안정성 원칙

AWS Well-Architected 프레임워크

2020년 4월

This paper has been archived.

The latest version is now available at:

https://docs.aws.amazon.com/ko_kr/wellarchitected/latest/reliability-pillar/welcome.html

고지 사항

고객은 본 문서에 포함된 정보를 독립적으로 평가할 책임이 있습니다. 본 문서는 (a) 정보 제공만을 위한 것이며, (b)사전 고지 없이 변경될 수 있는 현재의 AWS 제품 제공 서비스 및 사례를 보여 주며, (c) AWS 및 자회사, 공급업체 또는 라이선스 제공자로부터 어떠한 약정 또는 보증도 하지 않습니다. AWS 제품 또는 서비스는 명시적이든 묵시적이든 어떠한 종류의 보증, 진술 또는 조건 없이 "있는 그대로" 제공됩니다. 고객에 대한 AWS의 책임과 법적 책임은 AWS 계약서에 준하며 본 문서는 AWS와 고객 간의 계약에 포함되지 않고 계약을 변경하지도 않습니다.

목차

개요

AWS Well-Architected 프레임워크의 안정성 원칙에 중점을 두는 이 백서에서는 Amazon Web Services(AWS) 환경의 설계, 제공 및 유지 관리에 모범 사례를 적용할 때 참조할 수 있는 지침을 제공합니다.

서문

AWS Well-Architected 프레임워크는 AWS에서 워크로드를 구축할 때 내리는 의사 결정의 장점과 단점을 이해하는 데 도움이 됩니다. 이 프레임워크를 사용하면 클라우드에서 안정적이고 안전하며 효율적이고 경제적인 워크로드를 설계하고 운영하기 위한 설계 모범 사례를 알아볼 수 있습니다. 또한 모범 사례를 기준으로 아키텍처를 일관적으로 측정하고 개선할 영역을 식별할 수 있습니다. Well-Architected 워크로드를 갖추면 비즈니스 성공 가능성이 높아집니다.

AWS Well-Architected 프레임워크는 다음의 5가지 원칙을 기반으로 합니다.

- 운영 우수성
- 보안
- 안정성
- 성능 효율성
- 비용 최적화

이 백서에서는 안정성 원칙과 이 원칙을 솔루션에 적용하는 방법에 대해 중점적으로 설명합니다. 기존의 온프레미스 환경에서는 단일 실패 지점, 자동화 기능과 탄력성의 부재로 인해 안정성을 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 CTO(최고 기술 책임자), 아키텍트, 개발자, 운영 팀 팀원 등 기술 업무 담당자를 위해 작성되었습니다. 이 백서의 내용을 확인하고 나면 안정성이 뛰어난 클라우드 아키텍처를 설계할 때 사용할 AWS 모범 사례와 전략을 파악할 수 있습니다. 이 백서에는 대략적인 구현 세부 정보와 아키텍처 패턴 및 추가 리소스 참조가 포함되어 있습니다.

안정성

안정성 원칙에서는 워크로드의 의도한 기능이 수행될 것으로 예상되는 시기에 이 기능을 정확하고 일관되게 수행하는 역량에 대해 다룹니다. 여기에는 총 수명 주기에 걸쳐 워크로드를 운영 및 테스트할 수 있는 기능이 포함됩니다. 이 백서는 AWS에서 안정적인 워크로드를 구현하기 위한 세부적인 모범 사례 지침을 제공합니다.

설계 원칙

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

- **장애 자동 복구:** 워크로드의 KPI(핵심 성능 지표)를 모니터링하면 임계값이 위반될 때 자동화를 트리거할 수 있습니다. 이러한 KPI는 서비스 작동의 기술적 측면이 아닌 비즈니스 가치를 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 테스트는 복구 전략을 검증하는 데 사용되지 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 크기 조정을 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 불필요:** 워크로드에 대한 수요가 해당 워크로드의 용량을 넘어서는 리소스 포화 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(서비스 할당량 및 제약 조건 관리 참조).
- **자동화 변경 관리**: 인프라 변경은 자동화를 통해 수행되어야 합니다. 자동화 변경과 같은 변경을 관리해야 하며 이후에 이러한 변경을 추적하고 검토할 수 있습니다.

정의

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

- 기반
- 워크로드 아키텍처
- 변경 관리
- 장애 관리

안정성을 달성하려면 기반에서 시작해야 합니다. 이 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.

복원력과 안정성의 구성 요소

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

- **복원력**에는 인프라나 서비스의 시스템 장애를 복구하고, 컴퓨팅 리소스를 동적으로 확보하여 수요에 대응하거나, 구성 오류나 일시적 네트워크 문제와 같은 장애를 완화하는 워크로드의 기능이 포함됩니다.

워크로드 안정성에 영향을 미치는 다른 요인은 다음과 같습니다.

- 운영 우수성: 변경 자동화, 장애 대응을 위한 플레이북 사용, 애플리케이션의 프로덕션 운영 준비 상태를 확인하는 ORR(운영 준비 검토)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 크기 조정을 사용할지 여부와 같은 절충이 포함됩니다.

이 백서에서는 복원력을 중점적으로 다룹니다.

마찬가지로 중요한 다른 4가지 요소는 AWS Well-Architected 프레임워크의 해당하는 원칙에서 다룹니다. 모범 사례를 설명할 때 이 4가지 요소가 나오기는 하지만 이 백서의 중점은 복원력에 있습니다.

가용성

가용성(서비스 가용성이라고도 함)은 안정성을 양적으로 측정할 때 자주 사용되는 지표입니다.

- **가용성**은 워크로드를 사용할 수 있는 시간의 비율입니다.

사용할 수 있다는 말은 필요할 때 합의된 기능을 수행한다는 것을 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. 가용성을 정의할 때는 다음과 같은 기준이 사용됩니다.

- 가용성 = $\frac{\text{가용 시간}}{\text{총 시간}}$
- 일정 기간(대개 1년)의 가동 시간 백분율(예: 99.9%)

- "숫자 9의 수"만 나타내는 일반적인 약어(예: "5개의 9"는 99.999%의 가용성을 의미함)
- 첫 번째 글머리 기호에 있는 수식의 총 시간에서 예정된 서비스 가동 중지 시간(예: 계획된 유지 관리)을 제외하는 경우도 있지만 실제로 이 시간에도 서비스 사용을 원하는 사용자가 있을 수 있으므로 이는 잘못된 선택인 경우가 많습니다.

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
99.95%	4시간 22분	온라인 상거래, POS(Point of Sale)
99.99%	52분	비디오 전송, 브로드캐스트 워크로드
99.999%	5분	ATM 트랜잭션, 통신 워크로드

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

$$\text{Avail}_{\text{workload}} = \text{Avail}_{\text{invok}} \times \text{Avail}_{\text{dep1}} \times \text{Avail}_{\text{dep2}}$$

$$99.99\% \times 99.99\% \times 99.99\% = 99.97\%$$

따라서 가용성을 계산할 때는 종속성과 가용성 설계 목표를 파악해야 합니다.

중복 구성 요소를 사용한 가용성 계산. 시스템에서 중복 가용 영역 등의 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 해당 시스템의 가용성은 99.9999%입니다.

$$\text{Avail}_{\text{workload}} = \text{Avail}_{\text{MAX}} - \left(\left(100\% - \text{Avail}_{\text{dependency}}\right) \times \left(100\% - \text{Avail}_{\text{dependency}}\right)\right)$$

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

그런데 종속 구성 요소의 가용성을 알 수 없는 경우도 있습니다.

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

$$\text{Avail}_{\text{EST}} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

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

자세한 내용은 가용성 계산에 도움이 될 수 있는 이 문서(Calculating Total System Availability)를 참조하십시오.

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 테스트해야 합니다. 예를 들어 원하는 가용성 목표를 달성할 수 있도록 용량 추가/제거, 업데이트된 소프트웨어 또는 구성 변경 사항 배포/롤백, 시스템 데이터 마이그레이션 등을 수행해야 합니다. 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.

RPO(복구 시점 목표)와 RTO(복구 시간 목표)

이러한 용어는 재해 발생 시 워크로드 가용성을 복구하기 위한 목표 및 전략 세트인 DR(재해 복구)과 가장 자주 관련됩니다.

RTO(복구 시간 목표)는 워크로드의 구성 요소가 복구 단계에 있어 조직의 미션 또는 미션/비즈니스 프로세스에 부정적인 영향을 미치기 전에는 사용할 수 없는 시간의 전체 길이를 나타냅니다.

RPO(복구 시점 목표)는 조직의 미션 또는 미션/비즈니스 프로세스에 부정적인 영향을 미치기 전에 워크로드의 데이터를 사용할 수 없는 시간의 전체 길이를 나타냅니다.

가용성이 워크로드 데이터에 따라 달라지는 가장 일반적인 경우에 RPO는 RTO보다 작아야 합니다.

RTO는 중단이 시작된 시점과 워크로드 복구 사이의 시간을 측정한다는 점에서 MTTR(평균 복구 시간)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 있고, 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 설계 목표를 구분합니다.

AWS에서 서비스는 주로 "데이터 영역"과 "제어 영역"으로 구분됩니다. 데이터 영역에서는 실시간 서비스를 제공하고, 제어 영역은 환경을 구성하는 데 사용됩니다. 예를 들어 Amazon EC2 인스턴스, Amazon RDS 데이터베이스, Amazon DynamoDB 테이블 읽기/쓰기 작업은 모두 데이터 영역 작업입니다. 반면 새 EC2 인스턴스나 RDS 데이터베이스 시작 또는 DynamoDB의 테이블 메타데이터 추가/변경은 모두 제어 영역 작업으로 간주됩니다. 이러한 모든 기능에는 높은 수준의 가용성이 중요하지만, 일반적으로는 데이터 영역의 가용성 설계 목표가 제어 영역보다 높습니다.

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 실행해 시스템을 엔지니어링합니다. AWS에서는 가용성이 99.999% 이상인 서비스를 포함하여 광범위한 가용성 설계 목표가 적용되는 많은 환경 엔지니어링 애플리케이션이 제공됩니다. AWS 솔루션즈 아키텍트(SA)를 사용하면 가용성 목표에 적합한 설계를 생성할 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 파악하고, 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.

기반

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

온프레미스 환경의 경우, 이러한 요구 사항에 따른 종속성으로 인해 리드 시간이 길어질 수 있으므로 결국 초기 계획에 이를 반영해야 합니다. 그러나 AWS에는 이러한 기초 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 안정성 고려 사항에 중점을 둔 모범 사례를 설명합니다.

- 서비스 할당량 및 제약 조건 관리
- 네트워크 토폴로지 프로비저닝

서비스 할당량 및 제약 조건 관리

클라우드 기반 워크로드 아키텍처에는 서비스 할당량(서비스 한도라고도 함)이라는 것이 있습니다. 이러한 할당량은 실수로 필요한 것보다 많은 리소스를 프로비저닝하는 것을 방지하고 API 작업에 대한 요청 속도를 제한하여 서비스를 침해로부터 보호하기 위해 존재합니다. 광섬유 케이블을 통해 비트를 푸시할 수 있는 속도나 물리적 디스크의 스토리지 양과 같은 리소스 제약 조건도 있습니다.

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

서비스 할당량 및 제약 조건 인식: 워크로드 아키텍처에 대한 기본 할당량이 있으며 할당량 증가를 요청할 수 있습니다. 또한 디스크 또는 네트워크와 같은 리소스 제약 조건은 잠재적인 영향을 미칠 수 있습니다.

Service Quotas는 100개 이상의 AWS 서비스에 대한 할당량을 단일 위치에서 관리할 수 있는 AWS 서비스입니다. 할당량 값을 조회하는 것에 더해 Service Quotas 콘솔 또는 AWS SDK를 통해 할당량 증가를 요청하고 추적할 수 있습니다. AWS Trusted Advisor는 일부 서비스의 특정 측면에 대한 사용량 및 할당량을 표시하는 서비스 할당량 확인 기능을 제공합니다. 각 서비스의 기본 서비스 할당량은 해당하는 서비스의 AWS 설명서에도 문서화되어 있습니다. 예를 들어 Amazon VPC 할당량을 참조하십시오. 조절된 API에 대한 속도 제한은 API Gateway 자체에서 사용량 계획을 구성하는 방법으로 설정됩니다. 해당하는 서비스에서 구성으로 설정되는 기타 제한으로는 프로비저닝된 IOPS, 할당된 RDS 스토리지 및 EBS 볼륨 할당이 있습니다. Amazon Elastic Compute Cloud(Amazon EC2)에는 인스턴스, Amazon Elastic Block Store(Amazon EBS) 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 AWS Support에 문의하여 완화 방법이 있는지 확인하십시오.

계정 및 리전 전체 할당량 관리: 여러 AWS 계정 또는 AWS 리전을 사용하는 경우 프로덕션 워크로드가 실행되는 모든 환경에서 적절한 할당량을 요청해야 합니다.

서비스 할당량은 계정별로 추적됩니다. 다른 언급이 없는 한 각 할당량은 AWS 리전별로 다릅니다.

프로덕션 환경에 더해 적용 가능한 모든 비 프로덕션 환경에서도 할당량을 관리하여 테스트 및 개발에 방해가 되지 않도록 합니다.

아키텍처를 통해 고정된 서비스 할당량 및 제약 조건 수용: 변경할 수 없는 서비스 할당량과 물리적 리소스를 인지하고 안정성에 영향을 미치지 않도록 설계합니다.

예를 들어 네트워크 대역폭, AWS Lambda 페이로드 크기, API Gateway의 조절 버스트 속도, Amazon Redshift 클러스터에 대한 동시 사용자 연결 등이 여기에 포함됩니다.

할당량 모니터링 및 관리: 잠재적 사용량을 평가하고 할당량을 적절히 늘려 사용량 증가를 계획합니다.

지원되는 서비스의 경우 사용량을 모니터링하고 할당량에 근접할 때 알림을 받도록 CloudWatch 경보를 구성하여 할당량을 관리할 수 있습니다. 이러한 경보는 Service Quotas 또는 Trusted Advisor에서 트리거될 수 있습니다. CloudWatch Logs의 지표 필터를 사용하여 로그의 패턴을 검색하고 추출하여 사용량이 할당량 임계값에 근접하는지 여부를 확인할 수도 있습니다.

할당량 관리 자동화: 임계값에 근접할 때 알림을 제공하는 도구를 구현합니다. Service Quotas API를 사용하여 할당량 증가 요청을 자동화할 수 있습니다.

CMDB(구성 관리 데이터베이스) 또는 티켓팅 시스템을 Service Quotas와 통합하면 할당량 증가 요청 및 현재 할당량 추적을 자동화할 수 있습니다. Service Quotas에서는 AWS SDK에 더해 AWS 명령줄 도구를 사용하여 자동화를 수행할 수도 있습니다.

현재 할당량과 최대 사용량 사이에 장애 조치를 수용할 수 있는 충분한 간격이 있는지 확인: 리소스에 장애가 발생한 경우 해당 리소스는 종료되기 전까지 할당량 계산에 포함될 수 있습니다. 그러므로 할당량에는 장애 발생 리소스가 종료되기 전까지 장애가 발생한 모든 리소스와 교체 리소스의 중복이 포함되어야 합니다. 이 차이를 계산할 때는 가용 영역 장애를 고려해야 합니다.

리소스

동영상

- AWS Live re:Inforce 2019 - Service Quotas

설명서

- What Is Service Quotas?
- AWS Service Quotas(이전의 서비스 한도)
- Amazon EC2 서비스 한도
- AWS Trusted Advisor 모범 사례 점검 항목(**서비스 한도** 섹션 참조)

- AWS Answers의 AWS Limit Monitor
- AWS Marketplace: 한도 추적에 도움이 되는 CMDB 제품
- APN 파트너: 구성 관리를 지원할 수 있는 파트너

네트워크 토폴로지 계획

워크로드는 여러 환경에 존재할 수 있습니다. 여기에는 다중 클라우드 환경(퍼블릭 액세스 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/네트워크와의 통합을 예상하여 네트워크 토폴로지 및 주소 지정을 계획해야 합니다.

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

워크로드 퍼블릭 엔드포인트에 고가용성 네트워크 연결 사용: 이러한 엔드포인트와 엔드포인트에 대한 라우팅은 고가용성을 제공해야 합니다. 이러한 고가용성을 달성하려면 고가용성 DNS, CDN(콘텐츠 전송 네트워크), API Gateway, 로드 밸런싱 또는 역방향 프록시를 사용합니다.

Amazon Route 53, AWS Global Accelerator, Amazon CloudFront, Amazon API Gateway 및 Elastic Load Balancing(ELB)은 모두 고가용성의 퍼블릭 엔드포인트를 제공합니다. AWS Marketplace 소프트웨어 어플라이언스를 평가하여 로드 밸런싱 및 프록시에 사용할 어플라이언스를 선택할 수도 있습니다.

워크로드가 제공하는 서비스의 소비자는 최종 사용자든 다른 서비스든 이러한 서비스 엔드포인트로 요청을 보냅니다. 여러 AWS 리소스를 사용하여 고가용성 엔드포인트를 제공할 수 있습니다.

Elastic Load Balancing은 가용 영역에 걸쳐 로드 밸런싱을 제공하고, 계층 4(TCP) 또는 계층 7(http/https) 라우팅을 수행하고, AWS WAF와 통합되며, AWS Auto Scaling 통합을 통해 자가

복구 인프라를 생성하고 트래픽 증가를 흡수하는 동시에 트래픽이 감소할 때는 리소스를 해제합니다.

Amazon Route 53는 확장 가능한 고가용성의 DNS(Domain Name System) 서비스로, 사용자 요청을 Amazon EC2 인스턴스, Elastic Load Balancing 로드 밸런서 또는 Amazon S3 버킷처럼 AWS에서 실행되는 인프라에 효과적으로 연결하며, 사용자를 AWS 외부의 인프라로 라우팅하는 데에도 사용할 수 있습니다.

AWS Global Accelerator는 AWS 글로벌 네트워크를 통해 트래픽을 최적의 엔드포인트로 보내는 데 사용할 수 있는 네트워크 계층 서비스입니다.

DDoS(분산 서비스 거부) 공격은 합법적인 트래픽을 차단하고 사용자의 가용성을 낮출 위험이 있습니다. AWS Shield는 워크로드의 AWS 서비스 엔드포인트에 이러한 공격에 대한 자동 보호를 추가 비용 없이 제공합니다. AWS Marketplace 및 APN 파트너에서 제공하는 가상 어플라이언스를 사용하여 요구에 맞게 이러한 기능을 보강할 수 있습니다.

클라우드와 온프레미스 환경의 프라이빗 네트워크 간에 중복 연결 프로비저닝: 개별적으로 배포된 프라이빗 네트워크 간에 여러 AWS Direct Connect(DX) 연결 또는 VPN 터널을 사용합니다. 가용성을 높이려는 경우에는 여러 DX 위치를 사용합니다. 여러 AWS 리전을 사용하는 경우 두 개 이상의 AWS 리전에서 중복성을 확인하십시오. VPN을 종료하는 AWS Marketplace 어플라이언스를 평가해야 할 수 있습니다. AWS Marketplace 어플라이언스를 사용하는 경우 다른 가용 영역에서 고가용성을 위해 중복 인스턴스를 배포합니다.

Direct Connect는 온프레미스 환경에서 AWS로의 전용 네트워크 연결을 쉽게 설정할 수 있게 지원하는 클라우드 서비스입니다. Direct Connect Gateway를 사용하면 온프레미스 데이터 센터를 여러 AWS 리전에 분산된 다수의 AWS VPC에 연결할 수 있습니다.

이러한 중복성은 연결 복원력에 영향을 주는 잠재적 장애를 해결합니다.

- 토폴로지에서 장애가 발생하는 경우 복원할 방법
- 특정 항목을 잘못 구성하여 연결을 제거하는 경우의 결과
- 예기치 않은 트래픽/서비스 사용량 증가 처리 가능 여부
- DDoS(분산 서비스 거부) 공격 시도에서 정상 상태 유지 가능 여부

VPN을 통해 온프레미스 데이터 센터에 VPC를 연결하는 경우 어플라이언스를 실행해야 하는 인스턴스 크기 및 공급업체를 선택할 때 필요한 복원력 및 대역폭 요구 사항을 고려해야 합니다. 해당 구현에서 복원되지 않는 VPN 어플라이언스를 사용하는 경우에는 두 번째 어플라이언스를 통한 중복 연결을 설정해야 합니다. 이러한 모든 시나리오에서는 허용되는 복구 시간을 정의하고 테스트를 진행하여 해당 요구 사항을 충족할 수 있는지를 확인해야 합니다.

Direct Connect 연결을 사용하여 VPC를 데이터 센터에 연결하기로 선택한 경우 이 연결이 고가용성이어야 한다면 각 데이터 센터에서 중복 DX 연결을 사용하십시오. 중복 연결에는 첫 번째 연결과 다른 위치의 두 번째 DX 연결이 사용되어야 합니다. 데이터 센터가 다수인 경우 연결이 각기 다른 위치에서 종료되는지 확인합니다. Direct Connect 복원 도구 키트를 사용하여 설정할 수 있습니다.

AWS VPN을 사용하여 인터넷을 통해 VPN으로 장애 조치하려는 경우, VPN 터널당 최대 1.25Gbps의 처리량이 지원되기는 하지만 여러 AWS Managed VPN 터널이 같은 VGW에서 종료되는 경우에는 아웃바운드 트래픽용 ECMP(Equal Cost Multi Path)가 지원되지 않는다는 점을 알아야 합니다. 장애 조치 중 1Gbps 미만의 속도가 허용되는 경우가 아니라면 AWS Managed VPN을 Direct Connect 연결의 백업으로 사용하지 않는 것이 좋습니다.

또한 VPC 엔드포인트를 사용하여 지원되는 AWS 서비스와 AWS PrivateLink 기반 VPC 종단점 서비스에 퍼블릭 인터넷을 통하지 않고 비공개로 VPC를 연결할 수 있습니다. 엔드포인트는 가상 디바이스입니다. 수평적으로 확장되고 이중화된 고가용성의 VPC 구성 요소입니다. 엔드포인트를 사용하면 네트워크 트래픽에 미치는 가용성 위험이나 대역폭 제약 없이 VPC의 인스턴스와 서비스 간에 통신할 수 있습니다.

확장 및 가용성을 위한 IP 서브넷 할당 계정 확인: 개별 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 준수하십시오.

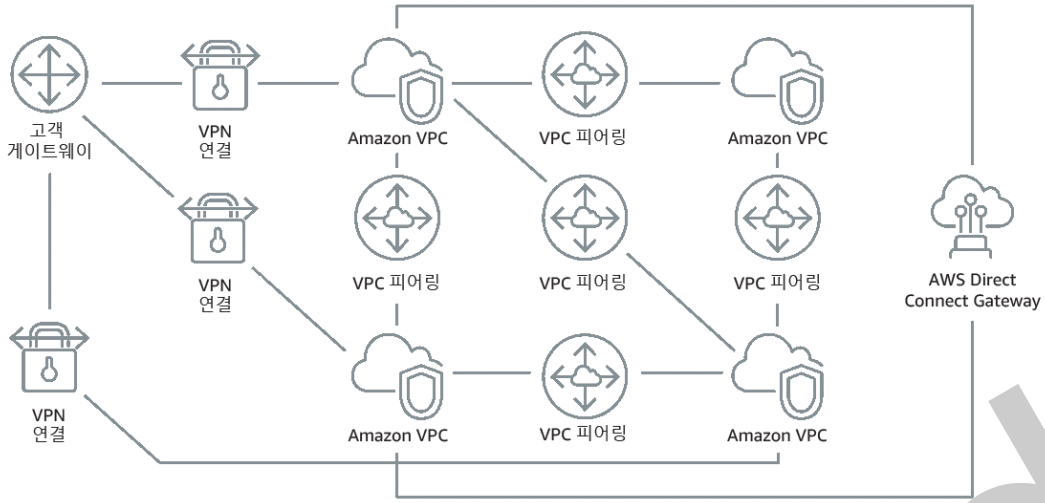
- 리전당 두 개 이상의 VPC에 대한 IP 주소 공간을 허용합니다.

