	120
c7i.48xlarge	14	120
c7i.metal-24xl	14	120
c7i.metal-48xl	14	120
c7i-flex.large	2	4
c7i-flex.xlarge	3	10
c7i-flex.2xlarge	3	20
c7i-flex.4xlarge	7	40
c7i-flex.8xlarge	7	60
c7i-flex.12xlarge	7	120
c7i-flex.16xlarge	14	120
c8g.medium	1	4
c8g.large	2	10
c8g.xlarge	3	20
c8g.2xlarge	3	40
c8g.4xlarge	7	60
c8g.8xlarge	7	60
c8g.12xlarge	7	60

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c8g.16xlarge	14	120
c8g.24xlarge	14	120
c8g.48xlarge	14	120
c8g.metal-24xl	14	120
c8g.metal-48xl	14	120
c8gd.medium	1	4
c8gd.large	2	10
c8gd.xlarge	3	20
c8gd.2xlarge	3	40
c8gd.4xlarge	7	60
c8gd.8xlarge	7	60
c8gd.12xlarge	7	60
c8gd.16xlarge	14	120
c8gd.24xlarge	14	120
c8gd.48xlarge	14	120
c8gd.metal-24xl	14	120
c8gd.metal-48xl	14	120
c8gn.medium	1	4
c8gn.large	2	10
c8gn.xlarge	3	20

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
c8gn.2xlarge	3	40
c8gn.4xlarge	7	60
c8gn.8xlarge	9	60
c8gn.12xlarge	11	60
c8gn.16xlarge	15	120
c8gn.24xlarge	23	120
c8gn.48xlarge	23	120
c8gn.metal-24xl	23	120
c8gn.metal-48xl	23	120

메모리 최적화

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
r5.large	2	10
r5.xlarge	3	20
r5.2xlarge	3	40
r5.4xlarge	7	60
r5.12xlarge	7	60
r5.16xlarge	14	120
r5.24xlarge	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
r5a.large	2	10
r5a.xlarge	3	20
r5a.2xlarge	3	40
r5a.4xlarge	7	60
r5a.8xlarge	7	60
r5a.12xlarge	7	60
r5a.16xlarge	14	120
r5a.24xlarge	14	120
r5ad.large	2	10
r5ad.xlarge	3	20
r5ad.2xlarge	3	40
r5ad.4xlarge	7	60
r5ad.8xlarge	7	60
r5ad.12xlarge	7	60
r5ad.16xlarge	14	120
r5ad.24xlarge	14	120
r5b.16xlarge	14	120
r5d.large	2	10
r5d.xlarge	3	20
r5d.2xlarge	3	40

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
r5d.4xlarge	7	60
r5d.8xlarge	7	60
r5d.12xlarge	7	60
r5d.16xlarge	14	120
r5d.24xlarge	14	120
r5dn.16xlarge	14	120
r6a.large	2	10
r6a.xlarge	3	20
r6a.2xlarge	3	40
r6a.4xlarge	7	60
r6a.8xlarge	7	90
r6a.12xlarge	7	120
r6a.16xlarge	14	120
r6a.24xlarge	14	120
r6a.32xlarge	14	120
r6a.48xlarge	14	120
r6a.metal	14	120
r6g.medium	1	4
r6g.large	2	10
r6g.xlarge	3	20

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
r6g.2xlarge	3	40
r6g.4xlarge	7	60
r6g.8xlarge	7	60
r6g.12xlarge	7	60
r6g.16xlarge	14	120
r6g.metal	14	120
r6gd.미디엄	1	4
r6gd.large	2	10
r6gd.xlarge	3	20
r6gd.2xlarge	3	40
r6gd.4xlarge	7	60
r6gd.8xlarge	7	60
r6gd.12xlarge	7	60
r6gd.16xlarge	14	120
r6gd.metal	14	120
r6i.large	2	10
r6i.xlarge	3	20
r6g.2xlarge	3	40
r6i.4xlarge	7	60
r6i.8xlarge	7	90

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
r6i.12xlarge	7	120
r6i.16xlarge	14	120
r6i.24xlarge	14	120
r6i.32xlarge	14	120
r6i.metal	14	120
r6idn.large	2	10
r6idn.xlarge	3	20
r6idn.2xlarge	3	40
r6idn.4xlarge	7	60
r6idn.8xlarge	7	90
r6idn.12xlarge	7	120
r6idn.16xlarge	14	120
r6idn.24xlarge	14	120
r6idn.32xlarge	15	120
r6idn.metal	15	120
r6in.large	2	10
r6in.xlarge	3	20
r6in.2xlarge	3	40
r6in.4xlarge	7	60
r6in.8xlarge	7	90

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
r6in.12xlarge	7	120
r6in.16xlarge	14	120
r6in.24xlarge	14	120
r6in.32xlarge	15	120
r6in.metal	15	120
r6id.large	2	10
r6id.xlarge	3	20
r6id.2xlarge	3	40
r6id.4xlarge	7	60
r6id.8xlarge	7	90
r6id.12xlarge	7	120
r6id.16xlarge	14	120
r6id.24xlarge	14	120
r6id.32xlarge	14	120
r6id.metal	14	120
r7a.medium	1	4
r7a.large	2	10
r7a.xlarge	3	20
r7a.2xlarge	3	40
r7a.4xlarge	7	60

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
r7a.8xlarge	7	90
r7a.12xlarge	7	120
r7a.16xlarge	14	120
r7a.24xlarge	14	120
r7a.32xlarge	14	120
r7a.48xlarge	14	120
r7a.metal-48xl	14	120
r7g.medium	1	4
r7g.large	2	10
r7g.xlarge	3	20
r7g.2xlarge	3	40
r7g.4xlarge	7	60
r7g.8xlarge	7	60
r7g.12xlarge	7	60
r7g.16xlarge	14	120
r7g.metal	14	120
r7gd.medium	1	4
r7gd.large	2	10
r7gd.xlarge	3	20
r7gd.2xlarge	3	40

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
r7gd.4xlarge	7	60
r7gd.8xlarge	7	60
r7gd.12xlarge	7	60
r7gd.16xlarge	14	120
r7gd.metal	14	120
r7i.large	2	10
r7i.xlarge	3	20
r7i.2xlarge	3	40
r7i.4xlarge	7	60
r7i.8xlarge	7	90
r7i.12xlarge	7	120
r7i.16xlarge	14	120
r7i.24xlarge	14	120
r7i.48xlarge	14	120
r7i.metal-24xl	14	120
r7i.metal-48xl	14	120
r7iz.large	2	10
r7iz.xlarge	3	20
r7iz.2xlarge	3	40
r7iz.4xlarge	7	60

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
r7iz.8xlarge	7	90
r7iz.12xlarge	7	120
r7iz.16xlarge	14	120
r7iz.32xlarge	14	120
r7iz.metal-16xl	14	120
r7iz.metal-32xl	14	120
r8g.medium	1	4
r8g.large	2	10
r8g.xlarge	3	20
r8g.2xlarge	3	40
r8g.4xlarge	7	60
r8g.8xlarge	7	60
r8g.12xlarge	7	60
r8g.16xlarge	14	120
r8g.24xlarge	14	120
r8g.48xlarge	14	120
r8g.metal-24xl	14	120
r8g.metal-48xl	14	120
r8gd.medium	1	4
r8gd.large	2	10

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
r8gd.xlarge	3	20
r8gd.2xlarge	3	40
r8gd.4xlarge	7	60
r8gd.8xlarge	7	60
r8gd.12xlarge	7	60
r8gd.16xlarge	14	120
r8gd.24xlarge	14	120
r8gd.48xlarge	14	120
r8gd.metal-24xl	14	120
r8gd.metal-48xl	14	120
u-3tb1.56xlarge	7	12
u-6tb1.56xlarge	14	12
u-18tb1.112xlarge	14	12
u-18tb1.metal	14	12
u-24tb1.112xlarge	14	12
u-24tb1.metal	14	12
u7i-6tb.112xlarge	14	120
u7i-8tb.112xlarge	14	120
u7i-12tb.224xlarge	14	120
u7in-16tb.224xlarge	15	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
u7in-24tb.224xlarge	15	120
u7in-32tb.224xlarge	15	120
u7inh-32tb.480xlarge	15	120
x2gd.medium	1	10
x2gd.large	2	10
x2gd.xlarge	3	20
x2gd.2xlarge	3	40
x2gd.4xlarge	7	60
x2gd.8xlarge	7	60
x2gd.12xlarge	7	60
x2gd.16xlarge	14	120
x2gd.metal	14	120
x2idn.16xlarge	14	120
x2idn.24xlarge	14	120
x2idn.32xlarge	14	120
x2idn.metal	14	120
x2iedn.xlarge	3	13
x2iedn.2xlarge	3	29
x2iedn.4xlarge	7	60
x2iedn.8xlarge	7	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
x2iedn.16xlarge	14	120
x2iedn.24xlarge	14	120
x2iedn.32xlarge	14	120
x2iedn.metal	14	120
x2iezn.2xlarge	3	64
x2iezn.4xlarge	7	120
x2iezn.6xlarge	7	120
x2iezn.8xlarge	7	120
x2iezn.12xlarge	14	120
x2iezn.metal	14	120
x8g.medium	1	4
x8g.large	2	10
x8g.xlarge	3	20
x8g.2xlarge	3	40
x8g.4xlarge	7	60
x8g.8xlarge	7	60
x8g.12xlarge	7	60
x8g.16xlarge	14	120
x8g.24xlarge	14	120
x8g.48xlarge	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
x8g.metal-24xl	14	120
x8g.metal-48xl	14	120

스토리지 최적화

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
i4g.large	2	10
i4g.xlarge	3	20
i4g.2xlarge	3	40
i4g.4xlarge	7	60
i4g.8xlarge	7	60
i4g.16xlarge	14	120
i4i.xlarge	3	8
i4i.2xlarge	3	28
i4i.4xlarge	7	58
i4i.8xlarge	7	118
i4i.12xlarge	7	118
i4i.16xlarge	14	248
i4i.24xlarge	14	118
i4i.32xlarge	14	498

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
i4i.metal	14	498
i7i.large	2	10
i7i.xlarge	3	20
i7i.2xlarge	3	40
i7i.4xlarge	7	60
i7i.8xlarge	7	90
i7i.12xlarge	7	90
i7i.16xlarge	14	120
i7i.24xlarge	14	120
i7i.48xlarge	14	120
i7i.metal-24xl	14	120
i7i.metal-48xl	14	120
i7ie.large	2	20
i7ie.xlarge	3	29
i7ie.2xlarge	3	29
i7ie.3xlarge	3	29
i7ie.6xlarge	7	60
i7ie.12xlarge	7	60
i7ie.18xlarge	14	120
i7ie.24xlarge	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
i7ie.48xlarge	14	120
i7ie.metal-24xl	14	120
i7ie.metal-48xl	14	120
i8g.large	2	10
i8g.xlarge	3	20
i8g.2xlarge	3	40
i8g.4xlarge	7	60
i8g.8xlarge	7	60
i8g.12xlarge	7	60
i8g.16xlarge	14	120
i8g.24xlarge	14	120
i8g.48xlarge	14	120
i8g.metal-24xl	14	120
im4gn.large	2	10
im4gn.xlarge	3	20
im4gn.2xlarge	3	40
im4gn.4xlarge	7	60
im4gn.8xlarge	7	60
im4gn.16xlarge	14	120
is4gen.medium	1	4

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
is4gen.large	2	10
is4gen.xlarge	3	20
4gn.2xlarge	3	40
is4gen.4xlarge	7	60
is4gen.8xlarge	7	60

액셀러레이티드 컴퓨팅

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
dl1.24xlarge	59	120
dl2q.24xlarge	14	120
f2.6xlarge	7	90
f2.12xlarge	7	120
f2.48xlarge	14	120
g4ad.xlarge	1	12
g4dn.2xlarge	1	12
g4ad.4xlarge	2	12
g4ad.8xlarge	3	12
g4ad.16xlarge	7	12
g5.xlarge	3	6

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
g5.2xlarge	3	19
g5.4xlarge	7	40
g5.8xlarge	7	90
g5.12xlarge	14	120
g5.16xlarge	7	120
g5.24xlarge	14	120
g5.48xlarge	6	120
g5g.xlarge	3	20
g5g.2xlarge	3	40
g5g.4xlarge	7	60
g5g.8xlarge	7	60
g5g.16xlarge	14	120
g5g.metal	14	120
g6.xlarge	3	20
g6.2xlarge	3	40
g6.4xlarge	7	60
g6.8xlarge	7	90
g6.12xlarge	7	120
g6.16xlarge	14	120
g6.24xlarge	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
g6.48xlarge	14	120
g6e.xlarge	3	20
g6e.2xlarge	3	40
g6e.4xlarge	7	60
g6e.8xlarge	7	90
g6e.12xlarge	9	120
g6e.16xlarge	14	120
g6e.24xlarge	19	120
g6e.48xlarge	39	120
g6f.large	1	10
g6f.xlarge	3	20
g6f.2xlarge	3	40
g6f.4xlarge	7	60
gr6.4xlarge	7	60
gr6.8xlarge	7	90
gr6f.4xlarge	7	60
inf2.xlarge	3	20
inf2.8xlarge	7	90
inf2.24xlarge	14	120
inf2.48xlarge	14	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
p4d.24xlarge	59	120
p4de.24xlarge	59	120
p5.48xlarge	63	242
p5e.48xlarge	63	242
p5en.48xlarge	63	242
p6-b200.48xlarge	31	242
p6e-gb200.36xlarge	38	120
trn1.2xlarge	3	19
trn1.32xlarge	39	120
trn1n.32xlarge	79	242
trn2.48xlarge	31	242
trn2u.48xlarge	31	242
vt1.3xlarge	3	40
vt1.6xlarge	7	60
vt1.24xlarge	14	120

고성능 컴퓨팅

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
hpc6a.48xlarge	1	120

인스턴스 유형	ENI 트렁킹이 없는 작업 제한	ENI 트렁킹을 포함하는 작업 제한
hpc6id.32xlarge	1	120
hpc7g.4xlarge	3	120
hpc7g.8xlarge	3	120
hpc7g.16xlarge	3	120

Amazon ECS Linux 컨테이너 인스턴스 메모리 예약

Amazon ECS 컨테이너 에이전트가 클러스터에 컨테이너 인스턴스를 등록할 때 에이전트는 컨테이너 인스턴스에서 태스크에 대해 예약할 수 있는 메모리 크기를 결정해야 합니다. 플랫폼 메모리 오버헤드와 운영 체제 커널이 차지하는 메모리로 인해 이 수치가 Amazon EC2 인스턴스에 대해 공급된 설치된 메모리 용량과 다르기 때문입니다 예를 들어, `m4.large` 인스턴스의 설치된 메모리는 8GiB입니다. 그러나 컨테이너 인스턴스 등록 시 항상 작업에 정확히 8,192MiB의 메모리를 사용할 수 있는 것은 아닙니다.

Amazon ECS 컨테이너 에이전트는 `ECS_RESERVED_MEMORY`라는 구성 변수를 제공합니다. 이를 사용하여 작업에 할당된 풀에서 지정된 메모리 크기(MiB)를 제거할 수 있습니다. 이를 통해 중요 시스템 프로세스에 대한 메모리를 효율적으로 예약합니다.

컨테이너 인스턴스의 모든 메모리를 작업에 사용하는 경우 작업 및 메모리의 중요 시스템 프로세스 경합으로 인해 시스템 오류가 발생할 수 있습니다.

예를 들어, 컨테이너 에이전트 구성 파일에서 `ECS_RESERVED_MEMORY=256`을 지정한 경우 에이전트는 인스턴스에 대해 총 메모리에서 256MiB를 제외하고 등록하며, 이 256MiB의 메모리는 ECS 작업에 할당될 수 없습니다. 에이전트 구성 변수 및 이를 설정하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성 및 데이터 전달을 위한 Amazon ECS Linux 컨테이너 인스턴스 부트스트래핑 섹션을 참조하세요.

작업에 대해 8,192MiB를 지정하고, 이 요구 사항을 충족하기 위해 8,192MiB 이상의 메모리를 사용할 수 있는 컨테이너 인스턴스가 없는 경우 작업은 클러스터에 배치될 수 없습니다. 관리형 컴퓨팅 환경을 사용 중인 경우 AWS Batch는 요청을 수용하기 위해 더 큰 인스턴스 유형을 시작해야 합니다.

또한, Amazon ECS 컨테이너 에이전트 및 컨테이너 인스턴스의 기타 핵심 시스템 프로세스에 대해 일부 메모리를 예약해야 합니다. 그래야 작업의 컨테이너가 동일한 메모리와 경합하지 않고, 그에 따라 시스템 오류가 발생하지 않을 수 있습니다.

Amazon ECS 컨테이너 에이전트는 Docker `ReadMemInfo()` 함수를 사용하여 운영 체제가 사용할 수 있는 전체 메모리를 쿼리합니다. Linux 및 Windows 모두 명령줄 유틸리티를 제공하여 총 메모리를 결정합니다.

Example - Linux 총 메모리 결정

free 명령은 운영 체제가 인식한 총 메모리를 반환합니다.

```
$ free -b
```

Amazon ECS 최적화된 Amazon Linux AMI를 실행하는 `m4.large` 인스턴스의 출력 예.

```
             total       used       free     shared    buffers     cached
Mem:    8373026816  348180480 8024846336      90112   25534464  205418496
-/+ buffers/cache:  117227520 8255799296
```

이 인스턴스의 총 메모리는 8,373,026,816바이트로, 이는 태스크에 대해 7,985MiB를 사용할 수 있음을 의미합니다.

Example - Windows 총 메모리 결정

wmic 명령은 운영 체제가 인식한 총 메모리를 반환합니다.

```
C:\> wmic ComputerSystem get TotalPhysicalMemory
```

Amazon ECS 최적화 Windows Server AMI를 실행하는 `m4.large` 인스턴스의 출력 예제.

```
TotalPhysicalMemory
8589524992
```

이 인스턴스의 총 메모리는 8,589,524,992바이트로, 이는 작업에 대해 8,191MiB를 사용할 수 있음을 의미합니다.

컨테이너 인스턴스 메모리 보기

컨테이너 인스턴스가 Amazon ECS 콘솔(또는 DescribeContainerInstances API 작업)에 등록하는 메모리 크기를 확인할 수 있습니다. 특정 인스턴스 유형에 대해 가능한 한 많은 메모리를 작업에 제공하

여 리소스 사용률을 극대화하려는 경우 해당 컨테이너 인스턴스에 사용 가능한 메모리를 관찰한 다음, 작업에 그만큼의 메모리를 할당할 수 있습니다.

컨테이너 인스턴스 메모리를 보는 방법

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택하고 컨테이너 인스턴스를 호스팅하는 클러스터를 선택합니다.
3. 인프라를 선택한 다음, 컨테이너 인스턴스에서 컨테이너 인스턴스를 선택합니다.
4. 리소스 섹션에서는 컨테이너 인스턴스에 대해 등록된 메모리와 사용 가능한 메모리를 보여줍니다.

   등록됨 메모리 값은 처음 시작할 때 Amazon ECS에 등록된 컨테이너 인스턴스의 값이며, 사용 가능 메모리 값은 작업에 할당되지 않은 값입니다.

AWS Systems Manager를 사용한 원격으로 Amazon ECS 컨테이너 인스턴스 관리

AWS Systems Manager(Systems Manager)의 Run Command 기능을 사용하면 Amazon ECS 컨테이너 인스턴스의 구성을 원격으로 안전하게 관리할 수 있습니다. Run Command를 사용하면 인스턴스에 로컬로 로그인하지 않고 간단하게 일반적인 관리 태스크를 수행할 수 있습니다. 여러 컨테이너 인스턴스에서 동시에 명령을 실행하여 클러스터의 구성 변경을 전체적으로 관리할 수 있습니다. Run Command는 각 명령의 상태와 결과를 보고합니다.

다음은 Run Command를 사용하여 수행할 수 있는 태스크 유형의 몇 가지 예입니다.

- 패키지 설치 또는 제거.
- 보안 업데이트 수행.
- Docker 이미지 정리.
- 서비스 중지 또는 시작.
- 시스템 리소스 보기.
- 로그 파일 보기.
- 파일 작업 수행.

Run Command에 대한 자세한 정보는 AWS Systems Manager 사용 설명서의 AWS Systems Manager Run Command를 참조하세요.

다음은 Amazon ECS에서 Systems Manager를 사용하기 위한 사전 조건입니다.

1. Systems Manager API에 액세스하려면 컨테이너 인스턴스 역할(ecsInstanceRole)에 권한을 부여해야 합니다. AmazonSSMManagedInstanceCore를 `ecsInstanceRole` 역할에 할당하면 됩니다. 역할에 정책을 연결하는 방법에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 역할 권한 업데이트를 참조하세요.
2. 컨테이너 인스턴스에 SSM Agent가 설치되었는지 확인합니다. 자세한 정보는 Linux용 EC2 인스턴스에 수동으로 SSM Agent 설치 및 제거를 참조하세요.

Systems Manager 관리형 정책을 AWS Systems Manager에 연결하고 `ecsInstanceRole` 에이전트(SSM Agent)가 컨테이너 인스턴스에 설치되어 있는지 확인한 후 Run Command를 사용하여 컨테이너 인스턴스로 명령 전송을 시작할 수 있습니다. 인스턴스에서 명령 및 shell 스크립트를 실행하고 결과 출력을 보는 방법에 대한 자세한 정보는 AWS Systems Manager 사용 설명서의 Systems Manager Run Command를 사용하여 명령 실행 및 Run Command 연습을 참조하세요.

일반적인 사용 사례는 실행 명령으로 컨테이너 인스턴스 소프트웨어를 업데이트하는 것입니다. 다음 파라미터를 사용하여 AWS Systems Manager 사용 설명서의 절차를 따를 수 있습니다.

파라미터	값
명령 문서	`AWS-RunShellScript`
명령	`$ yum update -y`
대상 인스턴스	사용자의 컨테이너 인스턴스

Amazon ECS Linux 컨테이너 인스턴스에 HTTP 프록시 사용

Amazon ECS 컨테이너 에이전트와 Docker 대몬 모두에 대해 HTTP 프록시를 사용하도록 Amazon ECS 컨테이너 인스턴스를 구성할 수 있습니다. 이 구성은 컨테이너 인스턴스가 Amazon VPC 인터넷 게이트웨이, NAT 게이트웨이 또는 인스턴스를 통해 외부 네트워크에 액세스하지 못할 경우 유용합니다.

HTTP 프록시를 사용하도록 Amazon ECS Linux 컨테이너 인스턴스를 구성하려면 시작 시(Amazon EC2 사용자 데이터를 사용해) 관련 파일에 있는 다음 변수를 설정합니다. 구성 파일을 수동으로 편집한 다음 에이전트를 재시작할 수도 있습니다.

`/etc/ecs/ecs.config`(Amazon Linux 2 및 Amazon Linux AMI)

`HTTP_PROXY=10.0.0.131:3128`

Amazon ECS 에이전트가 인터넷에 연결하는 데 사용할 HTTP 프록시의 호스트 이름(또는 IP 주소) 및 포트 번호로 이 값을 설정합니다. 예를 들어 컨테이너 인스턴스는 Amazon VPC 인터넷 게이트웨이, NAT 게이트웨이 또는 인스턴스를 통해 외부 네트워크에 액세스하지 못할 수 있습니다.

`NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock`

EC2 인스턴스 메타데이터, 작업에 대한 IAM 역할 및 프록시로부터의 Docker 대몬 트래픽을 필터링하려면 이 값을 `169.254.169.254,169.254.170.2,/var/run/docker.sock`으로 설정합니다.

`/etc/systemd/system/ecs.service.d/http-proxy.conf`(Amazon Linux 2 전용)

`Environment="HTTP_PROXY=10.0.0.131:3128/"`

`ecs-init`가 인터넷에 연결하는 데 사용할 HTTP 프록시의 호스트 이름(또는 IP 주소) 및 포트 번호로 이 값을 설정합니다. 예를 들어 컨테이너 인스턴스는 Amazon VPC 인터넷 게이트웨이, NAT 게이트웨이 또는 인스턴스를 통해 외부 네트워크에 액세스하지 못할 수 있습니다.

`Environment="NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock"`

EC2 인스턴스 메타데이터, 작업에 대한 IAM 역할 및 프록시로부터의 Docker 대몬 트래픽을 필터링하려면 이 값을 `169.254.169.254,169.254.170.2,/var/run/docker.sock`으로 설정합니다.

`/etc/init/ecs.override` (Amazon Linux AMI만 해당)

`env HTTP_PROXY=10.0.0.131:3128`

`ecs-init`가 인터넷에 연결하는 데 사용할 HTTP 프록시의 호스트 이름(또는 IP 주소) 및 포트 번호로 이 값을 설정합니다. 예를 들어 컨테이너 인스턴스는 Amazon VPC 인터넷 게이트웨이, NAT 게이트웨이 또는 인스턴스를 통해 외부 네트워크에 액세스하지 못할 수 있습니다.

`env NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock`

EC2 인스턴스 메타데이터, 작업에 대한 IAM 역할 및 프록시로부터의 Docker 대몬 트래픽을 필터링하려면 이 값을 `169.254.169.254,169.254.170.2,/var/run/docker.sock`으로 설정합니다.

`/etc/systemd/system/docker.service.d/http-proxy.conf`(Amazon Linux 2 전용)

`Environment="HTTP_PROXY=http://10.0.0.131:3128"`

Docker 대몬이 인터넷에 연결하는 데 사용할 HTTP 프록시의 호스트 이름(또는 IP 주소) 및 포트 번호로 이 값을 설정합니다. 예를 들어 컨테이너 인스턴스는 Amazon VPC 인터넷 게이트웨이, NAT 게이트웨이 또는 인스턴스를 통해 외부 네트워크에 액세스하지 못할 수 있습니다.

`Environment="NO_PROXY=169.254.169.254,169.254.170.2"`

프록시에서 EC2 인스턴스 메타데이터를 필터링하려면 이 값을 `169.254.169.254,169.254.170.2`로 설정합니다.

`/etc/sysconfig/docker` (Amazon Linux AMI 및 Amazon Linux 2 전용)

`export HTTP_PROXY=http://10.0.0.131:3128`

Docker 대몬이 인터넷에 연결하는 데 사용할 HTTP 프록시의 호스트 이름(또는 IP 주소) 및 포트 번호로 이 값을 설정합니다. 예를 들어 컨테이너 인스턴스는 Amazon VPC 인터넷 게이트웨이, NAT 게이트웨이 또는 인스턴스를 통해 외부 네트워크에 액세스하지 못할 수 있습니다.

`export NO_PROXY=169.254.169.254,169.254.170.2`

프록시에서 EC2 인스턴스 메타데이터를 필터링하려면 이 값을 `169.254.169.254,169.254.170.2`로 설정합니다.

이 환경 변수를 위 파일에서 설정하면 Amazon ECS 컨테이너 에이전트, `ecs-init` 및 Docker 대몬에만 영향을 미칩니다. 기타 서비스(예: yum)에서는 프록시를 사용하도록 구성하지 않습니다.

프록시를 구성하는 방법에 대한 자세한 내용은 Amazon Linux 2에서 Docker 및 Amazon ECS 컨테이너 에이전트에 대한 HTTP 프록시를 설정하려면 어떻게 해야 합니까?를 참조하세요.

Amazon ECS Auto Scaling 그룹에 대해 사전 초기화된 인스턴스 구성

Amazon ECS에서 Amazon EC2 Auto Scaling 웜 풀을 지원합니다. 웜 풀은 서비스에 배치할 준비가 되어 사전 초기화된 Amazon EC2 인스턴스의 그룹입니다. 애플리케이션을 확장해야 할 때마다 Amazon EC2 Auto Scaling은 콜드 인스턴스를 시작하는 대신 웜 풀에서 미리 초기화된 인스턴스를 사용하고 모든 최종 초기화 프로세스가 실행되도록 허용한 다음 인스턴스를 서비스에 배치합니다.

웜 풀 및 Auto Scaling에 웜 풀을 추가하는 방법에 대한 자세한 정보는 Amazon EC2 Auto Scaling 사용 설명서의 Amazon EC2 Auto Scaling의 웜 풀을 참조하세요.

Amazon ECS의 Auto Scaling 그룹에 대한 웜 풀을 생성하거나 업데이트할 경우, 스케일 인에서 웜 풀로 인스턴스를 반환하는 옵션을 설정할 수 없습니다(`ReuseOnScaleIn`). 자세한 내용은 AWS Command Line Interface 참조의 put-warm-pool을 참조하세요.

Amazon ECS 클러스터에서 웜 풀을 사용하려면 Amazon EC2 Auto Scaling 그룹 시작 템플릿의 사용자 데이터(User data) 필드에서 `ECS_WARM_POOLS_CHECK` 에이전트 구성 변수를 `true`로 설정합니다.

다음은 Amazon EC2 시작 템플릿의 사용자 데이터(User data) 필드에 에이전트 구성 변수를 지정하는 방법의 예를 보여줍니다. *`MyCluster`*를 사용자의 클러스터 이름으로 바꿉니다.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_WARM_POOLS_CHECK=true
EOF
```

`ECS_WARM_POOLS_CHECK` 변수는 에이전트 버전 `1.59.0` 이상에서만 지원됩니다. 변수에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성을 참조하세요.

Amazon ECS 컨테이너 에이전트 업데이트

때때로 버그 수정 및 새로운 기능을 적용하기 위해 Amazon ECS 컨테이너 에이전트를 업데이트해야 할 수도 있습니다. Amazon ECS 컨테이너 에이전트를 업데이트할 때 컨테이너 인스턴스의 실행 작업 또는 서비스가 중단되지 않습니다. 에이전트 업데이트 절차는 컨테이너 인스턴스가 Amazon ECS 최적화 AMI 또는 기타 운영 체제에서 시작되었는지에 따라 다릅니다.

> **Note**
>
> 에이전트 업데이트는 Windows 컨테이너 인스턴스에 적용되지 않습니다. 새로운 컨테이너 인스턴스를 시작하여 Windows 클러스터의 에이전트 버전을 업데이트하는 것을 권장합니다.

Amazon ECS 컨테이너 에이전트 버전 확인

컨테이너 인스턴스에서 실행 중인 컨테이너 에이전트의 버전을 확인하고 업데이트가 필요한지 판단할 수 있습니다. Amazon ECS 콘솔의 컨테이너 인스턴스 보기에 에이전트 버전이 표시됩니다. 에이전트 버전을 확인하려면 다음 절차를 사용합니다.

Amazon ECS console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.

2. 탐색 모음에서 외부 인스턴스가 등록되어 있는 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택하고 외부 인스턴스를 호스팅하는 클러스터를 선택합니다.
4. 클러스터: ***name***(Cluster : name) 페이지에서 인프라(Infrastructure) 탭을 선택합니다.
5. 컨테이너 인스턴스(Container instances)에서 컨테이너 인스턴스의 에이전트 버전(Agent version) 열을 확인합니다. 컨테이너 인스턴스에 최신 컨테이너 에이전트 버전이 포함되어 있지 않으면 콘솔이 메시지를 통해 경고하고 이전 에이전트 버전에 플래그를 지정합니다.

   에이전트 버전이 오래된 경우 다음 절차를 사용하여 컨테이너 에이전트를 업데이트할 수 있습니다.

   - 컨테이너 인스턴스가 Amazon ECS 최적화 AMI를 실행할 경우 Amazon ECS 최적화 AMI에서 Amazon ECS 컨테이너 에이전트 업데이트를 참조하세요.
   - 컨테이너 인스턴스가 Amazon ECS 최적화 AMI를 실행하지 않을 경우 Amazon ECS 컨테이너 에이전트를 수동으로 업데이트하는 방법(비 Amazon ECS 최적화 AMI)를 참조하세요.

   > **Important**
   >
   > Amazon ECS 최적화 AMI에서 v1.0.0 이전의 Amazon ECS 에이전트 버전을 업데이트하려면 현재 컨테이너 인스턴스를 종료하고 최신 AMI 버전을 사용하여 새 인스턴스를 시작하는 것이 좋습니다. 미리 보기 버전을 사용하는 컨테이너 인스턴스는 종료하고 최신 AMI로 대체해야 합니다. 자세한 정보는 Amazon ECS Linux 컨테이너 인스턴스 시작을 참조하세요.

Amazon ECS container agent introspection API

컨테이너 인스턴스 자체에서 에이전트 Amazon ECS 컨테이너 에이전트 내부 검사 API 버전을 확인하는 데 이를 사용할 수도 있습니다. 자세한 정보는 Amazon ECS 컨테이너 내부 검사을 참조하세요.

내부 검사 API를 사용하여 Amazon ECS 컨테이너 에이전트가 최신 버전을 실행 중인지 확인하는 방법

1. SSH를 통해 컨테이너 인스턴스에 로그인합니다.
2. 내부 검사 API를 쿼리합니다.

```
[ec2-user ~]$ curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
```

> **Note**
>
> 내부 검사 API는 Amazon ECS 컨테이너 에이전트의 버전 v1.0.0에서 `Version` 정보를 추가했습니다. 내부 검사 API를 쿼리했을 때 `Version`이 없는 경우, 또는 에이전트에 내부 검사 API 자체가 없는 경우 실행되는 버전은 v0.0.3 이하입니다. 버전을 업데이트해야 합니다.

Amazon ECS 최적화 AMI에서 Amazon ECS 컨테이너 에이전트 업데이트

Amazon ECS 최적화 AMI를 사용하는 경우, 최신 버전의 Amazon ECS 컨테이너 에이전트를 얻을 수 있는 옵션은 다음과 같이 여러 가지가 있습니다(권장 순서로 표시).

- 컨테이너 인스턴스를 종료하고 (수동으로 또는 Auto Scaling 시작 구성을 최신 AMI로 업데이트하여) 최신 버전의 Amazon ECS 최적화 Amazon Linux 2 AMI를 시작합니다. 그러면 테스트 및 검증된 최신 버전의 Amazon Linux, Docker, `ecs-init` 및 Amazon ECS 컨테이너 에이전트를 포함하는 컨테이너 인스턴스가 새로 시작됩니다. 자세한 정보는 Amazon ECS 최적화 Linux AMI을 참조하세요.
- SSH를 사용하여 인스턴스에 연결하고 `ecs-init` 패키지(및 해당 종속성)를 최신 버전으로 업데이트합니다. 이 작업은 Amazon Linux 리포지토리에서 사용 가능한 테스트 및 검증된 최신 버전의 도커 및 `ecs-init`와 최신 버전의 Amazon ECS 컨테이너 에이전트를 제공합니다. 자세한 정보는 Amazon ECS 최적화 AMI에서 `ecs-init` 패키지를 업데이트하는 방법을 참조하세요.
- 콘솔을 통해 또는 AWS CLI 또는 AWS SDK를 사용해 `UpdateContainerAgent` API 작업으로 컨테이너를 업데이트합니다. 자세한 정보는 `UpdateContainerAgent` API 태스크를 사용하여 Amazon ECS 컨테이너 에이전트 업데이트을 참조하세요.

> **Note**
>
> 에이전트 업데이트는 Windows 컨테이너 인스턴스에 적용되지 않습니다. 새로운 컨테이너 인스턴스를 시작하여 Windows 클러스터의 에이전트 버전을 업데이트하는 것을 권장합니다.

Amazon ECS 최적화 AMI에서 **`ecs-init`** 패키지를 업데이트하는 방법

1. SSH를 통해 컨테이너 인스턴스에 로그인합니다.
2. 다음 명령을 사용하여 `ecs-init` 패키지를 업데이트합니다.

```
sudo yum update -y ecs-init
```

> **Note**
>
> `ecs-init` 패키지와 Amazon ECS 컨테이너 에이전트가 즉시 업데이트됩니다. 그러나 새 버전의 Docker는 Docker 대몬이 다시 시작되기 전에는 로드되지 않습니다. 인스턴스를 재부팅하거나 인스턴스에서 다음 명령을 실행하여 다시 시작하세요.
>
> - Amazon ECS 최적화 Amazon Linux 2 AMI:
>
>   ```
>   sudo systemctl restart docker
>   ```
>
> - Amazon ECS 최적화 Amazon Linux AMI:
>
>   ```
>   sudo service docker restart && sudo start ecs
>   ```

`UpdateContainerAgent` API 태스크를 사용하여 Amazon ECS 컨테이너 에이전트 업데이트

> **Important**
>
> `UpdateContainerAgent` API는 Amazon ECS 최적화 AMI의 Linux 변형에서만 지원됩니다. 단, Amazon ECS 최적화 Amazon Linux 2(arm64) AMI만 예외입니다. Amazon ECS 최적화 Amazon Linux 2(arm64) AMI를 사용하는 컨테이너 인스턴스의 경우, `ecs-init` 패키지를 업데이트하여 에이전트를 업데이트하세요. 다른 운영 체제를 실행하는 컨테이너 인스턴스의 경우 Amazon ECS 컨테이너 에이전트를 수동으로 업데이트하는 방법(비 Amazon ECS 최적화 AMI) 섹션을 참조하세요. Windows 컨테이너 인스턴스를 사용하고 있을 경우 새로운 컨테이너 인스턴스를 시작하여 Windows 클러스터의 에이전트 버전을 업데이트하는 것을 권장합니다.

`UpdateContainerAgent` API 프로세스는 콘솔 또는 AWS CLI나 AWS SDK를 통해 에이전트 업데이트를 요청하면 시작됩니다. Amazon ECS는 현재 에이전트 버전을 사용 가능한 최신 에이전트 버전과 비교하여 업데이트가 가능한지 확인합니다. 업데이트가 가능하지 않으면(예: 이미 최신 버전이 실행 중) `NoUpdateAvailableException`이 반환됩니다.

위에 표시된 업데이트 프로세스의 단계는 다음과 같습니다.

`PENDING`

에이전트 업데이트가 가능하고, 업데이트 프로세스가 시작되었습니다.

`STAGING`

에이전트가 에이전트 업데이트를 다운로드하기 시작했습니다. 에이전트가 업데이트를 다운로드할 수 없는 경우 또는 업데이트의 내용이 부정확하거나 손상된 경우 에이전트가 실패 알림을 전송하고 업데이트가 `FAILED` 상태로 전환됩니다.

`STAGED`

에이전트 다운로드가 완료되었으며 에이전트 내용이 확인되었습니다.

`UPDATING`

`ecs-init` 서비스가 다시 시작되어 새 에이전트 버전을 선택합니다. 어떤 이유로 에이전트를 다시 시작할 수 없는 경우 업데이트가 `FAILED` 상태로 전환되고, 그렇지 않으면 에이전트가 Amazon ECS에게 업데이트 완료 신호를 보냅니다.

> **Note**
>
> 에이전트 업데이트는 Windows 컨테이너 인스턴스에 적용되지 않습니다. 새로운 컨테이너 인스턴스를 시작하여 Windows 클러스터의 에이전트 버전을 업데이트하는 것을 권장합니다.

콘솔에서 Amazon ECS 최적화 AMI에서 Amazon ECS 컨테이너 에이전트를 업데이트하는 방법

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 외부 인스턴스가 등록되어 있는 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택하고 클러스터를 선택합니다.
4. 클러스터: ***name***(Cluster : name) 페이지에서 인프라(Infrastructure) 탭을 선택합니다.
5. 컨테이너 인스턴스에서 업데이트할 인스턴스를 선택하고 작업, 에이전트 업데이트를 선택합니다.

Amazon ECS 컨테이너 에이전트를 수동으로 업데이트하는 방법(비 Amazon ECS 최적화 AMI)

때때로 버그 수정 및 새로운 기능을 적용하기 위해 Amazon ECS 컨테이너 에이전트를 업데이트해야 할 수도 있습니다. Amazon ECS 컨테이너 에이전트를 업데이트할 때 컨테이너 인스턴스의 실행 작업 또는 서비스가 중단되지 않습니다.

> **Note**
>
> 에이전트 업데이트는 Windows 컨테이너 인스턴스에 적용되지 않습니다. 새로운 컨테이너 인스턴스를 시작하여 Windows 클러스터의 에이전트 버전을 업데이트하는 것을 권장합니다.

1. SSH를 통해 컨테이너 인스턴스에 로그인합니다.
2. 에이전트가 ECS_DATADIR 환경 변수를 사용하여 해당 상태를 저장하는지 확인합니다.

   ```
   ubuntu:~$ docker inspect ecs-agent | grep ECS_DATADIR
   ```

   출력:

   ```
   "ECS_DATADIR=/data",
   ```

   > **Important**
   >
   > 이전 명령이 ECS_DATADIR 환경 변수를 반환하지 않으면 에이전트를 업데이트하기 전에 이 컨테이너 인스턴스에서 실행 중인 태스크를 모두 중지해야 합니다. ECS_DATADIR 환경 변수를 사용하는 새 버전의 에이전트는 해당 상태를 저장하며 태스크가 실행되는 동안 에이전트를 업데이트해도 문제가 발생하지 않습니다.

3. Amazon ECS 컨테이너 에이전트를 중지합니다.

   ```
   ubuntu:~$ docker stop ecs-agent
   ```

4. 에이전트 컨테이너를 삭제합니다.

   ```
   ubuntu:~$ docker rm ecs-agent
   ```

5. /etc/ecs 디렉터리와 Amazon ECS 컨테이너 에이전트 구성 파일이 /etc/ecs/ecs.config에 있는지 확인합니다.

   ```
   ubuntu:~$ sudo mkdir -p /etc/ecs && sudo touch /etc/ecs/ecs.config
   ```

6. /etc/ecs/ecs.config 파일을 편집하고 이 파일에 최소한 다음과 같은 변수 선언이 포함되어 있는지 확인합니다. 컨테이너 인스턴스가 기본 클러스터에 등록되지 않도록 하려면 클러스터 이름을 ECS_CLUSTER의 값으로 지정합니다.

```
ECS_DATADIR=/data
ECS_ENABLE_TASK_IAM_ROLE=true
ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true
ECS_LOGFILE=/log/ecs-agent.log
ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
ECS_LOGLEVEL=info
ECS_CLUSTER=default
```

이들을 비롯한 기타 에이전트 런타임 옵션에 대한 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

> **Note**
>
> 필요할 경우 (시작 시 Amazon EC2 사용자 데이터를 사용하여 컨테이너 인스턴스에 다운로드할 수 있는) Amazon S3에 에이전트 환경 변수를 저장할 수 있습니다. 프라이빗 리포지토리의 인증 자격 증명과 같은 민감한 정보에는 이 방법을 권장합니다. 자세한 내용은 Amazon S3에 Amazon ECS 컨테이너 인스턴스 구성 저장 및 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용 단원을 참조하세요.

7. Amazon Elastic Container Registry Public에서 최신 Amazon ECS 컨테이너 에이전트 이미지를 가져옵니다.

```
ubuntu:~$ docker pull public.ecr.aws/ecs/amazon-ecs-agent:latest
```

출력:

```
Pulling repository amazon/amazon-ecs-agent
a5a56a5e13dc: Download complete
511136ea3c5a: Download complete
9950b5d678a1: Download complete
c48ddcf21b63: Download complete
Status: Image is up to date for amazon/amazon-ecs-agent:latest
```

8. 컨테이너 인스턴스에서 최신 Amazon ECS 컨테이너 에이전트를 실행합니다.

> **Note**
>
> Docker 재시작 정책 또는 프로세스 관리자(예: upstart 또는 systemd)를 사용하여 컨테이너 에이전트를 서비스 또는 대몬으로 취급하고 종료 후에 다시 시작되도록 합니다. 이를

위해 Amazon ECS 최적화 AMI는 `ecs-init` RPM을 사용합니다. GitHub에서 이 RPM의 소스 코드를 확인할 수 있습니다.

다음의 에이전트 실행 명령 예제는 각 옵션을 표시하기 위해 여러 줄로 구분되어 있습니다. 이들을 비롯한 기타 에이전트 런타임 옵션에 대한 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

Important

운영 체제에서 SELinux를 사용하는 경우 docker run 명령에 `--privileged` 옵션이 필요합니다. 또한 SELinux 사용 컨테이너 인스턴스의 경우에는 `:Z` 옵션을 `/log` 및 `/data` 볼륨 마운트에 추가하는 것이 좋습니다. 그러나 명령을 실행하기 전에 이러한 볼륨에 대한 호스트 마운트가 존재해야 하며 그렇지 않으면 `no such file or directory` 오류가 발생합니다. SELinux 사용 컨테이너 인스턴스에서 Amazon ECS 에이전트를 실행하는 데 문제가 있는 경우 다음 조치를 취하세요.

- 컨테이너 인스턴스에서 호스트 볼륨 마운트 지점을 생성합니다.

```
ubuntu:~$ sudo mkdir -p /var/log/ecs /var/lib/ecs/data
```

- 아래의 docker run 명령에 `--privileged` 옵션을 추가합니다.
- 아래의 docker run 명령에 대한 `/log` 및 `/data` 컨테이너 볼륨 마운트(예: `--volume=/var/log/ecs/:/log:Z`)에 `:Z` 옵션을 추가합니다.

```
ubuntu:~$ sudo docker run --name ecs-agent \
--detach=true \
--restart=on-failure:10 \
--volume=/var/run:/var/run \
--volume=/var/log/ecs/:/log \
--volume=/var/lib/ecs/data:/data \
--volume=/etc/ecs:/etc/ecs \
--volume=/etc/ecs:/etc/ecs/pki \
--net=host \
--env-file=/etc/ecs/ecs.config \
amazon/amazon-ecs-agent:latest
```

> **Note**
>
> `Error response from daemon: Cannot start container` 메시지가 표시되는 경우 sudo docker rm ecs-agent 명령을 사용하여 실패한 컨테이너를 삭제하고 에이전트를 다시 실행해볼 수 있습니다.

Amazon ECS 최적화 Windows AMI

Amazon ECS 최적화 AMI는 Amazon ECS 워크로드를 실행하는 데 필요한 필수 구성 요소로 사전 구성됩니다. Amazon ECS에서 컨테이너화된 워크로드를 실행하는 데 필요한 기본 사양을 충족하도록 자체 컨테이너 인스턴스 AMI를 만들 수도 있지만 Amazon ECS 최적화 AMI는 AWS 엔지니어들에 의해 Amazon ECS에서 사전 구성 및 테스트됩니다. AWS에서 신속하게 시작하고 컨테이너를 실행할 수 있는 가장 간단한 방법입니다.

각 변형에 대한 AMI 이름, Amazon ECS 컨테이너 에이전트 버전 및 Docker 버전이 포함된 Amazon ECS 런타임 버전을 포함한 Amazon ECS 최적화 AMI 메타데이터는 프로그래밍 방식으로 검색할 수 있습니다. 자세한 내용은 the section called “Amazon ECS 최적화 Windows AMI 메타데이터 검색” 섹션을 참조하세요.

> **Important**
>
> 2022년 8월 이후에 출시된 모든 ECS 최적화 AMI 버전은 Docker EE(Mirantis)에서 Docker CE(Moby 프로젝트)로 마이그레이션됩니다.
> 고객들이 항상 최신 보안 업데이트를 갖출 수 있도록 하기 위해 Amazon ECS는 최소 세 개의 Windows Amazon ECS 최적화 AMI를 유지합니다. 새로운 Windows Amazon ECS 최적화 AMI를 출시한 후, Amazon ECS는 이전 프라이빗인 Windows Amazon ECS 최적화 AMI를 만듭니다. 액세스해야 하는 프라이빗 AMI가 있는 경우 Cloud Support를 통해 티켓을 제출하여 알려주세요.

Amazon ECS 최적화 AMI 변형

Amazon EC2 인스턴스에 사용할 수 있는 Amazon ECS 최적화 AMI의 Windows Server 변형은 다음과 같습니다.

> **Important**
>
> 8월 이후에 출시된 모든 ECS 최적화 AMI 버전은 Docker EE(Mirantis)에서 Docker CE(Moby 프로젝트)로 마이그레이션됩니다.

- Amazon ECS 최적화 Windows Server 2025 Full AMI
- Amazon ECS 최적화 Windows Server 2025 Core AMI
- Amazon ECS 최적화 Windows Server 2022 Full AMI
- Amazon ECS 최적화 Windows Server 2022 Core AMI
- Amazon ECS 최적화 Windows Server 2019 Full AMI
- Amazon ECS 최적화 Windows Server 2019 Core AMI
- Amazon ECS 최적화 Windows Server 2016 Full AMI

> **Important**
>
> Windows Server 2016은 최신 Docker 버전(예: 25.x.x)을 지원하지 않습니다. 따라서 Windows Server 2016 Full AMI는 Docker 런타임에 대한 보안 또는 버그 패치를 수신하지 않습니다. 다음 Windows 플랫폼 중 하나를 사용하여 이전하는 것이 좋습니다.
>
> - Windows Server 2022 Full
> - Windows Server 2022 Core
> - Windows Server 2019 Full
> - Windows Server 2019 Core

2022년 8월 9일에 Amazon ECS 최적화 Windows Server 20H2 Core AMI가 지원 종료 날짜에 도달했습니다. 이 AMI의 새 버전은 릴리스되지 않습니다. 자세한 정보는 Windows Server 릴리스 정보를 참조하세요.

Windows Server 2025, Windows Server 2022, Windows Server 2019, Windows Server 2016은 장기 서비스 채널(LTSC) 릴리스입니다. Windows Server 20H2는 반기 채널(SAC) 릴리스입니다. 자세한 정보는 Windows Server 릴리스 정보를 참조하세요.

고려 사항

다음은 Amazon EC2 Windows 컨테이너와 Amazon ECS에 관해 알아 두어야 할 몇 가지 사항입니다.

- Windows 컨테이너는 Linux 컨테이너 인스턴스에서 실행할 수 없으며 그 반대도 그렇습니다. Windows 및 Linux 태스크를 더욱 적절히 배치하려면 Windows 컨테이너 인스턴스와 Linux 컨테이너 인스턴스를 별도의 클러스터에 두고 Windows 클러스터에는 Windows 작업만 배치합니다. `memberOf(ecs.os-type=='windows')` 배치 제약을 설정하여 Windows 인스턴스에만 Windows 태스크 정의가 배치되어 있는지 확인할 수 있습니다.
- Windows 컨테이너는 EC2 및 Fargate 시작 유형을 사용하는 태스크에 대해 지원됩니다.
- Windows 컨테이너와 컨테이너 인스턴스는 Linux 컨테이너 및 컨테이너 인스턴스에서 사용할 수 있는 태스크 정의 파라미터를 모두 지원할 수 없습니다. 일부 파라미터는 전혀 지원되지 않고 Windows에서 Linux에서와 다르게 동작하는 파라미터도 있습니다. 자세한 내용은 Windows를 실행하는 EC2 인스턴스에 대한 Amazon ECS 작업 정의 차이 섹션을 참조하세요.
- 작업에 대한 IAM 역할 기능의 경우, 시작 시 해당 기능을 허용하도록 Windows 컨테이너 인스턴스를 구성해야 합니다. 컨테이너는 기능을 사용할 때 제공된 PowerShell 코드를 실행해야 합니다. 자세한 내용은 Amazon EC2 Windows 인스턴스 추가 구성 섹션을 참조하세요.
- 작업에 대한 IAM 역할 기능은 자격 증명 프록시를 사용하여 컨테이너에 자격 증명을 제공합니다. 이 자격 증명 프록시는 컨테이너 인스턴스에서 포트 80을 점유하므로 태스크를 위한 IAM 역할을 사용하는 경우, 해당 작업에 포트 80을 사용할 수 없습니다. 웹 서비스 컨테이너의 경우, Application Load Balancer 및 동적 포트 매핑을 사용하여 컨테이너에 표준 HTTP 포트 80 연결을 제공할 수 있습니다. 자세한 내용은 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산 섹션을 참조하세요.
- Windows 서버 Docker 이미지는 큽니다(9GiB). 따라서 Windows 컨테이너 인스턴스에는 Linux 컨테이너 인스턴스보다 더 많은 스토리지 공간이 필요합니다.
- Windows Server에서 Windows 컨테이너를 실행하려면 컨테이너와 호스트의 기본 이미지 OS 버전이 일치해야 합니다. 자세한 내용은 Microsoft 설명서 웹 사이트의 Windows 컨테이너 버전 호환성을 참조하세요. 클러스터에서 여러 Windows 버전을 실행하는 경우 배치 제약 조건 `memberOf(attribute:ecs.os-family == WINDOWS_SERVER_<OS_Release>_<FULL or CORE>)`를 사용하여 동일한 버전에서 실행되는 EC2 인스턴스에 작업이 배치되도록 할 수 있습니다. 자세한 내용은 the section called “Amazon ECS 최적화 Windows AMI 메타데이터 검색” 섹션을 참조하세요.

Amazon ECS 최적화 Windows AMI 메타데이터 검색

AMI ID, 이미지 이름, 운영 체제, 컨테이너 에이전트 버전 및 Amazon ECS 최적화 AMI의 각 변형에 대한 런타임 버전은 Systems Manager Parameter Store API를 쿼리함으로써 프로그래밍 방식으로 가

져올 수 있습니다. Systems Manager Parameter Store API에 대한 자세한 정보는 GetParameters 및 GetParametersByPath를 참조하세요.

> **Note**
>
> Amazon ECS 최적화 AMI 메타데이터를 검색하려면 관리 사용자에게 다음과 같은 IAM 권한이 있어야 합니다. 이러한 권한은 `AmazonECS_FullAccess` IAM 정책에 추가되었습니다.
>
> - ssm:GetParameters
> - ssm:GetParameter
> - ssm:GetParametersByPath

Systems Manager Parameter Store 파라미터 형식

> **Note**
>
> 다음 Systems Manager Parameter Store API 파라미터는 더 이상 사용되지 않으며 최신 Windows AMI를 검색하는 데 사용해서는 안 됩니다.
>
> - `/aws/service/ecs/optimized-ami/windows_server/2016/english/full/recommended/image_id`
> - `/aws/service/ecs/optimized-ami/windows_server/2019/english/full/recommended/image_id`

다음은 각 Amazon ECS 최적화 AMI 변형에 대한 파라미터 이름의 형식입니다.

- Windows Server 2025 Full AMI 메타데이터:

```
/aws/service/ami-windows-latest/Windows_Server-2025-English-Full-ECS_Optimized
```

- Windows Server 2025 Core AMI 메타데이터:

```
/aws/service/ami-windows-latest/Windows_Server-2025-English-Core-ECS_Optimized
```

- Windows Server 2022 Full AMI 메타데이터:

```
/aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

- Windows Server 2022 Core AMI 메타데이터:

```
/aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

- Windows Server 2019 Full AMI 메타데이터:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

- Windows Server 2019 Core AMI 메타데이터:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

- Windows Server 2016 Full AMI 메타데이터:

```
/aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

다음 파라미터 이름 형식은 안정적인 최신 Windows Server 2019 Full AMI의 메타데이터를 검색합니다.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-
English-Full-ECS_Optimized
```

다음은 파라미터 값에 대해 반환된 JSON 객체의 예입니다.

```
{
    "Parameters": [
        {
            "Name": "/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-
ECS_Optimized",
            "Type": "String",
            "Value": "{\"image_name\":\"Windows_Server-2019-English-Full-
ECS_Optimized-2023.06.13\",\"image_id\":\"ami-0debc1fb48e4aee16\",\"ecs_runtime_version
\":\"Docker (CE) version 20.10.21\",\"ecs_agent_version\":\"1.72.0\"}",
            "Version": 58,
            "LastModifiedDate": "2023-06-22T19:37:37.841000-04:00",
            "ARN": "arn:aws:ssm:us-east-1::parameter/aws/service/ami-windows-latest/
Windows_Server-2019-English-Full-ECS_Optimized",
            "DataType": "text"
        }
    ],
```

```
    "InvalidParameters": []
}
```

위 출력의 각 필드를 하위 파라미터 쿼리에 사용할 수 있습니다. 선택한 AMI 경로에 하위 파라미터 이름을 추가해 하위 파라미터에 대한 파라미터 경로를 구성합니다. 다음 하위 파라미터를 사용할 수 있습니다.

- `schema_version`
- `image_id`
- `image_name`
- `os`
- `ecs_agent_version`
- `ecs_runtime_version`

예시

다음 예에서는 Amazon ECS 최적화 AMI 변형에 대한 메타데이터를 검색할 수 있는 방법을 보여 줍니다.

안정적인 최신 Amazon ECS 최적화 AMI의 메타데이터 검색

AWS CLI와 다음 AWS CLI 명령을 사용하여 안정적인 최신 Amazon ECS 최적화 AMI를 가져올 수 있습니다.

- Amazon ECS 최적화 Windows Server 2025 Full AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Full-ECS_Optimized --region us-east-1
```

- Amazon ECS 최적화 Windows Server 2025 Core AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Core-ECS_Optimized --region us-east-1
```

- Amazon ECS 최적화 Windows Server 2022 Full AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized --region us-east-1
```

- Amazon ECS 최적화 Windows Server 2022 Core AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized --region us-east-1
```

- Amazon ECS 최적화 Windows Server 2019 Full AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized --region us-east-1
```

- Amazon ECS 최적화 Windows Server 2019 Core AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized --region us-east-1
```

- Amazon ECS 최적화 Windows Server 2016 Full AMI의 경우:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized --region us-east-1
```

AWS CloudFormation 템플릿에서 최신 권장 Amazon ECS 최적화 AMI 사용

Systems Manager 파라미터 스토어 이름을 참조하여 AWS CloudFormation 템플릿에서 최신 Amazon ECS 최적화 AMI를 참조할 수 있습니다.

```
Parameters:
  LatestECSOptimizedAMI:
    Description: AMI ID
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized/image_id
```

Amazon ECS 최적화 Windows AMI 버전

Amazon ECS 최적화 AMI의 현재 및 이전 버전과 이에 상응하는 Amazon ECS 컨테이너 에이전트, Docker 및 `ecs-init` 패키지의 버전을 확인합니다.

AMI ID를 포함하여 각 변형에 대한 Amazon ECS 최적화 AMI 메타데이터를 프로그래밍 방식으로 가져올 수 있습니다. 자세한 내용은 the section called “Amazon ECS 최적화 Windows AMI 메타데이터 검색” 섹션을 참조하세요.

다음 탭에는 Windows Amazon ECS 최적화 AMI 버전 목록이 표시됩니다. AWS CloudFormation 템플릿에서 Systems Manager Parameter Store 파라미터를 참조하는 방법에 대한 자세한 정보는 대한 자세한 정보는 AWS CloudFormation 템플릿에서 최신 권장 Amazon ECS 최적화 AMI 사용 섹션을 참조하세요.

Important

고객들이 항상 최신 보안 업데이트를 갖출 수 있도록 하기 위해 Amazon ECS는 최소 세 개의 Windows Amazon ECS 최적화 AMI를 유지합니다. 새로운 Windows Amazon ECS 최적화 AMI를 출시한 후, Amazon ECS는 이전 프라이빗인 Windows Amazon ECS 최적화 AMI를 만듭니다. 액세스해야 하는 프라이빗 AMI가 있는 경우 Cloud Support를 통해 티켓을 제출하여 알려주세요.

Windows Server 2016은 최신 Docker 버전(예: 25.x.x)을 지원하지 않습니다. 따라서 Windows Server 2016 Full AMI는 Docker 런타임에 대한 보안 또는 버그 패치를 수신하지 않습니다. 다음 Windows 플랫폼 중 하나를 사용하여 이전하는 것이 좋습니다.

- Windows Server 2022 Full
- Windows Server 2022 Core
- Windows Server 2019 Full
- Windows Server 2019 Core

Windows Server 2025 Full AMI versions

아래 테이블에는 Amazon ECS 최적화 Windows Server 2025 Full AMI의 현재 및 이전 버전과 이에 상응하는 Amazon ECS 컨테이너 에이전트 및 Docker의 버전이 나열되어 있습니다.

Amazon ECS 최적화 Windows Server 2025 Full AMI	Amazon ECS 컨테이너 에이전트 버전	Docker 버전	표시 여부
Windows_Server-2025-English-Full-ECS_Optimized-2025.06.13	`1.94.0`	`25.0.6 (Docker CE)`	퍼블릭

다음 AWS CLI 명령을 사용하여 현재 Amazon ECS 최적화 Windows Server 2025 Full AMI를 검색합니다.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Full-ECS_Optimized
```

Windows Server 2025 Core AMI versions

아래 테이블에는 Amazon ECS 최적화 Windows Server 2025 Core AMI의 현재 및 이전 버전과 이에 상응하는 Amazon ECS 컨테이너 에이전트 및 Docker의 버전이 나열되어 있습니다.

Amazon ECS 최적화 Windows Server 2025 Core AMI	Amazon ECS 컨테이너 에이전트 버전	Docker 버전	표시 여부
Windows_Server-2025-English-Core-ECS_Optimized-2025.06.13	1.94.0	25.0.6 (Docker CE)	퍼블릭

다음 AWS CLI 명령을 사용하여 현재 Amazon ECS 최적화 Windows Server 2025 Core AMI를 검색합니다.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Core-ECS_Optimized
```

Windows Server 2022 Full AMI versions

아래 표에는 Amazon ECS 최적화 Windows Server 2022 Full AMI의 현재 및 이전 버전과 이에 상응하는 Amazon ECS 컨테이너 에이전트 및 Docker의 버전이 나열되어 있습니다.

Amazon ECS 최적화 Windows Server 2022 Full AMI	Amazon ECS 컨테이너 에이전트 버전	Docker 버전	표시 여부
Windows_Server-2022-English-Full-ECS	1.94.0	25.0.6 (Docker CE)	퍼블릭

Amazon ECS 최적화 Windows Server 2022 Full AMI	Amazon ECS 컨테이너 에이전트 버전	Docker 버전	표시 여부
_Optimized-2025.06.13			
Windows_Server-2022-English-Full-ECS_Optimized-2025.05.17	1.93.1	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2022-English-Full-ECS_Optimized-2025.4.14	1.91.2	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2022-English-Full-ECS_Optimized-2025.3.17	1.91.0	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2022-English-Full-ECS_Optimized-2025.2.14	1.90.0	25.0.6 (Docker CE)	퍼블릭

다음 AWS CLI 명령을 사용해 현재 Amazon ECS 최적화 Windows Server 2022 Full AMI를 검색합니다.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

Windows Server 2022 Core AMI versions

아래 표에는 Amazon ECS 최적화 Windows Server 2022 Core AMI의 현재 및 이전 버전과 이에 상응하는 Amazon ECS 컨테이너 에이전트 및 Docker의 버전이 나열되어 있습니다.

Amazon ECS 최적화 Windows Server 2022 Core AMI	Amazon ECS 컨테이너 에이전트 버전	Docker 버전	표시 여부
Windows_Server-2022-English-Core-ECS_Optimized-2025.06.13	1.94.0	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2022-English-Core-ECS_Optimized-2025.05.17	1.93.1	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2022-English-Core-ECS_Optimized-2025.4.14	1.91.2	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2022-English-Core-ECS_Optimized-2025.3.17	1.91.0	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2022-English-Core-ECS_Optimized-2025.2.14	1.90.0	25.0.6 (Docker CE)	퍼블릭

다음 AWS CLI 명령을 사용해 현재 Amazon ECS 최적화 Windows Server 2022 Full AMI를 검색합니다.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

Windows Server 2019 Full AMI versions

아래 표에는 Amazon ECS 최적화 Windows Server 2019 Full AMI의 현재 및 이전 버전과 이에 상응하는 Amazon ECS 컨테이너 에이전트 및 Docker의 버전이 나열되어 있습니다.

Amazon ECS 최적화 Windows Server 2019 Full AMI	Amazon ECS 컨테이너 에이전트 버전	Docker 버전	표시 여부
Windows_Server-2019-English-Full-ECS_Optimized-2025.06.13	1.94.0	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2019-English-Full-ECS_Optimized-2025.05.17	1.93.1	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2019-English-Full-ECS_Optimized-2025.4.14	1.91.2	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2019-English-Full-ECS_Optimized-2025.3.17	1.91.0	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2019-English-Full-ECS_Optimized-2025.2.14	1.90.0	25.0.6 (Docker CE)	퍼블릭

다음 AWS CLI 명령을 사용해 현재 Amazon ECS 최적화 Windows Server 2019 Full AMI를 검색합니다.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

Windows Server 2019 Core AMI versions

아래 표에는 Amazon ECS 최적화 Windows Server 2019 Core AMI의 현재 및 이전 버전과 이에 상응하는 Amazon ECS 컨테이너 에이전트 및 Docker의 버전이 나열되어 있습니다.

Amazon ECS 최적화 Windows Server 2019 Core AMI	Amazon ECS 컨테이너 에이전트 버전	Docker 버전	표시 여부
Windows_Server-2019-English-Core-ECS_Optimized-2025.06.13	1.94.0	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2019-English-Core-ECS_Optimized-2025.05.17	1.93.1	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2019-English-Core-ECS_Optimized-2025.4.14	1.91.2	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2019-English-Core-ECS_Optimized-2025.3.17	1.91.0	25.0.6 (Docker CE)	퍼블릭
Windows_Server-2019-English-Core-ECS_Optimized-2025.2.14	1.90.0	25.0.6 (Docker CE)	퍼블릭

다음 AWS CLI 명령을 사용해 현재 Amazon ECS 최적화 Windows Server 2019 Full AMI를 검색합니다.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

Windows Server 2016 Full AMI versions

> **Important**
>
> Windows Server 2016은 최신 Docker 버전(예: 25.x.x)을 지원하지 않습니다. 따라서 Windows Server 2016 Full AMI는 Docker 런타임에 대한 보안 또는 버그 패치를 수신하지 않습니다. 다음 Windows 플랫폼 중 하나를 사용하여 이전하는 것이 좋습니다.
>
> - Windows Server 2022 Full
> - Windows Server 2022 Core
> - Windows Server 2019 Full
> - Windows Server 2019 Core

아래 표에는 Amazon ECS 최적화 Windows Server 2016 Full AMI의 현재 및 이전 버전과 이에 상응하는 Amazon ECS 컨테이너 에이전트 및 Docker의 버전이 나열되어 있습니다.

Amazon ECS 최적화 Windows Server 2016 Full AMI	Amazon ECS 컨테이너 에이전트 버전	Docker 버전	표시 여부
Windows_Server-2016-English-Full-ECS_Optimized-2025.06.13	1.94.0	20.10.23 (Docker CE)	퍼블릭
Windows_Server-2016-English-Full-ECS_Optimized-2025.05.17	1.93.1	20.10.23 (Docker CE)	퍼블릭

Amazon ECS 최적화 Windows Server 2016 Full AMI	Amazon ECS 컨테이너 에이전트 버전	Docker 버전	표시 여부
Windows_Server-2016-English-Full-ECS_Optimized-2025.4.14	1.91.2	20.10.23 (Docker CE)	퍼블릭
Windows_Server-2016-English-Full-ECS_Optimized-2025.3.17	1.91.0	20.10.23 (Docker CE)	퍼블릭
Windows_Server-2016-English-Full-ECS_Optimized-2025.2.14	1.90.0	20.10.23 (Docker CE)	퍼블릭

다음 AWS CLI Amazon ECS 최적화 Windows Server 2016 Full AMI를 사용합니다.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

고유 Amazon ECS 최적화 Windows AMI 구축

EC2 Image Builder를 사용하여 사용자 지정 Amazon ECS 최적화 Windows AMI를 구축합니다. 이를 통해 Amazon ECS에서 자체 라이선스로 Windows AMI를 쉽게 사용할 수 있습니다. Amazon ECS는 컨테이너를 호스트하기 위해 Windows 인스턴스를 실행하는 데 필요한 시스템 구성을 제공하는 관리형 Image Builder 구성 요소를 제공합니다. 각 Amazon ECS 관리형 구성 요소에는 특정 컨테이너 에이전트 및 Docker 버전이 포함됩니다. 최신 Amazon ECS 관리형 구성 요소를 사용하도록 이미지를 사용자 지정하거나 이전 컨테이너 에이전트 또는 Docker 버전이 필요한 경우 다른 구성 요소를 지정할 수 있습니다.

EC2 Image Builder를 사용하는 전체 연습은 EC2 Image Builder 사용 설명서의 EC2 Image Builder 시작하기를 참조하세요.

EC2 Image Builder를 사용하여 고유의 Amazon ECS 최적화 Windows AMI를 구축할 때 이미지 레시피를 생성합니다. 이미지 레시피는 다음 요구 사항을 충족해야 합니다.

- 소스 이미지는 Windows Server 2019 Core, Windows Server 2019 Full, Windows Server 2022 Core 또는 Windows Server 2022 Full을 기반으로 해야 합니다. 다른 Windows 운영 체제는 지원되지 않으며 구성 요소와 호환되지 않을 수 있습니다.
- 빌드 구성 요소를 지정할 때 `ecs-optimized-ami-windows` 구성 요소가 필요합니다. 이미지에 최신 보안 업데이트가 포함되도록 `update-windows` 구성 요소를 사용하는 것이 좋습니다.

  다른 구성 요소 버전을 지정하려면 버전 관리 옵션(Versioning options) 메뉴를 선택하고 사용할 구성 요소 버전을 지정합니다. 자세한 정보는 `ecs-optimized-ami-windows` 구성 요소 버전 나열 섹션을 참조하세요.

`ecs-optimized-ami-windows` 구성 요소 버전 나열

EC2 Image Builder 레시피를 생성하고 `ecs-optimized-ami-windows` 구성 요소를 지정할 때 기본 옵션을 사용하거나 특정 구성 요소 버전을 지정할 수 있습니다. Amazon ECS 컨테이너 에이전트 및 구성 요소 내에 포함된 Docker 버전과 함께 사용 가능한 구성 요소 버전을 확인하기 위해 AWS Management Console을 사용할 수 있습니다.

사용 가능한 **`ecs-optimized-ami-windows`** 구성 요소 버전을 나열하려면

1. https://console.aws.amazon.com/imagebuilder/에서 EC2 Image Builder 콘솔을 엽니다.
2. 탐색 모음에서 이미지를 구축하고 있는 리전을 선택합니다.
3. 탐색 창의 저장된 구성(Saved configurations) 메뉴에서 구성 요소(Components)를 선택합니다.
4. 구성 요소 페이지의 검색 창에 `ecs-optimized-ami-windows`를 입력하고 검증 메뉴를 아래로 당겨 빠른 시작(아마존 관리형)(Quick start (Amazon-managed))을 선택합니다.
5. 설명(Description) 열에서 Amazon ECS 컨테이너 에이전트와 이미지에 필요한 Docker 버전을 사용하는 구성 요소 버전을 확인할 수 있습니다.

Amazon ECS Windows 컨테이너 인스턴스 관리

Amazon ECS 워크로드에 EC2 인스턴스를 사용하는 경우 인스턴스를 유지 관리할 책임은 사용자에게 있습니다.

에이전트 업데이트는 Windows 컨테이너 인스턴스에 적용되지 않습니다. 새로운 컨테이너 인스턴스를 시작하여 Windows 클러스터의 에이전트 버전을 업데이트하는 것을 권장합니다.

관리 절차

- Amazon ECS Windows 컨테이너 인스턴스 시작
- 데이터 전달을 위한 Amazon ECS Windows 컨테이너 인스턴스 부트스트래핑
- Amazon ECS Windows 컨테이너 인스턴스에 HTTP 프록시 사용
- 스팟 인스턴스 알림을 수신하도록 Amazon ECS Windows 컨테이너 인스턴스 구성

Amazon ECS Windows 컨테이너 인스턴스 시작

Amazon ECS 컨테이너 인스턴스는 Amazon EC2 콘솔을 사용하여 생성됩니다. 시작하기 전에 먼저 Amazon ECS 사용 설정의 단계를 완료해야 합니다.

시작 마법사에 대한 자세한 내용은 Amazon EC2 사용 설명서의 새 인스턴스 시작 마법사를 사용하여 인스턴스 시작을 참조하세요.

새 Amazon EC2 마법사를 사용하여 인스턴스를 시작할 수 있습니다. 파라미터에 다음 목록을 사용할 수 있으며 파라미터는 기본값으로 나열되지 않은 상태로 둘 수 있습니다. 다음 지침은 각 파라미터 그룹을 안내합니다.

절차

시작하기 전에 Amazon ECS 사용 설정의 단계를 완료해야 합니다.

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 화면 상단의 탐색 모음에는 현재 AWS 리전이 표시됩니다(예: 미국 동부(오하이오)). 인스턴스를 시작할 리전을 선택합니다. 일부 Amazon EC2 리소스는 리전 간에 공유될 수 있지만 그렇지 않은 리소스도 있으므로 잘 선택해야 합니다.
3. Amazon EC2 콘솔 대시보드에서 인스턴스 시작을 선택합니다.

이름 및 태그

인스턴스 이름은 태그이며, 여기서 키는 이름이고 값은 사용자가 지정하는 이름입니다. 인스턴스, 볼륨 및 Elastic Graphics에 태그를 지정할 수 있습니다. 스팟 인스턴스의 경우 스팟 인스턴스 요청만 태깅할 수 있습니다.

인스턴스 이름과 추가 태그를 지정하는 것은 선택 사항입니다.

- 이름(Name)에 인스턴스를 설명하는 이름을 입력합니다. 이름을 지정하지 않으면 인스턴스를 시작할 때 자동으로 생성되는 ID로 인스턴스를 식별할 수 있습니다.

- 태그를 추가하려면 추가 태그 추가(Add additional tags)를 선택합니다. 태그 추가(Add tag)를 선택한 다음 키와 값을 입력하고 태그를 지정할 리소스 유형을 선택합니다. 추가할 각 추가 태그에 대해 태그 추가(Add tag)를 다시 선택합니다.

애플리케이션 및 OS 이미지(Amazon Machine Image)

Amazon Machine Image(AMI)에는 인스턴스를 생성하는 데 필요한 정보가 포함되어 있습니다. 예를 들어 AMI에는 웹 서버 역할을 수행하는 데 필요한 소프트웨어가 포함될 수 있습니다(예: Apache, 사용자의 웹 사이트).

최신 Amazon ECS 최적화 AMI 및 이들의 값은 Windows Amazon ECS 최적화 AMI를 참조하세요.

검색창을 사용하여 AWS에서 공개한 적절한 Amazon ECS 최적화 AMI를 찾습니다.

1. 요구 사항에 따라 다음의 AMI 중 하나를 Search(검색 창)에 입력하고 Enter 키를 누릅니다.
   - Windows_Server-2022-English-Full-ECS_Optimized
   - Windows_Server-2022-English-Core-ECS_Optimized
   - Windows_Server-2019-English-Full-ECS_Optimized
   - Windows_Server-2019-English-Core-ECS_Optimized
   - Windows_Server-2016-English-Full-ECS_Optimized
2. Amazon Machine Image(AMI) 선택 페이지에서 커뮤니티 AMI 탭을 선택합니다.
3. 표시되는 목록에서 게시 날짜가 가장 최신인 Microsoft 인증 AMI를 선택하고 선택(Select)을 클릭합니다.

인스턴스 유형

인스턴스 유형은 인스턴스의 하드웨어 구성과 크기를 정의합니다. 대형 인스턴스는 CPU와 메모리가 더 높습니다. 자세한 정보는 인스턴스 유형을 참조하세요.

- 인스턴스 유형(Instance type)에서 인스턴스에 대한 인스턴스 유형을 선택합니다.

  여기서 선택하는 인스턴스 유형은 실행할 작업에 사용 가능한 리소스를 결정합니다.

키 페어(로그인)

키 페어 이름(Key pair name)에서 기존 키 페어를 선택하거나 새로운 키 페어 생성(Create new key pair)을 선택하여 새로 생성합니다.

> **Important**
>
> 키 페어 없이 진행(Proceed without key pair)(권장하지 않음) 옵션을 선택할 경우 사용자가 다른 방법으로 로그인할 수 있도록 구성된 AMI를 선택해야만 인스턴스에 연결할 수 있습니다.

네트워크 설정

필요에 따라 네트워크 설정을 구성합니다.

- 네트워킹 플랫폼(Networking platform): Virtual Private Cloud (VPC)를 선택한 다음 네트워킹 인터페이스(Network interfaces) 섹션에서 서브넷을 지정합니다.
- VPC: 보안 그룹을 생성할 기존 VPC를 선택합니다.
- 서브넷(Subnet): 가용 영역, 로컬 영역, Wavelength Zone 또는 Outposts와 연결된 서브넷에서 인스턴스를 시작할 수 있습니다.

  가용 영역에서 인스턴스를 시작하려면 인스턴스를 시작할 서브넷을 선택합니다. 새 서브넷을 생성하려면 새 서브넷 생성을 선택하여 Amazon VPC 콘솔로 이동합니다. 마친 후에 인스턴스 시작 마법사로 돌아와 새로 고침 아이콘을 선택하면 해당 서브넷이 목록에 로딩됩니다.

  로컬 영역에서 인스턴스를 시작하려면 로컬 영역에 생성된 서브넷을 선택합니다.

  Outposts에서 인스턴스를 시작하려면 Outposts와 연결된 VPC의 서브넷을 선택합니다.
- 퍼블릭 IP 자동 할당(Auto-assign Public IP): 인터넷에서 인스턴스에 액세스할 수 있어야 한다면 퍼블릭 IP 자동 할당(Auto-assign Public IP) 필드가 사용 설정(Enable)으로 설정되어 있는지 확인합니다. 이렇게 설정이 되어 있지 않으면 이 필드를 비활성화(Disable)로 설정합니다.

> **Note**
>
> 컨테이너 인스턴스는 Amazon ECS 서비스 엔드포인트와 통신하기 위한 액세스 권한이 필요합니다. 액세스 권한은 인터페이스 VPC 엔드포인트를 통하거나 퍼블릭 IP 주소가 있는 컨테이너 인스턴스를 통해 부여할 수 있습니다.
> 인터페이스 VPC 엔드포인트에 대한 자세한 정보는 Amazon ECS 및 인터페이스 VPC 엔드포인트(AWS PrivateLink) 섹션을 참조하세요.
> 인터페이스 VPC 엔드포인트가 구성되어 있지 않고 컨테이너 인스턴스에 퍼블릭 IP 주소가 없는 경우, NAT(Network Address Translation)를 사용하여 이 액세스 권한을 제공해야 합

니다. 자세한 정보는 Amazon VPC 사용 설명서의 NAT 게이트웨이 및 이 가이드의 Amazon ECS Linux 컨테이너 인스턴스에 HTTP 프록시 사용 섹션을 참조하세요.

- 방화벽(보안 그룹)(Firewall (security groups)): 보안 그룹을 사용하여 컨테이너 인스턴스의 방화벽 규칙을 정의합니다. 이 규칙은 컨테이너 인스턴스에 전달되는 수신 네트워크 트래픽을 지정합니다. 다른 모든 트래픽은 무시됩니다.
  - 기존 보안 그룹을 선택하려면 기존 보안 그룹 선택(Select an existing security group)을 선택하고 Amazon ECS 사용 설정에서 생성한 보안 그룹을 선택합니다.

스토리지 구성

선택한 AMI에는 루트 볼륨을 포함한 하나 이상의 스토리지 볼륨이 있습니다. 인스턴스에 연결할 추가 볼륨을 지정할 수 있습니다.

간단(Simple) 보기를 사용할 수 있습니다.

- 스토리지 유형(Storage type): 컨테이너 인스턴스의 스토리지를 구성합니다.

  Amazon ECS 최적화 Amazon Linux AMI를 사용하는 경우 인스턴스에는 2개의 볼륨이 구성되어 있습니다. Root 볼륨은 운영 체제가 사용하고, 두 번째 Amazon EBS 볼륨(`/dev/xvdcz`에 연결)은 Docker가 사용합니다.

  애플리케이션 요구에 맞춰 인스턴스의 볼륨 크기를 선택적으로 늘리거나 줄일 수 있습니다.

고급 세부 정보

고급 세부 정보에서 필드를 볼 수 있도록 섹션을 확장하고 인스턴스를 위한 추가 파라미터를 지정합니다.

- 구매 옵션(Purchasing option): 스팟 인스턴스를 요청하려면 스팟 인스턴스 요청(Request Spot Instances)을 선택합니다. 스팟 인스턴스와 관련된 다른 필드도 설정해야 합니다. 자세한 정보는 스팟 인스턴스 요청을 참조하세요.

> **Note**
>
> 스팟 인스턴스를 사용하는 경우 `Not available` 메시지가 표시되면 다른 인스턴스 유형을 선택해야 할 수 있습니다.

.

- IAM 인스턴스 프로파일(IAM instance profile): 컨테이너 인스턴스 IAM 역할을 선택합니다. 이는 일반적으로 `ecsInstanceRole`로 이름이 지정됩니다.

  > **Important**
  >
  > 적절한 IAM 권한을 사용하여 컨테이너 인스턴스를 시작하지 않으면 Amazon ECS 에이전트가 클러스터에 연결할 수 없습니다. 자세한 정보는 Amazon ECS 컨테이너 인스턴스 IAM 역할을 참조하세요.

- (선택사항) 사용자 데이터(User data): 사용자 데이터(예: Amazon ECS 컨테이너 에이전트 구성의 에이전트 환경 변수)로 Amazon ECS 컨테이너 인스턴스를 구성합니다. Amazon EC2 사용자 데이터 스크립트는 인스턴스가 처음 시작될 때 한 번만 실행됩니다. 다음은 사용자 데이터의 일반적인 용례입니다.
  - 기본적으로 컨테이너 인스턴스는 기본 클러스터로 시작됩니다. 기본이 아닌 클러스터로 시작하려면 고급 세부 정보(Advanced Details) 목록을 선택합니다. 그런 다음 사용자 데이터(User data) 필드에 다음 스크립트를 붙여 넣고 *`your_cluster_name`*을 클러스터 이름으로 대체합니다.

    `EnableTaskIAMRole`은 태스크에 대한 태스크 IAM 역할 기능을 켭니다.

    또한 `awsvpc` 네트워크 모드를 사용할 때 다음과 같은 옵션을 사용할 수 있습니다.

    - `EnableTaskENI`: 이 플래그는 태스크 네트워킹을 활성화하며, `awsvpc` 네트워크 모드를 사용할 때 필요합니다.
    - `AwsvpcBlockIMDS`: 이 선택적 플래그는 `awsvpc` 네트워크 모드에서 실행 중인 태스크 컨테이너에 대한 IMDS 액세스를 차단합니다.
    - `AwsvpcAdditionalLocalRoutes`: 이 선택적 플래그를 사용하면 태스크 네임스페이스에 추가 경로를 가질 수 있습니다.

      `ip-address`를 추가 경로의 IP 주소(예: 172.31.42.23/32)로 교체합니다.

    ```
    <powershell>
    Import-Module ECSTools
    Initialize-ECSAgent -Cluster your_cluster_name -EnableTaskIAMRole -EnableTaskENI -
    AwsvpcBlockIMDS -AwsvpcAdditionalLocalRoutes
    '["ip-address"]'
    </powershell>
    ```

데이터 전달을 위한 Amazon ECS Windows 컨테이너 인스턴스 부트스트래핑

Amazon EC2 인스턴스를 시작하면 사용자 데이터를 EC2 인스턴스에 전달할 수 있습니다. 이 데이터는 일반적인 구성 태스크를 자동으로 수행하는 데 사용할 수 있고, 인스턴스가 부팅될 때 스크립트를 실행하는 데 사용할 수도 있습니다. Amazon ECS의 경우 사용자 데이터의 가장 일반적인 사용 사례는 구성 정보를 Docker 대몬 및 Amazon ECS 컨테이너 에이전트에 전달하는 것입니다.

클라우드 boothook, 셸 스크립트, `cloud-init` 명령을 비롯하여 여러 유형의 사용자 데이터를 Amazon EC2에 전달할 수 있습니다. 이러한 유형 및 다른 형식 유형에 대한 자세한 정보는 Cloud-Init 설명서를 참조하세요.

Amazon EC2 시작 마법사를 사용할 때 이 사용자 데이터를 전달할 수 있습니다. 자세한 정보는 Amazon ECS Linux 컨테이너 인스턴스 시작 섹션을 참조하세요.

기본 Windows 사용자 데이터

이 예제 사용자 데이터 스크립트에는 콘솔을 사용하는 경우 Windows 컨테이너 인스턴스가 수신하는 기본 사용자 데이터가 표시되어 있습니다. 아래의 스크립트는 다음을 수행합니다.

- 클러스터 이름을 입력한 이름으로 설정합니다.
- 태스크에 대한 IAM 역할을 설정합니다.
- `json-file` 및 `awslogs`를 사용 가능한 로깅 드라이버로 설정합니다.

또한 `awsvpc` 네트워크 모드를 사용할 때 다음과 같은 옵션을 사용할 수 있습니다.

- `EnableTaskENI`: 이 플래그는 태스크 네트워킹을 활성화하며, `awsvpc` 네트워크 모드를 사용할 때 필요합니다.
- `AwsvpcBlockIMDS`: 이 선택적 플래그는 `awsvpc` 네트워크 모드에서 실행 중인 태스크 컨테이너에 대한 IMDS 액세스를 차단합니다.
- `AwsvpcAdditionalLocalRoutes`: 이 선택적 플래그를 사용하면 추가 경로를 가질 수 있습니다.

  `ip-address`를 추가 경로의 IP 주소(예: 172.31.42.23/32)로 교체합니다.

자체 컨테이너 인스턴스(Amazon ECS 최적화 Windows Server AMI에서 시작된 경우)에 이 스크립트를 사용할 수 있습니다.

`-Cluster` *`cluster-name`* 줄을 바꾸어 자체 클러스터 이름을 지정합니다.

```
<powershell>
```

```
Initialize-ECSAgent -Cluster cluster-name -EnableTaskIAMRole -LoggingDrivers '["json-
file","awslogs"]' -EnableTaskENI -AwsvpcBlockIMDS -AwsvpcAdditionalLocalRoutes
'["ip-address"]'
</powershell>
```

awslogs 로깅 드라이버를 사용하도록 구성된 Windows 태스크의 경우 컨테이너 인스턴스에 ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE 환경 변수도 설정해야 합니다. 다음 구문을 사용합니다.

-Cluster *cluster-name* 줄을 바꾸어 자체 클러스터 이름을 지정합니다.

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE",
 $TRUE, "Machine")
Initialize-ECSAgent -Cluster cluster-name -EnableTaskIAMRole -LoggingDrivers '["json-
file","awslogs"]'
</powershell>
```

Windows 에이전트 설치 사용자 데이터

이 예제 사용자 데이터 스크립트에서는 Amazon ECS 컨테이너 에이전트를 Windows_Server-2016-English-Full-Containers AMI로 시작하는 인스턴스에 설치합니다. Amazon ECS Container Agent GitHub repository README 페이지의 에이전트 설치 지침에서 수정된 것입니다.

> **Note**
>
> 이 스크립트는 예시용으로 공유합니다. Amazon ECS 최적화 Windows Server AMI를 사용하여 Windows 컨테이너를 시작하는 것이 훨씬 쉽습니다. 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 클러스터 생성 섹션을 참조하세요.

Windows Server 2022 Full에 Amazon ECS 에이전트를 설치하는 방법에 대한 자세한 내용은 GitHub에서문제 3753을 참조하세요.

고유한 컨테이너 인스턴스(Windows_Server-2016-English-Full-Containers AMI 버전으로 시작된 경우)에 이 스크립트를 사용할 수 있습니다. *windows* 줄을 교체하여 고유한 클러스터 이름을 지정해야 합니다(이름이 windows인 클러스터를 사용하지 않는 경우).

```
<powershell>
# Set up directories the agent uses
```

```
New-Item -Type directory -Path ${env:ProgramFiles}\Amazon\ECS -Force
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS -Force
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS\data -Force
# Set up configuration
$ecsExeDir = "${env:ProgramFiles}\Amazon\ECS"
[Environment]::SetEnvironmentVariable("ECS_CLUSTER", "windows", "Machine")
[Environment]::SetEnvironmentVariable("ECS_LOGFILE", "${env:ProgramData}\Amazon\ECS\log
\ecs-agent.log", "Machine")
[Environment]::SetEnvironmentVariable("ECS_DATADIR", "${env:ProgramData}\Amazon\ECS
\data", "Machine")
# Download the agent
$agentVersion = "latest"
$agentZipUri = "https://s3.amazonaws.com/amazon-ecs-agent/ecs-agent-windows-
$agentVersion.zip"
$zipFile = "${env:TEMP}\ecs-agent.zip"
Invoke-RestMethod -OutFile $zipFile -Uri $agentZipUri
# Put the executables in the executable directory.
Expand-Archive -Path $zipFile -DestinationPath $ecsExeDir -Force
Set-Location ${ecsExeDir}
# Set $EnableTaskIAMRoles to $true to enable task IAM roles
# Note that enabling IAM roles will make port 80 unavailable for tasks.
[bool]$EnableTaskIAMRoles = $false
if (${EnableTaskIAMRoles}) {
  $HostSetupScript = Invoke-WebRequest https://raw.githubusercontent.com/aws/amazon-
ecs-agent/master/misc/windows-deploy/hostsetup.ps1
  Invoke-Expression $($HostSetupScript.Content)
}
# Install the agent service
New-Service -Name "AmazonECS" `
        -BinaryPathName "$ecsExeDir\amazon-ecs-agent.exe -windows-service" `
        -DisplayName "Amazon ECS" `
        -Description "Amazon ECS service runs the Amazon ECS agent" `
        -DependsOn Docker `
        -StartupType Manual
sc.exe failure AmazonECS reset=300 actions=restart/5000/restart/30000/restart/60000
sc.exe failureflag AmazonECS 1
Start-Service AmazonECS
</powershell>
```

Amazon ECS Windows 컨테이너 인스턴스에 HTTP 프록시 사용

Amazon ECS 컨테이너 에이전트와 Docker 대몬 모두에 대해 HTTP 프록시를 사용하도록 Amazon ECS 컨테이너 인스턴스를 구성할 수 있습니다. 이 구성은 컨테이너 인스턴스가 Amazon VPC 인터넷

게이트웨이, NAT 게이트웨이 또는 인스턴스를 통해 외부 네트워크에 액세스하지 못할 경우 유용합니다.

HTTP 프록시를 사용하도록 Amazon ECS Windows 컨테이너 인스턴스를 구성하려면 시작 시 (Amazon EC2 사용자 데이터를 사용해) 다음 변수를 설정합니다.

```
[Environment]::SetEnvironmentVariable("HTTP_PROXY",
"http://proxy.mydomain:port", "Machine")
```

Amazon ECS 에이전트가 인터넷에 연결하는 데 사용할 HTTP 프록시의 호스트 이름(또는 IP 주소) 및 포트 번호로 HTTP_PROXY를 설정합니다. 예를 들어 컨테이너 인스턴스는 Amazon VPC 인터넷 게이트웨이, NAT 게이트웨이 또는 인스턴스를 통해 외부 네트워크에 액세스하지 못할 수 있습니다.

```
[Environment]::SetEnvironmentVariable("NO_PROXY",
"169.254.169.254,169.254.170.2,\\.\pipe\docker_engine", "Machine")
```

NO_PROXY를 169.254.169.254,169.254.170.2,\\.\pipe\docker_engine으로 설정하여 EC2 인스턴스 메타데이터, 작업에 대한 IAM 역할 및 프록시의 Docker 대몬 트래픽을 필터링합니다.

Example Windows HTTP 프록시 사용자 데이터 스크립트

아래의 사용자 데이터 PowerShell 스크립트 예제는 사용자가 지정한 HTTP 프록시를 사용하도록 Amazon ECS 컨테이너 에이전트와 Docker 대몬을 구성합니다. 컨테이너 인스턴스가 자신을 등록할 클러스터를 지정할 수도 있습니다.

컨테이너 인스턴스를 시작할 때 이 스크립트를 사용하려면 the section called “컨테이너 인스턴스 시작”의 절차를 따르세요. 아래 PowerShell 스크립트를 복사하여 사용자 데이터 필드에 붙여넣기만 하면 됩니다(빨간색 예제 값을 실제 프록시 및 클러스터 정보로 바꿔야 함).

> **Note**
>
> 작업에 대해 IAM 역할을 활성화하려면 -EnableTaskIAMRole 옵션이 필요합니다. 자세한 내용은 Amazon EC2 Windows 인스턴스 추가 구성 섹션을 참조하세요.

```
<powershell>
Import-Module ECSTools
```

```
$proxy = "http://proxy.mydomain:port"
[Environment]::SetEnvironmentVariable("HTTP_PROXY", $proxy, "Machine")
[Environment]::SetEnvironmentVariable("NO_PROXY", "169.254.169.254,169.254.170.2,\\.
\pipe\docker_engine", "Machine")

Restart-Service Docker
Initialize-ECSAgent -Cluster MyCluster -EnableTaskIAMRole
</powershell>
```

스팟 인스턴스 알림을 수신하도록 Amazon ECS Windows 컨테이너 인스턴스 구성

스팟 가격이 요청의 최고가를 초과하거나 용량이 더 이상 제공되지 않는 경우 Amazon EC2는 스팟 인스턴스를 종료, 중지 또는 최대 절전 모드로 전환합니다. Amazon EC2는 스팟 인스턴스 중단 2분 전에 경고하는 스팟 인스턴스 중단 공지를 제공합니다. Amazon ECS 스팟 인스턴스 드레이닝이 인스턴스에서 활성화되면 ECS는 스팟 인스턴스 중단 공지를 수신하고 인스턴스를 DRAINING 상태로 둡니다.

> **Important**
>
> Amazon ECS는 terminate 및 stop instance-action을 포함하고 있는 스팟 인스턴스 중단 공지를 모니터링합니다. 스팟 인스턴스 또는 스팟 플릿 요청 시 hibernate 인스턴스 중단 동작을 지정한 경우, 이러한 인스턴스에 대해서는 Amazon ECS 스팟 인스턴스 드레이닝은 지원되지 않습니다.

컨테이너 인스턴스를 DRAINING으로 설정할 경우 Amazon ECS는 새 작업이 컨테이너 인스턴스에서 배치를 위해 예약되지 않도록 합니다. 드레이닝 컨테이너 인스턴스에서 PENDING 상태인 서비스 작업이 즉시 중지됩니다. 클러스터에 사용 가능한 컨테이너 인스턴스가 있는 경우 거기서 대체 서비스 작업이 시작됩니다.

인스턴스를 시작할 때 스팟 인스턴스 드레인을 켤 수 있습니다. 컨테이너 에이전트를 시작하기 전에 ECS_ENABLE_SPOT_INSTANCE_DRAINING 파라미터를 설정해야 합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
[Environment]::SetEnvironmentVariable("ECS_ENABLE_SPOT_INSTANCE_DRAINING", "true",
 "Machine")

# Initialize the agent
Initialize-ECSAgent -Cluster my-cluster
```

자세한 내용은 the section called “컨테이너 인스턴스 시작” 섹션을 참조하세요.

외부 시작 유형을 위한 Amazon ECS 클러스터

Amazon ECS Anywhere는 외부 인스턴스(예: 온프레미스 서버 또는 가상 머신(VM))을 Amazon ECS 클러스터에 등록하도록 지원합니다. 외부 인스턴스는 아웃바운드 트래픽 또는 프로세스 데이터를 생성하는 애플리케이션 실행에 최적화되어 있습니다. 애플리케이션에 인바운드 트래픽이 필요한 경우 Elastic Load Balancing 지원이 없기 때문에 이러한 워크로드를 실행하는 데 효율성이 떨어집니다. Amazon ECS는 외부 인스턴스에서 서비스를 생성하거나 태스크를 실행하는 데 사용할 수 있는 새로운 `EXTERNAL` 시작 유형을 추가했습니다.

지원되는 운영 체제 및 시스템 아키텍처

지원되는 운영 체제 및 시스템 아키텍처의 목록은 다음과 같습니다.

- Amazon Linux 2
- Amazon Linux 2023
- CentOS Stream 9
- RHEL 7, RHEL 8, RHEL 9 - Docker 또는 RHEL의 오픈 패키지 리포지토리는 RHEL에 기본적으로 Docker를 설치하는 것을 지원하지 않습니다. 이 문서에서 설명하는 설치 스크립트를 실행하기 전에 Docker가 설치되어 있는지 확인해야 합니다.
- Fedora 32, Fedora 33, Fedora 40
- OpenSUSE Tumbleweed
- Ubuntu 18, Ubuntu 20, Ubuntu 22, Ubuntu 24
- Debian 10

> **Important**
>
> Debian 9 LTS(Long Term Support) 지원은 2022년 6월 30일에 종료되었으며 Amazon ECS Anywhere에서 더 이상 지원되지 않습니다.

- Debian 11
- Debian 12
- SUSE Enterprise Server 15
- `x86_64` 및 `ARM64` CPU 아키텍처가 지원됩니다.

- 다음 Windows 운영 체제 버전이 지원됩니다.
  - Windows Server 2022
  - Windows Server 2019
  - Windows Server 2016
  - Windows Server 20H2

고려 사항

외부 인스턴스를 사용하기 전에 다음 사항을 고려해야 합니다.

- 한 번에 하나의 클러스터에 외부 인스턴스를 등록할 수 있습니다. 다른 클러스터에 외부 인스턴스를 등록하는 방법에 대한 지침은 Amazon ECS 외부 인스턴스 등록 취소 섹션을 참조하세요.
- 외부 인스턴스에는 AWS API와 통신할 수 있도록 하는 IAM 역할이 필요합니다. 자세한 내용은 Amazon ECS Anywhere IAM 역할 섹션을 참조하세요.
- 외부 인스턴스에는 사전 구성된 인스턴스 자격 증명 체인이 로컬로 정의되어 있지 않아야 합니다. 정의된 경우 등록 스크립트에 방해가 될 수 있습니다.
- 컨테이너 로그를 CloudWatch Logs로 보내려면 태스크 정의에 태스크 실행 IAM 역할을 생성하고 지정해야 합니다.
- 외부 인스턴스가 클러스터에 등록되면 `ecs.capability.external` 속성은 인스턴스와 연결되어 있습니다. 이 속성은 인스턴스를 외부 인스턴스로 식별합니다. 외부 인스턴스에 사용자 지정 속성을 추가하여 작업 배치 제약 조건으로 사용할 수 있습니다. 자세한 정보는 사용자 지정 속성 섹션을 참조하세요.
- 외부 인스턴스에 리소스 태그를 추가할 수 있습니다. 자세한 정보는 외부 컨테이너 인스턴스을 참조하세요.
- ECS Exec은 외부 인스턴스에서 지원됩니다. 자세한 정보는 ECS Exec를 사용하여 Amazon ECS 컨테이너 모니터링을 참조하세요.
- 외부 인스턴스와의 네트워킹과 관련된 추가 고려 사항은 다음과 같습니다. 자세한 정보는 네트워킹 섹션을 참조하세요.
  - 서비스 로드 밸런싱은 지원되지 않습니다.
  - 서비스 검색은 지원되지 않습니다.
  - 외부 인스턴스에서 실행되는 태스크는 `bridge`, `host` 또는 `none` 네트워크 모드를 사용해야 합니다. `awsvpc` 네트워크 모드는 지원되지 않습니다.
  - 각 AWS 리전에 Amazon ECS 서비스 도메인이 있습니다. 외부 인스턴스로 트래픽을 보낼 수 있도록 이러한 서비스 도메인이 허용되어야 합니다.

  - 외부 인스턴스에 설치된 SSM Agent는 하드웨어 지문을 사용하여 30분마다 교체되는 IAM 자격 증명을 유지합니다. 외부 인스턴스와 AWS의 연결이 끊어지는 경우 SSM Agent는 연결이 다시 설정된 후 자격 증명을 자동으로 새로 고칩니다. 자세한 정보는 AWS Systems Manager 사용 설명서의 하드웨어 지문을 사용하여 온프레미스 서버 및 가상 머신 유효성 검사를 참조하세요.
- `UpdateContainerAgent` API는 지원되지 않습니다. 외부 인스턴스에서 SSM Agent 또는 Amazon ECS 에이전트를 업데이트하는 방법에 대한 지침은 외부 인스턴스의 AWS Systems Manager 에이전트 및 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.
- Amazon ECS 용량 공급자는 지원되지 않습니다. 외부 인스턴스에서 서비스를 생성하거나 독립 실행형 태스크를 실행하려면 `EXTERNAL` 시작 유형을 사용합니다.
- SELinux는 지원되지 않습니다.
- Amazon EFS 볼륨 사용이나 `EFSVolumeConfiguration` 지정은 지원되지 않습니다.
- App Mesh와의 통합은 지원되지 않습니다.
- 콘솔을 사용하여 외부 인스턴스 작업 정의를 생성하는 경우 콘솔 JSON 편집기로 작업 정의를 생성해야 합니다.
- Windows에서 ECS Anywhere를 실행하는 경우 온프레미스 인프라에서 자체 Windows 라이선스를 사용해야 합니다.
- Amazon ECS 최적화가 아닌 AMI를 사용하는 경우 외부 컨테이너 인스턴스에서 다음 명령을 실행하여 작업에 IAM 역할을 사용하도록 규칙을 구성합니다. 자세한 내용은 외부 인스턴스 추가 구성 섹션을 참조하세요.

```
$ sysctl -w net.ipv4.conf.all.route_localnet=1
$ iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-
destination 127.0.0.1:51679
$ iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --
to-ports 51679
```

네트워킹

Amazon ECS 외부 인스턴스는 아웃바운드 트래픽 또는 프로세스 데이터를 생성하는 애플리케이션 실행에 최적화되어 있습니다. 애플리케이션에 웹 서비스와 같은 인바운드 트래픽이 필요한 경우 Elastic Load Balancing 지원이 부족하여 이러한 워크로드를 로드 밸런서 뒤에 배치할 수 없기 때문에 이러한 워크로드를 실행하는 데 효율성이 떨어집니다.

외부 인스턴스와의 네트워킹과 관련된 추가 고려 사항은 다음과 같습니다.

- 서비스 로드 밸런싱은 지원되지 않습니다.

- 서비스 검색은 지원되지 않습니다.
- 외부 인스턴스에서 실행되는 Linux 태스크는 `bridge`, `host` 또는 `none` 네트워크 모드를 사용해야 합니다. `awsvpc` 네트워크 모드는 지원되지 않습니다.

  각 네트워크 모드에 대한 자세한 내용은 EC2 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션을 참조하세요.
- 외부 인스턴스에서 실행되는 Windows 태스크는 `default` 네트워크 모드를 사용해야 합니다.
- 각 리전에 Amazon ECS 서비스 도메인이 있으며 외부 인스턴스로 트래픽을 보낼 수 있도록 허용해야 합니다.
- 외부 인스턴스에 설치된 SSM Agent는 하드웨어 지문을 사용하여 30분마다 교체되는 IAM 자격 증명을 유지합니다. 외부 인스턴스와 AWS의 연결이 끊어지는 경우 SSM Agent는 연결이 다시 설정된 후 자격 증명을 자동으로 새로 고칩니다. 자세한 정보는 AWS Systems Manager 사용 설명서의 하드웨어 지문을 사용하여 온프레미스 서버 및 가상 머신 유효성 검사를 참조하세요.

다음 도메인은 Amazon ECS 서비스와 외부 인스턴스에 설치된 Amazon ECS 에이전트 간의 통신에 사용됩니다. 트래픽이 허용되고 DNS 확인이 작동하는지 확인합니다. 각 엔드포인트의 `##`은 미국 동부(오하이오) 리전의 `us-east-2`와 같이 Amazon ECS가 지원하는 AWS 리전의 리전 식별자를 나타냅니다. 사용하는 모든 리전의 엔드포인트가 허용되어야 합니다. `ecs-a` 및 `ecs-t` 엔드포인트의 경우 별표를 포함해야 합니다(예: `ecs-a-*`).

- `ecs-a-*.region.amazonaws.com` — 이 엔드포인트는 태스크를 관리할 때 사용됩니다.
- `ecs-t-*.region.amazonaws.com` — 이 엔드포인트는 태스크 및 컨테이너 지표를 관리하는 데 사용됩니다.
- `ecs.region.amazonaws.com` — Amazon ECS의 서비스 엔드포인트입니다.
- `ssm.region.amazonaws.com` - AWS Systems Manager의 서비스 엔드포인트입니다.
- `ec2messages.region.amazonaws.com` - AWS Systems Manager에서 클라우드의 Systems Manager 에이전트와 Systems Manager 서비스 간에 통신하는 데 사용하는 서비스 엔드포인트입니다.
- `ssmmessages.region.amazonaws.com` - 이 서비스 엔드포인트는 클라우드의 Session Manager 서비스를 사용하여 세션 채널을 생성하고 삭제하는 데 필요합니다.
- 태스크에 다른 AWS 서비스와의 통신이 필요한 경우 해당 서비스 엔드포인트가 허용되는지 확인합니다. 예제 애플리케이션으로는 Amazon ECR을 사용하여 컨테이너 이미지를 가져오거나 CloudWatch 로그에 대해 CloudWatch를 사용하는 것이 있습니다. 자세한 정보는 AWS 일반 참조의 서비스 엔드포인트를 참조하세요.

Amazon FSx for Windows File Server와 ECS Anywhere

Amazon ECS 외부 인스턴스와 함께 Amazon FSx for Windows File Server를 사용하려면 온프레미스 데이터 센터와 AWS 클라우드 간에 연결을 설정해야 합니다. VPC에 네트워크를 연결하는 옵션에 대한에 대한 자세한 정보는 Amazon Virtual Private Cloud Connectivity Options(Amazon Virtual Private Cloud 연결 옵션)를 참조하세요.

gMSA와 ECS Anywhere

ECS Anywhere에 대해 다음 사용 사례가 지원됩니다.

- Active Directory가 AWS 클라우드에 있음 - 이 구성의 경우 AWS Direct Connect 연결을 사용하여 온프레미스 네트워크와 AWS 클라우드 간에 연결을 생성합니다. 연결을 생성하는 방법에 대한 자세한 정보는 Amazon Virtual Private Cloud 연결 옵션을 참조하세요. AWS 클라우드에서 Active Directory를 생성합니다. AWS Directory Service를 시작하는 방법에 대한 자세한 정보는 AWS Directory Service 관리 안내서의 AWS Directory Service 설정을 참조하세요. 그런 다음 AWS Direct Connect 연결을 사용하여 외부 인스턴스를 도메인에 조인할 수 있습니다. Amazon ECS에서 gMSA 작업에 대한 자세한 정보는 the section called “EC2 Windows 컨테이너에 대해 gMSA를 사용하는 방법을 알아봅니다.” 섹션을 참조하세요.
- Active Directory는 온프레미스 데이터 센터에 있습니다. - 이 구성의 경우 외부 인스턴스를 온프레미스 Active Directory에 조인합니다. 그런 다음 Amazon ECS 태스크를 실행할 때 로컬에서 사용 가능한 자격 증명을 사용합니다.

외부 시작 유형을 위한 Amazon ECS 클러스터 생성

클러스터를 생성하여 태스크와 서비스가 실행되는 인프라를 정의합니다.

시작하기 전에 먼저 Amazon ECS 사용 설정 단계를 완료하고 적절한 IAM 권한을 할당했는지 확인합니다. 자세한 내용은 the section called “Amazon ECS 클러스터 예제” 섹션을 참조하세요. Amazon ECS 콘솔은 AWS CloudFormation 스택을 생성하여 Amazon ECS 클러스터에 필요한 리소스를 생성하는 간단한 방법을 제공합니다.

클러스터 생성 프로세스를 가능한 한 쉽게 만들기 위해 콘솔에는 아래에서 설명하는 여러 선택 항목에 대한 기본 선택 항목이 있습니다. 추가 컨텍스트를 제공하는 콘솔의 대부분의 섹션에서 사용할 수 있는 도움말 패널도 있습니다.

다음 옵션을 수정할 수 있습니다.

- 클러스터에 네임스페이스를 추가합니다.

네임스페이스를 사용하면 클러스터에서 생성한 서비스를 추가 구성 없이 네임스페이스의 다른 서비스에 연결할 수 있습니다. 자세한 내용은 Amazon ECS 서비스 상호 연결 섹션을 참조하세요.

- 외부 인스턴스에 대한 클러스터를 구성합니다.
- 관찰성이 향상된 Container Insights 또는 Container Insights를 켭니다.

  CloudWatch Container Insights는 컨테이너 애플리케이션 및 마이크로서비스의 지표 및 로그를 수집하고, 종합하며, 요약합니다. 또한 Container Insights는 컨테이너 재시작 오류 등의 진단 정보를 제공합니다. 이 정보를 사용하여 문제를 격리하고 신속하게 해결할 수 있습니다. 자세한 내용은 the section called “관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링” 섹션을 참조하세요.

  2024년 12월 2일 AWS에서는 Amazon ECS의 향상된 관찰성을 갖춘 Container Insights를 출시했습니다. 이 버전은 Amazon EC2 및 Fargate 시작 유형을 사용하여 Amazon ECS 클러스터의 관찰성을 향상할 수 있도록 지원합니다. Amazon ECS에서 향상된 관찰성으로 Container Insights를 구성하면, Container Insights는 클러스터 수준에서 환경의 컨테이너 수준까지 상세한 인프라 원격 측정을 자동으로 수집하고, 대시보드에서 다양한 지표와 차원을 보여주는 데이터를 표시합니다. 그런 다음 Container Insights 콘솔에서 이러한 기본 대시보드를 사용하여 컨테이너 상태와 성능을 더 잘 이해하고 이상 징후를 식별하여 문제를 더 빠르게 완화할 수 있습니다.

  Container Insights 대신 관찰성이 향상된 Container Insights를 사용하는 것이 좋습니다. 컨테이너 환경에서 세부적인 가시성을 제공하여 평균 해결 시간을 단축하기 때문입니다.
- 관리형 스토리지에 AWS KMS 키를 할당합니다. 키 생성 방법에 대한 자세한 내용은 AWS Key Management Service 사용 설명서의 KMS 키 생성을 참조하세요.
- 클러스터를 식별하는 데 도움이 되는 태그를 추가합니다.

새 클러스터 생성(Amazon ECS 콘솔)

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터(Clusters) 페이지에서 클러스터 생성(Create cluster)을 선택합니다.
5. 클러스터 구성에서 다음을 구성합니다.

   - 클러스터 이름에 고유한 이름을 입력합니다.

     이름은 최대 255자(대/소문자), 숫자 및 하이픈을 포함할 수 있습니다.

- (선택 사항) 서비스 연결에 사용되는 네임스페이스를 클러스터 이름과 다르게 하려면 네임스페이스에 고유한 이름을 입력합니다.

6. (선택 사항) Container Insights를 사용하고 모니터링을 확장한 후에 다음 옵션 중 하나를 선택합니다.

   - 권장되는 관찰성이 향상된 Container Insights를 사용하려면 관찰성이 향상된 Container Insights를 선택합니다.
   - Container Insights를 사용하려면 Container Insights를 선택합니다.

7. (선택 사항) 관리형 스토리지의 데이터를 암호화합니다. 암호화에서 관리형 스토리지에 관리형 스토리지 데이터를 암호화하는 데 사용할 AWS KMS 키의 ARN을 입력합니다.
8. (선택 사항) 클러스터를 식별하려면 태그(Tags)를 펼친 다음, 태그를 구성합니다.

   [태그 추가] 태그 추가(Add tag)를 선택하고 다음을 수행합니다.

   - 키(Key)에 키 이름을 입력합니다.
   - 값에 키 값을 입력합니다.

9. 생성(Create)을 선택합니다.

다음 단계

클러스터에 인스턴스를 등록해야 합니다. 자세한 내용은 Amazon ECS 클러스터에 외부 인스턴스 등록 섹션을 참조하세요.

외부 시작 유형에 대한 태스크 정의를 생성합니다. 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

애플리케이션을 독립 실행형 태스크 또는 서비스의 일부로 실행합니다. 자세한 내용은 다음 자료를 참조하세요.

- 애플리케이션을 Amazon ECS 태스크로 실행
- Amazon ECS 롤링 업데이트 배포 생성

Amazon ECS 클러스터에 외부 인스턴스 등록

Amazon ECS 클러스터에 등록하는 각 외부 인스턴스에 대해 SSM Agent, Amazon ECS 컨테이너 에이전트 및 Docker가 설치되어 있어야 합니다. Amazon ECS 클러스터에 외부 인스턴스를 등록하려면

먼저 AWS Systems Manager 관리형 인스턴스로 등록되어야 합니다. Amazon ECS 콘솔에서 몇 번의 클릭만으로 설치 스크립트를 생성할 수 있습니다. 설치 스크립트에는 Systems Manager 활성화 키와 필요한 에이전트 및 Docker를 각각 설치하는 명령이 포함되어 있습니다. 설치 및 등록 단계를 완료하려면 온프레미스 서버 또는 VM에서 설치 스크립트를 실행해야 합니다.

> **Note**
>
> 클러스터에 외부 인스턴스를 등록하기 전에 Linux 외부 인스턴스에 `/etc/ecs/ecs.config` 파일을 열고 생성하고 원하는 컨테이너 에이전트 구성 파라미터를 추가합니다. 외부 인스턴스를 클러스터에 등록한 후에는 이 태스크를 수행할 수 없습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

AWS Management Console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터 페이지에서 외부 인스턴스를 등록할 클러스터를 선택합니다.
5. 클러스터: ***name***(Cluster : name) 페이지에서 인프라(Infrastructure) 탭을 선택합니다.
6. 외부 인스턴스 등록(Register external instances) 페이지에서 다음 단계를 수행합니다.
   a. 활성화 키 기간(일)(Activation key duration (in days))에 활성화 키가 활성 상태로 유지되는 일수를 입력합니다. 입력한 일수가 경과하면 외부 인스턴스를 등록할 때 키가 더 이상 작동하지 않습니다.
   b. 인스턴스 개수(Number of instances)에 활성화 키를 사용하여 클러스터에 등록할 외부 인스턴스 수를 입력합니다.
   c. 인스턴스 역할(Instance role에서 외부 인스턴스와 연결할 IAM 역할을 선택합니다. 역할이 아직 생성되지 않은 경우 새 역할 생성(Create new role)을 선택하여 Amazon ECS가 사용자를 대신하여 역할을 생성하도록 합니다. 외부 인스턴스에 필요한 IAM 권한에 대한 자세한 정보는 Amazon ECS Anywhere IAM 역할 섹션을 참조하세요.
   d. 등록 명령을 복사합니다. 이 명령은 클러스터에 등록하려는 각 외부 인스턴스에서 실행해야 합니다.

> **Important**
>
> 스크립트의 bash 부분은 루트로 실행해야 합니다. 명령이 루트로 실행되지 않으면 오류가 반환됩니다.

e. 닫기를 선택하세요.

AWS CLI for Linux operating systems

1. Systems Manager 정품 인증 쌍을 생성합니다. 이것은 Systems Manager 관리형 인스턴스 정품 인증에 사용됩니다. 출력에는 `ActivationId` 및 `ActivationCode`가 포함됩니다. 이것은 이후 단계에서 사용하게 됩니다. 생성한 ECS Anywhere IAM 역할을 지정해야 합니다. 자세한 내용은 Amazon ECS Anywhere IAM 역할 섹션을 참조하세요.

```
aws ssm create-activation --iam-role ecsAnywhereRole | tee ssm-activation.json
```

2. 온프레미스 서버 또는 가상 머신(VM)에서 설치 스크립트를 다운로드합니다.

```
curl --proto "https" -o "/tmp/ecs-anywhere-install.sh" "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install-latest.sh"
```

3. (선택 사항) 온프레미스 서버 또는 가상 머신(VM)에서 다음 단계를 통해 스크립트 서명 파일을 사용하여 설치 스크립트를 확인합니다.

   a. GnuPG를 다운로드하고 설치합니다. GNUpg에 대한 자세한 정보는 GnuPG 웹 사이트를 참조하세요. Linux 시스템의 경우 원하는 Linux 패키지 관리자를 사용하여 gpg를 설치합니다.

   b. Amazon ECS PGP 퍼블릭 키를 가져옵니다.

```
gpg --keyserver hkp://keys.gnupg.net:80 --recv BCE9D9A42D51784F
```

   c. 설치 스크립트 서명을 다운로드합니다. 서명은 `.asc` 확장자 파일에 저장된 ASCII 분리 PGP 서명입니다.

```
curl --proto "https" -o "/tmp/ecs-anywhere-install.sh.asc" "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install-latest.sh.asc"
```

   d. 키를 사용하여 설치 스크립트 파일을 확인합니다.

```
gpg --verify /tmp/ecs-anywhere-install.sh.asc /tmp/ecs-anywhere-install.sh
```

예상 출력은 다음과 같습니다.

```
gpg: Signature made Tue 25 May 2021 07:16:29 PM UTC
gpg:                using RSA key 50DECCC4710E61AF
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the
 owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
     Subkey fingerprint: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

4. 온프레미스 서버 또는 가상 머신(VM)에서 설치 스크립트를 실행합니다. 첫 번째 단계에서 클러스터 이름, 리전 및 Systems Manager 정품 인증 ID 및 정품 인증 코드를 지정합니다.

```
sudo bash /tmp/ecs-anywhere-install.sh \
    --region $REGION \
    --cluster $CLUSTER_NAME \
    --activation-id $ACTIVATION_ID \
    --activation-code $ACTIVATION_CODE
```

GPU 워크로드용으로 NVIDIA 드라이버가 설치된 온프레미스 서버 또는 가상 머신(VM)의 경우 `--enable-gpu` 플래그를 설치 스크립트에 추가해야 합니다. 이 플래그가 지정되면 설치 스크립트에서 NVIDIA 드라이버가 실행 중인지 확인한 다음 Amazon ECS 태스크를 실행하는 데 필요한 구성 변수를 추가합니다. GPU 워크로드 실행 및 태스크 정의에서 GPU 요구 사항 지정에 대한 자세한 정보는 Amazon ECS 작업 정의에서 GPU 지정 섹션을 참조하세요.

```
sudo bash /tmp/ecs-anywhere-install.sh \
    --region $REGION \
    --cluster $CLUSTER_NAME \
    --activation-id $ACTIVATION_ID \
    --activation-code $ACTIVATION_CODE \
    --enable-gpu
```

다음 단계를 사용하여 기존 외부 인스턴스를 다른 클러스터에 등록합니다.

기존 외부 인스턴스를 다른 클러스터에 등록하려면

1. Amazon ECS 컨테이너 에이전트를 중지합니다.

```
sudo systemctl stop ecs.service
```

2. `/etc/ecs/ecs.config` 파일을 편집하고 `ECS_CLUSTER` 행에서 클러스터 이름이 외부 인스턴스를 등록할 클러스터의 이름과 일치하는지 확인합니다.
3. 기존 Amazon ECS 에이전트 데이터를 제거합니다.

```
sudo rm /var/lib/ecs/data/agent.db
```

4. Amazon ECS 컨테이너 에이전트를 시작합니다.

```
sudo systemctl start ecs.service
```

AWS CLI for Windows operating systems

1. Systems Manager 정품 인증 쌍을 생성합니다. 이것은 Systems Manager 관리형 인스턴스 정품 인증에 사용됩니다. 출력에는 `ActivationId` 및 `ActivationCode`가 포함됩니다. 이것은 이후 단계에서 사용하게 됩니다. 생성한 ECS Anywhere IAM 역할을 지정해야 합니다. 자세한 내용은 Amazon ECS Anywhere IAM 역할 섹션을 참조하세요.

```
aws ssm create-activation --iam-role ecsAnywhereRole | tee ssm-activation.json
```

2. 온프레미스 서버 또는 가상 머신(VM)에서 설치 스크립트를 다운로드합니다.

```
Invoke-RestMethod -URI "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install.ps1" -OutFile “ecs-anywhere-install.ps1”
```

3. (선택 사항) PowerShell 스크립트는 Amazon에서 서명하므로 Windows에서 자동으로 인증서 검증을 수행합니다. 수동 검증을 수행할 필요가 없습니다.

   인증서를 수동으로 확인하려면 파일을 마우스 오른쪽 버튼으로 클릭하고 속성으로 이동한 다음 디지털 서명(Digital Signatures) 탭을 사용하여 자세한 정보를 확인합니다.

   이 옵션은 호스트의 인증서 스토어에 인증서가 있는 경우에만 사용할 수 있습니다.

   다음과 비슷한 정보가 반환됩니다.

```
# Verification (PowerShell)
Get-AuthenticodeSignature -FilePath .\ecs-anywhere-install.ps1

SignerCertificate                         Status      Path
-----------------                         ------      ----
EXAMPLECERTIFICATE                        Valid       ecs-anywhere-install.ps1

...

Subject              : CN="Amazon Web Services, Inc.",...

----
```

4. 온프레미스 서버 또는 가상 머신(VM)에서 설치 스크립트를 실행합니다. 첫 번째 단계에서 클러스터 이름, 리전 및 Systems Manager 정품 인증 ID 및 정품 인증 코드를 지정합니다.

```
.\ecs-anywhere-install.ps1 -Region $Region -Cluster $Cluster -
ActivationID $ActivationID -ActivationCode $ActivationCode
```

5. Amazon ECS 컨테이너 에이전트가 실행 중인지 확인합니다.

```
Get-Service AmazonECS

Status   Name               DisplayName
------   ----               -----------
Running  AmazonECS          Amazon ECS
```

다음 단계를 사용하여 기존 외부 인스턴스를 다른 클러스터에 등록합니다.

기존 외부 인스턴스를 다른 클러스터에 등록하려면

1. Amazon ECS 컨테이너 에이전트를 중지합니다.

```
Stop-Service AmazonECS
```

2. 클러스터 이름이 외부 인스턴스를 등록할 클러스터 이름과 일치하도록 ECS_CLUSTER 파라미터를 수정합니다.

```
[Environment]::SetEnvironmentVariable("ECS_CLUSTER", $ECSCluster,
 [System.EnvironmentVariableTarget]::Machine)
```

3. 기존 Amazon ECS 에이전트 데이터를 제거합니다.

```
Remove-Item -Recurse -Force $env:ProgramData\Amazon\ECS\data\*
```

4. Amazon ECS 컨테이너 에이전트를 시작합니다.

```
Start-Service AmazonECS
```

외부 인스턴스 등록 프로세스를 완료하기 위해 설치 스크립트를 실행하기 전에 AWS CLI를 사용하여 Systems Manager 정품 인증을 생성할 수 있습니다.

Amazon ECS 외부 인스턴스 등록 취소

인스턴스 사용을 마친 후에 Amazon ECS 및 AWS Systems Manager 모두에서 인스턴스를 등록 취소하는 것이 좋습니다. 등록 취소 후 외부 인스턴스는 더 이상 새 태스크를 받을 수 없습니다.

등록 취소 시 컨테이너 인스턴스에서 실행 중인 태스크가 있는 경우, 이러한 태스크는 다른 수단을 통해 중지될 때까지 계속 실행됩니다. 그러나 이러한 태스크는 Amazon ECS에서 더 이상 모니터링하거나 고려하지 않습니다. 외부 인스턴스의 이러한 태스크가 Amazon ECS 서비스의 일부인 경우, 서비스 스케줄러는 가능하다면 다른 인스턴스에서 해당 태스크의 다른 사본을 시작합니다.

인스턴스를 등록 취소한 후 인스턴스에서 남은 AWS 리소스를 정리합니다. 그런 다음, 새 클러스터에 등록할 수 있습니다.

절차

AWS Management Console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 외부 인스턴스가 등록되어 있는 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택하고 외부 인스턴스를 호스팅하는 클러스터를 선택합니다.
4. 클러스터: ***name***(Cluster : name) 페이지에서 인프라(Infrastructure) 탭을 선택합니다.
5. 컨테이너 인스턴스(Container instances)에서 등록 취소하려는 외부 인스턴스 ID를 선택합니다. 컨테이너 인스턴스 세부 정보 페이지로 리디렉션됩니다.

6. 컨테이너 인스턴스 : ***id*** 페이지에서 등록 취소(Deregister)를 선택합니다.
7. 등록 취소 메시지를 검토합니다. AWS Systems Manager에서 등록 취소(Deregister from SYSlong)를 선택하여 외부 인스턴스를 Systems Manager 관리형 인스턴스로 등록 취소할 수도 있습니다. 등록 취소(Deregister)를 선택합니다.

> **Note**
>
> Systems Manager 콘솔에서 외부 인스턴스를 Systems Manager 관리형 인스턴스로 등록 취소할 수 있습니다. 지침은 AWS Systems Manager 사용 설명서의 하이브리드 및 멀티클라우드 환경의 관리형 노드 등록 취소를 참조하세요.

8. 인스턴스를 등록 취소한 후 온프레미스 서버 또는 VM에서 AWS 리소스를 정리하세요.

운영 체제	단계	

Linux	a. 인스턴스에서 Amazon ECS 컨테이너 에이전트와 SSM Agent 서비스를 중지합니다.
	`sudo systemctl stop ecs amazon-ssm-agent`
	b. Amazon ECS 및 Systems Manager 패키지를 제거합니다.
	CentOS 7, CentOS 8 및 RHEL 7의 경우
	`sudo yum remove -y amazon-ecs-init amazon-ssm-agent`
	SUSE Enterprise Server 15의 경우	

운영 체제	단계	

	`sudo zypper remove -y amazon-ecs-init amazon-ssm-agent`
	Debian 및 Ubuntu의 경우
	`sudo apt remove -y amazon-ecs-init amazon-ssm-agent`
	c. 남은 디렉터리를 제거합니다.
	`sudo rm -rf /var/lib/ecs /etc/ecs / var/lib/amazon/ssm /var/log/ecs / var/log/amazon/ssm`	

Windows	a. 인스턴스에서 Amazon ECS 컨테이너 에이전트와 SSM Agent 서비스를 중지합니다.
	`Stop-Service AmazonECS`
	`Stop-Service AmazonSSMAgent`
	b. Amazon ECS 패키지를 제거합니다.
	`.\ecs-anywhere-install.ps1 -Uninstall`	

AWS CLI

1. 컨테이너 인스턴스를 등록 취소하려면 인스턴스 ID와 컨테이너 인스턴스 ARN이 필요합니다. 이러한 값이 없는 경우 다음 명령을 실행합니다.

   다음 명령을 실행하여 인스턴스 ID를 가져옵니다.

   인스턴스 ID(`instanceID`)를 사용하여 컨테이너 인스턴스 ARN(`containerInstanceARN`)을 가져옵니다.

   ```
   instanceId=$(aws ssm describe-instance-information --region "{{ region }}" |
    jq ".InstanceInformationList[] |select(.IPAddress==\"{{ IPv4 Address }}\")
    | .InstanceId" | tr -d'"'
   ```

   다음 명령을 실행합니다.

   `containerInstanceArn`을 명령의 파라미터로 사용하여 인스턴스 등록을 취소합니다(`deregister-container-instance`).

   ```
   instances=$(aws ecs list-container-instances --cluster "{{ cluster }}" --region
    "{{ region }}" | jq -c '.containerInstanceArns')
   containerInstanceArn=$(aws ecs describe-container-instances --cluster
    "{{ cluster }}" --region "{{ region }}" --container-instances $instances
    | jq ".containerInstances[] | select(.ec2InstanceId==\"{{ instanceId }}\")
    | .containerInstanceArn" | tr -d '"')
   ```

2. 다음 명령을 실행하여 인스턴스를 드레이닝합니다.

   ```
   aws ecs update-container-instances-state --cluster "{{ cluster }}" --region
    "{{ region }}" --container-instances "{{ containerInstanceArn }}" --status
    DRAINING
   ```

3. 컨테이너 인스턴스의 드레이닝이 완료되면 다음 명령을 실행하여 인스턴스 등록을 취소합니다.

   ```
   aws ecs deregister-container-instance --cluster "{{ cluster }}" --region
    "{{ region }}" --container-instance "{{ containerInstanceArn }}"
   ```

4. 다음 명령을 사용하여 SSM에서 인스턴스를 제거합니다.

```
aws ssm deregister-managed-instance --region "{{ region }}" --instance-id
 "{{ instanceId }}"
```

5. 인스턴스를 등록 취소한 후 온프레미스 서버 또는 VM에서 AWS 리소스를 정리하세요.

운영 체제	단계

Linux	a. 인스턴스에서 Amazon ECS 컨테이너 에이전트와 SSM Agent 서비스를 중지합니다.
	`sudo systemctl stop ecs amazon-ssm-agent`
	b. Amazon ECS 및 Systems Manager 패키지를 제거합니다.
	`sudo (yum/apt/zypper) remove amazon-ecs-init amazon-ssm-agent`
	c. 남은 디렉터리를 제거합니다.
	`sudo rm -rf /var/lib/ecs /etc/ecs / var/lib/amazon/ssm /var/log/ecs / var/log/amazon/ssm`

Windows	a. 인스턴스에서 Amazon ECS 컨테이너 에이전트와 SSM Agent 서비스를 중지합니다.

운영 체제	단계	

	`Stop-Service AmazonECS`
	
	`Stop-Service AmazonSSMAgent`
	
	b. Amazon ECS 패키지를 제거합니다.
	
	`.\ecs-anywhere-install.ps1 -Uninstall`	

외부 인스턴스의 AWS Systems Manager 에이전트 및 Amazon ECS 컨테이너 에이전트 업데이트

온프레미스 서버 또는 VM은 Amazon ECS 워크로드를 실행할 때 AWS Systems Manager 에이전트(SSM Agent) 및 Amazon ECS 컨테이너 에이전트를 모두 실행해야 합니다. AWS는 기능이 추가되거나 업데이트될 때 이러한 에이전트의 새 버전을 릴리스합니다. 외부 인스턴스에서 이전 버전의 에이전트를 사용하는 경우 다음 절차에 따라 에이전트를 업데이트할 수 있습니다.

외부 인스턴스에서 SSM Agent 업데이트

AWS Systems Manager는 인스턴스에서 SSM Agent를 업데이트하는 프로세스를 자동화하는 것이 좋습니다. 업데이트를 자동화하는 몇 가지 방법이 있습니다. 자세한 정보는 AWS Systems Manager 사용 설명서의 SSM Agent 업데이트 자동화를 참조하세요.

외부 인스턴스에서 Amazon ECS 에이전트 업데이트

외부 인스턴스에서 `ecs-init` 패키지를 업그레이드하여 Amazon ECS 컨테이너 에이전트를 업데이트합니다. Amazon ECS 에이전트를 업데이트해도 실행 중인 태스크나 서비스가 중단되지 않습니다. Amazon ECS는 각 리전의 Amazon S3 버킷에 `ecs-init` 패키지와 서명 파일을 제공합니다. `ecs-init` 버전 `1.52.1-1`을 시작할 때 Amazon ECS는 외부 인스턴스에서 사용하는 운영 체제 및 시스템 아키텍처에 따라 사용할 수 있는 별도의 `ecs-init` 패키지를 제공합니다.

다음 표를 사용하여 외부 인스턴스에서 사용하는 운영 체제 및 시스템 아키텍처를 기반으로 다운로드해야 하는 `ecs-init` 패키지를 결정합니다.

> **Note**
>
> 다음 명령을 사용하여 외부 인스턴스에서 사용하는 운영 체제 및 시스템 아키텍처를 결정할 수 있습니다.
>
> ```
> cat /etc/os-release
> uname -m
> ```

운영 체제(아키텍처)	ecs-init 패키지
CentOS 7(x86_64) CentOS 8(x86_64) CentOS Stream 9(x86_64) SUSE Enterprise Server 15(x86_64) RHEL 7(x86_64) RHEL 8 (x86_64)	`amazon-ecs-init-latest.x86_64.rpm`
CentOS 7(aarch64) CentOS 8(aarch64) CentOS Stream 9(aarch64) RHEL 7(aarch64)	`amazon-ecs-init-latest.aarch64.rpm`
Debian 9(x86_64) Debian 10(x86_64) Debian 11(x86_64) Debian 12(x86_64)	`amazon-ecs-init-latest.amd64.deb`

운영 체제(아키텍처)	ecs-init 패키지
Ubuntu 18(x86_64) Ubuntu 20(x86_64) Ubuntu 22(x86_64) Ubuntu 24(x86_64)	
Debian 9(aarch64) Debian 10(aarch64) Debian 11(aarch64) Debian 12(aarch64) Ubuntu 18(aarch64) Ubuntu 20(aarch64) Ubuntu 22(aarch64) Ubuntu 24(aarch64)	`amazon-ecs-init-latest.arm64.deb`

Amazon ECS 에이전트를 업데이트하려면 다음 단계를 따르세요.

Amazon ECS 에이전트를 업데이트하려면

1. 실행 중인 Amazon ECS 에이전트 버전을 확인합니다.

   ```
   curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
   ```

2. 운영 체제 및 시스템 아키텍처에 맞는 `ecs-init` 패키지를 다운로드합니다. Amazon ECS는 각 리전의 Amazon S3 버킷에 `ecs-init` 패키지 파일을 제공합니다. 명령의 *<region>* 식별자를 지리적으로 가장 가까운 리전 이름(예: `us-west-2`)으로 교체했는지 확인합니다.

   amazon-ecs-init-latest.x86_64.rpm

```
curl -o amazon-ecs-init.rpm https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.x86_64.rpm
```

amazon-ecs-init-latest.aarch64.rpm

```
curl -o amazon-ecs-init.rpm https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.aarch64.rpm
```

amazon-ecs-init-latest.amd64.deb

```
curl -o amazon-ecs-init.deb https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.amd64.deb
```

amazon-ecs-init-latest.arm64.deb

```
curl -o amazon-ecs-init.deb https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.arm64.deb
```

3. (선택 사항) PGP 서명을 사용하여 `ecs-init` 패키지 파일을 유효성을 확인합니다.

   a. GnuPG를 다운로드하고 설치합니다. GNUpg에 대한 자세한 정보는 GnuPG 웹 사이트를 참조하세요. Linux 시스템의 경우 원하는 Linux 패키지 관리자를 사용하여 gpg를 설치합니다.

   b. Amazon ECS PGP 퍼블릭 키를 가져옵니다.

   ```
   gpg --keyserver hkp://keys.gnupg.net:80 --recv BCE9D9A42D51784F
   ```

   c. `ecs-init` 패키지 서명을 다운로드합니다. 서명은 `.asc` 확장자 파일에 저장된 ASCII 분리 PGP 서명입니다. Amazon ECS는 각 리전의 Amazon S3 버킷에 서명 파일을 제공합니다. 명령의 *<region>* 식별자를 지리적으로 가장 가까운 리전 이름(예: `us-west-2`)으로 교체했는지 확인합니다.

   amazon-ecs-init-latest.x86_64.rpm

   ```
   curl -o amazon-ecs-init.rpm.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.x86_64.rpm.asc
   ```

   amazon-ecs-init-latest.aarch64.rpm

```
curl -o amazon-ecs-init.rpm.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.aarch64.rpm.asc
```

amazon-ecs-init-latest.amd64.deb

```
curl -o amazon-ecs-init.deb.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.amd64.deb.asc
```

amazon-ecs-init-latest.arm64.deb

```
curl -o amazon-ecs-init.deb.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.arm64.deb.asc
```

d. 키를 사용해 `ecs-init` 패키지 파일을 확인합니다.

`rpm` 패키지의 경우

```
gpg --verify amazon-ecs-init.rpm.asc ./amazon-ecs-init.rpm
```

`deb` 패키지의 경우

```
gpg --verify amazon-ecs-init.deb.asc ./amazon-ecs-init.deb
```

예상 출력은 다음과 같습니다.

```
gpg: Signature made Fri 14 May 2021 09:31:36 PM UTC
gpg:                using RSA key 50DECCC4710E61AF
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
     Subkey fingerprint: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

4. `ecs-init` 패키지를 설치합니다.

CentOS 7, CentOS 8, RHEL 7의 **`rpm`** 패키지의 경우

```
sudo yum install -y ./amazon-ecs-init.rpm
```

SUSE Enterprise Server 15의 **rpm** 패키지의 경우

```
sudo zypper install -y --allow-unsigned-rpm ./amazon-ecs-init.rpm
```

deb 패키지의 경우

```
sudo dpkg -i ./amazon-ecs-init.deb
```

5. ecs 서비스를 다시 시작합니다.

```
sudo systemctl restart ecs
```

6. Amazon ECS 에이전트 버전이 업데이트되었는지 확인합니다.

```
curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
```

Amazon ECS 클러스터 업데이트

다음 클러스터 속성을 수정할 수 있습니다.

- 기본 용량 공급자를 설정합니다.

  각 클러스터에는 하나 이상의 용량 공급자와 하나의 선택적 용량 공급자 전략이 있을 수 있습니다. 용량 공급자 전략은 태스크가 클러스터의 용량 공급자에 분배되는 방식을 결정합니다. 표준 실행 작업을 실행하거나 서비스를 생성할 때 클러스터의 기본 용량 공급자 전략 또는 기본 전략을 재정의하는 용량 공급자 전략을 사용할 수 있습니다.

- Container Insights를 설정합니다.

  CloudWatch Container Insights는 컨테이너 애플리케이션 및 마이크로서비스의 지표 및 로그를 수집하고, 종합하며, 요약합니다. 또한 Container Insights는 컨테이너 재시작 오류 등의 진단 정보를 제공합니다. 이 정보를 사용하여 문제를 격리하고 신속하게 해결할 수 있습니다. 자세한 정보는 the section called “관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링”을 참조하세요.

- 클러스터를 식별하는 데 도움이 되는 태그를 추가합니다.

절차

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
4. 클러스터: ***name*** 페이지에서 클러스터 업데이트를 선택합니다.
5. 기본 용량 공급자를 설정하려면 기본 용량 공급자 전략에서 더 추가를 선택합니다.

   a. 용량 공급자에서 용량 공급자를 선택합니다.

   b. (선택 사항) 기본에 용량 공급자에서 실행되는 최소 작업 수를 입력합니다.

      기본 값은 하나의 용량 공급자에 대해서만 설정할 수 있습니다.

   c. (선택 사항) 가중치에 지정된 용량 공급자를 사용하는 시작된 총 작업 수의 상대 백분율을 입력합니다.

   d. (선택 사항) 추가 용량 공급자가 있으면 단계를 반복합니다.
6. Container Insights를 켜려나 끄려면 모니터링을 확장한 다음 Container Insights 사용을 켭니다.
7. 클러스터를 식별하려면 태그를 확장한 다음, 태그를 구성합니다.

   [태그 추가] 태그 추가(Add tag)를 선택하고 다음을 수행합니다.

   - 키(Key)에 키 이름을 입력합니다.
   - 값에 키 값을 입력합니다.

   [태그 제거] 태그의 키와 값 오른쪽에 있는 제거를 선택합니다.
8. 업데이트를 선택합니다.

Amazon ECS 클러스터 삭제

클러스터 사용을 마치면 이를 삭제할 수 있습니다. 클러스터를 삭제하면 `INACTIVE` 상태로 전환됩니다. `INACTIVE` 상태인 클러스터는 일정 기간 동안 계정에서 검색 가능한 상태로 유지될 수 있습니다. 하지만 이 동작은 향후 변경될 수 있으므로 `INACTIVE` 클러스터가 지속되는 상태에 의존해서는 안 됩니다.

클러스터를 삭제하려면 다음 작업을 수행해야 합니다.

- 클러스터의 모든 서비스를 삭제합니다. 자세한 내용은 the section called “서비스 삭제” 섹션을 참조하세요.
- 현재 실행 중인 태스크를 모두 중지합니다. 자세한 내용은 the section called “작업 중지” 섹션을 참조하세요.
- 클러스터에 등록된 모든 컨테이너 인스턴스를 등록 취소합니다. 자세한 내용은 the section called “컨테이너 인스턴스 등록 취소” 섹션을 참조하세요.
- 네임스페이스를 삭제합니다. 자세한 내용은 AWS Cloud Map 개발자 안내서의 네임스페이스 삭제를 참조하세요.

절차

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터 페이지에서 삭제할 클러스터를 선택합니다.
5. 페이지 오른쪽 상단에서 클러스터 삭제를 선택합니다.

   클러스터와 연결된 모든 리소스를 삭제하지 않은 경우 메시지가 표시됩니다.
6. 확인란에 delete ***cluster name***를 입력합니다.

Amazon ECS 컨테이너 인스턴스 등록 취소

> **Important**
>
> 이 주제는 Amazon EC2에서 생성된 컨테이너 인스턴스에만 적용됩니다. 외부 인스턴스의 등록 취소에 대한 자세한 정보는 Amazon ECS 외부 인스턴스 등록 취소 섹션을 참조하세요.

Amazon EC2 지원 컨테이너 인스턴스 태스크를 마치면 클러스터에서 컨테이너 인스턴스 등록을 취소해야 합니다. 등록 취소 후 컨테이너 인스턴스는 더 이상 새 태스크를 받을 수 없습니다.

등록 취소 시 컨테이너 인스턴스에서 실행 중인 태스크가 있는 경우, 이러한 태스크는 인스턴스를 종료할 때까지 계속 실행되거나 몇 가지 다른 수단을 통해 중지됩니다. 그러나 이러한 태스크는 분리되어 Amazon ECS에서 더 이상 모니터링하거나 고려하지 않습니다. 컨테이너 인스턴스에서 분리된 태스크가 Amazon ECS 서비스의 일부인 경우, 서비스 스케줄러는 가능하다면 다른 컨테이너 인스턴스에서

해당 태스크의 다른 사본을 시작합니다. Application Load Balancer 대상 그룹에 등록된 분리된 서비스 작업의 모든 컨테이너의 등록이 취소됩니다. 이러한 컨테이너는 로드 밸런서 또는 대상 그룹에서의 설정에 따라 Connection Draining을 시작합니다. 분리된 작업이 awsvpc 네트워크 모드를 사용 중인 경우 탄력적 네트워크 인터페이스가 삭제됩니다.

등록 취소 후 컨테이너 인스턴스를 다른 목적으로 사용하려는 경우, 등록 취소 전에 컨테이너 인스턴스에서 실행 중인 모든 태스크를 중지해야 합니다. 이렇게 하면 분리된 태스크가 리소스를 소모하는 것을 방지합니다.

컨테이너 인스턴스의 등록을 취소할 때는 다음 사항을 고려해야 합니다.

- 각각의 컨테이너 인스턴스에는 고유의 상태 정보가 있기 때문에 클러스터에서 등록을 취소하고 다른 클러스터에 다시 등록해서는 안 됩니다. 컨테이너 인스턴스 리소스의 위치를 바꾸려면 클러스터에서 컨테이너 인스턴스를 종료한 다음, 새 클러스터에서 새 컨테이너 인스턴스를 시작하는 것이 좋습니다. 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 종료 및 Amazon ECS Linux 컨테이너 인스턴스 시작 섹션을 참조하세요.
- 컨테이너 인스턴스를 Auto Scaling 그룹 또는AWS CloudFormation 스택에서 관리하는 경우, Auto Scaling 그룹 또는 AWS CloudFormation 스택을 업데이트하여 인스턴스를 종료합니다. 그렇지 않은 경우 사용자가 인스턴스를 종료한 후 Auto Scaling 그룹 또는 AWS CloudFormation이 새 인스턴스를 생성합니다.
- Amazon ECS 컨테이너 에이전트가 연결된 상태로 실행 중인 컨테이너 인스턴스를 종료하는 경우, 에이전트는 자동으로 인스턴스를 클러스터에서 등록 취소합니다. 에이전트 연결이 끊긴 상태로 중지된 컨테이너 인스턴스 또는 인스턴스는 종료 시 자동으로 등록 취소되지 않습니다.
- 컨테이너 인스턴스의 등록을 취소하면 클러스터에서 인스턴스는 제거되지만 Amazon EC2 인스턴스는 종료되지 않습니다. 인스턴스 사용이 끝나면 인스턴스를 종료해 요금이 청구되지 않도록 해야 합니다. 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 종료를 참조하세요.

절차

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 외부 인스턴스가 등록되어 있는 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택하고 인스턴스를 호스팅하는 클러스터를 선택합니다.
4. 클러스터: ***name***(Cluster : name) 페이지에서 인프라(Infrastructure) 탭을 선택합니다.
5. 컨테이너 인스턴스(Container instances)에서 등록 취소하려는 인스턴스 ID를 선택합니다. 컨테이너 인스턴스 세부 정보 페이지로 리디렉션됩니다.
6. 컨테이너 인스턴스 : ***id*** 페이지에서 등록 취소(Deregister)를 선택합니다.

7. 확인 화면에서 Deregister(등록 취소)를 선택합니다.
8. 컨테이너 인스턴스 작업이 완료되면 기본 Amazon EC2 인스턴스를 종료합니다. 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 종료를 참조하세요.

Amazon ECS 컨테이너 인스턴스 드레이닝

예를 들어 시스템 업데이트를 수행하거나 클러스터 용량을 스케일 다운하기 위해 클러스터에서 컨테이너 인스턴스를 제거해야 하는 경우가 있을 수 있습니다. Amazon ECS는 컨테이너 인스턴스를 `DRAINING` 상태로 전환할 수 있는 기능을 제공합니다. 이것을 컨테이너 인스턴스 드레이닝이라고 합니다. 컨테이너 인스턴스를 `DRAINING`으로 설정할 경우 Amazon ECS는 새 작업이 컨테이너 인스턴스에서 배치를 위해 예약되지 않도록 합니다.

서비스에 대한 드레이닝 동작

`PENDING` 상태에 있는 서비스의 일부인 모든 태스크가 즉시 중지됩니다. 클러스터에 사용 가능한 컨테이너 인스턴스 용량이 있는 경우 서비스 스케줄러는 교체 태스크를 시작합니다. 컨테이너 인스턴스 용량이 충분하지 않으면 문제를 나타내는 서비스 이벤트 메시지가 전송됩니다.

컨테이너 인스턴스에 있는 서비스의 일부인 `RUNNING` 상태의 태스크가 `STOPPED` 상태로 전환됩니다. 서비스 스케줄러는 서비스의 배포 유형 및 배포 구성 파라미터인 `minimumHealthyPercent` 및 `maximumPercent`에 따라 작업을 교체하려고 시도합니다. 자세한 내용은 Amazon ECS 서비스 및 Amazon ECS 서비스 정의 파라미터(을)를 참조하세요.

- 만약 `minimumHealthyPercent`가 100% 미만이면 스케줄러는 태스크 대체 도중 `desiredCount`를 일시적으로 무시할 수 있습니다. 예를 들어 `desiredCount`가 4개 작업인 경우, 50%의 최솟값은 스케줄러가 새로운 2개 태스크를 시작하기 전에 2개의 기존 태스크를 중지하도록 허용합니다. 최솟값이 100%인 경우, 서비스 스케줄러는 대체 태스크가 이 정상 상태라고 간주될 때까지 기존 태스크를 제거할 수 없습니다. 로드 밸런서를 사용하지 않는 서비스의 태스크는 `RUNNING` 상태일 경우에 정상 상태로 간주됩니다. 로드 밸런서를 사용하는 서비스의 태스크는 `RUNNING` 상태이고 태스크가 호스팅된 컨테이너 인스턴스가 로드 밸런서에 의해 정상 상태로 보고되는 경우에 정상 상태로 간주됩니다.

> **Important**
>
> Spot Instances를 사용하고 `minimumHealthyPercent`가 100% 이상인 경우 Spot Instance 종료 전에 서비스에서 작업을 교체할 시간이 충분하지 않습니다.

- `maximumPercent` 파라미터는 작업 대체 도중 실행 중인 작업 수의 상한을 나타내며, 이를 통해 대체 배치 크기를 정의할 수 있습니다. 예를 들어 `desiredCount`가 4개 태스크인 경우, 200%의 최댓값은 드레이닝할 4개 태스크를 중지하기 전에 4개의 새 태스크를 시작합니다(이렇게 하는 데 필요한 클러스터 리소스를 사용할 수 있는 경우). 최댓값이 100%인 경우, 드레이닝 작업이 중지될 때까지 대체 태스크를 시작할 수 없습니다.

> **Important**
>
> `minimumHealthyPercent`와 `maximumPercent`가 모두 100%이면 서비스에서 기존 태스크를 제거할 수 없으며 교체 태스크도 시작할 수 없습니다. 그러면 컨테이너 인스턴스 드레이닝이 실패하고 새 배포가 만들어지지 않습니다.

독립 실행형 태스크에 대한 드레이닝 동작

`PENDING` 또는 `RUNNING` 상태의 모든 독립 실행형 태스크는 영향을 받지 않습니다. 스스로 중지할 때까지 기다리거나 수동으로 중지해야 합니다. 컨테이너 인스턴스는 `DRAINING` 상태를 유지합니다.

컨테이너 인스턴스는 인스턴스에서 실행 중인 모든 태스크가 `STOPPED` 상태로 전환되면 드레이닝을 완료합니다. 컨테이너 인스턴스는 다시 활성화되거나 삭제될 때까지 `DRAINING` 상태를 유지합니다. `containerInstance` 파라미터와 함께 ListTasks 태스크를 사용하여 컨테이너 인스턴스에 있는 태스크의 상태를 확인하고, 상태를 확인할 각 태스크의 Amazon 리소스 이름(ARN) 또는 ID와 함께 DescribeTasks 태스크를 사용하여 인스턴스에 있는 태스크 목록을 가져옵니다.

컨테이너 인스턴스가 태스크를 다시 호스팅할 준비가 되면 컨테이너 인스턴스의 상태를 `DRAINING`에서 `ACTIVE`로 변경합니다. 그러면 Amazon ECS 서비스 스케줄러는 컨테이너 인스턴스를 다시 작업 배치로 간주합니다.

절차

다음 단계를 사용하여 컨테이너 인스턴스를 새 AWS Management Console을 사용한 드레이닝으로 설정할 수 있습니다.

UpdateContainerInstancesState API 작업 또는 update-container-instances-state 명령을 사용하여 컨테이너 인스턴스의 상태를 `DRAINING`으로 변경할 수도 있습니다.

AWS Management Console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.

2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터(Clusters)페이지에서 인스턴스를 호스팅하는 클러스터를 선택합니다.
4. 클러스터: ***name***(Cluster : name) 페이지에서 인프라(Infrastructure) 탭을 선택합니다. 컨테이너 인스턴스(Container instances)에서 드레이닝할 각 컨테이너 인스턴스의 확인란을 선택합니다.
5. Actions(작업), Drain(드레이닝)을 차례로 선택합니다.

Amazon EC2 컨테이너 인스턴스

Amazon ECS 에이전트는 클러스터에 등록된 모든 컨테이너 인스턴스에서 실행되는 프로세스입니다. 컨테이너 인스턴스와 Amazon ECS 간의 통신을 용이하게 합니다.

> **Note**
>
> Linux 컨테이너 인스턴스에서 에이전트 컨테이너는 `/lib`, `/lib64` 및 `/proc`과 같은 최상위 디렉터리를 탑재합니다. 이는 Amazon EBS 볼륨, `awsvpc` 네트워크 모드, Amazon ECS Service Connect 및 Amazon ECS용 FireLens 같은 ECS 기능에 필요합니다.

각 Amazon ECS 컨테이너 에이전트 버전은 서로 다른 기능 세트를 지원하며 이전 버전에 대한 버그 수정을 제공합니다. 기능한 한 최신 버전의 Amazon ECS 컨테이너 에이전트를 사용할 것이 좋습니다. 컨테이너 에이전트를 최신 버전으로 업데이트하려면 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

Amazon ECS 컨테이너 에이전트에는 `amazon-ecs-pause` 이미지가 포함되어 있습니다.Amazon ECS는 `awsvpc` 네트워크 모드를 사용하는 작업에 이 이미지를 사용합니다.

각 에이전트 릴리스에 포함된 기능 및 기능 향상을 보려면 https://github.com/aws/amazon-ecs-agent/releases를 참조하세요.

> **Important**
>
> 신뢰할 수 있는 지표를 위한 최소 Docker 버전은 Docker 버전 `v20.10.13` 이상이며, Amazon ECS 최적화 AMI `20220607` 이상 버전에 포함되어 있습니다.
> Amazon ECS 에이전트 버전 `1.20.0` 이상에서는 `18.01.0` 이전 Docker 버전에 대한 지원이 중단되었습니다.

수명 주기

Amazon ECS 컨테이너 에이전트가 Amazon EC2 인스턴스를 클러스터에 등록하면 Amazon EC2 인스턴스는 상태를 ACTIVE로, 에이전트 연결 상태를 TRUE로 보고합니다. 이 컨테이너 인스턴스는 작업 실행 요청을 수락할 수 있습니다.

ACTIVE 컨테이너 인스턴스를 중지(종료가 아님)하면 상태는 계속 이지만 에이전트 연결 상태는 몇 분 안에 FALSE로 전환됩니다. 컨테이너 인스턴스에서 실행 중이던 모든 작업이 중지됩니다. 컨테이너 인스턴스를 다시 시작하면 컨테이너 에이전트는 Amazon ECS 서비스에 다시 연결되고 인스턴스에서 다시 태스크를 실행할 수 있습니다.

> **Important**
>
> 컨테이너 인스턴스를 중지했다가 시작하거나 해당 인스턴스를 재부팅하는 경우, 일부 구버전 Amazon ECS 컨테이너 에이전트는 원래의 컨테이너 인스턴스 ID를 등록 취소하지 않고 해당 인스턴스를 다시 등록합니다. 이 경우 Amazon ECS는 클러스터에 실제 있는 것보다 많은 컨테이너 인스턴스를 나열합니다. (동일한 Amazon EC2 인스턴스 ID에 대해 중복된 컨테이너 인스턴스 ID가 있는 경우, ACTIVE로 나열되고 에이전트 연결 상태가 FALSE인 중복을 안전하게 등록 취소할 수 있습니다.) 이 문제는 최신 버전 Amazon ECS 컨테이너 에이전트에서 수정되었습니다. 현재 버전 업데이트에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

컨테이너 인스턴스의 상태를 DRAINING로 변경하면 컨테이너 인스턴스에서 새 작업이 배치되지 않습니다. 시스템 업데이트를 수행할 수 있도록 컨테이너 인스턴스에서 실행 중인 모든 서비스 태스크는 가능한 경우 제거됩니다. 자세한 정보는 Amazon ECS 컨테이너 인스턴스 드레이닝 섹션을 참조하세요.

컨테이너 인스턴스 등록을 취소하거나 종료하면 해당 컨테이너 인스턴스 상태가 곧바로 INACTIVE로 바뀌며, 컨테이너 인스턴스를 나열할 때 해당 컨테이너 인스턴스가 더 이상 보고되지 않습니다. 하지만 종료 후 한 시간 동안은 여전히 컨테이너 인스턴스 설명이 가능합니다. 한 시간이 지나면 더 이상 인스턴스 설명을 사용할 수 없습니다.

> **Important**
>
> 인스턴스를 수동으로 드레이닝하거나 Auto Scaling 그룹 수명 주기 후크를 구축하여 인스턴스 상태를 DRAINING으로 설정할 수 있습니다. Auto Scaling 수명 주기 후크에 대한 자세한 정보는 Amazon EC2 Auto Scaling 수명 주기 후크를 참조하세요.

Docker 지원

Amazon ECS는 Amazon Linux에 게시된 Docker의 마지막 두 메이저 버전을 지원합니다. 현재 여기에는 Docker 20.10.x 및 Docker 25.x가 포함됩니다.

Amazon ECS에 필요한 최소 Docker 버전은 GitHub의 Amazon ECS Agent specification file에서 찾을 수 있습니다.

Amazon ECS 최적화 AMI를 사용하는 경우 Docker는 Amazon ECS 컨테이너 에이전트와 함께 작동하도록 사전 설치 및 구성됩니다. AMI는 Amazon ECS에서 테스트하고 지원하는 Docker 버전을 포함합니다.

> **Note**
>
> Amazon ECS는 여러 Docker 버전을 지원하지만 최상의 호환성과 지원을 위해 Amazon ECS 최적화 AMI와 함께 제공되는 Docker 버전을 사용하는 것이 좋습니다.

Amazon ECS 최적화 AMI

Amazon ECS 최적화 AMI의 Linux 변형은 Amazon Linux 2 AMI를 기반으로 사용합니다. Systems Manager Parameter Store API를 쿼리하여 각 변형에 대한 Amazon Linux 2 소스 AMI 이름을 검색할 수 있습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 메타데이터 검색 섹션을 참조하세요. 최신 Amazon ECS 최적화 Amazon Linux 2 AMI에서 컨테이너 인스턴스를 시작하면 현재 버전의 컨테이너 에이전트를 받을 수 있습니다. 최신 Amazon ECS 최적화 Amazon Linux 2 AMI로 컨테이너 인스턴스를 시작하는 방법은 Amazon ECS Linux 컨테이너 인스턴스 시작를 참조하세요.

추가 정보

다음 페이지에서는 변경 사항에 대한 추가 정보를 제공합니다.

- GitHub의 Amazon ECS Agent changelog
- `ecs-init` 애플리케이션의 소스 코드 및 에이전트 패키징을 위한 스크립트 및 구성은 이제 에이전트 리포지토리의 일부입니다. `ecs-init`의 이전 버전 및 패키징에 대해서는 GitHub의 Amazon ecs-init changelog를 참조하세요.
- Amazon Linux 2 릴리스 정보
- Docker 설명서의 Docker Engine 릴리스 정보

- NVIDIA 설명서의 NVIDIA 드라이버 설명서

Amazon ECS 컨테이너 에이전트 구성

적용 대상: EC2 인스턴스

Amazon ECS 컨테이너 에이전트는 다양한 구성 옵션을 지원하며 대부분은 환경 변수를 통해 설정합니다.

Amazon ECS 최적화 AMI의 Linux 변형을 사용하여 컨테이너 인스턴스를 시작한 경우, 이들 환경 변수를 `/etc/ecs/ecs.config` 파일에서 설정한 후 에이전트를 다시 시작할 수 있습니다. 실행 시 Amazon EC2 사용자 데이터를 사용하여 컨테이너 인스턴스에 구성 변수를 작성할 수도 있습니다. 자세한 내용은 데이터 전달을 위한 Amazon ECS Linux 컨테이너 인스턴스 부트스트래핑 섹션을 참조하세요.

컨테이너 인스턴스가 Amazon ECS 최적화 AMI의 Windows 변형으로 시작된 경우 PowerShell SetEnvironmentVariable 명령을 사용하여 이러한 환경 변수를 설정한 다음 에이전트를 다시 시작할 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 사용자 데이터 입력을 사용하여 EC2 인스턴스를 시작할 때 명령 실행과 the section called “컨테이너 인스턴스 부트스트래핑”를 참조하세요.

수동으로 Amazon ECS 컨테이너 에이전트(비 Amazon ECS 최적화 AMI 이외 용도)를 시작하는 경우 에이전트를 시작하는 데 사용하는 docker run 명령에서 이 환경 변수를 사용할 수 있습니다. `--env=`*`VARIABLE_NAME`*`=`*`VARIABLE_VALUE`* 구문에 이러한 변수를 사용합니다. 프라이빗 리포지토리의 인증 자격 증명과 같은 민감한 정보의 경우, 에이전트 환경 변수를 파일에 저장하고 `--env-file` *`path_to_env_file`* 옵션을 사용하여 한 번에 모두 전달해야 합니다. 이러한 명령을 사용하여 변수를 추가할 수 있습니다.

```
sudo systemctl stop ecs
sudo vi /etc/ecs/ecs.config
# And add the environment variables with VARIABLE_NAME=VARIABLE_VALUE format.
sudo systemctl start ecs
```

호스트 PID 네임스페이스로 Amazon ECS 에이전트 실행

기본적으로 Amazon ECS 에이전트는 자체 PID 네임스페이스로 실행됩니다. 다음 구성에서 호스트 PID 네임스페이스로 실행되도록 Amazon ECS 에이전트를 구성할 수 있습니다.

- SELinux 적용 모드가 활성화되었습니다.
- Docker의 SELinux 보안 정책이 true로 설정되었습니다.

/etc/ecs/ecs.config 파일에서 ECS_AGENT_PID_NAMESPACE_HOST 환경 변수를 true로 설정하여 이 동작을 구성할 수 있습니다. 이 변수가 활성화되면 ecs-init는 호스트의 PID 네임스페이스(--pid=host)로 Amazon ECS 에이전트 컨테이너를 시작하여 에이전트가 SELinux 적용 환경에서 올바르게 부트스트랩할 수 있도록 합니다. 이 기능은 Amazon ECS 에이전트 버전 1.94.0 이상에서 사용할 수 있습니다.

이 기능을 활성화하려면 /etc/ecs/ecs.config 파일에 다음 줄을 추가하세요.

```
ECS_AGENT_PID_NAMESPACE_HOST=true
```

이렇게 변경한 후 Amazon ECS 에이전트를 다시 시작하여 변경 사항을 적용하세요.

```
sudo systemctl restart ecs
```

SELinux 적용 모드가 활성화되어 있고 Docker 보안 정책이 true로 설정되어 있는 경우 ECS_AGENT_PID_NAMESPACE_HOST=true가 설정되어 있어도 다음 기능은 작동하지 않습니다.

- Amazon ECS Exec
- Amazon EBS 태스크 연결
- Service Connect
- Amazon ECS용 FireLens

사용 가능한 파라미터

사용 가능한 Amazon ECS 컨테이너 에이전트 구성 파라미터에 대한 자세한 정보는 GitHub의 Amazon ECS Container Agent를 참조하세요.

Amazon S3에 Amazon ECS 컨테이너 인스턴스 구성 저장

Amazon ECS 컨테이너 에이전트 구성은 환경 변수로 제어됩니다. Amazon ECS 최적화 AMI의 Linux 변형은 컨테이너 에이전트가 시작될 때 /etc/ecs/ecs.config에서 이들 변수를 확인하여 그에 따라 에이전트를 구성합니다. ECS_CLUSTER와 같이 민감하지 않은 환경 변수는 시작 시 Amazon EC2 사용자 데이터를 통해 컨테이너 인스턴스로 전달되고 아무 영향 없이 이 파일에 기록될 수 있습니다. 하지만 AWS 자격 증명 또는 ECS_ENGINE_AUTH_DATA 변수와 같은 기타 민감한 정보는 사용자 데이터에서 인스턴스로 전달되거나 .bash_history 파일에 표시되도록 허용할 /etc/ecs/ecs.config에 기록되어서는 절대 안 됩니다.

Amazon S3에서 구성 정보를 프라이빗 버킷에 저장하고 컨테이너 인스턴스 IAM 역할에 읽기 전용 액세스를 부여하면 시작 시 컨테이너 인스턴스 구성을 안전하고 편리하게 허용할 수 있습니다. 프라이빗 버킷에 `ecs.config` 파일의 사본을 저장할 수 있습니다. 그런 다음 Amazon EC2 사용자 데이터를 사용하여 AWS CLI를 설치하고 인스턴스가 시작될 때 구성 정보를 `/etc/ecs/ecs.config`에 복사할 수 있습니다.

Amazon S3에 **`ecs.config`** 파일을 저장하는 방법

1. Amazon S3에 대해 읽기 전용으로 액세스하려면 컨테이너 인스턴스 역할(ecsInstanceRole)에 권한을 부여해야 합니다. AmazonS3ReadOnlyAccess를 `ecsInstanceRole` 역할에 할당하면 됩니다. 역할에 정책을 연결하는 방법에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 역할 권한 업데이트를 참조하세요.
2. 다음 형식을 사용하여 유효한 Amazon ECS 에이전트 구성 변수를 포함하는 `ecs.config` 파일을 생성합니다. 이 예제에서는 프라이빗 레지스트리 인증을 구성합니다. 자세한 내용은 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용 섹션을 참조하세요.

   ```
   ECS_ENGINE_AUTH_TYPE=dockercfg
   ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
   {"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
   ```

   > **Note**
   >
   > 사용 가능한 Amazon ECS 에이전트 구성 변수의 전체 목록은 GitHub의 Amazon ECS Container Agent를 참조하세요.

3. 구성 파일을 저장하려면 Amazon S3에 프라이빗 버킷을 만듭니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서에서 버킷 생성을 참조하세요.
4. `ecs.config` 파일을 S3 버킷에 업로드합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 업로드를 참조하세요.

시작 시 Amazon S3에서 **`ecs.config`** 파일을 로드하려면

1. 이 섹션에서 앞서 설명한 절차를 완료하여 컨테이너 인스턴스에 읽기 전용 Amazon S3 액세스를 부여하고 프라이빗 S3 버킷에 `ecs.config` 파일을 저장합니다.
2. 새 컨테이너 인스턴스를 시작하고 EC2 사용자 데이터에서 다음 예제 스크립트를 사용합니다. 스크립트는 AWS CLI를 설치하고 구성 파일을 `/etc/ecs/ecs.config`에 복사합니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 시작 섹션을 참조하세요.

```
#!/bin/bash
yum install -y aws-cli
aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
```

Amazon ECS 컨테이너 에이전트 설치

Amazon ECS 클러스터에 Amazon EC2 인스턴스를 등록하려고 하는데 해당 인스턴스가 Amazon ECS 최적화 AMI 기반 AMI를 사용하지 않는 경우에는 다음 절차에 따라 Amazon ECS 컨테이너 에이전트를 수동으로 설치할 수 있습니다. 이렇게 하려면 리전의 Amazon S3 버킷 중 하나 또는 Amazon Elastic Container Registry Public에서 에이전트를 다운로드하면 됩니다. 리전 Amazon S3 버킷 중 하나에서 다운로드하는 경우 선택적으로 PGP 서명을 사용하여 컨테이너 에이전트 파일의 유효성을 확인할 수 있습니다.

> **Note**
>
> Amazon ECS 및 도커 서비스 모두를 위한 `systemd` 단위에는 이 두 서비스를 시작하기 전에 완료하기 위해 `cloud-init`를 대기하는 명령이 있습니다. `cloud-init` 프로세스는 Amazon EC2 사용자 데이터의 실행이 완료될 때까지 완료된 것으로 간주되지 않습니다. 따라서 Amazon EC2 사용자 데이터를 통해 Amazon ECS 또는 도커를 시작하면 교착 상태가 발생할 수 있습니다. Amazon EC2 사용자 데이터를 사용하여 컨테이너 에이전트를 시작하려면 `systemctl enable --now --no-block ecs.service`를 사용할 수 있습니다.

비 Amazon Linux EC2 에이전트에 Amazon ECS 컨테이너 에이전트 설치

Amazon EC2 인스턴스에 Amazon ECS 컨테이너 에이전트를 설치하려면 리전 Amazon S3 버킷 중 하나에서 에이전트를 다운로드하여 설치할 수 있습니다.

> **Note**
>
> 비 Amazon Linux AMI를 사용할 때 Amazon EC2 인스턴스는 `cgroupfs` 드라이버에 대한 `cgroup` 지원이 있어야 Amazon ECS 에이전트가 작업 수준 리소스 제한을 지원할 수 있습니다. 자세한 내용은 GitHub의 Amazon ECS 에이전트를 참조하세요.

참조용으로 아래에 각 시스템 아키텍처에 대해 최신 Amazon ECS 컨테이너 에이전트 파일이 리전별로 나열되어 있습니다.

리전	리전 이름	Amazon ECS init deb 파일	Amazon ECS init rpm 파일
us-east-2	미국 동부(오하이오)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
us-east-1	미국 동부(버지니아 북부)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
us-west-1	미국 서부(캘리포니아 북부)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
us-west-2	미국 서부(오리건)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
ap-east-1	아시아 태평양(홍콩)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
ap-northeast-1	아시아 태평양(도쿄)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)

리전	리전 이름	Amazon ECS init deb 파일	Amazon ECS init rpm 파일
ap-northeast-2	아시아 태평양(서울)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
ap-south-1	아시아 태평양(뭄바이)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
ap-southeast-1	아시아 태평양(싱가포르)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
ap-southeast-2	아시아 태평양(시드니)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
ca-central-1	캐나다(중부)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
eu-central-1	유럽(프랑크푸르트)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)

리전	리전 이름	Amazon ECS init deb 파일	Amazon ECS init rpm 파일
eu-west-1	유럽(아일랜드)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
eu-west-2	유럽(런던)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
eu-west-3	유럽(파리)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
sa-east-1	남아메리카(상파울루)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64 Amazon ECS init aarch64(aarch64)
us-gov-east-1	AWS GovCloud(미국 동부)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)
us-gov-west-1	AWS GovCloud(미국 서부)	Amazon ECS init amd64(amd64) Amazon ECS init arm64(arm64)	Amazon ECS init x86_64(x86_64) Amazon ECS init aarch64(aarch64)

비 Amazon Linux AMI를 사용하는 Amazon EC2 인스턴스에 Amazon ECS 컨테이너 에이전트를 설치하는 방법

1. Amazon ECS에 액세스할 수 있는 IAM 역할로 Amazon EC2 인스턴스를 시작합니다. 자세한 정보는 Amazon ECS 컨테이너 인스턴스 IAM 역할을 참조하세요.
2. 인스턴스에 연결합니다.
3. 인스턴스에 최신 버전의 Docker를 설치합니다.
4. 시스템이 최소 버전 요구 사항을 충족하는지 Docker 버전을 확인하세요.

   > **Note**
   >
   > 신뢰할 수 있는 지표를 위한 최소 Docker 버전은 Docker 버전 `v20.10.13` 이상이며, Amazon ECS 최적화 AMI `20220607` 이상 버전에 포함되어 있습니다.
   > Amazon ECS 에이전트 버전 `1.20.0` 이상에서는 `18.01.0` 이전 Docker 버전에 대한 지원이 중단되었습니다.

   ```
   docker --version
   ```

5. 운영 체제 및 시스템 아키텍처에 적합한 Amazon ECS 에이전트 파일을 다운로드하여 설치합니다.

   `deb` 아키텍처의 경우:

   ```
   ubuntu:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-ecs-agent-us-west-2/amazon-ecs-init-latest.amd64.deb
   ubuntu:~$ sudo dpkg -i amazon-ecs-init-latest.amd64.deb
   ```

   `rpm` 아키텍처의 경우:

   ```
   fedora:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-ecs-agent-us-west-2/amazon-ecs-init-latest.x86_64.rpm
   fedora:~$ sudo yum localinstall -y amazon-ecs-init-latest.x86_64.rpm
   ```

6. `/lib/systemd/system/ecs.service` 파일을 편집하고 `[Unit]` 섹션의 끝에 다음 줄을 추가합니다.

   ```
   After=cloud-final.service
   ```

7. (선택 사항) 인스턴스를 `default` 클러스터가 아닌 클러스터에 등록하려면, `/etc/ecs/ecs.config` 파일을 편집하고 다음 내용을 추가하세요. 다음 예제는 `MyCluster` 클러스터를 지정합니다.

```
ECS_CLUSTER=MyCluster
```

이들을 비롯한 기타 에이전트 런타임 옵션에 대한 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

> **Note**
>
> 필요할 경우 (시작 시 Amazon EC2 사용자 데이터를 사용하여 컨테이너 인스턴스에 다운로드할 수 있는) Amazon S3에 에이전트 환경 변수를 저장할 수 있습니다. 프라이빗 리포지토리의 인증 자격 증명과 같은 민감한 정보에는 이 방법을 권장합니다. 자세한 내용은 Amazon S3에 Amazon ECS 컨테이너 인스턴스 구성 저장 및 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용 단원을 참조하세요.

8. `ecs` 서비스를 시작합니다.

```
ubuntu:~$ sudo systemctl start ecs
```

호스트 네트워크 모드로 Amazon ECS 에이전트 실행

Amazon ECS 컨테이너 에이전트를 실행하는 경우 `ecs-init`는 `host` 네트워크 모드로 컨테이너 에이전트 컨테이너를 만듭니다. 이 모드는 컨테이너 에이전트 컨테이너에서 유일하게 지원되는 네트워크 모드입니다.

이렇게 하면 컨테이너 에이전트가 시작한 컨테이너가 Amazon EC2 인스턴스 메타데이터 서비스 엔드포인트(`http://169.254.169.254`)에 액세스하는 것을 차단할 수 있습니다. 이 경우 컨테이너가 컨테이너 인스턴스 프로필에서 IAM 역할 자격 증명에 액세스할 수 없으며 해당 작업이 IAM 태스크 역할 자격 증명만 사용합니다. 자세한 정보는 Amazon ECS 작업 IAM 역할을 참조하세요.

이렇게 하면 컨테이너 에이전트가 `docker0` 브리지에서 연결 및 네트워크 트래픽을 위해 경합하지 않습니다.

Amazon ECS 컨테이너 에이전트 로그 구성 파라미터

Amazon ECS 컨테이너 에이전트는 컨테이너 인스턴스에 로그를 저장합니다.

컨테이너 에이전트 버전 1.36.0 이상에서는 기본적으로 로그가 Linux 인스턴스의 `/var/log/ecs/ecs-agent.log`와 Windows 인스턴스의 `C:\ProgramData\Amazon\ECS\log\ecs-agent.log`에 있습니다.

컨테이너 에이전트 버전 1.35.0 및 이전 버전의 경우 기본적으로 로그는 Linux 인스턴스의 `/var/log/ecs/ecs-agent.log.`*`timestamp`*와 Windows 인스턴스의 `C:\ProgramData\Amazon\ECS\log\ecs-agent.log.`*`timestamp`*에 있습니다.

기본적으로 에이전트 로그는 매시간 교체되며 최대 24개의 로그가 저장됩니다.

다음은 기본 에이전트 로깅 동작을 변경하는 데 사용할 수 있는 컨테이너 에이전트 구성 변수입니다. 사용 가능한 모든 구성 파라미터에 대한 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 또는 GitHub의 Amazon ECS Agent README를 참조하세요.

다음은 컨테이너 에이전트 버전 1.36.0 이상에서 `logfmt` 형식을 사용할 때의 로그 파일 예시입니다.

```
level=info time=2019-12-12T23:43:29Z msg="Loading configuration" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-
ecs-agent:latest" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-
ecs-pause:0.1.0" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Amazon ECS agent Version: 1.36.0, Commit:
 ca640387" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Creating root ecs cgroup: /ecs"
 module=init_linux.go
level=info time=2019-12-12T23:43:29Z msg="Creating cgroup /ecs"
 module=cgroup_controller_linux.go
level=info time=2019-12-12T23:43:29Z msg="Loading state!" module=statemanager.go
level=info time=2019-12-12T23:43:29Z msg="Event stream ContainerChange start
 listening..." module=eventstream.go
level=info time=2019-12-12T23:43:29Z msg="Restored cluster 'auto-robc'" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Restored from checkpoint file. I
 am running as 'arn:aws:ecs:us-west-2:0123456789:container-instance/auto-
robc/3330a8a91d15464ea30662d5840164cd' in cluster 'auto-robc'" module=agent.go
```

다음은 JSON 형식을 사용할 때의 로그 파일 예시입니다.

```
{"time": "2019-11-07T22:52:02Z", "level": "info", "msg": "Starting Amazon Elastic
 Container Service Agent", "module": "engine.go"}
```

컨테이너 에이전트 버전 1.35.0 및 이전 버전의 경우 로그 파일의 형식은 다음과 같습니다.

```
2016-08-15T15:54:41Z [INFO] Starting Agent: Amazon ECS Agent - v1.12.0 (895f3c1)
2016-08-15T15:54:41Z [INFO] Loading configuration
2016-08-15T15:54:41Z [WARN] Invalid value for task cleanup duration, will be overridden
 to 3h0m0s, parsed value 0, minimum threshold 1m0s
2016-08-15T15:54:41Z [INFO] Checkpointing is enabled. Attempting to load state
2016-08-15T15:54:41Z [INFO] Loading state! module="statemanager"
2016-08-15T15:54:41Z [INFO] Detected Docker versions [1.17 1.18 1.19 1.20 1.21 1.22]
2016-08-15T15:54:41Z [INFO] Registering Instance with ECS
2016-08-15T15:54:41Z [INFO] Registered! module="api client"
```

프라이빗 Docker 이미지를 위한 Amazon ECS 컨테이너 인스턴스 구성

Amazon ECS 컨테이너 에이전트는 기본 인증을 사용하여 프라이빗 레지스트리로 인증할 수 있습니다. 프라이빗 레지스트리 인증을 활성화하면 태스크 정의에서 프라이빗 Docker 이미지를 사용할 수 있습니다. 이 기능은 EC2 시작 유형을 사용하는 태스크에서만 지원됩니다.

프라이빗 레지스트리 인증을 활성화하는 또 다른 방법은 AWS Secrets Manager를 사용하여 프라이빗 레지스트리 자격 증명을 안전하게 저장한 후 나중에 컨테이너 정의에서 참조합니다. 태스크에서 프라이빗 리포지토리의 이미지를 사용할 수 있는 것도 이러한 인증 기능 때문입니다. 이 방법은 EC2 또는 Fargate 시작 유형을 사용하는 작업만 지원합니다. 자세한 정보는 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용을 참조하세요.

Amazon ECS 컨테이너 에이전트는 시작할 때 다음 환경 변수 2개를 찾습니다.

- `ECS_ENGINE_AUTH_TYPE`, 전송되는 인증 데이터의 유형을 지정합니다.
- `ECS_ENGINE_AUTH_DATA`, 실제 인증 자격 증명이 포함됩니다.

Amazon ECS 최적화 AMI의 Linux 변형은 컨테이너 인스턴스가 시작될 때, 그리고 서비스가 시작될 때마다이 변수에 대해 `/etc/ecs/ecs.config` 파일을 스캔할 수 있습니다(sudo start ecs 명령 사용). Amazon ECS 최적화가 아닌 AMI는 이러한 환경 변수를 파일에 저장하고 `--env-file` *`path_to_env_file`* 옵션을 사용하여 컨테이너 에이전트를 시작하는 docker run 명령에 전달해야 합니다.

> **Important**
>
> 이러한 인증 환경 변수를 인스턴스 시작 시 Amazon EC2 사용자 데이터를 사용하여 첨가하거나 `--env` 옵션을 사용하여 docker run 명령으로 전달하지 않는 것이 좋습니다. 인증 자격 증명과 같은 민감한 정보인 경우 이러한 방법이 적절하지 않습니다. 인증 자격 증명을 컨테이너 인

스턴스에 안전하게 추가하는 방법에 대한 자세한 내용은 Amazon S3에 Amazon ECS 컨테이너 인스턴스 구성 저장 섹션을 참조하세요.

인증 형식

프라이빗 레지스트리 인증에 dockercfg 및 docker 두 가지 형식을 사용할 수 있습니다.

Dockercfg 인증 형식

dockercfg 형식은 docker login 명령을 실행하면 만들어지는 구성 파일에 저장된 인증 정보를 사용합니다. 이 파일은 로컬 시스템에서 docker login을 실행하고 레지스트리 사용자 이름, 암호 및 이메일 주소를 입력하여 만들 수 있습니다. 컨테이너 인스턴스에 로그인한 후 그 인스턴스에서 명령을 실행할 수도 있습니다. Docker 버전에 따라 이 파일은 ~/.dockercfg 또는 ~/.docker/config.json으로 저장됩니다.

```
cat ~/.docker/config.json
```

출력:

```
{
  "auths": {
    "https://index.docker.io/v1/": {
      "auth": "zq212MzEXAMPLE7o6T25Dk0i"
    }
  }
}
```

Important

Docker 버전이 새로우면 바깥쪽 auths 객체를 사용하여 위와 같은 구성 파일이 생성됩니다. Amazon ECS 에이전트는 auths 객체 없이 아래 형식의 dockercfg 인증 데이터만 지원합니다. jq 유틸리티가 설치된 경우에는 cat ~/.docker/config.json | jq .auths 명령으로 이 데이터를 추출할 수 있습니다.

```
cat ~/.docker/config.json | jq .auths
```

출력:

```
{
  "https://index.docker.io/v1/": {
    "auth": "zq212MzEXAMPLE7o6T25Dk0i",
    "email": "email@example.com"
  }
}
```

위 예제에서는 실행 시간에 Amazon ECS 컨테이너 에이전트가 로드하는 환경 변수 파일(Amazon ECS 최적화 AMI의 경우에는 `/etc/ecs/ecs.config`)에 다음 환경 변수를 추가해야 합니다. Amazon ECS 최적화 AMI를 사용하지 않고 docker run을 사용하여 수동으로 에이전트를 시작하는 경우 에이전트를 시작할 때 `--env-file` *path_to_env_file* 옵션을 통해 환경 변수 파일을 지정합니다.

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

다음 구문을 사용하면 여러 프라이빗 레지스트리를 구성할 수 있습니다.

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example-01.com"},"repo.example-02.com":
{"auth":"fQ172MzEXAMPLEoF7225DU0j","email":"email@example-02.com"}}
```

Docker 인증 형식

`docker` 형식은 에이전트가 인증할 때 사용하는 레지스트리 서버의 JSON 표현을 사용합니다. 여기에는 해당 레지스트리가 요구하는 인증 파라미터도 포함됩니다(예: 해당 계정의 사용자 이름, 암호, 이메일 주소). Docker Hub 계정의 경우 JSON 표현은 다음과 같습니다.

```
{
  "https://index.docker.io/v1/": {
    "username": "my_name",
    "password": "my_password",
    "email": "email@example.com"
  }
}
```

이 예제에서는 실행 시간에 Amazon ECS 컨테이너 에이전트가 로드하는 환경 변수 파일(Amazon ECS 최적화 AMI의 경우에는 `/etc/ecs/ecs.config`)에 다음 환경 변수를 추가해야 합니다.

Amazon ECS 최적화 AMI를 사용하지 않고 docker run을 사용하여 수동으로 에이전트를 시작하는 경우 에이전트를 시작할 때 `--env-file` *path_to_env_file* 옵션을 통해 환경 변수 파일을 지정합니다.

```
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username":"my_name","password":"my_password","email":"email@example.com"}}
```

다음 구문을 사용하면 여러 프라이빗 레지스트리를 구성할 수 있습니다.

```
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":
{"username":"my_name","password":"my_password","email":"email@example-01.com"},"repo.example-02
{"username":"another_name","password":"another_password","email":"email@example-02.com"}}
```

절차

컨테이너 인스턴스에 대한 프라이빗 레지스트리를 사용 설정하려면 다음 절차를 따르세요.

Amazon ECS 최적화 AMI에서 프라이빗 레지스트리를 활성화하는 방법

1. SSH를 사용해 컨테이너 인스턴스에 로그인합니다.
2. `/etc/ecs/ecs.config` 파일을 열고 레지스트리 및 계정으로 다음과 같이 `ECS_ENGINE_AUTH_TYPE` 및 `ECS_ENGINE_AUTH_DATA` 값을 추가합니다.

   ```
   sudo vi /etc/ecs/ecs.config
   ```

   이 예제에서는 Docker Hub 사용자 계정을 인증합니다.

   ```
   ECS_ENGINE_AUTH_TYPE=docker
   ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
   {"username":"my_name","password":"my_password","email":"email@example.com"}}
   ```

3. 에이전트가 `ECS_DATADIR` 환경 변수를 사용하여 해당 상태를 다음과 같이 저장하는지 확인합니다.

   ```
   docker inspect ecs-agent | grep ECS_DATADIR
   ```

   출력:

```
"ECS_DATADIR=/data",
```

> **Important**
>
> 이전 명령이 ECS_DATADIR 환경 변수를 반환하지 않으면 에이전트를 중지하기 전에 이 컨테이너 인스턴스에서 실행 중인 태스크를 모두 중지해야 합니다. ECS_DATADIR 환경 변수를 사용하는 새 버전의 에이전트는 해당 상태를 저장하며 태스크가 실행되는 동안 에이전트를 중지하고 시작해도 문제가 발생하지 않습니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트을 참조하세요.

4. 다음과 같이 ecs 서비스를 중단합니다.

```
sudo stop ecs
```

출력:

```
ecs stop/waiting
```

5. ecs 서비스를 다시 시작합니다.

   - Amazon ECS 최적화 Amazon Linux 2 AMI의 경우:

     ```
     sudo systemctl restart ecs
     ```

   - Amazon ECS 최적화 Amazon Linux AMI의 경우:

     ```
     sudo stop ecs && sudo start ecs
     ```

6. (선택 사항) 에이전트 내부 검사 API 태스크를 쿼리하여 에이전트가 실행 중인지 확인하고 새 컨테이너 인스턴스에 대한 일부 정보를 볼 수 있습니다. 자세한 내용은 the section called “컨테이너 내부 검사” 섹션을 참조하세요.

```
curl http://localhost:51678/v1/metadata
```

자동 Amazon ECS 태스크 및 이미지 정리

컨테이너 인스턴스에 작업이 배치될 때마다 Amazon ECS 컨테이너 에이전트가 작업에서 참조된 이미지가 리포지토리에서 지정된 태그의 최신 이미지인지 확인합니다. 최신이 아닌 경우 기본 동작은 에이전트가 해당 리포지토리에서 이미지를 가져오도록 허용합니다. 태스크와 서비스에서 이미지를 자주 업데이트하는 경우 컨테이너 인스턴스 스토리지가 더 이상 사용하지 않고 앞으로도 다시 사용하지 않을 Docker 이미지로 금세 가득 찰 수 있습니다. 예를 들어 지속적 통합 및 지속적 배포(CI/CD) 파이프라인을 사용할 수 있습니다.

> **Note**
>
> Amazon ECS 에이전트 이미지 가져오기 동작은 `ECS_IMAGE_PULL_BEHAVIOR` 파라미터를 사용하여 사용자 지정할 수 있습니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성을 참조하세요.

마찬가지로, 중지된 태스크에 속하는 컨테이너가 로그 정보, 데이터 볼륨 및 기타 아티팩트로 컨테이너 인스턴스 스토리지를 소비할 수도 있습니다. 이러한 아티팩트는 예기치 않게 중지한 컨테이너를 디버깅하는 데 유용하지만, 이 스토리지는 대부분 일정 시간 후에 안전하게 해제될 수 있습니다.

기본적으로 Amazon ECS 컨테이너 에이전트는 컨테이너 인스턴스에서 어떤 작업도 사용하고 있지 않은 중지된 작업 및 Docker 이미지를 자동으로 정리합니다.

> **Note**
>
> 자동 이미지 정리 기능을 사용하려면 Amazon ECS 컨테이너 에이전트 버전 1.13.0 이상이 필요합니다. 에이전트를 최신 버전으로 업데이트하려면 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

다음의 에이전트 구성 변수를 사용하여 자동 태스크 및 이미지 정리 환경을 조정할 수 있습니다. 컨테이너 인스턴스에서 이들 변수를 설정하는 방법에 대한 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

`ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION`

중지된 태스크의 컨테이너를 삭제하기 위해 대기하는 기본 시간. 값이 1보다 작게 설정된 경우 값은 무시됩니다. 기본적으로 이 파라미터는 3시간으로 설정되지만 애플리케이션에 필요할 경우 이 기간을 최소 1초로 줄일 수 있습니다.

> **Note**
>
> 이미지 정리 프로세스는 컨테이너가 이미지를 참조하는 한 해당 이미지를 삭제할 수 없습니다. 컨테이너가 제거되면 참조되지 않은 모든 이미지는 이미지 정리 구성 파라미터를 기반으로 정리 대상이 됩니다.

`ECS_DISABLE_IMAGE_CLEANUP`

이 변수를 `true`로 설정할 경우 컨테이너 인스턴스에서 자동 이미지 정리가 꺼지고 이미지가 자동으로 제거되지 않습니다.

`ECS_IMAGE_CLEANUP_INTERVAL`

이 변수는 자동 이미지 정리 프로세스가 삭제할 이미지가 있는지 확인하는 주기를 지정합니다. 기본값은 매 30분이지만 애플리케이션에 필요할 경우 사용자가 이 기간을 최소 10분으로 단축하여 이미지를 제거할 수 있습니다.

`ECS_IMAGE_MINIMUM_CLEANUP_AGE`

이 변수는 이미지를 끌어온 시점과 이미지가 제거 후보가 될 수 있는 시점 사이의 최소 시간 간격을 지정합니다. 이 변수는 방금 끌어온 이미지가 정리되는 것을 방지하기 위해 사용합니다. 기본값은 1시간입니다.

`ECS_NUM_IMAGES_DELETE_PER_CYCLE`

이 변수는 단일 정리 사이클에서 제거할 수 있는 이미지 수를 지정합니다. 기본값은 5이고, 최솟값은 1입니다.

Amazon ECS 컨테이너 에이전트가 실행 중이고 자동 이미지 정리가 꺼지지 않았으면 에이전트가 실행 중 또는 중지된 컨테이너에 의해 참조되지 않는 Docker 이미지가 있는지 `ECS_IMAGE_CLEANUP_INTERVAL` 변수로 지정된 주기로 확인합니다. 사용되지 않는 이미지가 발견되고 해당 이미지가 `ECS_IMAGE_MINIMUM_CLEANUP_AGE` 변수로 지정된 최소 정리 기간보다 오래된 경우 에이전트가 `ECS_NUM_IMAGES_DELETE_PER_CYCLE` 변수로 지정된 최대 이미지 수만큼 제거합니다. 가장 이전에 참조된 이미지가 먼저 삭제됩니다. 이미지가 제거된 후, 에이전트는 다음 주기까지 대기했다 프로세스를 반복합니다.

Amazon ECS에서 컨테이너 예약

Amazon Elastic Container Service(Amazon ECS)는 컨테이너화된 워크로드드에 대한 유연한 예약 기능을 제공하는 공유 상태의 낙관적 동시성 시스템입니다. Amazon ECS 스케줄러는 Amazon ECS API와 동일한 클러스터 상태 정보를 활용하여 적절한 배치 결정을 내립니다.

Amazon ECS는 장기 실행 태스크 및 애플리케이션을 위한 서비스 스케줄러를 제공합니다. 또한 배치 작업 또는 단일 실행 작업에 대해 독립 실행형 작업 또는 예약된 작업을 실행할 수 있는 기능을 제공합니다. 요구 사항에 가장 적합한 태스크를 실행하기 위해 작업 배치 전략과 제약 조건을 지정할 수 있습니다. 예를 들어 작업이 여러 가용 영역에서 실행되는지 또는 단일 가용 영역 내에서 실행되는지를 지정할 수 있습니다. 또한 필요에 따라 사용자 정의 또는 서드 파티 스케줄러와 태스크를 통합할 수 있습니다.

옵션	사용해야 하는 경우	추가 정보
Service	서비스 스케줄러는 장기 실행 상태 비저장 서비스 및 애플리케이션에 적합합니다. 또 서비스 스케줄러는 필요하다면 Elastic Load Balancing 로드 밸런서에 대해 작업이 등록되도록 합니다. 서비스 스케줄러에서 유지되는 서비스를 업데이트할 수 있습니다. 여기에는 새 태스크 정의를 배포하거나 실행 중인 원하는 작업 수를 변경하는 작업이 포함될 수 있습니다. 기본적으로 서비스 스케줄러는 여러 가용 영역에 태스크를 분산합니다. 그러나 작업 배치 전략과 제약을 사용하여 작업 배치 결정을 사용자 지정할 수 있습니다.	Amazon ECS 서비스
독립 실행형 작업	독립 실행형 태스크는 작업을 수행한 다음 중지하는 배치 작업과 같은 프로세스에 적합합	Amazon ECS 독립 실행형 작업

옵션	사용해야 하는 경우	추가 정보
	니다. 예를 들어 작업이 대기열에 들어올 때 프로세스가 RunTask를 호출하도록 할 수 있습니다. 이 태스크는 대기열에서 태스크를 가져와 해당 태스크를 수행한 다음 종료됩니다. RunTask를 사용하면 기본 작업 배치 전략이 태스크를 클러스터에 무작위로 분산하도록 할 수 있습니다. 이렇게 하면 단일 인스턴스에 불균형한 작업 수가 주어질 가능성이 최소화됩니다.	
예약된 작업	예약된 태스크는 클러스터에서 설정된 간격으로 실행할 태스크가 있는 경우에 적합하며 EventBridge 스케줄러를 사용하여 일정을 생성할 수 있습니다. 백업 태스크 또는 로그 스캔에 대한 태스크를 실행할 수 있습니다. 생성한 EventBridge 스케줄러 일정에 따라 지정된 시간에 클러스터에서 하나 이상의 작업을 실행할 수 있습니다. 예약된 이벤트는 특정 간격으로 설정할 수 있습니다(*N*분, 시간 또는 일마다 실행). 그렇지 않으면 더 복잡한 일정 지정을 위해 cron 표현식을 사용할 수 있습니다.	Amazon EventBridge Scheduler를 사용하여 Amazon ECS 태스크 예약

컴퓨팅 옵션

Amazon ECS를 사용하면 태스크 또는 서비스가 실행되는 인프라를 지정할 수 있습니다. 용량 공급자 전략 또는 시작 유형을 사용할 수 있습니다.

Fargate의 경우 용량 공급자는 Fargate와 Fargate Spot입니다. EC2의 경우 용량 공급자는 등록된 컨테이너 인스턴스가 있는 Auto Scaling 그룹입니다.

용량 공급자 전략은 클러스터와 연결된 용량 제공자 전체에 태스크를 분산합니다.

클러스터와 이미 연결되어 있고 `ACTIVE` 또는 `UPDATING` 상태인 용량 공급자만 용량 공급자 전략에 사용할 수 있습니다. 클러스터를 생성할 때 용량 공급자를 클러스터와 연결할 수 있습니다.

용량 공급자 전략에서 선택적 base 값은 지정된 용량 공급자에서 실행되는 최소 작업 수를 지정합니다. 용량 공급자 전략에서 하나의 용량 공급자만 기준을 정의할 수 있습니다.

weight 값은 지정된 용량 공급자를 사용하는 시작된 총 작업 수의 상대 백분율을 결정합니다. 다음 예제를 살펴보세요. 두 개의 용량 공급자가 있고, 가중치가 모두 1인 전략이 있습니다. 기본 백분율에 도달하면 작업은 두 용량 공급자에서 균등하게 분할됩니다. 동일한 로직을 사용하여 capacityProviderA의 가중치를 1로 지정하고, capacityProviderB의 가중치를 4로 지정한다고 가정합니다. 그런 다음 capacityProviderA를 사용하여 실행되는 각 작업에 대해 capacityProviderB를 사용하는 4개의 작업이 있습니다.

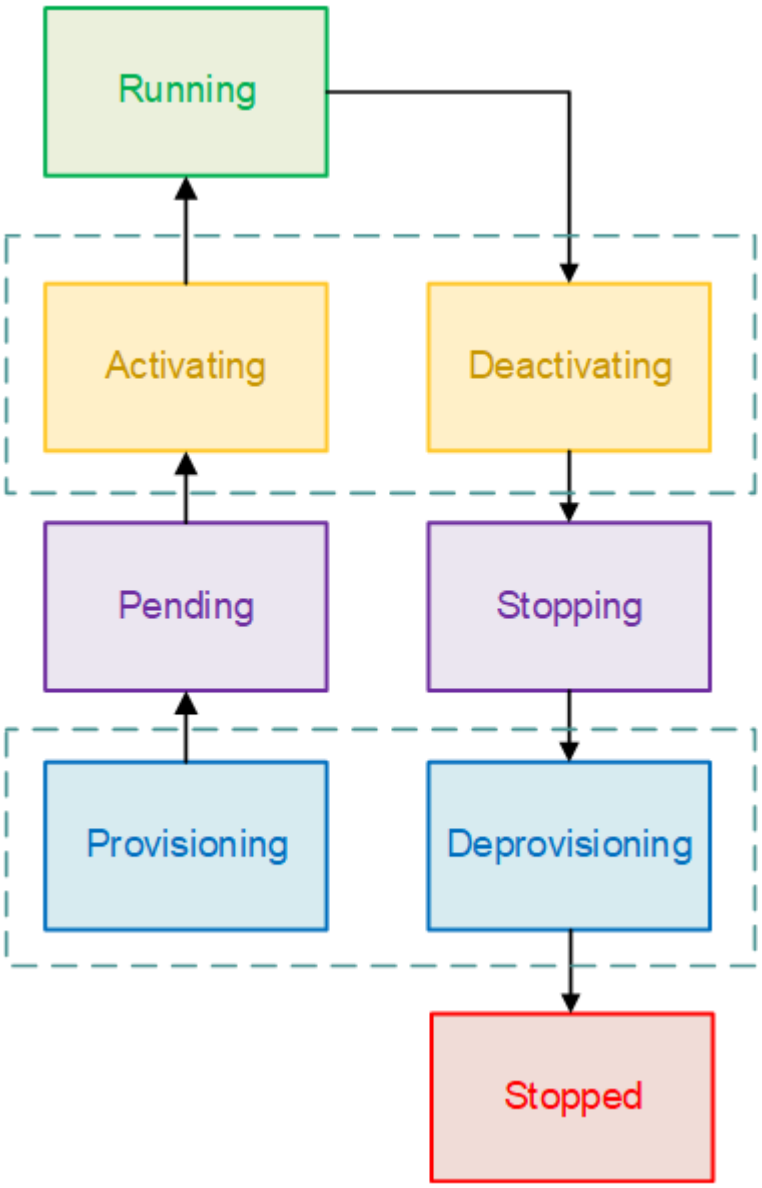
시작 유형은 Fargate 또는 클러스터에 수동으로 등록한 Amazon EC2 인스턴스에서 직접 태스크를 시작합니다.

Amazon ECS 작업 수명 주기

수동으로 또는 서비스의 일부로 시작되는 태스크는 스스로 또는 수동으로 중지될 때까지 몇 가지 상태를 거칠 수 있습니다. 일부 태스크는 `PENDING`에서 `RUNNING`을 거쳐 `STOPPED`까지 자연적으로 진행되는 배치 작업으로 실행되도록 되어 있습니다. 서비스의 일부일 수 있는 또 다른 태스크는 무한정 계속 실행되거나 필요에 따라 확장 및 축소되도록 되어 있습니다.

작업 중지 또는 원하는 서비스 수의 확장 또는 축소 업데이트 같은 작업 상태 변경이 요청되면 Amazon ECS 컨테이너 에이전트는 마지막으로 알려진 작업 상태(`lastStatus`)와 원하는 작업 상태(`desiredStatus`)로 이러한 변경을 추적합니다. 마지막으로 알려진 작업 상태와 원하는 작업 상태는 콘솔에서 보거나 API 또는 AWS CLI로 태스크를 설명하여 볼 수 있습니다.

아래의 순서도는 작업 수명 주기 흐름을 보여줍니다.



수명 주기 상태

작업 수명 주기 상태를 각각 설명하면 다음과 같습니다.

PROVISIONING

Amazon ECS는 작업이 시작되기 전에 추가 단계를 수행해야 합니다. 예를 들어, awsvpc 네트워크 모드를 사용하는 작업의 경우 탄력적 네트워크 인터페이스를 프로비저닝해야 합니다.

보류 중(PENDING)

Amazon ECS가 추가 태스크를 수행하기 위해 컨테이너 에이전트에서 대기하는 전환 상태입니다. 태스크에 사용할 수 있는 리소스가 있을 때까지 태스크는 보류 중 상태로 유지됩니다.

활성화 중(ACTIVATING)

이것은 Amazon ECS에서 작업이 시작된 후 RUNNING 상태로 전환되기 전에 추가 단계를 수행해야 하는 전환 상태입니다. Amazon ECS가 컨테이너 이미지를 가져오고, 컨테이너를 생성하고, 태스크 네트워킹을 구성하고, 로드 밸런서 대상 그룹을 등록하고, 서비스 검색을 구성하는 상태입니다.

실행 중(RUNNING)

작업이 실행 중입니다.

비활성화 중(DEACTIVATING)

이것은 Amazon ECS에서 작업이 중지되기 전에 추가 단계를 수행해야 하는 전환 상태입니다. 예를 들면, Elastic Load Balancing 대상 그룹을 사용하도록 구성된 서비스의 일부인 태스크의 경우 이 상태일 때 대상 그룹 등록 취소가 발생합니다.

중지 중(STOPPING)

Amazon ECS가 추가 태스크를 수행하기 위해 컨테이너 에이전트에서 대기하는 전환 상태입니다.

Linux 컨테이너의 경우 컨테이너 에이전트는 SIGTERM 신호를 보내 애플리케이션을 완료하고 종료해야 함을 알리고 작업 정의에 설정된 StopTimeout 지속 시간을 기다린 후 SIGKILL을 전송합니다.

프로비저닝 취소 중(DEPROVISIONING)

Amazon ECS는 작업이 중지된 후 작업이 STOPPED 상태로 전환되기 전에 추가 단계를 수행해야 합니다. 예를 들어, awsvpc 네트워크 모드를 사용하는 작업의 경우 탄력적 네트워크 인터페이스를 분리 및 삭제해야 합니다.

중지됨(STOPPED)

작업이 중지되었습니다.

오류로 인해 작업이 중지된 경우 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

DELETED

태스크가 중지될 때의 전환 상태입니다. 이 상태는 콘솔에는 표시되지 않지만 describe-tasks에는 표시됩니다.

Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법

작업 배치를 사용하여 특정 기준(예: 가용 영역 또는 인스턴스 유형)을 충족하는 컨테이너 인스턴스에 작업을 배치하도록 Amazon ECS를 구성할 수 있습니다.

다음은 작업 배치 구성 요소입니다.

- 작업 배치 전략 - 작업을 배치할 컨테이너 인스턴스 또는 종료할 작업을 선택하는 알고리즘입니다. 예를 들어 Amazon ECS는 무작위로 컨테이너 인스턴스를 선택하거나 인스턴스 그룹에 작업이 균등하게 배분되도록 컨테이너 인스턴스를 선택할 수 있습니다.
- 작업 그룹 - 관련 작업 그룹(예: 데이터베이스 작업)입니다.
- 작업 배치 제약 조건 - 컨테이너 인스턴스에 작업을 배치하기 위해 충족해야 하는 규칙입니다. 제약 조건이 충족되지 않으면 작업이 배치되지 않고 `PENDING` 상태로 유지됩니다. 예를 들어 제약 조건을 사용하여 특정 인스턴스 유형에만 작업을 배치할 수 있습니다.

Amazon ECS는 시작 유형에 따라 여러 알고리즘을 사용합니다.

EC2 시작 유형

EC2 시작 유형을 사용하는 작업의 경우 Amazon ECS는 CPU와 메모리 등 작업 정의에 지정된 요구 사항에 따라 작업 배치 위치를 결정해야 합니다. 마찬가지로 작업 수를 축소하는 경우, Amazon ECS는 어떤 태스크를 종료할지 결정해야 합니다. 작업 배치 전략과 제약을 적용하여 Amazon ECS가 태스크를 배치하고 종료하는 방법을 사용자 지정할 수 있습니다.

기본 작업 배치 전략은 작업을 수동으로 실행하는지 아니면 서비스 내에서 실행하는지에 따라 달라집니다. Amazon ECS 서비스의 일부로 실행되는 작업의 경우 기본 작업 배치 전략은 `attribute:ecs.availability-zone`을 사용하는 `spread`입니다. 서비스 중이 아닌 태스크에는 기본 태스크 배치 제약이 없습니다. 자세한 내용은 Amazon ECS에서 컨테이너 예약 단원을 참조하십시오.

> **Note**
>
> 작업 배치 전략은 최선의 노력입니다. Amazon ECS는 가장 최적의 배치 옵션을 사용할 수 없을 경우에도 작업 배치를 계속 시도합니다. 하지만 작업 배치 제약이 적용되므로 작업 배치를 막을 수 있습니다.

작업 배치 전략과 제약을 함께 사용할 수 있습니다. 예를 들어 작업 배치 전략과 작업 배치 제약을 사용하여 가용 영역과 각 가용 영역 내의 메모리를 기반으로 하는 휴지통 팩 작업에 태스크를 분산할 수 있으나, 이는 G2 인스턴스에만 해당됩니다.

Amazon ECS는 태스크를 배치할 때 다음 프로세스를 사용하여 컨테이너 인스턴스를 선택합니다.

1. 작업 정의의 CPU, GPU, 메모리, 포트 요구 사항을 충족하는 컨테이너 인스턴스를 식별합니다.

2. 작업 배치 제약 조건을 충족하는 컨테이너 인스턴스를 식별합니다.
3. 작업 배치 전략을 충족하는 컨테이너 인스턴스를 식별합니다.
4. 작업 배치를 위해 컨테이너 인스턴스를 선택합니다.

Fargate 시작 유형

Fargate 시작 유형을 사용하는 태스크에 대해서는 작업 배치 전략 및 제약이 지원되지 않습니다. Fargate는 액세스 가능한 가용 영역 전체에 작업을 분산시키기 위해 최선을 다합니다. 용량 공급자에 Fargate와 Fargate 스팟이 모두 포함되는 경우 분산 동작은 용량 공급자마다 독립적입니다.

전략을 사용하여 Amazon ECS 작업 배치 정의

EC2 시작 유형을 사용하는 작업의 경우 Amazon ECS는 CPU와 메모리 등 작업 정의에 지정된 요구 사항에 따라 작업 배치 위치를 결정해야 합니다. 마찬가지로 작업 수를 축소하는 경우, Amazon ECS는 어떤 태스크를 종료할지 결정해야 합니다. 작업 배치 전략과 제약을 적용하여 Amazon ECS가 태스크를 배치하고 종료하는 방법을 사용자 지정할 수 있습니다.

기본 작업 배치 전략은 작업을 수동으로 실행하는지 아니면 서비스 내에서 실행하는지에 따라 달라집니다. Amazon ECS 서비스의 일부로 실행되는 작업의 경우 기본 작업 배치 전략은 `attribute:ecs.availability-zone`을 사용하는 `spread`입니다. 서비스 중이 아닌 태스크에는 기본 태스크 배치 제약이 없습니다. 자세한 내용은 Amazon ECS에서 컨테이너 예약 단원을 참조하십시오.

> **Note**
>
> 작업 배치 전략은 최선의 노력입니다. Amazon ECS는 가장 최적의 배치 옵션을 사용할 수 없을 경우에도 작업 배치를 계속 시도합니다. 하지만 작업 배치 제약이 적용되므로 작업 배치를 막을 수 있습니다.

작업 배치 전략과 제약을 함께 사용할 수 있습니다. 예를 들어 작업 배치 전략과 작업 배치 제약을 사용하여 가용 영역과 각 가용 영역 내의 메모리를 기반으로 하는 휴지통 팩 작업에 태스크를 분산할 수 있으나, 이는 G2 인스턴스에만 해당됩니다.

Amazon ECS는 태스크를 배치할 때 다음 프로세스를 사용하여 컨테이너 인스턴스를 선택합니다.

1. 작업 정의의 CPU, GPU, 메모리, 포트 요구 사항을 충족하는 컨테이너 인스턴스를 식별합니다.
2. 작업 배치 제약 조건을 충족하는 컨테이너 인스턴스를 식별합니다.

3. 작업 배치 전략을 충족하는 컨테이너 인스턴스를 식별합니다.
4. 작업 배치를 위해 컨테이너 인스턴스를 선택합니다.

서비스 정의에서 작업 배치 전략을 지정하거나 `placementStrategy` 파라미터를 사용하여 작업 정의에서 지정합니다.

```
"placementStrategy": [
    {
        "field": "The field to apply the placement strategy against",
        "type": "The placement strategy to use"
    }
]
```

작업을 실행할 때(RunTask), 새 서비스를 생성할 때(CreateService), 또는 기존 서비스를 업데이트할 때(UpdateService) 전략을 지정할 수 있습니다.

다음 테이블에서는 사용 가능한 유형 및 필드를 설명합니다.

type	유효한 필드 값	
`binpack` 태스크는 사용되지 않는 CPU 또는 메모리를 최소한으로 남겨둘 수 있도록 컨테이너 인스턴스에 배치됩니다. 이 전략은 사용 중인 컨테이너 인스턴스의 수를 최소화합니다. 이 전략을 사용하고 축소 작업을 수행하면 Amazon ECS가 태스크를 종료합니다. 이는 태스크가 종료된 후 컨테이너 인스턴스에 남아 있는 리소스의 양을 기준으로 합니다. 태스크 종료 후 사용 가능한 리소스가 가장 많이 남은 컨테이너 인스	• cpu • 메모리	

type	유효한 필드 값	
턴스가 해당 태스크를 종료합니다.		
`random` 태스크가 무작위로 배치됩니다.	사용되지 않습니다	
`spread` 태스크가 지정된 값에 따라 균등하게 배치됩니다. 서비스 태스크는 해당 서비스의 태스크를 기준으로 분산됩니다. 독립 실행형 태스크는 동일한 작업 그룹의 태스크를 기준으로 분산됩니다. 작업 그룹에 대한 자세한 정보는 그룹 관련 Amazon ECS 작업 섹션을 참조하세요. `spread` 전략이 사용되고 축소 작업이 수행되면 Amazon ECS가 가용 영역 간에 균형을 유지하도록 종료할 태스크를 선택합니다. 가용 영역 내에서 태스크가 무작위로 선택됩니다.	• `instanceId` (또는 `host`, 효과가 동일함) • 컨테이너 인스턴스에 적용되는 모든 플랫폼 또는 사용자 지정 속성(예: `attribute:ecs.availability-zone`)	

기존 서비스에 대해서도 작업 배치 전략을 업데이트할 수 있습니다. 자세한 내용은 Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법 섹션을 참조하세요.

전략을 수행할 순서대로 전략 배열을 생성하여 여러 전략을 사용하는 작업 배치 전략을 생성할 수 있습니다. 예를 들어 가용 영역에 작업을 분산한 다음 각 가용 영역 내의 메모리를 기반으로 하는 작업을 빈 팩하려면 가용 영역 전략과 메모리 전략을 차례로 지정합니다. 예제 전략은 Amazon ECS 작업 배치 전략 예제를 참조하세요.

Amazon ECS 작업 배치 전략 예제

CreateService, UpdateService 및 RunTask 태스크를 사용하면 작업 배치 전략을 지정할 수 있습니다.

예시

- 가용 영역에 균등하게 작업 분산
- 모든 인스턴스에 균등하게 작업 분산
- 메모리를 기반으로 작업 빈팩
- 무작위로 작업 배치
- 가용 영역에 균등하게 작업을 분산한 다음 각 가용 영역 내에서 인스턴스에 균등하게 작업 분산
- 가용 영역에 균등하게 작업을 분산한 다음 각 가용 영역 내의 메모리를 기준으로 작업 빈팩
- 인스턴스에 균등하게 작업을 분산한 다음 메모리를 기반으로 작업 빈팩

가용 영역에 균등하게 작업 분산

다음 전략은 가용 영역에 균등하게 태스크를 분산합니다.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    }
]
```

모든 인스턴스에 균등하게 작업 분산

다음 전략은 모든 인스턴스에 균등하게 태스크를 분산합니다.

```
"placementStrategy": [
    {
        "field": "instanceId",
        "type": "spread"
    }
]
```

메모리를 기반으로 작업 빈팩

다음 전략은 메모리를 기준으로 태스크를 bin-pack합니다.

```
"placementStrategy": [
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

무작위로 작업 배치

다음 전략은 태스크를 무작위로 배치합니다.

```
"placementStrategy": [
    {
        "type": "random"
    }
]
```

가용 영역에 균등하게 작업을 분산한 다음 각 가용 영역 내에서 인스턴스에 균등하게 작업 분산

다음 전략은 가용 영역에 균등하게 태스크를 분산한 다음 각 가용 영역 내에서 인스턴스에 균등하게 태스크를 분산합니다.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "instanceId",
        "type": "spread"
    }
]
```

가용 영역에 균등하게 작업을 분산한 다음 각 가용 영역 내의 메모리를 기준으로 작업 빈팩

다음 전략은 가용 영역에 균등하게 태스크를 분산한 다음 각 가용 영역 내에서 메모리를 기준으로 태스크를 bin-pack합니다.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
```

```
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

인스턴스에 균등하게 작업을 분산한 다음 메모리를 기반으로 작업 빈팩

다음 전략에서는 모든 인스턴스에 균등하게 작업을 분산한 다음 각 인스턴스 내의 메모리를 기반으로 작업을 빈팩합니다.

```
"placementStrategy": [
    {
        "field": "instanceId",
        "type": "spread"
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

그룹 관련 Amazon ECS 작업

관련된 작업 세트를 식별하고 작업 그룹에 배치할 수 있습니다. 태스크 그룹 이름이 같은 모든 태스크는 `spread` 작업 배치 전략을 사용할 때 하나의 집합으로 간주됩니다. 예를 들어 데이터베이스와 웹 서버 등 하나의 클러스터에서 상이한 애플리케이션을 실행한다고 가정해 봅시다. 가용 영역에서 데이터베이스들이 균형을 이루도록 하려면 `databases`라는 이름의 태스크 그룹에 추가한 다음 `spread` 작업 배치 전략을 사용합니다. 자세한 정보는 전략을 사용하여 Amazon ECS 작업 배치 정의을 참조하세요.

태스크 그룹을 작업 배치 제약 조건으로 사용할 수도 있습니다. `memberOf` 제약 조건에 작업 그룹을 지정하면 작업은 지정된 작업 그룹에 실행되는 컨테이너 인스턴스로만 전송됩니다. 예시는 Amazon ECS 작업 배치 제약 조건 예제 섹션을 참조하세요.

기본적으로 독립 실행형 태스크에서는 사용자 정의 태스크 그룹 이름이 지정되지 않은 경우 태스크 정의 패밀리 이름을 태스크 그룹 이름으로 사용합니다(예: `family:my-task-definition`). 서비스의 일부로 시작된 태스크는 서비스 이름을 작업 그룹 이름으로 사용하며 변경할 수 없습니다.

작업 그룹에 대해 다음 요구 사항이 적용됩니다.

- 태스크 그룹 이름은 255자 이하여야 합니다.
- 각 태스크는 정확히 하나의 그룹에 속할 수 있습니다.
- 태스크를 시작한 후에는 해당 태스크 그룹을 수정할 수 없습니다.

Amazon ECS가 작업에 사용하는 컨테이너 인스턴스 정의

작업 배치 제약 조건은 Amazon ECS가 해당 인스턴스에서 작업을 실행할 수 있는지 여부를 결정하는 데 사용하는 컨테이너 인스턴스에 대한 규칙입니다. 하나 이상의 컨테이너 인스턴스가 제약과 일치해야 합니다. 제약과 일치하는 인스턴스가 없는 경우 작업은 PENDING 상태로 유지됩니다. 새 서비스를 생성하거나 기존 서비스를 업데이트할 때 서비스 작업에 대한 작업 배치 제약을 지정할 수 있습니다.

placementConstraint 파라미터를 사용하여 서비스 정의, 작업 정의 또는 작업에서 작업 배치 제약 조건을 지정할 수 있습니다.

```
"placementConstraints": [
    {
        "expression": "The expression that defines the task placement constraints",
        "type": "The placement constraint type to use"
    }
]
```

다음 테이블에서는 파라미터를 사용하는 방법을 설명합니다.

제약 유형	지정 가능한 경우	
distinctInstance 각 활성 태스크를 서로 다른 컨테이너 인스턴스에 배치합니다. Amazon ECS는 작업 배치를 위해 원하는 작업 상태를 확인합니다. 예를 들어 기존 작업의 원하는 상태가 STOPPED인 경우(하지만 마지막 상태는 그렇지 않음), distinctInstance 배치 제약에도 불구	• 작업 실행 RunTask • 새 서비스 생성 CreateService	

제약 유형	지정 가능한 경우	
하고 새로 들어오는 작업을 동일한 인스턴스에 배치할 수 있습니다. 따라서 동일한 인스턴스에 마지막 RUNNING 상태인 작업 2개가 표시될 수 있습니다. **Important** 엄격한 작업 격리를 원하는 고객은 Fargate를 사용하는 것이 좋습니다. Fargate는 하드웨어 가상화 환경에서 각 작업을 실행합니다. 이러한 컨테이너화된 워크로드는 네트워크 인터페이스, Fargate 임시 스토리지, CPU 또는 메모리를 다른 작업과 공유하지 않습니다. 자세한 내용은 Security Overview of AWS Fargate를 참조하세요.		

제약 유형	지정 가능한 경우	
`memberOf` 표현식을 충족하는 컨테이너 인스턴스에 태스크를 배치.	• 작업 실행 RunTask • 새 서비스 생성 CreateService • 새 작업 정의 생성 RegisterTaskDefinition • 작업 정의의 새 개정 생성 RegisterTaskDefinition • 서비스 업데이트 UpdateService	

`memberOf` 제약 조건 유형을 사용하는 경우 Amazon ECS가 작업을 배치할 수 있는 컨테이너 인스턴스를 정의하는 클러스터 쿼리 언어를 사용하여 표현식을 생성할 수 있습니다. 표현식은 속성별로 컨테이너 인스턴스를 그룹화하는 방법입니다. 표현식은 `placementConstraint`의 `expression` 파라미터에 포함됩니다.

Amazon ECS 컨테이너 인스턴스 속성

속성이라고 하는 사용자 지정 메타데이터를 컨테이너 인스턴스에 추가할 수 있습니다. 각 속성에는 이름과 선택 사항인 문자열 값이 있습니다. Amazon ECS가 제공하는 기본 속성을 사용하거나 사용자 지정 속성을 정의할 수 있습니다.

다음 섹션에는 샘플 기본 제공, 선택적 및 사용자 정의 속성이 있습니다.

기본 속성

Amazon ECS는 자동으로 다음 속성을 컨테이너 인스턴스에 적용합니다.

`ecs.ami-id`

인스턴스를 시작하는 데 사용된 AMI의 ID. 이 속성의 예시 값은 `ami-1234abcd`입니다.

`ecs.availability-zone`

인스턴스의 가용 영역. 이 속성의 예시 값은 `us-east-1a`입니다.

`ecs.instance-type`

인스턴스의 인스턴스 유형. 이 속성의 예시 값은 `g2.2xlarge`입니다.

`ecs.os-type`

인스턴스의 운영 체제. 이 속성에 사용 가능한 값은 `linux` 및 `windows`입니다.

`ecs.os-family`

인스턴스의 운영 체제 버전입니다.

Linux 인스턴스의 경우 유효한 값은 `LINUX`입니다. Windows 인스턴스의 경우 ECS는 값을 `WINDOWS_SERVER_<OS_Release>_<FULL or CORE>` 형식으로 설정합니다. 유효한 값은 `WINDOWS_SERVER_2022_FULL`, `WINDOWS_SERVER_2022_CORE`, `WINDOWS_SERVER_20H2_CORE`, `WINDOWS_SERVER_2019_FULL`, `WINDOWS_SERVER_2019_CORE` 및 `WINDOWS_SERVER_2016_FULL`입니다.

이는 Windows 컨테이너 및 Windows containers on AWS Fargate에서 중요한데, 모든 Windows 컨테이너의 OS 버전이 호스트의 OS 버전과 일치해야 하기 때문입니다. 컨테이너 이미지의 Windows 버전이 호스트와 다르면 컨테이너가 시작되지 않습니다. 자세한 내용은 Microsoft 설명서 웹 사이트의 Windows 컨테이너 버전 호환성을 참조하세요.

클러스터에서 여러 Windows 버전을 실행하는 경우 배치 제약 조건 `memberOf(attribute:ecs.os-family == WINDOWS_SERVER_<OS_Release>_<FULL or CORE>)`를 사용하여 동일한 버전에서 실행되는 EC2 인스턴스에 작업이 배치되도록 할 수 있습니다. 자세한 내용은 the section called “Amazon ECS 최적화 Windows AMI 메타데이터 검색” 섹션을 참조하세요.

`ecs.cpu-architecture`

인스턴스의 CPU 아키텍처입니다. 이 속성의 예시 값은 `x86_64`와 `arm64`입니다.

`ecs.vpc-id`

인스턴스가 시작된 VPC입니다. 이 속성의 예시 값은 `vpc-1234abcd`입니다.

`ecs.subnet-id`

인스턴스가 사용 중인 서브넷입니다. 이 속성의 예시 값은 `subnet-1234abcd`입니다.

선택적 속성

Amazon ECS는 컨테이너 인스턴스에 다음 속성을 추가할 수 있습니다.

`ecs.awsvpc-trunk-id`

이 속성이 있으면 인스턴스에 트렁크 네트워크 인터페이스가 있습니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 네트워크 인터페이스 증가 섹션을 참조하세요.

`ecs.outpost-arn`

이 속성이 있으면 Outpost의 Amazon 리소스 이름(ARN)이 포함됩니다. 자세한 내용은 the section called “AWS Outposts의 Amazon Elastic Container Service” 섹션을 참조하세요.

`ecs.capability.external`

이 속성이 있으면 인스턴스가 외부 인스턴스로 식별됩니다. 자세한 내용은 외부 시작 유형을 위한 Amazon ECS 클러스터 섹션을 참조하세요.

사용자 지정 속성

컨테이너 인스턴스에 사용자 지정 속성을 적용할 수 있습니다. 예를 들어 이름이 "stack"이고 값이 "prod"인 속성을 정의할 수 있습니다.

사용자 정의 속성을 지정할 때 다음 사항을 고려해야 합니다.

- `name`은 1~128자의 문자를 포함해야 하며 이름에는 문자(대문자 및 소문자), 숫자, 하이픈, 밑줄, 슬래시, 백슬래시 또는 마침표가 포함될 수 있습니다.
- `value`은 1~128자의 문자를 포함해야 하며 이름에는 문자(대문자 및 소문자), 숫자, 하이픈, 밑줄, 마침표, @ 기호, 슬래시, 백슬래시, 콜론 또는 공백이 포함될 수 있습니다. 값은 선행 또는 후행 공백을 포함할 수 없습니다.

Amazon ECS 작업에 대한 컨테이너 인스턴스를 정의하도록 표현식 생성

클러스터 쿼리는 객체를 그룹화할 수 있게 해 주는 표현식입니다. 예를 들어 가용 영역, 인스턴스 유형 또는 사용자 지정 메타데이터 같은 속성을 기준으로 컨테이너 인스턴스를 그룹화할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 속성 섹션을 참조하세요.

컨테이너 인스턴스 그룹을 정의한 후 그룹을 기준으로 태스크를 컨테이너 인스턴스에 배치하도록 Amazon ECS를 사용자 지정할 수 있습니다. 자세한 정보는 애플리케이션을 Amazon ECS 태스크로 실행 및 Amazon ECS 롤링 업데이트 배포 생성 섹션을 참조하세요. 컨테이너 인스턴스를 나열할 때 그룹 필터를 적용할 수도 있습니다.

표현식 구문

표현식에는 다음과 같은 구문이 있습니다.

```
subject operator [argument]
```

제목

평가할 속성 또는 필드.

`agentConnected`

Amazon ECS 컨테이너 에이전트 연결 상태를 기준으로 컨테이너 인스턴스를 선택합니다. 이 필터를 사용하여 분리된 컨테이너 에이전트가 있는 인스턴스를 검색할 수 있습니다.

유효 연산자: equals (==), not_equals (!=), in, not_in (!in), matches (=~), not_matches (!~)

`agentVersion`

Amazon ECS 컨테이너 에이전트 버전을 기준으로 컨테이너 인스턴스를 선택합니다. 이 필터를 사용하여 오래된 Amazon ECS 컨테이너 에이전트 버전을 실행하는 인스턴트를 검색할 수 있습니다.

유효 연산자: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

`attribute:`*`attribute-name`*

속성을 기준으로 컨테이너 인스턴스를 선택합니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 속성 섹션을 참조하세요.

`ec2InstanceId`

Amazon EC2 인스턴스 ID로 컨테이너 인스턴스를 선택합니다.

유효 연산자: equals (==), not_equals (!=), in, not_in (!in), matches (=~), not_matches (!~)

`registeredAt`

컨테이너 인스턴스 등록일을 기준으로 컨테이너 인스턴스를 선택합니다. 이 필터를 사용하여 새로 등록된 인스턴스 또는 매우 오래된 인스턴스를 검색할 수 있습니다.

유효 연산자: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

유효한 날짜 형식: 2018-06-18T22:28:28+00:00, 2018-06-18T22:28:28Z, 2018-06-18T22:28:28, 2018-06-18

`runningTasksCount`

실행 중인 작업 수를 기준으로 컨테이너 인스턴스를 선택합니다. 이 필터를 사용하여 비었거나 거의 비어 있는(실행 중인 작업이 거의 없는) 인스턴스를 검색할 수 있습니다.

유효 연산자: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

`task:group`

작업 그룹을 기준으로 컨테이너 인스턴스를 선택합니다. 자세한 정보는 [그룹 관련 Amazon ECS 작업](#) 을 참조하세요.

연산자

비교 연산자. 다음의 연산자가 지원됩니다.

연산자	설명
==, equals	문자열 같음
!=, not_equals	문자열 다름
>, greater_than	초과
>=, greater_than_equal	크거나 같음
<, less_than	미만
<=, less_than_equal	작거나 같음
exists	주체 있음
!exists, not_exists	주체 없음
in	인수 목록에 있는 값
!in, not_in	인수 목록에 없는 값
=~, matches	패턴 일치
!~, not_matches	패턴 불일치

> **Note**
>
> 단일 표현식은 괄호를 포함할 수 없습니다. 하지만 복합 표현식에서는 우선순위를 지정하기 위해 괄호를 사용할 수 있습니다.

인수

많은 연산자에서 인수는 리터럴 값입니다.

`in` 및 `not_in` 연산자에는 인수로 인수 목록이 필요합니다. 다음과 같이 인수를 지정합니다.

```
[argument1, argument2, ..., argumentN]
```

matches 및 not_matches 연산자에는 Java 정규식 구문을 준수하는 인수가 필요합니다. 자세한 내용은 java.util.regex.Pattern을 참조하세요.

복합 표현식

다음 부울 연산자를 사용하여 표현식을 결합할 수 있습니다.

- &&, 및
- ||, 또는
- !, not

괄호를 사용하여 우선순위를 지정할 수 있습니다.

```
(expression1 or expression2) and expression3
```

예제 표현식

다음은 예제 표현식입니다.

예제: 문자열 같음

다음 표현식은 지정된 인스턴스 유형을 가진 인스턴스를 선택합니다.

```
attribute:ecs.instance-type == t2.small
```

예제: 인수 목록

다음 표현식은 us-east-1a 또는 us-east-1b 가용 영역 내 인스턴스를 선택합니다.

```
attribute:ecs.availability-zone in [us-east-1a, us-east-1b]
```

예제: 복합 표현식

다음 표현식은 us-east-1d 가용 영역에 없는 G2 인스턴스를 선택합니다.

```
attribute:ecs.instance-type =~ g2.* and attribute:ecs.availability-zone != us-east-1d
```

예제: 태스크 선호도

다음 표현식은 `service:production` 그룹에서 태스크를 호스팅하고 있는 인스턴스를 선택합니다.

```
task:group == service:production
```

예제: 태스크 비선호도

다음 표현식은 database 그룹에서 태스크를 호스팅하고 있지 않은 인스턴스를 선택합니다.

```
not(task:group == database)
```

예: 실행 중인 태스크 개수

다음 표현식은 작업 한 개만 실행 중인 인스턴스를 선택합니다.

```
runningTasksCount == 1
```

예제: Amazon ECS 컨테이너 에이전트 버전

다음 표현식은 실행 중인 컨테이너 에이전트 버전이 1.14.5인 인스턴스를 선택합니다.

```
agentVersion < 1.14.5
```

예: 인스턴스 등록 시간

다음 표현식은 2018년 2월 13일 이전에 등록된 인스턴스를 선택합니다.

```
registeredAt < 2018-02-13
```

예제: Amazon EC2 인스턴스 ID

다음 표현식은 다음 Amazon EC2 인스턴스 ID를 갖는 인스턴스를 선택합니다.

```
ec2InstanceId in ['i-abcd1234', 'i-wxyx7890']
```

Amazon ECS 작업 배치 제약 조건 예제

다음은 작업 배치 제약 예제입니다.

이 예제에서는 `memberOf` 제약 조건을 사용하여 t2 인스턴스에 작업을 배치합니다. 다음 작업 CreateService, UpdateService, RegisterTaskDefinition 및 RunTask로 지정할 수 있습니다.

```
"placementConstraints": [
    {
        "expression": "attribute:ecs.instance-type =~ t2.*",
        "type": "memberOf"
    }
]
```

이 예제에서는 `memberOf` 제약을 사용하여 대몬(daemon) 서비스 `daemon-service` 작업 그룹의 다른 작업과 함께 인스턴스에 복제본 작업을 배치하고, 함께 지정된 모든 작업 배치 전략도 적용합니다. 이 제약으로 인해 대몬(daemon) 서비스 작업이 복제 서비스 작업보다 먼저 EC2 인스턴스에 배치됩니다.

`daemon-service`를 대몬(daemon) 서비스 이름으로 바꿉니다.

```
"placementConstraints": [
    {
        "expression": "task:group == service:daemon-service",
        "type": "memberOf"
    }
]
```

이 예제에서는 `memberOf` 제약을 사용하여 지정된 모든 작업 배치 전략을 적용한 `databases` 작업 그룹의 다른 태스크와 함께 인스턴스에 태스크를 배치합니다. 작업 그룹에 대한 자세한 내용은 그룹 관련 Amazon ECS 작업 섹션을 참조하세요. 다음 작업 CreateService, UpdateService, RegisterTaskDefinition 및 RunTask로 지정할 수 있습니다.

```
"placementConstraints": [
    {
        "expression": "task:group == databases",
        "type": "memberOf"
    }
]
```

distinctInstance 제약은 그룹의 각 태스크를 서로 다른 인스턴스에 배치합니다. 다음 작업 CreateService, UpdateService 및 RunTask로 지정할 수 있습니다.

Amazon ECS는 작업 배치를 위해 원하는 작업 상태를 확인합니다. 예를 들어 기존 작업의 원하는 상태가 STOPPED인 경우(하지만 마지막 상태는 그렇지 않음), distinctInstance 배치 제약에도 불구하고 새로 들어오는 작업을 동일한 인스턴스에 배치할 수 있습니다. 따라서 동일한 인스턴스에 마지막 RUNNING 상태인 작업 2개가 표시될 수 있습니다.

