AWS Well-Architected Framework

지속 가능성 원칙

지속 가능성 원칙: AWS Well-Architected Framework

Table of Contents

지속 가능성 원칙 - AWS Well-Architected Framework

게시 날짜: 2024년 6월 27일 (문서 개정)

이 백서에서는 Amazon Web Services(AWS) Well-Architected Framework의 지속 가능성 원칙에 중점을 둡니다. 여기에서는 설계 원칙, 운영 가이드, 모범 사례, 잠재적인 상충 관계, AWS 워크로드의 지속 가능성 목표를 달성하기 위해 사용할 수 있는 개선 계획이 제공됩니다.

소개

AWS Well-Architected Framework는 AWS에서 워크로드를 구축할 때 내리는 의사 결정의 장점과 단점을 이해하는 데 도움이 됩니다. 이 프레임워크를 사용하면 AWS 클라우드에서 안전하고 안정적이며 효율적이고 경제적이면서 지속 가능한 워크로드를 설계하고 운영하기 위한 설계 모범 사례를 알아볼 수 있습니다. 이 프레임워크는 모범 사례를 기준으로 아키텍처를 일관적으로 측정하고 개선할 영역을 파악하는 방법을 제시합니다. 워크로드가 효과적으로 설계되면 비즈니스 성과를 지원하는 역량이 매우 커집니다.

이 프레임워크는 다음 6가지 원칙을 기반으로 합니다.

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

이 문서에서는 지속 가능성 원칙에 중점을 두며 지속 가능성의 범위 내에서도 환경적 지속 가능성에 초점을 맞춥니다. 이 문서는 최고 기술 책임자(CTO), 아키텍트, 개발자와 같은 기술 업무 담당자와 운영 팀원을 대상으로 작성되었습니다.

이 문서의 내용을 확인하고 나면 지속 가능성을 염두에 두고 클라우드 아키텍처를 설계할 때 사용할 AWS의 현재 권장 사항과 전략을 파악할 수 있습니다. 이 문서에 제시된 관행을 실천하면 효율성을 극대화하고 낭비를 줄이는 아키텍처를 구축할 수 있습니다.

클라우드 지속 가능성

지속 가능성 원칙은 비즈니스 활동의 장기적인 환경, 경제 및 사회적 영향을 다룹니다. 유효 UN 세계 환경 개발 위원회는 지속 가능한 개발을 '미래 세대가 자신들의 니즈를 충족하는 능력을 저해하지 않으면서 현재의 니즈를 충족하는 개발'로 정의하고 있습니다. 귀사의 비즈니스 또는 조직은 직간접적인 탄소 배출, 재활용할 수 없는 폐기물, 깨끗한 물과 같은 공유 자원에 대한 손상 등의 부정적인 환경 영향을 미칠 수 있습니다.

클라우드 워크로드를 구축할 때 지속 가능성의 실천은 사용된 서비스의 영향을 이해하고, 전체 워크로드 수명 주기 동안 영향을 정량화하고, 이러한 영향을 줄이기 위한 설계 원칙과 모범 사례를 적용하는 것입니다. 이 문서에서는 환경 영향, 특히 에너지 소비 및 효율성에 중점을 두고 있는데, 이는 건축가가 자원 사용을 줄이기 위한 직접적인 조치를 알아낼 수 있는 중요한 수단이기 때문입니다.

환경 영향에 중점을 둘 때는 이 영향이 일반적으로 어떻게 설명되며 조직의 자체 배출량 계산에 어떤 영향을 미치는지 이해해야 합니다. 유효 Greenhouse Gas Protocol(온실 가스 협약) 은 탄소 배출을 다음과 같은 범위로 분류하며 각 범위에 AWS와 같은 클라우드 공급자를 위한 관련 배출 예시를 제시합니다.

- 범위 1: 조직 또는 조직의 통제하에 있는 활동의 모든 직접적인 배출. 예를 들면 데이터 센터 백업 발전기에서의 연소입니다.
- 범위 2: 데이터 센터 및 기타 시설에 전력을 공급하기 위해 구매하거나 사용한 전기로부터의 간접적 배출. 예를 들면 상업용 발전에서의 배출입니다.
- 범위 3: 조직에서 통제할 수 없는 원천에서 발생하는 조직 활동으로부터의 기타 모든 배출. AWS를 예로 들면 데이터 센터 건설 및 데이터 센터에 배포되는 IT 하드웨어의 제조 및 운송과 관련한 배출이 있습니다.

AWS 고객의 관점에서는 AWS에서 실행되는 고객 워크로드로부터의 배출이 간접적 배출 및 범위 3 배출에 해당합니다. 배포된 각 워크로드는 이전 범위의 각각에 해당하는 총 AWS 배출의 일부를 발생시킵니다. 실제 배출량은 워크로드에 따라 다르며 사용한 AWS 서비스, 해당 서비스에서 소비한 에너지, 워크로드가 실행되는 AWS 데이터 센터에 제공되는 전력망의 탄소 집약도, AWS의 재생 가능 에너지 조달과 같은 요인에 따라 달라집니다.

이 문서에서는 먼저 환경적 지속 가능성에 대한 공동 책임 모델을 설명한 후 AWS 데이터 센터에서 실행되는 데 필요한 총 리소스 양을 줄여 워크로드의 영향을 최소화할 수 있는 아키텍처상의 모범 사례를 제공합니다.

공동 책임 모델

환경적 지속 가능성은 고객과 AWS 간의 공동 책임입니다.

- AWS는 효율적인 공유 인프라와 물 관리, 재생 가능 전력 소싱을 통해 클라우드 작업량 자체의 지속 가능성을 최적화하는 데 책임이 있습니다.
- 고객은 워크로드와 리소스 사용량 최적화, 워크로드에 배포해야 하는 총 리소스 최소화를 통한 클라우드 in 내부의 지속 가능성에 책임이 있습니다.

공동 책임 모델

클라우드의 지속 가능성

클라우드 공급자는 일반적인 온프레미스 공급자보다 탄소 발자국을 적게 남기며 에너지 효율이 더 높습니다. 효율적인 전력 및 냉각 기술에 투자하고 에너지 효율적인 서버를 운영하며 높은 서버 활용률을 달성하기 때문입니다. 클라우드 워크로드는 네트워킹, 전력, 냉각, 물리적 시설과 같은 공유 리소스를 활용하여 영향을 축소합니다. 클라우드 워크로드를 새롭게 등장하는 더 효율적인 기술로 마이그레이션하고, 클라우드 기반 서비스를 이용하여 워크로드를 혁신함으로써 지속 가능성을 높일 수 있습니다.

리소스

- Amazon Web Services로의 이전을 통한 탄소 감축의 기회
- 지속 가능성 솔루션을 구현하는 AWS

클라우드의 지속 가능성

클라우드에서의 지속 가능성이란 주로 프로비저닝된 리소스의 이점을 극대화하고 필요한 총 리소스를 최소화하면서 워크로드의 모든 구성 요소에서 에너지 절감과 효율성에 초점을 맞춘 지속적인 노력을 말합니다. 이러한 노력은 초기에 효율적인 프로그래밍 언어를 선택하고, 현대적인 알고리즘을 채택하고, 능률적인 데이터 스토리지 기술을 사용하여 적절한 규모의 효과적인 컴퓨팅 인프라를 배포하고 고성능 최종 사용자 하드웨어 요구 사항을 최소화하는 등 다양하게 나타날 수 있습니다.

클라우드를 통한 가능성

배포한 워크로드의 영향을 최소화하는 데 더해 AWS 클라우드를 사용하여 더 광범위한 지속 가능성 과제를 해결하도록 설계된 워크로드를 실행할 수 있습니다. 이런 과제의 예로는 탄소 배출량 감축, 에너지 소비 절감, 물 재활용, 비즈니스 또는 조직의 다른 부분에서의 폐기물 감축 등이 있습니다.

지속 가능성 딥 러닝을 클라우드를 통한 지속 가능성은 AWS 기술을 사용하여 더 광범위한 지속 가능성의 과제를 해결할 때 실현됩니다. 예를 들어, Amazon Monitron 등의 기계 학습 서비스를 사용하여 산업용 기계의 이상 동작을 탐지할 수 있습니다. 이 탐지 데이터를 사용하면 선제적으로 유지 보수를 수행하여 예기치 않은 장비 오류로 인한 환경적 문제의 위험을 줄이고 기계가 계속해서 최고의 효율성으로 작동하도록 할 수 있습니다.

클라우드에서의 지속 가능성 설계 원칙

클라우드 워크로드를 아키텍팅할 때 지속 가능성을 극대화하고 영향을 최소화하도록 이 설계 원칙을 적용하세요.

- 영향 이해: 클라우드 워크로드의 영향을 측정하고 워크로드의 향후 영향을 모델링합니다. 고객의 제품 사용으로 인한 영향과 최종 폐기 및 사용 중지로 인한 영향을 포함하여 영향의 모든 원인을 포함합니다. 작업 단위당 필요한 리소스 및 배출량을 검토하여 생산량을 클라우드 워크로드의 총 영향과 비교합니다. 이 데이터를 사용하여 핵심 성과 지표(KPI)를 설정하고, 영향을 줄이면서 생산성을 개선하는 방법을 평가하며, 시간 경과에 따른 제안된 변경의 영향을 예측할 수 있습니다.
- 지속 가능성 목표 수립: 각 클라우드 워크로드에 대해 트랜잭션당 필요한 컴퓨팅 및 스토리지 리소스의 절감과 같은 장기적인 지속 가능성 목표를 설정합니다. 기존 워크로드에 대한 지속 가능성 개선의 투자 수익률을 모델링하고 소유자에게 지속 가능성 목표에 투자하는 데 필요한 리소스를 제공합니다. 성장을 계획하고 워크로드를 설계하여 성장으로 인해 사용자당 또는 트랜잭션당 등 적절한 단위에 대해 측정된 영향 강도를 줄입니다. 목표를 통해 비즈니스 또는 조직의 보다 광범위한 지속 가능성 목표를 지원하고, 회귀를 식별하며, 잠재적 개선 영역의 우선 순위를 지정할 수 있습니다.

- 활용률 극대화: 워크로드 크기를 적절하게 조정하고 효율적인 설계를 구현하여 높은 활용률을 보장하고 기본 하드웨어의 에너지 효율성을 극대화합니다. 호스트당 기준 전력 소비로 인해 30% 활용률로 실행되는 호스트 두 개는 60%로 실행되는 호스트 하나보다 효율성이 떨어집니다. 동시에 유휴 리소스, 프로세싱 및 스토리지를 없애거나 최소화하여 워크로드에 전력을 공급하는 데 필요한 총 에너지를 줄입니다.
- 보다 효율적인 최신 하드웨어와 소프트웨어 제품 및 서비스 예측 및 도입: 파트너와 공급업체가 업스트림 개선을 통해 클라우드 워크로드의 영향을 줄일 수 있도록 지원합니다. 새롭고 더 효율적인 하드웨어와 소프트웨어 제품 및 서비스를 지속적으로 모니터링하고 평가합니다. 새롭고 효율적인 기술을 신속하게 도입할 수 있도록 유연성을 고려하여 설계합니다.
- 관리형 서비스 사용: 광범위한 고객 기반에서 서비스를 공유하면 리소스 활용률을 극대화하여 클라우드 워크로드를 지원하는 데 필요한 인프라의 양을 줄일 수 있습니다. 예를 들어, 고객은 워크로드를 AWS 클라우드로 마이그레이션하고 AWS가 대규모로 운영하며 효율적인 운영을 책임지는 서버리스 컨테이너용 AWS Fargate 등 관리형 서비스를 채택하여 전력 및 네트워킹과 같은 일반적인 데이터 센터 구성 요소의 영향을 공유할 수 있습니다. Amazon S3 수명 주기 구성 또는 Amazon EC2 Auto Scaling을 사용하여 자주 액세스하지 않는 데이터를 콜드 스토리지로 자동 이동함으로써 수요에 맞게 용량을 조정하는 등 영향을 최소화할 수 있는 관리형 서비스를 사용합니다.
- 클라우드 워크로드의 다운스트림 영향 감소: 서비스를 사용하는 데 필요한 에너지 또는 리소스의 양을 줄입니다. 고객이 서비스를 사용하기 위해 디바이스를 업그레이드할 필요가 없도록 하거나 필요성을 줄입니다. Device Farm을 사용하여 테스트함으로써 예상되는 영향을 파악하고 고객과의 테스트를 통해 서비스 사용으로 인한 실제 영향을 이해합니다.

개선 프로세스

아키텍처 개선 프로세스에는 현재 보유하고 있는 것과 개선하기 위해 무엇을 할 수 있는지를 이해하고 개선 대상 선택, 개선 테스트, 성공적인 개선 도입, 성공 정량화, 그리고 다른 곳에 사용할 수 있도록 학습 내용을 공유한 다음 주기를 반복하는 것이 포함됩니다.

개선 목표는 다음과 같습니다.

- 폐기물, 저조한 활용률, 유휴 또는 미사용 리소스 제거
- 소비하는 리소스의 가치 극대화

> **Note**
>
> 프로비저닝하는 모든 리소스를 사용하고 가능한 한 최소한의 리소스로 같은 작업을 완료합니다.

최적화의 초기 단계에서는 먼저 낭비가 있거나 활용률이 낮은 영역을 제거한 다음 특정 워크로드에 맞는 보다 표적화된 최적화로 이동합니다.

시간 경과에 따른 리소스 사용의 변화를 모니터링합니다. 누적된 변경으로 인해 리소스 사용이 비효율적이거나 크게 증가하는 위치를 식별합니다. 사용의 변화를 해결하고 우선적인 개선 사항을 구현하기 위해 개선의 필요성을 결정합니다.

다음 단계는 클라우드 워크로드에 지속 가능성에 중점을 둔 개선 사항을 평가하고 우선순위를 설정하고 개선 사항을 테스트 및 배포하는 반복적인 프로세스로 설계되었습니다.

1. 개선 대상 파악: 이 문서에 설명된 지속 가능성을 위한 모범 사례를 기준으로 워크로드를 검토하고 개선 대상을 파악합니다.
2. 구체적인 개선 평가: 특정 변경 사항의 잠재적 개선, 예상 비용 및 비즈니스 위험 요인을 평가합니다.
3. 개선 우선순위 설정 및 계획: 최소한의 비용 및 위험으로 가장 큰 개선을 가져오는 변경 사항의 우선순위를 설정하고 테스트 및 구현을 위한 계획을 수립합니다.
4. 개선 사항 테스트 및 검증:테스트 환경에서 변경을 구현하여 잠재적인 개선 사항을 검증합니다.
5. 변경 사항을 프로덕션에 배포: 프로덕션 환경에 변경 사항을 구현합니다.
6. 결과 측정 및 성공 모사: 워크로드 전반에서 성공을 모사할 만한 기회를 찾고 수용할 수 없는 결과가 나오는 경우 변경 사항을 되돌립니다.

시나리오 예

다음 예시 시나리오는 개선 프로세스의 각 단계를 설명하기 위해 이 문서에서 이후에 언급됩니다.

여러분의 회사에는 Amazon EC2 인스턴스에서 복잡한 이미지 조작을 수행하고 사용자 액세스를 위해 수정 파일과 원본 파일을 저장하는 워크로드가 있습니다. 처리 활동에 CPU가 많이 사용되며 결과 파일의 크기가 매우 큽니다.

개선 대상 파악

지속 가능성 목표를 달성하는 데 도움이 되는 모범 사례를 파악합니다. 이 문서에서 이후에 개선을 위한 모범 사례와 권장 사항에 대한 상세한 설명을 확인할 수 있습니다.

사용되는 워크로드와 리소스를 검토합니다. 식별 용량이 큰 배포와 자주 사용되는 리소스 등의 핫 스팟을 파악합니다. 이 핫 스팟에 리소스의 효율적인 사용을 개선하고 비즈니스 성과를 달성하는 데 필요한 총 리소스 양을 줄일 기회가 있는지 평가합니다.

모범 사례를 기준으로 워크로드를 검토하고 개선 후보를 파악합니다.

이 단계를 다음에 적용합니다. 시나리오 예그 결과 다음 모범 사례를 개선을 위한 대상으로 파악합니다.

- 요구 사항을 충족하는 데 필요한 최소한의 하드웨어 사용
- 조직의 데이터 액세스 및 스토리지 패턴을 가장 잘 지원하는 기술 사용

리소스

- Optimizing your AWS Infrastructure for Sustainability, Part I: Compute(지속 가능성을 위한 AWS 인프라 최적화, 파트 I: 컴퓨팅)
- Optimizing your AWS Infrastructure for Sustainability, Part II: Storage(지속 가능성을 위한 AWS 인프라 최적화, 파트 II: 스토리지)
- Optimizing your AWS Infrastructure for Sustainability, Part III: Networking(지속 가능성을 위한 AWS 인프라 최적화, 파트 III: 네트워킹)

구체적인 개선 평가

작업 단위를 완료하기 위해 워크로드에서 프로비저닝한 리소스를 파악합니다. 잠재적인 개선 사항을 평가하고 그 영향과 구현 비용, 관련 위험을 예측합니다.

시간 경과에 따른 개선 사항을 측정하려면 먼저 AWS에 프로비저닝한 리소스를 파악하고 그 리소스가 어떻게 사용되는지 파악합니다.

AWS 사용량 전체를 개략적으로 살펴보고 AWS Cost and Usage Report를 사용하여 핫 스팟을 파악합니다. 이 AWS 샘플 코드를 사용하여 Amazon Athena의 도움으로 보고서를 검토 및 분석합니다.

프록시 지표

특정 변경 사항을 평가할 때는 해당 변경 사항이 관련 리소스에 미치는 영향을 가장 잘 수량화하는 지표도 평가해야 합니다. 이런 지표를 프록시 지표라고 합니다. 평가하는 개선 사항의 유형과 개선 대상 리소스를 가장 잘 반영하는 프록시 지표를 선택합니다. 이 지표는 시간 경과에 따라 변경될 수 있습니다.

워크로드를 지원하기 위해 프로비저닝되는 리소스에는 컴퓨팅, 스토리지, 네트워크 리소스가 있습니다. 프로비저닝된 리소스를 프록시 지표로 평가하여 리소스가 어떻게 사용되는지 살펴봅니다.

비즈니스 성과를 달성하기 위해 프로비저닝된 리소스를 프록시 지표를 사용하여 측정합니다.

리소스	프록시 지표 예시	개선 목표
컴퓨팅	vCPU 분	프로비저닝된 리소스의 활용률 극대화
스토리지	프로비저닝된 GB	총 프로비저닝된 양 절감
네트워크	전송된 GB 또는 전송된 패킷	총 전송된 양 절감 및 전송된 거리 단축

비즈니스 지표

비즈니스 성과 달성을 수량화하는 비즈니스 지표를 선택합니다. 비즈니스 지표는 워크로드에서 제공하는 가치(예: 동시 액티브 사용자의 수, 처리된 API 호출, 완료된 트랜잭션 수)를 반영해야 합니다. 이 지표는 시간 경과에 따라 변경될 수 있습니다. 재무 관련 비즈니스 지표를 평가할 때는 트랜잭션 값의 비일관성으로 인해 비교가 무의미해질 수 있으니 주의하시기 바랍니다.

핵심 성과 지표

다음 공식으로 프로비저닝된 리소스를 달성한 비즈니스 성과로 나누어 작업 단위당 프로비저닝된 리소스를 파악합니다.

$$\text{작업 단위당 프로비저닝된 리소스} = \frac{\text{프로비저닝된 리소스에 대한 프록시 지표}}{\text{결과에 대한 비즈니스 지표}}$$

KPI 공식

작업 단위당 리소스를 KPI로 사용합니다. 프로비저닝된 리소스를 비교의 근거로 삼아 기준을 정합니다.

리소스	KPI 예시	개선 목표
컴퓨팅	트랜잭션당 vCPU 분	프로비저닝된 리소스의 활용률 극대화
스토리지	트랜잭션당 GB	총 프로비저닝된 양 절감
네트워크	트랜잭션당 전송된 GB 또는 트랜잭션당 전송된 패킷	총 전송된 양 절감 및 전송된 거리 단축

개선 예측

프로비저닝된 리소스의 수량 감소(프록시 지표로 나타남)와 작업 단위당 프로비저닝된 기준 리소스의 비율 변화로 개선을 예측합니다.