- VPC 내에서 여러 가용 영역에 걸쳐 있는 여러 서브넷에 공간을 허용합니다.
- 사용되지 않은 CIDR 블록 공간은 향후 확장을 위해 항상 VPC 내에 남겨둡니다.
- 기계 학습용 스팟 플릿, Amazon EMR 클러스터 또는 Amazon Redshift 클러스터 등 사용할 수 있는 임시 EC2 인스턴스 플릿의 요구 사항을 충족할 IP 주소 공간이 있는지 확인합니다.
- 참고로 각 서브넷 CIDR 블록에서 처음 4개의 IP 주소와 마지막 IP 주소는 예약되므로 사용할 수 없습니다.

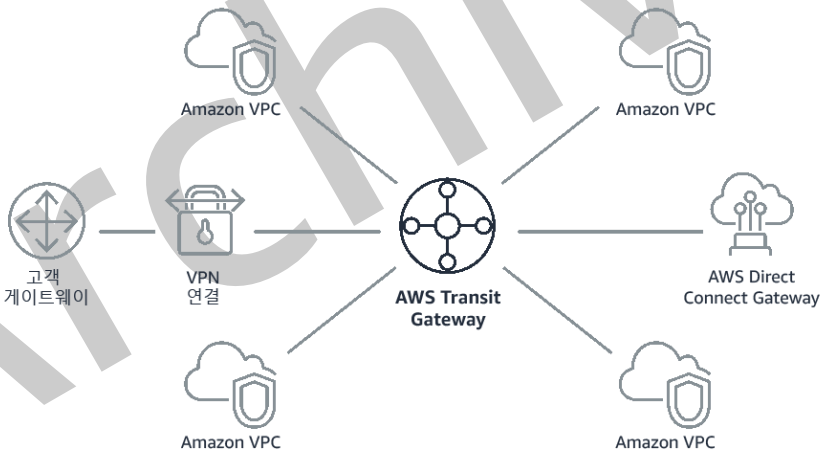
대규모 VPC CIDR 블록 배포를 계획해야 합니다. VPC에 할당된 초기 VPC CIDR 블록은 변경 또는 삭제가 불가능하지만 중첩되지 않은 추가 CIDR 블록을 VPC에 추가할 수 있습니다. 서브넷 IPv4 CIDR은 변경할 수 없지만 IPv6 CIDR은 변경할 수 있습니다. 가능한 최대 규모(/16)의 VPC를 배포하면 IP 주소 수가 65,000개를 초과하게 됩니다. 기본 10.x.x.x IP 주소 공간에만 255개의 VPC를 프로비저닝할 수 있습니다. 따라서 규모를 너무 작게 하는 것보다 지나치다 싶을 만큼 큰 규모로 배포해야 VPC를 더 쉽게 관리할 수 있습니다.

다대다 메시보다 허브 앤 스포크 토폴로지를 선택: 셋 이상의 네트워크 주소 공간(예: VPC와 온프레미스 네트워크)이 VPC 피어링, AWS Direct Connect 또는 VPN을 통해 연결되는 경우 AWS Transit Gateway가 제공하는 것과 같은 허브 앤 스포크 모델을 사용합니다.

이러한 네트워크가 두 개 뿐인 경우에는 서로 연결하면 되지만 네트워크 수가 증가하면 이 메시 기반 연결의 복잡성이 크게 증가합니다. AWS Transit Gateway는 유지 관리가 간편한 허브 앤 스포크 모델을 제공하므로 여러 네트워크에 걸쳐 트래픽을 라우팅할 수 있습니다.



AWS Transit Gateway가 없는 경우: VPN 연결을 사용하여 각 Amazon VPC를 서로 피어링하고 각 온사이트 위치에 피어링해야 합니다. 이 경우 확장에 따라 복잡성이 증가할 수 있습니다.



AWS Transit Gateway를 사용하는 경우: 각 Amazon VPC 또는 VPN을 AWS Transit Gateway에 연결하기만 하면 각 VPC 또는 VPN 간에 트래픽이 라우팅됩니다.

연결되는 모든 프라이빗 주소 공간에서 겹치지 않는 프라이빗 IP 주소 범위 적용: VPN을 통해 피어링하거나 연결할 때 각 VPC의 IP 주소 범위는 서로 겹치지 않아야 합니다. 마찬가지로, VPC와

온프레미스 환경 간의 IP 주소 충돌 또는 사용하는 다른 클라우드 공급자와의 IP 주소 충돌을 방지해야 합니다. 필요한 경우 프라이빗 IP 주소 범위를 할당할 수 있어야 합니다.

IPAM(IP 주소 관리) 시스템을 사용하는 것이 도움이 될 수 있습니다. AWS Marketplace에서 여러 IPAM을 사용할 수 있습니다.

리소스

동영상

- 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)

설명서

- What Is a Transit Gateway?
- Amazon VPC란 무엇인가?
- Direct Connect 게이트웨이 작업
- Direct Connect 복원 도구 키트를 사용하여 시작
- 여러 데이터 센터 HA 네트워크 연결
- What Is AWS Global Accelerator?
- Using redundant Site-to-Site VPN connections to provide failover
- VPC 엔드포인트 및 VPC 엔드포인트 서비스(AWS PrivateLink)
- Amazon Virtual Private Cloud Connectivity Options 백서
- 네트워크 인프라에 대한 AWS Marketplace
- APN 파트너: 네트워킹 계획을 지원할 수 있는 파트너

워크로드 아키텍처

안정적인 워크로드는 소프트웨어와 인프라에 대한 사전 설계 결정에서 시작됩니다. 아키텍처 선택은 모든 5가지 Well-Architected 원칙에서 워크로드 동작에 영향을 미칩니다. 안정성을 달성하려면 특정 패턴을 따라야 합니다.

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

- 적절한 서비스 아키텍처 구현
- 분산 시스템의 장애 방지를 위한 소프트웨어 설계
- 분산 시스템의 장애 완화를 위한 소프트웨어 설계

워크로드 서비스 아키텍처 설계

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

SOA(서비스 지향 아키텍처) 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 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.

워크로드 분할 방법 선택: 모놀리식 아키텍처는 피해야 합니다. 대신 SOA와 마이크로서비스 중에서 선택하십시오. 각 아키텍처를 선택할 때는 이점과 복잡성의 균형을 고려합니다. 신제품의 첫 출시에 필요한 것과 워크로드를 확장할 때 필요한 것은 다릅니다. 작은 세그먼트를 사용하면 민첩성, 조직의 유연성 및 확장성이 향상됩니다. 복잡성에는 지연 시간 증가, 디버깅 복잡성, 운영 부담 증가가 포함됩니다.

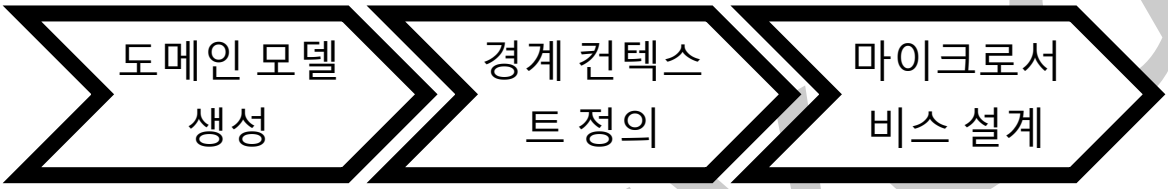
처음에 모놀리식 아키텍처를 선택하더라도, 사용자 채택에 따라 제품을 확장할 때 SOA 또는 마이크로서비스로 변경할 수 있는 기능이 있는 모듈식 아키텍처인지 확인해야 합니다. SOA와 마이크로서비스는 더 작게 분할할 수 있기 때문에 현대의 확장 가능하고 안정적인 아키텍처로 선호되지만 특히 마이크로서비스 아키텍처를 배포하는 경우 절충을 고려해야 합니다. 그 중 하나는 분산 컴퓨팅 아키텍처가 구축되므로 사용자 지연 시간 요구 사항을 충족하기가 더 어려워지며, 사용자 상호 작용을 디버그하고 추적하는 과정이 더 복잡해진다는 것입니다. AWS 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 아키텍처와 마이크로서비스 아키텍처 비교

특정 비즈니스 도메인 및 기능에 중점을 둔 서비스 구축: SOA는 비즈니스 요구 사항에 따라 명확하게 정의된 함수로 서비스를 구축합니다. 마이크로서비스는 도메인 모델과 경계 컨텍스트를 사용하여 이를 추가로 제한함으로써 각 서비스가 한 가지 작업을 수행하도록 합니다. 특정 기능에 집중하면 다양한 서비스의 안정성 요구 사항을 구분하고 보다 구체적으로 투자의

대상을 지정할 수 있습니다. 또한 비즈니스 문제가 간단해지고 각 서비스에 소규모 팀이 연결되므로 조직의 확장이 수월해집니다.

마이크로서비스 아키텍처를 설계할 때는 DDD(Domain-Driven Design)에서 엔터티를 사용하여 비즈니스 문제를 모델링하는 것이 도움이 됩니다. 예를 들어 Amazon.com 엔터티에는 패키지, 배송, 일정, 가격, 할인 및 통화가 포함될 수 있습니다. 이 모델은 경계 컨텍스트를 사용하여 유사한 기능 및 속성을 공유하는 엔터티를 그룹화하는 더 작은 모델로 구분됩니다. 따라서 Amazon 예제 패키지에서 배송 및 일정은 배송 컨텍스트에 포함되지만 가격, 할인 및 통화는 요금 컨텍스트에 포함됩니다. 모델을 컨텍스트로 나누면 마이크로서비스의 경계를 지정하는 방법에 대한 템플릿을 사용할 수 있게 됩니다.



API당 서비스 계약 제공: 서비스 계약은 서비스 통합에 대한 팀 간의 문서화된 계약으로, 머신 판독 가능한 API 정의, 속도 제한 및 성능 기대치를 포함합니다. 버전 관리 전략을 사용하면 클라이언트에서 기존 API를 계속 사용하면서 준비가 될 때 애플리케이션을 최신 API로 마이그레이션할 수 있습니다. 계약을 위반하지 않는 한 언제든지 배포가 가능합니다. 서비스 공급자 팀은 원하는 기술 스택을 사용하여 API 계약을 충족할 수 있습니다. 마찬가지로 서비스 소비자는 자체 기술을 사용할 수 있습니다.

마이크로서비스는 SOA의 개념을 최소한의 기능 세트를 포함한 서비스를 생성하는 지점으로 가져갑니다. 각 서비스는 API 및 설계 목표, 한도 및 서비스 사용을 위한 기타 고려 사항을 게시합니다. 그러면 호출 애플리케이션과의 "계약"이 설정됩니다. 이러한 방식에서 제공되는 세 가지 주요 이점은 다음과 같습니다.

- 서비스에서 간단한 업무상의 문제만 처리하면 되며, 소규모 팀이 업무상의 문제를 소유할 수 있습니다. 그러므로 조직 크기를 더 효율적으로 조정할 수 있습니다.
- 팀은 API 및 "계약" 요구 사항을 충족하는 경우에 한해 언제든지 서비스를 배포할 수 있습니다.

- 팀은 API 및 "계약" 요구 사항을 충족하는 경우에 한해 원하는 모든 기술 스택을 사용할 수 있습니다.

Amazon API Gateway는 어떤 규모에서든 개발자가 API를 손쉽게 생성, 게시, 유지 관리, 모니터링 및 보안할 수 있게 해주는 완전관리형 서비스입니다. 트래픽 관리, 권한 부여 및 액세스 제어, 모니터링 및 API 버전 관리를 비롯하여 최대 수십만 건의 동시 API 호출을 수락하고 처리하는 데 관련된 모든 작업을 처리합니다. 이전의 Swagger Specification인 OpenAPI Specification(OAS)을 사용하여 API 계약을 정의한 다음 이 정의를 API Gateway로 가져올 수 있습니다. 그런 다음 API Gateway를 사용하여 API를 버전 관리하고 배포할 수 있습니다.

리소스

설명서

- Amazon API Gateway: OpenAPI를 사용하여 REST API 구성
- AWS에서 마이크로서비스 구현
- AWS 기반 마이크로서비스

외부 링크

- Microservices - a definition of this new architectural term
- Microservice Trade-Offs
- Bounded Context(a central pattern in Domain-Driven Design)

분산 시스템의 장애 방지를 위한 상호 작용 설계

분산 시스템에서 서버 또는 서비스와 같은 구성 요소는 통신 네트워크를 사용하여 상호 연결됩니다. 워크로드는 이러한 네트워크에서 데이터 손실 또는 지연 시간이 발생하더라도 안정적으로 작동해야 합니다. 분산 시스템의 구성 요소는 다른 구성 요소나 워크로드에 부정적인 영향을 미치지 않는 방식으로 작동해야 합니다. 여기에 나온 모범 사례는 장애를 방지하고 MTBF(평균 장애 간격)를 개선합니다.

필요한 분산 시스템의 종류 식별: 하드 실시간 분산 시스템은 응답을 동기식으로 신속하게 제공해야 하며, 소프트 실시간 시스템은 몇 분 이상의 보다 관대한 기간에 응답을 제공할 수

있습니다. 오프라인 시스템은 배치 또는 비동기식 처리를 통해 응답을 처리합니다. 하드 실시간 분산 시스템은 안정성 요구 사항이 가장 엄격합니다.

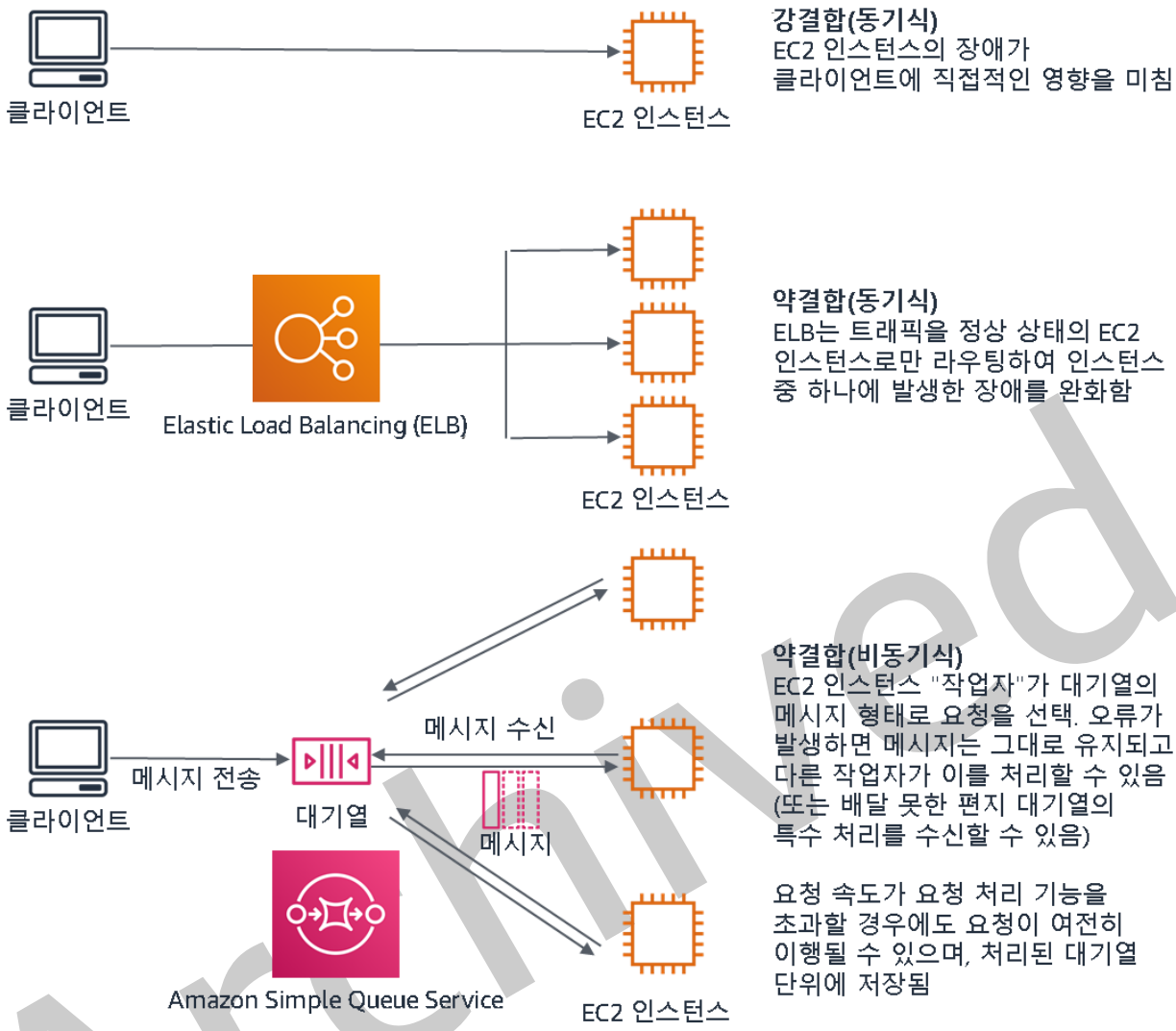
분산 시스템의 가장 어려운 문제는 요청/응답 서비스라고도 하는 하드 실시간 분산 시스템과 관련된 것입니다. 이 문제가 어려운 이유는 언제 도착할지 예상할 수 없는 요청에 대해 신속하게 응답을 제공해야 하기 때문입니다(예: 응답이 올 때까지 앞에서 대기하는 고객). 예를 들어 프런트엔드 웹 서버, 주문 파이프라인, 신용 카드 거래, 모든 AWS API 및 전화 통신이 여기에 포함됩니다.

약결합 종속성 구현: 대기열 처리 시스템, 스트리밍 시스템, 워크플로 및 로드 밸런서와 같은 종속성은 약하게 결합됩니다. 약한 결합은 한 구성 요소의 동작을 다른 종속 구성 요소에서 분리하여 복원력 및 민첩성을 높이는 데 도움이 됩니다.

한 구성 요소에 대한 변경이 다른 종속 구성 요소의 변경을 강제하는 경우 이러한 구성 요소는 강하게 결합된 것입니다. 약한 결합에서는 이 종속성이 분리되므로 종속 구성 요소에서는 버전이 지정되고 게시된 인터페이스만 알면 됩니다. 종속성 간에 약한 결합을 구현하면 한 구성 요소의 장애가 다른 구성 요소에 영향을 미치지 않도록 분리됩니다.

약한 결합을 사용하면 종속 구성 요소에 미치는 위험을 최소화하면서 추가 코드 또는 기능을 자유롭게 추가할 수 있습니다. 또한 종속성의 기본 구현을 확장하거나 변경할 수 있으므로 확장성이 개선됩니다.

약결합을 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 않으며 일반적으로 내구성이 있는 중간 스토리지 계층(예: SQS 대기열 또는 Amazon Kinesis 또는 AWS Step Functions와 같은 스트리밍 데이터 플랫폼)을 통해 통합됩니다.



Amazon SQS 대기열과 Elastic Load Balancer는 약결합을 위한 중간 계층을 추가할 수 있는 두 가지 방법의 예입니다. AWS 클라우드에서는 Amazon EventBridge를 사용하여 이벤트 기반 아키텍처를 구축할 수도 있습니다. Amazon EventBridge를 사용하면 클라이언트가 사용하는 서비스에서 클라이언트(이벤트 생산자)를 추상화할 수 있습니다. Amazon Simple Notification Service는 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 하드 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.

모든 응답의 멱등성 유지: 멱등성이 있는 서비스는 각 요청이 정확히 한 번만 완료되도록 합니다. 이렇게 하면 다수의 동일한 요청에서 단일 요청과 동일한 결과가 나옵니다. 멱등성이 있는 서비스를 사용하면 클라이언트가 요청이 오류로 여러 번 처리될 것이라는 염려 없이 재시도를 시행할 수 있습니다. 이를 위해 클라이언트는 멱등성 토큰을 사용하여 API 요청을 실행할 수 있으며 요청이 반복될 때마다 동일한 토큰이 사용됩니다. 멱등성이 있는 서비스 API는 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) 수행하기는 쉽습니다. 그러나 작업의 멱등성을 보장하기는 어렵습니다. 즉 작업을 정확히 한 번 수행하여 다수의 동일한 요청에서 단일 요청과 동일한 결과를 얻기는 어렵습니다. API에서 멱등성 토큰을 사용하면 서비스가 중복 레코드를 생성하지 않고 부작용 없이 변경 요청을 한 번 이상 수신할 수 있습니다.

일정한 작업 처리: 대규모 로드가 급속도로 변경되면 시스템에서 장애가 발생할 수 있습니다. 예를 들어 서버 수천 대의 상태를 모니터링하는 상태 확인 시스템은 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 상태 확인 시스템에서는 매번 현재 상태의 전체 스냅샷을 전송하는 것이 낫습니다. 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. 실제로 Amazon Route 53 상태 확인의 제어 영역은 이 방식으로 설계되었습니다.

리소스

동영상

- AWS re:Invent 2019: Moving to event-driven architectures(SVS308)
- AWS re:Invent 2018: Close Loops & Opening Minds: How to Take Control of Systems, Big & Small ARC337(약결합, 일정한 작업, 정적 안정성 포함)
- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205)(EventBridge, SQS, SNS 논의)

설명서

- CloudWatch 지표를 게시하는 AWS 서비스
- Amazon Simple Queue Service란 무엇입니까?
- Amazon EC2: Ensuring Idempotency
- Amazon Builders' Library: 분산 시스템의 도전 과제
- Centralized Logging 솔루션
- AWS Marketplace: 모니터링 및 알림에 사용할 수 있는 제품
- APN 파트너: 모니터링 및 로깅을 지원할 수 있는 파트너

분산 시스템의 장애 완화 또는 극복을 위한 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 안정적으로 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 MTTR(평균 복구 시간)이 개선됩니다.

다음 모범 사례는 장애를 방지하고 MTBF(평균 장애 간격)를 개선합니다.

단계적 성능 저하를 구현하여 해당하는 하드 종속성을 소프트 종속성으로 변환: 구성 요소의 종속성이 비정상인 경우 구성 요소 자체는 성능이 저하된 방식으로 계속해서 작동합니다. 예를 들어 종속성 호출이 실패하는 경우 미리 결정된 정적 응답이 대신 사용됩니다.

