회색 장애 감지 및 완화

고급 다중 AZ 복원 패턴

고급 다중 AZ 복원 패턴: 회색 장애 감지 및 완화

Table of Contents

고급 다중 AZ 복원 패턴

출판 날짜: 2023년 7월 11일(문서 수정)

많은 고객이 가용성이 높은 다중 가용 영역(AZ) 구성에서 워크로드를 실행합니다. 이러한 아키텍처는 바이너리 장애 발생 시에도 잘 작동하지만 회색 장애 문제가 발생하는 경우가 많습니다. 이러한 유형의 장애는 미묘할 수 있으며 빠르고 확실하게 감지하기 어려울 수 있습니다. 이 백서는 워크로드를 계측하여 단일 가용 영역에 격리된 회색 장애로 인한 영향을 감지한 다음 가용 영역에서 그러한 영향을 완화하기 위한 조치를 취하는 방법에 대한 지침을 제공합니다.

소개

이 문서의 목적은 복원력이 뛰어난 다중 AZ 아키텍처를 보다 효과적으로 구현하도록 돕는 것입니다. Amazon Virtual Private Cloud(VPC) 네트워크에서 복원력이 뛰어난 시스템을 구축하는 모범 사례 중 하나는 각 워크로드를 여러 가용 영역에 배포하는 것입니다.

가용 영역은 중복 전원, 네트워킹 및 연결이 있는 하나 이상의 개별 데이터 센터입니다. 여러 가용 영역을 사용하면 단일 데이터 센터로 가능한 것보다 더 높은 가용성, 내결함성 및 확장성을 갖춘 워크로드를 운영할 수 있습니다.

Amazon Elastic Compute Cloud(EC2), Auto Scaling 또는 Amazon Relational Database Service(RDS)와 같은 많은 AWS 서비스가 다중 AZ 구성을 제공합니다. 이러한 서비스를 사용하면 관찰성 또는 장애 조치 도구를 추가로 구축할 필요가 없습니다. 단일 가용 영역에 영향을 미치는 AWS 리전 내에서 쉽게 감지할 수 있는 바이너리 장애 모드에서도 워크로드를 복원할 수 있습니다. 이는 완전한 물리적 하드웨어 장애, 전력 손실 또는 대다수 리소스에 영향을 미치는 잠재적 소프트웨어 버그일 수 있습니다.

그러나 회색 장애라고 하는 또 다른 범주의 장애도 있는데, 이러한 장애 유형은 미묘하고 빠르고 확실하게 감지하기 어렵습니다. 결과적으로 고장으로 인한 영향을 완화하는 데 시간이 더 오래 걸립니다. 이 백서에서는 회색 장애가 다중 AZ 아키텍처에 미칠 수 있는 영향, 이를 감지하는 방법, 마지막으로 장애를 완화하는 방법을 중점적으로 다룹니다.

> 이 백서에 제공된 지침은 대부분 다음과 같은 특정 워크로드 클래스에 적용됩니다.
>
> - 주로 영역 AWS 서비스를 사용
> - 단일 지역 복원력 개선 필요
> - 필요한 관찰성 및 복원력 패턴을 구축하기 위해 상당한 투자를 할 의향이 있음

이러한 워크로드는 ???에서 제시한 장단점 중 일부 또는 전부를 고려하지 않을 수도 있고 여러 지역을 사용할 수 있는 옵션이 없을 수도 있습니다. 이러한 유형의 워크로드는 전체 포트폴리오의 일부에 불과할 가능성이 높으므로 플랫폼 수준과 워크로드 수준에서 이 지침을 고려해야 합니다.

회색 장애

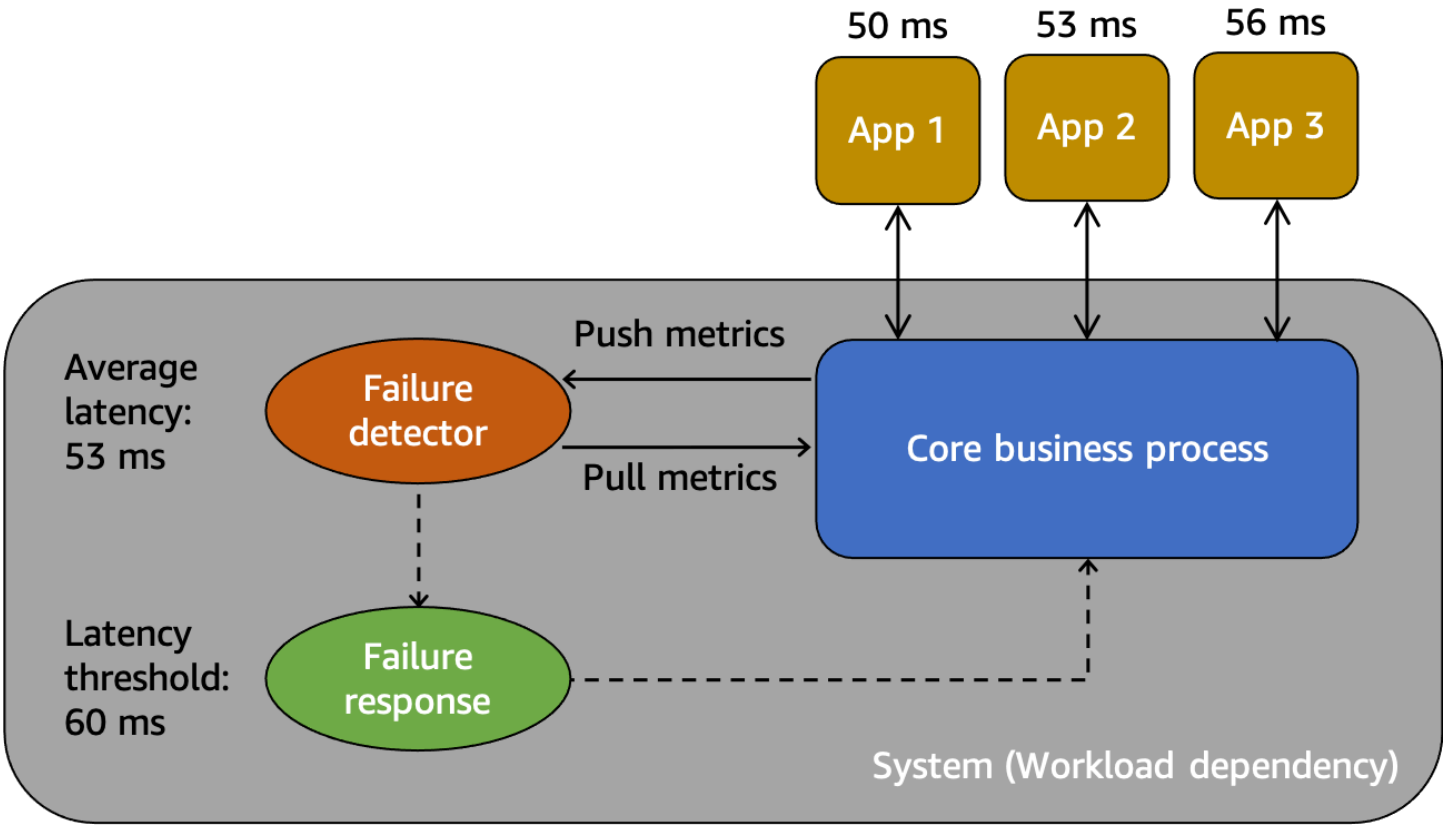
회색 장애는 차등 관찰성의 특성에 의해 정의됩니다. 즉, 각 개체가 장애를 다르게 관찰한다는 의미입니다. 이것이 무엇을 의미하는지 정의해 보겠습니다.

차등 관찰성

운영하는 워크로드에 일반적으로 종속성이 있습니다. 예를 들어 워크로드를 구축하는 데 사용하는 AWS 클라우드 서비스나 페더레이션에 사용하는 서드 파티 ID 제공업체(idP)일 수 있습니다. 이러한 종속성은 거의 항상 자체 관찰성을 구현하여 고객 사용에 따라 생성되는 오류, 가용성, 지연 시간 등에 대한 메트릭을 기록합니다. 이러한 지표 중 하나에 대한 임계값이 초과되면 종속성은 일반적으로 이를 수정하기 위해 몇 가지 조치를 취합니다.

이러한 종속성에는 일반적으로 여러 서비스 소비자가 있습니다. 또한 소비자는 자체 관찰성을 구현하고 종속성과의 상호 작용에 대한 메트릭과 로그를 기록하여 디스크 읽기 지연 시간, 실패한 API 요청 수, 데이터베이스 쿼리에 걸린 시간 등을 기록합니다.

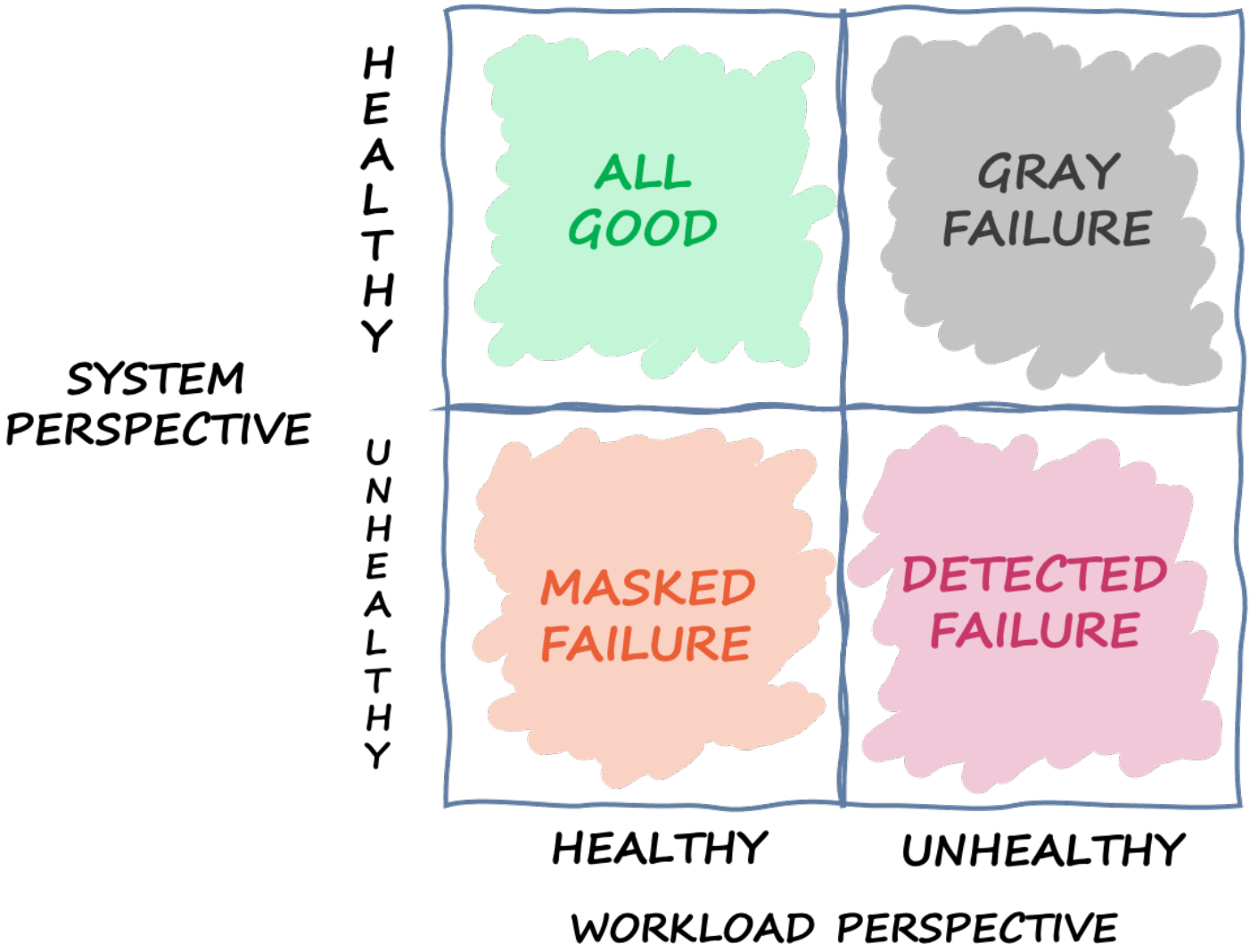
다음 그림에서는 이러한 상호 작용과 측정치를 추상 모델로 보여줍니다.



회색 장애를 이해하기 위한 추상 모델

먼저 시스템이 있는데, 이 시나리오에서는 소비자가 앱 1, 앱 2, 앱 3에 의존하는 시스템입니다. 시스템에는 핵심 비즈니스 프로세스에서 생성된 메트릭을 검사하는 장애 탐지기가 있습니다. 또한 장애 탐지기에서 관찰되는 문제를 완화하거나 수정할 수 있는 장애 대응 메커니즘도 있습니다. 시스템은 전체 평균 지연 시간을 53밀리초로 보고 있으며, 평균 지연 시간이 60ms를 초과할 경우 장애 대응 메커니즘을 간접적으로 호출하도록 임계값을 설정했습니다. 앱 1, 앱 2, 앱 3도 시스템과의 상호 작용을 자체적으로 관찰하여 각각 50ms, 53ms, 56ms의 평균 지연 시간을 기록하고 있습니다.

차등 관찰성은 시스템 소비자 중 한 명이 시스템 비정상 상태를 감지했지만 시스템 자체 모니터링으로 문제를 감지하지 못하거나 영향이 경보 임계값을 넘지 않는 경우입니다. 앱 1에서 평균 지연 시간이 50ms가 아닌 70ms로 시작한다고 가정해 보겠습니다. 앱 2와 앱 3의 평균 지연 시간은 변하지 않습니다. 이렇게 하면 기본 시스템의 평균 지연 시간이 59.66ms로 늘어나지만 지연 임계값을 넘지 않아 장애 대응 메커니즘이 활성화되지 않습니다. 하지만 앱 1의 지연 시간은 40% 증가합니다. 이는 앱 1에 대해 구성된 클라이언트 제한 시간을 초과하여 가용성에 영향을 미치거나, 더 긴 상호 작용 체인에 연쇄적인 영향을 미칠 수 있습니다. 앱 1의 관점에서 보면 이 시스템이 의존하는 기본 시스템은 비정상이지만 시스템 자체의 관점에서 보면 앱 2와 앱 3은 시스템은 정상입니다. 다음 그림은 이러한 다양한 관점을 요약한 것입니다.



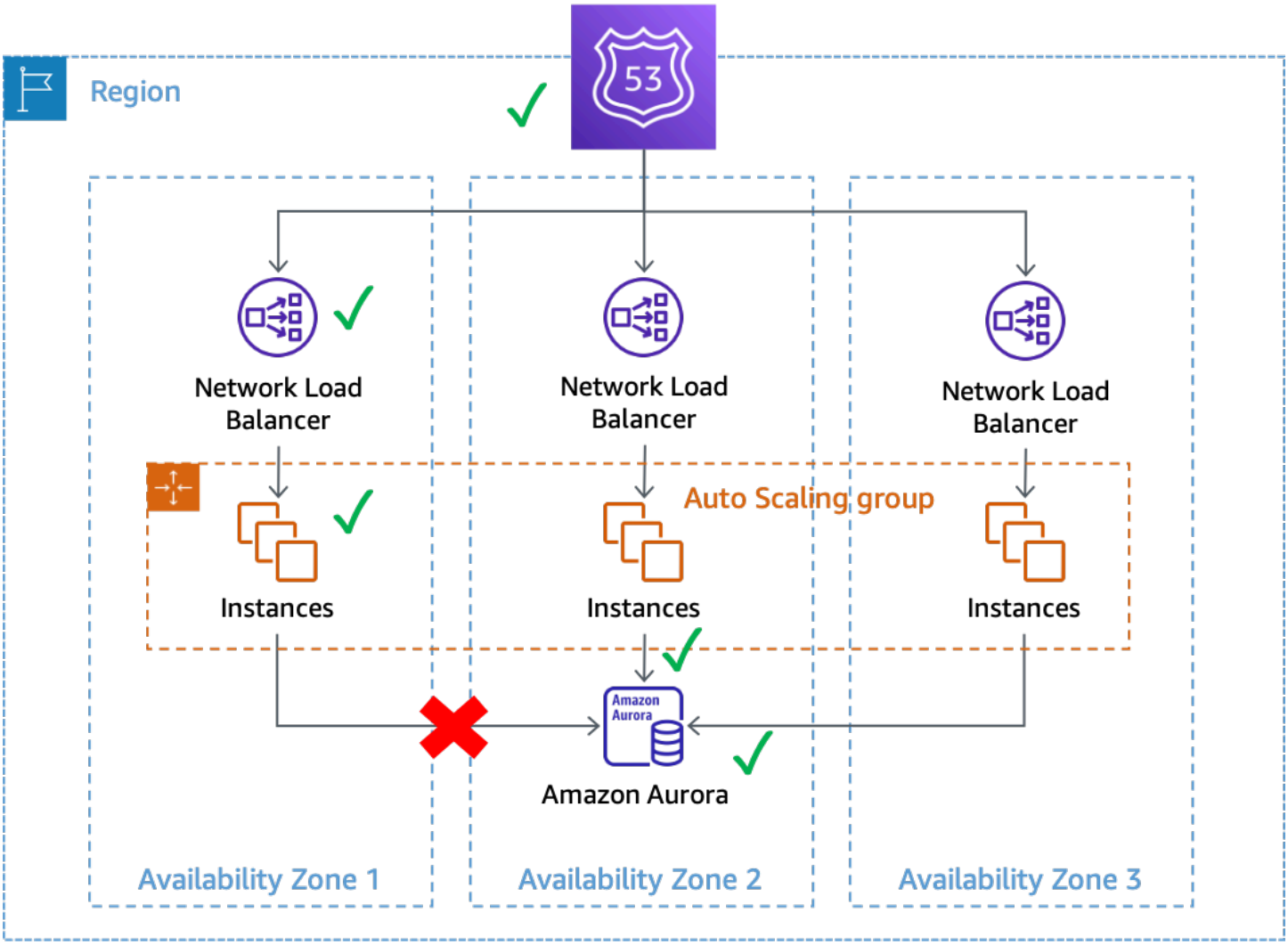
다양한 관점을 기반으로 시스템이 처할 수 있는 여러 상태를 정의하는 사분점

장애는 이 사분면을 넘어갈 수도 있습니다. 이벤트는 회색 장애로 시작했다가 감지된 오류가 되고, 마스킹된 장애로 이동했다가 다시 회색 장애로 돌아갈 수 있습니다. 주기가 정의되어 있지 않으며 근본 원인이 해결될 때까지 장애가 재발할 가능성은 거의 항상 존재합니다.

이를 통해 도출한 결론은 워크로드가 장애를 감지하고 완화하기 위해 항상 기본 시스템에 의존할 수는 없다는 것입니다. 기본 시스템이 아무리 정교하고 복원력이 뛰어나더라도 장애가 감지되지 않거나 반응 임계값 미만으로 유지될 가능성은 언제나 있습니다. 앱 1과 같은 해당 시스템을 사용하는 사용자는 회색 장애로 인한 영향을 신속하게 감지하고 완화할 수 있는 장비를 갖추어야 합니다. 이를 위해서는 이러한 상황에 대한 관찰성 및 복구 메커니즘을 구축해야 합니다.

회색 장애 예제

회색 장애는 AWS의 다중 AZ 시스템에 영향을 미칠 수 있습니다. 세 개의 가용 영역에 배포된 오토 스케일링의 Amazon EC2 인스턴스 플릿을 예로 들어 보겠습니다. 모두 하나의 가용 영역에 있는 Amazon Aurora 데이터베이스에 연결됩니다. 그러면 가용 영역 1과 가용 영역 2 간의 네트워킹에 영향을 미치는 회색 장애가 발생합니다. 이러한 손상의 결과로 가용 영역 1에 있는 인스턴스의 신규 및 기존 데이터베이스 연결 중 일정 비율에 장애가 발생합니다. 이 상황은 다음 그림에 나와있습니다.



가용 영역 1에 있는 인스턴스의 데이터베이스 연결에 영향을 미치는 회색 장애

이 예제에서 Amazon EC2는 가용 영역 1의 인스턴스가 시스템 및 인스턴스 상태 확인을 계속 통과하기 때문에 정상으로 간주합니다. 또한 Amazon EC2 Auto Scaling은 가용 영역에 대한 직접적인 영향을 감지하지 못하고 구성된 가용 영역에서 계속 용량을 시작합니다. 또한 Network Load Balancer(NLB)는 NLB 엔드포인트에 대해 수행되는 Route 53 상태 확인과 마찬가지로 배후의 인스턴스를 정상으로 간주합니다. 마찬가지로 Amazon Relational Database Service(RDS)는 데이터베이스 클러스터를 정상

상태로 간주하고 자동 장애 조치를 트리거하지 않습니다. 서비스와 리소스가 정상인 것으로 간주하는 다양한 서비스가 많지만, 워크로드는 가용성에 영향을 미치는 장애를 감지합니다. 이는 회색 장애입니다.

회색 장애 대응

AWS 환경에서 회색 장애가 발생하는 경우 일반적으로 다음과 같은 세 가지 옵션을 사용할 수 있습니다.

- 아무 것도 하지 말고 장애가 끝날 때까지 기다리십시오.
- 장애가 단일 가용 영역에만 국한된 경우 해당 가용 영역을 비우십시오.
- 다른 AWS 리전으로 장애 조치를 하고 AWS 리전 격리의 이점을 활용하여 영향을 완화하십시오.