리소스	KPI 예시	개선 목표
컴퓨팅	트랜잭션당 vCPU 절감률(%)	활용률 극대화
스토리지	트랜잭션당 GB 절감률(%)	총 프로비저닝된 양 절감
네트워크	트랜잭션당 전송된 GB 또는 트랜잭션당 전송된 패킷 절감률(%)	총 전송된 양 절감 및 전송된 거리 단축

개선 평가

기대되는 순이익을 기준으로 잠재적인 개선을 평가합니다. 구현 및 유지 관리를 위한 시간, 비용, 노력 수준과 예기치 않은 영향 등의 비즈니스 위험을 평가합니다.

대상이 분명한 개선에는 사용되는 리소스의 유형 간에 상충되는 부분이 있는 경우가 많습니다. 예를 들어, 컴퓨팅 소비를 줄이기 위해서는 결과를 저장하거나 전송되는 데이터를 제한하거나 결과를 클라이언트에게 전송하기 전에 데이터를 처리할 수 있습니다. 그러나 이러한 상충 관계는 향후에 자세히 논의합니다.

워크로드에 대한 위험을 평가할 때는 보안, 안정성, 성능 효율성, 비용 최적화, 워크로드 운영 역량에 대한 개선이 미치는 영향 등 기능과 관련이 없는 요구 사항을 포함합니다.

이 단계를 다음에 적용합니다. 시나리오 예다음 결과와 함께 대상 개선 사항을 평가합니다.

모범 사례	대상 개선 사항	잠재적	비용	위험
요구 사항을 충족하는 데 필요한 최소한의 하드웨어 사용	활용률이 저조한 기간을 줄이기 위해 예측 크기 조정 구현	보통	낮음	낮음
조직의 데이터 액세스 및 스토리지 패턴을 가장 잘 지원하는 기술 사용	총 스토리지와 이를 달성하기 위한 시간을 줄이기 위해 더 효과적인 압축 메커니즘 구현	높음	낮음	낮음

예측 스케줄링을 구현하면 활용이 저조하거나 사용되지 않는 인스턴스가 소비하는 vCPU 시간을 줄여 소비되는 리소스의 양이 11% 절감되므로 기존의 크기 조정 메커니즘보다 좀 더 나은 이점을 얻을 수 있습니다. 여기에 드는 비용은 적으며 클라우드 리소스 구성과 Amazon EC2 Auto Scaling에 대한 예측 크기 조정 작업이 필요합니다. 요구가 예측을 초과하여 사후 대응적으로 스케일 아웃을 수행할 때 성능이 제한된다는 위험이 있습니다.

더 효과적인 압축을 구현하면 원본 및 수정 이미지의 파일 크기가 크게 줄어들어 영향이 아주 크며 프로덕션에서 요구되는 스토리지가 25% 절감될 것으로 예측됩니다. 이 새로운 알고리즘을 구현하면 위험은 거의 없으며 노력을 많이 들이지 않고 대체할 수 있습니다.

개선 우선순위 설정 및 계획

예상되는 가장 큰 영향과 가장 낮은 비용 및 수용 가능한 위험을 기준으로 파악한 개선 사항의 우선순위를 설정합니다.

어떤 개선 사항에 처음으로 집중할지 결정하고 리소스 계획 및 개발 로드맵에 해당 개선 사항을 포함합니다.

이 단계를 다음에 적용합니다. 시나리오 예그 결과 다음과 같이 대상 개선 사항의 우선순위를 정합니다.

우선순위	개선	잠재적	비용	위험
1	더 효과적인 압축 메커니즘 구현	높음	낮음	낮음
2	예측 크기 조정 구현	보통	낮음	낮음

파일 압축 업데이트는 잠재력이 크고 비용은 많이 들지 않으며 위험이 낮기 때문에 회사에서 가치를 높게 두는 대상이며 예측 크기 조정 구현보다 우선순위가 높습니다. 예측 크기 조정은 잠재적 영향이 보통이며 비용과 위험은 낮으므로 새로운 파일 압축 구현이 완료된 후의 개선 우선순위로 정합니다.

팀원을 배정하여 개선된 파일 압축을 구현하고 백로그에 예측 크기 조정을 추가하도록 합니다.

개선 사항 테스트 및 검증

투자를 최소화하고 간단한 테스트를 수행하여 대규모 노력의 위험을 줄입니다.

워크로드를 대표하는 복사본을 테스트 환경에 구현하여 테스트 및 검증 수행의 비용과 위험을 제한합니다. 사전 정의된 테스트 트랜잭션 집합을 수행하고 프로비저닝된 리소스를 측정하고 작업 단위당 사용되는 리소스를 파악하여 테스트 기준을 정합니다.

대상 개선 사항을 테스트 환경에 구현하고 같은 조건에서 같은 방법론을 사용하여 테스트를 반복합니다. 그런 다음, 개선 사항을 적용하고 프로비저닝된 리소스와 작업 단위당 사용되는 리소스를 측정합니다.

작업 단위당 프로비저닝되는 리소스 기준의 비율 변화를 계산하고 프로덕션 환경에서 프로비저닝되는 리소스의 예상 절감량을 파악합니다. 이 값을 예상 값과 비교합니다. 결과가 수용 가능한 개선 수준인

지 판단합니다. 소비되는 추가 리소스의 상충 관계로 인해 개선 사항의 순 이익이 수용 가능한 범위를 벗어나게 되는지 평가합니다.

개선이 성공인지, 프로덕션에 변경을 구현하는 데 리소스를 투자해야 하는지 판단합니다. 이번에 변경이 성공적이지 않다고 평가되면 다음 대상을 테스트 및 검증하는 데 리소스를 사용하고 개선 주기에 맞게 계속 진행합니다.

작업 단위당 프로비저닝된 리소스 절감률(%)	프로비저닝된 리소스의 절감량	조치
기대 충족	기대 충족	개선 계속
기대 미충족	기대 충족	개선 계속
기대 충족	기대 미충족	다른 개선 진행
기대 미충족	기대 미충족	다른 개선 진행

이 단계를 다음에 적용합니다. 시나리오 예성공을 검증하기 위해 테스트를 진행합니다.

개선된 압축 알고리즘에서 테스트를 수행한 후 작업 단위당 프로비저닝된 리소스(원본 이미지 및 수정 이미지를 위해 필요한 스토리지)의 절감률이 프로비저닝된 스토리지에서 30% 절감으로 기대를 충족했으며 컴퓨팅 로드 증가는 무시할 만한 수준이었습니다.

여러분은 프로덕션에서 기존 파일에 개선된 압축 알고리즘을 적용하는 데 필요한 추가 컴퓨팅 리소스는 달성한 스토리지 절감에 비하면 대수롭지 않은 수준이라고 판단합니다. 필요한 리소스(스토리지 TB)의 절감량으로 성공을 확인했고 개선 사항을 프로덕션에 배포하도록 승인되었습니다.

변경 사항을 프로덕션에 배포

테스트, 검증, 승인을 마친 개선 사항을 프로덕션에 구현합니다. 제한된 배포를 사용하여 구현하고 워크로드의 작동을 확인하고 제한된 배포 내에서 프로비저닝된 리소스 및 작업 단위당 사용되는 리소스의 실제 절감을 테스트한 후 변경으로 인해 예기치 않은 결과가 나타나는지 확인합니다. 테스트에 성공하면 전체 배포를 진행합니다.

테스트에 실패하거나 변경으로 인한 의도치 않은 결과가 수용할 수 없는 수준이면 변경 사항을 되돌립니다.

이 단계를 다음에 적용합니다. 시나리오 예다음 조치를 취합니다.

제한된 배포를 사용하여 블루-그린 배포 방법론으로 프로덕션에 변경 사항을 구현합니다. 새로 배포된 인스턴스에 대한 기능 테스트에 성공합니다. 원본 및 수정 이미지 파일을 위해 프로비저닝된 스토리지가 평균 26% 절감된 것을 확인합니다. 새로운 파일 압축을 위해 컴퓨팅 로드가 증가하는 것은 확인되지 않습니다.

이미지 파일을 압축하는 데 소요되는 시간이 예기치 않게 줄어들었는데, 여러분은 이것이 새로운 압축 알고리즘을 위해 고도로 최적화된 코드 덕분이라고 생각합니다.

새로운 버전의 전체 배포를 진행합니다.

결과 측정 및 성공 모사

다음과 같이 결과를 측정하고 성공을 모사합니다.

- 작업 단위당 프로비저닝된 리소스에 대한 최초 개선 사항과 프로비저닝된 리소스의 절감량을 측정합니다.
- 초기 예측과 테스트 결과를 프로덕션 측정과 비교합니다. 차이에 기여한 요인을 파악하고 예측과 테스트 방법론을 적절히 업데이트합니다.
- 성공 여부와 성공의 수준을 판단하고 이해 관계자와 결과를 공유합니다.
- 테스트 실패 또는 변경으로 인한 의도치 않은 부정적 결과로 변경 사항을 되돌려야 했다면 기여 요인을 파악합니다. 가능한 곳에 반복 적용하거나 변경 사항의 목표를 달성할 수 있는 새로운 방법을 평가합니다.
- 배운 교훈을 바탕으로 표준을 만들고 유사한 이점을 얻을 수 있는 다른 시스템에 성공적인 개선 사항을 적용합니다. 방법론과 관련 아티팩트, 순 이익을 파악하여 팀과 조직 전체에 공유함으로써 다른 사람들이 표준을 적용하고 성공을 모사할 수 있도록 합니다.
- 작업 단위당 프로비저닝된 리소스를 모니터링하고 변경 사항과 시간 경과에 따른 총 영향을 추적합니다. 워크로드에 대한 변경 사항 또는 고객의 워크로드 사용 방식에 대한 변경 사항은 개선의 효과에 영향을 미칠 수 있습니다. 개선의 효과가 단기적으로 크게 줄어들거나 시간이 경과할수록 누적 효과가 줄어들면 개선 기회를 다시 평가하세요.
- 시간 경과에 따른 개선의 순 이익(개선 사항을 적용한 다른 팀에서 얻은 이점 포함)을 수량화하여 개선 활동의 투자 이익을 확인합니다.

이 단계를 다음에 적용합니다. 시나리오 예다음 결과를 측정합니다.

기존 이미지 파일에 새로운 압축 알고리즘을 배포하고 적용한 후 워크로드의 스토리지 요구 사항이 초기에 23% 줄어드는 개선을 보였습니다.

측정된 값은 초기 예측(25%)과 거의 일치하며 테스트(30%)와 비교했을 때의 큰 차이는 테스트에서 사용된 이미지 파일이 프로덕션에 있는 이미지 파일을 대표하지 않는 데서 온 결과로 판명 났습니다. 여러분은 테스트 이미지 집합이 프로덕션의 이미지를 더 적절히 반영하도록 수정합니다.

이 개선은 완전한 성공으로 간주됩니다. 프로비저닝된 스토리지의 총 절감률은 예측값인 25%보다 2% 부족했지만 23%는 여전히 지속 가능성 영향에서 큰 개선이며 이에 따른 비용 절감 효과도 얻었습니다.

이 변경으로 인한 의도치 않은 유일한 결과는 압축 수행에 소요되는 시간 단축이라는 이점이었으며 그에 따라 소비되는 vCPU도 줄었습니다. 이 개선은 고도로 최적화된 코드 덕분인 것으로 판단되었습니다.

여러분은 내부적으로 오픈 소스 프로젝트를 만들어 코드와 관련 아티팩트, 변경 사항 구현 가이드와 구현 결과를 공유합니다. 내부 오픈 소스 프로젝트 덕분에 팀이 각자의 영구 파일 스토리지 사용 사례에 코드를 사용하기가 쉬워졌습니다. 팀은 이 개선을 표준으로 적용합니다. 내부 오픈 소스 프로젝트의 부차적인 이점은 솔루션을 도입하는 사람은 누구나 솔루션의 개선 사항으로 이점을 얻을 수 있으며 누구나 프로젝트의 개선에 기여할 수 있다는 점입니다.

여러분은 성공을 게시하고 오픈 소스 프로젝트를 조직 전체에 공유합니다. 이 솔루션을 도입하는 모든 팀이 최소한의 투자로 같은 이점을 얻을 수 있으며 여러분의 투자에서 얻은 순 이익을 늘릴 수 있습니다. 여러분은 이 데이터를 지속적인 성공 사례로 게시합니다.

여러분은 계속해서 시간 경과에 따른 개선의 영향을 모니터링하고 필요에 따라 내부 오픈 소스 프로젝트를 변경할 것입니다.

비기능적 요구 사항으로서의 지속 가능성

비즈니스 요구 사항 목록에 지속 가능성을 추가하면 보다 경제적인 솔루션을 얻을 수 있습니다. 사용하는 리소스로부터 더 많은 가치를 얻고 더 적은 리소스를 사용하는 데 집중하면 AWS에서 직접적으로 비용을 절감할 수 있습니다. AWS에서는 사용한 만큼만 비용을 지불하기 때문입니다.

지속 가능성 목표를 달성하기 위해 가동 시간, 가용성 또는 반응 시간과 같은 다른 전통적인 지표를 포기할 필요가 없습니다. 서비스 수준에 눈에 띄는 영향 없이 지속 가능성을 상당한 수준으로 증진할 수 있습니다. 사소한 절충이 필요할 수 있지만 이러한 절충으로 얻은 지속 가능성의 개선 사항이 서비스 품질 변화보다 더 클 수 있습니다.

팀원들이 기능 요구 사항을 개발할 때 지속적으로 지속 가능성 개선 사항을 실험하도록 독려하세요. 또한 팀에서는 목표를 수립할 때 프록시 지표를 포함하여 워크로드를 개발할 때 리소스 밀도를 평가하도록 해야 합니다.

다음은 사용하는 클라우드를 줄일 수 있는 상충 관계의 예시입니다.

결과의 품질 조정: 근사 컴퓨팅을 통해 QoR(결과 품질)을 낮추는 대신 워크로드 집약도를 줄일 수 있습니다. 근사 컴퓨팅은 절대적으로 필요한 것과 실제 생산 결과물 사이의 차이를 이용할 수 있는 기회를 찾는 방법입니다. 예를 들어, 데이터를 정해진 데이터 구조에 배치하면 SQL에서 ORDER BY 연산자를 삭제하여 불필요한 처리를 제거하고 리소스를 절감하면서도 적절한 답변을 제공할 수 있습니다.

응답 시간 조정: 다소 느린 응답 시간으로 답변하면 다소 늦추면 공유 오버헤드를 최소화하여 탄소 배출량을 줄일 수 있습니다. 단기적인 임시 태스크를 처리할 경우 시작 오버헤드가 발생할 수 있습니다. 태스크가 도착할 때마다 오버헤드에 비용을 지불하는 대신 태스크를 그룹화하고 배치 처리합니다. 배치 처리는 응답 시간을 늘리는 대신 인스턴스 시작, 소스 코드 다운로드 및 프로세스 실행의 공유 오버헤드를 줄입니다.

가용성 조정: AWS를 사용하면 클릭 몇 번으로 이중화 구성을 추가하고 고가용성 목표를 충족할 수 있습니다. 언제나 활용률이 절감되는 유휴 리소스를 프로비저닝하여 정적 안정성과 같은 기법으로 중복성을 높일 수 있습니다. 목표를 설정할 때는 비즈니스의 요구를 평가합니다. 가용성을 조금 절충하면 활용도에서 훨씬 더 큰 개선을 얻을 수 있습니다. 예를 들어 정적 안정성 아키텍처 패턴에서는 구성 요소 장애 직후에 로드 처리를 담당할 유휴 장애 조치 용량을 프로비저닝합니다. 가용성 요구 사항을 완화하면 자동화를 통해 대체 리소스를 배포할 시간이 있으므로 유휴 온라인 용량이 필요 없습니다. 온디맨드로 장애 조치 용량을 추가하면 정상 운영 중에는 비즈니스에 영향을 미치지 않고 전반적인 사용률이 높아지며 비용 절감이라는 부차적인 이점도 얻을 수 있습니다.

클라우드의 지속 가능성을 위한 모범 사례

워크로드 배치를 최적화하고 수요, 소프트웨어, 데이터, 하드웨어 및 프로세스에 맞게 아키텍처를 최적화하여 에너지 효율성을 높입니다. 이러한 각 영역은 클라우드 워크로드의 지속 가능성에 미치는 영향을 줄여주는 모범 사례를 도입할 수 있는 기회 영역을 나타냅니다. 여기에는 활용률을 극대화하고 낭비를 최소화하며, 워크로드를 지원하기 위해 배포 및 구동되는 총 리소스를 최소화하는 방법 등이 포함됩니다.

주제

- 리전 선택
- 수요에 맞춘 조정
- 소프트웨어 및 아키텍처
- 데이터 관리
- 하드웨어 및 서비스
- 프로세스 및 문화

리전 선택

워크로드 리전의 선택은 성능, 비용 및 탄소 배출량을 포함한 KPI에 큰 영향을 미칩니다. 이러한 KPI를 효과적으로 개선하려면 비즈니스 요구 사항과 지속 가능성 목표를 기준으로 워크로드의 리전을 선택해야 합니다.

모범 사례

- SUS01-BP01 비즈니스 요구 사항과 지속 가능성 목표를 기준으로 리전 선택

SUS01-BP01 비즈니스 요구 사항과 지속 가능성 목표를 기준으로 리전 선택

비즈니스 요구 사항과 지속 가능성 목표를 기준으로 워크로드의 리전을 선택하여 성능, 비용 및 탄소 배출량을 비롯한 KPI를 최적화할 수 있습니다.

일반적인 안티 패턴:

- 자신의 고유한 위치를 기준으로 워크로드의 리전을 선택합니다.
- 모든 워크로드 리소스를 하나의 지리적 위치로 통합합니다.

이 모범 사례 수립의 이점: Amazon 재생 에너지 프로젝트와 가까운 곳이나 탄소 집약도가 낮은 리전에 워크로드를 배치하면 클라우드 워크로드의 탄소 배출량을 줄일 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

글로벌 네트워크 인프라가 함께 연결됨에 따라 AWS 클라우드는 리전 네트워크와 접속 지점(PoP)을 지속적으로 확장하고 있습니다. 워크로드 리전의 선택은 성능, 비용 및 탄소 배출량을 포함한 KPI에 큰 영향을 미칩니다. 이러한 KPI를 효과적으로 개선하려면 비즈니스 요구 사항과 지속 가능성 목표를 기준으로 워크로드의 리전을 선택해야 합니다.

구현 단계

- 규정 준수, 사용 가능한 기능, 비용 및 지연 시간을 비롯한 비즈니스 요구 사항을 기준으로 다음 단계를 따라 워크로드의 잠재적 리전을 평가하고 후보 명단에 추가할 수 있습니다.
  - 필수 지역 규정에 따라 해당 리전이 이를 준수하는지 확인합니다.
  - AWS 리전 서비스 목록을 사용하여 해당 리전에 워크로드를 실행하는 데 필요한 서비스와 기능이 있는지 확인합니다.
  - AWS Pricing Calculator를 사용하여 각 리전의 워크로드 비용을 계산합니다.
  - 최종 사용자 위치와 각 AWS 리전 사이의 네트워크 지연 시간을 테스트합니다.
- Amazon 재생 에너지 프로젝트 근처의 리전 및 그리드의 탄소 집약도가 다른 위치(또는 리전)보다 낮은 리전을 선택합니다.
  - 관련 지속 가능성 지침을 파악하여 온실 가스 협약(시장 기반 및 위치 기반 방법)을 기준으로 매년 탄소 배출량을 추적 및 비교합니다.
  - 탄소 배출량을 추적하는 데 사용하는 방법을 기준으로 리전을 선택합니다. 지속 가능성 지침에 따른 리전 선택에 대한 자세한 내용은 지속 가능성 목표를 기준으로 워크로드의 리전을 선택하는 방법을 참조하세요.