서비스 A가 서비스 B를 호출하고 서비스 B는 서비스 C를 호출하는 경우

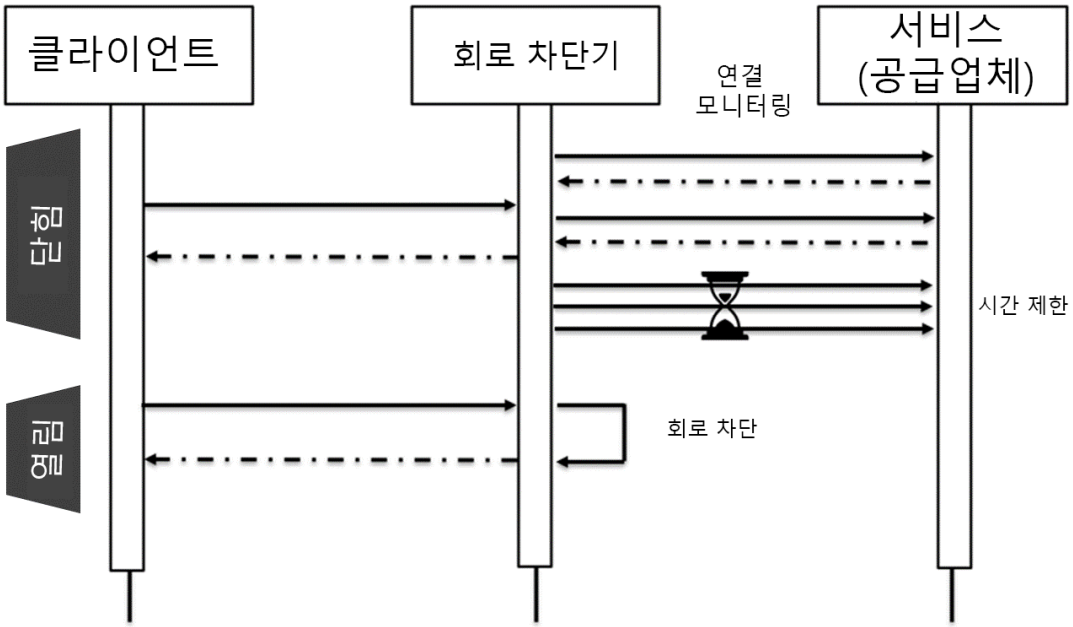
서비스 B가 서비스 C를 호출할 때 서비스 C에 장애가 발생하면 서비스 B는 성능이 저하된 응답을 서비스 A에 반환합니다.

서비스 B가 서비스 C를 호출할 때 서비스 B는 서비스 C로부터 오류 또는 시간 초과를 수신합니다. 서비스 B는 서비스 C(및 여기에 포함된 데이터)의 응답이 없는 대신 가능한 것을 반환합니다. 이 값은 마지막으로 캐시된 정상 값이 될 수 있습니다. 또는 서비스 B는 서비스 C에서 수신했을 수 있는 것에 대해 미리 결정된 정적 응답을 대체할 수 있습니다. 그런 다음 호출자인 서비스 A에 성능이 저하된 응답을 반환할 수 있습니다. 이 정적 응답이 없다면 서비스 C의 장애가 서비스 B를 통해 서비스 A로 계단식으로 전달되어 가용성이 손실됩니다.

하드 종속성에 대한 가용성 방정식의 승인자(하드 종속성의 가용성 계산)에 따라 C의 가용성 손실은 B의 유효 가용성에 심각한 영향을 미칩니다. 서비스 B는 정적 응답을 반환함으로써 C의 장애를 완화하고 성능이 저하되기는 하지만 서비스 C의 가용성을 100%처럼 보이게 만듭니다(오류 조건에서 정적 응답을 성공적으로 반환한다는 가정 하에). 정적 응답은 오류를 반환하는 것에 대한 단순한 대안이며 다른 수단을 사용하여 응답을 다시 계산하려는 시도가 아닙니다. 완전히 다른 메커니즘에서 동일한 결과를 얻기 위한 이러한 시도를 폴백 동작이라고 하는데 이는 피해야 할 안티패턴입니다.

단계적 성능 저하의 또 다른 예는 회로 차단기 패턴입니다. 장애가 일시적인 경우에는 재시도 전략을 사용해야 합니다. 일시적이지 않고 작업이 실패할 가능성이 있는 경우 회로 차단기 패턴은 실패할 가능성이 있는 요청을 수행하지 못하도록 클라이언트를 차단합니다. 요청이 정상적으로 처리될 때는 회로 차단기가 닫히고 요청이 전송됩니다. 원격 시스템에서 오류를 반환하기 시작하거나 지연 시간이 길어지면 회로 차단기가 열리고 종속성이 무시되거나 덜 포괄적이지만

간단하게 얻을 수 있는 응답(단순한 응답 캐시일 수 있음)으로 결과가 대체됩니다. 시스템은 주기적으로 종속성 호출을 시도해 복구 여부를 확인합니다. 종속성이 복구된 경우에는 회로 차단기가 닫힙니다.



닫힌 상태와 열린 상태를 보여주는 회로 차단기.

다이어그램에 표시된 닫힌 상태와 열린 상태 외에, 열린 상태에서 구성 가능한 기간이 지나면 회로 차단기가 반개방 상태로 전환될 수 있습니다. 이 상태에서는 평소보다 훨씬 낮은 속도로 주기적으로 서비스 호출이 시도됩니다. 이 프로브는 서비스의 상태를 확인하는 데 사용됩니다. 반개방 상태에서 여러 번 성공하면 회로 차단기가 닫힌 상태로 전환되고 정상 요청이 재개됩니다.

요청 조절: 예기치 않은 수요 증가에 대응하기 위한 완화 패턴입니다. 일부 요청은 처리되지만 정의된 제한을 초과하는 요청은 거부되고 병목 현상이 발생했음을 나타내는 메시지를 반환합니다. 클라이언트의 예상 동작은 중지 후 요청을 중단하거나 더 느린 속도로 다시 시도하는 것입니다.

각 노드나 셀이 처리할 수 있는 확인된 요청 용량으로 서비스를 설계해야 합니다. 이 용량은 로드 테스트를 통해 설정할 수 있습니다. 그런 다음 요청 도착 속도를 추적해야 하며, 임시 도착 속도가 이 한도를 초과하는 경우의 적절한 대응 방식은 요청이 쓰로틀링되었음을 알리는 것입니다. 그러면 사용자는 사용 가능 용량이 있을 수 있는 다른 노드/셀에 대해 요청을 다시 시도할 수

있습니다. Amazon API Gateway에서는 요청을 조절할 수 있는 방법을 제공합니다. Amazon SQS와 Amazon Kinesis는 요청을 버퍼링하여 요청 속도를 원활하게 하고 비동기식으로 처리할 수 있는 요청에 대한 조절 필요성을 완화할 수 있습니다.

재시도 호출 제어 및 제한: 지수 백오프를 사용하여 점진적으로 더 긴 간격 후에 다시 시도합니다. 지터를 도입하여 이러한 재시도 간격을 무작위로 지정하고 최대 재시도 횟수를 제한합니다.

분산 소프트웨어 시스템의 일반적인 구성 요소로는 서버, 로드 밸런서, 데이터베이스 및 DNS 서버가 있습니다. 작동 중일 때 장애가 발생하면 이러한 구성 요소 중 모든 구성 요소에서 오류가 생성되기 시작할 수 있습니다. 오류를 처리하는 기본 기술은 클라이언트 측에 재시도를 구현하는 것입니다. 이 기술은 애플리케이션의 안정성과 가용성을 높입니다. 그러나 오류가 발생하는 즉시 클라이언트가 실패한 작업을 대규모로 다시 시도할 경우 새 요청 및 만료된 요청이 각각 네트워크 대역폭을 두고 경합하여 네트워크가 빠르게 포화 상태가 될 수 있습니다. 이로 인해 재시도 폭풍이 발생하고 서비스 가용성이 떨어집니다. 이 패턴은 전체 시스템 장애가 발생할 때까지 계속될 수 있습니다.

이러한 상황을 방지하려면 일반적인 **지수 백오프**와 같은 백오프 알고리즘을 사용해야 합니다. 지수 백오프 알고리즘은 재시도가 수행되는 속도를 점진적으로 줄여 네트워크 정체를 방지합니다.

AWS의 SDK를 비롯한 많은 SDK 및 소프트웨어 라이브러리가 이러한 알고리즘 버전을 구현합니다. 하지만 백오프 알고리즘이 존재한다고 가정하지 말고 항상 이를 테스트하고 확인하십시오.

단순한 백오프만으로는 충분하지 않습니다. 분산 시스템에서는 모든 클라이언트가 동시에 백오프되어 재시도 호출의 클러스터가 생성될 수 있기 때문입니다. Marc Brooker의 블로그 게시물 지수 백오프와 지터에 지수 백오프의 `wait()` 함수를 수정하여 재시도 호출 클러스터를 방지하는 방법이 설명되어 있습니다. 해결 방법은 `wait()` 함수에 **지터**를 추가하는 것입니다. 장시간의 재시도를 방지하려면 구현 시 백오프를 최대값으로 제한해야 합니다.

마지막으로, 최대 재시도 횟수 또는 재시도를 실패로 처리할 경과 시간을 구성하는 것이 중요합니다. AWS SDK는 이 기능을 기본적으로 구현하며 이 기능은 구성이 가능합니다. 스택에서 아래에 있는 서비스의 경우 최대 재시도 제한이 0 또는 1이면 위험이 제한되지만, 재시도가 스택에서 더 위에 있는 서비스로 위임되므로 여전히 유효합니다.

빠른 실패 및 대기열 제한: 워크로드가 요청에 성공적으로 응답할 수 없는 경우 빠르게 실패합니다. 이렇게 하면 요청에 연결된 리소스를 해제할 수 있고 리소스가 부족한 경우 서비스 복구를 허용할 수 있습니다. 워크로드가 성공적으로 응답할 수 있지만 요청 속도가 너무 높은 경우에는 대기열을 사용하여 요청을 버퍼링합니다. 하지만 긴 대기열을 허용하지 마십시오. 대기열이 길면 클라이언트가 이미 포기한 무효 요청을 처리할 수 있습니다.

이 모범 사례는 요청의 서버 측 또는 수신자에 적용됩니다.

대기열은 시스템의 여러 수준에서 생성될 수 있습니다. 대기열에서는 응답을 필요로 하는 최근 요청 전에 더 이상 응답이 필요하지 않은 오래된 무효 요청이 처리되므로 빠른 복구 기능에 심각한 영향을 미칠 수 있습니다. 그러므로 대기열이 있는 위치를 숙지해야 합니다. 일반적으로 대기열은 워크플로 또는 데이터베이스에 기록된 작업에 숨겨집니다.

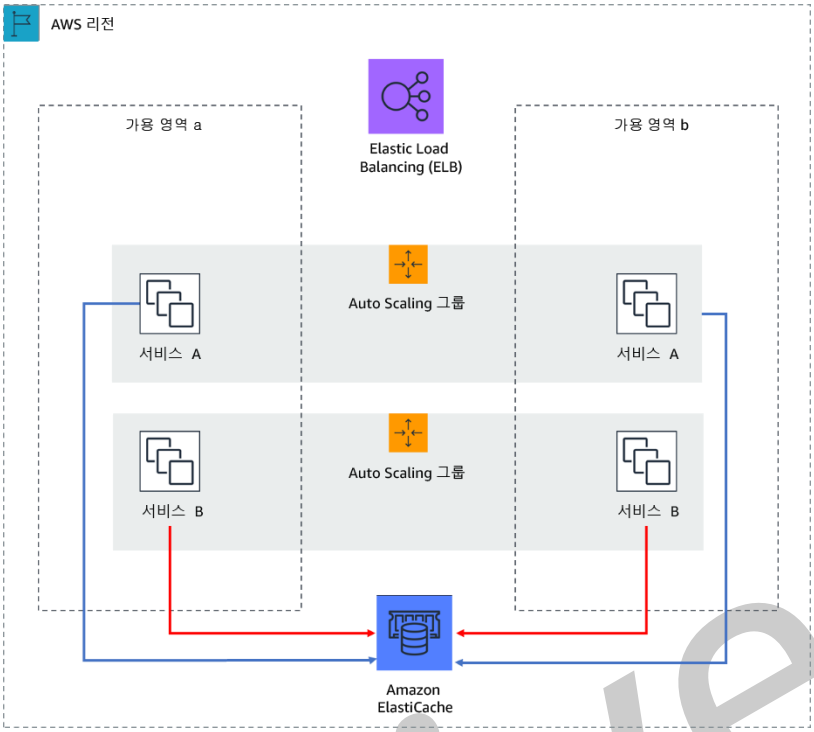
클라이언트 시간 제한 설정: 시간 제한을 적절히 설정하고, 체계적으로 확인합니다. 기본값을 사용하지 마십시오. 기본값은 일반적으로 너무 높게 설정됩니다.

이 모범 사례는 요청의 클라이언트 측 또는 발신자에 적용됩니다.

원격 호출과 일반적으로 프로세스 전체의 모든 호출에 연결 시간 제한과 요청 시간 제한을 모두 설정합니다. 기본적인 시간 제한 기능을 제공하는 프레임워크가 많지만 기본값이 무한하거나 너무 높은 경우가 많으므로 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 전체 중단이 발생할 수 있습니다.

Amazon에서 시간 제한, 재시도 및 지터를 사용한 백오프를 사용하는 방법에 대한 자세한 내용은 Builder's Library: 시간 제한, 재시도 및 지터를 사용한 백오프를 참조하십시오.

가능한 경우 서비스를 상태 비저장으로 설계: 서비스는 상태를 필요로 하지 않거나 디스크 또는 메모리의 로컬로 저장된 데이터에 종속성이 없도록 서로 다른 클라이언트 요청 간에 상태를 오프로드해야 합니다. 이렇게 하면 가용성에 미치는 영향 없이 서버를 대체할 수 있습니다. Amazon ElastiCache 또는 Amazon DynamoDB는 오프로드된 상태에 적합한 대상입니다.



이 상태 비저장 웹 애플리케이션에서는 세션 상태가 Amazon ElastiCache로 오프로드됩니다.

사용자 또는 서비스는 애플리케이션과 상호 작용할 때 세션을 구성하는 일련의 상호 작용을 수행하는 경우가 많습니다. 세션은 애플리케이션을 사용하는 동안 요청 간에 유지되는 사용자의 고유한 데이터입니다. 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 플랫폼을 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 수평적으로 확장할 수 있다는 것입니다.

비상 레버 구현: 이 프로세스는 워크로드의 가용성에 미치는 영향을 신속하게 완화할 수 있는 프로세스입니다. 근본 원인이 없는 상태에서 작동할 수 있습니다. 이상적인 비상 레버는 완전히 결정적인 활성화 및 비활성화 기준을 제공하여 확인자에 대한 인지 부담을 0으로 줄여줍니다. 레버의 예로는 모든 로봇 트래픽을 차단하거나 정적 응답을 제공하는 것이 있습니다. 레버는 종종 수동이지만 자동화할 수도 있습니다.

비상 레버 구현 및 사용을 위한 팁:

- 레버가 활성화되면 더 적은 수의 작업을 수행
- 단순하게 유지하고 바이모달 동작을 방지
- 레버를 정기적으로 테스트

다음은 비상 레버가 아닌 작업의 예입니다.

- 용량 추가
- 서비스를 사용하는 클라이언트의 서비스 소유자를 호출하고 호출을 줄이도록 요청
- 코드 변경 및 릴리스

리소스

동영상

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

설명서

- AWS에서 오류 재시도 및 지수 백오프
- Amazon API Gateway: 처리량 향상을 위해 API 요청 조절
- Amazon Builders' Library: 시간 제한, 재시도 및 지터를 사용한 백오프
- Amazon Builders' Library: 분산 시스템의 폴백 방지
- Amazon Builders' Library: 심각한 수준의 대기열 백로그 방지
- Amazon Builders' Library: 캐싱 관련 당면 과제 및 전략

실습

- Well-Architected lab: Level 300: Implementing Health Checks and Managing Dependencies to Improve Reliability

외부 링크

- CircuitBreaker("Release It!" 도서의 회로 차단기 요약)

도서

- Michael Nygard “Release It! Design and Deploy Production-Ready Software”

변경 관리

워크로드의 안정적인 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.

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

워크로드 리소스 모니터링

로그와 지표는 워크로드의 상태를 파악할 수 있는 유용한 도구입니다. 로그 및 지표를 모니터링하여 임계값을 초과하거나 중요한 이벤트가 발생하면 알림을 보내도록 워크로드를 구성할 수 있습니다. 모니터링을 수행하면 워크로드가 저성능 임계값을 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. 계측 프로세스 내 모든 종속성과 주요 작업에 대해 각 요청의 지연 시간, 오류율 및 가용성이 기록됩니다. 그리고 성공한 작업의 지표도 기록합니다. 그러면 곧 발생할 것으로 예상되는 문제를 발생 전에 파악할 수 있습니다. AWS는 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

생성 – 워크로드에 대한 모든 구성 요소 모니터링: Amazon CloudWatch 또는 타사 도구를 사용하여 워크로드의 구성 요소를 모니터링합니다. Personal Health Dashboard로 AWS 서비스를 모니터링하십시오.

프런트엔드, 비즈니스 로직 및 스토리지 계층을 포함하여 워크로드의 모든 구성 요소를 모니터링해야 합니다. 필요한 경우 주요 지표 및 로그에서 지표를 추출하는 방법을 정의하고 해당 경보 이벤트에 대한 임계값을 설정하고 생성합니다.

클라우드에서 모니터링은 새로운 기회를 제공합니다. 대부분의 클라우드 공급자는 워크로드의 여러 계층을 파악할 수 있는 사용자 지정 가능한 도구와 정보를 개발했습니다.

AWS에서 제공하는 다양한 모니터링 및 로그 정보를 사용하면 수요 변경 프로세스를 정의할 수 있습니다. 아래 목록에는 로그 및 지표 데이터를 생성하는 서비스와 기능 중 몇 가지가 나와 있습니다.

- Amazon ECS, Amazon EC2, Elastic Load Balancing, AWS Auto Scaling 및 Amazon EMR은 CPU, 네트워크 I/O 및 디스크 I/O 평균에 대한 지표를 게시합니다.
- Amazon Simple Storage Service(Amazon S3), Classic Load Balancer 및 Application Load Balancer에서는 Amazon CloudWatch Logs를 활성화할 수 있습니다.
- VPC Flow Logs를 활성화하여 VPC의 수신 및 발신 네트워크 트래픽을 분석할 수 있습니다.

- AWS CloudTrail은 AWS Management Console, AWS SDK, 명령줄 도구를 통해 수행된 작업을 포함하여 AWS 계정 활동을 기록합니다.
- Amazon EventBridge는 AWS 서비스의 변경 사항을 설명하는 시스템 이벤트 스트림을 실시간으로 제공합니다.
- AWS는 운영 체제 수준 로그를 수집하여 CloudWatch Logs로 스트리밍하는 도구를 제공합니다.
- 모든 차원의 지표에 사용자 지정 Amazon CloudWatch 지표를 사용할 수 있습니다.
- Amazon ECS 및 AWS Lambda는 로그 데이터를 CloudWatch Logs로 스트리밍합니다.
- Amazon Machine Learning(Amazon ML), Amazon Rekognition, Amazon Lex 및 Amazon Polly는 성공/실패한 요청 관련 지표를 제공합니다.
- AWS IoT는 규칙 실행 횟수 관련 지표와 규칙 관련 특정 성공/실패 지표를 제공합니다.
- Amazon API Gateway는 요청 수, 오류가 발생한 요청 및 API의 지연 시간 관련 지표를 제공합니다.
- Personal Health Dashboard는 AWS 리소스의 기반이 되는 AWS 서비스의 성능 및 가용성에 대한 맞춤형 보기를 제공합니다.

또한 원격 위치에서 모든 외부 엔드포인트를 모니터링하여 기본 구현과 독립되어 있는지 확인합니다. 이 능동적 모니터링은 애플리케이션 소비자가 수행하는 몇 가지 일반적인 작업을 주기적으로 실행하는 가상 트랜잭션("사용자 Canary"라고도 하지만 Canary 배포와 다름)을 사용하여 수행될 수 있습니다. 기간은 짧아야 하며 테스트 중에 워크플로에 과부하가 발생하지 않아야 합니다. Amazon CloudWatch Synthetics를 사용하면 엔드포인트 및 API 모니터링을 위한 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.

집계 – 지표 정의 및 계산: 로그 데이터를 저장하고 필요한 경우 필터를 적용하여 특정 로그 이벤트 수 또는 로그 이벤트 타임스탬프에서 계산된 지연 시간과 같은 지표를 계산합니다.

Amazon CloudWatch 및 Amazon S3는 기본 집계 및 스토리지 계층으로 사용됩니다. AWS Auto Scaling 및 ELB와 같은 일부 서비스에서는 클러스터나 인스턴스 전반에 걸쳐 CPU 로드 또는 평균

요청 지연 시간 관련 기본 지표를 "바로 사용 가능"하도록 제공됩니다. VPC Flow Logs 및 AWS CloudTrail과 같은 스트리밍 서비스의 경우에는 이벤트 데이터가 CloudWatch Logs로 전달되며, 이벤트 데이터에서 지표를 추출하려면 지표 필터를 정의하고 적용해야 합니다. 이렇게 하면 시계열 데이터가 나오며 알림을 트리거하도록 정의한 CloudWatch 경보에 대한 입력으로 이 데이터를 사용할 수 있습니다.

실시간 처리 및 경보 – 알림 전송: 중요한 이벤트가 발생할 때 알아야 하는 조직에 알림이 전송됩니다.

Amazon Simple Notification Service(Amazon SNS) 주제로 알림을 전송한 다음 원하는 수의 구독자에게 푸시할 수도 있습니다. 예를 들어 Amazon SNS는 기술 직원이 응답할 수 있도록 특정 이메일 별칭으로 알림을 전달할 수 있습니다.

실시간 처리 및 경보 – 응답 자동화: 이벤트가 감지되면 자동화를 사용하여 실패한 구성 요소를 대체하는 등의 조치를 취합니다.

알림을 사용하여 AWS Auto Scaling 이벤트를 트리거하면 클러스터가 수요 변경에 대응할 수 있습니다. 타사 티켓 시스템용 통합 지점으로 사용 가능한 Amazon Simple Queue Service(Amazon SQS)로 알림을 전송할 수도 있습니다. AWS Lambda에서도 알림을 구독하여 변경에 동적으로 대응하는 비동기 서버리스 모델을 사용자에게 제공할 수 있습니다. AWS Config는 AWS 리소스 구성을 지속적으로 모니터링하고 기록하며 AWS Systems Manager Automation을 트리거하여 문제를 해결할 수 있습니다.

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

Amazon CloudWatch Logs Insights는 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 쿼리가 지원됩니다. 자세한 내용은 Amazon Athena 사용 설명서의 지원되는 SerDes 및 데이터 형식을 참조하십시오. 방대한 로그 파일 세트를 분석하려면 Amazon EMR 클러스터를 실행하여 페타바이트 규모의 분석을 실행할 수 있습니다.

파트너와 타사에서 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 각 경로에 이 수명 주기 관리 기능을 각기 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 액세스 티어로 자동으로 이동하여 비용을 최적화하도록 설계되었습니다.

정기적인 검토 시행: 워크로드 모니터링이 구현되는 방식을 자주 검토하고 중요한 이벤트 및 변경 사항에 따라 업데이트합니다.

효과적인 모니터링의 기반은 주요 비즈니스 지표입니다. 비즈니스 우선 순위가 변경됨에 따라 이러한 지표가 워크로드에 반영되는지 확인하십시오.

모니터링을 감사하면 애플리케이션이 가용성 목표를 달성하는 시기를 확인하는 데 도움이 됩니다. 근본 원인 분석을 수행하려면 장애 발생 시에 수행된 작업을 검색하는 기능이 필요합니다. AWS는 인시던트 중에 서비스 상태를 추적하는 데 사용할 수 있는 다음과 같은 서비스를 제공합니다.

- **Amazon CloudWatch Logs:** 로그를 저장하고 해당 내용을 검사할 수 있는 서비스입니다.
- **Amazon CloudWatch Logs Insights**: 대량 로그를 몇 초 만에 분석할 수 있는 완전관리형 서비스입니다. 이 서비스는 빠른 대화형 쿼리 및 시각화를 제공합니다.
- **AWS Config:** 다양한 시점에서 사용된 AWS 인프라를 확인할 수 있습니다.
- **AWS CloudTrail:** 특정 시간에 호출된 AWS API 및 기준으로 사용된 원칙을 확인할 수 있는 서비스입니다.

