AWS Well-Architected 프레임워크

# 안정성 원칙

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

# 안정성 원칙: AWS Well-Architected 프레임워크

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

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon 계열사, 관련 업체 또는 Amazon의 지원 업체 여부에 상관없이 해당 소유자의 자산입니다.

# Table of Contents


요약 및 소개 ... 1
소개 ... 1
신뢰성 ... 2
복원력을 위한 공동 책임 모델 ... 2
설계 원칙 ... 5
정의 ... 6
복원력과 신뢰성의 구성 요소 ... 6
가용성 ... 7
재해 복구(DR) 목표 ... 10
가용성 요구 사항 파악 ... 11
기본 ... 13
서비스 할당량 및 제약 조건 관리 ... 13
REL01-BP01 서비스 할당량 및 제약 조건 인식 ... 14
REL01-BP02 계정 및 리전 전체에서 서비스 할당량 관리 ... 19
REL01-BP03 아키텍처를 통해 고정된 서비스 할당량 및 제약 조건 수용 ... 23
REL01-BP04 할당량 모니터링 및 관리 ... 26
REL01-BP05 할당량 관리 자동화 ... 30
REL01-BP06 현재의 할당량과 최대 사용량 간에 장애 조치를 수용할 만큼 여유가 충분히 있는지 확인 ... 32
네트워크 토폴로지 계획 ... 36
REL02-BP01 워크로드 퍼블릭 엔드포인트에 고가용성 네트워크 연결 사용 ... 36
REL02-BP02 클라우드와 온프레미스 환경의 프라이빗 네트워크 간에 중복 연결 프로비저닝 ... 41
REL02-BP03 확장 및 가용성을 위한 IP 서브넷 할당 계정 확인 ... 44
REL02-BP04 다대다 메시보다 허브 앤 스포크 토폴로지 선호 ... 46
REL02-BP05 연결된 모든 프라이빗 주소 공간에서 겹치지 않는 프라이빗 IP 주소 범위 적용 ... 49
워크로드 아키텍처 ... 52
워크로드 서비스 아키텍처 설계 ... 52
REL03-BP01 워크로드를 세그먼트화하는 방법 선택 ... 53
REL03-BP02 특정 비즈니스 도메인 및 기능을 중심으로 서비스 구축 ... 56
REL03-BP03 API별로 서비스 계약 제공 ... 59
장애 방지를 위해 분산 시스템에서 상호 작용 설계 ... 62
REL04-BP01 사용 중인 분산 시스템의 종류 파악 ... 63

REL04-BP02 느슨하게 결합된 종속성 구현 ... 67
REL04-BP03 일정한 작업 처리 ... 71
REL04-BP04 변경 작업에 멱등성 부여 ... 72
장애 완화 또는 극복을 위해 분산 시스템에서 상호 작용 설계 ... 77
REL05-BP01 관련 하드 종속성을 소프트 종속성으로 변환하는 단계적 성능 저하 구현 ... 78
REL05-BP02 요청 제한 ... 81
REL05-BP03 재시도 직접 호출 제어 및 제한 ... 85
REL05-BP04 빠른 실패 및 대기열 제한 ... 87
REL05-BP05 클라이언트 제한 시간 설정 ... 90
REL05-BP06 가능한 경우 시스템을 상태 비저장으로 설계 ... 94
REL05-BP07 비상 레버 구현 ... 96
변경 관리 ... 99
워크로드 리소스 모니터링 ... 99
REL06-BP01 워크로드의 모든 구성 요소 모니터링(생성) ... 100
REL06-BP02 지표 정의 및 계산(집계) ... 103
REL06-BP03 알림 전송(실시간 처리 및 경보) ... 107
REL06-BP04 응답 자동화(실시간 처리 및 경보) ... 111
REL06-BP05 로그 분석 ... 113
REL06-BP06 모니터링 범위 및 지표를 정기적으로 검토 ... 115
REL06-BP07 시스템을 통한 요청의 엔드 투 엔드 추적 모니터링 ... 118
수요 변경에 따라 조정되는 워크로드 설계 ... 120
REL07-BP01 리소스를 확보하거나 조정할 때 자동화 사용 ... 121
REL07-BP02 워크로드 장애 감지 시 리소스 확보 ... 124
REL07-BP03 워크로드에 더 많은 리소스가 필요한 것으로 감지되면 리소스 확보 ... 125
REL07-BP04 워크로드 로드 테스트 ... 129
변경 구현 ... 130
REL08-BP01 배포와 같은 표준 활동에 런북 사용 ... 131
REL08-BP02 배포의 일부로 기능 테스트 통합 ... 133
REL08-BP03 배포의 일부로 복원력 테스트 통합 ... 135
REL08-BP04 변경 불가능한 인프라를 사용하여 배포 ... 137
REL08-BP05 자동화를 통한 변경 사항 배포 ... 141
장애 관리 ... 144
데이터 백업 ... 144
REL09-BP01 백업해야 하는 모든 데이터 확인 및 백업 또는 소스에서 데이터 복제 ... 145
REL09-BP02 백업 보안 및 암호화 ... 148
REL09-BP03 자동으로 데이터 백업 수행 ... 150

REL09-BP04 백업 무결성 및 프로세스를 확인하기 위해 데이터의 주기적인 복구 수행 ..... 153
장애 격리를 사용하여 워크로드 보호 ..... 156
REL10-BP01 워크로드를 여러 위치에 배포 ..... 156
REL10-BP02 단일 위치로 제약된 구성 요소의 복구 자동화 ..... 164
REL10-BP03 격벽 아키텍처를 사용하여 영향 범위 제한 ..... 165
구성 요소 장애를 견디도록 워크로드 설계 ..... 169
REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지 ..... 170
REL11-BP02 정상 리소스로 장애 조치 ..... 173
REL11-BP03 모든 계층에서 복구 자동화 ..... 176
REL11-BP04 복구 중 컨트롤 플레인이 아닌 데이터 플레인 사용 ..... 180
REL11-BP05 정적 안정성을 사용하여 바이모달 동작 방지 ..... 184
REL11-BP06 이벤트가 가용성에 영향을 미치는 경우 알림 전송 ..... 187
REL11-BP07 가용성 목표 및 가동 시간 서비스 수준에 관한 계약(SLA)을 충족하도록 제품 설계 ..... 190
신뢰성 테스트 ..... 192
REL12-BP01 플레이북을 사용하여 장애 조사 ..... 193
REL12-BP02 인시던트 사후 분석 수행 ..... 195
REL12-BP03 확장성 및 성능 요구 사항 테스트 ..... 197
REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트 ..... 201
REL12-BP05 정기적으로 게임 데이 진행 ..... 210
재해 복구(DR) 계획 ..... 213
REL13-BP01 가동 중단 시간 및 데이터 손실 시의 복구 목표 정의 ..... 214
REL13-BP02 복구 목표 달성을 위해 정의된 복구 전략 사용 ..... 218
REL13-BP03 재해 복구 구현을 테스트하여 구현 확인 ..... 231
REL13-BP04 DR 사이트 또는 리전에서 구성 드리프트 관리 ..... 232
REL13-BP05 자동 복구 ..... 235
결론 ..... 239
기여자 ..... 240
참조 자료 ..... 241
문서 수정 ..... 242
고지 사항 ..... 247
AWS 용어집 ..... 248

# 신뢰성 원칙 - AWS Well-Architected Framework

발행일: 2024년 11월 6일(문서 수정)

이 백서에서는 AWS Well-Architected Framework의 신뢰성 원칙을 중점적으로 다룹니다. 또한 Amazon Web Services(AWS) 환경 설계, 제공 및 유지 관리에 모범 사례를 적용할 때 참조할 수 있는 지침을 제공합니다.

## 소개

AWS Well-Architected Framework를 활용하면 AWS에서 워크로드를 구축하면서 내리게 되는 결정의 장단점을 이해할 수 있습니다. 이 프레임워크를 사용하면 클라우드에서 신뢰할 수 있고 안전하며 효율적이고 경제적이면서 지속 가능한 워크로드를 설계하고 운영하기 위한 아키텍처 모범 사례를 알아볼 수 있습니다. 또한 모범 사례와 비교하여 아키텍처를 지속적으로 측정하고 개선할 영역을 파악할 수 있습니다. 워크로드를 제대로 설계하면 비즈니스 성공 가능성이 커집니다.

AWS Well-Architected Framework는 6가지 원칙을 기반으로 합니다.

- 운영 우수성
- 보안
- 신뢰성
- 성능 효율성
- 비용 최적화
- 지속 가능성

이 백서에서는 신뢰성 원칙 및 해당 부문을 솔루션에 적용하는 방법에 대해 중점적으로 설명합니다. 기존의 온프레미스 환경에서는 단일 장애 지점과 자동화의 부재, 탄력성의 부재로 인해 신뢰성을 높이기가 어렵습니다. 이 문서에서 설명하는 사례를 도입하면 안정적인 토대, 탁월한 복원력의 아키텍처와 일관성 있는 변경 관리 기능 및 검증된 장애 복구 프로세스를 갖춘 아키텍처를 구축할 수 있습니다.

이 문서는 최고 기술 책임자(CTO), 아키텍트, 개발자와 같은 기술 업무 담당자와 운영 팀원을 대상으로 작성되었습니다. 이 백서의 내용을 확인하고 나면 신뢰성이 뛰어난 클라우드 아키텍처를 설계할 때 사용할 AWS 모범 사례와 전략을 파악할 수 있습니다. 이 백서에는 개요 수준의 구현 세부 정보와 아키텍처 패턴 및 추가 리소스 참조가 포함되어 있습니다.

# 신뢰성

신뢰성 원칙에서는 워크로드의 기능이 필요한 때에 기능을 정확하고 일관되게 수행하는 역량에 대해 다룹니다. 여기에는 전체 수명 주기에 걸쳐 워크로드를 운영 및 테스트할 수 있는 기능이 포함됩니다. 이 백서는 AWS에서 신뢰할 수 있는 워크로드를 구현하기 위한 세부적인 모범 사례 지침을 제공합니다.

주제


- 복원력을 위한 공동 책임 모델
- 설계 원칙
- 정의
- 가용성 요구 사항 파악


## 복원력을 위한 공동 책임 모델

보안은 AWS와 고객이 공동으로 책임을 져야 하는 영역입니다. 복원력의 일부인 재해 복구(DR) 및 가용성이 이 공동 모델에서 어떻게 작동하는지 이해해야 합니다.

AWS 책임 - 클라우드의 복원력

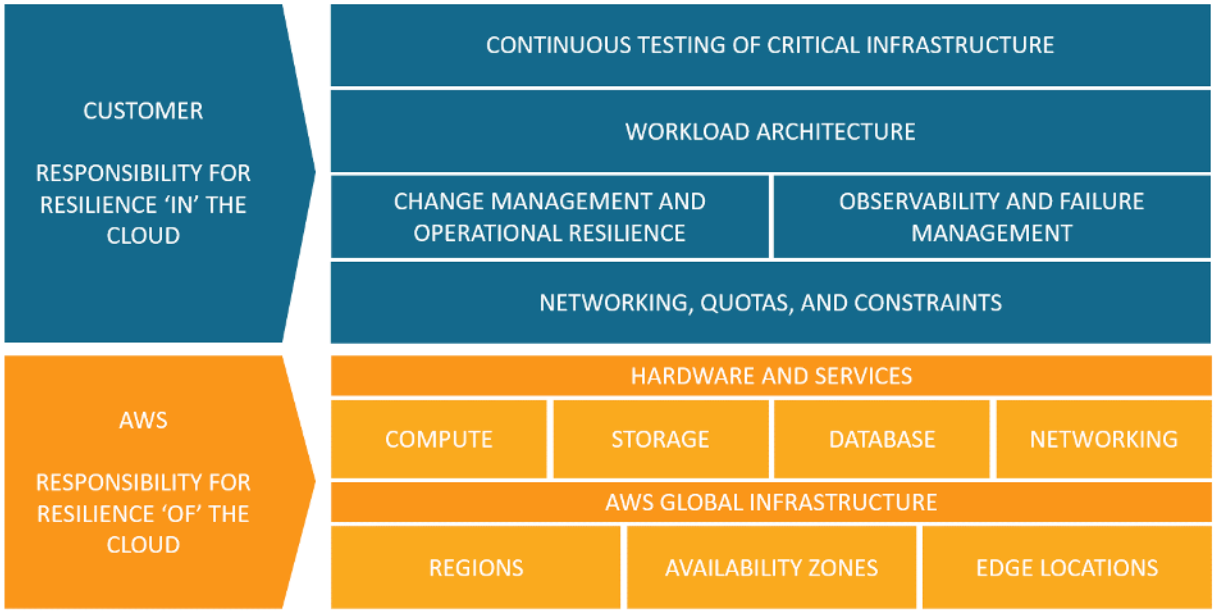
AWS는 AWS 클라우드에서 제공하는 모든 서비스가 실행되는 인프라의 복원력에 대한 책임이 있습니다. 이 인프라는 AWS 클라우드 서비스를 실행하는 하드웨어, 소프트웨어, 네트워킹 및 시설로 구성됩니다. AWS는 이러한 AWS 클라우드 서비스를 제공하기 위해 상업적으로 합당한 노력을 기울이며 서비스 가용성이 AWS 서비스 수준에 관한 계약(SLA) 이상을 충족하도록 보장합니다.

AWS 글로벌 클라우드 인프라는 고객이 복원력이 뛰어난 워크로드 아키텍처를 구축할 수 있도록 설계되었습니다. 각 AWS 리전은 완전히 격리되어 있으며 물리적으로 격리된 인프라 파티션인 여러 가용 영역으로 구성됩니다. 가용 영역은 워크로드 복원력에 영향을 줄 수 있는 결함을 격리하여 리전의 다른 영역에 영향을 미치지 않도록 합니다. 그러나 동시에 AWS 리전의 모든 영역은 완전히 중복된 전용 메트로 섬유를 사용하여 고대역폭과 짧은 지연 시간의 네트워킹으로 상호 연결되어 있기 때문에 영역 간에 높은 처리량과 짧은 지연 시간을 제공합니다. 영역 간의 모든 트래픽은 암호화됩니다. 네트워크 성능은 영역 간에 동기식 복제를 수행하기에 충분합니다. 하나의 애플리케이션이 여러 AZ에 파티셔닝되어 있는 경우 정전, 번개, 토네이도, 허리케인 등의 문제로부터 애플리케이션을 더 효과적으로 격리하고 보호할 수 있습니다.

고객 책임 - 클라우드의 복원력

고객의 책임은 고객이 선택한 AWS 클라우드 서비스에 따라 결정됩니다. 서비스에 따라 복원력 책임의 일환으로서 고객이 수행해야 할 구성 작업의 양이 달라집니다. 예를 들어 Amazon Elastic Compute Cloud(Amazon EC2)와 같은 서비스를 사용하려면 고객이 필요한 모든 복원력 구성 및 관리 작업을 수행해야 합니다. Amazon EC2 인스턴스를 배포하는 고객은 Amazon EC2 인스턴스를 여러 위치(예: AWS 가용 영역)에 배포하고, Auto Scaling과 같은 서비스를 사용하여 자가 복구를 구현하며, 인스턴스에 설치된 애플리케이션에 대한 복원력이 뛰어난 워크로드 아키텍처 모범 사례를 사용할 책임이 있습니다. Amazon S3 및 Amazon DynamoDB와 같은 관리형 서비스의 경우 AWS는 인프라 계층, 운영 체제, 플랫폼을 작동하고, 고객은 엔드포인트에 액세스하여 데이터를 저장 및 검색합니다. 백업, 버전 관리 및 복제 전략을 포함하여 데이터의 복원력을 관리할 책임은 고객에게 있습니다.

AWS 리전의 여러 가용 영역에 워크로드를 배포하는 것은 하나의 가용 영역으로 문제를 격리하고 다른 가용 영역의 중복성을 사용하여 요청을 계속 처리하여 워크로드를 보호하도록 설계된 고가용성 전략의 일부입니다. 다중 AZ 아키텍처는 정전, 낙뢰, 토네이도, 지진 등과 같은 문제로부터 워크로드를 더 잘 격리하고 보호하도록 설계된 DR 전략의 일부이기도 합니다. DR 전략은 여러 AWS 리전을 사용할 수도 있습니다. 예를 들어 액티브/패시브 구성에서 액티브 리전이 더 이상 요청을 처리할 수 없는 경우 워크로드에 대한 서비스가 액티브 리전에서 DR 리전으로 장애 조치됩니다.




클라우드 자체 및 내부 복원력에 대한 고객 및 AWS의 책임.

AWS 서비스를 사용하여 복원력 목표를 달성할 수 있습니다. 고객은 클라우드에서 복원력을 달성하기 위해 시스템의 다음 측면을 관리할 책임이 있습니다. 특히 각 서비스에 대한 자세한 내용은 AWS 설명서를 참조하세요.

네트워킹, 할당량 및 제약 조건

- 공동 책임 모델의 이 영역에 대한 모범 사례는 기초 페이지에 자세히 설명합니다.
- 해당하는 경우 예상되는 로드 요청 증가에 따라 포함하는 서비스의 서비스 할당량 및 제약 조건을 이해하고 규모를 조정할 수 있는 충분한 공간이 있는 아키텍처를 계획합니다.
- 고가용성의 확장 가능한 중복 네트워크로 네트워크 토폴로지를 설계합니다.

변경 관리 및 운영 복원력

- 변경 관리에는 환경에 변경 사항을 도입하고 관리하는 방법이 포함됩니다. 변경 사항을 구현하려면 애플리케이션 및 인프라에 대한 런북과 배포 전략을 구축하고 최신 상태로 유지해야 합니다.
- 워크로드 리소스를 모니터링하는 탄력적인 전략에서는 기술 및 비즈니스 지표, 알림, 자동화 및 분석을 포함한 모든 구성 요소를 고려합니다.
- 클라우드의 워크로드는 사용량 장애 또는 변동에 대응하여 스케일 인되는 수요 규모의 변화에 적응해야 합니다.

관찰성 및 장애 관리

- 워크로드가 구성 요소 장애를 견딜 수 있도록 복구를 자동화하려면 모니터링을 통해 장애를 관찰해야 합니다.
- 장애 관리를 위해 데이터를 백업하고, 워크로드가 구성 요소 장애를 견딜 수 있도록 모범 사례를 적용하고, 재해 복구를 계획해야 합니다.

워크로드 아키텍처

- 워크로드 아키텍처에는 비즈니스 도메인을 중심으로 서비스를 설계하는 방법, 장애를 방지하기 위해 SOA 및 분산 시스템 설계를 적용하는 방법, 스로틀링, 재시도, 대기열 관리, 제한 시간 및 비상 레버와 같은 기능을 구축하는 방법이 포함됩니다.
- 입증된 AWS 솔루션, Amazon Builders Library 및 서버리스 패턴을 활용하여 모범 사례에 맞춰 구현을 바로 시작할 수 있습니다.
- 지속적인 개선을 통해 시스템을 분산 서비스로 분해하여 더 빠르게 규모를 조정하고 혁신합니다. AWS 마이크로서비스 지침 및 관리형 서비스 옵션을 사용하여 변경을 도입하고 혁신하는 역량을 단순화하고 가속화합니다.

중요 인프라에 대한 지속적인 테스트

- 신뢰성 테스트는 기능, 성능, 카오스 수준에서 테스트하고, 인시던트 분석 및 게임 데이 관행을 채택하여 잘 이해되지 않은 문제를 해결하는 데 필요한 전문성을 구축함을 의미합니다.
- 클라우드 올인 및 하이브리드 애플리케이션 모두에서 문제가 발생하거나 구성 요소가 중단될 때 애플리케이션이 어떻게 작동하는지 알면 중단으로부터 빠르고 신뢰할 수 있는 방식으로 복구할 수 있습니다.
- 예상대로 작동하지 않을 때 시스템이 어떻게 작동하는지 이해하기 위해 반복 가능한 실험을 만들고 문서화합니다. 이러한 테스트는 전체 복원력의 효율성을 입증하고 실제 오류 시나리오에 직면하기 전에 운영 절차에 대한 피드백 루프를 제공합니다.

## 설계 원칙

클라우드에서는 몇 가지 원칙에 따라 신뢰성을 높일 수 있습니다. 여기서 설명하는 모범 사례와 관련하여 숙지해야 할 원칙은 다음과 같습니다.

- 장애 자동 복구: 워크로드의 핵심 성과 지표(KPI) 모니터링을 통해 임곗값을 위반하면 자동화를 실행할 수 있습니다. 이러한 KPI는 서비스의 기술적 측면이 아닌 비즈니스 가치에 기반한 측정한 값이어야 합니다. KPI를 모니터링하면 장애 추적 및 자동 알림을 지원하고, 자동화된 복구 프로세스에 따라 장애 지점을 우회하거나 복구할 수 있습니다. 보다 정교한 자동화를 구현할 경우 장애가 발생하기 전에 예측하여 해결하는 것도 가능합니다.
- 복구 절차 테스트: 온프레미스 환경에서 테스트는 워크로드가 특정 시나리오에서 작동하는 것을 증명하기 위해 시행됩니다. 일반적으로 복구 전략을 검증하기 위해 테스트하지는 않습니다. 클라우드에서는 워크로드의 장애 과정을 테스트하고 복구 절차를 검증할 수 있습니다. 자동화를 사용하여 다양한 장애를 시뮬레이션하거나 이전에 장애로 이어졌던 시나리오를 재현할 수 있습니다. 이 접근 방식은 장애 경로를 노출한 후 실제 장애 시나리오가 발생하기 전에 테스트하고 수정하여 위험을 줄일 수 있습니다.
- 수평적 스케일링을 통해 전체 워크로드 가용성 증대: 단일의 큰 리소스를 다수의 작은 리소스로 대체하여 단일 장애가 전체 워크로드에 미치는 영향을 축소합니다. 요청을 더 작은 리소스 여러 개로 분산시키면 공통의 장애 지점이 공유되지 않습니다.
- 용량 추정 중지: 워크로드에 대한 수요가 해당 워크로드의 용량을 넘어서는 리소스 부족 상태는 온프레미스 워크로드에서 흔히 발생하는 장애의 원인입니다(서비스 거부 공격의 대상). 클라우드에서는 수요 및 워크로드 사용량을 모니터링하고 리소스 추가 또는 제거를 자동화함으로써 프로비저닝 과다 또는 부족 현상 없이 최적의 수준으로 수요를 충족할 수 있습니다. 클라우드에도 제한은 있지만 할당량을 어느 정도 제어하고 관리하는 것이 가능합니다(Service Quotas 및 제약 조건 관리 참조).
- 자동화를 통한 변경 관리: 인프라 변경은 자동화를 통해 수행되어야 합니다. 관리가 필요한 변경에는 자동화 변경이 포함되며, 이후에 이러한 변경을 추적하고 검토할 수 있습니다.

# 정의

이 백서에서는 클라우드의 신뢰성에 대해 다루며 다음 4개 영역에 대한 모범 사례를 설명합니다.

- 기본
- 워크로드 아키텍처
- 변경 관리
- 장애 관리

신뢰성을 달성하려면 기반부터 시작해야 합니다. 이러한 기반은 서비스 할당량과 네트워크 토폴로지로 워크로드를 수용하는 환경을 의미합니다. 분산 시스템의 워크로드 아키텍처는 장애를 예방하고 완화하도록 설계되어야 합니다. 워크로드는 수요 또는 요구 사항의 변경을 처리해야 하며, 장애를 감지하고 자동으로 복구되도록 설계되어야 합니다.

주제

- 복원력과 신뢰성의 구성 요소
- 가용성
- 재해 복구(DR) 목표

## 복원력과 신뢰성의 구성 요소

클라우드에서 워크로드의 신뢰성은 몇 가지 요인에 의존하는데 가장 기본적인 요소는 복원력입니다.

- 복원력은 인프라 또는 서비스 중단을 복구하고, 수요에 따라 컴퓨팅 리소스를 탄력적으로 확보하고, 구성 오류나 일시적 네트워크 문제 같은 중단 사태를 완화할 수 있는 워크로드의 능력입니다.

워크로드 신뢰성에 영향을 미치는 다른 요인은 다음과 같습니다.

- 운영 우수성: 변경 자동화, 장애 대응을 위한 플레이북 사용, 애플리케이션의 프로덕션 운영 준비 상태를 확인하는 운영 준비 검토(ORR)가 포함됩니다.
- 보안: 악의적 행위자가 데이터 또는 인프라를 손상시켜 가용성에 영향을 주는 행위를 방지하는 조치가 포함됩니다. 예를 들어 백업을 암호화하여 데이터를 안전하게 보호합니다.
- 성능 효율성: 요청 속도를 최대화하고 워크로드 지연 시간을 최소화하는 설계가 포함됩니다.
- 비용 최적화: EC2 인스턴스에 대한 지출을 늘려 정적 안정성을 달성할지 추가 용량이 필요할 때 자동 규모 조정을 사용할지 여부와 같은 절충이 포함됩니다.

이 백서에서는 복원력을 중점적으로 다룹니다.

다른 네 가지 측면도 중요하며, AWS Well-Architected Framework의 각 원칙에서 다룹니다. 여기에 나오는 모범 사례의 많은 부분은 신뢰성의 그러한 측면을 다루지만 중점은 복원력입니다.

## 가용성

가용성(서비스 가용성이라고도 함)은 복원력을 정량적으로 측정하는 데 일반적으로 사용되는 지표이자 목표 복원력 목표이기도 합니다.

- 가용성은 워크로드를 사용할 수 있는 시간의 비율입니다.

사용 가능이란 용어는 필요할 때 합의된 기능을 성공적으로 수행할 수 있음을 의미합니다.

이 비율은 월, 연 또는 이후 3년과 같이 특정 기간에 걸쳐 계산됩니다. 가장 엄격한 해석을 적용할 때 가용성은 예정된 중단 및 예정되지 않은 중단을 포함하여 애플리케이션이 정상적으로 작동하지 않는 모든 시간에 감소합니다. 가용성은 다음과 같이 정의합니다.

$$Availability = \frac{Available\ for\ Use\ Time}{Total\ Time}$$

- 가용성은 일정 기간(대개 1개월 또는 1년)의 가동 시간 비율(예: 99.9%)입니다.
- 흔히 사용되는 '5개의 9'는 99.999%의 가용성을 의미합니다.
- 수식의 총 시간에서 예정된 서비스 가동 중지 시간(예: 계획된 유지 관리)을 제외하는 경우도 있습니다. 그러나 사용자가 이 기간에 서비스를 사용하고자 하므로 이것은 권장되는 방법이 아닙니다.

아래 표에는 일반적인 애플리케이션 가용성 설계 목표와 해당 목표를 충족하면서 1년 동안 허용되는 최대 중단 시간의 길이가 나와 있습니다. 이 표에는 각 가용성 계층에서 흔히 볼 수 있는 애플리케이션 유형의 예가 포함되어 있습니다. 여기에 나온 값이 이 문서 전체에서 참조됩니다.

| 가용성 | 최대 사용 불가(연간) | 애플리케이션 범주 |
| --- | --- | --- |
| 99% | 3일 15시간 | 배치 처리, 데이터 추출, 전송 및 로드 작업 |
| 99.9% | 8시간 45분 | 지식 관리, 프로젝트 추적과 같은 내부 도구 |

| 가용성 | 최대 사용 불가(연간) | 애플리케이션 범주 |
| --- | --- | --- |
| 99.95% | 4시간 22분 | 온라인 상거래, POS(Point of Sale) |
| 99.99% | 52분 | 비디오 전송, 브로드캐스트 워크로드 |
| 99.999% | 5분 | ATM 트랜잭션, 통신 워크로드 |

요청을 기반으로 가용성을 측정합니다. '사용할 수 있는 시간' 대신 성공 및 실패한 요청의 수를 세는 것이 서비스에 따라 더 용이할 수 있습니다. 그런 경우 다음 계산을 사용하면 됩니다.

$$Availability = \frac{Successful\ Responses}{Valid\ Requests}$$

이것은 대개 1분 또는 5분 기간에 측정됩니다. 그런 다음, 이 기간의 평균에서 월간 가동 시간 비율(시간 기반 가용성 측정)을 계산할 수 있습니다. 주어진 기간에 요청이 수신되지 않으면 해당 기간의 가용성이 100%로 계산됩니다.

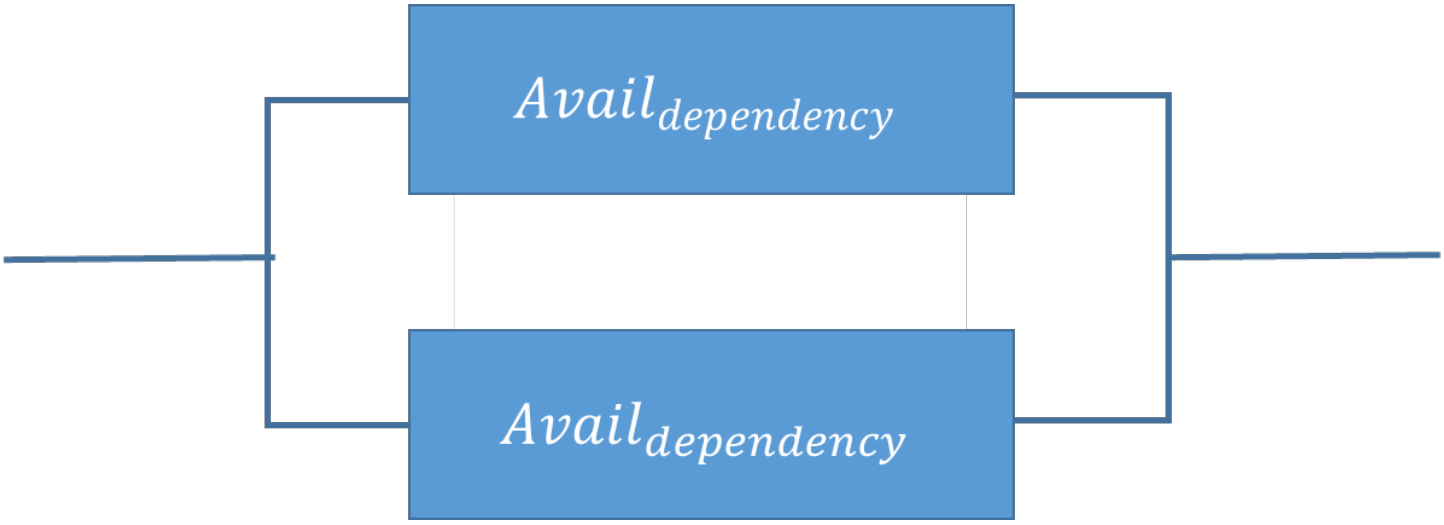
강한 종속성의 가용성 계산. 대다수 시스템에서는 다른 시스템에 대한 하드 종속성이 적용됩니다. 즉, 종속 시스템이 중단되면 간접 호출 시스템도 중단됩니다. 반면 약한 종속성이 적용되는 경우에는 종속 시스템의 장애가 애플리케이션에서 보정됩니다. 이러한 강한 종속성이 적용되는 경우 간접 호출 시스템 가용성은 종속 시스템 가용성을 곱한 결과입니다. 예를 들어 99.99%의 가용성을 제공하도록 설계된 시스템에 다른 두 독립 시스템에 대한 강한 종속성이 적용되며, 두 독립 시스템의 가용성도 99.99%라면 이론적으로 해당 워크로드가 달성할 수 있는 가용성은 99.97%입니다.

$Avail_{invok} \times Avail_{dep1} \times Avail_{dep2} = Avail_{workload}$

99.99% × 99.99% × 99.99% = 99.97%

따라서 가용성을 계산할 때는 종속성과 가용성 설계 목표를 파악해야 합니다.

중복 구성 요소를 사용한 가용성 계산. 시스템에서 예를 들어 서로 다른 가용 영역의 중복 리소스 등 독립된 중복 구성 요소를 사용하는 경우 이론적으로 가용성은 100%에서 구성 요소 장애율을 곱한 값을 뺀 결과로 계산됩니다. 예를 들어 시스템이 두 독립 구성 요소를 사용하며 각 구성 요소의 가용성이 99.9%인 경우 이 종속성의 유효 가용성은 99.9999%입니다.




$Avail_{effective} = Avail_{MAX} - ((100\% - Avail_{dependency}) \times (100\% - Avail_{dependency}))$

$99.9999\% = 100\% - (0.1\% \times 0.1\%)$

빠른 계산: 계산에 포함된 모든 구성 요소의 가용성이 9로만 이루어진 숫자라면 9의 개수를 더하여 답을 얻으면 됩니다. 위 예에서 두 개의 중복된 독립 구성 요소는 각각 가용성을 나타내는 9가 3개이므로 결과적으로 9가 6개입니다.

종속성의 가용성 계산. 대다수 AWS 서비스의 가용성 설계 목표를 포함하여 가용성 관련 지침을 제공하는 종속성도 있습니다. 하지만 제조업체에서 가용성 정보를 게시하지 않은 구성 요소와 같이 이 가용성이 제공되지 않는 경우 가용성을 예측할 수 있는 한 가지 방법은 평균 고장 간격(MTBF) 및 평균 복구 시간(MTTR)을 확인하는 것입니다. 예상 가용성을 계산하는 수식은 다음과 같습니다.

$$Avail_{EST} = \frac{MTBF}{MTBF + MTTR}$$

예를 들어 MTBF가 150일이고 MTTR이 1시간인 경우 예상 가용성은 99.97%입니다.

자세한 내용은 Availability and Beyond: Understanding and improving the resilience of distributed systems on AWS를 참조하세요. 가용성 계산에 도움을 줄 수 있습니다.

가용성에 대한 비용. 가용성 수준을 높이도록 애플리케이션을 설계하면 일반적으로 비용이 증가하므로, 애플리케이션 설계를 시작하기 전에 실제 가용성 요구를 파악하는 것이 좋습니다. 가용성 수준이 높으면 전체 장애 시나리오에서 더 엄격한 테스트 및 확인 요구 사항이 적용됩니다. 즉, 모든 장애에서 자동으로 복구할 수 있어야 하며 시스템 운영의 모든 측면을 유사하게 구축하고 동일한 표준에 따라 테

스트해야 합니다. 예를 들어 원하는 가용성 목표를 달성할 수 있도록 용량 추가 또는 제거, 업데이트된 소프트웨어 또는 구성 변경 사항 배포 또는 롤백, 시스템 데이터 마이그레이션 등을 수행해야 합니다. 원하는 가용성 수준이 매우 높으면 소프트웨어 개발 비용이 더 많이 발생하며, 그러면 시스템 배포 속도를 늦춰야 하므로 혁신을 추진하기가 어려워집니다. 따라서 표준을 철저하게 적용하고 전체 시스템 작동 수명 주기에서 적절한 가용성 목표를 고려해야 합니다.

가용성 설계 목표가 높게 설정된 상태로 작동하는 시스템에서는 종속성 선택으로 인해서도 비용이 증가할 수 있습니다. 가용성 목표를 높게 설정하면 앞에서 설명한 것과 같은 세부 투자를 진행했던 서비스에 따라 종속성으로 선택할 수 있는 소프트웨어나 서비스 세트가 줄어듭니다. 그러므로 가용성 설계 목표 수준이 높아지면 관계형 데이터베이스 등의 다목적 서비스 수는 줄이고 특별히 구축된 서비스 수는 늘리는 것이 일반적입니다. 특별히 구축된 서비스는 더 쉽게 평가/테스트/자동화할 수 있으며, 포함은 되어 있지만 사용하지는 않는 기능과 예기치 않게 상호 작용할 가능성을 줄일 수 있기 때문입니다.

## 재해 복구(DR) 목표

가용성 목표 외에도 복원력 전략에는 재해 이벤트 시 워크로드를 복구하기 위한 전략에 따른 재해 복구(DR) 목표를 포함해야 합니다. 재해 복구는 자연 재해, 대규모의 기술적 장애 또는 공격 또는 오류와 같은 사람의 위협 등에 대응하여 일회성 복구 목표에 집중합니다. 이는 구성 요소 장애, 로드 스파이크, 소프트웨어 버그에 대응하여 주어진 기간의 평균 복원력을 측정하는 가용성과 다릅니다.

Recovery Time Objective(RTO)는 조직에서 정의합니다. RTO는 서비스 중단과 서비스 복원 사이의 허용 가능한 최대 지연 시간입니다. 이는 서비스를 이용할 수 없을 때 허용 가능한 기간으로 간주되는 기간을 결정합니다.

Recovery Point Objective(RPO)는 조직에서 정의합니다. RPO는 마지막 데이터 복구 시점 이후 허용되는 최대 시간입니다. 이에 따라 마지막 복구 시점과 서비스 중단 사이에 허용되는 데이터 손실로 간주되는 범위가 결정됩니다.

Recovery Point Objective(RPO), Recovery Time Objective(RTO), 재해 이벤트의 관계.

RTO는 중단이 시작된 시점과 워크로드 복구 사이의 시간을 측정한다는 점에서 평균 복구 시간(MTTR)과 유사합니다. 하지만 MTTR은 일정 기간에 여러 가용성에 영향을 미치는 이벤트에 대한 평균 값이며, RTO는 단일 가용성에 영향을 미치는 이벤트의 목표 값 또는 최대 허용 값입니다.

## 가용성 요구 사항 파악

애플리케이션의 가용성은 대개 애플리케이션 전체의 단일 목표로 간주되는 경우가 많습니다. 하지만 자세히 살펴보면 애플리케이션이나 서비스의 특정 측면마다 가용성 요구 사항이 각기 다른 경우가 많습니다. 예를 들어 기존 데이터를 검색하기 전에 새 데이터를 수신하여 저장하는 기능이 더 중요한 시스템도 있습니다. 시스템 구성이나 환경을 변경하는 작업보다 실시간 작업을 우선적으로 처리하는 시스템도 있습니다. 그리고 특정 시간 동안에는 가용성 요구 사항이 매우 높지만 그 외의 시간에는 좀 더 오랫동안 중단되어도 되는 서비스도 있습니다. 이러한 점을 참조하여 애플리케이션 하나를 여러 구성 요소로 구분한 후 각 구성 요소의 가용성을 평가할 수 있습니다. 이렇게 하면 가장 엄격한 요구 사항에 따라 전체 시스템 엔지니어링을 진행하는 대신 특정 요구에 따라 가용성 관련 작업을 집중적으로 수행하고 경비를 중점 투입할 수 있습니다.

| 권장 사항 |
| --- |
| 애플리케이션의 고유한 측면을 중점적으로 평가하고, 해당하는 경우 비즈니스 요구 사항을 반영하도록 가용성 및 재해 복구 설계 목표를 구분합니다. |

AWS에서 서비스는 주로 '데이터 플레인'과 '컨트롤 플레인'으로 구분됩니다. 데이터 플레인에서는 실시간 서비스를 제공하고, 컨트롤 플레인은 환경을 구성하는 데 사용됩니다. 예를 들어 Amazon EC2 인스턴스, Amazon RDS 데이터베이스, Amazon DynamoDB 테이블 읽기/쓰기 작업은 모두 데이터 플레인 작업입니다. 반면 새 EC2 인스턴스나 RDS 데이터베이스 시작 또는 DynamoDB의 테이블 메타데이터 추가/변경은 모두 컨트롤 플레인 작업으로 간주됩니다. 이러한 모든 기능에는 높은 수준의 가용성이 중요하지만, 일반적으로는 데이터 플레인의 가용성 설계 목표가 컨트롤 플레인보다 높습니다. 따라서 가용성에 대한 요구 사항이 높은 워크로드는 컨트롤 플레인 작업에 대한 런타임 의존성을 피해야 합니다.

대부분의 AWS 고객은 비슷한 방식을 통해 애플리케이션을 면밀하게 평가한 다음 가용성 요구가 각기 다른 하위 구성 요소를 확인합니다. 그런 후에는 각 측면에 맞게 가용성 설계 목표를 조정하고 적절한 작업을 수행해 시스템을 엔지니어링합니다. AWS는 99.999% 이상의 가용성을 요구하는 서비스 등 다양한 가용성 설계 목표가 있는 애플리케이션을 엔지니어링하는 데 풍부한 경험을 가지고 있습니다. AWS 솔루션스 아키텍트(SA)는 가용성 목표에 적합한 설계를 생성하는 데 도움을 줄 수 있습니다. 설계 프로세스 초기에 AWS를 활용하면 가용성 목표를 더욱 효율적으로 충족할 수 있습니다. 가용성 계획은 워크로드를 시작하기 전에만 수행하는 작업이 아닙니다. 즉, 운영을 진행하면서 설계를 구체화하고, 실제 이벤트에서 정보를 파악하며, 다양한 유형의 장애 발생 시에 복구를 할 수 있도록 지속적으로 가용성을 계획해야 합니다. 그런 후에는 구현 개선을 위해 적합한 작업을 적용할 수 있습니다.

워크로드에 필요한 가용성 요구 사항은 비즈니스 요구 사항 및 중요도와 일치해야 합니다. 먼저 정의된 RTO, RPO 및 가용성을 사용하여 비즈니스 중요도 프레임워크를 정의한 다음 각 워크로드를 평가할 수 있습니다. 이러한 접근 방식을 사용하려면 워크로드 구현에 관여한 사람들이 프레임워크에 대해 잘 알고 있어야 하며, 워크로드가 비즈니스 요구 사항에 미치는 영향을 잘 알고 있어야 합니다.

# 기본

기반에 관한 요구 사항은 그 범위가 단일 워크로드 또는 프로젝트 이상으로 확장됩니다. 시스템을 설계할 때는 먼저 신뢰성을 좌우하는 기반에 관한 요구 사항부터 갖춰야 합니다. 예를 들어, 데이터 센터의 네트워크 대역폭을 충분히 확보해야 합니다.

온프레미스 환경의 경우, 이러한 요구 사항에 따른 종속성으로 인해 소요 시간이 길어질 수 있으므로 결국 초기 계획에 이를 반영해야 합니다. 그러나 AWS에는 이러한 기본 요구 사항이 대부분 이미 통합되어 있거나 필요에 따라 적용할 수 있습니다. 클라우드는 거의 한계가 없도록 설계되었기 때문에 충분한 네트워킹 및 컴퓨팅 파워에 대한 요구 사항을 충족할 책임은 AWS에 있습니다. 따라서 고객은 리소스 크기와 할당을 필요에 따라 자유롭게 변경할 수 있습니다.

다음 섹션에서는 이러한 신뢰성 고려 사항에 중점을 둔 모범 사례를 설명합니다.

주제

- 서비스 할당량 및 제약 조건 관리
- 네트워크 토폴로지 계획

## 서비스 할당량 및 제약 조건 관리

클라우드 기반 워크로드 아키텍처에는 서비스 할당량(서비스 한도라고도 함)이 있습니다. 이러한 할당량은 실수로 필요한 것보다 많은 리소스를 프로비저닝하는 것을 방지하고 API 작업에 대한 요청 비율을 제한하여 서비스가 남용되지 않도록 하기 위해 존재합니다. 또한 리소스에도 제약이 따릅니다. 예를 들면, 광섬유 케이블을 통해 비트를 전송할 수 있는 속도나 물리적 디스크의 스토리지 용량 등이 있습니다.

AWS Marketplace 애플리케이션을 사용하는 경우에는 이러한 애플리케이션의 제한에 대해 알고 있어야 합니다. 서드파티 웹 서비스 또는 서비스형 소프트웨어(SaaS)를 사용 중이라면 해당 서비스의 제한도 파악해야 합니다.

모범 사례

- REL01-BP01 서비스 할당량 및 제약 조건 인식
- REL01-BP02 계정 및 리전 전체에서 서비스 할당량 관리
- REL01-BP03 아키텍처를 통해 고정된 서비스 할당량 및 제약 조건 수용
- REL01-BP04 할당량 모니터링 및 관리
- REL01-BP05 할당량 관리 자동화

- REL01-BP06 현재의 할당량과 최대 사용량 간에 장애 조치를 수용할 만큼 여유가 충분히 있는지 확인

## REL01-BP01 서비스 할당량 및 제약 조건 인식

워크로드 아키텍처에 대한 기본 할당량을 파악하고 할당량 증가 요청을 관리합니다. 디스크 또는 네트워크 등 어떤 클라우드 리소스 제약 조건이 잠재적인 영향을 미치는지 파악합니다.

원하는 성과: 고객은 서비스 성능 저하 또는 중단을 일으킬 수 있는 서비스 할당량 및 제약 조건에 도달했는지 확인하기 위해 주요 지표, 인프라 검토 및 자동화 수정 단계를 모니터링하기 위한 적절한 지침을 구현하여 AWS 계정 계정에서 서비스 성능 저하 또는 중단을 방지할 수 있습니다.

일반적인 안티 패턴:

- 하드 또는 소프트 할당량과 사용되는 서비스에 대한 한도를 파악하지 않고 워크로드를 배포합니다.
- 필요한 할당량을 분석 및 재구성하거나 미리 지원 팀에 연락하지 않고 대체 워크로드를 배포합니다.
- 클라우드 서비스에는 한도가 없고 속도, 한도, 횟수, 수량 등을 고려하지 않고 서비스를 사용할 수 있다고 가정합니다.
- 할당량이 자동으로 증가할 것이라고 가정합니다.
- 할당량 요청의 프로세스 및 타임라인을 모릅니다.
- 기본 클라우드 서비스 할당량이 리전 간에 비교되는 모든 서비스에 대해 동일하다고 가정합니다.
- 서비스 제약 조건은 위반할 수 있고 시스템에서 리소스의 제약 조건을 벗어나 자동으로 규모를 조정하거나 한도 증가를 추가한다고 가정합니다.
- 리소스 사용률을 강조하기 위해 피크 트래픽에서 애플리케이션을 테스트하지 않습니다.
- 필요한 리소스 크기를 분석하지 않고 리소스를 프로비저닝합니다.
- 실제 필요 또는 예상 피크를 잘 벗어나는 리소스 유형을 선택하여 용량을 오버프로비저닝합니다.
- 새로운 고객 이벤트 또는 새로운 기술 배포에 앞서 새로운 수준의 트래픽에 대한 용량 요구 사항을 미리 평가하지 않습니다.

이 모범 사례 확립의 이점: 서비스 할당량 및 리소스 제약 조건을 모니터링하고 관리를 자동화하면 사전에 실패를 줄일 수 있습니다. 모범 사례를 따르지 않으면 고객 서비스에 대한 트래픽 패턴의 변화로 인해 서비스 중단 또는 성능 저하가 발생할 수 있습니다. 모든 리전과 계정에서 이러한 값을 모니터링 및 관리하여 애플리케이션이 불리하거나 계획되지 않은 이벤트 발생 시 복원력을 개선할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

Service Quotas은 한 곳에서 250개가 넘는 AWS 서비스에 대한 할당량을 관리할 수 있도록 도와주는 AWS 서비스입니다. 할당량 값을 조회하는 것에 더해 Service Quotas 콘솔 또는 AWS SDK를 사용하여 할당량 증가를 요청하고 추적할 수 있습니다. AWS Trusted Advisor에서는 일부 서비스의 특정 측면에 대한 사용량 및 할당량을 표시하는 서비스 할당량 확인 기능을 제공합니다. 서비스별 기본 서비스 할당량은 해당하는 서비스의 AWS 설명서에도 문서화되어 있습니다. 예를 들어 Amazon VPC 할당량을 참조하세요.

일부 서비스 한도(예: 조절된 API에 대한 속도 제한)은 Amazon API Gateway 자체에서 사용량 계획을 구성하는 방법으로 설정됩니다. 해당하는 서비스에서 구성으로 설정되는 일부 제한으로는 프로비저닝된 IOPS, 할당된 Amazon RDS 스토리지 및 Amazon EBS 볼륨 할당이 있습니다. Amazon Elastic Compute Cloud에는 인스턴스, Amazon Elastic Block Store 및 탄력적 IP 주소 제한을 관리하는 데 도움이 되는 자체 서비스 한도 대시보드가 있습니다. 서비스 할당량이 애플리케이션 성능에 영향을 미치고 요구 사항에 맞춰 조정할 수 없는 사용 사례가 있는 경우 지원에 문의하여 완화 방법이 있는지 확인하세요.

서비스 할당량은 리전에 따라 다를 수 있으며 특성상 글로벌로 적용될 수도 있습니다. 할당량에 도달한 AWS 서비스를 사용하면 정상적인 사용 시에도 예상대로 작동하지 않을 수 있으며 서비스 중단 또는 성능 저하를 일으킬 수 있습니다. 예를 들어, 서비스 할당량은 리전에서 사용되는 DL Amazon EC2 인스턴스의 수를 제한합니다. Auto Scaling 그룹(ASG)을 사용하는 트래픽 규모 조정 이벤트 중에 이 한도에 도달할 수 있습니다.

각 계정에 대한 서비스 할당량은 정기적으로 사용에 대해 평가해 해당 계정에 대해 적절한 서비스 한도를 확인해야 합니다. 서비스 할당량은 필요한 것보다 더 많은 리소스를 실수로 프로비저닝하지 않기 위한 운영 가드레일로 존재합니다. 또한 서비스의 남용을 방지하기 위해 API 작업에 대한 요청 속도를 제한하기도 합니다.

서비스 제약 조건은 서비스 할당량과 다릅니다. 서비스 제약 조건은 서비스 유형에 따라 정의된 특정 리소스 한도를 나타냅니다. 여기에는 스토리지 용량(예: gp2에는 1GB~16TB의 크기 제한이 있음) 또는 디스크 처리량이 있을 수 있습니다. 한도에 도달할 수 있는 사용량에 대해서는 리소스 유형의 제약 조건을 엔지니어링하고 지속적으로 평가해야 합니다. 예기치 않게 제약 조건에 도달한 경우 계정의 애플리케이션 또는 서비스가 성능이 저하되거나 중단될 수 있습니다.

서비스 할당량이 애플리케이션 성능에 영향을 미치는데 요구 사항에 맞춰 조정할 수 없는 사용 사례가 있는 경우 지원에 문의하여 완화 방법이 있는지 확인하세요. 고정 할당량 조정에 대한 자세한 내용은 REL01-BP03 아키텍처를 통해 고정된 서비스 할당량 및 제약 조건 수용 섹션을 참조하세요.

Service Quotas 모니터링 및 관리를 지원하기 위한 여러 가지 AWS 서비스 및 도구가 있습니다. 할당량 수준을 자동으로 또는 수동으로 확인하려면 이러한 서비스 및 도구를 활용해야 합니다.

- AWS Trusted Advisor는 일부 서비스의 특정 측면에 대한 사용량 및 할당량을 표시하는 서비스 할당량 확인 기능을 제공합니다. 따라서 거의 할당량에 도달한 서비스를 식별하는 데 도움이 됩니다.
- AWS Management Console에서는 서비스 할당량 값을 표시하고, 새로운 할당량을 관리 및 요청하며, 할당량 요청 상태를 모니터링하고, 할당량 이력을 표시하는 방법을 제공합니다.
- AWS CLI 및 CDK에서는 서비스 할당량 수준과 사용을 자동으로 관리 및 모니터링하기 위한 프로그래밍 방식을 제공합니다.

구현 단계

Service Quotas의 경우:

- AWS Service Quotas를 검토합니다.
- 기존 서비스 할당량을 파악하려면 사용되는 서비스(예: IAM Access Analyzer)를 확인합니다. 서비스 할당량으로 제어되는 AWS 서비스는 약 250개가 있습니다. 그런 다음, 각 계정 및 리전 내에서 사용 가능한 특정 서비스 할당량 이름을 확인합니다. 리전당 약 3,000개의 서비스 할당량 이름이 있습니다.
- AWS 계정에서 사용되는 AWS 리소스를 모두 찾기 위해 AWS Config를 사용하여 할당량 분석을 보강합니다.
- AWS CloudFormation 데이터를 사용하여 사용된 AWS 리소스를 확인하세요. AWS Management Console 또는 `list-stack-resources`AWS CLI 명령으로 생성된 리소스를 살펴보세요. 템플릿 자체에 배포하도록 구성된 리소스를 볼 수도 있습니다.
- 배포 코드를 확인하여 워크로드에 필요한 모든 서비스를 결정합니다.
- 적용되는 서비스 할당량을 확인합니다. Trusted Advisor 및 Service Quotas에서 프로그래밍 방식으로 액세스되는 정보를 사용합니다.
- 서비스 할당량이 한계에 근접하거나 도달하면 알리도록 자동화된 모니터링 방법을 설정합니다(REL01-BP02 계정 및 리전 전체에서 서비스 할당량 관리 및 REL01-BP04 할당량 모니터링 및 관리 참조).
- 동일한 계정 내에서 서비스 할당량이 한 리전에서는 변경되었지만 다른 리전에서는 변경되지 않은 경우를 확인하도록 자동화된 프로그래밍 방식을 설정합니다(REL01-BP02 계정 및 리전 전체에서 서비스 할당량 관리 및 REL01-BP04 할당량 모니터링 및 관리 참조).
- 할당량 또는 서비스 제약 조건 오류가 있는지 확인하도록 애플리케이션 로그 및 지표 검사를 자동화합니다. 이러한 오류가 있는 경우 모니터링 시스템에 알림을 보냅니다.
- 특정 서비스에 더 큰 할당량이 필요하다는 사실이 확인되면 필요한 할당량 변경을 계산하는 엔지니어링 절차를 수립합니다(REL01-BP05 할당량 관리 자동화 참조).

- 서비스 할당량 변경을 요청하기 위한 프로비저닝 및 승인 워크플로를 생성합니다. 여기에는 요청 거부 또는 부분 승인 시 예외 워크플로를 포함해야 합니다.
- 프로덕션 또는 로드된 환경으로 롤아웃하기 전에 새로운 AWS 서비스를 프로비저닝하고 사용하기에 앞서 서비스 할당량을 검토하는 엔지니어링 방법을 생성합니다(예: 로드 테스트 계정).

서비스 제약 조건의 경우:

- 리소스 제약 조건에 근접한 리소스에 대해 경고하는 모니터링 및 지표 방법을 설정합니다. CloudWatch를 지표 또는 로그 모니터링에 적절하게 활용합니다.
- 애플리케이션 또는 시스템에 의미 있는 제약 조건이 있는 각 리소스에 대해 알림 임곗값을 설정합니다.
- 제약 조건에 거의 도달하면 리소스 유형을 변경하는 워크플로 및 인프라 관리 절차를 생성합니다. 이 워크플로에는 새 유형이 새로운 제약 조건이 있는 올바른 리소스 유형인지 확인하기 위한 모범 사례로 로드 테스트를 포함해야 합니다.
- 기존 절차 및 프로세스를 사용하여 식별된 리소스를 권장되는 새 리소스 유형으로 마이그레이션합니다.

## 리소스

관련 모범 사례:

- REL01-BP02 계정 및 리전 전체에서 서비스 할당량 관리
- REL01-BP03 아키텍처를 통해 고정된 서비스 할당량 및 제약 조건 수용
- REL01-BP04 할당량 모니터링 및 관리
- REL01-BP05 할당량 관리 자동화
- REL01-BP06 현재의 할당량과 최대 사용량 간에 장애 조치를 수용할 만큼 여유가 충분히 있는지 확인
- REL03-BP01 워크로드를 세그먼트화하는 방법 선택
- REL10-BP01 워크로드를 여러 위치에 배포
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지
- REL11-BP03 모든 계층에서 복구 자동화
- REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트

관련 문서:

- AWS Well-Architected Framework 신뢰성 원칙: 가용성
- AWS Service Quotas(이전의 서비스 한도)
- AWS Trusted Advisor Best Practice Checks (see the Service Limits section)
- AWS Limit Monitor on AWS 질문
- Amazon EC2 서비스 한도
- What is Service Quotas?
- How to Request Quota Increase
- Service endpoints and quotas
- Service Quotas 사용 설명서
- Quota Monitor for AWS
- AWS Fault Isolation Boundaries
- Availability with redundancy
- AWS for Data
- 지속적 통합이란 무엇입니까?
- 지속적 전달이란 무엇입니까?
- APN 파트너: 구성 관리를 지원할 수 있는 파트너
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS
- 워크로드의 API 스로틀링 관리 및 모니터링
- View AWS Trusted Advisor recommendations at scale with AWS Organizations
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower

관련 비디오:

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas
- AWS IAM Quotas Demo

관련 도구:

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail

- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

## REL01-BP02 계정 및 리전 전체에서 서비스 할당량 관리

여러 계정 또는 리전을 사용하는 경우 프로덕션 워크로드가 실행되는 모든 환경에서 적절한 할당량을 요청해야 합니다.

원하는 성과: 서비스 및 애플리케이션은 여러 계정 또는 리전에 적용되는 구성 또는 영역, 리전 또는 계정 장애 조치를 사용하는 탄력적인 설계의 구성으로 인한 서비스 할당량 소진의 영향을 받지 않아야 합니다.

일반적인 안티 패턴:

- 다른 격리 영역에서 용량을 유지하는 메커니즘 없이 한 격리 리전의 리소스 사용량을 확장하도록 허용합니다.
- 격리 리전에서 모든 할당량을 독립적으로 수동으로 설정합니다.
- 기본 리전이 아닌 리전에서 성능이 저하되는 동안 향후 필요한 할당량에 복원력 아키텍처(액티브 또는 패시브)의 영향을 고려하지 않습니다.
- 할당량을 정기적으로 평가하지 않고 워크로드가 실행되는 모든 리전 및 계정에서 필요한 변경을 수행하지 않습니다.
- 여러 리전 및 계정 간에 증가를 요청하는 데 할당량 요청 템플릿을 사용하지 않습니다.
- 할당량 증가가 컴퓨팅 예약 요청과 같이 비용에 영향을 미친다고 잘못 생각하여 서비스 할당량을 업데이트하지 않습니다.

이 모범 사례 확립의 이점: 리전별 서비스를 사용할 수 없는 경우 보조 리전 또는 계정에서 현재 로드를 처리할 수 있는지 확인합니다. 이는 리전 손실 중 발생하는 오류 수 또는 성능 저하 수준을 줄이는 데 도움이 됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

서비스 할당량은 계정별로 추적됩니다. 다른 언급이 없는 한, 각 할당량은 AWS 리전별로 다릅니다. 프로덕션 환경에 더해 적용 가능한 모든 비프로덕션 환경에서도 할당량을 관리하여 테스트 및 개발에 방해가 되지 않도록 합니다. 높은 수준의 복원력을 유지하려면 지속적으로 서비스 할당량을 (자동 또는 수동으로) 평가해야 합니다.

액티브/액티브, 액티브/패시브 – 핫, 액티브/패시브-콜드 및 액티브/패시브-파일럿 라이트 접근 방식을 사용하는 설계의 구현으로 인해 여러 리전에 걸쳐 워크로드가 증가하는 경우 모든 리전 및 계정 할당량 수준을 파악해야 합니다. 서비스 할당량이 올바르게 설정되어 있더라도 과거 트래픽 패턴이 항상 좋은 지표는 아닙니다.

서비스 할당량 이름 제한이 모든 리전에 대해 항상 같은 것도 아닙니다. 한 리전에서 이 값은 5일 수 있으며 다른 리전에서는 10일 수 있습니다. 로드 발생 시 일정한 복원력을 제공하려면 이러한 할당량 관리는 동일한 서비스, 계정, 리전을 모두 포함해야 합니다.

여러 리전(액티브 리전 또는 패시브 리전) 간에 모든 서비스 할당량 차이를 조정하고 이러한 차이를 지속적으로 조정하기 위한 프로세스를 생성합니다. 패시브 리전 장애 조치의 테스트 계획은 피크 액티브 용량으로 조정되는 경우가 거의 없습니다. 즉, 게임 데이 또는 탁상 훈련(TTX) 방식은 리전 간 서비스 할당량의 차이를 찾지 못할 수 있고 올바른 한도를 유지하지 못할 수 있습니다.

서비스 할당량 드리프트는 지정된 특정 할당량에 대한 서비스 할당량 제한이 모든 리전이 아니라 한 리전에서 변경되는 조건으로, 추적 및 평가해야 하는 매우 중요한 조건입니다. 트래픽이 있는 리전 또는 트래픽이 발생할 수 있는 리전에서는 할당량 변경을 고려해야 합니다.

- 서비스 요구 사항, 지연 시간, 규정, 재해 복구(DR) 요구 사항을 기준으로 관련 계정 및 리전을 선택합니다.
- 모든 관련 계정, 리전 및 가용 영역의 서비스 할당량을 확인합니다. 한도는 계정 및 리전별로 관리됩니다. 이러한 값은 차이가 있는지 비교해야 합니다.

### 구현 단계

- 사용 위험 수준을 벗어나 위반될 수 있는 Service Quotas 값을 검토합니다. AWS Trusted Advisor에서는 80% 및 90% 임곗값 위반에 대한 알림을 제공합니다.
- (액티브/패시브 설계인 경우) 모든 패시브 리전에서 서비스 할당량에 대한 값을 검토합니다. 기본 리전에서 장애 발생 시 보조 리전에서 로드가 성공적으로 실행되는지 확인합니다.

- 동일한 계정 내 리전 간에 서비스 할당량 드리프트가 발생했는지 여부 평가를 자동화하고 한도 변경에 적절하게 대응합니다.
- 고객 조직 단위(OU)가 지원되는 방식으로 구성되어 있으면 여러 리전 및 계정에 적용해야 하는 모든 할당량의 변화를 반영하도록 서비스 할당량 템플릿을 업데이트해야 합니다.
  - 템플릿을 생성하고 할당량 변경에 리전을 연결합니다.
  - 필요한 모든 변경(리전, 한도 및 계정)에 대한 기존 서비스 할당량 템플릿을 모두 검토합니다.

## 리소스

관련 모범 사례:

- REL01-BP01 서비스 할당량 및 제약 조건 인식
- REL01-BP03 아키텍처를 통해 고정된 서비스 할당량 및 제약 조건 수용
- REL01-BP04 할당량 모니터링 및 관리
- REL01-BP05 할당량 관리 자동화
- REL01-BP06 현재의 할당량과 최대 사용량 간에 장애 조치를 수용할 만큼 여유가 충분히 있는지 확인
- REL03-BP01 워크로드를 세그먼트화하는 방법 선택
- REL10-BP01 워크로드를 여러 위치에 배포
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지
- REL11-BP03 모든 계층에서 복구 자동화
- REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트

관련 문서:

- AWS Well-Architected Framework 신뢰성 원칙: 가용성
- AWS Service Quotas(이전의 서비스 한도)
- AWS Trusted Advisor Best Practice Checks (see the Service Limits section)
- AWS Limit Monitor on AWS 질문
- Amazon EC2 서비스 한도
- What is Service Quotas?
- How to Request Quota Increase
- Service endpoints and quotas

- Service Quotas 사용 설명서
- Quota Monitor for AWS
- AWS Fault Isolation Boundaries
- Availability with redundancy
- AWS for Data
- 지속적 통합이란 무엇입니까?
- 지속적 전달이란 무엇입니까?
- APN 파트너: 구성 관리를 지원할 수 있는 파트너
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS
- Managing and monitoring API throttling in your workloads
- View AWS Trusted Advisor recommendations at scale with AWS Organizations
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower

관련 비디오:

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas
- AWS IAM Quotas Demo

관련 서비스:

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

# REL01-BP03 아키텍처를 통해 고정된 서비스 할당량 및 제약 조건 수용

변경할 수 없는 서비스 할당량, 서비스 제약 조건 및 물리적 리소스 한도를 알고 있어야 합니다. 이러한 한도가 신뢰성에 영향을 미치지 않도록 애플리케이션 및 서비스의 아키텍처를 설계합니다.

네트워크 대역폭, 서버리스 기능 간접 호출 페이로드 크기, API 게이트웨이의 제한 버스트 속도 및 데이터베이스에 대한 동시 사용자 연결 등이 여기에 포함됩니다.

원하는 성과: 애플리케이션 또는 서비스가 정상 조건 및 트래픽이 많은 조건에서 예상대로 수행됩니다. 해당 리소스의 고정 제약 조건 또는 서비스 할당량의 한계 내에서 작동하도록 설계되었습니다.

일반적인 안티 패턴:

- 규모를 조정할 때 해당 설계에서 장애를 유발하는 설계 제약 조건이 있다는 사실을 인지하지 못한 채 하나의 서비스 리소스를 사용하는 설계 방식을 선택합니다.
- 비현실적이며 테스트 중에 고정된 서비스 할당량에 도달하는 벤치마킹을 수행합니다. 예를 들어, 버스트 한도에서 테스트를 실행하면서, 오랜 시간 실행합니다.
- 고정된 서비스 할당량을 초과하는 경우 조정할 수 없거나 수정할 수 없는 설계를 선택합니다. 예를 들어, SQS 페이로드 크기는 256KB입니다.
- 높은 트래픽 이벤트 동안 위험에 처할 수 있는 서비스 할당량의 임곗값을 모니터링하고 경고하도록 관찰성이 설계 및 구현되지 않았습니다.

이 모범 사례 확립의 이점: 애플리케이션이 중단이나 성능 저하 없이 예상되는 모든 서비스 부하 수준에서 실행되는지 확인합니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 지침

더 큰 용량 단위로 대체되는 소프트 서비스 할당량 또는 리소스와 달리 AWS 서비스의 고정 할당량은 변경할 수 없습니다. 따라서 이러한 모든 유형의 AWS 서비스는 애플리케이션 설계에 사용될 때 잠재적인 하드 용량 한도에 대해 평가되어야 합니다.

하드 한도는 Service Quotas 콘솔에 표시됩니다. 열에 `ADJUSTABLE = No`가 표시된 경우 서비스에 하드 한도가 있는 것입니다. 하드 한도는 일부 리소스 구성 페이지에도 표시됩니다. 예를 들어 Lambda에는 조정할 수 없는 특정 하드 한도가 있습니다.

예를 들어 Python 애플리케이션을 Lambda 함수에서 실행하도록 설계할 때 Lambda가 15분 이상 실행될 가능성이 있는지 확인하기 위해 애플리케이션을 평가해야 합니다. 코드가 이 서비스 할당량 한도보

다 더 오래 실행될 수 있는 경우 대체 기술 또는 설계를 고려해야 합니다. 프로덕션 배포 후 이 한도에 도달하면 애플리케이션은 문제를 해결할 수 있을 때까지 성능 저하 및 중단이 발생합니다. 소프트 할당량과 달리 긴급 심각도 1 이벤트에서도 이러한 한도를 변경할 수 있는 방법이 없습니다.

애플리케이션이 테스트 환경에 배포되면 하드 한도에 도달할 수 있는지 확인하기 위한 전략을 사용해야 합니다. 스트레스 테스트, 부하 테스트 및 카오스 테스트는 도입 테스트 계획의 일부여야 합니다.

구현 단계

- 애플리케이션 설계 단계에서 사용할 수 있는 AWS 서비스의 전체 목록을 검토합니다.
- 이러한 모든 서비스에 대한 소프트 할당량 한도 및 하드 할당량 한도를 검토합니다. 모든 한도가 Service Quotas 콘솔에 표시되는 것은 아닙니다. 일부 서비스에서는 대체 위치에서 이러한 제한을 설명합니다.
- 애플리케이션을 설계할 때 비즈니스 결과, 사용 사례, 종속 시스템, 가용성 목표 및 재해 복구 개체와 같은 워크로드의 비즈니스 및 기술 동인을 검토합니다. 비즈니스 및 기술 동인이 워크로드에 적합한 분산 시스템을 식별하는 프로세스를 안내하도록 합니다.
- 리전 및 계정 전체에서 서비스 부하를 분석합니다. 서비스의 많은 하드 한도가 리전을 기반으로 합니다. 그러나 일부 한도는 계정을 기반으로 합니다.
- 영역 장애 및 리전 장애 시 리소스 사용량에 대한 복원력 아키텍처를 분석합니다. 액티브/액티브, 액티브/패시브 - 핫, 액티브/패시브 - 콜드 및 액티브/패시브 - 파일럿 라이트 접근 방식을 사용하는 다중 리전 설계의 진행에서 이러한 실패 사례는 더 높은 사용량을 야기할 것입니다. 이는 하드 한도에 도달할 수 있는 잠재적 사용 사례를 생성합니다.

## 리소스

관련 모범 사례:

- REL01-BP01 서비스 할당량 및 제약 조건 인식
- REL01-BP02 계정 및 리전 전체에서 서비스 할당량 관리
- REL01-BP04 할당량 모니터링 및 관리
- REL01-BP05 할당량 관리 자동화
- REL01-BP06 현재의 할당량과 최대 사용량 간에 장애 조치를 수용할 만큼 여유가 충분히 있는지 확인
- REL03-BP01 워크로드를 세그먼트화하는 방법 선택
- REL10-BP01 워크로드를 여러 위치에 배포
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지

- REL11-BP03 모든 계층에서 복구 자동화
- REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트

관련 문서:

- AWS Well-Architected Framework 신뢰성 원칙: 가용성
- AWS Service Quotas(이전의 서비스 한도)
- AWS Trusted Advisor Best Practice Checks (see the Service Limits section)
- AWS Limit Monitor on AWS 질문
- Amazon EC2 서비스 한도
- What is Service Quotas?
- How to Request Quota Increase
- Service endpoints and quotas
- Service Quotas 사용 설명서
- Quota Monitor for AWS
- AWS Fault Isolation Boundaries
- Availability with redundancy
- AWS for Data
- 지속적 통합이란 무엇입니까?
- 지속적 전달이란 무엇입니까?
- APN 파트너: 구성 관리를 지원할 수 있는 파트너
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS
- Managing and monitoring API throttling in your workloads
- View AWS Trusted Advisor recommendations at scale with AWS Organizations
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower
- Actions, resources, and condition keys for Service Quotas

관련 비디오:

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas
- AWS IAM Quotas Demo

- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small

관련 도구:

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

## REL01-BP04 할당량 모니터링 및 관리

잠재적 사용량을 평가하고 할당량을 적절히 늘려 사용량 증가를 계획합니다.

원하는 성과: 관리 및 모니터링하는 액티브 자동화된 시스템을 배포합니다. 이러한 운영 솔루션은 할당량 사용량 임곗값에 거의 도달하고 있는지 확인합니다. 이러한 문제는 요청된 할당량 변경에 의해 사전에 해결됩니다.

일반적인 안티 패턴:

- 서비스 할당량 임곗값을 확인하도록 모니터링을 구성하지 않습니다.
- 해당 값을 변경할 수 없는 경우에도 하드 한도에 대한 모니터링을 구성하지 않습니다.
- 소프트 할당량 변경을 요청하고 확보하는 데 필요한 시간이 즉각적이거나 짧은 기간이라고 가정합니다.
- 서비스 할당량에 근접할 경우 알리는 경보를 구성하지만, 알림에 응답하는 프로세스를 갖추지 않습니다.
- AWS Service Quotas에서 지원하는 서비스에 대해서만 경보를 구성하고 다른 AWS 서비스는 모니터링하지 않습니다.

- 액티브/액티브, 액티브/패시브 - 핫, 액티브/패시브 - 콜드 및 액티브/패시브 - 파일럿 라이트 접근 방식과 같은 여러 리전 복원력 설계에 대한 할당량 관리를 고려하지 않습니다.
- 리전 간 할당량 차이를 평가하지 않습니다.
- 특정 할당량 증가 요청에 대해 모든 리전의 요구 사항을 평가하지 않습니다.
- 다중 리전 할당량 관리에 템플릿을 활용하지 않습니다.

이 모범 사례 확립의 이점: AWS 서비스 할당량을 자동으로 추적하고 이러한 할당량을 기준으로 사용량을 모니터링하면 할당량 한도에 근접할 경우 이를 알 수 있습니다. 이 모니터링 데이터를 사용하여 할당량 소진으로 인한 성능 저하를 제한할 수도 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

지원되는 서비스의 경우 경고 또는 알람을 평가하고 보낼 수 있는 다양한 서비스를 구성하여 할당량을 모니터링할 수 있습니다. 사용량을 모니터링하는 데 도움이 될 수 있으며 할당량에 근접하면 알림을 받을 수 있습니다. 이러한 경보는 AWS Config Lambda 함수, Amazon CloudWatch 또는 AWS Trusted Advisor에서 간접 호출할 수 있습니다. CloudWatch Logs의 지표 필터를 사용하여 로그의 패턴을 검색하고 추출하여 사용량이 할당량 임곗값에 근접하는지 여부를 확인할 수도 있습니다.

구현 단계

모니터링의 경우:

- 현재 리소스 사용 내역(버킷, 인스턴스)을 파악합니다. 현재 리소스 사용 내역을 모두 보려면 Amazon EC2 `DescribeInstances` API와 같은 서비스 API를 사용합니다.
- 다음을 사용하여 서비스에 필수적이며 적용 가능한 현재 할당량을 캡처합니다.
  - AWS Service Quotas
  - AWS Trusted Advisor
  - AWS 설명서
  - AWS 서비스별 페이지
  - AWS Command Line Interface (AWS CLI)
  - AWS 클라우드 개발 키트 (AWS CDK)
- AWS Service Quotas를 사용합니다. 한 곳에서 250개가 넘는 AWS 서비스에 대한 할당량을 관리할 수 있도록 도와주는 AWS 서비스입니다.
- Trusted Advisor 서비스 한도를 사용하여 다양한 임곗값에서 현재 서비스 한도를 모니터링합니다.

- 서비스 할당량 내역(콘솔 또는 AWS CLI)을 사용하여 리전별 증가를 확인합니다.
- 필요한 경우 각 리전 및 각 계정의 서비스 할당량 변경 사항을 비교하여 동등성을 생성합니다.

관리의 경우:

- 자동: AWS Config 사용자 지정 규칙을 설정하여 리전 간 서비스 할당량을 스캔하고 차이점을 비교합니다.
- 자동: 예약된 Lambda 함수를 설정하여 리전 간 서비스 할당량을 스캔하고 차이점을 비교합니다.
- 수동: AWS CLI, API 또는 AWS 콘솔을 통해 서비스 할당량을 스캔하여 리전 간 서비스 할당량을 스캔하고 차이점을 비교합니다. 차이점을 보고합니다.
- 리전 간에 할당량 차이가 확인되면 필요한 경우 할당량 변경을 요청합니다.
- 모든 요청 결과를 검토합니다.

## 리소스

관련 모범 사례:

- REL01-BP01 서비스 할당량 및 제약 조건 인식
- REL01-BP02 계정 및 리전 전체에서 서비스 할당량 관리
- REL01-BP03 아키텍처를 통해 고정된 서비스 할당량 및 제약 조건 수용
- REL01-BP05 할당량 관리 자동화
- REL01-BP06 현재의 할당량과 최대 사용량 간에 장애 조치를 수용할 만큼 여유가 충분히 있는지 확인
- REL03-BP01 워크로드를 세그먼트화하는 방법 선택
- REL10-BP01 워크로드를 여러 위치에 배포
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지
- REL11-BP03 모든 계층에서 복구 자동화
- REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트

관련 문서:

- AWS Well-Architected Framework 신뢰성 원칙: 가용성
- AWS Service Quotas(이전의 서비스 한도)
- AWS Trusted Advisor Best Practice Checks (see the Service Limits section)

- AWS Limit Monitor on AWS 질문
- Amazon EC2 서비스 한도
- What is Service Quotas?
- How to Request Quota Increase
- Service endpoints and quotas
- Service Quotas 사용 설명서
- Quota Monitor for AWS
- AWS Fault Isolation Boundaries
- Availability with redundancy
- AWS for Data
- 지속적 통합이란 무엇입니까?
- 지속적 전달이란 무엇입니까?
- APN 파트너: 구성 관리를 지원할 수 있는 파트너
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS
- Managing and monitoring API throttling in your workloads
- View AWS Trusted Advisor recommendations at scale with AWS Organizations
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower
- Actions, resources, and condition keys for Service Quotas

관련 비디오:

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas
- AWS IAM Quotas Demo
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small

관련 도구:

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch

- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

## REL01-BP05 할당량 관리 자동화

AWS 서비스에서 한도라고도 하는 서비스 할당량은 AWS 계정 계정의 리소스에 대한 최댓값입니다. 각 AWS 서비스는 할당량 세트와 기본값을 정의합니다. 필요한 모든 리소스에 대한 액세스를 워크로드에 제공하려면 서비스 할당량 값을 늘려야 할 수 있습니다.

AWS 리소스의 워크로드 소비가 증가하면 워크로드 안정성이 위협받고 할당량이 초과되면 사용자 경험에 영향을 미칠 수 있습니다. 워크로드가 한도에 가까워지면 알림을 보내고 할당량 증가 요청을 자동으로 생성하는 것을 고려하는 도구를 구현합니다.

원하는 성과: 각 AWS 계정 및 리전에서 실행되는 워크로드에 맞게 할당량이 적절하게 구성되어 있습니다.

일반적인 안티 패턴:

- 워크로드 요구 사항을 충족하도록 할당량을 적절하게 고려하고 조정하지 못합니다.
- 스프레드시트와 같이 더 이상 효용이 없을 수 있는 방법을 사용하여 할당량 및 사용량을 추적합니다.
- 정기적인 일정에 따라서만 서비스 한도를 업데이트합니다.
- 조직은 기존 할당량을 검토하고 필요한 경우 서비스 할당량 증가를 요청하는 운영 프로세스가 부족합니다.

이 모범 사례 확립의 이점:

- 향상된 워크로드 복원력: AWS 리소스 할당량을 초과하여 발생하는 오류를 방지합니다.
- 간소화된 재해 복구: 다른 AWS 리전에서 DR을 설정하는 동안 기본 리전에 구축된 자동 할당량 관리 메커니즘을 재사용할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

AWS Service Quotas 콘솔, AWS Command Line Interface(AWS CLI) 및 AWS SDK 등의 메커니즘을 사용하여 현재 할당량을 보고 진행 중인 할당량 소비를 추적합니다. 구성 관리 데이터베이스(CMDB) 및 IT 서비스 관리(ITSM) 시스템을 AWS Service Quota API과 통합할 수도 있습니다.

할당량 사용량이 정의된 임계값에 도달하면 자동 알림을 생성하고 알림을 받을 때 할당량 증가 요청을 제출하는 프로세스를 정의합니다. 기본 워크로드가 비즈니스에 중요한 경우 할당량 증가 요청을 자동화할 수 있지만 성장 피드백 루프와 같은 런어웨이 작업의 위험을 방지하기 위해 자동화를 신중하게 테스트할 수 있습니다.

비교적 작은 할당량 증가는 종종 자동으로 승인됩니다. 더 큰 할당량 요청은 AWS 지원에서 수동으로 처리해야 할 수 있으며 검토 및 처리하는 데 추가 시간이 걸릴 수 있습니다. 여러 요청 또는 대규모 증가 요청을 처리하는 데 추가 시간이 드는 것을 감안합니다.