리소스

관련 문서:

- 탄소 배출량 추정치의 이해
- 세계 속의 Amazon
- 재생 에너지 방법론

- 워크로드의 리전을 선택할 때 고려해야 할 사항

관련 동영상:

- AWS re:Invent 2023 - Sustainability innovation in AWS Global Infrastructure
- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- AWS re:Invent 2022 - 지속 가능한 고성능 아키텍처 제공
- AWS re:Invent 2022 - 지속 가능성 설계와 AWS 탄소 배출량 절감
- AWS re:Invent 2022 - Sustainability in AWS global infrastructure

수요에 맞춘 조정

사용자 및 애플리케이션이 워크로드 및 기타 리소스를 사용하는 방식을 통해 지속 가능성 목표를 달성하기 위한 개선 사항을 식별할 수 있습니다. 인프라를 지속적으로 확장하여 수요를 충족하고 사용자를 지원하는 데 필요한 최소 리소스만 활용하는지 확인합니다. 고객 요구 사항에 맞게 서비스 수준을 조정합니다. 사용자가 리소스를 소비하는 데 필요한 네트워크를 제한하도록 리소스를 배치합니다. 사용되지 않는 자산을 제거합니다. 팀원에게 지속 가능성에 미치는 영향을 최소화하면서 요구 사항을 지원하는 디바이스를 제공합니다.

모범 사례

- SUS02-BP01 워크로드 인프라를 동적으로 확장
- SUS02-BP02 SLA를 지속 가능성 목표에 맞게 조정
- SUS02-BP03 미사용 자산의 생성 및 유지 관리 중지
- SUS02-BP04 네트워킹 요구 사항에 따라 워크로드의 지리적 배치 최적화
- SUS02-BP05 수행된 활동에 대한 팀원 리소스 최적화
- SUS02-BP06 버퍼링 또는 제한 개선으로 수요 곡선 완화

SUS02-BP01 워크로드 인프라를 동적으로 확장

클라우드의 탄력성을 활용하고 인프라를 동적으로 확장하여, 클라우드 리소스 공급을 수요에 맞게 조정하고 워크로드의 용량 초과 프로비저닝을 방지할 수 있습니다.

일반적인 안티 패턴:

- 사용자 로드에 따라 인프라를 확장하지 않습니다.

- 항상 인프라를 수동으로 확장합니다.
- 조정 이벤트 후에 다시 축소하는 대신 증가된 용량을 그대로 둡니다.

이 모범 사례 실천의 이점: 워크로드 탄력성을 구성하고 테스트하면 클라우드 리소스 공급을 수요에 효율적으로 일치시키고 용량 초과 프로비저닝을 방지할 수 있습니다. 클라우드의 탄력성을 활용하여 수요가 급증하는 도중과 그 이후에 용량을 자동으로 확장하여 비즈니스 요구 사항을 충족하는 데 필요한 리소스만큼만 사용할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

클라우드는 수요 변화에 맞춰 다양한 메커니즘을 통해 리소스를 동적으로 확장 또는 축소할 수 있는 유연성을 제공합니다. 공급과 수요를 최적으로 일치시키면 워크로드 환경에 미치는 영향이 최소화됩니다.

수요는 고정되거나 가변적일 수 있으므로 관리에 부담이 되지 않도록 측정 기준과 자동화가 필요합니다. 애플리케이션은 인스턴스 크기를 수정하여 수직(위로 또는 아래로)으로 확장하거나 인스턴스 수를 수정하여 수평(안 또는 밖으로)으로 확장하거나 둘을 모두 조합하여 확장할 수 있습니다.

다양한 접근 방식을 사용하여 리소스 공급과 수요를 일치시킬 수 있습니다.

- 타겟 추적 접근법: 스케일링 지표를 모니터링하고 필요에 따라 용량을 자동으로 늘리거나 줄입니다.
- 예측 확장: 일일 및 주간 추세를 예상하여 확장할 수 있습니다.
- 일정 기반 접근법: 예측 가능한 부하 변화에 따라 직접 스케일링 일정을 설정합니다.
- 서비스 스케일링: 기본적으로 설계별로 확장되는 서버리스와 같은 서비스를 선택하거나 자동 스케일링 기능을 제공합니다.

활용률이 낮거나 없는 기간을 식별하고 리소스의 크기를 조정하여 초과 용량을 제거하고 효율성을 개선합니다.

구현 단계

- 탄력성은 보유한 리소스의 공급을 해당 리소스의 수요에 맞춥니다. 인스턴스, 컨테이너 및 함수는 자동 스케일링과 함께, 또는 서비스의 기능을 사용하여 탄력성을 지원하는 메커니즘을 제공합니다. AWS는 사용자 로드가 적은 기간에 워크로드를 빠르고 쉽게 축소할 수 있도록 다양한 자동 스케일링 메커니즘을 제공합니다. 자동 스케일링 메커니즘의 예시는 다음과 같습니다.

Auto scaling mechanism	Where to use
Amazon EC2 Auto Scaling	애플리케이션의 사용자 로드를 처리하는 데 사용할 수 있는 적절한 수의 Amazon EC2 인스턴스가 있는지 확인하는 데 사용합니다.
Application Auto Scaling	Amazon EC2를 벗어난 개별 AWS 서비스(예: Lambda 기능 또는 Amazon Elastic Container Service (Amazon ECS) 서비스)에 대한 리소스를 자동으로 확장하는 데 사용합니다.
Kubernetes Cluster Autoscaler	AWS에서 Kubernetes 클러스터를 자동으로 확장하는 데 사용합니다.

- 확장은 대개 Amazon EC2 인스턴스나 AWS Lambda 함수 등의 컴퓨팅 서비스와 관련하여 설명하는 경우가 많습니다. Amazon DynamoDB 읽기/쓰기 용량 단위나 Amazon Kinesis Data Streams 샤드와 같은 컴퓨팅 외의 서비스를 구성할 때는 수요에 일치시키는 것이 좋습니다.
- 스케일 업 또는 스케일 다운 대한 지표가 배포 중인 워크로드 유형에 대해 검증되었는지 확인합니다. 동영상 트랜스코딩 애플리케이션을 배포하는 경우 100%의 CPU 활용률이 예상되므로, 기본 지표로 사용해서는 안 됩니다. 필요한 경우 맞춤형 지표(예: 메모리 사용률)을 스케일링 정책에 사용할 수 있습니다. 올바른 지표를 선택하려면 Amazon EC2에 대한 다음 지침을 고려하세요.
  - 지표는 유효한 사용률 지표여야 하며 인스턴스가 얼마나 많이 사용되는지를 설명해야 합니다.
  - 지표 값은 Auto Scaling 그룹 내 인스턴스 수에 비례하여 늘거나 줄어야 합니다.
- Auto Scaling 그룹에는 수동 스케일링 대신 동적 스케일링을 사용합니다. 또한, 동적 스케일링에는 타겟 추적 스케일링 정책을 사용할 것을 권합니다.
- 워크로드 배포에서 확장 및 축소 이벤트를 모두 처리할 수 있는지 확인합니다. 축소 이벤트에 대한 테스트 시나리오를 생성하여 워크로드가 예상대로 작동하고 사용자 환경에 영향(예: 스티키 세션 손실)을 미치지 않는지 확인합니다. 활동 기록을 사용하여 Auto Scaling 그룹의 스케일링 활동을 확인할 수 있습니다.
- 워크로드의 예측 가능한 패턴을 평가하고 예측 및 계획된 수요 변화에 따라 사전 예방적으로 확장합니다. 예측 스케일링에서는 용량을 과도하게 프로비저닝할 필요가 없습니다. 자세한 내용은 Amazon EC2 Auto Scaling을 사용한 예측 스케일링을 참조하세요.

리소스

관련 문서:

- Amazon EC2 Auto Scaling 시작하기
- Predictive Scaling for EC2, Powered by Machine Learning
- Analyze user behavior using Amazon OpenSearch Service, Amazon Data Firehose and Kibana
- Amazon CloudWatch란 무엇인가요?
- Amazon RDS의 성능 개선 도우미로 DB 로드 모니터링
- Introducing Native Support for Predictive Scaling with Amazon EC2 Auto Scaling
- Introducing Karpenter - An Open-Source, High-Performance Kubernetes Cluster Autoscaler
- 심층 분석: Amazon ECS 클러스터 Auto Scaling

관련 동영상:

- AWS re:Invent 2023 - Scaling on AWS for the first 10 million users
- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- AWS re:Invent 2022 - Build a cost-, energy-, and resource-efficient compute environment
- AWS re:Invent 2022 - Scaling containers from one user to millions
- AWS re:Invent 2023 - Scaling FM inference to hundreds of models with Amazon SageMaker
- AWS re:Invent 2023 - Harness the power of Karpenter to scale, optimize & upgrade Kubernetes

관련 예시:

- Autoscaling

SUS02-BP02 SLA를 지속 가능성 목표에 맞게 조정

지속 가능성 목표를 기준으로 서비스 수준에 관한 계약(SLA)을 검토 및 최적화하여 계속해서 비즈니스 필요를 충족하면서 워크로드를 지원하는 데 필요한 리소스를 최소화합니다.

일반적인 안티 패턴:

- 워크로드 SLA가 알려져 있지 않거나 모호합니다.

- 가용성 및 성능에 대해서만 SLA를 정의합니다.
- 모든 워크로드에 대해 동일한 설계 패턴(예: 다중 AZ 아키텍처)을 사용합니다.

이 모범 사례 확립의 이점: 지속 가능성 목표에 맞춰 SLA를 조정하면 비즈니스 요구 사항을 충족하면서 리소스 사용을 최적화할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출되는 위험 수준: 낮음

구현 가이드

SLA는 클라우드 워크로드에서 예상되는 서비스 수준(예: 응답 시간, 가용성, 데이터 보존 등)을 정의합니다. SLA는 클라우드 워크로드의 아키텍처, 리소스 사용 및 환경 영향에 영향을 미칩니다. 주기적으로 SLA를 검토하여 허용 가능한 수준으로 서비스를 줄여 리소스 사용을 크게 줄이는 절충안을 제시합니다.

구현 단계

- 지속 가능성 목표 이해: 탄소 배출 감소 또는 리소스 활용 개선과 같은 조직의 지속 가능성 목표를 식별합니다.
- SLA 검토: SLA가 비즈니스 요구 사항을 지원하는지 평가합니다. SLA를 초과 충족하는 경우 추가 검토를 수행합니다.
- 절충점 이해: 워크로드의 복잡성(예: 대량의 동시 사용자), 성능(예: 지연 시간), 지속 가능성에 미치는 영향(예: 필요한 리소스) 전반의 절충점을 이해합니다. 일반적으로 세 번째 요소 대신 첫 두 요소가 우선 고려됩니다.
- SLA 조정: SLA를 조정하여 허용 가능한 수준으로 서비스를 줄임으로써 지속 가능성에 미치는 영향을 크게 줄이는 절충점을 제시합니다.
  - 지속 가능성 및 신뢰성: 가용성이 뛰어난 워크로드는 리소스를 더 많이 사용하는 경향이 있습니다.
  - 지속 가능성 및 성능: 성능을 높이기 위해 리소스를 더 많이 사용하면 환경에 미치는 영향이 커질 수 있습니다.
  - 지속 가능성 및 보안: 지나치게 안전한 워크로드는 환경에 미치는 영향이 커질 수 있습니다.
- 가능한 경우 지속 가능성 SLA 정의: 워크로드에 지속 가능성 SLA를 포함합니다. 예를 들어 최소 사용률 수준을 컴퓨팅 인스턴스의 지속 가능성 SLA로 정의합니다.
- 효율적인 설계 패턴 사용: 비즈니스에 중요한 기능에 우선순위를 두고 중요하지 않은 기능에 대해 더 낮은 서비스 수준(예: 응답 시간 또는 복구 시간 목표)을 허용하는 설계 패턴(예: AWS의 마이크로서비스)을 사용합니다.

- 의사소통 및 책임 확립: 개발 팀 및 고객을 포함한 모든 관련 이해관계자와 SLA를 공유합니다. 보고 기능을 사용하여 SLA를 추적하고 모니터링합니다. SLA의 지속 가능성 목표를 달성하기 위한 책임을 할당합니다.
- 인센티브 및 보상 사용: 인센티브와 보상을 사용하여 지속 가능성 목표에 맞는 SLA를 달성하거나 초과 달성합니다.
- 검토 및 반복: SLA가 진화하는 지속 가능성 및 성능 목표에 부합하는지 정기적으로 검토하고 조정합니다.

리소스

관련 문서:

- 복원력 패턴과 절충점을 이해하여 클라우드를 효율적으로 설계하세요.
- SaaS 공급자의 서비스 수준 계약 중요성

관련 동영상:

- AWS re:Invent 2023 - 용량, 가용성, 비용 효율성: 세 가지 선택
- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- AWS re:Invent 2023 - 느슨하게 연결된 시스템을 위한 고급 통합 패턴 및 장단점
- AWS re:Invent 2022 - 지속 가능한 고성능 아키텍처 제공
- AWS re:Invent 2022 - Build a cost-, energy-, and resource-efficient compute environment

SUS02-BP03 미사용 자산의 생성 및 유지 관리 중지

워크로드에서 사용되지 않는 자산을 폐기하여 수요를 지원하는 데 필요한 클라우드 리소스 수를 줄이고 낭비를 최소화합니다.

일반적인 안티 패턴:

- 중복되거나 더 이상 필요하지 않은 자산에 대해 애플리케이션을 분석하지 않습니다.
- 중복되거나 더 이상 필요하지 않은 자산을 제거하지 않습니다.

이 모범 사례 확립의 이점: 사용되지 않는 자산을 제거하면 리소스가 확보되고 워크로드의 전체 효율성이 향상됩니다.

이 모범 사례를 따르지 않을 경우 노출되는 위험 수준: 낮음

구현 가이드

사용되지 않는 자산은 스토리지 공간 및 컴퓨팅 전력과 같은 클라우드 리소스를 소비합니다. 이러한 자산을 식별하고 제거함으로써 리소스를 확보하여 클라우드 아키텍처의 효율성을 높일 수 있습니다. 애플리케이션 자산(예: 사전 컴파일된 보고서, 데이터 세트, 정적 이미지 등)과 자산 액세스 패턴을 정기적으로 분석하여 중복된 자산, 활용률이 낮은 자산 및 잠재적 폐기 대상을 식별합니다. 이러한 중복 자산을 제거하여 워크로드의 리소스 낭비를 줄이세요.

구현 단계

- 인벤토리 구성: 포괄적인 인벤토리를 구성하여 워크로드 내의 모든 자산을 식별합니다.
- 사용량 분석: 지속적인 모니터링을 통해 더 이상 필요하지 않은 정적 자산을 식별합니다.
- 미사용 자산 제거: 더 이상 필요하지 않은 자산을 제거할 계획을 세웁니다.
  - 자산을 제거하기 전에 제거로 인해 아키텍처가 받는 영향을 평가합니다.
  - 중복 생성 자산을 통합하여 중복 처리를 제거합니다.
  - 애플리케이션을 업데이트하여 더 이상 필요 없는 자산을 생성하고 저장하지 않습니다.
- 제3자와 소통: 더 이상 필요하지 않은 관리 자산의 생산 및 저장을 중단하도록 제3자에게 지시합니다. 중복 자산 통합을 요청합니다.
- 수명 주기 정책 사용: 수명 주기 정책을 사용하여 사용하지 않는 자산을 자동으로 삭제합니다.
  - Amazon S3 수명 주기를 사용하여 전체 수명 주기 동안 객체를 관리할 수 있습니다.
  - Amazon Data Lifecycle Manager를 사용하여 Amazon EBS 스냅샷 및 Amazon EBS 지원 AMI의 생성, 보존 및 삭제를 자동화할 수 있습니다.
- 검토 및 최적화: 워크로드를 정기적으로 검토하여 사용하지 않는 자산을 식별하고 제거합니다.

리소스

관련 문서:

- 지속 가능성을 위한 AWS 인프라 최적화, 파트 II: 스토리지
- AWS 계정에서 더 이상 필요하지 않은 활성 리소스를 종료하려면 어떻게 해야 하나요?

관련 동영상:

- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future

- AWS re:Invent 2022 - Preserving and maximizing the value of digital media assets using Amazon S3
- AWS re:Invent 2023 - 다중 계정 환경의 비용 최적화

SUS02-BP04 네트워킹 요구 사항에 따라 워크로드의 지리적 배치 최적화

네트워크 트래픽이 이동해야 하는 거리를 단축하고 워크로드를 지원하는 데 필요한 총 네트워크 리소스를 줄일 수 있는 워크로드의 클라우드 위치 및 서비스를 선택합니다.

일반적인 안티 패턴:

- 자신의 고유한 위치를 기준으로 워크로드의 리전을 선택합니다.
- 모든 워크로드 리소스를 하나의 지리적 위치로 통합합니다.
- 모든 트래픽이 기존 데이터 센터를 통과합니다.

이 모범 사례 확립의 이점: 사용자에게 가깝게 워크로드를 배치하면 지연 시간을 최대한 단축할 수 있고 동시에 네트워크 간 데이터 이동을 줄이면서 환경에 미치는 영향까지 줄일 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

AWS Outposts, AWS 로컬 영역 등 AWS 클라우드 인프라는 리전, 가용 영역, 배치 그룹, 엣지 로케이션과 같은 위치 옵션을 중심으로 구축됩니다. 이러한 위치 옵션은 애플리케이션 구성 요소, 클라우드 서비스, 엣지 네트워크, 온프레미스 데이터 센터 간의 연결을 유지합니다.

워크로드의 네트워크 액세스 패턴을 분석하여 이러한 클라우드 위치 옵션을 사용하는 방법을 식별하고 네트워크 트래픽이 이동해야 하는 거리를 줄이십시오.

구현 단계

- 워크로드의 네트워크 액세스 패턴을 분석하여 사용자가 애플리케이션을 사용하는 방법을 식별합니다.
  - Amazon CloudWatch 및 AWS CloudTrail과 같은 모니터링 도구를 사용하여 네트워크 활동에 대한 데이터를 수집합니다.
  - 데이터를 분석하여 네트워크 액세스 패턴을 식별합니다.
- 다음과 같은 주요 요소를 토대로 하여 워크로드 배포용 리전을 선택합니다.

  - 지속 가능성 목표: 리전 선택에 설명되어 있습니다.
  - 데이터 위치: 데이터를 많이 사용하는 애플리케이션의 경우(예: 빅 데이터 및 기계 학습) 애플리케이션 코드는 최대한 데이터와 가까운 위치에서 실행되어야 합니다.
  - 사용자의 위치: 사용자가 직접 이용하는 애플리케이션의 경우 워크로드 사용자와 가까운 리전을 하나 또는 여러 개 선택합니다.
  - 기타 제약 조건: 워크로드의 리전을 선택할 때 고려해야 할 사항에 나와 있는 비용 및 규정 준수 등 제약 요건을 고려합니다.
- 자주 이용하는 자산에 로컬 캐싱 또는 AWS 캐싱 솔루션을 사용하여 성능을 개선하고, 데이터 이동을 줄이며, 환경에 미치는 영향을 줄입니다.

Service	When to use
Amazon CloudFront	이미지, 스크립트, 동영상 등의 정적 콘텐츠와 API 응답 또는 웹 애플리케이션 등의 동적 콘텐츠를 캐시하는 데 사용합니다.
Amazon ElastiCache	웹 애플리케이션의 콘텐츠를 캐시하는 데 사용합니다.
DynamoDB Accelerator	DynamoDB 테이블에 인 메모리 가속화를 추가하는 데 사용합니다.

- 워크로드 사용자에게 더 가까운 위치에서 코드를 실행할 수 있는 서비스를 사용합니다.

Service	When to use
Lambda@Edge	객체가 캐시에 없는 경우 시작되는 컴퓨팅 집약적 작업에 사용합니다.
Amazon CloudFront 함수	HTTP(s) 요청 또는 응답 조작 등과 같이 단기 실행 함수에 의해 시작될 수 있는 간단한 사용 사례에 사용합니다.
AWS IoT Greengrass	커넥티드 디바이스를 위한 로컬 컴퓨팅, 메시징 및 데이터 캐시를 실행하는 데 사용합니다.

- 연결 풀을 사용하여 연결을 재사용하고 필요한 리소스를 줄입니다.

- 지속적 연결 및 동기식 업데이트에 의존하지 않는 분산 데이터 스토어를 사용하여 리전별 사용자 집단을 일관되게 지원합니다.
- 사전 프로비저닝된 정적 네트워크 용량을 공유 동적 용량으로 교체하고 네트워크 용량의 지속 가능성에 미치는 영향을 다른 구독자와 공유합니다.