AWS에서는 주간 회의를 진행하여 운영 성능을 검토하고 팀 간에 파악한 내용을 공유합니다. AWS에는 많은 팀이 있기 때문에 검토할 워크로드를 무작위로 선택하는 The Wheel을

만들었습니다. 운영 성능 검토 및 지식 공유를 위한 정기 케이던스를 설정하면 운영 팀의 성과를 개선하는 역량을 발전시킬 수 있습니다.

시스템을 통한 엔드 투 엔드 요청 추적 모니터링: 개발자는 AWS X-Ray 또는 타사 도구를 사용하여 분산 시스템을 보다 쉽게 분석하고 디버깅하여 애플리케이션과 기반 서비스의 성능을 파악할 수 있습니다.

리소스

설명서

- Amazon CloudWatch 지표 사용
- Canary(Amazon CloudWatch Synthetics)
- Amazon CloudWatch Logs Insights 샘플 쿼리
- AWS Systems Manager Automation
- AWS X-Ray란 무엇일까요?
- Amazon CloudWatch Synthetics 및 AWS X-Ray를 사용한 디버깅
- Amazon Builders' Library: 운영 가시성을 위한 분산 시스템 계측

수요 변경에 따라 조정되는 워크로드 설계

확장 가능한 워크로드는 리소스를 자동으로 추가하거나 제거하여 특정 시기의 수요에 리소스 공급을 맞출 수 있는 탄력성을 제공합니다.

리소스를 확보하거나 조정할 때 자동화 사용: 손상된 리소스를 교체하거나 워크로드를 조정할 때 Amazon S3 및 AWS Auto Scaling과 같은 관리형 AWS 서비스를 사용하여 프로세스를 자동화합니다. 타사 도구 및 AWS SDK를 사용하여 크기를 자동 조정할 수도 있습니다.

관리형 AWS 서비스에는 Amazon S3, Amazon CloudFront, AWS Auto Scaling, AWS Lambda, Amazon DynamoDB, AWS Fargate 및 Amazon Route 53가 포함됩니다.

AWS Auto Scaling을 사용하면 손상된 인스턴스를 감지하고 교체할 수 있습니다. 또한 Amazon EC2 인스턴스 및 스팟 플릿, Amazon ECS 작업, Amazon DynamoDB 테이블 및 인덱스와 Amazon Aurora 복제본에 대한 조정 계획을 구축할 수도 있습니다.

EC2 인스턴스 또는 EC2 인스턴스에서 호스팅되는 Amazon ECS 컨테이너의 크기를 조정할 때는 여러 가용 영역(3개 이상 권장)을 사용하고 용량을 추가하거나 제거하여 이러한 가용 영역 간의 균형을 유지해야 합니다.

AWS Lambda를 사용하는 경우에는 자동으로 조정됩니다. 함수에 대한 이벤트 알림이 수신될 때마다 AWS Lambda가 컴퓨팅 플릿 내에서 여유 용량을 신속하게 찾고 할당된 동시성까지 코드를 실행합니다. 사용자는 필요한 동시성이 특정 Lambda와 Service Quotas에 구성되어 있는지 확인해야 합니다.

Amazon S3는 높은 요청 속도를 처리하도록 자동으로 확장됩니다. 예를 들어 애플리케이션은 버킷에서 접두사마다 초당 3,500개 이상의 PUT/COPY/POST/DELETE 및 5,500개의 GET/HEAD 요청을 전송할 수 있습니다. 버킷의 접두사 수에는 제한이 없습니다. 읽기를 병렬화하여 읽기 또는 쓰기 성능을 높일 수 있습니다. 예를 들어 Amazon S3 버킷에서 10개의 접두사를 생성하여 읽기를 병렬화하는 경우 읽기 성능을 초당 55,000개의 읽기 요청으로 확장할 수 있습니다.

Amazon CloudFront 또는 신뢰할 수 있는 콘텐츠 전송 네트워크를 구성해서 사용하십시오. CDN(콘텐츠 전송 네트워크)을 사용하면 최종 사용자 입장에서는 응답 시간을 단축할 수 있으며 워크로드를 불필요하게 확장할 수 있는 콘텐츠 요청을 처리할 수 있습니다.

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 리소스를 조정하도록 알리거나 자동화를 트리거하여 자동으로 리소스를 조정합니다.

워크로드에 적절하게 수동으로 규모를 조정할 수 있습니다. 예를 들어 콘솔 또는 AWS CLI를 통해 Auto Scaling 그룹의 EC2 인스턴스 수를 변경하거나 DynamoDB 테이블의 처리량을 수정할 수 있습니다. 하지만 가능한 경우에는 항상 자동화를 사용해야 합니다(**워크로드 확장 또는 축소 시 자동화 사용** 참조).

워크로드에 추가 리소스가 필요한 것으로 감지될 때 리소스 확보: 사전에 리소스를 확장하여 수요를 충족하고 가용성에 미치는 영향을 방지합니다.

많은 AWS 서비스가 수요에 맞춰 자동으로 확장됩니다(**워크로드 확장 또는 축소 시 자동화 사용** 참조). EC2 인스턴스 또는 Amazon ECS 클러스터를 사용하는 경우 워크로드 수요에 해당하는 사용량 지표에 따라 이러한 자동 조정이 수행되도록 구성할 수 있습니다. Amazon EC2의 경우 평균 CPU 사용률, 로드 밸런서 요청 수 또는 네트워크 대역폭을 사용하여 EC2 인스턴스를 스케일아웃(또는 스케일인)할 수 있습니다. Amazon ECS의 경우 평균 CPU 사용률, 로드 밸런서 요청 수 및 메모리 사용률을 사용하여 ECS 작업을 스케일아웃(또는 스케일인)할 수 있습니다. AWS에서 Target Auto Scaling을 사용하면 Autoscaler가 가정용 온도 조절기처럼 작동하여 리소스를 추가하거나 제거함으로써 지정한 목표 값(예: 70%의 CPU 사용률)을 유지합니다.

또한 AWS Auto Scaling은 기계 학습을 사용하여 각 리소스의 기간별 워크로드를 분석하고 향후 2일간의 로드를 주기적으로 예측하는 Predictive Auto Scaling을 수행할 수도 있습니다.

리틀의 법칙은 필요한 컴퓨팅 인스턴스(EC2 인스턴스, 동시 Lambda 함수 등) 수를 계산하는 데 도움이 됩니다.

$$L = \lambda W$$

L = 인스턴스 수(또는 시스템의 평균 동시성)

λ = 요청이 도착하는 평균 속도(요청/초)

W = 시스템이 각 요청에 소비하는 평균 시간(초)

예를 들어 100rps에서 각 요청을 처리하는 데 0.5초가 걸리는 경우 수요를 따라가려면 50개의 인스턴스가 필요합니다.

워크로드에 대한 로드 테스트: 확장 작업에서 워크로드 요구 사항이 충족되는지 여부를 측정하는 로드 테스트 방법론을 채택합니다.

지속적인 로드 테스트를 수행하는 것이 중요합니다. 로드 테스트에서는 워크로드의 한계점을 찾고 성능을 테스트해야 합니다. AWS를 사용하면 프로덕션 워크로드의 규모를 모델링하는 임시 테스트 환경을 손쉽게 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 몇 분의 일에 불과한 가격으로 실제 환경을 시뮬레이션할 수 있습니다.

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

리소스

설명서

- AWS Auto Scaling: 조정 계획 작동 방식
- Amazon EC2 Auto Scaling이란 무엇입니까?
- DynamoDB Auto Scaling을 통한 처리량 자동 관리
- Amazon CloudFront란?
- Distributed Load Testing on AWS: 수천 명의 연결된 사용자 시뮬레이션
- AWS Marketplace: Auto Scaling과 함께 사용할 수 있는 제품
- APN 파트너: 자동화된 컴퓨팅 솔루션의 생성을 지원할 수 있는 파트너

외부 링크

- Telling Stories About Little's Law

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 해결하기가 어려워집니다.

배포와 같은 표준 활동에 런북 사용: 런북은 특정 결과를 달성하기 위해 미리 정의된 단계입니다. 수동 또는 자동으로 표준 활동을 수행할 때는 런북을 사용하십시오. 워크로드 배포, 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 자동화된 실행을 트리거합니다.

고도로 자동화된 정교한 워크로드의 경우에도 실전 연습을 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.

배포의 일부로 기능 테스트 통합: 기능 테스트는 자동화된 배포의 일부로 실행됩니다. 성공 기준이 충족되지 않으면 파이프라인이 중지되거나 롤백됩니다.

이러한 테스트는 파이프라인에서 프로덕션 전에 준비되는 사전 프로덕션 환경에서 실행됩니다. 이 작업을 배포 파이프라인의 일부로 수행하는 것이 가장 좋습니다.

배포의 일부로 복원력 테스트 통합: 복원력 테스트는 카오스 엔지니어링의 일부로 사전 프로덕션 환경에서 자동화된 배포 파이프라인에 포함되어 실행됩니다.

이러한 테스트는 프로덕션 전에 파이프라인에서 준비되고 실행됩니다. 또한 프로덕션 환경에서도 실전 연습의 일부로 실행되어야 합니다.

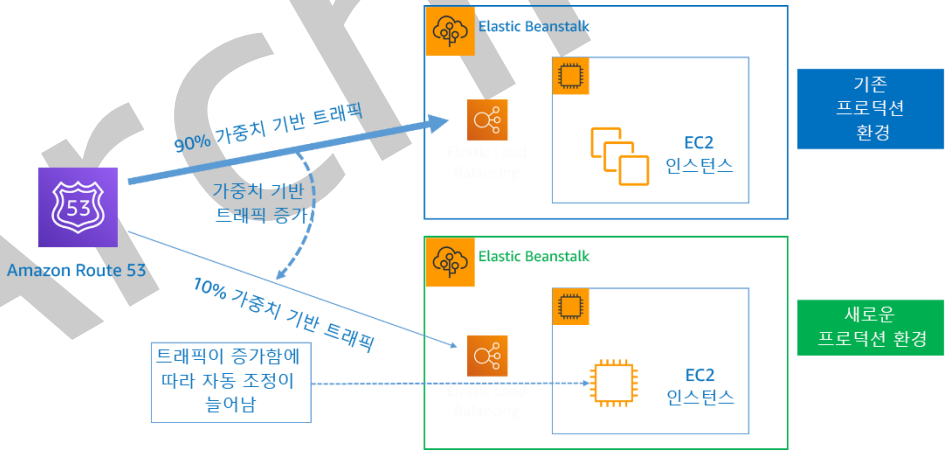
변경 불가능한 인프라를 사용하여 배포: 프로덕션 시스템의 현재 위치에서 업데이트, 보안 패치 또는 구성 변경이 발생하지 않도록 규정하는 모델입니다. 변경이 필요한 경우 아키텍처가 새 인프라에 구축되고 프로덕션에 배포됩니다.

변경 불가능한 인프라 패러다임의 가장 일반적인 구현은 변경 불가능한 서버입니다. 즉, 서버에 업데이트 또는 수정이 필요한 경우 이미 사용 중인 서버를 업데이트하는 대신 새 서버가 배포됩니다. 따라서 SSH를 통해 서버에 로그인하고 소프트웨어 버전을 업데이트하는 대신 애플리케이션의 모든 변경은 코드 리포지토리에 소프트웨어를 푸시(예: git push)하는 것으로 시작됩니다. 변경이 불가능한 인프라에서는 변경이 허용되지 않으므로 배포된 시스템의 상태를 확신할 수 있습니다. 변경이 불가능한 인프라는 본질적으로 더 일관되고 안정적이며 예측 가능하며 소프트웨어 개발 및 운영의 여러 측면을 간소화합니다.

변경이 불가능한 인프라에서 애플리케이션을 배포할 때는 Canary 또는 블루/그린 배포를 사용합니다.

Canary 배포는 일반적으로 단일 서비스 인스턴스에서 실행되는 새 버전(Canary)으로 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. Canary 배포를 지원하는 배포 구성을 사용하여 AWS Code Deploy를 구성할 수 있습니다.

블루-그린 배포는 Canary 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 Code Deploy와 AWS Elastic Beanstalk를 구성할 수 있습니다.



AWS Elastic Beanstalk 및 Amazon Route 53를 사용한 블루/그린 배포

변경 불가능한 인프라의 이점:

- 구성 드리프트 감소: 기본의 알려진 버전 제어 구성에서 서버를 자주 교체하면 인프라가 알려진 상태로 재설정되므로 구성 드리프트가 방지됩니다.

- 간소화된 배포: 업그레이드를 지원할 필요가 없으므로 배포가 간소화됩니다. 업그레이드는 단지 새로운 배포일 뿐입니다.
- 안정성 있는 원자 단위 배포: 배포가 성공적으로 완료되거나 아무 것도 변경되지 않습니다. 따라서 배포 프로세스의 신뢰도가 개선됩니다.
- 빠른 롤백 및 복구 프로세스를 통해 배포 안전성 개선: 이전 작업 버전이 변경되지 않으므로 배포가 더 안전합니다. 오류가 감지되면 롤백할 수 있습니다.
- 일관된 테스트 및 디버깅 환경: 모든 서버에 동일한 이미지가 사용되므로 환경 간에 차이가 없습니다. 하나의 빌드가 여러 환경에 배포됩니다. 또한 일관되지 않은 환경이 방지되고 테스트 및 디버깅이 간소화됩니다.
- 확장성 개선: 서버가 기본 이미지를 사용하고 일관적이며 반복 가능하므로 자동 조정이 간단합니다.
- 간소화된 도구 체인: 프로덕션 소프트웨어 업그레이드를 관리하는 구성 관리 도구를 제거할 수 있으므로 도구 체인이 간소화됩니다. 서버에 추가 도구 또는 에이전트가 설치되지 않습니다. 변경은 기본 이미지에 수행되고 테스트된 후 롤아웃됩니다.
- 보안 강화: 서버에 대한 모든 변경을 거부하면 인스턴스에서 SSH를 비활성화하고 키를 제거할 수 있습니다. 이렇게 하면 공격 벡터가 줄어들어 조직의 보안 태세를 개선할 수 있습니다.

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/배포, 용량 추가/제거, 데이터 마이그레이션 포함)에 자동화를 사용해야 합니다. AWS CodePipeline을 사용하면 워크로드를 해제하는 데 필요한 단계를 관리할 수 있습니다. 여기에는 AWS CodeDeploy를 사용하여 Amazon EC2 인스턴스, 온프레미스 인스턴스, 서버리스 Lambda 함수 또는 Amazon ECS 서비스에 대한 애플리케이션 코드 배포를 자동화하는 배포 상태가 포함됩니다.

권장 사항
일반적인 통념으로는 가장 어려운 작업 절차의 고리에 사람을 두는 것이 좋다고 하지만 AWS에서는 바로 그런 이유로 가장 어려운 절차를 자동화할 것을 권장합니다.

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 Canary 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: 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.

ORR(운영상 준비 검토)

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

권장 사항
초기 프로덕션 사용 전에 애플리케이션에 대한 ORR(운영 준비 상태 검토)을 시행한 후 주기적으로 실시합니다.

리소스

동영상

- AWS Summit 2019: CI/CD on AWS

설명서

- AWS CodePipeline이란 무엇입니까?
- CodeDeploy란 무엇입니까?
- 블루/그린 배포 개요
- Deploying Serverless Applications Gradually
- Amazon Builders' Library: 배포 중 롤백 안전 보장
- Amazon Builders' Library: 지속적 전달을 통한 신속한 배포
- AWS Marketplace: 배포 자동화에 사용할 수 있는 제품
- APN 파트너: 자동화된 배포 솔루션의 생성을 지원할 수 있는 파트너

실습

- Well-Architected lab: Level 300: Testing for Resiliency of EC2 RDS and S3

외부 링크

- CanaryRelease

장애 관리

> 장애는 기정 사실이며 시간이 지나면 라우터부터 하드 디스크, 운영 체제부터 메모리 장치 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 사실이다. - **Werner Vogels, CTO - Amazon.com**

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) 동안 99.999999999%의 객체 내구성을 제공합니다. 어떤 클라우드 공급자를 사용하든 워크로드에 영향을 주는 장애가 발생할 가능성이 있습니다. 따라서 워크로드 안정성을 유지하려면 복원력을 구현하기 위한 조치를 취해야 합니다.

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

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

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

데이터 백업

RTO(복구 시간 목표) 및 RPO(복구 시점 목표)에 대한 요구 사항을 충족하도록 데이터, 애플리케이션 및 구성을 백업합니다.

백업해야 하는 모든 데이터를 식별하고 백업하거나 원본의 데이터를 복제: Amazon S3를 여러 데이터 원본의 백업 대상으로 사용할 수 있습니다. Amazon EBS, Amazon RDS 및 Amazon DynamoDB와 같은 AWS 서비스에는 백업 생성 기능이 기본적으로 포함되어 있습니다. 또는 타사 백업 소프트웨어를 사용할 수도 있습니다. RPO 충족을 위해 다른 소스에서 데이터를 재현할 수 있는 경우에는 백업이 필요하지 않을 수도 있습니다.

온프레미스 데이터는 Amazon S3 버킷 및 AWS Storage Gateway를 사용하여 AWS 클라우드로 백업될 수 있습니다. 저렴하고 시간에 민감하지 않은 클라우드 스토리지를 제공하는 Amazon S3 Glacier 또는 S3 Glacier Deep Archive를 사용하여 백업 데이터를 보관할 수 있습니다.

다른 소스의 데이터를 복제할 수도 있습니다. 예를 들어 Amazon S3의 데이터를 데이터 웨어하우스(예: Amazon Redshift) 또는 MapReduce 클러스터(예: Amazon EMR)로 로드하여 해당 데이터를 분석할 수 있는데 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.

백업 보안 및 암호화: AWS Identity and Access Management(IAM)와 같은 인증 및 권한 부여를 사용하여 액세스를 감지하고 암호화를 사용하여 데이터 무결성 손상을 감지합니다.

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

Amazon RDS의 경우 데이터베이스를 암호화하도록 선택하면 백업도 암호화됩니다. DynamoDB 백업은 항상 암호화됩니다.

데이터 백업 자동 수행: 정기적인 일정 또는 데이터 세트의 변경에 따라 백업이 자동으로 생성되도록 구성합니다. RDS 인스턴스, EBS 볼륨, DynamoDB 테이블 및 S3 객체에 대해 모두 자동 백업을 구성할 수 있습니다. AWS Marketplace 솔루션 또는 타사 솔루션을 사용해도 됩니다.

Amazon Data Lifecycle Manager는 EBS 스냅샷을 자동화하는 데 사용할 수 있습니다. Amazon RDS와 Amazon DynamoDB는 특정 시점 복구를 통해 지속적 백업을 지원합니다. 활성화하면 Amazon S3 버전 관리가 자동으로 수행됩니다.

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 다른 AWS 리전의 다른 버킷에 자동으로 복제할 수 있습니다.

정기적인 데이터 복구를 수행하여 백업 무결성 및 프로세스 확인: 복구 테스트를 수행하여 백업 프로세스 구현이 RTO(목표 복구 시간) 및 RPO(목표 복구 시점)를 충족하는지 확인합니다.

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

또한 Amazon RDS와 Amazon DynamoDB는 PITR(특정 시점 복구)을 지원합니다. 연속 백업을 사용하면 데이터 세트를 지정된 날짜 및 시간의 상태로 복원할 수 있습니다.

리소스

동영상

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

설명서

- AWS Backup이란 무엇인가요?
- Amazon S3: 암호화를 사용하여 데이터 보호
- AWS의 백업 암호화
- DynamoDB용 온디맨드 백업 및 복원
- EFS-to-EFS Backup
- AWS Marketplace: 백업에 사용할 수 있는 제품
- APN 파트너: 백업을 지원할 수 있는 파트너

실습

- Well-Architected lab: Level 200: Testing Backup and Restore of Data

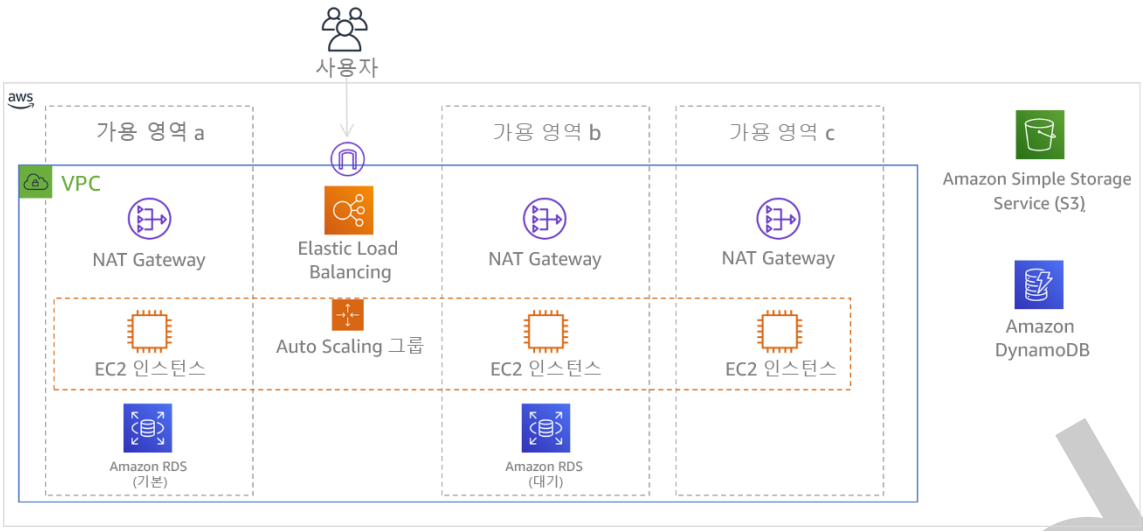
장애 격리를 사용하여 워크로드 보호

장애 격리 경계는 워크로드 내부 장애의 영향을 제한된 수의 구성 요소로 제한합니다. 경계 외부의 구성 요소는 장애가 발생하더라도 영향을 받지 않습니다. 다수의 장애 격리 경계를 사용하여 워크로드에 미치는 영향을 제한할 수 있습니다.

워크로드를 여러 위치에 배포: 워크로드 데이터와 리소스를 여러 가용 영역에 분산하거나 필요한 경우 AWS 리전 전체에 분산합니다. 필요에 따라 다양한 위치를 사용할 수 있습니다.

AWS의 서비스 설계 관련 기본 원칙 중 하나는 기본 물리적 인프라에서 단일 장애 지점이 없어야 한다는 것입니다. 이 원칙을 준수하려면 가용 영역 여러 개를 사용하며 단일 영역에서 장애가 발생해도 복원이 가능한 소프트웨어와 시스템을 구축해야 합니다. 마찬가지로, 시스템은 단일 컴퓨팅 노드, 단일 스토리지 볼륨 또는 단일 데이터베이스 인스턴스에서 장애가 발생하더라도 복원 가능하도록 구축됩니다. 중복 구성 요소를 사용하는 시스템을 구축할 때는 구성 요소가 독립적(AWS 리전의 경우 자율적)으로 작동해야 합니다. 중복 구성 요소를 사용한 이론적 가용성 계산에서 얻을 수 있는 이점은 이것이 사실인 경우에만 유효합니다.