구현 단계

- 서비스 할당량에 대한 자동 모니터링을 구현하고 워크로드의 리소스 사용률이 할당량 한도에 도달하면 알림을 발행합니다. 예를 들어 AWS용 Quota Monitor는 서비스 할당량에 대한 자동 모니터링을 제공할 수 있습니다. 이 도구는 AWS Organizations과 통합되고 Cloudformation StackSets를 사용하여 배포하므로 새 계정이 생성되면 자동으로 모니터링됩니다.
- Service Quotas 요청 템플릿 또는 AWS Control Tower와 같은 기능을 사용하여 새 계정에 대한 Service Quotas 설정을 간소화합니다.
- 모든 AWS 계정 및 리전에 대한 현재 서비스 할당량 사용 대시보드를 구축하고 할당량 초과를 방지하기 위해 필요에 따라 참조합니다. Cloud Intelligence Dashboards의 일부인 Trusted Advisor Organizational(TAO) Dashboard를 사용하면 이러한 대시보드를 빠르게 시작할 수 있습니다.
- 서비스 한도 증가 요청을 추적합니다. Consolidated Insights from Multiple Accounts(CIMA)는 모든 요청에 대한 조직 수준 보기를 제공할 수 있습니다.
- 비프로덕션 계정에서 할당량 임계값을 낮게 설정하여 알림 생성 및 할당량 증가 요청 자동화를 테스트합니다. 프로덕션 계정에서 이러한 테스트를 수행하지 마세요.

## 리소스

관련 모범 사례:

- OPS10-BP07 이벤트 대응 자동화

관련 문서:

- APN 파트너: 구성 관리를 지원할 수 있는 파트너
- AWS Marketplace: CMDB products that help track limits
- AWS Service Quotas(이전의 서비스 한도)
- AWS Trusted Advisor Best Practice Checks (see the Service Limits section)
- AWS의 Quota Monitor 솔루션 - AWS 솔루션
- What is Service Quotas?
- What is Service Quotas request templates?

관련 비디오:

- AWS Live re:Inforce 2019 - Service Quotas
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower

관련 도구:

- Quota Monitor for AWS

## REL01-BP06 현재의 할당량과 최대 사용량 간에 장애 조치를 수용할 만큼 여유가 충분히 있는지 확인

이 문서에서는 리소스 할당량과 사용량 사이에서 일정 간격을 유지하는 방법과 리소스 할당량이 조직에 어떤 이점을 줄 수 있는지 설명합니다. 리소스 사용을 완료한 후에도 사용량 할당량에서는 해당 리소스를 계속 고려할 수 있습니다. 이로 인해 리소스에서 장애가 실패하거나 리소스에 액세스하지 못할 수 있습니다. 액세스할 수 없는 리소스와 대체 리소스가 중복되는 부분이 할당량에 반영되는지 확인하여 리소스 장애를 방지합니다. 이 차이를 계산할 때 네트워크 장애, 가용 영역 장애 또는 리전 장애와 같은 사용 사례를 고려합니다.

원하는 성과: 리소스 또는 리소스 액세스 가능성에서 발생하는 작거나 큰 장애는 현재 서비스 임곗값 내에서 처리될 수 있습니다. 영역 장애, 네트워크 장애 또는 리전 장애도 리소스 계획에서 고려되었습니다.

일반적인 안티 패턴:

- 장애 조치 시나리오를 고려하지 않고 현재의 수요를 기준으로 서비스 할당량을 설정합니다.
- 서비스의 최대 할당량을 계산할 때 정적 안정성 원칙을 고려하지 않습니다.

- 각 리전에 필요한 총 할당량을 계산할 때 액세스할 수 없는 리소스의 가능성을 고려하지 않습니다.
- 일부 서비스에 대한 AWS 서비스 장애 격리 경계 및 잠재적인 비정상적인 사용 패턴을 고려하지 않습니다.

이 모범 사례 확립의 이점: 서비스 중단 이벤트가 애플리케이션 가용성에 영향을 미치는 경우 클라우드를 통해 이러한 이벤트를 복구하는 전략을 구현합니다. 추가 리소스를 만들어 서비스 한도를 소진하지 않으면서 장애 조치 조건을 수용할 수 있도록 액세스할 수 없는 리소스를 대체하는 전략이 한 가지 예입니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

할당량 한도를 평가할 때 일부 성능 저하로 인해 발생할 수 있는 장애 조치 사례를 고려합니다. 다음 장애 조치 사례를 고려합니다.

- 장애가 발생했거나 액세스할 수 없는 VPC.
- 액세스할 수 없는 서브넷.
- 리소스 액세스 가능성에 영향을 미치는 성능이 저하된 가용 영역.
- 네트워킹 경로 또는 수신 및 송신 지점이 차단되거나 변경됩니다.
- 리소스 액세스 가능성에 영향을 미치는 성능이 저하된 리전.
- 리전 또는 가용 영역에서 발생한 장애로 영향을 받는 리소스의 하위 세트.

장애 조치 결정은 비즈니스에 미치는 영향이 크게 다를 수 있으므로 상황마다 다릅니다. 애플리케이션 또는 서비스 장애 조치를 결정하기 전에 장애 조치 위치에서 리소스의 용량 계획 및 해당 리소스의 할당량을 해결합니다.

각 서비스의 할당량을 검토할 때 정상적인 활동 피크보다 높은 상황을 고려합니다. 이러한 피크는 네트워킹 또는 권한으로 인해 액세스할 수 없지만 여전히 활성 상태인 리소스와 관련이 있을 수 있습니다. 종료되지 않은 활성 리소스는 여전히 서비스 할당량 한도에 포함됩니다.

구현 단계

- 장애 조치와 액세스 가능성 손실을 수용할 수 있도록 서비스 할당량과 최대 사용량 사이에서 일정 간격을 유지합니다.
- 서비스 할당량을 결정합니다. 일반적인 배포 패턴, 가용성 요구 사항, 사용량 증가를 고려합니다.
- 필요한 경우 할당량 증가를 요청합니다. 할당량 증가 요청에 대한 대기 시간을 예상합니다.

- 신뢰성 요구 사항(9의 개수로도 표현)을 확인합니다.
- 구성 요소, 가용 영역 또는 리전의 손실과 같은 잠재적 장애 시나리오를 파악합니다.
- 배포 방법(예: canary, 블루/그린, 레드/블랙, 롤링)을 설정합니다.
- 현재 할당향 한도에 적절한 버퍼를 포함합니다. 예를 들어 버퍼는 15%일 수 있습니다.
- 적절한 경우 정적 안정성(영역 및 리전)에 대한 계산을 포함합니다.
- 사용량 증가 계획을 세우고 사용 추세를 모니터링합니다.
- 가장 중요한 워크로드에 대한 정적 안정성의 영향을 고려합니다. 모든 리전 및 가용 영역에서 정적으로 안정적인 시스템을 준수하는 리소스를 평가합니다.
- 온디맨드 용량 예약을 사용하여 장애 조치 전에 용량을 예약하는 것을 고려합니다. 이는 장애 조치 중에 올바른 양과 유형의 리소스를 확보하여 가장 중요한 비즈니스 일정에서 잠재적 위험을 줄일 수 있는 유용한 전략이 될 수 있습니다.

## 리소스

관련 모범 사례:

- REL01-BP01 서비스 할당량 및 제약 조건 인식
- REL01-BP02 계정 및 리전 전체에서 서비스 할당량 관리
- REL01-BP03 아키텍처를 통해 고정된 서비스 할당량 및 제약 조건 수용
- REL01-BP04 할당량 모니터링 및 관리
- REL01-BP05 할당량 관리 자동화
- REL03-BP01 워크로드를 세그먼트화하는 방법 선택
- REL10-BP01 워크로드를 여러 위치에 배포
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지
- REL11-BP03 모든 계층에서 복구 자동화
- REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트

관련 문서:

- AWS Well-Architected Framework 신뢰성 원칙: 가용성
- AWS Service Quotas(이전의 서비스 한도)
- AWS Trusted Advisor Best Practice Checks (see the Service Limits section)
- AWS Limit Monitor on AWS 질문

- Amazon EC2 서비스 한도
- What is Service Quotas?
- How to Request Quota Increase
- Service endpoints and quotas
- Service Quotas 사용 설명서
- Quota Monitor for AWS
- AWS Fault Isolation Boundaries
- Availability with redundancy
- AWS for Data
- 지속적 통합이란 무엇입니까?
- 지속적 전달이란 무엇입니까?
- APN 파트너: 구성 관리를 지원할 수 있는 파트너
- Managing the account lifecycle in account-per-tenant SaaS environments on AWS
- Managing and monitoring API throttling in your workloads
- View AWS Trusted Advisor recommendations at scale with AWS Organizations
- Automating Service Limit Increases and Enterprise Support with AWS Control Tower
- Actions, resources, and condition keys for Service Quotas

관련 비디오:

- AWS Live re:Inforce 2019 - Service Quotas
- View and Manage Quotas for AWS Services Using Service Quotas
- AWS IAM Quotas Demo
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small

관련 도구:

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge

- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

# 네트워크 토폴로지 계획

워크로드는 여러 환경에 존재하는 경우가 많습니다. 여기에는 여러 클라우드 환경(퍼블릭 액세스 가능 및 프라이빗)과 기존 데이터 센터 인프라가 포함됩니다. 따라서 시스템 내부 및 시스템 간 연결, 퍼블릭 IP 주소 관리, 프라이빗 IP 주소 관리 및 도메인 이름 확인과 같은 네트워크 고려 사항을 계획에 포함해야 합니다.

IP 주소 기반 네트워크를 사용하여 시스템을 설계할 때는 가능한 장애, 향후 성장 및 다른 시스템 및 네트워크와의 통합을 예상하여 네트워크 토폴로지 및 주소 지정을 계획해야 합니다.

Amazon Virtual Private Cloud(Amazon VPC)를 사용하면 AWS 클라우드에서 격리된 가상 네트워크를 프로비저닝하여 AWS 리소스를 시작할 수 있습니다.

모범 사례

- REL02-BP01 워크로드 퍼블릭 엔드포인트에 고가용성 네트워크 연결 사용
- REL02-BP02 클라우드와 온프레미스 환경의 프라이빗 네트워크 간에 중복 연결 프로비저닝
- REL02-BP03 확장 및 가용성을 위한 IP 서브넷 할당 계정 확인
- REL02-BP04 다대다 메시보다 허브 앤 스포크 토폴로지 선호
- REL02-BP05 연결된 모든 프라이빗 주소 공간에서 겹치지 않는 프라이빗 IP 주소 범위 적용

## REL02-BP01 워크로드 퍼블릭 엔드포인트에 고가용성 네트워크 연결 사용

워크로드의 퍼블릭 엔드포인트에 대한 고가용성 네트워크 연결을 구축하면 연결 손실로 인한 가동 중지 시간을 줄이고 워크로드의 가용성 및 SLA를 개선하는 데 도움이 될 수 있습니다. 이러한 고가용성을 달성하려면 고가용성 DNS, 콘텐츠 전송 네트워크(CDN), API Gateway, 로드 밸런싱 또는 역방향 프록시를 사용합니다.

원하는 성과: 퍼블릭 엔드포인트에 대한 고가용성 네트워크 연결을 계획, 구축 및 운영하는 것이 중요합니다. 연결 손실로 인해 워크로드에 연결할 수 없게 되면 워크로드가 실행 중이고 사용 가능한 경우

에도 고객은 시스템이 다운된 것으로 보게 됩니다. 워크로드 자체에 대한 복원력 있는 아키텍처와 함께 워크로드의 퍼블릭 엔드포인트를 위한 고가용성 및 복원력 있는 네트워크 연결을 결합하여 고객에게 가능한 최상의 가용성과 서비스 수준을 제공할 수 있습니다.

AWS Global Accelerator, Amazon CloudFront, Amazon API Gateway, AWS Lambda 함수 URL, AWS AppSync API, Elastic Load Balancing(ELB) 모두 고가용성 퍼블릭 엔드포인트를 제공합니다. Amazon Route 53은 퍼블릭 엔드포인트 주소를 확인할 수 있는지 검증하기 위해 도메인 이름 확인을 위한 고가용성 DNS 서비스를 제공합니다.

로드 밸런싱 및 프록싱을 위해 AWS Marketplace 소프트웨어 어플라이언스를 평가할 수도 있습니다.

일반적인 안티 패턴:

- 고가용성을 위한 DNS 및 네트워크 연결을 계획하지 않고 고가용성 워크로드를 설계합니다.
- 개별 인스턴스 또는 컨테이너에서 퍼블릭 인터넷 주소를 사용하고 DNS를 통해 이러한 주소에 대한 연결을 관리합니다.
- 서비스를 찾기 위해 도메인 이름 대신 IP 주소를 사용합니다.
- 퍼블릭 엔드포인트에 대한 연결이 끊어지는 시나리오를 테스트하지 않습니다.
- 네트워크 처리량 요구 사항 및 배포 패턴을 분석하지 않습니다.
- 워크로드의 퍼블릭 엔드포인트에 대한 인터넷 네트워크 연결이 중단될 수 있는 시나리오를 테스트하고 계획하지 않습니다.
- 콘텐츠 전송 네트워크를 사용하지 않고 대규모 지리적 영역에 콘텐츠(예: 웹 페이지, 정적 자산, 미디어 파일)를 제공합니다.
- 분산 서비스 거부(DDoS) 공격에 대한 계획이 없습니다. DDoS 공격은 합법적인 트래픽을 차단하고 사용자의 가용성을 낮출 위험이 있습니다.

이 모범 사례 확립의 이점: 가용성이 높고 복원력이 뛰어난 네트워크 연결을 설계하면 사용자가 워크로드에 액세스하고 사용할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

퍼블릭 엔드포인트에 대한 고가용성 네트워크 연결 구축의 핵심은 트래픽 라우팅입니다. 트래픽이 엔드포인트에 도달할 수 있는지 확인하려면 DNS가 도메인 이름을 해당 IP 주소로 확인할 수 있어야 합니다. Amazon Route 53과 같은 가용성과 확장성이 뛰어난 도메인 이름 시스템 (DNS) 을 사용하여 도메인의 DNS 레코드를 관리하세요. Amazon Route 53에서 제공하는 상태 확인을 사용할 수도 있습니다. 상태 확인은 애플리케이션이 도달 가능하고 사용 가능하며 작동하는지 확인하고, 웹 페이지 또는 특

정 URL 요청과 같은 사용자 행동을 모방하는 방식으로 설정할 수 있습니다. 장애가 발생하면 Amazon Route 53은 DNS 확인 요청에 응답하고 트래픽을 정상 상태인 엔드포인트로만 보냅니다. Amazon Route 53에서 제공하는 지리적 DNS 및 지연 시간 기반 라우팅 기능을 사용할 수도 있습니다.

워크로드 자체의 고가용성 여부를 확인하려면 Elastic Load Balancing(ELB)을 사용합니다. Amazon Route 53을 사용하여 트래픽을 대상 컴퓨팅 인스턴스로 분산하는 ELB로 트래픽 대상을 지정할 수 있습니다. 서버리스 솔루션을 위해 AWS Lambda와 함께 Amazon API Gateway를 사용할 수도 있습니다. 고객은 여러 AWS 리전에서 워크로드를 실행할 수도 있습니다. 멀티사이트 액티브/액티브 패턴을 사용하면 워크로드가 여러 리전의 트래픽을 처리할 수 있습니다. 다중 사이트 액티브/액티브 패턴을 사용하면 워크로드가 활성 리전의 트래픽을 지원합니다. 그동안 데이터는 보조 리전으로 복제되고 기본 리전에서 장애 발생 시 활성화됩니다. 그런 다음, Route 53 상태 확인을 통해 기본 리전의 모든 엔드포인트에서 보조 리전의 엔드포인트로의 DNS 장애 조치를 제어하여 사용자가 워크로드에 도달하고 사용할 수 있는지 확인할 수 있습니다.

Amazon CloudFront는 전 세계 엣지 로케이션 네트워크를 사용하여 요청을 처리함으로써 짧은 지연 시간과 높은 데이터 전송 속도로 콘텐츠를 배포하기 위한 간단한 API를 제공합니다. 콘텐츠 전송 네트워크(CDN)는 사용자와 가까운 위치에 있거나 캐시된 콘텐츠를 제공하여 고객에게 서비스를 제공합니다. 또한 콘텐츠에 대한 로드가 서버에서 CloudFront의 엣지 로케이션으로 이동되므로 애플리케이션의 가용성도 향상됩니다. 엣지 로케이션 및 리전 엣지 캐시는 콘텐츠의 캐시된 복사본을 사용자와 가까이에 유지하여 빠른 검색과 워크로드의 도달 가능성 및 가용성을 높입니다.

사용자가 지리적으로 분산된 워크로드의 경우 AWS Global Accelerator는 애플리케이션의 가용성과 성능을 개선하는 데 도움이 됩니다. AWS Global Accelerator는 하나 이상의 AWS 리전에서 호스팅되는 애플리케이션에 대한 고정 진입점 역할을 하는 애니캐스트 정적 IP 주소를 제공합니다. 이렇게 하면 트래픽이 가능한 한 사용자와 가까운 AWS 글로벌 네트워크로 유입되어 워크로드의 도달 가능성과 가용성이 향상됩니다. AWS Global Accelerator는 또한 TCP, HTTP 및 HTTPS 상태 확인을 사용하여 애플리케이션 엔드포인트의 상태를 모니터링합니다. 엔드포인트의 상태 또는 구성이 변경되면 사용자 트래픽이 정상 엔드포인트로 리디렉션되어 사용자에게 최상의 성능과 가용성을 제공합니다. 또한 AWS Global Accelerator에는 독립 네트워크 영역에서 서비스하는 두 개의 정적 IPv4 주소를 사용하여 애플리케이션의 가용성을 높이는 장애 격리 설계가 있습니다.

DDoS 공격으로부터 고객을 보호하기 위해 AWS에서는 AWS Shield Standard를 제공합니다. Shield Standard는 자동으로 활성화되며 SYN/UDP 플러드 및 반사 공격과 같은 일반적인 인프라(계층 3 및 4) 공격으로부터 보호하여 AWS에서 애플리케이션의 고가용성을 지원합니다. 더 정교하고 더 큰 규모의 공격(예: UDP 플러드), 상태 고갈 공격(예: TCP SYN 플러드)에 대한 추가 보호를 제공하고 Amazon Elastic Compute Cloud(Amazon EC2), Elastic Load Balancing(ELB), Amazon CloudFront, AWS Global Accelerator 및 Route 53에서 실행되는 애플리케이션을 보호하기 위해 AWS Shield Advanced 사용을 고려할 수 있습니다. HTTP POST 또는 GET 플러드와 같은 애플리케이션 계층 공격으로부

터 보호하려면 AWS WAF를 사용합니다. AWS WAF는 IP 주소, HTTP 헤더, HTTP 본문, URI 문자열, SQL 명령어 삽입 및 교차 사이트 스크립팅 조건을 사용하여 요청을 차단할지 또는 허용할지 결정할 수 있습니다.

구현 단계

1. 고가용성 DNS 설정: Amazon Route 53은 가용성과 확장성이 뛰어난 도메인 이름 시스템(DNS) 웹 서비스입니다. Route 53은 사용자 요청을 AWS 또는 온프레미스에서 실행되는 인터넷 애플리케이션에 연결합니다. 자세한 내용은 configuring Amazon Route 53 as your DNS service를 참조하세요.
2. 상태 확인 설정: Route 53을 사용할 때 정상 대상만 확인할 수 있는지 확인합니다. Route 53 상태 확인 생성 및 DNS 장애 조치 구성 작업부터 시작합니다. 상태 확인을 설정할 때 다음 측면을 고려해야 합니다.
   a. Amazon Route 53이 상태 확인의 정상 여부를 판단하는 방법
   b. 상태 확인의 생성, 업데이트 및 삭제
   c. 상태 확인의 상태 모니터링 및 알림 수신
   d. Amazon Route 53 DNS 모범 사례
3. DNS 서비스를 엔드포인트에 연결.
   a. Elastic Load Balancing을 트래픽의 대상으로 사용하는 경우 로드 밸런서의 리전 엔드포인트를 가리키는 Amazon Route 53을 사용하여 별칭 레코드를 생성합니다. 별칭 레코드를 생성하는 동안 대상 상태 평가 옵션을 예로 설정합니다.
   b. API Gateway가 사용되는 서버리스 워크로드 또는 프라이빗 API의 경우 Route 53을 사용하여 트래픽을 API Gateway로 전송합니다.
4. 콘텐츠 전송 네트워크를 결정합니다.
   a. 사용자에게 더 가까운 엣지 로케이션을 사용하여 콘텐츠를 제공하려면 먼저 CloudFront가 콘텐츠를 제공하는 방법을 이해합니다.
   b. 간단한 CloudFront 배포로 시작합니다. 그런 다음 CloudFront는 콘텐츠를 제공할 위치와 콘텐츠 제공을 추적 및 관리하는 방법에 대한 세부 정보를 확인합니다. CloudFront 배포를 설정할 때 다음 측면을 이해하고 고려해야 합니다.
      i. 캐싱과 CloudFront 엣지 로케이션의 작동 방식
      ii. CloudFront 캐시에서 직접 처리되는 요청의 비율 증가(캐시 적중률)
      iii. Amazon CloudFront Origin Shield 사용
      iv. CloudFront 오리진 장애 조치를 통한 고가용성 최적화
5. 애플리케이션 계층 보호 설정: AWS WAF는 가용성에 영향을 미치거나 보안을 손상시키거나 과도한 리소스를 소비할 수 있는 일반적인 웹 악용 및 봇으로부터 보호하는 데 도움이 됩니다. 더 깊이

이해하려면 AWS WAF의 작동 방식을 검토하고 애플리케이션 계층 HTTP POST 및 GET 플러드로부터 보호를 구현할 준비가 되면 Getting started with AWS WAF를 검토하세요. CloudFront와 함께 AWS WAF를 사용할 수도 있습니다.(how AWS WAF works with Amazon CloudFront features 참조).

6. 추가 DDoS 보호 설정: 기본적으로 모든 AWS 고객은 AWS Shield Standard를 사용하여 웹 사이트 또는 애플리케이션을 대상으로 하는 일반적이고 가장 자주 발생하는 네트워크 및 전송 계층 DDoS 공격으로부터 추가 비용 없이 보호를 받습니다. Amazon EC2, Elastic Load Balancing, Amazon CloudFront, AWS Global Accelerator 및 Amazon Route 53에서 실행되는 인터넷 연결 애플리케이션을 추가로 보호하기 위해 AWS Shield Advanced를 고려하고 복원력이 뛰어난 DDoS 아키텍처 예제를 검토할 수 있습니다. DDoS 공격으로부터 워크로드와 퍼블릭 엔드포인트를 보호하려면 Getting started with AWS Shield Advanced를 검토하세요.

## 리소스

관련 모범 사례:

- REL10-BP01 워크로드를 여러 위치에 배포
- REL11-BP04 복구 중 컨트롤 플레인이 아닌 데이터 플레인 사용
- REL11-BP06 이벤트가 가용성에 영향을 미치는 경우 알림 전송

관련 문서:

- APN 파트너: 네트워킹 계획을 지원할 수 있는 파트너
- AWS Marketplace for Network Infrastructure
- AWS Global Accelerator란 무엇입니까?
- What is Amazon CloudFront?
- What is Amazon Route 53?
- Elastic Load Balancing이란 무엇인가요?
- Network Connectivity capability - Establishing Your Cloud Foundations
- Amazon API Gateway란 무엇입니까?
- What are AWS WAF, AWS Shield, and AWS Firewall Manager?
- Amazon Application Recovery Controller란 무엇입니까?
- DNS 장애 조치에 대한 사용자 지정 상태 확인 구성

관련 비디오:

- AWS re:Invent 2022 - Improve performance and availability with AWS Global Accelerator
- AWS re:Invent 2020: Global traffic management with Amazon Route 53
- AWS re:Invent 2022 - Operating highly available Multi-AZ applications
- AWS re:Invent 2022 - Dive deep on AWS networking infrastructure
- AWS re:Invent 2022 - Building resilient networks

관련 예제:

- Amazon ARC(Application Recovery Controller)를 사용한 재해 복구
- AWS Global Accelerator 워크숍

## REL02-BP02 클라우드와 온프레미스 환경의 프라이빗 네트워크 간에 중복 연결 프로비저닝

클라우드와 온프레미스 환경의 프라이빗 네트워크 간 연결에 중복 구성을 구현하여 연결 복원력을 확보합니다. 이는 두 개 이상의 링크와 트래픽 경로를 배포하여 네트워크 장애 발생 시 연결을 유지함으로써 달성할 수 있습니다.

일반적인 안티 패턴:

- 하나의 네트워크 연결에만 의존하므로 단일 장애 지점이 발생합니다.
- 하나의 VPN 터널만 사용하거나 동일한 가용 영역에서 끝나는 여러 터널을 사용합니다.
- VPN 연결을 위해 하나의 ISP에 의존하므로 ISP 중단 시 완전한 장애가 발생할 수 있습니다.
- 네트워크 중단 발생 시 트래픽을 다시 라우팅하는 데 중요한 BGP와 같은 동적 라우팅 프로토콜을 구현하지 않습니다.
- VPN 터널의 대역폭 제한을 무시하고 백업 기능을 과대평가합니다.

이 모범 사례 확립의 이점: 클라우드 환경과 기업 또는 온프레미스 환경 간에 중복 연결을 구현하면 두 환경 간의 종속 서비스가 신뢰할 수 있는 기반에서 통신할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

AWS Direct Connect를 사용하여 온프레미스 네트워크를 AWS에 연결하는 경우 둘 이상의 온프레미스 위치와 둘 이상의 AWS Direct Connect 위치에 있는 개별 디바이스에서 끝나는 별도의 연결을 사용하여 최대 네트워크 복원력(99.99% SLA)을 달성할 수 있습니다. 이 토폴로지는 디바이스 장애, 연결 문제 및 완전한 위치 중단에 대한 복원력을 제공합니다. 또는 여러 위치에 대한 두 개의 개별 연결(각 온프레미스 위치가 단일 Direct Connect 위치에 연결됨)을 사용하여 높은 복원력(SLA 99.9%)을 달성할 수 있습니다. 이 접근 방식은 광케이블 절단이나 디바이스 장애로 인한 연결 중단을 방지하고 전체 위치 장애를 완화하는 데 도움이 됩니다. AWS Direct Connect 복원 도구는 AWS Direct Connect 토폴로지 설계에 도움이 될 수 있습니다.

또한 기본 AWS Direct Connect 연결에 대한 비용 효율적인 백업으로 AWS Transit Gateway에서 끝나는 AWS Site-to-Site VPN을 고려할 수 있습니다. 이 설정을 사용하면 여러 VPN 터널에서 Equal-Cost Multipath(ECMP) 라우팅이 가능하므로 각 VPN 터널이 1.25Gbps로 제한되어 있더라도 최대 50Gbps의 처리량이 가능합니다. 하지만 네트워크 중단을 최소화하고 안정적인 연결을 제공하는 데는 AWS Direct Connect가 여전히 가장 효과적인 선택이라는 점에 유의해야 합니다.

인터넷을 통해 VPN을 사용하여 클라우드 환경을 온프레미스 데이터 센터에 연결하는 경우, 단일 사이트 간 VPN 연결의 일부로 두 개의 VPN 터널을 구성합니다. 각 터널은 고가용성을 위해 서로 다른 가용 영역에서 끝나야 하며, 이중화된 하드웨어를 사용하여 온프레미스 디바이스 장애를 방지해야 합니다. 또한 단일 ISP 중단으로 인해 VPN 연결이 완전히 중단되는 것을 방지하려면 온프레미스 위치에서 다양한 인터넷 서비스 제공업체(ISP)의 여러 인터넷 연결을 고려합니다. 다양한 라우팅과 인프라를 갖춘 ISP, 특히 AWS 엔드포인트에 대한 별도의 물리적 경로가 있는 ISP를 선택하면 연결 가용성이 높아집니다.

다중 AWS Direct Connect 연결 및 다중 VPN 터널(또는 둘의 조합)을 통한 물리적 이중화 외에도 Border Gateway Protocol(BGP) 동적 라우팅을 구현하는 것도 중요합니다. 동적 BGP는 실시간 네트워크 상태 및 구성된 정책을 기반으로 경로 간에 트래픽을 자동으로 다시 라우팅합니다. 이러한 동적 동작은 링크 또는 네트워크 장애 발생 시 네트워크 가용성과 서비스 연속성을 유지하는 데 특히 유용합니다. 이는 대체 경로를 빠르게 선택하여 네트워크의 복원력과 신뢰성을 향상시킵니다.

구현 단계

- AWS 및 온프레미스 환경 간에 가용성이 뛰어난 연결을 확보합니다.
  - 별도로 배포된 프라이빗 네트워크 간에 여러 AWS Direct Connect 연결 또는 VPN 터널을 사용합니다.
  - 가용성을 높이려는 경우에는 여러 AWS Direct Connect 위치를 사용합니다.
  - 여러 AWS 리전을 사용하는 경우 2개 이상의 위치에서 중복 구성을 생성합니다.

- 가능하면 AWS Transit Gateway를 사용하여 VPN 연결을 종료합니다.
- AWS Marketplace 어플라이언스를 평가하여 VPN을 종료하거나 SD-WAN을 AWS로 확장합니다. AWS Marketplace 어플라이언스를 사용하는 경우 다른 가용 영역에서 고가용성을 위해 중복 인스턴스를 배포합니다.
- 온프레미스 환경에 중복 연결을 제공합니다.
  - 가용성 요구 사항을 충족하려면 여러 AWS 리전에 대한 중복 연결이 필요할 수 있습니다.
  - AWS Direct Connect Resiliency Toolkit을 사용하여 시작합니다.

## 리소스

관련 문서:

- AWS Direct Connect Resiliency Recommendations
- Using Redundant Site-to-Site VPN Connections to Provide Failover
- Routing policies and BGP communities
- Active/Active and Active/Passive Configurations in AWS Direct Connect
- APN 파트너: 네트워킹 계획을 지원할 수 있는 파트너
- AWS Marketplace for Network Infrastructure
- Amazon Virtual Private Cloud Connectivity Options 백서
- 확장 가능하고 안전한 다중 VPC AWS 네트워크 인프라 구축
- Using redundant Site-to-Site VPN connections to provide failover
- Using the AWS Direct Connect Resiliency Toolkit to get started
- VPC 엔드포인트 및 VPC 엔드포인트 서비스(AWS PrivateLink)
- Amazon VPC란 무엇인가?
- What is a transit gateway?
- AWS Site-to-Site VPN란 무엇입니까?
- Direct Connect 게이트웨이 사용

관련 비디오:

- AWS re:Invent 2018: Advanced VPC Design and New Capabilities for Amazon VPC
- AWS re:Invent 2019: AWS Transit Gateway reference architectures for many VPCs

# REL02-BP03 확장 및 가용성을 위한 IP 서브넷 할당 계정 확인

Amazon VPC IP 주소 범위는 가용 영역의 서브넷에 IP 주소를 할당하고 추후 확장을 고려하는 등 워크로드의 요구 사항을 수용할 수 있도록 충분히 커야 합니다. 여기에는 로드 밸런서, EC2 인스턴스 및 컨테이너 기반 애플리케이션이 포함됩니다.

네트워크 토폴로지를 계획할 때는 첫 단계로 IP 주소 공간 자체를 정의합니다. 각 VPC에는 RFC 1918 지침에 따라 프라이빗 IP 주소 범위를 할당해야 합니다. 이 프로세스의 일부로 다음 요구 사항을 준수하세요.

- 리전당 두 개 이상의 VPC에 대한 IP 주소 공간을 허용합니다.
- VPC 내에서 여러 가용 영역을 포함할 수 있도록 여러 서브넷을 위한 공간을 허용합니다.
- 사용되지 않은 CIDR 블록 공간은 향후 확장을 위해 VPC 내에 남겨둡니다.
- 기계 학습용 스팟 플릿, Amazon EMR 클러스터 또는 Amazon Redshift 클러스터 등 사용할 수 있는 임시 Amazon EC2 인스턴스 플릿의 요구 사항을 충족할 IP 주소 공간이 있는지 확인합니다. 각 Kubernetes 포드에는 기본적으로 VPC CIDR 블록의 라우팅 가능한 주소가 할당되므로 Amazon Elastic Kubernetes Service(Amazon EKS)와 같은 Kubernetes 클러스터도 비슷하게 고려해야 합니다.
- 참고로 각 서브넷 CIDR 블록에서 처음 4개의 IP 주소와 마지막 IP 주소는 예약되므로 사용할 수 없습니다.
- VPC에 할당된 초기 VPC CIDR 블록은 변경 또는 삭제가 불가능하지만 중첩되지 않은 추가 CIDR 블록을 VPC에 추가할 수 있습니다. 서브넷 IPv4 CIDR은 변경할 수 없지만 IPv6 CIDR은 변경할 수 있습니다.
- 가능한 가장 큰 VPC CIDR 블록은 /16이고 가장 작은 블록은 /28입니다.
- 다른 연결된 네트워크(VPC, 온프레미스 또는 기타 클라우드 제공업체)를 고려하고 중복되지 않는 IP 주소 공간을 확보하세요. 자세한 내용은 REL02-BP05 연결된 모든 프라이빗 주소 공간에서 겹치지 않는 프라이빗 IP 주소 범위 적용를 참조하세요.

원하는 성과: 확장 가능한 IP 서브넷은 향후 성장을 수용하고 불필요한 낭비를 방지하는 데 도움이 될 수 있습니다.

일반적인 안티 패턴:

- 향후 성장을 고려하지 않으면 CIDR 블록이 너무 작아지고 재구성이 필요하여 가동 중단이 발생할 수 있습니다.
- Elastic Load Balancer가 사용할 수 있는 IP 주소 수를 잘못 추정합니다.

- 트래픽이 많은 여러 로드 밸런서를 동일한 서브넷에 배포
- IP 주소 사용량 모니터링에 실패하면서 자동 규모 조정 메커니즘을 사용합니다.
- CIDR 범위가 미래 성장 기대치를 훨씬 상회하는 지나치게 큰 것으로 정의하면 주소 범위가 겹치는 다른 네트워크와의 피어링이 어려울 수 있습니다.

이 모범 사례 확립의 이점: 이렇게 하면 워크로드의 증가를 수용하고 확장 시 가용성을 계속 제공할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

성장, 규정 준수 및 다른 제품과 통합을 수용할 수 있도록 네트워크 계획을 수립합니다. 성장은 과소 평가될 수 있고 규정 준수는 변경될 수 있으며 적절한 계획 없이는 프라이빗 네트워크 연결을 구현하기가 어려울 수 있습니다.

- 서비스 요구 사항, 지연 시간, 규정, 재해 복구(DR) 요구 사항을 기준으로 관련 AWS 계정 및 리전을 선택합니다.
- 리전별 VPC 배포에 대한 요구 사항을 파악합니다.
- VPC 크기를 파악합니다.
  - 다중 VPC 연결을 배포할 것인지 여부를 결정합니다.
    - What Is a Transit Gateway?
    - Single Region Multi-VPC Connectivity
  - 규정 요구 사항에 따라 분리된 네트워킹이 필요한지 결정합니다.
  - 현재 및 미래의 요구 사항을 수용할 수 있도록 적절한 크기의 CIDR 블록으로 VPC를 만드세요.
    - 성장 전망을 알 수 없는 경우 향후 다시 구성할 필요가 없도록 더 큰 CIDR 블록을 사용하는 것이 좋습니다.
  - 이중 스택 VPC의 일부로 서브넷에 IPv6 주소 지정 지정을 고려하세요. IPv6은 대규모 IPv4 주소를 필요로 하는 임시 인스턴스 또는 컨테이너를 포함하는 프라이빗 서브넷에서 사용하기에 적합합니다.

## 리소스

관련 Well-Architected 모범 사례:

- REL02-BP05 연결된 모든 프라이빗 주소 공간에서 겹치지 않는 프라이빗 IP 주소 범위 적용

관련 문서:

- APN 파트너: 네트워킹 계획을 지원할 수 있는 파트너
- AWS Marketplace for Network Infrastructure
- Amazon Virtual Private Cloud Connectivity Options 백서
- 다중 데이터 센터 HA 네트워크 연결
- Single Region Multi-VPC Connectivity
- Amazon VPC란 무엇인가?
- AWS에서의 IPv6
- IPv6 on reference architectures
- Amazon Elastic Kubernetes Service launches IPv6 support
- VPC - Classic Load Balancer에 대한 권장 사항
- 가용 영역 서브넷 - Application Load Balancer
- 가용 영역 - Network Load Balancer

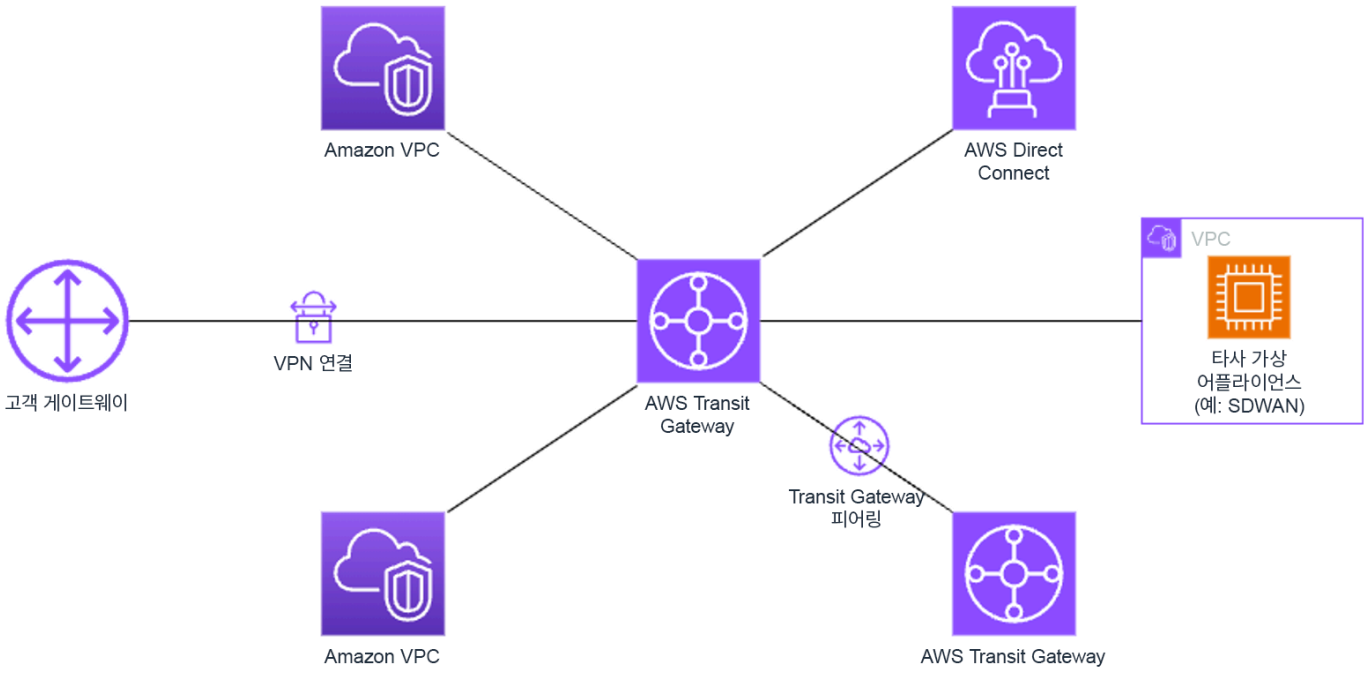
관련 비디오:

- AWS re:Invent 2018: Advanced VPC Design and New Capabilities for Amazon VPC (NET303)
- AWS re:Invent 2019: AWS Transit Gateway reference architectures for many VPCs (NET406-R1)
- AWS re:Invent 2023: AWS Ready for what's next? Designing networks for growth and flexibility (NET310)

## REL02-BP04 다대다 메시보다 허브 앤 스포크 토폴로지 선호

Virtual Private Cloud(VPC) 및 온프레미스 네트워크와 같은 여러 프라이빗 네트워크를 연결할 때는 메시 토폴로지보다 허브 앤 스포크 토폴로지를 선택하세요. 메시 토폴로지는 각 네트워크가 다른 네트워크에 직접 연결되어 복잡성과 관리 오버헤드가 증가하지만 허브 앤 스포크 아키텍처는 단일 허브를 통해 연결을 중앙 집중화합니다. 이러한 중앙 집중화로 네트워크 구조가 단순해지고 운영성, 확장성과 제어 가능성이 높아집니다.

AWS Transit Gateway는 AWS에서 허브 앤 스포크 네트워크를 구축할 수 있도록 설계되었으며 확장성과 가용성이 뛰어난 관리형 서비스입니다. 네트워크의 중앙 허브 역할을 하면서 네트워크 세분화, 중앙 집중식 라우팅, 클라우드와 온프레미스 환경 모두에 대한 간소화된 연결을 제공합니다. 다음 그림은 AWS Transit Gateway를 사용하여 허브 앤 스포크 토폴로지를 빌드하는 방법을 보여줍니다.




원하는 성과: 중앙 허브를 통해 가상 프라이빗 클라우드(VPC)와 온프레미스 네트워크를 연결했습니다. 확장성이 뛰어난 클라우드 라우터 역할을 하는 허브를 통해 피어링 연결을 구성합니다. 복잡한 피어링 관계를 사용할 필요가 없으므로 라우팅이 간소화됩니다. 네트워크 간 트래픽은 암호화되며 네트워크를 격리할 수 있습니다.

일반적인 안티 패턴:

- 복잡한 네트워크 피어링 규칙을 구축합니다.
- 네트워크 간에 서로 통신해서는 안 되는 경로를 제공합니다(예: 상호 의존성이 없는 별도의 워크로드).
- 허브 인스턴스의 거버넌스가 비효율적입니다.

이 모범 사례 확립의 이점: 연결된 네트워크 수가 증가함에 따라 메시 연결의 관리 및 확장은 점점 더 어려워집니다. 메시 아키텍처는 추가 인프라 구성 요소, 구성 요구 사항 및 배포 고려 사항과 같은 추가 문제를 야기합니다. 또한 메시는 데이터 플레인 및 컨트롤 플레인 구성 요소를 관리하고 모니터링하기 위한 추가 오버헤드를 도입합니다. 메시 아키텍처의 고가용성을 제공하는 방법, 메시 상태 및 성능을 모니터링하는 방법, 메시 구성 요소의 업그레이드를 처리하는 방법을 고려해야 합니다.

반면 허브 앤 스포크 모델은 여러 네트워크에서 중앙 집중식 트래픽 라우팅을 설정합니다. 데이터 플레인 및 컨트롤 플레인 구성 요소의 관리 및 모니터링에 대한 보다 간단한 접근 방식을 제공합니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

Network Services 계정이 없는 경우 계정을 생성합니다. 조직의 Network Services 계정에 허브를 배치합니다. 이 접근 방식을 사용하면 네트워크 엔지니어가 허브를 중앙에서 관리할 수 있습니다.

허브 앤 스포크 모델의 허브는 가상 프라이빗 클라우드(VPC)와 온프레미스 네트워크 간에 흐르는 트래픽에 대한 가상 라우터 역할을 합니다. 이 접근 방식은 네트워크 복잡성을 줄이고 네트워킹 문제를 더 쉽게 해결할 수 있도록 합니다.

상호 연결하려는 VPC, AWS Direct Connect 및 Site-to-Site VPN 연결을 포함하여 네트워크 설계를 고려합니다.

각 Transit Gateway VPC 연결에 대해 별도의 서브넷을 사용하는 것을 고려하세요. 서브넷별로 작은 CIDR(예: /28)을 사용하여 컴퓨팅 리소스를 위한 주소 공간을 더 많이 확보하세요. 또한 하나의 네트워크 ACL을 만들어 허브에 연결된 모든 서브넷과 연결합니다. 인바운드 및 아웃바운드 방향 모두에서 네트워크 ACL을 열어 둡니다.

통신해야 하는 네트워크 간에만 라우팅이 제공되도록 라우팅 테이블을 설계하고 구현합니다. 서로 통신해서는 안 되는 네트워크 간 경로를 생략합니다(예: 상호 종속성이 없는 별도의 워크로드 간).

구현 단계

1. 네트워크를 계획합니다. 연결할 네트워크를 결정하고 중복 CIDR 범위를 공유하지 않는지 확인합니다.
2. AWS Transit Gateway를 생성하고 VPC를 연결합니다.
3. 필요한 경우 VPN 연결 또는 Direct Connect 게이트웨이를 생성하여 Transit Gateway와 연결합니다.
4. 연결된 VPC와 기타 연결 간의 트래픽이 라우팅되는 방식을 Transit Gateway 라우팅 테이블의 구성을 통해 정의합니다.
5. 성능 및 비용 최적화를 위해 필요에 따라 구성을 모니터링하고 조정하는 데 Amazon CloudWatch를 사용합니다.

## 리소스

관련 모범 사례:

- REL02-BP03 확장 및 가용성을 위한 IP 서브넷 할당 계정 확인
- REL02-BP05 연결된 모든 프라이빗 주소 공간에서 겹치지 않는 프라이빗 IP 주소 범위 적용

관련 문서:

- What Is a Transit Gateway?
- 전송 게이트웨이 설계 모범 사례
- 확장 가능하고 안전한 다중 VPC AWS 네트워크 인프라 구축
- Building a global network using AWS Transit Gateway Inter-Region peering
- Amazon Virtual Private Cloud 연결 옵션
- APN 파트너: 네트워킹 계획을 지원할 수 있는 파트너
- AWS Marketplace for Network Infrastructure

관련 비디오:

- AWS re:Invent 2023 - AWS networking foundations
- AWS re:Invent 2023 - Advanced VPC designs and new capabilities

관련 워크숍:

- AWS Transit Gateway Workshop

## REL02-BP05 연결된 모든 프라이빗 주소 공간에서 겹치지 않는 프라이빗 IP 주소 범위 적용

피어링되거나, Transit Gateway를 통해 연결되거나, VPN을 통해 연결된 경우 각 VPC의 IP 주소 범위가 중첩되지 않아야 합니다. VPC와 온프레미스 환경 간의 IP 주소 충돌 또는 사용하는 다른 클라우드 제공업체와의 IP 주소 충돌을 방지해야 합니다. 필요한 경우 프라이빗 IP 주소 범위를 할당할 수 있어야 합니다. IP 주소 관리(IPAM) 시스템이 이러한 작업을 자동화하는 데 도움이 될 수 있습니다.

원하는 성과:

- VPC, 온프레미스 환경 또는 기타 클라우드 제공업체 간의 IP 주소 범위 충돌이 없습니다.
- 적절한 IP 주소 관리를 통해 네트워크 요구 사항이 성장하고 변화함에 따라 네트워크 인프라를 쉽게 확장할 수 있습니다.

일반적인 안티 패턴:

- VPC에서 온프레미스, 기업 네트워크 또는 기타 클라우드 제공업체와 동일한 IP 범위를 사용합니다.
- 워크로드를 배포하는 데 사용되는 VPC의 IP 범위를 추적하지 않습니다.
- 스프레드시트와 같은 수동 IP 주소 관리 프로세스를 사용합니다.
- CIDR 블록의 크기를 너무 크거나 작게 지정하여 IP 주소가 낭비되거나 워크로드를 위한 주소 공간이 부족합니다.

이 모범 사례 확립의 이점: 네트워크를 능동적으로 계획하면 상호 연결된 네트워크에서 동일한 IP 주소가 여러 번 사용되는 것을 방지할 수 있습니다. 이렇게 하면 다른 애플리케이션을 사용하는 워크로드의 일부에서 라우팅 문제가 발생하는 것을 방지할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

Amazon VPC IP Address Manager와 같은 IPAM을 사용하여 CIDR 사용을 모니터링하고 관리합니다. AWS Marketplace에서도 여러 IPAM을 사용할 수 있습니다. AWS에서 잠재적인 사용량을 평가하고, 기존 VPC에 CIDR 범위를 추가하고, VPC를 생성하여 사용량을 계획에 맞춰 늘릴 수 있습니다.

구현 단계

- 현재 CIDR 소비량을 파악합니다(예: VPC, 서브넷 등).
  - 서비스 API를 사용하여 현재 CIDR 소비량을 파악합니다.
  - Amazon VPC IP 주소 관리자를 사용하여 리소스를 검색합니다.
- 현재 서브넷 사용량을 파악합니다.
  - 서비스 API 작업을 사용하여 각 리전의 VPC당 서브넷을 수집합니다.
  - Amazon VPC IP 주소 관리자를 사용하여 리소스를 검색합니다.
- 현재 사용량을 기록합니다.
- 중첩되지 않는 IP 범위를 생성했는지 판단합니다.
- 여유 용량을 계산합니다.
- 중첩된 IP 범위를 파악합니다. 새 주소 범위로 마이그레이션하거나 중첩되는 범위를 연결해야 하는 경우 프라이빗 NAT 게이트웨이 또는 AWS PrivateLink와 같은 기술 사용을 고려할 수 있습니다.

## 리소스

관련 모범 사례:

- 네트워크 보호

관련 문서:

- APN 파트너: 네트워킹 계획을 지원할 수 있는 파트너
- AWS Marketplace for Network Infrastructure
- Amazon Virtual Private Cloud Connectivity Options 백서
- 다중 데이터 센터 HA 네트워크 연결
- Connecting Networks with Overlapping IP Ranges
- Amazon VPC란 무엇인가?
- IPAM이란?

관련 비디오:

- AWS re:Invent 2023 - Advanced VPC designs and new capabilities
- AWS re:Invent 2019: AWS Transit Gateway reference architectures for many VPCs
- AWS re:Invent 2023 - Ready for what’s next? Designing networks for growth and flexibility
- AWS re:Invent 2021 - {New Launch} Manage your IP addresses at scale on AWS

# 워크로드 아키텍처

신뢰할 수 있는 워크로드는 소프트웨어와 인프라에 대한 사전 설계 결정에서 시작됩니다. 아키텍처 선택은 모든 6가지 Well-Architected 원칙에서 워크로드 동작에 영향을 미칩니다. 신뢰성을 달성하려면 특정 패턴을 따라야 합니다.

다음 섹션에서는 신뢰성을 위해 이러한 패턴과 함께 사용하는 모범 사례를 설명합니다.

주제

- 워크로드 서비스 아키텍처 설계
- 장애 방지를 위해 분산 시스템에서 상호 작용 설계
- 장애 완화 또는 극복을 위해 분산 시스템에서 상호 작용 설계

## 워크로드 서비스 아키텍처 설계

서비스 지향 아키텍처(SOA) 또는 마이크로서비스 아키텍처를 사용하여 확장성과 신뢰성이 뛰어난 워크로드를 구축합니다. 서비스 지향 아키텍처(SOA)는 서비스 인터페이스를 통해 소프트웨어 구성 요소를 재사용 가능하게 만드는 방식입니다. 마이크로서비스 아키텍처는 구성 요소를 더 작고 간단하게 만듭니다.

서비스 지향 아키텍처(SOA) 인터페이스는 일반적인 통신 표준을 사용하므로 새로운 워크로드에 신속하게 통합할 수 있습니다. SOA는 상호 의존적이고 분할되지 않는 단위로 구성된 모놀리식 아키텍처 구축 방식을 대체했습니다.

AWS에서는 항상 SOA를 사용해왔지만 현재는 마이크로서비스를 사용한 시스템 구축을 받아들였습니다. 마이크로서비스에는 여러 가지 이점이 있지만 가용성과 관련하여 가장 중요한 이점은 크기가 더 작고 구조도 더 단순하다는 것입니다. 마이크로서비스를 사용하면 여러 서비스에 필요한 가용성을 각기 독립적으로 설정함으로써 가용성 요구 수준이 가장 높은 마이크로서비스에 더 집중적으로 투자를 할 수 있습니다. 예를 들어 Amazon.com에서 상품 정보 페이지('상세 페이지')를 제공하려는 경우 마이크로서비스 수백 개를 간접 호출하여 페이지의 개별 부분을 작성합니다. 가격 및 상품 상세 내용을 제공할 수 있어야 하는 서비스도 몇 가지 있지만, 서비스를 사용할 수 없는 경우에는 페이지의 대다수 콘텐츠를 제외하면 됩니다. 사진, 리뷰 등의 요소도 고객이 상품을 구매할 수 있는 환경을 제공하는 과정에서 반드시 필요한 것은 아닙니다.

모범 사례

- REL03-BP01 워크로드를 세그먼트화하는 방법 선택
- REL03-BP02 특정 비즈니스 도메인 및 기능을 중심으로 서비스 구축

- REL03-BP03 API별로 서비스 계약 제공

## REL03-BP01 워크로드를 세그먼트화하는 방법 선택

워크로드 세그먼트화는 애플리케이션의 복원력 요구 사항을 결정할 때 중요합니다. 되도록 모놀리식 아키텍처는 피해야 합니다. 대신 어떤 애플리케이션 구성 요소를 마이크로서비스로 나눌 수 있을지 신중하게 고려합니다. 가능한 경우 애플리케이션 요구 사항에 따라 서비스 지향 아키텍처(SOA)와 마이크로서비스의 결합으로 마무리될 수도 있습니다. 상태 비저장일 수 있는 워크로드는 마이크로서비스로 배포할 수 있습니다.

원하는 성과: 워크로드는 지원 가능하고 확장 가능하며 가능한 한 느슨하게 결합되어 있어야 합니다.

워크로드를 세그먼트화하는 방법을 선택할 때 복잡성 대비 이점의 균형을 고려합니다. 신제품의 첫 출시에 필요한 것과 처음부터 워크로드를 확장할 때 필요한 것은 다릅니다. 기존 모놀리식을 리팩터링할 때 애플리케이션이 상태 비저장을 향한 해체를 얼마나 잘 지원하는지 고려해야 합니다. 서비스를 더 작은 부분으로 나누면 잘 정의된 소규모 팀에서 이러한 부분을 개발하고 관리할 수 있습니다. 그러나 크기가 작은 서비스는 복잡성을 불러올 수 있고 여기에는 지연 시간 증가, 디버깅 복잡성, 운영 부담 증가가 포함됩니다.

일반적인 안티 패턴:

- 마이크로서비스 Death Star는 원자성 구성 요소의 상호 의존성이 커져 한 구성 요소의 장애가 훨씬 더 큰 장애로 이어져 구성 요소가 모놀리식처럼 경직되고 취약해질 수 있습니다.

이 모범 사례 확립의 이점:

- 보다 구체적인 세그먼트를 사용하면 민첩성, 조직의 유연성 및 확장성이 향상됩니다.
- 서비스 중단의 영향이 감소합니다.
- 애플리케이션 구성 요소가 원자성이 더 큰 세그먼트화로 지원할 수 있는 여러 가능성 요구 사항을 가질 수 있습니다.
- 워크로드를 지원하는 팀의 책임이 잘 정의됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

### 구현 가이드

워크로드를 분할하는 방법에 따라 아키텍처 유형을 선택합니다. SOA 또는 마이크로서비스 아키텍처(또는 드문 경우 모놀리식 아키텍처)를 선택합니다. 처음에 모놀리식 아키텍처를 선택하더라도, 사용

자 채택에 따라 제품을 확장할 때 SOA 또는 마이크로서비스로 변경할 수 있는 모듈식 아키텍처인지 확인해야 합니다. SOA와 마이크로서비스는 더 작게 분할할 수 있기 때문에 현대의 확장 가능하고 신뢰할 수 있는 아키텍처로 선호되지만 특히 마이크로서비스 아키텍처를 배포하는 경우 장단점을 고려해야 합니다.

한 가지 주요 장단점은 분산 컴퓨팅 아키텍처가 구축되었지만 사용자 지연 시간 요구 사항을 충족하기가 더 어려워지며, 사용자 상호 작용을 디버그하고 추적하는 과정이 더 복잡해진다는 점입니다. AWS X-Ray를 사용하면 이 문제를 해결할 수 있습니다. 관리 중인 애플리케이션의 수가 증가함에 따라 운영 복잡성이 증가하므로 다수의 독립 구성 요소를 배포해야 한다는 점도 고려해야 합니다.

모놀리식, 서비스 지향, 마이크로서비스 아키텍처

## 구현 단계

- 애플리케이션을 리팩터링 또는 구축하기에 적절한 아키텍처를 결정합니다. SOA 및 마이크로서비스는 상태적으로 더 세분화된 조각화를 제공하며, 이는 확장 가능하고 신뢰할 수 있는 최신 아키텍처로서 선호됩니다. SOA는 마이크로서비스의 복잡성을 어느 정도 피하면서 더 세분화된 조각화를 실현하기에 좋은 절충안이 될 수 있습니다. 자세한 내용은 Microservice Trade-Offs를 참조하세요.
- 이를 워크로드에 적용할 수 있고 조직에서 지원할 수 있는 경우, 마이크로서비스 아키텍처를 사용하여 최고의 민첩성과 신뢰성을 실현해야 합니다. 자세한 내용은 AWS에서 마이크로서비스 구현을 참조하세요.
- 모놀리식을 더 작은 구성 요소로 리팩터링하려면 Strangler Fig 패턴 준수를 고려하세요. 여기에는 특정 애플리케이션 구성 요소를 새 애플리케이션 및 서비스로 점차적으로 교체하는 작업이 포함됩니다. AWS Migration Hub Refactor Spaces는 증분 리팩터링의 시작점 역할을 합니다. 자세한 내용은 Seamlessly migrate on-premises legacy workloads using a strangler pattern을 참조하세요.

- 마이크로서비스를 구현하려면 이러한 분산된 서비스가 서로 통신하기 위한 서비스 검색 메커니즘이 필요할 수 있습니다. AWS App Mesh를 서비스 지향 아키텍처와 함께 사용하면 신뢰할 있는 서비스 검색 및 액세스를 제공할 수 있습니다. 동적 DNS 기반 서비스 검색에는 AWS Cloud Map도 사용할 수 있습니다.
- 모놀리식에서 SOA로 마이그레이션하는 경우 Amazon MQ는 클라우드에서 레거시 애플리케이션을 다시 설계할 때 서비스 버스로 격차를 해소하는 데 도움이 될 수 있습니다.
- 공유 데이터베이스 하나를 사용하는 기존 모놀리식의 경우 데이터를 더 작은 세그먼트로 재구성하는 방법을 선택합니다. 사업부, 액세스 패턴 또는 데이터 구조별로 선택할 수 있습니다. 리팩터링 프로세스 중 이 지점에서 데이터베이스의 관계형 또는 비관계형(NoSQL) 유형을 사용하여 진행할지 선택해야 합니다. 자세한 내용은 SQL에서 NoSQL로를 참조하세요.

구현 계획의 작업 수준: 높음

## 리소스

관련 모범 사례:

- REL03-BP02 특정 비즈니스 도메인 및 기능을 중심으로 서비스 구축

관련 문서:

- Amazon API Gateway: OpenAPI를 사용하여 REST API 구성
- 서비스 지향 아키텍처란 무엇인가요?
- Bounded Context(도메인 중심 설계의 중심 패턴)
- AWS에서 마이크로서비스 구현
- Microservice Trade-Offs
- Microservices - a definition of this new architectural term
- AWS의 마이크로서비스
- AWS App Mesh란 무엇입니까?

관련 예제:

- Iterative App Modernization 워크숍

관련 비디오:

- Delivering Excellence with Microservices on AWS

## REL03-BP02 특정 비즈니스 도메인 및 기능을 중심으로 서비스 구축

서비스 지향 아키텍처(SOA)는 비즈니스 요구 사항에 따라 명확하게 정의된 기능으로 서비스를 정의합니다. 마이크로서비스는 도메인 모델과 경계 컨텍스트를 사용하여 비즈니스 컨텍스트 경계를 따라 서비스 경계를 그립니다. 비즈니스 도메인 및 기능에 초점을 맞추면 팀이 서비스에 대한 독립적인 신뢰성 요구 사항을 정의하는 데 도움이 됩니다. 경계 컨텍스트는 비즈니스 로직을 분리하고 캡슐화하므로 팀이 장애 처리 방법을 더 잘 판단할 수 있습니다.

원하는 성과: 엔지니어와 비즈니스 이해관계자가 공동으로 경계 컨텍스트를 정의하고 이를 사용하여 특정 비즈니스 기능을 충족하는 서비스로 시스템을 설계합니다. 이러한 팀은 이벤트 스토밍과 같은 확립된 관행을 사용하여 요구 사항을 정의합니다. 새로운 애플리케이션은 잘 정의된 경계와 느슨한 결합이 가능하도록 설계됩니다. 기존 모놀리스는 경계 컨텍스트로 분해되고 시스템 설계는 SOA 또는 마이크로서비스 아키텍처로 이전됩니다. 모놀리스를 리팩터링하는 경우에는 버블 컨텍스트 및 모놀리스 분해 패턴과 같은 확립된 접근 방식이 적용됩니다.

도메인 지향 서비스는 상태를 공유하지 않는 하나 이상의 프로세스로 실행됩니다. 이들은 수요 변동에 독립적으로 대응하고 도메인별 요구 사항을 고려하여 장애 시나리오를 처리합니다.

일반적인 안티 패턴:

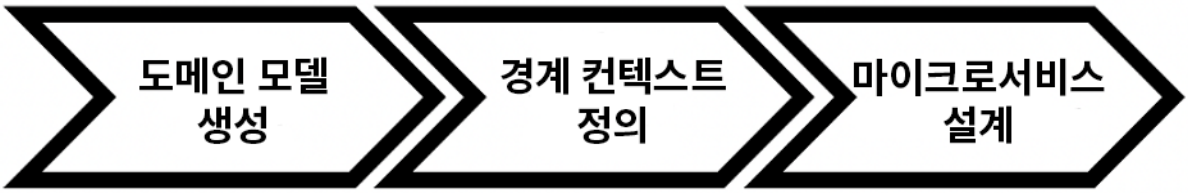
- 팀이 특정 비즈니스 도메인 대신 UI 및 UX, 미들웨어 또는 데이터베이스와 같은 특정 기술 도메인을 중심으로 구성됩니다.
- 애플리케이션이 여러 도메인 책임에 걸쳐 있습니다. 여러 경계 컨텍스트에 걸쳐 있는 서비스는 유지 관리가 더 어렵고 더 많은 테스트 작업이 필요하며 소프트웨어 업데이트에 여러 도메인 팀이 참여해야 할 수 있습니다.
- 도메인 엔터티 라이브러리와 같은 도메인 종속성은 서비스 간에 공유되므로 한 서비스 도메인을 변경하려면 다른 서비스 도메인도 변경해야 합니다.
- 서비스 계약과 비즈니스 로직은 엔터티를 공통적이고 일관된 도메인 언어로 표현하지 않으므로 변환 계층이 발생하여 시스템이 복잡해지고 디버깅 작업이 늘어납니다.

이 모범 사례 확립의 이점: 애플리케이션이 비즈니스 도메인을 기반으로 하는 독립적인 서비스로 설계되며 공통 비즈니스 언어를 사용합니다. 서비스를 독립적으로 테스트 및 배포할 수 있습니다. 서비스가 구현된 도메인에 대한 도메인별 복원력 요구 사항을 충족합니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

도메인 중심 의사 결정(DDD)은 비즈니스 도메인을 중심으로 소프트웨어를 설계하고 구축하는 기본 접근 방식입니다. 비즈니스 도메인에 초점을 맞춘 서비스를 구축할 때는 기존 프레임워크를 사용하는 것이 좋습니다. 기존 모놀리식 애플리케이션에서 작업할 때 애플리케이션을 서비스로 현대화하는 확립된 기술을 제공하는 분해 패턴을 활용할 수 있습니다.




도메인 기반 설계

## 구현 단계

- 팀은 이벤트 스토밍 워크숍을 통해 이벤트, 명령, 집계, 도메인을 가벼운 스티커 노트 형식으로 빠르게 파악할 수 있습니다.
- 도메인 컨텍스트에서 도메인 엔터티 및 기능이 형성되면 유사한 기능 및 특성을 공유하는 엔터티가 함께 그룹화되는 경계 컨텍스트를 사용하여 도메인을 서비스로 분할할 수 있습니다. 모델을 컨텍스트로 나누면 마이크로서비스의 경계를 지정하는 방법에 대한 템플릿을 사용할 수 있게 됩니다.
  - 예를 들어 Amazon.com 웹 사이트 엔터티에는 패키지, 배송, 일정, 가격, 할인 및 통화가 포함될 수 있습니다.
  - 패키지, 배송, 일정은 배송 컨텍스트로 그룹화되고 가격, 할인, 통화는 가격 컨텍스트로 그룹화됩니다.
- Decomposing monoliths into microservices에서는 마이크로서비스 리팩터링 패턴을 설명합니다. 비즈니스 역량, 하위 도메인 또는 트랜잭션별 분해 패턴을 사용하는 것은 도메인 중심 접근 방식과 부합됩니다.
- 버블 컨텍스트와 같은 전술적 기술을 사용하면 사전 재작성 및 DDD에 대한 완전한 약정 없이 기존 또는 레거시 애플리케이션에 DDD를 도입할 수 있습니다. 버블 컨텍스트 접근 방식에서는 새로 정의된 도메인 모델을 외부 영향으로부터 보호하는 손상 방지 계층 또는 서비스 매핑 및 조정을 사용하여 작은 경계 컨텍스트가 설정됩니다.

팀이 도메인 분석을 수행하고 엔티티 및 서비스 계약을 정의한 후에는 AWS 서비스를 활용하여 도메인 중심 설계를 클라우드 기반 서비스로 구현할 수 있습니다.

- 도메인의 비즈니스 규칙을 실행하는 테스트를 정의하여 개발을 시작합니다. 테스트 기반 개발(TDD)과 동작 기반 개발(BDD)은 팀이 서비스가 비즈니스 문제 해결에 초점을 맞출 수 있도록 도와줍니다.
- 비즈니스 도메인 요구 사항 및 마이크로서비스 아키텍처를 가장 잘 충족하는 AWS 서비스를 선택합니다.
  - AWS 서버리스를 사용하면 팀이 서버 및 인프라를 관리하는 대신 특정 도메인 로직에 집중할 수 있습니다.
  - AWS의 컨테이너는 인프라 관리를 간소화하므로 도메인 요구 사항에 집중할 수 있습니다.
  - 목적별 데이터베이스를 통해 도메인 요구 사항을 가장 적합한 데이터베이스 유형에 맞출 수 있습니다.
- Building hexagonal architectures on AWS에서는 기능적 요구 사항을 충족한 다음 통합 어댑터를 연결하기 위해 비즈니스 도메인에서 역방향으로 작업하여 서비스에 비즈니스 로직을 구축하는 프레임워크를 설명합니다. AWS 서비스를 통해 인터페이스 세부 정보를 비즈니스 로직과 분리하는 패턴은 팀이 도메인 기능에 집중하고 소프트웨어 품질을 개선하는 데 도움이 됩니다.

## 리소스

관련 모범 사례:

- REL03-BP01 워크로드를 세그먼트화하는 방법 선택
- REL03-BP03 API별로 서비스 계약 제공

관련 문서:

- AWS 마이크로서비스
- AWS에서 마이크로서비스 구현
- How to break a Monolith into Microservices
- Getting Started with DDD when Surrounded by Legacy Systems
- Domain-Driven Design: Tackling Complexity in the Heart of Software
- Building hexagonal architectures on AWS
- Decomposing monoliths into microservices
- Event Storming
- Messages Between Bounded Contexts
- Microservices
- Test-driven development

- Behavior-driven development

관련 예제:

- Designing Cloud Native Microservices on AWS (from DDD/EventStormingWorkshop)

관련 도구:

- AWS 클라우드 데이터베이스
- AWS의 서버리스
- AWS의 컨테이너

## REL03-BP03 API별로 서비스 계약 제공

서비스 계약은 컴퓨터가 인식할 수 있는 API 정의에 정의된 API 생산자 및 소비자 간의 문서화된 계약입니다. 계약 버전 관리 전략을 사용하면 소비자가 기존 API를 계속 사용하면서 준비가 될 때 애플리케이션을 최신 API로 마이그레이션할 수 있습니다. 생산자 배포는 계약을 준수하는 한 언제든지 가능합니다. 서비스 팀은 원하는 기술 스택을 사용하여 API 계약을 충족할 수 있습니다.

원하는 성과: 서비스 지향 또는 마이크로서비스 아키텍처로 구축된 애플리케이션은 통합 런타임 종속성을 유지하면서 독립적으로 작동할 수 있습니다. API 소비자 또는 생산자에게 배포되는 변경 사항은 양측이 공통 API 계약을 준수하는 경우 전체 시스템의 안정성을 방해하지 않습니다. 서비스 API를 통해 통신하는 구성 요소는 독립적인 기능 릴리스를 수행하거나 런타임 종속성을 업그레이드하거나 서로 거의 또는 전혀 영향을 주지 않고 재해 복구(DR) 사이트로 장애 조치할 수 있습니다. 또한 개별 서비스는 다른 서비스를 함께 확장할 필요 없이 독립적으로 확장하여 리소스 수요를 흡수할 수 있습니다.

일반적인 안티 패턴:

- 강력한 형식의 스키마 없이 서비스 API를 생성합니다. 이를 통해 프로그래밍 방식으로 검증할 수 없는 API 바인딩 및 페이로드를 생성하는 데 사용할 수 없는 API가 생성됩니다.
- 서비스 계약을 개발할 때 API 소비자가 업데이트 및 릴리스하거나 실패하도록 하는 버전 관리 전략을 채택하지 않습니다.
- 도메인 컨텍스트 및 언어에서의 통합 실패를 설명하는 대신 기본 서비스 구현의 세부 정보를 유출하는 오류 메시지.
- 서비스 구성 요소를 독립적으로 테스트할 수 있도록 API 계약을 사용하여 테스트 사례 및 모의 API 구현을 개발하지 않습니다.

이 모범 사례 확립의 이점: API 서비스 계약을 통해 통신하는 구성 요소로 구성된 분산 시스템은 신뢰성을 향상시킬 수 있습니다. 개발자는 컴파일 중에 형식 검사를 통해 개발 프로세스 초기에 잠재적 문제를 포착하여 요청 및 응답이 API 계약을 준수하고 필수 필드가 있는지 확인할 수 있습니다. API 계약은 API에 대한 명확한 자체 문서화 인터페이스를 제공하고 서로 다른 시스템과 프로그래밍 언어 간에 상호 운용성을 개선합니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

비즈니스 도메인을 식별하고 워크로드 세분화를 결정한 후에는 서비스 API를 개발할 수 있습니다. 먼저 컴퓨터가 인식할 수 있는 API 서비스 계약을 정의한 다음 API 버전 관리 전략을 구현합니다. REST, GraphQL 또는 비동기 이벤트와 같은 일반 프로토콜을 통해 서비스를 통합할 준비가 되면 AWS 서비스를 아키텍처에 통합하여 구성 요소를 강력한 형식의 API 계약과 통합할 수 있습니다.

서비스 API 계약을 위한 AWS 서비스

AWS 서비스(Amazon API Gateway, AWS AppSync, Amazon EventBridge 등)를 아키텍처에 통합하여 애플리케이션에서 API 서비스 계약을 사용합니다. Amazon API Gateway를 사용하면 기본 AWS 서비스 및 기타 웹 서비스와 직접 통합할 수 있습니다. API Gateway는 OpenAPI 사양 및 버전 관리를 지원합니다. AWS AppSync는 관리형 GraphQL 엔드포인트로, 쿼리, 변형, 구독을 위한 서비스 인터페이스를 정의하는 GraphQL 스키마를 정의하여 구성하는 엔드포인트입니다. Amazon EventBridge는 이벤트 스키마를 사용하여 이벤트를 정의하고 이벤트에 대한 코드 바인딩을 생성합니다.

## 구현 단계

- 먼저 API에 대한 계약을 정의합니다. 계약은 API의 기능을 표현할 뿐만 아니라 API 입출력에 대해 강력한 형식의 데이터 객체와 필드를 정의합니다.
- API Gateway에서 API를 구성할 때 엔드포인트의 OpenAPI 사양을 가져오고 내보낼 수 있습니다.
  - OpenAPI 정의를 가져오면 API 생성을 단순화하고 AWS Serverless Application Model 및 AWS 클라우드 개발 키트 (AWS CDK)와 같은 AWS의 코드형 인프라 도구와 통합될 수 있습니다.
  - API 정의를 내보래면 API 테스트 도구와의 통합을 단순화하고 서비스 소비자에게 통합 사양을 제공합니다.
- AWS AppSync로 GraphQL 스키마를 정의하여 GraphQL API를 정의하고 관리할 수 있습니다. 이를 통해 계약 인터페이스를 생성하고 복잡한 REST 모델, 여러 데이터베이스 테이블 또는 레거시 서비스와의 상호 작용을 단순화할 수 있습니다.

- AWS AppSync와 통합된 AWS Amplify 프로젝트는 애플리케이션에서 사용할 강력한 형식의 JavaScript 쿼리 파일과 Amazon DynamoDB 테이블용 AWS AppSync GraphQL 클라이언트 라이브러리를 생성합니다.
- Amazon EventBridge에서 서비스 이벤트를 사용하는 경우 이벤트는 스키마 레지스트리에 이미 있거나 OpenAPI Spec으로 정의한 스키마를 준수합니다. 레지스트리에 정의된 스키마를 사용하면 스키마 계약에서 클라이언트 바인딩을 생성하여 코드를 이벤트와 통합할 수도 있습니다.
- API 확장 또는 버전 관리. 선택적 필드 또는 필수 필드의 기본값으로 구성할 수 있는 필드를 추가할 때 더 간단한 옵션은 API를 확장하는 것입니다.
  - REST 및 GraphQL과 같은 프로토콜에 대한 JSON 기반 계약은 계약 확장에 적합할 수 있습니다.
  - SOAP와 같은 프로토콜에 대한 XML 기반 계약은 서비스 소비자와 함께 테스트하여 계약 연장 가능성을 결정해야 합니다.
- API 버전을 관리할 때는 단일 코드베이스에서 로직을 유지할 수 있도록 파사드를 사용하여 버전을 지원하는 프록시 버전 관리를 구현하는 것이 좋습니다.
  - API Gateway를 사용하면 요청 및 응답 매핑을 통해 새 필드에 기본값을 제공하거나 요청 또는 응답에서 제거된 필드를 제거하는 파사드를 설정하여 계약 변경 사항을 간단하게 흡수할 수 있습니다. 이 접근 방식을 사용하면 기본 서비스가 단일 코드베이스를 유지할 수 있습니다.

## 리소스

관련 모범 사례:

- REL03-BP01 워크로드를 세그먼트화하는 방법 선택
- REL03-BP02 특정 비즈니스 도메인 및 기능을 중심으로 서비스 구축
- REL04-BP02 느슨하게 결합된 종속성 구현
- REL05-BP03 재시도 직접 호출 제어 및 제한
- REL05-BP05 클라이언트 제한 시간 설정

관련 문서:

- 애플리케이션 프로그래밍 인터페이스(API)란 무엇인가요?
- Implementing Microservices on AWS
- Microservice Trade-Offs
- Microservices - a definition of this new architectural term
- AWS의 마이크로서비스

- OpenAPI에 대한 API Gateway 확장 작업
- OpenAPI-Specification
- GraphQL: Schemas and Types
- Amazon EventBridge code bindings

관련 예제:

- Amazon API Gateway: OpenAPI를 사용하여 REST API 구성
- Amazon API Gateway to Amazon DynamoDB CRUD application using OpenAPI
- Modern application integration patterns in a serverless age: API Gateway Service Integration
- Implementing header-based API Gateway versioning with Amazon CloudFront
- AWS AppSync: Building a client application

관련 비디오:

- Using OpenAPI in AWS SAM to manage API Gateway

관련 도구:

- Amazon API Gateway
- AWS AppSync
- Amazon EventBridge

# 장애 방지를 위해 분산 시스템에서 상호 작용 설계

분산 시스템은 통신 네트워크를 사용하여 서버 또는 서비스와 같은 구성 요소를 상호 연결합니다. 이러한 네트워크에서 데이터 손실이나 지연 시간이 발생하더라도 워크로드는 안정적으로 작동해야 합니다. 분산 시스템의 구성 요소는 다른 구성 요소나 워크로드에 부정적인 영향을 미치지 않는 방식으로 작동해야 합니다. 이러한 모범 사례는 장애를 예방하고 평균 고장 간격(MTBF)을 개선합니다.

모범 사례

- REL04-BP01 사용 중인 분산 시스템의 종류 파악
- REL04-BP02 느슨하게 결합된 종속성 구현
- REL04-BP03 일정한 작업 처리

- REL04-BP04 변경 작업에 멱등성 부여

## REL04-BP01 사용 중인 분산 시스템의 종류 파악

분산 시스템은 동기, 비동기 또는 배치 방식일 수 있습니다. 동기 시스템은 HTTP/S, REST 또는 원격 프로시저 직접 호출(RPC) 프로토콜을 사용하여 동기식 요청 및 응답 직접 호출을 수행함으로써 요청을 최대한 빨리 처리하고 서로 통신해야 합니다. 비동기 시스템은 개별 시스템을 결합하지 않고 중개 서비스를 통해 비동기식으로 데이터를 교환하여 서로 통신합니다. 배치 시스템은 대량의 입력 데이터를 수신하고, 사람의 개입 없이 자동화된 데이터 프로세스를 실행하며, 출력 데이터를 생성합니다.

원하는 성과: 동기, 비동기 및 배치 종속성과 효과적으로 상호 작용하는 워크로드를 설계합니다.

일반적인 안티 패턴:

- 워크로드가 종속성의 응답에 대해 무기한 대기하므로 요청이 수신되었는지 알지 못한 채로 워크로드 클라이언트가 제한 시간을 초과할 수 있습니다.
- 워크로드가 서로를 동기식으로 호출하는 종속 시스템 체인을 사용합니다. 이를 위해서는 전체 체인이 성공하기 전에 각 시스템을 사용할 수 있어야 하고 요청을 성공적으로 처리해야 합니다. 이로 인해 동작과 전체적인 가용성이 불안정질 수 있습니다.
- 워크로드가 종속성과 비동기적으로 통신하며, 중복 메시지를 수신할 수 있는 경우가 많지만 정확히 한 번 보장된 메시지 전달이라는 개념을 사용합니다.
- 워크로드가 적절한 배치 일정 예약 도구를 사용하지 않으며 동일한 배치 작업을 동시에 실행하도록 허용합니다.

이 모범 사례 확립의 이점: 특정 워크로드에서 동기, 비동기, 배치 중에 하나 이상의 통신 스타일을 구현하는 것이 일반적입니다. 이 모범 사례는 각 통신 스타일과 관련된 다양한 장단점을 식별하여 종속성에 중단이 발생했을 때 워크로드가 견딜 수 있도록 하는 데 도움이 됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

### 구현 가이드

다음 섹션에는 각 종속성 구현에 대한 일반적인 지침과 구체적인 구현 지침이 모두 포함되어 있습니다.

일반 지침

- 종속성이 제공하는 성능 및 신뢰성 서비스 수준 목표(SLO)가 워크로드의 성능 및 신뢰성 요구 사항을 충족하는지 확인하세요.