리소스

관련 문서:

- 지속 가능성을 위한 AWS 인프라 최적화, 파트 III: 네트워킹
- Amazon ElastiCache 설명서
- Amazon CloudFront란 무엇인가요?
- Amazon CloudFront 주요 기능
- AWS 글로벌 인프라
- AWS 로컬 영역 및 AWS Outposts, 엣지 워크로드에 적합한 기술 선택
- 배치 그룹
- AWS 로컬 영역
- AWS Outposts

관련 동영상:

- Demystifying data transfer on AWS
- 차세대 Amazon EC2 인스턴스의 네트워크 성능 확장
- AWS 로컬 영역 설명 비디오
- AWS Outposts: Overview and How it Works
- AWS re:Invent 2023 - A migration strategy for edge and on-premises workloads
- AWS re:Invent 2021 - AWS Outposts: Bringing the AWS experience on premises
- AWS re:Invent 2020 - AWS Wavelength: Run apps with ultra-low latency at 5G edge
- AWS re:Invent 2022 - AWS Local Zones: Building applications for a distributed edge
- AWS re:Invent 2021 - Building low-latency websites with Amazon CloudFront
- AWS re:Invent 2022 - Improve performance and availability with AWS Global Accelerator
- AWS re:Invent 2022 - Build your global wide area network using AWS

- AWS re:Invent 2020: Global traffic management with Amazon Route 53

관련 예시:

- AWS 네트워킹 워크숍
- 지속 가능성을 위한 설계 - 네트워크 간 데이터 이동 최소화

SUS02-BP05 수행된 활동에 대한 팀원 리소스 최적화

팀원에게 제공되는 리소스를 최적화하여 팀원에게 필요한 지원을 충분히 제공하면서도 환경 지속 가능성에 미치는 영향을 최소화합니다.

일반적인 안티 패턴:

- 팀원이 사용하는 디바이스가 클라우드 애플리케이션의 전반적인 효율성에 미치는 영향을 무시합니다.
- 팀원이 사용하는 리소스를 수동으로 관리하고 업데이트합니다.

이 모범 사례 확립의 이점: 팀원 리소스를 최적화하면 클라우드 지원 애플리케이션의 전반적인 효율성이 향상됩니다.

이 모범 사례를 따르지 않을 경우 노출되는 위험 수준: 낮음

구현 가이드

팀원이 서비스를 사용하는 데 활용하는 리소스, 예상 수명 주기, 재무 및 지속 가능성에 미치는 영향을 이해합니다. 이러한 리소스를 최적화하기 위한 전략을 구현합니다. 예를 들어, 활용률이 낮은 고성능 단일 사용자 시스템 대신 활용률이 높은 확장 가능한 인프라에서 렌더링 및 컴파일과 같은 복잡한 작업을 수행합니다.

구현 단계

- 에너지 효율적인 워크스테이션 사용: 팀원에게 에너지 효율적인 워크스테이션과 주변 장치를 제공합니다. 이러한 디바이스에서 효율적인 전력 관리 기능(예: 저전력 모드)을 사용하여 에너지 사용량을 줄입니다.
- 가상화 사용: 가상 데스크톱 및 애플리케이션 스트리밍을 사용하여 업그레이드 및 디바이스 요구 사항을 제한합니다.

- 원격 협업 장려: 팀원이 출장의 필요성과 이와 관련된 탄소 배출량을 줄일 수 있도록 Amazon Chime 또는 AWS Wickr와 같은 원격 협업 도구를 사용하도록 권장합니다.
- 에너지 효율적인 소프트웨어 사용: 불필요한 기능 및 프로세스를 제거하거나 비활성화하여 팀원에게 에너지 효율적인 소프트웨어를 제공합니다.
- 수명 주기 관리: 프로세스 및 시스템이 디바이스 수명 주기에 미치는 영향을 평가하고 비즈니스 요구 사항을 충족하면서 디바이스 교체 요구 사항을 최소화하는 솔루션을 선택합니다. 워크스테이션이나 소프트웨어를 정기적으로 유지 관리하고 업데이트하여 효율성을 유지하고 개선합니다.
- 원격 디바이스 관리: 디바이스에 대한 원격 관리를 구현하여 출장의 필요성을 줄입니다.
  - AWS Systems Manager Fleet Manager는 AWS 또는 온프레미스에서 실행 중인 노드를 원격으로 관리하는 데 도움이 되는 통합 사용자 인터페이스(UI) 환경입니다.

리소스

관련 문서:

- Amazon WorkSpaces란 무엇인가요?
- Amazon WorkSpaces용 Cost Optimizer
- Amazon AppStream 2.0 설명서
- NICE DCV

관련 동영상:

- Managing cost for Amazon WorkSpaces on AWS

SUS02-BP06 버퍼링 또는 제한 개선으로 수요 곡선 완화

버퍼링 및 제한은 수요 곡선을 완화하고 워크로드에 필요한 프로비저닝 용량을 줄입니다.

일반적인 안티 패턴:

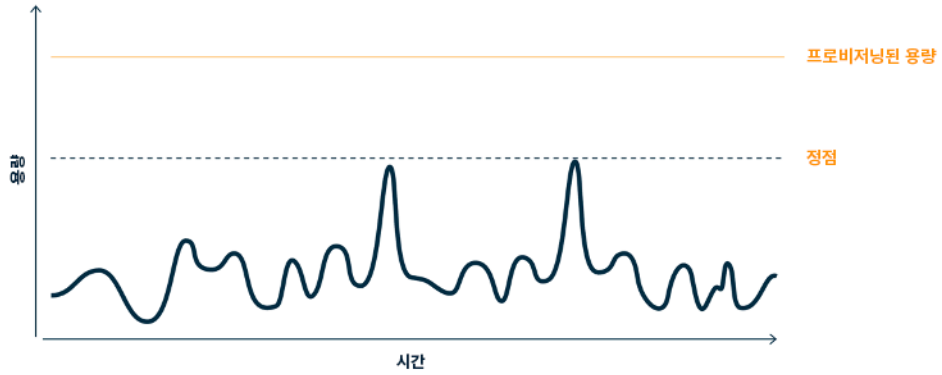
- 클라이언트 요청은 필요하지 않아도 즉시 처리합니다.
- 클라이언트 요청에 대한 요구 사항을 분석하지 않습니다.

이 모범 사례 확립의 이점: 수요 곡선을 완화하면 워크로드에 필요한 프로비저닝 용량이 줄어듭니다. 프로비저닝 용량을 줄이면 에너지 소비와 환경에 미치는 영향도 줄어듭니다.

이 모범 사례를 따르지 않을 경우 노출되는 위험 수준: 낮음

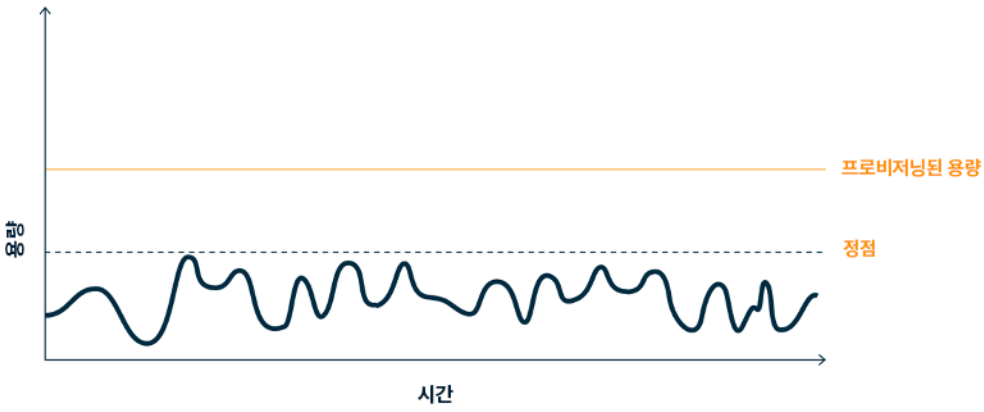
구현 가이드

워크로드 수요 곡선을 완화하면 워크로드에 프로비저닝된 용량을 줄이고 환경에 미치는 영향도 줄일 수 있습니다. 아래 그림에 표시된 수요 곡선을 바탕으로 워크로드를 가정합니다. 이 워크로드에는 2개의 정점이 있으며, 이러한 정점을 처리하기 위해 주황색 선으로 표시된 리소스 용량이 프로비저닝됩니다. 이 워크로드에 사용되는 리소스와 에너지는 수요 곡선 아래의 영역이 아니라 프로비저닝된 용량 선 아래의 영역으로 표시됩니다. 이 2가지 정점을 처리하려면 프로비저닝된 용량이 필요하기 때문입니다.



높은 프로비저닝 용량이 필요한 2개의 고유한 정점이 존재하는 수요 곡선입니다.

버퍼링 또는 제한을 사용하여 수요 곡선을 수정하고 정점을 완화할 수 있습니다. 이렇게 되면 프로비저닝된 용량과 소비되는 에너지가 줄어듭니다. 클라이언트가 재시도를 수행할 때 제한을 구현합니다. 요청을 저장하고 나중에 처리하도록 버퍼링을 구현합니다.



수요 곡선 및 프로비저닝된 용량에 대한 제한의 효과입니다.

구현 단계

- 클라이언트 요청을 분석하여 응답 방법을 결정합니다. 고려해야 할 질문은 다음과 같습니다.
  - 이 요청을 비동기식으로 처리할 수 있는가?
  - 클라이언트에 재시도 기능이 있는가?
- 클라이언트에 재시도 기능이 있는 경우 현재 요청을 처리할 수 없으면 나중에 다시 시도해야 함을 소스에 알려주는 제한 기능을 구현할 수 있습니다.
  - Amazon API Gateway를 사용하여 제한을 구현할 수 있습니다.
- 재시도를 수행할 수 없는 클라이언트의 경우 수요 곡선을 완화하려면 버퍼를 구현해야 합니다. 버퍼는 서로 다른 속도로 실행되는 애플리케이션이 효과적으로 통신할 수 있도록 요청 처리를 연기합니다. 버퍼 기반 접근 방식은 대기열 또는 스트림을 사용하여 생산자의 메시지를 수락합니다. 메시지는 소비자가 읽은 후 처리되므로 소비자의 비즈니스 요구 사항을 충족하는 속도로 메시지를 실행할 수 있습니다.
  - Amazon Simple Queue Service(Amazon SQS)는 단일 소비자가 개별 메시지를 읽을 수 있는 대기열을 제공하는 관리형 서비스입니다.
  - Amazon Kinesis에서는 여러 소비자가 같은 메시지를 읽을 수 있는 스트림을 제공합니다.
- 전체 수요, 변경률 및 필수 응답 시간을 분석하여 필요한 제한 또는 버퍼의 크기를 적절하게 조정합니다.

리소스

관련 문서:

- Amazon SQS 시작하기
- Application integration Using Queues and Messages
- 워크로드에서 API 제한 관리 및 모니터링
- Throttling a tiered, multi-tenant REST API at scale using API Gateway
- Application integration Using Queues and Messages

관련 동영상:

- AWS re:Invent 2022 - Application integration patterns for microservices
- AWS re:Invent 2023 - 스마트한 절약: Amazon EC2 비용 최적화 전략
- AWS re:Invent 2023 - 느슨하게 연결된 시스템을 위한 고급 통합 패턴 및 장단점

소프트웨어 및 아키텍처

로드 평준화를 수행하고 배포된 리소스의 높은 활용률을 일관되게 유지하여 소비되는 리소스를 최소화하기 위한 패턴을 구현합니다. 구성 요소는 시간 경과에 따른 사용자 행동의 변화로 인해 사용 부족으로 인해 유휴 상태가 될 수 있습니다. 패턴과 아키텍처를 수정하여 활용률이 낮은 구성 요소를 통합함으로써 전체 활용률을 높입니다. 더 이상 필요하지 않은 구성 요소를 폐기합니다. 워크로드 구성 요소의 성능을 이해하고 리소스를 가장 많이 사용하는 구성 요소를 최적화합니다. 고객이 서비스에 액세스하고 패턴을 구현하는 데 사용하는 디바이스를 숙지하여 디바이스 업그레이드 필요성을 최소화합니다.

모범 사례

- SUS03-BP01 비동기식 및 예약된 작업을 위한 소프트웨어 및 아키텍처 최적화
- SUS03-BP02 사용 빈도가 낮거나 전혀 없는 워크로드 구성 요소 제거 또는 리팩터링
- SUS03-BP03 가장 많은 시간 또는 리소스를 소모하는 코드 영역 최적화
- SUS03-BP04 디바이스 및 장비에 대한 영향 최적화
- SUS03-BP05 데이터 액세스 및 스토리지 패턴을 가장 잘 지원하는 소프트웨어 패턴 및 아키텍처 사용

SUS03-BP01 비동기식 및 예약된 작업을 위한 소프트웨어 및 아키텍처 최적화

배포된 리소스의 일관되고 높은 사용률을 유지할 수 있도록 대기열 기반과 같은 효율적인 소프트웨어 및 아키텍처 패턴을 사용합니다.

일반적인 안티 패턴:

- 클라우드 워크로드의 리소스를 과다하게 프로비저닝하여 예상치 못한 수요 급증이 발생합니다.
- 아키텍처가 메시징 구성 요소에 의한 비동기식 메시지의 발신자와 수신자를 분리하지 않습니다.

이 모범 사례 확립의 이점:

- 효율적인 소프트웨어 및 아키텍처 패턴이 워크로드의 사용되지 않는 리소스를 최소화하고 전체적인 효율성을 개선합니다.
- 비동기식 메시지 수신과 관계없이 처리를 확장할 수 있습니다.
- 메시징 구성 요소를 통해 가용성 요구 사항이 완화되어 더 적은 리소스로 이를 충족할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

이벤트 기반 아키텍처와 같은 효율적인 아키텍처 패턴을 사용하면 구성 요소의 사용률이 균등해지고 워크로드의 과다한 프로비저닝을 최소화할 수 있습니다. 효율적인 아키텍처 패턴을 사용하면 시간 경과에 따른 수요 변화로 인해 사용하지 않는 유휴 리소스를 최소화할 수 있습니다.

워크로드 구성 요소의 요구 사항을 이해하고 리소스의 전체 사용률을 높이는 아키텍처 패턴을 도입합니다. 더 이상 필요하지 않은 구성 요소를 폐기합니다.

구현 단계

- 워크로드에 대한 수요를 분석하여 이에 대한 대응 방법을 결정합니다.
- 동기식 응답이 필요하지 않은 요청이나 작업의 경우 대기열 기반 아키텍처 및 오토 스케일링 작업자를 사용하여 사용률을 극대화합니다. 대기열 기반 아키텍처를 고려해야 하는 몇 가지 예는 다음과 같습니다.

Queuing mechanism	Description
AWS Batch 작업 대기열	AWS Batch 작업은 컴퓨팅 환경에서 실행되도록 예약될 때까지 작업 대기열로 제출됩니다.
Amazon Simple Queue Service 및 Amazon EC2 스팟 인스턴스	Amazon SQS와 스팟 인스턴스를 페어링하여 내결함성이 있고 효율적인 아키텍처를 구축합니다.

- 언제든 처리할 수 있는 요청이나 작업의 경우 더 높은 효율을 위해 예약 메커니즘을 사용하여 작업을 일괄 처리합니다. AWS의 예약 메커니즘의 예는 다음과 같습니다.

Scheduling mechanism	Description
Amazon EventBridge 스케줄러	Amazon EventBridge의 기능을 통해 대규모로 예약된 작업을 생성, 실행 및 관리할 수 있습니다.
AWS Glue 시간 기반 일정	AWS Glue의 크롤러와 작업에 대한 시간 기반 일정을 정의합니다.

Scheduling mechanism	Description
Amazon Elastic Container Service (Amazon ECS) 예약된 작업	Amazon ECS는 예약된 작업 생성을 지원합니다. 예약된 작업은 Amazon EventBridge 규칙을 사용하여 예약에 따라 또는 EventBridge 이벤트에 대한 응답으로 작업을 실행합니다.
Instance Scheduler	Amazon EC2 및 Amazon Relational Database Service 인스턴스의 시작 및 종료 예약을 구성합니다.

- 아키텍처에서 폴링과 웹후크 메커니즘을 사용하는 경우 이를 이벤트로 바꿉니다. 이벤트 기반 아키텍처를 사용하여 고효율 워크로드를 구축합니다.
- AWS의 서버리스를 활용하여 과다하게 프로비저닝된 인프라를 제거합니다.
- 입력 대기 중인 유휴 리소스를 방지하기 위해 아키텍처의 개별 구성 요소의 적절한 크기를 지정합니다.
  - AWS Cost Explorer의 적정 크기 조정 권장 사항 또는 AWS Compute Optimizer를 사용하여 적정 크기 조정 기회를 식별할 수 있습니다.
  - 자세한 내용은 올바른 크기 조정: 워크로드에 맞게 인스턴스 프로비저닝을 참조하세요.

리소스

관련 문서:

- Amazon Simple Queue Service란 무엇인가요?
- Amazon MQ란 무엇인가요?
- Amazon SQS 기반 크기 조정
- AWS Step Functions란 무엇인가요?
- AWS Lambda란 무엇인가요?
- Amazon SQS에서 AWS Lambda 사용
- Amazon EventBridge란 무엇인가요?
- Managing Asynchronous Workflows with a REST API

관련 동영상:

- AWS re:Invent 2023 - Navigating the journey to serverless event-driven architecture
- AWS re:Invent 2023 - Using serverless for event-driven architecture & domain-driven design
- AWS re:Invent 2023 - Amazon EventBridge의 고급 이벤트 기반 패턴
- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- Asynchronous Message Patterns | AWS Events

관련 예시:

- Event-driven architecture with AWS Graviton Processors and Amazon EC2 Spot Instances

SUS03-BP02 사용 빈도가 낮거나 전혀 없는 워크로드 구성 요소 제거 또는 리팩터링

사용되지 않아 더 이상 필요하지 않은 구성 요소를 제거하고 활용률이 낮은 구성 요소를 리팩터링하여 워크로드에서 낭비되는 리소스를 최소화합니다.

일반적인 안티 패턴:

- 워크로드의 개별 구성 요소 사용률 수준을 정기적으로 확인하지 않습니다.
- AWS 적정 크기 조정 도구(예: AWS Compute Optimizer)에서 권장 사항을 확인하고 분석하지 않습니다.

이 모범 사례 확립의 이점: 사용되지 않는 구성 요소를 제거하면 낭비를 최소화하고 클라우드 워크로드의 전반적인 효율성을 향상할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

워크로드를 검토하여 유휴 상태이거나 사용하지 않는 구성 요소를 식별합니다. 이는 수요 변화 또는 새로운 클라우드 서비스 출시로 시작될 수 있는 반복적인 개선 프로세스입니다. 예를 들어, AWS Lambda 함수 실행 시간의 급격한 저하는 메모리 크기를 줄여야 하는 필요성을 나타내는 지표가 될 수 있습니다. 또한, AWS에서 새로운 서비스와 기능을 출시함에 따라 워크로드에 맞는 최적의 서비스와 아키텍처가 변경될 수 있습니다.

워크로드 활동을 지속적으로 모니터링하고 개별 구성 요소의 활용률 수준을 개선할 수 있는 기회를 찾아보세요. 유휴 상태인 구성 요소를 제거하고 적절한 크기 조정 작업을 수행하면 클라우드 리소스를 최소화하여 비즈니스 요구 사항을 충족할 수 있습니다.

구현 단계

- AWS 리소스의 인벤토리를 확보합니다. AWS에서는 AWS 리소스 탐색기를 켜서 AWS 리소스를 살펴보고 정리할 수 있습니다. 자세한 내용은 AWS re:Invent 2022 - How to manage resources and applications at scale on AWS를 참조하십시오.
- 워크로드의 중요한 구성 요소(예: Amazon CloudWatch 지표의 CPU 활용률, 메모리 활용률 또는 네트워크 처리량)에 대한 활용률 지표를 모니터링하고 캡처합니다.
- 아키텍처에서 사용되지 않거나 활용도가 낮은 구성 요소를 식별합니다.
  - 안정적인 워크로드를 위해 정기적으로 AWS 적정 크기 조정 도구(예: AWS Compute Optimizer)를 확인하여 유휴 상태이거나, 사용되지 않거나, 활용도가 낮은 구성 요소를 식별합니다.
  - 일시적인 워크로드의 경우 활용률 지표를 평가하여 유휴 상태이거나, 사용되지 않거나, 활용도가 낮은 구성 요소를 식별합니다.
- 더 이상 필요하지 않은 구성 요소 및 관련 자산(예: Amazon ECR 이미지)을 사용 중지합니다.
  - Amazon ECR에서 사용되지 않는 이미지 자동 정리
  - Delete unused Amazon Elastic Block Store (Amazon EBS) volumes by using AWS Config and AWS Systems Manager
- 사용률이 낮은 구성 요소를 리팩터링하거나 다른 리소스와 통합하여 사용 효율성을 개선합니다. 예를 들어, 사용률이 낮은 개별 인스턴스에서 데이터베이스를 실행하는 대신 단일 Amazon RDS 데이터베이스 인스턴스에 여러 개의 소규모 데이터베이스를 프로비저닝할 수 있습니다.
- 작업 단위를 완료하기 위해 워크로드에서 프로비저닝한 리소스를 파악합니다.