```
"placementConstraints": [
    {
        "type": "distinctInstance"
    }
]
```

Amazon ECS 독립 실행형 작업

일부 작업을 수행한 후 중지하는 애플리케이션(예: 배치 프로세스)이 있는 경우 애플리케이션을 작업으로 실행할 수 있습니다. 작업을 일회성으로 실행하려는 경우 콘솔, AWS CLI, API 또는 SDK를 사용할 수 있습니다.

속도 기반, cron 기반 또는 일회성 일정에 따라 애플리케이션을 실행해야 하는 경우 EventBridge 스케줄러를 사용하여 예약을 생성할 수 있습니다.

작업 워크플로

Amazon ECS 작업(독립 실행형 작업 또는 Amazon ECS 서비스)을 시작하면 작업이 생성되고 처음에 PROVISIONING 상태로 이동합니다. Amazon ECS는 작업을 배치하기 위한 컴퓨팅 용량을 찾아야 하기 때문에 작업이 PROVISIONING 상태인 경우 작업과 컨테이너 모두 존재하지 않습니다.

Amazon ECS는 시작 유형 또는 용량 공급자 구성을 기반으로 작업에 적합한 컴퓨팅 용량을 선택합니다. Fargate 및 Amazon EC2 시작 유형 모두에서 용량 공급자와 용량 공급자 전략을 사용할 수 있습

니다. Fargate를 사용하면 클러스터 용량의 프로비저닝, 구성 및 규모 조정에 대해 걱정할 필요가 없습니다. Fargate에서 작업에 필요한 모든 인프라 관리를 처리합니다. EC2 시작 유형의 경우 Amazon EC2 인스턴스를 클러스터에 등록하여 클러스터 용량을 관리하거나 클러스터 Auto Scaling을 사용하여 컴퓨팅 용량 관리를 단순화할 수 있습니다. 클러스터 Auto Scaling은 클러스터 용량을 동적으로 조정하므로 사용자는 작업 실행에만 집중할 수 있습니다. Amazon ECS는 CPU와 메모리, 배치 제약 조건 및 전략과 같이 작업 정의에서 지정한 요구 사항에 따라 작업 배치 위치를 결정합니다. 자세한 내용은 Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법 섹션을 참조하세요.

Managed Scaling이 활성화된 용량 공급자를 사용하는 경우 컴퓨팅 용량 부족으로 인해 시작할 수 없는 태스크는 즉시 실패하지 않고 `PROVISIONING` 상태로 이동됩니다. Amazon ECS는 작업을 배치할 용량을 찾은 후 필요한 연결(예: `awsvpc` 모드에서 작업의 탄력적 네트워크 인터페이스(ENI))을 프로비저닝합니다. Amazon ECS 컨테이너 에이전트를 사용하여 컨테이너 이미지를 가져온 다음, 컨테이너를 시작합니다. 프로비저닝이 완료되고 관련 컨테이너가 시작된 후 Amazon ECS는 작업을 `RUNNING` 상태로 이동합니다. 작업 상태에 대한 자세한 내용은 Amazon ECS 작업 수명 주기 섹션을 참조하세요.

애플리케이션을 Amazon ECS 태스크로 실행

AWS Management Console을 사용하여 일회성 프로세스에 대한 작업을 생성할 수 있습니다.

독립 실행형 작업(AWS Management Console)을 생성하는 방법

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. Amazon ECS 콘솔을 사용하면 클러스터 세부 정보 페이지 또는 작업 정의 개정 목록에서 독립 실행형 작업을 생성할 수 있습니다. 다음 단계를 사용하여 선택한 리소스 페이지에 따라 독립 실행형 작업을 생성합니다.

서비스 시작	단계
클러스터 세부 정보 페이지	a. 클러스터 페이지에서 서비스를 생성할 클러스터를 선택합니다. b. 작업 탭에서 작업 실행을 선택합니다.
작업 정의 개정 페이지	a. 작업 정의 페이지에서 작업 정의 패밀리를 선택하여 해당 패밀리에 대한 개정을 표시합니다.

서비스 시작	단계
	b. 사용할 개정을 선택합니다. c. 배포 메뉴에서 작업 실행을 선택합니다.

3. 기존 클러스터에 대해 클러스터를 선택합니다.

   클러스터 생성을 선택하여 새 클러스터에서 작업을 실행합니다.

4. 클러스터 인프라에 태스크를 배포하는 방식을 선택합니다. 컴퓨팅 구성에서 옵션을 선택합니다. 용량 공급자 전략을 사용하려면 용량 공급자를 클러스터 수준에서 구성해야 합니다.

   용량 공급자를 사용하도록 클러스터를 구성하지 않은 경우 대신 시작 유형을 사용합니다.

배포 방법	단계
용량 제공자 전략	a. 컴퓨팅 옵션(Compute options) 섹션에서 용량 공급자 전략(Capacity provider strategy)을 선택합니다. b. 전략을 선택합니다. • 클러스터의 기본 용량 공급자 전략을 사용하려면 클러스터 기본값 사용(Use cluster default)을 선택합니다. • 클러스터에 기본 용량 공급자 전략이 없거나 사용자 지정 전략을 사용하려면 사용자 지정 사용(Use custom), 용량 공급자 전략 추가(Add capacity provider strategy)를 선택하고 기본(Base), 용량 공급자(Capacity provider) 및

배포 방법	단계
	가중치(Weight)를 지정하여 사용자 지정 용량 공급자 전략을 정의합니다. **Note** 전략에서 용량 공급자를 사용하려면 용량 공급자가 클러스터와 연결되어야 합니다.
시작 유형	a. 컴퓨팅 옵션(Compute options) 섹션에서 시작 유형(Launch type)을 선택합니다. b. 시작 유형(Launch type)에서 시작 유형을 선택합니다. c. (선택 사항) Fargate 시작 유형이 지정되면 플랫폼 버전(Platform version)에서 사용할 플랫폼 버전을 지정합니다. 플랫폼 버전을 지정하지 않으면 LATEST 플랫폼 버전이 사용됩니다.

5. 배포 구성에서 다음을 수행합니다.

   a. 작업 정의에 작업 정의를 입력합니다.

> **Important**
>
> 콘솔은 선택 항목의 유효성을 검사하여 선택한 작업 정의 패밀리 및 개정이 정의된 컴퓨팅 구성과 호환되는지 확인합니다.

b. 원하는 작업(Desired tasks)에 시작할 작업 수를 입력합니다.

c. 작업 그룹에 작업 그룹 이름을 입력합니다.

6. 태스크 정의에서 awsvpc 네트워크 모드를 사용할 경우, 네트워킹(Networking)을 펼칩니다. 다음 단계에 따라 사용자 지정 구성을 지정합니다.

   a. VPC에서 사용할 VPC 선택합니다.

   b. 서브넷(Subnets)에서 태스크를 배치할 때 작업 스케줄러가 고려할 하나 이상의 서브넷을 VPC에서 선택합니다.

   c. 보안 그룹의 경우 기존 보안 그룹을 선택하거나 새 보안 그룹을 생성할 수 있습니다. 기존 보안 그룹을 사용하려면 보안 그룹을 선택하고 다음 단계로 이동합니다. 새 보안 그룹을 만들려면 새 보안 그룹 생성(Create a new security group)을 선택합니다. 보안 그룹 이름 및 설명을 지정한 다음 보안 그룹에 대해 하나 이상의 인바운드 규칙을 추가해야 합니다.

   d. 퍼블릭 IP(Public IP)에서 작업의 탄력적 네트워크 인터페이스(ENI)에 퍼블릭 IP 주소를 자동 할당할지를 선택합니다.

      AWS Fargate 작업은 퍼블릭 서브넷에서 실행될 때 퍼블릭 IP 주소가 할당될 수 있으므로 인터넷에 연결되는 경로가 있습니다. 이 필드를 사용하여 EC2 작업에 퍼블릭 IP를 할당할 수 없습니다. 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션 및 Amazon ECS 작업을 위한 네트워크 인터페이스 할당을 참조하십시오.

7. 작업에서 배포 시 구성과 호환되는 데이터 볼륨을 사용하는 경우 볼륨을 확장하여 볼륨을 구성할 수 있습니다.

   볼륨 이름과 볼륨 유형은 작업 정의 개정을 생성할 때 구성되며 독립 실행형 작업을 실행할 때는 변경할 수 없습니다. 볼륨 이름과 유형을 업데이트하려면 새 작업 정의 개정을 생성하고 새 개정을 사용해 작업을 실행해야 합니다.

이 볼륨 유형을 구성하는 방법	조치	
Amazon EBS	a. EBS 볼륨 유형에서 작업에 연결할 EBS 볼륨 유형을 선택합니다. b. 크기(GiB)에 유효한 볼륨 크기 값을 기비바이트(GiB) 단위로 입력합니다. 최소 1GiB 및 최대 16,384GiB 볼륨 크기를 지정할 수 있습니다. 스냅샷 ID를 제공하지 않는 한, 이 값은 필수입니다. c. IOPS의 경우 볼륨이 제공해야 하는 최대 초당 입출력 작업(IOPS) 수를 입력합니다. 이 값은 `io1`, `io2`, `gp3` 볼륨 유형에서만 구성할 수 있습니다. d. 처리량(MiB/s)에 볼륨에서 제공해야 하는 처리량을 초당 메비바이트 단위(MiBps 또는 MiB/s)로 입력합니다. 이 값은 `gp3` 볼륨 유형에서만 구성 가능합니다. e. 스냅샷 ID에서 기존 Amazon EBS 볼륨 스냅샷을 선택하거나 스냅샷에서 볼륨을 생성하려는 경우 스냅샷의 ARN을 입력합니다. 스냅샷 ID를 선택하거나 입력하지 않고 비어 있는 새 볼륨을 생성할 수도 있습니다.	

이 볼륨 유형을 구성하는 방법	조치	
	f. 스냅샷 ID를 지정하는 경우 볼륨 초기화 속도(MiB/s)를 지정할 수 있습니다. 100~300MiB/s 사이의 값을 입력하면 볼륨 생성을 위해 스냅샷 ID를 사용하여 지정된 스냅샷에서 데이터가 로드되는 속도가 결정됩니다. g. 종료 정책의 경우 작업이 종료된 후에도 작업에 연결하도록 구성된 볼륨을 보존하려면 확인란의 선택을 취소합니다. 기본적으로 작업에 연결된 EBS 볼륨은 작업이 종료될 때 삭제됩니다. h. 파일 시스템 유형에서 볼륨의 데이터 스토리지 및 검색에 사용할 파일 시스템 유형을 선택합니다. 운영 체제 기본값 또는 특정 파일 시스템 유형을 선택할 수 있습니다. Linux의 기본값은 XFS입니다. 스냅샷에서 생성된 볼륨의 경우 스냅샷을 생성할 때 볼륨에서 사용하던 것과 동일한 파일 시스템 유형을 지정해야 합니다. 파일 시스템 유형이 일치하지 않으면 작업이 시작되지 않습니다.	

이 볼륨 유형을 구성하는 방법	조치	
	i. 인프라 역할에서 Amazon ECS가 작업에 대한 Amazon EBS 볼륨을 관리할 수 있도록 하는 데 필요한 권한이 있는 IAM 역할을 선택합니다. `AmazonECSInfrastructureRolePolicyForVolumes` 관리형 정책을 역할에 연결하거나 정책을 지침으로 사용하여 특정 요구 사항에 맞는 권한을 보유한 자체 정책을 생성하고 연결할 수 있습니다. 필요한 권한에 대한 자세한 내용은 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요. j. 암호화에서 Amazon EBS 암호화를 기본 설정으로 사용하려면 기본값을 선택합니다. 계정에 암호화 기본 제공이 구성된 경우 설정에 지정된 AWS Key Management Service(AWS KMS) 키로 볼륨이 암호화됩니다. 기본값을 선택하고 Amazon EBS 기본 암호화가 켜져 있지 않으면 볼륨이 암호화되지 않습니다. 사용자 지정을 선택하면 볼륨 암호화에 대해 원하는	

이 볼륨 유형을 구성하는 방법	조치	
	AWS KMS key를 지정할 수 있습니다. 없음을 선택한 경우 암호화가 기본적으로 구성되지 않았거나 암호화된 스냅샷에서 볼륨을 생성하지 않으면 볼륨이 암호화되지 않습니다. k. 암호화에 사용자 지정을 선택한 경우 사용하려는 AWS KMS key를 지정해야 합니다. KMS 키에서 AWS KMS key를 선택하거나 키 ARN을 입력합니다. 대칭 고객 관리형 키를 사용하여 볼륨을 암호화하려는 경우 AWS KMS key 정책에 올바른 권한이 정의되었는지 확인합니다. 자세한 내용은 Data encryption for Amazon EBS volumes를 참조하세요. l. (선택 사항) 태그에서 작업 정의의 태그를 전파하거나 자체 태그를 제공하여 Amazon EBS 볼륨에 태그를 추가할 수 있습니다. 작업 정의에서 태그를 전파하려면 태그 전파 시작에서 작업 정의를 선택합니다. 전파 안 함을 선택하거나	

이 볼륨 유형을 구성하는 방법	조치	
	값을 선택하지 않으면 태그가 전파되지 않습니다. 고유한 태그를 제공하려면 태그 추가를 선택하고 추가하는 각 태그의 키와 값을 제공합니다. Amazon EBS 볼륨 태그 지정에 대한 자세한 내용은 Amazon EBS 볼륨 태그 지정을 참조하세요.	

8. (선택 사항) 기본 외에 다른 태스크 배치 전략을 사용하는 경우 태스크 배치(Task Placement)를 펼치고 다음의 옵션 중에서 선택하세요.

   자세한 내용은 Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법 섹션을 참조하세요.

   - AZ Balanced Spread - 작업을 가용 영역과 가용 영역의 컨테이너 인스턴스에 분산합니다.
   - AZ Balanced BinPack - 작업을 가용 영역과 가용 메모리가 최소인 컨테이너 인스턴스에 분산합니다.
   - BinPack - 사용 가능한 CPU 또는 메모리 최소량에 따라 작업을 분산합니다.
   - One Task Per Host - 각 컨테이너 인스턴스에 서비스의 작업을 최대 1개 배치합니다.
   - Custom - 자체 작업 배치 전략을 정의합니다.

   사용자 지정(Custom)을 선택하는 경우, 태스크를 배치할 알고리즘과 태스크 배치 중에 고려할 규칙을 정의합니다.

   - 전략(Strategy) 아래에 있는 유형(Type)과 필드(Field)의 경우, 알고리즘과 알고리즘에 사용할 엔터티를 선택합니다.

     최대 5개의 전략을 입력할 수 있습니다.
   - 제약 아래에 있는 유형과 표현식에서 해당 제약에 사용할 규칙과 속성을 선택합니다.

예를 들어 TC 인스턴스에 태스크를 배치하기 위한 제약을 설정할 경우, 식(Expression)에는 attribute:ecs.instance-type =~ t2.*를 입력합니다.

초대 10개의 제약을 입력할 수 있습니다.

9. (선택 사항) 태스크 IAM 역할이나 태스크 정의에 정의된 태스크 실행 역할을 재정의할 경우, 태스크 재정의(Task overrides)를 펼친 다음, 다음 단계를 완료하세요.

   a. 작업 역할에서 이 작업에 대한 IAM 역할을 선택합니다. 자세한 내용은 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.

      `ecs-tasks.amazonaws.com` 신뢰 관계를 가진 역할만 표시됩니다. 작업에 대한 IAM 역할을 생성하는 방법에 관한 지침은 작업 IAM 역할 생성 섹션을 참조하세요.

   b. 작업 실행 역할에서 작업 실행 역할을 선택합니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

10. (선택 사항) 컨테이너 명령과 환경 변수를 재정의하려면 컨테이너 재정의(Container Overrides)를 펼친 다음, 컨테이너를 펼칩니다.

    - 작업 정의 명령 이외의 명령을 컨테이너로 보내려면 명령 재정의에 Docker 명령을 입력합니다.
    - 환경 변수를 추가하려면 환경 변수 추가(Add Environment Variable)를 선택합니다. 키(Key)에 환경 변수의 이름을 입력합니다. 값(Value)에 환경 값의 문자열 값을 입력합니다(큰따옴표(" ")로 묶지 않음).

      AWS에서 문자열을 큰따옴표(" ")로 묶고 문자열을 다음 형식으로 컨테이너에 전달합니다.

      ```
      MY_ENV_VAR="This variable contains a string."
      ```

11. (선택 사항) 태스크를 식별하려면 태그(Tags) 섹션을 펼친 다음, 태그를 구성합니다.

    Amazon ECS가 새로 시작한 모든 작업에 클러스터 이름과 작업 정의 태그를 자동으로 지정하려면 Amazon ECS 관리형 태그(Turn on Amazon ECS managed tags) 사용을 선택한 다음 작업 정의(Task definitions)를 선택합니다.

    태그를 추가하거나 제거합니다.

    - [태그 추가] 새로운 태그(Add tag)를 선택하고 다음을 수행합니다.
      - 키에서 키 이름을 입력합니다.
      - 값에 키 값을 입력합니다.
    - [태그 제거] 태그 옆에 있는 태그 제거를 선택합니다.

12. 생성(Create)을 선택합니다.

Amazon EventBridge Scheduler를 사용하여 Amazon ECS 태스크 예약

EventBridge Scheduler는 하나의 중앙 관리형 서비스에서 작업을 생성, 실행 및 관리할 수 있는 서버리스 스케줄러입니다. 이벤트 버스 및 규칙과 관계없이 일회성 및 반복 예약 기능을 제공합니다. EventBridge 스케줄러는 고도로 사용자 지정이 가능하며, 광범위한 대상 API 작업 및 AWS 서비스를 통해 EventBridge 예약 규칙보다 향상된 확장성을 제공합니다. EventBridge 스케줄러는 EventBridge 스케줄러 콘솔에서 작업에 맞게 구성할 수 있는 다음과 같은 일정을 제공합니다.

- 비율 기반
- Cron 기반

  모든 시간대에서 cron 기반 일정을 구성할 수 있습니다.
- 일회성 일정

  모든 시간대에서 일회성 일정을 구성할 수 있습니다.

Amazon EventBridge Scheduler를 사용하여 Amazon ECS를 예약할 수 있습니다.

Amazon ECS 콘솔에서 예약된 작업을 생성할 수 있지만 현재 EventBridge 스케줄러 콘솔은 더 많은 기능을 제공합니다.

작업을 예약하려면 먼저 다음 단계를 완료해야 합니다.

1. VPC 콘솔을 사용하여 작업이 실행되는 서브넷 ID와 서브넷의 보안 그룹 ID를 가져옵니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC의 서브넷와 보안 그룹을 사용하여 AWS 리소스에 대한 트래픽 제어를 참조하세요.
2. EventBridge 스케줄러 실행 역할을 구성합니다. 자세한 내용은 Amazon EventBridge Scheduler 사용 설명서의 Set up the execution role을 참조하세요.
3. 용량 공급자를 사용하여 태스크를 실행하려면 클러스터와 연결된 용량 공급자가 있어야 합니다.

콘솔을 사용하여 새 일정을 생성하는 방법

1. https://console.aws.amazon.com/scheduler/home에서 Amazon EventBridge Scheduler 콘솔을 엽니다.
2. 일정 페이지에서 일정 생성을 선택합니다.

3. 일정 세부 정보 지정 페이지의 일정 이름 및 설명 섹션에서 다음을 수행합니다.

   a. 일정 이름에 일정의 이름을 입력합니다. 예를 들어 **MyTestSchedule**입니다.

   b. (선택 사항) 설명에 일정에 대한 설명을 입력합니다. 예를 들어 **TestSchedule**입니다.

   c. 일정 그룹에서 일정 그룹을 선택합니다. 그룹이 없는 경우 기본값을 선택합니다. 일정 그룹을 생성하려면 자체 일정 생성을 선택합니다.

      일정 그룹을 사용하여 일정 그룹에 태그를 추가합니다.

4. 일정 옵션을 선택합니다.

발생	수행할 작업	
일회성 일정 일회성 일정은 사용자가 지정하는 날짜와 시간에 한 번만 대상을 간접적으로 간접 호출합니다.	날짜 및 시간에 대해 다음을 수행합니다. • YYYY/MM/DD 형식으로 유효한 날짜를 입력합니다. • 24시간 hh:mm 형식으로 타임스탬프를 입력합니다. • 시간대에서 시간대를 선택하세요.	
반복되는 일정 반복 일정은 cron 표현식 또는 rate 표현식을 사용하여 지정한 속도로 대상을 간접적으로 간접 호출합니다.	a. 일정 유형에서 다음 중 하나를 수행합니다. • cron 표현식을 사용하여 일정을 정의하려면 Cron 기반 일정을 선택하고 cron 표현식을 입력합니다. • rate 표현식을 사용하여 일정을 정의하려면 Rate 기반 일정을 선택하고 rate 표현식을 입력합니다. cron 및 rate 표현식에 대한 자세한 내용은	

발생	수행할 작업	
	Amazon EventBridge Scheduler 사용 설명서의 EventBridge 스케줄러의 스케줄 유형을 참조하세요. b. 유연한 기간에서 끄기를 선택하여 옵션을 끄거나 미리 정의된 기간 중 하나를 선택합니다. 예를 들어, 15분을 선택하고 1시간에 한 번씩 대상을 간접적으로 간접 호출하도록 반복 일정을 설정하면 일정은 매시간 시작 후 15분 이내에 실행됩니다.	

5. (선택 사항) 이전 단계에서 반복 일정을 선택한 경우 기간 섹션에서 다음을 수행합니다.
   a. 시간대에서 시간대를 선택합니다.
   b. 시작 날짜 및 시간에 YYYY/MM/DD 형식으로 유효한 날짜를 입력한 다음 24시간 hh:mm 형식으로 타임스탬프를 지정합니다.
   c. 종료 날짜 및 시간에 YYYY/MM/DD 형식으로 유효한 날짜를 입력한 다음 24시간 hh:mm 형식으로 타임스탬프를 지정합니다.
6. 다음을 선택합니다.
7. 대상 선택 페이지에서 다음을 수행합니다.
   a. 모든 API를 선택한 다음 검색 상자에 ECS를 입력합니다.
   b. Amazon ECS를 선택합니다.
   c. 검색 상자에 RunTask를 입력한 다음 RunTask를 선택합니다.
   d. ECS 클러스터에서 클러스터를 선택합니다.
   e. ECS 작업에서 작업에 사용할 작업 정의를 선택합니다.
   f. 클러스터 인프라에 태스크를 배포하는 방식을 선택합니다. 컴퓨팅 옵션을 확장한 다음 다음 옵션 중 하나를 선택

컴퓨팅 옵션	단계	
용량 제공자 전략	a. 용량 공급자 전략을 선택합니다. b. 전략을 선택합니다. • 기본 용량 공급자 전략을 사용하려면 클러스터 기본값 사용을 선택합니다. • 사용자 지정 전략을 사용하려면 사용자 지정 사용을 선택합니다. 그런 다음 기본, 용량 공급자, 가중치를 입력합니다. EC2의 경우, 용량 공급자는 Auto Scaling 그룹입니다.	
시작 유형	a. 시작 유형을 선택합니다. b. 시작 유형(Launch type)에서 시작 유형을 선택합니다. c. Fargate 시작 유형이 지정되면 플랫폼 버전에서 사용할 플랫폼 버전을 지정합니다.	

g. 서브넷에 작업을 실행할 서브넷 ID를 입력합니다.

h. 보안 그룹에 서브넷의 보안 그룹 ID를 입력합니다.

i. (선택 사항) 기본값이 아닌 작업 배치 전략을 사용하려면 배치 제약을 확장한 다음 제약을 입력합니다.

자세한 내용은 Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법 섹션을 참조하세요.

j. (선택 사항) 작업을 식별하려면 태그에서 태그를 구성합니다.

Amazon ECS가 새로 시작되는 모든 작업에 작업 정의 태그를 자동으로 지정하도록 하려면 Amazon ECS 관리형 태그 활성화를 선택합니다.

8. 다음을 선택합니다.
9. 설정 페이지에서 다음 작업을 수행합니다.

a. 일정을 켜려면 일정 상태에서 일정 활성화를 토글합니다.

b. 일정에 대한 재시도 정책을 구성하려면 재시도 정책 및 데드-레터 큐(DLQ)에서 다음을 수행합니다.

- 재시도를 토글합니다.
- 이벤트의 최대 보존 시간에 EventBridge 스케줄러가 처리되지 않은 이벤트를 유지해야 하는 최대 시간과 분을 입력합니다.
- 최대 시간은 24시간입니다.
- 최대 재시도 횟수에는 대상이 오류를 반환할 경우 EventBridge 스케줄러가 일정을 재시도하는 최대 횟수를 입력합니다.

  최댓값은 185회입니다.

재시도 정책을 사용하면 일정이 대상을 간접적으로 간접 호출하지 못할 경우 EventBridge 스케줄러가 일정을 다시 실행합니다. 구성된 경우 일정에 대한 최대 보존 기간과 재시도 횟수를 설정해야 합니다.

c. EventBridge 스케줄러가 전송되지 않은 이벤트를 저장하는 위치를 선택합니다.

DLQ(Dead Letter Queue) 옵션	수행할 작업	
저장 안 함	None을 선택합니다.	
일정을 생성하는 위치와 같은 AWS 계정에 이벤트 저장	a. 내 AWS 계정의 Amazon SQS 대기열을 DLQ로 선택을 선택합니다.	

DLQ(Dead Letter Queue) 옵션	수행할 작업	
	b. Amazon SQS 대기열의 Amazon 리소스 이름(ARN)을 선택합니다.	
일정을 생성하는 위치와 다른 AWS 계정에 이벤트 저장	a. 다른 AWS 계정의 Amazon SQS 대기열을 DLQ로 지정을 선택합니다. b. Amazon SQS 대기열의 Amazon 리소스 이름(ARN)을 입력합니다.	

d. 고객 관리형 키를 사용하여 대상 입력을 암호화하려면 암호화에서 암호화 설정 사용자 지정(고급)을 선택합니다.

   이 옵션을 선택하는 경우 기존 KMS 키 ARN을 입력하거나 AWS KMS key 생성을 선택하여 AWS KMS 콘솔로 이동합니다. EventBridge 스케줄러가 저장 데이터를 암호화하는 방법에 대한 자세한 내용은 Amazon EventBridge Scheduler 사용 설명서의 Encryption at rest를 참조하세요.

e. 권한에서 기존 역할 사용을 선택한 다음 역할을 선택합니다.

   EventBridge 스케줄러가 새 실행 역할을 생성하도록 하려면 이 일정에 대한 새 역할 생성을 선택합니다. 그런 다음 역할 이름을 입력합니다. 이 옵션을 선택하면 EventBridge 스케줄러가 템플릿 대상에 필요한 필수 권한을 역할에 연결합니다.

10. 다음을 선택합니다.
11. 일정 검토 및 생성 페이지에서 일정의 세부 정보를 검토합니다. 각 섹션에서 편집을 선택하여 해당 단계로 돌아가서 세부 정보를 편집합니다.
12. 일정 생성을 선택합니다.

    일정 페이지에서 새 일정과 기존 일정 목록을 볼 수 있습니다. 상태 열에서 새 일정이 활성화됨 상태인지 확인합니다.

다음 단계

EventBridge 스케줄러 콘솔 또는 AWS CLI를 사용하여 일정을 관리할 수 있습니다. 자세한 내용은 Amazon EventBridge Scheduler 사용 설명서의 Managing a schedule을 참조하세요.

Amazon ECS 태스크 중지

더 이상 독립 실행형 태스크를 계속 실행할 필요가 없는 경우 작업을 중지할 수 있습니다. Amazon ECS 콘솔을 사용하면 하나 이상의 작업을 쉽게 중지할 수 있습니다.

독립형 중지된 태스크는 다시 시작할 수 없습니다.

서비스를 중지하려는 경우 콘솔을 사용하여 Amazon ECS 서비스 삭제 섹션을 참조하세요.

독립 실행형 작업을 중지하는 방법(AWS Management Console)

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 페이지에서 클러스터를 선택하여 클러스터 세부 정보 페이지로 이동합니다.
4. 클러스터 세부 정보 페이지에서 작업 탭을 선택합니다.
5. 시작 유형 필터링 목록을 사용하여 시작 유형별로 작업을 필터링할 수 있습니다.

중지할 작업	단계
하나 이상	a. 작업을 선택하고 중지, 선택 중지를 선택합니다. b. 작업 중지 확인 페이지에서 중지를 선택합니다.
모두	**Important** 콘솔을 사용하여 모든 작업을 중지하려는 경우 Amazon ECS는 모든 독립 실행형 작업과 서비스의 일부인 작업을 중지합니다. 따라서 이

중지할 작업	단계
	옵션을 사용할 때는 주의하는 것이 좋습니다. a. 중지, 모두 중지를 선택합니다. b. 작업 중지 확인 페이지에서 모든 작업 중지를 입력하고 중지를 선택합니다.

Amazon ECS 서비스

Amazon ECS 서비스를 사용하면 Amazon ECS 클러스터에서 지정된 수의 태스크 정의 인스턴스를 동시에 실행하고 유지 관리할 수 있습니다. 태스크가 실패하거나 중지되면 Amazon ECS 서비스 스케줄러가 태스크 정의의 다른 인스턴스를 시작하여 해당 태스크를 대체합니다. 이렇게 하면 서비스에서 원하는 수의 태스크를 유지 관리하는 데 도움이 됩니다.

또한 선택적으로 로드 밸런서 뒤에서 서비스를 실행할 수도 있습니다. 로드 밸런서는 서비스와 연결된 태스크 간에 트래픽을 분산합니다.

서비스 스케줄러는 장기 실행 상태 비저장 서비스 및 애플리케이션에 사용하는 것이 좋습니다. 서비스 스케줄러는 지정한 일정 전략을 따르는지 확인하고 태스크가 실패하는 경우 태스크 일정을 조정합니다. 예를 들어 기본 인프라가 실패하면 서비스 스케줄러가 태스크 일정을 조정합니다. 작업 배치 전략과 제약 조건을 사용하여 스케줄러가 태스크를 배치하고 종료하는 방법을 사용자 지정할 수 있습니다. 서비스의 태스크가 중지되는 경우 스케줄러는 새 태스크를 시작하여 대체합니다. 이 프로세스는 서비스가 사용하는 일정 전략에 따라 서비스가 원하는 태스크 수에 도달할 때까지 계속됩니다. 서비스의 일정 전략은 서비스 유형이라고도 합니다.

또한 서비스 스케줄러는 컨테이너 상태 확인 또는 로드 밸런서 대상 그룹 상태 확인이 실패한 후 비정상으로 확인된 작업을 대체합니다. 이러한 대체는 `maximumPercent` 및 `desiredCount` 서비스 정의 파라미터에 따라 달라집니다. 작업이 비정상으로 표시되면 서비스 스케줄러는 먼저 대체 작업을 시작합니다. 그러면 다음과 같이 진행됩니다.

- 대체 작업의 상태가 `HEALTHY`인 경우 서비스 스케줄러는 비정상 작업을 중지합니다.

- 대체 작업의 상태가 UNHEALTHY이면, 스케줄러는 비정상 대체 작업 또는 기존의 비정상 작업을 중지하여 총 작업 수가 desiredCount와 같아지도록 합니다.

maximumPercent 파라미터로 인해 스케줄러가 대체 작업을 먼저 시작하는 것이 제한되는 경우, 스케줄러는 비정상 작업을 한 번에 하나씩 임의로 중지하여 용량을 확보한 다음 대체 작업을 시작합니다. 비정상 작업이 모두 정상 작업으로 대체될 때까지 시작 및 중지 프로세스가 계속됩니다. 비정상 작업이 모두 대체되고 정상 작업만 실행 중인 경우 총 작업 수가 desiredCount를 초과하면 총 작업 수가 desiredCount와 같아질 때까지 정상 작업을 무작위로 중지합니다. maximumPercent 및 desiredCount에 대한 자세한 정보는 Service definition parameters를 참조하세요.

서비스 스케줄러에는 태스크가 반복적으로 시작하는 데 실패할 경우 태스크가 다시 시작되는 빈도를 제한하는 로직이 포함됩니다. 태스크가 RUNNING 상태가 되지 않고 중지된 경우, 서비스 스케줄러는 시작 시도 속도를 늦추기 시작하고 서비스 이벤트 메시지를 전송합니다. 이 동작은 해당 문제를 해결하기 전에 실패한 태스크에 불필요한 리소스가 사용되는 것을 방지합니다. 서비스가 업데이트되면 서비스 스케줄러는 정상적인 예약 동작을 다시 시작합니다. 자세한 내용은 Amazon ECS 서비스 제한 로직 및 Amazon ECS 서비스 이벤트 메시지 보기(을)를 참조하세요.

인프라 컴퓨팅 옵션

태스크를 분산하는 두 가지 컴퓨팅 옵션이 있습니다.

- 용량 공급자 전략은 Amazon ECS가 하나 또는 여러 개의 용량 공급자에 태스크를 분배하도록 합니다.

  용량 공급자 전략의 경우, 콘솔이 기본적으로 컴퓨팅 옵션을 선택합니다. 다음은 콘솔에서 기본값을 선택하는 순서에 대한 설명입니다.

  - 클러스터에 기본 용량 공급자 전략이 정의된 경우 이 전략이 선택됩니다.
  - 클러스터에 기본 용량 공급자 전략이 정의되어 있지 않지만 Fargate 용량 공급자가 클러스터에 추가되어 있는 경우 FARGATE 용량 공급자를 사용하는 사용자 지정 용량 공급자 전략이 선택됩니다.
  - 클러스터에 기본 용량 공급자 전략이 정의되어 있지 않지만 하나 이상의 Auto Scaling 그룹 용량 공급자가 클러스터에 추가된 경우 사용자 지정 사용(고급) 옵션이 선택되고 전략을 수동으로 정의해야 합니다.
  - 클러스터에 기본 용량 공급자 전략이 정의되어 있지 않고 클러스터에 용량 공급자가 추가되지 않은 경우 Fargate 시작 유형이 선택됩니다.
- 시작 유형에서는 Amazon ECS가 클러스터에 등록된 Fargate 또는 EC2 인스턴스에 대한 태스크가 Amazon ECS에서 곧바로 시작됩니다.

기본적으로 서비스는 클러스터 VPC의 서브넷에서 시작됩니다.

서비스 Auto Scaling

서비스 오토 스케일링은 Amazon ECS 서비스에서 원하는 태스크 수를 자동으로 늘리거나 줄이는 기능입니다. Amazon ECS는 Application Auto Scaling 서비스를 활용하여 이 기능을 제공합니다.

자세한 내용은 Amazon ECS 서비스 자동 조정 섹션을 참조하세요.

서비스 로드 밸런싱

AWS Fargate에서 호스팅되는 Amazon ECS 서비스는 Application Load Balancer, Network Load Balancer, Gateway Load Balancer를 지원합니다. 다음 테이블을 참조하여 사용할 로드 밸런서 유형에 대해 알아보세요.

로드 밸런서 유형	다음과 같은 경우에 사용	
Application Load Balancer	HTTP/HTTPS(또는 계층 7) 트래픽을 라우팅합니다. Application Load Balancer는 Amazon ECS 서비스에 사용할 수 있는 유용한 몇 가지 기능을 제공합니다. • 각 서비스는 여러 대상 그룹을 지정할 경우 여러 로드 밸런서의 트래픽을 처리하며 여러 로드 밸런스 포트를 노출합니다. • 이 기능은 Fargate와 EC2 인스턴스 모두에서 호스팅되는 작업에서 지원됩니다. • Application Load Balancer는 컨테이너가 동적 호스트 포트 매핑을 사용하도록 허용합니다(그래서 컨테이너 인	

로드 밸런서 유형	다음과 같은 경우에 사용	
	스턴스마다 동일 서비스의 여러 작업이 허용됩니다). • Application Load Balancer는 포트 기반 라우팅과 우선순위 규칙을 지원합니다(그래서 여러 서비스가 단일 Application Load Balancer에서 동일한 리스너 포터를 사용할 수 있습니다).	
Network Load Balancer	TCP 또는 UDP(또는 계층 4) 트래픽을 라우팅합니다.	
Gateway Load Balancer	TCP 또는 UDP(또는 계층 4) 트래픽을 라우팅합니다. 방화벽, 침입 감지 및 방지 시스템, 심층 패킷 검사 시스템과 같은 가상 어플라이언스를 사용하세요.	

자세한 내용은 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산 섹션을 참조하세요.

상호 연결 서비스

애플리케이션이 Amazon ECS 서비스로 실행되는 다른 애플리케이션에 연결해야 하는 경우 Amazon ECS에서는 다음과 같이 로드 밸런서 없이 수행하는 방법을 제공합니다.

- Service Connect - 짧은 이름과 표준 포트를 사용해 자동 검색을 통한 서비스 간 통신을 허용합니다.
- 서비스 검색 - 서비스 검색은 Route 53을 사용하여 서비스에 대한 네임스페이스를 생성합니다. 이를 통해 DNS를 사용하여 검색할 수 있습니다.
- Amazon VPC Lattice - VPC Lattice는 여러 계정과 VPC에서 서비스를 연결, 보호 및 모니터링하는 완전관리형 애플리케이션 네트워킹 서비스입니다. 이와 관련된 비용이 있습니다.

자세한 내용은 Amazon ECS 서비스 상호 연결 섹션을 참조하세요.

Amazon ECS 서비스 배포 컨트롤러 및 전략

서비스를 배포하기 전에 서비스를 배포하기 위한 옵션과 서비스가 사용하는 기능을 결정합니다.

일정 전략

사용 가능한 서비스 스케줄러 전략으로 다음 두 가지가 있습니다.

- REPLICA—복제본 일정 전략은 클러스터에 원하는 작업 수를 배치하고 유지합니다. 기본적으로 서비스 스케줄러는 가용 영역에 태스크를 분산합니다. 작업 배치 전략과 제약을 사용하여 작업 배치 결정을 사용자 지정할 수 있습니다. 자세한 내용은 복제본 예약 전략 섹션을 참조하세요.
- DAEMON—데몬 일정 전략은 사용자가 클러스터에 지정하는 작업 배치 제약을 모두 충족하는 각 활성 컨테이너 인스턴스에 한 작업씩 정확히 배포합니다. 이 전략을 사용하는 경우 원하는 태스크 수, 작업 배치 전략을 지정하거나 서비스 Auto Scaling 정책을 사용할 필요가 없습니다. 자세한 내용은 대몬 예약 전략 섹션을 참조하세요.

> **Note**
>
> Fargate 태스크는 DAEMON 일정 전략을 지원하지 않습니다.

복제본 예약 전략

복제본 일정 전략은 클러스터에 원하는 작업 수를 배치하고 유지합니다.

Fargate에서 작업을 실행하는 서비스의 경우, 서비스 스케줄러가 새 작업을 시작하거나 실행 중인 작업을 중지하면 서비스 스케줄러는 가용 영역 간의 밸런스를 유지하려고 최선을 다합니다. 작업 배치 전략이나 제약을 지정할 필요가 없습니다.

EC2 인스턴스에서 태스크를 실행하는 서비스를 생성할 때, 선택적으로 작업 배치 전략과 제약 조건을 지정하여 작업 배치 결정을 사용자 지정할 수 있습니다. 작업 배치 전략이나 제약 조건이 지정되지 않은 경우, 서비스 스케줄러는 기본적으로 가용 영역 간 태스크를 분산합니다. 서비스 스케줄러는 다음 로직을 사용합니다.

- 클러스터에서 어느 컨테이너 인스턴스가 서비스의 태스크 정의를 지원할 수 있는지 판단합니다(예: 필요한 CPU, 메모리, 포트, 컨테이너 인스턴스 속성).
- 서비스에 대해 정의된 배치 제약을 어느 컨테이너 인스턴스가 충족하는지 판단합니다.
- 대몬(daemon) 서비스에 의존하는 복제 서비스(예: 작업에서 로깅을 사용하려면 먼저 실행해야 하는 대몬(daemon) 로그 라우터 작업)가 있는 경우, 대몬(daemon) 서비스 작업이 복제 서비스 작업보다

먼저 EC2 인스턴스에 배치되도록 작업 배치 제약을 생성합니다. 자세한 내용은 Amazon ECS 작업 배치 제약 조건 예제 섹션을 참조하세요.

- 배치 전략이 정의되어 있으면 해당 전략을 사용하여 나머지 후보에서 인스턴스를 선택합니다.
- 배치 전략이 정의되어 있지 않으면 다음 로직을 사용하여 클러스터의 가용 영역 사이로 태스크를 분산합니다.
  - 유효한 컨테이너 인스턴스를 정렬합니다. 해당 가용 영역에서 이 서비스에 대해 실행되고 있는 태스크의 수가 가장 적은 인스턴스에 우선순위가 지정됩니다. 예를 들어 영역 A에는 실행 중인 서비스 작업이 1개이고, 영역 B 및 C에는 0개일 경우 영역 B 또는 C가 최적 배치로 간주됩니다.
  - 이전 단계에 따라 최적 가용 영역의 유효한 컨테이너 인스턴스에서 새 서비스 태스크를 배치합니다. 이 서비스에 대해 실행 중인 태스크 수가 가장 적은 컨테이너 인스턴스를 우선으로 합니다.

`REPLICA` 전략을 사용할 때는 서비스의 고가용성 확보에 도움이 되므로 서비스 리밸런싱 기능을 사용하는 것이 좋습니다.

대몬 예약 전략

대몬(Daemon) 일정 전략은 클러스터에 지정된 작업 배치 제약을 모두 충족하는 각 활성 컨테이너 인스턴스에 한 작업씩 정확히 배포합니다. 서비스 스케줄러는 실행 중인 태스크에 대한 작업 배치 제약을 평가하고 배치 제약을 충족하지 않는 태스크를 중지합니다. 이 전략을 사용하는 경우 원하는 태스크 수와 태스크 배치 전략을 지정하거나 서비스 Auto Scaling 정책을 사용할 필요가 없습니다.

Amazon ECS는 대몬 태스크를 위해 CPU, 메모리 및 네트워크 인터페이스를 포함한 컨테이너 인스턴스 컴퓨팅 리소스를 예약합니다. 다른 복제 서비스가 있는 클러스터에서 대몬 서비스를 시작하면 Amazon ECS가 대몬 태스크를 우선순위로 지정합니다. 즉, 대몬 태스크는 인스턴스에서 가장 먼저 실행되는 태스크이며 모든 복제본 태스크가 중지된 후 마지막으로 중지되는 태스크입니다. 이 전략은 보류 중인 복제 태스크에서 리소스를 사용하지 않고 대몬 작업에 사용할 수 있도록 합니다.

대몬 서비스 스케줄러는 `DRAINING` 상태인 인스턴스에는 어떤 태스크도 배치하지 않습니다. 컨테이너 인스턴스가 `DRAINING` 상태로 전환하면 이 컨테이너 인스턴스에 대한 대몬 태스크가 중지됩니다. 또한 클러스터에 새 컨테이너 인스턴스가 추가되는지 모니터링하고 새 컨테이너 인스턴스에 대몬 태스크를 추가합니다.

배포 구성을 지정할 때 `maximumPercent` 파라미터의 값은 설정하지 않은 경우 기본값으로 사용되는 `100`(백분율로 지정)이어야 합니다. `minimumHealthyPercent` 파라미터의 기본값은 `0`(비율로 지정)입니다.

대몬 서비스의 배치 제약을 변경할 때 서비스를 다시 시작해야 합니다. Amazon ECS는 대몬 태스크에 적합한 인스턴스에 예약된 리소스를 동적으로 업데이트합니다. 기존 인스턴스의 경우 스케줄러는 인스턴스에 태스크를 배치하려고 시도합니다.

작업 정의에서 작업 크기 또는 컨테이너 리소스 예약을 변경하면 새 배포가 시작됩니다. 서비스를 업데이트하거나 태스크 정의의 다른 개정을 설정할 때도 새 배포가 시작됩니다. Amazon ECS는 대몬에 대한 업데이트된 CPU 및 메모리 예약을 선택한 다음, 대몬 작업에 대한 해당 용량을 차단합니다.

위의 경우 중 하나에 대한 리소스가 부족하면 다음과 같은 상황이 발생합니다.

- 작업 배치가 실패합니다.
- CloudWatch 이벤트가 생성됩니다.
- Amazon ECS는 리소스를 사용할 수 있을 때까지 기다리면서 인스턴스에 대한 태스크를 계속 시도하고 예약합니다.
- Amazon ECS는 더 이상 배치 제약 기준을 충족하지 않는 예약 인스턴스를 확보하고 해당 대몬 태스크를 중지합니다.

대몬 일정 전략은 다음과 같은 경우에 사용할 수 있습니다.

- 애플리케이션 컨테이너 실행
- 로깅, 모니터링 및 추적 태스크를 위한 지원 컨테이너 실행

Fargate 시작 유형이나 CODE_DEPLOY 또는 EXTERNAL 배포 컨트롤러 유형을 사용하는 태스크는 대몬 일정 전략을 지원하지 않습니다.

서비스 스케줄러는 실행 태스크를 중지할 때 클러스터의 가용 영역 간에 밸런싱을 유지하려고 합니다. 스케줄러는 다음 로직을 사용합니다.

- 배치 전략이 정의된 경우 해당 전략을 사용하여 종료할 태스크를 선택합니다. 예를 들어 서비스에 가용 영역 분산 전략이 정의되어 있으면, 나머지 태스크를 최적 분산 상태로 만드는 태스크가 선택됩니다.
- 정의된 배치 전략이 없으면 다음 로직을 사용하여 클러스터 내 가용 영역 간 밸런스를 유지 관리합니다.
  - 유효한 컨테이너 인스턴스를 정렬합니다. 해당 가용 영역에서 이 서비스에 대해 실행되고 있는 태스크의 수가 가장 많은 인스턴스에 우선순위가 지정됩니다. 예를 들어 영역 A에는 실행 중인 서비스 태스크가 1개이고 영역 B 및 C에는 2개일 경우, 영역 B 또는 C의 컨테이너 인스턴스가 종료에 최적인 것으로 간주됩니다.

- 이전 단계에 따라 최적 가용 영역의 컨테이너 인스턴스에서 태스크를 중지합니다. 이 서비스에 대해 실행 중인 태스크 수가 가장 많은 컨테이너 인스턴스를 우선으로 합니다.

배포 컨트롤러

배포 컨트롤러는 서비스에 대해 태스크 배포 방법을 결정하는 메커니즘입니다. 유효한 옵션은 다음과 같습니다.

- ECS

  ECS 배포 컨트롤러를 사용하는 서비스를 생성하는 경우 다음 배포 전략 중에서 선택할 수 있습니다.

  - ROLLING: 롤링 업데이트(ROLLING) 배포 유형을 사용하는 서비스를 생성하는 경우 Amazon ECS 서비스 스케줄러가 현재 실행 중인 태스크를 새 태스크로 대체합니다. 롤링 업데이트 중에 Amazon ECS가 서비스에 추가하거나 서비스에서 제거하는 작업의 수는 서비스 배포 구성으로 제어합니다.

    롤링 업데이트 배포는 다음 시나리오에 가장 적합합니다.

    - 점진적 서비스 업데이트: 전체 서비스를 한 번에 오프라인으로 전환하지 않고 서비스를 점진적으로 업데이트해야 합니다.
    - 제한된 리소스 요구 사항: 블루/그린 배포에서 필요한 대로 두 개의 전체 환경을 동시에 실행하는 데 드는 추가 리소스 비용을 피하려고 합니다.
    - 허용되는 배포 시간: 롤링 업데이트는 태스크를 하나씩 대체하므로 애플리케이션이 더 긴 배포 프로세스를 허용할 수 있습니다.
    - 즉각적인 롤백 필요 없음: 서비스가 몇 초가 아닌 몇 분 정도 걸리는 롤백 프로세스를 허용할 수 있습니다.
    - 간단한 배포 프로세스: 여러 환경, 대상 그룹 및 리스너를 관리하는 복잡성 없이 간단한 배포 접근 방식을 선호합니다.
    - 로드 밸런서 요구 사항 없음: 서비스에서 로드 밸런서, Application Load Balancer, Network Load Balancer 또는 Service Connect(블루/그린 배포에 필요함)를 사용하거나 필요로 하지 않습니다.
    - 상태 저장 애플리케이션: 애플리케이션이 두 개의 병렬 환경을 실행하기 어려운 상태를 유지 관리합니다.
    - 비용 민감도: 배포 중에 중복 환경을 실행하지 않음으로써 배포 비용을 최소화하려고 합니다.

    롤링 업데이트는 서비스의 기본 배포 전략이며 많은 일반적인 애플리케이션 시나리오에서 배포 안전과 리소스 효율성 간의 균형을 제공합니다.

- BLUE_GREEN: 블루/그린 배포 전략(BLUE_GREEN)은 블루와 그린이라는 두 개의 동일한 프로덕션 환경을 실행하여 가동 중지 시간과 위험을 줄이는 릴리스 방법론입니다. Amazon ECS 블루/그린 배포를 사용하면 프로덕션 트래픽을 전달하기 전에 새 서비스 개정을 검증할 수 있습니다. 이 접근 방식은 필요한 경우 빠르게 롤백하는 기능을 통해 변경 내용을 배포하는 더 안전한 방법을 제공합니다.

  Amazon ECS 블루/그린 배포는 다음 시나리오에 가장 적합합니다.

  - 서비스 검증: 프로덕션 트래픽을 전달하기 전에 새 서비스 개정을 검증해야 하는 경우
  - 가동 중지 시간 없음: 서비스에 가동 중지 시간 없는 배포가 필요한 경우
  - 즉시 롤백: 문제가 감지되면 빠르게 롤백할 수 있는 기능이 필요한 경우
  - 로드 밸런서 요구 사항: 서비스가 Application Load Balancer, Network Load Balancer 또는 Service Connect를 사용하는 경우
- 외부

  타사 배포 컨트롤러를 사용합니다.
- 블루/그린 배포(AWS CodeDeploy 제공)

  CodeDeploy는 애플리케이션의 업데이트된 버전을 새 대체 태스크 세트로 설치하고 원래 애플리케이션 태스크 세트에서 대체 태스크 세트로 프로덕션 트래픽을 다시 라우팅합니다. 배포가 성공하면 기존 작업 세트는 종료됩니다. 이 배포 컨트롤러를 사용하여 프로덕션 트래픽을 전송하기 전에 서비스의 새로운 배포를 확인합니다.

Amazon ECS 배포 실패 감지

Amazon ECS는 배포 실패를 감지하는 두 가지 방법을 제공합니다.

- 배포 회로 차단기
- CloudWatch 경보

두 방법 모두 실패한 배포를 마지막으로 알려진 정상 상태로 자동으로 롤백하도록 구성할 수 있습니다.

다음을 고려하세요.

- 두 방법 모두 롤링 업데이트 배포 및 블루/그린 배포 유형만 지원합니다.
- 두 방법을 모두 사용하면 둘 중 하나가 배포 실패를 트리거할 수 있습니다.
- 롤백 메서드에는 완료 상태의 이전 배포가 필요합니다.

- 배포 상태 변경 시 EventBridge 이벤트가 생성됩니다.

Amazon ECS 배포 회로 차단기가 장애를 감지하는 방법

배포 회로 차단기는 작업이 안정 상태에 도달하는지 확인하는 롤링 업데이트 메커니즘입니다. 배포 회로 차단기에는 실패한 배포를 COMPLETED 상태의 배포로 자동 롤백하는 옵션이 있습니다.

서비스 배포의 상태가 변경되면 Amazon ECS는 서비스 배포 상태 변경 이벤트를 EventBridge로 전송합니다. 이를 통해 서비스 배포 상태를 프로그래밍 방식으로 모니터링할 수 있습니다. 자세한 내용은 Amazon ECS 서비스 배포 상태 변경 이벤트 섹션을 참조하세요. 수동 조치를 통해 배포를 시작할 수 있도록 `eventName`이 `SERVICE_DEPLOYMENT_FAILED`인 EventBridge 규칙을 생성하고 모니터링하는 것이 좋습니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 EventBridge 시작하기를 참조하세요.

배포 회로 차단기는 배포가 실패한 것으로 확인하면 COMPLETED 상태의 최신 배포를 찾습니다. 그리고 이 배포를 롤백 배포로 사용합니다. 롤백이 시작되면 배포가 COMPLETED에서 IN_PROGRESS로 변경됩니다. 즉, 배포가 COMPLETED 상태에 도달하기까지 다른 롤백을 사용할 수 없습니다. 배포 회로 차단기가 COMPLETED 상태의 배포를 찾지 못하면 회로 차단기는 새 작업을 시작하지 않으며 배포가 중단됩니다.

서비스를 생성하면 스케줄러는 시작하지 못한 작업을 2단계로 추적합니다.

- 1단계 - 스케줄러는 작업을 모니터링하여 작업이 실행 중 상태로 전환되는지 확인합니다.
  - 성공 - 실행 중 상태로 전환된 작업이 두 개 이상 있기 때문에 배포가 완료됨 상태로 전환될 가능성이 있습니다. 실패 기준을 건너뛰고 회로 차단기는 2단계로 진행합니다.
  - 실패 - 실행 중 상태로 전환되지 않은 작업이 연속적으로 발생하고 배포가 실패 상태로 전환될 수 있습니다.
- 2단계 - 실행 중 상태인 작업이 하나 이상 있을 때 배포가 이 단계로 설정됩니다. 회로 차단기는 평가 중인 현재 배포의 작업에 대해 상태 확인을 확인합니다. 검증된 상태 확인은 Elastic Load Balancing, AWS Cloud Map 서비스 상태 확인, 컨테이너 상태 확인입니다.
  - 성공 - 실행 중인 작업 중 상태 확인에 통과된 작업이 하나 이상 있습니다.
  - 실패 - 상태 확인 실패로 인해 교체된 작업이 실패 임계값에 도달했습니다.

서비스에 배포 회로 차단기를 사용할 때 다음 사항을 고려해야 합니다. EventBridge에서 규칙을 생성합니다.

- `DescribeServices` 응답은 배포 상태인 `rolloutState` 및 `rolloutStateReason`에 대한 인사이트를 제공합니다. 새 배포가 시작되면 롤아웃 상태는 `IN_PROGRESS` 상태에서 시작됩니다. 서비스가 정상 상태에 도달하면 롤아웃 상태가 `COMPLETED`로 전환됩니다. 서비스가 안정 상태에 도달하지 못하고 회로 차단기가 켜지면 배포가 `FAILED` 상태로 전환됩니다. `FAILED` 상태의 배포는 새로운 작업을 시작하지 않습니다.
- Amazon ECS는 시작되고 완료된 배포에 대해 서비스 배포 상태 변경 이벤트를 전송하는 것 외에도 회로 차단기가 켜진 배포가 실패할 경우 이벤트를 전송합니다. 이러한 이벤트는 배포가 실패한 이유 또는 롤백으로 인해 배포가 시작된 경우에 대한 세부 정보를 제공합니다. 자세한 내용은 Amazon ECS 서비스 배포 상태 변경 이벤트 섹션을 참조하세요.
- 이전 배포가 실패하고 롤백이 시작되었기 때문에 새 배포가 시작된 경우 서비스 배포의 `reason` 필드에 롤백으로 인해 배포가 시작되었음이 표시됩니다.
- 배포 회로 차단기는 롤링 업데이트(ECS) 배포 컨트롤러를 사용하는 Amazon ECS 서비스에만 지원됩니다.
- Amazon ECS 콘솔 또는 AWS CLI(CloudWatch 옵션과 함께 배포 회로 차단기를 사용할 때)를 사용해야 합니다. 자세한 내용은 AWS Command Line Interface 참조에서 the section called “정의된 파라미터를 사용하여 서비스 생성” 및 create-service를 참조하세요.

다음 `create-service` AWS CLI 예제에서는 배포 회로 차단기에 롤백 옵션이 사용된 경우 Linux 서비스를 생성하는 방법을 보여줍니다.

```
aws ecs create-service \
     --service-name MyService \
     --deployment-controller type=ECS \
     --desired-count 3 \
     --deployment-configuration "deploymentCircuitBreaker={enable=true,rollback=true}"
 \
     --task-definition sample-fargate:1 \
     --launch-type FARGATE \
     --platform-family LINUX \
     --platform-version 1.4.0 \
     --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=EN
```

예시:

배포 1은 `COMPLETED` 상태입니다.

배포 2를 시작할 수 없으므로 회로 차단기는 배포 1로 롤백합니다. 배포 1이 IN_PROGRESS 상태로 전환됩니다.

배포 3이 시작되고 COMPLETED 상태의 배포가 없으므로 배포 3은 롤백하거나 작업을 시작할 수 없습니다.

실패 임계값

배포 회로 차단기는 임계값을 계산한 다음 이 값을 사용하여 배포를 FAILED 상태로 이동할 시기를 결정합니다.

배포 회로 차단기의 최소 임계값은 3이고 최대 임계값은 200이며 다음 공식의 값을 사용하여 배포 실패를 확인합니다.

```
Minimum threshold <= 0.5 * desired task count => maximum threshold
```

계산 결과가 최솟값 3보다 크지만 최댓값인 200보다 작으면 실패 임계값은 계산된 임계값(반올림)으로 설정됩니다.

> **Note**
>
> 임계값은 변경할 수 없습니다.

배포 상태 검사에는 두 단계가 있습니다.

1. 배포 회로 차단기는 배포의 일부인 태스크를 모니터링하고 RUNNING 상태의 태스크를 확인합니다. 스케줄러는 현재 배포된 태스크가 RUNNING 상태일 때 실패 기준을 무시하고 다음 단계로 진행합니다. 태스크가 RUNNING 상태에 도달하지 못하면 배포 회로 차단기가 실패 횟수를 하나씩 늘립니다. 실패 횟수가 임계값과 같으면 배포가 FAILED로 표시됩니다.
2. RUNNING 상태의 작업이 하나 이상 있을 때 이 단계가 시작됩니다. 배포 회로 차단기는 현재 배포의 태스크에 대해 다음 리소스를 대상으로 상태 확인을 수행합니다.
   - Elastic Load Balancing 로드 밸런서
   - AWS Cloud Map 서비스
   - Amazon ECS 컨테이너 상태 확인

   태스크에 대한 상태 확인이 실패하면 배포 회로 차단기가 실패 횟수를 하나씩 늘립니다. 실패 횟수가 임계값과 같으면 배포가 FAILED로 표시됩니다.

다음 표에 몇 가지 예가 나와 있습니다.

원하는 작업 개수	계산	임계값
1	`3 <= 0.5 * 1 => 200`	3(계산된 값이 최솟값보다 작음)
25	`3 <= 0.5 * 25 => 200`	13(값이 반올림됨)
400	`3 <= 0.5 * 400 => 200`	200
800	`3 <= 0.5 * 800 => 200`	200(계산된 값이 최댓값보다 큼)

예를 들어, 임계값이 3인 경우 회로 차단기는 실패 횟수가 0으로 설정된 상태에서 시작됩니다. 작업이 RUNNING 상태에 도달하지 못하면 배포 회로 차단기가 실패 횟수를 하나씩 늘립니다. 실패 횟수가 3과 같아지면 배포가 FAILED로 표시됩니다.

롤백 옵션을 사용하는 방법에 대한 추가 예제는 Amazon ECS 배포 회로 차단기 발표를 참조하세요.

CloudWatch 경보가 Amazon ECS 배포 장애를 감지하는 방법

지정된 CloudWatch 경보가 ALARM 상태로 전환된 것을 탐지하면 배포를 실패로 설정하도록 Amazon ECS를 구성할 수 있습니다.

실패한 배포를 마지막으로 완료된 배포로 롤백하도록 구성을 설정할 수 있습니다.

다음 `create-service` AWS CLI 예제에서는 배포 경보에 롤백 옵션이 사용된 경우 Linux 서비스를 생성하는 방법을 보여줍니다.

```
aws ecs create-service \
     --service-name MyService \
     --deployment-controller type=ECS \
     --desired-count 3 \
     --deployment-configuration
 "alarms={alarmNames=[alarm1Name,alarm2Name],enable=true,rollback=true}" \
     --task-definition sample-fargate:1 \
     --launch-type FARGATE \
```

```
    --platform-family LINUX \
    --platform-version 1.4.0 \
    --network-configuration
"awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=EN
```

서비스에서 Amazon CloudWatch 경보 메서드를 사용하는 경우 다음을 고려하세요.

- 프로덕션 트래픽이 이동한 후 블루 및 그린 서비스 개정이 모두 동시에 실행되는 기간. Amazon ECS는 배포와 관련된 경보 구성을 기반으로 이 기간을 계산합니다. 이 값을 설정할 수 없습니다.
- 이제 `deploymentConfiguration` 요청 파라미터에 `alarms` 데이터 유형이 포함됩니다. 경보 이름, 메서드 사용 여부 및 경보가 배포 실패를 나타내는 경우 롤백을 시작할지 여부를 지정할 수 있습니다. 자세한 내용은 Amazon Elastic Container Service API 참조의 CreateService를 참조하세요.
- `DescribeServices` 응답은 배포 상태인 `rolloutState` 및 `rolloutStateReason`에 대한 인사이트를 제공합니다. 새 배포가 시작되면 롤아웃 상태는 `IN_PROGRESS` 상태에서 시작됩니다. 서비스가 안정 상태에 도달하고 베이크 소요 시간이 완료되면 롤아웃 상태가 `COMPLETED`로 전환됩니다. 서비스가 안정 상태에 도달하지 못하고 경보가 `ALARM` 상태로 전환되면 배포가 `FAILED` 상태로 전환됩니다. `FAILED` 상태의 배포는 새로운 태스크를 시작하지 않습니다.
- Amazon ECS는 시작되고 완료된 배포에 대해 서비스 배포 상태 변경 이벤트를 전송합니다. 또한 Amazon ECS는 해당 경보를 사용하는 배포가 실패할 경우 이벤트를 전송합니다. 이러한 이벤트는 배포가 실패한 이유 또는 롤백으로 인해 배포가 시작된 경우에 대한 세부 정보를 제공합니다. 자세한 내용은 Amazon ECS 서비스 배포 상태 변경 이벤트 섹션을 참조하세요.
- 이전 배포가 실패하고 롤백이 켜졌기 때문에 새 배포가 시작된 경우 서비스 배포 상태 변경 이벤트의 `reason` 필드에 롤백 때문에 배포가 시작되었음이 표시됩니다.
- 배포 회로 차단기와 Amazon CloudWatch 경보를 사용하여 실패를 탐지하는 경우 메서드 중 하나에 대한 기준이 충족되는 즉시 메서드 중 하나에서 배포 실패를 시작할 수 있습니다. 롤백은 배포 실패를 시작한 메서드에 대해 롤백 옵션을 사용할 때 발생합니다.
- Amazon CloudWatch 경보는 롤링 업데이트(ECS) 배포 컨트롤러를 사용하는 Amazon ECS 서비스에만 지원됩니다.
- Amazon ECS 콘솔 또는 AWS CLI를 사용하여 이 옵션을 구성할 수 있습니다. 자세한 내용은 AWS Command Line Interface 참조에서 the section called “정의된 파라미터를 사용하여 서비스 생성” 및 create-service를 참조하세요.
- 배포 상태가 장시간 동안 `IN_PROGRESS`로 유지되는 것을 알 수 있습니다. 그 이유는 활성 배포를 삭제할 때까지 Amazon ECS에서 상태가 변경되지 않기 때문이며 베이크 소요 시간이 지나야 이러한 상황이 발생하지 않습니다. 경보 구성에 따라 새 기본 작업 세트는 스케일 업되고 이전 배포는 스케일 다운되더라도 경보를 사용하지 않을 때보다 배포가 몇 분 더 오래 걸리는 것처럼 보일 수 있습니

다. CloudFormation 제한 시간을 사용하는 경우 제한 시간을 늘리는 것이 좋습니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 템플릿에 대기 조건 생성을 참조하세요.

- Amazon ECS는 `DescribeAlarms`를 호출하여 경보를 폴링합니다. `DescribeAlarms` 호출 수는 계정과 연결된 CloudWatch 서비스 할당량에 포함됩니다. `DescribeAlarms`를 호출하는 다른 AWS 서비스가 있는 경우 Amazon ECS에서 경보를 폴링하는 데 영향을 받을 수 있습니다. 예를 들어 다른 서비스가 할당량에 도달할 만큼 충분히 `DescribeAlarms`를 직접 호출하면 해당 서비스는 제한이 발생하고 Amazon ECS 역시 제한이 발생하여 경보를 폴링할 수 없습니다. 제한 기간에 경보가 생성되면 Amazon ECS에서 경보를 놓치고 롤백이 발생하지 않을 수 있습니다. 배포에 다른 영향을 미치지 않습니다. CloudWatch 서비스 할당량에 대한 자세한 내용은 CloudWatch 사용 설명서의 CloudWatch 서비스 할당량을 참조하세요.
- 배포 시작 시 경보가 `ALARM` 상태인 경우 Amazon ECS는 해당 배포 기간 동안 경보를 모니터링하지 않습니다(Amazon ECS는 경보 구성을 무시함). 이 동작은 초기 배포 실패를 수정하기 위해 새 배포를 시작하려는 경우를 다룹니다.

권장되는 경보

다음 경보 지표를 사용하는 것이 좋습니다.

- Application Load Balancer를 사용하는 경우 `HTTPCode_ELB_5XX_Count` 및 `HTTPCode_ELB_4XX_Count` Application Load Balancer 지표를 사용합니다. 이러한 지표는 HTTP 급증을 확인합니다. Application Load Balancer 지표에 대한 자세한 내용은 Application Load Balancer 사용 설명서의 Application Load Balancer의 CloudWatch 지표를 참조하세요.
- 기존 애플리케이션이 있는 경우 `CPUUtilization` 및 `MemoryUtilization` 지표를 사용합니다. 이러한 지표는 클러스터 또는 서비스가 사용하는 CPU 및 메모리의 비율을 확인합니다. 자세한 내용은 the section called “고려 사항” 섹션을 참조하세요.
- 작업에서 Amazon Simple Queue Service 대기열을 사용하는 경우 `ApproximateNumberOfMessagesNotVisible` Amazon SQS 지표를 사용합니다. 이 지표는 지연되어 즉시 읽을 수 없는 대기열의 메시지 수를 확인합니다. Amazon SQS 지표에 대한 자세한 내용은 Amazon Simple Queue Service 개발자 안내서의 Amazon SQS에 사용 가능한 CloudWatch 지표를 참조하세요.

Amazon ECS 서비스 배포에 대한 수명 주기 후크

배포가 시작되면 수명 주기 단계가 진행됩니다. 이러한 단계는 IN_PROGRESS 또는 성공과 같은 상태일 수 있습니다. Amazon ECS가 사용자를 대신해 지정된 수명 주기 단계에서 실행하는 Lambda 함수인 수명 주기 후크를 사용할 수 있습니다. 함수는 다음 중 하나일 수 있습니다.

- 15분 이내에 상태 확인을 검증하는 비동기 API.
- 수명 주기 후크 완료를 평가하는 다른 비동기 프로세스를 시작하는 폴링 API.

함수 실행이 완료된 후 배포를 계속하려면 hookStatus를 반환해야 합니다. hookStatus가 반환되지 않거나 함수에 실패하면 배포가 롤백됩니다. 다음은 hookStatus 값입니다.

- SUCCEEDED - 배포는 다음 수명 주기 단계로 계속 진행됨
- FAILED - 배포는 마지막으로 성공한 배포로 롤백됩니다.
- IN_PROGRESS - Amazon ECS는 짧은 시간 후에 함수를 다시 실행합니다. 기본적으로 30초 간격이지만 hookStatus와 함께 callBackDelaySeconds를 반환하여 이 값을 사용자 지정할 수 있습니다.

다음 예제에서는 사용자 지정 콜백 지연으로 hookStatus를 반환하는 방법을 보여줍니다. 이 예제에서 Amazon ECS는 기본 30초 대신 60초 후에 이 후크를 재시도합니다.

```
{
    "hookStatus": "IN_PROGRESS",
    "callBackDelaySeconds": 60
}
```

롤백이 수행되면 Amazon ECS는 다음 수명 주기 단계에 대해 수명 주기 후크를 실행합니다.

- PRODUCTION_TRAFFIC_SHIFT
- TEST_TRAFFIC_SHIFT

수명 주기 페이로드

ECS 서비스 배포에 대한 수명 주기 후크를 구성하면 Amazon ECS는 배포 프로세스의 특정 단계에서 이 후크를 간접 호출합니다. 각 수명 주기 단계는 배포의 현재 상태에 대한 정보를 포함하는 JSON 페이로드를 제공합니다. 이 문서에서는 각 수명 주기 단계의 페이로드 구조를 설명합니다.

공통 페이로드 구조

모든 수명 주기 단계 페이로드는 다음과 같은 공통 필드를 포함합니다.

- serviceArn - 서비스의 Amazon 리소스 이름(ARN).

- `targetServiceRevisionArn` - 배포 중인 대상 서비스 개정의 ARN.
- `testTrafficWeights` - 해당 테스트 트래픽 가중치 백분율에 대한 서비스 개정 ARN의 맵.
- `productionTrafficWeights` - 해당 프로덕션 트래픽 가중치 백분율에 대한 서비스 개정 ARN의 맵.

수명 주기 단계 페이로드

RECONCILE_SERVICE

이 단계는 서비스가 조정될 때 배포 프로세스 시작 시점에 나타납니다. 다음은 이 수명 주기 단계의 페이로드 예제를 보여줍니다.

```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/01275892":
 100,
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/78652123": 0
  },
  "productionTrafficWeights": {
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/01275892":
 100,
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/78652123": 0
  }
}
```

이 단계에서 기대 사항:

- 기본 태스크 세트 규모가 0%임

PRE_SCALE_UP

이 단계는 새 태스크가 스케일 업되기 전에 나타납니다. 다음은 이 수명 주기 단계의 페이로드 예제를 보여줍니다.

```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {},
  "productionTrafficWeights": {}
}
```

이 단계에서 기대 사항:

- 그린 서비스 개정 태스크 규모가 0%임

POST_SCALE_UP

이 단계는 새 태스크가 스케일 업되고 정상 상태가 된 후에 나타납니다. 다음은 이 수명 주기 단계의 페이로드 예제를 보여줍니다.

```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {},
  "productionTrafficWeights": {}
}
```

이 단계에서 기대 사항:

- 그린 서비스 개정 태스크 규모가 100%임
- 그린 서비스 개정의 태스크가 정상 상태임

TEST_TRAFFIC_SHIFT

이 단계는 테스트 트래픽이 그린 서비스 개정 태스크로 전환될 때 나타납니다.

다음은 이 수명 주기 단계의 페이로드 예제를 보여줍니다.

```
{
```

```
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/01275892":
 100,
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/78652123": 0
  },
  "productionTrafficWeights": {}
}
```

이 단계에서 기대 사항:

- 테스트 트래픽이 그린 서비스 개정 태스크로 이동하는 중입니다.

POST_TEST_TRAFFIC_SHIFT

이 단계는 테스트 트래픽이 새 태스크로 완전히 전환된 후에 나타납니다.

다음은 이 수명 주기 단계의 페이로드 예제를 보여줍니다.

```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {},
  "productionTrafficWeights": {}
}
```

이 단계에서 기대 사항:

- 테스트 트래픽의 100%가 그린 서비스 개정 태스크로 이동했습니다.

PRODUCTION_TRAFFIC_SHIFT

이 단계는 프로덕션 트래픽이 그린 서비스 개정 태스크로 전환될 때 나타납니다.

```
{
```

```
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {},
  "productionTrafficWeights": {
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/01275892":
 100,
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/78652123": 0
  }
}
```

이 단계에서 기대 사항:

- 프로덕션 트래픽이 그린 서비스 개정으로 이동하는 중입니다.

POST_PRODUCTION_TRAFFIC_SHIFT

이 단계는 프로덕션 트래픽이 그린 서비스 개정 태스크로 완전히 전환된 후에 나타납니다.

```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {},
  "productionTrafficWeights": {}
}
```

이 단계에서 기대 사항:

- 프로덕션 트래픽의 100%가 그린 서비스 개정 태스크로 이동했습니다.

수명 주기 단계 범주

수명 주기 단계는 다음과 같은 두 가지 범주로 구분됩니다.

1. 단일 간접 호출 단계 - 이 단계는 서비스 배포 중에 한 번만 간접 호출됩니다.
   - PRE_SCALE_UP
   - POST_SCALE_UP

- POST_TEST_TRAFFIC_SHIFT
- POST_PRODUCTION_TRAFFIC_SHIFT

2. 반복 간접 호출 단계 - 롤백 작업이 수행되는 경우와 같이 서비스 배포 중에 이 단계는 여러 번 간접 호출될 수 있습니다.
   - TEST_TRAFFIC_SHIFT
   - PRODUCTION_TRAFFIC_SHIFT

수명 주기 후크 중 배포 상태

수명 주기 후크가 실행되는 동안 배포 상태는 모든 수명 주기 단계에서 `IN_PROGRESS`입니다.

수명 주기 단계	배포 상태
RECONCILE_SERVICE	IN_PROGRESS
PRE_SCALE_UP	IN_PROGRESS
POST_SCALE_UP	IN_PROGRESS
TEST_TRAFFIC_SHIFT	IN_PROGRESS
POST_TEST_TRAFFIC_SHIFT	IN_PROGRESS
PRODUCTION_TRAFFIC_SHIFT	IN_PROGRESS
POST_PRODUCTION_TRAFFIC_SHIFT	IN_PROGRESS

Amazon ECS 서비스 배포 중지

회로 차단기 또는 CloudWatch 경보에서 실패한 배포를 감지하지 못한 경우 배포를 수동으로 중지할 수 있습니다. 다음과 같은 중지 유형을 사용할 수 있습니다.

- 롤백 - 이 옵션은 서비스 배포를 이전 서비스 개정으로 롤백합니다.

  서비스 배포에 롤백 옵션을 구성하지 않은 경우에도 이 옵션을 사용할 수 있습니다.

다음 상태 중 하나에 있는 배포를 중지할 수 있습니다. 서비스 배포 상태에 대한 자세한 내용은 Amazon ECS 서비스 배포를 사용하여 서비스 기록 보기 섹션을 참조하세요.

- PENDING - 서비스 배포가 ROLLBACK_REQUESTED 상태로 바뀐 다음 롤백 작업이 시작됩니다.
- IN_PROGRESS - 서비스 배포가 ROLLBACK_REQUESTED 상태로 바뀐 다음 롤백 작업이 시작됩니다.
- STOP_REQUESTED - 서비스 배포가 계속 중지됩니다.
- ROLLBACK_REQUESTED - 서비스 배포가 롤백 작업을 계속합니다.
- ROLLBACK_IN_PROGRESS - 서비스 배포가 롤백 작업을 계속합니다.

절차

시작하기 전에 서비스 배포를 보는 데 필요한 권한을 구성합니다. 자세한 내용은 Amazon ECS 서비스 배포를 보는 데 필요한 권한 섹션을 참조하세요.

Amazon ECS Console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
3. 클러스터 세부 정보 페이지의 서비스 단원에서 서비스를 선택합니다.

   서비스 세부 정보 페이지가 표시됩니다.
4. 배포 세부 정보 페이지에서 배포를 선택합니다.

   배포 페이지가 표시됩니다.
5. 진행 중인 배포에서 롤백을 선택합니다. 그런 다음 확인 창에서 롤백을 선택합니다.

AWS CLI

1. `list-service-deployments`를 실행하여 서비스 배포 ARN을 검색합니다.

   모든 *### ##*을 사용자의 값으로 바꿉니다.

   ```
   aws ecs list-service-deployments --cluster cluster-name --service service-name
   ```

   중지하려는 배포의 `serviceDeploymentArn`을 기록해 둡니다.

   ```
   {
       "serviceDeployments": [
           {
   ```

```
            "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-
deployment/cluster-name/service-name/NCWGC2ZR-taawPAYrIaU5",
            "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/cluster-
name/service-name",
            "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/cluster-
name",
            "targetServiceRevisionArn": "arn:aws:ecs:us-
west-2:123456789012:service-revision/cluster-name/service-
name/4980306466373577095",
            "status": "SUCCESSFUL"
        }
    ]
}
```

2. `stop-service-deployments`을(를) 실행합니다. `list-service-deployments`에서 반환된 `serviceDeploymentArn`을 사용합니다.

   모든 *### ##*을 사용자의 값으로 바꿉니다.

```
aws ecs stop-service-deployment --service-deployment-arn
 arn:aws:ecs:region:123456789012:service-deployment/cluster-name/service-
name/NCWGC2ZR-taawPAYrIaU5 --stop-type ROLLBACK
```

다음 단계

서비스에 어떤 변경이 필요한지 결정한 다음 서비스를 업데이트합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.

태스크를 대체하여 Amazon ECS 서비스 배포

롤링 업데이트(ECS) 배포 유형을 사용하는 서비스를 생성하면 Amazon ECS 서비스 스케줄러가 현재 실행 중인 태스크를 새 태스크로 대체합니다. 롤링 업데이트 중에 Amazon ECS가 서비스에 추가하거나 서비스에서 제거하는 작업의 수는 서비스 배포 구성으로 제어합니다.

Amazon ECS에서는 태스크 수 결정에 다음과 같은 파라미터가 사용됩니다.

- `minimumHealthyPercent`는 배포 중 또는 컨테이너 인스턴스가 드레이닝 시 서비스에 대해 실행되어야 하는 작업 수에 대한 하한을 서비스에 대해 원하는 작업 수의 백분율로 나타냅니다. 이 값은 반올림됩니다. 예를 들어, 최소 정상 상태 백분율이 50이고 원하는 작업 수가 4인 경우 스케줄러는 두 개의 새 태스크를 시작하기 전에 두 개의 기존 태스크를 중지할 수 있습니다. 마찬가지로 최소 정

상 상태 백분율이 75%이고 원하는 작업 수가 2이면 결과 값이 2이기 때문에 스케줄러는 태스크를 중지 할 수 없습니다.

작업이 비정상 상태가 되면 Amazon ECS 서비스 스케줄러가 먼저 교체 작업을 시작하고 교체 작업이 정상 상태가 될 때까지 `minimumHealthyPercent` 작업을 유지합니다. 교체 작업이 시작되고 정상 상태가 되면 비정상 작업은 점진적으로 중지됩니다.

- `maximumPercent`는 배포 중 또는 컨테이너 인스턴스가 드레이닝 시 서비스에 대해 실행되어야 하는 작업 수에 대한 상한을 서비스에 대해 원하는 작업 수의 백분율로 나타냅니다. 이 값은 내림됩니다. 예를 들어, 최대 백분율이 200이고 원하는 태스크 수가 4개인 경우 스케줄러는 4개의 기존 태스크를 중지하기 전에 4개의 새 태스크를 시작할 수 있습니다. 마찬가지로 최대 백분율이 125이고 원하는 작업 수가 3이면 결과 값이 3이기 때문에 스케줄러는 태스크를 시작할 수 없습니다.

> **Important**
>
> 최소 정상 상태 백분율 또는 최대 백분율을 설정할 때는 배포가 시작될 때 스케줄러가 하나 이상의 작업을 중지하거나 시작할 수 있는지 확인해야 합니다. 서비스에 잘못된 배포 구성으로 인해 중단된 배포가 있는 경우 서비스 이벤트 메시지가 전송됩니다. 자세한 내용은 서비스 배포 구성으로 인해 배포 중에 서비스(*service-name*)에서 태스크를 중지하거나 시작할 수 없습니다. minimumHealthyPercent 또는 maximumPercent 값을 업데이트하고 다시 시도하세요. 섹션을 참조하세요.

롤링 배포에는 서비스 배포에 실패한 시점을 빠르게 식별하는 방법이 제공되는 두 가지 방법이 있습니다.

- the section called “ 배포 회로 차단기가 장애를 감지하는 방법”
- the section called “CloudWatch 경보가 배포 장애를 감지하는 방법”

두 방법을 따로 또는 함께 사용할 수 있습니다. 두 방법을 모두 사용하는 경우 두 가지 실패 방법 중 하나에 대한 실패 기준이 충족되는 즉시 배포가 실패로 설정됩니다.

다음 가이드라인을 사용하면 사용할 방법을 결정하는 데 도움이 됩니다.

- 회로 차단기 - 작업을 시작할 수 없을 때 배포를 중지하려는 경우 이 방법을 사용합니다.
- CloudWatch 경보 - 애플리케이션 지표를 기반으로 배포를 중지하려는 경우 이 방법을 사용합니다.

두 가지 방법 모두 이전 서비스 개정으로 롤백이 지원됩니다.

컨테이너 이미지 확인

기본적으로 Amazon ECS에서는 태스크 정의에 지정된 컨테이너 이미지 태그를 컨테이너 이미지 다이제스트로 확인합니다. 단일 태스크가 실행되고 유지 관리되는 서비스를 생성하는 경우 태스크의 컨테이너에 대한 이미지 다이제스트 설정에 해당 태스크가 사용됩니다. 여러 태스크가 실행되고 유지 관리되는 서비스를 생성하는 경우 배포 중 서비스 스케줄러에 따라 시작된 첫 번째 태스크가 태스크의 컨테이너에 대한 이미지 다이제스트 설정에 사용됩니다.

컨테이너 이미지 다이제스트 설정 시도가 3회 이상 실패할 경우 이미지 다이제스트 확인 없이 배포가 계속됩니다. 배포 회로 차단기가 활성화된 경우 배포가 추가로 실패하고 롤백됩니다.

컨테이너 이미지 다이제스트가 설정되면 Amazon ECS에서는 이 다이제스트를 사용하여 필요한 모든 다른 작업을 시작하고 향후 서비스 업데이트를 진행합니다. 따라서 서비스의 모든 작업이 항상 동일한 컨테이너 이미지를 실행하게 되므로 소프트웨어의 버전 일관성이 유지됩니다.

컨테이너 정의의 `versionConsistency` 파라미터를 사용하여 태스크의 컨테이너마다 동작을 구성할 수 있습니다. 자세한 내용은 `versionConsistency` 섹션을 참조하세요.

> **Note**
>
> - 버전이 `1.31.0`보다 낮은 Amazon ECS 에이전트는 이미지 다이제스트 해상도를 지원하지 않습니다. `1.31.0~1.69.0` 버전의 에이전트는 Amazon ECR 리포지토리로 푸시된 이미지에 대해서만 이미지 다이제스트 해상도를 지원합니다. 버전이 `1.70.0` 이상인 에이전트는 모든 이미지에 대해 이미지 다이제스트 해상도를 지원합니다.
> - 이미지 다이제스트 해상도의 최소 Fargate Linux 플랫폼 버전은 `1.3.0`입니다. 이미지 다이제스트 해상도의 최소 Fargate Windows 플랫폼 버전은 `1.0.0`입니다.
> - Amazon ECS는 Amazon GuardDuty 보안 에이전트 또는 Service Connect 프록시와 같이 Amazon ECS에서 관리하는 사이드카 컨테이너의 다이제스트는 캡처하지 않습니다.
> - 여러 작업이 포함된 서비스에서 컨테이너 이미지 확인과 관련된 잠재적 지연 시간을 줄이려면 EC2 컨테이너 인스턴스에서 Amazon ECS 에이전트 버전 `1.83.0` 이상을 실행하세요. 잠재적 지연 시간을 방지하려면 태스크 정의에 컨테이너 이미지 다이제스트를 지정합니다.
> - 원하는 태스크 수가 0인 서비스를 생성하는 경우 원하는 태스크 수가 0보다 큰 서비스의 또 다른 배포를 트리거할 때까지 Amazon ECS에서 컨테이너 다이제스트를 설정할 수 없습니다.
> - 업데이트된 이미지 다이제스트를 설정하려면 새 배포를 강제로 적용할 수 있습니다. 업데이트된 다이제스트는 새 태스크를 시작하는 데 사용되며 이미 실행 중인 태스크에는 영향을 주

지 않습니다. 새 배포를 강제로 수행하는 방법에 대한 자세한 내용은 Amazon ECS API 참조의 ForceNewDeployment를 참조하세요.

- EC2 용량 공급자를 사용할 때 초기 배포 중에 태스크를 시작할 용량이 충분하지 않으면 소프트웨어 버전 일관성을 유지하지 못할 수 있습니다. 용량이 제한된 경우에도 버전 일관성을 유지 관리하려면 기본 동작에 의존하는 대신 태스크 정의 컨테이너 구성에서 명시적으로 `versionConsistency: true`를 설정합니다. 그러면 Amazon ECS는 배포를 진행하기 전에 용량을 사용할 수 있을 때까지 기다립니다.

Amazon ECS 서비스 파라미터의 모범 사례

애플리케이션 다운타임이 발생하지 않도록 배포 프로세스는 다음과 같습니다.

1. 기존 컨테이너를 계속 실행하면서 새 애플리케이션 컨테이너를 시작합니다.
2. 새 컨테이너가 정상 상태인지 확인합니다.
3. 이전 컨테이너를 중지합니다.

배포 구성과 클러스터의 예약되지 않은 여유 공간 크기에 따라 이전 작업을 모두 새 작업으로 교체하려면 이 작업을 여러 번 반복해야 할 수 있습니다.

숫자를 수정하는 데 사용할 수 있는 두 가지 서비스 구성 옵션이 있습니다.

- `minimumHealthyPercent`: 100%(기본값)

  배포 중 서비스가 `RUNNING` 상태를 유지해야 하는 작업 수의 하한. 가장 가까운 정수로 반올림된 `desiredCount`의 비율입니다. 이 파라미터를 통해 추가 클러스터 용량을 사용하지 않고 배포할 수 있습니다.
- `maximumPercent`: 200%(기본값)

  배포 중 `RUNNING` 또는 `PENDING` 상태를 유지해야 하는 서비스에 대한 작업 수의 상한. 가장 가까운 정수로 반내림된 `desiredCount`의 비율입니다.

예: 기본 구성 옵션

총 8개의 작업 공간이 있는 클러스터에 6개의 작업이 배포된 다음과 같은 서비스를 고려합니다. 기본 서비스 구성 옵션으로는 배포에서 원하는 6개 작업의 100% 미만으로 내려갈 수 없습니다.

배포 프로세스는 다음과 같습니다.

1. 목표는 6개 작업을 대체하는 것입니다.
2. 기본 설정에서는 실행 중인 작업이 6개여야 하기 때문에 스케줄러가 2개의 새 작업을 시작합니다.

   이제 기존 작업 6개와 새 작업 2개가 있습니다.
3. 스케줄러는 기존 작업 중 2개를 중지합니다.

   이제 기존 작업 4개와 새 작업 2개가 있습니다.
4. 스케줄러는 추가로 2개의 새 작업을 시작합니다.

   이제 기존 태스크 4개와 새 태스크 4개가 있습니다.
5. 스케줄러는 기존 작업 중 2개를 종료합니다.

   이제 기존 작업 2개와 새 작업 4개가 있습니다.
6. 스케줄러는 추가로 2개의 새 작업을 시작합니다.

   이제 기존 작업 2개와 새 작업 6개가 있습니다.
7. 스케줄러가 마지막 2개의 기존 작업을 종료합니다.

   이제 6개의 새 작업이 있습니다.

위 예제에서 옵션의 기본값을 사용하는 경우 새 작업이 시작될 때마다 2.5분의 대기 시간이 있습니다. 또한 로드 밸런서는 이전 작업이 중지될 때까지 5분을 기다려야 할 수도 있습니다.

예: **`minimumHealthyPercent`** 수정

`minimumHealthyPercent` 값을 50% 로 설정하여 배포 속도를 높일 수 있습니다.

총 8개의 작업 공간이 있는 클러스터에 6개의 작업이 배포된 다음과 같은 서비스를 고려합니다. 배포 프로세스는 다음과 같습니다.

1. 목표는 6개 작업을 대체하는 것입니다.
2. 스케줄러는 기존 작업 중 3개를 중지합니다.

   `minimumHealthyPercent` 값을 충족하는 3개의 기존 작업이 아직 실행 중입니다.
3. 스케줄러는 5개의 새 작업을 시작합니다.

   이제 기존 작업 3개와 새 작업 5개가 있습니다.
4. 스케줄러는 나머지 3개의 기존 작업을 중지합니다.

이제 5개의 새 작업이 있습니다.

5. 스케줄러는 마지막 새 작업을 시작합니다.

   이제 6개의 새 작업이 있습니다.

예: 클러스터 여유 공간 수정

추가 작업을 실행할 수 있도록 여유 공간을 더 추가할 수도 있습니다.

총 10개의 작업 공간이 있는 클러스터에 6개의 작업이 배포된 다음과 같은 서비스를 고려합니다. 배포 프로세스는 다음과 같습니다.

1. 목표는 기존 작업을 대체하는 것입니다.
2. 스케줄러는 기존 작업 중 3개를 중지합니다.

   이제 3개의 기존 작업이 있습니다.
3. 스케줄러는 6개의 새 작업을 시작합니다.

   이제 기존 작업 3개와 새 작업 6개가 있습니다.
4. 스케줄러는 기존 작업 3개를 중지합니다.

   이제 6개의 새 작업이 있습니다.

권장 사항

작업이 한동안 유휴 상태이고 사용률이 높지 않은 경우 서비스 구성 옵션에 다음 값을 사용합니다.

- `minimumHealthyPercent`: 50%
- `maximumPercent`: 200%

Amazon ECS 롤링 업데이트 배포 생성

클러스터에서 지정된 수의 태스크 정의 인스턴스를 동시에 실행하고 유지 관리하는 서비스를 생성합니다. 태스크가 실패하거나 중지되면 Amazon ECS 서비스 스케줄러가 태스크 정의의 다른 인스턴스를 시작하여 해당 태스크를 대체합니다. 이렇게 하면 서비스에서 원하는 수의 태스크를 유지 관리하는 데 도움이 됩니다.

서비스를 생성하기 전에 다음 구성 파라미터를 결정합니다.

- 태스크를 분산하는 두 가지 컴퓨팅 옵션이 있습니다.
  - 용량 공급자 전략은 Amazon ECS가 하나 또는 여러 개의 용량 공급자에 태스크를 분배하도록 합니다.
  - 시작 유형에서는 Amazon ECS가 클러스터에 등록된 Fargate 또는 EC2 인스턴스에 대한 태스크가 Amazon ECS에서 곧바로 시작됩니다.
- awsvpc 네트워크 모드 또는 로드 밸런서를 사용하도록 구성된 서비스를 사용하는 태스크 정의에 네트워킹 구성이 있어야 합니다. 기본적으로 콘솔은 기본 Amazon VPC와 모든 서브넷 및 기본 Amazon VPC 내의 기본 보안 그룹을 선택합니다.
- 배치 전략인 기본 태스크 배치 전략에서는 가용 영역 전체에 균등하게 태스크가 분산됩니다.

  서비스의 고가용성 확보에 도움이 되도록 가용 영역 리밸런싱을 사용하는 것이 좋습니다. 자세한 내용은 가용 영역 전체의 Amazon ECS 서비스 밸런싱 섹션을 참조하세요.
- 서비스 배포에 시작 유형을 사용하면 기본적으로 서비스가 클러스터 VPC에서 서브넷에서 시작됩니다.
- 용량 공급자 전략의 경우, 콘솔이 기본적으로 컴퓨팅 옵션을 선택합니다. 다음은 콘솔에서 기본값을 선택하는 순서에 대한 설명입니다.
  - 클러스터에 기본 용량 공급자 전략이 정의된 경우 이 전략이 선택됩니다.
  - 클러스터에 기본 용량 공급자 전략이 정의되어 있지 않지만 Fargate 용량 공급자가 클러스터에 추가되어 있는 경우 FARGATE 용량 공급자를 사용하는 사용자 지정 용량 공급자 전략이 선택됩니다.
  - 클러스터에 기본 용량 공급자 전략이 정의되어 있지 않지만 하나 이상의 Auto Scaling 그룹 용량 공급자가 클러스터에 추가된 경우 사용자 지정 사용(고급) 옵션이 선택되고 전략을 수동으로 정의해야 합니다.
  - 클러스터에 기본 용량 공급자 전략이 정의되어 있지 않고 클러스터에 용량 공급자가 추가되지 않은 경우 Fargate 시작 유형이 선택됩니다.
- 기본 배포 실패 감지 기본 옵션은 Amazon ECS 배포 회로 차단기 옵션과 실패 시 롤백 옵션을 함께 사용하는 것입니다.

  자세한 내용은 Amazon ECS 배포 회로 차단기가 장애를 감지하는 방법 섹션을 참조하세요.
- 블루/그린 배포 옵션을 사용하려면 CodeDeploy에서 애플리케이션을 이동하는 방식을 결정합니다. 다음과 같은 옵션을 사용할 수 있습니다.
  - CodeDeployDefault.ECSAllAtOnce: 모든 트래픽을 업데이트된 Amazon ECS 컨테이너로 한 번에 이동합니다.
  - CodeDeployDefault.ECSLinear10PercentEvery1Minutes: 모든 트래픽이 이동할 때까지 매분 트래픽의 10%를 이동합니다.

  - CodeDeployDefault.ECSLinear10PercentEvery3Minutes: 모든 트래픽이 이동할 때까지 3분마다 트래픽의 10%를 이동합니다.
  - CodeDeployDefault.ECSCanary10Percent5Minutes: 첫 번째 증분에서 트래픽의 10%를 이동합니다. 나머지 90%는 5분 이후 배포됩니다.
  - CodeDeployDefault.ECSCanary10Percent15Minutes: 첫 번째 증분에서 트래픽의 10%를 이동합니다. 나머지 90%는 15분 이후 배포됩니다.
- 서비스에서 원하는 태스크 수가 Amazon ECS에서 자동으로 증가하거나 감소하도록 할지를 결정합니다. 자세한 정보는 Amazon ECS 서비스 자동 조정 섹션을 참조하세요.
- 애플리케이션이 Amazon ECS에서 실행되는 다른 애플리케이션에 연결해야 하는 경우 아키텍처에 맞는 옵션을 결정합니다. 자세한 내용은 Amazon ECS 서비스 상호 연결 섹션을 참조하세요.
- Amazon ECS 회로 차단기가 사용되는 서비스를 생성할 때 Amazon ECS에서는 서비스 배포 및 서비스 개정이 생성됩니다. 이러한 리소스를 통해 서비스 기록에 대한 자세한 내용을 볼 수 있습니다. 자세한 내용은 Amazon ECS 서비스 배포를 사용하여 서비스 기록 보기 섹션을 참조하세요.

  AWS CLI를 사용하여 서비스를 생성하는 방법에 대한 자세한 내용은 AWS Command Line Interface 참조의 create-service를 참조하세요.

  AWS CloudFormation을 사용하여 서비스를 생성하는 방법에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS::ECS::Service를 참조하세요.

기본 옵션으로 서비스 생성

콘솔을 사용하여 서비스를 빠르게 생성하고 배포할 수 있습니다. 서비스에는 다음 구성이 있습니다.

- 클러스터와 연결된 VPC 및 서브넷에서 배포
- 하나의 태스크 배포
- 롤링 배포 사용
- 기본 용량 공급자로 기본 공급자 전략 사용
- 배포 회로 차단기를 사용하여 오류를 탐지하고 오류 발생 시 자동으로 롤백하는 옵션 설정

기본 파라미터를 사용하여 서비스를 배포하려면 다음의 단계를 따르세요.

서비스 생성 방법(Amazon ECS 콘솔)

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 페이지에서 Clusters(클러스터)를 선택합니다.

3. 클러스터 페이지에서 서비스를 생성할 클러스터를 선택합니다.
4. Services(서비스) 탭에서 Create(생성)를 선택합니다.

   서비스 생성 페이지가 표시됩니다.
5. 서비스 세부 정보에서 다음을 수행합니다.

   a. 작업 정의에 사용할 작업 정의 패밀리 및 개정을 입력합니다.

   b. 서비스 이름(Service name)에 서비스의 이름을 입력합니다.
6. 배포 구성에서 다음을 수행합니다.

   - 원하는 작업(Desired tasks)에 서비스에서 시작 및 유지 관리할 작업 수를 입력합니다.
7. (선택 사항) 서비스와 태스크를 식별하려면 태그(Tags) 섹션을 펼친 다음, 태그를 구성합니다.

   Amazon ECS가 새로 시작한 모든 작업에 클러스터 이름과 작업 정의 태그를 자동으로 지정하려면 Amazon ECS 관리형 태그(Turn on Amazon ECS managed tags) 사용을 선택한 다음 작업 정의(Task definitions)를 선택합니다.

   Amazon ECS가 새로 시작한 모든 작업에 클러스터 이름과 서비스 태그를 자동으로 지정하려면 Amazon ECS 관리형 태그(Turn on Amazon ECS managed tags) 사용을 선택한 다음 서비스(Service)를 선택합니다.

   태그를 추가하거나 제거합니다.

   - [태그 추가] 새로운 태그(Add tag)를 선택하고 다음을 수행합니다.
     - 키에서 키 이름을 입력합니다.
     - 값에 키 값을 입력합니다.
   - [태그 제거] 태그 옆에 있는 태그 제거를 선택합니다.

정의된 파라미터를 사용하여 서비스 생성

정의된 파라미터를 사용하여 서비스를 생성하려면 다음 단계를 수행합니다.

서비스 생성 방법(Amazon ECS 콘솔)

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 서비스를 시작할 리소스를 결정합니다.

서비스 시작	단계
클러스터	a. 클러스터 페이지에서 서비스를 생성할 클러스터를 선택합니다. b. Services(서비스) 탭에서 Create(생성)를 선택합니다.
태스크 정의	a. 작업 정의 페이지에서 작업 정의의 옆에 있는 옵션 버튼을 선택합니다. b. 배포 메뉴에서 서비스 생성을 선택합니다.

서비스 생성 페이지가 표시됩니다.

3. 서비스 세부 정보에서 다음을 수행합니다.

   a. 작업 정의에 사용할 작업 정의를 입력합니다. 그런 다음 개정에서 사용할 개정을 선택합니다.

   b. 서비스 이름(Service name)에 서비스의 이름을 입력합니다.

4. 기존 클러스터에 대해 클러스터를 선택합니다.

   클러스터 생성을 선택하여 새 클러스터에서 작업을 실행합니다.

5. 클러스터 인프라에 태스크를 배포하는 방식을 선택합니다. 컴퓨팅 구성에서 옵션을 선택합니다.

컴퓨팅 옵션	단계
용량 제공자 전략	a. 컴퓨팅 옵션에서 용량 공급자 전략을 선택합니다. b. 전략을 선택합니다. • 클러스터의 기본 용량 공급자 전략을 사용하려면 클러스터 기본값 사용

컴퓨팅 옵션	단계
	(Use cluster default)을 선택합니다. • 클러스터에 기본 용량 공급자 전략이 없거나 사용자 지정 전략을 사용하려면 사용자 지정 사용, 용량 공급자 전략 추가를 선택하고 기본, 용량 공급자 및 가중치를 지정하여 사용자 지정 용량 공급자 전략을 정의합니다. **Note** 전략에서 용량 공급자를 사용하려면 용량 공급자가 클러스터와 연결되어야 합니다.

컴퓨팅 옵션	단계
시작 유형	a. 컴퓨팅 옵션(Compute options) 섹션에서 시작 유형(Launch type)을 선택합니다. b. 시작 유형(Launch type)에서 시작 유형을 선택합니다. c. (선택 사항) Fargate 시작 유형이 지정되면 플랫폼 버전(Platform version)에서 사용할 플랫폼 버전을 지정합니다. 플랫폼 버전을 지정하지 않으면 LATEST 플랫폼 버전이 사용됩니다.

6. 배포 구성에서 다음을 수행합니다.

   a. 서비스 유형(Service type)의 경우 서비스 예약 전략을 선택합니다.

      - 스케줄러가 모든 태스크 배치 제약을 준수하는 각 활성 컨테이너 인스턴스에서 정확히 하나의 태스크만 배포하게 하려면 대몬(Daemon)을 선택합니다.
      - 스케줄러가 클러스터에 원하는 수의 태스크를 배치하고 관리하도록 하려면 복제본(Replica)을 선택합니다.

   b. 복제본(Replica)을 선택하는 경우, 원하는 태스크(Desired tasks)에 서비스에서 시작하고 유지할 태스크 개수를 입력합니다.

   c. 복제본을 선택한 경우 Amazon ECS에서 가용 영역 전체의 태스크 분산이 모니터링되고 불균형이 있으면 재분산되도록 하려면 가용 영역 서비스 리밸런싱에서 가용 영역 서비스 리밸런싱을 선택합니다.

   d. 상태 확인 유예 기간에 작업이 처음 시작된 후 서비스 스케줄러가 비정상 Elastic Load Balancing, VPC Lattice, 컨테이너 상태 확인을 무시하는 시간(초)을 입력합니다. 상태 확인 유예 기간 값을 지정하지 않으면 기본값인 0이 사용됩니다.

   e. 서비스의 배포 유형을 결정합니다. 배포 옵션을 확장한 후 다음 파라미터를 지정합니다.

배포 유형	단계	
롤링 업데이트	a. 최소 실행 작업(Min running tasks)의 경우, 배포 시 RUNNING 상태를 유지해야 하는 서비스 내 작업 수에 대한 하한을 원하는 작업 수의 백분율(가장 가까운 정수로 올림)로 입력합니다. 자세한 내용은 배포 구성을 참조하세요. b. 최대 실행 작업(Max running tasks)의 경우, 배포 시 RUNNING 또는 PENDING 상태가 허용되는 서비스 내 작업 수에 대한 상한을 원하는 작업 수의 백분율(가장 가까운 정수로 내림)로 입력합니다.	
블루/그린 배포	a. 배포 구성에서 CodeDeploy가 배포 중 프로덕션 트래픽을 대체 트래픽 세트로 라우팅하는 방법을 선택합니다. b. CodeDeploy를 위한 서비스 역할에서 서비스가 권한 있는 AWS 서비스 서비스에 대한 API 요청을 수행하는 데 사용하는 IAM 역할을 선택합니다.	

f. Amazon ECS에서 배포 오류를 탐지 및 처리하는 방법을 구성하려면 배포 오류 탐지(Deployment failure detection)를 펼친 다음, 옵션을 선택합니다.

i. 작업을 시작할 수 없을 때 배포를 중지하려면 Use the Amazon ECS deployment circuit breaker(Amazon ECS 배포 회로 차단기 사용)를 선택합니다.

배포 회로 차단기가 배포를 실패한 상태로 설정했을 때 소프트웨어가 마지막으로 완료한 배포로 자동 롤백하도록 하려면 실패 시 롤백을 선택합니다.

ii. 애플리케이션 지표를 기반으로 배포를 중지하려면 CloudWatch 경보 사용을 선택합니다. 그런 다음, CloudWatch 경보 이름에서 경보를 선택합니다. 새 경보를 생성하려면 CloudWatch 콘솔을 이동합니다.

CloudWatch 경보가 배포를 실패한 상태로 설정할 때 소프트웨어가 마지막으로 완료한 배포 상태로 자동 롤백하도록 하려면 실패 시 롤백을 선택합니다.

7. 작업 정의에서 awsvpc 네트워크 모드를 사용하는 경우 사용자 지정 네트워크 구성 확장 네트워킹을 지정한 후에 다음을 수행할 수 있습니다.

a. VPC에서 사용할 VPC 선택합니다.

b. 서브넷(Subnets)에서 태스크를 배치할 때 작업 스케줄러가 고려할 하나 이상의 서브넷을 VPC에서 선택합니다.

c. 보안 그룹(Security group)의 경우 기존 보안 그룹을 선택하거나 새 보안 그룹을 생성할 수 있습니다. 기존 보안 그룹을 사용하려면 보안 그룹을 선택하고 다음 단계로 이동합니다. 새 보안 그룹을 만들려면 새 보안 그룹 생성(Create a new security group)을 선택합니다. 보안 그룹 이름 및 설명을 지정한 다음 보안 그룹에 대해 하나 이상의 인바운드 규칙을 추가해야 합니다.

d. 퍼블릭 IP(Public IP)에서 작업의 탄력적 네트워크 인터페이스(ENI)에 퍼블릭 IP 주소를 자동 할당할지를 선택합니다.

AWS Fargate 작업은 퍼블릭 서브넷에서 실행될 때 퍼블릭 IP 주소가 할당될 수 있으므로 인터넷에 연결되는 경로가 있습니다. 이 필드를 사용하여 EC2 작업에 퍼블릭 IP를 할당할 수 없습니다. 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션 및 Amazon ECS 작업을 위한 네트워크 인터페이스 할당을 참조하십시오.

8. (선택 사항) Service Connect를 사용하여 서비스를 상호 연결하려면 Service Connect를 확장한 후 다음을 지정하세요.

a. Service Connect 켜기를 선택합니다.

b. Service Connect configuration(Service Connect 구성)에서 클라이언트 모드를 지정합니다.

- 서비스에서 네임스페이스의 다른 서비스에만 연결하면 되는 네트워크 클라이언트 애플리케이션을 실행하는 경우 클라이언트 측만을 선택합니다.

- 서비스가 네트워크 또는 웹 서비스 애플리케이션을 실행하고 이 서비스에 대한 엔드포인트를 제공해야 하며, 네임스페이스의 다른 서비스에 연결해야 하는 경우 Client and server(클라이언트 및 서버)를 선택합니다.

c. 기본 클러스터 네임스페이스가 아닌 네임스페이스를 사용하려면 Namespace(네임스페이스)에서 서비스 네임스페이스를 선택합니다.

d. (선택 사항) 로그 구성을 지정합니다. 로그 수집 사용을 선택합니다. 기본 옵션은 CloudWatch Logs로 컨테이너 로그를 전송합니다. 다른 로그 드라이버 옵션은 AWS FireLens를 사용하여 구성됩니다. 자세한 정보는 Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송을 참조하세요.

다음은 각 컨테이너 로그 대상에 대한 자세한 설명입니다.

- Amazon CloudWatch - CloudWatch Logs로 컨테이너 로그를 전송하도록 작업을 구성합니다. 사용자를 대신하여 CloudWatch 로그 그룹을 생성하는 기본 로그 드라이버 옵션이 제공됩니다. 다른 로그 그룹 이름을 지정하려면 드라이버 옵션 값을 변경합니다.
- Amazon Data Firehose - Firehose로 컨테이너 로그를 전송하도록 작업을 구성합니다. Firehose 전송 스트림으로 로그를 전송하는 기본 로그 드라이버 옵션이 제공됩니다. 다른 전송 스트림 이름을 지정하려면 드라이버 옵션 값을 변경합니다.
- Amazon Kinesis Data Streams - Kinesis Data Streams로 컨테이너 로그를 전송하도록 작업을 구성합니다. Kinesis Data Streams 스트림으로 로그를 전송하는 기본 로그 드라이버 옵션이 제공됩니다. 다른 스트림 이름을 지정하려면 드라이버 옵션 값을 변경합니다.
- Amazon OpenSearch Service - OpenSearch Service 도메인으로 컨테이너 로그를 전송하도록 작업을 구성합니다. 로그 드라이버 옵션이 제공되어야 합니다.
- Amazon S3 - Amazon S3 버킷으로 컨테이너 로그를 전송하도록 작업을 구성합니다. 기본 로그 드라이버 옵션이 제공되지만 유효한 Amazon S3 버킷 이름을 지정해야 합니다.

9. (선택 사항) 서비스 검색을 사용하여 서비스를 상호 연결하려면 서비스 검색을 확장한 후 다음을 수행하세요.

a. 서비스 검색 사용을 선택합니다.

b. 새 네임스페이스를 사용하려면 네임스페이스 구성에서 새 네임스페이스 생성을 선택하고 네임스페이스 이름과 설명을 입력합니다. 기존 네임스페이스를 사용하려면 기존 네임스페이스 선택을 선택하고 사용할 네임스페이스를 선택합니다.

c. 서비스 이름 및 설명과 같은 서비스 검색 서비스 정보를 제공합니다.

d. Amazon ECS에서 정기적으로 컨테이너 수준 상태 확인을 수행하도록 하려면 Amazon ECS 작업 상태 전파 활성화를 선택합니다.

e. DNS 레코드 유형(DNS record type)에서 서비스에 대해 생성할 DNS 레코드 유형을 선택합니다. Amazon ECS 서비스 검색은 작업 정의에서 지정하는 네트워크 모드에 따라 A 및 SRV 레코드만 지원합니다. 이 레코드 유형의 값 형식에 대한 자세한 정보는 Amazon Route 53 개발자 안내서의 지원되는 DNS 레코드 유형을 참조하세요.

- 서비스 작업에서 지정한 작업 정의가 `bridge` 또는 `host` 네트워크 모드를 사용하는 경우에는 SRV 레코드 유형만 지원됩니다. 레코드와 연결된 컨테이너 이름 및 포트 조합을 선택합니다.
- 서비스 작업에서 지정한 작업 정의가 `awsvpc` 네트워크 모드를 사용하는 경우에는 A 또는 SRV 레코드 유형을 선택합니다. A를 선택하면 다음 단계로 건너뜁니다. SRV를 선택한 경우에는 서비스를 찾을 수 있는 포트나 레코드와 연결할 컨테이너 이름 및 포트 조합을 지정합니다.

TTL에 DNS 확인자 및 웹 브라우저에서 레코드 세트를 캐시하는 기간(초)을 입력합니다.

10. (선택 사항) VPC Lattice를 사용하여 서비스를 상호 연결하려면 VPC Lattice를 확장한 후 다음을 수행하세요.

a. VPC Lattice 켜기를 선택합니다.

b. 인프라 역할에서 인프라 역할을 선택합니다.

역할을 생성하지 않은 경우 인프라 역할 생성을 선택합니다.

c. 대상 그룹에서 대상 그룹 또는 그룹을 선택합니다. 대상 그룹을 하나 이상 선택해야 하며 최대 5개를 선택할 수 있습니다. 대상 그룹을 더 추가하려면 대상 그룹 추가를 선택합니다. 선택한 각 대상 그룹의 포트 이름, 프로토콜 및 포트를 선택합니다.

대상 그룹을 삭제하려면 제거를 선택합니다.

> **Note**
>
> - 기존 대상 그룹을 추가하려면 AWS CLI를 사용해야 합니다. AWS CLI를 사용하여 대상 그룹을 추가하는 방법에 대한 지침은 AWS Command Line Interface 참조의 register-targets를 참조하세요.
> - VPC Lattice 서비스에는 여러 대상 그룹이 있을 수 있지만, 각 대상 그룹은 하나의 서비스에만 추가할 수 있습니다.

d. 리스너 기본 작업 또는 VPC Lattice 콘솔의 기존 VPC Lattice 서비스의 규칙에 새 대상 그룹을 포함시켜 VPC Lattice 구성을 완료합니다. 자세한 내용은 VPC Lattice 서비스를 위한 리스너 규칙을 참조하세요.

11. (선택 사항) 서비스에 대한 로드 밸런서를 구성하려면 로드 밸런싱(Load balancing) 섹션을 확장합니다.

    로드 밸런서를 선택합니다.

이 로드 밸런서 사용	조치
Application Load Balancer	a. 로드 밸런서 유형(Load balancer type)에서 Application Load Balancer를 선택합니다. b. 새 로드 밸런서 생성(Create a new load balancer)을 선택하여 새 Application Load Balancer를 생성하거나 기존 로드 밸런서 사용(Use an existing load balancer)을 선택하여 기존 Application Load Balancer를 선택합니다. c. 로드 밸런스 이름(Load balancer name)의 고유 이름을 입력합니다. d. 부하를 분산할 컨테이너 선택(Choose container to load balance)에서 서비스를 호스팅할 컨테이너를 선택합니다. e. 리스너(Listener)에서 Application Load Balancer에 대해 연결 요청을 수신

이 로드 밸런서 사용	조치
	할 포트 및 프로토콜을 입력합니다. 기본적으로 로드 밸런서는 포트 80과 HTTP를 사용하도록 구성됩니다. f. 대상 그룹 이름(Target group name)에 Application Load Balancer에서 요청을 라우팅할 대상 그룹에 대한 이름과 프로토콜을 입력합니다. 기본적으로 대상 그룹은 태스크 정의에 정의된 첫 번째 컨테이너로 요청을 라우팅합니다. g. 등록 취소 지연에 로드 밸런서가 대상 상태를 UNUSED로 변경하는 데 걸리는 시간(초)을 입력합니다. 기본값은 300초입니다. h. 상태 확인 경로(Health check path)에서 Application Load Balancer가 Application Load Balancer와 컨테이너 간의 연결 상태를 확인하기 위해 주기적으로 요청을 보내야 하는 컨테이너 내에 존재하는 경로를 입력합니다. 기본값은 루트 디렉터리입니다(/).

이 로드 밸런서 사용	조치
Network Load Balancer	a. 로드 밸런서 유형(Load balancer type)에서 Network Load Balancer를 선택합니다. b. Load Balancer에서 기존 Network Load Balancer를 선택합니다. c. 부하를 분산할 컨테이너 선택(Choose container to load balance)에서 서비스를 호스팅할 컨테이너를 선택합니다. d. Target group name(대상 그룹 이름)에 Network Load Balancer에서 요청을 라우팅할 대상 그룹에 대한 이름과 프로토콜을 입력합니다. 기본적으로 대상 그룹은 태스크 정의에 정의된 첫 번째 컨테이너로 요청을 라우팅합니다. e. 등록 취소 지연에 로드 밸런서가 대상 상태를 UNUSED로 변경하는 데 걸리는 시간(초)을 입력합니다. 기본값은 300초입니다. f. 상태 확인 경로(Health check path)에서 Network Load Balancer가 Application Load Balancer와 컨테이너 간의 연결 상태를 확인하기 위해 주기적으로 요청을 전송해야 하는 컨테이

이 로드 밸런서 사용	조치	
	너 내에 존재하는 경로를 입력합니다. 기본값은 루트 디렉터리입니다(/).	

12. (선택 사항) 서비스 오토 스케일링을 구성하려면 서비스 오토 스케일링을 확장한 후에 다음 파라미터를 지정합니다.트래픽 흐름에서 과거 로드 데이터를 살펴보는 예측 오토 스케일링을 사용하려면 서비스를 생성한 다음 예측 오토 스케일링을 구성합니다. 자세한 내용은 과거 패턴을 사용하여 예측 규모 조정으로 Amazon ECS 서비스 규모 조정 섹션을 참조하세요.

    a. 서비스 Auto Scaling을 사용하려면 서비스 Auto Scaling(Service auto scaling)을 선택합니다.

    b. 작업의 최소 개수에 서비스 Auto Scaling에서 사용할 작업 수의 하한을 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.

    c. 작업의 최대 개수에 서비스 Auto Scaling에서 사용할 작업 수의 상한을 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.

    d. 정책 유형을 선택합니다. 규모 조정 정책 유형에서 다음 옵션 중 하나를 선택합니다.

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
대상 추적	a. 조정 정책 유형(Scaling policy type)에서 대상 추적(Target tracking)을 선택합니다. b. 정책 이름(Policy name)에 정책 이름을 입력합니다. c. ECS 서비스 지표에서 다음 지표 중 하나를 선택합니다. • ECSServiceAverageCPUUtilization - 서비스의 평균 CPU 사용률입니다.	

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
	• ECSServiceAverageMemoryUtilization - 서비스의 평균 메모리 사용률입니다. • ALBRequestCountPerTarget - Application Load Balancer 대상 그룹에서 대상별로 완료된 요청 수입니다. d. 대상 값(Target value)에 서비스가 선택된 지표에 유지하는 값을 입력합니다. e. 스케일 아웃 휴지 기간에 스케일 아웃 활동(작업 추가) 이후 다른 스케일 아웃 활동이 시작되기 전 경과해야 하는 시간(초)을 입력합니다. f. 스케일 인 휴지 기간에 스케일 인 활동(작업 제거) 이후 다른 스케일 인 활동이 시작되기 전 경과해야 하는 시간(초)을 입력합니다. g. 정책에서 스케일 인 활동을 수행하지 않도록 방지하려면 스케일 인 비활성화(Turn off scale-in)를 선택합니다. h. • (선택 사항) 늘어난 트래픽에 맞춰 조정 정책을 스	

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
	케일 아웃하려고 하지만 트래픽이 감소해도 스케일 인할 필요가 없게 하려는 경우 스케일 인 끄기를 선택합니다.	

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
단계 조정	a. 조정 정책 유형(Scaling policy type)에서 단계 조정(Step scaling)을 선택합니다. b. 정책 이름에 정책 이름을 입력합니다. c. [Alarm name]에 경보의 고유한 이름을 입력합니다. d. Amazon ECS 서비스 지표에서 경보에 사용할 서비스 지표를 선택합니다. e. 통계에서 경보 통계를 선택합니다. f. 기간에서 경보의 기간을 선택합니다. g. 경보 조건에서 선택한 지표를 정의된 임계값과 비교하는 방법을 선택합니다. h. 지표 비교 임계값 및 경보 시작을 위한 평가 기간에 경보에 사용되는 임계값과 임계값을 평가할 기간을 입력합니다. i. 조정 작업에서 다음을 수행합니다. • 작업에서 서비스에 대해 작업을 특정 원하는 수를 추가, 제거 또는 설정할지 여부를 선택합니다.	

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
	• 작업을 추가하거나 제거하려는 경우 값에 조정 작업이 시작될 때 추가 또는 제거할 작업 수(또는 기존 작업에 대한 비율)를 입력합니다. 원하는 수 설정을 선택한 경우 작업 수를 입력합니다. 유형에서 값이 정수인지, 아니면 기존의 원하는 수에 대한 백분율 값인지 선택합니다. • 하한 및 상한에 단계 조정의 하한과 상한을 입력합니다. 기본적으로 추가 정책의 하한은 경보 임계값이고 상한은 양(+)의 무한대입니다. 기본적으로 제거 조정의 상한은 경보 임계값이고 하한은 음(-)의 무한대입니다. • (선택 사항) 조정 옵션을 추가합니다. 새 스케일링 액션 추가를 선택하고 조정 작업 단계를 반복합니다. • 휴지 기간에 이전 조정 활동이 적용될 때까지 대기하는 시간(초)을 입력합니다. 추가 정책의 경우 이 시간은 스케일 아웃 활동 이후 조정	

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
	정책이 스케일 인 활동을 차단하고 한 번에 스케일 아웃할 수 있는 작업 수를 제한하는 시간입니다. 제거 정책의 경우 스케일 인 활동 이후 다른 스케일 인 활동을 시작하기 전까지 경과해야 하는 시간입니다.	

13. (선택 사항) 기본 외에 다른 태스크 배치 전략을 사용하는 경우 태스크 배치(Task Placement)를 펼치고 다음의 옵션 중에서 선택하세요.

    자세한 내용은 Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법 섹션을 참조하세요.

    - AZ Balanced Spread - 작업을 가용 영역과 가용 영역의 컨테이너 인스턴스에 분산합니다.
    - AZ Balanced BinPack - 작업을 가용 영역과 가용 메모리가 최소인 컨테이너 인스턴스에 분산합니다.
    - BinPack - 사용 가능한 CPU 또는 메모리 최소량에 따라 작업을 분산합니다.
    - One Task Per Host - 각 컨테이너 인스턴스에 서비스의 작업을 최대 1개 배치합니다.
    - Custom - 자체 작업 배치 전략을 정의합니다.

    사용자 지정(Custom)을 선택하는 경우, 태스크를 배치할 알고리즘과 태스크 배치 중에 고려할 규칙을 정의합니다.

    - 전략(Strategy) 아래에 있는 유형(Type)과 필드(Field)의 경우, 알고리즘과 알고리즘에 사용할 엔터티를 선택합니다.

      최대 5개의 전략을 입력할 수 있습니다.
    - 제약 아래에 있는 유형과 표현식에서 해당 제약에 사용할 규칙과 속성을 선택합니다.

      예를 들어 TC 인스턴스에 태스크를 배치하기 위한 제약을 설정할 경우, 식(Expression)에는 attribute:ecs.instance-type =~ t2.*를 입력합니다.

      초대 10개의 제약을 입력할 수 있습니다.

14. 작업에서 배포 시 구성과 호환되는 데이터 볼륨을 사용하는 경우 볼륨을 확장하여 볼륨을 구성할 수 있습니다.

    볼륨 이름과 볼륨 유형은 작업 정의 개정을 생성할 때 구성되며 서비스를 생성할 때는 변경할 수 없습니다. 볼륨 이름과 유형을 업데이트하려면 새 작업 정의 개정을 생성하고 새 개정을 사용해 서비스를 생성해야 합니다.

이 볼륨 유형을 구성하는 방법	조치	
Amazon EBS	a. EBS 볼륨 유형에서 작업에 연결할 EBS 볼륨 유형을 선택합니다. b. 크기(GiB)에 유효한 볼륨 크기 값을 기비바이트(GiB) 단위로 입력합니다. 최소 1GiB 및 최대 16,384GiB 볼륨 크기를 지정할 수 있습니다. 스냅샷 ID를 제공하지 않는 한, 이 값은 필수입니다. c. IOPS의 경우 볼륨이 제공해야 하는 최대 초당 입출력 작업(IOPS) 수를 입력합니다. 이 값은 `io1`, `io2`, `gp3` 볼륨 유형에서만 구성할 수 있습니다. d. 처리량(MiB/s)에 볼륨에서 제공해야 하는 처리량을 초당 메비바이트 단위(MiBps 또는 MiB/s)로 입력합니다. 이 값은 `gp3` 볼륨 유형에서만 구성 가능합니다. e. 스냅샷 ID에서 기존 Amazon EBS 볼륨 스냅샷을 선택하거나 스냅샷에서 볼륨을 생성하려는 경우 스냅샷의 ARN을 입력합니다. 스냅샷 ID를 선택하거나 입력하지 않고 비어 있는 새 볼륨을 생성할 수도 있습니다.	

이 볼륨 유형을 구성하는 방법	조치	
	f. 스냅샷 ID를 지정하는 경우 볼륨 초기화 속도(MiB/s)를 지정할 수 있습니다. 100~300MiB/s 사이의 값을 입력하면 볼륨 생성을 위해 스냅샷 ID를 사용하여 지정된 스냅샷에서 데이터가 로드되는 속도가 결정됩니다. g. 파일 시스템 유형에서 볼륨의 데이터 스토리지 및 검색에 사용할 파일 시스템 유형을 선택합니다. 운영 체제 기본값 또는 특정 파일 시스템 유형을 선택할 수 있습니다. Linux의 기본값은 XFS입니다. 스냅샷에서 생성된 볼륨의 경우 스냅샷을 생성할 때 볼륨에서 사용하던 것과 동일한 파일 시스템 유형을 지정해야 합니다. 파일 시스템 유형이 일치하지 않으면 작업이 시작되지 않습니다. h. 인프라 역할에서 Amazon ECS가 작업에 대한 Amazon EBS 볼륨을 관리할 수 있도록 하는 데 필요한 권한이 있는 IAM 역할을 선택합니다. `AmazonECSInfrastructureRolePolicyForVolumes`	

이 볼륨 유형을 구성하는 방법	조치	
	관리형 정책을 역할에 연결하거나 정책을 지침으로 사용하여 특정 요구 사항에 맞는 권한을 보유한 자체 정책을 생성하고 연결할 수 있습니다. 필요한 권한에 대한 자세한 내용은 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요. i. 암호화에서 Amazon EBS 암호화를 기본 설정으로 사용하려면 기본값을 선택합니다. 계정에 암호화 기본 제공이 구성된 경우 설정에 지정된 AWS Key Management Service(AWS KMS) 키로 볼륨이 암호화됩니다. 기본값을 선택하고 Amazon EBS 기본 암호화가 켜져 있지 않으면 볼륨이 암호화되지 않습니다. 사용자 지정을 선택하면 볼륨 암호화에 대해 원하는 AWS KMS key를 지정할 수 있습니다. 없음을 선택한 경우 암호화가 기본적으로 구성되지 않았거나 암호화된 스냅샷에서 볼륨을 생성하지 않으면 볼륨이 암호화되지 않습니다.	

이 볼륨 유형을 구성하는 방법	조치	
	j. 암호화에 사용자 지정을 선택한 경우 사용하려는 AWS KMS key를 지정해야 합니다. KMS 키에서 AWS KMS key를 선택하거나 키 ARN을 입력합니다. 대칭 고객 관리형 키를 사용하여 볼륨을 암호화하려는 경우 AWS KMS key 정책에 올바른 권한이 정의되었는지 확인합니다. 자세한 내용은 Data encryption for Amazon EBS volumes를 참조하세요. k. (선택 사항) 태그에서 작업 정의 또는 서비스의 태그를 전파하거나 자체 태그를 제공하여 Amazon EBS 볼륨에 태그를 추가할 수 있습니다. 작업 정의에서 태그를 전파하려면 태그 전파 시작에서 작업 정의를 선택합니다. 서비스에서 태그를 전파하려면 태그 전파 시작에서 서비스를 선택합니다. 전파 안 함을 선택하거나 값을 선택하지 않으면 태그가 전파되지 않습니다. 고유한 태그를 제공하려면 태그 추가를 선택하고 추가	

이 볼륨 유형을 구성하는 방법	조치	
	하는 각 태그의 키와 값을 제공합니다. Amazon EBS 볼륨 태그 지정에 대한 자세한 내용은 Amazon EBS 볼륨 태그 지정을 참조하세요.	

15. (선택 사항) 서비스와 태스크를 식별하려면 태그(Tags) 섹션을 펼친 다음, 태그를 구성합니다.

    Amazon ECS에서 새로 시작된 모든 작업에 클러스터 이름과 작업 정의 태그를 자동으로 지정하도록 하려면 Amazon ECS 관리형 태그 켜기를 선택한 다음 작업 전파 시작에서 작업 정의를 선택합니다.

    Amazon ECS에서 새로 시작된 모든 작업에 클러스터 이름과 서비스 태그를 자동으로 지정하도록 하려면 Amazon ECS 관리형 태그 켜기를 선택한 다음 작업 전파 시작에서 서비스를 선택합니다.

    태그를 추가하거나 제거합니다.

    - [태그 추가] 새로운 태그(Add tag)를 선택하고 다음을 수행합니다.
      - 키에서 키 이름을 입력합니다.
      - 값에 키 값을 입력합니다.
    - [태그 제거] 태그 옆에 있는 태그 제거를 선택합니다.

16. 생성(Create)을 선택합니다.

다음 단계

다음은 서비스를 생성한 후의 추가 작업입니다.

- 트래픽 흐름의 과거 로드 데이터를 살펴보는 예측 오토 스케일링을 구성합니다. 자세한 내용은 과거 패턴을 사용하여 예측 규모 조정으로 Amazon ECS 서비스 규모 조정 섹션을 참조하세요.
- 배포를 추적하고 Amazon ECS 회로 차단기에 사용되는 서비스에 대한 서비스 기록을 봅니다. 자세한 내용은 Amazon ECS 서비스 배포를 사용하여 서비스 기록 보기 섹션을 참조하세요.

Amazon ECS 블루/그린 배포

블루/그린 배포는 블루와 그린이라는 두 개의 동일한 프로덕션 환경을 실행하여 가동 중지 시간과 위험을 줄이는 릴리스 방법론입니다. Amazon ECS 블루/그린 배포를 사용하면 프로덕션 트래픽을 전달하기 전에 새 서비스 개정을 검증할 수 있습니다. 이 접근 방식은 필요한 경우 빠르게 롤백하는 기능을 통해 변경 내용을 배포하는 더 안전한 방법을 제공합니다.

이점

블루/그린 배포는 다음과 같은 이점을 제공합니다.

- 프로덕션을 전환하기 전에 프로덕션 트래픽에서 테스트를 수행하여 위험을 줄입니다. 프로덕션 트래픽을 전달하기 전에 테스트 트래픽으로 새 배포를 검증할 수 있습니다.
- 가동 중지 시간이 발생하지 않는 배포. 프로덕션 환경은 배포 프로세스 전반에 걸쳐 계속 사용할 수 있으며 이를 통해 지속적인 서비스 가용성을 보장합니다.
- 문제가 감지되면 간편한 롤백. 그린 배포에 문제가 발생하면 서비스 중단을 연장하지 않고도 블루 배포로 빠르게 되돌릴 수 있습니다.
- 제어된 테스트 환경. 그린 환경은 전체 배포 전에 실제 트래픽 패턴으로 새 기능을 테스트할 격리된 공간을 제공합니다.
- 예측 가능한 배포 프로세스. 수명 주기 단계가 정의된 구조화된 접근 방식을 사용하면 배포의 일관성과 신뢰성이 향상됩니다.
- 수명 주기 후크를 통한 자동화된 검증. 배포의 여러 단계에서 자동화된 테스트를 구현하여 기능을 확인할 수 있습니다.

용어

다음은 Amazon ECS 블루/그린 배포 용어입니다.

- 베이크 소요 시간 - 프로덕션 트래픽이 이동한 후 블루 및 그린 서비스 개정이 모두 동시에 실행되는 기간.
- 블루 배포 - 교체하려는 현재 프로덕션 서비스 개정.
- 그린 배포 - 배포하려는 새 서비스 개정.
- 수명 주기 단계 - '프로덕션 트래픽 전환 후'와 같은 배포 작업에 존재하는 일련의 이벤트.
- 수명 주기 후크 - 특정 수명 주기 단계에서 배포를 확인하는 Lambda 함수.
- 리스너 - 사용자가 구성한 프로토콜 및 포트를 사용하여 연결 요청을 확인하는 Elastic Load Balancing 리소스. 리스너에 대해 사용자가 정의한 규칙에 따라 Amazon ECS가 등록된 대상으로 요청을 라우팅하는 방법이 결정됩니다.

- 규칙 - 리스너와 연결된 Elastic Load Balancing 리소스. 규칙은 요청이 라우팅되는 방식을 정의하며, 작업, 조건, 우선순위로 구성됩니다.
- 대상 그룹 - 요청을 하나 이상의 등록된 대상(예: EC2 인스턴스)으로 라우팅하는 데 사용되는 Elastic Load Balancing 리소스. 리스너를 생성할 때 기본 작업에 대한 대상 그룹을 지정합니다. 트래픽은 리스너 규칙에 지정된 대상 그룹으로 전달됩니다.
- 트래픽 전환 - Amazon ECS가 블루 배포에서 그린 배포로 트래픽을 전환하는 데 사용하는 프로세스. Amazon ECS 블루/그린 배포의 경우 모든 트래픽이 블루 서비스에서 그린 서비스로 한 번에 전환됩니다.

고려 사항

배포 유형을 선택할 때 다음을 고려하세요.

- 리소스 사용량: 블루/그린 배포는 블루 및 그린 서비스 개정을 동시에 일시적으로 실행합니다. 이 경우 배포 중에 리소스 사용량이 두 배가 될 수 있습니다.
- 배포 모니터링: 블루/그린 배포는 보다 자세한 배포 상태 정보를 제공하므로 이를 통해 배포 프로세스의 각 단계를 모니터링할 수 있습니다.
- 롤백: 블루/그린 배포를 사용하면 베이크 소요 시간이 만료될 때까지 블루 개정이 계속 실행되므로 문제가 감지되면 이전 버전으로 쉽게 롤백할 수 있습니다.

Amazon ECS 블루/그린 서비스 배포 워크플로

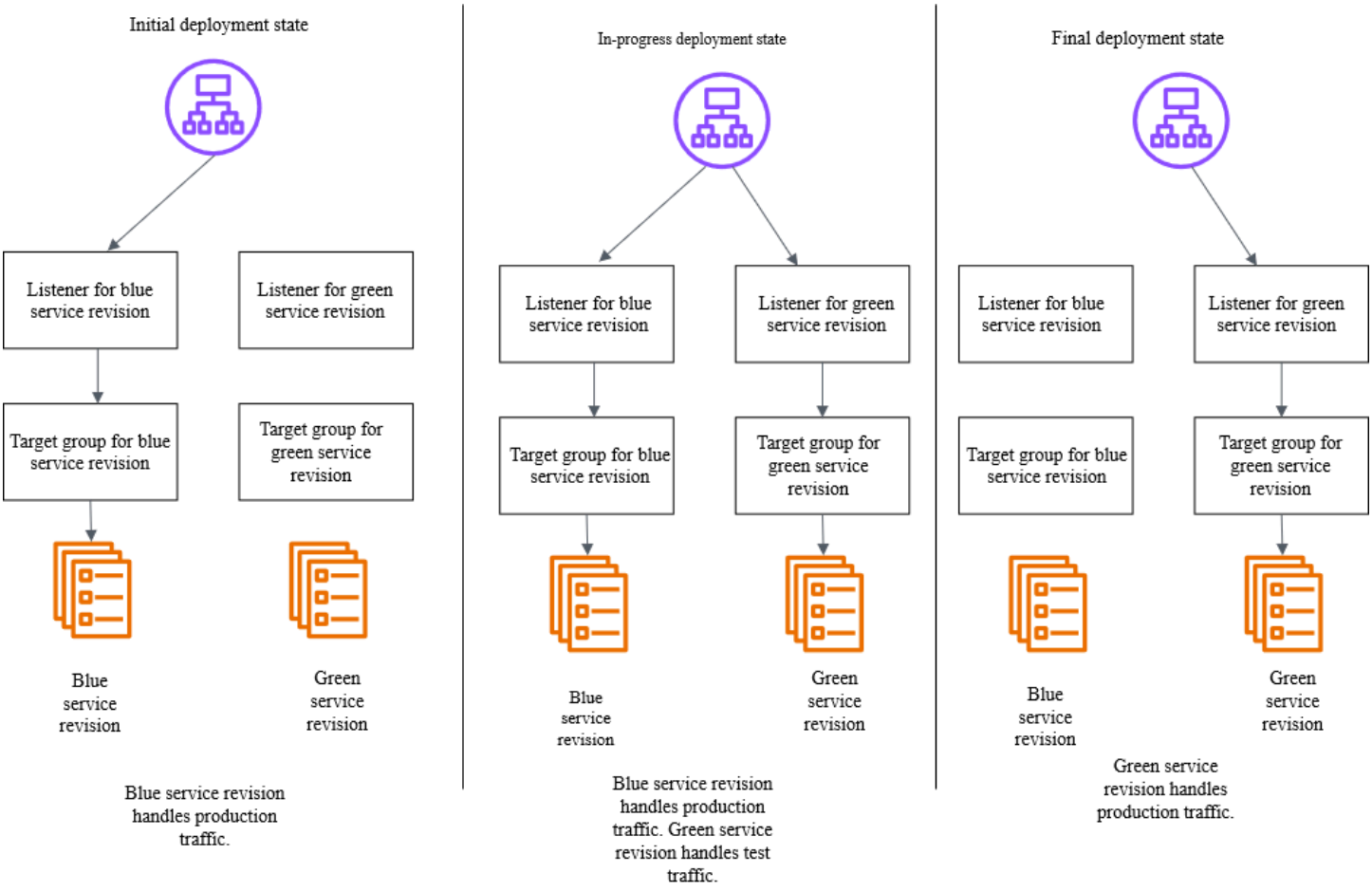
Amazon ECS 블루/그린 배포 프로세스는 안전하고 신뢰할 수 있는 애플리케이션 업데이트를 보장하는 6단계로 구성된 구조화된 접근 방식을 따릅니다. 각 단계는 애플리케이션을 검증하고 현재 버전(파란색)에서 새 버전(녹색)으로 전환하는 구체적인 목적을 지원합니다.

1. 준비 단계: 기존 블루 환경과 함께 그린 환경을 생성합니다. 여기에는 새 서비스 개정 프로비저닝 및 대상 그룹 준비가 포함됩니다.
2. 배포 단계: 그린 환경에 새 서비스 개정을 배포합니다. 블루 환경이 프로덕션 트래픽을 계속 처리하는 동안 Amazon ECS는 업데이트된 서비스 개정을 사용하여 새 태스크를 시작합니다.
3. 테스트 단계: 테스트 트래픽 라우팅을 사용하여 그린 환경을 검증합니다. Application Load Balancer는 프로덕션 트래픽이 블루 환경에서 유지되는 동안 테스트 요청을 그린 환경으로 보냅니다.
4. 트래픽 전환 단계: 구성된 배포 전략에 따라 프로덕션 트래픽을 파란색에서 녹색으로 전환합니다. 이 단계에서는 모니터링 및 검증 체크포인트를 포함합니다.
5. 모니터링 단계: 베이크 소요 시간 동안 애플리케이션 상태, 성능 지표 및 경보 상태를 모니터링합니다. 문제가 감지되면 롤백 작업이 시작됩니다.

6. 완료 단계: 구성에 따라 블루 환경을 종료하거나 잠재적 롤백 시나리오에 맞게 유지 관리하여 배포를 완료합니다.

워크플로

다음 다이어그램은 Amazon ECS와 Application Load Balancer 간 상호 작용을 표시하는 포괄적인 블루/그린 배포 워크플로를 보여줍니다.



향상된 배포 워크플로는 다음과 같은 세부 단계를 포함합니다.

1. 초기 상태: 블루 서비스(현재 프로덕션)가 프로덕션 트래픽의 100%를 처리합니다. Application Load Balancer에는 정상 상태의 블루 태스크를 포함하는 블루 대상 그룹으로 모든 요청을 라우팅하는 규칙을 사용하는 단일 리스너가 있습니다.
2. 그린 환경 프로비저닝: Amazon ECS는 업데이트된 태스크 정의를 사용하여 새 태스크를 생성합니다. 이러한 태스크는 새 그린 대상 그룹에 등록되지만 처음에는 트래픽을 수신하지 않습니다.
3. 상태 확인 검증: Application Load Balancer는 그린 태스크에 대해 상태 확인을 수행합니다. 그린 태스크가 상태 확인을 통과한 경우에만 배포는 다음 단계로 진행됩니다.

4. 테스트 트래픽 라우팅: 구성된 경우 Application Load Balancer의 리스너 규칙은 프로덕션 트래픽이 블루 환경에서 유지되는 동안 검증을 위해 특정 트래픽 패턴(예: 테스트 헤더가 있는 요청)을 그린 환경으로 라우팅합니다. 이는 요청 속성에 따라 서로 다른 규칙을 사용하여 프로덕션 트래픽을 처리하는 동일한 리스너에 의해 제어됩니다.
5. 프로덕션 트래픽 전환: 배포 구성에 따라 트래픽이 블루에서 그린으로 전환됩니다. ECS 블루/그린 배포에서는 트래픽의 100%가 블루에서 그린 환경으로 이동합니다(즉시 한 번에 전환). Application Load Balancer는 가중치를 기반으로 블루 및 그린 대상 그룹 간 트래픽 분산을 제어하는 리스너 규칙을 사용하는 단일 리스너를 사용합니다.
6. 모니터링 및 검증: Amazon ECS는 트래픽 전환 전반에 걸쳐 CloudWatch 지표, 경보 상태 및 배포 상태를 모니터링합니다. 문제가 감지되면 자동 롤백 트리거가 활성화됩니다.
7. 베이크 소요 시간(기간): 프로덕션 트래픽이 전환된 후 블루 및 그린 서비스 개정이 동시에 실행되는 기간.
8. 블루 환경 종료: 트래픽 전환 및 검증에 성공하면 클러스터 리소스를 확보하기 위해 블루 환경이 종료되거나 빠른 롤백 기능을 위해 유지됩니다.
9. 최종 상태: 그린 환경은 트래픽의 100%를 처리하며 새로운 프로덕션 환경이 됩니다. 배포가 성공으로 표시됩니다.

배포 수명 주기 단계

블루/그린 배포 프로세스는 각각 특정 책임과 검증 체크포인트가 있는 고유한 수명 주기 단계('프로덕션 트래픽 전환 이후'와 같은 배포 작업에서 일련의 이벤트)를 거칩니다. 이러한 단계를 이해하면 배포 진행 상황을 모니터링하고 문제를 효과적으로 해결할 수 있습니다.

각 수명 주기 단계는 최대 24시간 동안 지속될 수 있습니다. 값을 24시간 표시 미만으로 유지하는 것이 좋습니다. 비동기 프로세스가 후크를 트리거하는 데 시간이 필요하기 때문입니다. 시스템 제한 시간이 초과되어 배포에 실패하고 하나의 단계가 24시간에 도달한 후에 롤백을 시작합니다. AWS CloudFormation 배포에는 추가 제한 시간이 있습니다. 24시간의 단계 제한은 여전히 유효하지만 AWS CloudFormation은 전체 배포에 대해 36시간의 제한을 적용합니다. AWS CloudFormation에서 배포에 실패하고 프로세스가 36시간 이내에 완료되지 않으면 롤백을 시작합니다.

수명 주기 단계	설명	수명 주기 후크에서 이 단계 사용 여부
RECONCILE_SERVICE	이 단계는 활성 상태인 서비스 개정이 2개 이상인 새 서비스 배포를 시작하는 경우에만 나타납니다.	예

수명 주기 단계	설명	수명 주기 후크에서 이 단계 사용 여부
PRE_SCALE_UP	그린 서비스 개정이 시작되지 않았습니다. 블루 서비스 개정이 프로덕션 트래픽의 100%를 처리하는 중입니다. 테스트 트래픽이 없습니다.	예
SCALE_UP	그린 서비스 개정이 100%까지 스케일 업되고 새 태스크를 시작하는 시간. 그린 서비스 개정이 현재 트래픽을 처리하지 않습니다.	아니요
POST_SCALE_UP	그린 서비스 개정이 시작되었습니다. 블루 서비스 개정이 프로덕션 트래픽의 100%를 처리하는 중입니다. 테스트 트래픽이 없습니다.	예
TEST_TRAFFIC_SHIFT	블루 및 그린 서비스 개정이 실행 중입니다. 블루 서비스 개정이 프로덕션 트래픽의 100%를 처리합니다. 그린 서비스 개정이 테스트 트래픽의 0%~100%를 마이그레이션하고 있습니다.	예
POST_TEST_TRAFFIC_SHIFT	테스트 트래픽 전환이 완료되었습니다. 그린 서비스 개정이 테스트 트래픽의 100%를 처리합니다.	예
PRODUCTION_TRAFFIC_SHIFT	프로덕션 트래픽이 그린 서비스 개정으로 전환되고 있습니다. 그린 서비스 개정이 프로덕션 트래픽의 0%~100%를 마이그레이션하고 있습니다.	예
POST_PRODUCTION_TRAFFIC_SHIFT	프로덕션 트래픽 전환이 완료되었습니다.	예
BAKE_TIME	블루 및 그린 서비스 개정이 동시에 실행 중인 기간.	아니요

수명 주기 단계	설명	수명 주기 후크에서 이 단계 사용 여부
CLEAN_UP	블루 서비스 개정이 실행 중인 태스크가 0개인 상태로 완전히 스케일 다운되었습니다. 이제 그린 서비스 개정은 이 단계 이후에 프로덕션 서비스 개정이 됩니다.	아니요

각 수명 주기 단계는 다음 단계로 진행하기 전에 통과해야 하는 기본 제공 검증 체크포인트를 포함합니다. 검증에 실패하면 배포를 자동으로 롤백하여 서비스 가용성과 신뢰성을 유지 관리할 수 있습니다.

Amazon ECS 블루/그린 배포에 필요한 리소스

관리형 트래픽 전환을 통해 블루/그린 배포를 사용하려면 서비스에서 다음 기능 중 하나를 사용해야 합니다.

- Elastic Load Balancing
- Service Connect

서비스 검색, Service Connect, VPC Lattice 또는 Elastic Load Balancing을 사용하지 않는 서비스도 블루/그린 배포를 사용할 수 있지만 관리형 트래픽 전환의 이점은 얻을 수 없습니다.

다음 목록은 Amazon ECS 블루/그린 배포를 위해 구성해야 하는 사항에 대한 개략적인 개요를 제공합니다.

- 서비스는 Application Load Balancer, Network Load Balancer 또는 Service Connect를 사용합니다. 적절한 리소스를 구성합니다.
  - Application Load Balancer - 자세한 내용은 블루/그린 배포에 대한 Application Load Balancer 리소스 섹션을 참조하세요.
  - Network Load Balancer - 자세한 내용은 Amazon ECS 블루/그린 배포에 대한 Network Load Balancer 리소스 섹션을 참조하세요.
  - Service Connect - 자세한 내용은 Amazon ECS 블루/그린 배포에 대한 Service Connect 리소스 섹션을 참조하세요.
- 서비스 배포 컨트롤러를 ECS로 설정합니다.
- 서비스 정의에서 blue/green으로 배포 전략을 구성합니다.
- 선택적으로 다음과 같은 추가 파라미터를 구성합니다.

- 새 배포를 위한 베이크 소요 시간
- 자동 롤백에 대한 CloudWatch 경보
- 테스트에 대한 배포 수명 주기 후크(지정된 배포 단계에서 실행되는 Lambda 함수임)

모범 사례

성공적인 Amazon ECS 블루/그린 배포를 위해 다음 모범 사례를 따릅니다.

- 애플리케이션 상태를 정확하게 반영하는 적절한 상태 확인을 구성합니다.
- 그린 배포를 충분히 테스트할 수 있는 베이크 소요 시간을 설정합니다.
- CloudWatch 경보를 구현하여 문제를 자동으로 감지하고 롤백을 트리거합니다.
- 수명 주기 후크를 사용하여 각 배포 단계에서 자동화된 테스트를 수행합니다.
- 애플리케이션이 동시에 실행되는 블루 환경과 그린 환경을 모두 처리할 수 있는지 확인합니다.
- 배포 중에 두 환경을 모두 처리할 수 있는 충분한 클러스터 용량을 계획합니다.
- 프로덕션 환경에서 롤백 프로시저를 구현하기 전에 테스트합니다.

블루/그린 배포에 대한 Application Load Balancer 리소스

Amazon ECS 블루/그린 배포에서 Application Load Balancer를 사용하려면 블루 및 그린 서비스 개정 간 트래픽 라우팅을 허용하는 특정 리소스를 구성해야 합니다.

제한 사항

Application Load Balancer를 사용한 블루/그린 배포에는 현재 다음과 같은 제한 사항이 적용됩니다.

- 전달되지 않는 작업으로 구성된 여러 작업 유형을 포함하는 Application Load Balancer 리스너를 사용하는 경우 블루/그린 배포는 지원되지 않습니다. Application Load Balancer 리스너 규칙에 전달 작업과 비전달 작업(예: 리디렉션, 고정 응답 또는 인증 작업)이 혼합되어 포함된 경우 배포에 실패합니다.

대상 그룹

Elastic Load Balancing을 사용하는 블루/그린 배포의 경우 두 개의 대상 그룹을 생성해야 합니다.

- 블루 서비스 개정의 기본 대상 그룹(현재 프로덕션 트래픽)
- 그린 서비스 개정(새 버전)의 대체 대상 그룹

두 대상 그룹 모두 다음 설정으로 구성해야 합니다.

- 대상 유형: IP(awsvpc 네트워크 모드를 사용하는 Fargate 또는 EC2의 경우)
- 프로토콜: HTTP(또는 애플리케이션이 사용하는 프로토콜)
- 포트: 애플리케이션이 수신 대기하는 포트(일반적으로 HTTP의 경우 80)
- VPC: Amazon ECS 태스크와 동일한 VPC
- 상태 확인 설정: 애플리케이션 상태를 올바르게 확인하도록 구성됨

블루/그린 배포 중에 Amazon ECS는 배포 단계에 따라 적절한 대상 그룹에 태스크를 자동으로 등록합니다.

Example Application Load Balancer 대상 그룹 생성

다음 CLI 명령은 블루/그린 배포에서 Application Load Balancer와 함께 사용할 두 개의 대상 그룹을 생성합니다.

```
aws elbv2 create-target-group \
    --name blue-target-group \
    --protocol HTTP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-path / \
    --health-check-protocol HTTP \
    --health-check-interval-seconds 30 \
    --health-check-timeout-seconds 5 \
    --healthy-threshold-count 2 \
    --unhealthy-threshold-count 2

aws elbv2 create-target-group \
    --name green-target-group \
    --protocol HTTP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-path / \
    --health-check-protocol HTTP \
    --health-check-interval-seconds 30 \
    --health-check-timeout-seconds 5 \
    --healthy-threshold-count 2 \
```

```
    --unhealthy-threshold-count 2
```

Application Load Balancer

다음 구성을 사용하여 Application Load Balancer를 생성해야 합니다.

- 스키마: 요구 사항에 따라 인터넷 연결 또는 내부
- IP 주소 유형: IPv4
- VPC: Amazon ECS 태스크와 동일한 VPC
- 서브넷: 서로 다른 가용 영역에서 두 개 이상의 서브넷
- 보안 그룹: 리스너 포트에서 트래픽을 허용하는 보안 그룹

Application Load Balancer에 연결된 보안 그룹에는 Amazon ECS 태스크에 연결된 보안 그룹으로 트래픽을 허용하는 아웃바운드 규칙이 있어야 합니다.

Example Application Load Balancer 생성

다음 CLI 명령은 블루/그린 배포에 사용할 Application Load Balancer를 생성합니다.

```
aws elbv2 create-load-balancer \
    --name my-application-load-balancer \
    --type application \
    --security-groups sg-abcd1234 \
    --subnets subnet-12345678 subnet-87654321
```

리스너 및 규칙

블루/그린 배포의 경우 Application Load Balancer에서 다음과 같이 리스너를 구성해야 합니다.

- 프로덕션 리스너: 프로덕션 트래픽 처리(일반적으로 포트 80 또는 443)
  - 처음에 기본 대상 그룹으로 트래픽 전달(블루 서비스 개정)
  - 배포 후 대체 대상 그룹으로 트래픽 전달(그린 서비스 개정)
- 테스트 리스너(선택 사항): 프로덕션 트래픽을 전환하기 전에 테스트 트래픽을 처리하여 그린 서비스 개정 검증
  - 다른 포트(예: 8080 또는 8443)에서 구성 가능
  - 테스트 중에 대체 대상 그룹(그린 서비스 개정)으로 트래픽 전달

블루/그린 배포 중에 Amazon ECS는 배포 단계에 따라 적절한 대상 그룹으로 트래픽을 라우팅하도록 리스너 규칙을 자동으로 업데이트합니다.

Example 프로덕션 리스너 생성

다음 CLI 명령은 기본(블루) 대상 그룹으로 트래픽을 전달하는 프로덕션 리스너를 포트 80에서 생성합니다.

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
app/my-application-load-balancer/abcdef123456 \
    --protocol HTTP \
    --port 80 \
    --default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
blue-target-group/abcdef123456
```

Example 테스트 리스너 생성

다음 CLI 명령은 대체(그린) 대상 그룹으로 트래픽을 전달하는 테스트 리스너를 포트 8080에서 생성합니다.

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
app/my-application-load-balancer/abcdef123456 \
    --protocol HTTP \
    --port 8080 \
    --default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
green-target-group/ghijkl789012
```

Example 경로 기반 라우팅에 대한 리스너 규칙 생성

다음 CLI 명령은 테스트를 위해 특정 경로에 대한 트래픽을 그린 대상 그룹으로 전달하는 규칙을 생성합니다.

```
aws elbv2 create-rule \
    --listener-arn arn:aws:elasticloadbalancing:region:123456789012:listener/app/my-
application-load-balancer/abcdef123456/ghijkl789012 \
    --priority 10 \
    --conditions Field=path-pattern,Values='/test/*' \
```

```
    --actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
green-target-group/ghijkl789012
```

Example 헤더 기반 라우팅에 대한 리스너 규칙 생성

다음 CLI 명령은 테스트를 위해 특정 헤더를 포함한 트래픽을 그린 대상 그룹으로 전달하는 규칙을 생성합니다.

```
aws elbv2 create-rule \
    --listener-arn arn:aws:elasticloadbalancing:region:123456789012:listener/app/my-
application-load-balancer/abcdef123456/ghijkl789012 \
    --priority 20 \
    --conditions Field=http-header,HttpHeaderConfig='{Name=X-
Environment,Values=[test]}' \
    --actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
green-target-group/ghijkl789012
```

서비스 구성

Amazon ECS에서 사용자를 대신해 클러스터의 로드 밸런서 리소스를 관리하도록 허용할 권한이 있어야 합니다. 자세한 내용은 로드 밸런서에 대한 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.

Elastic Load Balancing을 사용하여 블루/그린 배포에 대한 Amazon ECS 서비스를 생성하거나 업데이트하는 경우 다음 구성을 지정해야 합니다.

모든 *### ##*을 사용자의 값으로 바꿉니다.

이 구성의 주요 구성 요소는 다음과 같습니다.

- `targetGroupArn`: 기본 대상 그룹 ARN(블루 서비스 개정).
- `alternateTargetGroupArn`: 대체 대상 그룹 ARN(그린 서비스 개정).
- `productionListenerRule`: 프로덕션 트래픽에 대한 리스너 규칙 ARN.
- `roleArn`: Amazon ECS에서 Elastic Load Balancing 리소스를 관리하도록 허용하는 역할의 ARN.
- `strategy`: 블루/그린 배포를 활성화하려면 `BLUE_GREEN`으로 설정합니다.
- `bakeTimeInMinutes`: 프로덕션 트래픽이 이동한 후 블루 및 그린 서비스 개정이 모두 동시에 실행되는 기간.

- `TestListenerRule`: 테스트 트래픽에 대한 리스너 규칙 ARN. 이는 선택 가능한 파라미터입니다.

```
{
    "loadBalancers": [
        {
            "targetGroupArn":
 "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/primary-target-group/
abcdef123456",
            "containerName": "container-name",
            "containerPort": 80,
            "advancedConfiguration": {
                "alternateTargetGroupArn":
 "arn:aws:elasticloadbalancing:region:account-id:targetgroup/alternate-target-group/
ghijkl789012",
                "productionListenerRule": "arn:aws:elasticloadbalancing:region:account-
id:listener-rule/app/load-balancer-name/abcdef123456/listener/ghijkl789012/rule/
mnopqr345678",
                "roleArn": "arn:aws:iam::123456789012:role/ecs-elb-role"
            }
        }
    ],
    "deploymentConfiguration": {
        "strategy": "BLUE_GREEN",
        "maximumPercent": 200,
        "minimumHealthyPercent": 100,
        "bakeTimeInMinutes": 5
    }
}
```

배포 중 트래픽 흐름

Elastic Load Balancing을 사용하는 블루/그린 배포 중에 트래픽은 다음과 같이 시스템을 통해 흐릅니다.

1. 초기 상태: 모든 프로덕션 트래픽이 기본 대상 그룹(블루 서비스 개정)으로 라우팅됩니다.
2. 그린 서비스 개정 배포: Amazon ECS는 새 태스크를 배포하고 대체 대상 그룹에 등록합니다.
3. 트래픽 테스트: 테스트 리스너가 구성된 경우 테스트 트래픽이 대체 대상 그룹으로 라우팅되어 그린 서비스 개정을 검증합니다.
4. 프로덕션 트래픽 전환: Amazon ECS는 트래픽을 대체 대상 그룹(그린 서비스 개정)으로 라우팅하도록 프로덕션 리스너 규칙을 업데이트합니다.

5. 베이크 소요 시간: 프로덕션 트래픽이 이동한 후 블루 및 그린 서비스 개정이 모두 동시에 실행되는 기간.
6. 완료: 배포에 성공하면 블루 서비스 개정이 종료됩니다.

배포 중에 문제가 감지되면 Amazon ECS는 트래픽을 기본 대상 그룹(블루 서비스 개정)으로 다시 라우팅하여 자동으로 롤백할 수 있습니다.

Amazon ECS 블루/그린 배포에 대한 Network Load Balancer 리소스

Amazon ECS 블루/그린 배포에서 Network Load Balancer를 사용하려면 블루 및 그린 서비스 개정 간 트래픽 라우팅을 허용하는 특정 리소스를 구성해야 합니다. 이 섹션에서는 필수 구성 요소 및 해당 구성에 대해 설명합니다.

구성에 Network Load Balancer가 포함된 경우 Amazon ECS는 다음 수명 주기 단계에 10분 지연을 추가합니다.

- PRE_SCALE_UP
- TEST_TRAFFIC_SHIFT
- PRODUCTION_TRAFFIC_SHIFT

이 지연은 구성된 트래픽 가중치와 데이터 플레인의 실제 트래픽 라우팅 간 불일치를 일으킬 수 있는 Network Load Balancer 타이밍 문제를 고려합니다.

대상 그룹

Network Load Balancer를 사용하는 블루/그린 배포의 경우 두 개의 대상 그룹을 생성해야 합니다.

- 블루 서비스 개정의 기본 대상 그룹(현재 프로덕션 트래픽)
- 그린 서비스 개정(새 서비스 개정)의 대체 대상 그룹

두 대상 그룹 모두 다음 설정으로 구성해야 합니다.

- 대상 유형: `ip`(`awsvpc` 네트워크 모드를 사용하는 Fargate 또는 EC2의 경우)
- 프로토콜: `TCP`(또는 애플리케이션이 사용하는 프로토콜)
- 포트: 애플리케이션이 수신 대기하는 포트(일반적으로 HTTP의 경우 `80`)
- VPC: Amazon ECS 태스크와 동일한 VPC

- 상태 확인 설정: 애플리케이션 상태를 올바르게 확인하도록 구성됨

  TCP 상태 확인을 위해 Network Load Balancer는 대상과 TCP 연결을 설정합니다. 연결에 성공하면 대상이 정상 상태로 간주됩니다.

  HTTP/HTTPS 상태 확인의 경우 Network Load Balancer는 대상에 HTTP/HTTPS 요청을 보내고 응답을 확인합니다.

블루/그린 배포 중에 Amazon ECS는 배포 단계에 따라 적절한 대상 그룹에 태스크를 자동으로 등록합니다.

Example Network Load Balancer에 대한 대상 그룹 생성

다음 AWS CLI 명령은 블루/그린 배포에서 Network Load Balancer와 함께 사용할 두 개의 대상 그룹을 생성합니다.

```
aws elbv2 create-target-group \
    --name blue-target-group \
    --protocol TCP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-protocol TCP

aws elbv2 create-target-group \
    --name green-target-group \
    --protocol TCP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-protocol TCP
```

Network Load Balancer

다음 구성을 사용하여 Network Load Balancer를 생성해야 합니다.

- 스키마: 요구 사항에 따라 인터넷 연결 또는 내부
- IP 주소 유형: IPv4
- VPC: Amazon ECS 태스크와 동일한 VPC
- 서브넷: 서로 다른 가용 영역에서 두 개 이상의 서브넷

Application Load Balancer와 달리 Network Load Balancer는 전송 계층(계층 4)에서 작동하며 보안 그룹을 사용하지 않습니다. 대신 Amazon ECS 태스크와 연결된 보안 그룹이 리스너 포트에서 Network Load Balancer로부터 수신되는 트래픽을 허용하는지 확인해야 합니다.

Example Network Load Balancer 생성

다음 AWS CLI 명령은 블루/그린 배포에 사용할 Network Load Balancer를 생성합니다.

```
aws elbv2 create-load-balancer \
    --name my-network-load-balancer \
    --type network \
    --subnets subnet-12345678 subnet-87654321
```

블루/그린 배포 사용에 관한 고려 사항

블루/그린 배포에 대해 Network Load Balancer를 사용하는 경우 다음 사항을 고려하세요.

- 계층 4 작업: Network Load Balancer는 전송 계층(계층 4)에서 작동하며 애플리케이션 계층(계층 7) 콘텐츠를 검사하지 않습니다. 즉, 라우팅 결정에 대해 HTTP 헤더 또는 경로를 사용할 수 없습니다.
- 상태 확인: Network Load Balancer 상태 확인은 TCP, HTTP 또는 HTTPS 프로토콜로 제한됩니다. TCP 상태 확인의 경우 Network Load Balancer는 연결을 설정할 수 있는지만 확인합니다.
- 연결 보존: Network Load Balancer는 보안 및 로깅 목적으로 유용할 수 있는 클라이언트의 소스 IP 주소를 보존합니다.
- 고정 IP 주소: Network Load Balancer는 각 서브넷에 대한 고정 IP 주소를 제공합니다. 이는 화이트리스트에 추가할 때 또는 클라이언트가 고정 IP 주소에 연결해야 하는 경우에 유용할 수 있습니다.
- 테스트 트래픽: Network Load Balancer는 콘텐츠 기반 라우팅을 지원하지 않으므로 테스트 트래픽을 프로덕션 트래픽과 다른 포트로 전송해야 합니다.

리스너 및 규칙

Network Load Balancer를 사용하는 블루/그린 배포의 경우 다음과 같이 리스너를 구성해야 합니다.

- 프로덕션 리스너: 프로덕션 트래픽 처리(일반적으로 포트 80 또는 443)
  - 처음에 기본 대상 그룹으로 트래픽 전달(블루 서비스 개정)
  - 배포 후 대체 대상 그룹으로 트래픽 전달(그린 서비스 개정)
- 테스트 리스너(선택 사항): 프로덕션 트래픽을 전환하기 전에 테스트 트래픽을 처리하여 그린 서비스 개정 검증
  - 다른 포트(예: 8080 또는 8443)에서 구성 가능

- 테스트 중에 대체 대상 그룹(그린 서비스 개정)으로 트래픽 전달

Application Load Balancer와 달리 Network Load Balancer는 콘텐츠 기반 라우팅 규칙을 지원하지 않습니다. 대신 트래픽은 리스너 포트 및 프로토콜에 기반하여 라우팅됩니다.

다음 AWS CLI 명령은 Network Load Balancer에 대한 프로덕션 및 테스트 리스너를 생성합니다.

모든 *### ##*을 사용자의 값으로 바꿉니다.

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
net/my-network-lb/1234567890123456 \
    --protocol TCP \
    --port 80 \
    --default-actions Type=forward,
 TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/blue-
target-group/1234567890123456

aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
net/my-network-lb/1234567890123456 \
    --protocol TCP \
    --port 8080 \
    --default-actions Type=forward,
 TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/green-
target-group/1234567890123456
```

서비스 구성

Amazon ECS에서 사용자를 대신해 클러스터의 로드 밸런서 리소스를 관리하도록 허용할 권한이 있어야 합니다. 자세한 내용은 로드 밸런서에 대한 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.

Network Load Balancer를 사용하여 블루/그린 배포에 대한 Amazon ECS 서비스를 생성하거나 업데이트하는 경우 다음 구성을 지정해야 합니다.

모든 *### ##*을 사용자의 값으로 바꿉니다.

이 구성의 주요 구성 요소는 다음과 같습니다.

- `targetGroupArn`: 기본 대상 그룹 ARN(블루 서비스 개정)
- `alternateTargetGroupArn`: 대체 대상 그룹 ARN(그린 서비스 개정)
- `productionListenerArn`: 프로덕션 트래픽에 대한 리스너 ARN

- `testListenerArn`: (선택 사항) 테스트 트래픽에 대한 리스너 ARN
- `roleArn`: Amazon ECS에서 Network Load Balancer 리소스를 관리하도록 허용하는 역할의 ARN.
- `strategy`: 블루/그린 배포를 활성화하려면 `BLUE_GREEN`으로 설정
- `bakeTimeInMinutes`: 프로덕션 트래픽을 전환하기 전에 그린 서비스 개정을 배포한 후 대기하는 시간

```
{
    "loadBalancers": [
        {
            "targetGroupArn":
 "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/blue-target-
group/1234567890123456",
            "containerName": "container-name",
            "containerPort": 80,
            "advancedConfiguration": {
                "alternateTargetGroupArn":
 "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/green-target-
group/1234567890123456",
                "productionListenerArn":
 "arn:aws:elasticloadbalancing:region:123456789012:listener/net/my-network-
lb/1234567890123456/1234567890123456",
                "testListenerArn":
 "arn:aws:elasticloadbalancing:region:123456789012:listener/net/my-network-
lb/1234567890123456/2345678901234567",
                "roleArn": "arn:aws:iam::123456789012:role/ecs-nlb-role"
            }
        }
    ],
    "deploymentConfiguration": {
        "strategy": "BLUE_GREEN",
        "maximumPercent": 200,
        "minimumHealthyPercent": 100,
        "bakeTimeInMinutes": 5
    }
}
```

배포 중 트래픽 흐름

Network Load Balancer를 사용하는 블루/그린 배포 중에 트래픽은 다음과 같이 시스템을 통해 흐릅니다.

1. 초기 상태: 모든 프로덕션 트래픽이 기본 대상 그룹(블루 서비스 개정)으로 라우팅됩니다.
2. 그린 서비스 개정 배포: Amazon ECS는 새 태스크를 배포하고 대체 대상 그룹에 등록합니다.
3. 트래픽 테스트: 테스트 리스너가 구성된 경우 테스트 트래픽이 대체 대상 그룹으로 라우팅되어 그린 서비스 개정을 검증합니다.
4. 프로덕션 트래픽 전환: Amazon ECS는 트래픽을 대체 대상 그룹(그린 서비스 개정)으로 라우팅하도록 프로덕션 리스너를 업데이트합니다.
5. 베이크 소요 시간: 프로덕션 트래픽이 이동한 후 블루 및 그린 서비스 개정이 모두 동시에 실행되는 기간.
6. 완료: 배포에 성공하면 블루 서비스 개정이 종료됩니다.

배포 중에 문제가 감지되면 Amazon ECS는 트래픽을 기본 대상 그룹(블루 서비스 개정)으로 다시 라우팅하여 자동으로 롤백할 수 있습니다.

Amazon ECS 블루/그린 배포에 대한 Service Connect 리소스

블루/그린 배포와 함께 Service Connect를 사용하는 경우 블루 및 그린 서비스 개정 사이에서 트래픽을 적절히 라우팅할 수 있도록 특정 구성 요소를 구성해야 합니다. 이 섹션에서는 필수 구성 요소 및 해당 구성에 대해 설명합니다.

아키텍처 개요

Service Connect는 Amazon ECS 태스크에 자동으로 삽입되는 관리형 사이드카 프록시를 통해 서비스 검색 및 서비스 메시 기능을 모두 빌드합니다. 이러한 프록시는 라우팅 결정, 재시도 및 지표 수집을 처리하는 반면 AWS Cloud Map에서는 서비스 레지스트리 백엔드를 제공합니다. Service Connect가 활성화된 상태로 서비스를 배포하면 서비스가 자체적으로 AWS Cloud Map에 등록되고 클라이언트 서비스가 네임스페이스를 통해 이를 검색합니다.

표준 Service Connect 구현에서 클라이언트 서비스는 논리적 서비스 이름에 연결되고 사이드카 프록시는 실제 서비스 인스턴스로의 라우팅을 처리합니다. 블루/그린 배포의 경우 이 모델은 `testTrafficRules` 구성을 통한 테스트 트래픽 라우팅을 포함하도록 확장됩니다.

블루/그린 배포 중에 다음 주요 구성 요소가 함께 작동합니다.

- Service Connect 프록시: 서비스 간 모든 트래픽은 구성을 기반으로 라우팅 결정을 내리는 Service Connect 프록시를 통과합니다.
- AWS Cloud Map 등록: 블루 및 그린 배포 모두가 AWS Cloud Map에 등록되지만 그린 배포는 처음에 "테스트" 엔드포인트로 등록됩니다.

- 테스트 트래픽 라우팅: Service Connect 구성의 `testTrafficRules`는 테스트 트래픽을 식별하고 그린 배포로 라우팅하는 방법을 결정합니다. 이는 요청의 특정 HTTP 헤더가 트래픽을 테스트 개정으로 보내는 헤더 기반 라우팅을 통해 수행됩니다. 기본적으로 Service Connect는 사용자 지정 규칙이 지정되지 않은 경우 HTTP 기반 프로토콜의 `x-amzn-ecs-blue-green-test` 헤더를 인식합니다.
- 클라이언트 구성: 네임스페이스의 모든 클라이언트는 프로덕션 및 테스트 경로를 모두 자동으로 수신하지만 테스트 규칙과 일치하는 요청만 그린 배포로 이동합니다.

이 접근 방식이 강력한 이유는 전환 중 서비스 검색의 복잡성을 처리한다는 점 때문입니다. 트래픽이 블루에서 그린 배포로 전환되면 모든 연결 및 검색 메커니즘이 자동으로 업데이트됩니다. DNS 레코드를 업데이트하거나 로드 밸런서를 재구성하거나, 서비스 검색 변경 내용을 별도로 배포할 필요가 없습니다. 서비스 메시가 이 작업을 모두 처리하기 때문입니다.

트래픽 라우팅 및 테스트

Service Connect는 테스트 시나리오에 대한 헤더 기반 라우팅 및 클라이언트 별칭 구성을 포함하여 블루/그린 배포에 대한 고급 트래픽 라우팅 기능을 제공합니다.

테스트 트래픽 헤더 규칙

블루/그린 배포 중에 테스트 목적을 위해 특정 요청을 그린(신규) 서비스 개정으로 라우팅하도록 테스트 트래픽 헤더 규칙을 구성할 수 있습니다. 이렇게 하면 배포를 완료하기 전에 트래픽을 제어하며 새 버전을 검증할 수 있습니다.

Service Connect는 헤더 기반 라우팅을 사용하여 테스트 트래픽을 식별합니다. 기본적으로 Service Connect는 사용자 지정 규칙이 지정되지 않은 경우 HTTP 기반 프로토콜의 `x-amzn-ecs-blue-green-test` 헤더를 인식합니다. 요청에 이 헤더가 있으면 Service Connect 프록시는 테스트를 위해 요청을 그린 배포로 자동으로 라우팅합니다.

테스트 트래픽 헤더 규칙을 사용하면 다음을 수행할 수 있습니다.

- 특정 헤더가 있는 요청을 그린 서비스 개정으로 라우팅
- 트래픽 하위 집합을 사용하여 새 기능 테스트
- 전체 트래픽 전환 전에 서비스 동작 검증
- 카나리 테스트 전략 구현
- 프로덕션과 유사한 환경에서 통합 테스트 수행

헤더 기반 라우팅 메커니즘은 기존 애플리케이션 아키텍처에서 원활하게 작동합니다. 클라이언트 서비스는 블루/그린 배포 프로세스를 인식하지 않아도 됩니다. 테스트 요청을 보낼 때 적절한 헤더를 포함하면 Service Connect 프록시가 라우팅 로직을 자동으로 처리합니다.

테스트 트래픽 헤더 규칙을 구성하는 방법에 대한 자세한 내용은 Amazon Elastic Container Service API 참조의 ServiceConnectTestTrafficHeaderRules를 참조하세요.

헤더 일치 규칙

헤더 일치 규칙은 블루/그린 배포 중에 테스트 트래픽을 라우팅하기 위한 기준을 정의합니다. 그린 서비스 개정으로 라우팅되는 요청을 정확하게 제어하기 위해 여러 일치 조건을 구성할 수 있습니다.

헤더 일치에서는 다음을 지원합니다.

- 정확한 헤더 값 일치
- 헤더 존재 확인
- 패턴 기반 일치
- 여러 헤더 조합

사용 사례 예제에는 테스트를 위해 특정 사용자 에이전트 문자열, API 버전 또는 기능 플래그가 있는 요청을 그린 서비스로 라우팅하는 작업이 포함됩니다.

헤더 일치 구성에 대한 자세한 내용은 Amazon Elastic Container Service API 참조의 ServiceConnectTestTrafficHeaderMatchRules를 참조하세요.

블루/그린 배포에 대한 클라이언트 별칭

클라이언트 별칭은 블루/그린 배포 중 서비스에 안정적인 DNS 엔드포인트를 제공합니다. 이를 통해 클라이언트 애플리케이션이 연결 엔드포인트를 변경하지 않고도 블루 및 그린 서비스 개정 간에 원활한 트래픽 라우팅을 지원합니다.

블루/그린 배포 중에 클라이언트 별칭은 다음과 같은 기능을 수행합니다.

- 클라이언트 연결을 위해 일관된 DNS 이름 유지 관리
- 서비스 개정 간 자동 트래픽 전환 활성화
- 점진적 트래픽 마이그레이션 전략 지원
- 트래픽을 블루 개정으로 리디렉션하여 롤백 기능 제공

여러 포트 또는 프로토콜에 대해 여러 클라이언트 별칭을 구성하여 배포 중에 복잡한 서비스 아키텍처에서 연결을 유지할 수 있습니다.

클라이언트 별칭 구성에 대한 자세한 내용은 Amazon Elastic Container Service API 참조의 ServiceConnectClientAlias를 참조하세요.

트래픽 라우팅 모범 사례

Service Connect를 사용하는 블루/그린 배포에 대한 트래픽 라우팅을 구현하는 경우 다음 모범 사례를 고려하세요.

- 헤더 기반 테스트로 시작: 테스트 트래픽 헤더 규칙을 사용하여 모든 트래픽을 전환하기 전에 트래픽을 제어하며 그린 서비스를 검증합니다.
- 상태 확인 구성: 블루 및 그린 서비스 모두에서 비정상 인스턴스로 트래픽을 라우팅하지 않도록 적절한 상태 확인이 구성되어 있는지 확인합니다.
- 서비스 지표 모니터링: 배포 중에 두 서비스 개정에 대한 주요 성능 지표를 추적하여 문제를 조기에 식별합니다.
- 롤백 전략 계획: 문제가 감지되면 블루 서비스로 빠르게 롤백할 수 있도록 클라이언트 별칭 및 라우팅 규칙을 구성합니다.
- 헤더 일치 로직 테스트: 프로덕션 배포에 적용하기 전에 비프로덕션 환경에서 헤더 일치 규칙을 검증합니다.

Service Connect 블루/그린 배포 워크플로

Service Connect가 블루/그린 배포 프로세스를 관리하는 방법을 이해하면 배포를 효과적으로 구현하고 문제를 해결하는 데 도움이 됩니다. 다음 워크플로는 배포의 각 단계에서 서로 다른 구성 요소가 상호 작용하는 방식을 보여줍니다.

배포 단계

Service Connect 블루/그린 배포는 다음과 같은 여러 개별 단계를 통해 진행됩니다.

1. 초기 상태: 블루 서비스가 프로덕션 트래픽의 100%를 처리합니다. 네임스페이스의 모든 클라이언트 서비스는 Service Connect에 구성된 논리적 서비스 이름을 통해 블루 서비스에 연결됩니다.
2. 그린 서비스 등록: 그린 배포가 시작되면 AWS Cloud Map에 '테스트' 엔드포인트로 등록됩니다. 클라이언트 서비스의 Service Connect 프록시는 프로덕션 및 테스트 라우팅 구성을 모두 자동으로 수신합니다.

3. 테스트 트래픽 라우팅: 테스트 트래픽 헤더(예: `x-amzn-ecs-blue-green-test`)가 포함된 요청이 Service Connect 프록시에 의해 자동으로 그린 서비스로 라우팅됩니다. 프로덕션 트래픽은 계속해서 블루 서비스로 흐릅니다.
4. 트래픽 전환 준비: 테스트에 성공하면 배포 프로세스가 프로덕션 트래픽 전환을 준비합니다. 블루 및 그린 서비스는 모두 등록되고 정상 상태로 유지됩니다.
5. 프로덕션 트래픽 전환: 프로덕션 트래픽을 그린 서비스로 라우팅하도록 Service Connect 구성이 업데이트됩니다. 이는 클라이언트 서비스 업데이트 또는 DNS 변경 없이 자동으로 수행됩니다.
6. 베이크 소요 시간(기간): 프로덕션 트래픽이 전환된 후 블루 및 그린 서비스 개정이 동시에 실행되는 기간.
7. 블루 서비스 등록 취소: 트래픽 이동 및 검증에 성공하면 블루 서비스가 AWS Cloud Map에서 등록 취소되고 종료되며 배포가 완료됩니다.

Service Connect 프록시 동작

Service Connect 프록시는 블루/그린 배포 중에 트래픽을 관리하는 데 중요한 역할을 합니다. 이 동작을 이해하면 효과적인 테스트 및 배포 전략을 설계하는 데 도움이 됩니다.

블루/그린 배포 중 주요 프록시 동작:

- 자동 경로 검색: 프록시는 애플리케이션을 다시 시작하거나 구성을 변경할 필요 없이 AWS Cloud Map에서 프로덕션 및 테스트 경로를 모두 자동으로 검색합니다.
- 헤더 기반 라우팅: 프록시는 수신 요청 헤더를 검사하고 구성된 테스트 트래픽 규칙에 따라 트래픽을 적절한 서비스 개정으로 라우팅합니다.
- 상태 확인 통합: 프록시는 정상 서비스 인스턴스로만 트래픽을 라우팅하며, 라우팅 풀에서 비정상 태스크를 자동으로 제외합니다.
- 재시도 및 회로 차단: 프록시는 기본 제공 재시도 로직 및 회로 차단 기능을 제공하며 이를 통해 배포 중 복원력을 개선합니다.
- 지표 수집: 프록시는 블루 및 그린 서비스 모두에 대한 자세한 지표를 수집하여 배포 중에 포괄적인 모니터링을 지원합니다.

서비스 검색 업데이트

블루/그린 배포에 Service Connect를 사용하는 경우 주요 이점 중 하나는 서비스 검색 업데이트의 자동 처리입니다. 기존 블루/그린 배포에는 복잡한 DNS 업데이트 또는 로드 밸런서 재구성이 필요한 경우가 많지만 Service Connect는 이러한 변경 내용을 투명하게 관리합니다.

배포 중에 Service Connect는 다음을 처리합니다.

- 네임스페이스 업데이트: Service Connect 네임스페이스는 적절한 라우팅 규칙과 함께 블루 및 그린 서비스 엔드포인트를 모두 자동으로 포함합니다.
- 클라이언트 구성: 네임스페이스의 모든 클라이언트 서비스는 다시 시작하거나 재배포할 필요 없이 업데이트된 라우팅 정보를 자동으로 수신합니다.
- 점진적 전환: 서비스 검색 업데이트가 점진적으로 안전하게 수행되므로 진행 중인 요청이 중단되지 않습니다.
- 롤백 지원: 롤백이 필요한 경우 Service Connect는 서비스 검색 구성을 빠르게 되돌려 트래픽을 블루 서비스로 다시 라우팅할 수 있습니다.

Amazon ECS 블루/그린 배포 생성

Amazon RDS 블루/그린 배포를 사용하면 프로덕션 환경에서 구현하기 전에 서비스를 변경하고 해당 변경 내용을 테스트할 수 있습니다.

사전 조건

블루/그린 배포를 시작하기 전에 다음 작업을 수행합니다.

1. 적절한 권한을 구성하세요.
   - Elastic Load Balancing 권한에 대한 자세한 내용은 로드 밸런서에 대한 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.
   - Lambda 권한에 대한 자세한 내용은 Amazon ECS 블루/그린 배포에서 Lambda 함수에 필요한 권한 섹션을 참조하세요.
2. Amazon ECS 블루/그린 배포에서는 서비스가 다음 기능 중 하나를 사용해야 합니다. 적절한 리소스를 구성하세요.
   - Application Load Balancer - 자세한 내용은 블루/그린 배포에 대한 Application Load Balancer 리소스 섹션을 참조하세요.
   - Network Load Balancer - 자세한 내용은 Amazon ECS 블루/그린 배포에 대한 Network Load Balancer 리소스 섹션을 참조하세요.
   - Service Connect - 자세한 내용은 Amazon ECS 블루/그린 배포에 대한 Service Connect 리소스 섹션을 참조하세요.
3. 그린 서비스 개정으로 트래픽을 라우팅하는 규칙을 생성하세요. 자세한 내용은 Network Load Balancer 사용 설명서의 리스너 규칙을 참조하세요.

4. 그린 서비스 개정에 대한 대상 그룹을 생성하세요. 태스크에서 `awsvpc` 네트워크 모드를 사용하는 경우 대상 유형은 `ip`여야 합니다. 대상 그룹에 대한 자세한 내용은 Network Load Balancer 사용 설명서의 대상 그룹을 참조하세요.
5. 수명 주기 이벤트에 대해 Lambda 함수를 실행할지를 결정하세요.
   - 사전 스케일 업
   - 스케일 업 이후
   - 트래픽 전환 테스트
   - 트래픽 전환 테스트 이후
   - 프로덕션 트래픽 전환
   - 프로덕션 트래픽 전환 이후

   각 수명 주기 이벤트에 대한 Lambda 함수를 생성하세요. 자세한 내용은 AWS Lambda 개발자 안내서의 콘솔로 Lambda 함수 생성을 참조하세요.

절차

콘솔 또는 AWS CLI를 사용하여 Amazon ECS 블루/그린 서비스를 생성할 수 있습니다.

Console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 서비스를 시작할 리소스를 결정합니다.

서비스 시작	단계	
클러스터	a. 클러스터 페이지에서 서비스를 생성할 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 표시됩니다. b. 서비스(Services) 탭에서 생성(Create)을 선택합니다.	

서비스 시작	단계	
태스크 정의	a. 태스크 정의 페이지에서 태스크 정의를 선택하세요. b. 배포 메뉴에서 서비스 생성을 선택하세요.	

서비스 생성 페이지가 표시됩니다.

3. 서비스 세부 정보에서 다음을 수행합니다.

   a. 태스크 정의에서 사용할 태스크 정의 패밀리 및 개정을 선택하세요. 그런 다음 태스크 정의 개정에 사용할 개정을 입력하세요.

   b. 서비스 이름(Service name)에 서비스의 이름을 입력합니다.

4. 기존 클러스터에서 서비스를 실행하려면 기존 클러스터에서 클러스터를 선택하세요. 새 클러스터에서 서비스를 실행하려면 클러스터 생성을 선택하세요.

5. 클러스터 인프라에 태스크를 배포하는 방식을 선택합니다. 컴퓨팅 구성에서 옵션을 선택합니다.

컴퓨팅 옵션	단계	
용량 제공자 전략	a. 컴퓨팅 옵션에서 용량 공급자 전략을 선택합니다. b. 전략을 선택합니다. • 클러스터의 기본 용량 공급자 전략을 사용하려면 클러스터 기본값 사용(Use cluster default)을 선택합니다. • 클러스터에 기본 용량 공급자 전략이 없거나 사용자 지정 전략을 사용하려면 사용자 지정 사용, 용량 공급자 전	

컴퓨팅 옵션	단계
	략 추가를 선택하고 기본, 용량 공급자 및 가중치를 지정하여 사용자 지정 용량 공급자 전략을 정의합니다. **Note** 전략에서 용량 공급자를 사용하려면 용량 공급자가 클러스터와 연결되어야 합니다.
시작 유형	a. 컴퓨팅 옵션(Compute options) 섹션에서 시작 유형(Launch type)을 선택합니다. b. 시작 유형(Launch type)에서 시작 유형을 선택합니다. c. (선택 사항) Fargate 시작 유형이 지정되면 플랫폼 버전(Platform version)에서 사용할 플랫폼 버전을 지정합니다. 플랫폼 버전을 지정하지 않으면 LATEST 플랫폼 버전이 사용됩니다.

6. 배포 구성에서 다음을 수행합니다.

   a. 서비스 유형에서 복제본을 선택하세요.

   b. 원하는 작업(Desired tasks)에 서비스에서 시작 및 유지 관리할 작업 수를 입력합니다.

c. Amazon ECS에서 가용 영역 전체의 태스크 분산을 모니터링하고 불균형이 있을 때 재분산하려면 가용 영역 서비스 리밸런싱에서 가용 영역 서비스 리밸런싱을 선택하세요.

d. 상태 확인 유예 기간에 태스크가 처음 시작된 후 서비스 스케줄러가 비정상 Elastic Load Balancing, VPC Lattice, 컨테이너 상태 확인을 무시하는 시간(초)을 입력하세요. 상태 확인 유예 기간 값을 지정하지 않으면 기본값인 0이 사용됩니다.

7. a. 베이크 소요 시간에 블루 개정이 종료되기 전에 블루 및 그린 서비스 개정이 동시에 실행되는 시간(분)을 입력하세요. 이를 통해 확인 및 테스트 시간을 확보할 수 있습니다.

   b. (선택 사항) 배포의 특정 단계에서 실행할 Lambda 함수를 실행하세요. 배포 수명 주기 후크에서 수명 주기 후크를 실행할 단계를 선택하세요.

      수명 주기 후크를 추가하는 방법:

      a. 추가를 선택합니다.

      b. Lambda 함수에서 함수 이름 또는 ARN을 입력하세요.

      c. 역할에서 Lambda 함수를 간접 호출할 권한이 있는 IAM 역할을 선택하세요.

      d. 수명 주기 단계에서 Lambda 함수를 실행해야 하는 단계를 선택하세요.

8. Amazon ECS에서 배포 오류를 탐지 및 처리하는 방법을 구성하려면 배포 오류 탐지(Deployment failure detection)를 펼친 다음, 옵션을 선택합니다.

   a. 작업을 시작할 수 없을 때 배포를 중지하려면 Use the Amazon ECS deployment circuit breaker(Amazon ECS 배포 회로 차단기 사용)를 선택합니다.

      배포 회로 차단기가 배포를 실패한 상태로 설정했을 때 소프트웨어가 마지막으로 완료한 배포로 자동 롤백하도록 하려면 실패 시 롤백을 선택합니다.

   b. 애플리케이션 지표를 기반으로 배포를 중지하려면 CloudWatch 경보 사용을 선택합니다. 그런 다음, CloudWatch 경보 이름에서 경보를 선택합니다. 새 경보를 생성하려면 CloudWatch 콘솔을 이동합니다.

      CloudWatch 경보가 배포를 실패한 상태로 설정할 때 소프트웨어가 마지막으로 완료한 배포 상태로 자동 롤백하도록 하려면 실패 시 롤백을 선택합니다.

9. (선택 사항) Service Connect를 사용하여 서비스를 상호 연결하려면 Service Connect를 확장한 후 다음을 지정하세요.

   a. Service Connect 켜기를 선택합니다.

   b. Service Connect configuration(Service Connect 구성)에서 클라이언트 모드를 지정합니다.

- 서비스에서 네임스페이스의 다른 서비스에만 연결하면 되는 네트워크 클라이언트 애플리케이션을 실행하는 경우 클라이언트 측만을 선택합니다.
- 서비스가 네트워크 또는 웹 서비스 애플리케이션을 실행하고 이 서비스에 대한 엔드포인트를 제공해야 하며, 네임스페이스의 다른 서비스에 연결해야 하는 경우 Client and server(클라이언트 및 서버)를 선택합니다.

c. 기본 클러스터 네임스페이스가 아닌 네임스페이스를 사용하려면 Namespace(네임스페이스)에서 서비스 네임스페이스를 선택합니다.

d. (선택 사항) 블루/그린 배포에 대한 테스트 트래픽 헤더 규칙을 구성하세요. 테스트 트래픽 라우팅에서 다음을 지정하세요.

i. 테스트 중에 특정 요청을 그린 서비스 개정으로 라우팅하려면 테스트 트래픽 헤더 규칙 활성화를 선택하세요.

ii. 헤더 일치 규칙에서 테스트 트래픽 라우팅 기준을 구성하세요.

- 헤더 이름: 일치시킬 HTTP 헤더의 이름을 입력합니다(예: `X-Test-Version` 또는 `User-Agent`).
- 일치 유형: 일치 기준을 선택합니다.
  - 정확한 일치: 헤더 값이 지정된 값과 정확히 일치하는 요청을 라우팅합니다.
  - 헤더 있음: 값에 관계없이 지정된 헤더를 포함하는 라우팅 요청
  - 패턴 일치: 헤더 값이 지정된 패턴과 일치하는 요청 라우팅
- 헤더 값(정확한 일치 또는 패턴 일치를 사용하는 경우): 일치시킬 값 또는 패턴을 입력합니다.

여러 헤더 일치 규칙을 추가하여 복잡한 라우팅 로직을 생성할 수 있습니다. 구성된 규칙과 일치하는 요청은 테스트를 위해 그린 서비스 개정으로 라우팅됩니다.

iii. 헤더 규칙 추가를 선택하여 추가 헤더 일치 조건을 구성하세요.

> **Note**
>
> 테스트 트래픽 헤더 규칙을 사용하면 전체 배포를 완료하기 전에 트래픽을 제어하며 새 기능을 검증할 수 있습니다. 이를 통해 블루 서비스 개정으로 정상적인 트래픽 흐름을 유지하면서 특정 요청(예: 내부 테스트 도구 또는 베타 사용자의 요청)으로 그린 서비스 개정을 테스트할 수 있습니다.

e. (선택 사항) 로그 구성을 지정합니다. 로그 수집 사용을 선택합니다. 기본 옵션은 CloudWatch Logs로 컨테이너 로그를 전송합니다. 다른 로그 드라이버 옵션은 AWS FireLens를 사용하여 구성됩니다. 자세한 정보는 Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송을 참조하세요.

   다음은 각 컨테이너 로그 대상에 대한 자세한 설명입니다.

   - Amazon CloudWatch - CloudWatch Logs로 컨테이너 로그를 전송하도록 작업을 구성합니다. 사용자를 대신하여 CloudWatch 로그 그룹을 생성하는 기본 로그 드라이버 옵션이 제공됩니다. 다른 로그 그룹 이름을 지정하려면 드라이버 옵션 값을 변경합니다.
   - Amazon Data Firehose - Firehose로 컨테이너 로그를 전송하도록 작업을 구성합니다. Firehose 전송 스트림으로 로그를 전송하는 기본 로그 드라이버 옵션이 제공됩니다. 다른 전송 스트림 이름을 지정하려면 드라이버 옵션 값을 변경합니다.
   - Amazon Kinesis Data Streams - Kinesis Data Streams로 컨테이너 로그를 전송하도록 작업을 구성합니다. Kinesis Data Streams 스트림으로 로그를 전송하는 기본 로그 드라이버 옵션이 제공됩니다. 다른 스트림 이름을 지정하려면 드라이버 옵션 값을 변경합니다.
   - Amazon OpenSearch Service - OpenSearch Service 도메인으로 컨테이너 로그를 전송하도록 작업을 구성합니다. 로그 드라이버 옵션이 제공되어야 합니다.
   - Amazon S3 - Amazon S3 버킷으로 컨테이너 로그를 전송하도록 작업을 구성합니다. 기본 로그 드라이버 옵션이 제공되지만 유효한 Amazon S3 버킷 이름을 지정해야 합니다.

10. (선택 사항) 블루/그린 배포를 위한 로드 밸런싱을 구성하세요.

Elastic Load Balancing 유형	단계	
Application Load Balancer	a. 로드 밸런서 유형으로 Application Load Balancer를 선택하세요. b. 새 로드 밸런서 생성(Create a new load balancer)을 선택하여 새 Application Load Balancer를 생성하거나 기존 로드 밸런서 사용(Use an existing load balancer)을 선택하여	

Elastic Load Balancing 유형	단계	
	기존 Application Load Balancer를 선택합니다. c. 컨테이너에서 서비스를 호스팅할 컨테이너를 선택하세요. d. 로드 밸런스 이름(Load balancer name)의 고유 이름을 입력합니다. e. 리스너(Listener)에서 Application Load Balancer에 대해 연결 요청을 수신할 포트 및 프로토콜을 입력합니다. 기본적으로 로드 밸런서는 포트 80과 HTTP를 사용하도록 구성됩니다. • 프로덕션 규칙에 규칙의 평가 순서 및 경로 패턴을 입력하세요. 이 규칙은 프로덕션(블루) 서비스 개정 트래픽에 적용됩니다. • 테스트 규칙에 규칙의 평가 순서 및 경로 패턴을 입력하세요. 이 규칙은 테스트(그린) 서비스 개정 트래픽에 적용됩니다. f. 대상 그룹에서 다음을 구성하세요. • 대상 그룹 이름(Target group name)에	

Elastic Load Balancing 유형	단계	
	Application Load Balancer에서 요청을 라우팅할 대상 그룹에 대한 이름과 프로토콜을 입력합니다. • 프로토콜에서 Application Load Balancer가 요청을 라우팅하는 대상 그룹의 프로토콜을 선택하세요. 기본적으로 대상 그룹은 태스크 정의에 정의된 첫 번째 컨테이너로 요청을 라우팅합니다. • 등록 취소 지연에 로드 밸런서가 대상 상태를 UNUSED로 변경하는 데 걸리는 시간(초)을 입력합니다. 기본값은 300초입니다. • 상태 확인 경로(Health check path)에서 Application Load Balancer가 Application Load Balancer와 컨테이너 간의 연결 상태를 확인하기 위해 주기적으로 요청을 보내야 하는 컨테이너 내에 존재하는 경로를 입력합니다. 기본값은 루트 디렉터리입니다(/). • 대체 그룹 이름에 테스트(그린) 서비스 개정의	

Elastic Load Balancing 유형	단계	
	대상 그룹에 대한 그룹 이름을 입력하세요.	

Elastic Load Balancing 유형	단계	
Network Load Balancer	a. 로드 밸런서 유형(Load balancer type)에서 Network Load Balancer를 선택합니다. b. Load Balancer에서 기존 Network Load Balancer를 선택합니다. c. 부하를 분산할 컨테이너 선택(Choose container to load balance)에서 서비스를 호스팅할 컨테이너를 선택합니다. d. 프로덕션 리스너에서 프로덕션 리스너 포트 및 프로덕션 리스너 프로토콜을 선택하세요. 프로덕션(블루) 서비스 개정 트래픽의 리스너입니다. e. 테스트 리스너에서 테스트 리스너 포트 및 테스트 리스너 프로토콜을 선택하세요. 테스트(그린) 서비스 개정 트래픽의 리스너입니다. f. 대상 그룹에서 다음을 구성하세요. • Target group name(대상 그룹 이름)에 Network Load Balancer에서 요청을 라우팅할 대상 그룹에	

Elastic Load Balancing 유형	단계	
	대한 이름과 프로토콜을 입력합니다. • 프로토콜에서 Network Load Balancer가 요청을 라우팅하는 대상 그룹의 프로토콜을 선택하세요. 기본적으로 대상 그룹은 태스크 정의에 정의된 첫 번째 컨테이너로 요청을 라우팅합니다. • 등록 취소 지연에 로드 밸런서가 대상 상태를 UNUSED로 변경하는 데 걸리는 시간(초)을 입력합니다. 기본값은 300초입니다. • 상태 확인 경로(Health check path)에서 Application Load Balancer가 Application Load Balancer와 컨테이너 간의 연결 상태를 확인하기 위해 주기적으로 요청을 보내야 하는 컨테이너 내에 존재하는 경로를 입력합니다. 기본값은 루트 디렉터리입니다(/). • 대체 그룹 이름에 테스트(그린) 서비스 개정의 대상 그룹에 대한 그룹 이름을 입력하세요.	

11. (선택 사항) 서비스와 태스크를 식별하려면 태그(Tags) 섹션을 펼친 다음, 태그를 구성합니다.

    Amazon ECS에서 새로 시작된 모든 작업에 클러스터 이름과 작업 정의 태그를 자동으로 지정하도록 하려면 Amazon ECS 관리형 태그 켜기를 선택한 다음 작업 전파 시작에서 작업 정의를 선택합니다.

    Amazon ECS에서 새로 시작된 모든 작업에 클러스터 이름과 서비스 태그를 자동으로 지정하도록 하려면 Amazon ECS 관리형 태그 켜기를 선택한 다음 작업 전파 시작에서 서비스를 선택합니다.

    태그를 추가하거나 제거합니다.

    - [태그 추가] 새로운 태그(Add tag)를 선택하고 다음을 수행합니다.
      - 키에서 키 이름을 입력합니다.
      - 값에 키 값을 입력합니다.
    - [태그 제거] 태그 옆에 있는 태그 제거를 선택합니다.
12. 생성(Create)을 선택합니다.

AWS CLI

1. 다음 콘텐츠가 포함된 `service-definition.json`이라는 파일을 생성합니다.

    모든 *### ##*을 사용자의 값으로 바꿉니다.

```
{
  "serviceName": "myBlueGreenService",
  "cluster": "arn:aws:ecs:us-west-2:123456789012:cluster/sample-fargate-
cluster",
  "taskDefinition": "sample-fargate:1",
  "desiredCount": 5,
  "launchType": "FARGATE",
  "networkConfiguration": {
    "awsvpcConfiguration": {
      "subnets": [
        "subnet-09ce6e74c116a2299",
        "subnet-00bb3bd7a73526788",
        "subnet-0048a611aaec65477"
      ],
      "securityGroups": [
        "sg-09d45005497daa123"
```

```
      ],
      "assignPublicIp": "ENABLED"
    }
  },
  "deploymentController": {
    "type": "ECS"
  },
  "deploymentConfiguration": {
    "strategy": "BLUE_GREEN",
    "maximumPercent": 200,
    "minimumHealthyPercent": 100,
    "bakeTimeInMinutes": 2,
    "alarms": {
      "alarmNames": [
        "myAlarm"
      ],
      "rollback": true,
      "enable": true
    },
    "lifecycleHooks": [
      {
        "hookTargetArn": "arn:aws:lambda:us-
west-2:7123456789012:function:checkExample",
        "roleArn": "arn:aws:iam::123456789012:role/ECSLifecycleHookInvoke",
        "lifecycleStages": [
          "PRE_SCALE_UP"
        ]
      }
    ]
  },
  "loadBalancers": [
    {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/blue-target-group/54402ff563af1197",
      "containerName": "fargate-app",
      "containerPort": 80,
      "advancedConfiguration": {
        "alternateTargetGroupArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/green-target-group/cad10a56f5843199",
        "productionListenerRule": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:listener-rule/app/my-blue-green-
demo/32e0e4f946c3c05b/9cfa8c482e204f7d/831dbaf72edb911",
        "roleArn": "arn:aws:iam::123456789012:role/LoadBalancerManagementforECS"
      }
```

```
    }
  ]
}
```

2. create-service을(를) 실행합니다.

   모든 *### ##*을 사용자의 값으로 바꿉니다.

```
aws ecs create-service --cli-input-json file://service-definition.json
```

   또는 로드 밸런서 구성으로 블루/그린 배포 서비스를 생성하는 다음 예제를 사용할 수 있습니다.

```
aws ecs create-service \
   --cluster "arn:aws:ecs:us-west-2:123456789012:cluster/MyCluster" \
   --service-name "blue-green-example-service" \
   --task-definition "nginxServer:1" \
   --launch-type "FARGATE" \
   --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12345,subnet-67890,subnet-abcdef,subnet-
fedcba],securityGroups=[sg-12345],assignPublicIp=ENABLED}" \
   --desired-count 3 \
   --deployment-controller "type=ECS" \
   --deployment-configuration
 "strategy=BLUE_GREEN,maximumPercent=200,minimumHealthyPercent=100,bakeTimeInMinutes=0"
 \
   --load-balancers "targetGroupArn=arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/MyBGtg1/
abcdef1234567890,containerName=nginx,containerPort=80,advancedConfiguration={alternateTa
west-2:123456789012:targetgroup/
MyBGtg2/0987654321fedcba,productionListenerRule=arn:aws:elasticloadbalancing:us-
west-2:123456789012:listener-rule/app/
MyLB/1234567890abcdef/1234567890abcdef,roleArn=arn:aws:iam::123456789012:role/
ELBManagementRole}"
```

다음 단계

- 배포를 시작하도록 서비스를 업데이트합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.
- 배포 프로세스를 모니터링하여 블루/그린 패턴을 따르는지 확인합니다.

- 그린 서비스 개정이 생성되고 스케일 업됨
- 테스트 트래픽이 그린 개정으로 라우팅됨(구성된 경우)
- 프로덕션 트래픽이 그린 개정으로 전환됨
- 베이크 소요 시간이 지나면 블루 개정이 종료됨

타사 컨트롤러를 사용한 Amazon ECS 서비스 배포

외부 배포 유형을 사용하면 타사 배포 컨트롤러를 사용하여 Amazon ECS 서비스의 배포 프로세스를 완벽하게 제어할 수 있습니다. 서비스에 대한 세부 정보는 서비스 관리 API 작업(`CreateService`, `UpdateService` 및 `DeleteService`) 또는 작업 세트 관리 API 작업(`CreateTaskSet`, `UpdateTaskSet`, `UpdateServicePrimaryTaskSet` 및 `DeleteTaskSet`)에 의해 관리됩니다. 각 API 작업은 서비스 정의 파라미터의 하위 집합을 관리합니다.

`UpdateService` API 동작은 서비스의 원하는 개수 및 상태 확인 유예 기간 파라미터를 업데이트합니다. 시작 유형, 플랫폼 버전, 로드 밸런서 세부 정보, 네트워크 구성 또는 태스크 정의를 업데이트해야 하는 경우 새 작업 세트를 만들어야 합니다.

`UpdateTaskSet` API 태스크는 작업 세트의 배율 파라미터만 업데이트합니다.

`UpdateServicePrimaryTaskSet` API 태스크는 서비스의 어떤 작업 세트가 기본 작업 세트인지 수정합니다. `DescribeServices` API 태스크를 호출하면 기본 작업 세트에 대해 지정된 모든 필드를 반환합니다. 서비스에 대한 기본 작업 세트가 업데이트되면 새 기본 작업 세트가 정의될 때 서비스에 설정된 이전 기본 작업 세트와 다른 새 기본 작업 세트에 있는 작업 세트 파라미터 값이 새 값으로 업데이트됩니다. 서비스에 대한 기본 작업 세트가 정의되지 않은 경우, 서비스를 설명할 때 작업 세트 필드가 null입니다.

외부 배포 고려 사항

외부 배포 유형을 사용할 때는 다음 사항을 고려합니다.

- 지원되는 로드 밸런서 유형은 Application Load Balancer 또는 Network Load Balancer입니다.
- Fargate 시작 유형이나 `EXTERNAL` 배포 컨트롤러 유형을 사용하는 태스크는 `DAEMON` 일정 전략을 지원하지 않습니다.
- Amazon ECS와의 Application AutoScaling 통합은 Amazon ECS 서비스만 대상으로 지원합니다. Amazon ECS에 대해 지원되는 확장 가능 차원은 `ecs:service:DesiredCount`(Amazon ECS 서비스의 태스크 수)입니다. Application AutoScaling과 Amazon ECS 태스크 세트 사이에서 직접적인 통합이 없습니다. Amazon ECS 태스크 세트는 Amazon ECS 서비스 `DesiredCount`에 기반하여 `ComputedDesiredCount`를 계산합니다.

외부 배포 워크플로

다음은 Amazon ECS에서 외부 배포를 관리하기 위한 기본 워크플로입니다.

외부 배포 컨트롤러를 사용하여 Amazon ECS 서비스를 관리하려면

1. Amazon ECS 서비스를 생성합니다. 이때는 서비스 이름 파라미터만 있으면 됩니다. 외부 배포 컨트롤러를 사용하여 서비스를 생성할 때 다음 파라미터를 지정할 수 있습니다. 다른 모든 서비스 파라미터는 서비스 내에 작업 세트를 생성할 때 지정됩니다.

   `serviceName`

   유형: 문자열

   필수 항목 여부: 예

   서비스의 이름입니다. 최대 255개의 문자(대문자 및 소문자), 숫자, 하이픈 및 밑줄이 허용됩니다. 서비스 이름은 클러스터 내에서 고유해야 하지만, 한 리전 또는 여러 리전에 걸쳐 존재하는 여러 클러스터에서 비슷한 서비스 이름을 사용할 수 있습니다.

   `desiredCount`

   서비스 내에 배포되어 실행되도록 지정된 작업 세트 태스크 정의의 인스턴스 수입니다.

   `deploymentConfiguration`

   배포 시 실행할 작업 수 및 작업 중지/시작 순서를 제어하는 선택적 배포 파라미터입니다.

   `tags`

   타입: 객체 배열

   필수 여부: 아니요

   서비스를 분류하고 구성하는 데 도움이 되도록 서비스에 적용하는 메타데이터입니다. 각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다. 서비스가 삭제되면 태그도 함께 삭제됩니다. 서비스에 최대 50개의 태그를 적용할 수 있습니다. 자세한 정보는 Amazon ECS 리소스 태그 지정 섹션을 참조하세요.

   `key`

   유형: 문자열

   길이 제약: 최소 길이는 1. 최대 길이는 128입니다.

필수 여부: 아니요

하나의 태그를 구성하는 키-값 쌍의 일부분입니다. 키는 더 구체적인 태그 값에 대해 범주와 같은 역할을 하는 일반적인 레이블입니다.

`value`

유형: 문자열

길이 제약: 최소 길이는 0. 최대 길이 256.

필수 여부: 아니요

하나의 태그를 구성하는 키-값 쌍의 선택적 부분입니다. 하나의 값은 태그 범주(키) 내에서 서술자 역할을 수행합니다.

`enableECSManagedTags`

서비스 내의 태스크에 대해 Amazon ECS 관리형 태그를 사용할지를 지정합니다. 자세한 정보는 결제에 태그 사용을 참조하세요.

`propagateTags`

유형: 문자열

유효한 값: `TASK_DEFINITION` | `SERVICE`

필수 여부: 아니요

태그를 태스크 정의 또는 서비스에서 서비스의 작업으로 복사할지를 지정합니다. 값을 지정하지 않을 경우 태그는 복사되지 않습니다. 태그는 서비스 생성 중에 서비스 내의 작업에만 복사할 수 있습니다. 서비스 생성 후 작업 생성에 태그를 추가하려면 `TagResource` API 태스크를 사용합니다.

`schedulingStrategy`

사용할 일정 전략입니다. 외부 배포 컨트롤러를 사용하는 서비스는 `REPLICA` 일정 전략만 지원합니다.

`placementConstraints`

서비스 내 작업에 사용할 배치 제약 객체의 배열입니다. 작업당 최대 10개의 제약을 지정할 수 있습니다(이 제한에는 태스크 정의 내 제약과 런타임 시 지정되는 제약이 포함됨). Fargate 시작 유형을 사용하는 경우 작업 배치 제약은 지원되지 않습니다.

`placementStrategy`

서비스 내 작업에 사용할 배치 전략 객체입니다. 서비스당 최대 4개까지 전략 규칙을 지정할 수 있습니다.

다음은 외부 배포 컨트롤러를 사용하는 서비스를 생성하기 위한 예제 서비스 정의입니다.

```
{
    "cluster": "",
    "serviceName": "",
    "desiredCount": 0,
    "role": "",
    "deploymentConfiguration": {
        "maximumPercent": 0,
        "minimumHealthyPercent": 0
    },
    "placementConstraints": [
        {
            "type": "distinctInstance",
            "expression": ""
        }
    ],
    "placementStrategy": [
        {
            "type": "binpack",
            "field": ""
        }
    ],
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "EXTERNAL"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "TASK_DEFINITION"
}
```

2. 초기 작업 세트를 생성합니다. 작업 세트에는 서비스에 대한 다음 세부 정보가 들어 있습니다.

`taskDefinition`

사용할 작업 세트의 작업에 대한 태스크 정의입니다.

`launchType`

유형: 문자열

유효한 값: `EC2` | `FARGATE` | `EXTERNAL`

필수 여부: 아니요

서비스를 실행할 시작 유형. 시작 유형을 지정하지 않으면 기본적으로 `capacityProviderStrategy`가 사용됩니다. 자세한 정보는 Amazon ECS 시작 유형을 참조하세요.

`launchType`이 지정된 경우 `capacityProviderStrategy` 파라미터를 생략해야 합니다.

`platformVersion`

유형: 문자열

필수 여부: 아니요

서비스의 작업이 실행 중인 플랫폼 버전입니다. 플랫폼 버전은 Fargate 시작 유형을 사용하는 작업에만 지정됩니다. 지정하지 않으면 기본적으로 최신 버전(`LATEST`)이 사용됩니다.

AWS Fargate 플랫폼 버전은 Fargate 태스크 인프라를 위한 특정 실행 시간 환경을 참조하는 데 사용합니다. 태스크를 실행하거나 서비스를 생성할 때 `LATEST` 플랫폼 버전을 지정하면 해당 작업에 사용 가능한 최신 플랫폼 버전을 얻을 수 있습니다. 서비스를 확장하면 해당 태스크는 서비스의 현재 배포에 지정된 플랫폼 버전을 받습니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.

> **Note**
>
> 플랫폼 버전은 EC2 시작 유형을 사용하는 작업에는 지정되지 않습니다.

`loadBalancers`

서비스와 함께 사용할 로드 밸런서를 나타내는 로드 밸런서 객체입니다. 외부 배포 컨트롤러를 사용하는 경우 Application Load Balancer 및 Network Load Balancer만 지원됩니다. Application Load Balancer를 사용하는 경우 작업 세트당 하나의 Application Load Balancer 대상 그룹만 허용됩니다.

다음 조각은 사용할 `loadBalancer` 객체의 예제를 보여줍니다.

```
"loadBalancers": [
        {
            "targetGroupArn": "",
            "containerName": "",
            "containerPort": 0
        }
]
```

> **Note**
>
> `loadBalancer` 객체를 지정할 때는 `targetGroupArn`을 지정하고 `loadBalancerName` 파라미터를 생략해야 합니다.

`networkConfiguration`

서비스에 대한 네트워크 구성. 이 파라미터는 `awsvpc` 네트워크 모드를 사용하는 태스크 정의가 고유한 탄력적 네트워크 인터페이스를 받는 데 필요하며 다른 네트워크 모드에 대해서는 지원되지 않습니다. Fargate 시작 유형의 네트워킹에 대한 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션 섹션을 참조하세요.

`serviceRegistries`

이 서비스에 할당할 서비스 검색 레지스트리의 세부 정보입니다. 자세한 정보는 서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결 섹션을 참조하세요.

`scale`

작업 세트에 배치하고 실행하려는 작업 수의 부동 소수점 백분율입니다. 이 값은 서비스의 `desiredCount`에 대한 총 백분율로 지정됩니다. 허용되는 값은 0과 100 사이의 숫자입니다.

다음은 외부 배포 컨트롤러에 대한 작업 세트를 생성하기 위한 JSON 예제입니다.

```
{
    "service": "",
    "cluster": "",
    "externalId": "",
    "taskDefinition": "",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    },
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "launchType": "EC2",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
            "weight": 0,
            "base": 0
        }
    ],
    "platformVersion": "",
```

```
    "scale": {
        "value": null,
        "unit": "PERCENT"
    },
    "clientToken": ""
}
```

3. 서비스 변경이 필요한 경우 업데이트 중인 파라미터에 따라 `UpdateService`, `UpdateTaskSet` 또는 `CreateTaskSet` API 태스크를 사용합니다. 작업 세트를 만든 경우 서비스의 각 작업 세트에 대해 `scale` 파라미터를 사용하여 서비스에서 실행 중인 작업 수를 결정합니다. 예를 들어, `tasksetA`가 포함된 서비스가 있고 `tasksetB`를 생성한 경우 프로덕션 트래픽을 전환하기 전에 `tasksetB`의 유효성을 테스트할 수 있습니다. 두 작업 세트의 `scale`을 `100`으로 설정할 수 있으며 모든 프로덕션 트래픽을 `tasksetB`로 전환할 준비가 되면 `tasksetA`의 `scale`을 `0`으로 업데이트하여 축소할 수 있습니다.

Amazon ECS에 대한 CodeDeploy 블루/그린 배포

Amazon ECS 블루/그린 배포를 사용하는 것이 좋습니다. 자세한 내용은 Amazon ECS 블루/그린 배포 생성 섹션을 참조하세요.

블루/그린 배포 유형은 CodeDeploy로 제어되는 블루/그린 배포 모델을 사용합니다. 이 배포 유형을 사용하면 프로덕션 트래픽을 전송하기 전에 서비스의 새로운 배포를 확인합니다. 자세한 내용은 AWS CodeDeploy 사용 설명서의 CodeDeploy란 무엇입니까?를 참조하세요.배포 전에 Amazon ECS 서비스의 상태 확인

블루/그린 배포 중 트래픽을 이동할 수 있는 세 가지 방법이 있습니다.

- Canary: 트래픽이 두 증분으로 나뉘어 이동합니다. 나머지 트래픽이 두 번째 증분으로 이동하기 전에 첫 번째 증분에서 업데이트된 작업 세트로 이동할 트래픽 비율(%)과 간격(분)을 지정하는 사전 정의된 Canary 옵션 중에서 선택할 수 있습니다.
- Linear: 트래픽이 동일한 증분 이동하며 각 증분 간에 시간 간격(분)이 동일합니다. 각 증분에서 이동되는 트래픽 비율(%)과 각 증분 간의 시간(분)을 지정하는 사전 정의된 Linear 옵션에서 선택할 수 있습니다.
- All-at-once: 모든 트래픽이 원래 작업 세트에서 업데이트된 작업 세트로 모두 한 번에 이동합니다.

서비스가 블루/그린 배포 유형을 사용할 때 Amazon ECS가 사용하는 CodeDeploy의 구성 요소들은 다음과 같습니다.

CodeDeploy 애플리케이션

CodeDeploy 리소스 컬렉션. 이것은 하나 이상의 배포 그룹으로 구성됩니다.

CodeDeploy 배포 그룹

배포 설정. 이것은 다음과 같은 요소로 구성됩니다.

- Amazon ECS 클러스터와 서비스
- 로드 밸런서 대상 그룹 및 리스너 정보
- 배포 롤백 전략
- 트래픽 다시 라우팅 설정
- 원래 개정 종료 설정
- 배포 구성
- 배포를 중지하도록 설정할 수 있는 CloudWatch 경보 구성
- SNS 또는 CloudWatch Events 알림 설정

자세한 정보는 AWS CodeDeploy 사용 설명서의 배포 그룹으로 작업을 참조하세요.

CodeDeploy 배포 구성

CodeDeploy가 배포 중 프로덕션 트래픽을 교체 작업 세트로 라우팅하는 방법을 지정합니다. 다음과 같은 사전 정의된 Linear 및 Canary 배포 구성을 사용할 수 있습니다. 사용자 정의 Linear 및 Canary 배포도 생성할 수 있습니다. 자세한 정보는 AWS CodeDeploy 사용 설명서의 배포 구성으로 작업을 참조하세요.

- CodeDeployDefault.ECSAllAtOnce: 모든 트래픽을 업데이트된 Amazon ECS 컨테이너로 한 번에 이동합니다.
- CodeDeployDefault.ECSLinear10PercentEvery1Minutes: 모든 트래픽이 이동할 때까지 매분 트래픽의 10%를 이동합니다.
- CodeDeployDefault.ECSLinear10PercentEvery3Minutes: 모든 트래픽이 이동할 때까지 3분마다 트래픽의 10%를 이동합니다.
- CodeDeployDefault.ECSCanary10Percent5Minutes: 첫 번째 증분에서 트래픽의 10%를 이동합니다. 나머지 90%는 5분 이후 배포됩니다.
- CodeDeployDefault.ECSCanary10Percent15Minutes: 첫 번째 증분에서 트래픽의 10%를 이동합니다. 나머지 90%는 15분 이후 배포됩니다.

개정

개정은 CodeDeploy 애플리케이션 사양 파일(AppSpec 파일)입니다. AppSpec 파일에서 태스크 정의의 전체 ARN을 지정하고, 새 배치가 생성될 때 트래픽이 라우팅되는 대상에 해당하는 교체

작업 세트의 컨테이너 및 포트도 지정합니다. 컨테이너 이름은 태스크 정의에서 참조되는 컨테이너 이름 중 하나여야 합니다. 서비스 정의에서 네트워크 구성 또는 플랫폼 버전이 업데이트된 경우, AppSpec 파일에서도 해당 세부 정보를 지정해야 합니다. 배포 수명 주기 이벤트 중에 실행할 Lambda 함수를 지정할 수도 있습니다. Lambda 함수를 사용하면 배포 중에 테스트를 실행하고 지표를 반환할 수 있습니다. 자세한 정보는 AWS CodeDeploy 사용 설명서의 AppSpec 파일 참조를 참조하세요.

고려 사항

블루/그린 배포 유형 사용 시 고려할 몇 가지 사항들은 다음과 같습니다.

- 블루/그린 배포 유형을 사용하는 Amazon ECS 서비스가 맨 처음 생성되면 Amazon ECS 작업 세트가 생성됩니다.
- Application Load Balancer 또는 Network Load Balancer를 사용하도록 서비스를 구성해야 합니다. 로드 밸런서의 요구 사항들은 다음과 같습니다.
  - 프로덕션 트래픽을 라우팅하는 데 사용되는 로드 밸런서에 프로덕션 리스너를 추가해야 합니다.
  - 테스트 트래픽을 라우팅하는 데 사용되는 로드 밸런서에는 테스트 리스너(선택 사항)를 추가해야 합니다. 테스트 리스너를 지정하면 CodeDeploy는 배포 중에 테스트 트래픽을 교체 작업 세트로 라우팅합니다.
  - 프로덕션 및 테스트 리스너는 모두 동일한 로드 밸런서에 속해야 합니다.
  - 로드 밸런서에 대한 대상 그룹을 정의해야 합니다. 대상 그룹은 트래픽을 프로덕션 리스너를 통해 서비스의 원래 작업 세트로 라우팅합니다.
  - Network Load Balancer를 사용하는 경우 `CodeDeployDefault.ECSAllAtOnce` 배포 구성만이 지원됩니다.
- 서비스 Auto Scaling 및 블루/그린 배포 유형을 사용하도록 구성된 서비스의 경우 배포 중에 Auto Scaling이 차단되지 않지만 일부 상황에서는 배포가 실패할 수 있습니다. 이 동작에 대한 자세한 설명은 아래와 같습니다.
  - 서비스가 조정되고 배포가 시작되면 그린 작업 세트가 생성되고, CodeDeploy는 그린 작업 세트가 정상 상태에 도달하도록 최대 1시간 동안 대기하며 트래픽이 이동하지 않습니다.
  - 서비스가 블루/그린 배포 중이고 조정 이벤트가 발생하면 트래픽은 5분 동안 계속 이동합니다. 서비스가 5분 이내에 정상 상태에 도달하지 않으면 CodeDeploy는 배포를 중지하고 실패로 표시합니다.
- Fargate 시작 유형이나 `CODE_DEPLOY` 배포 컨트롤러 유형을 사용하는 태스크는 `DAEMON` 일정 전략을 지원하지 않습니다.
- CodeDeploy 애플리케이션 및 배포 그룹을 처음 생성할 때 다음을 지정해야 합니다.

- 로드 밸런서에 대해 두 개의 대상 그룹을 정의해야 합니다. 하나의 대상 그룹은 Amazon ECS 서비스가 생성될 때 로드 밸런서에 대해 정의된 초기 대상 그룹이어야 합니다. 두 번째 대상 그룹의 유일한 요구 사항은 서비스가 사용하는 것과 다른 로드 밸런서에 연결할 수 없다는 것입니다.
- Amazon ECS 서비스에 대해 CodeDeploy 배포가 생성되면 CodeDeploy는 배포에서 교체 작업 세트(또는 그린 작업 세트)를 생성합니다. 테스트 리스너를 로드 밸런서에 추가한 경우 CodeDeploy는 테스트 트래픽을 교체 작업 세트로 라우팅합니다. 이때 모든 확인 테스트를 실행할 수 있습니다. 그런 다음, CodeDeploy는 프로덕션 트래픽을 원래 작업 세트에서 배포 그룹에 대한 트래픽 다시 라우팅 설정에 따라 교체 작업 세트로 다시 라우팅합니다.

필수 IAM 권한

블루/그린 배포는 Amazon ECS와 CodeDeploy API의 조합으로 가능합니다. 사용자는 AWS Management Console에서 또는 AWS CLI 또는 SDK를 사용하여 Amazon ECS 블루/그린 배포를 사용하기 전에 이들 서비스에 적합한 권한이 있어야 합니다.

Amazon ECS는 서비스 생성 및 업데이트를 위한 표준 IAM 권한 외에 다음과 같은 권한을 필요로 합니다. 이러한 권한은 `AmazonECS_FullAccess` IAM 정책에 추가되었습니다. 자세한 내용은 AmazonECS_FullAccess 섹션을 참조하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:CreateApplication",
                "codedeploy:CreateDeployment",
                "codedeploy:CreateDeploymentGroup",
                "codedeploy:GetApplication",
                "codedeploy:GetDeployment",
                "codedeploy:GetDeploymentGroup",
                "codedeploy:ListApplications",
                "codedeploy:ListDeploymentGroups",
                "codedeploy:ListDeployments",
                "codedeploy:StopDeployment",
                "codedeploy:GetDeploymentTarget",
                "codedeploy:ListDeploymentTargets",
                "codedeploy:GetDeploymentConfig",
                "codedeploy:GetApplicationRevision",
                "codedeploy:RegisterApplicationRevision",
```

```
                "codedeploy:BatchGetApplicationRevisions",
                "codedeploy:BatchGetDeploymentGroups",
                "codedeploy:BatchGetDeployments",
                "codedeploy:BatchGetApplications",
                "codedeploy:ListApplicationRevisions",
                "codedeploy:ListDeploymentConfigs",
                "codedeploy:ContinueDeployment",
                "sns:ListTopics",
                "cloudwatch:DescribeAlarms",
                "lambda:ListFunctions"
            ],
            "Resource": ["*"]
        }
    ]
}
```

> **Note**
>
> 작업과 서비스를 실행하는 데 필요한 표준 Amazon ECS 권한 이외에, 사용자는 작업에 대한 IAM 역할을 사용하기 위한 `iam:PassRole` 권한도 필요합니다.

CodeDeploy는 Amazon ECS API를 호출하고 Elastic Load Balancing을 수정하며 Lambda 함수를 호출하고 CloudWatch 경보를 설명할 수 있는 권한은 물론, 사용자를 대신하여 원하는 서비스 개수를 수정할 수 있는 권한도 필요합니다. 블루/그린 배포 유형을 사용하는 Amazon ECS 서비스를 생성하기 전에 IAM 역할(`ecsCodeDeployRole`)을 생성해야 합니다. 자세한 내용은 Amazon ECS CodeDeploy IAM 역할 섹션을 참조하세요.

Amazon ECS 서비스 예제 생성 및 Amazon ECS 서비스 예제 업데이트 IAM 정책 예제는 사용자가 AWS Management Console에서 Amazon ECS 블루/그린 배포를 사용하는 데 필요한 권한을 보여 줍니다.

CodeDeploy 블루/그린 배포를 Amazon ECS 블루/그린 배포로 마이그레이션

CodeDeploy 블루/그린 및 Amazon ECS 블루/그린 배포는 유사한 기능을 제공하지만 구성 및 관리 방법은 서로 다릅니다.

CodeDeploy 블루/그린 배포 개요

CodeDeploy를 사용하여 Amazon ECS 서비스를 생성하는 경우 다음을 수행합니다.

1. 프로덕션 리스너 및 (선택 사항) 테스트 리스너를 사용하여 로드 밸런서를 생성합니다. 각 리스너는 모든 트래픽을 단일 대상 그룹(기본 대상 그룹)으로 라우팅하는 단일(기본) 규칙으로 구성됩니다.
2. `deploymentController` 유형이 `CODE_DEPLOY`로 설정된 리스너 및 대상 그룹을 사용하도록 구성된 Amazon ECS 서비스를 생성합니다. 서비스를 생성하면 지정된 대상 그룹에 등록된 (블루) 태스크 세트가 생성됩니다.
3. CodeDeploy 배포 그룹(CodeDeploy 애플리케이션의 일부)을 생성하고 Amazon ECS 클러스터, 서비스 이름, 로드 밸런서 리스너, 두 개의 대상 그룹(프로덕션 리스너 규칙에 사용되는 기본 대상 그룹, 대체 태스크에 사용할 보조 대상 그룹), 서비스 역할(Amazon ECS 및 Elastic Load Balancing 리소스를 조작할 수 있는 권한을 CodeDeploy에 부여), 배포 동작을 제어하는 다양한 파라미터의 세부 정보로 구성합니다.

CodeDeploy를 사용하면 `CreateDeployment()`를 사용하여 새 버전의 서비스가 배포되고 이때 CodeDeploy 애플리케이션 이름, 배포 그룹 이름 및 새 개정과 선택적 수명 주기 후크에 대한 세부 정보를 제공하는 AppSpec 파일을 지정합니다. CodeDeploy 배포는 대체(그린) 태스크 세트를 생성하고 해당 태스크를 보조 대상 그룹에 등록합니다. 정상 상태가 되면 테스트(선택 사항) 및 프로덕션에서 사용할 수 있습니다. 두 경우 모두 그린 태스크 세트와 연결된 보조 대상 그룹을 가리키도록 각 리스너 규칙을 변경하여 다시 라우팅할 수 있습니다. 롤백은 프로덕션 리스너 규칙을 기본 대상 그룹으로 다시 변경하여 수행됩니다.

Amazon ECS 블루/그린 배포 개요

Amazon ECS 블루/그린 배포의 경우 배포 구성은 Amazon ECS 서비스 자체의 일부입니다.

1. 가중치가 1과 0인 두 개의 대상 그룹을 포함하는 규칙을 사용하여 로드 밸런서 프로덕션 리스너를 사전 구성해야 합니다.
2. 다음 리소스를 지정하거나 서비스 리소스를 업데이트해야 합니다.
   - 이 리스너 규칙의 ARN
   - 두 개의 대상 그룹
   - Amazon ECS에 Elastic Load Balancing API를 직접 호출할 권한을 부여하는 IAM 역할
   - Lambda 함수를 실행하기 위한 선택적 IAM 역할
   - `deploymentController` 유형을 `ECS`로 설정하고 `deploymentConfiguration.strategy`를 `BLUE_GREEN`으로 설정합니다. 그러면 태스크가 기본 대상 그룹에 등록된 (블루) 서비스 배포가 생성됩니다.

Amazon ECS 블루/그린을 사용하면 Amazon ECS `UpdateService()`를 직접 호출하고 새 개정의 세부 정보를 전달하여 새 서비스 개정이 생성됩니다. 서비스 배포는 새 (그린) 서비스 개정 태스크를 생성하고 보조 대상 그룹에 등록합니다. Amazon ECS는 리스너 규칙의 가중치를 전환하여 재라우팅 및 롤백 작업을 처리합니다.

주요 구현 차이

두 접근 방식 모두 초기 태스크 세트를 생성하지만 기본 구현은 서로 다릅니다.

- CodeDeploy는 태스크 세트를 사용하지만 Amazon ECS는 서비스 개정을 사용합니다. Amazon ECS 태스크 세트는 Amazon ECS 서비스 개정 및 배포로 대체된 이전 구문입니다. 후자는 배포 프로세스와 서비스 배포 및 서비스 개정 기록에 대한 더 높은 가시성을 제공합니다.
- CodeDeploy를 사용하면 수명 주기 후크가 `CreateDeployment()`에 제공되는 AppSpec 파일의 일부로 지정됩니다. 즉, 후크를 한 배포에서 다음 배포로 변경할 수 있습니다. Amazon ECS 블루/그린에서는 후크가 서비스 구성의 일부로 지정되며 모든 업데이트에 `UpdateService()` 직접 호출이 필요합니다.
- CodeDeploy와 Amazon ECS 블루/그린 모두 후크 구현에 Lambda를 사용하지만 예상되는 입력과 출력은 서로 다릅니다.

  CodeDeploy에서는 함수가 `PutLifecycleEventHookExecutionStatus()`를 직접 호출하여 후크 상태를 반환해야 합니다. 후크 상태는 `SUCCEEDED` 또는 `FAILED`일 수 있습니다. Amazon ECS에서는 Lambda 응답을 사용하여 후크 상태를 나타냅니다.
- CodeDeploy는 각 후크를 일회성 직접 호출로 간접 호출하며 1시간 이내에 최종 실행 상태가 반환될 것으로 예상합니다. Amazon ECS 후크는 `IN_PROGRESS` 표시기를 리턴할 수 있다는 점에서 보다 유연합니다. 이 표시기는 `SUCCEEDED` 또는 `FAILED`가 될 때까지 후크를 반복적으로 다시 간접 호출해야 함을 알립니다. 자세한 내용은 Amazon ECS 서비스 배포에 대한 수명 주기 후크 섹션을 참조하세요.

마이그레이션 접근 방식

CodeDeploy 블루/그린에서 Amazon ECS 블루/그린 배포로 마이그레이션하는 세 가지 기본 접근 방식이 있습니다. 각 접근 방식은 복잡성, 위험, 롤백 기능 및 잠재적 가동 중지 시간 측면에서 서로 다른 특성을 보입니다.

CodeDeploy에 사용된 것과 동일한 Elastic Load Balancing 리소스 재사용

CodeDeploy 배포 컨트롤러 대신 블루/그린 배포 전략으로 Amazon ECS 배포 컨트롤러를 사용하도록 기존 Amazon ECS 서비스를 업데이트합니다. 이 접근 방식을 사용하는 경우 다음 사항을 고려하세요.

- 기존 Amazon ECS 서비스 배포 컨트롤러 및 배포 전략을 업데이트하므로 마이그레이션 절차가 더 간단합니다.
- 올바르게 구성하고 마이그레이션하면 가동 중지 시간이 발생하지 않습니다.
- 롤백하려면 서비스 개정을 되돌려야 합니다.
- 병렬 블루/그린 구성이 없으므로 위험이 높습니다.

CodeDeploy에 사용되는 것과 동일한 로드 밸런서 리스너 및 대상 그룹을 사용합니다. AWS CloudFormation을 사용 중인 경우 AWS CloudFormation CodeDeploy 블루/그린 배포 템플릿을 Amazon ECS 블루/그린 AWS CloudFormation 템플릿으로 마이그레이션 섹션을 참조하세요.

1. 대체 대상 그룹을 포함하도록 프로덕션/테스트 리스너의 기본 규칙을 수정하고 기본 대상 그룹의 가중치를 1로, 대체 대상 그룹을 0으로 설정하세요.

   CodeDeploy의 경우 서비스에 연결된 로드 밸런서의 리스너는 모든 트래픽을 단일 대상 그룹으로 라우팅하는 단일(기본값) 규칙으로 구성됩니다. Amazon ECS 블루/그린의 경우 로드 밸런서 리스너는 가중치가 있는 두 개의 대상 그룹을 포함하는 규칙으로 사전 구성되어야 합니다.
2. `UpdateService` API를 직접 호출하고 파라미터 `deploymentController`를 `ECS`로, 파라미터 `deploymentStrategy`를 `BLUE_GREEN`으로 설정하여 기존 Amazon ECS 서비스를 업데이트하세요. 대상 그룹, 대체 대상 그룹, 프로덕션 리스너 및 선택적 테스트 리스너의 ARN을 지정하세요.
3. 서비스가 예상대로 작동하는지 확인하세요.
4. 이제 Amazon ECS 블루/그린을 사용하고 있으므로 이 Amazon ECS 서비스에 대한 CodeDeploy 설정을 삭제하세요.

기존 로드 밸런서를 사용하는 새 서비스

이 접근 방식은 마이그레이션에 블루/그린 전략을 사용합니다.

이 접근 방식을 사용하는 경우 다음 사항을 고려하세요.

- 중단이 최소화됩니다. Elastic Load Balancing 포트 스왑 중에만 발생합니다.
- 롤백하려면 Elastic Load Balancing 포트 스왑을 되돌려야 합니다.
- 병렬 구성이기 때문에 위험이 낮습니다. 따라서 트래픽 전환 전에 테스트할 수 있습니다.

1. 필요한 경우 이 설정으로 쉽게 롤백할 수 있도록 CodeDeploy 설정에 대한 리스너, 대상 그룹 및 Amazon ECS 서비스를 그대로 두세요.

2. 기존 로드 밸런서에서 새 대상 그룹과 새 리스너(원래 리스너와 다른 포트를 사용함)를 생성하세요. 그런 다음 배포 컨트롤러로 ECS, 배포 전략으로 BLUE_GREEN을 사용하는 경우를 제외하고 기존 Amazon ECS 서비스와 일치하는 새 Amazon ECS 서비스를 생성하고 새 대상 그룹 및 새 리스너 규칙에 대한 ARN을 전달하세요.
3. 서비스에 HTTP 트래픽을 수동으로 전송하여 새 설정을 확인하세요. 모두 잘 진행되면 원래 리스너와 새 리스너의 포트를 바꿔 트래픽을 새 설정으로 라우팅하세요.
4. 새 설정을 확인하고 모든 것이 예상대로 계속 작동하면 CodeDeploy 설정을 삭제하세요.

새 로드 밸런서를 사용하는 새 서비스

이전 접근 방식과 마찬가지로 이 접근 방식은 마이그레이션에 블루/그린 전략을 사용합니다. 주요 차이점은 CodeDeploy 설정에서 Amazon ECS 블루/그린 설정으로 전환이 로드 밸런서 위의 역방향 프록시 계층에서 발생한다는 점입니다. 역방향 프록시 계층의 구현 샘플은 Route 53 및 CloudFront입니다.

이 접근 방식은 이미 이 역방향 프록시 계층이 있고 서비스와의 모든 통신이 이를 통해 수행되는 고객(예: 로드 밸런서 수준에서 직접 통신하지 않음) 적합합니다.

이 접근 방식을 사용하는 경우 다음 사항을 고려하세요.

- 이를 위해서는 역방향 프록시 계층이 필요합니다.
- 기존 Amazon ECS 서비스 배포 컨트롤러 및 배포 전략을 업데이트해야 하므로 마이그레이션 절차가 더 복잡합니다.
- 중단이 최소화됩니다. Elastic Load Balancing 포트 스왑 중에만 발생합니다.
- 롤백하려면 프록시 구성 변경 내용을 되돌려야 합니다.
- 병렬 구성이기 때문에 위험이 낮습니다. 따라서 트래픽 전환 전에 테스트할 수 있습니다.

1. 기존 CodeDeploy 설정을 수정하지 마세요(로드 밸런서, 리스너, 대상 그룹, Amazon ECS 서비스 및 CodeDeploy 배포 그룹).
2. Amazon ECS 블루/그린 배포에 대해 구성된 새 로드 밸런서, 대상 그룹 및 리스너를 생성하세요.

   적절한 리소스를 구성합니다.

   - Application Load Balancer - 자세한 내용은 블루/그린 배포에 대한 Application Load Balancer 리소스 섹션을 참조하세요.
   - Network Load Balancer - 자세한 내용은 Amazon ECS 블루/그린 배포에 대한 Network Load Balancer 리소스 섹션을 참조하세요.

3. ECS를 배포 컨트롤러로 사용하고 BLUE_GREEN을 배포 전략으로 사용하여 새 로드 밸런서 리소스를 가리키는 새 서비스를 생성하세요.
4. 새 로드 밸런서를 통해 테스트하여 새 설정을 확인하세요.
5. 트래픽을 새 로드 밸런서로 라우팅하도록 역방향 프록시 구성을 업데이트하세요.
6. 새 서비스 개정을 관찰하고 모든 것이 예상대로 계속 작동하면 CodeDeploy 설정을 삭제하세요.

다음 단계

Amazon ECS 블루/그린 배포로 마이그레이션한 후:

- CodeDeploy UpdateService API 대신 Amazon ECS CreateDeployment API를 사용하도록 배포 스크립트 및 CI/CD 파이프라인을 업데이트합니다.
- CodeDeploy 배포 대신 Amazon ECS 서비스 배포를 추적하도록 모니터링 및 알림을 업데이트합니다.
- 새 배포 프로세스의 자동화된 테스트를 구현하여 예상대로 작동하는지 확인합니다.

CodeDeploy 블루/그린에서 Amazon ECS 블루/그린 서비스 배포로 마이그레이션

Amazon RDS 블루/그린 배포를 사용하면 프로덕션 환경에서 구현하기 전에 서비스를 변경하고 해당 변경 내용을 테스트할 수 있습니다.

Amazon ECS 블루/그린 배포에 대한 새 수명 주기 후크를 생성해야 합니다.

사전 조건

블루/그린 배포를 시작하기 전에 다음 작업을 수행합니다.

1. Amazon ECS CodeDeploy IAM 역할을 다음 권한으로 바꾸세요.
   - Elastic Load Balancing 권한에 대한 자세한 내용은 로드 밸런서에 대한 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.
   - Lambda 권한에 대한 자세한 내용은 Amazon ECS 블루/그린 배포에서 Lambda 함수에 필요한 권한 섹션을 참조하세요.
2. CodeDeploy 자동화를 끄세요. 자세한 내용은 CodeDeploy 사용 설명서의 Working with deployment groups in CodeDeploy를 참조하세요.
3. CodeDeploy 블루/그린 배포에서 다음 정보가 있는지 확인하세요. Amazon ECS 블루/그린 배포에서 이 정보를 재사용할 수 있습니다.

- 프로덕션 대상 그룹
- 프로덕션 리스너
- 프로덕션 규칙
- 테스트 대상 그룹

  그린 서비스 개정의 대상 그룹입니다.

4. Application Load Balancer 대상 그룹이 리스너 규칙과 올바르게 연결되어 있는지 확인하세요.
   - 테스트 리스너를 사용하지 않는 경우 두 개의 대상 그룹(프로덕션 및 테스트)이 프로덕션 리스너 규칙과 연결되어 있어야 합니다.
   - 테스트 리스너를 사용하는 경우 하나의 대상 그룹을 프로덕션 리스너 규칙에 연결하고 다른 대상 그룹을 테스트 리스너 규칙에 연결해야 합니다.

   이 요구 사항이 충족되지 않으면 서비스 배포에 실패하고 다음 오류가 발생합니다. `Service deployment rolled back because of invalid networking configuration. Both targetGroup and alternateTargetGroup must be associated with the productionListenerRule or testListenerRule.`
5. 서비스에 대해 진행 중인 서비스 배포가 없는지 확인하세요. 자세한 내용은 Amazon ECS 서비스 배포를 사용하여 서비스 기록 보기 섹션을 참조하세요.
6. Amazon ECS 블루/그린 배포에서는 서비스가 다음 기능 중 하나를 사용해야 합니다. 적절한 리소스를 구성하세요.
   - Application Load Balancer - 자세한 내용은 블루/그린 배포에 대한 Application Load Balancer 리소스 섹션을 참조하세요.
   - Network Load Balancer - 자세한 내용은 Amazon ECS 블루/그린 배포에 대한 Network Load Balancer 리소스 섹션을 참조하세요.
   - Service Connect - 자세한 내용은 Amazon ECS 블루/그린 배포에 대한 Service Connect 리소스 섹션을 참조하세요.
7. Amazon ECS 블루/그린 배포의 단계에 대한 수명 주기 단계에서 Lambda 함수를 실행할지를 결정하세요.
   - 사전 스케일 업
   - 스케일 업 이후
   - 트래픽 전환 테스트
   - 트래픽 전환 테스트 이후
   - 프로덕션 트래픽 전환

- 프로덕션 트래픽 전환 이후

각 수명 주기 단계에 대해 Lambda 함수를 생성하세요. 자세한 내용은 AWS Lambda 개발자 안내서의 콘솔로 Lambda 함수 생성을 참조하세요.

서비스의 배포 컨트롤러 업데이트에 대한 자세한 내용은 Amazon ECS 서비스 파라미터 업데이트 섹션을 참조하세요.

절차

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.

   클러스터 세부 정보 페이지가 표시됩니다.
3. 서비스 탭에서 서비스를 선택하세요.

   서비스 세부 정보 페이지가 표시됩니다.
4. 배너에서 배포 컨트롤러 유형 업데이트를 선택하세요.

   배포 컨트롤러 유형 마이그레이션 페이지가 표시됩니다.
5. 새로 만들기를 확장한 후 다음 파라미터를 지정하세요.
   a. 배포 컨트롤러 유형에서 ECS를 선택하세요.
   b. 배포 전략에서 블루/그린을 선택하세요.
   c. 베이크 소요 시간에 블루 및 그린 서비스 개정이 모두 실행되는 시간을 입력하세요.
   d. 수명 주기 단계에서 Lambda 함수를 실행하려면 배포 수명 주기 후크 아래 각 고유한 Lambda 함수에 대해 다음을 수행하세요.
      i. 추가를 선택합니다.

         실행하려는 모든 고유 함수에 대해 반복하세요.
      ii. Lambda 함수에서 함수 이름을 입력하세요.
      iii. 역할에서 블루/그린 권한이 있는 사전 요구 사항에서 생성한 역할을 선택하세요.

         자세한 내용은 Amazon ECS 블루/그린 배포에서 Lambda 함수에 필요한 권한 섹션을 참조하세요.
      iv. 수명 주기 단계에서 Lambda 함수가 실행되는 단계를 선택하세요.

v. (선택 사항) 후크 세부 정보에 후크에 대한 정보를 제공하는 키-값 페어를 입력하세요.

6. 로드 밸런싱을 확장한 후 다음을 구성하세요.

   a. 역할에서 블루/그린 권한이 있는 사전 요구 사항에서 생성한 역할을 선택하세요.

      자세한 내용은 Amazon ECS 블루/그린 배포에서 Lambda 함수에 필요한 권한 섹션을 참조하세요.

   b. 리스너에서 CodeDeploy 블루/그린 배포의 프로덕션 리스너를 선택하세요.

   c. 프로덕션 규칙에서 CodeDeploy 블루/그린 배포의 프로덕션 규칙을 선택하세요.

   d. 테스트 규칙에서 CodeDeploy 블루/그린 배포의 테스트 규칙을 선택하세요.

   e. 대상 그룹에서 CodeDeploy 블루/그린 배포의 프로덕션 대상 그룹을 선택하세요.

   f. 대체 대상 그룹에서 CodeDeploy 블루/그린 배포의 테스트 대상 그룹을 선택하세요.

7. 업데이트를 선택합니다.

다음 단계

- 배포를 시작하도록 서비스를 업데이트합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.
- 배포 프로세스를 모니터링하여 블루/그린 패턴을 따르는지 확인합니다.
  - 그린 서비스 개정이 생성되고 스케일 업됨
  - 테스트 트래픽이 그린 개정으로 라우팅됨(구성된 경우)
  - 프로덕션 트래픽이 그린 개정으로 전환됨
  - 베이크 소요 시간이 지나면 블루 개정이 종료됨

AWS CloudFormation CodeDeploy 블루/그린 배포 템플릿을 Amazon ECS 블루/그린 AWS CloudFormation 템플릿으로 마이그레이션

Amazon ECS 서비스에 대한 CodeDeploy 블루/그린 배포를 사용하는 AWS CloudFormation 템플릿을 네이티브 Amazon ECS 블루/그린 배포 전략을 사용하는 템플릿으로 마이그레이션합니다. 마이그레이션은 'CodeDeploy에 사용된 것과 동일한 Elastic Load Balancing 리소스 재사용' 접근 방식을 따릅니다. 자세한 내용은 the section called “CodeDeploy 블루/그린 배포를 Amazon ECS 블루/그린 배포로 마이그레이션” 설명서를 참조하세요.

소스 템플릿

이 템플릿은 AWS::CodeDeployBlueGreen 변환 및 AWS::CodeDeploy::BlueGreen 후크를 사용하여 Amazon ECS 서비스에 대한 블루/그린 배포를 구현합니다.

소스

CodeDeploy 블루/그린 배포를 사용하는 전체 AWS CloudFormation 템플릿입니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 Blue/green deployment template example을 참조하세요.

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Parameters": {
    "Vpc": {
      "Type": "AWS::EC2::VPC::Id"
    },
    "Subnet1": {
      "Type": "AWS::EC2::Subnet::Id"
    },
    "Subnet2": {
      "Type": "AWS::EC2::Subnet::Id"
    }
  },
  "Transform": [
    "AWS::CodeDeployBlueGreen"
  ],
  "Hooks": {
    "CodeDeployBlueGreenHook": {
      "Type": "AWS::CodeDeploy::BlueGreen",
      "Properties": {
        "TrafficRoutingConfig": {
          "Type": "TimeBasedCanary",
          "TimeBasedCanary": {
            "StepPercentage": 15,
            "BakeTimeMins": 5
          }
        },
        "Applications": [
          {
            "Target": {
              "Type": "AWS::ECS::Service",
              "LogicalID": "ECSDemoService"
            },
            "ECSAttributes": {
```

```
            "TaskDefinitions": [
              "BlueTaskDefinition",
              "GreenTaskDefinition"
            ],
            "TaskSets": [
              "BlueTaskSet",
              "GreenTaskSet"
            ],
            "TrafficRouting": {
              "ProdTrafficRoute": {
                "Type": "AWS::ElasticLoadBalancingV2::Listener",
                "LogicalID": "ALBListenerProdTraffic"
              },
              "TargetGroups": [
                "ALBTargetGroupBlue",
                "ALBTargetGroupGreen"
              ]
            }
          }
        }
      ]
    }
  }
},
"Resources": {
  "ExampleSecurityGroup": {
    "Type": "AWS::EC2::SecurityGroup",
    "Properties": {
      "GroupDescription": "Security group for ec2 access",
      "VpcId": {"Ref": "Vpc"},
      "SecurityGroupIngress": [
        {
          "IpProtocol": "tcp",
          "FromPort": 80,
          "ToPort": 80,
          "CidrIp": "0.0.0.0/0"
        },
        {
          "IpProtocol": "tcp",
          "FromPort": 8080,
          "ToPort": 8080,
          "CidrIp": "0.0.0.0/0"
        },
        {
```

```
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIp": "0.0.0.0/0"
          }
        ]
      }
    },
    "ALBTargetGroupBlue": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ALBTargetGroupGreen": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
```

```
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ExampleALB": {
      "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
      "Properties": {
        "Scheme": "internet-facing",
        "SecurityGroups": [
          {"Ref": "ExampleSecurityGroup"}
        ],
        "Subnets": [
          {"Ref": "Subnet1"},
          {"Ref": "Subnet2"}
        ],
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "Type": "application",
        "IpAddressType": "ipv4"
      }
    },
    "ALBListenerProdTraffic": {
      "Type": "AWS::ElasticLoadBalancingV2::Listener",
      "Properties": {
        "DefaultActions": [
          {
            "Type": "forward",
            "ForwardConfig": {
              "TargetGroups": [
                {
```

```
                "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                "Weight": 1
              }
            ]
          }
        }
      ],
      "LoadBalancerArn": {"Ref": "ExampleALB"},
      "Port": 80,
      "Protocol": "HTTP"
    }
  },
  "ALBListenerProdRule": {
    "Type": "AWS::ElasticLoadBalancingV2::ListenerRule",
    "Properties": {
      "Actions": [
        {
          "Type": "forward",
          "ForwardConfig": {
            "TargetGroups": [
              {
                "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                "Weight": 1
              }
            ]
          }
        }
      ],
      "Conditions": [
        {
          "Field": "http-header",
          "HttpHeaderConfig": {
            "HttpHeaderName": "User-Agent",
            "Values": [
              "Mozilla"
            ]
          }
        }
      ],
      "ListenerArn": {"Ref": "ALBListenerProdTraffic"},
      "Priority": 1
    }
  },
  "ECSTaskExecutionRole": {
```

```
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "",
              "Effect": "Allow",
              "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
        ]
      }
    },
    "BlueTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
        "ContainerDefinitions": [
          {
            "Name": "DemoApp",
            "Image": "nginxdemos/hello:latest",
            "Essential": true,
            "PortMappings": [
              {
                "HostPort": 80,
                "Protocol": "tcp",
                "ContainerPort": 80
              }
            ]
          }
        ],
        "RequiresCompatibilities": [
          "FARGATE"
        ],
        "NetworkMode": "awsvpc",
        "Cpu": "256",
        "Memory": "512",
```

```
        "Family": "ecs-demo"
      }
    },
    "ECSDemoCluster": {
      "Type": "AWS::ECS::Cluster",
      "Properties": {}
    },
    "ECSDemoService": {
      "Type": "AWS::ECS::Service",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "DesiredCount": 1,
        "DeploymentController": {
          "Type": "EXTERNAL"
        }
      }
    },
    "BlueTaskSet": {
      "Type": "AWS::ECS::TaskSet",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "LaunchType": "FARGATE",
        "NetworkConfiguration": {
          "AwsVpcConfiguration": {
            "AssignPublicIp": "ENABLED",
            "SecurityGroups": [
              {"Ref": "ExampleSecurityGroup"}
            ],
            "Subnets": [
              {"Ref": "Subnet1"},
              {"Ref": "Subnet2"}
            ]
          }
        },
        "PlatformVersion": "1.4.0",
        "Scale": {
          "Unit": "PERCENT",
          "Value": 100
        },
        "Service": {"Ref": "ECSDemoService"},
        "TaskDefinition": {"Ref": "BlueTaskDefinition"},
        "LoadBalancers": [
          {
            "ContainerName": "DemoApp",
```

```
            "ContainerPort": 80,
            "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"}
          }
        ]
      }
    },
    "PrimaryTaskSet": {
      "Type": "AWS::ECS::PrimaryTaskSet",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "Service": {"Ref": "ECSDemoService"},
        "TaskSetId": {"Fn::GetAtt": ["BlueTaskSet", "Id"]}
      }
    }
  }
}
```

마이그레이션 단계

CodeDeploy 관련 리소스 제거

다음 속성은 더 이상 필요하지 않습니다.

- `AWS::CodeDeployBlueGreen` 변환
- `CodeDeployBlueGreenHook` 후크
- `GreenTaskDefinition` 및 `GreenTaskSet` 리소스(Amazon ECS에서 관리됨)
- `PrimaryTaskSet` 리소스(Amazon ECS는 태스크 세트를 내부적으로 관리함)

로드 밸런서 리스너 재구성

두 개의 대상 그룹에 전달 작업을 사용하도록 `ALBListenerProdTraffic` 리소스를 수정합니다.

```
{
  "DefaultActions": [
    {
      "Type": "forward",
      "ForwardConfig": {
        "TargetGroups": [
          {
            "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
            "Weight": 1
```

```
          },
          {
            "TargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
            "Weight": 0
          }
        ]
      }
    }
  ]
}
```

배포 속성 업데이트

다음을 업데이트하고 추가합니다.

- DeploymentController 속성을 EXTERNAL에서 ECS로 업데이트합니다.
- Strategy 속성을 추가하고 BLUE_GREEN으로 설정합니다.
- BakeTimeInMinutes 속성을 추가합니다.

```
{
  "DeploymentConfiguration": {
    "MaximumPercent": 200,
    "MinimumHealthyPercent": 100,
    "DeploymentCircuitBreaker": {
      "Enable": true,
      "Rollback": true
    },
    "BakeTimeInMinutes": 5,
    "Strategy": "BLUE_GREEN"
  }
}
```

- 로드 밸런서 구성을 서비스에 추가합니다.

```
{
  "LoadBalancers": [
    {
      "ContainerName": "DemoApp",
      "ContainerPort": 80,
      "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
      "AdvancedConfiguration": {
        "AlternateTargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
```

```
        "ProductionListenerRule": {"Ref": "ALBListenerProdRule"},
        "RoleArn": {"Fn::GetAtt": ["ECSInfrastructureRoleForLoadBalancers", "Arn"]}
      }
    }
  ]
}
```

- 서비스에 태스크 정의 참조를 추가합니다.

```
{
  "TaskDefinition": {"Ref": "BlueTaskDefinition"}
}
```

AmazonECSInfrastructureRolePolicyForLoadBalancers 역할 생성

Amazon ECS가 로드 밸런서 리소스를 관리하도록 허용하는 새 IAM 역할을 추가합니다.

모든 *### ##*을 사용자의 값으로 바꿉니다.

```
{
  "ECSInfrastructureRoleForLoadBalancers": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Sid": "AllowAccessToECSForInfrastructureManagement",
            "Effect": "Allow",
            "Principal": {
              "Service": "ecs.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "ManagedPolicyArns": [
        "arn:aws:iam::aws:policy/AmazonECSInfrastructureRolePolicyForLoadBalancers"
      ]
    }
  }
}
```

테스트 권장 사항

1. 마이그레이션된 템플릿을 비프로덕션 환경에 배포하세요.
2. 서비스가 초기 구성으로 올바르게 배포되는지 확인하세요.
3. 태스크 정의를 업데이트하고 블루/그린 배포 프로세스를 관찰하여 배포를 테스트하세요.
4. 트래픽이 블루 배포와 그린 배포 사이에서 올바르게 전환되는지 확인하세요.
5. 배포 실패를 강제로 적용하여 롤백 기능을 테스트하세요.

마이그레이션 후 템플릿

최종 템플릿

Amazon ECS 블루/그린 배포를 사용하는 전체 AWS CloudFormation 템플릿입니다.

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Parameters": {
    "Vpc": {
      "Type": "AWS::EC2::VPC::Id"
    },
    "Subnet1": {
      "Type": "AWS::EC2::Subnet::Id"
    },
    "Subnet2": {
      "Type": "AWS::EC2::Subnet::Id"
    }
  },
  "Resources": {
    "ExampleSecurityGroup": {
      "Type": "AWS::EC2::SecurityGroup",
      "Properties": {
        "GroupDescription": "Security group for ec2 access",
        "VpcId": {"Ref": "Vpc"},
        "SecurityGroupIngress": [
          {
            "IpProtocol": "tcp",
            "FromPort": 80,
            "ToPort": 80,
            "CidrIp": "0.0.0.0/0"
          },
          {
```

```
            "IpProtocol": "tcp",
            "FromPort": 8080,
            "ToPort": 8080,
            "CidrIp": "0.0.0.0/0"
          },
          {
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIp": "0.0.0.0/0"
          }
        ]
      }
    },
    "ALBTargetGroupBlue": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ALBTargetGroupGreen": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
```

```
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ExampleALB": {
      "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
      "Properties": {
        "Scheme": "internet-facing",
        "SecurityGroups": [
          {"Ref": "ExampleSecurityGroup"}
        ],
        "Subnets": [
          {"Ref": "Subnet1"},
          {"Ref": "Subnet2"}
        ],
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "Type": "application",
        "IpAddressType": "ipv4"
      }
    },
    "ALBListenerProdTraffic": {
      "Type": "AWS::ElasticLoadBalancingV2::Listener",
      "Properties": {
```

```
        "DefaultActions": [
          {
            "Type": "forward",
            "ForwardConfig": {
              "TargetGroups": [
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                  "Weight": 1
                },
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
                  "Weight": 0
                }
              ]
            }
          }
        ],
        "LoadBalancerArn": {"Ref": "ExampleALB"},
        "Port": 80,
        "Protocol": "HTTP"
      }
    },
    "ALBListenerProdRule": {
      "Type": "AWS::ElasticLoadBalancingV2::ListenerRule",
      "Properties": {
        "Actions": [
          {
            "Type": "forward",
            "ForwardConfig": {
              "TargetGroups": [
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                  "Weight": 1
                },
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
                  "Weight": 0
                }
              ]
            }
          }
        ],
        "Conditions": [
          {
```

```
            "Field": "http-header",
            "HttpHeaderConfig": {
              "HttpHeaderName": "User-Agent",
              "Values": [
                "Mozilla"
              ]
            }
          }
        ],
        "ListenerArn": {"Ref": "ALBListenerProdTraffic"},
        "Priority": 1
      }
    },
    "ECSTaskExecutionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "",
              "Effect": "Allow",
              "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
        ]
      }
    },
    "ECSInfrastructureRoleForLoadBalancers": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AllowAccessToECSForInfrastructureManagement",
              "Effect": "Allow",
              "Principal": {
```

```
              "Service": "ecs.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "ManagedPolicyArns": [
        "arn:aws:iam::aws:policy/AmazonECSInfrastructureRolePolicyForLoadBalancers"
      ]
    }
  },
  "BlueTaskDefinition": {
    "Type": "AWS::ECS::TaskDefinition",
    "Properties": {
      "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
      "ContainerDefinitions": [
        {
          "Name": "DemoApp",
          "Image": "nginxdemos/hello:latest",
          "Essential": true,
          "PortMappings": [
            {
              "HostPort": 80,
              "Protocol": "tcp",
              "ContainerPort": 80
            }
          ]
        }
      ],
      "RequiresCompatibilities": [
        "FARGATE"
      ],
      "NetworkMode": "awsvpc",
      "Cpu": "256",
      "Memory": "512",
      "Family": "ecs-demo"
    }
  },
  "ECSDemoCluster": {
    "Type": "AWS::ECS::Cluster",
    "Properties": {}
  },
  "ECSDemoService": {
    "Type": "AWS::ECS::Service",
```

```
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "DesiredCount": 1,
        "DeploymentController": {
          "Type": "ECS"
        },
        "DeploymentConfiguration": {
          "MaximumPercent": 200,
          "MinimumHealthyPercent": 100,
          "DeploymentCircuitBreaker": {
            "Enable": true,
            "Rollback": true
          },
          "BakeTimeInMinutes": 5,
          "Strategy": "BLUE_GREEN"
        },
        "NetworkConfiguration": {
          "AwsvpcConfiguration": {
            "AssignPublicIp": "ENABLED",
            "SecurityGroups": [
              {"Ref": "ExampleSecurityGroup"}
            ],
            "Subnets": [
              {"Ref": "Subnet1"},
              {"Ref": "Subnet2"}
            ]
          }
        },
        "LaunchType": "FARGATE",
        "PlatformVersion": "1.4.0",
        "TaskDefinition": {"Ref": "BlueTaskDefinition"},
        "LoadBalancers": [
          {
            "ContainerName": "DemoApp",
            "ContainerPort": 80,
            "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
            "AdvancedConfiguration": {
              "AlternateTargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
              "ProductionListenerRule": {"Ref": "ALBListenerProdRule"},
              "RoleArn": {"Fn::GetAtt": ["ECSInfrastructureRoleForLoadBalancers",
"Arn"]}
            }
          }
        ]
```

```
      }
    }
  }
}
```

CodeDeploy 블루/그린에서 Amazon ECS 롤링 업데이트 서비스 배포로 마이그레이션

CodeDeploy 블루/그린 배포에서 Amazon ECS 롤링 업데이트 배포로 서비스 배포를 마이그레이션할 수 있습니다. 이렇게 하면 CodeDeploy 종속성을 벗어나 통합 배포를 사용할 수 있습니다.

Amazon ECS 서비스 스케줄러는 현재 실행 중인 태스크를 새 태스크로 대체합니다. 롤링 업데이트 중에 Amazon ECS가 서비스에 추가하거나 서비스에서 제거하는 작업의 수는 서비스 배포 구성으로 제어합니다.

사전 조건

블루/그린 배포를 시작하기 전에 다음 작업을 수행합니다.

1. 더 이상 Amazon ECS CodeDeploy IAM 역할이 필요하지 않습니다.
2. CodeDeploy 자동화를 끄세요. 자세한 내용은 CodeDeploy 사용 설명서의 Working with deployment groups in CodeDeploy를 참조하세요.
3. 서비스에 대해 진행 중인 서비스 배포가 없는지 확인하세요. 자세한 내용은 Amazon ECS 서비스 배포를 사용하여 서비스 기록 보기 섹션을 참조하세요.

서비스의 배포 컨트롤러 업데이트에 대한 자세한 내용은 Amazon ECS 서비스 파라미터 업데이트 섹션을 참조하세요.

절차

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.

   클러스터 세부 정보 페이지가 표시됩니다.
3. 서비스 탭에서 서비스를 선택하세요.

   서비스 세부 정보 페이지가 표시됩니다.
4. 배너에서 마이그레이션을 선택하세요.

   배포 구성 업데이트 페이지가 표시됩니다.
5. 배포 옵션을 확장한 후 다음 파라미터를 지정합니다.

a. 배포 컨트롤러 유형에서 ECS를 선택하세요.

b. 배포 전략에서 롤링 업데이트를 선택하세요.

c. 최소 실행 작업(Min running tasks)의 경우, 배포 시 RUNNING 상태를 유지해야 하는 서비스 내 작업 수에 대한 하한을 원하는 작업 수의 백분율(가장 가까운 정수로 올림)로 입력합니다. 자세한 내용은 배포 구성을 참조하세요.

d. 최대 실행 작업(Max running tasks)의 경우, 배포 시 RUNNING 또는 PENDING 상태가 허용되는 서비스 내 작업 수에 대한 상한을 원하는 작업 수의 백분율(가장 가까운 정수로 내림)로 입력합니다.

6. 로드 밸런싱을 확장한 후 다음을 구성하세요.

a. 역할에서 블루/그린 권한이 있는 사전 요구 사항에서 생성한 역할을 선택하세요.

자세한 내용은 Amazon ECS 블루/그린 배포에서 Lambda 함수에 필요한 권한 섹션을 참조하세요.

b. 리스너에서 CodeDeploy 블루/그린 배포의 프로덕션 리스너를 선택하세요.

c. 대상 그룹에서 CodeDeploy 블루/그린 배포의 프로덕션 대상 그룹을 선택하세요.

7. 업데이트를 선택합니다.

다음 단계

변경 내용을 적용하려면 서비스를 업데이트해야 합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.

Amazon ECS 배포 전략 업데이트

Amazon ECS는 서비스 업데이트에 대한 여러 배포 전략을 지원합니다. 애플리케이션 요구 사항에 따라 이러한 전략 사이에서 마이그레이션할 수 있습니다. 이 주제에서는 롤링 배포와 블루/그린 배포 사이에서 마이그레이션하는 방법에 대해 설명합니다.

Amazon ECS 배포 전략 이해

배포 전략 사이에서 마이그레이션하기 전에 각 전략의 작동 방식과 주요 차이점을 이해하는 것이 중요합니다.

롤링 배포

롤링 배포에서 Amazon ECS는 현재 실행 중인 애플리케이션 버전을 새 버전으로 바꿉니다. 서비스 스케줄러는 최소 및 최대 정상 상태 백분율 파라미터를 사용하여 배포 전략을 결정합니다.

롤링 배포는 설정이 더 간단하지만 배포 프로세스 및 트래픽 라우팅에 대한 제어는 덜합니다.

블루/그린 배포

블루/그린 배포에서 Amazon ECS는 기존 버전(블루)과 함께 새 서비스 버전(그린)을 생성합니다. 이렇게 하면 프로덕션 트래픽을 라우팅하기 전에 새 버전을 확인할 수 있습니다.

블루/그린 배포는 트래픽 전환, 테스트 및 롤백 기능을 포함하여 배포 프로세스에 대한 더 많은 제어를 제공합니다.

모범 사례

배포 전략 사이에서 마이그레이션하는 경우 다음 모범 사례를 따릅니다.

- 비프로덕션 환경에서 테스트: 프로덕션 서비스에 변경 내용을 적용하기 전에 항상 비프로덕션 환경에서 업데이트를 테스트합니다.
- 롤백 계획: 업데이트가 예상대로 작동하지 않는 경우에 대비하여 롤백 계획을 수립합니다.
- 전환 중 모니터링: 마이그레이션 도중 및 마이그레이션 후 서비스를 면밀히 모니터링하여 서비스가 계속 올바르게 작동하는지 확인합니다.
- 설명서 업데이트: 새 배포 전략을 반영하도록 배포 설명서를 업데이트합니다.
- 트래픽 영향 고려: 업데이트가 서비스 트래픽에 미치는 영향을 이해하고 적절히 계획합니다.

롤링 업데이트에서 Amazon ECS 블루/그린으로 배포 전략 업데이트

프로덕션 환경에서 구현하기 전에 서비스를 변경하고 해당 변경 내용을 테스트하려는 경우 롤링 업데이트 배포에서 Amazon ECS 블루/그린 배포로 마이그레이션할 수 있습니다.

사전 조건

서비스를 롤링에서 블루/그린 배포로 마이그레이션하기 전에 다음 사항을 갖추어야 합니다.

- 현재 배포가 완료될 때까지 기다립니다.
- 롤링 배포 전략을 사용하는 기존 Amazon ECS 서비스.
- 트래픽을 지원하는 서비스 개정이 여러 개 있는 경우 Amazon ECS는 마이그레이션 중에 트래픽을 단일 개정으로 통합하려고 시도합니다. 실패하면 마이그레이션하기 전에 단일 개정을 사용하도록 서비스를 수동으로 업데이트해야 할 수 있습니다.
- 적절한 권한을 구성하세요.

  - Elastic Load Balancing 권한에 대한 자세한 내용은 로드 밸런서에 대한 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.
  - Lambda 권한에 대한 자세한 내용은 Amazon ECS 블루/그린 배포에서 Lambda 함수에 필요한 권한 섹션을 참조하세요.
- 구성에 따라 다음 중 하나를 수행해야 합니다.
  - 서비스가 Elastic Load Balancing을 사용하는 경우 새 'advancedConfiguration'으로 서비스를 업데이트하고 롤링 배포를 시작합니다.
  - 서비스가 Service Connect를 사용하는 경우 서비스를 업데이트하고 롤링 배포를 시작합니다.
  - 서비스가 Elastic Load Balancing 및 Service Connect를 모두 사용하는 경우 위의 두 단계를 모두 수행합니다(단일 UpdateService 요청 사용 가능).
  - 서비스가 위의 항목을 사용하지 않는 경우 추가 작업이 필요하지 않습니다.
- Amazon ECS 블루/그린 배포에서는 서비스가 다음 기능 중 하나를 사용해야 합니다. 적절한 리소스를 구성합니다.
  - Application Load Balancer - 자세한 내용은 블루/그린 배포에 대한 Application Load Balancer 리소스 섹션을 참조하세요.
  - Network Load Balancer - 자세한 내용은 Amazon ECS 블루/그린 배포에 대한 Network Load Balancer 리소스 섹션을 참조하세요.
  - Service Connect - 자세한 내용은 Amazon ECS 블루/그린 배포에 대한 Service Connect 리소스 섹션을 참조하세요.

절차

1. https://console.aws.amazon.com/ecs/v2에서 Amazon ECS 콘솔을 여세요.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 페이지에서 마이그레이션하려는 서비스가 포함된 클러스터를 선택하세요.

   클러스터 세부 정보 페이지가 표시됩니다.
4. 클러스터 세부 정보 페이지에서 서비스 탭을 선택하세요.
5. 서비스를 선택하고 업데이트를 선택하세요.

   서비스 업데이트 페이지가 표시됨
6. 배포 옵션을 확장한 후 다음을 수행하세요.
7. 배포 전략에서 블루/그린을 선택하세요.
8. 다음과 같이 블루/그린 배포 설정을 구성하세요.

a. 베이크 소요 시간에 블루 개정이 종료되기 전에 블루 및 그린 서비스 개정이 동시에 실행되는 시간(분)을 입력하세요.

   이를 통해 확인 및 테스트 시간을 확보할 수 있습니다.

b. (선택 사항) 배포의 특정 단계에서 실행할 Lambda 함수를 구성하세요. 배포 수명 주기 후크에서 다음 단계에 맞게 Lambda 함수를 구성하세요.

   - 사전 스케일 업: 그린 서비스 개정을 스케일 업하기 전에 실행
   - 사후 스케일 업: 그린 서비스 개정을 스케일 업한 후 실행
   - 테스트 트래픽 전환: 그린 서비스 개정으로 트래픽 라우팅을 테스트하는 동안 실행
   - 사후 테스트 트래픽 전환: 테스트 트래픽이 그린 서비스 개정으로 라우팅된 후 실행
   - 프로덕션 트래픽 전환: 그린 서비스 개정으로 프로덕션 트래픽을 라우팅하는 동안 실행
   - 사후 프로덕션 트래픽 전환: 프로덕션 트래픽이 그린 서비스 개정으로 라우팅된 후 실행

   수명 주기 후크를 추가하는 방법:

   a. 추가를 선택합니다.
   b. Lambda 함수에서 함수 이름 또는 ARN을 입력하세요.
   c. 역할에서 Lambda 함수를 간접 호출할 권한이 있는 IAM 역할을 선택하세요.
   d. 수명 주기 단계에서 Lambda 함수를 실행해야 하는 단계를 선택하세요.
   e. 선택 사항: 후크 세부 정보에 후크에 대한 추가 정보를 제공하도록 키-값 페어를 입력하세요.

9. 다음과 같이 로드 밸런서를 구성하세요.

   a. 로드 밸런싱에서 서비스가 로드 밸런서를 사용하도록 구성되어 있는지 확인하세요.
   b. 대상 그룹에서 프로덕션(블루) 환경의 기본 대상 그룹을 선택하세요.
   c. 대체 대상 그룹에서 테스트(그린) 환경의 대상 그룹을 선택하세요.
   d. 프로덕션 리스너 규칙에서 프로덕션 트래픽을 라우팅하기 위한 리스너 규칙을 선택하세요.
   e. 선택 사항: 테스트 리스너 규칙에서 테스트 트래픽을 그린 환경으로 라우팅하기 위한 리스너 규칙을 선택하세요.
   f. 역할에서 Amazon ECS가 로드 밸런서를 관리하도록 허용하는 IAM 역할을 선택하세요.

10. 변경 내용을 검토한 다음 업데이트를 선택하세요.

다음 단계

- 배포를 시작하도록 서비스를 업데이트합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.
- 배포 프로세스를 모니터링하여 블루/그린 패턴을 따르는지 확인합니다.
  - 그린 서비스 개정이 생성되고 스케일 업됨
  - 테스트 트래픽이 그린 개정으로 라우팅됨(구성된 경우)
  - 프로덕션 트래픽이 그린 개정으로 전환됨
  - 베이크 소요 시간이 지나면 블루 개정이 종료됨

Amazon ECS 블루/그린에서 롤링 업데이트로 배포 전략 업데이트

블루/그린 배포를 롤링 업데이트 배포로 마이그레이션할 수 있습니다.

롤링 배포로 마이그레이션하는 경우 다음 사항을 고려하세요.

- 트래픽 처리: 롤링 배포를 사용하면 상태 확인을 통과하는 즉시 새 태스크가 트래픽을 수신하기 시작합니다. 블루/그린 배포와 마찬가지로 별도의 테스트 단계는 없습니다.
- 리소스 효율성: 롤링 배포는 완전한 중복 환경을 생성하는 대신 태스크를 점진적으로 대체하기 때문에 일반적으로 블루/그린 배포보다 적은 리소스를 사용합니다.
- 롤백 복잡성: 롤백 배포에서는 블루/그린 배포에 비해 롤백 작업이 더 복잡합니다. 롤백해야 하는 경우 이전 태스크 정의를 사용하여 새 배포를 시작해야 합니다.
- 배포 속도: 롤링 배포는 특히 태스크가 많은 서비스의 경우 블루/그린 배포보다 완료하는 데 더 오래 걸릴 수 있습니다.
- 로드 밸런서 구성: 기존 로드 밸런서 구성은 롤링 배포에서 계속 작동하지만 트래픽 전환 동작은 서로 다릅니다.

사전 조건

블루/그린에서 롤링 배포로 서비스를 마이그레이션하기 전에 다음 사항을 갖추어야 합니다.

- 블루/그린 배포 전략을 사용하는 기존 Amazon ECS 서비스
- 서비스에 대한 진행 중인 배포 없음(현재 배포가 완료될 때까지 대기)
- 롤링 배포에서 서비스가 동작하는 방식에 대한 명확한 이해

> **Note**
>
> 배포가 진행 중인 경우 서비스를 롤링 배포로 마이그레이션할 수 없습니다. 계속하기 전에 현재 배포가 완료될 때까지 기다립니다.

마이그레이션 절차

다음 단계에 따라 Amazon ECS 서비스를 블루/그린에서 롤링 배포로 마이그레이션합니다.

1. https://console.aws.amazon.com/ecs/v2에서 Amazon ECS 콘솔을 여세요.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 페이지에서 마이그레이션하려는 서비스가 포함된 클러스터를 선택하세요.
4. 클러스터 세부 정보 페이지에서 서비스 탭을 선택하세요.
5. 마이그레이션할 서비스를 선택하고 업데이트를 선택하세요.
6. 서비스 업데이트 페이지에서 배포 옵션 섹션으로 이동하여 필요한 경우 확장하세요.
7. 배포 전략에서 롤링 업데이트를 선택하세요.
8. 롤링 배포 설정을 구성하세요.
    a. 최소 정상 상태 백분율에서 배포 도중 `RUNNING` 상태를 유지해야 하는 태스크의 최소 백분율을 입력하세요. 이 값은 서비스에 대해 원하는 태스크 수의 백분율로 지정됩니다.
    b. 최대 백분율에서 배포 도중 `RUNNING` 또는 `PENDING` 상태로 허용되는 태스크의 최대 백분율을 입력하세요. 이 값은 서비스에 대해 원하는 태스크 수의 백분율로 지정됩니다.
9. 선택 사항: 배포 실패 감지에서 Amazon ECS가 배포 실패를 감지하고 처리하는 방법을 구성하세요.
    a. 배포 회로 차단기를 사용하려면 Amazon ECS 배포 회로 차단기 사용을 선택하세요.
    b. 실패한 배포를 자동으로 롤백하려면 실패 시 롤백을 선택하세요.
10. 구성 변경 내용을 검토한 다음 업데이트를 선택하여 변경 내용을 저장하고 서비스를 롤링 배포로 마이그레이션하세요.

Amazon ECS는 롤링 배포 전략을 사용하도록 서비스 구성을 업데이트합니다. 다음에 서비스를 업데이트할 때 롤링 배포 프로세스를 사용합니다.

> **Note**
>
> 블루/그린에서 롤링 배포로 마이그레이션하는 경우 Amazon ECS는 다음을 통해 전환을 처리합니다.
>
> 1. 트래픽을 처리하는 현재 활성 서비스 개정 식별.
> 2. 기존 로드 밸런서 구성을 유지하면서 새 배포 처리 방법 변경.
> 3. 향후 롤링 배포를 위해 서비스 준비.

다음 단계

- 배포를 시작하도록 서비스를 업데이트합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.

Amazon ECS 배포 전략 업데이트 문제 해결

이 섹션에서는 배포 전략을 마이그레이션하는 경우 발생할 수 있는 일반적인 문제에 대한 솔루션을 제공합니다.

여러 서비스 개정 또는 태스크 세트

다음 문제는 배포에 대한 여러 서비스 개정이 있는 상황과 관련이 있습니다.

ECS 배포 컨트롤러 업데이트하는 경우 여러 태스크 세트

오류 메시지: `Updating the deployment controller is not supported when there are multiple tasksets in the service. Please ensure your service has only one taskset and try again.`

솔루션: 이 오류는 활성 태스크 세트가 여러 개인 서비스의 배포 컨트롤러 유형을 변경하려는 경우 발생합니다. `CODE_DEPLOY` 또는 `EXTERNAL` 배포 컨트롤러에서 이 문제를 해결하는 방법:

1. 현재 태스크 세트를 확인하세요.

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].taskSets"
```

2. 진행 중인 배포가 완료될 때까지 기다리세요.
3. 새 배포를 강제로 실행하여 태스크 세트를 정리하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --force-new-deployment
```

4. 필요한 경우 추가 태스크 세트를 수동으로 삭제하세요.