- AWS 관찰성 서비스로 응답 시간과 오류율을 모니터링하여 종속성이 워크로드에 필요한 수준의 서비스를 제공하고 있는지 확인하세요.
- 워크로드가 종속성과 통신할 때 직면할 수 있는 잠재적 문제를 식별하세요. 분산 시스템에는 아키텍처 복잡성, 운영 부담 및 비용을 증가시킬 수 있는 다양한 문제가 있습니다. 일반적인 문제로는 지연 시간, 네트워크 장애, 데이터 손실, 규모 조정, 데이터 복제 지연 등이 있습니다.
- 강력한 오류 처리 및 로깅을 구현하면 종속성 문제가 발생할 때 문제를 해결하는 데 도움이 됩니다.

### 동기식 종속성

동기식 통신에서는 워크로드가 종속성에 요청을 보내고 응답을 기다리는 작업을 차단합니다. 종속성은 요청을 수신하면 최대한 바로 처리하려고 시도하고 워크로드에 응답을 다시 보냅니다. 동기식 통신의 중요한 문제는 일시적 결합이 발생하여 워크로드와 해당 종속성을 동시에 사용할 수 있어야 한다는 것입니다. 워크로드가 종속성과 동기적으로 통신해야 하는 경우 다음 지침을 고려하세요.

- 워크로드가 단일 기능을 수행하기 위해 다수의 동기 종속성에 의존해서는 안 됩니다. 이렇게 여러 종속성이 체인을 이루면 요청을 성공적으로 완료하기 위해 경로상의 모든 종속성을 사용할 수 있어야 하기 때문에 전반적인 불안정성이 높아집니다.
- 종속성이 비정상이거나 사용할 수 없는 경우 오류 처리 및 재시도 전략을 결정하세요. 바이모달 동작은 사용하지 마세요. 바이모달 동작은 정상 모드와 장애 모드에서 워크로드가 서로 다른 동작을 보일 때를 말합니다. 바이모달 동작에 대한 자세한 내용은 REL11-BP05 정적 안정성을 사용하여 바이모달 동작 방지를 참조하세요.
- 빠른 실패가 워크로드를 기다리게 하는 것보다 낫다는 점을 명심하세요. 예를 들어, AWS Lambda 개발자 안내서에서는 Lambda 함수를 간접 호출할 때 재시도 및 실패를 처리하는 방법을 설명합니다.
- 워크로드가 종속성을 직접 호출할 때 제한 시간을 설정하세요. 이 기법을 사용하면 응답을 너무 오래 기다리거나 무한정 기다리지 않아도 됩니다. 이 주제에 대한 유용한 논의는 Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications를 참조하세요.
- 단일 요청을 처리하기 위해 워크로드에서 종속성에 수행하는 직접 호출 수를 최소화하세요. 둘 사이에 직접 호출이 너무 많으면 결합과 지연 시간이 늘어납니다.

### 비동기식 종속성

워크로드를 종속성에서 일시적으로 분리하려면 이 둘이 비동기적으로 통신해야 합니다. 비동기식 접근 방식을 사용하면 종속성 또는 종속성 체인이 응답을 보낼 때까지 기다릴 필요 없이 워크로드가 다른 처리를 계속 진행할 수 있습니다.

워크로드가 종속성과 비동기적으로 통신해야 하는 경우 다음 지침을 고려하세요.

- 사용 사례와 요구 사항에 따라 메시징을 사용할지 아니면 이벤트 스트리밍을 사용할지 결정하세요. 메시징을 사용하면 메시지 브로커를 통해 메시지를 주고받는 방식으로 워크로드가 종속 항목과 통신할 수 있습니다. 이벤트 스트리밍을 사용하면 워크로드와 해당 종속성이 스트리밍 서비스를 사용하여 가능한 한 빨리 처리해야 하는 연속 데이터 스트림으로 전달되는 이벤트를 게시하고 구독할 수 있습니다.

- 메시징과 이벤트 스트리밍은 메시지를 다르게 처리하므로 다음을 기준으로 장단점을 고려하여 결정해야 합니다.
  - 메시지 우선순위: 메시지 브로커는 우선순위가 높은 메시지를 일반 메시지보다 먼저 처리할 수 있습니다. 이벤트 스트리밍에서는 모든 메시지의 우선순위가 동일합니다.
  - 메시지 소비: 메시지 브로커는 소비자가 메시지를 받을 수 있도록 보장합니다. 이벤트 스트리밍 소비자는 마지막으로 읽은 메시지를 추적해야 합니다.
  - 메시지 정렬: 메시징의 경우 선입선출(FIFO) 방식을 사용하지 않는 한 메시지를 전송된 순서대로 정확하게 수신하지 못합니다. 이벤트 스트리밍은 항상 데이터가 생성된 순서를 지킵니다.
  - 메시지 삭제: 메시징의 경우 소비자가 메시지를 처리한 후 메시지를 삭제해야 합니다. 이벤트 스트리밍 서비스는 메시지를 스트림에 추가하고 메시지 보존 기간이 만료될 때까지 메시지가 스트림에 남아 있습니다. 이 삭제 정책 때문에 이벤트 스트리밍은 메시지 재생에 적합합니다.
- 종속성이 작업을 완료하는 시점을 워크로드가 어떻게 인식하는지 정의하세요. 예를 들어, 워크로드가 Lambda 함수를 비동기식으로 간접 호출하면 Lambda는 이벤트를 대기열에 배치하고 추가 정보 없이 성공 응답을 반환합니다. 처리가 완료되면 Lambda 함수는 결과를 대상으로 전송할 수 있으며, 성공 또는 실패에 따라 구성할 수 있습니다.
- 멱등성을 활용하여 중복 메시지를 처리하도록 워크로드를 구축하세요. 멱등성이란 동일한 메시지에 대해 워크로드가 두 번 이상 생성되더라도 워크로드의 결과가 변경되지 않는 것을 의미합니다. 네트워크 장애가 발생하거나 확인이 수신되지 않은 경우 메시징 또는 스트리밍 서비스가 메시지를 다시 전송한다는 점을 명심합니다.
- 워크로드가 종속성에서 응답을 받지 못하는 경우 요청을 다시 제출해야 합니다. 다른 요청을 처리할 수 있도록 워크로드의 CPU, 메모리 및 네트워크 리소스를 보존하려면 재시도 횟수를 제한하는 것이 좋습니다. AWS Lambda 설명서에는 비동기 호출의 오류를 처리하는 방법이 나와 있습니다.
- 적절한 관찰성, 디버깅, 추적 도구를 활용하여 워크로드와 종속성의 비동기식 통신을 관리하고 운영하세요. Amazon CloudWatch를 사용하여 메시징 및 이벤트 스트리밍 서비스를 모니터링할 수 있습니다. 또한 AWS X-Ray로 워크로드를 계측하여 문제 해결에 필요한 인사이트를 빠르게 얻을 수 있습니다.

### 배치 종속성

배치 시스템은 입력 데이터를 가져와 일련의 작업을 시작하고 수동 개입 없이 일부 출력 데이터를 생성합니다. 데이터 크기에 따라 작업 실행이 몇 분에서 경우에 따라 며칠까지 지속될 수 있습니다. 워크로드가 배치 종속성과 통신할 때는 다음 지침을 고려하세요.

- 워크로드에서 배치 작업을 실행해야 하는 기간을 정의하세요. 워크로드에서 반복 패턴을 설정하여 배치 시스템을 간접 호출할 수 있습니다(예: 매시간 또는 매월 말).
- 데이터 입력 및 처리된 데이터 출력의 위치를 결정합니다. 대규모로 워크로드에서 파일을 읽고 쓸 수 있는 Amazon Simple Storage Services(S3), Amazon Elastic File System(Amazon EFS), Amazon FSx for Lustre 중에서 스토리지 서비스를 선택합니다.
- 워크로드에서 여러 배치 작업을 간접 호출해야 하는 경우 AWS Step Functions를 활용하여 AWS 또는 온프레미스에서 실행되는 배치 작업의 오케스트레이션을 간소화할 수 있습니다. 이 샘플 프로젝트에서는 Step Functions, AWS Batch, Lambda를 사용한 배치 작업의 오케스트레이션을 보여줍니다.
- 배치 작업을 모니터링하여 완료하는 데 예상 외로 오래 걸리는 작업과 같은 이상 현상이 없는지 확인합니다. CloudWatch Container Insights와 같은 도구를 사용하여 AWS Batch 환경 및 작업을 모니터링할 수 있습니다. 이 경우 워크로드는 다음 작업이 시작되지 못하도록 멈추고 관련 직원에게 예외를 알립니다.

## 리소스

관련 문서:

- AWS 클라우드 운영: 모니터링 및 관찰성
- Amazon Builders' Library: 분산 시스템의 도전 과제
- REL11-BP05 정적 안정성을 사용하여 바이모달 동작 방지
- AWS Lambda 개발자 안내서: AWS Lambda의 오류 처리 및 자동 재시도
- Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications
- AWS 메시징
- 스트리밍 데이터란 무엇인가요?
- AWS Lambda 개발자 안내서: 비동기 호출
- Amazon Simple Queue Service FAQ: FIFO 대기열
- Amazon Kinesis Data Streams 개발자 안내서: Handling Duplicate Records

- Amazon Simple Queue Service 개발자 안내서: Available CloudWatch metrics for Amazon SQS
- Amazon Kinesis Data Streams 개발자 안내서: Monitoring the Amazon Kinesis Data Streams Service with Amazon CloudWatch
- AWS X-Ray 개발자 안내서: AWS X-Ray concepts
- AWS Samples on GitHub: AWS Step functions Complex Orchestrator App
- AWS Batch 사용 설명서: AWS Batch CloudWatch Container Insights

관련 비디오:

- AWS Summit SF 2022 - Full-stack observability and application monitoring with AWS (COP310)

관련 도구:

- Amazon CloudWatch
- Amazon CloudWatch Logs
- AWS X-Ray
- Amazon Simple Storage Service(S3)
- Amazon Elastic File System(Amazon EFS)
- Amazon FSx for Lustre
- AWS Step Functions
- AWS Batch

## REL04-BP02 느슨하게 결합된 종속성 구현

대기열 처리 시스템, 스트리밍 시스템, 워크플로 및 로드 밸런서와 같은 종속성은 느슨히 결합됩니다. 느슨한 결합은 한 구성 요소의 동작을 다른 종속 구성 요소에서 분리하여 복원력 및 민첩성을 높이는 데 도움이 됩니다.

대기열 처리 시스템, 스트리밍 시스템, 워크플로와 같은 종속성을 분리하면 시스템에서 변경 또는 장애의 영향을 최소화하는 데 도움이 됩니다. 이러한 분리를 통해 구성 요소의 동작이 이와 종속된 다른 구성 요소에 영향을 주지 않으므로 복원력과 민첩성이 향상됩니다.

강한 결합으로 구성된 시스템에서는 한 구성 요소를 변경하면 해당 구성 요소를 사용하는 다른 구성 요소도 변경해야 할 수 있으므로 결과적으로 모든 구성 요소의 성능이 저하될 수 있습니다. 느슨한 결합에서는 이 종속성이 분리되므로 종속 구성 요소에서는 버전이 지정되고 게시된 인터페이스만 알면 됩

니다. 종속성 간에 느스한 결합을 구현하면 한 구성 요소의 장애가 다른 구성 요소에 영향을 미치지 않도록 분리됩니다.

느슨한 결합을 사용하면 다른 종속 구성 요소에 미치는 위험을 최소화하면서 구성 요소의 코드를 수정하거나 기능을 추가할 수 있습니다. 또한 구성 요소 수준에서 세분화된 복원력을 지원하므로 종속성의 기본 구현을 스케일 아웃하거나 변경할 수도 있습니다.

느슨한 결합을 통해 복원력을 추가로 개선하려면 가능한 경우 구성 요소가 비동기식으로 상호 작용하도록 합니다. 이 모델은 즉각적인 응답이 필요하지 않고 요청이 등록되었다는 확인으로 충분한 상호 작용에 적합합니다. 이러한 상호 작용에는 이벤트를 생성하는 구성 요소와 이벤트를 사용하는 구성 요소가 포함됩니다. 두 구성 요소는 직접적인 지점 간 상호 작용을 통해 통합되지 않고 일반적으로 내구성이 있는 중간 스토리지 계층(예: Amazon SQS 대기열, Amazon Kinesis 또는 AWS Step Functions와 같은 스트리밍 데이터 플랫폼)을 통해 통합됩니다.

그림 4: 대기열 처리 시스템 및 로드 밸런서와 같은 종속성은 느슨하게 결합됨

Amazon SQS 대기열 및 AWS Step Functions는 느슨한 결합을 위한 중간 계층을 추가할 수 있는 두 가지 방법의 예입니다. AWS 클라우드에서는 Amazon EventBridge를 사용하여 이벤트 기반 아키텍처를 구축할 수도 있습니다. 그러면 클라이언트가 의존하는 서비스(이벤트 소비자)에서 클라이언트(이벤트 생산자)를 추상화할 수 있습니다. Amazon Simple Notification Service(SNS)는 높은 처리량의 푸시 기반 다대다 메시징이 필요할 때 효과적인 솔루션입니다. Amazon SNS 주제를 사용하면 게시자 시스템에서 다수의 구독자 엔드포인트로 메시지를 팬아웃하여 병렬 처리를 수행할 수 있습니다.

대기열은 다수의 장점을 제공하지만 대부분의 강성 실시간 시스템에서 임계 시간(주로 초 단위)을 초과한 요청은 무효한 요청(클라이언트가 포기하여 더 이상 응답을 기다리지 않는 요청)으로 간주되어 처리되지 않습니다. 이렇게 하면 오래된 요청 대신 여전히 유효한 요청일 가능성이 큰 최근 요청을 처리할 수 있습니다.

원하는 성과: 느슨하게 결합된 종속성을 구현하면 장애 노출 영역을 구성 요소 수준으로 최소화할 수 있으므로 문제를 진단하고 해결하는 데 도움이 됩니다. 또한 개발 주기를 간소화하여 팀이 이를 사용하는 다른 구성 요소의 성능에 영향을 주지 않고 모듈 수준에서 변경 사항을 구현할 수 있습니다. 이 접근 방식은 리소스 요구 사항 및 구성 요소 활용도를 기반으로 구성 요소 수준에서 스케일 아웃할 수 있는 기능을 제공하여 비용 효율성에 기여합니다.

일반적인 안티 패턴:

- 모놀리식 워크로드를 배포합니다.
- 장애 조치 또는 비동기식 요청 처리 기능 없이 워크로드 티어 사이에서 API 직접 호출.

- 공유 데이터를 사용하는 강한 결합. 느슨하게 결합된 시스템은 공유 데이터베이스나 다른 형태의 밀결합된 데이터 스토리지를 통한 데이터 공유를 피해야 합니다. 그러지 않으면 밀결합이 다시 도입되어 확장성이 저해될 수 있습니다.
- 배압을 무시합니다. 워크로드는 구성 요소가 동일한 속도로 데이터를 처리할 수 없을 때 수신 데이터의 속도를 늦추거나 중지할 수 있어야 합니다.

이 모범 사례 확립의 이점: 느슨한 결합은 한 구성 요소의 동작을 다른 종속 구성 요소에서 분리하여 복원력 및 민첩성을 높이는 데 도움이 됩니다. 한 구성 요소에서 발생한 장애는 다른 구성 요소로부터 격리됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

느슨하게 결합된 종속성을 구현합니다. 느슨하게 결합된 애플리케이션을 구축할 수 있는 솔루션은 다양합니다. 여기에는 완전관리형 대기열, 자동화된 워크플로, 이벤트 대응, API 등을 구현하기 위한 서비스가 포함됩니다. 이러한 서비스는 구성 요소의 동작을 다른 구성 요소로부터 분리하고 복원력과 민첩성을 높이는 데 도움이 될 수 있습니다.

- 이벤트 기반 아키텍처 구축: Amazon EventBridge를 사용하면 느슨하게 결합되고 분산된 이벤트 기반 아키텍처를 구축할 수 있습니다.
- 분산 시스템에서 대기열 구현: Amazon Simple Queue Service(Amazon SQS)를 사용하여 분산 시스템을 통합하고 분리할 수 있습니다.
- 구성 요소를 마이크로서비스로 컨테이너화: 마이크로서비스를 사용하면 잘 정의된 API를 통해 통신하는 독립적인 소규모 구성 요소로 구성된 애플리케이션을 구축할 수 있습니다. Amazon Elastic Container Service(Amazon ECS), Amazon Elastic Kubernetes Service(Amazon EKS)는 컨테이너를 빠르게 시작하는 데 도움을 줄 수 있습니다.
- Step Functions로 워크플로 관리: Step Functions는 여러 AWS 서비스를 유연한 워크플로로 조율할 수 있도록 도와줍니다.
- 게시 및 구독(pub/sub) 메시징 아키텍처 활용: Amazon Simple Notification Service(SNS)는 게시자(생산자)가 구독자(소비자)에게 메시지를 전달하도록 합니다.

### 구현 단계

- 이벤트 기반 아키텍처의 구성 요소는 이벤트로 인해 시작됩니다. 이벤트는 사용자가 장바구니에 품목을 추가하는 것과 같이 시스템에서 발생하는 작업입니다. 작업이 성공하면 시스템의 다음 구성 요소를 작동시키는 이벤트가 생성됩니다.

  - Building Event-driven Applications with Amazon EventBridge
  - AWS re:Invent 2022 - Designing Event-Driven Integrations using Amazon EventBridge
- 분산 메시징 시스템에는 대기열 기반 아키텍처를 위해 구현해야 하는 세 가지 주요 부분이 있습니다. 분산 시스템의 구성 요소, 분리에 사용되는 대기열(Amazon SQS 서버에 분산됨), 대기열의 메시지가 포함됩니다. 일반적인 시스템에는 메시지를 대기열로 보내는 생산자와 대기열에서 메시지를 수신하는 소비자가 있습니다. 대기열은 중복성을 위해 여러 Amazon SQS 서버에 메시지를 저장합니다.
  - Basic Amazon SQS architecture
  - Send Messages Between Distributed Applications with Amazon Simple Queue Service
- 마이크로서비스를 잘 활용하면 개별 팀이 느슨하게 결합된 구성 요소를 관리하므로 유지 관리 가능성과 확장성이 향상됩니다. 또한 변경 시 동작을 단일 구성 요소로 분리할 수 있습니다.
  - Implementing Microservices on AWS
  - Let's Architect! Architecting microservices with containers
- AWS Step Functions를 사용하면 분산 애플리케이션을 구축하고, 프로세스를 자동화하며, 마이크로서비스를 오케스트레이션하는 등의 작업을 수행할 수 있습니다. 여러 구성 요소를 자동화된 워크플로로 오케스트레이션하면 애플리케이션의 종속성을 분리할 수 있습니다.
  - Create a Serverless Workflow with AWS Step Functions and AWS Lambda
  - AWS Step Functions 시작하기

## 리소스

관련 문서:

- Amazon EC2: Ensuring Idempotency
- Amazon Builders' Library: 분산 시스템의 도전 과제
- The Amazon Builders' Library: Reliability, constant work, and a good cup of coffee
- Amazon EventBridge란 무엇인가요?
- What Is Amazon Simple Queue Service?
- Break up with your monolith
- Orchestrate Queue-based Microservices with AWS Step Functions and Amazon SQS
- Basic Amazon SQS architecture
- Queue-Based Architecture

관련 비디오:

- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes loose coupling, constant work, static stability)
- AWS re:Invent 2019: Moving to event-driven architectures (SVS308)
- AWS re:Invent 2019: Scalable serverless event-driven applications using Amazon SQS and Lambda
- AWS re:Invent 2022 - Designing event-driven integrations using Amazon EventBridge
- AWS re:Invent 2017: Elastic Load Balancing Deep Dive and Best Practices

## REL04-BP03 일정한 작업 처리

대규모 로드가 급속도로 변경되면 시스템에서 장애가 발생할 수 있습니다. 예를 들어 서버 수천 대의 상태를 모니터링하는 상태 확인을 수행하는 워크로드의 경우 매번 동일한 크기의 페이로드(현재 상태의 전체 스냅샷)를 전송해야 합니다. 장애가 발생한 서버가 없든 모든 서버에서 장애가 발생하든, 상태 확인 시스템은 대규모의 급속한 변경이 없는 일정한 작업을 처리합니다.

예를 들어 상태 확인 시스템이 100,000개의 서버를 모니터링하는 경우 서버 장애율이 정상적으로 낮을 때는 서버에 가해지는 로드가 작습니다. 그러나 중대한 이벤트로 인해 이러한 서버의 절반이 비정상 상태가 될 때 상태 확인 시스템에서 알림 시스템을 업데이트하고 클라이언트로 상태를 전달하려면 상태 확인 시스템이 과부하가 될 수 있습니다. 그렇기 때문에 대신 상태 확인 시스템에서 매번 현재 상태의 전체 스냅샷을 전송해야 합니다. 100,000개의 서버 상태(각각 비트로 표시됨)는 12.5KB 페이로드에 불과합니다. 장애가 발생한 서버가 없든 모든 서버에서 장애가 발생하든 상태 확인 시스템은 일정한 작업을 처리하므로 대규모의 급속한 변경이 시스템 안정성에 위협이 되지 않습니다. 이 방법으로 Amazon Route 53이 엔드포인트(예: IP 주소)에 대한 상태 확인을 처리하여 최종 사용자가 어떻게 엔드포인트에 라우팅되는지 판단합니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 낮음

### 구현 가이드

- 대규모 로드가 급속도로 변경될 때 시스템에서 장애가 발생하지 않도록 일정한 작업을 처리합니다.
- 느슨하게 결합된 종속성을 구현합니다. 대기열 처리 시스템, 스트리밍 시스템, 워크플로 및 로드 밸런서와 같은 종속성은 느슨히 결합됩니다. 느슨한 결합은 한 구성 요소의 동작을 다른 종속 구성 요소에서 분리하여 복원력 및 민첩성을 높이는 데 도움이 됩니다.

- The Amazon Builders' Library: Reliability, constant work, and a good cup of coffee
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes constant work)
  - 100,000개의 서버를 모니터링하는 상태 확인 시스템의 예제에서 성공 또는 실패 횟수와 관계없이 페이로드 크기가 일정하게 유지되도록 워크로드를 엔지니어링합니다.

## 리소스

관련 문서:

- Amazon EC2: Ensuring Idempotency
- Amazon Builders' Library: 분산 시스템의 도전 과제
- The Amazon Builders' Library: Reliability, constant work, and a good cup of coffee

관련 비디오:

- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes constant work)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes loose coupling, constant work, static stability)
- AWS re:Invent 2019: Moving to event-driven architectures (SVS308)

# REL04-BP04 변경 작업에 멱등성 부여

멱등성이 있는 서비스는 각 요청이 정확히 한 번만 처리되도록 합니다. 이렇게 하면 다수의 동일한 요청에서 단일 요청과 동일한 결과가 나옵니다. 이렇게 하면 클라이언트가 요청이 오류로 여러 번 처리된다는 염려 없이 재시도를 시행할 수 있습니다. 이를 위해 클라이언트는 멱등성 토큰을 사용하여 API 요청을 실행할 수 있으며 요청이 반복될 때마다 이 토큰이 사용됩니다. 멱등성이 있는 서비스 API는 시스템의 기본 상태가 변경되더라도 토큰을 사용하여 요청이 처음 완료되었을 때 반환된 응답과 동일한 응답을 반환합니다.

분산 시스템에서 작업을 최대 한 번(클라이언트가 한 번만 요청) 또는 최소 한 번(클라이언트가 성공 확인을 수신할 때까지 계속 요청) 수행하기가 상대적으로 간단합니다. 작업이 정확히 한 번 수행되도록

하여 다수의 동일한 요청에서 단일 요청과 동일한 결과를 얻기는 더 어렵습니다. API에서 멱등성 토큰을 사용하면 서비스가 중복 레코드를 생성할 필요가 없고 부작용 없이 변경 요청을 한 번 이상 수신할 수 있습니다.

원하는 성과: 모든 구성 요소 및 서비스에서 멱등성을 보장하기 위해 일관되고 잘 문서화되고 널리 채택된 접근 방식을 사용합니다.

일반적인 안티 패턴:

- 필요하지 않은 경우에도 무차별적으로 멱등성을 적용합니다.
- 멱등성을 구현하기 위한 지나치게 복잡한 로직을 도입합니다.
- 타임스탬프를 멱등성의 키로 사용합니다. 이렇게 하면 클럭 스큐 또는 동일한 타임스탬프를 사용하여 변경 사항을 적용하는 여러 클라이언트로 인해 부정확해질 수 있습니다.
- 멱등성을 위해 전체 페이로드를 저장합니다. 이 접근 방식에서는 모든 요청에 대한 전체 데이터 페이로드를 저장하고 새 요청마다 덮어씁니다. 이로 인해 성능이 저하되고 확장성에 영향을 미칠 수 있습니다.
- 서비스 간에 키를 일관되지 않게 생성합니다. 일관된 키가 없으면 서비스가 중복 요청을 인식하지 못하여 의도하지 않은 결과가 발생할 수 있습니다.

이 모범 사례 확립의 이점:

- 확장성 향상: 시스템은 추가 로직 또는 복잡한 상태 관리를 수행할 필요 없이 재시도 및 중복 요청을 처리할 수 있습니다.
- 향상된 신뢰성: 멱등성은 서비스가 일관된 방식으로 여러 동일한 요청을 처리하도록 지원하므로 의도하지 않은 부작용 또는 중복 레코드의 위험이 줄어듭니다. 이는 네트워크 장애 및 재시도가 일반적인 분산 시스템에서 특히 중요합니다.
- 데이터 일관성 개선: 동일한 요청이 동일한 응답을 생성하기 때문에 멱등성은 분산 시스템에서 데이터 일관성을 유지하는 데 도움이 됩니다. 이는 트랜잭션 및 작업의 무결성을 유지하는 데 필수적입니다.
- 오류 처리: 멱등성 토큰을 사용하면 오류 처리가 더 간단해집니다. 클라이언트가 문제로 인해 응답을 받지 못하는 경우 동일한 멱등성 토큰으로 요청을 안전하게 다시 보낼 수 있습니다.
- 운영 투명성: 멱등성을 통해 모니터링 및 로깅을 개선할 수 있습니다. 서비스는 멱등성 토큰으로 요청을 로깅할 수 있으므로 문제를 더 쉽게 추적하고 디버깅할 수 있습니다.
- 간소화된 API 계약: 클라이언트와 서버 측 시스템 간의 계약을 간소화하고 잘못된 데이터 처리에 대한 두려움을 줄일 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

분산 시스템에서 작업을 최대 한 번(클라이언트가 한 번만 요청) 또는 최소 한 번(클라이언트가 성공이 확인될 때까지 계속 요청) 수행하기가 상대적으로 간단합니다. 그러나 정확히 한 번 동작을 구현하기는 어렵습니다. 이를 위해 클라이언트는 각 요청에 대한 멱등성 토큰을 생성하고 제공해야 합니다.

멱등성 토큰을 사용하면 서비스가 새 요청과 반복된 요청을 구분할 수 있습니다. 서비스가 멱등성 토큰이 포함된 요청을 수신하면 토큰이 이미 사용되었는지 확인합니다. 토큰이 사용된 경우 서비스는 저장된 응답을 검색하고 반환합니다. 새 토큰인 경우 서비스는 요청을 처리하고 토큰과 함께 응답을 저장한 다음, 응답을 반환합니다. 이 메커니즘은 모든 응답에 멱등성을 부여하여 분산 시스템의 신뢰성과 일관성을 향상시킵니다.

멱등성은 이벤트 기반 아키텍처의 중요한 동작이기도 합니다. 이러한 아키텍처는 일반적으로 Amazon SQS, Amazon MQ, Amazon Kinesis Streams 또는 Amazon Managed Streaming for Apache Kafka(Amazon MSK)와 같은 메시지 대기열에 의해 지원됩니다. 경우에 따라 한 번만 게시된 메시지가 실수로 두 번 이상 전달될 수 있습니다. 게시자가 메시지에 멱등성 토큰을 생성하고 포함할 때 수신된 중복 메시지를 처리해도 동일한 메시지에 대해 반복 작업이 발생하지 않도록 요청합니다. 소비자는 수신된 각 토큰을 추적하고 중복 토큰이 포함된 메시지를 무시해야 합니다.

또한 서비스와 소비자는 수신된 멱등성 토큰이 호출하는 다운스트림 서비스에 멱등성 토큰을 전달해야 합니다. 처리 체인의 모든 다운스트림 서비스는 메시지를 두 번 이상 처리할 때의 부작용을 방지하기 위해 멱등성을 구현해야 할 책임이 있습니다.

구현 단계

1. 멱등성이 있는 작업 식별

   어떤 작업에 멱등성이 필요한지 결정합니다. 여기에는 일반적으로 POST, PUT 및 DELETE HTTP 메서드와 데이터베이스 삽입, 업데이트 또는 삭제 작업이 포함됩니다. 읽기 전용 쿼리와 같이 상태를 변경하지 않는 작업은 부작용이 없는 한 일반적으로 멱등성이 필요하지 않습니다.

2. 고유한 식별자 사용

   발신자가 보낸 각 멱등성이 있는 작업 요청에 고유한 토큰을 포함합니다. 요청에 직접 포함하거나 메타데이터(예: HTTP 헤더)의 일부로 포함하면 됩니다. 이렇게 하면 수신자가 중복된 요청 또는 작업을 인식하고 처리할 수 있습니다. 토큰에 일반적으로 사용되는 식별자에는 Universally Unique Identifiers(UUID) 및 K-Sortable Unique Identifiers(KSUID)가 있습니다.

3. 상태 추적 및 관리

워크로드에서 각 작업 또는 요청의 상태를 유지 관리합니다. 이는 데이터베이스, 캐시 또는 기타 영구 스토어에 멱등성 토큰과 해당 상태(예: 보류, 완료 또는 실패)를 저장하여 달성할 수 있습니다. 이 상태 정보를 통해 워크로드는 중복 요청 또는 작업을 식별하고 처리할 수 있습니다.

잠금, 트랜잭션 또는 낙관적 동시성 제어와 같이 필요한 경우 적절한 동시성 제어 메커니즘을 사용하여 일관성과 원자성을 유지합니다. 여기에는 멱등성 토큰을 기록하고 요청 서비스와 관련된 모든 변경 작업을 실행하는 프로세스가 포함됩니다. 이렇게 하면 레이스 조건을 방지하고 멱등성이 있는 작업이 올바르게 실행되는지 확인할 수 있습니다.

스토리지 및 성능을 관리하기 위해 데이터 저장소에서 오래된 멱등성 토큰을 정기적으로 제거합니다. 스토리지 시스템이 지원하는 경우 데이터에 만료 타임스탬프(종종 Time To Live, 즉 TTL 값이라고 함)를 사용하는 것이 좋습니다. 멱등성 토큰 재사용 가능성은 시간이 지남에 따라 줄어듭니다.

일반적으로 멱등성 토큰 및 관련 상태를 저장하는 데 흔히 사용되는 AWS 스토리지 옵션은 다음과 같습니다.

- Amazon DynamoDB: DynamoDB는 지연 시간이 짧은 성능과 고가용성을 제공하는 NoSQL 데이터베이스 서비스이므로 멱등성 관련 데이터의 스토리지로 적합합니다. DynamoDB의 키-값 및 문서 데이터 모델을 사용하면 멱등성 토큰 및 관련 상태 정보를 효율적으로 저장하고 검색할 수 있습니다. DynamoDB는 애플리케이션이 멱등성 토큰을 삽입할 때 TTL 값을 설정하는 경우 멱등성 토큰을 자동으로 만료시킬 수도 있습니다.
- Amazon ElastiCache: ElastiCache는 처리량이 높고 지연 시간이 짧으며 비용이 저렴한 멱등성 토큰을 저장할 수 있습니다. 애플리케이션이 멱등성 토큰을 삽입할 때 TTL 값을 설정하는 경우 ElastiCache(Redis)와 ElastiCache(Memcached) 모두 멱등성 토큰을 자동으로 만료시킬 수 있습니다.
- Amazon Relational Database Service(Amazon RDS): Amazon RDS를 사용하여 멱등성 토큰 및 관련 상태 정보를 저장할 수 있습니다. 특히 애플리케이션이 이미 다른 목적으로 관계형 데이터베이스를 사용하는 경우 더욱 그렇습니다.
- Amazon Simple Storage Service(Amazon S3): Amazon S3는 확장성과 내구성이 뛰어난 객체 스토리지 서비스로, 멱등성 토큰 및 관련 메타데이터를 저장하는 데 사용할 수 있습니다. S3의 버전 관리 기능은 멱등성이 있는 작업 상태를 유지 관리하는 데 특히 유용할 수 있습니다. 스토리지 서비스의 선택은 일반적으로 멱등성 관련 데이터의 볼륨, 필요한 성능 특성, 내구성 및 가용성에 대한 요구, 멱등성 메커니즘이 전체 워크로드 아키텍처와 통합되는 방식과 같은 요인에 따라 달라집니다.

4. 멱등성이 있는 작업 구현

API 및 워크로드 구성 요소가 멱등성이 있도록 설계합니다. 워크로드 구성 요소에 멱등성 검사를 통합합니다. 요청을 처리하거나 작업을 수행하기 전에 고유 식별자가 이미 처리되었는지 확인합니다. 이미 처리된 경우 작업을 다시 실행하는 대신 이전 결과를 반환합니다. 예를 들어 클라이언트가 사용자를 생성하라는 요청을 보내는 경우 동일한 고유 식별자를 가진 사용자가 이미 존재하는지 확인합니다. 사용자가 있는 경우 새 사용자 정보를 생성하는 대신 기존 사용자 정보를 반환해야 합니다. 마찬가지로 대기열 소비자가 중복된 멱등성 토큰이 포함된 메시지를 받는 경우 소비자는 메시지를 무시해야 합니다.

요청의 멱등성을 검증하는 포괄적인 테스트 제품군을 생성합니다. 성공적인 요청, 실패한 요청 및 중복 요청과 같은 다양한 시나리오를 포함해야 합니다.

워크로드가 AWS Lambda 함수를 활용하는 경우 Powertools for AWS Lambda를 고려하세요. Powertools for AWS Lambda는 AWS Lambda 함수를 사용할 때 서버리스 모범 사례를 구현하고 개발자 속도를 높이기 위한 개발자 도구 모음입니다. 특히 Lambda 함수를 재시도해도 안전한 멱등성 작업으로 변환하는 유틸리티를 제공합니다.

5. 멱등성을 명확하게 전달

   API 및 워크로드 구성 요소를 문서화하여 작업의 멱등성을 명확하게 전달합니다. 이를 통해 클라이언트는 기대되는 동작 및 워크로드와 안정적으로 상호 작용하는 방법을 이해할 수 있습니다.

6. 모니터링 및 감사

   모니터링 및 감사 메커니즘을 구현하여 예상치 못한 응답 변경 또는 과도한 중복 요청 처리와 같이 응답의 멱등성과 관련된 문제를 탐지합니다. 이렇게 하면 워크로드의 문제 또는 예상치 못한 동작을 감지하고 조사하는 데 도움이 될 수 있습니다.

## 리소스

관련 모범 사례:

- REL05-BP03 재시도 직접 호출 제어 및 제한
- REL06-BP01 워크로드의 모든 구성 요소 모니터링(생성)
- REL06-BP03 알림 전송(실시간 처리 및 경보)
- REL08-BP02 배포의 일부로 기능 테스트 통합

관련 문서:

- The Amazon Builders' Library: Making retries safe with idempotent APIs
- Amazon Builders' Library: 분산 시스템의 도전 과제
- The Amazon Builders' Library: Reliability, constant work, and a good cup of coffee
- Amazon Elastic Container Service: Ensuring idempotency
- How do I make my Lambda function idempotent?
- Ensuring idempotency in Amazon EC2 API requests

관련 비디오:

- Building Distributed Applications with Event-driven Architecture - AWS Online Tech Talks
- AWS re:Invent 2,023 - Building next-generation applications with event-driven architecture
- AWS re:Invent 2,023 - Advanced integration patterns & trade-offs for loosely coupled systems
- AWS re:Invent 2,023 - Advanced event-driven patterns with Amazon EventBridge
- AWS re:Invent 2,018 - Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes loose coupling, constant work, static stability)
- AWS re:Invent 2,019 - Moving to event-driven architectures (SVS308)

관련 도구:

- Idempotency with AWS Lambda Powertools (Java)
- Idempotency with AWS Lambda Powertools (Python)
- AWS Lambda Powertools GitHub 페이지

## 장애 완화 또는 극복을 위해 분산 시스템에서 상호 작용 설계

분산 시스템에서 구성 요소(예: 서버 또는 서비스)는 통신 네트워크를 사용하여 상호 연결됩니다. 워크로드는 이러한 네트워크에서 데이터 손실 또는 지연 시간이 발생하더라도 신뢰할 수 있는 상태로 작동해야 합니다. 분산 시스템의 구성 요소는 다른 구성 요소나 워크로드에 부정적인 영향을 미치지 않는 방식으로 작동해야 합니다. 이러한 모범 사례를 준수하면 워크로드가 스트레스 또는 장애를 견디고, 더 빠르게 이를 복구하며, 이러한 장애의 영향을 완화할 수 있습니다. 그러면 결과적으로 평균 복구 시간(MTTR)이 개선됩니다.

이러한 모범 사례는 장애를 예방하고 평균 고장 간격(MTBF)을 개선합니다.

모범 사례

- REL05-BP01 관련 하드 종속성을 소프트 종속성으로 변환하는 단계적 성능 저하 구현
- REL05-BP02 요청 제한
- REL05-BP03 재시도 직접 호출 제어 및 제한
- REL05-BP04 빠른 실패 및 대기열 제한
- REL05-BP05 클라이언트 제한 시간 설정
- REL05-BP06 가능한 경우 시스템을 상태 비저장으로 설계
- REL05-BP07 비상 레버 구현

## REL05-BP01 관련 하드 종속성을 소프트 종속성으로 변환하는 단계적 성능 저하 구현

애플리케이션 구성 요소는 종속성을 사용할 수 없게 되더라도 핵심 기능을 계속 수행해야 합니다. 약간 오래된 데이터, 대체 데이터를 제공하거나 데이터를 제공하지 않을 수도 있습니다. 이를 통해 핵심 비즈니스 가치를 제공하는 동시에 지역화된 장애로 인한 전체 시스템 기능의 방해를 최소한으로 줄일 수 있습니다.

원하는 성과: 구성 요소의 종속성이 비정상 상태인 경우에도 구성 요소 자체가 성능이 저하된 방식으로 작동할 수 있습니다. 구성 요소의 장애 모드는 정상 작동으로 간주되어야 합니다. 워크플로는 이러한 장애가 완전한 장애로 이어지지 않거나 최소한 예측 가능하고 복구 가능한 상태로 이어지도록 설계되어야 합니다.

일반적인 안티 패턴:

- 필요한 핵심 비즈니스 기능을 식별하지 못합니다. 종속성 실패 시에도 구성 요소가 작동하는지 테스트하지 않습니다.
- 오류가 발생 시 또는 여러 종속성 중 하나만 사용할 수 없고 일부 결과가 여전히 반환될 수 있는 경우 데이터를 제공하지 않습니다.
- 트랜잭션이 부분적으로 실패하면 일관되지 않은 상태가 발생합니다.
- 중앙 파라미터 스토어에 액세스할 수 있는 다른 방법이 없습니다.
- 새로 고침 실패로 인한 결과를 고려하지 않고 로컬 상태를 무효화하거나 비웁니다.

이 모범 사례 확립의 이점: 단계적 성능 저하를 통해 시스템 전체의 가용성이 향상되고 장애가 발생하더라도 가장 중요한 기능의 작동을 유지할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

단계적 성능 저하를 구현하면 종속성 장애가 구성 요소 기능에 미치는 영향을 최소화하는 데 도움이 됩니다. 구성 요소가 종속성 장애를 감지하고 다른 구성 요소 또는 고객에게 미치는 영향을 최소화하는 방식으로 문제를 해결하는 것이 가장 좋습니다.

단계적 성능 저하를 고려하는 설계란 종속성 설계 시 잠재적인 장애 모드를 고려하는 것을 의미합니다. 각 장애 모드에서 구성 요소의 가장 중요한 기능을 대부분 또는 최소한 직접 호출자 또는 고객에게 제공할 수 있는 방법을 마련합니다. 이러한 고려 사항은 테스트 및 검증이 가능한 추가 요구 사항이 될 수 있습니다. 이상적으로는 구성 요소가 하나 이상의 종속성이 실패하더라도 적절한 방식으로 핵심 기능을 수행할 수 있어야 합니다.

이것은 기술적 논의인 만큼이나 비즈니스 논의이기도 합니다. 모든 비즈니스 요구 사항은 중요하며 가능하면 충족되어야 합니다. 그러나 모든 요구 사항이 충족되지 않을 때 어떤 일이 일어날지 묻는 것은 여전히 의미가 있습니다. 시스템은 가용성 및 일관성을 유지하도록 설계될 수 있지만 한 가지 요구 사항을 이루지 못하는 상황에서 어느 것이 더 중요할까요? 결제 처리의 경우 일관성이 필요할 수 있습니다. 실시간 애플리케이션의 경우 가용성이 필요할 수 있습니다. 고객 대상 웹 사이트의 경우 고객의 기대에 따라 답이 달라질 수 있습니다.

즉, 구성 요소의 요구 사항과 그 핵심 기능으로 간주되어야 하는 요소에 따라 달라진다는 의미입니다. 예시:

- 전자 상거래 웹 사이트는 맞춤형 추천, 최고 순위 제품, 고객 주문 상태 등 다양한 시스템의 데이터를 랜딩 페이지에 표시할 수 있습니다. 한 업스트림 시스템에 장애가 발생하더라도 고객에게 오류 페이지를 표시하는 대신 다른 모든 데이터를 표시하는 것이 좋습니다.
- 배치 쓰기를 수행하는 구성 요소는 개별 작업 중 하나가 실패하더라도 배치를 계속 처리할 수 있습니다. 재시도 메커니즘을 구현하는 것은 간단해야 합니다. 이렇게 하려면 어떤 작업이 성공했는지, 어떤 작업이 실패했는지, 왜 실패했는지에 대한 정보를 직접 호출자에게 반환하거나 실패한 요청을 DLQ(Dead Letter Queue)에 넣어 비동기 재시도를 구현하면 됩니다. 실패한 작업에 대한 정보도 기록해야 합니다.
- 트랜잭션을 처리하는 시스템은 개별 업데이트가 모두 실행되었는지 또는 전혀 실행되지 않았는지 확인해야 합니다. 분산 트랜잭션의 경우 동일한 트랜잭션의 이후 작업이 실패할 경우 Saga 패턴을 사용하여 이전 작업을 롤백할 수 있습니다. 여기서 핵심은 일관성을 유지하는 것입니다.
- 시간이 중요한 시스템은 적시에 응답하지 않는 종속성을 처리할 수 있어야 합니다. 이러한 경우 회로 차단기 패턴을 사용할 수 있습니다. 종속성의 응답이 시간 제한하기 시작하면 시스템은 추가적인 직접 호출이 이루어지지 않는 폐쇄 상태로 전환될 수 있습니다.

- 애플리케이션은 파라미터 스토어에서 파라미터를 읽을 수 있습니다. 기본 파라미터 세트를 사용하여 컨테이너 이미지를 생성하고 파라미터 스토어를 사용할 수 없는 경우에 이러한 이미지를 사용하는 것이 유용할 수 있습니다.

구성 요소 장애 시 사용되는 경로는 테스트가 필요하며 기본 경로보다 훨씬 간단해야 합니다. 일반적으로 폴백 전략은 피해야 합니다.

## 구현 단계

외부 및 내부 종속성을 식별합니다. 종속성에서 어떤 종류의 장애가 발생할 수 있는지 고려합니다. 이러한 장애 발생 시 업스트림 및 다운스트림 시스템과 고객에게 미치는 부정적인 영향을 최소화할 수 있는 방법을 강구합니다.

다음은 종속성 목록 및 장애 발생 시 단계적으로 성능을 저하시키는 방법입니다.

1. 종속성의 부분적 실패: 구성 요소가 다운스트림 시스템에 여러 요청을 보낼 수 있습니다. 이때 한 시스템에 여러 요청을 보내거나 여러 시스템에 한 번 요청할 수 있습니다. 비즈니스 상황에 따라 이를 처리하는 여러 방법이 적절할 수 있습니다(자세한 내용은 구현 지침의 이전 예제 참조).
2. 다운스트림 시스템이 높은 부하로 인해 요청 처리 불가: 다운스트림 시스템에 대한 요청이 지속적으로 실패하는 경우 계속 재시도하는 것은 의미가 없습니다. 이로 인해 이미 과부하된 시스템에 추가 부하가 발생하고 복구가 더 어려워질 수 있습니다. 여기서 회로 차단기 패턴을 활용하여 다운스트림 시스템에 대한 직접 호출 실패를 모니터링할 수 있습니다. 많은 수의 직접 호출이 실패하면 다운스트림 시스템으로의 추가 요청 전송이 중단되고 다운스트림 시스템을 다시 사용할 수 있는지 여부를 테스트하기 위한 직접 호출이 가끔 전송됩니다.
3. 파라미터 스토어 사용 불가: 파라미터 스토어를 변환하기 위해 컨테이너 또는 머신 이미지에 포함된 소프트 종속성 캐싱 또는 정상적인 기본값을 사용할 수 있습니다. 참고로 이러한 기본값은 최신 상태로 유지되어야 하며 테스트 모음에 포함되어야 합니다.
4. 모니터링 서비스 또는 작동하지 않는 기타 종속성 사용 불가: 구성 요소가 간헐적으로 로그, 지표 또는 추적을 중앙 모니터링 서비스로 보낼 수 없는 경우에도 비즈니스 기능을 평소처럼 실행하는 것이 가장 좋은 경우가 많습니다. 오랫동안 자동으로 지표를 로깅 또는 푸시하지 않는 것은 허용되지 않는 경우가 많습니다. 또한 일부 사용 사례에서는 규정 준수 요구 사항을 충족하기 위해 완전한 감사 항목이 필요할 수 있습니다.
5. 관계형 데이터베이스의 프라이머리 인스턴스 사용 불가: 거의 모든 관계형 데이터베이스와 마찬가지로 Amazon Relational Database Service는 기본 라이터 인스턴스를 하나만 가질 수 있습니다. 이로 인해 쓰기 워크로드에 단일 장애 지점이 생기고 규모 조정이 더 어려워집니다. 고가용성을 위한 다중 AZ 구성을 사용하거나 더 나은 규모 조정을 위해 Amazon Aurora Serverless를 사용하면 이러한 문제를 부분적으로 완화할 수 있습니다. 고가용성 요구 사항이 매우 높은 경우 기본 라이터에

게 전혀 의존하지 않는 것이 좋습니다. 읽기만 하는 쿼리의 경우 읽기 전용 복제본을 사용할 수 있습니다. 이 복제본은 중복성과 스케일 업뿐만 아니라 스케일 아웃도 가능합니다. 예를 들어 Amazon Simple Queue Service 대기열에 쓰기를 버퍼링하여 기본 항목을 일시적으로 사용할 수 없는 경우에도 고객의 쓰기 요청을 계속 수락할 수 있습니다.

## 리소스

관련 문서:

- Amazon API Gateway: 처리량 향상을 위해 API 요청 제한
- CircuitBreaker (summarizes Circuit Breaker from “Release It!” book)
- Error Retries and Exponential Backoff in AWS
- Michael Nygard “Release It! Design and Deploy Production-Ready Software”
- Amazon Builders' Library: 분산 시스템의 폴백 방지
- Amazon Builders' Library: 심각한 수준의 대기열 백로그 방지
- Amazon Builders' Library: 캐싱 관련 당면 과제 및 전략
- Amazon Builders' Library: 시간 제한, 재시도 및 지터를 사용한 백오프

관련 비디오:

- Retry, backoff, and jitter: AWS re:Invent 2019: Introducing The Amazon Builders’ Library (DOP328)

# REL05-BP02 요청 제한

예상치 못한 수요 증가로 인한 리소스 고갈을 완화하기 위해 요청을 제한합니다. 스로틀링 속도 이하의 요청은 처리되지만 정의된 한도를 초과한 요청은 거부되고 요청이 스로틀링되었음을 알리는 메시지가 표시됩니다.

원하는 성과: 갑작스러운 고객 트래픽 증가, 플러딩 공격 또는 대량 재시도로 인한 대규모 볼륨 급증이 요청 제한을 통해 완화되므로 워크로드가 지원되는 요청 볼륨을 정상적으로 계속 처리할 수 있습니다.

일반적인 안티 패턴:

- API 엔드포인트 제한이 구현되지 않거나 예상 볼륨을 고려하지 않고 기본값으로 유지됩니다.
- API 엔드포인트가 로드 테스트되지 않거나 제한 한도가 테스트되지 않습니다.

- 요청 크기 또는 복잡성을 고려하지 않고 요청 속도를 제한합니다.
- 최대 요청 속도 또는 최대 요청 크기를 테스트하지만 둘 다 테스트하지는 않습니다.
- 리소스가 테스트에서 설정된 한도대로 프로비저닝되지 않습니다.
- 사용 계획이 애플리케이션 간(A2A) API 소비자를 위해 구성되거나 고려되지 않습니다.
- 수평적으로 스케일링되는 대기열 소비자에게는 최대 동시성 설정이 구성되어 있지 않습니다.
- IP 주소별 속도 제한이 구현되지 않았습니다.

이 모범 사례 확립의 이점: 제한 한도를 설정한 워크로드는 예상치 못한 볼륨 급증 상황에서도 정상적으로 작동하고 수락된 요청 로드를 성공적으로 처리할 수 있습니다. API 및 대기열에 대한 요청이 갑자기 또는 지속적으로 급증하면 요청이 제한되어 요청 처리 리소스가 소진되지 않습니다. 속도 제한은 개별 요청자를 제한하므로 단일 IP 주소 또는 API 소비자로부터 오는 대량의 트래픽이 리소스를 소진하여 다른 소비자에게 영향을 미치는 일이 없습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

서비스는 알려진 용량의 요청을 처리하도록 설계되어야 합니다. 이 용량은 부하 테스트를 통해 설정할 수 있습니다. 요청 도착 속도가 한도를 초과할 경우 적절한 응답은 요청이 제한되었다는 신호를 보냅니다. 그러면 소비자가 오류를 처리하고 나중에 다시 시도할 수 있습니다.

서비스에 제한 구현이 필요한 경우 요청마다 토큰이 계산되는 토큰 버킷 알고리즘을 구현하는 것이 좋습니다. 토큰은 초당 제한 속도로 다시 충전되고 요청당 토큰 1개씩 비동기적으로 사용됩니다.

토큰 버킷 알고리즘.

Amazon API Gateway는 계정 및 리전 한도에 따라 토큰 버킷 알고리즘을 구현하며 사용 계획을 통해 클라이언트별로 구성할 수 있습니다. 또한 Amazon Simple Queue Service(Amazon SQS) 및 Amazon Kinesis는 요청을 버퍼링하여 요청 속도를 낮추고 처리 가능한 요청에 대해 더 높은 제한 속도를 허용할 수 있습니다. 마지막으로 AWS WAF를 사용하여 비정상적으로 높은 부하를 유발하는 특정 API 소비자를 제한하는 속도 제한을 구현할 수 있습니다.

## 구현 단계

API Gateway에서 API에 대한 제한 한도를 구성하고 제한이 초과되면 `429 Too Many Requests` 오류를 반환할 수 있습니다. AWS WAF를 AWS AppSync 및 API Gateway 엔드포인트와 함께 사용하여 IP 주소별 속도 제한을 활성화할 수 있습니다. 또한 시스템에서 비동기 처리를 허용할 수 있는 경우 메시지를 대기열 또는 스트림에 배치하여 서비스 클라이언트에 대한 응답 속도를 높일 수 있으며, 이를 통해 제한 속도를 높일 수 있습니다.

비동기 처리에서는 Amazon SQS를 AWS Lambda용 이벤트 소스로 구성한 경우 최대 동시성을 구성하여 높은 이벤트 속도가 워크로드 또는 계정의 다른 서비스에 필요한 사용 가능한 계정 동시 실행 할당량을 소모하지 않도록 할 수 있습니다.

API Gateway는 토큰 버킷의 관리형 구현을 제공하지만 API Gateway를 사용할 수 없는 경우 서비스용 토큰 버킷의 언어별 오픈 소스 구현(리소스의 관련 예제 참조)을 활용할 수 있습니다.

- 리전별 계정 수준, 단계별 API 및 사용 계획 수준별 API 키에서 API Gateway 제한 한도를 이해하고 구성합니다.
- 플러드로부터 보호하고 악성 IP를 차단하기 위해 AWS WAF 속도 제한 규칙을 API Gateway 및 AWS AppSync 엔드포인트에 적용합니다. A2A 소비자를 위한 AWS AppSync API 키에도 속도 제한 규칙을 구성할 수 있습니다.
- AWS AppSync API에 속도 제한보다 많은 제한 제어가 필요한지 고려하고 필요한 경우 AWS AppSync 엔드포인트 앞에 API Gateway를 구성합니다.
- Amazon SQS 대기열이 Lambda 대기열 소비자를 위한 트리거로 설정된 경우 최대 동시성을 서비스 수준 목표를 충족할 만큼 처리량은 충분하면서 다른 Lambda 함수에 영향을 미치는 동시성 한도를 소비하지 않는 값으로 설정합니다. Lambda에서 대기열을 사용할 때는 동일한 계정 및 리전의 다른 Lambda 함수에 예약된 동시성을 설정하는 것이 좋습니다.
- Amazon SQS 또는 Kinesis에 대한 기본 서비스 통합과 함께 API Gateway를 사용하여 요청을 버퍼링합니다.
- API Gateway를 사용할 수 없는 경우 언어별 라이브러리를 참조하여 워크로드에 토큰 버킷 알고리즘을 구현합니다. 예제 섹션을 확인하고 직접 조사하여 적합한 라이브러리를 찾습니다.

- 설정하려는 제한 또는 증가를 허용하려는 제한을 테스트하고 테스트된 제한을 문서화합니다.
- 테스트에서 설정한 범위를 초과하여 제한을 늘리지 않습니다. 제한을 늘릴 때는 증가를 적용하기 전에 프로비저닝된 리소스가 이미 테스트 시나리오의 리소스와 동일하거나 더 큰지 확인합니다.

## 리소스

관련 모범 사례:

- REL04-BP03 일정한 작업 처리
- REL05-BP03 재시도 직접 호출 제어 및 제한

관련 문서:

- Amazon API Gateway: 처리량 향상을 위해 API 요청 제한
- AWS WAF: Rate-based rule statement
- Introducing maximum concurrency of AWS Lambda when using Amazon SQS as an event source
- AWS Lambda: 최대 동시성

관련 예제:

- The three most important AWS WAF rate-based rules
- Java Bucket4j
- Python token-bucket
- Node token-bucket
- .NET System Threading Rate Limiting

관련 비디오:

- Implementing GraphQL API security best practices with AWS AppSync

관련 도구:

- Amazon API Gateway
- AWS AppSync

- Amazon SQS
- Amazon Kinesis
- AWS WAF
- Virtual Waiting Room on AWS

## REL05-BP03 재시도 직접 호출 제어 및 제한

지수 백오프를 사용하여 각 재시도 간에 점진적으로 더 긴 간격으로 요청을 다시 시도합니다. 재시도 사이에 지터를 도입하여 재시도 간격을 무작위로 지정합니다. 최대 재시도 횟수를 제한합니다.

원하는 성과: 분산 소프트웨어 시스템의 일반적인 구성 요소로는 서버, 로드 밸런서, 데이터베이스, DNS 서버가 있습니다. 이러한 구성 요소는 정상 작동 중에 일시적 또는 제한적인 오류로 또한 재시도에 관계없이 지속되는 오류로 요청에 응답할 수 있습니다. 클라이언트가 서비스에 요청을 전송할 때 요청은 메모리, 스레드, 연결, 포트 또는 기타 제한된 리소스를 포함하여 리소스를 소비합니다. 재시도를 제어하고 제한하는 것은 부하가 걸리는 시스템 구성 요소가 과부하되지 않도록 리소스 소비를 줄이고 최소화하기 위한 전략입니다.

클라이언트는 시간이 초과되거나 오류 응답을 수신하면 재시도 여부를 결정해야 합니다. 재시도할 경우 지터 및 최대 재시도 값이 포함된 지수 백오프가 발생합니다. 그 결과 백엔드 서비스 및 프로세스에서 부하가 감소하고 자가 복구 시간이 단축되어 복구 속도가 빨라지고 요청 처리가 성공적으로 이루어질 수 있습니다.

일반적인 안티 패턴:

- 지수 백오프, 지터, 최대 재시도 값을 추가하지 않고 재시도를 구현합니다. 백오프 및 지터는 의도치 않게 공통 간격으로 조정된 재시도로 인한 인위적인 트래픽 급증을 방지하는 데 도움이 됩니다.
- 효과를 테스트하지 않고 재시도를 구현하거나 재시도 시나리오를 테스트하지 않고 SDK에 재시도가 이미 내장되어 있다고 가정합니다.
- 종속성에서 게시된 오류 코드를 이해하지 못하여 권한 부족을 나타내는 명확한 오류, 구성 오류 또는 수동 개입 없이는 해결되지 않을 것으로 예측되는 기타 조건을 포함하여 모든 오류를 재시도합니다.
- 반복되는 서비스 장애에 대한 모니터링 및 경고를 포함하여 근본적인 문제를 파악하고 해결할 수 있도록 하는 관찰성 관행을 다루지 않습니다.
- 기본 제공 또는 서드파티 재시도 기능이 충분할 때 사용자 지정 재시도 메커니즘을 개발합니다.
- 재시도를 복잡하게 만드는 방식으로 애플리케이션 스택의 여러 계층에서 재시도하여 대량 재시도로 리소스가 더 소모됩니다. 이러한 오류가 애플리케이션의 종속성에 어떤 영향을 미치는지 파악한 다음 한 수준에서만 재시도를 구현해야 합니다.

- 멱등성이 아닌 서비스 직접 호출을 재시도하여 중복 결과 등 예상치 못한 부작용을 초래합니다.

이 모범 사례 확립의 이점: 재시도는 요청이 실패했을 때 클라이언트가 원하는 성과를 얻을 수 있도록 도와주지만, 원하는 응답을 받는 데 서버 시간을 더 많이 소비하기도 합니다. 오류가 드물거나 일시적인 경우 재시도가 유용합니다. 리소스 과부하로 인해 장애가 발생한 경우 재시도는 상황을 악화시킬 수 있습니다. 클라이언트 재시도 시 지터와 함께 지수 백오프를 추가하면 리소스 과부하로 인한 장애 발생 시 서버를 복구할 수 있습니다. 지터는 요청이 집중되어 급증하는 것을 방지하고 백오프는 일반 요청 부하에 재시도를 추가하여 발생하는 부하 에스컬레이션을 줄입니다. 마지막으로, 준안정 장애를 초래하는 백로그가 생성되지 않도록 최대 재시도 횟수 또는 경과 시간을 구성하는 것이 중요합니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

재시도 직접 호출을 제어하고 제한합니다. 지수 백오프를 사용하여 점진적으로 더 긴 간격 후에 다시 시도합니다. 지터를 도입하여 재시도 간격을 무작위로 지정하고 최대 재시도 횟수를 제한합니다.

일부 AWS SDK는 기본적으로 재시도와 지수 백오프를 구현합니다. 해당하는 경우 워크로드에 이러한 기본 제공 AWS 구현을 사용합니다. 멱등성이 있는 서비스를 직접 호출하고 재시도가 클라이언트 가용성을 향상시키는 경우 워크로드에 유사한 로직을 구현합니다. 사용 사례를 기반으로 시간 제한 대상 및 재시도 중단 시점을 결정합니다. 이러한 재시도 사용 사례에 대한 테스트 시나리오를 구축하고 실행합니다.

## 구현 단계

- 애플리케이션 스택에서 애플리케이션이 의존하는 서비스에 대한 재시도를 구현하기 위한 최적의 계층을 결정합니다.
- 선택한 언어에 대해 지수 백오프 및 지터가 포함된 검증된 재시도 전략을 구현하는 기존 SDK를 파악하고 직접 재시도 구현을 작성하는 것보다 이러한 전략을 우선적으로 사용합니다.
- 재시도를 구현하기 전에 서비스가 멱등성을 유지하는지 확인합니다. 재시도를 구현한 후에는 반드시 테스트를 거치고 프로덕션 환경에서 정기적으로 실행해야 합니다.
- AWS 서비스 API를 직접 호출할 때는 AWS SDK 및 AWS CLI를 사용하고 재시도 구성 옵션을 이해합니다. 기본값이 사용 사례에 맞는지 확인하고 필요에 따라 테스트하고 조정합니다.

## 리소스

관련 모범 사례:

- REL04-BP04 변경 작업에 멱등성 부여
- REL05-BP02 요청 제한
- REL05-BP04 빠른 실패 및 대기열 제한
- REL05-BP05 클라이언트 제한 시간 설정
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지

관련 문서:

- Error Retries and Exponential Backoff in AWS
- Amazon Builders' Library: 시간 제한, 재시도 및 지터를 사용한 백오프
- Exponential Backoff and Jitter
- Making retries safe with idempotent APIs

관련 예제:

- Spring Retry
- Resilience4j Retry

관련 비디오:

- Retry, backoff, and jitter: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)

관련 도구:

- AWS SDKs and Tools: Retry behavior
- AWS Command Line Interface: AWS CLI retries

## REL05-BP04 빠른 실패 및 대기열 제한

서비스가 요청에 제대로 응답하지 못하면 빠르게 실패합니다. 이렇게 하면 요청에 연결된 리소스를 해제할 수 있고 리소스가 부족한 경우 서비스 복구를 허용할 수 있습니다. 빠른 실패는 클라우드에서 매우 안정적인 워크로드를 구축하기 위해 활용할 수 있는 잘 정립된 소프트웨어 설계 패턴입니다. 대기열도 잘 정립된 엔터프라이즈 통합 패턴으로, 이를 통해 로드를 원활하게 수행하고 비동기 처리가 허용될 때 클라이언트가 리소스를 해제할 수 있습니다. 서비스가 정상 상태에서는 제대로 응답할 수 있지만 요

청 속도가 너무 높아서 실패하는 경우 대기열을 사용하여 요청을 버퍼링합니다. 하지만 긴 대기열 백로그가 쌓이도록 두지 않습니다. 그러면 클라이언트가 이미 포기한 무효 요청을 처리할 수 있습니다.

원하는 성과: 시스템에서 리소스 경합, 시간 제한, 예외 또는 회색 실패가 발생하여 서비스 수준 목표를 달성할 수 없는 경우 빠른 실패 전략을 통해 시스템 복구 시간을 단축할 수 있습니다. 트래픽 스파이크를 흡수해야 하고 비동기 처리를 수용할 수 있는 시스템은 클라이언트가 대기열을 사용하여 요청을 백엔드 서비스에 버퍼링함으로써 요청을 신속하게 해제할 수 있도록 하여 신뢰성을 높일 수 있습니다. 요청을 대기열로 버퍼링할 때 대처할 수 없는 백로그를 방지하기 위해 대기열 관리 전략이 구현됩니다.

일반적인 안티 패턴:

- 메시지 대기열을 구현하지만 DLQ(Dead Letter Queue) 볼륨에 DLQ 또는 경보를 구성하여 시스템 장애 시점을 감지하지는 않습니다.
- 대기열 소비자가 지연되거나 오류로 인해 재시도되는 시기를 파악하기 위해 대기열에 있는 메시지의 대기 기간을 측정하는 것이 아니라 지연 시간을 측정합니다.
- 업무상 더 이상 필요하지 않아 이러한 메시지를 처리할 가치가 없는 경우 대기열에서 백로깅된 메시지를 지우지 않습니다.
- 후입선출(LIFO) 대기열이 클라이언트 요구 사항을 더 잘 충족할 때 선입선출(FIFO) 대기열을 구성합니다. 예를 들어 엄격한 순서가 필요하지 않고 백로그 처리가 모든 신규 요청과 시간이 중요한 요청을 지연시켜 모든 클라이언트가 서비스 수준 위반을 경험하는 경우입니다.
- 작업 접수를 관리하고 요청을 내부 대기열에 배치하는 API를 노출하는 대신 내부 대기열을 클라이언트에 노출합니다.
- 아주 많은 작업 요청 유형을 단일 대기열에 결합합니다. 그러면 리소스 수요가 요청 유형 전체에 분산되어 백로그 상태가 악화될 수 있습니다.
- 서로 다른 모니터링, 시간 제한, 리소스 할당이 필요함에도 복잡한 요청과 단순한 요청을 동일한 대기열에서 처리합니다.
- 소프트웨어에서 오류를 정상적으로 처리할 수 있는 상위 수준 구성 요소에 예외를 발생시키는 빠른 실패 메커니즘을 구현하기 위해 입력의 유효성을 확인하거나 어설션을 사용하지 않습니다.
- 장애 리소스를 요청 라우팅에서 제거하지 않습니다(특히 장애가 충돌 및 다시 시작, 간헐적인 종속성 장애, 용량 감소 또는 네트워크 패킷 손실로 인한 실패 및 성공을 모두 표시하는 회색인 경우).

이 모범 사례 확립의 이점: 빠르게 실패하는 시스템은 디버깅과 수정이 더 쉬우며 릴리스가 프로덕션 환경에 게시되기 전에 코딩과 구성 문제를 드러내는 경우가 많습니다. 효과적인 대기열 전략이 통합된 시스템은 트래픽 급증 및 간헐적인 시스템 장애 상태에 대한 복원력과 신뢰성을 높입니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

빠른 실패 전략은 소프트웨어 솔루션에 코딩할 수 있을 뿐만 아니라 인프라에 구성할 수도 있습니다. 대기열은 빠른 실패뿐만 아니라 시스템 구성 요소를 분리하여 부하를 원활하게 하는 간단하면서도 강력한 아키텍처 기법입니다. Amazon CloudWatch에서는 장애를 모니터링하고 경보를 알리는 기능을 제공합니다. 시스템에 장애가 발생한 것으로 확인되면 손상된 리소스를 제거하는 등 완화 전략을 실행할 수 있습니다. 시스템에서 Amazon SQS 및 기타 대기열 기술로 대기열을 구현할 때 대기열 백로그와 메시지 소비 실패를 관리하는 방법을 고려해야 합니다.

## 구현 단계

- 소프트웨어에 프로그래밍 방식 어설션 또는 특정 지표를 구현하고 이를 사용하여 시스템 문제를 명시적으로 경고합니다. Amazon CloudWatch를 사용하면 애플리케이션 로그 패턴 및 SDK 계측을 기반으로 지표와 경보를 생성할 수 있습니다.
- CloudWatch 지표 및 경보를 사용하여 처리 지연 시간을 늘리거나 반복적으로 요청 처리를 실패하는 손상된 리소스를 차단합니다.
- 백엔드 대기열 소비자가 요청을 처리하는 동안 클라이언트가 리소스를 해제하고 다른 작업을 계속할 수 있도록 요청을 수락하고 Amazon SQS를 사용하여 내부 대기열에 요청을 추가한 다음 메시지 생성 클라이언트에 성공 메시지로 응답하도록 API를 설계하여 비동기 처리를 사용합니다.
- 현재 시간과 메시지 타임스탬프를 비교해 대기열에서 메시지를 제거할 때마다 CloudWatch 지표를 생성하여 대기열 처리 대기 시간을 측정하고 모니터링합니다.
- 장애가 발생하여 메시지 처리가 실패했거나 서비스 수준에 관한 계약 내에서 처리할 수 없는 트래픽 스파이크가 발생하는 경우 오래된 트래픽이나 초과 트래픽을 스필오버 대기열로 넘깁니다. 이를 통해 새 작업을 우선적으로 처리하고 용량이 사용 가능한 경우 이전 작업을 우선적으로 처리할 수 있습니다. 이 기법은 LIFO 처리와 유사하며 모든 새 작업에 대해 정상적인 시스템 처리가 가능합니다.
- DLQ(Dead Letter Queue) 또는 재구동 대기열을 사용하여 백로그에서 처리할 수 없는 메시지를 나중에 조사하고 해결할 수 있는 위치로 옮깁니다.
- 이전 메시지를 재시도하거나 허용되는 경우 현재 시간을 메시지 타임스탬프와 비교하고 더 이상 요청 클라이언트와 관련이 없는 메시지를 삭제합니다.

## 리소스

관련 모범 사례:

- REL04-BP02 느슨하게 결합된 종속성 구현
- REL05-BP02 요청 제한

- REL05-BP03 재시도 직접 호출 제어 및 제한
- REL06-BP02 지표 정의 및 계산(집계)
- REL06-BP07 시스템을 통한 요청의 엔드 투 엔드 추적 모니터링

관련 문서:

- 심각한 수준의 대기열 백로그 방지
- Fail Fast
- 내 Amazon SQS 대기열에서 메시지 백로그가 증가하는 것을 방지하려면 어떻게 해야 하나요?
- Elastic Load Balancing: Zonal Shift
- Amazon Application Recovery Controller: 트래픽 장애 조치에 대한 라우팅 컨트롤

관련 예제:

- Enterprise Integration Patterns: Dead Letter Channel

관련 비디오:

- AWS re:Invent 2022 - Operating highly available Multi-AZ applications

관련 도구:

- Amazon SQS
- Amazon MQ
- AWS IoT Core
- Amazon CloudWatch

## REL05-BP05 클라이언트 제한 시간 설정

연결 및 요청에 대한 시간 제한을 적절하게 설정하고 체계적으로 확인합니다. 워크로드 세부 사항을 인식하지 못하므로 기본값에 의존하지 않습니다.

원하는 성과: 클라이언트 시간 제한은 완료에 비정상적으로 많은 시간이 걸리는 요청 대기와 관련된 클라이언트, 서버, 워크로드의 비용을 고려해야 합니다. 시간 제한의 정확한 원인을 알 수 없기 때문에 클

라이언트는 서비스에 대한 지식을 활용하여 생각되는 원인과 적절한 시간 제한에 대한 기대치를 세워야 합니다.

구성된 값에 따라 클라이언트 연결이 시간 제한됩니다. 시간 제한이 발생한 후 클라이언트는 작업을 중단하고 재시도 또는 회로 차단기 개방을 결정합니다. 이러한 패턴에서는 근본적인 오류 상태를 악화시킬 수 있는 요청 발행이 방지됩니다.

일반적인 안티 패턴:

- 시스템 시간 제한 또는 기본 시간 제한을 인식하지 못합니다.
- 정상적인 요청 완료 타이밍을 인식하지 못합니다.
- 요청을 완료하는 데 비정상적으로 오래 걸리는 원인 또는 이러한 완료를 기다리는 데 따른 클라이언트, 서비스 또는 워크로드 성능의 비용을 인식하지 못합니다.
- 네트워크가 손상되어 시간 제한에 도달한 후에만 요청이 실패할 확률과 시간 제한을 더 짧게 설정하지 않아 클라이언트와 워크로드에 발생할 수 있는 비용을 인식하지 못합니다.
- 연결 및 요청 모두에 대한 시간 제한 시나리오를 테스트하지 않습니다.
- 시간 제한을 매우 높게 설정합니다. 그러면 지연 시간이 길어지고 리소스 사용률이 증가할 수 있습니다.
- 시간 제한을 매우 낮게 설정합니다. 그러면 인위적인 오류가 발생합니다.
- 회로 차단기 및 재시도와 같은 원격 직접 호출의 시간 제한 오류를 처리하기 위한 패턴을 간과합니다.
- 서비스 직접 호출 오류율, 지연 시간에 대한 서비스 수준 목표, 지연 시간 이상치에 대한 모니터링을 고려하지 않습니다. 이러한 지표를 통해 공격적이거나 허용되는 시간 제한의 인사이트를 얻을 수 있습니다.

이 모범 사례 확립의 이점:: 원격 직접 호출 시간 제한이 구성되고 시스템이 시간 제한을 정상적으로 처리하도록 설계되어 원격 직접 호출이 비정상적으로 느리게 응답하고 서비스 클라이언트에서 시간 제한 오류를 정상적으로 처리할 때 리소스가 절약됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

서비스 종속성 직접 호출과 일반적으로 프로세스 전체의 모든 직접 호출에 연결 시간 제한과 요청 시간 제한을 모두 설정합니다. 많은 프레임워크가 시간 제한 기능을 기본 제공하지만 일부 프레임워크에는 서비스 목표에 허용되는 것보다 높거나 무한한 기본값이 있으므로 주의해야 합니다. 값이 너무 높으면 클라이언트가 시간 제한이 발생할 때까지 대기하는 동안 리소스가 계속 소비되기 때문에 시간 제한의

유용성이 감소합니다. 값이 너무 낮으면 백엔드에서 트래픽이 증가하고 너무 많은 요청이 재시도되므로 지연 시간이 증가할 수 있습니다. 일부 경우에는 모든 요청이 재시도되기 때문에 이로 인해 전체가 중단될 수 있습니다.

시간 제한 전략을 결정할 때는 다음 사항을 고려하세요.

- 요청의 내용, 대상 서비스의 장애 또는 네트워킹 파티션 장애로 인해 요청을 처리하는 데 평소보다 시간이 오래 걸릴 수 있습니다.
- 비정상적으로 비용이 많이 드는 콘텐츠를 요청하면 불필요한 서버 및 클라이언트 리소스가 소모될 수 있습니다. 이 경우 이러한 요청을 시간 초과시키고 재시도하지 않는 것이 리소스를 보존할 수 있습니다. 또한 서비스는 제한 및 서버 측 시간 제한으로 비정상적으로 비용이 많이 드는 콘텐츠로부터 스스로를 보호해야 합니다.
- 서비스 장애로 인해 비정상적으로 오래 걸리는 요청은 시간 제한 후 재시도할 수 있습니다. 요청 및 재시도에 따른 서비스 비용을 고려해야 하지만, 원인이 국소적인 장애인 경우 재시도는 비용이 많이 들지 않으며 클라이언트 리소스 소비를 줄일 수 있습니다. 장애의 특성에 따라 시간 제한으로 인해 서버 리소스가 해제될 수도 있습니다.
- 네트워크에서 요청 또는 응답을 전달하지 못해 완료하는 데 시간이 오래 걸리는 요청은 시간 초과 후 재시도할 수 있습니다. 요청 또는 응답이 전달되지 않았으므로 시간 제한의 길이와 상관없이 실패했을 것입니다. 이 경우 시간 초과로 인해 서버 리소스가 해제되지는 않지만 클라이언트 리소스가 해제되고 워크로드 성능이 향상됩니다.

재시도 및 회로 차단기와 같이 잘 정립된 설계 패턴을 활용하여 시간 제한을 원활하게 처리하고 빠른 실패 접근 방식을 지원할 수 있습니다. AWS SDK 및 AWS CLI는 연결 및 요청 시간 제한을 모두 구성하고 지수 백오프 및 지터를 통한 재시도를 구성할 수 있습니다. AWS Lambda 함수는 시간 제한 구성을 지원하고 AWS Step Functions에서는 AWS 서비스 및 SDK와의 사전 구축된 통합을 활용하는 로우코드 회로 차단기를 구축할 수 있습니다. AWS App Mesh Envoy는 시간 제한 및 회로 차단기 기능을 제공합니다.

## 구현 단계

- 원격 서비스 직접 호출에 대한 시간 제한을 구성하고 기본 제공되는 언어 시간 제한 기능 또는 오픈소스 시간 제한 라이브러리를 활용합니다.
- 워크로드가 AWS SDK를 사용하여 호출하는 경우 설명서에서 언어별 시간 제한 구성을 검토합니다.
  - Python
  - PHP
  - .NET

  - Ruby
  - Java
  - Go
  - Node.js
  - C++
- 워크로드에서 AWS SDK 또는 AWS CLI 명령을 사용하는 경우 `connectTimeoutInMillis` 및 `tlsNegotiationTimeoutInMillis`에 대한 AWS 구성 기본값을 설정하여 기본 시간 제한을 구성합니다.
- 명령줄 옵션 `cli-connect-timeout` 및 `cli-read-timeout`을 적용하여 AWS 서비스에 대한 일회성 AWS CLI 명령을 제어합니다.
- 오류 시나리오를 사전에 처리할 수 있도록 원격 서비스 직접 호출의 시간 제한을 모니터링하고 지속적인 오류에 대한 경고를 설정합니다.
- 직접 호출 오류율, 대기 시간에 대한 서비스 수준 목표, 대기 시간 이상값에 대한 CloudWatch 지표 및 CloudWatch 이상 탐지를 구현하여 과도하게 공격적이거나 허용적인 시간 제한 관리의 인사이트를 얻습니다.
- Lambda 함수에서 시간 제한을 구성합니다.
- API Gateway 클라이언트는 시간 제한을 처리할 때 자체 재시도를 구현해야 합니다. API Gateway는 다운스트림 통합에 대해 50밀리초~29초의 통합 시간 제한을 지원하고 통합 요청의 제한 시간이 초과되면 재시도하지 않습니다.
- 제한 시간을 초과할 때 원격 직접 호출을 방지하기 위해 회로 차단기 패턴을 구현합니다. 직접 호출이 실패하지 않도록 회로를 개방하고 직접 호출이 정상적으로 응답할 때 회로를 폐쇄합니다.
- 컨테이너 기반 워크로드의 경우 기본 제공 시간 제한 및 회로 차단기를 활용하는 App Mesh Envoy 기능을 검토하세요.
- 특히 AWS Step Functions 기본 SDK 및 지원되는 Step Functions 통합을 ㅈ기접 호출하여 워크로드를 간소화하는 경우 AWS를 사용하여 원격 서비스 직접 호출을 위한 로우 코드 회로 차단기를 구축합니다.

## 리소스

관련 모범 사례:

- REL05-BP03 재시도 직접 호출 제어 및 제한
- REL05-BP04 빠른 실패 및 대기열 제한

- REL06-BP07 시스템을 통한 요청의 엔드 투 엔드 추적 모니터링

관련 문서:

- AWS SDK: Retries and Timeouts
- Amazon Builders' Library: 시간 제한, 재시도 및 지터를 사용한 백오프
- Amazon API Gateway 할당량 및 중요 정보
- AWS Command Line Interface: Command line options
- AWS SDK for Java 2.x: Configure API Timeouts
- AWS Botocore using the config object and Config Reference
- AWS SDK for .NET: 재시도 및 제한 시간
- AWS Lambda: Lambda 함수 옵션 구성

관련 예제:

- Using the circuit breaker pattern with AWS Step Functions and Amazon DynamoDB
- Martin Fowler: CircuitBreaker

관련 도구:

- AWS SDKs
- AWS Lambda
- Amazon SQS
- AWS Step Functions
- AWS Command Line Interface

## REL05-BP06 가능한 경우 시스템을 상태 비저장으로 설계

시스템은 상태를 필요로 하지 않거나, 서로 다른 클라이언트 요청 간에 디스크 및 메모리에 로컬로 저장된 데이터에 종속성이 없도록 상태를 오프로드해야 합니다. 이렇게 하면 가용성에 미치는 영향 없이 서버를 대체할 수 있습니다.