리소스

관련 문서:

- AWS Trusted Advisor
- Amazon CloudWatch란 무엇인가요?
- 올바른 크기 조정: 워크로드에 맞게 인스턴스 프로비저닝
- Optimizing your cost with Rightsizing Recommendations

관련 동영상:

- AWS re:Invent 2023 - 용량, 가용성, 비용 효율성: 세 가지 선택

관련 예시:

- 하드웨어 패턴 최적화 및 지속 가능성 KPI 관찰

SUS03-BP03 가장 많은 시간 또는 리소스를 소모하는 코드 영역 최적화

아키텍처의 여러 구성 요소 내에서 실행되는 코드를 최적화하여 리소스 사용을 최소화하고 성능을 극대화할 수 있습니다.

일반적인 안티 패턴:

- 리소스 사용에 대한 코드 최적화를 무시합니다.
- 일반적으로 리소스를 늘리는 방법으로 성능 문제에 대응합니다.
- 코드 검토 및 개발 프로세스에서 성능 변경을 추적하지 않습니다.

이 모범 사례 확립의 이점: 효율적인 코드를 활용하면 리소스 사용을 최소화하고 성능을 향상할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

리소스 사용 및 성능을 최적화하려면 클라우드 아키텍처 기반 애플리케이션의 코드를 포함한 모든 기능 영역을 검사해야 합니다. 빌드 환경 및 프로덕션에서 워크로드의 성능을 지속적으로 모니터링하고 리소스 사용량이 특히 높은 코드 스니펫을 개선할 기회를 식별합니다. 코드 내에서 리소스를 비효율적으로 사용하는 버그 또는 안티 패턴을 식별하기 위해 정기적인 검토 프로세스를 채택합니다. 사용 사례에 대해 동일한 결과를 생성하는 간단하고 효율적인 알고리즘을 활용합니다.

구현 단계

- 효율적인 프로그래밍 언어 사용: 워크로드에 효율적인 운영 체제와 프로그래밍 언어를 사용합니다. 에너지 효율적인 프로그래밍 언어(Rust 포함)에 대한 자세한 내용은 Sustainability with Rust를 참조하세요.
- AI 코딩 도우미 사용: 코드를 효율적으로 작성하기 위해 Amazon CodeWhisperer와 같은 AI 코딩 도우미를 사용하는 것을 고려해 봅니다.

- 코드 검토 자동화: 워크로드를 개발하는 동안 자동화된 코드 검토 프로세스를 채택하여 품질을 개선하고 버그와 안티 패턴을 식별합니다.
  - Amazon CodeGuru Reviewer를 사용한 코드 검토 자동화
  - Amazon CodeGuru를 통한 동시성 버그 탐지
  - Amazon CodeGuru를 사용한 Python 애플리케이션의 코드 품질 향상
- 코드 프로파일러 사용: 코드 프로파일러를 사용하여 가장 많은 시간 또는 리소스를 사용하는 코드 영역을 최적화 대상으로 식별합니다.
  - Amazon CodeGuru Profiler를 사용한 조직의 탄소 배출량 감소
  - Amazon CodeGuru Profiler를 통한 Java 애플리케이션 메모리 사용량 이해
  - Amazon CodeGuru Profiler를 사용한 고객 경험 개선 및 비용 절감
- 모니터링 및 최적화: 지속적인 모니터링 리소스를 사용하여 리소스 요구 사항이 높거나 구성이 최적화되지 않은 구성 요소를 식별합니다.
  - 컴퓨팅 집약적인 알고리즘을 동일한 결과를 내는 보다 단순하고 효율적인 버전으로 대체합니다.
  - 정렬 및 서식 지정과 같은 불필요한 코드를 제거합니다.
- 코드 리팩터링 또는 변환 사용: 애플리케이션 유지 관리 및 업그레이드에 Amazon Q 코드 변환을 사용할 수 있을지 살펴봅니다.
  - Upgrade language versions with Amazon Q Code Transformation
  - AWS re:Invent 2023 - Automate app upgrades & maintenance using Amazon Q Code Transformation

리소스

관련 문서:

- Amazon CodeGuru Profiler란 무엇인가요?
- FPGA 인스턴스
- AWS에서의 구축을 위한 도구의 AWS SDK

관련 동영상:

- Improve Code Efficiency Using Amazon CodeGuru Profiler
- AWS re:Invent 2023 - Best practices for Amazon CodeWhisperer
- Automate Code Reviews and Application Performance Recommendations with Amazon CodeGuru

관련 예시:

- Optimizing Code with Amazon CodeGuru

SUS03-BP04 디바이스 및 장비에 대한 영향 최적화

아키텍처에 사용되는 디바이스와 장비를 이해하고 전략을 바탕으로 사용량을 줄입니다. 이를 통해 클라우드 워크로드의 전반적인 환경 영향을 최소화할 수 있습니다.

일반적인 안티 패턴:

- 고객이 사용하는 디바이스가 환경에 미치는 영향을 무시합니다.
- 고객이 사용하는 리소스를 수동으로 관리하고 업데이트합니다.

이 모범 사례 확립의 이점: 고객 디바이스에 최적화된 소프트웨어 패턴 및 기능을 구현하면 클라우드 워크로드의 전반적인 환경 영향을 줄일 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

고객 디바이스에 최적화된 소프트웨어 패턴 및 기능을 구현하면 다음과 같은 여러 가지 방법으로 환경에 미치는 영향을 줄일 수 있습니다.

- 이전 버전과 호환되는 새 기능을 구현하면 하드웨어 교체 횟수를 줄일 수 있습니다.
- 디바이스에서 효율적으로 실행되도록 애플리케이션을 최적화하면 에너지 소비를 줄이고 배터리 수명을 연장할 수 있습니다(배터리로 작동하는 경우).
- 디바이스의 애플리케이션을 최적화하면 네트워크를 통한 데이터 전송도 줄일 수 있습니다.

아키텍처에 사용되는 디바이스와 장비, 예상 수명 주기 및 이러한 구성 요소 교체의 영향을 이해합니다. 디바이스 에너지 소비와 고객이 디바이스를 교체해야 하는 필요성, 수동 업그레이드를 최소화하는 소프트웨어 패턴과 기능을 구현합니다.

구현 단계

- 인벤토리 구성: 아키텍처에 사용된 디바이스의 인벤토리를 구성합니다. 디바이스는 모바일, 태블릿, IOT 디바이스, 스마트 라이트 또는 공장의 스마트 디바이스일 수 있습니다.

- 에너지 효율적인 디바이스 사용: 아키텍처에 에너지 효율적인 디바이스를 사용하는 것을 고려합니다. 사용하지 않을 때는 디바이스의 전원 관리 구성을 사용하여 저전력 모드로 전환합니다.
- 효율적인 애플리케이션 실행: 디바이스에서 실행되는 애플리케이션을 최적화합니다.
  - 백그라운드에서 태스크를 실행하는 것과 같은 전략을 사용하여 에너지 소비를 줄입니다.
  - 페이로드를 구축할 때 네트워크 대역폭과 지연 시간을 고려하고 애플리케이션이 지연 시간이 긴 저대역폭 링크에서 잘 작동하도록 지원하는 기능을 구현합니다.
  - 페이로드 및 파일을 디바이스에 필요한 최적화된 형식으로 변환합니다. 예를 들어, Amazon Elastic Transcoder 또는 AWS Elemental MediaConvert를 사용하여 대용량의 고품질 디지털 미디어 파일을 사용자가 모바일 디바이스, 태블릿, 웹 브라우저 및 연결된 TV에서 재생할 수 있는 형식으로 변환합니다.
  - 서버 측에서 계산 집약적인 활동(예: 이미지 렌더링)을 수행하거나 애플리케이션 스트리밍을 사용하여 구형 디바이스에서 사용자 경험을 개선합니다.
  - 페이로드를 관리하고 로컬 스토리지 요구 사항을 제한하기 위해 특히 대화형 세션의 경우 출력을 분할하고 페이지 번호를 매깁니다.
- 공급업체 참여: 지속 가능한 소재를 사용하고 공급망의 투명성과 환경 인증을 제공하는 디바이스 공급업체와 협력합니다.
- 무선 업데이트(OTA) 사용: 자동화된 무선 업데이트(OTA) 메커니즘을 사용하여 하나 이상의 디바이스에 업데이트를 배포합니다.
  - CI/CD 파이프라인을 사용하여 모바일 애플리케이션을 업데이트할 수 있습니다.
  - AWS IoT Device Management를 사용하여 연결된 디바이스를 규모에 맞게 원격으로 관리할 수 있습니다.
- 관리형 디바이스 팜 사용: 새로운 기능 및 업데이트를 테스트하려면 대표적인 하드웨어 집합에 관리형 디바이스 팜을 사용하고 개발을 반복하여 지원되는 디바이스를 최대화합니다. 자세한 내용은 SUS06-BP04 테스트에 관리형 Device Farm 사용 페이지를 참조하세요.
- 지속적인 모니터링 및 개선: 디바이스의 에너지 사용량을 추적하여 개선이 필요한 부분을 식별합니다. 새로운 기술이나 모범 사례를 사용하여 이러한 디바이스가 환경에 미치는 영향을 개선합니다.

리소스

관련 문서:

- AWS Device Farm이란 무엇인가요?
- AppStream 2.0 설명서
- NICE DCV

- OTA tutorial for updating firmware on devices running FreeRTOS
- Optimizing Your IoT Devices for Environmental Sustainability

관련 동영상:

- AWS re:Invent 2023 - Improve your mobile and web app quality using AWS Device Farm

SUS03-BP05 데이터 액세스 및 스토리지 패턴을 가장 잘 지원하는 소프트웨어 패턴 및 아키텍처 사용

데이터가 워크로드 내에서 사용되고, 사용자가 소비하고, 전송 및 저장되는 방식을 이해합니다. 데이터 액세스 및 스토리지를 가장 잘 지원하는 소프트웨어 패턴 및 아키텍처를 사용하여 워크로드를 지원하는 데 필요한 컴퓨팅, 네트워킹 및 스토리지 리소스를 최소화합니다.

일반적인 안티 패턴:

- 모든 워크로드의 데이터 스토리지 및 액세스 패턴이 비슷하다고 가정합니다.
- 모든 워크로드가 해당 계층 내에서 적합하다고 가정하고 하나의 스토리지 계층만 사용합니다.
- 시간이 지나면 데이터 액세스 패턴이 일관되게 유지될 것이라고 가정합니다.
- 아키텍처는 높을 가능성이 있는 데이터 액세스 버스트를 지원하므로, 리소스가 대부분 유휴 상태로 유지됩니다.

이 모범 사례 확립의 이점: 데이터 액세스 및 스토리지 패턴을 기반으로 아키텍처를 선택하고 최적화하면 개발 복잡성을 줄이고 전체 활용률을 높일 수 있습니다. 글로벌 테이블, 데이터 파티셔닝 및 캐싱을 사용해야 하는 시기를 파악하면 워크로드 요구 사항에 따라 운영 오버헤드를 줄이고 규모를 확장할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

데이터 특성 및 액세스 패턴에 가장 적합한 소프트웨어 및 아키텍처 패턴을 사용합니다. 예를 들어, AWS 기반 최신 데이터 아키텍처를 통해 고유한 분석 사용 사례에 최적화된 목적별 서비스를 사용할 수 있습니다. 이러한 아키텍처 패턴은 효율적인 데이터 처리를 가능하게 하고 리소스 사용을 줄여 줍니다.

구현 단계

- 데이터 특성 및 액세스 패턴을 분석하여 클라우드 리소스에 적합한 구성을 식별합니다. 고려해야 할 주요 특성은 다음과 같습니다.
  - 데이터 형식: 정형, 반정형 및 비정형
  - 데이터 증가: 제한, 무제한
  - 데이터 내구성: 영구, 임시, 일시적
  - 액세스 패턴(읽기 또는 쓰기, 업데이트 빈도, 급증 또는 일관성 여부)
- 데이터 액세스 및 스토리지 패턴을 가장 잘 지원하는 아키텍처 패턴을 사용합니다.
  - Patterns for enabling data persistence
  - 아키텍트를 시작하겠습니다! 현대적 데이터 아키텍처
  - Databases on AWS: The Right Tool for the Right Job
- 기본적으로 압축된 데이터와 함께 작동하는 기술을 사용합니다.
  - Athena 압축 지원 파일 형식
  - AWS Glue에서 ETL 입력 및 출력의 포맷 옵션
  - Amazon Redshift를 사용하여 Amazon S3에서 압축된 데이터 파일 로드
- 아키텍처의 데이터 처리에 목적별 분석 서비스를 사용합니다. AWS 목적별 분석 서비스에 대한 자세한 내용은 AWS re:Invent 2022 - Building modern data architectures on AWS를 참조하세요.
- 가장 많이 나타나는 쿼리 패턴을 가장 효과적으로 지원하는 데이터베이스 엔진을 사용합니다. 데이터베이스 인덱스를 관리하여 효율적인 쿼리를 보장합니다. 자세한 내용은 AWS 데이터베이스 및 AWS re:Invent 2022 - Modernize apps with purpose-built databases를 참조하세요.
- 아키텍처에 사용되는 네트워크 용량을 줄이는 네트워크 프로토콜을 선택합니다.

리소스

관련 문서:

- Amazon Redshift를 사용한 열 기반 데이터 형식에서 COPY 명령
- Converting Your Input Record Format in Firehose
- 열 형식으로 변환하여 Amazon Athena에서 쿼리 성능 향상
- Amazon Aurora의 성능 개선 도우미로 DB 로드 모니터링
- Amazon RDS의 성능 개선 도우미로 DB 로드 모니터링
- Amazon S3 Intelligent-Tiering 스토리지 클래스

- Build a CQRS event store with Amazon DynamoDB

관련 동영상:

- AWS re:Invent 2022 - Building data mesh architectures on AWS
- AWSre:Invent 2023 - Deep dive into Amazon Aurora and its innovations
- AWS re:Invent 2023 - Improve Amazon EBS efficiency and be more cost-efficient
- AWS re:Invent 2023 - Optimizing storage price and performance with Amazon S3
- AWS re:Invent 2023 - Amazon S3에서 데이터 레이크 구축 및 최적화
- AWS re:Invent 2023 - Amazon EventBridge의 고급 이벤트 기반 패턴

관련 예시:

- AWS 목적별 데이터베이스 워크숍
- AWS Modern Data Architecture Immersion Day
- Build a Data Mesh on AWS

데이터 관리

데이터 관리 원칙을 구현하여 워크로드를 지원하는 데 필요한 프로비저닝된 스토리지와 이를 사용하는 데 필요한 리소스를 줄입니다. 데이터를 이해하고 데이터의 비즈니스 가치와 데이터 사용 방식을 가장 잘 지원하는 스토리지 기술과 구성을 사용합니다. 요구 사항이 감소하면 데이터를 더 효율적이고 성능이 낮은 스토리지로 수명 주기를 변경하고 더 이상 필요하지 않은 데이터는 삭제합니다.

모범 사례

- SUS04-BP01 데이터 분류 정책 구현
- SUS04-BP02 데이터 액세스 및 스토리지 패턴을 지원하는 기술 사용
- SUS04-BP03 정책을 사용하여 데이터 세트의 수명 주기 관리
- SUS04-BP04 탄력성 및 자동화 기능을 사용하여 블록 스토리지 또는 파일 시스템 확장
- SUS04-BP05 불필요하거나 중복된 데이터 제거
- SUS04-BP06 공유 파일 시스템 또는 스토리지를 사용하여 공용 데이터에 액세스
- SUS04-BP07 네트워크 간 데이터 이동 최소화
- SUS04-BP08 다시 생성하기 어려운 경우에만 데이터 백업

SUS04-BP01 데이터 분류 정책 구현

데이터를 분류하여 비즈니스 성과에 대한 중요도를 파악하고 데이터를 저장할 에너지 효율적인 적절한 스토리지 티어를 선택합니다.

일반적인 안티 패턴:

- 처리 중이거나 저장된 데이터 자산 중 특성(예: 민감도, 비즈니스 중요도 또는 규제 요구 사항)이 유사한 데이터 자산을 식별하지 않습니다.
- 데이터 자산의 인벤토리 등록을 위한 데이터 카탈로그를 구현하지 않았습니다.

이 모범 사례 확립의 이점: 데이터 분류 정책을 구현하면 가장 에너지 효율성이 높은 데이터 스토리지 티어를 결정할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

데이터 분류에는 처리 중인 데이터 유형과 조직에서 소유하거나 운영하는 정보 시스템에 저장되는 데이터 유형을 식별하는 작업이 포함됩니다. 또한 데이터의 중요도와 데이터 손상, 손실 또는 남용의 영향을 확인하는 작업도 포함됩니다.

데이터의 상황별 사용에서부터 시작하여 거꾸로 작업하고 조직 운영에 대한 제공된 데이터 세트의 중요도 수준을 고려하는 분류 체계를 만들어 데이터 분류 정책을 구현합니다.

구현 단계

- 데이터 인벤토리 구성: 워크로드에 존재하는 다양한 데이터 유형의 인벤토리를 구성합니다.
- 데이터 그룹화: 조직에 대한 위험을 기준으로 데이터의 중요도, 기밀성, 무결성 및 가용성을 확인합니다. 이러한 요구 사항을 확인하여 채택한 데이터 분류 티어 중 하나로 데이터를 그룹화합니다. 예시는 데이터를 분류하고 Startup을 보호하는 간단한 4단계를 참조하세요.
- 데이터 분류 수준 및 정책 정의: 각 데이터 그룹에 대해 데이터 분류 수준(예: 공개 또는 기밀) 및 처리 정책을 정의합니다. 그에 따라 데이터에 태그를 지정합니다. 데이터 분류 범주에 대한 자세한 내용은 데이터 분류 백서를 참조하세요.
- 주기적 검토: 태그가 지정되지 않은 데이터와 분류되지 않은 데이터가 있는지 환경을 정기적으로 검토하고 감사합니다. 자동화를 사용하여 이 데이터를 식별하고 데이터를 적절하게 분류하고 태그를 지정합니다. 예시는 AWS Glue의 데이터 카탈로그 및 크롤러를 참조하세요.

- 데이터 카탈로그 구축: 감사 및 거버넌스 기능을 제공하는 데이터 카탈로그를 설정합니다.
- 문서화: 각 데이터 클래스에 대한 데이터 분류 정책 및 처리 절차를 문서화합니다.

리소스

관련 문서:

- AWS 클라우드를 활용하여 데이터 분류 지원
- AWS Organizations의 태그 정책

관련 동영상:

- AWS re:Invent 2022 - Enabling agility with data governance on AWS
- AWS re:Invent 2023 - Data protection and resilience with AWS storage

SUS04-BP02 데이터 액세스 및 스토리지 패턴을 지원하는 기술 사용

데이터 액세스 및 저장 방법을 가장 잘 지원하는 스토리지 기술을 사용하여 워크로드를 지원하면서 프로비저닝된 리소스를 최소화합니다.

일반적인 안티 패턴:

- 모든 워크로드의 데이터 스토리지 및 액세스 패턴이 비슷하다고 가정합니다.
- 모든 워크로드가 해당 계층 내에서 적합하다고 가정하고 하나의 스토리지 계층만 사용합니다.
- 시간이 지나면 데이터 액세스 패턴이 일관되게 유지될 것이라고 가정합니다.

이 모범 사례 확립의 이점: 데이터 액세스 및 스토리지 패턴을 기반으로 스토리지 기술을 선택 및 최적화하면 비즈니스 요구 사항을 충족하는 데 필요한 클라우드 리소스를 줄이고 클라우드 워크로드의 전반적인 효율성을 향상시킬 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출되는 위험 수준: 낮음

구현 가이드

따라서 성능 효율성을 극대화하려면 액세스 패턴에 가장 적합한 스토리지 솔루션을 선택하거나, 스토리지 솔루션에 따라 액세스 패턴을 변경하는 것이 좋습니다.