```
aws ecs delete-task-set --cluster your-cluster-name --service your-service-name
 --task-set task-set-id
```

5. 태스크 세트가 하나만 남게 되면 배포 컨트롤러 업데이트를 다시 시도하세요.

자세한 내용은 the section called “서비스 배포 컨트롤러 및 전략” 섹션을 참조하세요.

ECS 배포 컨트롤러 업데이트 시 기본 태스크 세트 누락

오류 메시지: `Updating the deployment controller requires a primary taskset in the service. Please ensure your service has a primary taskset and try again.`

솔루션: 이 오류는 기본 태스크 세트가 없는 서비스의 배포 컨트롤러 유형을 변경하려는 경우 발생합니다. 이 문제를 해결하려면:

1. 서비스 상태 및 태스크 세트를 확인하세요. 서비스에 태스크 세트가 있으면 ACTIVE로 표시됩니다.

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].taskSets[*].[status,id]
```

ACTIVE 상태의 태스크 세트가 없으면 배포를 마이그레이션하세요. 자세한 내용은 마이그레이션 접근 방식을 참조하세요.

2. 서비스에 실행 중인 태스크가 없으면 서비스를 업데이트하여 하나 이상의 태스크를 배포하세요.

```
aws ecs update-service-primary-task-set --cluster your-cluster-name --
service your-service-name --primary-task-set your-taskset-id
```

그러면 서비스에서 이전에 ACTIVE 상태였던 태스크 세트를 PRIMARY 상태로 표시합니다.

3. 태스크가 안정적인 실행 상태에 도달할 때까지 기다리세요. 다음을 통해 상태를 확인할 수 있습니다.

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].deployments"
```

4. 서비스에 실행 중인 태스크가 포함된 기본 태스크 세트가 있으면 배포 컨트롤러 업데이트를 다시 시도하세요.

자세한 내용은 the section called “서비스 배포 컨트롤러 및 전략” 섹션을 참조하세요.

배포 실패 감지 유형과 배포 컨트롤러 간 불일치

다음 문제는 배포 실패 감지 유형과 배포 컨트롤러 간 불일치와 관련이 있습니다.

비ECS 컨트롤러와 배포 회로 차단기

오류 메시지: `Deployment circuit breaker feature is only supported with ECS deployment controller. Update to ECS deployment controller and try again.`

솔루션: 이 오류는 ECS 배포 컨트롤러를 사용하지 않는 서비스에서 배포 회로 차단기 기능을 활성화하려는 경우 발생합니다. 배포 회로 차단기는 ECS 배포 컨트롤러와만 호환됩니다.

1. 서비스의 현재 배포 컨트롤러를 확인하세요.

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].deploymentController"
```

2. ECS 배포 컨트롤러를 사용하도록 서비스를 업데이트하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

3. 서비스가 ECS 배포 컨트롤러를 사용하면 배포 회로 차단기를 활성화하세요.

```
aws ecs update-service --cluster your-cluster-name --
service your-service-name --deployment-configuration
 "deploymentCircuitBreaker={enable=true,rollback=true}"
```

자세한 내용은 the section called “ 배포 회로 차단기가 장애를 감지하는 방법” 섹션을 참조하세요.

비ECS 컨트롤러와 경보 기반 롤백

오류 메시지: `Alarm based rollback feature is only supported with ECS deployment controller. Update to ECS deployment controller and try again.`

솔루션: 이 오류는 ECS 배포 컨트롤러를 사용하지 않는 서비스에서 경보 기반 롤백을 구성하려는 경우 발생합니다. 경보 기반 롤백 기능은 ECS 배포 컨트롤러와만 호환됩니다.

1. 서비스의 현재 배포 컨트롤러를 확인하세요.

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].deploymentController"
```

2. ECS 배포 컨트롤러를 사용하도록 서비스를 업데이트하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

3. 서비스가 ECS 배포 컨트롤러를 사용하면 경보 기반 롤백을 구성하세요.

```
aws ecs update-service --cluster your-cluster-name --services your-service-name --deployment-configuration "alarms={alarmNames=[your-alarm-name],enable=true,rollback=true}"
```

자세한 내용은 the section called “CloudWatch 경보가 배포 장애를 감지하는 방법” 섹션을 참조하세요.

Service Connect와 배포 컨트롤러 간 불일치

다음 문제는 Service Connect와 배포 컨트롤러 간 불일치와 관련이 있습니다.

Service Connect에서 EXTERNAL 컨트롤러

오류 메시지: `The EXTERNAL deployment controller type is not supported for services using Service Connect.`

솔루션: 이 오류는 Service Connect가 활성화된 서비스에서 EXTERNAL 배포 컨트롤러를 사용하려는 경우 발생합니다. EXTERNAL 컨트롤러는 Service Connect와 호환되지 않습니다.

1. 서비스에 Service Connect가 활성화되어 있는지 확인하세요.

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].serviceConnectConfiguration"
```

2. EXTERNAL 배포 컨트롤러를 사용해야 하는 경우 서비스를 업데이트하여 Service Connect를 비활성화하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --service-connect-configuration "{}"
```

3. 또는 Service Connect를 사용해야 하는 경우 ECS 배포 컨트롤러를 대신 사용하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

자세한 내용은 the section called “서비스 배포 컨트롤러 및 전략” 섹션을 참조하세요.

비ECS 컨트롤러와 Service Connect

오류 메시지: `Service Connect feature is only supported with ECS (rolling update) deployment controller. Update to ECS deployment controller and try again.`

솔루션: 이 오류는 ECS 배포 컨트롤러를 사용하지 않는 경우에 Service Connect를 활성화하려는 경우 발생합니다. Service Connect 기능은 ECS 배포 컨트롤러와만 호환됩니다.

1. 서비스의 현재 배포 컨트롤러를 확인하세요.

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].deploymentController"
```

2. ECS 배포 컨트롤러를 사용하도록 서비스를 업데이트하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

3. 서비스가 ECS 배포 컨트롤러를 사용하면 Service Connect를 활성화하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --service-connect-configuration "enabled=true,namespace=your-namespace"
```

자세한 내용은 the section called “서비스 배포 컨트롤러 및 전략” 섹션을 참조하세요.

컨트롤러 유형과 예약 전략 간 불일치

다음 문제는 컨트롤러 유형과 예약 전략 간 불일치와 관련이 있습니다.

DAEMON 예약 전략과 CODE_DEPLOY 컨트롤러

오류 메시지: `The CODE_DEPLOY deployment controller type is not supported for services using the DAEMON scheduling strategy.`

솔루션: 이 오류는 DAEMON 예약 전략을 사용하는 서비스에서 CODE_DEPLOY 배포 컨트롤러를 사용하려는 경우 발생합니다. CODE_DEPLOY 컨트롤러는 REPLICA 예약 전략과만 호환됩니다.

1. 서비스의 현재 예약 전략을 확인하세요.

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].schedulingStrategy"
```

2. 블루/그린 배포가 필요한 경우 REPLICA 예약 전략을 사용하도록 서비스를 변경하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --scheduling-strategy REPLICA
```

3. 또는 DAEMON 예약 전략을 사용해야 하는 경우 ECS 배포 컨트롤러를 대신 사용하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

자세한 내용은 the section called “서비스 배포 컨트롤러 및 전략” 섹션을 참조하세요.

DAEMON 예약 전략과 EXTERNAL 컨트롤러

오류 메시지: `The EXTERNAL deployment controller type is not supported for services using the DAEMON scheduling strategy.`

솔루션: 이 오류는 DAEMON 예약 전략을 사용하는 ECS 서비스에서 EXTERNAL 배포 컨트롤러를 사용하려는 경우 발생합니다. EXTERNAL 컨트롤러는 REPLICA 예약 전략과만 호환됩니다.

1. 서비스의 현재 예약 전략을 확인하세요.

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].schedulingStrategy"
```

2. EXTERNAL 배포 컨트롤러를 사용해야 하는 경우 REPLICA 예약 전략을 사용하도록 서비스를 변경하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --scheduling-strategy REPLICA
```

3. 또는 DAEMON 예약 전략을 사용해야 하는 경우 ECS 배포 컨트롤러를 대신 사용하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

자세한 내용은 the section called “서비스 배포 컨트롤러 및 전략” 섹션을 참조하세요.

외부 시작 유형과 서비스 레지스트리

오류 메시지: `Service registries are not supported for external launch type.`

솔루션: 이 오류는 EXTERNAL 시작 유형을 사용하는 서비스에서 서비스 검색(서비스 레지스트리)을 구성하려는 경우 발생합니다. 서비스 검색은 EXTERNAL 시작 유형과 호환되지 않습니다.

1. 서비스의 현재 시작 유형을 확인하세요.

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].launchType"
```

2. 서비스 검색이 필요한 경우 EC2 또는 FARGATE 시작 유형을 사용하도록 서비스를 변경하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --launch-type FARGATE
```

3. 또는 EXTERNAL 시작 유형을 사용해야 하는 경우 서비스 레지스트리 구성을 제거하세요.

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --service-registries "[]"
```

자세한 내용은 the section called “서비스 배포 컨트롤러 및 전략” 섹션을 참조하세요.

배포 컨트롤러 업데이트 되돌리기

이전 배포 컨트롤러로 돌아가려는 경우 다음 중 하나를 수행할 수 있습니다.

- AWS CloudFormation를 사용하는 경우 이전 템플릿을 사용하여 새 스택을 생성할 수 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서에서 스택 생성을 참조하세요.
- Amazon ECS 콘솔 또는 AWS CLI를 사용한 경우 서비스를 업데이트할 수 있습니다. 자세한 내용은 the section called “서비스 업데이트하기” 섹션을 참조하세요.

  update-service 명령을 사용하는 경우 `--deployment-controller` 옵션을 사용하고 이전 배포 컨트롤러로 설정합니다.

Amazon ECS 서비스 배포를 사용하여 서비스 기록 보기

서비스 배포에서는 배포에 대한 포괄적인 보기가 제공됩니다. 서비스 배포에서는 서비스에 대한 다음과 같은 정보가 제공됩니다.

- 현재 배포된 워크로드 구성(소스 서비스 개정)
- 배포 중인 워크로드 구성(대상 서비스 개정)
- 배포 상태
- 회로 차단이 감지된 실패한 태스크 수
- 경보에 있는 CloudWatch 경보
- 서비스 배포 시작 및 완료 시점
- 롤백 발생 시 세부 정보

서비스 배포 속성에 대한 내용은 Amazon ECS 서비스 배포에 포함된 속성 단원을 참조하세요.

서비스 배포는 읽기 전용이며 각각 고유한 ID가 있습니다.

세 가지 서비스 배포 스테이지가 있습니다.

단계	정의	연결된 상태
보류중	서비스 배포가 생성되었으나 시작되지 않았음	PENDING
지속적	서비스 배포 진행 중	• IN_PROGRESS

단계	정의	연결된 상태
		• STOP_REQUESTED • ROLLBACK_REQUESTED • ROLLBACK_IN_PROGRESS
완료됨	서비스 배포 마침(성공 또는 실패)	• SUCCESSFUL • STOPPED • ROLLBACK_SUCCESSFUL • ROLLBACK_FAILED

서비스 배포를 사용하여 서비스의 수명 주기를 이해하고 취해야 할 조치가 있는지 결정합니다. 예를 들면, 롤백이 발생한 경우 서비스 배포를 조사하고 서비스 이벤트를 살펴보는 것이 좋습니다.

콘솔, API 및 AWS CLI를 사용하여 2024년 10월 25일 이후에 생성된 배포에 대한 최근 90일 기록을 볼 수 있습니다.

완료되지 않은 배포를 중지할 수 있습니다. 자세한 내용은 Amazon ECS 서비스 배포 중지 섹션을 참조하세요.

서비스 배포 수명 주기

Amazon ECS에서는 다음과 같은 조치 중 하나라도 발생하면 자동으로 새 서비스 배포가 생성됩니다.

- 사용자가 서비스를 생성합니다.
- 사용자가 서비스를 업데이트하고 새 배포 강제 적용 옵션을 사용합니다.
- 배포가 필요한 하나 이상의 서비스 속성을 사용자가 업데이트합니다.

배포가 진행되는 동안 Amazon ECS에서는 서비스 배포의 진행률이 반영되도록 다음과 같은 서비스 배포 속성이 업데이트됩니다.

- 상태
- 실행 중인 태스크 수

서비스 개정에 표시된 실행 중인 태스크 수는 실행 중인 태스크의 실제 수와 같지 않을 수 있습니다. 이 숫자는 배포 완료 시 실행 중인 태스크 수를 나타냅니다. 예를 들면, 서비스 배포와 독립적으로 태스크를 시작한 경우 해당 태스크는 서비스 개정의 실행 중인 태스크 수에 포함되지 않습니다.

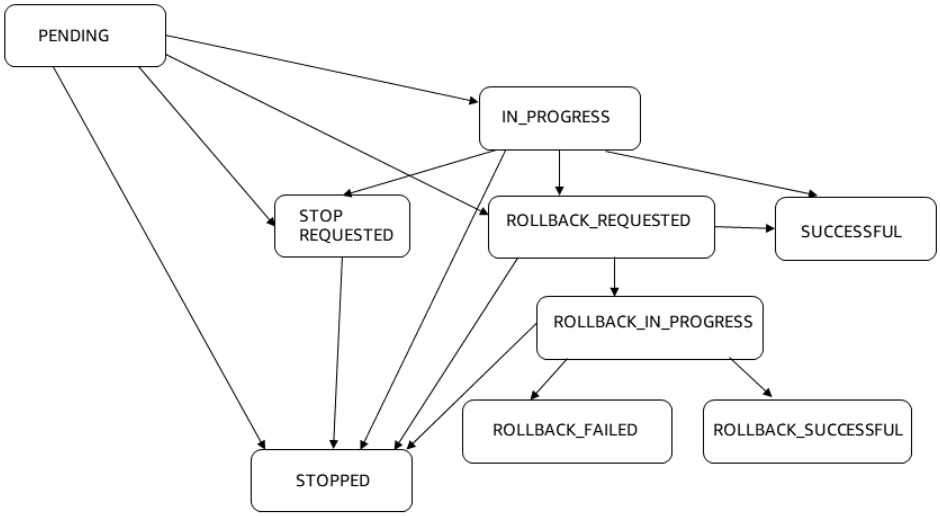
- 회로 차단기 실패 탐지:
  - 시작에 실패한 태스크 수
- CloudWatch 경보 실패 탐지
  - 활성 상태인 경보
- 롤백 정보:
  - 시작 시각
  - 롤백 사유
  - 롤백에 사용된 서비스 개정의 ARN
- 상태 사유

서비스를 삭제할 때 Amazon ECS에서 서비스 배포가 삭제됩니다.

서비스 배포 상태

서비스 배포는 PENDING 상태로 시작됩니다.

다음 그림에서는 PENDING 상태 이후에 발생할 수 있는 서비스 배포 상태인 IN_PROGRESS, ROLLBACK_REQUESTED, SUCCESSFUL, STOP_REQUESTED, ROLLBACK_IN_PROGRESSS, ROLLBACK_FAILED, ROLLBACK_SUCCESSFUL, STOPPED를 보여줍니다.



다음과 같은 정보에서는 서비스 배포 상태에 대한 세부 정보가 제공됩니다.

- PENDING - 서비스 배포가 생성되었으나 시작되지 않았습니다.

  상태가 IN_PROGRESS, ROLLBACK_REQUESTED, STOP_REQUESTED 또는 STOPPED로 바뀔 수 있습니다.

- IN_PROGRESS - 서비스 배포가 진행 중입니다.

  상태가 SUCCESSFUL, STOP_REQUESTED, ROLLBACK_REQUESTED, ROLLBACK_IN_PROGRESS 및 STOPPED로 바뀔 수 있습니다.

- STOP_REQUESTED - 다음 중 하나가 발생하면 서비스 배포 상태가 STOP_REQUESTED로 바뀝니다.
  - 사용자가 새 서비스 배포를 시작합니다.
  - 롤백 옵션은 실패 탐지 메커니즘(회로 차단기 또는 경보 기반)에 사용되지 않으며 서비스가 SUCCESSFUL 상태에 도달하지 않습니다.

  상태가 STOPPED로 바뀝니다.

- ROLLBACK_REQUESTED - 사용자가 콘솔, API 또는 CLI를 통해 롤백을 요청하면 서비스 배포 상태가 ROLLBACK_REQUESTED로 바뀝니다.

상태가 SUCCESSFUL, ROLLBACK_IN_PROGRESS 및 STOPPED로 바뀔 수 있습니다.

- SUCCESSFUL - 서비스 배포가 완료되면 서비스 배포 상태가 SUCCESSFUL로 바뀝니다.
- ROLLBACK_IN_PROGRESS - 롤백 옵션이 실패 탐지 메커니즘(회로 차단기 또는 경보 기반)에 사용 중이면 서비스 배포 상태가 ROLLBACK_IN_PROGRESS로 바뀌며 서비스에 실패합니다.

  상태가 ROLLBACK_SUCCESSFUL 또는 ROLLBACK_FAILED로 바뀝니다.

Amazon ECS 서비스 배포에 포함된 속성

다음과 같은 속성이 서비스 배포에 포함되어 있습니다.

속성	설명
서비스 배포 ARN	서비스 배포의 ARN입니다.
서비스 ARN	이 서비스 배포에 대한 서비스의 ARN입니다.
클러스터 ARN	서비스가 호스팅되는 클러스터의 ARN입니다.
서비스 배포 생성 시각	서비스 배포가 생성된 시각입니다.
서비스 배포 시작 시각	서비스 배포가 시작된 시각입니다.
서비스 배포 완료 시각	서비스 배포가 완료된 시각입니다.
서비스 배포 중지 시각	서비스 배포가 중지된 시각입니다.
서비스 배포 업데이트 시각	서비스 배포가 마지막으로 업데이트된 시각입니다.
소스 서비스 개정	현재 실행 중인 서비스 개정입니다.

속성	설명
	포함된 속성에 대한 내용은 Amazon ECS 서비스 개정에 포함된 속성 단원을 참조하세요.
배포 구성	회로 차단기 구성, 결정 경보가 포함된 배포 파라미터입니다. • 배포 중이거나 컨테이너 인스턴스 드레이닝 시 서비스에 대해 실행되어야 하는 태스크 수의 하한입니다(서비스에 대해 원하는 태스크 수의 백분율). • 배포 중이거나 컨테이너 인스턴스 드레이닝 시 서비스에 대해 실행되어야 하는 태스크 수의 상한입니다(서비스에 대해 원하는 태스크 수의 백분율). • 회로 차단기 구성입니다. • 실패 탐지에 사용되는 경보입니다.
대상 서비스 개정	배포되는 서비스 개정입니다. 배포가 완료되면 실행 중인 서비스 개정이 대상 서비스 개정입니다.

속성	설명
서비스 배포 상태	서비스 배포 상태입니다. 유효한 값은 PENDING, SUCCESSFUL, STOPPED, STOP_REQUESTED, STOP_IN_PROGRESS, IN_PROGRESS, ROLLBACK_IN_PROGRESS, ROLLBACK_SUCCESSFUL 및 ROLLBACK_FAILED입니다.
서비스 배포 상태 정보	서비스 배포가 현재 상태인 이유에 대한 정보입니다. 예: 실패가 회로 차단기에서 탐지되었습니다.
롤백 정보	배포 실패 시 서비스 배포에 사용되는 롤백 옵션입니다.
서비스 배포 회로 차단기 옵션	서비스 배포 실패가 결정되는 회로 차단기입니다.
서비스 배포에 대한 CloudWatch 경보	서비스 배포 실패 시점이 결정되는 CloudWatch 경보입니다.

Amazon ECS 서비스 배포를 보는 데 필요한 권한

최소 권한 부여의 모범 사례를 따는 경우 콘솔에서 서비스 배포를 보려면 추가 권한을 추가해야 합니다.

다음과 같은 조치에 대한 액세스 권한이 필요합니다.

- ListServiceDeployments
- DescribeServiceDeployments
- DescribeServiceRevisions

다음과 같은 리소스에 대한 액세스 권한이 필요합니다.

- Service
- 서비스 배포
- 서비스 개정

다음 예제 정책에는 필요한 권한이 있으며, 지정된 서비스로 조치가 제한됩니다.

`account`, `cluster-name` 및 `service-name`을 원하는 값으로 바꿉니다.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "ecs:ListServiceDeployments",
              "ecs:DescribeServiceDeployments",
              "ecs:DescribeServiceRevisions"
            ],
            "Resource": [
             "arn:aws:ecs:us-east-1:123456789012:service/cluster-name/service-name",
             "arn:aws:ecs:us-east-1:123456789012:service-deployment/cluster-name/
service-name/*",
             "arn:aws:ecs:us-east-1:123456789012:service-revision/cluster-name/service-
name/*"
            ]
        }
    ]
}
```

Amazon ECS 서비스 배포 보기

2024년 10월 25일 이후에 생성된 배포에 대한 최근 90일 기록을 참조할 수 있습니다. 서비스 배포 상태는 다음 중 하나일 수 있습니다.

- 진행 중
- 보류중
- 완료됨

이 정보를 사용하여 서비스가 배포되는 방식 또는 서비스 개정을 업데이트해야 하는지 결정할 수 있습니다. 포함된 속성에 대한 내용은 Amazon ECS 서비스 배포에 포함된 속성 단원을 참조하세요.

시작하기 전에 서비스 배포를 보는 데 필요한 권한을 구성합니다. 자세한 내용은 Amazon ECS 서비스 배포를 보는 데 필요한 권한 섹션을 참조하세요.

Amazon ECS Console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
3. 클러스터 세부 정보 페이지의 서비스 단원에서 서비스를 선택합니다.

   서비스 세부 정보 페이지가 표시됩니다.
4. 배포 세부 정보 페이지에서 배포를 선택합니다.
5. 보려는 서비스 배포를 선택합니다.

이 배포 유형에 대한 서비스 배포를 보는 방법	조치
진행 중인 배포	진행 중인 배포에서 배포 ID를 선택합니다.
마지막 배포	마지막 배포에서 배포 ID를 선택합니다.
완료된 배포	서비스 배포에서 배포 ID를 선택합니다.

   서비스 배포 세부 정보 페이지가 나타납니다.
6. (선택 사항) 서비스 개정을 비교하여 차이점을 봅니다.

   서비스 개정에서 개정 비교를 선택한 다음에 비교할 개정 2개를 선택합니다.

   서비스 개정은 차이점이 강조 표시되어 나란히 표시됩니다.

AWS CLI

1. `list-service-deployments`를 실행하여 서비스 배포 ARN을 검색합니다.

   변수를 원하는 값으로 바꿉니다.

   ```
   aws ecs list-service-deployments --cluster cluster-name --service service-name
   ```

   보려는 배포에 대한 serviceDeploymentArn을 기록해 둡니다.

   ```
   {
       "serviceDeployments": [
           {
               "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-
   deployment/example/sd-example/NCWGC2ZR-taawPAYrIaU5",
               "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/example/
   sd-example",
               "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/example",
               "targetServiceRevisionArn": "arn:aws:ecs:us-
   west-2:123456789012:service-revision/example/sd-example/4980306466373577095",
               "status": "SUCCESSFUL"
           }
       ]
   }
   ```

2. `describe-service-deployments`을(를) 실행합니다. `list-service-deployments`에서 반환된 `serviceDeploymentArn`을 사용합니다.

   변수를 원하는 값으로 바꿉니다.

   ```
   aws ecs describe-service-deployments --service-deployment-arns
    arn:aws:ecs:region:123456789012:service-deployment/cluster-name/service-
   name/NCWGC2ZR-taawPAYrIaU5
   ```

다음 단계

배포에서 서비스 개정에 대한 세부 정보를 볼 수 있습니다. 자세한 내용은 Amazon ECS 서비스 개정 세부 정보 보기 섹션을 참조하세요.

Amazon ECS 서비스 개정

서비스 개정에는 Amazon ECS에서 시도되고 있는 워크로드 구성의 레코드가 있습니다. 서비스를 생성하거나 배포할 때마다 Amazon ECS에서는 서비스 개정에서 배포하려는 구성이 자동으로 생성되고 캡처됩니다.

서비스 개정은 읽기 전용이며 고유한 식별자가 있습니다. 포함된 속성에 대한 내용은 Amazon ECS 서비스 개정에 포함된 속성 단원을 참조하세요.

서비스 개정에서는 다음과 같은 이점이 제공됩니다.

- 서비스 배포 중 현재 배포된 서비스 개정(소스)을 배포 중인 개정(대상)과 비교할 수 있습니다.
- 서비스 배포에 롤백 옵션을 사용하면 Amazon ECS에서는 서비스 배포가 성공적으로 배포된 마지막 서비스 개정으로 자동으로 롤백됩니다.
- 서비스 개정에는 리소스 하나에 워크로드 구성 레코드가 있습니다.

서비스 개정 수명 주기

서비스를 생성하거나 배포가 시작되는 서비스 속성을 업데이트할 때 새 서비스 개정이 Amazon ECS에서 자동으로 생성됩니다.

Amazon ECS에서는 롤백 작업에 대한 새 서비스 개정이 생성되지 않습니다. Amazon ECS에서는 롤백에 마지막으로 성공한 서비스 개정을 사용합니다.

서비스 개정은 변경할 수 없습니다.

서비스를 삭제할 때 Amazon ECS에서 서비스 개정이 삭제됩니다.

콘솔, API 및 CLI를 사용하여 2024년 10월 25일 이후에 생성된 서비스 개정을 볼 수 있습니다.

Amazon ECS 서비스 개정에 포함된 속성

다음과 같은 속성이 서비스 개정에 포함되어 있습니다.

리소스	설명	
서비스 ARN	서비스를 식별하는 ARN입니다.	

리소스	설명
클러스터 ARN	서비스가 호스팅되는 클러스터의 ARN입니다.
작업 정의 ARN	서비스 태스크에 사용되는 태스크 정의의 ARN입니다.
서비스 레지스트리	서비스 검색에 사용되는 서비스 레지스트리의 세부 정보입니다. • 컨테이너 이름 서비스 검색 서비스에 사용할 컨테이너 이름 값입니다. • 컨테이너 포트 서비스 검색 서비스에 사용할 포트 값입니다. 이 값은 태스크 정의에도 지정되어 있습니다. • 포트 서비스 복구 서비스에서 SRV 레코드를 지정한 경우 사용되는 포트 값입니다. • 레지스트리 ARN 서비스 레지스트리의 Amazon 리소스 이름(ARN)입니다.

리소스	설명
용량 공급자	용량 공급자 전략 세부 정보입니다. • 용량 공급자 이름입니다. 서비스에 사용되는 용량 공급자 이름입니다. • 용량 공급자 가중치 지정된 용량 공급자가 사용되어야 하는 시작된 총 태스크 수의 상대 백분율입니다. • 용량 공급자 기반 지정된 용량 공급자에서 실행되는 최소 태스크 수입니다.
컨테이너 이미지	컨테이너 이미지에 대한 세부 정보입니다. • 컨테이너 이름 • 이미지 다이제스트 • 컨테이너 이미지
네트워킹	서비스에 대한 네트워크 구성. • 퍼블릭 IP 주소가 있는지 여부 • 태스크가 실행되는 서브넷 • 서비스 트래픽이 제어되는 보안 그룹
시작 유형	서비스에 사용되는 시작 유형입니다.

리소스	설명	

Fargate별 속성	시작 유형이 Fargate인 경우 Fargate 버전에 대한 정보입니다.
	• 플랫폼 버전 및 플랫폼 패밀리
	• 임시 스토리지
	  • 배포에 할당되는 임시 스토리지 용량입니다.
	  • 스토리지 암호화에 사용되는 KMS 키입니다.	

리소스	설명	
배포 시 구성되는 Amazon EBS 볼륨입니다.	시작 시 구성되는 볼륨으로 태스크 정의에서 지정되는 볼륨에 대한 구성입니다. • Encrypted 볼륨이 암호화되는지 여부를 나타냅니다. • 파일 시스템 유형 볼륨의 Linux 파일 시스템 유형입니다. • IOPS 초당 I/O 작업 수(IOPS) 입니다. • KMS 키 • IAM 역할 이 볼륨과 연결되는 IAM 역할의 ARN입니다. • 크기 GiB 단위의 볼륨 크기입니다. • 스냅샷 ID 볼륨 스냅샷 ID입니다. • 태그 사양 볼륨의 태그 구성입니다. • 처리량 볼륨에 프로비저닝되는 처리량입니다. • 볼륨 유형	

리소스	설명
	볼륨 유형입니다.
Service Connect	Service Connect 구성입니다. • Service Connect가 사용 중인지 여부 표시 • 네임스페이스 • 상호 연결된 서비스의 목록 • Service Connect 로깅 구성
서비스 로드 밸런서	서비스 트래픽이 라우팅되는 로드 밸런서입니다. • 이름 • 컨테이너 이름 • 컨테이너 포트 • 대상 그룹 ARN
런타임 모니터링	Runtime Monitoring이 켜져 있는지를 나타냅니다.
생성 날짜	서비스 개정이 생성된 날짜입니다.
VPC Lattice	서비스 개정에 대한 VPC Lattice 구성입니다.

Amazon ECS 서비스 개정 세부 정보 보기

2024년 10월 25일 이후에 생성된 다음과 같은 서비스 개정 유형에 대한 내용을 볼 수 있습니다.

- 소스 - 현재 배포된 워크로드 구성
- 대상 - 배포 중인 워크로드 구성

Amazon ECS Console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
3. 클러스터 세부 정보 페이지의 서비스 단원에서 서비스를 선택합니다.

   서비스 세부 정보 페이지가 표시됩니다.
4. 배포 세부 정보 페이지에서 배포를 선택합니다.
5. 보려는 서비스 개정을 선택합니다.

이 배포 유형에 대한 서비스 개정을 보는 방법	조치	
진행 중인 배포	진행 중인 배포에서 다음을 수행합니다. • 소스 서비스 개정을 보려면 서비스 개정에서 소스 개정 유형의 서비스 개정 ID를 선택합니다. • 대상 서비스 개정을 보려면 서비스 개정에서 대상 개정 유형의 서비스 개정 ID를 선택합니다.	
마지막 배포	마지막 배포에서 대상 서비스 개정을 선택합니다.	
완료된 배포	서비스 배포에서 다음을 수행합니다. • 대상 서비스 개정에서 ID를 선택합니다.	

AWS CLI

1. `describe-service-deployments`를 실행하여 서비스 개정 ARN을 검색합니다.

변수를 원하는 값으로 바꿉니다.

```
aws ecs describe-service-deployments --service-deployment-arns
 arn:aws:ecs:region:account-id:service/cluster-name/service-name/NCWGC2ZR-
taawPAYrIaU5
```

sourceServiceRevisions 또는 targetServiceRevisions의 arn을 기록해 둡니다.

```
{
    "serviceDeployments": [
        {
            "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-
deployment/example/sd-example/NCWGC2ZR-taawPAYrIaU5",
            "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/example/
sd-example",
            "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/example",
            "updatedAt": "2024-09-10T16:49:35.572000+00:00",
            "sourceServiceRevision": {
                "arn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
example/sd-example/4980306466373578954",
                "requestedTaskCount": 0,
                "runningTaskCount": 0,
                "pendingTaskCount": 0
            },
            "targetServiceRevision": {
                "arn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
example/sd-example/4980306466373577095",
                "requestedTaskCount": 0,
                "runningTaskCount": 0,
                "pendingTaskCount": 0
            },
            "status": "IN_PROGRESS",
            "deploymentConfiguration": {
                "deploymentCircuitBreaker": {
                    "enable": false,
                    "rollback": false
                },
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            }
        }
    ],
```

```
    "failures": []
}
```

2. describe-service-revisions을(를) 실행합니다. describe-service-deployments에서 반환된 arn을 사용합니다.

   변수를 원하는 값으로 바꿉니다.

```
aws ecs describe-service-revisions --service-revision-arns
 arn:aws:ecs:region:123456789012:service-revision/cluster-name/service-
name/4980306466373577095
```

가용 영역 전체의 Amazon ECS 서비스 밸런싱

애플리케이션의 고가용성 달성에 도움이 되려면 여러 가용 영역에서 실행되도록 다중 태스크 서비스를 구성하는 것이 좋습니다. 첫 번째 배치 전략을 가용 영역 분산으로 지정하는 서비스의 경우 AWS에서는 사용 가능한 가용 영역에 서비스 태스크가 균등하게 분산되도록 최선을 다합니다.

그러나 한 가용 영역과 다른 가용 영역에서 실행되는 태스크 수가 동일하지 않은 경우가 있을 수 있습니다(예: 가용 영역 중단 후). 이 태스크 불균형이 해결되도록 가용 영역 리밸런싱 특성을 활성화할 수 있습니다.

가용 영역 리밸런싱을 통해 Amazon ECS에서는 각 서비스의 가용 영역 전체의 태스크 분산을 지속적으로 모니터링합니다. Amazon ECS에서는 균등하지 않은 태스크 분산이 감지되면 자동으로 가용 영역 사이에 워크로드가 리밸런싱되도록 조처합니다. 여기에는 태스크가 가장 적은 가용 영역에서 새 태스크를 시작하고 과부하가 걸린 가용 영역에서 태스크를 종료하는 것이 포함됩니다.

이 재분산은 장애 지점이 되는 가용 영역이 없어지므로 컨테이너화된 애플리케이션의 전체 가용성을 유지하는 데 도움이 됩니다. 자동 리밸런싱 프로세스를 통해 수동 개입 필요성이 제거되므로 이벤트 후 복구 시간이 단축됩니다.

다음은 가용 영역 리밸런싱 프로세스의 개요입니다.

1. Amazon ECS에서는 서비스의 상태가 안정되면 모니터링을 시작하고 각 가용 영역에서 실행되는 태스크 수를 조사합니다.
2. Amazon ECS에서는 각 가용 영역에서 실행되는 태스크 수에서 불균형이 탐지되면 다음과 같은 작업을 수행합니다.
   - 가용 영역 리밸런싱이 시작되고 있음을 나타내는 서비스 이벤트를 보냅니다.
   - 실행 중인 태스크 수가 가장 적은 가용 영역의 태스크를 시작합니다.

- 실행 중인 태스크 수가 가장 많은 가용 영역의 태스크를 중지합니다.
- 스케줄러에서는 과도하게 규모가 조정된 가용 영역의 태스크를 중지하기 전에 새로 시작된 태스크가 HEALTHY 및 RUNNING이 될 때까지 기다립니다.
- 가용 영역 리밸런싱 결과가 있는 서비스 이벤트를 보냅니다.

Amazon ECS에서 불균형 태스크 분배를 탐지하는 방법

Amazon ECS는 서비스의 원하는 태스크 수를 구성된 가용 영역 수로 나누어 각 가용 영역에서 실행되는 태스크 수의 불균형을 확인합니다. 원하는 태스크 수가 균등하게 분산되지 않는 경우 Amazon ECS는 나머지 태스크를 구성된 여러 가용 영역에서 균등하게 분산합니다. 각 가용 영역에는 태스크가 1개 이상 있어야 합니다.

2개의 가용 영역에 대해 원하는 태스크 수가 2개로 구성된 Amazon ECS 서비스를 예로 들어 보겠습니다. 이 시나리오에서는 원하는 태스크 수가 균등하게 배포됩니다. 균형 잡힌 분배는 가용 영역당 태스크 1개입니다. 가용 영역 1에 태스크가 2개 있고 가용 영역 2에 태스크가 없는 경우 Amazon ECS는 가용 영역 1에서 태스크를 중지하기 전에 가용 영역 2에서 태스크를 시작하여 재분배를 시작합니다.

이제 2개의 가용 영역에 대해 원하는 태스크 수가 3개로 구성된 Amazon ECS 서비스를 예로 들어 보겠습니다. 이 시나리오에서는 원하는 태스크 수가 균등하게 배포되지 않습니다. 각 가용 영역에는 태스크가 1개 이상 있고 나머지 태스크는 가용 영역 2에 배치되므로 균형 잡힌 분배는 가용 영역 1에 태스크 1개와 가용 영역 2에 태스크 2개입니다.

3개의 가용 영역에 대해 원하는 태스크 수가 5개로 구성된 Amazon ECS 서비스를 예로 들어 보겠습니다. 이 시나리오에서는 원하는 태스크 수가 균등하게 배포되지 않습니다. 균형 잡힌 분배는 가용 영역 1에 태스크 1개와 가용 영역 2와 3 각각에 태스크 2개입니다. 모든 가용 영역에 태스크가 1개씩 있음을 고려하여 나머지 2개의 태스크는 가용 영역에 균등하게 분배됩니다.

가용 영역 재분배 구성 시 고려 사항

가용 영역 재분배를 구성하려는 경우 다음 사항을 고려하세요.

- 가용 영역 리밸런싱에서는 Fargate 및 EC2 시작 유형을 지원합니다. Fargate의 경우 Amazon ECS에서는 사용 가능한 가용 영역 전체에 태스크를 자동으로 재분산하여 균형을 유지 관리합니다. EC2 시작 유형의 경우 Amazon ECS에서는 사용자가 정의한 배치 전략 및 제약 조건에 따라 기존 컨테이너 인스턴스 전체에 태스크를 리밸런싱합니다. 그러나 Amazon ECS에서는 리밸런싱 프로세스의 일부로 사용률이 낮은 가용 영역에서 새 인스턴스를 시작할 수 없으므로 기존 컨테이너 인스턴스로 리밸런싱이 제한됩니다.
- 가용 영역 리밸런싱은 다음과 같은 구성에서 작동합니다.

  - Replica 전략을 사용하는 서비스
  - 가용 영역 분산을 첫 번째 태스크 배치 전략으로 지정하지 않거나 배치 전략을 지정하지 않는 서비스입니다.
- 다음과 같은 기준 중 하나라도 서비스에서 충족되면 가용 영역 리밸런싱을 사용할 수 없습니다.
  - Daemon 전략 사용
  - EXTERNAL 시작 유형 사용(ECS Anywhere)
  - maximumPercent 값으로 100% 사용
  - Classic Load Balancer 사용
  - 태스크 배치 제약 조건으로 attribute:ecs.availability-zone 사용

가용 영역 리밸런싱 포함 배치 전략 및 배치 제약 조건

배치 전략에 따라 Amazon ECS에서 태스크 배치를 종료할 컨테이너 인스턴스와 가용 영역을 선택하는 방식이 결정됩니다. 태스크 배치 제약 조건은 태스크가 특정 컨테이너 인스턴스에서 실행될 수 있는지 여부가 결정되는 규칙입니다.

EC2 시작 유형의 경우 가용 영역 리밸런싱과 함께 배치 전략 및 배치 제약 조건을 사용할 수 있습니다. 그러나 가용 영역 리밸런싱이 작동하려면 지정된 첫 번째 전략이 가용 영역 분산 배치 전략이어야 합니다.

가용 영역 리밸런싱은 다양한 배치 전략 조합과 호환됩니다. 예를 들면, 가용 영역 전체에 균등하게 태스크를 분산한 다음에 각 가용 영역 내에서 메모리를 기준으로 태스크를 빈팩(bin pack)하는 전략을 생성할 수 있습니다. 이 경우 가용 영역 분산 전략이 먼저 지정되므로 가용 영역 리밸런싱이 작동합니다.

배치 전략 배열의 첫 번째 전략이 가용 영역 분산 구성 요소가 아닌 경우 가용 영역 리밸런싱이 작동하지 않는다는 점을 유의해야 합니다. 이 요구 사항에 따라 태스크 분산의 일차적인 초점이 고가용성에 매우 중요한 가용 영역 전체의 균형 유지 관리에 맞춰집니다.

태스크 배치 전략 및 제약 조건에 대한 자세한 내용은 Amazon ECS가 컨테이너 인스턴스에 작업을 배치하는 방법 단원을 참조하세요.

Fargate 시작 유형을 사용하는 태스크에 대해서는 작업 배치 전략 및 제약이 지원되지 않습니다. Fargate는 액세스 가능한 가용 영역 전체에 작업을 분산시키기 위해 최선을 다합니다. 용량 공급자에 Fargate와 Fargate 스팟이 모두 포함되는 경우 분산 동작은 용량 공급자마다 독립적입니다.

다음 예제 전략에서는 가용 영역 전체에 균등하게 태스크를 분산한 다음에 각 가용 영역 내에서 메모리를 기준으로 태스크를 빈팩(bin pack)합니다. 가용 영역 리밸런싱은 spread 전략이 첫 번째이므로 서비스와 호환됩니다.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

가용 영역 리밸런싱 켜기

신규 및 기존 서비스에 대해 가용 영역 리밸런싱을 활성화해야 합니다.

콘솔, API 또는 AWS CLI를 사용하여 가용 영역 리밸런싱을 활성화 및 비활성화할 수 있습니다.

서비스 유형	API	콘솔	CLI
기존	UpdateService	Amazon ECS 서비스 업데이트	update-service
New	CreateService	Amazon ECS 롤링 업데이트 배포 생성	create-service

다음 예제에서는 새 서비스를 생성할 때 서비스 리밸런싱을 활성화하는 방법을 보여줍니다.

```
aws ecs create-service \
    --cluster my-cluster \
    --service-name my-service \
    --task-definition my-task-definition:1 \
    --desired-count 6 \
    --availability-zone-rebalancing ENABLED
```

Amazon ECS 가용 영역 리밸런싱 추적

`describe-services`를 직접적으로 호출하거나 콘솔에서 서비스에 대해 가용 영역 리밸런싱이 활성화되어 있는지 확인할 수 있습니다. 다음 예제는 CLI를 통해 상태를 참조하는 데 사용할 수 있습니다.

응답은 `ENABLED` 또는 `DISABLED`입니다.

```
aws ecs describe-services \
    --services service-name \
    --cluster cluster-name \
    --query services[0].availabilityZoneRebalancing
```

서비스 이벤트

Amazon ECS에서는 가용 영역 리밸런싱 수명 주기를 이해하는 데 도움이 되는 서비스 작업 이벤트를 보냅니다.

Event	시나리오	유형	자세히 알아보기
SERVICE_REBALANCING_STARTED	Amazon ECS에서 가용 영역 리밸런싱 작업 시작	INFO	서비스(*service-name*)는 *## ## 1*의 *number-tasks* 태스크, *## ## 2*의 *number-tasks* 및 *## ## 3*의 *number-tasks* 로 밸런싱된 AZ가 아닙니다. AZ 리밸런싱이 진행 중입니다.
SERVICE_REBALANCING_COMPLETED	가용 영역 리밸런싱 작업 완료	INFO	서비스(*service-name*)는 *## ## 1*의 *number-tasks* 태스크, *## ## 2*의 *number-tasks* 태스크 및 *## ## 3*의 *number-tasks* 태스크로 밸런싱된 AZ입니다.
TASKS_STARTED	Amazon ECS에서 가용 영역 리밸런싱 작업의 일부로 태스크 시작	INFO	*service-name* 이 AZ 리밸런싱에 대한 *## ##*의 *number-ta*

Event	시나리오	유형	자세히 알아보기
			sks 태스크를 시작했습니다(*task-ids*).
TASKS_STOPPED	Amazon ECS에서 가용 영역 리밸런싱 작업의 일부로 태스크 중지	INFO	*service-name* 이 AZ 리밸런싱으로 인해 ## ##에서 실행 중인 태스크 *number-tasks* 를 중지했습니다(*task-id*).
SERVICE_TASK_PLACEMENT_FAILURE	Amazon ECS에서 가용 영역 리밸런싱 작업의 일부로 태스크 시작 실패	ERROR	EC2 시작 유형은 모든 요구 사항을 충족하는 컨테이너 인스턴스가 없기 때문에 서비스(*service-name*)가 태스크를 ## ##에 배치할 수 없습니다. 단원 참조 Fargate 시작 유형은 서비스(*service-name*)가 태스크를 ## ##에 배치할 수 없습니다. 단원 참조
TASKSET_SCALE_IN_FAILURE_BY_TASK_PROTECTION	태스크 보호가 사용되고 있어 가용 영역 리밸런싱 작업이 차단되었습니다.	INFO	*reason* 사유로 인해 *task-set-name* 태스크가 스케일 인 될 수 없어서 서비스(*service-name*)가 AZ 리밸런싱할 수 없었습니다.

Event	시나리오	유형	자세히 알아보기
SERVICE_REBALANCING_STOPPED	가용 영역 리밸런싱 작업이 중지되었습니다. Amazon ECS에서는 자세한 내용이 제공되는 추가 이벤트를 보냅니다.	INFO	서비스(*service-name*)가 AZ 리밸런싱을 중지했습니다.

작업 상태 변경 이벤트

Amazon ECS에서 리밸런싱 프로세스의 일부로 시작되는 태스크마다 태스크 상태 변경 이벤트(START)를 보냅니다.

Amazon ECS에서는 리밸런싱 프로세스의 일부로 중지되는 태스크마다 태스크 상태 변경 이벤트(STOPPED)를 보냅니다. 사유는 `Availability-zone rebalancing initiated by (deployment ecs-svc/`*`deployment-id`*`)`로 설정됩니다.

이벤트에 대한 자세한 내용은 Amazon ECS 작업 상태 변경 이벤트 단원을 참조하세요.

서비스 리밸런싱 문제 해결

서비스 리밸런싱 관련 문제가 발생하면 다음 문제 해결 단계를 고려하세요.

리밸런싱이 시작되지 않음

다음을 확인합니다.

- 서비스에 대해 서비스 리밸런싱이 활성화됨
- 서비스에서 지원되는 구성을 사용함(the section called “가용 영역 재분배 구성 시 고려 사항” 참조)
- 서비스가 안정된 상태에 도달함

리밸런싱 중 태스크 배치 실패

SERVICE_TASK_PLACEMENT_FAILURE 이벤트가 표시되는 경우:

- EC2 시작 유형의 경우: 대상 가용 영역에 사용 가능한 컨테이너 인스턴스가 있는지 확인
- Fargate 시작 유형의 경우: 리소스 제약 조건 또는 서비스 할당량 제한 태스크 배치가 있는지 확인
- 태스크 배치 제약 조건을 검토하여 적절한 태스크 분산을 방해하지 않는지 확인.

리밸런싱이 예기치 않게 중지됨

`SERVICE_REBALANCING_STOPPED` 이벤트가 표시되는 경우:

- 작업을 차단할 수도 있는 태스크 보호 확인
- 리밸런싱을 중단할 수도 있는 동시 서비스 배포 찾기
- 리밸런싱이 중지된 이유에 대한 추가 정보 확인을 위해 서비스 이벤트 검토

서비스 리밸런싱 모범 사례

서비스 리밸런싱을 최대한 활용하려면 다음 모범 사례를 따릅니다.

- 리밸런싱 작업 모니터링 - CloudWatch 경보를 설정하여 리밸런싱과 관련된 서비스 이벤트를 모니터링해 문제를 신속하게 식별합니다.
- 성능 영향 고려 - 이전 태스크가 중지되기 전에 새 태스크가 시작되면 리밸런싱 작업으로 인해 리소스 사용량이 일시적으로 증가할 수 있습니다.
- 전략적으로 태스크 보호 사용 - 리밸런싱 중에 종료해서는 안 되는 중요한 태스크가 있는 경우 태스크 보호 사용을 고려하세요.
- EC2 용량 계획 - EC2 시작 유형의 경우 모든 가용 영역에 걸쳐 효과적인 리밸런싱을 지원하기에 충분한 컨테이너 인스턴스가 있는지 확인합니다.
- 리밸런싱 동작 테스트 - 프로덕션 환경에서 리밸런싱에 의존하기 전에 비프로덕션 환경에서 리밸런싱 작업 중에 서비스가 어떻게 작동하는지 테스트합니다.

로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산

필요하다면 Elastic Load Balancing을 사용하여 서비스의 작업에 트래픽을 균등하게 분산하도록 서비스를 구성할 수 있습니다.

> **Note**
>
> 태스크 세트를 사용할 때 모두 Elastic Load Balancing을 사용하거나 Elastic Load Balancing을 사용하지 않도록 세트의 모든 태스크를 구성해야 합니다.

AWS Fargate에서 호스팅되는 Amazon ECS 서비스는 Application Load Balancer, Network Load Balancer, Gateway Load Balancer를 지원합니다. 다음 테이블을 참조하여 사용할 로드 밸런서 유형에 대해 알아보세요.

로드 밸런서 유형	다음과 같은 경우에 사용
Application Load Balancer	HTTP/HTTPS(또는 계층 7) 트래픽을 라우팅합니다. Application Load Balancer는 Amazon ECS 서비스에 사용할 수 있는 유용한 몇 가지 기능을 제공합니다. • 각 서비스는 여러 대상 그룹을 지정할 경우 여러 로드 밸런서의 트래픽을 처리하며 여러 로드 밸런스 포트를 노출합니다. • 이 기능은 Fargate와 EC2 인스턴스 모두에서 호스팅되는 작업에서 지원됩니다. • Application Load Balancer는 컨테이너가 동적 호스트 포트 매핑을 사용하도록 허용합니다(그래서 컨테이너 인스턴스마다 동일 서비스의 여러 작업이 허용됩니다). • Application Load Balancer는 포트 기반 라우팅과 우선순위 규칙을 지원합니다(그래서 여러 서비스가 단일 Application Load Balancer에서 동일한 리스너 포터를 사용할 수 있습니다).
Network Load Balancer	TCP 또는 UDP(또는 계층 4) 트래픽을 라우팅합니다.
Gateway Load Balancer	TCP 또는 UDP(또는 계층 4) 트래픽을 라우팅합니다.

로드 밸런서 유형	다음과 같은 경우에 사용	
	방화벽, 침입 감지 및 방지 시스템, 심층 패킷 검사 시스템과 같은 가상 어플라이언스를 사용하세요.	

서비스에 Network Load Balancer 또는 Gateway Load Balancer에만 제공되는 기능이 필요한 경우가 아니라면 이러한 최신 기능을 활용할 수 있도록 Amazon ECS 서비스에 Application Load Balancer를 사용하는 것을 권장합니다. Elastic Load Balancing에 대한 정보와 로드 밸런서 유형의 차이점에 대한 자세한 정보는 Elastic Load Balancing 사용 설명서를 참조하세요.

로드 밸런서에서는 사용한 만큼만 지불하면 됩니다. 자세한 정보는 Elastic Load Balancing 요금을 참조하세요.

Amazon ECS에 대한 로드 밸런서 상태 확인 파라미터 최적화

로드 밸런서는 로드 밸런서의 가용 영역에서 정상 상태의 대상에만 요청을 전송합니다. 각 대상은 대상 그룹에 등록됩니다. 로드 밸런서는 대상 그룹 상태 확인 설정을 사용하여 각 대상의 상태를 확인합니다. 대상을 등록한 후 대상은 상태 확인을 통과해야만 정상 상태로 간주됩니다. Amazon ECS는 로드 밸런서를 모니터링합니다. 로드 밸런서는 주기적으로 Amazon ECS 컨테이너에 상태 확인을 전송합니다. Amazon ECS 에이전트는 로드 밸런서를 모니터링하고 로드 밸런서가 컨테이너 상태를 보고할 때까지 기다립니다. 컨테이너가 정상 상태라고 판단하기 전에 이 작업을 수행합니다.

2개의 Elastic Load Balancing 상태 확인 파라미터가 배포 속도에 영향을 줍니다.

- 상태 확인 간격: 개별 컨테이너의 대략적인 상태 확인 사이의 대략적인 시간(초)을 결정합니다. 기본적으로 로드 밸런서는 30초마다 확인합니다.

  이 파라미터 이름은 다음과 같습니다.

  - Elastic Load Balancing API에서 `HealthCheckIntervalSeconds`
  - Amazon EC2 콘솔에서의 간격
- 정상 임곗값 수: 비정상 상태의 용기가 정상으로 간주되기 전에 필요한 연속된 상태 확인 성공 횟수를 결정합니다. 기본적으로 로드 밸런서는 대상 컨테이너가 정상임을 보고하기 전에 상태 확인을 5회 통과해야 합니다.

  이 파라미터 이름은 다음과 같습니다.

  - Elastic Load Balancing API에서 `HealthyThresholdCount`

- Amazon EC2 콘솔의 정상 임계값

중요: 새로 등록된 대상의 경우 정상 임곗값 수 설정에 관계없이 대상을 정상으로 간주하려면 한 번의 성공한 상태 확인만 필요합니다. 정상 임곗값 수는 대상이 비정상 상태에서 정상 상태로 다시 전환될 때만 적용됩니다.

기본 설정에서 대상이 비정상 상태가 되었다가 복구되는 경우 컨테이너 상태를 확인하는 데 걸리는 총 시간은 2분 30초(`30 seconds * 5 = 150 seconds`)입니다.

10초 이내에 서비스가 시작되고 안정화되면 상태 확인 프로세스의 속도를 높일 수 있습니다. 프로세스 속도를 높이려면 상태 확인 간격과 정상 임곗값 수를 줄입니다.

- `HealthCheckIntervalSeconds`(Elastic Load Balancing API 이름) 또는 간격(Amazon EC2 콘솔 이름): 5
- `HealthyThresholdCount`(Elastic Load Balancing API 이름) 또는 정상 임계값(Amazon EC2 콘솔 이름): 2

기본 설정에서 상태 확인 프로세스는 2분 30초가 걸리는 데 비해 이 설정을 사용하면 10초가 걸립니다.

Elastic Load Balancing 상태 확인에 대한 자세한 내용은 Elastic Load Balancing 사용 설명서의 Health checks for your target groups를 참조하세요.

Amazon ECS에 대한 로드 밸런서 Connection Draining 파라미터 최적화

최적화를 위해 클라이언트는 컨테이너 서비스에 대한 연결 유지 연결을 유지 관리합니다. 이렇게 하면 해당 클라이언트의 후속 요청에서 기존 연결을 재사용할 수 있습니다. 컨테이너로 향하는 트래픽을 중지하려면 로드 밸런서에 알립니다. 로드 밸런서는 클라이언트가 연결 유지 연결을 종료했는지 주기적으로 확인합니다. Amazon ECS 에이전트는 로드 밸런서를 모니터링하고 로드 밸런서에서 연결 유지 연결을 닫았음을 보고할 때까지 기다립니다(대상은 `UNUSED` 상태임).

대상을 `UNUSED` 상태로 전환하기 위해 로드 밸런서가 대기하는 시간을 등록 취소 지연이라고 합니다. 다음 로드 밸런서 파라미터를 구성하여 배포 속도를 높일 수 있습니다.

- `deregistration_delay.timeout_seconds`: 300(기본값)

서비스의 응답 시간이 1초 미만인 경우 파라미터를 다음 값으로 설정하여 로드 밸런서가 클라이언트 및 백엔드 서비스 간 연결이 끊어질 때까지 5초만 대기하도록 합니다.

- `deregistration_delay.timeout_seconds`: 5

> **Note**
>
> 파일 업로드 또는 스트리밍 연결 속도가 느린 경우와 같이 요청 수명이 긴 서비스의 경우 값을 5초로 설정하지 않습니다.

SIGTERM 응답성

Amazon ECS는 먼저 작업에 SIGTERM 신호를 보내 애플리케이션을 완료 및 종료해야 함을 알립니다. 그러면 Amazon ECS에서 SIGKILL 메시지를 보냅니다. 애플리케이션이 SIGTERM을 무시하는 경우 Amazon ECS 서비스는 프로세스를 종료하기 위한 SIGTERM 신호 전송을 기다려야 합니다.

Amazon ECS가 SIGKILL 메시지를 보내기 위해 대기하는 시간은 다음 Amazon ECS 에이전트 옵션으로 결정됩니다.

- `ECS_CONTAINER_STOP_TIMEOUT`: 30(기본값)

  컨테이너 에이전트 파라미터에 대한 자세한 내용은 GitHub의 Amazon ECS Container Agent를 참조하세요.

대기 시간을 단축하려면 Amazon ECS 에이전트 파라미터를 다음 값으로 설정합니다.

- `ECS_CONTAINER_STOP_TIMEOUT`: 2

  애플리케이션이 1초 넘게 걸리는 경우 값에 2를 곱하고 이 숫자를 값으로 사용합니다.

이 경우 Amazon ECS는 컨테이너가 종료될 때까지 2초 정도 기다린 다음, 애플리케이션이 중지되지 않으면 Amazon ECS에서 SIGKILL 메시지를 보냅니다.

또한 SIGTERM 신호를 트랩하고 이에 반응하도록 애플리케이션 코드를 수정할 수 있습니다. 다음은 JavaScript로 작성된 예제입니다.

```
process.on('SIGTERM', function() {
  server.close();
})
```

이 코드를 사용하면 HTTP 서버가 새 요청에 대한 수신을 중지하고 전송 중인 요청에 대한 응답을 완료한 다음 해당 이벤트 루프에 수행할 작업이 더 이상 없기 때문에 Node.js 프로세스가 종료됩니다. 이 경

우 프로세스가 전송 중인 요청을 완료하는 데 500밀리초 정도밖에 걸리지 않으면 중지 제한 시간을 기다리고 SIGKILL을 받지 않고도 조기에 종료합니다.

Amazon ECS에 대한 Application Load Balancer 사용

Application Load Balancer는 애플리케이션 계층(HTTP/HTTPS)에서 라우팅 결정을 내리고, 경로 기반 라우팅을 지원하며, 클러스터의 각 컨테이너 인스턴스 상의 하나 이상의 포트로 요청을 라우팅할 수 있습니다. Application Load Balancer는 동적 호스트 포트 매핑을 지원합니다. 예를 들어, 작업의 컨테이너 정의가 NGINX 컨테이너 포트로 포트 80을 지정하고, 호스트 포트로 포트 0을 지정하면 컨테이너 인스턴스의 임시 포트 범위(예: 최신 Amazon ECS 최적화 AMI에서 32768 ~ 61000)에서 호스트 포트가 동적으로 선택됩니다. 태스크가 시작되면 NGINX 컨테이너가 인스턴스 ID와 포트의 조합으로 Application Load Balancer에 등록되고 트래픽은 해당 컨테이너에 해당하는 인스턴스 ID와 포트에 분산됩니다. 이 동적 매핑을 통해 동일한 컨테이너 인스턴스에서 단일 서비스의 다중 작업이 가능합니다. 자세한 정보는 Application Load Balancer 사용 설명서를 참조하세요.

배포 속도를 높이기 위한 파라미터 설정 모범 사례에 대한 자세한 내용은 다음을 참조하세요.

- Amazon ECS에 대한 로드 밸런서 상태 확인 파라미터 최적화
- Amazon ECS에 대한 로드 밸런서 Connection Draining 파라미터 최적화

Amazon ECS와 함께 Application Load Balancer를 사용할 때는 다음 사항을 고려하세요.

- Amazon ECS에는 작업이 생성되고 중지될 때 로드 밸런서에 대상을 등록 및 등록 취소하는 데 필요한 권한을 제공하는 서비스 연결 IAM 역할이 필요합니다. 자세한 내용은 Amazon ECS에 대해 서비스 연결 역할 사용 섹션을 참조하세요.
- 대상 그룹은 IP 주소 유형이 IPv4로 설정되어 있어야 합니다.

  태스크는 IPv6를 지원하지만 IPv6 Elastic Load Balancing과의 서비스 통합은 지원되지 않습니다.
- `awsvpc` 네트워크 모드를 사용하는 작업이 있는 서비스의 경우 서비스의 대상 그룹을 만들 때 `instance`가 아닌 `ip`를 대상 유형으로 선택해야 합니다. 이는 `awsvpc` 네트워크 모드를 사용하는 태스크가 Amazon EC2 인스턴스가 아닌 탄력적 네트워크 인터페이스와 연결되기 때문입니다.
- 서비스에서 HTTP/HTTPS 서비스를 위한 포트 80 및 포트 443과 같은 여러 로드 밸런싱 포트에 대한 액세스가 필요한 경우 2개의 리스너를 구성할 수 있습니다. 한 리스너는 요청을 서비스로 전달하는 HTTPS를 담당하고 다른 리스너는 HTTP 요청을 적절한 HTTPS 포트로 경로를 재지정하는 책임을 집니다. 자세한 정보는 Application Load Balancers 사용 설명서의 Application Load Balancer에 리스너 생성을 참조하세요.
- 로드 밸런서 서브넷 구성에는 컨테이너 인스턴스가 상주하는 모든 가용 영역이 포함되어야 합니다.

- 서비스를 생성한 후에는 로드 밸런서 구성을 AWS Management Console에서 변경할 수 없습니다. AWS Copilot, AWS CloudFormation, AWS CLI 또는 SDK를 사용하여 로드 밸런서 구성을 AWS CodeDeploy 블루/그린 또는 외부가 아닌 ECS 롤링 배포 컨트롤러에 대해서만 변경할 수 있습니다. 로드 밸런서 구성을 추가, 업데이트 또는 제거하면 Amazon ECS가 업데이트된 Elastic Load Balancing 구성을 사용하여 새 배포를 시작합니다. 이로 인해 작업이 로드 밸런서에 등록되거나 로드 밸런서에서 등록 취소됩니다. Elastic Load Balancing 구성을 업데이트하기 전에 테스트 환경에서 이를 확인하는 것이 좋습니다. 구성을 변경하는 방법에 대한 자세한 정보는 Amazon Elastic Container Service API Reference(Amazon Elastic Container Service API 레퍼런스)의 UpdateService를 참조하세요.
- 서비스 작업이 로드 밸런서 상태 확인 기준에 실패하면 작업이 중단되고 다시 시작됩니다. 이 프로세스는 서비스가 원하는 실행 작업 수에 도달할 때까지 계속됩니다.
- 로드 밸런서-활성화 서비스에 문제가 있는 경우, Amazon ECS의 서비스 로드 밸런서 문제 해결을 참조하세요
- `instance` 대상 유형을 사용하는 경우 태스크 및 로드 밸런서는 동일한 VPC에 있어야 합니다. `ip` 대상 유형을 사용하는 경우 교차 VPC 연결이 지원됩니다.
- 각 서비스에 대해 고유한 대상 그룹을 사용합니다.

  여러 서비스에 대해 동일한 대상 그룹을 사용하면 서비스 배포 중에 문제가 발생할 수 있습니다.

Application Load Balancer를 생성하는 방법에 대한 자세한 내용은 Application Load Balancer의 Create an Application Load Balancer를 참조하세요.

Amazon ECS에 대해 Network Load Balancer 사용

Network Load Balancer는 전송 계층(TCP/SSL)에서 라우팅을 결정합니다. 초당 수백만 개의 요청을 처리할 수 있습니다. 로드 밸런서가 연결을 수신하면 해시 라우팅 알고리즘 흐름에 따라 기본 규칙의 대상 그룹에서 대상을 선택합니다. 리스너 구성에 지정된 포트에서 선택한 대상에 대한 TCP 연결을 열려고 시도합니다. 헤더를 수정하지 않고 요청을 전달합니다. Network Load Balancer는 동적 호스트 포트 매핑을 지원합니다. 예를 들어, 작업의 컨테이너 정의가 NGINX 컨테이너 포트로 포트 80을 지정하고, 호스트 포트로 포트 0을 지정하면 컨테이너 인스턴스의 임시 포트 범위(예: 최신 Amazon ECS 최적화 AMI에서 32768 ~ 61000)에서 호스트 포트가 동적으로 선택됩니다. 작업이 시작되면 NGINX 컨테이너가 인스턴스 ID-포트 조합으로 Network Load Balancer에 등록되고, 트래픽은 해당 컨테이너에 해당하는 인스턴스 ID와 포트에 분산됩니다. 이 동적 매핑을 통해 동일한 컨테이너 인스턴스에서 단일 서비스의 다중 작업이 가능합니다. Network Load Balancer에 대한 자세한 정보는 Network Load Balancer 사용 설명서를 참조하세요.

배포 속도를 높이기 위한 파라미터 설정 모범 사례에 대한 자세한 내용은 다음을 참조하세요.

- Amazon ECS에 대한 로드 밸런서 상태 확인 파라미터 최적화
- Amazon ECS에 대한 로드 밸런서 Connection Draining 파라미터 최적화

Amazon ECS와 함께 Network Load Balancer를 사용할 때는 다음을 고려하세요.

- Amazon ECS에는 작업이 생성되고 중지될 때 로드 밸런서에 대상을 등록 및 등록 취소하는 데 필요한 권한을 제공하는 서비스 연결 IAM 역할이 필요합니다. 자세한 내용은 Amazon ECS에 대해 서비스 연결 역할 사용 섹션을 참조하세요.
- 서비스에 5개 이상의 대상 그룹을 연결할 수 없습니다.
- `awsvpc` 네트워크 모드를 사용하는 작업이 있는 서비스의 경우 서비스의 대상 그룹을 만들 때 `instance`가 아닌 `ip`를 대상 유형으로 선택해야 합니다. 이는 `awsvpc` 네트워크 모드를 사용하는 작업이 Amazon EC2 인스턴스가 아닌 탄력적 네트워크 인터페이스와 연결되기 때문입니다.
- 로드 밸런서 서브넷 구성에는 컨테이너 인스턴스가 상주하는 모든 가용 영역이 포함되어야 합니다.
- 서비스를 생성한 후에는 로드 밸런서 구성을 AWS Management Console에서 변경할 수 없습니다. AWS Copilot, AWS CloudFormation, AWS CLI 또는 SDK를 사용하여 로드 밸런서 구성을 AWS CodeDeploy 블루/그린 또는 외부가 아닌 ECS 롤링 배포 컨트롤러에 대해서만 변경할 수 있습니다. 로드 밸런서 구성을 추가, 업데이트 또는 제거하면 Amazon ECS가 업데이트된 Elastic Load Balancing 구성을 사용하여 새 배포를 시작합니다. 이로 인해 작업이 로드 밸런서에 등록되거나 로드 밸런서에서 등록 취소됩니다. Elastic Load Balancing 구성을 업데이트하기 전에 테스트 환경에서 이를 확인하는 것이 좋습니다. 구성을 변경하는 방법에 대한 자세한 정보는 Amazon Elastic Container Service API Reference(Amazon Elastic Container Service API 레퍼런스)의 UpdateService를 참조하세요.
- 서비스 작업이 로드 밸런서 상태 확인 기준에 실패하면 작업이 중단되고 다시 시작됩니다. 이 프로세스는 서비스가 원하는 실행 작업 수에 도달할 때까지 계속됩니다.
- IP 주소를 대상으로 구성하고 클라이언트 IP 보존을 해제한 Gateway Load Balancer를 사용하면 요청이 Gateway Load Balancer의 프라이빗 IP 주소에서 오는 것으로 간주됩니다. 즉, Gateway Load Balancer 뒤에 있는 서비스는 대상 보안 그룹에서 들어오는 요청과 상태 확인을 허용하는 즉시 전 세계에 효과적으로 개방됩니다.
- Fargate 태스크의 경우 플랫폼 버전 `1.4.0`(Linux) 또는 `1.0.0`(Windows)을 사용해야 합니다.
- 로드 밸런서-활성화 서비스에 문제가 있는 경우, Amazon ECS의 서비스 로드 밸런서 문제 해결을 참조하세요
- `instance` 대상 유형을 사용하는 경우 태스크 및 로드 밸런서는 동일한 VPC에 있어야 합니다. `ip` 대상 유형을 사용하는 경우 교차 VPC 연결이 지원됩니다.
- Network Load Balancer 클라이언트 IP 주소 보존은 Fargate 대상과 호환됩니다.

- 각 서비스에 대해 고유한 대상 그룹을 사용합니다.

  여러 서비스에 대해 동일한 대상 그룹을 사용하면 서비스 배포 중에 문제가 발생할 수 있습니다.

Network Load Balancer를 생성하는 방법에 대한 자세한 내용은 Network Load Balancer의 Create a Network Load Balancer를 참조하세요.

> **Important**
>
> 서비스의 태스크 정의가 `awsvpc`네트워크 모드(Fargate 시작 유형의 경우에는 필수)를 사용하는 경우 대상 유형을 `instance`가 아닌 `ip`로 선택해야 합니다. 이는 `awsvpc` 네트워크 모드를 사용하는 작업이 Amazon EC2 인스턴스가 아닌 탄력적 네트워크 인터페이스와 연결되기 때문입니다.
> C1, CC1, CC2, CG1, CG2, CR1, G1, G2, HI1, HS1, M1, M2, M3, T1 인스턴스 유형인 경우 인스턴스 ID로 인스턴스를 등록할 수 없습니다. 이러한 인스턴스 유형은 IP 주소로 등록할 수 있습니다.

Amazon ECS에 Gateway Load Balancer 사용

Gateway Load Balancer는 오픈 시스템 상호 연결(OSI) 모델의 세 번째 계층인 네트워크 계층에서 작동합니다. 모든 포트에서 모든 IP 패킷을 수신하고 리스너 규칙에 지정된 대상 그룹으로 트래픽을 전달합니다. 5튜플(TCP/UDP 흐름의 경우) 또는 3튜플(비 TCP/UDP 흐름의 경우)을 사용하여 특정 대상 어플라이언스에 대한 흐름의 연결을 유지합니다. 예를 들어, 작업의 컨테이너 정의가 NGINX 컨테이너 포트로 포트 80을 지정하고, 호스트 포트로 포트 0을 지정하면 컨테이너 인스턴스의 임시 포트 범위(예: 최신 Amazon ECS 최적화 AMI에서 32768 ~ 61000)에서 호스트 포트가 동적으로 선택됩니다. 태스크가 시작되면 NGINX 컨테이너가 인스턴스 ID와 포트의 조합으로 Gateway Load Balancer에 등록되고 트래픽은 해당 컨테이너에 해당하는 인스턴스 ID와 포트에 분산됩니다. 이 동적 매핑을 통해 동일한 컨테이너 인스턴스에서 단일 서비스의 다중 작업이 가능합니다. 자세한 내용은 Gateway Load Balancer에서 Gateway Load Balancer란 무엇입니까?를 참조하세요.

배포 속도를 높이기 위한 파라미터 설정 모범 사례에 대한 자세한 내용은 다음을 참조하세요.

- Amazon ECS에 대한 로드 밸런서 상태 확인 파라미터 최적화
- Amazon ECS에 대한 로드 밸런서 Connection Draining 파라미터 최적화

Amazon ECS와 함께 Gateway Load Balancer를 사용할 때는 다음을 고려하세요.

- Amazon ECS에는 작업이 생성되고 중지될 때 로드 밸런서에 대상을 등록 및 등록 취소하는 데 필요한 권한을 제공하는 서비스 연결 IAM 역할이 필요합니다. 자세한 내용은 Amazon ECS에 대해 서비스 연결 역할 사용 섹션을 참조하세요.
- `awsvpc` 네트워크 모드를 사용하는 작업이 있는 서비스의 경우 서비스의 대상 그룹을 만들 때 `instance`가 아닌 `ip`를 대상 유형으로 선택해야 합니다. 이는 `awsvpc` 네트워크 모드를 사용하는 작업이 Amazon EC2 인스턴스가 아닌 탄력적 네트워크 인터페이스와 연결되기 때문입니다.
- `awsvpc` 이외의 네트워크 모드를 사용하는 작업이 있는 서비스의 경우게이트웨이 로드 밸런서가 지원되지 않습니다.
- 로드 밸런서 서브넷 구성에는 컨테이너 인스턴스가 상주하는 모든 가용 영역이 포함되어야 합니다.
- 서비스를 생성한 후에는 로드 밸런서 구성을 AWS Management Console에서 변경할 수 없습니다. AWS Copilot, AWS CloudFormation, AWS CLI 또는 SDK를 사용하여 로드 밸런서 구성을 AWS CodeDeploy 블루/그린 또는 외부가 아닌 ECS 롤링 배포 컨트롤러에 대해서만 변경할 수 있습니다. 로드 밸런서 구성을 추가, 업데이트 또는 제거하면 Amazon ECS가 업데이트된 Elastic Load Balancing 구성을 사용하여 새 배포를 시작합니다. 이로 인해 작업이 로드 밸런서에 등록되거나 로드 밸런서에서 등록 취소됩니다. Elastic Load Balancing 구성을 업데이트하기 전에 테스트 환경에서 이를 확인하는 것이 좋습니다. 구성을 변경하는 방법에 대한 자세한 정보는 Amazon Elastic Container Service API Reference(Amazon Elastic Container Service API 레퍼런스)의 UpdateService를 참조하세요.
- 서비스 작업이 로드 밸런서 상태 확인 기준에 실패하면 작업이 중단되고 다시 시작됩니다. 이 프로세스는 서비스가 원하는 실행 작업 수에 도달할 때까지 계속됩니다.
- IP 주소를 대상으로 구성한 Gateway Load Balancer를 사용하면 요청이 Gateway Load Balancer의 프라이빗 IP 주소에서 오는 것으로 간주됩니다. 즉, Gateway Load Balancer 뒤에 있는 서비스는 대상 보안 그룹에서 들어오는 요청과 상태 확인을 허용하는 즉시 전 세계에 효과적으로 개방됩니다.
- Fargate 태스크의 경우 플랫폼 버전 `1.4.0`(Linux) 또는 `1.0.0`(Windows)을 사용해야 합니다.
- 로드 밸런서-활성화 서비스에 문제가 있는 경우, Amazon ECS의 서비스 로드 밸런서 문제 해결을 참조하세요
- `instance` 대상 유형을 사용하는 경우 태스크 및 로드 밸런서는 동일한 VPC에 있어야 합니다. `ip` 대상 유형을 사용하는 경우 교차 VPC 연결이 지원됩니다.
- 각 서비스에 대해 고유한 대상 그룹을 사용합니다.

  여러 서비스에 대해 동일한 대상 그룹을 사용하면 서비스 배포 중에 문제가 발생할 수 있습니다.

Gateway Load Balancer를 생성하는 방법에 대한 자세한 내용은 Gateway Load Balancer의 Gateway Load Balancer 시작하기를 참조하세요.

> **Important**
>
> 서비스의 태스크 정의가 `awsvpc`네트워크 모드(Fargate 시작 유형의 경우에는 필수)를 사용하는 경우 대상 유형을 `instance`가 아닌 `ip`로 선택해야 합니다. 이는 `awsvpc` 네트워크 모드를 사용하는 작업이 Amazon EC2 인스턴스가 아닌 탄력적 네트워크 인터페이스와 연결되기 때문입니다.
>
> C1, CC1, CC2, CG1, CG2, CR1, G1, G2, HI1, HS1, M1, M2, M3, T1 인스턴스 유형인 경우 인스턴스 ID로 인스턴스를 등록할 수 없습니다. 이러한 인스턴스 유형은 IP 주소로 등록할 수 있습니다.

Amazon ECS 서비스에 여러 대상 그룹 등록

Amazon ECS 서비스는 서비스 정의에 여러 대상 그룹을 지정할 경우 여러 로드 밸런서의 트래픽을 처리하며 여러 로드 밸런스 포트를 노출합니다.

여러 대상 그룹을 지정하는 서비스를 생성하려는 경우 Amazon ECS API, SDK, AWS CLI 또는 AWS CloudFormation 템플릿을 사용하여 서비스를 생성해야 합니다. 서비스가 생성되면 AWS Management Console에서 서비스 및 서비스에 등록된 대상 그룹을 볼 수 있습니다. 기존 서비스의 로드 밸런서 구성을 수정하려면 `UpdateService`를 사용해야 합니다.

여러 대상 그룹은 다음 유형을 사용해 서비스 정의에서 정의될 수 있습니다. 서비스 정의의 전체 구문은 서비스 정의 템플릿 섹션을 참조하세요.

```
"loadBalancers":[
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"container_name",
      "containerPort":container_port
   },
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"container_name",
      "containerPort":container_port
   }
]
```

고려 사항

서비스 정의에 여러 대상 그룹을 지정할 때는 다음을 고려해야 합니다.

- Application Load Balancer 또는 Network Load Balancer를 사용하는 서비스의 경우 6개 이상의 대상 그룹을 서비스에 연결할 수 없습니다.
- 서비스 정의에 여러 대상 그룹을 지정하는 것은 다음 조건에서만 지원됩니다.
  - 서비스는 Application Load Balancer 또는 Network Load Balancer를 사용해야 합니다.
  - 서비스는 롤링 업데이트(ECS) 배포 컨트롤러 유형을 사용해야 합니다.
- 여러 대상 그룹 지정은 Fargate 및 EC2 시작 유형을 모두 사용하는 태스크를 포함하는 서비스에 대해 지원됩니다.
- 여러 대상 그룹을 지정하는 서비스를 생성하는 경우 Amazon ECS 서비스 연결 역할을 생성해야 합니다. 역할은 API 요청에서 `role` 파라미터를 생략하거나 AWS CloudFormation에서 `Role` 속성을 생략하여 생성됩니다. 자세한 정보는 Amazon ECS에 대해 서비스 연결 역할 사용 섹션을 참조하세요.

서비스 정의 예

다음은 서비스 정의에 여러 대상 그룹을 정의하는 사용 사례입니다. 서비스 정의의 전체 구문은 서비스 정의 템플릿 섹션을 참조하세요.

내부 및 외부 트래픽에 대해 개별 로드 밸런서 사용

다음 사용 사례에서 서비스는 동일한 컨테이너 및 포트에 내부 트래픽 및 인터넷 경계 트래픽용으로 두 개의 로드 밸런서를 사용합니다.

```
"loadBalancers":[
   //Internal ELB
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"nginx",
      "containerPort":8080
   },
   //Internet-facing ELB
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
```

```
      "containerName":"nginx",
      "containerPort":8080
   }
]
```

동일한 컨테이너에 여러 포트 노출

다음 사용 사례에서 서비스는 하나의 로드 밸런서를 사용하지만 동일한 컨테이너에 여러 포트를 노출합니다. 예를 들어 Jenkins 컨테이너는 Jenkins 웹 인터페이스용으로 8080 포트를 노출하고 API용으로 50000 포트를 노출할 수 있습니다.

```
"loadBalancers":[
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"jenkins",
      "containerPort":8080
   },
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"jenkins",
      "containerPort":50000
   }
]
```

여러 컨테이너의 포트 노출

다음 사용 사례에서 서비스는 하나의 로드 밸런서와 두 개의 대상 그룹을 사용해 각 컨테이너별로 개별 포트를 노출합니다.

```
"loadBalancers":[
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"webserver",
      "containerPort":80
   },
   {
```

```
 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"database",
      "containerPort":3306
   }
]
```

Amazon ECS 서비스 자동 조정

자동 크기 조정은 Amazon ECS 서비스에서 원하는 작업 수를 자동으로 늘리거나 줄이는 기능입니다. Amazon ECS는 Application Auto Scaling 서비스를 활용하여 이 기능을 제공합니다. 자세한 정보는 Application Auto Scaling 사용 설명서를 참조하세요.

Amazon ECS는 서비스의 평균 CPU 및 메모리 사용량과 함께 CloudWatch 지표를 게시합니다. 자세한 정보는 Amazon ECS 서비스 사용률 지표 섹션을 참조하세요. 이와 함께 다른 CloudWatch 지표를 사용하여 피크 시간에는 서비스를 확장(더 많은 태스크를 추가)하여 높은 수요를 처리하고 사용률이 낮은 기간에는 서비스를 축소(더 적은 태스크를 실행)하여 비용을 절감할 수 있습니다.

Amazon ECS 서비스 Auto Scaling은 다음과 같은 유형의 Auto Scaling을 지원합니다.

- 대상 지표를 사용하여 Amazon ECS 서비스 규모 조정— 특정 지표에 대한 대상 값을 기준으로 서비스가 실행하는 작업의 수를 늘리거나 줄입니다. 이 과정은 온도 조절기를 사용하여 집안 온도를 유지하는 방법과 비슷합니다. 사용자가 온도를 선택하면 나머지는 모두 온도 조절기에서 자동으로 수행됩니다.
- CloudWatch 경보를 기반으로 미리 정의된 증분을 사용하여 Amazon ECS 서비스 규모 조정— 일련의 조정 조절(경보 위반의 크기에 따라 달라지는 단계 조절)을 기준으로 서비스가 실행하는 작업의 수를 늘리거나 줄입니다.
- 예약된 조치를 사용하여 Amazon ECS 서비스 규모 조정 - 날짜 및 시간을 기준으로 서비스가 실행하는 작업 수를 늘리거나 줄입니다.
- 과거 패턴을 사용하여 예측 규모 조정으로 Amazon ECS 서비스 규모 조정 - 과거 로드 데이터 분석을 기반으로 서비스가 실행되는 태스크 수를 늘리거나 줄여 트래픽 흐름의 일간 또는 주간 패턴을 탐지합니다.

고려 사항

조정 정책을 사용하는 경우 다음 고려 사항을 참조하세요.

- Amazon ECS는 1분 간격으로 지표를 CloudWatch로 보냅니다. 클러스터 및 서비스가 CloudWatch로 지표를 전송할 때까지는 지표를 사용할 수 없으며, 아직 존재하지 않는 지표에 대해 CloudWatch 경보를 생성할 수 없습니다.
- 조정 정책은 휴지 기간을 지원합니다. 이것은 이전의 조정 활동이 적용될 때까지 대기할 시간(초)입니다.
  - 확장 이벤트의 목적은 지속적이지만 과도하지는 않게 확장하는 것입니다. 서비스 Auto Scaling에서 조정 정책을 사용하여 성공적으로 확장하면 휴지 기간이 계산되기 시작합니다. 조정 정책은 더 큰 조정이 시작되거나 휴지 기간이 종료되지 않는 한 원하는 용량을 다시 늘리지 않습니다. 확장 휴지 기간이 진행되는 동안 확장 활동을 시작하여 추가된 용량은 다음 확장 활동에 대해 원하는 용량의 일부로 계산됩니다.
  - 축소 이벤트의 목적은 애플리케이션의 가용성을 보호하기 위해 보수적으로 축소하여 휴지 기간이 만료될 때까지 축소 활동을 차단하는 것입니다. 그러나 축소 휴지 기간 중에 다른 경보가 확장 활동을 시작하면 서비스 Auto Scaling은 대상을 즉시 확장합니다. 이 경우 축소 휴지 기간이 중지되고 완료되지 않습니다.
- 서비스 스케줄러는 원하는 수를 항상 신뢰하지만 서비스에 활성화된 조정 정책과 경보가 있는 한 서비스 Auto Scaling은 사용자가 수동으로 설정한 원하는 수를 변경할 수 있습니다.
- 원하는 서비스 수가 최소 용량 값 이하로 설정되어 있고 경보가 확장 활동을 시작하는 경우 서비스 Auto Scaling은 원하는 수를 최소 용량 값까지 확장한 다음 경보에 연결된 조정 정책에 따라 필요한 만큼 계속 스케일 아웃합니다. 하지만 원하는 수가 이미 최소 용량 값 미만이기 때문에 축소 활동은 원하는 수를 조정하지 않습니다.
- 원하는 서비스 수가 최대 용량 값 이상으로 설정되어 있고 경보가 스케일 인 활동을 시작하는 경우 서비스 Auto Scaling은 원하는 수를 최대 용량 값까지 스케일 아웃한 다음, 경보에 연결된 조정 정책에 따라 필요한 만큼 계속 스케일 인합니다. 하지만 원하는 수가 이미 최대 용량 값을 초과하므로 확장 활동은 원하는 수를 조절하지 않습니다.
- 조정 활동 중에 서비스 Auto Scaling이 시작점으로 사용하는 값은 원하는 수가 아니라 서비스에서 실제 실행 중인 태스크 수입니다. 이것이 처리 용량입니다. 이로써, 가령 추가 태스크를 배치하기에 충분한 컨테이너 인스턴스 리소스가 없는 경우 등 충족될 수 없는 과도한(불필요한) 조정을 방지합니다. 나중에 컨테이너 인스턴스 용량을 사용할 수 있는 경우 대기 중인 조정 활동이 승계되며 휴지 기간 후에 추가 조정 활동이 계속될 수 있습니다.
- 수행할 작업이 없을 때 태스크 수를 0으로 조정하려면 최소 용량을 0으로 설정합니다. 대상 추적 조정 정책을 사용하는 경우 실제 용량이 0이고 지표에 워크로드 요구량이 있다고 나타나면 서비스 Auto Scaling이 확장하기 전에 하나의 데이터 포인트가 전송될 때까지 기다립니다. 이 경우 시작 지점으로 가능한 최소량만큼 확장한 다음 실제 실행 중인 작업 수를 기준으로 조정을 재개합니다.
- Application Auto Scaling은 Amazon ECS 배포가 진행 중인 동안 축소 프로세스를 비활성화합니다. 그러나 배포 중에 일시 중단되지 않는 한 확장 프로세스는 계속 발생합니다. 이 동작은 외부 배포 컨

트롤러를 사용하는 Amazon ECS 서비스에 적용되지 않습니다. 자세한 내용은 서비스 Auto Scaling 및 배포 섹션을 참조하세요.

- Amazon ECS 작업을 위한 여러 Application Auto Scaling 옵션이 있습니다. 대상 추적이 가장 사용하기 쉬운 모드입니다. 이 옵션을 사용하는 경우 CPU 평균 사용률과 같은 지표의 목표 값을 설정하기만 하면 됩니다. 그러면 자동 스케일러가 해당 값을 달성하는 데 필요한 작업 수를 자동으로 관리합니다. 단계 조정을 사용하면 조정 지표의 특정 임계값과 임계값을 초과했을 때 추가 또는 제거할 작업 수를 정의하므로 수요 변화에 더 빠르게 대응할 수 있습니다. 더욱이 임계값 경보를 위반하는 시간을 최소화하여 수요 변화에 매우 빠르게 대응할 수 있습니다.

서비스 오토 스케일링 모범 사례에 대한 자세한 내용은 Amazon ECS 서비스 오토 스케일링 최적화 섹션을 참조하세요.

서비스 Auto Scaling 및 배포

Application Auto Scaling은 Amazon ECS 배포가 진행 중인 동안 축소 프로세스를 비활성화합니다. 그러나 배포 중에 일시 중단되지 않는 한 확장 프로세스는 계속 발생합니다. 이 동작은 외부 배포 컨트롤러를 사용하는 Amazon ECS 서비스에 적용되지 않습니다. 배포가 진행되는 동안 확장 프로세스를 일시 중단하려면 다음 단계를 수행합니다.

1. describe-scalable-targets 명령을 호출하여 Application Auto Scaling에서 확장 가능한 대상과 연결된 서비스의 리소스 ID를 지정합니다(예:`service/default/sample-webapp`). 출력 결과를 기록합니다. 다음 명령을 호출할 때 필요합니다.
2. register-scalable-target 명령을 호출하여 리소스 ID, 네임스페이스 및 확장 가능한 차원을 지정합니다. `DynamicScalingInSuspended` 및 `DynamicScalingOutSuspended` 모두에 대해 `true`를 지정합니다.
3. 배포가 완료되면 register-scalable-target 명령을 호출하여 조정을 재개할 수 있습니다.

자세한 정보는 Application Auto Scaling의 조정 일시 중지 및 재개를 참조하세요.

대상 지표를 사용하여 Amazon ECS 서비스 규모 조정

대상 추적 조정 정책을 사용하는 경우 지표를 선택하고 목표 값을 설정합니다. Amazon ECS 서비스 Auto Scaling은 조정 정책을 제어하는 CloudWatch 경보를 생성 및 관리하고 지표와 대상 값을 기준으로 조정 조절을 계산합니다. 조정 정책은 필요에 따라 서비스 태스크를 추가하거나 제거하여 측정치를 지정된 대상 값과 같거나 비슷하게 유지합니다. 측정치를 대상 값과 비슷하게 유지하는 것 외에도, 대상 추적 조정 정책은 변동하는 로드 패턴으로 인한 측정치 변동에 맞게 조정하고 서비스에서 실행 중인 작업 수의 급격한 변동을 최소화합니다.

대상 추적 정책을 사용하면 CloudWatch 경보 및 조정 작업을 수동으로 정의할 필요가 없습니다. Amazon ECS에서는 사용자가 설정하는 대상에 따라 자동으로 처리됩니다.

대상 추적 정책을 사용할 때 다음을 고려하세요.

- 대상 추적 조정 정책은 지정한 지표가 목표 값을 초과할 때 한해서 확장을 수행해야 합니다. 대상 추적 조정 정책에서는 지정한 지표가 목표 값보다 작을 때 확장할 수 없습니다.
- 대상 추적 조정 정책에서는 지정한 지표에 데이터가 부족할 때 조정을 수행하지 않습니다. 데이터가 부족하다고 해서 사용량이 낮은 것으로 해석하지 않기 때문에 축소를 수행하지 않습니다.
- 목표 값과 실제 지표 데이터 포인트 사이에는 차이가 발생할 수 있습니다. 서비스 Auto Scaling이 추가하거나 제거할 용량을 결정할 때마다 항상 반올림 또는 내림을 통해 어림짐작으로 동작하기 때문입니다. 이는 용량을 부족하게 추가하거나 너무 많이 제거하는 일을 방지하기 위해서입니다.
- 애플리케이션 가용성을 보장하기 위해 서비스는 지표에 비례하여 가능한 한 빠르게 확장하지만, 비교적 점진적으로 축소합니다.
- Application Auto Scaling은 Amazon ECS 배포가 진행 중인 동안 축소 프로세스를 비활성화합니다. 그러나 배포 중에 일시 중단되지 않는 한 확장 프로세스는 계속 발생합니다. 이 동작은 외부 배포 컨트롤러를 사용하는 Amazon ECS 서비스에 적용되지 않습니다. 자세한 내용은 서비스 Auto Scaling 및 배포 섹션을 참조하세요.
- 각 정책이 다른 측정치를 사용한다면 한 Amazon ECS 서비스에 대해 여러 대상 추적 조정 정책을 생성할 수 있습니다. 서비스 Auto Scaling은 항상 가용성을 우선시하므로, 대상 추적 정책이 확장 또는 축소를 허용하는지에 따라 그 동작이 달라집니다. 대상 추적 정책 중 하나라도 스케일 아웃할 준비가 된 경우 서비스를 스케일 아웃하지만 모든 대상 추적 정책(스케일 인 부분이 켜진 상태)이 스케일 인할 준비가 된 경우에만 스케일 인합니다.
- 서비스 Auto Scaling의 대상 추적 조정 정책에서 관리되는 CloudWatch 경보는 편집하거나 삭제하지 마세요. 조정 정책을 삭제하면 서비스 Auto Scaling에서 경보가 자동으로 삭제됩니다.
- 대상 추적 조정 정책에 대한 `ALBRequestCountPerTarget` 지표는 블루/그린 배포 유형을 지원하지 않습니다.

대상 추적 조정 정책에 대한 자세한 내용을 알아보려면 Application Auto Scaling 사용 설명서의 대상 추적 조정 정책을 참조하세요.

Amazon ECS 서비스 오토 스케일링에 대한 대상 추적 규모 조정 정책 생성

서비스에서 원하는 태스크 수가 Amazon ECS에서 자동으로 증가하거나 감소하는 대상 추적 규모 조정 정책을 생성합니다. 대상 추적은 대상 지표 값을 기반으로 작동합니다.

콘솔

1. 서비스를 생성 및 업데이트하는 표준 IAM 권한 외에도 추가 권한이 필요합니다. 자세한 내용은 Amazon ECS 서비스 Auto Scaling에 필요한 IAM 권한 섹션을 참조하세요.
2. 정책에 사용되는 지표를 결정합니다. 다음과 같은 지표를 사용할 수 있습니다.
   - ECSServiceAverageCPUUtilization - 서비스에서 사용되는 평균 CPU 사용률입니다.
   - ECSServiceAverageMemoryUtilization - 서비스에서 사용되는 평균 메모리 사용률입니다.
   - ALBRequestCountPerTarget - 작업에 이상적으로 수신되는 분당 평균 요청 수입니다.
3. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
4. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
5. 클러스터 세부 정보 페이지의 서비스 단원에서 서비스를 선택합니다.

   서비스 세부 정보 페이지가 나타납니다.
6. 태스크 수 설정을 선택합니다.
7. Amazon ECS 서비스 태스크 수에서 오토 스케일링 사용을 선택합니다.

   태스크 수 단원이 나타납니다.

   a. 작업의 최소 개수에 서비스 Auto Scaling에서 사용할 작업 수의 하한을 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.

   b. 사용할 서비스 오토 스케일링에 대한 태스크 수의 상한을 최대에 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.

   c. 저장을 선택합니다.

      정책 페이지가 나타납니다.
8. 규모 조정 정책 생성을 선택합니다.

   정책 생성 페이지가 나타납니다.
9. 조정 정책 유형(Scaling policy type)에서 대상 추적(Target tracking)을 선택합니다.
10. 정책 이름(Policy name)에 정책 이름을 입력합니다.
11. 지표 유형에는 옵션 목록 중에서 지표를 선택합니다.
12. 목표 사용률에는 Amazon ECS에서 유지 관리되어야 하는 태스크의 백분율에 대한 대상 값을 입력합니다. 서비스 오토 스케일링에서는 평균 사용률이 목표 사용률에 도달하거나 사용자가 지정한 최대 태스크 수에 도달할 때까지 용량이 스케일 아웃됩니다.

13. 추가 설정에서 다음을 수행합니다

    a. 스케일 인 휴지 기간에는 스케일 인 활동 완료 후 다른 스케일 인 활동이 시작될 수 있는 시간(초)을 입력합니다.

    b. 스케일 아웃 휴지 기간에는 이전 스케일 아웃 활동이 적용될 때까지 대기하는 시간(초)을 입력합니다.

    c. 스케일 아웃 정책만 생성하려면 스케일 인 비활성화를 선택합니다.

14. 규모 조정 정책 생성을 선택합니다.

AWS CLI

1. register-scalable-target 명령을 사용하여 Amazon ECS 서비스를 조정 가능 대상으로 등록합니다.
2. put-scaling-policy 명령을 사용하여 조정 정책을 생성합니다.

CloudWatch 경보를 기반으로 미리 정의된 증분을 사용하여 Amazon ECS 서비스 규모 조정

단계 조정 정책을 사용하여 조정 프로세스를 호출하는 경보를 생성하고 관리합니다. 경보 위반이 발생하면 해당 경보와 관련된 규모 조정 정책이 Amazon ECS에서 시작됩니다. 단계 규모 조정 정책에서는 단계 조정이라는 조정 세트를 통해 태스크 규모가 조정됩니다. 조정 크기는 경보 위반 규모에 따라 다릅니다.

- 첫 번째 임계값을 초과한 위반의 경우 Amazon ECS에서는 첫 번째 단계 조정이 적용됩니다.
- 두 번째 임계값을 초과한 위반의 경우 Amazon ECS에서는 두 번째 단계 조정이 적용되는 식으로 진행됩니다.

목표 추적 조정 정책을 사용하여 목표당 평균 요청 수 또는 평균 CPU 사용률과 같은 지표에 따라 조정하는 것이 좋습니다 용량이 증가할 때 감소하고 용량이 감소할 때 증가하는 지표를 사용하여 비례적으로 스케일 아웃하거나 대상 추적을 사용하여 작업 수를 늘릴 수 있습니다. 이는 애플리케이션에 대한 수요 곡선을 Amazon ECS에서 긴밀하게 따르는 데 도움이 됩니다.

Amazon ECS 서비스 오토 스케일링에 대한 단계 규모 조정 정책 생성

서비스에서 원하는 태스크 수가 Amazon ECS에서 자동으로 증가하거나 감소하는 단계 규모 조정 정책을 생성합니다. 단계 규모 조정은 경보 위반의 크기에 따라 달라지는 단계 조정이라는 규모 조정 세트를 기반으로 실행됩니다.

콘솔

1. 서비스를 생성 및 업데이트하는 표준 IAM 권한 외에도 추가 권한이 필요합니다. 자세한 내용은 Amazon ECS 서비스 Auto Scaling에 필요한 IAM 권한 섹션을 참조하세요.
2. 정책에 사용되는 지표를 결정합니다. 다음과 같은 지표를 사용할 수 있습니다.

   - ECSServiceAverageCPUUtilization - 서비스에서 사용되는 평균 CPU 사용률입니다.
   - ECSServiceAverageMemoryUtilization - 서비스에서 사용되는 평균 메모리 사용률입니다.
   - ALBRequestCountPerTarget - 작업에 이상적으로 수신되는 분당 평균 요청 수입니다.
3. 지표에 대한 CloudWatch 경보를 생성합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 정적 임계값을 기반으로 CloudWatch 경보 생성을 참조하세요.
4. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
5. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
6. 클러스터 세부 정보 페이지의 서비스 단원에서 서비스를 선택합니다.

   서비스 세부 정보 페이지가 나타납니다.
7. 태스크 수 설정을 선택합니다.
8. Amazon ECS 서비스 태스크 수에서 오토 스케일링 사용을 선택합니다.

   태스크 수 단원이 나타납니다.

   a. 작업의 최소 개수에 서비스 Auto Scaling에서 사용할 작업 수의 하한을 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.
   b. 사용할 서비스 오토 스케일링에 대한 태스크 수의 상한을 최대에 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.
   c. 저장을 선택합니다.

      정책 페이지가 나타납니다.
9. 규모 조정 정책 생성을 선택합니다.

   정책 생성 페이지가 나타납니다.
10. 규모 조정 정책 유형에서 단계 규모 조정을 선택합니다.
11. 스케일 아웃 속성을 구성합니다. 태스크에 추가되는 단계에서 다음을 수행합니다.

    a. 정책 이름(Policy name)에 정책 이름을 입력합니다.
    b. CloudWatch 경보 이름에서 CloudWatch 경보를 선택합니다.

c. 지표 집계 유형에는 선택한 지표가 정의된 임계값과 비교되는 방식을 선택합니다.
d. 조정 유형에는 조정이 태스크 수 변경 또는 작업 백분율 변경에 따라 달라지는지 여부를 선택합니다.
e. 취할 조치에는 취할 조치의 값을 입력합니다.

단계 추가를 선택하여 추가 조치를 추가합니다.

12. 스케일 인 속성을 구성합니다. 태스크 제거 단계에서 다음을 수행합니다.

a. 정책 이름(Policy name)에 정책 이름을 입력합니다.
b. CloudWatch 경보 이름에서 CloudWatch 경보를 선택합니다.
c. 지표 집계 유형에는 선택한 지표가 정의된 임계값과 비교되는 방식을 선택합니다.
d. 조정 유형에는 조정이 태스크 수 변경 또는 작업 백분율 변경에 따라 달라지는지 여부를 선택합니다.
e. 취할 조치에는 취할 조치의 값을 입력합니다.

단계 추가를 선택하여 추가 조치를 추가합니다.

13. 휴지 기간에 이전 조정 활동이 적용될 때까지 대기하는 시간(초)을 입력합니다. 추가 정책의 경우 이 시간은 스케일 아웃 활동 이후 조정 정책이 스케일 인 활동을 차단하고 한 번에 스케일 아웃할 수 있는 작업 수를 제한하는 시간입니다. 제거 정책의 경우 스케일 인 활동 이후 다른 스케일 인 활동을 시작하기 전까지 경과해야 하는 시간입니다.
14. 규모 조정 정책 생성을 선택합니다.

AWS CLI

1. register-scalable-target 명령을 사용하여 Amazon ECS 서비스를 조정 가능 대상으로 등록합니다.
2. put-scaling-policy 명령을 사용하여 조정 정책을 생성합니다.

예약된 조치를 사용하여 Amazon ECS 서비스 규모 조정

예약된 조정을 사용하면 특정 시간에 태스크 수가 증가하거나 감소하는 예약된 조치를 생성하여 예측 가능한 로드 변경을 기반으로 애플리케이션의 자동 규모 조정을 설정할 수 있습니다. 이를 통해 예측 가능한 부하 변화에 맞춰 애플리케이션 규모를 사전에 조정할 수 있습니다.

이러한 예약된 규모 조정 작업을 통해 비용과 성능을 최적화할 수 있습니다. 주중 트래픽 피크를 처리할 충분한 태스크 수가 애플리케이션에 있지만, 다른 시간에는 태스크가 과도하게 프로비저닝되지 않습니다.

예약된 조정 및 조정 정책을 함께 사용하면 규모 조정에 대한 예방적 및 대응적 접근 방식의 이점을 모두 얻을 수 있습니다. 예약된 규모 조정 조치 실행 후 규모 조정 정책에서는 태스크 수의 추가 규모 조정 여부에 대한 결정이 계속될 수 있습니다. 이는 애플리케이션의 로드를 처리할 충분한 태스크 수를 확보하는 데 도움이 됩니다. 수요에 일치하도록 애플리케이션 규모가 조정되는 동안 현재 용량은 예약한 조치 따라 설정된 설정한 최소 및 최대 태스크 수 범위여야 합니다.

AWS CLI를 사용하여 일정 조정을 구성할 수 있습니다. 예약된 조정을 사용하는 방법에 대한 자세한 내용은 Application Auto Scaling 사용 설명서의 Scheduled Scaling을 참조하세요.

Amazon ECS 서비스 규모 조정에 대한 예약된 조치 생성

날짜 및 시간을 기준으로 서비스가 실행되는 태스크 수가 Amazon ECS에서 증가하거나 감소하는 예정된 조치를 생성합니다.

콘솔

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
3. 클러스터 세부 정보 페이지의 서비스 단원에서 서비스를 선택합니다.

   서비스 세부 정보 페이지가 나타납니다.
4. 서비스 오토 스케일링을 선택합니다.

   서비스 오토 스케일링 페이지가 나타납니다.
5. 서비스 오토 스케일링을 구성하지 않은 경우 작업 수 설정을 선택합니다.

   Amazon ECS 서비스 작업 수 섹션이 나타납니다.

   Amazon ECS 서비스 작업 수에서 서비스 오토 스케일링 사용을 선택하여 서비스의 원하는 작업 수를 조정합니다.

   태스크 수 단원이 나타납니다.

   a. 작업의 최소 개수에 서비스 Auto Scaling에서 사용할 작업 수의 하한을 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.
   b. 사용할 서비스 오토 스케일링에 대한 태스크 수의 상한을 최대에 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.
   c. 저장 선택을 선택합니다.

      정책 페이지가 나타납니다.

6. 예약된 작업과 생성을 차례로 선택합니다.

   예약된 작업 생성 페이지가 나타납니다.
7. 액션 이름에 고유한 이름을 입력합니다.
8. 시간대에서 시간대를 선택합니다.

   나열된 모든 표준 시간대는 IANA 표준 시간대 데이터베이스에서 가져온 것입니다. 자세한 내용은 tz 데이터베이스 시간대 목록을 참조하세요.
9. 시작 시간에는 작업이 시작되는 날짜와 시간을 입력합니다.

   반복되는 일정을 선택한 경우, 시작 시간은 일련의 반복에서 첫 번째 예약된 작업이 실행되는 시점을 정의합니다.
10. Recurrence(반복)에서 사용 가능한 옵션 중 하나를 선택합니다.

    - 반복되는 일정에 따라 규모를 조정하려면 예약된 작업이 Amazon ECS에서 실행되는 빈도를 선택합니다.
      - Rate로 시작하는 옵션을 선택하면 cron 표현식이 자동으로 생성됩니다.
      - Cron을 선택하는 경우, 작업을 수행하는 시기를 지정하는 Cron 식을 입력합니다.
    - 한 번만 규모를 조정하려면 한 번을 선택합니다.
11. 태스크 조정에서 다음을 수행합니다.

    - 최소에는 서비스가 실행되어야 하는 최소 태스크 수를 입력합니다.
    - 최대에는 서비스가 실행되어야 하는 최대 작업 수를 입력합니다.
12. 예약된 작업 생성을 선택합니다.

CLI

다음과 같이 AWS CLI를 사용하여 서비스의 예약된 규모 조정 정책을 구성합니다. *user input placeholder*를 사용자의 정보로 바꿉니다.

예: 일회성 조정

`--start-time "YYYY-MM-DDThh:mm:ssZ"`와 `--MinCapacity` 및 `--MaxCapacity` 옵션 중 하나 또는 둘 다와 함께 다음 put-scheduled-action 명령을 사용합니다.

```
aws application-autoscaling put-scheduled-action --service-namespace ecs \
  --resource-id service/my-cluster/my-service \
```

```
  --scheduled-action-name my-one-time-schedule \
  --start-time 2021-01-30T12:00:00 \
  --scalable-target-action MinCapacity=3,MaxCapacity=10
```

예: 반복되는 일정으로 조정 예약

다음 put-scheduled-action 명령을 사용합니다. 모든 ### ##을 사용자의 값으로 바꿉니다.

```
aws application-autoscaling put-scheduled-action --service-namespace ecs \
  --resource-id service/my-cluster/my-service \
  --scheduled-action-name my-recurring-action \
  --schedule "rate(5 hours)" \
  --start-time 2021-01-30T12:00:00 \
  --end-time 2021-01-31T22:00:00 \
  --scalable-target-action MinCapacity=3,MaxCapacity=10
```

지정된 반복 일정은 UTC 시간대를 기반으로 실행됩니다. 다른 시간대를 지정하려면 `--time-zone` 옵션을 포함하고 IANA 시간대의 이름을 다음 예와 같이 지정합니다.

```
--time-zone "America/New_York"
```

자세한 내용은 tz 데이터베이스 시간대 목록을 참조하세요.

과거 패턴을 사용하여 예측 규모 조정으로 Amazon ECS 서비스 규모 조정

예측 규모 조정에서는 트래픽 흐름의 과거 로드 데이터 조사를 통해 일간 또는 주간 패턴이 분석됩니다. 그런 다음에 이 분석이 향후 필요성을 예측하고 필요에 따라 서비스의 태스크 미리 늘리는 데 사용됩니다.

예측 오토 스케일링은 다음과 같은 상황에서 가장 유용합니다.

- 주기적 트래픽 - 정규 업무 시간에는 리소스 사용이 증가하고 저녁과 주말에는 리소스 사용이 감소했습니다.
- 반복적인 온/오프 워크로드 패턴 - 배치 처리, 테스트 또는 주기적 데이터 분석이 예에 포함됩니다.
- 초기화 시간이 오래 걸리는 애플리케이션 - 현저한 대기 시간의 원인이 되는 스케일 아웃 이벤트 동안 애플리케이션 성능에 영향을 줄 수 있습니다.

규칙적인 패턴으로 애플리케이션 초기화에 시간이 오래 걸리고 트래픽이 증가하는 경우 예측 규모 조정 사용을 고려해야 합니다. 대상 추적 또는 단계 규모 조정과 같은 동적 조정 정책을 단독으로 사용하

는 대신에 예상되는 로드에 대한 태스크 수를 사전에 늘려 더 빠르게 규모를 조정할 수 있습니다. 태스크 수를 과도하게 프로비저닝할 가능성을 방지하는 데 도움이 되므로 예측 규모 조정을 통해 잠재적으로 비용도 절감할 수도 있습니다.

예컨대, 업무 시간 중에는 사용량이 많고 밤새 사용량이 줄어드는 애플리케이션을 생각해 보겠습니다. 각 영업일이 시작될 때 예측 규모 조정은 트래픽이 처음 유입되기 전에 태스크를 스케일 아웃할 수 있습니다. 이를 통해 사용률이 낮은 기간에서 높은 기간으로 전환될 때 애플리케이션이 고가용성과 성능을 유지할 수 있습니다. 동적 조정이 변화하는 트래픽에 대응할 때까지 기다릴 필요가 없습니다. 또한 애플리케이션의 로드 패턴을 검토하고 예약된 조정을 사용하여 적절한 태스크를 예약하는 데 시간을 할애할 필요가 없습니다.

예측 크기 조정은 기본 컴퓨팅 용량(예: EC2 또는 Fargate)의 규모 조정과 독립적으로 서비스의 태스크 규모를 조정하는 서비스 수준 기능입니다. Fargate의 경우, AWS는 태스크 요구 사항에 따라 기본 용량을 관리하고 자동으로 규모를 조정합니다. EC2 용량의 경우, Auto Scaling 그룹 용량 공급자를 사용하여 태스크의 규모 조정 요구 사항에 따라 기본 EC2 인스턴스 규모를 자동으로 조정할 수 있습니다.

내용

- Amazon ECS에서 예측 크기 조정이 작동하는 방식
- Amazon ECS 서비스 오토 스케일링에 대한 예측 규모 조정 정책 생성
- Amazon ECS에 대한 예측 규모 조정 정책 평가
- 예약된 작업을 사용하여 Amazon ECS의 예측 값 재정의
- Amazon ECS의 사용자 정의 지표를 사용하는 고급 예측 규모 조정 정책

Amazon ECS에서 예측 크기 조정이 작동하는 방식

여기에서 예측 규모 조정 사용에 대한 고려 사항, 작동 방식 및 제한 사항에 대해 알아볼 수 있습니다.

예측 규모 조정 사용 시 고려 사항

- 예측 규모 조정이 워크로드에 적합한지 확인하는 것이 좋습니다. 예측 전용 모드에서 규모 조정 정책을 구성하여 이를 확인하고 콘솔의 권장 사항을 참조할 수 있습니다. 예측 규모 조정을 사용하기 전에 예측 및 권장 사항을 평가해야 합니다.
- 예측 규모 조정에서 예측이 시작될 수 있으려면 최소 24시간 동안의 기록 데이터가 필요합니다. 사용 가능한 기록 데이터가 많을수록 예측이 더 효과적이며 2주 분량이 이상적입니다. 또한 Amazon ECS 서비스를 삭제하고 새 서비스를 생성할 때 예측 규모 조정에서 새 예측이 생성될 수 있도록 24시간을 기다려야 합니다. 속도를 높이는 한 가지 방법은 사용자 지정 지표를 사용하여 이전 및 새 Amazon ECS 서비스 전체의 지표를 집계하는 것입니다.

- 애플리케이션의 전체 로드를 정확하게 나타내고 확장해야 하는 애플리케이션의 측면인 부하 지표를 선택합니다.
- 예측 규모 조정을 통한 동적 규모 조정은 애플리케이션에 대한 수요를 면밀히 따라가는 데 도움이 되므로 소강 상태인 동안 스케일 인하고 예기치 않게 트래픽이 증가하는 동안 스케일 아웃할 수 있습니다. 여러 규모 조정 정책이 활성화된 경우, 각 정책에서는 원하는 태스크 수가 독립적으로 결정되고 원하는 태스크 수가 최댓값으로 설정됩니다.
- 실시간 패턴과 과거 패턴을 모두 기반으로 하여 애플리케이션 규모가 조정되도록 대상 추적 또는 단계 규모 조정과 같은 동적 규모 조정 정책과 함께 예측 규모 조정을 사용할 수 있습니다. 예측 규모 조정 자체로는 태스크가 스케일 인되지 않습니다.
- `register-scalable-target` API를 직접적으로 호출할 때 사용자 지정 역할을 사용하는 경우 활성화된 SLR만 예측 규모 조정 정책과 연동한다는 오류가 발생할 수 있습니다. 이 경우에는 role-arn이 없는 `register-scalable-target`을 다시 직접적으로 호출해야 합니다. 확장 가능한 대상을 등록할 때 SLR을 사용하고 `put-scaling-policy` API를 직접적으로 호출합니다.

예측 조정의 작동 방식

모니터링하고 분석할 CloudWatch 지표가 지정되는 예측 규모 조정 정책을 생성하여 예측 규모 조정을 사용합니다. 예측 규모 조정에서 미래 값 예측이 시작되려면 예측 규모 조정에 24시간 이상의 데이터가 있어야 합니다.

정책을 생성한 후 예측 조정은 패턴을 식별하기 위해 지난 14일까지의 지표 데이터를 분석하기 시작합니다. 이 분석은 향후 48시간의 요구 사항 시간별 예측 생성에 사용됩니다. 최신 CloudWatch 데이터를 사용하여 6시간마다 예측이 업데이트됩니다. 새로운 데이터가 들어오면 예측 규모 조정을 통해 향후 예측의 정확도가 지속적으로 개선됩니다.

예측 조정을 처음 활성화하면 예측 전용 모드에서 실행됩니다. 이 모드로 예측이 생성되지만, 해당 예측을 기반으로 Amazon ECS 서비스의 규모가 조정되지는 않습니다. 따라서 예측의 정확성과 적합성을 평가할 수 있습니다. `GetPredictiveScalingForecast` API 작업 또는 AWS Management Console을 사용하여 예측 데이터를 봅니다.

예측 규모 조정 사용을 시작하기로 결정하면 규모 조정 정책을 예측 및 규모 조정 모드로 전환합니다. 이 모드에서는 다음과 같이 진행됩니다.

기본적으로 해당 시간의 예측을 기반으로 매시간 시작 시 Amazon ECS 서비스의 규모가 조정됩니다. `PutScalingPolicy` API 작업의 `SchedulingBufferTime` 속성을 사용하여 더 일찍 시작하도록 선택할 수 있습니다. 그러면 예측된 수요에 앞서 새 태스크가 시작되고 부팅되어 트래픽을 처리할 준비가 될 시간이 주어집니다.

최대 태스크 한도

규모 조정을 위해 Amazon ECS 서비스를 등록할 때 서비스당 시작할 수 있는 최대 태스크 수를 정의합니다. 기본적으로 규모 조정 정책이 설정되면 태스크 수를 최대한도보다 높게 늘릴 수 없습니다.

그 대신에 예측이 Amazon ECS 서비스의 최대 태스크 수에 근접하거나 이를 초과하는 경우 서비스의 최대 태스크 수가 자동으로 증가하도록 허용할 수 있습니다.

> **Warning**
>
> 최대 태스크 수가 자동으로 증가하도록 허용할 때는 주의해야 합니다. 증가한 최대 태스크 수를 모니터링하고 관리하지 않으면 의도한 것보다 더 많은 태스크가 시작될 수 있습니다. 그러면 증가한 최대 태스크 수는 수동으로 업데이트할 때까지 Amazon ECS 서비스의 새로운 정상 최대 태스크 수가 됩니다. 최대 태스크 수는 원래 최대 태스크 수로 자동으로 다시 감소하지 않습니다.

지원되는 리전

- 미국 동부(버지니아 북부)
- 미국 동부(오하이오)
- 미국 서부(캘리포니아 북부)
- 미국 서부(오리건)
- 아프리카(케이프타운)
- 아시아 태평양(홍콩)
- 아시아 태평양(자카르타)
- 아시아 태평양(뭄바이)
- 아시아 태평양(오사카)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 캐나다(중부)
- 중국(베이징)
- 중국(닝샤)

- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)
- 유럽(밀라노)
- 유럽(파리)
- 유럽(스톡홀름)
- 중동(바레인)
- 남아메리카(상파울루)
- AWS GovCloud(미국 동부)
- AWS GovCloud(미국 서부)

Amazon ECS 서비스 오토 스케일링에 대한 예측 규모 조정 정책 생성

기록 데이터를 기반으로 서비스가 실행되는 태스크 수가 Amazon ECS에서 증가하거나 감소하는 예측 규모 조정 정책을 생성합니다.

> **Note**
>
> 예측이 생성될 수 있으려면 새 서비스에서 최소 24시간의 데이터가 제공되어야 합니다.

콘솔

1. 서비스를 생성 및 업데이트하는 표준 IAM 권한 외에도 추가 권한이 필요합니다. 자세한 내용은 Amazon ECS 서비스 Auto Scaling에 필요한 IAM 권한 섹션을 참조하세요.
2. 정책에 사용되는 지표를 결정합니다. 다음과 같은 지표를 사용할 수 있습니다.

   - ECSServiceAverageCPUUtilization - 서비스에서 사용되는 평균 CPU 사용률입니다.
   - ECSServiceAverageMemoryUtilization - 서비스에서 사용되는 평균 메모리 사용률입니다.
   - ALBRequestCountPerTarget - 작업에 이상적으로 수신되는 분당 평균 요청 수입니다.

   그 대신에 사용자 지정 지표를 사용할 수 있습니다. 다음과 같은 값을 정의해야 합니다.

   - 로드 - 애플리케이션의 전체 로드를 정확하게 나타내는 지표이며, 규모 조정 시 가장 중요한 애플리케이션 측면입니다.

- 규모 조정 지표 - 애플리케이션에 이상적인 사용률에 대한 최상의 예측기입니다.

3. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
4. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
5. 클러스터 세부 정보 페이지의 서비스 단원에서 서비스를 선택합니다.

   서비스 세부 정보 페이지가 나타납니다.
6. 서비스 오토 스케일링을 선택한 다음 작업 수 설정을 선택합니다.
7. Amazon ECS 서비스 태스크 수에서 오토 스케일링 사용을 선택합니다.

   태스크 수 단원이 나타납니다.

   a. 작업의 최소 개수에 서비스 Auto Scaling에서 사용할 작업 수의 하한을 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.
   b. 사용할 서비스 오토 스케일링에 대한 태스크 수의 상한을 최대에 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.
   c. 저장을 선택합니다.

      정책 페이지가 나타납니다.
8. 규모 조정 정책 생성을 선택합니다.

   정책 생성 페이지가 나타납니다.
9. 규모 조정 정책 유형에는 예측 규모 조정을 선택합니다.
10. 정책 이름(Policy name)에 정책 이름을 입력합니다.
11. 지표 쌍에는 옵션 목록 중에서 지표를 선택합니다.

    Application Load Balancer request count per target(대상당 Application Load Balancer 요청 수)을 선택한 경우, Target group(대상 그룹)에서 대상 그룹을 선택합니다. 대상별 Application Load Balancer 요청 수는 서비스에 Application Load Balancer 대상 그룹을 연결한 경우에만 지원됩니다.

    사용자 지정 지표 쌍를 선택하는 경우 로드 지표 및 규모 조정 지표의 목록에서 개별 지표를 선택합니다.
12. 목표 사용률에는 Amazon ECS에서 유지 관리되어야 하는 태스크의 백분율에 대한 대상 값을 입력합니다. 서비스 오토 스케일링에서는 평균 사용률이 목표 사용률에 도달하거나 사용자가 지정한 최대 태스크 수에 도달할 때까지 용량이 스케일 아웃됩니다.

13. 규모 조정 정책 생성을 선택합니다.

AWS CLI

다음과 같이 AWS CLI를 사용하여 Amazon ECS 서비스에 대한 예측 규모 조정 정책을 구성합니다. *user input placeholder*를 사용자의 정보로 바꿉니다.

지정 가능한 CloudWatch 지표에 대한 자세한 내용은 Amazon EC2 Auto Scaling API 참조의 PredictiveScalingMetricSpecification을 참조하세요.

예제 1: 사전 정의된 메모리가 있는 예측 규모 조정 정책입니다.

다음은 사전 정의된 메모리 구성이 있는 예제 정책입니다.

```
cat policy.json
{
    "MetricSpecifications": [
        {
            "TargetValue": 40,
            "PredefinedMetricPairSpecification": {
                "PredefinedMetricType": "ECSServiceMemoryUtilization"
            }
        }
    ],
    "SchedulingBufferTime": 3600,
    "MaxCapacityBreachBehavior": "HonorMaxCapacity",
    "Mode": "ForecastOnly"
}
```

다음 예제에서는 지정된 구성 파일로 put-scaling-policy 명령을 실행하여 정책을 생성하는 것을 보여줍니다.

```
aws application-autoscaling put-scaling-policy \
--service-namespace ecs \
--region us-east-1 \
--policy-name predictive-scaling-policy-example \
--resource-id service/MyCluster/test \
--policy-type PredictiveScaling \
--scalable-dimension ecs:service:DesiredCount \
--predictive-scaling-policy-configuration file://policy.json
```

이 명령이 제대로 실행되면 정책의 ARN이 반환됩니다.

```
{
    "PolicyARN": "arn:aws:autoscaling:us-
east-1:012345678912:scalingPolicy:d1d72dfe-5fd3-464f-83cf-824f16cb88b7:resource/ecs/
service/MyCluster/test:policyName/predictive-scaling-policy-example",
    "Alarms": []
}
```

예제 2: 사전 정의된 CPU가 있는 예측 규모 조정 정책입니다.

다음은 사전 정의된 CPU 구성이 있는 예제 정책입니다.

```
cat policy.json
{
    "MetricSpecifications": [
        {
            "TargetValue": 0.00000004,
            "PredefinedMetricPairSpecification": {
                "PredefinedMetricType": "ECSServiceCPUUtilization"
            }
        }
    ],
    "SchedulingBufferTime": 3600,
    "MaxCapacityBreachBehavior": "HonorMaxCapacity",
    "Mode": "ForecastOnly"
}
```

다음 예제에서는 지정된 구성 파일로 put-scaling-policy 명령을 실행하여 정책을 생성하는 것을 보여줍니다.

```
aws aas put-scaling-policy \
--service-namespace ecs \
--region us-east-1 \
--policy-name predictive-scaling-policy-example \
--resource-id service/MyCluster/test \
--policy-type PredictiveScaling \
--scalable-dimension ecs:service:DesiredCount \
--predictive-scaling-policy-configuration file://policy.json
```

이 명령이 제대로 실행되면 정책의 ARN이 반환됩니다.

```
{
```

```
    "PolicyARN": "arn:aws:autoscaling:us-
east-1:012345678912:scalingPolicy:d1d72dfe-5fd3-464f-83cf-824f16cb88b7:resource/ecs/
service/MyCluster/test:policyName/predictive-scaling-policy-example",
    "Alarms": []
}
```

Amazon ECS에 대한 예측 규모 조정 정책 평가

예측 규모 조정 정책을 사용하여 서비스 규모를 조정하기 전에 Amazon ECS 콘솔에서 정책에 대한 권장 사항과 기타 데이터를 검토합니다. 이는 예측의 정확성이 확인될 때까지 예측적 조정 정책이 실제 용량을 조정하는 것을 원하지 않기 때문에 중요합니다.

서비스가 새로운 경우 첫 번째 예측을 생성하는 데 24시간이 걸립니다.

AWS에서는 예측을 생성할 때 기록 데이터를 사용합니다. 서비스에 아직 최근 기록 데이터가 많지 않은 경우, 예측 규모 조정에서는 현재 사용 가능한 기록 집계에서 생성된 집계로 예측을 일시적으로 채울 수 있습니다. 정책 생성 날짜 전 최대 2주 동안 예측이 채워집니다.

예측적 조정 권장 사항 보기

효과적인 분석을 위해 비교할 예측 규모 조정 정책이 2개 이상 서비스 오토 스케일링에 있어야 합니다. (그러나 여전히 단일 정책에 대한 결과를 검토할 수 있습니다.) 여러 정책을 생성할 때 한 지표를 사용하는 정책을 다른 지표를 사용하는 정책과 비교하여 평가할 수 있습니다. 또한 다양한 대상 값과 지표 조합의 영향을 평가할 수 있습니다. 예측 규모 조정 정책이 생성된 후 Amazon ECS에서는 즉시 어느 정책이 그룹 규모 조정 작업을 더 잘 수행할지 평가하기 시작합니다.

Amazon ECS 콘솔에서 권장 사항을 보는 방법

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
3. 클러스터 세부 정보 페이지의 서비스 단원에서 서비스를 선택합니다.

   서비스 세부 정보 페이지가 나타납니다.
4. 서비스 오토 스케일링을 선택합니다.
5. 예측 규모 조정 정책을 선택한 다음에 작업, 예측 규모 조정, 권장 사항 보기를 선택합니다.

   권장 사항과 함께 정책에 대한 세부 정보를 볼 수 있습니다. 권장 사항은 예측적 조정 정책이 사용하지 않는 것보다 더 나은 작업을 수행하는지 여부를 알려줍니다.

예측적 조정 정책이 그룹에 적합한지 확실하지 않은 경우, 가용성에 미치는 영향 열과 비용에 미치는 영향 열을 검토하여 적합한 정책을 선택합니다. 각 열에 대한 정보는 정책의 영향을 알려줍니다.

- 가용성에 미치는 영향: 정책을 사용하지 않을 때와 비교하여 정책을 통해 워크로드를 처리하기에 충분한 태스크를 프로비저닝함으로써 가용성에 미치는 부정적인 영향을 방지할 수 있는지 여부를 설명합니다.
- 비용에 미치는 영향: 정책을 사용하지 않을 때와 비교하여 정책을 통해 태스크를 과다 프로비저닝하지 않음으로써 비용에 미치는 부정적인 영향을 방지할 수 있는지 여부를 설명합니다. 과다 프로비저닝하면 서비스가 충분히 사용되지 않거나 유휴 상태가 되어 비용에 미치는 영향만 가중됩니다.

정책이 여러 개인 경우, 더 낮은 비용으로 가장 많은 가용성 이점을 제공하는 정책 이름 옆에 최상의 예측 태그가 표시됩니다. 가용성 영향에 더 많은 가중치가 부여됩니다.

6. (선택 사항) 권장 사항 결과에 대해 원하는 기간을 선택하려면 평가 기간 드롭다운에서 원하는 값(2일, 1주 또는 2주)을 선택합니다. 기본적으로 평가 기간은 지난 2주입니다. 평가 기간이 길수록 권장 사항 결과에 더 많은 데이터 포인트가 제공됩니다. 그러나 로드 패턴이 변경된 경우(예: 예외적인 수요 기간 이후) 데이터 포인트를 더 추가해도 결과가 개선되지 않을 수 있습니다. 이 경우, 보다 최근 데이터를 살펴봄으로써 보다 집중적인 권장 사항을 얻을 수 있습니다.

> **Note**
>
> 예상 전용 모드에 있는 정책에 대한 권장 사항만 생성됩니다. 권장 사항 기능은 평가 기간 내내 정책이 예상 전용 모드일 때 더 잘 작동합니다. 예상 및 조정 모드에서 정책을 시작하고 나중에 예상 전용 모드로 전환하면 해당 정책에 대한 결과가 편향될 가능성이 높습니다. 이는 정책이 이미 실제 용량에 기여했기 때문입니다.

예측적 조정 모니터링 그래프 검토

콘솔에서 이전 날, 주 또는 달의 예측을 검토하여 시간이 지남에 따라 정책이 얼마나 잘 수행되는지 시각화할 수 있습니다. 정책을 통해 실제 태스크 수 규모를 조정할지 여부를 결정할 때 이 정보를 사용하여 예측의 정확성을 평가할 수도 있습니다.

Amazon ECS 콘솔에서 예측 규모 조정 모니터링 그래프를 보는 방법

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
3. 클러스터 세부 정보 페이지의 서비스 단원에서 서비스를 선택합니다.

   서비스 세부 정보 페이지가 나타납니다.
4. 서비스 오토 스케일링을 선택합니다.
5. 예측 규모 조정 정책을 선택한 다음에 작업, 예측 규모 조정, 그래프 보기를 선택합니다.
6. 모니터링 섹션에서 로드 및 용량 정책에 대한 과거 및 미래 예측을 실제 값과 비교하여 볼 수 있습니다. 로드 그래프는 선택한 로드 지표에 대한 로드 예측 및 실제 값을 보여줍니다. 용량 그래프에는 정책에서 예측한 태스크 수가 표시됩니다. 시작된 실제 태스크 수도 포함되어 있습니다. 수직선은 과거 값과 미래 예측을 구분합니다. 정책이 생성되면 바로 이 그래프를 사용할 수 있습니다.
7. (옵션) 차트에 표시된 기록 데이터 양을 변경하려면 페이지 상단의 평가 기간 드롭다운에서 원하는 값을 선택합니다. 평가 기간은 어떤 식으로든 이 페이지의 데이터를 변환하지 않으며, 표시되는 기록 데이터의 양만 변경합니다.

로드 그래프의 데이터 비교

각 수평선은 1시간 간격으로 보고되는 서로 다른 데이터 포인트 세트를 나타냅니다.

1. 실제 관찰된 로드는 선택한 로드 지표에 대한 SUM 통계를 사용하여 과거의 총 시간별 로드를 보여줍니다.
2. 정책에 의해 예측되는 로드는 시간별 로드 예측을 보여줍니다. 이 예측은 지난 2주간의 실제 로드 관찰에 근거하여 합니다.

용량 그래프의 데이터 비교

각 수평선은 1시간 간격으로 보고되는 서로 다른 데이터 포인트 세트를 나타냅니다.

1. 실제 관찰된 태스크 수는 선택한 기간 동안 적용된 기타 규모 조정 정책과 최소 그룹 크기에 따라 달라지는 과거의 Amazon ECS 서비스 실제 용량을 보여줍니다.
2. 정책에서 예측한 용량은 정책이 예상 및 조정 모드일 때 매 시간 시작 시 예상할 수 있는 기본 용량을 보여줍니다.
3. 추론된 필수 태스크 수는 선택한 대상 값으로 규모 조정 지표를 유지 관리하는 데 이상적인 서비스의 태스크 수를 보여줍니다.

4. 최소 태스트 수는 서비스의 최소 태스크 수를 보여줍니다.

5. 최대 용량은 서비스의 최대 태스크 수를 보여줍니다.

추론된 필수 용량을 계산하기 위해 먼저 각 태스크가 지정된 대상 값에서 균등하게 사용된다고 가정합니다. 실제로 태스크 수는 균등하게 활용되지 않습니다. 그러나 사용률이 태스크 간에 균일하게 분산되어 있다고 가정하면 필요한 용량의 양을 예측할 수 있습니다. 그런 다음에 태스크 수 요구 사항은 예측 규모 조정 정책에 사용한 규모 조정 지표에 반비례하도록 계산됩니다. 즉, 태스크 수가 증가하면 동일한 비율로 규모 조정 지표가 감소합니다. 예를 들어, 태스크 수가 두 배로 증가하면 규모 조정 지표는 절반으로 감소해야 합니다.

추론된 필수 용량에 대한 공식:

```
sum of (actualServiceUnits*scalingMetricValue)/(targetUtilization)
```

예컨대, 지정된 시간에 대한 `actualServiceUnits`(`10`) 및 `scalingMetricValue`(`30`)를 가져옵니다. 그런 다음 예측적 조정 정책(`60`)에 지정한 `targetUtilization`을 가져오고 같은 시간에 대해 추론된 필수 용량을 계산합니다. 이 예는 값 `5`를 반환합니다. 즉, 조정 지표의 대상 값에 반비례하여 용량을 유지하는 데 필요한 추론 용량은 5입니다.

> **Note**
>
> 애플리케이션의 비용 절감 및 가용성을 조정하고 개선하기 위해 다양한 레버를 사용할 수 있습니다.
>
> - 기준 용량과 동적 조정에 예측적 조정을 사용하여 추가 용량을 처리합니다. 동적 조정은 예측적 조정과 독립적으로 작동하며 현재 사용률에 따라 스케일 아웃되고 축소됩니다. 먼저 Amazon ECS에서는 예약되지 않은 규모 조정 정책마다 권장 태스크 수를 계산합니다. 그런 다음에 최대 태스크 수가 제공되는 정책에 따라 규모가 조정됩니다.
> - 로드가 감소할 때 스케일 인이 발생하도록 하려면 스케일 인 부분이 활성화된 동적 조정 규모 정책이 항상 하나 이상 서비스에 있어야 합니다.
> - 최소 및 최대 용량이 너무 제한적이지 않은지 확인하여 조정 성능을 개선할 수 있습니다. 권장 태스크 수가 최소 및 최대 용량 범위에 속하지 않는 정책은 스케일 인 및 스케일 아웃되지 않습니다.

CloudWatch로 Amazon ECS에 대한 예측 규모 조정 지표 모니터링

Amazon CloudWatch를 사용하여 예측 규모 조정을 위한 데이터를 모니터링할 수 있습니다. 예측 규모 조정 정책에서는 향후 로드를 예측하는 데 사용되는 데이터를 수집합니다. 수집된 데이터는 정기적으로 CloudWatch에 자동으로 저장되며, 시간별로 정책이 얼마나 잘 작동하는지 시각화하는 데 사용할 수 있습니다. 정의한 한도 이상으로 성과 지표가 변화할 때 알려주는 CloudWatch 경보도 생성할 수 있습니다.

기록 예측 데이터 시각화

예측 규모 조정 정책의 로드 예측 데이터는 CloudWatch에서 볼 수 있으며, 단일 그래프에서 다른 CloudWatch 지표와 비교되는 예측을 시각화할 때 유용할 수 있습니다. 더 넓은 시간 범위를 보면서 시간별 추세를 참조할 수도 있습니다. 최대 15개월의 기록 지표에 액세스하여, 정책의 성능을 전반적으로 더 잘 파악할 수 있습니다.

CloudWatch 콘솔을 사용하여 기록 예측 데이터를 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 지표(Metrics)를 선택한 다음 모든 지표(All metrics)를 선택합니다.
3. Application Auto Scaling 지표 네임스페이스를 선택합니다.
4. 예측 규모 조정 로드 예측을 선택합니다.
5. 검색 필드에 예측 규모 조정 정책의 이름 또는 Amazon ECS 서비스 그룹의 이름을 입력한 다음에 Enter 키를 눌러 결과를 필터링합니다.
6. 측정치를 그래프로 표시하려면 측정치 옆에 있는 확인란을 선택합니다. 그래프 이름을 변경하려면 연필 아이콘을 선택합니다. 시간 범위를 변경하려면 제공되는 값 중 하나를 선택하거나 사용자 지정을 선택합니다. 자세한 설명은 Amazon CloudWatch 사용자 가이드에서 지표 그래프를 참조하세요.
7. 기간을 변경하려면 그래프로 표시된 지표 탭을 선택합니다. 열 머리글이나 개별 값을 선택한 후 다른 통계를 선택합니다. 각 지표의 어떤 통계든 선택할 수 있지만, 모든 통계가 PredictiveScalingLoadForecast 지표에 유용한 것은 아닙니다. 예컨대, Average, Minimum 및 Maximum 통계는 유용하지만 Sum 통계는 유용하지 않습니다.
8. 그래프에 다른 지표를 추가하려면 찾아보기(Browse)에서 모두(All)를 선택하고 특정 지표를 찾은 다음 해당 지표 옆에 있는 확인란을 선택합니다. 최대 10개의 지표를 추가할 수 있습니다.
9. (옵션) 이 그래프를 CloudWatch 대시보드에 추가하려면 작업(Actions), 대시보드에 추가(Add to dashboard)를 선택합니다.

지표 수식을 사용하여 정확도 지표 생성

지표 수식을 사용하면 여러 CloudWatch 지표를 쿼리하고 수학 표현식을 활용함으로써 이러한 지표에 근거하여 새로운 시계열을 만들 수 있습니다. CloudWatch 콘솔에서 결과 시계열을 시각화하고 대시보드에 추가할 수 있습니다. 지표 수학에 대한 자세한 설명은 Amazon CloudWatch 사용자 가이드에서 지표 수학 사용을 참조하세요.

지표 수식을 사용하면 예측 스케일 인을 위해 서비스 오토 스케일링에서 생성되는 데이터를 다양한 방식으로 그래프로 표시할 수 있습니다. 이는 시간 경과에 따른 정책 성능을 모니터링하고 지표 조합을 개선할 수 있는지 여부를 파악하는 데 도움이 됩니다.

예컨대, 지표 수학 표현식을 사용하여 평균 절대 백분율 오차(MAPE)를 모니터링할 수 있습니다. MAPE 지표는 예측된 값과 지정된 예측 기간 동안 관찰된 실제 값의 차이를 모니터링하는 데 유용합니다. MAPE 값의 변화는 시간 경과에 따라 애플리케이션의 특성이 변경되면서 정책의 성능이 저하되는지 여부를 나타낼 수 있습니다. MAPE가 증가하면 예측된 값과 실제 값의 차이가 더 커진다는 것을 나타냅니다.

예: 지표 수학 표현식

이 유형의 그래프로 시작하려면, 다음 예에 표시된 것과 같은 지표 수학 표현식을 생성하면 됩니다.

단일 지표 대신, `MetricDataQueries`의 지표 데이터 쿼리 구조의 배열을 사용할 수 있습니다. `MetricDataQueries`의 각 항목은 지표를 가져오거나 수학 표현식을 수행합니다. 첫 번째 항목 `e1`은 수학 표현식입니다. 이 지정된 표현식은 `ReturnData` 파라미터를 `true`로 설정합니다. 이는 궁극적으로 단일 시계열을 생성합니다. 다른 모든 지표의 경우, `ReturnData` 값은 `false`입니다.

이 예에서 지정된 표현식은 실제 값과 예측된 값을 입력으로 사용하여 새 지표(MAPE)를 반환합니다. `m1`은 실제 로드 값을 포함하는 CloudWatch 지표입니다(CPU 사용률이 `my-predictive-scaling-policy`라는 정책에 대해 원래 지정된 로드 지표라고 가정). `m2`는 예측된 로드 값을 포함하는 CloudWatch 지표입니다. MAPE 지표의 수식 구문은 다음과 같습니다.

(abs ((실제 - 예측)/(실제)))의 평균

정확도 지표 시각화 및 경보 설정

정확도 지표 데이터를 시각화하려면 CloudWatch 콘솔에서 지표(Metrics) 탭을 선택합니다. 여기에서 데이터를 그래프로 표시할 수 있습니다. 자세한 설명은 Amazon CloudWatch 사용자 가이드에서 CloudWatch 그래프에 수학 표현식 추가를 참조하세요.

지표(Metrics) 섹션에서 모니터링하는 지표에 대해 경보를 설정할 수도 있습니다. 그래프로 표시된 지표(Graphed metrics) 탭에서 작업(Actions) 열 아래에 있는 경보 생성(Create alarm) 아이콘을 선택합

니다. 이 경보 생성(Create alarm) 아이콘은 작은 종 모양으로 표시됩니다. 자세한 내용과 알림 옵션은 Amazon CloudWatch 사용 설명서의 지표 수학 표현식에 근거하여 CloudWatch 경보 생성 및 경보 변경 시 사용자에게 알림을 참조하세요.

또는 GetMetricData 및 PutMetricAlarm을 사용하여 지표 수식으로 계산을 수행하고 그 출력에 따라 경보를 생성할 수 있습니다.

예약된 작업을 사용하여 Amazon ECS의 예측 값 재정의

경우에 따라 예상 계산에서 고려할 수 없는 향후 애플리케이션 필요량에 대한 추가 정보가 있을 수 있습니다. 예컨대, 예상 계산은 향후 마케팅 이벤트에 필요한 작업을 과소 평가할 수 있습니다. 예약된 작업을 사용하여 미래 기간에 대한 예상을 임시로 재정의할 수 있습니다. 예약된 작업은 반복적으로 실행되거나 일회성 수요 변동이 있는 특정 날짜 및 시간에 실행될 수 있습니다.

예컨대, 예상한 것보다 높은 작업 수로 예약된 작업을 생성할 수 있습니다. 런타임 시 Amazon ECS에서는 서비스의 최소 태스크 수를 업데이트합니다. 예측 조정은 작업 수에 맞게 최적화하므로 최소 작업 수가 예상 값보다 높은 예약된 작업이 적용됩니다. 이렇게 하면 작업 수가 예상보다 적지 않습니다. 예상 재정의를 중지하려면 두 번째 예약된 작업을 사용하여 최소 작업 수를 원래 설정으로 되돌립니다.

다음 절차에서는 미래 기간의 예상을 재정의하는 단계를 간략하게 설명합니다.

주제

- 1단계: (옵션) 시계열 데이터 분석
- 2단계: 2개의 예약된 작업 생성

> **Important**
>
> 이 주제에서는 예측보다 더 큰 용량으로 확장하기 위해 예측을 재정의한다고 가정합니다. 예측 조정 정책의 간섭 없이 일시적으로 작업 수를 줄여야 하는 경우 예측 전용 모드를 대신 사용합니다. 예측 전용 모드에서 예측 조정은 계속 예측을 생성하지만 작업 수가 자동으로 증가하지는 않습니다. 그러면 리소스 사용률을 모니터링하고 필요에 따라 태스크 수를 수동으로 줄일 수 있습니다.

1단계: (옵션) 시계열 데이터 분석

예상 시계열 데이터 분석으로 시작합니다. 이 단계는 선택 사항이지만 예상의 세부 정보를 파악하려는 경우, 유용합니다.

1. 예상 검색

   예상이 생성되면 예상에서 특정 기간을 쿼리할 수 있습니다. 쿼리의 목표는 특정 기간에 대한 시계열 데이터의 전체 보기를 얻는 것입니다.

   쿼리에는 최대 2일간의 미래 예상 데이터가 포함될 수 있습니다. 예측 조정을 잠시 사용한 경우에도 과거 예상 데이터에 액세스할 수 있습니다. 그러나 시작 시간과 해지 시간 사이의 최대 기간은 30일입니다.

   get-predictive-scaling-forecast AWS CLI 명령을 사용하여 예상을 얻으려면 명령에 다음 파라미터를 제공합니다.

   - `resource-id` 파라미터에 클러스터의 이름을 입력합니다.
   - `--policy-name` 파라미터에 정책 이름을 입력합니다.
   - 지정한 시간 또는 그 이후의 예상 데이터만 반환하려면 `--start-time` 파라미터에 시작 시간을 입력합니다.
   - 지정한 시간 이전의 예상 데이터만 반환하려면 `--end-time` 파라미터에 해지 시간을 입력합니다.

```
aws application-autoscaling get-predictive-scaling-forecast \
    --service-namespace ecs \
    --resource-id service/MyCluster/test \
    --policy-name cpu40-predictive-scaling-policy \
    --scalable-dimension ecs:service:DesiredCount \
    --start-time "2021-05-19T17:00:00Z" \
    --end-time "2021-05-19T23:00:00Z"
```

   이 작업이 성공하면 명령에서 다음 예와 비슷한 데이터가 반환됩니다.

```
{
    "LoadForecast": [
        {
            "Timestamps": [
                "2021-05-19T17:00:00+00:00",
                "2021-05-19T18:00:00+00:00",
                "2021-05-19T19:00:00+00:00",
                "2021-05-19T20:00:00+00:00",
                "2021-05-19T21:00:00+00:00",
                "2021-05-19T22:00:00+00:00",
```

```
                "2021-05-19T23:00:00+00:00"
            ],
            "Values": [
                153.0655799339254,
                128.8288551285919,
                107.1179447150675,
                197.3601844551528,
                626.4039934516954,
                596.9441277518481,
                677.9675713779869
            ],
            "MetricSpecification": {
                "TargetValue": 40.0,
                "PredefinedMetricPairSpecification": {
                    "PredefinedMetricType": "ASGCPUUtilization"
                }
            }
        }
    ],
    "CapacityForecast": {
        "Timestamps": [
            "2021-05-19T17:00:00+00:00",
            "2021-05-19T18:00:00+00:00",
            "2021-05-19T19:00:00+00:00",
            "2021-05-19T20:00:00+00:00",
            "2021-05-19T21:00:00+00:00",
            "2021-05-19T22:00:00+00:00",
            "2021-05-19T23:00:00+00:00"
        ],
        "Values": [
            2.0,
            2.0,
            2.0,
            2.0,
            4.0,
            4.0,
            4.0
        ]
    },
    "UpdateTime": "2021-05-19T01:52:50.118000+00:00"
}
```

응답에는 LoadForecast 및 CapacityForecast라는 두 가지 예측 값이 포함됩니다. LoadForecast는 시간별 로드 예측을 보여 주고 CapacityForecast는 40.0(평균 CPU 사용률 40%)의 TargetValue를 유지하는 동안 예측 로드를 처리하는 데 시간당 필요한 용량에 대한 예측 값을 보여 줍니다.

2. 대상 기간 식별

   일회성 수요 변동이 발생해야 하는 시간을 식별합니다. 예상에 표시된 날짜와 시간은 UTC입니다.

2단계: 2개의 예약된 작업 생성

다음으로, 애플리케이션의 예상 로드보다 높은 특정 기간에 대해 2개의 예약된 작업을 생성합니다. 예컨대, 제한된 기간에 사이트에 대한 트래픽을 높이는 마케팅 이벤트가 있는 경우, 이벤트 시작 시 최소 용량을 업데이트하는 일회성 작업을 예약할 수 있습니다. 그런 다음 이벤트가 해지 시 최소 용량을 원래 설정으로 되돌리도록 다른 작업을 예약합니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
3. 클러스터 세부 정보 페이지의 서비스 단원에서 서비스를 선택합니다.

   서비스 세부 정보 페이지가 나타납니다.
4. 서비스 오토 스케일링을 선택합니다.

   정책 페이지가 나타납니다.
5. 예약된 작업과 생성을 차례로 선택합니다.

   예약 생성 작업 페이지가 나타납니다.
6. 액션 이름에 고유한 이름을 입력합니다.
7. 시간대에서 시간대를 선택합니다.

   나열된 모든 표준 시간대는 IANA 표준 시간대 데이터베이스에서 가져온 것입니다. 자세한 내용은 tz 데이터베이스 시간대 목록을 참조하세요.
8. 시작 시간에는 작업이 시작되는 날짜와 시간을 입력합니다.
9. 반복에서 1회(Once)를 선택합니다.
10. 작업 조정의 최솟값에 최대 작업 수보다 적거나 같은 값을 입력합니다.
11. 예약된 작업 생성을 선택합니다.

정책 페이지가 나타납니다.

12. 이벤트 종료 시 최소 작업 수가 원래 설정으로 되돌아 가도록 두 번째 예약된 작업을 구성합니다. 예측 조정은 최소에 대해 사용자가 설정한 값이 예상 값보다 적은 경우에만 작업 수를 조정할 수 있습니다.

일회성 이벤트에 대해 2개의 예약된 작업을 생성하려면(AWS CLI)

AWS CLI를 사용하여 예약된 작업을 생성하려면 put-scheduled-update-group-action 명령을 사용합니다.

예컨대, 5월 19일 오후 5시에 8시간 동안 최소 용량으로 인스턴스 3개를 유지하는 일정을 정의해 보겠습니다. 다음 명령은 이 시나리오를 구현하는 방법을 보여 줍니다.

첫 번째 put-scheduled-update-group-action 명령은 2021년 5월 19일 오후 5시(UTC)에 지정된 Auto Scaling 그룹의 최소 용량을 업데이트하도록 Amazon EC2 Auto Scaling에 지시합니다.

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-event-
start \
  --auto-scaling-group-name my-asg --start-time "2021-05-19T17:00:00Z" --minimum-
capacity 3
```

두 번째 명령은 그룹의 최소 용량을 2021년 5월 20일 오전 1시(UTC)의 용량으로 설정하도록 Amazon EC2 Auto Scaling에 지시합니다.

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-event-end
 \
  --auto-scaling-group-name my-asg --start-time "2021-05-20T01:00:00Z" --minimum-
capacity 1
```

이러한 예약된 작업을 Auto Scaling 그룹에 추가하면 Amazon EC2 Auto Scaling은 다음을 수행하십시오:

- 2021년 5월 19일 오후 5시(UTC)에 첫 번째 예약된 작업이 실행됩니다. 현재 해당 그룹에 인스턴스가 3개 미만 있는 경우, 그룹은 인스턴스 3개로 스케일 아웃됩니다. 이 시간과 향후 8시간 동안 예측 용량이 실제 용량보다 높거나 동적 조정 정책이 적용되는 경우, Amazon EC2 Auto Scaling이 계속 스케일 아웃을 수행할 수 있습니다.
- 2021년 5월 20일 오전 1시(UTC)에 두 번째 예약된 작업이 실행됩니다. 이렇게 하면 이벤트 해지 시 최소 용량이 원래 설정으로 돌아갑니다.

반복 일정에 따라 크기 조정

매주 동일한 기간에 대한 예상을 재정의하려면 두 개의 예약된 작업을 생성하고 cron 표현식을 사용하여 시간 및 날짜 로직을 제공합니다.

cron 표현식 형식은 다음과 같이 공백으로 구분된 다섯 개의 필드로 구성됩니다. [Minute] [Hour] [Day_of_Month] [Month_of_Year] [Day_of_Week]. 필드에는 특수 문자를 포함하여 허용되는 모든 값을 포함할 수 있습니다.

예컨대, 다음 cron 표현식은 매주 화요일 오전 6시 30분에 작업을 실행합니다. 별표는 필드의 모든 값을 일치시키기 위한 와일드카드로 사용됩니다.

```
30 6 * * 2
```

다음 사항도 참조하세요.

예약된 작업을 관리하는 방법에 대한 자세한 내용은 예약된 조치를 사용하여 Amazon ECS 서비스 규모 조정 단원을 참조하세요.

Amazon ECS의 사용자 정의 지표를 사용하는 고급 예측 규모 조정 정책

미리 정의된 지표 또는 사용자 정의 지표를 예측 규모 조정 정책에서 사용할 수 있습니다. 사용자 지정 지표는 CPU, 메모리 등과 같이 미리 정의된 지표가 애플리케이션 로드를 충분히 설명하기에 충분하지 않을 때 유용합니다.

사용자 지정 지표를 사용하여 예측 규모 조정 정책을 생성할 때 AWS에서 제공하는 다른 CloudWatch 지표를 지정할 수 있습니다. 그 대신에 직접 정의하고 게시하는 지표를 지정할 수도 있습니다. 또한 지표 수학을 사용하여 기존 지표를 집계하고 AWS가 자동으로 추적하지 않는 새 시계열로 변환할 수 있습니다. 예를 들면, 집계라는 새 합계 또는 평균 계산을 통해 데이터의 값을 결합할 수 있습니다. 결과 데이터를 집계라고 합니다.

다음 섹션에는 정책의 JSON 구조를 구성하는 방법에 대한 모범 사례와 예가 나와 있습니다.

사전 조건

정책에서 사용자 지정 지표를 지정하려면 `cloudwatch:GetMetricData` 권한이 있어야 합니다.

AWS에서 제공하는 지표 대신 자체 지표를 지정하려면 먼저 지표를 CloudWatch에 게시해야 합니다. 자세한 설명은 Amazon CloudWatch 사용자 가이드의 사용자 정의 지표 게시를 참조하세요.

자체 지표를 게시하는 경우, 최소 5분 간격으로 데이터 요소를 게시해야 합니다. 데이터 포인트는 필요한 기간을 기준으로 CloudWatch에서 검색됩니다. 예컨대, 로드 지표 사양은 시간별 지표를 사용하여

애플리케이션의 로드를 측정합니다. CloudWatch는 게시된 지표 데이터로 각 1시간 기간에 속하는 타임스탬프가 있는 모든 데이터 요소를 집계하여 1시간 기간 동안 단일 데이터 값을 제공합니다.

모범 사례

다음 모범 사례는 사용자 정의 지표를 보다 효과적으로 사용하는 데 도움이 될 수 있습니다.

- 로드 지표 사양에 가장 유용한 지표는 Auto Scaling 그룹 전체의 로드를 나타내는 지표입니다.
- 규모 조정 지표 사양의 규모 조정에 가장 유용한 지표는 태스크 지표당 평균 처리량 또는 사용률입니다.
- 목표 사용률은 조정 지표 유형과 일치해야 합니다. 예를 들면, CPU 사용률을 사용하는 정책 구성의 경우에는 목표 백분율입니다.
- 이러한 권장 사항을 따르지 않으면 시계열의 예측된 미래 값이 정확하지 않을 수 있습니다. 데이터가 올바른지 검증하기 위해 콘솔에서 예측된 값을 볼 수 있습니다. 또는 예측 규모 조정 정책을 생성한 후 GetPredictiveScalingForecast API 직접적으로 호출에서 반환된 `LoadForecast` 객체를 검사합니다.
- 예측 규모 조정에서 능동적으로 규모 조정이 시작되기 전에 예상을 평가할 수 있게 예상 전용 모드에서 예측적 조정을 구성하는 것이 좋습니다.

제한 사항

- 하나의 지표 사양에서 최대 10개 지표의 데이터 요소를 쿼리할 수 있습니다.
- 이 제한을 위해 하나의 표현식이 하나의 지표로 계산됩니다.

사용자 지정 지표로 예측 규모 조정 정책 문제 해결

사용자 정의 지표를 사용하는 동안 문제가 발생하면 다음을 수행하는 것이 좋습니다.

- 검색 표현식을 사용하는 동안 블루/그린 배포에서 문제가 발생하는 경우 정확히 일치가 아니라 부분 일치를 찾는 검색 표현식을 생성해야 합니다. 또한 쿼리가 특정 애플리케이션에서 실행 중인 Auto Scaling 그룹만 찾는지 확인해야 합니다. 검색 표현식 구문에 대한 자세한 설명은 Amazon CloudWatch 사용자 가이드의 CloudWatch 검색 표현식 구문을 참조하세요.
- 조정 규모 정책을 생성할 때 put-scaling-policy 명령이 표현식을 검증합니다. 그러나 이 명령이 탐지된 오류의 정확한 원인을 식별하지 못할 수도 있습니다. 문제를 해결하려면 get-metric-data 명령에 대한 요청의 응답에서 수신하는 오류를 해결합니다. CloudWatch 콘솔에서 표현식의 문제를 해결할 수도 있습니다.

- `MetricDataQueries`가 SUM()과 같은 수학 함수 없이 자체적으로 SEARCH() 함수를 지정하는 경우, `ReturnData`에 대해 `false`를 지정해야 합니다. 이는 검색 표현식이 여러 시계열을 반환할 수 있고 표현식에 근거하여 하는 지표 사양이 하나의 시계열만 반환할 수 있기 때문입니다.
- 검색 표현식과 관련된 모든 지표는 해상도가 동일해야 합니다.

Amazon ECS로 예측 규모 조정 사용자 지정 지표를 위한 JSON 구성

다음 섹션에는 CloudWatch에서 데이터를 쿼리하도록 예측 조정을 구성하는 방법에 대한 예가 나와 있습니다. 이 옵션을 구성하는 방법에는 두 가지가 있으며 선택하는 방법에 따라 예측 조정 정책에 사용할 JSON을 구성하는 데 사용하는 형식이 달라집니다. 지표 수학을 사용하는 경우, JSON의 형식은 수행되는 지표 수학에 따라 더 달라집니다.

1. AWS에서 제공하는 다른 CloudWatch 지표 또는 CloudWatch에 게시하는 지표에서 직접 데이터를 가져오는 정책을 생성하려면 AWS CLI를 사용하는 사용자 지정 로드 및 규모 조정 지표가 있는 예측 규모 조정 정책 예제을 참조하세요.

AWS CLI를 사용하는 사용자 지정 로드 및 규모 조정 지표가 있는 예측 규모 조정 정책 예제

AWS CLI를 사용하여 사용자 지정 로드 및 조정 지표로 예측 조정 정책을 생성하려면 `config.json`이라는 JSON 파일에 `--predictive-scaling-configuration`에 대한 인수를 저장합니다.

다음 예에서 교체 가능한 값을 지표 및 목표 사용률의 값으로 교체하여 사용자 지정 지표를 추가하기 시작합니다.

```
{
  "MetricSpecifications": [
    {
      "TargetValue": 50,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "scaling_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "MyUtilizationMetric",
                "Namespace": "MyNameSpace",
                "Dimensions": [
                  {
                    "Name": "MyOptionalMetricDimensionName",
```

```
                    "Value": "MyOptionalMetricDimensionValue"
                  }
                ]
              },
              "Stat": "Average"
            }
          }
        ]
      },
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "MyLoadMetric",
                "Namespace": "MyNameSpace",
                "Dimensions": [
                  {
                    "Name": "MyOptionalMetricDimensionName",
                    "Value": "MyOptionalMetricDimensionValue"
                  }
                ]
              },
              "Stat": "Sum"
            }
          }
        ]
      }
    }
  ]
}
```

자세한 설명은 Amazon EC2 Auto Scaling API 참조의 MetricDataQuery를 참조하세요.

> **Note**
>
> 다음은 CloudWatch 지표에 대한 지표 이름, 네임스페이스, 차원 및 통계를 찾는 데 도움이 되는 몇 가지 추가 리소스입니다.
>
> - AWS 서비스에 사용 가능한 지표에 대한 자세한 설명은 Amazon CloudWatch 사용자 가이드의 CloudWatch 지표를 게시하는 AWS 서비스를 참조하세요.

- AWS CLI를 사용하여 CloudWatch 지표에 대한 정확한 지표 이름, 네임스페이스 및 차원(해당되는 경우)을 얻으려면 list-metrics를 참조하세요.

이 정책을 생성하려면 다음 예에 나와 있는 것처럼 JSON 파일을 입력으로 사용하여 put-scaling-policy 명령을 실행합니다.

```
aws application-autoscaling put-scaling-policy --policy-name my-predictive-scaling-policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
```

이 명령이 제대로 실행되면 정책의 Amazon 리소스 이름(ARN)을 반환합니다.

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-predictive-scaling-policy",
  "Alarms": []
}
```

지표 수학 표현식 사용

다음 단원에서는 정책에서 예측 규모 조정 정책과 지표 수학을 함께 사용하는 방법에 대한 정보가 제공됩니다.

지표 수학 이해

기존 지표 데이터를 집계하기만 하면 CloudWatch 지표 수학을 통해 CloudWatch에 다른 지표를 게시하는 데 드는 노력과 비용을 절약할 수 있습니다. AWS에서 제공하는 모든 지표를 사용할 수 있으며 애플리케이션의 일부로 정의하는 지표를 사용할 수도 있습니다.

자세한 설명은 Amazon CloudWatch 사용자 가이드의 지표 수학 사용을 참조하세요.

예측적 조정 정책에서 지표 수학 표현식을 사용하기로 선택한 경우, 다음 사항을 고려하세요.

- 지표 수학 연산은 고유한 조합의 지표 이름, 네임스페이스 및 지표의 차원 키/값 페어의 데이터 요소를 사용합니다.
- 산술 연산자(+ - * / ^), 통계 함수(예: AVG 또는 SUM) 또는 CloudWatch에서 지원하는 기타 함수를 사용할 수 있습니다.
- 수학 표현식의 공식에서 지표 및 다른 수학 표현식의 결과를 모두 사용할 수 있습니다.

- 지표 수학 표현식은 다양한 집계로 구성될 수 있습니다. 그러나 최종 집계 결과에서 조정 지표에는 `Average`를 사용하고 로드 지표에는 `Sum`을 사용하는 것이 가장 좋습니다.
- 지표 사양에 사용된 표현식은 결국 단일 시계열을 반환해야 합니다.

지표 수학을 사용하려면 다음을 수행하십시오:

- 하나 이상의 CloudWatch 지표를 선택합니다. 그런 다음 표현식을 생성합니다. 자세한 설명은 Amazon CloudWatch 사용자 가이드의 지표 수학 사용을 참조하세요.
- CloudWatch 콘솔 또는 CloudWatch GetMetricData API를 사용하여 지표 수학 표현식이 유효한지 확인합니다.

지표 수학을 사용하여 지표를 결합하는 예측 조정 정책의 예(AWS CLI)

지표를 직접 지정하는 대신 어떤 방식으로든 해당 데이터를 먼저 처리해야 하는 경우가 있습니다. 예컨대, Amazon SQS 대기열에서 작업을 가져오는 애플리케이션이 있고 대기열의 항목 수를 예측적 조정의 기준으로 사용할 수 있습니다. 대기열의 메시지 수가 필요한 인스턴스 수를 단독으로 정의하지 않습니다. 따라서 인스턴스당 백로그를 계산하는 데 사용할 수 있는 지표를 생성하려면 더 많은 작업이 필요합니다.

다음은 이 시나리오에 대한 예측적 조정 정책의 예입니다. 대기열에서 검색할 수 있는 메시지 수인 Amazon SQS `ApproximateNumberOfMessagesVisible` 지표를 기준으로 하는 조정 및 로드 지표를 지정합니다. 또한 Amazon EC2 Auto Scaling `GroupInServiceInstances` 지표와 수학 표현식을 사용하여 조정 지표에 대한 인스턴스당 백로그를 계산합니다.

```
aws application-autoscaling put-scaling-policy --policy-name my-sqs-custom-metrics-
policy \
  --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
  --service-namespace ecs \
  --resource-id service/MyCluster/test \
  "MetricSpecifications": [
    {
      "TargetValue": 100,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
            "Label": "Get the queue size (the number of messages waiting to be
 processed)",
            "Id": "queue_size",
```

```
            "MetricStat": {
              "Metric": {
                "MetricName": "ApproximateNumberOfMessagesVisible",
                "Namespace": "AWS/SQS",
                "Dimensions": [
                  {
                    "Name": "QueueName",
                    "Value": "my-queue"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": false
          },
          {
            "Label": "Get the group size (the number of running instances)",
            "Id": "running_capacity",
            "MetricStat": {
              "Metric": {
                "MetricName": "GroupInServiceInstances",
                "Namespace": "AWS/AutoScaling",
                "Dimensions": [
                  {
                    "Name": "AutoScalingGroupName",
                    "Value": "my-asg"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": false
          },
          {
            "Label": "Calculate the backlog per instance",
            "Id": "scaling_metric",
            "Expression": "queue_size / running_capacity",
            "ReturnData": true
          }
        ]
      },
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
```

```
            "Id": "load_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "ApproximateNumberOfMessagesVisible",
                "Namespace": "AWS/SQS",
                "Dimensions": [
                  {
                    "Name": "QueueName",
                    "Value": "my-queue"
                  }
                ],
              },
              "Stat": "Sum"
            },
            "ReturnData": true
          }
        ]
      }
    }
  ]
}
```

이 예에서는 정책의 ARN을 반환합니다.

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-
b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-sqs-custom-metrics-policy",
  "Alarms": []
}
```

Amazon ECS 서비스 상호 연결

Amazon ECS 작업에서 실행되는 애플리케이션은 종종 인터넷으로부터 연결을 받거나 Amazon ECS 서비스에서 실행되는 다른 애플리케이션에 연결해야 합니다. 인터넷을 통한 외부 연결이 필요한 경우 Elastic Load Balancing을 사용하는 것이 좋습니다. 통합 로드 밸런싱에 대한 자세한 내용은 the section called “로드 밸런싱을 사용하여 서비스 트래픽 분산” 섹션을 참조하세요.

애플리케이션이 Amazon ECS 서비스에서 실행되는 다른 애플리케이션에 연결해야 하는 경우 Amazon ECS에서는 다음과 같은 방법으로 로드 밸런서 없이 할 수 있습니다.

- Amazon ECS Service Connect

서비스 검색, 연결 및 트래픽 모니터링을 위한 Amazon ECS 구성을 제공하는 Service Connect를 사용하는 것이 좋습니다. Service Connect를 사용하면 애플리케이션dms 짧은 이름과 표준 포트를 사용하여 동일한 AWS 리전에 있는 여러 VCP에 있는 항목을 포함하여 동일한 클러스터, 다른 클러스터의 Amazon ECS 서비스에 연결할 수 있습니다.

Service Connect를 사용하면 Amazon ECS가 서비스 검색의 모든 부분을 관리합니다. 즉, 검색할 수 있는 이름을 생성하고, 작업이 시작 및 중지될 때 각 작업에 대한 항목을 동적으로 관리하며, 이름을 검색하도록 구성된 각 작업에서 에이전트를 실행할 수 있습니다. 애플리케이션은 DNS 이름에 대한 표준 기능을 사용하고 연결을 설정하여 이름을 조회할 수 있습니다. 애플리케이션이 이미 이 작업을 수행하는 경우 Service Connect를 사용하기 위해 애플리케이션을 수정할 필요가 없습니다.

각 서비스 및 작업 정의 내에 전체 구성을 제공합니다. Amazon ECS는 각 서비스 배포에서 이 구성의 변경 내용을 관리하여 배포의 모든 작업이 동일한 방식으로 작동하도록 합니다. 예를 들어, 서비스 검색으로 DNS에서 흔히 발생하는 문제는 마이그레이션 제어와 관련됩니다. 새 교체 IP 주소를 가리키도록 DNS 이름을 변경하는 경우 모든 클라이언트가 새 서비스를 사용하기 시작하는 데 최대 TTL 시간이 걸릴 수 있습니다. Service Connect를 사용하면 클라이언트 배포에서 클라이언트 작업을 교체하여 구성을 업데이트합니다. 다른 배포와 동일한 방식으로 Service Connect 변경 내용에 영향을 주도록 배포 회로 차단기 및 기타 배포 구성을 구성할 수 있습니다.

자세한 내용은 Service Connect를 사용하여 짧은 이름으로 Amazon ECS 서비스 연결 섹션을 참조하세요.

- Amazon ECS 서비스 검색

  서비스 간 통신의 또 다른 접근 방식으로, 서비스 검색을 사용하는 직접 통신이 있습니다. 이 접근 방식에서는 Amazon ECS와의 AWS Cloud Map 서비스 검색 통합을 사용할 수 있습니다. Amazon ECS는 서비스 검색을 사용하여 시작된 작업 목록을 AWS Cloud Map과 동기화합니다. 이 제품에서는 해당 특정 서비스에서 하나 이상 작업에 대한 내부 IP 주소로 확인되는 DNS 호스트 이름을 유지 관리합니다. Amazon VPC의 다른 서비스는 이 DNS 호스트 이름을 사용하여 내부 IP 주소를 통해 다른 컨테이너로 직접 트래픽을 보낼 수 있습니다.

  서비스 간 통신에 대한 이러한 접근 방식은 지연 시간을 줄여줍니다. 컨테이너 사이에 추가 구성 요소가 없습니다. 트래픽은 한 컨테이너에서 다른 컨테이너로 직접 이동합니다.

  이 접근 방식은 각 작업의 IP 주소가 고유한 `awsvpc` 네트워크 모드를 사용할 때 적합합니다. 대부분의 소프트웨어는 IP 주소로 직접 확인되는 DNS `A` 레코드 사용만 지원합니다. `awsvpc` 네트워크 모드를 사용하는 경우 각 작업의 IP 주소는 `A` 레코드입니다. 하지만 `bridge` 네트워크 모드를 사용하는 경우 여러 컨테이너가 동일한 IP 주소를 공유할 수 있습니다. 또한 동적 포트 매핑으로 해당 단일 IP 주소에서 컨테이너에 포트 번호가 무작위로 할당됩니다. 이 경우 `A` 레코드는 더 이상 서비스 검색

에 충분하지 않습니다. 또한 SRV 레코드를 사용해야 합니다. 이 유형의 레코드는 IP 주소와 포트 번호를 모두 추적할 수 있지만 애플리케이션을 적절하게 구성해야 합니다. 사용하는 일부 사전 구축된 애플리케이션은 SRV 레코드를 지원하지 않을 수 있습니다.

awsvpc 네트워크 모드의 또 다른 이점은 각 서비스에 대해 고유한 보안 그룹이 보유한다는 점입니다. 해당 서비스와 통신해야 하는 특정 업스트림 서비스의 수신 연결만 허용하도록 이 보안 그룹을 구성할 수 있습니다.

서비스 검색을 사용하는 서비스 간 직접 통신의 주된 단점은 재시도 및 연결 실패 처리를 위해 추가 로직을 구현해야 한다는 점입니다. DNS 레코드에는 캐싱되는 시간을 제어하는 TTL(Time To Live) 기간이 있습니다. 애플리케이션이 최신 버전의 DNS 레코드를 선택할 수 있도록 DNS 레코드를 업데이트하고 캐시가 만료되기까지 어느 정도 시간이 걸립니다. 따라서 애플리케이션이 DNS 레코드를 확인할 때 더 이상 존재하지 않는 다른 컨테이너를 가리킬 수 있습니다. 애플리케이션은 재시도를 처리하고 잘못된 백엔드를 무시할 수 있는 로직을 포함해야 합니다.

자세한 내용은 서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결 섹션을 참조하세요.

- Amazon VPC Lattice

  Amazon VPC Lattice는 Amazon ECS 고객이 코드를 수정하지 않고 AWS 컴퓨팅 서비스, VPC 및 계정에 구축된 애플리케이션을 관찰, 보안 유지 및 모니터링하는 데 사용하는 관리형 애플리케이션 네트워킹 서비스입니다.

  VPC Lattice에서는 컴퓨팅 리소스 모음인 대상 그룹을 사용합니다. 이러한 대상은 애플리케이션 또는 서비스를 실행하며, Amazon EC2 인스턴스, IP 주소, Lambda 함수 및 Application Load Balancer일 수 있습니다. 이제는 고객이 Amazon ECS 서비스를 VPC Lattice 대상 그룹과 연결하여 Amazon ECS 작업을 VPC Lattice의 IP 대상으로 활성화할 수 있습니다. Amazon ECS에서는 등록된 서비스에 대한 태스크가 시작될 때 VPC Lattice 대상 그룹에 태스크를 자동으로 등록합니다.

  자세한 내용은 Amazon VPC Lattice를 사용하여 Amazon ECS 서비스 연결, 관찰 및 보안 유지 섹션을 참조하세요.

네트워크 모드 호환성 표

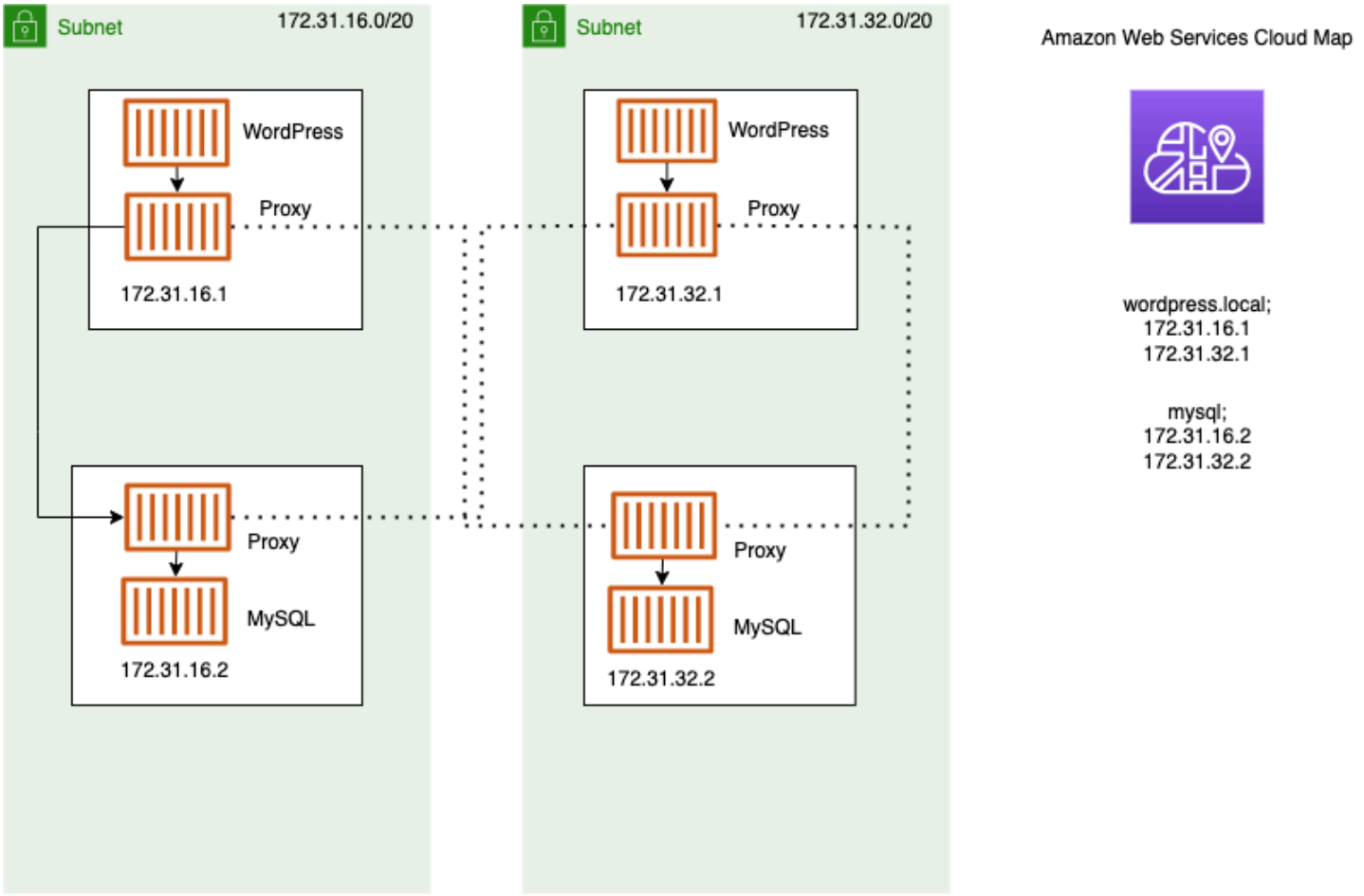
다음 표에서는 이러한 옵션과 작업 네트워크 모드 간의 호환성에 대해 설명합니다. 표에서 '클라이언트'는 Amazon ECS 작업 내에서 연결하는 애플리케이션을 가리킵니다.

상호 연결 옵션	브리징	**awsvpc**	Host
서비스 검색	예, 하지만 클라이언트는 `hostPort` 없이 DNS에서 SRV 레코드를 인식해야 합니다.	yes	예, 하지만 클라이언트는 `hostPort` 없이 DNS에서 SRV 레코드를 인식해야 합니다.
Service Connect	예	예	아니요
VPC Lattice	예	예	예

Service Connect를 사용하여 짧은 이름으로 Amazon ECS 서비스 연결

Amazon ECS Service Connect는 서비스 간 통신 관리를 Amazon ECS 구성으로 제공합니다. Amazon ECS에서 서비스 검색과 서비스 메시를 모두 구축합니다. 이를 통해 서비스 배포에서 관리하는 각 서비스 내부의 전체 구성, VPC DNS 구성에 의존하지 않는 네임스페이스 내 서비스를 참조하는 통합된 방법, 모든 애플리케이션을 모니터링하는 표준화된 지표와 로그를 제공합니다. Service Connect는 서비스만 상호 연결합니다.

다음 다이어그램에서는 VPC의 서브넷 2개와 서비스 2개가 있는 서비스 연결 네트워크 예제를 보여줍니다. 클라이언트 서비스는 각 서브넷에서 1개 작업으로 WordPress를 실행합니다. 서버 서비스는 각 서브넷에서 1개 작업으로 MySQL을 실행합니다. 각 서비스가 2개의 서브넷에 분산된 여러 작업을 실행하므로 두 서비스 모두 가용성이 높고 작업 및 가용 영역 문제에 대한 복원력이 뛰어납니다. 실선 화살표는 WordPress에서 MySQL로의 연결을 나타냅니다. 예를 들어, `mysql --host=mysql` CLI 명령은 작업의 WordPress 컨테이너 내에서 IP 주소 `172.31.16.1`로 실행됩니다. 이 명령은 MySQL의 기본 포트에서 약식 이름 `mysql`을 사용합니다. 이 이름과 포트는 동일한 작업의 서비스 연결 프록시에 연결합니다. WordPress 작업의 프록시는 라운드 로빈 로드 밸런싱과 이상값 탐지의 이전 실패 정보를 사용하여 연결할 MySQL 작업을 선택합니다. 다이어그램에서 실선 화살표로 표시된 것처럼 프록시는 IP 주소 `172.31.16.2`를 사용하여 MySQL 작업의 두 번째 프록시에 연결합니다. 두 번째 프록시는 동일한 작업의 로컬 MySQL 서버에 연결합니다. 두 프록시 모두 Amazon ECS 및 Amazon CloudWatch 콘솔에서 그래프로 확인할 수 있는 연결 성능을 보고하므로 동일한 방법으로 모든 종류의 애플리케이션에서 성능 지표를 얻을 수 있습니다.



다음 용어는 Service Connect에서 사용됩니다.

포트 이름

특정 포트 매핑에 이름을 할당하는 Amazon ECS 작업 정의 구성입니다. 이 구성은 Amazon ECS Service Connect에서만 사용됩니다.

클라이언트 별칭

엔드포인트에서 사용되는 포트 번호를 할당하는 Amazon ECS 서비스 구성입니다. 추가로 클라이언트 별칭은 엔드포인트의 DNS 이름을 할당하여 검색 이름을 재정의할 수 있습니다. 검색 이름이 Amazon ECS 서비스에 제공되지 않은 경우 클라이언트 별칭 이름이 엔드포인트 이름인 포트 이름보다 우선합니다. 엔드포인트 예제는 엔드포인트 정의를 참조하세요. 여러 클라이언트 별칭을 Amazon ECS 서비스에 할당할 수 있습니다. 이 구성은 Amazon ECS Service Connect에서만 사용됩니다.

검색 이름

작업 정의에서 지정된 포트에 대해 생성할 수 있는 선택적 중간 이름입니다. 이 이름은 AWS Cloud Map 서비스를 생성하는 데 사용됩니다. 이 이름이 제공되지 않으면 작업 정의의 포트 이름이 사용됩니다. 특정 포트 및 Amazon ECS 서비스에 여러 검색 이름을 할당할 수 있습니다. 이 구성은 Amazon ECS Service Connect에서만 사용됩니다.

AWS Cloud Map 서비스 이름은 네임스페이스 내에서 고유해야 합니다. 이러한 제한 때문에 각 네임스페이스의 특정 작업 정의에 대한 검색 이름이 없는 Service Connect 구성은 하나만 있을 수 있습니다.

엔드포인트

API 또는 웹 사이트에 연결하기 위한 URL입니다. URL에는 프로토콜, DNS 이름 및 포트가 포함됩니다. 일반적인 엔드포인트에 대한 자세한 내용은 Amazon Web Services 일반 참조의 AWS 용어집에 있는 엔드포인트를 참조하세요.

Service Connect는 Amazon ECS 서비스에 연결하는 엔드포인트를 생성하고 Amazon ECS 서비스에서 엔드포인트에 연결할 작업을 구성합니다. URL에는 프로토콜, DNS 이름 및 포트가 포함됩니다. 포트는 컨테이너 이미지 내에 있는 애플리케이션과 일치해야 하므로 작업 정의에서 프로토콜 및 포트 이름을 선택합니다. 서비스에서 이름에 따라 각 포트를 선택하고 DNS 이름을 할당할 수 있습니다. Amazon ECS 서비스 구성에서 DNS 이름을 지정하지 않으면 기본적으로 작업 정의의 포트 이름이 사용됩니다. 예를 들어, Service Connect 엔드포인트는 `http://blog:80`, `grpc://checkout:8080` 또는 `http://_db.production.internal:99`일 수 있습니다.

Service Connect 서비스

Amazon ECS 서비스의 단일 엔드포인트 구성입니다. 이는 Service Connect 구성의 일부로 콘솔의 Service Connect and discovery name configuration(Service Connect 및 검색 이름 구성)의 단일 행 또는 Amazon ECS 서비스의 JSON 구성에 있는 `services` 목록의 한 객체로 구성됩니다. 이 구성은 Amazon ECS Service Connect에서만 사용됩니다.

자세한 내용은 Amazon Elastic Container Service API 참조의 ServiceConnectService를 참조하세요.

네임스페이스

Service Connect와 함께 사용할 AWS Cloud Map 네임스페이스의 약식 이름 또는 전체 Amazon 리소스 이름(ARN)입니다. 네임스페이스는 Amazon ECS 서비스 및 클러스터와 같은 AWS 리전에 있어야 합니다. AWS Cloud Map의 네임스페이스 유형은 Service Connect에 영향을 주지 않습니다.

Service Connect는 AWS Cloud Map 네임스페이스를 서로 통신하는 Amazon ECS 작업의 논리적 그룹으로 사용합니다. 각 Amazon ECS 서비스는 하나의 네임스페이스에만 속할 수 있습니다. 네임

스페이스 내의 서비스는 동일한 AWS 계정의 동일한 AWS 리전 내에 있는 여러 Amazon ECS 클러스터에 분산될 수 있습니다. 어떤 기준으로든 서비스를 자유롭게 구성할 수 있습니다.

클라이언트 서비스

네트워크 클라이언트 애플리케이션을 실행하는 서비스입니다. 이 서비스에는 네임스페이스가 구성되어 있어야 합니다. 서비스의 각 작업은 Service Connect 프록시 컨테이너를 통해 네임스페이스의 모든 엔드포인트를 검색하고 연결할 수 있습니다.

작업의 컨테이너에서 네임스페이스의 서비스에서 엔드포인트에 연결해야 하는 경우 클라이언트 서비스를 선택합니다. 프런트엔드, 리버스 프록시 또는 로드 밸런서 애플리케이션이 Elastic Load Balancing 등의 다른 방법을 통해 외부 트래픽을 수신하는 경우 이 유형의 Service Connect 구성 유형을 사용할 수 있습니다.

클라이언트-서버 서비스

네트워크 또는 웹 서비스 애플리케이션을 실행하는 Amazon ECS 서비스입니다. 이 서비스에는 네임스페이스와 하나 이상의 엔드포인트가 구성되어 있어야 합니다. 서비스의 각 작업은 엔드포인트를 사용하여 연결할 수 있습니다. Service Connect 프록시 컨테이너는 엔드포인트 이름과 포트에서 수신 대기하여 트래픽을 작업의 앱 컨테이너로 보냅니다.

네트워크 트래픽을 위해 포트를 노출하고 해당 포트에서 수신 대기하는 컨테이너가 있는 경우 클라이언트-서버 서비스를 선택합니다. 이러한 애플리케이션은 동일한 네임스페이스의 다른 클라이언트-서버 서비스에 연결할 필요는 없지만 클라이언트 구성이 필요합니다. 백엔드, 미들웨어, 비즈니스 티어 또는 대부분의 마이크로서비스는 이 유형의 Service Connect 구성을 사용할 수 있습니다. 프런트엔드, 리버스 프록시 또는 로드 밸런서 애플리케이션이 동일한 네임스페이스의 Service Connect로 구성된 다른 서비스로부터 트래픽을 수신하도록 하려면 이러한 서비스가 이 유형의 Service Connect 구성을 사용해야 합니다.

Service Connect 기능은 관련 서비스의 가상 네트워크를 생성합니다. 동일한 서비스 구성을 여러 네임스페이스에서 사용하여 독립적이면서도 동일한 애플리케이션 세트를 실행할 수 있습니다. Service Connect는 Amazon ECS 서비스의 프록시 컨테이너를 정의합니다. 이렇게 하면 동일한 작업 정의를 사용하여 Service Connect 구성이 다른 여러 네임스페이스에서 동일한 애플리케이션을 실행할 수 있습니다. 서비스가 만드는 각 태스크는 태스크의 프록시 컨테이너를 실행합니다.

Service Connect는 동일한 네임스페이스 내의 Amazon ECS 서비스 간 연결에 적합합니다. 다음 애플리케이션의 경우 추가 상호 연결 방법을 사용하여 Service Connect로 구성된 Amazon ECS 서비스에 연결해야 합니다.

- 다른 네임스페이스에서 구성된 태스크

- Service Connect에 대해 구성되지 않은 태스크
- Amazon ECS 외부의 기타 애플리케이션

이러한 애플리케이션은 Service Connect 프록시를 통해 연결할 수 있지만 Service Connect 엔드포인트 이름을 확인할 수는 없습니다.

이러한 애플리케이션이 Amazon ECS 작업의 IP 주소를 확인하려면 다른 상호 연결 방법을 사용해야 합니다.

주제

- 요금
- Amazon ECS Service Connect 구성 요소
- Amazon ECS Service Connect 구성 개요
- Amazon ECS Service Connect 트래픽 암호화
- AWS CLI를 사용한 Amazon ECS Service Connect 구성

요금

- Amazon ECS Service Connect 요금은 AWS Fargate 또는 컨테이너식 워크로드를 호스팅하는 Amazon EC2 인프라의 사용 여부에 따라 달라집니다. AWS Outposts에서 Amazon ECS를 사용하는 경우 요금은 Amazon EC2를 직접 사용할 때와 동일한 모델을 따릅니다. 자세한 내용은 Amazon ECS 요금을 참조하세요.
- Amazon ECS Service Connect 사용에 따른 추가 요금은 없습니다.
- 서비스 연결이 사용하는 경우 AWS Cloud Map 사용은 완전 무료입니다.
- 고객은 vCPU 및 메모리를 비롯하여 Amazon ECS Service Connect에서 사용하는 컴퓨팅 리소스에 대한 비용을 지불합니다. Amazon ECS Service Connect 에이전트는 고객 태스크 내에서 실행되므로 실행에 따른 추가 요금은 없습니다. 태스크 리소스는 고객 워크로드와 Amazon ECS Service Connect 에이전트 간에 공유됩니다.
- AWS Private CA에서 Amazon ECS Service Connect 트래픽 암호화 기능을 사용하는 경우, 고객은 생성한 프라이빗 인증 기관과 발급된 각 TLS 인증서에 대한 비용을 지불합니다. 자세한 내용은 AWS Private Certificate Authority 요금을 참조하세요. TLS 인증서의 월별 비용을 추정하려면 고객은 TLS가 활성화된 Amazon ECS 서비스의 수를 파악하여 인증서 비용을 곱한 다음 6을 곱해야 합니다. Amazon ECS Service Connect는 5일마다 TLS 인증서를 자동으로 교체하므로, Amazon ECS 서비스당 월 평균 6개의 인증서가 발급됩니다.

Amazon ECS Service Connect 구성 요소

Amazon ECS Service Connect를 사용하는 경우에는 네트워크 요청을 받는 서버 애플리케이션(클라이언트-서버 서비스)을 실행하거나 요청을 생성하는 클라이언트 애플리케이션(클라이언트 서비스)을 실행하도록 각 Amazon ECS 서비스를 구성합니다.

Service Connect 사용을 시작할 준비가 되면 클라이언트-서버 서비스로 시작합니다. 새 서비스 또는 기존 서비스에 Service Connect 구성을 추가할 수 있습니다. Amazon ECS는 네임스페이스에 Service Connect 엔드포인트를 생성합니다. 추가로 Amazon ECS는 서비스에 새 배포를 생성하여 현재 실행 중인 작업을 대체합니다.

기존 작업 및 기타 애플리케이션은 기존 엔드포인트 및 외부 애플리케이션에 계속 연결할 수 있습니다. 클라이언트-서버 서비스가 스케일 아웃을 통해 태스크를 추가하면 클라이언트의 새 연결이 모든 태스크 간에 즉시 밸런싱됩니다. 클라이언트-서버 서비스가 업데이트된 경우 클라이언트의 새 연결은 새 버전의 태스크 사이에서 밸런싱됩니다.

기존 작업은 확인할 수 없으며 새 엔드포인트에 연결할 수 없습니다. 동일한 네임스페이스에 Service Connect 구성이 있고 이 배포 이후에 실행을 시작하는 새 태스크만 이 엔드포인트를 확인하고 여기에 연결할 수 있습니다.

즉, 서버 애플리케이션 운영자가 언제든지 구성을 변경할 수 있더라도 클라이언트 애플리케이션 운영자가 앱 구성이 변경되는 시기를 결정합니다. 네임스페이스의 엔드포인트 목록은 네임스페이스의 서비스가 배포될 때마다 변경될 수 있습니다. 기존 태스크와 대체 태스크는 최근 배포 후에도 동일하게 작동합니다.

다음 예제를 살펴보세요.

먼저 퍼블릭 인터넷에서 사용할 수 있는 애플리케이션을 단일 AWS CloudFormation 템플릿 및 단일 AWS CloudFormation 스택으로 생성하고 있다고 가정해 보겠습니다. 공개 검색 및 연결 가능성은 프런트엔드 클라이언트 서비스를 포함하여 AWS CloudFormation에서 마지막으로 생성해야 합니다. 프런트엔드 클라이언트 서비스가 실행되고 퍼블릭에서 사용 가능하지만 백엔드는 사용할 수 없는 기간을 방지하려면 이 순서로 서비스를 생성해야 합니다. 이렇게 하면 해당 기간 동안 오류 메시지가 퍼블릭으로 전송되는 것을 방지할 수 있습니다. AWS CloudFormation에서는 `dependsOn`을 사용하여 AWS CloudFormation에 여러 Amazon ECS 서비스를 병렬로 또는 동시에 만들 수 없음을 나타내야 합니다. 클라이언트 작업이 연결되는 각 백엔드 클라이언트-서버 서비스의 프런트엔드 클라이언트 서비스에 `dependsOn`을 추가해야 합니다.

두 번째로 프런트엔드 서비스가 Service Connect 구성 없이 존재한다고 가정합니다. 작업은 기존 백엔드 서비스에 연결되고 있습니다. 먼저 DNS 또는 프런트엔드가 사용하는 `clientAlias`에서의 동일한

이름을 사용하여 백엔드 서비스에 클라이언트-서버 Service Connect 구성을 추가합니다. 이렇게 하면 새 배포가 생성되므로 모든 배포 롤백 탐지 또는 AWS Management Console, AWS CLI, AWS SDK 및 기타 메서드가 백엔드 서비스를 롤백하고 이전 배포 및 구성으로 되돌릴 수 있습니다. 백엔드 서비스의 성능과 동작이 만족스러우면 프런트엔드 서비스에 클라이언트 또는 클라이언트-서버 Service Connect 구성을 추가합니다. 새 배포의 작업만 새 작업에 추가된 Service Connect 프록시를 사용합니다. 이 구성에 문제가 있는 경우 배포 롤백 탐지 또는 AWS Management Console, AWS CLI, AWS SDK 및 기타 메서드를 사용하여 백엔드 서비스를 롤백하고 이전 배포 및 구성으로 되돌려 이전 구성으로 롤백하고 되돌릴 수 있습니다. Service Connect 대신 DNS를 기반으로 하는 다른 서비스 검색 시스템을 사용하는 경우 로컬 DNS 캐시가 만료된 후 프런트엔드 또는 클라이언트 애플리케이션이 새 엔드포인트를 사용하고 엔드포인트 구성을 변경하기 시작하며, 일반적으로 몇 시간이 걸립니다.

네트워킹

기본적으로 Service Connect 프록시는 태스크 정의 포트 매핑의 `containerPort`에서 수신 대기합니다. 보안 그룹 규칙은 클라이언트가 실행될 서브넷에서 이 포트로 들어오는(수신) 트래픽을 허용해야 합니다.

Service Connect 서비스 구성에서 포트 번호를 설정하더라도 Service Connect 프록시가 수신 대기하는 클라이언트-서버 서비스용 포트는 변경되지 않습니다. 이 포트 번호를 설정하면 Amazon ECS는 해당 작업 내의 Amazon ECS 프록시에서 클라이언트 서비스가 연결하는 엔드포인트의 포트를 변경합니다. 클라이언트 서비스의 프록시는 `containerPort`를 사용하여 클라이언트-서버 서비스의 프록시에 연결합니다.

Service Connect 프록시가 수신 대기하는 포트를 변경하려면 클라이언트-서버 서비스의 Service Connect 구성에서 `ingressPortOverride`를 변경합니다. 이 포트 번호를 변경하는 경우 이 서비스에 대한 트래픽에 사용되는 이 포트의 인바운드 트래픽을 허용해야 합니다.

애플리케이션이 Service Connect용으로 구성된 Amazon ECS 서비스로 전송하는 트래픽의 경우 Amazon VPC 및 서브넷에 사용 중인 `containerPort` 및 `ingressPortOverride` 포트 번호를 허용하는 라우팅 테이블 규칙 및 네트워크 ACL 규칙이 있어야 합니다.

Service Connect를 사용하여 VPC 간에 트래픽을 보낼 수 있습니다. 라우팅 테이블 규칙, 네트워크 ACL, 보안 그룹에 대한 동일한 요구 사항이 두 VPC에 모두 적용됩니다.

예를 들어, 두 개의 클러스터는 서로 다른 VPC에서 작업을 생성합니다. 각 클러스터의 서비스는 동일한 네임스페이스를 사용하도록 구성됩니다. 이 두 서비스의 애플리케이션은 VPC DNS 구성 없이 네임스페이스의 모든 엔드포인트를 확인할 수 있습니다. 하지만 VPC 피어링, VPC 또는 서브넷 라우팅 테이블, VPC 네트워크 ACL이 `containerPort` 및 `ingressPortOverride` 포트 번호의 트래픽을 허용하는 경우를 제외하면 프록시를 연결할 수 없습니다.

bridge 네트워킹 모드를 사용하는 작업의 경우 상위 동적 포트 범위에서 트래픽을 허용하는 인바운드 규칙이 포함된 보안 그룹을 생성해야 합니다. 그런 다음, Service Connect 클러스터의 모든 EC2 인스턴스에 보안 그룹을 할당합니다.

Service Connect 프록시

Service Connect 구성을 사용하여 서비스를 생성하거나 업데이트하는 경우 Amazon ECS는 새 태스크가 시작될 때마다 새 컨테이너를 추가합니다. 별도의 컨테이너를 사용하는 이러한 패턴을 sidecar라고 합니다. 이 컨테이너는 작업 정의에 없으며 구성할 수 없습니다. Amazon ECS는 서비스에서 컨테이너 구성을 관리합니다. 이렇게 하면 Service Connect 없이도 여러 서비스, 네임스페이스, 태스크 간에 동일한 태스크 정의를 재사용할 수 있습니다.

프록시 리소스

- 태스크 정의의 경우 CPU 및 메모리 파라미터를 설정해야 합니다.

  Service Connect 프록시 컨테이너용 작업 CPU 및 메모리에 256개의 추가 CPU 유닛과 최소 64MiB의 메모리를 추가하는 것이 좋습니다. AWS Fargate에서 설정할 수 있는 최소 메모리양은 512MiB입니다. Amazon EC2에는 태스크 정의 메모리가 필요합니다.
- 서비스의 경우 Service Connect 구성에서 로그 구성을 설정합니다.
- 이 서비스의 작업이 피크 부하 시 초당 500개 이상의 요청을 받을 것으로 예상되는 경우 이 작업 정의에서 Service Connect 프록시 컨테이너에 대한 작업 CPU에 512개의 CPU 유닛을 추가하는 것이 좋습니다.
- 네임스페이스에서 100개 이상의 Service Connect 서비스를 생성하거나 네임스페이스 내의 모든 Amazon ECS 서비스에서 총 2000개 이상의 작업을 생성할 것으로 예상되는 경우 Service Connect 프록시 컨테이너의 작업 메모리에 128MiB의 메모리를 추가하는 것이 좋습니다. 네임스페이스의 모든 Amazon ECS 서비스에서 사용하는 모든 작업 정의에서 이 작업을 수행해야 합니다.

프록시 구성

애플리케이션은 수행 중인 동일한 작업에서 사이드카 컨테이너의 프록시에 연결합니다. Amazon ECS는 애플리케이션이 동일한 네임스페이스의 엔드포인트 이름에 연결되는 경우에만 애플리케이션이 프록시에 연결되도록 태스크와 컨테이너를 구성합니다. 다른 모든 트래픽은 프록시를 사용하지 않습니다. 다른 트래픽에는 동일한 VPC의 IP 주소, AWS 서비스 엔드포인트 및 외부 트래픽이 포함됩니다.

로드 밸런싱

서비스 연결은 서비스 연결 엔드포인트의 작업 간의 로드 밸런싱에 라운드 로빈 전략을 사용하도록 프록시를 구성합니다. 연결이 시작되는 작업에 있는 로컬 프록시는 엔드포인트를 제공하는 클라이언트-서버 서비스의 작업 중 하나를 선택합니다.

예를 들어 로컬이라는 네임스페이스에 클라이언트 서비스로 구성된 서비스에서 WordPress를 실행하는 태스크가 있다고 가정합니다. MySQL 데이터베이스를 실행하는 2개의 작업이 있는 다른 서비스도 있습니다. 이 서비스는 동일한 네임스페이스에서 서비스 연결을 통해 `mysql`이라는 엔드포인트를 제공하도록 구성되어 있습니다. WordPress작업에서 WordPress 애플리케이션은 엔드포인트 이름을 사용하여 데이터베이스에 연결합니다. 이 이름에 대한 연결은 동일한 태스크의 사이드카 컨테이너에서 실행되는 프록시로 이동합니다. 그러면 프록시는 라운드 로빈 전략을 사용하여 MySQL 작업 중 하나에 연결할 수 있습니다.

로드 밸런싱 전략: 라운드 로빈

이상치 탐지

이 기능은 프록시에 있는 이전에 실패한 연결에 대한 데이터를 사용하여 연결이 실패한 호스트에 새 연결이 전송되지 않도록 합니다. 서비스 연결은 프록시의 이상치 탐지 기능을 구성하여 수동 상태 확인을 제공합니다.

이전 예제를 사용하여 프록시는 MySQL 태스크 중 하나에 연결할 수 있습니다. 프록시가 특정 MySQL 작업에 여러 번 연결했는데 지난 30초간 5개 이상의 연결이 실패한 경우 프록시는 30~300초간 해당 MySQL 작업을 방지합니다.

재시도

서비스 연결은 프록시를 통과하여 실패한 연결을 다시 시도하도록 프록시를 구성하고, 두 번째 시도에서는 이전 연결의 호스트를 사용하지 않습니다. 이렇게 하면 서비스 연결을 통한 각 연결이 일회성 이유로 실패하지 않게 됩니다.

재시도 횟수: 2

제한 시간

서비스 연결은 클라이언트-서버 애플리케이션이 응답할 때까지 최대 시간을 대기하도록 프록시를 구성합니다. 기본 제한 시간 값은 15초이지만 업데이트할 수 있습니다.

선택적 파라미터:

idleTimeout - 유휴 상태에서 연결이 활성 상태로 유지되는 시간(초)입니다. 값이 `0`이면 `idleTimeout`이 비활성화됩니다.

HTTP/HTTP2/GRPC의 `idleTimeout` 기본값은 5분입니다.

TCP의 `idleTimeout` 기본값은 1시간입니다.

perRequestTimeout - 업스트림이 요청당 완전한 응답으로 응답할 때까지 기다리는 시간입니다. 값이 `0`이면 `perRequestTimeout`이 꺼집니다. 애플리케이션 컨테이너에 대한 `appProtocol`이 HTTP/HTTP2/GRPC인 경우에만 설정할 수 있습니다. 기본값은 15초입니다.

> **Note**
>
> `idleTimeout`이 `perRequestTimeout`보다 짧은 시간으로 설정되는 경우 `perRequestTimeout`가 아닌 `idleTimeout`에 도달하면 연결이 닫힙니다.

고려 사항

Service Connect를 사용할 때는 다음을 고려하세요.

- Service Connect를 사용하려면 Fargate에서 실행되는 태스크가 Fargate Linux 플랫폼 버전 `1.4.0` 이상을 사용해야 합니다.
- 컨테이너 인스턴스의 Amazon ECS 에이전트 버전은 `1.67.2` 이상이어야 합니다.
- 서비스 연결을 사용하려면 컨테이너 인스턴스가 Amazon ECS 최적화 Amazon Linux 2023 AMI 버전 `20230428` 이상 또는 Amazon ECS 최적화 Amazon Linux 2 AMI 버전 `2.0.20221115`를 실행해야 합니다. 이러한 버전에는 Amazon ECS 컨테이너 에이전트 외에 서비스 연결 에이전트가 있습니다. 서비스 연결 에이전트에 대한 자세한 내용은 GitHub의 Amazon ECS Service Connect Agent를 참조하세요.
- 컨테이너 인스턴스에 리소스 `arn:aws:ecs:`*`region`*`:`*`0123456789012`*`:task-set/`*`cluster`*`/*`에 대한 `ecs:Poll` 권한이 있어야 합니다. `ecsInstanceRole`을 사용하는 경우에는 권한을 더 추가할 필요가 없습니다. `AmazonEC2ContainerServiceforEC2Role` 관리형 정책에 필요한 권한이 있습니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.
- `bridge` 네트워크 모드를 사용하고 Service Connect를 사용하는 작업은 `hostname` 컨테이너 정의 파라미터를 지원하지 않습니다.
- 작업 정의는 Service Connect를 사용하는 작업 메모리 제한을 설정해야 합니다. 자세한 내용은 Service Connect 프록시 섹션을 참조하세요.
- 컨테이너 메모리 제한을 설정하는 태스크 정의는 지원되지 않습니다.

  컨테이너에 컨테이너 메모리 제한을 설정할 수 있지만 작업 메모리 제한을 컨테이너 메모리 제한의 합보다 큰 값으로 설정해야 합니다. 컨테이너 제한에 할당되지 않은 작업 제한의 추가 CPU 및 메모

리는 Service Connect 프록시 컨테이너 및 컨테이너 제한을 설정하지 않은 다른 컨테이너에서 사용됩니다. 자세한 내용은 Service Connect 프록시 섹션을 참조하세요.

- 동일한 AWS 계정의 동일한 리전에서 모든 AWS Cloud Map 네임스페이스를 사용하도록 Service Connect를 구성할 수 있습니다.
- 각 서비스는 하나의 네임스페이스에만 속할 수 있습니다.
- 서비스에서 생성하는 태스크만 지원됩니다.
- 모든 엔드포인트는 네임스페이스 내에서 고유해야 합니다.
- 모든 검색 이름은 네임스페이스 내에서 고유해야 합니다.
- 애플리케이션이 새 엔드포인트를 확인하기 전에 기존 서비스를 다시 배포해야 합니다. 최신 배포 후 네임스페이스에 추가된 새 엔드포인트는 작업 구성에 추가되지 않습니다. 자세한 내용은 the section called “Service Connect 구성 요소” 섹션을 참조하세요.
- Service Connect는 클러스터가 삭제될 때 네임스페이스를 삭제하지 않습니다. AWS Cloud Map에서 네임스페이스를 삭제해야 합니다.
- Application Load Balancer 트래픽은 기본적으로 `awsvpc` 네트워크 모드에서 Service Connect 에이전트를 통해 라우팅됩니다. 서비스 이외의 트래픽이 Service Connect 에이전트를 우회하도록 하려면 Service Connect 서비스 구성에서 `ingressPortOverride` 파라미터를 사용합니다.
- TLS를 사용하는 Service Connect는 브리지 네트워킹 모드를 지원하지 않습니다. `awsvpc` 네트워킹 모드만 지원됩니다.

Service Connect는 다음을 지원하지 않습니다.

- Windows 컨테이너
- HTTP 1.0
- 독립 실행형 작업
- CodeDeploy 및 외부 배포 유형에서 지원하는 블루/그린을 사용하는 서비스
- Amazon ECS Anywhere용 `External` 컨테이너 인스턴스는 Service Connect에서 지원되지 않습니다.
- PPv2

Amazon ECS Service Connect 구성 개요

Service Connect를 사용하는 경우 리소스에서 구성해야 하는 파라미터가 있습니다.

다음 표에는 Amazon ECS 리소스의 구성 파라미터가 설명되어 있습니다.

파라미터 위치	앱 유형	설명	필수
태스크 정의	클라이언트	클라이언트 작업 정의에서 Service Connect에 사용할 수 있는 변경 사항이 없습니다.	N/A
태스크 정의	클라이언트-서버	서버는 컨테이너의 `portMappings` 에 있는 포트에 `name` 필드를 추가해야 합니다. 자세한 내용은 `portMappings` 섹션을 참조하세요.	예
태스크 정의	클라이언트-서버	서버는 선택적으로 애플리케이션 프로토콜(예: HTTP)을 제공하여 서버 애플리케이션에 대한 프로토콜별 지표를 수신할 수 있습니다(예: `HTTP 5xx`).	아니요
서비스 정의	클라이언트	조인할 네임스페이스를 구성하려면 클라이언트 서비스에서 `serviceConnectConfiguration` 을 추가해야 합니다. 이 네임스페이스에는 이 서비스가 검색해야 하는 모든 서버 서비스가 포함되어야 합니다. 자세한 내용은 `serviceConnectConfiguration` 섹션을 참조하세요.	예
서비스 정의	클라이언트-서버	서버 서비스에서 `serviceConnectConfiguration` 을 추가하여 서비스에서 사용할 수 있는 DNS 이름, 포트 번호 및 네임스페이스를 구성해야 합니다. 자세한 내용은 `serviceConnectConfiguration` 섹션을 참조하세요.	예
클러스터	클라이언트	클러스터는 기본 Service Connect 네임스페이스를 추가할 수 있습니다. 클러스터의 새 서비스는 Service Connect가 서	아니요

파라미터 위치	앱 유형	설명	필수
		비스에서 구성된 경우 네임스페이스를 상속합니다.	
클러스터	클라이언트-서버	서버 서비스에 적용되는 클러스터의 Service Connect에 사용할 수 있는 변경 사항이 없습니다. 서버 작업 정의 및 서비스는 해당 구성을 설정해야 합니다.	N/A

Service Connect 구성 단계 개요

다음 단계는 Service Connect 구성 방법의 개요를 제공합니다.

Important

- Service Connect에서 사용자 계정에 AWS Cloud Map 서비스를 생성합니다. 인스턴스를 수동으로 등록/등록 취소하거나, 인스턴스 속성을 변경하거나, 서비스를 삭제하여 이러한 AWS Cloud Map 리소스를 수정하면 애플리케이션 트래픽 또는 후속 배포에서 예상치 못한 동작이 발생할 수 있습니다.
- Service Connect는 태스크 정의에서 링크를 지원하지 않습니다.

1. 작업 정의의 포트 매핑에 포트 이름을 추가합니다. 또한 애플리케이션의 계층 7 프로토콜을 식별하여 추가 지표를 얻을 수 있습니다.
2. AWS Cloud Map 네임스페이스를 사용하여 ECS 클러스터를 생성하거나 네임스페이스를 별도로 생성합니다. 간단한 구성을 위해 네임스페이스에 원하는 이름을 사용하여 클러스터를 생성하고 네임스페이스에 동일한 이름을 지정합니다. 이 경우 Amazon ECS는 필요한 구성으로 새 HTTP 네임스페이스를 생성합니다. Service Connect는 Amazon Route 53에서 DNS 호스팅 영역을 사용하거나 생성하지 않습니다.
3. 네임스페이스 내에 Service Connect 엔드포인트를 생성하도록 서비스를 구성합니다.
4. 서비스를 배포하여 엔드포인트를 생성합니다. Amazon ECS는 각 작업에 Service Connect 프록시 컨테이너를 추가하고 AWS Cloud Map에 Service Connect 엔드포인트를 생성합니다. 이 컨테이너는 작업 정의에 구성되어 있지 않고, 작업 정의를 수정하지 않고 재사용하여 동일한 네임스페이스 또는 여러 네임스페이스에 여러 서비스를 생성할 수 있습니다.

5. 엔드포인트에 연결할 클라이언트 앱을 서비스로 배포합니다. Amazon ECS는 각 작업에서 Service Connect 프록시를 통해 이를 Service Connect 엔드포인트에 연결합니다.

   애플리케이션은 프록시를 서비스 연결 엔드포인트에 연결하는 데에만 사용합니다. 프록시를 사용하기 위한 추가 구성은 없습니다. 프록시는 라운드 로빈 로드 밸런싱, 이상값 탐지 및 재시도를 수행합니다. 프록시에 대한 자세한 내용은 Service Connect 프록시를 참조하세요.
6. Amazon CloudWatch의 Service Connect 프록시를 통해 트래픽을 모니터링합니다.

클러스터 구성

클러스터를 생성하거나 업데이트할 때 Service Connect의 기본 네임스페이스를 설정할 수 있습니다. 동일한 AWS 리전 및 계정에 없는 네임스페이스 이름을 지정하면 새 HTTP 네임스페이스가 생성됩니다.

클러스터를 생성하고 기본 Service Connect 네임스페이스를 지정하면 클러스터는 Amazon ECS가 네임스페이스를 생성하는 동안 `PROVISIONING` 상태로 대기합니다. 네임스페이스의 상태를 나타내는 클러스터 상태에서 `attachment`를 볼 수 있습니다. 연결은 기본적으로 AWS CLI에 표시되지 않으므로 연결을 보려면 `--include ATTACHMENTS`를 추가해야 합니다.

서비스 구성

Service Connect는 최소 구성을 요구하도록 설계됩니다. 작업 정의의 Service Connect에서 사용할 각 포트 매핑의 이름을 설정해야 합니다. 서비스에서 클라이언트 서비스를 만들려면 Service Connect를 켜고 네임스페이스를 선택해야 합니다. 클라이언트-서버 서비스를 만들려면 포트 매핑 중 하나의 이름과 일치하는 단일 Service Connect 서비스 구성을 추가해야 합니다. Amazon ECS는 작업 정의의 포트 번호와 포트 이름을 재사용하여 Service Connect 서비스와 엔드포인트를 정의합니다. 이러한 값을 재정의하려면 콘솔에서 다른 파라미터인 Discovery, DNS 및 Port를 사용하거나 Amazon ECS API에서 `discoveryName`과 `clientAliases`를 각각 사용할 수 있습니다.

Amazon ECS Service Connect 트래픽 암호화

Amazon ECS Service Connect는 Amazon ECS 서비스에 대해 전송 계층 보안(TLS) 인증서를 사용한 자동 트래픽 암호화를 지원합니다. Amazon ECS 서비스에서 AWS Private Certificate Authority(AWS Private CA)를 가리키면 Amazon ECS는 자동으로 TLS 인증서를 프로비저닝하여 Amazon ECS Service Connect 서비스 사이에서 트래픽을 암호화합니다. Amazon ECS는 트래픽 암호화에 사용되는 TLS 인증서를 생성, 교체 및 분산합니다.

Service Connect를 사용하는 자동 트래픽 암호화는 업계 최고의 암호화 기능을 사용하여 보안 요구 사항을 충족할 수 있도록 서비스 간 통신을 보호합니다. `256-bit ECDSA` 및 `2048-bit RSA` 암호화를

사용하는 AWS Private Certificate Authority TLS 인증서를 지원합니다. 또한 규정 준수 요구 사항을 충족할 수 있도록 프라이빗 인증서 및 서명 키를 완벽하게 제어합니다. 기본적으로 TLS 1.3은 지원되지만 TLS 1.0~1.2는 지원되지 않습니다. 서비스 연결은 다음 암호와 함께 TLS 1.3을 지원합니다.

- `TLS_AES_128_GCM_SHA256`
- `TLS_AES_256_GCM_SHA384`
- `TLS_CHACHA20_POLY1305_SHA256`

> **Note**
>
> TLS 1.3을 사용하려면 이를 대상의 리스너에서도 활성화해야 합니다.
> Amazon ECS 에이전트를 통과하는 인바운드 및 아웃바운드 트래픽만 암호화됩니다.

Service Connect 및 Application Load Balancer 상태 확인

Application Load Balancer 상태 확인 및 TLS 1.3 암호화와 함께 Service Connect를 사용할 수 있습니다.

Application Load Balancer 구성

다음 설정을 사용하여 Application Load Balancer를 구성합니다.

- TLS 1.3 보안 정책(예: `ELBSecurityPolicy-TLS13-1-2-2021-06`)을 사용하여 TLS 리스너를 구성합니다. 자세한 내용은 Security policies for your Application Load Balancer를 참조하세요.
- 다음 설정을 사용하여 대상 그룹을 생성합니다.
  - 프로토콜을 HTTPS로 설정
  - 대상 그룹을 TLS 리스너에 연결
  - Service Connect 서비스의 컨테이너 포트와 일치하도록 상태 확인 포트 구성

Service Connect 구성

다음 설정을 사용하여 서비스를 구성합니다.

- `bridge` 네트워크 모드는 지원되지 않으므로 `awsvpc` 네트워크 모드를 사용하도록 서비스를 구성합니다.
- 서비스에 Service Connect를 활성화합니다.

- 다음 설정을 사용하여 로드 밸런서 구성을 설정합니다.
  - Application Load Balancer에 구성한 대상 그룹 지정
  - Service Connect TLS 서비스의 컨테이너 포트와 일치하도록 컨테이너 포트 설정
- 서비스에 `ingressPortOverride`를 설정하지 마세요. 자세한 내용은 Amazon Elastic Container Service API 참조의 ServiceConnectService를 참조하세요.

고려 사항

Application Load Balancer, TLS 및 Service Connect를 사용할 때는 다음 사항을 고려하세요.

- Service Connect를 TLS 암호화와 함께 사용하는 경우 HTTPS 상태 확인에 `bridge` 네트워크 모드 대신 `awsvpc` 네트워크 모드를 사용합니다. HTTP 상태 확인은 `bridge` 모드에서 계속 작동합니다.
- 대상 그룹 상태 확인 포트를 기본 HTTPS 포트(443)가 아닌 Service Connect 서비스의 컨테이너 포트와 일치하도록 구성합니다.

AWS Private Certificate Authority 인증서 및 Service Connect

인프라 IAM 역할이 있어야 합니다. 이 역할에 대한 자세한 내용은 Amazon ECS 인프라 IAM 역할을 참조하세요.

Service Connect에 대한 AWS Private Certificate Authority 모드

AWS Private Certificate Authority는 범용 모드와 단기 모드와 같은 두 가지 모드로 실행할 수 있습니다.

- 범용 - 만료 날짜를 포함하여 구성할 수 있는 인증서.
- 단기 - 최대 유효 기간이 7일인 인증서.

Amazon ECS는 두 모드를 모두 지원하지만 단기 인증서를 사용하는 것이 좋습니다. 기본적으로 인증서는 5일마다 교체되며, 단기 모드로 실행할 경우 범용 모드에 비해 비용을 크게 절감할 수 있습니다.

Service Connect는 인증서 해지를 지원하지 않으며, 대신 인증서 교체가 잦은 단기 인증서를 활용합니다. Secrets Manager에서 관리형 교체를 사용하여 교체 빈도를 수정하거나 보안 암호를 비활성화 또는 삭제할 수 있는 권한을 보유하지만, 이 경우 다음과 같은 결과가 발생할 수 있습니다.

- 더 짧은 교체 빈도 - 교체 빈도가 짧을수록 AWS Private CA, AWS KMS, Secrets Manager와 Auto Scaling에서 교체 워크로드가 증가하기 때문에 비용이 증가합니다.
- 더 긴 교체 빈도 - 교체 빈도가 7일을 초과하면 애플리케이션 통신이 실패합니다.

- 보안 암호 삭제 - 보안 암호를 삭제하면 교체에 실패하고 고객 애플리케이션 통신에 영향을 미칩니다.

보안 암호 교체에 실패하는 경우 `RotationFailed` 이벤트가 AWS CloudTrail에 게시됩니다. `RotationFailed`에 대한 CloudWatch 경보를 설정할 수도 있습니다.

> **Important**
>
> 보안 암호에 복제 리전을 추가하지 마세요. 이렇게 하면 Amazon ECS는 복제에서 리전을 제거할 수 있는 권한이 없으므로 Amazon ECS가 보안 암호를 삭제할 수 없습니다. 이미 복제를 추가한 경우 다음 명령을 실행합니다.
>
> ```
> aws secretsmanager remove-regions-from-replication \
>  --secret-id SecretId \
>  --remove-replica-regions region-name
> ```

하위 인증 기관

AWS Private CA, 루트 또는 하위 항목을 Service Connect TLS로 가져와 서비스에 대한 최종 엔터티 인증서를 발급할 수 있습니다. 제공된 발행자는 어디서나 서명자 및 신뢰 루트로 처리됩니다. 서로 다른 하위 CA에서 애플리케이션의 여러 부분에 대한 최종 엔터티 인증서를 발급할 수 있습니다. AWS CLI를 사용하는 경우 CA의 Amazon 리소스 이름(ARN)을 제공하여 신뢰 체인을 설정합니다.

온프레미스 인증 기관

온프레미스 CA를 사용하려면 AWS Private Certificate Authority에서 하위 CA를 생성하고 구성합니다. 이렇게 하면 Amazon ECS 워크로드용으로 발급된 모든 TLS 인증서가 온프레미스에서 실행하는 워크로드와 신뢰 체인을 공유하고 안전하게 연결할 수 있습니다.

> **Important**
>
> AWS Private CA에서 필수 태그 `AmazonECSManaged : true`를 추가합니다.

코드형 인프라

코드형 인프라(IaC) 도구와 함께 Service Connect TLS를 사용할 때 서비스가 드레이닝 상태로 멈추는 등의 문제가 발생하지 않도록 종속성을 올바르게 구성하는 것이 중요합니다. Amazon ECS 서비스 이후에 AWS KMS 키(제공된 경우), IAM 역할 및 AWS Private CA 종속성을 삭제해야 합니다.

Service Connect 및 Secrets Manager

Amazon ECS Service Connect를 TLS 암호화와 함께 사용하는 경우 서비스는 다음과 같은 방식으로 Secrets Manager와 상호 작용합니다.

Service Connect에서는 제공된 인프라 역할을 활용하여 Secrets Manager 내에 보안 암호를 생성합니다. 이러한 보안 암호는 Service Connect 서비스 간 트래픽 암호화를 위해 TLS 인증서에 연결된 프라이빗 키를 저장하는 데 사용됩니다.

> **Warning**
>
> Service Connect를 통한 이러한 보안 암호의 자동 생성 및 관리는 서비스에 TLS 암호화를 구현하는 프로세스를 간소화합니다. 그러나 보안에 미치는 잠재적인 영향을 인식하는 것이 중요합니다. Secrets Manager에 대한 읽기 액세스 권한이 있는 다른 IAM 역할에서는 자동으로 생성된 이러한 보안 암호에 액세스할 수 있습니다. 그러면 액세스 제어가 제대로 구성되지 않은 경우 민감한 암호화 자료가 권한이 없는 당사자에게 노출될 수 있습니다.
> 이 위험을 완화하려면 다음과 같은 모범 사례를 따릅니다.
>
> - 특히 Service Connect에서 생성된 보안 암호의 경우 Secrets Manager에 대한 액세스 권한을 신중하게 관리하고 제한합니다.
> - 최소 권한의 원칙이 유지되도록 IAM 역할 및 권한을 정기적으로 감사합니다.

Secrets Manager에 대한 읽기 액세스 권한을 부여할 때는 Service Connect에서 생성된 TLS 프라이빗 키를 제외하는 것이 좋습니다. IAM 정책의 조건을 사용하여 패턴과 일치하는 ARN이 있는 보안 암호를 제외하여 이렇게 할 수 있습니다.

```
"arn:aws:secretsmanager:::secret:ecs-sc!"
```

`ecs-sc!` 접두사가 있는 모든 보안 암호에 대한 `GetSecretValue` 작업을 거부하는 IAM 정책의 예:

JSON

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:*:*:secret:ecs-sc!*"
        }
    ]
}
```

> **Note**
>
> 이는 일반적인 예이며 구체적인 사용 사례 및 AWS 계정 구성에 따라 조정하는 것이 좋습니다. 항상 IAM 정책을 철저히 테스트하여 보안 유지 관리되면서 의도한 액세스 권한이 제공되는지 확인합니다.

Service Connect가 Secrets Manager와 상호 작용하는 방식을 이해하면 Amazon ECS 서비스의 보안을 더 잘 관리하는 동시에 자동 TLS 암호화의 이점을 활용할 수 있습니다.

Service Connect 및 AWS Key Management Service

AWS Key Management Service를 사용하여 Service Connect 리소스를 암호화하고 해독할 수 있습니다. AWS KMS는 데이터를 보호하는 암호화 키를 만들고 관리할 수 있는 AWS 관리형 서비스입니다.

Service Connect에서 AWS KMS를 사용할 경우 사용자를 대신해 AWS에서 관리하는 AWS 소유 키를 사용하거나 기존 AWS KMS 키를 선택할 수 있습니다. 사용할 새 AWS KMS 키를 생성할 수도 있습니다.

자체 암호화 키 제공

자체 키 자료를 제공할 수도 있고, AWS Key Management Service를 통해 외부 키 저장소를 사용할 수 있습니다. 자체 키를 AWS KMS에 가져오고 Amazon ECS Service Connect에서 해당 키의 Amazon 리소스 이름(ARN)을 지정합니다.

다음은 예제 AWS KMS 정책입니다. 모든 ### ##을 고유한 값으로 바꿉니다.

JSON

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "id",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/role-name"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyPair"
      ],
      "Resource": "*"
    }
  ]
}
```

키 정책을 생성하는 방법에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 Creating a key policy를 참조하세요.

> **Note**
>
> Service Connect는 대칭 암호화 AWS KMS 키만 지원합니다. 다른 유형의 AWS KMS 키를 사용하여 Service Connect 리소스를 암호화할 수 없습니다. AWS KMS 키가 대칭 암호화 키인지 확인하는 것과 관련된 도움말은 비대칭 KMS 키 식별을 참조하세요.

AWS Key Management Service 대칭 암호화 키에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 Symmetric encryption AWS KMS keys를 참조하세요.

Amazon ECS Service Connect에 대한 TLS 활성화

Service Connect 서비스를 생성하거나 업데이트할 때 트래픽 암호화를 활성화합니다.

AWS Management Console을 사용하여 기존 네임스페이스에서 서비스의 트래픽 암호화를 활성화하는 방법

1. 인프라 IAM 역할이 있어야 합니다. 이 역할에 대한 자세한 내용은 Amazon ECS 인프라 IAM 역할을 참조하세요.

2. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
3. 탐색 창에서 네임스페이스를 선택합니다.
4. 트래픽 암호화를 활성화하려는 서비스가 포함된 네임스페이스를 선택합니다.
5. 트래픽 암호화를 활성화하려는 서비스를 선택합니다.
6. 오른쪽 상단에서 서비스 업데이트를 선택하고 Service Connect 섹션까지 아래로 스크롤합니다.
7. 서비스 정보에서 트래픽 암호화 켜기를 선택하여 TLS를 활성화합니다.
8. Service Connect TLS 역할에서 기존 인프라 IAM 역할을 선택하거나 새로 생성합니다.
9. 서명자 인증 기관의 경우 기존 인증 기관을 선택하거나 새로 생성합니다.

   자세한 내용은 AWS Private Certificate Authority 인증서 및 Service Connect을 참조하십시오.
10. AWS KMS key 선택에서 AWS 소유 및 관리형 키를 선택하거나 다른 키를 선택할 수 있습니다. 새로 생성하도록 선택할 수도 있습니다.

AWS CLI를 사용하여 서비스에 대한 TLS를 구성하는 예는 AWS CLI를 사용한 Amazon ECS Service Connect 구성 섹션을 참조하세요.

Amazon ECS Service Connect에 TLS가 활성화되어 있는지 확인

Service Connect는 Service Connect 에이전트에서 TLS를 시작하고 대상 에이전트에서 종료합니다. 따라서 애플리케이션 코드에서는 TLS 상호 작용을 확인할 수 없습니다. 다음 단계에 따라 TLS가 활성화되어 있는지 확인합니다.

1. 애플리케이션 이미지에 `openssl` CLI를 포함합니다.
2. 서비스에서 ECS Exec를 활성화하여 SSM을 통해 작업에 연결합니다. 또는 서비스와 동일한 Amazon VPC에서 Amazon EC2 인스턴스를 시작할 수 있습니다.
3. 확인하려는 서비스에서 작업의 IP와 포트를 검색합니다. AWS Cloud Map 콘솔에서 작업 IP 주소를 검색할 수 있습니다. 이 정보는 네임스페이스의 서비스 세부 정보 페이지에서 찾을 수 있습니다.
4. 다음 예제와 같이 `execute-command`를 사용하여 태스크 중 하나에 로그온합니다. 또는 2단계에서 생성한 Amazon EC2 인스턴스에 로그온합니다.

```
$ aws ecs execute-command --cluster cluster-name \
    --task task-id  \
    --container container-name \
    --interactive \
    --command "/bin/sh"
```

> **Note**
>
> DNS 이름을 직접 호출해도 인증서가 공개되지 않습니다.

5. 연결된 쉘에서 openssl CLI를 사용하여 작업에 연결된 인증서를 확인하고 조회합니다.

   예시:

```
openssl s_client -connect 10.0.147.43:6379 < /dev/null 2> /dev/null \
| openssl x509 -noout -text
```

   응답 예제:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            <serial-number>
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: <issuer>
        Validity
            Not Before: Jan 23 21:38:12 2024 GMT
            Not After : Jan 30 22:38:12 2024 GMT
        Subject: <subject>
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    <pub>
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:redis.yelb-cftc
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:<key-id>

            X509v3 Subject Key Identifier:
                1D:<id>
```

```
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
    Signature Algorithm: ecdsa-with-SHA256
        <hash>
```

AWS CLI를 사용한 Amazon ECS Service Connect 구성

AWS CLI를 통해 Service Connect를 사용하는 Fargate 태스크에 대한 Amazon ECS 서비스를 생성할 수 있습니다.

> **Note**
>
> 듀얼 스택 서비스 엔드포인트를 사용하면 AWS CLI, SDK 및 Amazon ECS API에서 IPv4 및 IPv6 모두를 통해 Amazon ECS와 상호 작용할 수 있습니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

사전 조건

다음은 Service Connect의 사전 조건입니다.

- 최신 버전의 AWS CLI가 설치되고 구성되어 있는지 확인합니다. 자세한 내용은 Installing or updating to the latest version of the AWS CLI를 참조하세요.
- IAM 사용자는 AmazonECS_FullAccess IAM 정책 예제에 지정된 필수 권한을 가집니다.
- 사용할 VPC, 서브넷, 라우팅 테이블 및 보안 그룹이 생성되었습니다. 자세한 내용은 the section called “Virtual Private Cloud 생성” 섹션을 참조하세요.
- 이름이 `ecsTaskExecutionRole`인 작업 실행 역할이 있고 `AmazonECSTaskExecutionRolePolicy` 관리형 정책이 역할에 연결되어 있습니다. 이 역할을 통해 Fargate는 NGINX 애플리케이션 로그와 Service Connect 프록시 로그를 Amazon CloudWatch Logs에 쓸 수 있습니다. 자세한 내용은 작업 실행 역할 생성 섹션을 참조하세요.

1단계: 클러스터 생성

다음 단계에 따라 Amazon ECS 클러스터 및 네임스페이스를 생성합니다.

Amazon ECS 클러스터와 AWS Cloud Map 네임스페이스 생성

1. 사용할 `tutorial`이라는 Amazon ECS 클러스터를 생성합니다. 파라미터 `--service-connect-defaults`는 클러스터의 기본 네임스페이스를 설정합니다. 예제 출력에서는 이름이 `service-connect`인 AWS Cloud Map 네임스페이스가 이 계정과 AWS 리전에 존재하지 않으므로 Amazon ECS에서 네임스페이스를 생성합니다. 네임스페이스는 계정의 AWS Cloud Map에서 생성되고, 다른 모든 네임스페이스에서 볼 수 있으므로 용도를 나타내는 이름을 사용합니다.

```
aws ecs create-cluster --cluster-name tutorial --service-connect-defaults
 namespace=service-connect
```

출력:

```
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
        "clusterName": "tutorial",
        "serviceConnectDefaults": {
            "namespace": "arn:aws:servicediscovery:us-
west-2:123456789012:namespace/ns-EXAMPLE"
        },
        "status": "PROVISIONING",
        "registeredContainerInstancesCount": 0,
        "runningTasksCount": 0,
        "pendingTasksCount": 0,
        "activeServicesCount": 0,
        "statistics": [],
        "tags": [],
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
        ],
        "capacityProviders": [],
        "defaultCapacityProviderStrategy": [],
        "attachments": [
            {
                "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
                "type": "sc",
                "status": "ATTACHING",
                "details": []
```

```
            }
        ],
        "attachmentsStatus": "UPDATE_IN_PROGRESS"
    }
}
}
```

2. 클러스터가 생성되었는지 확인합니다.

```
aws ecs describe-clusters --clusters tutorial
```

출력:

```
{
    "clusters": [
        {
            "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
            "clusterName": "tutorial",
            "serviceConnectDefaults": {
                "namespace": "arn:aws:servicediscovery:us-
west-2:123456789012:namespace/ns-EXAMPLE"
            },
            "status": "ACTIVE",
            "registeredContainerInstancesCount": 0,
            "runningTasksCount": 0,
            "pendingTasksCount": 0,
            "activeServicesCount": 0,
            "statistics": [],
            "tags": [],
            "settings": [],
            "capacityProviders": [],
            "defaultCapacityProviderStrategy": []
        }
    ],
    "failures": []
}
```

3. (선택 사항) AWS Cloud Map에서 네임스페이스가 생성되었는지 확인합니다. AWS Cloud Map에서 생성되었으므로 AWS Management Console 또는 일반 AWS CLI 구성을 사용할 수 있습니다.

   예를 들어 AWS CLI를 사용합니다.

```
aws servicediscovery get-namespace --id ns-EXAMPLE
```

출력:

```
{
    "Namespace": {
        "Id": "ns-EXAMPLE",
        "Arn": "arn:aws:servicediscovery:us-west-2:123456789012:namespace/ns-
EXAMPLE",
        "Name": "service-connect",
        "Type": "HTTP",
        "Properties": {
            "DnsProperties": {
                "SOA": {}
            },
            "HttpProperties": {
                "HttpName": "service-connect"
            }
        },
        "CreateDate": 1661749852.422,
        "CreatorRequestId": "service-connect"
    }
}
```

2단계: 서버용 서비스 생성

Service Connect 기능은 Amazon ECS에서 여러 애플리케이션을 상호 연결하는 데 사용됩니다. 이러한 애플리케이션 중 하나 이상은 연결할 웹 서비스를 제공해야 합니다. 이 단계에서는 다음을 생성합니다.

- 수정되지 않은 공식 NGINX 컨테이너 이미지를 사용하고 Service Connect 구성을 포함하는 작업 정의.
- 이 서비스에 대한 트래픽에 대해 서비스 검색 및 서비스 메시 프록시를 제공하도록 Service Connect를 구성하는 Amazon ECS 서비스 정의입니다. 구성은 클러스터 구성의 기본 네임스페이스를 재사용하여 각 서비스에 대한 서비스 구성의 양을 줄입니다.
- Amazon ECS 서비스입니다. 작업 정의를 사용하여 하나의 작업을 실행하고, Service Connect 프록시에 대한 추가 컨테이너를 삽입합니다. 프록시는 작업 정의의 컨테이너 포트 매핑에 지정된 포트에

서 수신을 대기합니다. Amazon ECS에서 실행되는 클라이언트 애플리케이션에서 클라이언트 작업의 프록시는 작업 정의 포트 이름, 서비스 검색 이름 또는 서비스 클라이언트 별칭 이름, 클라이언트 별칭의 포트 번호에 대한 아웃바운드 연결을 수신 대기합니다.

Service Connect를 사용하여 웹 서비스를 생성하려면

1. Fargate와 호환되며 awsvpc 네트워크 모드를 사용하는 태스크 정의를 등록합니다. 다음 단계를 따릅니다.

   a. 다음과 같은 태스크 정의의 내용으로 이름이 service-connect-nginx.json인 파일을 생성합니다.

      이 작업 정의는 포트 매핑에 name 및 appProtocol 파라미터를 추가하여 Service Connect를 구성합니다. 포트 이름을 사용하면 여러 포트가 사용될 때 서비스 구성에서 이 포트를 더욱 쉽게 식별할 수 있습니다. 포트 이름은 기본적으로 네임스페이스의 다른 애플리케이션에서 사용할 검색 가능한 이름으로도 사용됩니다.

      서비스에서 ECS Exec이 활성화되어 있으므로 작업 정의에 작업 IAM 역할이 포함됩니다.

      > **Important**
      >
      > 이 작업 정의는 logConfiguration을 사용하여 stdout 및 stderr의 nginx 출력을 Amazon CloudWatch Logs로 전송합니다. 이 작업 실행 역할에는 CloudWatch Logs 로그 그룹을 만드는 데 필요한 추가 권한이 없습니다. AWS Management Console 또는 AWS CLI를 사용하여 CloudWatch Logs에서 로그 그룹을 생성합니다. nginx 로그를 CloudWatch Logs로 전송하지 않으려면 logConfiguration을 제거합니다.
      > 태스크 실행 역할의 AWS 계정 ID를 사용자의 AWS 계정 ID로 바꿉니다.

```
{
    "family": "service-connect-nginx",
    "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
    "taskRoleArn": "arn:aws:iam::123456789012:role/ecsTaskRole",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
        "name": "webserver",
```

```
        "image": "public.ecr.aws/docker/library/nginx:latest",
        "cpu": 100,
        "portMappings": [
            {
                "name": "nginx",
                "containerPort": 80,
                "protocol": "tcp",
                "appProtocol": "http"
            }
        ],
        "essential": true,
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-nginx",
                "awslogs-region": "region",
                "awslogs-stream-prefix": "nginx"
            }
        }
        }
    ],
    "cpu": "256",
    "memory": "512"
}
```

b. `service-connect-nginx.json` 파일을 사용하여 작업 정의를 등록합니다.

```
aws ecs register-task-definition --cli-input-json file://service-connect-
nginx.json
```

2. 서비스 생성:

a. 생성하려는 Amazon ECS 서비스의 내용으로 이름이 `service-connect-nginx-service.json`인 파일을 생성합니다. 이 예에서는 이전 단계에서 생성된 태스크 정의를 사용합니다. 예제 태스크 정의에서 `awsvpcConfiguration` 네트워크 모드가 사용되기 때문에 `awsvpc`이 필요합니다.

Fargate 시작 유형을 지정하고 Service Connect를 지원하는 `LATEST` 플랫폼 버전을 지정하여 ECS 서비스를 생성합니다. `securityGroups`와 `subnets`는 Amazon ECS 사용에 대한 요구 사항을 갖춘 VPC에 속해야 합니다. Amazon VPC 콘솔에서 보안 그룹 및 서브넷 ID를 얻을 수 있습니다.

이 서비스는 `serviceConnectConfiguration` 파라미터를 추가하여 Service Connect를 구성합니다. 클러스터에는 기본 네임스페이스가 구성되어 있으므로 네임스페이스는 필요하지 않습니다. 네임스페이스의 ECS에서 실행되는 클라이언트 애플리케이션은 `portName` 및 `clientAliases`의 포트를 사용하여 이 서비스에 연결합니다. 예를 들어, nginx는 루트 위치 `/`에 시작 페이지를 제공하므로 `http://nginx:80/`을 사용하여 이 서비스에 연결할 수 있습니다. Amazon ECS에서 실행되지 않거나 동일한 네임스페이스에 있지 않은 외부 애플리케이션은 작업의 IP 주소와 작업 정의의 포트 번호를 사용하여 Service Connect 프록시를 통해 이 애플리케이션에 연결할 수 있습니다. `tls` 구성을 위해 `awsPcaAuthorityArn`, `kmsKey`, IAM 역할의 `roleArn`에 대한 인증서 `arn`을 추가합니다.