대부분의 워크로드에 대해 옵션 1을 사용해도 괜찮다는 AWS 고객이 많습니다. 그들은 추가적인 관찰성 혹은 탄력성 솔루션을 구축할 필요가 없다는 점을 감안하여 확장된 Recovery Time Objective(RTO)를 수용합니다. 다양한 장애 모드에 대한 완화 계획으로 세 번째 옵션인 Multi-Region 재해 복구(DR)를 구현하는 고객도 있습니다. 다중 리전 아키텍처는 이러한 시나리오에서 잘 작동할 수 있습니다. 그러나 이 접근 방식을 사용할 때는 몇 가지 장단점이 있습니다(다중 리전 고려 사항에 대한 전체 논의는 AWS Multi-Region Fundamentals 참조).

먼저, 다중 리전 아키텍처를 구축하고 운영하는 작업은 까다롭고 복잡하며 잠재적으로 비용이 많이 들 수 있습니다. 다중 리전 아키텍처에서는 어떤 DR 전략을 선택할지 신중하게 고려해야 합니다. 백업 및 복원 전략이 복원력 요구 사항을 충족하지 못할 수도 있지만 영역 장애만 처리하기 위해 다중 리전 액티브-액티브 DR 솔루션을 구현하는 것은 재정적으로 실행 가능하지 않을 수 있습니다. 또한 필요할 때 제대로 작동할 수 있도록 프로덕션 환경에서 다중 리전 장애 조치를 지속적으로 실행해야 합니다. 이 모든 작업에서는 구축, 운영 및 테스트에 많은 전용 시간과 리소스가 필요합니다.

둘째, 현재 AWS 리전 전반에 걸쳐 사용 중인 AWS 서비스의 데이터 복제는 모두 비동기적으로 이루어집니다. 비동기 복제는 데이터가 손실될 수 있습니다. 즉, 리전 장애 조치 중에 어느 정도의 데이터 손실과 불일치가 발생할 수 있습니다. 데이터 손실량에 대한 허용 한도는 Recovery Point Objective(RPO)로 정의됩니다. 강력한 데이터 일관성이 필요한 고객은 기본 리전을 다시 사용할 수 있게 되면 조정 시스템을 구축하여 이러한 일관성 문제를 해결해야 합니다. 또는 자체 동기식 복제 또는 이중 쓰기 시스템을 구축해야 하므로 응답 지연 시간, 비용 및 복잡성에 상당한 영향을 미칠 수 있습니다. 또한 모든 트랜잭션에 대해 보조 리전을 엄격하게 종속시켜 전체 시스템의 가용성을 잠재적으로 떨어뜨릴 수 있습니다.

마지막으로, 액티브/스탠바이 방식을 사용하는 많은 워크로드의 경우 다른 리전으로 장애 조치를 수행하는 데 걸리는 시간은 0이 아닙니다. 기본 리전의 워크로드 포트폴리오를 특정 순서로 중단하거나, 연

결을 끊거나, 특정 프로세스를 중지해야 할 수 있습니다. 그런 다음 서비스를 특정 순서로 다시 가져와야 할 수도 있습니다. 새 리소스를 프로비저닝해야 하거나 서비스를 시작하기 전에 필요한 상태 확인을 통과하는 데 시간이 필요할 수도 있습니다. 이러한 장애 조치 프로세스는 완전히 사용할 수 없는 기간으로 발생할 수 있습니다. 이것이 바로 RTO의 관심사입니다.

리전 내의 많은 AWS 서비스가 매우 일관된 데이터 지속성을 제공합니다. Amazon RDS 다중 AZ 배포는 동기 복제를 사용합니다. Amazon Simple Storage Service(S3)는 강력한 쓰기 후 읽기 일관성을 제공합니다. Amazon Elastic Block Storage(Amazon EBS)는 여러 볼륨의 중단 일관성이 보장되는 스냅샷을 제공합니다. Amazon DynamoDB는 강력히 일관된 읽기를 수행할 수 있습니다. 이러한 특성을 사용하면 다중 리전 아키텍처보다 단일 리전에서 더 낮은 RPO(대부분의 경우 제로 RPO)를 달성할 수 있습니다.

인프라와 리소스가 이미 가용 영역 전체에 프로비저닝되어 있기 때문에 가용 영역 제거는 다중 리전 전략보다 RTO가 낮을 수 있습니다. 다중 AZ 아키텍처는 서비스 중단 및 백업을 신중하게 주문하거나 연결을 드레이닝할 필요 없이 가용 영역이 손상되더라도 정적인 방식으로 계속 운영될 수 있습니다. 리전별 장애 조치 중에 일정 기간 동안 완전히 사용할 수 없는 상태가 발생하는 대신 가용 영역 대피 중 작업이 나머지 가용 영역으로 이동되므로 대부분의 시스템에서 약간의 성능 저하만 발생할 수 있습니다. 시스템에서 가용 영역 장애 발생 시 정적으로 안정되도록 설계된 경우(이 경우 부하를 흡수하기 위해 다른 가용 영역에 용량을 미리 프로비저닝해야 함) 워크로드 고객은 전혀 영향을 받지 않을 수 있습니다.

> 단일 가용 영역의 손상은 워크로드 외에도 하나 이상의 AWS 리전 서비스에 영향을 미칠 수 있습니다. 리전 영향을 관찰하는 경우 해당 영향의 원인이 단일 가용 영역에서 발생하더라도 해당 이벤트를 리전 서비스 장애로 처리해야 합니다. 가용 영역을 제거한다고 해서 이러한 유형의 문제가 해결되지는 않습니다. 현재 마련되어 있는 대응 계획을 사용하여 리전 서비스 장애가 발생할 경우 이에 대응하십시오.

이 문서의 나머지 부분에서는 단일 AZ 회색 장애의 RTO와 RPO를 낮추기 위한 방법으로 두 번째 옵션인 가용 영역 제거에 중점을 둡니다. 이러한 패턴을 사용하면 다중 AZ 아키텍처의 가치와 효율성을 높이는 데 도움이 될 수 있으며, 대부분의 워크로드 클래스에서 이러한 유형의 이벤트를 처리하기 위해 다중 리전 아키텍처를 만들 필요성을 줄일 수 있습니다.

다중 AZ 관찰성

단일 가용 영역으로 격리된 이벤트 중에 가용 영역을 제거할 수 있으려면 먼저 장애가 실제로 단일 가용 영역에 격리되어 있음을 감지할 수 있어야 합니다. 이를 위해서는 각 가용 영역에서 시스템이 어떻게 작동하는지 정확하게 파악할 수 있어야 합니다. 많은 AWS 서비스는 리소스에 대한 운영 통찰력을 제공하는 기본 제공 메트릭을 제공합니다. 예를 들어 Amazon EC2는 CPU 사용률, 디스크 읽기 및 쓰기, 들어오고 나가는 네트워크 트래픽과 같은 다양한 지표를 제공합니다.

하지만 이러한 서비스를 사용하는 워크로드를 구축할 때는 표준 지표보다 더 많은 가시성이 필요합니다. 워크로드가 제공하는 고객 경험에 대한 가시성이 필요합니다. 또한 지표가 생성되는 가용 영역에 맞게 지표를 조정해야 합니다. 이는 차등적으로 관찰 가능한 회색 장애를 탐지하는 데 필요한 통찰력입니다. 이러한 수준의 가시성을 확보하려면 계측이 필요합니다.

계측을 사용하려면 명시적 코드를 작성해야 합니다. 이 코드는 작업에 걸리는 시간 기록, 성공 또는 실패한 항목 수 계산, 요청에 대한 메타데이터 수집 등과 같은 작업을 수행해야 합니다. 또한 임계값을 미리 정의하여 정상으로 간주되는 항목과 그렇지 않은 항목을 정의해야 합니다. 워크로드의 지연 시간, 가용성, 오류 수에 대한 목표와 다양한 심각도 임계값을 간략하게 설명해야 합니다. Amazon Builders' Library 문서 운영 가시성을 위한 분산 시스템 계측에 여러 모범 사례가 나와 있습니다.

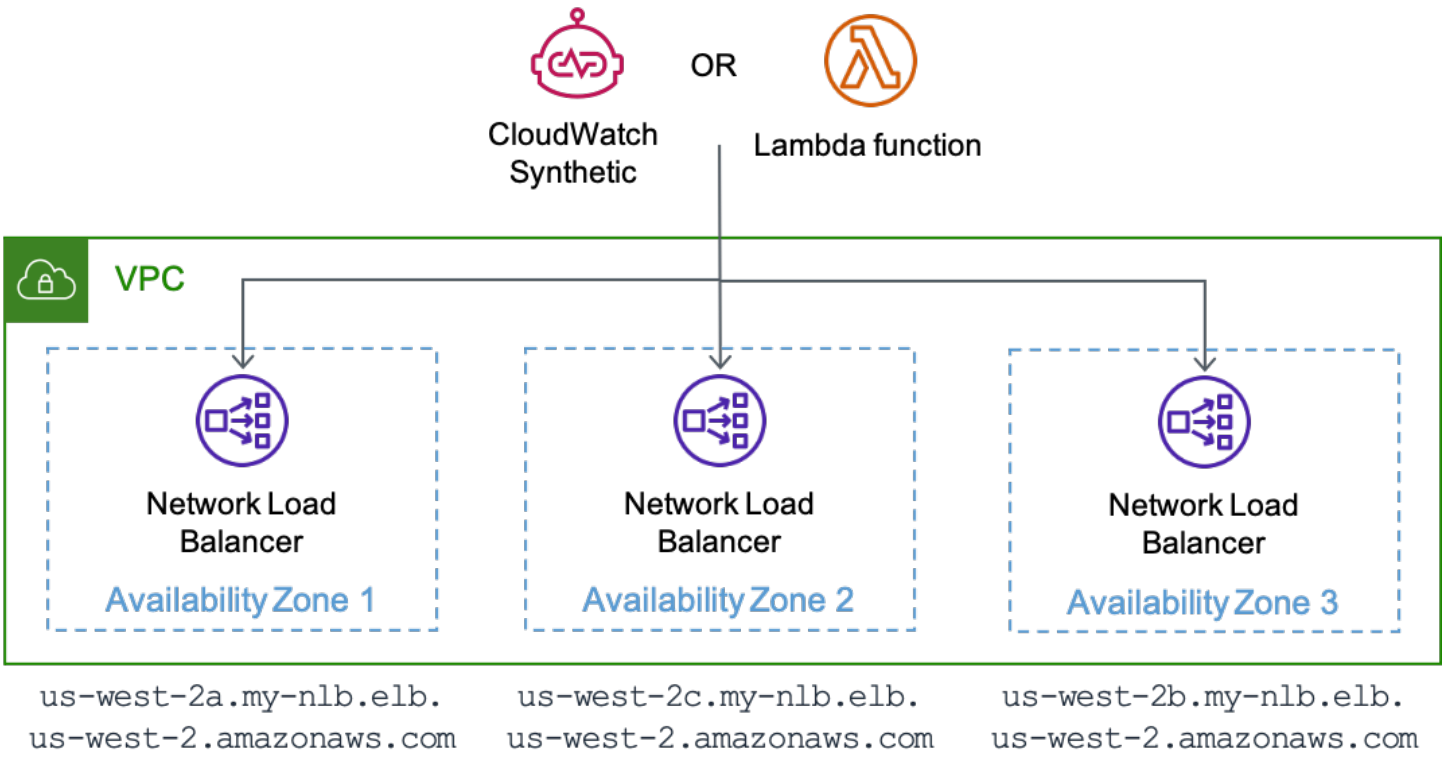
지표는 서버 측과 클라이언트 측 모두에서 생성되어야 합니다. 클라이언트측 지표를 생성하고 고객 경험을 이해하기 위한 모범 사례는 정기적으로 워크로드를 조사하고 지표를 기록하는 소프트웨어인 canary를 사용하는 것입니다.

이러한 지표를 생성하는 것 외에도 해당 지표의 컨텍스트를 이해해야 합니다. 이를 수행하는 한 가지 방법은 차원을 사용하는 것입니다. 차원은 지표에 고유한 정체성을 부여하고 지표가 의미하는 바를 설명하는 데 도움이 됩니다. 워크로드의 장애를 식별하는 데 사용되는 지표(예: 지연 시간, 가용성 또는 오류 수)의 경우 장애 격리 경계에 맞는 차원을 사용해야 합니다.

예를 들어, Model-View-Controller(MVC) 웹 프레임워크를 사용하여 한 리전의 여러 가용 영역에서 웹 서비스를 실행하는 경우 `Region` `Availability Zone ID`,`Controller`,`Action`, 및 `InstanceId`를 차원 세트의 차원으로 사용해야 합니다(마이크로서비스를 사용하는 경우 컨트롤러 및 작업 이름 대신 서비스 이름 및 HTTP 메서드를 사용할 수 있음). 이는 이러한 한계로 인해 여러 유형의 장애가 격리될 것으로 예상되기 때문입니다. 제품을 나열하는 기능에 영향을 미치는 웹 서비스 코드의 버그가 홈 페이지에도 영향을 미칠 것으로 예상하지 못할 것입니다. 마찬가지로 단일 EC2 인스턴스의 전체 EBS 볼륨이 있어도 웹 콘텐츠를 제공하는 다른 EC2 인스턴스가 영향을 받을 것으로 예상하지는 않습니다. 가용 영역 ID 차원을 사용하면 가용 영역 관련 영향을 AWS 계정 전반에 걸쳐 일관되게 식별할 수 있습니다. 다양한 방법으로 워크로드 내에서 가용 영역 ID를 찾을 수 있습니다. 몇 가지 예는 부록 A — 가용 영역 ID 가져오기 를 참조하십시오.

이 문서에서는 주로 Amazon EC2를 컴퓨팅 리소스로 사용하지만, `InstanceId`는 차원의 구성 요소인 Amazon Elastic Container Servic(Amazon ECS) 및 Amazon Elastic Kubernetes Service(Amazon EKS) 컴퓨팅 리소스에 대한 컨테이너 ID로 대체 수 있습니다.

워크로드에 영역 엔드포인트가 있는 경우 canary는 `Controller`, `Action`, `AZ-ID` 및 `Region`를 지표의 차원으로 사용할 수도 있습니다. 이 경우 테스트 중인 가용 영역에서 실행되도록 canary 조정을 하십시오. 이렇게 하면 격리된 가용 영역 이벤트에서 canary가 실행 중인 가용 영역에 영향을 미치는 경우 테스트 중인 다른 가용 영역을 비정상으로 표시하는 지표를 기록하지 않습니다. 예를 들어, canary는 영역 DNS 이름을 사용하여 Network Load Balancer(NLB) 또는 Application Load Balancer(ALB) 기반 서비스의 각 영역 엔드포인트를 테스트할 수 있습니다.



CloudWatch Synthetics에서 실행되는 카나리아 또는 NLB의 각 영역 엔드포인트를 테스트하는 AWS Lambda 함수

이러한 차원으로 지표를 생성하면 해당 범위 내에서 가용성이나 지연 시간에 변화가 발생할 때 알려주는 Amazon CloudWatch 경보를 설정할 수 있습니다. 또한 대시보드를 사용하여 해당 데이터를 빠르게 분석할 수 있습니다. 지표와 로그를 모두 효율적으로 사용하기 위해 Amazon CloudWatch는 로그 데이터에 사용자 지정 지표를 내장할 수 있는 내장된 지표 형식(EMF)을 제공합니다. CloudWatch는 사용자 지정 지표를 자동으로 추출하므로 이를 시각화하고 경보를 확인할 수 있습니다. AWS는 EMF를 쉽게 시작할 수 있도록 다양한 프로그래밍 언어를 위한 여러 클라이언트 라이브러리를 제공합니다. Amazon EC2, Amazon ECS, Amazon EKS, AWS Lambda 및 온프레미스 환경에서 사용할 수 있습니

다. 지표가 로그에 내장되어 있으면 Amazon CloudWatch Contributor Insights를 사용하여 기여자 데이터를 표시하는 시계열 그래프를 생성할 수도 있습니다. 이 시나리오에서는 AZ-ID,InstanceId 또는 Controller와 같은 차원별로 그룹화된 데이터를 표시할 수 있을 뿐만 아니라 SuccessLatency나 HttpResponseCode와 같은 로그 내 다른 필드도 표시할 수 있습니다.

```
{
  "_aws": {
    "Timestamp": 1634319245221,
    "CloudWatchMetrics": [
      {
        "Namespace": "workloadname/frontend",
        "Metrics": [
          { "Name": "2xx", "Unit": "Count" },
          { "Name": "3xx", "Unit": "Count" },
          { "Name": "4xx", "Unit": "Count" },
          { "Name": "5xx", "Unit": "Count" },
          { "Name": "SuccessLatency", "Unit": "Milliseconds" }
        ],
        "Dimensions": [
          [ "Controller", "Action", "Region", "AZ-ID", "InstanceId"],
          [ "Controller", "Action", "Region", "AZ-ID"],
          [ "Controller", "Action", "Region"]
        ]
      }
    ],
    "LogGroupName": "/loggroupname"
  },
  "CacheRefresh": false,
  "Host": "use1-az2-name.example.com",
  "SourceIp": "34.230.82.196",
  "TraceId": "|e3628548-42e164ee4d1379bf.",
  "Path": "/home",
  "OneBox": false,
  "Controller": "Home",
  "Action": "Index",
  "Region": "us-east-1",
  "AZ-ID": "use1-az2",
  "InstanceId": "i-01ab0b7241214d494",
  "LogGroupName": "/loggroupname",
  "HttpResponseCode": 200,
  "2xx": 1,
  "3xx": 0,
  "4xx": 0,
```

```
  "5xx": 0,
  "SuccessLatency": 20
}
```

이러한 로그 내에 세 가지 차원 세트가 있습니다. 인스턴스부터 가용 영역, 리전에 이르기까지 세분화된 순서대로 진행됩니다 (`Controller` 및 `Action`은 예제에 항상 포함됨). 단일 인스턴스, 단일 가용 영역 또는 전체 AWS 리전 내에서 특정 컨트롤러 작업에 영향이 있을 때를 나타내는 경보를 워크로드 전반에 생성할 수 있도록 지원합니다. 이러한 차원은 2xx, 3xx, 4xx, 5xx HTTP 응답 지표의 수와 성공적인 요청 지표의 지연 시간에 사용됩니다 (요청이 실패하면 실패한 요청 지연 시간에 대한 지표도 기록됨). 로그는 HTTP 경로, 요청자의 소스 IP, 이 요청에 로컬 캐시를 새로 고쳐야 하는지 여부와 같은 기타 정보도 기록합니다. 그런 다음 이러한 데이터 포인트를 사용하여 워크로드가 제공하는 각 API의 가용성과 지연 시간을 계산할 수 있습니다.

> **가용성 지표에 HTTP 응답 코드를 사용하는 방법에 대한 참고 사항**
>
> 일반적으로 2xx 및 3xx 응답은 성공한 것으로 간주하고 5xx 응답은 실패로 간주할 수 있습니다. 4xx 응답 코드는 중간에 위치합니다. 일반적으로 클라이언트 오류로 인해 생성됩니다. 매개변수가 범위를 벗어나서 400 응답이 발생하거나, 존재하지 않는 것을 요청하여 404 응답이 발생할 수 있습니다. 이러한 응답을 워크로드의 가용성에 포함시키지는 않을 것입니다. 하지만 이는 소프트웨어 버그의 결과일 수도 있습니다.
>
> 예를 들어 이전에 성공했을 요청을 거부하는 보다 엄격한 입력 검증을 도입한 경우 400 응답은 가용성 저하로 간주될 수 있습니다. 아니면 고객을 제한 및 429 응답을 반환하는 것일 수도 있습니다. 고객이 서비스의 가용성을 유지하기 위해 서비스 제한을 하는 동안, 고객의 관점에서는 서비스가 고객의 요청을 처리할 수 없습니다. 4xx 응답 코드를 가용성 계산에 포함할지 여부를 결정해야 합니다.

이 섹션에서는 지표를 수집하고 분석하는 방법으로 CloudWatch를 사용하는 방법을 개괄적으로 설명했지만 이것이 사용할 수 있는 유일한 솔루션은 아닙니다. 또한 지표를 Amazon Managed Service for Prometheus 및 Amazon Managed Grafana, Amazon DynamoDB 테이블로 보내거나 타사 모니터링 솔루션을 사용하도록 선택할 수도 있습니다. 핵심은 워크로드가 생성하는 지표에 워크로드의 장애 격리 경계에 대한 컨텍스트를 포함해야 한다는 것입니다.

장애 격리 경계에 맞게 차원이 정렬된 메트릭을 생성하는 워크로드를 사용하면 가용 영역 격리 장애를 탐지하는 관찰성을 만들 수 있습니다. 다음 섹션에서는 단일 가용 영역의 손상으로 인해 발생하는 장애를 탐지하기 위한 세 가지 보완적인 접근 방식을 설명합니다.