구현 단계

- 데이터 및 액세스 특성 평가: 데이터 특성 및 액세스 패턴을 평가하여 스토리지 요구 사항의 주요 특성을 수집합니다. 고려해야 할 주요 특성은 다음과 같습니다.
  - 데이터 형식: 정형, 반정형 및 비정형
  - 데이터 증가: 제한, 무제한
  - 데이터 내구성: 영구, 임시, 일시적
  - 액세스 패턴: 읽기 또는 쓰기, 빈도, 급증하는지 또는 일관적인지
- 적합한 스토리지 기술 선택: 데이터 특성 및 액세스 패턴을 지원하는 적절한 스토리지 기술로 데이터를 마이그레이션합니다. AWS 스토리지 기술의 몇 가지 예와 주요 특성은 다음과 같습니다.

Type	Technology	Key characteristics
객체 스토리지	Amazon S3	무제한 확장성, 높은 가용성과 액세스 가능성을 위한 여러 옵션을 갖춘 객체 스토리지 서비스입니다. Amazon S3 내외부에서 객체를 전송하고 액세스하면 서비스(예: Transfer Acceleration 또는액세스 포인트)를 사용하여 위치, 보안 요구 사항 및 액세스 패턴을 지원할 수 있습니다.
아카이빙 스토리지	Amazon S3 Glacier	데이터 아카이빙을 위해 구축된 Amazon S3의 스토리지 클래스입니다.
공유 파일 시스템	Amazon Elastic File System(Amazon EFS)	여러 유형의 컴퓨팅 솔루션에서 액세스할 수 있는 탑재 가능한 파일 시스템입니다. Amazon EFS는 스토리지를 자동으로 늘리고 줄이며 성능이 최적화되어 일관되게 지연 시간이 짧습니다.

Type	Technology	Key characteristics
공유 파일 시스템	Amazon FSx	일반적으로 사용되는 네 가지 파일 시스템(NetApp ONTAP, OpenZFS, Windows File Server 및 Lustre)을 지원하기 위해 최신 AWS 컴퓨팅 솔루션을 기반으로 빌드되었습니다. Amazon FSx 지연 시간, 처리량 및 IOPS는 파일 시스템에 따라 달라지며 워크로드의 요구 사항에 적합한 파일 시스템을 선택할 때 고려해야 합니다.
블록 스토리지	Amazon Elastic Block Store(Amazon EBS)	Amazon Elastic Compute Cloud(Amazon EC2)를 위해 설계된 확장 가능한 고성능 블록 스토리지 서비스입니다. Amazon EBS에는 IOPS 집약적 트랜잭션 워크로드를 위한 SSD 지원 스토리지와 처리량 집약적 워크로드를 위한 HDD 지원 스토리지가 포함됩니다.
관계형 데이터베이스	Amazon Aurora, Amazon RDS, Amazon Redshift	ACID(원자성, 일관성, 격리, 내구성) 트랜잭션을 지원하고 참조 무결성과 강력한 데이터 일관성을 유지하도록 설계되었습니다. 많은 기존 애플리케이션, 엔터프라이즈 리소스 계획(ERP), 고객 관계 관리(CRM), 전자 상거래 시스템의 데이터가 관계형 데이터베이스를 사용하여 저장됩니다.

Type	Technology	Key characteristics
키-값 데이터베이스	Amazon DynamoDB	대개 대량의 데이터를 저장 및 검색하는 일반적인 접근 패턴에 최적화되어 있습니다. 트래픽이 많은 웹 앱, 전자 상거래 시스템, 게임 애플리케이션은 키 값 데이터베이스의 일반적인 사용 사례입니다.

- 스토리지 할당 자동화: 크기가 고정된 스토리지 시스템(예: Amazon EBS 또는 Amazon FSx)의 경우 사용 가능한 스토리지 공간을 모니터링하고 임계값에 도달 시 스토리지 할당을 자동화합니다. Amazon CloudWatch를 활용하여 Amazon EBS 및 Amazon FSx에 대한 다양한 지표를 수집 및 분석할 수 있습니다.
- 적합한 스토리지 클래스 선택: 데이터에 적합한 스토리지 클래스를 선택합니다.
  - Amazon S3 스토리지 클래스는 객체 수준에서 구성할 수 있습니다. 단일 버킷에는 모든 스토리지 클래스에 저장된 객체가 포함될 수 있습니다.
  - Amazon S3 수명 주기 정책을 사용하여 애플리케이션 변경 없이 스토리지 클래스 간에 객체를 자동으로 전환하거나 데이터를 제거할 수 있습니다. 이러한 스토리지 메커니즘을 고려할 때 리소스 효율성, 액세스 지연 시간 및 신뢰성 간에 절충해야 합니다.

리소스

관련 문서:

- Amazon EBS 볼륨 유형
- Amazon EC2 인스턴스 스토어
- Amazon S3 Intelligent-Tiering
- Amazon EBS I/O 특성
- Amazon S3 스토리지 클래스 사용
- Amazon S3 Glacier란 무엇인가요?

관련 동영상:

- AWS re:Invent 2023 - Improve Amazon EBS efficiency and be more cost-efficient

- AWS re:Invent 2023 - Optimizing storage price and performance with Amazon S3
- AWS re:Invent 2023 - Amazon S3에서 데이터 레이크 구축 및 최적화
- AWS re:Invent 2022 - AWS에서 현대적 데이터 아키텍처 구축
- AWS re:Invent 2022 - 목적별 데이터베이스로 앱 현대화
- AWS re:Invent 2022 - Building data mesh architectures on AWS
- AWSre:Invent 2023 - Deep dive into Amazon Aurora and its innovations
- AWS re:Invent 2023 - Advanced data modeling with Amazon DynamoDB

관련 예시:

- Amazon S3 예시
- AWS 목적별 데이터베이스 워크숍
- Databases for Developers
- AWS Modern Data Architecture Immersion Day
- Build a Data Mesh on AWS

SUS04-BP03 정책을 사용하여 데이터 세트의 수명 주기 관리

모든 데이터의 수명 주기를 관리하고 삭제를 자동으로 적용하여 워크로드에 필요한 총 스토리지를 최소화합니다.

일반적인 안티 패턴:

- 데이터를 수동으로 삭제합니다.
- 워크로드 데이터를 삭제하지 않습니다.
- 보존 및 액세스 요구 사항에 따라 데이터를 더 에너지 효율적인 스토리지로 이동하지 않습니다.

이 모범 사례 확립의 이점: 데이터 수명 주기 정책을 사용하여 워크로드에서 데이터에 효율적으로 액세스하고 보존할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

데이터 세트는 일반적으로 수명 주기 동안 보존 및 액세스 요구 사항이 각각 다릅니다. 예를 들어, 애플리케이션이 한정된 기간 동안 일부 데이터 세트에 자주 액세스해야 할 수 있습니다. 그 후 해당 데이터 세트에 자주 액세스하지 않습니다.

전체 수명 주기 동안 데이터 세트를 효율적으로 관리하려면 수명 주기 정책을 구성해야 하며, 이것은 데이터 세트를 처리하는 방법을 정의하는 규칙입니다.

수명 주기 구성 규칙을 통해, 데이터 세트를 보다 에너지 효율적인 스토리지 계층으로 전환하고 아카이브하거나 삭제하도록 특정 스토리지 서비스에 요청할 수 있습니다.

구현 단계

- 워크로드의 데이터 세트를 분류합니다.
- 각 데이터 클래스의 처리 절차를 정의합니다.
- 수명 주기 규칙을 적용하는 자동화된 수명 주기 정책을 설정합니다. 다양한 AWS 스토리지 서비스에 대한 자동화된 수명 주기 정책을 설정하는 몇 가지 예는 다음과 같습니다.

Storage service	How to set automated lifecycle policies
Amazon S3	Amazon S3 수명 주기를 사용하여 전체 수명 주기 동안 객체를 관리할 수 있습니다. 액세스 패턴을 알 수 없거나 변화하거나 예측할 수 없는 경우 Amazon S3 Intelligent-Tiering을 사용할 수 있으며, 이를 통해 액세스 패턴을 모니터링하고, 액세스하지 않은 객체를 더 저렴한 액세스 계층으로 자동으로 이동할 수 있습니다. Amazon S3 스토리지 렌즈 지표를 활용하여 수명 주기 관리의 최적화 기회와 격차를 파악할 수 있습니다.
Amazon Elastic Block Store	Amazon Data Lifecycle Manager를 사용하여 Amazon EBS 스냅샷 및 Amazon EBS 지원 AMI의 생성, 보존 및 삭제를 자동화할 수 있습니다.

Storage service	How to set automated lifecycle policies
Amazon Elastic File System	Amazon EFS 수명 주기 관리는 파일 시스템의 파일 스토리지를 자동으로 관리합니다.
Amazon Elastic Container Registry	Amazon ECR 수명 주기 정책은 수명 또는 개수를 기준으로 만료되는 이미지에 의한 컨테이너 이미지 정리를 자동화합니다.
AWS Elemental MediaStore	MediaStore 컨테이너에 객체를 얼마 동안 저장해야 하는지 제어하는 객체 수명 주기 정책을 사용할 수 있습니다.

- 사용되지 않은 볼륨, 스냅샷 및 보존 기간이 지난 데이터를 삭제합니다. 삭제할 Amazon DynamoDB Time To Live 또는 Amazon CloudWatch 로그 보존과 같은 네이티브 서비스 기능을 활용합니다.
- 수명 주기 규칙에 따라 관련 데이터를 집계 및 압축합니다.

리소스

관련 문서:

- Amazon S3 스토리지 클래스 분석을 통해 Amazon S3 수명 주기 규칙 최적화
- AWS Config 규칙으로 리소스 평가

관련 동영상:

- AWS re:Invent 2021 - Amazon S3 Lifecycle best practices to optimize your storage spend
- AWS re:Invent 2023 - Optimizing storage price and performance with Amazon S3
- Amazon S3 수명 주기로 데이터 수명 주기 간소화 및 스토리지 비용 최적화
- Amazon S3 스토리지 렌즈를 사용한 스토리지 비용 절감

SUS04-BP04 탄력성 및 자동화 기능을 사용하여 블록 스토리지 또는 파일 시스템 확장

데이터가 늘어나면서 블록 스토리지 또는 파일 시스템을 확장하여 프로비저닝된 총 스토리지를 최소화하는 탄력성 및 자동화 기능을 사용합니다.

일반적인 안티 패턴:

- 향후 필요에 따라 대규모 블록 스토리지 또는 파일 시스템을 조달합니다.
- 파일 시스템의 초당 입출력 작업 처리량(IOPS)을 초과 프로비저닝합니다.
- 데이터 볼륨의 사용률을 모니터링하지 않습니다.

이 모범 사례 확립의 이점: 스토리지 시스템에 대한 초과 프로비저닝을 최소화하면 유휴 리소스가 줄어들고 워크로드의 전반적인 효율성이 향상됩니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

워크로드에 적합한 크기 할당, 처리량 및 지연 시간으로 블록 스토리지 및 파일 시스템을 생성합니다. 이러한 스토리지 서비스를 과도하게 프로비저닝하지 않고도 데이터 증가에 따라 블록 스토리지 또는 파일 시스템을 확장할 수 있는 탄력성 및 자동화 기능을 사용합니다.

구현 단계

- Amazon EBS와 같이 크기가 고정된 스토리지의 경우 전체 스토리지 크기 대비 사용된 스토리지의 양을 모니터링해야 하며, 가능하다면 임계값에 도달할 때 스토리지 크기를 늘릴 수 있는 자동화 기능을 생성해야 합니다.
- 탄력적 볼륨 및 관리형 블록 데이터 서비스를 사용하여 영구 데이터의 증가에 따른 추가 스토리지 할당을 자동화합니다. 예를 들어, Amazon EBS 탄력적 볼륨을 사용하면 볼륨 크기와 볼륨 유형을 변경하거나 Amazon EBS 볼륨의 성능을 조정할 수 있습니다.
- 파일 시스템에 적합한 스토리지 클래스, 성능 모드 및 처리량 모드를 선택하여 비즈니스 요구 사항을 충족하세요. 초과할 필요는 없습니다.
  - Amazon EFS 성능
  - Amazon EBS volume performance on Linux instances
- 데이터 볼륨의 목표 사용률 수준을 설정하고 예상 범위를 벗어나는 볼륨 크기를 조정합니다.
- 데이터에 적합하도록 읽기 전용 볼륨의 크기를 알맞게 조정합니다.
- 블록 스토리지의 고정 볼륨 크기로 초과 용량을 프로비저닝하지 않도록 데이터를 객체 스토어로 마이그레이션합니다.
- 탄력적인 볼륨 및 파일 시스템을 정기적으로 검토하여 유휴 볼륨을 종료하고 과도하게 프로비저닝된 리소스를 현재 데이터 크기에 맞게 축소합니다.

리소스

관련 문서:

- Extend the file system after resizing an EBS volume
- Modify a volume using Amazon EBS Elastic Volumes
- Amazon FSx 설명서
- Amazon Elastic File System이란 무엇일까요?

관련 동영상:

- Deep Dive on Amazon EBS Elastic Volumes
- Amazon EBS and Snapshot Optimization Strategies for Better Performance and Cost Savings
- Optimizing Amazon EFS for cost and performance, using best practices

SUS04-BP05 불필요하거나 중복된 데이터 제거

불필요하거나 중복된 데이터를 제거하여 데이터세트를 저장하는 데 필요한 스토리지 리소스를 최소화합니다.

일반적인 안티 패턴:

- 쉽게 얻을 수 있거나 다시 생성할 수 있는 데이터를 중복합니다.
- 데이터의 중요도를 고려하지 않고 모든 데이터를 백업합니다.
- 데이터를 불규칙하게 또는 운영 이벤트에만 삭제하거나 전혀 삭제하지 않습니다.
- 스토리지 서비스의 내구성에 관계없이 데이터를 중복 저장합니다.
- 업무상 타당한 이유 없이 Amazon S3 버전 관리를 켭니다.

이 모범 사례 확립의 이점: 불필요한 데이터를 제거하면 워크로드 및 워크로드의 환경 영향에 필요한 스토리지 크기를 줄일 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

필요하지 않은 데이터를 저장하지 않습니다. 불필요한 데이터의 삭제를 자동화합니다. 파일 및 블록 수준에서 데이터 중복을 제거하는 기술을 사용합니다. 서비스의 네이티브 데이터 복제 및 중복성 기능을 활용합니다.

구현 단계

- AWS Data Exchange의 기존 공개 데이터세트 및 AWS의 개방형 데이터를 사용하여 데이터 저장을 방지할 수 있는지 여부를 평가합니다.
- 블록 및 객체 수준에서 데이터 중복을 제거할 수 있는 메커니즘을 사용합니다. 다음은 AWS의 데이터 중복을 제거하는 방법의 몇 가지 예입니다.

Storage service	Deduplication mechanism
Amazon S3	새로운 FindMatches ML Transform을 사용하여 데이터세트 전체(식별자가 없는 것 포함)에서 일치하는 레코드를 찾으려면 AWS Lake Formation FindMatches를 사용합니다.
Amazon FSx	Amazon FSx for Windows에서 데이터 중복 제거를 사용합니다.
Amazon Elastic Block Store 스냅샷	스냅샷은 증분식 백업입니다. 즉, 가장 최근 스냅샷 이후에 변경된 디바이스의 블록만 저장됩니다.

- 데이터 액세스를 분석하여 불필요한 데이터를 식별합니다. 수명 주기 정책을 자동화합니다. 삭제할 Amazon DynamoDB Time To Live, Amazon S3 수명 주기 또는 Amazon CloudWatch 로그 보존과 같은 네이티브 서비스 기능을 활용합니다.
- AWS의 데이터 가상화 기능을 사용하여 소스의 데이터를 유지 관리하고 데이터 중복을 방지합니다.
  - AWS의 클라우드 네이티브 데이터 가상화
  - Amazon Redshift 데이터 공유를 사용하여 데이터 패턴 최적화
- 증분식 백업을 만들 수 있는 백업 기술을 사용합니다.
- Amazon S3의 내구성 및 Amazon EBS의 복제를 활용하여 자체 관리형 기술(예: 독립 디스크의 이중화 어레이(RAID)) 대신 내구성 목표를 달성합니다.

- 로그 및 추적 데이터를 중앙 집중화하고, 동일한 로그 항목을 중복 제거하며, 필요에 따라 세부적으로 조정하는 메커니즘을 설정합니다.
- 적절한 경우에만 캐시를 미리 채웁니다.
- 캐시 모니터링 및 자동화를 설정하여 그에 따라 캐시 크기를 조정합니다.
- 새 버전의 워크로드를 푸시할 때 객체 스토어 및 엣지 캐시에서 오래된 배포 및 자산을 제거합니다.

리소스

관련 문서:

- CloudWatch Logs에서 로그 데이터 보존 변경
- Amazon FSx for Windows File Server에서 데이터 중복 제거
- 데이터 중복 제거를 포함한 Amazon FSx for ONTAP의 기능
- Amazon CloudFront에서의 파일 무효화
- AWS Backup을 사용하여 Amazon EFS 파일 시스템 백업 및 복구
- Amazon CloudWatch Logs란 무엇인가요?
- Amazon RDS의 백업 작업
- Integrate and deduplicate datasets using AWS Lake Formation

관련 동영상:

- Amazon Redshift Data Sharing Use Cases

관련 예시:

- Amazon Athena를 사용하여 Amazon S3 서버 액세스 로그를 분석하려면 어떻게 해야 합니까?

SUS04-BP06 공유 파일 시스템 또는 스토리지를 사용하여 공용 데이터에 액세스

공유 파일 시스템 또는 스토리지를 채택하여 데이터 중복을 방지하고 워크로드를 위한 보다 효율적인 인프라를 지원합니다.

일반적인 안티 패턴:

- 각 개별 클라이언트에 대해 스토리지를 프로비저닝합니다.
- 비활성 클라이언트에서 데이터 볼륨을 분리하지 않습니다.
- 플랫폼 및 시스템 전반에 걸쳐 스토리지에 액세스할 권한을 제공하지 않습니다.

이 모범 사례 확립의 이점: 공유 파일 시스템 또는 스토리지를 사용하면 데이터를 복사하지 않고도 1명 이상의 소비자에게 데이터를 공유하도록 할 수 있습니다. 이를 통해 워크로드에 필요한 스토리지 리소스를 줄일 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

여러 사용자 또는 애플리케이션이 동일한 데이터세트에 액세스하는 경우 워크로드에 효율적인 인프라를 사용하기 위해서는 공유 스토리지 기술을 사용해야 합니다. 공유 스토리지 기술은 데이터세트를 저장 및 관리하고 데이터 중복을 방지할 수 있는 중앙 위치를 제공합니다. 또한 여러 시스템에 걸쳐 데이터의 일관성을 적용합니다. 나아가 공유 스토리지 기술을 사용하면 여러 컴퓨팅 리소스가 동시에 데이터에 액세스하고 이를 처리할 수 있으므로 컴퓨팅 성능을 보다 효율적으로 사용할 수 있습니다.

이러한 공유 스토리지 서비스에서 필요한 경우에만 데이터를 가져오고 사용하지 않는 볼륨을 분리하여 리소스를 확보하세요.

구현 단계

- 데이터의 소비자가 다수인 경우 데이터를 공유 스토리지로 마이그레이션합니다. AWS 기반 공유 스토리지 기술의 몇 가지 예는 다음과 같습니다.

Storage option	When to use
Amazon EBS 다중 연결	Amazon EBS 다중 연결을 사용하면 하나의 프로비저닝된 IOPS SSD(io1 또는 io2) 볼륨을 동일한 가용 영역에 있는 여러 인스턴스에 연결할 수 있습니다.
Amazon EFS	When to Choose Amazon EFS(Amazon EFS 선택 시점)를 참조하세요.
Amazon FSx	Amazon FSx 파일 시스템 선택을 참조하세요.

Storage option	When to use
Amazon S3	파일 시스템 구조가 필요하지 않고 객체 스토리지와 함께 작동하도록 설계된 애플리케이션은 Amazon S3를 대규모로 확장 가능하고 내구성이 뛰어난 저비용 객체 스토리지 솔루션으로 사용할 수 있습니다.