이 서비스는 `logConfiguration`을 사용하여 `stdout` 및 `stderr`의 서비스 연결 프록시 출력을 Amazon CloudWatch Logs로 전송합니다. 이 작업 실행 역할에는 CloudWatch Logs 로그 그룹을 만드는 데 필요한 추가 권한이 없습니다. AWS Management Console 또는 AWS CLI를 사용하여 CloudWatch Logs에서 로그 그룹을 생성합니다. 이 로그 그룹을 생성하고 CloudWatch Logs에 프록시 로그를 저장하는 것이 좋습니다. 프록시 로그를 CloudWatch Logs로 전송하지 않으려면 `logConfiguration`을 제거합니다.

```
{
    "cluster": "tutorial",
    "deploymentConfiguration": {
        "maximumPercent": 200,
        "minimumHealthyPercent": 0
    },
    "deploymentController": {
        "type": "ECS"
    },
    "desiredCount": 1,
    "enableECSManagedTags": true,
    "enableExecuteCommand": true,
    "launchType": "FARGATE",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "assignPublicIp": "ENABLED",
            "securityGroups": [
                "sg-EXAMPLE"
            ],
            "subnets": [
                "subnet-EXAMPLE",
                "subnet-EXAMPLE",
```

```
                "subnet-EXAMPLE"
            ]
           }
    },
    "platformVersion": "LATEST",
    "propagateTags": "SERVICE",
    "serviceName": "service-connect-nginx-service",
    "serviceConnectConfiguration": {
        "enabled": true,
        "services": [
            {
                "portName": "nginx",
                "clientAliases": [
                    {
                        "port": 80
                    }
                ],
                "tls": {
                   "issuerCertificateAuthority": {
                      "awsPcaAuthorityArn": "certificateArn"
                   },
                   "kmsKey": "kmsKey",
                   "roleArn": "iamRoleArn"
                }
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-proxy",
                "awslogs-region": "region",
                "awslogs-stream-prefix": "service-connect-proxy"
            }
        }
    },
    "taskDefinition": "service-connect-nginx"
}
```

b. `service-connect-nginx-service.json` 파일을 사용하여 서비스 생성:

```
aws ecs create-service --cluster tutorial --cli-input-json file://service-
connect-nginx-service.json
```

출력:


```
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/tutorial/
service-connect-nginx-service",
        "serviceName": "service-connect-nginx-service",
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
        "loadBalancers": [],
        "serviceRegistries": [],
        "status": "ACTIVE",
        "desiredCount": 1,
        "runningCount": 0,
        "pendingCount": 0,
        "launchType": "FARGATE",
        "platformVersion": "LATEST",
        "platformFamily": "Linux",
        "taskDefinition": "arn:aws:ecs:us-west-2:123456789012:task-definition/
service-connect-nginx:1",
        "deploymentConfiguration": {
            "deploymentCircuitBreaker": {
                "enable": false,
                "rollback": false
            },
            "maximumPercent": 200,
            "minimumHealthyPercent": 0
        },
        "deployments": [
            {
                "id": "ecs-svc/3763308422771520962",
                "status": "PRIMARY",
                "taskDefinition": "arn:aws:ecs:us-west-2:123456789012:task-
definition/service-connect-nginx:1",
                "desiredCount": 1,
                "pendingCount": 0,
                "runningCount": 0,
                "failedTasks": 0,
                "createdAt": 1661210032.602,
                "updatedAt": 1661210032.602,
                "launchType": "FARGATE",
                "platformVersion": "1.4.0",
                "platformFamily": "Linux",
```

```
                "networkConfiguration": {
                    "awsvpcConfiguration": {
                        "assignPublicIp": "ENABLED",
                        "securityGroups": [
                            "sg-EXAMPLE"
                        ],
                        "subnets": [
                            "subnet-EXAMPLEf",
                            "subnet-EXAMPLE",
                            "subnet-EXAMPLE"
                        ]
                    }
                },
                "rolloutState": "IN_PROGRESS",
                "rolloutStateReason": "ECS deployment ecs-
svc/3763308422771520962 in progress.",
                "failedLaunchTaskCount": 0,
                "replacedTaskCount": 0,
                "serviceConnectConfiguration": {
                    "enabled": true,
                    "namespace": "service-connect",
                    "services": [
                        {
                            "portName": "nginx",
                            "clientAliases": [
                                {
                                    "port": 80
                                }
                            ]
                        }
                    ],
                    "logConfiguration": {
                        "logDriver": "awslogs",
                        "options": {
                            "awslogs-group": "/ecs/service-connect-proxy",
                            "awslogs-region": "us-west-2",
                            "awslogs-stream-prefix": "service-connect-proxy"
                        },
                        "secretOptions": []
                    }
                },
                "serviceConnectResources": [
                    {
                        "discoveryName": "nginx",
```

```
                        "discoveryArn": "arn:aws:servicediscovery:us-
west-2:123456789012:service/srv-EXAMPLE"
                    }
                ]
            }
        ],
        "roleArn": "arn:aws:iam::123456789012:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
        "version": 0,
        "events": [],
        "createdAt": 1661210032.602,
        "placementConstraints": [],
        "placementStrategy": [],
        "networkConfiguration": {
            "awsvpcConfiguration": {
                "assignPublicIp": "ENABLED",
                "securityGroups": [
                    "sg-EXAMPLE"
                ],
                "subnets": [
                    "subnet-EXAMPLE",
                    "subnet-EXAMPLE",
                    "subnet-EXAMPLE"
                ]
            }
        },
        "schedulingStrategy": "REPLICA",
        "enableECSManagedTags": true,
        "propagateTags": "SERVICE",
        "enableExecuteCommand": true
    }
}
```

제공한 `serviceConnectConfiguration`이 출력의 첫 번째 deployment 내부에 표시됩니다. 작업을 변경해야 하는 방식으로 ECS 서비스를 변경하면 Amazon ECS에서 새 배포를 생성합니다.

3단계: 연결할 수 있는지 확인

Service Connect의 구성 및 작동 여부를 확인하려면 다음 단계를 따라 외부 애플리케이션에서 웹 서비스에 연결합니다. 그런 다음 Service Connect 프록시가 생성하는 CloudWatch의 추가 지표를 참조하세요.

외부 애플리케이션에서 웹 서비스에 연결

- 작업 IP 주소와 작업 IP 주소를 사용하는 컨테이너 포트에 연결

  AWS CLI를 사용하여 `aws ecs list-tasks --cluster tutorial`로 작업 ID를 가져옵니다.

  서브넷과 보안 그룹이 작업 정의의 포트에서 퍼블릭 인터넷의 트래픽을 허용하는 경우 컴퓨터에서 퍼블릭 IP에 연결할 수 있습니다. 하지만 퍼블릭 IP는 'describe-tasks'에서 제공되지 않으므로 단계 중 Amazon EC2 AWS Management Console 또는 AWS CLI로 이동하여 탄력적 네트워크 인터페이스의 세부 정보를 가져오는 작업이 포함됩니다.

  이 예시에서는 동일한 VPC의 Amazon EC2 인스턴스가 작업의 프라이빗 IP를 사용합니다. 애플리케이션은 nginx지만 `server: envoy` 헤더는 Service Connect 프록시가 사용됨을 보여줍니다. Service Connect 프록시는 작업 정의의 컨테이너 포트에서 수신 대기합니다.

```
$ curl -v 10.0.19.50:80/
*   Trying 10.0.19.50:80...
* Connected to 10.0.19.50 (10.0.19.50) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.0.19.50
> User-Agent: curl/7.79.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< server: envoy
< date: Tue, 23 Aug 2022 03:53:06 GMT
< content-type: text/html
< content-length: 612
< last-modified: Tue, 16 Apr 2019 13:08:19 GMT
< etag: "5cb5d3c3-264"
< accept-ranges: bytes
< x-envoy-upstream-service-time: 0
<
<!DOCTYPE html>
```

```
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Service Connect 지표 보기

Service Connect 프록시는 CloudWatch 지표에 애플리케이션(HTTP, HTTP2, gRPC 또는 TCP 연결) 지표를 생성합니다. CloudWatch 콘솔을 사용하는 경우 Amazon ECS 네임스페이스에서 DiscoveryName, (DiscoveryName, ServiceName, ClusterName), TargetDiscoveryName, (TargetDiscoveryName, ServiceName, ClusterName)의 추가 지표 차원을 확인합니다. 이러한 지표 및 차원에 대한 자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 View Available Metrics를 참조하세요.

서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결

Amazon ECS 서비스는 Amazon ECS 서비스 검색을 사용하도록 구성할 수도 있습니다. 서비스 검색은 AWS Cloud Map API 태스크를 사용하여 Amazon ECS 서비스에 대한 HTTP 및 DNS 네임스페이스를 관리합니다. 자세한 내용은 AWS Cloud Map 개발자 안내서의 AWS Cloud Map란 무엇입니까?를 참조하세요.

서비스 검색은 다음 AWS 리전에서 사용할 수 있습니다.

리전 이름	리전
미국 동부(버지니아 북부)	us-east-1
미국 동부(오하이오)	us-east-2
미국 서부(캘리포니아 북부)	us-west-1
미국 서부(오리건)	us-west-2
아프리카(케이프타운)	af-south-1
아시아 태평양(홍콩)	ap-east-1
아시아 태평양(뭄바이)	ap-south-1
아시아 태평양(하이데라바드)	ap-south-2
아시아 태평양(도쿄)	ap-northeast-1
아시아 태평양(서울)	ap-northeast-2
아시아 태평양(오사카)	ap-northeast-3
아시아 태평양(싱가포르)	ap-southeast-1
아시아 태평양(시드니)	ap-southeast-2
아시아 태평양(자카르타)	ap-southeast-3
아시아 태평양(멜버른)	ap-southeast-4
캐나다(중부)	ca-central-1
캐나다 서부(캘거리)	ca-west-1
중국(베이징)	cn-north-1
중국(닝샤)	cn-northwest-1

리전 이름	리전
유럽(프랑크푸르트)	eu-central-1
유럽(취리히)	eu-central-2
유럽(아일랜드)	eu-west-1
유럽(런던)	eu-west-2
유럽(파리)	eu-west-3
유럽(밀라노)	eu-south-1
유럽(스톡홀름)	eu-north-1
이스라엘(텔아비브)	il-central-1
유럽(스페인)	eu-south-2
중동(UAE)	me-central-1
중동(바레인)	me-south-1
남아메리카(상파울루)	sa-east-1
AWS GovCloud(미국 동부)	us-gov-east-1
AWS GovCloud(미국 서부)	us-gov-west-1

서비스 검색 개념

서비스 검색은 다음과 같은 구성 요소로 이루어집니다.

- 서비스 검색 네임스페이스: 트래픽을 라우팅할 동일한 도메인 이름(예: `example.com`)을 공유하는 서비스 검색 서비스의 논리적 그룹입니다. `aws servicediscovery create-private-dns-namespace` 명령을 직접 호출하거나 Amazon ECS 콘솔을 사용하여 네임스페이스를 생성할 수 있습니다. `aws servicediscovery list-namespaces` 명령을 사용하여 현재 계정에서 생성한 네임스페이스에 대한 요약 정보를 볼 수 있습니다. 서비스 검색 명령에 대한 자세한 정보는 AWS

Cloud Map(서비스 검색) AWS CLI 참조 가이드의 `create-private-dns-namespace` 및 `list-namespaces`를 참조하세요.

- 서비스 검색 서비스: 서비스 검색 네임스페이스 내에 존재하며 네임스페이스에 대한 서비스 이름과 DNS 구성으로 구성되어 있습니다. 핵심 구성 요소는 다음과 같습니다.
  - 서비스 레지스트리: DNS 또는 AWS Cloud Map API 태스크를 통해 서비스를 검색하고 서비스 연결에 사용할 수 있는 하나 이상의 가용 엔드포인트를 가져올 수 있도록 허용합니다.
- 서비스 검색 인스턴스: 서비스 검색 서비스 내에 존재하며 서비스 디렉터리의 각 Amazon ECS 서비스와 연결된 속성들로 구성됩니다.
  - 인스턴스 속성: 다음 메타데이터는 서비스 검색을 사용하도록 구성된 각 Amazon ECS 서비스에 대한 사용자 지정 속성으로 추가됩니다.
    - **`AWS_INSTANCE_IPV4`** – A 레코드의 경우, DNS 쿼리 및 AWS Cloud Map에 응답하여 Route 53이 반환하는 IPv4 주소는 인스턴스 세부 정보(예: `192.0.2.44`)를 검색할 때 반환됩니다.
    - **`AWS_INSTANCE_PORT`** – 서비스 검색 서비스와 연계된 포트 값입니다.
    - **`AVAILABILITY_ZONE`** – 작업이 시작된 가용 영역입니다. EC2 시작 유형을 사용하는 작업의 경우, 이것은 컨테이너 인스턴스가 존재하는 가용 영역입니다. Fargate 시작 유형을 사용하는 작업의 경우, 이것은 탄력적 네트워크 인터페이스가 존재하는 가용 영역입니다.
    - **`REGION`** – 작업이 존재하는 리전입니다.
    - **`ECS_SERVICE_NAME`** – 작업이 속한 Amazon ECS 서비스의 이름입니다.
    - **`ECS_CLUSTER_NAME`** – 작업이 속한 Amazon ECS 클러스터의 이름입니다.
    - **`EC2_INSTANCE_ID`** – 작업이 배치된 컨테이너 인스턴스의 ID입니다. 작업에서 Fargate 시작 유형을 사용하는 경우, 이 사용자 지정 속성은 추가되지 않습니다.
    - **`ECS_TASK_DEFINITION_FAMILY`** – 작업에서 사용하는 태스크 정의 패밀리입니다.
    - **`ECS_TASK_SET_EXTERNAL_ID`** – 작업 세트가 외부 배포를 위해 생성되고 서비스 검색 레지스트리와 연관되는 경우 `ECS_TASK_SET_EXTERNAL_ID` 속성에는 작업 세트의 외부 ID가 포함됩니다.
- Amazon ECS 상태 확인: Amazon ECS는 컨테이너 수준의 상태 확인을 정기적으로 수행합니다. 상태 확인을 전달하지 않는 엔드포인트는 DNS 라우팅에서 제거되고 비정상으로 상태가 표시됩니다.

서비스 검색 고려 사항

서비스 검색을 사용할 때는 다음 사항을 고려해야 합니다.

- 플랫폼 버전 v1.1.0 이상을 사용 중인 Fargate의 작업에 대해 서비스 검색이 지원됩니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.

- 서비스 검색을 사용하도록 구성된 서비스는 서비스당 작업 수가 1,000개로 제한됩니다. 이는 Route 53 서비스 할당량 때문입니다.
- Amazon ECS 콘솔의 서비스 생성 워크플로는 서비스를 프라이빗 DNS 네임스페이스에 등록하는 것만 지원합니다. AWS Cloud Map 프라이빗 DNS 네임스페이스가 생성되면 Route 53 프라이빗 호스팅 영역이 자동으로 생성됩니다.
- DNS 확인에 성공하려면 VPC DNS 속성을 구성해야 합니다. 속성을 구성하는 방법에 대한 자세한 정보는 Amazon VPC 사용 설명서의 VPC에서 DNS 지원을 참조하세요.
- 서비스 검색 서비스용으로 생성된 DNS 레코드는 퍼블릭 네임스페이스가 사용될 때도 퍼블릭 IP 주소 대신 항상 작업의 프라이빗 IP 주소로 등록합니다.
- 서비스 검색을 위해서는 작업에서 `awsvpc`, `bridge` 또는 `host` 네트워크 모드(`none`는 지원되지 않음)를 지정해야 합니다.
- 서비스 태스크 정의가 `awsvpc` 네트워크 모드를 사용하는 경우 각 서비스 태스크에 대해 `A` 또는 SRV 레코드를 임의로 조합해 생성할 수 있습니다. SRV 레코드를 사용하는 경우에는 포트가 필요합니다.
- 서비스 태스크 정의에서 `bridge` 또는 `host` 네트워크 모드를 사용하는 경우 SRV 레코드는 유일하게 지원되는 DNS 레코드 유형입니다. 각 서비스 태스크에 대한 SRV 레코드를 생성합니다. SRV 레코드는 태스크 정의로부터 컨테이너 이름과 컨테이너 포트 조합을지정해야 합니다.
- 서비스 검색 서비스를 위한 DNS 레코드는 VPC 내에서 쿼리가 가능합니다. 이들은 `<service-discovery-service-name>.<service-discovery-namespace>` 형식을 사용합니다.
- 서비스 이름에 대해 DNS 쿼리를 수행할 때 `A` 레코드는 태스크에 해당하는 여러 개의 IP 주소를 반환합니다. SRV 레코드는 각 작업당 여러 개의 IP 주소와 포트를 반환합니다.
- 정상 레코드가 8개 이하일 경우 Route 53은 모든 DNS 쿼리에 모든 정상 레코드로 응답합니다.
- 모든 레코드가 비정상일 경우 Route 53는 최대 8개의 비정상 레코드로 DNS 쿼리에 응답합니다.
- 로드 밸런서 뒤에 있는 서비스에 대한 서비스 검색을 구성할 수 있지만, 서비스 검색 트래픽은 항상 로드 밸런서가 아닌 태스크로 라우팅됩니다.
- 서비스 검색은 Classic Load Balancer의 사용을 지원하지 않습니다.
- 서비스 검색 서비스에 대해 Amazon ECS가 관리하는 컨테이너 수준의 상태 확인을 사용하는 것이 좋습니다.
  - HealthCheckCustomConfig—Amazon ECS가 사용자를 대신하여 상태 확인을 관리합니다. Amazon ECS는 컨테이너와 상태 확인, 태스크 상태의 정보를 사용하여 AWS Cloud Map을 통해 상태를 업데이트합니다. 서비스 검색 서비스를 생성할 때 `--health-check-custom-config` 파라미터를 사용하여 이를 지정합니다. 자세한 정보는 AWS Cloud Map API 참조의 HealthCheckCustomConfig를 참조하세요.

- 서비스 검색이 사용될 때 생성되는 AWS Cloud Map 리소스는 수동으로 정리해야 합니다.
- 작업과 인스턴스는 컨테이너 상태 확인이 값을 반환할 때까지 UNHEALTHY로 등록됩니다. 상태 확인을 통과하면 상태가 HEALTHY로 업데이트됩니다. 컨테이너 상태 확인이 실패하면 서비스 검색 인스턴스의 등록이 취소됩니다.

서비스 검색 가격

Amazon ECS 서비스 검색을 사용하는 고객들에게는 Route 53 리소스 및 AWS Cloud Map 검색 API 작업에 대해 요금이 부과됩니다. 여기에는 서비스 레지스트리에 대한 Route 53 호스팅 영역 및 쿼리 생성을 위한 비용이 포함됩니다. 자세한 정보는 AWS Cloud Map 개발자 안내서의 AWS Cloud Map 요금을 참조하세요.

Amazon ECS는 컨테이너 수준의 상태 확인을 수행하고 이를 AWS Cloud Map 사용자 지정 상태 확인 API 작업에 공개합니다. 고객들은 추가 비용 없이 이를 사용할 수 있습니다. 공개된 태스크에 대해 추가적인 네트워크 상태 확인을 구성하는 경우에는 이러한 상태 확인에 대해 요금이 부과됩니다.

서비스 검색을 사용하는 새 Amazon ECS 서비스 생성

AWS CLI를 사용하여 서비스 검색을 사용하는 Fargate 태스크가 포함된 서비스를 생성하는 방법을 알아보세요.

서비스 검색을 지원하는 AWS 리전 목록은 서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결 섹션을 참조하세요.

Fargate가 지원되는 리전에 대한 자세한 정보는 the section called “AWS Fargate 리전” 섹션을 참조하세요.

> **Note**
>
> 듀얼 스택 서비스 엔드포인트를 사용하면 AWS CLI, SDK 및 Amazon ECS API에서 IPv4 및 IPv6 모두를 통해 Amazon ECS와 상호 작용할 수 있습니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

사전 조건

이 자습서를 시작하기 전에 다음 사전 조건을 충족하는지 확인하세요.

- 최신 버전의 AWS CLI가 설치 및 구성됩니다. 자세한 내용은 AWS CLI 최신 버전 설치 또는 업데이트를 참조하세요.

- Amazon ECS 사용 설정 섹션에 설명된 단계를 완료했습니다.
- IAM 사용자는 AmazonECS_FullAccess IAM 정책 예제에 지정된 필수 권한을 가집니다.
- 하나 이상의 VPC 및 보안 그룹을 생성했습니다. 자세한 내용은 the section called “Virtual Private Cloud 생성” 섹션을 참조하세요.

1단계: AWS Cloud Map에서 서비스 검색 리소스 생성

이 단계에 따라 서비스 검색 네임스페이스 및 서비스 검색 서비스를 생성합니다.

1. 프라이빗 Cloud Map 서비스 검색 네임스페이스를 생성합니다. 이 예에서는 이름이 `tutorial`인 네임스페이스를 생성합니다. *`vpc-abcd1234`*를 기존 VPC 중 하나의 ID로 교체합니다.

   ```
   aws servicediscovery create-private-dns-namespace \
         --name tutorial \
         --vpc vpc-abcd1234
   ```

   이 명령의 출력은 다음과 같습니다.

   ```
   {
       "OperationId": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e"
   }
   ```

2. 이전 단계의 출력에서 `OperationId`를 사용하여 프라이빗 네임스페이스가 성공적으로 생성되었는지 확인합니다. 후속 명령에서 이를 사용하기 때문에 네임스페이스 ID를 기록해 둡니다.

   ```
   aws servicediscovery get-operation \
         --operation-id h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e
   ```

   출력값은 다음과 같습니다.

   ```
   {
       "Operation": {
           "Id": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e",
           "Type": "CREATE_NAMESPACE",
           "Status": "SUCCESS",
           "CreateDate": 1519777852.502,
           "UpdateDate": 1519777856.086,
           "Targets": {
              "NAMESPACE": "ns-uejictsjen2i4eeg"
           }
   ```

```
    }
}
```

3. 이전 단계의 출력에서 NAMESPACE ID를 사용하여 서비스 검색 서비스를 생성합니다. 이 예제에서는 myapplication라는 서비스를 생성합니다. 후속 명령에서 이를 사용하기 때문에 서비스 ID 및 ARN을 기록해 둡니다.

```
aws servicediscovery create-service \
      --name myapplication \
      --dns-config "NamespaceId="ns-
uejictsjen2i4eeg",DnsRecords=[{Type="A",TTL="300"}]" \
      --health-check-custom-config FailureThreshold=1
```

출력값은 다음과 같습니다.

```
{
    "Service": {
       "Id": "srv-utcrh6wavdkggqtk",
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-
utcrh6wavdkggqtk",
        "Name": "myapplication",
        "DnsConfig": {
            "NamespaceId": "ns-uejictsjen2i4eeg",
            "DnsRecords": [
                {
                    "Type": "A",
                    "TTL": 300
                }
            ]
        },
        "HealthCheckCustomConfig": {
            "FailureThreshold": 1
        },
        "CreatorRequestId": "e49a8797-b735-481b-a657-b74d1d6734eb"
    }
}
```

2단계: Amazon ECS 리소스 생성

이 단계에 따라 Amazon ECS 클러스터, 태스크 정의, 서비스를 생성합니다.

1. Amazon ECS 클러스터를 생성합니다. 이 예에서는 이름이 tutorial인 클러스터를 생성합니다.

```
aws ecs create-cluster \
      --cluster-name tutorial
```

2. Fargate와 호환되며 awsvpc 네트워크 모드를 사용하는 태스크 정의를 등록합니다. 다음 단계를 따릅니다.

   a. 다음과 같은 태스크 정의의 내용으로 이름이 fargate-task.json인 파일을 생성합니다.

```
{
    "family": "tutorial-task-def",
        "networkMode": "awsvpc",
        "containerDefinitions": [
            {
                "name": "sample-app",
                "image": "public.ecr.aws/docker/library/httpd:2.4",
                "portMappings": [
                    {
                        "containerPort": 80,
                        "hostPort": 80,
                        "protocol": "tcp"
                    }
                ],
                "essential": true,
                "entryPoint": [
                    "sh",
                    "-c"
                ],
                "command": [
                    "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style>
 </head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample
 App</h1> <h2>Congratulations!</h2> <p>Your application is now running on a
 container in Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/
htdocs/index.html && httpd-foreground\""
                ]
            }
        ],
        "requiresCompatibilities": [
            "FARGATE"
        ],
        "cpu": "256",
```

```
        "memory": "512"
}
```

b. fargate-task.json 파일을 사용하여 태스크 정의를 등록합니다.

```
aws ecs register-task-definition \
      --cli-input-json file://fargate-task.json
```

3. 다음 단계에 따라 ECS 서비스를 생성합니다.

a. 생성하려는 ECS 서비스의 내용으로 이름이 ecs-service-discovery.json인 파일을 생성합니다. 이 예에서는 이전 단계에서 생성된 태스크 정의를 사용합니다. 예제 태스크 정의에서 awsvpcConfiguration 네트워크 모드가 사용되기 때문에 awsvpc이 필요합니다.

Fargate 시작 유형을 지정하고 서비스 검색을 지원하는 LATEST 플랫폼 버전을 지정하여 ECS 서비스를 생성합니다. 서비스 검색 서비스를 AWS Cloud Map에서 생성할 때 registryArn은 반환된 ARN입니다. securityGroups 및 subnets은 반드시 Cloud Map 네임스페이스를 생성하는 데 사용되는 VPC에 속해야 합니다. Amazon VPC 콘솔에서 보안 그룹 및 서브넷 ID를 얻을 수 있습니다.

```
{
    "cluster": "tutorial",
    "serviceName": "ecs-service-discovery",
    "taskDefinition": "tutorial-task-def",
    "serviceRegistries": [
       {
          "registryArn":
 "arn:aws:servicediscovery:region:aws_account_id:service/srv-utcrh6wavdkggqtk"
       }
    ],
    "launchType": "FARGATE",
    "platformVersion": "LATEST",
    "networkConfiguration": {
       "awsvpcConfiguration": {
          "assignPublicIp": "ENABLED",
          "securityGroups": [ "sg-abcd1234" ],
          "subnets": [ "subnet-abcd1234" ]
       }
    },
    "desiredCount": 1
}
```

b. ecs-service-discovery.json를 사용하여 ECS 서비스를 생성합니다.

```
aws ecs create-service \
      --cli-input-json file://ecs-service-discovery.json
```

3단계: AWS Cloud Map에서 서비스 검색 확인

서비스 검색 정보를 쿼리하면 모든 것이 적절하게 생성되었는지 확인할 수 있습니다. 서비스 검색이 구성된 후, AWS Cloud Map API 작업을 사용하거나 VPC 내의 인스턴스로부터 dig를 호출할 수 있습니다. 다음 단계를 따릅니다.

1. 서비스 검색 서비스 ID를 사용하여 서비스 검색 인스턴스를 나열합니다. 리소스 정리를 위해 인스턴스 ID(굵게 표시됨)를 기록해 둡니다.

```
aws servicediscovery list-instances \
      --service-id srv-utcrh6wavdkggqtk
```

출력값은 다음과 같습니다.

```
{
    "Instances": [
        {
            "Id": "16becc26-8558-4af1-9fbd-f81be062a266",
            "Attributes": {
                "AWS_INSTANCE_IPV4": "172.31.87.2"
                "AWS_INSTANCE_PORT": "80",
                "AVAILABILITY_ZONE": "us-east-1a",
                "REGION": "us-east-1",
                "ECS_SERVICE_NAME": "ecs-service-discovery",
                "ECS_CLUSTER_NAME": "tutorial",
                "ECS_TASK_DEFINITION_FAMILY": "tutorial-task-def"
            }
        }
    ]
}
```

2. 서비스 검색 네임스페이스 및 서비스, ECS 클러스터 이름과 같은 추가 파라미터를 사용하여 서비스 검색 인스턴스에 대한 세부 정보를 쿼리합니다.

```
aws servicediscovery discover-instances \
```

```
    --namespace-name tutorial \
    --service-name myapplication \
    --query-parameters ECS_CLUSTER_NAME=tutorial
```

3. 다음 AWS CLI 명령을 사용하면 서비스 검색 서비스의 Route 53 호스팅 영역에서 생성된 DNS 레코드를 쿼리할 수 있습니다.

   a. 네임스페이스 ID를 사용하여 Route 53 호스팅 영역 ID가 포함된 네임스페이스에 대한 정보를 가져옵니다.

   ```
   aws servicediscovery \
         get-namespace --id ns-uejictsjen2i4eeg
   ```

   출력값은 다음과 같습니다.

   ```
   {
       "Namespace": {
           "Id": "ns-uejictsjen2i4eeg",
           "Arn": "arn:aws:servicediscovery:region:aws_account_id:namespace/ns-
   uejictsjen2i4eeg",
           "Name": "tutorial",
           "Type": "DNS_PRIVATE",
           "Properties": {
                "DnsProperties": {
                   "HostedZoneId": "Z35JQ4ZFDRYPLV"
               }
           },
           "CreateDate": 1519777852.502,
           "CreatorRequestId": "9049a1d5-25e4-4115-8625-96dbda9a6093"
       }
   }
   ```

   b. 이전 단계에서 Route 53 호스팅 영역 ID를 사용하여(굵게 표시된 텍스트 참조) 해당 호스팅 영역에 대한 리소스 레코드 세트를 가져옵니다.

   ```
   aws route53 list-resource-record-sets \
         --hosted-zone-id Z35JQ4ZFDRYPLV
   ```

4. dig를 사용하여 VPC 내의 인스턴스에서 DNS를 쿼리할 수 있습니다.

   ```
   dig +short myapplication.tutorial
   ```

4단계: 정리

이 자습서를 완료한 후에 사용하지 않는 리소스에 대한 요금이 발생하는 것을 방지하기 위해 연결된 리소스를 정리합니다. 다음 단계를 따릅니다.

1. 이전에 기록한 서비스 ID 및 인스턴스 ID를 사용하여 서비스 검색 서비스 인스턴스의 등록을 취소합니다.

```
aws servicediscovery deregister-instance \
      --service-id srv-utcrh6wavdkggqtk \
      --instance-id 16becc26-8558-4af1-9fbd-f81be062a266
```

   출력값은 다음과 같습니다.

```
{
    "OperationId": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv"
}
```

2. 이전 단계의 출력에서 OperationId를 사용하여 서비스 검색 서비스 인스턴스의 등록이 성공적으로 취소되었는지 확인합니다.

```
aws servicediscovery get-operation \
      --operation-id xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv
```

```
{
  "Operation": {
        "Id": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv",
        "Type": "DEREGISTER_INSTANCE",
        "Status": "SUCCESS",
        "CreateDate": 1525984073.707,
        "UpdateDate": 1525984076.426,
        "Targets": {
            "INSTANCE": "16becc26-8558-4af1-9fbd-f81be062a266",
            "ROUTE_53_CHANGE_ID": "C5NSRG1J4I1FH",
            "SERVICE": "srv-utcrh6wavdkggqtk"
        }
    }
}
```

3. 서비스 ID를 사용하여 서비스 검색 서비스를 삭제합니다.

```
aws servicediscovery delete-service \
      --id srv-utcrh6wavdkggqtk
```

4. 네임스페이스 ID를 사용하여 서비스 검색 네임스페이스를 삭제합니다.

```
aws servicediscovery delete-namespace \
      --id ns-uejictsjen2i4eeg
```

출력값은 다음과 같습니다.

```
{
    "OperationId": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj"
}
```

5. 이전 단계의 출력에서 OperationId를 사용하여 서비스 검색 네임스페이스가 성공적으로 삭제되었는지 확인합니다.

```
aws servicediscovery get-operation \
      --operation-id c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj
```

출력값은 다음과 같습니다.

```
{
    "Operation": {
        "Id": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj",
        "Type": "DELETE_NAMESPACE",
        "Status": "SUCCESS",
        "CreateDate": 1525984602.211,
        "UpdateDate": 1525984602.558,
        "Targets": {
            "NAMESPACE": "ns-rymlehshst7hhukh",
            "ROUTE_53_CHANGE_ID": "CJP2A2M86XW3O"
        }
    }
}
```

6. Amazon ECS 서비스에 대해 원하는 개수를 0으로 업데이트합니다. 다음 단계에서 서비스를 삭제하려면 이를 반드시 수행해야 합니다.

```
aws ecs update-service \
```

```
      --cluster tutorial \
      --service ecs-service-discovery \
      --desired-count 0
```

7. Amazon ECS 서비스를 삭제합니다.

```
aws ecs delete-service \
      --cluster tutorial \
      --service ecs-service-discovery
```

8. Amazon ECS 클러스터를 삭제합니다.

```
aws ecs delete-cluster \
      --cluster tutorial
```

Amazon VPC Lattice를 사용하여 Amazon ECS 서비스 연결, 관찰 및 보안 유지

Amazon VPC Lattice는 Amazon ECS 고객이 코드를 변경하지 않고도 여러 AWS 컴퓨팅 서비스, VPC, 계정에 빌드된 애플리케이션을 관찰, 보안 및 모니터링하도록 지원하는 완전관리형 애플리케이션 네트워킹 서비스입니다.

VPC Lattice에서는 컴퓨팅 리소스 모음인 대상 그룹을 사용합니다. 이러한 대상은 애플리케이션 또는 서비스를 실행하며, Amazon EC2 인스턴스, IP 주소, Lambda 함수 및 Application Load Balancer일 수 있습니다. 이제는 고객이 Amazon ECS 서비스를 VPC Lattice 대상 그룹과 연결하여 Amazon ECS 작업을 VPC Lattice의 IP 대상으로 활성화할 수 있습니다. Amazon ECS에서는 등록된 서비스에 대한 태스크가 시작될 때 VPC Lattice 대상 그룹에 태스크를 자동으로 등록합니다.

> **Note**
>
> 5개 VPC Lattice 구성을 사용하면 구성을 더 적게 사용할 때보다 배포 시간이 약간 더 길어질 수 있습니다.

리스너 규칙은 조건이 충족될 때 지정된 대상 그룹에 트래픽을 전달하는 데 사용됩니다. 리스너에서는 사용자가 구성한 포트의 프로토콜을 사용하여 연결 요청을 확인합니다. 서비스는 사용자가 리스너를 구성할 때 정의한 규칙에 따라 등록된 대상으로 요청을 라우팅합니다.

또한 Amazon ECS에서는 VPC Lattice 상태 확인에 따라 비정상이 되는 경우 태스크를 자동으로 바꿉니다. VPC Lattice와 연결되면 Amazon ECS 고객이 클러스터, VPC 및 AWS Resource Access

Manager가 있는 계정 전체의 서비스에 연결, 권한 부여 및 인증을 위한 IAM 통합과 고급 트래픽 관리 특성과 같은 VPC Lattice의 다른 여러 컴퓨팅 간 연결, 보안 및 관찰성 특성도 활용할 수 있습니다.

Amazon ECS 고객이 다음과 같은 방법으로 VPC Lattice의 이점을 누릴 수 있습니다.

- 증가하는 개발자 생산성 - VPC Lattice는 사용자가 특성 구축에 집중할 수 있도록 하여 개발자 생산성을 높이는 한편, VPC Lattice는 모든 컴퓨팅 플랫폼에서 네트워킹, 보안 및 관찰성 과제를 균일한 방법으로 처리합니다.
- 좋아지는 보안 태세 - VPC Lattice를 사용하면 개발자가 애플리케이션 및 컴퓨팅 플랫폼 간의 통신을 쉽게 인증하고 보호하며, 전송 중 암호화를 적용하고, VPC Lattice 인증 정책을 통해 세부적인 액세스 제어를 적용할 수 있습니다. 이에 따라 업계를 선도하는 규제 및 규정 준수 요구 사항을 충족하는 더 강력한 보안 태세를 채택할 수 있습니다.
- 향상되는 애플리케이션 확장성 및 복원력 - VPC Lattice를 사용하면 경로, 헤더 및 메서드 기반 라우팅, 인증, 권한 부여 및 모니터링과 같은 특성을 사용하여 배포된 애플리케이션의 네트워크를 생성할 수 있습니다. 이러한 이점은 워크로드에 대한 리소스 오버헤드 없이 제공되며 상당한 지연 시간 추가 없이 초당 수백만 개의 요청을 생성하는 다중 클러스터 배포를 지원할 수 있습니다.
- 이기종 인프라를 통한 배포 유연성 - VPC Lattice에서는 Amazon ECS, Fargate, Amazon EC2, Amazon EKS 및 Lambda와 같은 모든 컴퓨팅 서비스에 걸쳐 일관된 특성이 제공되며, 조직에서 각 애플리케이션에 적합한 인프라를 선택할 수 있는 유연성이 허용됩니다.

VPC Lattice가 다른 Amazon ECS 서비스와 연동하는 방식

VPC Lattice와 Amazon ECS를 함께 사용하면 다른 Amazon ECS 서비스를 사용하는 방식이 변경될 수 있지만 다른 서비스는 동일하게 유지됩니다.

Application Load Balancers

더는 VPC Lattice에서 Application Load Balancer 대상 그룹 유형과 함께 사용하여 Amazon ECS 서비스에 연결하는 특정 Application Load Balancer를 생성할 필요가 없습니다. 그 대신에 VPC Lattice 대상 그룹으로 Amazon ECS 서비스만 구성하면 됩니다. 또한 Application Load Balancer를 Amazon ECS와 동시에 사용하도록 선택할 수도 있습니다.

Amazon ECS 롤링 배포

Amazon ECS 롤링 배포만 VPC Lattice와 연동하며, Amazon ECS에서는 배포 중 태스크를 안전하게 가져오고 서비스에서 제거합니다. 코드 배포 및 블루/그린 배포는 지원되지 않습니다.

VPC Lattice에 대한 자세한 내용은 Amazon VPC Lattice 사용 설명서를 참조하세요.

VPC Lattice를 사용하는 서비스 생성

AWS Management Console 또는 AWS CLI를 사용하여 VPC Lattice로 서비스를 생성할 수 있습니다.

사전 조건

이 자습서를 시작하기 전에 다음 사전 조건을 충족하는지 확인하세요.

- 최신 버전의 AWS CLI가 설치 및 구성됩니다. 자세한 정보는 AWS Command Line Interface 설치 섹션을 참조하세요.

> **Note**
>
> 듀얼 스택 서비스 엔드포인트를 사용하면 AWS CLI, SDK 및 Amazon ECS API에서 IPv4 및 IPv6 모두를 통해 Amazon ECS와 상호 작용할 수 있습니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

- Amazon ECS 사용 설정 섹션에 설명된 단계를 완료했습니다.
- IAM 사용자는 AmazonECS_FullAccess IAM 정책 예제에 지정된 필수 권한을 가집니다.

AWS Management Console에서 VPC Lattice를 사용하는 서비스 생성

다음과 같은 단계에 따라 AWS Management Console을 사용하여 VPC Lattice로 서비스를 생성합니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 페이지에서 Clusters(클러스터)를 선택합니다.
3. 클러스터 페이지에서 서비스를 생성할 클러스터를 선택합니다.
4. Services(서비스) 탭에서 Create(생성)를 선택합니다.

   이전에 서비스를 생성한 적이 없는 경우 콘솔을 사용하여 Amazon ECS 서비스 생성에서 찾을 수 있는 단계를 따른 다음에 VPC Lattice 단원에 도달하면 다음과 같은 단계를 계속 진행합니다.
5. 버튼을 선택하여 VPC Lattice 활성화를 선택합니다.
6. Amazon ECS에 대한 ECS 인프라 역할에 대해 기존 역할을 사용하려면 VPC Lattice 대상 그룹을 생성할 때 이미 생성하여 사용하는 역할을 사용합니다. 새 역할을 생성하려면 ECS 인프라 역할을 생성합니다.
7. VPC를 선택합니다.

VPC는 태스크 정의를 등록할 때 선택한 네트워킹 모드에 따라 달라집니다. EC2 시작 유형에서 `host` 또는 `network` 모드를 사용하는 경우 VPC를 선택합니다.

`awsvpc` 모드의 경우 네트워킹에서 선택한 VPC를 기반으로 VPC가 자동으로 선택되며 변경할 수 없습니다.

8. 대상 그룹에서 대상 그룹 또는 그룹을 선택합니다. 대상 그룹을 하나 이상 선택해야 하며 최대 5개를 선택할 수 있습니다. 대상 그룹을 더 추가하려면 대상 그룹 추가를 선택합니다. 선택한 각 대상 그룹의 포트 이름, 프로토콜 및 포트를 선택합니다. 대상 그룹을 삭제하려면 제거를 선택합니다.

> **Note**
>
> - 기존 대상 그룹을 추가하려면 AWS CLI를 사용해야 합니다. AWS CLI를 사용하여 대상 그룹을 추가하는 방법에 대한 지침은 AWS Command Line Interface 참조의 register-targets를 참조하세요.
> - VPC Lattice 서비스에는 여러 대상 그룹이 있을 수 있지만, 각 대상 그룹은 하나의 서비스에만 추가할 수 있습니다.

9. 이제 VPC Lattice 콘솔로 이동하여 설정을 계속합니다. 여기에서는 리스너 기본 작업 또는 기존 VPC Lattice 서비스의 규칙에 새 대상 그룹을 포함합니다.

자세한 내용은 VPC Lattice 서비스를 위한 리스너 규칙을 참조하세요.

> **Important**
>
> 보안 그룹 또는 태스크에 대한 인바운드 규칙 `vpc-lattice` 접두사를 허용해야 하며, 상태 확인에 실패할 수 있습니다.

AWS CLI에서 VPC Lattice를 사용하는 서비스 생성

AWS CLI를 사용하여 VPC Lattice로 서비스를 생성합니다. *`user input placeholder`*를 사용자의 정보로 바꿉니다.

1. 대상 그룹 구성 파일을 생성합니다. 다음은 `tg-config.json`이라는 예제입니다.

```
{
    "ipAddressType": "IPV4",
```

```
    "port": 443,
    "protocol": "HTTPS",
    "protocolVersion": "HTTP1",
    "vpcIdentifier": "vpc-f1663d9868EXAMPLE"
}
```

2. 다음과 같은 명령을 사용하여 VPC Lattice 대상 그룹을 생성합니다.

```
aws vpc-lattice create-target-group \
    --name my-lattice-target-group-ip \
    --type IP \
    --config file://tg-config.json
```

출력 예시:

```
{
    "arn": "arn:aws:vpc-lattice:us-east-2:123456789012:targetgroup/
tg-0eaa4b9ab4EXAMPLE",
    "config": {
        "healthCheck": {
            "enabled": true,
            "healthCheckIntervalSeconds": 30,
            "healthCheckTimeoutSeconds": 5,
            "healthyThresholdCount": 5,
            "matcher": {
                "httpCode": "200"
            },
            "path": "/",
            "protocol": "HTTPS",
            "protocolVersion": "HTTP1",
            "unhealthyThresholdCount": 2
        },
        "ipAddressType": "IPV4",
        "port": 443,
        "protocol": "HTTPS",
        "protocolVersion": "HTTP1",
        "vpcIdentifier": "vpc-f1663d9868EXAMPLE"
    },
    "id": "tg-0eaa4b9ab4EXAMPLE",
    "name": "my-lattice-target-group-ip",
    "status": "CREATE_IN_PROGRESS",
    "type": "IP"
```

```
}
```

3. *ecs-service-vpc-lattice.json*이라는 다음과 같은 JSON 파일은 Amazon ECS 서비스를 VPC Lattice 대상 그룹에 연결하는 데 사용하는 예제입니다. 아래 예제의 `portName`은 태스크 정의의 `portMappings` 속성의 `name` 필드에 정의한 것과 동일합니다.

```
{
    "serviceName": "ecs-service-vpc-lattice",
    "taskDefinition": "ecs-task-def",
        "vpcLatticeConfigurations": [
        {
            "targetGroupArn": "arn:aws:vpc-lattice:us-
west-2:123456789012:targetgroup/tg-0eaa4b9ab4EXAMPLE",
            "portName": "testvpclattice",
            "roleArn": "arn:aws:iam::123456789012:role/
ecsInfrastructureRoleVpcLattice"
        }
    ],
    "desiredCount": 5,
    "role": "ecsServiceRole"
}
```

다음과 같은 명령을 사용하여 Amazon ECS 서비스를 생성하고 위의 json 예제를 사용하여 VPC Lattice 대상 그룹에 연결합니다.

```
aws ecs create-service \
    --cluster clusterName \
    --serviceName ecs-service-vpc-lattice \
    --cli-input-json file://ecs-service-vpc-lattice.json
```

> **Note**
>
> VPC Lattice는 Amazon ECS Anywhere에서 지원되지 않습니다.

Amazon ECS 작업이 스케일 인 이벤트로 인해 종료되지 않도록 보호

Amazon ECS 작업 축소 보호를 사용하여 서비스 Auto Scaling 또는 배포의 스케일 인 이벤트로 인해 작업이 종료되지 않도록 보호할 수 있습니다.

특정 애플리케이션에는 사용률이 낮은 시간이나 서비스 배포 중에 스케일 인 이벤트로 인해 미션 크리티컬 작업이 종료되지 않도록 보호하는 메커니즘이 필요합니다. 예:

- 누적 서비스 사용률이 낮더라도 일부 작업을 몇 시간 동안 실행해야 하는 비디오 트랜스코딩 작업과 같은 큐 처리 비동기 애플리케이션이 있습니다.
- 모든 사용자가 로그아웃한 경우에도 서버 재부팅으로 인한 시작 지연 시간을 줄이기 위해 게임 서버를 Amazon ECS 작업으로 실행하는 게임 애플리케이션이 있습니다.
- 새 코드 버전을 배포할 때는 재처리하는 데 비용이 많이 들기 때문에 작업을 계속 실행해야 합니다.

서비스에 속한 작업이 스케일 인 이벤트로 인해 종료되지 않도록 보호하려면 `ProtectionEnabled` 속성을 `true`로 설정합니다. `ProtectionEnabled`을 true로 설정하면 작업이 기본적으로 2시간 동안 보호됩니다. 그러면 `ExpiresInMinutes` 속성을 사용하여 보호 기간을 사용자 지정할 수 있습니다. 최소 1분, 최대 2,880분(48시간) 동안 작업을 보호할 수 있습니다. AWS CLI를 사용하는 경우 `--protection-enabled` 옵션을 지정할 수 있습니다.

작업이 필요한 작업을 완료한 후 `ProtectionEnabled` 속성을 `false`로 설정하여 후속 스케일 인 이벤트에 의해 작업이 종료되도록 할 수 있습니다. AWS CLI를 사용하는 경우 `--no-protection-enabled` 옵션을 지정할 수 있습니다.

작업 축소 보호 메커니즘

Amazon ECS 컨테이너 에이전트 엔드포인트 또는 Amazon ECS API를 사용하여 작업 축소 보호를 설정하고 가져올 수 있습니다.

- Amazon ECS 컨테이너 에이전트 엔드포인트

  보호 필요성을 스스로 결정할 수 있는 작업에는 Amazon ECS 컨테이너 에이전트 엔드포인트를 사용하는 것이 좋습니다. 대기열 기반 또는 작업 처리 워크로드에 이 접근 방식을 사용합니다.

  예를 들어, 컨테이너가 SQS 메시지를 사용하여 작업 처리를 시작하면 컨테이너 내에서 작업 축소 보호 엔드포인트 경로 `$ECS_AGENT_URI/task-protection/v1/state`를 통해 `ProtectionEnabled` 속성을 설정할 수 있습니다. Amazon ECS는 스케일 인 이벤트 중에 이 작업을 종료하지 않습니다. 작업이 완료된 후 동일한 엔드포인트를 사용하여 `ProtectionEnabled` 속성 선택을 해제하여 후속 스케일 인 이벤트 중에 작업을 종료할 수 있습니다.

  Amazon ECS 컨테이너 에이전트 엔드포인트 사용에 대한 자세한 내용은 Amazon ECS 작업 축소 보호 엔드포인트 섹션을 참조하세요.

- Amazon ECS API

애플리케이션에 활성 작업의 상태를 추적하는 구성 요소가 있는 경우 Amazon ECS API를 사용하여 작업 축소 보호를 설정할 수 있습니다. `UpdateTaskProtection`을 사용하여 하나 이상의 작업을 보호됨으로 표시합니다. `GetTaskProtection`을 사용하여 보호 상태를 검색합니다.

애플리케이션이 게임 서버 세션을 Amazon ECS 작업으로 호스팅하는 경우를 예로 들 수 있습니다. 사용자가 서버의 세션(작업)에 로그인하면 작업을 보호된 것으로 표시할 수 있습니다. 사용자가 로그아웃한 후에는 서버를 유휴 상태로 유지해야 하는 요구 사항에 따라 이 작업에 대한 보호 설정을 해제하거나 더 이상 활성 세션이 없는 유사한 작업에 대한 보호 설정을 주기적으로 해제할 수 있습니다.

자세한 내용은 Amazon Elastic Container Service API 참조의 UpdateTaskProtection 및 GetTaskProtection을 참조하세요.

두 가지 방법을 결합할 수 있습니다. 예를 들어 Amazon ECS 에이전트 엔드포인트를 사용하여 컨테이너 내에서 작업 보호를 설정하고 Amazon ECS API를 사용하여 외부 컨트롤러 서비스에서 작업 보호를 제거할 수 있습니다.

고려 사항

작업 축소 보호 사용 전에 다음 사항을 고려하세요.

- 태스크 스케일 인 보호는 서비스에서 배포된 태스크에서만 지원됩니다.
- Amazon ECS 에이전트에는 재시도 메커니즘과 더 간단한 인터페이스가 내장되어 있으므로 Amazon ECS 컨테이너 에이전트 엔드포인트를 사용하는 것이 좋습니다.
- 이미 보호 기능이 켜진 작업에서 `UpdateTaskProtection`을 호출하여 작업 스케일 인 보호 만료 기간을 재설정할 수 있습니다.
- 작업이 필요한 작업을 완료하는 데 필요한 시간을 결정하고 그에 따라 `expiresInMinutes` 속성을 설정합니다. 보호 만료를 필요 이상으로 길게 설정하면 비용이 발생하고 새 작업의 배포가 지연될 수 있습니다.
- 작업 스케일 인 보호는 Amazon ECS 컨테이너 에이전트 `1.65.0` 이상에서 지원됩니다.

  에이전트를 최신 버전으로 업데이트하여 이전 버전의 Amazon ECS 컨테이너 인스턴스를 사용하여 Amazon EC2 인스턴스에서 이 기능에 대한 지원을 추가할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.
- 배포 고려 사항

- 서비스가 롤링 업데이트를 사용하는 경우 새 작업이 생성되지만 이전 버전을 실행하는 작업은 protectionEnabled가 설정 해제되거나 만료될 때까지 종료되지 않습니다. 배포 구성의 maximumPercentage 파라미터를 이전 작업이 보호될 때 새 작업을 생성할 수 있는 값으로 조정할 수 있습니다.
- 블루/그린 업데이트가 적용된 경우 작업에 protectionEnabled가 있으면 보호된 작업이 있는 블루 배포는 제거되지 않습니다. 트래픽은 새로 가동되는 작업으로 전환되고 이전 작업은 protectionEnabled가 해제되거나 만료된 경우에만 제거됩니다. CodeDeploy 또는 CloudFormation 업데이트의 제한 시간에 따라 배포 시간이 초과될 수 있으며 이전 Blue 작업이 여전히 존재할 수 있습니다.
- CloudFormation을 사용하는 경우 업데이트 스택의 제한 시간은 3시간입니다. 따라서 작업 보호를 3시간 이상으로 설정하면 CloudFormation 배포가 실패하고 롤백이 발생할 수 있습니다.

  이전 작업이 보호되는 동안 CloudFormation 스택은 UPDATE_IN_PROGRESS를 표시합니다. 작업 스케일 인 보호가 제거되거나 3시간 내에 만료되면 배포가 성공하고 UPDATE_COMPLETE 상태로 전환됩니다. 배포가 3시간 이상 UPDATE_IN_PROGRESS 상태로 유지되면 실패하고 UPDATE_FAILED 상태가 표시되며, 이전 작업 세트로 롤백됩니다.
- Amazon ECS는 보호 대상 작업으로 인해 배포(롤링 또는 블루/그린)가 안정 상태에 도달하지 못하는 경우 서비스 이벤트를 전송하여 수정 조치를 취할 수 있도록 합니다. 작업의 보호 상태를 업데이트하려고 시도하는 동안 DEPLOYMENT_BLOCKED 오류 메시지가 표시되면 서비스에 대해 바람직한 작업 수보다 보호되는 작업 수가 많음을 의미합니다. 이 오류를 해결하려면 다음 중 하나 이상을 수행합니다.
  - 현재 작업 보호가 만료될 때까지 기다립니다. 그런 다음 작업 보호를 설정합니다.
  - 중지할 수 있는 작업을 확인합니다. 그런 다음 이러한 작업에 대해 protectionEnabled 옵션을 false로 설정한 상태에서 UpdateTaskProtection을 사용합니다.
  - 서비스의 바람직한 작업 수를 보호되는 작업 수 이상으로 늘립니다.

작업 축소 보호에 필요한 IAM 권한

태스크에는 다음 권한이 있는 Amazon ECS 태스크 역할이 있어야 합니다.

- ecs:GetTaskProtection: Amazon ECS 컨테이너 에이전트가 GetTaskProtection을 호출할 수 있습니다.
- ecs:UpdateTaskProtection: Amazon ECS 컨테이너 에이전트가 UpdateTaskProtection을 호출할 수 있습니다.

Amazon ECS 작업 축소 보호 엔드포인트

Amazon ECS 컨테이너 에이전트는 Amazon ECS 작업의 컨테이너에 ECS_AGENT_URI 환경 변수를 자동으로 주입하여 컨테이너 에이전트 API 엔드포인트와 상호 작용하는 방법을 제공합니다.

보호 필요성을 스스로 결정할 수 있는 작업에는 Amazon ECS 컨테이너 에이전트 엔드포인트를 사용하는 것이 좋습니다.

컨테이너가 작업 처리를 시작하면 컨테이너 내에서 작업 축소 보호 엔드포인트 경로 $ECS_AGENT_URI/task-protection/v1/state를 통해 protectionEnabled 속성을 설정할 수 있습니다.

컨테이너 내에서 이 URI에 대한 PUT 요청은 작업 축소 보호를 설정합니다. 이 URI에 대한 GET 요청은 작업의 현재 보호 상태를 반환합니다.

작업 축소 보호 요청 파라미터

다음 요청 파라미터와 함께 ${ECS_AGENT_URI}/task-protection/v1/state 엔드포인트를 사용하여 작업 축소 보호를 설정할 수 있습니다.

ProtectionEnabled

작업을 보호 대상으로 표시하려면 true를 지정합니다. 보호를 제거하고 작업을 종료할 수 있도록 표시하려면 false를 지정합니다.

유형: 부울

필수 여부: 예

ExpiresInMinutes

작업이 보호되는 시간(분)입니다. 최소 1분에서 최대 2,880분(48시간)까지 지정할 수 있습니다. 이 기간 동안에는 서비스 Auto Scaling 또는 배포의 스케일 인 이벤트로 인해 작업이 종료되지 않습니다. 이 기간이 경과하면 protectionEnabled 파라미터가 false로 설정됩니다.

시간을 지정하지 않으면 작업이 120분(2시간) 동안 자동으로 보호됩니다.

유형: 정수

필수 여부: 아니요

다음은 각기 다른 기간으로 작업 보호를 설정하는 예입니다.

기본 기간에 작업을 보호하는 방법 예제

이 예제에서는 기본 시간인 2시간으로 작업을 보호하는 방법을 보여줍니다.

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-
protection/v1/state --data '{"ProtectionEnabled":true}'
```

60분간 작업을 보호하는 방법 예제

이 예제에서는 `expiresInMinutes` 파라미터를 사용하여 60분 동안 작업을 보호하는 방법을 보여줍니다.

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-
protection/v1/state --data '{"ProtectionEnabled":true,"ExpiresInMinutes":60}'
```

24시간 동안 작업을 보호하는 방법 예제

이 예제에서는 `expiresInMinutes` 파라미터를 사용하여 24시간 동안 작업을 보호하는 방법을 보여줍니다.

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-
protection/v1/state --data '{"ProtectionEnabled":true,"ExpiresInMinutes":1440}'
```

Windows 컨테이너에 대한 예제

Windows 컨테이너의 경우 curl 대신 PowerShell의 `Invoke-RestMethod` cmdlet을 사용할 수 있습니다. 다음 예제에서는 이전 curl 명령과 동일한 PowerShell을 보여줍니다.

기본 기간에 Windows 컨테이너 태스크를 보호하는 방법 예제

이 예제에서는 기본 2시간 동안 태스크를 보호하는 방법을 보여줍니다.

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Put -Body
 '{"ProtectionEnabled":true}' -ContentType 'application/json'
```

60분간 Windows 컨테이너 태스크를 보호하는 방법 예제

이 예제에서는 PowerShell에서 `expiresInMinutes` 파라미터를 사용하여 60분 동안 태스크를 보호하는 방법을 보여줍니다.

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Put -Body
 '{"ProtectionEnabled":true,"ExpiresInMinutes":60}' -ContentType 'application/json'
```

24시간 동안 Windows 컨테이너 태스크를 보호하는 방법 예제

이 예제에서는 PowerShell에서 `expiresInMinutes` 파라미터를 사용하여 24시간 동안 태스크를 보호하는 방법을 보여줍니다.

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Put -Body
 '{"ProtectionEnabled":true,"ExpiresInMinutes":1440}' -ContentType 'application/json'
```

PUT 요청은 다음과 같은 응답을 반환합니다.

```
{
  "protection": {
    "ExpirationDate": "2023-12-20T21:57:44.837Z",
    "ProtectionEnabled": true,
    "TaskArn": "arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0"
  }
}
```

작업 축소 보호 응답 파라미터

다음 정보가 작업 스케일 인 보호 엔드포인트(`${ECS_AGENT_URI}/task-protection/v1/state`)에서 JSON 응답으로 반환됩니다.

`ExpirationDate`

작업에 대한 보호가 만료되는 Epoch 시간입니다. 작업이 보호되지 않는 경우 이 값은 null입니다.

`ProtectionEnabled`

작업의 보호 상태입니다. 작업에 대해 축소 보호가 활성화되면 값은 `true`이고, 그렇지 않으면 `false`입니다.

`TaskArn`

컨테이너가 속한 태스크의 전체 Amazon 리소스 이름(ARN)

다음 예제에서는 보호된 작업에 대해 반환된 세부 정보를 보여줍니다.

```
curl --request GET ${ECS_AGENT_URI}/task-protection/v1/state
```

Windows 컨테이너의 경우 다음 PowerShell 명령을 사용하여 보호 상태를 가져옵니다.

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Get
```

```
{
    "protection":{
        "ExpirationDate":"2023-12-20T21:57:44Z",
        "ProtectionEnabled":true,
        "TaskArn":"arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0"
    }
}
```

결함이 발생하면 다음 정보가 반환됩니다.

`Arn`

작업의 전체 Amazon 리소스 이름(ARN)입니다.

`Detail`

실패와 관련된 세부 정보입니다.

`Reason`

실패 이유

다음 예제에서는 보호되지 않는 작업에 대해 반환된 세부 정보를 보여줍니다.

```
{
    "failure":{
        "Arn":"arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0",
        "Detail":null,
        "Reason":"TASK_NOT_VALID"
    }
}
```

예외가 발생하면 다음 정보가 반환됩니다.

`requestID`

예외를 발생시키는 Amazon ECS API 호출에 대한 AWS 요청 ID입니다.

`Arn`

작업 또는 서비스의 전체 Amazon 리소스 이름(ARN)입니다.

`Code`

오류 코드입니다.

`Message`

오류 메시지입니다.

> **Note**
>
> `RequestError` 또는 `RequestTimeout` 오류가 표시되면 이는 네트워킹 문제일 수 있습니다. Amazon ECS에 VPC 엔드포인트를 사용해 보세요.

다음 예제에서는 오류 발생 시 반환되는 세부 정보를 보여줍니다.

```
{
    "requestID":"12345-abc-6789-0123-abc",
    "error":{
        "Arn":"arn:aws:ecs:us-west-2:555555555555:task/my-cluster-
name/1234567890abcdef0",
        "Code":"AccessDeniedException",
        "Message":"User: arn:aws:sts::444455556666:assumed-role/my-ecs-task-
role/1234567890abcdef0 is not authorized to perform: ecs:GetTaskProtection on resource:
 arn:aws:ecs:us-west-2:555555555555:task/test/1234567890abcdef0 because no identity-
based policy allows the ecs:GetTaskProtection action"
    }
}
```

네트워크 문제 또는 Amazon ECS 컨트롤 플레인 중단과 같은 이유로 Amazon ECS 에이전트가 Amazon ECS 엔드포인트에서 응답을 받을 수 없는 경우 다음 오류가 표시됩니다.

```
{
  "error": {
    "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",
    "Code": "RequestCanceled",
    "Message": "Timed out calling Amazon ECS Task Protection API"
  }
}
```

Amazon ECS 에이전트가 Amazon ECS에서 제한 예외를 받으면 다음 오류가 표시됩니다.

```
{
  "requestID": "12345-abc-6789-0123-abc",
  "error": {
    "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",
    "Code": "ThrottlingException",
    "Message": "Rate exceeded"
  }
}
```

Amazon ECS 및 Fargate 워크로드에서 결함 주입 사용

고객은 Amazon EC2 및 Fargate 모두에서 Amazon ECS를 사용한 결함 주입을 활용하여 애플리케이션이 특정 장애 시나리오에 어떻게 대응하는지 테스트할 수 있습니다. 이러한 테스트는 애플리케이션의 성능과 복원력을 최적화하는 데 사용 가능한 정보를 제공합니다.

결함 주입이 활성화되면 Amazon ECS 컨테이너 에이전트가 작업이 새 결함 주입 엔드포인트에 액세스하도록 허용합니다. 결함 주입을 사용하려면 `enableFaultInjection` 작업 정의 파라미터를 값을 `true`로 설정하여 옵트인해야 합니다. 기본값은 `false`입니다.

```
{
    ...
   "enableFaultInjection": true
}
```

> **Note**
>
> 결함 주입은 `awsvpc` 또는 `host` 네트워크 모드를 사용하는 작업에서만 작동합니다. Windows에서는 오류 주입을 사용할 수 없습니다.

AWS Management Console에서 결함 주입을 활성화하는 방법에 대한 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성을 참조하세요.

AWS Fault Injection Service에서 테스트할 기능을 활성화해야 합니다. 자세한 내용은 AWS FISaws:ecs:task 작업 사용을 참조하세요.

> **Note**
>
> 새로운 Amazon ECS 최적화 AMI를 사용하지 않거나 사용자 지정 AMI가 있는 경우 다음 종속성을 설치합니다.

- tc
- sch_netem 커널 모듈

Amazon ECS 결함 주입 엔드포인트

Amazon ECS 컨테이너 에이전트는 Amazon ECS 작업의 컨테이너에 ECS_AGENT_URI 환경 변수를 자동으로 주입하여 컨테이너 에이전트 API 엔드포인트와 상호 작용하는 방법을 제공합니다. 각 엔드포인트에는 /start, /stop, /status 엔드포인트가 포함됩니다. 엔드포인트는 결함 주입이 활성화된 작업의 요청만 수락하며, 각 엔드포인트의 요청 속도 제한은 컨테이너별로 5초당 1개입니다. 제한을 초과할 경우 오류가 발생합니다.

> **Note**
>
> Amazon ECS Agent version 1.88.0+는 결함 주입 엔드포인트를 사용해야 합니다.

결함 주입에 사용할 세 가지 엔드포인트는 다음과 같습니다.

- 네트워크 블랙홀 포트 엔드포인트
- 네트워크 패킷 손실 엔드포인트
- 네트워크 지연 시간 엔드포인트

요청이 성공하면 /start 엔드포인트를 호출할 때 running 메시지와 함께 200 응답 코드가 생성되며, /stop 엔드포인트에 대해 stopped, /status 엔드포인트에 대해 running 또는 not-running이 생성됩니다.

```
{
    "Status": <string>
}
```

요청이 실패하면 다음 오류 코드 중 하나가 반환됩니다.

- 400-잘못된 요청
- 409-결함 주입 요청이 실행 중인 다른 오류와 충돌
- 429-요청에 병목 현상 발생

- 500-서버에 예기치 않은 오류 발생

```
{
 "Error":  <string message>
}
```

> **Note**
>
> 네트워크 지연 시간 오류 하나 또는 네트워크 패킷 손실 오류를 한 번에 하나씩 주입할 수 있습니다. 둘 이상의 결과를 주입하려고 하면 요청이 거부됩니다.

네트워크 블랙홀 포트 엔드포인트

`{ECS_AGENT_URI}/fault/v1/network-blackhole-port` 엔드포인트는 작업의 네트워크 네임스페이스에 있는 특정 포트 및 프로토콜에 대한 인바운드 또는 아웃바운드 트래픽을 삭제하며 다음 두 가지 모드와 호환됩니다.

- awsvpc-변경 사항이 작업 네트워크 네임스페이스에 적용됨
- 호스트-변경 사항이 기본 네트워크 네임스페이스 컨테이너 인스턴스에 적용됨

{ECS_AGENT_URI}/fault/v1/network-blackhole-port/start

이 엔드포인트는 네트워크 블랙홀 포트 결함 주입을 시작하며 다음 파라미터를 갖습니다.

Port

블랙홀 포트 결함 주입에 사용할 지정된 포트입니다.

유형: 정수

필수 여부: 예

프로토콜

블랙홀 포트 결함 주입에 사용할 프로토콜입니다.

타입: 문자열

유효값: `tcp | udp`

필수 여부: 예

TrafficType

결함 주입에 사용되는 트래픽 유형입니다.

타입: 문자열

유효값: `ingress | egress`

필수 여부: 예

SourcesToFilter

오류로부터 보호되는 IPv4 주소 또는 CIDR 블록의 JSON 배열입니다.

유형: 문자열 배열

필수 여부: 아니요

다음은 `start` 엔드포인트 사용에 대한 요청의 예입니다(*###* 값을 자체 값으로 대체).

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-blackhole-port/start

Http method:POST

Request payload:
{
    "Port": 1234,
    "Protocol": "tcp|udp",
    "TrafficType": "ingress|egress"
    "SourcesToFilter": ["${IP1}", "${IP2}", ...],
}
```

{ECS_AGENT_URI}/fault/v1/network-blackhole-port/stop

이 엔드포인트는 요청에 지정된 오류를 중지합니다. 이 엔드포인트에는 다음과 같은 파라미터가 있습니다.

Port

중지해야 하는 오류의 영향을 받는 포트입니다.

유형: 정수

필수 여부: 예

프로토콜

오류를 중지하는 데 사용할 프로토콜입니다.

타입: 문자열

유효값: `tcp | udp`

필수 여부: 예

TrafficType

결함 주입에 사용되는 트래픽 유형입니다.

타입: 문자열

유효값: `ingress | egress`

필수 여부: 예

다음은 `stop` 엔드포인트 사용에 대한 요청의 예입니다(*###* 값을 자체 값으로 대체).

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-blackhole-port/stop

Http method: POST

Request payload:
{
    "Port": 1234,
    "Protocol": "tcp|udp",
    "TrafficType": "ingress|egress",
}
```

{ECS_AGENT_URI}/fault/v1/network-blackhole-port/status

이 엔드포인트는 결함 주입의 상태를 확인하는 데 사용됩니다. 이 엔드포인트에는 다음과 같은 파라미터가 있습니다.

Port

오류 상태를 확인할 때 영향을 받는 포트입니다.

유형: 정수

필수 여부: 예

프로토콜

오류 상태를 확인할 때 사용할 프로토콜입니다.

타입: 문자열

유효값: `tcp | udp`

필수 여부: 예

TrafficType

결함 주입에 사용되는 트래픽 유형입니다.

타입: 문자열

유효값: `ingress | egress`

필수 여부: 예

다음은 `status` 엔드포인트 사용에 대한 요청의 예입니다(### 값을 자체 값으로 대체).

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-blackhole-port/status

Http method: POST

Request payload:
{
   "Port": 1234,
   "Protocol": "tcp|udp",
   "TrafficType": "ingress|egress",
}
```

네트워크 지연 시간 엔드포인트

`{ECS_AGENT_URI}/fault/v1/network-latency` 엔드포인트는 특정 소스로 가는 트래픽 처리를 위해 작업의 네트워크 인터페이스에 지연과 지터를 추가합니다. 엔드포인트는 다음 두 가지 모드와 호환됩니다.

- awsvpc-변경 사항이 작업 네트워크 인터페이스에 적용됨
- 호스트-변경 사항이 기본 네트워크 인터페이스에 적용됨

{ECS_AGENT_URI}/fault/v1/network-latency/start

이 /start 엔드포인트는 네트워크 지연 시간 결함 주입을 시작하며 다음 파라미터를 갖습니다.

DelayMilliseconds

결함 주입에 사용하기 위해 네트워크 인터페이스에 추가할 지연 시간(밀리초)입니다.

유형: 정수

필수 여부: 예

JitterMilliseconds

결함 주입에 사용하기 위해 네트워크 인터페이스에 추가할 지터(밀리초)입니다.

유형: 정수

필수 여부: 예

소스

결함 주입에 사용할 대상인 IPv4 주소 또는 CIDR 블록의 JSON 배열입니다.

유형: 문자열 배열

필수 여부: 예

SourcesToFilter

오류로부터 보호되는 IPv4 주소 또는 CIDR 블록의 JSON 배열입니다. SourcesToFilter는 Sources보다 우선순위가 높습니다.

유형: 문자열 배열

필수 여부: 아니요

다음은 /start 엔드포인트 사용에 대한 요청의 예입니다(### 값을 자체 값으로 대체).

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-latency/start

Http method: POST

Request payload:
{
    "DelayMilliseconds": 123,
```

```
    "JitterMilliseconds": 123,
    "Sources": ["${IP1}", "${IP2}", ...],
    "SourcesToFilter": ["${IP1}", "${IP2}", ...],
}
```

{ECS_AGENT_URI}/fault/v1/network-latency/stop and /status

{ECS_AGENT_URI}/fault/v1/network-latency/stop 엔드포인트가 오류를 중지하고 {ECS_AGENT_URI}/fault/v1/network-latency/status가 오류 상태를 확인합니다.

다음은 /stop 및 /status 엔드포인트 사용에 대한 두 가지 예제 요청입니다. 둘 다 POST HTTP 메서드를 사용합니다.

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-latency/stop
```

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-latency/status
```

네트워크 패킷 손실 엔드포인트

{ECS_AGENT_URI}/fault/v1/network-packet-loss 엔드포인트는 지정된 네트워크 인터페이스에 패킷 손실을 추가합니다. 이 엔드포인트는 다음 두 가지 모드와 호환됩니다.

- awsvpc-변경 사항이 작업 네트워크 인터페이스에 적용됨
- 호스트-변경 사항이 기본 네트워크 인터페이스에 적용됨

{ECS_AGENT_URI}/fault/v1/network-packet-loss/start

이 /start 엔드포인트는 네트워크 패킷 손실 결함 주입을 시작하며 다음 파라미터를 갖습니다.

LossPercent

패킷 손실의 백분율

유형: 정수

필수 여부: 예

소스

결함 주입 테스트에 사용할 IPv4 주소 또는 CIDR 블록의 JSON 배열입니다.

유형: 문자열 배열

필수 여부: 예

SourcesToFilter

오류로부터 보호되는 IPv4 주소 또는 CIDR 블록의 JSON 배열입니다. SourcesToFilter는 Sources보다 우선순위가 높습니다.

유형: 문자열 배열

필수 여부: 아니요

다음은 start 엔드포인트 사용에 대한 요청의 예입니다(### 값을 자체 값으로 대체).

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-packet-loss/start

Http method: POST

{
    "LossPercent": 6,
    "Sources": ["${IP1}", "${IP2}", ...],
    "SourcesToFilter": ["${IP1}", "${IP2}", ...],
}
```

{ECS_AGENT_URI}/fault/v1/network-packet-loss/stop and /status

{ECS_AGENT_URI}/fault/v1/network-packet-loss/stop 엔드포인트가 오류를 중지하고 {ECS_AGENT_URI}/fault/v1/network-packet-loss/status가 오류 상태를 확인합니다. 한 번에 각 유형의 오류 중 하나만 지원됩니다.

다음은 /stop 및 /status 엔드포인트 사용에 대한 두 가지 예제 요청입니다. 둘 다 POST HTTP 메서드를 사용합니다.

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-packet-loss/stop
```

```
Endpoint: ${{ECS_AGENT_URI}/fault/v1/network-packet-loss/status
```

Amazon ECS 서비스 파라미터 업데이트

서비스를 생성한 후 태스크 수 등의 서비스 파라미터를 업데이트해야 하는 경우가 있습니다.

서비스 스케줄러는 새 태스크를 시작할 때 다음 로직을 사용하여 클러스터에 태스크 배치를 결정합니다.

- 클러스터에서 어떤 컨테이너 인스턴스가 서비스의 태스크 정의를 지원할 수 있는지 결정합니다. 예를 들어, 필요한 CPU, 메모리, 포트 및 컨테이너 인스턴스 속성이 있습니다.
- 기본적으로 서비스 스케줄러는 다른 배치 전략을 선택할 수 있더라도 이러한 방식으로 가용 영역 전체에 태스크 분배를 시도합니다.
  - 인스턴스와 동일한 가용 영역에서 이 서비스에 대해 실행 중인 태스크 수가 가장 적은 순서로 유효한 컨테이너 인스턴스를 정렬합니다. 예를 들어 영역 A에는 실행 중인 서비스 작업이 1개이고, 영역 B 및 C에는 0개일 경우 영역 B 또는 C가 최적 배치로 간주됩니다.
  - 새 서비스 태스크를 최적 가용 영역(이전 단계에서 결정됨)의 유효한 컨테이너 인스턴스에 배치하되, 이 서비스의 실행 작업 수가 가장 작은 컨테이너 인스턴스를 우선으로 합니다.

서비스 스케줄러는 실행 태스크를 중지할 때 다음 논리를 사용하여 클러스터의 가용 영역 간에 밸런싱을 유지하려고 합니다.

- 인스턴스와 동일한 가용 영역에서 이 서비스에 대해 실행 중인 태스크 수가 가장 많은 순서로 컨테이너 인스턴스를 정렬합니다. 예를 들어 영역 A에는 실행 중인 서비스 작업이 1개이고, 영역 B 및 C에는 2개일 경우 영역 B 또는 C의 컨테이너 인스턴스가 최적 종료로 간주됩니다.
- 최적 가용 영역(이전 단계에서 결정됨)의 컨테이너 인스턴스에서 태스크를 중지하되, 이 서비스의 실행 작업 수가 가장 큰 컨테이너 인스턴스를 우선으로 합니다.

목록을 사용하여 서비스 파라미터를 변경할 수 있는지 확인합니다.

가용 영역 재조정

서비스에 가용 영역 재분배를 사용할지 여부를 나타냅니다.

롤링 배포에 대해 이 파라미터를 변경할 수 있습니다.

용량 제공자 전략

용량 공급자 전략의 세부 정보입니다. 클러스터를 생성하거나, 태스크를 실행하거나, 서비스를 업데이트할 때 용량 제공자를 설정할 수 있습니다.

Fargate를 사용하는 경우 용량 제공자는 `FARGATE` 또는 `FARGATE_SPOT`입니다.

Amazon EC2를 사용하는 경우 용량 제공자는 Auto Scaling 그룹입니다.

롤링 배포와 블루/그린 배포에 대해 용량 제공자를 변경할 수 있습니다.

다음 목록은 유효한 전환을 제공합니다.

- Fargate 시작 유형을 Auto Scaling 그룹 용량 제공자로 업데이트합니다.

- Amazon EC2 시작 유형을 Fargate 용량 제공자로 업데이트합니다.
- Fargate 용량 제공자를 Auto Scaling 그룹 용량 제공자로 업데이트합니다.
- Amazon EC2 용량 제공자를 Fargate 용량 제공자로 업데이트합니다.
- Auto Scaling 그룹 또는 Fargate 용량 제공자를 시작 유형으로 다시 업데이트합니다. CLI나 API를 사용할 때 `capacityProviderStrategy` 파라미터에 빈 목록을 전달합니다.

클러스터

클러스터 이름은 변경할 수 없습니다.

배포 구성

배포 구성에는 CloudWatch 경보, 장애 감지에 사용되는 회로 차단기 및 필요한 구성이 포함됩니다.

배포 회로 차단기는 서비스가 정상 상태에 도달할 수 없는 경우 서비스 배포가 실패할지 여부를 결정합니다. 배포 회로 차단기를 사용하는 경우 서비스 배포가 실패 상태로 전환되고 새 태스크 시작이 중지됩니다. 롤백 옵션을 사용하는 경우 서비스 배포가 실패하면 성공적으로 완료된 마지막 배포로 서비스가 롤백됩니다.

Amazon ECS 회로 차단기가 사용되는 서비스를 업데이트할 때 Amazon ECS에서는 서비스 배포 및 서비스 개정이 생성됩니다. 이러한 리소스를 통해 서비스 기록에 대한 자세한 내용을 볼 수 있습니다. 자세한 내용은 Amazon ECS 서비스 배포를 사용하여 서비스 기록 보기 섹션을 참조하세요.

서비스 스케줄러는 (서비스에 대한 배포 구성에서) 최소 정상 상태 백분율 및 최대 백분율 파라미터를 사용하여 배포 전략을 결정합니다.

서비스에서 롤링 업데이트(ECS) 배포 유형을 사용하는 경우 최소 정상 백분율은 배포 중에 `RUNNING` 상태를 유지해야 하는 서비스의 태스크 수에 대한 하한을 나타내며, 원하는 태스크 수(가장 가까운 정수로 반올림됨)의 백분율로 표시됩니다. EC2 시작 유형을 사용하는 작업이 서비스에 포함된 경우, 이 파라미터는 컨테이너 인스턴스가 `DRAINING` 상태에 있는 동안에도 적용됩니다. 추가 클러스터 용량을 사용하지 않고 배포하려면 이 파라미터를 사용합니다. 예를 들어 서비스에서 원하는 작업 수가 4개이고 최소 정상 상태 백분율이 50%일 경우 스케줄러가 새 작업 2개를 시작하기 전에 기존 작업 2개를 중지하여 클러스터 용량을 확보할 수 있습니다. 로드 밸런서를 사용하지 않는 서비스의 태스크는 `RUNNING` 상태일 경우에 정상으로 간주됩니다. 로드 밸런서를 사용하는 서비스의 태스크는 `RUNNING` 상태이고 로드 밸런서에 의해 정상으로 보고되는 경우에 정상으로 간주됩니다. 최소 정상 상태 백분율의 기본값은 100%입니다.

서비스에서 롤링 업데이트(ECS) 배포 유형을 사용하는 경우 최대 백분율 파라미터는 배포 중에 `PENDING`, `RUNNING` 또는 `STOPPING` 상태로 허용되는 서비스의 태스크 수에 대한 상한을 나타내며, 원하는 태스크 수(가장 가까운 정수로 반내림)의 백분율로 표시됩니다. EC2 시작 유형을 사용하는 작업이 서비스에 포함된 경우, 이 파라미터는 컨테이너 인스턴스가 `DRAINING` 상태에 있는 동안

에도 적용됩니다. 이 파라미터를 사용하여 배포 배치 크기를 정의합니다. 예를 들면, 서비스에서 원하는 작업의 수가 4개이고 최대 백분율 값이 200%인 경우, 스케줄러가 기존 작업 4개를 중지하기 전에 새 작업 4개를 시작할 수 있습니다. 단, 이렇게 하는 데 필요한 클러스터 리소스를 사용할 수 있는 경우를 전제로 합니다. 최대 백분율의 기본값은 200%입니다.

서비스 스케줄러가 업데이트 도중 태스크를 교체할 때 서비스는 먼저 로드 밸런서(사용될 경우)에서 태스크를 제거하고 연결이 드레이닝될 때까지 대기합니다. 그런 다음, 태스크를 실행하는 컨테이너에 docker stop과 동등한 명령이 전송됩니다. 그 결과 SIGTERM 신호 및 30초 제한 시간이 발생하고, 이 제한 시간이 경과되면 SIGKILL이 전송되어 컨테이너가 강제로 중지됩니다. 컨테이너가 SIGTERM 신호를 정상적으로 처리하여 신호 수신 후 30초 이내에 종료할 경우 SIGKILL 신호가 전송되지 않습니다. 서비스 스케줄러는 최소 정상 상태 백분율 및 최대 백분율 파라미터 설정에 정의된 대로 태스크를 시작 및 중지합니다.

또한 서비스 스케줄러는 컨테이너 상태 확인 또는 로드 밸런서 대상 그룹 상태 확인이 실패한 후 비정상으로 확인된 작업을 대체합니다. 이러한 대체는 maximumPercent 및 desiredCount 서비스 정의 파라미터에 따라 달라집니다. 작업이 비정상으로 표시되면 서비스 스케줄러는 먼저 대체 작업을 시작합니다. 그러면 다음과 같이 진행됩니다.

- 대체 작업의 상태가 HEALTHY인 경우 서비스 스케줄러는 비정상 작업을 중지합니다.
- 대체 작업의 상태가 UNHEALTHY이면, 스케줄러는 비정상 대체 작업 또는 기존의 비정상 작업을 중지하여 총 작업 수가 desiredCount와 같아지도록 합니다.

maximumPercent 파라미터로 인해 스케줄러가 대체 작업을 먼저 시작하는 것이 제한되는 경우, 스케줄러는 비정상 작업을 한 번에 하나씩 임의로 중지하여 용량을 확보한 다음 대체 작업을 시작합니다. 비정상 작업이 모두 정상 작업으로 대체될 때까지 시작 및 중지 프로세스가 계속됩니다. 비정상 작업이 모두 대체되고 정상 작업만 실행 중인 경우 총 작업 수가 desiredCount를 초과하면 총 작업 수가 desiredCount와 같아질 때까지 정상 작업을 무작위로 중지합니다. maximumPercent 및 desiredCount에 대한 자세한 정보는 Service definition parameters를 참조하세요.

배포 컨트롤러

서비스에 사용할 배포 컨트롤러입니다. 사용 가능한 배포 컨트롤러 유형은 세 가지입니다.

- ECS
- EXTERNAL
- CODE_DEPLOY

서비스를 업데이트할 때 사용하는 배포 컨트롤러를 업데이트할 수 있습니다. 다음 목록은 유효한 전환을 제공합니다.

- CodeDeploy 블루/그린 배포(CODE_DEPLOY)에서 ECS 롤링 또는 블루/그린 배포(ECS)로 업데이트합니다.
- CodeDeploy 블루/그린 배포(CODE_DEPLOY)에서 외부 배포(EXTERNAL)로 업데이트합니다.
- ECS 롤링 또는 블루/그린 배포(ECS)에서 외부 배포(EXTERNAL)로 업데이트합니다.
- 외부 배포(EXTERNAL)에서 ECS 롤링 또는 블루/그린 배포(ECS)로 업데이트합니다.

서비스의 배포 컨트롤러를 업데이트하는 경우 다음 사항을 고려하세요.

- VPC Lattice 또는 Amazon ECS Service Connect를 사용하는 경우 서비스의 배포 컨트롤러를 ECS 배포 컨트롤러에서 다른 컨트롤러로 업데이트할 수 없습니다.
- 진행 중인 서비스 배포 중에는 서비스의 배포 컨트롤러를 업데이트할 수 없습니다.
- 서비스에 로드 밸런서가 없는 경우 서비스의 배포 컨트롤러를 CODE_DEPLOY로 업데이트할 수 없습니다.
- deploymentConfiguration에 경보, 배포 회로 차단기 또는 BLUE_GREEN 배포 전략이 포함된 경우 서비스의 배포 컨트롤러를 ECS에서 다른 컨트롤러로 업데이트할 수 없습니다. 자세한 내용은 Amazon ECS 서비스 배포 컨트롤러 및 전략 섹션을 참조하세요.
- 서비스의 배포 컨트롤러를 ECS에서 다른 컨트롤러로 업데이트하는 경우 컨테이너 정의에서 versionConsistency에 대해 지정하는 값은 Amazon ECS에서 사용되지 않습니다.
- 서비스의 배포 컨트롤러를 ECS에서 다른 컨트롤러로 업데이트하는 경우에도 UpdateService 및 DescribeService API 응답은 taskSets 대신 여전히 deployments를 반환합니다. UpdateService 및 CreateService에 대한 자세한 내용은 Amazon ECS API 참조의 UpdateService 및 CreateService를 참조하세요.
- 서비스가 롤링 업데이트 배포 전략을 사용하는 경우 배포 컨트롤러를 ECS에서 다른 컨트롤러로 업데이트하면 deploymentConfiguration의 maximumPercent 값이 사용되는 방식이 변경됩니다. maximumPercent는 롤링 업데이트 배포에서 총 태스크 수 한도로만 사용되는 대신 비정상 태스크를 대체하는 데 사용됩니다. 스케줄러가 비정상 태스크를 바꾸는 방법에 대한 자세한 내용은 Amazon ECS 서비스 섹션을 참조하세요.
- 서비스의 배포 컨트롤러를 ECS에서 다른 배포 컨트롤러로 업데이트하면 로드 밸런서 구성으로 지정한 모든 advancedConfiguration이 무시됩니다. 자세한 내용은 Amazon ECS API 참조의 LoadBalancer 및 AdvancedConfiguration을 참조하세요.

CloudFormation을 사용하여 서비스의 배포 컨트롤러를 업데이트하는 경우 수행 중인 마이그레이션 유형에 따라 다음을 고려하세요.

- EXTERNAL 배포 컨트롤러 정보와 TaskSet 및 PrimaryTaskSet 리소스가 포함된 CloudFormation 템플릿이 있고 EXTERNAL에서 ECS로 업데이트할 때 템플릿에서 태스크 세트 리소스를 제거하는 경우 배포 컨트롤러가 ECS로 업데이트된 후 DescribeTaskSet 및

DeleteTaskSet API 직접 호출에서 400 오류가 반환됩니다. 이로 인해 CloudFormation 스택이 UPDATE_COMPLETE 상태로 전환되더라도 태스크 세트 리소스에서 CloudFormation 삭제에 실패합니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 리소스가 스택에서 제거되었지만 삭제되지 않음을 참조하세요. 이 문제를 해결하려면 Amazon ECS DeleteTaskSet API를 사용하여 태스크 세트를 직접 삭제합니다. 태스크 세트를 삭제하는 방법에 대한 자세한 내용은 Amazon Elastic Container Service API 참조의 DeleteTaskSet를 참조하세요.

- 새 태스크 정의를 사용하여 CODE_DEPLOY에서 ECS로 마이그레이션하고 CloudFormation에서 롤백 작업을 수행하는 경우 Amazon ECS UpdateService 요청에 실패하고 다음 오류가 발생합니다.

```
Resource handler returned message: "Invalid request provided: Unable to update
task definition on services with a CODE_DEPLOY deployment controller. Use AWS
CodeDeploy to trigger a new deployment. (Service: Ecs, Status Code: 400,
Request ID: 0abda1e2-f7b3-4e96-b6e9-c8bc585181ac) (SDK Attempt Count: 1)"
(RequestToken: ba8767eb-c99e-efed-6ec8-25011d9473f0, HandlerErrorCode:
 InvalidRequest)
```

- ECS에서 EXTERNAL 배포 컨트롤러로 성공적으로 마이그레이션한 후에는 Amazon ECS가 더 이상 배포를 관리하지 않으므로 ACTIVE 태스크 세트를 수동으로 제거해야 합니다. 태스크 세트를 삭제하는 방법에 대한 자세한 내용은 Amazon Elastic Container Service API 참조의 DeleteTaskSet를 참조하세요.

원하는 태스크 개수

서비스에서 배치하고 계속 실행할 태스크의 인스턴스화 횟수입니다.

서비스를 일시적으로 중지하려면 값을 0으로 설정합니다. 그런 다음, 서비스를 시작할 준비가 되면 원래 값으로 서비스를 업데이트합니다.

롤링 배포와 블루/그린 배포에 대해 이 파라미터를 변경할 수 있습니다.

관리형 태그 활성화

서비스의 태스크에 대해 Amazon ECS 관리형 태그를 켤지 여부를 결정합니다.

업데이트 후 시작된 태스크에만 업데이트가 반영됩니다. 모든 태스크에서 태그를 업데이트하려면 강제 배포 옵션을 사용합니다.

롤링 배포와 블루/그린 배포에 대해 이 파라미터를 변경할 수 있습니다.

ECS Exec 활성화

Amazon ECS Exec 사용 여부를 결정합니다.

서비스가 생성될 때 설정된 값을 재정의하지 않으려면 이 작업을 수행할 때 이를 null로 설정할 수 있습니다.

롤링 배포에 대해 이 파라미터를 변경할 수 있습니다.

상태 확인 유예 기간

이 기간(초) 동안 Amazon ECS 서비스 스케줄러는 태스크가 처음 시작된 후 비정상 Elastic Load Balancing, VPC Lattice, 컨테이너 상태 확인을 무시합니다. 상태 확인 유예 기간 값을 지정하지 않으면 기본값인 0이 사용됩니다. 상태 확인을 사용하지 않으면 `healthCheckGracePeriodSeconds`는 사용되지 않습니다.

서비스의 태스크가 시작하고 상태 확인에 응답하는 데 시간이 다소 걸린다면 최대 2,147,483,647초(약 69년)의 상태 확인 유예 기간을 지정할 수 있습니다. 이 기간에는 Amazon ECS 서비스 스케줄러가 상태 확인의 상태를 무시합니다. 이 유예 기간은 서비스 스케줄러가 태스크를 비정상으로 표시하여 작업 처리 전에 이를 중단시키는 일이 없도록 해줍니다.

롤링 배포와 블루/그린 배포에 대해 이 파라미터를 변경할 수 있습니다.

로드 밸런서

로드 밸런서를 업데이트할 때 서비스 연결 역할을 사용해야 합니다.

Elastic Load Balancing 로드 밸런서 객체의 목록입니다. 여기에는 로드 밸런서 이름, 컨테이너 이름 및 로드 밸런서에서 액세스할 컨테이너 포트가 포함됩니다. 컨테이너 이름은 컨테이너 정의에 표시되는 이름과 같습니다.

Amazon ECS는 Elastic Load Balancing 로드 밸런서 또는 Amazon ECS 컨테이너 인스턴스에 연결된 보안 그룹을 자동으로 업데이트하지 않습니다.

로드 밸런서 구성을 추가, 업데이트 또는 제거하면 Amazon ECS가 업데이트된 Elastic Load Balancing 구성으로 새 태스크를 시작한 다음 새 태스크가 실행 중일 때 이전 태스크를 중지합니다.

롤링 업데이트를 사용하는 서비스의 경우 Elastic Load Balancing 대상 그룹을 추가, 업데이트 또는 제거할 수 있습니다. 단일 대상 그룹에서 여러 대상 그룹으로 업데이트하고, 여러 대상 그룹에서 단일 대상 그룹으로 업데이트할 수 있습니다.

블루/그린 배포를 사용하는 서비스의 경우 CodeDeploy를 통해 `CreateDeployment`를 사용하여 Elastic Load Balancing 대상 그룹을 업데이트할 수 있습니다. 블루/그린 배포에는 여러 대상 그룹이 지원되지 않는다는 점에 유의하세요. 자세한 내용은 Amazon ECS 서비스에 여러 대상 그룹 등록을 참조하세요.

외부 배포 컨트롤러를 사용하는 서비스의 경우 CreateTaskSet를 사용하여 로드 밸런서를 추가, 업데이트 또는 제거할 수 있습니다. 외부 배포에는 여러 대상 그룹이 지원되지 않는다는 점에 유의하세요. 자세한 내용은 Amazon ECS 서비스에 여러 대상 그룹 등록을 참조하세요.

로드 밸런서를 제거하려면 빈 목록을 전달합니다.

롤링 배포에 대해 이 파라미터를 변경할 수 있습니다.

네트워크 구성

서비스 네트워크 구성입니다.

롤링 배포에 대해 이 파라미터를 변경할 수 있습니다.

배치 제약 조건

사용할 서비스를 업데이트할 태스크 배치 제약 조건 객체의 배열입니다. 값을 지정하지 않으면 서비스에 대한 기존 배치 제약 조건이 변경되지 않습니다. 이 값을 지정하면 서비스에 대해 정의된 기존 배치 제약 조건이 재정의됩니다. 기존 배치 제약 조건을 모두 제거하려면 빈 배열을 지정합니다.

태스크당 최대 10개의 제약 조건을 지정할 수 있습니다. 이 제한에는 태스크 정의 내의 제약 조건과 런타임 시 지정되는 제약 조건이 포함됩니다.

롤링 배포와 블루/그린 배포에 대해 이 파라미터를 변경할 수 있습니다.

배치 전략

사용할 서비스를 업데이트할 태스크 배치 전략 객체입니다. 값을 지정하지 않으면 서비스에 대한 기존 배치 전략이 변경되지 않습니다. 이 값을 지정하면 서비스에 대해 정의된 기존 배치 전략이 재정의됩니다. 기존 배치 전략을 제거하려면 빈 객체를 지정합니다.

롤링 배포와 블루/그린 배포에 대해 이 파라미터를 변경할 수 있습니다.

플랫폼 버전

서비스가 실행되는 Fargate 플랫폼 버전입니다.

Linux 플랫폼 버전을 사용하는 서비스는 Windows 플랫폼 버전을 사용하도록 업데이트할 수 없으며 그 반대의 경우도 마찬가지입니다.

롤링 배포에 대해 이 파라미터를 변경할 수 있습니다.

태그 전파

태그를 태스크 정의 또는 서비스에서 태스크로 전파할지 여부를 결정합니다. 값을 지정하지 않으면 태그가 전파되지 않습니다.

업데이트 후 시작된 태스크에만 업데이트가 반영됩니다. 모든 태스크의 태그를 업데이트하려면 Amazon ECS가 업데이트된 태그를 사용하여 새 태스크를 시작하도록 `forceNewDeployment`를 `true`로 설정합니다.

롤링 배포와 블루/그린 배포에 대해 이 파라미터를 변경할 수 있습니다.

Service Connect 구성

Amazon ECS Service Connect에 대한 구성입니다. 이 파라미터는 서비스가 애플리케이션 내의 다른 서비스에 연결되는 방법을 결정합니다.

롤링 배포에 대해 이 파라미터를 변경할 수 있습니다.

서비스 레지스트리

서비스 레지스트리를 업데이트할 때 서비스 연결 역할을 사용해야 합니다.

이 서비스에 할당할 서비스 검색 레지스트리의 세부 정보입니다. 자세한 내용은 서비스 검색을 참조하세요.

서비스 레지스트리 구성을 추가, 업데이트 또는 제거하면 Amazon ECS기 업데이트된 서비스 레지스트리 구성으로 새 태스크를 시작한 다음 새 태스크가 실행 중일 때 이전 태스크를 중지합니다.

서비스 레지스트리를 제거하려면 빈 목록을 전달합니다.

롤링 배포에 대해 이 파라미터를 변경할 수 있습니다.

태스크 정의

서비스에 사용할 태스크 정의와 개정입니다.

태스크 정의에서 컨테이너가 사용하는 포트를 변경하는 경우 업데이트된 포트와 함께 작동하도록 컨테이너 인스턴스에 대한 보안 그룹을 업데이트해야 합니다.

서비스에 대한 작업 정의를 업데이트하는 경우, 로드 밸런서 구성에서 지정한 컨테이너 이름 및 컨테이너 포트는 작업 정의에서 그대로 유지해야 합니다.

컨테이너 이미지 가져오기 동작은 시작 유형에 따라 다릅니다. 자세한 내용은 다음 중 하나를 참조하십시오.

- Amazon ECS의 Fargate 시작 유형
- Amazon ECS의 EC2 시작 유형
- Amazon ECS의 외부(Amazon ECS Anywhere) 시작 유형

롤링 배포에 대해 이 파라미터를 변경할 수 있습니다.

볼륨 구성

`configuredAtLaunch`였던 볼륨의 세부 정보입니다. 태스크 정의에서 `configuredAtLaunch`를 `true`로 설정하면 이 서비스 파라미터는 서비스의 각 서비스에 대해 배포 중에 생성 및 연결할 하나의 Amazon EBS 볼륨을 구성합니다. ServiceManagedEBSVolumeConfiguration에서 크기, volumeType, IOPS, 처리량, 스냅샷 및 암호화를 구성할 수 있습니다. 볼륨의 `name`은 태스크 정의의 `name`과 일치해야 합니다. null로 설정하면 새 배포가 트리거되지 않습니다. 그렇지 않으면 이 구성이 기존 구성과 다를 경우 새 배포가 트리거됩니다.

롤링 배포에 대해 이 파라미터를 변경할 수 있습니다.

VPC Lattice 구성

서비스에 대한 VPC Lattice 구성입니다. 이는 서비스 간 통신을 위해 서비스가 VPC Lattice와 통합되는 방법을 정의합니다.

롤링 배포에 대해 이 파라미터를 변경할 수 있습니다.

AWS CDK 고려 사항

AWS CDK는 리소스 상태를 추적하지 않습니다. 서비스를 생성하는지, 업데이트하는지는 알 수 없습니다. 고객은 이스케이프 해치를 사용하여 ecs `Service` L1 구문에 직접 액세스해야 합니다.

이스케이프 해치에 대한 자세한 내용은 AWS 클라우드 개발 키트 (AWS CDK) v2 Developer Guide의 Customize constructs from the AWS Construct Library를 참조하세요.

기존 서비스를 `ecs.Service` 구문으로 마이그레이션하려면 다음을 수행하세요.

1. 이스케이프 해치를 사용하여 `Service` L1 구문에 액세스합니다.
2. `Service` L1 구문에서 다음 속성을 수동으로 설정합니다.

   서비스에서 Amazon EC2 용량을 사용하는 경우

   - `daemon?`
   - `placementConstraints?`
   - `placementStrategies?`
   - `awsvpc` 네트워크 모드를 사용하는 경우 `vpcSubnets?` 및 `securityGroups?` 구문을 설정해야 합니다.

서비스에서 Fargate를 사용하는 경우

- `FargatePlatformVersion`
- `vpcSubnets?` 및 `securityGroups?` 구문입니다.

3. 다음과 같이 `launchType`을 설정합니다.

```
const cfnEcsService = service.node.findChild('Service') as ecs.CfnService;
cfnEcsService.launchType = "FARGATE";
```

시작 유형에서 용량 제공자로 마이그레이션하려면 다음을 수행하세요.

1. 이스케이프 해치를 사용하여 `Service` L1 구문에 액세스합니다.
2. `capacityProviderStrategies?` 구문을 추가합니다.
3. 서비스를 배포합니다.

Amazon ECS 서비스 업데이트

서비스를 생성한 후 태스크 수 등의 서비스 파라미터를 업데이트해야 하는 경우가 있습니다.

Amazon ECS 회로 차단기가 사용되는 서비스를 업데이트할 때 Amazon ECS에서는 서비스 배포 및 서비스 개정이 생성됩니다. 이러한 리소스를 통해 서비스 기록에 대한 자세한 내용을 볼 수 있습니다. 자세한 내용은 Amazon ECS 서비스 배포를 사용하여 서비스 기록 보기 섹션을 참조하세요.

사전 조건

서비스를 업데이트하기 전에 배포 유형에 대해 변경할 수 있는 서비스 파라미터를 확인합니다. 변경 가능한 파라미터의 전체 목록은 Amazon ECS 서비스 파라미터 업데이트 섹션을 참조하세요.

절차

Console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
3. 클러스터 세부 정보 페이지의 서비스 섹션에서 서비스 옆의 확인란을 선택하고 업데이트를 선택합니다.

4. 서비스에서 새 배포를 시작하도록 하려면 Force new deployment(새 배포 적용)를 선택합니다.
5. 작업 정의에서 사용할 작업 정의 패밀리 및 개정을 선택합니다.

> **Important**
>
> 콘솔은 선택한 작업 정의 패밀리 및 개정이 정의된 컴퓨팅 구성과 호환되는지 검증합니다. 경고 메시지가 나타나면 작업 정의 호환성과 컴퓨팅 구성이 모두 선택되어 있는지 확인합니다.

6. 복제본(Replica)을 선택하는 경우, 원하는 태스크(Desired tasks)에 서비스에서 시작하고 유지할 태스크 개수를 입력합니다.
7. 복제본을 선택한 경우 Amazon ECS에서 가용 영역 전체의 태스크 분산이 모니터링되고 불균형이 있으면 재분산되도록 하려면 가용 영역 서비스 리밸런싱에서 가용 영역 서비스 리밸런싱을 선택합니다.
8. 최소 실행 작업(Min running tasks)의 경우, 배포 시 `RUNNING` 상태를 유지해야 하는 서비스 내 작업 수에 대한 하한을 원하는 작업 수의 백분율(가장 가까운 정수로 올림)로 입력합니다. 자세한 내용은 배포 구성을 참조하세요.
9. 최대 실행 작업(Max running tasks)의 경우, 배포 시 `RUNNING` 또는 `PENDING` 상태가 허용되는 서비스 내 작업 수에 대한 상한을 원하는 작업 수의 백분율(가장 가까운 정수로 내림)로 입력합니다.
10. 서비스에 대한 태스크 배포 방법을 구성하려면 배포 옵션을 확장한 후 옵션을 구성하세요.

    a. 배포 컨트롤러 유형에서 서비스 배포 컨트롤러를 지정하세요. Amazon ECS 콘솔은 `ECS`와 같은 컨트롤러 유형을 지원합니다.

    b. 배포 전략에서 Amazon ECS가 새 버전의 서비스를 배포하는 데 사용하는 전략을 선택하세요.

    c. 배포 전략의 선택 항목에 따라 다음을 수행하세요.

배포 전략	단계	
롤링 업데이트	a. 최소 실행 작업 비율(%)에서 서비스 배포 중에 실행해야 하는 태스크의 최소 백분율 값을 지정하세요. 자세한 내용은 태스크를 대체하여	

배포 전략	단계	
	Amazon ECS 서비스 배포 섹션을 참조하세요. b. 최대 실행 작업 비율(%)에서 서비스 배포 중에 실행해야 하는 태스크의 최대 백분율 값을 지정하세요. 자세한 내용은 태스크를 대체하여 Amazon ECS 서비스 배포 섹션을 참조하세요.	
블루/그린	베이크 소요 시간에서 블루 및 그린 서비스 개정이 동시에 실행되어야 하는 기간(분)을 지정하세요. 자세한 내용은 Amazon ECS 블루/그린 배포 섹션을 참조하세요.	

d. 수명 주기 단계에서 Lambda 함수를 실행하려면 배포 수명 주기 후크 아래 각 고유한 Lambda 함수에 대해 다음을 수행하세요.

i. 추가를 선택합니다.

실행하려는 모든 고유 함수에 대해 반복하세요.

ii. Lambda 함수에서 함수 이름을 입력하세요.

iii. 역할에서 블루/그린 권한이 있는 사전 요구 사항에서 생성한 역할을 선택하세요.

자세한 내용은 Amazon ECS 블루/그린 배포에서 Lambda 함수에 필요한 권한 섹션을 참조하세요.

iv. 수명 주기 단계에서 Lambda 함수가 실행되는 단계를 선택하세요.

v. (선택 사항) 후크 세부 정보에 후크에 대한 정보를 제공하는 키 값 페어를 입력하세요.

11. Amazon ECS에서 배포 오류를 탐지 및 처리하는 방법을 구성하려면 배포 오류 탐지(Deployment failure detection)를 펼친 다음, 옵션을 선택합니다.

a. 작업을 시작할 수 없을 때 배포를 중지하려면 Use the Amazon ECS deployment circuit breaker(Amazon ECS 배포 회로 차단기 사용)를 선택합니다.

   배포 회로 차단기가 배포를 실패한 상태로 설정했을 때 소프트웨어가 마지막으로 완료한 배포로 자동 롤백하도록 하려면 실패 시 롤백을 선택합니다.

b. 애플리케이션 지표를 기반으로 배포를 중지하려면 CloudWatch 경보 사용을 선택합니다. 그런 다음, CloudWatch 경보 이름에서 경보를 선택합니다. 새 경보를 생성하려면 CloudWatch 콘솔을 이동합니다.

   CloudWatch 경보가 배포를 실패한 상태로 설정할 때 소프트웨어가 마지막으로 완료한 배포 상태로 자동 롤백하도록 하려면 실패 시 롤백을 선택합니다.

12. 컴퓨팅 옵션을 변경하려면 컴퓨팅 구성을 확장하고 다음을 수행합니다.

    a. AWS Fargate에 대한 서비스에서 플랫폼 버전(Platform version)에 새로운 버전을 선택합니다.

    b. 용량 공급자 전략을 사용하는 서비스의 경우 용량 공급자 전략에서 다음을 수행합니다.

       - 용량 공급자를 추가하려면 더 추가를 선택합니다. 그런 다음 용량 공급자에서 용량 공급자를 선택합니다.
       - 용량 공급자를 제거하려면 용량 공급자 오른쪽의 제거를 선택합니다.

       Auto Scaling 그룹 공급자를 사용하는 서비스는 Fargate 용량 공급자를 사용하도록 업데이트할 수 없습니다. Fargate 용량 공급자를 사용하는 서비스는 Auto Scaling 그룹 공급자를 사용하도록 업데이트할 수 없습니다.

13. (선택 사항) 서비스 오토 스케일링을 구성하려면 서비스 오토 스케일링을 확장한 후에 다음 파라미터를 지정합니다.트래픽 흐름에서 과거 로드 데이터를 살펴보는 예측 오토 스케일링을 사용하려면 서비스를 생성한 다음 예측 오토 스케일링을 구성합니다. 자세한 내용은 과거 패턴을 사용하여 예측 규모 조정으로 Amazon ECS 서비스 규모 조정 섹션을 참조하세요.

    a. 서비스 Auto Scaling을 사용하려면 서비스 Auto Scaling(Service auto scaling)을 선택합니다.

    b. 작업의 최소 개수에 서비스 Auto Scaling에서 사용할 작업 수의 하한을 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.

    c. 작업의 최대 개수에 서비스 Auto Scaling에서 사용할 작업 수의 상한을 입력합니다. 바람직한 수는 이 숫자 이내여야 합니다.

    d. 정책 유형을 선택합니다. 규모 조정 정책 유형에서 다음 옵션 중 하나를 선택합니다.

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
대상 추적	a. 조정 정책 유형(Scaling policy type)에서 대상 추적(Target tracking)을 선택합니다. b. 정책 이름(Policy name)에 정책 이름을 입력합니다. c. ECS 서비스 지표에서 다음 지표 중 하나를 선택합니다. • ECSServiceAverageCPUUtilization - 서비스의 평균 CPU 사용률입니다. • ECSServiceAverageMemoryUtilization - 서비스의 평균 메모리 사용률입니다. • ALBRequestCountPerTarget - Application Load Balancer 대상 그룹에서 대상별로 완료된 요청 수입니다. d. 대상 값(Target value)에 서비스가 선택된 지표에 유지하는 값을 입력합니다. e. 스케일 아웃 휴지 기간에 스케일 아웃 활동	

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
	(작업 추가) 이후 다른 스케일 아웃 활동이 시작되기 전 경과해야 하는 시간(초)을 입력합니다. f. 스케일 인 휴지 기간에 스케일 인 활동(작업 제거) 이후 다른 스케일 인 활동이 시작되기 전 경과해야 하는 시간(초)을 입력합니다. g. 정책에서 스케일 인 활동을 수행하지 않도록 방지하려면 스케일 인 비활성화(Turn off scale-in)를 선택합니다. h. • (선택 사항) 늘어난 트래픽에 맞춰 조정 정책을 스케일 아웃하려고 하지만 트래픽이 감소해도 스케일 인할 필요가 없게 하려는 경우 스케일 인 끄기를 선택합니다.	

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
단계 조정	a. 조정 정책 유형(Scaling policy type)에서 단계 조정(Step scaling)을 선택합니다. b. 정책 이름에 정책 이름을 입력합니다. c. [Alarm name]에 경보의 고유한 이름을 입력합니다. d. Amazon ECS 서비스 지표에서 경보에 사용할 서비스 지표를 선택합니다. e. 통계에서 경보 통계를 선택합니다. f. 기간에서 경보의 기간을 선택합니다. g. 경보 조건에서 선택한 지표를 정의된 임계값과 비교하는 방법을 선택합니다. h. 지표 비교 임계값 및 경보 시작을 위한 평가 기간에 경보에 사용되는 임계값과 임계값을 평가할 기간을 입력합니다. i. 조정 작업에서 다음을 수행합니다. • 작업에서 서비스에 대해 작업을 특정 원하는 수를 추가, 제거 또	

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
	는 설정할지 여부를 선택합니다. • 작업을 추가하거나 제거하려는 경우 값에 조정 작업이 시작될 때 추가 또는 제거할 작업 수(또는 기존 작업에 대한 비율)를 입력합니다. 원하는 수 설정을 선택한 경우 작업 수를 입력합니다. 유형에서 값이 정수인지, 아니면 기존의 원하는 수에 대한 백분율 값인지 선택합니다. • 하한 및 상한에 단계 조정의 하한과 상한을 입력합니다. 기본적으로 추가 정책의 하한은 경보 임계값이고 상한은 양(+)의 무한대입니다. 기본적으로 제거 조정의 상한은 경보 임계값이고 하한은 음(-)의 무한대입니다. • (선택 사항) 조정 옵션을 추가합니다. 새 스케일링 액션 추가를 선택하고 조정 작업 단계를 반복합니다.	

이 정책 유형을 사용하려면 다음을 수행하세요.	조치	
	• 휴지 기간에 이전 조정 활동이 적용될 때까지 대기하는 시간(초)을 입력합니다. 추가 정책의 경우 이 시간은 스케일 아웃 활동 이후 조정 정책이 스케일 인 활동을 차단하고 한 번에 스케일 아웃할 수 있는 작업 수를 제한하는 시간입니다. 제거 정책의 경우 스케일 인 활동 이후 다른 스케일 인 활동을 시작하기 전까지 경과해야 하는 시간입니다.	

14. (선택 사항) Service Connect를 사용하려면 Turn on Service Connect(Service Connect 켜기)를 선택하고 다음을 지정합니다.

    a. Service Connect configuration(Service Connect 구성)에서 클라이언트 모드를 지정합니다.

        • 서비스에서 네임스페이스의 다른 서비스에만 연결하면 되는 네트워크 클라이언트 애플리케이션을 실행하는 경우 클라이언트 측만을 선택합니다.

        • 서비스가 네트워크 또는 웹 서비스 애플리케이션을 실행하고 이 서비스에 대한 엔드포인트를 제공해야 하며, 네임스페이스의 다른 서비스에 연결해야 하는 경우 Client and server(클라이언트 및 서버)를 선택합니다.

    b. 기본 클러스터 네임스페이스가 아닌 네임스페이스를 사용하려면 Namespace(네임스페이스)에서 서비스 네임스페이스를 선택합니다.

15. 작업에서 배포 시 구성과 호환되는 데이터 볼륨을 사용하는 경우 볼륨을 확장하여 볼륨을 구성할 수 있습니다.

볼륨 이름과 볼륨 유형은 작업 정의 개정을 생성할 때 구성되며 서비스를 업데이트할 때는 변경할 수 없습니다. 볼륨 이름과 유형을 업데이트하려면 새 작업 정의 개정을 생성하고 새 개정을 사용해 서비스를 업데이트해야 합니다.

이 볼륨 유형을 구성하는 방법	조치	
Amazon EBS	a. EBS 볼륨 유형에서 작업에 연결할 EBS 볼륨 유형을 선택합니다. b. 크기(GiB)에 유효한 볼륨 크기 값을 기비바이트(GiB) 단위로 입력합니다. 최소 1GiB 및 최대 16,384GiB 볼륨 크기를 지정할 수 있습니다. 스냅샷 ID를 제공하지 않는 한, 이 값은 필수입니다. c. IOPS의 경우 볼륨이 제공해야 하는 최대 초당 입출력 작업(IOPS) 수를 입력합니다. 이 값은 `io1`, `io2`, `gp3` 볼륨 유형에서만 구성할 수 있습니다. d. 처리량(MiB/s)에 볼륨에서 제공해야 하는 처리량을 초당 메비바이트 단위(MiBps 또는 MiB/s)로 입력합니다. 이 값은 `gp3` 볼륨 유형에서만 구성 가능합니다. e. 스냅샷 ID에서 기존 Amazon EBS 볼륨 스냅샷을 선택하거나 스냅샷에서 볼륨을 생성하려는 경우 스냅샷의 ARN을 입력합니다. 스냅샷 ID를 선택하거나 입력하지 않고	

이 볼륨 유형을 구성하는 방법	조치	
	비어 있는 새 볼륨을 생성할 수도 있습니다. f. 스냅샷 ID를 지정하는 경우 볼륨 초기화 속도(MiB/s)를 지정할 수 있습니다. 100~300MiB/s 사이의 값을 입력하면 볼륨 생성을 위해 스냅샷 ID를 사용하여 지정된 스냅샷에서 데이터가 로드되는 속도가 결정됩니다. g. 파일 시스템 유형에서 볼륨의 데이터 스토리지 및 검색에 사용할 파일 시스템 유형을 선택합니다. 운영 체제 기본값 또는 특정 파일 시스템 유형을 선택할 수 있습니다. Linux의 기본값은 XFS입니다. 스냅샷에서 생성된 볼륨의 경우 스냅샷을 생성할 때 볼륨에서 사용하던 것과 동일한 파일 시스템 유형을 지정해야 합니다. 파일 시스템 유형이 일치하지 않으면 작업이 시작되지 않습니다. h. 인프라 역할에서 Amazon ECS가 작업에 대한 Amazon EBS 볼륨을 관리할 수 있도록 하는 데 필요한 권한이 있는 IAM 역할을 선택	

이 볼륨 유형을 구성하는 방법	조치	
	합니다. `AmazonECSInfrastructureRolePolicyForVolumes` 관리형 정책을 역할에 연결하거나 정책을 지침으로 사용하여 특정 요구 사항에 맞는 권한을 보유한 자체 정책을 생성하고 연결할 수 있습니다. 필요한 권한에 대한 자세한 내용은 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요. i. 암호화에서 Amazon EBS 암호화를 기본 설정으로 사용하려면 기본값을 선택합니다. 계정에 암호화 기본 제공이 구성된 경우 설정에 지정된 AWS Key Management Service(AWS KMS) 키로 볼륨이 암호화됩니다. 기본값을 선택하고 Amazon EBS 기본 암호화가 켜져 있지 않으면 볼륨이 암호화되지 않습니다. 사용자 지정을 선택하면 볼륨 암호화에 대해 원하는 AWS KMS key를 지정할 수 있습니다.	

이 볼륨 유형을 구성하는 방법	조치	
	없음을 선택한 경우 암호화가 기본적으로 구성되지 않았거나 암호화된 스냅샷에서 볼륨을 생성하지 않으면 볼륨이 암호화되지 않습니다. j. 암호화에 사용자 지정을 선택한 경우 사용하려는 AWS KMS key를 지정해야 합니다. KMS 키에서 AWS KMS key를 선택하거나 키 ARN을 입력합니다. 대칭 고객 관리형 키를 사용하여 볼륨을 암호화하려는 경우 AWS KMS key 정책에 올바른 권한이 정의되었는지 확인합니다. 자세한 내용은 Data encryption for Amazon EBS volumes를 참조하세요. k. (선택 사항) 태그에서 작업 정의 또는 서비스의 태그를 전파하거나 자체 태그를 제공하여 Amazon EBS 볼륨에 태그를 추가할 수 있습니다. 작업 정의에서 태그를 전파하려면 태그 전파 시작에서 작업 정의를 선택합니다. 서비스에서 태그를 전파하려면 태그 전파	

이 볼륨 유형을 구성하는 방법	조치	
	시작에서 서비스를 선택합니다. 전파 안 함을 선택하거나 값을 선택하지 않으면 태그가 전파되지 않습니다. 고유한 태그를 제공하려면 태그 추가를 선택하고 추가하는 각 태그의 키와 값을 제공합니다. Amazon EBS 볼륨 태그 지정에 대한 자세한 내용은 Amazon EBS 볼륨 태그 지정을 참조하세요.	

16. (선택 사항) 서비스를 식별하려면 태그(Tags) 섹션을 펼친 다음, 태그를 구성합니다.
    - [태그 추가] 태그 추가를 선택하고 다음을 수행합니다.
      - 키에서 키 이름을 입력합니다.
      - 값에 키 값을 입력합니다.
    - [태그 제거] 태그 옆에 있는 태그 제거를 선택합니다.
17. 업데이트를 선택합니다.

AWS CLI

- `update-service`을(를) 실행합니다. 명령 실행에 대한 자세한 내용은 AWS Command Line Interface Reference의 update-service를 참조하세요.

  다음 `update-service` 예제에서는 `my-http-service` 서비스의 원하는 작업 수를 2로 업데이트합니다.

  모든 `### ##`을 사용자의 값으로 바꿉니다.

```
aws ecs update-service \
```

```
    --cluster MyCluster \
    --service my-http-service \
    --desired-count 2
```

다음 단계

배포를 추적하고 Amazon ECS 회로 차단기에 사용되는 서비스에 대한 서비스 기록을 봅니다. 자세한 내용은 Amazon ECS 서비스 배포를 사용하여 서비스 기록 보기 섹션을 참조하세요.

콘솔을 사용하여 Amazon ECS 서비스 삭제

다음은 서비스를 삭제하게 되는 사유 중 일부입니다.

- 더는 애플리케이션이 필요하지 않음
- 새 환경으로 서비스 마이그레이션 중
- 애플리케이션을 적극적으로 사용하지 않음
- 애플리케이션에서 리소스가 필요 이상으로 사용되고 있으며, 비용을 최적화하려고 함

삭제 전에 서비스가 자동으로 0으로 축소됩니다. 서비스와 관련된 로드 밸런서 리소스 또는 서비스 검색 리소스는 서비스 삭제의 영향을 받지 않습니다. Elastic Load Balancing 리소스를 삭제하려면, 로드 밸런서의 유형에 따라 Application Load Balancer 삭제(Delete an Application Load Balancer)나 Network Load Balancer 삭제(Delete a Network Load Balancer)를 참조하세요.

서비스를 삭제하면 Amazon ECS에서는 서비스에 대한 모든 서비스 배포와 서비스 개정이 삭제됩니다.

서비스를 삭제할 때 정리가 필요한 작업이 계속 실행 중인 경우, 서비스 상태가 `ACTIVE`에서 `DRAINING`으로 전환되고 서비스는 더 이상 콘솔 또는 `ListServices` API 작업에 표시되지 않습니다. 모든 작업이 `STOPPING` 또는 `STOPPED` 상태로 전환되면 서비스 상태가 `DRAINING`에서 `INACTIVE`로 전환됩니다. `DRAINING` 또는 `INACTIVE` 상태의 서비스는 `DescribeServices` API 작업을 사용하여 계속 볼 수 있습니다.

> **Important**
>
> `ACTIVE` 또는 `DRAINING` 상태의 기존 서비스와 동일한 이름으로 새 서비스를 생성하려고 하면 오류가 발생합니다.

절차

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 서비스에 대한 클러스터를 선택합니다.
3. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
4. 클러스터: **##**(Cluster : name) 페이지에서 서비스(Services) 탭을 선택합니다.
5. 서비스를 선택한 다음 삭제(Delete)를 선택합니다.
6. 0개의 태스크로 축소되지 않았더라도 서비스를 삭제하려면 강제 서비스 삭제(Force delete service)를 선택합니다.
7. 확인 프롬프트에서 삭제를 입력하고 삭제를 선택합니다.

Amazon ECS 단기 서비스 ARN을 긴 ARN으로 마이그레이션

Amazon ECS는 각 서비스에 고유한 Amazon 리소스 이름(ARN)을 할당합니다. 2021년 이전에 생성된 서비스에는 짧은 ARN 형식이 사용됩니다.

`arn:aws:ecs:region:aws_account_id:service/service-name`

클러스터 이름을 포함하도록 Amazon ECS의 ARN 형식이 변경되었습니다. 긴 ARN 형식은 다음과 같습니다.

`arn:aws:ecs:region:aws_account_id:service/cluster-name/service-name`

서비스에 태깅하려면 서비스에 긴 ARN 형식이 있어야 합니다.

서비스를 다시 생성할 필요 없이, 짧은 ARN 형식의 서비스를 긴 ARN 형식으로 마이그레이션할 수 있습니다. 이는 API, CLI 또는 콘솔을 사용하여 수행할 수 있습니다. 마이그레이션 작업은 취소할 수 없습니다.

마이그레이션 프로세스는 원활하며 서비스의 가동 중지 시간을 0으로 유지합니다. 마이그레이션 도중:

- 서비스 가용성: 트래픽 또는 기능을 중단하지 않고 서비스가 정상적으로 계속 실행됩니다.
- 실행 중인 태스크: 기존 태스크는 중단 없이 계속 실행됩니다. `taskLongArnFormat` 계정 설정이 활성화된 경우 마이그레이션 후 시작된 새 태스크는 긴 ARN 형식을 사용합니다.
- 컨테이너 인스턴스: 컨테이너 인스턴스는 서비스 ARN 마이그레이션의 영향을 받지 않으며 정상적으로 계속 작동합니다.

- 서비스 구성: 태스크 정의, 네트워킹, 로드 밸런서 구성을 포함한 모든 서비스 설정은 변경되지 않습니다.

AWS CloudFormation을 사용하여 짧은 ARN 형식으로 서비스에 태깅하려면 API, CLI 또는 콘솔을 사용하여 서비스를 마이그레이션해야 합니다. 마이그레이션이 완료되면 AWS CloudFormation을 사용하여 서비스에 태깅할 수 있습니다.

Terraform을 사용하여 짧은 ARN 형식으로 서비스에 태깅하려면 API, CLI 또는 콘솔을 사용하여 서비스를 마이그레이션해야 합니다. 마이그레이션이 완료되면 Terraform을 사용하여 서비스에 태깅할 수 있습니다.

마이그레이션이 완료되면 서비스가 다음과 같이 변경됩니다.

- 긴 ARN 형식

  `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`cluster-name`*`/`*`service-name`*

- 콘솔을 사용하여 마이그레이션하는 경우 Amazon ECS는 키가 'ecs:serviceArnMigratedAt'로 설정되고, 값이 마이그레이션 타임스탬프(UTC 형식)로 설정된 태그를 서비스에 추가합니다.

  이 태그는 태그 할당량에 반영됩니다.

- AWS CloudFormation 스택의 `PhysicalResourceId`가 서비스 ARN을 나타내는 경우 값은 변경되지 않으며 짧은 서비스 ARN으로 유지됩니다.

사전 조건

서비스 ARN을 마이그레이션하기 전에 다음 작업을 수행합니다.

1. 짧은 서비스 ARN이 있는지 확인하려면 Amazon ECS 콘솔에서 서비스 세부 정보를 보거나(서비스의 ARN 형식이 짧으면 경고가 표시됨) `describe-services`에서 반환되는 `serviceARN` 반환 파라미터를 확인합니다. ARN에 클러스터 이름이 포함되지 않은 경우 짧은 ARN입니다. 짧은 ARN의 형식은 다음과 같습니다.

   `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`service-name`*
2. 생성 날짜를 기록해 둡니다.
3. 짧은 ARN 형식을 사용하는 IAM 정책이 있는 경우 긴 ARN 형식으로 업데이트합니다.

   *`user input placeholder`*를 사용자의 정보로 바꿉니다.

   `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`cluster-name`*`/`*`service-name`*

자세한 내용은 AWS Identity and Access Management 사용 설명서에서 IAM 정책 편집을 참조하세요.

4. 짧은 ARN 형식을 사용하는 도구가 있는 경우 긴 ARN 형식으로 업데이트합니다.

   *user input placeholder*를 사용자의 정보로 바꿉니다.

   arn:aws:ecs:*region*:*aws_account_id*:service/*cluster-name*/*service-name*

5. 긴 서비스 ARN 형식을 활성화합니다. `serviceLongArnFormat` 옵션을 `enabled`로 설정하여 `put-account-setting`를 생성합니다. 자세한 내용은 Amazon Elastic Container Service API 참조에서 put-account-setting을 참조하세요.

   서비스의 `createdAt` 날짜를 알 수 없는 경우 루트 사용자로 명령을 실행합니다.

   ```
   aws ecs put-account-setting --name serviceLongArnFormat --value enabled
   ```

   출력 예시

   ```
   {
       "setting": {
           "name": "serviceLongArnFormat",
           "value": "enabled",
           "principalArn": "arn:aws:iam::123456789012:role/your-role",
           "type": user
       }
   }
   ```

6. 긴 태스크 ARN 형식을 활성화합니다. 이 계정 설정은 서비스 마이그레이션이 완료된 후 시작되는 새 태스크에 대한 ARN 형식을 제어합니다. `taskLongArnFormat` 옵션을 `enabled`로 설정하여 `put-account-setting`를 생성합니다. 자세한 내용은 Amazon Elastic Container Service API 참조에서 put-account-setting을 참조하세요.

   서비스의 `createdAt` 날짜를 알 수 없는 경우 루트 사용자로 명령을 실행합니다.

   ```
   aws ecs put-account-setting --name taskLongArnFormat --value enabled
   ```

   출력 예시

   ```
   {
       "setting": {
   ```

```
        "name": "taskLongArnFormat",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:role/your-role",
        "type": user
    }
}
```

> **Note**
>
> `taskLongArnFormat` 설정은 기존 태스크를 직접 마이그레이션하지 않습니다. 설정이 활성화된 후 생성되는 새 태스크의 ARN 형식에만 영향을 줍니다. 실행 중인 기존 태스크는 일반적인 서비스 작업(예: 배포 또는 조정 활동)을 통해 대체될 때까지 현재 ARN 형식을 유지합니다.

절차

다음을 사용하여 서비스 ARN을 마이그레이션합니다.

콘솔

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
3. 서비스 섹션에서 ARN 열에 경고가 있는 서비스를 선택합니다.

   서비스 세부 정보 페이지가 나타납니다.
4. 긴 ARN으로 마이그레이션을 선택합니다.

   서비스 마이그레이션 대화 상자가 나타납니다.
5. [Migrate]를 선택합니다.

CLI

사전 조건을 완료하고 나면 서비스에 태깅할 수 있습니다. 다음 명령 실행:

짧은 ARN이 있는 서비스에 대한 `tag-resource` API 요청에서 긴 ARN 형식을 전달할 경우, Amazon ECS는 긴 ARN 형식을 사용하도록 서비스를 마이그레이션하라는 신호로 간주합니다.

```
aws ecs tag-resource \
```

```
    --resource-arn arn:aws:ecs:region:aws_account_id:service/cluster-name/service-name
    --tags key=key1,value=value1
```

다음 예에서는 키가 'TestService'로, 값이 'WebServers'로 각각 설정된 태그로 MyService를 태깅합니다.

```
aws ecs tag-resource \
    --resource-arn arn:aws:ecs:us-east-1:123456789012:service/MyCluster/MyService
    --tags key=TestService1,value=WebServers
```

Terraform

사전 조건을 완료하고 나면 서비스에 태깅할 수 있습니다. `aws_ecs_service` 리소스를 생성하고 `tags` 참조를 설정합니다. 자세한 내용은 Terraform 설명서에서 Resource: aws_ecs_service를 참조하세요.

```
resource "aws_ecs_service" "MyService" {
  name    = "example"
  cluster = aws_ecs_cluster.MyService.id

 tags = {
 "Name"  =  "MyService"
 "Environment"  =  "Production"
 "Department"  =  "QualityAssurance"
  }
}
```

다음 단계

서비스에 태그를 추가할 수 있습니다. 자세한 내용은 Amazon ECS 리소스에 태그 추가 섹션을 참조하세요.

Amazon ECS가 태스크 정의 또는 서비스에서 태스크로 태그를 전파하도록 하려면 `propagateTags` 파라미터를 사용하여 `update-service`를 실행합니다. 자세한 내용은 AWS Command Line Interface 참조에서 update-service를 참조하세요.

문제 해결

일부 사용자는 짧은 ARN 형식에서 긴 ARN 형식으로 마이그레이션할 때 다음 오류가 발생할 수 있습니다.

`There was an error while migrating the ARN of service service-name. The specified account does not have serviceLongArnFormat or taskLongArnFormat account settings enabled. Add account settings in order to enable tagging.`

`serviceLongArnFormat` 계정 설정을 이미 활성화했지만 여전히 이 오류가 발생하는 경우 서비스를 처음 생성한 특정 IAM 위탁자에 대해 긴 ARN 형식의 계정 설정이 활성화되지 않았기 때문일 수 있습니다.

1. 서비스를 생성한 위탁자를 식별합니다.

   1. 콘솔에서 정보는 Amazon ECS 콘솔의 서비스 세부 정보 페이지에 있는 구성 및 네트워킹 탭의 생성자 필드에서 확인할 수 있습니다.
   2. AWS CLI의 경우 다음 명령을 실행합니다.

      모든 *### ##*을 사용자의 값으로 바꿉니다.

      ```
      aws ecs describe-services --cluster cluster-name --services service-name --query
       'services[0].{createdBy: createdBy}'
      ```

2. 해당 위탁자에 대해 필요한 계정 설정을 활성화합니다. 이 작업을 다음 중 한 가지 방법으로 수행할 수 있습니다.

   a. 해당 위탁자의 IAM 사용자 또는 역할을 수임합니다. 그런 다음 `put-account-setting`을(를) 실행합니다.

   b. 루트 사용자를 사용하여 명령을 실행하고 `principal-arn`으로 생성 위탁자를 지정합니다.

      예제.

      *principal-arn*을 1단계의 값으로 바꿉니다.

      ```
      aws ecs put-account-setting --name serviceLongArnFormat --value enabled --
      principal-arn arn:aws:iam::123456789012:role/jdoe
      ```

두 방법 모두 서비스를 생성한 위탁자에서 필요한 `serviceLongArnFormat` 계정 설정을 활성화하여 ARN 마이그레이션을 진행할 수 있습니다.

Amazon ECS 서비스 제한 로직

Amazon ECS 서비스 스케줄러는 태스크 시작이 반복적으로 실패하는 경우 태스크 시작 빈도를 스로틀링하는 로직을 포함합니다. 이를 통해 불필요한 리소스 소비를 방지하고 비용을 절감할 수 있습니다.

서비스의 태스크가 PENDING에서 RUNNING 상태로 전환되지 않고 대신 STOPPED로 직접 이동되는 경우 스케줄러는 다음을 수행합니다.

- 재시작 시도 간격을 점진적으로 늘림
- 시도 사이에 지연을 계속 늘림(최대 27분)
- 서비스 이벤트 메시지를 생성하여 사용자에게 문제를 알림

> **Note**
>
> 최대 지연 기간인 27분은 향후 업데이트에서 변경될 수 있습니다.

스로틀링이 활성화되면 다음 서비스 이벤트 메시지가 수신됩니다.

```
(service service-name) is unable to consistently start tasks successfully.
```

스로틀 로직의 중요한 특성:

- 서비스에서 무기한으로 계속해서 재시도함
- 재시작 사이에 늘어나는 시간만 수정 가능
- 사용자가 구성할 수 있는 파라미터가 없음

스로틀링 문제 해결

스로틀링을 해결하기 위해 다음을 수행할 수 있습니다.

- 새로운 태스크 정의를 사용하도록 서비스를 업데이트합니다. 그러면 서비스는 스로틀링되지 않은 정상 작업으로 즉시 돌아갑니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.
- 태스크 실패의 근본 원인을 해결합니다.

스로틀링을 트리거하는 태스크 실패의 일반적인 원인은 다음과 같습니다.

- 클러스터 리소스(포트, 메모리 또는 CPU) 부족
  - 리소스 부족 서비스 이벤트 메시지로 표시됨
- 컨테이너 이미지 풀 실패
  - 유효하지 않은 이미지 이름, 태그 또는 권한 부족으로 인해 발생할 수 있음
  - Amazon ECS 중지된 작업 오류 보기에서 `CannotPullContainerError` 발생
- 디스크 공간 부족
  - 중지된 태스크 오류에서 `CannotCreateContainerError` 발생
  - 해결 단계는 Amazon ECS의 Docker API `error (500): devmapper` 문제 해결 섹션을 참조하세요.

> **Important**
>
> 다음 시나리오에서는 스로틀 로직을 트리거하지 않습니다.
>
> - `RUNNING` 상태에 도달한 후 중지되는 태스크
> - 실패한 Elastic Load Balancing 상태 확인으로 인해 중지되는 태스크
> - `RUNNING` 상태에 도달한 후 0이 아닌 코드와 함께 컨테이너 명령이 종료되는 태스크

Amazon ECS 서비스 정의 파라미터

서비스 정의는 Amazon ECS 서비스 실행 방법을 정의합니다. 서비스 정의에서 다음 파라미터를 지정할 수 있습니다.

시작 유형

`launchType`

유형: 문자열

유효한 값: `EC2` | `FARGATE` | `EXTERNAL`

필수 여부: 아니요

서비스를 실행할 시작 유형. 시작 유형을 지정하지 않으면 기본적으로 `capacityProviderStrategy`가 사용됩니다. 자세한 정보는 Amazon ECS 시작 유형을 참조하세요.

launchType이 지정된 경우 capacityProviderStrategy 파라미터를 생략해야 합니다.

용량 제공자 전략

capacityProviderStrategy

유형: 객체 배열

필수 여부: 아니요

서비스에 사용할 용량 공급자 전략입니다.

용량 공급자 전략은 할당할 base 및 weight와 함께 하나 이상의 용량 공급자로 구성됩니다. 용량 공급자는 용량 공급자 전략에 사용할 클러스터와 연결되어야 합니다. PutClusterCapacityProviders API는 용량 공급자를 클러스터와 연결하는 데 사용됩니다. ACTIVE 또는 UPDATING 상태의 용량 공급자만 사용할 수 있습니다.

capacityProviderStrategy가 지정된 경우 launchType 파라미터를 생략해야 합니다. capacityProviderStrategy 또는 launchType이 지정되지 않은 경우 클러스터에 대한 defaultCapacityProviderStrategy가 사용됩니다.

Auto Scaling 그룹을 사용하는 용량 공급자를 지정하려는 경우 용량 공급자가 이미 생성되어 있어야 합니다. CreateCapacityProvider API 태스크를 사용하여 새 용량 공급자를 생성할 수 있습니다.

AWS Fargate 용량 공급자를 사용하려면 FARGATE 또는 FARGATE_SPOT 용량 공급자를 지정합니다. AWS Fargate 용량 공급자는 모든 계정에 사용할 수 있으며 사용할 클러스터와 연결하기만 하면 됩니다.

PutClusterCapacityProviders API 태스크는 클러스터를 생성한 후 클러스터에 사용 가능한 용량 공급자 목록을 업데이트하는 데 사용됩니다.

capacityProvider

유형: 문자열

필수 항목 여부: 예

용량 공급자의 짧은 이름 또는 전체 Amazon 리소스 이름(ARN)입니다.

weight

유형: 정수

유효한 범위: 0에서 1,000 사이의 정수

필수 여부: 아니요

가중치 값은 지정된 용량 공급자를 사용하는 시작된 총 태스크 수의 상대 백분율을 지정합니다.

예를 들어 두 개의 용량 공급자를 포함하는 전략이 있고 둘 다 1의 가중치를 갖는 경우를 가정합니다. 기본이 충족되면 태스크는 두 용량 공급자에 균등하게 분할됩니다. 동일한 로직을 사용하여 capacityProviderA의 가중치를 1, capacityProviderB의 가중치를 4로 지정한다고 가정합니다. 그런 다음 capacityProviderA를 사용하여 실행되는 각 태스크에 대해 4개의 태스크가 capacityProviderB를 사용합니다.

`base`

유형: 정수

유효한 범위: 0에서 100,000 사이의 정수

필수 여부: 아니요

최소한 기준 값은 지정된 용량 공급자에서 실행할 태스크 수를 지정합니다. 용량 공급자 전략에서 하나의 용량 공급자만 기준을 정의할 수 있습니다.

태스크 정의

`taskDefinition`

유형: 문자열

필수 여부: 아니요

서비스에서 실행할 태스크 정의의 `family` 및 `revision`(`family:revision`) 또는 전체 Amazon 리소스 이름(ARN)입니다. `revision`이 지정되지 않으면 지정된 패밀리의 최신 `ACTIVE` 개정이 사용됩니다.

롤링 업데이트(ECS) 배포 컨트롤러를 사용할 때 태스크 정의를 지정해야 합니다.

플랫폼 운영 체제

`platformFamily`

타입: 문자열

필수 항목 여부: 조건부

기본값: Linux

이 파라미터는 Fargate에서 호스팅되는 Amazon ECS 서비스에 필요합니다.

이 파라미터는 Amazon EC2에 호스팅된 Amazon ECS 서비스에서 무시됩니다.

서비스를 실행하는 컨테이너의 운영 체제입니다. 유효한 값은 `LINUX`, `WINDOWS_SERVER_2019_FULL`, `WINDOWS_SERVER_2019_CORE`, `WINDOWS_SERVER_2022_FULL` 및 `WINDOWS_SERVER_2022_CORE`입니다.

서비스에 대해 지정하는 모든 태스크의 `platformFamily` 값은 서비스 `platformFamily` 값과 일치해야 합니다. 예를 들어 `platformFamily`를 `WINDOWS_SERVER_2019_FULL`로 설정하는 경우 모든 태스크에 대한 `platformFamily` 값은 `WINDOWS_SERVER_2019_FULL`이어야 합니다.

플랫폼 버전

`platformVersion`

유형: 문자열

필수 여부: 아니요

서비스의 작업이 실행 중인 플랫폼 버전입니다. 플랫폼 버전은 Fargate 시작 유형을 사용하는 작업에만 지정됩니다. 지정하지 않으면 기본적으로 최신 버전(`LATEST`)이 사용됩니다.

AWS Fargate 플랫폼 버전은 Fargate 태스크 인프라를 위한 특정 실행 시간 환경을 참조하는 데 사용합니다. 태스크를 실행하거나 서비스를 생성할 때 `LATEST` 플랫폼 버전을 지정하면 해당 작업에 사용 가능한 최신 플랫폼 버전을 얻을 수 있습니다. 서비스를 확장하면 해당 태스크는 서비스의 현재 배포에 지정된 플랫폼 버전을 받습니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.

> **Note**
>
> 플랫폼 버전은 EC2 시작 유형을 사용하는 작업에는 지정되지 않습니다.

클러스터

`cluster`

유형: 문자열

필수 항목 여부: 아니요

서비스를 실행할 클러스터의 짧은 이름 또는 전체 Amazon 리소스 이름(ARN)입니다. 클러스터를 지정하지 않으면 `default` 클러스터가 가정됩니다.

서비스 이름

`serviceName`

유형: 문자열

필수 항목 여부: 예

서비스의 이름입니다. 최대 255개의 문자(대문자 및 소문자), 숫자, 하이픈 및 밑줄이 허용됩니다. 서비스 이름은 클러스터 내에서 고유해야 하지만, 한 리전 또는 여러 리전에 걸쳐 존재하는 여러 클러스터에서 비슷한 서비스 이름을 사용할 수 있습니다.

일정 전략

`schedulingStrategy`

유형: 문자열

유효한 값: `REPLICA` | `DAEMON`

필수 여부: 아니요

사용할 일정 전략입니다. 일정 전략이 지정되지 않은 경우 `REPLICA` 전략이 사용됩니다. 자세한 내용은 Amazon ECS 서비스 섹션을 참조하세요.

사용 가능한 서비스 스케줄러 전략으로 다음 두 가지가 있습니다.

- `REPLICA`—복제본 일정 전략은 클러스터에 원하는 작업 수를 배치하고 유지합니다. 기본적으로 서비스 스케줄러는 가용 영역에 태스크를 분산합니다. 작업 배치 전략과 제약을 사용하여 작업 배치 결정을 사용자 지정할 수 있습니다. 자세한 내용은 복제본 예약 전략 섹션을 참조하세요.
- `DAEMON`—데몬 일정 전략은 사용자가 클러스터에 지정하는 작업 배치 제약을 모두 충족하는 각 활성 컨테이너 인스턴스에 한 작업씩 정확히 배포합니다. 이 전략을 사용하는 경우 원하는 태스크 수, 작업 배치 전략을 지정하거나 서비스 Auto Scaling 정책을 사용할 필요가 없습니다. 자세한 내용은 대몬 예약 전략 섹션을 참조하세요.

> **Note**
>
> Fargate 태스크는 DAEMON 일정 전략을 지원하지 않습니다.

원하는 개수

`desiredCount`

유형: 정수

필수 여부: 아니요

서비스에서 배치되고 계속 실행될 지정된 작업 정의의 인스턴스화 수입니다.

REPLICA 일정 전략이 사용되는 경우 이 파라미터가 필요합니다. 서비스에서 DAEMON 일정 전략을 사용하는 경우 이 파라미터는 선택 사항입니다.

서비스 오토 스케일링을 사용하는 경우, 현재 실행 중인 서비스를 현재 실행 중인 태스크 수보다 적은 desiredCount로 업데이트하면 서비스가 지정된 desiredCount로 스케일 다운됩니다.

배포 구성

`deploymentConfiguration`

유형: 객체

필수 여부: 아니요

배포 시 실행할 작업 수 및 작업 중지/시작 순서를 제어하는 선택적 배포 파라미터입니다.

`maximumPercent`

유형: 정수

필수 여부: 아니요

서비스가 롤링 업데이트(ECS) 배포 유형을 사용하는 경우, maximumPercent 파라미터는 배포 중에 RUNNING, STOPPING 또는 PENDING 상태에서 허용되는 서비스의 작업 수에 대한 상한을 나타냅니다. 이는 가장 가까운 정수로 반내림된 desiredCount의 백분율로 표현됩니다. 이 파라미터를 사용하여 배포 배치 크기를 정의할 수 있습니다. 예를 들어 서비스에서 REPLICA 서비스 스케줄러를 사용 중이고 desiredCount가 태스크 4개이고 maximumPercent 값이 200%

인 경우, 스케줄러에서는 기존 태스크 4개를 중지하기 전에 새 태스크 4개를 시작합니다. 단, 이렇게 하는 데 필요한 클러스터 리소스를 사용할 수 있는 경우를 전제로 합니다. REPLICA 서비스 스케줄러를 사용하는 서비스의 기본 maximumPercent 값은 200%입니다.

Amazon ECS 스케줄러에서는 대체 작업을 시작할 클러스터 리소스가 있으면 이 파라미터를 사용하여 대체 태스크를 먼저 시작한 다음에 비정상 태스크를 중지하여 비정상 태스크를 바꿉니다. 스케줄러가 비정상 태스크를 바꾸는 방식에 대한 자세한 내용은 Amazon ECS 서비스 단원을 참조하세요.

서비스가 DAEMON 서비스 스케줄러 유형을 사용하는 경우 maximumPercent는 100%를 유지해야 합니다. 이것이 기본값입니다.

배포 시 최대 작업 수는 desiredCount에 maximumPercent/100을 곱하여 가장 가까운 정수로 내림한 값입니다.

서비스가 EC2 시작 유형을 사용하는 블루/그린(CODE_DEPLOY) 또는 EXTERNAL 배포 유형 및 작업을 사용하는 경우, 최대 백분율 값은 기본값으로 설정됩니다. 값은 컨테이너 인스턴스가 DRAINING 상태에 있을 때 RUNNING 상태로 유지되는 서비스의 작업 수에 대한 상한을 정의하는 용도로만 사용됩니다.

> **Note**
>
> 블루/그린 (CODE_DEPLOY) 또는 EXTERNAL 배포 유형을 사용하고 EC2 시작 유형을 사용하는 작업이 있는 서비스에는 사용자 지정 maximumPercent 값을 지정할 수 없습니다.

서비스가 블루/그린(CODE_DEPLOY) 또는 EXTERNAL 배포 유형을 사용하고 서비스의 작업이 Fargate 시작 유형을 사용하는 경우 최대 백분율 값이 사용되지 않습니다. 서비스를 설명할 때 값은 여전히 반환됩니다.

`minimumHealthyPercent`

유형: 정수

필수 여부: 아니요

서비스가 롤링 업데이트(ECS) 배포 유형을 사용하는 경우, minimumHealthyPercent는 배포 중에 RUNNING 상태로 유지되어야 하는 서비스의 태스크 수에 대한 하한을 나타냅니다. 이는 가장 가까운 정수로 반올림된 desiredCount의 백분율로 표현됩니다. 이 파라미터를 사용하여 추가 클러스터 용량을 사용하지 않고 배포할 수 있습니다.

예를 들면, 서비스에서 desiredCount의 태스크는 4개, minimumHealthyPercent는 50%, maximumPercent는 100%인 경우 서비스 스케줄러에서는 새 태스크 2개를 시작하기 전에 기존 태스크 2개를 중지하여 클러스터 용량을 확보할 수 있습니다.

비정상인 태스크가 있고 maximumPercent에서 Amazon ECS 스케줄러의 대체 태스크 시작이 허용되지 않는 경우 스케줄러에서는 제약 조건으로 minimumHealthyPercent를 사용하여 비정상 태스크를 하나씩 중지하면서 용량을 정리하여 대체 태스크를 시작합니다. 스케줄러가 비정상 태스크를 바꾸는 방식에 대한 자세한 내용은 Amazon ECS 서비스 단원을 참조하세요.

로드 밸런서를 사용하지 않는 서비스의 경우 다음 사항에 유의해야 합니다.

- 서비스의 작업 내 모든 필수 컨테이너가 상태 확인을 통과하면 서비스가 정상 상태로 간주됩니다.
- 정의된 상태 확인이 있는 필수 컨테이너가 태스크에 없는 경우, 서비스 스케줄러는 태스크가 최소 정상 백분율 합계로 계산되기 전에 태스크가 RUNNING 상태에 도달한 후 40초 동안 대기합니다.
- 정의된 상태 확인이 있는 하나 이상의 필수 컨테이너가 태스크에 있는 경우, 서비스 스케줄러는 태스크가 최소 정상 백분율 합계로 계산되기 전에 태스크가 정상 상태에 도달할 때까지 대기합니다. 작업 내 모든 필수 컨테이너가 상태 확인을 통과하면 서비스가 정상 상태로 간주됩니다. 서비스 스케줄러가 대기할 수 있는 시간은 컨테이너 상태 확인 설정에 따라 결정됩니다. 자세한 정보는 상태 확인을 참조하세요.

로드 밸런서를 사용하는 서비스의 경우 다음 사항에 유의해야 합니다.

- 태스크에 상태 확인이 정의된 필수 컨테이너가 없는 경우, 서비스 스케줄러는 로드 밸런서 대상 그룹 상태 확인이 정상 상태를 반환할 때까지 기다렸다가 작업을 최소 정상 백분율 합계로 계산합니다.
- 태스크에 상태 확인이 정의된 필수 컨테이너가 있는 경우, 서비스 스케줄러는 작업이 정상 상태에 도달하고 로드 밸런서 대상 그룹 상태 확인이 모두 정상 상태를 반환할 때까지 기다렸다가 작업을 최소 정상 백분율 합계로 계산합니다.

minimumHealthyPercent에 대한 복제본 서비스의 기본값은 100%입니다. DAEMON 서비스 일정을 사용하는 서비스의 기본 minimumHealthyPercent 값은 AWS CLI, AWS SDK 및 API의 경우 0%이고, AWS Management Console의 경우 50%입니다.

배포 중 정상 작업의 최소 개수는 desiredCount에 minimumHealthyPercent/100을 곱하여 가장 가까운 정수로 반올림한 값입니다.

서비스가 EC2 시작 유형을 사용하는 블루/그린(CODE_DEPLOY) 또는 EXTERNAL 배포 유형을 사용하고 작업을 실행하는 경우, 최소 정상 상태 백분율 값이 기본값으로 설정됩니다. 값은 컨테

이너 인스턴스가 DRAINING 상태에 있을 때 RUNNING 상태로 유지되는 서비스의 작업 수에 대한 하한을 정의하는 용도로만 사용됩니다.

> **Note**
>
> 블루/그린 (CODE_DEPLOY) 또는 EXTERNAL 배포 유형을 사용하고 EC2 시작 유형을 사용하는 작업이 있는 서비스에는 사용자 지정 maximumPercent 값을 지정할 수 없습니다.

서비스가 블루/그린(CODE_DEPLOY) 또는 EXTERNAL 배포 유형을 사용하고 Fargate 시작 유형을 사용하는 태스크를 실행하는 경우, 최소 정상 상태 백분율 값은 서비스를 설명할 때 반환되지만 사용되지는 않습니다.

배포 컨트롤러

deploymentController

유형: 객체

필수 여부: 아니요

서비스에 사용할 배포 컨트롤러입니다. 배포 컨트롤러가 지정되지 않은 경우 ECS 컨트롤러가 사용됩니다. 자세한 내용은 Amazon ECS 서비스 섹션을 참조하세요.

type

유형: 문자열

유효한 값: ECS | CODE_DEPLOY | EXTERNAL

필수 항목 여부: 예

사용할 배포 컨트롤러 유형입니다. 사용 가능한 배포 컨트롤러 유형은 세 가지입니다.

ECS

롤링 업데이트(ECS) 배포 유형은 현재 실행 중인 컨테이너 버전을 최신 버전으로 바꿉니다. 롤링 업데이트 중에 Amazon ECS가 서비스에서 추가하거나 제거하는 컨테이너의 수는 deploymentConfiguration에 지정된 대로 서비스 배포 중에 허용되는 정상 작업의 최소 및 최대 수를 조정하여 제어합니다.

`CODE_DEPLOY`

블루/그린(`CODE_DEPLOY`) 배포 유형은 CodeDeploy를 기반으로 하는 블루/그린 배포 모델을 사용하므로 프로덕션 트래픽을 전송하기 전에 서비스의 새로운 배포를 확인할 수 있습니다.

`EXTERNAL`

외부 배포 유형을 사용하면 타사 배포 컨트롤러를 사용하여 Amazon ECS 서비스의 배포 프로세스를 완벽하게 제어할 수 있습니다.

작업 배치

`placementConstraints`

타입: 객체 배열

필수 여부: 아니요

서비스 내 작업에 사용할 배치 제약 객체의 배열입니다. 태스크당 최대 10개의 제약 조건을 지정할 수 있습니다. 이 제한에는 태스크 정의 내의 제약 조건과 런타임 시 지정되는 제약 조건이 포함됩니다. Fargate 시작 유형을 사용하는 경우 작업 배치 제약은 지원되지 않습니다.

`type`

유형: 문자열

필수 여부: 아니요

제약의 유형입니다. `distinctInstance`를 사용하여 특정 그룹의 각 작업이 다른 컨테이너 인스턴스에서 실행 중인지 확인합니다. `memberOf`를 사용하여 선택을 유효한 후보 그룹으로 제한합니다. 값 `distinctInstance`는 태스크 정의에서 지원되지 않습니다.

`expression`

유형: 문자열

필수 여부: 아니요

제약에 적용할 클러스터 쿼리 언어 표현식입니다. 제약 유형이 `distinctInstance`일 경우, 표현식을 지정할 수 없습니다. 자세한 정보는 Amazon ECS 작업에 대한 컨테이너 인스턴스를 정의하도록 표현식 생성을 참조하세요.

`placementStrategy`

유형: 객체 배열

필수 여부: 아니요

서비스 내 작업에 사용할 배치 전략 객체입니다. 서비스당 최대 4개까지 전략 규칙을 지정할 수 있습니다.

`type`

유형: 문자열

유효한 값: `random` | `spread` | `binpack`

필수 여부: 아니요

배치 전략의 유형입니다. `random` 배치 전략은 사용 가능한 후보에 태스크를 임의로 배치합니다. `spread` 배치 전략은 `field` 파라미터를 기반으로 사용 가능한 후보 간에 배치를 고르게 분산시킵니다. `binpack` 전략은 `field` 파라미터로 지정된 사용 가능한 최소량의 리소스가 있는 사용 가능한 후보에 태스크를 배치합니다. 예를 들어 메모리에 빈팩 전략을 적용하면 남은 메모리 양이 가장 적지만 여전히 태스크를 실행하기에 충분한 인스턴스에 태스크가 배치됩니다.

`field`

유형: 문자열

필수 여부: 아니요

배치 전략을 적용할 필드입니다. `spread` 배치 전략의 경우 유효한 값은 `instanceId`(또는 `host`, 효과는 동일함)이거나, 모든 플랫폼 또는 컨테이너 인스턴스에 적용되는 사용자 지정 속성(예: `attribute:ecs.availability-zone`)입니다. `binpack` 배치 전략의 경우 유효한 값은 `cpu` 및 `memory`입니다. `random` 배치 전략의 경우 이 필드가 사용되지 않습니다.

Tags

`tags`

타입: 객체 배열

필수 여부: 아니요

서비스를 분류하고 구성하는 데 도움이 되도록 서비스에 적용하는 메타데이터입니다. 각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다. 서비스가 삭제되면 태그도 함께 삭제됩니다. 서비스에 최대 50개의 태그를 적용할 수 있습니다. 자세한 정보는 Amazon ECS 리소스 태그 지정 섹션을 참조하세요.

`key`

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 128입니다.

필수 여부: 아니요

하나의 태그를 구성하는 키-값 쌍의 일부분입니다. 키는 더 구체적인 태그 값에 대해 범주와 같은 역할을 하는 일반적인 레이블입니다.

`value`

유형: 문자열

길이 제약: 최소 길이는 0. 최대 길이 256.

필수 여부: 아니요

하나의 태그를 구성하는 키-값 쌍의 선택적 부분입니다. 하나의 값은 태그 범주(키) 내에서 서술자 역할을 수행합니다.

`enableECSManagedTags`

유형: 부울

유효한 값: `true` | `false`

필수 여부: 아니요

서비스의 태스크에 대해 Amazon ECS 관리형 태그를 사용할지를 지정합니다. 값을 지정하지 않을 경우, 기본값은 `false`입니다. 자세한 정보는 결제에 태그 사용 섹션을 참조하세요.

`propagateTags`

유형: 문자열

유효한 값: `TASK_DEFINITION` | `SERVICE`

필수 여부: 아니요

태그를 태스크 정의 또는 서비스에서 서비스의 작업으로 복사할지를 지정합니다. 값을 지정하지 않을 경우 태그는 복사되지 않습니다. 태그는 서비스 생성 중에 서비스 내의 작업에만 복사할 수 있습니다. 서비스 생성 후 작업 생성에 태그를 추가하려면 `TagResource` API 태스크를 사용합니다.

네트워크 구성

`networkConfiguration`

유형: 객체

필수 여부: 아니요

서비스에 대한 네트워크 구성. 이 파라미터는 `awsvpc` 네트워크 모드를 사용하는 태스크 정의가 고유한 탄력적 네트워크 인터페이스를 받는 데 필요하며 다른 네트워크 모드에 대해서는 지원되지 않습니다. Fargate 시작 유형을 사용하는 경우 `awsvpc` 네트워크 모드가 필수입니다. Amazon EC2 시작 유형의 네트워킹에 대한 자세한 내용은 EC2 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션 섹션을 참조하세요. Fargate 시작 유형의 네트워킹에 대한 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 태스크 네트워킹 옵션을 참조하세요.

`awsvpcConfiguration`

유형: 객체

필수 여부: 아니요

작업 또는 서비스를 위한 서브넷 및 보안 그룹을 나타내는 객체.

`subnets`

유형: 문자열 배열

필수 여부: 예

태스크 또는 서비스와 연결된 서브넷입니다. `awsvpcConfiguration`에 따라 지정할 수 있는 서브넷의 한도는 16개입니다.

`securityGroups`

유형: 문자열 배열

필수 여부: 아니요

작업 또는 서비스와 연결된 보안 그룹. 보안 그룹을 지정하지 않을 경우 VPC에 대해 기본 설정된 보안 그룹이 사용됩니다. awsvpcConfiguration에 따라 지정할 수 있는 보안 그룹의 한도는 5개입니다.

assignPublicIP

유형: 문자열

유효한 값: ENABLED | DISABLED

필수 여부: 아니요

작업의 탄력적 네트워크 인터페이스가 퍼블릭 IP 주소를 수신하는지 여부입니다. 값을 지정하지 않으면 DISABLED의 기본값이 사용됩니다.

healthCheckGracePeriodSeconds

유형: 정수

필수 여부: 아니요

이 기간(초) 동안 Amazon ECS 서비스 스케줄러는 태스크가 처음 시작된 후 비정상 Elastic Load Balancing, VPC Lattice, 컨테이너 상태 확인을 무시합니다. 상태 확인 유예 기간 값을 지정하지 않으면 기본값인 0이 사용됩니다. 상태 확인을 사용하지 않으면 healthCheckGracePeriodSeconds는 사용되지 않습니다.

서비스의 태스크가 시작하고 응답하는 데 시간이 다소 걸린다면 최대 2,147,483,647초(약 69년)의 상태 확인 유예 기간을 지정할 수 있습니다. 이 기간에는 Amazon ECS 서비스 스케줄러가 상태 확인의 상태를 무시합니다. 이 유예 기간은 서비스 스케줄러가 태스크를 비정상으로 표시하여 작업 처리 전에 이를 중단시키는 일이 없도록 해줍니다.

loadBalancers

타입: 객체 배열

필수 여부: 아니요

서비스와 함께 사용할 로드 밸런서를 나타내는 로드 밸런서 객체입니다. Application Load Balancer 또는 Network Load Balancer를 사용하는 서비스의 경우, 서비스에 연결할 수 있는 대상 그룹은 5개로 제한됩니다.

서비스를 생성한 후에는 로드 밸런서 구성을 AWS Management Console에서 변경할 수 없습니다. AWS Copilot, AWS CloudFormation, AWS CLI 또는 SDK를 사용하여 로드 밸런서 구성을 AWS CodeDeploy 블루/그린 또는 외부가 아닌 ECS 롤링 배포 컨트롤러에 대해서만 변경할 수 있습니다. 로드 밸런서 구성을 추가, 업데이트 또는 제거하면 Amazon ECS가 업데이트된 Elastic Load Balancing 구성을 사용하여 새 배포를 시작합니다. 이로 인해 작업이 로드 밸런서에 등록되거나 로드 밸런서에서 등록 취소됩니다. Elastic Load Balancing 구성을 업데이트하기 전에 테스트 환경에서 이를 확인하는 것이 좋습니다. 구성을 변경하는 방법에 대한 자세한 정보는 Amazon Elastic Container Service API Reference(Amazon Elastic Container Service API 레퍼런스)의 UpdateService를 참조하세요.

Application Load Balancer와 Network Load Balancer의 경우 이 객체는 로드 밸런서 대상 그룹 ARN, 컨테이너 이름(컨테이너 정의 내 이름과 동일), 로드 밸런서로부터 액세스할 컨테이너 포트를 포함해야 합니다. 이 서비스의 태스크가 컨테이너 인스턴스에 배치되면 컨테이너 인스턴스 및 포트 조합이 지정된 대상 그룹의 대상으로 등록됩니다.

`targetGroupArn`

유형: 문자열

필수 여부: 아니요

서비스와 연결된 Elastic Load Balancing 대상 그룹의 전체 Amazon 리소스 이름(ARN)입니다.

대상 그룹 ARN은 Application Load Balancer 또는 Network Load Balancer를 사용할 때만 지정됩니다.

`loadBalancerName`

유형: 문자열

필수 여부: 아니요

서비스와 연결할 로드 밸런서의 이름입니다.

Application Load Balancer 또는 Network Load Balancer를 사용하는 경우 로드 밸런서 이름 파라미터를 생략합니다.

`containerName`

유형: 문자열

필수 여부: 아니요

로드 밸런서와 연결할 컨테이너의 이름(컨테이너 정의 내 이름과 동일)입니다.

`containerPort`

유형: 정수

필수 여부: 아니요

로드 밸런서와 연결할 컨테이너의 포트입니다. 이 포트는 서비스 내 작업이 사용하는 태스크 정의의 `containerPort`와 일치해야 합니다. EC2 시작 유형을 사용하는 태스크의 경우, 컨테이너 인스턴스가 포트 매핑의 `hostPort`에서 인바운드 트래픽을 허용해야 합니다.

`role`

유형: 문자열

필수 여부: 아니요

Amazon ECS가 사용자 대신 로드 밸런서를 호출하도록 허용하는 IAM 역할의 짧은 이름 또는 전체 ARN입니다. 이 파라미터는 서비스에 대한 단일 대상 그룹과 함께 로드 밸런서를 사용하며 태스크 정의에서 `awsvpc` 네트워크 모드를 사용하지 않는 경우에만 허용됩니다. `role` 파라미터를 지정하는 경우 `loadBalancers` 파라미터를 사용하여 로드 밸런서 객체도 지정해야 합니다.

지정된 역할이 `/` 이외의 다른 경로를 갖는 경우 전체 역할 ARN을 지정(권장 방법)하거나 경로에 역할 이름을 접두사로 추가해야 합니다. 예를 들어 이름이 `bar`이고 경로가 `/foo/`인 역할에 대해 `/foo/bar`를 역할 이름으로 지정합니다. 자세한 정보는 IAM 사용 설명서의 표시 이름 및 경로를 참조하세요.

> **Important**
>
> 해당 계정에서 Amazon ECS 서비스 연결 역할을 이미 생성한 경우, 여기에 역할을 지정하지 않는 한 해당 역할이 기본적으로 서비스에 사용됩니다. 태스크 정의에서 awsvpc 네트워크 모드를 사용할 경우 서비스 연결 역할이 필요합니다. 이 경우 여기에 역할을 지정하면 안 됩니다. 자세한 내용은 Amazon ECS에 대해 서비스 연결 역할 사용 섹션을 참조하세요.

`serviceConnectConfiguration`

유형: 객체

필수 여부: 아니요

서비스를 검색 및 연결하고, 네임스페이스 내의 다른 서비스에서 검색하거나 연결하기 위한 이 서비스의 구성입니다.

자세한 내용은 Service Connect를 사용하여 짧은 이름으로 Amazon ECS 서비스 연결 섹션을 참조하세요.

`enabled`

유형: 부울

필수 여부: 예

이 서비스에서 Service Connect를 사용할지 여부를 지정합니다.

`namespace`

유형: 문자열

필수 여부: 아니요

Service Connect와 함께 사용할 AWS Cloud Map 네임스페이스의 약식 이름 또는 전체 Amazon 리소스 이름(ARN)입니다. 네임스페이스는 Amazon ECS 서비스 및 클러스터와 같은 AWS 리전에 있어야 합니다. 네임스페이스 유형은 서비스 연결에 영향을 미치지 않습니다. AWS Cloud Map에 대한 자세한 내용은 AWS Cloud Map 개발자 안내서의 Working with Services(서비스 작업)를 참조하세요.

`services`

타입: 객체 배열

필수 여부: 아니요

Service Connect 서비스 객체의 배열입니다. 이는 다른 Amazon ECS 서비스에서 이 서비스에 연결하는 데 사용하는 이름과 별칭(엔드포인트라고도 함)입니다.

네임스페이스의 구성원인 '클라이언트' Amazon ECS 서비스가 네임스페이스 내의 다른 서비스에만 연결하려는 경우에는 이 필드가 필요하지 않습니다. 서비스에 연결된 로드 밸런서나 다른 수단을 통해 들어오는 요청을 수락하는 프런트엔드 애플리케이션을 예로 들 수 있습니다.

객체는 작업 정의에서 포트를 선택하고, AWS Cloud Map 서비스 이름을 할당하고, 클라이언트 애플리케이션이 이 서비스를 참조할 수 있는 별칭 배열(엔드포인트라고도 함) 및 포트를 할당합니다.

`portName`

유형: 문자열

필수 항목 여부: 예

`portName`은 이 Amazon ECS 서비스의 작업 정의에 있는 모든 컨테이너에 있는 `portMappings` 중 하나의 `name`과 일치해야 합니다.

`discoveryName`

유형: 문자열

필수 여부: 아니요

`discoveryName`은 이 Amazon ECS 서비스에 대해 Amazon ECS에서 생성하는 새 AWS Cloud Map 서비스의 이름입니다. AWS Cloud Map 네임스페이스 내에서 고유해야 합니다.

이 필드가 지정되지 않은 경우 `portName`이 사용됩니다.

`clientAliases`

타입: 객체 배열

필수 여부: 아니요

이 서비스 연결 서비스의 클라이언트 별칭 목록입니다. 이를 사용하여 클라이언트 애플리케이션에서 사용할 수 있는 이름을 할당합니다. 이 목록에 포함할 수 있는 최대 클라이언트 별칭 수는 1개입니다.

각 별칭('엔드포인트')은 다른 Amazon ECS 서비스('클라이언트')가 이 서비스에 연결하는 데 사용할 수 있는 DNS 이름 및 포트 번호입니다.

각 이름과 포트 조합은 네임스페이스 내에서 고유해야 합니다.

이러한 이름은 클라이언트 서비스의 각 작업 내에서 구성되며, AWS Cloud Map에서는 구성되지 않습니다. 이러한 이름을 확인하기 위한 DNS 요청은 작업을 종료하지 않으며, 탄력적 네트워크 인터페이스별 초당 DNS 요청 할당량에 포함되지 않습니다.

`port`

유형: 정수

필수 여부: 예

서비스 연결 프록시의 수신 대기 포트 번호입니다. 이 포트는 동일한 네임스페이스 내의 모든 작업 안에서 사용할 수 있습니다.

클라이언트 Amazon ECS 서비스에서 애플리케이션을 변경하지 않으려면 클라이언트 애플리케이션이 기본적으로 사용하는 포트와 동일한 포트로 설정합니다.

`dnsName`

유형: 문자열

필수 여부: 아니요

`dnsName`은 클라이언트 작업의 애플리케이션에서 이 서비스에 연결하는 데 사용하는 이름입니다. 이름은 유효한 DNS 레이블이어야 합니다.

이 필드가 지정되지 않으면 기본값은 `discoveryName.namespace`입니다. `discoveryName`이 지정되지 않으면 작업 정의에서 `portName`이 사용됩니다.

클라이언트 Amazon ECS 서비스에서 애플리케이션을 변경하지 않으려면 클라이언트 애플리케이션이 기본적으로 사용하는 포트와 동일한 이름으로 설정합니다. 예를 들어 몇 가지 일반적인 이름은 `database`, `db` 또는 데이터베이스의 소문자 이름(`mysql` 또는 `redis`)입니다.

`ingressPortOverride`

유형: 정수

필수 여부: 아니요

(선택 사항) Service Connect 프록시가 수신 대기할 포트 번호입니다.

이 필드의 값을 사용하면 이 애플리케이션의 작업 정의에서 명명된 `portMapping`에 지정된 포트 번호의 트래픽에 대한 프록시를 우회하고, Amazon VPC 보안 그룹에서 이를 사용하여 이 Amazon ECS 서비스의 프록시로 들어오는 트래픽을 허용할 수 있습니다.

`awsvpc` 모드에서 기본값은 이 애플리케이션의 작업 정의에서 `portMapping`에 지정된 컨테이너 포트 번호입니다. `bridge` 브리지 모드에서 기본값은 Service Connect 프록시의 동적 임시 포트입니다.

`logConfiguration`

유형: LogConfiguration 객체

필수 여부: 아니요

Service Connect 프록시 로그가 게시되는 위치를 정의합니다. 예상치 못한 이벤트 발생 시 로그를 디버깅하는 데 사용합니다. 이 구성은 이 Amazon ECS 서비스의 각 작업에 있는 Service Connect 프록시 컨테이너의 `logConfiguration` 파라미터를 설정합니다. 프록시 컨테이너는 작업 정의에 지정되지 않습니다.

이 Amazon ECS 서비스에 대한 작업 정의의 애플리케이션 컨테이너와 동일한 로그 구성을 사용하는 것이 좋습니다. FireLens의 경우 이는 애플리케이션 컨테이너의 로그 구성입니다. `fluent-bit` 또는 `fluentd` 컨테이너 이미지를 사용하는 FireLens 로그 라우터 컨테이너가 아닙니다.

`serviceRegistries`

타입: 객체 배열

필수 여부: 아니요

서비스의 서비스 검색 구성에 대한 세부 정보입니다. 자세한 정보는 서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결 섹션을 참조하세요.

`registryArn`

유형: 문자열

필수 여부: 아니요

서비스 레지스트리의 Amazon 리소스 이름(ARN)입니다. 현재 지원되는 서비스 레지스트리는 AWS Cloud Map입니다. 자세한 정보는 AWS Cloud Map 개발자 안내서에서 서비스 태스크를 참조하세요.

`port`

유형: 정수

필수 여부: 아니요

서비스 검색 서비스에서 SRV 레코드를 지정한 경우 사용되는 포트 값입니다. `awsvpc` 네트워크 모드와 SRV 레코드를 모두 사용하는 경우 이 필드는 필수입니다.

`containerName`

유형: 문자열

필수 여부: 아니요

서비스 검색 서비스에 사용할 컨테이너 이름 값입니다. 이 값은 태스크 정의에 지정되어 있습니다. 서비스 작업이 지정하는 태스크 정의가 bridge 또는 host 네트워크 모드를 사용하는 경우, 태스크 정의에서 containerName 및 containerPort 조합을 지정해야 합니다. 서비스 작업이 지정하는 태스크 정의가 awsvpc 네트워크 모드를 사용하고 유형 SRV DNS 레코드가 사용되는 경우, containerName 및 containerPort 조합 또는 port 값 중 하나를 지정해야 합니다.

containerPort

유형: 정수

필수 여부: 아니요

서비스 검색 서비스에 사용할 포트 값입니다. 이 값은 태스크 정의에 지정되어 있습니다. 서비스 작업이 지정하는 태스크 정의가 bridge 또는 host 네트워크 모드를 사용하는 경우, 태스크 정의에서 containerName 및 containerPort 조합을 지정해야 합니다. 서비스 작업이 지정하는 태스크 정의가 awsvpc 네트워크 모드를 사용하고 유형 SRV DNS 레코드가 사용되는 경우, containerName 및 containerPort 조합 또는 port 값 중 하나를 지정해야 합니다.

클라이언트 토큰

clientToken

유형: 문자열

필수 여부: 아니요

요청 멱등성을 보장하기 위해 제공하는 고유한 대소문자 구분 식별자입니다. 최대 32자의 ASCII 문자를 사용할 수 있습니다.

가용 영역 재조정

availabilityZoneRebalancing

유형: 문자열

필수 여부: 아니요

서비스가 가용 영역 리밸런싱을 사용하는지 여부를 나타냅니다. 유효한 값은 ENABLED 및 입니다DISABLED. 가용 영역 리밸런싱에 대한 자세한 내용은 가용 영역 전체의 Amazon ECS 서비스 밸런싱을 참조하세요.

볼륨 구성

`volumeConfigurations`

유형: 객체

필수 여부: 아니요

서비스에서 관리하는 작업에 사용할 볼륨을 생성하는 데 사용되는 구성입니다. 작업 정의에 `configuredAtLaunch`로 표시된 볼륨만 이 객체를 사용하여 구성할 수 있습니다.

이 객체는 Amazon EBS 볼륨을 서비스에서 관리하는 작업에 연결하는 데 필요합니다. 자세한 내용은 Amazon ECS에서 Amazon EBS 볼륨 사용 섹션을 참조하세요.

`name`

유형: 문자열

필수 항목 여부: 예

서비스를 생성하거나 업데이트할 때 구성된 볼륨의 이름입니다. 최대 255자의 문자(대문자 및 소문자), 숫자, 밑줄(_) 및 하이픈(-)이 허용됩니다. 이 값은 작업 정의에 지정된 볼륨 이름과 일치해야 합니다.

`managedEBSVolume`

유형: 객체

필수 여부: 아니요

서비스가 생성 또는 업데이트될 때 서비스에서 유지 관리하는 태스크에 연결되는 Amazon EBS 볼륨을 생성하는 데 사용되는 볼륨 구성입니다. 태스크당 하나의 볼륨이 연결됩니다.

`encrypted`

유형: 부울

필수 여부: 아니요

유효한 값: `true|false`

생성된 각 Amazon EBS 볼륨을 암호화할지 여부를 지정합니다. 특정 AWS 리전의 AWS 계정에 대해 기본적으로 Amazon EBS 암호화를 활성화했지만 이 파라미터를 `false`로 설정

한 경우 이 파라미터는 재정의되고 볼륨은 기본적으로 암호화에 지정된 KMS 키로 암호화됩니다. 기본적으로 Amazon EBS 암호화에 대한 자세한 내용은 Amazon EBS 사용 설명서의 기본적으로 Amazon EBS 암호화 활성화를 참조하세요. Amazon ECS 태스크에 연결된 Amazon EBS 볼륨 암호화에 대한 자세한 내용은 Amazon ECS 태스크에 연결된 Amazon EBS 볼륨에 저장된 데이터 암호화 섹션을 참조하세요.

`kmsKeyId`

유형: 문자열

필수 여부: 아니요

Amazon EBS 암호화에 사용할 AWS Key Management Service(AWS KMS) 키의 식별자입니다. `kmsKeyId`가 지정되면 암호화된 상태가 `true`여야 합니다.

이 파라미터를 사용하여 지정한 키는 Amazon EBS 기본값 또는 사용자가 Amazon ECS 관리형 스토리지 암호화에 대해 지정했을 수 있는 클러스터 수준 KMS 키를 재정의합니다. 자세한 내용은 Amazon ECS 태스크에 연결된 Amazon EBS 볼륨에 저장된 데이터 암호화 섹션을 참조하세요.

다음 중 하나를 사용하여 KMS 키를 지정할 수 있습니다.

- 키 ID - 예: `1234abcd-12ab-34cd-56ef-1234567890ab`.
- 키 별칭 - 예: `alias/ExampleAlias`.
- 키 ARN - 예: `arn:aws:kms:us-east-1:012345678910:key/1234abcd-12ab-34cd-56ef-1234567890ab`.
- 별칭 ARN - 예: `arn:aws:kms:us-east-1:012345678910:alias/ExampleAlias`.

> **Important**
>
> AWS에서는 KMS 키를 비동기적으로 인증합니다. 따라서 유효하지 않은 ID, 별칭 또는 ARN을 지정하면 작업이 완료된 것처럼 보여도 실제로 실패합니다. 자세한 내용은 Amazon EBS 볼륨 연결 문제 해결을 참조하세요.

`volumeType`

유형: 문자열

필수 여부: 아니요

유효한 값: gp2|gp3|io1|io2|sc1|st1|standard

Amazon EBS 볼륨 유형입니다. 볼륨 유형에 대한 자세한 내용은 Amazon EBS 사용 설명서의 Amazon EBS 볼륨 유형을 참조하세요. 기본 볼륨 유형은 gp3입니다.

> **Note**
>
> Fargate 태스크에는 standard 볼륨 유형이 지원되지 않습니다.

`sizeInGiB`

유형: 정수

필수 여부: 아니요

유효한 범위: 1~16,384의 정수

기비바이트(GiB) 단위의 EBS 볼륨 크기입니다. 연결할 볼륨을 구성하기 위한 스냅샷 ID를 제공하지 않는 경우 크기 값을 제공해야 합니다. 스냅샷을 사용하여 연결할 볼륨을 구성하는 경우 기본값은 스냅샷 크기입니다. 스냅샷 크기 이상인 크기를 지정할 수 있습니다.

gp2 및 gp3 볼륨 유형의 경우 유효한 범위는 1~16,384입니다.

io1 및 io2 볼륨 유형의 경우 유효한 범위는 4~16,384입니다.

st1 및 sc1 볼륨 유형의 경우 유효한 범위는 125~16,384입니다.

standard 볼륨 유형의 경우 유효한 범위는 1~1,024입니다.

`snapshotId`

유형: 문자열

필수 여부: 아니요

Amazon ECS가 연결할 새 볼륨을 생성하는 데 사용하는 기존 Amazon EBS 볼륨의 스냅샷 ID입니다. snapshotId 또는 sizeInGiB를 지정해야 합니다.

`volumeInitializationRate`

유형: 정수

필수 여부: 아니요

연결할 새 볼륨을 생성하기 위해 기존 Amazon EBS 볼륨의 스냅샷에서 데이터를 가져오는 속도(MiB/s 단위)입니다. `snapshotId`를 지정한 경우에만 이 속성을 지정할 수 있습니다. 지원되는 초기화 속도 범위를 포함하여 이 볼륨 초기화 속도에 대한 자세한 내용은 Amazon EBS 사용 설명서의 Amazon EBS 볼륨 초기화를 참조하세요.

`iops`

유형: 정수

필수 여부: 아니요

초당 I/O 작업 수(IOPS) 입니다. `gp3`, `io1` 및 `io2` 볼륨의 경우 이 값은 해당 볼륨에 프로비저닝되는 IOPS의 개수를 나타냅니다. `gp2` 볼륨의 경우 이 값은 볼륨이 버스트에서 I/O 크레딧을 축적하는 볼륨과 속도에 대한 기준 성능을 나타냅니다. 이 파라미터는 `io1` 및 `io2` 볼륨에 필요합니다. `gp2`, `st1`, `sc1` 또는 `standard` 볼륨에서 이 파라미터는 지원되지 않습니다.

`gp3` 볼륨의 경우 유효한 값의 범위는 3,000~16,000입니다.

`io1` 볼륨의 경우 유효한 값의 범위는 100~64,000입니다.

`io2` 볼륨의 경우 유효한 값의 범위는 100~64,000입니다.

`throughput`

유형: 정수

필수 여부: 아니요

서비스에서 유지 관리하는 태스크에 연결된 볼륨을 프로비저닝하기 위한 처리량입니다.

> **Important**
> 이 파라미터는 `gp3` 볼륨에서만 지원됩니다.

`roleArn`

유형: 문자열

필수 항목 여부: 예

작업에 대한 Amazon EBS 리소스를 관리할 수 있는 권한을 Amazon ECS에 제공하는 인프라 AWS Identity and Access Management(IAM) 역할의 Amazon 리소스 ARN입니다. 자세한 내용은 Amazon ECS 인프라 IAM 역할 섹션을 참조하세요.

`tagSpecifications`

유형: 객체

필수 여부: 아니요

각 Amazon EBS 볼륨에 적용할 태그의 사양입니다.

`resourceType`

유형: 문자열

필수 항목 여부: 예

유효값: `volume`

생성 시 태그를 지정할 리소스 유형입니다.

`tags`

타입: 객체 배열

필수 여부: 아니요

작업을 분류하고 구성하는 데 도움이 되도록 작업에 적용하는 메타데이터입니다. 각 태그는 모두 사용자가 정의하는 키 및 선택적 값으로 구성됩니다. `AmazonECSCreated` 및 `AmazonECSManaged`는 Amazon ECS에서 사용자를 대신하여 추가한 예약된 태그이므로, 사용자는 최대 48개의 고유한 태그를 지정할 수 있습니다. 볼륨이 삭제되면 태그도 삭제됩니다. 자세한 내용은 Amazon ECS 리소스 태그 지정 섹션을 참조하세요.

`key`

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 128입니다.

필수 여부: 아니요

하나의 태그를 구성하는 키-값 쌍의 일부분. 키는 더 구체적인 태그 값에 대해 범주와 같은 역할을 하는 일반적인 레이블입니다.

value

유형: 문자열

길이 제약: 최소 길이는 0. 최대 길이 256.

필수 여부: 아니요

하나의 태그를 구성하는 키-값 쌍의 선택적 부분. 하나의 값은 태그 범주(키) 내에서 서술자 역할을 수행합니다.

propagateTags

유형: 문자열

유효한 값: TASK_DEFINITION | SERVICE | NONE

필수 여부: 아니요

태그를 작업 정의 또는 서비스에서 볼륨으로 복사할지 여부를 지정합니다. NONE이 지정되었거나 값이 지정되지 않은 경우 태그는 복사되지 않습니다.

fileSystemType

유형: 문자열

필수 여부: 아니요

유효한 값: xfs|ext3|ext4|NTFS

볼륨의 파일 시스템 유형입니다. 볼륨의 파일 시스템 유형에 따라 볼륨에서 데이터를 저장 및 검색하는 방법이 결정됩니다. 스냅샷에서 생성된 볼륨의 경우 스냅샷을 생성할 때 볼륨에서 사용하던 것과 동일한 파일 시스템 유형을 지정해야 합니다. 파일 시스템 유형이 일치하지 않으면 작업이 시작되지 않습니다.

Linux의 유효한 값은 xfs, ext3, and ext4입니다. Linux 태스크에 연결된 볼륨의 기본값은 XFS입니다.

Windows의 유효한 값은 NTFS입니다. Windows 태스크에 연결된 볼륨의 기본값은 NTFS입니다.

서비스 정의 템플릿

다음은 Amazon ECS 서비스 정의의 JSON 표현을 보여줍니다.

Amazon EC2 시작 유형

```
{
    "cluster": "",
    "serviceName": "",
    "taskDefinition": "",
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "loadBalancerName": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "desiredCount": 0,
    "clientToken": "",
    "launchType": "EC2",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
            "weight": 0,
            "base": 0
        }
    ],
    "platformVersion": "",
    "role": "",
    "deploymentConfiguration": {
        "deploymentCircuitBreaker": {
            "enable": true,
            "rollback": true
        },
        "maximumPercent": 0,
        "minimumHealthyPercent": 0,
        "alarms": {
            "alarmNames": [
                ""
```

```
            ],
            "enable": true,
            "rollback": true
        }
    },
    "placementConstraints": [
        {
            "type": "distinctInstance",
            "expression": ""
        }
    ],
    "placementStrategy": [
        {
            "type": "binpack",
            "field": ""
        }
    ],
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    },
    "healthCheckGracePeriodSeconds": 0,
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "EXTERNAL"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "TASK_DEFINITION",
    "enableExecuteCommand": true,
    "availabilityZoneRebalancing": "ENABLED",
    "serviceConnectConfiguration": {
```

```
        "enabled": true,
        "namespace": "",
        "services": [
            {
                "portName": "",
                "discoveryName": "",
                "clientAliases": [
                    {
                        "port": 0,
                        "dnsName": ""
                    }
                ],
                "ingressPortOverride": 0
            }
        ],
        "logConfiguration": {
            "logDriver": "journald",
            "options": {
                "KeyName": ""
            },
            "secretOptions": [
                {
                    "name": "",
                    "valueFrom": ""
                }
            ]
        }
    },
    "volumeConfigurations": [
        {
            "name": "",
            "managedEBSVolume": {
                "encrypted": true,
                "kmsKeyId": "",
                "volumeType": "",
                "sizeInGiB": 0,
                "snapshotId": "",
                "volumeInitializationRate": 0,
                "iops": 0,
                "throughput": 0,
                "tagSpecifications": [
                    {
                        "resourceType": "volume",
                        "tags": [
```

```
                            {
                                "key": "",
                                "value": ""
                            }
                        ],
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn": "",
                "filesystemType": ""
            }
        }
    ]
}
```

Fargate 시작 유형

```
{
    "cluster": "",
    "serviceName": "",
    "taskDefinition": "",
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "loadBalancerName": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "desiredCount": 0,
    "clientToken": "",
    "launchType": "FARGATE",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
```

```
            "weight": 0,
            "base": 0
        }
    ],
    "platformVersion": "",
    "platformFamily": "",
    "role": "",
    "deploymentConfiguration": {
        "deploymentCircuitBreaker": {
            "enable": true,
            "rollback": true
        },
        "maximumPercent": 0,
        "minimumHealthyPercent": 0,
        "alarms": {
            "alarmNames": [
                ""
            ],
            "enable": true,
            "rollback": true
        }
    },
    "placementStrategy": [
        {
            "type": "binpack",
            "field": ""
        }
    ],
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    },
    "healthCheckGracePeriodSeconds": 0,
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "EXTERNAL"
    },
```

```
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "TASK_DEFINITION",
    "enableExecuteCommand": true,
    "availabilityZoneRebalancing": "ENABLED",
    "serviceConnectConfiguration": {
        "enabled": true,
        "namespace": "",
        "services": [
            {
                "portName": "",
                "discoveryName": "",
                "clientAliases": [
                    {
                        "port": 0,
                        "dnsName": ""
                    }
                ],
                "ingressPortOverride": 0
            }
        ],
        "logConfiguration": {
            "logDriver": "journald",
            "options": {
                "KeyName": ""
            },
            "secretOptions": [
                {
                    "name": "",
                    "valueFrom": ""
                }
            ]
        }
    },
    "volumeConfigurations": [
        {
            "name": "",
            "managedEBSVolume": {
                "encrypted": true,
```

```
                "kmsKeyId": "",
                "volumeType": "",
                "sizeInGiB": 0,
                "snapshotId": "",
                "volumeInitializationRate": 0,
                "iops": 0,
                "throughput": 0,
                "tagSpecifications": [
                    {
                        "resourceType": "volume",
                        "tags": [
                            {
                                "key": "",
                                "value": ""
                            }
                        ],
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn": "",
                "filesystemType": ""
            }
        }
    ]
}
```

다음 AWS CLI 명령을 사용하여 이 서비스 정의 템플릿을 생성할 수 있습니다.

```
aws ecs create-service --generate-cli-skeleton
```

Amazon ECS 리소스 태그 지정

Amazon ECS 리소스 관리를 지원하려면 태그를 사용하여 각 리소스에 고유한 메타데이터를 선택적으로 할당할 수 있습니다. 각 태그는 키와 값(선택 사항)으로 구성됩니다.

태그를 사용하여 Amazon ECS 리소스를 용도, 소유자, 환경 등으로 다양하게 분류할 수 있습니다. 이는 동일한 유형의 리소스가 많을 때 유용합니다. 지정한 태그를 기반으로 특정 리소스를 신속하게 식별할 수 있습니다. 예를 들어, 계정의 Amazon ECS 컨테이너 인스턴스에 대해 태그 세트를 정의할 수 있습니다. 이렇게 하면 각 인스턴스의 소유자와 스택 수준을 추적하는 데 도움이 됩니다.

비용 및 사용량 보고서에 태그를 사용할 수 있습니다. 이 보고서를 사용하면 Amazon ECS 리소스의 비용 및 사용량을 분석할 수 있습니다. 자세한 내용은 the section called “사용 보고서” 섹션을 참조하세요.

> **Warning**
>
> 태그 키 및 해당 값을 반환하는 여러 API가 있습니다. `DescribeTags`에 대한 액세스를 거부해도 다른 API가 반환한 태그에 대한 액세스는 자동으로 거부되지 않습니다. 민감한 데이터를 태그에 포함하지 않는 것이 가장 좋습니다.

각 리소스 유형에 대한 요건을 충족하는 태그 키 세트를 고안하는 것이 좋습니다. 일관된 태그 키 세트를 사용하면 리소스를 보다 쉽게 관리할 수 있습니다. 추가하는 태그에 따라 리소스를 검색하고 필터링할 수 있습니다.

태그는 Amazon ECS에는 의미가 없으며 엄격하게 문자열로 해석됩니다. 태그 키와 값을 편집할 수 있으며 언제든지 리소스에서 태그를 제거할 수 있습니다. 태그의 값을 빈 문자열로 설정할 수 있지만 태그의 값을 Null로 설정할 수는 없습니다. 해당 리소스에 대해 키가 기존 태그와 동일한 태그를 추가하는 경우 새 값이 이전 값을 덮어씁니다. 리소스를 삭제하면 리소스 태그도 함께 삭제됩니다.

AWS Identity and Access Management(IAM)를 사용하는 경우 AWS 계정에서 태그를 관리할 수 있는 권한을 가진 사용자를 제어할 수 있습니다.

리소스 태그 지정 방법

Amazon ECS 작업, 서비스, 작업 정의, 클러스터에 태그를 지정하는 방법은 여러 가지입니다.

- 사용자는 AWS Management Console, Amazon ECS API, AWS CLI 또는 AWS SDK를 사용하여 리소스에 수동으로 태그를 지정합니다.

- 사용자가 서비스를 생성하거나 독립 실행형 작업을 실행하고 Amazon ECS 관리형 태그 옵션을 선택합니다.

  Amazon ECS는 새로 시작된 모든 작업에 자동으로 태그를 지정합니다. 자세한 내용은 the section called “Amazon ECS 관리형 태그” 섹션을 참조하세요.
- 사용자는 콘솔을 사용하여 리소스를 생성합니다. 콘솔은 자동으로 리소스에 태그를 지정합니다.

  이러한 태그는 AWS CLI 및 AWS SDK 응답에 반환되고 콘솔에 표시됩니다. 이러한 태그는 수정하거나 삭제할 수 없습니다.

  추가된 태그에 대한 자세한 내용은 Amazon ECS 리소스 태그 지정 지원 테이블의 콘솔에서 자동으로 추가된 태그를 참조하세요.

리소스를 생성할 때 태그를 지정했는데 태그를 적용할 수 없는 경우 Amazon ECS가 생성 프로세스를 롤백합니다. 이는 태그를 사용하여 리소스가 생성되거나 아예 리소스가 생성되지 않도록 하고 언제든 태그 지정되지 않은 리소스가 남지 않게 합니다. 생성하는 동안 리소스에 태그를 지정하면 리소스 생성 후 사용자 지정 태그 지정 스크립트를 실행할 필요가 없습니다.

다음 표에는 태그 지정을 지원하는 Amazon ECS 리소스가 설명되어 있습니다.

리소스	태그 지원	태그 전달 지원	콘솔에서 자동으로 추가된 태그
Amazon ECS 작업	예	예, 태스크 정의에서	키: `aws:ecs:clusterName` 값: `cluster-name`
Amazon ECS 서비스	예	예, 태스크 정의 또는 서비스에서 서비스의 작업으로	키: `ecs:service:stackId` 값 `arn:aws:cloudformation: arn`
Amazon ECS 작업 세트	예	아니요	N/A

리소스	태그 지원	태그 전달 지원	콘솔에서 자동으로 추가된 태그
Amazon ECS 태스크 정의	예	아니요	키: `ecs:taskDefinition:createdFrom` 값: `ecs-console-v2`
Amazon ECS 클러스터	예	아니요	키: `aws:cloudformation:logical-id` 값: `ECSCluster` 키: `aws:cloudformation:stack-id` 값: `arn:aws:cloudformation:` *`arn`* 키: `aws:cloudformation:stack-name` 값: `ECS-Console-V2-Cluster-` *`EXAMPLE`*
Amazon ECS 컨테이너 인스턴스	예	예, Amazon EC2 인스턴스에서 자세한 내용은 Amazon ECS 컨테이너 인스턴스에 태그 추가 섹션을 참조하세요.	N/A

리소스	태그 지원	태그 전달 지원	콘솔에서 자동으로 추가된 태그
Amazon ECS 외부 인스턴스	예	아니요	N/A
Amazon ECS 용량 공급자	예. 사전 정의된 FARGATE 및 FARGATE_SPOT 용량 공급자에는 태그를 지정할 수 없습니다.	아니요	N/A

생성 시 리소스 태그 지정

다음 리소스는 Amazon ECS API, AWS CLI 또는 AWS SDK를 사용하여 생성 시 태그 지정을 지원합니다.

- Amazon ECS 작업
- Amazon ECS 서비스
- Amazon ECS 작업 정의
- Amazon ECS 작업 세트
- Amazon ECS 클러스터
- Amazon ECS 컨테이너 인스턴스
- Amazon ECS 용량 공급자 생성

Amazon ECS에는 리소스 생성에 대해 태그 지정 권한 부여를 사용하는 옵션이 있습니다. AWS 계정에 태그 지정 권한이 있도록 구성된 경우 사용자에게는 리소스를 생성하는 작업을 위한 권한(예: `ecsCreateCluster`)이 있어야 합니다. 리소스 생성 작업에서 태그를 지정하면 AWS는 추가 권한 부여를 수행하여 사용자 또는 역할에 태그를 생성할 권한이 있는지 확인합니다. 따라서 `ecs:TagResource` 작업을 사용할 수 있는 명시적 권한을 부여해야 합니다. 자세한 내용은 the section called “생성 시 리소스에 태그 지정” 섹션을 참조하세요. 옵션을 구성하는 방법에 대한 자세한 내용은 the section called “태그 지정 권한 부여” 섹션을 참조하세요.

제한 사항

태그에 적용되는 제한은 다음과 같습니다.

- 최대 50개의 태그를 리소스와 연결할 수 있습니다.
- 한 리소스에 대해 태그 키를 반복할 수 없습니다. 각 태그 키는 고유해야 하며 하나의 값만 가질 수 있습니다.
- 키는 최대 128자(UTF-8 형식)까지 가능합니다.
- 값은 최대 256자(UTF-8 형식)까지 가능합니다.
- 여러 AWS 서비스 및 리소스에서 태그 지정 스키마를 사용하는 경우 사용하는 문자 유형을 제한합니다. 일부 서비스는 허용되는 문자에 제한이 있을 수 있습니다. 일반적으로 허용되는 문자는 문자, 숫자, 공백 및 특수 문자 **+ - = . _ : / @**입니다.
- 태그 키와 값은 대소문자를 구분합니다.
- 키 또는 값에 `aws:`, `AWS:` 등 접두사의 대문자 또는 소문자 조합을 사용할 수 없습니다. 이러한 이름은 AWS 전용으로 예약되어 있습니다. 이 접두사가 지정된 태그 키나 값은 편집하거나 삭제할 수 없습니다. 이 접두사가 지정된 태그는 리소스당 태그 수 제한에 포함되지 않습니다.

Amazon ECS 관리형 태그

Amazon ECS 관리형 태그를 사용하는 경우 Amazon ECS는 클러스터 정보와 사용자가 추가한 작업 정의 태그 또는 서비스 태그를 사용하여 새로 시작된 모든 작업 및 작업에 연결된 모든 Amazon EBS 볼륨에 자동으로 태그를 지정합니다. 다음은 추가된 태그에 대한 설명입니다.

- 독립 실행형 작업 - 키가 `aws:ecs:clusterName`, 값이 클러스터 이름으로 설정된 태그. 사용자가 추가한 모든 작업 정의 태그. 독립 실행형 작업에 연결된 Amazon EBS 볼륨은 키를 `aws:ecs:clusterName`, 값을 클러스터 이름으로 설정한 태그를 수신합니다. Amazon EBS 볼륨 태그 지정에 대한 자세한 내용은 Amazon EBS 볼륨 태그 지정을 참조하세요.
- 서비스의 일부인 작업 - 키가 `aws:ecs:clusterName`, 값이 클러스터 이름으로 설정된 태그. 키가 `aws:ecs:serviceName`, 값이 서비스 이름으로 설정된 태그. 다음 리소스 중 하나의 태그:
  - 작업 정의 - 사용자가 추가한 모든 작업 정의 태그.
  - 서비스 - 사용자가 추가한 모든 서비스 태그.

    서비스의 일부인 작업에 연결된 Amazon EBS 볼륨은 키를 `aws:ecs:clusterName`, 값을 클러스터 이름으로 설정한 태그 및 키를 `aws:ecs:serviceName`, 값을 서비스 이름으로 설정한 태

그를 수신합니다. Amazon EBS 볼륨 태그 지정에 대한 자세한 내용은 Amazon EBS 볼륨 태그 지정을 참조하세요.

이 기능에서는 다음 옵션이 필요합니다.

- 새로운 Amazon 리소스 이름(ARN) 및 리소스 식별자(ID) 형식을 옵트인해야 합니다. 자세한 내용은 Amazon 리소스 이름(ARN) 및 ID 섹션을 참조하세요.
- API를 사용하여 서비스를 생성하거나 작업을 실행할 때 `run-task` 및 `create-service`에서 `true`에 대한 `enableECSManagedTags`을(를) 설정해야 합니다. 자세한 내용은 AWS Command Line Interface API 참조에서 create-service 및 run-task를 참조하세요.
- Amazon ECS는 관리형 태그를 사용하여 클러스터 Auto Scaling과 같은 일부 기능이 활성화되는 시기를 결정합니다. Amazon ECS에서 기능을 효율적으로 관리할 수 있도록 태그를 수동으로 수정하지 않는 것이 좋습니다.

결제에 태그 사용

AWS는 Amazon ECS 리소스의 비용 및 사용량을 분석할 수 있는 Cost Explorer라는 보고 도구를 제공합니다.

Cost Explorer를 사용하여 사용량 및 비용 차트를 볼 수 있습니다. 지난 13개월의 데이터를 볼 수 있으며 향후 3개월 동안의 지출을 예상해볼 수 있습니다. Cost Explorer를 사용하여 시간의 경과에 따른 AWS 리소스의 지출 패턴을 확인할 수 있습니다. 예를 들어 Cost Explorer를 사용하여 추가 조사가 필요한 영역을 알아내고, 비용 이해에 사용할 수 있는 추세를 파악할 수 있습니다. 또한 데이터의 시간 범위를 지정하고 일별 또는 월별 시간 데이터를 볼 수도 있습니다.

Amazon ECS 관리형 태그 또는 사용자 추가 태그를 비용 및 사용량 보고서에 사용할 수 있습니다. 자세한 내용은 Amazon ECS 사용 보고서 섹션을 참조하세요.

결합된 리소스의 비용을 확인하려면 태그 키 값을 동일한 리소스에 따라 결제 정보를 구성할 수 있습니다. 예를 들어, 특정 애플리케이션 이름으로 여러 리소스에 태그를 지정한 다음 결제 정보를 구성하여 여러 서비스에 걸친 해당 애플리케이션의 총 비용을 볼 수 있습니다. 태그를 사용한 비용 할당 보고서 설정에 대한 자세한 내용은 AWS Billing 사용 설명서에서 월간 비용 할당 보고서를 참조하세요.

또한 비용 할당 데이터 분할을 켜서 비용 및 사용 보고서에서 작업 수준의 CPU 및 메모리 사용 데이터를 얻을 수 있습니다. 자세한 내용은 작업 수준 비용 및 사용 보고서 섹션을 참조하세요.

> **Note**
>
> 보고를 켠 경우, 최대 24시간 후에 이번 달의 데이터를 볼 수 있습니다.

Amazon ECS 리소스에 태그 추가

새 태스크, 기존 태스크, 서비스, 태스크 정의 또는 클러스터에 태그를 지정할 수 있습니다. 컨테이너 인스턴스 태그 지정에 대한 자세한 내용은 Amazon ECS 컨테이너 인스턴스에 태그 추가 섹션을 참조하세요.

> **Warning**
>
> 개인 식별 정보(PII)나 기타 기밀 정보 또는 민감한 정보를 태그에 추가하지 않습니다. 청구를 비롯한 여러 AWS 서비스에서 태그에 액세스할 수 있습니다. 태그는 개인 데이터나 민감한 데이터에 사용하기 위한 것이 아닙니다.

다음의 리소스를 사용하면 리소스를 생성할 때 태그를 지정할 수 있습니다.

Task	콘솔	AWS CLI	API 작업
하나 이상의 태스크를 실행.	애플리케이션을 Amazon ECS 태스크로 실행	run-task	RunTask
서비스를 생성합니다.	Amazon ECS 롤링 업데이트 배포 생성	create-service	CreateService
작업 세트 생성.	타사 컨트롤러를 사용한 Amazon ECS 서비스 배포	create-task-set	CreateTaskSet
태스크 정의 등록.	the section called “콘솔을 사용하여 작업 정의 생성”	register-task-definition	RegisterTaskDefinition

Task	콘솔	AWS CLI	API 작업
클러스터를 생성합니다.	Fargate 시작 유형에 대한 Amazon ECS 클러스터 생성	create-cluster	CreateCluster
하나 이상의 컨테이너 인스턴스를 실행.	Amazon ECS Linux 컨테이너 인스턴스 시작	run-instances	RunInstances

기존 리소스에 태그 추가(Amazon ECS 콘솔)

리소스 페이지에서 클러스터, 서비스, 작업 및 작업 정의와 연결된 태그를 추가하거나 삭제할 수 있습니다.

개별 리소스에 대한 태그 수정

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 모음에서 사용할 AWS 리전(을)를 선택합니다.
3. 탐색 창에서 리소스 유형(예: 클러스터)을 선택합니다.
4. 리소스 목록에서 리소스를 선택하고, 태그 탭을 선택한 다음 태그 관리를 선택합니다.
5. 태그를 구성합니다.

   [태그 추가] 태그 추가를 선택하고 다음을 수행합니다.

   - 키에서 키 이름을 입력합니다.
   - 값에 키 값을 입력합니다.
6. 저장을 선택합니다.

기존 리소스에 태그 추가(AWS CLI)

AWS CLI 또는 API를 사용하여 하나 이상의 태그를 추가하거나 덮어쓸 수 있습니다.

> **Note**
>
> 듀얼 스택 서비스 엔드포인트를 사용하면 AWS CLI, SDK 및 Amazon ECS API에서 IPv4 및 IPv6 모두를 통해 Amazon ECS와 상호 작용할 수 있습니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

- AWS CLI - tag-resource
- API 작업 - TagResource

Amazon ECS 컨테이너 인스턴스에 태그 추가

다음 방법 중 하나를 사용하면 태그를 컨테이너 인스턴스와 연결할 수 있습니다.

- 방법 1 – Amazon EC2 API, CLI 또는 콘솔을 사용하여 컨테이너 인스턴스를 생성할 때 컨테이너 에이전트 구성 파라미터 `ECS_CONTAINER_INSTANCE_TAGS`를 사용하여 사용자 데이터를 인스턴스에 전달해 태그를 지정합니다. 이렇게 하면 오로지 Amazon ECS의 컨테이너 인스턴스와 연결된 태그가 생성되며, 이러한 태그는 Amazon EC2 API를 사용하여 볼 수 없습니다. 자세한 내용은 데이터 전달을 위한 Amazon ECS Linux 컨테이너 인스턴스 부트스트래핑 섹션을 참조하세요.

  > **Important**
  >
  > Amazon EC2 Auto Scaling 그룹을 사용하여 컨테이너 인스턴스를 시작한 경우 ECS_CONTAINER_INSTANCE_TAGS 에이전트 구성 파라미터를 사용하여 태그를 추가해야 합니다. 이는 Auto Scaling 그룹을 사용하여 시작된 Amazon EC2 인스턴스에 태그가 추가되는 방식 때문입니다.

  태그를 컨테이너 인스턴스와 연결하는 사용자 데이터 스크립트를 예시하면 다음과 같습니다.

  ```
  #!/bin/bash
  cat <<'EOF' >> /etc/ecs/ecs.config
  ECS_CLUSTER=MyCluster
  ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
  EOF
  ```

- 방법 2 – Amazon EC2 API, CLI 또는 콘솔을 사용하여 컨테이너 인스턴스를 생성하려면 먼저 `TagSpecification.N` 파라미터를 사용하여 태그를 지정해야 합니다. 그런 다음, 컨테이너 에이전

트 구성 파라미터 ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM을 사용하여 사용자 데이터를 인스턴스에 전달합니다. 이렇게 하면 Amazon EC2에서 Amazon ECS로 전파됩니다.

Amazon EC2 인스턴스와 연결된 태그를 전파하고 MyCluster라고 하는 클러스터에 인스턴스를 등록하는 사용자 데이터 스크립트의 예는 다음과 같습니다.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
EOF
```

컨테이너 인스턴스 태그가 Amazon EC2에서 Amazon ECS로 전파되도록 액세스를 제공하려면 Amazon ECS 컨테이너 인스턴스 IAM 역할에 대해 다음 권한을 인라인 정책으로 직접 추가합니다. 자세한 내용은 IAM 정책 추가 및 제거 섹션을 참조하세요.

- ec2:DescribeTags

다음은 이러한 권한을 추가하는 데 사용되는 정책 예제입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
             "ec2:DescribeTags"
            ],
            "Resource": "*"
        }
    ]
}
```

외부 컨테이너 인스턴스

다음 방법 중 하나를 사용하면 태그를 외부 컨테이너 인스턴스와 연결할 수 있습니다.

- 방법 1 — 외부 인스턴스를 클러스터에 등록하기 위해 설치 스크립트를 실행하기 전에 /etc/ecs/ecs.config에 Amazon ECS 컨테이너 에이전트 구성 파일을 생성하거나 편집하고

`ECS_CONTAINER_INSTANCE_TAGS` 컨테이너 에이전트 구성 파라미터를 추가합니다. 이렇게 하면 외부 인스턴스와 연결된 태그가 생성됩니다.

다음은 구문의 예제입니다.

```
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
```

- 방법 2 — 외부 인스턴스가 클러스터에 등록되면 AWS Management Console을 사용해 태그를 추가합니다. 자세한 내용은 기존 리소스에 태그 추가(Amazon ECS 콘솔) 섹션을 참조하세요.

Amazon ECS 사용 보고서

AWS는 Amazon ECS 리소스의 비용 및 사용량을 분석할 수 있는 Cost Explorer라는 보고 도구를 제공합니다.

Cost Explorer를 사용하여 사용량 및 비용 차트를 볼 수 있습니다. 지난 13개월의 데이터를 볼 수 있으며 향후 3개월 동안의 지출을 예상해볼 수 있습니다. Cost Explorer를 사용하여 시간의 경과에 따른 AWS 리소스의 지출 패턴을 확인할 수 있습니다. 예를 들어 Cost Explorer를 사용하여 추가 조사가 필요한 영역을 알아내고, 비용 이해에 사용할 수 있는 추세를 파악할 수 있습니다. 또한 데이터의 시간 범위를 지정하고 일별 또는 월별 시간 데이터를 볼 수도 있습니다.

비용 및 사용 보고서의 측정 데이터는 모든 Amazon ECS 작업에서의 사용량을 보여줍니다. 측정 데이터에는 실행된 각 작업에 대한 CPU 사용량이 `vCPU-Hours`로 포함되고 메모리 사용량이 `GB-Hours`로 포함됩니다. 데이터가 표시되는 방식은 작업의 시작 유형에 의해 결정됩니다.

Fargate 시작 유형을 사용하는 작업의 경우, `lineItem/Operation` 열에 `FargateTask`가 표시되고 각 태스크와 연관된 비용이 표시됩니다.

EC2 시작 유형을 사용하는 작업의 경우 `lineItem/Operation` 열에 `ECSTask-EC2`가 표시되고 태스크와 연관된 직접 비용은 표시되지 않습니다. 보고서에 표시된 측정 데이터(메모리 사용량 등)은 표시된 청구 기간 동안 태스크에 예약된 총 리소스를 나타냅니다. 이 데이터를 사용하여 Amazon EC2 인스턴스의 기본 클러스터 비용을 결정하는 데 사용할 수 있습니다. Amazon EC2 인스턴스에 대한 비용 및 사용량 데이터는 Amazon EC2 서비스 아래에 별도로 나열됩니다.

Amazon ECS 관리형 태그를 사용하면 각 작업이 속한 서비스 또는 클러스터를 식별할 수도 있습니다. 자세한 내용은 결제에 태그 사용 섹션을 참조하세요.

> **Important**
>
> 측정 데이터는 2018년 11월 16일 또는 그 이후에 시작된 작업에 대해서만 볼 수 있습니다. 이 날짜 이전에 시작된 작업에는 측정 데이터가 표시되지 않습니다.

Cost Explorer에서 사용자가 비용 할당 데이터를 정렬할 수 있는 필드 중 일부를 예로 들면 다음과 같습니다.

- 클러스터 이름
- 서비스 이름
- 리소스 태그
- 시작 유형
- AWS 리전
- 사용 유형

AWS 비용 및 사용 보고서 생성에 대한 자세한 내용은 AWS Billing 사용 설명서의 AWS 비용 및 사용 보고서를 참조하세요.

작업 수준 비용 및 사용 보고서

AWS Cost Management에서의 Fargate의 작업 및 EC2의 작업을 비롯한 Amazon ECS의 각 작업에 대한 CPU 및 메모리 사용 데이터를 AWS Cost and Usage Report에서 제공할 수 있습니다. 이 데이터를 비용 할당 데이터 분할이라고 합니다. 이 데이터를 사용하여 애플리케이션의 비용 및 사용량을 분석할 수 있습니다. 또한 비용 할당 태그와 비용 범주를 사용하여 개별 사업부 및 팀으로 비용을 분할하고 할당할 수 있습니다. 비용 할당 데이터 분할에 대한 자세한 내용은 AWS Cost and Usage Report 사용 설명서의 Understanding split cost allocation data를 참조하세요.

AWS Cost Management Console에서 계정에 대해 작업 수준 비용 할당 데이터 분할을 옵트인할 수 있습니다. 관리(지급인) 계정이 있는 경우 지급인 계정에서 옵트인하여 모든 연결 계정에 이 구성을 적용할 수 있습니다.

비용 할당 데이터 분할을 설정하면 보고서에서 splitLineItem 헤더 아래에 추가 열이 표시됩니다. 자세한 내용은 AWS Cost and Usage Report 사용 설명서의 Split line item details를 참조하세요.

EC2 작업의 경우 이 데이터에서는 리소스 사용량 또는 예약 및 인스턴스의 나머지 리소스를 기준으로 EC2 인스턴스의 비용을 분할합니다.

사전 조건은 다음과 같습니다.

- `ECS_DISABLE_METRICS` Amazon ECS 에이전트 구성 파라미터를 `false`로 설정합니다.

  이 설정이 `false`이면 Amazon ECS 에이전트는 지표를 Amazon CloudWatch로 전송합니다. Linux에서는 이 설정이 기본적으로 `false`이고 지표가 CloudWatch로 전송됩니다. Windows에서는 이 설정이 기본적으로 `true`이므로, AWS Cost Management에서 사용할 지표를 CloudWatch로 전송하려면 설정을 `false`로 변경해야 합니다. ECS 에이전트 구성에 대한 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성을 참조하세요.
- 신뢰할 수 있는 지표를 위한 최소 Docker 버전은 Docker 버전 v20.10.13 이상이며, 이 버전은 Amazon ECS 최적화 AMI 20220607 이상에 포함되어 있습니다.

비용 할당 데이터 분할을 사용하려면 보고서를 생성하고 비용 할당 데이터 분할을 선택해야 합니다. 자세한 내용은 AWS Cost and Usage Report 사용 설명서의 Creating Cost and Usage Reports를 참조하세요.

AWS Cost Management는 작업 CPU 및 메모리 사용량을 사용하여 비용 할당 데이터 분할을 계산합니다. 사용량을 사용할 수 없는 경우 AWS Cost Management에서는 사용량 대신 작업 CPU 및 메모리 예약을 사용할 수 있습니다. CUR에서 예약을 사용하는 것으로 확인되면 컨테이너 인스턴스가 사전 조건을 충족하고 작업 리소스 사용량 지표가 CloudWatch에 나타나는지 확인합니다.

Amazon ECS 모니터링

모니터링은 Amazon ECS와 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. 발생하는 다중 지점 실패를 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분으로부터 모니터링 데이터를 수집해야 합니다. Amazon ECS에 대한 모니터링을 시작하기 전에 다음 질문에 대한 답변을 포함하는 모니터링 계획을 생성합니다.

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

사용 가능한 지표는 클러스터 내 작업 및 서비스의 시작 유형에 따라 달라집니다. 해당 서비스에 Fargate 시작 유형을 사용하는 경우 CPU 및 메모리 사용률 지표를 제공하여 서비스 모니터링을 지원합니다. Amazon EC2 시작 유형의 경우에는 기본 인프라를 구성하는 EC2 인스턴스를 소유하며 이 인스턴스를 모니터링해야 합니다. 클러스터, 서비스 및 작업에서 추가 CPU 및 메모리의 예약과 사용률 지표를 사용할 수 있습니다.

다음 단계에서는 다양한 시간과 다양한 부하 조건에서 성능을 측정하여 환경에서 일반 Amazon ECS 성능의 기준선을 설정합니다. Amazon ECS가 과거 모니터링 데이터를 저장하는 것을 모니터링하면서 현재 성능 데이터를 이 과거 데이터와 비교하면 일반적인 성능 패턴과 성능 이상을 식별하고 이를 해결할 방법을 고안할 수 있습니다.

기준선을 설정하려면 최소한 다음 항목을 모니터링해야 합니다.

- Amazon ECS 클러스터의 CPU 및 메모리 예약과 사용률 지표
- Amazon ECS 서비스의 CPU 및 메모리 사용률 지표

  자세한 내용은 Amazon ECS 지표 보기 섹션을 참조하세요.

Amazon ECS 모니터링 모범 사례

Amazon ECS를 모니터링하기 위해 다음 모범 사례를 사용합니다.

- 큰 문제로 확대되기 전에 작은 문제를 미리 방지하도록 모니터링 우선순위를 지정하세요.
- 다음 질문에 대한 답변을 포함하는 모니터링 계획 생성
  - 모니터링의 목표
  - 모니터링할 리소스
  - 이러한 리소스를 모니터링하는 빈도
  - 사용할 모니터링 도구
  - 모니터링 작업을 수행할 사람
  - 문제 발생 시 알려야 할 대상
- 모니터링을 최대한 자동화합니다.
- Amazon ECS 로그 파일을 확인합니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 로그 보기 섹션을 참조하세요.
- 런타임 모니터링을 사용하면 AWS 환경 내의 계정, 컨테이너, 워크로드 및 데이터를 보호할 수 있습니다. 자세한 내용은 Runtime Monitoring을 사용하여 무단 동작 식별 섹션을 참조하세요.

Amazon ECS 모니터링 도구

AWS는 Amazon ECS를 모니터링하는 데 사용할 수 있는 다양한 도구를 제공합니다. 이들 도구 중에는 모니터링을 자동으로 수행하도록 구성할 수 있는 도구도 있지만, 수동 작업이 필요한 도구도 있습니다. 모니터링 작업은 최대한 자동화하는 것이 좋습니다.

자동 모니터링 도구

다음과 같은 자동 모니터링 도구를 사용하여 Amazon ECS를 관찰하고 문제 발생 시 보고할 수 있습니다.

- Amazon CloudWatch 경보 – 지정한 기간 동안 단일 지표를 감시하고, 여러 기간에 대해 지정된 임계값과 관련하여 지표 값을 기준으로 하나 이상의 태스크를 수행합니다. 이 작업은 Amazon Simple Notification Service(Amazon SNS) 주제 또는 Amazon EC2 Auto Scaling 정책에 전송되는 알림입니다. CloudWatch 경보는 특정 상태에 있다는 이유만으로는 작업을 호출하지 않습니다. 상태가 변경되고 지정한 기간 동안 유지되어야 합니다. 자세한 정보는 CloudWatch를 사용하여 Amazon ECS 모니터링 섹션을 참조하세요.

  Fargate 시작 유형을 사용하는 태스크가 있는 서비스의 경우, CloudWatch 경보를 사용하면 CPU 및 메모리 사용률과 같은 CloudWatch 지표를 기반으로 하여 서비스의 태스크를 축소 및 확장할 수 있습니다. 자세한 정보는 Amazon ECS 서비스 자동 조정 섹션을 참조하세요.

EC2 시작 유형을 사용하는 태스크 또는 서비스가 있는 클러스터의 경우, CloudWatch 경보를 사용하면 클러스터 메모리 예약과 같은 CloudWatch 지표를 기반으로 컨테이너 인스턴스를 축소 및 확장할 수 있습니다.

Amazon ECS 최적화 Amazon Linux AMI를 사용하여 시작된 컨테이너 인스턴스의 경우, 한 곳에서 간편하게 CloudWatch Logs를 사용하여 컨테이너 인스턴스의 여러 로그를 볼 수 있습니다. CloudWatch 에이전트를 컨테이너 인스턴스에 설치해야 합니다. 자세한 정보는 Amazon CloudWatch 사용 설명서의 명령줄을 사용하여 CloudWatch 에이전트 다운로드 및 구성을 참조하세요. `ecsInstanceRole` 역할에 `ECS-CloudWatchLogs` 정책도 추가해야 합니다. 자세한 내용은 컨테이너 인스턴스 모니터링 권한 섹션을 참조하세요.

- Amazon CloudWatch Logs – 태스크 정의에서 `awslogs` 로그 드라이버를 지정하여 Amazon ECS 작업 내의 컨테이너에서 로그 파일을 모니터링, 저장 및 액세스합니다. 자세한 정보는 Amazon ECS 로그를 CloudWatch로 전송 섹션을 참조하세요.

  Amazon ECS 컨테이너 인스턴스에서 운영 체제 및 Amazon ECS 컨테이너 에이전트 로그 파일을 모니터링, 저장 및 액세스할 수도 있습니다. 이와 같은 로그 액세스 방법은 EC2 시작 유형을 사용하는 컨테이너에 사용할 수 있습니다.
- Amazon CloudWatch Events – 이벤트를 일치시키고 하나 이상의 대상 함수 또는 스트림으로 라우팅하여 값을 변경하거나 상태 정보를 캡처하거나 수정 태스크를 수행합니다. 자세한 내용은 이 가이드의 EventBridge를 사용하여 Amazon ECS 오류에 대한 응답 자동화 섹션과 Amazon EventBridge 사용 안내서의 EventBridge는 Amazon CloudWatch Events의 진화를 참조하세요.
- Container Insights - 컨테이너화된 애플리케이션 및 마이크로서비스에서 지표 및 로그를 수집하고 집계하며 요약합니다. Container Insights는 임베디드 지표 형식을 사용하여 데이터를 '성능 로그 이벤트'로 수집합니다. 이러한 성능 로그 이벤트는 카디널리티가 높은 데이터를 대규모로 수집 및 저장할 수 있게 하는 구조화된 JSON 스키마를 사용하는 항목입니다. 이 데이터를 통해 CloudWatch는 클러스터, 작업 및 서비스 수준에서 집계된 지표를 CloudWatch 지표로 생성합니다. Container Insights가 수집하는 지표는 CloudWatch 자동 대시보드에서 사용할 수 있으며 CloudWatch 콘솔의 지표 섹션에서도 볼 수 있습니다.
- AWS CloudTrail 로그 모니터링 – 계정 간에 로그 파일을 공유하고, CloudTrail 로그 파일을 CloudWatch Logs에 전송하여 실시간으로 모니터링하며, Java에서 로그 처리 애플리케이션을 작성하고, CloudTrail에서 전송한 후 로그 파일이 변경되지 않았는지 확인합니다. 자세한 정보는 이 가이드의 AWS CloudTrail을 사용하여 Amazon ECS API 직접 호출 로깅 섹션 및 AWS CloudTrail 사용 설명서의 CloudTrail 로그 파일 작업을 참조하세요.

- Runtime Monitoring - AWS 환경 내 클러스터 및 컨테이너에 대한 위협을 감지합니다. Runtime Monitoring은 파일 액세스, 프로세스 실행 및 네트워크 연결과 같은 개별 Amazon ECS 워크로드에 대한 런타임 가시성을 추가하는 GuardDuty 보안 에이전트를 사용합니다.

수동 모니터링 도구

Amazon ECS 모니터링의 또 한 가지 중요한 부분은 CloudWatch 경보에 포함되지 않는 항목을 수동으로 모니터링해야 한다는 점입니다. CloudWatch, Trusted Advisor 및 기타 AWS 콘솔 대시보드에서는 AWS 환경의 상태를 한 눈에 볼 수 있습니다. 또한 태스크 내 컨테이너 인스턴스와 컨테이너에서 로그 파일을 확인하는 것이 좋습니다.

- Amazon ECS 콘솔:
  - EC2 시작 유형에 대한 클러스터 지표
  - 서비스 지표
  - 서비스 상태
  - 서비스 배포 이벤트
- CloudWatch 홈 페이지:
  - 현재 경보 및 상태
  - 경보 및 리소스 그래프
  - 서비스 상태

  또한 CloudWatch를 사용하여 다음을 수행할 수 있습니다.
  - 사용자 지정 대시보드를 만들어 원하는 서비스 모니터링.
  - 지표 데이터를 그래프로 작성하여 문제를 해결하고 추세 파악.
  - 모든 AWS 리소스 지표 검색 및 찾아보기
  - 문제에 대해 알려주는 경보 생성 및 편집
- 컨테이너 상태 확인 - 컨테이너에서 로컬로 실행되며 애플리케이션 상태 및 가용성을 검증하는 명령입니다. 작업 정의에서 컨테이너별로 구성합니다.
- AWS Trusted Advisor 사용을 통해 AWS 리소스를 모니터링함으로써 성능, 안정성, 보안 및 경제성을 향상할 수 있습니다. 모든 사용자에 대해 4가지 Trusted Advisor 검사를 수행할 수 있고, 비즈니스 또는 엔터프라이즈 지원 플랜에 따라 사용자에 대해 50개 이상의 검사를 수행할 수 있습니다. 자세한 내용은 AWS Trusted Advisor 섹션을 참조하세요.

  Trusted Advisor에서 다음 검사는 Amazon ECS와 관련됩니다.

  - 서비스가 단일 가용 영역에서 실행 중임을 나타내는 내결함성.
  - 여러 가용 영역에 대한 분산 배치 전략을 사용하지 않음을 나타내는 내결함성.
- AWS Compute Optimizer는 AWS 리소스의 구성 및 사용률 지표를 분석하는 서비스입니다. 이는 리소스가 최적 상태인지 여부를 보고하고 최적화 권장 사항을 생성하여 비용을 절감하고 워크로드의 성능을 개선합니다.

  자세한 내용은 Amazon ECS에 대한 AWS Compute Optimizer 권장 사항 섹션을 참조하세요.

CloudWatch를 사용하여 Amazon ECS 모니터링

Amazon ECS에서 원시 데이터를 수집하여 읽기 가능하며 실시간에 가까운 지표로 처리하는 Amazon CloudWatch를 사용해 Amazon ECS 리소스를 모니터링할 수 있습니다. 이러한 통계는 2주간 기록되므로 기록 정보를 보고 클러스터나 서비스가 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다. Amazon ECS 지표 데이터는 1분 간격으로 CloudWatch로 자동 전송됩니다. CloudWatch에 대한 자세한 정보는 Amazon CloudWatch 사용 설명서를 참조하세요.

Amazon ECS는 클러스터 및 서비스에 대한 무료 지표를 제공합니다. 추가 비용을 지불하면 클러스터에서 CPU, 메모리, EBS 파일 시스템 사용률을 포함한 작업별 지표에 대한 Amazon ECS CloudWatch Container Insights를 켤 수 있습니다. Container Insights에 대한 자세한 정보는 관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.

고려 사항

Amazon ECS CloudWatch 지표를 사용하는 경우 다음을 고려해야 합니다.

- Fargate에 호스팅된 모든 Amazon ECS 서비스는 CloudWatch CPU 및 메모리 사용률 지표를 자동으로 지원하므로 수동 단계를 수행하지 않아도 됩니다.

- Amazon EC2 인스턴스에 호스팅된 모든 Amazon ECS 작업 또는 서비스의 경우 CloudWatch 지표를 생성하려면 Amazon EC2 인스턴스에 컨테이너 에이전트 버전 `1.4.0` 이상(Linux) 또는 `1.0.0` 이상(Windows)이 필요합니다. 그러나 최신 버전의 컨테이너 에이전트를 사용하는 것이 좋습니다. 에이전트 버전을 확인하고 최신 버전으로 업데이트하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.
- 신뢰할 수 있는 CloudWatch 지표를 위한 최소 Docker 버전은 Docker 버전 `20.10.13` 이상입니다.
- Amazon EC2 인스턴스에는 Amazon EC2 인스턴스를 시작할 때 사용하는 IAM 역할에 대한 `ecs:StartTelemetrySession` 권한도 필요합니다. Amazon ECS에서 CloudWatch 지표를 사용

하기 전에 Amazon ECS 컨테이너 인스턴스 IAM 역할을 생성한 경우 이 권한을 추가해야 할 수 있습니다. 컨테이너 인스턴스 IAM 역할 및 컨테이너 인스턴스에 대한 관리형 IAM 정책 연결 방법에 대한 자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.

- Amazon ECS 컨테이너 에이전트 구성에서 `ECS_DISABLE_METRICS=true`를 설정하여 Amazon EC2 인스턴스에서 CloudWatch 지표 수집을 비활성화할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

권장 지표

Amazon ECS에서는 리소스를 모니터링하는 데 사용할 수 있는 무료 CloudWatch 지표를 제공합니다. 클러스터 전체의 CPU와 메모리 예약과 CPU, 메모리 및 EBS 파일 시스템 사용률, 그리고 클러스터의 서비스에 대한 CPU, 메모리, EBS 파일 시스템 사용률을 이 지표를 사용해 측정할 수 있습니다. GPU 워크로드의 경우 클러스터 전체의 GPU 예약을 측정할 수 있습니다.

클러스터에서 Amazon ECS 작업이 호스팅되는 인프라에 따라 사용 가능한 지표가 결정됩니다. Fargate 인프라에 호스팅된 작업의 경우 Amazon ECS는 CPU, 메모리, EBS 파일 시스템 사용률 지표를 제공하여 서비스 모니터링을 지원합니다. EC2 인스턴스에 호스팅된 작업의 경우 Amazon ECS는 CPU, 메모리, GPU 예약 지표와 CPU 및 메모리 사용률 지표를 클러스터 및 서비스 수준에서 제공합니다. 기본 인프라를 별도로 구성하는 Amazon EC2 인스턴스를 모니터링해야 합니다. Amazon EC2 인스턴스 모니터링에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 모니터링을 참조하세요.

Amazon ECS에서 사용할 권장 경보에 대한 자세한 내용은 Amazon CloudWatch Logs 사용 설명서에서 다음 중 하나를 참조하세요.

- Amazon ECS
- Container Insights가 포함된 Amazon ECS

Amazon ECS 지표 보기

클러스터에서 실행 중인 리소스가 존재하면 Amazon ECS 및 CloudWatch 콘솔에서 지표를 볼 수 있습니다. Amazon ECS 콘솔은 클러스터 및 서비스 지표의 24시간 최대, 최소 및 평균 값을 제공합니다. CloudWatch 콘솔은 세분화됨과 동시에 사용자 정의가 가능한 리소스 표시뿐만 아니라 서비스에서 실행중인 작업 수를 제공합니다.

Amazon ECS 콘솔

Amazon ECS 콘솔에서 Amazon ECS 서비스 CPU 및 메모리 사용률 지표를 확인할 수 있습니다. 서비스 지표에 대한 보기에는 직전 24시간의 평균, 최소 및 최대 값, 그리고 5분 간격의 데이터 포인트가 표시됩니다. 자세한 내용은 Amazon ECS 서비스 사용률 지표 섹션을 참조하세요.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 지표를 확인할 클러스터를 선택합니다.
3. 보려는 지표를 결정합니다.

보려는 지표 출처	단계
클러스터	클러스터 세부 정보 페이지에서 지표 탭을 선택합니다. 또한 CloudWatch Container Insights 지표를 활성화한 경우 해당 지표를 볼 수 있는 링크가 CloudWatch 콘솔에 제공됩니다.
서비스	클러스터 세부 정보 페이지의 서비스 탭에서 서비스를 선택합니다. 지표는 상태 및 지표에서 사용할 수 있습니다.

CloudWatch 콘솔

Fargate 시작 유형의 경우 CloudWatch 콘솔에서도 Amazon ECS 서비스 지표를 확인할 수 있습니다. 콘솔은 Amazon ECS 지표에 대한 가장 상세한 보기를 제공하며 필요에 맞춰 보기를 사용자 지정할 수 있습니다. 서비스 사용률 및 서비스 RUNNING 작업 수를 볼 수 있습니다.

EC2 시작 유형의 경우 CloudWatch 콘솔에서도 Amazon ECS 클러스터 및 서비스 지표를 확인할 수 있습니다. 콘솔은 Amazon ECS 지표에 대한 가장 상세한 보기를 제공하며 필요에 맞춰 보기를 사용자 지정할 수 있습니다.

지표를 보는 방법에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 사용 가능한 지표 보기를 참조하세요.

Amazon ECS CloudWatch 지표

CloudWatch 사용량 지표를 사용하여 계정의 리소스 사용량을 확인할 수 있습니다. 이러한 지표를 사용하여 CloudWatch 그래프 및 대시보드에서 현재 서비스 사용량을 시각화합니다.

Amazon ECS는 1분 간격으로 지표를 CloudWatch에 전송합니다. 이러한 지표는 RUNNING 상태의 태스크가 있는 리소스에 대해 수집됩니다. 클러스터, 서비스 또는 기타 리소스에 실행 중인 태스크가 없는 경우 해당 기간 동안 해당 리소스에 대한 지표가 보고되지 않습니다. 예를 들어, 하나의 서비스가 포함된 클러스터에서 해당 서비스에 RUNNING 상태의 태스크가 없는 경우 CloudWatch에 지표가 전송되지 않습니다. 마찬가지로, 두 개의 서비스가 있는데 그중 하나에는 실행 중인 태스크가 있고 다른 하나에는 없는 경우, 실행 중인 태스크가 있는 서비스에 대한 지표만 전송됩니다.

지표	설명	유효한 차원	유용한 통계	단위
CPUReservation	클러스터 또는 서비스에서 예약된 CPU 단위의 비율(%)입니다. CPU 예약(ClusterName 으로 필터링됨)은 클러스터의 Amazon ECS 작업이 예약한 총 CPU 단위를 클러스터에 등록된 모든 Amazon EC2 인스턴스에 대한 총 CPU 단위로 나눈 값으로 측정됩니다. ACTIVE 또는 DRAINING 상태의 Amazon EC2 인스턴스만 CPU 예약 지표에 영향을 줍니다. 이 지표는	ClusterName .	평균, 최솟값, 최댓값	%

지표	설명	유효한 차원	유용한 통계	단위
	Amazon EC2 인스턴스에 호스팅된 작업에서만 지원됩니다.			

지표	설명	유효한 차원	유용한 통계	단위
CPUUtilization	클러스터 또는 서비스에서 사용하는 CPU 단위의 비율(%)입니다. 클러스터 수준 CPU 사용률(ClusterName 으로 필터링됨)은 클러스터의 Amazon ECS 작업이 사용 중인 총 CPU 단위를 클러스터에 등록된 모든 Amazon EC2 인스턴스에 대한 총 CPU 단위로 나눈 값으로 측정됩니다. ACTIVE 또는 DRAINING 상태의 Amazon EC2 인스턴스만 CPU 예약 지표에 영향을 줍니다. 클러스터 수준 지표는 Amazon EC2 인스턴스에 호스팅된 작업에서만 지원됩니다. 서비스 수준 CPU 사용률(ClusterName 및 ServiceNa	ClusterName , ServiceName	평균, 최솟값, 최댓값	%

지표	설명	유효한 차원	유용한 통계	단위
	me 으로 필터링됨)은 서비스에 속하는 작업이 사용 중인 총 CPU 단위를 서비스에 속하는 작업에 예약된 총 CPU 단위 수로 나눈 값으로 측정됩니다. 서비스 수준 지표는 Amazon EC2 인스턴스 및 Fargate에 호스팅된 작업에서 지원됩니다.			

지표	설명	유효한 차원	유용한 통계	단위
MemoryReservation	클러스터 내 실행 작업이 예약한 메모리의 비율(%)입니다. 클러스터 메모리 예약은 클러스터의 Amazon ECS 작업이 예약한 총 메모리를 클러스터에 등록된 모든 Amazon EC2 인스턴스에 대한 총 메모리 크기로 나눈 값으로 측정됩니다. 이 지표는 ClusterName 으로만 필터링 가능합니다. ACTIVE 또는 DRAINING 상태의 Amazon EC2 인스턴스만 메모리 예약 지표에 영향을 줍니다. 클러스터 수준 메모리 예약 지표는 Amazon EC2 인스턴스에 호스팅된 작업에서만 지원됩니다.	ClusterName .	평균, 최솟값, 최댓값	%

지표	설명	유효한 차원	유용한 통계	단위
	Note 메모리 사용률을 계산할 때 `MemoryReservation` 지정된 경우 총 메모리 대신 이 값이 계산에 사용됩니다.			

지표	설명	유효한 차원	유용한 통계	단위
MemoryUtilization	클러스터 또는 서비스에서 사용 중인 메모리의 비율입니다. 클러스터 수준 메모리 사용률(ClusterName 으로 필터링됨)은 클러스터의 Amazon ECS 작업이 사용 중인 총 메모리를 클러스터에 등록된 모든 Amazon EC2 인스턴스에 대한 총 메모리로 나눈 값으로 측정됩니다. ACTIVE 또는 DRAINING 상태의 Amazon EC2 인스턴스만 메모리 사용률 지표에 영향을 줍니다. 클러스터 수준 지표는 Amazon EC2 인스턴스에 호스팅된 작업에서만 지원됩니다. 서비스 수준 메모리 사용률(ClusterName 및 ServiceName 으로 필터링	ClusterName , ServiceName	평균, 최솟값, 최댓값	%

지표	설명	유효한 차원	유용한 통계	단위
	됨)은 서비스에 속하는 작업이 사용 중인 총 메모리를 서비스에 속하는 작업에 예약된 총 메모리로 나눈 값으로 측정됩니다. 서비스 수준 지표는 Amazon EC2 인스턴스 및 Fargate에 호스팅된 작업에서 지원됩니다.			

지표	설명	유효한 차원	유용한 통계	단위
EBSFilesystemUtilization	서비스의 작업에서 사용하는 Amazon EBS 파일 시스템의 비율입니다. 서비스 수준 EBS 파일 시스템 사용률 지표(ClusterName 및 ServiceName 으로 필터링됨)는 서비스에 속하는 작업이 사용 중인 EBS 파일 시스템 총 크기를 서비스에 속하는 모든 작업에 할당된 EBS 파일 시스템 스토리지의 총 크기로 나눈 값으로 측정됩니다. 서비스 수준 EBS 파일 시스템 사용률 지표는 EBS 볼륨이 연결된 Amazon EC2 인스턴스(컨테이너 에이전트 버전 1.79.0 사용) 및 Fargate(플랫폼 버전 1.4.0 사용)에 호스팅된 작업에서만 사용할 수 있습니다.	ClusterName , ServiceName	평균, 최솟값, 최댓값	%

지표	설명	유효한 차원	유용한 통계	단위
	Note Fargate에 호스팅된 작업의 경우 Fargate에서만 사용하는 디스크의 공간이 있습니다. Fargate가 사용하는 공간에는 비용이 들지 않지만 `df`와 같은 도구를 사용하면 이 추가 스토리지를 확인할 수 있습니다.			

지표	설명	유효한 차원	유용한 통계	단위
GPUReservation	클러스터 내 실행 작업이 예약한 전체 가용 GPU 백분율입니다. 클러스터 수준 GPU 예약 지표는 클러스터의 Amazon ECS 작업이 예약한 GPU 수를 클러스터에 등록된 GPU를 포함한 모든 Amazon EC2 인스턴스에서 사용 가능한 총 GPU 수로 나눈 값으로 측정됩니다. ACTIVE 또는 DRAINING 상태의 Amazon EC2 인스턴스만 GPU 예약 지표에 영향을 줍니다.	ClusterName	평균, 최솟값, 최댓값	%

지표	설명	유효한 차원	유용한 통계	단위
ActiveConnectionCount	클라이언트로부터 선택한 DiscoveryName 을 공유하는 작업에서 실행되는 Amazon ECS Service Connect 프록시로의 활성 동시 연결의 총 수입니다. 이 지표는 Amazon ECS Service Connect 를 구성한 경우에만 사용할 수 있습니다. 유효한 차원: DiscoveryName 및 DiscoveryName, ServiceName, ClusterName .	DiscoveryName 및 DiscoveryName , ServiceName , ClusterName	평균, 최솟값, 최댓값, 합계	개수

지표	설명	유효한 차원	유용한 통계	단위
NewConnectionCount	클라이언트로부터 선택한 DiscoveryName 을 공유하는 작업에서 실행되는 Amazon ECS Service Connect 프록시로 설정된 새 연결의 총 수입니다. 이 지표는 Amazon ECS Service Connect 를 구성한 경우에만 사용할 수 있습니다.	DiscoveryName 및 DiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값, 합계	개수
ProcessedBytes	Service Connect 프록시에서 처리하는 인바운드 트래픽의 총 바이트 수입니다. 이 지표는 Amazon ECS Service Connect 를 구성한 경우에만 사용할 수 있습니다.	DiscoveryName 및 DiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값, 합계	바이트

지표	설명	유효한 차원	유용한 통계	단위
RequestCount	Service Connect 프록시에서 처리한 인바운드 트래픽 요청의 수입니다. 이 지표는 Amazon ECS Service Connect를 구성한 경우에만 사용할 수 있습니다. 또한 태스크 정의의 포트 매핑에서 appProtocol을 구성해야 합니다.	DiscoveryName 및 DiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값, 합계	개수
GrpcRequestCount	Service Connect 프록시에서 처리한 gRPC 인바운드 트래픽 요청의 수입니다. 이 지표는 Amazon ECS Service Connect를 구성했고 작업 정의의 포트 매핑에서 appProtocol이 GRPC인 경우에만 사용할 수 있습니다.	DiscoveryName 및 DiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값, 합계	개수

지표	설명	유효한 차원	유용한 통계	단위
HTTPCode_Target_2XX_Count	작업의 애플리케이션에서 생성된 번호 200~299를 사용하는 HTTP 응답 코드의 수입니다. 이러한 작업이 대상입니다. 이 지표는 직접 전송되는 응답이 아닌 이러한 작업의 애플리케이션에서 Service Connect 프록시로 전송하는 응답의 수만 계산합니다. 이 지표는 Amazon ECS Service Connect를 구성했고 작업 정의의 포트 매핑에서 appProtocol 이 HTTP 또는 HTTP2인 경우에만 사용할 수 있습니다. 유효한 차원:	TargetDiscoveryName 및 TargetDiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값, 합계	개수

지표	설명	유효한 차원	유용한 통계	단위
HTTPCode_Target_3XX_Count	작업의 애플리케이션에서 생성된 번호 300~399를 사용하는 HTTP 응답 코드의 수입니다. 이러한 작업이 대상입니다. 이 지표는 직접 전송되는 응답이 아닌 이러한 작업의 애플리케이션에서 Service Connect 프록시로 전송하는 응답의 수만 계산합니다. 이 지표는 Amazon ECS Service Connect를 구성했고 작업 정의의 포트 매핑에서 appProtocol 이 HTTP 또는 HTTP2인 경우에만 사용할 수 있습니다.	TargetDiscoveryName 및 TargetDiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값, 합계	개수

지표	설명	유효한 차원	유용한 통계	단위
HTTPCode_Target_4XX_Count	작업의 애플리케이션에서 생성된 번호 400~499를 사용하는 HTTP 응답 코드의 수입니다. 이러한 작업이 대상입니다. 이 지표는 직접 전송되는 응답이 아닌 이러한 작업의 애플리케이션에서 Service Connect 프록시로 전송하는 응답의 수만 계산합니다. 이 지표는 Amazon ECS Service Connect를 구성했고 작업 정의의 포트 매핑에서 appProtocol 이 HTTP 또는 HTTP2인 경우에만 사용할 수 있습니다.	TargetDiscoveryName 및 TargetDiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값, 합계	개수

지표	설명	유효한 차원	유용한 통계	단위
HTTPCode_Target_5XX_Count	작업의 애플리케이션에서 생성된 번호 500~599를 사용하는 HTTP 응답 코드의 수입니다. 이러한 작업이 대상입니다. 이 지표는 직접 전송되는 응답이 아닌 이러한 작업의 애플리케이션에서 Service Connect 프록시로 전송하는 응답의 수만 계산합니다. 이 지표는 Amazon ECS Service Connect를 구성했고 작업 정의의 포트 매핑에서 appProtocol 이 HTTP 또는 HTTP2인 경우에만 사용할 수 있습니다.	TargetDiscoveryName 및 TargetDiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값, 합계	개수

지표	설명	유효한 차원	유용한 통계	단위
RequestCountPerTarget	선택한 DiscoveryName 을 공유하는 각 대상에서 수신한 평균 요청 수입니다. 이 지표는 Amazon ECS Service Connect를 구성한 경우에만 사용할 수 있습니다.	TargetDiscoveryName 및 TargetDiscoveryName, ServiceName, ClusterName	평균	개수
TargetProcessedBytes	Service Connect 프록시에서 처리한 총 바이트 수입니다. 이 지표는 Amazon ECS Service Connect를 구성한 경우에만 사용할 수 있습니다.	TargetDiscoveryName 및 TargetDiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값, 합계	바이트

지표	설명	유효한 차원	유용한 통계	단위
TargetResponseTime	애플리케이션 요청 처리의 지연 시간입니다. 대상 작업의 Service Connect 프록시에 요청이 도달한 후 대상 애플리케이션으로부터의 응답이 프록시로 돌아올 때까지 걸린 시간(밀리초). 이 지표는 Amazon ECS Service Connect를 구성한 경우에만 사용할 수 있습니다.	TargetDiscoveryName 및 TargetDiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값	밀리초
ClientTLSNegotiationErrorCount	TLS 연결이 실패한 총 횟수입니다. 이 지표는 TLS가 활성화된 경우에만 사용됩니다. 이 지표는 Amazon ECS Service Connect를 구성한 경우에만 사용할 수 있습니다.	DiscoveryName 및 DiscoveryName, ServiceName, ClusterName	평균, 최솟값, 최댓값, 합계	개수

지표	설명	유효한 차원	유용한 통계	단위
TargetTLSNegotiationErrorCount	클라이언트 인증서 누락, AWS Private CA 확인 실패 또는 SAN 확인 실패로 인해 TLS 연결이 실패한 총 횟수입니다. 이 지표는 TLS가 활성화된 경우에만 사용됩니다. 이 지표는 Amazon ECS Service Connect를 구성한 경우에만 사용할 수 있습니다.	ServiceName , ClusterName , TargetDiscoveryName 및 TargetDiscoveryName	평균, 최솟값, 최댓값, 합계	개수

Amazon ECS 지표 차원

Amazon ECS 지표는 AWS/ECS 네임스페이스를 사용하며 다음 차원의 지표를 제공합니다. Amazon ECS는 RUNNING 상태의 작업이 있는 리소스에 대한 지표만 전송합니다. 예를 들어, 하나의 서비스가 포함된 클러스터에서 해당 서비스에 RUNNING 상태의 작업이 없는 경우 CloudWatch에 지표가 전송되지 않습니다. 두 개의 서비스가 있는데 그 중 하나에는 실행 중인 작업이 있고 다른 하나에는 없는 경우, 실행 중인 작업이 있는 서비스에 대한 지표만 전송됩니다.

차원	정의
ClusterName	이 차원은 사용자가 지정된 클러스터의 모든 리소스에 대해 요청하는 데이터를 필터링합니다. 모든 Amazon ECS 지표는 ClusterName 으로 필터링됩니다.

차원	정의
`ServiceName`	이 차원은 사용자가 지정된 클러스터에서 지정된 서비스의 모든 리소스에 대해 요청하는 데이터를 필터링합니다.
`DiscoveryName`	이 차원은 트래픽 지표에 대해 요청하는 데이터를 모든 Amazon ECS 클러스터에서 지정된 Service Connect 검색 이름으로 필터링합니다. 실행 중인 컨테이너의 특정 포트에는 여러 개의 검색 이름이 있을 수 있습니다.
`DiscoveryName, ServiceName, ClusterName`	이 차원은 트래픽 지표에 대해 요청한 데이터를 검색 이름이 있고 이 클러스터의 서비스에서 생성한 작업의 지정된 Service Connect 검색 이름으로 필터링합니다. 다른 네임스페이스의 여러 서비스에서 동일한 검색 이름을 재사용한 경우 이 차원을 사용하여 특정 서비스에 대한 인바운드 트래픽 지표를 확인할 수 있습니다. 실행 중인 컨테이너의 특정 포트에는 여러 개의 검색 이름이 있을 수 있습니다.

차원	정의
`TargetDiscoveryName`	이 차원은 트래픽 지표에 대해 요청하는 데이터를 모든 Amazon ECS 클러스터에서 지정된 Service Connect 검색 이름으로 필터링합니다. `DiscoveryName` 과 달리 이러한 트래픽 지표는 이 네임스페이스에 Service Connect 구성이 다른 Amazon ECS 작업에서 들어오는 이 `DiscoveryName` 에 대한 인바운드 트래픽만 측정합니다. 여기에는 클라이언트 전용 또는 클라이언트-서버 Service Connect 구성으로 서비스에서 수행한 작업이 포함됩니다. 실행 중인 컨테이너의 특정 포트에는 여러 개의 검색 이름이 있을 수 있습니다.
`TargetDiscoveryName, ServiceName, ClusterName`	이 차원은 트래픽 지표에 대해 요청한 데이터를 지정된 Service Connect 검색 이름으로 필터링하지만 이 클러스터의 서비스에서 생성한 작업의 트래픽만 계산합니다. 이 차원을 사용하여 다른 서비스의 특정 클라이언트에서 들어오는 인바운드 트래픽 지표를 확인할 수 있습니다. `DiscoveryName, ServiceName, ClusterName` 과 달리 이러한 트래픽 지표는 이 네임스페이스에 Service Connect 구성이 있는 다른 Amazon ECS 작업에서 들어오는 이 `DiscoveryName` 에 대한 인바운드 트래픽만 측정합니다. 여기에는 클라이언트 전용 또는 클라이언트-서버 Service Connect 구성으로 서비스에서 수행한 작업이 포함됩니다. 실행 중인 컨테이너의 특정 포트에는 여러 개의 검색 이름이 있을 수 있습니다.

AWS Fargate 사용량 지표

CloudWatch 사용량 지표를 사용하여 계정의 리소스 사용량을 확인할 수 있습니다. 이러한 지표를 사용하여 CloudWatch 그래프 및 대시보드에서 현재 서비스 사용량을 시각화합니다.

AWS Fargate 사용량 지표는 AWS 서비스 할당량에 해당합니다. 사용량이 서비스 할당량에 가까워지면 경고하는 경보를 구성할 수 있습니다. Fargate 서비스 할당량에 대한 자세한 내용은 Amazon Web Services 일반 참조의 Amazon ECS 엔드포인트 및 할당량을 참조하세요.

AWS Fargate는 `AWS/Usage` 네임스페이스에 다음 지표를 게시합니다.

지표	설명
`ResourceCount`	계정에서 실행 중인 지정된 리소스의 총 수입니다. 리소스는 지표와 연결된 차원으로 정의됩니다.

다음 차원은 AWS Fargate에 의해 게시되는 사용량 지표를 구체화하는 데 사용됩니다.

차원	설명
`Service`	리소스가 포함된 AWS 서비스의 이름 AWS Fargate 사용량 지표의 경우 이 차원 값은 `Fargate`입니다.
`Type`	보고되는 엔터티의 유형입니다. 현재 AWS Fargate 사용량 지표에 대한 유일한 유효 값은 `Resource`입니다.
`Resource`	실행 중인 리소스의 유형입니다. 실행 중인 리소스의 유형입니다. 현재 AWS Fargate 사용량 지표에 대해 유일하게 유효한 값은 실행 중인 인스턴스에 대한 정보를 반환하는 vCPU입니다.
`Class`	추적 중인 리소스의 클래스입니다. 추적 중인 리소스의 클래스입니다. 리소스 차원의 값으로서의 vCPU가 포함된 AWS Fargate 사용량 지표의 경우 유효한 값은 `Standard/OnDemand` 및 `Standard/Spot` 입니다.

Service Quotas 콘솔을 사용하여 그래프에 사용량을 시각화하고 AWS Fargate 사용량이 서비스 할당량에 가까워지면 경고하는 경보를 구성할 수 있습니다. 할당량 임계값에 가까워지면 알림을 보내도

록 CloudWatch 경보를 생성하는 방법에 대한 자세한 내용은 Service Quotas 사용 설명서의 Service Quotas and Amazon CloudWatch alarms를 참조하세요.

.