사용자 또는 서비스는 애플리케이션과 상호 작용할 때 세션을 구성하는 일련의 상호 작용을 수행하는 경우가 많습니다. 세션은 애플리케이션을 사용하는 동안 요청 간에 유지되는 사용자의 고유한 데이터

입니다. 상태 비저장 애플리케이션은 이전 상호 작용에 대한 지식을 필요로 하지 않으며 세션 정보를 저장하지 않는 애플리케이션입니다.

상태 비저장으로 설계된 경우 AWS Lambda 또는 AWS Fargate와 같은 서버리스 컴퓨팅 서비스를 사용할 수 있습니다.

서버 대체 외에 상태 비저장 애플리케이션의 또 다른 이점은 사용 가능한 컴퓨팅 리소스(예: EC2 인스턴스 및 AWS Lambda 함수)로 모든 요청을 처리할 수 있기 때문에 수평적 스케일링이 가능합니다.

이 모범 사례 확립의 이점: 상태 비저장으로 설계된 시스템은 수평적 스케일링에 더 잘 적응하므로 변동하는 트래픽과 수요에 따라 용량을 추가하거나 제거할 수 있습니다. 또한 기본적으로 장애에 대한 복원력이 뛰어나며 애플리케이션 개발에 유연성과 민첩성을 제공합니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

애플리케이션을 상태 비저장으로 설계합니다. 상태 비저장 애플리케이션은 수평적 스케일링을 가능하게 하며, 개별 노드의 장애에 대한 내결함성이 있습니다. 아키텍처 내에서 상태를 유지하는 애플리케이션 구성 요소를 분석하고 이해합니다. 이를 통해 상태 비저장 설계로의 전환이 미칠 수 있는 영향을 평가할 수 있습니다. 상태 비저장 아키텍처는 사용자 데이터를 분리하고 세션 데이터를 오프로드합니다. 이를 통해 각 구성 요소를 독립적으로 규모 조정하여 다양한 워크로드 수요를 충족하고 리소스 활용도를 최적화할 수 있는 유연성을 제공합니다.

구현 단계

- 애플리케이션의 상태 저장 구성 요소를 식별하고 이해합니다.
- 핵심 애플리케이션 논리에서 사용자 데이터를 분리 및 관리하여 데이터를 분리합니다.
  - Amazon Cognito는 자격 증명 풀, 사용자 풀 및 Amazon Cognito Sync 등의 기능을 사용하여 애플리케이션 코드에서 사용자 데이터를 분리할 수 있습니다.
  - AWS Secrets Manager를 사용하면 보안 암호를 안전한 중앙 위치에 저장하여 사용자 데이터를 분리할 수 있습니다. 즉, 애플리케이션 코드에 보안 암호를 저장할 필요가 없으므로 보안이 강화됩니다.
  - 이미지 및 문서와 같은 대용량의 비정형 데이터를 저장하는 데 Amazon S3 사용을 고려합니다. 애플리케이션은 필요할 때 이 데이터를 검색할 수 있으므로 데이터를 메모리에 저장할 필요가 없습니다.
  - Amazon DynamoDB를 사용하여 사용자 프로필과 같은 정보를 저장합니다. 애플리케이션은 이 데이터를 거의 실시간으로 쿼리할 수 있습니다.

- 세션 데이터를 데이터베이스, 캐시 또는 외부 파일로 오프로드합니다.
  - Amazon ElastiCache, Amazon DynamoDB, Amazon Elastic File System (Amazon EFS), Amazon MemoryDB는 세션 데이터를 오프로드하는 데 사용할 수 있는 AWS 서비스의 예입니다.
- 선택한 스토리지 솔루션에 어떤 상태 및 사용자 데이터를 유지해야 하는지 파악한 후 상태 비저장 아키텍처를 설계합니다.

## 리소스

관련 모범 사례:

- REL11-BP03 모든 계층에서 복구 자동화

관련 문서:

- Amazon Builders' Library: 분산 시스템의 폴백 방지
- Amazon Builders' Library: 심각한 수준의 대기열 백로그 방지
- Amazon Builders' Library: 캐싱 관련 당면 과제 및 전략
- AWS에서 상태 비저장 웹 티어 모범 사례

# REL05-BP07 비상 레버 구현

비상 레버는 워크로드의 가용성에 미치는 영향을 신속하게 완화할 수 있는 프로세스입니다.

비상 레버는 알려진 메커니즘과 테스트를 거친 메커니즘을 사용하여 구성 요소 또는 종속성의 동작을 비활성화, 제한 또는 변경하는 방식으로 작동합니다. 이를 통해 예상치 못한 수요 증가로 인한 리소스 고갈이 유발하는 워크로드 장애를 완화하고 워크로드 내 비핵심 구성 요소에 장애가 미치는 영향을 줄일 수 있습니다.

원하는 성과: 비상 레버를 구현하면 바람직한 사례로 알려진 프로세스를 구축하여 워크로드에서 핵심 구성 요소의 가용성을 유지할 수 있습니다. 비상 레버를 작동시키는 동안에도 워크로드는 단계적으로 줄어들고 비즈니스의 핵심 기능을 계속 수행해야 합니다. 단계적 성능 저하에 대한 자세한 내용은 REL05-BP01 관련 하드 종속성을 소프트 종속성으로 변환하는 단계적 성능 저하 구현을 참조하세요.

일반적인 안티 패턴:

- 비핵심 종속성의 장애가 핵심 워크로드의 가용성에 영향을 미칩니다.
- 비핵심 구성 요소가 손상되었을 때 핵심 구성 요소 동작을 테스트하거나 확인하지 않습니다.

- 비상 레버의 활성화 또는 비활성화에 대한 명확하고 결정적인 기준이 정의되어 있지 않습니다.

이 모범 사례 확립의 이점: 비상 레버를 구현하면 해석기에 확립된 프로세스를 제공하여 예상치 못한 수요 급증 또는 비핵심 종속성 장애에 대응하도록 함으로써 워크로드에서 핵심 구성 요소의 가용성을 높일 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

- 워크로드의 핵심 구성 요소를 식별합니다.
- 비핵심 구성 요소의 장애를 견딜 수 있도록 워크로드의 핵심 구성 요소를 설계하고 설계합니다.
- 테스트를 수행하여 비핵심 구성 요소에 장애가 발생한 경우 핵심 구성 요소의 동작을 검증합니다.
- 관련 지표 또는 트리거를 정의하고 모니터링하여 비상 레버 절차를 시작합니다.
- 비상 레버를 구성하는 절차를 수동 또는 자동으로 정의합니다.

### 구현 단계

- 워크로드에서 비즈니스의 핵심 구성 요소를 식별합니다.
  - 워크로드의 각 기술 구성 요소를 관련 비즈니스 기능에 매핑하고 핵심 또는 비핵심으로 평가해야 합니다. Amazon의 핵심 기능과 비핵심 기능의 예를 보려면 Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second를 참조하세요.
  - 이런 과정은 기술 의사 결정이자 동시에 비즈니스 의사 결정이며 조직과 워크로드에 따라 다릅니다.
- 비핵심 구성 요소의 장애를 견딜 수 있도록 워크로드의 핵심 구성 요소를 설계하고 설계합니다.
  - 종속성 분석 중에 잠재적 장애 모드를 모두 고려하고 비상 레버 메커니즘이 다운스트림 구성 요소에 핵심 기능을 제공하는지 확인합니다.
- 비상 레버를 작동시키는 동안 핵심 구성 요소의 동작을 검증하기 위한 테스트를 실시합니다.
  - 바이모달 동작은 사용하지 마세요. 자세한 내용은 REL11-BP05 정적 안정성을 사용하여 바이모달 동작 방지를 참조하세요.
- 관련 지표를 정의 및 모니터링하고 알림을 설정하여 비상 레버 절차를 시작합니다.
  - 모니터링할 올바른 지표를 찾는 방법은 워크로드에 따라 다릅니다. 지표의 예로는 지연 시간 또는 종속성에 대한 요청 실패 횟수가 있습니다.
- 비상 레버를 구성하는 절차를 수동 또는 자동으로 정의합니다.

- 여기에는 로드 감소, 요청 제한 또는 단계적 성능 저하 구현과 같은 메커니즘이 포함될 수 있습니다.

## 리소스

관련 모범 사례:

- REL05-BP01 관련 하드 종속성을 소프트 종속성으로 변환하는 단계적 성능 저하 구현
- REL05-BP02 요청 제한
- REL11-BP05 정적 안정성을 사용하여 바이모달 동작 방지

관련 문서:

- 안전하고 간편한 배포 자동화
- Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second

관련 비디오:

- AWS re:Invent 2020: Reliability, consistency, and confidence through immutability

# 변경 관리

워크로드의 신뢰할 수 있는 운영을 위해서는 워크로드 또는 환경에 대한 변경을 예상하고 수용해야 합니다. 변경에는 수요 급증과 같이 워크로드에 적용되는 변경은 물론 기능 배포 및 보안 패치와 같은 워크로드 내부의 변경이 포함됩니다.

다음 섹션에서는 변경 관리에 대한 모범 사례를 설명합니다.

주제

- 워크로드 리소스 모니터링
- 수요 변경에 따라 조정되는 워크로드 설계
- 변경 구현

## 워크로드 리소스 모니터링

로그와 지표는 워크로드 상태를 파악할 수 있는 강력한 도구입니다. 로그와 지표를 모니터링하고 임계값을 초과하거나 중요한 이벤트가 발생할 경우 알림을 보내도록 워크로드를 구성할 수 있습니다. 모니터링을 수행하면 워크로드가 저성능 임곗값을 초과하거나 장애가 발생할 때를 인식하고 이에 대응하여 자동으로 복구할 수 있습니다.

가용성 요구 사항이 충족되려면 모니터링이 중요합니다. 모니터링을 통해 장애를 잘 감지해야합니다. 최악의 장애 형태는 설정된 기능이 더 이상 작동하지 않아 '알림이 되지 않는' 오류이지만 간접적인 방법 이외에는 이를 감지할 다른 방법은 없습니다. 이렇게 되면 오류가 감지되기 전에 고객이 먼저 영향을 받습니다. 모니터링의 주된 이유 중 하나는 문제 발생을 알리기 위해서입니다. 이때 시스템을 알림에서 최대한 분리해야 합니다. 서비스 중단으로 인해 알림 기능이 제거되면 중단 시간은 더 길어집니다.

AWS는 여러 수준에서 애플리케이션을 계측합니다. 그리고 계측 프로세스 내에서 모든 종속성과 주요 작업에 대해 각 요청의 지연 시간, 오류율 및 가용성을 기록합니다. 그리고 성공한 작업의 지표도 기록합니다. 그러면 곧 발생할 것으로 예상되는 문제를 발생 전에 파악할 수 있습니다. 저희는 단순히 평균 지연 시간만 고려하지 않습니다. 지연 시간 이상값을 더 집중적으로 살핍니다. 예를 들어 99.9번째 백분위수와 99.99번째 백분위수를 살핍니다. 1,000개 또는 10,000개 요청 중에서 요청이 하나만 느려져도 고객 경험이 영향을 받기 때문입니다. 평균은 적정 수준이지만 요청 100건 중 하나의 지연 시간이 매우 길다면 결국 트래픽이 증가하면서 문제가 됩니다.

AWS에서 모니터링은 다음의 4개의 고유한 단계로 구성됩니다.

1. 생성 – 워크로드에 대한 모든 구성 요소 모니터링
2. 집계 – 지표 정의 및 계산
3. 실시간 처리 및 경보 – 알림 전송 및 대응 자동화
4. 저장 및 분석

모범 사례

- REL06-BP01 워크로드의 모든 구성 요소 모니터링(생성)
- REL06-BP02 지표 정의 및 계산(집계)
- REL06-BP03 알림 전송(실시간 처리 및 경보)
- REL06-BP04 응답 자동화(실시간 처리 및 경보)
- REL06-BP05 로그 분석
- REL06-BP06 모니터링 범위 및 지표를 정기적으로 검토
- REL06-BP07 시스템을 통한 요청의 엔드 투 엔드 추적 모니터링

## REL06-BP01 워크로드의 모든 구성 요소 모니터링(생성)

Amazon CloudWatch 또는 서드파티 도구를 사용하여 워크로드의 구성 요소를 모니터링합니다. AWS Health 대시보드에서 AWS 서비스를 모니터링합니다.

프론트엔드, 비즈니스 로직 및 스토리지 계층을 포함하여 워크로드의 모든 구성 요소를 모니터링해야 합니다. 주요 지표를 정의하고, 필요한 경우 로그에서 주요 지표를 추출하는 방법을 정의하며, 해당 경보 이벤트를 간접 호출하는 임곗값을 설정합니다. 지표가 워크로드의 핵심 성과 지표(KPI)와 연관이 있도록 해야 하며 지표와 로그를 사용하여 서비스 성능 저하의 조기 지표를 파악합니다. 예를 들어, 분당 성공적으로 처리된 주문 수와 같은 비즈니스 성과와 관련된 지표는 CPU 사용량과 같은 기술적 지표보다 워크로드 문제를 더 빠르게 알려줍니다. AWS 리소스의 기반이 되는 AWS 서비스의 성능 및 가용성에 대한 맞춤형 보기를 보려면 AWS Health Dashboard를 사용합니다.

클라우드에서 모니터링은 새로운 기회를 제공합니다. 대부분의 클라우드 공급업체는 사용자 지정 가능한 후크를 개발했으며 여러 계층의 워크로드를 모니터링하는 데 도움이 되는 인사이트를 제공할 수 있습니다. Amazon CloudWatch와 같은 AWS 서비스는 통계 및 기계 학습 알고리즘을 적용하여 시스템 및 애플리케이션의 지표를 지속적으로 분석하고, 일반적인 기준을 결정하며, 사용자의 개입을 최소화하면서 이상 현상을 알립니다. 이상 탐지 알고리즘은 지표의 계절성 및 추세 변화를 설명합니다.

AWS는 사용할 수 있는 모니터링 및 로그 정보를 풍부하게 제공하며, 이를 통해 사용자는 워크로드별 지표를 정의하고, 수요 변화가 있는 프로세스를 정의하며, ML 전문성과 상관없이 기계 학습 기법을 도입하는 데 이 정보를 사용할 수 있습니다.

또한 모든 외부 엔드포인트를 모니터링하여 기본 구현과 독립되어 있는지 확인합니다. 이 능동적 모니터링은 가상 트랜잭션에서도 수행할 수 있습니다(사용자 canary라고도 함, 단, 카나리 배포와 혼동하지 않도록 주의). 그러면 워크로드의 클라이언트가 수행하는 작업에 부합하는 일반적인 여러 작업을 주기적으로 실행합니다. 작업 기간은 짧아야 하며 테스트 중에 워크로드에 과부하가 발생하지 않아야 합니다. Amazon CloudWatch Synthetics를 사용하면 엔드포인트 및 API 모니터링을 위한 가상 canary를 생성할 수 있습니다. 가상 Canary 클라이언트 노드를 AWS X-Ray 콘솔과 함께 사용하여 선택한 기간에 오류, 장애 또는 조절 속도 문제를 경험하는 가상 Canary를 식별할 수도 있습니다.

원하는 성과:

워크로드의 모든 구성 요소로부터 핵심적인 지표를 수집하고 사용하여 워크로드 신뢰성과 최적의 사용자 경험을 보장합니다. 워크로드가 비즈니스 성과를 달성하지 못하고 있음을 탐지하면 재해 상황임을 빠르게 선언하고 인시던트에서 복구할 수 있습니다.

일반적인 안티 패턴:

- 워크로드에 대한 외부 인터페이스만 모니터링합니다.
- 워크로드별 지표를 생성하지 않고 워크로드가 사용하는 AWS 서비스에서 제공하는 지표에만 의존합니다.
- 워크로드에서 기술적 지표만 사용하고 워크로드가 기여하는 비기술적 KPI와 관련된 지표는 모니터링하지 않습니다.
- 프로덕션 트래픽과 간단한 상태 확인에 의존하여 워크로드 상태를 모니터링하고 평가합니다.

이 모범 사례 확립의 이점: 워크로드의 모든 티어에서 모니터링할 경우 워크로드를 구성하는 구성 요소의 문제를 보다 신속하게 예측하고 해결할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

1. 가능한 경우 로깅을 활성화합니다. 워크로드의 모든 구성 요소에서 모니터링 데이터를 가져와야 합니다. S3 액세스 로그와 같은 추가 로깅을 활성화하고 워크로드에서 워크로드별 데이터를 로깅하도록 허용합니다. Amazon ECS, Amazon EKS, Amazon EC2, Elastic Load Balancing, AWS Auto Scaling, Amazon EMR과 같은 서비스에서 CPU, 네트워크 I/O 및 디스크 I/O 평균에 대한 지표를 수

집합니다. CloudWatch에 지표를 게시하는 AWS 서비스 목록은 CloudWatch 지표를 게시하는 AWS 서비스를 참조하세요.

2. 모든 기본 지표를 검토하고 데이터 수집 격차를 살펴봅니다. 모든 서비스에서는 기본 지표를 생성합니다. 기본 지표를 수집하면 워크로드 구성 요소 간 종속성과 구성 요소 신뢰성 및 성능이 워크로드에 미치는 영향을 더 잘 이해할 수 있습니다. 또한 AWS CLI 또는 API를 사용하여 자체 지표를 생성해 CloudWatch에 자체 지표를 게시할 수 있습니다.
3. 모든 지표를 평가하여 워크로드의 각 AWS 서비스에 대해 어떤 지표를 경고할지 결정합니다. 워크로드 신뢰성에 큰 영향을 미치는 지표의 하위 세트를 선택할 수 있습니다. 핵심 지표와 임곗값에 집중하면 알림의 수를 세분화하고 오탐지를 최소화할 수 있습니다.
4. 알림을 정의하고 알림이 간접 호출된 후 워크로드에 대한 복구 프로세스를 정의합니다. 알림을 정의하면 인시던트로부터 복구하는 데 필요한 단계를 빠르게 알리고, 에스컬레이션하며, 단계를 따라 사전에 정해진 Recovery Time Objective(RTO)를 달성할 수 있습니다. Amazon CloudWatch 경보를 사용하여 자동화된 워크플로를 간접 호출하고 정의된 임곗값에 따라 복구 절차를 시작할 수 있습니다.
5. 워크로드 상태에 대한 관련 데이터를 수집하기 위해 가상 트랜잭션 사용 방법을 알아봅니다. 가상 트랜잭션은 동일한 경로를 따라 고객과 동일한 작업을 수행하므로 워크로드에 고객 트래픽이 없는 경우에도 고객 경험을 지속적으로 확인할 수 있습니다. 가상 트랜잭션을 사용하면 고객보다 먼저 문제를 발견할 수 있습니다.

## 리소스

관련 모범 사례:

- REL11-BP03 모든 계층에서 복구 자동화

관련 문서:

- Getting started with your AWS Health Dashboard – Your account health
- CloudWatch 지표를 게시하는 AWS 서비스
- Access Logs for Your Network Load Balancer
- Access logs for your application load balancer
- AWS Lambda에 대한 Amazon CloudWatch logs 액세스
- Amazon S3 서버 액세스 로깅
- Classic Load Balancer 액세스 로그 활성화
- Exporting log data to Amazon S3

- Amazon EC2 인스턴스에 CloudWatch 에이전트 설치
- 사용자 지정 지표 게시
- Amazon CloudWatch 대시보드 사용
- Amazon CloudWatch 지표 사용
- Using Canaries (Amazon CloudWatch Synthetics)
- What are Amazon CloudWatch Logs?

사용 설명서:

- 추적 생성
- Amazon EC2 Linux 인스턴스의 메모리 및 디스크 지표 모니터링
- 컨테이너 인스턴스와 함께 CloudWatch Logs 사용
- VPC 흐름 로그
- What is Amazon DevOps Guru?
- AWS X-Ray란 무엇입니까?

관련 블로그:

- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray

관련 예제:

- Amazon Builders' Library: 운영 가시성을 위한 분산 시스템 계측
- One Observability 워크숍

## REL06-BP02 지표 정의 및 계산(집계)

워크로드 구성 요소에서 지표와 로그를 수집하고 관련 집계 지표를 계산합니다. 이러한 지표를 통해 워크로드를 광범위하고 심층적으로 관찰할 수 있으며 복원 태세를 크게 개선할 수 있습니다.

관찰성은 단순히 워크로드 구성 요소에서 지표를 수집하고 이를 보고 관련 알림을 보내는 데 그치지 않습니다. 워크로드의 동작에 대해 전체적으로 이해하는 것입니다. 이 동작 정보는 워크로드의 모든 구성 요소에서 가져옵니다. 여기에는 워크로드가 의존하는 클라우드 서비스, 잘 작성된 로그 및 지표가 포함됩니다. 이 데이터를 통해 워크로드의 전체 동작을 감독할 수 있을 뿐만 아니라 모든 구성 요소와 모든 작업 단위 간의 상호 작용을 세부적으로 파악할 수 있습니다.

원하는 성과:

- 워크로드 구성 요소 및 AWS 서비스 종속성에서 로그를 수집하여 쉽게 액세스하고 처리할 수 있는 중앙 위치에 게시합니다.
- 로그에는 충실도가 높고 정확한 타임스탬프가 포함되어 있습니다.
- 로그에는 추적 식별자, 사용자 또는 계정 식별자, 원격 IP 주소와 같은 처리 컨텍스트에 대한 관련 정보가 포함되어 있습니다.
- 개괄적인 관점에서 워크로드의 동작을 나타내는 집계 지표를 로그에서 생성합니다.
- 집계된 로그를 쿼리하여 워크로드에 대한 심층적이고 관련 있는 인사이트를 얻고 실제 및 잠재적 문제를 식별할 수 있습니다.

일반적인 안티 패턴:

- 워크로드가 실행되는 컴퓨팅 인스턴스 또는 워크로드가 사용하는 클라우드 서비스에서 관련 로그 또는 지표를 수집하지 않습니다.
- 비즈니스 핵심 성과 지표(KPI)와 관련된 로그로그 및 지표 수집을 간과합니다.
- 집계 및 상관관계 없이 워크로드 관련 원격 측정을 개별적으로 분석합니다.
- 지표와 로그가 너무 빨리 만료되도록 허용하여 추세 분석과 반복되는 문제 식별이 방해됩니다.

이러한 모범 사례 확립의 이점: 더 많은 이상을 감지하고 워크로드의 다양한 구성 요소 간에 이벤트와 지표의 상관관계를 파악할 수 있습니다. 지표만으로는 파악하기 힘든 경우가 많은, 로그에 포함된 정보를 기반으로 워크로드 구성 요소에서 인사이트를 생성할 수 있습니다. 대규모로 로그를 쿼리하여 장애 원인을 더 빠르게 확인할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

워크로드 및 해당 구성 요소와 관련된 원격 측정 데이터의 소스를 식별합니다. 이 데이터는 운영 체제(OS) 및 Java 등의 애플리케이션 런타임과 같은 지표를 게시하는 구성 요소뿐만 아니라 애플리케이션 및 클라우드 서비스 로그에서도 제공됩니다. 예를 들어 웹 서버는 일반적으로 타임스탬프, 처리 지연 시간, 사용자 ID, 원격 IP 주소, 경로 및 쿼리 문자열과 같은 세부 정보와 함께 각 요청을 기록합니다. 이러한 로그의 세부 정보 수준은 세부 쿼리를 수행하고 다른 방법으로는 제공되지 않는 지표를 생성하는 데 도움이 됩니다.

적절한 도구와 프로세스를 사용하여 지표와 로그를 수집합니다. Amazon EC2 인스턴스에서 실행되는 애플리케이션에서 생성된 로그는 Amazon CloudWatch Agent와 같은 에이전트가 수집하여 Amazon

CloudWatch Logs와 같은 중앙 스토리지 서비스에 게시될 수 있습니다. AWS Lambda 및 Amazon Elastic Container Service와 같은 AWS 관리형 컴퓨팅 서비스는 자동으로 CloudWatch Logs에 로그를 게시합니다. 워크로드에서 사용하는 Amazon CloudFront, Amazon S3, Elastic Load Balancing 및 Amazon API Gateway와 같은 AWS 스토리지 및 처리 서비스에 대한 로그 수집을 활성화합니다.

동작 패턴을 더 명확하게 보고 관련 구성 요소 그룹에 상관관계가 있는 문제를 격리하는 데 도움이 되는 차원으로 원격 측정 데이터를 강화합니다. 추가한 후에는 구성 요소 동작을 더 세부적으로 관찰하고, 상관관계가 있는 장애를 감지하고, 적절한 수정 조치를 취할 수 있습니다. 유용한 차원의 예로는 가용 영역, EC2 인스턴스 ID, 컨테이너 작업 또는 포드 ID가 있습니다.

지표와 로그를 수집한 후에는 쿼리를 작성하고 정상 동작과 이상 동작 모두에 관해 유용한 인사이트를 제공하는 집계 지표를 생성할 수 있습니다. 예를 들어 Amazon CloudWatch Logs Insights를 사용하여 애플리케이션 로그에서 사용자 지정 지표를 도출하고, Amazon CloudWatch Metrics Insights를 사용하여 대규모로 지표를 쿼리하고, Amazon CloudWatch Container Insights를 사용하여 컨테이너화된 애플리케이션 및 마이크로서비스에서 지표와 로그를 수집, 집계 및 요약할 수 있고, AWS Lambda 함수를 사용하는 경우 Amazon CloudWatch Lambda Insights를 사용할 수 있습니다. 집계 오류율 지표를 생성하려면 구성 요소 로그에서 오류 응답 또는 메시지가 발견될 때마다 카운터를 늘리거나 기존 오류율 지표의 집계 값을 계산할 수 있습니다. 이 데이터를 사용하여 성능이 가장 낮은 요청 또는 프로세스와 같은 말단 행동을 보여주는 히스토그램을 생성할 수 있습니다. CloudWatch Logs 이상 탐지와 같은 솔루션을 사용하여 이 데이터를 실시간으로 스캔하여 이상 패턴을 확인할 수도 있습니다. 이러한 인사이트는 대시보드에 배치하여 요구 사항과 기본 설정에 따라 정리할 수 있습니다.

로그 쿼리를 통해 워크로드 구성 요소가 특정 요청을 처리하는 방법을 이해하고 워크로드의 복원력에 영향을 미치는 요청 패턴 또는 기타 맥락을 파악할 수 있습니다. 필요에 따라 더 쉽게 실행할 수 있도록 애플리케이션 및 기타 구성 요소의 작동 방식에 대한 지식을 기반으로 쿼리를 미리 조사하고 준비하면 유용할 수 있습니다. 예를 들어, CloudWatch Logs Insights를 사용하면 CloudWatch Logs 내 로그 데이터를 대화식으로 검색하고 분석할 수 있습니다. Amazon Athena를 사용하여 많은 AWS 서비스를 포함하여 여러 소스의 로그를 페타바이트 규모로 쿼리할 수도 있습니다.

로그 보존 정책을 정의할 때 기록 로그의 값을 고려합니다. 기록 로그는 워크로드 성능의 장기 사용 및 동작 패턴, 회귀 및 개선을 식별하는 데 도움이 될 수 있습니다. 영구적으로 삭제된 로그는 나중에 분석할 수 없습니다. 그러나 기록 로그의 가치는 장기간에 걸쳐 감소하는 경향이 있습니다. 적절하게 균형을 맞추고 적용될 수 있는 법적 또는 계약상의 요구 사항을 준수하는 정책을 선택합니다.

구현 단계

1. 관찰성 데이터에 대한 수집, 스토리지, 분석 및 표시 메커니즘을 선택합니다.
2. 워크로드의 적절한 구성 요소(예: Amazon EC2 인스턴스 및 사이드카 컨테이너)에 지표 및 로그 수집기를 설치하고 구성합니다. 예기치 않게 중지되는 경우 자동으로 다시 시작하도록 이러한 수집기

를 구성합니다. 임시 게시 실패가 애플리케이션에 영향을 미치거나 데이터가 손실되지 않도록 수집기에 디스크 또는 메모리 버퍼링을 활성화합니다.

3. 워크로드의 일부로 사용하는 AWS 서비스에 대한 로깅을 활성화하고 필요한 경우 선택한 스토리지 서비스에 해당 로그를 전달합니다. 자세한 지침은 해당 서비스의 사용 설명서 또는 개발자 안내서를 참조하세요.
4. 원격 측정 데이터를 기반으로 워크로드와 관련된 운영 지표를 정의합니다. 이는 워크로드 구성 요소에서 방출되는 직접 지표를 기반으로 할 수 있으며, 여기에는 비즈니스 KPI 관련 지표 또는 합계, 비율, 백분위수 또는 히스토그램과 같은 집계된 계산 결과가 포함될 수 있습니다. 로그 분석기를 사용하여 이러한 지표를 계산하고 적절하게 대시보드에 배치합니다.
5. 필요에 따라 워크로드 구성 요소, 요청 또는 트랜잭션 동작을 분석하기 위해 적절한 로그 쿼리를 준비합니다.
6. 구성 요소 로그에 대한 로그 보존 정책을 정의하고 활성화합니다. 정책에서 허용하는 것보다 오래된 로그는 주기적으로 삭제합니다.

## 리소스

관련 모범 사례:

- REL06-BP01 워크로드의 모든 구성 요소 모니터링(생성)
- REL06-BP03 알림 전송(실시간 처리 및 경보)
- REL06-BP04 응답 자동화(실시간 처리 및 경보)
- REL06-BP05 로그 분석
- REL06-BP06 모니터링 범위 및 지표를 정기적으로 검토
- REL06-BP07 시스템을 통한 요청의 엔드 투 엔드 추적 모니터링

관련 설명서:

- How Amazon CloudWatch works
- Amazon Managed Prometheus
- Amazon Managed Grafana
- Analyzing log data with CloudWatch Logs Insights
- Amazon CloudWatch Lambda Insights
- Amazon CloudWatch Container Insights
- Query your metrics with CloudWatch Metrics Insights

- AWS Distro for OpenTelemetry
- Amazon CloudWatch Logs Insights Sample Queries
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- Searching and Filtering Log Data
- Sending Logs Directly to Amazon S3
- Amazon Builders' Library: 운영 가시성을 위한 분산 시스템 계측

관련 워크숍:

- One Observability 워크숍

관련 도구:

- AWS Distro for OpenTelemetry (GitHub)

## REL06-BP03 알림 전송(실시간 처리 및 경보)

조직은 잠재적인 문제를 탐지하면 해당 직원과 시스템에 실시간 알림과 경고를 보내 이러한 문제에 신속하고 효과적으로 대응할 수 있도록 합니다.

원하는 성과: 서비스 및 애플리케이션 지표를 기반으로 관련 경보를 구성하여 운영 이벤트에 신속하게 대응할 수 있습니다. 경보 임곗값이 위반되면 적절한 인력과 시스템에 알림이 전송되어 근본적인 문제를 해결할 수 있습니다.

일반적인 안티 패턴:

- 임곗값이 지나치게 높은 경보를 구성하여 중요한 알림을 보내지 못합니다.
- 임곗값이 너무 낮은 경보를 구성하여 과도한 알림 노이즈로 인해 중요한 알림에 대한 조치를 취하지 않습니다.
- 사용량이 변경될 때 경보 및 임곗값을 업데이트하지 않습니다.
- 자동화된 작업을 통해 가장 잘 해결되는 경보에 대해, 자동화된 작업을 생성하지 않고 담당자에게 알림을 보내 과도한 알림을 전송합니다.

이 모범 사례 확립의 이점: 적절한 인력과 시스템에 실시간 알림과 경고를 전송하면 문제를 조기에 탐지하고 운영 인시던트에 신속하게 대응할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

워크로드는 애플리케이션의 가용성에 영향을 미칠 수 있는 문제의 탐지 가능성을 개선하고 자동화된 대응을 위한 트리거 역할을 할 수 있도록 실시간 처리 및 경보 기능을 갖추어야 합니다. 조직은 정의된 지표로 경고를 생성하여 중요한 이벤트가 발생하거나 지표가 임곗값을 초과할 때마다 알림을 수신하여 실시간 처리 및 경보를 수행할 수 있습니다.

Amazon CloudWatch를 사용하면 지표 및 정적 임곗값, 이상 탐지 및 기타 기준에 기반한 CloudWatch 경보를 사용하는 복합 경보를 만들 수 있습니다. CloudWatch를 사용하여 구성할 수 있는 경보 유형에 대한 자세한 내용은 CloudWatch 설명서의 경보 섹션을 참조하세요.

팀의 AWS 리소스 지표 및 알림에 대한 사용자 지정 보기를 구성할 수 있습니다. 이 경우 CloudWatch 대시보드를 사용할 수 있습니다. CloudWatch 콘솔의 사용자 지정 가능한 홈 페이지를 사용하면 여러 리전에 걸쳐 단일 보기에서 리소스를 모니터링할 수 있습니다.

경보는 Amazon SNS 주제로 알림 전송, Amazon EC2 작업이나 Amazon EC2 Auto Scaling 작업 수행, AWS Systems Manager에서 인시던트 또는 OpsItem 생성과 같은 하나 이상의 작업을 수행할 수 있습니다.

Amazon CloudWatch는 경보에서 상태가 변경되면 Amazon SNS를 사용하여 알림을 전송하고, 게시자(생산자)에서 구독자(소비자)에게 메시지를 전달합니다. Amazon SNS 알림 설정에 대한 자세한 내용은 Configuring Amazon SNS를 참조하세요.

CloudWatch 경보가 생성, 업데이트, 삭제되거나 상태가 변경될 때마다 CloudWatch는 EventBridge에 이벤트를 전송합니다. 이러한 이벤트와 함께 EventBridge를 사용하여 경보 상태가 변경될 때마다 사용자에게 알리거나 Systems Manager Automation을 사용하여 계정에서 이벤트를 자동으로 트리거하는 등의 작업을 수행하는 규칙을 만들 수 있습니다.

AWS Health로 최신 정보를 확인하세요. AWS Health는 AWS 클라우드 리소스 상태에 대한 신뢰할 수 있는 정보 소스입니다. AWS Health를 사용하면 확인된 서비스 이벤트에 대한 알림을 받아 영향을 완화하기 위한 조치를 신속하게 취할 수 있습니다. AWS 사용자 알림를 통해 이메일 및 채팅 채널에 적합한 AWS Health 이벤트 알림을 생성하고, Amazon EventBridge를 통해 모니터링 및 알림 도구와 프로그래밍 방식으로 통합할 수 있습니다. AWS Organizations를 사용하는 경우 여러 계정에서 AWS Health 이벤트를 집계합니다.

EventBridge 또는 Amazon SNS는 언제 사용해야 하나요?

이벤트 기반 애플리케이션을 개발하는 데 EventBridge 및 Amazon SNS를 모두 사용할 수 있으며, 특정 요구 사항에 따라 선택이 달라집니다.

Amazon EventBridge는 자체 애플리케이션, SaaS 애플리케이션 및 AWS 서비스의 이벤트에 대응하는 애플리케이션을 구축하려는 경우 권장됩니다. EventBridge는 서드파티 SaaS 파트너와 직접 통합되는 유일한 이벤트 기반 서비스입니다. 또한 EventBridge는 개발자가 계정에 리소스를 생성할 필요 없이 200개가 넘는 AWS 서비스에서 이벤트를 자동으로 수집합니다.

EventBridge는 이벤트에 대해 정의된 JSON 기반 구조를 사용하며, 이벤트 본문 전체에 적용되는 규칙을 생성하여 대상으로 전달한 이벤트를 선택합니다. EventBridge는 현재 20개가 넘는 AWS 서비스(AWS Lambda, Amazon SQS, Amazon SNS, Amazon Kinesis Data Streams, Amazon Data Firehose 등)를 대상으로 지원합니다.

Amazon SNS는 높은 팬 아웃(수천 또는 수백만 개의 엔드포인트)이 필요한 애플리케이션에 권장됩니다. 일반적으로 고객이 Amazon SNS를 규칙 대상으로 사용하여 필요한 이벤트를 필터링하고 여러 엔드포인트로 팬아웃하는 패턴을 볼 수 있습니다.

메시지는 비정형 양식으며, 어떤 형식이든 가능합니다. Amazon SNS는 Lambda, Amazon SQS, HTTP/S 엔드포인트, SMS, 모바일 푸시, 이메일과 같은 6가지 여러 유형의 대상으로 메시지 전달을 지원합니다. Amazon SNS의 일반적인 지연 시간은 30밀리초 미만입니다. 다양한 AWS 서비스에서 Amazon SNS 메시지를 전송하도록 서비스를 구성하여 이를 수행합니다(Amazon EC2, Amazon S3, Amazon RDS 등 30개가 넘는 서비스 지원).

구현 단계

1. Amazon CloudWatch 경보를 사용하여 경보를 생성합니다.
   a. 지표 경보에서는 단일 CloudWatch 지표 또는 CloudWatch 지표에 종속된 표현식을 모니터링합니다. 경보는 여러 시간 간격에 걸쳐 임곗값과 비교한 지표 또는 표현식의 값에 따라 하나 이상의 작업을 시작합니다. Amazon SNS 주제에 알림 전송, Amazon EC2 작업이나 Amazon EC2 Auto Scaling 작업 수행, AWS Systems Manager에서 인시던트 또는 OpsItem 생성으로 작업이 구성될 수 있습니다.
   b. 복합 경보는 사용자가 만든 다른 경보의 경보 조건을 고려하는 규칙 표현식으로 구성됩니다. 복합 경보는 모든 규칙 조건이 충족되는 경우에만 경보 상태로 전환됩니다. 복합 경보의 규칙 표현식에 지정된 경보에는 지표 경보 및 추가 복합 경보가 포함될 수 있습니다. 복합 경보는 경보 상태가 변경될 때 Amazon SNS 알림을 전송할 수 있고, 경보 상태로 진입할 때 Systems Manager OpsItem 또는 인시던트를 생성할 수 있지만, Amazon EC2 작업 또는 Auto Scaling 작업은 수행할 수 없습니다.
2. Amazon SNS 알림을 설정합니다. CloudWatch 경보 생성 시 경보 상태가 변경될 때 알림을 보내도록 Amazon SNS 주제를 포함할 수 있습니다.
3. 지정된 CloudWatch 경보와 일치하는 규칙을 EventBridge에서 생성합니다. 각 규칙은 Lambda 함수를 비롯한 여러 대상을 지원합니다. 예를 들어, 사용 가능한 디스크 공간이 부족할 때 시작되는 경

보를 정의하여 EventBridge 규칙을 통해 Lambda 함수를 트리거하여 공간을 정리할 수 있습니다. EventBridge 대상에 대한 자세한 내용은 EventBridge targets를 참조하세요.

## 리소스

관련 Well-Architected 모범 사례:

- REL06-BP01 워크로드의 모든 구성 요소 모니터링(생성)
- REL06-BP02 지표 정의 및 계산(집계)
- REL12-BP01 플레이북을 사용하여 장애 조사

관련 문서:

- Amazon CloudWatch
- CloudWatch Logs insights
- Amazon CloudWatch 경보 사용
- Amazon CloudWatch 대시보드 사용
- Amazon CloudWatch 지표 사용
- Setting up Amazon SNS notifications
- CloudWatch 이상 탐지
- CloudWatch Logs data protection
- Amazon EventBridge
- Amazon Simple Notification Service

관련 비디오:

- reinvent 2022 observability 동영상
- AWS re:Invent 2022 - Observability best practices at Amazon

관련 예제:

- One Observability 워크숍
- Amazon EventBridge to AWS Lambda with feedback control by Amazon CloudWatch Alarms

# REL06-BP04 응답 자동화(실시간 처리 및 경보)

이벤트가 감지되면 자동화를 사용하여 실패한 구성 요소를 대체하는 등의 조치를 취합니다.

경보의 자동화된 실시간 처리가 구현되어 경보가 트리거될 때 시스템이 신속한 시정 조치를 취하고 장애 또는 서비스 저하를 방지할 수 있습니다. 경보에 대한 자동 대응에는 장애가 발생한 구성 요소 교체, 컴퓨팅 용량 조정, 정상적인 호스트, 가용 영역 또는 기타 리전으로 트래픽 리디렉션, 운영자에게 알림 등이 포함될 수 있습니다.

원하는 성과: 실시간 경보를 식별하고 서비스 수준 목표 및 서비스 수준에 관한 계약(SLA)을 유지하기 위해 적절한 조치를 취하도록 경보 자동 처리를 설정합니다. 자동화는 단일 구성 요소의 자가 복구 작업부터 전체 사이트 장애 조치에 이르기까지 다양합니다.

일반적인 안티 패턴:

- 주요 실시간 경보의 명확한 인벤토리 또는 카탈로그가 없습니다.
- 핵심 경보에 대한 자동 응답이 없습니다(예: 컴퓨팅이 거의 고갈될 때 자동 규모 조정 시행).
- 경보의 대응 조치가 상충합니다.
- 운영자가 경고 알림을 받을 때 따라야 하는 표준 운영 절차(SOP)가 없습니다.
- 구성 변경을 모니터링하지 않습니다(감지되지 않은 구성 변경으로 인해 워크로드에 다운타임이 발생할 수 있음).
- 의도하지 않은 구성 변경을 취소할 전략이 없습니다.

이 모범 사례 확립의 이점: 경보 처리를 자동화하면 시스템 복원력을 개선할 수 있습니다. 시스템이 자동으로 시정 조치를 취하므로 사람이 개입하여 오류가 발생하기 쉬운 수동 작업이 줄어듭니다. 워크로드 운영이 가용성 목표를 충족하고 서비스 중단을 줄입니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 지침

알림을 효과적으로 관리하고 대응을 자동화하려면 중요도와 영향을 기준으로 알림을 분류하고, 대응 절차를 문서화하며, 작업에 순위를 매기기 전에 대응을 계획합니다.

구체적인 조치가 필요한 작업을 식별하고(런북에 자세히 설명되어 있는 경우가 많음), 모든 런북과 플레이북을 검토하여 자동화할 수 있는 작업을 결정합니다. 작업을 정의할 수 있는 경우 대개 자동화할 수 있습니다. 작업을 자동화할 수 없는 경우 SOP에 수동 단계를 문서화하고 운영자에게 이 단계를 교

육합니다. 알림 대응을 자동화하기 위한 계획을 수립하고 유지할 수 있는 자동화 기회를 위해 수동 프로세스에 지속적으로 검토합니다.

구현 단계

1. 경보 인벤토리 생성: 모든 경보 목록을 가져오려면 AWS CLI에서 Amazon CloudWatch `describe-alarms` 명령을 사용할 수 있습니다. 설정한 경보 수에 따라 페이지 매김을 사용하여 각 직접 호출에 대한 경보의 일부를 검색해야 할 수도 있고, AWS SDK를 사용하여 API 직접 호출을 통해 경보를 가져올 수도 있습니다.
2. 모든 경보 동작 문서화: 수동이든 자동이든 관계없이 모든 경보와 해당 작업으로 런북을 업데이트합니다. AWS Systems Manager에서는 사전 정의된 런북을 제공합니다. 실행서에 대한 자세한 내용은 실행서 작업을 참조하세요. 런북 콘텐츠를 보는 방법에 대한 자세한 내용은 View runbook content를 참조하세요.
3. 경보 작업 설정 및 관리: 작업이 필요한 모든 경보의 경우 CloudWatch SDK를 사용하여 자동화된 작업을 지정합니다. 예를 들어, 경보에 대한 작업을 생성 및 활성화하거나 비활성화하여 CloudWatch 경보를 기반으로 Amazon EC2 인스턴스 상태를 자동으로 변경할 수 있습니다.

   Amazon EventBridge를 사용하여 애플리케이션 가용성 문제나 리소스 변경 등의 시스템 이벤트에 자동으로 대응할 수 있습니다. 관심 있는 이벤트와 이벤트가 규칙과 일치할 때 수행할 작업을 표시하는 규칙을 생성할 수 있습니다. 자동으로 시작할 수 있는 작업으로, AWS Lambda 함수 간접 호출, Amazon EC2 `Run Command` 간접 호출, Amazon Kinesis Data Streams에 이벤트 중계, EventBridge를 사용하여 Amazon EC2 자동화 참조 등이 포함될 수 있습니다.
4. 표준 운영 절차(SOP): 애플리케이션 구성 요소를 기반으로 AWS Resilience Hub에서 여러 개의 SOP 템플릿을 권장합니다. 이러한 SOP를 사용하여 경보가 발생한 경우 운영자가 따라야 하는 모든 프로세스를 문서화할 수 있습니다. Resilience Hub의 권장 사항에 따라 SOP를 구성할 수도 있습니다. 이 경우 관련 복원력 정책이 포함된 Resilience Hub 애플리케이션과 해당 애플리케이션에 대한 복원력 평가 내역이 필요합니다. SOP에 대한 추천은 복원력 평가를 통해 작성됩니다.

   Resilience Hub는 Systems Manager와 연동하여 SOP의 기반으로 사용할 수 있는 다양한 SSM 문서를 제공함으로써 SOP의 단계를 자동화합니다. 예를 들어, 은 기존 SSM Automation 문서를 기반으로 디스크 스페이스를 추가하기 위한 SOP를 권장할 수 있습니다.
5. Amazon DevOps Guru를 사용하여 자동화된 작업 수행: Amazon DevOps Guru는 애플리케이션 리소스가 비정상적으로 작동하는지를 자동으로 모니터링하고 표적화된 권장 사항을 제공하여 문제 파악 및 해결 시간을 단축합니다. DevOps Guru를 사용하면 Amazon CloudWatch 지표, AWS Config, AWS CloudFormation, AWS X-Ray를 비롯한 여러 소스에서 운영 데이터 스트림을 거의 실시간으로 모니터링할 수 있습니다. 또한 DevOps Guru를 사용하여 OpsCenter에서 OpsItems를 자동으로 생성하고 추가 자동화를 위해 EventBridge에 이벤트를 전송할 수 있습니다.

## 리소스

관련 모범 사례:

- REL06-BP01 워크로드의 모든 구성 요소 모니터링(생성)
- REL06-BP02 지표 정의 및 계산(집계)
- REL06-BP03 알림 전송(실시간 처리 및 경보)
- REL08-BP01 배포와 같은 표준 활동에 런북 사용

관련 문서:

- AWS Systems Manager 자동화
- Creating an EventBridge Rule That Triggers on an Event from an AWS Resource
- One Observability 워크숍
- Amazon Builders' Library: 운영 가시성을 위한 분산 시스템 계측
- What is Amazon DevOps Guru?
- 자동화 문서(플레이북) 작업

관련 비디오:

- AWS re:Invent 2022 - Observability best practices at Amazon
- AWS re:Invent 2020: Automate anything with AWS Systems Manager
- Introduction to AWS Resilience Hub
- Create Custom Ticket Systems for Amazon DevOps Guru Notifications
- Enable Multi-Account Insight Aggregation with Amazon DevOps Guru

관련 예제:

- Amazon CloudWatch 및 Systems Manager 워크숍

# REL06-BP05 로그 분석

로그 파일 및 지표 기록을 수집하고 이를 분석하여 더 광범위한 추세 및 워크로드 인사이트를 확보합니다.

Amazon CloudWatch 로그 인사이트는 로그 데이터를 분석하는 데 사용할 수 있는 단순하지만 강력한 쿼리 언어를 지원합니다. Amazon CloudWatch Logs 또한 구독을 지원하므로 데이터를 Amazon S3로 원활하게 보내 여기서 데이터를 사용하거나 Amazon Athena로 보내 데이터를 쿼리할 수 있습니다. 다양한 형식의 쿼리도 지원됩니다. 자세한 내용은 Amazon Athena 사용 설명서의 지원되는 SerDes 및 데이터 형식을 참조하세요. 방대한 로그 파일 세트를 분석하려면 Amazon EMR 클러스터를 실행하여 페타바이트 규모의 분석을 실행할 수 있습니다.

AWS 파트너와 서드파티에서 제공하는 다양한 도구를 집계, 처리, 저장 및 분석에 사용할 수 있습니다. 이러한 도구에는 New Relic, Splunk, Loggly, Logstash, CloudHealth 및 Nagios가 포함됩니다. 그러나 시스템과 애플리케이션 외부에서 생성되는 로그는 각 클라우드 공급자별로 다르며 각 서비스별로 다른 경우도 많습니다.

모니터링 프로세스에서 간과되는 경우가 많은 작업 중 하나로 데이터 관리를 들 수 있습니다. 데이터 모니터링을 위한 보존 요구 사항을 확인한 후 그에 따라 수명 주기 정책을 적용해야 합니다. Amazon S3는 S3 버킷 수준에서 수명 주기 관리를 지원합니다. 버킷의 여러 경로에 이 수명 주기 관리 기능을 다르게 적용할 수 있습니다. 수명 주기 종료가 가까워지면 장기 저장을 위해 데이터를 Amazon S3 Glacier로 전환한 다음 보존 기간이 종료되면 데이터를 만료 처리할 수 있습니다. S3 Intelligent-Tiering 스토리지 클래스는 성능 영향이나 운영 오버헤드 없이 데이터를 가장 비용 효율적인 티어로 자동으로 이동하여 비용을 최적화하도록 설계되었습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

- CloudWatch 로그 인사이트를 사용하면 Amazon CloudWatch Logs 내 로그 데이터를 대화식으로 검색해 분석할 수 있습니다.
  - Analyzing Log Data with CloudWatch Logs Insights
  - Amazon CloudWatch Logs Insights Sample Queries
- Amazon CloudWatch Logs를 사용하여 Amazon S3으로 로그를 전송한 후, 로그 데이터를 사용하거나 Amazon Athena로 데이터를 쿼리할 수 있습니다.
  - Athena를 사용하여 Amazon S3 서버 액세스 로그를 분석하려면 어떻게 해야 하나요?
    - 서버 액세스 로그 버킷에 대한 S3 수명 주기 정책을 생성합니다. 주기적으로 로그 파일을 제거하도록 수명 주기 정책을 구성하십시오. 그러면 Athena에서 쿼리마다 분석하는 데이터의 양이 줄어듭니다.
      - S3 버킷에 대한 수명 주기 정책을 생성하려면 어떻게 해야 하나요?

## 리소스

관련 문서:

- Amazon CloudWatch Logs Insights Sample Queries
- Analyzing Log Data with CloudWatch Logs Insights
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- S3 버킷에 대한 수명 주기 정책을 생성하려면 어떻게 해야 하나요?
- Athena를 사용하여 Amazon S3 서버 액세스 로그를 분석하려면 어떻게 해야 하나요?
- One Observability 워크숍
- Amazon Builders' Library: 운영 가시성을 위한 분산 시스템 계측

# REL06-BP06 모니터링 범위 및 지표를 정기적으로 검토

워크로드 모니터링이 구현되는 방법을 자주 검토하고 워크로드와 아키텍처가 변화함에 따라 업데이트합니다. 모니터링을 정기적으로 감사하면 문제 지표를 놓치거나 간과할 위험을 줄이고 워크로드가 가용성 목표를 달성하는 데 도움이 됩니다.

효과적인 모니터링은 주요 비즈니스 지표에 기반을 두고 있으며, 이는 비즈니스 우선순위가 변경될 때 변화합니다. 모니터링 검토 프로세스는 서비스 수준 지표(SLI)를 강조하고 인프라, 애플리케이션, 클라이언트 및 사용자의 인사이트를 통합해야 합니다.

원하는 성과: 정기적으로, 그리고 중요한 이벤트 또는 변경 사항이 발생한 후에 검토 및 업데이트되는 효과적인 모니터링 전략이 있습니다. 워크로드 및 비즈니스 요구 사항이 진화함에 따라 주요 애플리케이션 상태 지표가 여전히 관련이 있는지 확인합니다.

일반적인 안티 패턴:

- 기본 지표만 수집합니다.
- 모니터링 전략을 설정하지만 검토하지는 않습니다.
- 주요 변경 사항이 배포될 때 모니터링에 대해 논의하지 않습니다.
- 오래된 지표를 신뢰하여 워크로드 상태를 판단합니다.
- 운영 팀은 오래된 지표와 임계값으로 인해 발생하는 오탐지로 업무 부담이 큽니다.
- 모니터링되지 않는 애플리케이션 구성 요소를 관찰할 수 없습니다.
- 모니터링에서 비즈니스 지표를 제외하고 하위 수준의 기술 지표에만 중점을 둡니다.

이 모범 사례 확립의 이점: 모니터링을 정기적으로 검토하면 잠재적 문제를 예측하고 감지할 수 있는지 확인할 수 있습니다. 또한 이전 검토 중에 놓쳤을 수 있는 사각지대를 발견할 수 있으므로 문제를 감지하는 능력이 더욱 향상됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

운영 준비도 검토(ORR) 프로세스 중에 모니터링 지표와 범위를 검토합니다. 일관된 일정에 따라 정기적인 운영 준비도 상태 검토를 수행하여 현재 워크로드와 구성한 모니터링 사이에 격차가 있는지 평가합니다. 운영 성능 검토 및 지식 공유를 위한 정기 케이던스를 설정하여 운영 팀의 성과를 개선하는 역량을 발전시킵니다. 기존 알림 임계값이 여전히 적절한지 확인하고 운영 팀이 오탐지 알림을 수신하거나 모니터링해야 하는 애플리케이션의 측면을 모니터링하지 않는 상황을 확인합니다.

Resilience Analysis Framework는 프로세스를 탐색하는 데 도움이 되는 유용한 지침을 제공합니다. 프레임워크의 초점은 잠재적 장애 모드와 그 영향을 완화하는 데 사용할 수 있는 예방 및 수정을 위한 제어 조치를 식별하는 것입니다. 이 지식은 모니터링 및 알림에 적합한 지표와 이벤트를 식별하는 데 도움이 될 수 있습니다.

구현 단계

1. 워크로드 대시보드에 대한 정기적인 검토를 예약하고 수행합니다. 검사하는 세부 수준을 나타내는 다양한 케이던스를 구성할 수 있습니다.
2. 지표에서 추세가 나타나는지 검사합니다. 지표 값을 과거 값과 비교하여 조사해야 할 문제가 있음을 시사하는 추세가 나타나는지 확인합니다. 이러한 예로는 지연 시간 증가, 주요 비즈니스 기능 감소, 장애 응답 증가 등이 있습니다.
3. 평균 또는 중앙값으로 마스킹할 수 있는 지표의 이상치 및 이상을 검사합니다. 기간 중 가장 높은 값과 가장 낮은 값을 살펴보고 정상 범위를 훨씬 벗어난 관찰의 원인을 조사합니다. 이러한 원인을 계속 제거하면 워크로드 성능의 일관성 향상에 따라 예상 지표 범위를 좁힐 수 있습니다.
4. 동작에 급격한 변화가 있는지 알아봅니다. 지표의 수량 또는 방향이 갑자기 바뀔 경우, 애플리케이션이 변경되었거나 외부 요인이 발생한 것일 수 있습니다. 이 같은 외부 요인으로 인해 추적할 지표를 추가해야 할 수 있습니다.
5. 현재 모니터링 전략이 애플리케이션과 관련이 있는지 검토합니다. 이전 인시던트 분석(또는 Resilience Analysis Framework)을 기반으로 모니터링 범위에 추가로 포함해야 하는 측면이 있는지 평가합니다.
6. 실제 사용자 모니터링(RUM) 지표를 검토하여 애플리케이션 기능 적용 범위에 격차가 있는지 확인합니다.

7. 변경 관리 프로세스를 검토합니다. 필요한 경우 절차를 업데이트하여 변경을 승인하기 전에 수행해야 하는 모니터링 분석 단계를 포함합니다.
8. 운영 준비도 검토 및 오류 프로세스 수정의 일환으로 모니터링 검토를 구현합니다.

## 리소스

### 관련 모범 사례

- REL06-BP01 워크로드의 모든 구성 요소 모니터링(생성)
- REL06-BP02 지표 정의 및 계산(집계)
- REL06-BP07 시스템을 통한 요청의 엔드 투 엔드 추적 모니터링
- REL12-BP02 인시던트 사후 분석 수행
- REL12-BP06 정기적으로 게임 데이 진행

### 관련 문서:

- Why you should develop a correction of error (COE)
- Amazon CloudWatch 대시보드 사용
- 운영 가시성을 위한 대시보드 구축
- Advanced Multi-AZ Resilience Patterns - Gray failures
- Amazon CloudWatch Logs Insights Sample Queries
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- One Observability 워크숍
- Amazon Builders' Library: 운영 가시성을 위한 분산 시스템 계측
- Amazon CloudWatch 대시보드 사용
- AWS Observability Best Practices
- Resilience analysis framework
- Resilience Analysis Framework - Observability
- Operational Readiness Review - ORR

# REL06-BP07 시스템을 통한 요청의 엔드 투 엔드 추적 모니터링

제품 팀이 문제를 더 쉽게 분석 및 디버그하고 성능을 개선할 수 있도록 서비스 구성 요소를 통해 처리되는 요청을 추적합니다.

원하는 성과: 모든 구성 요소를 포괄적으로 추적할 수 있는 워크로드는 디버깅하기 쉬우므로 근본 원인 찾기를 단순화하여 오류의 평균 해결 시간(MTTR) 및 지연 시간이 개선됩니다. 엔드 투 엔드 추적은 영향을 받는 구성 요소를 찾고 오류 또는 지연 시간의 근본 원인을 자세히 조사하는 데 걸리는 시간을 줄여줍니다.

일반적인 안티 패턴:

- 추적이 모든 구성 요소가 아니라 일부 구성 요소에서만 사용됩니다. 예를 들어 AWS Lambda를 추적하지 않으면 팀이 급증하는 워크로드에서 콜드 스타트로 인한 지연 시간을 명확하게 이해하지 못할 수 있습니다.
- 가상 canary 또는 실제 사용자 모니터링(RUM)이 추적이 가능하도록 구성되지 않습니다. Canary 또는 RUM이 없으면 추적 분석에서 클라이언트 상호 작용 원격 측정이 생략되어 불완전한 성능 프로파일이 생성됩니다.
- 하이브리드 워크로드에 클라우드 네이티브 및 서드파티 추적 도구가 모두 포함되지만 단일 추적 솔루션을 선택하고 완전히 통합하는 단계는 아직 수행하지 않았습니다. 선택한 추적 솔루션에 따라 클라우드 네이티브 추적 SDK를 사용하여 클라우드 네이티브가 아닌 구성 요소를 측정하거나 서드파티 도구를 클라우드 네이티브 추적 원격 측정을 수집하도록 구성해야 합니다.

이 모범 사례 확립의 이점: 개발 팀은 문제에 대한 경고를 받으면 구성 요소별 로깅, 성능, 장애와의 상관관계를 포함하여 시스템 구성 요소 상호 작용을 종합적으로 파악할 수 있습니다. 추적을 통해 근본 원인을 시각적으로 쉽게 식별할 수 있으므로 근본 원인을 조사하는 데 소요되는 시간이 줄어듭니다. 구성 요소 상호 작용을 자세히 이해하는 팀은 문제를 해결할 때 더 현명하고 빠른 의사 결정을 내릴 수 있습니다. 시스템 추적을 분석하면 재해 복구(DR) 장애 조치를 언제 실행할지, 자가 복구 전략을 가장 잘 구현할 수 있는 위치 등을 더 제대로 결정할 수 있어 궁극적으로 서비스에 대한 고객 만족도를 향상시킬 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

분산된 애플리케이션을 운영하는 팀은 추적 도구를 사용하여 상관관계 식별자를 설정하고 요청 추적을 수집하며 연결된 구성 요소의 서비스 맵을 구축할 수 있습니다. 서비스 클라이언트, 미들웨어 게이트웨이 및 이벤트 버스, 컴퓨팅 구성 요소, 키 값 저장소 및 데이터베이스가 포함된 스토리지 등 모든 애

플리케이션 구성 요소가 요청 추적에 포함되어야 합니다. 서비스 수준에 관한 계약과 목표에 대해 시스템 성능을 정확하게 평가할 수 있도록 엔드 투 엔드 추적 구성에 가상 canary와 실제 사용자 모니터링을 포함하여 원격 클라이언트 상호 작용과 지연 시간을 측정합니다.

AWS X-Ray 및 Amazon CloudWatch 애플리케이션 모니터링 계측 서비스를 사용하여 애플리케이션에서 요청이 이동할 때 해당 요청을 전체적으로 파악할 수 있습니다. X-Ray는 애플리케이션 원격 측정을 수집합니다. 페이로드, 함수, 추적, 서비스, API에서 이를 시각화하고 필터링할 수 있으며 노코드 또는 로우 코드 시스템 구성 요소에 대해 활성화할 수 있습니다. CloudWatch 애플리케이션 모니터링에는 추적을 지표, 로그 및 경보와 통합하는 Servicelens가 포함되어 있습니다. 또한 CloudWatch 애플리케이션 모니터링에는 엔드포인트와 API를 모니터링하기 위한 가상 및 웹 애플리케이션 클라이언트를 측정하기 위한 실제 사용자 모니터링도 포함됩니다.

## 구현 단계

- Amazon S3, AWS Lambda, Amazon API Gateway와 같은 지원되는 모든 기본 서비스에서 AWS X-Ray를 사용합니다. 이러한 AWS 서비스에서는 코드형 인프라, AWS SDK 또는 AWS Management Console을 사용하여 구성 토글을 통해 X-Ray가 활성화됩니다.
- 애플리케이션 AWS Distro for Open Telemetry 및 X-Ray 또는 서드파티 수집 에이전트를 계측합니다.
- 프로그래밍 언어별 구현에 대한 자세한 내용은 AWS X-Ray 개발자 안내서를 검토하세요. 이러한 설명서 섹션에서는 애플리케이션 프로그래밍 언어와 관련된 HTTP 요청, SQL 쿼리 및 기타 프로세스의 계측 방법을 자세히 설명합니다.
- Amazon CloudWatch Synthetic Canary 및 Amazon CloudWatch RUM에 X-Ray 추적을 사용하여 최종 사용자 클라이언트에서 다운스트림 AWS 인프라를 통과하는 요청 경로를 분석합니다.
- 팀이 문제에 대해 빠르게 경고를 받은 후 ServiceLens를 사용하여 추적 및 서비스 맵을 심층적으로 분석할 수 있도록 리소스 상태와 canary 원격 측정을 기반으로 CloudWatch 지표 및 경보를 구성합니다.
- 기본 추적 솔루션에 서드파티 도구를 사용하는 경우 Datadog, New Relic 또는 Dynatrace와 같은 서드파티 추적 도구에 대한 X-Ray 통합을 활성화합니다

## 리소스

관련 모범 사례:

- REL06-BP01 워크로드의 모든 구성 요소 모니터링(생성)
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지

관련 문서:

- AWS X-Ray란 무엇입니까?
- Amazon CloudWatch: 애플리케이션 모니터링
- Debugging with Amazon CloudWatch Synthetics and AWS X-Ray
- Amazon Builders' Library: 운영 가시성을 위한 분산 시스템 계측
- Integrating AWS X-Ray with other AWS services
- AWS Distro for OpenTelemetry 및 AWS X-Ray
- Amazon CloudWatch: 가상 모니터링 사용
- Amazon CloudWatch: CloudWatch RUM 사용
- Set up Amazon CloudWatch synthetics canary and Amazon CloudWatch alarm
- Availability and Beyond: Understanding and Improving the Resilience of Distributed Systems on AWS

관련 예제:

- One Observability 워크숍

관련 비디오:

- AWS re:Invent 2022 - How to monitor applications across multiple accounts
- How to Monitor your AWS Applications

관련 도구:

- AWS X-Ray
- Amazon CloudWatch
- Amazon Route 53

# 수요 변경에 따라 조정되는 워크로드 설계

확장 가능한 워크로드는 리소스를 자동으로 추가 또는 제거할 수 있는 탄력성을 제공하여 리소스 공급이 특정 시점의 수요와 거의 일치하도록 합니다.

모범 사례

- REL07-BP01 리소스를 확보하거나 조정할 때 자동화 사용
- REL07-BP02 워크로드 장애 감지 시 리소스 확보
- REL07-BP03 워크로드에 더 많은 리소스가 필요한 것으로 감지되면 리소스 확보
- REL07-BP04 워크로드 로드 테스트

## REL07-BP01 리소스를 확보하거나 조정할 때 자동화 사용

클라우드에서 신뢰성의 초석은 인프라 및 리소스의 프로그래밍 방식 정의, 프로비저닝 및 관리입니다. 자동화를 통해 리소스 프로비저닝을 간소화하고, 일관되고 안전한 배포를 촉진하며, 전체 인프라에서 리소스를 확장할 수 있습니다.

원하는 성과: 코드형 인프라(IaC)를 관리합니다. 버전 제어 시스템(VCS)에서 인프라 코드를 정의하고 유지 관리합니다. AWS 리소스 프로비저닝을 자동화된 메커니즘에 위임하고 Application Load Balancer(ALB), Network Load Balancer(NLB) 및 Auto Scaling 그룹과 같은 관리형 서비스를 활용합니다. 지속적 통합/지속적 전송(CI/CD) 파이프라인을 사용하여 리소스를 프로비저닝하면 코드 변경이 Auto Scaling 구성에 대한 업데이트를 포함하여 리소스 업데이트를 자동으로 시작합니다.

일반적인 안티 패턴:

- 명령줄을 사용하거나 AWS Management Console(클릭 작업이라고도 함)에서 리소스를 수동으로 배포합니다.
- 애플리케이션 구성 요소 또는 리소스를 긴밀하게 결합하고 결과적으로 유연하지 않은 아키텍처를 생성합니다.
- 변화하는 비즈니스 요구 사항, 트래픽 패턴 또는 새로운 리소스 유형에 맞춰 조정되지 않는, 유연하지 않은 크기 조정 정책을 구현합니다.
- 예상 수요를 충족하기 위해 용량을 수동으로 추정합니다.

이 모범 사례 확립의 이점: 코드형 인프라(IaC)를 사용하면 인프라를 프로그래밍 방식으로 정의할 수 있습니다. 이렇게 하면 동일한 소프트웨어 개발 수명 주기를 통해 애플리케이션 변경과 동일하게 인프라 변경을 관리할 수 있으므로 일관성과 반복성을 높이고 오류가 발생하기 쉬운 수동 작업의 위험을 줄일 수 있습니다. 자동화된 전송 파이프라인으로 IaC를 구현하여 리소스를 프로비저닝하고 업데이트하는 프로세스를 더욱 간소화할 수 있습니다. 인프라 업데이트를 수동 개입 없이 안정적이고 효율적으로 배포할 수 있습니다. 이러한 민첩성은 변동하는 수요에 맞게 리소스 크기를 조정할 때 특히 중요합니다.

IaC 및 전송 파이프라인과 함께 동적이고 자동화된 리소스 크기 조정을 달성할 수 있습니다. Auto Scaling은 주요 지표를 모니터링하고 사전 정의된 크기 조정 정책을 적용하여 필요에 따라 리소스를 자동으로 프로비저닝하거나 프로비저닝을 해제할 수 있으므로 성능과 비용 효율성이 향상됩니다. 이렇게 하면 애플리케이션 또는 워크로드 요구 사항의 변경에 대한 대응으로 수동 오류 또는 지연의 가능성이 줄어듭니다.

IaC, 자동 전송 파이프라인 및 Auto Scaling의 조합을 통해 조직은 환경을 자신 있게 프로비저닝, 업데이트 및 확장할 수 있습니다. 이 자동화는 응답성과 복원력이 뛰어나며 효율적으로 관리되는 클라우드 인프라를 유지 관리하는 데 필수적입니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

AWS 아키텍처의 CI/CD 파이프라인 및 코드형 인프라(IaC)를 사용하여 자동화를 설정하려면 Git과 같은 버전 관리 시스템을 선택하여 IaC 템플릿 및 구성을 저장합니다. 이러한 템플릿은 AWS CloudFormation과 같은 도구를 사용하여 작성할 수 있습니다. 시작하려면 이러한 템플릿 내에서 인프라 구성 요소(예: AWS VPC, Amazon EC2 Auto Scaling 그룹 및 Amazon RDS 데이터베이스)를 정의합니다.

다음으로 이러한 IaC 템플릿을 CI/CD 파이프라인과 통합하여 배포 프로세스를 자동화합니다. AWS CodePipeline은 원활한 AWS 네이티브 솔루션을 제공하며, 다른 서드파티 CI/CD 솔루션을 사용할 수도 있습니다. 버전 관리 리포지토리가 변경될 때 활성화되는 파이프라인을 생성합니다. IaC 템플릿을 린팅하고 검증하는 단계를 포함하도록 파이프라인을 구성하고, 인프라를 스테이징 환경에 배포하고, 자동 테스트를 실행하고, 마지막으로 프로덕션에 배포합니다. 필요한 경우 승인 단계를 포함하여 변경 사항에 대한 관리를 유지합니다. 이 자동화된 파이프라인은 배포 속도를 높일 뿐만 아니라 환경 전반에서 일관성과 신뢰성을 높입니다.

필요에 따라 자동 스테일 아웃 및 스케일 인을 제공하도록 Amazon EC2 인스턴스, Amazon ECS 작업, IaC의 데이터베이스 복제본과 같은 리소스의 오토 스케일링을 구성합니다. 이 접근 방식은 수요에 따라 리소스 크기를 동적으로 조정하여 애플리케이션 가용성과 성능을 개선하고 비용을 최적화합니다. 지원되는 리소스 목록은 Amazon EC2 Auto Scaling 및 AWS Auto Scaling 섹션을 참조하세요.

### 구현 단계

1. 소스 코드 리포지토리를 생성하고 사용하여 인프라 구성을 제어하는 코드를 저장합니다. 이 리포지토리에 변경 사항을 커밋하여 원하는 지속적인 변경 사항을 반영합니다.
2. AWS CloudFormation과 같은 코드형 인프라 솔루션을 선택하여 인프라를 최신 상태로 유지하고 의도한 상태에서 불일치(드리프트)를 감지합니다.

3. IaC 플랫폼을 CI/CD 파이프라인과 통합하여 배포를 자동화합니다.
4. 리소스의 자동 크기 조정에 적합한 지표를 결정하고 수집합니다.
5. 워크로드 구성 요소에 적합한 스케일 아웃 및 스케일 인 정책을 사용하여 리소스의 자동 크기 조정을 구성합니다. 예측 가능한 사용 패턴을 위해 예약된 크기 조정을 사용하는 것을 고려해 보세요.
6. 배포를 모니터링하여 장애 및 회귀를 감지합니다. CI/CD 플랫폼 내에서 롤백 메커니즘을 구현하여 필요한 경우 변경 사항을 되돌립니다.

## 리소스

관련 문서:

- AWS Auto Scaling: How Scaling Plans Work
- AWS Marketplace: Auto Scaling과 함께 사용할 수 있는 제품
- DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리
- Using a load balancer with an Auto Scaling group
- What Is AWS Global Accelerator?
- What Is Amazon EC2 Auto Scaling?
- AWS Auto Scaling란 무엇입니까?
- What is Amazon CloudFront?
- What is Amazon Route 53?
- Elastic Load Balancing이란 무엇인가요?
- Network Load Balancer란 무엇인가요?
- Application Load Balancer란 무엇인가요?
- Integrating Jenkins with AWS CodeBuild and AWS CodeDeploy
- Creating a four stage pipeline with AWS CodePipeline

관련 비디오:

- Back to Basics: Deploy Your Code to Amazon EC2
- AWS Supports You | Starting Your Infrastructure as Code Solution Using AWS CloudFormation Templates
- Streamline Your Software Release Process Using AWS CodePipeline

- Monitor AWS Resources Using Amazon CloudWatch Dashboards
- Create Cross Account & Cross Region CloudWatch Dashboards | Amazon Web Services

## REL07-BP02 워크로드 장애 감지 시 리소스 확보

가용성이 영향을 받는 경우 필요에 따라 리소스를 사후에 확장하여 워크로드 가용성을 복원합니다.

먼저 상태 확인과 이러한 확인에 대한 기준을 구성하여 리소스 부족으로 인해 가용성이 영향을 받는 시기를 나타내야 합니다. 그런 다음 적절한 담당자에게 수동으로 리소스 규모를 조정하도록 알리거나 자동화를 시작하여 자동으로 리소스 규모를 조정합니다.

워크로드에 맞게 수동으로 규모를 조정할 수 있습니다. 예를 들어 AWS Management Console 또는 AWS CLI를 통해 DynamoDB 테이블의 처리량을 수정하거나 Auto Scaling 그룹의 EC2 인스턴스 수를 변경합니다. 하지만 가능하면 자동화를 사용해야 합니다(리소스를 확보하거나 조정할 때 자동화 사용 참조).

원하는 성과: 장애 또는 고객 경험 저하가 감지되면 가용성을 복원하기 위해 규모 조정 활동(자동 또는 수동)이 시작됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

### 구현 가이드

워크로드의 모든 구성 요소에 대한 관찰성 및 모니터링을 구현하여 고객 경험을 모니터링하고 장애를 감지합니다. 필요한 리소스의 규모를 조정하는 절차를 수동 또는 자동으로 정의합니다. 자세한 내용은 REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지를 참조하세요.

구현 단계

- 필요한 리소스 규모를 조정하는 절차를 수동 또는 자동으로 정의합니다.
  - 규모 조정 절차는 워크로드 내의 다양한 구성 요소가 어떻게 설계되었는지에 따라 달라집니다.
  - 규모 조정 절차는 사용되는 기본 기술에 따라서도 달라집니다.
    - AWS Auto Scaling을 사용하는 구성 요소는 규모 조정 계획을 사용하여 리소스 규모 조정을 위한 일련의 지침을 구성할 수 있습니다. AWS CloudFormation을 사용하거나 AWS 리소스에 태그를 추가하는 경우 애플리케이션별로 다양한 리소스 세트에 대한 크기 조정 계획을 설정할 수 있습니다. Auto Scaling은 각 리소스별로 맞춤화된 조정 전략에 대한 권장 사항을 제공합니다. 규모 조정 계획을 생성하면 Auto Scaling이 동적 규모 조정과 예측 규모 조정 방식을 결합하여 규모 조정 전략을 지원합니다. 자세한 내용은 How scaling plans work를 참조하세요.

- Amazon EC2 Auto Scaling을 사용하면 애플리케이션의 로드를 처리할 수 있는 정확한 수의 Amazon EC2 인스턴스를 유지할 수 있습니다. Auto Scaling 그룹이라는 EC2 인스턴스 모음을 생성합니다. 각 Auto Scaling 그룹의 최소 및 최대 인스턴스 수를 지정할 수 있으며 Amazon EC2 Auto Scaling은 그룹이 이러한 한도에 미달하거나 한도를 초과하지 않도록 합니다. 자세한 내용은 What is Amazon EC2 Auto Scaling?을 참조하세요.
- Amazon DynamoDB Auto Scaling은 Application Auto Scaling 서비스를 사용하여 프로비저닝된 처리 능력을 실제 트래픽 패턴에 따라 사용자 대신 동적으로 조정합니다. 따라서 테이블 또는 글로벌 보조 인덱스에 따라 할당된 읽기 및 쓰기 용량을 늘려 병목 현상 없이 갑작스러운 트래픽 증가를 처리할 수 있습니다. 자세한 내용은 DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리를 참조하세요.

## 리소스

관련 모범 사례:

- REL07-BP01 리소스를 확보하거나 조정할 때 자동화 사용
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지

관련 문서:

- AWS Auto Scaling: How Scaling Plans Work
- DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리
- What Is Amazon EC2 Auto Scaling?

# REL07-BP03 워크로드에 더 많은 리소스가 필요한 것으로 감지되면 리소스 확보

클라우드 컴퓨팅의 가장 중요한 기능 중 하나는 리소스를 동적으로 프로비저닝하는 기능입니다.

기존 온프레미스 컴퓨팅 환경에서는 피크 수요를 충족하기에 충분한 용량을 미리 식별하고 프로비저닝해야 합니다. 이는 비용이 많이 들고 워크로드의 피크 용량 요구 사항을 과소평가할 경우 가용성에 위험을 초래하기 때문에 문제가 됩니다.

클라우드에서는 이렇게 할 필요가 없습니다. 필요에 따라 컴퓨팅, 데이터베이스 및 기타 리소스 용량을 프로비저닝하여 현재 및 예상 수요를 충족할 수 있습니다. Amazon EC2 Auto Scaling 및 Application Auto Scaling과 같은 자동화된 솔루션은 지정한 지표를 기반으로 리소스를 온라인 상태로 만들 수 있습

니다. 이렇게 하면 규모 조정 프로세스가 더 쉽고 예측 가능하며, 항상 충분한 리소스를 사용할 수 있도록 하여 워크로드의 신뢰성을 크게 높일 수 있습니다.

원하는 성과: 수요에 맞게 컴퓨팅 및 기타 리소스의 자동 크기 조정을 구성합니다. 크기 조정 정책에 충분한 여유를 제공하여 추가 리소스를 온라인 상태로 가져오는 동안 트래픽 버스트를 허용합니다.

일반적인 안티 패턴:

- 확장 가능한 리소스를 일정 개수로 프로비저닝합니다.
- 실제 수요와 상관관계가 없는 크기 조정 지표를 선택합니다.
- 크기 조정 계획에 수요 버스트를 수용할 수 있는 충분한 여유를 제공하지 못합니다.
- 크기 조정 정책은 용량을 너무 늦게 추가하여 추가 리소스를 온라인 상태로 전환하는 동안 용량 소진 및 서비스 저하로 이어집니다.
- 최소 및 최대 리소스 수를 올바르게 구성하지 못하여 조정에 실패합니다.

이 모범 사례 확립의 이점: 워크로드의 고가용성을 제공하고 정의된 서비스 수준 목표(SLO)를 준수하려면 현재 수요를 충족할 수 있는 충분한 리소스를 확보하는 것이 중요합니다. 자동 크기 조정을 사용하면 현재 및 예측된 수요를 제공하기 위해 워크로드에 필요한 적절한 양의 컴퓨팅, 데이터베이스 및 기타 리소스를 제공할 수 있습니다. 피크 용량 요구 사항을 결정하고 리소스를 정적으로 할당하여 제공할 필요가 없습니다. 대신 수요가 증가함에 따라 더 많은 리소스를 할당하여 수용하고 수요가 감소한 후에는 리소스를 비활성화하여 비용을 절감할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

먼저 워크로드 구성 요소가 자동 크기 조정에 적합한지 확인합니다. 이러한 구성 요소는 동일한 리소스를 제공하고 동일하게 작동하기 때문에 수평 크기 조정이 가능하다고 합니다. 수평 크기 조정이 가능한 구성 요소의 예로는 동일하게 구성된 EC2 인스턴스, Amazon Elastic Container Service(Amazon ECS) 작업, Amazon Elastic Kubernetes Service(Amazon EKS)에서 실행되는 포드 등이 있습니다. 이러한 컴퓨팅 리소스는 일반적으로 로드 밸런서 뒤에 위치하며 복제본 이라고 합니다.

복제된 다른 리소스에는 데이터베이스 읽기 전용 복제본, Amazon DynamoDB 테이블 및 Amazon ElastiCache(Redis OSS) 클러스터가 포함될 수 있습니다. 지원되는 리소스의 전체 목록은 AWS services that you can use with Application Auto Scaling을 참조하세요.