주제

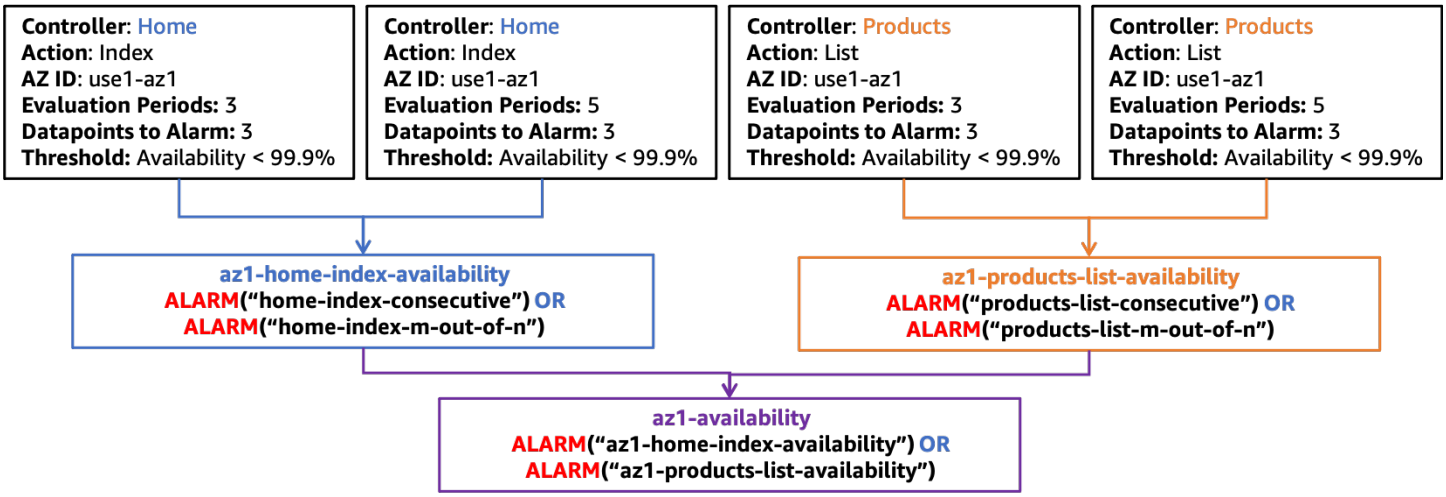
CloudWatch 복합 경보를 통한 장애 감지

CloudWatch 지표에서 각 차원 세트는 고유한 지표이며, 각 차원에 대해 CloudWatch 경보를 생성할 수 있습니다. 그런 다음 Amazon CloudWatch 복합 경보를 생성하여 이러한 지표를 집계할 수 있습니다.

영향을 정확하게 감지하기 위해 이 백서의 예시에서는 경보가 적용되는 각 차원 설정에 대해 서로 다른 두 가지 CloudWatch 경보 구조를 사용합니다. 각 경보는 1분의 기간을 사용합니다. 즉, 지표는 분당 한 번 평가됩니다. 첫 번째 접근 방식은 평가 기간 및 경고할 데이터 포인트를 설정하여 총 3분 동안 영향을 미치는 세 개의 연속 침해 데이터 포인트를 사용하는 것입니다. 두 번째 접근 방식은 5분 동안 데이터 요소 3개가 위반되는 경우 'M out of N'을 사용하여 평가 기간을 5로 설정하고 경고할 데이터 포인트를 3으로 설정하는 것입니다. 이를 통해 일정한 신호는 물론 짧은 시간 동안 변동하는 신호도 감지할 수 있습니다. 여기에 포함된 기간 및 데이터 포인트 수는 권장 사항이며 워크로드에 적합한 값을 사용하십시오.

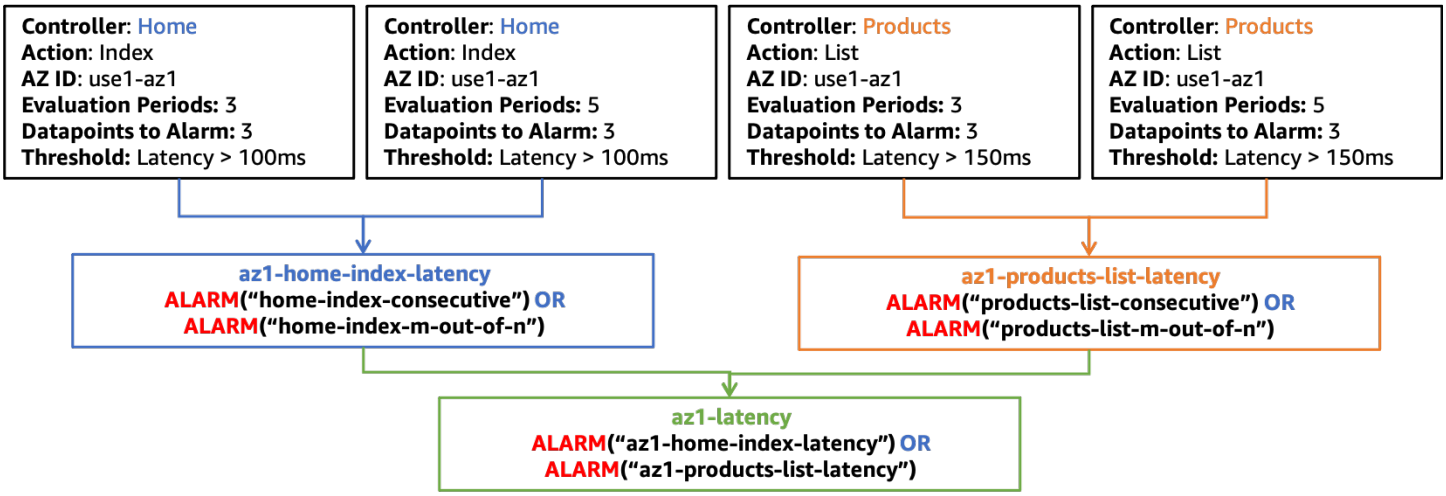
단일 가용 영역에서 영향 감지

이 구성을 사용할 때는, `Controller`, `Action`, `InstanceId`, `AZ-ID`, 및 `Region`을 차원으로 사용하는 워크로드를 고려해 보십시오. 워크로드는 Products와 Home이라는 두 개의 컨트롤러와 컨트롤러당 하나의 작업, 즉 List와 Index를 포함합니다. `us-east-1` 리전 내 세 가용 영역에서 운영됩니다. 각 가용 영역의 가용성 `Controller` 및 `Action` 조합에 대한 두 개의 경보와 각 가용 영역의 지연 시간에 대한 두 개의 경보를 생성해야 합니다. 그런 다음 각 `Controller` 조합과 `Action` 조합의 가용성에 대한 복합 경보를 생성하도록 선택할 수도 있습니다. 마지막으로 가용 영역에 대한 모든 가용성 경보를 집계하는 복합 경보를 생성합니다. 다은 그림은 각 `Controller` 및 `Action` 조합에 대해 선택적 복합 경보를 사용하여 단일 가용 영역, `use1-az1`을 보여줍니다(`use1-az2` 및 `use1-az3` 가용 영역에도 유사한 경보가 존재하지만 단순화를 위해 표시하지 않음).



use1-az1 내 가용성에 대한 복합 경보 구조

다음 그림과 같이 지연 시간에 대해서도 유사한 경보 구조를 구축할 수 있습니다.

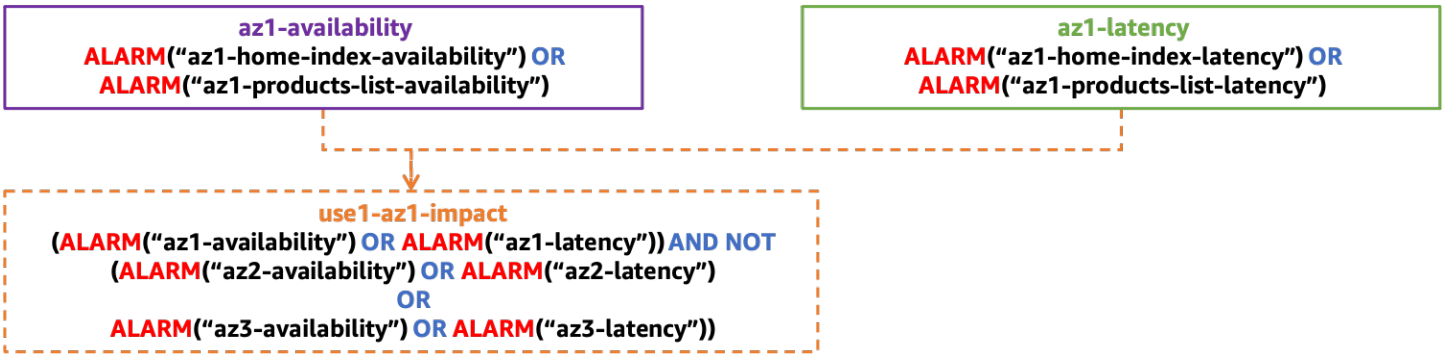


use1-az1 내 지연 시간에 대한 복합 경보 구조

이 섹션의 나머지 그림에서는 최상위 레벨에 `az1-availability` 및 `az1-latency` 복합 경보만 표시됩니다. 이러한 복합 경보 `az1-availability` 및 `az1-latency`는 워크로드의 특정 부분에 대해 특정 가용 영역에서 가용성이 정의된 임계값 이하로 떨어지거나 지연 시간이 증가하는지 알려줍니다. 단일 가용 영역의 워크로드가 작업을 받지 못하게 하는 영향을 감지하기 위해 처리량 측정을 고려할 수도 있습니다. canary로 내보낸 지표에서 생성된 경보를 이러한 복합 경보에 통합할 수도 있습니다. 이렇게 하여 서버 측 또는 클라이언트 측에서 가용성이나 지연 시간에 미치는 영향을 확인하면 경보가 알림을 생성합니다.

영향이 지역적이지 않은지 확인

또 다른 복합 경보 세트를 사용하여 격리된 가용 영역 이벤트만 경보가 활성화되도록 할 수 있습니다. 이는 다른 가용 영역에 대한 복합 경보가 ALARM 상태에 있는 동안 가용 영역 복합 경보가 OK 상태에 있는지 확인하여 수행됩니다. 그러면 사용하는 가용 영역당 하나의 복합 경보가 생성됩니다. 다음 그림에 예가 나와 있습니다 (use1-az2 및 use1-az3, az2-latency, az2-availability, az3-latency 및 az3-availability에는 단순화를 위해 그림으로 표시되지 않은 지연 시간 및 가용성에 대한 경보가 있다는 점을 기억하십시오).



단일 AZ에 격리된 영향을 감지하기 위한 복합 경보 구조

단일 인스턴스로 인한 영향이 아닌지 확인

단일 인스턴스(또는 전체 플릿의 작은 비율)는 가용성 및 지연 시간 지표에 불균형적인 영향을 미칠 수 있으며, 이로 인해 전체 가용 영역이 영향을 받는 것처럼 보이지만 실제로는 그렇지 않을 수 있습니다. 문제가 되는 단일 인스턴스를 제거하는 것이 가용 영역을 제거하는 것보다 더 빠르고 효과적입니다.

인스턴스와 컨테이너는 일반적으로 임시 리소스로 취급되며, AWS Auto Scaling와 같은 서비스로 대체되는 경우가 많습니다. 새 인스턴스가 생성될 때마다 새로운 CloudWatch 경보를 생성하는 것은 어렵습니다(하지만 Amazon EventBridge 또는Amazon EC2 Auto Scaling 수명 주기 후크를 사용하면 확실히 가능합니다). 대신 CloudWatch Contributor Insights를 사용하여 가용성 및 지연 시간 지표에 기여하는 사람의 수를 파악할 수 있습니다.

예를 들어, HTTP 웹 애플리케이션의 경우 각 가용 영역에서 5xx HTTP 응답의 상위 기여자를 식별하는 규칙을 생성할 수 있습니다. 이를 통해 어떤 인스턴스가 가용성 저하의 원인인지 식별할 수 있습니다 (위에서 정의한 가용성 지표는 5xx 오류의 존재 여부에 따라 결정됨). EMF 로그 예제를 사용하여 InstanceId의 키를 사용하여 규칙을 생성합니다. 그런 다음 HttpResponseCode 필드를 기준으로 로그를 필터링합니다. 이 예는 use1-az1 가용 영역에 대한 규칙입니다.

```
{
    "AggregateOn": "Count",
```

```
    "Contribution": {
        "Filters": [
            {
                "Match": "$.InstanceId",
                "IsPresent": true
            },
            {
                "Match": "$.HttpStatusCode",
                "IsPresent": true
            },
            {
                "Match": "$.HttpStatusCode",
                "GreaterThan": 499
            },
            {
                "Match": "$.HttpStatusCode",
                "LessThan": 600
            },
            {
                "Match": "$.AZ-ID",
                "In": ["use1-az1"]
            },
        ],
        "Keys": [
            "$.InstanceId"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/loggroupname"
    ],
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    }
}
```

이러한 규칙을 기반으로 CloudWatch 경보를 생성할 수도 있습니다. 지표 수학 및 UniqueContributors 지표가 포함된 INSIGHT_RULE_METRIC 함수를 사용하여 Contributor Insights 규칙을 기반으로 경보를 생성할 수 있습니다. 가용성에 대한 규칙 외에도 지연 시간 또는 오류 수와 같은 지표에 대한 CloudWatch 경보를 사용하여 추가 Contributor Insights 규칙을 생성할 수도 있

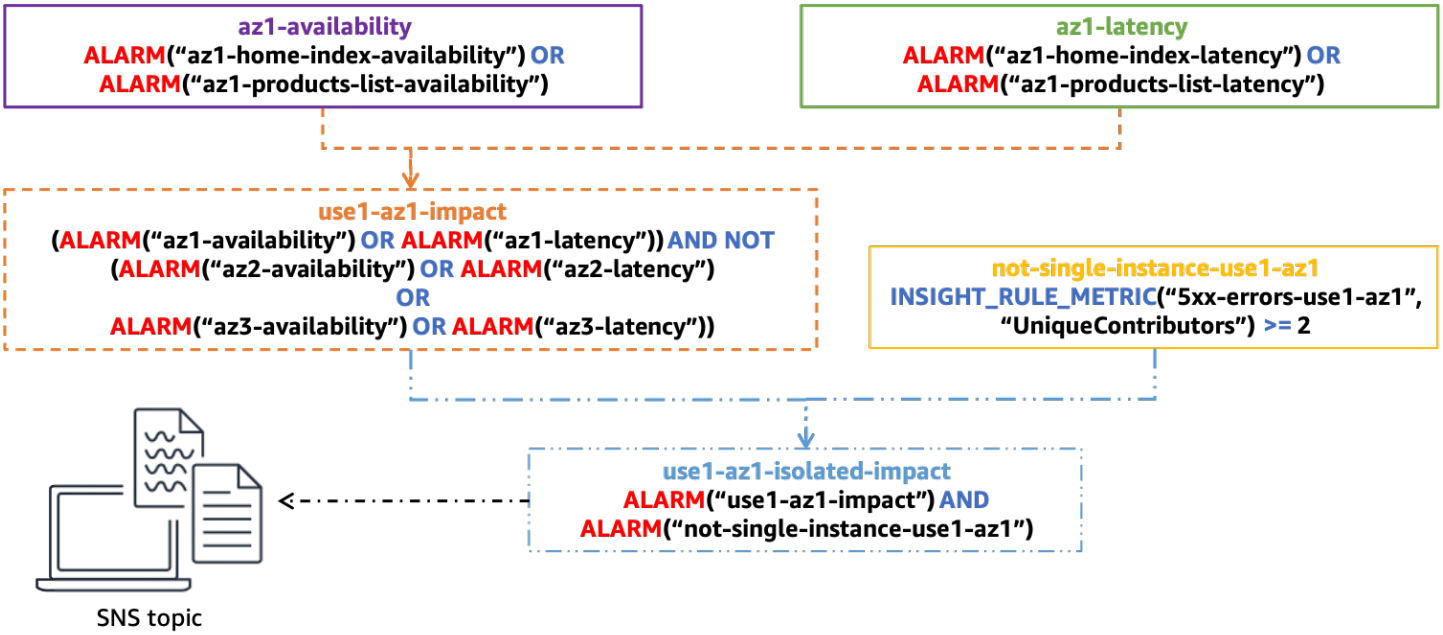
습니다. 이러한 경보를 격리된 가용 영역 영향 복합 경보와 함께 사용하여 단일 인스턴스가 경보를 활성화하지 않도록 할 수 있습니다. use1-az1에 대한 인사이트 규칙 지표는 다음과 같을 수 있습니다.

```
INSIGHT_RULE_METRIC("5xx-errors-use1-az1", "UniqueContributors")
```

이 지표가 임계값(이 예시에서는 2) 보다 클 때 경보를 정의할 수 있습니다. 5xx 응답의 고유 기여자가 해당 임계값을 초과할 때 활성화되며, 이는 두 개 이상의 인스턴스 내에서 영향이 발생하고 있음을 나타냅니다. 이 경보가 비교보다 작은 값보다 큰 값을 사용하는 이유는 고유 기여자의 값이 0이라고 해서 경보가 발생하지 않도록 하기 위함입니다. 이는 단일 인스턴스가 미치는 영향이 아님을 나타냅니다. 개별 워크로드에 맞게 이 임계값을 조정하세요. 일반적인 지침은 이 숫자를 가용 영역 전체 리소스의 5% 이상으로 만드는 것입니다. 영향을 받는 리소스의 5% 이상은 충분한 표본 크기를 고려할 때 통계적 유의성을 나타냅니다.

모두 통합

다음 그림은 단일 가용 영역에 대한 전체 복합 경보 구조를 보여줍니다.



단일 AZ 영향을 파악하기 위한 완전한 복합 경보 구조

최종 복합 경보 use1-az1-isolated-impact는 지연 시간 또는 가용성으로 인한 격리된 가용 영역 영향을 나타내는 복합 경보 use1-az1-aggregate-alarm가 ALARM 상태에 있고 동일한 가용 영역 not-single-instance-use1-az1에 대한 기여자 인사이트 규칙에 기반한 경보가 ALARM 상태에 있을 때 활성화됩니다(즉, 영향이 단일 인스턴스 이상이라는 의미). 워크로드가 사용하는 각 가용 영역에 대해 이 경보 스택을 생성합니다.

이 최종 경보에 Amazon Simple Notification Service(SNS) 알림을 첨부할 수 있습니다. 이전의 모든 경보는 별도의 조치 없이 구성됩니다. 경고는 이메일을 통해 운영자에게 수동 조사를 시작하라고 알릴 수 있습니다. 또한 자동화를 시작하여 가용 영역을 제거할 수도 있습니다. 하지만 이러한 경고에 대응하기 위해 구성 자동화에 주의를 기울여야 합니다. 가용 영역 대피 상황이 발생하면 증가된 오류율이 완화되고 경보가 OK 상태로 돌아가게 됩니다. 다른 가용 영역에서 영향이 발생하는 경우 자동화로 인해 두 번째 또는 세 번째 가용 영역이 제거되어 워크로드의 가용 용량이 모두 제거될 수 있습니다. 자동화에서는 조치를 취하기 전에 대피가 이미 수행되었는지 확인해야 합니다. 대피가 성공하기 전에 다른 가용 영역의 리소스를 확장해야 할 수도 있습니다.

MVC 웹 앱, 새 마이크로서비스 또는 일반적으로 별도로 모니터링하려는 추가 기능에 새 컨트롤러나 작업을 추가할 때는 이 설정에서 경보를 몇 개만 수정하면 됩니다. 해당 새 기능에 대한 새로운 가용성 및 지연 시간 경보를 생성한 다음 이를 여기에서 사용한 예의 적절한 가용 영역 정렬 가용성 및 지연 시간 복합 경보 `az1-latency`와 `az1-availability`에 추가합니다. 나머지 복합 경보는 구성된 후에도 정적 상태를 유지합니다. 따라서 이 접근 방식을 통해 새 기능을 온보딩하는 프로세스가 더 간단해집니다.

이상치 감지를 사용한 장애 감지

여러 가용 영역에서 상관관계가 없는 이유로 발생하는 오류율 상승을 볼 때 이전 접근 방식과 한 가지 차이가 발생할 수 있습니다. 세 개의 가용 영역에 EC2 인스턴스를 배포하고 가용성 경보 임계값이 99%인 시나리오를 상상해 보십시오. 그러면 단일 가용 영역 장애가 발생하여 많은 인스턴스가 격리되고 해당 영역의 가용성이 55% 로 떨어집니다. 동시에 다른 가용 영역에서는 단일 EC2 인스턴스가 EBS 볼륨의 모든 스토리지를 소모하여 더 이상 로그 파일을 쓸 수 없게 됩니다. 이로 인해 오류가 반환되기 시작하지만 로드 밸런서 상태 검사를 통과하면 로그 파일 작성이 트리거되지 않기 때문에 여전히 로드 밸런서 상태 검사를 통과합니다. 그 결과 해당 가용 영역의 가용성이 98% 로 떨어집니다. 이 경우 여러 가용 영역에서 가용성에 영향을 미치기 때문에 단일 가용 영역 영향 경보가 활성화되지 않습니다. 하지만 손상된 가용 영역을 제거하면 거의 모든 영향을 완화할 수 있습니다.

일부 유형의 워크로드 내에서 이전 가용성 지표가 유용하지 않을 수 있는 모든 가용 영역에서 일관되게 오류가 발생할 수 있습니다. AWS Lambda를 예를 들어 보겠습니다. AWS를 통해 고객이 Lambda 함수 내에서 실행할 자체 코드를 생성할 수 있습니다. 서비스를 사용하려면 종속성을 포함한 ZIP 파일에 코드를 업로드하고 함수의 진입점을 정의해야 합니다. 그러나 고객이 이 부분을 잘못 이해하는 경우가 있습니다. 예를 들어 ZIP 파일의 중요한 종속성을 잊어버리거나 Lambda 함수 정의에서 메서드 이름을 잘못 입력할 수 있습니다. 이로 인해 함수를 간접적으로 호출하는 것에 실패하고 오류가 발생합니다. AWS Lambda는 이러한 종류의 오류를 항상 확인하지만 이는 반드시 비정상임을 나타내는 것은 아닙니다. 하지만 가용 영역 장애와 같은 원인으로 인해 이러한 오류가 나타날 수도 있습니다.

이러한 종류의 노이즈에서 신호를 찾으려면 이상치 감지를 사용하여 가용 영역 간의 오류 수에 통계적으로 유의한 편차가 있는지 확인할 수 있습니다. 여러 가용 영역에서 오류가 발생하긴 하지만 그 중 하나에 실제로 장애가 발생한 경우 해당 가용 영역에서 다른 가용 영역에 비해 오류율이 훨씬 높거나 더 낮을 수 있습니다. 하지만 얼마나 높거나 낮을까요?