Amazon ECS 클러스터 예약 지표

클러스터 예약 지표는 클러스터의 각 활성 컨테이너 인스턴스에 대해 등록된 전체 CPU, 메모리 및 GPU를 기준으로 클러스터의 모든 Amazon ECS 작업에 의해 예약된 CPU, 메모리 및 GPU의 백분율로 측정됩니다. `ACTIVE` 또는 `DRAINING` 상태의 컨테이너 인스턴스만 클러스터 예약 지표에 영향을 줍니다. 이 지표는 EC2 인스턴스에서 호스팅된 작업 또는 서비스가 있는 클러스터에서만 사용됩니다. AWS Fargate에 작업이 호스팅된 클러스터에서는 지원되지 않습니다.

```
                              (Total CPU units reserved by tasks in cluster) x 100
Cluster CPU reservation =
 ---------------------------------------------------------------
                           (Total CPU units registered by container instances in
 cluster)
```

```
                                 (Total MiB of memory reserved by tasks in cluster x
 100)
Cluster memory reservation =
 ---------------------------------------------------------------------
                              (Total MiB of memory registered by container instances in
 cluster)
```

```
                                (Total GPUs reserved by tasks in cluster x 100)
Cluster GPU reservation =
 ---------------------------------------------------------------------
                              (Total GPUs registered by container instances in cluster)
```

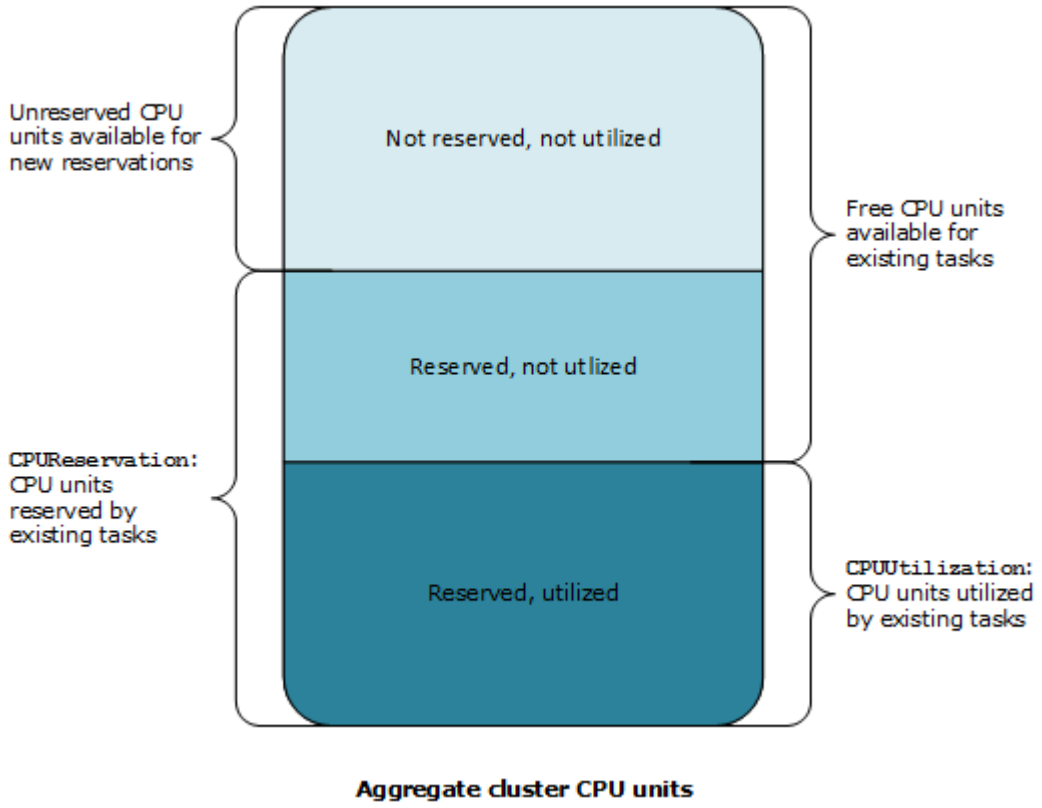
클러스터에서 태스크를 실행하면 Amazon ECS가 해당 태스크 정의를 구문 분석하고 컨테이너 정의에서 지정된 전체 CPU 단위, 메모리(MiB) 및 GPU를 예약합니다. 1분 단위로 Amazon ECS가 현재 클러스터에서 실행 중인 각 작업에 대해 예약된 CPU 단위, 메모리 MiB 및 GPU를 계산합니다. 클러스터에서 실행 중인 모든 작업에 대해 예약된 전체 CPU, 메모리 및 GPU가 계산되며, 해당 수치는 클러스터의 전체 등록 리소스에 대한 백분율로 CloudWatch에 보고됩니다. 작업 정의에서 소프트 제한(`memoryReservation`)을 지정하면 이는 예약된 메모리의 양을 계산하는 데 사용됩니다. 그렇지 않으면 하드 제한(`memory`)이 사용됩니다. 클러스터의 작업에 의해 예약된 총 메모리 MiB에는 임시 파일 시스템(`tmpfs`) 볼륨 크기 및 `sharedMemorySize`(작업 정의에 정의된 경우)도 포함됩니다. 하드 및 소

프트 제한, 공유 메모리 크기 및 tmpfs 볼륨 크기에 대한 자세한 정보는 Task Definition Parameters를 참조하세요.

예를 들어 클러스터에 2개의 활성 컨테이너 인스턴스 `c4.4xlarge` 인스턴스 및 `c4.large` 인스턴스가 등록되어 있습니다. `c4.4xlarge` 인스턴스는 16,384개 CPU 단위와 30,158MiB 및 메모리로 클러스터에 등록되어 있습니다. `c4.large` 인스턴스는 2,048개 CPU 단위와 3,768MiB 메모리로 등록되어 있습니다. 이 클러스터의 전체 리소스는 18,432개 CPU 단위와 33,926MiB 메모리입니다.

태스크 정의가 1,024개 CPU 단위와 2,048MiB 메모리를 예약하고 이 클러스터에서 이 태스크 정의로 10개의 작업이 시작될 경우(현재 실행 중인 다른 태스크는 없음) 총 10,240개의 CPU 단위와 20,480MiB의 메모리가 예약되며 이러한 수치는 클러스터에 대한 55% CPU 예약 및 60% 메모리 예약으로 CloudWatch에 보고됩니다.

다음 그림은 클러스터의 총 등록 CPU 단위와 CPU 단위 예약 및 사용이 기존 태스크와 새 작업 배치에 어떤 의미를 갖는지 보여줍니다. 하단(예약됨, 사용됨) 및 중간(예약됨, 사용되지 않음) 블록은 클러스터에서 실행 중인 기존 작업에 대해 예약된 전체 CPU 단위, 즉 `CPUReservation` CloudWatch 지표를 나타냅니다. 하단 블록은 클러스터에서 실행 작업이 실제로 사용 중인 예약된 CPU 단위, 즉 `CPUUtilization` CloudWatch 지표를 나타냅니다. 상단 블록은 기존 작업에 의해 예약되지 않은 CPU 단위를 나타내며, 새 작업 배치에 이들 CPU 단위를 사용할 수 있습니다. 기존 작업도 CPU 리소스 수요가 증가할 경우, 예약되지 않은 CPU 단위를 사용할 수 있습니다. 자세한 정보는 `cpu` 태스크 정의 파라미터 설명서를 참조하세요.



Amazon ECS 클러스터 사용률 지표

CPU, 메모리, 그리고 작업에 EBS 볼륨이 연결된 경우 EBS 파일 시스템 사용률에 대한 클러스터 사용률 지표를 사용할 수 있습니다. 이 지표는 Amazon EC2 인스턴스에서 호스팅된 작업 또는 서비스가 있는 클러스터에서만 사용할 수 있습니다. AWS Fargate에 호스팅된 작업을 포함하는 클러스터에서 지원되지 않습니다.

Amazon ECS 클러스터 수준 CPU 및 메모리 사용률 지표

CPU 및 메모리 사용률은 클러스터에 등록된 각 활성 Amazon EC2 인스턴스에 대해 등록된 전체 CPU 및 메모리를 기준으로, 클러스터의 모든 작업에서 사용하는 CPU 및 메모리의 비율로 측정됩니다. `ACTIVE` 또는 `DRAINING` 상태의 Amazon EC2 인스턴스만 클러스터 사용률 지표에 영향을 줍니다.

```
                                  (Total CPU units used by tasks in cluster) x 100
Cluster CPU utilization =
 ------------------------------------------------------------------
                           (Total CPU units registered by container instances in
 cluster)
```

```
                                    (Total MiB of memory used by tasks in cluster x
 100)
Cluster memory utilization =
 ------------------------------------------------------------------
                              (Total MiB of memory registered by container instances in
 cluster)
```

1분 단위로 각 Amazon EC2 인스턴스의 Amazon ECS 컨테이너 에이전트가 해당 Amazon EC2 인스턴스에서 실행 중인 각 작업에 대해 현재 사용 중인 CPU 단위 수 및 메모리(MiB) 크기를 계산합니다. 이 정보는 Amazon ECS에 다시 보고됩니다. 클러스터에서 실행 중인 모든 작업에 사용되는 전체 CPU 및 메모리가 계산되며, 해당 수치는 클러스터의 전체 등록 리소스에 대한 백분율로 CloudWatch에 보고됩니다.

예를 들어 클러스터에 2개의 활성 Amazon ECS 인스턴스, `c4.4xlarge` 인스턴스 및 `c4.large` 인스턴스가 등록되어 있습니다. `c4.4xlarge` 인스턴스는 `16,384`개의 CPU 단위와 `30,158`MiB의 메모리로 클러스터에 등록되어 있습니다. `c4.large` 인스턴스는 `2,048`개 CPU 단위와 `3,768`MiB의 메모리로 등록되어 있습니다. 이 클러스터의 전체 리소스는 `18,432`개 CPU 단위와 `33,926`MiB의 메모리입니다.

이 클러스터에서 10개의 작업이 실행 중이고 작업마다 `1,024`개 CPU 단위와 `2,048`MiB의 메모리를 사용할 경우, 클러스터에서 총 `10,240`개의 CPU 단위와 `20,480`의 MiB 메모리가 사용됩니다. 이러한 수치는 클러스터에 대한 55% CPU 사용 및 60% 메모리 사용으로 CloudWatch에 보고됩니다.

Amazon ECS 클러스터 수준 Amazon EBS 파일 시스템 사용률

클러스터 수준 EBS 파일 시스템 사용률 지표는 클러스터에서 실행 중인 작업이 사용하는 EBS 파일 시스템의 총 크기를 클러스터의 모든 작업에 대해 할당된 EBS 파일 시스템 스토리지의 총 크기로 나눈 값으로 측정됩니다.

```
                                       (Total GB of EBS filesystem used by tasks in
 cluster x 100)
Cluster EBS filesystem utilization =
 ------------------------------------------------------------------
                                       (Total GB of EBS filesystem allocated to tasks
 in cluster)
```

Amazon ECS 서비스 사용률 지표

CPU, 메모리, 그리고 작업에 EBS 볼륨이 연결된 경우 EBS 파일 시스템 사용률에 대한 서비스 사용률 지표를 사용할 수 있습니다. 서비스 수준 지표는 Amazon EC2 인스턴스 및 Fargate에 호스팅된 작업이 있는 서비스에서 지원됩니다.

서비스 수준 CPU 및 메모리 사용률

CPU 및 메모리 사용률 지표는 서비스의 작업 정의에 지정된 CPU 및 메모리를 기준으로 클러스터의 특정 서비스에 속하는 Amazon ECS 작업이 사용하는 CPU 및 메모리의 비율로 측정됩니다.

CloudWatch에서 이러한 지표를 볼 때 다양한 통계를 선택할 수 있습니다.

- 평균: 서비스의 모든 태스크에 대한 평균 사용률입니다. 아래 공식을 사용하여 계산됩니다.
- 최소: 서비스에서 리소스 사용량이 가장 낮은 태스크의 사용률입니다. 이는 태스크 정의에 지정된 것과 비교하여 리소스 집약도가 가장 낮은 태스크에서 사용하는 CPU 또는 메모리의 백분율을 나타냅니다.
- 최대: 서비스에서 리소스 사용량이 가장 높은 태스크의 사용률입니다. 이는 태스크 정의에 지정된 것과 비교하여 리소스 집약도가 가장 높은 태스크에서 사용하는 CPU 또는 메모리의 백분율을 나타냅니다.

다음 공식은 평균 통계가 계산되는 방식을 보여줍니다.

```
                                  (Total CPU units used by tasks in service) x 100
Service CPU utilization =
 ---------------------------------------------------------------
                           (Total CPU units specified in the task definition) x (Number
 of tasks in the service)
```

```
                                     (Total MiB of memory used by tasks in service x
 100)
Service memory utilization =
 ---------------------------------------------------------------------
                              (Total MiB of memory specified in the task definition)  x
 (Number of tasks in the service)
```

> **Note**
>
> 위의 공식은 평균 통계에만 적용됩니다. 최소 및 최대 통계는 모든 태스크의 집계 계산이 아니라 각각 리소스 사용률이 가장 낮고 가장 높은 개별 태스크를 나타냅니다.

Amazon ECS는 20초마다 지표를 수집합니다. Amazon ECS 컨테이너 에이전트는 매분 서비스가 소유한 실행 중인 태스크 각각에 대해 현재 사용 중인 CPU 단위 수 및 메모리(MiB) 크기를 계산합니다. 본 정보는 Amazon ECS에 다시 보고됩니다. 클러스터에서 실행 중인 서비스가 소유한 모든 작업이 사용하는 전체 CPU 및 메모리가 계산되며, 이 수치는 서비스의 태스크 정의에서 해당 서비스에 대해 지정된 전체 리소스의 백분율로 CloudWatch에 보고됩니다. 최솟값 및 최댓값은 20초 지표 중 가장 작은 값과 가장 큰 값입니다. 평균은 3개 값의 집계입니다.

소프트 제한(`memoryReservation`)을 지정하면 이는 예약된 메모리의 양을 계산하는 데 사용됩니다. 그렇지 않으면 하드 제한(`memory`)이 사용됩니다. 하드 한도 및 소프트 한도에 대한 자세한 내용은 태스크 크기 섹션을 참조하세요.

예를 들어 서비스의 태스크 정의는 모든 컨테이너에 대해 총 512개 CPU 단위 및 1,024MiB 메모리(하드 제한 `memory` 파라미터 사용)를 지정합니다. 이 서비스의 원하는 실행 작업 수는 1이고, 서비스는 1개의 `c4.large` 컨테이너 인스턴스(2,048개 CPU 단위와 총 메모리 3,768MiB)에서 실행되며, 클러스터에서 실행되는 다른 태스크는 없습니다. 작업에서 512개 CPU 단위를 지정하더라도 2,048개 CPU 유닛이 있는 컨테이너 인스턴스에서 실행 중인 작업이기 때문에 지정된 수(2,048 / 512)의 4배까지 사용할 수 있습니다. 그러니 지정된 1,024 MiB의 메모리는 하드 제한이라 초과할 수 없기 때문에 서비스 메모리 사용이 100%를 초과할 수 없습니다.

앞의 예제에서 하드 제한 `memoryReservation` 파라미터 대신 소프트 제한 `memory`을 사용한 경우, 필요하다면 서비스 태스크는 지정된 1,024MiB 메모리보다 더 많은 메모리를 사용할 수 있습니다. 이 경우 서비스의 메모리 사용률은 100%를 초과할 수 있습니다.

애플리케이션에서 짧은 시간 동안 메모리 사용률이 갑자기 급증하는 경우 Amazon ECS가 1분마다 여러 데이터 포인트를 수집한 다음 CloudWatch로 전송되는 하나의 데이터 포인트로 집계하기 때문에 서비스 메모리 사용률이 증가하는 것을 볼 수 없습니다.

작업이 일정 기간 동안 CPU 집약적 태스크를 수행하여 가용 CPU 단위 2,048개를 모두 사용하고 메모리 512MiB를 사용하는 경우 서비스가 CPU 사용률 400%와 메모리 사용률 50%를 보고합니다. 작업이 유휴 상태이고 128개 CPU 단위와 128MiB 메모리를 사용하는 경우 서비스가 CPU 사용률 25%와 메모리 사용률 12.5%를 보고합니다.

> **Note**
>
> 이 예에서 CPU 사용률은 CPU 단위가 컨테이너 수준에서 정의된 경우에만 100%를 초과합니다. 작업 수준에서 CPU 단위를 정의하면 사용률이 정의된 작업 수준 제한을 초과하지 않습니다.

서비스 수준 EBS 파일 시스템 사용률

서비스 수준 EBS 파일 시스템 사용률은 서비스에 속하는 작업이 사용 중인 EBS 파일 시스템 총 크기를 서비스에 속하는 모든 작업에 할당된 EBS 파일 시스템 스토리지의 총 크기로 나눈 값으로 측정됩니다.

서비스 **RUNNING** 작업 수

CloudWatch 지표를 사용하여 서비스에서 RUNNING 상태인 작업 수를 확인할 수 있습니다. 예를 들어 서비스 내 실행 작업 수가 지정된 값을 하회하면 경보를 작동하도록 이 지표에 CloudWatch 경보를 설정할 수 있습니다.

Amazon ECS CloudWatch Container Insights의 서비스 **RUNNING** 태스크 수

'실행 중인 태스크 수'(RunningTaskCount) 지표는 Amazon ECS CloudWatch Container Insights를 사용하는 경우 클러스터별 및 서비스별로 사용할 수 있습니다. Container Insights는 containerInsights 계정 설정을 옵트인하여 생성한 모든 새 클러스터에 사용할 수 있으며, 클러스터 생성 중 클러스터 설정을 켜서 개별 클러스터에 사용하거나 UpdateClusterSettings API를 사용하여 기존 클러스터에 사용할 수 있습니다. CloudWatch Container Insights에서 수집된 지표는 사용자 지정 지표로 청구됩니다. CloudWatch 요금에 대한 자세한 정보는 CloudWatch 요금을 참조하세요.

이 지표를 보려면 Amazon CloudWatch 사용 설명서의 Amazon ECS Container Insights 지표를 참조하세요.

Amazon ECS 서비스 사용률 지표 사용 사례

다음 목록은 Amazon ECS 지표를 사용하는 시기에 대한 정보를 제공합니다.

- 리소스 사용률 모니터링: 평균 통계를 사용하여 CPU 및 메모리 소비 패턴을 모니터링하고 성능 기준을 설정하며 점진적 성능 저하 추세를 감지합니다.
- 비용 최적화: 평균 통계를 사용하여 리소스 사용량을 모니터링하고 컨테이너 크기를 조정하며 사용 패턴을 기반으로 예약을 조정하고 예약된 조정을 구현합니다.

- 성능 벤치마킹: 평균 통계를 사용하여 서비스 개정 간 지표를 비교하고 성능 KPI를 설정하며 최적화 개선 사항을 검증합니다.
- 리소스 최소 수준 감지: 평균 통계를 사용하여 유휴 기간에 최소 리소스 요구 사항을 식별하고 적절한 예약을 설정하며 비정상적인 감소를 감지합니다.
- 이상 탐지: 최소 통계를 사용하여 초기화 실패 또는 예상치 못한 유휴 기간과 같은 잠재적 문제를 나타내는 비정상적인 리소스 사용률 저하를 찾아냅니다.
- 조정 정책 세분화: 최소 통계를 사용하여 공격적인 조정을 방지하기 위해 실행 가능한 최소 사용률을 기반으로 최적의 스케일 인 임계치를 설정합니다.
- 용량 계획: 최대 통계를 사용하여 적절한 태스크 크기를 설정하고 트래픽 스파이크에 충분한 여유 공간을 고려하여 인프라를 계획합니다.
- 성능 병목 현상 식별: 최대 통계를 사용하여 리소스 포화 지점을 감지하고 병목 현상을 식별하며 태스크 크기를 늘릴 시기를 결정합니다.
- 조정 정책 구성: 최대 통계를 사용하여 피크 패턴을 기반으로 최적의 스케일 아웃 임계치를 설정하고 버스트 용량을 구성합니다.
- SLA 규정 준수 모니터링: 최대 통계를 사용하여 피크 응답 시간과 오류율을 추적하여 서비스 성능이 정의된 SLA를 충족하는지 확인합니다.

관련 지표 정보

Container Insights에 대한 자세한 내용은 CloudWatch 사용 설명서의 Container Insights 사용 사례를 참조하세요.

EventBridge를 사용하여 Amazon ECS 오류에 대한 응답 자동화

Amazon EventBridge를 사용하면 AWS 서비스를 자동화하고 애플리케이션 가용성 문제나 리소스 변경 같은 시스템 이벤트에 자동으로 대응할 수 있습니다. AWS 서비스의 이벤트는 거의 실시간으로 EventBridge로 전송됩니다. 원하는 이벤트만 표시하도록 간단한 규칙을 작성한 후 규칙과 일치하는 이벤트 발생 시 실행할 자동화 태스크를 지정할 수 있습니다. 자동으로 구성할 수 있는 태스크는 다음과 같습니다.

- CloudWatch Logs의 로그 그룹에 이벤트 추가
- AWS Lambda 함수 호출
- Amazon EC2 Run Command 호출
- Amazon Kinesis Data Streams로 이벤트 릴레이
- AWS Step Functions 상태 머신 활성화

- Amazon SNS 주제 또는 Amazon Simple Queue Service(Amazon SQS) 대기열에 알림

자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge 시작하기를 참조하세요.

EventBridge용 Amazon ECS 이벤트를 사용하여 Amazon ECS 클러스터의 현재 상태에 관해 실시간에 가까운 알림을 받을 수 있습니다. 해당 작업에서 EC2 시작 유형을 사용 중인 경우, 컨테이너 인스턴스의 상태와 해당 컨테이너 인스턴스에서 실행되는 모든 작업의 현재 상태를 모두 확인할 수 있습니다. 작업에서 Fargate 시작 유형을 사용하는 경우 컨테이너 인스턴스 상태를 볼 수 있습니다.

EventBridge를 사용하여 클러스터 간 작업 조정을 수행하고 클러스터의 상태를 거의 실시간으로 모니터링하는 Amazon ECS에서 사용자 지정 스케줄러를 빌드할 수 있습니다. Amazon ECS 서비스에서 상태 변경을 지속적으로 폴링하는 예약 및 모니터링 코드를 생략하고, 대신 EventBridge 대상을 사용하여 Amazon ECS 상태 변경을 비동기적으로 처리할 수 있습니다. 대상은 AWS Lambda, Amazon Simple Queue Service, Amazon Simple Notification Service 또는 Amazon Kinesis Data Streams를 포함할 수 있습니다.

Amazon ECS 이벤트 스트림은 모든 이벤트를 한 번 이상 전달합니다. 중복 이벤트가 전송되는 경우, 해당 이벤트는 중복을 식별하기에 충분한 정보를 제공합니다. 자세한 정보는 Amazon ECS 이벤트 처리 섹션을 참조하세요.

이벤트는 상대적으로 순서가 지정되므로 한 이벤트가 다른 이벤트보다 빨리 또는 늦게 발생했는지 쉽게 알 수 있습니다.

주제

- Amazon ECS 이벤트
- Amazon ECS 이벤트 처리

Amazon ECS 이벤트

Amazon ECS는 각 작업 및 서비스의 상태를 추적합니다. 작업 또는 서비스 상태가 변경되면 이벤트가 생성되고 Amazon EventBridge로 전달됩니다. 이러한 이벤트는 작업 상태 변경 이벤트 및 서비스 작업 이벤트로 분류됩니다. 다음 섹션에서 이러한 이벤트 및 가능한 원인을 보다 자세하게 설명합니다.

Amazon ECS는 다음과 같은 유형의 이벤트를 생성하여 EventBridge로 전송합니다.

- 컨테이너 인스턴스 상태 변경
- 작업 상태 변경
- 배포 상태 변경

- 서비스 작업

> **Note**
>
> 향후 Amazon ECS가 다른 이벤트 유형, 소스 및 세부 정보를 추가할 수 있습니다. 코드의 이벤트 JSON 데이터를 역직렬화하는 경우 추가 속성이 추가되었을 때 문제를 방지하기 위해 애플리케이션이 알 수 없는 속성을 처리할 수 있도록 준비해야 합니다.

경우에 따라 동일한 활동에 대해 복수의 이벤트가 생성됩니다. 예를 들어 작업이 컨테이너 인스턴스에서 시작되면 새 작업에 대해 작업 상태 변경 이벤트가 생성됩니다. 컨테이너 인스턴스의 가용 리소스(예: CPU, 메모리 및 가용 포트)의 변동을 설명하기 위해 컨테이너 인스턴스 상태 변경 이벤트가 생성됩니다. 또한 컨테이너 인스턴스가 종료될 때도 컨테이너 인스턴스, 컨테이너 에이전트 연결 상태 및 해당 컨테이너 인스턴스에서 실행되는 모든 작업에 대해 이벤트가 생성됩니다.

컨테이너 상태 변경 및 작업 상태 변경 이벤트에는 두 개의 `version` 필드가 있습니다. 하나는 이벤트의 기본 본문에 있고 다른 하나는 이벤트의 `detail` 객체에 있습니다. 다음은 이러한 두 필드 간의 차이점에 대해 설명합니다.

- 이벤트의 본문 내 `version` 필드는 모든 이벤트에서 `0`으로 설정됩니다. Eventbridge 파라미터에 대한 자세한 내용은 Amazon EventBridge 사용 설명서의 AWS 서비스 이벤트 메타데이터를 참조하세요.
- 이벤트의 `detail` 객체 내 `version` 필드는 연결된 리소스의 버전을 설명합니다. 리소스 상태가 바뀔 때마다 이 버전이 증가합니다. 이벤트를 여러 번 보낼 수 있으므로 이 필드를 사용하여 중복 이벤트를 식별할 수 있습니다. 중복 이벤트는 `detail` 객체에서 동일한 버전을 갖습니다. EventBridge를 사용하여 Amazon ECS 컨테이너 인스턴스 및 작업 상태를 복제하는 경우, Amazon ECS API에서 보고된 리소스 버전과 리소스(`detail` 객체 내부)에 대해 EventBridge에서 보고된 버전을 비교하여 이벤트 스트림 버전이 최신임을 확인할 수 있습니다.

서비스 작업 이벤트는 본문의 `version` 필드만 포함합니다.

서비스 작업 이벤트는 2개의 필드에서 서비스를 지정합니다.

- `create-service`에서 생성된 이벤트의 경우 서비스는 `serviceName` 필드에 있습니다.
- `update-service`에서 생성된 이벤트의 경우 서비스는 `service` 필드에 있습니다.

서비스 이벤트에 자동 도구를 사용하는 경우 두 필드 모두에 대해 코드를 지정해야 합니다.

서비스 작업 이벤트에 대한 규칙을 생성하는 방법에 대한 자세한 내용은 Amazon ECS 서비스 작업 이벤트 섹션을 참조하세요.

Amazon ECS 및 EventBridge를 통합하는 방법에 대한 자세한 내용은 Amazon EventBridge 및 Amazon ECS 통합을 참조하세요.

Amazon ECS 컨테이너 인스턴스 상태 변경 이벤트

다음과 같은 상황에서 컨테이너 인스턴스 상태 변경 이벤트가 발생합니다.

직접 또는 `StartTask`이나 SDK를 사용하여 `RunTask`, `StopTask` 또는 AWS Management Console API 태스크를 호출합니다.

> 컨테이너 인스턴스에서 태스크를 배치 또는 중지하면 컨테이너 인스턴스의 가용 리소스(예: CPU, 메모리, 가용 포트)가 수정됩니다.

Amazon ECS 서비스 스케줄러가 태스크를 시작 또는 중지하는 경우

> 컨테이너 인스턴스에서 태스크를 배치 또는 중지하면 컨테이너 인스턴스의 가용 리소스(예: CPU, 메모리, 가용 포트)가 수정됩니다.

Amazon ECS 컨테이너 에이전트가 원하는 작업 상태 `RUNNING`의 작업에 대해 `STOPPED` 상태의 `SubmitTaskStateChange` API 태스크를 호출하는 경우

> Amazon ECS 컨테이너 에이전트는 컨테이너 인스턴스에서 작업 상태를 모니터링하며 모든 상태 변경을 보고합니다. `RUNNING` 상태여야 할 작업이 `STOPPED` 상태로 전환될 경우 에이전트가 중지한 작업에 할당된 리소스(예: CPU, 메모리, 가용 포트)를 해제합니다.

직접 또는 `DeregisterContainerInstance`이나 SDK를 사용하여 AWS Management Console API 태스크를 사용하여 컨테이너 인스턴스를 등록 해제합니다.

> 컨테이너 인스턴스를 등록 해제하면 해당 컨테이너 인스턴스의 상태와 Amazon ECS 컨테이너 에이전트의 연결 상태가 변경됩니다.

EC2 인스턴스를 중지했을 때 작업이 중지된 경우

> 컨테이너 인스턴스를 중지하면 해당 컨테이너 인스턴스에서 실행 중이던 작업이 `STOPPED` 상태로 전환됩니다.

Amazon ECS 컨테이너 에이전트가 컨테이너 인스턴스를 처음 등록하는 경우

> Amazon ECS 컨테이너가 처음 컨테이너 인스턴스를 등록하면(시작 시 또는 수동으로 처음 실행할 때) 해당 인스턴스에 대해 상태 변경 이벤트가 생성됩니다.

Amazon ECS 컨테이너 에이전트가 Amazon ECS를 연결하거나 연결을 해제하는 경우

Amazon ECS 컨테이너 에이전트가 Amazon ECS 백엔드에 연결되거나 연결이 해제되면 컨테이너 인스턴스의 `agentConnected` 상태가 변경됩니다.

> **Note**
>
> Amazon ECS 컨테이너 에이전트는 정상 작업의 일부로 시간당 여러 번 연결 해제와 재연결을 반복하므로 에이전트 연결 이벤트가 예상됩니다. 이러한 이벤트가 컨테이너 에이전트 또는 컨테이너 인스턴스에 문제가 있음을 나타내는 것은 아닙니다.

사용자가 인스턴스에서 Amazon ECS 컨테이너 에이전트를 업그레이드하는 경우

컨테이너 인스턴스 세부 정보에는 컨테이너 에이전트 버전 개체가 포함됩니다. 에이전트를 업그레이드하면 이 버전 정보가 변경되고 이벤트가 발생합니다.

Example 컨테이너 인스턴스 상태 변경 이벤트

컨테이너 인스턴스 상태 변경 이벤트는 다음 형식으로 전달됩니다. 다음 `detail` 섹션은 Amazon Elastic 컨테이너 서비스 API 참조의 DescribeContainerInstances API 작업에서 반환되는 ContainerInstance 객체와 유사합니다. Eventbridge 파라미터에 대한 자세한 내용은 Amazon EventBridge 사용 설명서의 AWS 서비스 이벤트 메타데이터를 참조하세요.

```
{
  "version": "0",
  "id": "8952ba83-7be2-4ab5-9c32-6687532d15a2",
  "detail-type": "ECS Container Instance State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2016-12-06T16:41:06Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ecs:us-east-1:111122223333:container-instance/
b54a2a04-046f-4331-9d74-3f6d7f6ca315"
  ],
  "detail": {
    "agentConnected": true,
    "attributes": [
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
```

```
      },
      {
        "name": "com.amazonaws.ecs.capability.task-iam-role-network-host"
      },
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.awslogs"
      },
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
      },
      {
        "name": "com.amazonaws.ecs.capability.privileged-container"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
      },
      {
        "name": "com.amazonaws.ecs.capability.ecr-auth"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.20"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.21"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.22"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.23"
      },
      {
        "name": "com.amazonaws.ecs.capability.task-iam-role"
      }
    ],
    "clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",
    "containerInstanceArn": "arn:aws:ecs:us-east-1:111122223333:container-instance/
b54a2a04-046f-4331-9d74-3f6d7f6ca315",
```

```
    "ec2InstanceId": "i-f3a8506b",
    "registeredResources": [
      {
        "name": "CPU",
        "type": "INTEGER",
        "integerValue": 2048
      },
      {
        "name": "MEMORY",
        "type": "INTEGER",
        "integerValue": 3767
      },
      {
        "name": "PORTS",
        "type": "STRINGSET",
        "stringSetValue": [
          "22",
          "2376",
          "2375",
          "51678",
          "51679"
        ]
      },
      {
        "name": "PORTS_UDP",
        "type": "STRINGSET",
        "stringSetValue": []
      }
    ],
    "remainingResources": [
      {
        "name": "CPU",
        "type": "INTEGER",
        "integerValue": 1988
      },
      {
        "name": "MEMORY",
        "type": "INTEGER",
        "integerValue": 767
      },
      {
        "name": "PORTS",
        "type": "STRINGSET",
        "stringSetValue": [
```

```
          "22",
          "2376",
          "2375",
          "51678",
          "51679"
        ]
      },
      {
        "name": "PORTS_UDP",
        "type": "STRINGSET",
        "stringSetValue": []
      }
    ],
    "status": "ACTIVE",
    "version": 14801,
    "versionInfo": {
      "agentHash": "aebcbca",
      "agentVersion": "1.13.0",
      "dockerVersion": "DockerVersion: 1.11.2"
    },
    "updatedAt": "2016-12-06T16:41:06.991Z"
  }
}
```

Amazon ECS 작업 상태 변경 이벤트

다음과 같은 상황에서 작업 상태 변경 이벤트가 발생합니다.

직접 또는 AWS Management Console, AWS CLI나 SDK를 사용하여 `StartTask`, `RunTask` 또는 `StopTask` API 태스크를 호출합니다.

태스크를 시작 또는 중지하면 새로운 작업 리소스가 생기거나 기존 작업 리소스의 상태가 수정됩니다.

Amazon ECS 서비스 스케줄러가 태스크를 시작 또는 중지하는 경우

태스크를 시작 또는 중지하면 새로운 작업 리소스가 생기거나 기존 작업 리소스의 상태가 수정됩니다.

Amazon ECS 컨테이너 에이전트가 `SubmitTaskStateChange` API 태스크를 호출하는 경우

Amazon EC2 시작 유형의 경우 Amazon ECS 컨테이너 에이전트는 컨테이너 인스턴스에서 작업 상태를 모니터링합니다. Amazon ECS 컨테이너 에이전트는 모든 상태 변경을 보고합니다. 상태 변

경에는 PENDING에서 RUNNING으로 또는 RUNNING에서 STOPPED로의 변경이 포함될 수 있습니다.

직접 또는 AWS Management Console이나 SDK를 통해 DeregisterContainerInstance API 태스크와 force 플래그를 사용하여 기본 컨테이너 인스턴스를 강제로 등록 해제합니다.

컨테이너 인스턴스를 등록 해제하면 해당 컨테이너 인스턴스의 상태와 Amazon ECS 컨테이너 에이전트의 연결 상태가 변경됩니다. 컨테이너 인스턴스에서 작업이 실행 중인 경우 등록을 해제하려면 force 플래그가 설정되어야 합니다. 그러면 인스턴스에서 모든 작업이 중지합니다.

기본 컨테이너 인스턴스가 중지 또는 종료되는 경우

컨테이너 인스턴스를 중지 또는 종료하면 해당 컨테이너 인스턴스에서 실행 중이던 작업이 STOPPED 상태로 전환됩니다.

작업 내 컨테이너의 상태가 변경되는 경우

Amazon ECS 컨테이너 에이전트는 작업 내 컨테이너의 상태를 모니터링합니다. 예를 들어 작업에서 실행 중인 컨테이너가 중지하는 경우 이 컨테이너 상태 변경은 이벤트를 발생시킵니다.

Fargate 스팟 용량 공급자를 사용하는 태스크는 종료 공지를 받습니다.

작업이 FARGATE_SPOT 용량 공급자를 사용 중이고 스팟 중단으로 인해 중지되면 작업 상태 변경 이벤트가 발생시킵니다.

Example 작업 상태 변경 이벤트

작업 상태 변경 이벤트는 다음 형식으로 제공됩니다. 다음은 이러한 필드에 대한 유의 사항입니다.

- 태스크 상태 변경 이벤트에서 이벤트의 상태를 사용할 수 없습니다. 태스크 상태가 필요한 경우 describe-tasks를 실행할 수 있습니다.
- 컨테이너에서 Amazon ECR로 호스팅되는 이미지를 사용하는 경우 imageDigest 필드가 반환됩니다.
- createdAt, connectivityAt, pullStartedAt, startedAt, pullStoppedAt 및 updatedAt 필드의 값은 ISO 문자열 타임스탬프입니다.
- detail-type 값은 'ECS 태스크 상태 변경'입니다.
- 중지된 태스크에 대해 이벤트가 생성되면 stoppedReason 및 stoppedCode 필드는 태스크가 중지된 이유에 대한 추가 정보를 제공합니다(예: '사용자 시작').

Eventbridge 파라미터에 대한 자세한 내용은 Amazon EventBridge 이벤트 참조의 AWS 서비스 이벤트 메타데이터를 참조하세요.

필수 컨테이너 중 하나가 종료되어 태스크 실행이 중지된 태스크 이벤트만 캡처하는 Amazon EventBridge 이벤트 규칙을 구성하는 방법에 대한 자세한 내용은 Amazon ECS 작업 중지 이벤트에 대한 Amazon Simple Notification Service 알림 보내기 섹션을 참조하세요.


```
{
    "version": "0",
    "id": "105f6bb1-4da6-c630-4965-35383018cbca",
    "detail-type": "ECS Task State Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2025-05-06T11:02:34Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/a1173316d40a45dea9"
    ],
    "detail": {
        "attachments": [
            {
                "id": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                "type": "eni",
                "status": "ATTACHED",
                "details": [
                    {
                        "name": "subnetId",
                        "value": "subnet-0d0eab1bb38d5ca64"
                    },
                    {
                        "name": "networkInterfaceId",
                        "value": "eni-0103a2f01bad57d71"
                    },
                    {
                        "name": "macAddress",
                        "value": "0e:50:d1:c1:77:81"
                    },
                    {
                        "name": "privateDnsName",
                        "value": "ip-10-0-1-163.ec2.internal"
                    },
                    {
                        "name": "privateIPv4Address",
```

```
                        "value": "10.0.1.163"
                    }
                ]
            }
        ],
        "attributes": [
            {
                "name": "ecs.cpu-architecture",
                "value": "x86_64"
            }
        ],
        "availabilityZone": "us-east-1b",
        "capacityProviderName": "FARGATE",
        "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/example-cluster",
        "connectivity": "CONNECTED",
        "connectivityAt": "2025-05-06T11:02:17.19Z",
        "containers": [
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example-
cluster/a1173316d40a45dea9/a0a99b87-baa8-4bf6-b9f1-a9a95917a635",
                "lastStatus": "RUNNING",
                "name": "web",
                "image": "nginx",
                "imageDigest":
 "sha256:c15da6c91de8d2f436196f3a768483ad32c258ed4e1beb3d367a27ed67253e66",
                "runtimeId": "a1173316d40a45dea9-0265927825",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
                "networkInterfaces": [
                    {
                        "attachmentId": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                        "privateIpv4Address": "10.0.1.163"
                    }
                ],
                "cpu": "99",
                "memory": "100"
            },
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example-
cluster/a1173316d40a45dea9/a2010e2d-ba7c-4135-8b79-e0290ff3cd8c",
                "lastStatus": "RUNNING",
                "name": "aws-guardduty-agent-nm40lC",
                "imageDigest":
 "sha256:bf9197abdf853607e5fa392b4f97ccdd6ca56dd179be3ce8849e552d96582ac8",
```

```
                "runtimeId": "a1173316d40a45dea9-2098416933",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
                "networkInterfaces": [
                    {
                        "attachmentId": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                        "privateIpv4Address": "10.0.1.163"
                    }
                ],
                "cpu": "null"
            },
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example-
cluster/a1173316d40a45dea9/dccf0ca2-d929-471f-a5c3-98006fd4379e",
                "lastStatus": "RUNNING",
                "name": "aws-otel-collector",
                "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.32.0",
                "imageDigest":
 "sha256:7a1b3560655071bcacd66902c20ebe9a69470d5691fe3bd36baace7c2f3c4640",
                "runtimeId": "a1173316d40a45dea9-4027662657",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
                "networkInterfaces": [
                    {
                        "attachmentId": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                        "privateIpv4Address": "10.0.1.163"
                    }
                ],
                "cpu": "0"
            }
        ],
        "cpu": "256",
        "createdAt": "2025-05-06T11:02:13.877Z",
        "desiredStatus": "RUNNING",
        "enableExecuteCommand": false,
        "ephemeralStorage": {
            "sizeInGiB": 20
        },
        "group": "family:webserver",
        "launchType": "FARGATE",
        "lastStatus": "RUNNING",
        "memory": "512",
        "overrides": {
            "containerOverrides": [
```

```
                {
                    "name": "web"
                },
                {
                    "environment": [
                        {
                            "name": "CLUSTER_NAME",
                            "value": "example-cluster"
                        },
                        {
                            "name": "REGION",
                            "value": "us-east-1"
                        },
                        {
                            "name": "HOST_PROC",
                            "value": "/host_proc"
                        },
                        {
                            "name": "AGENT_RUNTIME_ENVIRONMENT",
                            "value": "ecsfargate"
                        },
                        {
                            "name": "STAGE",
                            "value": "prod"
                        }
                    ],
                    "memory": 128,
                    "name": "aws-guardduty-agent-nm40lC"
                },
                {
                    "name": "aws-otel-collector"
                }
            ]
        },
        "platformVersion": "1.4.0",
        "pullStartedAt": "2025-05-06T11:02:24.162Z",
        "pullStoppedAt": "2025-05-06T11:02:33.493Z",
        "startedAt": "2025-05-06T11:02:34.325Z",
        "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/
webserver:5",
        "updatedAt": "2025-05-06T11:02:34.325Z",
        "version": 3
```

```
    }
}
```

Example

다음은 EC2 시작 유형에 대한 태스크 상태 변경 이벤트의 예입니다.

```
{
    "version": "0",
    "id": "a65cf262-f104-0dd5-ceda-4b09ba71a441",
    "detail-type": "ECS Task State Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2025-05-12T13:12:06Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ecs:us-east-1:123456789012:task/example/c1ffa94f19a540ed8d9f7e1d2a5d"
    ],
    "detail": {
        "attachments": [
            {
                "id": "52333e3b-b812-41a8-b057-9ed184bbe5e1",
                "type": "eni",
                "status": "ATTACHED",
                "details": [
                    {
                        "name": "subnetId",
                        "value": "subnet-0d0eab1bb38d5ca64"
                    },
                    {
                        "name": "networkInterfaceId",
                        "value": "eni-0ea90f746500773a4"
                    },
                    {
                        "name": "macAddress",
                        "value": "0e:d5:9b:ce:49:fb"
                    },
                    {
                        "name": "privateDnsName",
                        "value": "ip-10-0-1-37.ec2.internal"
                    },
                    {
                        "name": "privateIPv4Address",
                        "value": "10.0.1.37"
```

```
                    }
                ]
            }
        ],
        "attributes": [
            {
                "name": "ecs.cpu-architecture",
                "value": "x86_64"
            }
        ],
        "availabilityZone": "us-east-1b",
        "capacityProviderName": "Infra-ECS-Cluster-example-fa84e0cc-
AsgCapacityProvider-OseQJU9pizmp",
        "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/example",
        "connectivity": "CONNECTED",
        "connectivityAt": "2025-05-12T13:11:44.98Z",
        "containerInstanceArn": "arn:aws:ecs:us-east-1:123456789012:container-instance/
example/d1d84798400f49f3b21cb61610c1e",
        "containers": [
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example/
c1ffa94f19a540ed8d9f7e1d2a5d3626/197d0994-5367-4a6d-9f9a-f075e4a6",
                "lastStatus": "RUNNING",
                "name": "aws-otel-collector",
                "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.32.0",
                "imageDigest":
 "sha256:7a1b3560655071bcacd66902c20ebe9a69470d5691fe3bd36baace7c2f3c4640",
                "runtimeId":
 "8e926f0ccd8fe2b459926f49584ba6d33a3d9f61398dbabe944ee6a13a8ff3a1",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example/
c1ffa94f19a540ed8d9f7e1d2a5d",
                "networkInterfaces": [
                    {
                        "attachmentId": "52333e3b-b812-41a8-b057-9ed184bbe5e1",
                        "privateIpv4Address": "10.0.1.37"
                    }
                ],
                "cpu": "0"
            },
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example/
c1ffa94f19a540ed8d9f7e1d2a5d3626/cab39ef0-9c50-459d-844b-b9d51d73d",
                "lastStatus": "RUNNING",
                "name": "web",
```

```
                "image": "nginx",
                "imageDigest":
 "sha256:c15da6c91de8d2f436196f3a768483ad32c258ed4e1beb3d367a27ed67253e66",
                "runtimeId":
 "9f1c73f0094f051541d9e5c2ab1e172d83c4eb5171bcc857c4504b02770ff3b8",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example/
c1ffa94f19a540ed8d9f7e1d2a5d",
                "networkInterfaces": [
                    {
                        "attachmentId": "52333e3b-b812-41a8-b057-9ed184bbe5e1",
                        "privateIpv4Address": "10.0.1.37"
                    }
                ],
                "cpu": "99",
                "memory": "100"
            }
        ],
        "cpu": "256",
        "createdAt": "2025-05-12T13:11:44.98Z",
        "desiredStatus": "RUNNING",
        "enableExecuteCommand": false,
        "group": "family:webserver",
        "launchType": "EC2",
        "lastStatus": "RUNNING",
        "memory": "512",
        "overrides": {
            "containerOverrides": [
                {
                    "name": "aws-otel-collector"
                },
                {
                    "name": "web"
                }
            ]
        },
        "pullStartedAt": "2025-05-12T13:11:59.491Z",
        "pullStoppedAt": "2025-05-12T13:12:05.896Z",
        "startedAt": "2025-05-12T13:12:06.053Z",
        "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example/
c1ffa94f19a540ed8d9f7e1d2a5d",
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/
webserver",
        "updatedAt": "2025-05-12T13:12:06.053Z",
        "version": 4
```

```
    }
}
```

Amazon ECS 서비스 작업 이벤트

Amazon ECS는 세부 정보 유형 ECS 서비스 태스크와 함께 서비스 작업 이벤트를 보냅니다. 컨테이너 인스턴스 및 작업 상태 변경 이벤트와 달리 서비스 작업 이벤트는 `details` 응답 필드에 버전 번호를 포함하지 않습니다. 다음은 Amazon ECS 서비스 작업 이벤트에 대한 EventBridge 규칙을 만드는 데 사용되는 이벤트 패턴입니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge 시작하기를 참조하세요.

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Service Action"
    ]
}
```

Amazon ECS는 `INFO`, `WARN` 및 `ERROR`이벤트 유형과 함께 이벤트를 보냅니다. 다음은 서비스 작업 이벤트입니다.

`INFO` 이벤트 유형이 있는 서비스 작업 이벤트

`SERVICE_STEADY_STATE`

이 서비스는 건강하고 원하는 수의 작업으로 안정적인 상태에 도달합니다. 서비스 스케줄러는 주기적으로 상태를 보고하므로 이 메시지를 여러 번 받을 수 있습니다.

`TASKSET_STEADY_STATE`

작업 세트는 건강하고 원하는 수의 작업으로 안정적인 상태에 도달합니다.

`CAPACITY_PROVIDER_STEADY_STATE`

서비스와 관련된 용량 공급자가 안정적인 상태에 도달합니다.

`SERVICE_DESIRED_COUNT_UPDATED`

서비스 스케줄러가 서비스 또는 작업 세트에 대해 계산된 원하는 수를 업데이트하는 경우. 사용자가 원하는 수를 수동으로 업데이트하면 이 이벤트가 전송되지 않습니다.

TASKS_STOPPED

실행 중인 태스크가 서비스에서 중지되었습니다.

SERVICE_DEPLOYMENT_IN_PROGRESS

서비스 배포가 진행 중입니다. 서비스 배포가 롤백 또는 새 서비스 개정일 수 있습니다.

SERVICE_DEPLOYMENT_COMPLETED

서비스 배포가 안정 상태이고 완료되었습니다. 서비스 배포가 롤백이거나 업데이트된 서비스 개정을 배포하는 것일 수 있습니다.

WARN 이벤트 유형이 있는 서비스 작업 이벤트

SERVICE_TASK_START_IMPAIRED

서비스가 일부 태스크를 시작할 수 없습니다.

SERVICE_DISCOVERY_INSTANCE_UNHEALTHY

서비스 검색을 사용하는 서비스에 비정상 작업이 포함되어 있습니다. 서비스 스케줄러는 서비스 레지스트리 내의 작업이 비정상임을 감지합니다.

VPC_LATTICE_TARGET_UNHEALTHY

VPC Lattice가 사용되는 서비스에서 VPC Lattice의 대상 중 하나가 비정상이라는 것이 탐지되었습니다.

ERROR 이벤트 유형이 있는 서비스 작업 이벤트

SERVICE_DAEMON_PLACEMENT_CONSTRAINT_VIOLATED

DAEMON 서비스 스케줄러 전략을 사용하는 서비스의 작업이 더 이상 서비스에 대한 배치 제약 전략을 충족하지 않습니다.

ECS_OPERATION_THROTTLED

Amazon ECS API 스로틀 한도로 인해 서비스 스케줄러가 제한되었습니다.

SERVICE_DISCOVERY_OPERATION_THROTTLED

AWS Cloud Map API 스로틀 한도로 인해 서비스 스케줄러가 제한되었습니다. 이 문제는 서비스 검색을 사용하도록 구성된 서비스에서 발생할 수 있습니다.

`SERVICE_TASK_PLACEMENT_FAILURE`

서비스 스케줄러가 태스크를 배치할 수 없습니다. 원인은 `reason` 필드에 설명됩니다.

이 서비스 이벤트가 발생하는 일반적인 원인은 태스크를 배치할 클러스터의 리소스가 부족하기 때문입니다. 예를 들어 사용 가능한 컨테이너 인스턴스의 CPU 또는 메모리 용량이 충분하지 않거나 사용 가능한 컨테이너 인스턴스가 없는 경우입니다. 또 다른 일반적인 원인은 Amazon ECS 컨테이너 에이전트가 컨테이너 인스턴스에서 연결 해제되어 스케줄러가 태스크를 배치 할 수 없는 경우입니다.

`SERVICE_TASK_CONFIGURATION_FAILURE`

서비스 스케줄러가 구성 오류로 인해 태스크를 배치할 수 없습니다. 원인은 `reason` 필드에 설명됩니다.

이 서비스 이벤트가 발생하는 일반적인 원인은 태그가 서비스에 적용되었지만 사용자 또는 역할이 해당 지역의 새로운 Amazon 리소스 이름(ARN) 형식을 선택하지 않았기 때문입니다. 자세한 정보는 Amazon 리소스 이름(ARN) 및 ID을 참조하세요. 또 다른 일반적인 원인은 제공된 작업 IAM 역할을 Amazon ECS가 맡을 수 없기 때문입니다.

`SERVICE_HEALTH_UNKNOWN`

서비스에서 태스크의 상태 데이터가 설명되지 못했습니다.

`SERVICE_DEPLOYMENT_FAILED`

서비스 배포가 안정에 도달하지 않았습니다. 이는 CloudWatch가 트리거되거나 회로 차단기에서 서비스 배포 실패가 탐지되면 발생합니다.

Example 서비스 안정 상태 이벤트

서비스 안정 상태 이벤트는 다음과 같은 형식으로 제공됩니다. EventBridge 파라미터에 대한 자세한 내용은 Amazon EventBridge 사용 설명서의 EventBridge의 이벤트를 참조하세요.

```
{
    "version": "0",
    "id": "af3c496d-f4a8-65d1-70f4-a69d52e9b584",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:27:22Z",
    "region": "us-west-2",
    "resources": [
```

```
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_STEADY_STATE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "createdAt": "2019-11-19T19:27:22.695Z"
    }
}
```

Example 용량 공급자 안정 상태 이벤트

용량 공급자 안정 상태 이벤트는 다음 형식으로 제공됩니다.

```
{
    "version": "0",
    "id": "b9baa007-2f33-0eb1-5760-0d02a572d81f",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:37:00Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "CAPACITY_PROVIDER_STEADY_STATE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "capacityProviderArns": [
            "arn:aws:ecs:us-west-2:111122223333:capacity-provider/ASG-tutorial-
capacity-provider"
        ],
        "createdAt": "2019-11-19T19:37:00.807Z"
    }
}
```

Example 서비스 작업 시작 장애 이벤트

서비스 작업 시작 장애 이벤트는 다음 형식으로 제공됩니다.

```
{
    "version": "0",
```

```
    "id": "57c9506e-9d21-294c-d2fe-e8738da7e67d",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "WARN",
        "eventName": "SERVICE_TASK_START_IMPAIRED",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "createdAt": "2019-11-19T19:55:38.725Z"
    }
}
```

Example 서비스 작업 배치 실패 이벤트

서비스 작업 배치 실패 이벤트는 다음 형식으로 제공됩니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge 이벤트를 참조하세요.

다음 예제에서는 작업이 FARGATE_SPOT 용량 공급자를 사용하려고 시도했지만 서비스 스케줄러가 Fargate 스팟 용량을 획득할 수 없습니다.

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3a6d0468b",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "ERROR",
        "eventName": "SERVICE_TASK_PLACEMENT_FAILURE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "capacityProviderArns": [
            "arn:aws:ecs:us-west-2:111122223333:capacity-provider/FARGATE_SPOT"
        ],
```

```
        "reason": "RESOURCE:FARGATE",
        "createdAt": "2019-11-06T19:09:33.087Z"
    }
}
```

EC2 시작 유형에 대한 다음 예에서는 컨테이너 인스턴스 2dd1b186f39845a584488d2ef155c131에서 태스크를 시작하려고 했지만 CPU가 부족하여 서비스 스케줄러가 태스크를 배치할 수 없습니다.

```
{
  "version": "0",
  "id": "ddca6449-b258-46c0-8653-e0e3a6d0468b",
  "detail-type": "ECS Service Action",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2019-11-19T19:55:38Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
  ],
  "detail": {
    "eventType": "ERROR",
    "eventName": "SERVICE_TASK_PLACEMENT_FAILURE",
    "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
    "containerInstanceArns": [
    "arn:aws:ecs:us-west-2:111122223333:container-instance/
default/2dd1b186f39845a584488d2ef155c131"
    ],
    "reason": "RESOURCE:CPU",
    "createdAt": "2019-11-06T19:09:33.087Z"
  }
}
```

Amazon ECS 서비스 배포 상태 변경 이벤트

Amazon ECS는 세부 정보 유형 ECS 배포 상태 변경과 함께 서비스 배포 변경 상태 이벤트를 보냅니다. 다음은 Amazon ECS 서비스 배포 상태 변경 이벤트에 대한 EventBridge 규칙을 만드는 데 사용되는 이벤트 패턴입니다. EventBridge 규칙 생성에 대한 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge 시작하기를 참조하세요.

```
{
    "source": [
```

```
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Deployment State Change"
    ]
}
```

Amazon ECS는 INFO 및 ERROR 이벤트 유형과 함께 이벤트를 보냅니다. 자세한 내용은 Amazon ECS 서비스 작업 이벤트 섹션을 참조하세요.

서비스 배포 상태 변경 이벤트는 다음과 같습니다.

SERVICE_DEPLOYMENT_IN_PROGRESS

서비스 배포가 진행 중입니다. 이 이벤트는 초기 배포 및 롤백 배포 모두에 대해 전송됩니다.

SERVICE_DEPLOYMENT_COMPLETED

서비스 배포가 완료되었습니다. 배포 후 서비스가 안정적인 상태에 도달하면 이 이벤트가 전송됩니다.

SERVICE_DEPLOYMENT_FAILED

서비스 배포가 실패했습니다. 이 이벤트는 배포 회로 차단기 논리가 켜진 서비스에 대해 전송됩니다.

Example 서비스 배포 진행 중 이벤트

서비스 배포 진행 중 이벤트는 초기 배포와 롤백 배포가 모두 시작될 때 전달됩니다. 두 항목의 차이점은 reason 필드에 있습니다. Eventbridge 파라미터에 대한 자세한 내용은 Amazon EventBridge 사용 설명서의 AWS 서비스 이벤트 메타데이터를 참조하세요.

다음은 초기 배포가 시작되는 경우의 출력 예입니다.

```
{
   "version": "0",
   "id": "ddca6449-b258-46c0-8653-e0e3a6EXAMPLE",
   "detail-type": "ECS Deployment State Change",
   "source": "aws.ecs",
   "account": "111122223333",
   "time": "2020-05-23T12:31:14Z",
   "region": "us-west-2",
   "resources": [
```

```
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
        "deploymentId": "ecs-svc/123",
        "updatedAt": "2020-05-23T11:11:11Z",
        "reason": "ECS deployment deploymentId in progress."
    }
}
```

다음은 롤백 배포가 시작되는 경우의 출력 예입니다. `reason` 필드는 서비스가 롤백되는 배포의 ID를 제공합니다.

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
    "detail-type": "ECS Deployment State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-05-23T12:31:14Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
        "deploymentId": "ecs-svc/123",
        "updatedAt": "2020-05-23T11:11:11Z",
        "reason": "ECS deployment circuit breaker: rolling back to
 deploymentId deploymentID."
    }
}
```

Example 서비스 배포 완료 이벤트

서비스 배포 완료 상태 이벤트는 다음과 같은 형식으로 제공됩니다. 자세한 정보는 태스크를 대체하여 Amazon ECS 서비스 배포 섹션을 참조하세요.

```
{
    "version": "0",
```

```
    "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
    "detail-type": "ECS Deployment State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-05-23T12:31:14Z",
    "region": "us-west-2",
    "resources": [
         "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
         "eventType": "INFO",
         "eventName": "SERVICE_DEPLOYMENT_COMPLETED",
         "deploymentId": "ecs-svc/123",
         "updatedAt": "2020-05-23T11:11:11Z",
         "reason": "ECS deployment deploymentID completed."
    }
}
```

Example 서비스 배포 실패 이벤트

서비스 배포 실패 상태 이벤트는 다음과 같은 형식으로 제공됩니다. 서비스 배포 실패 상태 이벤트는 배포 회로 차단기 논리가 켜진 서비스에 대해서만 전송됩니다. 자세한 내용은 태스크를 대체하여 Amazon ECS 서비스 배포 섹션을 참조하세요.

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
    "detail-type": "ECS Deployment State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-05-23T12:31:14Z",
    "region": "us-west-2",
    "resources": [
         "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
         "eventType": "ERROR",
         "eventName": "SERVICE_DEPLOYMENT_FAILED",
         "deploymentId": "ecs-svc/123",
         "updatedAt": "2020-05-23T11:11:11Z",
         "reason": "ECS deployment circuit breaker: task failed to start."
    }
}
```

Amazon ECS 이벤트 처리

Amazon ECS는 이벤트를 한 번 이상 전송합니다. 즉, 지정된 이벤트의 복사본을 여러 개 받을 수 있습니다. 또한 이벤트가 이벤트 발생 순서대로 이벤트 리스너에게 전달되지 않을 수 있습니다.

이벤트 순서가 적절히 지정될 수 있도록 각 이벤트의 `detail` 섹션에 `version` 속성이 포함됩니다. 리소스 상태가 변경될 때마다 이 `version`이 증가합니다. 중복 이벤트는 `detail` 객체에서 동일한 `version`을 갖습니다. EventBridge를 사용하여 Amazon ECS 컨테이너 인스턴스 및 작업 상태를 복제하는 경우, Amazon ECS API에서 보고된 리소스 버전과 리소스에 대해 EventBridge에서 보고된 `version`을 비교하여 이벤트 스트림 버전이 최신임을 확인할 수 있습니다. 버전 속성 번호가 높은 이벤트는 낮은 버전 번호의 이벤트보다 나중에 발생한 것으로 처리해야 합니다.

예제: AWS Lambda 함수에서 이벤트 처리

다음 예제는 Python 3.9로 작성된 Lambda 함수로서, 작업과 컨테이너 인스턴스 상태 변경 이벤트를 모두 캡처하여 두 개의 Amazon DynamoDB 테이블 중 하나에 저장합니다.

- ECSCtrInstanceState – 컨테이너 인스턴스의 최신 상태를 저장합니다. 테이블 ID는 컨테이너 인스턴스의 `containerInstanceArn` 값입니다.
- ECSTaskState – 작업의 최신 상태를 저장합니다. 테이블 ID는 작업의 `taskArn` 값입니다.

```
import json
import boto3

def lambda_handler(event, context):
    id_name = ""
    new_record = {}

    # For debugging so you can see raw event format.
    print('Here is the event:')
    print((json.dumps(event)))

    if event["source"] != "aws.ecs":
       raise ValueError("Function only supports input from events with a source type
 of: aws.ecs")

    # Switch on task/container events.
    table_name = ""
    if event["detail-type"] == "ECS Task State Change":
        table_name = "ECSTaskState"
```

```
        id_name = "taskArn"
        event_id = event["detail"]["taskArn"]
    elif event["detail-type"] == "ECS Container Instance State Change":
        table_name = "ECSCtrInstanceState"
        id_name =  "containerInstanceArn"
        event_id = event["detail"]["containerInstanceArn"]
    else:
        raise ValueError("detail-type for event is not a supported type. Exiting
without saving event.")

    new_record["cw_version"] = event["version"]
    new_record.update(event["detail"])

    # "status" is a reserved word in DDB, but it appears in containerPort
    # state change messages.
    if "status" in event:
        new_record["current_status"] = event["status"]
        new_record.pop("status")


    # Look first to see if you have received a newer version of an event ID.
    # If the version is OLDER than what you have on file, do not process it.
    # Otherwise, update the associated record with this latest information.
    print("Looking for recent event with same ID...")
    dynamodb = boto3.resource("dynamodb", region_name="us-east-1")
    table = dynamodb.Table(table_name)
    saved_event = table.get_item(
        Key={
            id_name : event_id
        }
    )
    if "Item" in saved_event:
        # Compare events and reconcile.
        print(("EXISTING EVENT DETECTED: Id " + event_id + " - reconciling"))
        if saved_event["Item"]["version"] < event["detail"]["version"]:
            print("Received event is a more recent version than the stored event -
updating")
            table.put_item(
                Item=new_record
            )
        else:
            print("Received event is an older version than the stored event -
ignoring")
    else:
```

```
        print(("Saving new event - ID " + event_id))

        table.put_item(
            Item=new_record
        )
```

다음 Fargate 예제는 Python 3.9로 작성된 Lambda 함수로, 작업 상태 변경 이벤트를 캡처하여 다음 Amazon DynamoDB 테이블에 저장합니다.

```
import json
import boto3

def lambda_handler(event, context):
    id_name = ""
    new_record = {}

    # For debugging so you can see raw event format.
    print('Here is the event:')
    print((json.dumps(event)))

    if event["source"] != "aws.ecs":
       raise ValueError("Function only supports input from events with a source type
 of: aws.ecs")

    # Switch on task/container events.
    table_name = ""
    if event["detail-type"] == "ECS Task State Change":
        table_name = "ECSTaskState"
        id_name = "taskArn"
        event_id = event["detail"]["taskArn"]
    else:
        raise ValueError("detail-type for event is not a supported type. Exiting
 without saving event.")

    new_record["cw_version"] = event["version"]
    new_record.update(event["detail"])

    # "status" is a reserved word in DDB, but it appears in containerPort
    # state change messages.
    if "status" in event:
        new_record["current_status"] = event["status"]
        new_record.pop("status")
```

```
    # Look first to see if you have received a newer version of an event ID.
    # If the version is OLDER than what you have on file, do not process it.
    # Otherwise, update the associated record with this latest information.
    print("Looking for recent event with same ID...")
    dynamodb = boto3.resource("dynamodb", region_name="us-east-1")
    table = dynamodb.Table(table_name)
    saved_event = table.get_item(
        Key={
            id_name : event_id
        }
    )
    if "Item" in saved_event:
        # Compare events and reconcile.
        print(("EXISTING EVENT DETECTED: Id " + event_id + " - reconciling"))
        if saved_event["Item"]["version"] < event["detail"]["version"]:
            print("Received event is a more recent version than the stored event -
updating")
            table.put_item(
                Item=new_record
            )
        else:
            print("Received event is an older version than the stored event -
ignoring")
    else:
        print(("Saving new event - ID " + event_id))

        table.put_item(
            Item=new_record
        )
```

관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링

Container Insights는 컨테이너 애플리케이션 및 마이크로서비스의 지표 및 로그를 수집하고 종합하며 요약합니다. 관찰성이 개선된 Container Insights는 모든 Container Insights 지표와 더불어, 추가 태스크 및 컨테이너 지표를 제공합니다.

Container Insights는 클러스터에서 실행 중인 모든 컨테이너를 검색하고 성능 스택의 모든 계층에서 성능 데이터를 수집합니다. 운영 데이터는 성능 로그 이벤트로 수집됩니다. 이들은 카디널리티가 높은 데이터를 적정 규모로 수집 및 저장할 수 있는 정형 JSON 스키마를 사용하는 항목입니다. 이러한 데이

터를 토대로 CloudWatch는 CloudWatch 지표로서 클러스터, 서비스 및 태스크 수준에서 상위 수준의 집계 지표를 생성합니다. 이 지표에는 CPU, 메모리, 디스크, 네트워크 같은 리소스 사용률이 포함되어 있습니다. 지표는 CloudWatch 자동 대시보드에서 볼 수 있습니다.

이러한 지표는 지정된 시간 범위 동안 실행 중인 작업이 있는 리소스만 반영합니다. 예를 들어, 하나의 서비스가 포함된 클러스터에서 해당 서비스에 `RUNNING` 상태의 작업이 없는 경우 CloudWatch에 지표가 전송되지 않습니다. 두 개의 서비스가 있는데 그 중 하나에는 실행 중인 작업이 있고 다른 하나에는 없는 경우, 실행 중인 작업이 있는 서비스에 대한 지표만 전송됩니다.

2024년 12월 2일 AWS에서는 Amazon ECS의 향상된 관찰성을 갖춘 Container Insights를 출시했습니다. 이 버전은 Amazon EC2 및 Fargate 시작 유형을 사용하여 Amazon ECS 클러스터의 관찰성을 향상할 수 있도록 지원합니다. Amazon ECS에서 개선된 관찰성으로 Container Insights를 구성하면, Container Insights는 클러스터 수준에서 환경의 컨테이너 수준까지 상세한 인프라 원격 측정을 자동으로 수집하고, 이러한 중요한 성능 데이터를 큐레이트된 대시보드에 표시하므로 관찰성 설정에서 가중되는 과도한 부담을 없앱니다. 개선된 관찰성을 사용하여 Container Insights를 설정하는 방법에 대한 자세한 내용은 관찰성이 향상된 Container Insights 섹션을 참조하세요.

Container Insights 대신 관찰성이 향상된 Container Insights를 사용하는 것이 좋습니다. 컨테이너 환경에서 세부적인 가시성을 제공하여 평균 해결 시간을 단축하기 때문입니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 향상된 관찰성 지표를 갖춘 Amazon ECS Container Insights를 참조하세요.

확인할 수 있는 이벤트는 Amazon ECS가 Amazon EventBridge로 보낸 이벤트입니다. 자세한 내용은 Amazon ECS 이벤트를 참조하세요.

> **Important**
>
> CloudWatch Container Insights에서 수집된 지표는 사용자 지정 지표로 청구됩니다. CloudWatch 요금에 대한 자세한 정보는 CloudWatch 요금을 참조하세요.

컨테이너 상태 확인을 사용하여 Amazon ECS 작업 상태 확인

태스크 정의를 생성할 때 컨테이너의 상태 확인을 구성할 수 있습니다. 상태 확인은 컨테이너에서 로컬로 실행되며 애플리케이션 상태 및 가용성을 검증하는 명령입니다.

Amazon ECS 컨테이너 에이전트는 태스크 정의에 지정된 상태 확인만 모니터링하여 보고합니다. Amazon ECS는 컨테이너 이미지에 포함되었지만 컨테이너 정의에 지정되지 않은 Docker 상태 확인을

모니터링하지 않습니다. 컨테이너 정의에 지정된 상태 확인 파라미터는 컨테이너 이미지에 존재하는 Docker 상태 확인을 재정의합니다.

작업 정의에 상태 확인이 정의된 경우 컨테이너는 컨테이너 내에서 상태 확인 프로세스를 실행하고, 종료 코드를 평가하여 애플리케이션 상태를 확인합니다.

상태 확인은 다음 파라미터로 구성됩니다.

- 명령 - 정상 상태인지 확인하기 위해 컨테이너가 실행하는 명령입니다. 명령 인수를 직접 실행하려면 문자열 어레이가 `CMD`으로 시작하면 되고, 컨테이너의 기본 셸에서 명령을 실행하려면 `CMD-SHELL`로 시작하면 됩니다. 파이프, 리디렉션, 명령 체인 또는 환경 변수 확장과 같은 쉘 기능이 필요한 경우 `CMD-SHELL`을 사용합니다. 예를 들어 `CMD-SHELL`을 사용하면 쉘이 `||` 연산자를 해석하는 `curl -f http://localhost/ || exit 1`과 같은 명령을 사용할 수 있습니다. 쉘 해석이 필요하지 않은 간단한 명령에는 `CMD`를 사용합니다.
- 간격 - 각 상태 확인 사이의 시간(초)입니다.
- 제한 시간 - 실패로 간주되기 전에 상태 확인이 성공하기까지의 대기 시간(초)입니다.
- 재시도 - 컨테이너 상태가 비정상이라고 간주되기 전에 실패한 상태 확인을 재시도하는 횟수입니다.
- 시작 기간 - 실패한 상태 확인이 최대 재시도 횟수에 포함되기 전에 컨테이너에 부트스트랩 시간이 제공되는 유예 기간 옵션입니다.

  상태 검사가 `startPeriod` 내에 성공한 경우 컨테이너는 정상으로 간주되고 모든 후속 실패는 최대 재시도 횟수에 포함됩니다.

작업 정의에서 상태 확인을 지정하는 방법에 대한 자세한 내용은 상태 확인 섹션을 참조하세요.

다음은 컨테이너에 대해 가능한 상태 값에 대한 설명입니다.

- `HEALTHY` - 컨테이너 상태 확인을 성공적으로 통과했습니다.
- `UNHEALTHY` - 컨테이너 상태 확인에 실패했습니다.
- `UNKNOWN` - 컨테이너 상태 확인을 평가 중이거나, 컨테이너 상태 확인이 정의되지 않았거나, Amazon ECS에서 컨테이너 상태를 알 수 없습니다.

상태 확인 명령은 컨테이너에서 실행됩니다. 따라서 컨테이너 이미지에 명령을 포함해야 합니다.

상태 확인은 `localhost` 또는 `127.0.0.1`에서 컨테이너의 루프백 인터페이스를 통해 애플리케이션에 연결됩니다. `0`의 종료 코드는 성공을 나타내고, `0`이 아닌 다른 코드는 실패를 나타냅니다.

컨테이너 상태 확인을 사용할 때 다음을 고려합니다.

- 컨테이너 상태 확인을 위해서는 버전이 1.17.0 이상인 Amazon ECS 컨테이너 에이전트가 필요합니다.
- Linux 플랫폼 버전 `1.1.0` 이상 또는 Windows 플랫폼 버전 `1.1.0` 이상을 사용하는 경우 Fargate 작업에서 컨테이너 상태 확인이 지원됩니다.

Amazon ECS에서 작업 상태를 확인하는 방법

작업 정의에 상태 확인 명령이 있는 필수 컨테이너는 작업 상태를 확인할 때 고려할 유일한 컨테이너입니다.

다음 규칙이 순서대로 평가됩니다.

1. 하나의 필수 컨테이너 상태가 `UNHEALTHY`인 경우 작업 상태는 `UNHEALTHY`입니다.
2. 하나의 필수 컨테이너 상태가 `UNKNOWN`인 경우 작업 상태는 `UNKNOWN`입니다.
3. 모든 필수 컨테이너 상태가 `HEALTHY`인 경우 작업 상태는 `HEALTHY`입니다.

필수 컨테이너 2개가 포함된 다음 작업 상태 예제를 고려합니다.

컨테이너 1 상태	컨테이너 2 상태	작업 상태
UNHEALTHY	UNKNOWN	UNHEALTHY
UNHEALTHY	HEALTHY	UNHEALTHY
HEALTHY	UNKNOWN	UNKNOWN
HEALTHY	HEALTHY	HEALTHY

컨테이너 3개가 포함된 다음 작업 상태 예제를 고려합니다.

컨테이너 1 상태	컨테이너 2 상태	컨테이너 3 상태	작업 상태
UNHEALTHY	UNKNOWN	UNKNOWN	UNHEALTHY
UNHEALTHY	UNKNOWN	HEALTHY	UNHEALTHY
UNHEALTHY	HEALTHY	HEALTHY	UNHEALTHY

컨테이너 1 상태	컨테이너 2 상태	컨테이너 3 상태	작업 상태
HEALTHY	UNKNOWN	HEALTHY	UNKNOWN
HEALTHY	UNKNOWN	UNKNOWN	UNKNOWN
HEALTHY	HEALTHY	HEALTHY	HEALTHY

에이전트 연결 끊김이 상태 확인에 미치는 영향

Amazon ECS 컨테이너 에이전트가 Amazon ECS 서비스와의 연결이 끊어져도 컨테이너는 UNHEALTHY 상태로 전환되지 않습니다. 에이전트가 재시작되거나 일시적으로 사용할 수 없는 동안에도 컨테이너가 계속 실행되도록 하기 위한 것입니다. 상태 확인 상태는 Amazon ECS 에이전트의 '마지막으로 들은' 응답이므로 연결이 끊기기 전에 컨테이너가 HEALTHY로 간주된 경우 에이전트가 다시 연결되고 다른 상태 확인이 발생할 때까지 해당 상태가 유지됩니다. 컨테이너 상태 확인의 상태에 대한 가정은 하지 않습니다.

Amazon ECS 컨테이너 상태 확인

콘솔에서 컨테이너 상태를 확인하고 DescribeTasks 응답에서 API를 사용할 수 있습니다. 자세한 내용은 Amazon Elastic Container Service API 참조의 DescribeTasks를 참조하세요.

Amazon CloudWatch Logs와 같이 컨테이너에 대한 로깅을 사용하는 경우 컨테이너 상태 출력을 로그로 전달하도록 상태 확인 명령을 구성할 수 있습니다. stdout 및 stderr 정보를 모두 포착하려면 2&1을 사용해야 합니다. 다음 예제에서는 파이프(>>) 및 논리적 OR(||) 연산자와 같은 쉘 기능이 필요하기 때문에 CMD-SHELL을 사용합니다.

```
"command": [
     "CMD-SHELL",
     "curl -f http://localhost/ >> /proc/1/fd/1 2>&1  || exit 1"
   ],
```

Amazon ECS 컨테이너 인스턴스 상태 모니터링

Amazon ECS는 컨테이너 인스턴스 상태 모니터링을 제공합니다. Amazon ECS가 컨테이너 인스턴스에서 컨테이너를 실행하지 못하게 할 수 있는 문제를 감지했는지를 빠르게 확인할 수 있습니다. Amazon ECS는 문제를 식별하기 위해 에이전트 버전 1.57.0 이상으로 실행 중인 모든 컨테이너 인스

턴스에 대해 자동 검사를 수행합니다. 컨테이너 인스턴스의 에이전트 버전 확인에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

AWS CLI 버전 1.22.3 이상 또는 AWS CLI 버전 2.3.6 이상을 사용해야 합니다. AWS CLI 업데이트 방법에 대한 자세한 정보는 AWS Command Line Interface 버전 2 사용 설명서의 최신 버전의 AWS CLI 설치 또는 업데이트를 참조하세요.

CONTAINER_INSTANCE_HEALTH 옵션과 함께 describe-container-instances를 실행하여 컨테이너 인스턴스를 보는 방법.

overallStatus의 유효한 값은 다음과 같습니다.

- OK
- IMPAIRED
- INSUFFICIENT_DATA
- INITIALIZING

다음은 describe-container-instances를 실행하는 방법을 보여주는 예제입니다.

```
aws ecs describe-container-instances \
     --cluster cluster_name \
     --container-instances 47279cd2cadb41cbaef2dcEXAMPLE \
     --include CONTAINER_INSTANCE_HEALTH
```

다음은 출력에 있는 상태 객체의 예입니다.

```
"healthStatus": {
 "overallStatus": "OK",
 "details": [{
  "type": "CONTAINER_RUNTIME",
  "status": "OK",
  "lastUpdated": "2021-11-10T03:30:26+00:00",
  "lastStatusChange": "2021-11-10T03:26:41+00:00"
 }]
}
```

컨테이너 인스턴스 상태 문제

OK 이외의 overallStatus 상태가 있을 경우 다음을 시도하세요.

- `describe-container-instances` 대기 후 실행
- EC2 콘솔에서 또는 CLI를 사용하여 컨테이너 인스턴스 상태를 확인합니다.
- CloudWatch 지표 검토 자세한 내용은 CloudWatch를 사용하여 Amazon ECS 모니터링 섹션을 참조하세요.
- AWS Health Dashboard를 확인하여 서비스에 알려진 문제가 있는지 확인합니다.

애플리케이션 추적 데이터를 사용하여 Amazon ECS 최적화 기회 식별

Amazon ECS는 AWS Distro for OpenTelemetry와 통합되어 애플리케이션에서 추적 데이터를 수집합니다. Amazon ECS는 AWS Distro for OpenTelemetry 사이드카 컨테이너를 사용하여 추적 데이터를 수집하고 AWS X-Ray로 라우팅합니다. 자세한 정보는 Amazon ECS에서 AWS Distro for OpenTelemetry Collector 설정을 참조하세요. 그런 다음, AWS X-Ray를 사용하여 오류 및 예외를 식별하고 성능 병목 현상과 응답 시간을 분석할 수 있습니다.

AWS Distro for OpenTelemetry Collector가 AWS X-Ray로 추적 데이터를 전송하려면 추적 데이터를 생성하도록 애플리케이션을 구성해야 합니다. 자세한 정보는 AWS X-Ray 개발자 안내서의 AWS X-Ray용 애플리케이션 계측을 참조하세요.

AWS Distro for OpenTelemetry와 AWS X-Ray 통합에 필요한 IAM 권한

Amazon ECS와 AWS Distro for OpenTelemetry를 통합하려면 태스크 역할을 생성하고 태스크 정의에서 역할을 지정해야 합니다. 컨테이너 로그를 CloudWatch Logs로 라우팅하도록 AWS Distro for OpenTelemetry 사이드카를 구성하는 것이 좋습니다.

> **Important**
>
> 또한 AWS Distro for OpenTelemetry 통합을 사용하여 애플리케이션 지표를 수집하는 경우 태스크 IAM 역할에 해당 통합에 필요한 권한도 포함되어 있는지 확인합니다. 자세한 내용은 애플리케이션 지표를 사용하여 Amazon ECS 애플리케이션 성능 상호 연관 섹션을 참조하세요.

다음 정책을 생성한 다음 태스크 실행 역할에 연결합니다.

JSON 정책 편집기를 사용하여 정책을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 창에서 정책을 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작을 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. 정책 편집기 섹션에서 JSON 옵션을 선택합니다.
5. 다음 JSON 정책 문서를 입력합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "logs:PutRetentionPolicy",
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        }
    ]
}
```

6. 다음을 선택합니다.

> **Note**
>
> 언제든지 시각적 편집기 옵션과 JSON 편집기 옵션 간에 전환할 수 있습니다. 그러나 변경을 적용하거나 시각적 편집기에서 다음을 선택한 경우 IAM은 시각적 편집기에 최적화되

도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 정책 재구성을 참조하세요.

7. 검토 및 생성 페이지에서 생성하는 정책에 대한 정책 이름과 설명(선택 사항)을 입력합니다. 이 정책에 정의된 권한을 검토하여 정책이 부여한 권한을 확인합니다.
8. 정책 생성을 선택하고 새로운 정책을 저장합니다.

태스크 정의에서 AWS X-Ray 통합을 위한 AWS Distro for OpenTelemetry 사이드카 지정

Amazon ECS 콘솔은 추적 수집 사용 옵션으로 AWS Distro for OpenTelemetry 사이드카 컨테이너 생성을 단순화합니다. 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

Amazon ECS 콘솔을 사용하지 않는 경우 태스크 정의에 AWS Distro for OpenTelemetry 사이드카 컨테이너를 추가할 수 있습니다. 다음 태스크 정의 조각은 AWS X-Ray 통합을 위한 AWS Distro for OpenTelemetry 사이드카를 추가하기 위한 컨테이너 정의를 보여줍니다.

```
{
 "family": "otel-using-xray",
 "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryXrayRole",
 "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
 "containerDefinitions": [{
   "name": "aws-otel-emitter",
   "image": "application-image",
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-create-group": "true",
     "awslogs-group": "/ecs/aws-otel-emitter",
     "awslogs-region": "us-east-1",
     "awslogs-stream-prefix": "ecs"
    }
   },
   "dependsOn": [{
    "containerName": "aws-otel-collector",
    "condition": "START"
   }]
  },
  {
   "name": "aws-otel-collector",
```

```
    "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",
    "essential": true,
    "command": [
     "--config=/etc/ecs/otel-instance-metrics-config.yaml"
    ],
    "logConfiguration": {
     "logDriver": "awslogs",
     "options": {
      "awslogs-create-group": "True",
      "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "ecs"
     }
    }
   }
  ],
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
   "FARGATE"
  ],
  "cpu": "1024",
  "memory": "3072"
}
```

애플리케이션 지표를 사용하여 Amazon ECS 애플리케이션 성능 상호 연관

Fargate의 Amazon ECS는 Fargate에서 실행되는 애플리케이션에서 지표를 수집하고 이를 Amazon CloudWatch 또는 Amazon Managed Service for Prometheus로 내보내는 것을 지원합니다.

수집된 메타데이터를 사용하여 애플리케이션 성능 데이터를 기본 인프라 데이터와 상호 연관시켜 문제 해결에 걸리는 평균 시간을 줄일 수 있습니다.

Amazon ECS는 AWS Distro for OpenTelemetry사이드카 컨테이너를 사용하여 애플리케이션 지표를 수집하고 대상으로 라우팅합니다. Amazon ECS 콘솔 환경은 작업 정의를 생성할 때 이 통합을 추가하는 프로세스를 간소화합니다.

주제

- Amazon CloudWatch로 애플리케이션 지표 내보내기
- Amazon Managed Service for Prometheus로 애플리케이션 지표 내보내기

Amazon CloudWatch로 애플리케이션 지표 내보내기

Fargate의 Amazon ECS는 사용자 정의 애플리케이션 지표를 Amazon CloudWatch에 사용자 정의 지표로 내보내기를 지원합니다. 이는 태스크 정의에 AWS Distro for OpenTelemetry 사이드카 컨테이너를 추가하는 방식으로 수행됩니다. Amazon ECS 콘솔은 새 작업 정의를 생성할 때 지표 수집 사용 옵션을 추가하여 이 프로세스를 간소화합니다. 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

애플리케이션 지표는 로그 그룹 이름이 `/aws/ecs/application/metrics`인 CloudWatch Logs로 내보내지며 `ECS/AWSOTel/Application` 네임스페이스에서 볼 수 있습니다. 애플리케이션은 OpenTelemetry SDK로 계측되어야 합니다. 자세한 정보는 AWS Distro for OpenTelemetry 설명서의 AWS Distro for OpenTelemetry 소개를 참조하세요.

고려 사항

Amazon CloudWatch에 애플리케이션 지표를 전송하기 위해 Fargate의 Amazon ECS와 AWS Distro for OpenTelemetry 통합을 사용할 때 다음 사항을 고려해야 합니다.

- 이 통합은 사용자 지정 애플리케이션 지표만 CloudWatch로 전송합니다. 작업 수준 지표를 원하는 경우 Amazon ECS 클러스터 구성에서 컨테이너 인사이트를 켭니다. 자세한 내용은 관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.
- AWS Distro for OpenTelemetry 통합은 Fargate에서 호스팅되는 Amazon ECS 워크로드 및 Amazon EC2 인스턴스에서 호스팅되는 Amazon ECS 워크로드에 대해 지원됩니다. 외부 인스턴스는 현재 지원되지 않습니다.
- CloudWatch는 지표당 최대 30개의 차원을 지원합니다. Amazon ECS는 기본적으로 `TaskARN`, `ClusterARN`, `LaunchType`, `TaskDefinitionFamily` 및 `TaskDefinitionRevision` 차원을 지표에 포함합니다. 나머지 25개 차원은 애플리케이션에서 정의할 수 있습니다. 30개 이상의 차원이 구성된 경우 CloudWatch는 이를 표시할 수 없습니다. 이 경우 애플리케이션 지표는 `ECS/AWSOTel/Application` CloudWatch 지표 네임스페이스에 표시되지만 차원은 없습니다. 애플리케이션을 계측하여 차원을 추가할 수 있습니다. 자세한 정보는 AWS Distro for OpenTelemetry 설명서의 AWS Distro for OpenTelemetry로 CloudWatch 지표 사용을 참조하세요.

AWS Distro for OpenTelemetry와 Amazon CloudWatch 통합에 필요한 IAM 권한

Amazon ECS와 AWS Distro for OpenTelemetry를 통합하려면 태스크 IAM 역할을 생성하고 태스크 정의에서 역할을 지정해야 합니다. AWS Distro for OpenTelemetry 사이드카도 컨테이너 로그를 CloudWatch Logs로 라우팅하도록 구성하는 것이 좋습니다. 이 경우 태스크 실행 IAM 역할도 생성하고

태스크 정의에 지정해야 합니다. Amazon ECS 콘솔은 사용자를 대신하여 작업 실행 IAM 역할을 처리하지만 작업 IAM 역할은 수동으로 생성하고 작업 정의에 추가해야 합니다. 태스크 실행 IAM 역할에 대한 자세한 정보는 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

> **Important**
>
> AWS Distro for OpenTelemetry 통합을 사용하여 애플리케이션 추적 데이터도 수집하는 경우 태스크 IAM 역할에 해당 통합에 필요한 권한도 포함되어 있는지 확인합니다. 자세한 내용은 애플리케이션 추적 데이터를 사용하여 Amazon ECS 최적화 기회 식별 섹션을 참조하세요. 애플리케이션에 추가 권한이 필요한 경우 이 정책에 추가해야 합니다. 각 태스크 정의는 하나의 태스크 IAM 역할만 지정할 수 있습니다. 예를 들어 Systems Manager에 저장된 사용자 정의 구성 파일을 사용하는 경우 이 IAM 정책에 `ssm:GetParameters` 권한을 추가해야 합니다.

Elastic Container Service에 대한 서비스 역할을 생성하는 방법(IAM 콘솔)

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할을 선택하고 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티 유형에 AWS 서비스를 선택합니다.
4. 서비스 또는 사용 사례의 경우 Elastic Container Service를 선택하고 Elastic Container Service 작업 사용 사례를 선택합니다.
5. 다음을 선택합니다.
6. 권한 추가 섹션에서 AWSDistroOpenTelemetryPolicyForXray를 검색하고 정책을 선택합니다.
7. (선택 사항) 권한 경계를 선택합니다. 이는 서비스 역할에서 가능한 고급 기능이며 서비스 링크된 역할은 아닙니다.

   a. 권한 경계 설정 섹션을 열고 최대 역할 권한을 관리하기 위한 권한 경계 사용을 선택합니다.

      IAM은 계정의 AWS 관리형 또는 고객 관리형 정책 목록을 포함합니다.

   b. 정책을 선택하여 권한 경계를 사용하세요.
8. 다음을 선택합니다.
9. 역할의 목적을 식별하는 데 도움이 되는 역할 이름이나 역할 이름 접미사를 입력합니다.

> **Important**
>
> 역할 이름을 지정할 때는 다음 사항에 유의하세요.

- 역할 이름은 AWS 계정 내에서 고유해야 하지만 대소문자를 구분하지는 않습니다.

  예를 들어, 이름이 **PRODROLE**과 **prodrole**, 두 가지로 지정된 역할을 만들지 마세요. 역할 이름이 정책 또는 ARN의 일부로 사용되는 경우 역할 이름은 대소문자를 구분합니다. 그러나 로그인 프로세스와 같이 콘솔에서 역할 이름이 고객에게 표시되는 경우에는 역할 이름이 대소문자를 구분하지 않습니다.
- 다른 엔터티가 역할을 참조할 수 있기 때문에 역할이 생성된 후에는 역할 이름을 편집할 수 없습니다.

10. (선택 사항)설명에 역할에 대한 설명을 입력합니다.
11. (선택 사항) 역할에 대한 사용 사례와 권한을 편집하려면 1단계: 신뢰할 수 있는 엔터티 선택 또는 2단계: 권한 추가 섹션에서 편집을 선택합니다.
12. (선택 사항) 태그를 키-값 페어로 연결하여 역할을 식별, 구성 또는 검색합니다. IAM에서 태그 사용에 대한 자세한 내용은 IAM 사용 설명서의 AWS Identity and Access Management 리소스용 태그를 참조하세요.
13. 역할을 검토한 다음 역할 생성을 선택합니다.

태스크 정의에서 AWS Distro for OpenTelemetry 사이드카 지정

Amazon ECS 콘솔은 지표 수집 사용 옵션으로 AWS Distro for OpenTelemetry 사이드카 컨테이너 생성 환경을 단순화합니다. 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

Amazon ECS 콘솔을 사용하지 않는 경우 태스크 정의에 AWS Distro for OpenTelemetry 사이드카 컨테이너를 수동으로 추가할 수 있습니다. 다음 태스크 정의 예는 Amazon CloudWatch 통합을 위한 AWS Distro for OpenTelemetry 사이드카를 추가하기 위한 컨테이너 정의를 보여줍니다.

```
{
 "family": "otel-using-cloudwatch",
 "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryCloudWatchRole",
 "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
 "containerDefinitions": [
    {
   "name": "aws-otel-emitter",
   "image": "application-image",
   "logConfiguration": {
    "logDriver": "awslogs",
```

```
    "options": {
     "awslogs-create-group": "true",
     "awslogs-group": "/ecs/aws-otel-emitter",
     "awslogs-region": "us-east-1",
     "awslogs-stream-prefix": "ecs"
    }
   },
   "dependsOn": [{
    "containerName": "aws-otel-collector",
    "condition": "START"
   }]
  },
  {
   "name": "aws-otel-collector",
   "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",
   "essential": true,
   "command": [
    "--config=/etc/ecs/ecs-cloudwatch.yaml"
   ],
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-create-group": "True",
     "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
     "awslogs-region": "us-east-1",
     "awslogs-stream-prefix": "ecs"
    }
   }
  }
 ],
 "networkMode": "awsvpc",
 "requiresCompatibilities": [
  "FARGATE"
 ],
 "cpu": "1024",
 "memory": "3072"
}
```

Amazon Managed Service for Prometheus로 애플리케이션 지표 내보내기

Amazon ECS는 태스크 수준 CPU, 메모리, 네트워크 및 스토리지 지표와 사용자 정의 애플리케이션 지표를 Amazon Managed Service for Prometheus로 내보내기를 지원합니다. 이는 태스크 정의에 AWS Distro for OpenTelemetry 사이드카 컨테이너를 추가하는 방식으로 수행됩니다. Amazon ECS 콘솔은

새 작업 정의를 생성할 때 지표 수집 사용 옵션을 추가하여 이 프로세스를 간소화합니다. 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

지표는 Amazon Managed Service for Prometheus로 내보내지며 Amazon Managed Grafana 대시보드를 사용하여 볼 수 있습니다. 애플리케이션은 Prometheus 라이브러리 또는 OpenTelemetry SDK로 계측되어야 합니다. OpenTelemetry SDK로 애플리케이션 계측에 대한 자세한 정보는 AWS Distro for OpenTelemetry 설명서의 AWS Distro for OpenTelemetry 소개를 참조하세요.

Prometheus 라이브러리를 사용할 때 애플리케이션은 지표 데이터를 스크레이핑하는 데 사용되는 `/metrics` 엔드포인트를 노출해야 합니다. Prometheus 라이브러리로 애플리케이션 계측에 대한 자세한 정보는 Prometheus 설명서의 Prometheus 클라이언트 라이브러리를 참조하세요.

고려 사항

Amazon Managed Service for Prometheus에 애플리케이션 지표를 전송하기 위해 Fargate의 Amazon ECS와 AWS Distro for OpenTelemetry 통합을 사용하는 경우 다음 사항을 고려해야 합니다.

- AWS Distro for OpenTelemetry 통합은 Fargate에서 호스팅되는 Amazon ECS 워크로드 및 Amazon EC2 인스턴스에서 호스팅되는 Amazon ECS 워크로드에 대해 지원됩니다. 외부 인스턴스는 현재 지원되지 않습니다.
- 기본적으로 AWS Distro for OpenTelemetry에는 Amazon Managed Service for Prometheus로 내보낼 때 애플리케이션 지표에 사용 가능한 모든 태스크 수준 차원이 포함됩니다. 애플리케이션을 계측하여 차원을 추가할 수도 있습니다. 자세한 정보는 AWS Distro for OpenTelemetry 설명서의 Amazon Managed Service for Prometheus용 Prometheus Remote Write Exporter 시작하기(Getting Started with Prometheus Remote Write Exporter for Amazon Managed Service for Prometheus)를 참조하세요.

AWS Distro for OpenTelemetry와 Amazon Managed Service for Prometheus 통합에 필요한 IAM 권한

AWS Distro for OpenTelemetry 사이드카를 사용하여 Amazon ECS와 Amazon Managed Service for Prometheus를 통합하려면 태스크 IAM 역할을 생성하고 태스크 정의에서 역할을 지정해야 합니다. 이 태스크 IAM 역할은 태스크 정의를 등록하기 전에 아래 단계를 사용하여 수동으로 생성해야 합니다.

AWS Distro for OpenTelemetry 사이드카도 컨테이너 로그를 CloudWatch Logs로 라우팅하도록 구성하는 것이 좋습니다. 이 경우 태스크 실행 IAM 역할도 생성하고 태스크 정의에 지정해야 합니다. Amazon ECS 콘솔은 사용자를 대신하여 작업 실행 IAM 역할을 처리하지만 태스크 IAM 역할은 수동으로 생성해야 합니다. 태스크 실행 IAM 역할 생성에 대한 자세한 정보는 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

> **Important**
>
> AWS Distro for OpenTelemetry 통합을 사용하여 애플리케이션 추적 데이터도 수집하는 경우 태스크 IAM 역할에 해당 통합에 필요한 권한도 포함되어 있는지 확인합니다. 자세한 내용은 애플리케이션 추적 데이터를 사용하여 Amazon ECS 최적화 기회 식별 섹션을 참조하세요.

Elastic Container Service에 대한 서비스 역할을 생성하는 방법(IAM 콘솔)

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할을 선택하고 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티 유형에 AWS 서비스를 선택합니다.
4. 서비스 또는 사용 사례의 경우 Elastic Container Service를 선택하고 Elastic Container Service 작업 사용 사례를 선택합니다.
5. 다음을 선택합니다.
6. 권한 추가 섹션에서 AmazonPrometheusRemoteWriteAccess를 검색하고 정책을 선택합니다.
7. (선택 사항) 권한 경계를 선택합니다. 이는 서비스 역할에서 가능한 고급 기능이며 서비스 링크된 역할은 아닙니다.

   a. 권한 경계 설정 섹션을 열고 최대 역할 권한을 관리하기 위한 권한 경계 사용을 선택합니다.

      IAM은 계정의 AWS 관리형 또는 고객 관리형 정책 목록을 포함합니다.

   b. 정책을 선택하여 권한 경계를 사용하세요.
8. 다음을 선택합니다.
9. 역할의 목적을 식별하는 데 도움이 되는 역할 이름이나 역할 이름 접미사를 입력합니다.

> **Important**
>
> 역할 이름을 지정할 때는 다음 사항에 유의하세요.
>
> - 역할 이름은 AWS 계정 내에서 고유해야 하지만 대소문자를 구분하지는 않습니다.
>
>   예를 들어, 이름이 **PRODROLE**과 **prodrole**, 두 가지로 지정된 역할을 만들지 마세요. 역할 이름이 정책 또는 ARN의 일부로 사용되는 경우 역할 이름은 대소문자를 구분합니다. 그러나 로그인 프로세스와 같이 콘솔에서 역할 이름이 고객에게 표시되는 경우에는 역할 이름이 대소문자를 구분하지 않습니다.

- 다른 엔터티가 역할을 참조할 수 있기 때문에 역할이 생성된 후에는 역할 이름을 편집할 수 없습니다.

10. (선택 사항)설명에 역할에 대한 설명을 입력합니다.
11. (선택 사항) 역할에 대한 사용 사례와 권한을 편집하려면 1단계: 신뢰할 수 있는 엔터티 선택 또는 2단계: 권한 추가 섹션에서 편집을 선택합니다.
12. (선택 사항) 태그를 키-값 페어로 연결하여 역할을 식별, 구성 또는 검색합니다. IAM에서 태그 사용에 대한 자세한 내용은 IAM 사용 설명서의 AWS Identity and Access Management 리소스용 태그를 참조하세요.
13. 역할을 검토한 다음 역할 생성을 선택합니다.

태스크 정의에서 AWS Distro for OpenTelemetry 사이드카 지정

Amazon ECS 콘솔은 지표 수집 사용 옵션으로 AWS Distro for OpenTelemetry 사이드카 컨테이너 생성 환경을 단순화합니다. 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.

Amazon ECS 콘솔을 사용하지 않는 경우 태스크 정의에 AWS Distro for OpenTelemetry 사이드카 컨테이너를 수동으로 추가할 수 있습니다. 다음 태스크 정의 예는 Amazon Managed Service for Prometheus 통합을 위한 AWS Distro for OpenTelemetry 사이드카를 추가하기 위한 컨테이너 정의를 보여줍니다.

```
{
 "family": "otel-using-cloudwatch",
 "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryCloudWatchRole",
 "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
 "containerDefinitions": [{
   "name": "aws-otel-emitter",
   "image": "application-image",
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-create-group": "true",
     "awslogs-group": "/ecs/aws-otel-emitter",
     "awslogs-region": "aws-region",
     "awslogs-stream-prefix": "ecs"
    }
   },
   "dependsOn": [{
```

```
    "containerName": "aws-otel-collector",
    "condition": "START"
   }]
  },
  {
   "name": "aws-otel-collector",
   "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",
   "essential": true,
   "command": [
    "--config=/etc/ecs/ecs-amp.yaml"
   ],
   "environment": [{
    "name": "AWS_PROMETHEUS_ENDPOINT",
    "value": "https://aps-workspaces.aws-region.amazonaws.com/workspaces/
ws-a1b2c3d4-5678-90ab-cdef-EXAMPLE11111/api/v1/remote_write"
   }],
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-create-group": "True",
     "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
     "awslogs-region": "aws-region",
     "awslogs-stream-prefix": "ecs"
    }
   }
  }
 ],
 "networkMode": "awsvpc",
 "requiresCompatibilities": [
  "FARGATE"
 ],
 "cpu": "1024",
 "memory": "3072"
}
```

AWS CloudTrail을 사용하여 Amazon ECS API 직접 호출 로깅

Amazon Neptune은 Neptune에서 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. CloudTrail은 Amazon ECS에 대한 모든 API 직접 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 Amazon ECS 콘솔로부터의 호출과 Amazon ECS API 작업에 대한 코드 호출이 포함됩니다. CloudTrail에서 수집한 정보를 사용하여 Amazon ECS에 전송된 요청, 요청이 이루어진 IP 주소, 요청이 이루어진 시간, 추가 세부 정보를 확인할 수 있습니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. 자격 증명을 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트 사용자로 했는지 사용자 보안 인증으로 했는지 여부.
- IAM Identity Center 사용자를 대신하여 요청이 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자에 대한 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

계정을 생성할 때 AWS 계정에서 CloudTrail이 활성 상태이며, CloudTrail 이벤트 기록에 자동으로 액세스할 수 있습니다. CloudTrail 이벤트 기록은 지난 90일 간 AWS 리전의 관리 이벤트에 대해 보기, 검색 및 다운로드가 가능하고, 수정이 불가능한 레코드를 제공합니다. 자세한 설명은 AWS CloudTrail 사용 설명서의 CloudTrail 이벤트 기록 작업을 참조하세요. Event history(이벤트 기록) 보기는 CloudTrail 요금이 부과되지 않습니다.

지난 90일 동안 AWS 계정에서 진행 중인 이벤트 기록을 보려면 추적 또는 CloudTrail Lake 이벤트 데이터 스토어를 생성합니다.

CloudTrail 추적

CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. AWS Management Console을 사용하여 만든 추적은 모두 다중 리전입니다. AWS CLI를 사용하여 단일 리전 또는 다중 리전 추적을 생성할 수 있습니다. 계정의 모든 AWS 리전에서 활동을 캡처하므로, 다중 리전 추적 생성이 권장됩니다. 단일 리전 추적을 생성하는 경우 추적의 AWS 리전에 로깅된 이벤트만 볼 수 있습니다. 추적에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 Creating a trail for your AWS 계정 및 Creating a trail for an organization을 참조하세요.

CloudTrail에서 추적을 생성하여 진행 중인 관리 이벤트의 사본 하나를 Amazon S3 버킷으로 무료로 전송할 수는 있지만, Amazon S3 스토리지 요금이 부과됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요. Amazon S3 요금에 대한 자세한 내용은 Amazon S3 요금을 참조하세요.

CloudTrail Lake 이벤트 데이터 스토어

CloudTrail Lake를 사용하면 이벤트에 대해 SQL 기반 쿼리를 실행할 수 있습니다. CloudTrail Lake는 행 기반 JSON 형식의 기존 이벤트를 Apache ORC 형식으로 변환합니다. ORC는 빠른 데이터 검색에 최적화된 열 기반 스토리지 형식입니다. 이벤트는 이벤트 데이터 스토어로 집계되며, 이벤트 데이터 스토어는 고급 이벤트 선택기를 적용하여 선택한 기준을 기반으로 하는 변경 불가능한 이벤트 컬렉션입니다. 이벤트 데이터 스토어에 적용하는 선택기는 어떤 이벤트가 지속되고 쿼리

할 수 있는지 제어합니다. CloudTrail Lake에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 Working with AWS CloudTrail Lake를 참조하세요.

CloudTrail Lake 이벤트 데이터 스토어 및 쿼리에는 비용이 발생합니다. 이벤트 데이터 스토어를 생성할 때 이벤트 데이터 스토어에 사용할 요금 옵션을 선택합니다. 요금 옵션에 따라 이벤트 모으기 및 저장 비용과 이벤트 데이터 스토어의 기본 및 최대 보존 기간이 결정됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요.

CloudTrail의 Amazon ECS 관리 이벤트

관리 이벤트는 AWS 계정의 리소스에 대해 수행되는 관리 작업에 대한 정보를 제공합니다. 이를 컨트롤 플레인 작업이라고도 합니다. 기본적으로 CloudTrail은 관리 이벤트를 로깅합니다.

Amazon Elastic Container Service는 모든 Amazon ECS 컨트롤 플레인 작업을 관리 이벤트로 로깅합니다. 예를 들어, `CreateService`, `RunTask` 및 `DeleteCluster`을 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다. Amazon ECS가 CloudTrail에 기록하는 Amazon Elastic Container Service 컨트롤 플레인 작업의 목록은 Amazon Elastic Container Service API 참조를 참조하십시오.

Amazon ECS 이벤트 예제

이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청된 API 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 추적이 아니므로 이벤트가 특정 순서로 표시되지 않습니다.

다음은 `CreateService` 작업을 설명하는 CloudTrail 이벤트를 보여 주는 예제입니다.

> **Note**
>
> 이 예제는 가독성을 높이도록 형식이 지정되었습니다. CloudTrail 로그 파일에서는 모든 항목 및 이벤트가 한 줄로 연결되어 있습니다. 또한 이 예제는 단일 Amazon ECS 항목으로 제한된 것입니다. 실제 CloudTrail 로그 파일에는 여러 AWS 서비스의 항목 및 이벤트가 기록됩니다.


```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
```

```
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-06-20T18:32:25Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Mary_Major"
            }
        }
    },
    "eventTime": "2018-06-20T19:04:36Z",
    "eventSource": "ecs.amazonaws.com",
    "eventName": "CreateCluster",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "clusterName": "default"
    },
    "responseElements": {
        "cluster": {
            "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/default",
            "pendingTasksCount": 0,
            "registeredContainerInstancesCount": 0,
            "status": "ACTIVE",
            "runningTasksCount": 0,
            "statistics": [],
            "clusterName": "default",
            "activeServicesCount": 0
        }
    },
    "requestID": "cb8c167e-EXAMPLE",
    "eventID": "e3c6f4ce-EXAMPLE",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

CloudTrail 레코드 콘텐츠에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail record contents를 참조하세요.

Amazon ECS 서비스 및 태스크에 대한 CloudWatch Logs Live Tail 보기

CloudWatch Logs Live Tail을 사용하면 모이는 새 로그 이벤트의 스트리밍 목록을 보고 인시던트를 신속하게 해결할 수 있습니다. Amazon ECS 콘솔에서 태스크 및 서비스 이벤트를 볼 수 있습니다. 이 보기는 태스크 및 서비스 상태에 대한 일관된 보기를 제공합니다.

Amazon ECS 서비스의 각 태스크에는 각 컨테이너에 대한 전용 로그 스트림이 있습니다. 서비스가 스케일 업되면 각 태스크 인스턴스는 고유한 로그 스트림 세트를 보유합니다. 로그 스트림의 이름 지정 규칙은 다음 패턴을 따릅니다.

```
log-group-prefix/container-name/task-id
```

단일 태스크는 해당 수명 동안 같은 로그 스트림에 씁니다. 로그 스트림에는 해당 태스크 컨테이너의 메시지와 애플리케이션 코드의 출력이 포함됩니다. 사용자 지정 로그를 포함하여 모든 메시지에는 타임스탬프가 지정됩니다.

> **Note**
>
> Live Tail 세션은 세션 사용 시간(분)별로 비용이 발생합니다. 요금에 대한 자세한 정보는 Amazon CloudWatch 요금을 참조하세요.

필수 권한

CloudWatch Logs Live Tail 세션을 시작하고 중지하려면 콘솔 IAM 사용자에 대해 다음 권한이 필요합니다.

- `logs:DescribeLogGroups`
- `logs:StartLiveTail`
- `logs:StopLiveTail`

절차

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 보려는 리소스 결정

리소스	단계
업무	a. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 표시됩니다. b. 작업 탭을 선택합니다. 태스크 세부 정보 페이지가 표시됩니다. c. 로그 탭을 선택합니다.
서비스	a. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다. 클러스터 세부 정보 페이지가 표시됩니다. b. 서비스를 선택하세요. 서비스 세부 정보 페이지가 표시됩니다. c. 로그 탭을 선택합니다.

3. CloudWatch Logs Live Tail을 선택한 다음 시작을 선택하세요.
4. (선택 사항) 스트림을 필터링하려면 필터 아래 로그 그룹 선택에서 로그 그룹을 선택하세요.

AWS CLI 참조

AWS CLI를 사용하여 CloudWatch Logs에 대한 Live Tail 세션을 시작할 수도 있습니다. 다음 리소스는 Live Tail에서 AWS CLI를 사용하는 방법에 대한 자세한 정보를 제공합니다.

- start-live-tail 명령 참조 - `aws logs start-live-tail` 명령에 대한 전체 명령 구문, 파라미터 및 예제.
- CloudWatch Logs Live Tail 사용 설명서 - 인쇄 전용 모드와 대화형 모드 모두에서 AWS CLI 사용을 포함한 포괄적인 안내서.
- StartLiveTail SDK 예제 - 다양한 AWS SDK와 함께 StartLiveTail API를 사용하기 위한 프로그래밍 방식 예제.

Amazon ECS 메타데이터를 사용한 워크로드 모니터링

태스크 및 컨테이너 메타데이터를 사용하여 워크로드 문제를 해결하고 런타임 환경에 따라 구성을 변경할 수 있습니다.

메타데이터에는 다음 범주가 포함됩니다.

- 작업이 실행되는 위치에 대한 정보를 제공하는 작업 수준 속성.
- Docker ID, 이름, 이미지 세부 정보를 제공하는 컨테이너 수준 속성.

  이를 통해 컨테이너에 대한 가시성을 제공합니다.
- IP 주소, 서브넷, 네트워크 모드와 같은 네트워크 설정.

  이를 통해 네트워크 구성 및 문제 해결에 도움을 줍니다.
- 작업 상태 및 정상 상태

  이를 통해 태스크가 실행 중인지 알 수 있습니다.

다음 방법 중 하나를 사용하여 메타데이터를 볼 수 있습니다.

- 컨테이너 메타데이터 파일

  1.15.0 버전의 Amazon ECS 컨테이너 에이전트를 비롯해 컨테이너 또는 호스트 컨테이너 인스턴스 내에서 다양한 컨테이너 메타데이터를 얻을 수 있습니다. 이 기능을 활성화하여 컨테이너 또는 호스트 컨테이너 인스턴스 내에서 태스크, 컨테이너 및 컨테이너 인스턴스에 대한 정보를 쿼리할 수 있

습니다. 메타데이터 파일은 호스트 인스턴스에서 생성되고 Docker 볼륨으로 탑재되므로 태스크가 AWS Fargate에서 호스팅될 때는 사용할 수 없습니다.

- 태스크 메타데이터 엔드포인트

  Amazon ECS 컨테이너 에이전트가 태스크 메타데이터 엔드포인트라고 하는 각 컨테이너에 환경 변수를 주입합니다. 이 컨테이너는 다양한 태스크 메타데이터와 Docker 통계를 컨테이너에 제공합니다.

- 컨테이너 내부 검사

  Amazon ECS 컨테이너 에이전트는 에이전트가 실행 중인 컨테이너 인스턴스와 이 인스턴스에서 실행 중인 관련 태스크에 대한 세부 정보를 수집하기 위한 API 태스크를 제공합니다.

Amazon ECS 환경 변수

환경 변수는 태스크를 실행하는 데 사용되는 용량 등의 태스크의 동작을 제어합니다. Amazon ECS는 태스크에 대해 다음과 같은 환경 변수를 설정합니다.

- `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` - 자격 증명을 요청할 때 SDK가 사용할 전체 HTTP URL 엔드포인트입니다. 여기에는 스키마와 호스트가 모두 포함됩니다. 자세한 내용은 AWS SDK 참조 가이드의 컨테이너 자격 증명 공급자를 참조하세요.
- `ECS_CONTAINER_METADATA_URI_V4` - 작업 메타데이터 버전 4의 주소입니다. 자세한 내용은 Amazon ECS 작업 메타데이터 엔드포인트 버전 4 섹션을 참조하세요.
- `ECS_CONTAINER_METADATA_URI` - 작업 메타데이터 버전 3의 주소입니다. 자세한 내용은 Amazon ECS 작업 메타데이터 엔드포인트 버전 3 섹션을 참조하세요.
- `ECS_AGENT_URI` - Fargate에서 지원하는 다양한 엔드포인트의 기본 주소입니다. 자세한 내용은 다음을 참조하세요.
  - Amazon ECS 작업 축소 보호 엔드포인트
  - Amazon ECS 결함 주입 엔드포인트
- `AWS_EXECUTION_ENV` - 작업이 실행되는 시작 유형에 대한 정보입니다.
  - Fargate의 경우 Amazon ECS는 이를 `AWS_ECS_FARGATE`로 설정합니다.
  - EC2의 경우 Amazon ECS는 이를 `AWS_ECS_EC2`로 설정합니다.
- `AWS_DEFAULT_REGION` - AWS 계정의 기본 AWS 리전입니다. 태스크가 실행되는 기본 리전입니다.
- `AWS_REGION` – 태스크가 실행되는 리전입니다. 정의되면 이 값이 `AWS_DEFAULT_REGION`을 재정의합니다.

태스크 메타데이터에서 환경 변수를 볼 수 있습니다. 자세한 내용은 다음 중 하나의 주제를 참조하세요:

- Fargate - Fargate의 작업에 대해 사용 가능한 Amazon ECS 작업 메타데이터
- EC2 - EC2의 Amazon ECS 작업에 대해 사용 가능한 작업 메타데이터

Amazon ECS 컨테이너 메타데이터 파일

1.15.0 버전의 Amazon ECS 컨테이너 에이전트를 비롯해 컨테이너 또는 호스트 컨테이너 인스턴스 내에서 다양한 컨테이너 메타데이터를 얻을 수 있습니다. 이 기능을 활성화하여 컨테이너 또는 호스트 컨테이너 인스턴스 내에서 태스크, 컨테이너 및 컨테이너 인스턴스에 대한 정보를 쿼리할 수 있습니다. 메타데이터 파일은 호스트 인스턴스에서 생성되고 Docker 볼륨으로 탑재되므로 태스크가 AWS Fargate에서 호스팅될 때는 사용할 수 없습니다.

컨테이너 메타데이터 파일은 컨테이너가 정리될 때 호스트 인스턴스에서 정리됩니다. 이런 일이 발생하면 `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` 컨테이너 에이전트 변수를 사용하여 조정할 수 있습니다. 자세한 내용은 자동 Amazon ECS 태스크 및 이미지 정리 섹션을 참조하세요.

주제

- 컨테이너 메타데이터 파일 위치
- Amazon ECS 컨테이너 메타데이터 켜기
- Amazon ECS 컨테이너 메타데이터 파일 형식

컨테이너 메타데이터 파일 위치

기본적으로 컨테이너 메타데이터 파일은 다음 호스트 및 컨테이너 경로에 작성됩니다.

- Linux 인스턴스의 경우:
  - 호스트 경로: `/var/lib/ecs/data/metadata/`*`cluster_name`*`/`*`task_id`*`/`*`container_name`*`/ecs-container-metadata.json`

> **Note**
>
> Linux 호스트 경로에서는 에이전트가 시작될 때 기본 데이터 디렉터리 마운트 경로(`/var/lib/ecs/data`)를 사용하는 것으로 가정합니다. Amazon ECS 최적화 AMI(또는 컨테이너 에이전트를 시작하고 유지하는 `ecs-init` 패키지)를 사용하지 않는 경우, 컨테이너 에

이전트의 상태 파일이 있는 호스트 경로에 ECS_HOST_DATA_DIR 에이전트 구성 변수를 설정해야 합니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성을 참조하세요.

  - 컨테이너 경로: /opt/ecs/metadata/*random_ID*/ecs-container-metadata.json
- Windows 인스턴스의 경우:
  - 호스트 경로: C:\ProgramData\Amazon\ECS\data\metadata*task_id**container_name*\ecs-container-metadata.json
  - 컨테이너 경로: C:\ProgramData\Amazon\ECS\metadata*random_ID*\ecs-container-metadata.json

그러나 손쉬운 액세스를 위해 컨테이너 메타데이터 파일 위치는 컨테이너 내부에 ECS_CONTAINER_METADATA_FILE 환경 변수로 설정됩니다. 다음 명령을 사용하여 컨테이너 내부에서 파일 콘텐츠를 읽을 수 있습니다.

- Linux 인스턴스의 경우:

```
cat $ECS_CONTAINER_METADATA_FILE
```

- Windows 인스턴스의 경우(PowerShell):

```
Get-Content -path $env:ECS_CONTAINER_METADATA_FILE
```

Amazon ECS 컨테이너 메타데이터 켜기

ECS_ENABLE_CONTAINER_METADATA 컨테이너 에이전트 변수를 true로 설정하여 컨테이너 인스턴스 수준에서 컨테이너 메타데이터를 켤 수 있습니다. /etc/ecs/ecs.config 구성 파일에서 이 변수를 설정하고 에이전트를 다시 시작할 수 있습니다. 또는 이 변수를 에이전트 컨테이너가 시작되는 실행 시간의 Docker 환경 변수로 설정할 수도 있습니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성을 참조하세요.

에이전트 시작 시 ECS_ENABLE_CONTAINER_METADATA가 true로 설정되면, 그 시점 이후에 생성된 모든 컨테이너에 대해 메타데이터 파일이 생성됩니다. Amazon ECS 컨테이너 에이전트는 ECS_ENABLE_CONTAINER_METADATA 컨테이너 에이전트 변수가 true로 설정되기 전에 생성된 컨테이너에 대해서는 메타데이터 파일을 생성할 수 없습니다. 모든 컨테이너가 메타데이터 파일을 받을 수 있도록 하려면 컨테이너 인스턴스 시작 시 이 에이전트 변수를 설정해야 합니다. 다음은 이 변수를 설정하고 컨테이너 인스턴스를 클러스터에 등록하는 사용자 데이터 스크립트 예제입니다.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_ENABLE_CONTAINER_METADATA=true
EOF
```

Amazon ECS 컨테이너 메타데이터 파일 형식

다음 정보는 컨테이너 메타데이터 JSON 파일에 저장됩니다.

`Cluster`

컨테이너의 태스크가 실행되는 클러스터의 이름입니다.

`ContainerInstanceARN`

호스트 컨테이너 인스턴스의 전체 Amazon 리소스 이름(ARN)

`TaskARN`

컨테이너가 속한 태스크의 전체 Amazon 리소스 이름(ARN)

`TaskDefinitionFamily`

컨테이너가 사용 중인 태스크 정의 패밀리의 이름입니다.

`TaskDefinitionRevision`

컨테이너가 사용 중인 태스크 정의 수정 버전입니다.

`ContainerID`

컨테이너에 대한 Docker 컨테이너 ID(Amazon ECS 컨테이너 ID는 아님)

`ContainerName`

컨테이너에 대한 Amazon ECS 태스크 정의의 컨테이너 이름

`DockerContainerName`

Docker 대몬이 컨테이너에 대해 사용하는 컨테이너 이름(예: docker ps 명령 출력에 표시되는 이름)

`ImageID`

컨테이너를 시작하는 데 사용되는 Docker 이미지에 대한 SHA 다이제스트

`ImageName`

컨테이너를 시작하는 데 사용되는 Docker 이미지에 대한 이미지 이름 및 태그

`PortMappings`

컨테이너에 연결된 모든 포트 매핑

`ContainerPort`

노출된 컨테이너 상의 포트

`HostPort`

노출된 호스트 컨테이너 인스턴스 상의 포트

`BindIp`

Docker가 컨테이너에 할당하는 바인드 IP 주소 이 IP 주소는 `bridge` 네트워크 모드에만 적용되고 컨테이너 인스턴스에서만 접근할 수 있습니다.

`Protocol`

포트 매핑에 사용되는 네트워크 프로토콜

`Networks`

컨테이너에 대한 네트워크 모드 및 IP 주소

`NetworkMode`

컨테이너가 속하는 태스크에 대한 네트워크 모드

`IPv4Addresses`

컨테이너에 연결된 IP 주소입니다.

> **Important**
>
> 태스크에서 awsvpc 네트워크 모드를 사용 중인 경우 컨테이너의 IP 주소가 반환되지 않습니다. 이 경우 다음 명령으로 /etc/hosts 파일을 읽어 IP 주소를 검색할 수 있습니다.
>
> ```
> tail -1 /etc/hosts | awk '{print $1}'
> ```

`MetadataFileStatus`

메타데이터 파일의 상태. 상태가 `READY`이면 메타데이터 파일이 완전한 최신 파일이라는 것입니다. 파일이 아직 준비되지 않은 경우(예: 작업이 시작되는 순간) 축약형 버전의 파일 형식을 사용할 수 있습니다. 컨테이너가 시작되었지만 메타데이터가 아직 작성되지 않은 경우 있을 수 있는 교착 상

태를 방지하기 위해 메타데이터 파일을 구문 분석하고 메타데이터에 의존하기 전에 이 파라미터가 READY로 설정될 때까지 기다릴 수 있습니다. 이는 대개 컨테이너가 시작되는 시점으로부터 1초 내에 가능합니다.

`AvailabilityZone`

호스트 컨테이너 인스턴스가 상주하는 가용 영역입니다.

`HostPrivateIPv4Address`

컨테이너가 속한 태스크의 프라이빗 IP 주소입니다.

`HostPublicIPv4Address`

컨테이너가 속한 태스크의 퍼블릭 IP 주소입니다.

Example Amazon ECS 컨테이너 메타데이터 파일(**READY**)

다음 예에서는 READY 상태의 컨테이너 메타데이터 파일을 보여줍니다.

```
{
    "Cluster":"arn:aws:ecs:us-east-1:123456789012:cluster/MyCluster",
    "TaskARN":"arn:aws:ecs:us-east-1:123456789012:task/MyCluster/
b593651c4d6b44a6b2b583f45c957e15",
    "Family":"curltest-container",
    "Revision":"2",
    "DesiredStatus":"RUNNING",
    "KnownStatus":"RUNNING",
    "Limits":
        {
            "CPU":0.25,
            "Memory":512
        },
    "PullStartedAt":"2025-01-17T20:56:17.394610044Z",
    "PullStoppedAt":"2025-01-17T20:56:25.282708213Z",
    "AvailabilityZone":"us-east-1b",
    "LaunchType":"FARGATE",
    "Containers":[
        {
            "DockerId":"b593651c4d6b44a6b2b583f45c957e15-3356213583",
            "Name":"curltest","DockerName":"curltest",
            "Image":"public.ecr.aws/amazonlinux/amazonlinux:latest",

 "ImageID":"sha256:7f371357694782356b65c7fd60dd1ca124c47bd5ed1b1ffe7c0e17f562898367",
            "Labels":
```

```
                {
                    "com.amazonaws.ecs.cluster":"arn:aws:ecs:us-
east-1:123456789012:cluster/MyCluster",
                    "com.amazonaws.ecs.container-name":"curltest",
                    "com.amazonaws.ecs.task-arn":"arn:aws:ecs:us-
east-1:123456789012:task/MyCluster/b593651c4d6b44a6b2b583f45c957e15",
                    "com.amazonaws.ecs.task-definition-family":"curltest-
container","com.amazonaws.ecs.task-definition-version":"2"
                },
            "DesiredStatus":"RUNNING",
            "KnownStatus":"RUNNING",
            "Limits":
                {
                    "CPU":2
                },
            "CreatedAt":"2025-01-17T20:56:26.180347056Z",
            "StartedAt":"2025-01-17T20:56:26.180347056Z",
            "Type":"NORMAL",
            "LogDriver":"awslogs",
            "LogOptions":
                {
                    "awslogs-create-group":"true",
                    "awslogs-group":"/ecs/curltest-container",
                    "awslogs-region":"us-east-1",
                    "awslogs-stream":"ecs/curltest/b593651c4d6b44a6b2b583f45c957e15"
                    },
            "ContainerARN":"arn:aws:ecs:us-east-1:123456789012:container/MyCluster/
b593651c4d6b44a6b2b583f45c957e15/934575e8-5bdb-478f-b763-2341a85b690e",
            "Networks":[
                {
                    "NetworkMode":"awsvpc",
                    "IPv4Addresses":["10.0.1.58"]
                }
            ],
            "Snapshotter":"overlayfs"
        }
    ],
    "ClockDrift":
        {
            "ClockErrorBound":0.487801,"ReferenceTimestamp":"2025-01-17T20:56:02Z",
            "ClockSynchronizationStatus":"SYNCHRONIZED"
        },
    "FaultInjectionEnabled":false
```

```
}
```

Example 완료되지 않은 Amazon ECS 컨테이너 메타데이터 파일(아직 **READY** 상태가 아님)

다음 예에서는 READY 상태에 아직 이르지 못한 컨테이너 메타데이터 파일을 보여줍니다. 파일에 있는 정보는 태스크 정의에서 알 수 있는 파라미터 몇 개로 제한됩니다. 컨테이너 메타데이터 파일은 컨테이너가 시작된 후 1초 내에 준비되어야 합니다.

```
{
    "Cluster": "default",
    "ContainerInstanceARN": "arn:aws:ecs:us-west-2:012345678910:container-instance/
default/1f73d099-b914-411c-a9ff-81633b7741dd",
    "TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/default/
d90675f8-1a98-444b-805b-3d9cabb6fcd4",
    "ContainerName": "metadata"
}
```

EC2의 Amazon ECS 작업에 대해 사용 가능한 작업 메타데이터

Amazon ECS 컨테이너 에이전트는 다양한 태스크 메타데이터와 Docker 통계를 조회하는 메서드를 제공합니다. 이를 일컬어 태스크 메타데이터 엔드포인트라고 합니다. 다음과 같은 버전을 사용할 수 있습니다.

- 태스크 메타데이터 엔드포인트 버전 4 – 다양한 메타데이터와 Docker 통계를 컨테이너에 제공합니다. 네트워크 속도 데이터도 제공할 수 있습니다. Amazon ECS 컨테이너 에이전트 버전 `1.39.0` 이상에서 실행되는 Amazon EC2 Linux 인스턴스에서 시작된 Amazon ECS 태스크에 제공됩니다. `awsvpc` 네트워크 모드를 사용하는 Amazon EC2 Windows 인스턴스의 경우, Amazon ECS 컨테이너 에이전트 버전이 `1.54.0` 이상이어야 합니다. 자세한 정보는 Amazon ECS 작업 메타데이터 엔드포인트 버전 4을 참조하세요.
- 태스크 메타데이터 엔드포인트 버전 3 – 다양한 메타데이터 및 Docker 통계를 컨테이너에 제공합니다. Amazon ECS 컨테이너 에이전트 버전 `1.21.0` 이상에서 실행되는 Amazon EC2 Linux 인스턴스에서 시작된 Amazon ECS 태스크에 제공됩니다. `awsvpc` 네트워크 모드를 사용하는 Amazon EC2 Windows 인스턴스의 경우, Amazon ECS 컨테이너 에이전트 버전이 `1.54.0` 이상이어야 합니다. 자세한 정보는 Amazon ECS 작업 메타데이터 엔드포인트 버전 3을 참조하세요.
- 태스크 메타데이터 엔드포인트 버전 2 - Amazon ECS 컨테이너 에이전트 버전 `1.17.0` 이상에서 실행되는 Amazon EC2 Linux 인스턴스에서 시작된 Amazon ECS 태스크에 제공됩니다. `awsvpc` 네트워크 모드를 사용하는 Amazon EC2 Windows 인스턴스의 경우, Amazon ECS 컨테이너 에이전트 버전이 `1.54.0` 이상이어야 합니다. 자세한 내용은 Amazon ECS 작업 메타데이터 엔드포인트 버전 2 섹션을 참조하세요.

Amazon EC2에서 Amazon ECS 작업을 호스팅하는 경우, 인스턴스 메타데이터 서비스(IMDS) 엔드포인트를 사용하여 작업 호스트 메타데이터에 액세스할 수도 있습니다. 다음 명령을 작업을 호스팅하는 인스턴스 내에서 실행하면 호스트 인스턴스의 ID가 나열됩니다.

```
curl http://169.254.169.254/latest/meta-data/instance-id
```

엔드포인트에서 얻을 수 있는 정보는 *instance-id*와 같은 범주로 구분됩니다. 엔드포인트를 사용하여 얻을 수 있는 호스트 인스턴스 메타데이터의 다양한 범주에 대한 자세한 내용은 인스턴스 메타데이터 범주를 참조하세요.

Amazon ECS 작업 메타데이터 엔드포인트 버전 4

Amazon ECS 컨테이너 에이전트가 태스크 메타데이터 엔드포인트라고 하는 각 컨테이너에 환경 변수를 주입합니다. 이 컨테이너는 다양한 태스크 메타데이터와 Docker 통계를 컨테이너에 제공합니다.

태스크 메타데이터와 네트워크 속도 통계는 CloudWatch Container Insights로 전송되고 AWS Management Console에서 확인할 수 있습니다. 자세한 정보는 관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링을 참조하세요.

> **Note**
>
> Amazon ECS는 이전 버전의 태스크 메타데이터 엔드포인트를 제공합니다. 나중에 새 태스크 메타데이터 엔드포인트 버전을 생성할 필요가 없도록 버전 4 출력에 메타데이터를 추가할 수 있습니다. 기존 메타데이터를 제거하거나 메타데이터 필드 이름을 변경하지 않습니다.

환경 변수는 기본적으로 Amazon ECS 컨테이너 에이전트 버전 `1.39.0` 이상을 실행하는 Amazon EC2 Linux 인스턴스에서 시작된 Amazon ECS 태스크의 컨테이너에 주입됩니다. `awsvpc` 네트워크 모드를 사용하는 Amazon EC2 Windows 인스턴스의 경우, Amazon ECS 컨테이너 에이전트 버전이 `1.54.0` 이상이어야 합니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 관리 섹션을 참조하세요.

> **Note**
>
> 에이전트를 최신 버전으로 업데이트하여 이전 버전의 Amazon ECS 컨테이너 인스턴스를 사용하여 Amazon EC2 인스턴스에서 이 기능에 대한 지원을 추가할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요.

태스크 메타데이터 예제 출력은 Amazon ECS 작업 메타데이터 v4 예제를 참조하세요.

태스크 메타데이터 엔드포인트 버전 4 경로

다음의 태스크 메타데이터 엔드포인트 경로를 컨테이너에 사용할 수 있습니다.

`${ECS_CONTAINER_METADATA_URI_V4}`

이 경로는 컨테이너에 대한 메타데이터를 반환합니다.

`${ECS_CONTAINER_METADATA_URI_V4}/task`

이 경로는 태스크와 연결된 모든 컨테이너의 컨테이너 ID 및 이름 목록을 포함하여 작업에 대한 메타데이터를 반환합니다. 이 엔드포인트의 응답에 대한 자세한 내용은 Amazon ECS 작업 메타데이터 V4 JSON 응답 섹션을 참조하세요.

`${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags`

이 경로는 `ListTagsForResource` API를 사용하여 검색할 수 있는 태스크와 컨테이너 인스턴스 태그 외에도 `/task` 엔드포인트에 포함된 태스크에 대한 메타데이터를 반환합니다. 태그 메타데이터를 검색하면서 수신한 오류는 응답의 `Errors` 필드에 포함됩니다.

> **Note**
>
> `Errors` 필드는 컨테이너 에이전트 버전`1.50.0` 이상을 실행하는 Amazon EC2 Linux 인스턴스에서 호스팅되는 태스크에 대한 응답에만 포함됩니다. `awsvpc` 네트워크 모드를 사용하는 Amazon EC2 Windows 인스턴스의 경우, Amazon ECS 컨테이너 에이전트 버전이 `1.54.0` 이상이어야 합니다.
> 이 엔드포인트에는 `ecs.ListTagsForResource` 권한이 필요합니다.

> **Important**
>
> `${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags` 엔드포인트를 사용하는 경우 직접 호출할 때마다 `ecs:ListTagsForResource`에 대한 최대 2번의 API 요청(컨테이너 인스턴스 태그 및 태스크 태그에 한 번씩)을 수행하고 태스크의 사이드카 컨테이너는 사용자를 대신해 이 직접 호출을 수행합니다. 엔드포인트 직접 호출이 잦으면 API 스로틀링이 발생할 수 있습니다.
> 특히 트래픽이 높은 애플리케이션에서 직접 호출 빈도를 줄이기 위해 캐싱이나 배치 처리 전략을 구현하고 AWS CloudTrail을 사용해 API 스로틀링을 디버깅하는 방법을 고려하세요. `ListTagsForResource` API의 스로틀링 제한에 대한 자세한 내용은 Amazon Elastic

Container Service API 참조의 Request throttling for the Amazon ECS API를 참조하세요. AWS CloudTrail을 사용해 Amazon ECS API 직접 호출을 디버깅하는 방법에 대한 자세한 내용은 AWS CloudTrail을 사용하여 Amazon ECS API 직접 호출 로깅 섹션을 참조하세요.

`${ECS_CONTAINER_METADATA_URI_V4}/stats`

이 경로는 특정 Docker 컨테이너에 대한 Docker 통계를 반환합니다. 반환된 각 통계에 대한 자세한 내용은 Docker API 설명서의 ContainerStats를 참조하세요.

컨테이너 에이전트 버전 `1.43.0` 이상을 실행하는 Amazon EC2 Linux 인스턴스에서 호스팅되고 `awsvpc` 또는 `bridge` 네트워크 모드를 사용하는 Amazon ECS 태스크의 경우, 응답에 추가적 네트워크 속도 통계가 포함됩니다. 그 외에 다른 모든 태스크는 응답에 누적 네트워크 통계만 포함됩니다.

`${ECS_CONTAINER_METADATA_URI_V4}/task/stats`

이 경로는 태스크와 연결된 모든 컨테이너에 대한 Docker 통계를 반환합니다. 사이드카 컨테이너에서 네트워크 지표를 추출하는 데 사용할 수 있습니다. 반환된 각 통계에 대한 자세한 내용은 Docker API 설명서의 ContainerStats를 참조하세요.

컨테이너 에이전트 버전 `1.43.0` 이상을 실행하는 Amazon EC2 Linux 인스턴스에서 호스팅되고 `awsvpc` 또는 `bridge` 네트워크 모드를 사용하는 Amazon ECS 태스크의 경우, 응답에 추가적 네트워크 속도 통계가 포함됩니다. 그 외에 다른 모든 태스크는 응답에 누적 네트워크 통계만 포함됩니다.

Amazon ECS 작업 메타데이터 V4 JSON 응답

다음 정보가 태스크 메타데이터 엔드포인트(`${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON 응답에서 반환됩니다. 여기에는 태스크 내 각 컨테이너의 메타데이터와 해당 태스크와 연결된 메타데이터가 포함됩니다.

`Cluster`

태스크가 속한 Amazon ECS 클러스터의 Amazon 리소스 이름(ARN) 또는 약어입니다.

`ServiceName`

태스크가 속한 서비스의 이름입니다. ServiceName은 태스크가 서비스와 연결된 경우에 Amazon EC2 및 Amazon ECS Anywhere 컨테이너 인스턴스에 대해 표시됩니다.

> **Note**
>
> 이 `ServiceName` 메타데이터는 Amazon ECS 컨테이너 버전 `1.63.1` 이상을 사용할 때만 포함됩니다.

`VPCID`

Amazon EC2 컨테이너 인스턴스의 VPC ID입니다. 이 필드는 Amazon EC2 인스턴스에 대해서만 표시됩니다.

> **Note**
>
> 이 `VPCID` 메타데이터는 Amazon ECS 컨테이너 버전 `1.63.1` 이상을 사용할 때만 포함됩니다.

`TaskARN`

컨테이너가 속한 태스크의 Amazon 리소스 이름(ARN)입니다.

`Family`

태스크에 대한 Amazon ECS 태스크 정의의 패밀리입니다.

`Revision`

작업에 대한 Amazon ECS 태스크 정의의 개정입니다.

`DesiredStatus`

Amazon ECS의 태스크에 대해 원하는 상태입니다.

`KnownStatus`

Amazon ECS의 태스크에 대해 알려진 상태입니다.

`Limits`

작업 수준에서 지정된 리소스 제한입니다(예: CPU(vCPU로 표시) 및 메모리). 리소스 제한이 정의되지 않은 경우 이 파라미터가 생략됩니다.

`PullStartedAt`

첫 번째 컨테이너 이미지 풀이 시작된 시간에 대한 타임스탬프입니다.

`PullStoppedAt`

마지막 컨테이너 이미지 풀이 완료된 시간에 대한 타임스탬프입니다.

`AvailabilityZone`

태스크가 위치한 가용 영역입니다.

> **Note**
>
> 가용 영역 메타데이터는 플랫폼 버전 1.4 이상(Linux) 또는 1.0.0(Windows)을 사용하는 Fargate 태스크에만 사용할 수 있습니다.

`LaunchType`

태스크에서 사용 중인 시작 유형. 클러스터 용량 공급자를 사용할 경우 해당 태스크가 Fargate 또는 EC2 인프라를 사용한다는 것을 의미합니다.

> **Note**
>
> 이 `LaunchType` 메타데이터는 Amazon ECS Linux 컨테이너 에이전트 버전 `1.45.0` 이상(Linux) 또는 1.0.0 이상(Windows)을 사용할 때만 포함됩니다.

`Containers`

태스크와 연결된 각 컨테이너에 대한 컨테이너 메타데이터의 목록입니다.

`DockerId`

컨테이너의 Docker ID입니다.

Fargate를 사용할 때 ID는 32자리 16진수이고 그 뒤에 10자리 숫자가 붙습니다.

`Name`

태스크 정의에 지정된 컨테이너의 이름입니다.

`DockerName`

Docker에 제공된 컨테이너의 이름입니다. 동일한 태스크 정의의 여러 복사본이 단일 인스턴스에서 실행될 때 이름 충돌을 방지하기 위해 Amazon ECS 컨테이너 에이전트는 컨테이너에 고유의 이름을 생성합니다.

`Image`

컨테이너에 대한 이미지입니다.

`ImageID`

이미지 매니페스트의 SHA-256 다이제스트입니다. `repository-url/image@sha256:digest` 형식을 통해 이미지를 가져오는 데 사용할 수 있는 다이제스트입니다.

`Ports`

컨테이너에 대해 노출된 포트입니다. 노출된 포트가 없는 경우 이 파라미터가 생략됩니다.

`Labels`

컨테이너에 적용된 레이블입니다. 적용된 레이블이 없는 경우 이 파라미터가 생략됩니다.

`DesiredStatus`

Amazon ECS의 컨테이너에 대해 원하는 상태입니다.

`KnownStatus`

Amazon ECS의 컨테이너에 대해 알려진 상태입니다.

`ExitCode`

컨테이너에 대한 종료 코드입니다. 컨테이너가 종료되지 않은 경우 이 파라미터가 생략됩니다.

`Limits`

컨테이너 수준에서 지정된 리소스 제한입니다(예: CPU(CPU 단위로 표시) 및 메모리). 리소스 제한이 정의되지 않은 경우 이 파라미터가 생략됩니다.

`CreatedAt`

컨테이너가 생성된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 생성되지 않은 경우 이 파라미터가 생략됩니다.

`StartedAt`

컨테이너가 시작된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 시작되지 않은 경우 이 파라미터가 생략됩니다.

`FinishedAt`

컨테이너가 중지된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 중지되지 않은 경우 이 파라미터가 생략됩니다.

`Type`

컨테이너의 유형입니다. 태스크 정의에서 지정된 컨테이너는 NORMAL 유형입니다. Amazon ECS 컨테이너 에이전트가 내부 작업 리소스 프로비저닝에 사용하는 기타 컨테이너 유형은 무시할 수 있습니다.

`LogDriver`

컨테이너가 사용하는 로그 드라이버입니다.

> **Note**
>
> 이 `LogDriver` 메타데이터는 Amazon ECS Linux 컨테이너 버전 `1.45.0` 이상을 사용할 때만 포함됩니다.

`LogOptions`

컨테이너에 정의된 로그 드라이버 옵션입니다.

> **Note**
>
> 이 `LogOptions` 메타데이터는 Amazon ECS Linux 컨테이너 버전 `1.45.0` 이상을 사용할 때만 포함됩니다.

`ContainerARN`

컨테이너의 Amazon 리소스 이름(ARN)입니다.

> **Note**
>
> 이 `ContainerARN` 메타데이터는 Amazon ECS Linux 컨테이너 버전 `1.45.0` 이상을 사용할 때만 포함됩니다.

`Networks`

컨테이너에 대한 네트워크 정보(예: 네트워크 모드 및 IP 주소)입니다. 네트워크 정보가 정의되지 않은 경우 이 파라미터가 생략됩니다.

`RestartCount`

컨테이너가 재시작된 횟수입니다.

> **Note**
>
> `RestartCount` 메타데이터는 컨테이너에 대해 재시작 정책이 활성화된 경우에만 포함됩니다. 자세한 내용은 컨테이너 재시작 정책이 있는 Amazon ECS 작업의 개별 컨테이너 재시작 섹션을 참조하세요.

`ExecutionStoppedAt`

태스크의 `DesiredStatus`가 `STOPPED`로 이동한 시간에 대한 타임스탬프입니다. 이 동작은 필수 컨테이너가 `STOPPED`로 이동할 때 발생합니다.

Amazon ECS 작업 메타데이터 v4 예제

다음 예제에서는 각 태스크 메타데이터 엔드포인트의 예시 출력을 보여줍니다.

예제 컨테이너 메타데이터 응답

`${ECS_CONTAINER_METADATA_URI_V4}` 엔드포인트를 쿼리할 때 컨테이너 자체에 대한 메타데이터만 반환됩니다. 다음은 서비스(MyService)의 일부로 실행되는 태스크의 출력 예제입니다.

```
{
    "DockerId": "ea32192c8553fbff06c9340478a2ff089b2bb5646fb718b4ee206641c9086d66",
    "Name": "curl",
    "DockerName": "ecs-curltest-24-curl-cca48e8dcadd97805600",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/8f03e41243824aea923aca126495f665",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "24"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
```

```
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-02T00:15:07.620912337Z",
    "StartedAt": "2020-10-02T00:15:08.062559351Z",
    "Type": "NORMAL",
    "LogDriver": "awslogs",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/8f03e41243824aea923aca126495f665"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/0206b271-
b33f-47ab-86c6-a0ba208a70a9",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.100"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:9e:32:c7:48:85",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-100.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
}
```

예제 태스크 메타데이터 응답

${ECS_CONTAINER_METADATA_URI_V4}/task 엔드포인트를 쿼리할 때 태스크 내의 각 컨테이너에 대한 메타데이터 외에도 컨테이너가 속한 태스크에 대한 메타데이터가 반환됩니다. 다음은 예시 출력입니다.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
```

```
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId":
 "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED",
            "Limits": {
                "CPU": 0,
                "Memory": 0
            },
            "CreatedAt": "2020-10-02T00:43:05.602352471Z",
            "StartedAt": "2020-10-02T00:43:06.076707576Z",
            "Type": "CNI_PAUSE",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
```

```
                }
            ]
        },
        {
            "DockerId":
 "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
            "Name": "curl",
            "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
            "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
            "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "curl",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 10,
                "Memory": 128
            },
            "CreatedAt": "2020-10-02T00:43:06.326590752Z",
            "StartedAt": "2020-10-02T00:43:06.767535449Z",
            "Type": "NORMAL",
            "LogDriver": "awslogs",
            "LogOptions": {
                "awslogs-create-group": "true",
                "awslogs-group": "/ecs/metadata",
                "awslogs-region": "us-west-2",
                "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
            },
            "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
```

```
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        }
    ]
}
```

태스크 메타데이터 응답이 포함된 예제 태스크

${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags 엔드포인트를 쿼리할 때 태스크와 컨테이너 인스턴스 태그를 포함한 컨테이너가 속한 태스크에 대한 메타데이터가 반환됩니다. 다음은 예시 출력입니다.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "TaskTags": {
        "tag-use": "task-metadata-endpoint-test"
    },
    "ContainerInstanceTags":{
        "tag_key":"tag_value"
    },
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId":
 "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
```

```
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED",
            "Limits": {
                "CPU": 0,
                "Memory": 0
            },
            "CreatedAt": "2020-10-02T00:43:05.602352471Z",
            "StartedAt": "2020-10-02T00:43:06.076707576Z",
            "Type": "CNI_PAUSE",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        },
        {
            "DockerId":
 "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
            "Name": "curl",
            "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
            "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
            "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "curl",
```

```
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 10,
                "Memory": 128
            },
            "CreatedAt": "2020-10-02T00:43:06.326590752Z",
            "StartedAt": "2020-10-02T00:43:06.767535449Z",
            "Type": "NORMAL",
            "LogDriver": "awslogs",
            "LogOptions": {
                "awslogs-create-group": "true",
                "awslogs-group": "/ecs/metadata",
                "awslogs-region": "us-west-2",
                "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
            },
            "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        }
    ]
}
```

오류 메타데이터 응답과 태그가 포함된 예제 태스크

${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags 엔드포인트를 쿼리할 때 태스크와 컨테이너 인스턴스 태그를 포함한 컨테이너가 속한 태스크에 대한 메타데이터가 반환됩니다. 태깅 데이터를 검색할 때 오류가 발생한 경우 이 오류는 응답에서 반환됩니다. 다음은 컨테이너 인스턴스와 연결된 IAM 역할에 ecs:ListTagsForResource 권한이 없을 경우에 발생하는 출력의 예제입니다.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "Errors": [
        {
            "ErrorField": "ContainerInstanceTags",
            "ErrorCode": "AccessDeniedException",
            "ErrorMessage": "User: arn:aws:sts::111122223333:assumed-
role/ecsInstanceRole/i-0744a608689EXAMPLE is not authorized to perform:
 ecs:ListTagsForResource on resource: arn:aws:ecs:us-west-2:111122223333:container-
instance/default/2dd1b186f39845a584488d2ef155c131",
            "StatusCode": 400,
            "RequestId": "cd597ef0-272b-4643-9bd2-1de469870fa6",
            "ResourceARN": "arn:aws:ecs:us-west-2:111122223333:container-instance/
default/2dd1b186f39845a584488d2ef155c131"
        },
        {
            "ErrorField": "TaskTags",
            "ErrorCode": "AccessDeniedException",
            "ErrorMessage": "User: arn:aws:sts::111122223333:assumed-
role/ecsInstanceRole/i-0744a608689EXAMPLE is not authorized to perform:
 ecs:ListTagsForResource on resource: arn:aws:ecs:us-west-2:111122223333:task/
default/9ef30e4b7aa44d0db562749cff4983f3",
            "StatusCode": 400,
            "RequestId": "862c5986-6cd2-4aa6-87cc-70be395531e1",
```

```
            "ResourceARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/9ef30e4b7aa44d0db562749cff4983f3"
        }
    ],
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId":
 "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED",
            "Limits": {
                "CPU": 0,
                "Memory": 0
            },
            "CreatedAt": "2020-10-02T00:43:05.602352471Z",
            "StartedAt": "2020-10-02T00:43:06.076707576Z",
            "Type": "CNI_PAUSE",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        },
```

```
        {
            "DockerId":
 "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
            "Name": "curl",
            "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
            "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
            "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "curl",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 10,
                "Memory": 128
            },
            "CreatedAt": "2020-10-02T00:43:06.326590752Z",
            "StartedAt": "2020-10-02T00:43:06.767535449Z",
            "Type": "NORMAL",
            "LogDriver": "awslogs",
            "LogOptions": {
                "awslogs-create-group": "true",
                "awslogs-group": "/ecs/metadata",
                "awslogs-region": "us-west-2",
                "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
            },
            "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
```

```
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        }
    ]
}
```

예제 컨테이너 통계 응답

${ECS_CONTAINER_METADATA_URI_V4}/stats 엔드포인트를 쿼리할 때 컨테이너에 대한 네트워크 지표가 반환됩니다. 컨테이너 에이전트 버전 1.43.0 이상을 실행하는 Amazon EC2 인스턴스에서 호스팅되고 awsvpc 또는 bridge 네트워크 모드를 사용하는 Amazon ECS 태스크의 경우, 응답에 추가적 네트워크 속도 통계가 포함됩니다. 그 외에 다른 모든 태스크는 응답에 누적 네트워크 통계만 포함됩니다.

다음은 bridge 네트워크 모드를 사용하는 Amazon EC2에서 Amazon ECS 태스크의 예제 출력입니다.

```
{
    "read": "2020-10-02T00:51:13.410254284Z",
    "preread": "2020-10-02T00:51:12.406202398Z",
    "pids_stats": {
        "current": 3
    },
    "blkio_stats": {
        "io_service_bytes_recursive": [

        ],
        "io_serviced_recursive": [

        ],
        "io_queue_recursive": [

        ],
        "io_service_time_recursive": [

        ],
        "io_wait_time_recursive": [

        ],
        "io_merged_recursive": [

        ],
```

```
        "io_time_recursive": [

        ],
        "sectors_recursive": [

        ]
    },
    "num_procs": 0,
    "storage_stats": {

    },
    "cpu_stats": {
        "cpu_usage": {
            "total_usage": 360968065,
            "percpu_usage": [
                182359190,
                178608875
            ],
            "usage_in_kernelmode": 40000000,
            "usage_in_usermode": 290000000
        },
        "system_cpu_usage": 13939680000000,
        "online_cpus": 2,
        "throttling_data": {
            "periods": 0,
            "throttled_periods": 0,
            "throttled_time": 0
        }
    },
    "precpu_stats": {
        "cpu_usage": {
            "total_usage": 360968065,
            "percpu_usage": [
                182359190,
                178608875
            ],
            "usage_in_kernelmode": 40000000,
            "usage_in_usermode": 290000000
        },
        "system_cpu_usage": 13937670000000,
        "online_cpus": 2,
        "throttling_data": {
            "periods": 0,
            "throttled_periods": 0,
```

```
            "throttled_time": 0
        }
    },
    "memory_stats": {
        "usage": 1806336,
        "max_usage": 6299648,
        "stats": {
            "active_anon": 606208,
            "active_file": 0,
            "cache": 0,
            "dirty": 0,
            "hierarchical_memory_limit": 134217728,
            "hierarchical_memsw_limit": 268435456,
            "inactive_anon": 0,
            "inactive_file": 0,
            "mapped_file": 0,
            "pgfault": 4185,
            "pgmajfault": 0,
            "pgpgin": 2926,
            "pgpgout": 2778,
            "rss": 606208,
            "rss_huge": 0,
            "total_active_anon": 606208,
            "total_active_file": 0,
            "total_cache": 0,
            "total_dirty": 0,
            "total_inactive_anon": 0,
            "total_inactive_file": 0,
            "total_mapped_file": 0,
            "total_pgfault": 4185,
            "total_pgmajfault": 0,
            "total_pgpgin": 2926,
            "total_pgpgout": 2778,
            "total_rss": 606208,
            "total_rss_huge": 0,
            "total_unevictable": 0,
            "total_writeback": 0,
            "unevictable": 0,
            "writeback": 0
        },
        "limit": 134217728
    },
    "name": "/ecs-curltest-26-curl-c2e5f6e0cf91b0bead01",
    "id": "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af",
```

```
    "networks": {
        "eth0": {
            "rx_bytes": 84,
            "rx_packets": 2,
            "rx_errors": 0,
            "rx_dropped": 0,
            "tx_bytes": 84,
            "tx_packets": 2,
            "tx_errors": 0,
            "tx_dropped": 0
        }
    },
    "network_rate_stats": {
        "rx_bytes_per_sec": 0,
        "tx_bytes_per_sec": 0
    }
}
```

태스크 통계 응답 예

${ECS_CONTAINER_METADATA_URI_V4}/task/stats 엔드포인트를 쿼리할 때 컨테이너가 속한 태스크에 대한 네트워크 지표가 반환됩니다. 다음은 예시 출력입니다.

```
{
    "01999f2e5c6cf4df3873f28950e6278813408f281c54778efec860d0caad4854": {
        "read": "2020-10-02T00:51:32.51467703Z",
        "preread": "2020-10-02T00:51:31.50860463Z",
        "pids_stats": {
            "current": 1
        },
        "blkio_stats": {
            "io_service_bytes_recursive": [

            ],
            "io_serviced_recursive": [

            ],
            "io_queue_recursive": [

            ],
            "io_service_time_recursive": [

            ],
```

```
            "io_wait_time_recursive": [

            ],
            "io_merged_recursive": [

            ],
            "io_time_recursive": [

            ],
            "sectors_recursive": [

            ]
        },
        "num_procs": 0,
        "storage_stats": {

        },
        "cpu_stats": {
            "cpu_usage": {
                "total_usage": 177232665,
                "percpu_usage": [
                    13376224,
                    163856441
                ],
                "usage_in_kernelmode": 0,
                "usage_in_usermode": 160000000
            },
            "system_cpu_usage": 13977820000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
        "precpu_stats": {
            "cpu_usage": {
                "total_usage": 177232665,
                "percpu_usage": [
                    13376224,
                    163856441
                ],
                "usage_in_kernelmode": 0,
                "usage_in_usermode": 160000000
```

```
            },
            "system_cpu_usage": 13975800000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
        "memory_stats": {
            "usage": 532480,
            "max_usage": 6279168,
            "stats": {
                "active_anon": 40960,
                "active_file": 0,
                "cache": 0,
                "dirty": 0,
                "hierarchical_memory_limit": 9223372036854771712,
                "hierarchical_memsw_limit": 9223372036854771712,
                "inactive_anon": 0,
                "inactive_file": 0,
                "mapped_file": 0,
                "pgfault": 2033,
                "pgmajfault": 0,
                "pgpgin": 1734,
                "pgpgout": 1724,
                "rss": 40960,
                "rss_huge": 0,
                "total_active_anon": 40960,
                "total_active_file": 0,
                "total_cache": 0,
                "total_dirty": 0,
                "total_inactive_anon": 0,
                "total_inactive_file": 0,
                "total_mapped_file": 0,
                "total_pgfault": 2033,
                "total_pgmajfault": 0,
                "total_pgpgin": 1734,
                "total_pgpgout": 1724,
                "total_rss": 40960,
                "total_rss_huge": 0,
                "total_unevictable": 0,
                "total_writeback": 0,
                "unevictable": 0,
```

```
                "writeback": 0
            },
            "limit": 4073377792
        },
        "name": "/ecs-curltest-26-internalecspause-a6bcc3dbadfacfe85300",
        "id": "01999f2e5c6cf4df3873f28950e6278813408f281c54778efec860d0caad4854",
        "networks": {
            "eth0": {
                "rx_bytes": 84,
                "rx_packets": 2,
                "rx_errors": 0,
                "rx_dropped": 0,
                "tx_bytes": 84,
                "tx_packets": 2,
                "tx_errors": 0,
                "tx_dropped": 0
            }
        },
        "network_rate_stats": {
            "rx_bytes_per_sec": 0,
            "tx_bytes_per_sec": 0
        }
    },
    "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af": {
        "read": "2020-10-02T00:51:32.512771349Z",
        "preread": "2020-10-02T00:51:31.510597736Z",
        "pids_stats": {
            "current": 3
        },
        "blkio_stats": {
            "io_service_bytes_recursive": [

            ],
            "io_serviced_recursive": [

            ],
            "io_queue_recursive": [

            ],
            "io_service_time_recursive": [

            ],
            "io_wait_time_recursive": [
```

```
            ],
            "io_merged_recursive": [

            ],
            "io_time_recursive": [

            ],
            "sectors_recursive": [

            ]
        },
        "num_procs": 0,
        "storage_stats": {

        },
        "cpu_stats": {
            "cpu_usage": {
                "total_usage": 379075681,
                "percpu_usage": [
                    191355275,
                    187720406
                ],
                "usage_in_kernelmode": 60000000,
                "usage_in_usermode": 310000000
            },
            "system_cpu_usage": 13977800000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
        "precpu_stats": {
            "cpu_usage": {
                "total_usage": 378825197,
                "percpu_usage": [
                    191104791,
                    187720406
                ],
                "usage_in_kernelmode": 60000000,
                "usage_in_usermode": 310000000
            },
            "system_cpu_usage": 13975800000000,
```

```
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
        "memory_stats": {
            "usage": 1814528,
            "max_usage": 6299648,
            "stats": {
                "active_anon": 606208,
                "active_file": 0,
                "cache": 0,
                "dirty": 0,
                "hierarchical_memory_limit": 134217728,
                "hierarchical_memsw_limit": 268435456,
                "inactive_anon": 0,
                "inactive_file": 0,
                "mapped_file": 0,
                "pgfault": 5377,
                "pgmajfault": 0,
                "pgpgin": 3613,
                "pgpgout": 3465,
                "rss": 606208,
                "rss_huge": 0,
                "total_active_anon": 606208,
                "total_active_file": 0,
                "total_cache": 0,
                "total_dirty": 0,
                "total_inactive_anon": 0,
                "total_inactive_file": 0,
                "total_mapped_file": 0,
                "total_pgfault": 5377,
                "total_pgmajfault": 0,
                "total_pgpgin": 3613,
                "total_pgpgout": 3465,
                "total_rss": 606208,
                "total_rss_huge": 0,
                "total_unevictable": 0,
                "total_writeback": 0,
                "unevictable": 0,
                "writeback": 0
            },
```

```
            "limit": 134217728
        },
        "name": "/ecs-curltest-26-curl-c2e5f6e0cf91b0bead01",
        "id": "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af",
        "networks": {
            "eth0": {
                "rx_bytes": 84,
                "rx_packets": 2,
                "rx_errors": 0,
                "rx_dropped": 0,
                "tx_bytes": 84,
                "tx_packets": 2,
                "tx_errors": 0,
                "tx_dropped": 0
            }
        },
        "network_rate_stats": {
            "rx_bytes_per_sec": 0,
            "tx_bytes_per_sec": 0
        }
    }
}
```


Amazon ECS 작업 메타데이터 엔드포인트 버전 3

> **Important**
>
> 태스크 메타데이터 버전 3 엔드포인트는 더 이상 능동적으로 관리되지 않습니다. 최신 메타데이터 엔드포인트 정보를 가져오려면 태스크 메타데이터 버전 4 엔드포인트를 업데이트하는 것이 좋습니다. 자세한 내용은 the section called “태스크 메타데이터 엔드포인트 버전 4” 섹션을 참조하세요.
>
> AWS Fargate에 호스팅된 Amazon ECS 작업을 사용하는 경우 Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 엔드포인트 버전 3을 참조하세요.

Amazon ECS 컨테이너 에이전트의 버전 1.21.0부터 에이전트는 `ECS_CONTAINER_METADATA_URI`라는 이름의 환경 변수를 작업의 각 컨테이너에 주입합니다. 태스크 메타데이터 버전 3 엔드포인트를 쿼리하면 다양한 태스크 메타데이터와 Docker 통계를 태스크에 사용할 수 있습니다. `bridge` 네트워크 모드를 사용하는 태스크의 경우 `/stats` 엔드포인트를 쿼리할 때 네트워크 지표를 사용할 수 있습니다.

태스크 메타데이터 엔드포인트 버전 3 기능은 기본적으로 플랫폼 버전 v1.3.0 이상에서 Fargate 시작 유형을 사용하는 태스크에서 사용할 수 있으며, EC2 시작 유형을 사용하면서 버전 1.21.0 이상의 Amazon EC2 컨테이너 에이전트를 실행하는 Amazon EC2 Linux 인프라 또는 Amazon ECS 컨테이너 에이전트 버전 `1.54.0`을 실행하고 `awsvpc` 네트워크 모드를 사용하는 Amazon ECS2 Windows 인프라에서 시작되는 태스크에도 사용할 수 있습니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 관리 섹션을 참조하세요.

에이전트를 최신 버전으로 업데이트하여 이전 버전의 컨테이너 인스턴스에서 이 기능에 대한 지원을 추가할 수 있습니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트을 참조하세요.

> **Important**
>
> Fargate 시작 유형 및 v1.3.0 이전 버전의 플랫폼을 사용하는 태스크의 경우 태스크 메타데이터 버전 2 엔드포인트가 지원됩니다. 자세한 정보는 Amazon ECS 작업 메타데이터 엔드포인트 버전 2을 참조하세요.

작업 메타데이터 엔드포인트 버전 3 경로

다음의 태스크 메타데이터 엔드포인트를 컨테이너에 사용할 수 있습니다.

`${ECS_CONTAINER_METADATA_URI}`

이 경로는 컨테이너에 대한 메타데이터 JSON을 반환합니다.

`${ECS_CONTAINER_METADATA_URI}/task`

이 경로는 태스크와 연결된 모든 컨테이너의 컨테이너 ID 및 이름 목록을 포함하여 태스크에 대한 메타데이터 JSON을 반환합니다. 이 엔드포인트의 응답에 대한 자세한 내용은 Amazon ECS 작업 메타데이터 v3 JSON 응답 섹션을 참조하세요.

`${ECS_CONTAINER_METADATA_URI}/taskWithTags`

이 경로는 `ListTagsForResource` API를 사용하여 검색할 수 있는 태스크와 컨테이너 인스턴스 태그 외에도 `/task` 엔드포인트에 포함된 태스크에 대한 메타데이터를 반환합니다.

`${ECS_CONTAINER_METADATA_URI}/stats`

이 경로는 특정 Docker 컨테이너에 대한 Docker 통계 JSON을 반환합니다. 반환된 각 통계에 대한 자세한 내용은 Docker API 설명서의 ContainerStats를 참조하세요.

`${ECS_CONTAINER_METADATA_URI}/task/stats`

이 경로는 태스크와 연결된 모든 컨테이너에 대한 Docker 통계 JSON을 반환합니다. 반환된 각 통계에 대한 자세한 내용은 Docker API 설명서의 ContainerStats를 참조하세요.

Amazon ECS 작업 메타데이터 v3 JSON 응답

다음 정보가 태스크 메타데이터 엔드포인트(`${ECS_CONTAINER_METADATA_URI}/task`) JSON 응답에서 반환됩니다.

`Cluster`

태스크가 속한 Amazon ECS 클러스터의 Amazon 리소스 이름(ARN) 또는 약어입니다.

`TaskARN`

컨테이너가 속한 태스크의 Amazon 리소스 이름(ARN)입니다.

`Family`

태스크에 대한 Amazon ECS 태스크 정의의 패밀리입니다.

`Revision`

작업에 대한 Amazon ECS 태스크 정의의 개정입니다.

`DesiredStatus`

Amazon ECS의 태스크에 대해 원하는 상태입니다.

`KnownStatus`

Amazon ECS의 태스크에 대해 알려진 상태입니다.

`Limits`

작업 수준에서 지정된 리소스 제한입니다(예: CPU(vCPU로 표시) 및 메모리). 리소스 제한이 정의되지 않은 경우 이 파라미터가 생략됩니다.

`PullStartedAt`

첫 번째 컨테이너 이미지 풀이 시작된 시간에 대한 타임스탬프입니다.

`PullStoppedAt`

마지막 컨테이너 이미지 풀이 완료된 시간에 대한 타임스탬프입니다.

`AvailabilityZone`

태스크가 위치한 가용 영역입니다.

> **Note**
>
> 가용 영역 메타데이터는 플랫폼 버전 1.4 이상(Linux) 또는 1.0.0 이상(Windows)을 사용하는 Fargate 태스크에만 사용할 수 있습니다.

`Containers`

태스크와 연결된 각 컨테이너에 대한 컨테이너 메타데이터의 목록입니다.

`DockerId`

컨테이너의 Docker ID입니다.

`Name`

태스크 정의에 지정된 컨테이너의 이름입니다.

`DockerName`

Docker에 제공된 컨테이너의 이름입니다. 동일한 태스크 정의의 여러 복사본이 단일 인스턴스에서 실행될 때 이름 충돌을 방지하기 위해 Amazon ECS 컨테이너 에이전트는 컨테이너에 고유의 이름을 생성합니다.

`Image`

컨테이너에 대한 이미지입니다.

`ImageID`

이미지 매니페스트의 SHA-256 다이제스트입니다. `repository-url/image@sha256:digest` 형식을 통해 이미지를 가져오는 데 사용할 수 있는 다이제스트입니다.

`Ports`

컨테이너에 대해 노출된 포트입니다. 노출된 포트가 없는 경우 이 파라미터가 생략됩니다.

`Labels`

컨테이너에 적용된 레이블입니다. 적용된 레이블이 없는 경우 이 파라미터가 생략됩니다.

`DesiredStatus`

Amazon ECS의 컨테이너에 대해 원하는 상태입니다.

`KnownStatus`

Amazon ECS의 컨테이너에 대해 알려진 상태입니다.

`ExitCode`

컨테이너에 대한 종료 코드입니다. 컨테이너가 종료되지 않은 경우 이 파라미터가 생략됩니다.

`Limits`

컨테이너 수준에서 지정된 리소스 제한입니다(예: CPU(CPU 단위로 표시) 및 메모리). 리소스 제한이 정의되지 않은 경우 이 파라미터가 생략됩니다.

`CreatedAt`

컨테이너가 생성된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 생성되지 않은 경우 이 파라미터가 생략됩니다.

`StartedAt`

컨테이너가 시작된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 시작되지 않은 경우 이 파라미터가 생략됩니다.

`FinishedAt`

컨테이너가 중지된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 중지되지 않은 경우 이 파라미터가 생략됩니다.

`Type`

컨테이너의 유형입니다. 태스크 정의에서 지정된 컨테이너는 `NORMAL` 유형입니다. Amazon ECS 컨테이너 에이전트가 내부 작업 리소스 프로비저닝에 사용하는 기타 컨테이너 유형은 무시할 수 있습니다.

`Networks`

컨테이너에 대한 네트워크 정보(예: 네트워크 모드 및 IP 주소)입니다. 네트워크 정보가 정의되지 않은 경우 이 파라미터가 생략됩니다.

`ClockDrift`

기준 시간과 시스템 시간의 차이에 대한 정보입니다. 이는 Linux 운영 체제에 적용됩니다. 이 기능은 Amazon Time Sync Service를 사용하여 클럭 정확도를 측정하고 컨테이너에 대한 클록 오류 범

위를 제공합니다. 자세한 내용은 Amazon EC2 Linux용 사용 설명서의 Linux 인스턴스에 대한 시간 설정을 참조하세요.

`ReferenceTime`

클록 정확도의 기초입니다. Amazon ECS는 NTP를 통해 협정 세계시(UTC) 글로벌 표준을 사용합니다(예: `2021-09-07T16:57:44Z`).

`ClockErrorBound`

UTC에 대한 오프셋으로 정의되는 클록 오류 측정값입니다. 이 오류는 기준 시간과 시스템 시간의 차이(밀리초)입니다.

`ClockSynchronizationStatus`

시스템 시간과 기준 시간 간의 가장 최근 동기화 시도가 성공했는지를 나타냅니다.

유효 값은 `SYNCHRONIZED` 및 `NOT_SYNCHRONIZED`입니다.

`ExecutionStoppedAt`

태스크의 `DesiredStatus`가 `STOPPED`로 이동한 시간에 대한 타임스탬프입니다. 이 동작은 필수 컨테이너가 `STOPPED`로 이동할 때 발생합니다.

Amazon ECS 작업 메타데이터 v3 예제

다음 예제에서는 태스크 메타데이터 엔드포인트의 샘플 출력을 보여줍니다.

예제 컨테이너 메타데이터 응답

`${ECS_CONTAINER_METADATA_URI}` 엔드포인트를 쿼리할 때 컨테이너 자체에 대한 메타데이터만 반환됩니다. 다음은 예시 출력입니다.

```
{
    "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
    "Name": "nginx-curl",
    "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
    "Image": "nrdlngr/nginx-curl",
    "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
```

```
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 512,
        "Memory": 512
    },
    "CreatedAt": "2018-02-01T20:55:10.554941919Z",
    "StartedAt": "2018-02-01T20:55:11.064236631Z",
    "Type": "NORMAL",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.106"
            ]
        }
    ]
}
```

예제 작업 메타데이터 응답

${ECS_CONTAINER_METADATA_URI}/task 엔드포인트를 쿼리할 때 컨테이너가 속한 태스크에 대한 메타데이터가 반환됩니다. 다음은 예시 출력입니다.

다음은 단일 컨테이너 태스크에 대한 JSON 응답입니다.

```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
```

```
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
```

```
      "Limits": {
        "CPU": 512,
        "Memory": 512
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```


Amazon ECS 작업 메타데이터 엔드포인트 버전 2

> **Important**
>
> 태스크 메타데이터 버전 2 엔드포인트는 더 이상 능동적으로 관리되지 않습니다. 최신 메타데이터 엔드포인트 정보를 가져오려면 태스크 메타데이터 버전 4 엔드포인트를 업데이트하는 것이 좋습니다. 자세한 정보는 the section called “태스크 메타데이터 엔드포인트 버전 4”을 참조하세요.

Amazon ECS 컨테이너 에이전트 버전 1.17.0부터 다양한 태스크 메타데이터와 Docker 통계가 Amazon ECS 컨테이너 에이전트에서 제공하는 HTTP 엔드포인트에서 awsvpc 네트워크 모드를 사용하는 태스크에 제공됩니다.

awsvpc 네트워크 모드를 사용하여 시작되는 태스크에 속한 모든 컨테이너는 미리 정의된 링크-로컬 주소 범위 내의 로컬 IPv4 주소를 받습니다. 컨테이너가 메타데이터 엔드포인트를 쿼리할 때 Amazon ECS 컨테이너 에이전트는 고유의 IP 주소를 기반으로 컨테이너가 어떤 태스크에 속하는지를 확인할 수 있으며 해당 태스크에 대한 메타데이터와 통계가 반환됩니다.

태스크 메타데이터 활성화

태스크 메타데이터 버전 2 기능은 기본적으로 다음 작업에 대해 사용하도록 설정됩니다.

- 플랫폼 버전 v1.1.0 이상을 사용하는 Fargate 시작 유형을 사용하는 작업. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.
- 똑같이 `awsvpc` 네트워크 모드를 사용하는 EC2 시작 유형을 사용하고, Amazon ECS 컨테이너 에이전트 버전 1.17.0 이상에서 실행되는 Amazon EC2 Linux 인프라 또는 Amazon ECS 컨테이너 에이전트 버전 `1.54.0` 이상에서 실행되는 Amazon EC2 Windows 인프라에서 시작된 태스크입니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 관리 섹션을 참조하세요.

에이전트를 최신 버전으로 업데이트하여 이전 버전의 컨테이너 인스턴스에서 이 기능에 대한 지원을 추가할 수 있습니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트을 참조하세요.

태스크 메타데이터 엔드포인트 경로

다음 API 엔드포인트를 컨테이너에 사용할 수 있습니다.

`169.254.170.2/v2/metadata`

이 엔드포인트는 태스크와 연결된 모든 컨테이너의 컨테이너 ID 및 이름 목록을 포함하여 태스크에 대한 메타데이터 JSON을 반환합니다. 이 엔드포인트의 응답에 대한 자세한 내용은 태스크 메타데이터 JSON 응답 섹션을 참조하세요.

`169.254.170.2/v2/metadata/<container-id>`

이 엔드포인트는 지정된 Docker 컨테이너 ID에 대한 메타데이터 JSON을 반환합니다.

`169.254.170.2/v2/metadata/taskWithTags`

이 경로는 `ListTagsForResource` API를 사용하여 검색할 수 있는 태스크와 컨테이너 인스턴스 태그 외에도 `/task` 엔드포인트에 포함된 태스크에 대한 메타데이터를 반환합니다.

`169.254.170.2/v2/stats`

이 엔드포인트는 태스크와 연결된 모든 컨테이너에 대한 Docker 통계 JSON을 반환합니다. 반환된 각 통계에 대한 자세한 내용은 Docker API 설명서의 ContainerStats를 참조하세요.

`169.254.170.2/v2/stats/<container-id>`

이 엔드포인트는 지정된 Docker 컨테이너 ID에 대한 Docker 통계 JSON을 반환합니다. 반환된 각 통계에 대한 자세한 내용은 Docker API 설명서의 ContainerStats를 참조하세요.

태스크 메타데이터 JSON 응답

다음 정보가 태스크 메타데이터 엔드포인트(169.254.170.2/v2/metadata) JSON 응답에서 반환됩니다.

`Cluster`

태스크가 속한 Amazon ECS 클러스터의 Amazon 리소스 이름(ARN) 또는 약어입니다.

`TaskARN`

컨테이너가 속한 태스크의 Amazon 리소스 이름(ARN)입니다.

`Family`

태스크에 대한 Amazon ECS 태스크 정의의 패밀리입니다.

`Revision`

작업에 대한 Amazon ECS 태스크 정의의 개정입니다.

`DesiredStatus`

Amazon ECS의 태스크에 대해 원하는 상태입니다.

`KnownStatus`

Amazon ECS의 태스크에 대해 알려진 상태입니다.

`Limits`

작업 수준에서 지정된 리소스 제한입니다(예: CPU(vCPU로 표시) 및 메모리). 리소스 제한이 정의되지 않은 경우 이 파라미터가 생략됩니다.

`PullStartedAt`

첫 번째 컨테이너 이미지 풀이 시작된 시간에 대한 타임스탬프입니다.

`PullStoppedAt`

마지막 컨테이너 이미지 풀이 완료된 시간에 대한 타임스탬프입니다.

`AvailabilityZone`

태스크가 위치한 가용 영역입니다.

> **Note**
>
> 가용 영역 메타데이터는 플랫폼 버전 1.4 이상(Linux) 또는 1.0.0 이상(Windows)을 사용하는 Fargate 태스크에만 사용할 수 있습니다.

`Containers`

태스크와 연결된 각 컨테이너에 대한 컨테이너 메타데이터의 목록입니다.

`DockerId`

컨테이너의 Docker ID입니다.

`Name`

태스크 정의에 지정된 컨테이너의 이름입니다.

`DockerName`

Docker에 제공된 컨테이너의 이름입니다. 동일한 태스크 정의의 여러 복사본이 단일 인스턴스에서 실행될 때 이름 충돌을 방지하기 위해 Amazon ECS 컨테이너 에이전트는 컨테이너에 고유의 이름을 생성합니다.

`Image`

컨테이너에 대한 이미지입니다.

`ImageID`

이미지 매니페스트의 SHA-256 다이제스트입니다. `repository-url/image@sha256:digest` 형식을 통해 이미지를 가져오는 데 사용할 수 있는 다이제스트입니다.

`Ports`

컨테이너에 대해 노출된 포트입니다. 노출된 포트가 없는 경우 이 파라미터가 생략됩니다.

`Labels`

컨테이너에 적용된 레이블입니다. 적용된 레이블이 없는 경우 이 파라미터가 생략됩니다.

`DesiredStatus`

Amazon ECS의 컨테이너에 대해 원하는 상태입니다.

`KnownStatus`

Amazon ECS의 컨테이너에 대해 알려진 상태입니다.

`ExitCode`

컨테이너에 대한 종료 코드입니다. 컨테이너가 종료되지 않은 경우 이 파라미터가 생략됩니다.

`Limits`

컨테이너 수준에서 지정된 리소스 제한입니다(예: CPU(CPU 단위로 표시) 및 메모리). 리소스 제한이 정의되지 않은 경우 이 파라미터가 생략됩니다.

`CreatedAt`

컨테이너가 생성된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 생성되지 않은 경우 이 파라미터가 생략됩니다.

`StartedAt`

컨테이너가 시작된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 시작되지 않은 경우 이 파라미터가 생략됩니다.

`FinishedAt`

컨테이너가 중지된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 중지되지 않은 경우 이 파라미터가 생략됩니다.

`Type`

컨테이너의 유형입니다. 태스크 정의에서 지정된 컨테이너는 `NORMAL` 유형입니다. Amazon ECS 컨테이너 에이전트가 내부 작업 리소스 프로비저닝에 사용하는 기타 컨테이너 유형은 무시할 수 있습니다.

`Networks`

컨테이너에 대한 네트워크 정보(예: 네트워크 모드 및 IP 주소)입니다. 네트워크 정보가 정의되지 않은 경우 이 파라미터가 생략됩니다.

`ClockDrift`

기준 시간과 시스템 시간의 차이에 대한 정보입니다. 이는 Linux 운영 체제에 적용됩니다. 이 기능은 Amazon Time Sync Service를 사용하여 클럭 정확도를 측정하고 컨테이너에 대한 클록 오류 범위를 제공합니다. 자세한 내용은 Amazon EC2 Linux용 사용 설명서의 Linux 인스턴스에 대한 시간 설정을 참조하세요.

`ReferenceTime`

클록 정확도의 기초입니다. Amazon ECS는 NTP를 통해 협정 세계시(UTC) 글로벌 표준을 사용합니다(예: `2021-09-07T16:57:44Z`).

`ClockErrorBound`

UTC에 대한 오프셋으로 정의되는 클록 오류 측정값입니다. 이 오류는 기준 시간과 시스템 시간의 차이(밀리초)입니다.

`ClockSynchronizationStatus`

시스템 시간과 기준 시간 간의 가장 최근 동기화 시도가 성공했는지를 나타냅니다.

유효 값은 `SYNCHRONIZED` 및 `NOT_SYNCHRONIZED`입니다.

`ExecutionStoppedAt`

태스크의 `DesiredStatus`가 `STOPPED`로 이동한 시간에 대한 타임스탬프입니다. 이 동작은 필수 컨테이너가 `STOPPED`로 이동할 때 발생합니다.

예제 태스크 메타데이터 응답

다음은 단일 컨테이너 태스크에 대한 JSON 응답입니다.

```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
```

```
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
```

```
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```

Fargate의 작업에 대해 사용 가능한 Amazon ECS 작업 메타데이터

Fargate의 Amazon ECS는 속한 작업 및 컨테이너에 대한 다양한 메타데이터, 네트워크 지표 및 Docker 통계를 검색하는 방법을 제공합니다. 이를 일컬어 태스크 메타데이터 엔드포인트라고 합니다. Fargate의 Amazon ECS 작업에 사용할 수 있는 작업 메타데이터 엔드포인트 버전은 다음과 같습니다.

- 작업 메타데이터 엔드포인트 버전 4 – 플랫폼 버전 1.4.0 이상을 사용하는 작업에 사용할 수 있습니다.
- 작업 메타데이터 엔드포인트 버전 3 – 플랫폼 버전 1.1.0 이상을 사용하는 작업에 사용할 수 있습니다.

awsvpc 네트워크 모드를 사용하여 시작되는 태스크에 속한 모든 컨테이너는 미리 정의된 링크-로컬 주소 범위 내의 로컬 IPv4 주소를 받습니다. 컨테이너가 메타데이터 엔드포인트를 쿼리할 때 컨테이너 에이전트는 고유의 IP 주소를 기반으로 컨테이너가 어떤 작업에 속하는지를 확인할 수 있으며 해당 작업에 대한 메타데이터와 통계가 반환됩니다.

주제

- Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 엔드포인트 버전 4
- Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 엔드포인트 버전 3

Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 엔드포인트 버전 4

> **Important**
>
> Amazon EC2 인스턴스에서 호스팅된 Amazon ECS 작업을 사용하는 경우 Amazon ECS 작업 메타데이터 엔드포인트를 참조하세요.

Fargate 플랫폼 버전 `1.4.0`부터 `ECS_CONTAINER_METADATA_URI_V4`라는 환경 변수가 태스크의 각 컨테이너에 주입됩니다. 작업 메타데이터 엔드포인트 버전 4를 쿼리하면 다양한 작업 메타데이터와 Docker 통계를 작업에 사용할 수 있습니다.

작업 메타데이터 엔드포인트 버전 4는 버전 3 엔드포인트와 유사하지만 컨테이너 및 작업에 대한 추가 네트워크 메타데이터를 제공합니다. `/stats` 엔드포인트를 쿼리할 때도 추가 네트워크 지표를 사용할 수 있습니다.

작업 메타데이터 엔드포인트는 기본적으로 플랫폼 버전 `1.4.0` 이상을 사용하는 AWS Fargate에서 실행되는 모든 Amazon ECS 작업에 대해 켜집니다.

> **Note**
>
> 나중에 새 태스크 메타데이터 엔드포인트 버전을 생성할 필요가 없도록 버전 4 출력에 메타데이터를 추가할 수 있습니다. 기존 메타데이터를 제거하거나 메타데이터 필드 이름을 변경하지 않습니다.

Fargate 작업 메타데이터 엔드포인트 버전 4 경로

다음의 태스크 메타데이터 엔드포인트를 컨테이너에 사용할 수 있습니다.

`${ECS_CONTAINER_METADATA_URI_V4}`

이 경로는 컨테이너에 대한 메타데이터를 반환합니다.

`${ECS_CONTAINER_METADATA_URI_V4}/task`

이 경로는 태스크와 연결된 모든 컨테이너의 컨테이너 ID 및 이름 목록을 포함하여 작업에 대한 메타데이터를 반환합니다. 이 엔드포인트의 응답에 대한 자세한 내용은 Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 v4 JSON 응답 섹션을 참조하세요.

`${ECS_CONTAINER_METADATA_URI_V4}/stats`

이 경로는 Docker 컨테이너에 대한 Docker 통계를 반환합니다. 반환된 각 통계에 대한 자세한 내용은 Docker API 설명서의 ContainerStats를 참조하세요.

> **Note**
>
> AWS Fargate의 Amazon ECS 태스크는 컨테이너 통계를 반환하기 전에 ~1초 동안 컨테이너를 실행해야 합니다.

`${ECS_CONTAINER_METADATA_URI_V4}/task/stats`

이 경로는 태스크와 연결된 모든 컨테이너에 대한 Docker 통계를 반환합니다. 반환된 각 통계에 대한 자세한 내용은 Docker API 설명서의 ContainerStats를 참조하세요.

> **Note**
>
> AWS Fargate의 Amazon ECS 태스크는 컨테이너 통계를 반환하기 전에 ~1초 동안 컨테이너를 실행해야 합니다.

Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 v4 JSON 응답

다음 메타데이터가 작업 메타데이터 엔드포인트(`${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON 응답에서 반환됩니다.

`Cluster`

태스크가 속한 Amazon ECS 클러스터의 Amazon 리소스 이름(ARN) 또는 약어입니다.

`VPCID`

Amazon EC2 컨테이너 인스턴스의 VPC ID입니다. 이 필드는 Amazon EC2 인스턴스에 대해서만 표시됩니다.

> **Note**
>
> 이 `VPCID` 메타데이터는 Amazon ECS 컨테이너 버전 `1.63.1` 이상을 사용할 때만 포함됩니다.

`TaskARN`

컨테이너가 속한 태스크의 Amazon 리소스 이름(ARN)입니다.

`Family`

태스크에 대한 Amazon ECS 태스크 정의의 패밀리입니다.

`Revision`

작업에 대한 Amazon ECS 태스크 정의의 개정입니다.

`DesiredStatus`

Amazon ECS의 태스크에 대해 원하는 상태입니다.

`KnownStatus`

Amazon ECS의 태스크에 대해 알려진 상태입니다.

`Limits`

작업 수준에서 지정된 리소스 제한입니다(예: CPU(vCPU로 표시) 및 메모리). 리소스 제한이 정의되지 않은 경우 이 파라미터가 생략됩니다.

`PullStartedAt`

첫 번째 컨테이너 이미지 풀이 시작된 시간에 대한 타임스탬프입니다.

`PullStoppedAt`

마지막 컨테이너 이미지 풀이 완료된 시간에 대한 타임스탬프입니다.

`AvailabilityZone`

태스크가 위치한 가용 영역입니다.

> **Note**
>
> 가용 영역 메타데이터는 플랫폼 버전 1.4 이상(Linux) 또는 1.0.0(Windows)을 사용하는 Fargate 태스크에만 사용할 수 있습니다.

`LaunchType`

태스크에서 사용 중인 시작 유형. 클러스터 용량 공급자를 사용할 경우 해당 태스크가 Fargate 또는 EC2 인프라를 사용한다는 것을 의미합니다.

> **Note**
>
> 이 LaunchType 메타데이터는 Amazon ECS Linux 컨테이너 에이전트 버전 1.45.0 이상(Linux) 또는 1.0.0 이상(Windows)을 사용할 때만 포함됩니다.

EphemeralStorageMetrics

이 작업의 임시 스토리지에 대한 예약된 크기와 현재 사용량입니다.

> **Note**
>
> Fargate는 디스크 공간을 예약합니다. Fargate에서만 사용됩니다. 이에 대한 요금은 청구되지 않습니다. 이러한 지표에는 표시되지 않습니다. 그러나 df 등의 다른 도구에서는 이 추가 스토리지를 볼 수 있습니다.

Utilized

이 작업의 현재 임시 스토리지 사용량(MiB)입니다.

Reserved

이 작업의 예약된 임시 스토리지(MiB)입니다. 임시 스토리지의 크기는 실행 중인 작업에서 변경할 수 없습니다. 작업 정의에서 ephermalStorage 객체를 지정하여 임시 스토리지의 양을 변경할 수 있습니다. ephermalStorage는 MiB가 아닌 GiB 단위로 지정됩니다. ephermalStorage 및 EphemeralStorageMetrics는 Fargate Linux 플랫폼 버전 1.4.0 이상에서만 사용할 수 있습니다.

Containers

태스크와 연결된 각 컨테이너에 대한 컨테이너 메타데이터의 목록입니다.

DockerId

컨테이너의 Docker ID입니다.

Fargate를 사용할 때 ID는 32자리 16진수이고 그 뒤에 10자리 숫자가 붙습니다.

Name

태스크 정의에 지정된 컨테이너의 이름입니다.

`DockerName`

Docker에 제공된 컨테이너의 이름입니다. 동일한 태스크 정의의 여러 복사본이 단일 인스턴스에서 실행될 때 이름 충돌을 방지하기 위해 Amazon ECS 컨테이너 에이전트는 컨테이너에 고유의 이름을 생성합니다.

`Image`

컨테이너에 대한 이미지입니다.

`ImageID`

이미지 매니페스트의 SHA-256 다이제스트입니다. `repository-url/image@sha256:digest` 형식을 통해 이미지를 가져오는 데 사용할 수 있는 다이제스트입니다.

`Ports`

컨테이너에 대해 노출된 포트입니다. 노출된 포트가 없는 경우 이 파라미터가 생략됩니다.

`Labels`

컨테이너에 적용된 레이블입니다. 적용된 레이블이 없는 경우 이 파라미터가 생략됩니다.

`DesiredStatus`

Amazon ECS의 컨테이너에 대해 원하는 상태입니다.

`KnownStatus`

Amazon ECS의 컨테이너에 대해 알려진 상태입니다.

`ExitCode`

컨테이너에 대한 종료 코드입니다. 컨테이너가 종료되지 않은 경우 이 파라미터가 생략됩니다.

`Limits`

컨테이너 수준에서 지정된 리소스 제한입니다(예: CPU(CPU 단위로 표시) 및 메모리). 리소스 제한이 정의되지 않은 경우 이 파라미터가 생략됩니다.

`CreatedAt`

컨테이너가 생성된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 생성되지 않은 경우 이 파라미터가 생략됩니다.

`StartedAt`

컨테이너가 시작된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 시작되지 않은 경우 이 파라미터가 생략됩니다.

`FinishedAt`

컨테이너가 중지된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 중지되지 않은 경우 이 파라미터가 생략됩니다.

`Type`

컨테이너의 유형입니다. 태스크 정의에서 지정된 컨테이너는 `NORMAL` 유형입니다. Amazon ECS 컨테이너 에이전트가 내부 작업 리소스 프로비저닝에 사용하는 기타 컨테이너 유형은 무시할 수 있습니다.

`LogDriver`

컨테이너가 사용하는 로그 드라이버입니다.

> **Note**
>
> 이 `LogDriver` 메타데이터는 Amazon ECS Linux 컨테이너 버전 `1.45.0` 이상을 사용할 때만 포함됩니다.

`LogOptions`

컨테이너에 정의된 로그 드라이버 옵션입니다.

> **Note**
>
> 이 `LogOptions` 메타데이터는 Amazon ECS Linux 컨테이너 버전 `1.45.0` 이상을 사용할 때만 포함됩니다.

`ContainerARN`

컨테이너의 Amazon 리소스 이름(ARN)입니다.

> **Note**
>
> 이 `ContainerARN` 메타데이터는 Amazon ECS Linux 컨테이너 버전 `1.45.0` 이상을 사용할 때만 포함됩니다.

`Networks`

컨테이너에 대한 네트워크 정보(예: 네트워크 모드 및 IP 주소)입니다. 네트워크 정보가 정의되지 않은 경우 이 파라미터가 생략됩니다.

`Snapshotter`

containerd가 이 컨테이너 이미지를 다운로드하는 사용한 snapshotter입니다. 유효한 값은 기본값인 `overlayfs` 및 SOCI 인덱스로 지연 로드할 때 사용되는 `soci`입니다. 이 파라미터는 Linux 플랫폼 버전 `1.4.0`에서 실행되는 작업에만 사용할 수 있습니다.

`RestartCount`

컨테이너가 재시작된 횟수입니다.

> **Note**
>
> `RestartCount` 메타데이터는 컨테이너에 대해 재시작 정책이 활성화된 경우에만 포함됩니다. 자세한 내용은 컨테이너 재시작 정책이 있는 Amazon ECS 작업의 개별 컨테이너 재시작 섹션을 참조하세요.

`ClockDrift`

기준 시간과 시스템 시간의 차이에 대한 정보입니다. 이 기능은 Amazon Time Sync Service를 사용하여 클럭 정확도를 측정하고 컨테이너에 대한 클록 오류 범위를 제공합니다. 자세한 내용은 Amazon EC2 Linux용 사용 설명서의 Linux 인스턴스에 대한 시간 설정을 참조하세요.

`ReferenceTime`

클록 정확도의 기초입니다. Amazon ECS는 NTP를 통해 협정 세계시(UTC) 글로벌 표준을 사용합니다(예: `2021-09-07T16:57:44Z`).

`ClockErrorBound`

UTC에 대한 오프셋으로 정의되는 클록 오류 측정값입니다. 이 오류는 기준 시간과 시스템 시간의 차이(밀리초)입니다.

ClockSynchronizationStatus

시스템 시간과 기준 시간 간의 가장 최근 동기화 시도가 성공했는지를 나타냅니다.

유효 값은 SYNCHRONIZED 및 NOT_SYNCHRONIZED입니다.

ExecutionStoppedAt

태스크의 DesiredStatus가 STOPPED로 이동한 시간에 대한 타임스탬프입니다. 이 동작은 필수 컨테이너가 STOPPED로 이동할 때 발생합니다.

Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 v4 예제

다음 예제에서는 AWS Fargate에서 실행되는 Amazon ECS 작업에 대한 작업 메타데이터 엔드포인트의 샘플 출력을 보여줍니다.

컨테이너에서 curl 뒤에 작업 메타데이터 엔드포인트를 사용하여 엔드포인트를 쿼리할 수 있습니다(예: `curl ${ECS_CONTAINER_METADATA_URI_V4}/task`).

예제 컨테이너 메타데이터 응답

`${ECS_CONTAINER_METADATA_URI_V4}` 엔드포인트를 쿼리할 때 컨테이너 자체에 대한 메타데이터만 반환됩니다. 다음은 예시 출력입니다.

```
{
    "DockerId": "cd189a933e5849daa93386466019ab50-2495160603",
    "Name": "curl",
    "DockerName": "curl",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID":
 "sha256:25f3695bedfb454a50f12d127839a68ad3caf91e451c1da073db34c542c4d2cb",
    "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-west-2:111122223333:cluster/
default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/
cd189a933e5849daa93386466019ab50",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "2"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
```

```
        "Memory": 128
    },
    "CreatedAt": "2020-10-08T20:09:11.44527186Z",
    "StartedAt": "2020-10-08T20:09:11.44527186Z",
    "Type": "NORMAL",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "192.0.2.3"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0a:de:f6:10:51:e5",
            "IPv4SubnetCIDRBlock": "192.0.2.0/24",
            "DomainNameServers": [
                "192.0.2.2"
            ],
            "DomainNameSearchList": [
                "us-west-2.compute.internal"
            ],
            "PrivateDNSName": "ip-10-0-0-222.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "192.0.2.0/24"
        }
    ],
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/05966557-
f16c-49cb-9352-24b3a0dcd0e1",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/containerlogs",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/cd189a933e5849daa93386466019ab50"
    },
    "LogDriver": "awslogs",
    "Snapshotter": "overlayfs"
}
```

Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 v4 예제

`${ECS_CONTAINER_METADATA_URI_V4}/task` 엔드포인트를 쿼리할 때 컨테이너가 속한 태스크에 대한 메타데이터가 반환됩니다. 다음은 예시 출력입니다.

```
{
  "Cluster": "arn:aws:ecs:us-east-1:123456789012:cluster/MyEmptyCluster",
```

```
  "TaskARN": "arn:aws:ecs:us-east-1:123456789012:task/MyEmptyCluster/
bfa2636268144d039771334145e490c5",
  "Family": "sample-fargate",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Limits": {
    "CPU": 0.25,
    "Memory": 512
  },
  "PullStartedAt": "2023-07-21T15:45:33.532811081Z",
  "PullStoppedAt": "2023-07-21T15:45:38.541068435Z",
  "AvailabilityZone": "us-east-1d",
  "Containers": [
    {
      "DockerId": "bfa2636268144d039771334145e490c5-1117626119",
      "Name": "curl-image",
      "DockerName": "curl-image",
      "Image": "curlimages/curl",
      "ImageID":
 "sha256:daf3f46a2639c1613b25e85c9ee4193af8a1d538f92483d67f9a3d7f21721827",
      "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-east-1:123456789012:cluster/
MyEmptyCluster",
        "com.amazonaws.ecs.container-name": "curl-image",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-1:123456789012:task/
MyEmptyCluster/bfa2636268144d039771334145e490c5",
        "com.amazonaws.ecs.task-definition-family": "sample-fargate",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": { "CPU": 128 },
      "CreatedAt": "2023-07-21T15:45:44.91368314Z",
      "StartedAt": "2023-07-21T15:45:44.91368314Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": ["172.31.42.189"],
          "AttachmentIndex": 0,
          "MACAddress": "0e:98:9f:33:76:d3",
          "IPv4SubnetCIDRBlock": "172.31.32.0/20",
          "DomainNameServers": ["172.31.0.2"],
```

```
          "DomainNameSearchList": ["ec2.internal"],
          "PrivateDNSName": "ip-172-31-42-189.ec2.internal",
          "SubnetGatewayIpv4Address": "172.31.32.1/20"
        }
      ],
      "ContainerARN": "arn:aws:ecs:us-east-1:123456789012:container/MyEmptyCluster/
bfa2636268144d039771334145e490c5/da6cccf7-1178-400c-afdf-7536173ee209",
      "Snapshotter": "overlayfs"
    },
    {
      "DockerId": "bfa2636268144d039771334145e490c5-3681984407",
      "Name": "fargate-app",
      "DockerName": "fargate-app",
      "Image": "public.ecr.aws/docker/library/httpd:latest",
      "ImageID":
 "sha256:8059bdd0058510c03ae4c808de8c4fd2c1f3c1b6d9ea75487f1e5caa5ececa02",
      "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-east-1:123456789012:cluster/
MyEmptyCluster",
        "com.amazonaws.ecs.container-name": "fargate-app",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-1:123456789012:task/
MyEmptyCluster/bfa2636268144d039771334145e490c5",
        "com.amazonaws.ecs.task-definition-family": "sample-fargate",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": { "CPU": 2 },
      "CreatedAt": "2023-07-21T15:45:44.954460255Z",
      "StartedAt": "2023-07-21T15:45:44.954460255Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": ["172.31.42.189"],
          "AttachmentIndex": 0,
          "MACAddress": "0e:98:9f:33:76:d3",
          "IPv4SubnetCIDRBlock": "172.31.32.0/20",
          "DomainNameServers": ["172.31.0.2"],
          "DomainNameSearchList": ["ec2.internal"],
          "PrivateDNSName": "ip-172-31-42-189.ec2.internal",
          "SubnetGatewayIpv4Address": "172.31.32.1/20"
        }
      ],
```

```
      "ContainerARN": "arn:aws:ecs:us-east-1:123456789012:container/MyEmptyCluster/
bfa2636268144d039771334145e490c5/f65b461d-aa09-4acb-a579-9785c0530cbc",
      "Snapshotter": "overlayfs"
    }
  ],
  "LaunchType": "FARGATE",
  "ClockDrift": {
    "ClockErrorBound": 0.446931,
    "ReferenceTimestamp": "2023-07-21T16:09:17Z",
    "ClockSynchronizationStatus": "SYNCHRONIZED"
  },
  "EphemeralStorageMetrics": {
    "Utilized": 261,
    "Reserved": 20496
  }
}
```

작업 통계 응답 예

${ECS_CONTAINER_METADATA_URI_V4}/task/stats 엔드포인트를 쿼리할 때 컨테이너가 속한 태스크에 대한 네트워크 지표가 반환됩니다. 다음은 예시 출력입니다.

```
{
  "3d1f891cded94dc795608466cce8ddcf-464223573": {
    "read": "2020-10-08T21:24:44.938937019Z",
    "preread": "2020-10-08T21:24:34.938633969Z",
    "pids_stats": {},
    "blkio_stats": {
      "io_service_bytes_recursive": [
        {
          "major": 202,
          "minor": 26368,
          "op": "Read",
          "value": 638976
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Write",
          "value": 0
        },
        {
          "major": 202,
```

```
          "minor": 26368,
          "op": "Sync",
          "value": 638976
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Async",
          "value": 0
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Total",
          "value": 638976
        }
      ],
      "io_serviced_recursive": [
        {
          "major": 202,
          "minor": 26368,
          "op": "Read",
          "value": 12
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Write",
          "value": 0
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Sync",
          "value": 12
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Async",
          "value": 0
        },
        {
          "major": 202,
```

```
          "minor": 26368,
          "op": "Total",
          "value": 12
        }
      ],
      "io_queue_recursive": [],
      "io_service_time_recursive": [],
      "io_wait_time_recursive": [],
      "io_merged_recursive": [],
      "io_time_recursive": [],
      "sectors_recursive": []
    },
    "num_procs": 0,
    "storage_stats": {},
    "cpu_stats": {
      "cpu_usage": {
        "total_usage": 1137691504,
        "percpu_usage": [
          696479228,
          441212276,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0
        ],
        "usage_in_kernelmode": 80000000,
        "usage_in_usermode": 810000000
      },
      "system_cpu_usage": 9393210000000,
      "online_cpus": 2,
      "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
      }
```

```
    },
    "precpu_stats": {
      "cpu_usage": {
        "total_usage": 1136624601,
        "percpu_usage": [
          695639662,
          440984939,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0
        ],
        "usage_in_kernelmode": 80000000,
        "usage_in_usermode": 810000000
      },
      "system_cpu_usage": 9373330000000,
      "online_cpus": 2,
      "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
      }
    },
    "memory_stats": {
      "usage": 6504448,
      "max_usage": 8458240,
      "stats": {
        "active_anon": 1675264,
        "active_file": 557056,
        "cache": 651264,
        "dirty": 0,
        "hierarchical_memory_limit": 536870912,
        "hierarchical_memsw_limit": 9223372036854772000,
        "inactive_anon": 0,
        "inactive_file": 3088384,
```

```
        "mapped_file": 430080,
        "pgfault": 11034,
        "pgmajfault": 5,
        "pgpgin": 8436,
        "pgpgout": 7137,
        "rss": 4669440,
        "rss_huge": 0,
        "total_active_anon": 1675264,
        "total_active_file": 557056,
        "total_cache": 651264,
        "total_dirty": 0,
        "total_inactive_anon": 0,
        "total_inactive_file": 3088384,
        "total_mapped_file": 430080,
        "total_pgfault": 11034,
        "total_pgmajfault": 5,
        "total_pgpgin": 8436,
        "total_pgpgout": 7137,
        "total_rss": 4669440,
        "total_rss_huge": 0,
        "total_unevictable": 0,
        "total_writeback": 0,
        "unevictable": 0,
        "writeback": 0
      },
      "limit": 9223372036854772000
    },
    "name": "curltest",
    "id": "3d1f891cded94dc795608466cce8ddcf-464223573",
    "networks": {
      "eth1": {
        "rx_bytes": 2398415937,
        "rx_packets": 1898631,
        "rx_errors": 0,
        "rx_dropped": 0,
        "tx_bytes": 1259037719,
        "tx_packets": 428002,
        "tx_errors": 0,
        "tx_dropped": 0
      }
    },
    "network_rate_stats": {
      "rx_bytes_per_sec": 43.298687872232854,
      "tx_bytes_per_sec": 215.39347269466413
```

```
    }
  }
}
```

Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 엔드포인트 버전 3

> **Important**
>
> 태스크 메타데이터 버전 3 엔드포인트는 더 이상 능동적으로 관리되지 않습니다. 최신 메타데이터 엔드포인트 정보를 가져오려면 태스크 메타데이터 버전 4 엔드포인트를 업데이트하는 것이 좋습니다. 자세한 내용은 the section called “Fargate의 작업에 대한 작업 메타데이터 엔드포인트 버전 4” 섹션을 참조하세요.

Fargate 플랫폼 버전 `1.1.0`부터 `ECS_CONTAINER_METADATA_URI`라는 환경 변수가 태스크의 각 컨테이너에 주입됩니다. 태스크 메타데이터 버전 3 엔드포인트를 쿼리하면 다양한 태스크 메타데이터와 Docker 통계를 태스크에 사용할 수 있습니다.

작업 메타데이터 엔드포인트는 기본적으로 플랫폼 버전 `1.1.0` 이상을 사용하는 Fargate에서 호스팅되는 Amazon ECS 작업에 대해 활성화됩니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.

Fargate의 작업에 대한 작업 메타데이터 엔드포인트 경로

다음 API 엔드포인트를 컨테이너에 사용할 수 있습니다.

`${ECS_CONTAINER_METADATA_URI}`

이 경로는 컨테이너에 대한 메타데이터 JSON을 반환합니다.

`${ECS_CONTAINER_METADATA_URI}/task`

이 경로는 태스크와 연결된 모든 컨테이너의 컨테이너 ID 및 이름 목록을 포함하여 태스크에 대한 메타데이터 JSON을 반환합니다. 이 엔드포인트의 응답에 대한 자세한 내용은 Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 v3 JSON 응답 섹션을 참조하세요.

`${ECS_CONTAINER_METADATA_URI}/stats`

이 경로는 특정 Docker 컨테이너에 대한 Docker 통계 JSON을 반환합니다. 반환된 각 통계에 대한 자세한 내용은 Docker API 설명서의 ContainerStats를 참조하세요.

`${ECS_CONTAINER_METADATA_URI}/task/stats`

이 경로는 태스크와 연결된 모든 컨테이너에 대한 Docker 통계 JSON을 반환합니다. 반환된 각 통계에 대한 자세한 내용은 Docker API 설명서의 ContainerStats를 참조하세요.

Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 v3 JSON 응답

다음 정보가 태스크 메타데이터 엔드포인트(`${ECS_CONTAINER_METADATA_URI}/task`) JSON 응답에서 반환됩니다.

`Cluster`

태스크가 속한 Amazon ECS 클러스터의 Amazon 리소스 이름(ARN) 또는 약어입니다.

`TaskARN`

컨테이너가 속한 태스크의 Amazon 리소스 이름(ARN)입니다.

`Family`

태스크에 대한 Amazon ECS 태스크 정의의 패밀리입니다.

`Revision`

작업에 대한 Amazon ECS 태스크 정의의 개정입니다.

`DesiredStatus`

Amazon ECS의 태스크에 대해 원하는 상태입니다.

`KnownStatus`

Amazon ECS의 태스크에 대해 알려진 상태입니다.

`Limits`

작업 수준에서 지정된 리소스 제한입니다(예: CPU(vCPU로 표시) 및 메모리). 리소스 제한이 정의되지 않은 경우 이 파라미터가 생략됩니다.

`PullStartedAt`

첫 번째 컨테이너 이미지 풀이 시작된 시간에 대한 타임스탬프입니다.

`PullStoppedAt`

마지막 컨테이너 이미지 풀이 완료된 시간에 대한 타임스탬프입니다.

`AvailabilityZone`

태스크가 위치한 가용 영역입니다.

> **Note**
>
> 가용 영역 메타데이터는 플랫폼 버전 1.4 이상(Linux) 또는 1.0.0 이상(Windows)을 사용하는 Fargate 태스크에만 사용할 수 있습니다.

`Containers`

태스크와 연결된 각 컨테이너에 대한 컨테이너 메타데이터의 목록입니다.

`DockerId`

컨테이너의 Docker ID입니다.

`Name`

태스크 정의에 지정된 컨테이너의 이름입니다.

`DockerName`

Docker에 제공된 컨테이너의 이름입니다. 동일한 태스크 정의의 여러 복사본이 단일 인스턴스에서 실행될 때 이름 충돌을 방지하기 위해 Amazon ECS 컨테이너 에이전트는 컨테이너에 고유의 이름을 생성합니다.

`Image`

컨테이너에 대한 이미지입니다.

`ImageID`

이미지 매니페스트의 SHA-256 다이제스트입니다. `repository-url/image@sha256:digest` 형식을 통해 이미지를 가져오는 데 사용할 수 있는 다이제스트입니다.

`Ports`

컨테이너에 대해 노출된 포트입니다. 노출된 포트가 없는 경우 이 파라미터가 생략됩니다.

`Labels`

컨테이너에 적용된 레이블입니다. 적용된 레이블이 없는 경우 이 파라미터가 생략됩니다.

`DesiredStatus`

Amazon ECS의 컨테이너에 대해 원하는 상태입니다.

`KnownStatus`

Amazon ECS의 컨테이너에 대해 알려진 상태입니다.

`ExitCode`

컨테이너에 대한 종료 코드입니다. 컨테이너가 종료되지 않은 경우 이 파라미터가 생략됩니다.

`Limits`

컨테이너 수준에서 지정된 리소스 제한입니다(예: CPU(CPU 단위로 표시) 및 메모리). 리소스 제한이 정의되지 않은 경우 이 파라미터가 생략됩니다.

`CreatedAt`

컨테이너가 생성된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 생성되지 않은 경우 이 파라미터가 생략됩니다.

`StartedAt`

컨테이너가 시작된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 시작되지 않은 경우 이 파라미터가 생략됩니다.

`FinishedAt`

컨테이너가 중지된 시간에 대한 타임스탬프입니다. 컨테이너가 아직 중지되지 않은 경우 이 파라미터가 생략됩니다.

`Type`

컨테이너의 유형입니다. 태스크 정의에서 지정된 컨테이너는 `NORMAL` 유형입니다. Amazon ECS 컨테이너 에이전트가 내부 작업 리소스 프로비저닝에 사용하는 기타 컨테이너 유형은 무시할 수 있습니다.

`Networks`

컨테이너에 대한 네트워크 정보(예: 네트워크 모드 및 IP 주소)입니다. 네트워크 정보가 정의되지 않은 경우 이 파라미터가 생략됩니다.

`ClockDrift`

기준 시간과 시스템 시간의 차이에 대한 정보입니다. 이는 Linux 운영 체제에 적용됩니다. 이 기능은 Amazon Time Sync Service를 사용하여 클럭 정확도를 측정하고 컨테이너에 대한 클록 오류 범

위를 제공합니다. 자세한 내용은 Amazon EC2 Linux용 사용 설명서의 Linux 인스턴스에 대한 시간 설정을 참조하세요.

`ReferenceTime`

클록 정확도의 기초입니다. Amazon ECS는 NTP를 통해 협정 세계시(UTC) 글로벌 표준을 사용합니다(예: `2021-09-07T16:57:44Z`).

`ClockErrorBound`

UTC에 대한 오프셋으로 정의되는 클록 오류 측정값입니다. 이 오류는 기준 시간과 시스템 시간의 차이(밀리초)입니다.

`ClockSynchronizationStatus`

시스템 시간과 기준 시간 간의 가장 최근 동기화 시도가 성공했는지를 나타냅니다.

유효 값은 `SYNCHRONIZED` 및 `NOT_SYNCHRONIZED`입니다.

`ExecutionStoppedAt`

태스크의 `DesiredStatus`가 `STOPPED`로 이동한 시간에 대한 타임스탬프입니다. 이 동작은 필수 컨테이너가 `STOPPED`로 이동할 때 발생합니다.

Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 v3 예제

다음은 단일 컨테이너 태스크에 대한 JSON 응답입니다.

```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
```

```
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
```

```
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```

Amazon ECS 컨테이너 내부 검사

Amazon ECS 컨테이너 에이전트는 에이전트가 실행 중인 컨테이너 인스턴스와 이 인스턴스에서 실행 중인 관련 태스크에 대한 세부 정보를 수집하기 위한 API 태스크를 제공합니다. 컨테이너 인스턴스 안에서 curl 명령을 사용하여 Amazon ECS 컨테이너 에이전트를 쿼리하고(포트 51678) 컨테이너 인스턴스 메타데이터 또는 작업 정보를 반환할 수 있습니다.

> **Important**
>
> 메타데이터를 가져오려면 Amazon ECS에 대한 액세스가 가능한 IAM 역할이 컨테이너 인스턴스에 필요합니다. 자세한 정보는 Amazon ECS 컨테이너 인스턴스 IAM 역할을 참조하세요.

컨테이너 인스턴스 메타데이터를 보려면 SSH를 통해 컨테이너 인스턴스에 로그인하고 다음 명령을 실행합니다. 메타데이터에는 컨테이너 인스턴스 ID, 컨테이너 인스턴스가 등록된 Amazon ECS 클러스터,Amazon ECS 컨테이너 에이전트 버전 정보가 포함되어 있습니다.

```
curl -s http://localhost:51678/v1/metadata | python3 -mjson.tool
```

출력:

```
{
```

```
    "Cluster": "cluster_name",
    "ContainerInstanceArn": "arn:aws:ecs:region:aws_account_id:container-
instance/cluster_name/container_instance_id",
    "Version": "Amazon ECS Agent - v1.30.0 (02ff320c)"
}
```

컨테이너 인스턴스에서 실행 중인 모든 태스크에 대한 정보를 보려면 SSH를 통해 컨테이너 인스턴스에 로그인하고 다음 명령을 실행합니다.

```
curl http://localhost:51678/v1/tasks
```

출력:

```
{
    "Tasks": [
        {
            "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/example5-58ff-46c9-
ae05-543f8example",
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Family": "hello_world",
            "Version": "8",
            "Containers": [
                {
          "DockerId":
 "9581a69a761a557fbfce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1",
          "DockerName": "ecs-hello_world-8-mysql-fcae8ac8f9f1d89d8301",
          "Name": "mysql",
          "CreatedAt": "2023-10-08T20:09:11.44527186Z",
          "StartedAt": "2023-10-08T20:09:11.44527186Z",
          "ImageID":
 "sha256:2ae34abc2ed0a22e280d17e13f9c01aaf725688b09b7a1525d1a2750e2c0d1de"
        },
        {
          "DockerId":
 "bf25c5c5b2d4dba68846c7236e75b6915e1e778d31611e3c6a06831e39814a15",
          "DockerName": "ecs-hello_world-8-wordpress-e8bfddf9b488dff36c00",
          "Name": "wordpress"
        }
    ]
}
    ]
```

```
}
```

컨테이너 인스턴스에서 실행 중인 특정 태스크에 대한 정보를 볼 수 있습니다. 특정 태스크나 컨테이너를 지정하려면 다음 중 하나를 요청에 추가합니다.

- 태스크 ARN(?taskarn=*task_arn*)
- 컨테이너의 Docker ID(?dockerid=*docker_id*)

컨테이너의 Docker ID를 사용하여 작업 정보를 보려면 SSH를 통해 컨테이너 인스턴스에 로그인하고 다음 명령을 실행합니다.

> **Note**
>
> 1.14.2 이전 버전의 Amazon ECS 컨테이너 에이전트는 docker ps를 통해 표시되는 짧은 버전이 아니라 내부 검사 API에 대한 전체 Docker 컨테이너 ID를 요구합니다. 컨테이너 인스턴스에서 docker ps --no-trunc 명령을 실행하면 컨테이너에 대한 전체 Docker ID를 가져올 수 있습니다.

```
curl http://localhost:51678/v1/tasks?dockerid=79c796ed2a7f
```

출력:

```
{
    "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/e01d58a8-151b-40e8-
bc01-22647b9ecfec",
    "Containers": [
        {
            "DockerId":
 "79c796ed2a7f864f485c76f83f3165488097279d296a7c05bd5201a1c69b2920",
            "DockerName": "ecs-nginx-efs-2-nginx-9ac0808dd0afa495f001",
            "Name": "nginx",
            "CreatedAt": "2023-10-08T20:09:11.44527186Z",
            "StartedAt": "2023-10-08T20:09:11.44527186Z",
            "ImageID":
 "sha256:2ae34abc2ed0a22e280d17e13f9c01aaf725688b09b7a1525d1a2750e2c0d1de"
        }
    ],
    "DesiredStatus": "RUNNING",
```

```
    "Family": "nginx-efs",
    "KnownStatus": "RUNNING",
    "Version": "2"
}
```

Runtime Monitoring을 사용하여 무단 동작 식별

Amazon GuardDuty는 AWS 환경 내 계정, 컨테이너, 워크로드 및 데이터를 보호하는 데 도움이 되는 위협 감지 서비스입니다. GuardDuty는 기계 학습(ML) 모델, 이상 및 위협 감지 기능을 통해 다양한 로그 소스와 런타임 활동을 지속적으로 모니터링하여 사용자 환경의 잠재적 보안 위험과 악의적 활동을 식별하고 우선순위를 지정합니다.

GuardDuty의 Runtime Monitoring은 AWS 로그 및 네트워킹 활동을 지속적으로 모니터링하여 악의적 동작 또는 무단 동작을 식별함으로써 Fargate 및 EC2 컨테이너 인스턴스에서 실행되는 워크로드를 보호합니다. Runtime Monitoring은 파일 액세스, 프로세스 실행 및 네트워크 연결과 같은 호스트 내 동작을 분석하는 경량의 완전관리형 GuardDuty 보안 에이전트를 사용합니다. 여기에는 권한 에스컬레이션, 노출된 자격 증명 사용 또는 악의적인 IP 주소 및 도메인과의 통신, 그리고 Amazon EC2 인스턴스 및 컨테이너 워크로드에 존재하는 맬웨어 같은 문제가 포함됩니다. 자세한 내용은 GuardDuty 사용 설명서의 GuardDuty Runtime Monitoring을 참조하세요.

보안 관리자는 GuardDuty의 AWS Organizations에서 단일 또는 여러 계정에 대해 Runtime Monitoring을 활성화합니다. 또한 Fargate를 사용할 때 GuardDuty가 GuardDuty 보안 에이전트를 자동으로 배포할지 여부도 선택합니다. 모든 클러스터는 자동으로 보호되며 GuardDuty는 사용자를 대신하여 보안 에이전트를 관리합니다.

다음과 같은 경우에도 GuardDuty 보안 에이전트를 수동으로 구성할 수 있습니다.

- EC2 컨테이너 인스턴스 사용
- 클러스터 수준에서 Runtime Monitoring을 활성화하려면 세분화된 제어가 필요합니다.

Runtime Monitoring을 사용하려면 보호되는 클러스터를 구성하고 EC2 컨테이너 인스턴스에 GuardDuty 보안 에이전트를 설치 및 관리해야 합니다.

Amazon ECS에서 Runtime Monitoring이 작동하는 방식

Runtime Monitoring은 애플리케이션의 요청, 액세스 확보, 기본 시스템 리소스 소비 방식에 관한 Amazon ECS 워크로드 활동을 모니터링하는 경량 GuardDuty 보안 에이전트를 사용합니다.

Fargate 작업의 경우 GuardDuty 보안 에이전트는 각 작업의 사이드카 컨테이너로 실행됩니다.

EC2 컨테이너 인스턴스의 경우 GuardDuty 보안 에이전트는 인스턴스에서 프로세스로 실행됩니다.

GuardDuty 보안 에이전트는 다음 리소스에서 데이터를 수집하고, GuardDuty로 데이터를 전송하여 처리합니다. 조사 결과는 GuardDuty 콘솔에서 볼 수 있습니다. 또한 집계 및 문제 해결을 위해 다른 AWS 서비스(예: AWS Security Hub) 또는 타사 보안 공급업체에 전송할 수도 있습니다. 조사 결과를 보고 관리하는 방법에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서의 Managing Amazon GuardDuty findings를 참조하세요.

- 다음 Amazon ECS API 직접 호출에서 받은 응답:
  - DescribeClusters

    `--include TAGS` 옵션을 사용할 때 응답 파라미터에 Runtime Monitoring 태그(태그가 설정된 경우)가 포함됩니다.
  - DescribeTasks

    Fargate 시작 유형의 경우 응답 파라미터는 GuardDuty 사이드카 컨테이너를 포함합니다.
  - ListAccountSettings

    응답 파라미터에는 보안 관리자가 설정하는 Runtime Monitoring 계정 설정이 포함됩니다.
- 컨테이너 에이전트 내부 검사 데이터. 자세한 내용은 Amazon ECS 컨테이너 내부 검사 섹션을 참조하세요.
- 시작 유형의 작업 메타데이터 엔드포인트:
  - Amazon ECS 작업 메타데이터 엔드포인트 버전 4
  - Fargate의 작업에 대한 Amazon ECS 작업 메타데이터 엔드포인트 버전 4

고려 사항

Runtime Monitoring을 사용할 때는 다음 사항을 고려합니다.

- Runtime Monitoring에는 관련 비용이 있습니다. 자세한 내용은 Amazon GuardDuty 요금을 참조하세요.
- Amazon ECS Anywhere에서는 Runtime Monitoring이 지원되지 않습니다.
- Runtime Monitoring은 Windows 운영 체제에서 지원되지 않습니다.
- Fargate에서 Amazon ECS Exec을 사용하는 경우 GuardDuty 보안 에이전트가 사이드카 컨테이너로 실행되기 때문에 컨테이너 이름을 지정해야 합니다.
- GuardDuty 보안 에이전트 사이드카 컨테이너에서 Amazon ECS Exec를 사용할 수 없습니다.

- 클러스터 수준에서 Runtime Monitoring을 제어하는 IAM 사용자는 태그 지정을 위한 적절한 IAM 권한을 보유해야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 튜토리얼: 태그를 기반으로 AWS 리소스에 액세스할 수 있는 권한 정의를 참조하세요.
- Fargate 작업에서는 작업 실행 역할을 사용해야 합니다. 이 역할은 사용자를 대신하여 Amazon ECR 프라이빗 리포지토리에 저장된 GuardDuty 보안 에이전트를 검색, 업데이트 및 관리할 권한을 작업에 부여합니다.

리소스 사용률

클러스터에 추가한 태그는 클러스터 태그 할당량 계산에 포함됩니다.

GuardDuty 에이전트 사이드카 컨테이너는 작업 정의 할당량당 컨테이너 수 계산에 포함되지 않습니다.

대부분의 보안 소프트웨어와 마찬가지로 GuardDuty에도 약간의 오버헤드가 있습니다. Fargate 메모리 제한에 대한 자세한 내용은 GuardDuty 사용 설명서의 CPU and memory limits를 참조하세요. Amazon EC2 메모리 제한에 대한 자세한 내용은 CPU and memory limit for GuardDuty agent를 참조하세요.

Amazon ECS Fargate 워크로드에 대한 Runtime Monitoring

EC2 컨테이너 인스턴스를 사용하는 경우 Runtime Monitoring을 수동으로 구성해야 합니다. 자세한 내용은 Amazon ECS의 EC2 워크로드에 대한 Runtime Monitoring 섹션을 참조하세요.

GuardDuty가 컨테이너 인스턴스의 보안 에이전트를 관리하도록 할 수 있습니다. 이 옵션은 Fargate에서만 사용할 수 있습니다. 이 옵션(GuardDuty 에이전트 관리)은 GuardDuty에서 사용할 수 있음

GuardDuty 에이전트 관리를 사용하는 경우 GuardDuty는 다음 작업을 수행합니다.

- 클러스터를 호스팅하는 각 VPC에 대해 GuardDuty용 VPC 엔드포인트를 생성합니다.
- 검색 후 최신 GuardDuty 보안 에이전트를 모든 새로운 독립 실행형 Fargate 작업 및 새로운 서비스 배포에서 사이드카 컨테이너로 설치합니다.

  서비스를 처음 시작하거나 새 배포 강제 적용 옵션으로 기존 서비스를 업데이트할 때 새 서비스 배포가 수행됩니다.

Amazon ECS에 대한 Runtime Monitoring 켜기

모든 Fargate 클러스터의 보안 에이전트를 자동으로 관리하도록 GuardDuty를 구성할 수 있습니다.

사전 조건

다음은 Runtime Monitoring 사용을 위한 필수 조건입니다.

- Fargate 플랫폼 버전은 Linux의경우 `1.4.0` 이상이어야 합니다.
- Amazon ECS에 대한 IAM 역할 및 권한:
  - Fargate 작업에서는 작업 실행 역할을 사용해야 합니다. 이 역할은 사용자를 대신하여 GuardDuty 보안 에이전트를 검색, 업데이트 및 관리할 권한을 작업에 부여합니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할을 참조하세요.
  - 사전 정의된 태그를 사용하여 클러스터의 Runtime Monitoring을 제어합니다. 액세스 정책이 태그를 기반으로 액세스를 제한하는 경우 IAM 사용자에게 클러스터에 태그를 지정할 수 있는 명시적 권한을 부여해야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 튜토리얼: 태그를 기반으로 AWS 리소스에 액세스할 수 있는 권한 정의를 참조하세요.
- Amazon ECR 리포지토리에 연결:

  GuardDuty 보안 에이전트는 Amazon ECR 리포지토리에 저장됩니다. 각 독립 실행형 작업 및 서비스 작업에는 리포지토리에 대한 액세스 권한이 있어야 합니다. 다음 옵션 중 하나를 사용할 수 있습니다.
  - 퍼블릭 서브넷에 있는 작업의 경우 작업에 퍼블릭 IP 주소를 사용하거나 작업이 실행되는 서브넷에서 Amazon ECR용 VPC 엔드포인트를 생성할 수 있습니다. 자세한 정보는 Amazon Elastic Container Registry 사용 설명서의 Amazon ECR 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.
  - 프라이빗 서브넷에 있는 작업의 경우 Network Address Translation(NAT) 게이트웨이를 사용하거나 작업이 실행되는 서브넷에서 Amazon ECR용 VPC 엔드포인트를 생성할 수 있습니다.

    자세한 내용은 프라이빗 서브넷 및 NAT 게이트웨이를 참조하세요.
- GuardDuty에 대한 `AWSServiceRoleForAmazonGuardDuty` 역할이 필요합니다. 자세한 내용은 Amazon GuardDuty 사용 설명서의 Service-linked role permissions for GuardDuty를 참조하세요.
- Runtime Monitoring으로 보호하려는 모든 파일에는 루트 사용자가 액세스할 수 있어야 합니다. 파일의 권한을 수동으로 변경한 경우 해당 권한을 `755`로 설정해야 합니다.

다음은 EC2 컨테이너 인스턴스에서 Runtime Monitoring을 사용하기 위한 필수 조건입니다.

- Amazon ECS-AMI의 버전 `20230929` 이상을 사용해야 합니다.
- 컨테이너 인스턴스에서 버전 `1.77` 이상으로 Amazon ECS 에이전트를 실행해야 합니다.

- 커널 버전 5.10 이상을 사용해야 합니다.
- 지원되는 Linux 운영 체제 및 아키텍처에 대한 자세한 내용은 GuardDuty Runtime Monitoring에서 지원하는 운영 모델 및 워크로드는 무엇인가요?를 참조하세요.
- System Manager를 사용하여 컨테이너 인스턴스를 관리할 수 있습니다. 자세한 내용은 AWS Systems Manager Session Manager 사용 설명서의 EC2 인스턴스용 Systems Manager 설정을 참조하세요.

절차

GuardDuty에서 Runtime Monitoring을 활성화합니다. 기능을 활성화하는 방법에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서의 Enabling Runtime Monitoring을 참조하세요.

기존 Amazon ECS Fargate 태스크에 Runtime Monitoring 추가

Runtime Monitoring을 켜면 클러스터의 모든 새 독립 실행형 작업 및 새 서비스 배포가 자동으로 보호됩니다. 불변성 제약 조건을 유지하기 위해 기존 작업은 영향을 받지 않습니다.

사전 조건

1. Runtime Monitoring을 켭니다. 자세한 내용은 Amazon ECS에 대한 Runtime Monitoring 켜기 섹션을 참조하세요.
2. Fargate 작업에서는 작업 실행 역할을 사용해야 합니다. 이 역할은 사용자를 대신하여 GuardDuty 보안 에이전트를 검색, 업데이트 및 관리할 권한을 작업에 부여합니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할을 참조하세요.

절차

- 작업을 즉시 보호하려면 다음 작업 중 하나를 수행해야 합니다.
  - 독립 실행형 작업의 경우 작업을 중지한 다음, 시작합니다. 자세한 내용은 Amazon ECS 태스크 중지 및 애플리케이션을 Amazon ECS 태스크로 실행 섹션을 참조하세요.
  - 서비스의 일부인 작업의 경우 '새 배포 강제 적용' 옵션을 사용하여 서비스를 업데이트합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.

Amazon ECS 클러스터에서 Runtime Monitoring 제거

특정 클러스터(예: 테스트에 사용하는 클러스터)를 보호에서 제외할 수 있습니다. 그러면 GuardDuty는 클러스터의 리소스에 대해 다음 작업을 수행합니다.

- 더 이상 GuardDuty 보안 에이전트를 새로운 독립 실행형 Fargate 작업이나 새로운 서비스 배포에 배포하지 않아도 됩니다.

  불변성 제약 조건을 유지하기 위해 Runtime Monitoring이 활성화된 기존 작업 및 배포는 영향을 받지 않습니다.
- 요금 청구를 중지하고 작업에 대한 런타임 이벤트를 더 이상 수락하지 않습니다.

절차

클러스터에서 Runtime Monitoring을 제거하려면 다음 단계를 수행하세요.

1. Amazon ECS 콘솔 또는 AWS CLI를 사용하여 클러스터에서 `GuardDutyManaged` 태그 키를 `false`로 설정합니다. 자세한 내용은 클러스터 업데이트 또는 CLI 또는 API를 사용한 태그 작업을 참조하세요. 태그에 대해 다음 값을 사용합니다.

   > **Note**
   >
   > 키와 값은 대소문자를 구분하며 문자열과 정확히 일치해야 합니다.

   키 = `GuardDutyManaged`, 값 = `false`
2. 클러스터의 GuardDuty VPC 엔드포인트를 삭제합니다. VPC 엔드포인트를 삭제하는 방법에 대한 자세한 내용은 AWS PrivateLink 사용 설명서의 Delete an interface endpoint를 참조하세요.

계정에서 Amazon ECS에 대한 Runtime Monitoring 제거

Runtime Monitoring을 더 이상 사용하지 않으려면 GuardDuty에서 해당 기능을 비활성화합니다. 기능을 비활성화하는 방법에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서의 Enabling Runtime Monitoring을 참조하세요.

GuardDuty는 다음 작업을 수행합니다.

- 클러스터를 호스팅하는 각 VPC에 대해 GuardDuty용 VPC 엔드포인트를 삭제합니다.
- 더 이상 GuardDuty 보안 에이전트를 새로운 독립 실행형 Fargate 작업이나 새로운 서비스 배포에 배포하지 않아도 됩니다.

  불변성 제약 조건을 유지하기 위해 기존 작업 및 배포는 중지, 복제 또는 규모 조정될 때까지 영향을 받지 않습니다.

- 요금 청구를 중지하고 작업에 대한 런타임 이벤트를 더 이상 수락하지 않습니다.

Amazon ECS의 EC2 워크로드에 대한 Runtime Monitoring

용량에 맞는 EC2 인스턴스를 사용하거나 Fargate의 클러스터 수준에서 Runtime Monitoring에 대한 세분화된 제어가 필요한 경우 이 옵션을 사용합니다.

사전 정의된 태그를 추가하여 Runtime Monitoring에 대한 클러스터를 프로비저닝합니다.

EC2 컨테이너 인스턴스의 경우 GuardDuty 보안 에이전트를 다운로드, 설치 및 관리합니다.

Fargate의 경우 GuardDuty에서 사용자를 대신하여 보안 에이전트를 관리합니다.

Amazon ECS에 대한 Runtime Monitoring 켜기

EC2 인스턴스가 있는 클러스터에 대해 또는 Fargate의 클러스터 수준에서 Runtime Monitoring에 대한 세분화된 제어가 필요한 경우 Runtime Monitoring을 켤 수 있습니다.

다음은 Runtime Monitoring 사용을 위한 필수 조건입니다.

- Fargate 플랫폼 버전은 Linux의경우 `1.4.0` 이상이어야 합니다.
- Amazon ECS에 대한 IAM 역할 및 권한:
  - Fargate 작업에서는 작업 실행 역할을 사용해야 합니다. 이 역할은 사용자를 대신하여 GuardDuty 보안 에이전트를 검색, 업데이트 및 관리할 권한을 작업에 부여합니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할을 참조하세요.
  - 사전 정의된 태그를 사용하여 클러스터의 Runtime Monitoring을 제어합니다. 액세스 정책이 태그를 기반으로 액세스를 제한하는 경우 IAM 사용자에게 클러스터에 태그를 지정할 수 있는 명시적 권한을 부여해야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 튜토리얼: 태그를 기반으로 AWS 리소스에 액세스할 수 있는 권한 정의를 참조하세요.
- Amazon ECR 리포지토리에 연결:

  GuardDuty 보안 에이전트는 Amazon ECR 리포지토리에 저장됩니다. 각 독립 실행형 작업 및 서비스 작업에는 리포지토리에 대한 액세스 권한이 있어야 합니다. 다음 옵션 중 하나를 사용할 수 있습니다.
  - 퍼블릭 서브넷에 있는 작업의 경우 작업에 퍼블릭 IP 주소를 사용하거나 작업이 실행되는 서브넷에서 Amazon ECR용 VPC 엔드포인트를 생성할 수 있습니다. 자세한 정보는 Amazon Elastic Container Registry 사용 설명서의 Amazon ECR 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

- 프라이빗 서브넷에 있는 작업의 경우 Network Address Translation(NAT) 게이트웨이를 사용하거나 작업이 실행되는 서브넷에서 Amazon ECR용 VPC 엔드포인트를 생성할 수 있습니다.

  자세한 내용은 프라이빗 서브넷 및 NAT 게이트웨이를 참조하세요.

- GuardDuty에 대한 `AWSServiceRoleForAmazonGuardDuty` 역할이 필요합니다. 자세한 내용은 Amazon GuardDuty 사용 설명서의 Service-linked role permissions for GuardDuty를 참조하세요.
- Runtime Monitoring으로 보호하려는 모든 파일에는 루트 사용자가 액세스할 수 있어야 합니다. 파일의 권한을 수동으로 변경한 경우 해당 권한을 `755`로 설정해야 합니다.

다음은 EC2 컨테이너 인스턴스에서 Runtime Monitoring을 사용하기 위한 필수 조건입니다.

- Amazon ECS-AMI의 버전 `20230929` 이상을 사용해야 합니다.
- 컨테이너 인스턴스에서 버전 `1.77` 이상으로 Amazon ECS 에이전트를 실행해야 합니다.
- 커널 버전 `5.10` 이상을 사용해야 합니다.
- 지원되는 Linux 운영 체제 및 아키텍처에 대한 자세한 내용은 GuardDuty Runtime Monitoring에서 지원하는 운영 모델 및 워크로드는 무엇인가요?를 참조하세요.
- System Manager를 사용하여 컨테이너 인스턴스를 관리할 수 있습니다. 자세한 내용은 AWS Systems Manager Session Manager 사용 설명서의 EC2 인스턴스용 Systems Manager 설정을 참조하세요.

GuardDuty에서 Runtime Monitoring을 켭니다. 기능을 활성화하는 방법에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서의 Enabling Runtime Monitoring을 참조하세요.

Amazon ECS 클러스터에 Runtime Monitoring 추가

클러스터에 대한 Runtime Monitoring을 구성하고 EC2 컨테이너 인스턴스에 GuardDuty 보안 에이전트를 설치합니다.

사전 조건

1. Runtime Monitoring을 켭니다. 자세한 내용은 Amazon ECS에 대한 Runtime Monitoring 켜기 섹션을 참조하세요.
2. 사전 정의된 태그를 사용하여 클러스터의 Runtime Monitoring을 제어합니다. 액세스 정책이 태그를 기반으로 액세스를 제한하는 경우 IAM 사용자에게 클러스터에 태그를 지정할 수 있는 명시적 권한을 부여해야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 튜토리얼: 태그를 기반으로 AWS 리소스에 액세스할 수 있는 권한 정의를 참조하세요.

절차

클러스터에 Runtime Monitoring을 추가하려면 다음 작업을 수행합니다.

1. 각 클러스터 VPC에 대해 GuardDuty용 VPC 엔드포인트를 생성합니다. 자세한 내용은 GuardDuty 사용 설명서의 Creating Amazon VPC endpoint manually를 참조하세요.
2. EC2 컨테이너 인스턴스를 구성합니다.

   a. 클러스터의 EC2 컨테이너 인스턴스에서 Amazon ECS 에이전트를 버전 `1.77` 이상으로 업데이트합니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 업데이트을 참조하세요.

   b. 클러스터의 EC2 컨테이너 인스턴스에 GuardDuty 보안 에이전트를 설치합니다. 자세한 내용은 GuardDuty 사용 설명서의 Managing the security agent on an Amazon EC2 instance manually를 참조하세요.

      GuardDuty 보안 에이전트는 EC2 컨테이너 인스턴스에서 프로세스로 실행되므로 모든 신규 및 기존 작업과 배포가 즉시 보호됩니다.
3. Amazon ECS 콘솔 또는 AWS CLI를 사용하여 클러스터에서 `GuardDutyManaged` 태그 키를 `true`로 설정합니다. 자세한 내용은 클러스터 업데이트 또는 CLI 또는 API를 사용한 태그 작업을 참조하세요. 태그에 대해 다음 값을 사용합니다.

   > **Note**
   >
   > 키와 값은 대소문자를 구분하며 문자열과 정확히 일치해야 합니다.

   키 = `GuardDutyManaged`, 값 = `true`

기존 Amazon ECS 태스크에 Runtime Monitoring 추가

Runtime Monitoring을 켜면 클러스터의 모든 새 독립 실행형 작업 및 새 서비스 배포가 자동으로 보호됩니다. 불변성 제약 조건을 유지하기 위해 기존 작업은 영향을 받지 않습니다.

사전 조건

- Runtime Monitoring을 켭니다. 자세한 내용은 Amazon ECS에 대한 Runtime Monitoring 켜기 섹션을 참조하세요.

절차

- 작업을 즉시 보호하려면 다음 작업 중 하나를 수행해야 합니다.
  - 독립 실행형 작업의 경우 작업을 중지한 다음, 시작합니다. 자세한 내용은 Amazon ECS 태스크 중지 및 애플리케이션을 Amazon ECS 태스크로 실행(을)를 참조하세요.
  - 서비스의 일부인 작업의 경우 '새 배포 강제 적용' 옵션을 사용하여 서비스를 업데이트합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 섹션을 참조하세요.

Amazon ECS 클러스터에서 Runtime Monitoring 제거

클러스터에서 Runtime Monitoring을 제거할 수 있습니다. 이 경우 GuardDuty는 클러스터에서 모든 리소스 모니터링을 중지합니다.

클러스터에서 Runtime Monitoring을 제거하려면 다음을 수행하세요.

1. Amazon ECS 콘솔 또는 AWS CLI를 사용하여 클러스터에서 `GuardDutyManaged` 태그 키를 `false`로 설정합니다. 자세한 내용은 클러스터 업데이트 또는 CLI 또는 API를 사용한 태그 작업을 참조하세요.

   > **Note**
   >
   > 키와 값은 대소문자를 구분하며 문자열과 정확히 일치해야 합니다.

   키 = `GuardDutyManaged`, 값 = `false`
2. 클러스터의 EC2 컨테이너 인스턴스에서 GuardDuty 보안 에이전트를 제거합니다.

   자세한 내용은 GuardDuty 사용 설명서의 Uninstalling the security agent manually를 참조하세요.
3. 각 클러스터 VPC의 GuardDuty VPC 엔드포인트를 삭제합니다. VPC 엔드포인트를 삭제하는 방법에 대한 자세한 내용은 AWS PrivateLink 사용 설명서의 Delete an interface endpoint를 참조하세요.

Amazon ECS 컨테이너 인스턴스에서 GuardDuty 보안 에이전트 업데이트

EC2 컨테이너 인스턴스의 GuardDuty 보안 에이전트를 업데이트하는 방법에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서의 Updating GuardDuty security agent를 참조하세요.

계정에서 Amazon ECS에 대한 Runtime Monitoring 제거

Runtime Monitoring을 더 이상 사용하지 않으려면 GuardDuty에서 해당 기능을 비활성화합니다. 기능을 비활성화하는 방법에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서의 Enabling Runtime Monitoring을 참조하세요.

모든 클러스터에서 Runtime Monitoring을 제거합니다. 자세한 내용은 Amazon ECS 클러스터에서 Runtime Monitoring 제거 섹션을 참조하세요.

Runtime Monitoring 문제 해결

Runtime Monitoring이 활성화되어 작업 및 컨테이너에서 실행 중인지 확인하거나 문제를 해결해야 할 수 있습니다.

주제

- 내 계정에서 Runtime Monitoring이 활성화되어 있는지 어떻게 알 수 있나요?
- 클러스터에서 Runtime Monitoring이 활성 상태인지 어떻게 알 수 있나요?
- GuardDuty 보안 에이전트가 Fargate 작업에서 실행 중인지 어떻게 알 수 있나요?
- GuardDuty 보안 에이전트가 EC2 컨테이너 인스턴스에서 실행 중인지 어떻게 알 수 있나요?
- 클러스터에서 실행 중인 작업에 대한 작업 실행 역할이 없으면 어떻게 되나요?
- Runtime Monitoring을 위해 클러스터에 태그를 지정할 수 있는 올바른 권한이 있는지 어떻게 알 수 있나요?
- Amazon ECR에 연결되어 있지 않으면 어떻게 되나요?
- Runtime Monitoring을 활성화한 후 Fargate 작업의 메모리 부족 오류를 해결하려면 어떻게 해야 하나요?

내 계정에서 Runtime Monitoring이 활성화되어 있는지 어떻게 알 수 있나요?

Amazon ECS 콘솔에서 정보는 계정 설정 페이지에 있습니다.

`list-account-settings`를 `effective-settings` 옵션과 함께 실행할 수도 있습니다.

```
aws ecs list-account-settings --effective-settings
```

출력

이름이 `guardDutyActivate`로 설정되고 값이 `on`으로 설정된 설정은 계정이 구성되었음을 나타냅니다. GuardDuty 관리자에게 문의하여 관리가 자동인지, 수동인지 확인해야 합니다.

```
{
    "setting": {
        "name": "guardDutyActivate",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:root",
        "type": "aws-managed"
    }
}
```

클러스터에서 Runtime Monitoring이 활성 상태인지 어떻게 알 수 있나요?

GuardDuty 콘솔에서 적용 범위 통계를 검토할 수 있습니다. 본인 계정 또는 멤버 계정과 연결된 Amazon ECS 리소스에 대한 정보는 선택한 AWS 리전의 모든 클러스터에 대한 정상적인 클러스터의 비율입니다. 여기에는 Fargate 및 EC2 시작 유형을 사용하는 클러스터에 대한 적용 범위가 포함됩니다. 자세한 내용은 Amazon GuardDuty 사용 설명서의 적용 범위 통계 검토를 참조하세요.

GuardDuty 보안 에이전트가 Fargate 작업에서 실행 중인지 어떻게 알 수 있나요?

GuardDuty 보안 에이전트는 Fargate 작업의 사이드카 컨테이너로 실행됩니다.

Amazon ECS 콘솔에서 사이드카는 작업 세부 정보 페이지의 컨테이너 아래에 표시됩니다.

`describe-tasks`를 실행하고 이름이 `aws-gd-agent`로 설정되고 LastStatus가 `RUNNING`으로 설정된 컨테이너를 찾을 수 있습니다.

다음 예제에서는 작업 `aws:ecs:us-east-1:123456789012:task/0b69d5c0-d655-4695-98cd-5d2d5EXAMPLE`에 대한 기본 클러스터의 출력을 보여줍니다.

```
aws ecs describe-tasks --cluster default --tasks aws:ecs:us-
east-1:123456789012:task/0b69d5c0-d655-4695-98cd-5d2d5EXAMPLE
```

출력

이름이 `gd-agent`인 컨테이너가 `RUNNING` 상태입니다.

```
"containers": [
      {
        "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/4df26bb4-
f057-467b-a079-96167EXAMPLE",
        "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/0b69d5c0-
d655-4695-98cd-5d2d5EXAMPLE",
```

```
        "lastStatus": "RUNNING",
        "healthStatus": "UNKNOWN",
        "memory": "string",
        "name": "aws-gd-agent"
      }
    ]
```

GuardDuty 보안 에이전트가 EC2 컨테이너 인스턴스에서 실행 중인지 어떻게 알 수 있나요?

다음 명령을 실행하여 상태를 봅니다.

```
sudo systemctl status amazon-guardduty-agent
```

로그 파일은 다음 위치에 있습니다.

```
/var/log/amzn-guardduty-agent
```

클러스터에서 실행 중인 작업에 대한 작업 실행 역할이 없으면 어떻게 되나요?

Fargate 작업의 경우 GuardDuty 보안 에이전트 사이드카 컨테이너 없이 작업이 시작됩니다. GuardDuty 대시보드는 적용 범위 통계 대시보드에서 작업에 보호 기능이 누락되었음을 표시합니다.

Runtime Monitoring을 위해 클러스터에 태그를 지정할 수 있는 올바른 권한이 있는지 어떻게 알 수 있나요?

클러스터에 태그를 지정하려면 `CreateCluster` 및 `ecs:TagResource` 모두에 대한 `UpdateCluster` 작업이 있어야 합니다.

다음은 예제 정책의 코드 조각입니다.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
         "ecs:TagResource"
      ],
      "Resource": "*",
      "Condition": {
         "StringEquals": {
```

```
            "ecs:CreateAction" : "CreateCluster",
            "ecs:CreateAction" : "UpdateCluster",
          }
        }
    }
  ]
}
```

Amazon ECR에 연결되어 있지 않으면 어떻게 되나요?

Fargate 작업의 경우 GuardDuty 보안 에이전트 사이드카 컨테이너 없이 작업이 시작됩니다. GuardDuty 대시보드는 적용 범위 통계 대시보드에서 작업에 보호 기능이 누락되었음을 표시합니다.

Runtime Monitoring을 활성화한 후 Fargate 작업의 메모리 부족 오류를 해결하려면 어떻게 해야 하나요?

GuardDuty 보안 에이전트는 경량 프로세스입니다. 그러나 이 프로세스는 워크로드 크기에 따라 여전히 리소스를 소비합니다. Amazon CloudWatch Container Insights와 같은 컨테이너 리소스 추적 도구를 사용하여 클러스터에서 GuardDuty 배포를 스테이징하는 것이 좋습니다. 이러한 도구를 사용하면 애플리케이션에 대한 GuardDuty 보안 에이전트의 소비 프로파일을 검색할 수 있습니다. 그런 다음, 필요한 경우 Fargate 작업 크기를 조정하여 잠재적인 메모리 부족 조건을 방지할 수 있습니다.

ECS Exec를 사용하여 Amazon ECS 컨테이너 모니터링

Amazon ECS Exec을 사용하면 먼저 호스트 컨테이너 운영 체제와 상호 작용하거나 인바운드 포트를 열거나 SSH 키를 관리할 필요 없이 컨테이너와 직접 상호 작용할 수 있습니다. ECS Exec을 사용하여 명령을 실행하거나 Amazon EC2 인스턴스 또는 AWS Fargate에서 실행하는 컨테이너에 셸을 가져올 수 있습니다. 이를 통해 진단 정보를 쉽게 수집하고 오류를 신속하게 해결할 수 있습니다. 예를 들어 개발 컨텍스트에서 ECS Exec을 사용하여 컨테이너의 다양한 프로세스와 쉽게 상호 작용하고 애플리케이션 문제를 해결할 수 있습니다. 또한 프로덕션 시나리오에서 이를 사용하여 컨테이너에 중단 없이 액세스하고 문제를 디버깅할 수 있습니다.

Amazon ECS API, AWS Command Line Interface(AWS CLI), AWS SDK 또는 AWS Copilot CLI에서 ECS Exec을 사용하여 실행 중인 Linux 또는 Windows 컨테이너에서 명령을 실행할 수 있습니다. ECS Exec 사용 및 AWS Copilot CLI를 사용하는 비디오 연습에 대한 자세한 내용은 Copilot GitHub 설명서를 참조하세요.

ECS Exec을 사용하여 더 엄격한 액세스 제어 정책을 유지할 수도 있습니다. 이 기능을 선택적으로 설정하여 명령을 실행할 수 있는 사용자와 해당 명령을 실행할 수 있는 태스크를 제어할 수 있습니다. 각

명령의 로그와 그 출력을 통해 ECS Exec을 사용하여 실행된 태스크를 보고 CloudTrail을 사용하여 컨테이너에 액세스한 사용자를 감사할 수 있습니다.

고려 사항

이 항목에서는 ECS Exec 사용과 관련된 다음 측면을 익혀야 합니다.

- ECS Exec은 현재 AWS Management Console 사용을 지원하지 않습니다.

  클러스터 세부 정보 페이지에는 로컬 클라이언트와 컨테이너 간의 데이터를 암호화하는 데 사용되는 로깅 및 AWS KMS 고객 관리형 키가 표시됩니다.
- Systems Manager에서 지원하지 않는 운영 체제에서 실행하는 경우 ECS Exec이 예상대로 작동하지 않을 수 있습니다. 지원되는 운영 체제에 대한 자세한 내용은 AWS Systems Manager 사용 설명서의 운영 체제 유형을 참조하세요.
- ECS Exec은 다음 인프라에서 실행되는 작업에 지원됩니다.
  - Bottlerocket을 포함한 Amazon ECS 최적화 AMI에 있는 Amazon EC2의 Linux 컨테이너
  - 외부 인스턴스(Amazon ECS Anywhere)의 Linux 및 Windows 컨테이너
  - AWS Fargate의 Linux 및 Windows 컨테이너
  - 다음 Windows Amazon ECS 최적화 AMI(컨테이너 에이전트 버전 `1.56` 이상)에 있는 Amazon EC2의 Windows 컨테이너:
    - Amazon ECS 최적화 Windows Server 2022 Full AMI
    - Amazon ECS 최적화 Windows Server 2022 Core AMI
    - Amazon ECS 최적화 Windows Server 2019 Full AMI
    - Amazon ECS 최적화 Windows Server 2019 Core AMI
    - Amazon ECS 최적화 Windows Server 20H2 Core AMI
- 태스크에 HTTP 프록시를 구성한 경우 EC2 인스턴스 메타데이터 및 IAM 역할 트래픽에 대한 프록시를 우회하도록 `NO_PROXY` 환경 변수를 `"NO_PROXY=169.254.169.254,169.254.170.2"`로 설정합니다. `NO_PROXY` 환경 변수를 구성하지 않으면 컨테이너 내의 메타데이터 엔드포인트에서 인스턴스 메타데이터 또는 IAM 역할 자격 증명을 검색할 때 오류가 발생할 수 있습니다. `NO_PROXY` 환경 변수를 권장 사항으로 설정하면 `169.254.169.254 and 169.254.170.2`에 대한 요청이 HTTP 프록시를 통과하지 않도록 메타데이터 및 IAM 트래픽이 필터링됩니다.
- ECS Exec 및 Amazon VPC
  - Amazon ECS와 인터페이스 Amazon VPC 엔드포인트를 사용하는 경우 Systems Manager Session Manager(`ssmmessages`)용 인터페이스 Amazon VPC 엔드포인트를 생성해야 합니다.

Systems Manager VPC 엔드포인트에 대한 자세한 정보는 AWS Systems Manager 사용 설명서의 AWS PrivateLink를 사용하여 Session Manager에 대한 VPC 엔드포인트 설정을 참조하세요.