컨테이너 기반 아키텍처의 경우 두 가지 방법으로 크기를 조정해야 할 수 있습니다. 먼저 수평적으로 크기 조정이 가능한 서비스를 제공하는 컨테이너의 크기를 조정해야 할 수 있습니다. 둘째, 컴퓨팅 리소스를 확장하여 새 컨테이너를 위한 공간을 확보해야 할 수 있습니다. 각 계층마다 다양한 자동 크

기 조정 메커니즘이 있습니다. ECS 작업의 크기를 조정하려면 Application Auto Scaling을 사용할 수 있습니다. Kubernetes 포드의 크기를 조정하려면 Horizontal Pod Autoscaler(HPA) 또는 Kubernetes Event-driven Autoscaling(KEDA)을 사용할 수 있습니다. 컴퓨팅 리소스를 확장하려면 ECS용 또는 Kubernetes용 용량 공급자를 사용하거나 Karpenter 또는 Cluster Autoscaler를 사용할 수 있습니다.

그런 다음 자동 크기 조정을 수행하는 방법을 선택합니다. 지표 기반 크기 조정, 예약된 크기 조정, 예측 크기 조정의 세 가지 주요 옵션이 있습니다.

지표 기반 크기 조정

지표 기반 크기 조정은 하나 이상의 크기 조정 지표 값을 기반으로 리소스를 프로비저닝합니다. 크기 조정 지표는 워크로드의 수요에 해당하는 지표입니다. 적절한 크기 조정 지표를 결정하는 좋은 방법은 비프로덕션 환경에서 로드 테스트를 수행하는 것입니다. 로드 테스트 중에 크기 조정 가능한 리소스 수를 고정하고 수요(예: 처리량, 동시성 또는 시뮬레이션된 사용자)를 천천히 증가시킵니다. 그런 다음 수요가 증가함에 따라 증가(또는 감소)하고 수요가 감소하면 반대로 감소(또는 증가)하는 지표를 찾습니다. 일반적인 크기 조정 지표에는 CPU 사용률, 작업 대기열 깊이(예: Amazon SQS 대기열), 활성 사용자 수 및 네트워크 처리량이 포함됩니다.

> **Note**
>
> AWS는 대부분의 애플리케이션에서 애플리케이션이 워밍업된 다음 일정한 값에 도달하면 메모리 사용률이 증가한다는 것을 확인했습니다. 수요가 감소하면 일반적으로 메모리 사용률이 병렬로 감소하지 않고 상승 상태로 유지됩니다. 메모리 사용률은 두 방향의 수요에 상응하지 않기 때문에, 즉 수요에 따라 증가하거나 감소하지 않기 때문에 자동 크기 조정을 위해 이 지표를 선택하기 전에 신중하게 고려하세요.

지표 기반 크기 조정은 잠정적인 작업입니다. 사용률 지표가 오토 스케일링 메커니즘으로 전파되는 데 몇 분 정도 걸릴 수 있으며, 이러한 메커니즘은 일반적으로 수요 증가의 명확한 신호를 기다렸다가 반응합니다. 그런 다음 오토 스케일러가 새 리소스를 생성하므로 완전히 서비스를 제공하는 데 시간이 더 걸릴 수 있습니다. 따라서 크기 조정 지표 목표를 전체 사용률에 너무 가깝게(예: CPU 사용률의 90%) 설정하지 않는 것이 중요합니다. 전체 사용률에 가깝게 설정하면 추가 용량이 온라인 상태가 되기 전에 기존 리소스 용량이 소진될 위험이 있습니다. 일반적으로 최적의 가용성을 위한 리소스 사용률 목표는 수요 패턴 및 추가 리소스를 프로비저닝하는 데 필요한 시간에 따라 50~70% 범위일 수 있습니다.

예약된 크기 조정

예약된 크기 조정은 달력 또는 시간대에 따라 리소스를 프로비저닝하거나 제거합니다. 주중 영업 시간 또는 세일 이벤트 중 피크 사용률과 같이 예측 가능한 수요가 있는 워크로드에 자주 사용됩니다.

Amazon EC2 Auto Scaling과 Application Auto Scaling 모두 예약된 크기 조정을 지원합니다. KEDA의 cron scaler는 Kubernetes 포드의 예약된 크기 조정을 지원합니다.

예측 크기 조정

예측 크기 조정은 기계 학습을 사용하여 예상 수요에 따라 리소스 크기를 자동으로 조정합니다. 예측 크기 조정은 제공하는 사용률 지표의 과거 값을 분석하고 미래 값을 지속적으로 예측합니다. 그런 다음 예측 값을 사용하여 리소스를 확장하거나 축소합니다. Amazon EC2 Auto Scaling은 예측 크기 조정을 수행할 수 있습니다.

구현 단계

1. 워크로드 구성 요소가 자동 크기 조정에 적합한지 확인합니다.
2. 지표 기반 크기 조정, 예약된 크기 조정 또는 예측 크기 조정 중에서 워크로드에 가장 적합한 크기 조정 메커니즘을 결정합니다.
3. 구성 요소에 적합한 자동 크기 조정 메커니즘을 선택합니다. Amazon EC2 인스턴스의 경우 Amazon EC2 Auto Scaling을 사용합니다. 다른 AWS 서비스의 경우 Application Auto Scaling을 사용합니다. Kubernetes 포드(예: Amazon EKS 클러스터에서 실행되는 포드)의 경우 Horizontal Pod Autoscaler(HPA) 또는 Kubernetes Event-driven Autoscaling(KEDA)을 고려합니다. Kubernetes 또는 EKS 노드의 경우 Karpenter 및 Cluster Auto Scaler(CAS)를 고려합니다.
4. 지표 또는 예약된 크기 조정의 경우 로드 테스트를 수행하여 워크로드에 적합한 크기 조정 지표와 목표 값을 결정합니다. 예약된 크기 조정의 경우 선택한 날짜 및 시간에 필요한 리소스 수를 결정합니다. 예상 피크 트래픽을 처리하는 데 필요한 최대 리소스 수를 결정합니다.
5. 위에서 수집한 정보를 기반으로 오토 스케일러를 구성합니다. 자세한 내용은 오토 스케일링 서비스의 설명서를 참조하세요. 최대 및 최소 크기 조정 제한이 올바르게 구성되었는지 확인합니다.
6. 크기 조정 구성이 예상대로 작동하는지 확인합니다. 비프로덕션 환경에서 로드 테스트를 수행하고 시스템이 어떻게 반응하는지 관찰하고 필요에 따라 조정합니다. 프로덕션에서 오토 스케일링을 활성화할 때는 예기치 않은 동작이 발생할 경우 알리도록 적절한 경보를 구성합니다.

## 리소스

관련 문서:

- What Is Amazon EC2 Auto Scaling?
- AWS 규범적 지침: Load testing applications
- AWS Marketplace: Auto Scaling과 함께 사용할 수 있는 제품
- DynamoDB Auto Scaling을 사용하여 자동으로 처리량 용량 관리

- Predictive Scaling for EC2, Powered by Machine Learning
- Scheduled Scaling for Amazon EC2 Auto Scaling
- Telling Stories About Little's Law

## REL07-BP04 워크로드 로드 테스트

확장 작업이 워크로드의 필요 사항을 충족하는지 측정하기 위한 로드 테스트 방식을 채택하세요.

지속적인 로드 테스트를 수행하는 것이 중요합니다. 로드 테스트를 통해 한계점을 찾고 워크로드의 성능을 테스트할 수 있습니다. AWS를 사용하면 프로덕션 워크로드의 규모를 모델링하는 임시 테스트 환경을 쉽게 설정할 수 있습니다. 클라우드에서는 온디맨드 방식으로 프로덕션 규모의 테스트 환경을 만들고, 테스트를 완료한 다음 해당 리소스를 폐기할 수 있습니다. 테스트 환경을 실행하는 동안에만 비용을 지불하면 되기 때문에 온프레미스 테스트 비용의 몇분의 일에 불과한 가격으로 실제 환경을 시뮬레이션할 수 있습니다.

또한 프로덕션에서의 로드 테스트는 프로덕션 시스템에 스트레스가 가해지는 게임 데이의 일부로 간주되어야 하며 고객 사용량이 적은 시간에는 모든 직원이 결과를 해석하고 발생하는 문제를 해결해야 합니다.

일반적인 안티 패턴:

- 구성이 프로덕션 환경과 동일하지 않은 배포에 대해 로드 테스트를 수행합니다.
- 전체 워크로드가 아니라 워크로드의 개별 부분에 대해서만 로드 테스트를 수행합니다.
- 대표적인 실제 요청 세트가 아니라 요청의 하위 세트를 사용하여 로드 테스트를 수행합니다.
- 예상 로드보다 작은 안전 계수로 로드 테스트를 수행합니다.

이 모범 사례 확립의 이점: 로드 시 장애가 발생하는 아키텍처의 구성 요소를 알고 있으며, 해당 로드에 곧 도달할 것임을 나타내는 지표를 식별하고 조사하여 문제를 해결할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

### 구현 가이드

- 로드 테스트를 수행하여 워크로드의 어느 측면에 용량을 추가 또는 제거해야 하는지 파악합니다. 로드 테스트에는 프로덕션 환경에서 수신하는 것과 유사한 대표 트래픽이 있어야 합니다. 계측한 지표를 모니터링하면서 로드를 늘려 리소스를 추가 또는 제거해야 하는 시점을 나타내는 지표를 결정합니다.

- Distributed Load Testing on AWS: 수천 명의 사용자가 접속한 상황을 시뮬레이션
  - 요청의 조합을 식별합니다. 다양한 요청 조합이 있을 수 있으므로 트래픽 조합을 식별할 때 다양한 시간 프레임을 살펴봐야 합니다.
  - 로드 드라이버를 구현합니다. 사용자 지정 코드, 오픈 소스 또는 상용 소프트웨어를 사용하여 로드 드라이버를 구현할 수 있습니다.
  - 처음에는 작은 용량을 사용하여 로드 테스트를 수행합니다. 인스턴스 또는 컨테이너 하나 정도의 작은 용량으로 로드를 유도하면 즉각적인 효과가 나타납니다.
  - 더 큰 용량에 대해 로드 테스트를 수행합니다. 분산된 로드에서는 효과가 다르게 나타나므로 가능한 한 제품 환경에 가깝게 테스트해야 합니다.

### 리소스

관련 문서:

- Distributed Load Testing on AWS: 수천 명의 사용자가 접속한 상황을 시뮬레이션
- 부하 테스트 애플리케이션

관련 비디오:

- AWS Summit ANZ 2023: Accelerate with confidence through AWS Distributed Load Testing

## 변경 구현

새로운 기능을 배포하고 워크로드와 운영 환경에서 적절한 패치가 적용된 알려진 소프트웨어를 실행하려면 변경 제어가 필요합니다. 이러한 변경이 제어되지 않으면 변경의 영향을 예측하거나 변경으로 인해 발생하는 문제를 해결하는 것이 어려워집니다.

위험을 최소화하기 위한 추가 배포 패턴

기능 플래그(기능 토글이라고도 함)는 애플리케이션의 구성 옵션입니다. 고객에게 특정 기능이 표시되지 않도록 기능을 해제한 상태로 소프트웨어를 배포할 수 있습니다. 그런 다음 카나리 배포에서와 같이 기능을 설정할 수도 있고, 변경 속도를 100%로 설정하여 효과를 표시할 수도 있습니다. 배포에 문제가 발생하더라도 롤백할 필요 없이 기능만 다시 해제하면 됩니다.

장애 격리 영역 배포:AWS AWS에서 배포와 관련하여 설정한 가장 중요한 규칙 중 하나는 한 리전에서 여러 가용 영역에 동시에 연결하지 않는 것입니다. 가용성 계산에서 가용 영역의 독립성을 유지하려면 이 규칙을 준수해야 합니다. 실제 배포에서도 이와 유사한 고려 사항을 포함하는 것이 좋습니다.

운영 준비 상태 검토(ORR)

AWS는 테스트의 완전성, 모니터링 기능 그리고 무엇보다도 애플리케이션 성능이 SLA를 충족하는지를 감사하고 서비스가 중단되거나 비정상적인 작업이 수행되는 경우 데이터를 제공하는 기능을 평가하는 운영 준비 검토를 수행하는 데 유용합니다. 공식 ORR은 초기 프로덕션 배포 전에 수행됩니다. AWS는 ORR을 주기적으로(매년 1회 또는 성능이 중요한 기간) 반복 수행하여 운영상의 기대치를 벗어난 경우(드리포트)가 없었는지를 확인합니다. 운영 준비 상태에 대한 자세한 내용은 AWS Well-Architected Framework의 운영 우수성 원칙을 참조하세요.

모범 사례

- REL08-BP01 배포와 같은 표준 활동에 런북 사용
- REL08-BP02 배포의 일부로 기능 테스트 통합
- REL08-BP03 배포의 일부로 복원력 테스트 통합
- REL08-BP04 변경 불가능한 인프라를 사용하여 배포
- REL08-BP05 자동화를 통한 변경 사항 배포

## REL08-BP01 배포와 같은 표준 활동에 런북 사용

런북은 특정 결과를 달성하기 위한 미리 정의된 절차입니다. 수동 또는 자동으로 표준 활동을 수행할 때 런북을 사용합니다. 워크로드 배포, 워크로드 패치 적용 또는 DNS 수정과 같은 활동이 여기에 포함됩니다.

예를 들어 배포 중에 롤백이 안전한지 확인하는 프로세스를 준비합니다. 서비스를 안정적으로 유지하려면 고객 중단 없이 배포를 롤백할 수 있는지 확인하는 것이 중요합니다.

런북 절차를 수행할 때는 유효하고 효과적인 수동 프로세스에서 시작하고, 이를 코드에 구현한 다음, 필요한 경우 자동으로 실행하도록 간접 호출합니다.

고도로 자동화된 정교한 워크로드의 경우에도 게임 데이를 실행하거나 엄격한 보고 및 감사 요구 사항을 충족할 때 런북을 유용하게 사용할 수 있습니다.

플레이북은 특정 인시던트에 대응하여 사용되며 런북은 특정 결과를 달성하기 위해 사용됩니다. 런북은 일상적인 활동에 대한 것이고, 플레이북은 일상적이지 않은 이벤트에 대응하는 데 사용되는 경우가 많습니다.

일반적인 안티 패턴:

- 프로덕션 환경에서 계획되지 않은 구성 변경을 수행합니다.

- 더 빠르게 배포하기 위해 계획의 단계를 건너뛰고, 그 결과 배포에 실패합니다.
- 변경 사항 되돌리기를 테스트하지 않고 변경을 수행합니다.

이 모범 사례 확립의 이점: 변경 계획을 효과적으로 수립하면 영향을 받는 모든 시스템을 파악할 수 있으므로 변경 사항을 성공적으로 실행할 수 있습니다. 테스트 환경에서 변경 사항을 검증하면 신뢰성을 높일 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

- 절차를 런북으로 문서화하면 적절하게 파악한 이벤트에 일관된 방식으로 신속하게 대응할 수 있습니다.
  - AWS Well-Architected Framework: Concepts: Runbook
- 코드형 인프라의 원칙을 사용해 인프라를 정의합니다. AWS CloudFormation 또는 신뢰할 수 있는 서드파티를 사용하여 인프라를 명확히 하면 버전 관리 소프트웨어를 통한 변경 사항의 각 버전을 차례대로 확인할 수 있습니다.
  - AWS CloudFormation(또는 신뢰할 수 있는 서드파티 제품)을 사용하여 인프라를 정의합니다.
    - AWS CloudFormation란 무엇입니까?
  - 우수한 소프트웨어 설계 원칙으로 분리된 단일 템플릿을 생성합니다.
    - 구현 시 권한, 템플릿 및 책임자를 결정합니다.
      - AWS Identity and Access Management를 통한 액세스 제어
    - Git과 같은 인기 있는 기술을 기반으로 하는 호스팅 소스 코드 관리 시스템을 사용하여 소스 코드 및 코드형 인프라(IaC) 구성을 저장합니다.

## 리소스

관련 문서:

- APN 파트너: 자동화된 배포 솔루션의 생성을 지원할 수 있는 파트너
- AWS Marketplace: 배포 자동화에 사용할 수 있는 제품
- AWS Well-Architected Framework: Concepts: Runbook
- AWS CloudFormation란 무엇입니까?

관련 예제:

- Automating operations with Playbooks and Runbooks

## REL08-BP02 배포의 일부로 기능 테스트 통합

단위 테스트 및 필수 기능을 검증하는 통합 테스트와 같은 기술을 사용합니다.

유닛 테스트는 코드의 가장 작은 기능 유닛을 테스트하여 동작을 검증하는 프로세스입니다. 통합 테스트에서는 각 애플리케이션 기능이 소프트웨어 요구 사항에 따라 작동하는지 검증합니다. 유닛 테스트는 애플리케이션의 일부를 개별적으로 테스트하는 데 중점을 두지만 통합 테스트는 부작용(예: 변경 작업을 통해 데이터가 변경될 때의 영향)을 고려합니다. 어느 경우든 테스트를 배포 파이프라인에 포함해야 하며, 성공 기준이 충족되지 않으면 파이프라인이 중단되거나 롤백됩니다. 이러한 테스트는 파이프라인에서 프로덕션 전에 준비되는 사전 프로덕션 환경에서 실행됩니다.

이러한 테스트는 구축 및 배포 작업의 일부로 자동으로 실행될 때 최상의 결과를 제공합니다. 예를 들어 개발자가 AWS CodePipeline을 사용하여 소스 리포지토리에 변경 사항을 커밋하면 CodePipeline이 변경 사항을 자동으로 감지합니다. 애플리케이션이 구축되고 유닛 테스트가 실행됩니다. 유닛 테스트를 통과한 후 구축된 코드를 테스트를 위해 스테이징 서버로 배포합니다. 스테이징 서버에서 CodePipeline은 통합이나 로드 테스트 같은 추가 테스트를 실행합니다. 이러한 테스트가 성공적으로 완료되면 CodePipeline은 테스트되고 승인된 코드를 프로덕션 인스턴스에 배포합니다.

원하는 성과: 자동화를 사용하여 유닛 테스트 및 통합 테스트를 수행하여 코드가 예상대로 작동하는지 검증합니다. 이러한 테스트가 배포 프로세스에 포함되며 테스트 실패 시 배포를 중단합니다.

일반적인 안티 패턴:

- 배포 타임라인을 가속화하기 위해 배포 프로세스 중에 테스트 실패 및 계획을 무시하거나 우회합니다.
- 배포 파이프라인 외부에서 수동으로 테스트를 수행합니다.
- 수동 긴급 워크플로를 통해 자동화의 테스트 단계를 건너뜁니다.
- 프로덕션 환경과 별로 유사하지 않은 환경에서 자동 테스트를 실행합니다.
- 유연성이 충분하지 않고 애플리케이션이 발전함에 따라 유지 관리, 업데이트 또는 크기 조정이 어려운 테스트 세트를 구축합니다.

이 모범 사례 확립의 이점: 배포 프로세스 중 자동 테스트는 문제를 조기에 포착하여 버그 또는 예상치 못한 동작으로 프로덕션으로 릴리스될 위험을 줄입니다. 유닛 테스트에서는 코드 동작이 원하는 대로 작동하는지, API 계약을 준수하는지 확인합니다. 통합 테스트에서는 시스템이 지정된 요구 사항에 따

라 작동하는지 확인합니다. 이러한 테스트 유형은 사용자 인터페이스, API, 데이터베이스 및 소스 코드와 같은 구성 요소의 의도된 작동 순서를 확인하는 데 사용됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

테스트 사례를 개발하여 코드를 지정하고 검증하는 테스트 기반 개발(TDD) 접근 방식을 채택합니다. 시작하려면 각 기능에 대한 테스트 사례를 생성합니다. 테스트가 실패하면 새 코드를 작성하여 테스트를 통과합니다. 이 접근 방식은 각 기능의 예상 결과를 검증하는 데 도움이 됩니다. 유닛 테스트를 실행하고 소스 코드 리포지토리에 코드를 커밋하기 전에 전달되는지 확인합니다.

CI/CD 파이프라인의 구축, 테스트 및 배포 단계의 일부로 유닛 및 통합 테스트를 모두 구현합니다. 테스트를 자동화하고 새 버전의 애플리케이션을 배포할 준비가 될 때마다 테스트를 자동으로 시작합니다. 성공 기준이 충족되지 않으면 파이프라인이 중지되거나 롤백됩니다.

애플리케이션이 웹 또는 모바일 앱인 경우 여러 데스크톱 브라우저 또는 실제 디바이스에서 자동 통합 테스트를 수행합니다. 이 접근 방식은 다양한 디바이스에서 모바일 앱의 호환성과 기능을 검증하는 데 특히 유용합니다.

구현 단계

1. 기능 코드(테스트 기반 개발, 즉 TDD)를 작성하기 전에 유닛 테스트를 작성합니다. 유닛 테스트 작성 및 실행이 비기능적 코딩 요구 사항이 되도록 코드 지침을 만듭니다.
2. 식별된 테스트 가능한 기능을 포함하는 자동화된 통합 테스트 세트를 생성합니다. 이러한 테스트는 사용자 상호 작용을 시뮬레이션하고 예상 결과를 검증해야 합니다.
3. 통합 테스트를 실행하는 데 필요한 테스트 환경을 생성합니다. 여기에는 프로덕션 환경을 유사하게 모방한 스테이징 또는 프로덕션 전 환경이 포함될 수 있습니다.
4. AWS CodePipeline 콘솔 또는 AWS Command Line Interface(CLI)를 사용하여 소스, 구축, 테스트 및 배포 단계를 설정합니다.
5. 코드를 구축하고 테스트한 후 애플리케이션을 배포합니다. AWS CodeDeploy는 스테이징(테스트) 및 프로덕션 환경에 배포할 수 있습니다. 이러한 환경에는 Amazon EC2 인스턴스, AWS Lambda 함수 또는 온프레미스 서버가 포함될 수 있습니다. 애플리케이션을 모든 환경에 배포하려면 동일한 배포 메커니즘을 사용해야 합니다.
6. 파이프라인의 진행 상황과 각 단계의 상태를 모니터링합니다. 품질 검사를 사용하여 테스트 상태에 따라 파이프라인을 차단합니다. 파이프라인 단계 장애 또는 파이프라인 완료에 대한 알림을 받을 수도 있습니다.

7. 테스트 결과를 지속적으로 모니터링하고 더 많은 주의가 필요한 패턴, 회귀 또는 영역을 찾습니다. 이 정보를 사용하여 테스트 세트를 개선하고, 더 강력한 테스트가 필요한 애플리케이션 영역을 식별하고, 배포 프로세스를 최적화합니다.

## 리소스

관련 모범 사례:

- REL07-BP04 워크로드 로드 테스트
- REL08-BP03 배포의 일부로 복원력 테스트 통합
- REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트

관련 문서:

- AWS Prescriptive Guidance: Test automation
- Continuous Delivery and Continuous Integration
- Indicators for functional testing
- Monitoring pipelines
- Use AWS CodePipeline with AWS CodeBuild to test code and run builds
- AWS Device Farm

# REL08-BP03 배포의 일부로 복원력 테스트 통합

장애 시나리오가 발생할 경우에 대비해 시스템에 장애를 의도적으로 도입하여 성능을 측정함으로써 복원력 테스트를 통합합니다. 복원력 테스트는 시스템에서 예상치 못한 장애를 식별하는 데 중점을 두기 때문에 일반적으로 배포 주기에 통합되는 단위 및 기능 테스트와는 다릅니다. 프로덕션 전 단계에서 복원력 테스트 통합을 시작하는 것이 안전하지만, 게임 데이의 일부로 프로덕션 환경에서 이러한 테스트를 구현한다는 목표를 설정하세요.

원하는 성과: 복원력 테스트는 프로덕션에서의 성능 저하를 견디는 시스템의 능력에 대해 확신을 얻는 데 도움이 됩니다. 실험을 통해 장애로 이어질 수 있는 약점을 식별하면 시스템을 개선하여 장애 및 성능 저하를 자동으로 효율적으로 완화할 수 있습니다.

일반적인 안티 패턴:

- 배포 프로세스의 관찰성과 모니터링이 부족합니다.

- 시스템 장애 해결을 위해 사람에게 의존합니다.
- 품질 분석 메커니즘이 열악합니다.
- 시스템의 알려진 문제에 초점을 맞추고, 알려지지 않은 문제를 식별하기 위한 실험은 부족합니다.
- 장애를 식별하지만 해결책은 없습니다.
- 조사 결과 및 런북에 대한 문서화 과정이 없습니다.

이 모범 사례 확립의 이점: 배포에 통합된 복원력 테스트는 시스템 내에서 미처 알아채지 못하다가 프로덕션에서 가동 중단으로 이어질 수 있는 알려지지 않은 문제를 식별하는 데 도움이 됩니다. 시스템에서 알려지지 않은 문제를 식별하면 조사 결과를 문서화하고, 테스트를 CI/CD 프로세스에 통합하며, 효율적이고 반복 가능한 메커니즘을 통해 완화를 간소화하는 런북을 빌드할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

시스템의 배포에 통합할 수 있는 가장 일반적인 복원력 테스트 형식은 재해 복구와 카오스 엔지니어링입니다.

- 중요한 배포 시 재해 복구 계획 및 표준 운영 절차(SOP)에 대한 업데이트를 포함하세요.
- 신뢰성 테스트를 자동화된 배포 파이프라인에 통합하세요. AWS Resilience Hub와 같은 서비스를 CI/CD 파이프라인에 통합하여 배포할 때마다 배포의 일부로 자동으로 평가되는 지속적인 복원력 평가를 설정할 수 있습니다.
- AWS Resilience Hub에서 애플리케이션을 정의하세요. 복원력 평가는 복구 절차를 애플리케이션에 대한 AWS Systems Manager 문서로 생성하고 권장되는 Amazon CloudWatch 모니터 및 경보의 목록을 제공할 수 있는 코드 스니펫을 생성합니다.
- DR 계획과 SOP가 업데이트되면 재해 복구 테스트를 완료하여 효과가 있는지 확인하세요. 재해 복구 테스트는 이벤트 발생 후 시스템을 복원하고 정상 운영 상태로 돌아갈 수 있는지를 결정하는 데 도움이 됩니다. 다양한 재해 복구 전략을 시뮬레이션하고 계획이 가동 시간 요구 사항을 충족하기에 충분한지 확인할 수 있습니다. 일반적인 재해 복구 전략에는 백업 및 복원, 파일럿 라이트, 수동 대기 방식, 예열 대기 방식, 상시 대기 방식, 액티브-액티브가 포함되며 비용과 복잡성이 각기 다릅니다. 재해 복구 테스트 전에 Recovery Time Objective(RTO) 및 Recovery Point Objective(RPO)를 정의하여 시뮬레이션할 전략의 선택지를 간소화하는 것이 좋습니다. AWS는 AWS Elastic Disaster Recovery와 같은 재해 복구 도구를 제공하여 계획과 테스트를 시작하는 데 도움을 줍니다.
- 카오스 엔지니어링 실험은 네트워크 중단 및 서비스 장애와 같은 시스템 장애를 초래합니다. 통제된 장애로 시뮬레이션하면 주입된 장애의 영향을 억제하면서 시스템의 취약성을 발견할 수 있습니다.

다른 전략과 마찬가지로 AWS Fault Injection Service와 같은 서비스를 사용하여 비프로덕션 환경에서 통제된 장애 시뮬레이션을 실행하면 프로덕션에 배포하기 전에 확신을 얻을 수 있습니다.

## 리소스

관련 문서:

- Experiment with failure using resilience testing to build recovery preparedness
- Continually assessing application resilience with AWS Resilience Hub and AWS CodePipeline
- Disaster recovery (DR) architecture on AWS, part 1: Strategies for recovery in the cloud
- Verify the resilience of your workloads using Chaos Engineering
- Principles of Chaos Engineering
- Chaos Engineering 워크숍

관련 비디오:

- AWS re:Invent 2020: Testing Resilience using Chaos Engineering
- Improve Application Resilience with AWS Fault Injection Service
- Prepare & Protect Your Applications From Disruption With AWS Resilience Hub

# REL08-BP04 변경 불가능한 인프라를 사용하여 배포

변경 불가능한 인프라는 프로덕션 워크로드의 현재 위치에서 업데이트, 보안 패치 또는 구성 변경이 발생하지 않도록 규정하는 모델입니다. 변경이 필요한 경우 아키텍처가 새 인프라에 구축되고 프로덕션 환경에 배포됩니다.

변경 불가능한 인프라 배포 전략을 따라 워크로드 배포의 신뢰성, 일관성 및 재현성을 높입니다.

원하는 성과: 변경 불가능한 인프라를 사용했을 때 워크로드 내에서 인프라 리소스를 실행하기 위한 인플레이스 수정이 허용되지 않습니다. 대신 변경이 필요한 경우 필요한 변경 사항이 모두 포함하도록 업데이트된 신규 인프라 리소스 세트를 기존 리소스와 병렬로 배포합니다. 이 배포는 자동으로 검증되며, 성공하면 트래픽이 점차 새로운 리소스 세트로 이동합니다.

이 배포 전략은 소프트웨어 업데이트, 보안 패치, 인프라 변경, 구성 업데이트, 애플리케이션 업데이트 등에 적용됩니다.

일반적인 안티 패턴:

- 실행 중인 인프라 리소스에 인플레이스 변경을 구현합니다.

이 모범 사례 확립의 이점:

- 환경 간 일관성 향상: 환경 간에 인프라 리소스의 차이가 없으므로 일관성이 향상되고 테스트가 간소화됩니다.
- 구성 드리프트 감소: 인프라 리소스를 버전이 관리되는 알려진 구성으로 대체하면 인프라가 테스트를 거쳐 신뢰할 수 있는 알려진 상태로 설정되므로 구성 드리프트가 발생하지 않습니다.
- 신뢰할 수 있는 원자 단위 배포: 배포가 성공적으로 완료되거나 변경 사항이 없으므로 배포 프로세스의 일관성과 신뢰성이 향상됩니다.
- 간소화된 배포: 업그레이드를 지원할 필요가 없으므로 배포가 간소화됩니다. 업그레이드는 단지 새로운 배포일 뿐입니다.
- 빠른 롤백 및 복구 프로세스를 통해 배포 안전성 개선: 이전 작업 버전이 변경되지 않으므로 배포가 더 안전합니다. 오류가 감지되면 롤백할 수 있습니다.
- 보안 태세 강화: 인프라 변경을 허용하지 않음으로써 원격 액세스 메커니즘(예: SSH)을 비활성화할 수 있습니다. 이렇게 하면 공격 벡터가 줄어들어 조직의 보안 태세를 개선할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

### 자동화

변경 불가능한 인프라 배포 전략을 정의할 때는 재현성을 높이고 인적 오류 가능성을 최소화하기 위해 최대한 자동화를 사용하는 것이 좋습니다. 자세한 내용은 REL08-BP05 자동화를 통한 변경 사항 배포 및 안전하고 간편한 배포 자동화를 참조하세요.

코드형 인프라(IaC)를 사용하면 인프라 프로비저닝, 오케스트레이션 및 배포 단계가 프로그래밍, 설명 및 선언적 방식으로 정의되고 소스 제어 시스템에 저장됩니다. 코드형 인프라를 활용하면 인프라 배포를 더 간단하게 자동화할 수 있고 인프라 불변성을 달성하는 데 도움이 됩니다.

### 배포 패턴

워크로드 변경이 필요한 경우 변경 불가능한 인프라 배포 전략에서는 필요한 모든 변경을 포함하여 새로운 인프라 리소스 세트를 배포해야 합니다. 이 새로운 리소스 세트는 사용자에게 미치는 영향을 최소화하는 롤아웃 패턴을 따르는 것이 중요합니다. 이 배포에는 두 가지 주요 전략이 있습니다.

카나리 배포: 일반적으로 단일 서비스 인스턴스(canary)에서 실행되는 새 버전으로 소수의 사용자를 연결하는 방식입니다. 그런 다음 생성되는 동작 변경 또는 오류를 면밀히 조사합니다. 중대한 문제가 발생하여 사용자에게 이전 버전을 다시 제공해야 하는 경우에는 Canary에서 트래픽을 제거할 수 있습니다. 배포가 정상적으로 진행되면 배포가 완료될 때까지 변경 사항에서 오류를 모니터링하면서 원하는 속도로 배포를 계속 진행할 수 있습니다. 카나리 배포를 허용하는 배포 구성을 사용하여 AWS CodeDeploy를 구성할 수 있습니다.

블루/그린 배포: 카나리 배포와 비슷합니다. 단, 전체 애플리케이션 플릿이 병렬로 배포됩니다. 이 패턴에서는 블루와 그린의 두 스택에서 번갈아 가며 배포를 수행합니다. 이 패턴에서도 새 버전으로 트래픽을 전송한 다음 배포에 문제가 발생하면 이전 버전으로 장애 복구할 수 있습니다. 일반적으로 모든 트래픽은 한 번에 전환되지만, Amazon Route 53의 가중치 기반 DNS 라우팅 기능을 사용하면 각 버전에 대한 트래픽의 일부를 사용하여 새 버전의 채택을 유도할 수도 있습니다. 블루/그린 배포를 지원하는 배포 구성을 사용하여 AWS CodeDeploy 및 AWS Elastic Beanstalk를 구성할 수 있습니다.

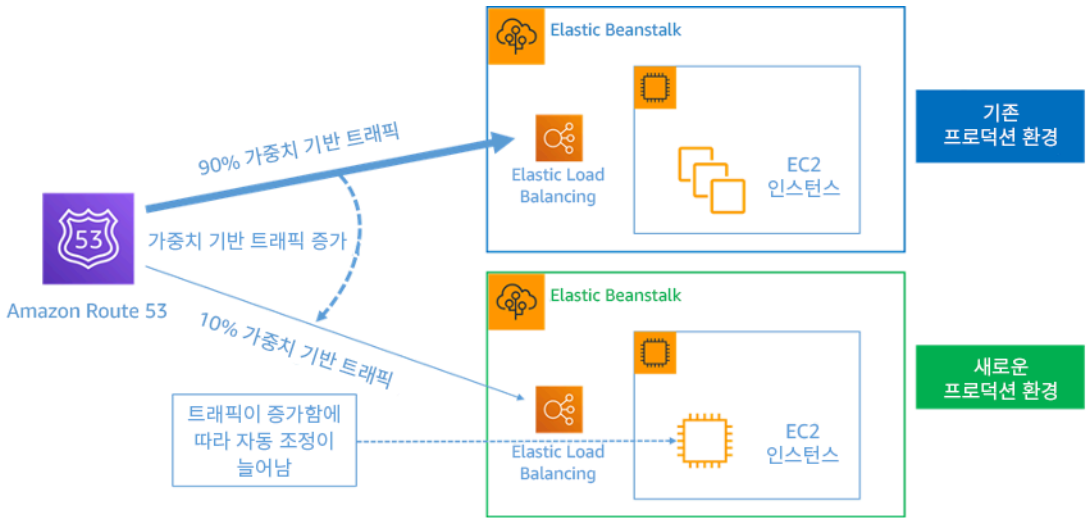



그림 8: AWS Elastic Beanstalk 및 Amazon Route 53을 사용한 블루/그린 배포

드리프트 감지

드리프트는 인프라 리소스의 상태 또는 구성이 예상과 달라지는 모든 변경으로 정의됩니다. 모든 유형의 관리되지 않는 구성 변경은 변경 불가능한 인프라의 개념에 어긋나므로 변경 불가능한 인프라를 성공적으로 구현하려면 이를 감지하고 수정해야 합니다.

구현 단계

- 실행 중인 인프라 리소스의 인플레이스 수정을 허용하지 않습니다.
  - AWS Identity and Access Management(IAM)를 사용하여 AWS에서 서비스 및 리소스에 액세스할 수 있는 사용자 또는 대상을 지정하고, 세분화된 권한을 중앙에서 관리하며, 액세스를 분석하여 AWS에서 권한을 세분화할 수 있습니다.

- 인프라 리소스 배포를 자동화하여 재현성을 높이고 인적 오류 가능성을 최소화합니다.
  - AWS의 DevOps 소개 백서에 설명되어 있는 것처럼 자동화는 AWS 서비스의 초석이며 모든 서비스, 기능 및 오퍼링에서 내부적으로 지원됩니다.
  - Amazon Machine Image(AMI)를 프리베이킹하면 시작 시간을 단축할 수 있습니다. EC2 Image Builder는 사용자 지정되고 안전한 최신 Linux 또는 Windows 사용자 지정 AMI의 생성, 유지 관리, 검증, 공유 및 배포를 자동화하는 데 도움이 되는 완전관리형 AWS 서비스입니다.
  - 자동화를 지원하는 서비스의 예를 몇 가지 들자면 다음과 같습니다.
    - AWS Elastic Beanstalk는 Java, .NET, PHP, Node.js, Python, Ruby, Go, Docker를 사용하여 개발된 웹 애플리케이션 및 서비스를 Apache, NGINX, Passenger, IIS와 같은 친숙한 서버에 빠르게 배포하고 규모를 조정하기 위한 서비스입니다.
    - AWS Proton은 인프라 프로비저닝, 코드 배포, 모니터링 및 업데이트를 위해 개발 팀에 필요한 모든 도구를 플랫폼 팀이 연결하고 조정할 수 있도록 지원합니다. AWS Proton은 서버리스 및 컨테이너 기반 애플리케이션의 코드형 인프라 자동 프로비저닝 및 배포를 지원합니다.
  - 코드형 인프라를 활용하면 인프라 배포를 쉽게 자동화할 수 있고 인프라 불변성을 달성하는 데 도움이 됩니다. AWS는 프로그래밍, 설명 및 선언적 방식으로 인프라를 생성, 배포 및 유지 관리할 수 있는 서비스를 제공합니다.
    - AWS CloudFormation은 개발자가 질서 있고 예측 가능한 방식으로 AWS 리소스를 만들 수 있도록 도와줍니다 리소스는 JSON 또는 YAML 형식을 사용하여 텍스트 파일로 작성됩니다. 템플릿에는 생성 및 관리되는 리소스 유형에 따라 특정 구문과 구조가 필요합니다. 코드 편집기를 사용하여 JSON 또는 YAML로 리소스를 작성하고 이를 버전 관리 시스템에 체크인하면 AWS CloudFormation이 명시된 서비스를 안전하고 반복 가능한 방식으로 구축합니다.
    - AWS Serverless Application Model(AWS SAM)은 AWS에서 서버리스 애플리케이션을 구축하는 데 사용할 수 있는 오픈 소스 프레임워크입니다. AWS SAM은 다른 AWS 서비스와 통합되며 AWS CloudFormation의 확장 기능입니다.
    - AWS 클라우드 개발 키트 (AWS CDK)는 익숙한 프로그래밍 언어를 사용하여 클라우드 애플리케이션 리소스를 모델링하고 프로비저닝하기 위한 오픈 소스 소프트웨어 개발 프레임워크입니다. TypeScript, Python, Java 및 .NET을 사용하여 애플리케이션 인프라를 모델링하는 데 AWS CDK를 사용할 수 있습니다. AWS CDK는 백그라운드에서 AWS CloudFormation을 사용하여 안전하고 반복 가능한 방식으로 리소스를 프로비저닝합니다.
    - AWS Cloud Control API는 개발자들이 쉽고 일관된 방식으로 클라우드 인프라를 관리할 수 있도록 CRUDL(Create, Read, Update, Delete, List) API로 구성된 일반적인 세트를 제공합니다. 개발자는 Cloud Control API를 사용하여 AWS 및 서드파티 서비스의 수명 주기를 일관적으로 관리할 수 있습니다.
- 사용자에게 미치는 영향을 최소화하는 배포 패턴을 구현합니다.

- 카나리 배포:
  - API Gateway Canary 릴리스 배포 설정
  - Create a pipeline with canary deployments for Amazon ECS using AWS App Mesh
- 블루/그린 배포: Blue/Green Deployments on AWS whitepaper에서는 블루/그린 배포 전략을 구현하기 위한 기술 예제를 설명합니다.
- 구성 또는 상태 드리프트를 감지합니다. 자세한 내용은 스택 및 리소스에 대한 비관리형 구성 변경 감지를 참조하세요.

## 리소스

관련 모범 사례:

- REL08-BP05 자동화를 통한 변경 사항 배포

관련 문서:

- 안전하고 간편한 배포 자동화
- Leveraging AWS CloudFormation to create an immutable infrastructure at Nubank
- 코드형 인프라
- Implementing an alarm to automatically detect drift in AWS CloudFormation stacks

관련 비디오:

- AWS re:Invent 2020: Reliability, consistency, and confidence through immutability

# REL08-BP05 자동화를 통한 변경 사항 배포

배포 및 패치 적용이 자동화되므로 부정적인 영향이 제거됩니다.

프로덕션 시스템을 변경하는 것은 조직에서 가장 위험 부담이 큰 영역에 속합니다. 배포는 소프트웨어를 통해 해결할 수 있는 업무상의 문제와 더불어 해결해야 하는 가장 중요한 문제로 간주됩니다. 오늘날에는 배포 관련 문제를 해결하려면 운영 과정에서 해당하는 모든 영역(변경 사항 테스트 및 배포, 용량 추가 또는 제거, 데이터 마이그레이션 포함)에 자동화를 사용해야 합니다.

원하는 성과: 광범위한 프로덕션 전 테스트, 자동 롤백, 시차를 둔 프로덕션 배포를 통해 릴리스 프로세스에 자동화된 배포 안전을 구축합니다. 이러한 자동화를 통해 배포 실패로 인한 프로덕션 환경의 잠재

적 영향을 최소화할 수 있으며, 개발자는 더 이상 프로덕션으로의 배포를 적극적으로 관찰할 필요가 없습니다.

일반적인 안티 패턴:

- 수동 변경을 수행합니다.
- 수동 비상 워크플로를 통해 자동화의 단계를 건너뜁니다.
- 일정을 앞당기기 위해 정해진 계획과 프로세스를 따르지 않습니다.
- 베이크 소요 시간을 허용하지 않고 신속한 후속 배포를 수행합니다.

이 모범 사례 확립의 이점: 자동화를 사용하여 모든 변경 사항을 배포하면 인적 오류가 발생할 가능성을 없애고 프로덕션을 변경하기 전에 테스트하는 기능을 제공합니다. 프로덕션 푸시 전에 이 프로세스를 수행하면 계획이 완전한지 확인할 수 있습니다. 또한 릴리스 프로세스로의 자동 롤백을 통해 프로덕션 문제를 식별하고 워크로드를 이전의 작동 상태로 되돌릴 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

배포 파이프라인을 자동화하세요. 배포 파이프라인을 사용하면 이상 징후의 자동 테스트 및 탐지를 간접 호출하여 프로덕션 배포 전 특정 단계에서 파이프라인을 중지하거나 변경 사항을 자동으로 롤백할 수 있습니다. 여기에 핵심은 통합 및 지속적 전달/배포(CI/CD) 문화를 채택하는 것입니다. 이 방식을 사용하면 커밋 또는 코드 변경이 구축 및 테스트 단계부터 프로덕션 환경에서의 배포에 이르기까지 다양한 자동화된 단계 게이트를 통과합니다.

일반적인 통념으로는 가장 어려운 작업 절차에 사람을 투입하는 것이 좋다고 하지만 바로 그런 이유로 가장 어려운 절차를 자동화하는 것이 좋습니다.

구현 단계

다음 단계에 따라 배포를 자동화하여 수동 작업을 제거할 수 있습니다.

- 코드를 안전하게 저장하도록 코드 리포지토리 설정: Git과 같은 인기 있는 기술을 기반으로 하는 호스팅 소스 코드 관리 시스템을 사용하여 소스 코드 및 코드형 인프라(IaC) 구성을 저장합니다.
- 소스 코드를 컴파일하고, 테스트를 실행하며, 배포 아티팩트를 생성하도록 지속적 통합 서비스 구성: 이를 위한 빌드 프로젝트를 설정하려면 Getting started with AWS CodeBuild using the console을 참조하세요.
- 오류가 발생하기 쉬운 수동 배포에 의존하지 않고 애플리케이션 배포를 자동화하고 애플리케이션 업데이트의 복잡성을 처리하는 배포 서비스 설정: AWS CodeDeploy는 Amazon EC2, AWS Fargate,

AWS Lambda, 온프레미스 서버와 같은 다양한 컴퓨팅 서비스에 대한 소프트웨어 배포를 자동화합니다. 이러한 단계를 구성하려면 Getting started with CodeDeploy를 참조하세요.

- 더 빠르고 안정적인 애플리케이션 및 인프라 업데이트를 위해 릴리스 파이프라인을 자동화하는 지속적 전달 서비스 설정: 릴리스 파이프라인을 자동화하는 데 도움이 되도록 AWS CodePipeline 사용을 고려해 보세요. 자세한 내용은 CodePipeline 자습서를 참조하세요.

## 리소스

관련 모범 사례:

- OPS05-BP04 구축 및 배포 관리 시스템 사용
- OPS05-BP10 통합 및 배포 완전 자동화
- OPS06-BP02 테스트 배포
- OPS06-BP04 테스트 및 롤백 자동화

관련 문서:

- Continuous Delivery of Nested AWS CloudFormation Stacks Using AWS CodePipeline
- APN 파트너: 자동화된 배포 솔루션의 생성을 지원할 수 있는 파트너
- AWS Marketplace: 배포 자동화에 사용할 수 있는 제품
- Automate chat messages with webhooks.
- Amazon Builders' Library: 배포 중 롤백 안전 보장
- Amazon Builders' Library: 지속적 전달을 통한 신속한 배포
- AWS CodePipeline란 무엇입니까?
- What Is CodeDeploy?
- AWS Systems Manager Patch Manager
- What is Amazon SES?
- What is Amazon Simple Notification Service?

관련 비디오:

- AWS Summit 2019: CI/CD on AWS

# 장애 관리

> 장애는 기정 사실이며 시간이 지나면 라우터부터 하드 디스크, 운영 체제부터 메모리 디바이스 오류로 인한 TCP 패킷 손상, 일시적 오류부터 영구적 장애 등 모든 것에서 결국 장애가 발생하기 마련이다. 오류는 최고 품질의 하드웨어를 사용하든 가장 저렴한 구성 요소를 사용하든 기정 사실입니다. - Werner Vogels, Amazon.com의 CTO

온프레미스 데이터 센터에서는 하위 수준의 하드웨어 구성 요소 장애가 매일 발생합니다. 그러나 클라우드에서는 이러한 유형의 장애 대부분으로부터 보호되어야 합니다. 예를 들어 Amazon EBS 볼륨은 단일 구성 요소에 장애가 발생할 경우 사용자를 보호하기 위해 자동으로 복제되는 특정 가용 영역에 배치됩니다. 모든 EBS 볼륨은 99.999%의 가용성을 제공하도록 설계되었습니다. Amazon S3 객체는 최소 3개의 가용 영역에 걸쳐 저장되어 지정된 기간(1년)에 99.999999999%의 객체 내구성을 제공합니다. 어떤 클라우드 제공업체를 사용하든 워크로드에 영향을 주는 장애가 발생할 가능성이 있습니다. 따라서 워크로드 신뢰성을 유지하려면 복원력을 구현하기 위한 조치를 취해야 합니다.

여기에 설명된 모범 사례를 적용하기 위한 전제 조건은 워크로드의 설계, 구현 및 운영에 관여하는 직원들이 비즈니스 목표와 신뢰성 목표를 인지하고 이를 달성해야 한다는 것입니다. 이러한 직원들은 이러한 신뢰성 요구 사항을 숙지하고 배워야 합니다.

다음 섹션에서는 장애 관리를 통해 워크로드에 미치는 영향을 방지하는 모범 사례를 설명합니다.

주제

- 데이터 백업
- 장애 격리를 사용하여 워크로드 보호
- 구성 요소 장애를 견디도록 워크로드 설계
- 신뢰성 테스트
- 재해 복구(DR) 계획

## 데이터 백업

Recovery Time Objective(RTO) 및 Recovery Point Objective(RPO)에 대한 요구 사항을 충족하도록 데이터, 애플리케이션 및 구성을 백업합니다.

모범 사례

- REL09-BP01 백업해야 하는 모든 데이터 확인 및 백업 또는 소스에서 데이터 복제
- REL09-BP02 백업 보안 및 암호화
- REL09-BP03 자동으로 데이터 백업 수행
- REL09-BP04 백업 무결성 및 프로세스를 확인하기 위해 데이터의 주기적인 복구 수행

## REL09-BP01 백업해야 하는 모든 데이터 확인 및 백업 또는 소스에서 데이터 복제

워크로드에 사용되는 서비스 및 리소스의 백업 기능을 숙지하고 사용합니다. 대부분의 서비스는 워크로드 데이터를 백업할 수 있는 기능을 제공합니다.

원하는 성과: 중요도에 따라 데이터 소스를 식별하고 분류합니다. 그런 다음 RPO에 따라 데이터 복구 전략을 수립합니다. 이 전략에는 데이터 소스 백업 또는 다른 소스에서 데이터를 복제하는 기능이 포함됩니다. 데이터가 손실될 경우 구현된 전략에 따라 정의된 RPO 또는 RTO 내에 복구 또는 데이터 재현이 가능합니다.

클라우드 성숙도 단계: 기초

일반적인 안티 패턴:

- 워크로드의 모든 데이터 소스와 그 중요도를 알지 못합니다.
- 중요한 데이터 소스의 백업을 생성하지 않습니다.
- 중요도를 기준으로 사용하지 않고 일부 데이터 소스의 백업만 생성합니다.
- RPO가 정의되지 않았거나 백업 주기가 RPO에 부합하지 않습니다.
- 백업이 필요한지 또는 데이터를 다른 소스에서 복제할 수 있는지 평가하지 않습니다.

이 모범 사례 확립의 이점: 백업이 필요한 지점을 파악하고 백업 생성을 위한 메커니즘을 구현하거나 외부 소스에서 데이터를 복제할 수 있는 경우 중단 시 데이터를 복원하고 복구하는 역량이 향상됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

### 구현 지침

모든 AWS 데이터 스토어는 백업 기능을 제공합니다. Amazon RDS 및 Amazon DynamoDB와 같은 서비스는 추가적으로 시점 복구(PITR)를 활성화하는 자동화된 백업을 지원합니다. 따라서 현재 시간으로부터 최대 5분 전까지 원하는 시점으로 백업을 복원할 수 있습니다. 많은 AWS 서비스는 백업을 다른

AWS 리전으로 복사할 수 있는 기능을 제공합니다. AWS Backup은 AWS 서비스 전체에서 데이터 보호를 중앙 집중화하고 자동화하는 기능을 제공하는 도구입니다. AWS Elastic Disaster Recovery를 사용하면 초 단위로 측정되는 Recovery Point Objective(RPO)를 사용하여 전체 서버 워크로드를 복사하고 온프레미스, 크로스 AZ 또는 크로스 리전에서 지속적인 데이터 보호를 유지할 수 있습니다.

Amazon S3는 자체 관리형 및 AWS 관리형 데이터 소스의 백업 목적지로 사용할 수 있습니다. Amazon EBS, Amazon RDS, Amazon DynamoDB와 같은 AWS 서비스에는 기본적으로 백업을 생성하는 기능이 내장되어 있습니다. 서드파티 백업 소프트웨어를 사용할 수도 있습니다.

온프레미스 데이터는 AWS Storage Gateway 또는 AWS DataSync를 사용하여 AWS 클라우드에 백업할 수 있습니다. Amazon S3 버킷은 AWS에 이 데이터를 저장하는 데 사용할 수 있습니다. Amazon S3는 데이터 스토리지 비용을 줄이기 위해 Amazon S3 Glacier 또는 S3 Glacier Deep Archive와 같은 여러 스토리지 계층을 제공합니다.

다른 소스에서 데이터를 재현하여 데이터 복구 요구 사항을 충족할 수 있습니다. 예를 들어 Amazon ElastiCache 복제본 노드 또는 Amazon RDS 읽기 복제본은 기본 항목이 손실된 경우 데이터를 복제하는 데 사용할 수 있습니다. 이와 같은 소스를 사용하여 Recovery Point Objective(RPO) 및 Recovery Time Objective(RTO)를 충족할 수 있는 경우 백업이 필요하지 않을 수 있습니다. 또 다른 예로 Amazon EMR로 작업하는 경우 Amazon S3에서 Amazon EMR로 데이터를 복제할 수 있는 한, HDFS 데이터 스토어를 백업하지 않아도 됩니다.

백업 전략을 선택할 때는 데이터 복구에 걸리는 시간을 고려하시기 바랍니다. 데이터를 복구하는 데 필요한 시간은 백업의 유형(백업 전략의 경우) 또는 데이터 복제 메커니즘의 복잡성에 따라 달라집니다. 이 시간은 워크로드의 RTO 이내여야 합니다.

구현 단계

1. 워크로드의 모든 데이터 소스를 식별합니다. 데이터는 데이터베이스, 볼륨, 파일 시스템, 로깅 시스템 및 객체 스토리지와 같은 여러 리소스에 저장할 수 있습니다. 데이터가 저장된 다양한 AWS 서비스와 이러한 서비스가 제공하는 백업 기능에 대한 관련 문서를 찾으려면 리소스 섹션을 참조하세요.
2. 중요도에 따라 데이터 소스를 분류합니다. 데이터세트마다 워크로드의 중요도가 다르므로 복원력에 대한 요구 사항도 달라집니다. 예를 들어, 어떤 데이터는 매우 중요하며 0에 가까운 RPO가 필요하지만 어떤 데이터는 덜 중요하며 더 높은 RPO와 일부 데이터 손실을 용인할 수 있습니다. 마찬가지로, 데이터세트마다 RTO 요구 사항이 다릅니다.
3. AWS 또는 서드파티 서비스를 사용하여 데이터 백업을 생성합니다. AWS Backup은 AWS에서 다양한 데이터 소스의 백업을 생성할 수 있는 관리형 서비스입니다. AWS Elastic Disaster Recovery에서는 AWS 리전에 대한 자동화된 1초 미만의 데이터 복제를 처리합니다. 이런 AWS 서비스의 대부분은 백업을 생성하는 기능이 기본적으로 내장되어 있습니다. AWS Marketplace에도 이런 기능을 제

공하는 솔루션이 많이 있습니다. 다양한 AWS 서비스에서 데이터 백업을 생성하는 방법에 대한 자세한 내용은 아래 나열된 리소스를 참조하세요.

4. 백업되지 않은 데이터에 대해서는 데이터 재현 메커니즘을 설정합니다. 여러 가지 이유로 다른 소스에서 복제될 수 있는 데이터는 백업하지 않기로 결정할 수 있습니다. 백업을 저장하는 데는 비용이 들기 때문에 백업을 생성하기보다는 필요할 때 다른 소스에서 데이터를 복제하는 것이 더 저렴한 상황이 있을 수도 있습니다. 또는 백업을 복원하는 작업이 다른 소스에서 데이터를 복제하는 것보다 더 오래 걸려 RTO를 준수할 수 없을 수도 있습니다. 그런 경우에는 장단점을 비교하여 데이터 복구가 필요할 때 다른 소스에서 데이터를 복제하는 방법에 대한 프로세스를 효과적으로 정의하여 수립해야 합니다. 예를 들어 Amazon S3의 데이터를 데이터 웨어하우스(예: Amazon Redshift) 또는 MapReduce 클러스터(예: Amazon EMR)로 로드하여 해당 데이터를 분석하는 경우 이 사례는 다른 소스에서 복제할 수 있는 데이터 예제에 해당할 수 있습니다. 이러한 분석 결과가 어딘가에 저장되어 있거나 재현될 수만 있다면 데이터 웨어하우스 또는 MapReduce 클러스터의 장애로 인해 데이터 손실이 발생하지 않습니다. 소스에서 복제할 수 있는 다른 예로는 캐시(예: Amazon ElastiCache) 또는 RDS 읽기 전용 복제본이 있습니다.
5. 데이터 백업 주기를 설정합니다. 데이터 소스의 백업을 생성하는 일은 주기적으로 이루어져야 하며 빈도는 RPO에 따라 다릅니다.

구현 계획의 작업 수준: 보통

## 리소스

관련 모범 사례:

REL13-BP01 가동 중단 시간 및 데이터 손실 시의 복구 목표 정의

REL13-BP02 복구 목표 달성을 위해 정의된 복구 전략 사용

관련 문서:

- AWS Backup란 무엇입니까?
- What is AWS DataSync?
- What is Volume Gateway?
- APN 파트너: 백업을 지원할 수 있는 파트너
- AWS Marketplace: 백업에 사용할 수 있는 제품
- Amazon EBS 스냅샷
- Backing Up Amazon EFS

- Backing up Amazon FSx for Windows File Server
- ElastiCache for Redis 백업 및 복원
- Creating a DB Cluster Snapshot in Neptune
- DB 스냅샷 생성
- Creating an EventBridge Rule That Triggers on a Schedule
- Amazon S3에서 크로스 리전 복제
- EFS-to-EFS AWS Backup
- Exporting Log Data to Amazon S3
- 객체 수명 주기 관리
- DynamoDB에 대한 온디맨드 백업 및 복원
- DynamoDB의 특정 시점으로 복구
- Working with Amazon OpenSearch Service Index Snapshots
- What is AWS Elastic Disaster Recovery?

관련 비디오:

- AWS re:Invent 2021 - Backup, disaster recovery, and ransomware protection with AWS
- AWS Backup Demo: Cross-Account and Cross-Region Backup
- AWS re:Invent 2019: Deep dive on AWS Backup, ft. Rackspace (STG341)

## REL09-BP02 백업 보안 및 암호화

인증 및 권한 부여를 사용하여 백업에 대한 액세스를 제어하고 감지합니다. 백업의 데이터 무결성이 침해되었을 경우 암호화 기법을 사용하여 예방 및 감지합니다.

일반적인 안티 패턴:

- 백업과 복원 자동화에 대한 액세스 권한을 데이터에 대한 액세스 권한과 동일하게 설정합니다.
- 백업을 암호화하지 않습니다.

이 모범 사례 확립의 이점: 백업의 보안을 유지하면 데이터 변조가 방지되며, 데이터 암호화는 데이터가 실수로 노출될 경우 해당 데이터에 대한 액세스를 방지합니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

AWS Identity and Access Management(IAM) 등의 인증 및 권한 부여를 사용하여 백업에 대한 액세스를 제어하고 감지합니다. 백업의 데이터 무결성이 침해되었을 경우 암호화 기법을 사용하여 예방 및 감지합니다.

Amazon S3는 저장 데이터 암호화를 위한 다양한 방법을 지원합니다. Amazon S3는 서버 측 암호화를 사용하여 객체를 암호화되지 않은 데이터로 수락한 다음 저장 시 암호화합니다. 클라이언트측 암호화를 사용하면 데이터가 Amazon S3로 전송되기 전에 워크로드 애플리케이션에서 데이터가 암호화됩니다. 두 방법 모두 AWS Key Management Service(AWS KMS)를 사용하여 데이터 키를 생성 및 저장하거나 사용자가 관리하는 자체 키를 제공할 수 있습니다. AWS KMS를 사용하면 IAM을 사용하여 데이터 키와 해독된 데이터에 접근할 수 있는 사용자와 접근할 수 없는 사용자에 대한 정책을 설정할 수 있습니다.

Amazon RDS의 경우 데이터베이스를 암호화하도록 선택하면 백업도 암호화됩니다. DynamoDB 백업은 항상 암호화됩니다. AWS Elastic Disaster Recovery를 사용하면 전송 중이거나 저장된 모든 데이터가 암호화됩니다. Elastic Disaster Recovery를 사용하면 기본 Amazon EBS 암호화 볼륨 암호화 키 또는 사용자 지정 고객 관리형 키를 사용하여 저장 데이터를 암호화할 수 있습니다.

구현 단계

1. 각 데이터 스토어에서 암호화를 사용합니다. 소스 데이터가 암호화되면 백업도 암호화됩니다.
   - Amazon RDS에서 암호화를 사용합니다. RDS 인스턴스를 생성할 때 AWS Key Management Service를 사용하여 저장 데이터의 암호화를 구성할 수 있습니다.
   - Amazon EBS 볼륨에서 암호화를 사용합니다. 볼륨 생성 시 기본 암호화를 구성하거나 고유한 키를 지정할 수 있습니다.
   - 필요한 Amazon DynamoDB 암호화를 사용합니다. DynamoDB는 모든 저장 데이터를 암호화합니다. 계정에 저장된 키를 지정하여 AWS 소유 AWS KMS 키 또는 AWS 관리형 KMS 키를 사용할 수 있습니다.
   - Amazon EFS에 저장된 데이터를 암호화합니다. 파일 시스템을 생성할 때 암호화를 구성합니다.
   - 소스 및 대상 리전에 암호화를 구성합니다. KMS에 저장된 키를 사용하여 Amazon S3에서 저장 데이터 암호화를 구성할 수 있지만 키는 리전별로 다릅니다. 복제를 구성할 때 대상 키를 지정할 수 있습니다.
   - 기본 또는 사용자 지정 Amazon EBS 암호화를 Elastic Disaster Recovery에 대해 사용할지 여부를 선택합니다. 이 옵션은 스테이징 영역 서브넷 디스크와 복제된 디스크에 복제된 저장 데이터를 암호화합니다.

2. 백업에 액세스할 수 있는 최소 권한을 구현합니다. 보안 모범 사례에 따라 백업, 스냅샷 및 복제본에 대한 액세스를 제한합니다.

## 리소스

관련 문서:

- AWS Marketplace: 백업에 사용할 수 있는 제품
- Amazon EBS Encryption
- Amazon S3: 암호화로 데이터 보호
- CRR 추가 구성: AWS KMS에 저장된 암호화 키를 사용하는 서버 측 암호화(SSE)로 생성된 객체 복제
- DynamoDB 저장 데이터 암호화
- Amazon RDS 리소스 암호화
- Encrypting Data and Metadata in Amazon EFS
- Encryption for Backups in AWS
- 암호화된 테이블 관리
- 보안 원칙 - AWS Well-Architected Framework
- What is AWS Elastic Disaster Recovery?

# REL09-BP03 자동으로 데이터 백업 수행

Recovery Point Objective(RPO)에 정의된 정기적인 일정 또는 데이터세트의 변경에 따라 백업이 자동으로 생성되도록 구성합니다. 적은 데이터 손실을 요구하는 중요한 데이터세트는 자주 자동으로 백업되어야 합니다. 반면 일부 손실은 용인하는 중요도가 상대적으로 낮은 데이터의 경우 더 낮은 빈도로 백업할 수 있습니다.

원하는 성과: 정해진 주기로 데이터 소스의 백업을 생성하는 자동화된 프로세스.

일반적인 안티 패턴:

- 수동으로 백업을 수행합니다.
- 백업을 지원하는 리소스를 사용하지만 자동화 대상에 백업을 포함하지 않습니다.

이 모범 사례 확립의 이점: 백업을 자동화하면 RPO를 기준으로 주기적으로 백업이 생성되며, 생성되지 않은 경우 알림을 보냅니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 지침

AWS Backup은 다양한 AWS 데이터 소스의 자동 데이터 백업을 생성하는 데 사용할 수 있습니다. Amazon RDS 인스턴스는 5분마다 거의 지속적으로 백업할 수 있으며 Amazon S3 객체는 15분마다 거의 지속적으로 백업될 수 있으므로 백업 기록에서 특정 시점으로 시점 복구(PITR)가 가능합니다. Amazon EBS 볼륨, Amazon DynamoDB 테이블 또는 Amazon FSx 파일 시스템과 같은 다른 AWS 데이터 소스의 경우 AWS Backup은 최대 빈도 1시간 간격으로 자동화된 백업을 실행할 수 있습니다. 이러한 서비스는 기본 백업 기능도 제공합니다. 시점 복구와 함께 자동 백업을 제공하는 AWS 서비스에는 Amazon DynamoDB, Amazon RDS 및 Amazon Keyspaces(Apache Cassandra용)가 포함되며, 백업 기록 내의 특정 시점으로 복원할 수 있습니다. 다른 AWS 데이터 스토리지 서비스는 대부분 최대 빈도 1시간 간격으로 주기적 백업 일정을 수립하는 기능을 제공합니다.

Amazon RDS 및 Amazon DynamoDB는 시점 복구를 통해 지속적 백업을 지원합니다. Amazon S3 버전 관리를 활성화한 경우 자동으로 수행됩니다. Amazon Data Lifecycle Manager를 사용하여 Amazon EBS 스냅샷의 생성, 복사 및 삭제를 자동화할 수 있습니다. 또한 Amazon EBS 지원 Amazon Machine Image(AMI) 및 기본 Amazon EBS 스냅샷의 생성, 복사, 사용 중단 및 등록 취소도 자동화할 수 있습니다.

AWS Elastic Disaster Recovery는 소스 환경(온프레미스 또는 AWS)에서 대상 복구 리전으로 지속적인 블록 수준 복제를 제공합니다. 특정 시점 Amazon EBS 스냅샷은 서비스에서 자동으로 생성 및 관리됩니다.

AWS Backup은 백업 자동화 및 기록을 중앙 집중식으로 볼 수 있는 완전관리형 정책 기반 백업 솔루션을 제공합니다. AWS Storage Gateway를 사용하여 클라우드뿐 아니라 온프레미스의 여러 AWS 서비스에 걸쳐 데이터 백업을 중앙 집중화하고 자동화합니다.

Amazon S3는 버전 관리에 더해 복제 기능도 제공합니다. 전체 S3 버킷을 같거나 다른 AWS 리전의 다른 버킷에 자동으로 복제할 수 있습니다.

구현 단계

1. 현재 수동으로 백업되고 있는 데이터 소스를 식별합니다. 자세한 내용은 REL09-BP01 백업해야 하는 모든 데이터 확인 및 백업 또는 소스에서 데이터 복제 섹션을 참조하세요.
2. 워크로드의 RPO를 결정합니다. 자세한 내용은 REL13-BP01 가동 중단 시간 및 데이터 손실 시의 복구 목표 정의 섹션을 참조하세요.

3. 자동화된 백업 솔루션 또는 관리형 서비스를 사용합니다. AWS Backup은 클라우드 및 온프레미스에서 AWS 서비스 전반에 걸쳐 데이터 보호를 쉽게 중앙 집중화하고 자동화할 수 있는 완전관리형 서비스입니다. AWS Backup에서 백업 계획을 사용하여 백업할 리소스와 이러한 백업을 생성해야 하는 빈도를 정의하는 규칙을 만듭니다. 이 빈도는 2단계에서 설정한 RPO를 기반으로 해야 합니다. AWS Backup을 사용하여 자동 백업을 생성하는 방법에 대한 실습 지침은 Testing Backup and Restore of Data를 참조하세요. 데이터를 저장하는 AWS 서비스에서는 대부분 기본적으로 백업 기능을 제공합니다. 예를 들어, RDS를 사용하여 시점 복구(PITR) 기능이 있는 자동화된 백업을 생성할 수 있습니다.
4. 온프레미스 데이터 소스 또는 메시지 대기열과 같이 자동화된 백업 솔루션 또는 관리형 서비스에서 지원되지 않는 데이터 소스의 경우 신뢰할 수 있는 서드파티 솔루션을 사용하여 자동화된 백업을 생성하는 것을 고려하세요. 아니면 AWS CLI 또는 SDK를 사용하여 백업 생성을 위한 자동화를 생성할 수 있습니다. AWS Lambda 함수 또는 AWS Step Functions를 사용하여 데이터 백업 생성에 포함된 로직을 정의하고, Amazon EventBridge를 사용하여 RPO를 기반으로 정해진 빈도에 간접 호출할 수 있습니다.

구현 계획의 작업 수준: 낮음

## 리소스

관련 문서:

- APN 파트너: 백업을 지원할 수 있는 파트너
- AWS Marketplace: 백업에 사용할 수 있는 제품
- Creating an EventBridge Rule That Triggers on a Schedule
- AWS Backup란 무엇입니까?
- AWS Step Functions란 무엇입니까?
- What is AWS Elastic Disaster Recovery?

관련 비디오:

- AWS re:Invent 2019: Deep dive on AWS Backup, ft. Rackspace (STG341)

# REL09-BP04 백업 무결성 및 프로세스를 확인하기 위해 데이터의 주기적인 복구 수행

복구 테스트를 수행하여 백업 프로세스 구현이 Recovery Time Objective(RTO) 및 Recovery Point Objective(RPO)를 충족하는지 검증합니다.

원하는 성과: 효과적으로 정의된 메커니즘을 사용하여 백업의 데이터가 주기적으로 복구되어 워크로드에 정해진 Recovery Time Objective(RTO) 내에 복구가 가능하도록 합니다. 원본 데이터가 손상되거나 액세스가 불가능하지 않고 Recovery Point Objective(RPO)에서 허용되는 데이터 손실만 이루어진 채로 백업이 원본 데이터가 포함된 리소스로 복원되는지 확인합니다.

일반적인 안티 패턴:

- 백업을 복원하지만 복원이 사용 가능한 상태인지 확인하기 위해 데이터를 쿼리하거나 검색하지 않습니다.
- 백업이 존재한다고 가정합니다.
- 시스템의 백업이 완전히 작동하며 시스템에서 데이터를 복구할 수 있다고 가정합니다.
- 복원 시점 또는 백업에서의 데이터 복구가 워크로드의 RTO 이내에 이루어진다고 가정합니다.
- 백업에 포함된 데이터가 워크로드의 RPO에 부합한다고 가정합니다.
- 런북을 사용하지 않거나 설정된 자동화 절차를 벗어나 필요에 따라 복원합니다.

이 모범 사례 확립의 이점: 백업의 복구를 테스트하면 데이터가 누락되거나 손상되는 것을 걱정하지 않고 필요할 때 데이터를 복원할 수 있으며, 복원 및 복구가 워크로드의 RTO 내에 이루어지도록 하며, 데이터 손실이 있는 경우 워크로드의 RPO에 부합하도록 할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 지침

백업 및 복원 기능을 테스트하면 중단 시 백업 및 복원 수행의 역량에 대한 신뢰도를 높일 수 있습니다. 주기적으로 백업을 새로운 위치에 복원하고 테스트를 실행하여 데이터의 무결성을 확인합니다. 수행해야 하는 몇 가지 일반적인 테스트는 모든 데이터가 사용 가능한지, 손상되지 않았는지, 액세스 가능한지, 모든 데이터 손실이 워크로드에 대한 RPO 내에 속하는지 확인하는 것입니다. 이러한 테스트는 복구 메커니즘의 속도가 워크로드의 RTO에 부합할 만큼 빠른지 확인하는 데도 도움이 됩니다.

AWS를 사용하면 테스트 환경을 구축하고 백업을 복원하여 RTO 및 RPO 기능을 평가하고 데이터 콘텐츠와 무결성에 대한 테스트를 실행할 수 있습니다.

또한 Amazon RDS와 Amazon DynamoDB는 시점 복구(PITR)를 지원합니다. 연속 백업을 사용하면 데이터세트를 지정된 날짜 및 시간의 상태로 복원할 수 있습니다.

모든 데이터를 사용 가능한지, 데이터가 손상되지 않았는지, 모든 데이터에 액세스할 수 있는지, 데이터 손실이 있는 경우 워크로드의 RPO에 부합하는지 확인하는 작업이 있습니다. 이러한 테스트는 복구 메커니즘의 속도가 워크로드의 RTO에 부합할 만큼 빠른지 확인하는 데도 도움이 됩니다.

AWS Elastic Disaster Recovery에서는 Amazon EBS 볼륨의 지속적인 시점 복구 스냅샷을 제공합니다. 소스 서버가 복제되면 구성된 정책을 기반으로 특정 시점 상태가 시간 경과에 따라 기록됩니다. Elastic Disaster Recovery는 트래픽을 리디렉션하지 않고 테스트 및 드릴 목적으로 인스턴스를 시작하여 이러한 스냅샷의 무결성을 확인하는 데 도움이 됩니다.

구현 단계

1. 현재 백업되고 있는 데이터 소스와 이러한 백업이 저장되는 위치를 식별합니다. 구현 지침은 REL09-BP01 백업해야 하는 모든 데이터 확인 및 백업 또는 소스에서 데이터 복제 섹션을 참조하세요.
2. 각 데이터 소스에 대해 데이터 검증 기준을 설정합니다. 다양한 유형의 데이터에는 다양한 유효성 검사 메커니즘이 필요할 수 있는 다양한 속성이 있습니다. 확신을 갖고 프로덕션에 사용하기 전에 이 데이터가 어떻게 검증될 수 있는지 고려하세요. 데이터를 검증하는 몇 가지 일반적인 방법은 데이터 유형, 형식, 체크섬, 크기 등의 데이터 및 백업 속성을 사용하거나 이러한 속성과 사용자 지정 검증 로직을 결합하는 것입니다. 예를 들어, 복원된 리소스와 백업이 생성된 당시의 데이터 소스의 체크섬 값을 비교해 볼 수 있습니다.
3. 데이터 중요도에 따라 데이터를 복원하기 위한 RTO 및 RPO를 설정합니다. 구현 지침은 REL13-BP01 가동 중단 시간 및 데이터 손실 시의 복구 목표 정의 섹션을 참조하세요.
4. 복구 역량을 평가합니다. 백업 및 복원 전략을 검토하여 RTO 및 RPO를 충족하는지 파악하고 필요에 따라 전략을 조정합니다. AWS Resilience Hub를 사용하여 워크로드 평가를 실행할 수 있습니다. 이 평가를 통해 애플리케이션 구성을 복원력 정책과 비교하고 RTO 및 RPO 목표가 충족될 수 있는지 보고합니다.
5. 프로덕션에서 데이터 복원을 위해 현재 확립된 프로세스를 사용하여 테스트 복원을 수행합니다. 이 프로세스는 원본 데이터 소스가 백업되는 방법, 백업 자체의 형식 및 스토리지 위치 또는 다른 소스에서의 데이터 복제 여부에 따라 달라집니다. 예를 들어, AWS Backup과 같은 관리형 서비스를 사용하는 경우 백업을 새 리소스에 간단하게 복원할 수 있습니다. AWS Elastic Disaster Recovery를 사용한 경우 복구 드릴을 시작할 수 있습니다.
6. 데이터 유효성 검사를 위해 이전에 설정한 기준에 따라 복원된 리소스에서 데이터 복구 유효성을 검사합니다. 복원되고 복구된 데이터에 백업 시 가장 최신 레코드 또는 항목이 포함되어 있나요? 이 데이터가 워크로드의 RPO에 부합하나요?

7. 복원 및 복구에 필요한 시간을 측정하고 이를 설정된 RTO와 비교합니다. 프로세스가 워크로드의 RTO에 부합하나요? 예를 들어, 복원 프로세스가 시작됐을 때의 타임스탬프와 복구 검증이 완료됐을 때의 타임스탬프를 비교하여 프로세스에 걸린 시간을 계산합니다. 모든 AWS API 직접 호출에는 타임스탬프가 지정되며 이 정보는 AWS CloudTrail에서 확인할 수 있습니다. 이 정보가 복원 프로세스가 시작된 시간에 대한 세부 정보를 제공하지만 검증이 완료된 종료 타임스탬프는 검증 로직에서 기록되어야 합니다. 자동화된 프로세스를 사용하는 경우 Amazon DynamoDB와 같은 서비스를 사용하여 이 정보를 저장할 수 있습니다. 또한 많은 AWS 서비스에서 특정 작업이 발생한 시점의 타임스탬프가 기록된 정보가 표시되는 이벤트 기록을 제공합니다. AWS Backup 내에서 백업 및 복원 작업을 작업이라고 하며, 이러한 작업에는 복원 및 복구에 필요한 시간을 측정하는 데 사용할 수 있는 타임스탬프 정보가 메타데이터의 일부로 포함됩니다.
8. 데이터 검증이 실패하거나 복원 및 복구에 필요한 시간이 워크로드에 확립된 RTO를 초과하는 경우 이해관계자에게 알립니다. 이 실습과 같이 자동화를 구현하는 경우 Amazon Simple Notification Service(SNS)와 같은 서비스를 사용하여 이해관계자에게 이메일 또는 SMS와 같은 푸시 알림을 보낼 수 있습니다. 이러한 메시지는 Amazon Chime, Slack 또는 Microsoft Teams와 같은 메시징 애플리케이션에 게시하거나 AWS Systems Manager OpsCenter를 사용하여 작업을 OpsItems로 생성하는 데 사용할 수도 있습니다.
9. 이 프로세스를 주기적으로 실행하도록 자동화합니다. 예를 들어, AWS Lambda 또는 AWS Step Functions의 State Machine과 같은 서비스를 사용하면 복원 및 복구 프로세스를 자동화할 수 있으며, Amazon EventBridge를 사용하여 아래 아키텍처 다이어그램에서 보이는 것처럼 이 자동 워크플로를 주기적으로 간접 호출할 수 있습니다. AWS Backup에서 데이터 복구 유효성 검사 자동화 방법에 대해 알아보세요. 또한 이 Well-Architected 실습은 여기에 있는 여러 단계에 대해 자동화를 수행하는 한 가지 방법에 대한 실습 경험을 제공합니다.

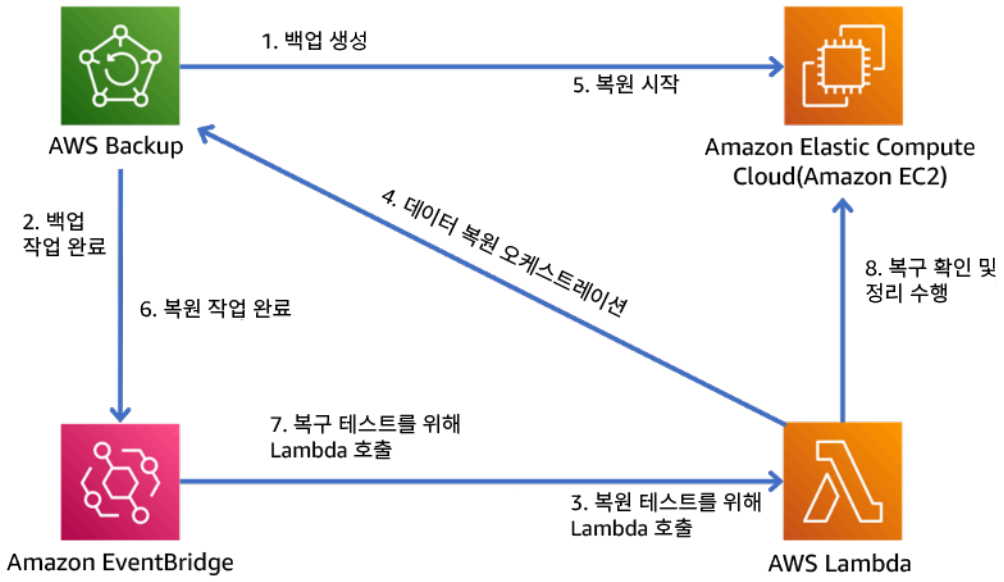


그림 9. 자동화된 백업 및 복원 프로세스

구현 계획의 작업 수준: 보통~높음(유효성 검사 기준의 복잡성에 따라 달라짐).