가용 영역: AWS 리전은 독립적으로 설계된 두 개 이상의 가용 영역으로 구성됩니다. 화재, 홍수, 태풍 등의 자연 재해로 인한 상관 장애 시나리오를 방지하기 위해 각 가용 영역은 서로 멀리 떨어져 있도록 분리됩니다. 각 가용 영역에는 독립된 물리적 인프라(다목적 전원에 대한 전용 연결, 대기 백업 전원, 독립 기계 서비스, 가용 영역 내/외부의 독립네트워크 연결)가 포함되어 있습니다. 가용 영역은 지리적으로는 분리되지만 같은 지역에 배치됩니다. 따라서 애플리케이션 지연 시간이 지나치게 길어지지 않도록 유지하면서 데이터베이스 간의 복제와 같은 동기 데이터 복제를 수행할 수 있습니다. 그러면 고객은 액티브/액티브 또는 액티브/대기 구성에서 가용 영역을 사용할 수 있습니다. 가용 영역은 서로 독립되어 있으므로 여러 영역을 사용하는 경우 애플리케이션 가용성이 높아집니다. EC2 인스턴스 데이터 영역을 비롯한 일부 AWS 서비스는 엄격한 영역 서비스로 배포되며, 전체 가용 영역과 동일한 상태로 설정됩니다. 이러한 서비스는 특정 가용 영역 내의 리소스(인스턴스, 데이터베이스 및 기타 인프라)를 독립적으로 작동하는 데 사용됩니다. AWS의 리전에서는 오랫동안 여러 가용 영역이 제공되어 왔습니다.



3개의 가용 영역에 배포된 다중 계층 아키텍처. Amazon S3와 Amazon DynamoDB는 항상 자동으로 다중 AZ에 유지됩니다. 또한 ELB는 3개 영역 모두에 배포됩니다.

AWS 제어 영역에서는 대개 전체 리전(여러 가용 영역) 내의 리소스를 관리하는 기능이 제공되지만, Amazon EC2 및 Amazon EBS 등의 특정 제어 영역에는 단일 가용 영역으로 결과를 필터링하는 기능도 있습니다. 이처럼 결과가 필터링되면 요청은 지정된 가용 영역에서만 처리되므로 다른 가용 영역이 중단될 가능성이 낮아집니다. 반면 리전별 AWS 서비스는 설정된 가용성 설계 목표를 달성하기 위해 내부적으로 액티브/액티브 구성에서 여러 가용 영역을 사용합니다.

권장 사항

하나의 **가용 영역**이 중단되는 동안 애플리케이션이 제어 영역 API의 가용성에 의존하는 경우, 각 API 요청에서 API 필터를 사용하여 단일 가용 영역에 대한 결과를 요청합니다(예: DescribeInstances 사용).

AWS 로컬 영역은 서브넷 및 EC2 인스턴스와 같은 영역 단위 AWS 리소스에 대한 배치 위치로 선택될 수 있다는 점에서 해당 AWS 리전 내의 가용 영역과 유사하게 작동합니다. 이러한 로컬 영역은 연결된 AWS 리전이 아니라 현재 AWS 리전이 없는 대규모 인구, 산업 및 IT 센터에 가까운 곳에 위치한다는 점에서 특별합니다. 그럼에도 불구하고 로컬 영역의 로컬 워크로드와 AWS 리전에서 실행 중인 워크로드 간에는 고대역폭의 안전한 연결이 유지됩니다. 지연 시간이 짧아야

하는 요구 사항을 충족하기 위해 사용자에게 더 가까운 위치에 워크로드를 배포하려면 AWS 로컬 영역을 사용해야 합니다.

Amazon 글로벌 엣지 네트워크는 전 세계 도시의 엣지 로케이션으로 구성됩니다. Amazon CloudFront는 이 네트워크를 사용하여 최종 사용자에게 더 짧은 지연 시간으로 콘텐츠를 전송합니다. AWS Global Accelerator를 사용하면 이러한 엣지 로케이션에 워크로드 엔드포인트를 생성하여 사용자와 가까운 AWS 글로벌 네트워크로의 온보딩을 제공할 수 있습니다. Amazon API Gateway는 CloudFront 배포를 사용하여 엣지 최적화 API 엔드포인트를 활성화함으로써 가장 가까운 엣지 로케이션을 통한 클라이언트 액세스를 촉진합니다.

AWS 리전: 리전은 자율적으로 작동하도록 설계되므로 다중 리전 접근 방식을 사용하려면 각 리전에 서비스의 전용 복사본을 배포해야 합니다.

권장 사항
단일 AWS 리전 안에서 다중 AZ 전략을 사용하여 워크로드에 대한 대부분의 안정성 목표를 충족할 수 있습니다. 다중 리전에 대한 요구 사항이 있는 워크로드에 대해서만 다중 리전 아키텍처를 고려해야 합니다.

AWS는 교차 리전에서 서비스를 운영할 수 있는 기능을 제공합니다. 예를 들어 Amazon Aurora Global Database, Amazon DynamoDB 글로벌 테이블, Amazon S3의 리전 간 복제, Amazon RDS의 리전 간 읽기 전용 복제본과 다양한 스냅샷 및 Amazon Machine Image(AMI)를 다른 리전으로 복사할 수 있는 기능이 있습니다. 그러나 이러한 기능은 리전의 자율권을 보존하는 방식으로 수행됩니다. 이 방식의 예외는 Amazon CloudFront 및 Amazon Route 53와 같이 글로벌 엣지 제공 기능을 제공하는 서비스와, AWS Identity and Access Management(IAM) 서비스용 제어 영역 등의 몇 가지뿐입니다. 대다수 서비스는 전적으로 단일 리전 내에서 작동합니다.

온프레미스 데이터 센터: 온프레미스 데이터 센터에서 실행되는 워크로드의 경우 가능하면 하이브리드 환경을 설계합니다. AWS Direct Connect는 온프레미스에서 AWS로 연결되는 전용 네트워크 연결을 제공하므로 두 환경에서 모두 워크로드를 실행할 수 있습니다.

또 다른 옵션은 AWS Outposts를 사용하여 온프레미스에서 AWS 인프라 및 서비스를 실행하는 것입니다. AWS Outposts는 AWS 인프라, AWS 서비스, API 및 도구를 온프레미스 데이터 센터로 확장하는 완전관리형 서비스입니다. AWS 클라우드에서 사용되는 것과 동일한 하드웨어 인프라가 데이터 센터에 설치됩니다. Outposts는 가장 가까운 AWS 리전에 연결됩니다. 그런 다음에는 Outpost를 사용하여 지연 시간이 짧아야 하거나 로컬 데이터 처리 요구 사항이 있는 워크로드를 지원할 수 있습니다.

단일 위치로 제한된 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 구현해야 합니다.

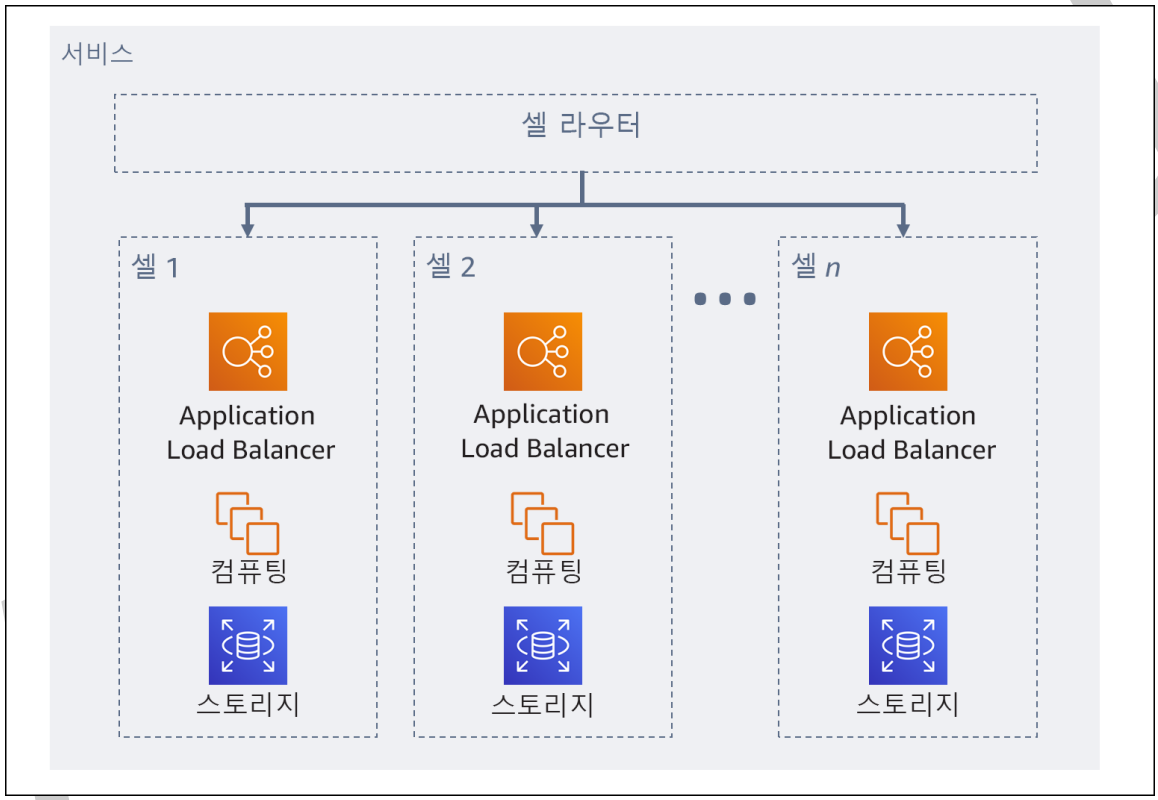
기술적 제약으로 인해 워크로드를 여러 위치에 배포하는 모범 사례를 따를 수 없다면 복원력을 달성할 수 있는 대체 경로를 구현해야 합니다. 이러한 경우를 위해 필요한 인프라를 다시 생성하고, 애플리케이션을 다시 배포하고, 필요한 데이터를 다시 생성하는 기능을 자동화해야 합니다.

예를 들어 Amazon EMR은 지정된 클러스터의 모든 노드를 동일한 가용 영역에서 시작합니다. 동일한 영역에서 클러스터를 실행하면 데이터 액세스 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 마스터 노드를 프로비저닝할 수 있습니다. EMRFS(EMR 파일 시스템)를 사용하면 EMR의 데이터를 Amazon S3에 저장한 다음 여러 가용 영역 또는 AWS 리전에 걸쳐 복제할 수 있습니다.

Amazon Redshift와 마찬가지로 클러스터는 기본적으로 사용자가 선택한 AWS 리전 내에서 임의로 선택된 가용 영역에 프로비저닝됩니다. 모든 클러스터 노드는 동일한 영역에 프로비저닝됩니다.

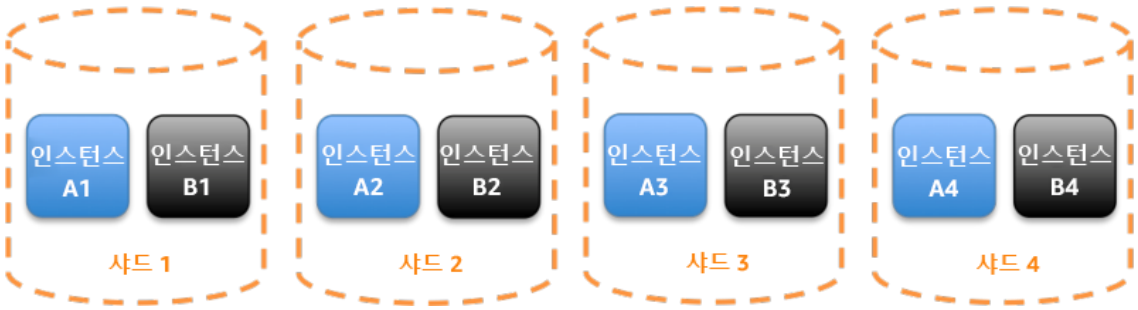
격벽 아키텍처 사용: 이 패턴은 선박의 격벽처럼 장애를 요청/사용자의 소수의 하위 집합으로 제한하여 손상된 요청 수를 제한하고 대부분의 요청은 오류 없이 계속될 수 있도록 합니다. 데이터에 대한 격벽은 일반적으로 파티션 또는 샤드로 불리며 서비스에 대한 격벽은 셀이라고 합니다.

셀 기반 아키텍처에서 각 셀은 서비스의 완전한 독립 인스턴스이며 최대 크기가 고정되어 있습니다. 로드가 증가하면 셀을 추가하는 방법으로 워크로드 규모를 늘립니다. 파티션 키는 수신 트래픽에서 요청을 처리할 셀을 결정하는 데 사용됩니다. 모든 장애는 장애가 발생한 단일 셀로 제한되므로 다른 셀은 오류 없이 계속되고 손상된 요청의 수가 제한됩니다. 따라서 셀 간의 상호 작용을 최소화하고 각 요청에서 복잡한 매핑 서비스를 실행하지 않아도 되도록 적절한 파티션 키를 확인해야 합니다. 복잡한 매핑을 실행해야 하는 서비스에서는 문제가 매핑 서비스로 이전될 뿐이며, 셀 간 상호 작용을 수행해야 하는 서비스에서는 개별 셀의 독립성 수준이 낮아지므로 셀의 독립성이 유지되는 경우 개선 가능한 가용성의 수준도 낮아집니다.



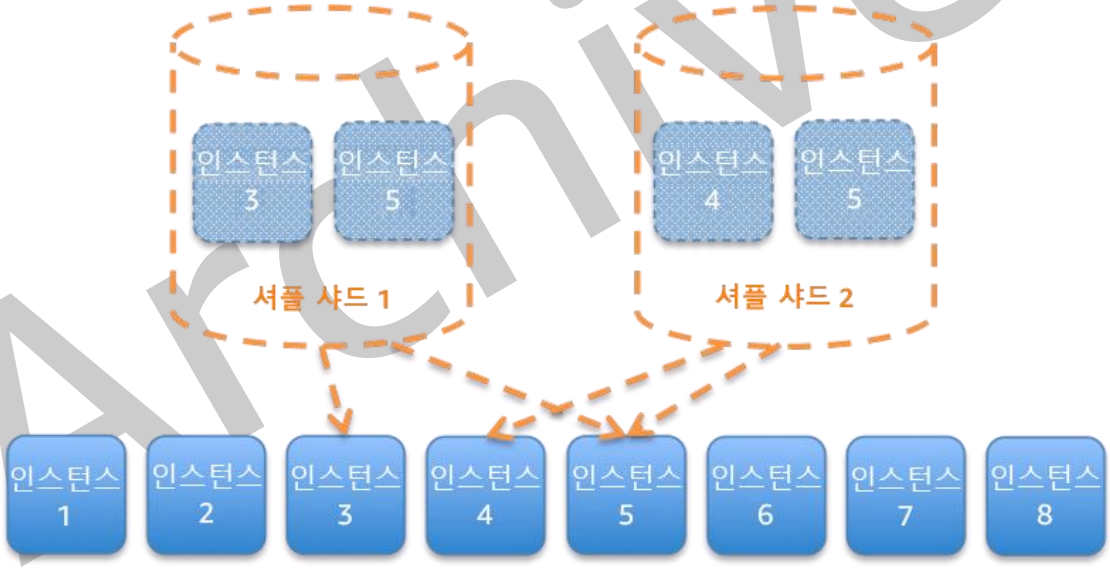
셀 기반 아키텍처

Colm MacCarthaigh의 AWS 블로그 게시물에 Amazon Route 53에서 셔플 샤딩 개념을 사용하여 고객 요청을 샤드로 격리하는 방법이 설명되어 있습니다. 이 사례에서 샤드는 2개 이상의 셀로 구성됩니다. 고객(또는 리소스 또는 격리하려는 대상)의 트래픽은 파티션 키에 따라 할당된 샤드로 라우팅됩니다. 샤드당 2개씩 8개의 셀이 있고 샤드 4개에 고객이 나누어져 있는 경우 문제가 발생하면 25%의 고객이 영향을 받게 됩니다.



각각 2개의 셀로 구성된 기존 샤드 4개에 나누어져 있는 서비스

셔플 샤딩을 사용하면 각각 2개의 셀로 구성된 가상 샤드를 생성하고 고객을 이러한 가상 샤드 중 하나에 할당할 수 있습니다. 문제가 발생하면 전체 서비스의 1/4이 손실될 수 있지만 이 방식의 고객 또는 리소스 할당은 셔플 샤딩의 영향 범위가 25%보다 크게 낮아짐을 의미합니다. 8개의 셀에는 2개 셀로 구성된 28개의 고유한 조합이 있습니다. 즉, 28개의 셔플 샤드(가상 샤드)가 가능합니다. 고객이 수백 또는 수천 명이고 각 고객을 셔플 샤드에 할당하는 경우 문제의 영향 범위는 1/28에 불과합니다. 이는 일반 샤딩보다 7배 더 뛰어난 수치입니다.



각각 2개의 셀로 구성된 28개의 셔플 샤드(가상 공유)에 나누어져 있는 서비스(28개 중 2개의 셔플 샤드만 표시되어 있음)

샤드는 셀 외에 서버, 대기열 또는 기타 리소스에도 사용될 수 있습니다.

리소스

동영상

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications(ARC209-R2)
- Shuffle-sharding: AWS re:Invent 2019: Introducing The Amazon Builders' Library(DOP328)
- AWS re:Invent 2018: How AWS Minimizes the Blast Radius of Failures(ARC338)
- AWS re:Invent 2019: Innovation and operation of the AWS global network infrastructure(NET339)

설명서

- What is AWS Outposts?
- 전역 테이블: DynamoDB를 사용한 다중 리전 복제
- AWS 로컬 영역 FAQ
- AWS 글로벌 인프라
- 리전, 가용 영역 및 로컬 영역
- Amazon Builders' Library: 셔플 샤딩을 사용한 워크로드 격리

구성 요소 장애를 견디도록 워크로드 설계

고가용성 및 낮은 MTTR(평균 복구 시간)이 요구되는 워크로드는 복원력을 고려하여 설계되어야 합니다.

워크로드의 모든 구성 요소를 모니터링하여 장애 감지: 워크로드 상태를 지속적으로 모니터링하여 성능 저하 또는 완전한 장애가 발생하는 즉시 수동 및 자동화된 시스템으로 이를 인식할 수 있도록 합니다. 비즈니스 가치를 기반으로 KPI(핵심 성능 지표)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 비즈니스 가치를 탐지 및 수정 전략의 핵심 척도로 사용해야 합니다.

영향을 받지 않은 위치의 정상 리소스로 장애 조치: 위치 장애가 발생하는 경우 정상 위치의 데이터 및 리소스로 계속해서 요청을 처리할 수 있어야 합니다. 다중 영역 워크로드의 경우 Elastic Load Balancing 및 AWS Auto Scaling과 같은 AWS 서비스로 가용 영역에 로드를 분산할 수 있으므로 이 작업이 쉽습니다. 다중 리전 워크로드의 경우 이 작업이 더 복잡합니다. 예를 들어 리전 간 읽기 전용 복제본을 사용하여 데이터를 여러 AWS 리전에 배포할 수 있지만, 기본 위치에 장애가 발생할 경우 읽기 전용 복제본을 마스터로 승격하고 트래픽을 이 위치로 지정해야 합니다. Amazon Route 53와 AWS Global Accelerator는 AWS 리전 간에 트래픽을 라우팅하는 데 도움이 됩니다.

워크로드에서 Amazon S3 또는 Amazon DynamoDB와 같은 AWS 서비스를 사용하는 경우 이러한 서비스는 여러 가용 영역에 자동으로 배포됩니다. 장애가 발생하면 AWS 제어 영역이 자동으로 정상적인 위치로 트래픽을 라우팅합니다. Amazon RDS의 경우 구성 옵션으로 다중 AZ를 선택해야 합니다. 그러면 장애 시 AWS가 자동으로 트래픽을 정상 인스턴스로 보냅니다. Amazon EC2 인스턴스 또는 Amazon ECS 작업의 경우에는 배포할 가용 영역을 선택합니다. 그러면 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.

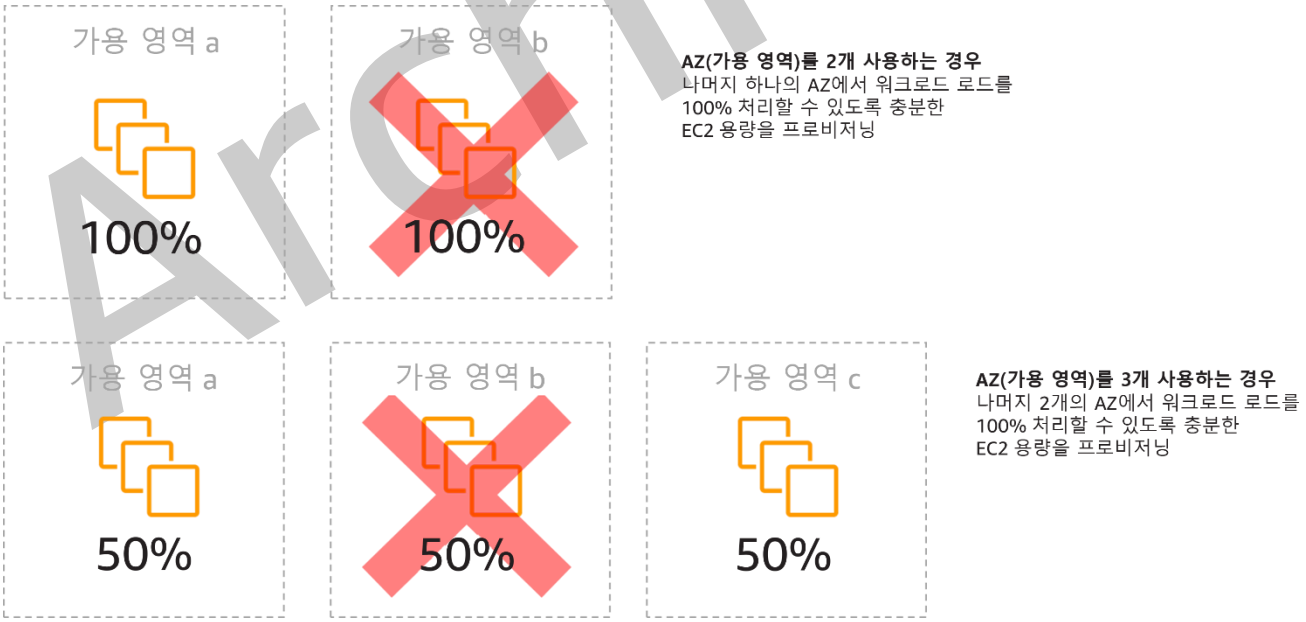
다중 리전 접근 방식(온프레미스 데이터 센터도 포함될 수 있음)의 경우 Amazon Route 53를 사용하여 인터넷 도메인을 정의하고, 트래픽이 정상적인 리전으로 라우팅되도록 상태 확인을 포함하는 라우팅 정책을 할당할 수 있습니다. 또는 AWS Global Accelerator가 제공하는 정적 IP 주소를 애플리케이션의 고정된 진입점으로 사용하고 인터넷 대신 AWS 글로벌 네트워크를 사용하여 선택한 AWS 리전의 엔드포인트로 라우팅하여 성능과 안정성을 개선할 수 있습니다.

AWS에서는 장애 복구를 고려한 서비스 설계 방식이 사용됩니다. 즉, 장애에서 복구하는 시간과 데이터에 대한 영향을 최소화하는 방식으로 서비스가 설계됩니다. AWS 서비스는 기본적으로 여러 복제본에 저장된 요청만 승인하는 데이터 스토어를 사용합니다. 이러한 서비스와 리소스에는 Amazon Aurora, Amazon Relational Database Service(Amazon RDS) Multi-AZ DB 인스턴스, Amazon S3, Amazon DynamoDB, Amazon Simple Queue Service(Amazon SQS), Amazon Elastic File System(Amazon EFS) 등이 있습니다. 이러한 서비스/리소스는 셀 기반 격리와 독립적인 가용 영역을 사용하도록 구성됩니다. AWS의 운영 절차에서는 자동화가

광범위하게 사용됩니다. 또한 서비스 중단 시 빠르게 복구할 수 있도록 교체 및 다시 시작 기능도 최적화됩니다.