  - Amazon ECS와 함께 인터페이스 Amazon VPC 엔드포인트를 사용하고 있으며 암호화에 AWS KMS key를 사용하는 경우, AWS KMS key에 대한 인터페이스 Amazon VPC 엔드포인트를 생성해야 합니다. 자세한 내용은 AWS Key Management Service 개발자 안내서에서 VPC 엔드포인트 구성을 통해 AWS KMS key에 연결을 참조하세요.
  - Amazon EC2 인스턴스에서 실행되는 작업이 있는 경우 `awsvpc` 네트워킹 모드를 사용합니다. 인터넷에 액세스할 수 없는 경우(예: NAT 게이트웨이를 사용하도록 구성되지 않음) Systems Manager Session Manager(`ssmmessages`)에 대한 인터페이스 Amazon VPC 엔드포인트를 생성해야 합니다. `awsvpc` 네트워크 모드 고려 사항에 대한 자세한 내용은 Considerations를 참조하세요. Systems Manager VPC 엔드포인트에 대한 자세한 정보는 AWS Systems Manager 사용 설명서의 AWS PrivateLink를 사용하여 Session Manager에 대한 VPC 엔드포인트 설정을 참조하세요.
- ECS Exec 및 SSM
  - 사용자가 ECS Exec을 사용하여 컨테이너에서 명령을 실행하면 이러한 명령은 `root` 사용자로 실행됩니다. SSM 에이전트와 하위 프로세스는 컨테이너에 대한 사용자 ID를 지정하는 경우에도 루트로 실행됩니다.
  - SSM 에이전트는 필요한 디렉터리 및 파일을 생성하기 위해 컨테이너 파일 시스템을 기록할 수 있어야 합니다. 따라서 `readonlyRootFilesystem` 태스크 정의 파라미터 또는 다른 메서드를 사용해 루트 파일 시스템을 읽기 전용으로 만드는 것은 지원되지 않습니다.
  - `execute-command` 작업 외부에서 SSM 세션을 시작하는 것은 가능하지만, 세션이 로깅되지 않고 세션 제한에 대해 계산됩니다. IAM 정책을 사용하여 `ssm:start-session` 태스크를 거부함으로써 이 액세스를 제한하는 것이 좋습니다. 자세한 내용은 세션 시작 작업에 대한 액세스 제한 섹션을 참조하세요.
- 다음 기능은 사이드카 컨테이너로 실행됩니다. 따라서 명령을 실행할 컨테이너 이름을 지정해야 합니다.
  - 런타임 모니터링
  - Service Connect
- 사용자는 컨테이너 컨텍스트 내에서 사용할 수 있는 모든 명령을 실행할 수 있습니다. 컨테이너의 주 프로세스 종료, 명령 에이전트 종료 및 종속성 삭제와 같은 태스크를 수행하면 분리된 좀비 프로세스가 발생할 수 있습니다. 좀비 프로세스를 정리하려면 `initProcessEnabled` 플래그를 태스크 정의에 추가하는 것이 좋습니다.
- ECS Exec는 약간의 CPU와 메모리를 사용합니다. 태스크 정의에서 CPU 및 메모리 리소스 할당을 지정할 때 이를 수용하고 싶을 것입니다.

- AWS CLI 버전 `1.22.3` 이상 또는 AWS CLI 버전 `2.3.6` 이상을 사용해야 합니다. AWS CLI 업데이트 방법에 대한 자세한 정보는 AWS Command Line Interface 버전 2 사용 설명서의 최신 버전의 AWS CLI 설치 또는 업데이트를 참조하세요.
- 프로세스 ID(PID) 네임스페이스당 ECS Exec 세션을 하나만 사용할 수 있습니다. 작업에서 PID 네임스페이스를 공유하는 경우 ECS Exec 세션을 하나의 컨테이너에서만 시작할 수 있습니다.
- ECS Exec 세션의 유휴 제한 시간은 20분입니다. 이 값은 변경할 수 없습니다.
- 기존 작업에 대해 ECS Exec을 켤 수 없습니다. 새 작업에 대해서만 켤 수 있습니다.
- `run-task`를 사용하여 비동기식 배치로 관리형 규모 조정을 사용하는 클러스터에서 작업을 시작하는 경우(인스턴스 없이 작업 시작) ECS Exec를 사용할 수 없습니다.
- Microsoft Nano Server 컨테이너에서 ECS Exec를 실행할 수 없습니다.

사전 조건

ECS Exec를 사용하기 전에 다음 작업을 완료했는지 확인합니다.

- AWS CLI를 설치하고 구성합니다. 자세한 내용은 AWS CLI 시작하기를 참조하세요.
- AWS CLI에 대한 Session Manager 플러그인을 설치합니다. 자세한 내용은 AWS CLI에 대한 Session Manager 플러그인 설치를 참조하세요.
- ECS Exec에 대한 적절한 권한이 있는 태스크 역할을 사용해야 합니다. 자세한 내용은 Task IAM role을 참조하세요.
- ECS Exec은 작업이 Amazon EC2 또는 AWS Fargate에 호스팅되는지 여부에 따라 버전 요구 사항이 있습니다.
  - Amazon EC2를 사용하는 경우 2021년 1월 20일 이후에 출시된 Amazon ECS 최적화 AMI를 에이전트 버전 1.50.2 이상으로 사용해야 합니다. 자세한 내용은 Amazon ECS 최적화 AMI를 참조하세요.
  - AWS Fargate을(를) 사용하는 경우 플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0`(Windows)을(를) 사용해야 합니다. 자세한 내용은 AWS Fargate 플랫폼 버전을 참조하세요.

아키텍처

ECS Exec은 AWS Systems Manager(SSM) Session Manager를 사용하여 실행 중인 컨테이너와의 연결을 설정하고 AWS Identity and Access Management(IAM) 정책을 사용하여 실행 중인 컨테이너에서 실행 중인 명령에 대한 액세스를 제어할 수 있습니다. 이는 필요한 SSM 에이전트 바이너리를 컨테이너에 바인드 탑재하여 수행할 수 있습니다. Amazon ECS 또는 AWS Fargate 에이전트는 애플리케이

션 코드와 함께 컨테이너 내부의 SSM 핵심 에이전트를 시작합니다. 자세한 내용은 Systems Manager Session Manager를 참조하세요.

AWS CloudTrail의 `ExecuteCommand` 이벤트를 사용하여 컨테이너에 액세스한 사용자를 감사할 수 있으며 각 명령(및 출력)을 Amazon S3 또는 Amazon CloudWatch Logs에 로깅할 수 있습니다. 사용자 고유의 암호화 키를 사용하여 로컬 클라이언트와 컨테이너 간의 데이터를 암호화하려면 AWS Key Management Service(AWS KMS) 키를 제공해야 합니다.

ECS Exec 사용

선택적 태스크 정의 변경 사항

작업 정의 파라미터 `initProcessEnabled`를 `true`로 설정하면 컨테이너 내부에서 init 프로세스가 시작됩니다. 그러면 발견된 좀비 SSM 에이전트 하위 프로세스가 제거됩니다. 다음은 예제를 제공합니다.

```
{
    "taskRoleArn": "ecsTaskRole",
    "networkMode": "awsvpc",
    "requiresCompatibilities": [
        "EC2",
        "FARGATE"
    ],
    "executionRoleArn": "ecsTaskExecutionRole",
    "memory": ".5 gb",
    "cpu": ".25 vcpu",
    "containerDefinitions": [
        {
            "name": "amazon-linux",
            "image": "amazonlinux:latest",
            "essential": true,
            "command": ["sleep","3600"],
            "linuxParameters": {
                "initProcessEnabled": true
            }
        }
    ],
    "family": "ecs-exec-task"
}
```

작업 및 서비스에 대해 ECS Exec 켜기

AWS CLI 명령 create-service, update-service, start-task 또는 run-task 중 하나를 사용할 때 --enable-execute-command 플래그를 지정하여 서비스 및 독립 실행형 작업에 대해 ECS Exec 기능을 켤 수 있습니다.

예를 들어 다음 명령을 실행하면 Fargate에서 실행되는 새로 생성된 서비스에 대해 ECS Exec 기능이 켜집니다. 서비스 생성에 대한 자세한 내용은 create-service 섹션을 참조하세요.

```
aws ecs create-service \
    --cluster cluster-name \
    --task-definition task-definition-name \
    --enable-execute-command \
    --service-name service-name \
    --launch-type FARGATE \
     --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=EN
 \
    --desired-count 1
```

작업에 대해 ECS Exec을 켠 후 다음 명령을 실행하여 작업을 사용할 준비가 되었는지 확인할 수 있습니다. ExecuteCommandAgent의 lastStatus 속성이 RUNNING으로 나열되고 enableExecuteCommand 속성이 true로 설정되면 작업이 준비가 된 것입니다.

```
aws ecs describe-tasks \
    --cluster cluster-name \
    --tasks task-id
```

볼 수 있는 출력 조각의 예는 다음과 같습니다.

```
{
    "tasks": [
        {
            ...
            "containers": [
                {
                    ...
                    "managedAgents": [
                        {
                            "lastStartedAt": "2021-03-01T14:49:44.574000-06:00",
                            "name": "ExecuteCommandAgent",
```

```
                            "lastStatus": "RUNNING"
                        }
                    ]
                }
            ],
            ...
            "enableExecuteCommand": true,
            ...
        }
    ]
}
```

ECS Exec을 사용하여 명령 실행

`ExecuteCommandAgent`가 실행 중임을 확인한 후 다음 명령을 사용하여 컨테이너에서 대화형 셸을 열 수 있습니다. 작업에 컨테이너가 여러 개 포함되어 있는 경우 `--container` 플래그를 사용하여 컨테이너 이름을 지정해야 합니다. Amazon ECS는 대화형 세션 시작만 지원하므로 `--interactive` 플래그를 사용해야 합니다.

다음 명령은 ID가 *`task-id`*인 작업에 대해 *`container-name`*이라는 컨테이너에서 대화형 `/bin/sh` 명령을 실행합니다.

*`task-id`*는 작업의 Amazon 리소스 이름(ARN)입니다.

```
aws ecs execute-command --cluster cluster-name \
    --task task-id \
    --container container-name \
    --interactive \
    --command "/bin/sh"
```

ECS Exec을 사용한 로깅

태스크 및 서비스 로그인 켜기

> **Important**
>
> CloudWatch 요금에 대한 자세한 정보는 CloudWatch 요금을 참조하세요. 또한 Amazon ECS는 추가 비용 없이 모니터링 지표도 제공합니다. 자세한 내용은 CloudWatch를 사용하여 Amazon ECS 모니터링 섹션을 참조하세요.

Amazon ECS는 ECS Exec을 사용하여 실행되는 로깅 명령에 대한 기본 구성을 제공합니다. 기본값은 작업 정의에 구성된 awslogs 로그 드라이버를 사용하여 CloudWatch Logs로 로그를 전송하는 것입니다. 사용자 지정 구성을 제공하려는 경우 AWS CLI는 create-cluster 및 update-cluster 명령 모두에 대해 --configuration 플래그를 지원합니다. 컨테이너 이미지는 script 및 cat을 설치하여 명령 로그가 Amazon S3 또는 CloudWatch Logs에 올바르게 업로드되게 해야 합니다. 클러스터 생성에 대한 자세한 내용은 create-cluster를 참조하세요.

> **Note**
>
> 이 구성은 execute-command 세션의 로깅만을 처리합니다. 애플리케이션의 로깅에는 영향을 미치지 않습니다.

다음 예제에서는 클러스터를 생성한 다음 출력을 cloudwatch-log-group-name이라는 CloudWatch Logs LogGroup과 s3-bucket-name이라는 Amazon S3 버킷에 로깅합니다.

CloudWatchEncryptionEnabled 옵션을 true로 설정할 때 AWS KMS 고객 관리형 키를 사용하여 로그 그룹을 암호화해야 합니다. 로그 그룹을 암호화하는 방법에 대한 자세한 내용은Amazon CloudWatch Logs 사용 설명서의 AWS Key Management Service를 사용하여 CloudWatch Logs의 로그 데이터 암호화를 참조하세요.

```
aws ecs create-cluster \
    --cluster-name cluster-name \
    --configuration executeCommandConfiguration="{ \
        kmsKeyId=string, \
        logging=OVERRIDE, \
        logConfiguration={ \
            cloudWatchLogGroupName=cloudwatch-log-group-name, \
            cloudWatchEncryptionEnabled=true, \
            s3BucketName=s3-bucket-name, \
            s3EncryptionEnabled=true, \
            s3KeyPrefix=demo \
        } \
    }"
```

logging 속성은 ECS Exec의 로깅 기능 동작을 결정합니다.

- NONE: 로깅이 꺼져 있습니다.
- DEFAULT: 구성된 awslogs 드라이버로 로그가 전송됩니다. 드라이버가 구성되지 않은 경우 로그는 저장되지 않습니다.

- `OVERRIDE`: 제공된 Amazon CloudWatch Logs LogGroup, Amazon S3 버킷 또는 두 위치 모두에 로그가 전송됩니다.

Amazon CloudWatch Logs 또는 Amazon S3 로깅에 필요한 IAM 권한

로깅을 활성화하려면 태스크 정의에서 참조되는 Amazon ECS 작업 역할에 추가 권한이 있어야 합니다. 이러한 추가 권한은 작업 역할에 정책으로 추가할 수 있습니다. Amazon CloudWatch Logs 또는 Amazon S3에 로그를 전달하는지 여부에 따라 다릅니다.

Amazon CloudWatch Logs

다음 예제 정책은 필수 Amazon CloudWatch Logs 권한을 추가합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
               "logs:CreateLogStream",
               "logs:DescribeLogStreams",
               "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:arn:aws:: 111122223333:log-group:/aws/
ecs/cloudwatch-log-group-name:*"
        }
   ]
}
```

Amazon S3

다음 예제 정책은 필수 Amazon S3 권한을 추가합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "s3:GetBucketLocation"
            ],
            "Resource": "*"
        },
        {
           "Effect": "Allow",
           "Action": [
               "s3:GetEncryptionConfiguration"
           ],
           "Resource": "arn:aws:s3:::s3-bucket-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::s3-bucket-name/*"
        }
    ]
   }
```

자체 AWS KMS key(KMS 키)를 사용한 암호화에 필요한 IAM 권한

기본적으로 로컬 클라이언트와 컨테이너 간에 전송되는 데이터는 AWS에서 제공하는 TLS 1.2 암호화를 사용합니다. 자체 KMS 키를 사용하여 데이터를 추가로 암호화하려면 KMS 키를 생성하고 `kms:Decrypt` 권한을 작업 IAM 역할에 추가해야 합니다. 이 권한은 데이터를 복호화하기 위해 컨테이너에서 사용됩니다. KMS 키 생성에 대한 자세한 내용은 키 생성을 참조하세요.

AWS KMS 권한이 필요한 작업 IAM 역할에 다음과 같은 인라인 정책을 추가합니다. 자세한 내용은 ECS Exec 권한 섹션을 참조하세요.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource":
 "arn:aws:kms:arn:aws::111122223333:key/12345678-1234-1234-1234-123456789012"
        }
    ]
}
```

자체 KMS 키를 사용하여 데이터를 암호화하려면 `execute-command` 태스크를 사용하는 사용자 또는 그룹에 `kms:GenerateDataKey` 권한을 부여해야 합니다.

사용자 또는 그룹에 대한 다음 예제 정책에는 자체 KMS 키를 사용하는 데 필요한 권한이 포함되어 있습니다. KMS 키의 Amazon 리소스 이름(ARN)을 지정해야 합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:arn:aws::111122223333:key/key"
        }
    ]
}
```

IAM 정책을 사용하여 ECS Exec에 대한 액세스 제한

다음 IAM 정책 조건 키 중 하나 이상을 사용하여 execute-command API 작업에 대한 사용자 액세스를 제한합니다.

- `aws:ResourceTag/clusterTagKey`
- `ecs:ResourceTag/clusterTagKey`
- `aws:ResourceTag/taskTagKey`
- `ecs:ResourceTag/taskTagKey`
- `ecs:container-name`
- `ecs:cluster`
- `ecs:task`
- `ecs:enable-execute-command`

사용자는 다음 예제 IAM 정책을 사용하여 `environment` 키와 `development` 값이 있는 태그가 포함된 작업 내에서 실행 중인 컨테이너와 `cluster-name`이라는 클러스터에서 명령을 실행할 수 있습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:ExecuteCommand",
                "ecs:DescribeTasks"
            ],
            "Resource": [
                   "arn:aws:ecs:arn:aws::111122223333:task/cluster-name/*",
                   "arn:aws:ecs:arn:aws::111122223333:cluster/cluster-name"
            ],
            "Condition": {
                "StringEquals": {
                    "ecs:ResourceTag/environment": "development"
                }
            }
```

```
        }
    ]
}
```

다음 IAM 정책 예제에서 사용자는 컨테이너 이름이 `production-app`일 때 `execute-command` API를 사용할 수 없습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "ecs:ExecuteCommand"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:container-name": "production-app"
                }
            }
        }
    ]
}
```

다음 IAM 정책을 사용하면 ECS Exec이 꺼진 경우에만 작업을 시작할 수 있습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask",
                "ecs:StartTask",
```

```
                "ecs:CreateService",
                "ecs:UpdateService"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:enable-execute-command": "false"
                }
            }
        }
    ]
}
```

> **Note**
>
> 왜냐하면 `execute-command` API 작업에는 요청의 작업 및 클러스터 리소스만 포함되며 클러스터 및 태스크 태그만 평가되기 때문입니다.

IAM 정책 조건 키에 대한 자세한 내용은 서비스 승인 참조의 Amazon Elastic Container Service에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

세션 시작 작업에 대한 액세스 제한

ECS Exec 외부의 컨테이너에서 SSM 세션을 시작할 수 있지만 이로 인해 세션이 로깅되지 않을 수 있습니다. ECS Exec 외부에서 시작된 세션도 세션 할당량에 대해 계산됩니다. IAM 정책을 사용하여 Amazon ECS 작업에 대한 `ssm:start-session` 태스크를 직접 거부함으로써 이 액세스를 제한하는 것이 좋습니다. 모든 Amazon ECS 작업 또는 사용된 태그를 기반으로 특정 작업에 대한 액세스를 거부할 수 있습니다.

다음은 클러스터 이름이 지정된 모든 리전의 작업에서 `ssm:start-session` 작업에 대한 액세스를 거부하는 IAM 정책 예제입니다. 선택적으로 *`cluster-name`*이 있는 와일드카드를 포함할 수 있습니다.

JSON

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

```
        {
            "Effect": "Deny",
            "Action": "ssm:StartSession",
            "Resource": [
            "arn:aws:ecs:arn:aws:;: 111122223333:task/cluster-name/*"
            ]
        }
    ]
}
```

다음은 태그 키 Task-Tag-Key, 태그 값 Exec-Task로 태그가 지정된 모든 리전의 리소스에서 ssm:start-session 작업에 대한 액세스를 거부하는 IAM 정책 예제입니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "ssm:StartSession",
            "Resource": "arn:aws:ecs:*:*:task/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Task-Tag-Key": "Exec-Task"
                }
            }
        }
    ]
}
```

Amazon ECS Exec를 사용할 때 발생할 수 있는 문제에 대한 도움이 필요하면 Exec 관련 문제 해결을 참조하세요.

Amazon ECS에 대한 AWS Compute Optimizer 권장 사항

AWS Compute Optimizer는 Amazon ECS 작업 및 컨테이너 크기에 대한 권장 사항을 생성합니다. 자세한 내용은 AWS Compute Optimizer 사용 설명서의 AWS Compute Optimizer이란 무엇입니까? 섹션을 참조하세요.

AWS Fargate의 Amazon ECS 서비스 작업 및 컨테이너 크기에 대한 권장 사항

AWS Compute Optimizer는 AWS Fargate의 Amazon ECS 서비스 최적화에 대한 권장 사항을 생성합니다. AWS Compute Optimizer는 작업 CPU 및 작업 메모리 크기 및 컨테이너 CPU, 컨테이너 메모리, 컨테이너 메모리 예약 크기를 권장합니다. 이러한 권장 사항은 Compute Optimizer 콘솔의 다음 페이지에 표시됩니다.

- Fargate의 Amazon ECS 서비스에 대한 권장 사항 페이지
- Fargate의 Amazon ECS 서비스 세부 정보 페이지

자세한 내용은 AWS Compute Optimizer 사용 설명서의 Fargate의 Amazon ECS 서비스에 대한 권장 사항 보기를 참조하세요.

Amazon ECS 문제 해결

로드 밸런서, 작업, 서비스 또는 컨테이너 인스턴스의 문제를 해결해야 하는 경우가 있을 수 있습니다. 이 장은 Amazon ECS 컨테이너 에이전트, 컨테이너 인스턴스의 Docker 대몬, Amazon ECS 콘솔의 서비스 이벤트 로그에서 진단 정보를 찾는 데 도움이 됩니다.

중지된 작업에 대한 자세한 내용은 다음 중 하나를 참조하세요.

작업	자세히 알아보기
중지된 작업 오류를 해결합니다.	Amazon ECS 중지된 작업 오류 보기
중지된 작업 오류를 봅니다.	Amazon ECS 중지된 작업 오류 해결
중지된 작업 오류 코드를 검토합니다.	Amazon ECS 중지된 작업 오류 메시지
CannotPullContainer 작업 오류를 검토합니다.	Amazon ECS에서 CannotPullContainer 작업 오류
작업 IAM 역할 요청을 봅니다.	Amazon ECS 태스크에 대한 IAM 역할 요청 보기

서비스 오류에 대한 자세한 내용은 다음 중 하나를 참조하세요.

작업	자세히 알아보기
서비스 이벤트 메시지를 봅니다.	Amazon ECS 서비스 이벤트 메시지 보기
서비스 이벤트 메시지를 검토합니다.	Amazon ECS 서비스 이벤트 메시지
로드 밸런서 문제를 검토합니다.	Amazon ECS의 서비스 로드 밸런서 문제 해결

작업	자세히 알아보기	
서비스 Auto Scaling 문제를 검토합니다.	Amazon ECS에서 서비스 Auto Scaling 문제 해결	

작업 정의 오류에 대한 자세한 내용은 다음 중 하나를 참조하세요.

작업	자세히 알아보기	
작업 정의 메모리 오류를 해결합니다.	Amazon ECS 작업 정의 잘못된 CPU 또는 메모리 오류 문제 해결	

Amazon ECS 에이전트 오류에 대한 자세한 내용은 다음 중 하나를 참조하세요.

작업	자세히 알아보기	
Amazon ECS 컨테이너 에이전트 로그를 봅니다.	Amazon ECS 컨테이너 에이전트 로그 보기	
Amazon ECS 로그를 수집하는 방법을 알아봅니다.	Amazon ECS 로그 수집기를 사용하여 컨테이너 로그 수집	
Amazon ECS 에이전트를 사용하여 진단 세부 정보를 검색합니다.	에이전트 내부 검사를 통해 Amazon ECS 진단 세부 정보 검색	

Docker 오류에 대한 자세한 내용은 다음 중 하나를 참조하세요.

작업	자세히 알아보기	
Docker 진단을 사용합니다.	Amazon ECS의 Docker 진단	
Docker 디버그 모드를 켭니다.	Amazon ECS에서 Docker 대몬의 자세한 출력 구성	

작업	자세히 알아보기	
Docker API 오류 500 문제를 해결합니다.	Amazon ECS의 Docker API error (500): devmapper 문제 해결	

ECS Exec 및 Amazon ECS Anywhere 오류에 대한 자세한 내용은 다음 중 하나를 참조하세요.

작업	자세히 알아보기	
ECS Exec 관련 문제를 해결합니다.	Amazon ECS Exec 문제 해결	
Amazon ECS Anywhere 관련 문제를 해결합니다.	Amazon ECS Anywhere 문제 해결	

제한 문제에 대한 자세한 내용은 다음 중 하나를 참조하세요.

작업	자세히 알아보기	
Fargate 제한 할당량에 대해 알아봅니다.	AWS Fargate 제한 할당량	
Amazon ECS 제한 모범 사례에 대해 알아봅니다.	Amazon ECS 제한 문제 처리	

API 오류에 대한 자세한 내용은 다음 중 하나를 참조하세요.

작업	자세히 알아보기	
API 오류를 해결합니다.	Amazon ECS의 API 실패 이유	

Amazon ECS 중지된 작업 오류 해결

작업이 시작되지 않으면 콘솔과 `describe-tasks` 출력 파라미터(`stoppedReason` 및 `stoppedCode`)에 오류 메시지가 표시됩니다.

1시간 동안 콘솔에서 중지된 작업을 볼 수 있습니다. 중지된 작업을 보려면 필터 옵션을 변경해야 합니다. 자세한 내용은 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

다음 페이지에서는 중지된 작업에 대한 정보를 제공합니다.

- 중지된 작업 오류 메시지의 변경사항에 대해 알아봅니다.

  Amazon ECS 중지된 작업 오류 메시지 업데이트
- 중지된 작업을 확인하여 원인에 대한 정보를 얻을 수 있습니다.

  Amazon ECS 중지된 작업 오류 보기
- 중지된 작업 오류 메시지와 오류의 가능한 원인에 대해 알아봅니다.

  Amazon ECS 중지된 작업 오류 메시지
- 중지된 작업 연결을 확인하고 오류를 수정하는 방법을 알아봅니다.

  Amazon ECS 중지된 작업 오류 연결 확인

Amazon ECS 중지된 작업 오류 메시지 업데이트

2024년 6월 14일부터 Amazon ECS 팀은 다음 테이블에 설명된 대로 중지된 작업 오류 메시지를 변경하고 있습니다. `stopCode`는 변경되지 않습니다. 애플리케이션이 정확한 오류 메시지 문자열을 사용하는 경우 애플리케이션을 새 문자열로 업데이트해야 합니다. 질문이나 문제에 대한 도움이 필요하면 AWS Support에 문의하세요.

> **Note**
>
> 오류 메시지는 변경될 수 있으므로 자동화에서 오류 메시지에 의존하지 않는 것이 좋습니다.

CannotPullContainerError

오래된 오류 메시지입니다.	새 오류 메시지	
CannotPullContainerError: Error response from daemon: pull access denied for *repository* , repository does not exist or may require 'docker login': denied: User: *roleARN*	• CannotPullContainerError: The task can't pull the image. Check that the role has the permissions to pull images from the registry. Error response from daemon: pull access denied for *repository* , repository does not exist or may require 'docker login': denied: User: *roleARN* is not authorized to perform: ecr:BatchGetImage on resource: *image* because no identity-based policy allows the ecr:Batch GetImage action. • CannotPullContainerError: The task can't pull the image. Check whether the image exists. Error response from daemon: pull access denied for *repository* , repositor y does not exist or may require 'docker login': denied: requested access to the resource is denied.	
CannotPullContainerError: Error response from daemon: Get *imageURI*: net/http:	CannotPullContainerError: The task can't pull the image. Check your network configura tion. Error response from	

오래된 오류 메시지입니다.	새 오류 메시지
request canceled while waiting for connection	daemon: Get *image*: net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)
CannotPullContainerError: ref pull has been retried 5 time(s): failed to copy: httpReadSeeker: failed open: failed to do request: Get *registry-uri* : dial tcp <ip>:<port> i/o timeout	CannotPullContainerError: The task cannot pull *image-uri* from the registry *registry-uri* . There is a connection issue between the task and the registry. Check your task network configuration. : failed to copy: httpReadSeeker: failed open: failed to do request: Get *registry-uri* : dial tcp <ip>:<port> i/o timeout

ResourceNotFoundException

오래된 오류 메시지입니다.	새 오류 메시지
Fetching secret data from AWS Secrets Manager in region *region*: secret *sercretARN* : ResourceNotFoundException: Secrets Manager can't find the specified secret.	ResourceNotFoundException: The task can't retrieve the secret with ARN '*sercretARN* ' from AWS Secrets Manager. Check whether the secret exists in the specified Region. ResourceNotFoundException: Fetching secret data from AWS Secrets Manager in region *region*: secret *sercretARN* : ResourceN

오래된 오류 메시지입니다.	새 오류 메시지	
	otFoundException: Secrets Manager can't find the specified secret.	

ResourceInitializationError

오래된 오류 메시지입니다.	새 오류 메시지	
ResourceInitializationError : unable to pull secrets or registry auth: execution resource retrieval failed: unable to retrieve ecr registry auth: service call has been retried 3 time(s): RequestEr ror: send request failed caused by: Post "https:// api.ecr.us-east-1.amazonaws .com/": dial tcp <ip>:<port>: i/ o timeout. Please check your task network configuration.	ResourceInitializationError : unable to pull secrets or registry auth: The task cannot pull registry auth from Amazon ECR: There is a connection issue between the task and Amazon ECR. Check your task network configuration. RequestError: send request failed caused by: Post "https:// api.ecr.us-east-1.amazonaws .com": dial tcp <ip>:<port>: i/o timeout	
ResourceInitializationError : unable to pull secrets or registry auth: execution resource retrieval failed: unable to retrieve secrets from ssm: service call has been retried 5 time(s): RequestCa nceled: request context canceled caused by: context deadline exceeded	ResourceInitializationError : unable to pull secrets or registry auth: unable to retrieve secrets from ssm: The task cannot pull secrets from AWS Systems Manager. There is a connectio n issue between the task and AWS Systems Manager Parameter Store. Check your task network configuration. RequestCanceled: request	

오래된 오류 메시지입니다.	새 오류 메시지	
	context canceled caused by: context deadline exceeded	
ResourceInitializationError: failed to download env files: file download command: non empty error stream: RequestCanceled: request context canceled caused by: context deadline exceeded	ResourceInitializationError: failed to download env files: The task can't download the environment variable files from Amazon S3. There is a connection issue between the task and Amazon S3. Check your task network configuration. service call has been retried 5 time(s): RequestCanceled: request context canceled caused by: context deadline exceeded	
ResourceInitializationError : failed to validate logger args::signal:killed	ResourceInitializationError : failed to validate logger args: The task cannot find the Amazon CloudWatch log group defined in the task definition. There is a connectio n issue between the task and Amazon CloudWatch. Check your network configuration. : signal: killed	
ResourceInitializationError : unable to pull secrets or registry auth: pull command failed: : signal: killed	ResourceInitializationError : unable to pull secrets or registry auth: Check your task network configuration. : signal: killed	

Amazon ECS 중지된 작업 오류 보기

작업을 시작하는 데 문제가 있는 경우, 오류 때문에 작업이 중지된 것일 수 있습니다. 예를 들면, 작업 실행 시 작업이 `PENDING` 상태를 표시한 후 사라지는 경우입니다.

작업이 Amazon ECS 서비스에서 생성된 경우 Amazon ECS가 서비스를 유지하기 위해 수행하는 작업은 서비스 이벤트에 게시됩니다. AWS Management Console, AWS CLI, AWS SDK, Amazon ECS API 또는 SDK 및 API를 사용하는 도구에서 이벤트를 볼 수 있습니다. 이러한 이벤트에는 작업의 컨테이너가 실행을 중지했거나 Elastic Load Balancing에서 너무 많은 상태 확인에 실패했기 때문에 작업을 중지하고 교체하는 Amazon ECS가 포함됩니다.

Amazon EC2 또는 외부 컴퓨터의 컨테이너 인스턴스에서 작업이 실행된 경우 컨테이너 런타임 및 Amazon ECS Agent의 로그도 볼 수 있습니다. 이 로그는 호스트 Amazon EC2 인스턴스 또는 외부 컴퓨터에 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 로그 보기 섹션을 참조하세요.

절차

Console

AWS Management Console

콘솔을 사용하여 오류로 인해 중단된 작업을 확인하려면 다음의 단계를 사용할 수 있습니다. 중지된 작업을 보려면 필터 옵션을 변경해야 합니다.

중지된 태스크는 1시간 동안만 콘솔에 나타납니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
4. 클러스터: ***name***(Cluster : name) 페이지에서 태스크(Tasks) 탭을 선택합니다.
5. 중지된 작업을 표시하도록 필터를 구성합니다. 원하는 상태 필터링에서 중지됨을 선택합니다.

   중지됨 옵션에는 중지된 작업이 표시되고 원하는 상태에는 모든 작업이 표시됩니다.
6. 검사할 중지된 태스크를 선택합니다.
7. 중지된 작업 행의 마지막 상태 열에서 중지됨을 선택합니다.

   팝업 창에 중지된 이유가 표시됩니다.

AWS CLI

1. 클러스터의 태스크를 나열합니다. 결과에는 작업의 Amazon 리소스 이름(ARN)이 포함되어 있으며 이를 통해 태스크를 설명해야 합니다.

```
aws ecs list-tasks \
     --cluster cluster_name \
     --desired-status STOPPED \
     --region region
```

2. 중지된 작업을 설명하여 정보를 검색합니다. 자세한 내용은 AWS Command Line Interface 참조의 describe-tasks를 참조하세요.

```
aws ecs describe-tasks \
     --cluster cluster_name \
     --tasks arn:aws:ecs:region:account_id:task/cluster_name/task_ID \
     --region region
```

다음 출력 파라미터를 사용합니다.

- `stopCode` - 중지 코드는 작업이 중지된 이유를 나타냅니다(예: ResourceInitializationError).
- `StoppedReason` - 작업이 중지된 이유입니다.
- `reason`(`containers` 구조 내) - 이유에서는 중지된 컨테이너에 대한 추가 세부 정보를 제공합니다.

다음 단계

중지된 작업을 확인하여 원인에 대한 정보를 얻을 수 있습니다. 자세한 내용은 Amazon ECS 중지된 작업 오류 메시지 섹션을 참조하세요.

Amazon ECS 중지된 작업 오류 메시지

다음은 작업이 예기치 않게 중지될 때 나타날 수 있는 가능한 오류 메시지입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

중지된 태스크 오류 코드에는 이와 연결된 카테고리가 있습니다(예: 'ResourceInitializationError'). 각 카테고리에 대한 자세한 내용은 다음을 참조하세요.

범주	자세히 알아보기
TaskFailedToStart	Amazon ECS TaskFailedToStart 오류 문제 해결
ResourceInitializationError	Amazon ECS ResourceInitializationError 오류 문제 해결
ResourceNotFoundException	Amazon ECS ResourceNotFoundException 오류 문제 해결
SpotInterruptionError	Amazon ECS SpotInterruption 오류 문제 해결
InternalError	Amazon ECS InternalError 오류 문제 해결
OutOfMemoryError	Amazon ECS OutOfMemoryError 오류 문제 해결
ContainerRuntimeError	Amazon ECS ContainerRuntimeError 오류 문제 해결
ContainerRuntimeTimeoutError	Amazon ECS ContainerRuntimeTimeoutError 오류 문제 해결
CannotStartContainerError	Amazon ECS CannotStartContainerError 오류 문제 해결
CannotStopContainerError	Amazon ECS CannotStopContainerError 오류 문제 해결

범주	자세히 알아보기	
CannotInspectContainerError	Amazon ECS CannotInspectContainerError 오류 문제 해결	
CannotCreateVolumeError	Amazon ECS CannotCreateVolumeError 오류 문제 해결	
CannotPullContainer	Amazon ECS에서 CannotPullContainer 작업 오류	

Amazon ECS TaskFailedToStart 오류 문제 해결

다음은 몇 가지 `TaskFailedToStart` 오류 메시지와 오류를 수정하기 위해 수행할 수 있는 작업입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

Unexpected EC2 error while attempting to Create Network Interface with public IP assignment enabled in subnet '***subnet-id***

이 오류는 Fargate 작업이 `awsvpc` 네트워크 모드를 사용하고 퍼블릭 IP 주소가 있는 서브넷에서 실행되지만 서브넷에 충분한 IP 주소가 없는 경우에 발생합니다.

Amazon EC2 콘솔의 서브넷 세부 정보 페이지나 `describe-subnets`를 사용하여 가용 IP 주소 개수가 표시됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 View your subnet을 참조하세요.

이 문제를 해결하기 위해 작업을 실행할 새 서브넷을 생성할 수 있습니다.

InternalError: ***<reason>***

이 오류는 ENI 연결을 요청할 때 발생합니다. Amazon EC2는 ENI의 프로비저닝을 비동기식으로 처리합니다. 프로비저닝 프로세스는 시간이 소요됩니다. 대기 시간이 길거나 보고되지 않은 실패가 있을 경우 Amazon ECS에 시간 초과가 발생합니다. ENI가 프로비저닝되는 시간이 있지만, 실패 시간 초과 이후에 보고서가 Amazon ECS로 전달됩니다. 이 경우, Amazon ECS에 사용 중인 ENI와 함께 보고된 태스크 실패가 표시됩니다.

The selected task definition is not compatible with the selected compute strategy

이 오류는 시작 유형이 클러스터 용량 유형과 일치하지 않는 작업 정의를 선택한 경우에 발생합니다. 자세한 내용은 Amazon ECS 시작 유형 섹션을 참조하세요. 클러스터에 할당된 용량 공급자와 일치하는 태스크 정의를 선택해야 합니다.

사용하지 않는 디바이스 인덱스에 네트워크 인터페이스를 연결할 수 없음

이 오류는 awsvpc 네트워킹 유형을 사용할 때 작업에 필요한 CPU/메모리가 충분하지 않을 때 발생합니다. 먼저 CPU에서 인스턴스를 확인합니다. 자세한 내용은 Amazon EC2 인스턴스 유형의 Amazon EC2 인스턴스 유형 사양을 참조하십시오. 인스턴스의 CPU 값을 가져와 해당 인스턴스의 ENI 수를 곱합니다. 이 값을 태스크 정의에서 사용하세요.

AGENT

태스크를 시작하려고 시도한 컨테이너 인스턴스에 현재 연결이 끊긴 에이전트가 있습니다. 작업 배치를 위한 대기 시간이 길어지는 것을 방지하기 위해 요청이 거부되었습니다.

연결이 끊어진 에이전트 문제 해결 방법에 대한 자세한 내용은 연결이 끊어진 Amazon ECS 에이전트 문제를 해결하려면 어떻게 해야 하나요?를 참조하세요.

Amazon ECS ResourceInitializationError 오류 문제 해결

다음은 몇 가지 ResourceInitialization 오류 메시지와 오류를 수정하기 위해 수행할 수 있는 작업입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

오류

- The task cannot pull registry authentication from Amazon ECR. There is a connection issue between the task and Amazon ECR. Check your task network configuration.
- The task can't download the environment variable files from Amazon S3. There is a connection issue between the task and Amazon S3. Check your task network configuration.
- The task cannot pull secrets from AWS Systems Manager Parameter Store. Check your network connection between the task and AWS Systems Manager.
- The task can’t pull secrets from AWS Secrets Manager. There is a connection issue between the task and Secrets Manager. Check your task network configuration.

- The task can’t pull the secret from Secrets Manager. The task can't retrieve the secret with ARN ‘secretARN' from Secrets Manager. Check whether the secret exists in the specified Region.
- pull command failed: unable to pull secrets or registry auth Check your task network configuration.
- The task cannot find the Amazon CloudWatch log group defined in the task definition. There is a connection issue between the task and Amazon CloudWatch. Check your network configuration.
- failed to initialize logging driver
- failed to invoke EFS utils commands to set up EFS volumes

The task cannot pull registry authentication from Amazon ECR. There is a connection issue between the task and Amazon ECR. Check your task network configuration.

이 오류는 작업을 Amazon ECR에 연결할 수 없음을 나타냅니다.

태스크와 Amazon ECR 간의 연결을 확인하세요. 자세한 내용은 Amazon ECS 중지된 작업 오류 연결 확인 섹션을 참조하세요.

The task can't download the environment variable files from Amazon S3. There is a connection issue between the task and Amazon S3. Check your task network configuration.

이 오류는 작업이 Amazon S3에서 환경 파일을 다운로드할 수 없는 경우에 발생합니다.

태스크와 Amazon S3 VPC 엔드포인트 간의 연결을 확인하세요. 자세한 내용은 Amazon ECS 중지된 작업 오류 연결 확인 섹션을 참조하세요.

The task cannot pull secrets from AWS Systems Manager Parameter Store. Check your network connection between the task and AWS Systems Manager.

이 오류는 작업이 Systems Manager의 자격 증명을 사용하여 작업 정의에서 정의한 이미지를 가져올 수 없는 경우에 발생합니다.

태스크와 Systems Manager VPC 엔드포인트 간의 연결을 확인하세요. 자세한 내용은 Amazon ECS 중지된 작업 오류 연결 확인 섹션을 참조하세요.

The task can’t pull secrets from AWS Secrets Manager. There is a connection issue between the task and Secrets Manager. Check your task network configuration.

이 오류는 작업이 Secrets Manager의 자격 증명을 사용하여 작업 정의에서 정의한 이미지를 가져올 수 없는 경우에 발생합니다.

이 오류는 Systems Manager VPC 엔드포인트 및 작업 간 네트워크 연결 문제가 있음을 나타냅니다.

작업 및 엔드포인트 간 연결을 확인하는 방법에 대한 자세한 내용은 Amazon ECS 중지된 작업 오류 연결 확인 섹션을 참조하세요.

The task can’t pull the secret from Secrets Manager. The task can't retrieve the secret with ARN ‘***secretARN***' from Secrets Manager. Check whether the secret exists in the specified Region.

이 오류는 작업이 Secrets Manager의 자격 증명을 사용하여 작업 정의에서 정의한 이미지를 가져올 수 없는 경우에 발생합니다.

이 문제의 가능한 원인은 다음 중 하나입니다.

오류 원인	수행할 작업	
Secrets Manager VPC 엔드포인트 및 작업 간 네트워크 연결 문제입니다. 오류 메시지에 다음 문자열 중 하나가 표시되는 경우 네트워크 문제에 해당합니다. • dial tcp • dial udp • <ip>:<port>: i/o timeout • net/http: TLS handshake timeout • read: connection timed out • Client.Timeout exceeded while awaiting headers • net/http: request canceled while waiting for connection • signal: killed • context deadline exceeded	작업 및 Secrets Manager 엔드포인트 간 연결을 확인합니다. 자세한 내용은 Amazon ECS 중지된 작업 오류 연결 확인 섹션을 참조하세요.	
작업 정의에 정의된 작업 실행 역할에서 Secrets Manager에	작업 실행 역할에 필요한 Secrets Manager 권한을 추가합니다. 자세한 내용	

오류 원인	수행할 작업	
대한 권한을 보유하지 않습니다.	은 Secrets Manager 또는 Systems Manager 권한 섹션을 참조하세요.	
암호 ARN이 존재하지 않음	Secrets Manager에 ARN이 있는지 확인합니다. 이미지를 보는 방법에 대한 자세한 내용은 Secrets Manager 개발자 안내서의 Find secrets in Secrets Manager를 참조하세요.	

pull command failed: unable to pull secrets or registry auth Check your task network configuration.

이 오류는 태스크가 Amazon ECR, Systems Manager 또는 Secrets Manager에 연결할 수 없을 때 발생합니다. 이는 네트워크의 잘못된 구성 때문입니다.

이 문제를 해결하려면 태스크와 Amazon ECR 간의 연결을 확인하세요. 또한 태스크와 보안 암호를 저장하는 서비스(Systems Manager 또는 Secrets Manager) 간의 연결도 확인해야 합니다. 자세한 내용은 Amazon ECS 중지된 작업 오류 연결 확인 섹션을 참조하세요.

The task cannot find the Amazon CloudWatch log group defined in the task definition. There is a connection issue between the task and Amazon CloudWatch. Check your network configuration.

이 오류는 작업에서 작업 정의에 정의한 CloudWatch 로그 그룹을 찾지 못하는 경우에 발생합니다.

이 오류는 CloudWatch VPC 엔드포인트와 작업 간에 네트워크 연결 문제가 있음을 나타냅니다.

작업 및 엔드포인트 간 연결을 확인하는 방법에 대한 자세한 내용은 Amazon ECS 중지된 작업 오류 연결 확인 섹션을 참조하세요.

failed to initialize logging driver

이 오류는 작업에서 작업 정의에 정의한 CloudWatch 로그 그룹을 찾지 못하는 경우에 발생합니다.

이 오류는 작업 정의에 CloudWatch 그룹이 없음을 나타냅니다.

다음 단계에 따라 누락된 CloudWatch를 찾으세요.

1. 다음 명령을 실행하여 태스크 정의 정보를 가져옵니다.

```
aws ecs describe-task-definition \
    --task-definition task-def-name
```

각 컨테이너의 출력을 확인하여 awslogs-group 값을 기록해 둡니다.

```
"logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/example-group",
                    "awslogs-create-group": "true",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "ecs"
                },
```

2. 그룹이 CloudWatch에 있는지 확인합니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 로그 그룹 및 로그 스트림 작업을 참조하세요.

   문제는 태스크 정의에 지정된 그룹이 잘못되었거나 로그 그룹이 존재하지 않는 것입니다.

3. 문제를 해결합니다.

문제	수행할 작업
태스크 정의에 잘못된 로그 그룹이 지정되었습니다.	컨테이너 정의에 로그 그룹 구성을 포함하도록 태스크 정의를 업데이트합니다. 태스크 정의 업데이트에 대한 자세한 내용은 Amazon Elastic Container Service API 참조의 콘솔을 사용하여 Amazon ECS 작업 정의 업데이트 또는 RegisterTaskDefinition 섹션을 참조하세요.
CloudWatch에 로그 그룹이 존재하지 않습니다.	로그 그룹을 생성합니다. 자세한 내용은 Amazon CloudWatch Logs User Guide의 Create a log group

문제	수행할 작업	
	in CloudWatch Logs를 참조하세요.	

failed to invoke EFS utils commands to set up EFS volumes

다음 문제로 인해 작업에 Amazon EFS 볼륨을 탑재하지 못할 수 있습니다.

- Amazon EFS 파일 시스템이 올바르게 구성되지 않았습니다.
- 작업에 필요한 권한이 없습니다.
- 네트워크 및 VPC 구성과 관련된 문제가 있습니다.

이 문제를 디버깅하고 해결하는 방법에 대한 자세한 내용은 AWS re:Post의 Amazon EFS 볼륨을 AWS Fargate 작업에 마운트할 수 없는 이유는 무엇인가요?를 참조하세요.

Amazon ECS ResourceNotFoundException 오류 문제 해결

다음은 몇 가지 `ResourceNotFoundException` 오류 메시지와 오류를 수정하기 위해 수행할 수 있는 작업입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

The task can't retrieve the secret with ARN '***sercretARN***' from AWS Secrets Manager. Check whether the secret exists in the specified Region.

이 오류는 작업이 Secrets Manager에서 암호를 검색할 수 없는 경우에 발생합니다. 즉, 작업 정의에 지정되고 암호(오류 메시지에 포함된 암호)가 Secrets Manager에 없습니다.

리전은 오류 메시지에 있습니다.

Fetching secret data from AWS Secrets Manager in region *region*: secret *sercretARN*: ResourceNotFoundException: Secrets Manager can't find the specified secret.

암호에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 Find secrets in AWS Secrets Manager를 참조하세요.

다음 테이블을 사용하여 오류를 확인하고 해결합니다.

문제	작업	
암호는 작업 정의와 다른 리전에 있습니다.	a. 작업과 같은 리전에서 암호를 생성합니다. 자세한 내용은 AWS Secrets Manager 보안 암호 생성을 참조하세요. b. 새 암호로 작업 정의를 업데이트합니다. 자세한 내용은 Amazon Elastic Container Service API 참조의 RegisterTaskDefinition 또는 콘솔을 사용하여 Amazon ECS 작업 정의 업데이트 섹션을 참조하세요.	
작업 정의에 잘못된 암호 ARN이 있습니다. 올바른 암호는 Secrets Manager에 있습니다.	올바른 암호로 작업 정의를 업데이트합니다. 자세한 내용은 Amazon Elastic Container Service API 참조의 RegisterTaskDefinition 또는 콘솔을 사용하여 Amazon ECS 작업 정의 업데이트 섹션을 참조하세요.	
암호가 더 이상 존재하지 않습니다.	a. 작업과 같은 리전에서 암호를 생성합니다. 자세한 내용은 AWS Secrets Manager 보안 암호 생성을 참조하세요. b. 새 암호로 작업 정의를 업데이트합니다. 자세한 내용은 Amazon Elastic Container Service API 참조의 RegisterTaskDefinition 또는 콘솔을 사용하여	

문제	작업	
	Amazon ECS 작업 정의 업데이트 섹션을 참조하세요.	

Amazon ECS SpotInterruption 오류 문제 해결

`SpotInterruption` 오류의 원인은 Fargate 및 EC2 시작 유형마다 다릅니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

Fargate 시작 유형

`SpotInterruption` 오류는 Fargate 스팟 용량이 없거나 Fargate에서 스팟 용량을 회수하는 경우에 발생합니다.

여러 가용 영역에서 작업을 실행하여 더 많은 용량을 확보할 수 있습니다.

EC2 시작 유형

이 오류는 사용 가능한 스팟 인스턴스가 없거나 EC2가 스팟 인스턴스 용량을 회수하는 경우에 발생합니다.

여러 가용 영역에서 인스턴스를 실행하여 더 많은 용량을 확보할 수 있습니다.

Amazon ECS InternalError 오류 문제 해결

적용 대상: Fargate 시작 유형

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

이 `InternalError` 오류는 에이전트에서 예기치 않은 비런타임 관련 내부 오류가 발생하는 경우에 나타납니다.

이 오류는 버전이 `1.4` 이상인 플랫폼을 사용하는 경우에만 발생합니다.

이 문제를 디버깅하고 해결하는 방법에 대한 자세한 내용은 Amazon ECS 중지된 작업 오류 메시지 섹션을 참조하세요.

Amazon ECS OutOfMemoryError 오류 문제 해결

다음은 몇 가지 OutOfMemoryError 오류 메시지와 오류를 해결하기 위해 수행할 수 있는 작업입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

container killed due to memory usage

이 오류는 컨테이너의 프로세스가 태스크 정의에 할당된 것보다 많은 메모리를 소비하거나 호스트 또는 운영 체제 제약 조건에 따라 컨테이너가 종료될 때 발생합니다.

Amazon ECS ContainerRuntimeError 오류 문제 해결

다음은 몇 가지 ContainerRuntimeError 오류 메시지와 오류를 해결하기 위해 수행할 수 있는 작업입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

ContainerRuntimeError

이 오류는 에이전트가 런타임별 작업에 대해 `containerd`에서 예기치 않은 오류를 수신할 때 발생합니다. 이 오류는 일반적으로 에이전트 또는 `containerd` 런타임의 내부 오류로 인해 발생합니다.

이 오류는 플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0` 이상(Windows)을 사용하는 경우에만 발생합니다.

이 문제를 디버깅하고 해결하는 방법에 대한 자세한 내용은 AWS re:Post에서 Amazon ECS 태스크가 중지되었는데 이유가 무엇인가요?를 참조하세요.

Amazon ECS ContainerRuntimeTimeoutError 오류 문제 해결

다음은 몇 가지 ContainerRuntimeTimeoutError 오류 메시지와 오류를 해결하기 위해 수행할 수 있는 작업입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

Could not transition to running; timed out after waiting 1m or Docker timeout error

이 오류는 컨테이너가 제한 시간 내에 `RUNNING` 또는 `STOPPED` 상태로 전환 할 수 없을 때 발생합니다. 오류 메시지에 원인 및 제한 시간 값이 제공됩니다.

Amazon ECS CannotStartContainerError 오류 문제 해결

다음은 몇 가지 CannotStartContainerError 오류 메시지와 오류를 해결하기 위해 수행할 수 있는 작업입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

failed to get container status: ***<reason>***

이 오류는 컨테이너를 시작할 수 없을 때 발생합니다.

컨테이너가 여기에 지정된 메모리를 초과하려 하면 해당 컨테이너가 중지됩니다. 컨테이너에 제공되는 메모리를 늘립니다. 태스크 정의에서 `memory` 파라미터를 사용하면 됩니다. 자세한 내용은 the section called “메모리” 섹션을 참조하세요.

Amazon ECS CannotStopContainerError 오류 문제 해결

다음은 몇 가지 CannotStopContainerError 오류 메시지와 오류를 해결하기 위해 수행할 수 있는 작업입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

CannotStopContainerError

이 오류는 컨테이너가 중지되었을 때 발생합니다.

이 문제를 디버깅하고 해결하는 방법에 대한 자세한 내용은 AWS re:Post에서 Amazon ECS 태스크가 중지되었는데 이유가 무엇인가요?를 참조하세요.

Amazon ECS CannotInspectContainerError 오류 문제 해결

다음은 몇 가지 CannotInspectContainerError 오류 메시지와 오류를 해결하기 위해 수행할 수 있는 작업입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

CannotInspectContainerError

이 오류는 컨테이너 에이전트가 컨테이너 런타임을 통해 컨테이너를 설명할 수 없을 때 발생합니다.

플랫폼 버전 `1.3` 이하를 사용하면 Amazon ECS 에이전트가 Docker에서 이유를 반환합니다.

플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0` 이상(Windows)을 사용하는 경우 Fargate 에이전트는 `containerd`에서 이유를 반환합니다.

이 문제를 디버깅하고 해결하는 방법에 대한 자세한 내용은 AWS re:Post에서 Amazon ECS 태스크가 중지되었는데 이유가 무엇인가요?를 참조하세요.

Amazon ECS CannotCreateVolumeError 오류 문제 해결

다음은 몇 가지 CannotCreateVolumeError 오류 메시지와 오류를 해결하기 위해 수행할 수 있는 작업입니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

CannotCreateVolumeError

이 오류는 에이전트가 태스크 정의에 지정된 볼륨 마운트를 생성할 수 없는 경우에 발생합니다.

이 오류는 플랫폼 버전 `1.4.0` 이상(Linux) 또는 `1.0.0` 이상(Windows)을 사용하는 경우에만 발생합니다.

이 문제를 디버깅하고 해결하는 방법에 대한 자세한 내용은 AWS re:Post에서 Amazon ECS 태스크가 중지되었는데 이유가 무엇인가요?를 참조하세요.

Amazon ECS에서 CannotPullContainer 작업 오류

다음 오류는 Amazon ECS가 지정된 컨테이너 이미지를 검색할 수 없어 작업을 시작하지 못했음을 나타냅니다.

> **Note**
>
> 1.4 Fargate 플랫폼 버전은 긴 오류 메시지를 자릅니다.

AWS Management Console을 사용하여 중지된 작업에서 오류 메시지를 확인하려면 Amazon ECS 중지된 작업 오류 보기 섹션을 참조하세요.

오류

- The task can't pull the image. Check that the role has the permissions to pull images from the registry.

- The task cannot pull ‘image-name’ from the Amazon ECR repository ‘repository URI’. There is a connection issue between the task and Amazon ECR. Check your task network configuration.
- The task can’t pull the image. Check your network configuration
- CannotPullContainerError: 풀 이미지 매니페스트가 5회 재시도됨: 참조를 확인하지 못함
- API error (500): Get https://111122223333.dkr.ecr.us-east-1.amazonaws.com/v2/: net/http: request canceled while waiting for connection
- API 오류
- write /var/lib/docker/tmp/GetImageBlob111111111: no space left on device
- ERROR: toomanyrequests: Too Many Requests or You have reached your pull rate limit.
- Error response from daemon: Get url: net/http: request canceled while waiting for connection
- ref pull has been retried 1 time(s): failed to copy: httpReaderSeeker: failed open: unexpected status code
- pull access denied
- pull command failed: panic: runtime error: invalid memory address or nil pointer dereference
- error pulling image conf/error pulling image configuration
- 컨텍스트 취소됨

The task can’t pull the image. Check that the role has the permissions to pull images from the registry.

이 오류는 권한 문제 때문에 작업이 작업 정의에 지정된 이미지를 가져올 수 없음을 나타냅니다.

이 문제를 해결하려면:

1. *Rrepository*에 이미지가 있는지 확인합니다. 이미지 보기에 대한 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 Viewing image details in Amazon ECR을 참조하세요.
2. *role-arn*에 이미지를 가져올 수 있는 올바른 권한이 있는지 확인합니다.

   역할을 업데이트하는 방법에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 역할에 대한 권한 업데이트를 참조하세요.

   작업은 다음 역할 중 하나를 사용합니다.

   - Fargate 시작 유형의 작업인 경우 작업 실행 역할입니다. Amazon ECR의 추가 권한에 대한 자세한 내용은 인터페이스 엔드포인트 권한을 통해 Amazon ECR 이미지를 가져오는 Fargate 작업 섹션을 참조하세요.

- EC2 시작 유형의 작업인 경우 컨테이너 인스턴스 역할입니다. Amazon ECR의 추가 권한에 대한 자세한 내용은 Amazon ECR 권한 섹션을 참조하세요.

The task cannot pull '***image-name***' from the Amazon ECR repository '***repository URI***'. There is a connection issue between the task and Amazon ECR. Check your task network configuration.

이 오류는 작업을 Amazon ECR에 연결할 수 없음을 나타냅니다. *repository URI* 리포지토리에 대한 연결을 확인하세요.

이 문제를 확인하고 해결하는 방법에 대한 자세한 내용은 Amazon ECS 중지된 작업 오류 연결 확인 섹션을 참조하세요.

The task can't pull the image. Check your network configuration

이 오류는 작업을 Amazon ECR에 연결할 수 없음을 나타냅니다.

이 문제를 확인하고 해결하는 방법에 대한 자세한 내용은 Amazon ECS 중지된 작업 오류 연결 확인 섹션을 참조하세요.

CannotPullContainerError: 풀 이미지 매니페스트가 5회 재시도됨: 참조를 확인하지 못함

이 오류는 태스크가 이미지를 가져올 수 없음을 나타냅니다.

이 문제는 다음과 같은 방법으로 해결할 수 있습니다.

- 태스크 정의에 지정된 이미지가 리포지토리의 이미지와 일치하는지 확인하세요.
- Amazon ECS는 이미지 버전 안정성을 강화합니다. 원본 이미지를 더 이상 사용할 수 없는 경우 이 오류가 발생합니다. 이미지 태그는 이 동작을 적용하는 데 포함됩니다. 태스크 정의의 이미지를 태그로 :latest를 사용하는 것에서 특정 버전으로 변경합니다. 자세한 내용은 컨테이너 이미지 확인 섹션을 참조하세요.

이 문제를 확인하고 해결하는 방법에 대한 자세한 내용은 Amazon ECS 중지된 작업 오류 연결 확인 섹션을 참조하세요.

API error (500): Get https://111122223333.dkr.ecr.us-east-1.amazonaws.com/v2/: net/http: request canceled while waiting for connection

이 오류는 인터넷 경로가 존재하지 않아 연결 제한 시간이 초과되었음을 나타냅니다.

다음과 같은 방법으로 이 문제를 해결할 수 있습니다.

- 퍼블릭 서브넷에 있는 작업의 경우 태스크를 시작할 때 퍼블릭 IP 자동 할당(Auto-assign public IP)을 활성화됨(ENABLED)으로 지정합니다. 자세한 내용은 애플리케이션을 Amazon ECS 태스크로 실행 섹션을 참조하세요.
- 프라이빗 서브넷에 있는 작업의 경우 태스크를 시작할 때 퍼블릭 IP 자동 할당(Auto-assign public IP)을 비활성화됨(DISABLED)으로 지정하고 요청을 인터넷으로 라우팅하도록 VPC에 NAT 게이트웨이를 구성합니다. 자세한 내용은 Amazon VPC 사용 설명서의 NAT 게이트웨이 섹션을 참조하세요.

API 오류

이 오류는 Amazon ECR 엔드포인트에 연결 문제가 있음을 나타냅니다.

이 문제를 해결하는 방법에 대한 자세한 내용은 지원 웹 사이트의 Amazon ECS에서 Amazon ECR 오류 'CannotPullContainerError: API error'를 해결하려면 어떻게 해야 하나요?를 참조하세요.

write /var/lib/docker/tmp/***GetImageBlob111111111***: no space left on device

이 오류는 디스크 공간이 부족함을 나타냅니다.

이 문제를 해결하려면 디스크 공간을 늘리세요.

Amazon ECS 최적화 AMI를 사용하는 경우, 다음 명령을 사용하여 파일 시스템에서 가장 큰 파일 20개를 검색할 수 있습니다.

```
du -Sh / | sort -rh | head -20
```

출력 예시:

```
5.7G    /var/lib/docker/
containers/50501b5f4cbf90b406e0ca60bf4e6d4ec8f773a6c1d2b451ed8e0195418ad0d2
1.2G    /var/log/ecs
594M    /var/lib/docker/devicemapper/mnt/
c8e3010e36ce4c089bf286a623699f5233097ca126ebd5a700af023a5127633d/rootfs/data/logs
...
```

일부 경우에 실행 중인 컨테이너로 루트 볼륨을 채울 수 있습니다. 컨테이너에서 기본 `json-file` 로그 드라이브를 `max-size` 제한 없이 사용하는 경우, 로그 파일이 공간의 대부분을 사용하는 것일 수 있습니다. `docker ps` 명령으로 위 출력의 디렉토리 이름을 컨테이너 ID로 매핑하여 공간을 사용하는 컨테이너를 확인할 수 있습니다. 예:

```
CONTAINER ID   IMAGE                                COMMAND             CREATED
 STATUS              PORTS                               NAMES
50501b5f4cbf   amazon/amazon-ecs-agent:latest   "/agent"            4 days ago
 Up 4 days                                              ecs-agent
```

기본적으로 `json-file` 로그 드라이버를 사용할 때 Docker는 전체 컨테이너의 표준 출력(및 표준 오류)를 캡처하고 이것을 JSON 형식으로 파일에 작성합니다. 로그 파일이 공간을 너무 많이 차지하는 것을 방지하는 로그 드라이버 옵션으로 `max-size`를 설정할 수 있습니다. 자세한 내용은 Docker 설명서의 JSON 파일 로깅 드라이버를 참조하세요.

다음은 이 옵션을 사용하는 방법을 보여주는 컨테이너 정의 조각입니다.

```
{
    "log-driver": "json-file",
    "log-opts": {
        "max-size": "256m"
    }
}
```

또는 컨테이너 로그가 디스크 공간을 너무 많이 차지할 경우 `awslogs` 로그 드라이버를 사용하는 방법도 있습니다. `awslogs` 로그 드라이버는 CloudWatch로 로그를 전송하기 때문에 컨테이너 인스턴스에서 컨테이너 로그용으로 사용되는 만큼의 디스크 공간이 생깁니다. 자세한 내용은 Amazon ECS 로그를 CloudWatch로 전송 섹션을 참조하세요.

Docker에서 액세스할 수 있는 디스크 크기를 업데이트해야 할 수 있습니다.

자세한 내용은 CannotPullContainerError: no space left on device를 참조하세요.

ERROR: toomanyrequests: Too Many Requests or You have reached your pull rate limit.

이 오류는 Docker Hub 속도 제한이 있음을 나타냅니다.

다음 오류 중 하나가 발생하는 경우 Docker Hub 속도 제한에 도달했을 가능성이 있습니다.

Docker Hub 속도 제한에 대한 자세한 내용은 Docker Hub 속도 제한 이해 섹션을 참조하세요.

Docker Hub 속도 제한을 늘리고 컨테이너 인스턴스에 대한 Docker 가져오기를 인증해야 하는 경우 Private registry authentication for container instances를 참조하세요.

Error response from daemon: Get ***url***: net/http: request canceled while waiting for connection

이 오류는 인터넷 경로가 존재하지 않아 연결 제한 시간이 초과되었음을 나타냅니다.

다음과 같은 방법으로 이 문제를 해결할 수 있습니다.

- 퍼블릭 서브넷에 있는 작업의 경우 태스크를 시작할 때 퍼블릭 IP 자동 할당(Auto-assign public IP)을 활성화됨(ENABLED)으로 지정합니다. 자세한 내용은 애플리케이션을 Amazon ECS 태스크로 실행 섹션을 참조하세요.
- 프라이빗 서브넷에 있는 작업의 경우 태스크를 시작할 때 퍼블릭 IP 자동 할당(Auto-assign public IP)을 비활성화됨(DISABLED)으로 지정하고 요청을 인터넷으로 라우팅하도록 VPC에 NAT 게이트웨이를 구성합니다. 자세한 내용은 Amazon VPC 사용 설명서의 NAT 게이트웨이 섹션을 참조하세요.

ref pull has been retried 1 time(s): failed to copy: httpReaderSeeker: failed open: unexpected status code

이 오류는 이미지를 복사할 때 실패했음을 나타냅니다.

이 문제를 해결하려면 다음 문서 중 하나를 검토합니다.

- Fargate 작업에 대한 자세한 내용은 Fargate의 내 Amazon ECS 작업에 대한 "cannotpullcontainererror" 오류를 해결하는 방법을 참조하세요.
- 기타 작업에 대한 자세한 내용은 내 Amazon ECS 작업에 대한 "cannotpullcontainererror" 오류를 해결하는 방법을 참조하세요.

pull access denied

이 오류는 이미지에 대한 액세스 권한이 없음을 나타냅니다.

이 문제를 해결하려면 Amazon ECR을 사용하여 Docker 클라이언트를 인증해야 할 수 있습니다. 자세한 내용은 Amazon ECR 사용 설명서의 프라이빗 레지스트리 인증을 참조하세요.

pull command failed: panic: runtime error: invalid memory address or nil pointer dereference

이 오류는 잘못된 메모리 주소 또는 nil 포인터 역참조로 인해 이미지에 액세스할 수 없음을 나타냅니다.

이 문제를 해결하려면:

- Amazon S3에 접속하기 위한 보안 그룹 규칙이 있는지 확인합니다.
- 게이트웨이 엔드포인트를 사용하는 경우, 엔드포인트에 액세스하려면 라우팅 테이블에 경로를 추가해야 합니다.

error pulling image conf/error pulling image configuration

이 오류는 속도 제한에 도달했거나 네트워크 오류가 있음을 나타냅니다.

이 문제를 해결하려면 How can I resolve the "CannotPullContainerError" error in my Amazon ECS EC2 Launch Type Task를 참조하세요.

컨텍스트 취소됨

이 오류는 컨텍스트가 취소되었음을 나타냅니다.

이 오류의 일반적인 원인은 작업에서 사용 중인 VPC에 Amazon ECR에서 컨테이너 이미지를 가져오기 위한 경로가 없기 때문입니다.

Amazon ECS 중지된 작업 오류 연결 확인

네트워크 연결 문제로 인해 작업이 중지되는 경우가 있습니다. 간헐적으로 발생하는 문제일 수 있지만 대부분의 경우 작업을 엔드포인트에 연결할 수 없기 때문에 발생합니다.

작업 연결 테스트

`AWSSupport-TroubleshootECSTaskFailedToStart` 런북을 사용하여 작업 연결을 테스트할 수 있습니다. 런북을 사용하는 경우 다음 리소스 정보가 필요합니다.

- 작업 ID

  최근 실패한 작업을 사용합니다.
- 작업이 있는 클러스터

런북을 사용하는 방법에 대한 자세한 내용은 AWS Systems Manager 자동화 런북 참조의 `AWSSupport-TroubleshootECSTaskFailedToStart` 섹션을 참조하세요.

런북에서 작업을 분석합니다. 작업 시작을 방해할 수 있는 다음 문제에 대한 결과는 출력 섹션에서 볼 수 있습니다.

- 구성된 컨테이너 레지스트리에 대한 네트워크 연결
- VPC 엔드포인트 연결
- 보안 그룹 규칙 구성

VPC 엔드포인트 문제 해결

`AWSSupport-TroubleshootECSTaskFailedToStart` 런북 결과에 VPC 엔드포인트 문제가 표시되면 다음 구성을 확인합니다.

- 엔드포인트를 생성하는 VPC는 프라이빗 DNS를 사용해야 합니다.
- 서비스에서 작업과 동일한 VPC에 작업을 연결할 수 없는 AWS PrivateLink 엔드포인트가 있는지 확인합니다. 자세한 내용은 다음 중 하나를 참조하세요.

Service	서비스의 VPC 엔드포인트 정보
Amazon ECR	Amazon ECR 인터페이스 VPC 엔드포인트(AWS PrivateLink)
Systems Manager	Systems Manager용 VPC 엔드포인트를 사용하여 EC2 인스턴스의 보안 개선
Secrets Manager	AWS Secrets Manager VPC 엔드포인트 사용
CloudWatch	CloudWatch VPC 엔드포인트
Amazon S3	Amazon S3용 AWS PrivateLink

- 포트 443 DNS(TCP) 트래픽에서 HTTPS를 허용하는 작업 서브넷의 아웃바운드 규칙을 구성합니다. 자세한 내용은 Amazon Elastic Compute Cloud 사용 설명서의 보안 그룹 규칙 구성을 참조하세요.
- 사용자 지정 이름 도메인 서버를 사용하는 경우 DNS 쿼리의 설정을 확인합니다. 쿼리는 포트 53에 아웃바운드 액세스 권한이 있어야 하며 UDP 및 TCP 프로토콜을 사용해야 합니다. 또한 포트 443에 HTTPS 액세스 권한이 있어야 합니다. 자세한 내용은 Amazon Elastic Compute Cloud 사용 설명서에서 보안 그룹 규칙 구성을 참조하세요.
- 서브넷에 네트워크 ACL이 있는 경우 다음 ACL 규칙이 필요합니다.
  - 포트 1024~65535의 트래픽을 허용하는 아웃바운드 규칙.
  - 포트 443의 TCP 트래픽을 허용하는 인바운드 규칙.

규칙을 구성하는 방법에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 네트워크 ACL을 사용하여 서브넷에 대한 트래픽 제어를 참조하세요.

네트워크 문제 해결

`AWSSupport-TroubleshootECSTaskFailedToStart` 런북 결과에 네트워크 문제가 표시되면 다음 구성을 확인합니다.

퍼블릭 서브넷에서 awsvpc 네트워크 모드를 사용하는 작업

런북을 기반으로 다음 구성을 수행합니다.

- 퍼블릭 서브넷에 있는 작업의 경우 태스크를 시작할 때 퍼블릭 IP 자동 할당(Auto-assign public IP)을 활성화됨(ENABLED)으로 지정합니다. 자세한 내용은 애플리케이션을 Amazon ECS 태스크로 실행 섹션을 참조하세요.
- 인터넷 트래픽을 처리하려면 게이트웨이가 필요합니다. 작업 서브넷의 라우팅 테이블에는 게이트웨이로 향하는 트래픽의 경로가 있어야 합니다.

  자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 라우팅 테이블에서 경로 추가 및 제거를 참조하세요.

게이트웨이 유형	라우팅 테이블 대상	라우팅 테이블 대상
NAT	0.0.0.0/0	NAT 게이트웨이 ID
인터넷 게이트웨이	0.0.0.0/0	인터넷 게이트웨이 ID

- 작업 서브넷에 네트워크 ACL이 있는 경우 다음 ACL 규칙이 필요합니다.
  - 포트 1024~65535의 트래픽을 허용하는 아웃바운드 규칙.
  - 포트 443의 TCP 트래픽을 허용하는 인바운드 규칙.

  규칙을 구성하는 방법에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 네트워크 ACL을 사용하여 서브넷에 대한 트래픽 제어를 참조하세요.

프라이빗 서브넷에서 awsvpc 네트워크 모드를 사용하는 작업

런북을 기반으로 다음 구성을 수행합니다.

- 작업을 시작할 때 퍼블릭 IP 자동 할당을 수행하려면 비활성화됨을 선택합니다.
- 요청을 인터넷으로 라우팅하도록 VPC에서 NAT 게이트웨이를 구성합니다. 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 NAT 게이트웨이를 참조하세요.
- 작업 서브넷의 라우팅 테이블에는 NAT 게이트웨이로 향하는 트래픽의 경로가 있어야 합니다.

  자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 라우팅 테이블에서 경로 추가 및 제거를 참조하세요.

게이트웨이 유형	라우팅 테이블 대상	라우팅 테이블 대상
NAT	0.0.0.0/0	NAT 게이트웨이 ID

- 작업 서브넷에 네트워크 ACL이 있는 경우 다음 ACL 규칙이 필요합니다.
  - 포트 1024~65535의 트래픽을 허용하는 아웃바운드 규칙.
  - 포트 443의 TCP 트래픽을 허용하는 인바운드 규칙.

  규칙을 구성하는 방법에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 네트워크 ACL을 사용하여 서브넷에 대한 트래픽 제어를 참조하세요.

퍼블릭 서브넷에서 awsvpc 네트워크 모드를 사용하지 않는 작업

런북을 기반으로 다음 구성을 수행합니다.

- 클러스터를 생성할 때 Amazon EC2 인스턴스를 위한 네트워킹 아래 IP 자동 할당에서 켜기를 선택합니다.

  이 옵션은 인스턴스의 기본 네트워크 인터페이스에 퍼블릭 IP 주소를 할당합니다.
- 인터넷 트래픽을 처리하려면 게이트웨이가 필요합니다. 인스턴스 서브넷의 라우팅 테이블에는 게이트웨이로 향하는 트래픽의 경로가 있어야 합니다.

  자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 라우팅 테이블에서 경로 추가 및 제거를 참조하세요.

게이트웨이 유형	라우팅 테이블 대상	라우팅 테이블 대상
NAT	0.0.0.0/0	NAT 게이트웨이 ID

게이트웨이 유형	라우팅 테이블 대상	라우팅 테이블 대상
인터넷 게이트웨이	0.0.0.0/0	인터넷 게이트웨이 ID

- 인스턴스 서브넷에 네트워크 ACL이 있는 경우 다음 ACL 규칙이 필요합니다.
  - 포트 1024~65535의 트래픽을 허용하는 아웃바운드 규칙.
  - 포트 443의 TCP 트래픽을 허용하는 인바운드 규칙.

  규칙을 구성하는 방법에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 네트워크 ACL을 사용하여 서브넷에 대한 트래픽 제어를 참조하세요.

프라이빗 서브넷에서 awsvpc 네트워크 모드를 사용하지 않는 태스크

런북을 기반으로 다음 구성을 수행합니다.

- 클러스터를 생성할 때 Amazon EC2 인스턴스를 위한 네트워킹 아래 IP 자동 할당에서 끄기를 선택합니다.
- 요청을 인터넷으로 라우팅하도록 VPC에서 NAT 게이트웨이를 구성합니다. 자세한 정보는 Amazon VPC 사용 설명서의 NAT 게이트웨이 섹션을 참조하세요.
- 인스턴스 서브넷의 라우팅 테이블에는 NAT 게이트웨이로 향하는 트래픽의 경로가 있어야 합니다.

  자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 라우팅 테이블에서 경로 추가 및 제거를 참조하세요.

게이트웨이 유형	라우팅 테이블 대상	라우팅 테이블 대상
NAT	0.0.0.0/0	NAT 게이트웨이 ID

- 작업 서브넷에 네트워크 ACL이 있는 경우 다음 ACL 규칙이 필요합니다.
  - 포트 1024~65535의 트래픽을 허용하는 아웃바운드 규칙.
  - 포트 443의 TCP 트래픽을 허용하는 인바운드 규칙.

  규칙을 구성하는 방법에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 네트워크 ACL을 사용하여 서브넷에 대한 트래픽 제어를 참조하세요.

Amazon ECS 태스크에 대한 IAM 역할 요청 보기

IAM 역할에서 태스크 자격 증명으로 공급자를 사용하면 공급자 요청이 감사 로그에 저장됩니다. 감사 로그는 컨테이너 에이전트 로그와 동일한 로그 교체 설정을 상속합니다. ECS_LOG_ROLLOVER_TYPE, ECS_LOG_MAX_FILE_SIZE_MB 및 ECS_LOG_MAX_ROLL_COUNT 컨테이너 에이전트 구성 변수는 감사 로그의 동작에 영향을 미치도록 설정할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 로그 구성 파라미터 섹션을 참조하세요.

컨테이너 에이전트 버전 1.36.0 이상에서는 감사 로그가 /var/log/ecs/audit.log에 있습니다. 로그가 교체되면 *YYYY*-*MM*-*DD*-*HH* 형식의 타임스탬프가 로그 파일 이름의 끝에 추가됩니다.

컨테이너 에이전트 버전 1.35.0 및 이전 버전의 경우 감사 로그는 /var/log/ecs/audit.log.*YYYY*-*MM*-*DD*-*HH*에 있습니다.

로그 항목 형식은 다음과 같습니다.

- Timestamp
- HTTP 응답 코드
- 요청 오리진의 IP 주소와 포트 번호
- 자격 증명 공급자의 상대 URI
- 요청을 한 사용자 에이전트
- 요청 컨테이너가 속한 작업의 ARN
- GetCredentials API 이름 및 버전 번호
- 컨테이너 인스턴스가 등록된 Amazon ECS 클러스터의 이름
- 컨테이너 인스턴스 ARN

다음 명령을 사용하여 로그 파일을 볼 수 있습니다.

```
cat /var/log/ecs/audit.log.2016-07-13-16
```

출력:

```
2016-07-13T16:11:53Z 200 172.17.0.5:52444 "/v1/credentials" "python-requests/2.7.0
 CPython/2.7.6 Linux/4.4.14-24.50.amzn1.x86_64" TASK_ARN GetCredentials
 1 CLUSTER_NAME CONTAINER_INSTANCE_ARN
```

Amazon ECS 서비스 이벤트 메시지 보기

서비스 문제를 해결하려는 경우, 제일 먼저 진단 정보를 확인할 곳은 서비스 이벤트 로그입니다. `DescribeServices` API, AWS CLI 또는 AWS Management Console을 사용하여 서비스 이벤트를 볼 수 있습니다.

Amazon ECS API를 사용하여 서비스 이벤트 메시지를 보는 경우 서비스 스케줄러의 이벤트만 반환됩니다. 여기에는 가장 최근의 작업 배치 및 인스턴스 상태 이벤트가 포함됩니다. 그러나 Amazon ECS 콘솔에는 다음 소스의 서비스 이벤트가 표시됩니다.

- Amazon ECS 서비스 스케줄러의 작업 배치 및 인스턴스 상태 이벤트. 이러한 이벤트의 접두사는 service ***(service-name)***입니다. 이 이벤트 보기가 도움이 되도록 `100`개의 최신 이벤트만 표시합니다. 원인이 해결되거나 6시간이 경과할 때까지 중복 이벤트 메시지가 생략됩니다. 6시간 이내에 원인이 해결되지 않으면 해당 원인에 대한 다른 서비스 이벤트 메시지를 수신합니다.
- 서비스 Auto Scaling 이벤트. 이러한 이벤트는 Message 접두사가 있고 서비스가 Application Auto Scaling 오토 스케일링 정책을 사용하여 구성된 경우에만 발생합니다.

다음 단계에 따라 현재 서비스 이벤트 메시지를 확인합니다.

Console

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터(Clusters) 페이지에서 클러스터를 선택합니다.
4. 검사할 서비스를 선택합니다.
5. Events(이벤트)에서 Deployments and events(배포 및 이벤트)를 선택하고 메시지를 봅니다.

AWS CLI

describe-services 명령을 사용하여 지정된 서비스에 대한 서비스 이벤트 메시지를 확인합니다.

다음 AWS CLI 예제에서 설명하는 *##* 클러스터의 *service-name* 서비스는 최신 서비스 이벤트 메시지를 제공합니다.

```
aws ecs describe-services \
    --cluster default \
    --services service-name \
```

```
    --region us-west-2
```

Amazon ECS 서비스 이벤트 메시지

다음은 Amazon ECS 콘솔에서 볼 수 있는 서비스 이벤트 메시지의 예제입니다.

서비스(*service-name*)가 안정적인 상태에 도달했습니다.

서비스 스케줄러는 `service (service-name) has reached a steady state.` 서비스가 정상이고 원하는 수의 작업에 도달하면 서비스 이벤트를 보내 안정적인 상태에 도달합니다.

서비스 스케줄러는 주기적으로 상태를 보고하므로 이 메시지를 여러 번 받을 수 있습니다.

모든 요구 사항을 충족하는 컨테이너 인스턴스가 없기 때문에 서비스(*service-name*)가 태스크를 배치할 수 없습니다.

서비스 스케줄러는 다른 작업을 추가하는 데 사용 가능한 리소스를 찾을 수 없을 때 이 이벤트 메시지를 보냅니다. 가능한 원인은 다음과 같습니다.

용량 공급자를 사용하여 자동으로 EC2 인스턴스 규모를 조정합니다. 자세한 내용은 EC2 시작 유형을 위한 Amazon ECS 용량 공급자 섹션을 참조하세요.

> 용량 공급자를 사용하려는 경우 용량 공급자 전략을 전달하거나 클러스터와 연결된 기본 용량 공급자 전략이 있고 시작 유형 및 용량 공급자 전략을 입력으로 전달하지 않아야 합니다.

클러스터에서 컨테이너 인스턴스를 찾을 수 없음

> 작업을 실행하려는 클러스터에 등록된 컨테이너 인스턴스가 없는 경우 이 오류가 발생합니다. 클러스터에 컨테이너 인스턴스를 추가해야 합니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 시작 섹션을 참조하세요.

포트가 충분하지 않음

> 작업에서 고정된 호스트 포트 매핑을 사용하는 경우(예를 들어 작업이 웹 서버 호스트에서 포트 80을 사용하는 경우), 하나의 컨테이너만 한 번에 하나의 호스트 포트를 사용할 수 있기 때문에 작업당 하나 이상의 컨테이너 인스턴스가 있어야 합니다. 클러스터에 컨테이너 인스턴스를 추가하거나 원하는 작업 수를 줄여야 합니다.

너무 많은 포트가 등록되었습니다.

작업 배치에 가장 근접하게 일치하는 컨테이너 인스턴스는 컨테이너 인스턴스당 허용되는 최대 예약 포트 제한인 100개의 호스트 포트를 초과할 수 없습니다. 동적 호스트 포트 매핑을 사용하면 문제가 해결될 수 있습니다.

포트 이미 사용 중

이 작업의 작업 정의는 선택된 컨테이너 인스턴스에서 이미 실행 중인 작업과 동일한 포트를 포트 매핑에 사용합니다. 서비스 이벤트 메시지에는 아래 메시지의 일부로 선택된 컨테이너 인스턴스 ID가 있습니다.

```
The closest matching container-instance is already using a port required by your
 task.
```

메모리가 충분하지 않음

태스크 정의에서 지정된 메모리가 1000MiB이고 클러스터의 컨테이너 인스턴스 각각의 메모리가 1024MiB이라면 컨테이너 인스턴스당 이 작업의 사본 하나만 실행할 수 있습니다. 컨테이너 인스턴스당 2개 이상의 태스크를 시작하거나 보다 많은 컨테이너 인스턴스를 클러스터로 시작할 수 있도록 태스크 정의에서 이보다 적은 메모리를 실험할 수 있습니다.

> **Note**
>
> 특정 인스턴스 유형에 대해 태스크에 가능한 한 많은 메모리를 제공하여 리소스 사용률을 최대화하려는 경우 Amazon ECS Linux 컨테이너 인스턴스 메모리 예약 섹션을 참조하세요.

CPU가 충분하지 않음

하나의 컨테이너 인스턴스에는 CPU 코어마다 1,024개의 CPU 단위가 있습니다. 태스크 정의에서 지정된 CPU 단위가 1,000이고 클러스터의 컨테이너 인스턴스 각각의 CPU 단위가 1,024라면 컨테이너 인스턴스당 이 작업의 사본 하나만 실행할 수 있습니다. 컨테이너 인스턴스당 2개 이상의 태스크를 시작하거나 보다 많은 컨테이너 인스턴스를 클러스터로 시작할 수 있도록 태스크 정의에서 이보다 적은 CPU 단위를 시험해 볼 수 있습니다.

사용 가능한 ENI 연결 지점이 충분하지 않음

awsvpc 네트워크 모드를 사용하는 태스크는 각각 고유한 탄력적 네트워크 인터페이스(ENI)를 받습니다. 이 ENI는 이를 호스팅하는 컨테이너 인스턴스에 연결되어 있습니다. Amazon EC2 인스턴

스는 연결할 수 있는 ENI 개수에 대한 제한이 있고 클러스터에 사용 가능한 ENI 용량이 있는 컨테이너 인스턴스가 없습니다.

개별 컨테이너 인스턴스에 대한 ENI 제한은 다음 조건에 따라 다릅니다.

- awsvpcTrunking 계정 설정에 옵트인하지 않은 경우 각 컨테이너 인스턴스에 대한 ENI 제한은 인스턴스 유형에 따라 다릅니다. 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 유형별/네트워크 인터페이스당 IP 주소를 참조하세요.
- awsvpcTrunking 계정 설정에 옵트인했지만 옵트인 후 지원되는 인스턴스 유형을 사용하여 새로운 컨테이너 인스턴스를 시작하지 않은 경우 각 컨테이너 인스턴스에 대한 ENI 제한은 여전히 기본값으로 설정됩니다. 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 유형별/네트워크 인터페이스당 IP 주소를 참조하세요.
- awsvpcTrunking 계정 설정에 옵트인했으며 옵트인 후 지원되는 인스턴스 유형을 사용하여 새 컨테이너 인스턴스를 시작한 경우 추가 ENI를 사용할 수 있습니다. 자세한 내용은 증가한 Amazon ECS 컨테이너 네트워크 인터페이스에 대해 지원되는 인스턴스 섹션을 참조하세요.

awsvpcTrunking 계정 설정 옵트인에 대한 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 네트워크 인터페이스 증가 섹션을 참조하세요.

클러스터에 컨테이너 인스턴스를 추가하여 가용 네트워크 어댑터를 더 많이 제공할 수 있습니다.

필요한 속성이 누락된 컨테이너 인스턴스

일부 태스크 정의 파라미터의 경우, 특정 도커 원격 API 버전을 컨테이너 인스턴스에 설치해야 합니다. 로깅 드라이버 옵션 등 다른 태스크 정의 파라미터의 경우, 컨테이너 인스턴스가 ECS_AVAILABLE_LOGGING_DRIVERS 에이전트 구성 변수를 사용하여 이러한 로깅 드라이버를 등록해야 합니다. 태스크 정의에 특정 컨테이너 인스턴스 속성을 필요로 하는 파라미터가 포함되어 있고 이 요구 사항을 충족할 수 있는 사용 가능한 컨테이너 인스턴스가 없는 경우, 해당 작업을 배치할 수 없습니다.

서비스에서 awsvpc 네트워크 모드 및 EC2 시작 유형을 사용하는 작업을 사용하고 있는 경우가 이 오류의 일반적인 원인입니다. 서비스를 생성할 때 awsvpcConfiguration에 지정된 동일한 서브넷에서 지정한 클러스터에 컨테이너 인스턴스가 등록되어 있지 않습니다.

AWSSupport-TroubleshootECSContainerInstance 런북을 사용하여 문제를 해결할 수 있습니다. 런북에서는 인스턴스의 사용자 데이터에 올바른 클러스터 정보가 포함되어 있는지 여부, 인스턴스 프로파일에 필요한 권한이 포함되어 있는지 여부, 그리고 네트워크 구성 문제를 검토합니다. 자세한 내용은 AWS Systems Manager 자동화 런북 레퍼런스 사용 설명서의 AWSSupport-TroubleshootECSContainerInstance를 참조하세요.

특정 태스크 정의 파라미터와 에이전트 구성 변수에 어떤 속성이 필요한지에 대한 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터 및 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

모든 요구 사항을 충족하는 컨테이너 인스턴스가 없기 때문에 서비스(*service-name*)가 태스크를 배치할 수 없습니다. 가장 가깝게 일치하는 컨테이너 인스턴스 *container-instance-id*에 가용 CPU 단위가 부족합니다.

작업 배치를 위해 가장 가깝게 일치하는 컨테이너 인스턴스에는 CPU 단위가 부족하여 작업 정의의 요구 사항을 충족할 수 없습니다. 태스크 정의의 작업 크기 및 컨테이너 정의 파라미터 모두의 CPU 요구 사항을 검토합니다.

모든 요구 사항을 충족하는 컨테이너 인스턴스가 없기 때문에 서비스(*service-name*)가 태스크를 배치할 수 없습니다. 가장 가깝게 일치하는 컨테이너 인스턴스 *container-instance-id*에 오류 "AGENT"가 발생했습니다.

작업 배치에 가장 가깝게 일치하는 컨테이너 인스턴스 상의 Amazon ECS 컨테이너 에이전트 연결이 끊겼습니다. SSH를 사용해 컨테이너 인스턴스에 연결할 수 있는 경우, 에이전트 로그를 검사할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 로그 구성 파라미터 섹션을 참조하세요. 인스턴스에서 에이전트가 실행 중인지도 확인해야 합니다. Amazon ECS 최적화 AMI를 사용하는 경우 다음 명령을 사용하여 에이전트를 중지하고 다시 시작해 볼 수 있습니다.

- Amazon ECS 최적화 Amazon Linux 2 AMI 및 Amazon ECS 최적화 Amazon Linux 2023 AMI의 경우

```
sudo systemctl restart ecs
```

- Amazon ECS 최적화 Amazon Linux AMI의 경우

```
sudo stop ecs && sudo start ecs
```

(UnhealthyThreshold의 상태 확인 수 이상에서 연속으로 인스턴스가 실패한 이유)로 인해 서비스(*service-name*)(태스크 *task-id*)(인스턴스 *instance-id*)가 (elb *elb-name*)에서 비정상 상태임

이 서비스가 로드 밸런서에 등록되어 있고 로드 밸런서 상태 확인이 실패합니다. 해당 메시지에는 상태 확인에 실패한 특정 태스크를 식별하는 데 도움이 되는 태스크 ID가 포함되어 있습니다. 자세한 내용은 Amazon ECS의 서비스 로드 밸런서 문제 해결 섹션을 참조하세요.

서비스(*service-name*)는 일부 태스크를 시작할 수 없습니다.

이 서비스에는 연속 시도 후 시작에 실패한 작업이 포함되어 있습니다. 이 지점에서 서비스 스케줄러는 재시도 간격을 점차적으로 증가시키기 시작합니다. 작업 시작이 왜 실패하는지 문제를 해결해야 합니다. 자세한 내용은 Amazon ECS 서비스 제한 로직 섹션을 참조하세요.

서비스가 업데이트되면 업데이트된 태스크 정의 등을 사용하여 서비스 스케줄러가 정상적인 동작을 다시 시작합니다.

서비스(*service-name*) 작업이 제한되고 있습니다. 나중에 다시 시도합니다.

이 서비스는 API 조절 제한으로 인해 더 많은 태스크를 시작할 수 없습니다. 서비스 스케줄러가 더 많은 태스크를 시작할 수 있게 되면 재개됩니다.

API 속도 제한 할당량 증가를 요청하려면 AWS Support 센터 페이지를 열고 필요한 경우 로그인한 후 사례 생성(Create case)을 선택합니다. 서비스 한도 증가(Service Limit increase)를 선택합니다. 양식을 작성하고 제출합니다.

서비스 배포 구성으로 인해 배포 중에 서비스(*service-name*)에서 태스크를 중지하거나 시작할 수 없습니다. minimumHealthyPercent 또는 maximumPercent 값을 업데이트하고 다시 시도하세요.

배포 구성으로 인해 서비스 배포 중에 이 서비스에서 태스크를 중지하거나 시작할 수 없습니다. 배포 구성은 `minimumHealthyPercent` 및 `maximumPercent` 값으로 구성되며, 이 값은 서비스가 생성될 때 정의됩니다. 이 값은 기존 서비스에서도 업데이트할 수 있습니다.

`minimumHealthyPercent`는 배포 중 또는 컨테이너 인스턴스가 드레이닝될 때 서비스에 대해 실행되어야 하는 작업 수의 하한을 나타냅니다. 이는 서비스에서 원하는 작업 수의 비율입니다. 이 값은 반올림됩니다. 예를 들어, 최소 정상 상태 백분율이 `50`이고 원하는 작업 수가 4개인 경우 스케줄러는 두 개의 새 작업을 시작하기 전에 두 개의 기존 작업을 중지할 수 있습니다. 마찬가지로 최소 정상 상태 백분율이 75%이고 원하는 작업 수가 2이면 결과 값이 2이기 때문에 스케줄러는 태스크를 중지 할 수 없습니다.

maximumPercent는 배포 중 또는 컨테이너 인스턴스가 드레이닝될 때 서비스에 대해 실행되어야 하는 작업 수의 상한을 나타냅니다. 이는 서비스에서 원하는 작업 수의 비율입니다. 이 값은 내림됩니다. 예를 들어, 최대 백분율이 200이고 원하는 작업 수가 4개인 경우 스케줄러는 4개의 기존 작업을 중지하기 전에 4개의 새 작업을 시작할 수 있습니다. 마찬가지로 최대 백분율이 125이고 원하는 작업 수가 3이면 결과 값이 3이기 때문에 스케줄러는 태스크를 시작할 수 없습니다.

최소 정상 상태 백분율 또는 최대 백분율을 설정할 때는 배포가 트리거될 때 스케줄러가 하나 이상의 태스크를 중지하거나 시작할 수 있는지 확인해야 합니다.

서비스(***service-name***)에서 태스크를 배치할 수 없습니다. 이유: 동시에 실행할 수 있는 작업 수가 제한에 도달했습니다.

오류를 일으킨 리소스에 대한 할당량 증가를 요청할 수 있습니다. 자세한 내용은 Service quotas 섹션을 참조하세요. 할당량 증가를 요청하려면 Service Quotas 사용 설명서의 할당량 증가 요청을 참조하세요.

서비스(***service-name***)에서 태스크를 배치할 수 없습니다. 이유: 내부 오류.

이 오류의 잠재적 원인은 다음과 같습니다.

서브넷이 지원되지 않는 가용 영역에 있기 때문에 서비스가 태스크를 시작할 수 없습니다.

지원되는 Fargate 리전 및 가용 영역에 대한 자세한 내용은 the section called “AWS Fargate 리전” 섹션을 참조하세요.

서브넷 가용 영역을 보는 방법에 대한 자세한 내용은 Amazon VPC 사용 설명서의 서브넷 보기를 참조하세요.

서비스(***service-name***)에서 태스크를 배치할 수 없습니다. 이유: 요청된 CPU 구성이 제한을 초과합니다.

오류를 일으킨 리소스에 대한 할당량 증가를 요청할 수 있습니다. 자세한 내용은 Service quotas 섹션을 참조하세요. 할당량 증가를 요청하려면 Service Quotas 사용 설명서의 할당량 증가 요청을 참조하세요.

서비스(***service-name***)에서 태스크를 배치할 수 없습니다. 이유: 요청된 메모리 구성이 제한을 초과합니다.

오류를 일으킨 리소스에 대한 할당량 증가를 요청할 수 있습니다. 자세한 내용은 Service quotas 섹션을 참조하세요. 할당량 증가를 요청하려면 Service Quotas 사용 설명서의 할당량 증가 요청을 참조하세요.

서비스(*service-name*)에서 태스크를 배치할 수 없습니다. 이유: 동시에 실행할 수 있는 vCPU 수가 제한에 도달했습니다.

AWS Fargate은 태스크 수 기반 할당량에서 vCPU 기반 할당량으로 전환하고 있습니다.

Fargate vCPU 기반 할당량에 대한 할당량 증가를 요청할 수 있습니다. 자세한 내용은 Service quotas 섹션을 참조하세요. Fargate 할당량 증가을 요청하려면 Service Quotas 사용 설명서의 할당량 증가 요청을 참조하세요.

작업 세트(*taskSet-ID*)를 스케일 인할 수 없기 때문에 서비스(*service-name*)가 안정 상태에 도달할 수 없습니다. 사유: 보호된 태스크 수가 원하는 태스크 수보다 많음

이 서비스에는 원하는 태스크 수보다 많은 보호된 태스크가 있습니다. 다음 중 하나를 수행할 수 있습니다.

- 현재 작업에 대한 보호가 만료될 때까지 기다리면 작업이 종료됩니다.
- 중지할 수 있는 작업을 결정하고 `protectionEnabled` 옵션을 `false`로 설정한 상태에서 `UpdateTaskProtection` API를 사용하여 이러한 작업에 대한 보호를 해제합니다.
- 서비스의 바람직한 작업 수를 보호되는 작업 수 이상으로 늘립니다.

서비스(*service-name*)가 안정 상태에 도달할 수 없습니다. 원인: 용량 공급자에서 컨테이너 인스턴스를 찾을 수 없습니다.

서비스 스케줄러는 다른 작업을 추가하는 데 사용 가능한 리소스를 찾을 수 없을 때 이 이벤트 메시지를 보냅니다. 가능한 원인은 다음과 같습니다.

클러스터에 연결된 용량 공급자가 없음

`describe-services`를 사용하여 클러스터와 연결된 용량 공급자가 있는지 확인합니다. 서비스에 대한 용량 공급자 전략을 업데이트할 수 있습니다.

용량 공급자에서 사용 가능한 용량이 있는지 확인합니다. EC2 시작 유형의 경우 컨테이너 인스턴스가 작업 정의 요구 사항을 충족하는지 확인합니다.

클러스터에서 컨테이너 인스턴스를 찾을 수 없음

작업을 실행하려는 클러스터에 등록된 컨테이너 인스턴스가 없는 경우 이 오류가 발생합니다. 클러스터에 컨테이너 인스턴스를 추가해야 합니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 시작 섹션을 참조하세요.

포트가 충분하지 않음

작업에서 고정된 호스트 포트 매핑을 사용하는 경우(예를 들어 작업이 웹 서버 호스트에서 포트 80을 사용하는 경우) 작업당 하나 이상의 컨테이너 인스턴스가 있어야 합니다. 한 번에 하나의 컨테이너만 단일 호스트 포트를 사용할 수 있습니다. 클러스터에 컨테이너 인스턴스를 추가하거나 원하는 작업 수를 줄여야 합니다.

너무 많은 포트가 등록되었습니다.

작업 배치에 가장 근접하게 일치하는 컨테이너 인스턴스는 컨테이너 인스턴스당 허용되는 최대 예약 포트 제한인 100개의 호스트 포트를 초과할 수 없습니다. 동적 호스트 포트 매핑을 사용하면 문제가 해결될 수 있습니다.

포트 이미 사용 중

이 작업의 작업 정의는 선택된 컨테이너 인스턴스에서 이미 실행 중인 작업과 동일한 포트를 포트 매핑에 사용합니다. 서비스 이벤트 메시지에는 아래 메시지의 일부로 선택된 컨테이너 인스턴스 ID가 있습니다.

```
The closest matching container-instance is already using a port required by your
 task.
```

메모리가 충분하지 않음

태스크 정의에서 지정된 메모리가 1000MiB이고 클러스터의 컨테이너 인스턴스 각각의 메모리가 1024MiB이라면 컨테이너 인스턴스당 이 작업의 사본 하나만 실행할 수 있습니다. 컨테이너 인스턴스당 2개 이상의 태스크를 시작하거나 보다 많은 컨테이너 인스턴스를 클러스터로 시작할 수 있도록 태스크 정의에서 이보다 적은 메모리를 실험할 수 있습니다.

> **Note**
>
> 특정 인스턴스 유형에 대해 작업에 가능한 한 많은 메모리를 제공하여 리소스 사용률을 최대화하려는 경우 Amazon ECS Linux 컨테이너 인스턴스 메모리 예약 섹션을 참조하세요.

사용 가능한 ENI 연결 지점이 충분하지 않음

awsvpc 네트워크 모드를 사용하는 태스크는 각각 고유한 탄력적 네트워크 인터페이스(ENI)를 받습니다. 이 ENI는 이를 호스팅하는 컨테이너 인스턴스에 연결되어 있습니다. Amazon EC2 인스턴스는 연결할 수 있는 ENI 개수에 대한 제한이 있고 클러스터에 사용 가능한 ENI 용량이 있는 컨테이너 인스턴스가 없습니다.

개별 컨테이너 인스턴스에 대한 ENI 제한은 다음 조건에 따라 다릅니다.

- awsvpcTrunking 계정 설정에 옵트인하지 않은 경우 각 컨테이너 인스턴스에 대한 ENI 제한은 인스턴스 유형에 따라 다릅니다. 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 유형별/네트워크 인터페이스당 IP 주소를 참조하세요.
- awsvpcTrunking 계정 설정에 옵트인했지만 옵트인 후 지원되는 인스턴스 유형을 사용하여 새로운 컨테이너 인스턴스를 시작하지 않은 경우 각 컨테이너 인스턴스에 대한 ENI 제한은 여전히 기본값으로 설정됩니다. 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 유형별/네트워크 인터페이스당 IP 주소를 참조하세요.
- awsvpcTrunking 계정 설정에 옵트인했으며 옵트인 후 지원되는 인스턴스 유형을 사용하여 새 컨테이너 인스턴스를 시작한 경우 추가 ENI를 사용할 수 있습니다. 자세한 내용은 증가한 Amazon ECS 컨테이너 네트워크 인터페이스에 대해 지원되는 인스턴스 섹션을 참조하세요.

awsvpcTrunking 계정 설정 옵트인에 대한 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 네트워크 인터페이스 증가 섹션을 참조하세요.

클러스터에 컨테이너 인스턴스를 추가하여 가용 네트워크 어댑터를 더 많이 제공할 수 있습니다.

필요한 속성이 누락된 컨테이너 인스턴스

일부 태스크 정의 파라미터의 경우, 특정 도커 원격 API 버전을 컨테이너 인스턴스에 설치해야 합니다. 로깅 드라이버 옵션 등 다른 태스크 정의 파라미터의 경우, 컨테이너 인스턴스가 ECS_AVAILABLE_LOGGING_DRIVERS 에이전트 구성 변수를 사용하여 이러한 로깅 드라이버를 등록해야 합니다. 태스크 정의에 특정 컨테이너 인스턴스 속성을 필요로 하는 파라미터가 포함되어 있고 이 요구 사항을 충족할 수 있는 사용 가능한 컨테이너 인스턴스가 없는 경우, 해당 작업을 배치할 수 없습니다.

이 오류의 일반적인 원인은 서비스에서 awsvpc 네트워크 모드 및 EC2 시작 유형을 사용하는 작업을 사용하고 지정한 클러스터에 서비스를 생성할 때 awsvpcConfiguration에서 지정한 것과 동일한 서브넷에 컨테이너 인스턴스가 등록되어 있지 않은 경우입니다.

AWSSupport-TroubleshootECSContainerInstance 런북을 사용하여 문제를 해결할 수 있습니다. 런북에서는 인스턴스의 사용자 데이터에 올바른 클러스터 정보가 포함되어 있는지 여부, 인스턴스 프로파일에 필요한 권한이 포함되어 있는지 여부, 그리고 네트워크 구성 문제를 검토합니다. 자세한 내용은 AWS Systems Manager 자동화 런북 레퍼런스 사용 설명서의 AWSSupport-TroubleshootECSContainerInstance를 참조하세요.

특정 태스크 정의 파라미터와 에이전트 구성 변수에 어떤 속성이 필요한지에 대한 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터 및 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

서비스(*service-name*)에서 태스크를 배치할 수 없습니다. 원인: 지금은 용량을 사용할 수 없습니다. 나중에 다시 시도하거나 다른 가용 영역에서 시도하세요.

현재 서비스를 실행할 수 있는 가용 용량이 없습니다.

다음 중 하나를 수행할 수 있습니다.

- Fargate 용량 또는 EC2 컨테이너 인스턴스를 사용할 수 있을 때까지 기다리세요.
- 서비스를 다시 시작하고 추가 서브넷을 지정하세요.

서비스(*service-name*) 배포 실패: 작업을 시작하지 못했습니다.

서비스의 작업을 시작하지 못했습니다.

중지된 작업을 디버깅하는 방법에 대한 자세한 내용은 Amazon ECS 중지된 작업 오류 메시지 섹션을 참조하세요.

서비스(*service-name*) Amazon ECS Agent가 시작되길 기다리는 중 제한 시간을 초과했습니다. /var/log/ecs/ecs-agent.log에서 로그를 확인하세요.

작업 배치에 가장 가깝게 일치하는 컨테이너 인스턴스 상의 Amazon ECS 컨테이너 에이전트 연결이 끊겼습니다. SSH를 사용해 컨테이너 인스턴스에 연결할 수 있는 경우, 에이전트 로그를 검사할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 로그 구성 파라미터 섹션을 참조하세요. 인스턴스에서 에이전트가 실행 중인지도 확인해야 합니다. Amazon ECS 최적화 AMI를 사용하는 경우 다음 명령을 사용하여 에이전트를 중지하고 다시 시작해 볼 수 있습니다.

- Amazon ECS 최적화 Amazon Linux 2 AMI의 경우

```
sudo systemctl restart ecs
```

- Amazon ECS 최적화 Amazon Linux AMI의 경우

```
sudo stop ecs && sudo start ecs
```

TARGET GROUP IS NOT FOUND로 인해 서비스(*service-name*) 태스크 세트(*taskSet-ID*)(태스크 *task-id*)가 대상 그룹(*targetGroup-ARN)*)에서 비정상 상태입니다.

대상 그룹을 찾을 수 없으므로 서비스에 설정된 작업의 상태 확인에 실패합니다. 해당 메시지에는 상태 확인에 실패한 특정 태스크를 식별하는 데 도움이 되는 태스크 ID가 포함되어 있습니다. 서비스를 삭제하고 다시 생성해야 합니다. 해당 Amazon ECS 서비스가 이미 삭제된 경우가 아니면 Elastic Load Balancing 대상 그룹을 삭제하지 않습니다.

TARGET IS NOT FOUND로 인해 서비스(*service-name*) 태스크 세트(*taskSet-ID*)(태스크 *task-id*)가 대상 그룹(*targetGroup-ARN)*)에서 비정상 상태입니다.

대상을 찾을 수 없으므로 서비스에 설정된 작업의 상태 확인에 실패합니다. 해당 메시지에는 상태 확인에 실패한 특정 태스크를 식별하는 데 도움이 되는 태스크 ID가 포함되어 있습니다.

Amazon ECS 서비스 비정상 이벤트 메시지

다음은 서비스 비정상 이벤트 메시지의 예제입니다.

EC2: (이유 *failure-reason*)로 인해 (서비스 *service-name*)(태스크 *task-id*)(인스턴스 *instance-id*)(포트 *port-number*)가 (대상 그룹 *target-group-name*)에서 비정상 상태임

이 메시지는 EC2 인스턴스에서 실행 중인 태스크가 상태 확인에 실패했음을 나타냅니다. 자세한 내용은 다음 자료를 참조하세요.

- Amazon EC2 시작 유형을 사용하여 Application Load Balancer 상태 확인을 전달하는 Amazon ECS 태스크를 가져오려면 어떻게 해야 합니까?

태스크 세트가 있는 EC2: (이유 *failure-reason*)로 인해 (서비스 *service-name*, 태스크 세트 *taskSet-id*)(태스크 *task-id*)(인스턴스 *instance-id*)(포트 *port-number*)가 (대상 그룹 *target-group-name*)에서 비정상 상태임

이 메시지는 EC2 인스턴스에서 실행 중인 태스크 세트의 태스크가 상태 확인에 실패했음을 나타냅니다. 자세한 내용은 다음 자료를 참조하세요.

- Amazon EC2 시작 유형을 사용하여 Application Load Balancer 상태 확인을 전달하는 Amazon ECS 태스크를 가져오려면 어떻게 해야 합니까?

Fargate: (이유 *failure-reason*)로 인해 (서비스 *service-name*)(태스크 *task-id*)(포트 *port-number*)가 (대상 그룹 *target-group-name*)에서 비정상 상태임

이 메시지는 Fargate 태스크가 상태 확인에 실패했음을 나타냅니다.

Fargate 태스크의 상태 확인 실패 문제 해결에 대한 자세한 내용은 Fargate의 Amazon ECS 태스크에 대한 상태 확인 실패 문제를 해결하려면 어떻게 해야 합니까?를 참조하세요.

태스크 세트가 있는 Fargate: (이유 *failure-reason*)로 인해 (서비스 *service-name*, 태스크 세트 *taskSet-id*)(태스크 *task-id*)(포트 *port-number*)가 (대상 그룹 *target-group-name*)에서 비정상 상태임

이 메시지는 Fargate에서 실행되는 태스크 세트의 태스크가 상태 확인에 실패했음을 나타냅니다.

Fargate 태스크의 상태 확인 실패 문제 해결에 대한 자세한 내용은 Fargate의 Amazon ECS 태스크에 대한 상태 확인 실패 문제를 해결하려면 어떻게 해야 합니까?를 참조하세요.

Amazon ECS 가용 영역 서비스 리밸런싱 서비스 이벤트 메시지

다음은 참조할 수 있는 서비스 이벤트 메시지 예제입니다.

서비스(*service-name*)는 *## ## 1*의 *number-tasks* 태스크, *## ## 2*의 *number-tasks* 및 *## ## 3*의 *number-tasks*로 밸런싱된 AZ가 아닙니다. AZ 리밸런싱이 진행 중입니다.

서비스 스케줄러에서는 태스크 수가 가용 영역에 균등하게 분산되지 않을 때 `service (service-name) is not AZ balanced` 서비스 이벤트를 보냅니다. 취할 조치는 없습니다. 이는 정보 이벤트입니다.

서비스(*service-name*)는 *## ## 1*의 *number-tasks* 태스크, *## ## 2*의 *number-tasks* 태스크 및 *## ## 3*의 *number-tasks* 태스크로 밸런싱된 AZ입니다.

서비스 스케줄러에서는 가용 영역 서비스 리밸런싱이 완료되면 `service (service-name) is AZ balanced` 서비스 이벤트를 보냅니다. 취할 조치는 없습니다. 이는 정보 이벤트입니다.

*service-name*이 AZ 리밸런싱에 대한 *## ##*의 *number-tasks* 태스크를 시작했습니다(*task-ids*).

서비스 스케줄러에서는 서비스 리밸런싱 때문에 가용 영역에서 태스크가 시작될 때 *## ##* 서비스 이벤트에서 *number* 태스크를 시작한 *service-name*/*task-set-name*을 보냅니다. 취할 조치는 없습니다. 이는 정보 이벤트입니다.

*service-name*이 AZ 리밸런싱으로 인해 *## ##*에서 실행 중인 태스크 *number-tasks*를 중지했습니다(*task-id*).

서비스 스케줄러에서는 서비스 리밸런싱 때문에 가용 영역에서 태스크가 중지될 때 *## ##* 서비스 이벤트에서 *number*개 태스크를 중지한 *service-name*/*task-set-name*을 보냅니다. 취할 조치는 없습니다. 이는 정보 이벤트입니다.

모든 요구 사항을 충족하는 컨테이너 인스턴스가 없기 때문에 서비스(*service-name*)가 태스크를 *## ##*에 배치할 수 없습니다.

서비스 스케줄러에서는 모든 요구 사항을 충족하는 컨테이너 인스턴스가 없기 때문에 *service-name*에서 태스크를 *## ##* 배치할 수 없다는 서비스 이벤트를 보냅니다. 문제를 해결하려면 가용 영역에서 인스턴스를 시작합니다.

서비스(*service-name*)가 태스크를 *## ##*에 배치할 수 없습니다.

Fargate 시작 유형을 사용하고 사용 가능한 용량이 없으면 서비스 스케줄러에서는 *service-name*에서 *## ##*에 태스크를 배치할 수 없다는 서비스 이벤트를 보냅니다.

오류 메시지의 가용 영역에 추가 서브넷을 추가하거나 지원에 문의하여 추가 용량을 얻을 수 있습니다.

reason 사유로 인해 *task-set-name* 태스크가 스케일 인될 수 없어서 서비스(*service-name*)가 AZ 리밸런싱할 수 없었습니다.

서비스 스케줄러에서는 태스크 스케일 인 보호 사용 시 서비스 스케줄러에서는 *reason*(으)로 인해 *task-set-name*이(가) 스케일 인될 수 없어서 *service-name*이(가) AZ 리밸런싱할 수 없다는 서비스 이벤트를 보냅니다.

다음 중 하나를 수행할 수 있습니다.

- 현재 작업에 대한 보호가 만료될 때까지 기다리면 작업이 종료됩니다.
- 중지할 수 있는 작업을 결정하고 `protectionEnabled` 옵션을 `false`로 설정한 상태에서 `UpdateTaskProtection` API를 사용하여 이러한 작업에 대한 보호를 해제합니다.

- 서비스의 바람직한 작업 수를 보호되는 작업 수 이상으로 늘립니다.

서비스(*service-name*)가 AZ 리밸런싱을 중지했습니다.

가용 영역 리밸런싱 작업이 중지되면 서비스 스케줄러에서는 *service-name* 서비스가 AZ 리밸런싱을 중지했다는 서비스 이벤트를 보냅니다. 이는 정보 이벤트입니다. Amazon ECS에서는 자세한 내용이 제공되는 추가 이벤트를 보냅니다.

Amazon ECS의 서비스 로드 밸런서 문제 해결

Amazon ECS 서비스는 Elastic Load Balancing 로드 밸런서에 태스크를 등록할 수 있습니다. 로드 밸런서 구성 오류는 중지된 작업의 일반적 원인입니다. 중지된 작업이 로드 밸런서를 사용하는 서비스에 의해 시작된 경우, 가능한 다음 원인을 고려하세요.

Amazon ECS 서비스 연결 역할이 없음

Amazon ECS 서비스 연결 역할을 사용하면 Amazon ECS 서비스가 Elastic Load Balancing 로드 밸런서에 컨테이너 인스턴스를 등록할 수 있습니다. 계정에서 서비스 연결 역할이 생성되어야 합니다. 자세한 내용은 Amazon ECS에 대해 서비스 연결 역할 사용 섹션을 참조하세요.

컨테이너 인스턴스 보안 그룹

컨테이너가 컨테이너 인스턴스에서 포트 80에 매핑된 경우, 로드 밸런서 상태 확인에 통과하려면 컨테이너 인스턴스 보안 그룹이 포트 80에서 인바운드 트래픽을 허용해야 합니다.

모든 가용 영역에 대해 Elastic Load Balancing 로드 밸런서가 구성되지 않음

리전 내의 모든 가용 영역 또는 적어도 컨테이너 인스턴스가 상주하는 모든 가용 영역을 사용하도록 로드 밸런서를 구성해야 합니다. 서비스가 로드 밸런서를 사용하고 로드 밸런서가 사용하도록 구성되지 않은 가용 영역에 상주하는 컨테이너 인스턴스에서 작업을 시작하는 경우, 해당 태스크는 절대로 상태 확인을 통과하지 못합니다. 그 결과 작업이 종료됩니다.

Elastic Load Balancing 로드 밸런서 상태 확인이 잘못 구성됨

로드 밸런서 상태 확인 파라미터가 지나치게 제한적이거나 존재하지 않는 리소스를 가리키고 있을 수 있습니다. 컨테이너 인스턴스 상태가 비정상이라고 판단되면 로드 밸런서에서 제거됩니다. 서비스 로드 밸런서에 대해 다음 파라미터가 올바로 구성되어 있는지 확인해야 합니다.

Ping Port

로드 밸런서 상태 확인의 Ping Port 값은 컨테이너 인스턴스 상의 포트로서 정상 여부는 로드 밸런서가 확인합니다. 이 포트가 잘못 구성되면 로드 밸런서가 이 포트에서 컨테이너 인스턴스 등

록을 취소할 수도 있습니다. 이 포트는 상태 확인에 사용하는 서비스의 태스크 정의에서 컨테이너에 `hostPort` 값을 사용하도록 구성되어야 합니다.

Ping Path

로드 밸런서 상태 확인의 일부입니다. 애플리케이션이 정상일 때 성공 상태 코드(예: 200)를 반환할 수 있는 애플리케이션의 엔드포인트입니다. 이 값은 흔히 `index.html`로 설정되지만 서비스가 해당 요청에 응답하지 않는 경우에는 상태 확인에 실패합니다. 컨테이너에 `index.html` 파일이 없는 경우, 이를 `/`로 설정하여 컨테이너 인스턴스의 기본 URL을 대상으로 지정할 수 있습니다.

Response Timeout

상태 확인 ping에 대해 응답을 반환해야 하는 시간입니다. 이 값이 응답에 필요한 시간보다 낮으면 상태 확인이 실패합니다.

Health Check Interval

상태 확인 ping 사이의 시간입니다. 상태 확인 간격이 짧을수록 컨테이너 인스턴스가 비정상 임계값(Unhealthy Threshold)에 빠르게 도달할 수 있습니다.

Unhealthy Threshold

컨테이너 인스턴스 상태가 비정상이라고 간주될 때까지 허용되는 상태 확인 실패 횟수입니다. 비정상 임계값이 2이고 상태 확인 간격이 30초인 경우, 작업이 60초 동안 상태 확인 ping에 응답하지 않으면 비정상으로 간주됩니다. 비정상 임계값 또는 상태 확인 간격을 늘리면 작업의 응답 시간을 늘릴 수 있습니다.

`servicename` 서비스를 업데이트할 수 없음: 작업 정의에서 로드 밸런서 컨테이너 이름 또는 포트가 변경됨

서비스에서 로드 밸런서를 사용하는 경우 AWS CLI 또는 SDK를 사용하여 로드 밸런서 구성을 수정할 수 있습니다. 구성을 변경하는 방법에 대한 자세한 정보는 Amazon Elastic Container Service API Reference(Amazon Elastic Container Service API 레퍼런스)의 UpdateService를 참조하세요. 서비스에 대한 작업 정의를 업데이트하는 경우, 로드 밸런서 구성에서 지정한 컨테이너 이름 및 컨테이너 포트는 작업 정의에서 그대로 유지해야 합니다.

동시에 실행할 수 있는 작업 수의 제한에 도달했습니다.

새 계정의 경우 할당량이 서비스 할당량보다 낮을 수 있습니다. 계정의 서비스 할당량은 Service Quotas 콘솔에서 볼 수 있습니다. 할당량 증가를 요청하려면 Service Quotas 사용 설명서의 할당량 증가 요청을 참조하세요.

Amazon ECS에서 서비스 Auto Scaling 문제 해결

Application Auto Scaling은 Amazon ECS 배포가 진행 중인 동안 스케일 인 프로세스를 끄고 배포가 완료되면 다시 시작합니다. 그러나 배포 중에 일시 중단되지 않는 한 확장 프로세스는 계속 발생합니다. 자세한 내용은 Application Auto Scaling의 조정 일시 중지 및 재개를 참조하세요.

Amazon ECS 작업 정의 잘못된 CPU 또는 메모리 오류 문제 해결

Amazon ECS API 또는 AWS CLI를 사용하여 태스크 정의를 등록할 때 잘못된 `cpu` 또는 `memory` 값을 지정하면 다음 오류가 반환됩니다.

```
An error occurred (ClientException) when calling the RegisterTaskDefinition operation:
 Invalid 'cpu' setting for task.
```

> **Note**
>
> Terraform을 사용할 때 다음 오류가 반환될 수 있습니다.
>
> ```
> Error: ClientException: No Fargate configuration exists for given values.
> ```

이 문제를 해결하려면 태스크 정의에서 작업 CPU 및 메모리에 지원되는 값을 지정해야 합니다. `cpu` 값은 작업 정의에서 CPU 단위 또는 vCPU로 표현될 수 있습니다. 작업 정의를 등록할 때 CPU 단위를 나타내는 정수로 변환됩니다. `memory` 값은 작업 정의에서 MiB 또는 GB로 표현될 수 있습니다. 태스크 정의를 등록할 때 MiB를 나타내는 정수로 변환됩니다.

`requiresCompatibilities` 파라미터에 `FARGATE`를 지정하는 작업 정의의 경우(`EC2`도 함께 지정되는 경우라도) 다음 테이블에 나오는 값 중 하나를 사용해야 합니다. 이 값에 따라 CPU 및 메모리 파라미터에 대해 지원되는 값의 범위가 결정됩니다.

다음 표에서는 Fargate에서 호스팅되는 태스크에 대해 유효한 CPU와 메모리 조합을 보여줍니다. JSON 파일의 메모리 값은 MiB 단위로 지정됩니다. 값에 1024를 곱하여 GB 값을 MiB로 변환할 수 있습니다. 예를 들어 1GB는 1024MiB입니다.

CPU 값	메모리 값	AWS Fargate에 지원되는 운영 체제
256(.25 vCPU)	512MiB, 1GB, 2GB	Linux

CPU 값	메모리 값	AWS Fargate에 지원되는 운영 체제
512(.5 vCPU)	1GB, 2GB, 3GB, 4GB	Linux
1024(1 vCPU)	2GB, 3GB, 4GB, 5GB, 6GB, 7GB, 8GB	Linux, Windows
2048(2 vCPU)	4~16GB(1GB 증분)	Linux, Windows
4096(4 vCPU)	8~30GB(1GB 증분)	Linux, Windows
8192 (8 vCPU) **Note** 이 옵션은 Linux 플랫폼 `1.4.0` 이상이 필요합니다.	16~60GB(4GB 증분)	Linux
16384 (16vCPU) **Note** 이 옵션은 Linux 플랫폼 `1.4.0` 이상이 필요합니다.	32~120GB(8GB 증분)	Linux

Amazon EC2에서 호스팅되는 작업의 경우 지원되는 작업 CPU 값은 0.25 vCPU에서 192 vCPU 사이입니다.

CPU 제어 메커니즘은 Amazon EC2 시작 유형과 Fargate 시작 유형 간에 다릅니다.

- Amazon EC2에서 호스팅되는 태스크의 경우: Amazon ECS에서는 CPU 기간과 CPU 할당량을 사용하여 태스크 크기 CPU 하드 제한을 제어합니다. 태스크 정의에서 vCPU를 지정하면 Amazon ECS에서는 값을 `cgroup`에 적용되는 CPU 기간 및 CPU 할당량 설정으로 변환합니다.

- Fargate에서 호스팅되는 태스크의 경우: Amazon ECS는 CPU 공유를 사용하여 CPU 할당을 제어합니다. CPU 할당량 및 기간 값은 Fargate 태스크의 CPU 제한에 사용되지 않습니다.

Amazon EC2 태스크의 경우 CPU 할당량은 주어진 CPU 기간 중 cgroup에 부여된 CPU 시간을 제어합니다. 두 가지 설정은 모두 마이크로초 단위로 표시됩니다. CPU 할당량이 CPU 기간과 같으면 cgroup에서는 하나의 vCPU에서 최대 100%(또는 여러 vCPU의 경우 합계가 100%인 다른 모든 비율)까지 실행할 수 있다는 것을 의미합니다. CPU 할당량은 최대 1,000,000us이고 CPU 기간은 최소 1ms입니다. 이러한 값을 사용하여 CPU 수 한도를 설정할 수 있습니다. CPU 할당량을 변경하지 않고 CPU 기간을 변경하면 태스크 정의에 지정한 것과 다른 유효 한도가 있습니다.

100ms 기간에서는 0.125~10 범위의 vCPU가 허용됩니다.

> **Note**
>
> Windows 컨테이너에 대해서는 태스크 레벨 CPU와 메모리 파라미터가 무시됩니다.

Amazon ECS 컨테이너 에이전트 로그 보기

Amazon ECS는 컨테이너 인스턴스의 /var/log/ecs 폴더에 로그를 저장합니다. 컨테이너 인스턴스에는 Amazon ECS 컨테이너 에이전트에서 사용할 수 있는 로그와 에이전트 상태(시작/중지)를 제어하는 ecs-init 서비스에서 사용할 수 있는 로그가 있습니다. 이러한 로그 파일은 SSH를 사용하여 컨테이너 인스턴스에 연결하면 볼 수 있습니다.

> **Note**
>
> 컨테이너 인스턴스에서 모든 로그를 수집하는 방법을 잘 모르는 경우 Amazon ECS 로그 수집기를 사용하면 됩니다. 자세한 내용은 Amazon ECS 로그 수집기를 사용하여 컨테이너 로그 수집 섹션을 참조하세요.

Linux 운영 체제

ecs-init 프로세스는 /var/log/ecs/ecs-init.log에 로그를 저장합니다.

ecs-init.log 파일에는 컨테이너 에이전트 수명 주기 관리, 구성, 부트스트래핑에 대한 정보가 포함되어 있습니다.

다음 명령을 사용하여 로그 파일을 볼 수 있습니다.

```
cat /var/log/ecs/ecs-init.log
```

출력:

```
2018-02-16T18:13:54Z [INFO] pre-start
2018-02-16T18:13:56Z [INFO] start
2018-02-16T18:13:56Z [INFO] No existing agent container to remove.
2018-02-16T18:13:56Z [INFO] Starting Amazon Elastic Container Service Agent
```

Windows 운영 체제

Windows용 Amazon ECS 로그 수집기를 사용할 수 있습니다. 자세한 내용은 Github의 Amazon ECS Logs Collector For Windows를 참조하세요.

1. 인스턴스에 연결합니다.
2. PowerShell을 열고 관리자 권한으로 다음 명령을 실행합니다. 명령은 스크립트를 다운로드하고 로그를 수집합니다.

   ```
   Invoke-WebRequest -OutFile ecs-logs-collector.ps1 https://
   raw.githubusercontent.com/awslabs/aws-ecs-logs-collector-for-windows/master/ecs-
   logs-collector.ps1
   .\ecs-logs-collector.ps1
   ```

Amazon ECS 에이전트 및 Docker 대몬에 대한 디버그 로깅을 켤 수 있습니다. 이 옵션을 사용하면 디버그 모드를 켜기 전에 스크립트에서 로그를 수집할 수 있습니다. 스크립트는 Docker 대몬 및 Amazon ECS 에이전트를 다시 시작한 다음, 인스턴스에서 실행 중인 모든 컨테이너를 종료합니다. 다음 명령을 실행하기 전에 컨테이너 인스턴스를 드레이닝하고 중요한 작업을 다른 컨테이너 인스턴스로 이동합니다.

다음 명령을 실행하여 로깅을 켭니다.

```
.\ecs-logs-collector.ps1 -RunMode debug
```

Amazon ECS 로그 수집기를 사용하여 컨테이너 로그 수집

컨테이너 인스턴스에서 다양한 로그를 모두 수집하는 방법을 잘 모르겠다면 Amazon ECS 로그 수집기를 사용하면 됩니다. 이것은 Linux 및 Windows용 GitHub에서 모두 사용할 수 있습니다. 스크립트

는 일반적인 운영 체제 로그와 AWS Support 사례 문제 해결에 도움이 될 수 있는 Docker 및 Amazon ECS 컨테이너 에이전트 로그를 수집합니다. 그런 다음 수집된 정보를 진단 목적을 위해 쉽게 공유할 수 있는 단일 파일로 압축해 보관합니다. 또한 Amazon ECS 최적화 AMI와 같은 Amazon Linux 변형에서 Docker 대몬과 Amazon ECS 컨테이너 에이전트의 디버그 모드 활성화도 지원합니다. 현재 Amazon ECS 로그 수집기가 지원하는 운영 체제는 다음과 같습니다.

- Amazon Linux
- Red Hat Enterprise Linux 7
- Debian 8
- Ubuntu 14.04
- Ubuntu 16.04
- Ubuntu 18.04
- Windows Server 2016

> **Note**
>
> Amazon ECS 로그 수집기의 소스 코드는 Linux 및 Windows용 GitHub에서 제공됩니다. 포함하고 싶은 변경에 대해서는 풀 요청을 제출할 것을 권장합니다. 하지만 Amazon Web Services는 현재 이 소프트웨어의 변경된 사본을 실행하도록 지원하지 않습니다.

Linux용 Amazon ECS 로그 수집기를 다운로드하고 실행하려면

1. 컨테이너 인스턴스에 연결합니다.
2. Amazon ECS 로그 수집기 스크립트를 다운로드합니다.

```
curl -O https://raw.githubusercontent.com/awslabs/ecs-logs-collector/master/ecs-logs-collector.sh
```

3. 스크립트를 실행해 로그를 수집하고 아카이브를 생성합니다.

> **Note**
>
> Docker 대몬과 Amazon ECS 컨테이너 에이전트에 대해 디버그 모드를 활성화하려면 `--mode=enable-debug` 옵션을 다음 명령에 추가합니다. 인스턴스에서 실행 중인 모든 컨테이너를 종료하는 Docker 대몬이 다시 시작될 수 있습니다. 디버그 모드를 활성화하기 전

에 컨테이너 인스턴스 드레이닝과 다른 컨테이너 인스턴스로의 중요한 작업 이동을 고려합니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 드레이닝 섹션을 참조하세요.

```
[ec2-user ~]$ sudo bash ./ecs-logs-collector.sh
```

Important

로그 파일을 편집하여 민감한 데이터를 모두 제거하는 것이 좋습니다. 파일에서 알려진 데이터를 검색하거나 `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN` 등과 같은 환경 변수를 검색할 수도 있습니다.

스크립트를 실행한 후 스크립트가 생성한 `collect` 폴더에서 수집된 로그를 검사할 수 있습니다. `collect.tgz` 파일은 모든 로그가 압축된 아카이브로서 진단 지원을 위해 AWS Support와 공유할 수 있습니다.

Windows용 Amazon ECS 로그 수집기를 다운로드하고 실행하려면

1. 컨테이너 인스턴스에 연결합니다. 자세한 내용은 Amazon EC2 사용 설명서의 RDP를 사용하여 Windows 인스턴스에 연결을 참조하세요.
2. PowerShell을 사용하여 Amazon ECS 로그 수집기 스크립트를 다운로드합니다.

```
Invoke-WebRequest -OutFile ecs-logs-collector.ps1 https://
raw.githubusercontent.com/awslabs/aws-ecs-logs-collector-for-windows/master/ecs-
logs-collector.ps1
```

3. 스크립트를 실행해 로그를 수집하고 아카이브를 생성합니다.

Note

Docker 대몬과 Amazon ECS 컨테이너 에이전트에 대해 디버그 모드를 활성화하려면 `-RunMode debug` 옵션을 다음 명령에 추가합니다. 인스턴스에서 실행 중인 모든 컨테이너를 종료하는 Docker 대몬이 다시 시작됩니다. 디버그 모드를 활성화하기 전에 컨테이너 인스턴스 드레이닝과 다른 컨테이너 인스턴스로의 중요한 작업 이동을 고려합니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 드레이닝 섹션을 참조하세요.

```
.\ecs-logs-collector.ps1
```

> **Important**
>
> 로그 파일을 편집하여 민감한 데이터를 모두 제거하는 것이 좋습니다. 파일에서 알려진 데이터를 검색하거나 `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN` 등과 같은 환경 변수를 검색할 수도 있습니다.

스크립트를 실행한 후 스크립트가 생성한 `collect` 폴더에서 수집된 로그를 검사할 수 있습니다. `collect.tgz` 파일은 모든 로그가 압축된 아카이브로서 진단 지원을 위해 AWS Support와 공유할 수 있습니다.

에이전트 내부 검사를 통해 Amazon ECS 진단 세부 정보 검색

Amazon ECS 에이전트 내부 검사 API는 Amazon ECS 에이전트 및 컨테이너 인스턴스의 전체 상태에 대한 정보를 제공합니다.

에이전트 내부 검사 API를 사용해 작업에서 컨테이너의 Docker ID를 가져올 수 있습니다. SSH를 사용하여 컨테이너 인스턴스에 연결하면 에이전트 내부 검사 API를 사용할 수 있습니다.

> **Important**
>
> 내부 검사 API에 접근하려면 Amazon ECS에 대한 액세스가 가능한 IAM 역할이 컨테이너 인스턴스에 필요합니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.

다음 예제는 2개의 작업(현재 실행 중인 작업 및 중지된 작업)을 보여줍니다.

> **Note**
>
> 다음 명령은 가독성 향상을 위해 python -mjson.tool을 통해 파이프됩니다.

```
curl http://localhost:51678/v1/tasks | python -mjson.tool
```

출력:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1095  100  1095    0     0   117k      0 --:--:-- --:--:-- --:--:--  133k
{
    "Tasks": [
        {
            "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/090eff9b-1ce3-4db6-848a-
a8d14064fd24",
            "Containers": [
                {
                    "DockerId":
 "189a8ff4b5f04affe40e5160a5ffadca395136eb5faf4950c57963c06f82c76d",
                    "DockerName": "ecs-console-sample-app-static-6-simple-
app-86caf9bcabe3e9c61600",
                    "Name": "simple-app"
                },
                {
                    "DockerId":
 "f7f1f8a7a245c5da83aa92729bd28c6bcb004d1f6a35409e4207e1d34030e966",
                    "DockerName": "ecs-console-sample-app-static-6-busybox-
ce83ce978a87a890ab01",
                    "Name": "busybox"
                }
            ],
            "Family": "console-sample-app-static",
            "KnownStatus": "STOPPED",
            "Version": "6"
        },
        {
            "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/1810e302-eaea-4da9-
a638-097bea534740",
            "Containers": [
                {
                    "DockerId":
 "dc7240fe892ab233dbbcee5044d95e1456c120dba9a6b56ec513da45c38e3aeb",
                    "DockerName": "ecs-console-sample-app-static-6-simple-app-
f0e5859699a7aecfb101",
                    "Name": "simple-app"
                },
```

```
                {
                    "DockerId":
 "096d685fb85a1ff3e021c8254672ab8497e3c13986b9cf005cbae9460b7b901e",
                    "DockerName": "ecs-console-sample-app-static-6-
busybox-92e4b8d0ecd0cce69a01",
                    "Name": "busybox"
                }
            ],
            "DesiredStatus": "RUNNING",
            "Family": "console-sample-app-static",
            "KnownStatus": "RUNNING",
            "Version": "6"
        }
    ]
}
```

위의 예제에서 중지된 작업(*090eff9b-1ce3-4db6-848a-a8d14064fd24*)에는 2개의 컨테이너가 있습니다. docker inspect ***container-ID***를 사용하여 각 컨테이너의 세부 정보를 볼 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 내부 검사 섹션을 참조하세요.

Amazon ECS의 Docker 진단

Docker는 컨테이너 및 작업 문제 해결에 도움이 되는 몇 가지 진단 도구를 제공합니다. 사용 가능한 모든 Docker 명령줄 유틸리티에 대한 자세한 내용은 Docker 설명서의 Docker CLI 참조를 참조하세요. SSH를 사용하여 컨테이너 인스턴스에 연결하면 Docker 명령줄 유틸리티에 액세스할 수 있습니다.

Docker 컨테이너가 보고하는 종료 코드도 일부 진단 정보를 제공할 수 있습니다(예를 들어 종료 코드 137은 컨테이너가 SIGKILL 신호를 수신했음을 뜻합니다). 자세한 내용은 도커 설명서의 종료 상태를 참조하세요.

Amazon ECS에 Docker 컨테이너 나열

컨테이너 인스턴스에서 docker ps 명령을 사용하여 실행 중인 컨테이너를 나열할 수 있습니다. 다음 예제에서는 Amazon ECS 컨테이너 에이전트만 실행 중입니다. 자세한 내용은 도커 설명서의 docker ps를 참조하세요.

```
docker ps
```

출력:

```
CONTAINER ID        IMAGE                            COMMAND             CREATED
      STATUS              PORTS                            NAMES
cee0d6986de0        amazon/amazon-ecs-agent:latest   "/agent"            22 hours ago
      Up 22 hours         127.0.0.1:51678->51678/tcp   ecs-agent
```

docker ps -a 명령을 사용하면 모든 컨테이너를(심지어 중지됐거나 사망한 컨테이너도) 볼 수 있습니다. 이것은 예기치 않게 중지되는 컨테이너를 나열하는 데 유용합니다. 다음 예제에서 컨테이너 f7f1f8a7a245는 9초 전에 종료됐기 때문에 -a 플래그 없이 docker ps 출력에 나타나지 않습니다.

```
docker ps -a
```

출력:

```
CONTAINER ID        IMAGE                                              COMMAND
 CREATED             STATUS                          PORTS                          NAMES
db4d48e411b1        amazon/ecs-emptyvolume-base:autogenerated   "not-applicable"
  19 seconds ago                                                                     ecs-
console-sample-app-static-6-internalecs-emptyvolume-source-c09288a6b0cba8a53700
f7f1f8a7a245        busybox:buildroot-2014.02                   "\"sh -c '/bin/sh -c
  22 hours ago        Exited (137) 9 seconds ago                                     ecs-
console-sample-app-static-6-busybox-ce83ce978a87a890ab01
189a8ff4b5f0        httpd:2                                     "httpd-foreground"
  22 hours ago        Exited (137) 40 seconds ago                                    ecs-
console-sample-app-static-6-simple-app-86caf9bcabe3e9c61600
0c7dca9321e3        amazon/ecs-emptyvolume-base:autogenerated   "not-applicable"
  22 hours ago                                                                       ecs-
console-sample-app-static-6-internalecs-emptyvolume-source-90fefaa68498a8a80700
cee0d6986de0        amazon/amazon-ecs-agent:latest              "/agent"
  22 hours ago        Up 22 hours                 127.0.0.1:51678->51678/tcp   ecs-
agent
```

Amazon ECS에서 Docker 로그 보기

docker logs 명령을 사용하여 컨테이너의 STDOUT 및 STDERR 스트림을 볼 수 있습니다. 이 예제에서 로그는 *dc7240fe892a* 컨테이너에 대해 표시되고 간결성을 위해 head 명령을 통해 파이프됩니다. 자세한 내용은 Docker 설명서의 docker logs를 참조하세요.

> **Note**
>
> 기본 json 로그 드라이버를 사용 중인 경우에는 Docker 로그를 컨테이너 인스턴스에서만 사용할 수 있습니다. awslogs 로그 드라이버를 사용하도록 태스크를 구성한 경우에는 컨테이

너 로그를 CloudWatch Logs에서 사용할 수 있습니다. 자세한 내용은 Amazon ECS 로그를 CloudWatch로 전송 섹션을 참조하세요.

```
docker logs dc7240fe892a | head
```

출력:

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
 using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
 using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
[Thu Apr 23 19:48:36.956682 2015] [mpm_event:notice] [pid 1:tid 140327115417472]
 AH00489: Apache/2.4.12 (Unix) configured -- resuming normal operations
[Thu Apr 23 19:48:36.956827 2015] [core:notice] [pid 1:tid 140327115417472] AH00094:
 Command line: 'httpd -D FOREGROUND'
10.0.1.86 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:29 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.0.154 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.1.86 - - [23/Apr/2015:19:49:58 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:50:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:50:29 +0000] "GET / HTTP/1.1" 200 348
time="2015-04-23T20:11:20Z" level="fatal" msg="write /dev/stdout: broken pipe"
```

Amazon ECS에서 Docker 컨테이너 검사

컨테이너의 Docker ID가 있다면 docker inspect 명령을 사용하여 검사할 수 있습니다. 컨테이너 검사는 컨테이너가 시작된 환경에 대한 가장 상세한 보기를 제공합니다. 자세한 내용은 도커 설명서의 docker inspect를 참조하세요.

```
docker inspect dc7240fe892a
```

출력:

```
[{
    "AppArmorProfile": "",
```

```
    "Args": [],
    "Config": {
        "AttachStderr": false,
        "AttachStdin": false,
        "AttachStdout": false,
        "Cmd": [
            "httpd-foreground"
        ],
        "CpuShares": 10,
        "Cpuset": "",
        "Domainname": "",
        "Entrypoint": null,
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
local/apache2/bin",
            "HTTPD_PREFIX=/usr/local/apache2",
            "HTTPD_VERSION=2.4.12",
            "HTTPD_BZ2_URL=https://www.apache.org/dist/httpd/httpd-2.4.12.tar.bz2"
        ],
        "ExposedPorts": {
            "80/tcp": {}
        },
        "Hostname": "dc7240fe892a",
...
```

Amazon ECS에서 Docker 대몬의 자세한 출력 구성

Docker 컨테이너나 이미지에 문제가 있는 경우, Docker 대몬에서 디버그 모드를 켤 수 있습니다. 디버깅을 사용하면 대몬에서 더 자세한 출력이 제공됩니다. 이를 사용하여 Amazon ECR과 같은 컨테이너 레지스트리에서 전송된 오류 메시지를 검색할 수 있습니다.

> **Important**
>
> 이 절차는 Amazon ECS 최적화 Amazon Linux AMI용으로 작성되었습니다. 다른 운영 체제의 경우 Docker 설명서의 Enable debugging 및 Control and configure Docker with systemd를 참조하세요.

Amazon ECS 최적화 Amazon Linux AMI에서 Docker 대몬 디버그 모드를 사용하려면

1. 컨테이너 인스턴스에 연결합니다.

2. 텍스트 편집기로 Docker 옵션 파일을 엽니다(예: vi). Amazon ECS 최적화 Amazon Linux AMI의 경우 Docker 옵션 파일은 `/etc/sysconfig/docker`에 있습니다.
3. Docker 옵션 문장을 찾아 문자열 따옴표 안에 `-D` 옵션을 추가합니다.

   > **Note**
   >
   > Docker 옵션 구문이 #로 시작하는 경우, 해당 문자를 제거해 문장의 주석 처리를 해제하고 옵션을 활성화해야 합니다.

   Amazon ECS 최적화 Amazon Linux AMI의 경우 Docker 옵션 문장을 일컬어 `OPTIONS`라고 합니다. 예:

   ```
   # Additional startup options for the Docker daemon, for example:
   # OPTIONS="--ip-forward=true --iptables=true"
   # By default we limit the number of open files per container
   OPTIONS="-D --default-ulimit nofile=1024:4096"
   ```

4. 파일을 저장하고 텍스트 편집기를 종료합니다.
5. Docker 대몬을 다시 시작합니다.

   ```
   sudo service docker restart
   ```

   출력값은 다음과 같습니다.

   ```
   Stopping docker:                                           [  OK  ]
   Starting docker: .                                    [  OK  ]
   ```

6. Amazon ECS 에이전트를 다시 시작합니다.

   ```
   sudo service ecs restart
   ```

이제 Docker 로그에 더 상세한 출력이 표시됩니다.

```
time="2015-12-30T21:48:21.907640838Z" level=debug msg="Unexpected response from
 server: \"{\\\"errors\\\":[{\\\"code\\\":\\\"DENIED\\\",\\\"message\\\":\\\"User:
 arn:aws:sts::1111:assumed-role/ecrReadOnly/i-abcdefg is not authorized to perform:
 ecr:InitiateLayerUpload on resource: arn:aws:ecr:us-east-1:1111:repository/nginx_test
```

```
\\\"}]}\\n\" http.Header{\"Connection\":[]string{\"keep-alive\"}, \"Content-Type\":
[]string{\"application/json; charset=utf-8\"}, \"Date\":[]string{\"Wed, 30 Dec 2015
 21:48:21 GMT\"}, \"Docker-Distribution-Api-Version\":[]string{\"registry/2.0\"},
 \"Content-Length\":[]string{\"235\"}}"
```

Amazon ECS의 Docker **API error (500): devmapper** 문제 해결

다음 Docker 오류는 컨테이너 인스턴스의 씬(thin) 풀 스토리지가 꽉 찼으며 Docker 대몬이 새 컨테이너를 만들 수 없음을 나타냅니다.

```
CannotCreateContainerError: API error (500): devmapper: Thin Pool has 4350 free data
 blocks which is less than minimum required 4454 free data blocks. Create more free
 space in thin pool or use dm.min_free_space option to change behavior
```

기본적으로 버전 2015.09.d 이상의 Amazon ECS 최적화 Amazon Linux AMI는 /dev/xvda에서 연결되고 파일 시스템의 루트로 마운트되는 운영 체제를 위한 8GiB 볼륨으로 시작됩니다. 여기에 Docker가 이미지 및 메타데이터 저장에 사용하는 /dev/xvdcz에서 연결된 22GiB의 볼륨이 추가됩니다. 이 스토리지 공간이 가득 찬 경우 Docker 대몬은 새 컨테이너를 생성할 수 없습니다.

컨테이너 인스턴스에 스토리지를 추가하는 가장 쉬운 방법은 기존 인스턴스를 종료하고 데이터 스토리지 볼륨이 더 큰 새 인스턴스를 시작하는 것입니다. 하지만 이것이 불가능한 경우, Amazon ECS 최적화 Linux AMI의 절차에 따라 Docker가 사용하는 볼륨 그룹에 스토리지를 추가하여 논리 볼륨을 확장할 수 있습니다.

컨테이너 인스턴스 스토리지가 너무 빨리 채워지는 경우, 몇 가지 조치로 이를 완화할 수 있습니다.

- 씬 폴링 정보를 보려면 컨테이너 인스턴스에서 다음 명령을 실행합니다.

  ```
  docker info
  ```

- (Amazon ECS 컨테이너 에이전트 1.8.0 이상) 중지되거나 종료된 컨테이너가 컨테이너 인스턴스에 남아 있는 시간을 줄일 수 있습니다. ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION 에이전트 구성 변수는 작업이 중지된 때부터 Docker 컨테이너가 제거될 때까지 기다리는 시간을 설정합니다(기본값: 3시간). 이때 Docker 컨테이너 데이터가 제거되므로 이 값을 너무 낮게 설정할 경우 데이터가 제거되기 전에 중지된 컨테이너를 검사하거나 로그를 볼 수 없을 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

- 컨테이너 인스턴스에서 실행되고 있지 않은 컨테이너와 사용되지 않는 이미지를 제거할 수 있습니다. 다음 예제 명령을 사용하여 중지된 컨테이너와 사용하지 않은 이미지를 수동으로 제거할 수 있습니다. 삭제된 컨테이너는 나중에 검사할 수 없으며, 삭제된 이미지는 이미지에서 새 컨테이너를 시작하기 전에 다시 가져와야 합니다.

  실행되고 있지 않은 컨테이너를 제거하려면 컨테이너 인스턴스에서 다음 명령을 실행합니다.

  ```
  docker rm $(docker ps -aq)
  ```

  사용하지 않은 이미지를 제거하려면 컨테이너 인스턴스에서 다음 명령을 실행합니다.

  ```
  docker rmi $(docker images -q)
  ```

- 컨테이너 내에서 사용하지 않은 데이터 블록을 제거할 수 있습니다. 다음 명령을 사용하여 실행 중인 컨테이너에서 fstrim을 실행하고 컨테이너 파일 시스템에서 사용하지 않은 데이터 블록을 삭제할 수 있습니다.

  ```
  sudo sh -c "docker ps -q | xargs docker inspect --format='{{ .State.Pid }}' | xargs -IZ fstrim /proc/Z/root/"
  ```

Amazon ECS Exec 문제 해결

다음은 ECS Exec을 사용할 때 오류가 발생할 수 있는 이유를 진단하는 데 도움이 되는 문제 해결 정보입니다.

Exec Checker를 사용하여 확인

ECS Exec 검사기 스크립트는 Amazon ECS 클러스터와 작업이 ECS Exec 기능을 사용하기 위한 필수 조건을 충족했는지 확인하고 검증하는 방법을 제공합니다. ECS Exec 검사기 스크립트는 사용자를 대신하여 다양한 API를 호출하여 AWS CLI 환경과 클러스터 및 작업이 ECS Exec에 대해 모두 준비되었는지 확인합니다. 이 도구에는 AWS CLI의 최신 버전이 필요하며 `jq`를 사용할 수 있습니다. 자세한 내용은 GitHub의 ECS Exec Checker를 참조하세요.

`execute-command` 호출 중 오류 발생

`The execute command failed` 오류가 발생하는 경우 가능한 원인은 다음과 같습니다.

- 작업에 필요한 권한이 없습니다. 태스크를 시작하는 데 사용되는 태스크 정의에 작업 IAM 역할이 정의되어 있고 역할에 필요한 권한이 있는지 확인합니다. 자세한 내용은 ECS Exec 권한 섹션을 참조하세요.
- SSM 에이전트가 설치되지 않았거나 실행 중이 아닙니다.
- Amazon ECS용 인터페이스 Amazon VPC 엔드포인트가 있지만 Systems Manager Session Manager용 인터페이스 Amazon VPC 엔드포인트는 없습니다.

Amazon ECS Anywhere 문제 해결

Amazon ECS Anywhere는 외부 인스턴스(예: 온프레미스 서버 또는 가상 머신(VM))을 Amazon ECS 클러스터에 등록하도록 지원합니다. 발생할 수 있는 일반적인 문제와 이에 대한 일반적인 문제 해결 권장 사항은 다음과 같습니다.

주제

- 외부 인스턴스 등록 문제
- 외부 인스턴스 네트워크 문제
- 외부 인스턴스에서의 작업 실행 문제

외부 인스턴스 등록 문제

Amazon ECS 클러스터에 외부 인스턴스를 등록할 때 다음 요구 사항을 충족해야 합니다.

- AWS Systems Manager 활성화(활성화 ID 및 활성화 코드로 구성됨)를 검색해야 합니다. 외부 인스턴스를 Systems Manager 관리형 인스턴스로 등록하는 데 사용합니다. Systems Manager 활성화를 요청하면 등록 제한 및 만료 날짜를 지정합니다. 등록 제한은 정품 인증을 사용하여 등록할 수 있는 인스턴스의 최대 수를 지정합니다. 등록 제한의 기본값은 1개의 인스턴스입니다. 만료 날짜는 정품 인증이 만료되는 시기를 지정합니다. 기본값은 24시간입니다. 외부 인스턴스를 등록하는 데 사용하는 Systems Manager 정품 인증이 유효하지 않은 경우 새 인스턴스를 요청합니다. 자세한 내용은 Amazon ECS 클러스터에 외부 인스턴스 등록 섹션을 참조하세요.
- IAM 정책은 AWS API 작업과 통신하는 데 필요한 권한을 외부 인스턴스에 제공하는 데 사용됩니다. 이 관리형 정책이 제대로 생성되지 않고 필요한 권한이 포함되어 있지 않으면 외부 인스턴스 등록이 실패합니다. 자세한 내용은 Amazon ECS Anywhere IAM 역할 섹션을 참조하세요.
- Amazon ECS는 Docker, Amazon ECS 컨테이너 에이전트 및 Systems Manager Agent를 외부 인스턴스에 설치하는 설치 스크립트를 제공합니다. 설치 스크립트가 실패하면 오류가 발생하지 않고 동일한 인스턴스에서 스크립트를 다시 실행할 수 없습니다. 이 경우 정리 프로세스에 따라 인스턴스에

서 AWS 리소스를 지우고 설치 스크립트를 다시 실행할 수 있습니다. 자세한 내용은 Amazon ECS 외부 인스턴스 등록 취소 섹션을 참조하세요.

> **Note**
>
> 설치 스크립트가 Systems Manager 정품 인증을 성공적으로 요청하여 사용한 경우 설치 스크립트를 다시 실행하면 Systems Manager 정품 인증이 다시 사용됩니다. 이로 인해 해당 정품 인증에 대한 등록 제한에 도달하게 될 수 있습니다. 이 제한에 도달하면 새 정품 인증을 생성해야 합니다.

- GPU 워크로드용 외부 인스턴스에서 설치 스크립트를 실행할 때 NVIDIA 드라이버가 제대로 감지되지 않거나 구성되지 않으면 오류가 발생합니다. 설치 스크립트는 `nvidia-smi` 명령을 사용하여 NVIDIA 드라이버의 존재를 확인합니다.

외부 인스턴스 네트워크 문제

변경 사항을 전달하려면 외부 인스턴스에 AWS에 대한 네트워크 연결이 필요합니다. 외부 인스턴스에 AWS에 대한 네트워크 연결이 끊긴 경우 수동으로 중지하지 않는 한 인스턴스에서 실행 중인 작업이 계속 실행됩니다. AWS에 대한 연결이 복원되면 외부 인스턴스의 Amazon ECS 컨테이너 에이전트 및 Systems Manager Agent에서 사용하는 AWS 자격 증명이 자동으로 갱신됩니다. 외부 인스턴스와 AWS 간의 통신에 사용되는 AWS 도메인에 관한 자세한 내용은 네트워킹 섹션을 참조하세요.

외부 인스턴스에서의 작업 실행 문제

외부 인스턴스에서 작업 또는 컨테이너가 실행되지 않는 경우 가장 일반적인 원인은 네트워크 또는 권한과 관련이 있습니다. 컨테이너가 Amazon ECR에서 이미지를 가져오거나 컨테이너 로그를 CloudWatch Logs로 보내도록 구성된 경우 태스크 정의에서 유효한 작업 실행 IAM 역할을 지정해야 합니다. 유효한 작업 실행 IAM 역할이 없으면 컨테이너가 시작되지 않습니다. 네트워크 관련 문제에 대한 자세한 내용은 외부 인스턴스 네트워크 문제 섹션을 참조하세요.

> **Important**
>
> Amazon ECS는 Amazon ECS 로그 수집 도구를 제공합니다. 문제 해결을 위해 외부 인스턴스에서 로그를 수집하는 데 사용할 수 있습니다. 자세한 내용은 Amazon ECS 로그 수집기를 사용하여 컨테이너 로그 수집 섹션을 참조하세요.

AWS Fargate 제한 할당량

AWS Fargate은(는) 리전별로 각 AWS 계정에 대한 토큰 버킷 알고리즘을 사용하여 Amazon ECS 작업 및 Amazon EKS 포드 시작률을 할당량(이전에는 한도로 지칭)으로 제한합니다. 이 알고리즘을 사용하면 계정에 특정 수의 토큰을 보관하는 버킷이 있습니다. 버킷의 토큰 수는 지정된 초당 비율 할당량을 나타냅니다. 각 고객 계정에는 작업 및 포드 토큰 버킷이 있고, 이는 고객 계정에서 실행된 작업 및 포드의 수에 따라 고갈됩니다. 이 토큰 버킷에는 주기적으로 더 많은 수의 요청을 할 수 있는 최대 버킷과 필요한 기간 동안 꾸준한 요청률을 유지할 수 있는 리필률이 있습니다.

예를 들어, Fargate 고객 계정의 작업 및 포드 토큰 버킷 크기는 100토큰이고 리필률은 초당 20토큰입니다. 따라서 고객 계정당 최대 100개의 Amazon ECS 작업 및 Amazon EKS 포드를 즉시 시작할 수 있으며, 초당 20개의 Amazon ECS 작업 및 Amazon EKS 포드의 지속적인 시작률을 제공합니다.

작업	버킷 최대 용량(또는 버스트율)	버킷 리필율(또는 지속률)
On Demand Amazon ECS 작업 및 Amazon EKS 포드에 대한 Fargate 리소스 비율 할당량[1]	100	20
Spot Amazon ECS 작업에 대한 Fargate 리소스 비율 할당량	100	20

[1]Amazon EKS 포드만 시작하는 계정은 Amazon EKS 플랫폼 버전에서 불러온 플랫폼 버전을 사용할 때 초당 20의 포드 시작의 지속적인 포드 시작률로 20의 버스트율을 가집니다.

Fargate에서 **RunTask** API 제한

또한 Fargate는 별도의 할당량을 사용하는 Amazon ECS RunTask API를 사용하여 작업을 시작할 때 요청률을 제한합니다. Fargate는 리전별로 각 AWS 계정에 대한 Amazon ECS RunTask API 요청을 제한합니다. 제출한 각 요청은 버킷에서 하나의 토큰을 제거합니다. 당사는 서비스 수행을 돕고 모든 Fargate 고객에게 공정한 사용을 보장하기 위해 이를 수행합니다. API 호출은 Amazon Elastic Container Service 콘솔, 명령줄 도구 또는 서드 파티 애플리케이션에서 발생했는지와 관계없이 요청 할당량이 적용됩니다. Amazon ECS RunTask API에 대한 호출에 대한 비율 할당량은 초당 20개의 호출(버스트 및 지속)입니다. 그러나 이 API를 호출할 때마다 최대 10개의 작업을 시작할 수 있습니다. 즉, 이 API를 10번 호출하여 각 호출마다 10개의 작업을 시작하도록 요청하여 1초에 100개의 작업을 시작할 수 있습니다. 마찬가지로 이 API를 20번 호출하여 각 호출마다 5개의 작업을 시작하도록 요청

할 수도 있습니다. Amazon ECS RunTask API의 API 제한에 대한 자세한 내용은 Amazon ECS API 참조의 API request throttling을 참조하세요.

실제로 작업 및 포드 시작률은 다운로드 및 압축 해제할 컨테이너 이미지, 상태 확인 및 로드 밸런서에 작업 또는 포드 등록과 같은 활성화된 기타 통합과 같은 다른 고려 사항에 따라 달라집니다. 고객에 대해 활성화한 기능에 따라 위에 표시된 할당량과 비교하여 작업 및 포드 시작률이 달라질 수 있습니다.

Fargate에서 요금 할당량 조정

AWS 계정에 대한 Fargate 비율 제한 할당량 증가를 요청할 수 있습니다. 자세한 설명은 Service Quotas 사용자 가이드에서 할당량 증가 요청을 참조하세요.

Amazon ECS의 API 실패 이유

Amazon ECS API, 콘솔 또는 AWS CLI를 통해 트리거한 API 작업이 종료되고 `failures` 오류 메시지가 나타나면 다음을 통해 원인 해결에 도움을 줄 수 있습니다. 실패로 인해 실패와 연관된 리소스의 이유 및 Amazon 리소스 이름(ARN)이 반환됩니다.

대부분의 리소스는 리전 전용이므로 콘솔을 사용할 때에는 해당 리소스에 대한 올바른 리전을 설정했는지 확인해야 합니다. AWS CLI를 사용할 때에는 AWS CLI 명령이 `--region` *region* 파라미터가 포함된 올바른 리전으로 전송되고 있는지 확인합니다.

`Failure` 데이터 형식의 구조에 대한 자세한 내용은 Amazon Elastic Container Service API 참조의 실패를 참조하세요.

다음은 API 명령을 실행할 때 수신할 수 있는 실패 메시지 예제입니다.

API 작업	실패 이유 또는 중단 사유	원인
`DescribeClusters`	`MISSING`	지정한 클러스터를 찾을 수 없습니다. 클러스터 이름의 맞춤법을 확인합니다.
`DescribeInstances`	`MISSING`	지정한 컨테이너 인스턴스를 찾을 수 없습니다. 컨테이너 인스턴스가 등록된 클러스터를 지정했는지, 컨테이너 인스턴스 ARN 또는 ID가 모두 올바른지 확인합니다.

API 작업	실패 이유 또는 중단 사유	원인
DescribeServices	MISSING	지정한 서비스를 찾을 수 없습니다. 올바른 클러스터 또는 리전이 지정되었고 서비스 ARN 또는 이름이 유효한지 확인합니다.
DescribeTasks	MISSING	지정한 태스크를 찾을 수 없습니다. 올바른 클러스터 또는 리전이 지정되었고 작업 ARN 또는 ID가 모두 유효한지 확인합니다.
DescribeTasks	TaskFailedToStart: RESOURCE:*	RESOURCE:CPU 오류의 경우 작업에서 요청한 CPU 수를 컨테이너 인스턴스에서 사용할 수 없습니다. 이는 일반적으로 작업 정의의 CPU 단위 요구 사항이 용량 공급자에 매핑된 Auto Scaling 그룹에 정의된 Amazon EC2 인스턴스의 CPU 크기보다 클 때 발생합니다. 용량 공급자 구성을 확인해야 합니다. RESOURCE:MEMORY 오류의 경우 작업에서 요청한 메모리 양을 컨테이너 인스턴스에서 사용할 수 없습니다. 이는 일반적으로 작업 정의의 메모리 크기 요구 사항이 용량 공급자에 매핑된 Auto Scaling 그룹에 정의된 Amazon EC2 인스턴스의 지원되는 메모리보다 클 때 발생합니다. 용량 공급자 구성을 확인해야 합니다.

API 작업	실패 이유 또는 중단 사유	원인
	TaskFailedToStart: AGENT	태스크를 시작하려고 시도한 컨테이너 인스턴스에 현재 연결이 끊긴 에이전트가 있습니다. 작업 배치를 위한 대기 시간이 길어지는 것을 방지하기 위해 요청이 거부되었습니다. 연결이 끊어진 에이전트 문제 해결 방법에 대한 자세한 내용은 연결이 끊어진 Amazon ECS 에이전트 문제를 해결하려면 어떻게 해야 하나요?를 참조하세요.
	TaskFailedToStart: MemberOf placement constraint unsatisfied	작업 정의에 정의된 배치 제약 조건을 충족하는 컨테이너 인스턴스가 없습니다.

API 작업	실패 이유 또는 중단 사유	원인
	TaskFailedToStart: ATTRIBUTE	태스크 정의에 컨테이너 인스턴스에서 사용할 수 없는 특정 컨테이너 인스턴스 속성을 필요로 하는 파라미터가 포함되어 있습니다. 예를 들어 해당 작업에서 awsvpc 네트워크 모드를 사용하지만 지정된 서브넷에 ecs.capability.task-eni 속성을 지닌 인스턴스가 없는 경우가 있습니다. 특정 태스크 정의 파라미터와 에이전트 구성 변수에 어떤 속성이 필요한지에 대한 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터 및 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.
	TaskFailedToStart: NO ACTIVE INSTANCES	용량 공급자에 활성 인스턴스가 없습니다. Auto Scaling 그룹을 관리하는 방법에 대한 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 Auto Scaling 그룹을 참조하세요.

API 작업	실패 이유 또는 중단 사유	원인
	`TaskFailedToStart: EMPTY CAPACITY PROVIDER`	클러스터에 인스턴스가 없습니다. 이는 대부분 용량 공급자가 비어 있거나 용량 공급자의 인스턴스가 클러스터에 등록되지 않았기 때문일 수 있습니다. Auto Scaling 그룹을 관리하는 방법에 대한 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 Auto Scaling 그룹을 참조하세요.
`GetTaskProtection`	`MISSING`	지정한 태스크를 찾을 수 없습니다. 클러스터 이름 또는 ARN과 작업 ARN 또는 ID가 유효한지 확인합니다.
	`TASK_NOT_VALID`	지정된 작업은 Amazon ECS 서비스의 일부가 아닙니다. Amazon ECS 서비스 관리형 작업만 보호할 수 있습니다. 작업 ARN 또는 ID를 확인하고 다시 시도하세요.

API 작업	실패 이유 또는 중단 사유	원인
RunTask 또는 StartTask	RESOURCE:*	작업에서 요청한 리소스를 클러스터의 컨테이너 인스턴스에서 사용할 수 없습니다. 리소스가 CPU, 메모리, 포트 또는 탄력적 네트워크 인터페이스인 경우 클러스터에 추가 컨테이너 인스턴스를 추가해야 하는 경우도 있습니다. RESOURCE:ENI 오류의 경우 클러스터에 사용 가능한 탄력적 네트워크 인터페이스 연결 지점이 없습니다. 이러한 연결 지점은 awsvpc 네트워크 모드를 사용하는 작업에 필요합니다. Amazon EC2 인스턴스에 연결할 수 있는 네트워크 인터페이스의 개수에는 한도가 있고 기본 네트워크 인터페이스는 한 개로 계산됩니다. 각 인스턴스 유형에 대해 지원되는 네트워크 인터페이스 수에 대한 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 유형별 네트워크 인터페이스당 IP 주소를 참조하세요. RESOURCE:GPU 오류의 경우 작업에서 요청한 GPU 수를 사용할 수 없으며 GPU 지원 컨테이너 인스턴스를 클러스터에 추가해야 할 수 있습니다. 자세한 내용은 GPU 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요.

API 작업	실패 이유 또는 중단 사유	원인
	AGENT	태스크를 시작하려고 시도한 컨테이너 인스턴스에 현재 연결이 끊긴 에이전트가 있습니다. 작업 배치를 위한 대기 시간이 길어지는 것을 방지하기 위해 요청이 거부되었습니다. 연결이 끊어진 에이전트 문제 해결 방법에 대한 자세한 내용은 연결이 끊어진 Amazon ECS 에이전트 문제를 해결하려면 어떻게 해야 하나요?를 참조하세요.
	LOCATION	태스크를 시작하려고 시도한 컨테이너 인스턴스가 awsVpcConfiguration 에서 지정한 서브넷과 다른 가용 영역에 있습니다.

API 작업	실패 이유 또는 중단 사유	원인
	ATTRIBUTE	태스크 정의에 컨테이너 인스턴스에서 사용할 수 없는 특정 컨테이너 인스턴스 속성을 필요로 하는 파라미터가 포함되어 있습니다. 예를 들어 해당 작업에서 awsvpc 네트워크 모드를 사용하지만 지정된 서브넷에 ecs.capability.task-eni 속성을 지닌 인스턴스가 없는 경우가 있습니다. 특정 태스크 정의 파라미터와 에이전트 구성 변수에 어떤 속성이 필요한지에 대한 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 태스크 정의 파라미터 및 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.
StartTask	MISSING	작업을 시작하려고 시도한 컨테이너 인스턴스를 찾을 수 없습니다. 잘못된 클러스터 또는 리전이 지정되었거나 컨테이너 인스턴스 ARN 또는 ID를 잘못 입력했는지 확인합니다.
	INACTIVE	태스크를 시작하려고 시도한 컨테이너 인스턴스가 이전에 Amazon ECS에 대한 등록이 취소되었고 사용할 수 없습니다.
StopServiceDeployment	ECS deployment failed	사기 계정이 StopServiceDeployment API를 실행했습니다.

API 작업	실패 이유 또는 중단 사유	원인
TagResource	InvalidParameterException	태깅하려는 서비스의 ARN은 짧은 형식입니다. 긴 형식으로 마이그레이션해야 합니다. ARN 마이그레이션 방법에 대한 자세한 내용은 Amazon ECS 단기 서비스 ARN을 긴 ARN으로 마이그레이션 섹션을 참조하세요.
UpdateTaskProtection	DEPLOYMENT_BLOCKED	하나 이상의 보호된 작업으로 인해 서비스 배포가 안정 상태에 도달하지 못하므로 작업 보호를 설정할 수 없습니다. 기존 작업에 대한 작업 보호를 해제하거나 작업 보호가 만료될 때까지 기다립니다.
	MISSING	지정한 태스크를 찾을 수 없습니다. 클러스터 이름 또는 ARN과 작업 ARN 또는 ID가 유효한지 확인합니다.
	TASK_NOT_VALID	지정된 작업은 Amazon ECS 서비스의 일부가 아닙니다. Amazon ECS 서비스 관리형 작업만 보호할 수 있습니다. 작업 ARN 또는 ID를 확인하고 다시 시도하세요.

Note

여기에 설명된 실패 시나리오 외에도 API 작업은 예외로 인해 실패하여 오류 응답이 발생할 수도 있습니다. 이러한 예외 목록은 일반 오류를 참조하세요.

Amazon Elastic Container Service의 보안

AWS는 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객으로서 여러분은 가장 높은 보안 요구 사항을 충족하기 위해 설계된 데이터 센터 및 네트워크 아키텍처의 혜택을 받게 됩니다.

보안은 AWS과 사용자의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS클라우드에서 AWS서비스를 실행하는 인프라를 보호합니다. AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사원은 정기적으로 AWS규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. Amazon Elastic Container Service에 적용되는 규정 준수 프로그램에 대한 자세한 정보는 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하세요.
- 클라우드 내 보안 – 귀하의 책임은 귀하가 사용하는 AWS서비스에 의해 결정됩니다. 또한 귀하는 귀사의 데이터 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Amazon ECS를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목적에 맞게 Amazon ECS를 구성하는 방법을 보여줍니다. 또한 Amazon ECS 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다.

주제

- Amazon Elastic Container Service용 Identity and Access Management
- Amazon Elastic Container Service의 로깅 및 모니터링
- Amazon Elastic Container Service에 대한 규정 준수 확인
- AWS Fargate Federal Information Processing Standard(FIPS-140)
- Amazon Elastic Container Service의 인프라 보안
- Amazon ECS에 대한 AWS 공동 책임 모델
- Amazon ECS에 대한 네트워크 보안 모범 사례
- Amazon ECS 태스크 및 컨테이너 보안 모범 사례

Amazon Elastic Container Service용 Identity and Access Management

AWS Identity and Access Management(IAM)은 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있도록 지원하는 AWS 서비스입니다. IAM 관리자는 어떤 사용자가 Amazon ECS 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 보유)을 받을 수 있는지를 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

주제

- 고객
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- Amazon Elastic Container Service가 IAM과 작동하는 방식
- Amazon Elastic Container Service의 자격 증명 기반 정책 예
- Amazon Elastic Container Service용 AWS 관리형 정책
- Amazon ECS에 대해 서비스 연결 역할 사용
- Amazon ECS에 대한 IAM 역할
- Amazon ECS 콘솔에 필요한 권한
- Amazon ECS 블루/그린 배포에서 Lambda 함수에 필요한 권한
- Amazon ECS 서비스 Auto Scaling에 필요한 IAM 권한
- 생성 시 리소스에 태그를 지정할 수 있는 권한 부여
- Amazon Elastic Container Service Identity and Access 문제 해결
- Amazon ECS에 대한 IAM 모범 사례

고객

AWS Identity and Access Management(IAM)를 사용하는 방법은 Amazon ECS에서 수행하는 작업에 따라 달라집니다.

서비스 사용자 - Amazon ECS 서비스를 사용하여 태스크를 수행하는 경우 필요한 자격 증명과 권한은 관리자가 제공합니다. 더 많은 Amazon ECS 기능을 사용하여 태스크를 수행한다면 추가 권한이 필

요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. Amazon ECS의 기능에 액세스할 수 없다면 Amazon Elastic Container Service Identity and Access 문제 해결 섹션을 참조하세요.

서비스 관리자 – 회사에서 Amazon ECS 리소스를 책임지고 있다면 Amazon ECS에 대한 완전한 액세스 권한이 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 Amazon ECS 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사가 Amazon ECS에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 Amazon Elastic Container Service가 IAM과 작동하는 방식 섹션을 참조하세요.

IAM 관리자 - IAM 관리자라면 Amazon ECS에 대한 액세스 권한 관리를 위한 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 Amazon ECS 자격 증명 기반 정책 예제를 보려면 Amazon Elastic Container Service의 자격 증명 기반 정책 예를 참조하세요.

ID를 통한 인증

인증은 ID 자격 증명을 사용하여 AWS에 로그인하는 방식입니다. AWS 계정 루트 사용자이나 IAM 사용자로, 또는 IAM 역할을 수임하여 인증(AWS에 로그인)받아야 합니다.

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 AWS에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증, Google 또는 Facebook 보안 인증이 페더레이션 ID의 예제입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 연동을 사용하여 AWS에 액세스하면 간접적으로 역할을 수임합니다.

사용자 유형에 따라AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. AWS에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법을 참조하세요.

AWS에 프로그래밍 방식으로 액세스하는 경우, AWS에서는 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK) 및 명령줄 인터페이스(CLI)를 제공합니다. AWS 도구를 사용하지 않는 경우, 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 자세한 방법은 IAM 사용 설명서에서 API 요청용 AWS Signature Version 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS은(는) 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 다중 인증 및 IAM 사용 설명서에서 IAM의 AWS 다중 인증을 참조하세요.

AWS 계정 루트 사용자

AWS 계정을(를) 생성할 때는 해당 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 단일 로그인 ID로 시작합니다. 이 ID는 AWS 계정루트 사용자라고 하며, 계정을 생성할 때 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 보안 인증이 필요한 작업을 참조하세요.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 포함한 사용자가 ID 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 AWS 서비스에 액세스하도록 요구합니다.

페더레이션 ID는 엔터프라이즈 사용자 디렉터리, 웹 ID 공급자, AWS Directory Service, Identity Center 디렉터리의 사용자 또는 자격 증명 소스를 통해 제공된 자격 증명을 사용하여 AWS 서비스에 액세스하는 모든 사용자입니다. 페더레이션 ID는 AWS 계정에 액세스할 때 역할을 수임하고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을(를) 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 모든 AWS 계정 및 애플리케이션에서 사용하기 위해 고유한 ID 소스의 사용자 및 그룹 집합에 연결하고 동기화할 수 있습니다. IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇인가요?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있

지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한을 가지고 있는 AWS 계정 계정 내 ID입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. AWS Management Console에서 일시적으로 IAM 역할을 수임하려면 사용자에서 IAM 역할로 전환(콘솔)하면 됩니다. AWS CLI 또는 AWS API 작업을 직접적으로 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 관련 역할에 대한 자세한 내용은 IAM 사용 설명서의 Create a role for a third-party identity provider (federation)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 집합을 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 하지만 일부 AWS 서비스을(를) 사용하면 리소스에 정책을 직접 연결할 수 있습니다(역할을 프록시로서 사용하는 대신) . 교차 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하세요.
- 교차 서비스 액세스 - 일부 AWS 서비스는 다른 AWS 서비스의 특성을 사용합니다. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 위탁자의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여 AWS에서 작업을 수행하는 사람은 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 직접 호출하는 위탁자의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집

니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

- 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.
- 서비스 연결 역할 – 서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 유형입니다. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 링크 역할은 AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수는 없습니다.

- Amazon EC2에서 실행 중인 애플리케이션 – IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS역할을 할당하고 해당 역할을 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

정책을 사용하여 액세스 관리

정책을 생성하고 AWS ID 또는 리소스에 연결하여 AWS에서 내 액세스를 제어합니다. 정책은 ID 또는 리소스와 연결될 때 해당 권한을 정의하는 AWS의 객체입니다. AWS는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청을 보낼 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 AWS에 JSON 문서로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console, AWS CLI 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 AWS 계정에 속한 다수의 사용자, 그룹 및 역할에 독립적으로 추가할 수 있는 정책입니다. 관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함되어 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 AWS 서비스가 포함될 수 있습니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3, AWS WAF 및 Amazon VPC는 ACL을 지원하는 대표적인 서비스입니다. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS는 비교적 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 ID 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔티티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCP) – SCP는 AWS Organizations에서 조직 또는 조직 단위(OU)에 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations는 기업이 소유하는 여러 개의 AWS 계정을 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 특성을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 AWS 계정 루트 사용자을 비롯하여 멤버 계정의 엔티티에 대한 권한을 제한합니다. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서에서 Service control policies을 참조하세요.
- 리소스 제어 정책(RCP) - RCP는 소유한 각 리소스에 연결된 IAM 정책을 업데이트하지 않고 계정의 리소스에 대해 사용 가능한 최대 권한을 설정하는 데 사용할 수 있는 JSON 정책입니다. RCP는 멤버 계정의 리소스에 대한 권한을 제한하며 조직에 속하는지 여부에 관계없이 AWS 계정 루트 사용자를 포함한 ID에 대한 유효 권한에 영향을 줄 수 있습니다. RCP를 지원하는 AWS 서비스 목록을 포함하여 Organizations 및 RCP에 대한 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCP)을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 ID 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련될 때 AWS가 요청을 허용할지를 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

Amazon Elastic Container Service가 IAM과 작동하는 방식

IAM을 사용하여 Amazon ECS에 대한 액세스를 관리하기 전에 Amazon ECS에서 사용할 수 있는 IAM 기능에 대해 알아봅니다.

IAM 기능	Amazon ECS 지원
ID 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	부분
정책 조건 키	예
ACLs	아니요
ABAC(정책의 태그)	예
임시 보안 인증	예
전달 액세스 세션(FAS)	예
서비스 역할	예
서비스 연결 역할	예

Amazon ECS 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스를 참조하세요.

Amazon ECS의 자격 증명 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. ID 기반 정책에서는 위탁자가 연결된 사용자 또는 역할에 적용되므로 위탁자를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

Amazon ECS의 자격 증명 기반 정책 예

Amazon ECS 자격 증명 기반 정책 예제를 보려면 Amazon Elastic Container Service의 자격 증명 기반 정책 예 섹션을 참조하세요.

Amazon ECS 내의 리소스 기반 정책

리소스 기반 정책 지원: 아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 AWS 서비스가 포함될 수 있습니다.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 위탁자로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하세요. 보안 주체와 리소스가 서로 다른 AWS 계정에 있는 경우, 신뢰할 수 있는 계정의 IAM 관리자는 보안 주체 엔터티(사용자 또는 역할)에도 리소스 액세스 권한을 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 위탁자에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

Amazon ECS의 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action` 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 일반적으로 정책 작업의 이름은 연결된 AWSAPI 작업의 이름과 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

Amazon ECS 작업의 목록을 보려면 서비스 승인 참조의 Amazon Elastic Kubernetes Service에서 정의한 작업을 참조하세요.

Amazon ECS의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
ecs
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "ecs:action1",
      "ecs:action2"
         ]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, Describe라는 단어로 시작하는 모든 태스크를 지정하려면 다음 태스크를 포함합니다.

```
"Action": "ecs:Describe*"
```

Amazon ECS 자격 증명 기반 정책 예제를 보려면 Amazon Elastic Container Service의 자격 증명 기반 정책 예 섹션을 참조하세요.

Amazon ECS의 정책 리소스

정책 리소스 지원: 부분적

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Resource JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 Resource또는 NotResource요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

Amazon ECS 리소스 유형 및 해당 ARN의 목록을 보려면 서비스 승인 참조의 Amazon Elastic Container Service에서 정의한 리소스 유형을 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Amazon Elastic Container Service에서 정의한 작업을 참조하세요.

일부 Amazon ECS API 작업은 여러 리소스를 지원합니다. 예를 들어, `DescribeClusters` API 태스크를 호출할 때 여러 클러스터를 참조할 수 있습니다. 단일 문에서 여러 리소스를 지정하려면 ARN을 쉼표로 구분합니다.

```
"Resource": [
      "EXAMPLE-RESOURCE-1",
      "EXAMPLE-RESOURCE-2"
```

예를 들어, Amazon ECS 클러스터 리소스의 ARN은 다음과 같습니다.

```
arn:${Partition}:ecs:${Region}:${Account}:cluster/${clusterName}
```

예를 들어 문에서 `my-cluster-1` 및 `my-cluster-2` 클러스터를 지정하려면 다음 ARN을 사용합니다.

```
"Resource": [
        "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster-1",
        "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster-2"
```

특정 계정에 속하는 모든 클러스터를 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/*"
```

태스크 정의의 경우 최신 개정이나 특정 개정을 지정할 수 있습니다.

작업 정의에 대한 모든 개정을 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource:arn:${Partition}:ecs:${Region}:${Account}:task-definition/
${TaskDefinitionFamilyName}:*"
```

특정 태스크 정의 개정을 지정하려면 ${TaskDefinitionRevisionNumber}를 사용합니다.

```
"Resource:arn:${Partition}:ecs:${Region}:${Account}:task-definition/
${TaskDefinitionFamilyName}:${TaskDefinitionRevisionNumber}"
```

Amazon ECS 자격 증명 기반 정책 예제를 보려면 Amazon Elastic Container Service의 자격 증명 기반 정책 예 섹션을 참조하세요.

Amazon ECS의 정책 조건 키

서비스별 정책 조건 키 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Condition` 요소(또는 `Condition` 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 `Condition` 요소를 지정하거나 단일 `Condition` 요소에서 여러 키를 지정하는 경우, AWS는 논리적 `AND` 작업을 사용하여 평가합니다. 단일 조건 키의 여러 값을 지정하는 경우, AWS는 논리적 `OR` 작업을 사용하여 조건을 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

Amazon ECS는 IAM 정책에 대한 세분화된 필터링을 제공하는 데 사용할 수 있는 다음과 같은 서비스별 조건 키를 지원합니다.

조건 키	설명	평가 유형
aws:RequestTag/${TagKey}	이 컨텍스트 키는 `"aws:RequestTag/ tag-key":"tag-value "` 형식으로, 여기서 *tag-key* 및 *tag-value* 는 태그 키 및 값 쌍입니다. AWS 요청에 태그 키-값 쌍이 있는지 확인하세요. 예를 들어, 요청에 태그 키 `"Dept"`가 포함되어 있으며 그 값이 `"Accounting"` 인지 확인할 수 있습니다.	String
aws:ResourceTag/${TagKey}	이 컨텍스트 키는 `"aws:ResourceTag/ tag-key":"tag-value "` 형식으로, 여기서 *tag-key* 및 *tag-value* 는 태그 키 및 값 페어입니다.	String

조건 키	설명	평가 유형
	자격 검증 리소스(사용자 또는 역할)에 연결된 태그가 지정된 키 이름 및 값과 일치하는지 확인합니다.	
aws:TagKeys	이 컨텍스트 키는 "aws:TagKeys":" *tag-key*" 형식이며, 여기서 *tag-key*는 값이 없는 태그 키 목록입니다(예: ["Dept","Cost-Center"]). AWS 요청에 있는 태그 키를 확인합니다.	String
ecs:ResourceTag/${TagKey}	이 컨텍스트 키는 "ecs:ResourceTag/ *tag-key*":"*tag-value* " 형식으로, 여기서 *tag-key* 및 *tag-value* 는 태그 키 및 값 페어입니다. 자격 검증 리소스(사용자 또는 역할)에 연결된 태그가 지정된 키 이름 및 값과 일치하는지 확인합니다.	String
ecs:account-setting	컨텍스트 키는 "ecs:account-setting":" *account-setting* " 형식으로 지정됩니다. 여기서 *account-setting* 은 계정 설정의 이름입니다.	String
ecs:auto-assign-public-ip	컨텍스트 키는 "ecs:auto-assign-public-ip":" *value*" 형식으로 지정됩니다. 여기서 *value*-는 "true" 또는 "false"입니다.	String
ecs:capacity-provider	컨텍스트 키는 "ecs:capacity-provider":" *capacity-provider-arn* " 형식으로 지정됩니다. 여기서 *capacity-provider-arn* 은 용량 공급자의 ARN입니다.	ARN, Null
ecs:cluster	이 컨텍스트 키는 "ecs:cluster":" *cluster-arn* " 형식으로, 여기서 *cluster-arn* 은 Amazon ECS 클러스터의 ARN입니다.	ARN, Null
ecs:compute-compatability	컨텍스트 키는 "ecs:compute-compatability":" *compatability-type* " 형식으로 지정됩니다. 여기서 *compatability-type* 은 'FARGATE', 'EC2' 또는 'EXTERNAL'입니다.	String

조건 키	설명	평가 유형
ecs:container-instances	이 컨텍스트 키는 "ecs:container-instances":" *container-instance-arns* " 형식으로, 여기서 *container-instance-arns* 는 하나 이상의 컨테이너 인스턴스 ARN입니다.	ARN, Null
ecs:container-name	컨텍스트 키는 "ecs:container-name":" *container-name* " 형식으로 지정됩니다. 여기서 *container-instance-* 는 작업 정의에 정의된 Amazon ECS 컨테이너의 이름입니다.	String
ecs:enable-execute-command	컨텍스트 키는 "ecs:enable-execute-command":" *value*" 형식으로 지정됩니다. 여기서 *value-*는 "true" 또는 "false"입니다.	String
ecs:enable-service-connect	컨텍스트 키는 "ecs:enable-service-connect":" *value*" 형식으로 지정되며, 여기서 *value*는 "true" 또는 "false"입니다.	String
ecs:enable-ebs-volumes	컨텍스트 키는 "ecs:enable-ebs-volumes":" *value*" 형식으로 지정되며, 여기서 *value*는 "true" 또는 "false"입니다.	String
ecs:enable-managed-tags	컨텍스트 키는 "ecs:enable-managed-tags":" *value*" 형식으로 지정되며, 여기서 *value*는 "true" 또는 "false"입니다.	String
ecs:enable-vpc-lattice	컨텍스트 키는 "ecs:vpc-lattice":" *value*" 형식으로 지정되며, 여기서 *value*는 "true" 또는 "false"입니다.	String
ecs:fargate-ephemeral-storage-kms-key	컨텍스트 키는 "ecs:fargate-ephemeral-storage-kms-key":" *key*" 형식으로 지정됩니다. 여기서 *key*는 AWS KMS 키의 ID입니다.	String

조건 키	설명	평가 유형
ecs:namespace	컨텍스트 키는 "ecs:namespace":" *namespace-arn*" 형식으로 지정되며, 여기서 *namespace-arn* 은 AWS Cloud Map 네임스페이스의 ARN입니다.	ARN, Null
ecs:propagate-tags	컨텍스트 키는 "ecs:propagate-tags":" *value*" 형식으로 지정됩니다. 여기서 *value*는 "TASK_DEFINITION" , "SERVICE" 또는 "NONE"입니다.	String
ecs:service	이 컨텍스트 키는 "ecs:service":" *service-arn* " 형식으로, 여기서 *service-arn* 은 Amazon ECS 서비스의 ARN입니다.	ARN, Null
ecs:task-definition	이 컨텍스트 키는 "ecs:task-definition":" *task-definition-arn* " 형식으로, 여기서 *task-definition-arn* 은 Amazon ECS 태스크 정의의 ARN입니다.	ARN, Null
ecs:subnet	컨텍스트 키는 "ecs:subnet":" *subnets*" 형식으로 지정됩니다. 여기서 *subnets*는 awsvpc 모드를 사용하는 작업 또는 서비스의 서브넷입니다.	String
ecs:task	컨텍스트 키는 "ecs:task":" *task-arn*" 형식으로 지정됩니다. 여기서 *task-arn*은 Amazon ECS 서비스의 ARN입니다.	ARN, Null
ecs:task-cpu	컨텍스트는 "ecs:task-cpu":" *task-cpu*" 형식으로 지정됩니다. 여기서 *task-cpu*는 작업 CPU입니다(1024가 있는 정수 = vCPU 1개).	Integer
ecs:task-memory	컨텍스트 키는 "ecs:task-memory":" *task-memory* " 형식으로 지정됩니다. 여기서 *task-memory* 는 MiB 단위의 작업 메모리입니다.	Integer

Amazon ECS 조건 키 목록을 보려면 서비스 승인 참조의 Amazon Elastic Container Service에 사용되는 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 Amazon Elastic Container Service에서 정의한 작업을 참조하세요.

Amazon ECS 자격 증명 기반 정책 예제를 보려면 Amazon Elastic Container Service의 자격 증명 기반 정책 예 섹션을 참조하세요.

Amazon ECS의 액세스 제어 목록(ACL)

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon ECS를 사용한 속성 기반 액세스 제어(ABAC)

> **Important**
>
> Amazon ECS는 모든 Amazon ECS 리소스에 대한 속성 기반 액세스 제어를 지원합니다. 속성을 사용하여 작업 범위를 지정할 수 있는지 여부를 결정하려면 서비스 승인 참조의 Amazon ECS에서 정의한 작업 테이블을 사용하세요. 먼저 리소스 열에 리소스가 있는지 확인하세요. 그런 다음 조건 키 열을 사용하여 작업/리소스 조합의 키를 확인하세요.

ABAC 지원(정책의 태그): 예

속성 기반 액세스 제어(ABAC)는 속성에 근거하여 권한을 정의하는 권한 부여 전략입니다. AWS에서는 이러한 속성을 태그라고 합니다. IAM 엔티티(사용자 또는 역할) 및 많은 AWS리소스에 태그를 연결할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 위탁자의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

Amazon ECS 리소스 태그 지정에 대한 자세한 정보는 Amazon ECS 리소스 태그 지정 섹션을 참조하세요.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예시는 태그를 기반으로 Amazon ECS Services 설명에서 확인할 수 있습니다.

Amazon ECS에서 임시 자격 증명 사용

임시 자격 증명 지원: 예

일부 AWS 서비스는 임시 자격 증명을 사용하여 로그인할 때 작동하지 않습니다. 임시 보안 인증으로 작동하는 AWS 서비스를 비롯한 추가 정보는 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스를 참조하세요.

사용자 이름과 암호를 제외한 다른 방법을 사용하여 AWS Management Console에 로그인하면 임시 자격 증명을 사용하는 것입니다. 예를 들어, 회사의 Single Sign-On(SSO) 링크를 사용하여 AWS에 액세스하면 해당 프로세스에서 자동으로 임시 자격 증명을 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 자격 증명을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 사용자에서 IAM 역할로 전환(콘솔)을 참조하세요.

AWS CLI 또는 AWS API를 사용하여 임시 자격 증명을 수동으로 만들 수 있습니다 그런 다음 이러한 임시 자격 증명을 사용하여 AWS에 액세스할 수 있습니다. AWS에서는 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성할 것을 권장합니다. 자세한 정보는 IAM의 임시 보안 자격 증명 섹션을 참조하세요.

Amazon ECS에 대한 액세스 세션 전달

전달 액세스 세션(FAS) 지원: 예

IAM 사용자 또는 역할을 사용하여 AWS에서 작업을 수행하는 사람은 위탁자로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 직접 호출하는 위탁자의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

Amazon ECS의 서비스 역할

서비스 역할 지원: 예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 Amazon ECS 기능이 중단될 수 있습니다. Amazon ECS에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집합니다.

Amazon ECS의 서비스 연결 역할

서비스 링크 역할 지원: 예

서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 유형입니다. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 링크 역할은 AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

Amazon ECS Z서비스 연결 역할 생성 또는 관리에 대한 자세한 정보는 Amazon ECS에 대해 서비스 연결 역할 사용 섹션을 참조하세요.

Amazon Elastic Container Service의 자격 증명 기반 정책 예

기본적으로 사용자 및 역할은 Amazon ECS 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 AWS API를 사용해 태스크를 수행할 수 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 맡을 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 ARN 형식을 비롯하여 Amazon ECS에서 정의되는 작업 및 리소스 유형에 대한 자세한 내용은 서비스 승인 참조의 Amazon Elastic Container Service에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

주제

- Amazon ECS 정책 모범 사례
- Amazon ECS 사용자가 자신의 권한을 볼 수 있도록 허용

- Amazon ECS 클러스터 예제
- Amazon ECS 컨테이너 인스턴스 예제
- Amazon ECS 태스크 정의 예제
- Amazon ECS 태스크 예제 실행
- Amazon ECS 작업 예제 시작
- Amazon ECS 작업 예제 나열 및 설명
- Amazon ECS 서비스 예제 생성
- Amazon ECS 서비스 예제 업데이트
- 태그를 기반으로 Amazon ECS Services 설명
- Amazon ECS 서비스 연결 네임스페이스 재정의 거부 예제

Amazon ECS 정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 Amazon ECS 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책으로 시작하고 최소 권한을 향해 나아가기 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. AWS 계정에서 사용할 수 있습니다. 사용 사례에 고유한 AWS 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS CloudFormation와 같이, 특정 AWS 서비스를 통해 사용되는 경우에만 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하

여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.

- 다중 인증(MFA) 필요 – AWS 계정에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우, 추가 보안을 위해 MFA를 설정합니다. API 작업을 직접 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

Amazon ECS 사용자가 자신의 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로 이 태스크를 완료할 수 있는 권한이 포함됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon ECS 클러스터 예제

다음 IAM 정책은 클러스터를 생성하고 나열할 수 있는 권한을 허용합니다. `CreateCluster` 및 `ListClusters` 태스크는 어떤 리소스도 수락하지 않으므로 리소스 정의는 * 모든 리소스에 대해 설정됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:CreateCluster",
                "ecs:ListClusters"
            ],
            "Resource": ["*"]
        }
    ]
}
```

다음 IAM 정책은 특정 클러스터를 설명하고 삭제할 수 있는 권한을 허용합니다. `DescribeClusters` 및 `DeleteCluster` 태스크는 클러스터 ARN을 리소스로 수락합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeClusters",
                "ecs:DeleteCluster"
            ],
            "Resource": ["arn:aws:ecs:us-east-1:<aws_account_id>:cluster/
<cluster_name>"]
        }
    ]
```

```
}
```

한 사용자 또는 그룹만 특정 클러스터에서 태스크를 수행하도록 허용하는 다음 IAM 정책을 해당 사용자 또는 그룹에 연결할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecs:Describe*",
                "ecs:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "ecs:DeleteCluster",
                "ecs:DeregisterContainerInstance",
                "ecs:ListContainerInstances",
                "ecs:RegisterContainerInstance",
                "ecs:SubmitContainerStateChange",
                "ecs:SubmitTaskStateChange"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/default"
        },
        {
            "Action": [
                "ecs:DescribeContainerInstances",
                "ecs:DescribeTasks",
                "ecs:ListTasks",
                "ecs:UpdateContainerAgent",
                "ecs:StartTask",
                "ecs:StopTask",
                "ecs:RunTask"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "ArnEquals": {"ecs:cluster": "arn:aws:ecs:us-
east-1:<aws_account_id>:cluster/default"}
```

```
            }
        }
    ]
}
```

Amazon ECS 컨테이너 인스턴스 예제

컨테이너 인스턴스 등록은 Amazon ECS 에이전트에 의해 처리되지만 사용자가 수동으로 클러스터에서 인스턴스 등록을 취소할 수 있도록 하는 것이 좋은 경우가 있을 수 있습니다. 컨테이너 인스턴스가 잘못된 클러스터에 실수로 등록되었거나 태스크가 여전히 실행되고 있는 상태에서 인스턴스가 종료된 경우가 그렇습니다.

다음 IAM 정책은 사용자가 지정한 클러스터의 컨테이너 인스턴스 목록을 보고 등록을 해제할 수 있도록 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:DeregisterContainerInstance",
                "ecs:ListContainerInstances"
            ],
            "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>"]
        }
    ]
}
```

다음 IAM 정책은 사용자가 지정된 클러스터의 지정된 컨테이너 인스턴스를 기술할 수 있도록 합니다. 이 권한을 클러스터의 모든 컨테이너 인스턴스까지 열기 위해 컨테이너 인스턴스 UUID를 *로 교체할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:DescribeContainerInstances"],
            "Condition": {
```

```
                "ArnEquals": {"ecs:cluster":
 "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"}
            },
            "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:container-instance/
<cluster_name>/<container_instance_UUID>"]
        }
    ]
}
```

Amazon ECS 태스크 정의 예제

태스크 정의 IAM 정책은 리소스 수준 권한을 지원하지 않지만 다음 IAM 정책은 사용자가 태스크 정의를 등록, 나열, 설명하도록 허용합니다.

콘솔을 사용하는 경우 `CloudFormation: CreateStack`을 `Action`으로 추가해야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RegisterTaskDefinition",
                "ecs:ListTaskDefinitions",
                "ecs:DescribeTaskDefinition"
            ],
            "Resource": ["*"]
        }
    ]
}
```

Amazon ECS 태스크 예제 실행

`RunTask`에 대한 리소스가 태스크 정의입니다. 사용자가 태스크 정의를 실행할 수 있는 클러스터를 제한하기 위해 `Condition` 블록에서 클러스터를 지정할 수 있습니다. 이 방법의 장점은 적절한 액세스를 허용하기 위해 리소스에서 태스크 정의와 클러스터를 모두 나열하지 않아도 된다는 점입니다. 전자나 후자 또는 둘 다 적용할 수 있습니다.

다음 IAM 정책은 특정 클러스터에서 특정 태스크 정의의 어떤 개정이든 실행할 수 있는 권한을 허용합니다.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:RunTask"],
            "Condition": {
                "ArnEquals": {"ecs:cluster":
 "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"}
            },
            "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:task-definition/
<task_family>:*"]
        }
    ]
}
```

Amazon ECS 작업 예제 시작

StartTask에 대한 리소스가 태스크 정의입니다. 사용자가 태스크 정의를 시작할 수 있는 클러스터와 컨테이너 인스턴스를 제한하기 위해 Condition 블록에서 이를 지정할 수 있습니다. 이 방법의 장점은 적절한 액세스를 허용하기 위해 리소스에서 태스크 정의와 클러스터를 모두 나열하지 않아도 된다는 점입니다. 전자나 후자 또는 둘 다 적용할 수 있습니다.

다음 IAM 정책은 특정 클러스터와 특정 컨테이너 인스턴스에서 특정 태스크 정의의 어떤 개정이든 시작할 수 있는 권한을 허용합니다.

> **Note**
>
> 이 예에서 StartTask 또는 다른 AWS CLI SDK로 AWS API를 호출할 때 태스크 정의 개정을 지정해야 Resource 매핑이 일치합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:StartTask"],
            "Condition": {
                "ArnEquals": {
                    "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>",
```

```
                    "ecs:container-instances":
 ["arn:aws:ecs:<region>:<aws_account_id>:container-instance/<cluster_name>/
<container_instance_UUID>"]
                }
            },
            "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:task-definition/
<task_family>:*"]
        }
    ]
}
```

Amazon ECS 작업 예제 나열 및 설명

다음 IAM 정책은 사용자가 특정 클러스터에 대해 태스크를 나열하도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:ListTasks"],
            "Condition": {
                "ArnEquals": {"ecs:cluster":
 "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"}
            },
            "Resource": ["*"]
        }
    ]
}
```

다음 IAM 정책은 사용자가 지정된 클러스터에서 지정된 태스크를 설명하도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:DescribeTasks"],
            "Condition": {
                "ArnEquals": {"ecs:cluster":
 "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"}
            },
```

```
            "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:task/<cluster_name>/
<task_UUID>"]
        }
    ]
}
```

Amazon ECS 서비스 예제 생성

다음 IAM 정책은 사용자가 AWS Management Console에서 Amazon ECS 서비스를 생성하도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:Describe*",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:RegisterScalableTarget",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm",
                "ecs:List*",
                "ecs:Describe*",
                "ecs:CreateService",
                "elasticloadbalancing:Describe*",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:ListAttachedRolePolicies",
                "iam:ListRoles",
                "iam:ListGroups",
                "iam:ListUsers"
            ],
            "Resource": ["*"]
        }
    ]
}
```

Amazon ECS 서비스 예제 업데이트

다음 IAM 정책은 사용자가 AWS Management Console에서 Amazon ECS 서비스를 업데이트하도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:Describe*",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:RegisterScalableTarget",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm",
                "ecs:List*",
                "ecs:Describe*",
                "ecs:UpdateService",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:ListAttachedRolePolicies",
                "iam:ListRoles",
                "iam:ListGroups",
                "iam:ListUsers"
            ],
            "Resource": ["*"]
        }
    ]
}
```

태그를 기반으로 Amazon ECS Services 설명

자격 증명 기반 정책의 조건을 사용하여 태그를 기반으로 Amazon ECS 리소스에 대한 액세스를 제어할 수 있습니다. 이 예제에서는 서비스 설명을 허용하는 정책을 생성할 수 있는 방법을 보여 줍니다. 하지만 `Owner` 서비스 태그가 해당 사용자의 사용자 이름 값을 가지고 있는 경우에만 권한이 부여됩니다. 이 정책은 콘솔에서 이 태스크를 완료하는 데 필요한 권한도 부여합니다.

JSON

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

```
            "Sid": "DescribeServices",
            "Effect": "Allow",
            "Action": "ecs;:DescribeServices",
            "Resource": "*"
        },
        {
            "Sid": "ViewServiceIfOwner",
            "Effect": "Allow",
            "Action": "ecs:DescribeServices",
            "Resource": "arn:aws:ecs:*:*:service/*",
            "Condition": {
                "StringEquals": {"ecs:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

이 정책을 계정의 IAM 사용자에게 연결할 수 있습니다. `richard-roe`라는 사용자가 Amazon ECS 서비스를 설명하려고 할 경우 서비스에 `Owner=richard-roe` 또는 `owner=richard-roe` 태그가 지정되어야 합니다. 그렇지 않으면 액세스가 거부됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 조건 태그 키 `Owner`는 `Owner` 및 `owner` 모두와 일치합니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

Amazon ECS 서비스 연결 네임스페이스 재정의 거부 예제

다음 IAM 정책은 사용자가 서비스 구성의 기본 Service Connect 네임스페이스를 재정의하는 것을 거부합니다. 기본 네임스페이스는 클러스터에 설정됩니다. 그러나 서비스 구성에서 이를 재정의할 수 있습니다. 일관성을 위해 모든 새 서비스가 동일한 네임스페이스를 사용하도록 설정하는 것이 좋습니다. 서비스가 특정 네임스페이스를 사용하도록 하려면 다음 컨텍스트 키를 사용합니다. 다음 예제에서 `<region>`, `<aws_account_id>`, `<cluster_name>` 및 `<namespace_id>`를 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:CreateService",
                "ecs:UpdateService"
            ],
            "Condition": {
```

```
                "ArnEquals": {
                    "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>",
                    "ecs:namespace":
 "arn:aws:servicediscovery:<region>:<aws_account_id>:namespace/<namespace_id>"
                }
            },
            "Resource": "*"
        }
    ]
}
```

Amazon Elastic Container Service용 AWS 관리형 정책

사용자, 그룹 또는 역할에 권한을 추가할 때 정책을 직접 작성하는 것보다 AWS관리형 정책을 사용하는 것이 더욱 편리합니다. 팀에 필요한 권한만 제공하는 IAM 고객 관리형 정책을 생성하려면 시간과 전문 지식이 필요합니다. 빨리 시작하려면 AWS관리형 정책을 사용할 수 있습니다. 이러한 정책은 일반적인 사용 사례에 적용되며 AWS계정에서 사용할 수 있습니다. AWS 관리형 정책에 대한 자세한 정보는 IAM 사용 설명서에서 AWS관리형 정책을 참조하세요.

AWS 서비스 유지 관리 및 AWS관리형 정책 업데이트입니다. AWS 관리형 정책에서는 권한을 변경할 수 없습니다. 서비스에서 때때로 추가 권한을 AWS 관리형 정책에 추가하여 새로운 기능을 지원합니다. 이 유형의 업데이트는 정책이 연결된 모든 ID(사용자, 그룹 및 역할)에 적용됩니다. 서비스는 새로운 기능이 시작되거나 새 태스크를 사용할 수 있을 때 AWS 관리형 정책에 업데이트됩니다. 서비스는 AWS관리형 정책에서 권한을 제거하지 않기 때문에 정책 업데이트로 인해 기존 권한이 손상되지 않습니다.

또한 AWS는 여러 서비스의 직무에 대한 관리형 정책을 지원합니다. 예를 들어 ReadOnlyAccess라는 이름의 AWS 관리형 정책은 모든 AWS 서비스 및 리소스에 대한 읽기 전용 액세스 권한을 제공합니다. 서비스에서 새 기능을 시작하면 AWS가 새 작업 및 리소스에 대한 읽기 전용 권한을 추가합니다. 직무 정책의 목록과 설명은 IAM 사용 설명서의 직무에 관한 AWS 관리형 정책을 참조하세요.

Amazon ECS와 Amazon ECR은 사용자, 그룹, 역할, Amazon EC2 인스턴스, Amazon ECS 작업에 연결하여 리소스 및 API 작업을 다양한 수준으로 제어할 수 있는 여러 개의 관리형 정책과 신뢰 관계를 제공합니다. 이러한 정책은 직접 적용할 수도 있고, 사용자 고유의 정책을 생성하기 위한 시작 지점으로 정책을 사용할 수도 있습니다. Amazon ECR 관리형 정책에 대한 자세한 정보는 Amazon ECR 관리형 정책을 참조하세요.

AmazonECS_FullAccess

`AmazonECS_FullAccess` 정책을 IAM 보안 인증에 연결할 수 있습니다. 이 정책은 Amazon ECS 리소스에 대한 관리 액세스 권한을 부여하고 모든 Amazon ECS 기능을 사용하기 위해 Amazon ECS를 통합하는 AWS 서비스에 대한 IAM 자격 증명(예: 사용자, 그룹 또는 역할) 액세스 권한을 부여합니다. 이 정책을 사용하면 AWS Management Console에서 사용할 수 있는 모든 Amazon ECS 기능에 액세스할 수 있습니다.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AmazonECS_FullAccess를 참조하세요.

AmazonECSInfrastructureRolePolicyForVolumes

`AmazonECSInfrastructureRolePolicyForVolumes` 관리형 IAM 정책을 IAM 엔터티에 연결할 수 있습니다.

정책에서는 사용자를 대신하여 AWS API를 직접적으로 호출하기 위해 Amazon ECS에 필요한 권한을 부여합니다. Amazon ECS 태스크 및 서비스를 시작할 때 볼륨 구성과 함께 제공하는 IAM 역할에 이 정책을 연결할 수 있습니다. 이 역할을 통해 Amazon ECS는 태스크에 연결된 볼륨을 관리할 수 있습니다. 자세한 내용은 Amazon ECS 인프라 IAM 역할을 참조하세요.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AmazonECSInfrastructureRolePolicyForVolumes를 참조하세요.

AmazonEC2ContainerServiceforEC2Role

`AmazonEC2ContainerServiceforEC2Role` 정책을 IAM 보안 인증에 연결할 수 있습니다. 이 정책은 Amazon ECS 컨테이너 인스턴스가 사용자를 대신하여 AWS를 호출할 수 있도록 하는 관리 권한을 부여합니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.

Amazon ECS는 Amazon EC2 인스턴스 또는 외부 인스턴스에 대해 사용자를 대신해 태스크를 수행할 수 있는 서비스 역할에 이 정책을 연결합니다.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AmazonEC2ContainerServiceforEC2Role을 참조하세요.

고려 사항

`AmazonEC2ContainerServiceforEC2Role` 관리형 IAM 정책을 사용할 때 다음 권장 사항 및 고려 사항을 고려해야 합니다.

- 최소 권한 부여에 대한 표준 보안 권고 사항에 따라 특정 요구 사항에 맞게 `AmazonEC2ContainerServiceforEC2Role` 관리형 정책을 수정할 수 있습니다. 자신의 사용 사

례에서 관리형 정책에서 부여된 권한이 필요하지 않은 경우 사용자 지정 정책을 생성하고 필요한 권한만 추가합니다. 예를 들어 `UpdateContainerInstancesState` 권한은 스팟 인스턴스 드레이닝에 제공됩니다. 자신의 사용 사례에 이 권한이 필요하지 않은 경우 사용자 지정 정책으로 해당 권한을 제외하세요.

- 컨테이너 인스턴스에서 실행되는 컨테이너는 인스턴스 메타데이터를 통해 컨테이너 인스턴스 역할에 제공된 모든 권한에 액세스할 수 있습니다. 컨테이너 인스턴스 역할의 권한을 관리형 `AmazonEC2ContainerServiceforEC2Role` 정책에서 제공된 최소 권한 목록으로 제한하는 것이 좋습니다. 태스크의 컨테이너에 나열되지 않은 추가 권한이 필요할 경우, 이러한 태스크에 자체 IAM 역할을 제공하는 것이 좋습니다. 자세한 정보는 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.

  `docker0` 브리지의 컨테이너가 컨테이너 인스턴스 역할에 제공된 권한에 액세스하지 못하도록 할 수 있습니다. 컨테이너 인스턴스에서 다음iptables 명령을 실행하여 Amazon ECS 작업 IAM 역할에서 제공하는 권한을 계속 허용하면서 이 태스크를 수행할 수 있습니다. 컨테이너는 이 규칙을 적용한 상태에서는 인스턴스 메타데이터를 쿼리할 수 없습니다. 이 명령은 기본 Docker 브리지 구성을 가정하며, `host` 네트워크 모드를 사용하는 컨테이너에는 적용되지 않습니다. 자세한 정보는 네트워크 모드 섹션을 참조하세요.

  ```
  sudo yum install -y iptables-services; sudo iptables --insert DOCKER USER 1 --in-interface docker+ --destination 169.254.169.254/32 --jump DROP
  ```

  재부팅 시 이 iptables 규칙이 유지되게 하려면 규칙을 컨테이너 인스턴스에 저장해야 합니다. Amazon ECS 최적화 AMI의 경우 다음 명령을 사용합니다. 다른 운영 체제의 경우 해당 운영 체제 설명서를 참조하세요.

  - Amazon ECS 최적화 Amazon Linux 2 AMI:

    ```
    sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now iptables
    ```

  - Amazon ECS 최적화 Amazon Linux AMI의 경우:

    ```
    sudo service iptables save
    ```

AmazonEC2ContainerServiceEventsRole

`AmazonEC2ContainerServiceEventsRole` 정책을 IAM 보안 인증에 연결할 수 있습니다. 이 정책은 Amazon EventBridge(이전의 CloudWatch Events)가 사용자를 대신하여 태스크를 실행할 수 있도

록 허용하는 권한을 부여합니다. 이 정책은 예약된 태스크를 생성할 때 지정한 IAM 역할에 연결할 수 있습니다. 자세한 내용은 Amazon ECS EventBridge IAM 역할 섹션을 참조하세요.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AmazonEC2ContainerServiceEventsRole을 참조하세요.

AmazonECSTaskExecutionRolePolicy

`AmazonECSTaskExecutionRolePolicy` 관리형 IAM 정책은 사용자를 대신하여 AWS API를 호출하기 위해 Amazon ECS 컨테이너 에이전트 및 AWS Fargate 컨테이너 에이전트에 필요한 권한을 부여합니다. 이 정책은 태스크 실행 IAM 역할에 추가할 수 있습니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AmazonECSTaskExecutionRolePolicy를 참조하세요.

AmazonECSServiceRolePolicy

`AmazonECSServiceRolePolicy` 관리형 IAM 정책을 사용하면 Amazon Elastic Container Service에서 클러스터를 관리할 수 있습니다. 이 정책은 AWSServiceRoleForECS 서비스 연결 역할에 추가할 수 있습니다.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AmazonECSServiceRolePolicy를 참조하세요.

AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecur

`AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity` 정책을 IAM 엔터티에 연결할 수 있습니다. 이 정책에서는 사용자를 대신하여 Amazon ECS Service Connect TLS 기능을 관리하는 데 필요한 AWS Private Certificate Authority, Secrets Manager 및 기타 AWS 서비스에 대한 관리 액세스 권한을 부여합니다.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity를 참조하세요.

AWSApplicationAutoscalingECSServicePolicy

`AWSApplicationAutoscalingECSServicePolicy`를 IAM 엔터티에 연결할 수 없습니다. 이 정책은 Application Auto Scaling에서 사용자를 대신하여 태스크를 수행할 수 있도록 서비스 연결 역할에 연결됩니다. 자세한 정보는 Application Auto Scaling에 대한 서비스 연결 역할을 참조하세요.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AWSApplicationAutoscalingECSServicePolicy를 참조하세요.

AWSCodeDeployRoleForECS

AWSCodeDeployRoleForECS를 IAM 엔터티에 연결할 수 없습니다. 이 정책은 CodeDeploy에서 사용자를 대신하여 태스크를 수행할 수 있도록 서비스 연결 역할에 연결됩니다. 자세한 정보는 AWS CodeDeploy 사용 설명서의 CodeDeploy에 대한 서비스 역할 생성을 참조하세요.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AWSCodeDeployRoleForECS를 참조하세요.

AWSCodeDeployRoleForECSLimited

AWSCodeDeployRoleForECSLimited를 IAM 엔터티에 연결할 수 없습니다. 이 정책은 CodeDeploy에서 사용자를 대신하여 태스크를 수행할 수 있도록 서비스 연결 역할에 연결됩니다. 자세한 정보는 AWS CodeDeploy 사용 설명서의 CodeDeploy에 대한 서비스 역할 생성을 참조하세요.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AWSCodeDeployRoleForECSLimited를 참조하세요.

AmazonECSInfrastructureRolePolicyForLoadBalancers

AmazonECSInfrastructureRolePolicyForLoadBalancers 정책을 IAM 엔터티에 연결할 수 있습니다. 이 정책은 Amazon ECS가 사용자를 대신해 Elastic Load Balancing 리소스를 관리할 수 있도록 하는 권한을 부여합니다. 정책에는 다음이 포함됩니다.

- 리스너, 규칙, 대상 그룹, 대상 상태를 설명하는 읽기 전용 권한
- 대상 그룹에 대상을 등록 및 등록 취소하는 권한
- Application Load Balancer 및 Network Load Balancer에 대한 리스너를 수정하는 권한
- Application Load Balancer에 대한 규칙을 수정하는 권한

이러한 권한을 통해 Amazon ECS는 서비스가 생성되거나 업데이트될 때 로드 밸런서 구성을 자동으로 관리하여 컨테이너로 트래픽을 적절하게 라우팅하도록 보장할 수 있습니다.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AmazonECSInfrastructureRolePolicyForLoadBalancers를 참조하세요.

AmazonECSInfrastructureRolePolicyForVpcLattice

AmazonECSInfrastructureRolePolicyForVpcLattice 정책을 IAM 엔터티에 연결할 수 있습니다. 이 정책에서는 사용자를 대신하여 Amazon ECS 워크로드에서 VPC Lattice 특성을 관리하는 데 필요한 다른 AWS 서비스 리소스에 대한 액세스 권한이 제공됩니다.

이 정책의 권한을 보려면 AWS 관리형 정책 참조의 AmazonECSInfrastructureRolePolicyForVpcLattice를 참조하세요.

사용자를 대신하여 Amazon ECS 워크로드에서 VPC Lattice 특성을 관리하는 데 필요한 다른 AWS 서비스 리소스에 대한 액세스 권한이 제공됩니다.

AWS 관리형 정책에 대한 Amazon ECS 업데이트

이 서비스가 이러한 변경 내용을 추적하기 시작한 이후부터 Amazon ECS의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 Amazon ECS 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
the section called “AmazonEC2ContainerServiceforEC2Role”에 권한 추가	`ecs:ListTagsForResource` 권한을 포함하도록 `AmazonEC2ContainerServiceforEC2Role` 관리형 IAM 정책이 업데이트되었습니다. 이 권한을 통해 Amazon ECS 에이전트는 태스크 메타데이터 엔드포인트(`${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags`)를 통해 태스크 및 컨테이너 인스턴스 태그를 검색할 수 있습니다.	2025년 8월 4일
AmazonECSInfrastructureRolePolicyForLoadBalancers에 권한을 추가합니다.	`AmazonECSInfrastructureRolePolicyForLoadBalancers` 관리형 IAM 정책이 대상 그룹을 설명, 등록, 등록 취소하기 위한 새 권한으로 업데이트되었습니다.	2025년 7월 25일
새 the section called “AmazonECSInfrastructureRolePolicyFor	Amazon ECS 워크로드와 연결된 로드 밸런서를 관리하는 데 필요한 다른 AWS 서비스 리소	2025년 7월 15일

변경 사항	설명	날짜
LoadBalancers ” IAM 정책 추가	스에 대한 액세스를 제공하는 새로운 AmazonECSInfrastructureRolePolicyForLoadBalancers 정책을 추가했습니다.	
AmazonECSServiceRolePolicy에 권한을 추가합니다.	AmazonECSServiceRolePolicy 관리형 IAM 정책은 Amazon ECS가 관리하는 서비스의 AWS Cloud Map 서비스 속성을 업데이트할 수 있는 새 AWS Cloud Map 권한으로 업데이트되었습니다.	2025년 7월 15일
AmazonECSServiceRolePolicy에 권한 추가	AmazonECSServiceRolePolicy 관리형 IAM 정책은 Amazon ECS가 관리하는 서비스의 AWS Cloud Map 서비스 속성을 업데이트할 수 있는 새 AWS Cloud Map 권한으로 업데이트되었습니다.	2025년 6월 24일
AmazonECSInfrastructureRolePolicyForVolumes에 권한 추가	AmazonECSInfrastructureRolePolicyForVolumes 정책이 업데이트되어 ec2:DescribeInstances 권한이 추가되었습니다. 이 권한은 동일한 컨테이너 인스턴스에서 실행되는 Amazon ECS 태스크에 연결된 Amazon EBS 볼륨의 디바이스 이름 충돌을 방지하는 데 도움이 됩니다.	2025년 6월 2일

변경 사항	설명	날짜
새 AmazonECSInfrastructureRolePolicyForVpcLattice 추가	사용자를 대신하여 Amazon ECS 워크로드에서 VPC Lattice 특성을 관리하는 데 필요한 다른 AWS 서비스 리소스에 대한 액세스 권한이 제공됩니다.	2024년 11월 18일
AmazonECSInfrastructureRolePolicyForVolumes에 권한 추가	고객이 스냅샷에서 Amazon EBS 볼륨을 생성할 수 있도록 `AmazonECSInfrastructureRolePolicyForVolumes` 정책이 업데이트되었습니다.	2024년 10월 10일
the section called “AmazonECS_FullAccess”에 권한 추가	`ecsInfrastructureRole` 이라는 역할에 대한 IAM 역할 권한 `iam:PassRole` 을 추가하도록 `AmazonECS_FullAccess` 정책이 업데이트되었습니다. 이 역할은 AWS Management Console에서 생성한 기본 IAM 역할로, Amazon ECS가 ECS 작업에 연결된 Amazon EBS 볼륨을 관리할 수 있도록 하는 ECS 인프라 역할로 사용하기 위한 것입니다.	2024년 8월 13일

변경 사항	설명	날짜
새 AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity 정책 추가	AWS KMS, AWS Private Certificate Authority, Secrets Manager에 대한 관리 액세스 권한을 제공하고 Amazon ECS Service Connect TLS가 올바르게 작동하도록 지원하는 새 AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity 정책이 추가되었습니다.	2024년 1월 22일
새 AmazonECSInfrastructureRolePolicyForVolumes 정책 추가	`AmazonECSInfrastructureRolePolicyForVolumes` 정책이 추가되었습니다. 정책에서는 Amazon ECS가 Amazon ECS 워크로드에 연결된 Amazon EBS 볼륨을 관리하기 위해 AWS API 직접 호출을 수행하는 데 필요한 권한을 부여합니다.	2024년 1월 11일
AmazonECSServiceRolePolicy에 권한 추가	`AmazonECSServiceRolePolicy` 관리형 IAM 정책이 새 `events` 권한과 추가적인 `autoscaling` 및 `autoscaling-plans` 권한으로 업데이트되었습니다.	2023년 12월 4일
AmazonEC2ContainerServiceEventsRole에 권한 추가	`AmazonECSServiceRolePolicy` 관리형 IAM 정책이 AWS Cloud Map `DiscoverInstancesRevision` API 작업에 대한 액세스를 허용하도록 업데이트되었습니다.	2023년 10월 4일

변경 사항	설명	날짜
AmazonEC2ContainerServiceforEC2Role에 권한 추가	새로 생성된 클러스터 및 등록된 컨테이너 인스턴스로만 권한을 제한하는 조건이 포함된 `ecs:TagResource` 권한을 추가하도록 `AmazonEC2ContainerServiceforEC2Role` 정책이 수정되었습니다.	2023년 3월 6일
the section called “AmazonECS_FullAccess”에 권한 추가	새로 생성된 로드 밸런서, 대상 그룹, 규칙 및 생성된 리스너로만 권한을 제한하는 조건이 포함된 `elasticloadbalancing:AddTags` 권한을 추가하도록 `AmazonECS_FullAccess` 정책이 수정되었습니다. 이 권한은 이미 생성된 Elastic Load Balancing 리소스에 태그를 추가하는 것을 허용하지 않습니다.	2023년 1월 4일
Amazon ECS 변경 사항 추적 시작	Amazon ECS가 AWS 관리형 정책에 대한 변경 내용 추적을 시작했습니다.	2021년 6월 8일

Amazon Elastic Container Service의 AWS 관리형 IAM 정책 단계적 폐지

단계적으로 폐지되는 AWS 관리형 IAM 정책은 다음과 같습니다. 이제 이 정책은 업데이트된 정책으로 대체됩니다. 업데이트된 정책을 사용하려면 사용자 또는 역할을 업데이트하는 것이 좋습니다.

AmazonEC2ContainerServiceFullAccess

> **Important**
>
> `iam:passRole` 권한을 이용한 보안 결과에 응답하여 `AmazonEC2ContainerServiceFullAccess` 관리형 IAM 정책이 2021년 1월 29일 현재 단

계적으로 폐지되었습니다. 이 권한은 계정 역할에 대한 자격 증명을 비롯하여 모든 리소스에 대한 액세스 권한을 부여합니다. 정책이 단계적으로 폐지되었으므로 새 사용자 또는 역할에 정책을 연결할 수 없습니다. 이미 정책이 연결된 사용자 또는 역할은 정책을 계속 사용할 수 있습니다. 하지만 사용자 또는 역할을 업데이트하여 AmazonECS_FullAccess 관리형 정책을 대신 사용하는 것이 좋습니다. 자세한 내용은 AmazonECS_FullAccess 관리형 정책으로 마이그레이션 섹션을 참조하세요.

AmazonEC2ContainerServiceRole

Important

AmazonEC2ContainerServiceRole 관리형 IAM 정책이 단계적으로 폐지됩니다. 이제 Amazon ECS 서비스 링크 역할로 대체됩니다. 자세한 정보는 Amazon ECS에 대해 서비스 연결 역할 사용을 참조하세요.

AmazonEC2ContainerServiceAutoscaleRole

Important

AmazonEC2ContainerServiceAutoscaleRole 관리형 IAM 정책이 단계적으로 폐지됩니다. 이제 Amazon ECS의 Application Auto Scaling 서비스 연결 역할로 대체됩니다. 자세한 정보는 Application Auto Scaling 사용 설명서의 Application Auto Scaling 서비스 연결 역할을 참조하세요.

AmazonECS_FullAccess 관리형 정책으로 마이그레이션

iam:passRole 권한을 이용한 보안 결과에 응답하여 AmazonEC2ContainerServiceFullAccess 관리형 IAM 정책이 2021년 1월 29일에 단계적으로 폐지되었습니다. 이 권한은 계정 역할에 대한 자격 증명을 비롯하여 모든 리소스에 대한 액세스 권한을 부여합니다. 정책이 단계적으로 폐지되었으므로 새 그룹, 사용자 또는 역할에 정책을 연결할 수 없습니다. 이미 정책이 연결된 그룹, 사용자 또는 역할은 정책을 계속 사용할 수 있습니다. 그러나 그룹, 사용자 또는 역할을 업데이트하여 AmazonECS_FullAccess 관리형 정책을 대신 사용하는 것이 좋습니다.

AmazonECS_FullAccess 정책에 따라 부여되는 권한에는 ECS를 관리자로 사용하는 데 필요한 전체 권한 목록이 포함되어 있습니다. 현재 AmazonECS_FullAccess 정책에 없는

AmazonEC2ContainerServiceFullAccess 정책에서 부여한 권한을 사용하는 경우 인라인 정책 설명에 추가할 수 있습니다. 자세한 내용은 Amazon Elastic Container Service용 AWS 관리형 정책 섹션을 참조하세요.

현재 AmazonEC2ContainerServiceFullAccess 관리형 IAM 정책을 사용 중인 그룹, 사용자 또는 역할이 있는지 확인하려면 다음 단계를 따르세요. 그런 다음 이전 정책을 분리하고 AmazonECS_FullAccess 정책을 연결하도록 업데이트합니다.

AmazonECS_FullAccess 정책(AWS Management Console)을 사용하도록 그룹, 사용자 또는 역할 업데이트

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택하고 AmazonEC2ContainerServiceFullAccess 정책을 검색하여 선택합니다.
3. 현재 이 정책을 사용하고 있는 IAM 역할이 표시된 정책 사용 탭을 선택합니다.
4. 현재 AmazonEC2ContainerServiceFullAccess 정책을 사용하고 있는 각 IAM 역할을 선택하고 다음 단계에 따라 단계적으로 폐지된 정책을 분리하고 AmazonECS_FullAccess 정책을 연결합니다.
   a. 권한 탭에서 AmazonEC2ContainerServiceFullAccess 정책 옆에 있는 X를 선택합니다.
   b. 권한 추가를 선택합니다.
   c. 기존 정책 직접 연결을 선택하고 S3을 검색한 후 검색 결과에서 AmazonECS_FullAccess를 선택합니다.
   d. 변경 사항을 검토한 후 권한 추가를 선택합니다.
   e. AmazonEC2ContainerServiceFullAccess 정책을 사용하고 있는 각 그룹, 사용자 또는 역할에 대해 이 단계를 반복합니다.

AmazonECS_FullAccess 정책(AWS CLI)을 사용하도록 그룹, 사용자 또는 역할 업데이트

1. generate-service-last-accessed-details 명령을 사용하여 단계적 중단 정책이 마지막으로 사용된 시점에 대한 세부 정보를 포함하는 보고서를 생성합니다.

```
aws iam generate-service-last-accessed-details \
     --arn arn:aws:iam::aws:policy/AmazonEC2ContainerServiceFullAccess
```

   출력 예시:

```
{
    "JobId": "32bb1fb0-1ee0-b08e-3626-ae83EXAMPLE"
}
```

2. get-service-last-accessed-details 명령과 함께 이전 출력의 작업 ID를 사용하여 서비스의 마지막으로 액세스한 보고서를 검색합니다. 이 보고서는 단계적으로 폐지된 정책을 마지막으로 사용한 IAM 엔터티의 Amazon 리소스 이름(ARN)을 표시합니다.

```
aws iam get-service-last-accessed-details \
      --job-id 32bb1fb0-1ee0-b08e-3626-ae83EXAMPLE
```

3. 다음 명령 중 하나를 사용하여 AmazonEC2ContainerServiceFullAccess 정책을 그룹, 사용자 또는 역할에서 분리합니다.

   - detach-group-policy
   - detach-role-policy
   - detach-user-policy

4. 다음 명령 중 하나를 사용하여 AmazonECS_FullAccess 정책을 그룹, 사용자 또는 역할에 연결합니다.

   - attach-group-policy
   - attach-role-policy
   - attach-user-policy

Amazon ECS에 대해 서비스 연결 역할 사용

Amazon Elastic Container Service는 AWS Identity and Access Management(IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Amazon ECS에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon ECS에서 사전 정의하며, 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon ECS를 더 쉽게 설정할 수 있습니다. Amazon ECS에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon ECS만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 IAM으로 작업하는 AWS 서비스를 참조하고 서비스 연결 역할(Service-linked roles) 열에 예(Yes)가 있는 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

Amazon ECS에 대한 서비스 연결 역할 권한

Amazon ECS는 AWSServiceRoleForECS라는 서비스 연결 역할을 사용합니다.

AWSServiceRoleForECS 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `ecs.amazonaws.com`

AmazonECSServiceRolePolicy라는 역할 권한 정책을 사용하면 Amazon ECS가 지정된 리소스에 대해 다음 조치를 완료할 수 있습니다.

- 조치: Amazon ECS 태스크에 대한 `awsvpc` 네트워크 모드를 사용할 때 Amazon ECS가 이 태스크와 연결된 탄력적 네트워크 인터페이스의 수명 주기를 관리합니다. Amazon ECS가 탄력적 네트워크 인터페이스에 추가하는 태그도 포함됩니다.
- 조치: Amazon ECS 서비스로 로드 밸런서를 사용할 경우, Amazon ECS가 로드 밸런서로 리소스의 등록 및 등록 해제를 관리합니다.
- 조치: Amazon ECS 서비스 검색을 사용할 경우, Amazon ECS가 서비스 검색을 작동하기 위한 필수적인 Route 53 및 AWS Cloud Map 리소스를 관리합니다.
- 조치: Amazon ECS 서비스 Auto Scaling을 사용할 경우, Amazon ECS가 필수적인 Auto Scaling 리소스를 관리합니다.
- 조치: Amazon ECS는 Amazon ECS 리소스의 모니터링을 지원하는 CloudWatch 경보와 로그 스트림을 생성 및 관리합니다.
- 조치: Amazon ECS Exec를 사용할 경우, Amazon ECS가 태스크에 대해 Amazon ECS Exec 세션을 시작하는 데 필요한 권한을 관리합니다.
- 조치: Amazon ECS Service Connect를 사용할 때 Amazon ECS는 기능을 사용하는 데 필요한 AWS Cloud Map 리소스를 관리합니다.
- 조치: Amazon ECS 용량 공급자를 사용할 경우, Amazon ECS가 Auto Scaling 그룹 및 이 그룹의 Amazon EC2 인스턴스를 수정하는 데 필요한 권한을 관리합니다.
- 조치: Amazon ECS는 Amazon ECS가 관리하는 서비스의 AWS Cloud Map 서비스 속성을 업데이트할 수 있습니다.

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.

Amazon ECS에 대한 서비스 연결 역할 생성

대부분의 경우 서비스 연결 역할을 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 클러스터를 생성하거나 서비스를 생성 또는 업데이트하는 경우, Amazon ECS가 사용자를 대신해 서비스에 연결된 역할을 생성합니다. 클러스터를 생성한 후 AWSServiceRoleForECS 역할이 표시되지 않는 경우, 다음을 수행하여 문제를 해결하세요.

- Amazon ECS가 사용자 대신 서비스 연결 역할을 생성, 편집 또는 삭제할 수 있도록 권한을 확인 및 구성합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.
- 클러스터 생성 작업을 재시도하거나 서비스 연결 역할을 수동으로 생성합니다.

  IAM 콘솔을 사용하여 AWSServiceRoleForECS 서비스 연결 역할을 생성할 수 있습니다. AWS CLI 또는 AWS API에서 `ecs.amazonaws.com` 서비스 이름의 서비스 연결 역할을 생성합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 생성을 참조하세요.

> **Important**
>
> 이러한 서비스 연결 역할은 해당 역할이 지원하는 기능을 사용하는 다른 서비스에서 작업을 완료했을 경우 계정에 나타날 수 있습니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 클러스터를 생성하거나 서비스를 생성 또는 업데이트하는 경우, Amazon ECS는 다시 사용자를 대신해 서비스에 연결된 역할을 생성합니다.

이 서비스 연결 역할을 삭제하는 경우 동일한 IAM 프로세스를 사용하여 역할을 다시 생성할 수 있습니다.

Amazon ECS에 대한 서비스 연결 역할 편집

Amazon ECS는 AWSServiceRoleForECS 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 업데이트를 참조하세요.

Amazon ECS에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 단, 서비스 링크 역할에 대한 리소스를 먼저 정리해야 수동으로 삭제할 수 있습니다.

> **Note**
>
> 리소스를 삭제하려고 할 때 Amazon ECS 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

서비스 연결 역할에 활성 세션이 있는지 확인하는 방법

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택한 다음 확인란이 아닌 AWSServiceRoleForECS 이름을 선택합니다.
3. 요약 페이지에서 액세스 고문을 선택하고 서비스 연결 역할의 최근 활동을 검토합니다.

> **Note**
>
> Amazon ECS에서 AWSServiceRoleForECS 역할을 사용하는지 잘 모를 경우에는 역할을 삭제할 수 있습니다. 서비스에서 역할을 사용하는 경우에는 삭제가 안 되어 역할이 사용 중인 리전을 볼 수 있습니다. 역할이 사용 중인 경우에는 세션이 종료될 때까지 기다렸다가 역할을 삭제해야 합니다. 서비스 연결 역할에 대한 세션은 취소할 수 없습니다.

AWSServiceRoleForIVSRecordToS3 서비스 연결 역할에서 사용하는 Amazon ECS 리소스를 제거하려면

AWSServiceRoleForECS 역할을 삭제하려면 먼저 모든 AWS 리전에서 Amazon ECS 클러스터를 모두 삭제해야 합니다.

1. 모든 리전에서 모든 Amazon ECS 서비스의 원하는 개수를 0으로 조정한 후 해당 서비스를 삭제합니다. 자세한 내용은 Amazon ECS 서비스 업데이트 및 콘솔을 사용하여 Amazon ECS 서비스 삭제(을)를 참조하세요.
2. 모든 리전에서 모든 클러스터의 모든 컨테이너 인스턴스를 강제로 등록 취소합니다. 자세한 정보는 Amazon ECS 컨테이너 인스턴스 등록 취소 섹션을 참조하세요.

3. 모든 리전에서 Amazon ECS 클러스터를 모두 삭제합니다. 자세한 내용은 Amazon ECS 클러스터 삭제 섹션을 참조하세요.

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하려면 다음을 수행하세요.

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 AWSServiceRoleForECS 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하십시오.

Amazon ECS 서비스 연결 역할을 지원하는 리전

Amazon ECS는 서비스가 제공되는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용을 알아보려면 AWS service endpoints(서비스 엔드포인트)를 참조하세요.

Amazon ECS에 대한 IAM 역할

IAM 역할은 계정에 생성할 수 있는, 특정 권한을 지닌 IAM 자격 증명입니다. Amazon ECS에서는 역할을 생성하여 컨테이너 또는 서비스와 같은 Amazon ECS 리소스에 권한을 부여할 수 있습니다.

Amazon ECS에 필요한 역할은 작업 정의 시작 유형 및 사용하는 기능에 따라 다릅니다. 다음 테이블을 사용하여 Amazon ECS에 필요한 IAM 역할을 결정합니다.

역할	정의	필요한 경우	추가 정보
태스크 실행 역할	이 역할을 통해 Amazon ECS가 사용자를 대신해 다른 AWS 서비스를 사용할 수 있습니다.	작업은 AWS Fargate 또는 외부 인스턴스에 호스팅됩니다. 그리고, • Amazon ECR 프라이빗 리포지토리에서 컨테이너 이미지를 가져옵니다. • 작업을 실행하는 계정과 다른 계정의 Amazon ECR 프라이빗 리포지토리에서 컨테이너 이미지를 가져옵니다. • `awslogs` 로그 드라이버를 사용해	Amazon ECS 태스크 실행 IAM 역할

역할	정의	필요한 경우	추가 정보
		CloudWatch Logs에 컨테이너 로그를 보냅니다. 작업은 AWS Fargate 또는 Amazon EC2 인스턴스에 호스팅됩니다. 그리고, • 프라이빗 레지스트리 인증을 사용합니다. • Runtime Monitoring을 사용합니다. • 작업 정의에서 Secrets Manager 비밀 또는 AWS Systems Manager Parameter Store 파라미터를 사용하여 민감한 데이터를 참조합니다.	
태스크 역할	이 역할을 통해 컨테이너의 애플리케이션 코드가 다른 AWS 서비스를 사용할 수 있습니다.	애플리케이션은 Amazon S3와 같은 다른 AWS 서비스에 액세스합니다.	Amazon ECS 작업 IAM 역할
컨테이너 인스턴스 역할	이 역할을 통해 EC2 인스턴스 또는 외부 인스턴스를 클러스터에 등록할 수 있습니다.	작업은 Amazon EC2 인스턴스 또는 외부 인스턴스에 호스팅됩니다.	Amazon ECS 컨테이너 인스턴스 IAM 역할

역할	정의	필요한 경우	추가 정보
Amazon ECS Anywhere 역할	이 역할을 통해 외부 인스턴스에서 AWS API에 액세스할 수 있습니다.	작업은 외부 인스턴스에 호스팅됩니다.	Amazon ECS Anywhere IAM 역할
로드 밸런서에 대한 Amazon ECS 인프라 역할	이 역할을 통해 Amazon ECS는 블루/그린 배포에서 사용자를 대신해 클러스터의 로드 밸런서 리소스를 관리할 수 있습니다.	또는 Amazon RDS 블루/그린 배포를 사용하길 원합니다.	로드 밸런서에 대한 Amazon ECS 인프라 IAM 역할
Amazon ECS CodeDeploy 역할	이 역할을 통해 CodeDeploy에서 서비스를 업데이트할 수 있습니다.	CodeDeploy 블루/그린 배포 유형을 사용하여 서비스를 배포합니다.	Amazon ECS CodeDeploy IAM 역할
Amazon ECS EventBridge 역할	이 역할을 통해 EventBridge에서 서비스를 업데이트할 수 있습니다.	EventBridge 규칙 및 대상을 사용하여 작업을 예약합니다.	Amazon ECS EventBridge IAM 역할

역할	정의	필요한 경우	추가 정보
Amazon ECS 인프라 역할	이 역할을 통해 Amazon ECS에서 클러스터의 인프라 리소스를 관리할 수 있습니다.	• Amazon EBS 볼륨을 Fargate 또는 EC2 시작 유형의 Amazon ECS 작업에 연결하려고 합니다. 인프라 역할을 통해 Amazon ECS에서 작업에 사용할 Amazon EBS 볼륨을 관리할 수 있습니다. • 전송 계층 보안(TLS)을 사용하여 Amazon ECS Service Connect 서비스 간 트래픽을 암호화하려고 합니다. • VPC Lattice 대상 그룹을 생성하려는 참입니다.	Amazon ECS 인프라 IAM 역할

Amazon ECS의 IAM 역할 모범 사례

Amazon ECS에 필요한 역할은 작업 정의 시작 유형 및 사용하는 기능에 따라 다릅니다. 역할을 공유하는 대신 테이블에 별도의 역할을 생성하는 것이 좋습니다.

역할	정의	필요한 경우	추가 정보
태스크 실행 역할	이 역할을 통해 Amazon ECS가 사용자를 대신해 다른 AWS 서비스를 사용할 수 있습니다.	작업은 AWS Fargate 또는 외부 인스턴스에 호스팅됩니다. 그리고, • Amazon ECR 프라이빗 리포지토리에	Amazon ECS 태스크 실행 IAM 역할

역할	정의	필요한 경우	추가 정보
		서 컨테이너 이미지를 가져옵니다. • 작업을 실행하는 계정과 다른 계정의 Amazon ECR 프라이빗 리포지토리에서 컨테이너 이미지를 가져옵니다. • `awslogs` 로그 드라이버를 사용해 CloudWatch Logs에 컨테이너 로그를 보냅니다. 작업은 AWS Fargate 또는 Amazon EC2 인스턴스에 호스팅됩니다. 그리고, • 프라이빗 레지스트리 인증을 사용합니다. • Runtime Monitoring을 사용합니다. • 작업 정의에서 Secrets Manager 비밀 또는 AWS Systems Manager Parameter Store 파라미터를 사용하여 민감한 데이터를 참조합니다.	

역할	정의	필요한 경우	추가 정보
태스크 역할	이 역할을 통해 컨테이너의 애플리케이션 코드가 다른 AWS 서비스를 사용할 수 있습니다.	애플리케이션은 Amazon S3와 같은 다른 AWS 서비스에 액세스합니다.	Amazon ECS 작업 IAM 역할
컨테이너 인스턴스 역할	이 역할을 통해 EC2 인스턴스 또는 외부 인스턴스를 클러스터에 등록할 수 있습니다.	작업은 Amazon EC2 인스턴스 또는 외부 인스턴스에 호스팅됩니다.	Amazon ECS 컨테이너 인스턴스 IAM 역할
Amazon ECS Anywhere 역할	이 역할을 통해 외부 인스턴스에서 AWS API에 액세스할 수 있습니다.	작업은 외부 인스턴스에 호스팅됩니다.	Amazon ECS Anywhere IAM 역할
Amazon ECS CodeDeploy 역할	이 역할을 통해 CodeDeploy에서 서비스를 업데이트할 수 있습니다.	CodeDeploy 블루/그린 배포 유형을 사용하여 서비스를 배포합니다.	Amazon ECS CodeDeploy IAM 역할
Amazon ECS EventBridge 역할	이 역할을 통해 EventBridge에서 서비스를 업데이트할 수 있습니다.	EventBridge 규칙 및 대상을 사용하여 작업을 예약합니다.	Amazon ECS EventBridge IAM 역할

역할	정의	필요한 경우	추가 정보
Amazon ECS 인프라 역할	이 역할을 통해 Amazon ECS에서 클러스터의 인프라 리소스를 관리할 수 있습니다.	• Amazon EBS 볼륨을 Fargate 또는 EC2 시작 유형의 Amazon ECS 작업에 연결하려고 합니다. 인프라 역할을 통해 Amazon ECS에서 작업에 사용할 Amazon EBS 볼륨을 관리할 수 있습니다. • 전송 계층 보안(TLS)을 사용하여 Amazon ECS Service Connect 서비스 간 트래픽을 암호화하려고 합니다. • VPC Lattice 대상 그룹을 생성하려는 참입니다.	Amazon ECS 인프라 IAM 역할

태스크 역할

태스크 역할을 할당하는 것을 권장합니다. 작업의 역할과 작업이 실행되는 Amazon EC2 인스턴스의 역할은 구분할 수 있습니다. 각 작업에 역할을 할당하면 최소 권한 액세스 원칙에 부합하며, 작업과 리소스를 보다 정밀하게 제어할 수 있습니다.

작업 정의에 작업 역할을 추가하면 Amazon ECS 컨테이너 에이전트는 작업에 대한 고유한 보안 인증 ID(예: `12345678-90ab-cdef-1234-567890abcdef`)를 사용하여 토큰을 자동으로 생성합니다. 그러면 이 토큰과 역할 보안 인증이 에이전트의 내부 캐시에 추가됩니다. 에이전트는 컨테이너의 환경 변수 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`를 보안 인증 ID의 URI(예: `/v2/credentials/12345678-90ab-cdef-1234-567890abcdef`)로 채웁니다.

Amazon ECS 컨테이너 에이전트의 IP 주소에 환경 변수를 추가하고 결과 문자열에서 `curl` 명령을 실행하여 컨테이너 내에서 임시 역할 보안 인증을 수동으로 검색할 수 있습니다.

```
curl 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
```

예상 출력은 다음과 같습니다.

```
{
 "RoleArn": "arn:aws:iam::123456789012:role/SSMTaskRole-SSMFargateTaskIAMRole-
DASWWSF2WGD6",
 "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
 "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
 "Token": "IQoJb3JpZ2luX2VjEEM/Example==",
 "Expiration": "2021-01-16T00:51:53Z"
}
```

최신 버전의 AWS SDK는 AWS API 호출 시 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` 환경 변수에서 이러한 보안 인증을 자동으로 가져옵니다. 자격 증명을 갱신하는 방법에 대한 자세한 내용은 RePost의 AWS 자격 증명 갱신을 참조하십시오.

출력에는 애플리케이션이 AWS 리소스에 액세스하는 데 사용하는 비밀 액세스 키 ID와 보안 암호 키로 구성된 액세스 키 쌍이 포함됩니다. AWS에서 보안 인증이 유효한지 확인하는 데 사용하는 토큰도 포함됩니다. 기본적으로 작업 역할을 사용하여 작업에 할당된 보안 인증은 6시간 동안 유효합니다. 그런 다음 Amazon ECS 컨테이너 에이전트에 의해 자동으로 교체됩니다.

태스크 실행 역할

작업 실행 역할은 Amazon ECS 컨테이너 에이전트에 사용자를 대신하여 특정 AWS API 호출을 수행할 권한을 부여하는 데 사용됩니다. 예를 들어, AWS Fargate를 사용하는 경우 Fargate에는 Amazon ECR에서 이미지를 풀하고 CloudWatch Logs에 로그를 기록하도록 허용하는 IAM 역할이 필요합니다. 작업에서 이미지 풀 보안 암호와 같이 AWS Secrets Manager에 저장된 보안 암호를 참조하는 경우에도 IAM 역할이 필요합니다.

> **Note**
>
> 인증된 사용자로 이미지를 풀하는 경우 Docker Hub의 풀 속도 제한 변경으로 인한 영향을 덜 받을 수 있습니다. 자세한 내용은 Private registry authentication for container instances를 참조하세요.
>
> Amazon ECR 및 Amazon ECR Public을 사용하면 Docker에서 적용한 제한을 피할 수 있습니다. Amazon ECR에서 이미지를 풀하면 네트워크 풀 시간을 단축하는 데에도 도움이 되고 트래픽이 VPC를 나갈 때 데이터 전송 변경 사항이 줄어듭니다.

> **Important**
>
> Fargate를 사용할 때는 `repositoryCredentials`를 사용하여 프라이빗 이미지 레지스트리에 인증해야 합니다. Fargate에 호스팅된 작업의 경우 Amazon ECS 컨테이너 에이전트 환경 변수 `ECS_ENGINE_AUTH_TYPE` 또는 `ECS_ENGINE_AUTH_DATA`를 설정하거나 `ecs.config` 파일을 수정할 수 없습니다. 자세한 내용은 Private registry authentication for tasks를 참조하세요.

컨테이너 인스턴스 역할

`AmazonEC2ContainerServiceforEC2Role` 관리형 IAM 정책에 다음 권한이 포함되어야 합니다. 최소 권한 부여에 대한 표준 보안 권고 사항에 따라 `AmazonEC2ContainerServiceforEC2Role` 관리형 정책을 가이드로 사용할 수 있습니다. 사용 사례에 대해 관리형 정책에서 부여한 권한이 필요하지 않은 경우 사용자 지정 정책을 생성하고 필요한 권한만 추가합니다.

- `ec2:DescribeTags` – (선택 사항) 위탁자가 Amazon EC2 인스턴스와 연결된 태그를 설명할 수 있습니다. 이 권한은 Amazon ECS 컨테이너 에이전트가 리소스 태그 전파를 지원하는 데 사용합니다. 자세한 내용은 리소스 태그 지정 방법 섹션을 참조하세요.
- `ecs:CreateCluster` – (선택 사항) 보안 주체가 Amazon ECS 클러스터를 생성할 수 있습니다. 이 권한은 `default` 클러스터가 이미 존재하지 않는 경우 이를 생성하기 위해 Amazon ECS 컨테이너 에이전트에서 사용합니다.
- `ecs:DeregisterContainerInstance` – (선택 사항) 보안 주체가 클러스터에서 Amazon ECS 컨테이너 인스턴스의 등록을 취소할 수 있습니다. Amazon ECS 컨테이너 에이전트는 이 API 작업을 직접 호출하지 않지만 이전 버전과의 호환성을 보장하기 위해 이 권한을 유지합니다.
- `ecs:DiscoverPollEndpoint` – (필수) 이 작업은 Amazon ECS 컨테이너 에이전트가 업데이트를 폴링하는 데 사용하는 엔드포인트를 반환합니다.
- `ecs:Poll` – (필수) Amazon ECS 컨테이너 에이전트가 Amazon ECS 컨트롤 플레인과 통신하여 태스크 상태 변경 사항을 보고할 수 있습니다.
- `ecs:RegisterContainerInstance` – (필수) 보안 주체가 클러스터에 컨테이너 인스턴스를 등록할 수 있습니다. 이 권한은 Amazon EC2 인스턴스를 클러스터에 등록하고 리소스 태그 전파를 지원하기 위해 Amazon ECS 컨테이너 에이전트에서 사용합니다.
- `ecs:StartTelemetrySession` – (선택 사항) Amazon ECS 컨테이너 에이전트가 Amazon ECS 컨트롤 플레인과 통신하여 각 컨테이너 및 태스크에 대한 상태 정보 및 지표를 보고할 수 있습니다.

이 권한이 필수는 아니지만 컨테이너 인스턴스 지표를 통해 규모 조정 작업을 시작하고 상태 확인 명령과 관련된 보고서를 수신할 수 있도록 하려면 이 권한을 추가하는 것이 좋습니다.