## 리소스

관련 문서:

- Automate data recovery validation with AWS Backup
- APN 파트너: 백업을 지원할 수 있는 파트너
- AWS Marketplace: 백업에 사용할 수 있는 제품
- Creating an EventBridge Rule That Triggers on a Schedule
- DynamoDB에 대한 온디맨드 백업 및 복원
- AWS Backup란 무엇입니까?
- AWS Step Functions란 무엇입니까?
- AWS Elastic Disaster Recovery이란 무엇입니까?
- AWS Elastic Disaster Recovery

# 장애 격리를 사용하여 워크로드 보호

장애 격리는 구성 요소 또는 시스템 장애의 영향을 정의된 경계로 제한합니다. 올바르게 격리하면 경계 외부의 구성 요소는 장애의 영향을 받지 않습니다. 여러 장애 격리 경계에서 워크로드를 실행하면 장애에 대한 복원력이 향상될 수 있습니다.

모범 사례

- REL10-BP01 워크로드를 여러 위치에 배포
- REL10-BP02 단일 위치로 제약된 구성 요소의 복구 자동화
- REL10-BP03 격벽 아키텍처를 사용하여 영향 범위 제한

## REL10-BP01 워크로드를 여러 위치에 배포

워크로드 데이터와 리소스를 여러 가용 영역에 분산하거나 필요한 경우 AWS 리전 전체에 분산합니다.

AWS에서 서비스 설계의 기본 원칙은 기본 물리적 인프라를 포함하여 단일 장애 지점을 방지하는 것입니다. AWS는 리전이라는 여러 지리적 위치에서 전 세계적으로 클라우드 컴퓨팅 리소스 및 서비스를 제공합니다. 각 리전은 물리적 및 논리적으로 독립적이며 3개 이상의 가용 영역(AZ)으로 구성됩니다.

가용 영역은 지리적으로 서로 가깝지만 물리적으로 분리되고 격리되어 있습니다. 가용 영역과 리전 간에 워크로드를 분산하면 화재, 홍수, 날씨 관련 재해, 지진, 인적 오류와 같은 위협의 위험을 완화할 수 있습니다.

워크로드에 적합한 고가용성을 제공하는 위치 전략을 수립하세요.

원하는 성과: 프로덕션 워크로드는 내결함성과 고가용성을 달성하기 위해 여러 가용 영역(AZ) 또는 리전에 분산됩니다.

일반적인 안티 패턴:

- 프로덕션 워크로드는 단일 가용 영역에만 존재합니다.
- 다중 AZ 아키텍처로 비즈니스 요구 사항을 충족할 수 있는 경우 다중 리전 아키텍처를 구현합니다.
- 배포 또는 데이터가 비동기화되어 구성이 드리프트되거나 복제가 저조해지지 않습니다.
- 복원력과 다중 위치 요구 사항이 구성 요소 간에 다를 경우 애플리케이션 구성 요소 간의 종속성을 고려하지 않습니다.

이 모범 사례 확립의 이점:

- 워크로드는 전력 또는 환경 제어 장애, 자연 재해, 업스트림 서비스 장애 또는 AZ나 전체 리전에 영향을 미치는 네트워크 문제와 같은 인시던트에 대해 복원력이 더 높습니다.
- 더 광범위한 Amazon EC2 인스턴스 인벤토리에 액세스하고 특정 EC2 인스턴스 유형을 시작할 때 InsufficientCapacityExceptions(ICE)의 가능성을 줄일 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

한 리전에서 최소 2개 이상의 가용 영역(AZ)에 모든 프로덕션 워크로드를 배포하고 운영합니다.

다중 가용 영역 사용

가용 영역은 리소스를 호스팅하는 위치로, 화재, 홍수 및 토네이도와 같은 위험으로 인한 상관관계가 있는 장애를 방지하기 위해 물리적으로 서로 분리되어 있습니다. 각 가용 영역에는 유틸리티 전원 연결, 백업 전원, 기계 서비스 및 네트워크 연결을 포함한 독립적인 물리적 인프라가 있습니다. 이로 인해 이러한 구성 요소에 장애가 발생하면 영향을 받는 가용 영역으로만 장애가 제한됩니다. 예를 들어 AZ 전체에 발생한 인시던트로 인해 영향을 받는 가용 영역에서 EC2 인스턴스를 사용할 수 없는 경우 다른 가용 영역의 인스턴스는 여전히 사용할 수 있습니다.

물리적으로 분리되어 있음에도 불구하고 동일한 AWS 리전의 가용 영역은 처리량이 높고 지연 시간이 짧은(10밀리초 미만) 네트워킹을 제공할 수 있을 만큼 충분히 가깝습니다. 사용자 경험에 큰 영향을 주지 않고 대부분의 워크로드에 대해 가용 영역 간에 데이터를 동기식으로 복제할 수 있습니다. 즉, 액티브/액티브 또는 액티브/대기 구성에서 가용 영역을 사용할 수 있습니다.

워크로드와 연결된 모든 컴퓨팅은 여러 가용 영역에 분산되어야 합니다. 여기에는 Amazon EC2 인스턴스, AWS Fargate 작업 및 VPC 연결 AWS Lambda 함수가 포함됩니다. EC2 Auto Scaling, Amazon Elastic Container Service(Amazon ECS) 및 Amazon Elastic Kubernetes Service(Amazon EKS)를 포함한 AWS 컴퓨팅 서비스는 가용 영역에서 컴퓨팅을 시작하고 관리할 수 있는 방법을 제공합니다. 가용성을 유지하기 위해 필요에 따라 다른 가용 영역에서 컴퓨팅을 자동으로 대체하도록 구성합니다. 트래픽을 사용 가능한 가용 영역으로 전달하려면 컴퓨팅 앞에 Application Load Balancer 또는 Network Load Balancer와 같은 로드 밸런서를 배치합니다. AWS 로드 밸런서는 가용 영역 장애가 발생할 경우 사용 가능한 인스턴스로 트래픽을 다시 라우팅할 수 있습니다.

또한 워크로드에 대한 데이터를 복제하여 여러 가용 영역에서 사용할 수 있도록 해야 합니다. Amazon S3, Amazon Elastic File Service(Amazon EFS), Amazon Aurora, Amazon DynamoDB, Amazon Simple Queue Service(Amazon SQS) 및 Amazon Kinesis Data Streams와 같은 일부 AWS 관리형 데이터 서비스는 기본적으로 여러 가용 영역에 데이터를 복제하며 가용 영역 장애에 대해 견고합니다. Amazon Relational Database Service(Amazon RDS), Amazon Redshift 및 Amazon ElastiCache 등 다른 AWS 관리형 데이터 서비스를 사용하려면 다중 AZ 복제를 활성화해야 합니다. 활성화하면 이러한 서비스는 자동으로 가용 영역 장애를 감지하고, 요청을 사용할 수 있는 가용 영역으로 리디렉션하고, 고객 개입 없이 복구 후 필요에 따라 데이터를 다시 복제합니다. 다중 AZ 기능, 동작 및 작업을 이해하기 위해 사용하는 각 AWS 관리형 데이터 서비스에 대한 사용 설명서를 숙지합니다.

Amazon Elastic Block Store(Amazon EBS) 볼륨 또는 Amazon EC2 인스턴스 스토리지와 같은 자체 관리형 스토리지를 사용하는 경우 다중 AZ 복제를 직접 관리해야 합니다.

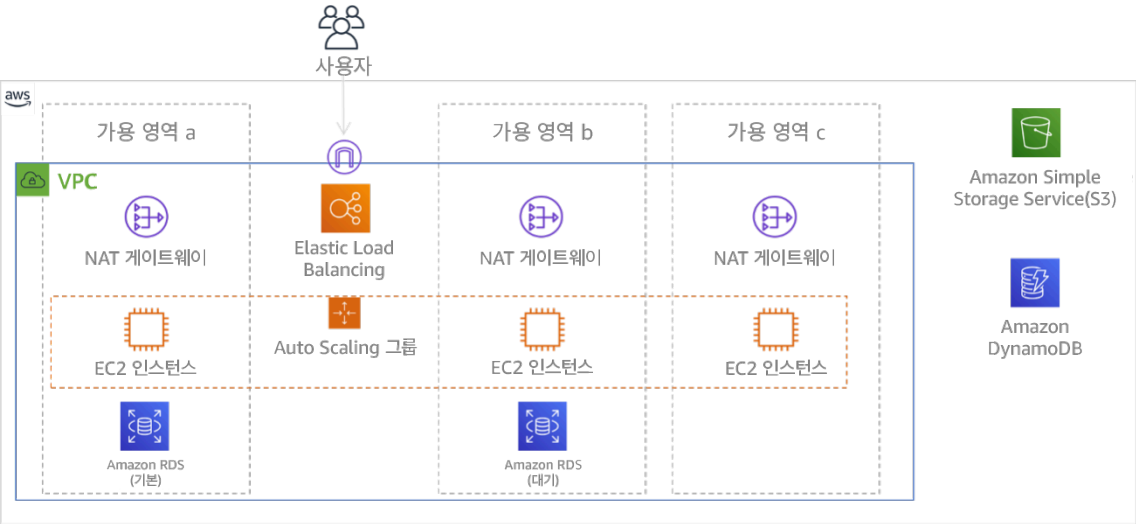


그림 9: 3개의 가용 영역에 배포된 다중 계층 아키텍처. Amazon S3 및 Amazon DynamoDB는 항상 자동으로 다중 AZ에 유지됩니다. 또한 ELB는 3개 영역 모두에 배포됩니다.

다중 AWS 리전 사용

극도의 복원력이 필요한 워크로드(예: 중요한 인프라, 건강 관련 애플리케이션 또는 엄격한 고객 요구 사항 또는 필수 가용성 요구 사항이 있는 서비스)가 있는 경우 단일 AWS 리전이 제공할 수 있는 것 이상의 추가 가용성이 필요할 수 있습니다. 이 경우 최소 2개의 AWS 리전에 워크로드를 배포하고 운영해야 합니다(데이터 레지던시 요구 사항에서 허용한다고 가정).

AWS 리전은 전 세계의 여러 지리적 지역과 여러 대륙에 위치해 있습니다. AWS 리전은 가용 영역만 있는 것보다 물리적 분리 및 격리의 정도가 훨씬 더 큽니다. AWS 서비스는 거의 예외 없이 이 설계를 활용하여 서로 다른 리전 간에 완전히 독립적으로 운영됩니다(리전 서비스라고도 함). 한 AWS 리전 서비스의 장애는 다른 리전의 서비스에 영향을 주지 않도록 설계되었습니다.

여러 리전에서 워크로드를 운영할 때는 추가 요구 사항을 고려해야 합니다. 서로 다른 리전의 리소스는 서로 별개이고 독립적이므로 각 리전에서 워크로드의 구성 요소를 복제해야 합니다. 여기에는 컴퓨팅 및 데이터 서비스 외에도 VPC와 같은 기본 인프라가 포함됩니다.

참고: 다중 리전 설계를 고려할 때는 워크로드가 단일 리전에서 실행될 수 있는지 확인합니다. 한 리전의 구성 요소가 다른 리전의 서비스 또는 구성 요소에 의존하는 리전 간 종속성을 생성하는 경우 장애 위험을 높이고 신뢰성 태세를 크게 약화시킬 수 있습니다.

다중 리전 배포를 용이하게 하고 일관성을 유지 관리하기 위해 AWS CloudFormation StackSets는 여러 리전에 전체 AWS 인프라를 복제할 수 있습니다. 또한 AWS CloudFormation은 구성 드리프트를 감지하고 리전의 AWS 리소스가 동기화되지 않을 때 알려줍니다. 많은 AWS 서비스가 중요한 워크로드 자산에 대해 다중 리전 복제를 제공합니다. 예를 들어 EC2 Image Builder는 사용하는 각 리전에 구축할 때마다 EC2 머신 이미지(AMI)를 게시할 수 있습니다. Amazon Elastic Container Registry(Amazon ECR)는 컨테이너 이미지를 선택한 리전에 복제할 수 있습니다.

또한 선택한 각 리전에 데이터를 복제해야 합니다. Amazon S3, Amazon DynamoDB, Amazon RDS, Amazon Aurora, Amazon Redshift, Amazon Elasticache 및 Amazon EFS를 비롯한 많은 AWS 관리형 데이터 서비스는 리전 간 복제 기능을 제공합니다. Amazon DynamoDB 글로벌 테이블은 지원되는 모든 리전의 쓰기를 수락하고 구성된 다른 모든 리전에 데이터를 복제합니다. 다른 리전에는 읽기 전용 복제본이 포함되어 있으므로 다른 서비스에서는 쓰기를 위한 기본 리전을 지정해야 합니다. 워크로드가 사용하는 각 AWS 관리형 데이터 서비스에 관해 사용 설명서 및 개발자 안내서를 참조하여 다중 리전 기능과 제한 사항을 이해하세요. 쓰기를 지시해야 하는 위치, 트랜잭션 기능 및 제한 사항, 복제 수행 방식, 리전 간 동기화를 모니터링하는 방법에 특히 주의를 기울이세요.

AWS는 요청 트래픽을 뛰어난 유연성으로 리전 배포로 라우팅할 수 있는 기능도 제공합니다. 예를 들어, Amazon Route 53를 사용하여 트래픽을 사용자에게 가장 가까운 가용 리전으로 전달하도록 DNS 레코드를 구성할 수 있습니다. 또는 한 리전을 기본 리전으로 지정하고 기본 리전이 비정상이 되는 경우에만 리전 복제본으로 돌아가는 액티브/대기 구성으로 DNS 레코드를 구성할 수 있습니다. 비정상 엔드포인트를 감지하고 자동 장애 조치를 수행하고 Amazon Application Recovery Controller(ARC)를 추가로 사용하여 필요에 따라 수동으로 다시 트래픽 라우팅을 할 수 있는 고가용성 라우팅 제어를 제공하도록 Route 53 상태 확인을 구성할 수 있습니다.

고가용성을 위해 여러 리전에서 운영하지 않기로 선택하더라도 재해 복구(DR) 전략의 일환으로 여러 리전을 고려해 보세요. 가능하면 워크로드의 인프라 구성 요소와 데이터를 보조 리전의 웜 대기 또는 파일럿 라이트 구성으로 복제합니다. 이 설계에서는 기본 리전에서 VPC, Auto Scaling 그룹, 컨테이너 오케스트레이터 및 기타 구성 요소와 같은 기준 인프라를 복제하지만 대기 리전의 가변 크기 구성 요소(예: EC2 인스턴스 및 데이터베이스 복제본 수)는 작동 가능한 최소 크기로 구성합니다. 또한 기본 리전에서 대기 리전으로의 지속적인 데이터 복제를 준비합니다. 인시던트가 발생하면 대기 리전의 리소스를 스케일 아웃, 즉 증가시킨 다음 기본 리전으로 승격할 수 있습니다.

구현 단계

1. 비즈니스 이해관계자 및 데이터 레지던시 전문가와 협력하여 리소스 및 데이터를 호스팅하는 데 사용할 수 있는 AWS 리전을 결정합니다.
2. 비즈니스 및 기술 이해관계자와 협력하여 워크로드를 평가하고 다중 AZ 접근 방식(단일 AWS 리전)으로 복원력 요구 사항을 충족할 수 있는지, 아니면 다중 리전 접근 방식(여러 리전이 허용되는 경우)이 필요한지 확인합니다. 여러 리전을 사용하면 가용성이 향상될 수 있지만 복잡성이 늘어나고 비용이 추가로 발생할 수 있습니다. 평가에서 다음 요소를 고려하세요.
   a. 비즈니스 목표 및 고객 요구 사항: 가용 영역 또는 리전에서 워크로드에 영향을 미치는 인시던트가 발생할 경우 가동 중지 시간이 얼마나 허용되나요? REL13-BP01 가동 중지 시간 및 데이터 손실에 대한 복구 목표 정의에 설명된 대로 복구 시점 목표를 평가합니다.
   b. 재해 복구(DR) 요구 사항: 어떤 종류의 잠재적 재해에 대비하고 싶은가요? 단일 가용 영역에서 전체 리전에 이르기까지 다양한 영향 범위에서 데이터 손실 또는 장기적인 사용 불가의 가능성을 고려합니다. 여러 가용 영역에 데이터와 리소스를 복제하고 단일 가용 영역에 지속적인 장애가 발생하는 경우 다른 가용 영역에서 서비스를 복구할 수 있습니다. 리전 간에 데이터 및 리소스를 복제하는 경우 다른 리전에서 서비스를 복구할 수 있습니다.
3. 컴퓨팅 리소스를 여러 가용 영역에 배포합니다.
   a. VPC에서 서로 다른 가용 영역에 여러 서브넷을 생성합니다. 인시던트 발생 중에도 워크로드를 처리하는 데 필요한 리소스를 수용할 수 있도록 각 서브넷을 충분히 크게 구성합니다. 자세한 내용은 REL02-BP03 확장 및 가용성을 위한 IP 서브넷 할당 계정 확인을 참조하세요.

b. Amazon EC2 인스턴스를 사용하는 경우 EC2 Auto Scaling을 사용하여 인스턴스를 관리합니다. Auto Scaling 그룹을 생성할 때 이전 단계에서 선택한 서브넷을 지정합니다.

c. Amazon ECS 또는 Amazon EKS용 AWS Fargate 컴퓨팅을 사용하는 경우 ECS 서비스를 생성하거나 ECS 작업을 시작하거나 EKS용 Fargate 프로필을 생성할 때 첫 번째 단계에서 선택한 서브넷을 선택합니다.

d. VPC에서 실행해야 하는 AWS Lambda 함수를 사용하는 경우 Lambda 함수를 생성할 때 첫 번째 단계에서 선택한 서브넷을 선택합니다. VPC 구성이 없는 함수의 경우 AWS Lambda는 자동으로 가용성을 관리합니다.

e. 로드 밸런서와 같은 트래픽 디렉터를 컴퓨팅 리소스 앞에 배치합니다. 교차 영역 로드 밸런싱이 활성화된 경우 AWS Application Load Balancer 및 Network Load Balancer는 가용 영역 장애로 인해 EC2 인스턴스 및 컨테이너와 같은 대상에 연결할 수 없을 때 이를 감지하고 정상 가용 영역의 대상으로 트래픽을 다시 라우팅합니다. 교차 영역 로드 밸런싱을 비활성화하는 경우 Amazon Application Recovery Controller(ARC)를 사용하여 영역 전환 기능을 제공합니다. 서드파티 로드 밸런서를 사용하거나 자체 로드 밸런서를 구현한 경우 여러 가용 영역에서 여러 프런트엔드로 구성합니다.

4. 여러 가용 영역에서 워크로드의 데이터를 복제합니다.

   a. Amazon RDS, Amazon ElastiCache 또는 Amazon FSx와 같은 AWS 관리형 데이터 서비스를 사용하는 경우 사용 설명서를 살펴보고 데이터 복제 및 복원력 기능을 이해합니다. 필요한 경우 교차 AZ 복제 및 장애 조치를 활성화합니다.

   b. Amazon S3, Amazon EFS 및 Amazon FSx와 같은 AWS 관리형 스토리지 서비스를 사용하는 경우 높은 내구성을 필요로 하는 데이터에 단일 AZ 또는 One Zone 구성을 사용하지 마세요. 이러한 서비스에는 다중 AZ 구성을 사용합니다. 각 서비스의 사용 설명서를 확인하여 다중 AZ 복제가 기본적으로 활성화되어 있는지, 아니면 직접 활성화해야 하는지 확인합니다.

   c. 자체 관리형 데이터베이스, 대기열 또는 기타 스토리지 서비스를 실행하는 경우 애플리케이션의 지침 또는 모범 사례에 따라 다중 AZ 복제를 준비합니다. 애플리케이션의 장애 조치 절차를 숙지합니다.

5. AZ 장애를 감지하고 트래픽을 정상 가용 영역으로 다시 라우팅하도록 DNS 서비스를 구성합니다. Amazon Route 53를 Elastic Load Balancer와 함께 사용하면 이 작업을 자동으로 수행할 수 있습니다. Route 53는 상태 확인을 사용하여 쿼리에 정상 IP 주소로만 응답하는 장애 조치 레코드로 구성할 수도 있습니다. 장애 조치에 사용되는 DNS 레코드의 경우 레코드 캐싱이 복구를 방해하지 않도록 짧은 Time to Live(TTL) 값(예: 60초 이하)을 지정합니다(Route 53 별칭 레코드가 적합한 TTL을 제공함).

여러 AWS 리전을 사용할 때의 추가 단계

1. 워크로드에서 사용하는 모든 운영 체제(OS) 및 애플리케이션 코드를 선택한 리전에 복제합니다. 필요한 경우 Amazon EC2 Image Builder와 같은 솔루션을 사용하여 Amazon EC2 Machine Images(AMI)를 복제합니다. Amazon ECR 리전 간 복제와 같은 솔루션을 사용하여 레지스트리에 저장된 컨테이너 이미지를 복제합니다. 애플리케이션 리소스를 저장하는 데 사용되는 모든 Amazon S3 버킷에 대해 리전 복제를 활성화합니다.
2. 컴퓨팅 리소스 및 구성 메타데이터(예: AWS Systems Manager Parameter Store에 저장된 파라미터)를 여러 리전에 배포합니다. 이전 단계에서 설명한 것과 동일한 절차를 사용하되, 워크로드에 사용 중인 각 리전의 구성을 복제합니다. AWS CloudFormation과 같은 코드형 인프라 솔루션을 사용하여 리전 간에 구성을 동일하게 복제합니다. 재해 복구를 위해 파일럿 라이트 구성에서 보조 리전을 사용하는 경우 컴퓨팅 리소스 수를 최솟값으로 줄여 비용을 절감할 수 있습니다. 이로 인해 복구 시간이 늘어날 수 있습니다.
3. 기본 리전의 데이터를 보조 리전으로 복제합니다.
   a. Amazon DynamoDB 글로벌 테이블은 지원되는 모든 리전에서 작성할 수 있는 데이터의 글로벌 복제본을 제공합니다. Amazon RDS, Amazon Aurora, Amazon Elasticache와 같은 다른 AWS 관리형 데이터 서비스를 사용하면 기본(읽기/쓰기) 리전과 복제본(읽기 전용) 리전을 지정합니다. 리전 복제에 대한 자세한 내용은 각 서비스의 사용자 및 개발자 안내서를 참조하세요.
   b. 자체 관리형 데이터베이스를 실행하는 경우 애플리케이션의 지침 또는 모범 사례에 따라 다중 리전 복제를 준비합니다. 애플리케이션의 장애 조치 절차를 숙지합니다.
   c. 워크로드가 AWS EventBridge를 사용하는 경우 선택한 이벤트를 기본 리전에서 보조 리전으로 전달해야 할 수 있습니다. 이렇게 하려면 보조 리전의 이벤트 버스를 기본 리전의 일치하는 이벤트의 대상으로 지정합니다.
4. 리전 간에 동일한 암호화 키를 사용할지, 어느 정도의 범위로 사용할지를 고려합니다. 보안과 사용 편의성의 균형을 맞추는 일반적인 접근 방식은 리전-로컬 데이터 및 인증에 리전 범위 키를 사용하고, 다른 리전 간에 복제되는 데이터의 암호화에는 전역 범위 키를 사용하는 것입니다. AWS Key Management Service(KMS)는 다중 리전 키를 지원하여 리전 간에 공유되는 키를 안전하게 배포하고 보호합니다.
5. 트래픽을 정상 엔드포인트가 포함된 리전으로 전달하여 애플리케이션의 가용성을 개선하려면 AWS Global Accelerator를 고려하세요.

## 리소스

관련 모범 사례:

- REL02-BP03 확장 및 가용성을 위한 IP 서브넷 할당 계정 확인
- REL11-BP05 정적 안정성을 사용하여 바이모달 동작 방지

- REL13-BP01 가동 중지 시간 및 데이터 손실에 대한 복구 목표 정의

관련 문서:

- AWS 글로벌 인프라
- 백서: AWS Fault Isolation Boundaries
- Resilience in Amazon EC2 Auto Scaling
- Amazon EC2 Auto Scaling: Example: Distribute instances across Availability Zones
- How EC2 Image Builder works
- How Amazon ECS places tasks on container instances (includes Fargate)
- AWS Lambda의 복원성
- Amazon S3: Replicating objects overview
- Private image replication in Amazon ECR
- 글로벌 테이블: DynamoDB를 사용한 다중 리전 복제
- Amazon Elasticache for Redis OSS: Replication across AWS 리전 using global datastores
- Resilience in Amazon RDS
- Amazon Aurora Global Database 사용
- AWS Global Accelerator 개발자 안내서
- Multi-Region keys in AWS KMS
- Amazon Route 53: Configuring DNS failover
- Amazon Application Recovery Controller (ARC) Developer Guide
- Sending and receiving Amazon EventBridge events between AWS 리전
- Creating a Multi-Region Application with AWS Services blog series
- Disaster Recovery (DR) Architecture on AWS, Part I: Strategies for Recovery in the Cloud
- Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby

관련 비디오:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications
- AWS re:Invent 2019: Innovation and operation of the AWS global network infrastructure

# REL10-BP02 단일 위치로 제약된 구성 요소의 복구 자동화

워크로드의 구성 요소를 단일 가용 영역 또는 온프레미스 데이터 센터에서만 실행해야 하는 경우 정의된 복구 목표 내에서 워크로드를 완전히 재구축할 수 있는 기능을 구현합니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

기술적 제약으로 인해 워크로드를 여러 위치에 배포하는 모범 사례를 따를 수 없다면 복원력을 달성할 수 있는 대체 경로를 구현해야 합니다. 이러한 경우를 위해 필요한 인프라를 다시 생성하고, 애플리케이션을 다시 배포하며, 필요한 데이터를 다시 생성하는 기능을 자동화해야 합니다.

예를 들어 Amazon EMR은 지정된 클러스터의 모든 노드를 동일한 가용 영역에서 시작합니다. 동일한 영역에서 클러스터를 실행하면 데이터 접근 속도가 빨라져 작업 흐름의 성능이 개선되기 때문입니다. 워크로드 복원력에 이 구성 요소가 필요한 경우 클러스터와 해당 데이터를 다시 배포할 수 있어야 합니다. 또한 Amazon EMR의 경우 다중 AZ를 사용하는 것 이외의 방법으로 중복성을 프로비저닝해야 합니다. 여러 노드를 프로비저닝할 수 있습니다. EMR 파일 시스템(EMRFS)을 사용하면 EMR의 데이터를 Amazon S3에 저장할 수 있으며, 이는 다시 여러 가용 영역 또는 AWS 리전에 걸쳐 복제할 수 있습니다.

마찬가지로 Amazon Redshift의 경우 기본적으로 사용자가 선택한 AWS 리전 내에서 임의로 선택된 가용 영역에 클러스터를 프로비저닝합니다. 클러스터 노드는 모두 동일한 영역에 프로비저닝됩니다.

온프레미스 데이터 센터에 배포된 상태 저장 서버 기반 워크로드의 경우 AWS Elastic Disaster Recovery를 사용하여 AWS에서 워크로드를 보호할 수 있습니다. AWS에 이미 호스팅된 경우 Elastic Disaster Recovery를 사용하여 워크로드를 대체 가용 영역 또는 리전으로 보호할 수 있습니다. Elastic Disaster Recovery는 가벼운 스테이징 영역으로의 지속적 블록 수준 복제를 사용하여 온프레미스 및 클라우드 기반 애플리케이션에 대한 빠르고 신뢰할 수 있는 복구를 지원합니다.

구현 단계

1. 자가 복구를 구현합니다. 가능한 경우 자동 크기 조정을 사용하여 인스턴스 또는 컨테이너를 배포합니다. 자동 크기 조정을 사용할 수 없는 경우 EC2 인스턴스에 대한 자동 복구를 사용하거나 Amazon EC2 또는 ECS 컨테이너 수명 주기 이벤트를 기반으로 자가 복구 자동화를 구현합니다.
   - 단일 인스턴스 IP 주소, 프라이빗 IP 주소, 탄력적 IP 주소 및 인스턴스 메타데이터가 필요하지 않은 인스턴스 및 컨테이너 워크로드에 Amazon EC2 Auto Scaling 그룹을 사용합니다.
     - 시작 템플릿 사용자 데이터는 대부분의 워크로드를 자가 복구할 수 있는 자동화를 구현하는 데 사용할 수 있습니다.

- 단일 인스턴스 ID 주소, 프라이빗 IP 주소, 탄력적 IP 주소 및 인스턴스 메타데이터가 필요한 워크로드에 Amazon EC2 인스턴스 자동 복구를 사용합니다.
  - 자동 복구는 인스턴스 장애가 감지될 때 SNS 주제로 복구 상태 알림을 전송합니다.
- 자동 규모 조정 또는 EC2 복구를 사용할 수 없는 경우 Amazon EC2 인스턴스 수명 주기 이벤트 또는 Amazon ECS 이벤트를 사용하여 자가 복구를 자동화합니다.
  - 이벤트를 사용하여 필요한 프로세스 로직에 따라 구성 요소를 복구하는 자동화를 호출합니다.
- AWS Elastic Disaster Recovery을 사용하여 단일 위치로 제한된 상태 저장 워크로드를 보호합니다.

## 리소스

관련 문서:

- Amazon ECS 이벤트
- Amazon EC2 Auto Scaling 수명 주기 후크
- 인스턴스를 복구합니다.
- 서비스 자동 규모 조정
- What Is Amazon EC2 Auto Scaling?
- AWS Elastic Disaster Recovery

# REL10-BP03 격벽 아키텍처를 사용하여 영향 범위 제한

격벽 아키텍처(셀 기반 아키텍처라고도 함)를 구현하여 워크로드 내 장애의 영향을 제한된 수의 구성 요소로 제한합니다.

원하는 성과: 셀 기반 아키텍처는 워크로드의 여러 격리된 인스턴스를 사용하며 여기에서 각 인스턴스를 셀이라고 합니다. 각 셀은 독립적이고 다른 셀과 상태를 공유하지 않으며 전체 워크로드 요청의 하위 세트를 처리합니다. 이렇게 하면 잘못된 소프트웨어 업데이트와 같은 오류의 잠재적 영향이 개별 셀과 처리 중인 요청으로 축소됩니다. 워크로드가 10개의 셀을 사용하여 100개의 요청을 처리하는 경우 장애가 발생하면 전체 요청의 90%는 장애의 영향을 받지 않습니다.

일반적인 안티 패턴:

- 셀이 경계 없이 성장할 수 있도록 합니다.
- 동시에 모든 셀에 코드 업데이트 또는 배포를 적용합니다.

- 라우터 계층을 제외하고 셀 간에 상태 또는 구성 요소를 공유합니다.
- 복잡한 비즈니스 또는 라우팅 로직을 라우터 계층에 추가합니다.
- 셀 간 상호 작용을 최소화하지 않습니다.

이 모범 사례 확립의 이점: 셀 기반 아키텍처를 사용하면 많은 일반적인 유형의 오류가 셀 자체 내에 억제되어 추가적인 장애 격리를 제공합니다. 이러한 결함 경계는 실패한 코드 배포 또는 손상되거나 포이즌 필 요청이라고도 하는 특정 실패 모드를 간접 호출하는 요청과 같이 억제하기 어려운 실패 유형에 대한 복원력을 제공할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

선박의 격벽은 선체 파손이 선체의 한 섹션 내에 억제되도록 합니다. 복잡한 시스템에서 이 패턴은 장애 격리를 활성화하기 위해 종종 복제됩니다. 장애 격리 경계는 워크로드 내부 장애의 영향을 제한된 수의 구성 요소로 제한합니다. 경계 외부의 구성 요소는 장애의 영향을 받지 않습니다. 장애 격리 경계를 여러 개 사용하면 워크로드에 미치는 영향을 제한할 수 있습니다. AWS에서 고객은 여러 가용 영역 및 리전을 사용하여 장애 격리를 제공할 수 있지만 장애 격리의 개념은 워크로드의 아키텍처로도 확장될 수 있습니다.

전체 워크로드는 파티션 키로 분할된 셀입니다. 이 키는 서비스의 세부 수준 또는 최소한의 크로스 셀 상호 작용으로 서비스의 워크로드를 세분화할 수 있는 자연스러운 방식에 맞춰 조정되어야 합니다. 파티션 키로는 고객 ID, 리소스 ID 또는 대부분의 API 직접 호출에서 쉽게 액세스할 수 있는 기타 파라미터를 예로 들 수 있습니다. 셀 라우팅 계층은 파티션 키를 기반으로 개별 셀에 요청을 배포하고 클라이언트에 단일 엔드포인트를 제공합니다.

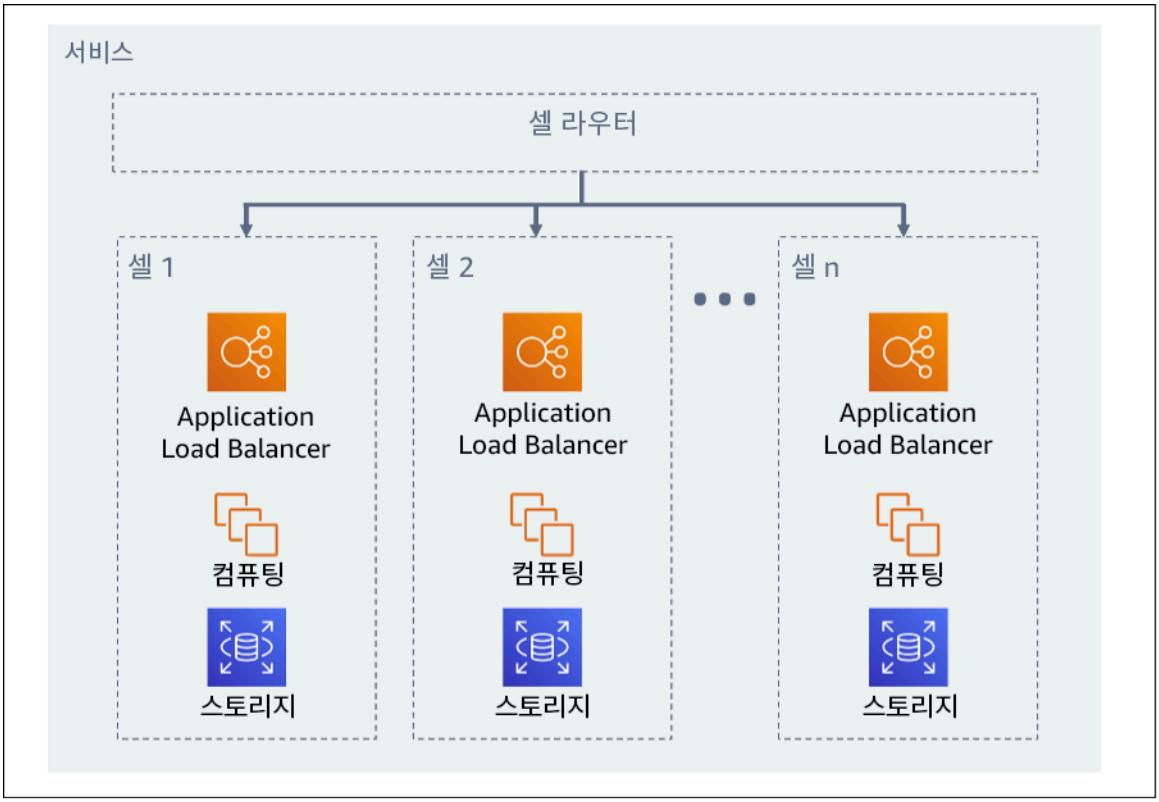



그림 11:; 셀 기반 아키텍처

구현 단계

셀 기반 아키텍처를 설계할 때 고려해야 할 몇 가지 설계 고려 사항이 있습니다.

1. 파티션 키: 파티션 키를 선택할 때는 특별한 고려 사항에 주의해야 합니다.
   - 이 키는 서비스의 세부 수준 또는 최소한의 크로스 셀 상호 작용으로 서비스의 워크로드를 세분화할 수 있는 자연스러운 방식에 맞춰 조정되어야 합니다. `customer ID` 또는 `resource ID`를 예로 들 수 있습니다.
   - 파티션 키는 모든 요청에서 직접 또는 다른 파라미터에 의해 결정론적으로 쉽게 추론될 수 있는 방식으로 사용 가능해야 합니다.
2. 영구 셀 매핑: 업스트림 서비스는 리소스의 수명 주기 동안 단일 셀과만 상호 작용해야 합니다.
   - 워크로드에 따라 한 셀에서 다른 셀로 데이터를 마이그레이션하는 데 셀 마이그레이션 전략이 필요할 수 있습니다. 셀 마이그레이션이 필요할 수 있는 가능한 시나리오는 워크로드의 특정 사용자 또는 리소스가 너무 커져 전용 셀이 필요한 경우입니다.
   - 셀은 셀 간에 상태 또는 구성 요소를 공유해서는 안 됩니다.
   - 결과적으로 셀 간 상호 작용은 피하거나 최소로 유지해야 합니다. 이러한 상호 작용은 셀 간의 종속성을 생성하여 장애 격리 개선을 감소시키기 때문입니다.

3. 라우터 계층: 라우터 계층은 셀 간에 공유되는 구성 요소이므로 셀과 동일한 구획 전략을 따를 수 없습니다.
   - 라우터 계층은 파티션 키를 셀에 매핑하기 위해 암호화 해시 함수와 모듈식 산술을 결합하는 것과 같이 컴퓨팅적으로 효율적인 방식으로 파티션 매핑 알고리즘을 사용하여 요청을 개별 셀에 배포하는 것이 좋습니다.
   - 다중 셀 영향을 방지하려면 라우팅 계층을 가능한 한 단순하고 수평적으로 확장할 수 있어야 하므로 이 계층 내에서 복잡한 비즈니스 로직을 피해야 합니다. 그러면 예상되는 동작을 항상 쉽게 이해할 수 있게 하여 철저한 테스트 가능성을 허용하는 추가적인 이점이 있습니다. Colm MacCárthaigh의 Reliability, constant work, and a good cup of coffee에서 설명한 대로, 단순한 설계와 일정한 작업 패턴은 신뢰할 수 있는 시스템을 만들고 취약성을 줄여줍니다.
4. 셀 크기: 셀은 최대 크기를 가져야 하며 이 수치를 초과하여 성장하지 않도록 해야 합니다.
   - 중단점에 도달하고 안전한 작동 마진이 설정될 때까지 철저한 테스트를 수행하여 최대 크기를 식별해야 합니다. 테스트 사례를 구현하는 방법에 대한 자세한 내용은 REL07-BP04 워크로드 로드 테스트 섹션을 참조하세요.
   - 전체 워크로드는 셀 추가를 통해 증가하므로, 수요 증가에 따라 워크로드를 확장할 수 있습니다.
5. 다중 AZ 또는 다중 리전 전략: 다양한 장애 도메인으로부터 보호하려면 여러 계층의 복원력을 활용해야 합니다.
   - 복원력을 위해서는 방어 계층을 구축하는 접근 방식을 사용해야 합니다. 하나의 계층은 다중 AZ를 사용하여 가용성이 높은 아키텍처를 구축함으로써 더 작고 더 일반적인 장애를 예방합니다. 또 다른 방어 계층은 만연한 자연 재해와 리전 수준의 장애와 같은 드문 이벤트를 예방하도록 설계합니다. 이 두 번째 계층에는 애플리케이션이 여러 AWS 리전에 분산되도록 하는 아키텍팅이 포함됩니다. 워크로드에 다중 리전 전략을 구현하면 한 나라의 광범위한 리전에 영향을 미치는 만연한 자연 재해나 리전 전체에 발생한 기술적 장애로부터 워크로드를 보호할 수 있습니다. 다중 리전 아키텍처를 구현하는 일은 매우 복잡할 수 있으며 대개 대부분의 워크로드에는 필요하지 않다는 점을 참고하시기 바랍니다. 자세한 내용은 REL10-BP01 워크로드를 여러 위치에 배포 섹션을 참조하세요.
6. 코드 배포: 코드 변경 사항을 모든 셀에 동시에 배포하는 것보다 시차를 두고 코드를 배포하는 전략을 사용하는 것이 좋습니다.
   - 이렇게 하면 잘못된 배포 또는 사람의 실수로 인해 여러 셀에 미치는 잠재적인 장애를 최소화하는 데 도움이 됩니다. 자세한 내용은 안전하고 간편한 배포 자동화를 참조하세요.

## 리소스

관련 모범 사례:

- REL07-BP04 워크로드 로드 테스트
- REL10-BP01 워크로드를 여러 위치에 배포

관련 문서:

- Reliability, constant work, and a good cup of coffee
- AWS and Compartmentalization
- 셔플 샤딩을 사용한 워크로드 격리
- 안전하고 간편한 배포 자동화

관련 비디오:

- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small
- AWS re:Invent 2018: How AWS Minimizes the Blast Radius of Failures (ARC338)
- Shuffle-sharding: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)
- AWS Summit ANZ 2021 - Everything fails, all the time: Designing for resilience

## 구성 요소 장애를 견디도록 워크로드 설계

고가용성 및 낮은 MTTR(평균 복구 시간)이 요구되는 워크로드는 복원력을 고려하여 설계해야 합니다.

모범 사례

- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지
- REL11-BP02 정상 리소스로 장애 조치
- REL11-BP03 모든 계층에서 복구 자동화
- REL11-BP04 복구 중 컨트롤 플레인이 아닌 데이터 플레인 사용
- REL11-BP05 정적 안정성을 사용하여 바이모달 동작 방지
- REL11-BP06 이벤트가 가용성에 영향을 미치는 경우 알림 전송
- REL11-BP07 가용성 목표 및 가동 시간 서비스 수준에 관한 계약(SLA)을 충족하도록 제품 설계

# REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지

워크로드 상태를 지속적으로 모니터링하여 장애 또는 성능 저하가 발생하는 즉시 사용자 및 자동화된 시스템이 이를 인식할 수 있도록 합니다. 비즈니스 가치를 기반으로 핵심 성과 지표(KPI)를 모니터링합니다.

모든 복구 메커니즘은 문제를 신속하게 탐지하는 기능에서 시작되어야 합니다. 기술적 장애를 먼저 감지하여 해결합니다. 그러나 가용성은 비즈니스 가치를 제공하는 워크로드의 기능에 따라 결정되므로 이 요구 사항을 측정하는 핵심 성과 지표(KPI)를 탐지 및 수정 전략의 핵심 척도로 사용해야 합니다.

원하는 성과: 워크로드의 필수 구성 요소를 독립적으로 모니터링하여 언제 어디서 장애가 발생하는지 감지하고 이에 대해 경고합니다.

일반적인 안티 패턴:

- 경보가 구성되지 않았기 때문에 알림 없이 중단이 발생합니다.
- 경보가 존재하지만 대응 시간이 충분하지 않은 임계치에 있습니다.
- 지표는 Recovery Time Objective(RTO)를 충족하기에 충분한 지표가 수집되지 않는 경우가 많습니다.
- 워크로드의 고객 대상 인터페이스만 능동적으로 모니터링됩니다.
- 기술 지표만 수집하며 비즈니스 기능 지표는 수집하지 않습니다.
- 워크로드의 사용자 경험을 측정하는 지표가 없습니다.
- 너무 많은 모니터가 생성되었습니다.

이 모범 사례 확립의 이점: 모든 계층에서 적절한 모니터링을 사용하면 감지 시간을 단축하여 복구 시간을 줄일 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

모니터링을 위해 검토할 모든 워크로드를 식별합니다. 모니터링해야 할 워크로드의 모든 구성 요소를 식별했으면 이제 모니터링 간격을 결정해야 합니다. 모니터링 간격은 장애 감지에 걸리는 시간을 기준으로 복구를 시작할 수 있는 속도에 직접적인 영향을 미칩니다. 평균 감지 시간(MTTD)은 장애가 발생한 시점부터 수리 작업이 시작되는 시점까지의 시간입니다. 서비스 목록은 광범위하고 완전해야 합니다.

모니터링은 애플리케이션, 플랫폼, 인프라 및 네트워크를 포함한 애플리케이션 스택의 모든 계층을 포괄해야 합니다.

모니터링 전략에서는 회색 장애의 영향을 고려해야 합니다. 회색 장애에 대한 자세한 내용은 Advanced Multi-AZ Resilience Patterns 백서의 Gray failures를 참조하세요.

구현 단계

- 모니터링 간격은 필요한 복구 속도에 따라 달라집니다. 복구 시간은 복구에 걸리는 시간에 따라 결정되므로 이 시간과 Recovery Time Objective(RTO)를 고려하여 수집 빈도를 결정해야 합니다.
- 구성 요소 및 관리형 서비스에 대한 세부 모니터링을 구성합니다.
  - Auto Scaling 및 EC2 인스턴스에 대한 세부 모니터링이 필요한지 결정합니다. 세부 모니터링은 1분 간격 지표를 제공하고, 기본 모니터링은 5분 간격 지표를 제공합니다.
  - RDS에 대한 향상된 모니터링이 필요한지 결정합니다. 향상된 모니터링은 RDS 인스턴스의 에이전트를 사용하여 여러 프로세스 또는 스레드에 대한 유용한 정보를 얻습니다.
  - Lambda, API Gateway, Amazon EKS, Amazon ECS, 모든 유형의 로드 밸런서에 대한 주요 서버리스 구성 요소의 모니터링 요구 사항을 결정합니다.
  - Amazon S3, Amazon FSx, Amazon EFS, Amazon EBS에 대한 스토리지 구성 요소의 모니터링 요구 사항을 결정합니다.
- 비즈니스 핵심 성과 지표(KPI)를 측정하는 사용자 지정 지표를 생성합니다. 워크로드는 주요 비즈니스 기능을 구현하며, 이는 간접적인 문제 발생 시점을 파악하는 데 도움이 되는 KPI로 사용되어야 합니다.
- 사용자 Canary를 사용하여 사용자 경험에서 장애가 발생했는지 모니터링합니다. 가장 중요한 테스트 프로세스 중 하나는 고객 행동을 실행하고 시뮬레이션할 수 있는 가상 트랜잭션 테스트(canary 테스트라고도 하지만 카나리 배포와는 다름)입니다. 다양한 원격 위치에서 워크로드 엔드포인트에 대해 이러한 테스트를 지속적으로 실행합니다.
- 사용자 경험을 추적하는 사용자 지정 지표를 생성합니다. 고객의 경험을 계측할 수 있으면 소비자 경험이 저하되는 시기를 결정할 수 있습니다.
- 워크로드의 일부가 제대로 작동하지 않는 시기를 감지하고 리소스 규모를 자동 조정해야 하는 시점을 알려주도록 경보를 설정합니다. 경보를 사용하면 대시보드에 경보를 시각적으로 표시하고, Amazon SNS 또는 이메일을 통해 알림을 전송하며, Auto Scaling을 통해 워크로드의 리소스를 스케일 업 또는 스케일 다운할 수 있습니다.
- 지표를 시각화하는 대시보드를 생성합니다. 대시보드를 사용하면 추세, 이상값 및 기타 잠재적 문제의 지표를 시각적으로 표시하거나, 조사가 필요할 수 있는 문제를 표시할 수 있습니다.
- 서비스에 대한 분산 추적 모니터링을 생성합니다. 분산 모니터링을 사용하면 애플리케이션과 해당하는 기본 서비스의 성능을 파악하여 성능 문제 및 오류의 근본 원인을 식별하고 해결할 수 있습니다.

- 별도의 리전 및 계정에서 모니터링 시스템(CloudWatch 또는 X-Ray 사용) 대시보드 및 데이터 수집을 생성합니다.
- AWS Health를 사용하여 서비스 성능 저하에 대한 최신 정보를 확인하세요. AWS 사용자 알림를 통해 이메일 및 채팅 채널에 적합한 AWS Health 이벤트 알림을 생성하고, Amazon EventBridge를 통해 모니터링 및 알림 도구와 프로그래밍 방식으로 통합할 수 있습니다.

## 리소스

관련 모범 사례:

- 가용성 정의
- REL11-BP06 이벤트가 가용성에 영향을 미치는 경우 알림 전송

관련 문서:

- Amazon CloudWatch Synthetics로 사용자 Canary 생성 지원
- 인스턴스에 대한 세부 모니터링 활성화 또는 비활성화
- 확장 모니터링
- Monitoring Your Auto Scaling Groups and Instances Using Amazon CloudWatch
- 사용자 지정 지표 게시
- Amazon CloudWatch 경보 사용
- CloudWatch 대시보드 사용
- Account CloudWatch의 크로스 리전 및 크로스 계정 대시보드 사용
- X-Ray의 크로스 리전 및 크로스 계정 추적 사용
- Understanding availability

관련 비디오:

- Mitigating gray failures

관련 예제:

- One Observability 워크숍: Explore X-Ray

관련 도구:

- CloudWatch
- CloudWatch X-Ray

## REL11-BP02 정상 리소스로 장애 조치

리소스 장애가 발생하는 경우 정상 리소스가 계속해서 요청을 처리해야 합니다. 위치 장애(예: 가용 영역 또는 AWS 리전)의 경우, 손상되지 않은 위치의 정상 리소스로 장애 조치할 수 있는 시스템을 갖추고 있어야 합니다.

서비스를 설계할 때는 리소스, 가용 영역 또는 리전에 부하를 분산하세요. 따라서 트래픽을 정상적인 리소스로 전환하여 개별 리소스의 장애 또는 중단을 완화할 수 있습니다. 장애 발생 시 서비스를 검색하고 라우팅하는 방법을 고려해 보세요.

장애 복구를 염두에 두고 서비스를 설계하세요. AWS에서는 장애에서 복구하는 시간과 데이터에 대한 영향을 최소화하는 방식으로 서비스가 설계됩니다. 자사 서비스는 기본적으로 한 리전 내의 여러 복제본에 저장된 요청만 승인하는 데이터 스토어를 사용합니다. 이들은 셀 기반 격리와 가용 영역에서 제공되는 장애 격리 기능을 사용하도록 구성됩니다. 자사 운영 절차에서는 자동화가 광범위하게 사용됩니다. 또한 서비스 중단 시 빠르게 복구할 수 있도록 교체 및 다시 시작 기능도 최적화됩니다.

장애 조치를 허용하는 패턴과 디자인은 AWS 플랫폼 서비스마다 다릅니다. 대부분의 AWS 기본 관리형 서비스는 기본적으로 다중 가용 영역(예: Lambda 또는 API Gateway)입니다. 다른 AWS 서비스(예: EC2 및 EKS)에서는 AZ에서 리소스 또는 데이터 스토리지의 장애 조치를 지원하기 위한 구체적인 모범 사례 설계가 필요합니다.

장애 조치 리소스가 정상인지 확인하고, 장애 조치 중인 리소스의 진행 상황을 추적하며, 비즈니스 프로세스 복구를 모니터링하도록 설정해야 합니다.

원하는 성과: 시스템은 새 리소스를 자동 또는 수동으로 사용하여 성능 저하를 복구할 수 있습니다.

일반적인 안티 패턴:

- 장애에 대한 계획은 계획 및 설계 단계의 일부가 아닙니다.
- RTO와 RPO가 설정되지 않았습니다.
- 모니터링이 충분하지 않아 장애가 발생한 리소스를 감지할 수 없습니다.
- 장애 도메인의 적절한 격리가 되지 않습니다.
- 다중 리전 장애 조치는 고려되지 않습니다.
- 장애 탐지는 장애 조치를 결정할 때 너무 민감하거나 공격적입니다.

- 장애 조치 설계를 테스트하거나 검증하지 않습니다.
- 자동 복구 자동화를 수행하지만 복구가 필요한 것을 알리지 않습니다.
- 너무 빨리 페일백하지 않도록 하는 완충 기간이 부족합니다.

이 모범 사례 확립의 이점: 점진적으로 성능이 저하되고 빠르게 복구되므로 장애 발생 시 신뢰성을 유지하는 복원력이 뛰어난 시스템을 구축할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

AWS 서비스(예: Elastic Load Balancing 및 Amazon EC2 Auto Scaling)는 리소스 및 가용 영역 전체에 부하를 분산하는 데 도움이 됩니다. 따라서 남아 있는 상태가 양호한 리소스로 트래픽을 이동시켜 개별 리소스(예: EC2 인스턴스)의 장애 또는 가용 영역의 장애를 완화할 수 있습니다.

다중 리전 워크로드의 경우 설계가 더 복잡합니다. 예를 들어, 리전 간 읽기 전용 복제본을 사용하면 데이터를 여러 AWS 리전에 배포할 수 있습니다. 하지만 읽기 전용 복제본을 기본 복제본으로 승격한 다음 트래픽이 새 엔드포인트로 향하도록 하려면 여전히 장애 조치가 필요합니다. Amazon Route 53, Amazon Application Recovery Controller(ARC), Amazon CloudFront 및 AWS Global Accelerator는 AWS 리전에서 트래픽을 라우팅하는 데 도움이 될 수 있습니다.

AWS 서비스(예: Amazon S3, Lambda, API Gateway, Amazon SQS, Amazon SNS, Amazon SES, Amazon Pinpoint, Amazon ECR, AWS Certificate Manager, EventBridge 또는 Amazon DynamoDB)는 AWS에 의해 다중 가용 영역에 자동으로 배포됩니다. 장애가 발생할 경우 이러한 AWS 서비스는 자동으로 트래픽을 정상 위치로 라우팅합니다. 데이터가 여러 가용 영역에 중복으로 저장되고 가용성이 유지됩니다.

Amazon RDS, Amazon Aurora, Amazon Redshift, Amazon EKS 또는 Amazon ECS의 경우 다중 AZ는 구성 옵션으로 제공됩니다. AWS에서는 장애 조치가 시작된 경우 정상 인스턴스로 트래픽을 전달할 수 있습니다. 이 장애 조치는 AWS에 의해 또는 고객의 요구에 따라 수행될 수 있습니다.

Amazon EC2 인스턴스, Amazon Redshift, Amazon ECS 작업 또는 Amazon EKS 포드의 경우 배포할 가용 영역을 선택할 수 있습니다. 일부 설계에서는 Elastic Load Balancing이 비정상 영역에서 인스턴스를 감지하고 정상적인 영역으로 트래픽을 라우팅합니다. Elastic Load Balancing을 사용하는 경우 온프레미스 데이터 센터의 구성 요소로 트래픽을 라우팅할 수도 있습니다.

다중 리전 트래픽 장애 조치의 경우 재라우팅은 Amazon Route 53, Amazon Application Recovery Controller, AWS Global Accelerator, Route 53 Private DNS for VPC 또는 CloudFront를 통해 인터넷 도메인을 정의하고 상태 확인을 포함한 라우팅 정책을 할당하여 트래픽을 정상 리전으로 라우팅하는

방법을 제공할 수 있습니다. AWS Global Accelerator에서는 애플리케이션에 대한 고정 진입점 역할을 하는 고정 IP 주소를 제공한 다음, 성능 및 신뢰성 향상을 위해 인터넷 대신 AWS 글로벌 네트워크를 사용하여 선택한 AWS 리전 엔드포인트로 라우팅합니다.

구현 단계

- 모든 적절한 애플리케이션과 서비스를 위한 장애 조치 설계를 생성합니다. 각 아키텍처 구성 요소를 분리하고 각 구성 요소에 대한 RTO 및 RPO를 충족하는 장애 조치 설계를 생성합니다.
- 장애 조치 계획을 수립하는 데 필요한 모든 서비스를 갖춘 하위 환경(예: 개발 또는 테스트)을 구성합니다. 코드형 인프라(IaC)를 사용하여 솔루션을 배포해 반복성을 보장합니다.
- 복구 사이트(예: 보조 리전)를 구성하여 장애 조치 설계를 구현하고 테스트합니다. 필요한 경우 테스트 리소스를 임시로 구성하여 추가 비용을 제한할 수 있습니다.
- AWS를 통해 어떤 장애 조치 계획을 자동화할지, DevOps 프로세스로 어떤 장애 조치 계획을 자동화할 수 있는지, 어떤 장애 조치 계획을 수동으로 할지 결정하세요. 각 서비스의 RTO 및 RPO를 문서화하고 측정합니다.
- 장애 조치 플레이북을 만들고 각 리소스, 애플리케이션, 서비스를 장애 조치하기 위한 모든 단계를 포함하세요.
- 페일백 플레이북을 만들고 각 리소스, 애플리케이션, 서비스를 페일백하기 위한 모든 단계(타이밍 포함)를 포함합니다.
- 계획을 세워 플레이북을 시작하고 연습하세요. 시뮬레이션과 카오스 테스트를 사용하여 플레이북 단계 및 자동화를 테스트하세요.
- 위치 장애(예: 가용 영역 또는 AWS 리전)의 경우, 중단되지 않은 위치의 정상 리소스로 장애 조치할 수 있는 시스템을 갖추고 있어야 합니다. 장애 조치 테스트 전에 할당량, 자동 규모 조정 수준, 실행 중인 리소스를 확인하세요.

## 리소스

관련 Well-Architected 모범 사례:

- REL13 - 재해 복구 계획
- REL10 - 장애 격리를 사용하여 워크로드 보호

관련 문서:

- Setting RTO and RPO Targets
- Failover using Route 53 Weighted routing

- Amazon Application Recovery Controller를 사용하여 재해 복구
- EC2 with autoscaling
- EC2 Deployments - Multi-AZ
- ECS Deployments - Multi-AZ
- Amazon Application Recovery Controller를 사용하여 트래픽 전환
- Lambda with an Application Load Balancer and Failover
- ACM Replication and Failover
- Parameter Store Replication and Failover
- ECR cross region replication and Failover
- Secrets manager cross region replication configuration
- Enable cross region replication for EFS and Failover
- EFS Cross Region Replication and Failover
- Networking Failover
- S3 Endpoint failover using MRAP
- S3에 대한 크로스 리전 복제 생성
- AWS에서 리전 간 장애 조치 및 Graceful 장애 복구에 대한 지침
- Failover using multi-region global accelerator
- Failover with DRS

관련 예제:

- Disaster Recovery on AWS
- Elastic Disaster Recovery on AWS

## REL11-BP03 모든 계층에서 복구 자동화

장애가 감지되면 자동화된 기능을 사용하여 수정 작업을 수행합니다. 성능 저하는 내부 서비스 메커니즘을 통해 자동으로 해결되거나 수정 조치를 통해 리소스를 재시작하거나 제거해야 할 수 있습니다.

자체 관리 애플리케이션 및 크로스 리전 간 복구를 위해 기존 모범 사례에서 복구 설계 및 자동화된 복구 프로세스를 가져올 수 있습니다.

리소스를 재시작하거나 제거하는 기능은 장애를 해결하는 데 중요한 도구입니다. 가장 좋은 방법은 가능한 경우 서비스를 상태 비저장으로 만드는 것입니다. 이렇게 하면 리소스 재시작 시 데이터 손실 또

는 가용성이 손실되는 것을 방지할 수 있습니다. 클라우드에서는 재시작의 일부로 전체 리소스(예: 컴퓨팅 인스턴스 또는 서버리스 함수)를 대체할 수 있으며 이러한 대체는 일반적으로 필수적입니다. 재시작은 그 자체로 장애를 복구할 수 있는 단순하면서도 신뢰할 수 있는 방법입니다. 워크로드에는 다양한 유형의 장애가 발생합니다. 장애는 하드웨어, 소프트웨어, 통신 및 작업 과정에서 발생할 수 있습니다.

재시작 또는 재시도는 네트워크 요청에도 적용됩니다. 네트워크 시간 제한 장애와 종속성 장애(종속성이 오류를 반환함)에 대해 같은 복구 방식이 적용됩니다. 두 이벤트는 모두 시스템에 비슷한 영향을 주므로, 한 이벤트를 특수 사례로 처리하는 대신 지수 백오프 및 지터를 통해 제한적으로 재시도하는 비슷한 전략이 적용됩니다. 재시작 기능은 복구 중심 컴퓨팅 및 고가용성 클러스터 아키텍처에 포함된 복구 메커니즘입니다.

원하는 성과: 장애 감지를 해결하기 위해 자동화된 조치가 수행됩니다.

일반적인 안티 패턴:

- 자동 규모 조정 없이 리소스를 프로비저닝합니다.
- 인스턴스 또는 컨테이너에 개별적으로 애플리케이션을 배포합니다.
- 자동 복구를 사용하지 않고 여러 위치에 배포할 수 없는 애플리케이션을 배포합니다.
- 자동 규모 조정 및 자동 복구로 복구하지 못한 애플리케이션을 수동으로 복구합니다.
- 데이터베이스 장애 조치를 자동화할 수 없습니다.
- 트래픽을 새 엔드포인트로 재라우팅하는 자동화된 방법이 부족합니다.
- 스토리지 복제가 없습니다.

이 모범 사례 확립의 이점: 자동 복구를 통해 평균 복구 시간을 줄이고 가용성을 개선할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

Amazon EKS 또는 기타 Kubernetes 서비스를 위한 설계에는 최소 및 최대 복제본 또는 상태 저장 세트와 최소 클러스터 및 노드 그룹 크기가 모두 포함되어야 합니다. 이러한 메커니즘은 Kubernetes 컨트롤 플레인을 사용하여 장애를 자동으로 해결하면서 지속적으로 사용할 수 있는 최소한의 처리 리소스를 제공합니다.

컴퓨팅 클러스터를 사용하여 로드 밸런서를 통해 액세스하는 설계 패턴은 Auto Scaling 그룹을 활용해야 합니다. Elastic Load Balancing(ELB)은 애플리케이션 인바운드 트래픽을 하나 이상의 가용 영역에 있는 여러 대상 및 가상 어플라이언스에 자동으로 분산합니다.

부하 분산을 사용하지 않는 클러스터링된 컴퓨팅 기반 설계는 최소 하나 이상의 노드 손실을 고려하여 크기가 설계되어야 합니다. 이렇게 하면 새 노드를 복구하는 동안 서비스가 잠재적으로 줄어든 용량으로 계속 운영될 수 있습니다. 서비스 예로는 Mongo, DynamoDB Accelerator, Amazon Redshift, Amazon EMR, Cassandra, Kafka, MSK-EC2, Couchbase, ELK, Amazon OpenSearch Service가 있습니다. 이러한 서비스 중 다수는 추가 자동 복구 기능을 사용하여 설계할 수 있습니다. 일부 클러스터 기술은 노드가 손실되면 자동 또는 수동 워크플로를 트리거하여 새 노드를 다시 생성하는 경고를 생성해야 합니다. AWS Systems Manager를 사용하여 이 워크플로를 자동화하여 문제를 신속하게 해결할 수 있습니다.

Amazon EventBridge를 사용하면 CloudWatch 경보 또는 다른 AWS 서비스의 상태 변경과 같은 이벤트를 모니터링하고 필터링할 수 있습니다. 그런 다음 이벤트 정보를 기반으로 AWS Lambda, Systems Manager Automation 또는 다른 대상을 간접 호출하여 워크로드에 대한 사용자 지정 수정 로직을 실행할 수 있습니다. EC2 인스턴스 상태를 확인하도록 Amazon EC2 Auto Scaling을 구성할 수 있습니다. 인스턴스가 실행 중 이외의 상태이거나 시스템 상태가 손상된 경우 Amazon EC2 Auto Scaling은 해당 인스턴스를 비정상으로 간주하고 대체 인스턴스를 시작합니다. 대규모 교체(예: 전체 가용 영역이 손실됨)의 경우 정적 안정성을 통해 고가용성을 유지하는 것이 좋습니다.

구현 단계

- Auto Scaling 그룹을 사용하여 워크로드에 티어를 배포합니다. Auto Scaling은 상태 비저장 애플리케이션에서 자가 복구를 수행하고 용량을 추가 및 제거할 수 있습니다.
- 앞서 언급한 컴퓨팅 인스턴스의 경우 로드 밸런싱을 사용하고 적절한 유형의 로드 밸런서를 선택합니다.
- Amazon RDS에서 복구를 고려합니다. 대기 인스턴스를 사용하여 대기 인스턴스로의 자동 장애 조치를 구성합니다. Amazon RDS 읽기 전용 복제본의 경우 읽기 전용 복제본을 기본 복제본으로 만들려면 자동화된 워크플로가 필요합니다.
- 여러 위치에 배포할 수 없고 장애 발생 시 재부팅이 허용되는 애플리케이션이 배포되어 있는 EC2 인스턴스에 자동 복구를 구현합니다. 애플리케이션을 여러 위치에 배포할 수 없는 경우 자동 복구를 사용하여 장애가 발생한 하드웨어를 교체하고 인스턴스를 다시 시작할 수 있습니다. 인스턴스 메타데이터 및 관련 IP 주소가 유지되며, EBS 볼륨과 Amazon Elastic File System 및 File Systems for Lustre 및 File Systems for Windows에 대한 탑재 지점도 유지됩니다. AWS OpsWorks를 사용하면 계층 수준에서 EC2 인스턴스의 자동 복구를 구성할 수 있습니다.
- 자동 규모 조정 또는 자동 복구를 사용할 수 없거나 자동 복구가 실패할 경우 AWS Step Functions 및 AWS Lambda를 사용하여 자동 복구를 구현합니다. 자동 크기 조정을 사용할 수 없고, 자동 복구를 사용할 수 없거나 자동 복구가 실패하는 경우 AWS Step Functions 및 AWS Lambda를 사용하여 복구를 자동화할 수 있습니다.

- Amazon EventBridge를 사용하면 CloudWatch 경보 또는 다른 AWS 서비스의 상태 변경과 같은 이벤트를 모니터링하고 필터링할 수 있습니다. 그런 다음 이벤트 정보를 기반으로 AWS Lambda(또는 다른 대상)를 간접 호출하여 워크로드에 대한 사용자 지정 수정 로직을 실행할 수 있습니다.

## 리소스

관련 모범 사례:

- 가용성 정의
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지

관련 문서:

- AWS Auto Scaling 작동 방식
- Amazon EC2 자동 복구
- Amazon Elastic Block Store(Amazon EBS)
- Amazon Elastic File System(Amazon EFS)
- What is Amazon FSx for Lustre?
- What is Amazon FSx for Windows File Server?
- AWS OpsWorks: Using Auto Healing to Replace Failed Instances
- AWS Step Functions란 무엇입니까?
- AWS Lambda란 무엇입니까?
- Amazon EventBridge란 무엇인가요?
- Amazon CloudWatch 경보 사용
- Amazon RDS Failover
- SSM - Systems Manager Automation
- Resilient Architecture Best Practices

관련 비디오:

- Automatically Provision and Scale OpenSearch Service
- Amazon RDS Failover Automatically

관련 예제:

- Amazon RDS Failover 워크숍

관련 도구:

- CloudWatch
- CloudWatch X-Ray

## REL11-BP04 복구 중 컨트롤 플레인이 아닌 데이터 플레인 사용

컨트롤 플레인은 리소스를 생성, 읽기 및 설명, 업데이트, 삭제, 나열(CRUDL)하는 데 사용되는 관리 API를 제공하는 반면, 데이터 플레인은 일상적인 서비스 트래픽을 처리합니다. 복원력에 영향을 미칠 수 있는 이벤트에 대한 복구 또는 완화 대응을 구현할 때는 최소한의 컨트롤 플레인 작업을 사용하여 서비스를 복구, 재조정, 복원, 복구 또는 장애 조치하는 데 집중합니다. 데이터 플레인 작업은 이러한 성능 저하 이벤트가 발생하는 동안의 모든 활동을 대체해야 합니다.

예를 들어, 새 컴퓨팅 인스턴스 시작, 블록 스토리지 생성, 대기열 서비스 설명 등은 모두 컨트롤 플레인 작업입니다. 컴퓨팅 인스턴스를 시작할 때 컨트롤 플레인은 용량이 있는 물리적 호스트 찾기, 네트워크 인터페이스 할당, 로컬 블록 스토리지 볼륨 준비, 자격 증명 생성, 보안 규칙 추가와 같은 여러 작업을 수행해야 합니다. 컨트롤 플레인은 복잡한 오케스트레이션이 필요한 경우가 많습니다.

원하는 성과: 리소스가 손상된 상태가 되면 시스템은 트래픽을 손상된 리소스에서 정상 리소스로 전환하여 자동 또는 수동으로 복구할 수 있습니다.

일반적인 안티 패턴:

- 트래픽을 재라우팅하기 위해 DNS 레코드 변경에 의존합니다.
- 불충분하게 프로비저닝된 리소스로 인해 손상된 구성 요소를 대체하기 위한 컨트롤 플레인 규모 조정 작업에 의존합니다.
- 광범위한 다중 서비스, 다중 API 컨트롤 플레인 조치를 활용하여 모든 범주의 장애를 해결합니다.

이 모범 사례 확립의 이점: 자동화된 문제 해결의 성공률이 높아지면 평균 시간이 단축되고 워크로드의 가용성이 향상됩니다.

이 모범 사례가 확립되지 않은 경우 노출되는 위험 수준: 중간: 특정 유형의 서비스 성능 저하에서는 컨트롤 플레인이 영향을 받습니다. 개선을 위해 컨트롤 플레인을 광범위하게 사용하는 것에 의존하면 복구 시간(RTO)과 평균 복구 시간(MTTR)이 증가할 수 있습니다.

## 구현 지침

데이터 플레인 작업을 제한하려면 서비스를 복원하는 데 필요한 조치가 무엇인지 각 서비스를 평가하세요.

Amazon Application Recovery Controller를 활용하여 DNS 트래픽을 전환합니다. 이러한 기능은 애플리케이션의 장애 복구 성능을 지속적으로 모니터링하며, 이를 통해 여러 AWS 리전, 가용 영역 및 온프레미스에서 애플리케이션 복구를 제어할 수 있습니다.

Route 53 라우팅 정책은 컨트롤 플레인을 사용하므로 복구 시 컨트롤 플레인을 사용하지 마세요. Route 53 데이터 플레인은 DNS 쿼리에 응답하고 상태 확인을 수행 및 평가합니다. 전 세계에 분산되어 있으며, 100% 가용성 서비스 수준에 관한 계약(SLA)을 위해 설계됩니다.

Route 53 리소스를 생성, 업데이트, 삭제하는 데 사용하는 Route 53 관리 API 및 콘솔은 DNS를 관리할 때 필요한 강력한 일관성과 내구성을 우선시하도록 설계된 컨트롤 플레인에서 실행됩니다. 이를 위해 컨트롤 플레인은 하나의 리전(미국 동부, 버지니아 북부)에 위치합니다. 두 시스템 모두 신뢰성이 높게 구축되었지만 컨트롤 플레인은 SLA에 포함되지 않습니다. 데이터 플레인의 복원력 높은 설계가 컨트롤 플레인과 달리 가용성을 유지하는 상황이 드물게 있을 수 있습니다. 재해 복구 및 장애 조치 메커니즘에는 데이터 플레인 기능을 사용하여 가능한 한 최고 수준의 신뢰성을 제공합니다.

인시던트 중에 컨트롤 플레인을 사용하지 않도록 컴퓨팅 인프라를 정적으로 안정적으로 설계합니다. 예를 들어 Amazon EC2 인스턴스를 사용하는 경우 새 인스턴스를 수동으로 프로비저닝하거나 Auto Scaling 그룹에 응답으로 인스턴스를 추가하도록 지시하지 마십시오. 최고 수준의 복원력을 얻으려면 장애 조치에 사용되는 클러스터에 충분한 용량을 프로비저닝하세요. 이 용량 임곗값을 제한해야 하는 경우 전체 엔드 투 엔드 시스템에 제한을 설정하여 총 트래픽이 제한된 리소스 세트에 도달하도록 안전하게 제한합니다.

Amazon DynamoDB, Amazon API Gateway, 로드 밸런서 및 AWS Lambda 서버리스와 같은 서비스의 경우 이러한 서비스를 사용하면 데이터 플레인을 활용할 수 있습니다. 하지만 새 함수, 로드 밸런서, API 게이트웨이 또는 DynamoDB 테이블을 생성하는 것은 컨트롤 플레인 작업이므로 이벤트 준비 및 장애 조치 리허설로 성능 저하 전에 완료해야 합니다. Amazon RDS의 경우, 데이터 플레인 작업을 통해 데이터에 액세스할 수 있습니다.

데이터 플레인, 컨트롤 플레인 및 AWS가 고가용성 목표를 충족하기 위해 서비스를 구축하는 방법에 대한 자세한 내용은 가용 영역을 사용한 정적 안정성DMF 참조하세요.

데이터 플레인을 사용할 작업과 컨트롤 플레인을 사용할 작업을 파악합니다.

구현 단계

성능 저하 이벤트 후 복원해야 하는 각 워크로드에 대해 장애 조치 런북, 고가용성 설계, 자동 복구 설계 또는 HA 리소스 복원 계획을 평가하세요. 컨트롤 플레인 작업으로 간주될 수 있는 각 작업을 식별하세요.

컨트롤 작업을 데이터 플레인 작업으로 변경하는 것을 고려해 보세요.

- 오토 스케일링(컨트롤 플레인)과 사전 규모 조정된 Amazon EC2 리소스(데이터 플레인) 비교
- Amazon EC2 인스턴스 조정(컨트롤 플레인)과 AWS Lambda 조정(데이터 플레인) 비교
- Kubernetes를 사용하는 모든 설계와 컨트롤 플레인 작업의 특성을 평가하세요. 포드를 추가하는 것은 Kubernetes의 데이터 플레인 작업입니다. 작업은 포드를 추가하고 노드는 추가하지 않는 것으로 제한해야 합니다. 과다 프로비저닝된 노드 사용은 컨트롤 플레인 작업을 제한하는 데 선호되는 방법입니다.

데이터 플레인 작업이 동일한 문제 해결에 영향을 줄 수 있는 다른 접근 방식을 고려해 보세요.

- Route 53 레코드 변경(컨트롤 플레인) 또는 Amazon Application Recovery Controller(데이터 플레인)
- 보다 자동화된 업데이트를 위한 Route 53 상태 확인

서비스가 업무상 중요한 경우 영향을 받지 않는 리전에서 더 많은 컨트롤 플레인 및 데이터 플레인 작업을 수행할 수 있도록 보조 리전의 일부 서비스를 고려해 보세요.

- 기본 리전의 Amazon EC2 Auto Scaling 또는 Amazon EKS와 보조 리전의 Amazon EC2 Auto Scaling 또는 Amazon EKS 비교 및 보조 리전으로의 트래픽 라우팅(컨트롤 플레인 작업)
- 보조 기본 리전에서 읽기 전용 복제본을 만들거나 기본 리전에서 동일한 작업 시도(컨트롤 플레인 작업)

## 리소스

관련 모범 사례:

- 가용성 정의
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지

관련 문서:

- APN 파트너: 내결함성 자동화를 지원할 수 있는 파트너
- AWS Marketplace: 내결함성에 사용할 수 있는 제품
- Amazon Builders' Library: Avoiding overload in distributed systems by putting the smaller service in control
- Amazon DynamoDB API(컨트롤 플레인 및 데이터 플레인)
- AWS Lambda 실행(컨트롤 플레인 및 데이터 플레인으로 분할)
- AWS Elemental MediaStore 데이터 플레인
- Building highly resilient applications using Amazon Application Recovery Controller, Part 1: Single-Region stack
- Building highly resilient applications using Amazon Application Recovery Controller, Part 2: Multi-Region stack
- Creating Disaster Recovery Mechanisms Using Amazon Route 53
- Amazon Application Recovery Controller란 무엇입니까?
- Kubernetes 컨트롤 플레인 및 데이터 플레인

관련 비디오:

- Back to Basics - Using Static Stability
- Building resilient multi-site workloads using AWS global services

관련 예제:

- Amazon Application Recovery Controller 소개
- Amazon Builders' Library: Avoiding overload in distributed systems by putting the smaller service in control
- Building highly resilient applications using Amazon Application Recovery Controller, Part 1: Single-Region stack
- Building highly resilient applications using Amazon Application Recovery Controller, Part 2: Multi-Region stack
- 가용 영역을 사용한 정적 안정성

관련 도구:

- Amazon CloudWatch
- AWS X-Ray

## REL11-BP05 정적 안정성을 사용하여 바이모달 동작 방지

워크로드는 정적으로 안정적인 상태를 유지하고 단일 정상 모드에서만 작동해야 합니다. 바이모달 동작은 정상 모드와 장애 모드에서 워크로드가 서로 다른 동작을 보일 때를 말합니다.

예를 들어 서로 다른 가용 영역에서 새 인스턴스를 시작하여 가용 영역 장애 복구를 시도할 수 있습니다. 이로 인해 장애 모드 중에 바이모달 응답이 발생할 수 있습니다. 그러나 이 방법 대신 정적으로 안정적이며 한 모드에서만 작동하는 워크로드를 구축해야 합니다. 이 예제에서 이러한 인스턴스는 장애가 발생하기 전에 두 번째 가용 영역에 프로비저닝되었어야 합니다. 이 정적 안정성 설계는 워크로드가 단일 모드에서만 작동하는지 확인합니다.

원하는 성과: 정상 모드와 장애 모드에서 워크로드가 바이모달 동작을 보이지 않습니다.

일반적인 안티 패턴:

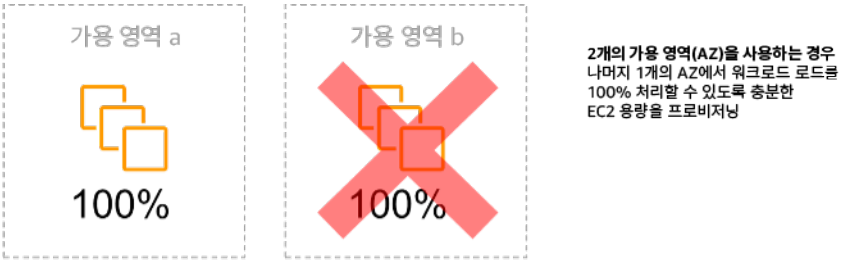
- 장애 범위에 관계없이 항상 리소스를 프로비저닝할 수 있다고 가정합니다.
- 장애 발생 시 동적으로 리소스를 확보하려고 시도합니다.
- 장애가 발생할 때까지 여러 영역 또는 리전에 걸쳐 적절한 리소스를 프로비저닝하지 않습니다.
- 컴퓨팅 리소스에 대해서만 정적이고 안정적인 설계를 고려합니다.