- 필요한 경우에만 공유 파일 시스템에 데이터를 복사하거나 데이터를 가져옵니다. 예를 들어, Amazon S3를 바탕으로 하는 Amazon FSx for Lustre 파일 시스템을 생성하고 작업을 처리하는 데 필요한 데이터의 하위 집합만 Amazon FSx로 로드할 수 있습니다.
- SUS04-BP03 정책을 사용하여 데이터 세트의 수명 주기 관리에 나와 있는 사용 패턴에 따라 데이터를 삭제합니다.
- 자주 사용하지 않는 클라이언트에서 볼륨을 분리합니다.

리소스

관련 문서:

- Linking your file system to an Amazon S3 bucket(파일 시스템을 S3 버킷에 연결)
- Using Amazon EFS for AWS Lambda in your serverless applications(서버리스 애플리케이션에서 AWS Lambda용 Amazon EFS 사용)
- Amazon EFS Intelligent-Tiering Optimizes Costs for Workloads with Changing Access Patterns(Amazon EFS Intelligent-Tiering을 통해 액세스 패턴 변화에 따른 워크로드 비용 최적화)
- Using Amazon FSx with your on-premises data repository(온프레미스 데이터 리포지토리에 Amazon FSx 사용)

관련 동영상:

- Storage cost optimization with Amazon EFS
- AWS re:Invent 2023 - What's new with AWS file storage
- AWS re:Invent 2023 - File storage for builders and data scientists on Amazon Elastic File System

SUS04-BP07 네트워크 간 데이터 이동 최소화

공유 파일 시스템 또는 객체 스토리지를 사용하여 공통 데이터에 액세스하고 워크로드의 데이터 이동을 지원하는 데 필요한 총 네트워킹 리소스를 최소화합니다.

일반적인 안티 패턴:

- 데이터 사용자의 위치에 관계없이 모든 데이터를 동일한 AWS 리전에 저장합니다.
- 네트워크를 통해 데이터를 이동하기 전에 데이터 크기와 형식을 최적화하지 않습니다.

이 모범 사례 확립의 이점: 네트워크 간 데이터 이동을 최적화하면 워크로드에 필요한 총 네트워킹 리소스가 줄어들고 환경에 미치는 영향도 줄어듭니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

조직 전체에서 데이터를 옮기려면 컴퓨팅, 네트워킹 및 스토리지 리소스가 필요합니다. 기술을 사용하여 데이터 이동을 최소화하고 워크로드의 전반적인 효율성을 개선합니다.

구현 단계

- 워크로드의 리전을 선택할 때 데이터 또는 사용자와의 근접성을 결정 요소로 고려하세요.
- 리전별 사용 서비스를 분할하여 해당 리전별 데이터가 사용되는 리전 내에 저장되도록 합니다.
- 네트워크를 통해 데이터를 이동하기 전에 효율적인 파일 형식(예: Parquet 또는 ORC)을 사용하고 데이터를 압축합니다.
- 사용하지 않는 데이터는 옮기지 마십시오. 사용하지 않는 데이터의 이동을 방지하는 데 도움이 되는 몇 가지 예는 다음과 같습니다.
  - 관련 데이터에 대해서만 API 응답을 줄입니다.
  - 상세한 경우 데이터를 집계합니다(레코드 수준 정보는 필요하지 않음).
  - Well-Architected 실습 - Amazon Redshift 데이터 공유를 사용하여 데이터 패턴 최적화를 참조하세요.
  - Cross-account data sharing in AWS Lake Formation을 고려하세요.
- 워크로드 사용자에게 더 가까운 위치에서 코드를 실행할 수 있는 서비스를 사용합니다.

Service	When to use
Lambda@Edge	객체가 캐시에 없는 경우 실행되는 컴퓨팅 집약적 작업에 사용합니다.
CloudFront 함수	HTTP(s) 요청/응답 조작 등과 같이 단기 실행 함수에 의해 시작될 수 있는 간단한 사용 사례에 사용합니다.
AWS IoT Greengrass	커넥티드 디바이스를 위한 로컬 컴퓨팅, 메시징 및 데이터 캐시를 실행합니다.

리소스

관련 문서:

- Optimizing your AWS Infrastructure for Sustainability, Part III: Networking(지속 가능성을 위한 AWS 인프라 최적화, 파트 III: 네트워킹)
- AWS 글로벌 인프라
- CloudFront 글로벌 엣지 네트워크를 포함한 Amazon CloudFront 주요 기능
- Amazon OpenSearch Service에서 HTTP 요청 압축
- Amazon EMR을 사용하여 중간 데이터 압축
- Amazon S3에서 Amazon Redshift로 압축된 데이터 파일 로드
- Amazon CloudFront를 사용하여 압축된 파일 제공

관련 동영상:

- Demystifying data transfer on AWS

관련 예시:

- 지속 가능성을 위한 설계 - 네트워크 간 데이터 이동 최소화

SUS04-BP08 다시 생성하기 어려운 경우에만 데이터 백업

비즈니스 가치가 없는 데이터는 백업하지 않으면서 워크로드의 스토리지 리소스 요구 사항을 최소화합니다.

일반적인 안티 패턴:

- 데이터에 대한 백업 전략이 없습니다.
- 쉽게 다시 생성할 수 있는 데이터를 백업합니다.

이 모범 사례 확립의 이점: 중요하지 않은 데이터를 백업하지 않으면 워크로드에 필요한 스토리지 리소스를 줄이고 환경에 미치는 영향도 줄일 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

불필요한 데이터를 백업하지 않으면 비용을 절감하고 워크로드에 사용되는 스토리지 리소스를 줄일 수 있습니다. 비즈니스 가치가 있거나 규정 준수 요구 사항을 충족하는 데 필요한 데이터만 백업합니다. 백업 정책을 검토하고 복구 시나리오에서 가치를 제공하지 않는 임시 스토리지는 제외합니다.

구현 단계

- SUS04-BP01 데이터 분류 정책 구현에 나와 있는 대로 데이터 분류 정책을 구현합니다.
- 데이터 분류 중요도를 바탕으로 Recovery Time Objective(RTO) 및 Recovery Point Objective(RPO) 기반 백업 전략을 설계합니다. 중요하지 않은 데이터는 백업하지 마세요.
  - 쉽게 다시 생성할 수 있는 데이터는 제외합니다.
  - 백업 대상에서 임시 데이터를 제외합니다.
  - 공용 위치에서 데이터를 복원하는 데 필요한 시간이 서비스 수준에 관한 계약(SLA)을 초과하지 않는 한, 데이터의 로컬 사본을 제외합니다.
- 자동화된 솔루션 또는 관리형 서비스를 사용하여 비즈니스 크리티컬 데이터를 백업합니다.
  - AWS Backup은 완전관리형 서비스로, AWS 서비스 전반, 클라우드 및 온프레미스에서 데이터 보호를 쉽게 중앙 집중화하고 자동화할 수 있습니다. AWS Backup을 사용하여 자동화된 백업을 생성하는 방법에 대한 실습 지침은 Well-Architected Labs - Testing Backup and Restore of Data(Well-Architected 실습 - 데이터 백업 및 복원 테스트)를 참조하세요.
  - AWS Backup을 사용하여 Amazon EFS용 백업을 자동화하고 백업 비용을 최적화합니다.

리소스

관련 모범 사례:

- REL09-BP01 백업해야 하는 모든 데이터 확인 및 백업 또는 소스에서 데이터 재현
- REL09-BP03 자동으로 데이터 백업 수행
- REL13-BP02 복구 목표 달성을 위해 정의된 복구 전략 사용

관련 문서:

- AWS Backup을 사용하여 Amazon EFS 파일 시스템 백업 및 복구
- Amazon EBS 스냅샷
- Amazon Relational Database Service의 백업 작업
- APN 파트너: 백업을 지원할 수 있는 파트너
- AWS Marketplace: 백업에 사용할 수 있는 제품
- Amazon EFS 백업
- Backing Up Amazon FSx for Windows File Server(Amazon FSx for Windows File Server 백업)
- Amazon ElastiCache for Redis 백업 및 복구

관련 동영상:

- AWS re:Invent 2023 - Backup and disaster recovery strategies for increased resilience
- AWS re:Invent 2023 - What's new with AWS Backup
- AWS re:Invent 2021 - Backup, disaster recovery, and ransomware protection with AWS

관련 예시:

- Well-Architected Lab - Backup data

하드웨어 및 서비스

하드웨어 관리 방식을 변경하여 지속 가능성에 미치는 워크로드의 영향을 줄일 수 있는 기회를 모색합니다. 프로비저닝 및 배포에 필요한 하드웨어의 양을 최소화하고 개별 워크로드에 가장 효율적인 하드웨어 및 서비스를 선택합니다.

모범 사례

- SUS05-BP01 요구 사항을 충족하는 데 필요한 최소한의 하드웨어 사용
- SUS05-BP02 영향이 가장 적은 인스턴스 유형 사용
- SUS05-BP03 관리형 서비스 사용
- SUS05-BP04 하드웨어 기반 컴퓨팅 액셀러레이터의 사용 최적화

SUS05-BP01 요구 사항을 충족하는 데 필요한 최소한의 하드웨어 사용

비즈니스 요구 사항을 효율적으로 충족하기 위해 워크로드에 필요한 최소한의 하드웨어를 사용합니다.

일반적인 안티 패턴:

- 리소스 사용률을 모니터링하지 않습니다.
- 아키텍처의 사용률 수준이 낮은 리소스가 있습니다.
- 크기 조정이 필요한지 여부를 결정하기 위해 정적 하드웨어의 사용률 검토를 수행하지 않습니다.
- 비즈니스 KPI를 기반으로 컴퓨팅 인프라의 하드웨어 사용률 목표를 설정하지 않습니다.

이 모범 사례 확립의 이점: 클라우드 리소스의 크기를 적절하게 조정하면 워크로드의 환경 영향을 줄이고, 비용을 절감하며, 성능 벤치마크를 유지하는 데 도움이 됩니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

워크로드에 필요한 총 하드웨어 수를 최적으로 선택하여 전반적인 효율성을 개선합니다. AWS 클라우드는 다양한 메커니즘(예: AWS Auto Scaling)을 통해 리소스 수를 동적으로 확장하거나 줄일 수 있는 유연성을 제공하고 수요 변화에 대응합니다. 또한, 최소한의 노력으로 리소스를 수정할 수 있는 API 및 SDK도 제공합니다. 이러한 기능을 사용하여 워크로드 구현을 자주 변경할 수 있습니다. 또한 AWS 도구의 적정 크기 조정 지침을 바탕으로 클라우드 리소스를 효율적으로 운영하고 비즈니스 요구 사항을 충족할 수 있습니다.

구현 단계

- 인스턴스 유형 선택: 요구 사항에 가장 적합한 인스턴스 유형을 선택합니다. Amazon Elastic Compute Cloud 인스턴스를 선택하고 속성 기반 인스턴스 선택과 같은 메커니즘을 사용하는 방법에 대해 알아보려면 다음을 참조하세요.

  - 워크로드에 적합한 Amazon EC2 인스턴스 유형을 선택하려면 어떻게 해야 하나요?
  - Amazon EC2 플릿에 대한 속성 기반 인스턴스 유형 선택
  - 속성 기반 인스턴스 유형 선택을 사용하여 Auto Scaling 그룹 생성
- 규모 조정: 작은 증분 단위로 가변 워크로드의 규모를 조정합니다.
- 다양한 컴퓨팅 구매 옵션 사용: 여러 컴퓨팅 구매 옵션으로 인스턴스 유연성, 확장성, 비용 절감의 균형을 조정합니다.
  - Amazon EC2 온디맨드 인스턴스는 인스턴스 유형, 위치 또는 시간을 유연하게 지정할 수 없는 상태 저장 방식의 신규 급증 워크로드에 가장 적합합니다.
  - Amazon EC2 스팟 인스턴스는 내결함성 및 유연성이 뛰어난 애플리케이션에서 다른 옵션을 보완하는 데 유용한 방법입니다.
  - 안정적인 상태의 워크로드에는 컴퓨팅 절감형 플랜을 활용하여 변화하는 요구 사항(예: AZ, 리전, 인스턴스 제품군 또는 인스턴스 유형)에 유연성을 제공합니다.
- 인스턴스 및 가용 영역 다양성 사용: 인스턴스와 가용 영역을 다양화하여 애플리케이션 가용성을 극대화하고 초과 용량을 활용합니다.
- 인스턴스 적정 크기 조정: AWS 도구에서 적정 크기 조정 권장 사항을 사용하여 워크로드를 조정합니다. 자세한 내용은 Optimizing your cost with Rightsizing Recommendations 및 올바른 크기 조정: 워크로드에 맞게 인스턴스 프로비저닝을 참조하세요.
  - AWS Cost Explorer의 적정 크기 조정 권장 사항 또는 AWS Compute Optimizer를 사용하여 적정 크기 조정 기회를 식별합니다.
- 서비스 수준에 관한 계약(SLA) 협상: 자동화가 대체 리소스를 배포하는 동안 일시적으로 용량을 줄이도록 허용하는 SLA를 협상합니다.

리소스

관련 문서:

- Optimizing your AWS Infrastructure for Sustainability, Part I: Compute(지속 가능성을 위한 AWS 인프라 최적화, 파트 I: 컴퓨팅)
- Amazon EC2 플릿의 Auto Scaling을 위한 속성 기반 인스턴스 유형 선택
- AWS Compute Optimizer 설명서
- Operating Lambda: Performance optimization(Lambda 운영: 성능 최적화)
- Auto Scaling 설명서

관련 동영상:

- AWS re:Invent 2023 - What's new with Amazon EC2
- AWS re:Invent 2023 - Smart savings: Amazon Elastic Compute Cloud cost-optimization strategies
- AWS re:Invent 2022 - Optimizing Amazon Elastic Kubernetes Service for performance and cost on AWS
- AWS re:Invent 2023 - Sustainable compute: reducing costs and carbon emissions with AWS

SUS05-BP02 영향이 가장 적은 인스턴스 유형 사용

새로운 인스턴스 유형을 지속적으로 모니터링하고 사용하여 에너지 효율 개선을 활용합니다.

일반적인 안티 패턴:

- 인스턴스 패밀리는 하나만 사용합니다.
- x86 인스턴스만 사용합니다.
- Amazon EC2 Auto Scaling 구성에서 인스턴스 유형을 하나만 지정합니다.
- AWS 인스턴스를 설계되지 않은 방식으로 사용합니다(예: 컴퓨팅에 최적화된 인스턴스를 메모리 집약적 워크로드에 사용).
- 새 인스턴스 유형을 정기적으로 평가하지 않습니다.
- AWS 적정 크기 조정 도구(예: AWS Compute Optimizer)에서 권장 사항을 확인하지 않습니다.

이 모범 사례 확립의 이점: 적정 크기로 조정된 에너지 효율적인 인스턴스를 사용하면 환경 영향 및 워크로드 비용을 크게 줄일 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

클라우드 워크로드에서 효율적인 인스턴스를 사용하는 것은 리소스 사용량을 줄이고 비용 효율성을 높이는 데 매우 중요합니다. 새로운 인스턴스 유형의 릴리스를 지속적으로 모니터링하고 기계 학습 훈련 및 추론, 동영상 트랜스코딩과 같은 특정 워크로드를 지원하도록 설계된 인스턴스 유형을 포함하여 에너지 효율성 개선의 이점을 활용합니다.

구현 단계

- 인스턴스 유형 알아보기 및 탐색: 워크로드가 환경에 미치는 영향을 줄일 수 있는 인스턴스 유형을 찾습니다.
  - AWS의 새로운 소식을 구독하여 최신 AWS 기술 및 인스턴스 소식을 파악합니다.
  - 다양한 AWS 인스턴스 유형에 대해 알아봅니다.
  - re:Invent 2020 - Deep dive on AWS Graviton2 processor-powered Amazon EC2 instances 및 Deep dive into AWS Graviton3 and Amazon EC2 C7g instances를 시청하여 Amazon EC2에서 에너지 사용 와트당 최고의 성능을 제공하는 AWS Graviton 기반 인스턴스에 대해 알아보세요.
- 영향이 가장 적은 인스턴스 유형 사용: 워크로드를 계획하고 영향이 가장 적은 인스턴스 유형으로 워크로드를 전환합니다.
  - 워크로드를 위한 새로운 기능 또는 인스턴스를 평가하기 위한 프로세스를 정의합니다. 클라우드에서 민첩성을 활용하여 새 인스턴스 유형이 워크로드 환경 지속 가능성을 어떻게 개선할 수 있는지 신속하게 테스트합니다. 프록시 지표를 사용하여 작업 단위를 완료하는 데 필요한 리소스를 측정합니다.
  - 가능한 경우 다양한 vCPU 수와 다양한 메모리 용량으로 작동하도록 워크로드를 수정하여 인스턴스 유형 선택의 폭을 극대화합니다.
  - 워크로드의 성능 효율성을 개선하려면 워크로드를 Graviton 기반 인스턴스로 전환할 것을 고려합니다. AWS Graviton으로 워크로드를 이동하는 방법에 대한 자세한 내용은 AWS Graviton Fast Start 및 Considerations when transitioning workloads to AWS Graviton-based Amazon Elastic Compute Cloud instances를 참조하세요.
  - AWS 관리형 서비스를 사용할 때 AWS Graviton 옵션을 선택하는 것을 고려해 보세요.
  - 지속 가능성에 미치는 영향이 가장 적고 비즈니스 요구 사항을 충족하는 인스턴스를 제공하는 리전으로 워크로드를 마이그레이션합니다.
  - 기계 학습 워크로드의 경우 AWS Trainium, AWS Inferentia 및 Amazon EC2 DL1과 같이 워크로드에 따라 다른 특별히 구축된 하드웨어의 장점을 활용합니다. Inf2 인스턴스와 같은 AWS Inferentia 인스턴스는 동급 Amazon EC2 인스턴스에 비해 최대 50% 더 우수한 와트당 성능을 제공합니다.
  - Amazon SageMaker Inference Recommender를 사용하여 ML 추론 엔드포인트의 크기를 올바르게 설정합니다.
  - 사용량이 급증하는 워크로드의 경우(추가 용량에 대한 요구 사항이 적은 워크로드) 성능 버스트 가능 인스턴스를 사용합니다.
  - 상태 비저장 및 내결함성 워크로드의 경우 Amazon EC2 스팟 인스턴스를 사용하여 클라우드의 전체 활용률을 높이고 사용되지 않는 리소스가 지속 가능성에 미치는 영향을 줄입니다.
- 운영 및 최적화: 워크로드 인스턴스를 운영하고 최적화합니다.

- 임시 워크로드의 경우 `CPUUtilization`과 같은 인스턴스 Amazon CloudWatch 지표를 평가하여 인스턴스가 유휴 상태인지 활용률이 낮은지 확인합니다.
- 안정적인 워크로드를 위해 정기적으로 AWS Compute Optimizer 등의 AWS 적정 크기 조정 도구를 확인하여 인스턴스를 최적화하고 크기를 조정할 수 있는 기회를 식별합니다.
  - Well-Architected 실습 - 적정 크기 조정 권장 사항
  - Well-Architected 실습 - Compute Optimizer로 적정 크기 조정
  - Well-Architected 실습 - 하드웨어 패턴 최적화 및 지속 가능성 KPI 관찰

리소스

관련 문서:

- Optimizing your AWS Infrastructure for Sustainability, Part I: Compute(지속 가능성을 위한 AWS 인프라 최적화, 파트 I: 컴퓨팅)
- AWS Graviton
- Amazon EC2 DL1
- Amazon EC2 용량 예약 플릿
- Amazon EC2 스팟 플릿
- 함수: Lambda 함수 구성
- Amazon EC2 플릿에 대한 속성 기반 인스턴스 유형 선택
- Building Sustainable, Efficient, and Cost-Optimized Applications on AWS
- Contino 지속 가능성 대시보드가 고객의 탄소 발자국 줄이기를 지원하는 방법

관련 동영상:

- AWS re:Invent 2023 - AWS Graviton: The best price performance for your AWS workloads
- AWS re:Invent 2023 - New Amazon Elastic Compute Cloud generative AI capabilities in AWS Management Console
- AWS re:Invent 2023 = What's new with Amazon Elastic Compute Cloud
- AWS re:Invent 2023 - Smart savings: Amazon Elastic Compute Cloud cost-optimization strategies
- AWS re:Invent 2021 - Deep dive into AWS Graviton3 and Amazon EC2 C7g instances
- AWS re:Invent 2022 - Build a cost-, energy-, and resource-efficient compute environment

관련 예시:

- 솔루션: AWS에서 지속 가능성을 위해 딥 러닝 워크로드를 최적화하기 위한 지침
- Migrating Amazon Relational Database Service Databases to Graviton

SUS05-BP03 관리형 서비스 사용

관리형 서비스를 사용하여 클라우드에서 보다 효율적으로 운영합니다.