- `ecs:TagResource` - (선택 사항) Amazon ECS 컨테이너 에이전트가 클러스터 생성 시 클러스터에 태그를 지정하고, 컨테이너 인스턴스가 클러스터에 등록될 때 인스턴스에 태그를 지정할 수 있습니다.
- `ecs:UpdateContainerInstancesState` – 보안 주체가 Amazon ECS 컨테이너 인스턴스의 상태를 수정할 수 있습니다. 이 권한은 Amazon ECS 컨테이너 에이전트가 스팟 인스턴스 드레이닝에 사용합니다.
- `ecs:Submit*` – (필수) 여기에는 `SubmitAttachmentStateChanges`, `SubmitContainerStateChange` 및 `SubmitTaskStateChange` API 작업이 포함됩니다. 각 리소스의 상태 변경 사항을 Amazon ECS 제어 영역에 보고하기 위해 Amazon ECS 컨테이너 에이전트에서 사용합니다. `SubmitContainerStateChange` 권한은 Amazon ECS 컨테이너 에이전트에서 더 이상 사용하지 않지만 이전 버전과의 호환성을 보장하기 위해 유지됩니다.
- `ecr:GetAuthorizationToken` – (선택 사항) 보안 주체가 권한 부여 토큰을 검색할 수 있습니다. 권한 부여 토큰은 IAM 인증 자격 증명을 나타내며 IAM 보안 주체가 액세스할 수 있는 모든 Amazon ECR 레지스트리에 액세스하는 데 사용할 수 있습니다. 수신된 권한 부여 토큰은 12시간 동안 유효합니다.
- `ecr:BatchCheckLayerAvailability` – (선택 사항) 컨테이너 이미지가 Amazon ECR 프라이빗 리포지토리에 푸시될 때 이미지가 이미 푸시되었는지 확인하기 위해 각 이미지 계층을 검사합니다. 이미지가 푸시되었다면 이미지 계층을 건너뜁니다.
- `ecr:GetDownloadUrlForLayer` – (선택 사항) Amazon ECR 프라이빗 리포지토리에서 컨테이너 이미지를 가져올 때 아직 캐시되지 않은 각 이미지 레이어에 대해 이 API를 한 번 호출합니다.
- `ecr:BatchGetImage` – (선택 사항) Amazon ECR 프라이빗 리포지토리에서 컨테이너 이미지를 가져올 때 이미지 매니페스트를 검색하기 위해 이 API를 한 번 호출합니다.
- `logs:CreateLogStream` – (선택 사항) 보안 주체가 지정된 로그 그룹에 대한 CloudWatch Logs 로그 스트림을 생성할 수 있습니다.
- `logs:PutLogEvents` – (선택 사항) 보안 주체가 로그 이벤트의 배치를 지정된 로그 스트림에 업로드할 수 있습니다.

서비스 연결 역할

Amazon ECS에 대한 서비스 연결 역할을 사용하면 사용자를 대신하여 다른 서비스 API를 호출할 수 있는 권한을 Amazon ECS 서비스에 부여할 수 있습니다. Amazon ECS에는 네트워크 인터페이스를 생성 및 삭제하고, 대상을 대상 그룹에 등록 및 등록 취소할 수 있는 권한이 필요합니다. 또한 조정 정책

을 생성 및 삭제하는 데 필요한 권한도 필요합니다. 이러한 권한은 서비스 연결 역할을 통해 부여합니다. 이 역할은 서비스를 처음 사용할 때 자동으로 생성됩니다.

> **Note**
>
> 서비스 연결 역할을 실수로 삭제한 경우 다시 생성할 수 있습니다. 자세한 내용은 Create the service-linked role을 참조하세요.

역할 권장 사항

작업 IAM 역할 및 정책을 설정할 때 다음을 수행하는 것이 좋습니다.

Amazon EC2 메타데이터에 대한 액세스 차단

Amazon EC2 인스턴스에서 작업을 실행할 때 Amazon EC2 메타데이터에 대한 액세스를 차단하여 컨테이너가 해당 인스턴스에 할당된 역할을 상속하지 못하도록 하는 것이 좋습니다. 애플리케이션에서 AWS API 작업을 호출해야 하는 경우 대신 작업에 대한 IAM 역할을 사용하세요.

브리지 모드로 실행 중인 작업이 Amazon EC2 메타데이터에 액세스하지 못하도록 하려면 다음 명령을 실행하거나 인스턴스의 사용자 데이터를 업데이트하세요. 인스턴스의 사용자 데이터 업데이트에 대한 자세한 지침은 이 AWS 지원 문서를 참조하세요. 작업 정의 브리지 모드에 대한 자세한 내용은 task definition network mode를 참조하세요.

```
sudo yum install -y iptables-services; sudo iptables --insert FORWARD 1 --in-interface
 docker+ --destination 169.254.169.254/32 --jump DROP
```

이 변경 사항이 재부팅 후에도 유지되도록 하려면 Amazon Machine Image(AMI)에만 적용되는 다음 명령을 실행하세요.

- Amazon Linux 2

```
sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now
 iptables
```

- Amazon Linux

```
sudo service iptables save
```

awsvpc 네트워크 모드를 사용하는 작업의 경우 /etc/ecs/ecs.config 파일에서 환경 변수 ECS_AWSVPC_BLOCK_IMDS를 true로 설정하세요.

host 네트워크 내에서 실행 중인 컨테이너가 Amazon EC2 메타데이터에 액세스하지 못하도록 하려면 ecs-agent config 파일에서 ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST 변수를 false로 설정해야 합니다.

awsvpc 네트워크 모드 사용

네트워크 awsvpc 네트워크 모드를 사용하면 다른 작업 간이나 Amazon VPC 내에서 실행되는 다른 서비스와 다른 작업 간의 트래픽 흐름을 제한할 수 있습니다. 이 경우 보안 계층이 또 하나 추가됩니다. awsvpc 네트워크 모드에서는 Amazon EC2에서 실행되는 작업에 대한 작업 수준 네트워크 격리를 제공합니다. 이는 AWS Fargate에서 기본 모드이며, 작업에 보안 그룹을 할당하는 데 사용할 수 있는 유일한 네트워크 모드입니다.

마지막으로 액세스한 정보를 사용하여 역할 세분화

한 번도 사용하지 않았거나 한동안 사용하지 않은 작업은 제거하는 것이 좋습니다. 이렇게 하면 원치 않는 액세스를 방지할 수 있습니다. 이렇게 하려면 IAM에서 제공되는 마지막으로 액세스한 정보를 검토한 다음에 사용한 적이 없거나 최근에 사용하지 않은 작업을 제거합니다. 다음 단계에 따라 이 작업을 수행할 수 있습니다.

다음 명령을 실행하여 참조된 정책에 대한 마지막 액세스 정보를 보여주는 보고서를 생성합니다.

```
aws iam generate-service-last-accessed-details --arn arn:aws:iam::123456789012:policy/
ExamplePolicy1
```

출력에 있는 JobId를 사용하여 다음 명령을 실행합니다. 이를 수행한 후 보고서의 결과를 볼 수 있습니다.

```
aws iam get-service-last-accessed-details --job-id 98a765b4-3cde-2101-2345-example678f9
```

자세한 내용은 마지막으로 액세스한 정보를 사용하여 AWS에서의 권한 세분화를 참조하세요.

AWS CloudTrail을 모니터링하여 의심스러운 활동 식별

AWS CloudTrail을 모니터링하여 의심스러운 활동을 찾아낼 수 있습니다. 대부분의 AWS API 호출은 AWS CloudTrail에 이벤트로 기록됩니다. 이러한 호출을 AWS CloudTrail Insights에서 분석하며 write API 호출과 관련된 의심스러운 동작이 있으면 알림을 받게 됩니다. 호출 볼륨의 급증도 여기에

포함될 수 있습니다. 이러한 알림에는 비정상적인 활동이 발생한 시간, API에 기여한 상위 자격 증명 ARN과 같은 정보가 포함됩니다.

이벤트의 `userIdentity` 속성을 보면 AWS CloudTrail에서 IAM 역할의 작업으로 수행된 작업을 식별할 수 있습니다. 다음 예제에서 `arn`에는 수임된 역할의 이름 `s3-write-go-bucket-role`과 작업 이름 `7e9894e088ad416eb5cab92afExample` 순으로 포함됩니다.

```
"userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROA36C6WWEJ2YEXAMPLE:7e9894e088ad416eb5cab92afExample",
    "arn": "arn:aws:sts::123456789012:assumed-role/s3-write-go-bucket-
role/7e9894e088ad416eb5cab92afExample",
    ...
}
```

> **Note**
>
> 역할을 수임하는 작업이 Amazon EC2 컨테이너 인스턴스에서 실행되는 경우 Amazon ECS 컨테이너 에이전트는 `/var/log/ecs/audit.log.YYYY-MM-DD-HH` 형식의 주소에 있는 에이전트의 감사 로그에 요청을 기록합니다. 자세한 내용은 Task IAM Roles Log 및 Logging Insights Events for Trails를 참조하세요.

Amazon ECS 태스크 실행 IAM 역할

태스크 실행 IAM 역할은 Amazon ECS 컨테이너 에이전트 및 Fargate 에이전트에 사용자를 대신하여 AWS API 호출을 수행할 권한을 부여합니다. 태스크의 요구 사항에 따라 태스크 실행 IAM 역할이 필요합니다. 계정과 연결된 다른 용도 및 서비스에 사용할 여러 태스크 실행 역할이 있을 수 있습니다.

> **Note**
>
> 이러한 권한은 Amazon ECS가 역할의 임시 자격 증명을 주기적으로 전송하여 인스턴스에서 실행 중인 에이전트에서 사용할 수 있지만 태스크의 컨테이너에서 직접 액세스할 수 없습니다. 컨테이너 내에서 애플리케이션을 실행하는 데 필요한 IAM 권한은 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.

태스크 실행 IAM 역할의 일반 사용 사례는 다음과 같습니다.

- 작업은 AWS Fargate 또는 외부 인스턴스에 호스팅됩니다. 그리고,

  - Amazon ECR 프라이빗 리포지토리에서 컨테이너 이미지를 가져옵니다.
  - 작업을 실행하는 계정과 다른 계정의 Amazon ECR 프라이빗 리포지토리에서 컨테이너 이미지를 가져옵니다.
  - `awslogs` 로그 드라이버를 사용해 CloudWatch Logs에 컨테이너 로그를 보냅니다. 자세한 내용은 Amazon ECS 로그를 CloudWatch로 전송 섹션을 참조하세요.
- 작업은 AWS Fargate 또는 Amazon EC2 인스턴스에 호스팅됩니다. 그리고,
  - 프라이빗 레지스트리 인증을 사용합니다. 자세한 내용은 프라이빗 레지스트리 인증 권한 섹션을 참조하세요.
  - Runtime Monitoring을 사용합니다.
  - 작업 정의에서 Secrets Manager 비밀 또는 AWS Systems Manager Parameter Store 파라미터를 사용하여 민감한 데이터를 참조합니다. 자세한 내용은 Secrets Manager 또는 Systems Manager 권한 섹션을 참조하세요.

> **Note**
>
> 태스크 실행 역할은 Amazon ECS 컨테이너 에이전트 버전 1.16.0 이상에서 지원됩니다.

Amazon ECS는 위에서 설명한 일반 사용 사례에 필요한 권한을 포함하는 `AmazonECSTaskExecutionRolePolicy`라는 관리형 정책을 제공합니다. 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonECSTaskExecutionRolePolicy를 참조하세요. 특수 사용 사례에서는 인라인 정책을 작업 실행 역할에 추가해야 할 수도 있습니다.

Amazon ECS 콘솔에서 작업 실행 역할을 생성합니다. 향후 기능 및 개선 사항이 도입될 때 Amazon ECS가 이에 대한 권한을 추가할 수 있도록 하려면 수동으로 작업에 대한 관리형 IAM 정책을 연결할 수 있습니다. IAM 콘솔 검색을 사용하여 `ecsTaskExecutionRole`을 검색하고 계정에 이미 작업 실행 역할이 있는지 확인할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 콘솔 검색을 참조하세요.

인증된 사용자로 이미지를 가져오는 경우 Docker Hub 사용량과 제한 변경으로 인한 영향을 덜 받을 수 있습니다. 자세한 내용은 Private registry authentication for container instances를 참조하세요.

Amazon ECR 및 Amazon ECR Public을 사용하면 Docker에서 적용한 제한을 피할 수 있습니다. Amazon ECR에서 이미지를 풀하면 네트워크 풀 시간을 단축하는 데에도 도움이 되고 트래픽이 VPC를 나갈 때 데이터 전송 변경 사항이 줄어듭니다.

Fargate를 사용할 때는 `repositoryCredentials`를 사용하여 프라이빗 이미지 레지스트리에 인증해야 합니다. Fargate에 호스팅된 작업의 경우 Amazon ECS 컨테이너 에이전트 환경 변수

ECS_ENGINE_AUTH_TYPE 또는 ECS_ENGINE_AUTH_DATA를 설정하거나 ecs.config 파일을 수정할 수 없습니다. 자세한 내용은 Private registry authentication for tasks를 참조하세요.

작업 실행 역할 생성

계정에 아직 작업 실행 역할이 없는 경우 다음 단계를 사용하여 역할을 생성합니다.

AWS Management Console

Elastic Container Service에 대한 서비스 역할을 생성하는 방법(IAM 콘솔)

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할을 선택하고 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티 유형에 AWS 서비스를 선택합니다.
4. 서비스 또는 사용 사례의 경우 Elastic Container Service를 선택하고 Elastic Container Service 작업 사용 사례를 선택합니다.
5. 다음을 선택합니다.
6. 권한 추가 섹션에서 AmazonECSTaskExecutionRolePolicy를 검색하고 정책을 선택합니다.
7. 다음을 선택합니다.
8. 역할 이름에 ecsTaskExecutionRole을 입력합니다.
9. 역할을 검토한 다음 역할 생성을 선택합니다.

AWS CLI

모든 ### ##을 사용자 정보로 바꿉니다.

1. IAM 역할에 사용할 신뢰 정책이 포함된 ecs-tasks-trust-policy.json이라는 이름의 파일을 생성합니다. 파일에 다음을 포함해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
```

```
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. 이전 단계에서 만든 신뢰 정책을 사용하여 ecsTaskExecutionRole이라는 IAM 역할을 생성합니다.

```
aws iam create-role \
      --role-name ecsTaskExecutionRole \
      --assume-role-policy-document file://ecs-tasks-trust-policy.json
```

3. AWS 관리형 AmazonECSTaskExecutionRolePolicy 정책을 ecsTaskExecutionRole 역할에 연결합니다.

```
aws iam attach-role-policy \
      --role-name ecsTaskExecutionRole \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy
```

역할을 생성한 후 다음 기능에 대한 추가 권한을 역할에 추가합니다.

Feature	추가 권한
Secrets Manager 자격 증명을 사용하여 AWS 외부의 프라이빗 레지스트리(예: Docker Hub, Quay.io 또는 자체 프라이빗 레지스트리)에서 컨테이너 이미지 가져오기	프라이빗 레지스트리 인증 권한
Systems Manager 또는 Secrets Manager를 사용하여 민감한 데이터 전달	Secrets Manager 또는 Systems Manager 권한
Fargate 작업에서 인터페이스 엔드포인트를 통해 Amazon ECR 이미지 가져오기	인터페이스 엔드포인트 권한을 통해 Amazon ECR 이미지를 가져오는 Fargate 작업
Amazon S3 버킷에서 구성 파일 호스팅	Amazon S3 파일 스토리지 권한
Container Insights를 구성하여 Amazon ECS 수명 주기 이벤트 보기	Container Insights에서 Amazon ECS 수명 주기 이벤트를 활성화하기 위해 필요한 권한

Feature	추가 권한
Container Insights에서 Amazon ECS 수명 주기 이벤트 보기	Container Insights에서 Amazon ECS 수명 주기 이벤트를 확인하기 위해 필요한 권한

프라이빗 레지스트리 인증 권한

프라이빗 레지스트리 인증을 사용하면 Amazon ECS 태스크가 인증 자격 증명이 필요한 AWS 외부의 프라이빗 레지스트리(예: Docker Hub, Quay.io 또는 자체 프라이빗 레지스트리)에서 컨테이너 이미지를 가져올 수 있습니다. 이 기능은 Secrets Manager를 사용하여 레지스트리 자격 증명을 안전하게 저장한 다음, `repositoryCredentials` 파라미터를 사용하여 태스크 정의에서 참조합니다.

프라이빗 레지스트리 인증 구성에 대한 자세한 내용은 Amazon ECS에서 AWS 외의 컨테이너 이미지 사용을 참조하세요.

프라이빗 레지스트리 자격 증명이 포함된 암호에 액세스 권한을 부여하려면 다음 권한을 인라인 정책으로 태스크 실행 역할에 추가합니다. 자세한 정보는 IAM 정책 추가 및 제거 섹션을 참조하세요.

- `secretsmanager:GetSecretValue` - Secrets Manager에서 프라이빗 레지스트리 자격 증명을 검색하는 데 필요합니다.
- `kms:Decrypt` – 암호가 사용자 지정 KMS 키를 사용하고 기본 키를 사용하지 않는 경우에 필요합니다. 사용자 지정 키의 Amazon 리소스 이름(ARN)을 리소스로 추가해야 합니다.

다음 예제에서는 인라인 정책이 권한을 추가합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:secret_name",
                "arn:aws:kms:<region>:<aws_account_id>:key/key_id"
            ]
        }
```

```
    ]
}
```

Secrets Manager 또는 Systems Manager 권한

컨테이너 에이전트가 필요한 AWS Systems Manager 또는 Secrets Manager 리소스를 가져올 수 있는 권한입니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.

Secrets Manager 사용

사용자가 생성한 Secrets Manager 보안 암호에 대한 액세스 권한을 제공하려면 작업 실행 역할에 다음 권한을 수동으로 추가합니다. 권한 관리 방법에 대한 자세한 내용은 IAM 사용 설명서의 IAM 자격 증명 권한 추가 및 제거를 참조하세요.

- `secretsmanager:GetSecretValue` – Secrets Manager 보안 암호를 참조하는 경우에 필요합니다. Secrets Manager에서 암호를 검색할 수 있는 권한을 추가합니다.

다음 예제에서는 정책이 필수 권한을 추가합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
      ]
    }
  ]
}
```

Systems Manager 사용

> **Important**
>
> EC2 시작 유형을 사용하는 태스크의 경우, 이 기능을 사용하려면 ECS 에이전트 구성 변수인 `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true`를 사용해야 합니다. 컨테이너

인스턴스를 생성할 때 이를 `./etc/ecs/ecs.config` 파일에 추가하거나 기존 인스턴스에 추가한 다음 ECS 에이전트를 다시 시작할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 섹션을 참조하세요.

사용자가 생성한 Systems Manager Parameter Store 파라미터에 대한 액세스 권한을 제공하려면 작업 실행 역할에 다음 권한을 정책으로 직접 추가합니다. 권한 관리 방법에 대한 자세한 내용은 IAM 사용 설명서의 IAM 자격 증명 권한 추가 및 제거를 참조하세요.

- `ssm:GetParameters` - 작업 정의에서 Systems Manager Parameter Store 파라미터를 참조하는 경우에 필요합니다. Systems Manager 파라미터를 검색할 수 있는 권한을 추가합니다.
- `secretsmanager:GetSecretValue` - Secrets Manager 보안 암호를 직접 참조하는 경우 또는 Systems Manager Parameter Store 파라미터가 작업 정의에서 Secrets Manager 보안 암호를 참조하는 경우에 필요합니다. Secrets Manager에서 암호를 검색할 수 있는 권한을 추가합니다.
- `kms:Decrypt` - 보안 암호가 기본 키가 아닌 고객 관리 키를 사용하는 경우에만 필요합니다. 사용자 지정 키의 ARN을 리소스로 추가해야 합니다. 고객 관리형 키의 암호를 해독할 수 있는 권한을 추가합니다.

다음 예제 정책은 필요한 권한을 추가합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameters",
        "secretsmanager:GetSecretValue",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:ssm:region:aws_account_id:parameter/parameter_name",
        "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name",
        "arn:aws:kms:region:aws_account_id:key/key_id"
      ]
    }
  ]
}
```

인터페이스 엔드포인트 권한을 통해 Amazon ECR 이미지를 가져오는 Fargate 작업

인터페이스 VPC 엔드포인트를 사용하도록 Amazon ECR을 구성한 경우 Amazon ECR에서 이미지를 가져오는 Fargate 시작 유형을 사용하는 태스크를 시작할 때 특정 VPC 또는 VPC 엔드포인트에 대한 태스크 액세스를 제한할 수 있습니다. 이를 위해선 IAM 조건 키를 사용하는 태스크에 대한 태스크 실행 역할을 생성합니다.

다음 IAM 전역 조건 키를 사용하여 특정 VPC 또는 VPC 엔드포인트에 대한 액세스를 제한합니다. 자세한 정보는 AWS 전역 조건 컨텍스트 키를 참조하세요.

- `aws:SourceVpc`–특정 VPC에 대한 액세스를 제한합니다. VPC를 태스크와 엔드포인트를 호스팅하는 VPC로 제한할 수 있습니다.
- `aws:SourceVpce`–특정 VPC 엔드포인트에 대한 액세스를 제한합니다.

다음 태스크 실행 역할 정책은 조건 키 추가에 대한 예를 제공합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
```

```
                    "arn:aws:ec2:arn:aws::111122223333:vpc-
endpoint/vpce-0123456789abcdef0",
                    "arn:aws:ec2:arn:aws::111122223333:vpc/vpc-0123456789abcdef0"
                }
            }
        }
    ]
}
```

Amazon ECR 권한

Amazon ECR 프라이빗 리포지토리에서 컨테이너 이미지를 가져와야 하는 경우 다음 권한이 필요합니다. 태스크 실행 역할에 이러한 권한이 있어야 Amazon ECS 컨테이너와 Fargate 에이전트가 사용자를 대신하여 컨테이너 이미지를 가져올 수 있습니다. 기본 ECS 구현의 경우 태스크 IAM 역할 대신 태스크 실행 역할에 이러한 권한을 추가해야 합니다.

Amazon ECS 태스크 실행 역할 관리형 정책(`AmazonECSTaskExecutionRolePolicy`)에는 Amazon ECR에서 이미지를 가져오는 데 필요한 권한이 포함되어 있습니다. 관리형 정책을 사용하는 경우 이러한 권한을 별도로 추가할 필요가 없습니다.

사용자 지정 정책을 생성하는 경우 Amazon ECR에서 이미지를 가져올 수 있도록 다음 권한을 포함합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

이러한 권한은 애플리케이션 코드가 Amazon ECR API와 직접 상호 작용해야 하는 경우 태스크 IAM 역할에 필요할 수 있는 권한과는 다르다는 점에 유의하세요. Amazon ECR의 태스크 IAM 역할 권한에 대한 자세한 내용은 Amazon ECR 권한 섹션을 참조하세요.

Amazon S3 파일 스토리지 권한

Amazon S3에 호스팅된 구성 파일을 지정하는 경우 태스크 실행 역할은 구성 파일에 대한 `s3:GetObject` 권한과 파일이 있는 Amazon S3 버킷에 대한 `s3:GetBucketLocation` 권한을 포함해야 합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3의 정책 작업을 참조하세요.

다음 정책 예제에서는 Amazon S3에서 파일을 검색하는 데 필요한 권한을 추가합니다. Amazon S3 버킷의 이름과 구성 파일 이름을 지정합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/folder_name/config_file_name"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket"
      ]
    }
  ]
}
```

Amazon ECS 작업 IAM 역할

Amazon ECS 작업에는 IAM 역할이 연결될 수 있습니다. IAM 역할에서 부여된 권한은 태스크에서 실행 중인 컨테이너에 제공됩니다. 이 역할을 통해 컨테이너에서 실행되는 애플리케이션 코드가 다른 AWS 서비스를 사용할 수 있습니다. 애플리케이션이 Amazon S3와 같은 다른 AWS 서비스에 액세스할 때 작업 역할이 필요합니다.

> **Note**
>
> 이 권한은 Amazon ECS 컨테이너 및 Fargate 에이전트가 액세스하지 않습니다. Amazon ECS가 컨테이너 이미지를 가져오고 작업을 실행하는 데 필요한 IAM 권한에 대해서는 Amazon ECS 태스크 실행 IAM 역할을 참조하세요.

다음은 작업 역할 사용 시 이점입니다.

- 우려 사항 분리: EC2 시작 유형을 사용하는 경우 태스크 IAM 역할을 통해 EC2 인스턴스 프로파일을 사용하여 이러한 권한을 지정할 필요 없이 컨테이너에 대한 IAM 권한을 지정할 수 있습니다(자세한 내용은 AWS Identity and Access Management 사용 설명서의 인스턴스 프로파일 사용 참조). 따라서 EC2 인스턴스와 연결된 IAM 권한을 수정할 필요 없이 ECS 컨테이너 인스턴스에 애플리케이션을 독립적으로 균일하게 배포할 수 있습니다.
- 감사: CloudTrail을 통한 액세스 및 이벤트 로깅을 사용하여 소급적 감사를 보장합니다. 태스크 자격 증명에는 세션에 연결된 '`taskArn`'의 컨텍스트가 있으므로 CloudTrail 로그에 어느 태스크에 대해 역할 자격 증명이 제공되는지 표시됩니다.
- 균일한 자격 증명 제공: ECS는 컨테이너에 IAM 역할 자격 증명을 제공하고 태스크와 연결된 시작 유형에 관계없이 잘 정의된 인터페이스를 통해 액세스할 수 있도록 합니다. ECS Fargate에서는 태스크의 컨테이너에 EC2 인스턴스 프로파일을 사용할 수 없습니다. 태스크 IAM 역할을 사용하면 컨테이너에서 AWS SDK 또는 AWS CLI를 사용할 때 시작 유형에 관계없이 IAM 권한을 컨테이너에 연결할 수 있습니다. AWS SDK로 이러한 자격 증명에 액세스하는 방법에 대한 자세한 내용은 컨테이너 자격 증명 공급자를 참조하세요.

> **Important**
>
> 컨테이너는 보안 경계가 아니며 태스크 IAM 역할을 사용해도 변경되지 않습니다. Fargate에서 실행되는 각 태스크에는 자체 격리 경계가 있으며 기본 커널, CPU 리소스, 메모리 리소스 또는 탄력적 네트워크 인터페이스를 다른 태스크와 공유하지 않습니다. EC2 및 ECS의 외부 컨테이너 인스턴스의 경우 (Fargate와 달리)태스크 격리가 없으며 컨테이너는 동일한 컨테이

너 인스턴스의 다른 작업에 대한 자격 증명에 액세스할 수 있습니다. 또한 ECS 컨테이너 인스턴스 역할에 할당된 권한에 액세스할 수 있습니다. 역할 권장 사항의 권장 사항에 따라 컨테이너용 Amazon EC2 인스턴스 메타데이터 서비스에 대한 액세스를 차단합니다(자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 메타데이터 서비스를 사용하여 인스턴스 메타데이터에 액세스 참조).
태스크에 대한 IAM 역할을 지정할 때 해당 태스크에 대한 컨테이너의 AWS CLI 또는 다른 SDK는 태스크 역할에 의해 단독으로 제공된 AWS 자격 증명을 사용하며, Amazon EC2 또는 실행 중인 외부 인스턴스에서 IAM 권한을 상속하지 않습니다.

작업 IAM 역할 생성

작업에서 사용할 IAM 정책을 생성할 때 작업의 컨테이너에서 수임하려는 권한을 정책에 포함해야 합니다. 기존 AWS 관리형 정책을 사용하거나 특정 요구 사항에 맞는 사용자 지정 정책을 새로 생성할 수 있습니다. 자세한 내용은 IAM 사용자 설명서에서 IAM 정책 생성을 참조하세요.

Important

(모든 시작 유형에 대한) Amazon ECS 태스크의 경우 태스크에 대해 IAM 정책과 역할을 사용하는 것이 좋습니다. 이 자격 증명을 사용하면 태스크가 `sts:AssumeRole`을 호출하지 않고 AWS API 요청을 보내고 태스크와 이미 연결된 것과 동일한 역할을 수입할 수 있습니다. 태스크가 자체적으로 역할을 수임하는 경우 해당 역할이 자체적으로 수임하도록 명시적으로 허용하는 신뢰 정책을 생성해야 합니다. 자세한 내용은 IAM 사용 설명서의 역할 트러스트 정책 업데이트를 참조하세요.

IAM 정책이 생성되면 Amazon ECS 작업 정의에 참조하는 정책을 포함하는 IAM 역할을 생성할 수 있습니다. IAM 콘솔에서 Elastic Container Service Task 사용 사례를 사용해 역할을 생성할 수 있습니다. 그런 다음, 작업의 컨테이너에 필요한 권한을 부여하는 특정 IAM 정책을 역할과 연결할 수 있습니다. 아래 절차에 이렇게 하는 방법이 나와 있습니다.

IAM 권한이 필요한 태스크 정의 또는 서비스가 여러 개인 경우 각 태스크 정의 또는 서비스에 대해 태스크에 필요한 권한을 최소한으로 포함하는 역할을 생성하여 각 태스크에 제공하는 액세스를 최소화할 것을 고려해야 합니다.

리전의 서비스 엔드포인트에 대한 자세한 내용은 Amazon Web Services 일반 참조 참조 가이드의 Service endpoints를 참조하세요.

IAM 작업 역할에는 `ecs-tasks.amazonaws.com` 서비스를 지정하는 신뢰 정책이 있어야 합니다. `sts:AssumeRole` 권한을 사용하면 작업이 Amazon EC2 인스턴스에서 사용하는 것과 다른 IAM 역할을 맡을 수 있습니다. 이렇게 하면 작업이 Amazon EC2 인스턴스와 연결된 역할을 상속하지 않습니다. 다음은 신뢰 정책 예시입니다. 리전 식별자를 바꾸고 작업을 시작할 때 사용하는 AWS 계정 번호를 지정합니다.

> **Important**
>
> 태스크 IAM 역할을 생성할 때는 혼동된 대리인 보안 문제를 방지하기 위해 추가로 권한 범위를 지정하도록 역할에 연결된 신뢰 관계 정책에서 `aws:SourceAccount` 또는 `aws:SourceArn` 조건 키를 사용하는 것이 좋습니다. `aws:SourceArn` 조건 키를 사용하여 특정 클러스터를 지정하는 작업은 현재 지원되지 않으므로 와일드카드를 사용하여 모든 클러스터를 지정해야 합니다. 혼동되는 대리인 문제 및 AWS 계정 보호 방법에 대해 자세히 알아보려면, IAM 사용 설명서의 혼동되는 대리인 문제를 참조하세요.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":[
               "ecs-tasks.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "ArnLike":{
            "aws:SourceArn":"arn:aws:ecs:us-west-2:111122223333:*"
            },
            "StringEquals":{
               "aws:SourceAccount":"111122223333"
            }
         }
      }
   ]
}
```

예제 정책을 사용하여 Amazon S3에서 객체를 검색하는 정책을 생성하려면 다음 절차를 사용합니다. 모든 *### ##*을 고유한 값으로 바꿉니다.

AWS Management Console

JSON 정책 편집기를 사용하여 정책을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 창에서 정책을 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작을 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. 정책 편집기 섹션에서 JSON 옵션을 선택합니다.
5. 다음 JSON 정책 문서를 입력합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
         ],
         "Resource":[
            "arn:aws:s3:::my-task-secrets-bucket/*"
         ]
      }
   ]
}
```

6. 다음을 선택합니다.

> **Note**
>
> 언제든지 시각적 편집기 옵션과 JSON 편집기 옵션 간에 전환할 수 있습니다. 그러나 변경을 적용하거나 시각적 편집기에서 다음을 선택한 경우 IAM은 시각적 편집기에 최적화되도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 정책 재구성을 참조하세요.

7. 검토 및 생성 페이지에서 생성하는 정책에 대한 정책 이름과 설명(선택 사항)을 입력합니다. 이 정책에 정의된 권한을 검토하여 정책이 부여한 권한을 확인합니다.
8. 정책 생성을 선택하고 새로운 정책을 저장합니다.

AWS CLI

모든 *### ##*을 고유한 값으로 바꿉니다.

1. 다음 콘텐츠를 통해 `s3-policy.json`이라는 파일을 생성합니다.

   JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
         ],
         "Resource":[
            "arn:aws:s3:::my-task-secrets-bucket/*"
         ]
      }
   ]
}
```

2. 다음 명령을 사용하여 JSON 정책 문서 파일을 사용해 IAM 정책을 생성합니다. 모든 *### ##*을 고유한 값으로 바꿉니다.

```
aws iam create-policy \
      --policy-name taskRolePolicy \
      --policy-document file://s3-policy.json
```

다음 절차에 따라 서비스 역할을 생성합니다.

AWS Management Console

Elastic Container Service에 대한 서비스 역할을 생성하는 방법(IAM 콘솔)

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할을 선택하고 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티 유형에 AWS 서비스를 선택합니다.
4. 서비스 또는 사용 사례의 경우 Elastic Container Service를 선택하고 Elastic Container Service 작업 사용 사례를 선택합니다.
5. 다음을 선택합니다.
6. 권한 추가에서 사용자가 생성한 정책을 검색하고 선택합니다.
7. 다음을 선택합니다.
8. 역할 이름에 역할의 이름을 입력합니다. 이 예에서는 역할 이름에 `AmazonECSTaskS3BucketRole`을(를) 입력합니다.
9. 역할을 검토한 다음 역할 생성을 선택합니다.

AWS CLI

1. 작업 IAM 역할에 사용할 신뢰 정책을 포함하는 `ecs-tasks-trust-policy.json` 이름의 파일을 생성합니다. 파일에 다음을 포함해야 합니다. 리전 식별자를 바꾸고 작업을 시작할 때 사용하는 AWS 계정 번호를 지정합니다.

   JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":[
               "ecs-tasks.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole",
         "Condition":{
```

```
            "ArnLike":{
            "aws:SourceArn":"arn:aws:ecs:us-west-2:111122223333:*"
            },
            "StringEquals":{
               "aws:SourceAccount":"111122223333"
            }
         }
      }
   ]
}
```

2. 이전 단계에서 만든 신뢰 정책을 사용하여 ecsTaskRole이라는 IAM 역할을 생성합니다.

```
aws iam create-role \
      --role-name ecsTaskRole \
      --assume-role-policy-document file://ecs-tasks-trust-policy.json
```

3. 다음 명령을 사용하여 생성한 IAM 정책의 ARN을 검색합니다. *taskRolePolicy*를 사용자가 생성한 정책 이름으로 바꿉니다.

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`taskRolePolicy`].Arn'
```

4. 생성한 IAM 정책을 ecsTaskRole 역할에 연결합니다. policy-arn을 사용자가 생성한 정책의 ARN으로 바꿉니다.

```
aws iam attach-role-policy \
      --role-name ecsTaskRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/taskRolePolicy
```

역할을 생성한 후 다음 기능에 대한 추가 권한을 역할에 추가합니다.

Feature	추가 권한
ECS Exec 사용	ECS Exec 권한
프라이빗 Amazon ECR 리포지토리의 이미지 사용	Amazon ECR 권한
EC2 인스턴스 사용(Windows 및 Linux)	Amazon EC2 인스턴스 추가 구성

Feature	추가 권한
외부 인스턴스 사용	외부 인스턴스 추가 구성
Windows EC2 인스턴스 사용	Amazon EC2 Windows 인스턴스 추가 구성

Amazon ECR 권한

애플리케이션 코드가 Amazon ECR 리포지토리와 직접 상호 작용해야 하는 경우 다음 권한이 필요합니다. Amazon ECR에서 이미지만 가져오면 되는 기본 구현의 경우 태스크 IAM 역할 수준에서는 이러한 권한이 필요하지 않습니다. 대신 Amazon ECS 태스크 실행 역할에는 이러한 권한이 있어야 합니다. 태스크 실행 역할에 대한 자세한 정보는 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

컨테이너에서 실행되는 애플리케이션 코드가 Amazon ECR API와 직접 상호 작용해야 하는 경우 태스크 IAM 역할에 다음 권한을 추가하고 태스크 정의에 태스크 IAM 역할을 포함해야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

태스크 IAM 역할에 다음 정책을 사용하여 Amazon ECR과 직접 상호 작용해야 하는 컨테이너 애플리케이션에 필요한 Amazon ECR 권한을 추가합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

ECS Exec 권한

ECS Exec 기능을 사용하려면 관리형 SSM 에이전트(`execute-command` 에이전트)와 SSM 서비스 간의 통신에 필요한 권한을 컨테이너에 부여하기 위한 작업 IAM 역할이 필요합니다. 작업 IAM 역할에 다음 권한을 추가하고 태스크 정의에 작업 IAM 역할을 포함해야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

작업 IAM 역할에 대해 다음 정책을 사용하여 필수 SSM 권한을 추가합니다.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
       "Effect": "Allow",
       "Action": [
            "ssmmessages:CreateControlChannel",
            "ssmmessages:CreateDataChannel",
            "ssmmessages:OpenControlChannel",
            "ssmmessages:OpenDataChannel"
       ],
      "Resource": "*"
      }
   ]
}
```

Amazon EC2 인스턴스 추가 구성

컨테이너 인스턴스 역할의 권한을 `AmazonEC2ContainerServiceforEC2Role` 관리형 IAM 정책에서 사용되는 최소 권한 목록으로 제한하는 것이 좋습니다.

작업 역할을 사용하려면 Amazon EC2 인스턴스에 버전 `1.11.0` 이상의 컨테이너 에이전트가 필요합니다. 하지만 최신 컨테이너 에이전트 버전을 사용할 것을 권장합니다. 에이전트 버전을 확인하고 최신 버전으로 업데이트하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요. Amazon ECS 최적화 AMI를 사용하는 경우 인스턴스에는 `1.11.0-1` 이상의 `ecs-init` 패키지가 필요합니다. 인스턴스가 Amazon ECS 최적화 AMI를 사용하는 경우 필요한 버전의 컨테이너 에이전트 및 `ecs-init`이(가) 포함되어 있습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.

컨테이너 인스턴스에 Amazon ECS 최적화 AMI를 사용하지 않는 경우 에이전트를 시작하는 `--net=host` 명령에 docker run 옵션과 원하는 구성에 대한 다음 에이전트 변수를 추가합니다(자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 참조).

`ECS_ENABLE_TASK_IAM_ROLE=true`

`bridge` 및 `default` 네트워크 모드의 컨테이너 작업에 대한 IAM 역할을 활성화합니다.

`ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true`

`host` 네트워크 모드의 컨테이너 작업에 대한 IAM 역할을 사용합니다. 이 변수는 에이전트 버전 1.12.0 이상에서만 지원됩니다.

예제 실행 명령은 Amazon ECS 컨테이너 에이전트를 수동으로 업데이트하는 방법(비 Amazon ECS 최적화 AMI) 섹션을 참조하세요. 또한 태스크의 컨테이너가 AWS 자격 증명을 검색할 수 있도록 컨테이너 인스턴스에서 다음 네트워킹 명령을 설정해야 합니다.

```
sudo sysctl -w net.ipv4.conf.all.route_localnet=1
sudo iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-
destination 127.0.0.1:51679
sudo iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --
to-ports 51679
```

재부팅 시 이러한 iptables 규칙이 유지되게 하려면 규칙을 컨테이너 인스턴스에 저장해야 합니다. iptables-save 명령과 iptables-restore 명령을 사용하여 iptables 규칙을 저장하고 부팅 시 복원할 수 있습니다. 자세한 정보는 해당 운영 체제 설명서를 참조하세요.

`awsvpc` 네트워크 모드를 사용하는 작업이 실행하는 컨테이너가 Amazon EC2 인스턴스 프로파일에 제공된 자격 증명 정보에 액세스하지 못하게 방지하려면(단, 작업 역할에 제공된 권한은 허용) 에이전트 구성 파일의 `ECS_AWSVPC_BLOCK_IMDS` 에이전트 구성 변수를 `true`(으)로 설정하고 에이전트를 다시 시작합니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성을 참조하세요.

`bridge` 네트워크 모드를 사용하는 작업이 실행하는 컨테이너가 Amazon EC2 인스턴스 프로파일에 제공된 자격 증명에 액세스하지 못하게 방지하려면(단, 작업 역할에 제공된 권한은 허용) Amazon EC2 인스턴스에서 다음 iptables 명령을 실행합니다. 이 명령은 `host` 또는 `awsvpc` 네트워크 모드를 사용하는 작업 내 컨테이너에는 영향을 미치지 않습니다. 자세한 정보는 네트워크 모드을 참조하세요.

- ```
  sudo yum install -y iptables-services; sudo iptables --insert DOCKER-USER 1 --in-
  interface docker+ --destination 169.254.169.254/32 --jump DROP
  ```
  ```

재부팅 시 이 iptables 규칙이 유지되게 하려면 규칙을 Amazon EC2 인스턴스에 저장해야 합니다. Amazon ECS 최적화 AMI를 사용하는 경우 다음 명령을 사용할 수 있습니다. 다른 운영 체제의 경우 해당 운영 체제 설명서를 참조하세요.

```
sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now
 iptables
```

외부 인스턴스 추가 구성

작업 IAM 역할을 사용하려면 외부 인스턴스에 버전 `1.11.0` 이상의 컨테이너 에이전트가 필요합니다. 하지만 최신 컨테이너 에이전트 버전을 사용할 것을 권장합니다. 에이전트 버전을 확인하고 최신 버전으로 업데이트하는 방법에 대한 자세한 정보는 Amazon ECS 컨테이너 에이전트 업데이트 섹션을 참조하세요. Amazon ECS 최적화 AMI를 사용하는 경우 해당 인스턴스에는 `1.11.0-1` 이상의 `ecs-init` 패키지가 필요합니다. 인스턴스가 Amazon ECS 최적화 AMI를 사용하는 경우 필요한 버전의 컨테이너 에이전트 및 `ecs-init`이(가) 포함되어 있습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.

컨테이너 인스턴스에 Amazon ECS 최적화 AMI를 사용하지 않는 경우 에이전트를 시작하는 `--net=host` 명령에 docker run 옵션과 원하는 구성에 대한 다음 에이전트 변수를 추가합니다(자세한 내용은 Amazon ECS 컨테이너 에이전트 구성 참조).

`ECS_ENABLE_TASK_IAM_ROLE=true`

`bridge` 및 `default` 네트워크 모드의 컨테이너 작업에 대한 IAM 역할을 활성화합니다.

`ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true`

`host` 네트워크 모드의 컨테이너 작업에 대한 IAM 역할을 사용합니다. 이 변수는 에이전트 버전 1.12.0 이상에서만 지원됩니다.

예제 실행 명령은 Amazon ECS 컨테이너 에이전트를 수동으로 업데이트하는 방법(비 Amazon ECS 최적화 AMI) 섹션을 참조하세요. 또한 태스크의 컨테이너가 AWS 자격 증명을 검색할 수 있도록 컨테이너 인스턴스에서 다음 네트워킹 명령을 설정해야 합니다.

```
sudo sysctl -w net.ipv4.conf.all.route_localnet=1
sudo iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-
destination 127.0.0.1:51679
```

```
sudo iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --
to-ports 51679
```

재부팅 시 이러한 iptables 규칙이 유지되게 하려면 규칙을 컨테이너 인스턴스에 저장해야 합니다. iptables-save 명령과 iptables-restore 명령을 사용하여 iptables 규칙을 저장하고 부팅 시 복원할 수 있습니다. 자세한 정보는 해당 운영 체제 설명서를 참조하세요.

Amazon EC2 Windows 인스턴스 추가 구성

> **Important**
> 이는 작업 역할을 사용하는 EC2의 Windows 컨테이너에만 적용됩니다.

Windows 기능이 포함된 작업 역할을 사용하려면 EC2에서 추가 구성이 필요합니다.

- 컨테이너 인스턴스를 시작할 때 컨테이너 인스턴스 사용자 데이터 스크립트에 -EnableTaskIAMRole 옵션을 설정해야 합니다. EnableTaskIAMRole은 태스크에 대한 태스크 IAM 역할 기능을 켭니다. 예:

  ```
  <powershell>
  Import-Module ECSTools
  Initialize-ECSAgent -Cluster 'windows' -EnableTaskIAMRole
  </powershell>
  ```

- Amazon ECS 컨테이너 부트스트랩 스크립트에서 제공되는 네트워킹 명령을 사용하여 컨테이너를 부트스트랩해야 합니다.
- 태스크를 위한 IAM 역할과 정책을 생성해야 합니다. 자세한 내용은 작업 IAM 역할 생성 섹션을 참조하세요.
- 작업 자격 증명 공급자를 위한 IAM 역할은 컨테이너 인스턴스에서 포트 80을 사용합니다. 따라서 컨테이너 인스턴스에서 태스크에 대한 IAM 역할을 구성하면 컨테이너는 어떤 포트 매핑에서도 호스트 포트로 포트 80을 사용할 수 없습니다. 포트 80에서 컨테이너를 노출하려면 로드 밸런싱을 사용하는 컨테이너를 위한 서비스를 구성하는 것이 좋습니다. 로드 밸런서에서 포트 80을 사용할 수 있습니다. 이렇게 하면 트래픽을 컨테이너 인스턴스 상의 다른 호스트 포트로 라우팅할 수 있습니다. 자세한 정보는 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산 섹션을 참조하세요.
- Windows 인스턴스가 다시 시작되면 프록시 인터페이스를 삭제하고 Amazon ECS 컨테이너 에이전트를 다시 초기화하여 자격 증명 프록시를 다시 불러와야 합니다.

Amazon ECS 컨테이너 부트스트랩 스크립트

컨테이너가 컨테이너 인스턴스에서 자격 증명 프록시에 액세스하여 자격 증명을 얻으려면 먼저 필요한 네트워킹 명령을 사용하여 컨테이너를 부트스트랩해야 합니다. 다음 코드 예제 스크립트는 시작할 때 컨테이너에서 실행되어야 합니다.

> **Note**
>
> Windows에서 awsvpc 네트워크 모드를 사용할 때 이 스크립트를 실행할 필요가 없습니다.

Powershell이 포함된 Windows 컨테이너를 실행하는 경우 다음 스크립트를 사용하세요.

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

$gateway = (Get-NetRoute | Where { $_.DestinationPrefix -eq '0.0.0.0/0' } | Sort-Object
 RouteMetric | Select NextHop).NextHop
$ifIndex = (Get-NetAdapter -InterfaceDescription "Hyper-V Virtual Ethernet*" | Sort-
Object | Select ifIndex).ifIndex
New-NetRoute -DestinationPrefix 169.254.170.2/32 -InterfaceIndex $ifIndex -NextHop
 $gateway -PolicyStore ActiveStore # credentials API
New-NetRoute -DestinationPrefix 169.254.169.254/32 -InterfaceIndex $ifIndex -NextHop
 $gateway -PolicyStore ActiveStore # metadata API
```

Command 셸만 있는 Windows 컨테이너를 실행하는 경우 다음 스크립트를 사용합니다.

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
```

```
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

for /f "tokens=1" %i  in ('netsh interface ipv4 show interfaces ^| findstr /x /r
 ".*vEthernet.*"') do set interface=%i
for /f "tokens=3" %i  in ('netsh interface ipv4 show addresses %interface% ^| findstr /
x /r ".*Default.Gateway.*"') do set gateway=%i
netsh interface ipv4 add route prefix=169.254.170.2/32 interface="%interface%"
 nexthop="%gateway%" store=active # credentials API
netsh interface ipv4 add route prefix=169.254.169.254/32 interface="%interface%"
 nexthop="%gateway%" store=active # metadata API
```

Amazon ECS 컨테이너 인스턴스 IAM 역할

Amazon EC2와 외부 인스턴스를 포함한 Amazon ECS 컨테이너 인스턴스는 Amazon ECS 컨테이너 에이전트를 실행하며, 에이전트가 사용자에게 속한다는 것을 서비스에 알리기 위해 IAM 역할이 필요합니다. 컨테이너 인스턴스를 시작해 클러스터에 등록하기 전에 사용할 컨테이너 인스턴스의 IAM 역할을 생성해야 합니다. 역할은 콘솔에 로그인하거나 AWS CLI 명령을 실행하는 데 사용되는 계정에 생성됩니다.

> **Important**
>
> 클러스터에 외부 인스턴스를 등록하는 경우 사용하는 IAM 역할에도 Systems Manager 권한이 필요합니다. 자세한 내용은 Amazon ECS Anywhere IAM 역할 섹션을 참조하세요.

Amazon ECS는 전체 Amazon ECS 기능 세트를 사용하는 데 필요한 권한이 포함된 `AmazonEC2ContainerServiceforEC2Role` 관리형 IAM 정책을 제공합니다. 이 관리형 정책은 IAM 연결에 연결하고, 컨테이너 인스턴스와 연결할 수 있습니다. 또는 사용할 사용자 지정 정책을 생성할 때 관리형 정책을 가이드로 사용할 수 있습니다. 컨테이너 인스턴스 역할은 Amazon ECS 컨테이너 에이전트 및 Docker 대몬이 사용자를 대신하여 AWS API를 호출하는 데 필요한 권한을 제공합니다. 관리형 정책에 대한 자세한 정보는 AmazonEC2ContainerServiceforEC2Role 섹션을 참조하세요.

컨테이너 인스턴스 역할 생성

> **Important**
>
> 클러스터에 외부 인스턴스를 등록하는 경우 Amazon ECS Anywhere IAM 역할 섹션을 참조하세요.

장래 기능 및 개선 사항이 도입될 때 Amazon ECS가 이에 대한 권한을 추가할 수 있도록 하려면 수동으로 역할을 생성하고 컨테이너 인스턴스의 관리형 IAM 정책을 연결해야 합니다. 필요한 경우 다음 절차에 따라 관리형 IAM 정책을 연결합니다.

AWS Management Console

Elastic Container Service에 대한 서비스 역할을 생성하는 방법(IAM 콘솔)

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할을 선택하고 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티 유형에 AWS 서비스를 선택합니다.
4. 서비스 또는 사용 사례의 경우 Elastic Container Service를 선택하고 Elastic Container Service의 EC2 역할 사용 사례를 선택합니다.
5. 다음을 선택합니다.
6. 권한 정책 섹션에서 AmazonEC2ContainerServiceforEC2Role 정책을 선택했는지 확인합니다.

   > **Important**
   >
   > AmazonEC2ContainerServiceforEC2Role 관리형 정책이 컨테이너 인스턴스 IAM 역할에 연결되어야 합니다. 그렇지 않으면 AWS Management Console을 사용하여 클러스터를 생성할 때 오류가 표시됩니다.

7. 다음을 선택합니다.
8. 역할 이름에 ecsInstanceRole을 입력합니다.
9. 역할을 검토한 다음 역할 생성을 선택합니다.

AWS CLI

모든 *### ##*을 고유한 값으로 바꿉니다.

1. 다음 콘텐츠를 가진 `instance-role-trust-policy.json`이라는 파일을 생성합니다:

   JSON

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": { "Service": "ec2.amazonaws.com"},
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

2. 다음 명령을 사용하여 신뢰 정책 문서를 사용해 인스턴스 IAM 역할을 생성합니다.

   ```
   aws iam create-role \
       --role-name ecsInstanceRole \
       --assume-role-policy-document file://instance-role-trust-policy.json
   ```

3. create-instance-profile 명령을 사용하여 `ecsInstanceRole-profile`라는 인스턴스 프로파일을 만듭니다.

   ```
   aws iam create-instance-profile --instance-profile-name ecsInstanceRole-profile
   ```

   응답의 예

   ```
   {
       "InstanceProfile": {
           "InstanceProfileId": "AIPAJTLBPJLEGREXAMPLE",
           "Roles": [],
           "CreateDate": "2022-04-12T23:53:34.093Z",
           "InstanceProfileName": "ecsInstanceRole-profile",
           "Path": "/",
           "Arn": "arn:aws:iam::123456789012:instance-profile/ecsInstanceRole-
   profile"
   ```

```
    }
}
```

4. *ecsInstanceRole* 인스턴스 프로파일에 *ecsInstanceRole-profile* 역할을 추가합니다.

```
aws iam add-role-to-instance-profile \
    --instance-profile-name ecsInstanceRole-profile \
    --role-name ecsInstanceRole
```

5. 다음 명령으로 AmazonEC2ContainerServiceRoleForEC2Role 관리형 정책을 역할에 연결합니다.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role \
    --role-name ecsInstanceRole
```

역할을 생성한 후 다음 기능에 대한 추가 권한을 역할에 추가합니다.

Feature	추가 권한
Amazon ECR에 컨테이너 이미지가 있음	Amazon ECR 권한
CloudWatch Logs에서 컨테이너 인스턴스 모니터링	컨테이너 인스턴스 모니터링 권한
Amazon S3 버킷에서 구성 파일 호스팅	Amazon S3 읽기 전용 액세스

Amazon ECR 권한

컨테이너 인스턴스와 함께 사용하는 Amazon ECS 컨테이너 인스턴스 역할은 Amazon ECR에 대해 다음 IAM 정책 권한을 보유해야 합니다.

JSON

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

```
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

컨테이너 인스턴스에 AmazonEC2ContainerServiceforEC2Role 관리형 정책을 사용하는 경우, 역할에 적절한 권한이 부여됩니다. 역할이 Amazon ECR을 지원하는지 확인하려면 Amazon Elastic Container Service 개발자 안내서의 Amazon ECS 컨테이너 인스턴스 IAM 역할을 참조하세요.

awsvpcTrunking 계정 설정을 설정하는 데 필요한 권한

Amazon ECS에서는 지원되는 Amazon EC2 인스턴스 유형을 사용하여 늘어난 ENI 밀도와 함께 컨테이너 인스턴스를 시작할 수 있습니다. 이 기능을 사용하는 경우 2개의 컨테이너 인스턴스 역할을 생성하는 것이 좋습니다. 역할 하나에 awsvpcTrunking 계정 설정을 활성화하고 ENI 트렁킹이 필요한 작업에 해당 역할을 사용합니다. awsvpcTrunking 계정 설정에 관한 자세한 내용은 계정 설정을 사용하여 Amazon ECS 기능에 액세스 섹션을 참조하세요.

컨테이너 인스턴스와 함께 사용하는 컨테이너 인스턴스 역할에는 계정 설정을 위한 다음 IAM 정책 권한이 있어야 합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:ListAccountSettings",
                "ecs:ListAttributes",
                "ecs:PutAccountSetting"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

컨테이너 인스턴스 역할을 사용하려면 인스턴스 사용자 데이터에 다음을 추가합니다.

```
#!/bin/bash
aws ecs put-account-setting --name awsvpcTrunking --value enabled --region region
ECS_CLUSTER=MyCluster>> /etc/ecs/ecs.config
EOF
```

EC2 인스턴스에 사용자 데이터 추가에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Linux 인스턴스를 시작할 때 명령 실행을 참조하세요.

Amazon S3 읽기 전용 액세스

Amazon S3에서 구성 정보를 프라이빗 버킷에 저장하고 컨테이너 인스턴스 IAM 역할에 읽기 전용 액세스를 부여하면 시작 시점에 컨테이너 인스턴스 구성을 안전하고 편리하게 허용할 수 있습니다. `ecs.config` 파일의 사본을 프라이빗 버킷에 저장했다가 인스턴스가 시작할 때 Amazon EC2 사용자 데이터를 사용하여 AWS CLI를 설치하고 구성 정보를 `/etc/ecs/ecs.config`에 복사할 수 있습니다.

`ecs.config` 파일을 생성하여 Amazon S3에 저장하고 이 구성으로 인스턴스를 시작하는 자세한 방법은 Amazon S3에 Amazon ECS 컨테이너 인스턴스 구성 저장 섹션을 참조하세요.

다음 AWS CLI 명령을 사용하여 컨테이너 인스턴스 역할에 Amazon S3 읽기 전용 액세스를 허용할 수 있습니다. *ecsInstanceRole*을 사용자가 생성한 역할 이름으로 바꿉니다.

```
aws iam attach-role-policy \
      --role-name ecsInstanceRole \
      --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

IAM 콘솔을 사용하여 역할에 Amazon S3 읽기 전용 액세스(`AmazonS3ReadOnlyAccess`)를 추가할 수도 있습니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 역할 권한 업데이트를 참조하세요.

awsvpcTrunking 계정 설정을 설정하는 데 필요한 권한

Amazon ECS 컨테이너 인스턴스와 함께 사용하는 컨테이너 인스턴스 역할에는 계정 설정을 위한 다음 IAM 정책 권한이 있어야 합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:ListAccountSettings",
                "ecs:ListAttributes",
                "ecs:PutAccountSetting"
            ],
            "Resource": "*"
        }
    ]
}
```

컨테이너 인스턴스 모니터링 권한

컨테이너 인스턴스가 로그 데이터를 CloudWatch Logs로 전송할 수 있으려면 먼저 Amazon ECS 에이전트가 고객의 애플리케이션 로그를 CloudWatch(일반적으로 `awslogs` 드라이버를 통해 처리됨)에 쓸 수 있는 IAM 정책을 생성해야 합니다. 정책을 생성한 후 `ecsInstanceRole`에 정책을 연결합니다.

AWS Management Console

JSON 정책 편집기를 사용하여 정책을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 창에서 정책을 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작을 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. 정책 편집기 섹션에서 JSON 옵션을 선택합니다.
5. 다음 JSON 정책 문서를 입력합니다.

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

```
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": ["arn:aws:logs:*:*:*"]
        }
    ]
}
```

6. 다음을 선택합니다.

> **Note**
>
> 언제든지 시각적 편집기 옵션과 JSON 편집기 옵션 간에 전환할 수 있습니다. 그러나 변경을 적용하거나 시각적 편집기에서 다음을 선택한 경우 IAM은 시각적 편집기에 최적화되도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 정책 재구성을 참조하세요.

7. 검토 및 생성 페이지에서 생성하는 정책에 대한 정책 이름과 설명(선택 사항)을 입력합니다. 이 정책에 정의된 권한을 검토하여 정책이 부여한 권한을 확인합니다.
8. 정책 생성을 선택하고 새로운 정책을 저장합니다.

정책을 생성한 후 컨테이너 인스턴스 역할에 정책을 연결합니다. 정책을 역할에 연결하는 방법에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 역할 권한 업데이트를 참조하세요.

AWS CLI

1. 다음 콘텐츠를 통해 `instance-cw-logs.json`이라는 파일을 생성합니다.

   JSON

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

```
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": ["arn:aws:logs:*:*:*"]
        }
    ]
}
```

2. 다음 명령을 사용하여 JSON 정책 문서 파일을 사용해 IAM 정책을 생성합니다.

```
aws iam create-policy \
      --policy-name cwlogspolicy \
      --policy-document file://instance-cw-logs.json
```

3. 다음 명령을 사용하여 생성한 IAM 정책의 ARN을 검색합니다. *cwlogspolicy*를 사용자가 생성한 정책 이름으로 바꿉니다.

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`cwlogspolicy`].Arn'
```

4. 다음 명령을 사용하여 정책 ARN을 사용해 정책을 컨테이너 인스턴스 IAM 역할에 연결합니다.

```
aws iam attach-role-policy \
      --role-name ecsInstanceRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/cwlogspolicy
```

Amazon ECS Anywhere IAM 역할

온프레미스 서버 또는 가상 머신(VM)을 클러스터에 등록하는 경우 서버 또는 가상 머신이 AWS API와 통신하려면 IAM 역할이 필요합니다. 각 AWS 계정에 대해 한 번만 이 IAM 역할을 생성하면 됩니다. 그러나 이 IAM 역할은 클러스터에 등록한 각 서버 또는 VM과 연결되어야 합니다. 이 역할은 `ECSAnywhereRole`입니다. 이 역할을 수동으로 만들 수 있습니다. 또는 AWS Management Console에 외부 인스턴스를 등록할 때 Amazon ECS가 사용자를 대신하여 역할을 생성할 수 있습니다. IAM 콘솔 검색을 사용하여 `ecsAnywhereRole`을 검색하고 계정에 이미 역할이 있는지 확인할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 콘솔 검색을 참조하세요.

AWS는 ECS Anywhere IAM 역할을 생성할 때 사용할 수 있는 두 개의 관리형 IAM 정책인 `AmazonSSMManagedInstanceCore` 및 `AmazonEC2ContainerServiceforEC2Role` 정책을 제공합니다. `AmazonEC2ContainerServiceforEC2Role` 정책에는 필요한 것보다 더 많은 액세스 권한을 제공할 가능성이 있는 권한이 포함되어 있습니다. 따라서 특정 사용 사례에 따라 해당 정책에서 필요한 권한만 추가하는 사용자 지정 정책을 만드는 것이 좋습니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.

태스크 실행 IAM 역할은 Amazon ECS 컨테이너 에이전트에 사용자를 대신하여 AWS API 호출을 수행할 권한을 부여합니다. 태스크 실행 IAM 역할을 사용하는 경우 태스크 정의에 IAM 역할을 지정해야 합니다. 자세한 정보는 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

다음 조건 중 하나라도 적용되는 경우 태스크 실행 역할이 필요합니다.

- `awslogs` 로그 드라이버를 사용해 CloudWatch Logs에 컨테이너 로그를 보냅니다.
- 태스크 정의는 Amazon ECR 프라이빗 리포지토리에서 호스팅되는 컨테이너 이미지를 지정합니다. 그러나 외부 인스턴스와 연결된 `ECSAnywhereRole` 역할에는 Amazon ECR에서 이미지를 가져오는 데 필요한 권한도 포함되어 있으므로 작업 실행 역할에 이미지를 포함할 필요가 없습니다.

Amazon ECS Anywhere 역할 생성

모든 *### ##*을 사용자 정보로 바꿉니다.

1. 다음 신뢰 정책을 통해 이름이 `ssm-trust-policy.json`인 로컬 파일을 생성합니다.

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"Service": [
      "ssm.amazonaws.com"
    ]},
    "Action": "sts:AssumeRole"
  }
}
```

2. 다음 AWS CLI 명령을 사용하여 역할을 생성한 후 신뢰 정책에 연결합니다.

```
aws iam create-role --role-name ecsAnywhereRole --assume-role-policy-document
 file://ssm-trust-policy.json
```

3. 다음 명령을 사용하여 AWS 관리형 정책을 연결합니다.

```
aws iam attach-role-policy --role-name ecsAnywhereRole --policy-arn
 arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
aws iam attach-role-policy --role-name ecsAnywhereRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role
```

IAM 사용자 지정 신뢰 정책 워크플로를 사용하여 역할을 생성할 수도 있습니다. 지침은 IAM 사용자 설명서의 사용자 지정 신뢰 정책을 사용하여 역할 생성(콘솔)을 참조하세요.

Amazon ECS 인프라 IAM 역할

Amazon ECS 인프라 IAM 역할을 사용하면 Amazon ECS가 사용자를 대신하여 클러스터의 인프라 리소스를 관리할 수 있으며, 다음과 같은 경우에 사용됩니다.

- Amazon EBS 볼륨을 Fargate 또는 EC2 시작 유형의 Amazon ECS 작업에 연결하려고 합니다. 인프라 역할을 통해 Amazon ECS에서 작업에 사용할 Amazon EBS 볼륨을 관리할 수 있습니다.
- 전송 계층 보안(TLS)을 사용하여 Amazon ECS Service Connect 서비스 간 트래픽을 암호화하려고 합니다.
- Amazon VPC Lattice 대상 그룹을 생성하려는 참입니다.

Amazon ECS에서 이 역할을 수임하여 사용자를 대신해 작업을 수행하면 AWS CloudTrail에 이벤트가 표시됩니다. Amazon ECS가 이 역할을 사용하여 작업에 연결된 Amazon EBS 볼륨을 관리하는 경우 CloudTrail 로그 `roleSessionName`은 `ECSTaskVolumesForEBS`입니다. 역할을 사용하여 Service Connect 서비스 간 트래픽을 암호화하는 경우 CloudTrail 로그 `roleSessionName`은 `ECSServiceConnectForTLS`입니다. 역할을 사용하여 VPC Lattice의 대상 그룹을 생성하는 경우 CloudTrail 로그 `roleSessionName`은 `ECSNetworkingWithVPCLattice`입니다. 이 이름을 사용하여 사용자 이름을 필터링해 CloudTrail 콘솔에서 이벤트를 검색할 수 있습니다.

Amazon ECS에서는 볼륨 연결과 TLS에 필요한 권한을 포함하는 관리형 정책을 제공합니다. 자세한 내용은 AWS 관리형 정책 참조 안내서의 AmazonECSInfrastructureRolePolicyForVolumes, AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity 및 AmazonECSInfrastructureRolePolicyForVpcLattice를 참조하세요.

Amazon ECS 인프라 역할 생성

모든 *### ##*을 사용자 정보로 바꿉니다.

1. IAM 역할에 사용할 신뢰 정책이 포함된 `ecs-infrastructure-trust-policy.json`이라는 이름의 파일을 생성합니다. 파일에 다음을 포함해야 합니다.

   JSON

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "AllowAccessToECSForInfrastructureManagement",
         "Effect": "Allow",
         "Principal": {
           "Service": "ecs.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

2. 다음 AWS CLI 명령을 사용하여 이전 단계에서 생성한 신뢰 정책을 사용해 `ecsInfrastructureRole`이라는 이름의 역할을 생성합니다.

   ```
   aws iam create-role \
         --role-name ecsInfrastructureRole \
         --assume-role-policy-document file://ecs-infrastructure-trust-policy.json
   ```

3. 사용 사례에 따라 AWS 관리형 `AmazonECSInfrastructureRolePolicyForVolumes`, `AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity` 또는 `AmazonECSInfrastructureRolePolicyForVpcLattice` 정책을 `ecsInfrastructureRole` 역할에 연결합니다.

   ```
   aws iam attach-role-policy \
         --role-name ecsInfrastructureRole \
         --policy-arn arn:aws:iam::aws:policy/service-role/
   AmazonECSInfrastructureRolePolicyForVolumes
   ```

```
aws iam attach-role-policy \
      --role-name ecsInfrastructureRole \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity
```

IAM 콘솔의 사용자 지정 신뢰 정책 워크플로를 사용하여 역할을 생성할 수도 있습니다. 지침은 IAM 사용자 설명서의 사용자 지정 신뢰 정책을 사용하여 역할 생성(콘솔)을 참조하세요.

Important

Amazon ECS에서 ECS 인프라 역할을 사용해 작업에 연결된 Amazon EBS 볼륨을 관리하는 경우 Amazon EBS 볼륨을 사용하는 작업을 중지하기 전에 다음 사항을 확인합니다.

- 역할은 삭제되지 않습니다.
- 역할에 대한 신뢰 정책은 Amazon ECS 액세스 권한(`ecs.amazonaws.com`)을 제거하도록 수정되지 않습니다.
- 관리형 정책 `AmazonECSInfrastructureRolePolicyForVolumes`는 제거되지 않습니다. 역할의 권한을 수정해야 하는 경우 볼륨 삭제를 위해 적어도 `ec2:DetachVolume`, `ec2:DeleteVolume`, `ec2:DescribeVolumes`를 유지합니다.

Amazon EBS 볼륨이 연결된 작업을 중지하기 전에 역할을 삭제하거나 수정하면 작업이 `DEPROVISIONING` 상태로 멈추고 관련 Amazon EBS 볼륨이 삭제되지 않습니다. Amazon ECS는 필요한 권한이 복원될 때까지 작업을 중지하고 볼륨을 삭제하기 위해 정기적으로 자동 재시도합니다. DescribeTasks API를 사용하여 작업의 볼륨 연결 상태 및 관련 상태 이유를 볼 수 있습니다.

파일을 생성한 후 Amazon ECS로 역할을 전달할 수 있는 권한을 사용자에게 부여해야 합니다.

인프라 역할을 Amazon ECS로 전달하는 권한

ECS 인프라 IAM 역할을 사용하려면 Amazon ECS로 역할을 전달할 수 있는 권한을 사용자에게 부여해야 합니다. 사용자에게 다음 `iam:PassRole` 권한을 연결합니다. *ecsInfrastructureRole*을 사용자가 생성한 인프라 역할 이름으로 바꿉니다.

JSON

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

        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": ["arn:aws:iam::*:role/ecsInfrastructureRole"],
            "Condition": {
                "StringEquals": {"iam:PassedToService": "ecs.amazonaws.com"}
            }
        }
    ]
}
```

`iam:Passrole` 및 사용자 권한 업데이트에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 사용자에게 AWS 서비스에 역할을 전달할 권한 부여 및 IAM 사용자의 권한 변경을 참조하세요.

로드 밸런서에 대한 Amazon ECS 인프라 IAM 역할

로드 밸런서에 대한 Amazon ECS 인프라 IAM 역할을 사용하면 Amazon ECS가 사용자를 대신해 클러스터의 로드 밸런서를 관리할 수 있으며, 이 역할은 다음과 같은 경우에 사용됩니다.

- Amazon ECS에서 블루/그린 배포를 사용하길 원합니다. 인프라 역할을 사용하면 Amazon ECS가 배포에 대한 로드 밸런서 리소스를 관리할 수 있습니다.
- 배포 작업 중에 대상 그룹 및 리스너와 같은 로드 밸런서 리소스를 생성, 수정 또는 삭제하려면 Amazon ECS가 필요합니다.

Amazon ECS에서 이 역할을 수임하여 사용자를 대신해 작업을 수행하면 AWS CloudTrail에 이벤트가 표시됩니다. Amazon ECS가 역할을 사용하여 블루/그린 배포에 대한 로드 밸런서 리소스를 관리하는 경우 CloudTrail 로그 `roleSessionName`은 `ECSNetworkingWithELB` 또는 `ecs-service-scheduler`입니다. 이 이름을 사용하여 사용자 이름을 필터링해 CloudTrail 콘솔에서 이벤트를 검색할 수 있습니다.

Amazon ECS는 로드 밸런서 관리에 필요한 권한을 포함하는 관리형 정책
을 제공합니다. 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 안내서의
AmazonECSInfrastructureRolePolicyForLoadBalancers를 참조하세요.

로드 밸런서에 대한 Amazon ECS 인프라 역할 생성

모든 ### ##을 사용자 정보로 바꿉니다.

1. IAM 역할에 사용할 신뢰 정책이 포함된 ecs-infrastructure-trust-policy.json이라는 이름의 파일을 생성합니다. 파일에 다음을 포함해야 합니다.

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToECSForInfrastructureManagement",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. 다음 AWS CLI 명령을 사용하여 이전 단계에서 생성한 신뢰 정책을 사용해 ecsInfrastructureRoleForLoadBalancers이라는 이름의 역할을 생성합니다.

```
aws iam create-role \
      --role-name ecsInfrastructureRoleForLoadBalancers \
      --assume-role-policy-document file://ecs-infrastructure-trust-policy.json
```

3. AWS 관리형 AmazonECSInfrastructureRolePolicyForLoadBalancers 정책을 ecsInfrastructureRoleForLoadBalancers 역할에 연결합니다.

```
aws iam attach-role-policy \
      --role-name ecsInfrastructureRoleForLoadBalancers \
      --policy-arn arn:aws:iam::aws:policy/
AmazonECSInfrastructureRolePolicyForLoadBalancers
```

IAM 콘솔의 사용자 지정 신뢰 정책 워크플로를 사용하여 역할을 생성할 수도 있습니다. 지침은 IAM 사용자 설명서의 사용자 지정 신뢰 정책을 사용하여 역할 생성(콘솔)을 참조하세요.

Important

Amazon ECS에서 블루/그린 배포에 대한 로드 밸런서 리소스를 관리하기 위해 인프라 역할을 사용하는 경우 역할을 삭제하거나 수정하기 전에 다음을 확인합니다.

- 활성 배포가 진행 중인 동안에 역할은 삭제되지 않습니다.
- 역할에 대한 신뢰 정책은 Amazon ECS 액세스 권한(`ecs.amazonaws.com`)을 제거하도록 수정되지 않습니다.
- 활성 배포가 진행되는 동안에 관리형 정책 `AmazonECSInfrastructureRolePolicyForLoadBalancers`는 제거되지 않습니다.

활성 블루/그린 배포 중에 역할을 삭제하거나 수정하면 배포에 실패하고 서비스가 일관되지 않은 상태로 남을 수 있습니다.

파일을 생성한 후 Amazon ECS로 역할을 전달할 수 있는 권한을 사용자에게 부여해야 합니다.

인프라 역할을 Amazon ECS로 전달하는 권한

로드 밸런서에 대해 ECS 인프라 IAM 역할을 사용하려면 Amazon ECS로 역할을 전달할 권한을 사용자에게 부여해야 합니다. 사용자에게 다음 `iam:PassRole` 권한을 연결합니다. *`ecsInfrastructureRoleForLoadBalancers`*를 사용자가 생성한 인프라 역할 이름으로 바꿉니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource":
 ["arn:aws:iam::*:role/ecsInfrastructureRoleForLoadBalancers"],
            "Condition": {
                "StringEquals": {"iam:PassedToService": "ecs.amazonaws.com"}
```

```
            }
        }
    ]
}
```

`iam:Passrole` 및 사용자 권한 업데이트에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 사용자에게 AWS 서비스에 역할을 전달할 권한 부여 및 IAM 사용자의 권한 변경을 참조하세요.

Amazon ECS CodeDeploy IAM 역할

Amazon ECS와 함께 CodeDeploy 블루/그린 배포 유형을 사용하기 전에 CodeDeploy 서비스는 사용자를 대신하여 Amazon ECS 서비스를 업데이트할 수 있는 권한이 필요합니다. 이러한 권한은 CodeDeploy IAM 역할(`ecsCodeDeployRole`)에 의해 제공됩니다.

> **Note**
>
> 사용자는 CodeDeploy를 사용할 권한도 필요합니다. 이러한 권한은 필수 IAM 권한에 설명되어 있습니다.

관리형 정책은 2가지가 있습니다. 자세한 내용은 AWS 관리형 정책 참조 안내서의 다음 중 하나를 참조하세요.

- AWSCodeDeployRoleForECS - CodeDeploy에 연결된 작업을 사용해 리소스를 업데이트하는 권한을 제공합니다.
- AWSCodeDeployRoleForECSLimited - CodeDeploy에 더 제한된 권한을 제공합니다.

CodeDeploy 역할 생성

다음 절차에 따라 Amazon ECS에 대한 CodeDeploy 역할을 생성할 수 있습니다.

AWS Management Console

CodeDeploy에 대한 서비스 역할을 생성하는 방법(IAM 콘솔)

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할을 선택하고 역할 생성을 선택합니다.

3. 신뢰할 수 있는 엔터티 유형에 AWS 서비스를 선택합니다.
4. 서비스 또는 사용 사례에서 CodeDeploy를 선택하고 CodeDeploy - ECS 사용 사례를 선택합니다.
5. 다음을 선택합니다.
6. 권한 정책 연결 섹션에서 AWSCodeDeployRoleForECS 정책이 선택되어 있는지 확인합니다.
7. 다음을 선택합니다.
8. 역할 이름에 ecsCodeDeployRole을 입력합니다.
9. 역할을 검토한 다음 역할 생성을 선택합니다.

AWS CLI

모든 *### ##*을 사용자 정보로 바꿉니다.

1. CodeDeploy IAM 역할에 사용할 신뢰 정책이 포함된 `codedeploy-trust-policy.json`이라는 이름의 파일을 생성합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": ["codedeploy.amazonaws.com"]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. 이전 단계에서 만든 신뢰 정책을 사용하여 `ecsCodedeployRole`이라는 IAM 역할을 생성합니다.

```
aws iam create-role \
      --role-name ecsCodedeployRole \
      --assume-role-policy-document file://codedeploy-trust-policy.json
```

3. AWSCodeDeployRoleForECS 또는 AWSCodeDeployRoleForECSLimited 관리형 정책을 ecsTaskRole 역할에 연결합니다.

```
aws iam attach-role-policy \
      --role-name ecsCodedeployRole \
      --policy-arn arn:aws:iam::aws:policy/AWSCodeDeployRoleForECS
```

```
aws iam attach-role-policy \
      --role-name ecsCodedeployRole \
      --policy-arn arn:aws:iam::aws:policy/AWSCodeDeployRoleForECSLimited
```

서비스의 작업에 작업 실행 역할이 필요한 경우 각 작업 실행 역할 또는 작업 역할 재정의에 대해 iam:PassRole 권한을 CodeDeploy 역할에 정책으로 추가해야 합니다.

작업 실행 역할 권한

서비스의 작업에 작업 실행 역할이 필요한 경우 각 작업 실행 역할 또는 작업 역할 재정의에 대해 iam:PassRole 권한을 CodeDeploy 역할에 정책으로 추가해야 합니다. 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 및 Amazon ECS 작업 IAM 역할(을)를 참조하세요. 그런 다음, 해당 정책을 CodeDeploy 역할에 연결합니다.

정책을 생성합니다.

AWS Management Console

JSON 정책 편집기를 사용하여 정책을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 창에서 정책을 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작을 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. 정책 편집기 섹션에서 JSON 옵션을 선택합니다.
5. 다음 JSON 정책 문서를 입력합니다.

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["arn:aws:iam::<aws_account_id>:role/
<ecsCodeDeployRole>"]
        }
    ]
}
```

6. 다음을 선택합니다.

> **Note**
>
> 언제든지 시각적 편집기 옵션과 JSON 편집기 옵션 간에 전환할 수 있습니다. 그러나 변경을 적용하거나 시각적 편집기에서 다음을 선택한 경우 IAM은 시각적 편집기에 최적화되도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 정책 재구성을 참조하세요.

7. 검토 및 생성 페이지에서 생성하는 정책에 대한 정책 이름과 설명(선택 사항)을 입력합니다. 이 정책에 정의된 권한을 검토하여 정책이 부여한 권한을 확인합니다.
8. 정책 생성을 선택하고 새로운 정책을 저장합니다.

정책을 생성한 후 CodeDeploy 역할에 정책을 연결합니다. 정책을 역할에 연결하는 방법에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 역할 권한 업데이트를 참조하세요.

AWS CLI

모든 `### ##`을 사용자 정보로 바꿉니다.

1. 다음 콘텐츠를 통해 `blue-green-iam-passrole.json`이라는 파일을 생성합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": "iam:PassRole",
            "Resource": [
                  "arn:arn:aws:iam::111122223333:role/ecs-code-deploy-
role"
            ]
        }
    ]
}
```

2. 다음 명령을 사용하여 JSON 정책 문서 파일을 사용해 IAM 정책을 생성합니다.

```
aws iam create-policy \
      --policy-name cdTaskExecutionPolicy \
      --policy-document file://blue-green-iam-passrole.json
```

3. 다음 명령을 사용하여 생성한 IAM 정책의 ARN을 검색합니다.

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`cdTaskExecutionPolicy`].Arn'
```

4. 다음 명령을 사용하여 CodeDeploy IAM 역할에 정책을 연결합니다.

```
aws iam attach-role-policy \
      --role-name ecsCodedeployRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/cdTaskExecutionPolicy
```

Amazon ECS EventBridge IAM 역할

EventBridge 규칙 및 대상과 함께 Amazon ECS의 예약된 작업을 사용하려면 먼저 EventBridge 서비스에 사용자를 대신하여 Amazon ECS 작업을 실행할 수 있는 권한이 필요합니다. 이러한 권한은 EventBridge IAM 역할(`ecsEventsRole`)에 의해 제공됩니다.

`AmazonEC2ContainerServiceEventsRole` 정책은 아래 나와 있습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": ["ecs:RunTask"],
            "Resource": ["*"]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["*"],
            "Condition": {
                "StringLike": {"iam:PassedToService": "ecs-tasks.amazonaws.com"}
            }
        },
        {
            "Effect": "Allow",
            "Action": "ecs:TagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:CreateAction": ["RunTask"]
                }
            }
        }
    ]
}
```

예약된 작업에서 작업 실행 역할, 작업 역할 또는 작업 역할 재정의를 사용해야 하는 경우 각 작업 실행 역할, 작업 역할 또는 작업 역할 재정의를 위한 `iam:PassRole` 권한을 EventBridge IAM 역할에 추가해야 합니다. 태스크 실행 역할에 대한 자세한 정보는 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.

> **Note**
>
> 태스크 실행 역할 또는 태스크 역할 재정의의 전체 ARN을 지정합니다.

JSON

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

```
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [

 "arn:aws:iam::111122223333:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
}
```

예약된 작업을 구성할 때 AWS Management Console에서 EventBridge 역할을 생성하도록 선택할 수 있습니다. 자세한 내용은 Amazon EventBridge Scheduler를 사용하여 Amazon ECS 태스크 예약 섹션을 참조하세요.

EventBridge 역할 생성

모든 ### ##을 사용자 정보로 바꿉니다.

1. IAM 역할에 사용할 신뢰 정책이 포함된 eventbridge-trust-policy.json이라는 이름의 파일을 생성합니다. 파일에 다음을 포함해야 합니다.

   JSON

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "",
         "Effect": "Allow",
         "Principal": {
           "Service": "events.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

2. 다음 명령을 사용하여 이전 단계에서 생성한 신뢰 정책을 사용해 ecsEventsRole이라는 이름의 IAM 역할을 생성합니다.

   ```
   aws iam create-role \
   ```

```
      --role-name ecsEventsRole \
      --assume-role-policy-document file://eventbridge-trust-policy.json
```

3. 다음 명령을 사용하여 AWS 관리형 AmazonEC2ContainerServiceEventsRole을 ecsEventsRole 역할에 연결합니다.

```
aws iam attach-role-policy \
      --role-name ecsEventsRole \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceEventsRole
```

또한 IAM 콘솔의 사용자 지정 신뢰 정책 워크플로(https://console.aws.amazon.com/iam/)를 사용하여 역할을 생성할 수 있습니다. 지침은 IAM 사용자 설명서의 사용자 지정 신뢰 정책을 사용하여 역할 생성(콘솔)을 참조하세요.

ecsEventsRole 역할에 정책 연결

다음 절차를 사용하여 작업 실행 역할에 대한 권한을 EventBridge IAM 역할에 추가할 수 있습니다.

AWS Management Console

JSON 정책 편집기를 사용하여 정책을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 창에서 정책을 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작을 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. 정책 편집기 섹션에서 JSON 옵션을 선택합니다.
5. 다음 JSON 정책 문서를 입력합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["arn:aws:iam::<aws_account_id>:role/
<ecsTaskExecutionRole_or_TaskRole_name>"]
```

```
        }
    ]
}
```

6. 다음을 선택합니다.

> **Note**
>
> 언제든지 시각적 편집기 옵션과 JSON 편집기 옵션 간에 전환할 수 있습니다. 그러나 변경을 적용하거나 시각적 편집기에서 다음을 선택한 경우 IAM은 시각적 편집기에 최적화되도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 정책 재구성을 참조하세요.

7. 검토 및 생성 페이지에서 생성하는 정책에 대한 정책 이름과 설명(선택 사항)을 입력합니다. 이 정책에 정의된 권한을 검토하여 정책이 부여한 권한을 확인합니다.
8. 정책 생성을 선택하고 새로운 정책을 저장합니다.

정책을 생성한 후 EventBridge 역할에 정책을 연결합니다. 정책을 역할에 연결하는 방법에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 역할 권한 업데이트를 참조하세요.

AWS CLI

모든 *### ##*을 사용자 정보로 바꿉니다.

1. 다음 콘텐츠를 통해 `ev-iam-passrole.json`이라는 파일을 생성합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [

 "arn:aws:iam::111122223333:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
```

```
    ]
}
```

2. 다음 AWS CLI 명령을 사용하여 JSON 정책 문서 파일을 사용해 IAM 정책을 생성합니다.

```
aws iam create-policy \
      --policy-name eventsTaskExecutionPolicy \
      --policy-document file://ev-iam-passrole.json
```

3. 다음 명령을 사용하여 생성한 IAM 정책의 ARN을 검색합니다.

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`eventsTaskExecutionPolicy`].Arn'
```

4. 다음 명령을 사용하여 정책 ARN을 사용해 정책을 EventBridge IAM 역할에 연결합니다.

```
aws iam attach-role-policy \
      --role-name ecsEventsRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/eventsTaskExecutionPolicy
```

Amazon ECS 콘솔에 필요한 권한

최소 권한을 부여하는 모범 사례를 따르면, AmazonECS_FullAccess 관리형 정책을 사용자 고유의 사용자 지정 정책을 생성하기 위한 템플릿으로 사용할 수 있습니다. 이렇게 하면 특정 요구 사항에 따라 관리형 정책에서 권한을 제거하거나 추가할 수 있습니다. 자세한 내용은 AWS Managed Policy 참조 안내서의 AmazonECS_FullAccess를 참조하세요.

IAM 역할을 생성하기 위한 권한

작업을 완료하려면 다음 조치를 위한 추가 권한이 필요합니다.

- 외부 인스턴스 등록 - 자세한 내용은 Amazon ECS Anywhere IAM 역할을 참조하세요.
- 작업 정의 등록 - 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할을 참조하세요.
- 작업 예약에 사용할 EventBridge 규칙 생성 - 자세한 내용은 Amazon ECS EventBridge IAM 역할을 참조하세요.

이러한 권한을 Amazon ECS 콘솔에서 사용하려면 먼저 IAM에서 역할을 생성하여 이러한 권한을 추가할 수 있습니다. 역할을 생성하지 않으면 Amazon ECS 콘솔이 사용자를 대신하여 역할을 생성합니다.

클러스터에 외부 인스턴스를 등록하는 데 필요한 권한

외부 인스턴스를 클러스터에 등록하고 새 외부 인스턴스(ecsExternalInstanceRole) 역할을 생성하려면 추가 권한이 필요합니다.

다음과 같은 추가 권한이 필요합니다.

- iam – 보안 주체가 IAM 역할 및 연결된 정책을 생성 및 나열할 수 있습니다.
- ssm - 보안 주체가 외부 인스턴스를 Systems Manager에 등록할 수 있습니다.

> **Note**
>
> 기존 ecsExternalInstanceRole을 선택하려면 iam:GetRole 및 iam:PassRole 권한이 있어야 합니다.

다음 정책은 필요한 권한을 포함하여 ecsExternalInstanceRole 역할로 작업을 제한합니다.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "iam:AttachRolePolicy",
              "iam:CreateRole",
              "iam:CreateInstanceProfile",
              "iam:AddRoleToInstanceProfile",
              "iam:ListInstanceProfilesForRole",
              "iam:GetRole"
            ],
            "Resource": "arn:aws:iam::*:role/ecsExternalInstanceRole"
        },
        {
            "Effect": "Allow",
            "Action": ["iam:PassRole","ssm:CreateActivation"],
            "Resource": "arn:aws:iam::*:role/ecsExternalInstanceRole"
        }
    ]
}
```

작업 정의를 등록하는 데 필요한 권한

작업 정의를 등록하고 새 작업 실행(ecsTaskExecutionRole) 역할을 생성하려는 경우 추가 권한이 필요합니다.

다음과 같은 추가 권한이 필요합니다.

- iam – 보안 주체가 IAM 역할 및 연결된 정책을 생성 및 나열할 수 있습니다.

> **Note**
>
> 기존 ecsTaskExecutionRole을 선택하려면 iam:GetRole 권한이 있어야 합니다.

다음 정책은 필요한 권한을 포함하여 ecsTaskExecutionRole 역할로 작업을 제한합니다.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "iam:AttachRolePolicy",
              "iam:CreateRole",
              "iam:GetRole"
            ],
            "Resource": "arn:aws:iam::*:role/ecsTaskExecutionRole"
        }
    ]
}
```

예약된 작업에 대한 EventBridge 규칙을 생성하는 데 필요한 권한

작업을 예약하고 새 CloudWatch Events 역할(ecsEventsRole)을 생성하려면 추가 권한이 필요합니다.

다음과 같은 추가 권한이 필요합니다.

- iam - 보안 주체가 IAM 역할 및 연결된 정책을 생성 및 나열하고, Amazon ECS가 역할을 다른 서비스에 전달하여 역할을 수임하도록 허용할 수 있습니다.

> **Note**
>
> 기존 ecsEventsRole을 선택하려면 iam:GetRole 및 iam:PassRole 권한이 있어야 합니다.

다음 정책은 필요한 권한을 포함하여 ecsEventsRole 역할로 작업을 제한합니다.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "iam:AttachRolePolicy",
              "iam:CreateRole",
              "iam:GetRole",
              "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::*:role/ecsEventsRole"
        }
    ]
}
```

서비스 배포를 보는 데 필요한 권한

최소 권한 부여의 모범 사례를 따는 경우 콘솔에서 서비스 배포를 보려면 추가 권한을 추가해야 합니다.

다음과 같은 조치에 대한 액세스 권한이 필요합니다.

- ListServiceDeployments
- DescribeServiceDeployments
- DescribeServiceRevisions

다음과 같은 리소스에 대한 액세스 권한이 필요합니다.

- Service
- 서비스 배포
- 서비스 개정

다음 예제 정책에는 필요한 권한이 있으며, 지정된 서비스로 조치가 제한됩니다.

`account`, `cluster-name` 및 `service-name`을 원하는 값으로 바꿉니다.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "ecs:ListServiceDeployments",
              "ecs:DescribeServiceDeployments",
              "ecs:DescribeServiceRevisions"
            ],
            "Resource": [
             "arn:aws:ecs:us-east-1:123456789012:service/cluster-name/service-name",
             "arn:aws:ecs:us-east-1:123456789012:service-deployment/cluster-name/
service-name/*",
             "arn:aws:ecs:us-east-1:123456789012:service-revision/cluster-name/service-
name/*"
            ]
        }
    ]
}
```

Container Insights에서 Amazon ECS 수명 주기 이벤트를 확인하기 위해 필요한 권한

수명 주기 이벤트를 보려면 다음의 권한이 필요합니다. 다음 권한을 역할에 인라인 정책으로 추가합니다. 자세한 정보는 IAM 정책 추가 및 제거 섹션을 참조하세요.

- events:DescribeRule
- events:ListTargetsByRule
- logs:DescribeLogGroups

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "events:DescribeRule",
        "events:ListTargetsByRule",
        "logs:DescribeLogGroups"
      ],
      "Resource": "*"
    }
  ]
}
```

Container Insights에서 Amazon ECS 수명 주기 이벤트를 활성화하기 위해 필요한 권한

수명 주기 이벤트를 구성하려면 다음의 권한이 필요합니다.

- events:PutRule
- events:PutTargets
- logs:CreateLogGroup

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutRule",
        "events:PutTargets",
        "logs:CreateLogGroup"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS CloudFormation이 있는 Amazon ECS 콘솔에 필요한 권한

AWS Management Console을 사용하여 리소스를 생성하기 전에 올바른 IAM 권한이 있는지 확인해야 합니다. 일반적으로 Amazon ECS 콘솔에 대한 권한을 설정하는 방법에 대한 자세한 내용은 Amazon ECS 콘솔에 필요한 권한을 참조하세요.

Amazon ECS 콘솔은 AWS CloudFormation을 기반으로 하며 다음과 같은 경우 추가 IAM 권한이 필요합니다.

- 클러스터 생성
- 서비스 생성
- 용량 공급자 생성

추가 권한에 대한 정책을 생성한 다음 콘솔에 액세스하는 데 사용하는 IAM 역할에 연결할 수 있습니다. 자세한 내용은 IAM 사용자 설명서에서 IAM 정책 생성을 참조하세요.

클러스터를 생성하는 데 필요한 권한

콘솔에서 클러스터를 생성하는 경우 AWS CloudFormation 스택을 관리할 수 있는 권한을 부여하는 추가 권한이 필요합니다.

다음과 같은 추가 권한이 필요합니다.

- `cloudformation` – 보안 주체가 AWS CloudFormation 스택을 생성 및 관리할 수 있습니다. 이는 AWS Management Console 및 이러한 클러스터의 후속 관리 태스크를 사용하여 Amazon ECS 클러스터를 생성할 때 필요합니다.
- `ssm` - AWS CloudFormation가 최신 Amazon ECS 최적화 AMI를 참조할 수 있도록 허용합니다. 이는 AWS Management Console를 사용하여 Amazon ECS 클러스터를 생성할 때 필요합니다.

다음 정책은 필요한 AWS CloudFormation 권한을 포함하여 Amazon ECS 콘솔에서 생성된 리소스로 작업을 제한합니다.

```
{
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
```

```
                "cloudformation:DescribeStack*",
                "cloudformation:UpdateStack"
             ],
            "Resource": [
                "arn:*:cloudformation:*:*:stack/Infra-ECS-Cluster-*"
            ]
      },
      {
          "Effect": "Allow",
          "Action": "ssm:GetParameters",
          "Resource": [
            "arn:aws:ssm:*:*:parameter/aws/service/ecs/optimized-ami/amazon-linux-2*/
*",
            "arn:aws:ssm:*:*:parameter/aws/service/ecs/optimized-ami/amazon-
linux-2023*/*"
          ]
      }
   ]
}
```

Amazon ECS 컨테이너 인스턴스 역할(ecsInstanceRole)을 생성하지 않고 Amazon EC2 인스턴스를 사용하는 클러스터를 생성하는 경우 콘솔에서 사용자를 대신하여 역할을 생성합니다.

또한 Auto Scaling 그룹을 사용하는 경우에는 클러스터 Auto Scaling 기능을 사용할 때 콘솔이 Auto Scaling 그룹에 태그를 추가할 수 있도록 추가 권한이 필요합니다.

다음과 같은 추가 권한이 필요합니다.

- autoscaling - 콘솔에서 Amazon EC2 Auto Scaling 그룹에 태그를 지정할 수 있습니다. 클러스터 Auto Scaling 기능을 사용할 때 Amazon EC2 auto scaling을 관리할 때 필요합니다. 태그는 콘솔에서 생성되었음을 나타내기 위해 콘솔이 그룹에 자동으로 추가하는 ECS 관리형 태그입니다.
- iam – 보안 주체가 IAM 역할 및 연결된 정책을 나열할 수 있습니다. 또한 보안 주체는 Amazon EC2 인스턴스에서 사용 가능한 인스턴스 프로파일을 나열할 수 있습니다.

다음 정책은 필요한 IAM 권한을 포함하여 ecsInstanceRole 역할로 작업을 제한합니다.

Auto Scaling 권한은 제한되지 않습니다.

```
{
  "Statement": [
      {
```

```
          "Effect": "Allow",
            "Action": [
              "iam:AttachRolePolicy",
              "iam:CreateRole",
              "iam:CreateInstanceProfile",
              "iam:AddRoleToInstanceProfile",
              "iam:ListInstanceProfilesForRole",
              "iam:GetRole"
            ],
            "Resource": "arn:aws:iam::*:role/ecsInstanceRole"
        },
        {
            "Effect": "Allow",
            "Action": "autoscaling:CreateOrUpdateTags",
            "Resource": "*"
        }
    ]
}
```

서비스를 생성하는 데 필요한 권한

콘솔에서 서비스를 생성하는 경우 AWS CloudFormation 스택을 관리할 수 있는 권한을 부여하는 추가 권한이 필요합니다. 다음과 같은 추가 권한이 필요합니다.

- `cloudformation` – 보안 주체가 AWS CloudFormation 스택을 생성 및 관리할 수 있습니다. 이는 AWS Management Console을 사용하여 Amazon ECS 서비스를 생성하고 이후에 해당 서비스를 관리할 때 필요합니다.

다음 정책은 필요한 권한을 포함하여 Amazon ECS 콘솔에서 생성된 리소스로 작업을 제한합니다.

```
{
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStack*",
                "cloudformation:UpdateStack"
             ],
            "Resource": [
                "arn:*:cloudformation:*:*:stack/ECS-Console-V2-Service-*"
```

```
            ]
        }
    ]
}
```

Amazon ECS 블루/그린 배포에서 Lambda 함수에 필요한 권한

Amazon ECS 블루/그린 배포에서 Lambda 함수를 배포 수명 주기 후크로 사용하는 경우 특정 권한이 있는 IAM 역할을 생성해야 합니다. 이 역할을 사용하면 Amazon ECS가 배포 수명 주기의 다양한 단계에서 Lambda 함수를 간접 호출할 수 있습니다.

다음과 같은 추가 권한이 필요합니다.

- `lambda:InvokeFunction` - Amazon ECS가 배포 프로세스 중에 수명 주기 후크로 구성된 Lambda 함수를 간접 호출할 수 있습니다.

다음 정책은 Lambda 수명 주기 후크에 필요한 권한을 포함합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:*:*:function:*"
        }
    ]
}
```

신뢰 정책에 대해 서비스가 다음 역할을 수임하도록 허용해야 합니다.

JSON

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ecs.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Amazon ECS 서비스 Auto Scaling에 필요한 IAM 권한

서비스 Auto Scaling은 Amazon ECS, CloudWatch 및 Application Auto Scaling API의 조합을 통해 수행됩니다. 서비스는 Amazon ECS를 통해 생성 및 업데이트됩니다. CloudWatch는 경보를 생성합니다. 크기 조정 정책은 Application Auto Scaling을 통해 생성됩니다.

서비스 생성 및 업데이트를 위한 표준 IAM 권한 외에도 다음 예제 정책과 같이 Service Auto Scaling 설정과 상호 작용하는 데 다음 권한이 필요합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:*",
                "ecs:DescribeServices",
                "ecs:UpdateService",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:DescribeAlarmHistory",
                "cloudwatch:DescribeAlarmsForMetric",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "cloudwatch:DisableAlarmActions",
                "cloudwatch:EnableAlarmActions",
                "iam:CreateServiceLinkedRole",
                "sns:CreateTopic",
```

```
                "sns:Subscribe",
                "sns:Get*",
                "sns:List*"
            ],
            "Resource": ["*"]
        }
    ]
}
```

Amazon ECS 서비스 예제 생성 및 Amazon ECS 서비스 예제 업데이트 IAM 정책 예제에서는 AWS Management Console에서 서비스 Auto Scaling을 사용하는 데 필요한 권한을 보여줍니다.

Application Auto Scaling 서비스에는 Amazon ECS 서비스 및 CloudWatch 경보를 설명할 권한과 사용자 대신 원하는 서비스 수를 수정할 권한이 필요합니다. `sns:` 권한은 임계값이 초과되었을 때 CloudWatch가 Amazon SNS 주제로 보내는 알림에 대한 것입니다. Amazon ECS 서비스에 대해 Auto Scaling을 사용하면 `AWSServiceRoleForApplicationAutoScaling_ECSService`라는 서비스 연결 역할이 생성됩니다. 이 서비스 연결 역할은 정책에 대한 경보를 설명하고, 서비스의 현재 실행 중인 작업 수를 모니터링하고, 원하는 서비스 수를 수정할 수 있는 Application Auto Scaling 권한을 부여합니다. Application Auto Scaling의 관리형 Amazon ECS 역할은 원래 `ecsAutoscaleRole`이었지만 더 이상 필요하지 않습니다. 서비스 연결 역할은 Application Auto Scaling의 기본 역할입니다. 자세한 정보는 Application Auto Scaling 사용 설명서의 Application Auto Scaling 서비스 연결 역할을 참조하세요.

Amazon ECS에서 CloudWatch 지표를 활성화하기 전에 Amazon ECS 컨테이너 인스턴스 역할을 생성한 경우 `ecs:StartTelemetrySession` 권한을 추가해야 할 수 있습니다. 자세한 내용은 고려 사항 섹션을 참조하세요.

생성 시 리소스에 태그를 지정할 수 있는 권한 부여

다음과 같은 생성 시 태그 지정 Amazon ECS API 작업을 사용하면 리소스를 생성할 때 태그를 지정할 수 있습니다. 리소스 생성 작업에서 태그가 지정되면 AWS에서는 추가 권한 부여를 수행하여 태그를 생성할 올바른 권한이 할당되었는지 확인합니다.

- `CreateCapacityProvider`
- `CreateCluster`
- `CreateService`
- `CreateTaskSet`
- `RegisterContainerInstance`

- RegisterTaskDefinition
- RunTask
- StartTask

리소스 태그를 사용하여 속성 기반 제어(ABAC)를 구현할 수 있습니다. 자세한 내용은 the section called “리소스 태그를 사용하여 Amazon ECS 리소스에 대한 액세스 제어” 및 리소스에 태그 지정(을)를 참조하세요.

생성 시 태그 지정을 허용하려면 리소스를 생성하는 작업(예: ecs:CreateCluster 또는 ecs:RunTask)과 ecs:TagResource 작업을 사용할 수 있는 권한을 모두 포함하도록 정책을 만들거나 수정하세요.

다음 예제에서는 사용자가 클러스터를 생성하고 클러스터 생성 중에 태그를 추가할 수 있도록 허용하는 정책을 보여줍니다. 사용자는 기존 리소스에 태그를 지정할 수 없습니다(ecs:TagResource 작업을 직접 호출할 수 없습니다).

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
         "ecs:CreateCluster"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
         "ecs:TagResource"
      ],
      "Resource": "*",
      "Condition": {
         "StringEquals": {
                  "ecs:CreateAction": [
                      "CreateCluster",
                      "CreateCapacityProvider",
                      "CreateService",
                      "CreateTaskSet",
                      "RegisterContainerInstance",
                      "RegisterTaskDefinition",
                      "RunTask",
```

```
                    "StartTask"
                ]
            }
        }
    }
  ]
}
```

ecs:TagResource 작업은 리소스 생성 작업 도중 태그가 적용되는 경우에만 평가됩니다. 따라서 리소스를 생성할 권한이 있는 사용자(태그 지정 조건은 없다고 가정)는 요청에서 태그가 지정되지 않은 경우, ecs:TagResource 작업을 사용할 권한이 필요하지 않습니다. 하지만 사용자가 태그를 사용하여 리소스 생성을 시도하는 경우, 사용자에게 ecs:TagResource 작업을 사용할 권한이 없다면 요청은 실패합니다.

Amazon ECS는 특정 태그에 대한 액세스를 제어

IAM 정책의 Condition 요소에 추가 조건을 사용하여 리소스에 적용할 수 있는 태그 키와 값을 제어할 수 있습니다.

앞 섹션의 예제에 다음 조건 키를 사용할 수 있습니다.

- aws:RequestTag: 특정 태그 키 또는 태그 키 및 값이 요청에 존재해야 함을 표시. 요청에서 다른 태그도 지정할 수 있습니다.
  - 특정한 태그와 키 및 가치의 조합을 적용하려면(예를 들어 태그 StringEquals=cost-center:를 적용하려면) cc123 조건 연산자와 함께 사용하십시오.

    ```
    "StringEquals": { "aws:RequestTag/cost-center": "cc123" }
    ```

  - 요청에서 특정 태그 키를 적용하려면(예를 들어 태그 키 StringLike:를 적용하려면) purpose 조건 연산자와 함께 사용하십시오.

    ```
    "StringLike": { "aws:RequestTag/purpose": "*" }
    ```

- aws:TagKeys: 요청에서 사용되는 태그 키를 적용.
  - 요청 시 지정하려면 ForAllValues 변경자와 함께 특정 태그 키를 적용하십시오(요청에서 태그가 지정되면 특정 태그 키만 허용되고 다른 태그는 허용되지 않습니다). 예를 들어 태그 키 environment 또는 cost-center가 허용됩니다.

    ```
    "ForAllValues:StringEquals": { "aws:TagKeys": ["environment","cost-center"] }
    ```

- 요청에서 지정된 태그 키 중 최소한 1개의 존재를 적용하려면 `ForAnyValue` 변경자와 함께 사용하십시오. 예를 들어 요청:에 태그 키 `environment` 또는 `webserver` 중 최소한 1개가 존재해야 합니다.

```
"ForAnyValue:StringEquals": { "aws:TagKeys": ["environment","webserver"] }
```

이들 조건 키는 `ecs:TagResource` 작업뿐 아니라 태그 지정을 지원하는 리소스 생성 작업에 적용될 수 있습니다. Amazon ECS API 작업에서 태그 지정을 지원하는지 알아보려면 Amazon Elastic Container Service에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

사용자가 리소스를 생성할 때 강제로 태그를 지정하도록 하려면 리소스 생성 작업에서`aws:RequestTag` 조건 키 또는 `aws:TagKeys` 조건 키를 `ForAnyValue` 변경자와 함께 사용해야 합니다. 이때 사용자가 리소스 생성 시 태그를 지정하지 않으면 `ecs:TagResource` 작업이 평가되지 않습니다.

조건의 경우 조건 키는 대소문자를 구분하지 않고 조건 값은 대소문자를 구분합니다. 따라서, 태그 키의 대소문자 구별을 설정하려면 태그 키가 조건의 값으로 지정된 `aws:TagKeys` 조건 키를 사용합니다.

다중 값 조건에 대한 자세한 내용은 IAM 사용 설명서의 다수의 컨텍스트 키 또는 값을 사용하는 조건을 참조하세요.

리소스 태그를 사용하여 Amazon ECS 리소스에 대한 액세스 제어

사용자에게 Amazon ECS 리소스를 사용할 수 있는 권한을 부여하는 IAM 정책을 생성할 때 정책의 `Condition` 요소에 태그 정보를 포함하면 태그를 기반으로 액세스를 제어할 수 있습니다. 이를 속성 기반 액세스 제어(ABAC)라고 합니다. ABAC를 통해 사용자는 수정, 사용 또는 삭제할 수 있는 리소스를 더욱 정확하게 제어할 수 있습니다. 자세한 내용은 AWS용 ABAC란 무엇입니까? 단원을 참조하세요.

예를 들어 사용자가 클러스터를 삭제할 수 있도록 허용하지만 클러스터에 `environment=production` 태그가 있는 경우 작업을 거부하는 정책을 생성할 수 있습니다. 이렇게 하려면 `aws:ResourceTag` 조건 키를 사용하여 리소스에 연결된 태그를 기반으로 리소스에 대한 액세스를 허용하거나 거부합니다.

```
"StringEquals": { "aws:ResourceTag/environment": "production" }
```

Amazon ECS API 작업에서 `aws:ResourceTag` 조건 키를 사용한 액세스 제어를 지원하는지 알아보려면 Amazon Elastic Container Service에 사용되는 작업, 리소스 및 조건 키를 참조하세요. `Describe` 작업은 리소스 수준 권한을 지원하지 않기 때문에 조건 없이 별도의 명령문에 지정해야 합니다.

예제 IAM 정책은 Amazon ECS 예제 정책 섹션을 참조하세요.

태그를 기준으로 리소스에 대한 사용자 액세스를 허용 또는 거부하는 경우 동일한 리소스에서 태그를 추가 또는 제거할 수 있도록 사용자를 명시적으로 거부할 것을 고려해야 합니다. 그렇지 않으면 사용자가 제한을 피해 태그를 수정하여 리소스에 대한 액세스 권한을 얻을 수 있습니다.

Amazon ECS 예제 정책

IAM 정책을 사용하여 Amazon ECS 콘솔에서 특정 리소스를 보고 관련 작업을 수행할 권한을 사용자에게 부여할 수 있습니다. 이전 섹션의 예제 정책을 사용할 수 있지만 해당 정책은 AWS CLI 또는 AWS SDK를 통한 요청에 맞게 설계되었습니다.

예: 사용자가 태그를 기반으로 Amazon ECS 클러스터를 삭제하도록 허용

다음 정책은 태그의 키/값 쌍이 'Purpose/Testing'인 경우 사용자가 클러스터를 삭제할 수 있도록 허용합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecs:DeleteCluster"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:ecs:us-east-1:111122223333:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Purpose": "Testing"
                }
            }
        }
    ]
}
```

Amazon Elastic Container Service Identity and Access 문제 해결

다음 정보를 사용하여 Amazon ECS 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

주제

- Amazon ECS에서 작업을 수행할 권한이 없음
- iam:PassRole을 수행하도록 인증되지 않음
- 내 AWS 계정 외부의 사람이 내 Amazon ECS 리소스에 액세스할 수 있게 허용하고 싶음
- 추가 문제 해결 리소스

Amazon ECS에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 오류가 수신되면, 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음의 예제 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `ecs:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 ecs:GetWidget on resource: my-example-widget
```

이 경우, `ecs:`*`GetWidget`* 작업을 사용하여 *`my-example-widget`* 리소스에 액세스할 수 있도록 `mateojackson` 사용자 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Amazon ECS에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스에서는 새로운 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 해당 서비스에 기존 역할을 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 `marymajor`라는 IAM 사용자가 콘솔을 사용하여 Amazon ECS에서 태스크를 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 `iam:PassRole` 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 AWS 계정 외부의 사람이 내 Amazon ECS 리소스에 액세스할 수 있게 허용하고 싶음

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- Amazon ECS에서 이러한 기능을 지원하는지를 알아보려면 Amazon Elastic Container Service가 IAM과 작동하는 방식을 참조하세요.
- 소유하고 있는 AWS 계정의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 자신이 소유한 다른 AWS 계정의 IAM 사용자에 대한 액세스 권한 제공을 참조하세요.
- 리소스에 대한 액세스 권한을 서드 파티 AWS 계정에게 제공하는 방법을 알아보려면 IAM 사용 설명서의 서드 파티가 소유한 AWS 계정에 대한 액세스 제공을 참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

추가 문제 해결 리소스

다음 페이지에서는 오류 코드에 대한 정보를 제공합니다.

- Amazon ECS 중지된 작업 오류 메시지
- Amazon ECS 서비스 이벤트 메시지 보기

Amazon ECS에 대한 IAM 모범 사례

AWS Identity and Access Management(IAM)를 사용하면 인증 및 권한 부여 목적의 규칙 기반 정책을 통해 AWS 서비스와 리소스에 대한 액세스를 관리하고 제어할 수 있습니다. 더 구체적으로는 이 서비스를 통해 사용자, 그룹 또는 역할에 적용되는 정책을 사용하여 AWS 리소스에 대한 액세스를 제어할 수 있습니다. 이 세 가지 중 사용자는 리소스에 액세스할 수 있는 계정입니다. 그리고 IAM 역할은 IAM 외부의 특정 ID와 연결되지 않은 인증된 ID가 수임할 수 있는 권한 집합입니다. 자세한 내용은 Amazon ECS 액세스 관리 개요: 권한 및 정책 섹션을 참조하세요.

최소 권한 액세스 정책 따르기

사용자가 규정된 작업을 수행할 수 있도록 범위를 지정한 정책을 생성하세요. 예를 들어 개발자가 작업을 정기적으로 중지해야 하는 경우 해당하는 특정 작업만 허용하는 정책을 만드세요. 다음 예제에서는 사용자가 특정 Amazon 리소스 이름(ARN)을 갖는 클러스터의 특정 `task_family`에 속한 작업을 중지할 수만 있도록 허용합니다. 조건에서 ARN을 참조하는 것은 리소스 수준 권한을 사용하는 예이기도 합니다. 리소스 수준 권한을 사용하면 작업을 적용할 리소스를 지정할 수 있습니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:StopTask"
            ],
            "Condition": {
                "ArnEquals": {
                       "ecs:cluster": "arn:aws:ecs:us-
east-1:111122223333:cluster/cluster_name"
                }
            },
            "Resource": [
                   "arn:aws:ecs:us-east-1:111122223333:task-
definition/task_family:*"
            ]
        }
    ]
}
```

클러스터 리소스가 관리 경계 역할 수행

정책의 범위가 너무 좁으면 역할이 너무 많아지고 관리 오버헤드가 증가할 수 있습니다. 특정 작업이나 서비스로만 범위가 지정된 역할을 생성하는 대신 클러스터로 범위가 지정된 역할을 만들고 클러스터를 기본 관리 경계로 사용하세요.

API에서 최종 사용자를 격리하도록 자동화된 파이프라인 생성

자동으로 애플리케이션을 패키징하고 Amazon ECS 클러스터에 배포하는 파이프라인을 생성하여 사용자가 사용할 수 있는 작업을 제한할 수 있습니다. 이렇게 하면 작업을 생성, 업데이트 및 삭제하는 작업이 파이프라인에 실제로 위임됩니다. 자세한 내용은 사용 AWS CodePipeline 사용 설명서에서 Tutorial: Amazon ECS standard deployment with CodePipeline을 참조하세요.

보안 계층 추가를 위한 정책 조건 사용

추가 보안 계층이 필요한 경우 정책에 조건을 추가하세요. 이는 권한 있는 작업을 수행하는 경우나 특정 리소스에 대해 수행할 수 있는 작업 집합을 제한해야 하는 경우에 유용할 수 있습니다. 다음 예제 정책에서는 클러스터를 삭제할 때 멀티 팩터 권한 부여를 요구합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DeleteCluster"
      ],
      "Condition": {
        "Bool": {
          "aws:MultiFactorAuthPresent": "true"
        }
      },
    "Resource": ["*"]
    }
  ]
}
```

서비스에 적용된 태그는 해당 서비스에 속하는 모든 작업으로 전파됩니다. 따라서 특정 태그를 사용하여 Amazon ECS 리소스로 범위를 지정한 역할을 생성할 수 있습니다. 다음 정책에서는 IAM 보안 주체가 태그 키가 `Department`이고 태그 값이 `Accounting`인 모든 태스크를 시작하고 중지합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:StartTask",
                "ecs:StopTask",
                "ecs:RunTask"
            ],
            "Resource": "arn:aws:ecs:arn:aws::111122223333:task/my-cluster/*",
            "Condition": {
                "StringEquals": {"ecs:ResourceTag/Department": "Accounting"}
            }
        }
    ]
}
```

API에 대한 액세스를 주기적으로 감사

사용자가 역할을 변경할 수 있습니다. 역할을 변경하면 이전에 부여된 권한은 더 이상 적용되지 않을 수 있습니다. 누가 Amazon ECS API에 액세스할 수 있고, 해당 액세스가 여전히 적절한지 감사해야 합니다. 사용자가 조직을 떠나면 액세스 권한이 자동으로 취소되도록 IAM을 사용자 수명 주기 관리 솔루션과 통합하는 것이 좋습니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 AWS 보안 감사 지침을 참조하세요.

Amazon Elastic Container Service의 로깅 및 모니터링

모니터링은 Amazon Elastic Container Service와 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. 다중 지점 실패가 발생할 경우 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집해야 합니다. AWS는 Amazon ECS 리소스를 모니터링하고 잠재적 인시던트에 대응하기 위한 여러 도구를 제공합니다.

Amazon CloudWatch 경보

사용자가 지정한 기간 동안 단일 지표를 감시하고 여러 기간에 걸쳐 특정 임계값과 관련된 지표 값을 기준으로 하나 이상의 태스크를 수행합니다. 이 태스크는 Amazon Simple Notification Service(Amazon SNS) 주제 또는 Amazon EC2 Auto Scaling 정책에 전송되는 알림입니다. CloudWatch 경보는 특정 상태에 있다는 이유만으로는 작업을 호출하지 않습니다. 상태가 변경되고 지정한 기간 동안 유지되어야 합니다. 자세한 정보는 CloudWatch를 사용하여 Amazon ECS 모니터링 섹션을 참조하세요.

Fargate 시작 유형을 사용하는 태스크가 있는 서비스의 경우, CloudWatch 경보를 사용하면 CPU 및 메모리 사용률과 같은 CloudWatch 지표를 기반으로 하여 서비스의 태스크를 축소 및 확장할 수 있습니다. 자세한 정보는 Amazon ECS 서비스 자동 조정 섹션을 참조하세요.

EC2 시작 유형을 사용하는 태스크 또는 서비스가 있는 클러스터의 경우, CloudWatch 경보를 사용하면 클러스터 메모리 예약과 같은 CloudWatch 지표를 기반으로 컨테이너 인스턴스를 축소 및 확장할 수 있습니다.

Amazon CloudWatch Logs

태스크 정의에서 `awslogs` 로그 드라이버를 지정하여 Amazon ECS 태스크 내의 컨테이너에서 로그 파일을 모니터링, 저장 및 액세스합니다. 자세한 내용은 Using the awslogs driver를 참조하세요.

Amazon ECS 컨테이너 인스턴스에서 운영 체제 및 Amazon ECS 컨테이너 에이전트 로그 파일을 모니터링, 저장 및 액세스할 수도 있습니다. 이와 같은 로그 액세스 방법은 EC2 시작 유형을 사용하는 컨테이너에 사용할 수 있습니다.

Amazon CloudWatch Events

이벤트를 매칭하고 하나 이상의 대상 함수 또는 스트림으로 라우팅하여 값을 변경하고, 상태 정보를 캡처하고, 교정 조치를 취합니다. 자세한 내용은 이 가이드의 EventBridge를 사용하여 Amazon ECS 오류에 대한 응답 자동화 섹션과 Amazon EventBridge 사용 안내서의 EventBridge는 Amazon CloudWatch Events의 진화를 참조하세요.

AWS CloudTrail 로그

CloudTrail은 Amazon ECS에서 사용자, 역할 또는 AWS 서비스가 수행한 작업의 기록을 제공합니다. CloudTrail에서 수집한 정보를 사용하여 Amazon ECS에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다. 자세한 정보는 AWS CloudTrail을 사용하여 Amazon ECS API 직접 호출 로깅 섹션을 참조하세요.

AWS Trusted Advisor

Trusted Advisor는 수십만 명의 AWS 고객에게 서비스를 제공하면서 익힌 모범 사례를 활용합니다. Trusted Advisor는 AWS 환경을 검사한 후 비용 절감, 시스템 가용성 및 성능 향상 또는 보안 격차를 해결할 기회가 있을 때 권장 사항을 제시합니다. 모든 AWS 고객은 5개의 Trusted Advisor 점검 항목에 액세스할 수 있습니다. Business 또는 Enterprise Support 플랜을 보유한 고객은 모든 Trusted Advisor 점검 항목을 볼 수 있습니다.

자세한 정보는 지원 사용 설명서의 AWS Trusted Advisor 섹션을 참조하세요.

AWS Compute Optimizer

AWS Compute Optimizer는 AWS 리소스의 구성 및 사용률 지표를 분석하는 서비스입니다. 이는 리소스가 최적 상태인지 여부를 보고하고 최적화 권장 사항을 생성하여 비용을 절감하고 워크로드의 성능을 개선합니다.

자세한 내용은 Amazon ECS에 대한 AWS Compute Optimizer 권장 사항 섹션을 참조하세요.

Amazon ECS 모니터링의 또 한 가지 중요한 부분은 CloudWatch 경보에 포함되지 않는 항목을 수동으로 모니터링해야 한다는 점입니다. CloudWatch, Trusted Advisor 및 기타 AWS 콘솔 대시보드에서는 AWS 환경의 상태를 한 눈에 볼 수 있습니다. 또한 태스크 내 컨테이너 인스턴스와 컨테이너에서 로그 파일을 확인하는 것이 좋습니다.

Amazon Elastic Container Service에 대한 규정 준수 확인

AWS 서비스가 특정 규정 준수 프로그램의 범위에 포함되는지 알아보려면 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하고 관심 있는 규정 준수 프로그램을 선택하세요. 일반적인 정보는 AWS 규정 준수 프로그램을 참조하세요.

AWS Artifact(을)를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 AWS Artifact에서 보고서 다운로드를 참조하세요.

AWS 서비스 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다.AWS에서는 규정 준수를 지원할 다음과 같은 리소스를 제공합니다.

- 보안 규정 준수 및 거버넌스 - 이러한 솔루션 구현 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수 기능을 배포하는 단계를 제공합니다.
- HIPAA 적격 서비스 참조 - HIPAA 적격 서비스가 나열되어 있습니다. 모든 AWS 서비스에 HIPAA 자격이 있는 것은 아닙니다.

- AWS 규정 준수 리소스 - 고객 조직이 속한 산업 및 위치에 적용될 수 있는 워크북 및 가이드 컬렉션입니다.
- AWS 고객 규정 준수 가이드 - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드에서는 AWS 서비스를 보호하기 위한 모범 사례를 요약하고 여러 프레임워크(미국 표준 기술 연구소(NIST), 결제 카드 산업 보안 표준 위원회(PCI), 국제 표준화기구(ISO) 등)에서 보안 컨트롤에 대한 지침을 매핑합니다.
- AWS Config 개발자 가이드의 규칙을 사용하여 리소스 평가 – AWS Config 서비스는 내부 사례, 산업 지침 및 규제에 대한 리소스 구성의 준수 상태를 평가합니다.
- AWS Security Hub – 이 AWS 서비스(은)는 AWS 내 보안 상태에 대한 포괄적인 보기를 제공합니다. Security Hub는 보안 컨트롤을 사용하여 AWS리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하세요.
- Amazon GuardDuty - 이 AWS 서비스는 의심스럽고 악의적인 활동이 있는지 환경을 모니터링하여 AWS 계정, 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 탐지합니다. GuardDuty는 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 따르는 데 도움을 줄 수 있습니다.
- AWS Audit Manager - 이 AWS 서비스은(는) AWS 사용을 지속해서 감사하여 위험을 관리하고 규정 및 업계 표준을 준수하는 방법을 간소화할 수 있도록 지원합니다.

Amazon ECS에 대한 규정 준수 및 보안 모범 사례

Amazon ECS 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다.

PCI DSS(지불 카드 산업 데이터 보안 표준)

PCI DSS를 준수할 때는 환경 내 카드 소유자 데이터(CHD)의 전체 흐름을 이해하는 일이 중요합니다. CHD 흐름에 따라 PCI DSS의 적용 가능성이 결정되고, 카드 소유자 데이터 환경(CDE)의 경계와 구성 요소가 정의되므로, PCI DSS 평가의 범위도 정해집니다. PCI DSS 범위를 정확히 결정하는 일은 보안 태세를 정의하는 데 핵심이며, 궁극적으로 평가 성공으로 이어집니다. 고객은 범위의 완전성을 보장하고 범위에서 변경되거나 벗어나는 사항을 감지하는 범위 결정을 위한 절차를 마련해야 합니다.

컨테이너식 애플리케이션의 일시적인 특성으로 인해 구성을 감사할 때 복잡성이 추가됩니다. 따라서 고객은 컨테이너 수명 주기의 전체 단계에서 규정 준수 요구 사항이 충족되도록 모든 컨테이너 구성 파라미터를 지속적으로 파악해야 합니다.

Amazon ECS에서 PCI DSS 규정 준수를 달성하는 방법에 대한 자세한 내용은 다음 백서를 참조하세요.

- Architecting on Amazon ECS for PCI DSS compliance

미국 건강 보험 양도 및 책임에 관한 법(HIPAA)

보호 대상 건강 정보(PHI)를 처리하는 워크로드와 함께 Amazon ECS를 사용하면 추가 구성이 필요하지 않습니다. Amazon ECS는 Amazon EC2에서 컨테이너 시작을 조정하는 오케스트레이션 서비스 역할을 합니다. 오케스트레이션되는 워크로드 내의 데이터와 관련해서는 작동하지 않습니다. HIPAA 규정 및 AWS 비즈니스 관련자 부록에 따라 PHI는 Amazon ECS에서 시작된 컨테이너에서 액세스하는 경우 전송 중 및 저장 시 암호화해야 합니다.

Amazon S3, Amazon EBS 및 AWS KMS와 같은 각 AWS 스토리지 옵션과 함께 저장 시 암호화하기 위한 다양한 메커니즘을 사용할 수 있습니다. 오버레이 네트워크(예: VNS3 또는 Weave Net)를 배포하여 컨테이너 간에 전송되는 PHI를 완전히 암호화하거나 중복 암호화 계층을 제공할 수 있습니다. 또한 전체 로깅을 활성화하고 모든 컨테이너 로그를 Amazon CloudWatch로 전달해야 합니다. 인프라 보안에 대한 모범 사례를 사용하는 방법을 자세히 알아보려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS Security Hub

AWS Security Hub를 사용합니다. 이 AWS 서비스는 AWS 내 보안 상태에 대한 포괄적인 보기를 제공합니다. Security Hub는 보안 컨트롤을 사용하여 AWS리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하세요.

Amazon ECS Runtime Monitoring이 포함된 Amazon GuardDuty

Amazon GuardDuty는 AWS 환경 내 계정, 컨테이너, 워크로드 및 데이터를 보호하는 데 도움이 되는 위협 감지 서비스입니다. GuardDuty는 기계 학습(ML) 모델, 이상 및 위협 감지 기능을 통해 다양한 로그 소스와 런타임 활동을 지속적으로 모니터링하여 사용자 환경의 잠재적 보안 위험과 악의적 활동을 식별하고 우선순위를 지정합니다.

GuardDuty의 Runtime Monitoring을 사용하여 악의적 또는 무단 동작을 식별합니다. Runtime Monitoring은 AWS 로그 및 네트워킹 활동을 지속적으로 모니터링하여 악의적 동작 또는 무단 동작을 식별함으로써 Fargate 및 EC2에서 실행되는 워크로드를 보호합니다. Runtime Monitoring은 파일 액세스, 프로세스 실행 및 네트워크 연결과 같은 호스트 내 동작을 분석하는 경량의 완전관리형 GuardDuty 보안 에이전트를 사용합니다. 여기에는 권한 에스컬레이션, 노출된 자격 증명 사용 또는 악의적인 IP 주소 및 도메인과의 통신, 그리고 Amazon EC2 인스턴스 및 컨테이너 워크로드에 존재하는 맬웨어 같은 문제가 포함됩니다. 자세한 내용은 GuardDuty 사용 설명서의 GuardDuty Runtime Monitoring을 참조하세요.

규정 준수 권장 사항

규정 준수 프로그램 소유자를 조기에 비즈니스에 참여시키고 AWS 공동 책임 모델을 사용하여 관련 규정 준수 프로그램의 성공을 위해 규정 준수 제어 소유권을 식별하는 것이 좋습니다. 자세한 내용은 Amazon ECS에 대한 AWS 공동 책임 모델 섹션을 참조하세요.

AWS Fargate Federal Information Processing Standard(FIPS-140)

FIPS(Federal Information Processing Standard). FIPS-140은 미국 및 캐나다 정부 표준으로서, 기밀 정보를 보호하는 암호화 모듈의 보안 요건을 규정하고 있습니다. FIPS-140은 전송 중 데이터와 저장 데이터를 암호화하는 데 사용할 수 있는 일련의 검증된 암호화 기능을 정의합니다.

FIPS-140 규정 준수를 켜면 Fargate에서 FIPS-140 규정을 준수하는 방식으로 워크로드를 실행할 수 있습니다. FIPS-140 규정 준수에 대한 자세한 내용은 FIPS(Federal Information Processing Standard) 140-3을 참조하세요.

AWS Fargate FIPS-140 고려 사항

Fargate에서 FIPS-140 규정 준수를 사용할 때는 다음 사항을 고려하세요.

- FIPS-140 규정 준수는 AWS GovCloud (US) 리전에서만 사용할 수 있습니다.
- Fargate는 FIPS-140 버전 140.3을 지원합니다.
- FIPS-140 규정 준수는 기본적으로 꺼져 있습니다. 켜야 합니다.
- FIPS-140 규정 준수를 위해 작업에서 다음 구성을 사용해야 합니다.
  - `operatingSystemFamily`이 `LINUX`이어야 합니다.
  - `cpuArchitecture`이 `X86_64`이어야 합니다.
  - Fargate 플랫폼 버전은 `1.4.0` 이상이어야 합니다.

Fargate에서 FIPS 사용

Fargate에서 FIPS-140 규정 준수를 사용하려면 다음 절차를 따르세요.

1. FIPS-140 규정 준수를 켭니다. 자세한 내용은 the section called “AWS Fargate Federal Information Processing Standard(FIPS-140) 규정 준수” 섹션을 참조하세요.
2. 필요할 경우 ECS Exec을 통해 다음 명령을 실행하여 클러스터의 FIPS-140 규정 준수 상태를 확인할 수 있습니다.

*cluster-name*을 클러스터 이름으로 바꾸고, *task-id*를 태스크의 ID 또는 ARN으로 바꾸고, *container-name*을 명령을 실행하려는 태스크의 컨테이너 이름으로 바꿉니다.

반환 값이 '1'이면 FIPS를 사용하고 있음을 나타냅니다.

```
aws ecs execute-command \
    --cluster cluster-name \
    --task task-id \
    --container container-name \
    --interactive \
    --command "cat /proc/sys/crypto/fips_enabled"
```

Fargate FIPS-140 감사에 CloudTrail 사용

CloudTrail은 계정 생성 시 AWS 계정에서 켜집니다. Amazon ECS에서 API 및 콘솔 활동이 수행되면 해당 활동은 이벤트 기록에서 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 설명은 CloudTrail 이벤트 기록으로 이벤트 보기를 참조하세요.

Amazon ECS에 대한 이벤트를 포함하여 AWS 계정의 이벤트에 대한 지속적인 기록을 보려면 CloudTrail이 로그 파일을 Amazon S3 버킷으로 전송하는 데 사용하는 추적을 생성하세요. 콘솔에서 트레일을 생성하면 기본적으로 모든 리전에 트레일이 적용됩니다. 추적은 AWS 파티션에 있는 모든 지역의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 작업하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 the section called "AWS CloudTrail을 사용하여 Amazon ECS API 직접 호출 로깅" 섹션을 참조하세요.

다음 예제는 `PutAccountSettingDefault` API 작업을 보여주는 CloudTrail 로그 항목을 나타냅니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAIV5AJI5LXF5EXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/jdoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIPWIOFC3EXAMPLE",
    },
```

```
    "eventTime": "2023-03-01T21:45:18Z",
    "eventSource": "ecs.amazonaws.com",
    "eventName": "PutAccountSettingDefault",
    "awsRegion": "us-gov-east-1",
    "sourceIPAddress": "52.94.133.131",
    "userAgent": "aws-cli/2.9.8 Python/3.9.11 Windows/10 exe/AMD64 prompt/off command/
ecs.put-account-setting",
    "requestParameters": {
        "name": "fargateFIPSMode",
        "value": "enabled"
    },
    "responseElements": {
        "setting": {
            "name": "fargateFIPSMode",
            "value": "enabled",
            "principalArn": "arn:aws:iam::123456789012:user/jdoe"
        }
    },
    "requestID": "acdc731e-e506-447c-965d-f5f75EXAMPLE",
    "eventID": "6afced68-75cd-4d44-8076-0beEXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "ecs-fips.us-gov-east-1.amazonaws.com"
    }
}
```


Amazon Elastic Container Service의 인프라 보안

관리형 서비스인 Amazon Elastic Container Service는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 Amazon ECS에 액세스합니다. 고객은 다음을 지원해야 합니다.

- Transport Layer Security(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 위탁자와 관련된 보안 암호 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 자격 증명을 생성하여 요청에 서명할 수 있습니다.

어떤 네트워크 위치에서든 이러한 API 작업을 호출할 수 있습니다. Amazon ECS는 소스 IP 주소를 기반으로 한 제한을 포함할 수 있는 리소스 기반 액세스 정책을 지원하므로 정책이 네트워크 위치의 IP 주소를 설명하는지 확인합니다. Amazon ECS 정책을 사용하여 특정 Amazon Virtual Private Cloud 엔드포인트 또는 특정 VPC에서 액세스를 제어할 수도 있습니다. 그러면 AWS 네트워크 내의 특정 VPC에서만 특정 Amazon ECS 리소스에 대한 네트워크 액세스가 효과적으로 격리됩니다. 자세한 정보는 Amazon ECS 및 인터페이스 VPC 엔드포인트(AWS PrivateLink) 섹션을 참조하세요.

Amazon ECS 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)

인터페이스 VPC 엔드포인트를 사용하도록 Amazon ECS를 구성하여 VPC의 보안 상태를 향상시킬 수 있습니다. 인터페이스 엔드포인트는 프라이빗 IP 주소를 사용하여 비공개로 Amazon ECS API에 액세스할 수 있도록 허용하는 AWS PrivateLink 기술로 구동됩니다. AWS PrivateLink는 VPC와 서비스 간의 모든 네트워크 트래픽을 Amazon ECS 네트워크로 제한합니다. 인터넷 게이트웨이, NAT 디바이스 또는 가상 프라이빗 게이트웨이가 필요 없습니다.

AWS PrivateLink 및 VPC 엔드포인트에 대한 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 참조하세요.

고려 사항

2023년 12월 23일부터 도입된 리전의 엔드포인트에 대한 고려 사항

Amazon ECS용 인터페이스 VPC 엔드포인트를 설정하기 전에 다음 사항을 고려하세요.

- 다음과 같은 리전별 VPC 엔드포인트가 있어야 합니다.

> **Note**
>
> 모든 엔드포인트를 구성하지 않으면 트래픽은 VPC 엔드포인트가 아닌 퍼블릭 엔드포인트를 통해 전달됩니다.

- com.amazonaws.*region*.ecs-agent
- com.amazonaws.*region*.ecs-telemetry
- com.amazonaws.*region*.ecs

예를 들어, 캐나다 서부(캘거리)(ca-west-1) 리전에는 다음 VPC 엔드포인트가 있어야 합니다.

- com.amazonaws.ca-west-1.ecs-agent
- com.amazonaws.ca-west-1.ecs-telemetry
- com.amazonaws.ca-west-1.ecs

- 템플릿을 사용하여 새 리전에서 AWS 리소스를 생성하고 2023년 12월 23일 이전에 도입된 리전에서 템플릿을 복사한 경우 복사 원본 리전에 따라 다음 작업 중 하나를 수행합니다.

  예를 들어, 복사 원본 리전이 미국 동부(버지니아 북부)(us-east-1)입니다. 복사 대상 리전은 캐나다 서부(캘거리)(ca-west-1)입니다.

구성	작업
복사 원본 리전에 VPC 엔드포인트가 없습니다.	새 리전에 대해 모두 3개의 VPC 엔드포인트를 생성합니다(예: com.amazonaws.ca-west-1.ecs-agent).
복사 원본 리전에 리전별 VPC 엔드포인트가 포함되어 있습니다.	a. 새 리전에 대해 모두 3개의 VPC 엔드포인트를 생성합니다(예: com.amazonaws.ca-west-1.ecs-agent). b. 복사 원본 리전에 대해 모두 3개의 VPC 엔드포인트를 삭제합니다(예: com.amazonaws.us-east-1.ecs-agent).

Fargate 시작 유형을 위한 Amazon ECS VPC 엔드포인트에 대한 고려 사항

Fargate 작업이 배포되는 동일한 VPC에서 `ecr.dkr` 및 `ecr.api`에 대한 VPC 엔드포인트가 있는 경우 해당 VPC 엔드포인트를 사용합니다. VPC 엔드포인트가 없는 경우 Fargate 인터페이스를 사용합니다.

Amazon ECS용 인터페이스 VPC 엔드포인트를 설정하기 전에 다음 사항을 고려하세요.

- Fargate 시작 유형을 사용하는 태스크는 Amazon ECS용 인터페이스 VPC 엔드포인트가 필요 없지만 다음 포인트에서 설명하는 Amazon ECR, Secrets Manager 또는 Amazon CloudWatch Logs용 인터페이스 VPC 엔드포인트는 필요할 수 있습니다.
  - 태스크에서 Amazon ECR의 프라이빗 이미지를 가져오도록 허용하려면 Amazon ECR용 인터페이스 VPC 엔드포인트를 생성해야 합니다. 자세한 정보는 Amazon Elastic Container Registry 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

    > **Important**
    >
    > 인터페이스 VPC 엔드포인트를 사용하도록 Amazon ECR을 구성하는 경우 특정 VPC 또는 VPC 엔드포인트에 대한 액세스를 제한하는 조건 키를 포함하는 태스크 실행 역할을 생성할 수 있습니다. 자세한 정보는 인터페이스 엔드포인트 권한을 통해 Amazon ECR 이미지를 가져오는 Fargate 작업 섹션을 참조하세요.

  - 태스크에서 Secrets Manager에서 민감한 데이터를 가져오도록 허용하려면 Secrets Manager용 인터페이스 VPC 엔드포인트를 생성해야 합니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 VPC 엔드포인트와 함께 Secrets Manager 사용을 참조하세요.
  - VPC에 인터넷 게이트웨이가 없고 태스크에서 `awslogs` 로그 드라이버를 사용하여 로그 정보를 CloudWatch Logs로 전송하는 경우에는 CloudWatch Logs용 인터페이스 VPC 엔드포인트를 생성해야 합니다. 자세한 정보는 Amazon CloudWatch Logs 사용 설명서의 인터페이스 VPC 엔드포인트에서 CloudWatch Logs 사용을 참조하세요.
- VPC 엔드포인트는 교차 리전 요청을 현재 지원하지 않습니다. API 호출을 Amazon ECS로 발행할 계획인 동일 리전에서 엔드포인트를 생성해야 합니다. 예를 들어 미국 동부(버지니아 북부)에서 작업을 실행하려 한다고 가정해 보겠습니다. 그런 다음 미국 동부(버지니아 북부)에 Amazon ECS VPC 엔드포인트를 생성해야 합니다. 다른 리전에서 생성된 Amazon ECS VPC 엔드포인트는 미국 동부(버지니아 북부)에서 작업을 실행할 수 없습니다.
- VPC 엔드포인트는 Amazon Route 53을 통해 Amazon이 제공하는 DNS만 지원합니다. 자신의 DNS를 사용하는 경우에는 조건적인 DNS 전송을 사용할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 DHCP 옵션 세트를 참조하십시오.

- VPC 엔드포인트에 연결된 보안 그룹은 VPC의 프라이빗 서브넷에서 443 TCP 포트로 들어오는 연결을 허용해야 합니다.
- Envoy 프록시의 서비스 연결 관리에서는 VPC `com.amazonaws.region.ecs-agent` 엔드포인트를 사용합니다. VPC 엔드포인트를 사용하지 않는 경우 Envoy 프록시의 Service Connect 관리에서는 해당 리전의 `ecs-sc` 엔드포인트를 사용합니다. 각 리전의 Amazon ECS 엔드포인트 목록은 Amazon ECS endpoints and quotas를 참조하세요.

EC2 시작 유형을 위한 Amazon ECS VPC 엔드포인트에 대한 고려 사항

Amazon ECS용 인터페이스 VPC 엔드포인트를 설정하기 전에 다음 사항을 고려하세요.

- EC2 시작 유형을 사용하는 태스크는 시작되는 컨테이너 인스턴스가 Amazon ECS 컨테이너 에이전트의 `1.25.1` 이상 버전으로 실행되어야 합니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 관리 섹션을 참조하세요.
- 태스크에서 Secrets Manager에서 민감한 데이터를 가져오도록 허용하려면 Secrets Manager용 인터페이스 VPC 엔드포인트를 생성해야 합니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 VPC 엔드포인트와 함께 Secrets Manager 사용을 참조하세요.
- VPC에 인터넷 게이트웨이가 없고 태스크에서 `awslogs` 로그 드라이버를 사용하여 로그 정보를 CloudWatch Logs로 전송하는 경우에는 CloudWatch Logs용 인터페이스 VPC 엔드포인트를 생성해야 합니다. 자세한 정보는 Amazon CloudWatch Logs 사용 설명서의 인터페이스 VPC 엔드포인트에서 CloudWatch Logs 사용을 참조하세요.
- VPC 엔드포인트는 교차 리전 요청을 현재 지원하지 않습니다. API 호출을 Amazon ECS로 발행할 계획인 동일 리전에서 엔드포인트를 생성해야 합니다. 예를 들어 미국 동부(버지니아 북부)에서 작업을 실행하려 한다고 가정해 보겠습니다. 그런 다음 미국 동부(버지니아 북부)에 Amazon ECS VPC 엔드포인트를 생성해야 합니다. 다른 리전에서 생성된 Amazon ECS VPC 엔드포인트는 미국 동부(버지니아 북부)에서 작업을 실행할 수 없습니다.
- VPC 엔드포인트는 Amazon Route 53을 통해 Amazon이 제공하는 DNS만 지원합니다. 자신의 DNS를 사용하는 경우에는 조건적인 DNS 전송을 사용할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 DHCP 옵션 세트를 참조하십시오.
- VPC 엔드포인트에 연결된 보안 그룹은 VPC의 프라이빗 서브넷에서 443 TCP 포트로 들어오는 연결을 허용해야 합니다.

Amazon ECS 엔드포인트 이름 지정 패턴 이해

Amazon ECS 에이전트는 다음과 같이 번호가 매겨진 접미사를 사용하여 엔드포인트에 요청할 수 있다는 점을 이해하는 것이 중요합니다.

- 에이전트 엔드포인트의 경우 `ecs-a-1.region.amazonaws.com`, `ecs-a-2.region.amazonaws.com` 등
- 원격 분석 엔드포인트의 경우 `ecs-t-1.region.amazonaws.com`, `ecs-t-2.region.amazonaws.com` 등

이 동작은 Amazon ECS 에이전트가 DiscoverPollEndpoint API를 사용하여 연결할 특정 엔드포인트를 동적으로 결정하기 때문에 발생합니다. VPC 엔드포인트가 번호가 매겨진 이러한 엔드포인트 변형을 제대로 처리하지 못하면 기본 이름에 대해 VPC 엔드포인트를 구성했더라도 에이전트는 퍼블릭 엔드포인트 사용으로 돌아갑니다.

DiscoverPollEndpoint API의 역할

DiscoverPollEndpoint API는 Amazon ECS 에이전트가 작업을 폴링할 적절한 엔드포인트를 검색하는 데 사용됩니다. 에이전트가 이 API를 직접적으로 호출하면 번호가 매겨진 접미사가 포함될 수 있는 특정 엔드포인트 URL을 수신합니다. VPC 엔드포인트가 제대로 작동하려면 네트워크 구성에서 에이전트가 다음을 수행하도록 허용해야 합니다.

1. DiscoverPollEndpoint API에 액세스
2. 번호가 매겨진 접미사가 있는 엔드포인트 URL을 포함하여 반환된 엔드포인트 URL에 연결합니다.

VPC 엔드포인트 연결 문제를 해결하는 경우 에이전트가 기본 엔드포인트와 DiscoverPollEndpoint API에서 반환할 수 있는 번호가 매겨진 변형 모두에 연결할 수 있는지 확인하세요.

Amazon ECS용 VPC 엔드포인트 생성

Amazon ECS 서비스에 대한 VPC 엔드포인트를 생성하려면 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 VPC 엔드포인트를 사용하여 AWS 서비스 액세스 절차를 사용하여 다음 엔드포인트를 생성합니다. VPC 내에 기존 컨테이너 인스턴스가 있는 경우에는 표시 순서대로 엔드포인트를 생성해야 합니다. VPC 엔드포인트 생성 후에 컨테이너 인스턴스를 생성할 경우에는 순서가 중요하지 않습니다.

> **Note**
>
> 모든 엔드포인트를 구성하지 않으면 트래픽은 VPC 엔드포인트가 아닌 퍼블릭 엔드포인트를 통해 전달됩니다.
>
> 엔드포인트를 생성할 때 Amazon ECS는 엔드포인트에 대한 프라이빗 DNS 이름도 생성합니다. 예를 들어 ecs-agent의 경우 `ecs-a.region.amazonaws.com`, ecs-telemetry의 경우 `ecs-t.region.amazonaws.com`입니다.

- com.amazonaws.*region*.ecs-agent
- com.amazonaws.*region*.ecs-telemetry
- com.amazonaws.*region*.ecs

> **Note**
>
> *##*은 미국 동부(오하이오) 리전의 us-east-2 같이 Amazon ECS가 지원하는 AWS 리전의 리전 식별자를 나타냅니다.

ecs-agent 엔드포인트는 ecs:poll API를 사용하고, ecs-telemetry 엔드포인트는 ecs:poll 및 ecs:StartTelemetrySession API를 사용합니다.

EC2 시작 유형을 사용하는 기존 태스크가 있는 경우에는 VPC 엔드포인트를 생성한 후 각 컨테이너 인스턴스에서 새 구성을 선택해야 합니다. 이를 위해서는 각 컨테이너 인스턴스를 재부팅하거나 각 컨테이너 인스턴스에서 Amazon ECS 컨테이너 에이전트를 다시 시작해야 합니다. 다음과 같이 컨테이너 에이전트를 다시 시작합니다.

Amazon ECS 컨테이너 에이전트 재시작

1. SSH를 통해 컨테이너 인스턴스에 로그인합니다.
2. 컨테이너 에이전트를 중지합니다.

```
sudo docker stop ecs-agent
```

3. 컨테이너 에이전트를 시작합니다.

```
sudo docker start ecs-agent
```

VPC 엔드포인트를 생성하고 각 컨테이너 인스턴스에서 Amazon ECS 컨테이너 에이전트를 다시 시작하면 새로 시작된 모든 태스크에 새 구성이 적용됩니다.

Amazon ECS에 대한 VPC 엔드포인트 정책 생성

Amazon ECS에 대한 액세스를 제어하는 VPC 엔드포인트에 엔드포인트 정책을 연결할 수 있습니다. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 위탁자.

- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 정보는 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하세요.

예제: Amazon ECS 작업에 대한 VPC 엔드포인트 정책

다음은 Amazon ECS에 대한 엔드포인트 정책의 예입니다. 엔드포인트에 연결되면 이 정책은 클러스터를 생성하고 나열할 수 있는 권한을 부여합니다. `CreateCluster` 및 `ListClusters` 작업은 리소스를 허용하지 않으므로 리소스 정의는 모든 리소스에 대해 *로 설정됩니다.

```
{
   "Statement":[
    {
      "Principal":"*",
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:ListClusters"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Amazon ECS에 대한 AWS 공동 책임 모델

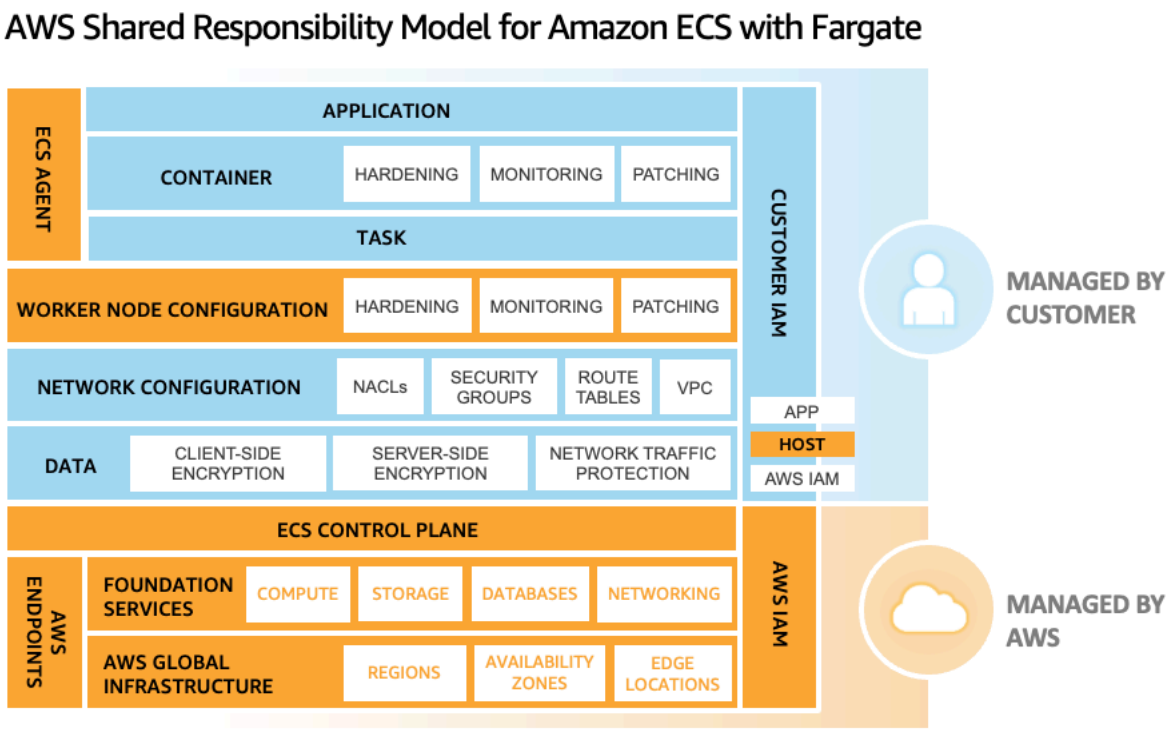
보안과 규정 준수는 AWS와 고객의 공동 책임입니다. 이 공동 모델은 고객의 운영 부담을 더는 데 도움이 될 수 있습니다. 호스트 운영 체제 및 가상화 계층부터 서비스 운영 시설의 물리적 보안에 이르는 구성 요소를 AWS에서 운영, 관리 및 제어하기 때문입니다. 고객의 책임 및 관리 범위에는 AWS가 제공하는 보안 그룹 방화벽의 구성과 게스트 운영 체제(업데이트 및 보안 패치 포함), 기타 관련 애플리케이션 소프트웨어가 포함됩니다. 사용하는 서비스, 서비스를 IT 환경에 통합하는 과정 및 준거법과 규제에 따라 책임 범위가 다르기 때문에 고객은 선택하고자 하는 서비스에 대해 신중해야 합니다. 또한 이러한 공동 책임은 본질적으로 유연성을 제공하며 배포를 허용하는 제어 권한을 고객에게 부여합니다.

Fargate 시작 유형

다음 그림은 Fargate 시작 유형에 대한 공동 책임 모델을 보여줍니다. Fargate는 격리된 하드웨어 가상화 환경에서 각 작업을 실행합니다. 따라서 각 태스크는 전용 인프라 용량을 얻습니다. Fargate에서 실행되는 컨테이너화된 워크로드는 운영 체제, Linux 커널, 네트워크 인터페이스, 임시 스토리지, CPU 또는 메모리를 다른 작업과 공유하지 않습니다. Fargate를 사용하는 경우 고객은 컨테이너를 실행하는 컴퓨팅 인프라를 보호할 책임이 없습니다. Fargate는 고객 워크로드가 실행되는 인프라를 프로비저닝하고 패치합니다. 자세한 내용은 Amazon ECS에서 AWS Fargate에 대한 태스크 사용 중지 및 유지 관리 섹션을 참조하세요.

사용자는 다음 리소스를 관리할 책임이 있습니다.

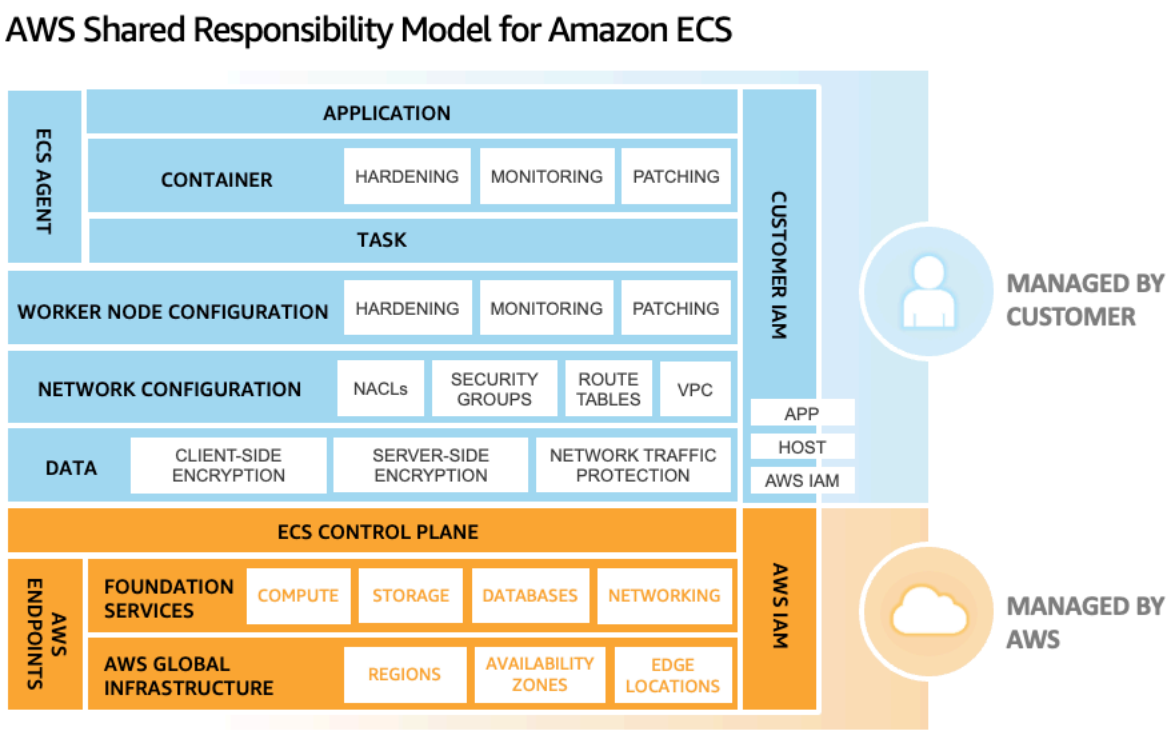
- VPC, NACL, 보안 그룹 및 라우팅 테이블을 포함한 네트워크 구성
- 클라이언트 및 서비스 스토리지 암호화. 자세한 내용은 Amazon ECS 작업에 대한 스토리지 옵션 섹션을 참조하세요.
- 컨테이너 이미지. 자세한 내용은 Amazon ECS 태스크 및 컨테이너 보안 모범 사례 섹션을 참조하세요.
- 태스크 역할을 사용하여 애플리케이션에 대한 IAM 권한입니다. 자세한 내용은 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.



EC2 시작 유형

다음 그림은 EC2 시작 유형에 대한 공동 책임을 보여줍니다. EC2 인스턴스에서 작업을 실행할 때 다음 리소스 외에도 EC2 인스턴스를 유지 관리할 책임이 있습니다.

- Amazon ECS 에이전트입니다.
- • 패치 적용 및 강화를 포함한 EC2 인스턴스 AMI.
- VPC, NACL, 보안 그룹 및 라우팅 테이블을 포함한 네트워크 구성입니다.
- 클라이언트 및 서비스 스토리지 암호화. 자세한 내용은 Amazon ECS 작업에 대한 스토리지 옵션 섹션을 참조하세요.
- 컨테이너 이미지. 자세한 내용은 Amazon ECS 태스크 및 컨테이너 보안 모범 사례 섹션을 참조하세요.
- 태스크 역할을 사용하여 애플리케이션에 대한 IAM 권한입니다. 자세한 내용은 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.



Amazon ECS에 대한 네트워크 보안 모범 사례

네트워크 보안은 여러 하위 항목을 포함하는 광범위한 항목입니다. 여기에는 전송 중 암호화, 네트워크 분할 및 격리, 방화벽, 트래픽 라우팅, 가시성 등이 포함됩니다.

전송 중 암호화

네트워크 트래픽을 암호화하면 데이터가 네트워크에서 전송될 때 권한이 없는 사용자가 데이터를 가로채거나 읽지 못하도록 할 수 있습니다. Amazon ECS를 사용하면 네트워크 암호화를 다음 방법 중 하나로 구현할 수 있습니다.

- Nitro 인스턴스 사용:

  기본적으로 C5n, G4, I3en, M5dn, M5n, P3dn, R5dn, R5n 등과 같은 Nitro 인스턴스 유형 간의 트래픽은 자동으로 암호화됩니다. 트래픽이 전송 게이트웨이, 로드 밸런서 또는 유사한 중간 서비스를 통해 라우팅되는 경우에는 트래픽이 암호화되지 않습니다.

  - 전송 중 데이터 암호화
  - 2019년의 새로운 발표
  - re:Force 2019의 강연

- Application Load Balancer와 함께 SNI(Server Name Indication) 사용:

  Application Load Balancer(ALB) 및 Network Load Balancer(NLB)는 Server Name Indication(SNI)을 지원합니다. SNI를 사용하면 단일 리스너 뒤에 여러 보안 애플리케이션을 배치할 수 있습니다. 이 경우 리스너마다 고유한 TLS 인증서가 있습니다. AWS Certificate Manager(ACM)를 사용하여 로드 밸런서에 대한 인증서를 프로비저닝한 다음 리스너의 인증서 목록에 추가하는 것이 좋습니다. AWS 로드 밸런서는 SNI를 지원하는 스마트 인증서 선택 알고리즘을 사용합니다. 클라이언트가 제공한 호스트 이름이 인증서 목록의 단일 인증서와 일치하면 로드 밸런서는 이 인증서를 선택합니다. 클라이언트가 제공한 호스트 이름이 인증서 목록의 여러 인증서와 일치하면 로드 밸런서는 클라이언트가 지원할 수 있는 인증서를 선택합니다. 예를 들어 자체 서명된 인증서 또는 ACM을 통해 생성된 인증서가 있습니다.

  - Application Load Balancer를 사용하는 SNI
  - Network Load Balancer를 사용하는 SNI

- TLS 인증서를 사용한 엔드 투 엔드 암호화:

  여기에는 작업에 TLS 인증서를 배포하는 작업이 포함됩니다. 이 인증서는 자체 서명된 인증서나 신뢰할 수 있는 인증 기관의 인증서일 수 있습니다. 인증서의 보안 암호를 참조하여 인증서를 받을 수 있습니다. 그렇지 않으면 ACM에 인증서 서명 요청(CSR)을 발행한 다음 결과로 얻은 보안 암호를 공유 볼륨에 탑재하는 컨테이너를 실행할 수도 있습니다.

  - Maintaining transport layer security all the way to your containers using the Network Load Balancer with Amazon ECS part 1

- Maintaining Transport Layer Security (TLS) all the way to your container part 2: Using AWS Private Certificate Authority

태스크 네트워킹

다음은 Amazon ECS 작동 방식을 고려하여 권장하는 사항입니다. Amazon ECS는 오버레이 네트워크를 사용하지 않습니다. 대신 다양한 네트워크 모드에서 작동하도록 작업을 구성합니다. 예를 들어 `bridge` 모드를 사용하도록 구성된 작업은 각 호스트에서 실행되는 Docker 네트워크에서 라우팅할 수 없는 IP 주소를 가져옵니다. `awsvpc` 네트워크 모드를 사용하도록 구성된 작업은 호스트의 서브넷에서 IP 주소를 가져옵니다. `host` 네트워킹으로 구성된 작업은 호스트의 네트워크 인터페이스를 사용합니다. `awsvpc`가 기본 네트워크 모드입니다. 이 모드에서만 작업에 보안 그룹을 할당할 수 있기 때문입니다. 또한 Amazon ECS에서 AWS Fargate 작업에 사용할 수 있는 유일한 모드이기도 합니다.

작업의 보안 그룹

`awsvpc` 네트워크 모드를 사용하도록 작업을 구성하는 것이 좋습니다. 이 모드를 사용하도록 작업을 구성하면 Amazon ECS 에이전트는 자동으로 탄력적 네트워크 인터페이스(ENI)를 프로비저닝하고 작업에 연결합니다. ENI가 프로비저닝되면 작업이 AWS 보안 그룹에 등록됩니다. 보안 그룹은 인바운드 및 아웃바운드 트래픽을 제어하는 데 사용할 수 있는 가상 방화벽 역할을 합니다.

태스크 또는 서비스와 함께 사용자 지정 방화벽을 사용하는 경우 Amazon ECS 에이전트 관리 엔드포인트('`ecs-a-*.region.amazonaws.com`'), 원격 측정 엔드포인트('`ecs-t-*.region.amazonaws.com`'), Service Connect Envoy 관리 엔드포인트('`ecs-sc.region.api.aws`')에 대한 트래픽을 허용하는 아웃바운드 규칙을 추가합니다.

AWS PrivateLink 및 Amazon ECS

AWS PrivateLink는 Amazon ECS를 비롯한 다양한 AWS 서비스를 위한 프라이빗 엔드포인트를 생성할 수 있는 네트워킹 기술입니다. 엔드포인트는 Amazon VPC에 연결된 인터넷 게이트웨이(IGW)가 없고 인터넷에 대한 대체 경로가 없는 샌드박스 환경에서 필요합니다. AWS PrivateLink를 사용하면 Amazon ECS 서비스에 대한 호출이 Amazon VPC 내에서 유지되고 인터넷을 통과하지 않습니다. Amazon ECS 및 기타 관련 서비스를 위한 AWS PrivateLink 엔드포인트를 생성하는 방법에 대한 지침은 Amazon ECS interface Amazon VPC endpoints를 참조하세요.

> **Important**
>
> AWS Fargate 태스크에는 Amazon ECS에 대한 AWS PrivateLink 엔드포인트가 필요하지 않습니다.

Amazon ECR 및 Amazon ECS 모두 엔드포인트 정책을 지원합니다. 이러한 정책을 통해 서비스 API에 대한 액세스를 구체화할 수 있습니다. 예를 들어, 이미지를 특정 AWS 계정의 레지스트리로만 푸시하도록 허용하는 Amazon ECR에 대한 엔드포인트 정책을 생성할 수 있습니다. 이와 같은 정책을 사용하면 데이터가 컨테이너 이미지를 통해 유출되는 것을 방지하면서도 사용자가 승인된 Amazon ECR 레지스트리로 푸시하도록 허용할 할 수 있습니다. 자세한 정보는 VPC 엔드포인트 정책 사용을 참조하세요.

다음 정책은 계정의 모든 AWS 보안 주체가 Amazon ECR 리포지토리에서만 모든 작업을 수행할 수 있도록 허용합니다.

```
{
  "Statement": [
    {
      "Sid": "LimitECRAccess",
      "Principal": "*",
      "Action": "*",
      "Effect": "Allow",
      "Resource": "arn:aws:ecr:region:account_id:repository/*"
    },
  ]
}
```

새 PrincipalOrgID 속성을 사용하는 조건을 설정하여 이 정책을 더욱 강화할 수 있습니다. 이렇게 하면 AWS Organizations에 속하지 않는 IAM 보안 주체가 이미지를 푸시하거나 풀하지 못하게 합니다. 자세한 내용은 aws:PrincipalOrgID를 참조하세요.

com.amazonaws.*region*.ecr.dkr 및 com.amazonaws.*region*.ecr.api 엔드포인트 모두에 동일한 정책을 적용하는 것이 좋습니다.

컨테이너 에이전트 설정

Amazon ECS 컨테이너 에이전트 구성 파일에는 네트워크 보안과 관련된 여러 환경 변수가 포함되어 있습니다. ECS_AWSVPC_BLOCK_IMDS 및 ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST는 작업의 Amazon EC2 메타데이터 액세스를 차단하는 데 사용됩니다. HTTP_PROXY는 HTTP 프록시를 통해 라우팅하여 인터넷에 연결하도록 에이전트를 구성하는 데 사용됩니다. 프록시를 통해 라우팅하도록 에이전트 및 Docker 런타임을 구성하는 방법에 대한 지침은 HTTP Proxy Configuration을 참조하세요.

> **Important**
>
> AWS Fargate를 사용할 때는 이러한 설정을 사용할 수 없습니다.

네트워크 보안 권장 사항

Amazon VPC, 로드 밸런서 및 네트워크를 설정할 때 다음 작업을 수행하는 것이 좋습니다.

Amazon ECS에서 해당하는 경우 네트워크 암호화 사용

해당하는 경우 네트워크 암호화를 사용해야 합니다. PCI DSS와 같은 특정 규정 준수 프로그램에 따르면 데이터에 카드 소유자 데이터가 포함된 경우 전송 중 데이터를 암호화해야 합니다. 워크로드의 요구 사항이 유사한 경우 네트워크 암호화를 구성하세요.

최신 브라우저에서는 안전하지 않은 사이트에 연결할 경우 사용자에게 경고합니다. 서비스가 공용 로드 밸런서를 통해 제공되는 경우 TLS/SSL을 사용하여 클라이언트 브라우저에서 로드 밸런서로 전달되는 트래픽을 암호화하고 필요한 경우 백엔드로 다시 암호화하세요.

`awsvpc` 네트워크 모드 및 보안 그룹을 사용하여 Amazon ECS의 태스크와 다른 리소스 간의 트래픽 제어

작업 간이나 작업과 다른 네트워크 리소스 간의 트래픽을 제어해야 하는 경우 `awsvpc` 네트워크 모드 및 보안 그룹을 사용해야 합니다. 서비스가 ALB를 기반으로 하는 경우 보안 그룹을 사용하여 ALB와 동일한 보안 그룹을 사용하는 다른 네트워크 리소스에서 들어오는 인바운드 트래픽만 허용합니다. 애플리케이션이 NLB를 기반으로 하는 경우 Amazon VPC CIDR 범위 및 NLB에 할당된 고정 IP 주소에서 들어오는 인바운드 트래픽만 허용하도록 작업의 보안 그룹을 구성하세요.

또한 보안 그룹을 사용하여 Amazon RDS 데이터베이스와 같은 Amazon VPC 내의 다른 리소스와 작업 간의 트래픽을 제어해야 합니다.

네트워크 트래픽을 엄격하게 격리해야 하는 경우 별도의 Amazon VPC에 Amazon ECS 클러스터 생성

네트워크 트래픽을 엄격하게 격리해야 하는 경우 별도의 Amazon VPC에 클러스터를 생성해야 합니다. 보안 요구 사항이 엄격한 워크로드를 해당 요구 사항을 준수할 필요가 없는 워크로드가 포함된 클러스터에서 실행하지 마세요. 엄격한 네트워크 격리가 필수인 경우 별도의 Amazon VPC에 클러스터를 생성하고 Amazon VPC 엔드포인트를 사용하여 다른 Amazon VPC에 서비스를 선택적으로 노출하세요. 자세한 내용은 VPC 엔드포인트를 참조하십시오.

Amazon ECS에 대해 보증되는 경우 AWS PrivateLink 엔드포인트 구성

적정한 사유가 있는 경우 AWS PrivateLink 엔드포인트를 구성해야 합니다. 보안 정책으로 인해 인터넷 게이트웨이(IGW)를 Amazon VPC에 연결할 수 없는 경우, Amazon ECS 및 기타 서비스(예: Amazon

ECR, AWS Secrets Manager 및 Amazon CloudWatch)에 대한 AWS PrivateLink 엔드포인트를 구성하세요.

Amazon VPC 흐름 로그를 사용하여 Amazon ECS에서 장기 실행 태스크와 주고받는 트래픽 분석

Amazon VPC 흐름 로그를 사용하여 장기 실행 작업과 주고받는 트래픽을 분석해야 합니다. awsvpc 네트워크 모드를 사용하는 작업은 자체 ENI를 가져옵니다. 이렇게 하면 개별 작업과 주고받는 트래픽을 Amazon VPC 흐름 로그를 사용하여 모니터링할 수 있습니다. Amazon VPC 흐름 로그(v3)의 최신 업데이트에서는 VPC ID, 서브넷 ID 및 인스턴스 ID를 비롯한 트래픽 메타데이터를 포함하도록 로그를 보강합니다. 이 메타데이터를 사용하면 조사 범위를 좁힐 수 있습니다. 자세한 내용은 Amazon VPC 흐름 로그를 참조하세요.

> **Note**
>
> 컨테이너의 일시적인 특성 때문에 흐름 로그가 다른 컨테이너 사이나 컨테이너와 다른 네트워크 리소스 사이의 트래픽 패턴을 분석하는 데 항상 효과적인 방법은 아닙니다.

Amazon ECS 태스크 및 컨테이너 보안 모범 사례

컨테이너 이미지를 공격에 대한 첫 번째 방어선으로 보아야 합니다. 이미지가 안전하지 않고 잘못 구성되어 있으면 공격자가 컨테이너의 경계를 벗어나 호스트에 액세스할 수 있습니다. 이러한 상황이 발생할 위험을 줄이려면 다음과 같이 해야 합니다.

작업 및 컨테이너를 설정할 때 다음을 수행하는 것이 좋습니다.

최소로 생성하거나 distroless 이미지 사용

먼저 컨테이너 이미지에서 불필요한 바이너리를 모두 제거합니다. Amazon ECR 퍼블릭 갤러리의 익숙하지 않은 이미지를 사용하는 경우 이미지를 검사하여 각 컨테이너 레이어의 콘텐츠를 참조하는지 확인합니다. 이를 위해 Dive와 같은 애플리케이션을 사용할 수 있습니다.

또는 애플리케이션과 해당 런타임 종속성만 포함하는 distroless 이미지를 사용할 수도 있습니다. 이런 이미지는 패키지 관리자 또는 셸을 포함하지 않습니다. Distroless 이미지는 “스캐너의 신호 대 노이즈 비율을 개선하고 프로비넌스를 필요한 만큼으로 설정하여 부담을 줄입니다”. 자세한 내용을 알아보려면 GitHub 설명서의 distroless를 참조하세요.

Docker에는 scratch라고 하는 예약된 최소 이미지에서 이미지를 생성하는 메커니즘이 있습니다. 자세한 내용은 Docker 설명서의 Creating a simple parent image using scratch를 참조하세요. Go와 같은

언어를 사용하면 정적 연결 바이너리를 생성하고 Dockerfile에서 참조할 수 있습니다. 다음 예제에서 이 작업을 수행하는 방법을 안내합니다.

```
############################
# STEP 1 build executable binary
############################
FROM golang:alpine AS builder
# Install git.
# Git is required for fetching the dependencies.
RUN apk update && apk add --no-cache git
WORKDIR $GOPATH/src/mypackage/myapp/
COPY . .
# Fetch dependencies.
# Using go get.
RUN go get -d -v
# Build the binary.
RUN go build -o /go/bin/hello
############################
# STEP 2 build a small image
############################
FROM scratch
# Copy our static executable.
COPY --from=builder /go/bin/hello /go/bin/hello
# Run the hello binary.
ENTRYPOINT ["/go/bin/hello"]
This creates a container image that consists of your application and nothing else,
 making it extremely secure.
```

앞의 예제는 다단계 빌드의 예이기도 합니다. 이러한 유형의 빌드는 컨테이너 레지스트리로 푸시되는 최종 이미지의 크기를 최소화하는 데 사용할 수 있으므로 보안 측면에서 매력적입니다. 빌드 도구 및 기타 불필요한 바이너리가 없는 컨테이너 이미지는 이미지의 공격 표면을 줄여 보안 태세를 향상합니다. 다단계 빌드에 대한 자세한 내용은 Docker 설명서의 다단계 빌드를 참조하세요.

이미지를 스캔하여 취약성 확인

컨테이너 이미지는 해당하는 가상 머신과 마찬가지로 취약성이 있는 바이너리 및 애플리케이션 라이브러리를 포함하거나 시간에 따라 취약성이 증가할 수 있습니다. 악용을 방지하는 가장 좋은 방법은 이미지 스캐너로 이미지를 정기적으로 스캔하는 것입니다.

Amazon ECR은 일반적인 취약성 및 노출도(CVE) 데이터베이스를 사용하는 두 가지 버전의 기본 스캔을 제공합니다.

- AWS 네이티브 기본 스캔 - 현재 GA이며 권장되는 AWS 네이티브 기술을 사용합니다. 이 향상된 기본 스캔은 널리 사용되는 다양한 운영 체제에서 고객에게 더 나은 스캔 결과와 취약성 감지를 제공하도록 설계되었습니다. 이를 통해 고객은 컨테이너 이미지의 보안을 더욱 강화할 수 있습니다. 모든 신규 고객 레지스트리에는 기본적으로 이 향상된 버전이 선택됩니다.
- Clair 기본 스캔 - 오픈 소스 Clair 프로젝트를 사용하는 이전 기본 스캔 버전으로 더 이상 사용되지 않습니다. Clair에 대한 자세한 내용은 GitHub의 Clair를 참조하세요.

AWS 네이티브 기본 스캔과 Clair 기본 스캔 모두 2024년 9월 이후 추가된 리전을 제외하고 리전별 AWS 서비스에 나열된 모든 리전에서 지원됩니다. Clair 지원은 더 이상 사용되지 않으므로 Clair는 새 리전이 추가될 때 지원되지 않으며 2025년 10월 1일부터 모든 리전에서 더 이상 지원되지 않습니다.

Amazon ECR은 가능한 경우 업스트림 배포 소스의 CVE에 대한 심각도를 사용합니다. 그렇지 않으면 공통 취약성 평가 시스템(CVSS) 점수가 사용됩니다. CVSS 점수를 사용하여 NVD 취약성 심각도 등급을 얻을 수 있습니다. 자세한 내용은 NVD 취약성 심각도 등급을 참조하세요.

또한 스캔 결과를 Amazon ECR 콘솔 내에서 보거나 DescribeImageScanFindings API를 호출하여 확인할 수 있습니다. `HIGH` 또는 `CRITICAL` 취약성이 있는 이미지는 삭제하거나 다시 빌드해야 합니다. 배포된 이미지에서 취약성이 발견되면 가능한 한 빨리 교체해야 합니다.

Docker Desktop Edge 버전 2.3.6.0 이상에서는 로컬 이미지를 스캔할 수 있습니다. 스캔은 애플리케이션 보안 서비스인 Snyk에서 제공합니다. 취약성이 발견되면 Snyk는 Dockerfile에서 취약성이 있는 계층 및 종속성을 식별합니다. 또한 취약성이 적은 더 슬림한 기본 이미지를 사용하거나 특정 패키지를 최신 버전으로 업그레이드하는 것과 같은 안전한 대안도 추천합니다. Docker 스캔을 사용하면 개발자가 이미지를 레지스트리에 푸시하기 전에 잠재적인 보안 문제를 해결할 수 있습니다.

- Automating image compliance using Amazon ECR and AWS Security Hub에서는 AWS Security Hub에서 Amazon ECR의 취약성 정보를 표시하고 취약한 이미지에 대한 액세스를 차단하여 문제 해결을 자동화하는 방법을 설명합니다.

이미지에서 특수 권한 제거

액세스 권한 플래그 `setuid` 및 `setgid`가 지정되면 실행 파일의 소유자 또는 그룹의 권한으로 실행 파일을 실행할 수 있습니다. 이러한 액세스 권한이 있는 바이너리는 권한을 에스컬레이션하는 데 사용될 수 있으므로 이미지에서 모두 제거하세요. 악의적인 목적으로 사용될 수 `nc` 및 `curl`과 같은 셸 및 유틸리티를 모두 제거하는 것이 좋습니다. `setuid` 및 `setgid` 액세스 권한이 있는 파일을 찾으려면 다음 명령을 사용할 수 있습니다.

```
find / -perm /6000 -type f -exec ls -ld {} \;
```

이러한 파일에서 이러한 특수 권한을 제거하려면 컨테이너 이미지에 다음 지시문을 추가합니다.

```
RUN find / -xdev -perm /6000 -type f -exec chmod a-s {} \; || true
```

큐레이팅한 이미지 세트 생성

개발자가 직접 이미지를 생성하도록 하지 말고, 조직의 다양한 애플리케이션 스택에 대해 검증된 이미지 세트를 생성하세요. 이렇게 하면 개발자는 Dockerfile 작성 방법을 배울 필요 없이 코드 작성에 집중할 수 있습니다. 변경 사항이 코드베이스에 병합되면 CI/CD 파이프라인은 자산을 자동으로 컴파일한 다음 아티팩트 리포지토리에 저장할 수 있습니다. 그리고 마지막으로, 아티팩트를 적절한 이미지에 복사한 다음 Amazon ECR과 같은 Docker 레지스트리로 푸시할 수 있습니다. 최소한 기본 이미지 세트를 생성해야 합니다. 그래야만 개발자가 이를 바탕으로 자체 Dockerfile을 생성할 수 있습니다. 이미지를 Docker Hub에서 가져오지 마세요. 이미지에 무엇이 들어 있는지 항상 알 수 있는 것은 아니며, 상위 1000개 이미지 중 약 5분의 1에 취약성이 있습니다. 이러한 이미지의 목록과 취약성은 https://vulnerablecontainers.org/에서 확인할 수 있습니다.

애플리케이션 패키지 및 라이브러리에서 취약성 스캔

이제는 오픈 소스 라이브러리를 일반적으로 사용합니다. 운영 체제 및 OS 패키지와 마찬가지로 이러한 라이브러리에는 취약성이 있을 수 있습니다. 개발 수명 주기의 일부로 이러한 라이브러리를 스캔하고 중요한 취약성이 발견되면 업데이트해야 합니다.

Docker Desktop은 Snyk를 사용하여 로컬 스캔을 수행합니다. Snyk는 오픈 소스 라이브러리의 취약성과 잠재적 라이선스 문제를 찾는 데에도 사용할 수 있습니다. 개발자 워크플로에 직접 통합하여 오픈 소스 라이브러리로 인한 위험을 완화할 수 있습니다. 자세한 정보는 다음의 주제를 참조하세요.

- Open Source Application Security Tools에는 애플리케이션의 취약성을 탐지하기 위한 도구 목록이 포함되어 있습니다.

정적 코드 분석 수행

컨테이너 이미지를 빌드하기 전에 정적 코드 분석을 수행해야 합니다. 이 분석은 소스 코드를 대상으로 수행되며 코딩 오류뿐 아니라 오류 주입과 같이 악의적인 행위자가 악용할 수 있는 코드를 식별하는 데 사용됩니다. Amazon Inspector를 사용할 수 있습니다. 자세한 내용은 Amazon Inspector 사용 설명서의 Amazon Inspector로 Amazon ECR 컨테이너 이미지 스캔을 참조하세요.

루트가 아닌 사용자로 컨테이너 실행

루트가 아닌 사용자로 컨테이너를 실행해야 합니다. 기본적으로 컨테이너는 Dockerfile에 USER 지시문이 포함되어 있지 않으면 `root` 사용자로 실행됩니다. Docker에서 할당하는 기본 Linux 기능에 따라 `root`로 실행할 수 있는 작업이 제한되지만, 그 효과는 미미합니다. 예를 들어 `root`로 실행되는 컨테이너는 여전히 디바이스에 액세스할 수 없습니다.

CI/CD 파이프라인의 일부로 Dockerfiles를 lint하여 USER 지시문을 찾고 이 지시문이 없으면 빌드할 수 없게 해야 합니다. 자세한 정보는 다음의 주제를 참조하세요.

- Dockerfile-lint는 파일이 모범 사례를 따르는지 확인하는 데 사용할 수 있는 RedHat의 오픈 소스 도구입니다.
- Hadolint는 모범 사례를 준수하는 Docker 이미지를 빌드하기 위한 또 다른 도구입니다.

읽기 전용 루트 파일 시스템 사용

읽기 전용 루트 파일 시스템을 사용해야 합니다. 컨테이너의 루트 파일 시스템은 기본적으로 쓰기 가능합니다. RO(읽기 전용) 루트 파일 시스템으로 컨테이너를 구성하면 데이터를 유지할 수 있는 위치를 명시적으로 정의해야 합니다. 이 경우 특별히 권한을 부여받지 않는 한 컨테이너의 파일 시스템에 쓸 수 없으므로 공격 표면이 줄어듭니다.

> **Note**
>
> 읽기 전용 루트 파일 시스템을 사용하면 파일 시스템에 쓸 수 있어야 하는 특정 OS 패키지에서 문제가 발생할 수 있습니다. 읽기 전용 루트 파일 시스템을 사용할 계획이라면 사전에 철저히 테스트하세요.

CPU 및 메모리 제한과 함께 작업 구성(Amazon EC2)

CPU 및 메모리 제한과 함께 작업을 구성하여 다음과 같은 위험을 최소화해야 합니다. 작업의 리소스 제한은 작업 내의 모든 컨테이너에서 예약할 수 있는 CPU 및 메모리 양의 상한을 설정합니다. 제한을 설정하지 않으면 작업에서 호스트의 CPU와 메모리에 액세스할 수 있습니다. 이로 인해 공유 호스트에 배포된 작업이 다른 작업의 시스템 리소스를 부족하게 만드는 문제가 발생할 수 있습니다.

> **Note**
>
> AWS Fargate 기반 Amazon ECS 작업에서는 CPU 및 메모리 제한 값이 결제 목적으로 사용되므로 이러한 값을 지정해야 합니다. Amazon ECS Fargate에서는 각 작업이 자체 전용 인스턴스에서 실행되므로 하나의 작업이 시스템 리소스를 모두 독차지하더라도 문제가 되지 않습니다. 메모리 제한을 지정하지 않는 경우 Amazon ECS는 각 컨테이너에 최소 4MB를 할당합니다. 마찬가지로 작업의 CPU 제한을 설정하지 않으면 Amazon ECS 컨테이너 에이전트는 최소 2개 CPU를 할당합니다.

Amazon ECR에서 변경 불가능한 태그 사용

Amazon ECR에서는 변경 불가능한 태그를 사용하여 이미지를 구성할 수 있으며, 구성해야 합니다. 이렇게 하면 이미지의 변경 또는 업데이트된 버전을 동일한 태그를 사용하여 이미지 리포지토리로 푸시하지 못하게 됩니다. 따라서 공격자가 손상된 버전의 이미지를 동일한 태그가 지정된 이미지에 푸시하는 것을 방지할 수 있습니다. 변경 불가능한 태그를 사용하면 결과적으로 변경할 때마다 다른 태그의 새 이미지를 푸시할 수밖에 없게 됩니다.

컨테이너를 privileged로 실행하지 않음(Amazon EC2)

컨테이너를 privileged로 실행하지 않아야 합니다. 백그라운드의 경우 `privileged`로 실행되는 컨테이너는 호스트에서 확장된 권한으로 실행됩니다. 즉, 이 컨테이너는 호스트에서 `root`에 할당된 Linux 기능을 모두 상속합니다. 이 파라미터 사용을 엄격하게 제한하거나 금지해야 합니다. Amazon ECS 컨테이너 에이전트 환경 변수 `ECS_DISABLE_PRIVILEGED`를 `true`로 설정하여 `privileged`가 필요하지 않은 경우 특정 호스트에서 컨테이너가 `privileged`로 실행되지 않게 하는 것이 좋습니다. 또는 AWS Lambda를 사용하여 작업 정의를 스캔하고 `privileged` 파라미터를 사용하는지 확인할 수 있습니다.

> **Note**
>
> AWS Fargate 기반 Amazon ECS에서는 컨테이너를 `privileged`로 실행할 수 없습니다.

컨테이너에서 불필요한 Linux 기능 제거

다음은 Docker 컨테이너에 할당되는 기본 Linux 기능의 목록입니다. 각 기능에 대한 자세한 내용은 Overview of Linux Capabilities를 참조하세요.

```
CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_FOWNER, CAP_FSETID, CAP_KILL,
CAP_SETGID, CAP_SETUID, CAP_SETPCAP, CAP_NET_BIND_SERVICE,
CAP_NET_RAW, CAP_SYS_CHROOT, CAP_MKNOD, CAP_AUDIT_WRITE,
CAP_SETFCAP
```

컨테이너에서 위에 나열된 Docker 커널 기능 중 필요하지 않은 것이 있으면 컨테이너에서 삭제하는 것이 좋습니다. 각 Docker 커널 기능에 대한 자세한 내용은 KernalCapabilities를 참조하세요. 다음을 수행하여 사용 중인 기능을 확인할 수 있습니다.

- OS 패키지 libcap-ng를 설치하고 `pscap` 유틸리티를 실행하여 각 프로세스에서 사용 중인 기능을 나열합니다.
- 또한 capsh를 사용하여 프로세스에서 사용 중인 기능의 암호를 해제할 수 있습니다.

고객 관리형 키(CMK)를 사용하여 Amazon ECR로 푸시된 이미지 암호화

고객 관리형 키(CMK)를 사용하여 Amazon ECR로 푸시된 이미지를 암호화해야 합니다. Amazon ECR로 푸시된 이미지는 저장 시 AWS Key Management Service(AWS KMS) 관리형 키를 사용하여 자동으로 암호화됩니다. 대신 자체 키를 사용하고 싶은 경우 이제 Amazon ECR에서는 고객 관리형 키(CMK)를 사용한 AWS KMS 암호화를 지원합니다. CMK를 사용하여 서버 측 암호화를 활성화하기 전에 설명서의 저장된 데이터 암호화에 열거된 고려 사항을 검토하세요.

Amazon ECS에 관한 자습서

다음 자습서는 Amazon ECS 사용 시 일반적인 태스크를 수행하는 방법을 설명합니다.

다음 자습서 중 하나를 사용하여 Amazon ECS를 시작하는 방법에 대해 자세히 알아볼 수 있습니다.

자습서 개요	자세히 알아보기	
Fargate에서 Amazon ECS를 시작합니다.	Fargate 시작 유형에 대한 Amazon ECS Linux 작업을 생성하는 방법 알아보기	
Fargate에서 Windows 컨테이너를 시작합니다.	Fargate 시작 유형에 대한 Amazon ECS Windows 작업을 생성하는 방법 알아보기	
EC2 시작 유형의 Windows 컨테이너를 시작합니다.	EC2 시작 유형에 대한 Amazon ECS Windows 작업을 생성하는 방법 알아보기	

다음 자습서 중 하나를 사용하여 AWS CLI를 통해 Amazon ECS에 작업을 배포할 수 있습니다.

자습서 개요	자세히 알아보기	
Fargate 시작 유형에 대한 Linux 작업 생성	AWS CLI를 사용하여 Fargate 시작 유형에 대한 Amazon ECS Linux 작업 생성	
Fargate 시작 유형에 대한 Windows 작업을 생성합니다.	AWS CLI를 사용하여 Fargate 시작 유형에 대한 Amazon ECS Windows 작업 생성	
EC2 시작 유형에 대한 Linux 작업을 생성합니다.	AWS CLI를 사용하여 EC2 시작 유형에 대한 Amazon ECS 작업 생성	

다음 자습서 중 하나를 사용하여 모니터링 및 로깅에 대해 자세히 알아볼 수 있습니다.

자습서 개요	자세히 알아보기	
Amazon ECS 작업 이벤트를 수신 대기하고 이를 CloudWatch Logs 로그 스트림에 기록하는 간단한 Lambda 함수를 설정합니다.	CloudWatch Events 이벤트를 수신 대기하도록 Amazon ECS 구성	
필수 컨테이너 중 하나가 종료되어 작업 실행이 중지된 작업 이벤트만 캡처하는 Amazon EventBridge 이벤트 규칙을 구성합니다.	Amazon ECS 작업 중지 이벤트에 대한 Amazon Simple Notification Service 알림 보내기	
원래 하나의 컨텍스트에 속하지만 여러 레코드 또는 여러 로그 줄로 분할된 로그 메시지를 연결합니다.	여러 줄 또는 스택 추적 Amazon ECS 로그 메시지 연결	
Amazon ECS에서 실행 중인 Windows 인스턴스에 Fluent Bit 컨테이너를 배포하여 중앙 집중식 로깅을 위해 Windows 작업에서 생성된 로그를 Amazon CloudWatch로 스트리밍합니다.	Amazon ECS Windows 컨테이너에서 Fluent Bit 배포	

다음 자습서 중 하나를 사용하여 Amazon ECS의 그룹 관리형 서비스 계정과 함께 Active Directory 인증을 사용하는 방법에 대해 자세히 알아볼 수 있습니다.

자습서 개요	자세히 알아보기	
EC2의 Linux 컨테이너에서 그룹 관리형 서비스 계정을 사용합니다.	Amazon ECS에서 EC2 Linux 컨테이너에 대해 gMSA 사용	

자습서 개요	자세히 알아보기	
EC2의 Windows 컨테이너에서 그룹 관리형 서비스 계정을 사용합니다.	Amazon ECS의 EC2 Windows 컨테이너에 대해 gMSA를 사용하는 방법 알아보기	
Fargate의 Linux 컨테이너에서 그룹 관리형 서비스 계정을 사용합니다.	Fargate의 Linux 컨테이너에서 gMSA 사용	
도메인이 없는 그룹 관리형 서비스 계정으로 Active Directory에 액세스할 수 있도록 자격 증명을 포함하는 Windows 컨테이너를 실행하는 작업을 생성합니다.	AWS CLI를 사용하여 도메인이 없는 gMSA로 Amazon ECS Windows 컨테이너 사용	

AWS CLI를 사용하여 Fargate 시작 유형에 대한 Amazon ECS Linux 작업 생성

다음 단계는 AWS CLI를 사용하여 Amazon ECS에서 클러스터를 설정하고, 태스크 정의를 등록하고, Linux 태스크를 실행하고, 기타 일반적인 시나리오를 수행하는 데 도움이 됩니다. AWS CLI의 최신 버전을 사용합니다. 최신 버전으로 업그레이드하는 방법에 대한 자세한 내용은 AWS CLI 최신 버전의 설치 또는 업데이트를 참조하세요.

> **Note**
>
> 듀얼 스택 서비스 엔드포인트를 사용하면 AWS CLI, SDK 및 Amazon ECS API에서 IPv4 및 IPv6 모두를 통해 Amazon ECS와 상호 작용할 수 있습니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

주제

- 사전 조건
- 1단계: 클러스터 생성
- 2단계: Linux 태스크 정의 등록

- 3단계: 작업 정의 나열
- 4단계: 서비스 생성
- 5단계: 서비스 나열
- 6단계: 실행 서비스 설명
- 7단계: 테스트
- 8단계: 정리

사전 조건

이 자습서에서는 다음 사전 조건이 충족되었다고 가정합니다.

- 최신 버전의 AWS CLI가 설치 및 구성됩니다. AWS CLI 설치 또는 업그레이드에 대한 자세한 내용은 AWS CLI 최신 버전 설치 또는 업데이트를 참고하세요.
- Amazon ECS 사용 설정의 단계가 완료되었습니다.
- IAM 사용자는 AmazonECS_FullAccess IAM 정책 예제에 지정된 필수 권한을 가집니다.
- 사용할 VPC 및 보안 그룹이 생성되었습니다. 이 자습서에서는 Amazon ECR Public에서 호스팅되는 컨테이너 이미지를 사용하므로 태스크에서 인터넷에 액세스할 수 있어야 합니다. 작업에 인터넷 경로를 제공하려면 다음 옵션 중 하나를 사용합니다.
  - 탄력적 IP 주소가 있는 NAT 게이트웨이와 함께 프라이빗 서브넷을 사용합니다.
  - 퍼블릭 서브넷을 사용하고 퍼블릭 IP 주소를 작업에 할당합니다.

  자세한 내용은 the section called “Virtual Private Cloud 생성” 섹션을 참조하세요.

  보안 그룹 및 규칙에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 VPC에 대한 기본 보안 그룹과 규칙 예를 참조하세요.
- 프라이빗 서브넷을 사용하여 이 자습서를 따르면 Amazon ECS Exec을 사용하여 컨테이너와 직접 상호 작용하고 배포를 테스트할 수 있습니다. ECS Exec을 사용하려면 작업 IAM 역할을 생성해야 합니다. 태스크 IAM 역할 및 기타 사전 조건에 대한 자세한 내용은 Amazon ECS Exec을 사용하여 Amazon ECS 컨테이너 모니터링을 참조하세요.
- (선택 사항) AWS CloudShell은 고객에게 자체 EC2 인스턴스를 생성할 필요 없이 명령줄을 제공하는 도구입니다. 자세한 내용은 AWS CloudShell 사용 설명서의 AWS CloudShell이란 무엇입니까? 섹션을 참조하세요.

1단계: 클러스터 생성

계정에는 기본적으로 default 클러스터가 할당됩니다.

> **Note**
>
> 제공된 default 클러스터를 사용하는 이점은 후속 명령에서 --cluster *cluster_name* 옵션을 지정할 필요가 없다는 것입니다. 기본 클러스터가 아닌 자체 클러스터를 생성하는 경우, 해당 클러스터에 사용할 각 명령에 --cluster *cluster_name*을 지정해야 합니다.

다음 명령을 사용하여 고유한 이름의 자체 클러스터를 생성합니다.

```
aws ecs create-cluster --cluster-name fargate-cluster
```

출력:

```
{
    "cluster": {
        "status": "ACTIVE",
        "defaultCapacityProviderStrategy": [],
        "statistics": [],
        "capacityProviders": [],
        "tags": [],
        "clusterName": "fargate-cluster",
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
        ],
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"
    }
}
```

2단계: Linux 태스크 정의 등록

ECS 클러스터에서 작업을 실행하려면 먼저 작업 정의를 등록해야 합니다. 태스크 정의는 그룹화된 컨테이너의 목록입니다. 다음 예제는 Docker Hub에서 호스팅되는 httpd 컨테이너 이미지를 사용하여 PHP 웹 앱을 생성하는 간단한 작업 정의입니다. 사용 가능한 태스크 정의 파라미터에 대한 자세한 정보는 Amazon ECS 작업 정의 섹션을 참조하세요. 이 자습서에서 `taskRoleArn`는 프라이빗 서브넷에 작업을 배포하고 배포를 테스트하려는 경우에만 필요합니다. `taskRoleArn`을 사전 조건에 설명한 대로 ECS Exec을 사용하기 위해 생성한 IAM 작업 역할로 바꿉니다.

```
{
       "family": "sample-fargate",
       "networkMode": "awsvpc",
       "taskRoleArn": "arn:aws:iam::aws_account_id:role/execCommandRole",
       "containerDefinitions": [
           {
               "name": "fargate-app",
               "image": "public.ecr.aws/docker/library/httpd:latest",
               "portMappings": [
                   {
                       "containerPort": 80,
                       "hostPort": 80,
                       "protocol": "tcp"
                   }
               ],
               "essential": true,
               "entryPoint": [
                   "sh",
                   "-c"
               ],
               "command": [
                   "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
               ]
           }
       ],
       "requiresCompatibilities": [
           "FARGATE"
       ],
```

```
        "cpu": "256",
        "memory": "512"
}
```

작업 정의 JSON을 파일로 저장하고 `--cli-input-json file://path_to_file.json` 옵션을 사용하여 전달합니다.

컨테이너 정의에 JSON 파일을 사용하려면

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/fargate-task.json
```

register-task-definition 명령은 등록을 완료한 후 작업 정의의 설명을 반환합니다.

3단계: 작업 정의 나열

언제라도 list-task-definitions 명령을 사용하여 계정의 작업 정의를 나열할 수 있습니다. 이 명령은 run-task 또는 start-task를 호출할 때 함께 사용할 수 있는 `family` 및 `revision` 값을 출력합니다.

```
aws ecs list-task-definitions
```

출력:

```
{
    "taskDefinitionArns": [
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate:1"
    ]
}
```

4단계: 서비스 생성

계정에 대한 작업을 등록한 후 클러스터에서 등록된 작업에 대해 서비스를 생성할 수 있습니다. 이 예제에서는 클러스터에서 실행 중인 `sample-fargate:1` 작업 정의 인스턴스 하나를 사용하여 서비스를 생성합니다. 이 작업에는 인터넷 경로가 필요하므로 이 작업을 수행할 수 있는 두 가지 방법이 있습니다. 한 가지 방법은 퍼블릭 서브넷에서 탄력적 IP 주소가 있는 NAT 게이트웨이로 구성된 프라이빗 서브넷을 사용하는 것입니다. 또 다른 방법은 퍼블릭 서브넷을 사용하고 작업에 퍼블릭 IP 주소를 할당하는 것입니다. 아래 두 가지 예를 모두 제공합니다.

프라이빗 서브넷을 사용하는 예. Amazon ECS Exec을 사용하려면 이 `enable-execute-command` 옵션이 필요합니다.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234]}" --enable-execute-command
```

퍼블릭 서브넷을 사용하는 예.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234],assignPublicIp=ENABLED}"
```

create-service 명령은 등록을 완료한 후 작업 정의의 설명을 반환합니다.

5단계: 서비스 나열

클러스터의 서비스를 나열합니다. 이전 섹션에서 생성한 서비스가 보일 것입니다. 이 명령에서 반환된 서비스 이름 또는 전체 ARN을 기록해 두었다가 나중에 서비스를 설명하는 데 사용할 수 있습니다.

```
aws ecs list-services --cluster fargate-cluster
```

출력:

```
{
    "serviceArns": [
        "arn:aws:ecs:region:aws_account_id:service/fargate-cluster/fargate-service"
    ]
}
```

6단계: 실행 서비스 설명

앞서 검색한 서비스 이름으로 서비스를 설명하여 작업에 관한 정보를 더 많이 가져옵니다.

```
aws ecs describe-services --cluster fargate-cluster --services fargate-service
```

성공하면 서비스 실패 및 서비스에 대한 설명이 반환됩니다. 예를 들어 `services` 섹션에서 실행 중이거나 보류 중인 작업 상태와 같이 배포에 대한 정보를 찾을 수 있습니다. 작업 정의, 네트워크 구성 및 타임스탬프가 지정된 이벤트에 대한 정보도 찾을 수 있습니다. 실패 섹션에서는 호출과 관련된 실패(있

는 경우)에 대한 정보를 찾을 수 있습니다. 문제 해결에 대한 자세한 정보는 서비스 이벤트 메시지를 참조하세요. 서비스 설명에 대한 자세한 정보는 서비스 설명을 참조하세요.


```
{
    "services": [
        {
            "networkConfiguration": {
                "awsvpcConfiguration": {
                    "subnets": [
                        "subnet-abcd1234"
                    ],
                    "securityGroups": [
                        "sg-abcd1234"
                    ],
                    "assignPublicIp": "ENABLED"
                }
            },
            "launchType": "FARGATE",
            "enableECSManagedTags": false,
            "loadBalancers": [],
            "deploymentController": {
                "type": "ECS"
            },
            "desiredCount": 1,
            "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster",
            "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-service",
            "deploymentConfiguration": {
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            },
            "createdAt": 1692283199.771,
            "schedulingStrategy": "REPLICA",
            "placementConstraints": [],
            "deployments": [
                {
                    "status": "PRIMARY",
                    "networkConfiguration": {
                        "awsvpcConfiguration": {
                            "subnets": [
                                "subnet-abcd1234"
                            ],
                            "securityGroups": [
                                "sg-abcd1234"
```

```
                        ],
                        "assignPublicIp": "ENABLED"
                    }
                },
                "pendingCount": 0,
                "launchType": "FARGATE",
                "createdAt": 1692283199.771,
                "desiredCount": 1,
                "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-
definition/sample-fargate:1",
                "updatedAt": 1692283199.771,
                "platformVersion": "1.4.0",
                "id": "ecs-svc/9223370526043414679",
                "runningCount": 0
            }
        ],
        "serviceName": "fargate-service",
        "events": [
            {
                "message": "(service fargate-service) has started 2 tasks: (task
 53c0de40-ea3b-489f-a352-623bf1235f08) (task d0aec985-901b-488f-9fb4-61b991b332a3).",
                "id": "92b8443e-67fb-4886-880c-07e73383ea83",
                "createdAt": 1510811841.408
            },
            {
                "message": "(service fargate-service) has started 2 tasks: (task
 b4911bee-7203-4113-99d4-e89ba457c626) (task cc5853e3-6e2d-4678-8312-74f8a7d76474).",
                "id": "d85c6ec6-a693-43b3-904a-a997e1fc844d",
                "createdAt": 1510811601.938
            },
            {
                "message": "(service fargate-service) has started 2 tasks: (task
 cba86182-52bf-42d7-9df8-b744699e6cfc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
                "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
                "createdAt": 1510811364.691
            }
        ],
        "runningCount": 0,
        "status": "ACTIVE",
        "serviceRegistries": [],
        "pendingCount": 0,
        "createdBy": "arn:aws:iam::aws_account_id:user/user_name",
        "platformVersion": "LATEST",
        "placementStrategy": [],
```

```
            "propagateTags": "NONE",
            "roleArn": "arn:aws:iam::aws_account_id:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
            "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-fargate:1"
        }
    ],
    "failures": []
}
```

7단계: 테스트

퍼블릭 서브넷을 사용하여 배포된 작업 테스트

태스크에 대한 탄력적 네트워크 인터페이스(ENI)를 가져올 수 있도록 서비스의 태스크를 설명합니다.

먼저 태스크 ARN을 가져오세요.

```
aws ecs list-tasks --cluster fargate-cluster --service fargate-service
```

출력에는 태스크 ARN이 포함됩니다.

```
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:123456789012:task/fargate-service/EXAMPLE
    ]
}
```

태스크를 설명하고 ENI ID를 찾습니다. `tasks` 파라미터에 태스크 ARN을 사용합니다.

```
aws ecs describe-tasks --cluster fargate-cluster --tasks arn:aws:ecs:us-
east-1:123456789012:task/service/EXAMPLE
```

연결 정보가 출력에 나열됩니다.

```
{
    "tasks": [
        {
            "attachments": [
                {
```

```
                    "id": "d9e7735a-16aa-4128-bc7a-b2d5115029e9",
                    "type": "ElasticNetworkInterface",
                    "status": "ATTACHED",
                    "details": [
                        {
                            "name": "subnetId",
                            "value": "subnetabcd1234"
                        },
                        {
                            "name": "networkInterfaceId",
                            "value": "eni-0fa40520aeEXAMPLE"
                        },
                    ]
                }
…
}
```

퍼블릭 IP 주소를 가져오기 위해 ENI를 설명합니다.

```
aws ec2 describe-network-interfaces --network-interface-id  eni-0fa40520aeEXAMPLE
```

퍼블릭 IP 주소가 출력에 있습니다.

```
{
    "NetworkInterfaces": [
        {
            "Association": {
                "IpOwnerId": "amazon",
                "PublicDnsName": "ec2-34-229-42-222.compute-1.amazonaws.com",
                "PublicIp": "198.51.100.2"
            },
…
}
```

웹 브라우저에 퍼블릭 IP 주소를 입력하면 Amazon ECS 샘플 애플리케이션이 표시되는 웹 페이지가 보여야 합니다.

프라이빗 서브넷을 사용하여 배포된 작업 테스트

작업을 설명하고 `ExecuteCommandAgent`가 실행 중인지 확인하기 위해 `managedAgents`를 찾습니다. 나중에 사용할 수 있도록 `privateIPv4Address`를 기록해 둡니다.

```
aws ecs describe-tasks --cluster fargate-cluster --tasks arn:aws:ecs:us-
east-1:123456789012:task/fargate-service/EXAMPLE
```

관리형 에이전트 정보가 출력에 나열됩니다.

```
{
     "tasks": [
        {
            "attachments": [
                {
                    "id": "d9e7735a-16aa-4128-bc7a-b2d5115029e9",
                    "type": "ElasticNetworkInterface",
                    "status": "ATTACHED",
                    "details": [
                        {
                            "name": "subnetId",
                            "value": "subnetabcd1234"
                        },
                        {
                            "name": "networkInterfaceId",
                            "value": "eni-0fa40520aeEXAMPLE"
                        },
                        {
                            "name": "privateIPv4Address",
                            "value": "10.0.143.156"
                        }
                    ]
                }
            ],
     ...
     "containers": [
         {
         ...
        "managedAgents": [
                        {
                            "lastStartedAt": "2023-08-01T16:10:13.002000+00:00",
                            "name": "ExecuteCommandAgent",
                            "lastStatus": "RUNNING"
                        }
                ],
         ...
    }
```

ExecuteCommandAgent가 실행 중인지 확인한 후 다음 명령을 실행하여 작업의 컨테이너에서 대화형 셸을 실행할 수 있습니다.

```
  aws ecs execute-command --cluster fargate-cluster \
      --task  arn:aws:ecs:us-east-1:123456789012:task/fargate-service/EXAMPLE  \
      --container  fargate-app \
      --interactive \
      --command "/bin/sh"
```

대화형 셸을 실행한 후 다음 명령을 실행하여 cURL을 설치합니다.

```
apt update
```

```
apt install curl
```

cURL을 설치한 후 이전에 얻은 프라이빗 IP 주소를 사용하여 다음 명령을 실행합니다.

```
 curl 10.0.143.156
```

Amazon ECS 샘플 애플리케이션 웹 페이지에 해당하는 HTML이 표시됩니다.

```
<html>
    <head>
     <title>Amazon ECS Sample App</title>
     <style>body {margin-top: 40px; background-color: #333;} </style>
    </head>
      <body>
      <div style=color:white;text-align:center>
      <h1>Amazon ECS Sample App</h1>
      <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p>
      </div>
      </body>
</html>
```

8단계: 정리

이 자습서로 완료를 한 후에 사용하지 않는 리소스에 대해 요금이 발생하는 것을 방지하기 위해 연결된 리소스를 정리해야 합니다.

서비스를 삭제합니다.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

클러스터를 삭제합니다.

```
aws ecs delete-cluster --cluster fargate-cluster
```

AWS CLI를 사용하여 Fargate 시작 유형에 대한 Amazon ECS Windows 작업 생성

다음 단계는 AWS CLI를 사용하여 Amazon ECS에서 클러스터를 설정하고, 태스크 정의를 등록하고, Windows 태스크를 실행하고, 기타 일반적인 시나리오를 수행하는 데 도움이 됩니다. 최신 버전의 AWS CLI를 사용하고 있는지 확인합니다. 최신 버전으로 업그레이드하는 방법에 대한 자세한 내용은 AWS CLI 최신 버전의 설치 또는 업데이트를 참조하세요.

> **Note**
>
> 듀얼 스택 서비스 엔드포인트를 사용하면 AWS CLI, SDK 및 Amazon ECS API에서 IPv4 및 IPv6 모두를 통해 Amazon ECS와 상호 작용할 수 있습니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

주제

- 사전 조건
- 1단계: 클러스터 생성
- 2단계: Windows 태스크 정의 등록
- 3단계: 태스크 정의 나열
- 4단계: 서비스 생성
- 5단계: 서비스 나열
- 6단계: 실행 서비스 설명
- 7단계: 정리

사전 조건

이 자습서에서는 다음 사전 조건이 충족되었다고 가정합니다.

- 최신 버전의 AWS CLI가 설치 및 구성됩니다. AWS CLI 설치 및 업그레이드에 대한 자세한 내용은 AWS CLI 최신 버전의 설치 또는 업데이트를 참고하세요.
- Amazon ECS 사용 설정의 단계가 완료되었습니다.
- IAM 사용자는 AmazonECS_FullAccess IAM 정책 예제에 지정된 필수 권한을 가집니다.
- 사용할 VPC 및 보안 그룹이 생성되었습니다. 이 자습서에서는 Docker Hub에서 호스팅되는 컨테이너 이미지를 사용하므로 작업에서 인터넷에 액세스할 수 있어야 합니다. 작업에 인터넷 경로를 제공하려면 다음 옵션 중 하나를 사용합니다.
  - 탄력적 IP 주소가 있는 NAT 게이트웨이와 함께 프라이빗 서브넷을 사용합니다.
  - 퍼블릭 서브넷을 사용하고 퍼블릭 IP 주소를 작업에 할당합니다.

  자세한 내용은 the section called “Virtual Private Cloud 생성” 섹션을 참조하세요.

  보안 그룹 및 규칙에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 VPC에 대한 기본 보안 그룹과 규칙 예를 참조하세요.
- (선택 사항) AWS CloudShell은 고객에게 자체 EC2 인스턴스를 생성할 필요 없이 명령줄을 제공하는 도구입니다. 자세한 내용은 AWS CloudShell 사용 설명서의 AWS CloudShell이란 무엇입니까? 섹션을 참조하세요.

1단계: 클러스터 생성

계정에는 기본적으로 `default` 클러스터가 할당됩니다.

> **Note**
>
> 제공된 `default` 클러스터를 사용하는 이점은 후속 명령에서 `--cluster` *`cluster_name`* 옵션을 지정할 필요가 없다는 것입니다. 기본 클러스터가 아닌 자체 클러스터를 생성하는 경우, 해당 클러스터에 사용할 각 명령에 `--cluster` *`cluster_name`*을 지정해야 합니다.

다음 명령을 사용하여 고유한 이름의 자체 클러스터를 생성합니다.

```
aws ecs create-cluster --cluster-name fargate-cluster
```

출력:

```
{
```

```
    "cluster": {
        "status": "ACTIVE",
        "statistics": [],
        "clusterName": "fargate-cluster",
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"
    }
}
```

2단계: Windows 태스크 정의 등록

Amazon ECS 클러스터에서 Windows 태스크를 실행하려면 먼저 태스크 정의를 등록해야 합니다. 태스크 정의는 그룹화된 컨테이너의 목록입니다. 다음은 웹 앱을 생성하는 단순 태스크 정의의 예입니다. 사용 가능한 태스크 정의 파라미터에 대한 자세한 정보는 Amazon ECS 작업 정의 섹션을 참조하세요.

```
{
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
```

```
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "networkMode": "awsvpc",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["FARGATE"]
}
```

위의 예시 JSON은 두 가지 방법으로 AWS CLI에 전달할 수 있습니다. 작업 정의 JSON을 파일로 저장하고 `--cli-input-json file://path_to_file.json` 옵션을 사용하여 전달할 수 있습니다.

컨테이너 정의에 JSON 파일을 사용하려면

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/fargate-task.json
```

register-task-definition 명령은 등록을 완료한 후 작업 정의의 설명을 반환합니다.

3단계: 태스크 정의 나열

언제라도 list-task-definitions 명령을 사용하여 계정의 작업 정의를 나열할 수 있습니다. 이 명령은 run-task 또는 start-task를 호출할 때 함께 사용할 수 있는 `family` 및 `revision` 값을 출력합니다.

```
aws ecs list-task-definitions
```

출력:

```
{
    "taskDefinitionArns": [
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate-windows:1"
    ]
}
```

4단계: 서비스 생성

계정에 대한 작업을 등록한 후 클러스터에서 등록된 작업에 대해 서비스를 생성할 수 있습니다. 이 예제에서는 클러스터에서 실행 중인 `sample-fargate:1` 작업 정의 인스턴스 하나를 사용하여 서비스

를 생성합니다. 이 작업에는 인터넷 경로가 필요하므로 이 작업을 수행할 수 있는 두 가지 방법이 있습니다. 한 가지 방법은 퍼블릭 서브넷에서 탄력적 IP 주소가 있는 NAT 게이트웨이로 구성된 프라이빗 서브넷을 사용하는 것입니다. 또 다른 방법은 퍼블릭 서브넷을 사용하고 작업에 퍼블릭 IP 주소를 할당하는 것입니다. 아래 두 가지 예를 모두 제공합니다.

프라이빗 서브넷을 사용하는 예.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service
 --task-definition sample-fargate-windows:1 --desired-count 1 --launch-type
 "FARGATE" --network-configuration "awsvpcConfiguration={subnets=[subnet-
abcd1234],securityGroups=[sg-abcd1234]}"
```

퍼블릭 서브넷을 사용하는 예.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service
 --task-definition sample-fargate-windows:1 --desired-count 1 --launch-type
 "FARGATE" --network-configuration "awsvpcConfiguration={subnets=[subnet-
abcd1234],securityGroups=[sg-abcd1234],assignPublicIp=ENABLED}"
```

create-service 명령은 등록을 완료한 후 작업 정의의 설명을 반환합니다.

5단계: 서비스 나열

클러스터의 서비스를 나열합니다. 이전 섹션에서 생성한 서비스가 보일 것입니다. 이 명령에서 반환된 서비스 이름 또는 전체 ARN을 기록해 두었다가 나중에 서비스를 설명하는 데 사용할 수 있습니다.

```
aws ecs list-services --cluster fargate-cluster
```

출력:

```
{
    "serviceArns": [
        "arn:aws:ecs:region:aws_account_id:service/fargate-service"
    ]
}
```

6단계: 실행 서비스 설명

앞서 검색한 서비스 이름으로 서비스를 설명하여 작업에 관한 정보를 더 많이 가져옵니다.

```
aws ecs describe-services --cluster fargate-cluster --services fargate-service
```

성공하면 서비스 실패 및 서비스에 대한 설명이 반환됩니다. 예를 들어 서비스 섹션에서 실행 중이거나 보류 중인 작업 상태와 같이 배포에 대한 정보를 찾을 수 있습니다. 작업 정의, 네트워크 구성 및 타임스탬프가 지정된 이벤트에 대한 정보도 찾을 수 있습니다. 실패 섹션에서는 호출과 관련된 실패(있는 경우)에 대한 정보를 찾을 수 있습니다. 문제 해결에 대한 자세한 정보는 서비스 이벤트 메시지를 참조하세요. 서비스 설명에 대한 자세한 정보는 서비스 설명을 참조하세요.

```
{
    "services": [
        {
            "status": "ACTIVE",
            "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-fargate-windows:1",
            "pendingCount": 2,
            "launchType": "FARGATE",
            "loadBalancers": [],
            "roleArn": "arn:aws:iam::aws_account_id:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
            "placementConstraints": [],
            "createdAt": 1510811361.128,
            "desiredCount": 2,
            "networkConfiguration": {
                "awsvpcConfiguration": {
                    "subnets": [
                        "subnet-abcd1234"
                    ],
                    "securityGroups": [
                        "sg-abcd1234"
                    ],
                    "assignPublicIp": "DISABLED"
                }
            },
            "platformVersion": "LATEST",
            "serviceName": "fargate-service",
            "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster",
            "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-service",
            "deploymentConfiguration": {
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            },
            "deployments": [
```

```
                {
                    "status": "PRIMARY",
                    "networkConfiguration": {
                        "awsvpcConfiguration": {
                            "subnets": [
                                "subnet-abcd1234"
                            ],
                            "securityGroups": [
                                "sg-abcd1234"
                            ],
                            "assignPublicIp": "DISABLED"
                        }
                    },
                    "pendingCount": 2,
                    "launchType": "FARGATE",
                    "createdAt": 1510811361.128,
                    "desiredCount": 2,
                    "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-
definition/sample-fargate-windows:1",
                    "updatedAt": 1510811361.128,
                    "platformVersion": "0.0.1",
                    "id": "ecs-svc/9223370526043414679",
                    "runningCount": 0
                }
            ],
            "events": [
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 53c0de40-ea3b-489f-a352-623bf1235f08) (task d0aec985-901b-488f-9fb4-61b991b332a3).",
                    "id": "92b8443e-67fb-4886-880c-07e73383ea83",
                    "createdAt": 1510811841.408
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 b4911bee-7203-4113-99d4-e89ba457c626) (task cc5853e3-6e2d-4678-8312-74f8a7d76474).",
                    "id": "d85c6ec6-a693-43b3-904a-a997e1fc844d",
                    "createdAt": 1510811601.938
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 cba86182-52bf-42d7-9df8-b744699e6cfc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
                    "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
                    "createdAt": 1510811364.691
                }
```

```
            ],
            "runningCount": 0,
            "placementStrategy": []
        }
    ],
    "failures": []
}
```

7단계: 정리

이 자습서로 완료를 한 후에 사용하지 않는 리소스에 대해 요금이 발생하는 것을 방지하기 위해 연결된 리소스를 정리해야 합니다.

서비스를 삭제합니다.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

클러스터를 삭제합니다.

```
aws ecs delete-cluster --cluster fargate-cluster
```

AWS CLI를 사용하여 EC2 시작 유형에 대한 Amazon ECS 작업 생성

다음 단계는 AWS CLI를 사용하여 Amazon ECS에서 클러스터를 설정하고, 태스크 정의를 등록하고, 태스크를 실행하고, 기타 일반적인 시나리오를 수행하는 데 도움이 됩니다. AWS CLI의 최신 버전을 사용합니다. 최신 버전으로 업그레이드하는 방법에 대한 자세한 내용은 AWS CLI 최신 버전의 설치 또는 업데이트를 참조하세요.

> **Note**
>
> 듀얼 스택 서비스 엔드포인트를 사용하면 AWS CLI, SDK 및 Amazon ECS API에서 IPv4 및 IPv6 모두를 통해 Amazon ECS와 상호 작용할 수 있습니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

주제

- 사전 조건
- 클러스터 생성
- Amazon ECS AMI로 컨테이너 인스턴스 시작
- 컨테이너 인스턴스 나열
- 컨테이너 인스턴스 설명
- 태스크 정의 등록
- 태스크 정의 나열
- 서비스 생성
- 서비스 나열
- 서비스 설명
- 실행 중인 태스크 설명
- 웹 서버 테스트
- 리소스 정리

사전 조건

이 자습서에서는 다음 사전 조건이 충족되었다고 가정합니다.

- 최신 버전의 AWS CLI가 설치 및 구성됩니다. AWS CLI 설치 및 업그레이드에 대한 자세한 내용은 AWS CLI 최신 버전의 설치 또는 업데이트를 참고하세요.
- Amazon ECS 사용 설정의 단계가 완료되었습니다.
- IAM 사용자는 AmazonECS_FullAccess IAM 정책 예제에 지정된 필수 권한을 가집니다.
- 사용할 컨테이너 인스턴스 IAM 역할이 생성되었습니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할 섹션을 참조하세요.
- 사용할 VPC가 생성되었습니다. 자세한 내용은 the section called “Virtual Private Cloud 생성” 섹션을 참조하세요.
- (선택 사항) AWS CloudShell은 고객에게 자체 EC2 인스턴스를 생성할 필요 없이 명령줄을 제공하는 도구입니다. 자세한 내용은 AWS CloudShell 사용 설명서의 AWS CloudShell이란 무엇입니까? 섹션을 참조하세요.

클러스터 생성

기본적으로 첫 번째 컨테이너 인스턴스를 시작할 때 계정에 `default` 클러스터가 생성됩니다.

> **Note**
>
> 제공된 `default` 클러스터를 사용하는 이점은 후속 명령에서 `--cluster` *`cluster_name`* 옵션을 지정할 필요가 없다는 것입니다. 기본 클러스터가 아닌 자체 클러스터를 생성하는 경우, 해당 클러스터에 사용할 각 명령에 `--cluster` *`cluster_name`*을 지정해야 합니다.

다음 명령을 사용하여 고유한 이름의 자체 클러스터를 생성합니다.

```
aws ecs create-cluster --cluster-name MyCluster
```

출력:

```
{
    "cluster": {
        "clusterName": "MyCluster",
        "status": "ACTIVE",
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/MyCluster"
    }
}
```

Amazon ECS AMI로 컨테이너 인스턴스 시작

컨테이너 인스턴스는 Amazon ECS 컨테이너 에이전트를 실행하고 클러스터에 등록된 EC2 인스턴스입니다. 이 섹션에서는 ECS 최적화 AMI를 사용하여 EC2 인스턴스를 시작합니다.

AWS CLI로 컨테이너 인스턴스 시작하려면

1. 다음 명령을 사용하여 AWS 리전에 대한 최신 ECS 최적화 Amazon Linux 2 AMI ID를 검색합니다. 이 명령은 AWS Systems Manager Parameter Store를 사용하여 최신 ECS 최적화 AMI ID를 가져옵니다. AMI에는 Amazon ECS 컨테이너 에이전트와 Docker 런타임이 사전 설치되어 있습니다.

   ```
   aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
   recommended --query 'Parameters[0].Value' --output text | jq -r '.image_id'
   ```

   출력:

   ```
   ami-abcd1234
   ```

2. 컨테이너 인스턴스를 관리할 SSH 액세스와 웹 서버에 대한 HTTP 액세스를 허용하는 보안 그룹을 생성합니다.

```
aws ec2 create-security-group --group-name ecs-tutorial-sg --description "ECS
 tutorial security group"
```

출력:

```
{
    "GroupId": "sg-abcd1234"
}
```

3. 다음 명령을 실행하여 보안 그룹에 인바운드 규칙을 추가합니다.

```
aws ec2 authorize-security-group-ingress --group-id sg-abcd1234 --protocol tcp --
port 80 --cidr 0.0.0.0/0
```

출력:

```
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-efgh5678",
            "GroupId": "sg-abcd1234",
            "GroupOwnerId": "123456789012",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 80,
            "ToPort": 80,
            "CidrIpv4": "0.0.0.0/0"
        }
    ]
}
```

이제 보안 그룹은 지정된 IP 범위에서 SSH 액세스를 허용하며 어디에서나 HTTP 액세스를 허용합니다. 프로덕션 환경에서는 SSH 액세스를 특정 IP 주소로 제한하고 필요에 따라 HTTP 액세스를 제한하는 것을 고려해야 합니다.

4. 컨테이너 인스턴스에 대한 SSH 액세스를 위해 EC2 키 페어를 생성합니다.

```
aws ec2 create-key-pair --key-name ecs-tutorial-key --query 'KeyMaterial' --output
 text > ecs-tutorial-key.pem
chmod 400 ecs-tutorial-key.pem
```

프라이빗 키는 SSH 액세스에 대한 적절한 권한이 있는 로컬 시스템에 저장됩니다.

5. ECS 최적화 AMI를 사용하여 EC2 인스턴스를 시작하고 클러스터에 조인하도록 구성합니다.

```
aws ec2 run-instances --image-id ami-abcd1234 --instance-type t3.micro --key-
name ecs-tutorial-key --security-group-ids sg-abcd1234 --iam-instance-profile
 Name=ecsInstanceRole --user-data '#!/bin/bash
echo ECS_CLUSTER=MyCluster >> /etc/ecs/ecs.config'
{
    "Instances": [
        {
            "InstanceId": "i-abcd1234",
            "ImageId": "ami-abcd1234",
            "State": {
                "Code": 0,
                "Name": "pending"
            },
            "PrivateDnsName": "",
            "PublicDnsName": "",
            "StateReason": {
                "Code": "pending",
                "Message": "pending"
            },
            "InstanceType": "t3.micro",
            "KeyName": "ecs-tutorial-key",
            "LaunchTime": "2025-01-13T10:30:00.000Z"
        }
    ]
}
```

사용자 데이터 스크립트는 인스턴스를 `MyCluster`에 등록하도록 Amazon ECS 에이전트를 구성합니다. 인스턴스는 에이전트에 필요한 권한을 제공하는 `ecsInstanceRole` IAM 역할을 사용합니다.

컨테이너 인스턴스 나열

컨테이너 인스턴스를 시작한지 몇 분 이내에 Amazon ECS 에이전트가 기본 클러스터에 인스턴스를 등록합니다. 다음 명령을 실행하여 클러스터의 컨테이너 인스턴스를 나열할 수 있습니다.

```
aws ecs list-container-instances --cluster default
```

출력:

```
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:container-instance/container_instance_ID"
    ]
}
```

컨테이너 인스턴스 설명

컨테이너 인스턴스의 ARN 또는 ID를 지정한 후 describe-container-instances 명령을 사용하여 잔여 및 등록된 CPU 및 메모리 리소스와 같은 인스턴스에 대한 가치 있는 정보를 가져올 수 있습니다.

```
aws ecs describe-container-instances --cluster MyCluster --container-
instances container_instance_ID
```

출력:

```
{
    "failures": [],
    "containerInstances": [
        {
            "status": "ACTIVE",
            "registeredResources": [
                {
                    "integerValue": 1024,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 995,
                    "longValue": 0,
```

```
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                },
                {
                    "name": "PORTS",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
                        "22",
                        "2376",
                        "2375",
                        "51678"
                    ],
                    "type": "STRINGSET",
                    "integerValue": 0
                },
                {
                    "name": "PORTS_UDP",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [],
                    "type": "STRINGSET",
                    "integerValue": 0
                }
            ],
            "ec2InstanceId": "instance_id",
            "agentConnected": true,
            "containerInstanceArn": "arn:aws:ecs:us-west-2:aws_account_id:container-
instance/container_instance_ID",
            "pendingTasksCount": 0,
            "remainingResources": [
                {
                    "integerValue": 1024,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 995,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
```

```
                "doubleValue": 0.0
            },
            {
                "name": "PORTS",
                "longValue": 0,
                "doubleValue": 0.0,
                "stringSetValue": [
                    "22",
                    "2376",
                    "2375",
                    "51678"
                ],
                "type": "STRINGSET",
                "integerValue": 0
            },
            {
                "name": "PORTS_UDP",
                "longValue": 0,
                "doubleValue": 0.0,
                "stringSetValue": [],
                "type": "STRINGSET",
                "integerValue": 0
            }
        ],
        "runningTasksCount": 0,
        "attributes": [
            {
                "name": "com.amazonaws.ecs.capability.privileged-container"
            },
            {
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
            },
            {
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
            },
            {
                "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
            },
            {
                "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
            },
            {
                "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
            }
```

```
            ],
            "versionInfo": {
                "agentVersion": "1.5.0",
                "agentHash": "b197edd",
                "dockerVersion": "DockerVersion: 1.7.1"
            }
        }
    ]
}
```

Amazon EC2 콘솔이나 aws ec2 describe-instances --instance-id ***instance_id*** 명령을 사용하여 인스턴스를 모니터링하는 데 사용할 수 있는 Amazon EC2 인스턴스 ID를 찾을 수도 있습니다.

태스크 정의 등록

Amazon ECS 클러스터에서 태스크를 실행하려면 먼저 태스크 정의를 등록해야 합니다. 태스크 정의는 그룹화된 컨테이너의 목록입니다. 다음 예제는 `nginx` 이미지를 사용하는 간단한 태스크 정의입니다. 사용 가능한 태스크 정의 파라미터에 대한 자세한 정보는 Amazon ECS 작업 정의 섹션을 참조하세요.

```
{
    "family": "nginx-task",
    "containerDefinitions": [
        {
            "name": "nginx",
            "image": "public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest",
            "cpu": 256,
            "memory": 512,
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "requiresCompatibilities": ["EC2"],
    "networkMode": "bridge"
}
```

위의 예시 JSON은 두 가지 방법으로 AWS CLI에 전달할 수 있습니다. 작업 정의 JSON을 파일로 저장하고 `--cli-input-json file://path_to_file.json` 옵션을 사용하여 전달할 수 있습니다. 또는 JSON에서 따옴표를 제거하고 명령줄에서 JSON 컨테이너 정의를 전달할 수 있습니다. 명령줄에서 컨테이너 정의를 전달하려는 경우 명령에 여러 버전의 작업 정의를 서로 연결하는 데 사용되는 `--family` 파라미터가 추가로 필요합니다.

컨테이너 정의에 JSON 파일을 사용하려면

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/nginx.json
```

register-task-definition은 등록 후 작업 정의의 설명을 반환합니다.

```
{
    "taskDefinition": {
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/nginx-
task:1",
        "family": "nginx-task",
        "revision": 1,
        "status": "ACTIVE",
        "containerDefinitions": [
            {
                "name": "nginx",
                "image": "public.ecr.aws/docker/library/nginx:latest",
                "cpu": 256,
                "memory": 512,
                "essential": true,
                "portMappings": [
                    {
                        "containerPort": 80,
                        "hostPort": 80,
                        "protocol": "tcp"
                    }
                ],
                "environment": [],
                "mountPoints": [],
                "volumesFrom": []
            }
        ],
        "volumes": [],
        "networkMode": "bridge",
        "compatibilities": [
            "EC2"
```

```
        ],
        "requiresCompatibilities": [
            "EC2"
        ]
    }
}
```

태스크 정의 나열

언제라도 list-task-definitions 명령을 사용하여 계정의 작업 정의를 나열할 수 있습니다. 이 명령은 create-service를 호출할 때 함께 사용할 수 있는 `family` 및 `revision` 값을 출력합니다.

```
aws ecs list-task-definitions
```

출력:

```
{
    "taskDefinitionArns": [
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep360:1",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep360:2",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/nginx-task:1",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:3",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:4",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:5",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:6"
    ]
}
```

서비스 생성

계정에 태스크를 등록하고 클러스터에 등록된 컨테이너 인스턴스를 시작한 후, 등록한 태스크 정의를 사용하여 원하는 수의 태스크를 동시에 실행하고 유지 관리하는 Amazon ECS 서비스를 생성할 수 있습니다. 이 예제에서는 기본 클러스터에 `nginx:1` 작업 정의의 단일 인스턴스를 배치합니다.

```
aws ecs create-service --cluster default --service-name nginx-service --task-
definition nginx-task:1 --desired-count 1
```

출력:

```
{
```

```
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:aws_account_id:service/MyCluster/nginx-
service",
        "serviceName": "nginx-service",
        "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/MyCluster",
        "taskDefinition": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/nginx-
task:1",
        "desiredCount": 1,
        "runningCount": 0,
        "pendingCount": 0,
        "launchType": "EC2",
        "status": "ACTIVE",
        "createdAt": "2025-01-13T10:45:00.000Z"
    }
}
```

서비스 나열

클러스터의 서비스를 나열합니다. 이전 섹션에서 생성한 서비스가 보일 것입니다. 이 명령에서 반환된 서비스 ID 또는 전체 ARN을 기록해 두었다가 나중에 서비스를 설명하는 데 사용할 수 있습니다.

```
aws ecs list-services --cluster MyCluster
```

출력:

```
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID"
    ]
}
```

서비스 설명

서비스에 대한 자세한 내용을 보려면 다음 명령을 사용하여 서비스를 설명을 표시합니다.

```
aws ecs describe-services --cluster MyCluster --services nginx-service
```

출력:

```
{
    "services": [
```

```
        {
            "serviceArn": "arn:aws:ecs:us-east-1:aws_account_id:service/MyCluster/
nginx-service",
            "serviceName": "nginx-service",
            "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/MyCluster",
            "taskDefinition": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/
nginx-task:1",
            "desiredCount": 1,
            "runningCount": 1,
            "pendingCount": 0,
            "launchType": "EC2",
            "status": "ACTIVE",
            "createdAt": "2025-01-13T10:45:00.000Z",
            "events": [
                {
                    "id": "abcd1234-5678-90ab-cdef-1234567890ab",
                    "createdAt": "2025-01-13T10:45:30.000Z",
                    "message": "(service nginx-service) has started 1 tasks: (task
 abcd1234-5678-90ab-cdef-1234567890ab)."
                }
            ]
        }
    ]
}
```

실행 중인 태스크 설명

서비스를 설명한 후 다음 명령을 실행하여 서비스의 일부로 실행 중인 태스크에 대한 자세한 정보를 가져옵니다.

```
aws ecs list-tasks --cluster MyCluster --service-name nginx-service
```

출력:

```
{
    "tasks": [
        {
            "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/MyCluster/
abcd1234-5678-90ab-cdef-1234567890ab",
            "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/MyCluster",
            "taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/
nginx-task:1",
```

```
            "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:container-
instance/MyCluster/abcd1234-5678-90ab-cdef-1234567890ab",
            "lastStatus": "RUNNING",
            "desiredStatus": "RUNNING",
            "containers": [
                {
                    "containerArn": "arn:aws:ecs:us-east-1:aws_account_id:container/
MyCluster/abcd1234-5678-90ab-cdef-1234567890ab/abcd1234-5678-90ab-cdef-1234567890ab",
                    "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/MyCluster/
abcd1234-5678-90ab-cdef-1234567890ab",
                    "name": "nginx",
                    "lastStatus": "RUNNING",
                    "networkBindings": [
                        {
                            "bindIP": "0.0.0.0",
                            "containerPort": 80,
                            "hostPort": 80,
                            "protocol": "tcp"
                        }
                    ]
                }
            ],
            "createdAt": "2025-01-13T10:45:00.000Z",
            "startedAt": "2025-01-13T10:45:30.000Z"
        }
    ]
}
```

웹 서버 테스트

웹 서버를 테스트하려면

1. 다음 명령을 실행하여 컨테이너 인스턴스의 퍼블릭 IP 주소를 검색합니다.

   ```
   aws ec2 describe-instances --instance-ids i-abcd1234 --query
    'Reservations[0].Instances[0].PublicIpAddress' --output text
   ```

   출력:

   ```
   203.0.113.25
   ```

2. IP 주소를 검색한 후 IP 주소로 다음 curl 명령을 실행합니다.

```
curl http://203.0.113.25
```

출력:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you can see this page, the nginx web server is successfully installed and
 working.</p>
...
</body>
</html>
```

nginx 시작 페이지에서 서비스가 성공적으로 실행되고 인터넷에서 액세스할 수 있는지 확인합니다.

리소스 정리

비용이 발생하지 않도록 하려면 이 자습서에서 생성한 리소스를 정리합니다.

리소스를 정리하려면

1. 원하는 태스크가 없도록 서비스를 업데이트한 다음, 서비스를 삭제합니다.

```
aws ecs update-service --cluster MyCluster --service nginx-service --desired-count
 0
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:123456789012:service/MyCluster/nginx-
service",
        "serviceName": "nginx-service",
        "desiredCount": 0,
        "runningCount": 1,
        "pendingCount": 0,
        "status": "ACTIVE"
```

```
    }
}
```

2. 실행 중인 태스크가 중지될 때까지 기다린 다음, 서비스를 삭제합니다.

```
aws ecs delete-service --cluster MyCluster --service nginx-service
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:123456789012:service/MyCluster/nginx-
service",
        "serviceName": "nginx-service",
        "status": "DRAINING"
    }
}
```

3. 생성된 컨테이너 인스턴스를 종료합니다.

```
aws ec2 terminate-instances --instance-ids i-abcd1234
{
    "TerminatingInstances": [
        {
            "InstanceId": "i-abcd1234",
            "CurrentState": {
                "Code": 32,
                "Name": "shutting-down"
            },
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
```

4. 생성한 보안 그룹 및 키 페어를 정리합니다.

```
aws ec2 delete-security-group --group-id sg-abcd1234
aws ec2 delete-key-pair --key-name ecs-tutorial-key
rm ecs-tutorial-key.pem
```

5. Amazon ECS 클러스터를 삭제합니다.

```
aws ecs delete-cluster --cluster MyCluster
```

```
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/MyCluster",
        "clusterName": "MyCluster",
        "status": "INACTIVE"
    }
}
```

CloudWatch Events 이벤트를 수신 대기하도록 Amazon ECS 구성

Amazon ECS 작업 이벤트를 수신 대기하고 이를 CloudWatch Logs 로그 스트림에 기록하는 간단한 Lambda 함수를 설정하는 방법을 알아봅니다.

필수 조건: 테스트 클러스터 설정

이벤트를 캡처할 실행 클러스터가 없는 경우 the section called “Fargate 시작 유형에 대한 클러스터 생성”의 단계를 따라 클러스터를 하나 생성합니다. 본 자습서의 말미에서는 이 클러스터에서 태스크를 실행하여 Lambda 함수가 올바로 구성되었는지 테스트합니다.

1단계: Lambda 함수 생성

이 절차에서는 Amazon ECS 이벤트 스트림 메시지의 대상으로 사용할 간단한 Lambda 함수를 생성합니다.

1. https://console.aws.amazon.com/lambda/에서 AWS Lambda 콘솔을 엽니다.
2. 함수 생성(Create function)을 선택합니다.
3. Author from scratch(새로 작성) 화면에서 다음과 같이 합니다.
    a. 이름(Name)에 값을 입력합니다.
    b. 런타임(Runtime)에서 Python 버전(예: Python 3.9)을 선택합니다.
    c. 역할(Role)에서 기본 Lambda 권한을 가진 새 역할 생성(Create a new role with basic Lambda permissions)을 선택합니다.
4. 함수 생성(Create function)을 선택합니다.
5. 함수 코드(Function code) 섹션에서 다음 예제와 일치하도록 샘플 코드를 수정합니다.

```
import json

def lambda_handler(event, context):
```

```
    if event["source"] != "aws.ecs":
        raise ValueError("Function only supports input from events with a source
type of: aws.ecs")

    print('Here is the event:')
    print(json.dumps(event))
```

다음은 Amazon ECS에서 전송하는 이벤트를 인쇄하는 간단한 Python 3.9 함수입니다. 모든 설정이 올바르게 구성되면 이 자습서가 끝날 때 이 Lambda 함수와 연결된 CloudWatch Logs 로그 스트림에 이벤트 세부 정보가 표시됩니다.

6. 저장을 선택합니다.

2단계: 이벤트 규칙 등록

다음에는 Amazon ECS 클러스터로부터 작업 이벤트를 캡처하는 CloudWatch Events 이벤트 규칙을 만듭니다. 이 규칙은 규칙이 정의된 계정의 모든 클러스터에서 전송하는 모든 이벤트를 캡처합니다. 작업 메시지 자체에 작업이 상주하는 클러스터와 같은 이벤트 소스에 대한 정보가 포함되며, 이 정보를 사용하여 프로그래밍 방식으로 이벤트를 필터링 및 정렬할 수 있습니다.

> **Note**
>
> AWS Management Console을 사용하여 이벤트 규칙을 만들 경우 콘솔이 Lambda 함수를 호출할 CloudWatch Events 권한을 부여하는 데 필요한 IAM 권한을 자동으로 추가합니다. AWS CLI를 사용하여 이벤트 규칙을 만드는 경우 이 권한을 명시적으로 부여해야 합니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge의 이벤트 및 Amazon EventBridge 이벤트 패턴을 참조하세요.

Lambda 함수로 이벤트를 라우팅하려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 이벤트, 규칙, 규칙 생성을 선택합니다.
3. 이벤트 소스(Event Source)에서 ECS를 이벤트 소스로 선택합니다. 기본적으로 규칙이 모든 Amazon ECS 그룹의 모든 Amazon ECS 이벤트에 적용됩니다. 또는 특정 이벤트 또는 특정 Amazon ECS 그룹을 선택할 수 있습니다.
4. 대상(Targets)에서 대상 추가(Add target)를 선택하고 대상 유형(Target type)에서 Lambda 함수(Lambda function)를 선택한 다음 사용자의 Lambda 함수를 선택합니다.

5. 세부 정보 구성(Configure details)을 선택합니다.
6. 규칙 정의(Rule definition)에 규칙 이름과 설명을 입력한 다음 규칙 생성(Create rule)을 선택합니다.

3단계: 태스크 정의 생성

작업 정의를 생성합니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 태스크 정의를 선택합니다.
3. 새 태스크 정의 생성(Create new Task Definition), JSON으로 새 수정 생성(Create new revision with JSON)을 선택합니다.
4. 다음 예제 태스크 정의를 복사하여 상자에 붙여 넣은 다음 저장(Save)을 선택합니다.

```
{
   "containerDefinitions": [
      {
         "command": [
            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</
title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
         ],
         "entryPoint": [
            "sh",
            "-c"
         ],
         "essential": true,
         "image": "public.ecr.aws/docker/library/httpd:2.4",
         "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
               "awslogs-group" : "/ecs/fargate-task-definition",
               "awslogs-region": "us-east-1",
               "awslogs-stream-prefix": "ecs"
            }
         },
         "name": "sample-fargate-app",
```

```
         "portMappings": [
            {
               "containerPort": 80,
               "hostPort": 80,
               "protocol": "tcp"
            }
         ]
      }
   ],
   "cpu": "256",
   "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family": "fargate-task-definition",
   "memory": "512",
   "networkMode": "awsvpc",
   "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
    },
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

5. 생성(Create)을 선택합니다.

4단계: 규칙 테스트

마지막으로 Amazon ECS 클러스터로부터 작업 이벤트를 캡처하는 CloudWatch Events 이벤트 규칙을 만듭니다. 이 규칙은 규칙이 정의된 계정의 모든 클러스터에서 전송하는 모든 이벤트를 캡처합니다. 작업 메시지 자체에 작업이 상주하는 클러스터와 같은 이벤트 소스에 대한 정보가 포함되며, 이 정보를 사용하여 프로그래밍 방식으로 이벤트를 필터링 및 정렬할 수 있습니다.

규칙을 테스트하려면

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. Task definitions(작업 정의)를 선택합니다.
3. console-sample-app-static을 선택한 다음 Deploy(배포), Run new task(새 작업 실행)를 선택합니다.
4. Cluster(클러스터)에서 기본값을 선택한 다음 Deploy(배포)를 선택합니다.
5. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.

6. 탐색 창에서 로그를 선택하고 Lambda 함수의 로그 그룹을 선택합니다(예: /aws/lambda/*my-function*).
7. 이벤트 데이터를 보려면 로그 스트림을 선택합니다.

Amazon ECS 작업 중지 이벤트에 대한 Amazon Simple Notification Service 알림 보내기

필수 컨테이너 중 하나가 종료되어 작업 실행이 중지된 작업 이벤트만 캡처하는 Amazon EventBridge 이벤트 규칙을 구성합니다. 이 이벤트는 특정 Amazon SNS 속성을 포함하는 작업 이벤트만 지정된 `stoppedReason` 주제로 전송합니다.

필수 조건: 테스트 클러스터 설정

이벤트를 캡처할 실행 클러스터가 없는 경우 Getting started with the console using Linux containers on AWS Fargate의 단계를 따라 클러스터를 하나 생성합니다. 본 자습서의 말미에서는 이 클러스터에서 태스크를 실행하여 Amazon SNS 주제와 EventBridge 이벤트 규칙이 올바로 구성되었는지 테스트합니다.

전제 조건: Amazon SNS 대한 권한 구성

EventBridge가 Amazon SNS 주제에 게시할 수 있도록 허용하려면 aws sns get-topic-attributes 및 aws sns set-topic-attributes 명령을 사용하세요.

권한을 추가하는 방법에 대한 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 Amazon SNS 권한을 참조하세요.

다음 권한을 추가합니다.

```
{
  "Sid": "PublishEventsToMyTopic",
  "Effect": "Allow",
  "Principal": {
     "Service": "events.amazonaws.com"
  },
  "Action": "sns: Publish",
  "Resource": "arn:aws:sns:region:account-id:TaskStoppedAlert",
}
```

1단계: Amazon SNS 주제 생성 및 구독

본 자습서를 위해 새 이벤트 규칙의 이벤트 대상으로 사용할 Amazon SNS 주제를 구성합니다.

Amazon SNS 주제 생성 및 구독 방법에 대한 자세한 정보는 Amazon Simple Notification Service 개발자 안내서의 Amazon SNS 시작하기를 참조하고 다음 표를 사용하여 선택할 옵션을 결정하세요.

옵션	값
유형	표준
명칭	TaskStoppedAlert
프로토콜	이메일
엔드포인트	현재 액세스 권한이 있는 이메일 주소

2단계: 이벤트 규칙 등록

다음에는 컨테이너가 중지된 작업에 대한 작업 중지 이벤트만 캡처하는 이벤트 규칙을 등록합니다.

Amazon SNS 주제 생성 및 구독 방법에 대한 자세한 정보는 Amazon EventBridge User Guide(Amazon EventBridge 사용 설명서)의 Create a rule in Amazon EventBridge(Amazon EventBridge에서 규칙 생성)를 참조하고 다음 표를 사용하여 선택할 옵션을 결정하세요.

옵션	값
규칙 타입	이벤트 패턴이 있는 규칙
이벤트 소스	AWS 이벤트 또는 EventBridge 파트너 이벤트
이벤트 패턴	사용자 정의 패턴(JSON 편집기)

이벤트 패턴	<pre>{
	  "source":[</pre>

옵션	값

	<pre>"aws.ecs"
	],
	"detail-type":[
	"ECS Task State Change"
	],
	"detail":{
	"lastStatus":[
	"STOPPED"
	],
	"stoppedReason":[
	"Essential container in task exited"
	]
	}
	}</pre>

대상 유형	AWS 서비스
대상	SNS 주제
주제	TaskStoppedAlert(1단계에서 생성한 주제)

3단계: 규칙 테스트

규칙이 시작된 직후 종료되는 태스크를 실행하여 규칙이 작동하고 있는지 확인합니다. 이벤트 규칙이 올바로 구성되었다면 몇 분 후에 이벤트 텍스트가 포함된 이메일 메시지를 수신할 것입니다. 규칙 요구 사항을 충족할 수 있는 기존 태스크 정의가 있는 경우 이를 사용하여 태스크를 실행합니다. 없는 경우 다음 단계에 따라 Fargate 태스크 정의를 등록하고 실행할 수 있습니다.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 작업 정의를 선택합니다.
3. 새 태스크 정의 생성(Create new task definition), JSON으로 새 태스크 정의 생성(Create new task definition with JSON)을 선택합니다.
4. JSON 편집기 상자에서 JSON 파일을 편집하고 다음을 편집기에 복사합니다.

```
{
   "containerDefinitions":[
      {
         "command":[
            "sh",
            "-c",
            "sleep 5"
         ],
         "essential":true,
         "image":"public.ecr.aws/amazonlinux/amazonlinux:latest",
         "name":"test-sleep"
      }
   ],
   "cpu":"256",
   "executionRoleArn":"arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family":"fargate-task-definition",
   "memory":"512",
   "networkMode":"awsvpc",
   "requiresCompatibilities":[
      "FARGATE"
   ]
}
```

5. 생성(Create)을 선택합니다.

콘솔에서 작업을 실행하려면 다음을 수행하세요.

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 클러스터 페이지에서 사전 조건에서 생성한 클러스터를 선택합니다.
3. 작업(Tasks) 탭에서 새 작업 실행(Run new task)을 선택합니다.
4. 애플리케이션 유형(Application type)에서 작업(Task)을 선택합니다.
5. 작업 정의에서 fargate-task-definition을 선택합니다.
6. 원하는 작업(Desired tasks)에 시작할 작업 수를 입력합니다.
7. 생성(Create)을 선택합니다.

여러 줄 또는 스택 추적 Amazon ECS 로그 메시지 연결

AWS for Fluent Bit 버전 2.22.0부터 여러 줄 필터가 포함됩니다. 여러 줄 필터를 사용하면 원래 하나의 컨텍스트에 속하지만 여러 레코드 또는 로그 라인으로 분할된 로그 메시지를 연결할 수 있습니다. 여러 줄 필터에 대한 자세한 내용은 Fluent Bit 문서를 참조하세요.

분할 로그 메시지의 일반적인 예는 다음과 같습니다.

- 스택 추적.
- 여러 줄에 로그를 인쇄하는 애플리케이션.
- 지정된 런타임 최대 버퍼 크기보다 길기 때문에 분할된 메시지를 로그합니다. GitHub의 예제를 따라 컨테이너 런타임으로 분할된 로그 메시지를 연결할 수 있습니다. FireLens 예제: 부분/분할 컨테이너 로그 연결.

필수 IAM 권한

컨테이너 에이전트가 Amazon ECR에서 컨테이너 이미지를 가져오고 컨테이너가 로그를 CloudWatch Logs로 라우팅하는 데 필요한 IAM 권한이 있는지 확인해야 합니다.

이러한 권한의 경우 다음 역할이 있어야 합니다.

- 태스크 IAM 역할.
- 작업 실행 IAM 역할.

JSON 정책 편집기를 사용하여 정책을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 창에서 정책을 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작을 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. 정책 편집기 섹션에서 JSON 옵션을 선택합니다.
5. 다음 JSON 정책 문서를 입력합니다.

```
{
```

```
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": [
   "logs:CreateLogStream",
   "logs:CreateLogGroup",
   "logs:PutLogEvents"
  ],
  "Resource": "*"
 }]
}
```

6. 다음을 선택합니다.

> **Note**
>
> 언제든지 시각적 편집기 옵션과 JSON 편집기 옵션 간에 전환할 수 있습니다. 그러나 변경을 적용하거나 시각적 편집기에서 다음을 선택한 경우 IAM은 시각적 편집기에 최적화되도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 정책 재구성을 참조하세요.

7. 검토 및 생성 페이지에서 생성하는 정책에 대한 정책 이름과 설명(선택 사항)을 입력합니다. 이 정책에 정의된 권한을 검토하여 정책이 부여한 권한을 확인합니다.
8. 정책 생성을 선택하고 새로운 정책을 저장합니다.

여러 줄 로그 설정을 사용할 시기 지정

기본 로그 설정과 함께 CloudWatch Logs 콘솔에 표시되는 예제 로그 코드 조각은 다음과 같습니다. log로 시작하는 줄을 보고 여러 줄의 필터가 필요한지 결정할 수 있습니다. 컨텍스트가 동일한 경우, 여러 줄 로그 설정을 사용할 수 있습니다. 이 예제에서는 컨텍스트가 'com.myproject.model.MyProject'입니다.