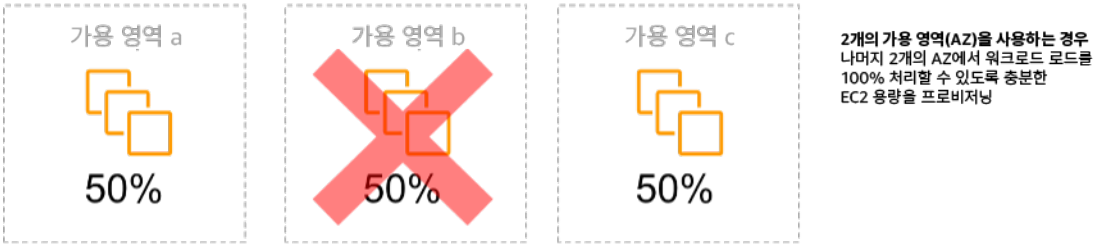
이 모범 사례 확립의 이점: 정적이고 안정적인 설계로 실행되는 워크로드는 정상 및 장애 이벤트 발생 시 예측 가능한 결과를 얻을 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

### 구현 가이드

바이모달 동작은 워크로드가 정상 모드와 장애 모드에서 다른 동작을 보일 때 발생합니다. 예를 들어 가용 영역에 장애가 발생할 경우 새 인스턴스를 시작하는 방법을 사용할 수 있습니다. 바이모달 동작의 예로는 한 AZ가 제거된 경우 안정적인 Amazon EC2 설계가 워크로드 로드를 처리하기에 충분한 인스턴스를 각 가용 영역에 프로비저닝하는 경우를 들 수 있습니다. 그런 다음 Elastic Load Balancing 또는 Amazon Route 53 상태를 확인하여 손상된 인스턴스로부터 로드를 이동합니다. 트래픽이 전환된 후에는 AWS Auto Scaling을 사용하여 장애가 발생한 영역의 인스턴스를 비동기식으로 교체하고 정상 영역에서 인스턴스를 시작합니다. 컴퓨팅 배포(예: EC2 인스턴스 또는 컨테이너)에서 정적 안정성은 최고의 신뢰성을 제공합니다.




가용 영역에 걸친 EC2 인스턴스의 정적 안정성

모든 복원력 사례에서 워크로드를 유지 관리하는 데 따르는 비즈니스 가치 및 이 모델의 비용을 이 정적 안정성과 비교해야 합니다. 컴퓨팅 용량을 적게 프로비저닝하고 장애 발생 시 새 인스턴스를 시작하는 방법이 비용은 더 저렴하지만, 대규모 장애(예: 가용 영역 또는 리전 장애)의 경우 이 접근 방식은 영향을 받지 않는 영역 또는 리전에서 제공되는 충분한 리소스와 운영 플레인을 활용하기 때문에 덜 효과적입니다.

따라서 워크로드의 신뢰성 요구 사항과 비용 요구 사항도 비교해야 합니다. 정적 안정성 아키텍처는 가용 영역에 분산된 컴퓨팅 인스턴스, 데이터베이스 읽기 전용 복제본 설계, Kubernetes(Amazon EKS) 클러스터 설계, 다중 리전 장애 조치 아키텍처 등 다양한 아키텍처에 적용됩니다.

또한 각 영역에서 더 많은 리소스를 사용하여 더 정적으로 안정적인 설계를 구현할 수도 있습니다. 영역을 더 추가할수록 정적 안정성을 유지하는 데 필요한 추가 컴퓨팅 용량은 줄어듭니다.

바이모달 동작의 한 예는 시스템이 전체 시스템의 구성 상태를 새로 고치려고 시도할 수 있는 네트워크 시간 제한입니다. 그러면 다른 구성 요소에 예기치 않은 로드가 더해져 해당 구성 요소에 장애가 발생하고 또 다른 예기치 않은 결과가 이어질 수 있습니다. 이 부정적인 피드백 루프는 워크로드의 가용성에 영향을 미칩니다. 대신 정적으로 안정적이며 한 모드에서만 작동하는 시스템을 구축할 수 있습니다. 일정한 작업을 수행하고 항상 고정된 케이던스에서 구성 상태를 새로 고치는 것이 정적으로 안정적인 설계의 하나일 것입니다. 호출이 실패하면 워크로드는 이전에 캐시된 값을 사용하고 경보를 트리거합니다.

바이모달 동작의 또 다른 예로 장애 발생 시 클라이언트에서 워크로드 캐시를 우회하는 것을 허용하는 동작이 있습니다. 이는 클라이언트 요구 사항을 수용한 솔루션처럼 보이지만 워크로드의 수요가 크게 변경될 수 있고 장애를 초래할 가능성이 높습니다.

중요 워크로드를 평가하여 이러한 유형의 복원력 설계가 필요한 워크로드를 결정합니다. 중요하다고 판단되는 워크로드에 대해서는 각 애플리케이션 구성 요소를 검토해야 합니다. 정적 안정성 평가가 필요한 서비스 유형의 예는 다음과 같습니다.

- 컴퓨팅: Amazon EC2, EKS-EC2, ECS-EC2, EMR-EC2
- 데이터베이스: Amazon Redshift, Amazon RDS, Amazon Aurora
- 스토리지: Amazon S3(단일 영역), Amazon EFS(탑재), Amazon FSx(탑재)
- 로드 밸런서: 특정 설계 시

구현 단계

- 정적으로 안정적이고 한 모드에서만 작동하는 시스템을 구축합니다. 이 경우 가용 영역 또는 리전 하나가 제거된 경우 각 가용 영역 또는 리전에서 워크로드 용량을 처리할 만큼 충분한 인스턴스를 프로비저닝합니다. 다음과 같은 다양한 서비스를 사용하여 정상 리소스로 라우팅할 수 있습니다.
  - 크로스 리전 DNS 라우팅
  - MRAP Amazon S3 다중 리전 라우팅
  - AWS Global Accelerator
  - Amazon Application Recovery Controller
- 단일 기본 인스턴스 또는 읽기 전용 복제본의 손실을 고려하도록 데이터베이스 읽기 복제본을 구성합니다. 읽기 전용 복제본으로 트래픽을 처리하는 경우 각 가용 영역 및 각 리전의 용량은 영역 또는 리전 장애 발생 시 필요한 전체 용량과 동일해야 합니다.
- 가용 영역 장애 발생 시 저장된 데이터에 대해 정적으로 안정적으로 설계된 Amazon S3 스토리지에서 중요한 데이터를 구성합니다. Amazon S3 One Zone-IA 스토리지 클래스를 사용하는 경우, 해당 영역이 손실되면 저장된 데이터에 대한 액세스가 최소화되므로 이 스토리지 클래스를 정적으로 안정적인 것으로 간주해서는 안 됩니다.
- 로드 밸런서가 잘못 구성되거나 특정 가용 영역을 서비스하도록 설계되는 경우가 간혹 있습니다. 이 경우 보다 복잡한 설계에서 여러 AZ에 워크로드를 분산하는 것이 정적으로 안정적인 설계일 수 있습니다. 원래 설계는 보안, 지연 시간 또는 비용상의 이유로 영역 간 트래픽을 줄이는 데 사용될 수 있습니다.

## 리소스

관련 Well-Architected 모범 사례:

- 가용성 정의

- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지
- REL11-BP04 복구 중 컨트롤 플레인이 아닌 데이터 플레인 사용

관련 문서:

- Minimizing Dependencies in a Disaster Recovery Plan
- Amazon Builders' Library: 가용 영역을 사용한 정적 안정성
- Fault Isolation Boundaries
- 가용 영역을 사용한 정적 안정성
- 다중 영역 RDS
- Minimizing Dependencies in a Disaster Recovery Plan
- 크로스 리전 DNS 라우팅
- MRAP Amazon S3 다중 리전 라우팅
- AWS Global Accelerator
- Amazon Application Recovery Controller
- 단일 영역 Amazon S3
- Cross Zone Load Balancing

관련 비디오:

- Static stability in AWS: AWS re:Invent 2019: Introducing The Amazon Builders' Library (DOP328)

## REL11-BP06 이벤트가 가용성에 영향을 미치는 경우 알림 전송

임곗값 위반이 감지되면 문제를 일으킨 이벤트가 자동으로 해결된 경우에도 알림이 전송됩니다.

자동 복구를 사용하면 워크로드의 신뢰성을 유지할 수 있습니다. 그러나 자동 복구로 인해 해결해야 할 근본적인 문제가 가려질 수도 있습니다. 적절한 모니터링 및 이벤트를 구현하면 자동 복구로 해결된 문제를 포함한 문제의 패턴을 감지하여 근본 원인 문제를 해결할 수 있습니다.

복원력이 뛰어난 시스템은 성능 저하 이벤트가 해당 팀에 즉시 전달되도록 설계되었습니다. 이러한 알림은 하나 이상의 통신 채널을 통해 전송되어야 합니다.

원하는 성과: 오류율, 지연 시간 또는 기타 중요한 핵심 성과 지표(KPI)와 같은 임곗값을 위반하면 운영 팀에 즉시 알림이 전송되므로 이러한 문제를 최대한 빨리 해결하고 사용자에게 미치는 영향을 피하거나 최소화할 수 있습니다.

일반적인 안티 패턴:

- 너무 많은 경보를 전송합니다.
- 실행 불가능한 경보를 전송합니다.
- 경보 임곗값을 너무 높거나(민감도 높음) 너무 낮게(민감도 낮음) 설정합니다.
- 외부 종속성에 대한 경보를 전송하지 않습니다.
- 모니터링 및 경보를 설계할 때 회색 장애를 고려하지 않습니다.
- 복구 자동화를 수행하지만 해당 팀에 복구가 필요하다는 사실을 알리지 않습니다.

이 모범 사례 확립의 이점: 운영 팀과 비즈니스 팀은 복구 알림을 통해 서비스 저하를 인지하고 즉시 대응하여 평균 탐지 시간(MTTD)과 평균 복구 시간(MTTR)을 모두 최소화할 수 있습니다. 또한 복구 이벤트에 대한 알림이 전송되면 어쩌다 발생하는 문제도 지나치지 않을 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간. 적절한 모니터링 및 이벤트 알림 메커니즘을 구현하지 않으면 자동 복구로 해결된 문제를 포함하여 문제의 패턴을 감지하지 못할 수 있습니다. 팀은 사용자가 고객 서비스에 문의할 때나 우연한 경우에만 시스템 성능 저하 사실을 인지합니다.

## 구현 지침

모니터링 전략을 정의할 때 경보가 울리는 것은 흔한 이벤트입니다. 이 이벤트에는 경보의 식별자, 즉 경보 상태(예: `IN ALARM` 또는 `OK`) 및 유발 원인에 대한 세부 정보가 포함되어 있을 것입니다. 대부분의 경우 경보 이벤트가 감지되고 이메일 알림이 전송되어야 합니다. 이것이 경보에 대한 작업의 예입니다. 경보 알림은 적절한 사용자에게 문제가 있음을 알려주기 때문에 관찰성에서 매우 중요합니다. 그러나 관찰성 솔루션에서 이벤트에 대한 작업이 성숙해지면 사람의 개입 없이 자동으로 문제를 해결할 수 있습니다.

KPI 모니터링 경보가 설정되면 임곗값을 초과할 경우 해당 팀에 알림이 전송되어야 합니다. 이러한 알림은 성능 저하를 해결하기 위한 자동화된 프로세스를 트리거하는 데에도 사용될 수 있습니다.

보다 복잡한 임곗값 모니터링의 경우 복합 경보를 고려해야 합니다. 복합 경보는 여러 KPI 모니터링 경보를 사용하여 운영 비즈니스 로직에 기반한 알림을 생성합니다. 이메일을 보내거나 Amazon SNS 통합 또는 Amazon EventBridge를 사용하여 서드파티 인시던트 추적 시스템에 인시던트를 기록하도록 CloudWatch 경보를 구성할 수 있습니다.

### 구현 단계

워크로드 모니터링 방식에 따라 다음과 같은 다양한 유형의 경보를 생성합니다.

- 애플리케이션 경보는 워크로드의 일부가 제대로 작동하지 않는 경우를 감지하는 데 사용됩니다.
- 인프라 경보는 리소스 규모를 조정할 시기를 나타냅니다. 경보를 사용하면 대시보드에 경보를 시각적으로 표시하고, Amazon SNS 또는 이메일을 통해 알림을 전송하며, Auto Scaling을 통해 워크로드의 리소스를 스케일 인 또는 스케일 아웃할 수 있습니다.
- 지정된 평가 기간에 지표가 정적 임곗값을 위반하는 시점을 모니터링하기 위해 간단한 정적 경보를 생성할 수 있습니다.
- 여러 소스의 복잡한 경보에 대해서는 복합 경보를 고려할 수 있습니다.
- 경보가 생성되면 적절한 알림 이벤트를 생성합니다. Amazon SNS API 간접 호출을 직접 수행하여 알림을 보내고 문제 해결 또는 커뮤니케이션을 위한 자동화를 연결할 수 있습니다.
- AWS Health를 사용하여 서비스 성능 저하에 대한 최신 정보를 확인하세요. AWS 사용자 알림를 통해 이메일 및 채팅 채널에 적합한 AWS Health 이벤트 알림을 생성하고, Amazon EventBridge를 통해 모니터링 및 알림 도구와 프로그래밍 방식으로 통합할 수 있습니다.

## 리소스

관련 Well-Architected 모범 사례:

- 가용성 정의

관련 문서:

- 정적 임곗값을 기반으로 CloudWatch 경보 생성
- Amazon EventBridge란 무엇인가요?
- What is Amazon Simple Notification Service?
- 사용자 지정 지표 게시
- Amazon CloudWatch 경보 사용
- CloudWatch 복합 경보 설정
- What's new in AWS Observability at re:Invent 2022

관련 도구:

- CloudWatch
- CloudWatch X-Ray

## REL11-BP07 가용성 목표 및 가동 시간 서비스 수준에 관한 계약(SLA)을 충족하도록 제품 설계

가용성 목표 및 가동 시간 서비스 수준에 관한 계약(SLA)을 충족하도록 제품을 설계합니다. 가용성 목표 또는 가동 시간 SLA를 게시하거나 비공개로 동의하는 경우 아키텍처 및 운영 프로세스가 이를 지원하도록 설계되었는지 확인합니다.

원하는 성과: 각 애플리케이션에는 비즈니스 성과를 달성하기 위해 모니터링 및 유지 관리할 수 있는 성과 지표에 대해 정의된 가용성 및 SLA 목표가 있습니다.

일반적인 안티 패턴:

- SLA를 설정하지 않고 워크로드를 설계 및 배포합니다.
- SLA 지표가 근거나 비즈니스 요구 사항 없이 너무 높게 설정됩니다.
- 종속성 및 기본 SLA를 고려하지 않고 SLA를 설정합니다.
- 애플리케이션 설계는 복원력에 대한 공동 책임 모델을 고려하지 않고 생성됩니다.

이 모범 사례 확립의 이점: 주요 복원력 목표를 기반으로 애플리케이션을 설계하면 비즈니스 목표와 고객 기대치를 충족하는 데 도움이 됩니다. 이러한 목표는 다양한 기술을 평가하고 다양한 장단점을 고려하는 애플리케이션 설계 프로세스를 추진하는 데 도움이 됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

### 구현 지침

애플리케이션 설계는 비즈니스, 운영 및 재무 목표에서 파생된 다양한 요구 사항 세트를 고려해야 합니다. 운영 요구 사항 내에서 워크로드는 적절하게 모니터링되고 지원될 수 있도록 특정 복원력 지표 대상을 가져야 합니다. 복원력 지표는 워크로드를 배포한 후에 설정하거나 파생해서는 안 됩니다. 설계 단계에서 정의해야 하며 다양한 결정과 장단점을 안내하는 데 도움이 됩니다.

- 모든 워크로드에는 고유한 복원력 지표 세트가 있어야 합니다. 이러한 지표는 다른 비즈니스 애플리케이션과 다를 수 있습니다.
- 종속성을 줄이면 가용성에 긍정적인 영향을 미칠 수 있습니다. 각 워크로드는 종속성과 해당 SLA를 고려해야 합니다. 일반적으로 가용성 목표가 워크로드 목표 이상인 종속성을 선택합니다.

- 가능한 경우 종속성 손상에도 불구하고 워크로드가 올바르게 작동할 수 있도록 느슨하게 결합된 설계를 고려하세요.
- 특히 복구 또는 성능 저하 중에 컨트롤 플레인 종속성을 줄입니다. 미션 크리티컬 워크로드에 대해 정적으로 안정적인 설계를 평가합니다. 리소스 스페어링을 사용하여 워크로드에서 이러한 종속성의 가용성을 높입니다.
- 평균 탐지 시간(MTTD 및 평균 복구 시간(MTTR)을 줄임으로써 SLA를 달성하기 위해서는 관찰성과 계측이 중요합니다.
- 고장 빈도 감소(MTBF 연장), 고장 감지 시간 단축(MTTD 단축), 수리 시간 단축(MTTR 단축)은 분산 시스템에서 가용성을 개선하는 데 사용되는 세 가지 요소입니다.
- 워크로드에 대한 복원력 지표를 설정하고 충족하는 것은 모든 효과적인 설계의 기초입니다. 이러한 설계는 설계 복잡성, 서비스 종속성, 성능, 확장성 및 비용의 균형을 고려해야 합니다.

구현 단계

- 다음 질문을 고려하여 워크로드 설계를 검토하고 문서화합니다.
  - 워크로드에서 컨트롤 플레인은 어디에 사용되나요?
  - 워크로드는 내결함성을 어떻게 구현하나요?
  - 규모 조정, 자동 규모 조정, 중복성 및 고가용성 구성 요소에 대한 디자인 패턴은 무엇인가요?
  - 데이터 일관성 및 가용성에 대한 요구 사항은 무엇인가요?
  - 리소스 절약 또는 리소스 정적 안정성에 대한 고려 사항이 있나요?
  - 서비스 종속성은 무엇인가요?
- 이해관계자와 협력하면서 워크로드 아키텍처를 기반으로 SLA 지표를 정의합니다. 워크로드에서 사용하는 모든 종속성의 SLA를 고려합니다.
- SLA 목표가 설정되면 SLA를 충족하도록 아키텍처를 최적화합니다.
- SLA를 충족하는 설계가 설정되면 운영 변경, 프로세스 자동화 및 MTTD 및 MTTR 감소에 중점을 둔 런북을 구현합니다.
- 배포되면 SLA를 모니터링하고 보고합니다.

## 리소스

관련 모범 사례:

- REL03-BP01 워크로드를 세그먼트화하는 방법 선택
- REL10-BP01 워크로드를 여러 위치에 배포

- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지
- REL11-BP03 모든 계층에서 복구 자동화
- REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트
- REL13-BP01 가동 중단 시간 및 데이터 손실 시의 복구 목표 정의
- 워크로드 상태 파악

관련 문서:

- Availability with redundancy
- 신뢰성 원칙 - 가용성
- Measuring availability
- AWS Fault Isolation Boundaries
- 복원력을 위한 공동 책임 모델
- 가용 영역을 사용한 정적 안정성
- AWS 서비스 수준에 관한 계약(SLA)
- Guidance for Cell-based Architecture on AWS
- AWS infrastructure
- Advanced Multi-AZ Resilience Patterns whitepaper

관련 서비스:

- Amazon CloudWatch
- AWS Config
- AWS Trusted Advisor

# 신뢰성 테스트

프로덕션 환경의 스트레스에 대한 복원력을 가지도록 워크로드를 설계한 후 설계대로 작동하고 예상한 복원력을 제공하는지 확인할 수 있는 유일한 방법은 테스트입니다.

테스트를 통해 워크로드가 기능 및 비기능적인 요구 사항을 충족하는지 확인합니다. 버그 또는 성능 병목 현상은 워크로드의 안정성에 영향을 미칠 수 있습니다. 워크로드의 복원력을 테스트하면 프로덕션 환경에서만 나타나는 잠재적인 버그를 찾는 데 도움이 됩니다. 이러한 테스트를 정기적으로 수행합니다.

모범 사례

- REL12-BP01 플레이북을 사용하여 장애 조사
- REL12-BP02 인시던트 사후 분석 수행
- REL12-BP03 확장성 및 성능 요구 사항 테스트
- REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트
- REL12-BP05 정기적으로 게임 데이 진행

## REL12-BP01 플레이북을 사용하여 장애 조사

잘 알려지지 않은 장애 시나리오에 일관되고 신속하게 대응할 수 있도록 플레이북에 조사 프로세스를 문서화합니다. 플레이북은 장애 시나리오에 영향을 미치는 요인을 식별하기 위해 수행되는 미리 정의된 단계입니다. 문제가 확인되거나 에스컬레이션될 때까지 각 프로세스 단계의 결과를 사용하여 다음에 수행할 단계를 결정합니다.

플레이북은 사후 대응적 조치를 효과적으로 수행하기 위해 반드시 수행해야 하는 사전 예방적 계획입니다. 플레이북에 포함되지 않은 장애 시나리오가 프로덕션 환경에서 발생할 경우 이 문제를 먼저 해결합니다(진압). 그런 다음 돌아가서 문제를 해결하기 위해 취한 단계를 살펴보고 이를 사용하여 플레이북에 새 항목을 추가합니다.

플레이북은 특정 인시던트에 대응하여 사용되며 런북은 특정 결과를 달성하기 위해 사용됩니다. 런북은 일상적인 활동에 대해 사용되고, 플레이북은 일상적이지 않은 이벤트에 대응하는 데 사용되는 경우가 많습니다.

일반적인 안티 패턴:

- 문제를 진단하거나 인시던트에 대응하는 프로세스를 모른 상태에서 워크로드 배포를 계획합니다.
- 이벤트를 조사할 때 로그 및 지표를 수집할 시스템에 대한 계획되지 않은 의사 결정을 내립니다.
- 데이터를 검색할 수 있을 만큼 오래 지표 및 이벤트를 유지하지 않습니다.

이 모범 사례 확립의 이점: 플레이북을 캡처하면 프로세스를 일관되게 따를 수 있습니다. 플레이북을 코드화하면 수작업으로 인한 오류 발생을 최소화할 수 있습니다. 플레이북을 자동화하면 팀원의 개입 요구 사항을 없애거나 개입을 시작할 때 추가 정보를 제공함으로써 이벤트에 대응하는 시간을 단축할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

- 플레이북을 사용하여 문제를 파악합니다. 플레이북은 문제 조사를 위한 문서화된 프로세스입니다. 플레이북에 프로세스를 문서화하면 장애 발생 시나리오에 일관되고 빠르게 대응할 수 있습니다. 플레이북은 적절한 기술을 보유한 팀원이 해당하는 정보를 수집하고, 장애의 잠재적 출처를 확인하며, 결함 위치를 구분하고, 발생 요인을 확인(인시던트 사후 분석 수행)하는 데 필요한 정보와 지침을 포함해야 합니다.
  - 플레이북을 코드로 구현합니다. 플레이북을 스크립트로 작성하여 작업을 코드로 수행하면 일관성을 유지하고 수동 프로세스에서 발생하는 오류를 최소화할 수 있습니다. 플레이북은 문제의 발생 요인을 식별하는 데 필요할 수 있는 다양한 단계를 나타내는 여러 스크립트로 구성할 수 있습니다. 플레이북 활동의 일부분으로 런북 활동을 간접 호출하거나 수행할 수도 있습니다. 또는 식별된 이벤트의 대응 과정에서 플레이북 실행 여부를 묻는 메시지를 표시할 수도 있습니다.
    - AWS Systems Manager를 사용하여 운영 플레이북 자동화
    - AWS Systems Manager Run Command
    - AWS Systems Manager Automation
    - AWS Lambda란 무엇입니까?
    - Amazon EventBridge란 무엇인가요?
    - Amazon CloudWatch 경보 사용

## 리소스

관련 문서:

- AWS Systems Manager Automation
- AWS Systems Manager Run Command
- AWS Systems Manager를 사용하여 운영 플레이북 자동화
- Amazon CloudWatch 경보 사용
- Using Canaries (Amazon CloudWatch Synthetics)
- Amazon EventBridge란 무엇인가요?
- AWS Lambda란 무엇입니까?

관련 예제:

- Automating operations with Playbooks and Runbooks

# REL12-BP02 인시던트 사후 분석 수행

고객에게 영향을 주는 이벤트를 검토하고 발생 요인과 예방 조치 항목을 식별합니다. 이 정보를 사용하여 재발을 제한하거나 방지하는 완화 기능을 개발합니다. 신속하고 효과적인 대응을 위한 절차를 개발합니다. 목표 대상에 맞게 적절히 발생 요인과 수정 조치를 전달합니다. 필요한 경우 다른 관계자들에게 이러한 원인을 전달하는 방법을 마련합니다.

기존 테스트에서 문제가 발견되지 않은 이유를 평가합니다. 테스트가 아직 없는 경우 이 사례에 대한 테스트를 추가합니다.

원하는 성과: 팀이 인시던트 이후 분석을 처리하기 위해 일관되고 합의된 접근 방식을 취합니다. 한 가지 메커니즘으로 오류 수정 (COE) 프로세스가 있습니다. COE 프로세스는 인시던트의 근본 원인을 식별, 이해 및 해결하는 동시에 동일한 인시던트가 다시 발생할 가능성을 제한하는 메커니즘과 가드레일을 구축하는 데 도움이 됩니다.

일반적인 안티 패턴:

- 발생 요인을 찾지만, 다른 잠재적 문제와 해결 방법을 계속 자세히 살펴보지는 않습니다.
- 인적 오류 원인만 식별하며, 인적 오류를 예방할 수 있는 교육이나 자동화는 제공하지 않습니다.
- 근본 원인을 이해하기보다는 책임을 전가하는 데 집중하여 공포의 문화를 조성하고 열린 의사소통을 방해합니다.
- 인사이트를 공유하는 데 실패하여 인시던트 분석 결과를 소수의 사람만 알고 있고, 배운 교훈을 다른 사람들이 활용하지 못하게 합니다.
- 제도적 지식을 포착할 메커니즘이 없기 때문에 학습한 교훈을 업데이트된 모범 사례의 형태로 보존하지 못해 귀중한 인사이트를 잃고 근본 원인이 동일하거나 유사한 인시던트가 반복적으로 발생합니다.

이 모범 사례 확립의 이점: 인시던트 사후 분석을 수행하고 결과를 공유하면 다른 워크로드에서 동일한 발생 요인이 나타날 경우 위험을 완화할 수 있으며 인시던트가 발생하기 전에 해결하거나 자동 복구를 구현할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

우수한 인시던트 사후 분석은 시스템의 다른 위치에서 사용되는 아키텍처 패턴이 있는 문제에 대해 공통 솔루션을 제안할 수 있는 기회를 제공합니다.

COE 프로세스의 초석은 문제를 문서화하고 해결하는 것입니다. 주요 근본 원인을 문서로 작성하고 검토 및 해결할 수 있는 표준 방식을 정의하는 것이 좋습니다. 인시던트 사후 분석 프로세스에 명확한 소유권을 부여합니다. 인시던트 조사 및 후속 조치를 감독할 책임이 있는 팀 또는 개인을 지정합니다.

책임을 전가하기보다는 학습과 개선에 초점을 맞추는 문화를 장려합니다. 개인에게 불이익을 주는 것이 아니라 향후 인시던트를 예방하는 것이 목표라는 점을 강조합니다.

인시던트 사후 분석을 수행하기 위한 잘 정의된 절차를 개발합니다. 이러한 절차에는 취해야 할 단계, 수집할 정보 및 분석 중에 해결해야 할 주요 질문이 요약되어 있어야 합니다. 즉각적인 원인을 넘어 근본 원인과 기여 요인을 식별하여 사고를 철저히 조사합니다. 다섯 가지 이유와 같은 기법을 사용하여 근본적인 문제를 심층적으로 분석합니다.

인시던트 분석을 통해 얻은 교훈의 리포지토리를 유지 관리합니다. 이러한 제도적 지식을 향후 인시던트 및 예방 노력에 참고할 수 있습니다. 인시던트 사후 분석에서 얻은 결과와 인사이트를 공유하고, 배운 교훈에 관해 논의하기 위해 누구나 참여할 수 있는 인시던트 사후 검토 모임을 개최합니다.

구현 단계

- 인시던트 사후 분석을 수행하는 과정에서 서로 비난하지 않도록 합니다. 이를 통해 인시던트에 관련된 사람들이 제안된 시정 조치에 대해 감정을 배제하고 팀 간의 솔직한 자체 평가와 협업을 촉진할 수 있습니다.
- 중요한 문제를 문서화하는 표준화된 방법을 정의합니다. 이러한 문서의 구조를 예로 들면 다음과 같습니다.
  - 어떻게 된 걸까요?
  - 문제가 고객과 비즈니스에 미친 영향
  - 근본 원인은 무엇인가요?
  - 문제 지원을 위한 데이터는 무엇인가요?
    - 예: 지표 및 그래프
  - 핵심 원칙(특히 보안)과 관련하여 어떤 의미가 있나요?
    - 워크로드를 설계할 때는 업무 상황에 따라 이러한 핵심 요소를 절충해야 합니다. 이러한 비즈니스 의사결정에 따라 엔지니어링 우선 순위가 달라질 수 있습니다. 예를 들어 개발 환경에서는 신뢰성이 다소 낮아지더라도 비용을 절감하도록 최적화할 수 있습니다. 또는 미션 크리티컬 솔루션의 경우에는 비용 증가를 감수하고 신뢰성을 기준으로 최적화할 수 있습니다. 하지만 고객 보호가 최우선이므로 모든 작업의 시작점은 항상 보안입니다.
  - 어떤 점을 배웠나요?
  - 어떤 시정 조치를 취했나요?

  - 작업 항목
  - 관련 항목
- 인시던트 사후 분석을 수행하기 위한 잘 정의된 표준 운영 절차를 만듭니다.
- 표준화된 인시던트 보고 프로세스를 설정합니다. 초기 인시던트 보고서, 로그, 통신 및 인시던트 중에 취한 조치를 포함하여 모든 인시던트를 포괄적으로 문서화합니다.
- 참고로 가동 중단이 일어나지 않더라도 인시던트일 수 있습니다. 중단이나 장애가 발생할 뻔한 상황, 시스템이 비즈니스 기능을 수행하면서도 예상치 못한 방식으로 작동하는 경우도 인시던트에 해당합니다.
- 피드백과 교훈을 바탕으로 인시던트 사후 분석 프로세스를 지속적으로 개선합니다.
- 지식 관리 시스템에서 주요 조사 결과를 캡처하고 개발자 가이드 또는 배포 전 체크리스트에 추가해야 할 패턴이 있는지 고려합니다.

## 리소스

관련 문서:

- Why you should develop a correction of error (COE)

관련 비디오:

- Amazon’s approach to failing successfully
- AWS re:Invent 2021 - Amazon Builders’ Library: Operational Excellence at Amazon

# REL12-BP03 확장성 및 성능 요구 사항 테스트

로드 테스트와 같은 기술을 사용하여 워크로드가 크기 조정 및 성능 요구 사항을 충족하는지 확인합니다.

클라우드에서는 워크로드에 대한 프로덕션 규모의 테스트 환경을 온디맨드로 생성할 수 있습니다. 프로덕션 동작의 부정확한 예측으로 이어질 수 있는 스케일 다운된 테스트 환경에 의존하는 대신 클라우드를 사용하여 예상 프로덕션 환경을 유사하게 반영하는 테스트 환경을 프로비저닝할 수 있습니다. 이 환경은 애플리케이션이 직면하는 실제 조건을 보다 정확하게 시뮬레이션하여 테스트하는 데 도움이 됩니다.

성능 테스트를 위한 노력과 더불어 기본 리소스, 크기 조정 설정, 서비스 할당량 및 복원력 설계가 로드 조건에서 예상대로 작동하는지 검증하는 것이 중요합니다. 이 전체론적 접근 방식을 통해 가장 까다로

운 조건에서도 애플리케이션이 필요에 따라 안정적으로 크기가 조정되고 작동하는지 확인할 수 있습니다.

원하는 성과: 워크로드가 피크 로드에 노출되더라도 예상 동작을 유지합니다. 애플리케이션이 성장하고 발전함에 따라 발생할 수 있는 성능 관련 문제를 사전에 해결합니다.

일반적인 안티 패턴:

- 프로덕션 환경과 유사하지 않은 테스트 환경을 사용합니다.
- 로드 테스트를 배포 지속적 통합(CI) 파이프라인의 통합된 부분이 아닌 별도의 일회성 활동으로 취급합니다.
- 응답 시간, 처리량 및 확장성 목표와 같은 명확하고 측정 가능한 성능 요구 사항을 정의하지 않습니다.
- 비현실적이거나 불충분한 로드 시나리오로 테스트를 수행하며 피크 로드, 갑작스러운 스파이크 및 지속적으로 높은 로드를 테스트하지 못합니다.
- 예상 로드 한도를 초과하여 워크로드에 스트레스 테스트를 수행하지 않습니다.
- 불충분하거나 부적절한 로드 테스트 및 성능 프로파일링 도구를 사용합니다.
- 성능 지표를 추적하고 이상을 감지할 수 있는 포괄적인 모니터링 및 알림 시스템이 부족합니다.

이 모범 사례 확립의 이점:

- 로드 테스트를 통해 시스템이 프로덕션으로 전환되기 전에 시스템의 잠재적 성능 병목 현상을 식별할 수 있습니다. 프로덕션 수준 트래픽 및 워크로드를 시뮬레이션할 때 느린 응답 시간, 리소스 제약 또는 시스템 장애와 같이 시스템이 로드를 처리하는 데 어려움을 겪을 수 있는 영역을 식별할 수 있습니다.
- 다양한 로드 조건에서 시스템을 테스트할 때 워크로드를 지원하는 데 필요한 리소스 요구 사항을 더 잘 이해할 수 있습니다. 이 정보는 리소스 할당에 대해 정보에 입각한 결정을 내리고 리소스의 과다 프로비저닝 또는 과소 프로비저닝을 방지하는 데 도움이 될 수 있습니다.
- 잠재적 장애 지점을 식별하려면 로드가 높은 조건에서 워크로드가 어떻게 작동하는지 관찰하면 됩니다. 이 정보는 적절한 경우 내결함성 메커니즘, 장애 조치 전략 및 중복 조치를 구현하여 워크로드의 신뢰성과 복원력을 개선하는 데 도움이 됩니다.
- 성능 문제를 조기에 식별하고 해결하므로 시스템 중단, 느린 응답 시간 및 사용자 불만족으로 인한 비용이 많이 드는 결과를 방지할 수 있습니다.
- 테스트 중에 수집된 자세한 성능 데이터 및 프로파일링 정보는 프로덕션에서 발생할 수 있는 성능 관련 문제를 해결하는 데 도움이 될 수 있습니다. 이로 인해 인시던트 대응 및 해결 속도가 빨라져 사용자와 조직의 운영에 미치는 영향이 줄어들 수 있습니다.

- 특정 산업에서 사전 성능 테스트를 수행하면 워크로드가 규정 준수 표준을 충족하는 데 도움이 되므로 처벌 또는 법적 문제의 위험이 줄어듭니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 지침

첫 번째 단계는 크기 조정 및 성능 요구 사항의 모든 측면을 아우르는 포괄적인 테스트 전략을 정의하는 것입니다. 시작하려면 처리량, 지연 시간 히스토그램, 오류율과 같은 비즈니스 요구 사항에 따라 워크로드의 서비스 수준 목표(SLO)를 명확하게 정의하세요. 다음으로, 평균 사용량부터 갑작스러운 스파이크 및 지속적인 피크 로드에 이르기까지 다양한 로드 시나리오를 시뮬레이션할 수 있는 테스트 세트를 설계하고 워크로드의 동작이 SLO를 충족하는지 확인합니다. 이러한 테스트는 개발 프로세스 초기에 성능 회귀를 포착하기 위해 자동화되어야 하고 지속적인 통합 및 배포 파이프라인에 통합되어야 합니다.

크기 조정 및 성능을 효과적으로 테스트하려면 적절한 도구와 인프라에 투자하세요. 여기에는 실제 사용자 트래픽을 생성할 수 있는 로드 테스트 도구, 병목 현상을 식별하는 성능 프로파일링 도구, 주요 지표를 추적하는 모니터링 솔루션이 포함됩니다. 중요한 점은 테스트 결과가 최대한 정확하도록 하려면 인프라 및 환경 조건 측면에서 테스트 환경이 프로덕션 환경과 최대한 일치하는지 확인해야 한다는 것입니다. 프로덕션과 유사한 설정을 안정적으로 복제하고 확장할 수 있도록 코드형 인프라와 컨테이너 기반 애플리케이션을 사용합니다.

크기 조정 및 성능 테스트는 일회성 활동이 아닌 지속적인 프로세스입니다. 포괄적인 모니터링 및 알림을 구현하여 애플리케이션의 프로덕션 성능을 추적하고 이 데이터를 사용하여 테스트 전략 및 최적화 노력을 지속적으로 개선합니다. 성능 데이터를 정기적으로 분석하여 새로운 문제를 식별하고, 새로운 크기 조정 전략을 테스트하고, 최적화를 구현하여 애플리케이션의 효율성과 신뢰성을 개선합니다. 반복적 접근 방식을 채택하고 프로덕션 데이터에서 지속적으로 교훈을 얻으면 애플리케이션이 다양한 사용자 요구 사항에 맞춰 조정하고 시간이 지남에 따라 복원력과 최적의 성능을 유지할 수 있는지 확인할 수 있습니다.

구현 단계

1. 응답 시간, 처리량 및 확장성 목표와 같은 명확하고 측정 가능한 성능 요구 사항을 설정합니다. 이러한 요구 사항은 워크로드의 사용 패턴, 사용자 기대치 및 비즈니스 요구 사항을 기반으로 해야 합니다.
2. 프로덕션 환경에서 로드 패턴 및 사용자 동작을 정확하게 모방할 수 있는 로드 테스트 도구를 선택하고 구성합니다.

3. 인프라 및 환경 조건을 포함하여 프로덕션 환경과 거의 일치하는 테스트 환경을 설정하여 테스트 결과의 정확도를 개선합니다.
4. 평균 사용 패턴부터 피크 로드, 빠른 스파이크, 지속적으로 높은 로드에 이르기까지 다양한 시나리오를 포함하는 테스트 세트를 생성합니다. 테스트를 지속적 통합 및 배포 파이프라인에 통합하여 개발 프로세스 초기에 성능 회귀를 파악합니다.
5. 로드 테스트를 수행하여 실제 사용자 트래픽을 시뮬레이션하고 애플리케이션이 다양한 로드 조건에서 어떻게 작동하는지 파악합니다. 애플리케이션에 스트레스 테스트를 수행하려면 예상 로드를 초과하고 응답 시간 저하, 리소스 소진 또는 시스템 장애와 같은 동작을 관찰하여 애플리케이션의 중단점을 식별하고 이를 크기 조정 전략에 반영합니다. 로드를 점진적으로 늘려 워크로드의 확장성을 평가하고 성능 영향을 측정하여 크기 조정 한도를 식별하고 향후 용량 요구 사항에 맞게 계획합니다.
6. 종합적인 모니터링 및 알림을 구현하여 성능 지표를 추적하고, 이상을 감지하고, 임계값이 초과되면 조정 작업 또는 알림을 시작합니다.
7. 성능 데이터를 지속적으로 모니터링하고 분석하여 개선이 필요한 영역을 식별합니다. 테스트 전략과 최적화 노력을 반복하여 개선합니다.

## 리소스

관련 모범 사례:

- REL01-BP04 할당량 모니터링 및 관리
- REL06-BP01 워크로드의 모든 구성 요소 모니터링(생성)
- REL06-BP03 알림 전송(실시간 처리 및 경보)

관련 문서:

- 부하 테스트 애플리케이션
- Distributed Load Testing on AWS
- 애플리케이션 성능 모니터링
- Amazon EC2 Testing Policy

관련 예제:

- Distributed Load Testing on AWS (GitHub)

관련 도구:

- Amazon CodeGuru Profiler
- Amazon CloudWatch RUM
- Apache JMeter
- K6
- Vegeta
- Hey
- ab
- wrk
- Distributed Load Testing on AWS

## REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트

시스템이 불리한 조건에서 어떻게 반응하는지 파악하려면 프로덕션 내 환경 또는 프로덕션과 최대한 가까운 환경에서 카오스 실험을 정기적으로 실행합니다.

원하는 성과:

워크로드의 복원력은 이벤트 중 워크로드의 알려진 예상 동작을 확인하는 복원력 테스트 이외에 결함 주입 실험 또는 예기치 않은 부하 발생의 형태로 카오스 엔지니어링을 적용하여 정기적으로 확인합니다. 카오스 엔지니어링과 복원력 테스트를 결합하여 구성 요소 장애 시 워크로드가 중단되지 않고, 예기치 않은 중단이 발생한 경우에도 영향을 최소화하거나 아무런 영향 없이 복구할 수 있다는 확신을 얻을 수 있습니다.

일반적인 안티 패턴:

- 복원력을 뛰어나게 설계했으나 장애 발생 시 워크로드가 전체적으로 작동하는 방식을 확인하지 않습니다.
- 실제 조건 및 예상되는 부하에서 절대 실험하지 않습니다.
- 실험을 코드로 처리하지 않고 개발 주기 전체에서 유지 관리하지 않습니다.
- 카오스 실험을 CI/CD 파이프라인의 일부로 또한 배포 범위 외부에서도 실행하지 않습니다.
- 어떤 결함을 실험할지 결정할 때 과거의 인시던트 후 분석 사용에 소홀합니다.

이 모범 사례 확립의 이점: 워크로드의 복원력을 확인하기 위해 장애를 도입하면 탄력적인 설계의 복구 절차가 실제 장애 발생 시에도 잘 작동할 것으로 확신할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 지침

카오스 엔지니어링은 서비스 공급업체, 인프라, 워크로드 및 구성 요소 수준에서 고객에게 미치는 영향을 최소화하거나 아무런 영향을 미치지 않으면서 제어되는 방식으로 실제 중단(시뮬레이션)을 지속적으로 도입할 수 있는 역량을 팀에 제공합니다. 이렇게 하면 팀이 장애로부터 배우고 워크로드의 복원력을 관찰, 측정 및 개선하고 이벤트 발생 시 알림이 전달되고 팀이 알림을 받는지 확인할 수 있습니다.

지속적으로 수행하면 카오스 엔지니어링은 해결하지 않고 두면 가용성과 운영에 부정적인 영향을 미칠 수 있는 결함을 강조해서 보여줄 수 있습니다.

> **Note**
>
> 카오스 엔지니어링은 프로덕션 환경의 급변하는 조건을 견딜 수 있는 시스템의 역량을 확신하기 위해 시스템에 대해 수행하는 실험 분야입니다. – Principles of Chaos Engineering

시스템이 중단을 견딜 수 있는 경우 카오스 엔지니어링은 자동화되어 회귀 테스트로 유지 관리해야 합니다. 이러한 방식으로 카오스 실험은 시스템 개발 수명 주기(SDLC) 및 CI/CD 파이프라인의 일부로 수행해야 합니다.

구성 요소 장애 발생 시 워크로드가 중단되지 않는지 확인하기 위해 실험의 일부로 실제 이벤트를 도입합니다. 예를 들어, Amazon EC2 인스턴스 손실 또는 기본 Amazon RDS 데이터베이스 인스턴스 장애 조치로 실험하고 워크로드가 영향을 받지 않는지(또는 최소한의 영향만 받는지) 확인합니다. 구성 요소 장애를 결합하여 가용 영역에서 중단으로 인해 발생할 수 있는 이벤트를 시뮬레이션합니다.

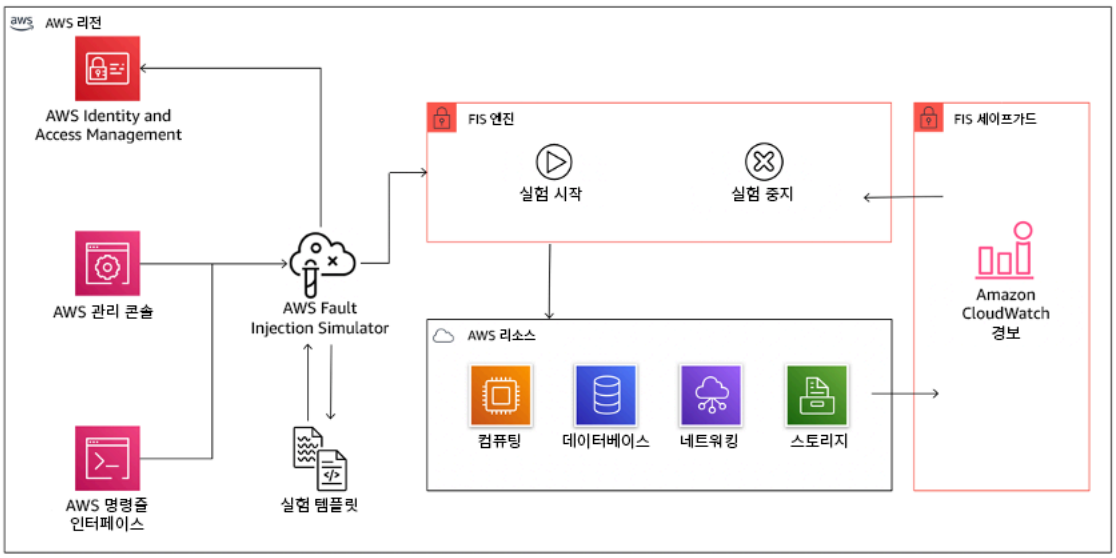
애플리케이션 수준 장애(예: 충돌)의 경우 메모리 및 CPU 소진 등과 같은 원인으로 시작할 수 있습니다.

폴백 또는 장애 조치 메커니즘을 검증하려면 구성 요소가 몇 초에서 몇 시간까지 지속될 수 있는 지정된 기간에 서드파티 제공업체에 대한 액세스를 차단하여 해당 이벤트를 시뮬레이션해야 합니다.

그 외의 모드에서 성능이 저하되면 기능이 저하되고 응답 속도가 느려져서 서비스가 일시적으로 중단되는 경우가 많습니다. 이러한 성능 저하는 대개 중요 서비스의 지연 시간 증가 및 신뢰할 수 없는 네트워크 연결(패킷이 삭제됨)로 인해 발생합니다. 지연 시간, 삭제된 메시지 및 DNS 장애 등과 같은 네트워킹 효과를 비롯한 장애를 사용한 실험에는 이름 확인 불가, DNS 서비스에 연결 불가 또는 종속적 서비스에 연결 설정 불가가 포함될 수 있습니다.

카오스 엔지니어링 도구:

AWS Fault Injection Service(AWS FIS)는 CD 파이프라인의 일부로 또는 파이프라인 외부에서 사용할 수 있는 결함 주입 실험을 실행하는 데 사용되는 완전관리형 서비스입니다. AWS FIS는 카오스 엔지니어링 게임 데이 중 사용하기 좋습니다. Amazon EC2, Amazon Elastic Container Service(Amazon ECS), Amazon Elastic Kubernetes Service(Amazon EKS), Amazon RDS를 비롯한 여러 유형의 리소스에서 동시에 결함 주입 기능을 지원합니다. 이러한 장애에는 리소스 종료, 강제 장애 조치, CPU 또는 메모리에 스트레스 유발, 제한, 지연 시간 및 패킷 손실이 포함됩니다. Amazon CloudWatch 경보와 통합되므로 테스트가 예상치 못한 영향을 미치는 경우 실험을 롤백하기 위해 중지 조건을 가드레일로 설정할 수 있습니다.




워크로드에 결함 주입 실험을 실행할 수 있도록 AWS 리소스와 통합된 AWS Fault Injection Service.

결함 주입 실험을 위한 서드파티 옵션도 몇 가지 있습니다. 여기에는 오픈 소스 도구(예: Chaos Toolkit, Chaos Mesh 및 Litmus Chaos)와 상용 옵션(예: Gremlin)이 포함됩니다. AWS에 삽입할 수 있는 결함 범위를 확장하기 위해 AWS FIS는 Chaos Mesh 및 Litmus Chaos와 통합되어 여러 도구 간에 결함 주입 워크플로를 조정할 수 있습니다. 예를 들어, 임의로 선택된 비율의 클러스터 노드를 AWS FIS 장애 작업을 사용하여 종료하는 동안 Chaos Mesh 또는 Litmus 결함을 사용하여 포드의 CPU에 대한 스트레스 테스트를 실행할 수 있습니다.

## 구현 단계

1. 실험에 사용할 결함을 결정합니다.

   워크로드 설계의 복원력을 평가합니다. Well-Architected Framework의 모범 사례를 사용하여 생성된 이러한 설계는 중요한 종속성, 과거 이벤트, 알려진 문제 및 규정 준수 요구 사항을 기반으로 위험을 고려합니다. 복원력을 유지하기 위한 설계 요소와 완화하도록 설계된 장애를 나열합니다. 이러한 목록 작성에 대한 자세한 내용은 Operational Readiness Review 백서를 참조하세요. 여기에서

는 이전 인시던트 재발을 방지하기 위한 프로세스 생성 방법을 안내합니다. 장애 모드 및 영향 분석(FMEA) 프로세스는 장애의 구성 요소 수준 및 장애가 워크로드에 미치는 영향을 분석하기 위한 프레임워크를 제공합니다. FMEA는 Adrian Cockcroft의 Failure Modes and Continuous Resilience에서 자세히 설명됩니다.

2. 각 장애에 우선순위를 할당합니다.

   처음에는 높음, 보통 또는 낮음 등과 같이 대략적인 분류로 시작합니다. 우선순위를 평가하려면 장애 빈도와 장애가 전체 워크로드에 미치는 영향을 고려해야 합니다.

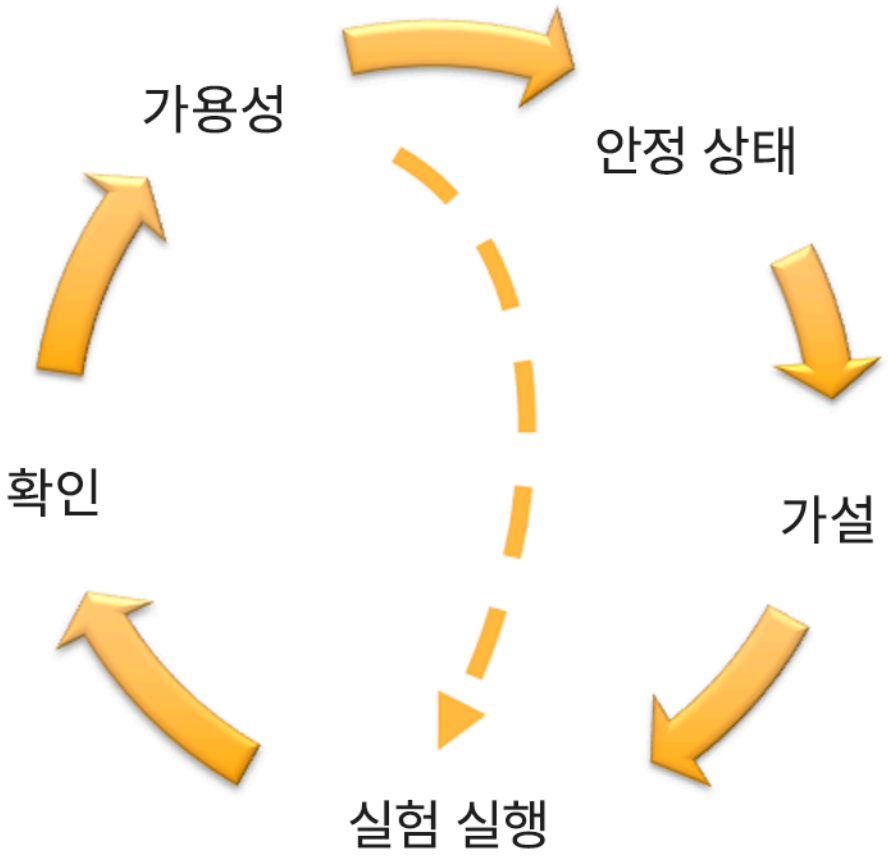
   주어진 장애의 빈도를 고려할 때 확인 가능한 경우 이 워크로드에 대한 이전 데이터를 분석합니다. 확인할 수 없는 경우에는 유사한 환경에서 실행되는 다른 워크로드의 데이터를 사용합니다.

   주어진 장애의 영향을 고려할 때 장애의 범위가 클수록 일반적으로 영향도 커집니다. 또한 워크로드 설계 및 용도도 고려합니다. 예를 들어, 소스 데이터 스토어에 액세스하는 기능은 데이터 변환 및 분석을 수행하는 워크로드에 중요합니다. 이러한 경우 액세스 장애와 제한된 액세스 및 지연 시간 삽입에 대한 실험의 우선순위를 지정할 수 있습니다.

   인시던트 후 분석은 장애 모드의 빈도 및 영향을 파악하기 위한 좋은 데이터 소스입니다.

   할당된 우선순위를 사용하여 어떤 결함을 먼저 실험할지, 새로운 결함 주입 실험을 개발할 순서를 결정합니다.

3. 수행하는 각 실험에 대해 카오스 애플리케이션 및 지속적인 복원력 플라이휠을 따릅니다(다음 그림 참조).




Adrian Hornsby의 과학적 방법을 사용하는 카오스 엔지니어링 및 연속 복원력 플라이휠.

a. 안정 상태를 정상 동작을 나타내는 워크로드의 측정 가능한 출력으로 정의합니다.

   예상한 대로 신뢰할 수 있는 상태로 작동하면 워크로드가 안정 상태를 보이는 것입니다. 따라서 안정 상태를 정의하기 전에 워크로드가 정상인지 확인합니다. 특정 비율의 장애는 허용 가능한 한도 내에서 발생할 수 있기 때문에 안정 상태라고 해서 장애 발생 시 워크로드에 미치는 영향이 반드시 없는 것은 아닙니다. 안정 상태는 실험 중 관찰하는 기준선으로, 다음 단계에서 정의하는 가설이 예상대로 나타나지 않는 경우 이상이 있음을 강조합니다.

   예를 들어, 결제 시스템의 안정 상태를 성공률 99%, 왕복 시간 500ms의 300 TPS 처리로 정의할 수 있습니다.

b. 워크로드가 장애에 반응할 방법에 대한 가설을 작성합니다.

   올바른 가설은 안정 상태 유지를 위해 장애 완화에 기대되는 작동 방식을 기반으로 합니다. 가설은 워크로드가 특정 완화를 수행하도록 설계되었기 때문에 특정 유형의 장애 발생 시 시스템 또

는 워크로드가 계속해서 안정 상태를 유지할 것이라고 가정합니다. 특정 유형의 장애 및 완화가 가설에 명시되어 있어야 합니다.

가설에는 다음 템플릿을 사용할 수 있습니다(하지만 다르게 표현할 수도 있음).

> **Note**
>
> ## ##가 발생하면 #### ## 워크로드가 ## ## ##를 기술하여 #### ## ## ## # #을 유지 관리합니다.

예시:

- Amazon EKS 노드 그룹 중 20%의 노드가 종료되면 트랜잭션 생성 API가 100ms에서 요청의 99%를 계속해서 제공합니다(안정 상태). Amazon EKS 노드는 5분 이내에 복구되고 포드는 실험 시작 후 8분 이내에 트래픽을 예약하고 처리합니다. 3분 이내에 알림이 전송됩니다.
- 단일 Amazon EC2 인스턴스 장애가 발생하면 주문 시스템의 Elastic Load Balancing 상태 확인은 Elastic Load Balancing에서 나머지 정상 인스턴스로만 요청을 보내는 반면, Amazon EC2 Auto Scaling은 장애가 발생한 인스턴스를 교체하여 서버 측(5xx) 오류의 증가율을 0.01% 미만으로 유지합니다(안정 상태).
- 기본 Amazon RDS 데이터베이스 인스턴스에서 장애가 발생하면 공급망 데이터 컬렉션 워크로드는 장애 조치를 취하고 대기 Amazon RDS 데이터베이스 인스턴스에 연결하여 데이터베이스 읽기 또는 쓰기 오류를 1분 미만으로 유지합니다(안정 상태).

c. 결함을 삽입하여 실험을 실행합니다.

실험은 기본적으로 장애 발생 시 안전해야 하며 워크로드에서 허용해야 합니다. 워크로드에서 장애가 발생할 것임을 아는 경우 실험을 실행하지 마세요. known-unknowns 또는 unknown-unknowns를 찾으려면 카오스 엔지니어링을 사용해야 합니다. known-unknowns는 알고 있지만 완전히 파악하지 못한 항목이며, unknown-unknowns는 알지 못할 뿐만 아니라 완전히 파악하지 못한 항목입니다. 실패할 것을 알고 있는 워크로드에 대한 실험에서는 새로운 인사이트를 얻을 수 없습니다. 실험은 주의해서 계획해야 하고 영향 범위가 명확해야 하며 예기치 못한 변동 발생 시 적용할 수 있는 롤백 메커니즘을 제공해야 합니다. 실사에서 워크로드가 실험을 통과해야 한다고 나타나면 실험을 계속 진행합니다. 결함 주입을 위한 옵션은 여러 가지가 있습니다. AWS의 워크로드에 대해 AWS FIS에서는 작업이라고 하는 미리 정의된 여러 장애 시뮬레이션을 제공합니다. 또한 AWS Systems Manager 문서를 사용하여 AWS FIS에서 실행하는 사용자 지정 작업을 정의할 수도 있습니다.

스크립트에 워크로드의 현재 상태를 파악하는 기능이 없고, 스크립트가 로그를 내보낼 수 없고, 가능한 경우 롤백 메커니즘 및 중지 조건을 제공할 수 없는 경우에는 카오스 실험에 사용자 지정 스크립트를 사용하지 않는 것이 좋습니다.

카오스 엔지니어링을 지원하는 효율적인 프레임워크 또는 도구 세트는 실험의 현재 상태를 추적하고, 로그를 내보내며, 실험의 제어되는 실행을 지원하기 위한 롤백 메커니즘을 제공해야 합니다. 실험 시 예기치 못한 변동이 발생하는 경우 실험을 롤백하는 안전 메커니즘과 분명하게 정의된 범위가 있는 실험을 수행하도록 허용하는 AWS FIS 등과 같은 확립된 서비스로 시작합니다. AWS FIS를 사용한 다양한 실험에 대해 알아보려면 Resilient and Well-Architected Apps with Chaos Engineering lab도 참조하세요. AWS Resilience Hub에서는 워크로드를 분석하여 AWS FIS에서 구현 및 실행하도록 선택할 수 있는 실험을 생성합니다.

> **Note**
>
> 모든 실험에 대해 범위와 그 영향을 명확하게 이해해야 합니다. 프로덕션 환경에서 실행하기 전에 비프로덕션 환경에서 먼저 장애를 시뮬레이션하는 것이 좋습니다.

실험은 가능한 경우 제어 및 실험적 시스템 배포를 둘 다 실행하는 카나리 배포를 사용하여 실제 로드를 받는 프로덕션 환경에서 실행해야 합니다. 프로덕션 환경에서 처음으로 실험하는 경우 잠재적인 영향을 완화하기 위해 사용량이 적은 시간에 실험을 실행하는 것이 좋습니다. 또한 실제 고객 트래픽 사용의 위험이 너무 큰 경우에는 프로덕션 인프라에서 제어 및 실험적 배포에 대해 가상 트래픽을 사용하여 실험을 실행할 수 있습니다. 프로덕션 환경을 사용할 수 없는 경우 가급적 프로덕션 환경과 유사한 프로덕션 전 환경에서 실험을 실행합니다.

실험이 허용 가능한 제한을 벗어나 프로덕션 트래픽 또는 다른 시스템에 영향을 미치지 않도록 가드레일을 설정하고 모니터링해야 합니다. 가드레일 지표에 대해 정의한 임곗값에 도달한 경우 실험을 중지하기 위한 중지 조건을 설정합니다. 중지 조건에는 워크로드의 안정 상태에 대한 지표와 장애를 도입하는 구성 요소에 대한 지표를 포함해야 합니다. 가상 모니터(사용자 canary라고도 함)는 일반적으로 사용자 프록시로 포함해야 하는 한 가지 지표입니다. AWS FIS의 중지 조건은 실험 템플릿의 일부로 지원되며 템플릿당 최대 5개의 중지 조건을 사용할 수 있습니다.

카오스 원칙 중 한 가지는 실험 범위 및 그 영향을 최소화하는 것입니다.

단기적인 부정적 영향 중 일부를 허용해야 하지만 실험의 여파를 최소화하고 제한하는 것은 카오스 엔지니어의 책임이자 의무입니다.

실험 범위 및 잠재적인 영향을 확인하기 위한 방법은 먼저 비프로덕션 환경에서 실험을 수행하여 실험 중 중지 조건의 임곗값이 예상대로 활성화되고 프로덕션 환경에서 직접 실험하지 않고 관찰을 통해 예외를 포착할 수 있는지 확인하는 것입니다.

결함 주입 실험을 실행할 때 책임 당사자 모두가 알림을 잘 받았는지 확인합니다. 운영 팀, 서비스 신뢰성 팀, 고객 지원 팀 등과 같은 적절한 팀과 소통하여 실험 실행 시점과 기대하는 결과에 대해 알립니다. 이러한 팀에 통신 도구를 제공하여 부정적인 영향을 확인한 경우 실험을 실행하는 팀에 알릴 수 있도록 합니다.

워크로드와 기본 시스템을 원래 정상 상태로 복원해야 합니다. 보통, 워크로드의 탄력적인 설계는 스스로 복구됩니다. 하지만 일부 장애 설계 또는 장애가 발생한 실험에서는 워크로드를 예기치 않은 장애 상태로 둘 수 있습니다. 따라서 실험을 마치면 이 점을 인식하고 워크로드 및 시스템을 복원해야 합니다. AWS FIS를 사용하면 작업 파라미터 내에서 롤백 구성을 설정할 수 있습니다(후속 작업이라고도 함). 사후 작업은 대상을 작업이 실행되기 전의 상태로 되돌립니다. 자동 작업(예: AWS FIS 사용)이든 수동 작업이든 상관 없이 후속 작업은 장애 감지 및 처리 방법을 설명하는 플레이북의 일부여야 합니다.

d. 가설을 확인합니다.

Principles of Chaos Engineering에서는 워크로드의 안정 상태를 확인하는 방법에 대한 다음 지침을 제공합니다.

시스템의 내부 특성보다는 시스템의 측정 가능한 결과에 중점을 둡니다. 단기간의 결과에 대한 측정값은 시스템의 안정 상태에 대한 프록시를 구성합니다. 전체 시스템의 처리량, 오류 발생률 및 지연 시간 백분위수는 모두 안정 상태 동작을 나타내는 관심 지표일 수 있습니다. 실험 중 체계적인 동작 패턴에 집중하여 카오스 엔지니어링은 시스템 작동 방식 확인이 아니라 시스템이 잘 작동하는지를 확인합니다.

이전 두 가지 예에서는 서버측(5xx) 오류 증가를 0.01% 미만으로, 데이터베이스 읽기 및 쓰기 오류를 1분 미만으로 지정한 안정 상태 지표를 포함합니다.

5xx 오류는 워크로드의 클라이언트가 직접 경험하는 장애 모드의 결과이기 때문에 좋은 지표입니다. 데이터베이스 오류 측정은 장애의 직접적인 결과이기 때문에 적절하긴 하지만 실패한 고객 요청 수 또는 고객에게 표시된 오류 수 등과 같은 클라이언트 영향 측정으로 보충해야 합니다. 또한 워크로드의 클라이언트가 직접 액세스할 수 있는 API 또는 URI에 종합 모니터(사용자 canary라고도 함)를 포함합니다.

e. 복원력을 위해 워크로드 설계를 개선합니다.

안정 상태가 유지되지 않는 경우 장애를 완화하기 위해 워크로드 설계를 어떻게 개선할 수 있는지 조사하여 AWS Well-Architected 신뢰성 원칙의 모범 사례를 적용합니다. 추가 지침 및 리소스는 AWS Builder's Library에서 찾을 수 있습니다. 여기에서는 특히, 상태 확인 개선 방법 또는 애플리케이션 코드에서 백오프를 사용하여 재시도 수행 방법에 대한 문서를 제공합니다.

이러한 변경 사항을 구현한 후에는 실험을 다시 실행하여(카오스 엔지니어링 프라이휠에서 점선으로 표시됨) 변경 사항의 영향을 확인합니다. 확인 단계에서 가설이 참으로 입증되면 워크로드가 안정 상태이므로 주기가 계속됩니다.

4. 실험을 정기적으로 실행합니다.

   카오스 실험은 주기이고 카오스 엔지니어링의 일부로 주기적으로 실행해야 합니다. 워크로드가 실험의 가설을 충족하면 CI/CD 파이프라인의 회귀 부분으로 계속해서 실행되도록 실험을 자동화해야 합니다. 이를 수행하는 방법을 알아보려면 how to run AWS FIS experiments using AWS CodePipeline에서 이 블로그를 참조하세요. 반복된 AWS FIS 실험(CI/CD 파이프라인 내)에 관한 이 실습에서 실제로 수행해볼 수 있습니다.

   결함 주입 실험은 게임 데이의 일부이기도 합니다(REL12-BP05 정기적으로 게임 데이 진행 참조). 게임 데이에서는 장애나 이벤트를 시뮬레이션하여 시스템, 프로세스 및 팀 대응을 확인합니다. 게임 데이의 목적은 이례적인 이벤트 발생 시 팀이 수행해야 할 작업을 실제로 수행해보는 것입니다.

5. 실험 결과를 캡처하고 저장합니다.

   결함 주입 실험의 결과는 캡처한 다음 지속해야 합니다. 나중에 실험 결과 및 추세를 분석할 수 있도록 필요한 데이터(예: 시간, 워크로드 및 조건)를 모두 포함합니다. 결과의 예에는 대시보드 스냅샷, 지표 데이터베이스의 CSV 덤프 또는 이벤트에 대해 손으로 작성한 기록, 실험에서 관찰한 내용 등이 있을 수 있습니다. AWS FIS에서 실험 로깅은 이 데이터 캡처에 포함될 수 있습니다.

## 리소스

관련 모범 사례:

- REL08-BP03 배포의 일부로 복원력 테스트 통합
- REL13-BP03 재해 복구 구현을 테스트하여 구현 확인

관련 문서:

- AWS Fault Injection Service란 무엇입니까?
- AWS Resilience Hub란 무엇입니까?

- Principles of Chaos Engineering
- Chaos Engineering: Planning your first experiment
- Resilience Engineering: Learning to Embrace Failure
- Chaos Engineering stories
- 분산 시스템의 폴백 방지
- Canary Deployment for Chaos Experiments

관련 비디오:

- AWS re:Invent 2020: Testing resiliency using chaos engineering (ARC316)
- AWS re:Invent 2019: Improving resiliency with chaos engineering (DOP309-R1)
- AWS re:Invent 2019: Performing chaos engineering in a serverless world (CMY301)

관련 도구:

- AWS Fault Injection Service
- AWS Marketplace: Gremlin Chaos Engineering Platform
- Chaos Toolkit
- Chaos Mesh
- Litmus

## REL12-BP05 정기적으로 게임 데이 진행

게임 데이를 수행하여 워크로드에 영향을 미치는 이벤트 및 장애에 대응하는 절차를 정기적으로 연습합니다. 프로덕션 시나리오를 처리할 책임이 있는 동일한 팀을 참여시킵니다. 이러한 연습은 프로덕션 이벤트로 인한 사용자 영향을 방지하기 위한 조치를 시행하는 데 도움이 됩니다. 현실적인 조건에서 대응 절차를 연습할 때 실제 이벤트가 발생하기 전에 격차나 약점을 식별하고 해결할 수 있습니다.

게임 데이는 프로덕션과 유사한 환경에서 이벤트를 시뮬레이션하여 시스템, 프로세스 및 팀 응답을 테스트합니다. 게임 데이의 목적은 이벤트가 실제로 발생할 때 팀이 수행해야 하는 것과 동일한 작업을 수행해보는 것입니다. 이 연습은 어느 분야에서 개선이 필요한지 파악하고, 조직이 이벤트 및 장애에 대처하는 경험을 쌓는 데 도움이 될 수 있습니다. 팀이 대응 방법을 습관으로 체득할 수 있도록 게임 데이를 정기적으로 진행해야 합니다.

게임 데이는 팀이 더 큰 확신을 갖고 프로덕션 이벤트를 처리할 수 있도록 준비합니다. 연습이 잘 된 팀은 다양한 시나리오를 더 빠르게 탐지하고 대응할 수 있습니다. 이로 인해 준비도와 복원력이 크게 향상됩니다.

원하는 성과: 일정에 따라 일관되게 복원력 게임 데이를 실행합니다. 이러한 게임 데이는 비즈니스 수행의 정상적이고 예상되는 부분으로 간주됩니다. 조직은 준비 문화를 구축했으며, 프로덕션 문제가 발생할 때 팀은 효과적으로 대응하고, 문제를 효율적으로 해결하고, 고객에게 미치는 영향을 완화할 준비가 잘 되어 있습니다.

일반적인 안티 패턴:

- 절차를 문서화하지만 결코 연습하지 않습니다.
- 테스트 연습에서는 비즈니스 의사 결정권자를 제외합니다.
- 게임 데이를 실행하지만 모든 관련 이해관계자에게 알리지는 않습니다.
- 기술 장애에만 집중하지만 비즈니스 이해관계자는 참여시키지 않습니다.
- 게임 데이에서 얻은 교훈을 복구 프로세스에 반영하지 않습니다.
- 장애 또는 버그에 대해 팀을 비난합니다.

이 모범 사례 확립의 이점:

- 대응 스킬 향상: 게임 데이에 팀은 시뮬레이션된 이벤트 중에 임무를 연습하고 커뮤니케이션 메커니즘을 테스트하여 프로덕션 상황에서 보다 조율되고 효율적인 대응 조치를 만듭니다.
- 종속성 식별 및 해결: 복잡한 환경에는 다양한 시스템, 서비스 및 구성 요소 간의 복잡한 종속성이 수반되는 경우가 많습니다. 게임 데이는 이러한 종속성을 식별하고 해결하는 데 도움이 될 수 있으며, 중요한 시스템과 서비스가 런북 절차에서 적절하게 다루어지고 적시에 스케일 업하거나 복구할 수 있는지 확인할 수 있습니다.
- 복원력 문화 조성: 게임 데이는 조직 내에서 복원력에 대한 사고방식을 키우는 데 도움이 될 수 있습니다. 여러 부서의 팀과 이해관계자를 참여시킬 때 이러한 연습은 조직 전체에서 복원력의 중요성에 대한 인식을 높이고 협업과 공통의 이해를 촉진합니다.
- 지속적인 개선 및 조정: 정기적인 게임 데이는 복원력 전략을 지속적으로 평가하고 조정하는 데 도움이 되므로 변화하는 상황에 대비하여 관련성과 효율성을 유지할 수 있습니다.
- 시스템에 대한 신뢰도 향상: 성공적인 게임 데이를 통해 시스템 중단 상황을 견디고 복구하는 능력에 대한 신뢰도를 높일 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

필요한 복원력 조치를 설계하고 구현한 후에는 게임 데이를 수행하여 모든 것이 프로덕션에서 계획대로 작동하는지 확인합니다. 특히 게임 데이의 첫날에는 모든 팀원이 참여해야 하며 모든 이해관계자와 참가자에게 날짜, 시간 및 시뮬레이션된 시나리오에 대해 미리 알려야 합니다.

게임 데이를 진행하는 동안 관련 팀이 규정된 절차에 따라 다양한 이벤트와 잠재적 시나리오를 시뮬레이션합니다. 참가자는 이러한 시뮬레이션된 이벤트의 영향을 면밀히 모니터링하고 평가합니다. 시스템이 설계된 대로 작동하는 경우 자동 탐지, 크기 조정 및 자체 복구 메커니즘이 활성화되어 사용자에게 거의 또는 전혀 영향을 미치지 않습니다. 팀이 부정적인 영향을 발견하면 테스트를 롤백하고 해당 런북에 문서화된 자동화된 수단 또는 수동 개입을 통해 식별된 문제를 해결합니다.

복원력을 지속적으로 개선하려면 얻은 교훈을 문서화하고 반영하는 것이 중요합니다. 이 프로세스는 게임 데이의 인사이트를 체계적으로 캡처하고 이를 사용하여 시스템, 프로세스 및 팀 기능을 개선하는 피드백 루프입니다.

시스템 구성 요소 또는 서비스가 예기치 않게 실패할 수 있는 실제 시나리오를 재현하는 데 도움이 되도록 게임 데이 연습으로 시뮬레이션된 장애를 주입합니다. 팀은 시스템의 복원력과 내결함성을 테스트하고 통제된 환경에서 인시던트 대응 및 복구 프로세스를 시뮬레이션할 수 있습니다.

AWS에서는 코드형 인프라를 사용하여 프로덕션 환경의 복제본으로 게임 데이를 수행할 수 있습니다. 이 프로세스를 통해 프로덕션 환경과 매우 유사한 안전한 환경에서 테스트할 수 있습니다. 다양한 장애 시나리오를 생성하기 위해 AWS Fault Injection Service를 고려해 보세요. Amazon CloudWatch 및 AWS X-Ray와 같은 서비스를 사용하여 게임 데이 기간 동안 시스템 동작을 모니터링합니다. AWS Systems Manager를 사용하여 플레이북을 관리 및 실행하고 AWS Step Functions을 사용하여 반복되는 게임 데이 워크플로를 오케스트레이션합니다.

구현 단계

- 게임 데이 프로그램 설정: 게임 데이의 빈도, 범위 및 목표를 정의하는 구조화된 프로그램을 개발합니다. 이러한 연습을 계획하고 실행하는 데 주요 이해관계자와 주제 전문가를 참여시킵니다.
- 게임 데이 준비:
  1. 게임 데이에서 중점을 둘 핵심 비즈니스 크리티컬 서비스를 결정합니다. 이러한 서비스를 지원하는 사람, 프로세스 및 기술을 카탈로그화하고 매핑합니다.
  2. 게임 데이의 어젠다를 설정하고 관련 팀이 이벤트에 참여하도록 준비시킵니다. 계획된 시나리오를 시뮬레이션하고 적절한 복구 프로세스를 실행하도록 자동화 서비스를 준비합니다. AWS Fault Injection Service, AWS Step Functions 및 AWS Systems Manager와 같은 AWS 서비스는 장애 주입 및 복구 작업 시작과 같은 게임 데이의 다양한 측면을 자동화하는 데 도움이 될 수 있습니다.

- 시뮬레이션 실행: 게임 데이 당일에 계획된 시나리오를 실행합니다. 사람, 프로세스 및 기술이 시뮬레이션된 이벤트에 어떻게 반응하는지 관찰하고 문서화합니다.
- 연습 후 검토 수행: 게임 데이가 끝나면 되돌아보는 세션을 갖고 얻은 교훈을 검토합니다. 개선이 필요한 영역과 운영 복원력을 개선하는 데 필요한 모든 조치를 식별합니다. 조사 결과를 문서화하고 필요한 변경 사항을 추적하여 복원력 전략과 완료 준비도를 강화합니다.

## 리소스

관련 모범 사례:

- REL12-BP01 플레이북을 사용하여 장애 조사
- REL12-BP04 카오스 엔지니어링을 이용한 복원력 테스트
- OPS04-BP01 핵심 성과 지표 파악
- OPS07-BP03 런북을 사용한 절차 수행
- OPS10-BP01 이벤트, 인시던트 및 문제 관리 프로세스 사용

관련 문서:

- AWS GameDay란?
- AWS Well-Architected Concepts - Game Day

관련 비디오:

- AWS re:Invent 2,023 - Practice like you play: How Amazon scales resilience to new heights

관련 예제:

- AWS Workshop - Navigate the storm: Unleashing controlled chaos for resilient systems
- Build Your Own Game Day to Support Operational Resilience

# 재해 복구(DR) 계획

DR 전략의 시작은 백업 및 이중화 워크로드 구성 요소를 갖추는 것입니다. RTO 및 RPO는 워크로드 복원을 위한 목표입니다. 비즈니스 요구 사항에 따라 이러한 목표를 설정합니다. 워크로드 리소스 및 데이터의 위치와 기능을 고려하여 이러한 목표를 충족하는 전략을 구현합니다. 중단 가능성과 복구 비

용도 워크로드에 대한 재해 복구 옵션을 갖추는 것이 지니는 비즈니스 가치를 파악하는 데 도움이 되는 주요 요소입니다.

가용성과 재해 복구 모두 장애 모니터링, 여러 위치에 배포, 자동 장애 조치와 같은 동일한 모범 사례를 기반으로 합니다. 그러나 가용성은 워크로드의 구성 요소에 초점을 맞추고, 재해 복구는 전체 워크로드의 개별 복사본에 초점을 맞춥니다. 재해 복구의 목표는 가용성과 달리, 재해 발생 후 복구까지의 시간에 초점을 맞춥니다.

모범 사례

- REL13-BP01 가동 중단 시간 및 데이터 손실 시의 복구 목표 정의
- REL13-BP02 복구 목표 달성을 위해 정의된 복구 전략 사용
- REL13-BP03 재해 복구 구현을 테스트하여 구현 확인
- REL13-BP04 DR 사이트 또는 리전에서 구성 드리프트 관리
- REL13-BP05 자동 복구

## REL13-BP01 가동 중단 시간 및 데이터 손실 시의 복구 목표 정의

장애가 발생하면 여러 가지 방법으로 비즈니스에 영향을 미칠 수 있습니다. 첫째, 장애는 서비스 중단(작동 중지 시간)을 유발할 수 있습니다. 둘째, 장애는 데이터 손실, 비일관성, 기한 경과를 유발할 수 있습니다. 장애에 대응하고 복구하는 방법을 안내하려면 각 워크로드에 대해 목표 복구 시간(RTO) 및 목표 복구 시점(RPO)을 정의하세요. Recovery Time Objective(RTO)는 서비스 중단과 서비스 복원 사이의 허용 가능한 최대 지연 시간입니다. 목표 복구 시점(RPO)은 마지막 데이터 복구 시점 후 허용되는 최대 시간입니다.

원하는 성과: 모든 워크로드에 기술적 고려 사항 및 비즈니스 영향을 기반으로 지정된 RTO 및 RPO가 있습니다.

일반적인 안티 패턴:

- 복구 목표를 지정하지 않았습니다.
- 임의의 복구 목표를 선택합니다.
- 너무 관대하고 비즈니스 목표를 충족하지 못하는 복구 목표를 선택합니다.
- 가동 중지 시간 및 데이터 손실의 영향을 평가하지 않았습니다.
- 워크로드 구성에서 달성할 수 없는 즉각 복구 또는 데이터 무손실과 같이 비현실적인 복구 목표를 선택합니다.

- 실제 비즈니스 목표보다 더 엄격한 복구 목표를 선택합니다. 이로 인해 워크로드에 필요한 수준 이상으로 복구 구현의 비용이 높아지고 복구 구현이 복잡해집니다.
- 종속된 워크로드의 복구 목표와 호환되지 않는 복구 목표를 선택합니다.
- 규제 및 규정 준수 요구 사항을 고려하지 않습니다.