이 분석을 수행하는 한 가지 방법은 카이 제곱(χ^2) 검정을 사용하여 가용 영역 간 오류율의 통계적으로 유의한 차이를 탐지하는 것입니다(이상값 탐지를 수행하는 알고리즘은 매우 다양함). 카이 제곱 테스트의 작동 방식을 살펴보겠습니다.

카이 제곱 테스트는 결과의 일부 분포가 발생할 확률을 평가합니다. 이 경우에는 정의된 일부 AZ 집합에 걸친 오차 분포를 살펴보겠습니다. 이 예제에서는 계산을 더 쉽게 하기 위해 가용 영역 4개를 고려해 보겠습니다.

먼저 기본 결과가 무엇이라고 생각하는지 정의하는 귀무가설을 세우십시오. 이 테스트에서 귀무가설은 오류가 각 가용 영역에 고르게 분포될 것으로 예상한다는 것입니다. 그런 다음 오류가 균등하게 분포되지 않아 가용 영역이 손상되었다는 대안가설을 세우십시오. 이제 메트릭의 데이터를 사용하여 이러한 가설을 테스트할 수 있습니다. 이를 위해 5분 동안 지표를 샘플링해 보겠습니다. 해당 창에 1,000개의 게시된 데이터 포인트가 있고 총 100개의 오류가 표시된다고 가정해 보겠습니다. 균등한 분포를 사용하면 네 가용 영역 각각에서 25%의 시간 동안 오류가 발생할 것으로 예상합니다. 다음 표에 예상한 내용과 실제로 본 내용이 비교되어 나타난다고 가정해 보겠습니다.

표 1: 발생한 예상 오류와 실제 오류 비교

AZ	예상	실제
use1-az1	25	20
use1-az2	25	20
use1-az3	25	25
use1-az4	25	35

따라서 실제 분포는 균일하지 않다는 것을 알 수 있습니다. 하지만 샘플링한 데이터 포인트가 어느 정도 무작위적이기 때문에 이런 일이 발생했다고 생각할 수도 있습니다. 표본 집합에서 이러한 유형의 분포가 발생할 확률은 어느 정도 있지만 귀무가설이 참이라고 가정할 수 있습니다. 이는 다음과 같은 질문으로 이어집니다. 적어도 이 극단적인 결과를 얻을 확률은 얼마일까요? 해당 확률이 정의된 임계값보다 낮으면 귀무가설을 기각합니다. 통계적으로 유의하려면, 이 확률은 5% 이하여야 합니다.[1]

[1] Craparo, Robert M. (2007). “중요도 수준”. Salkind, Neil J. 측정 및 통계 백과사전 3. 캘리포니아주 사우전드 오크스: SAGE 간행물. 889~891페이지. ISBN 1-412-91611-9.

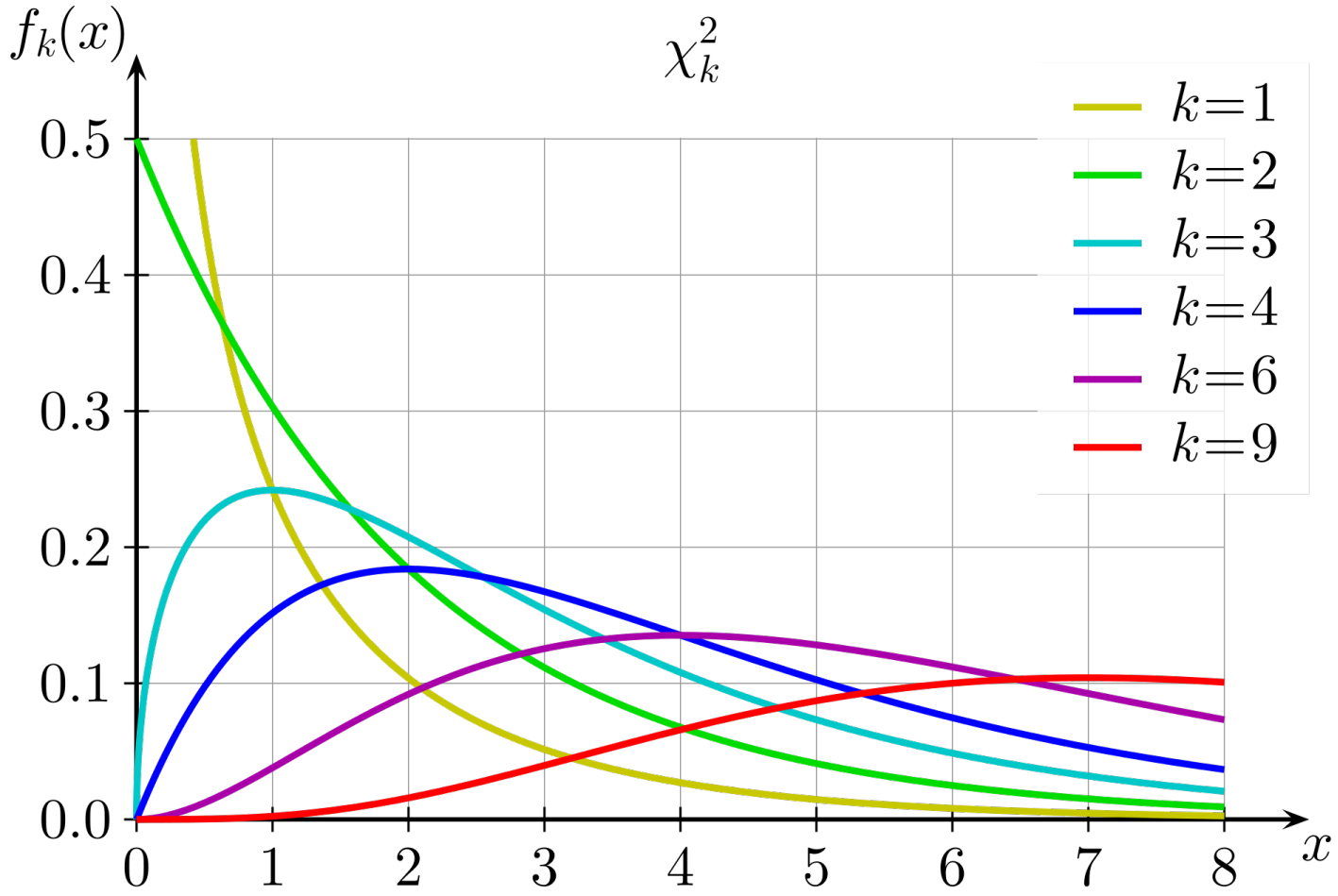
이 결과가 나올 확률은 어떻게 계산할까요? 매우 잘 연구된 분포를 제공하는 χ^2 통계를 용하여 이 극단적 또는 더 극단적인 결과가 나올 확률을 결정하는 데 이를 사용할 수 있습니다.

$$E_i = expected\ observations\ of\ type\ i$$
$$O_i = actual\ observations\ of\ type\ i \qquad (1)$$
$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

이 예제의 결과는 다음과 같습니다.

$$\chi^2 = \frac{(20-25)^2}{25} + \frac{(20-25)^2}{25} + \frac{(25-25)^2}{25} + \frac{(35-25)^2}{25}$$
$$\chi^2 = \frac{-5^2}{25} + \frac{-5^2}{25} + \frac{0^2}{25} + \frac{10^2}{25} \qquad (2)$$
$$\chi^2 = 1 + 1 + 0 + 4$$
$$\chi^2 = 6$$

그렇다면 확률 측면에서 6은 무엇을 의미합니까? 적절한 자유도를 가진 카이-제곱 분포를 살펴봐야 합니다. 다음 그림은 다양한 자유도에 대한 여러 카이-제곱 분포를 보여줍니다.



다양한 자유도에 대한 카이 제곱 분포

자유도는 테스트에서 선택한 항목 수보다 하나 적은 것으로 계산됩니다. 이 경우 가용 영역이 4개이므로 자유도는 3입니다. 그 다음 k = 3 그림의 x ≥ 6에 대한 곡선 아래 면적(적분) 을 구해보겠습니다. 일반적으로 사용되는 값이 포함된 미리 계산된 표를 사용하여 해당 값의 근사치를 계산할 수도 있습니다.

표 2: 카이 제곱 임계값

자유도	임계값보다 낮은 확률				
	0.75	0.90	0.95	0.99	0.999
1	1.323	2.706	3.841	6.635	10.828
2	2.773	4.605	5.991	9.210	13.816
3	4.108	6.251	7.815	11.345	16.266

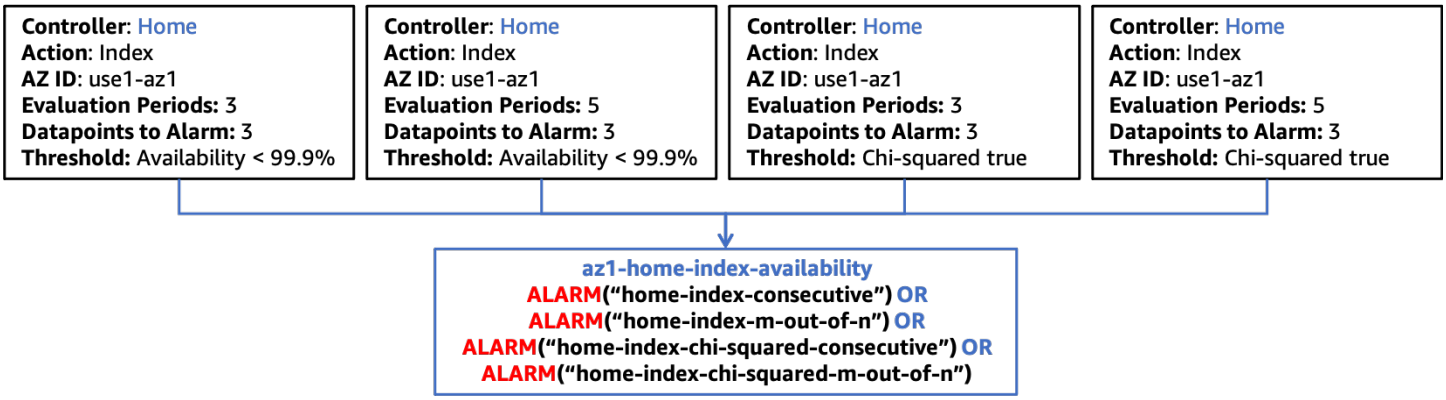
자유도	임계값보다 낮은 확률				
	0.75	0.90	0.95	0.99	0.999
4	5.385	7.779	9.488	13.277	18.467

자유도가 3인 경우 카이-제곱 값인 6은 확률 열 0.75와 0.9 사이에 속합니다. 이것이 의미하는 바는 이 분포가 발생할 확률이 10% 이상이며, 이는 5% 임계값 이상이라는 것입니다. 따라서 귀무가설을 받아들이고 가용 영역 간의 오류율에 통계적으로 유의한 차이가 없다고 판단합니다.

CloudWatch 지표 수학에서는 카이-제곱 통계 테스트 수행이 기본적으로 지원되지 않으므로 CloudWatch에서 해당하는 오류 측정치를 수집하여 Lambda와 같은 컴퓨팅 환경에서 테스트를 실행해야 합니다. 이 테스트를 MVC 컨트롤러/액션, 개별 마이크로서비스 수준 또는 가용 영역 수준에서 수행할지 결정할 수 있습니다. 가용 영역 장애가 각 컨트롤러/액션 또는 마이크로서비스에 동일하게 영향을 미칠지, DNS 장애와 같은 문제가 처리량이 적은 서비스에 영향을 미치고 처리량이 높은 서비스에서는 영향을 미치지 않아 집계 시 영향을 가릴 수 있는지 고려해야 합니다. 어느 경우든 적절한 차원을 선택하여 쿼리를 생성하십시오. 세분 수준은 생성되는 CloudWatch 경보의 결과에도 영향을 미칩니다.

지정된 기간 내에 각 AZ 및 컨트롤러/작업에 대한 오류 수 지표를 수집합니다. 먼저 카이-제곱 테스트 결과를 참(통계적으로 유의한 편차가 있음) 또는 거짓(존재하지 않았으므로 귀무가설 성립)으로 계산합니다. 결과가 거짓이면 메트릭 스트림에 0 데이터 포인트를 게시하여 각 가용 영역에 대한 카이-제곱 결과를 얻으십시오. 결과가 참인 경우, 예상 값에서 가장 멀리 떨어진 오류가 있는 가용 영역에 대해 1 데이터 포인트를 게시하고 나머지는 0으로 게시합니다 (Lambda 함수 내에서 사용할 수 있는 샘플 코드는 부록 B — 카이 제곱 계산의 예 참조). Lambda 함수 내에서 생성되는 데이터 포인트를 기반으로 3 연속 CloudWatch 지표 경보와 5점 중 3점의 CloudWatch 지표 경보를 생성하여 이전 가용성 경보와 동일한 접근 방식을 따를 수 있습니다. 이전 예시에서처럼 이 접근 방식을 수정하여 더 짧거나 더 긴 기간에 더 많거나 적은 데이터 포인트를 사용할 수 있습니다.

그 다음, 다음 그림에 표시된 컨트롤러 및 작업 조합에 대한 기존 가용 영역 가용성 경보에 이러한 경보를 추가하십시오.



카이 제곱 통계 테스트를 복합 경보와 통합

앞서 언급했듯이 워크로드에 새 기능을 온보딩할 때는 새 기능에 맞는 적절한 CloudWatch 지표 경보를 생성하고 이러한 경보를 포함하도록 복합 경보 계층 구조의 다음 계층을 업데이트하기만 하면 됩니다. 나머지 경보 구조는 정적으로 유지됩니다.

단일 인스턴스 영역 리소스의 장애 감지

경우에 따라 영역 리소스의 단일 활성 인스턴스가 있을 수 있으며, 대부분의 경우 관계형 데이터베이스(예: Amazon RDS) 또는 분산 캐시(예: Amazon ElastiCache for Redis) 와 같은 단일 라이터 구성 요소가 필요한 시스템입니다. 단일 가용 영역 손상이 기본 리소스가 속한 가용 영역에 영향을 미치는 경우 해당 리소스에 액세스하는 모든 가용 영역에 영향을 미칠 수 있습니다. 이로 인해 모든 가용 영역에서 가용성 임계값이 초과될 수 있으며, 이는 첫 번째 접근 방식으로는 영향의 단일 가용 영역 원인을 정확하게 식별하지 못할 수 있습니다. 또한 각 가용 영역에서 유사한 오류율이 나타날 수 있습니다. 즉, 이상치 분석으로도 문제를 발견하지 못할 수 있습니다. 즉, 이 시나리오를 구체적으로 탐지하려면 추가 관찰성을 구현해야 합니다.

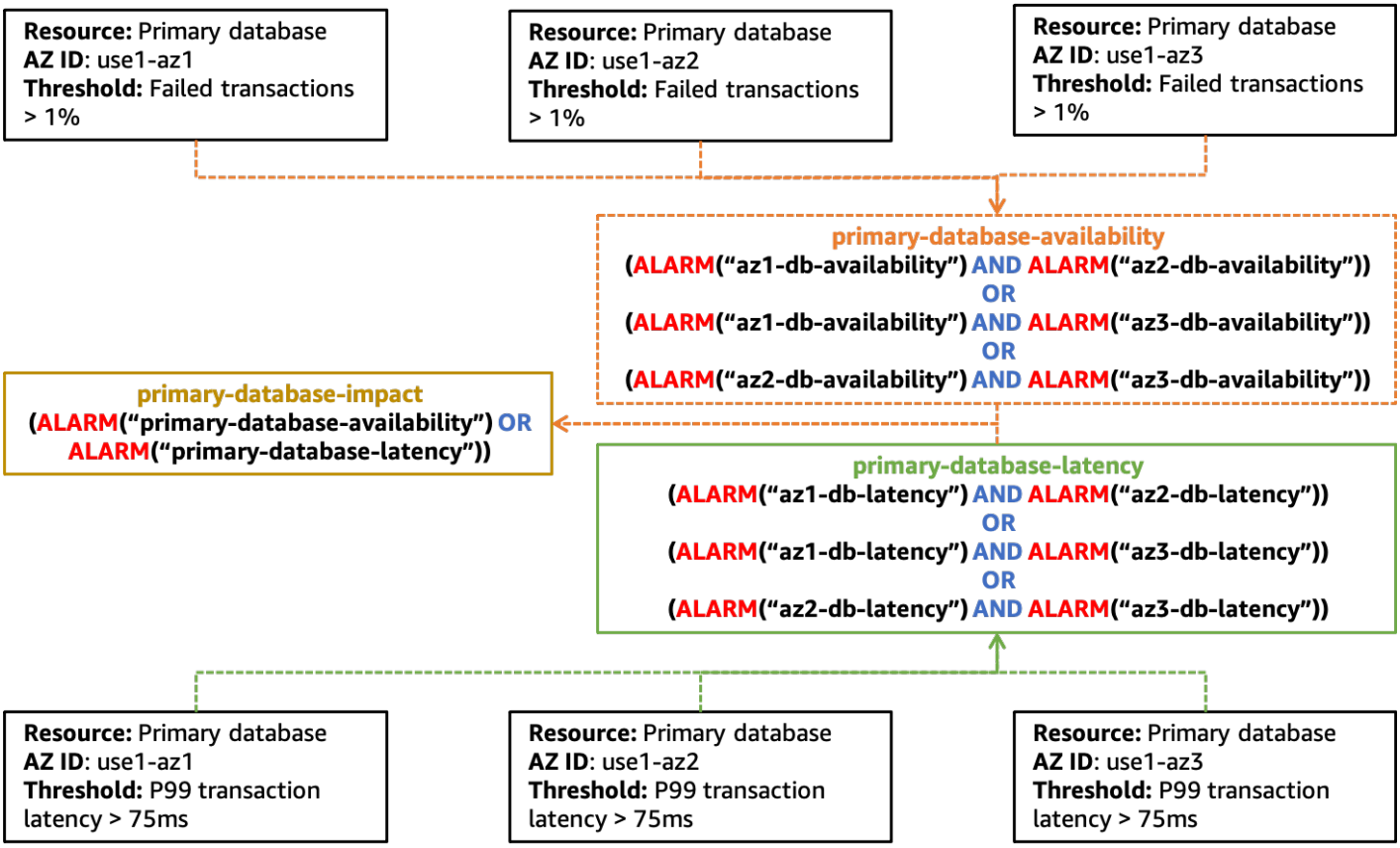
우려되는 리소스가 자체 상태에 대한 지표를 생성할 가능성이 높지만 가용 영역에 장애가 발생하면 해당 리소스가 해당 지표를 제공하지 못할 수 있습니다. 이 시나리오에서는 플라잉 블라인드 상태가 언제인지 알 수 있도록 경보를 만들거나 업데이트해야 합니다. 이미 모니터링하고 경보를 발령한 중요 지표가 있는 경우 누락된 데이터를 위반으로 처리하도록 경보를 구성할 수 있습니다. 이렇게 하면 리소스가 데이터 보고를 중단하는지 여부를 알 수 있으며, 이전에 사용한 동일한 경보와 n개 중 m개의 경보를 연속으로 포함할 수 있습니다.

또한 리소스 상태를 나타내는 일부 지표에서는 활동이 없을 때 값이 0인 데이터 포인트를 게시할 수도 있습니다. 장애로 인해 리소스와의 상호 작용이 불가능하다면 이러한 종류의 지표에는 누락된 데이터 접근 방식을 사용할 수 없습니다. 또한 값이 정상 임계값 내에 있는 합법적인 시나리오가 있을 수 있으므로 값이 0이라고 해서 경보를 울리는 것은 좋지 않을 것입니다. 이러한 유형의 문제를 탐지하는 가장 좋은 방법은 리소스가 이 종속성을 사용하여 산출한 지표를 사용하는 것입니다. 이 경우 복합 경보를

사용하여 다중 가용 영역에 미치는 영향을 감지하고자 합니다. 이러한 경보는 리소스와 관련된 몇 가지 중요한 지표 카테고리를 사용해야 합니다. 몇 가지 예가 아래에 나열되어 있습니다.

- 처리량 — 들어오는 작업 단위의 비율. 여기에는 트랜잭션, 읽기, 쓰기 등이 포함될 수 있습니다.
- 가용성 — 성공한 작업 단위와 실패한 작업 단위의 수를 측정합니다.
- 지연 시간 — 중요한 작업에서 성공적으로 수행된 작업의 지연 시간을 여러 백분위수로 측정합니다.

다시 한 번 말씀드리지만, 측정하려는 각 지표 범주의 각 지표에 대해 연속 및 m개 중 n개 지표 경보를 생성할 수 있습니다. 이전과 마찬가지로 이러한 경보를 복합 경보로 결합하여 이 공유 리소스가 가용 영역 전체에 미치는 영향의 근원인지 확인할 수 있습니다. 복합 경보를 사용하여 둘 이상의 가용 영역에 미치는 영향을 식별하고자 하시겠지만, 그 영향이 반드시 모든 가용 영역일 필요는 없습니다. 이러한 접근 방식에 대한 상위 수준의 복합 경보 구조가 다음 그림에 나와 있습니다.



단일 리소스로 인해 여러 가용 영역에 미치는 영향을 감지하기 위한 경보를 생성하는 예

이 다이어그램이 사용해야 하는 지표 경보 유형과 복합 경보의 계층 구조에 대해 덜 규정적인 것을 알 수 있습니다. 이런 종류의 문제를 발견하는 것은 어려울 수 있으며 공유 리소스에 대한 올바른 신호에 주의를 기울여야 하기 때문입니다. 이러한 신호를 특정한 방식으로 평가해야 할 수도 있습니다.