```
2022-09-20T15:47:56:595-05-00                              {"container_id":
 "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-
app", "source=": "stdout", "log": ": "     at com.myproject.modele.
(MyProject.badMethod.java:22)",
    {
      "container_id":  "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
      "container_name: ": "example-app",
      "source": "stdout",
```

```
      "log": ": "     at com.myproject.model.MyProject.badMethod(MyProject.java:22)",
      "ecs_cluster": "default",
      "ecs_task_arn": "arn:aws:region:123456789012:task/default/
b23c940d29ed4714971cba72cEXAMPLE",
      "ecs_task_definition": "firelense-example-multiline:3"
     }
```

```
2022-09-20T15:47:56:595-05-00                              {"container_id":
 "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-app", "stdout",
 "log": ": "     at com.myproject.modele.(MyProject.oneMoreMethod.java:18)",
    {
      "container_id":  "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
      "container_name: ": "example-app",
      "source": "stdout",
      "log": ": "     at
 com.myproject.model.MyProject.oneMoreMethod(MyProject.java:18)",
      "ecs_cluster": "default",
      "ecs_task_arn": "arn:aws:region:123456789012:task/default/
b23c940d29ed4714971cba72cEXAMPLE,
      "ecs_task_definition": "firelense-example-multiline:3"
     }
```

여러 줄 로그 설정을 사용한 이후에는 아래의 예시와 유사한 출력이 나옵니다.

```
2022-09-20T15:47:56:595-05-00                              {"container_id":
 "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-app",
 "stdout",...
    {
      "container_id":  "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
      "container_name: ": "example-app",
      "source": "stdout",
      "log:    "September 20, 2022 06:41:48 Exception in thread \"main\"
 java.lang.RuntimeException: Something has gone wrong, aborting!\n
    at com.myproject.module.MyProject.badMethod(MyProject.java:22)\n    at
    at com.myproject.model.MyProject.oneMoreMethod(MyProject.java:18)
 com.myproject.module.MyProject.main(MyProject.java:6)",
      "ecs_cluster": "default",
      "ecs_task_arn": "arn:aws:region:123456789012:task/default/
b23c940d29ed4714971cba72cEXAMPLE",
      "ecs_task_definition": "firelense-example-multiline:2"
     }
```

구문 분석 및 연결 옵션

줄 바꿈으로 인해 로그를 구문 분석하고 분할된 줄을 연결하려면, 다음 두 옵션 중 하나를 사용할 수 있습니다.

- 동일한 메시지에 속하는 줄을 구문 분석하고 연결하는 규칙이 포함된 고유한 구문 분석기 파일을 사용합니다.
- Fluent Bit 기본 제공 구문 분석을 사용합니다. Fluent Bit 기본 제공 구문 분석에서 지원하는 언어 목록은 Fluent Bit 설명서를 참조하세요.

다음 자습서에서는 각 사용 사례에 대한 단계를 안내합니다. 이 단계에서는 여러 줄을 연결하고 Amazon CloudWatch로 로그를 전송하는 방법을 보여줍니다. 로그에 대해 다른 대상을 지정할 수 있습니다.

예: 생성한 구문 분석기 사용

이 예에서는 다음 단계를 완료합니다.

1. Fluent Bit 컨테이너용 이미지를 빌드하고 업로드합니다.
2. 여러 줄 스택 추적을 실행, 실패 및 생성하는 데모 여러 줄 애플리케이션에 대한 이미지를 빌드하고 업로드합니다.
3. 태스크 정의를 생성하고 태스크를 실행합니다.
4. 로그를 보고 여러 줄에 걸쳐 있는 메시지가 연결된 것처럼 보이는지 확인합니다.

Fluent Bit 컨테이너용 이미지 빌드 및 업로드

이 이미지에는 정규식을 지정하는 구문 분석기 파일과 구문 분석기 파일을 참조하는 구성 파일이 포함됩니다.

1. `FluentBitDockerImage`라는 폴더를 생성합니다.
2. 폴더 내에 동일한 메시지에 속하는 줄을 구문 분석하고 연결하는 규칙이 포함된 구문 분석기 파일을 생성합니다.

   a. 구문 분석기 파일에 다음 내용을 붙여넣습니다.

```
[MULTILINE_PARSER]
```

```
    name          multiline-regex-test
    type          regex
    flush_timeout 1000
    #
    # Regex rules for multiline parsing
    # ---------------------------------
    #
    # configuration hints:
    #
    #  - first state always has the name: start_state
    #  - every field in the rule must be inside double quotes
    #
    # rules |   state name  | regex pattern                  | next state
    # ------|---------------|--------------------------------------------
    rule      "start_state"   "/(Dec \d+ \d+\:\d+\:\d+)(.*)/"  "cont"
    rule      "cont"          "/^\s+at.*/"                     "cont"
```

정규식 패턴을 사용자 지정할 때 정규식 편집기를 사용하여 표현식을 테스트하는 것이 좋습니다.

b. 파일을 `parsers_multiline.conf`(으)로 저장합니다.

3. `FluentBitDockerImage` 폴더 내에 이전 단계에서 생성한 구문 분석기 파일을 참조하는 사용자 정의 구성 파일을 생성합니다.

   사용자 정의 구성 파일에 대한 자세한 정보는 Amazon Elastic Container Service 개발자 안내서의 사용자 정의 구성 파일 지정을 참조하세요.

   a. 파일에 다음 내용을 붙여넣습니다.

```
[SERVICE]
    flush                1
    log_level            info
    parsers_file         /parsers_multiline.conf

[FILTER]
    name                 multiline
    match                *
    multiline.key_content log
    multiline.parser     multiline-regex-test
```

> **Note**
>
> 구문 분석기의 절대 경로를 사용해야 합니다.

b. 파일을 `extra.conf`(으)로 저장합니다.

4. `FluentBitDockerImage` 폴더 내에서 Fluent Bit 이미지와 생성한 구문 분석기 및 구성 파일로 Dockerfile을 생성합니다.

a. 파일에 다음 내용을 붙여넣습니다.

```
FROM public.ecr.aws/aws-observability/aws-for-fluent-bit:latest

ADD parsers_multiline.conf /parsers_multiline.conf
ADD extra.conf /extra.conf
```

b. 파일을 `Dockerfile`(으)로 저장합니다.

5. Dockerfile을 사용하여 구문 분석기 및 사용자 정의 구성 파일이 포함된 사용자 정의 Fluent Bit 이미지를 빌드합니다.

> **Note**
>
> 이 파일 경로는 FireLens에서 사용하므로 `/fluent-bit/etc/fluent-bit.conf`를 제외한 Docker 이미지의 아무 곳에나 구문 분석기 파일과 구성 파일을 배치할 수 있습니다.

a. 이미지를 빌드합니다. `docker build -t fluent-bit-multiline-image.`

여기서 `fluent-bit-multiline-image`는 이 예에서 이미지의 이름입니다.

b. 이미지가 올바르게 생성되었는지 확인합니다. `docker images –filter reference=fluent-bit-multiline-image`

성공하면 출력에 이미지와 `latest` 태그가 표시됩니다.

6. Amazon Elastic Container Registry에 사용자 정의 Fluent Bit 이미지를 업로드합니다.

a. 이미지를 저장할 Amazon ECR 리포지토리를 생성합니다. `aws ecr create-repository --repository-name fluent-bit-multiline-repo --region us-east-1`

여기서 `fluent-bit-multiline-repo`는 리포지토리의 이름이고 `us-east-1`은 이 예에서 리전입니다.

출력은 새 리포지토리의 세부 정보를 제공합니다.

b. 이전 출력의 `repositoryUri` 값으로 이미지에 태깅합니다. `docker tag` *`fluent-bit-multiline-image repositoryUri`*

예시: `docker tag fluent-bit-multiline-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

c. Docker 이미지를 실행하여 올바르게 실행되었는지 확인합니다. `docker images –filter reference=`*`repositoryUri`*

출력에서 리포지토리 이름이 fluent-bit-multiline-repo에서 `repositoryUri`로 변경됩니다.

d. `aws ecr get-login-password` 명령을 실행하고 인증하려는 레지스트리 ID를 지정하여 Amazon ECR에 인증합니다. `aws ecr get-login-password | docker login --username AWS --password-stdin` *`registry ID`*`.dkr.ecr.`*`region`*`.amazonaws.com`

예시: `ecr get-login-password | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com`

로그인 성공 메시지가 나타납니다.

e. 이미지를 Amazon ECR에 푸시합니다. `docker push` *`registry ID`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository name`*

예시: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

데모 여러 줄 애플리케이션용 이미지 빌드 및 업로드

이 이미지에는 애플리케이션을 실행하는 Python 스크립트 파일과 샘플 로그 파일이 포함됩니다.

태스크를 실행하면 애플리케이션이 실행을 시뮬레이션한 다음 실패하고 스택 추적을 생성합니다.

1. `multiline-app`이라는 폴더를 생성합니다. `mkdir multiline-app`
2. Python 스크립트 파일을 생성합니다.

   a. `multiline-app` 폴더 내에 파일을 생성하고 이름을 `main.py`로 지정합니다.

b. 파일에 다음 내용을 붙여넣습니다.

```
import os
import time
file1 = open('/test.log', 'r')
Lines = file1.readlines()

count = 0

for i in range(10):
    print("app running normally...")
    time.sleep(1)

# Strips the newline character
for line in Lines:
    count += 1
    print(line.rstrip())
print(count)
print("app terminated.")
```

c. `main.py` 파일을 저장합니다.

3. 샘플 로그 파일을 생성합니다.

a. `multiline-app` 폴더 내에 파일을 생성하고 이름을 `test.log`로 지정합니다.

b. 파일에 다음 내용을 붙여넣습니다.

```
single line...
Dec 14 06:41:08 Exception in thread "main" java.lang.RuntimeException:
 Something has gone wrong, aborting!
    at com.myproject.module.MyProject.badMethod(MyProject.java:22)
    at com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)
    at com.myproject.module.MyProject.anotherMethod(MyProject.java:14)
    at com.myproject.module.MyProject.someMethod(MyProject.java:10)
    at com.myproject.module.MyProject.main(MyProject.java:6)
another line...
```

c. `test.log` 파일을 저장합니다.

4. `multiline-app` 폴더 내에 Dockerfile을 생성합니다.

a. 파일에 다음 내용을 붙여넣습니다.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
ADD test.log /test.log

RUN yum upgrade -y && yum install -y python3

WORKDIR /usr/local/bin

COPY main.py .

CMD ["python3", "main.py"]
```

b. Dockerfile 파일을 저장합니다.

5. Dockerfile을 사용하여 이미지를 빌드합니다.

a. 이미지를 빌드합니다. `docker build -t multiline-app-image`

여기서 `multiline-app-image`는 이 예에서 이미지의 이름입니다.

b. 이미지가 올바르게 생성되었는지 확인합니다. `docker images –filter reference=multiline-app-image`

성공하면 출력에 이미지와 `latest` 태그가 표시됩니다.

6. Amazon Elastic 컨테이너 레지스트리로 이미지를 업로드합니다.

a. 이미지를 저장할 Amazon ECR 리포지토리를 생성합니다. `aws ecr create-repository --repository-name multiline-app-repo --region us-east-1`

여기서 `multiline-app-repo`는 리포지토리의 이름이고 `us-east-1`은 이 예에서 리전입니다.

출력은 새 리포지토리의 세부 정보를 제공합니다. 다음 단계에서 필요하므로 `repositoryUri` 값을 기록해 둡니다.

b. 이전 출력의 `repositoryUri` 값으로 이미지에 태깅합니다. `docker tag` *`multiline-app-image repositoryUri`*

예시: `docker tag multiline-app-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

c. Docker 이미지를 실행하여 올바르게 실행되었는지 확인합니다. `docker images –filter reference=`*`repositoryUri`*

출력에서 리포지토리 이름이 multiline-app-repo에서 repositoryUri 값으로 변경됩니다.

d. 이미지를 Amazon ECR에 푸시합니다. docker push *aws_account_id*.dkr.ecr.*region*.amazonaws.com/*repository name*

예시: docker push *xxxxxxxxxxxx*.dkr.ecr.*us-east-1*.amazonaws.com/multiline-app-repo

태스크 정의 생성 및 태스크 시작

1. 파일 이름이 multiline-task-definition.json인 태스크 정의 파일을 생성합니다.
2. multiline-task-definition.json 파일에 다음 내용을 붙여넣습니다.

```
{
    "family": "firelens-example-multiline",
    "taskRoleArn": "task role ARN",
    "executionRoleArn": "execution role ARN",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-
multiline-image:latest",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit",
                "options": {
                    "config-file-type": "file",
                    "config-file-value": "/extra.conf"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/multiline-app-
image:latest",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
```

```
                    "Name": "cloudwatch_logs",
                    "region": "us-east-1",
                    "log_group_name": "multiline-test/application",
                    "auto_create_group": "true",
                    "log_stream_prefix": "multiline-"
                }
            },
            "memoryReservation": 100
        }
    ],
    "requiresCompatibilities": ["FARGATE"],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512"
}
```

`multiline-task-definition.json` 태스크 정의에서 다음을 바꿉니다.

a. *`task role ARN`*

태스크 역할 ARN을 찾으려면 IAM 콘솔로 이동합니다. 역할(Roles)을 선택하고 생성한 `ecs-task-role-for-firelens` 태스크 역할을 찾습니다. 역할을 선택하고 요약(Summary) 섹션에 표시되는 ARN을 복사합니다.

b. *`execution role ARN`*

실행 역할 ARN을 찾으려면 IAM 콘솔로 이동합니다. 역할(Roles)을 선택하고 `ecsTaskExecutionRole` 역할을 찾습니다. 역할을 선택하고 요약(Summary) 섹션에 표시되는 ARN을 복사합니다.

c. *`aws_account_id`*

`aws_account_id`를 찾으려면 AWS Management Console에 로그인합니다. 오른쪽 상단에서 사용자 이름을 선택하고 계정 ID를 복사합니다.

d. *`us-east-1`*

필요한 경우 리전을 바꿉니다.

3. 태스크 정의 파일을 등록합니다. `aws ecs register-task-definition --cli-input-json file://multiline-task-definition.json --region` *`region`*
4. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.

5. 탐색 창에서 태스크 정의를 선택한 다음 위의 태스크 정의의 첫 번째 줄에서 이 제품군에 태스크 정의를 등록했기 때문에 `firelens-example-multiline` 제품군을 선택합니다.
6. 최신 버전을 선택합니다.
7. 배포, 작업 실행을 선택합니다.
8. 작업 실행 페이지의 클러스터에서 클러스터를 선택한 다음 네트워킹 아래의 서브넷에서 작업에 사용할 수 있는 서브넷을 선택합니다.
9. 생성(Create)을 선택합니다.

Amazon CloudWatch의 여러 줄 로그 메시지가 연결된 것으로 나타나는지 확인

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 로그를 확장하고 로그 그룹을 선택합니다.
3. `multiline-test/applicatio` 로그 그룹을 선택합니다.
4. 로그를 선택합니다. 메시지를 봅니다. 구문 분석기 파일의 규칙과 일치하는 줄은 연결되어 단일 메시지로 나타납니다.

   다음 로그 조각은 단일 Java 스택 추적 이벤트에 연결된 줄을 보여줍니다.

```
{
    "container_id": "xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "log": "Dec 14 06:41:08 Exception in thread \"main\"
 java.lang.RuntimeException: Something has gone wrong, aborting!\n
 at com.myproject.module.MyProject.badMethod(MyProject.java:22)\n    at
 com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)\n
 at com.myproject.module.MyProject.anotherMethod(MyProject.java:14)\n
 at com.myproject.module.MyProject.someMethod(MyProject.java:10)\n    at
 com.myproject.module.MyProject.main(MyProject.java:6)",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:2"
}
```

   다음 로그 조각은 여러 줄 로그 메시지를 연결하도록 구성되지 않은 Amazon ECS 컨테이너를 실행할 경우 한 줄로 동일한 메시지가 어떻게 나타나는지 보여줍니다.

```
{
```

```
    "log": "Dec 14 06:41:08 Exception in thread \"main\"
 java.lang.RuntimeException: Something has gone wrong, aborting!",
    "container_id": "xxxxxx-xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:3"
}
```

예: Fluent Bit 기본 제공 구문 분석 사용

이 예에서는 다음 단계를 완료합니다.

1. Fluent Bit 컨테이너용 이미지를 빌드하고 업로드합니다.
2. 여러 줄 스택 추적을 실행, 실패 및 생성하는 데모 여러 줄 애플리케이션에 대한 이미지를 빌드하고 업로드합니다.
3. 태스크 정의를 생성하고 태스크를 실행합니다.
4. 로그를 보고 여러 줄에 걸쳐 있는 메시지가 연결된 것처럼 보이는지 확인합니다.

Fluent Bit 컨테이너용 이미지 빌드 및 업로드

이 이미지에는 Fluent Bit 구문 분석기를 참조하는 구성 파일이 포함됩니다.

1. `FluentBitDockerImage`라는 폴더를 생성합니다.
2. `FluentBitDockerImage` 폴더 내에 Fluent Bit 기본 제공 구문 분석기 파일을 참조하는 사용자 정의 구성 파일을 생성합니다.

   사용자 정의 구성 파일에 대한 자세한 정보는 Amazon Elastic Container Service 개발자 안내서의 사용자 정의 구성 파일 지정을 참조하세요.

   a. 파일에 다음 내용을 붙여넣습니다.

```
[FILTER]
    name                  multiline
    match                 *
    multiline.key_content log
    multiline.parser      go
```

b. 파일을 `extra.conf`(으)로 저장합니다.

3. `FluentBitDockerImage` 폴더 내에서 Fluent Bit 이미지와 생성한 구문 분석기 및 구성 파일로 Dockerfile을 생성합니다.

   a. 파일에 다음 내용을 붙여넣습니다.

   ```
   FROM public.ecr.aws/aws-observability/aws-for-fluent-bit:latest
   ADD extra.conf /extra.conf
   ```

   b. 파일을 `Dockerfile`(으)로 저장합니다.

4. Dockerfile을 사용하여 사용자 정의 구성 파일이 포함된 사용자 정의 Fluent Bit 이미지를 빌드합니다.

   > **Note**
   >
   > 이 파일 경로는 FireLens에서 사용하므로 `/fluent-bit/etc/fluent-bit.conf`를 제외한 Docker 이미지의 아무 곳에나 구성 파일을 배치할 수 있습니다.

   a. 이미지를 빌드합니다. `docker build -t fluent-bit-multiline-image.`

      여기서 `fluent-bit-multiline-image`는 이 예에서 이미지의 이름입니다.

   b. 이미지가 올바르게 생성되었는지 확인합니다. `docker images –filter reference=fluent-bit-multiline-image`

      성공하면 출력에 이미지와 `latest` 태그가 표시됩니다.

5. Amazon Elastic Container Registry에 사용자 정의 Fluent Bit 이미지를 업로드합니다.

   a. 이미지를 저장할 Amazon ECR 리포지토리를 생성합니다. `aws ecr create-repository --repository-name fluent-bit-multiline-repo --region us-east-1`

      여기서 `fluent-bit-multiline-repo`는 리포지토리의 이름이고 `us-east-1`은 이 예에서 리전입니다.

      출력은 새 리포지토리의 세부 정보를 제공합니다.

   b. 이전 출력의 `repositoryUri` 값으로 이미지에 태깅합니다. `docker tag` *`fluent-bit-multiline-image repositoryUri`*

예시: `docker tag fluent-bit-multiline-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

c. Docker 이미지를 실행하여 올바르게 실행되었는지 확인합니다. `docker images –filter reference=`*`repositoryUri`*

출력에서 리포지토리 이름이 fluent-bit-multiline-repo에서 `repositoryUri`로 변경됩니다.

d. `aws ecr get-login-password` 명령을 실행하고 인증하려는 레지스트리 ID를 지정하여 Amazon ECR에 인증합니다. `aws ecr get-login-password | docker login --username AWS --password-stdin` *`registry ID`*`.dkr.ecr.`*`region`*`.amazonaws.com`

예시: `ecr get-login-password | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com`

로그인 성공 메시지가 나타납니다.

e. 이미지를 Amazon ECR에 푸시합니다. `docker push` *`registry ID`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository name`*

예시: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

데모 여러 줄 애플리케이션용 이미지 빌드 및 업로드

이 이미지에는 애플리케이션을 실행하는 Python 스크립트 파일과 샘플 로그 파일이 포함됩니다.

1. `multiline-app`이라는 폴더를 생성합니다. `mkdir multiline-app`
2. Python 스크립트 파일을 생성합니다.
   a. `multiline-app` 폴더 내에 파일을 생성하고 이름을 `main.py`로 지정합니다.
   b. 파일에 다음 내용을 붙여넣습니다.

```
import os
import time
file1 = open('/test.log', 'r')
Lines = file1.readlines()

count = 0
```

```
for i in range(10):
    print("app running normally...")
    time.sleep(1)

# Strips the newline character
for line in Lines:
    count += 1
    print(line.rstrip())
print(count)
print("app terminated.")
```

c. main.py 파일을 저장합니다.

3. 샘플 로그 파일을 생성합니다.

a. multiline-app 폴더 내에 파일을 생성하고 이름을 test.log로 지정합니다.

b. 파일에 다음 내용을 붙여넣습니다.

```
panic: my panic

goroutine 4 [running]:
panic(0x45cb40, 0x47ad70)
  /usr/local/go/src/runtime/panic.go:542 +0x46c fp=0xc42003f7b8 sp=0xc42003f710
 pc=0x422f7c
main.main.func1(0xc420024120)
  foo.go:6 +0x39 fp=0xc42003f7d8 sp=0xc42003f7b8 pc=0x451339
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003f7e0
 sp=0xc42003f7d8 pc=0x44b4d1
created by main.main
  foo.go:5 +0x58

goroutine 1 [chan receive]:
runtime.gopark(0x4739b8, 0xc420024178, 0x46fcd7, 0xc, 0xc420028e17, 0x3)
  /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc420053e30 sp=0xc420053e00
 pc=0x42503c
runtime.goparkunlock(0xc420024178, 0x46fcd7, 0xc, 0x1000f010040c217, 0x3)
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc420053e70 sp=0xc420053e30
 pc=0x42512e
runtime.chanrecv(0xc420024120, 0x0, 0xc420053f01, 0x4512d8)
  /usr/local/go/src/runtime/chan.go:506 +0x304 fp=0xc420053f20 sp=0xc420053e70
 pc=0x4046b4
runtime.chanrecv1(0xc420024120, 0x0)
```

```
  /usr/local/go/src/runtime/chan.go:388 +0x2b fp=0xc420053f50 sp=0xc420053f20
 pc=0x40439b
main.main()
  foo.go:9 +0x6f fp=0xc420053f80 sp=0xc420053f50 pc=0x4512ef
runtime.main()
  /usr/local/go/src/runtime/proc.go:185 +0x20d fp=0xc420053fe0 sp=0xc420053f80
 pc=0x424bad
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc420053fe8
 sp=0xc420053fe0 pc=0x44b4d1

goroutine 2 [force gc (idle)]:
runtime.gopark(0x4739b8, 0x4ad720, 0x47001e, 0xf, 0x14, 0x1)
  /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003e768 sp=0xc42003e738
 pc=0x42503c
runtime.goparkunlock(0x4ad720, 0x47001e, 0xf, 0xc420000114, 0x1)
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003e7a8 sp=0xc42003e768
 pc=0x42512e
runtime.forcegchelper()
  /usr/local/go/src/runtime/proc.go:238 +0xcc fp=0xc42003e7e0 sp=0xc42003e7a8
 pc=0x424e5c
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003e7e8
 sp=0xc42003e7e0 pc=0x44b4d1
created by runtime.init.4
  /usr/local/go/src/runtime/proc.go:227 +0x35

goroutine 3 [GC sweep wait]:
runtime.gopark(0x4739b8, 0x4ad7e0, 0x46fdd2, 0xd, 0x419914, 0x1)
  /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003ef60 sp=0xc42003ef30
 pc=0x42503c
runtime.goparkunlock(0x4ad7e0, 0x46fdd2, 0xd, 0x14, 0x1)
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003efa0 sp=0xc42003ef60
 pc=0x42512e
runtime.bgsweep(0xc42001e150)
  /usr/local/go/src/runtime/mgcsweep.go:52 +0xa3 fp=0xc42003efd8
 sp=0xc42003efa0 pc=0x419973
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003efe0
 sp=0xc42003efd8 pc=0x44b4d1
created by runtime.gcenable
  /usr/local/go/src/runtime/mgc.go:216 +0x58
one more line, no multiline
```

c. `test.log` 파일을 저장합니다.

4. `multiline-app` 폴더 내에 Dockerfile을 생성합니다.

a. 파일에 다음 내용을 붙여넣습니다.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
ADD test.log /test.log

RUN yum upgrade -y && yum install -y python3

WORKDIR /usr/local/bin

COPY main.py .

CMD ["python3", "main.py"]
```

b. `Dockerfile` 파일을 저장합니다.

5. Dockerfile을 사용하여 이미지를 빌드합니다.

a. 이미지를 빌드합니다. `docker build -t multiline-app-image`

여기서 `multiline-app-image`는 이 예에서 이미지의 이름입니다.

b. 이미지가 올바르게 생성되었는지 확인합니다. `docker images –filter reference=multiline-app-image`

성공하면 출력에 이미지와 `latest` 태그가 표시됩니다.

6. Amazon Elastic 컨테이너 레지스트리로 이미지를 업로드합니다.

a. 이미지를 저장할 Amazon ECR 리포지토리를 생성합니다. `aws ecr create-repository --repository-name multiline-app-repo --region us-east-1`

여기서 `multiline-app-repo`는 리포지토리의 이름이고 `us-east-1`은 이 예에서 리전입니다.

출력은 새 리포지토리의 세부 정보를 제공합니다. 다음 단계에서 필요하므로 `repositoryUri` 값을 기록해 둡니다.

b. 이전 출력의 `repositoryUri` 값으로 이미지에 태깅합니다. `docker tag` *`multiline-app-image repositoryUri`*

예시: `docker tag multiline-app-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

c. Docker 이미지를 실행하여 올바르게 실행되었는지 확인합니다. `docker images –filter reference=`*`repositoryUri`*

출력에서 리포지토리 이름이 `multiline-app-repo`에서 `repositoryUri` 값으로 변경됩니다.

d. 이미지를 Amazon ECR에 푸시합니다. `docker push` *`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository name`*

예시: `docker push` *`xxxxxxxxxxxx`*`.dkr.ecr.`*`us-east-1`*`.amazonaws.com/multiline-app-repo`

태스크 정의 생성 및 태스크 시작

1. 파일 이름이 `multiline-task-definition.json`인 태스크 정의 파일을 생성합니다.
2. `multiline-task-definition.json` 파일에 다음 내용을 붙여넣습니다.

```
{
    "family": "firelens-example-multiline",
    "taskRoleArn": "task role ARN,
    "executionRoleArn": "execution role ARN",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-
multiline-image:latest",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit",
                "options": {
                    "config-file-type": "file",
                    "config-file-value": "/extra.conf"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
```

```
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/multiline-app-
image:latest",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "cloudwatch_logs",
                    "region": "us-east-1",
                    "log_group_name": "multiline-test/application",
                    "auto_create_group": "true",
                    "log_stream_prefix": "multiline-"
                }
            },
            "memoryReservation": 100
        }
    ],
    "requiresCompatibilities": ["FARGATE"],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512"
}
```

multiline-task-definition.json 태스크 정의에서 다음을 바꿉니다.

a. *task role ARN*

   태스크 역할 ARN을 찾으려면 IAM 콘솔로 이동합니다. 역할(Roles)을 선택하고 생성한 ecs-task-role-for-firelens 태스크 역할을 찾습니다. 역할을 선택하고 요약(Summary) 섹션에 표시되는 ARN을 복사합니다.

b. *execution role ARN*

   실행 역할 ARN을 찾으려면 IAM 콘솔로 이동합니다. 역할(Roles)을 선택하고 ecsTaskExecutionRole 역할을 찾습니다. 역할을 선택하고 요약(Summary) 섹션에 표시되는 ARN을 복사합니다.

c. *aws_account_id*

   aws_account_id를 찾으려면 AWS Management Console에 로그인합니다. 오른쪽 상단에서 사용자 이름을 선택하고 계정 ID를 복사합니다.

d. *us-east-1*

   필요한 경우 리전을 바꿉니다.

3. 태스크 정의 파일을 등록합니다. `aws ecs register-task-definition --cli-input-json file://multiline-task-definition.json --region us-east-1`
4. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
5. 탐색 창에서 태스크 정의를 선택한 다음 위의 태스크 정의의 첫 번째 줄에서 이 제품군에 태스크 정의를 등록했기 때문에 `firelens-example-multiline` 제품군을 선택합니다.
6. 최신 버전을 선택합니다.
7. 배포, 작업 실행을 선택합니다.
8. 작업 실행 페이지의 클러스터에서 클러스터를 선택한 다음 네트워킹 아래의 서브넷에서 작업에 사용할 수 있는 서브넷을 선택합니다.
9. 생성(Create)을 선택합니다.

Amazon CloudWatch의 여러 줄 로그 메시지가 연결된 것으로 나타나는지 확인

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 로그를 확장하고 로그 그룹을 선택합니다.
3. `multiline-test/applicatio` 로그 그룹을 선택합니다.
4. 로그를 선택하고 메시지를 봅니다. 구문 분석기 파일의 규칙과 일치하는 줄은 연결되어 단일 메시지로 나타납니다.

   다음 로그 조각은 단일 이벤트로 연결된 Go 스택 추적을 보여줍니다.

```
{
    "log": "panic: my panic\n\ngoroutine 4 [running]:\npanic(0x45cb40,
 0x47ad70)\n  /usr/local/go/src/runtime/panic.go:542 +0x46c fp=0xc42003f7b8
 sp=0xc42003f710 pc=0x422f7c\nmain.main.func1(0xc420024120)\n  foo.go:6
 +0x39 fp=0xc42003f7d8 sp=0xc42003f7b8 pc=0x451339\nruntime.goexit()\n  /usr/
local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003f7e0 sp=0xc42003f7d8
 pc=0x44b4d1\ncreated by main.main\n  foo.go:5 +0x58\n\ngoroutine 1 [chan receive]:
\nruntime.gopark(0x4739b8, 0xc420024178, 0x46fcd7, 0xc, 0xc420028e17, 0x3)\n  /usr/
local/go/src/runtime/proc.go:280 +0x12c fp=0xc420053e30 sp=0xc420053e00 pc=0x42503c
\nruntime.goparkunlock(0xc420024178, 0x46fcd7, 0xc, 0x1000f010040c217, 0x3)\n
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc420053e70 sp=0xc420053e30
 pc=0x42512e\nruntime.chanrecv(0xc420024120, 0x0, 0xc420053f01, 0x4512d8)\n
  /usr/local/go/src/runtime/chan.go:506 +0x304 fp=0xc420053f20 sp=0xc420053e70
 pc=0x4046b4\nruntime.chanrecv1(0xc420024120, 0x0)\n  /usr/local/go/src/runtime/
chan.go:388 +0x2b fp=0xc420053f50 sp=0xc420053f20 pc=0x40439b\nmain.main()\n
 foo.go:9 +0x6f fp=0xc420053f80 sp=0xc420053f50 pc=0x4512ef\nruntime.main()\n
  /usr/local/go/src/runtime/proc.go:185 +0x20d fp=0xc420053fe0 sp=0xc420053f80
```

```
 pc=0x424bad\nruntime.goexit()\n  /usr/local/go/src/runtime/asm_amd64.s:2337
 +0x1 fp=0xc420053fe8 sp=0xc420053fe0 pc=0x44b4d1\n\ngoroutine 2 [force gc
 (idle)]:\nruntime.gopark(0x4739b8, 0x4ad720, 0x47001e, 0xf, 0x14, 0x1)\n  /
usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003e768 sp=0xc42003e738
 pc=0x42503c\nruntime.goparkunlock(0x4ad720, 0x47001e, 0xf, 0xc420000114, 0x1)\n
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003e7a8 sp=0xc42003e768
 pc=0x42512e\nruntime.forcegchelper()\n  /usr/local/go/src/runtime/proc.go:238
 +0xcc fp=0xc42003e7e0 sp=0xc42003e7a8 pc=0x424e5c\nruntime.goexit()\n  /usr/
local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003e7e8 sp=0xc42003e7e0
 pc=0x44b4d1\ncreated by runtime.init.4\n  /usr/local/go/src/runtime/proc.go:227
 +0x35\n\ngoroutine 3 [GC sweep wait]:\nruntime.gopark(0x4739b8, 0x4ad7e0,
 0x46fdd2, 0xd, 0x419914, 0x1)\n  /usr/local/go/src/runtime/proc.go:280 +0x12c
 fp=0xc42003ef60 sp=0xc42003ef30 pc=0x42503c\nruntime.goparkunlock(0x4ad7e0,
 0x46fdd2, 0xd, 0x14, 0x1)\n  /usr/local/go/src/runtime/proc.go:286 +0x5e
 fp=0xc42003efa0 sp=0xc42003ef60 pc=0x42512e\nruntime.bgsweep(0xc42001e150)\n
  /usr/local/go/src/runtime/mgcsweep.go:52 +0xa3 fp=0xc42003efd8 sp=0xc42003efa0
 pc=0x419973\nruntime.goexit()\n  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1
 fp=0xc42003efe0 sp=0xc42003efd8 pc=0x44b4d1\ncreated by runtime.gcenable\n  /usr/
local/go/src/runtime/mgc.go:216 +0x58",
    "container_id": "xxxxxx-xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:2"
}
```

다음 로그 조각은 여러 줄 로그 메시지를 연결하도록 구성되지 않은 ECS 컨테이너를 실행할 경우 동일한 이벤트가 어떻게 나타나는지 보여줍니다. 로그 필드에는 한 줄이 포함됩니다.

```
{
    "log": "panic: my panic",
    "container_id": "xxxxxx-xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:3"
```

> **Note**
>
> 로그가 표준 출력 대신 로그 파일로 이동하는 경우 필터 대신 Tail 입력 플러그 인에서 `multiline.parser` 및 `multiline.key_content` 구성 파라미터를 지정하는 것이 좋습니다.

Amazon ECS Windows 컨테이너에서 Fluent Bit 배포

Fluent Bit는 다양한 운영 체제에서 지원하는 빠르고 유연한 로그 프로세서 및 라우터입니다. Amazon CloudWatch Logs, Firehose Amazon S3 및 Amazon OpenSearch Service와 같은 다양한 AWS 대상으로 로그를 라우팅하는 데 사용할 수 있습니다. Fluent Bit는 Datadog, Splunk 및 사용자 지정 HTTP 서버와 같은 일반적인 파트너 솔루션을 지원합니다. Fluent Bit에 대한 자세한 내용은 Fluent Bit 웹 사이트를 참조하세요.

AWS for Fluent Bit 이미지는 고가용성을 위해 대부분 리전의 Amazon ECR 퍼블릭 갤러리와 Amazon ECR 리포지토리에 있는 Amazon ECR에 사용할 수 있습니다. 자세한 내용은 GitHub 웹 사이트의 aws-for-fluent-bit 섹션을 참조하세요.

이 자습서에서는 Amazon ECS에서 실행 중인 Windows 인스턴스에 Fluent Bit 컨테이너를 배포하여 중앙 집중식 로깅을 위해 Windows 작업에서 생성된 로그를 Amazon CloudWatch로 스트리밍하는 방법을 안내합니다.

이 자습서에서는 다음 접근 방식을 사용합니다.

- Fluent Bit는 대몬(daemon) 스케줄링 전략과 함께 서비스로 실행됩니다. 이 전략은 Fluent Bit의 단일 인스턴스가 항상 클러스터의 컨테이너 인스턴스에서 실행되도록 합니다.
  - 전달 입력 플러그인을 사용하여 포트 24224에서 수신 대기합니다.
  - 도커 런타임에서 노출된 포트를 사용하여 Fluent Bit로 로그를 전송할 수 있도록 포트 24224를 호스트에 노출합니다.
  - Fluent Bit가 지정된 대상으로 로그 레코드를 전송하도록 허용하는 구성이 있습니다.
- fluentd 로깅 드라이버를 사용하여 다른 모든 Amazon ECS 작업 컨테이너를 실행합니다. 자세한 내용은 도커 설명서 웹 사이트의 Fluentd logging driver(Fluentd 로깅 드라이버)를 참조하세요.
  - 도커 는 호스트 네임스페이스 내의 로컬 호스트에 있는 TCP 소켓 24224에 연결합니다.
  - Amazon ECS 에이전트는 클러스터 이름, 작업 정의 패밀리 이름, 작업 정의 개정 번호, 작업 ARN 및 컨테이너 이름을 포함하는 레이블을 컨테이너에 추가합니다. fluentd docker 로깅 드라이버의 label 옵션을 사용하여 동일한 정보가 로그 레코드에 추가됩니다. 자세한 내용은 도커 설명서 웹 사

이트의 labels, labels-regex, env, and env-regex(labels, labels-regex, env 및 env-regex)를 참조하세요.

- fluentd 로깅 드라이버의 `async` 옵션이 `true`로 설정되어 있기 때문에 Fluent Bit 컨테이너가 다시 시작되면 도커는 Fluent Bit 컨테이너가 다시 시작될 때까지 로그를 버퍼링합니다. fluentd-buffer-limit 옵션을 설정하여 버퍼 제한을 높일 수 있습니다. 자세한 내용은 도커 설명서 웹 사이트의 fluentd-buffer-limit을 참조하세요.

작업 흐름은 다음과 같습니다.

- Fluent Bit 컨테이너는 호스트에 노출된 포트 24224에서 시작하고 수신 대기합니다.
- Fluent Bit는 작업 정의에 지정된 작업 IAM 역할 자격 증명을 사용합니다.
- 동일한 인스턴스에서 실행되는 다른 작업은 fluentd docker 로깅 드라이버를 사용하여 포트 24224의 Fluent Bit 컨테이너에 연결합니다.
- 애플리케이션 컨테이너가 로그를 생성하면 도커 런타임은 해당 레코드에 태그를 지정하고, 레이블에 지정된 추가 메타데이터를 추가한 다음 호스트 네임스페이스의 포트 24224에 전달합니다.
- Fluent Bit는 포트 24224에서 로그 레코드를 수신하는데, 이 포트가 호스트 네임스페이스에 노출되기 때문입니다.
- Fluent Bit는 내부 처리를 수행하고 지정된 대로 로그를 라우팅합니다.

이 자습서에서는 다음을 수행하는 기본 CloudWatch Fluent Bit 구성을 사용합니다.

- 각 클러스터 및 작업 정의 패밀리에 대한 새 로그 그룹을 생성합니다.
- 새 작업이 시작될 때마다 위에 생성된 로그 그룹의 각 작업 컨테이너에 대한 새 로그 스트림을 생성합니다. 각 스트림에는 컨테이너가 속한 작업 ID가 표시됩니다.
- 클러스터 이름, 작업 ARN, 작업 컨테이너 이름, 작업 정의 패밀리, 작업 정의 개정 번호를 비롯한 추가 메타데이터를 각 로그 항목에 추가합니다.

  예를 들어 `container_1` 및 `container_2`가 포함된 `task_1`과 `container_3`이 포함된 `task_2`가 있는 경우 CloudWatch 로그 스트림은 다음과 같습니다.

  - `/aws/ecs/windows.ecs_task_1`

    `task-out.`*`TASK_ID`*`.container_1`

    `task-out.`*`TASK_ID`*`.container_2`
  - `/aws/ecs/windows.ecs_task_2`

```
task-out.TASK_ID.container_3
```

> **Note**
>
> 듀얼 스택 서비스 엔드포인트를 사용하면 AWS CLI, SDK 및 Amazon ECS API에서 IPv4 및 IPv6 모두를 통해 Amazon ECS와 상호 작용할 수 있습니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

단계

- 사전 조건
- 1단계: IAM 액세스 역할 생성
- 2단계: Amazon ECS Windows 컨테이너 인스턴스 생성
- 3단계: Fluent Bit 구성
- 4단계: 로그를 CloudWatch로 라우팅하는 Windows Fluent Bit 작업 정의 등록
- 5단계: 대몬(daemon) 스케줄링 전략을 사용하여 ecs-windows-fluent-bit 작업 정의를 Amazon ECS 서비스로 실행
- 6단계: 로그를 생성하는 Windows 작업 정의 등록
- 7단계: windows-app-task 작업 정의 실행
- 8단계: CloudWatch에서 로그 확인
- 9단계: 정리

사전 조건

이 자습서에서는 다음 사전 조건이 충족되었다고 가정합니다.

- 최신 버전의 AWS CLI가 설치 및 구성됩니다. 자세한 내용은 AWS CLI 최신 버전 설치 또는 업데이트를 참조하세요.
- `aws-for-fluent-bit` 컨테이너 이미지는 다음 Windows 운영 체제에서 사용할 수 있습니다.
  - Windows Server 2019 Core
  - Windows Server 2019 Full
  - Windows Server 2022 Core
  - Windows Server 2022 Full

- Amazon ECS 사용 설정의 단계가 완료되었습니다.
- 클러스터가 있습니다. 이 자습서에서 클러스터 이름은 FluentBit-cluster입니다.
- EC2 인스턴스를 실행할 퍼블릭 서브넷이 포함된 VPC가 있습니다. 기본 VPC를 사용할 수 있습니다. Amazon CloudWatch 엔드포인트가 서브넷에 도달할 수 있도록 하는 프라이빗 서브넷을 사용할 수도 있습니다. Amazon CloudWatch 엔드포인트에 대한 자세한 내용은 AWS 일반 참조의 Amazon CloudWatch endpoints and quotas를 참조하세요. VPC 생성을 위해 Amazon VPC 마법사를 사용하는 방법에 대한 자세한 내용은 the section called “Virtual Private Cloud 생성” 섹션을 참조하세요.

1단계: IAM 액세스 역할 생성

Amazon ECS IAM 역할을 생성합니다.

1. Amazon ECS 컨테이너 인스턴스 역할 "ecsInstanceRole"을 생성합니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 IAM 역할을 참조하세요.
2. 이름이 `fluentTaskRole`인 Fluent Bit 작업에 대한 IAM 역할을 생성합니다. 자세한 내용은 the section called “태스크 IAM 역할” 섹션을 참조하세요.

   이 IAM 역할에 부여된 IAM 권한은 작업 컨테이너에서 수임합니다. Fluent Bit에서 CloudWatch로 로그를 전송하도록 허용하려면 작업 IAM 역할에 다음 권한을 연결해야 합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogStream",
            "logs:CreateLogGroup",
            "logs:DescribeLogStreams",
            "logs:PutLogEvents"
        ],
        "Resource": "*"
    }
    ]
}
```

3. 책을 역할에 연결합니다.

a. `fluent-bit-policy.json` 파일에 위 내용을 저장합니다.

b. 다음 명령을 실행하여 인라인 정책을 `fluentTaskRole` IAM 역할에 연결합니다.

```
aws iam put-role-policy --role-name fluentTaskRole --policy-name
 fluentTaskPolicy --policy-document file://fluent-bit-policy.json
```

2단계: Amazon ECS Windows 컨테이너 인스턴스 생성

Amazon ECS Windows 컨테이너 인스턴스를 생성합니다.

Amazon ECS 인스턴스 생성

1. `aws ssm get-parameters` 명령을 사용하여 VPC를 호스팅하는 리전에 대한 AMI ID를 검색합니다. 자세한 내용은 Amazon ECS 최적화 AMI 메타데이터 검색을 참조하세요.
2. Amazon EC2 콘솔을 사용하여 인스턴스를 시작합니다.

   a. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.

   b. 탐색 모음에서 사용할 리전을 선택합니다.

   c. EC2 대시보드에서 인스턴스 시작(Launch Instance)을 선택합니다.

   d. Name(이름)에 고유한 이름을 입력합니다.

   e. Application and OS Images (Amazon Machine Image)(애플리케이션 및 OS 이미지(Amazon Machine Image))에서 첫 번째 단계에서 검색한 AMI를 선택합니다.

   f. 인스턴스 유형에서 `t3.xlarge`를 선택합니다.

   g. Key pair (login)(키 페어(로그인))에서 키 페어를 선택합니다.

   h. Network settings(네트워크 설정)의 Security group(보안 그룹)에서 기존 보안 그룹을 선택하거나 새 보안 그룹을 생성합니다.

   i. Network settings(네트워크 설정)의 Auto-assign Public IP(퍼블릭 IP 자동 할당)에서 Enable(활성화)을 선택합니다.

   j. Advanced details(고급 세부 정보)의 IAM instance profile(IAM 인스턴스 프로파일)에서 ecsInstanceRole을 선택합니다.

   k. 다음 사용자 데이터로 Amazon ECS 컨테이너 인스턴스를 구성합니다. Advanced Details(고급 세부 정보)에서 다음 스크립트를 User data(사용자 데이터) 필드에 붙여 넣고 *`cluster_name`*을 클러스터의 이름으로 바꿉니다.

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster cluster-name -EnableTaskENI -EnableTaskIAMRole -
LoggingDrivers '["awslogs","fluentd"]'
</powershell>
```

l. 준비가 되었으면 승인 필드를 선택한 다음 인스턴스 시작(Launch Instances)을 선택합니다.

m. 확인 페이지에서 인스턴스가 실행 중인지 확인할 수 있습니다. 인스턴스 보기를 선택하여 확인 페이지를 닫고 콘솔로 돌아갑니다.

3단계: Fluent Bit 구성

AWS에서 제공하는 다음 기본 구성을 사용하여 빠르게 시작할 수 있습니다.

- Fluent Bit 공식 설명서의 Amazon CloudWatch용 Fluent Bit 플러그인을 기반으로 하는 Amazon CloudWatch

또는 AWS에서 제공하는 다른 기본 구성을 사용할 수 있습니다. 자세한 내용은 Github 웹 사이트의 `aws-for-fluent-bit`에서 Windows 이미지용 엔트리포인트 재정의를 참조하세요.

기본 Amazon CloudWatch Fluent Bit 구성은 아래와 같습니다.

다음 변수를 바꿉니다.

- *region*을 Amazon CloudWatch Logs를 전송하려는 리전으로 바꿉니다.

```
[SERVICE]
    Flush               5
    Log_Level           info
    Daemon              off

[INPUT]
    Name                forward
    Listen              0.0.0.0
    Port                24224
    Buffer_Chunk_Size   1M
    Buffer_Max_Size     6M
    Tag_Prefix          ecs.
```

```
# Amazon ECS agent adds the following log keys as labels to the docker container.
# We would use fluentd logging driver to add these to log record while sending it to
 Fluent Bit.
[FILTER]
    Name                modify
    Match               ecs.*
    Rename              com.amazonaws.ecs.cluster ecs_cluster
    Rename              com.amazonaws.ecs.container-name ecs_container_name
    Rename              com.amazonaws.ecs.task-arn ecs_task_arn
    Rename              com.amazonaws.ecs.task-definition-family
 ecs_task_definition_family
    Rename              com.amazonaws.ecs.task-definition-version
 ecs_task_definition_version

[FILTER]
    Name                rewrite_tag
    Match               ecs.*
    Rule                $ecs_task_arn ^([a-z-:0-9]+)/([a-zA-Z0-9-_]+)/([a-z0-9]+)$
 out.$3.$ecs_container_name false
    Emitter_Name        re_emitted

[OUTPUT]
    Name                cloudwatch_logs
    Match               out.*
    region              region
    log_group_name      fallback-group
    log_group_template  /aws/ecs/$ecs_cluster.$ecs_task_definition_family
    log_stream_prefix   task-
    auto_create_group   On
```

Fluent Bit에 들어오는 모든 로그에는 지정한 태그가 있으며, 태그를 지정하지 않은 경우에는 자동으로 생성됩니다. 태그를 사용하여 다양한 로그를 각기 다른 대상으로 라우팅할 수 있습니다. 자세한 내용은 Fluent Bit 공식 설명서의 Tag(태그)를 참조하세요.

위에서 설명한 Fluent Bit 구성에는 다음과 같은 속성이 있습니다.

- 전달 입력 플러그인은 TCP 포트 24224에서 들어오는 트래픽을 수신 대기합니다.
- 해당 포트에서 수신된 각 로그 항목에는 전달 입력 플러그인이 `ecs.` 문자열을 레코드 접두사로 수정하는 태그가 있습니다.
- Fluent Bit 내부 파이프라인은 Match 정규식을 사용하여 필터를 수정하기 위해 로그 항목을 라우팅합니다. 이 필터는 로그 레코드 JSON의 키를 Fluent Bit가 사용할 수 있는 형식으로 바꿉니다.

- 그러면 수정된 로그 항목이 rewrite_tag 필터에서 사용됩니다. 이 필터는 로그 레코드의 태그를 out.*TASK_ID*.*CONTAINER_NAME* 형식으로 변경합니다.
- 새 태그는 CloudWatch 출력 플러그인의 `log_group_template` 및 `log_stream_prefix` 옵션을 사용하여 앞서 설명한 것과 같이 로그 그룹과 스트림을 생성하는 출력 cloudwatch_logs 플러그인으로 라우팅됩니다. 자세한 내용은 Fluent Bit 공식 설명서의 Configuration parameters(구성 파라미터)를 참조하세요.

4단계: 로그를 CloudWatch로 라우팅하는 Windows Fluent Bit 작업 정의 등록

로그를 CloudWatch로 라우팅하는 Windows Fluent Bit 작업 정의를 등록합니다.

> **Note**
>
> 이 작업 정의는 Fluent Bit 컨테이너 포트 24224를 호스트 포트 24224에 노출시킵니다. 외부로부터의 액세스를 방지하기 위해 EC2 인스턴스 보안 그룹에서 이 포트가 열려 있지 않은지 확인합니다.

태스크 정의를 등록하려면

1. 다음 콘텐츠를 가진 `fluent-bit.json`이라는 파일을 생성합니다:

   다음 변수를 바꿉니다.

   - task-iam-role을 작업 IAM 역할의 Amazon 리소스 이름(ARN)으로 바꿉니다.
   - region을 작업이 실행되는 리전으로 바꿉니다.

```
{
  "family": "ecs-windows-fluent-bit",
  "taskRoleArn": "task-iam-role",
  "containerDefinitions": [
    {
      "name": "fluent-bit",
      "image": "public.ecr.aws/aws-observability/aws-for-fluent-
bit:windowsservercore-latest",
      "cpu": 512,
      "portMappings": [
```

```
        {
          "hostPort": 24224,
          "containerPort": 24224,
          "protocol": "tcp"
        }
      ],
      "entryPoint": [
        "Powershell",
        "-Command"
      ],
      "command": [
        "C:\\entrypoint.ps1 -ConfigFile C:\\ecs_windows_forward_daemon\
\cloudwatch.conf"
      ],
      "environment": [
        {
          "name": "AWS_REGION",
          "value": "region"
        }
      ],
      "memory": 512,
      "essential": true,
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/fluent-bit-logs",
          "awslogs-region": "region",
          "awslogs-stream-prefix": "flb",
          "awslogs-create-group": "true"
        }
      }
    }
  ],
  "memory": "512",
  "cpu": "512"
}
```

2. 다음 명령을 실행하여 작업 정의를 등록합니다.

```
aws ecs register-task-definition --cli-input-json file://fluent-bit.json --
region region
```

`list-task-definitions` 명령을 실행하여 계정의 작업 정의를 나열할 수 있습니다. 출력에는 `run-task` 또는 `start-task`에 사용할 수 있는 패밀리 및 리비전 값이 표시됩니다.

5단계: 대몬(daemon) 스케줄링 전략을 사용하여 **ecs-windows-fluent-bit** 작업 정의를 Amazon ECS 서비스로 실행

계정에 대한 작업 정의를 등록한 후 클러스터에서 작업을 실행할 수 있습니다. 이 자습서에서는 `FluentBit-cluster` 클러스터에서 `ecs-windows-fluent-bit:1` 작업 정의의 인스턴스 1개를 실행합니다. Fluent Bit의 단일 인스턴스가 항상 각 컨테이너 인스턴스에서 실행되도록 하는 대몬(daemon) 스케줄링 전략을 사용하는 서비스에서 작업을 실행합니다.

태스크를 실행하려면

1. 다음 명령을 실행하여 `ecs-windows-fluent-bit:1` 작업 정의(이전 단계에서 등록됨)를 서비스로 시작합니다.

   > **Note**
   >
   > 이 작업 정의는 `awslogs` 로깅 드라이버를 사용하며, 컨테이너 인스턴스에는 필요한 권한이 있어야 합니다.

   다음 변수를 바꿉니다.

   - region을 서비스가 실행되는 리전으로 바꿉니다.

   ```
   aws ecs create-service \
       --cluster FluentBit-cluster \
       --service-name FluentBitForwardDaemonService \
       --task-definition ecs-windows-fluent-bit:1 \
       --launch-type EC2 \
       --scheduling-strategy DAEMON \
       --region region
   ```

2. 다음 명령을 실행하여 작업을 나열합니다.

   다음 변수를 바꿉니다.

- region을 서비스 작업이 실행되는 리전으로 바꿉니다.

```
aws ecs list-tasks --cluster FluentBit-cluster --region region
```

6단계: 로그를 생성하는 Windows 작업 정의 등록

로그를 생성하는 작업 정의를 등록합니다. 이 작업 정의는 1초마다 stdout에 증분 숫자를 기록하는 Windows 컨테이너 이미지를 배포합니다.

작업 정의는 Fluent Bit 플러그인이 수신 대기하는 포트 24224에 연결되는 fluentd 로깅 드라이버를 사용합니다. Amazon ECS 에이전트는 클러스터 이름, 작업 ARN, 작업 정의 패밀리 이름, 작업 정의 개정 번호 및 작업 컨테이너 이름을 포함하는 태그를 사용하여 각 Amazon ECS 컨테이너에 레이블을 추가합니다. 이러한 키-값 레이블은 Fluent Bit에 전달됩니다.

> **Note**
>
> 이 작업은 default 네트워크 모드를 사용합니다. 하지만 작업에서 awsvpc 네트워크 모드를 사용할 수도 있습니다.

태스크 정의를 등록하려면

1. 다음 콘텐츠를 가진 windows-app-task.json이라는 파일을 생성합니다:

```
{
  "family": "windows-app-task",
  "containerDefinitions": [
    {
      "name": "sample-container",
      "image": "mcr.microsoft.com/windows/servercore:ltsc2019",
      "cpu": 512,
      "memory": 512,
      "essential": true,
      "entryPoint": [
        "Powershell",
        "-Command"
      ],
      "command": [
```

```
        "$count=1;while(1) { Write-Host $count; sleep 1; $count=$count+1;}"
      ],
      "logConfiguration": {
        "logDriver": "fluentd",
        "options": {
          "fluentd-address": "localhost:24224",
          "tag": "{{ index .ContainerLabels \"com.amazonaws.ecs.task-definition-
family\" }}",
          "fluentd-async": "true",
          "labels": "com.amazonaws.ecs.cluster,com.amazonaws.ecs.container-
name,com.amazonaws.ecs.task-arn,com.amazonaws.ecs.task-definition-
family,com.amazonaws.ecs.task-definition-version"
        }
      }
    }
  ],
  "memory": "512",
  "cpu": "512"
}
```

2. 다음 명령을 실행하여 작업 정의를 등록합니다.

   다음 변수를 바꿉니다.

   - region을 작업이 실행되는 리전으로 바꿉니다.

   ```
   aws ecs register-task-definition --cli-input-json file://windows-app-task.json --
   region region
   ```

   `list-task-definitions` 명령을 실행하여 계정의 작업 정의를 나열할 수 있습니다. 출력에는 `run-task` 또는 `start-task`에 사용할 수 있는 패밀리 및 리비전 값이 표시됩니다.

7단계: `windows-app-task` 작업 정의 실행

`windows-app-task` 작업 정의를 등록한 후 `FluentBit-cluster` 클러스터에서 실행합니다.

태스크를 실행하려면

1. 이전 단계에서 등록한 `windows-app-task:1` 작업 정의를 실행합니다.

   다음 변수를 바꿉니다.

- region을 작업이 실행되는 리전으로 바꿉니다.

```
aws ecs run-task --cluster FluentBit-cluster --task-definition windows-app-task:1
 --count 2 --region region
```

2. 다음 명령을 실행하여 작업을 나열합니다.

```
aws ecs list-tasks --cluster FluentBit-cluster
```

8단계: CloudWatch에서 로그 확인

Fluent Bit 설정을 확인하려면 CloudWatch 콘솔에서 다음 로그 그룹을 확인합니다.

- `/ecs/fluent-bit-logs` - 컨테이너 인스턴스에서 실행 중인 Fluent Bit 대몬(daemon) 컨테이너에 해당하는 로그 그룹입니다.
- `/aws/ecs/FluentBit-cluster.windows-app-task` - `FluentBit-cluster` 클러스터 내 `windows-app-task` 작업 정의 패밀리에 대해 실행된 모든 작업에 해당하는 로그 그룹입니다.

  `task-out.FIRST_TASK_ID.sample-container` - 이 로그 스트림에는 sample-container 작업 컨테이너에 있는 작업의 첫 번째 인스턴스에서 생성된 모든 로그가 포함됩니다.

  `task-out.SECOND_TASK_ID.sample-container` - 이 로그 스트림에는 sample-container 작업 컨테이너에 있는 작업의 두 번째 인스턴스에서 생성된 모든 로그가 포함됩니다.

`task-out.TASK_ID.sample-container` 로그 스트림에는 다음과 유사한 필드가 있습니다.

```
{
    "source": "stdout",
    "ecs_task_arn": "arn:aws:ecs:region:0123456789012:task/FluentBit-
cluster/13EXAMPLE",
    "container_name": "/ecs-windows-app-task-1-sample-container-cEXAMPLE",
    "ecs_cluster": "FluentBit-cluster",
    "ecs_container_name": "sample-container",
    "ecs_task_definition_version": "1",
    "container_id": "61f5e6EXAMPLE",
    "log": "10",
    "ecs_task_definition_family": "windows-app-task"
```

```
}
```

Fluent Bit 설정 확인

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 로그 그룹을 선택합니다. 컨테이너에 Fluent Bit를 배포한 리전에 있는지 확인합니다.

   해당 AWS 리전의 로그 그룹 목록에서 다음을 확인해야 합니다.

   - `/ecs/fluent-bit-logs`
   - `/aws/ecs/FluentBit-cluster.windows-app-task`

   이러한 로그 그룹을 보면 Fluent Bit 설정이 확인됩니다.

9단계: 정리

이 자습서를 완료한 후에는 사용하지 않는 리소스에 요금이 발생하지 않도록 연결된 리소스를 정리합니다.

자습서 리소스를 정리하려면

1. `windows-simple-task` 작업과 `ecs-fluent-bit` 작업을 중지합니다. 자세한 내용은 the section called “작업 중지” 섹션을 참조하세요.
2. 다음 명령을 실행하여 `/ecs/fluent-bit-logs` 로그 그룹을 삭제합니다. 로그 그룹 삭제에 대한 자세한 내용은 AWS Command Line Interface 참조의 delete-log-group을 참조하세요.

   ```
   aws logs delete-log-group --log-group-name /ecs/fluent-bit-logs
   aws logs delete-log-group --log-group-name /aws/ecs/FluentBit-cluster.windows-app-task
   ```

3. 다음 명령을 실행하여 인스턴스를 종료합니다.

   ```
   aws ec2 terminate-instances --instance-ids instance-id
   ```

4. 다음 명령을 실행하여 IAM 역할을 삭제합니다.

   ```
   aws iam delete-role --role-name ecsInstanceRole
   aws iam delete-role --role-name fluentTaskRole
   ```

5. 다음 명령을 실행하여 Amazon ECS 클러스터를 삭제합니다.

```
aws ecs delete-cluster --cluster FluentBit-cluster
```

Amazon ECS에서 EC2 Linux 컨테이너에 대해 gMSA 사용

Amazon ECS는 그룹 관리형 서비스 계정(gMSA)이라는 특수한 종류의 서비스 계정을 통해 EC2의 Linux 컨테이너에 대한 Active Directory 인증을 지원합니다.

.NET Core 애플리케이션과 같은 Linux 기반 네트워크 애플리케이션은 Active Directory를 사용하여 사용자와 서비스 간의 인증 및 권한 부여 관리를 용이하게 할 수 있습니다. Active Directory와 통합되고 도메인 가입된 서버에서 실행되는 애플리케이션을 설계하여 이 기능을 사용할 수 있습니다. 하지만 Linux 컨테이너는 도메인에 가입할 수 없으므로 gMSA로 실행할 Linux 컨테이너를 구성해야 합니다.

gMSA로 실행되는 Linux 컨테이너는 컨테이너의 호스트 Amazon EC2 인스턴스에서 실행되는 `credentials-fetcher` 대몬(daemon)을 사용합니다. 즉, 대몬(daemon)은 Active Directory 도메인 컨트롤러에서 gMSA 보안 인증을 검색한 다음 이러한 보안 인증을 컨테이너 인스턴스로 전송합니다. 서비스 계정에 대한 자세한 내용은 Microsoft Learn 웹 사이트에서 Create gMSAs for Windows containers를 참조하세요.

고려 사항

Linux 컨테이너에 gMSA를 사용하기 전에 다음 사항을 고려하세요.

- 컨테이너가 EC2에서 실행되는 경우 Windows 컨테이너와 Linux 컨테이너에 gMSA를 사용할 수 있습니다. Fargate의 Linux 컨테이너에서 gMSA를 사용하는 방법에 대한 자세한 내용은 Fargate의 Linux 컨테이너에서 gMSA 사용 섹션을 참조하세요.
- 사전 조건을 완료하려면 도메인에 가입된 Windows 컴퓨터가 필요할 수 있습니다. 예를 들어, PowerShell을 사용하여 Active Directory에서 gMSA를 생성하려면 도메인에 가입된 Windows 컴퓨터가 필요할 수 있습니다. RSAT Active Director PowerShell 도구는 Windows에서만 사용할 수 있습니다. 자세한 내용은 Installing the Active Directory administration tools를 참조하세요.
- 도메인 없는 gMSA 및 각 인스턴스를 단일 도메인에 가입 중에서 선택할 수 있습니다. 도메인 없는 gMSA를 사용하면 컨테이너 인스턴스는 도메인에 가입되지 않고, 인스턴스의 다른 애플리케이션은 보안 인증을 사용하여 도메인에 액세스할 수 없고, 다른 도메인에 가입하는 작업은 동일한 인스턴스에서 실행될 수 있습니다.

  그런 다음 CredSpec 및 선택적으로 도메인 없는 gMSA에 대한 Active Directory 사용자 보안 인증으로 데이터 스토리지를 선택합니다.

Amazon ECS는 Active Directory 보안 인증 사양 파일(CredSpec)을 사용합니다. 이 파일은 gMSA 계정 컨텍스트를 컨테이너에 전파하는 데 사용되는 gMSA 메타데이터를 포함합니다. CredSpec 파일을 생성하고 다음 표의 컨테이너 인스턴스의 운영 체제별 CredSpec 스토리지 옵션 중 하나에 저장합니다. 도메인 없는 방법을 사용하려면 CredSpec 파일의 선택적 섹션에서 다음 표의 컨테이너 인스턴스의 운영 체제별 domainless user credentials스토리지 옵션 중 하나에 보안 인증을 지정할 수 있습니다.

스토리지 위치	Linux	Windows
Amazon Simple Storage Service(S3)	CredSpec	CredSpec
AWS Secrets Manager	도메인 없는 사용자 보안 인증	도메인 없는 사용자 보안 인증
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec, 도메인 없는 사용자 보안 인증
로컬 파일	N/A	CredSpec

사전 조건

Amazon ECS에서 Linux 컨테이너 기능에 gMSA를 사용하려면 먼저 다음 사항을 충족해야 합니다.

- 컨테이너에서 액세스할 리소스를 포함하는 Active Directory 도메인을 설정합니다. Amazon ECS는 다음 설정을 지원합니다.
  - AWS Directory Service Active Directory. AWS Directory Service는 Amazon EC2에 호스트되는 AWS 관리형 Active Directory입니다. 자세한 내용은 AWS Directory Service 관리 안내서의 AWS 관리형 Microsoft AD 시작하기를 참조하세요.
  - 온프레미스 Active Directory. Amazon ECS Linux 컨테이너 인스턴스가 도메인에 가입할 수 있도록 해야 합니다. 자세한 내용은 AWS Direct Connect 섹션을 참조하세요.
- Active Directory에 기존 gMSA 계정이 있습니다. 자세한 내용은 Amazon ECS에서 EC2 Linux 컨테이너에 대해 gMSA 사용 섹션을 참조하세요.
- Amazon ECS Linux 컨테이너 인스턴스에 `credentials-fetcher` 대몬(daemon)을 설치하여 실행하고 있습니다. 또한 Active Directory에 인증하기 위해 `credentials-fetcher` 대몬(daemon)에 초기 보안 인증 세트를 추가했습니다.

> **Note**
>
> `credentials-fetcher` 대몬(daemon)은 Amazon Linux 2023 및 Fedora 37 이상에서만 사용할 수 있습니다. Amazon Linux 2에서는 대몬(daemon)을 사용할 수 없습니다. 자세한 내용은 GitHub의 aws/credentials-fetcher를 참조하세요.

- `credentials-fetcher` 대몬(daemon)을 Active Directory에 인증하기 위해 보안 인증을 설정합니다. 보안 인증은 gMSA 계정에 액세스할 수 있는 Active Directory 보안 그룹의 멤버여야 합니다. 인스턴스를 도메인에 가입시킬지, 아니면 도메인 없는 gMSA를 사용할지 결정합니다.에 에는 여러 옵션이 있습니다.
- 필요한 IAM 권한을 추가했습니다. 필요한 권한은 초기 보안 인증 및 보안 인증 사양 저장을 위해 선택하는 방법에 따라 달라집니다.
  - 초기 보안 인증에 도메인 없는 gMSA를 사용하는 경우 AWS Secrets Manager에 대한 IAM 권한이 작업 실행 역할에 필요합니다.
  - 보안 인증 사양을 SSM Parameter Store에 저장하는 경우 Amazon EC2 Systems Manager Parameter Store에 대한 IAM 권한이 작업 실행 역할에 필요합니다.
  - 보안 인증 사양을 Amazon S3에 저장하는 경우에는 Amazon Simple Storage Service에 대한 IAM 권한이 작업 실행 역할에 필요합니다.

Amazon ECS에서 gMSA 지원 Linux 컨테이너 설정

인프라 준비

다음 단계는 고려 사항과 한 번 수행하는 설정입니다. 이 단계를 완료하면 컨테이너 인스턴스 생성을 자동화하여 이 구성을 재사용할 수 있습니다.

초기 보안 인증을 제공하는 방법을 결정하고 재사용 가능한 EC2 시작 템플릿에 EC2 사용자 데이터를 구성하여 `credentials-fetcher` 대몬(daemon)을 설치합니다.

1. 인스턴스를 도메인에 가입시킬지, 아니면 도메인 없는 gMSA를 사용할지 결정합니다.
   - EC2 인스턴스를 Active Directory 도메인에 가입

- 사용자 데이터로 인스턴스 가입

  EC2 시작 템플릿의 EC2 사용자 데이터에 Active Directory 도메인에 가입하는 단계를 추가합니다. 여러 Amazon EC2 Auto Scaling 그룹에서 동일한 시작 템플릿을 사용할 수 있습니다.

  Fedora Docs의 Joining an Active Directory or FreeIPA domain 단계를 사용할 수 있습니다.

- 도메인 없는 gMSA를 위한 Active Directory 사용자 만들기

  `credentials-fetcher` 대몬(daemon)에는 도메인 없는 gMSA라는 기능이 있습니다. 이 기능에는 도메인이 필요하지만 EC2 인스턴스를 도메인에 가입시킬 필요는 없습니다. 도메인 없는 gMSA를 사용하면 컨테이너 인스턴스는 도메인에 가입되지 않고, 인스턴스의 다른 애플리케이션은 보안 인증을 사용하여 도메인에 액세스할 수 없고, 다른 도메인에 가입하는 작업은 동일한 인스턴스에서 실행될 수 있습니다. 대신 AWS Secrets Manager에서 CredSpec 파일에 보안 암호의 이름을 입력합니다. 보안 암호에는 사용자 이름, 암호 및 로그인할 도메인이 포함되어야 합니다.

  이 기능은 Linux 및 Windows 컨테이너에서 지원되며 사용할 수 있습니다.

  이 기능은 gMSA support for non-domain-joined container hosts 기능과 유사합니다. Windows 기능에 대한 자세한 내용은 Microsoft Learn 웹 사이트의 gMSA architecture and improvements를 참조하세요.

  a. Active Directory 도메인에서 사용자를 만듭니다. Active Directory의 사용자는 작업에서 사용하는 gMSA 서비스 계정에 액세스할 수 있는 권한이 있어야 합니다.

  b. Active Directory에서 사용자를 만든 후 AWS Secrets Manager에서 비밀을 생성합니다. 자세한 내용은 AWS Secrets Manager 보안 암호 생성을 참조하세요.

  c. 사용자의 사용자 이름, 암호 및 도메인을 각각 `username`, `password` 및 `domainName`이라는 JSON 키-값 쌍에 입력합니다.

  ```
  {"username":"username","password":"passw0rd", "domainName":"example.com"}
  ```

  d. 서비스 계정의 CredSpec 파일에 구성을 추가합니다. 추가 `HostAccountConfig`에는 Secrets Manager에 있는 보안 암호의 Amazon 리소스 이름(ARN)이 포함됩니다.

Windows에서 PluginGUID는 다음 예제 코드 조각의 GUID와 일치해야 합니다. Linux에서는 PluginGUID가 무시됩니다. 예제에서 MySecret은 보안 암호의 Amazon 리소스 이름 (ARN)으로 바꿉니다.

```
    "ActiveDirectoryConfig": {
        "HostAccountConfig": {
            "PortableCcgVersion": "1",
            "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
            "PluginInput": {
                "CredentialArn": "arn:aws:secretsmanager:aws-
region:111122223333:secret:MySecret"
            }
        }
```

e. 도메인 없는 gMSA 기능을 사용하려면 작업 실행 역할에 추가 권한이 필요합니다. (선택 사항) 도메인 없는 gMSA 보안 암호 단계를 따릅니다.

2. 인스턴스 구성 및 **credentials-fetcher** 대몬(daemon) 설치

EC2 시작 템플릿에서 사용자 데이터 스크립트를 사용하여 credentials-fetcher 대몬(daemon)을 설치할 수 있습니다. 다음 예제에서는 두 가지 유형의 사용자 데이터인 cloud-config YAML 또는 bash 스크립트를 보여줍니다. 이러한 예제는 Amazon Linux 2023(AL2023)용입니다. MyCluster를 이러한 인스턴스를 조인할 Amazon ECS 클러스터의 이름으로 바꿉니다.

- **cloud-config** YAML

```
Content-Type: text/cloud-config
package_reboot_if_required: true
packages:
  # prerequisites
  - dotnet
  - realmd
  - oddjob
  - oddjob-mkhomedir
  - sssd
  - adcli
  - krb5-workstation
  - samba-common-tools
  # https://github.com/aws/credentials-fetcher gMSA credentials management for
 containers
  - credentials-fetcher
write_files:
```

```
# configure the ECS Agent to join your cluster.
# replace MyCluster with the name of your cluster.
- path: /etc/ecs/ecs.config
  owner: root:root
  permissions: '0644'
  content: |
    ECS_CLUSTER=MyCluster
    ECS_GMSA_SUPPORTED=true
runcmd:
# start the credentials-fetcher daemon and if it succeeded, make it start after
 every reboot
- "systemctl start credentials-fetcher"
- "systemctl is-active credentials-fetcher && systemctl enable credentials-
fetcher"
```

- bash 스크립트

  bash 스크립트에 더 익숙하고 /etc/ecs/ecs.config에 쓸 변수가 여러 개인 경우 다음 heredoc 형식을 사용합니다. 이 형식은 cat으로 시작하는 라인과 EOF 사이의 모든 항목을 구성 파일에 작성합니다.

```
#!/usr/bin/env bash
set -euxo pipefail

# prerequisites
timeout 30 dnf install -y dotnet realmd oddjob oddjob-mkhomedir sssd adcli
 krb5-workstation samba-common-tools
# install https://github.com/aws/credentials-fetcher gMSA credentials
 management for containers
timeout 30 dnf install -y credentials-fetcher

# start credentials-fetcher
systemctl start credentials-fetcher
systemctl is-active credentials-fetcher && systemctl enable credentials-fetcher

cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_GMSA_SUPPORTED=true
EOF
```

credentials-fetcher 대몬(daemon)에 대한 선택적 구성 변수를 /etc/ecs/ecs.config에서 설정할 수 있습니다. 위 예제와 유사한 YAML 블록 또는 heredoc의 사용자 데이터에 변수를 설정하는 것이 좋습니다. 이렇게 하면 파일을 여러 번 편집하여 발생할 수 있는 부분 구성 문제를 방지할 수 있습니다. ECS 에이전트 구성에 대한 자세한 내용은 GitHub의 Amazon ECS Container Agent를 참조하세요.

- 또는, 소켓을 다른 위치로 이동하도록 credentials-fetcher 대몬(daemon) 구성을 변경하는 경우 CREDENTIALS_FETCHER_HOST 변수를 사용할 수 있습니다.

권한 및 보안 암호 설정

각 애플리케이션과 각 작업 정의에 대해 다음 단계를 한 번 수행합니다. 최소 권한을 부여하는 모범 사례를 따르고 정책에서 사용되는 권한의 범위를 좁히는 것이 좋습니다. 이렇게 하면 각 작업에서 필요한 보안 암호만 읽을 수 있습니다.

1. (선택 사항) 도메인 없는 gMSA 보안 암호

   인스턴스가 도메인에 가입되지 않은 도메인 없는 방법을 사용하는 경우 이 단계를 따릅니다.

   작업 실행 IAM 역할에 다음 권한을 인라인 정책으로 추가해야 합니다. 이렇게 하면 credentials-fetcher 대몬(daemon)이 Secrets Manager 보안 암호에 액세스할 수 있게 됩니다. MySecret 예제를 Resource 목록에 있는 보안 암호의 Amazon 리소스 이름(ARN)으로 바꿉니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                   "arn:aws:ssm:arn:aws: 111122223333:secret:MySecret"
            ]
        }
    ]
```

```
}
```

> **Note**
>
> 자체 KMS 키를 사용하여 보안 암호를 암호화하는 경우 이 역할에 필요한 권한을 추가하고 이 역할을 AWS KMS 키 정책에 추가해야 합니다.

2. SSM Parameter Store 또는 S3를 사용하여 CredSpec를 저장할지 결정

   Amazon ECS에서는 작업 정의의 `credentialSpecs` 필드에서 파일 경로를 참조하는 다음과 같은 방법을 지원합니다.

   인스턴스를 단일 도메인에 가입시키는 경우 문자열에서 ARN 시작 부분에 `credentialspec:` 접두사를 사용합니다. 도메인 없는 gMSA를 사용하는 경우에는 `credentialspecdomainless:`를 사용합니다.

   CredSpec에 대한 자세한 내용은 보안 인증 사양 파일 섹션을 참조하세요.

   - Amazon S3 버킷

     보안 인증 사양을 Amazon S3 버킷에 추가합니다. 그런 다음 작업 정의의 `credentialSpecs` 필드에서 Amazon S3 버킷의 Amazon 리소스 이름(ARN)을 참조합니다.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:s3:::${BucketName}/
${ObjectName}"
            ],
            ...
        }
    ],
    ...
}
```

또한 작업에 Amazon S3 버킷에 대한 액세스 권한을 부여하려면 Amazon ECS 작업 실행 IAM 역할에 다음 권한을 인라인 정책으로 추가합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/{object}"
            ]
        }
    ]
}
```

- SSM Parameter Store 파라미터

  보안 인증 사양을 SSM Parameter Store 파라미터에 추가합니다. 그런 다음 작업 정의의 `credentialSpecs` 필드에서 SSM Parameter Store 파라미터의 Amazon 리소스 이름(ARN)을 참조합니다.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:ssm:aws-
region:111122223333:parameter/parameter_name"
            ],
            ...
```

```
        }
    ],
    ...
}
```

또한 작업에 SSM Parameter Store 파라미터에 대한 액세스 권한을 부여하려면 Amazon ECS 작업 실행 IAM 역할에 다음 권한을 인라인 정책으로 추가합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
            "arn:aws:ssm:arn:aws::111122223333:parameter/parameter_name"
            ]
        }
    ]
}
```

보안 인증 사양 파일

Amazon ECS는 Active Directory 보안 인증 사양 파일(CredSpec)을 사용합니다. 이 파일은 gMSA 계정 컨텍스트를 Linux 컨테이너에 전파하는 데 사용되는 gMSA 메타데이터를 포함합니다. CredSpec를 생성하여 작업 정의의 `credentialSpecs` 필드에서 참조합니다. CredSpec 파일에는 보안 암호가 포함되어 있지 않습니다.

다음은 예 CredSpec 파일입니다.

```
{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
```

```
        "MachineAccountName": "WebApp01",
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "example.com",
        "DnsName": "example.com",
        "NetBiosName": "example"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "example.com"
            }
        ],
        "HostAccountConfig": {
            "PortableCcgVersion": "1",
            "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
            "PluginInput": {
                "CredentialArn": "arn:aws:secretsmanager:aws-
region:111122223333:secret:MySecret"
            }
        }
    }
}
```

CredSpec 생성

도메인에 가입된 Windows 컴퓨터에서 CredSpec PowerShell 모듈을 사용하여 CredSpec를 생성합니다. Microsoft Learn 웹 사이트에서 Create a credential spec의 단계를 따릅니다.

Fargate의 Linux 컨테이너에서 gMSA 사용

Amazon ECS는 그룹 관리형 서비스 계정(gMSA)이라는 특수한 종류의 서비스 계정을 통해 Fargate의 Linux 컨테이너에 대한 Active Directory 인증을 지원합니다.

.NET Core 애플리케이션과 같은 Linux 기반 네트워크 애플리케이션은 Active Directory를 사용하여 사용자와 서비스 간의 인증 및 권한 부여 관리를 용이하게 할 수 있습니다. Active Directory와 통합되고 도메인 가입된 서버에서 실행되는 애플리케이션을 설계하여 이 기능을 사용할 수 있습니다. 하지만 Linux 컨테이너는 도메인에 가입할 수 없으므로 gMSA로 실행할 Linux 컨테이너를 구성해야 합니다.

고려 사항

Fargate의 Linux 컨테이너에 gMSA를 사용하기 전에 다음을 고려합니다.

- 플랫폼 버전 1.4 이상을 실행해야 합니다.
- 사전 조건을 완료하려면 도메인에 가입된 Windows 컴퓨터가 필요할 수 있습니다. 예를 들어, PowerShell을 사용하여 Active Directory에서 gMSA를 생성하려면 도메인에 가입된 Windows 컴퓨터가 필요할 수 있습니다. RSAT Active Director PowerShell 도구는 Windows에서만 사용할 수 있습니다. 자세한 내용은 Installing the Active Directory administration tools를 참조하세요.
- 도메인이 없는 gMSA를 사용해야 합니다.

  Amazon ECS는 Active Directory 보안 인증 사양 파일(CredSpec)을 사용합니다. 이 파일은 gMSA 계정 컨텍스트를 컨테이너에 전파하는 데 사용되는 gMSA 메타데이터를 포함합니다. CredSpec 파일을 생성하여 Amazon S3 버킷에 저장합니다.
- 작업은 하나의 Active Directory만 지원할 수 있습니다.

사전 조건

Amazon ECS에서 Linux 컨테이너 기능에 gMSA를 사용하려면 먼저 다음 사항을 충족해야 합니다.

- 컨테이너에서 액세스할 리소스를 포함하는 Active Directory 도메인을 설정합니다. Amazon ECS는 다음 설정을 지원합니다.
  - AWS Directory Service Active Directory. AWS Directory Service는 Amazon EC2에 호스트되는 AWS 관리형 Active Directory입니다. 자세한 내용은 AWS Directory Service 관리 안내서의 AWS 관리형 Microsoft AD 시작하기를 참조하세요.
  - 온프레미스 Active Directory. Amazon ECS Linux 컨테이너 인스턴스가 도메인에 가입할 수 있도록 해야 합니다. 자세한 내용은 AWS Direct Connect 섹션을 참조하세요.
- Active Directory에 기존 gMSA 계정이 있고 gMSA 서비스 계정에 액세스할 권한이 있는 사용자가 있습니다. 자세한 내용은 도메인 없는 gMSA를 위한 Active Directory 사용자 만들기 섹션을 참조하세요.
- Amazon S3 버킷이 있습니다. 자세한 내용은 Amazon S3 사용자 안내서의 버킷 생성을 참조하세요.

Amazon ECS에서 gMSA 지원 Linux 컨테이너 설정

인프라 준비

다음 단계는 고려 사항과 한 번 수행하는 설정입니다.

- 도메인 없는 gMSA를 위한 Active Directory 사용자 만들기

  도메인이 없는 gMSA를 사용하는 경우 컨테이너는 도메인에 참여하지 않습니다. 컨테이너에서 실행되는 다른 애플리케이션은 자격 증명을 사용하여 도메인에 액세스할 수 없습니다. 다른 도메인을 사용하는 작업은 동일한 컨테이너에서 실행할 수 있습니다. AWS Secrets Manager에서 CredSpec 파일에 보안 암호의 이름을 제공합니다. 보안 암호에는 사용자 이름, 암호 및 로그인할 도메인이 포함되어야 합니다.

  이 기능은 gMSA support for non-domain-joined container hosts 기능과 유사합니다. Windows 기능에 대한 자세한 내용은 Microsoft Learn 웹 사이트의 gMSA architecture and improvements를 참조하세요.

  a. Active Directory 도메인에서 사용자를 구성합니다. Active Directory의 사용자는 작업에서 사용하는 gMSA 서비스 계정에 액세스할 수 있는 권한이 있어야 합니다.

  b. Active Directory 도메인 이름을 확인할 수 있는 VPC와 서브넷이 있습니다. Active Directory 서비스 이름을 가리키는 도메인 이름을 사용하여 DHCP 옵션을 통해 VPC를 구성합니다. VPC에 대한 DHCP 옵션 구성 방법에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 Work with DHCP option sets를 참조하세요.

  c. AWS Secrets Manager에서 암호를 생성합니다.

  d. 자격 증명 사양 파일을 생성합니다.

권한 및 보안 암호 설정

각 애플리케이션과 각 작업 정의에 대해 다음 단계를 한 번 수행합니다. 최소 권한을 부여하는 모범 사례를 따르고 정책에서 사용되는 권한의 범위를 좁히는 것이 좋습니다. 이렇게 하면 각 작업에서 필요한 보안 암호만 읽을 수 있습니다.

1. Active Directory 도메인에서 사용자를 만듭니다. Active Directory의 사용자는 작업에서 사용하는 gMSA 서비스 계정에 액세스할 수 있는 권한이 있어야 합니다.
2. Active Directory 사용자를 만든 후 AWS Secrets Manager에서 보안 암호를 생성합니다. 자세한 내용은 AWS Secrets Manager 보안 암호 생성을 참조하세요.
3. 사용자의 사용자 이름, 암호 및 도메인을 각각 `username`, `password` 및 `domainName`이라는 JSON 키-값 쌍에 입력합니다.

```
{"username":"username","password":"passw0rd", "domainName":"example.com"}
```

4. 작업 실행 IAM 역할에 다음 권한을 인라인 정책으로 추가해야 합니다. 이렇게 하면 `credentials-fetcher` 대몬(daemon)이 Secrets Manager 보안 암호에 액세스할 수 있게 됩니다. `MySecret` 예제를 `Resource` 목록에 있는 보안 암호의 Amazon 리소스 이름(ARN)으로 바꿉니다.

   JSON

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "secretsmanager:GetSecretValue"
               ],
               "Resource": [
               "arn:aws:secretsmanager:arn:aws::111122223333:secret:MySecret"
               ]
           }
       ]
   }
   ```

   > **Note**
   >
   > 자체 KMS 키를 사용하여 보안 암호를 암호화하는 경우 이 역할에 필요한 권한을 추가하고 이 역할을 AWS KMS 키 정책에 추가해야 합니다.

5. 보안 인증 사양을 Amazon S3 버킷에 추가합니다. 그런 다음 작업 정의의 `credentialSpecs` 필드에서 Amazon S3 버킷의 Amazon 리소스 이름(ARN)을 참조합니다.

   ```
   {
       "family": "",
       "executionRoleArn": "",
       "containerDefinitions": [
           {
               "name": "",
               ...
               "credentialSpecs": [
                   "credentialspecdomainless:arn:aws:s3:::${BucketName}/${ObjectName}"
   ```

```
            ],
            ...
        }
    ],
    ...
}
```

또한 작업에 Amazon S3 버킷에 대한 액세스 권한을 부여하려면 Amazon ECS 작업 실행 IAM 역할에 다음 권한을 인라인 정책으로 추가합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListObjects"
            ],
            "Resource": [
                "arn:aws:s3:::{bucket_name}",
                "arn:aws:s3:::{bucket_name}/{object}"
            ]
        }
    ]
}
```

보안 인증 사양 파일

Amazon ECS는 Active Directory 보안 인증 사양 파일(CredSpec)을 사용합니다. 이 파일은 gMSA 계정 컨텍스트를 Linux 컨테이너에 전파하는 데 사용되는 gMSA 메타데이터를 포함합니다. CredSpec를 생성하여 작업 정의의 `credentialSpecs` 필드에서 참조합니다. CredSpec 파일에는 보안 암호가 포함되어 있지 않습니다.

다음은 예 CredSpec 파일입니다.

```
{
```

```
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
        "MachineAccountName": "WebApp01",
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "example.com",
        "DnsName": "example.com",
        "NetBiosName": "example"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "example.com"
            }
        ],
        "HostAccountConfig": {
            "PortableCcgVersion": "1",
            "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
            "PluginInput": {
                "CredentialArn": "arn:aws:secretsmanager:aws-
region:111122223333:secret:MySecret"
            }
        }
    }
}
```

CredSpec 생성 및 Amazon S3에 업로드

도메인에 가입된 Windows 컴퓨터에서 CredSpec PowerShell 모듈을 사용하여 CredSpec를 생성합니다. Microsoft Learn 웹 사이트에서 Create a credential spec의 단계를 따릅니다.

자격 증명 사양 파일을 생성한 후 Amazon S3 버킷에 업로드합니다. AWS CLI 명령을 실행 중인 컴퓨터나 환경에 CredSpec 파일을 복사합니다.

다음 AWS CLI 명령을 실행하여 Amazon S3에 CredSpec를 업로드합니다. *amzn-s3-demo-bucket*을 Amazon S3 버킷 이름으로 바꿉니다. 파일을 모든 버킷과 위치에 개체로 저장할 수 있지만 작업 실행 역할에 연결하는 정책에서 해당 버킷과 위치에 대한 액세스를 허용해야 합니다.

PowerShell의 경우 다음 명령을 사용합니다.

```
$ Write-S3Object -BucketName "amzn-s3-demo-bucket" -Key "ecs-domainless-gmsa-credspec"
 -File "gmsa-cred-spec.json"
```

다음 AWS CLI 명령은 sh 및 호환 쉘에서 사용되는 백슬래시 연속 문자를 사용합니다.

```
$ aws s3 cp gmsa-cred-spec.json \
s3://amzn-s3-demo-bucket/ecs-domainless-gmsa-credspec
```

AWS CLI를 사용하여 도메인이 없는 gMSA로 Amazon ECS Windows 컨테이너 사용

다음 자습서에서는 AWS CLI를 사용하여 Active Directory에 액세스할 수 있는 보안 인증이 있는 Windows 컨테이너를 실행하는 Amazon ECS 작업을 생성하는 방법을 보여 줍니다 도메인 없는 gMSA를 사용하면 컨테이너 인스턴스는 도메인에 가입되지 않고, 인스턴스의 다른 애플리케이션은 보안 인증을 사용하여 도메인에 액세스할 수 없고, 다른 도메인에 가입하는 작업은 동일한 인스턴스에서 실행될 수 있습니다.

주제

- 사전 조건
- 1단계: Active Directory 도메인 서비스(AD DS)에서 gMSA 계정 생성 및 구성
- 2단계: Secrets Manager로 보안 인증 업로드
- 3단계: 도메인이 없는 gMSA 정보를 포함하도록 CredSpec JSON 수정
- 4단계: Amazon S3에 CredSpec 업로드
- 5단계: (선택 사항) Amazon ECS 클러스터 생성
- 6단계: 컨테이너 인스턴스에 대한 IAM 역할 생성
- 7단계: 사용자 지정 작업 실행 역할 생성
- 8단계: Amazon ECS Exec에 대한 작업 역할 생성
- 9단계: 도메인 없는 gMSA를 사용하는 작업 정의 등록
- 10단계: 클러스터에 Windows 컨테이너 인스턴스 등록
- 11단계: 컨테이너 인스턴스 확인
- 12단계: Windows 작업 실행
- 13단계: 컨테이너에 gMSA 보안 인증이 있는지 확인
- 14단계: 정리

- Windows 컨테이너에 대한 Amazon ECS 도메인 없는 gMSA 디버깅

사전 조건

이 자습서에서는 다음 사전 조건이 충족되었다고 가정합니다.

- Amazon ECS 사용 설정의 단계가 완료되었습니다.
- IAM 사용자는 AmazonECS_FullAccess IAM 정책 예제에 지정된 필수 권한을 가집니다.
- 최신 버전의 AWS CLI가 설치 및 구성됩니다. AWS CLI 설치 또는 업그레이드에 관한 자세한 정보는 AWS Command Line Interface 설치를 참조하세요.

  > **Note**
  >
  > 듀얼 스택 서비스 엔드포인트를 사용하면 AWS CLI, SDK 및 Amazon ECS API에서 IPv4 및 IPv6 모두를 통해 Amazon ECS와 상호 작용할 수 있습니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용 섹션을 참조하세요.

- 컨테이너에서 액세스할 리소스를 포함하는 Active Directory 도메인을 설정합니다. Amazon ECS는 다음 설정을 지원합니다.
  - AWS Directory Service Active Directory. AWS Directory Service는 Amazon EC2에 호스트되는 AWS 관리형 Active Directory입니다. 자세한 내용은 AWS Directory Service 관리 안내서의 AWS 관리형 Microsoft AD 시작하기를 참조하세요.
  - 온프레미스 Active Directory. Amazon ECS Linux 컨테이너 인스턴스가 도메인에 가입할 수 있도록 해야 합니다. 자세한 내용은 AWS Direct Connect 섹션을 참조하세요.
- Active Directory 도메인 이름을 확인할 수 있는 VPC와 서브넷이 있습니다.
- 도메인 없는 gMSA 및 각 인스턴스를 단일 도메인에 가입 중에서 선택할 수 있습니다. 도메인 없는 gMSA를 사용하면 컨테이너 인스턴스는 도메인에 가입되지 않고, 인스턴스의 다른 애플리케이션은 보안 인증을 사용하여 도메인에 액세스할 수 없고, 다른 도메인에 가입하는 작업은 동일한 인스턴스에서 실행될 수 있습니다.

  그런 다음 CredSpec 및 선택적으로 도메인 없는 gMSA에 대한 Active Directory 사용자 보안 인증으로 데이터 스토리지를 선택합니다.

  Amazon ECS는 Active Directory 보안 인증 사양 파일(CredSpec)을 사용합니다. 이 파일은 gMSA 계정 컨텍스트를 컨테이너에 전파하는 데 사용되는 gMSA 메타데이터를 포함합니다. CredSpec 파일을 생성하고 다음 표의 컨테이너 인스턴스의 운영 체제별 CredSpec 스토리지 옵션 중 하나에 저장합니다. 도메인 없는 방법을 사용하려면 CredSpec 파일의 선택적 섹션에서 다음 표의 컨테이너 인

스턴스의 운영 체제별 domainless user credentials스토리지 옵션 중 하나에 보안 인증을 지정할 수 있습니다.

스토리지 위치	Linux	Windows
Amazon Simple Storage Service(S3)	CredSpec	CredSpec
AWS Secrets Manager	도메인 없는 사용자 보안 인증	도메인 없는 사용자 보안 인증
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec, 도메인 없는 사용자 보안 인증
로컬 파일	N/A	CredSpec

- (선택 사항) AWS CloudShell은 고객에게 자체 EC2 인스턴스를 생성할 필요 없이 명령줄을 제공하는 도구입니다. 자세한 내용은 AWS CloudShell 사용 설명서의 AWS CloudShell이란 무엇입니까? 섹션을 참조하세요.

1단계: Active Directory 도메인 서비스(AD DS)에서 gMSA 계정 생성 및 구성

Active Directory 도메인에서 gMSA 계정을 생성하고 구성합니다.

> **Note**
>
> 이 단계에서는 컨테이너의 자격 증명을 제공하는 그룹 관리형 서비스 계정(gMSA) 및 도메인 인증에 사용되는 일반 사용자 계정과 같은 두 개의 개별 계정을 생성합니다. 이러한 계정은 용도가 서로 다르며 이름이 서로 달라야 합니다.

1. 키 배포 서비스 루트 키 생성

   > **Note**
   >
   > AWS Directory Service를 사용할 경우 이 단계를 건너뛸 수 있습니다.

   KDS 루트 키와 gMSA 권한은 AWS 관리형 Microsoft AD를 사용하여 구성됩니다.

도메인에서 아직 gMSA 서비스 계정을 생성하지 않았다면 먼저 키 배포 서비스(KDS) 루트 키를 생성해야 합니다. KDS는 gMSA 암호를 생성하고 교체하여 승인된 호스트에 배포하는 역할을 합니다. `ccg.exe`는 gMSA 보안 인증을 검색해야 하는 경우 KDS에 연락하여 현재 암호를 검색합니다.

KDS 루트 키가 이미 생성되었는지 확인하려면 `ActiveDirectory` PowerShell 모듈을 사용하여 도메인 컨트롤러에서 도메인 관리자 권한으로 다음 PowerShell cmdlet을 실행합니다. 모듈에 대한 자세한 내용은 Microsoft Learn 웹 사이트의 ActiveDirectory Module을 참조하세요.

```
PS C:\> Get-KdsRootKey
```

명령에서 키 ID를 반환하는 경우 이 단계의 나머지 부분은 건너뛸 수 있습니다. 그렇지 않으면 다음 명령을 실행하여 KDS 루트 키를 생성합니다.

```
PS C:\> Add-KdsRootKey -EffectiveImmediately
```

명령의 인수 `EffectiveImmediately`는 키가 즉시 유효함을 의미하지만 KDS 루트 키가 복제되어 모든 도메인 컨트롤러에서 사용할 수 있으려면 10시간을 기다려야 합니다.

2. gMSA 계정 생성

gMSA 계정을 생성하고 `ccg.exe`에서 gMSA 암호를 검색하도록 허용하려면 도메인에 대한 액세스 권한이 있는 Windows Server 또는 클라이언트에서 다음 PowerShell 명령을 실행합니다. `ExampleAccount`를 gMSA 계정에 대해 원하는 이름으로 바꾸고 `example-domain`을 Active Directory 도메인 이름(예: 도메인이 `contoso.com`인 경우 `contoso` 사용)으로 바꾸세요.

a.
```
PS C:\> Install-WindowsFeature RSAT-AD-PowerShell
```

b.
```
PS C:\> New-ADGroup -Name "ExampleAccount Authorized Hosts" -SamAccountName
 "ExampleAccountHosts" -GroupScope DomainLocal
```

c.
```
PS C:\> New-ADServiceAccount -Name "ExampleAccount" -DnsHostName "example-
domain" -ServicePrincipalNames "host/ExampleAccount", "host/example-domain" -
PrincipalsAllowedToRetrieveManagedPassword "ExampleAccountHosts"
```

d. 만료되지 않는 영구 암호로 사용자를 생성합니다. 이러한 보안 인증은 AWS Secrets Manager에 저장되며 각 작업에서 도메인에 가입하는 데 사용됩니다. 위에서 생성한 gMSA 계정과는

별도의 사용자 계정입니다. ExampleServiceUser를 이 서비스 사용자 계정에 대해 원하는 이름으로 바꾸세요.

```
PS C:\> New-ADUser -Name "ExampleServiceUser" -AccountPassword (ConvertTo-
SecureString -AsPlainText "Test123" -Force) -Enabled 1 -PasswordNeverExpires 1
```

e.
```
PS C:\> Add-ADGroupMember -Identity "ExampleAccountHosts" -Members
 "ExampleServiceUser"
```

f. 액티브 디렉터리에 CredSpec 개체를 생성하기 위한 PowerShell 모듈을 설치하고 CredSpec JSON을 출력합니다.

```
PS C:\> Install-PackageProvider -Name NuGet -Force
```

```
PS C:\> Install-Module CredentialSpec
```

g.
```
PS C:\> New-CredentialSpec -AccountName ExampleAccount
```

3. 이전 명령의 JSON 출력을 gmsa-cred-spec.json이라는 파일에 복사합니다. 이는 CredSpec 파일입니다. 3단계 3단계: 도메인이 없는 gMSA 정보를 포함하도록 CredSpec JSON 수정에서 사용됩니다.

2단계: Secrets Manager로 보안 인증 업로드

Active Directory 보안 인증을 안전한 자격 증명 스토리지 시스템에 복사하여 각 작업에서 검색할 수 있도록 합니다. 이는 도메인 없는 gMSA 방법입니다. 도메인 없는 gMSA를 사용하면 컨테이너 인스턴스는 도메인에 가입되지 않고, 인스턴스의 다른 애플리케이션은 보안 인증을 사용하여 도메인에 액세스할 수 없고, 다른 도메인에 가입하는 작업은 동일한 인스턴스에서 실행될 수 있습니다.

이 단계에서는 AWS CLI를 사용합니다. 기본 셸 bash의 AWS CloudShell에서 이러한 명령을 실행할 수 있습니다.

- 다음 AWS CLI 명령을 실행하고 환경에 맞게 사용자 이름, 암호 및 도메인 이름을 변경합니다. 사용자 이름에 대해 서비스 사용자 계정 이름(gMSA 계정 이름 아님)을 사용합니다. 보안 암호의 ARN을 보관하고 다음 단계 3단계: 도메인이 없는 gMSA 정보를 포함하도록 CredSpec JSON 수정에서 사용합니다.

다음 명령은 sh 및 호환 셸에서 사용되는 백슬래시 연속 문자를 사용합니다. 이 명령은 PowerShell과 호환되지 않습니다. PowerShell에서 사용하려면 명령을 수정해야 합니다.

```
$ aws secretsmanager create-secret \
--name gmsa-plugin-input \
--description "Amazon ECS - gMSA Portable Identity." \
--secret-string "{\"username\":\"ExampleServiceUser\",\"password\":\"Test123\",
\"domainName\":\"contoso.com\"}"
```

3단계: 도메인이 없는 gMSA 정보를 포함하도록 CredSpec JSON 수정

CredSpec를 스토리지 옵션 중 하나에 업로드하기 전에 이전 단계의 Secrets Manager에 있는 보안 암호의 ARN을 사용하여 CredSpec에 정보를 추가합니다. 자세한 내용은 Microsoft Learn 웹 사이트에서 Additional credential spec configuration for non-domain-joined container host use case를 참조하세요.

1. 다음 정보를 CredSpec 파일의 `ActiveDirectoryConfig` 안에 추가합니다. ARN을 이전 단계의 Secrets Manager에 있는 보안 암호로 바꿉니다.

   단, `PluginGUID` 값은 다음 예제 코드 조각의 GUID와 일치해야 하며 필수 항목입니다.

   ```
   "HostAccountConfig": {
         "PortableCcgVersion": "1",
         "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
         "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
   region:111122223333:secret:gmsa-plugin-input\"}"
       }
   ```

   다음 형식(\"arn:aws:ssm:*aws-region*:111122223333:parameter/*gmsa-plugin-input*\")의 ARN을 사용하여 SSM Parameter Store의 보안 암호를 사용할 수도 있습니다.
2. 수정한 CredSpec 파일은 다음 예제와 같아야 합니다.

   ```
   {
     "CmsPlugins": [
       "ActiveDirectory"
     ],
     "DomainJoinConfig": {
       "Sid": "S-1-5-21-4066351383-705263209-1606769140",
       "MachineAccountName": "ExampleAccount",
       "Guid": "ac822f13-583e-49f7-aa7b-284f9a8c97b6",
   ```

```
    "DnsTreeName": "example-domain",
    "DnsName": "example-domain",
    "NetBiosName": "example-domain"
  },
  "ActiveDirectoryConfig": {
    "GroupManagedServiceAccounts": [
      {
        "Name": "ExampleAccount",
        "Scope": "example-domain"
      },
      {
        "Name": "ExampleAccount",
        "Scope": "example-domain"
      }
    ],
    "HostAccountConfig": {
      "PortableCcgVersion": "1",
      "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
      "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
region:111122223333:secret:gmsa-plugin-input\"}"
    }
  }
}
```

4단계: Amazon S3에 CredSpec 업로드

이 단계에서는 AWS CLI를 사용합니다. 기본 셸 bash의 AWS CloudShell에서 이러한 명령을 실행할 수 있습니다.

1. AWS CLI 명령을 실행 중인 컴퓨터나 환경에 CredSpec 파일을 복사합니다.
2. 다음 AWS CLI 명령을 실행하여 Amazon S3에 CredSpec를 업로드합니다. `MyBucket`을 Amazon S3 버킷 이름으로 바꿉니다. 파일을 모든 버킷과 위치에 개체로 저장할 수 있지만 작업 실행 역할에 연결하는 정책에서 해당 버킷과 위치에 대한 액세스를 허용해야 합니다.

   다음 명령은 sh 및 호환 셸에서 사용되는 백슬래시 연속 문자를 사용합니다. 이 명령은 PowerShell과 호환되지 않습니다. PowerShell에서 사용하려면 명령을 수정해야 합니다.

```
$ aws s3 cp gmsa-cred-spec.json \
s3://MyBucket/ecs-domainless-gmsa-credspec
```

5단계: (선택 사항) Amazon ECS 클러스터 생성

기본적으로 계정에는 `default`라는 Amazon ECS 클러스터가 있습니다. 이 클러스터는 AWS CLI, SDK 및 AWS CloudFormation에서 기본적으로 사용됩니다. 추가 클러스터를 사용하여 작업 및 인프라를 그룹화 및 구성하고 일부 구성에 기본값을 할당할 수 있습니다.

AWS Management Console, AWS CLI, SDK 또는 AWS CloudFormation를 사용하여 클러스터를 생성할 수 있습니다. 클러스터의 설정 및 구성은 gMSA에 영향을 주지 않습니다.

이 단계에서는 AWS CLI를 사용합니다. 기본 셸 `bash`의 AWS CloudShell에서 이러한 명령을 실행할 수 있습니다.

```
$ aws ecs create-cluster --cluster-name windows-domainless-gmsa-cluster
```

> **Important**
>
> 자체 클러스터를 생성하는 경우, 해당 클러스터에 사용할 각 명령에 `--cluster clusterName`을 지정해야 합니다.

6단계: 컨테이너 인스턴스에 대한 IAM 역할 생성

컨테이너 인스턴스는 ECS 작업에서 컨테이너를 실행하는 호스트 컴퓨터입니다(예: Amazon EC2 인스턴스). 각 컨테이너 인스턴스는 Amazon ECS 클러스터에 등록합니다. Amazon EC2 인스턴스를 시작해 클러스터에 등록하기 전에 사용할 컨테이너 인스턴스의 IAM 역할을 생성해야 합니다.

컨테이너 인스턴스 역할을 생성하려면 Amazon ECS 컨테이너 인스턴스 IAM 역할을 참조하세요. 기본값 `ecsInstanceRole`에는 이 자습서를 완료하기에 충분한 권한이 있습니다.

7단계: 사용자 지정 작업 실행 역할 생성

Amazon ECS는 각 작업을 시작하는 데 필요한 권한에 대해 컨테이너 인스턴스 역할 대신 다른 IAM 역할을 사용할 수 있습니다. 이 역할을 작업 실행 역할이라고 합니다. ECS에서 작업을 실행하는 데 필요한 권한(최소 권한 권한이라고도 함)만 포함된 작업 실행 역할을 생성하는 것이 좋습니다. 최소 권한 원칙에 대한 자세한 내용은 AWS Well-Architected Framework의 SEC03-BP02 Grant least privilege access를 참조하세요.

1. 작업 실행 역할을 생성하려면 작업 실행 역할 생성을 참조하세요. 기본 권한을 사용하면 컨테이너 인스턴스가 Amazon Elastic Container Registry와 애플리케이션의 `stdout` 및 `stderr`에 컨테이너 이미지를 풀하고 Amazon CloudWatch Logs에 기록할 수 있습니다.

   이 자습서에서는 역할에 사용자 지정 권한이 필요하므로 역할에 `ecsTaskExecutionRole`과 다른 이름을 지정할 수 있습니다. 이 자습서의 이후 단계에서는 `ecsTaskExecutionRole`을 사용합니다.

2. 이 역할에만 존재하는 인라인 정책이나 재사용할 수 있는 정책으로 사용자 지정 정책을 생성하여 다음 권한을 추가합니다. 첫 번째 명령문에서 `Resource`의 ARN을 Amazon S3 버킷 및 위치로 바꾸고, 두 번째 명령문에서 `Resource`를 Secrets Manager의 보안 암호의 ARN으로 바꿉니다.

   Secrets Manager에서 보안 암호를 사용자 지정 키로 암호화하는 경우 키의 `kms:Decrypt`도 허용해야 합니다.

   Secrets Manager 대신 SSM Parameter Store를 사용하는 경우 `secretsmanager:GetSecretValue` 대신 파라미터에 대해 `ssm:GetParameter`를 허용해야 합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::MyBucket/ecs-domainless-gmsa-credspec/
gmsa-cred-spec.json"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource":
 "arn:aws:secretsmanager:arn:aws:: 111122223333:secret:gmsa-plugin-AbCdEf"
        }
    ]
```

```
}
```

8단계: Amazon ECS Exec에 대한 작업 역할 생성

이 자습서에서는 Amazon ECS Exec을 사용하여 실행 중인 작업 내에서 명령을 실행하여 기능을 확인합니다. ECS Exec를 사용하려면 서비스 또는 작업에서 ECS Exec을 켜고 작업 실행 역할이 아닌 작업 역할에 ssmmessages 권한이 있어야 합니다. 필요한 IAM 정책은 ECS Exec 권한를 참조하세요.

이 단계에서는 AWS CLI를 사용합니다. 기본 셸 bash의 AWS CloudShell에서 이러한 명령을 실행할 수 있습니다.

AWS CLI를 사용하여 작업 역할을 생성하려면 다음 단계를 따릅니다.

1. 다음 콘텐츠로 ecs-tasks-trust-policy.json이라는 파일을 생성합니다.

   JSON

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "ecs-tasks.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

2. IAM 역할을 생성합니다. 이름 ecs-exec-demo-task-role은 바꿀 수 있지만 다음 단계를 위해 그대로 유지하세요.

   다음 명령은 sh 및 호환 셸에서 사용되는 백슬래시 연속 문자를 사용합니다. 이 명령은 PowerShell과 호환되지 않습니다. PowerShell에서 사용하려면 명령을 수정해야 합니다.

   ```
   $ aws iam create-role --role-name ecs-exec-demo-task-role \
   --assume-role-policy-document file://ecs-tasks-trust-policy.json
   ```

   파일 ecs-tasks-trust-policy.json을 삭제할 수 있습니다.

3. 다음 콘텐츠로 ecs-exec-demo-task-role-policy.json이라는 파일을 생성합니다.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateControlChannel",
                "ssmmessages:CreateDataChannel",
                "ssmmessages:OpenControlChannel",
                "ssmmessages:OpenDataChannel"
            ],
            "Resource": "*"
        }
    ]
}
```

4. IAM 역할을 생성하여 이전 단계의 역할에 연결합니다.

   다음 명령은 sh 및 호환 셸에서 사용되는 백슬래시 연속 문자를 사용합니다. 이 명령은 PowerShell과 호환되지 않습니다. PowerShell에서 사용하려면 명령을 수정해야 합니다.

```
$ aws iam put-role-policy \
    --role-name ecs-exec-demo-task-role \
    --policy-name ecs-exec-demo-task-role-policy \
    --policy-document file://ecs-exec-demo-task-role-policy.json
```

   파일 ecs-exec-demo-task-role-policy.json을 삭제할 수 있습니다.

9단계: 도메인 없는 gMSA를 사용하는 작업 정의 등록

이 단계에서는 AWS CLI를 사용합니다. 기본 셸 bash의 AWS CloudShell에서 이러한 명령을 실행할 수 있습니다.

1. 다음 콘텐츠로 windows-gmsa-domainless-task-def.json이라는 파일을 생성합니다.

```
{
```

```
  "family": "windows-gmsa-domainless-task",
  "containerDefinitions": [
    {
      "name": "windows_sample_app",
      "image": "mcr.microsoft.com/windows/servercore/iis",
      "cpu": 1024,
      "memory": 1024,
      "essential": true,
      "credentialSpecs": [
                "credentialspecdomainless:arn:aws:s3:::ecs-domainless-gmsa-
credspec/gmsa-cred-spec.json"
      ],
      "entryPoint": [
        "powershell",
        "-Command"
      ],
      "command": [
        "New-Item -Path C:\\inetpub\\wwwroot\\index.html -ItemType file -Value
 '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>' -Force ; C:\
\ServiceMonitor.exe w3svc"
      ],
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80,
          "hostPort": 8080
        }
      ]
    }
  ],
  "taskRoleArn": "arn:aws:iam::111122223333:role/ecs-exec-demo-task-role",
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole"
}
```

2. 다음 명령을 실행하여 작업 정의를 등록합니다.

   다음 명령은 sh 및 호환 셸에서 사용되는 백슬래시 연속 문자를 사용합니다. 이 명령은 PowerShell과 호환되지 않습니다. PowerShell에서 사용하려면 명령을 수정해야 합니다.

```
$ aws ecs register-task-definition \
```

```
--cli-input-json file://windows-gmsa-domainless-task-def.json
```

10단계: 클러스터에 Windows 컨테이너 인스턴스 등록

Amazon EC2 Windows 인스턴스를 시작하고 ECS 컨테이너 에이전트를 실행하여 클러스터의 컨테이너 인스턴스로 등록합니다. ECS는 작업이 시작되는 클러스터에 등록된 컨테이너 인스턴스에서 작업을 실행합니다.

1. AWS Management Console에서 Amazon ECS용으로 구성된 Amazon EC2 Windows 인스턴스를 시작하려면 Amazon ECS Windows 컨테이너 인스턴스 시작를 참조하세요. 사용자 데이터에 대한 단계에서 중지합니다.
2. gMSA의 경우 ECS 컨테이너 에이전트를 시작하기 전에 사용자 데이터에서 환경 변수 `ECS_GMSA_SUPPORTED`를 설정해야 합니다.

   ECS Exec의 경우 에이전트는 `-EnableTaskIAMRole` 인수로 시작해야 합니다.

   작업이 EC2 IMDS 웹 서비스에 도달하여 역할 보안 인증을 검색하는 것을 방지하여 인스턴스 IAM 역할을 보호하려면 `-AwsvpcBlockIMDS` 인수를 추가합니다. 이는 `awsvpc` 네트워크 모드를 사용하는 작업에만 적용됩니다.

   ```
   <powershell>
   [Environment]::SetEnvironmentVariable("ECS_GMSA_SUPPORTED", $TRUE, "Machine")
   Import-Module ECSTools
   Initialize-ECSAgent -Cluster windows-domainless-gmsa-cluster -EnableTaskIAMRole -
   AwsvpcBlockIMDS
   </powershell>
   ```
3. 요약 패널에서 인스턴스 구성 요약을 검토하고 준비가 되면 인스턴스 시작을 선택합니다.

11단계: 컨테이너 인스턴스 확인

AWS Management Console을 사용하여 클러스터에 컨테이너 인스턴스가 있는지 확인할 수 있습니다. 하지만 gMSA의 경우 속성으로 표시되는 추가 기능이 필요합니다. 이러한 속성은 AWS Management Console에 표시되지 않으므로 이 자습서에서는 AWS CLI를 사용합니다.

이 단계에서는 AWS CLI를 사용합니다. 기본 셸 bash의 AWS CloudShell에서 이러한 명령을 실행할 수 있습니다.

1. 클러스터의 컨테이너 인스턴스를 나열합니다. 컨테이너 인스턴스의 ID는 EC2 인스턴스의 ID와 다릅니다.

   ```
   $ aws ecs list-container-instances
   ```

   출력:

   ```
   {
       "containerInstanceArns": [
           "arn:aws:ecs:aws-region:111122223333:container-
   instance/default/MyContainerInstanceID"
       ]
   }
   ```

   예를 들어 526bd5d0ced448a788768334e79010fd는 유효한 컨테이너 인스턴스 ID입니다.
2. 이전 단계의 컨테이너 인스턴스 ID를 사용하여 컨테이너 인스턴스의 세부 정보를 가져옵니다. MyContainerInstanceID를 ID로 바꿉니다.

   다음 명령은 sh 및 호환 셸에서 사용되는 백슬래시 연속 문자를 사용합니다. 이 명령은 PowerShell과 호환되지 않습니다. PowerShell에서 사용하려면 명령을 수정해야 합니다.

   ```
   $ aws ecs describe-container-instances \
        ----container-instances MyContainerInstanceID
   ```

   참고로 출력 내용이 매우 깁니다.
3. attributes 목록에 name이라는 키와 ecs.capability.gmsa-domainless 값을 갖는 개체가 있는지 확인합니다. 다음은 이 개체의 예입니다.

   출력:

   ```
   {
       "name": "ecs.capability.gmsa-domainless"
   }
   ```

12단계: Windows 작업 실행

Amazon ECS 작업을 실행합니다. 클러스터에 컨테이너 인스턴스가 1개만 있는 경우 run-task를 사용할 수 있습니다. 컨테이너 인스턴스가 여러 개 있는 경우 start-task을 사용하고 작업을 실행할 컨

테이너 인스턴스 ID를 지정한 다음 작업 정의에 배치 제약을 추가하여 이 작업이 실행되는 컨테이너 인스턴스의 유형을 제어하는 것이 더 쉬울 수 있습니다.

이 단계에서는 AWS CLI를 사용합니다. 기본 셸 bash의 AWS CloudShell에서 이러한 명령을 실행할 수 있습니다.

1. 다음 명령은 sh 및 호환 셸에서 사용되는 백슬래시 연속 문자를 사용합니다. 이 명령은 PowerShell과 호환되지 않습니다. PowerShell에서 사용하려면 명령을 수정해야 합니다.

   ```
   aws ecs run-task --task-definition windows-gmsa-domainless-task \
       --enable-execute-command --cluster windows-domainless-gmsa-cluster
   ```

   명령에서 반환된 작업 ID를 메모합니다.
2. 다음 명령을 실행하여 작업이 시작되었는지 확인합니다. 이 명령은 대기하며 작업이 시작될 때까지 셸 프롬프트를 반환하지 않습니다. `MyTaskID`을 전 단계의 작업 ID로 바꿉니다.

   ```
   $ aws ecs wait tasks-running --task MyTaskID
   ```

13단계: 컨테이너에 gMSA 보안 인증이 있는지 확인

작업의 컨테이너에 Kerberos 토큰 gMSA가 있는지 확인합니다.

이 단계에서는 AWS CLI를 사용합니다. 기본 셸 bash의 AWS CloudShell에서 이러한 명령을 실행할 수 있습니다.

1. 다음 명령은 sh 및 호환 셸에서 사용되는 백슬래시 연속 문자를 사용합니다. 이 명령은 PowerShell과 호환되지 않습니다. PowerShell에서 사용하려면 명령을 수정해야 합니다.

   ```
   $ aws ecs execute-command \
   --task MyTaskID \
   --container windows_sample_app \
   --interactive \
   --command powershell.exe
   ```

   PowerShell 프롬프트가 출력됩니다.
2. 컨테이너 내의 PowerShell 터미널에서 다음 명령을 실행합니다.

```
PS C:\> klist get ExampleAccount$
```

출력에서 `Principal`이 이전에 생성한 것인지 확인합니다.

14단계: 정리

이 자습서로 완료를 한 후에 사용하지 않는 리소스에 대해 요금이 발생하는 것을 방지하기 위해 연결된 리소스를 정리해야 합니다.

이 단계에서는 AWS CLI를 사용합니다. 기본 셸 bash의 AWS CloudShell에서 이러한 명령을 실행할 수 있습니다.

1. 작업을 중지합니다. `MyTaskID`를 12단계 12단계: Windows 작업 실행의 작업 ID로 바꿉니다.

   ```
   $ aws ecs stop-task --task MyTaskID
   ```

2. Amazon EC2 인스턴스를 종료합니다. 이후 클러스터의 컨테이너 인스턴스는 1시간 후에 자동으로 삭제됩니다.

   Amazon EC2 콘솔을 사용하여 인스턴스를 찾고 종료할 수 있습니다. 또는 다음 명령을 실행할 수 있습니다. 명령을 실행하려면 1단계 11단계: 컨테이너 인스턴스 확인의 `aws ecs describe-container-instances` 명령 출력에서 EC2 인스턴스 ID를 찾습니다. i-10a64379는 EC2 인스턴스 ID의 예입니다.

   ```
   $ aws ec2 terminate-instances --instance-ids MyInstanceID
   ```

3. Amazon S3에서 CredSpec 파일을 삭제합니다. `MyBucket`을 Amazon S3 버킷 이름으로 바꿉니다.

   ```
   $ aws s3api delete-object --bucket MyBucket --key ecs-domainless-gmsa-credspec/
   gmsa-cred-spec.json
   ```

4. Secrets Manager에서 보안 암호를 삭제합니다. SSM Parameter Store를 대신 사용한 경우 파라미터를 삭제합니다.

   다음 명령은 sh 및 호환 셸에서 사용되는 백슬래시 연속 문자를 사용합니다. 이 명령은 PowerShell과 호환되지 않습니다. PowerShell에서 사용하려면 명령을 수정해야 합니다.

   ```
   $ aws secretsmanager delete-secret --secret-id gmsa-plugin-input \
   ```

```
    --force-delete-without-recovery
```

5. 작업 정의를 등록 취소하고 삭제합니다. 작업 정의를 등록 취소하면 시작하는 데 사용할 수 없도록 비활성으로 표시됩니다. 그러면 작업 정의를 삭제할 수 있습니다.

   a. 버전을 지정하여 작업 정의를 등록 취소합니다. ECS는 1부터 시작하는 번호를 지정하여 작업 정의 버전을 자동으로 만듭니다. 컨테이너 이미지의 레이블과 같은 형식으로 버전을 참조합니다(예: `:1`)

   ```
   $ aws ecs deregister-task-definition --task-definition windows-gmsa-domainless-task:1
   ```

   b. 작업 정의를 삭제합니다.

   ```
   $ aws ecs delete-task-definitions --task-definition windows-gmsa-domainless-task:1
   ```

6. (선택 사항) 클러스터를 생성한 경우 ECS 클러스터를 삭제합니다.

```
$ aws ecs delete-cluster --cluster windows-domainless-gmsa-cluster
```

Windows 컨테이너에 대한 Amazon ECS 도메인 없는 gMSA 디버깅

Amazon ECS 작업 상태

ECS는 작업 시작을 정확히 한 번 시도합니다. 문제가 있는 모든 작업은 중지되고 `STOPPED` 상태로 설정됩니다. 작업과 관련된 문제에는 두 가지 일반적인 유형이 있습니다. 첫 번째는 시작할 수 없는 작업입니다. 둘째, 컨테이너 중 하나에서 애플리케이션이 중지된 작업입니다. AWS Management Console에서 작업의 중지 이유 필드에서 작업이 중지된 이유를 확인합니다. AWS CLI에서는 작업을 설명하고 `stoppedReason`을 봅니다. AWS Management Console 및 AWS CLI의 단계에 대한 내용은 Amazon ECS 중지된 작업 오류 보기를 참조하세요.

Windows 이벤트

컨테이너의 gMSA에 대한 Windows 이벤트는 `Microsoft-Windows-Containers-CCG` 로그 파일에 기록되며 이벤트 뷰어에서 `Logs\Microsoft\Windows\Containers-CCG\Admin`의 애플리케이션 및 서비스 섹션에서 확인할 수 있습니다. 자세한 디버깅 팁은 Microsoft Learn 웹사이트의 Troubleshoot gMSAs for Windows containers를 참조하세요.

ECS 에이전트 gMSA 플러그인

Windows 컨테이너 인스턴스의 ECS 에이전트용 gMSA 플러그인에 대한 로깅은 `C:/ProgramData/Amazon/gmsa-plugin/` 디렉터리에 있습니다. 이 로그에서 도메인 없는 사용자 보안 인증이 Secrets Manager와 같은 스토리지 위치에서 다운로드되었는지, 보안 인증 형식을 올바르게 읽었는지 확인합니다.

Amazon ECS의 EC2 Windows 컨테이너에 대해 gMSA를 사용하는 방법 알아보기

Amazon ECS는 그룹 관리형 서비스 계정(gMSA)이라는 특수한 종류의 서비스 계정을 통해 Windows 컨테이너에 대한 Active Directory 인증을 지원합니다.

.NET 애플리케이션과 같은 Windows 기반 네트워크 애플리케이션은 종종 Active Directory를 사용하여 사용자와 서비스 간의 인증 및 권한 부여 관리를 용이하게 합니다. 개발자는 일반적으로 이러한 목적으로 Active Directory와 통합되고 도메인에 가입된 서버에서 실행되도록 애플리케이션을 설계합니다. Windows 컨테이너는 도메인에 가입할 수 없으므로 gMSA와 함께 실행되도록 Windows 컨테이너를 구성해야 합니다.

gMSA로 실행 중인 Windows 컨테이너는 호스트 Amazon EC2 인스턴스에 의존하여 Active Directory 도메인 컨트롤러에서 gMSA 자격 증명을 검색하고 이를 컨테이너 인스턴스에 제공합니다. 자세한 내용은 Windows 컨테이너용 gMSA 생성 섹션을 참조하세요.

> **Note**
>
> 이 기능은 Fargate의 Windows 컨테이너에서 지원되지 않습니다.

주제

- 고려 사항
- 사전 조건
- Amazon ECS에서 Windows 컨테이너에 대한 gMSA 설정

고려 사항

Windows 컨테이너에 gMSA를 사용할 때는 다음 사항을 고려해야 합니다.

- 컨테이너 인스턴스에 Amazon ECS 최적화 Windows Server 2016 Full AMI를 사용할 경우 컨테이너 호스트 이름은 자격 증명 사양 파일에 정의된 gMSA 계정 이름과 같아야 합니다. 컨테이너에 대한 호스트 이름을 지정하려면 `hostname` 컨테이너 정의 파라미터를 사용합니다. 자세한 내용은 네트워크 설정 섹션을 참조하세요.
- 도메인 없는 gMSA 및 각 인스턴스를 단일 도메인에 가입 중에서 선택할 수 있습니다. 도메인 없는 gMSA를 사용하면 컨테이너 인스턴스는 도메인에 가입되지 않고, 인스턴스의 다른 애플리케이션은 보안 인증을 사용하여 도메인에 액세스할 수 없고, 다른 도메인에 가입하는 작업은 동일한 인스턴스에서 실행될 수 있습니다.

  그런 다음 CredSpec 및 선택적으로 도메인 없는 gMSA에 대한 Active Directory 사용자 보안 인증으로 데이터 스토리지를 선택합니다.

  Amazon ECS는 Active Directory 보안 인증 사양 파일(CredSpec)을 사용합니다. 이 파일은 gMSA 계정 컨텍스트를 컨테이너에 전파하는 데 사용되는 gMSA 메타데이터를 포함합니다. CredSpec 파일을 생성하고 다음 표의 컨테이너 인스턴스의 운영 체제별 CredSpec 스토리지 옵션 중 하나에 저장합니다. 도메인 없는 방법을 사용하려면 CredSpec 파일의 선택적 섹션에서 다음 표의 컨테이너 인스턴스의 운영 체제별 domainless user credentials스토리지 옵션 중 하나에 보안 인증을 지정할 수 있습니다.

스토리지 위치	Linux	Windows
Amazon Simple Storage Service(S3)	CredSpec	CredSpec
AWS Secrets Manager	도메인 없는 사용자 보안 인증	도메인 없는 사용자 보안 인증
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec, 도메인 없는 사용자 보안 인증
로컬 파일	N/A	CredSpec

사전 조건

Amazon ECS에서 Windows 컨테이너 기능에 gMSA를 사용하려면 먼저 다음 사항을 충족해야 합니다.

- 컨테이너에서 액세스할 리소스를 포함하는 Active Directory 도메인을 설정합니다. Amazon ECS는 다음 설정을 지원합니다.

  - AWS Directory Service Active Directory. AWS Directory Service는 Amazon EC2에 호스트되는 AWS 관리형 Active Directory입니다. 자세한 내용은 AWS Directory Service 관리 안내서의 AWS 관리형 Microsoft AD 시작하기를 참조하세요.
  - 온프레미스 Active Directory. Amazon ECS Linux 컨테이너 인스턴스가 도메인에 가입할 수 있도록 해야 합니다. 자세한 내용은 AWS Direct Connect 섹션을 참조하세요.
- Active Directory에 기존 gMSA 계정이 있습니다. 자세한 내용은 Windows 컨테이너용 gMSA 생성 섹션을 참조하세요.
- 도메인 없는 gMSA 또는 Amazon ECS를 사용하도록 선택한 경우 Amazon ECS 작업을 호스팅하는 Windows 컨테이너 인스턴스는 Active Directory에 '도메인 가입'되어 있어야 하며 gMSA 계정에 액세스할 수 있는 Active Directory 보안 그룹의 구성원이어야 합니다.

  도메인 없는 gMSA를 사용하면 컨테이너 인스턴스는 도메인에 가입되지 않고, 인스턴스의 다른 애플리케이션은 보안 인증을 사용하여 도메인에 액세스할 수 없고, 다른 도메인에 가입하는 작업은 동일한 인스턴스에서 실행될 수 있습니다.
- 필요한 IAM 권한을 추가했습니다. 필요한 권한은 초기 보안 인증 및 보안 인증 사양 저장을 위해 선택하는 방법에 따라 달라집니다.
  - 초기 보안 인증에 도메인 없는 gMSA를 사용하는 경우 AWS Secrets Manager에 대한 IAM 권한이 Amazon EC2 인스턴스 역할에 필요합니다.
  - 보안 인증 사양을 SSM Parameter Store에 저장하는 경우 Amazon EC2 Systems Manager Parameter Store에 대한 IAM 권한이 작업 실행 역할에 필요합니다.
  - 보안 인증 사양을 Amazon S3에 저장하는 경우에는 Amazon Simple Storage Service에 대한 IAM 권한이 작업 실행 역할에 필요합니다.

Amazon ECS에서 Windows 컨테이너에 대한 gMSA 설정

Amazon ECS에서 Windows 컨테이너에 대한 gMSA를 설정하려면 사전 조건 구성을 포함하는 전체 자습서 AWS CLI를 사용하여 도메인이 없는 gMSA로 Amazon ECS Windows 컨테이너 사용를 따르면 됩니다.

다음 섹션에서는 CredSpec 구성에 대해 자세히 설명합니다.

주제

- 예: CredSpec
- 도메인 없는 gMSA 설정
- 태스크 정의에서 자격 증명 사양 파일 참조

예: CredSpec

Amazon ECS는 gMSA 계정 컨텍스트를 Windows 컨테이너에 전파하는 데 사용되는 gMSA 메타데이터가 포함된 보안 인증 사양 파일을 사용합니다. 자격 증명 사양 파일을 생성하여 태스크 정의의 `credentialSpec` 필드에서 참조할 수 있습니다. 자격 증명 사양 파일에 암호가 포함되어 있지 않습니다.

다음은 자격 증명 사양 파일의 예입니다.

```
{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
        "MachineAccountName": "WebApp01",
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "contoso.com",
        "DnsName": "contoso.com",
        "NetBiosName": "contoso"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "contoso.com"
            }
        ]
    }
}
```

도메인 없는 gMSA 설정

컨테이너 인스턴스를 단일 도메인에 가입시키는 대신 도메인 없는 gMSA를 사용하는 것이 좋습니다. 도메인 없는 gMSA를 사용하면 컨테이너 인스턴스는 도메인에 가입되지 않고, 인스턴스의 다른 애플리케이션은 보안 인증을 사용하여 도메인에 액세스할 수 없고, 다른 도메인에 가입하는 작업은 동일한 인스턴스에서 실행될 수 있습니다.

1. CredSpec를 스토리지 옵션 중 하나에 업로드하기 전에 Secrets Manager 또는 SSM Parameter Store에서 보안 암호의 ARN을 사용하여 CredSpec에 정보를 추가합니다. 자세한 내용은

Microsoft Learn 웹 사이트에서 Additional credential spec configuration for non-domain-joined container host use case를 참조하세요.

도메인 없는 gMSA 보안 인증 형식

다음은 Active Directory에 대한 도메인 없는 gMSA 보안 인증의 JSON 형식입니다. 보안 인증을 Secrets Manager 또는 SSM Parameter Store에 저장합니다.

```
{
    "username":"WebApp01",
    "password":"Test123!",
    "domainName":"contoso.com"
}
```

2. 다음 정보를 CredSpec 파일의 `ActiveDirectoryConfig` 안에 추가합니다. ARN을 Secrets Manager 또는 SSM Parameter Store의 보안 암호를 바꿉니다.

   단, `PluginGUID` 값은 다음 예제 코드 조각의 GUID와 일치해야 하며 필수 항목입니다.

```
    "HostAccountConfig": {
        "PortableCcgVersion": "1",
        "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
        "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
region:111122223333:secret:gmsa-plugin-input\"}"
    }
```

   다음 형식(\"arn:aws:ssm:*aws-region*:111122223333:parameter/*gmsa-plugin-input*\")의 ARN을 사용하여 SSM Parameter Store의 보안 암호를 사용할 수도 있습니다.

3. 수정한 CredSpec 파일은 다음 예제와 같아야 합니다.

```
{
  "CmsPlugins": [
    "ActiveDirectory"
  ],
  "DomainJoinConfig": {
    "Sid": "S-1-5-21-4066351383-705263209-1606769140",
    "MachineAccountName": "WebApp01",
    "Guid": "ac822f13-583e-49f7-aa7b-284f9a8c97b6",
    "DnsTreeName": "contoso",
    "DnsName": "contoso",
```

```
    "NetBiosName": "contoso"
  },
  "ActiveDirectoryConfig": {
    "GroupManagedServiceAccounts": [
      {
        "Name": "WebApp01",
        "Scope": "contoso"
      },
      {
        "Name": "WebApp01",
        "Scope": "contoso"
      }
    ],
    "HostAccountConfig": {
      "PortableCcgVersion": "1",
      "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
      "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
region:111122223333:secret:gmsa-plugin-input\"}"
    }
  }
}
```

태스크 정의에서 자격 증명 사양 파일 참조

Amazon ECS에서는 작업 정의의 `credentialSpecs` 필드에서 파일 경로를 참조하는 다음과 같은 방법을 지원합니다. 이러한 각 옵션에 컨테이너 인스턴스를 단일 도메인에 가입시키는지, 아니면 도메인 없는 gMSA를 사용하는지에 따라 각각 `credentialspec:` 또는 `domainlesscredentialspec:`를 입력할 수 있습니다.

Amazon S3 버킷

Amazon S3 버킷에 자격 증명 사양을 추가한 다음 태스크 정의의 `credentialSpecs` 필드에서 Amazon S3 버킷의 Amazon 리소스 이름(ARN)을 참조합니다.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
```

```
                "credentialspecdomainless:arn:aws:s3:::${BucketName}/${ObjectName}"
            ],
            ...
        }
    ],
    ...
}
```

또한 작업에 Amazon S3 버킷에 대한 액세스 권한을 부여하려면 Amazon ECS 작업 실행 IAM 역할에 다음 권한을 인라인 정책으로 추가해야 합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::{bucket_name}",
                "arn:aws:s3:::{bucket_name}/{object}"
            ]
        }
    ]
}
```

SSM Parameter Store 파라미터

SSM Parameter Store 파라미터에 자격 증명 사양을 추가한 다음 태스크 정의의 `credentialSpecs` 필드에서 SSM Parameter Store 파라미터의 Amazon 리소스 이름(ARN)을 참조합니다.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
```

```
            "name": "",
            ...
            "credentialSpecs": [

 "credentialspecdomainless:arn:aws:ssm:region:111122223333:parameter/parameter_name"
            ],
            ...
        }
    ],
    ...
}
```

또한 작업에 SSM Parameter Store 파라미터에 대한 액세스 권한을 부여하려면 Amazon ECS 작업 실행 IAM 역할에 다음 권한을 인라인 정책으로 추가해야 합니다.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
            "arn:aws:ssm:arn:aws::111122223333:parameter/parameter_name"
            ]
        }
    ]
}
```

로컬 파일

로컬 파일의 자격 증명 사양 세부 정보를 사용하여 태스크 정의의 `credentialSpecs` 필드에 있는 파일 경로를 참조합니다. 참조되는 파일 경로는 `C:\ProgramData\Docker\CredentialSpecs` 디렉터리를 기준으로 해야 하며 백슬래시('\')를 파일 경로 구분자로 사용해야 합니다.

```
{
    "family": "",
    "executionRoleArn": "",
```

```
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspec:file://CredentialSpecDir\CredentialSpecFile.json"
            ],
            ...
        }
    ],
    ...
}
```

EC2 Image Builder를 사용하여 사용자 지정된 Amazon ECS 최적화 AMI 구축

AWS에서는 Amazon ECS 최적화 AMI가 컨테이너 워크로드를 실행하기 위한 요구 사항 및 권장 사항으로 미리 구성되어 있으므로 이를 사용하도록 권장합니다. 소프트웨어를 더 추가하기 위해 AMI를 사용자 지정해야 하는 경우가 있을 수 있습니다. 서버 이미지의 생성, 관리 및 배포를 위해 EC2 Image Builder를 사용할 수 있습니다. 계정에서 생성된 사용자 지정된 이미지에 대한 소유권은 사용자에게 있습니다. EC2 Image Builder 파이프라인을 사용하여 이미지에 대한 업데이트 및 시스템 패치를 자동화하거나 독립 실행형 명령을 사용하여 정의된 구성 리소스로 이미지를 생성할 수 있습니다.

이미지에 대한 레시피를 생성합니다. 레시피에는 상위 이미지와 추가 구성 요소가 포함됩니다. 또한 사용자 지정된 AMI를 배포하는 파이프라인도 생성합니다.

이미지에 대한 레시피를 생성합니다. Image Builder 이미지 레시피는 출력 AMI 이미지에 대해 원하는 구성을 생성하기 위해 기본 이미지 및 기본 이미지에 적용할 구성 요소를 정의하는 문서입니다. 또한 사용자 지정된 AMI를 배포하는 파이프라인도 생성합니다. 자세한 내용은 EC2 Image Builder 사용 설명서의 How EC2 Image Builder works를 참조하세요.

EC2 Image Builder에서 다음과 같은 Amazon ECS 최적화 AMI 중 하나를 '상위 이미지'로 사용하는 것이 좋습니다.

- Linux
  - Amazon ECS 최적화 AL2023 x86
  - Amazon ECS 최적화 Amazon Linux 2023(arm64) AMI
  - Amazon ECS 최적화 Amazon Linux 2 커널 5 AMI
  - Amazon ECS 최적화 Amazon Linux 2 x86 AMI

- Windows
  - Amazon ECS 최적화 Windows 2022 Full x86
  - Amazon ECS 최적화 Windows 2022 Core x86
  - Amazon ECS 최적화 Windows 2019 Full x86
  - Amazon ECS 최적화 Windows 2019 Core x86
  - Amazon ECS 최적화 Windows 2016 Full x86

'사용 가능한 최신 OS 버전 사용'을 선택하는 것이 좋습니다. 파이프라인은 상위 이미지의 시맨틱 버전 관리를 사용하므로 자동으로 예약된 작업에서 종속성 업데이트를 감지하는 데 도움이 됩니다. 자세한 내용은 EC2 Image Builder 사용 설명서의 Semantic versioning을 참조하세요.

AWS에서는 보안 패치와 새 컨테이너 에이전트 버전을 사용해 Amazon ECS 최적화 AMI 이미지를 정기적으로 업데이트합니다. 이미지 레시피에서 AMI ID를 상위 이미지로 사용하는 경우 상위 이미지에 대한 업데이트를 정기적으로 확인해야 합니다. 업데이트가 있는 경우 업데이트된 AMI를 사용하여 레시피의 새 버전을 생성해야 합니다. 그러면 사용자 지정 이미지가 상위 이미지의 최신 버전을 통합됩니다. 새로 생성된 AMI로 Amazon ECS 클러스터에서 EC2 인스턴스를 자동으로 업데이트하는 워크플로를 생성하는 방법에 대한 자세한 내용은 How to create an AMI hardening pipeline and automate updates to your ECS instance fleet을 참조하세요.

관리형 EC2 Image Builder 파이프라인을 통해 게시되는 상위 이미지의 Amazon 리소스 이름(ARN)을 지정할 수도 있습니다. Amazon은 관리형 파이프라인을 통해 Amazon ECS 최적화 AMI 이미지를 정기적으로 게시합니다. 이 이미지에는 퍼블릭 액세스가 가능합니다. 이미지에 액세스하려면 올바른 권한이 있어야 합니다. Image Builder 레시피에서 AMI 대신 이미지 ARN을 사용하는 경우 파이프라인은 실행될 때마다 상위 이미지의 최신 버전을 자동으로 사용합니다. 이 접근 방식을 사용하면 업데이트할 때마다 새 레시피 버전을 수동으로 생성할 필요가 없습니다.

코드형 인프라(IaC)에서 이미지 ARN 사용

EC2 Image Builder 콘솔 또는 코드형 인프라(예: AWS CloudFormation)나 AWS SDK를 사용하여 레시피를 구성할 수 있습니다. 레시피에서 상위 이미지를 지정하는 경우 EC2 AMI ID, Image Builder 이미지 ARN, AWS Marketplace 제품 ID 또는 컨테이너 이미지를 지정할 수 있습니다. AWS에서는 Amazon ECS 최적화 AMI의 AMI ID와 Image Builder 이미지 ARN을 모두 공개적으로 게시합니다. 다음은 이미지의 ARN 형식입니다.

```
arn:${Partition}:imagebuilder:${Region}:${Account}:image/${ImageName}/${ImageVersion}
```

`ImageVersion` 형식은 다음과 같습니다. *`major`*, *`minor`*, *`patch`*를 최신 값으로 바꿉니다.

```
<major>.<minor>.<patch>
```

major, minor, patch를 특정 값으로 바꾸거나 이미지의 버전 없는 ARN을 사용하여 파이프라인이 상위 이미지의 최신 버전을 사용해 최신 상태를 유지할 수 있습니다. 버전이 없는 ARN은 와일드카드 형식 'x.x.x'를 사용하여 이미지 버전을 나타냅니다. 이 접근 방식을 사용하면 Image Builder 서비스에서 이미지의 최신 버전을 자동으로 확인할 수 있습니다. 버전 없는 ARN을 사용하면 참조가 항상 사용 가능한 최신 이미지를 가리키므로 배포에서 이미지를 최신 상태로 유지하는 프로세스가 간소화됩니다. 콘솔에서 레시피를 생성하면 EC2 Image Builder는 상위 이미지의 ARN을 자동으로 식별합니다. IaC를 사용하여 레시피를 생성할 때 ARN을 식별하고 원하는 이미지 버전을 선택하거나 버전 없는 ARN을 사용하여 사용 가능한 최신 이미지를 표시해야 합니다. 기준에 맞는 이미지만 필터링하여 표시할 수 있도록 자동화된 스크립트를 생성하는 것이 좋습니다. 다음 Python 스크립트는 Amazon ECS 최적화 AMI 목록을 검색하는 방법을 보여줍니다.

스크립트는 두 개의 선택적 인수(owner 및 platform, 기본값: 각각 'Amazon' 및 'Windows')를 허용합니다. 소유자 인수의 유효한 값은 Self, Shared, Amazon 및 ThirdParty입니다. 플랫폼 인수의 유효한 값은 Windows 및 Linux입니다. 예를 들어, owner 인수를 Amazon으로 설정하고 platform을 Linux로 설정한 상태에서 스크립트를 실행하면 스크립트는 Amazon ECS 최적화 이미지를 포함하여 Amazon에서 게시한 이미지 목록을 생성합니다.

```
import boto3
import argparse

def list_images(owner, platform):
    # Create a Boto3 session
    session = boto3.Session()

    # Create an EC2 Image Builder client
    client = session.client('imagebuilder')

    # Define the initial request parameters
    request_params = {
        'owner': owner,
        'filters': [
            {
                'name': 'platform',
                'values': [platform]
            }
        ]
    }
```

```
    # Initialize the results list
    all_images = []

    # Get the initial response with the first page of results
    response = client.list_images(**request_params)

    # Extract images from the response
    all_images.extend(response['imageVersionList'])

    # While 'nextToken' is present, continue paginating
    while 'nextToken' in response and response['nextToken']:
        # Update the token for the next request
        request_params['nextToken'] = response['nextToken']

        # Get the next page of results
        response = client.list_images(**request_params)

        # Extract images from the response
        all_images.extend(response['imageVersionList'])

    return all_images

def main():
    # Initialize the parser
    parser = argparse.ArgumentParser(description="List AWS images based on owner and
 platform")

    # Add the parameters/arguments
    parser.add_argument("--owner", default="Amazon", help="The owner of the images.
 Default is 'Amazon'.")
    parser.add_argument("--platform", default="Windows", help="The platform type of the
 images. Default is 'Windows'.")

    # Parse the arguments
    args = parser.parse_args()

    # Retrieve all images based on the provided owner and platform
    images = list_images(args.owner, args.platform)

    # Print the details of the images
    for image in images:
        print(f"Name: {image['name']}, Version: {image['version']}, ARN:
 {image['arn']}")
```

```
if __name__ == "__main__":
    main()
```

AWS CloudFormation에서 이미지 ARN 사용

Image Builder 이미지 레시피는 출력 이미지의 의도한 구성을 달성하는 데 필요한 상위 이미지 및 구성 요소를 지정하는 청사진입니다. `AWS::ImageBuilder::ImageRecipe` 리소스를 사용합니다. `ParentImage` 값을 이미지 ARN으로 설정합니다. 파이프라인이 항상 최신 버전의 이미지를 사용하도록 원하는 이미지의 버전 없는 ARN을 사용합니다. 예를 들어 `arn:aws:imagebuilder:us-east-1:aws:image/amazon-linux-2023-ecs-optimized-x86/x.x.x`입니다. 다음 `AWS::ImageBuilder::ImageRecipe` 리소스 정의는 Amazon 관리형 이미지 ARN을 사용합니다.

```
ECSRecipe:
    Type: AWS::ImageBuilder::ImageRecipe
    Properties:
      Name: MyRecipe
      Version: '1.0.0'
      Components:
        - ComponentArn: [<The component arns of the image recipe>]
      ParentImage: "arn:aws:imagebuilder:us-east-1:aws:image/amazon-linux-2023-ecs-optimized-x86/x.x.x"
```

`AWS::ImageBuilder::ImageRecipe` 리소스에 대한 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하세요.

`AWS::ImageBuilder::ImagePipeline` 리소스의 `Schedule` 속성을 설정하여 파이프라인에서 새 이미지의 생성을 자동화할 수 있습니다. 일정에는 시작 조건과 cron 표현식이 포함됩니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 `AWS::ImageBuilder::ImagePipeline`를 참조하세요.

다음 `AWS::ImageBuilder::ImagePipeline` 예제에서 파이프라인은 매일 오전 10시 협정 세계시(UTC)에 빌드를 실행합니다. 다음 `Schedule` 값을 설정합니다.

- `PipelineExecutionStartCondition`를 `EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE`으로 설정합니다. 빌드는 시맨틱 버전에서 와일드카드 'x'를 사용하는 상위 이미지나 구성 요소와 같은 종속 리소스가 업데이트된 경우에만 시작됩니다. 이렇게 하면 빌드는 해당 리소스의 최신 업데이트를 통합합니다.
- ScheduleExpression을 cron 표현식 (`0 10 * * ? *`)로 설정합니다.

```
ECSPipeline:
    Type: AWS::ImageBuilder::ImagePipeline
    Properties:
      Name: my-pipeline
      ImageRecipeArn: <arn of the recipe you created in previous step>
      InfrastructureConfigurationArn: <ARN of the infrastructure configuration
 associated with this image pipeline>
      Schedule:
        PipelineExecutionStartCondition:
 EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE
        ScheduleExpression: 'cron(0 10 * * ? *)'
```

Terraform에서 이미지 ARN 사용

Terraform에서 파이프라인의 상위 이미지 및 일정을 지정하는 접근 방식은 AWS CloudFormation의 방식과 일관됩니다. `aws_imagebuilder_image_recipe` 리소스를 사용합니다. `parent_image` 값을 이미지 ARN으로 설정합니다. 파이프라인이 항상 최신 버전의 이미지를 사용하도록 원하는 이미지의 버전 없는 ARN을 사용합니다. 자세한 내용은 Terraform 설명서의 `aws_imagebuilder_image_recipe`을 참조하세요.

`aws_imagebuilder_image_pipeline resource`의 일정 구성 블록에서 `schedule_expression` 인수 값을 원하는 cron 표현식으로 설정하여 파이프라인 실행 빈도를 지정하고 `pipeline_execution_start_condition`을 `EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE`로 설정합니다. 자세한 내용은 Terraform 설명서의 `aws_imagebuilder_image_pipeline`을 참조하세요.

Amazon ECS에서 AWS Deep Learning Containers 사용

AWS Deep Learning Containers는 Amazon ECS의 TensorFlow 및 Apache MXNet(Incubating)에서 모델을 교육하고 제공하기 위한 Docker 이미지 세트를 제공합니다. Deep Learning Containers는 TensorFlow, NVIDIA CUDA(GPU 인스턴스용) 및 Intel MKL(CPU 인스턴스용) 라이브러리에 최적의 환경을 사용할 수 있습니다. Deep Learning Containers용 컨테이너 이미지는 Amazon ECS 태스크 정의에서 참조하기 위해 Amazon ECR에서 사용할 수 있습니다. Amazon Elastic Inference와 함께 Deep Learning Containers를 사용하면 추론 비용을 낮출 수 있습니다

Amazon ECS에서 Elastic Inference 없이 Deep Learning Containers 사용을 시작하려면 AWS Deep Learning AMIs 개발자 안내서의 Amazon ECS 설정을 참조하세요.

Amazon ECS 서비스 할당량

AWS 계정을 생성하면 AWS 리소스에 대한 기본 할당량(한도라고도 함)이 리전별로 설정됩니다. 리소스 할당량을 초과하려고 하면 요청이 실패합니다.

Service Quotas 콘솔은 AWS 서비스에 대한 할당량을 보고 관리할 수 있으며 사용하는 많은 리소스에 대한 할당량 증가를 요청할 수 있는 중앙 위치입니다. 제공되는 할당량 정보를 사용하여 리소스를 관리하세요. 실제로 필요할 시점보다 미리 할당량 증가를 요청하도록 계획하세요.

Amazon ECS 및 Fargate 할당량에 대한 자세한 내용은 Amazon Web Services 일반 참조의 Amazon ECS 엔드포인트 및 할당량에서 확인하세요. Amazon ECS API의 API 제한에 대한 자세한 내용은 Amazon ECS API에 대한 요청 제한을 참조하세요.

AWS Management Console에서 Amazon ECS 및 AWS Fargate 서비스 할당량 관리

Amazon ECS는 중앙 위치에서 할당량을 보고 관리할 수 있는 AWS 서비스인 Service Quotas와 통합됩니다. 자세한 내용은 Service Quotas 사용 설명서의 Service Quotas는 무엇입니까?를 참조하세요.

Service Quotas를 사용하면 모든 Amazon ECS 서비스 할당량의 값을 쉽게 찾을 수 있습니다.

AWS Management Console

AWS Management Console을 사용하여 Amazon ECS 및 Fargate 서비스 할당량을 보려면

1. https://console.aws.amazon.com/servicequotas/에서 Service Quotas 콘솔을 엽니다.
2. 탐색 창에서 AWS 서비스를 선택합니다.
3. AWS 서비스(services) 목록에서 Amazon Elastic Container Service(Amazon ECS)(Amazon Elastic Container Service(Amazon ECS)) 또는 AWS Fargate을(를) 검색하고 선택합니다.

   서비스 할당량(Service quotas) 목록에서 서비스 할당량 이름, 적용된 값(제공된 경우), AWS 기본 할당량 및 할당량 값 조정 가능 여부를 확인할 수 있습니다.
4. 설명 등 서비스 할당량에 대한 추가 정보를 보려면 할당량 이름을 선택합니다.
5. (선택 사항) 할당량 증가를 요청하려면 증가시킬 할당량을 선택하고 할당량 증가 요청(Request quota increase)을 선택한 다음 필요한 정보를 입력하거나 선택한 다음 요청(Request)을 선택합니다.

AWS Management Console을 사용하여 서비스 할당량에 대한 추가 태스크를 수행하려면 Service Quotas 사용 설명서를 참조하세요. 할당량 증가을 요청하려면 Service Quotas 사용 설명서의 할당량 증가 요청을 참조하세요.

AWS CLI

AWS CLI을 사용하여 Amazon ECS 및 Fargate 서비스 할당량을 보려면

다음 명령을 실행하여 기본 Amazon ECS 할당량을 확인합니다.

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code ecs \
    --output table
```

다음 명령을 실행하여 기본 Fargate 할당량을 확인합니다.

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code fargate \
    --output table
```

다음 명령을 실행하여 적용된 Fargate 할당량을 확인합니다.

```
aws service-quotas list-service-quotas \
    --service-code fargate
```

> **Note**
>
> Amazon ECS는 적용된 할당량을 지원하지 않습니다.

AWS CLI를 사용하여 서비스 할당량 작업에 대한 자세한 내용은 Service Quotas AWS CLI 명령 참조를 참조하세요. 할당량 증가를 요청하려면 AWS CLI 명령 참조에서 `request-service-quota-increase` 명령을 참조하세요.

Amazon ECS 서비스 할당량 및 API 제한 한도 처리

Amazon ECS는 Elastic Load Balancing, AWS Cloud Map, Amazon EC2를 비롯한 여러 AWS 서비스에 통합됩니다. Amazon ECS는 이러한 긴밀한 통합을 통해 서비스 로드 밸런싱, Service Connect, 작업 네트워킹, 클러스터 Auto Scaling 등의 여러 기능을 포함합니다. Amazon ECS 및 통합되는 기타 AWS 서비스는 모두 서비스 할당량 및 API 속도 제한을 유지 관리하여 일관된 성능 및 사용률을 보장합니다. 또한 이러한 서비스 할당량은 필요한 수준을 초과하여 많은 리소스가 실수로 프로비저닝되는 것을 방지하고 청구 비용을 늘릴 수 있는 악의적인 행동으로부터 보호합니다.

서비스 할당량 및 AWS API 속도 제한을 숙지하면 예상치 못한 성능 저하에 대한 걱정 없이 워크로드 규모 조정을 계획할 수 있습니다. 자세한 내용은 Amazon ECS API에 대한 요청 제한을 참조하세요.

Amazon ECS에서 워크로드 규모를 조정할 때는 다음 서비스 할당량을 고려하는 것이 좋습니다.

- AWS Fargate에는 각 AWS 리전에서 동시에 실행되는 작업 수를 제한하는 할당량이 있습니다. Amazon ECS에는 온디맨드 작업 및 Fargate 스팟 작업 모두에 대한 할당량이 있습니다. 각 서비스 할당량에는 Fargate에서 실행하는 모든 Amazon EKS 포드도 포함됩니다.
- Amazon EC2 인스턴스에서 실행되는 작업의 경우 각 클러스터에 등록할 수 있는 Amazon EC2 인스턴스의 최대 개수는 5,000개입니다. Auto Scaling 그룹 용량 공급자와 함께 Amazon ECS 클러스터 Auto Scaling을 사용하거나 클러스터의 Amazon EC2 인스턴스를 자체적으로 관리하는 경우 이 할당량으로 인해 배포 병목 현상이 발생할 수 있습니다. 더 많은 용량이 필요한 경우 클러스터를 생성하거나 서비스 할당량 증가를 요청할 수 있습니다.
- Auto Scaling 그룹 용량 공급자와 함께 Amazon ECS 클러스터 Auto Scaling을 사용하는 경우 서비스 규모를 확장할 때 `Tasks in the PROVISIONING state per cluster` 할당량을 고려합니다. 이 할당량은 각 클러스터에서 용량 공급자가 용량을 늘릴 수 있는 `PROVISIONING` 상태의 최대 작업 수입니다. 동시에 많은 작업을 시작하면 이 할당량을 쉽게 충족할 수 있습니다. 한 가지 예로, 각각 수백 개의 작업을 포함하는 수십 개의 서비스를 동시에 배포하는 경우가 있습니다. 이 경우 용량 공급자는 클러스터의 용량이 부족할 때 새 컨테이너 인스턴스를 시작하여 작업을 배치해야 합니다. 용량 공급자가 추가 Amazon EC2 인스턴스를 시작하는 동안에도 Amazon ECS 서비스 스케줄러는 작업을 계속 병렬로 시작할 수 있습니다. 하지만 클러스터 용량이 충분하지 않아 이 활동이 제한될 수 있습니다. Amazon ECS 서비스 스케줄러는 새 컨테이너 인스턴스가 시작될 때 작업 배치를 재시도하기 위한 백오프 및 지수 제한 전략을 구현합니다. 따라서 배포 또는 스케일 아웃 시간이 느려질 수 있습니다. 이러한 상황을 방지하려면 다음 중 하나로 서비스 배포를 계획할 수 있습니다. 클러스터 용량을 늘릴 필요 없이 많은 작업을 배포하거나 새 작업을 시작할 수 있도록 예비 클러스터 용량을 확보합니다.

워크로드 규모를 조정할 때 Amazon ECS 서비스 할당량을 고려하는 것 외에도 Amazon ECS와 통합된 다른 AWS 서비스 서비스의 서비스 할당량도 고려합니다.

Elastic Load Balancing

Elastic Load Balancing을 사용하여 작업 전체에 균일하게 트래픽 분산하도록 Amazon ECS 서비스를 구성할 수 있습니다. 로드 밸런서를 선택하는 방법에 대한 자세한 내용 및 권장 모범 사례는 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산 섹션을 참조하세요.

Elastic Load Balancing 서비스 할당량

워크로드 규모를 조정할 때는 다음과 같은 Elastic Load Balancing 서비스 할당량을 고려하세요. 대부분의 Elastic Load Balancing 서비스 할당량은 조정 가능하며 Service Quotas 콘솔에서 증가를 요청할 수 있습니다.

Application Load Balancer

Application Load Balancer를 사용할 때는 사용 사례에 따라 다음에 대한 할당량 증가를 요청해야 할 수 있습니다.

- `Targets per Application Load Balancer` 할당량: Application Load Balancer 뒤에 있는 대상의 수입니다.
- `Targets per Target Group per Region` 할당량: 대상 그룹 뒤에 있는 대상의 수입니다.

자세한 내용을 알아보려면 Application Load Balancer 사용 설명서의 Quotas for your Application Load Balancers를 참조하세요.

Network Load Balancer

Network Load Balancer에 등록할 수 있는 대상 수에는 더 엄격한 제한이 적용됩니다. Network Load Balancer를 사용할 때 종종 교차 영역 지원을 활성화하려고 합니다. 이 경우 각 Network Load Balancer의 가용 영역당 최대 대상 수와 함께 `Targets per Availability Zone Per Network Load Balancer`의 추가 조정 제한이 적용됩니다. 자세한 내용은 Network Load Balancers 사용 설명서의 Quotas for your Network Load Balancers를 참조하세요.

Elastic Load Balancing API 제한

Amazon ECS 서비스가 로드 밸런서를 사용하도록 구성하는 경우 대상 그룹 상태 확인을 통과해야 서비스가 정상으로 간주됩니다. 이러한 상태 확인을 수행하기 위해 Amazon ECS는 사용자를 대신하여

Elastic Load Balancing API 작업을 간접 호출합니다. 계정에 로드 밸런서로 구성된 서비스가 많으면 특히 `RegisterTarget`, `DeregisterTarget`, `DescribeTargetHealth` Elastic Load Balancing API 작업에서 제한이 발생할 수 있으므로 서비스 배포 속도가 느려질 수 있습니다. 제한이 발생하면 Amazon ECS 서비스 이벤트 메시지에서 제한 오류가 발생합니다.

AWS Cloud Map API 제한이 발생하는 경우 지원에 AWS Cloud Map API 제한 한도 상향에 지침을 문의할 수 있습니다. 이러한 제한 오류의 모니터링 및 문제 해결에 대한 자세한 내용은 Amazon ECS 제한 문제 처리 섹션을 참조하세요.

탄력적 네트워크 인터페이스

작업이 `awsvpc` 네트워크 모드를 사용하는 경우, Amazon ECS는 각 작업에 대해 고유한 탄력적 네트워크 인터페이스(ENI)를 프로비저닝합니다. Amazon ECS 서비스가 Elastic Load Balancing 로드 밸런서를 사용하는 경우 이러한 네트워크 인터페이스도 서비스에 정의된 적절한 대상 그룹에 대한 대상으로 등록됩니다.

탄력적 네트워크 인터페이스 서비스 할당량

`awsvpc` 네트워크 모드를 사용하는 작업을 실행하면 고유한 탄력적 네트워크 인터페이스가 각 작업에 연결됩니다. 인터넷을 통해 이러한 작업에 도달해야 할 경우에는 해당 작업의 탄력적 네트워크 인터페이스에 퍼블릭 IP 주소를 할당합니다. Amazon ECS 워크로드 규모를 조정할 때는 다음과 같은 두 가지 중요한 할당량을 고려합니다.

- `Network interfaces per Region` 할당량: 계정의 AWS 리전에 있는 최대 네트워크 인터페이스 수입니다.
- `Elastic IP addresses per Region` 할당량: AWS 리전에 있는 최대 탄력적 IP 주소 수입니다.

이 두 서비스 할당량은 모두 조정 가능하며, Service Quotas 콘솔에서 할당량 증가를 요청할 수 있습니다. 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 Amazon VPC service quotas를 참조하세요.

Amazon EC2 인스턴스에 호스팅되는 Amazon ECS 워크로드의 경우 `awsvpc` 네트워크 모드를 사용하는 작업을 실행할 때는 각 Amazon EC2 인스턴스의 최대 네트워크 인스턴스 수에 해당하는 `Maximum network interfaces` 서비스 할당량을 고려합니다. 이 할당량은 인스턴스에 배치할 수 있는 작업 수를 제한합니다. 할당량은 조정할 수 없으며 Service Quotas 콘솔에서 사용할 수 없습니다. 자세한 내용은 Amazon EC2 사용 설명서의 IP addresses per network interface per instance type를 참조하세요.

Amazon EC2 인스턴스에 연결할 수 있는 네트워크 인터페이스 수를 변경할 수는 없지만 탄력적 네트워크 인터페이스 트렁킹 기능을 사용하여 사용 가능한 네트워크 인터페이스 수를 늘릴 수 있습니다. 예

를 들어 c5.large 인스턴스는 최대 3개의 네트워크 인터페이스를 보유할 수 있습니다. 인스턴스에 대한 기본 네트워크 인터페이스는 한 개로 계산됩니다. 따라서 인스턴스에 네트워크 인터페이스를 2개 더 연결할 수 있습니다. awsvpc 네트워크 모드를 사용하는 각 작업에 네트워크 인터페이스가 필요하므로 대개 이 인스턴스 유형에서는 이러한 작업을 2개만 실행할 수 있습니다. 이로 인해 클러스터 용량 사용률이 낮아질 수 있습니다. 탄력적 네트워크 인터페이스 트렁킹을 활성화하면 네트워크 인터페이스 밀도를 높여 각 인스턴스에 더 많은 작업을 배치할 수 있습니다. 트렁킹을 켜면 c5.large 인스턴스는 최대 12개의 네트워크 인터페이스를 보유할 수 있습니다. 인스턴스에는 기본 네트워크 인터페이스가 있으며, Amazon ECS는 '트렁크' 네트워크 인터페이스를 생성하고 컨테이너 인스턴스에 연결합니다. 따라서 이 구성을 사용하면 인스턴스에서 기본 2개의 작업 대신 10개의 작업을 실행할 수 있습니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 네트워크 인터페이스 증가 섹션을 참조하세요.

탄력적 네트워크 인터페이스 API 제한

awsvpc 네트워크 모드를 사용하는 작업을 실행할 때 Amazon ECS는 다음 Amazon EC2 API에 의존합니다. 이러한 API마다 API 제한이 서로 다릅니다. 자세한 내용은 Amazon EC2 API Reference의 Request throttling for the Amazon EC2 API를 참조하세요.

- CreateNetworkInterface
- AttachNetworkInterface
- DetachNetworkInterface
- DeleteNetworkInterface
- DescribeNetworkInterfaces
- DescribeVpcs
- DescribeSubnets
- DescribeSecurityGroups
- DescribeInstances

탄력적 네트워크 인터페이스 프로비저닝 워크플로 중에 Amazon EC2 API 직접 호출이 제한되는 경우 Amazon ECS 서비스 스케줄러는 지수 백오프를 통해 자동으로 재시도합니다. 이러한 자동 사용 중지로 인해 때때로 작업 시작이 지연되어 배포 속도가 느려질 수 있습니다. API 제한이 발생하면 서비스 이벤트 메시지에 Operations are being throttled. Will try again later. 메시지가 표시됩니다. Amazon EC2 API 제한을 일관되게 충족하는 경우 API 제한 한도 상향 방법에 대한 지침을 지원에 문의할 수 있습니다. 제한 오류의 모니터링 및 문제 해결에 대한 자세한 내용은 제한 문제 처리를 참조하세요.

AWS Cloud Map

Amazon ECS 서비스 검색 및 Service Connect는 AWS Cloud Map API를 사용하여 Amazon ECS 서비스에 대한 네임스페이스를 관리합니다. 서비스에 작업 수가 많은 경우 다음 권장 사항을 고려하세요.

AWS Cloud Map service quotas

Amazon ECS 서비스가 서비스 검색 또는 Service Connect를 사용하도록 구성된 경우 서비스의 최대 작업 수에 해당하는 `Tasks per service` 할당량은 해당 서비스의 최대 인스턴스 수인 AWS Cloud Map `Instances per service` 서비스 할당량의 영향을 받습니다. 특히 AWS Cloud Map 서비스 할당량은 서비스에서 실행할 수 있는 인스턴스 수를 서비스당 최대 1,000개 인스턴스로 줄입니다. AWS Cloud Map 할당량은 변경할 수 없습니다. 자세한 내용은 AWS Cloud Map 서비스 할당량을 참조하세요.

AWS Cloud Map API 조절

Amazon ECS는 사용자를 대신하여 `ListInstances`, `GetInstancesHealthStatus`, `RegisterInstance` 및 `DeregisterInstance` AWS Cloud Map API를 직접 호출합니다. 이를 통해 서비스 검색을 지원하고 작업을 시작할 때 상태 확인을 수행할 수 있습니다. 많은 작업에서 서비스 검색을 사용하는 여러 서비스를 동시에 배포하는 경우, 이로 인해 AWS Cloud Map API 제한 한도가 초과될 수 있습니다. 이 경우 Amazon ECS 서비스 이벤트 메시지에 `Operations are being throttled. Will try again later` 메시지가 표시되고, 배포 및 작업 시작 속도가 느려집니다. AWS Cloud Map에서는 이러한 API의 제한 한도를 문서화하지 않습니다. 이로 인해 제한이 발생하는 경우 지원에 API 제한 한도 상향에 지침을 문의할 수 있습니다. 이러한 제한 오류의 모니터링 및 문제 해결에 대한 자세한 권장 사항은 Amazon ECS 제한 문제 처리 섹션을 참조하세요.

Amazon ECS API 참조

Amazon ECS는 AWS Management Console 및 AWS Command Line Interface(AWS CLI) 외에 API도 제공합니다. 이 API를 사용하여 Amazon ECS 리소스 관리 작업을 자동화할 수 있습니다.

- Amazon ECS 리소스별 API 작업 목록은 Actions by Amazon ECS resource를 참조하세요.
- API 작업의 알파벳순 목록은 작업을 참조하십시오.
- 데이터 형식에 대한 알파벳순 목록은 데이터 형식을 참조하십시오.
- 공통 쿼리 파라미터 목록은 공통 파라미터를 참조하십시오.
- 오류 코드에 대한 설명은 공통 오류를 참조하십시오.

AWS CLI에 대한 자세한 내용은 AWS Command Line Interface reference for Amazon ECS를 참조하세요.

문서 기록

다음 표에서 Amazon Elastic Container Service 개발자 안내서의 중요한 업데이트 및 새 기능이 나와 있습니다. 사용자로부터 받은 의견을 수렴하기 위해 설명서가 자주 업데이트됩니다.

변경 사항	설명	날짜
AmazonEC2ContainerServiceforEC2Role 관리형 정책에 권한을 추가합니다.	`ecs:ListTagsForResource` 권한을 포함하도록 `AmazonEC2ContainerServiceforEC2Role` 관리형 IAM 정책이 업데이트되었습니다. 이 권한을 통해 Amazon ECS 에이전트는 태스크 메타데이터 엔드포인트(`${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags`)를 통해 태스크 및 컨테이너 인스턴스 태그를 검색할 수 있습니다. 자세한 내용은 AmazonEC2ContainerServiceforEC2Role을 참조하세요.	2025년 8월 4일
AmazonECSInfrastructureRolePolicyForLoadBalancers에 권한을 추가합니다.	`AmazonECSInfrastructureRolePolicyForLoadBalancers` 관리형 IAM 정책이 대상 그룹을 설명, 등록, 등록 취소하기 위한 새 권한으로 업데이트되었습니다.	2025년 7월 25일
서비스 배포 컨트롤러 업데이트에 대한 지원	Amazon ECS는 서비스 배포 컨트롤러 업데이트를 지원하므로 기존 서비스에서 서로 다른 배포 유형 사이를 변경할 수 있습니다. 자세한 내용은 Amazon ECS 서비스 파라미터 업데이트를 참조하세요.	2025년 7월 15일
Amazon ECS 기본 블루/그린 배포	Amazon ECS는 기본 블루/그린 배포를 지원하므로 프로덕션 트래픽을 지정하기 전에 새 서비스 개정을 검증할 수 있습니다. 자세한 내용은 Amazon ECS 블루/그린 배포를 참조하세요.	2025년 7월 15일
새 AmazonECSInfrastructureRolePolicyForLoadBalancers 관리형 정책을 추가합니다.	이 정책에서는 사용자를 대신해 Amazon ECS 워크로드와 연결된 로드 밸런서를 관리하는 데 필요한 다른 AWS 서비스 리소스에 대한 액세스를 제공합니다. 자세한 내용은 AmazonECSInfrastructureRolePolicyForLoadBalancers를 참조하세요.	2025년 7월 15일

변경 사항	설명	날짜
SOCI 인덱스 매니페스트 v2 지원	SOCI를 사용하려는 Fargate 고객은 SOCI 인덱스 매니페스트 v2만 사용할 수 있습니다. 이전에 Fargate에서 SOCI를 사용한 기존 고객은 SOCI 인덱스 매니페스트 v1을 계속 사용할 수 있지만, SOCI 인덱스 매니페스트 v2로 마이그레이션하는 것이 좋습니다. 자세한 내용은 Seekable OCI(SOCI)를 사용하여 컨테이너 이미지 지연 로딩을 참조하세요.	2025년 7월 2일
상태 실패로 인해 생성된 서비스 작업 이벤트에 태스크 ID가 추가되었습니다.	자세한 내용은 Amazon ECS 서비스 비정상 이벤트 메시지를 참조하세요.	2025년 6월 30일
기본 로그 드라이버 모드로 업데이트합니다.	Amazon ECS의 기본 로깅 모드가 `blocking`에서 `non-blocking` 으로 업데이트되었습니다. 컨테이너 정의의 `logConfiguration` 에 `mode`를 지정하여 이 기본 모드를 재정의하거나 Amazon ECS 콘솔 또는 AWS CLI를 통해 `defaultLogDriverMode` 계정 설정을 설정하여 고유한 기본값을 구성할 수 있습니다. 자세한 내용은 Default log driver mode를 참조하세요.	2025년 6월 25일
AmazonECSServiceRolePolicy에 권한을 추가합니다.	`AmazonECSServiceRolePolicy` 관리형 IAM 정책은 Amazon ECS가 관리하는 서비스의 AWS Cloud Map 서비스 속성을 업데이트할 수 있는 새 AWS Cloud Map 권한으로 업데이트되었습니다.	2025년 6월 24일
Amazon ECS 최적화 Windows Server 2025 Full AMI 및 Amazon ECS 최적화 Windows Server 2025 Core AMI를 지원합니다.	자세한 내용은 Amazon ECS 최적화 Windows AMI를 참조하세요.	2025년 6월 20일
서비스 용량 제공자 전략 업데이트를 지원합니다.	이제 서비스 용량 제공자 전략을 업데이트할 수 있습니다. 자세한 내용은 Update service parameters를 참조하세요.	2025년 6월 12일

변경 사항	설명	날짜
AmazonECSInfrastructureRolePolicyForVolumes에 권한 추가	AmazonECSInfrastructureRolePolicyForVolumes 정책이 업데이트되어 ec2:DescribeInstances 권한이 추가되었습니다. 이 권한은 동일한 컨테이너 인스턴스에서 실행되는 Amazon ECS 태스크에 연결된 Amazon EBS 볼륨의 디바이스 이름 충돌을 방지하는 데 도움이 됩니다.	2025년 6월 2일
Amazon ECS 태스크에 연결된 Amazon EBS 볼륨에서 볼륨 초기화 속도를 지원합니다.	이제 volumeIntiializationRate 를 설정하여 Amazon ECS 태스크에 연결할 새 볼륨을 생성하기 위해 기존 Amazon EBS 볼륨의 스냅샷에서 데이터를 가져오는 속도를 MiB/s 단위로 결정할 수 있습니다. 자세한 내용은 volumeInitializationRate를 참조하세요.	2025년 5월 13일
서비스 배포 중지를 지원합니다.	이제 서비스 배포를 중지할 수 있습니다. 자세한 내용은 Amazon ECS 서비스 배포 중지를 참조하세요.	2025년 5월 2일
Amazon ECS 듀얼 스택 엔드포인트를 지원합니다.	이제 듀얼 스택 엔드포인트를 사용하여 Amazon ECS와 상호 작용할 수 있습니다. 듀얼 스택 엔드포인트는 인터넷 프로토콜 버전 4(IPv4) 및 인터넷 프로토콜 버전 6(IPv6) 모두를 통해 Amazon ECS에 대한 요청을 지원합니다. 자세한 내용은 Amazon ECS 듀얼 스택 엔드포인트 사용을 참조하세요.	2025년 4월 30일
AWS 리전의 계정 수준에서 기본 로그 드라이버 모드 구성을 지원합니다.	이제 defaultLogDriverMode 계정 설정을 사용하여 컨테이너의 로그 메시지를 logConfiguration 에서 지정한 로그 드라이버로 전달하는 기본 전송 모드를 구성할 수 있습니다. 자세한 내용은 Default log driver mode를 참조하세요.	2025년 4월 17일
짧은 ARN 형식의 서비스를 긴 ARN 형식으로 마이그레이션하도록 지원합니다.	이제 짧은 ARN 형식의 서비스를 긴 ARN 형식의 패킷으로 마이그레이션할 수 있습니다. 자세한 내용은 Amazon ECS 단기 서비스 ARN을 긴 ARN으로 마이그레이션을 참조하세요.	2025년 2월 13일

변경 사항	설명	날짜
Amazon ECS 및 Fargate에서 세 가지 유형의 결함 주입을 지원합니다.	이제 Amazon ECS 및 Fargate에서 네트워크 블랙홀 포트, 네트워크 지연 시간, 네트워크 패킷 손실 결함 주입을 사용할 수 있습니다. 자세한 내용은 UseAWS Fault Injection Service to test your Amazon ECS workloads를 참조하세요.	2024년 12월 19일
관찰성이 향상된 CloudWatch Container Insights를 지원합니다.	관찰성이 향상된 CloudWatch Container Insights를 사용할 수 있습니다. 자세한 내용은 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링를 참조하세요.	2024년 12월 2일
예측 규모 조정을 지원합니다.	예측 규모 조정을 사용하여 향후 필요성을 예측하고 필요에 따라 서비스의 태스크 미리 늘릴 수 있습니다. 자세한 내용은 과거 패턴을 사용하여 예측 규모 조정으로 CloudWatch 서비스 규모 조정을 참조하세요.	2024년 11월 21일
서비스 리밸런싱을 지원합니다.	이제 Amazon ECS에서 가용 영역 전체의 서비스 태스크 밸런스를 자동으로 조정되도록 할 수 있습니다. 자세한 내용은 가용 영역 전체의 Amazon ECS 서비스 밸런싱 섹션을 참조하세요.	2024년 11월 19일
컨테이너 이미지 태그의 해상도를 이미지 다이제스트에 알맞게 구성하는 것이 지원됩니다.	이제 서비스의 일부인 태스크에서 컨테이너 이미지 태그의 해상도를 각 컨테이너의 다이제스트에 알맞게 구성할 수 있습니다. 자세한 내용은 컨테이너 이미지 확인 및 `versionConsistency` (을)를 참조하세요.	2024년 11월 19일
새 `AmazonECSInfrastructureRolePolicyForVpcLattice` IAM 정책 추가	`AmazonECSInfrastructureRolePolicyForVpcLattice` 정책에서는 사용자를 대신하여 Amazon ECS 워크로드에서 VPC Lattice 특성을 관리하는 데 필요한 다른 AWS 서비스 리소스에 대한 액세스 권한이 제공됩니다.	2024년 11월 18일

변경 사항	설명	날짜
이제 Amazon ECS 서비스에 VPC Lattice를 사용할 수 있습니다.	이제 VPC Lattice를 사용하여 서비스 간 연결, 보안 및 관찰성을 표준화할 수 있습니다. 자세한 내용은 Amazon VPC Lattice를 사용하여 Amazon ECS 서비스 연결, 관찰 및 보안 유지를 참조하세요.	2024년 11월 18일
서비스 배포 및 서비스 개정을 사용하여 서비스 기록을 지원합니다.	서비스 배포 및 서비스 개정을 사용하여 배포 기록을 볼 수 있습니다. 자세한 내용은 Amazon ECS 서비스 배포를 사용하여 서비스 기록 보기와 Amazon ECS 서비스 개정을 참조하세요.	2024년 10월 30일
Windows 운영 체제를 사용하여 EC2 시작 유형에 대한 Amazon EBS 볼륨에 대한 지원.	Windows 운영 체제를 사용하여 EC2 인스턴스에서 실행되는 태스크에 연결된 Amazon EBS 볼륨에 대한 Amazon ECS 지원. 자세한 내용은 Amazon ECS에서 Amazon EBS 볼륨 사용을 참조하세요.	2024년 10월 24일
AmazonECSInfrastructureRolePolicyForVolumes에 대한 권한이 추가되었습니다.	고객이 스냅샷에서 Amazon EBS 볼륨을 생성할 수 있도록 `AmazonECSInfrastructureRolePolicyForVolumes` 정책이 업데이트되었습니다. 자세한 내용은 AmazonECSInfrastructureRolePolicyForVolumes를 참조하세요.	2024년 10월 10일
Fargate 스팟은 리눅스 ARM에서 사용할 수 있습니다.	리눅스 ARM의 Fargate Spot에 대한 지원이 Amazon ECS에 추가되었습니다.	2024년 9월 6일
Amazon ECS 작업의 개별 컨테이너에 대한 컨테이너 재시작 정책.	작업 정의에 정의된 필수 및 비필수 컨테이너에 대한 재시작 정책을 사용하여 일시적 장애를 더 빠르게 극복하고 작업 가용성을 유지할 수 있습니다. 자세한 내용은 컨테이너 재시작 정책이 있는 Amazon ECS 작업의 개별 컨테이너 재시작 섹션을 참조하세요.	2024년 8월 15일

변경 사항	설명	날짜
the section called “AmazonECS_FullAccess”에 권한 추가	`ecsInfrastructureRole` 이라는 역할에 대한 IAM 역할 권한 `iam:PassRole` 을 추가하도록 `AmazonECS_FullAccess` 정책이 업데이트되었습니다. 이 역할은 AWS Management Console에서 생성한 기본 IAM 역할로, Amazon ECS가 ECS 작업에 연결된 Amazon EBS 볼륨을 관리할 수 있도록 하는 ECS 인프라 역할로 사용하기 위한 것입니다.	2024년 8월 13일
ECS Anywhere에 대한 Amazon Linux 2023, CentOS Stream 9, Fedora 40 및 Ubuntu 24 지원 추가	ECS Anywhere에 Amazon Linux 2023, CentOS Stream 9, Fedora 40 및 Ubuntu 24 운영 체제에 대한 지원이 추가되었습니다. 자세한 내용은 지원되는 운영 체제 및 시스템 아키텍처 섹션을 참조하세요.	2024년 7월 23일
롤링 업데이트(ECS) 배포 유형을 사용하는 서비스의 컨테이너 이미지 확인입니다.	롤링 업데이트 배포 컨트롤러를 사용하는 서비스의 모든 작업이 동일한 컨테이너 이미지를 사용하도록 보장하기 위해 Amazon ECS에서는 태스크 정의에 지정된 컨테이너 이미지 이름과 모든 이미지 태그를 컨테이너 이미지 다이제스트로 확인합니다. 자세한 내용은 컨테이너 이미지 확인을 참조하세요.	2024년 7월 10일
ECS Anywhere에 대한 Debian 11 및 Debian 12 지원 추가	ECS Anywhere에 Debian 11 및 Debian 12 운영 체제 지원이 추가되었습니다. 자세한 내용은 지원되는 운영 체제 및 시스템 아키텍처 섹션을 참조하세요.	2024년 3월 28일
Fargate의 Linux 컨테이너용 gMSA 지원	Amazon ECS는 그룹 관리형 서비스 계정(gMSA)이라는 특수한 종류의 서비스 계정을 통해 Fargate의 Linux 컨테이너에 대한 Active Directory 인증을 지원합니다. 자세한 내용은 Fargate의 Linux 컨테이너에 대해 gMSA 사용을 참조하세요.	2024년 3월 5일
작업에 연결된 Amazon EBS 볼륨에 대해 CloudWatch 지표 추가됨	이제 Amazon ECS는 Amazon EBS 볼륨이 연결된 작업의 Amazon EBS 스토리지 사용률에 대한 CloudWatch 지표를 게시합니다. 자세한 내용은 Amazon ECS CloudWatch 지표를 참조하세요.	2024년 2월 8일

변경 사항	설명	날짜
Service Connect TLS	이제 Service Connect에서 TLS를 사용할 수 있습니다.	2024년 1월 22일
Service Connect TLS 관리형 정책	새 AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity 정책이 추가되었습니다.	2024년 1월 22일
Service Connect 제한 시간 구성 업데이트	이제 Service Connect 제한 시간 구성을 업데이트할 수 있으며, 여기에 두 개의 선택적 파라미터(`idleTimeout` 및 `perRequestTimeout`)가 포함됩니다.	2024년 1월 22일
Amazon ECS 관리형 인스턴스 드레이닝	Amazon ECS 관리형 인스턴스 드레이닝을 사용하여 Amazon ECS 인스턴스를 정상적으로 종료할 수 있습니다.	2024년 1월 19일
ECS Anywhere에 대해 Ubuntu 22 지원 추가됨	ECS Anywhere에 Ubuntu 22 운영 체제 지원이 추가되었습니다. 자세한 내용은 지원되는 운영 체제 및 시스템 아키텍처 섹션을 참조하세요.	2024년 1월 16일
`AmazonECSInfrastructureRolePolicyForVolumes` IAM 정책 추가	`AmazonECSInfrastructureolePolicyForVolumes` 이 추가되었습니다. 정책에서는 Amazon ECS가 Amazon ECS 워크로드에 연결된 Amazon EBS 볼륨을 관리하기 위해 AWS API 직접 호출을 수행하는 데 필요한 권한을 부여합니다.	2024년 1월 11일
Amazon ECS 작업을 위한 Amazon EBS 데이터 볼륨	독립 실행형 Amazon ECS 작업 또는 ECS 서비스에서 관리하는 작업에 연결하기 위해 배포 중 작업당 1개의 Amazon EBS 데이터 볼륨을 구성할 수 있습니다. 배포 시 볼륨을 구성하면 특정 볼륨 유형이나 설정으로 제한되지 않는 재사용이 가능한 작업 정의를 생성할 수 있습니다. Amazon EBS 볼륨은 데이터 집약적인 컨테이너화된 워크로드에 대해 가용성이 높고 비용 효율적이며 내구성이 뛰어난 고성능 블록 스토리지를 제공합니다.	2024년 1월 11일
Amazon ECS 클래식 콘솔 수명 종료됨	Amazon ECS 콘솔의 수명이 종료되었습니다.	2023년 12월 4일

변경 사항	설명	날짜
정책 업데이트	AmazonECSServiceRolePolicy 관리형 IAM 정책이 새 events 권한과 추가적인 autoscaling 및 autoscaling-plans 권한으로 업데이트되었습니다.	2023년 12월 4일
Runtime Monitoring 지원	Runtime Monitoring을 통해 Amazon ECS 워크로드를 지속적으로 모니터링하여 악의적 또는 무단 동작을 식별할 수 있습니다. 자세한 내용은 Runtime Monitoring을 참조하세요.	2023년 11월 26일
정책 업데이트	AmazonECSServiceRolePolicy 관리형 IAM 정책이 AWS Cloud Map DiscoverInstancesRevision API에 대한 액세스를 허용하도록 업데이트되었습니다.	2023년 10월 4일
AWS Fargate 작업 사용 중지 구성	Fargate 작업이 사용 중지되기 전의 대기 기간을 구성할 수 있습니다. 자세한 내용은 AWS Fargate task maintenance를 참조하세요.	2023년 9월 5일
AWS Fargate의 추가 작업 정의 파라미터	AWS Fargate는 Linux 플랫폼 버전 1.4.0에서 pidMode 및 systemControls 에 대한 지원을 추가합니다. 자세한 내용은 Task definitions를 참조하세요.	2023년 8월 9일
Amazon ECS 콘솔 작업 정의 페이지 다시 설계	Amazon ECS 콘솔의 작업 정의 페이지가 새롭게 디자인되어 추가 옵션을 포함합니다. 자세한 내용은 Creating a task definition using the console을 참조하세요.	2023년 7월 26일
Fargate에서 Seekable OCI 인덱스를 사용한 지연 로딩 지원	AWS Fargate에서 Seekable OCI(SOCI) 인덱스를 도입합니다. SOCI를 사용하면 컨테이너가 이미지 풀에 몇 초만 소비하고 시작할 수 있으므로, 이미지가 백그라운드에서 다운로드되는 동안 환경 설정 및 애플리케이션 인스턴스화에 필요한 시간을 얻을 수 있습니다. 자세한 내용은 Seekable OCI(SOCI)를 사용하여 컨테이너 이미지 지연 로딩을 참조하세요.	2023년 7월 17일

변경 사항	설명	날짜
Linux 및 Windows에서 gMSA 지원 개선	작업 정의에 Linux 및 Windows용 gMSA에 대한 새 `credentialSpecs` 필드가 있습니다. Windows의 도메인 없는 gMSA에 대한 전체 자습서가 새로 추가되었습니다. Tutorial: Using Windows Containers with Domainless gMSA using the AWS CLI를 참조하세요. 자세한 내용은 Using gMSAs for Linux Containers 및 Using gMSAs for Windows Containers를 참조하세요.	2023년 7월 14일
ECS 에이전트 버전 설명서 개선	Amazon ECS 에이전트 버전에 대한 설명서가 업데이트되었습니다. `v20.10.13` 버전 이상의 Docker와 Amazon ECS 컨테이너 에이전트 최신 버전을 사용하는 것이 좋습니다. 에이전트의 릴리스 버전 및 변경 사항은 GitHub에서 확인할 수 있습니다. 자세한 내용은 Amazon ECS Linux container agent versions를 참조하세요.	2023년 6월 20일
Fargate ARM64 지원을 위한 리전 가용성 업데이트	Fargate ARM64 지원을 위한 리전 가용성이 업데이트되었습니다. 자세한 내용은 고려 사항 섹션을 참조하세요.	2023년 6월 19일
클러스터 Auto Scaling 설명서 개선	Amazon EC2 Auto Scaling의 Amazon ECS 조정에 대한 설명서의 정확성과 가독성이 크게 향상되었습니다. 자세한 내용은 Deep dive on Amazon ECS cluster auto scaling을 참조하세요.	2023년 5월 4일
리소스 생성을 위한 태그 지정 권한 부여	사용자는 `ecsCreateCluster` 와 같이 리소스를 생성하는 작업을 위한 권한이 있어야 합니다. 리소스를 생성하고 해당 리소스에 대한 태그를 지정하면 AWS는 추가 권한 부여를 수행하여 태그를 생성할 권한이 있는지 확인합니다. 자세한 내용은 Tagging authorization 및 Grant permission to tag resources on creation을 참조하세요.	2023년 4월 18일
EC2에서 Linux 컨테이너용 gMSA 지원	gMSA를 사용하여 EC2의 Linux 컨테이너용 Active Directory에 인증할 수 있습니다. 자세한 내용은 Using gMSAs for Linux Containers를 참조하세요.	2023년 4월 14일

변경 사항	설명	날짜
AWS Fargate에서 Windows 컨테이너용 임시 스토리지 지원	AWS Fargate에서 Windows 컨테이너용 임시 스토리지를 사용할 수 있습니다. 자세한 내용은 Fargate task storage를 참조하세요.	2023년 4월 14일
AWS Cost Management에서 작업 수준 CUR 데이터 지원	비용 및 사용 보고서에서 작업 수준 비용 및 리소스 사용을 켤 수 있습니다. 그러면 AWS Fargate 및 EC2에서 실행되는 작업에 대한 비용 할당 데이터 분할이 추가됩니다. 자세한 내용은 작업 수준 비용 및 사용 보고서 섹션을 참조하세요.	2023년 8월 12일
Amazon Linux 2023 Amazon ECS 최적화 AMI	Amazon Linux 2023 Amazon ECS 최적화 AMI에 워크로드를 배포할 수 있습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2023년 4월 10일
AWS Fargate Federal Information Processing Standard(FIPS-140)	FIPS(Federal Information Processing Standard) 140을 준수하는 방식으로 AWS Fargate 기반 Amazon ECS에서 워크로드를 배포할 수 있습니다. 자세한 내용은 AWS Fargate Federal Information Processing Standard(FIPS-140) 섹션을 참조하세요.	2023년 4월 10일
작업 정의 삭제	Amazon ECS 콘솔, SDK, AWS CLI를 사용하여 작업 정의를 삭제할 수 있습니다. 자세한 내용은 Deleting a task definition revision using the console 및 Task definitions를 참조하세요.	2023년 2월 24일
Compute Optimizer의 AWS Fargate 서비스 권장 사항	AWS Compute Optimizer는 AWS Fargate의 Amazon ECS 서비스에서 실행 중인 작업의 사용률을 기반으로 작업 및 컨테이너 크기 권장 사항을 생성합니다. 자세한 내용은 Viewing recommendations for Amazon ECS services on Fargate를 참조하세요.	2023년 1월 27일
Amazon ECS 콘솔	이제 새 환경이 Amazon ECS 콘솔의 기본 환경입니다. 자세한 내용은 새 Amazon ECS 콘솔을 참조하세요.	2023년 1월 19일

변경 사항	설명	날짜
업데이트된 AmazonECS_FullAccess IAM 정책	생성 중에 로드 밸런서에 태그를 추가할 수 있는 권한을 포함하도록 AmazonECS_FullAccess IAM 정책이 업데이트되었습니다. 자세한 내용은 AmazonECS_FullAccess 섹션을 참조하세요.	2023년 1월 4일
CloudWatch 경보를 사용하여 Amazon ECS 서비스 배포 실패 탐지	지정된 CloudWatch 경보가 ALARM 상태로 전환된 것을 탐지하면 배포를 실패로 설정하도록 Amazon ECS를 구성할 수 있습니다. 자세한 내용은 the section called “롤링 업데이트 배포” 섹션을 참조하세요.	2022년 12월 19일
컨테이너 포트 매핑 지원	동적으로 매핑된 호스트 포트 범위에 바인딩된 컨테이너의 포트 번호 범위를 설정할 수 있습니다. 자세한 내용은 the section called “포트 매핑” 섹션을 참조하세요.	2022년 12월 15일
Amazon ECS Service Connect 정식 출시	이 기능은 Amazon ECS 서비스 배포로 제어되는 서비스 검색 및 서비스 메시를 추가합니다. 자세한 내용은 the section called “ Service Connect” 섹션을 참조하세요.	2022년 11월 27일
태스크 정의에 대한 새로운 Amazon ECS 콘솔 환경이 업데이트됨	새로운 Amazon ECS 콘솔 환경은 이제 태스크 정의를 위한 JSON 편집기를 포함합니다. 자세한 내용은 the section called “콘솔을 사용하여 작업 정의 생성” 섹션을 참조하세요.	2022년 10월 27일
태스크 정의에 대한 새로운 Amazon ECS 콘솔 환경이 업데이트됨	새로운 Amazon ECS 콘솔 환경은 이제 태스크 정의를 위한 JSON 편집기를 포함합니다. 자세한 내용은 the section called “콘솔을 사용하여 작업 정의 생성” 섹션을 참조하세요.	2022년 10월 27일
새로운 Amazon ECS 콘솔 환경이 업데이트됨	새로운 Amazon ECS 콘솔 경험은 추가적 서비스 및 태스크 파라미터로 업데이트되었습니다. 자세한 내용은 the section called “롤링 업데이트 배포 생성” 및 the section called “애플리케이션을 태스크로 실행”(을)를 참조하세요.	2022년 10월 7일

변경 사항	설명	날짜
태스크 메타데이터 엔드포인트 버전 4의 새로운 정보	이제 태스크 메타데이터 엔드포인트 버전 4에는 VPC ID 및 서비스 이름이 포함됩니다. 자세한 내용은 the section called “태스크 메타데이터 엔드포인트 버전 4” 섹션을 참조하세요.	2022년 10월 7일
새로운 태스크 정의 크기	이제 Fargate의 Amazon ECS에서 8개 vCPU와 16개 vCPU 태스크 크기를 지원합니다. 자세한 내용은 the section called “태스크 크기” 섹션을 참조하세요.	2022년 9월 16일
ECS CLI 페이지가 아카이빙됨	ECS CLI 문서가 보관되었습니다. 명령줄 도구 요구 사항에 AWS Copilot을 사용하는 것이 있습니다. 자세한 내용은 AWS Copilot 명령줄 인터페이스를 사용하여 Amazon ECS 리소스 생성 섹션을 참조하세요.	2022년 9월 15일
새로운 Fargate 할당량	Fargate는 태스크 수 기반 할당량에서 vCPU 기반 할당량으로 전환하고 있습니다. 자세한 내용은 Amazon ECS 엔드포인트 및 할당량을 참조하세요.	2022년 9월 8일
Amazon EC2 Auto Scaling의 웜 풀 지원	이제 Amazon EC2 Auto Scaling 웜 풀을 사용하여 애플리케이션을 더 빠르게 확장하고 비용을 절감할 수 있습니다. 자세한 내용은 Amazon ECS Auto Scaling 그룹에 대해 사전 초기화된 인스턴스 구성 섹션을 참조하세요.	2022년 3월 23일
ECS Anywhere의 Windows 지원.	이제 ECS Anywhere에서 Windows 인스턴스를 지원합니다. 자세한 내용은 외부 시작 유형을 위한 Amazon ECS 클러스터 섹션을 참조하세요.	2022년 3월 3일
외부 인스턴스에 대한 ECS Exec 지원 추가됨	이제 외부 인스턴스에 대해 ECS Exec이 지원됩니다. 자세한 내용은 ECS Exec를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.	2022년 1월 24일

변경 사항	설명	날짜
새로운 Amazon ECS 콘솔 환경 업데이트됨	새로운 Amazon ECS 콘솔 환경에서 클러스터 생성 및 삭제, 태스크 정의 업데이트, 태스크 정의 등록 취소를 지원합니다. 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 클러스터 생성, Amazon ECS 클러스터 삭제, 콘솔을 사용하여 Amazon ECS 작업 정의 업데이트 및 콘솔을 사용하여 Amazon ECS 작업 정의 개정 등록 취소 부분을 참조하세요.	2021년 12월 8일
새로운 Amazon ECS 콘솔 환경 업데이트됨	새로운 Amazon ECS 콘솔 환경에서 태스크 정의 생성을 지원합니다. 자세한 내용은 콘솔을 사용하여 Amazon ECS 작업 정의 생성 섹션을 참조하세요.	2021년 11월 23일
Amazon ECS에서 Linux 용 64비트 ARM 아키텍처 지원	Amazon ECS에서 Linux 운영 체제용 64비트 ARM CPU 아키텍처를 지원합니다. 자세한 내용은 the section called “64비트 ARM 워크로드에 대한 작업 정의” 섹션을 참조하세요.	2021년 11월 23일
fluentd log-driver-buffer-limit 옵션에 대한 Amazon ECS 지원	Amazon ECS에서 fluentd `log-driver-buffer-limit` 옵션을 지원합니다. 자세한 내용은 Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송 섹션을 참조하세요.	2021년 11월 22일
Amazon ECS 최적화 Linux AMI 구축 스크립트	Amazon ECS에서 Amazon ECS 최적화 AMI의 Linux 변형을 구축하는 데 사용되는 구축 스크립트를 오픈 소스로 제공합니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 구축 스크립트 섹션을 참조하세요.	2021년 11월 19일
컨테이너 인스턴스 상태	Amazon ECS에서 컨테이너 인스턴스 상태 모니터링에 대한 지원을 추가합니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 상태 모니터링 섹션을 참조하세요.	2021년 11월 10일
Windows Amazon ECS Exec 지원	Amazon ECS Exec에서 Windows를 지원합니다. 자세한 내용은 ECS Exec를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.	2021년 11월 1일

변경 사항	설명	날짜
Fargate에서 Windows 컨테이너 지원	Amazon ECS에서 Fargate의 Windows 컨테이너를 지원합니다. 자세한 내용은 Fargate Windows 플랫폼 버전 변경 로그 섹션을 참조하세요.	2021년 10월 28일
Amazon ECS Anywhere의 외부 인스턴스에 대한 GPU 지원	Amazon ECS에서 외부 인스턴스에서 실행되는 태스크에 대한 태스크 정의에 GPU 요구 사항 지정을 지원합니다. 자세한 내용은 GPU 워크로드에 대한 Amazon ECS 작업 정의 및 Amazon ECS 클러스터에 외부 인스턴스 등록(을)를 참조하세요.	2021년 10월 8일
Windows에서 awsvpc 네트워크 모드 지원	Amazon ECS가 Windows에서 awsvpc 네트워크 모드를 지원합니다. 자세한 내용은 Amazon ECS 작업에 대한 네트워크 인터페이스 할당 섹션을 참조하세요.	2021년 7월 15일
Bottlerocket 정식 출시	Amazon ECS에서 Bottlerocket 운영 체제의 Amazon ECS 최적화 AMI 버전을 AMI로 제공하도록 지원합니다. 자세한 내용은 Amazon ECS 최적화 Bottlerocket AMI 섹션을 참조하세요.	2021년 6월 30일
Amazon ECS 예약된 태스크 업데이트	Amazon EventBridge에서 Amazon ECS 예약 태스크를 트리거하는 규칙을 생성할 때 추가적 파라미터 지원을 추가했습니다.	2021년 6월 25일
Amazon ECS에 대한 AWS 관리형 정책	Amazon ECS에서 서비스 연결 역할에 대한 AWS 관리형 정책 문서를 추가했습니다. 자세한 내용은 Amazon Elastic Container Service용 AWS 관리형 정책 섹션을 참조하세요.	2021년 6월 8일
AWS CDK 시작하기	Amazon ECS로 AWS CDK를 사용하기 위한 시작 가이드를 추가했습니다. 자세한 내용은 AWS CDK를 사용하여 Amazon ECS 리소스 생성 섹션을 참조하세요.	2021년 5월 27일
Amazon ECS Anywhere	Amazon ECS에서 클러스터에 온프레미스 서버 또는 가상 머신(VM)을 등록하기 위한 지원을 추가했습니다. 자세한 내용은 외부 시작 유형을 위한 Amazon ECS 클러스터 섹션을 참조하세요.	2021년 5월 25일

변경 사항	설명	날짜
Amazon ECS 최적화 Windows Server 20H2 Core AMI	Amazon ECS에서 Windows Server 20H2 Core에 대해 새로운 Windows Amazon ECS 최적화 AMI 버전에 대한 지원을 추가했습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2021년 4월 19일
Amazon ECS Exec	Amazon ECS에서 ECS Exec라는 새로운 디버깅 도구를 릴리스했습니다. 자세한 내용은 ECS Exec를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.	2021년 3월 15일
VPC 엔드포인트 정책 지원	Amazon ECS에서 이제 VPC 엔드포인트 정책을 지원합니다. 자세한 내용은 Amazon ECS에 대한 VPC 엔드포인트 정책 생성 섹션을 참조하세요.	2021년 1월 11일
새로운 콘솔 환경	Amazon ECS에서 서비스를 생성하거나 업데이트하거나 독립 실행형 태스크를 실행하는 태스크를 지원하는 새로운 콘솔 환경을 릴리스했습니다. 자세한 내용은 Amazon ECS 롤링 업데이트 배포 생성 및 애플리케이션을 Amazon ECS 태스크로 실행(을)를 참조하세요.	2020년 12월 28일
용량 공급자 업데이트	Amazon ECS에서 기존 Auto Scaling 그룹 용량 공급자 업데이트 지원을 추가했습니다.	2020년 11월 23일
ECS에서 Windows 태스크에 대해 Amazon FSx for Windows File Server 지원	Amazon ECS는 Windows 태스크 정의에서 Amazon FSx for Windows File Server 볼륨을 지정하는 지원이 추가되었습니다. 자세한 내용은 Amazon ECS에서 FSx for Windows File Server 볼륨 사용 섹션을 참조하세요.	2020년 11월 11일
VPC 듀얼 스택 모드 지원 추가	Amazon ECS에서 IPv6 주소에 대한 지원을 제공하는 awsvpc 네트워크 모드를 사용하여 태스크에 듀얼 스택 모드 VPC를 사용하는 데 대한 지원이 추가되었습니다. 자세한 내용은 듀얼 스택 모드로 VPC 사용하기 섹션을 참조하세요.	2020년 11월 5일

변경 사항	설명	날짜
태스크 메타데이터 엔드포인트 v4 업데이트	Amazon ECS에서 태스크 메타데이터 엔드포인트 v4 출력에 추가 메타데이터를 추가했습니다. 자세한 내용은 Amazon ECS 작업 메타데이터 엔드포인트 버전 4 섹션을 참조하세요.	2020년 11월 5일
Local Zone 및 Wavelength Zone에 대한 지원	Amazon ECS에서 Local Zone와 Wavelength Zone의 워크로드에 대한 지원을 추가했습니다. 자세한 내용은 공유 서브넷, 로컬 영역 및 Wavelength Zone의 Amazon ECS 애플리케이션 섹션을 참조하세요.	2020년 9월 4일
Bottlerocket AMI의 Amazon ECS 버전	Bottlerocket은 AWS에서 컨테이너를 실행하기 위해 특별히 구축한 Linux 기반 오픈 소스 운영 체제입니다. Bottlerocket 운영 체제의 Amazon ECS 최적화 AMI 변형은 Amazon ECS 컨테이너 인스턴스를 시작할 때 사용할 수 있는 AMI로 제공됩니다. 자세한 내용은 Amazon ECS 최적화 Bottlerocket AMI 섹션을 참조하세요.	2020년 8월 31일
태스크 메타데이터 엔드포인트 버전 4가 네트워크 속도 통계에 대해 업데이트됨	태스크 메타데이터 엔드포인트 버전 4는 컨테이너 에이전트 버전 `1.43.0` 이상을 실행하는 Amazon EC2 인스턴스에서 호스팅되는 `awsvpc` 또는 `bridge` 네트워크 모드를 사용하는 Amazon ECS 태스크에 대한 네트워크 속도 통계를 제공하도록 업데이트되었습니다. 자세한 내용은 Amazon ECS 작업 메타데이터 엔드포인트 버전 4 섹션을 참조하세요.	2020년 8월 10일
Fargate 사용량 지표	AWS Fargate는 Fargate 온디맨드 및 Fargate Spot 리소스의 계정 사용에 대한 가시성을 제공하는 CloudWatch 사용량 지표를 제공합니다. 자세한 내용은 사용량 지표 섹션을 참조하세요.	2020년 8월 3일

변경 사항	설명	날짜
AWS Copilot 버전 0.1.0	새로운 AWS Copilot CLI가 출시되었으며, 로컬 개발 환경에서 Amazon ECS를 기반으로 컨테이너화된 애플리케이션의 모델링, 생성, 릴리스, 관리를 단순화하는 간단한 명령을 제공합니다. 자세한 내용은 AWS Copilot 명령줄 인터페이스를 사용하여 Amazon ECS 리소스 생성 섹션을 참조하세요.	2020년 7월 9일
AWS Fargate 플랫폼 버전 사용 중단 일정	Fargate 플랫폼 버전 사용 중단 일정이 추가되었습니다. 자세한 내용은 AWS Fargate Linux 플랫폼 버전 사용 중단 섹션을 참조하세요.	2020년 7월 8일
AWS Fargate 리전 확장	Amazon ECS의 AWS Fargate가 유럽(밀라노) 리전으로 확장되었습니다.	2020년 6월 25일
Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI 릴리스	Amazon ECS에서 추론 워크로드를 위한 Amazon ECS 최적화 Amazon Linux 2(Neuron) AMI를 릴리스했습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2020년 6월 24일
용량 공급자 삭제에 대한 지원이 추가되었습니다.	Amazon ECS에 Auto Scaling 그룹 용량 공급자 삭제에 대한 지원이 추가되었습니다.	2020년 6월 11일
AWS Fargate 플랫폼 버전 1.4.0 업데이트	2020년 5월 28일부터 플랫폼 버전 1.4.0을 사용하여 시작된 모든 새 Fargate 태스크는 AWS Fargate-관리형 암호화 키를 사용하여 AES-256 암호화 알고리즘으로 암호화된 20GB 임시 스토리지를 갖게 됩니다. 자세한 내용은 Amazon ECS에 대한 Fargate 작업 임시 스토리지 섹션을 참조하세요.	2020년 5월 28일
환경 변수 파일 지원	환경 변수를 컨테이너에 대량으로 추가할 수 있는 태스크 정의에서 환경 변수 파일을 지정하는 지원이 추가되었습니다. 자세한 내용은 개별 환경 변수를 Amazon ECS 컨테이너로 전달 섹션을 참조하세요.	2020년 5월 18일

변경 사항	설명	날짜
AWS Fargate 리전 확장	Amazon ECS의 AWS Fargate가 아프리카(케이프타운) 리전으로 확장되었습니다.	2020년 5월 11일
서비스 할당량이 업데이트 됨	다음 서비스 할당량이 업데이트되었습니다. • 계정당 클러스터가 2,000에서 10,000으로 증가되었습니다 . 자세한 내용은 Amazon ECS 엔드포인트 및 할당량을 참조하세요.	2020년 4월 17일

변경 사항	설명	날짜
AWS Fargate 플랫폼 버전 1.4.0	AWS Fargate 플랫폼 버전 1.4.0이 출시되었으며 다음과 같은 기능이 포함되어 있습니다. • 영구 태스크 스토리지를 위한 Amazon EFS 파일 시스템 볼륨 사용에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS에서 Amazon EFS 볼륨 사용 섹션을 참조하세요. • 임시 태스크 스토리지가 20GB까지 증가했습니다. 자세한 정보는 Amazon ECS에 대한 Fargate 작업 임시 스토리지 섹션을 참조하세요. • 태스크를 주고받은 네트워크 트래픽 동작이 업데이트되었습니다. 플랫폼 버전 1.4부터 모든 Fargate 태스크는 단일 탄력적 네트워크 인터페이스(작업 ENI라고 함)를 수신하고 모든 네트워크 트래픽은 VPC 내의 해당 ENI를 통해 흐르고 VPC 흐름 로그를 통해 사용자에게 표시됩니다. 자세한 내용은 Fargate 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션을 참조하세요. • 태스크 ENI는 점보 프레임에 대한 지원을 추가합니다. 네트워크 인터페이스는 단일 프레임 내에 맞는 가장 큰 페이로드 크기인 최대 전송 단위(MTU)로 구성됩니다. MTU가 클수록 단일 프레임 내에 더 많은 애플리케이션 페이로드가 들어갈 수 있으므로 프레임당 오버헤드가 줄어들고 효율성이 향상됩니다. 점보 프레임을 지원하면 VPC 내에 남아 있는 모든 트래픽과 같이 태스크와 대상 간의 네트워크 경로가 점보 프레임을 지원할 때 오버헤드가 줄어듭니다. • CloudWatch Container Insights가 Fargate 태스크에 대한 네트워크 성능 지표를 포함합니다. 자세한 정보는 관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링을 참조하세요.	2020년 4월 8일

변경 사항	설명	날짜
	• 태스크에 대한 네트워크 통계 및 태스크가 실행 중인 가용 영역 등 Fargate 태스크에 대한 추가 정보를 제공하는 태스크 메타데이터 엔드포인트 v4에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 작업 메타데이터 엔드포인트 버전 4 섹션을 참조하세요. • 컨테이너 정의에 SYS_PTRACE Linux 파라미터에 대한 지원이 추가되었습니다. 자세한 정보는 Linux 파라미터 섹션을 참조하세요. • Fargate 컨테이너 에이전트가 모든 Fargate 태스크에 대해 Amazon ECS 컨테이너 에이전트의 사용을 대체합니다. 이 변경 사항은 태스크 실행 방식에 영향을 주지 않습니다. • 컨테이너 런타임은 이제 Docker 대신 Containerd를 사용합니다. 이 변경 사항은 태스크 실행 방식에 영향을 주지 않습니다. 컨테이너 런타임에 시작되는 일부 오류 메시지는 Docker를 언급하는 것에서 보다 일반적인 오류로 변경됩니다. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.	
태스크 볼륨에 대한 Amazon EFS 파일 시스템 지원	Amazon EFS 파일 시스템을 Amazon ECS 및 Fargate 태스크 모두에 대한 데이터 볼륨으로 사용할 수 있습니다. 자세한 내용은 Amazon ECS에서 Amazon EFS 볼륨 사용 섹션을 참조하세요.	2020년 4월 8일

변경 사항	설명	날짜
Amazon ECS 태스크 메타데이터 엔드포인트 버전 4	Amazon ECS 컨테이너 에이전트 버전 1.39.0 및 Fargate 플랫폼 버전 1.4.0부터 `ECS_CONTAINER_METADATA_URI_V4` 라는 환경 변수가 태스크의 각 컨테이너에 주입됩니다. 태스크 메타데이터 버전 4 엔드포인트를 쿼리하면 다양한 태스크 메타데이터와 Docker 통계를 태스크에 사용할 수 있습니다. 자세한 내용은 Amazon ECS 작업 메타데이터 엔드포인트 버전 4 섹션을 참조하세요.	2020년 4월 8일
특정 버전의 Secrets Manager 암호를 환경 변수로 주입할 수 있도록 지원	특정 버전의 Secrets Manager 암호를 사용하여 중요한 데이터를 지정하는 것에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.	2020년 2월 24일
블루/그린 배포를 위해 추가 CodeDeploy 배포 구성 옵션이 추가되었습니다.	CodeDeploy 서비스는 Amazon ECS 배포 유형에 대한 새로운 Canary 및 Linear 배포 구성을 추가했습니다. 사용자 지정 배포 구성을 정의하는 기능도 사용할 수 있습니다. 자세한 내용은 Amazon ECS에 대한 CodeDeploy 블루/그린 배포 섹션을 참조하세요.	2020년 2월 6일
efsVolumeConfiguration 태스크 정의 파라미터를 추가함	`efsVolumeConfiguration` 태스크 정의 파라미터는 공개 미리 보기에 있으므로 Amazon ECS 태스크에 Amazon EFS 파일 시스템을 더 쉽게 사용할 수 있습니다. 자세한 내용은 Amazon ECS에서 Amazon EFS 볼륨 사용 섹션을 참조하세요.	2020년 1월 17일
Amazon ECS 컨테이너 에이전트 로깅 동작이 업데이트됨	Amazon ECS 컨테이너 에이전트 로깅 위치 및 교체 동작이 업데이트되었습니다. 자세한 내용은 Amazon ECS 컨테이너 에이전트 로그 구성 파라미터 섹션을 참조하세요.	2020년 1월 13일
Fargate 스팟	Amazon ECS에 Fargate 스팟을 사용한 태스크 실행에 대한 지원이 추가되었습니다. 자세한 내용은 Fargate용 Amazon ECS 클러스터 섹션을 참조하세요.	2019년 12월 3일

변경 사항	설명	날짜
클러스터 Auto Scaling	Amazon ECS 클러스터 Auto Scaling을 사용하면 클러스터 내에서 태스크를 확장하는 방법을 더 세부적으로 제어할 수 있습니다. 자세한 내용은 클러스터 Auto Scaling을 통해 Amazon ECS 용량 자동 관리 섹션을 참조하세요.	2019년 12월 3일
클러스터 용량 공급자	Amazon ECS 클러스터 용량 공급자는 태스크에 사용할 인프라를 결정합니다. 자세한 내용은 Amazon ECS 클러스터 섹션을 참조하세요.	2019년 12월 3일
AWS Outposts에서 클러스터 생성	이제 Amazon ECS가 AWS Outposts에 클러스터를 생성하도록 지원합니다. 자세한 내용은 the section called “AWS Outposts의 Amazon Elastic Container Service” 섹션을 참조하세요.	2019년 12월 3일
서비스 작업 이벤트	Amazon ECS는 이제 특정 서비스 동작이 발생할 때 Amazon EventBridge로 이벤트를 보냅니다. 자세한 내용은 Amazon ECS 서비스 작업 이벤트 섹션을 참조하세요.	2019년 11월 25일
Amazon ECS GPU 최적화 AMI가 G4 인스턴스를 지원합니다.	Amazon ECS에서는 Amazon ECS GPU 최적화 AMI 사용 시 g4 인스턴스 유형 패밀리 지원을 추가했습니다. 자세한 내용은 GPU 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요.	2019년 10월 8일
Amazon ECS용 FireLens	Amazon ECS용 FireLens는 정식으로 출시되었습니다. Amazon ECS용 FireLens를 사용하면 태스크 정의 매개변수를 사용하여 로그를 AWS 서비스 또는 파트너 대상으로 라우팅하여 로그를 저장 및 분석할 수 있습니다. 자세한 내용은 Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송 섹션을 참조하세요.	2019년 9월 30일
AWS Fargate 리전 확장	AWS Fargate는 Amazon ECS와 함께 EU(파리), EU(스톡홀름) 및 중동(바레인) 리전으로 확장되었습니다.	2019년 9월 30일

변경 사항	설명	날짜
Amazon ECS에 Elastic Inference가 탑재된 Deep Learning Containers	Amazon ECS는 컨테이너에 Amazon Elastic Inference 액셀러레이터를 연결하여 딥 러닝 추론 워크로드를 보다 효율적으로 실행할 수 있도록 지원합니다.	2019년 9월 3일
Amazon ECS용 FireLens	Amazon ECS용 FireLens는 공개 미리보기입니다. Amazon ECS용 FireLens를 사용하면 태스크 정의 매개변수를 사용하여 로그를 AWS 서비스 또는 파트너 대상으로 라우팅하여 로그를 저장 및 분석할 수 있습니다. 자세한 내용은 Amazon ECS 로그를 AWS 서비스 또는 AWS Partner로 전송 섹션을 참조하세요.	2019년 8월 30일
CloudWatch Container Insights	CloudWatch 이제 컨테이너 인사이트를 사용할 수 있습니다. 컨테이너 인사이트는 컨테이너 애플리케이션 및 마이크로서비스의 지표 및 로그를 수집하고 종합하며 요약합니다. 자세한 내용은 관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.	2019년 8월 30일
컨테이너 수준 스왑 구성	Amazon ECS는 컨테이너 수준에서 Linux 컨테이너 인스턴스의 스왑 메모리 공간 사용을 제어하는 것에 대한 지원을 추가했습니다. 컨테이너 별 스왑 구성을 사용하면 태스크 정의 내의 각 컨테이너에서 스왑을 활성화 또는 비활성화할 수 있으며 이를 활성화한 컨테이너의 경우 사용되는 최대 스왑 공간을 제한할 수 있습니다. 자세한 내용은 Amazon ECS에서 컨테이너 스왑 메모리 공간 관리 섹션을 참조하세요.	2019년 8월 16일
AWS Fargate 리전 확장	이제 아시아 태평양(홍콩) 리전에서 Amazon ECS의 AWS Fargate를 사용할 수 있습니다.	2019년 8월 6일
탄력적 네트워크 인터페이스 트렁킹	ENI 트렁킹 기능을 위해 지원되는 Amazon EC2 인스턴스 유형이 추가되었습니다. 자세한 내용은 증가한 Amazon ECS 컨테이너 네트워크 인터페이스에 대해 지원되는 인스턴스 섹션을 참조하세요.	2019년 8월 1일

변경 사항	설명	날짜
서비스에 여러 대상 그룹 등록	서비스 정의에 특정 여러 대상 그룹 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 서비스에 여러 대상 그룹 등록 섹션을 참조하세요.	2019년 7월 30일
Secrets Manager를 사용해 민감한 데이터 지정	Secrets Manager 암호를 사용하는 민감한 데이터를 지정하는 자습서를 추가했습니다. 자세한 내용은 Amazon ECS에서 Secrets Manager 암호를 사용하여 민감한 데이터 지정 섹션을 참조하세요.	2019년 7월 20일
CloudWatch Container Insights	Amazon ECS에 CloudWatch 컨테이너 인사이트에 대한 지원이 추가되었습니다. 자세한 내용은 관찰성이 향상된 Container Insights를 사용하여 Amazon ECS 컨테이너 모니터링 섹션을 참조하세요.	2019년 7월 9일
Amazon ECS 서비스 및 태스크 세트에 대한 리소스 수준 권한	Amazon ECS는 Amazon ECS 서비스 및 태스크에 대한 리소스 수준 권한 지원을 확장했습니다. 자세한 내용은 Amazon Elastic Container Service가 IAM과 작동하는 방식 섹션을 참조하세요.	2019년 6월 27일
AWS-2019-005에 패치된 새 Amazon ECS 최적화 AMI	Amazon ECS가 AWS-2019-005에서 설명하는 취약성을 해결하기 위해 Amazon ECS 최적화 AMI를 업데이트했습니다.	2019년 6월 17일
탄력적 네트워크 인터페이스 트렁킹	Amazon ECS에서는 탄력적 네트워크 인터페이스(ENI) 밀도가 늘어난 지원되는 Amazon EC2 인스턴스 유형을 사용하여 컨테이너 인스턴스를 시작하는 기능이 도입됩니다. 이러한 인스턴스 유형을 사용하고 `awsvpcTrunking` 계정 설정에 옵트인하면 새로 시작된 컨테이너 인스턴스에 대해 늘어난 ENI 밀도가 제공되므로 각 컨테이너 인스턴스에 추가 태스크를 배치할 수 있습니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 네트워크 인터페이스 증가 섹션을 참조하세요.	2019년 6월 6일

변경 사항	설명	날짜
AWS Fargate 플랫폼 버전 1.3.0 업데이트	2019년 5월 1일부터 시작되는 새로운 Fargate 태스크는 `awslogs` 로그 드라이버 외에 `splunk` 로그 드라이버도 지원합니다. 자세한 내용은 스토리지 및 로깅 섹션을 참조하세요.	2019년 5월 1일
AWS Fargate 플랫폼 버전 1.3.0 업데이트	2019년 5월 1일부터 시작되는 새로운 Fargate 태스크는 `secretOptions` 컨테이너 정의 파라미터를 사용하여 컨테이너의 로그 구성에서 민감한 데이터를 참조하는 것을 지원합니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.	2019년 5월 1일
AWS Fargate 플랫폼 버전 1.3.0 업데이트	2019년 4월 2일부터 새로운 Fargate 태스크는 AWS Secrets Manager 암호 또는 AWS Systems Manager 파라미터 스토어 파라미터에 중요한 데이터를 저장한 다음 컨테이너 정의에서 참조하여 중요한 데이터를 컨테이너에 주입할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.	2019년 4월 2일
AWS Fargate 플랫폼 버전 1.3.0 업데이트	2019년 3월 27일부터 새로운 Fargate 태스크는 프록시 구성, 컨테이너 시작 및 종료는 물론, 컨테이너별 시작 및 중지 제한 시간 값에 대한 종속성을 정의하는 데 사용할 수 있는 태스크 정의 파라미터를 추가로 출시했습니다. 자세한 내용은 프록시 구성, 컨테이너 종속성, 컨테이너 제한 시간 섹션을 참조하세요.	2019년 3월 27일
Amazon ECS에서 외부 배포 유형 소개	외부 배포 유형을 사용하면 타사 배포 컨트롤러를 사용하여 Amazon ECS 서비스의 배포 프로세스를 완벽하게 제어할 수 있습니다. 자세한 내용은 타사 컨트롤러를 사용한 Amazon ECS 서비스 배포 섹션을 참조하세요.	2019년 3월 27일

변경 사항	설명	날짜
Amazon ECS의 AWS Deep Learning Containers	AWS Deep Learning Containers는 Amazon Elastic Container Service(Amazon ECS)의 TensorFlow에서 모델을 교육하고 제공하기 위한 Docker 이미지 세트입니다. Deep Learning Containers는 TensorFlow, Nvidia CUDA(GPU 인스턴스용) 및 Intel MKL(CPU 인스턴스용) 라이브러리에 최적의 환경을 제공하며, Amazon ECR에서 사용할 수 있습니다. 자세한 내용은 Amazon ECS에서 AWS Deep Learning Containers 사용 섹션을 참조하세요.	2019년 3월 27일
Amazon ECS는 향상된 컨테이너 종속성 관리를 소개합니다.	Amazon ECS는 컨테이너 시작 및 종료는 물론, 컨테이너별 시작 및 중지 제한 시간 값에 대한 종속성을 정의하는 데 사용할 수 있는 태스크 정의 파라미터를 추가로 출시했습니다. 자세한 내용은 컨테이너 종속성 섹션을 참조하세요.	2019년 3월 7일
Amazon ECS에서 `PutAccountSettingDefault` API 도입	Amazon ECS는 계정 내 모든 사용자와 역할에 대해 기본 ARN/ID 형식 옵트인 상태를 설정할 수 있게 해주는 `PutAccountSettingDefault` API를 제공합니다. 전에는 계정의 기본 옵트인 상태를 설정하려면 계정 소유자를 사용해야 했습니다. 자세한 내용은 Amazon 리소스 이름(ARN) 및 ID 섹션을 참조하세요.	2019년 2월 8일
Amazon ECS에서 GPU 워크로드 지원	Amazon ECS는 GPU 지원 컨테이너 인스턴스로 클러스터를 생성하게 해주어 GPU 워크로드 지원을 제공할 수 있게 해줍니다. 태스크 정의 시 필요한 GPU 수를 지정할 수 있으며, ECS 에이전트가 실제 GPU를 컨테이너로 고정합니다. 자세한 내용은 GPU 워크로드에 대한 Amazon ECS 작업 정의 섹션을 참조하세요.	2019년 2월 4일

변경 사항	설명	날짜
Amazon ECS에서 암호 지원 확대	Amazon ECS에서는 태스크 정의에서 직접 AWS Secrets Manager 암호를 사용하여 중요한 데이터를 컨테이너에 주입하기 위한 지원이 확장되었습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.	2019년 1월 21일
인터페이스 VPC 엔드포인트(AWS PrivateLink)	AWS PrivateLink에서 제공하는 인터페이스 VPC 엔드포인트 구성을 위한 지원이 추가되었습니다. 따라서 NAT 인스턴스, VPN 연결 또는 AWS Direct Connect를 통해 인터넷에 액세스하지 않고도 VPC와 Amazon ECS 간에 프라이빗 연결을 생성할 수 있습니다. 자세한 내용은 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.	2018년 12월 26일
AWS Fargate 플랫폼 버전 1.3.0	새로 출시된 AWS Fargate 플랫폼 버전으로 다음을 포함하고 있습니다. • AWS Systems Manager 파라미터 스토어 파라미터를 사용하여 중요한 데이터를 컨테이너에 주입하기 위한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요. • Fargate 태스크의 태스크 재생이 추가되었습니다. 이 기능은 Amazon ECS 서비스에 속하는 태스크를 새로 고치는 프로세스입니다. 자세한 내용은 Amazon ECS의 AWS Fargate에 대한 태스크 사용 중지 및 유지 관리를 참조하세요. 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.	2018년 12월 17일

변경 사항	설명	날짜
서비스 제한 업데이트됨	업데이트된 서비스 제한은 다음과 같습니다. • 계정 하나의 리전당 클러스터 수가 1000에서 2000으로 증가했습니다. • 클러스터당 컨테이너 인스턴스 수가 1000에서 2000으로 증가했습니다. • 클러스터당 서비스 수가 500에서 1000으로 증가했습니다. 자세한 내용은 Amazon ECS 서비스 할당량 섹션을 참조하세요.	2018년 12월 14일
AWS Fargate 리전 확장	Amazon ECS의 AWSFargate가 아시아 태평양(뭄바이) 및 캐나다(중부) 리전으로 확장되었습니다. 자세한 내용은 AWS Fargate의 Amazon ECS에 대해 지원되는 리전 섹션을 참조하세요.	2018년 12월 7일
Amazon ECS 블루/그린 배포	Amazon ECS는 CodeDeploy를 사용하여 블루/그린 배포 지원을 추가했습니다. 이 배포 유형을 사용하면 프로덕션 트래픽을 전송하기 전에 서비스의 새로운 배포를 확인할 수 있습니다. 자세한 내용은 Amazon ECS에 대한 CodeDeploy 블루/그린 배포 섹션을 참조하세요.	2018년 11월 27일
Amazon ECS 최적화 Amazon Linux 2(arm64) AMI 릴리스	Amazon ECS에서 arm64 아키텍처에 대해 Amazon ECS 최적화 Amazon Linux 2 AMI를 릴리스했습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2018년 11월 26일

변경 사항	설명	날짜
태스크 정의에서 추가 Docker 플래그 지원이 추가되었습니다.	Amazon ECS는 태스크 정의에서 다음 Docker 플래그에 대한 지원을 도입하였습니다. • IPC 모드 • PID 모드	2018년 11월 16일
Amazon ECS 암호 지원	Amazon ECS는 AWS Systems Manager 파라미터 스토어 파라미터를 사용하여 중요한 데이터를 컨테이너에 주입하기 위한 지원을 추가했습니다. 자세한 내용은 Amazon ECS 컨테이너로 민감한 데이터 전달 섹션을 참조하세요.	2018년 11월 15일
리소스에 태깅	Amazon ECS에서는 서비스, 태스크 정의, 태스크, 클러스터 및 컨테이너 인스턴스에 메타데이터 태그를 추가하기 위한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 리소스 태그 지정 섹션을 참조하세요.	2018년 11월 15일
AWS Fargate 리전 확장	Amazon ECS의 AWS Fargate는 미국 서부(캘리포니아 북부) 및 아시아 태평양(서울) 리전으로 확장되었습니다. 자세한 내용은 Amazon ECS용 AWS Fargate 섹션을 참조하세요.	2018년 11월 7일

변경 사항	설명	날짜
서비스 제한 업데이트됨	업데이트된 서비스 제한은 다음과 같습니다. • 리전별로 20 Fargate 시작 유형을 사용하는 태스크 개수를 에서 50으로 늘렸습니다. • 20 Fargate 시작 유형을 사용하는 태스크에 대한 퍼블릭 IP 주소의 개수를 에서 50으로 늘렸습니다. 자세한 내용은 Amazon ECS 서비스 할당량 섹션을 참조하세요.	2018년 10월 31일
AWS Fargate 리전 확장	Amazon ECS의 AWSFargate는 유럽(런던) 리전으로 확장되었습니다. 자세한 내용은 Amazon ECS용 AWS Fargate 섹션을 참조하세요.	2018년 10월 26일
Amazon ECS 최적화 Amazon Linux 2 AMI 릴리스	Amazon ECS는 이 서비스에 최적화된 Linux AMI를 두 가지 형태로 판매합니다. 최신 권장 버전은 x를 기준으로 합니다. Amazon ECS는 AMI도 제공하지만, Amazon Linux AMI에 대한 지원이 2020년 6월 30일에 종료되었으므로 Amazon Linux 2 변형으로 워크로드를 마이그레이션하는 것이 좋습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2018년 10월 18일

변경 사항	설명	날짜
Amazon ECS 태스크 메타데이터 엔드포인트 버전 3	Amazon ECS 컨테이너 에이전트의 버전 1.21.0부터 에이전트는 `ECS_CONTAINER_METADATA_URI` 라는 이름의 환경 변수를 태스크의 각 컨테이너에 주입합니다. 태스크 메타데이터 버전 3 엔드포인트를 쿼리할 때 Amazon ECS 컨테이너 에이전트에서 제공하는 HTTP 엔드포인트에서 `awsvpc` 네트워크 모드를 사용하는 태스크에 다양한 태스크 메타데이터 및 Docker 통계가 제공됩니다. 자세한 내용은 Amazon ECS 메타데이터를 사용한 워크로드 모니터링 섹션을 참조하세요.	2018년 10월 18일
Amazon ECS 서비스 검색 리전 확장	Amazon ECS 서비스 검색 지원이 캐나다(중부), 남아메리카(상파울루), 아시아 태평양(서울), 아시아 태평양(뭄바이), 유럽(파리) 리전까지 확장되었습니다. 자세한 내용은 서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결 섹션을 참조하세요.	2018년 9월 27일
컨테이너 정의에서 추가 Docker 플래그 지원이 추가되었습니다.	Amazon ECS는 컨테이너 정의에서 다음 Docker 플래그에 대한 지원을 도입하였습니다. • System Controls • 대화형 • 의사 터미널	2018년 9월 17일
AWS Fargate 태스크를 사용하는 Amazon ECS의 프라이빗 레지스트리 인증 지원	Amazon ECS는 AWS Secrets Manager을 사용하는 프라이빗 레지스트리 인증 지원을 사용하는 Fargate 태스크에 대한 지원을 도입하였습니다. 이 기능의 추가로 이제는 자격 증명을 안전하게 저장한 후 컨테이너 정의에서 참조할 뿐만 아니라 태스크에서 프라이빗 이미지를 사용하는 것도 가능합니다. 자세한 내용은 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용 섹션을 참조하세요.	2018년 9월 10일

변경 사항	설명	날짜
Amazon ECS 서비스 검색 리전 확장	Amazon ECS 서비스 검색 지원이 아시아 태평양(싱가포르), 아시아 태평양(시드니), 아시아 태평양(도쿄), 유럽(프랑크푸르트), 유럽(런던) 리전까지 확장되었습니다. 자세한 내용은 서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결 섹션을 참조하세요.	2018년 8월 30일
Fargate 태스크가 지원되는 예약 태스크	Amazon ECS는 Fargate 시작 유형의 예약 태스크 지원을 도입하였습니다.	2018년 8월 28일
AWS Secrets Manager을 사용한 프라이빗 레지스트리 인증 지원 기능	Amazon ECS는 AWS Secrets Manager을 사용하는 프라이빗 레지스트리 인증 지원 기능을 도입하였습니다. 이 기능의 추가로 이제는 자격 증명을 안전하게 저장한 후 컨테이너 정의에서 참조할 뿐만 아니라 태스크에서 프라이빗 이미지를 사용하는 것도 가능합니다. 자세한 내용은 Amazon ECS에서 AWS 컨테이너가 아닌 이미지 사용 섹션을 참조하세요.	2018년 8월 16일
Docker 볼륨 지원 추가됨	Amazon ECS는 Docker 볼륨에 대한 지원 기능을 도입하였습니다. 자세한 내용은 Amazon ECS 작업에 대한 스토리지 옵션 섹션을 참조하세요.	2018년 8월 9일
AWS Fargate 리전 확장	Amazon ECS의 AWS Fargate가 유럽(프랑크푸르트), 아시아 태평양(싱가포르) 및 아시아 태평양(시드니) 리전으로 확장되었습니다. 자세한 내용은 Amazon ECS용 AWS Fargate 섹션을 참조하세요.	2018년 7월 19일

변경 사항	설명	날짜
Amazon ECS 서비스 스케줄러 전략이 추가됨	Amazon ECS가 서비스 스케줄러 전략 개념을 도입했습니다. 사용 가능한 서비스 스케줄러 전략으로 다음 두 가지가 있습니다. • REPLICA—복제본 일정 전략은 클러스터에 원하는 작업 수를 배치하고 유지합니다. 기본적으로 서비스 스케줄러는 가용 영역에 태스크를 분산합니다. 작업 배치 전략과 제약을 사용하여 작업 배치 결정을 사용자 지정할 수 있습니다. 자세한 내용은 복제본 예약 전략 섹션을 참조하세요. • DAEMON—데몬 일정 전략은 사용자가 클러스터에 지정하는 작업 배치 제약을 모두 충족하는 각 활성 컨테이너 인스턴스에 한 작업씩 정확히 배포합니다. 이 전략을 사용하는 경우 원하는 태스크 수, 작업 배치 전략을 지정하거나 서비스 Auto Scaling 정책을 사용할 필요가 없습니다. 자세한 내용은 대몬 예약 전략 섹션을 참조하세요. **Note** Fargate 태스크는 DAEMON 일정 전략을 지원하지 않습니다. .	2018년 6월 12일

변경 사항	설명	날짜
Amazon ECS 컨테이너 에이전트 v1.18.0	다음 기능이 추가된 Amazon ECS 컨테이너 에이전트의 새 버전이 출시되었습니다. • `ECS_IMAGE_PULL_BEHAVIOR` 파라미터를 사용한 컨테이너 에이전트 이미지 가져오기 동작 사용자 지정에 대한 지원을 추가했습니다. 자세한 정보는 Amazon ECS 컨테이너 에이전트 구성을 참조하세요. 자세한 내용은 amazon-ecs-agent github를 참조하세요.	2018년 5월 24일
서비스 검색 구성 때 `bridge` 및 `host` 네트워크 모드 지원 추가	`bridge` 또는 `host` 네트워크 모드를 지정하는 태스크 정의를 사용하여 Amazon ECS 서비스에 대해 서비스 검색을 구성할 수 있도록 지원을 추가했습니다. 자세한 내용은 서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결 섹션을 참조하세요.	2018년 5월 22일
추가 Amazon ECS 최적화 AMI 메타데이터 파라미터에 대한 지원을 추가했습니다.	Amazon ECS 최적화 AMI ID, 이미지 이름, 운영 체제, 컨테이너 에이전트 버전 및 실행 시간 버전을 프로그래밍 방식으로 검색할 수 있는 하위 파라미터를 추가했습니다. Systems Manager 파라미터 스토어 API를 사용하여 메타데이터를 쿼리합니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 메타데이터 검색 섹션을 참조하세요.	2018년 5월 9일
AWS Fargate 리전 확장	Amazon ECS의 AWS Fargate가 미국 동부(오하이오), 미국 서부(오레곤) 및 EU 서부(아일랜드) 리전으로 확장되었습니다. 자세한 내용은 Amazon ECS용 AWS Fargate 섹션을 참조하세요.	2018년 4월 26일

변경 사항	설명	날짜
Amazon ECS 최적화 AMI 메타데이터 검색	Systems Manager 파라미터 스토어 API를 사용하여 Amazon ECS 최적화 AMI 메타데이터를 프로그래밍 방식으로 가져오는 기능이 추가되었습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 메타데이터 검색 섹션을 참조하세요.	2018년 10월 4일
AWS Fargate 플랫폼 버전	새로 출시된 AWS Fargate 플랫폼 버전으로 다음을 포함하고 있습니다. • Amazon ECS 메타데이터를 사용한 워크로드 모니터링에 대한 지원이 추가되었습니다. • 상태 확인 지원이 추가되었습니다. • 서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결에 대한 지원 추가 자세한 내용은 Amazon ECS에 대한 Fargate 플랫폼 버전 섹션을 참조하세요.	2018년 3월 26일
Amazon ECS 서비스 검색	Amazon ECS 서비스 검색을 지원하는 Route 53과의 통합 기능이 추가되었습니다. 자세한 내용은 서비스 검색을 사용하여 Amazon ECS 서비스를 DNS 이름으로 연결 섹션을 참조하세요.	2018년 3월 22일
Docker shm-size 및 tmpfs 지원	Amazon ECS 태스크 정의에서 Docker shm-size 및 tmpfs 파라미터에 대한 지원이 추가되었습니다. 업데이트된 ECS CLI 구문에 대한 자세한 내용은 Linux 파라미터 섹션을 참조하세요.	2018년 3월 20일
컨테이너 상태 확인	컨테이너 정의에서 Docker 상태 확인에 대한 지원이 추가되었습니다. 자세한 내용은 상태 확인 섹션을 참조하세요.	2018년 3월 8일

변경 사항	설명	날짜
AWS Fargate	AWS Fargate를 사용하여 Amazon ECS에 대한 개요를 추가했습니다. 자세한 내용은 Amazon ECS용 AWS Fargate 섹션을 참조하세요.	2018년 2월 22일
Amazon ECS 태스크 메타데이터 엔드포인트	Amazon ECS 컨테이너 에이전트 버전 1.17.0부터 다양한 태스크 메타데이터와 Docker 통계가 Amazon ECS 컨테이너 에이전트에서 제공하는 HTTP 엔드포인트에서 awsvpc 네트워크 모드를 사용하는 태스크에 제공됩니다. 자세한 내용은 Amazon ECS 메타데이터를 사용한 워크로드 모니터링 섹션을 참조하세요.	2018년 2월 8일
대상 추적 정책을 사용하는 Amazon ECS 서비스 Auto Scaling	Amazon ECS 콘솔에서 대상 추적 정책을 사용하는 ECS 서비스 Auto Scaling에 대한 지원을 추가했습니다. 자세한 내용은 대상 지표를 사용하여 Amazon ECS 서비스 규모 조정 섹션을 참조하세요. ECS에서 처음 실행된 마법사에서의 확장/축소 단계에 대한 이전 자습서가 제거되었습니다. 대신에 대상 추적을 위한 새 자습서로 바뀌었습니다.	2018년 2월 8일
Docker 17.09 지원	Docker 17.09 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2018년 1월 18일
새로운 서비스 스케줄러 동작	시작에 실패한 서비스 태스크의 동작에 대한 정보가 업데이트되었습니다. 서비스 태스크가 연속으로 실패하는 경우에 트리거되는 새로운 서비스 이벤트 메시지가 기록되었습니다.	2018년 1월 11일
Elastic Load Balancing 상태 확인 초기화 대기 시간	상태 확인의 대기 기간을 지정하는 기능이 추가되었습니다.	2017년 12월 27일
태스크 수준 CPU와 메모리	태스크 정의에서 태스크 수준에 CPU와 메모리를 지정하기 위한 지원이 추가되었습니다. 자세한 내용은 TaskDefinition을 참조하세요.	2017년 12월 12일

변경 사항	설명	날짜
태스크 실행 역할	Amazon ECS 컨테이너 에이전트는 사용자를 대신하여 Amazon ECS API 태스크를 호출하므로 에이전트가 사용자에게 속한다는 것을 서비스가 알기 위해서는 IAM 정책과 역할이 필요합니다. 다음 태스크는 태스크 실행 역할이 담당합니다. • 컨테이너 이미지를 가져오기 위한 Amazon ECR 호출 • 컨테이너 애플리케이션 로그를 저장하기 위한 CloudWatch 호출 자세한 내용은 Amazon ECS 태스크 실행 IAM 역할 섹션을 참조하세요.	2017년 12월 7일
Windows 컨테이너 지원 GA	Windows Server 2016 컨테이너에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 최적화 AMI 변형 섹션을 참조하세요.	2017년 12월 5일
AWS Fargate GA	Fargate 시작 유형을 사용하는 Amazon ECS 서비스 시작에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 시작 유형 섹션을 참조하세요.	2017년 11월 29일
Amazon ECS 이름 변경	Amazon Elastic Container Service 이름이 변경되었습니다(이전: Amazon EC2 Container Service).	2017년 11월 21일

변경 사항	설명	날짜
태스크 네트워킹	awsvpc 네트워크 모드에서 제공하는 태스크 네트워킹 기능은 Amazon ECS 태스크에 Amazon EC2 인스턴스와 동일한 네트워킹 속성을 제공합니다. 태스크 정의에 awsvpc 네트워크 모드를 사용하는 경우 태스크 정의에서 시작되는 모든 태스크는 고유한 탄력적 네트워크 인터페이스, 기본 프라이빗 IP 주소 및 내부 DNS 호스트 이름을 받습니다. 태스크 네트워킹 기능은 컨테이너 네트워킹을 간소화하고 컨테이너화된 애플리케이션이 서로, 그리고 VPC 내 다른 서비스와 통신하는 방식을 더 세부적으로 제어할 수 있습니다. 자세한 내용은 EC2 시작 유형에 대한 Amazon ECS 작업 네트워킹 옵션 섹션을 참조하세요.	2017년 11월 14일
Amazon ECS 컨테이너 메타데이터	Amazon ECS 컨테이너가 이제 Docker 컨테이너 또는 이미지 ID, 네트워킹 구성, Amazon ARN 등과 같은 메타데이터를 액세스할 수 있습니다. 자세한 내용은 Amazon ECS 컨테이너 메타데이터 파일 섹션을 참조하세요.	2017년 11월 2일
Docker 17.06 지원	Docker 17.06 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2017년 11월 2일
Docker 플래그: device 및 init에 대한 지원	`LinuxParameters` 파라미터(`devices` 및 `initProcessEnabled`)를 사용하여 태스크 정의에서 Docker의 device 및 init 기능에 대한 지원이 추가되었습니다. 자세한 내용은 LinuxParameters를 참조하세요.	2017년 11월 2일
Docker 플래그: cap-add 및 cap-drop에 대한 지원	`LinuxParameters` 파라미터(`capabilities`)를 사용하여 태스크 정의에서 Docker의 cap-add 및 cap-drop 기능에 대한 지원이 추가되었습니다. 자세한 내용은 LinuxParameters를 참조하세요.	2017년 9월 22일
Network Load Balancer 지원	Amazon ECS에서 Amazon ECS 콘솔에 Network Load Balancer에 대한 지원을 추가했습니다.	2017년 9월 7일

변경 사항	설명	날짜
태스크 실행 재정의	태스크 실행 시 태스크 정의 재정의에 대한 지원이 추가되었습니다. 이제, 태스크 정의 개정을 새로 만들 필요 없이 태스크 정의를 변경하는 동시에 태스크를 실행할 수 있습니다. 자세한 내용은 애플리케이션을 Amazon ECS 태스크로 실행 섹션을 참조하세요.	2017년 27월 6일
Amazon ECS 예약된 태스크	cron을 사용한 태스크 예약에 대한 지원이 추가되었습니다.	2017년 6월 7일
Amazon ECS 콘솔의 스팟 인스턴스	Amazon ECS 콘솔에서 스팟 플릿 컨테이너 인스턴스 생성을 위한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS Linux 컨테이너 인스턴스 시작 섹션을 참조하세요.	2017년 6월 6일
새로운 Amazon ECS 최적화 AMI 릴리스에 대한 Amazon SNS 알림	새로운 Amazon ECS 최적화 AMI 릴리스에 대한 SNS 알림 구독 기능이 추가되었습니다.	2017년 3월 23일
마이크로서비스 및 배치 작업	Amazon ECS에서의 두 가지 일반 사용 사례인 마이크로서비스와 배치(batch) 작업에 대한 설명서가 추가되었습니다.	2017년 2월
컨테이너 인스턴스 드레이닝	클러스터에서 컨테이너 인스턴스를 제거하기 위한 방법을 제공하는 컨테이너 인스턴스 드레이닝에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 컨테이너 인스턴스 드레이닝 섹션을 참조하세요.	2017년 1월 24일
Docker 1.12 지원	Docker 1.12 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2017년 1월 24일
새로운 작업 배치 전략	작업 배치 전략인 속성 기반 배치, bin 팩, 가용 영역 및 호스트당 한 개에 대한 지원이 추가되었습니다. 자세한 내용은 전략을 사용하여 Amazon ECS 작업 배치 정의 섹션을 참조하세요.	2016년 12월 29일

변경 사항	설명	날짜
베타에서 Windows 컨테이너 지원	Windows Server 2016 컨테이너에 대한 지원이 추가되었습니다(베타). 자세한 내용은 Amazon ECS 최적화 AMI 변형 섹션을 참조하세요.	2016년 12월 20일
Blox OSS 지원	사용자 지정 태스크 스케줄러를 허용하는 Blox OSS에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS에서 컨테이너 예약 섹션을 참조하세요.	2016년 1월 12일
CloudWatch Events에 대한 Amazon ECS 이벤트 스트림	이제 Amazon ECS에서 컨테이너 인스턴스와 태스크 상태 변경을 CloudWatch Events로 전송합니다. 자세한 내용은 EventBridge를 사용하여 Amazon ECS 오류에 대한 응답 자동화 섹션을 참조하세요.	2016년 11월 21일
CloudWatch Logs에 대한 Amazon ECS 컨테이너 로깅	컨테이너 로그 스트림을 CloudWatch Logs로 전송하기 위해 awslogs 드라이버에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 로그를 CloudWatch로 전송 섹션을 참조하세요.	2016년 9월 12일
동적 포트에 Elastic Load Balancing을 지원하는 Amazon ECS 서비스	리스너당 여러 인스턴스:포트 조합을 지원하여 컨테이너에 대한 유연성을 높이기 위해 로드 밸런서에 대한 지원이 추가되었습니다. 이제 Docker가 동적으로 컨테이너의 호스트 포트를 정의하게 할 수 있으며, ECS 스케줄러가 인스턴스:포트를 로드 밸런서에 등록합니다. 자세한 내용은 로드 밸런싱을 사용하여 Amazon ECS 서비스 트래픽 분산 섹션을 참조하세요.	2016년 8월 11일
Amazon ECS 태스크에 대한 IAM 역할	IAM 역할과 태스크를 연결하기 위한 지원이 추가되었습니다. 이에 따라 전체 컨테이너 인스턴스에 대해 한 가지 역할을 사용하는 대신 컨테이너에 대한 권한을 세부 정의할 수 있습니다. 자세한 내용은 Amazon ECS 작업 IAM 역할 섹션을 참조하세요.	2016년 7월 13일
Docker 1.11 지원	Docker 1.11 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2016년 5월 31일

변경 사항	설명	날짜
태스크 자동 조정	Amazon ECS가 서비스에서 실행하는 태스크를 자동으로 조정하는 지원을 추가했습니다. 자세한 내용은 Amazon ECS 서비스 자동 조정 섹션을 참조하세요.	2016년 5월 18일
태스크 패밀리에서 태스크 정의 필터링	태스크 정의 패밀리에 따라 태스크 정의 목록을 필터링하는 지원이 추가되었습니다. 자세한 내용은 ListTaskDefinitions를 참조하세요.	2016년 17월 5일
Docker 컨테이너 및 Amazon ECS 에이전트 로깅	문제 해결을 위해 Amazon ECS가 컨테이너 인스턴스의 ECS 에이전트 및 Docker 컨테이너 로그를 CloudWatch Logs로 보내는 기능을 추가했습니다.	2016년 5월 5일
ECS 최적화 AMI가 이제 Amazon Linux 2016.03을 지원합니다.	ESC 최적화 AMI가 Amazon Linux 2016.03에 대한 지원을 추가했습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2016년 4월 5일
Docker 1.9 지원	Docker 1.9 지원이 추가되었습니다. 자세한 내용은 Amazon ECS 최적화 Linux AMI 섹션을 참조하세요.	2015년 12월 22일
클러스터 CPU 및 메모리 예약에 대한 CloudWatch 지표	Amazon ECS가 클러스터 CPU 및 메모리 예약에 대한 사용자 지정 CloudWatch 지표를 추가했습니다.	2015년 12월 22일
새로운 Amazon ECS 최초 실행 경험	Amazon ECS 콘솔 최초 실행 시 클릭 없이 역할을 생성하는 기능이 추가되었습니다.	2015년 11월 23일
가용 영역 전역에 작업 배치	Amazon ECS 서비스 스케줄러가 가용 영역 전체에 태스크를 배치하기 위한 지원을 추가했습니다.	2015년 10월 8일
Amazon ECS 클러스터 및 서비스에 대한 CloudWatch 지표	Amazon ECS에서 클러스터의 각 컨테이너 인스턴스, 서비스 및 태스크 정의 패밀리의 CPU 및 메모리 사용에 대한 사용자 지정 CloudWatch 지표를 추가했습니다. 이 새로운 지표를 통해 Auto Scaling 그룹을 사용하여 클러스터의 컨테이너 인스턴스를 조정하거나 사용자 지정 CloudWatch 경보를 생성할 수 있습니다.	2015년 8월 17일

변경 사항	설명	날짜
UDP 포트 지원	태스크 정의에서 UDP 포트에 대한 지원이 추가되었습니다.	2015년 7월 7일
환경 변수 재정의	runTask에 대한 환경 변수 재정의 및 deregisterTaskDefinition에 대한 지원이 추가되었습니다.	2015년 6월 18일
자동 Amazon ECS 에이전트 업데이트	컨테이너 인스턴스에서 실행 중인 ECS 에이전트 버전을 확인하는 기능이 추가되었습니다. 또한 AWS Management Console, AWS CLI, SDK에서 ECS 에이전트를 업데이트할 수 있습니다.	2015년 6월 11일
Amazon ECS 서비스 스케줄러 및 Elastic Load Balancing 통합	서비스를 정의하고 이 서비스를 Elastic Load Balancing 로드 밸런서와 연결하는 기능이 추가되었습니다.	2015년 4월 9일
Amazon ECS GA	Amazon ECS가 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄) 및 유럽(아일랜드) 리전에서 정식으로 출시되었습니다.	2015년 4월 9일