정적 안정성을 사용하여 바이모달 동작 방지: 바이모달 동작은 워크로드가 정상 모드와 장애 모드에서 다른 동작을 보이는 것을 말합니다. 예를 들어 가용 영역에 장애가 발생할 경우 새 인스턴스를 시작하는 방법을 사용할 수 있습니다. 그러나 이 방법 대신 정적으로 안정적이고 한 모드에서만 작동하는 시스템을 구축해야 합니다. 한 영역이 제거된 경우 제거된 영역의 워크로드 로드를 처리하기에 충분한 인스턴스를 각 영역에 프로비저닝한 다음 Elastic Load Balancing 또는 Amazon Route 53 상태 확인을 사용하여 손상된 인스턴스에서 로드를 이동합니다.

컴퓨팅 배포(예: EC2 인스턴스 또는 컨테이너)에서 정적 안정성은 최고의 안정성을 제공합니다. 정적 안정성은 비용 문제와 비교하여 검토되어야 합니다. 컴퓨팅 파워를 적게 프로비저닝하고 장애 발생 시 새 인스턴스를 시작하는 방법이 비용은 더 저렴합니다. 그러나 대규모 장애(예: 가용 영역 장애)의 경우 이 접근 방식은 장애가 발생하기 전에 대비하는 것이 아니라 장애가 발생할 때 대응하는 방법에 의존하기 때문에 효율성이 떨어집니다. 따라서 워크로드의 안정성 요구 사항과 비용 요구 사항을 비교해야 합니다. 사용하는 가용 영역이 많을수록 정적 안정성을 유지하는 데 필요한 추가 컴퓨팅 용량은 줄어듭니다.



트래픽이 전환된 후에는 AWS Auto Scaling을 사용하여 장애가 발생한 영역의 인스턴스를 비동기식으로 교체하고 정상 영역에서 인스턴스를 시작합니다.

바이모달 동작의 또 다른 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 야기될 수 있습니다. 이 부정적인 피드백 루프는 워크로드의 가용성에 영향을 미칩니다. 대신 정적으로 안정적이며 한 모드에서만 작동하는 시스템을 구축해야 합니다. 일정한 작업을 수행하고 항상 고정된 케이던스에서 구성 상태를 새로 고치는 것이 정적으로 안정된 설계의 하나일 수 있습니다. 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 변경되고 장애를 초래할 가능성이 높으므로 허용해서는 안 됩니다.

모든 계층에서 복구 자동화: 장애가 감지되면 자동화된 기능을 사용하여 수정 작업을 수행합니다.

재시작 기능은 장애를 해결하는 데 사용할 수 있는 중요한 도구입니다. 분산 시스템에서 모범 사례는 앞서 설명한 것처럼 가능한 경우 서비스를 상태 비저장으로 만드는 것입니다. 이렇게 하면 재시작 시 데이터 손실 또는 가용성이 손실되는 것을 방지할 수 있습니다. 클라우드에서는 재시작의 일부로 전체 리소스(예: EC2 인스턴스 또는 Lambda 함수)를 대체할 수 있으며 이러한 대체는 일반적으로 필수적입니다. 재시작은 그 자체로 장애를 복구할 수 있는 단순하면서도 안정적인 방법입니다. 워크로드에는 다양한 유형의 장애가 발생합니다. 장애는 하드웨어, 소프트웨어, 통신 및 작업 과정에서 발생할 수 있습니다. 서로 다른 유형의 장애를 각각 격리, 식별 및 수정하는 새로운 메커니즘을 구성하는 대신 여러 범주의 장애를 동일한 복구 전략에 매핑하는 것이 좋습니다. 인스턴스는 하드웨어 결함, 운영 체제 버그, 메모리 누수 또는 기타 원인으로 인해 장애를 경험할 수 있습니다. 이러한 장애가 발생할 경우 각 상황에 맞춰진 수정 조치를 구축하는 대신 인스턴스 장애로 처리하십시오. 인스턴스를 종료하고 AWS Auto Scaling을 통해 인스턴스를 교체합니다. 나중에 환경 외부에서 장애 발생 리소스 분석을 수행할 수 있습니다.

또 다른 예는 네트워크 요청을 다시 시작하는 기능입니다. 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 사례”로 처리하는 대신 지수 백오프 및 지터를 통해 제한적으로 재시도하는 비슷한 전략이 적용됩니다.

재시작 기능은 ROC(복구 중심 컴퓨팅) 및 고가용성 클러스터 아키텍처에 포함된 복구 메커니즘입니다.

Amazon EventBridge를 사용하면 CloudWatch Alarms 또는 다른 AWS 서비스의 상태 변경과 같은 이벤트를 모니터링하고 필터링할 수 있습니다. 그런 다음 이벤트 정보를 기반으로 AWS Lambda(또는 다른 대상)를 트리거하여 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. AWS OpsWorks를 사용하는 경우 계층 수준에서 EC2 인스턴스의 자동 복구 기능을 구성할 수 있습니다.

대규모 교체(예: 전체 가용 영역이 손실됨)의 경우 한 번에 여러 개의 새 리소스를 확보하는 대신 정적 안정성을 통해 고가용성을 유지하는 것이 좋습니다.

이벤트가 가용성에 영향을 미치는 경우 알림 전송: 중대한 이벤트가 감지되면 이벤트로 인해 야기된 문제가 자동으로 해결된 경우에도 알림이 전송됩니다.

자동 복구를 사용하면 워크로드의 안정성을 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. 장애 발생을 기준으로 Amazon CloudWatch Alarms를 트리거할 수 있으며 자동화된 복구 작업의 실행을 기준으로 트리거할 수도 있습니다. 이메일을 보내거나 Amazon SNS 통합을 사용하여 타사 인시던트 추적 시스템에 인시던트를 기록하도록 CloudWatch Alarms를 구성할 수 있습니다.

리소스

동영상

- Static stability in AWS: AWS re:Invent 2019: Introducing The Amazon Builders' Library(DOP328)

설명서

- AWS OpsWorks: 자동 복구를 사용하여 실패한 인스턴스 대체

- Amazon EventBridge란 무엇입니까?
- Amazon Route 53: 라우팅 정책 선택
- What Is AWS Global Accelerator?
- Amazon Builders' Library: 가용 영역을 사용한 정적 안정성
- Amazon Builders' Library: 상태 확인 구현
- AWS Marketplace: 내결함성에 사용할 수 있는 제품
- APN 파트너: 내결함성 자동화를 지원할 수 있는 파트너

실습

- Well-Architected lab: Level 300: Implementing Health Checks and Managing Dependencies to Improve Reliability

외부 링크

- The Berkeley/Stanford Recovery-Oriented Computing (ROC) Project

안정성 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 확인하십시오. 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 수행하십시오.

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(화재 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.

인시던트 이후 분석 수행: 고객에게 영향을 주는 이벤트를 검토하고 발생 요인과 예방 조치 항목을 식별합니다. 이 정보를 사용하여 재발을 제한하거나 방지하는 완화 기능을 개발하고, 신속하고 효과적인 대응을 위한 절차를 개발합니다. 목표 대상에 맞게 적절히 원인과 교정 조치를 전달합니다.

기존 테스트에서 문제가 발견되지 않은 이유를 평가합니다. 테스트가 아직 없는 경우 이 사례에 대한 테스트를 추가합니다.

기능적 요구 사항 테스트: 여기에는 필수 기능을 검증하는 단위 테스트와 통합 테스트가 포함됩니다.

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. 이러한 변경 사항이 구축되고 테스트가 실행됩니다. 테스트가 완료되면 테스트를 위해 구축된 코드가 스테이징 서버에 배포됩니다. CodePipeline은 스테이징 서버에서 통합 또는 로드 테스트와 같은 더 많은 테스트를 실행합니다. 이러한 테스트가 성공적으로 완료되면 CodePipeline은 테스트되고 승인된 코드를 프로덕션 인스턴스에 배포합니다.

경험에 기반할 때 가장 중요한 테스트 중 하나는 고객 행동을 실행하고 시뮬레이션할 수 있는 가상 트랜잭션 테스트("Canary 테스트"라고도 하지만 Canary 배포와는 다름)입니다. 다양한 원격 위치에서 워크로드 엔드포인트에 대해 이러한 테스트를 지속적으로 실행하십시오. Amazon CloudWatch Synthetics를 사용하면 엔드포인트 및 API 모니터링을 위한 Canary를 생성할 수 있습니다.

조정 및 성능 요구 사항 테스트: 여기에는 워크로드가 조정 및 성능 요구 사항을 충족하는지 확인하는 로드 테스트가 포함됩니다.

클라우드에서는 워크로드에 대한 프로덕션 규모의 테스트 환경을 필요할 때 생성할 수 있습니다. 축소된 인프라에서 이러한 테스트를 실행하는 경우 관찰된 결과를 프로덕션 환경에서 발생할 것으로 생각되는 범위까지 확장해야 합니다. 실제 사용자에게 영향을 주지 않도록 주의하고 실제 사용자 데이터 및 손상된 사용 통계 또는 프로덕션 보고서와 충돌하지 않도록 테스트 데이터에 태그를 지정하면 프로덕션에서도 로드 및 성능 테스트를 수행할 수 있습니다.

테스트를 통해 기본 리소스, 조정 설정, 서비스 할당량 및 복원력 설계가 로드 조건에서 예상대로 작동하는지 확인합니다.

카오스 엔지니어링을 사용한 복원력 테스트: 사전 프로덕션 및 프로덕션 환경에 정기적으로 장애를 주입하는 테스트를 실행합니다. 워크로드에서 장애에 대응할 방법에 대한 가설을 테스트 결과와 비교하고 일치하지 않는 경우 반복합니다. 프로덕션 테스트가 사용자에게 영향을 미치지 않는지 확인합니다.

클라우드에서는 워크로드의 장애 과정을 테스트하고 복구 절차를 검증할 수 있습니다. 자동화를 사용하여 다양한 장애를 시뮬레이션하거나 이전에 장애로 이어졌던 시나리오를 재현할 수 있습니다. 이렇게 하면 장애 경로를 노출한 후 실제 장애 시나리오가 발생하기 전에 테스트하고 수정하여, 위험을 줄일 수 있습니다.

> 카오스 엔지니어링은 프로덕션에서 격변의 조건을 견딜 수 있는 시스템의 기능에 대한 신뢰를 쌓기 위해 시스템을 실험하는 분야입니다. – 카오스 엔지니어링 원칙

사전 프로덕션 및 테스트 환경에서는 CI/CD 주기의 일부로 카오스 엔지니어링을 정기적으로 수행해야 합니다. 프로덕션에서는 가용성이 중단되지 않도록 주의를 기울여야 하며 실전 연습을 사용하여 카오스 엔지니어링이 프로덕션에 미치는 위험을 제어해야 합니다.

테스트 작업은 가용성 목표에 부합해야 합니다. 가용성 목표를 달성할 수 있는지 테스트하는 것이 이러한 목표를 달성할 수 있다는 확신을 가질 수 있는 유일한 방법입니다.

워크로드 복원력 설계의 기준으로 사용된 구성 요소 장애를 테스트합니다. 여기에는 EC2 인스턴스 손실, 기본 Amazon RDS 데이터베이스 인스턴스 장애 및 가용 영역 중단이 포함됩니다.

외부 종속성 사용 불가를 테스트합니다. 1초 미만~몇 시간 범위의 지속 시간 동안 일시적인 종속성 장애가 발생할 때의 워크로드 복원력을 테스트해야 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 불안정한 네트워크 연결(패킷이 삭제됨)로 인해 발생합니다. 네트워킹 영향(예: 지연 시간, 메시지 삭제), DNS 장애(예: 이름을 확인할 수 없거나 종속 서비스에 대한 연결을 설정할 수 없음)를 비롯한 장애를 주입하는 기능을 사용할 수도 있습니다.

몇 가지 타사 옵션을 사용하여 장애를 주입할 수 있습니다. 예를 들어 Netflix Simian Army, The Chaos ToolKit 및 Shopify Toxiproxy와 같은 오픈 소스 옵션과 Gremlin과 같은 상용 옵션을 사용할 수 있습니다. 카오스 엔지니어링을 구현하는 방법에 대한 초기 조사에서는 자체 작성된 스크립트를 사용하는 것이 좋습니다. 이렇게 하면 엔지니어링 팀에서 워크로드에 카오스를 도입하는 방법을 익힐 수 있습니다. 이에 대한 예제는 Bash, Python, Java 및 PowerShell과 같은 다수의 언어를 사용한 EC2 RDS 및 S3의 복원력 테스트를 참조하십시오. Systems Manager를 사용하여 Amazon EC2에 카오스를 주입하는 작업도 구현해야 합니다. 이 작업을 구현하면 AWS Systems Manager Documents를 사용하여 전압 저하 및 높은 CPU 조건을 시뮬레이션할 수 있습니다.

정기적인 실전 연습 시행: 실제 장애 시나리오에 참여할 직원들과 함께 실전 연습을 정기적으로 수행하여 프로덕션에 최대한 근접한 장애 절차(프로덕션 환경의 장애 절차 포함)를 연습합니다. 실전 연습에서는 프로덕션 테스트가 사용자에게 영향을 미치지 않도록 하는 조치가 시행됩니다.

프로덕션 환경에서 이벤트를 시뮬레이션하기 위한 실전 테스트를 정기적으로 실시하여 아키텍처 및 프로세스가 어떻게 작동하는지 테스트할 수 있습니다. 이 테스트는 어느 분야에서 개선이 필요한지 파악하고, 조직이 이벤트에 대처하는 경험을 쌓는 데 도움이 됩니다.

복원력을 위한 설계가 마련되고 비 프로덕션 환경에서 이 설계를 테스트한 후에는 실전 연습을 통해 모든 구성 요소가 프로덕션에서 예상대로 작동하는지 확인합니다. 실전 연습, 특히 첫 번째 실전 연습에서는 엔지니어와 운영 팀이 모두 모여 실전 연습의 일정과 내용에 대한 정보를 숙지합니다. 또한, 플레이북이 준비되어 있어야 합니다. 그런 다음 프로덕션 시스템에 규정된 방식으로 장애를 주입하고 영향을 평가합니다. 모든 시스템이 설계대로 작동할 경우 감지 및 자가 복구가 수행됩니다. 영향은 거의 없습니다. 그러나 부정적인 영향이 관찰되면 테스트가 롤백되고 워크로드 문제가 해결됩니다. 필요한 경우 플레이북을 사용하여 수동으로 문제를 해결합니다. 실전 연습은 프로덕션에서 수행되므로 고객의 가용성에 영향을 미치는 일이 없도록 모든 예방 조치를 취해야 합니다.

리소스

동영상

- AWS re:Invent 2019: Improving resiliency with chaos engineering(DOP309-R1)

설명서

- 지속적 제공 및 지속적 통합
- Cananry 사용(Amazon CloudWatch Synthetics)
- CodePipeline를 AWS CodeBuild와 함께 사용하여 코드 테스트 및 빌드 실행
- AWS Systems Manager를 사용하여 운영 플레이북 자동화
- AWS Marketplace: 지속적인 통합에 사용할 수 있는 제품
- APN 파트너: 지속적인 통합 파이프라인 구현을 지원할 수 있는 파트너

실습

- Well-Architected lab: Level 300: Testing for Resiliency of EC2 RDS and S3

외부 링크

- Principles of Chaos Engineering
- Resilience Engineering: Learning to Embrace Failure
- Apache JMeter

도서

- Casey Rosenthal, Lorin Hochstein, Aaron Blohowiak, Nora Jones, Ali Basiri. "Chaos Engineering"(2017년 8월)

DR(재해 복구) 계획

DR 전략의 시작은 백업 및 중복 워크로드 구성 요소를 갖추는 것입니다. RTO 및 RPO는 가용성 복원에 대한 목표입니다. 비즈니스 요구 사항에 따라 이러한 목표를 설정하십시오. 워크로드 리소스 및 데이터의 위치와 기능을 고려하여 이러한 목표를 충족하는 전략을 구현합니다.

가동 중지 및 데이터 손실 시의 복구 목표 정의: 워크로드에는 RTO(복구 시간 목표)와 RPO(복구 시점 목표)가 있습니다.

RTO(복구 시간 목표)는 워크로드의 구성 요소가 복구 단계에 있어 조직의 미션 또는 미션/비즈니스 프로세스에 부정적인 영향을 미치기 전에는 사용할 수 없는 시간의 전체 길이를 나타냅니다.

RPO(복구 시점 목표)는 조직의 미션 또는 미션/비즈니스 프로세스에 부정적인 영향을 미치기 전에 워크로드의 데이터를 사용할 수 없는 시간의 전체 길이를 나타냅니다.

중요한 데이터로 가용성을 복원할 수는 없으므로 RPO는 RTO보다 작아야 합니다.

복구 목표 달성을 위해 정의된 복구 전략 사용: DR(재해 복구) 전략은 워크로드 목표를 충족하도록 정의됩니다.

다중 리전 전략이 필요하지 않은 한 단일 AWS 리전 내에서 여러 가용 영역을 사용하여 AWS의 복구 목표를 달성하는 것이 좋습니다.

필요에 따라 워크로드에 대한 다중 리전 전략을 설계할 때는 다음 전략 중 하나를 선택해야 합니다. 이러한 전략은 복잡성이 증가하고 RTO 및 RPO가 감소하는 순서로 나열되어 있습니다. DR 리전은 워크로드에 사용된 AWS 리전이 아닌 리전을 말합니다(워크로드가 온프레미스인 경우 모든 AWS 리전).

- **백업 및 복원**(시간 단위 RPO, 24시간 이하의 RTO): 데이터와 애플리케이션을 DR 리전에 백업합니다. 재해 복구에 필요한 경우 이 데이터를 복원합니다.
- **파일럿 라이트**(분 단위 RPO, 시간 단위 RTO): 시스템의 가장 중요한 핵심 요소를 항상 실행하는 최소 버전의 환경을 DR 리전에 유지합니다. 복구 시기가 되면 중요한 핵심 요소를 중심으로 전체 프로덕션 환경을 신속하게 프로비저닝할 수 있습니다.

- **웜 대기**(초 단위 RPO, 분 단위 RTO): 항상 실행되는 모든 기능을 갖춘 환경의 축소된 버전을 DR 리전에 유지합니다. 비즈니스 크리티컬 시스템은 완전히 복제되고 항상 실행되지만 플릿은 축소됩니다. 복구 시기가 되면 시스템은 프로덕션 로드를 처리하기 위해 신속하게 확장됩니다.
- **다중 리전 액티브–액티브**(초 단위 RPO 또는 RTO 없음, 초 단위 RTO): 워크로드가 여러 AWS 리전에 배포되고 능동적으로 트래픽을 처리합니다. 이 전략에서는 사용 중인 리전 간에 사용자 및 데이터를 동기화해야 합니다. 복구 시기가 되면 Amazon Route 53 또는 AWS Global Accelerator와 같은 서비스를 사용하여 워크로드가 정상인 위치로 사용자 트래픽을 라우팅합니다.

권장 사항
파일럿 라이트와 웜 대기 간의 차이를 이해하기 어려울 수 있습니다. 두 전략 모두 DR 리전에서 환경이 실행됩니다. DR 전략에 추가 인프라 배포가 포함되는 경우에는 파일럿 라이트를 사용합니다. 기존 인프라 스케일업 및 스케일아웃만 포함되는 경우 웜 대기를 사용합니다. RTO 및 RPO 요구 사항에 따라 두 전략 중에서 선택하십시오.

재해 복구 구현을 테스트하여 구현 확인: DR로의 장애 조치를 정기적으로 테스트하여 RTO 및 RPO가 충족되는지 확인합니다.

거의 실행되지 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 효과가 있는 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 실행해야 합니다. 앞에서 설명한 예의 경우에는 필요 여부에 관계없이 대기 스토어로 정기 장애 조치를 수행해야 합니다.

DR 사이트 또는 리전의 구성 드리프트 관리: 인프라, 데이터 및 구성이 필요에 따라 DR 사이트 또는 리전에 있는지 확인합니다. 예를 들어 AMI와 서비스 할당량이 최신 상태인지 확인합니다.

AWS Config는 AWS 리소스 구성을 지속적으로 모니터링하고 기록합니다. 이 서비스를 사용하면 드리프트를 감지하고, AWS Systems Manager Automation을 트리거하여 드리프트를 수정한 후, 경보를 생성할 수 있습니다. AWS CloudFormation은 배포된 스택의 드리프트를 추가로 감지할 수 있습니다.

복구 자동화: AWS 또는 타사 도구를 사용하여 시스템 복구를 자동화하고 트래픽을 DR 사이트 또는 리전으로 라우팅합니다.

구성된 상태 확인에 따라 Elastic Load Balancing 및 AWS Auto Scaling과 같은 AWS 서비스를 사용하여 정상 상태인 가용 영역에 로드를 분산하고, Amazon Route 53 및 AWS Global Accelerator와 같은 서비스를 사용하여 정상 상태인 AWS 리전으로 로드를 라우팅할 수 있습니다.

기존의 물리 또는 가상 데이터 센터 또는 프라이빗 클라우드의 워크로드의 경우 AWS Marketplace를 통해 제공되는 CloudEndure Disaster Recovery를 사용하여 AWS로의 자동화된 재해 복구 전략을 설정할 수 있습니다. CloudEndure는 AWS에서 교차 리전/교차 AZ 재해 복구도 지원합니다.

리소스

동영상

- AWS re:Invent 2019: Backup-and-restore and disaster-recovery solutions with AWS(STG208)

설명서

- AWS Backup이란 무엇인가요?
- AWS Config Rules에 따른 규정 미준수 AWS 리소스 문제 해결
- AWS Systems Manager Automation
- AWS CloudFormation: 전체 CloudFormation 스택의 드리프트 감지

- Amazon RDS: 리전 간 백업 복사
- RDS: 리전 간 읽기 전용 복제본 복제
- S3: 리전 간 복제
- Route 53: DNS 장애 조치 구성
- CloudEndure Disaster Recovery
- AWS에서 인프라 구성 관리 솔루션을 구현하려면 어떻게 해야 합니까?
- CloudEndure Disaster Recovery to AWS
- AWS Marketplace: 재해 복구에 사용할 수 있는 제품
- APN 파트너: 재해 복구를 지원할 수 있는 파트너

가용성 목표 달성을 위한 구현 예제

이 섹션에서는 역방향 프록시, Amazon S3의 정적 콘텐츠, 애플리케이션 서버 및 영구 데이터 저장용 SQL 데이터베이스로 구성된 일반적인 웹 애플리케이션 배포를 사용하여 워크로드 설계를 검토합니다. 각 가용성 목표에 대해 예제 구현이 제공됩니다. 이 워크로드에서는 컴퓨팅에 컨테이너나 AWS Lambda를 대신 사용하고 데이터베이스에 NoSQL(예: Amazon DynamoDB)을 사용할 수도 있지만 접근 방식은 비슷합니다. 각 시나리오에서는 이러한 주제에 대한 워크로드 설계를 통해 가용성 목표를 달성하는 방법을 보여줍니다.

주제	자세한 내용을 참조할 수 있는 섹션
리소스 모니터링	워크로드 리소스 모니터링
수요 변화에 맞게 구현 조정	수요 변경에 따라 조정되는 워크로드 설계
변경 구현	변경 구현
데이터 백업	데이터 백업
복원력을 위한 아키텍처 설계	장애 격리를 사용하여 워크로드 보호 구성 요소 장애를 견디도록 워크로드 설계
복원력 테스트	안정성 테스트
DR(재해 복구) 계획	DR(재해 복구) 계획

종속성 선택

이 구현의 애플리케이션에는 Amazon EC2가 사용됩니다. Amazon RDS 및 여러 가용 영역을 사용하여 애플리케이션의 가용성을 개선하는 방법을 살펴보겠습니다. DNS에는 Amazon Route 53이 사용됩니다. 여러 가용 영역 사용 시에는 Elastic Load Balancing을 사용합니다. 백업 및 정적 콘텐츠에는 Amazon S3가 사용됩니다. 이 구현은 고가용성을 고려하여 설계되므로 그 자체의 가용성이 높은 서비스만 사용해야 합니다. 각 AWS 서비스의 설계 목표는 부록 A: 일부 AWS 서비스 설계 가용성을 참조하십시오.