또한 primary-database-impact 경보가 특정 가용 영역과 연결되어 있지 않다는 것도 알 수 있습니다. 이는 기본 데이터베이스 인스턴스가 사용하도록 구성된 모든 가용 영역에 위치할 수 있으며, 데이터베이스 인스턴스의 위치를 지정하는 CloudWatch 지표가 없기 때문입니다. 이 경보가 활성화되면 이를 리소스에 문제가 있을 수 있다는 신호로 사용하고 자동으로 수행되지 않은 경우 다른 가용 영역으로 장애 조치를 시작해야 합니다. 리소스를 다른 가용 영역으로 이동한 후에는 격리된 가용 영역 경보가 활성화되었는지 기다리거나 가용 영역 제거 계획을 선제적으로 간접적으로 호출하도록 선택할 수 있습니다.

요약

이 섹션에서는 단일 가용 영역 장애를 식별하는 데 도움이 되는 세 가지 접근 방식을 설명했습니다. 각 접근 방식을 함께 사용하여 워크로드 상태를 전체적으로 파악해야 합니다.

CloudWatch 복합 경보 접근 방식을 사용하면 단일 공유 리소스로 발생하지 않고 가용성의 불균형이 통계적으로 유의하지 않은 문제(가령 98%(손상된 가용 영역), 100%, 99.99%의 가용성)를 찾을 수 있습니다.

이상치 감지는 다중 가용 영역에서 상관 관계가 없는 오류가 발생하여 경보 임계값을 모두 초과하는 단일 가용 영역 장애를 감지하는 데 도움이 됩니다.

마지막으로, 단일 인스턴스 영역 리소스의 성능 저하를 식별하면 가용 영역 장애가 가용 영역 전체에서 공유되는 리소스에 영향을 미치는 시기를 파악하는 데 도움이 됩니다.

이러한 각 패턴의 결과 경보를 CloudWatch 복합 경보 계층 구조로 결합하여 단일 가용 영역 장애가 발생하고 워크로드의 가용성 또는 지연 시간에 영향을 미치는 시기를 찾아낼 수 있습니다.

가용 영역 대피 패턴

단일 가용 영역에서 영향을 감지한 후 다음 단계는 해당 가용 영역을 제거하는 것입니다. 대피로 달성해야 할 결과는 두 가지입니다.

먼저, 영향을 받는 가용 영역으로 작업을 보내는 것을 중단해야 합니다. 이는 아키텍처마다 의미가 다를 수 있습니다. 요청/응답 워크로드 내에서 이는 고객으로부터 오는 HTTP 또는 gRPC 요청과 같은 것들이 로드 밸런서 또는 가용 영역의 다른 리소스로 전송되지 않도록 하는 것을 의미합니다. 일괄 처리 또는 대기열 처리 시스템에서 이는 영향을 받는 가용 영역에서 컴퓨팅 리소스가 작업을 처리하는 것을 중단하는 것을 의미할 수 있습니다. 또한 영향을 받지 않는 가용 영역의 리소스가 영향을 받는 가용 영역의 리소스와 상호 작용하는 것을 방지해야 합니다. 예를 들어 EC2 인스턴스가 영향을 받는 가용 영역의 인터페이스 VPC 엔드포인트로 트래픽을 전송하거나 데이터베이스의 기본 인스턴스에 연결하는 경우 등이 이에 해당합니다.

두 번째 결과는 영향을 받는 가용 영역에 새 용량이 프로비저닝되지 않도록 하는 것입니다. 영향을 받는 가용 영역에 프로비저닝되는 EC2 인스턴스 또는 컨테이너와 같은 새 리소스가 기존 리소스와 동일한 영향을 받을 가능성이 높기 때문에 이는 중요합니다. 또한 첫 번째 결과에서는 작업이 해당 사용자에게 전달되지 않기 때문에 처리하도록 프로비저닝된 부하를 흡수할 수 없습니다. 이로 인해 기존 리소스의 부하가 증가하여 결국 워크로드가 브라운 아웃되거나 완전히 사용할 수 없게 될 수 있습니다. 이 기능이 적용되는 AWS에서는 Amazon EC2 Auto Scaling, Application Auto Scaling 및 AWS Auto Scaling등 여러 가지 Auto Scaling 서비스를 사용할 수 있습니다. 또한 Amazon ECS, Amazon EKS 및 AWS Batch와 같은 서비스는 정상 운영의 일환으로 VPC의 가용 영역 전반에 걸친 호스트에 대한 작업을 예약할 수 있습니다.

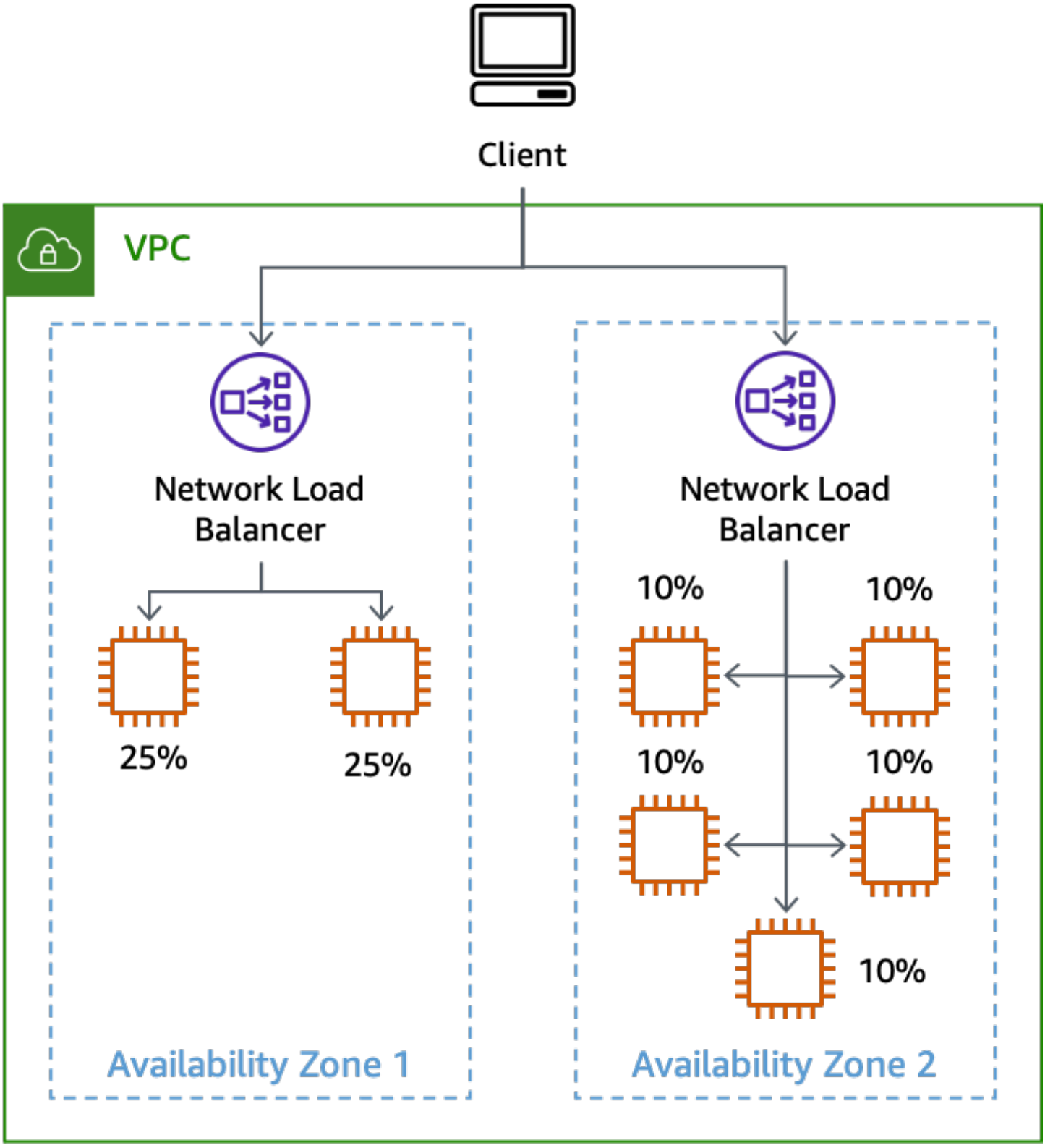
주제

가용 영역 독립성

첫 번째 결과를 달성하기 위해, 영향을 받는 가용 영역으로의 작업 전송을 중단하려면 대피 시 가용 영역 독립성(AZI)(가용 영역 선호도라고도 함)을 구현해야 합니다. 이 아키텍처 패턴은 리소스를 가용 영

역 내에 격리하고 다른 가용 영역의 기본 데이터베이스 인스턴스로 연결하는 것과 같이 꼭 필요한 경우를 제외하고는 서로 다른 가용 영역에 있는 리소스 간의 상호 작용을 방지합니다.

요청/응답 유형 워크로드 내에서 AZI를 구현하려면 Classic Load Balancer(CLB), Network Load Balancer(NLB), Application Load Balancers(ALB)에 대한 영역 간 로드 밸런싱을 비활성화해야 합니다(NLB에서는 영역 간 로드 밸런싱이 기본적으로 비활성화됨). 교차 영역 로드 밸런싱을 비활성화하면 몇 가지 단점이 있습니다. 영역 간 로드 밸런싱을 비활성화하면 각 가용 영역의 인스턴스 수에 관계없이 트래픽이 각 가용 영역 간에 균등하게 분할됩니다. 리소스나 오토 스케일링이 불균형한 경우 다른 가용 영역보다 리소스가 적은 가용 영역의 리소스에 추가 부하오토 스케일링가 발생할 수 있습니다. 다음 그림은 가용 영역 1의 인스턴스 2개가 각각 25% 의 부하를 받고 가용 영역 2의 5개 인스턴스가 각각 10% 의 부하를 받는 것을 보여줍니다.



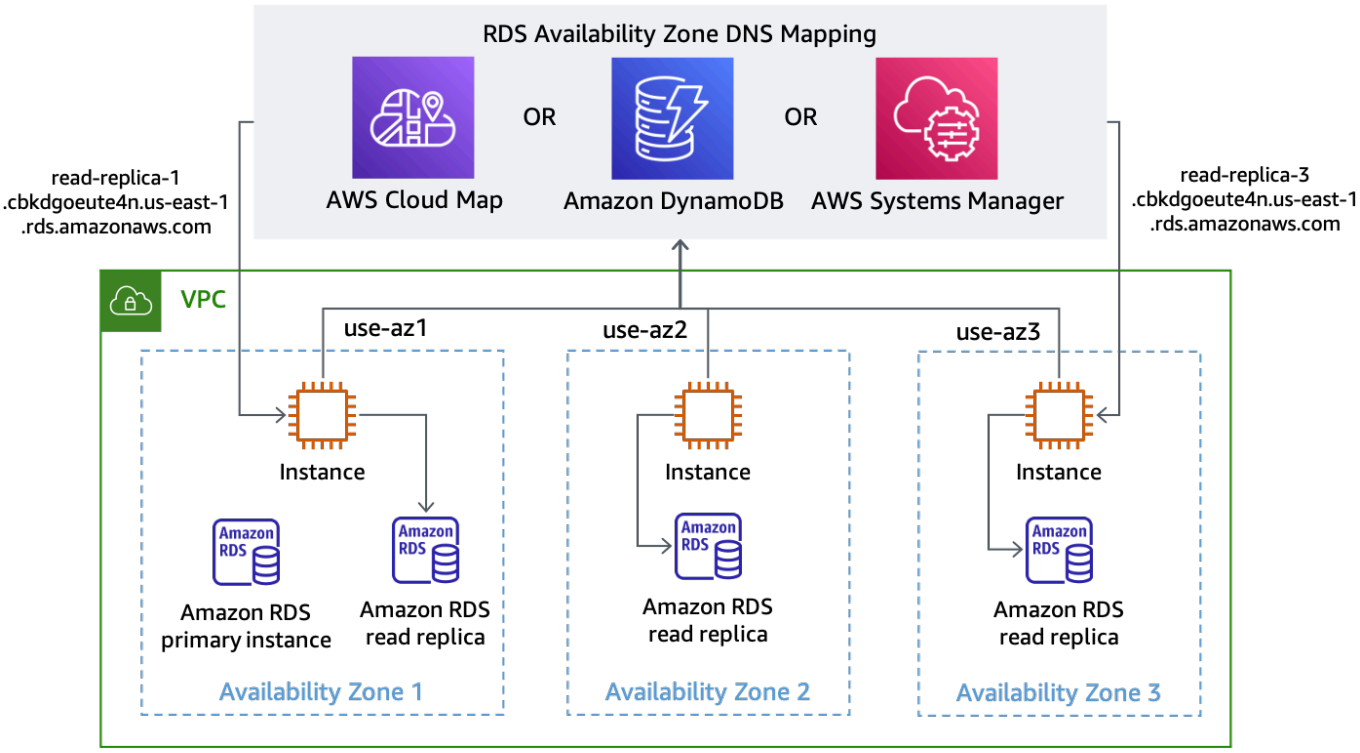
불균형 인스턴스로 교차 영역 로드 밸런싱을 비활성한 결과

사용하는 다른 영역 서비스에서도 효과적인 가용 영역 대피를 지원하려면 AZI 패턴을 사용하여 구현해야 합니다. 예를 들어 인터페이스 VPC 엔드포인트는 인터페이스 엔드포인트를 사용할 수 있는 각 가용 영역에 대한 특정 DNS 이름을 제공합니다.

AZI를 구현할 때 발생하는 한 가지 문제는 데이터베이스를 사용하는 것입니다. 특히 대부분의 관계형 데이터베이스는 항상 단일 기본 작성자만 지원하기 때문입니다. 기본 인스턴스와 통신할 때는 가용 영역 경계를 넘어야 할 수도 있습니다. 많은 AWS 데이터베이스 서비스는 사용자 정의 다중 AZ 구성을 지원하며 Amazon RDS 또는 Amazon Aurora와 같은 다중 AZ 장애 조치 특성이 내장되어 있습니다. 대부분의 장애 시나리오에서 서비스는 영향을 감지하고 문제 발생 시 데이터베이스를 다른 가용 영역으로 자동으로 장애 조치를 할 수 있습니다. 그러나 회색 장애가 발생하는 경우 서비스가 워크로드에 영향을 미치는 영향을 감지하지 못하거나 데이터베이스와 전혀 관련이 없을 수 있습니다. 이러한 경우 가용 영역에서 영향을 감지하면 수동으로 장애 조치를 간접적으로 호출하여 기본 데이터베이스를 이동할 수 있습니다. 이를 통해 단일 가용 영역 장애에 효과적으로 대응할 수 있습니다.

이러한 데이터베이스에서 읽기 전용 복제본을 사용하는 경우 기본 데이터베이스처럼 읽기 전용 복제본을 다른 가용 영역으로 장애 조치를 할 수 없으므로 이러한 데이터베이스를 위한 AZI를 구현하는 것도 좋습니다. 가용 영역 1에 읽기 전용 복제본이 하나 있고 3개 가용 영역의 인스턴스가 이를 사용하도록 구성된 경우 가용 영역 1에 영향을 미치는 장애는 다른 두 가용 영역의 작업에도 영향을 미칩니다. 이것이 바로 여러분이 막고자 하는 영향입니다.

RDS 인스턴스의 경우 특정 가용 영역의 복제본에 액세스할 수 있는 DNS 엔드포인트를 받습니다. AZI를 달성하려면 가용 영역별 읽기 전용 복제본이 있어야 하고 애플리케이션에서 해당 가용 영역에 어떤 복제본 엔드포인트를 사용할지 알 수 있는 방법이 필요합니다. 취할 수 있는 접근 방식 중 하나는 가용 영역 ID를 데이터베이스 식별자의 일부로 사용하는 것입니다. 예를 들면 `use1-az1-read-replica.cbkdgoeute4n.us-east-1.rds.amazonaws.com`과 같습니다. 서비스 검색(예: AWS Cloud Map)을 사용하거나 AWSSystems Manager Parameter Store 또는 DynamoDB 테이블에 저장된 간단한 맵을 조회하여 이 작업을 수행할 수도 있습니다. 이 개념은 다음 그림에 나와 있습니다.

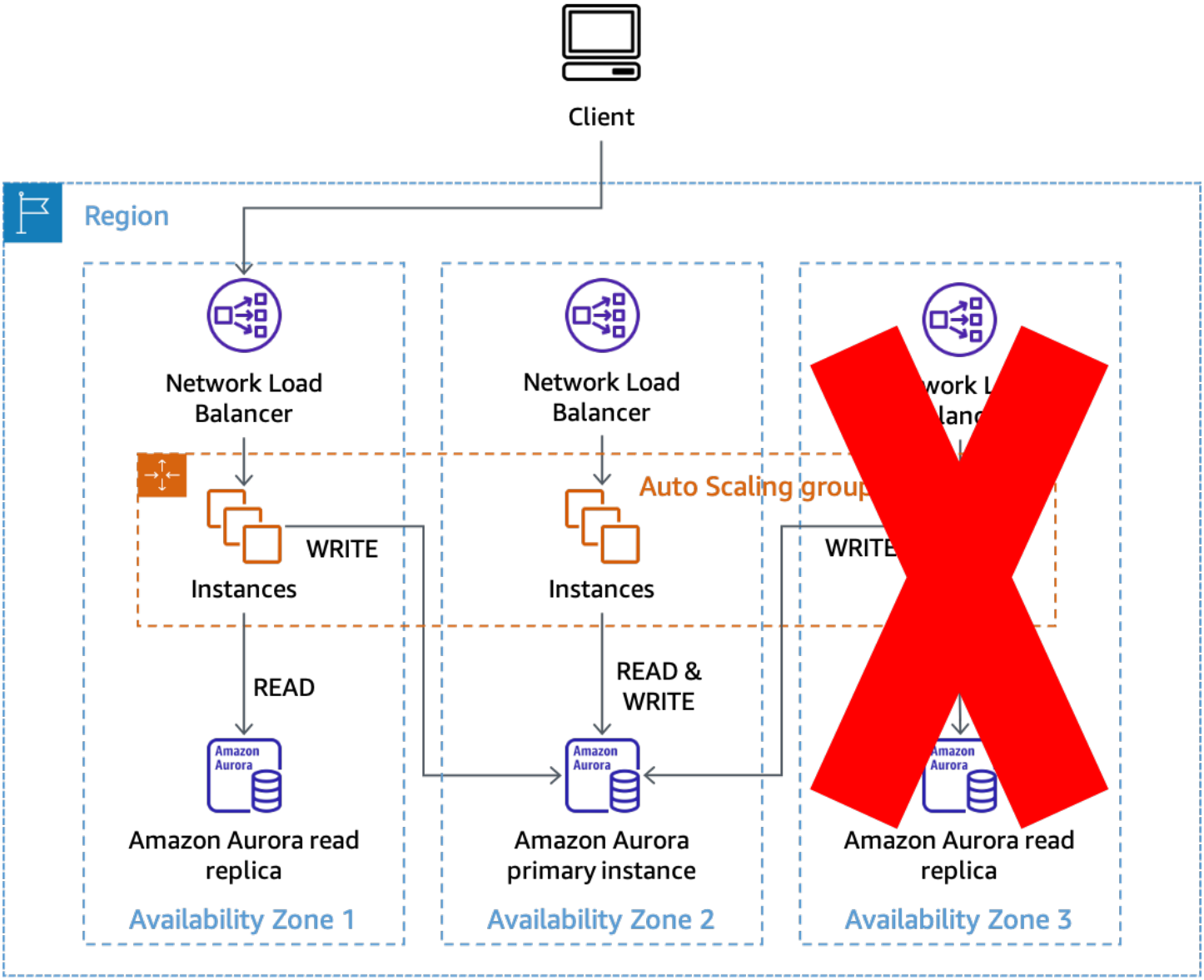


각 가용 영역의 RDS 엔드포인트 DNS 이름 검색

Amazon Aurora의 기본 구성은 사용 가능한 읽기 전용 복제본 간에 요청을 로드 밸런싱하는 단일 리더 엔드포인트를 제공하는 것입니다. Aurora를 사용하여 AZI를 구현하려면 ANY 유형을 사용하여 각 읽기 전용 복제본에 대한 사용자 지정 엔드포인트를 사용할 수 있습니다. 따라서 필요한 경우 읽기 전용 복제본을 승격할 수 있습니다. 복제본이 배포된 가용 영역 ID를 기반으로 사용자 지정 엔드포인트의 이름을 지정합니다. 그런 다음 사용자 지정 엔드포인트 내에서 제공하는 DNS 이름을 사용하여 다음 그림에 표시된 것처럼 특정 가용 영역의 특정 읽기 전용 복제본에 연결할 수 있습니다.

Aurora 읽기 전용 복제본에 사용자 지정 엔드포인트 사용

시스템이 이러한 방식으로 설계되면 가용 영역 제거 작업이 훨씬 간단해집니다. 예를 들어, 다음 그림에서 가용 영역 3에 영향을 미치는 장애가 있더라도 가용 영역 1과 2의 읽기 및 쓰기 작업은 모두 영향을 받지 않습니다.



AZI를 사용하여 Amazon Aurora 읽기 전용 복제본에 미치는 영향 방지

가용 영역 2가 영향을 받더라도, 그 대신에 가용 영역 1과 3에서의 읽기 작업은 계속 성공합니다. 그러면 Amazon Aurora가 기본 데이터베이스를 자동으로 장애 조치하지 않는 경우 다른 가용 영역으로 장애 조치를 수동으로 간접적으로 호출하여 쓰기 처리 기능을 복원할 수 있습니다. 이 접근 방식을 사용하면 가용 영역을 제거해야 할 때 데이터베이스 연결의 구성을 변경할 필요가 없습니다. 필요한 변경을 최소화하고 프로세스를 최대한 단순하게 유지하면 신뢰성을 높일 수 있습니다.