이 모범 사례 확립의 이점: 워크로드에 대한 RTO 및 RPO를 설정할 때 비즈니스 요구 사항에 따라 명확하고 측정 가능한 복구 목표를 설정합니다. 이러한 목표를 설정한 후에는 목표에 맞게 조정된 재해 복구(DR) 계획을 수립할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

재해 복구 계획을 수립하는 데 도움이 되는 매트릭스 또는 워크시트를 구성합니다. 매트릭스에서 비즈니스 영향(예: 중요, 높음, 중간, 낮음)과 각각에 대해 목표로 삼을 관련 RTO 및 RPO를 기반으로 다양한 워크로드 범주 또는 계층을 생성합니다. 다음 매트릭스를 예시로 따라 만들 수 있습니다(RTO 값과 RPO 값이 실제와 다를 수 있음).

| 재해 복구 매트릭스 | | | | | | |
|---|---|---|---|---|---|---|
| | | 복구 시점 목표 | | | | |
| | | < 1분 | < 1시간 | < 6시간 | < 1일 | 1일 이상 |
| 복구 시간 목표 | < 10분 | 심각 | 심각 | 높음 | 보통 | 보통 |
| | < 2시간 | 심각 | 높음 | 보통 | 보통 | 낮음 |
| | < 8시간 | 높음 | 보통 | 보통 | 낮음 | 낮음 |
| | < 24시간 | 보통 | 보통 | 낮음 | 낮음 | 낮음 |
| | 24시간 이상 | 보통 | 낮음 | 낮음 | 낮음 | 낮음 |

재해 복구 매트릭스 예

각 워크로드에서 가동 중단 시간 및 데이터 손실이 비즈니스에 미치는 영향을 조사하고 이해합니다. 영향은 일반적으로 가동 중지 시간 및 데이터 손실에 따라 증가하지만, 영향의 형태는 워크로드 유형에 따라 다를 수 있습니다. 예를 들어 최대 1시간의 가동 중지는 영향이 낮을 수 있지만, 그 후에는 영향이 빠르게 심해질 수 있습니다. 영향은 재정적 영향(예: 수익 손실), 평판 영향(고객 신뢰 상실 포함), 운영 영향(예: 급여 누락 또는 생산성 감소), 규제 위험을 포함한 다양한 형태로 나타날 수 있습니다. 완료되면 워크로드를 적절한 계층에 할당합니다.

장애의 영향을 분석할 때 다음 질문을 고려하세요.

1. 허용할 수 없는 영향을 비즈니스에 미치기 전에 워크로드가 사용 불가능해도 되는 시간은 최대 어느 정도인가요?
2. 워크로드 중단으로 인해 비즈니스에 어떤 종류의 영향이 얼마나 많이 나타나나요? 재무, 평판, 운영 및 규제를 포함한 모든 종류의 영향을 고려합니다.
3. 허용할 수 없는 영향을 비즈니스에 미치기 전에 손실되어도 되거나 복구할 수 없어도 되는 데이터의 양은 최대 어느 정도인가요?
4. 손실된 데이터를 다른 소스에서 다시 생성할 수 있나요(파생 데이터라고도 함)? 그렇다면 워크로드 데이터를 다시 생성하는 데 사용되는 모든 소스 데이터의 RPO도 고려해 보세요.
5. 이 워크로드가 의존하는 워크로드(다운스트림)의 복구 목표 및 가용성 기대치는 어느 정도인가요? 다운스트림 종속성의 복구 기능을 고려할 때 워크로드의 목표를 달성할 수 있어야 합니다. 이 워크로드의 복구 기능을 개선할 수 있는 가능한 다운스트림 종속성 해결 방법 또는 완화 방법을 고려합니다.
6. 이 워크로드에 의존하는 워크로드(업스트림)의 복구 목표 및 가용성 기대치는 어느 정도인가요? 업스트림 워크로드 목표를 사용하려면 이 워크로드가 처음 보기보다 더 엄격한 복구 기능을 갖추어야 할 수 있습니다.
7. 인시던트 유형에 따라 복구 목표가 다른가요? 예를 들어 인시던트가 하나의 가용 영역에 영향을 미치는지, 아니면 전체 리전에 영향을 미치는지에 따라 RTO와 RPO가 다를 수 있습니다.
8. 복구 목표가 특정 이벤트 또는 연중 특정 시간에 변경되나요? 예를 들어 연말 쇼핑 시즌, 스포츠 이벤트, 특별 세일 및 신제품 출시 시기에는 각기 다른 RTO와 RPO가 있을 수 있습니다.
9. 사업부 및 조직의 재해 복구 전략이 있다면 복구 목표가 그러한 전략에 어떻게 부합하나요?
10. 고려해야 할 법적 또는 계약상의 영향이 있나요? 예를 들어 계약상 지정된 RTO 또는 RPO에 따라 서비스를 제공할 의무가 있나요? 이를 충족하지 못하면 어떤 처벌을 받을 수 있나요?
11. 규제 또는 규정 준수 요구 사항을 충족하기 위해 데이터 무결성을 유지해야 하나요?

다음 워크시트는 각 워크로드를 평가하는 데 도움이 될 수 있습니다. 질문을 더 추가하는 등 특정 요구 사항에 맞게 이 워크시트를 수정할 수 있습니다.

| 2단계: 기본 질문 | 워크로드에 적용 여부 | 워크로드 RTO | 워크로드 RPO | 조정된 RTO | 조정된 RPO | 지침 |
|---|---|---|---|---|---|---|
| [1] 워크로드가 다운되어도 괜찮은 최대 시간 | | | | | | 중단 시작부터 복구까지 측정된 시간 |
| [2] 손실되어도 되는 최대한의 데이터 | | | | | | 마지막으로 알려진 복원 가능한 정상 데이터 세트 이후로 측정된 시간 |
| [3a] 업스트림 종속성 | | | | | | 가장 엄격한 업스트림 복구 목표 입력 |
| [3b] 다운스트림 종속성 | | | | | | 가장 덜 엄격한 다운스트림 복구 목표 입력 |
| [3a] 조정된 업스트림 종속성 | | | | | | 업스트림 값이 현재 값보다 적고 다운스트림 값이 더 크면 종속성을 조정하고 |
| [3b] 조정된 다운스트림 종속성 | | | | | | 조정된 값을 여기에 입력 |
| [3] 종속성 | | | | | | 업스트림 종속성을 충족하도록 값을 줄이거나<br>다운스트림 종속성 기능에 따라 값 늘리기 |
| | | | | | | |
| **2단계: 추가 질문** | | | | | | 질문이 적용되면 표기. 질문이 적용되지 않으면 건너뜀 |
| 기본 RTO/RPO | | | | | | 위의 RTO 및 RPO 값을 여기로 가져오기 |
| [4] 중단의 유형 | [ ]Y / [ ]N | | | | | 요구 사항이 가장 엄격한 이벤트 유형에 대한 복구 목표 입력 |
| [5] 구체적인 시간 목표 | [ ]Y / [ ]N | | | | | 요구 사항이 가장 엄격한 시기에 대한 복구 목표 입력 |
| [6] 중단된 고객 | [ ]Y / [ ]N | | | | | 가동 중지 시간 또는 데이터 손실을 기준으로 영향을 받은 고객의 그래프 작성.<br>이 그래프를 사용하여 고객 영향에 따라 수용 가능한 최대 RTO 및 RPO 입력 |
| [7] 평판에 미치는 영향 | [ ]Y / [ ]N | | | | | 비즈니스와 조율하여 평판에 미치는 영향에 따라<br>최대 RTO 및 RPO 결정 |
| [8] 운영에 미치는 영향 | [ ]Y / [ ]N | | | | | 운영에 미치는 영향에 따라 최대 RTO 및 RPO 입력 |
| [9] 조직 차원의 조율 | [ ]Y / [ ]N | | | | | LOB 및 조직 요구 사항에 따라 이 워크로드 유형의<br>최대 RTO 및 RPO 입력 |
| [10] 계약상 의무 | [ ]Y / [ ]N | | | | | 계약상의 의무에 따라 최대 RTO 및 RPO 입력 |
| [11] 규제 준수 | [ ]Y / [ ]N | | | | | 해당하는 규제 준수 요구 사항에 따라 최대 RTO 및 RPO 입력 |
| 추가 질문에 따른 타겟 | | | | | | 4~11번 질문의 최솟값(더 엄격한 값)을 여기에 입력 |
| 조정된 타겟 | | | | | | 위의 목표를 달성할 수 없는 경우 이해 관계자들과 조율하여 제약을 완화하거나<br>새로운 최솟값을 여기에 입력 |
| | | | | | | |
| 조정된 RTO/RPO | | | | | | 기본 RPO/RTO 값 또는 조정된 타겟 중 더 낮은 값 입력 |
| | | | | | | |
| **3단계** | | | | | | |
| 사전 정의된 범주 또는 티어에 매핑 | | | | | | 정의된 티어 중 가장 근접한 티어와 일치하도록 두 값을 아래로(더 엄격하게) 조정 |

## 워크시트

## 구현 단계

1. 각 워크로드를 담당하는 비즈니스 이해관계자와 기술 팀을 식별하고 참여시킵니다.
2. 워크로드가 조직에 미치는 영향에 관한 중요도를 나타내는 범주 또는 계층을 생성합니다. 범주의 예로는 치명적, 높음, 중간, 낮음이 있습니다. 각 범주에서 비즈니스 목표와 요구 사항을 반영하는 RTO 및 RPO를 선택합니다.
3. 이전 단계에서 생성한 영향 범주 중 하나를 각 워크로드에 할당합니다. 워크로드가 범주에 매핑되는 방법을 결정하려면 비즈니스에 대한 워크로드의 중요성과 중단 또는 데이터 손실의 영향을 고려하고 위의 질문을 활용하세요. 그러면 각 워크로드에 대한 RTO 및 RPO가 도출됩니다.
4. 이전 단계에서 결정된 각 워크로드에 대한 RTO 및 RPO를 고려합니다. 워크로드의 비즈니스 및 기술 팀을 참여시켜 목표를 조정해야 하는지 결정합니다. 예를 들어 비즈니스 이해관계자는 더 엄격한 목표가 필요하다고 판단할 수 있습니다. 반면 기술 팀은 가용 리소스와 기술적 제약을 기준으로 목표를 달성할 수 있도록 수정해야 한다고 판단할 수 있습니다.

## 리소스

관련 모범 사례:

- REL09-BP04 백업 무결성 및 프로세스를 확인하기 위해 데이터의 주기적인 복구 수행
- REL12-BP01 플레이북을 사용하여 장애 조사
- REL13-BP02 복구 목표 달성을 위해 정의된 복구 전략 사용
- REL13-BP03 재해 복구 구현을 테스트하여 구현 확인

관련 문서:

- AWS Architecture Blog: Disaster Recovery Series
- AWS에서 워크로드 재해 복구: 클라우드에서의 복구(AWS 백서)
- Managing resiliency policies with AWS Resilience Hub
- APN 파트너: 재해 복구를 지원할 수 있는 파트너
- AWS Marketplace: 재해 복구에 사용할 수 있는 제품

관련 비디오:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications
- Disaster Recovery of Workloads on AWS

# REL13-BP02 복구 목표 달성을 위해 정의된 복구 전략 사용

워크로드의 복구 목표에 부합하는 재해 복구(DR) 전략을 정의합니다. 백업 및 복원, 대기(액티브/패시브), 액티브/액티브 등의 전략을 선택합니다.

원하는 성과: 각 워크로드에 대해 워크로드가 DR 목표를 달성하도록 하는 DR 전략이 정의되고 구현되어 있습니다. 워크로드 간의 DR 전략은 재사용 가능한 패턴(예: 이전에 설명한 전략)을 활용합니다.

일반적인 안티 패턴:

- DR 목표가 유사한 워크로드에 일관적이지 않은 복구 절차를 구현합니다.
- 재해가 발생했을 때 DR 전략이 임시로 구현되도록 합니다.
- 재해 복구 계획을 마련하지 않았습니다.
- 복구 시 컨트롤 플레인 작업에 의존합니다.

이 모범 사례 확립의 이점:

- 정의된 복구 전략을 사용하면 공통적인 도구 및 테스트 절차를 사용할 수 있습니다.
- 정의된 복구 전략을 사용하면 팀 간의 지식 공유와 팀이 소유한 워크로드에 대한 DR 구현이 향상됩니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음. DR 전략을 계획, 구현, 테스트하지 않으면 재해 발생 시 복구 목표를 달성하지 못할 가능성이 큽니다.

## 구현 지침

DR 전략은 기본 위치에서 워크로드를 실행할 수 없게 되었을 때 복구 사이트에서 워크로드를 실행하는 능력에 달려 있습니다. 가장 흔한 복구 목표는 REL13-BP01 가동 중단 시간 및 데이터 손실 시의 복구 목표 정의에서 논의한 RTO 및 RPO입니다.

하나의 AWS 리전 내에서 여러 가용 영역에 걸친 DR 전략은 화재, 홍수, 대규모의 정전과 같은 재해 이벤트 시 피해를 완화해 줍니다. 워크로드를 특정 AWS 리전에서 실행할 수 없게 되는 흔치 않은 이벤트에 대한 예방 조치를 구현하는 것이 요구 사항이라면 여러 리전을 사용하는 DR 전략을 사용할 수 있습니다.

여러 리전에 걸쳐 DR 전략을 설계할 때 다음 전략 중 하나를 사용해야 합니다. 전략은 복잡성과 비용이 증가하고 RTO와 RPO가 감소하는 순서로 나열되어 있습니다. 복구 리전은 워크로드에 사용되는 기본 리전이 아닌 AWS 리전을 말합니다.

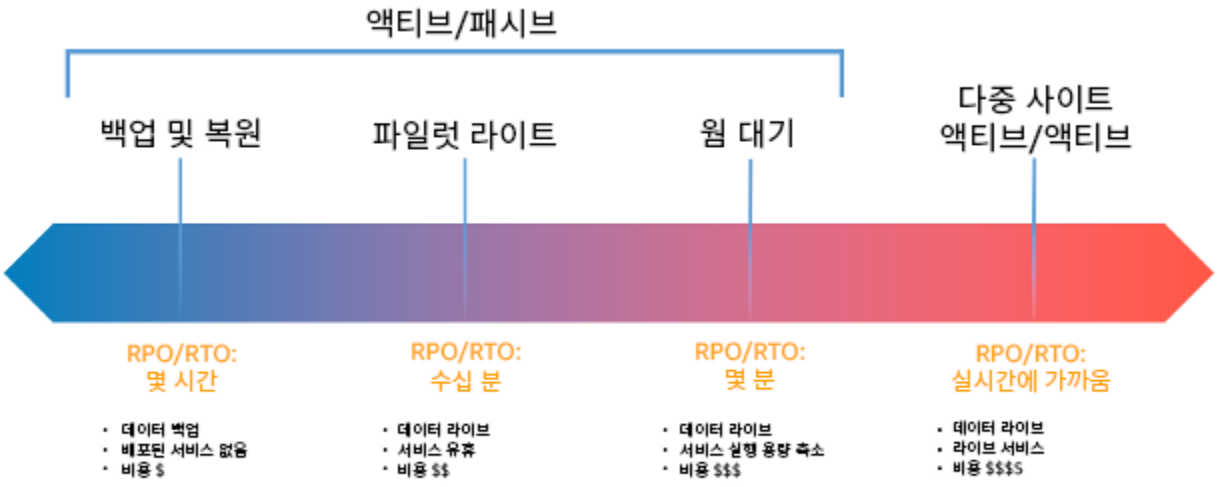



그림 17: 재해 복구(DR) 전략

- 백업 및 복원(시간 단위 RPO, 24시간 이하의 RTO): 데이터와 애플리케이션을 복구 리전에 백업합니다. 자동화된 백업 또는 지속적인 백업을 사용하면 시점 복구(PITR)가 가능하여 경우에 따라서는 RPO를 5분까지 줄일 수 있습니다. 재해 이벤트 시 인프라를 배포하고(RTO를 단축하기 위해 코드형 인프라 사용), 코드를 배포하며, 백업 데이터를 복원하여 복구 리전에서 재해로부터 복구합니다.
- 파일럿 라이트(분 단위 RPO, 10분 단위 RTO): 코어 워크로드의 인프라 복사본을 복구 리전에 프로비저닝합니다. 데이터를 복구 리전에 복제하고 복구 리전에서 백업을 생성합니다. 데이터베이스 및 객체 스토리지 등 데이터 복제 및 백업을 지원하는 데 필요한 리소스가 항상 실행됩니다. 애플리케이션 서버 또는 서버리스 컴퓨팅과 같은 기타 요소는 배포되지 않지만 필요에 따라 필수 구성 및 애플리케이션 코드로 생성될 수 있습니다.
- 웜 대기(초 단위 RPO, 분 단위 RTO): 항상 복구 리전에서 실행되는 모든 기능을 갖춘 워크로드의 스케일 다운된 버전을 유지합니다. 비즈니스 크리티컬 시스템은 완전히 복제되고 항상 실행되지만 플릿은 축소됩니다. 데이터가 복구 리전에 복제되며 실행됩니다. 복구 시기가 되면 시스템은 프로덕션 로드를 처리하기 위해 신속하게 스케일 업됩니다. 웜 대기 방식이 스케일 업될수록 RTO 및 컨트롤 플레인 의존도는 낮아집니다. 완전히 확장된 경우 상시 대기 방식이라고 합니다.
- 다중 리전(다중 사이트) 액티브/액티브(0에 가까운 RPO, 0일 수 있는 RTO): 워크로드가 여러 AWS 리전에 배포되고 능동적으로 트래픽을 처리합니다. 이 전략을 사용하려면 리전 전체에서 데이터를 동기화해야 합니다. 서로 다른 두 개 리전에 있는 복제본에 같은 레코드를 쓸 때 나타날 수 있는 충돌을 피하거나 처리해야 하는데, 그 방법이 복잡할 수 있습니다. 데이터 복제는 데이터 동기화에 유용하며 일부 유형의 재해로부터 보호해 주지만, 솔루션에 특정 시점 복구 옵션이 포함되지 않은 이상 데이터 손상 또는 중단으로부터 보호해 주지는 않습니다.

> **Note**
>
> 파일럿 라이트와 웜 대기 간의 차이를 이해하기 어려울 수 있습니다. 둘 다 복구 리전에 속한 환경과 기본 리전 자산의 복사본을 포함합니다. 차이점은 파일럿 라이트의 경우 먼저 추가 조치를 취하지 않으면 요청을 처리할 수 없지만 웜 대기는 축소된 용량 수준으로 트래픽을 즉시 처리할 수 있다는 점입니다. 파일럿 라이트를 사용하려면 서버를 켜야 하고 코어 인프라가 아닌 인프라를 추가로 배포해야 할 수 있으며 스케일 업해야 합니다. 반면 웜 대기를 사용하려면 스케일 업만 하면 됩니다. 다른 것은 이미 모두 배포되고 실행되는 상태입니다. RTO 및 RPO 요구 사항에 따라 두 전략 중에서 선택합니다.
>
> 비용이 문제이고 웜 대기 전략에 정의된 것과 유사한 RPO 및 RTO 목표를 달성하려는 경우 파일럿 라이트 접근 방식을 취하고 향상된 RPO 및 RTO 목표를 제공하는 AWS Elastic Disaster Recovery와 같은 클라우드 네이티브 솔루션을 고려할 수 있습니다

## 구현 단계

1. 이 워크로드의 복구 요구 사항을 충족하는 DR 전략을 결정합니다.

   DR 전략을 선택할 때는 가동 중단 시간과 데이터 손실을 줄이는 것(RTO 및 RPO)과 전략 구현의 비용과 복잡성을 줄이는 것 사이에서 절충해야 합니다. 필요 이상으로 엄중한 전략을 구현하지 말아야 합니다. 불필요한 비용이 발생하기 때문입니다.

   예를 들어, 다음 다이어그램에서 비즈니스는 허용 가능한 최대 RTO와 서비스 복원 전략에 지출할 수 있는 비용의 한계를 결정했습니다. 비즈니스의 목표를 감안할 때 파일럿 라이트 또는 웜 대기 DR 전략이 RTO와 비용 기준을 둘 다 만족시킵니다.

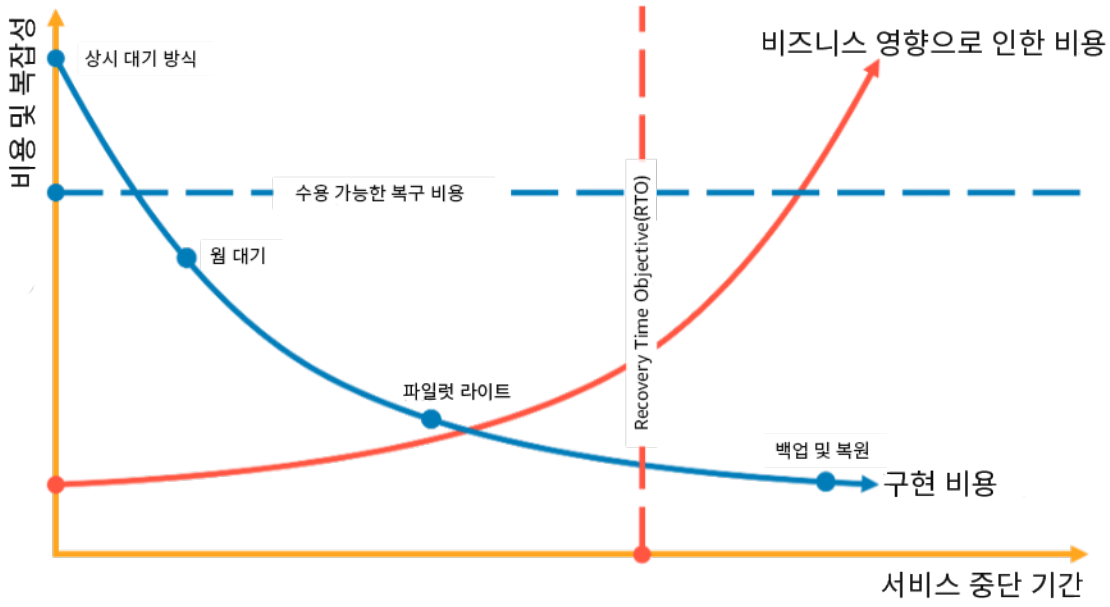



   그림 18: RTO 및 비용에 따른 DR 전략 선택

   자세한 내용은 비즈니스 연속성 계획(BCP)을 참조하세요.

2. 선택한 DR 전략을 구현할 수 있는 방법에 대한 패턴을 검토합니다.

   이 단계는 선택한 전략을 구현하는 방법을 파악하기 위한 것입니다. 전략은 AWS 리전을 기본 사이트 및 복구 사이트로 사용하여 설명되어 있습니다. 그러나 하나의 리전 내에서 DR 전략으로 가용 영역을 선택할 수도 있습니다. 그런 경우 이런 전략 중 여러 개를 활용하게 됩니다.

   다음 단계에서는 특정 워크로드에 전략을 적용할 수 있습니다.

   백업 및 복원

백업 및 복원은 구현하기 가장 복잡한 전략이지만 워크로드를 복원하는 데 더 많은 시간과 노력이 필요하므로 RTO 및 RPO도가 높아집니다. 항상 데이터의 백업을 만들어 다른 사이트(예: 다른 AWS 리전)에 복사해 두는 것이 좋습니다.

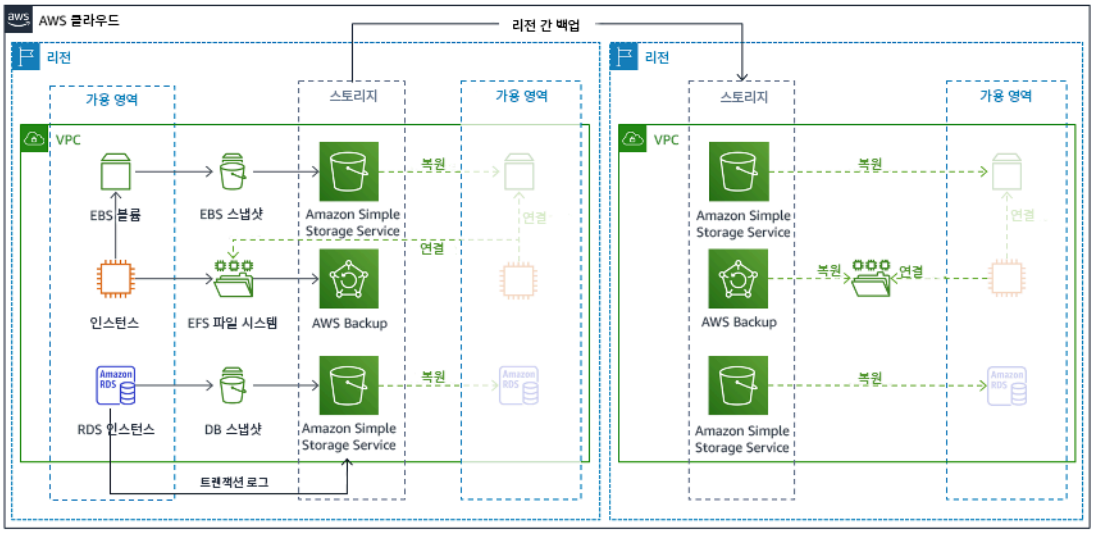



그림 19: 백업 및 복원 아키텍처

이 전략에 대한 자세한 내용은 Disaster Recovery (DR) Architecture on AWS, Part II: Backup and Restore with Rapid Recovery를 참조하세요.

파일럿 라이트

파일럿 라이트 접근 방식에서는 기본 리전에서 복구 리전으로 데이터를 복제합니다. 워크로드 인프라에 사용되는 코어 리소스가 복구 리전에 배포되지만 기능하는 스택이 되려면 기타 리소스 및 그 종속성이 여전히 필요합니다. 예를 들어 그림 20에서는 컴퓨팅 인스턴스가 배포되지 않았습니다.

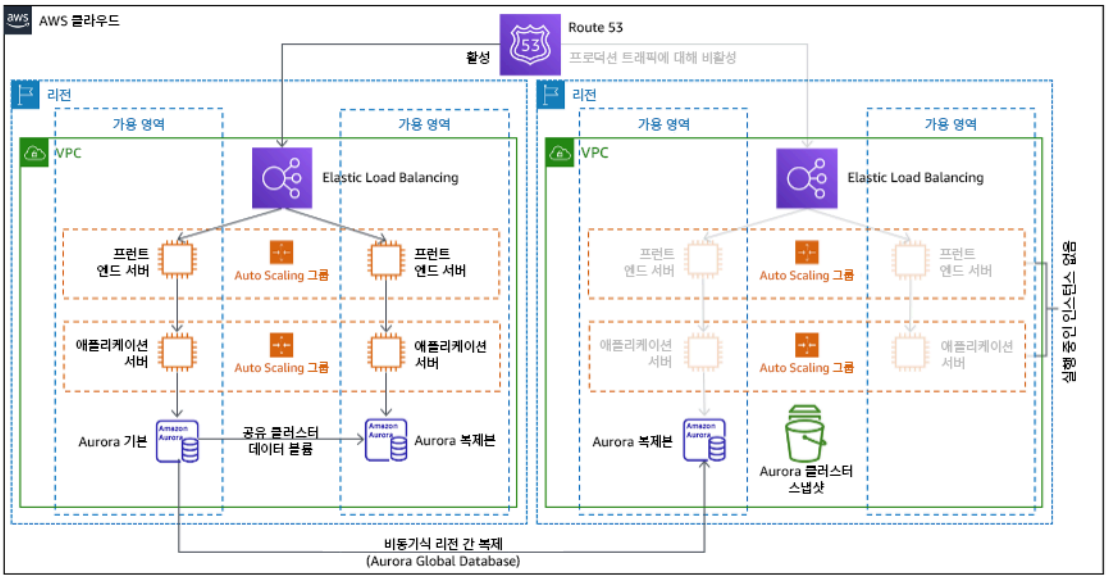



그림 20: 파일럿 라이트 아키텍처

이 전략에 대한 자세한 내용은 Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby를 참조하세요.

웜 대기

웜 대기 접근 방식에는 스케일 다운되었지만 완전히 기능하는 프로덕션 환경의 복사본이 다른 리전에 복사됩니다. 이 접근법은 파일럿 라이트의 개념을 확대하고 복구 시간을 단축합니다. 워크로드가 다른 리전에서 상시 실행되기 때문입니다. 복구 리전이 완전한 용량으로 배포되면 상시 대기 방식이라고 합니다.

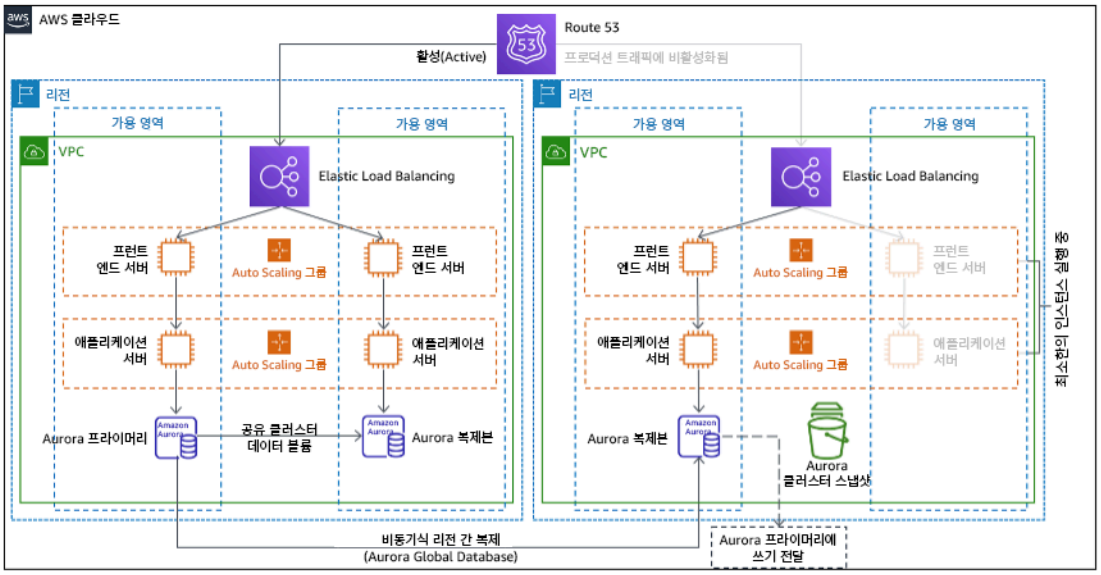



그림 21: 웜 대기 방식의 아키텍처

웜 대기 방식 또는 파일럿 라이트를 사용하려면 복구 리전에서 리소스를 스케일 업해야 합니다. 필요할 때 용량을 사용할 수 있는지 확인하려면 EC2 인스턴스에 대한 용량 예약을 사용하세요. AWS Lambda를 사용하는 경우 프로비저닝된 동시성은 함수 간접 호출에 즉시 응답할 준비가 되도록 실행 환경을 제공할 수 있습니다.

이 전략에 대한 자세한 내용은 Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby를 참조하세요.

다중 사이트 액티브/액티브

다중 사이트 액티브/액티브 전략의 일환으로 여러 리전에서 동시에 워크로드를 실행할 수 있습니다. 다중 사이트 액티브/액티브는 배포된 모든 리전에서 트래픽을 처리합니다. 고객은 DR 이외의 이유로 이 전략을 선택할 수도 있습니다. 이 전략은 가용성을 높이기 위해서 또는 글로벌 사용자에게 워크로드를 배포하는 경우(사용자에게 엔드포인트를 가까이 가져가거나 해당 리전의 사용자에게 현지화된 스택을 배포하기 위해) 사용될 수 있습니다. DR 전략으로, 워크로드가 배포된 AWS 리전 중 하나에서 지원되지 않는다면 해당 리전이 철수되며 가용성을 유지하는 데 나머지 리전이 사용됩니다. 다중 사이트 액티브/액티브는 DR 전략 중 운영 측면에서 가장 복잡하며 비즈니스 요구 사항에 따라 필요한 경우에만 선택해야 합니다.

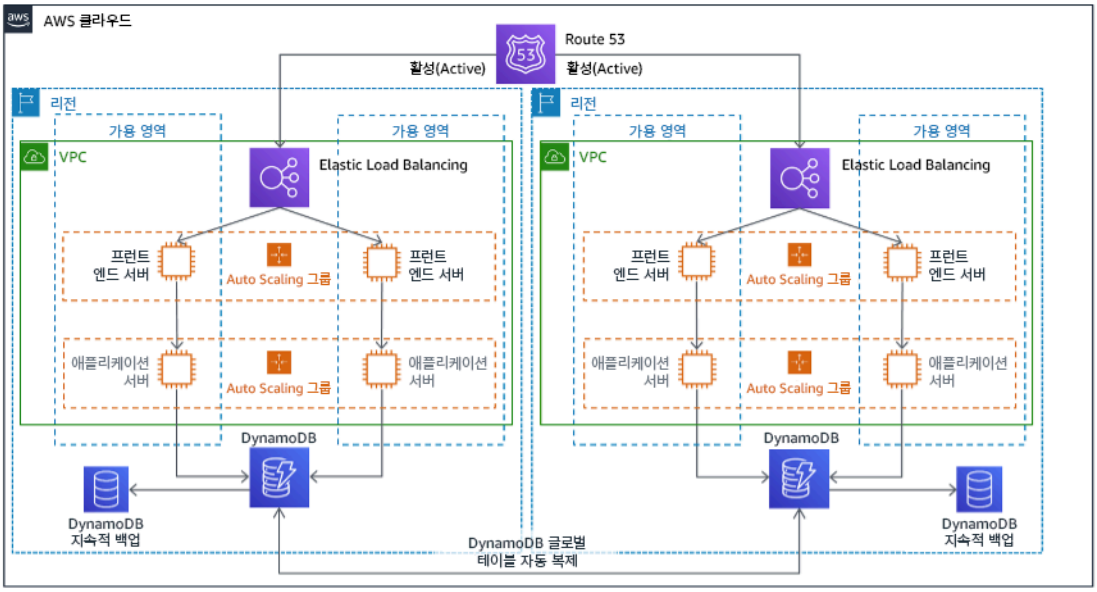



그림 22: 다중 사이트 액티브/액티브 아키텍처

이 전략에 대한 자세한 내용은 Disaster Recovery (DR) Architecture on AWS, Part IV: Multi-site Active/Active를 참조하세요.

AWS Elastic Disaster Recovery

재해 복구를 위한 파일럿 라이트 또는 웜 대기 방식 전략을 고려 중인 경우 AWS Elastic Disaster Recovery에서는 향상된 이점을 제공하는 대안 접근 방식을 제공할 수 있습니다. Elastic Disaster Recovery에서는 웜 대기 방식과 유사한 RPO 및 RTO 목표를 제공하면서도 파일럿 라이트의 비용이 저렴한 접근 방식을 유지할 수 있습니다. Elastic Disaster Recovery는 지속적인 데이터 보호를 통해 기본 리전에서 복구 리전으로 데이터를 복제하여 초 단위의 RPO와 분 단위로 측정 가능한 RTO를 달성할 수 있습니다. 데이터를 복제하는 데 필요한 리소스만 복구 영역에 배포되므로 파일럿 라이트 전략과 유사하게 비용이 절감됩니다. Elastic Disaster Recovery를 사용할 때 서비스는 장애 조치 또는 복구 드릴의 일부로 시작될 때 컴퓨팅 리소스의 복구를 조정하고 오케스트레이션합니다.

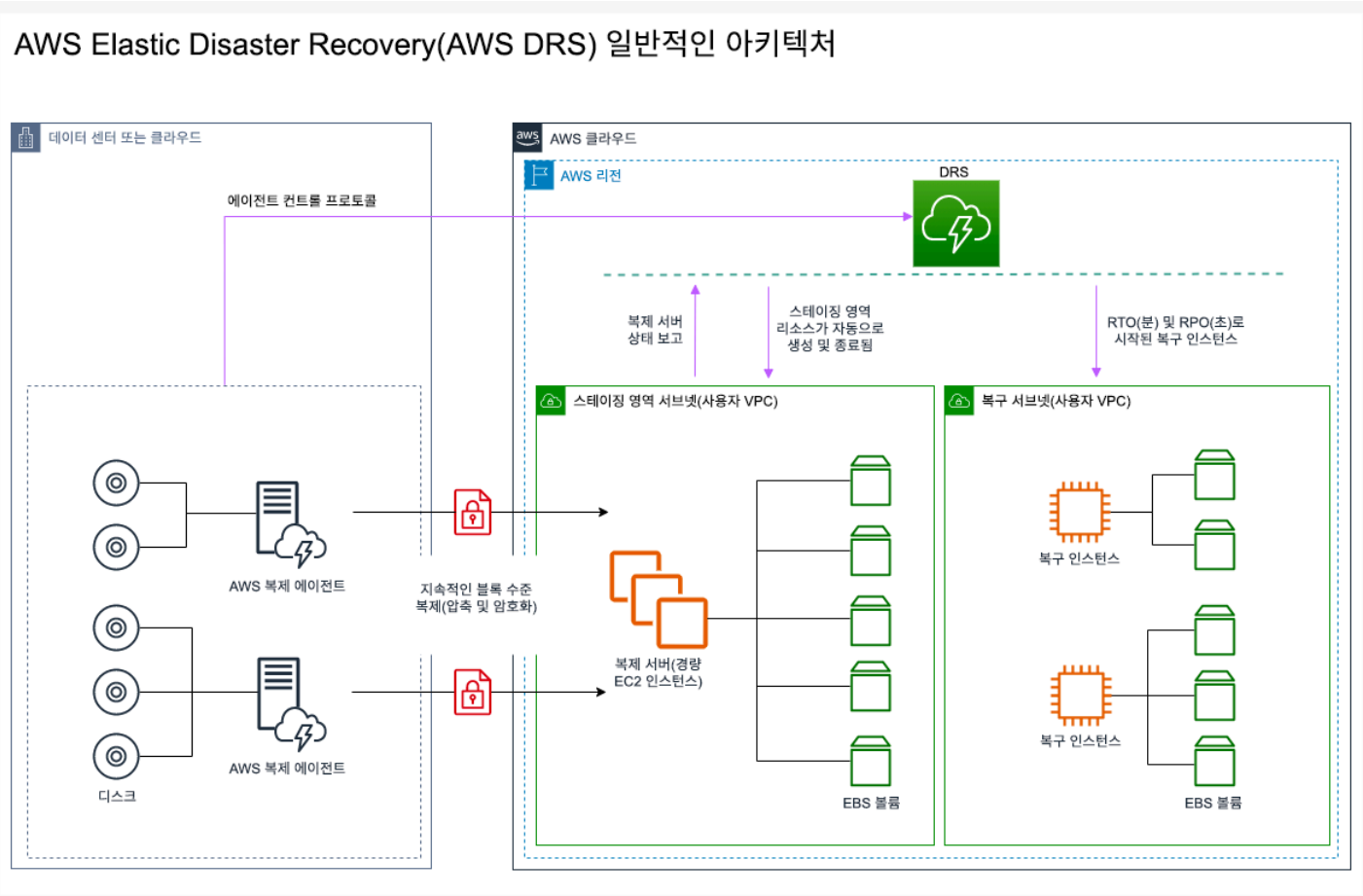



그림 23: AWS Elastic Disaster Recovery 아키텍처

데이터 보호를 위한 추가 관행

모든 전략에서 데이터 재해를 완화해야 합니다. 지속적인 데이터 복제는 일부 유형의 재해로부터 보호해 주지만, 전략에 저장된 데이터의 버전 관리 또는 특정 시점 복구 옵션이 포함되지 않은 이상 데이터 손상 또는 중단으로부터 보호해 주지는 않습니다. 복구 사이트에서 복제된 데이터를 백업하여 복제본 외에도 특정 시점 백업을 생성해야 합니다.

단일 AWS 리전에서 단일 가용 영역(AZ) 사용

하나의 리전에서 여러 개의 AZ를 사용하면 DR 구현에서 위 전략 중 여러 요소를 사용하게 됩니다. 먼저, 그림 23에서처럼 여러 개의 AZ를 사용하여 고가용성(HA) 아키텍처를 생성해야 합니다. 이 아키텍처는 Amazon EC2 인스턴스와 Elastic Load Balancer가 여러 AZ에 리소스를 배포하여 적극적으로 요청을 처리하므로 다중 사이트 액티브/액티브 접근 방식을 사용합니다. 또한 이 아키텍처는 기본 Amazon RDS 인스턴스에 장애가 발생하면(또는 AZ 자체에 장애 발생) 대기 인스턴스가 기본 인스턴스로 승격되는 상시 대기 방식도 보여줍니다.

그림 24: 다중 AZ 아키텍처

이 HA 아키텍처 외에도 워크로드를 실행하는 데 필요한 모든 데이터의 백업을 추가해야 합니다. Amazon EBS 볼륨 또는 Amazon Redshift 클러스터와 같이 단일 영역으로 제한되는 데이터에 특히 중요합니다. AZ에 장애가 발생하면 이 데이터를 다른 AZ에 복원해야 합니다. 가능하다면 추가적인 보호 조치로 데이터 백업을 다른 AWS 리전에 복사해야 합니다.

단일 리전에 대한 덜 일반적인 대안인 다중 AZ DR은 블로그 게시물, Building highly resilient applications using Amazon Application Recovery Controller, Part 1: Single-Region stack에 나와 있습니다. 여기에 나오는 전략은 리전의 작동 방식처럼 AZ 간에 가능한 한 많은 격리를 유지하기 위한 것입니다. 이 대안을 사용하면 액티브/액티브 또는 액티브/패시브 접근법 중에서 선택할 수 있습니다.

> **Note**
>
> 일부 워크로드에는 규제 데이터 상주 요구 사항이 적용됩니다. 현재 하나의 AWS 리전만 있는 근처 워크로드에 상주 요구 사항이 적용되는 경우 복수 리전이 비즈니스 요구 사항에 적합하지 않을 수 있습니다. 다중 AZ 전략은 대부분의 재해로부터 안전하게 보호해 줍니다.

3. 워크로드 리소스를 평가하고, 장애 조치(정상 운영 중) 전에 복구 리전에 존재하는 해당 리소스의 구성을 평가합니다.

   인프라 및 AWS 리소스의 경우 코드형 인프라(예: AWS CloudFormation 또는 Hashicorp Terraform과 같은 서드파티 도구)를 사용합니다. 단일 작업으로 여러 계정 및 리전에 배포하기 위해 AWS

CloudFormation StackSets를 사용할 수 있습니다. 다중 사이트 액티브/액티브 및 상시 대기 방식 전략의 경우 복구 리전에 배포된 인프라의 리소스는 기본 리전의 리소스와 같습니다. 파일럿 라이트 및 웜 대기 방식 전략의 경우 배포된 인프라가 프로덕션으로 사용되려면 추가 조치가 필요합니다. CloudFormation 파라미터 및 조건부 로직을 사용하면 단일 템플릿으로 배포된 스택이 활성 상태인지, 대기 상태인지 제어할 수 있습니다. Elastic Disaster Recovery를 사용할 때 서비스는 애플리케이션 구성 및 컴퓨팅 리소스의 복원을 복제하고 오케스트레이션합니다.

모든 DR 전략에서는 데이터 소스가 AWS 리전 내에서 백업된 다음 해당 백업이 복구 리전으로 복사되어야 합니다. AWS Backup은 이러한 리소스에 대한 백업을 구성, 예약 및 모니터링할 수 있는 중앙 집중식 보기를 제공합니다. 파일럿 라이트, 웜 대기 방식 및 다중 사이트 액티브/액티브의 경우 기본 리전의 데이터를 Amazon Relational Database Service(RDS) DB 인스턴스 또는 Amazon DynamoDB 테이블과 같은 복구 리전의 데이터 리소스로 복제해야 합니다. 그래야 이런 데이터 리소스가 복구 리전에서 실행되고 요청을 처리할 준비가 됩니다.

여러 리전에서 AWS 서비스 운영 방식에 대해 자세히 알아보려면 Creating a Multi-Region Application with AWS Services의 이 블로그 시리즈를 참조하세요.

4. 필요한 경우 재해 이벤트 중 장애 조치를 위해 복구 리전을 구성하는 방법을 결정하고 구현합니다.

   다중 사이트 액티브/액티브의 경우 장애 조치는 한 리전을 철수하고 나머지 액티브 리전에 의존하는 것입니다. 일반적으로 이러한 리전은 트래픽을 받을 준비가 되어 있습니다. 파일럿 라이트 및 웜 대기 방식 전략의 경우 복구 조치로 그림 20의 EC2 인스턴스와 같은 누락된 리소스와 기타 누락된 리소스를 배포해야 합니다.

   위에 설명된 모든 전략에서 데이터베이스의 읽기 전용 인스턴스를 기본 읽기/쓰기 인스턴스로 승격해야 합니다.

   백업 및 복원의 경우 백업에서 데이터를 복원하면 해당 데이터에 대해 EBS 볼륨, RDS DB 인스턴스, DynamoDB 테이블 등의 리소스가 생성됩니다. 또한 인프라와 배포 코드를 복원해야 합니다. AWS Backup을 사용하여 복구 리전에서 데이터를 복원할 수 있습니다. 자세한 내용은 REL09-BP01 백업해야 하는 모든 데이터 확인 및 백업 또는 소스에서 데이터 복제 섹션을 참조하세요. 인프라 재구축에는 필요한 Amazon Virtual Private Cloud(VPC), 서브넷 및 보안 그룹 외에 EC2 인스턴스와 같은 리소스 생성이 포함됩니다. 이러한 복원 작업의 대부분은 자동화할 수 있습니다. 방법을 알아보려면 이 블로그 게시물을 참조하세요.

5. 필요한 경우 재해 이벤트 중 장애 조치를 위해 트래픽을 다시 라우팅하는 방법을 결정하고 구현합니다.

   이 장애 조치 작업은 자동 또는 수동으로 시작할 수 있습니다. 상태 확인 또는 경보를 기반으로 자동으로 시작된 장애 조치는 신중하게 사용해야 합니다. 불필요한 장애 조치(거짓 경보)는 비가용성 및

데이터 손실과 같은 비용을 발생시키기 때문입니다. 따라서 수동으로 시작된 장애 조치를 자주 사용합니다. 이 경우에도 여전히 장애 조치 단계는 자동화하여 수동 시작은 버튼을 누르는 것 정도가 되도록 해야 합니다.

AWS 서비스를 사용할 때는 몇 가지 트래픽 관리 옵션이 있습니다. 한 가지 옵션은 Amazon Route 53을 사용하는 것입니다. Amazon Route 53을 사용하면 하나 이상의 AWS 리전에서 여러 개의 IP 엔드포인트를 하나의 Route 53 도메인 이름과 연결할 수 있습니다. 수동으로 시작되는 장애 조치를 구현하려면 트래픽을 복구 리전으로 다시 라우팅하는 고가용성 데이터 플레인 API를 제공하는 Amazon Application Recovery Controller를 사용할 수 있습니다. 장애 조치를 구현할 때 데이터 플레인 작업을 사용하고 컨트롤 플레인 작업을 피합니다(REL11-BP04 복구 중 컨트롤 플레인이 아닌 데이터 플레인 사용 참조).

이 옵션 및 기타 옵션에 대해 자세히 알아보려면 재해 복구 백서의 이 섹션을 참조하세요.

6. 워크로드 페일백 방식에 대한 계획을 설계합니다.

   페일백은 재해 이벤트가 수그러든 후 워크로드 작업을 기본 리전으로 돌려놓는 것입니다. 인프라 및 코드를 기본 리전에 프로비저닝하는 작업은 일반적으로 처음 사용된 것과 같은 단계를 따르며 코드형 인프라 및 코드 배포 파이프라인을 사용합니다. 페일백의 어려운 점은 데이터 스토어 복원과 작동하는 복구 리전에서 그 일관성을 확보하는 것입니다.

   장애 조치된 상태에서는 복구 리전에서 데이터베이스가 실행되고 복구 리전에 최신 데이터가 있게 됩니다. 이때 목표는 복구 리전에서 기본 리전으로 재동기화하여 최신 상태를 확보하는 것입니다.

   일부 AWS 서비스에서는 이 작업이 자동으로 이루어집니다. Amazon DynamoDB 글로벌 테이블을 사용하는 경우, 기본 리전의 테이블을 사용할 수 없게 되었더라도 다시 온라인 상태가 되면 DynamoDB는 보류 중인 모든 쓰기의 전파를 재개합니다. Amazon Aurora Global Database를 사용하고 관리형 계획된 장애 조치 기능을 사용하는 경우, Aurora 글로벌 데이터베이스의 기존 복제 토폴로지가 유지됩니다. 따라서 기본 리전에 있는 이전의 읽기/쓰기 인스턴스가 복제본이 되고 복구 리전에서 업데이트를 수신합니다.

   이 작업이 자동화되지 않는 경우 기본 리전에서 복구 리전의 데이터베이스의 복제본으로 데이터베이스를 다시 구축해야 합니다. 이때 이전의 기본 데이터베이스가 삭제되고 새로운 복제본이 생성되는 경우가 많습니다.

   장애 조치 후 복구 리전에서 계속 실행할 수 있는 경우 이 리전을 새로운 기본 리전으로 만드는 것이 좋습니다. 이전의 기본 리전을 복구 리전으로 만들려면 위의 단계를 모두 따르면 됩니다. 일부 조직에서는 예약된 교체를 수행하여 기본 및 복구 리전을 주기적으로(예: 3개월마다) 교체합니다.

장애 조치 및 장애 복구가 필요한 모든 단계는 팀의 모든 구성원이 사용할 수 있는 플레이북에 유지 관리해야 하며 주기적으로 검토해야 합니다.

Elastic Disaster Recovery를 사용할 때 서비스는 페일백 프로세스를 오케스트레이션하고 자동화하는 데 도움이 됩니다. 자세한 내용은 Performing a failback을 참조하세요.

구현 계획의 작업 수준: 높음

## 리소스

관련 모범 사례:

- the section called “REL09-BP01 백업해야 하는 모든 데이터 확인 및 백업 또는 소스에서 데이터 복제”
- the section called “REL11-BP04 복구 중 컨트롤 플레인이 아닌 데이터 플레인 사용”
- the section called “REL13-BP01 가동 중단 시간 및 데이터 손실 시의 복구 목표 정의”

관련 문서:

- AWS Architecture Blog: Disaster Recovery Series
- AWS에서 워크로드 재해 복구: 클라우드에서의 복구(AWS 백서)
- 클라우드의 재해 복구 옵션
- Build a serverless multi-region, active-active backend solution in an hour
- Multi-region serverless backend - reloaded
- RDS: 리전 간 읽기 전용 복제본 복제
- Route 53: Configuring DNS Failover
- S3: 크로스 리전 복제
- AWS Backup란 무엇입니까?
- Amazon Application Recovery Controller란 무엇입니까?
- AWS Elastic Disaster Recovery
- HashiCorp Terraform: Get Started - AWS
- APN 파트너: 재해 복구를 지원할 수 있는 파트너
- AWS Marketplace: 재해 복구에 사용할 수 있는 제품

관련 비디오:

- Disaster Recovery of Workloads on AWS
- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)
- Get Started with AWS Elastic Disaster Recovery | Amazon Web Services

## REL13-BP03 재해 복구 구현을 테스트하여 구현 확인

복구 사이트에 대한 장애 조치를 정기적으로 테스트하여 제대로 작동하고 RTO 및 RPO가 충족되는지 확인합니다.

일반적인 안티 패턴:

- 프로덕션 환경에서 장애 조치를 테스트하지 않습니다.

이 모범 사례 확립의 이점: 재해 복구 계획을 정기적으로 테스트하면 필요할 때 제대로 작동하도록 보장하고 팀이 전략 실행 방법을 숙지하고 있는지 확인할 수 있습니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

### 구현 지침

거의 사용되지 않는 복구 경로를 개발하는 것은 피해야 할 패턴입니다. 읽기 전용 쿼리에 사용되는 보조 데이터 스토어를 예로 들 수 있습니다. 데이터 스토어에 데이터를 쓸 때 기본 스토어에서 장애가 발생하면 보조 데이터 스토어로 장애 조치를 진행할 수 있습니다. 이 장애 조치를 자주 테스트하지 않으면 보조 데이터 스토어의 기능에 대한 가정이 잘못될 수 있습니다. 예를 들어 마지막으로 테스트했을 때는 보조 용량이 충분했지만 이 시나리오에서는 더 이상 로드를 모두 처리하지 못할 수도 있습니다. 경험에 따르면 자주 테스트하는 경로만이 유일하게 작동하는 오류 복구 방법입니다. 이러한 이유로 인해 복구 경로를 적게 갖는 것이 가장 좋습니다. 복구 패턴을 설정하고 정기적으로 테스트할 수 있습니다. 복잡하거나 중요한 복구 경로가 있는 경우 해당 복구 경로의 작동을 확신하기 위해 프로덕션 환경에서 해당 장애를 정기적으로 연습해야 합니다. 앞에서 설명한 예의 경우에는 필요 여부에 관계없이 대기 스토어로 정기 장애 조치를 수행해야 합니다.

구현 단계

1. 복구가 가능하도록 워크로드를 설계합니다. 복구 경로를 정기적으로 테스트합니다. 복구 지향 컴퓨팅은 복구를 향상시키는 시스템의 특성을 식별합니다. 이러한 특성으로는 격리 및 중복성, 변경 사항을 롤백하는 시스템 전체 기능, 상태 모니터링 및 결정 기능, 진단 제공 기능, 자동 복구, 모듈식 설

계 및 재시작 기능 등이 있습니다. 지정된 시간에 지정한 상태로 복구를 수행할 수 있도록 복구 경로에 대해 연습하세요. 이 복구 과정에 런북을 사용하여 문제를 문서화하고 다음 테스트 전에 해결 방법을 찾습니다.

2. Amazon EC2 기반 워크로드의 경우 AWS Elastic Disaster Recovery를 사용하여 DR 전략을 위한 드릴 인스턴스를 구현하고 시작합니다. AWS Elastic Disaster Recovery에서는 훈련을 효율적으로 실행할 수 있는 기능을 제공하여 장애 조치 이벤트를 준비하는 데 도움이 됩니다. 트래픽을 리디렉션하지 않고 테스트 및 드릴 목적으로 Elastic Disaster Recovery를 사용하여 인스턴스를 자주 시작할 수도 있습니다.

## 리소스

관련 문서:

- APN 파트너: 재해 복구를 지원할 수 있는 파트너
- AWS Architecture Blog: Disaster Recovery Series
- AWS Marketplace: 재해 복구에 사용할 수 있는 제품
- AWS Elastic Disaster Recovery
- AWS에서 워크로드 재해 복구: 클라우드에서의 복구(AWS 백서)
- AWS Elastic Disaster Recovery Preparing for Failover
- The Berkeley/Stanford recovery-oriented computing project
- What is AWS Fault Injection Simulator?

관련 비디오:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications
- AWS re:Invent 2019: Backup-and-restore and disaster-recovery solutions with AWS

# REL13-BP04 DR 사이트 또는 리전에서 구성 드리프트 관리

성공적인 재해 복구(DR) 절차를 수행하려면 DR 환경이 온라인 상태가 되면 관련 기능 또는 데이터의 손실 없이 워크로드가 적시에 정상 작업을 재개할 수 있어야 합니다. 이 목표를 달성하려면 DR 환경과 기본 환경 간에 일관된 인프라, 데이터 및 구성을 유지해야 합니다.

원하는 성과: 재해 복구 사이트의 구성 및 데이터가 기본 사이트와 동등하여 필요할 때 빠르고 완전하게 복구할 수 있습니다.

일반적인 안티 패턴:

- 기본 위치를 변경할 때 복구 위치를 업데이트하지 못하여 오래된 구성으로 인해 복구 작업이 지연됩니다.
- 기본 위치와 복구 위치 간의 서비스 차이와 같은 잠재적 제한 사항을 고려하지 않아 장애 조치 중에 예상치 못한 장애가 발생할 수 있습니다.
- 수동 프로세스를 사용하여 DR 환경을 업데이트하고 동기화하므로 인적 오류와 불일치의 위험이 증가합니다.
- 구성 드리프트를 감지하지 못하여 인시던트 발생 전에 DR 사이트 준비 상태를 잘못 인식합니다.

이 모범 사례 확립의 이점: DR 환경과 기본 환경 간의 일관성은 인시던트 후 성공적인 복구 가능성을 크게 개선하고 복구 절차 실패 위험을 줄입니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 높음

## 구현 가이드

구성 관리 및 장애 조치 준비에 대한 포괄적인 접근 방식을 통해 DR 사이트가 지속적으로 업데이트되고 기본 사이트 장애 발생 시 인계받을 준비가 되었는지 확인할 수 있습니다.

기본 환경과 재해 복구(DR) 환경 간의 일관성을 얻으려면 전송 파이프라인이 기본 사이트와 DR 사이트 모두에 애플리케이션을 배포하는지 검증합니다. 적절한 평가 기간(시차 배포라고도 함) 후 DR 사이트에 대한 변경 사항을 롤아웃하여 기본 사이트의 문제를 감지하고 문제가 퍼지기 전에 배포를 중지합니다. 모니터링을 구현하여 구성 드리프트를 감지하고 환경 전반의 변경 사항 및 규정 준수를 추적합니다. DR 사이트에서 자동 수정을 수행하여 완전한 일관성을 유지하고 인시던트 발생 시 즉시 인계할 수 있도록 합니다.

구현 단계

1. DR 리전에 DR 계획을 성공적으로 실행하는 데 필요한 AWS 서비스와 기능이 포함되어 있는지 검증합니다.
2. 코드형 인프라(IaC)를 사용합니다. 프로덕션 인프라 및 애플리케이션 구성 템플릿을 정확하게 유지하고 재해 복구 환경에 정기적으로 적용합니다. AWS CloudFormation은 CloudFormation 템플릿이 지정하는 것과 실제로 배포된 것 사이의 드리프트를 감지할 수 있습니다.
3. 기본 및 DR 사이트를 포함한 모든 환경에 애플리케이션 및 인프라 업데이트를 배포하도록 CI/CD 파이프라인을 구성합니다. AWS CodePipeline과 같은 CI/CD 솔루션은 배포 프로세스를 자동화할 수 있으므로 구성 드리프트의 위험이 줄어듭니다.

4. 기본 환경과 DR 환경의 배포에 시차를 둡니다. 이 접근 방식을 사용하면 기본 환경에서 업데이트를 처음 배포하고 테스트할 수 있습니다. 이렇게 하면 문제가 DR 사이트에 전파되기 전에 기본 사이트에 격리됩니다. 이 접근 방식은 결함이 동시에 프로덕션 및 DR 사이트로 푸시되는 것을 방지하고 DR 환경의 무결성을 유지합니다.
5. 기본 환경과 DR 환경 모두에서 리소스 구성을 지속적으로 모니터링합니다. AWS Config와 같은 솔루션은 구성 규정 준수를 적용하고 드리프트를 감지하는 데 도움이 되므로 환경 전반에서 일관된 구성을 유지하는 데 유용합니다.
6. 구성 드리프트, 데이터 복제 중단 또는 지연을 추적하고 알릴 수 있는 알림 메커니즘을 구현합니다.
7. 감지된 구성 드리프트의 수정을 자동화합니다.
8. 기본 구성과 DR 구성 간의 지속적인 일치를 확인하기 위해 정기적인 감사 및 규정 준수 검사 일정을 수립합니다. 정기 검토를 통해 정의된 규칙을 준수하고 해결해야 할 불일치를 식별할 수 있습니다.
9. AWS 프로비저닝된 용량, 서비스 할당량, 스로틀 제한, 구성 및 버전 불일치가 있는지 확인합니다.

## 리소스

관련 모범 사례:

- REL01-BP01 서비스 할당량 및 제약 조건 인식
- REL01-BP02 계정 및 리전 전체에서 서비스 할당량 관리
- REL01-BP04 할당량 모니터링 및 관리
- REL13-BP03 재해 복구 구현을 테스트하여 구현 확인

관련 문서:

- Remediating Noncompliant AWS Resources by AWS Config 규칙
- AWS Systems Manager Automation
- AWS CloudFormation: 스택 및 리소스에 대한 비관리형 구성 변경 감지
- AWS CloudFormation: 전체 CloudFormation 스택의 드리프트 감지
- AWS Systems Manager Automation
- AWS에서 워크로드 재해 복구: 클라우드에서의 복구(AWS 백서)
- AWS에서 인프라 구성 관리 솔루션을 구현하려면 어떻게 해야 하나요?
- Remediating Noncompliant AWS Resources by AWS Config 규칙

관련 비디오:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)

관련 예제:

- AWS CloudFormation 레지스트리
- Quota Monitor for AWS
- Implement automatic drift remediation for AWS CloudFormation using Amazon CloudWatch and AWS Lambda
- AWS Architecture Blog: Disaster Recovery Series
- AWS Marketplace: 재해 복구에 사용할 수 있는 제품
- 안전하고 간편한 배포 자동화

## REL13-BP05 자동 복구

장애로 인한 위험과 비즈니스 영향을 줄이기 위해 안정적이고 관찰 가능하며 재현 가능하며 테스트되고 자동화된 복구 메커니즘을 구현합니다.

원하는 성과: 복구 프로세스를 위해 잘 문서화되고 표준화되고 철저하게 테스트된 자동화 워크플로를 구현했습니다. 복구 자동화는 데이터 손실 또는 사용 불가 위험이 낮은 사소한 문제를 자동으로 수정합니다. 심각한 인시던트에 대한 복구 프로세스를 빠르게 호출하고, 운영 중에 복구 동작을 관찰하고, 위험한 상황이나 장애가 관찰되면 프로세스를 종료할 수 있습니다.

일반적인 안티 패턴:

- 복구 계획의 일환으로 실패하거나 성능이 저하된 상태에 있는 구성 요소 또는 메커니즘에 의존합니다.
- 복구 프로세스에 콘솔 액세스(클릭 작업이라고도 함)와 같은 수동 개입이 필요합니다.
- 데이터 손실 또는 사용 불가 위험이 높은 상황에서 복구 절차를 자동으로 시작합니다.
- 작동하지 않거나 추가 위험이 있는 복구 절차를 중단하는 메커니즘(예: Andon 코드 또는 큰 빨간색 중지 버튼)을 포함하지 않습니다.

이 모범 사례 확립의 이점:

- 복구 작업의 신뢰성, 예측 가능성 및 일관성이 높아집니다.
- 목표 복구 시간(RTO) 및 목표 복구 시점(RPO)을 포함하여 더 엄격한 복구 목표를 충족할 수 있습니다.
- 인시던트 발생 시 복구 실패 가능성이 줄어듭니다.
- 인적 오류가 발생하기 쉬운 수동 복구 프로세스와 관련된 장애 위험이 감소합니다.

이 모범 사례가 확립되지 않을 경우 노출되는 위험 수준: 중간

## 구현 가이드

자동 복구를 구현하려면 AWS 서비스와 모범 사례를 사용하는 포괄적인 접근 방식이 필요합니다. 시작하려면 워크로드에서 중요한 구성 요소와 잠재적 장애 지점을 식별하세요. 사람의 개입 없이 워크로드와 데이터를 장애로부터 복구할 수 있는 자동화된 프로세스를 개발합니다.

코드형 인프라(IaC) 원칙을 사용하여 복구 자동화를 개발합니다. 이렇게 하면 복구 환경이 소스 환경과 일관적이고 복구 프로세스의 버전을 관리할 수 있습니다. 복잡한 복구 워크플로를 오케스트레이션하려면 AWS Systems Manager Automations 또는 AWS Step Functions과 같은 솔루션을 고려하세요.

복구 프로세스의 자동화는 상당한 이점을 제공하며 목표 복구 시간(RTO) 및 목표 복구 시점(RPO)을 보다 쉽게 달성하는 데 도움이 될 수 있습니다. 그러나 예상치 못한 상황이 발생하여 장애가 발생하거나 추가 가동 중지 시간 및 데이터 손실과 같은 자체 위험을 초래할 수 있습니다. 이 위험을 완화하려면 진행 중인 복구 자동화를 빠르게 중단할 수 있는 기능을 제공합니다. 중단되면 조사하고 수정 조치를 취할 수 있습니다.

지원되는 워크로드의 경우 자동 장애 조치를 제공하기 위해 AWS Elastic Disaster Recovery(AWS DRS)와 같은 솔루션을 고려합니다. AWS DRS는 운영 체제, 시스템 상태 구성, 데이터베이스, 애플리케이션 및 파일을 포함한 시스템을 대상 AWS 계정 및 선호 리전의 스테이징 영역에 지속적으로 복제합니다. 인시던트가 발생하면 AWS DRS는 복제된 서버를 AWS의 복구 리전에서 완전히 프로비저닝된 워크로드로 자동 변환합니다.

자동 복구의 유지 관리 및 개선은 지속적인 프로세스입니다. 얻은 교훈을 기반으로 복구 절차를 지속적으로 테스트하고 개선하며 복구 기능을 향상할 수 있는 새로운 AWS 서비스와 기능에 대한 최신 정보를 파악하세요.

### 구현 단계

1. 자동 복구 계획
   a. 워크로드 아키텍처, 구성 요소 및 종속성을 철저히 검토하여 자동화된 복구 메커니즘을 식별하고 계획합니다. 워크로드의 종속성을 하드 종속성과 소프트 종속성으로 분류합니다. 하드 종속성은

존재하지 않으면 워크로드가 작동할 수 없고 대체할 수 없는 종속성입니다. 소프트 종속성은 워크로드가 일반적으로 사용하지만 임시 대체 시스템 또는 프로세스로 대체할 수 있거나 단계적 성능 저하로 처리할 수 있는 종속성입니다.

b. 누락되거나 손상된 데이터를 식별하고 복구하는 프로세스를 설정합니다.

c. 복구 작업이 완료된 후 복구된 정상 상태를 확인하는 단계를 정의합니다.

d. 사전 워밍 및 캐시 채우기 등 복구된 시스템을 완전한 서비스를 위해 준비하는 데 필요한 모든 작업을 고려합니다.

e. 복구 프로세스 중에 발생할 수 있는 문제와 이를 감지하고 해결하는 방법을 고려합니다.

f. 기본 사이트와 해당 컨트롤 플레인에 액세스할 수 없는 시나리오를 고려합니다. 기본 사이트에 의존하지 않고 복구 작업을 독립적으로 수행할 수 있는지 확인합니다. DNS 레코드를 수동으로 변경하지 않고도 트래픽을 리디렉션할 수 있는 Amazon Application Recovery Controller(ARC)와 같은 솔루션을 고려해 보세요.

2. 자동 복구 프로세스 개발

a. 핸즈프리 복구를 위한 자동 장애 탐지 및 장애 조치 메커니즘을 구현합니다. Amazon CloudWatch와 같은 대시보드를 구축하여 자동 복구 절차의 진행 상황과 상태를 보고합니다. 성공적인 복구를 검증하는 절차를 포함합니다. 진행 중인 복구를 중단하는 메커니즘을 제공합니다.

b. 자동으로 복구할 수 없는 장애에 대한 대체 프로세스로 플레이북을 구축하고 재해 복구 계획을 고려합니다.

c. REL13-BP03에서 설명한 대로 복구 프로세스를 테스트합니다.

3. 복구 준비

a. 복구 사이트의 상태를 평가하고 중요한 구성 요소를 미리 배포합니다. 자세한 내용은 REL13-BP04를 참조하세요.

b. 조직 전반에서 관련 이해관계자 및 팀을 관여시켜 복구 작업에 대한 명확한 역할, 책임 및 의사 결정 프로세스를 정의합니다.

c. 복구 프로세스를 시작할 조건을 정의합니다.

d. 복구 프로세스를 되돌리고 필요한 경우 또는 안전한 것으로 간주된 후 기본 사이트로 되돌릴 계획을 수립합니다.

## 리소스

관련 모범 사례:

- REL07-BP01 리소스를 확보하거나 조정할 때 자동화 사용
- REL11-BP01 워크로드의 모든 구성 요소를 모니터링하여 장애 감지

- REL13-BP02 복구 목표 달성을 위해 정의된 복구 전략 사용
- REL13-BP03 재해 복구 구현을 테스트하여 구현 확인
- REL13-BP04 사이트 또는 리전에서 구성 드리프트 관리

관련 문서:

- AWS Architecture Blog: Disaster Recovery Series
- AWS에서 워크로드 재해 복구: 클라우드에서의 복구(AWS 백서)
- Orchestrate Disaster Recovery Automation using Amazon Route 53 ARC and AWS Step Functions
- Build AWS Systems Manager Automation runbooks using AWS CDK
- AWS Marketplace: Products That Can Be Used for Disaster Recovery
- AWS Systems Manager Automation
- AWS Elastic Disaster Recovery
- Using Elastic Disaster Recovery for Failover and Failback
- AWS Elastic Disaster Recovery 리소스
- APN 파트너: 재해 복구를 지원할 수 있는 파트너

관련 비디오:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)
- AWS re:Invent 2022: AWS On Air ft. AWS Failback for AWS Elastic Disaster Recovery

# 결론

가용성과 신뢰성에 관한 주제를 처음 접하는 독자 또는 미션 크리티컬한 워크로드 가용성을 극대화하는 데 활용할 수 있는 인사이트를 찾는 독자 모두에게 이 백서가 의견을 떠올리거나, 사고 방식을 전환하거나, 새로운 의문을 제기하는 데 도움이 되었기를 바랍니다. 이어서 비즈니스 요구 사항에 적절한 수준의 가용성과 이러한 가용성을 달성하도록 안정적인 아키텍처를 설계하는 방법에 대한 이해를 높이는 계기가 되었기를 바랍니다. 이 백서에서 제공된 설계, 운영 및 복구 중심 권장 사항과 AWS 솔루션즈 아키텍트의 지식과 경험을 유용하게 활용하시기를 바랍니다. 여러분의 의견, 특히 AWS에서 가용성 수준을 높일 수 있었던 성공 사례를 알려 주십시오. 계정 팀에 문의하거나 웹 사이트에서 문의하시면 됩니다.

# 기여자

다음은 이 문서의 기여자입니다.

- Michael Haken, Amazon Web Services의 Principal Solutions Architect
- Seth Eliot, Amazon Web Services의 Principal Developer Advocate
- Mahanth Jayadeva, Amazon Web Services의 Well-Architected, Solutions Architect
- Amulya Sharma, Amazon Web Services의 Principal Solutions Architect
- Jason DiDomenico, Amazon Web Services의 Cloud Foundations, Senior Solutions Architect
- Marcin Bednarz, Amazon Web Services의 Principal Solutions Architect
- Tyler Applebaum, Amazon Web Services의 Senior Solutions Architect
- Rodney Lester, Amazon Web Services의 Principal Solutions Architect
- Joe Chapman, Amazon Web Services의 Senior Solutions Architect
- Adrian Hornsby, Amazon Web Services의 Principal System Development Engineer
- Kevin Miller, Amazon Web Services의 S3, Vice President
- Shannon Richards, Amazon Web Services의 Principal Technical Program Manager
- Laurent Domb, Amazon Web Services의 Fed Fin, Chief Technologist
- Kevin Schwarz, Amazon Web Services의 Sr. Solutions Architect
- Rob Martell, Amazon Web Services의 Principal Cloud Resilience Architect
- Priyam Reddy, Amazon Web Services의 Senior Solutions Architect Manager DR
- Jeff Ferris, Amazon Web Services의 Principal Technologist
- Matias Battaglia, Amazon Web Services의 Senior Solutions Architect

# 참조 자료

자세한 내용은 섹션을 참조하세요.

- AWS Well-Architected Framework
- AWS 아키텍처 센터

# 문서 수정

이 백서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

| 변경 사항 | 설명 | 날짜 |
| --- | --- | --- |
| 모범 사례 지침 업데이트됨 | 모범 사례는 REL 1, REL 2, REL 4, REL 6, REL 7, REL 8, REL 10, REL 12, REL 13 영역에서 새로운 지침으로 업데이트되었습니다. 원칙 전반에서 지침이 확장되고 명확해졌습니다. REL10-BP02 및 REL12-BP03의 지침은 다른 모범 사례에 통합되었습니다. 원칙 전체의 리소스가 업데이트되었습니다. | 2024년 11월 6일 |
| 모범 사례 지침 업데이트됨 | REL 2, 4, 5, 6, 7, 8의 모범 사례에서 소규모 업데이트. | 2024년 6월 27일 |
| 모범 사례 지침 업데이트됨 | 장애 방지를 위해 분산 시스템에서 상호 작용 설계, 장애 완화 또는 극복을 위해 분산 시스템에서 상호 작용 설계, 워크로드 리소스 모니터링, 수요 변경에 따라 조정되는 워크로드 설계, 변경 구현, 신뢰성 테스트와 같은 영역에서 새로운 지침으로 모범 사례가 업데이트되었습니다. | 2023년 12월 6일 |
| 모범 사례 지침 업데이트됨 | 워크로드 리소스 모니터링 및 구성 요소 장애를 견딜 수 있는 워크로드 설계와 같은 영역에서 새로운 지침으로 모범 사례가 업데이트되었습니다. | 2023년 10월 3일 |

| | | |
|---|---|---|
| 모범 사례 지침 업데이트됨 | 워크로드 서비스 아키텍처 설계, 장애 완화 또는 극복을 위해 분산 시스템에서 상호 작용 설계, 워크로드 리소스 모니터링과 같은 영역에서 새로운 지침으로 모범 사례가 업데이트되었습니다. | 2023년 7월 13일 |
| 마이너 업데이트 | 포용적이지 않은 표현을 삭제했습니다. | 2023년 4월 13일 |
| 새 프레임워크 관련 업데이트 | 권장 가이드 및 새로운 모범 사례를 추가하여 모범 사례를 업데이트했습니다. | 2023년 4월 10일 |
| 백서 업데이트 | 새로운 구현 가이드와 함께 모범 사례를 업데이트했습니다. | 2022년 12월 15일 |
| 마이너 업데이트 | 그림 번호를 수정하고 전체적으로 사소한 변경을 진행했습니다. | 2022년 11월 17일 |
| 백서 업데이트 | 모범 사례를 확대하고 개선 계획을 추가했습니다. | 2022년 10월 20일 |
| 백서 업데이트 | 신뢰성 원칙의 장애 격리를 통한 워크로드 보호 및 구성 요소 장애를 견디도록 워크로드 설계 섹션에 두 가지 새로운 모범 사례가 추가되었습니다. | 2022년 5월 5일 |

| | | |
|---|---|---|
| 백서 업데이트 | Route 53 Application Recovery Controller를 포함하도록 재해 복구 가이드를 업데이트했습니다. DevOps Guru에 대한 참조를 추가했습니다. 몇 가지 리소스 링크가 업데이트되고 사소한 편집 변경 사항이 있었습니다. | 2021년 10월 26일 |
| 마이너 업데이트 | AWS Fault Injection Service(AWS FIS)에 대한 정보를 추가했습니다. | 2021년 3월 15일 |
| 마이너 업데이트 | 사소한 텍스트 업데이트가 적용되었습니다. | 2021년 1월 4일 |
| 백서 업데이트 | 부록 A에서 Amazon SQS, Amazon SNS, Amazon MQ의 가용성 설계 목표를 업데이트했습니다. 쉽게 찾아볼 수 있도록 테이블의 행을 재배치했습니다. 가용성과 재해 복구의 차이점과 그 둘이 복원력에 어떻게 기여하는지에 대한 설명을 개선했습니다. 가용성을 위한 다중 리전 아키텍처 및 재해 복구를 위한 다중 리전 전략에 대한 설명을 확장했습니다. 참조 도서를 최신 버전으로 업데이트했습니다. 요청 기반 계산과 단축키 계산을 포함하도록 가용성 계산을 확대했습니다. 게임 데이에 대한 설명을 개선했습니다. | 2020년 12월 7일 |

| | | |
|---|---|---|
| 마이너 업데이트 | 부록 A에서 AWS Lambda의 가용성 설계 목표를 업데이트했습니다. | 2020년 10월 27일 |
| 마이너 업데이트 | 부록 A에 AWS Global Accelerator의 가용성 설계 목표를 추가했습니다. | 2020년 7월 24일 |
| 새 프레임워크 관련 업데이트 | 중요한 업데이트가 적용되었고, 콘텐츠가 새로 추가되거나 수정되었습니다('워크로드 아키텍처' 모범 사례 섹션 추가, 변경 관리 및 장애 관리 섹션에서 모범 사례 재편, 리소스 업데이트, 최신 AWS 리소스 및 서비스(예: AWS Global Accelerator, AWSService Quotas 및 AWS Transit Gateway)를 포함하도록 업데이트, 신뢰성, 가용성, 복원력에 대한 정의 추가 또는 업데이트, Well-Architected 검토에 사용되는 AWS Well-Architected Tool에 맞게 백서의 질문 및 모범 사례 조정, 설계 원칙 재정렬(복구 절차 테스트 이전으로 장애 자동 복구 이전), 수식의 형식 및 다이어그램 업데이트, 주요 서비스 섹션을 제거하고 대신 주요 AWS 서비스에 대한 참조를 모범 사례에 통합). | 2020년 7월 8일 |
| 마이너 업데이트 | 연결되지 않는 링크 수정 | 2019년 10월 1일 |
| 백서 업데이트 | 부록 A 업데이트됨 | 2019년 4월 1일 |

| | | |
|---|---|---|
| 백서 업데이트 | 특정 AWS Direct Connect 네트워킹 권장 사항 및 추가 서비스 설계 목표 추가됨 | 2018년 9월 1일 |
| 백서 업데이트 | 설계 원칙 및 제한 관리 섹션을 추가했습니다. 링크가 업데이트되었으며, 업스트림/다운스트림 용어에 대한 모호성이 제거되었으며, 가용성 시나리오의 나머지 신뢰성 원칙 주제에 대한 명시적 참조가 추가되었습니다. | 2018년 6월 1일 |
| 백서 업데이트 | DynamoDB 크로스 리전 솔루션을 DynamoDB 글로벌 테이블로 변경했습니다. 서비스 설계 목표 추가됨 | 2018년 3월 1일 |
| 마이너 업데이트 | 애플리케이션 가용성을 포함하도록 가용성 계산 일부 수정 | 2017년 12월 1일 |
| 백서 업데이트 | 개념, 모범 사례 및 구현 예제를 비롯하여 고가용성 설계에 대한 지침을 제공하도록 업데이트되었습니다. | 2017년 11월 1일 |
| 최초 게시 | 신뢰성 원칙 - AWS Well-Architected Framework를 게시했습니다. | 2016년 11월 1일 |

# 고지 사항

고객은 본 문서의 정보를 독립적으로 평가할 책임이 있습니다. 본 문서는 (a) 정보 제공의 목적으로만 제공되고, (b) 사전 통지 없이 변경될 수 있는 현재 AWS 제품 및 관행을 나타내고, (c) AWS 및 그 계열사, 공급업체 또는 라이선스 제공자로부터 어떠한 약속이나 보증도 하지 않습니다. AWS 제품 또는 서비스는 명시적이든 묵시적이든 어떠한 종류의 보증, 진술 또는 조건 없이 '있는 그대로' 제공됩니다. 고객에 대한 AWS의 책임 및 채무는 AWS 계약에 준거합니다. 본 문서는 AWS와 고객 간의 어떠한 계약도 구성하지 않으며 이를 변경하지도 않습니다.

© 2023 Amazon Web Services, Inc. 또는 계열사. All rights reserved.

# AWS 용어집

최신 AWS 용어는 AWS 용어집 참조서의 AWS 용어집을 참조하십시오.