단일 리전 시나리오

가용성 99% 시나리오

이러한 워크로드는 비즈니스에 도움이 되지만 사용할 수 없다면 불편할 뿐입니다. 이러한 유형의 워크로드에는 내부 도구, 내부 지식 관리 또는 프로젝트 추적이 포함될 수 있습니다. 실험적

서비스에서 제공되며 필요한 경우 서비스를 숨길 수 있는 기능 토글이 포함된 고객용 워크로드도 여기에 포함될 수 있습니다.

하나의 리전 및 하나의 가용 영역에서 이러한 워크로드를 배포할 수 있습니다.

리소스 모니터링
그 후에는 서비스 홈 페이지가 HTTP 200 OK 상태를 반환하는지 여부를 확인하는 간단한 모니터링을 수행합니다. 문제가 발생하면 인스턴스의 로깅을 사용하여 근본 원인을 설정한다는 내용이 플레이북에 표시됩니다.

수요 변화에 맞게 구현 조정
플레이북에서 일반적인 하드웨어 장애, 긴급 소프트웨어 업데이트 및 기타 서비스가 중단되는 변경 관련 설명을 확인할 수 있습니다.

변경 구현
또한 AWS CloudFormation을 사용하여 인프라를 코드로 정의하고 장애 발생 시 재구성 속도를 개선합니다.

그리고 런북을 사용해 서비스를 설치하고 다시 시작하는 데 필요한 가동 중단 시간을 적용하여 소프트웨어 업데이트를 수동으로 수행합니다. 배포 중에 문제가 발생하면 런북의 설명에 따라 이전 버전으로 롤백할 수 있습니다.

오류 수정은 운영 팀과 개발 팀이 로그를 분석하여 수행하며 우선 순위가 지정되고 완료된 후 배포됩니다.

데이터 백업
그리고 공급업체 또는 특별히 구축된 백업 솔루션을 사용하여 런북(Runbook)을 통해 Amazon S3로 암호화된 백업 데이터를 전송합니다. 그런 다음 데이터를 복원하고 런북을 사용하여 정기적으로 데이터를 사용할 수 있는지 확인하는 방식으로 백업이 작동함을 테스트합니다. 그 후에는 Amazon S3 객체에 대해 버전 관리를 구성하고, 백업 삭제 권한을 제거합니다. 그리고 Amazon S3 버킷 수명 주기 정책을 사용해 요구 사항에 따라 백업을 아카이브하거나 영구적으로 삭제합니다.

복원력을 위한 아키텍처 설계

워크로드는 단일 리전 및 단일 가용 영역을 사용하여 배포됩니다. 여기서는 데이터베이스를 포함한 애플리케이션을 단일 인스턴스에 배포합니다.

복원력 테스트

그 후에는 새 소프트웨어의 배포 파이프라인을 예약하고 몇 가지 단위 테스트(대개 조립된 워크로드의 화이트 박스/블랙 박스 테스트)를 수행합니다.

DR(재해 복구) 계획

장애 중에는 장애 상황이 종료될 때까지 대기하며, 필요에 따라 런북을 통해 DNS 수정 기능을 사용하여 정적 웹 사이트로 요청을 라우팅합니다. 이 과정을 진행하기 위한 복구 시간은 인프라를 배포하고 데이터베이스를 최신 백업으로 복원할 수 있는 속도에 따라 결정됩니다. 이 배포는 동일한 가용 영역에서 수행할 수도 있고, 가용 영역에서 장애가 발생하는 경우에는 런북을 사용하여 다른 가용 영역에서 수행할 수도 있습니다.

가용성 설계 목표

30분 동안 복구를 파악하고 복구 실행을 결정한 다음 10분 동안 AWS CloudFormation에서 전체 스택을 배포합니다. 이때 스택은 새 가용 영역에 배포하며, 30분 내에 데이터베이스를 복원할 수 있다고 가정합니다. 그러므로 장애 발생부터 복구까지 걸리는 시간은 약 70분입니다.
분기당 장애가 1회 발생한다고 가정하면 연간 예상 영향 시간은 280분(4시간 40분)입니다.

즉, 가용성의 상한은 99.9%입니다. 실제 가용성은 장애의 실제 속도, 장애 지속 시간 및 각 장애가 실제로 복구되는 속도에 따라서도 달라집니다. 이 아키텍처의 경우에는 업데이트를 위해 애플리케이션을 오프라인으로 설정해야 하며(연간 24시간 예상: 변경당 4시간 소요, 연간 6회 변경) 실제 이벤트 시간도 고려해야 합니다. 따라서 이 백서 앞부분에 나왔던 애플리케이션 가용성 표를 참조할 때 **가용성 설계 목표**는 99%입니다.

요약

주제	구현
리소스 모니터링	사이트 상태 확인만, 알림 없음
수요 변화에 맞게 구현 조정	재배포를 통한 수직적 확장
변경 구현	배포 및 롤백 런북
데이터 백업	백업 및 복원 런북
복원력을 위한 아키텍처 설계	재구축을 완료하고 백업에서 복원
복원력 테스트	재구축을 완료하고 백업에서 복원
DR(재해 복구) 계획	암호화된 백업, 필요한 경우 다른 가용 영역으로 복원

가용성 99.9% 시나리오

다음 가용성 목표는 고가용성을 유지해야 하지만 단기간 사용할 수 없는 애플리케이션은 허용하는 것입니다. 이러한 유형의 워크로드는 대개 가동이 중단되면 직원에게 영향이 있는 내부 작업에 사용됩니다. 그리고 기업에 큰 수익을 제공하지는 않으며 복구 시간이나 복구 지점 목표가 다소 길어도 되는 고객용 워크로드도 여기에 포함될 수 있습니다. 이러한 워크로드에는 계정 또는 정보 관리용 관리 애플리케이션이 포함됩니다.

여기서는 배포에 두 가용 영역을 사용하고 애플리케이션을 개별 계층으로 분리하여 워크로드의 가용성을 개선할 수 있습니다.

리소스 모니터링

모니터링 범위가 확장되어 홈 페이지에서 HTTP 200 OK 상태를 확인하는 방식으로 전반적인 웹 사이트 가용성 관련 알림이 생성됩니다. 또한 웹 서버를 교체하고 데이터베이스를 장애 조치할 때마다 알림이 표시됩니다. 그리고 Amazon S3의 정적 콘텐츠 가용성도 모니터링하며, 해당

콘텐츠를 사용할 수 없으면 알림이 전송됩니다. 관리 작업을 손쉽게 수행하고 근본 원인을 원활하게 분석할 수 있도록 로깅이 집계됩니다.

수요 변화에 맞게 구현 조정

자동 조정은 EC2 인스턴스의 CPU 사용률을 모니터링하고 인스턴스를 추가 또는 제거하여 CPU 목표를 70%로 유지하되 가용 영역당 EC2 인스턴스를 1개 이상으로 유지하도록 구성됩니다. RDS 인스턴스의 로드 패턴에서 확장이 필요한 징후가 나타나면 유지 관리 기간 중에 인스턴스 유형을 변경합니다.

변경 구현

인프라 배포 기술은 이전 시나리오와 동일하게 유지됩니다.

새 소프트웨어는 고정된 일정(2~4주마다)에 따라 제공됩니다. 소프트웨어 업데이트는 자동화되므로 Canary 또는 블루/그린 배포 패턴을 사용하는 대신 현재 위치 교체를 사용합니다. 롤백 여부는 런북을 사용하여 결정합니다.

문제의 근본 원인을 파악하기 위한 플레이북을 마련합니다. 근본 원인이 확인되면 운영 팀과 개발 팀이 함께 오류 수정 방법을 파악합니다. 수정 기능이 개발되면 수정 방법이 배포됩니다.

데이터 백업

Amazon RDS를 사용하여 백업 및 복원을 수행할 수 있습니다. 런북을 사용해 백업과 복원을 정기적으로 실행하여 복구 요구 사항을 충족할 수 있는지를 확인합니다.

복원력을 위한 아키텍처 설계

여기서는 배포에 두 가용 영역을 사용하고 애플리케이션을 개별 계층으로 분리하여 애플리케이션의 가용성을 개선할 수 있습니다. 이를 위해 Elastic Load Balancing, Auto Scaling 및 Amazon RDS Multi-AZ(AWS Key Management Service를 통해 암호화된 스토리지 포함)와 같이 여러 가용 영역에서 작동하는 서비스를 사용합니다. 이렇게 하면 리소스 수준과 가용 영역 수준에서 장애가 발생해도 서비스를 계속 사용할 수 있습니다.

로드 밸런서는 정상 상태의 애플리케이션 인스턴스로만 트래픽을 라우팅합니다. 데이터 영역/애플리케이션 계층에서 상태 확인을 수행하여 인스턴스의 애플리케이션 기능을 확인해야 합니다. 제어 영역에 대해서는 이 확인을 수행하면 안 됩니다. 웹 애플리케이션의 상태 확인 URL이 제공되며, 로드 밸런서와 Auto Scaling에 사용되도록 구성됩니다(장애가 발생하는

인스턴스가 제거/교체됨). Amazon RDS는 기본 가용 영역에서 인스턴스에 장애가 발생하는 경우 보조 가용 영역에서 사용 가능하도록 활성 데이터베이스 엔진을 관리하며, 복구를 통해 동일한 복원력을 제공하도록 인스턴스를 복원합니다.

계층을 분리한 후에는 분산 시스템 복원력 패턴을 사용하여 애플리케이션의 안정성을 높일 수 있습니다. 그러면 가용 영역 장애 조치 중에 데이터베이스를 일시적으로 사용할 수 없을 때도 애플리케이션은 계속 사용할 수 있습니다.

복원력 테스트

이전 시나리오와 마찬가지로 기능 테스트를 수행합니다. ELB, 자동 조정 또는 RDS 장애 조치의 자가 복구 기능은 테스트하지 않습니다.

일반적인 데이터베이스 문제, 보안 관련 인시던트 및 장애가 발생한 배포에는 플레이북을 사용합니다.

DR(재해 복구) 계획

전체 워크로드 복구 및 일반 보고용 런북이 제공됩니다. 복구에는 워크로드와 동일한 리전에 저장된 백업이 사용됩니다.

가용성 설계 목표

최소한 일부 장애에서는 복구 실행을 수동으로 결정해야 합니다. 하지만 이 시나리오에서는 자동화 범위를 확대하여 매년 2개 이벤트에 대해서만 복구 실행을 수동으로 결정하면 된다고 가정합니다. 30분 동안 복구 실행을 결정하며, 복구는 30분 내에 완료된다고 가정합니다. 그러므로 장애에서 복구하는 데 걸리는 시간은 60분입니다. 인시던트가 매년 2회 발생한다고 가정할 때 연간 예상 영향 시간은 120분입니다.

즉, 가용성의 상한은 99.95%입니다. 실제 가용성은 장애의 실제 속도, 장애 지속 시간 및 각 장애가 실제로 복구되는 속도에 따라서도 달라집니다. 이 아키텍처에서는 업데이트를 위해 애플리케이션을 잠시 오프라인으로 설정해야 합니다. 하지만 이러한 업데이트는 자동으로 수행됩니다. 업데이트에 걸릴 것으로 예상되는 시간은 매년 150분입니다(변경당 15분 소요, 연간 10회 변경). 그러므로 매년 서비스 사용 불가 시간이 270분 추가되므로 **가용성 설계 목표**는 99.9%입니다.

요약

주제	구현
리소스 모니터링	사이트 상태 확인만, 중단 시 알림이 전송됨
수요 변화에 맞게 구현 조정	웹 및 자동 조정 애플리케이션 계층에 ELB 사용, 다중 AZ RDS 크기 조정
변경 구현	인플레이스 자동 배포 및 롤백 런북
데이터 백업	RDS를 통해 RPO를 충족하는 자동 백업 및 복원 런북
복원력을 위한 아키텍처 설계	자동 조정을 통해 자가 복구 웹 및 애플리케이션 계층 제공. RDS는 다중 AZ에 위치함
복원력 테스트	ELB 및 애플리케이션이 자가 복구됨. RDS는 다중 AZ에 위치하며 명시적 테스트는 없음
DR(재해 복구) 계획	RDS를 통해 동일한 AWS 리전으로의 암호화된 백업

가용성 99.99% 시나리오

애플리케이션의 이 가용성 목표를 달성하려면 애플리케이션의 가용성이 높아야 하며 구성 요소에 장애가 발생해도 애플리케이션을 계속 사용할 수 있어야 합니다. 장애 발생 시 추가 리소스를 조달하지 않아도 애플리케이션이 계속 실행되어야 합니다. 이 가용성 목표는 전자상거래 사이트, B2B 웹 서비스 또는 트래픽이 많이 생성되는 콘텐츠/미디어 사이트 등 기업의 주요/중요 수익 창출원인 미션 크리티컬 애플리케이션에 적용됩니다.

리전 내에서 정적으로 안정 상태를 유지하는 아키텍처를 사용하면 가용성을 더욱 개선할 수 있습니다. 이 가용성 목표를 달성하기 위해 결함 시에도 계속 사용 가능하도록 워크로드 동작에서 제어 영역을 변경할 필요는 없습니다. 예를 들어 가용 영역 하나를 사용할 수 없어도 애플리케이션을 계속 사용할 수 있을 만큼 충분한 용량이 있어야 합니다. 그리고 Amazon Route 53 DNS 업데이트도 수행할 필요가 없어야 합니다. 또한 새 인프라를 생성할 필요도 없어야 합니다. 예를 들어 S3 버킷을 생성 또는 수정하거나, 새 IAM 정책 생성/기존 정책 수정을 수행하거나, Amazon ECS 작업 구성을 수정할 필요가 없어야 합니다.

리소스 모니터링

모니터링에는 성공 지표와 문제 발생 시의 알림이 모두 포함됩니다. 또한 장애 발생 웹 서버를 교체하고, 데이터베이스를 장애 조치하고, AZ에서 장애가 발생할 때마다 알림이 표시됩니다.

수요 변화에 맞게 구현 조정

Amazon Aurora를 RDS로 사용하여 읽기 전용 복제본의 자동 조정을 지원합니다. 이러한 애플리케이션에서는 기본 콘텐츠의 쓰기 가용성보다 읽기 가용성을 우선적으로 고려한 엔지니어링도 주요 아키텍처 결정 사항에 포함됩니다. 또한 Aurora를 사용하면 필요에 따라 스토리지를 10GB 단위로 최대 64TB까지 자동으로 확장할 수 있습니다.

변경 구현

여기서는 각 격리 영역별로 Canary 또는 블루/그린 배포를 사용하여 업데이트를 배포합니다. 배포는 완전히 자동화되며 KPI에서 문제가 확인되는 롤백하는 작업도 자동으로 수행됩니다.

런북은 엄격한 보고 요구 사항 및 성능 추적을 위해 사용됩니다. 정상적으로 진행된 작업에서 장애 발생 추세가 나타나는 경우에는 성능 또는 가용성 목표 충족을 위해 플레이북을 사용하여 해당 추세의 원인을 파악합니다. 플레이북은 검색되지 않은 장애 모드 및 보안 인시던트 확인용으로, 혹은 장애의 근본 원인 파악을 위해서 사용됩니다. 또한 Infrastructure Event Management 오퍼링과 관련하여 AWS Support에서 지원을 제공합니다.

웹 사이트를 구축하고 운영하는 팀은 예기치 않은 장애를 발생시키는 오류 수정 사항을 확인하고, 해당 오류의 수정이 구현된 후 우선적으로 배포되도록 우선 순위를 지정합니다.

데이터 백업

Amazon RDS를 사용하여 백업 및 복원을 수행할 수 있습니다. 런북을 사용해 백업과 복원을 정기적으로 실행하여 복구 요구 사항을 충족할 수 있는지를 확인합니다.

복원력을 위한 아키텍처 설계

이 방식을 적용할 때는 가용 영역 3개를 사용하는 것이 좋습니다. 가용 영역이 3개인 배포를 사용하는 경우 각 AZ의 정적 용량은 최대 용량의 50%입니다. 가용 영역 2개를 사용할 수도 있지만 이 경우 정적으로 안정 상태인 용량 관련 비용이 증가합니다. 두 영역의 정적 용량이 모두 최대 용량의 100%여야 하기 때문입니다. 지리적 캐싱 기능을 제공하고 애플리케이션 데이터 영역의 요청 수를 줄이기 위해 Amazon CloudFront를 추가합니다.

Amazon Aurora를 RDS로 사용하고 3개 영역 모두에 읽기 전용 복제본을 배포합니다.

애플리케이션은 모든 계층의 소프트웨어/애플리케이션 복원력 패턴을 사용하여 구축됩니다.

복원력 테스트

배포 파이프라인에는 성능, 로드 및 장애 주입 테스트를 비롯한 전체 테스트 집합이 포함됩니다.

이 시나리오에서는 실전 연습을 통해 장애 복구 절차를 지속적으로 연습합니다. 이때 런북을 사용하여 해당 절차를 준수하면서 작업을 수행할 수 있는지 확인합니다. 웹 사이트 개발 팀이 운영 또한 담당합니다.

DR(재해 복구) 계획

전체 워크로드 복구 및 일반 보고용 런북이 제공됩니다. 복구에는 워크로드와 동일한 리전에 저장된 백업이 사용됩니다. 복원 절차는 실전 연습의 일부로 정기적으로 진행됩니다.

가용성 설계 목표

최소한 일부 장애에서는 복구 실행을 수동으로 결정해야 합니다. 하지만 이 시나리오에서는 자동화 범위를 확대하여 매년 2개 이벤트에 대해서만 복구 실행을 수동으로 결정하면 되며, 복구 작업은 빠르게 진행된다고 가정합니다. 10분 동안 복구 실행을 결정하며, 복구는 5분 내에 완료된다고 가정합니다. 그러므로 장애 발생부터 복구까지 걸리는 시간은 15분입니다. 장애가 매년 2회 발생한다고 가정할 때 연간 예상 영향 시간은 30분입니다.

즉, 가용성의 상한은 99.99%입니다. 실제 가용성은 장애의 실제 속도, 장애 지속 시간 및 각 장애가 실제로 복구되는 속도에 따라서도 달라집니다. 이 아키텍처에서는 업데이트 과정 전반에 걸쳐 애플리케이션이 계속 온라인 상태라고 가정합니다. 이 가정을 토대로 할 때 **가용성 설계 목표**는 99.99%입니다.

요약

주제	구현
리소스 모니터링	모든 계층과 KPI에서 상태 확인. 구성된 경보가 작동하면 알림이 전송됨. 모든 장애 발생 시 알림이 전송됨. 운영 회의에서 철저하게 추세를 탐지하고 목표 설계를 관리함
수요 변화에 맞게 구현 조정	ELB를 웹 및 자동 조정 애플리케이션 계층에 사용. Aurora RDS의 여러 영역에 자동 조정 스토리지 및 읽기 전용 복제본 배포
변경 구현	Canary 또는 블루/그린을 통한 자동 배포. KPI 또는 알림이 애플리케이션에서 감지되지 않은 문제를 나타내는 경우 자동으로 롤백. 배포는 격리 영역별로 수행됨
데이터 백업	RDS를 통한 자동 백업으로 RPO를 충족하고 실전 연습에서 자동 복원을 주기적으로 시행
복원력을 위한 아키텍처 설계	애플리케이션을 위한 장애 격리 영역을 구현하고 자동 조정을 통해 자가 복구 웹 및 애플리케이션 계층 제공. RDS는 다중 AZ에 위치함
복원력 테스트	구성 요소 및 격리 영역 장애 테스트가 파이프라인에 포함되고 운영 직원이 실전 연습을 통해 주기적으로 시행. 알려지지

주제	구현
	않은 문제 진단을 위한 플레이북과 근본 원인 분석 프로세스가 마련되어 있음
DR(재해 복구) 계획	실전 연습이 시행된 동일한 AWS 리전으로의 RDS를 통한 암호화된 백업

다중 리전 시나리오

여러 AWS 리전에서 애플리케이션을 구현하면 운영 비용이 증가합니다. 그 이유 중 하나는 독립성을 유지하기 위해 리전을 격리하기 때문입니다. 이러한 방식은 매우 신중하게 결정해야 합니다. 이는 리전 간의 격리 경계가 명확하며 리전 간에 상관된 장애를 방지하기는 매우 어렵기 때문입니다. 여러 리전을 사용하는 경우에는 리전별 AWS 서비스에서 하드 종속성 장애가 발생하는 경우 복구 시간을 더욱 효과적으로 제어할 수 있습니다. 이 섹션에서는 다양한 구현 패턴 및 이러한 패턴의 일반적인 가용성에 대해 설명합니다.

가용성 99.95%, 복구 시간 5~30분

애플리케이션의 이 가용성 목표를 달성하려면 가동 중단 시간이 매우 짧아야 하며 특정 시간에 손실되는 데이터의 양도 거의 없어야 합니다. 이 가용성 목표가 설정되는 애플리케이션에는 뱅킹, 투자, 긴급 서비스, 데이터 캡처 분야의 애플리케이션이 포함됩니다. 이러한 애플리케이션의 복구 시간과 복구 지점은 매우 짧습니다.

두 AWS 리전에서 웜 대기 방식을 사용하면 복구 시간을 더 단축할 수 있습니다. 이 시나리오에서는 전체 워크로드를 두 리전에 배포합니다. 패시브 사이트는 축소되고 모든 데이터는 최종 일관성을 유지합니다. 두 배포 모두 해당 리전 내에서 정적 안정성을 유지합니다. 애플리케이션은 분산 시스템 복원력 패턴을 사용하여 구축되어야 합니다. 워크로드 상태를 모니터링하는 경량 라우팅 구성 요소를 생성해야 하며, 필요한 경우 트래픽을 패시브 리전으로 라우팅하도록 구성할 수 있습니다.

리소스 모니터링
웹 서버를 교체하고 데이터베이스와 리전을 장애 조치할 때마다 알림이 표시됩니다. 그리고 Amazon S3의 정적 콘텐츠 가용성도 모니터링하며, 해당 콘텐츠를 사용할 수 없으면 알림이 전송됩니다. 각 리전에서 관리 작업을 손쉽게 수행하고 근본 원인을 원활하게 분석할 수 있도록 로깅이 집계됩니다.

라우팅 구성 요소는 애플리케이션 상태와 리전별 하드 종속성을 모두 모니터링합니다.

수요 변화에 맞게 구현 조정
가용성 99.99% 시나리오와 동일합니다.

변경 구현
새 소프트웨어는 고정된 일정(2~4주마다)에 따라 제공됩니다. 소프트웨어 업데이트는 Canary 또는 블루/그린 배포 패턴을 사용하여 자동화됩니다.

리전 장애 조치가 수행되는 경우, 해당 이벤트 발생 과정에서 나타나는 일반적인 고객 문제가 확인되는 경우, 그리고 일반적인 보고가 진행되는 경우 런북이 제공됩니다.

일반적인 데이터베이스 문제, 보안 관련 인시던트, 장애가 발생한 배포, 리전 장애 조치 과정의 예기치 않은 고객 문제 및 문제의 근본 원인 결정에는 플레이북이 사용됩니다. 근본 원인이 확인되면 운영 팀과 개발 팀이 함께 오류 수정 방법을 파악하며, 해당 오류의 수정이 개발된 후에 배포됩니다.

또한 Infrastructure Event Management와 관련하여 AWS Support에서 지원을 제공합니다.