컨트롤 플레인 및 데이터 영역

가용 영역 제거를 수행하는 데 사용할 수 있는 실제 패턴을 살펴보기 전에 먼저 컨트롤 플레인과 데이터 영역의 개념을 설명해야 합니다. AWS는 당사 서비스의 컨트롤 플레인과 데이터 영역을 구분합니

다. 컨트롤 플레인은 리소스 추가, 리소스 삭제, 리소스 수정 등 시스템을 변경하고 이러한 변경 사항을 적용하기 위해 필요한 곳(예: ALB용 네트워크 구성 업데이트 또는 AWS Lambda 함수 생성)에 관련된 기계입니다.

데이터 영역은 EC2 인스턴스를 실행하거나 Amazon DynamoDB 테이블에서 항목을 가져오거나 Amazon DynamoDB 테이블에 항목을 넣는 등 이러한 리소스의 기본 기능입니다. 컨트롤 플레인과 데이터 영역에 대한 자세한 내용은 AWS 장애 격리 경계 및 가용 영역을 사용한 정적 안정성을 참조하십시오.

이 문서의 목적에 따라 컨트롤 플레인이 데이터 영역보다 유동적인 부분과 종속성이 더 많은 경향이 있다는 점을 고려해 보십시오. 따라서 데이터 영역에 비해 컨트롤 플레인이 손상될 가능성이 통계적으로 더 높습니다. 이는 Amazon EC2 및 EBS와 같이 AZI를 제공하는 서비스의 경우 특히 중요합니다. 이러한 서비스의 일부에는 영역별로 독립적이기도 한 컨트롤 플레인이 있어 단일 AZ 이벤트 중에 영향을 받을 수 있기 때문입니다.

이전 정보에 따르면 컨트롤 플레인 작업을 사용하여 AZ 대피를 수행할 수 있지만, 특히 장애 발생 시 성공 확률이 낮을 수 있습니다. 충격을 성공적으로 완화할 확률을 높이려면 두 가지 패턴을 사용할 수 있습니다. 첫 번째 패턴은 데이터 영역 작업에만 의존하여 작업이 영향을 받는 가용 영역으로 라우팅되는 것을 막거나 작업이 중단되는 것을 방지함으로써 초기에 영향을 완화합니다. 그런 다음, 영향을 받는 가용 영역에 용량이 프로비저닝되지 않도록 하고 해당 가용 영역과의 가용 영역 간 통신을 중지하도록 컨트롤 플레인 작업을 통해 리소스 구성을 업데이트하는 두 번째 패턴을 시도할 수 있습니다.

이 섹션에서 설명하는 복구 패턴은 커다란 빨간색 버튼입니다. 이러한 메커니즘은 크게 규모를 조정하는 조치를 신속하게 취하는 데 사용하는 메커니즘으로, 조립 라인에서 Andon 코드를 당기는 것과 비슷합니다. 이는 워크로드가 일시적인 오류를 극복하기 위해 코드에 지터를 포함한 지수 백오프를 포함하는 재시도와 같은 전략을 이미 시도했다고 가정합니다. 즉, 고립된 가용 영역의 영향이 감지되면 가용 영역으로 가용성이나 지연 시간에 미치는 영향이 심각하기 때문에 가용 영역을 제거해야 효과적으로 완화할 수 있습니다.

데이터 영역 제어 대피

데이터 영역 전용 작업을 사용하여 가용 영역 제거를 수행하기 위해 구현할 수 있는 몇 가지 솔루션이 있습니다. 이 섹션에서는 이 중 세 가지 그리고 둘 중 하나를 선택할 수 있는 사용 사례에 대해 설명합니다.

이러한 솔루션 중 하나를 사용할 때는 다른 가용 영역의 부하를 처리할 수 있을 만큼 나머지 가용 영역에 충분한 용량이 있는지 확인해야 합니다. 가장 탄력적인 방법은 각 가용 영역에 필요한 용량을 미리 프로비저닝하는 것입니다. 세 개의 가용 영역을 사용하는 경우 각 가용 영역에 배포된 최대 부하를 처

리하는 데 필요한 용량의 50% 가 필요하므로 단일 가용 영역이 손실되더라도 컨트롤 플레인에 의존하여 추가로 프로비저닝할 필요 없이 필요한 용량을 100% 유지할 수 있습니다.

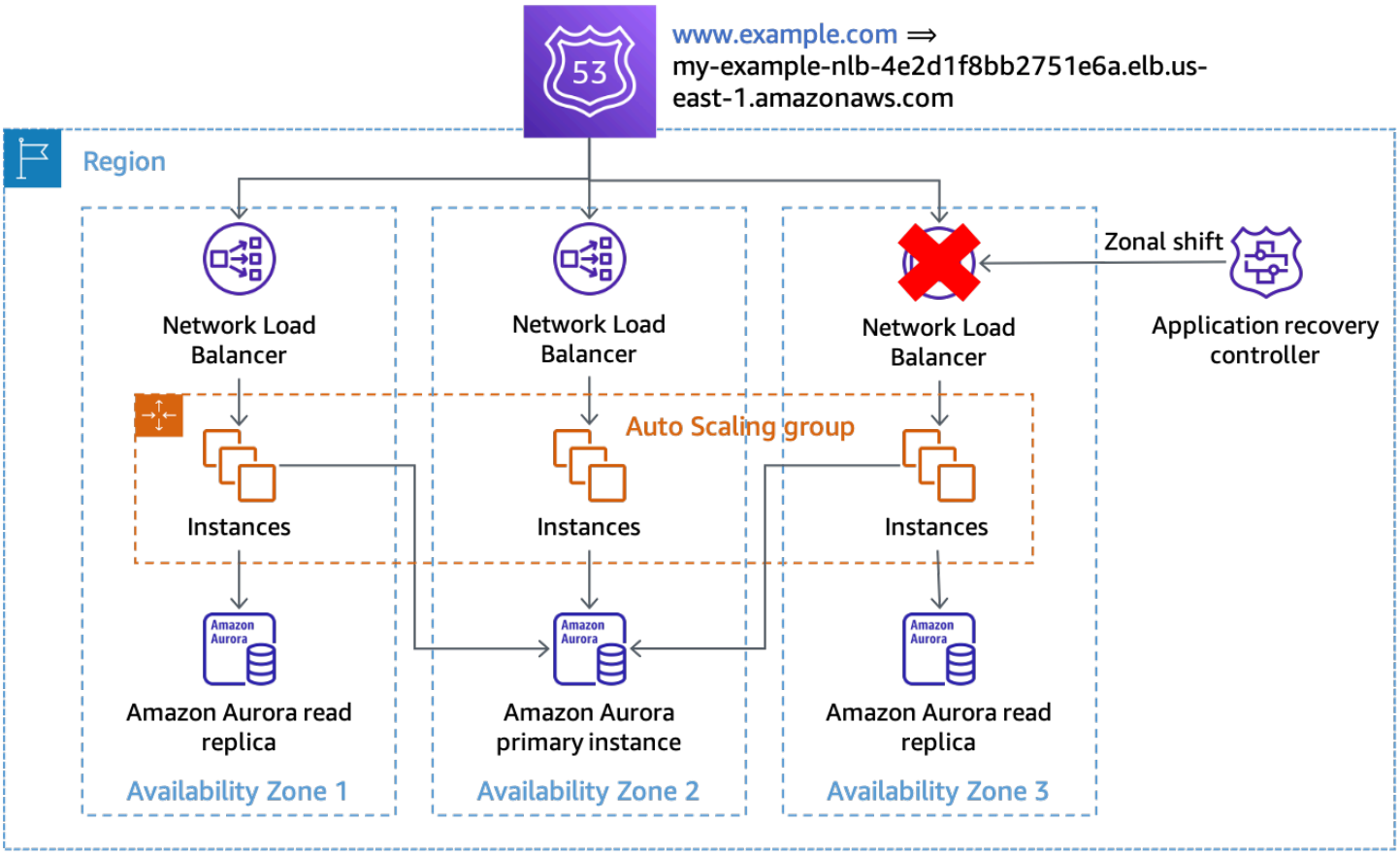
또한 EC2 Auto Scaling을 사용하는 경우 근무 시간 중에 오토 스케일링(ASG)의 스케일 인을 하지 않도록 해야 합니다. 그래야 근무가 끝난 후에도 그룹에서 고객 트래픽을 처리할 수 있는 충분한 용량을 확보할 수 있습니다. ASG의 최소 희망 용량이 현재 고객 부하를 처리할 수 있도록 하면 됩니다. 또한 P90 또는 P99와 같은 특이 백분위수 지표와 달리 지표의 평균을 사용하여 실수로 ASG의 스케일 인을 방지할 수 있습니다.

근무 시간 중에는 더 이상 트래픽을 처리하지 않는 리소스의 사용률이 매우 낮아야 하지만 다른 리소스는 새 트래픽으로 사용률을 높여 평균을 상당히 일정하게 유지하므로 스케일 인 작업을 방지할 수 있습니다. 마지막으로 ALB 및 NLB의 대상 그룹 상태 설정을 사용하여 정상 호스트의 백분율 또는 수로 DNS 장애 조치를 지정할 수도 있습니다. 이렇게 하면 정상 호스트가 충분하지 않은 가용 영역으로 트래픽이 라우팅되는 것을 방지할 수 있습니다.

Route 53 Application Recovery Controller(ARC)의 영역 전환

가용 영역 대피를 위한 첫 번째 솔루션은 Route 53 ARC의 영역 전환을 사용합니다. 이 솔루션은 NLB 또는 ALB를 고객 트래픽의 수신 지점으로 사용하는 요청/응답 워크로드 내에서 사용할 수 있습니다.

가용 영역이 손상된 것을 감지하면 Route 53 ARC를 사용하여 영역 전환을 시작할 수 있습니다. 이 작업이 완료되고 기존의 캐시된 DNS 응답이 만료되면 모든 새 요청은 나머지 가용 영역의 리소스로만 라우팅됩니다. 다음 그림은 영역 전환 작동 방식을 보여 줍니다. 다음 그림에는 `my-example-nlb-4e2d1f8bb2751e6a.elb.us-east-1.amazonaws.com` 지점을 가리키는 `www.example.com`에 관한 Route 53 별칭 레코드가 있습니다. 영역 전환은 가용 영역 3에 대해 수행됩니다.



영역 전환

예제에서 기본 데이터베이스 인스턴스가 가용 영역 3에 없는 경우 영역 전환은 첫 번째 대피 결과를 얻기 위해 필요한 유일한 조치이며, 이는 영향을 받는 가용 영역에서 작업이 처리되지 않도록 합니다. 프라이머리 노드가 가용 영역 3에 있는 경우 Amazon RDS가 아직 자동 장애 조치를 수행하지 않았다면 영역 전환에 맞춰 수동으로 시작된 장애 조치(Amazon RDS 컨트롤 플레인에 의존)를 수행할 수 있습니다. 이는 이 섹션의 모든 데이터 영역 제어 솔루션에 해당됩니다.

대피를 시작하는 데 필요한 종속성을 최소화하려면 CLI 명령 또는 API를 사용하여 영역 전환을 시작해야 합니다. 대피 프로세스가 간단할수록 신뢰성이 높아집니다. 특정 명령은 대기 중인 엔지니어가 쉽게 액세스할 수 있는 로컬 런북에 저장할 수 있습니다. 영역 전환은 가용 영역을 제거할 때 가장 선호되며 간단한 솔루션입니다.

Route 53 ARC

두 번째 솔루션은 Route 53 ARC의 기능을 사용하여 특정 DNS 레코드의 상태를 수동으로 지정합니다. 이 솔루션은 가용성이 높은 Route 53 ARC 클러스터 데이터 영역을 사용하여 최대 두 개의 서로 다른 AWS 리전의 손상에 대한 복원력을 제공한다는 이점을 보입니다. 추가 비용이 드는 단점이 있으며

DNS 레코드를 추가로 구성해야 합니다. 이 패턴을 구현하려면 로드 밸런서(ALB 또는 NLB) 에서 제공하는 가용 영역별 DNS 이름에 대한 별칭 레코드를 만들어야 합니다. 이 내용은 다음 표와 같습니다.

표 3: 로드 밸런서의 영역 DNS 이름에 대해 구성된 Route 53 별칭 레코드

라우팅 정책: 가중치 적용	라우팅 정책: 가중치 적용	라우팅 정책: 가중치 적용
이름: `www.example.com`	이름: `www.example.com`	이름: `www.example.com`
유형: A(별칭)	유형: A(별칭)	유형: A(별칭)
값: `us-east-1b.load-balancer-name.elb.us-east-1.amazonaws.com`	값 `us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com`	값: `us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com`
가중치: `100`	가중치: `100`	가중치: `100`
대상 상태 평가: 참	대상 상태 평가: `true`	대상 상태 평가: `true`

이러한 각 DNS 레코드에 대해 Route 53 ARC 라우팅 컨트롤과 연결된 Route 53 상태 확인을 구성합니다. 가용 영역 대피를 시작하려면 라우팅 제어 상태를 `Off`로 설정하십시오. AWS는 가용 영역 대피를 시작하는 데 필요한 종속성을 최소화하기 위해 CLI 또는 API를 사용하여 이 작업을 수행할 것을 권장합니다. 가장 좋은 방법은 Route 53 ARC 클러스터 엔드포인트의 로컬 사본을 보관하여 대피를 수행해야 할 때 ARC 컨트롤 플레인에서 해당 엔드포인트를 검색할 필요가 없도록 하는 것입니다.

이 접근 방식을 사용할 때 비용을 최소화하기 위해 단일 AWS 계정 내 Route 53 ARC 클러스터와 상태 확인을 한 번에 생성하고 조직 내 다른 AWS 계정와 상태 확인을 공유할 수 있습니다. 이 방법을 사용할 때는 라우팅 제어에 가용 영역 이름(예:`us-east-1a`) 대신 가용 영역 ID(AZ-ID)(예:`use1-az1`)를 사용해야 합니다. AWS에서 물리적 가용 영역은 각각의 AWS 계정 가용 영역 이름에 무작위로 매핑되므로 AZ-ID를 사용하면 동일한 물리적 위치를 일관되게 참조할 수 있습니다. 예를 들어 `use1-az2`에 대해 가용 영역 대피를 시작할 때는 각각 AWS 계정의 Route 53 레코드 세트가 AZ-ID 매핑을 사용하여 각 NLB 레코드에 대해 올바른 상태 확인을 구성하는지 확인해야 합니다.

예를 들어, ID가 `0385ed2d-d65c-4f63-a19b-2412a31ef431`이고 `use1-az2`에 대한 Route 53 ARC 라우팅 제어와 연결된 Route 53 상태 확인이 있다고 가정해 보겠습니다. 이 상태 확인을 사용하려는 다른 AWS 계정에서 `us-east-1c`가 `use1-az2`으로 매핑된 경우, 레코드 `us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com`에 대한 `use1-az2` 상태 확인을 사용해야 합니다. 해당 리소스 레코드 세트와 함께 상태 확인 ID `0385ed2d-d65c-4f63-a19b-2412a31ef431`를 사용할 수 있습니다.

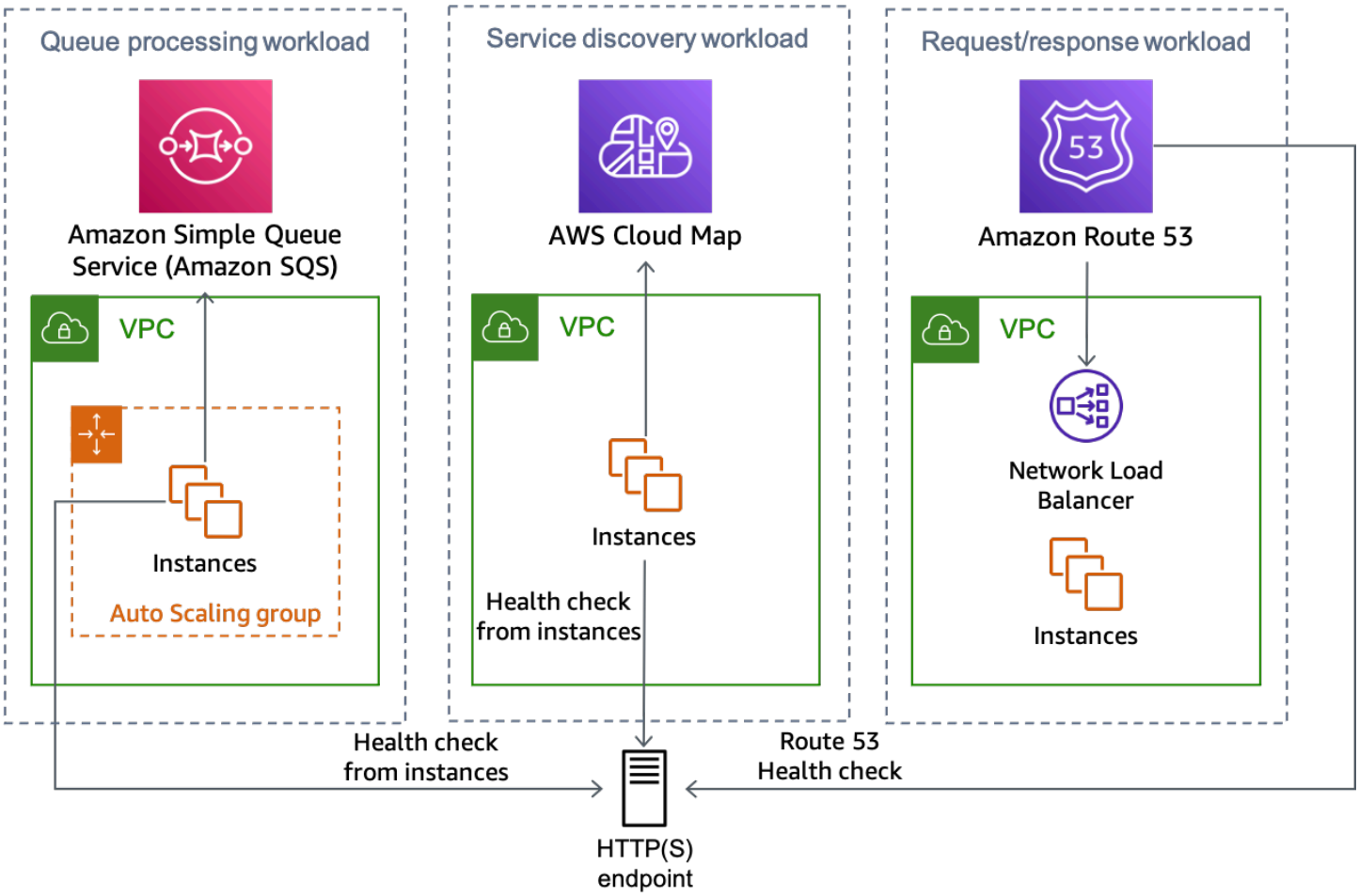
자체 관리형 HTTP 엔드포인트 사용

특정 가용 영역의 상태를 나타내는 자체 HTTP 엔드포인트를 관리하여 이 솔루션을 구현할 수도 있습니다. 이를 통해 HTTP 엔드포인트의 응답을 기반으로 가용 영역이 비정상인 경우를 수동으로 지정할 수 있습니다. 이 솔루션은 Route 53 ARC를 사용하는 것보다 비용이 적게 들지만 영역 전환보다 비용이 많이 들고 추가 인프라 관리가 필요합니다. 이는 다양한 시나리오에서 훨씬 더 유연하다는 이점이 있습니다.

패턴은 NLB 또는 ALB 아키텍처 및 Route 53 상태 확인과 함께 사용할 수 있습니다. 또한 워커 노드가 자체 상태 확인을 수행하는 서비스 검색 또는 대기열 처리 시스템과 같이 비-로드 밸런서 아키텍처에서도 사용할 수 있습니다. 이러한 시나리오에서 호스트는 백그라운드 스레드를 사용하여 주기적으로 AZ-ID를 사용하여 HTTP 엔드포인트에 요청을 보내고(이를 찾는 방법에 대한 자세한 내용은 부록 A — 가용 영역 ID 가져오기 참조) 가용 영역의 상태에 대한 응답을 다시 받을 수 있습니다.

가용 영역이 비정상으로 선언된 경우 가용 영역은 여러 가지 대응 방법을 선택할 수 있습니다. ELB, Route 53과 같은 소스의 외부 상태 점검 또는 서비스 검색 아키텍처의 사용자 지정 상태 확인에 실패하여 해당 서비스가 비정상으로 표시되도록 할 수도 있습니다. 또한 요청을 받으면 즉시 오류로 응답하여 클라이언트가 백오프하고 재시도할 수 있도록 할 수 있습니다. 이벤트 기반 아키텍처에서는 의도적으로 SQS 메시지를 대기열에 반환하는 등 노드가 의도적으로 작업을 처리하지 못할 수 있습니다. 중앙 서비스 스케줄이 특정 호스트에서 작동하는 업무용 라우터 아키텍처에서는 이 패턴을 사용할 수도 있습니다. 라우터는 작업자, 엔드포인트 또는 셀을 선택하기 전에 가용 영역의 상태를 확인할 수 있습니다. AWS Cloud Map를 사용하는 서비스 검색 아키텍처에서, 요청에 필터(예: AZ-ID)를 제공하여 엔드포인트 검색을 할 수 있습니다.

다음 그림은 이 접근 방식을 여러 유형의 워크로드에 사용할 수 있는 방법을 보여줍니다.



여러 워크로드 유형은 모두 HTTP 엔드포인트 솔루션을 사용할 수 있습니다

HTTP 엔드포인트 접근 방식을 구현하는 방법은 여러 가지가 있으며, 그 중 두 가지가 다음에 설명되어 있습니다.

Amazon S3 사용

이 패턴은 다중 리전 재해 복구에 관한 이 블로그 게시물에 처음 소개되었습니다. 가용 영역 대피에도 동일한 패턴을 사용할 수 있습니다.

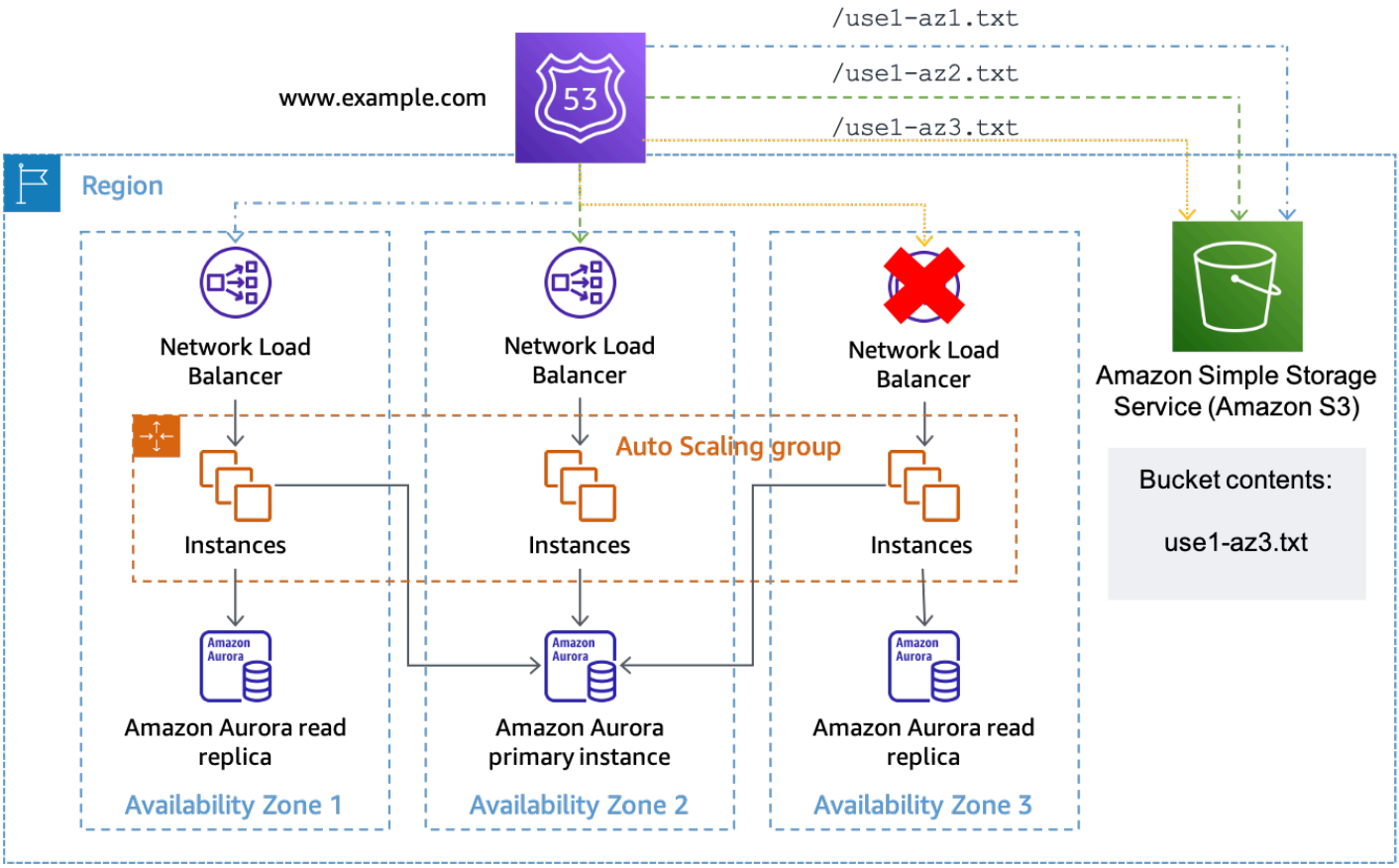
이 시나리오에서는 위의 Route 53 ARC 시나리오와 마찬가지로 각 영역 DNS 레코드에 대해 Route 53 DNS 리소스 레코드 세트를 생성하고 관련 상태 확인을 생성합니다. 그러나 이 구현에서는 상태 확인을 Route 53 ARC 라우팅 제어와 연결하는 대신 HTTP 엔드포인트를 사용하도록 구성되며 실수로 대피를 유발하는 Amazon S3의 장애로부터 보호하기 위해 반전됩니다. 상태 확인은 객체가 없을 때는 정상으로 간주되고 객체가 있을 때는 비정상으로 간주됩니다. 이 설정은 다음 표와 같습니다.

표 4: 가용 영역별 Route 53 상태 확인을 사용하기 위한 DNS 레코드 구성

상태 확인 유형: 엔드포인트 모니터링 프로토콜: HTTPS ID: dddd-4444 URL: https://*bucketname*.s3.us-east-1.amazonaws.com/use1-az1.txt	상태 확인 유형: 엔드포인트 모니터링 프로토콜: HTTPS ID: eeee-5555 URL: https://*bucketname*.s3.us-east-1.amazonaws.com/use1-az3.txt	상태 확인 유형: 엔드포인트 모니터링 프로토콜: HTTPS ID: ffff-6666 URL: https://*bucketname*.s3.us-east-1.amazonaws.com/use1-az2.txt	←	상태 확인
↑	↑	↑		
라우팅 정책: 가중치 적용 이름: www.example.com 유형: A(별칭) 값: us-east-1b.load-balancer-name.elb.us-east-1.amazonaws.com 가중치: 100	라우팅 정책: 가중치 적용 이름: www.example.com 유형: A(별칭) 값: us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com 가중치: 100	라우팅 정책: 가중치 적용 이름: www.example.com 유형: A(별칭) 값: us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com 가중치: 100	←	가중치가 균등한 최상위 별칭 A 레코드는 NLB AZ 전용 엔드포인트를 가리킵니다

대상 상태 평가: true	대상 상태 평가: true	대상 상태 평가: true		

가용 영역 대피를 수행하려는 워크로드가 있는 계정의 가용 영역 `us-east-1a`이 `use1-az3`에 매핑되어 있다고 가정해 보겠습니다. `us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com`에 대하여 생성된 리소스 레코드 세트의 경우 URL `https://bucket-name.s3.us-east-1.amazonaws.com/use1-az3.txt`를 테스트하는 상태 확인을 연결합니다. `use1-az3`에 대한 가용 영역 대피를 시작하려면 CLI 또는 API를 사용하여 `use1-az3.txt`라는 이름이 지정된 파일을 버킷에 업로드합니다. 파일은 콘텐츠를 포함할 필요는 없지만, Route 53 상태 확인이 파일에 액세스할 수 있으려면 공개 파일이어야 합니다. 다음 그림은 이 구현이 대피 `use1-az3`에 사용되는 것을 보여줍니다.



Amazon S3를 Route 53 상태 확인 대상으로 사용

API 게이트웨이 및 DynamoDB 사용

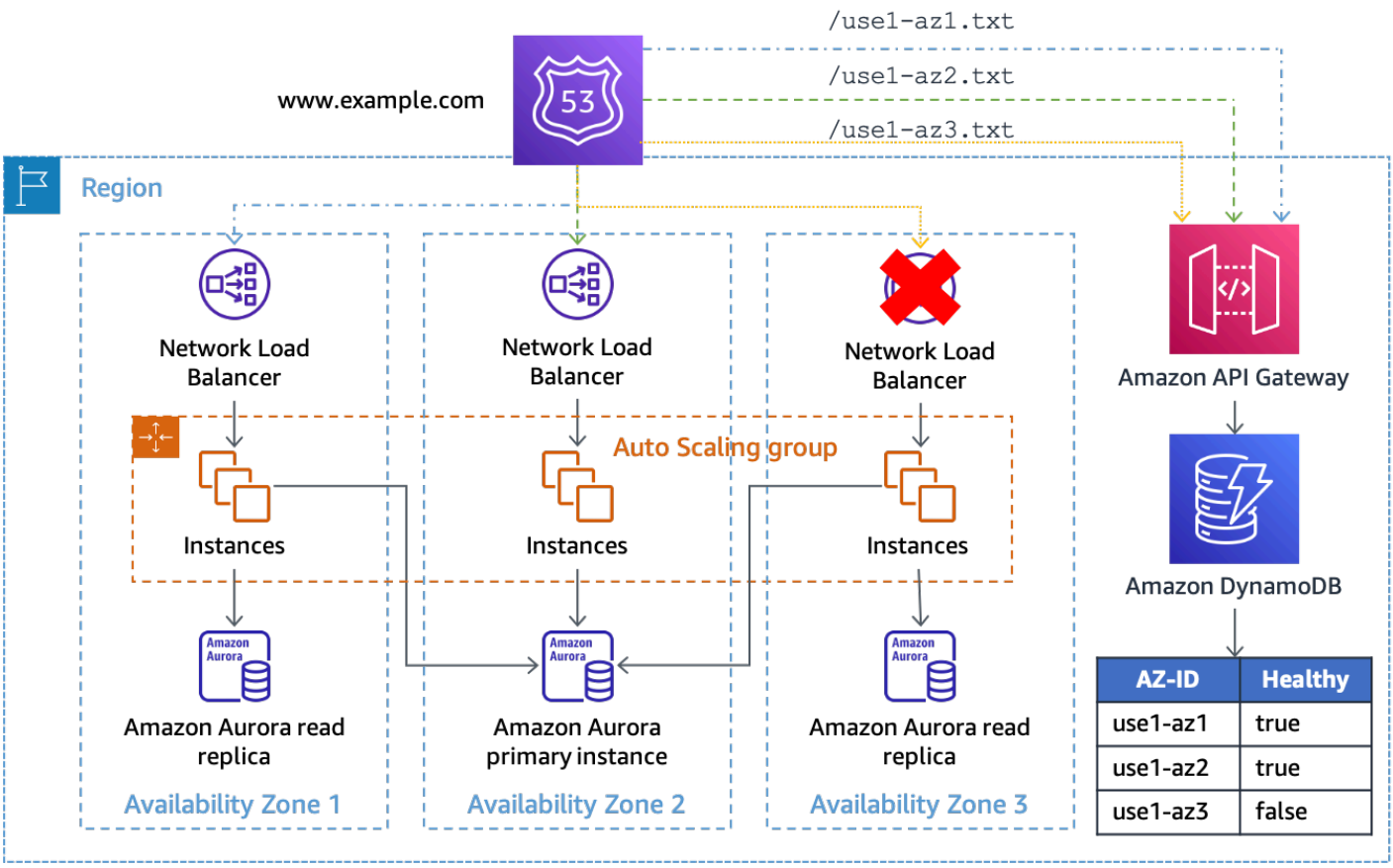
이 패턴의 두 번째 구현에서는 Amazon API Gateway REST API를 사용합니다. API는 사용 중인 각 가용 영역의 상태가 저장되는 Amazon DynamoDB에 대한 서비스 통합을 통해 구성됩니다. 이 구현

은 Amazon S3 접근 방식보다 유연하지만 더 많은 인프라를 구축, 운영 및 모니터링해야 합니다. 또한 Route 53 상태 확인 및 개별 호스트가 수행하는 상태 확인과 함께 사용할 수도 있습니다.

이 솔루션을 NLB 또는 ALB 아키텍처와 함께 사용하는 경우, API 게이트웨이 엔드포인트를 사용하도록 상태 확인 경로를 변경하고 URL 경로 내 `AZ-ID`에 제공한다는 점을 제외하면 위의 Amazon S3 예제와 동일한 방식으로 DNS 레코드를 설정하십시오. 예를 들어, `az-status.example.com`의 사용자 지정 도메인을 API 게이트웨이로 구성한 경우 `use1-az1`에 대한 전체 요청은 `https://az-status.example.com/status/use1-az1`와 같습니다. 가용 영역 제거를 시작하려는 경우 CLI 또는 API를 사용하여 DynamoDB 항목을 생성하거나 업데이트할 수 있습니다. 항목은 `AZ-ID`를 프라이머리 키로 사용하고 API 게이트웨이 응답 방식을 나타내는 데 사용되는 `Healthy`라 불리는 Boolean 속성을 가집니다. 다음은 API 게이트웨이 구성에서 이러한 결정을 내리는 데 사용되는 예제 코드입니다.

```
#set($inputRoot = $input.path('$'))
#if ($inputRoot.Item.Healthy['BOOL'] == (false))
    #set($context.responseOverride.status = 500)
#end
```

속성이 `true`인 경우(또는 존재하지 않는 경우), API 게이트웨이는 HTTP 200으로 상태 확인에 응답하고, 속성이 거짓이면 HTTP 500으로 응답합니다. 방법은 다음 그림과 같습니다.



API 게이트웨이 및 DynamoDB를 Route 53 상태 확인의 대상으로 사용

이 솔루션에서는 DynamoDB 앞에 API 게이트웨이를 사용해야 엔드포인트에 공개적으로 액세스할 수 있을 뿐만 아니라 요청 URL을 DynamoDB `GetItem` 요청으로 조작할 수 있습니다. 이 솔루션은 요청에 추가 데이터를 포함하려는 경우에도 유연성을 제공합니다. 예를 들어 애플리케이션별로 더 세분화된 상태를 생성하려는 경우, 경로 또는 쿼리 문자열에 DynamoDB 항목과도 일치하는 애플리케이션 ID를 제공하도록 상태 확인 URL을 구성할 수 있습니다.

가용 영역 상태 엔드포인트를 중앙에 배포할 수 있으므로 AWS 계정 전체의 여러 상태 확인 리소스가 모두 가용 영역 상태에 대한 동일한 일관된 보기를 사용할 수 있고(API 게이트웨이 REST API 및 DynamoDB 테이블이 부하 처리에 맞게 확장되는지 확인) Route 53 상태 확인을 공유할 필요가 없습니다.

또한 이 솔루션은 Amazon DynamoDB 글로벌 테이블과 각 리전의 API 게이트웨이 REST API 사본을 사용하여 여러 AWS 리전으로 확장할 수도 있습니다. 이렇게 하면 이 솔루션이 단일 리전에 종속되지 않고 가용성이 향상됩니다. 3~5개 리전에 솔루션을 배포하고 대다수 엔드포인트의 결과를 사용하여 가용 영역 상태를 각 리전에 쿼리하여 쿼럼을 보장할 수 있습니다. 이를 통해 최종적으로 글로벌 테이블 전체에 일관되게 업데이트를 복제할 수 있을 뿐만 아니라 엔드포인트의 응답을 방해할 수 있는 장애

도 완화할 수 있습니다. 예를 들어 5개 리전을 사용 중이고 3개 엔드포인트가 가용 영역을 비정상으로 보고하고, 한 엔드포인트는 가용 영역을 정상으로 보고하고, 한 엔드포인트는 응답하지 않는 경우 가용 영역을 비정상으로 처리하도록 선택할 수 있습니다. m of n 계산을 사용하여 Route 53의 계산된 상태 확인을 생성하여 이 로직을 수행하여 가용 영역 상태를 확인할 수도 있습니다.

개별 호스트가 AZ 상태를 확인하는 메커니즘으로 사용할 솔루션을 구축하려는 경우, 대안으로 상태 확인을 위한 풀 메커니즘을 제공하는 대신 푸시 알림을 사용할 수 있습니다. 이를 위한 한 가지 방법은 소비자가 구독하는 SNS 주제를 사용하는 것입니다. 회로 차단기를 트리거하려면 어떤 가용 영역이 손상되었는지 알려주는 메시지를 SNS 주제에 게시하십시오. 이 접근 방식은 전자와 절충점이 있습니다. 이를 통해 API 게이트웨이 인프라를 생성 및 운영하고 용량 관리를 수행할 필요가 없습니다. 또한 가용 영역 상태를 더 빠르게 통합할 수도 있습니다. 임시 쿼리를 수행하는 기능을 제거하고 SNS 전송 재시도 정책 사용하여 각 엔드포인트가 알림을 수신하도록 합니다. 또한 각 워크로드 또는 서비스가 SNS 알림을 수신하고 이에 대한 조치를 취하는 방법을 구축해야 합니다.

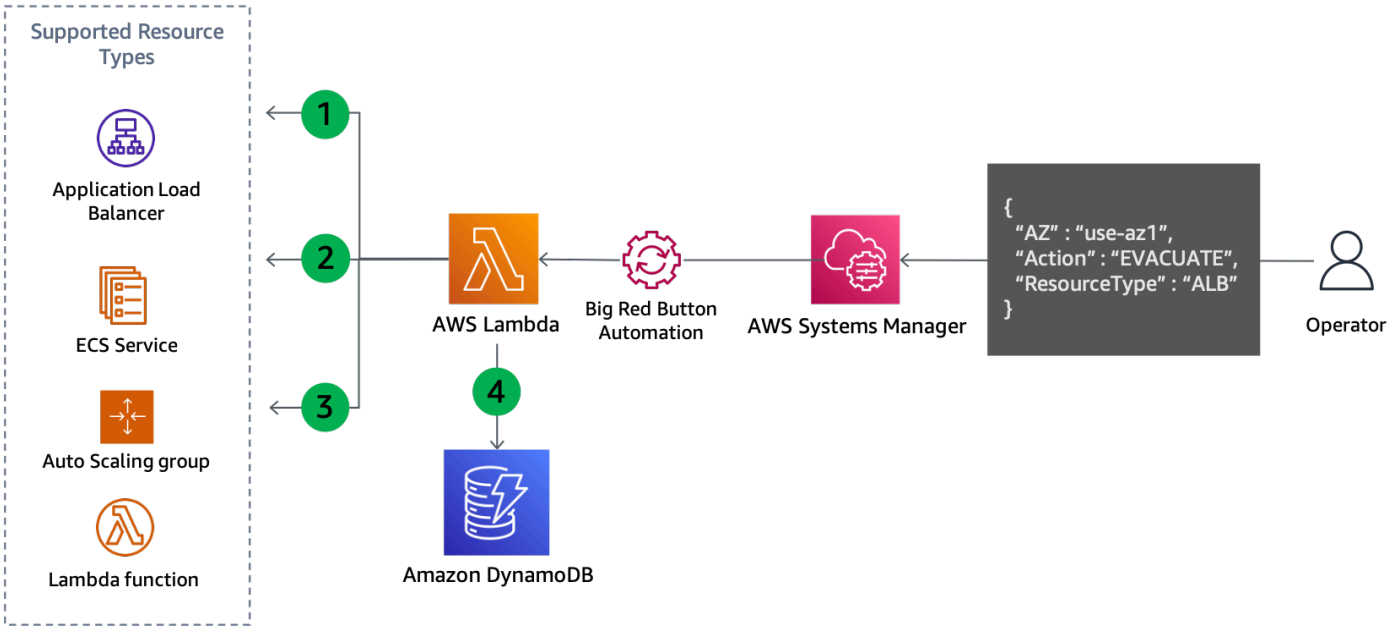
예를 들어, 새로 시작되는 각 EC2 인스턴스 또는 컨테이너는 부트스트랩 중에 HTTP 엔드포인트가 있는 주제를 구독해야 합니다. 그런 다음 각 인스턴스는 알림이 전달되는 이 엔드포인트에서 수신 대기하는 소프트웨어를 구현해야 합니다. 또한 인스턴스가 이벤트의 영향을 받는 경우 푸시 알림을 받지 못하고 작업을 계속할 수 있습니다. 반면 풀 알림을 사용하면 인스턴스가 끌어오기 요청에 실패하는지 알게 되고 이에 대응하여 취해야 할 조치를 선택할 수 있습니다.

푸시 알림을 보내는 두 번째 방법은 수명이 긴 WebSocket 연결을 사용하는 것입니다. Amazon API Gateway는 소비자가 백엔드에서 메시지를 전송할 때 연결하여 메시지를 수신할 수 있는 WebSocket API를 제공하는 데 사용할 수 있습니다. WebSocket을 사용하면 인스턴스가 주기적인 폴링을 수행하여 연결 상태를 유지하고 지연 시간이 짧은 푸시 알림을 수신할 수 있습니다.

컨트롤 플레인-제어식 대피

첫 번째 패턴은 데이터 영역 작업을 사용하여 영향을 받는 가용 영역에서 작업이 수행되지 않도록 방지함으로써 이벤트의 영향을 완화합니다. 하지만 로드 밸런서 미사용 혹은 호스트 개인별 상태 확인을 구성할 수 없는 아키텍처를 사용하고 있을 수 있습니다. 또는 Auto Scaling 혹은 일반적인 작업 스케줄링을 통해 영향을 받는 가용 영역에 새 용량이 배포되는 것을 방지할 수 있습니다.