데이터 백업
가용성 99.99% 시나리오와 마찬가지로 RDS 백업을 자동화하고 S3 버전 관리를 사용합니다. 데이터는 액티브 리전의 Aurora RDS 클러스터에서 패시브 리전의 리전 간 읽기 전용 복제본으로 비동기식으로 자동 복제됩니다. S3 리전 간 복제는 데이터를 액티브 리전에서 패시브 리전으로 비동기식으로 자동 이동하는 데 사용됩니다.

복원력을 위한 아키텍처 설계
가용성 99.99% 시나리오와 동일하며 리전 장애 조치가 가능합니다. 리전 장애 조치는 수동으로 관리됩니다. 장애 조치 중에는 두 번째 리전에서 복구가 완료될 때까지 DNS 장애 조치를 사용하여 정적 웹 사이트로 요청을 라우팅합니다.

복원력 테스트

가용성 99.99% 시나리오와 동일하며 실전 연습에서 런북을 사용하여 아키텍처를 검증합니다. 또한 RCA 수정은 즉각적인 구현 및 배포를 위해 기능 릴리스보다 우선적으로 적용됩니다.

DR(재해 복구) 계획

리전 장애 조치는 수동으로 관리됩니다. 모든 데이터는 비동기식으로 복제됩니다. 웜 대기 상태의 인프라는 스케일아웃됩니다. AWS Step Functions에서 실행되는 워크플로를 사용하여 이 작업을 자동화할 수 있습니다. AWS Systems Manager(SSM)도 이 자동화에 도움이 될 수 있습니다. AWS Systems Manager(SSM)를 사용하면 Auto Scaling 그룹을 업데이트하고 인스턴스 크기를 조정하는 SSM 문서를 생성할 수 있습니다.

가용성 설계 목표

최소한 일부 장애에서는 복구 실행을 수동으로 결정해야 합니다. 하지만 이 시나리오에서는 자동화가 효율적으로 진행되어 매년 2개 이벤트에 대해서만 복구 실행을 수동으로 결정하면 된다고 가정합니다. 20분 동안 복구 실행을 결정하며, 복구는 10분 내에 완료된다고 가정합니다. 그러므로 장애 발생부터 복구까지 걸리는 시간은 약 30분입니다. 장애가 매년 2회 발생한다고 가정할 때 연간 예상 영향 시간은 60분입니다.

즉, 가용성의 상한은 99.95%입니다. 실제 가용성은 장애의 실제 속도, 장애 지속 시간 및 각 장애가 실제로 복구되는 속도에 따라서도 달라집니다. 이 아키텍처에서는 업데이트 과정 전반에 걸쳐 애플리케이션이 계속 온라인 상태라고 가정합니다. 이 가정을 토대로 할 때 **가용성 설계 목표**는 99.95%입니다.

요약

주제	구현
리소스 모니터링	AWS 리전의 DNS 상태를 포함하여 모든 계층과 KPI에서 상태 확인. 구성된 경보가 작동하면 알림이 전송됨. 모든 장애 발생 시 알림이 전송됨. 운영 회의에서 철저하게 추세를 탐지하고 목표 설계를 관리함

주제	구현
수요 변화에 맞게 구현 조정	ELB를 웹 및 자동 조정 애플리케이션 계층에 사용. Aurora RDS의 액티브 및 패시브 리전에 있는 여러 영역에 자동 조정 스토리지 및 읽기 전용 복제본 배포. 정적 안정성을 위해 AWS 리전 간에 데이터 및 인프라가 동기화됨
변경 구현	Canary 또는 블루/그린을 통한 자동 배포. KPI 또는 알림이 애플리케이션에서 감지되지 않은 문제를 나타내는 경우 자동으로 롤백. 배포는 한 번에 하나의 AWS 리전에 있는 단일 격리 영역에 수행됨
데이터 백업	각 AWS 리전에서 RDS를 통한 자동 백업으로 RPO를 충족하고 실전 연습에서 자동 복원을 주기적으로 시행. Aurora RDS 및 S3 데이터는 액티브 리전에서 패시브 리전으로 비동기식으로 자동 복제됨
복원력을 위한 아키텍처 설계	자동 조정을 통해 자가 복구 웹 및 애플리케이션 계층 제공. RDS는 다중 AZ에 위치함. 리전 장애 조치는 장애 조치 중에 제공되는 정적 사이트를 통해 수동으로 관리됨
복원력 테스트	구성 요소 및 격리 영역 장애 테스트가 파이프라인에 포함되고 운영 직원이 실전 연습을 통해 주기적으로 시행. 알려지지 않은 문제 진단을 위한 플레이북과 근본

주제	구현
	원인 분석 프로세스가 마련되어 있으며 문제의 내용과 수정 또는 차단 방법을 알리는 커뮤니케이션 경로가 있음. RCA 수정은 즉각적인 구현 및 배포를 위해 기능 릴리스보다 우선적으로 적용됨
DR(재해 복구) 계획	웜 대기가 다른 리전에 배포됨. 인프라는 AWS Step Functions 또는 AWS Systems Manager Documents를 사용하여 실행되는 워크플로를 사용하여 확장됨. RDS를 통한 암호화된 백업. 두 AWS 리전 간의 리전 간 읽기 전용 복제본. 정적 자산의 리전 간 복제본이 S3에 위치함. 복원 대상은 현재 활성 AWS 리전이며 실전 연습에서 AWS와 조율하여 시행됩니다.

복구 시간이 1분 미만인 가용성 99.999% 이상의 시나리오

애플리케이션의 이 가용성 목표를 달성하려면 가동 중단 시간이나 특정 시간에 손실되는 데이터가 거의 없어야 합니다. 이 가용성 목표를 설정할 수 있는 애플리케이션에는 특정 뱅킹, 투자, 금융, 정부 애플리케이션 및 매출 수준이 매우 높은 초대형 기업의 핵심 업무에 사용되는 중요한 비즈니스 애플리케이션이 포함됩니다. 이러한 목표를 달성하려면 모든 계층에서 항상 일정한 데이터 스토어와 완벽한 중복성을 갖춰야 합니다. 이 시나리오에서는 SQL 기반 데이터 스토어를 선택했습니다. 하지만 매우 짧은 RPO를 달성하기가 어려운 시나리오도 있습니다. 데이터를 분할할 수 있다면 데이터 손실을 방지할 수 있습니다. 단, 이 경우에는 여러 지리적 위치 간에 데이터 일관성이 유지되도록 애플리케이션 논리와 지연 시간을 추가해야 할 수 있으며,

파티션 간에 데이터를 이동하거나 복사하는 기능도 추가해야 할 수 있습니다. NoSQL 데이터베이스를 사용하는 경우 이 파티셔닝을 더 쉽게 수행할 수 있습니다.

여러 AWS 리전에서 액티브/액티브 또는 다중 마스터 방식을 사용하면 가용성을 더 높일 수 있습니다. 워크로드는 리전 간 정적 안정성(한 리전이 손실될 경우 나머지 리전에서 로드를 처리할 수 있음)을 유지하는 원하는 모든 리전에 워크로드가 배포됩니다. 라우팅 계층은 현재 정상 상태인 지리적 위치로 트래픽을 전송하고, 특정 위치가 비정상 상태가 되면 대상을 자동으로 변경하는 동시에 데이터 복제 계층을 일시적으로 중지합니다. Amazon Route 53에서는 10초 간격으로 상태 확인이 진행되며, 레코드 세트의 TTL도 매우 짧습니다(최저 1초부터).

리소스 모니터링
가용성 99.95% 시나리오와 동일하며 리전이 비정상인 것으로 감지되고 트래픽이 비정상 리전에서 먼 위치로 라우팅될 때 알림이 생성됩니다.

수요 변화에 맞게 구현 조정
가용성 99.95% 시나리오와 동일합니다.

변경 구현
배포 파이프라인에는 성능, 로드 및 장애 주입 테스트를 비롯한 전체 테스트 집합이 포함됩니다. 이 시나리오에서는 Canary 또는 블루/그린 배포를 사용하여 한 번에 한 격리 영역에 업데이트를 배포합니다(한 리전에 업데이트를 배포한 후에 다음 리전으로의 배포를 시작함). 배포 중에는 롤백을 더 빠르게 수행할 수 있도록 이전 버전이 인스턴스에서 계속 실행됩니다. 이 과정은 완전히 자동화되며 KPI에서 문제가 확인되는 롤백 작업도 자동으로 수행됩니다. 모니터링에는 성공 지표와 문제 발생 시의 알림이 모두 포함됩니다.

런북은 엄격한 보고 요구 사항 및 성능 추적을 위해 사용됩니다. 정상적으로 진행된 작업에서 장애 발생 추세가 나타나는 경우에는 성능 또는 가용성 목표 충족을 위해 플레이북을 사용하여 해당 추세의 원인을 파악합니다. 플레이북은 검색되지 않은 장애 모드 및 보안 인시던트 확인용으로, 혹은 장애의 근본 원인 파악을 위해서 사용됩니다.

웹 사이트 개발 팀이 운영 또한 담당합니다. 해당 팀은 예기치 않은 장애를 발생시키는 오류 수정 사항을 확인하고, 해당 오류의 수정이 구현된 후 우선적으로 배포되도록 우선 순위를 지정합니다. 또한 Infrastructure Event Management와 관련하여 AWS Support에서 지원을 제공합니다.

데이터 백업
가용성 99.95% 시나리오와 동일합니다.

복원력을 위한 아키텍처 설계
애플리케이션은 소프트웨어/애플리케이션 복원력 패턴을 사용하여 구축되어야 합니다. 필요한 가용성을 구현하려면 기타 여러 라우팅 계층을 구현해야 할 수도 있습니다. 이 추가 구현은 복잡하므로 가용성 설계 시에 이 부분을 충분히 감안해야 합니다. 애플리케이션은 배포 결함 격리 영역에서 구현되며, 리전 전체에서 이벤트가 발생하더라도 모든 고객에게 영향을 주지는 않도록 분할/배포됩니다.

복원력 테스트
이 시나리오에서는 실전 연습을 통해 아키텍처를 검증합니다. 이때 런북을 사용하여 해당 절차를 준수하면서 작업을 수행할 수 있는지 확인합니다.

DR(재해 복구) 계획
전체 워크로드 인프라 및 데이터가 여러 리전에 배포되는 액티브-액티브 다중 리전 배포. 로컬 읽기, 전역 쓰기 전략을 사용하면 한 리전이 모든 쓰기에 대한 마스터 데이터베이스가 되고 데이터는 읽기 작업을 위해 다른 리전으로 복제됩니다. 마스터 DB 리전에 장애가 발생하면 새 DB를 승격해야 합니다. 로컬 읽기, 전역 쓰기에서는 DB 쓰기가 처리되는 홈 리전에 사용자를 할당합니다. 따라서 사용자는 어느 리전에서든 읽거나 쓸 수 있지만, 서로 다른 리전에 있는 쓰기 간의 잠재적인 데이터 충돌을 관리하기 위한 복잡한 로직이 필요합니다.

리전이 비정상인 것으로 감지되면 라우팅 계층이 나머지 정상 리전으로 트래픽을 자동으로 라우팅합니다. 수동 개입은 필요하지 않습니다.

충돌 가능성을 해결할 수 있는 방식으로 리전 간에 데이터 스토어를 복제해야 합니다. 지연 시간을 단축하고 각 파티션에서 요청 수나 데이터 양의 균형을 유지할 수 있도록 파티션 간에 데이터를 복사하거나 이동하기 위한 도구와 자동화된 프로세스를 생성해야 합니다. 데이터 충돌을 해결하려면 추가 운영 런북도 필요합니다.

가용성 설계 목표
모든 복구를 자동화하기 위해 대규모 투자를 한 결과 1분 이내에 복구를 완료할 수 있다고 가정합니다. 그리고 복구는 수동으로 트리거되지 않으며 분기별로 자동 복구 작업이 1회까지만 수행된다고 가정합니다. 즉, 연간 복구 소요 시간은 4분입니다. 업데이트 과정 전반에 걸쳐

애플리케이션은 계속 온라인 상태라고 가정합니다. 이 가정을 토대로 할 때 **가용성 설계 목표**는 99.999%입니다.

요약

주제	구현
리소스 모니터링	AWS 리전의 DNS 상태를 포함하여 모든 계층과 KPI에서 상태 확인. 구성된 경보가 작동하면 알림이 전송됨. 모든 장애 발생 시 알림이 전송됨. 운영 회의에서 철저하게 추세를 탐지하고 목표 설계를 관리함
수요 변화에 맞게 구현 조정	ELB를 웹 및 자동 조정 애플리케이션 계층에 사용. Aurora RDS의 액티브 및 패시브 리전에 있는 여러 영역에 자동 조정 스토리지 및 읽기 전용 복제본 배포. 정적 안정성을 위해 AWS 리전 간에 데이터 및 인프라가 동기화됨
변경 구현	Canary 또는 블루/그린을 통한 자동 배포. KPI 또는 알림이 애플리케이션에서 감지되지 않은 문제를 나타내는 경우 자동으로 롤백. 배포는 한 번에 하나의 AWS 리전에 있는 단일 격리 영역에 수행됨
데이터 백업	각 AWS 리전에서 RDS를 통한 자동 백업으로 RPO를 충족하고 실전 연습에서 자동 복원을 주기적으로 시행. Aurora RDS 및 S3 데이터는 액티브 리전에서

주제	구현
	패시브 리전으로 비동기식으로 자동 복제됨
복원력을 위한 아키텍처 설계	애플리케이션을 위한 장애 격리 영역을 구현하고 자동 조정을 통해 자가 복구 웹 및 애플리케이션 계층 제공. RDS는 다중 AZ에 위치함. 리전 장애 조치는 자동화됨
복원력 테스트	구성 요소 및 격리 영역 장애 테스트가 파이프라인에 포함되고 운영 직원이 실전 연습을 통해 주기적으로 시행. 알려지지 않은 문제 진단을 위한 플레이북과 근본 원인 분석 프로세스가 마련되어 있으며 문제의 내용과 수정 또는 차단 방법을 알리는 커뮤니케이션 경로가 있음. RCA 수정은 즉각적인 구현 및 배포를 위해 기능 릴리스보다 우선적으로 적용됨
DR(재해 복구) 계획	2개 이상의 리전에 배포되는 액티브–액티브 방식. 인프라는 리전 전체로 완전히 확장되며 정적으로 안정적임. 데이터는 리전 간에 분할되고 동기화됨. RDS를 통한 암호화된 백업. 리전 장애는 실전 연습에서 AWS와 조율하여 시행됩니다. 복원 중에 새 데이터베이스 마스터를 승격해야 할 수 있습니다.

리소스

설명서

- Amazon Builders' Library - Amazon의 소프트웨어 구축 및 운영 방식
- AWS 아키텍처 센터

실습

- AWS Well-Architected 안정성 실습

외부 링크

- Adaptive Queuing Pattern: Fail at Scale
- Calculating Total System Availability

도서

- Robert S. Hammer "Patterns for Fault Tolerant Software"
- Andrew Tanenbaum 및 Marten van Steen "Distributed Systems: Principles and Paradigms"

결론

가용성과 안정성에 관한 주제를 처음 접하는 독자 또는 미션 크리티컬 워크로드 가용성을 극대화하는 데 활용할 수 있는 인사이트를 찾는 독자 모두에게 이 백서가 의견을 떠올리거나, 새 아이디어를 제공하거나, 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 백서에 제공된 설계, 운영 및 복구 중심 권장 사항과 AWS 솔루션 아키텍트의 지식과 경험을 유용하게 활용하시기를 바랍니다. 여러분의 의견, 특히 AWS에서 가용성 수준을 높일 수 있었던 성공 사례를 알려 주십시오. 계정 팀에 문의하거나 웹 사이트에서 AWS로 문의하시면 됩니다.

기고자

이 문서를 작성하는 데 도움을 주신 분들입니다.

- Seth Eliot, Well-Architected 부문 수석 안정성 솔루션 아키텍트, Amazon Web Services
- Adrian Hornsby, 아키텍처 부문 수석 기술 에반젤리스트, Amazon Web Services
- Philip Fitzsimons, Well-Architected 선임 관리자(Amazon Web Services)
- Rodney Lester, Well-Architected 안정성 부문 담당자(Amazon Web Services)
- Kevin Miller, 소프트웨어 개발 책임자(Amazon Web Services)
- Shannon Richards, 선임 기술 프로그램 관리자, Amazon Web Services

추가 자료

다음에서 추가 정보를 참조하십시오.

- AWS Well-Architected 프레임워크

문서 개정

날짜	설명
2020년 4월	다음과 같은 중요한 업데이트 및 신규/수정된 콘텐츠: • "워크로드 아키텍처" 모범 사례 섹션 추가 • 모범 사례를 변경 관리 및 장애 관리 섹션으로 재구성 • 리소스 업데이트 • AWS Global Accelerator, AWS Service Quotas 및 AWS Transit Gateway와 같은 최신 AWS 리소스 및 서비스를 포함하도록 업데이트 • 안정성, 가용성, 복원력에 대한 정의 추가/업데이트 • Well-Architected 검토에 사용되는 AWS Well-Architected Tool(질문 및 모범 사례)과 일치하도록 백서 수정

날짜	설명
	• 설계 원칙 순서 변경, **장애 자동 복구**를 **복구 절차 테스트** 앞으로 이동 • 방정식에 대한 다이어그램 및 형식 업데이트 • 주요 서비스 섹션을 제거하고 대신 주요 AWS 서비스에 대한 참조를 모범 사례에 통합
2019년 10월	연결되지 않는 링크 수정
2019년 4월	부록 A 업데이트
2018년 9월	특정 AWS Direct Connect 네트워킹 권장 사항 및 추가 서비스 설계 목표 추가
2018년 6월	설계 원칙 및 제한 관리 섹션 추가. 링크 업데이트, 업스트림/다운스트림 용어에 대한 모호성 제거, 가용성 시나리오의 나머지 안정성 원칙 주제에 대한 명시적 참조 추가.
2018년 3월	DynamoDB 교차 리전 솔루션을 DynamoDB 글로벌 테이블로 변경 서비스 설계 목표 추가
2017년 12월	애플리케이션 가용성을 포함하도록 가용성 계산 일부 수정
2017년 11월	개념, 모범 사례 및 구현 예제를 비롯하여 고가용성 설계에 대한 지침을 제공하도록 업데이트.
2016년 11월	초판

부록 A: 일부 AWS 서비스 설계 가용성

아래에는 일부 AWS 서비스의 설계 의도에 부합하는 가용성이 나와 있습니다. 이러한 값은 서비스 수준 계약 또는 보증을 나타내는 것이 아니라 각 서비스의 설계 목표에 대한 인사이트를 제공하기 위한 것입니다. 일부의 경우, 서비스에서 가용성 설계 목표에 유의미한 차이가 있는 부분을 구분했습니다. 이 목록은 모든 AWS 서비스가 포함된 것이 아니며 정기적으로 추가 서비스로 정보를 업데이트할 예정입니다. Amazon CloudFront, Amazon Route 53 및 Identity and Access Management Control Plane은 글로벌 서비스를 제공하며 그에 따라 구성 요소 가용성 목표가 제시됩니다. 다른 서비스는 AWS 리전 내에서 서비스를 제공하며 그에 따라 가용성 목표가 제시됩니다. 많은 서비스가 가용 영역 간에 독립성을 유지합니다. 이러한 경우 단일 AZ에 대한 가용성 설계 목표 및 2개 이상의 가용 영역이 사용되는 경우의 가용성 설계 목표를 제시합니다.

참고: 다음 표의 수치는 내구성(데이터의 장기적 보존)이 아니라 가용성(데이터 또는 기능에 대한 액세스)을 의미합니다.

서비스	구성 요소	가용성 설계 목표
Amazon API Gateway	제어 영역	99.950%
	데이터 영역	99.990%
Amazon Aurora	제어 영역	99.950%
	단일 AZ 데이터 영역	99.950%
	다중 AZ 데이터 영역	99.990%
AWS CloudFormation	서비스	99.950%
Amazon CloudFront	제어 영역	99.900%
	데이터 영역(콘텐츠 전송)	99.990%
Amazon CloudSearch	제어 영역	99.950%

서비스	구성 요소	가용성 설계 목표
	데이터 영역	99.950%
Amazon CloudWatch	CW 지표(서비스)	99.990%
	CW 이벤트(서비스)	99.990%
	CW 로그(서비스)	99.950%
AWS Database Migration Service	제어 영역	99.900%
	데이터 영역	99.950%
AWS Data Pipeline	서비스	99.990%
Amazon DynamoDB	서비스(표준)	99.990%
	서비스(글로벌 테이블)	99.999%
Amazon EC2	제어 영역	99.950%
	단일 AZ 데이터 영역	99.950%
	다중 AZ 데이터 영역	99.990%
Amazon ElastiCache	서비스	99.990%
Amazon Elastic Block Store	제어 영역	99.950%
	데이터 영역(볼륨 가용성)	99.999%
Amazon Elasticsearch	제어 영역	99.950%
	데이터 영역	99.950%
Amazon EMR	제어 영역	99.950%
Amazon S3 Glacier	서비스	99.900%
AWS Glue	서비스	99.990%

서비스	구성 요소	가용성 설계 목표
Amazon Kinesis Data Streams	서비스	99.990%
Amazon Kinesis Data Firehose	서비스	99.900%
Amazon Kinesis Video Streams	서비스	99.900%
Amazon Neptune	서비스	99.900%
Amazon RDS	제어 영역	99.950%
	단일 AZ 데이터 영역	99.950%
	다중 AZ 데이터 영역	99.990%
Amazon Rekognition	서비스	99.980%
Amazon Redshift	제어 영역	99.950%
	데이터 영역	99.950%
Amazon Route 53	제어 영역	99.950%
	데이터 영역(쿼리 해상도)	100.000%
Amazon SageMaker	데이터 영역(모델 호스팅)	99.990%
	제어 영역	99.950%
Amazon S3	서비스(표준)	99.990%
AWS Auto Scaling	제어 영역	99.900%
	데이터 영역	99.990%
AWS Batch	제어 영역	99.900%

서비스	구성 요소	가용성 설계 목표
	데이터 영역	99.950%
AWS CloudHSM	제어 영역	99.900%
	단일 AZ 데이터 영역	99.900%
	다중 AZ 데이터 영역	99.990%
AWS CloudTrail	제어 영역(구성)	99.900%
	데이터 영역(데이터 이벤트)	99.990%
	데이터 영역(관리 이벤트)	99.999%
AWS Config	서비스	99.950%
AWS Direct Connect	제어 영역	99.900%
	단일 위치 데이터 영역	99.900%
	다중 위치 데이터 영역	99.990%
Amazon Elastic File System	제어 영역	99.950%
	데이터 영역	99.990%
AWS Identity and Access Management	제어 영역	99.900%
	데이터 영역(인증)	99.995%
AWS IoT Core	서비스	99.900%
AWS IoT Device Management	서비스	99.900%
AWS IoT Greengrass	서비스	99.900%
AWS Lambda	함수 호출	99.950%

서비스	구성 요소	가용성 설계 목표
AWS Secrets Manager	서비스	99.900%
AWS Shield	제어 영역	99.500%
	데이터 영역(탐지)	99.000%
	데이터 영역(완화)	99.900%
AWS Storage Gateway	제어 영역	99.950%
	데이터 영역	99.950%
AWS X-Ray	제어 영역(콘솔)	99.900%
	데이터 영역	99.950%
EC2 Container Service	제어 영역	99.900%
	EC2 Container Registry	99.990%
	EC2 Container Service	99.990%
Elastic Load Balancing	제어 영역	99.950%
	데이터 영역	99.990%
Key Management System(KMS)	제어 영역	99.990%
	데이터 영역	99.995%