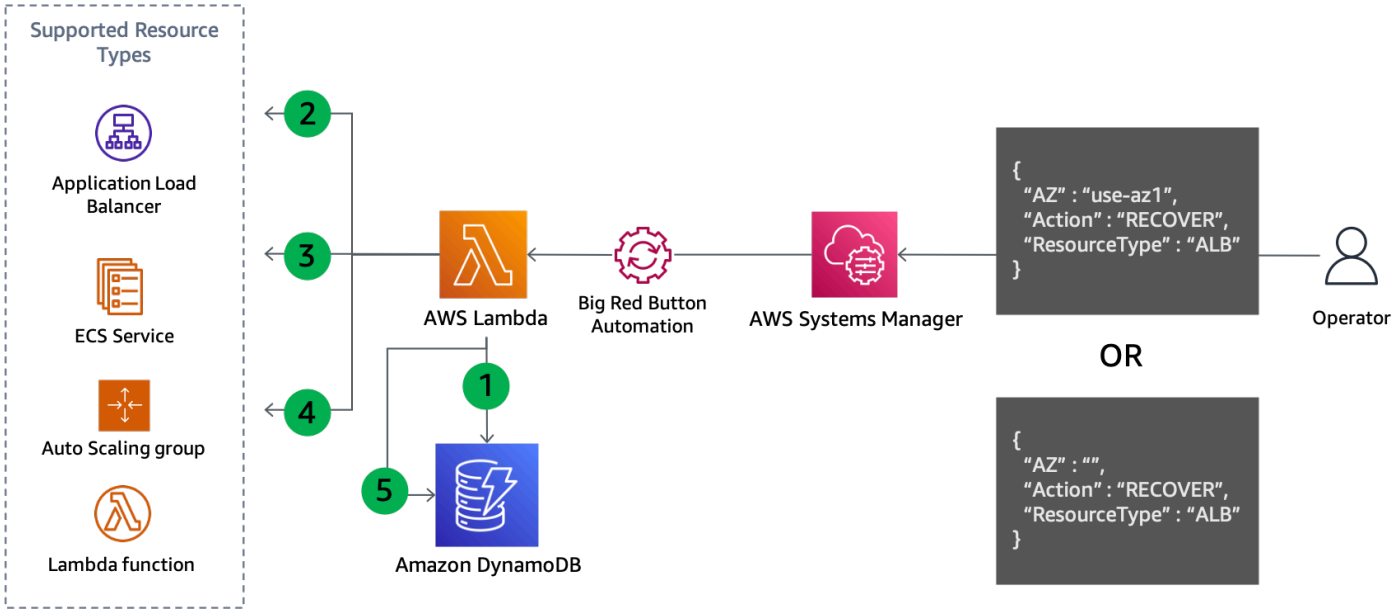
두 상황을 모두 해결하려면 리소스 구성을 업데이트하기 위한 컨트롤 플레인 조치가 필요합니다. 이 패턴은 네트워크 구성을 업데이트할 수 있는 모든 서비스(예: EC2 Auto Scaling, Amazon ECS, Lambda 등)에서 사용할 수 있습니다. 각 서비스에 대한 코드를 작성해야 하지만 비즈니스 로직은 표준 패턴을 따릅니다. 필요한 종속성을 최소화하려면 이벤트에 응답하는 운영자가 로컬에서 코드를 실행해야 합니다. 스크립트 로직의 기본 흐름은 다음 그림과 같습니다.



가용 영역 제거를 위한 컨트롤 플레인 업데이트

1. 스크립트는 오토 스케일링, ECS 서비스 또는 Lambda 함수와 같은 지정된 유형의 모든 리소스를 나열하고 리소스 정보에서 해당 서브넷을 검색합니다. 지원되는 리소스는 스크립트가 지원하도록 구성된 항목에 따라 달라집니다.
2. 각 서브넷의 가용 영역 이름을 입력 파라미터로 제공된 매핑된 가용 영역 ID와 비교하여 제거해야 할 서브넷을 결정합니다.
3. 리소스의 네트워크 구성이 업데이트되어 식별된 서브넷이 제거됩니다.
4. 업데이트의 세부 정보는 DynamoDB 테이블에 기록됩니다. 가용 영역 ID는 파티션 키로 저장되고 리소스 ARN 또는 이름은 정렬 키로 저장됩니다. 제거된 서브넷은 문자열 배열로 저장됩니다. 마지막으로, 리소스 유형도 저장되어 글로벌 보조 인덱스(GSI)의 해시 키로 사용됩니다.

4단계는 업데이트 내용을 기록하므로 다음 그림과 같이 이 접근 방식을 사용하면 복구할 준비가 되었을 때 쉽게 되돌릴 수 있습니다.



가용 영역 제거 시 복구를 위한 컨트롤 플레인 업데이트

복구 단계:

1. GSI를 쿼리하여 지정된 가용 영역(또는 가용 영역이 지정되지 않은 경우 모든 가용 영역)에서 지정된 유형의 각 리소스에 대해 제거된 서브넷을 가져옵니다.
2. DynamoDB 쿼리에서 찾은 각 리소스를 설명하여 현재 네트워크 구성을 가져오십시오.
3. 현재 네트워크 구성의 서브넷을 DynamoDB 쿼리에서 검색된 서브넷과 결합합니다.
4. 리소스의 네트워크 구성을 새 서브넷 세트로 업데이트하십시오.
5. 업데이트가 성공적으로 완료된 후 DynamoDB 테이블에서 레코드를 제거합니다.

이 일반화된 패턴은 영향을 받는 가용 영역으로의 라우팅 작업을 방해하고 새 용량이 해당 가용 영역에 배포되는 것을 방지합니다. 다음은 여러 서비스에서 이 작업을 수행하는 방법의 예시입니다.

- Lambda — 함수의 VPC 구성을 업데이트하여 지정된 가용 영역에서 서브넷을 제거합니다.
- 오토 스케일링 — ASG 구성에서 서브넷을 제거하여 나머지 가용 영역의 해당 용량을 대체합니다.
- Amazon ECS — ECS 서비스 VPC 구성을 업데이트하여 서브넷을 제거합니다.
- Amazon EKS — 영향을 받는 가용 영역의 노드에 테인트를 적용하여 기존 포드를 제거하고 추가 포드가 예약되지 않도록 합니다.

각 서비스는 구성 업데이트에 다르게 반응합니다. 예를 들어, Amazon ECS는 업데이트 후 서비스의 배포 구성을 따라 새 작업의 롤링 배포 또는 블루/그린 배포를 트리거합니다.

이러한 업데이트로 인해 일부 워크로드의 경우 정상 가용 영역으로 작업이 너무 빨리 전환될 수 있습니다. 장애 발생 시 정적으로 안정적이도록 구성되어 있지만(영향을 받는 가용 영역의 작업을 처리할 수 있을 만큼 나머지 가용 영역에 충분한 용량이 미리 프로비저닝되어 있음), 영향을 받는 가용 영역의 용량을 단계적으로 줄이는 것도 좋습니다.

영역 간 로드 밸런싱이 비활성화된 로드 밸런서 대상 그룹인 오토 스케일링의 네트워크 구성을 업데이트하려는 경우 이 지침을 따르세요.

Auto Scaling은 가용 영역 재조정 로직을 사용하여 이러한 변화에 대응합니다. 원하는 용량을 채울 수 있도록 다른 가용 영역의 인스턴스를 시작하고 제거한 가용 영역의 인스턴스를 종료합니다. 하지만 로드 밸런서 내에서는 인스턴스가 종료되는 동안에도 ASG에서 제거한 가용 영역을 포함하여 각 가용 영역에 걸쳐 트래픽을 균등하게 분배합니다. 이로 인해 모든 인스턴스가 성공적으로 종료될 때까지 해당 가용 영역의 남은 용량이 부족해질 수 있습니다. 이는 영역 간 부하 분산이 비활성화된 경우의 가용 영역 불균형과 관련된 가용 영역 독립성에 설명된 것과 동일한 문제입니다. 이러한 문제가 발생하지 않도록 하려면 다음 중 하나를 수행하십시오.

- 트래픽이 나머지 가용 영역 사이에서만 분리되도록 항상 가용 영역 대피를 먼저 수행하십시오.
- 해당 가용 영역에 필요한 최소 대상 수와 일치하도록 DNS 장애 조치와 함께 최소 정상 대상 수를 지정하십시오.

이렇게 하면 인스턴스가 종료되기 시작한 후 제거한 가용 영역으로 트래픽이 전송되지 않도록 할 수 있습니다.

요약

다음 표에 설명된 대피 패턴의 장단점이 요약되어 있습니다.

표 5: 대피 패턴의 장단점

접근 방식	장점	단점
데이터 영역에서 제어하는 대피	데이터 영역 작업에만 의존 영향을 받는 가용 영역에서 작업이 수행되지 않도록 신속하게 방지	영향을 받는 가용 영역에 용량이 배포되는 것을 막지 않음

접근 방식	장점	단점
	가용 영역 상태를 중앙 집중식으로 볼 수 있는 유연한 접근 방식	모든 워크로드 유형에서 이 접근 방식을 쉽게 사용할 수 있는 것은 아님
제어 플레인 제어식 대피	영향을 받는 가용 영역에 새 용량이 배포되는 것을 방지 영향을 받는 가용 영역에서 기존 용량을 제거	각 서비스의 제어 플레인에 의존 각 서비스에 대해 코드 작성 필요 서비스별로 서비스를 완료해야 함 업데이트 중에 용량이 과부하되지 않도록 주의 필요

가용 영역 대피 계획의 일환으로 두 접근 방식을 함께 사용할 가능성이 높습니다. 영향을 받는 가용 영역에서 처리 작업을 빠르게 중단하려면 성공할 가능성이 더 높은 데이터 영역 내에서 제어되는 대피 조치부터 시작하십시오. 그런 다음 초기 영향이 완화되면 필요하다고 판단되는 경우 컨트롤 플레인 내에서 제어하는 대피 조치로 후속 조치를 취하십시오.

결론

이 백서에서는 회색 장애의 개요와 매니페스트 방식을 설명하고, 이러한 유형의 이벤트가 발생할 경우 이를 완화하기 위해 관찰성 및 대피 도구를 구축해야 하는 이유를 설명했습니다. 다음 섹션에서는 다중 AZ 관찰성과 단일 가용 영역의 영향을 탐지하기 위해 구현할 수 있는 세 가지 접근 방식을 검토했습니다. 마지막 섹션에서 이 백서는 가용 영역 제거를 수행하기 위한 두 가지 일반적인 접근 방식을 제시했습니다. 첫 번째 접근 방식은 데이터 영역 작업을 사용하여 작업이 영향을 받는 가용 영역으로 라우팅되는 것을 방지하는 반면, 두 번째 접근 방식은 컨트롤 플레인 작업을 사용하여 영향을 받는 가용 영역에 용량이 프로비저닝되지 않도록 합니다. 이 두 가지 접근 방식을 함께 사용하면 가용 영역 제거가 의도하는 두 가지 결과를 얻을 수 있습니다.

이 백서에 설명된 복구 패턴은 대규모 모니터링 및 장애 복구 솔루션의 일부일 가능성이 높습니다. 단일 가용 영역 회색 장애를 처리하는 이러한 접근 방식을 사용하려면 이를 탐지하는 데 필요한 계측기와 이에 대응하는 도구를 구축하는 엔지니어링 작업이 필요합니다. 그러나 많은 워크로드의 경우 이 접근 방식이 다중 리전 아키텍처를 구축하는 것보다 더 간단하고 비용이 적게 드는 대안이 될 수 있습니다. 또한 다중 리전 DR에 비해 RPO 및 RTO를 더 적게 달성하여 워크로드의 가용성을 높이는 데도 도움이 될 수 있습니다.

부록 A — 가용 영역 ID 가져오기

AWS .NET SDK(JavaScript와 같은 다른 SDK 포함)를 사용하거나 EC2 인스턴스(Amazon ECS 및 Amazon EKS 포함)에서 시스템을 실행하는 경우 가용 영역 ID를 직접 가져올 수 있습니다.

- AWS .NET SDK

```
Amazon.Util.EC2InstanceMetadata.GetData(“/placement/availability-zone-id”)
```

- EC2 인스턴스 메타데이터 서비스

```
curl http://169.254.169.254/latest/meta-data/placement/availability-zone-id
```

Lambda 및 Fargate와 같은 다른 플랫폼에서는 가용 영역 이름을 검색한 다음 가용 영역 ID에 대한 매핑을 찾아야 합니다. 가용 영역 이름을 사용하여 다음과 같이 가용 영역 ID를 찾을 수 있습니다.

```
aws ec2 describe-availability-zones --zone-names $AZ --output json
    --query ‘AvailabilityZones[0].ZoneId’
```

위 예제에서 사용할 가용 영역 이름을 찾기 위한 다음 예시는 bash에서 AWS CLI 및 jq 패키지를 사용하여 작성한 것입니다. 워크로드에 사용되는 프로그래밍 언어로 변환해야 합니다.

- Amazon ECS - 호스트가 인스턴스 메타데이터 서비스(IMDS)를 차단한 경우 컨테이너 메타데이터 파일을 대신 사용할 수 있습니다.

```
AZ=$(cat $ECS_CONTAINER_METADATA_FILE | jq --raw-output
    .AvailabilityZone)
```

- Fargate(플랫폼 버전 1.4 이상)

```
AZ=$(curl $ECS_CONTAINER_METADATA_URI_V4/task | jq --raw-output
    .AvailabilityZone)
```

- Lambda — 가용 영역은 함수에 직접 노출되지 않습니다. 찾으려면 몇 단계를 완료해야 합니다. 이렇게 하려면 요청자의 IP 주소를 반환하는 프라이빗 API 게이트웨이 REST 엔드포인트를 구축해

야 합니다. 이렇게 하면 함수에서 사용 중인 탄력적 네트워크 인터페이스에 할당된 사설 IP가 식별됩니다.

- Lambda `GetFunction` API를 직접적으로 호출하여 함수의 VPC ID를 찾습니다.
- API 게이트웨이 서비스를 직접적으로 호출하여 함수의 IP를 가져옵니다.
- IP 및 VPC ID를 사용하여 연결된 네트워크 인터페이스를 찾고 가용 영역을 추출합니다.

```
VPC_ID=$(aws lambda get-function --function-name $ AWS_LAMBDA_FUNCTION_NAME --
region $AWS_REGION --output json --query 'Configuration.VpcConfig.VpcId')

MY_IP=$(curl http://whats-my-private-ip.internal)

AZ=$(aws ec2 describe-network-interfaces --filters Name=private-ip-address,Values=
$MY_IP Name=vpc-id,Values=$VPC_ID --region $AWS_REGION --output json –query
 'NetworkInterfaces[0].AvailabilityZone')
```

부록 B — 카이 제곱 계산의 예

다음은 오류 지표를 수집하고 데이터에 대해 카이 제곱 검정을 수행하는 예제입니다. 코드는 프로덕션 준비나 필요한 오류 처리를 수행하지는 않지만 로직 작동 방식에 대한 개념 증명을 제공합니다. 필요에 맞게 이 예제를 업데이트해야 합니다.

먼저 Amazon EventBridge의 예약된 이벤트에 의해 1분마다 Lambda 함수를 간접적으로 호출합니다. 이벤트 콘텐츠는 다음 데이터로 구성됩니다.

```
{
  "timestamp": "2023-03-15T15:26:37.527Z",
  "namespace": "multi-az/frontend",
  "metricName": "5xx",
  "dimensions": [
    { "Name": "Region", "Value": "us-east-1" },
    { "Name": "Controller", "Value": "Home" },
    { "Name": "Action", "Value": "Index" }
  ],
  "period": 60,
  "stat": "Sum",
  "unit": "Count",
  "chiSquareMetricName": "multi-az/chi-squared",
  "azs": [ "use1-az2", "use1-az4", "use1-az6" ]
}
```

이 데이터는 적절한 CloudWatch 지표(예: 네임스페이스, 지표 이름, 차원)를 검색하는 데 필요한 공통 데이터를 지정한 다음 각 가용 영역에 대한 카이 제곱 결과를 게시하는 데 사용됩니다. Lambda 함수의 코드는 Python 3.9를 사용하여 다음과 같이 표시됩니다. 상위 수준에서 이전 1분간 지정된 CloudWatch 측정치를 수집하고, 해당 데이터에 대해 카이 제곱 테스트를 실행한 다음, 지정된 각 가용 영역에 대한 테스트 결과에 대한 CloudWatch 측정치를 게시합니다.

```
import os
import boto3
import datetime
import copy
import json
from datetime import timedelta
from scipy.stats import chisquare
from aws_embedded_metrics import metric_scope
```

```
cw_client = boto3.client("cloudwatch", os.environ.get("AWS_REGION", "us-east-1"))

@metric_scope
def handler(event, context, metrics):
    metrics.set_property("Event", json.loads(json.dumps(event, default = str)))
    time = datetime.datetime.strptime(event["timestamp"], "%Y-%m-%dT%H:%M:%S.%fZ")

    # Round down to the previous minute
    end: datetime = roundTime(time)

    # Subtract a minute for the start
    start: datetime = end - timedelta(minutes = 1)

    # Get all the metrics that match the query
    results = get_all_metrics(event, start, end, metrics)
    metrics.set_property("MetricCounts", results)

    # Calculate the chi squared result
    chi_sq_result = chisquare(list(results.values()))
    expected = sum(list(results.values())) / len(results.values())
    metrics.set_property("ChiSquaredResult", chi_sq_result)

    # Put the chi square metrics into CloudWatch
    put_all_metrics(event, results, chi_sq_result[1], expected, start, metrics)

def get_all_metrics(detail: dict, start: datetime, end: datetime, metrics):
    """
    Gets all of the error metrics for each AZ specified
    """
    metric_query = {
        "MetricDataQueries": [
        ],
        "StartTime": start,
        "EndTime": end
    }

    for az in detail["azs"]:

        dim = copy.deepcopy(detail["dimensions"])
        dim.append({"Name": "AZ-ID", "Value": az})

        query = {
            "Id": az.replace("-", "_"),
            "MetricStat": {
```

```
                "Metric": {
                    "Namespace": detail["namespace"],
                    "MetricName": detail["metricName"],
                    "Dimensions": dim
                },
                "Period": int(detail["period"]),
                "Stat": detail["stat"],
                "Unit": detail["unit"]
            },
            "Label": az,
            "ReturnData": True
        }

        metric_query["MetricDataQueries"].append(query)

   metrics.set_property("GetMetricRequest", json.loads(json.dumps(metric_query,
default=str)))
   next_token: str = None
   results = {}

   while True:
       if next_token is not None:
           metric_query["NextToken"] = next_token

       data = cw_client.get_metric_data(**metric_query)

       if next_token is not None:
           metrics.set_property("GetMetricResult::" + next_token,
json.loads(json.dumps(data, default = str)))
       else:
           metrics.set_property("GetMetricResult", json.loads(json.dumps(data, default
= str)))

       for item in data["MetricDataResults"]:
           key = item["Id"].replace("_", "-")
           if key not in results:
             results[key] = 0

           results[key] += sum(item["Values"])

       if "NextToken" in data:
           next_token = data["NextToken"]

       if next_token is None:
```

```
            break

    return results

def put_all_metrics(detail: dict, results: dict, chi_sq_value: float, expected: float,
 timestamp: datetime, metrics):
    """
    Adds the chi squared metric for all AZs to CloudWatch
    """
    farthest_from_expected = None
    if len(results) > 0:
        keys = list(results.keys())
        farthest_from_expected = keys[0]

        for key in keys:
            if abs(results[key] - expected) > abs(results[farthest_from_expected] -
 expected):
                farthest_from_expected = key

    metric_query = {
        "Namespace": detail["namespace"],
        "MetricData": []
    }

    for az in detail["azs"]:
        dim = copy.deepcopy(detail["dimensions"])
        dim.append({"Name": "AZ-ID", "Value": az})

        query = {
            "MetricName": detail["chiSquareMetricName"],
            "Dimensions": dim,
            "Timestamp": timestamp,
        }

        if chi_sq_value <= 0.05 and az == farthest_from_expected:
            query["Value"] = 1
        else:
            query["Value"] = 0

        metric_query["MetricData"].append(query)

    metrics.set_property("PutMetricRequest", json.loads(json.dumps(metric_query,
 default = str)))
```

```
    cw_client.put_metric_data(**metric_query)

def roundTime(dt=None, roundTo=60):
   """Round a datetime object to any time lapse in seconds
   dt : datetime.datetime object, default now.
   roundTo : Closest number of seconds to round to, default 1 minute.
   """
   if dt == None : dt = datetime.datetime.now()
   seconds = (dt.replace(tzinfo=None) - dt.min).seconds
   rounding = (seconds+roundTo/2) // roundTo * roundTo
   return dt + datetime.timedelta(0,rounding-seconds,-dt.microsecond)
```

그런 다음 AZ별로 경보를 생성할 수 있습니다. 다음 예제는 최대값이 1인 연속 1분 데이터 요소 3개에 대한 use1-az2 신호와 경보를 보여줍니다 (1은 카이 제곱 검정을 통해 오류율의 통계적으로 유의한 편차가 확인될 때 게시되는 지표).

```
{
    "Type": "AWS::CloudWatch::Alarm",
    "Properties": {
        "AlarmName": "use1-az2-chi-squared",
        "ActionsEnabled": true,
        "OKActions": [],
        "AlarmActions": [],
        "InsufficientDataActions": [],
        "MetricName": "multi-az/chi-squared",
        "Namespace": "multi-az/frontend",
        "Statistic": "Maximum",
        "Dimensions": [
            {
                "Name": "AZ-ID",
                "Value": "use1-az2"
            },
            {
                "Name": "Action",
                "Value": "Index"
            },
            {
                "Name": "Region",
                "Value": "us-east-1"
            },
            {
```

```
                "Name": "Controller",
                "Value": "Home"
            }
        ],
        "Period": 60,
        "EvaluationPeriods": 3,
        "DatapointsToAlarm": 3,
        "Threshold": 1,
        "ComparisonOperator": "GreaterThanOrEqualToThreshold",
        "TreatMissingData": "missing"
    }
}
```

m-of-n 경보를 만들고 이 두 경보를 복합 경보와 결합할 수도 있습니다. 또한 각 가용 영역에 있는 컨트롤러/액션 조합 또는 마이크로서비스에 대해 동일한 경보를 생성해야 합니다. 마지막으로, 이상치 감지를 사용한 장애 감지에서와 같이 각 컨트롤러/작업 조합에 대한 가용 영역별 경보에 카이 제곱 복합 경보를 추가할 수 있습니다.

기여자

다음은 이 문서의 기여자입니다.

- Michael Haken, Amazon Web Services 주요 솔루션 아키텍트

문서 수정

이 백서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

변경 사항	설명	날짜
백서 업데이트	추가 관찰성 지침과 새로운 영역 전환 특징을 사용하도록 업데이트되었습니다.	2023년 7월 11일
최초 게시	백서가 처음 게시되었습니다.	2022년 3월 2일

> **Note**
> RSS 업데이트를 구독하려면 사용 중인 브라우저에서 RSS 플러그인을 활성화해야 합니다.

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.

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조의 AWS 용어집을 참조하세요.

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