일반적인 안티 패턴:

- 활용률이 낮은 Amazon EC2 인스턴스를 사용하여 애플리케이션을 실행합니다.
- 사내 팀이 혁신이나 단순화에 집중할 시간 없이 워크로드만 관리합니다.
- 관리형 서비스에서 보다 효율적으로 실행할 수 있는 태스크를 위한 기술을 배포하고 유지 관리합니다.

이 모범 사례 확립의 이점:

- 관리형 서비스를 사용하면 새로운 혁신과 효율성을 추진하는 데 도움이 될 수 있는 수백만 명의 고객 인사이트를 보유한 AWS에 책임을 맡길 수 있습니다.
- 관리형 서비스는 멀티 테넌트 컨트롤 플레인으로 인해 많은 사용자에게 서비스의 환경 영향을 분산시킵니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

관리형 서비스는 배포된 하드웨어의 높은 사용률과 지속 가능성 최적화를 유지하는 책임을 AWS로 이전합니다. 또한 관리형 서비스를 사용하면 서비스를 유지 관리해야 하는 운영 및 관리 부담이 줄어들기 때문에 팀이 혁신에 더 많은 시간을 할애하고 집중할 수 있습니다.

워크로드를 검토하여 AWS 관리형 서비스로 대체할 수 있는 구성 요소를 식별합니다. 예를 들어, Amazon RDS, Amazon Redshift, Amazon ElastiCache에서는 관리형 데이터베이스 서비스를 제공합니다. Amazon Athena, Amazon EMR, Amazon OpenSearch Service에서는 관리형 분석 서비스를 제공합니다.

구현 단계

1. 워크로드 인벤토리 작성: 서비스 및 구성 요소에 대한 워크로드의 인벤토리를 작성합니다.
2. 후보 파악: 관리형 서비스로 대체할 수 있는 구성 요소를 평가하고 식별합니다. 관리형 서비스 사용을 고려할 수 있는 몇 가지 예는 다음과 같습니다.

Task	What to use on AWS
데이터베이스 호스팅	Amazon Elastic Compute Cloud(Amazon EC2)에서 자체 Amazon RDS 인스턴스를 유지 관리하는 대신 관리형 Amazon Relational Database Service(Amazon RDS) 인스턴스를 사용합니다.
컨테이너 워크로드 호스팅	자체 컨테이너 인프라를 구현하는 대신 AWS Fargate를 사용합니다.
웹 앱 호스팅	AWS Amplify 호스팅을 정적 웹 사이트 및 서버 측 렌더링 웹 앱을 위한 완전관리형 CI/CD 및 호스팅 서비스로 사용합니다.

3. 마이그레이션 계획 수립: 종속성을 식별하고 마이그레이션 계획을 수립합니다. 그에 따라 런북과 플레이북을 업데이트합니다.
    - AWS Application Discovery Service는 애플리케이션 종속성 및 활용에 대한 세부적인 정보를 자동으로 수집하고 제공하여 마이그레이션을 계획할 때 보다 정확한 결정을 내릴 수 있도록 합니다.
4. 테스트 수행: 관리형 서비스로 마이그레이션하기 전에 서비스를 테스트합니다.
5. 셀프 호스팅 서비스 교체: 마이그레이션 계획을 사용하여 셀프 호스팅 서비스를 관리형 서비스로 교체합니다.
6. 모니터링 및 조정: 마이그레이션이 완료된 후 서비스를 지속적으로 모니터링하여 필요에 따라 조정하고 서비스를 최적화합니다.

리소스

관련 문서:

- AWS 클라우드 제품
- AWS 총 소유 비용(TCO) 계산기
- Amazon DocumentDB

- Amazon Elastic Kubernetes Service(EKS)
- Amazon Managed Streaming for Apache Kafka(Amazon MSK)

관련 동영상:

- AWS re:Invent 2021 - Cloud operations at scale with AWS Managed Services
- AWS re:Invent 2023 - Best practices for operating on AWS

SUS05-BP04 하드웨어 기반 컴퓨팅 액셀러레이터의 사용 최적화

가속 컴퓨팅 인스턴스의 사용을 최적화하여 워크로드의 물리적 인프라 요구를 줄입니다.

일반적인 안티 패턴:

- GPU 사용을 모니터링하지 않습니다.
- 특별히 구축된 인스턴스가 더 높은 성능, 더 낮은 비용 및 더 나은 와트당 성능을 제공함에도 불구하고 워크로드에 범용 인스턴스를 사용합니다.
- CPU 기반 대안을 사용하는 것이 더 효율적인 작업에 하드웨어 기반 컴퓨팅 액셀러레이터를 사용합니다.

이 모범 사례 확립의 이점: 하드웨어 기반 액셀러레이터의 사용을 최적화하여 워크로드의 물리적 인프라 요구를 줄일 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

높은 처리 용량이 필요한 경우, 그래픽 처리 장치(GPU) 및 필드 프로그래밍 가능 게이트 어레이(FPGA)와 같은 하드웨어 기반 컴퓨팅 액셀러레이터에 대한 액세스를 제공하는 가속 컴퓨팅 인스턴스 사용의 이점을 활용할 수 있습니다. 이러한 하드웨어 액셀러레이터는 CPU 기반 대안보다 더 효율적인 그래픽 처리 또는 데이터 패턴 일치와 같은 특정 기능을 수행합니다. 렌더링, 트랜스코딩, 기계 학습 등 많은 가속 워크로드는 리소스 사용 면에서 매우 가변적입니다. 필요한 시간 동안만 이 하드웨어를 실행하고 필요하지 않은 경우 자동화를 통해 이를 폐기하여 리소스 사용을 최소화합니다.

구현 단계

- 어떤 가속 컴퓨팅 인스턴스가 요구 사항을 해결할 수 있는지 파악합니다.

- 기계 학습 워크로드의 경우 AWS Trainium, AWS Inferentia 및 Amazon EC2 DL1과 같이 워크로드에 따라 다른 특별히 구축된 하드웨어의 장점을 활용합니다. Inf2 인스턴스와 같은 AWS Inferentia 인스턴스는 동급 Amazon EC2 인스턴스에 비해 최대 50% 더 우수한 와트당 성능을 제공합니다.
- 가속 컴퓨팅 인스턴스의 사용량 지표를 수집합니다. 예를 들어, Amazon CloudWatch에서 NVIDIA GPU 지표 수집에서와 같이 CloudWatch 에이전트를 사용하여 GPU에 대한 `utilization_gpu` 및 `utilization_memory`와 같은 지표를 수집할 수 있습니다.
- 코드, 네트워크 운영 및 하드웨어 액셀러레이터 설정을 최적화하여 기본 하드웨어가 반드시 제대로 활용되도록 해야 합니다.
  - GPU 설정 최적화
  - 딥 러닝 AMI에서 GPU 모니터링 및 최적화
  - Amazon SageMaker에서 딥 러닝 훈련의 GPU 성능 튜닝을 위한 I/O 최적화
- 최신 고성능 라이브러리 및 GPU 드라이버를 사용합니다.
- 자동화를 사용하여 사용하지 않는 GPU 인스턴스 사용을 해제합니다.

리소스

관련 문서:

- 가속화된 컴퓨팅
- 아키텍트를 시작하겠습니다! 맞춤형 칩과 액셀러레이터를 사용한 아키텍트
- 워크로드에 적합한 Amazon EC2 인스턴스 유형을 선택하려면 어떻게 해야 하나요?
- Amazon EC2 VT1 인스턴스
- Amazon SageMaker을 통해 컴퓨터 비전 추론을 위한 최고의 AI 액셀러레이터 및 모델 컴파일 선택

관련 동영상:

- AWS re:Invent 2021 - How to select Amazon EC2 GPU instances for deep learning
- AWS Online Tech Talks - Deploying Cost-Effective Deep Learning Inference
- AWS re:Invent 2023 - Cutting-edge AI with AWS and NVIDIA
- AWS re:Invent 2022 - [NEW LAUNCH!] Introducing AWS Inferentia2-based Amazon EC2 Inf2 instances
- AWS re:Invent 2022 - Accelerate deep learning and innovate faster with AWS Trainium
- AWS re:Invent 2022 - Deep learning on AWS with NVIDIA: From training to deployment

프로세스 및 문화

개발, 테스트 및 배포 방식을 변경하여 지속 가능성에 미치는 영향을 줄일 수 있는 기회를 모색합니다.

모범 사례

- SUS06-BP01 지속 가능성 개선을 신속하게 도입할 수 있는 방법 채택
- SUS06-BP02 워크로드를 최신 상태로 유지
- SUS06-BP03 구축 환경의 사용률 제고
- SUS06-BP04 테스트에 관리형 Device Farm 사용

SUS06-BP01 지속 가능성 개선을 신속하게 도입할 수 있는 방법 채택

잠재적인 개선 사항을 검증하고, 테스트 비용을 최소화하며, 경미한 개선 사항을 제공하는 방법과 프로세스를 채택합니다.

일반적인 안티 패턴:

- 지속 가능성을 위한 애플리케이션 검토는 프로젝트 시작 시 1번만 수행합니다.
- 릴리스 프로세스가 너무 번거로워 리소스 효율성을 위해 사소한 변경 사항을 적용할 수 없어 워크로드가 오래되었습니다.
- 지속 가능성을 위한 워크로드를 개선할 수 있는 메커니즘이 없습니다.

이 모범 사례 확립의 이점: 지속 가능성 개선 사항을 도입하고 추적하기 위한 프로세스를 수립함으로써 새로운 기능을 꾸준히 채택하고, 문제를 제거하고, 워크로드 효율성을 개선할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출 위험도: 중간

구현 가이드

잠재적인 지속 가능성 개선 사항을 프로덕션에 배포하기 전에 테스트하고 검증합니다. 개선 사항으로 실현될 미래의 잠재적 이익을 계산할 때 테스트 비용을 고려합니다. 적은 비용으로 경미한 개선 사항을 적용할 수 있는 테스트 방법을 개발합니다.

구현 단계

- 조직의 지속 가능성 목표 이해 및 전달: 탄소 저감 또는 수자원 스튜어드십과 같은 조직의 지속 가능성 목표를 이해합니다. 이러한 목표를 클라우드 워크로드의 지속 가능성 요구 사항으로 작성합니다. 이러한 요구 사항을 주요 이해관계자에게 전달합니다.

- 백로그에 지속 가능성 요구 사항 추가: 개발 백로그에 지속 가능성 개선을 위한 요구 사항을 추가합니다.
- 반복 및 개선: 반복적인 개선 프로세스를 사용하여 이러한 개선 사항을 식별 및 평가하고, 우선순위를 지정하고, 테스트 및 배포합니다.
- 최소 기능 제품(MVP)을 사용하여 테스트: 테스트의 비용과 환경에 미치는 영향을 줄이기 위해 최소 기능 구성 요소를 사용하여 잠재적인 개선 사항을 개발하고 테스트합니다.
- 프로세스 간소화: 개발 프로세스를 지속적으로 개선하고 간소화합니다. 예를 들어, 지속적 통합(CI) 및 지속적 전달(CD) 파이프라인을 통해 소프트웨어 전송 프로세스를 자동화함으로써 잠재적인 개선 사항을 테스트하고 배포하여 작업량을 줄이고 수동 프로세스로 인한 오류를 제한합니다.
- 교육 및 인식: 팀원들에게 지속 가능성에 대해 알리고 이들의 활동이 조직의 지속 가능성 목표에 미치는 영향을 교육할 수 있는 교육 프로그램을 운영합니다.
- 평가 및 조정: 개선의 영향을 지속적으로 평가하고 필요에 따라 조정합니다.

리소스

관련 문서:

- AWS가 지원하는 지속 가능성 솔루션
- Scalable agile development practices based on AWS CodeCommit(AWS CodeCommit에 기반한 확장 가능한 민첩한 개발 관행)

관련 동영상:

- AWS re:Invent 2023 - Sustainable architecture: Past, present, and future
- AWS re:Invent 2022 - Delivering sustainable, high-performing architectures
- AWS re:Invent 2022 - Architecting sustainably and reducing your AWS carbon footprint
- AWS re:Invent 2022 - Sustainability in AWS global infrastructure
- AWS re:Invent 2023 - What's new with AWS observability and operations

관련 예시:

- Well-Architected Lab - Turning cost & usage reports into efficiency reports(Well-Architected 실습 - 비용 및 사용량 보고서를 효율성 보고서로 전환)

SUS06-BP02 워크로드를 최신 상태로 유지

워크로드를 최신 상태로 유지하여 효율적인 기능을 채택하고, 문제를 제거하며, 워크로드의 전반적인 효율성을 개선합니다.

일반적인 안티 패턴:

- 시간이 지나면 현재 아키텍처가 정적 아키텍처가 되고 업데이트되지 않는다고 가정합니다.
- 업데이트된 소프트웨어 및 패키지가 워크로드와 호환되는지 평가하는 시스템 또는 정기적인 주기가 없습니다.

이 모범 사례 확립의 이점: 워크로드를 최신 상태로 유지하기 위한 프로세스를 확립하면 새로운 기능을 도입하고 문제를 해결하며 워크로드 효율성을 개선할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출되는 위험 수준: 낮음

구현 가이드

최신 운영 체제, 런타임, 미들웨어, 라이브러리 및 애플리케이션을 사용하면 워크로드 효율성을 개선하고 보다 효율적인 기술을 쉽게 채택할 수 있습니다. 공급업체가 자체적인 지속 가능성 목표를 충족할 수 있는 기능을 제공함에 따라, 최신 소프트웨어에는 워크로드의 지속 가능성에 미치는 영향을 보다 정확하게 측정하는 기능이 포함될 수도 있습니다. 정기적인 주기로 최신 기능 및 릴리스와 함께 워크로드를 최신 상태로 유지합니다.

구현 단계

- 프로세스 정의: 워크로드의 새로운 기능 또는 인스턴스를 평가하기 위한 프로세스 및 일정을 사용합니다. 클라우드에서 민첩성을 활용하여 새 기능이 다음 작업을 수행하도록 워크로드를 어떻게 개선할 수 있는지 신속하게 테스트합니다.
  - 지속 가능성 영향 줄이기
  - 성능 효율성을 높이기
  - 계획된 개선 작업의 장애 요인 제거
  - 지속 가능성에 미치는 영향을 측정 및 관리할 수 있는 능력 증진
- 인벤토리 작성: 워크로드 소프트웨어 및 아키텍처의 인벤토리를 작성하여 업데이트가 필요한 구성 요소를 식별합니다.
  - AWS Systems Manager 인벤토리를 사용하여 Amazon EC2 인스턴스에서 운영 체제(OS), 애플리케이션, 인스턴스 메타데이터를 수집하고 소프트웨어를 실행 중인 인스턴스, 소프트웨어 정책에 필요한 구성, 업데이트해야 할 인스턴스를 신속하게 파악합니다.

- 업데이트 절차 학습: 워크로드의 구성 요소를 업데이트하는 방법을 이해합니다.

Workload component	How to update
머신 이미지	EC2 Image Builder를 사용하여 Linux 또는 Windows 서버 이미지용 Amazon Machine Image(AMI) 업데이트를 관리합니다.
컨테이너 이미지	Amazon Elastic Container Registry (Amazon ECR)를 기존 파이프라인과 함께 사용하여 Amazon Elastic Container Service (Amazon ECS) 이미지를 관리합니다.
AWS Lambda	AWS Lambda에는 버전 관리 기능이 있습니다.

- 자동화 사용: 업데이트를 자동화하여 새 기능 배포에 필요한 작업 수준을 줄이고 수동 프로세스로 인한 오류를 제한합니다.
  - CI/CD를 사용하면 클라우드 애플리케이션과 관련된 AMI, 컨테이너 이미지 및 기타 아티팩트를 자동으로 업데이트할 수 있습니다.
  - AWS Systems Manager Patch Manager와 같은 도구를 사용하여 시스템 업데이트 프로세스를 자동화하고, AWS Systems Manager Maintenance Windows를 사용하여 활동을 예약할 수 있습니다.

리소스

관련 문서:

- AWS 아키텍처 센터
- What's New with AWS
- AWS 개발자 도구

관련 동영상:

- AWS re:Invent 2022 - Optimize your AWS workloads with best-practice guidance
- All Things Patch: AWS Systems Manager

관련 예시:

- Well-Architected 실습 - 인벤토리 및 패치 관리
- 실습: AWS Systems Manager

SUS06-BP03 구축 환경의 사용률 제고

워크로드를 개발, 테스트 및 구축하기 위한 리소스 활용도를 높입니다.

일반적인 안티 패턴:

- 구축 환경을 수동으로 프로비저닝하거나 종료합니다.
- 테스트, 구축 또는 릴리스 작업과 별개로 구축 환경을 계속 실행 중인 상태로 유지합니다(예: 개발 팀원이 근무 시간 외에 환경을 실행).
- 구축 환경에 리소스를 과도하게 프로비저닝합니다.

이 모범 사례 확립의 이점: 빌드 환경의 활용률을 높임으로써 클라우드 워크로드의 전반적인 효율성을 개선하는 동시에 효과적으로 개발, 테스트 및 구축 작업을 수행하도록 빌더에 리소스를 할당할 수 있습니다.

이 모범 사례를 따르지 않을 경우 노출되는 위험 수준: 낮음

구현 가이드

자동화와 코드형 인프라를 사용하여 필요 시 빌드 환경을 가동하고 사용하지 않을 때는 해당 환경을 종료합니다. 일반적인 패턴은 개발 담당 팀원의 근무 시간과 일치하는 가용 기간을 예약하는 것입니다. 테스트 환경은 프로덕션 구성과 매우 유사해야 합니다. 그러나 버스트 용량, Amazon EC2 스팟 인스턴스, 자동 확장 데이터베이스 서비스, 컨테이너 및 서버리스 기술과 함께 인스턴스 유형을 사용하여 개발 및 테스트 용량을 용도에 맞게 조정할 수 있는 기회를 찾아야 합니다. 테스트 요구 사항만 충족하도록 데이터 볼륨을 제한합니다. 테스트에서 프로덕션 데이터를 사용하는 경우 프로덕션 데이터를 공유하고, 데이터를 이동하지 않을 수 있는지 알아봅니다.

구현 단계

- 코드형 인프라 사용: 코드형 인프라를 사용하여 빌드 환경을 프로비저닝합니다.
- 자동화 사용: 자동화를 사용하여 개발 및 테스트 환경의 수명 주기를 관리하고 구축 리소스의 효율성을 극대화합니다.
- 활용도 극대화: 개발 및 테스트 환경의 활용도를 극대화하기 위한 전략을 사용합니다.

- 현실적인 최소한의 재현 환경을 사용하여 잠재적 개선 사항을 개발 및 테스트합니다.
- 가능한 경우 서버리스 기술을 사용합니다.
- 온디맨드 인스턴스를 사용하여 개발자 디바이스를 보완합니다.
- 버스트 용량이 포함된 인스턴스 유형, 스팟 인스턴스 및 기타 기술을 사용하여 사용량에 맞게 구축 용량을 조정합니다.
- 배스천 호스트 플릿을 배포하는 대신 네이티브 클라우드 서비스를 도입하여 보안 인스턴스 셸에 액세스합니다.
- 구축 작업에 따라 구축 리소스를 자동으로 확장합니다.

리소스

관련 문서:

- AWS Systems Manager Session Manager
- Amazon EC2 버스트 가능 성능 인스턴스
- AWS CloudFormation이란 무엇인가요?
- AWS CodeBuild란 무엇인가요?
- AWS의 인스턴스 스케줄러

관련 동영상:

- AWS re:Invent 2023 - Continuous integration and delivery for AWS

SUS06-BP04 테스트에 관리형 Device Farm 사용

관리형 Device Farm을 사용하여 대표적인 하드웨어 집합에서 새 기능을 효율적으로 테스트합니다.

일반적인 안티 패턴:

- 물리적 개별 디바이스에서 애플리케이션을 수동으로 테스트하고 배포합니다.
- 앱 테스트 서비스를 사용하여 실제 물리적 디바이스에서 앱(예: Android, iOS 및 웹 앱)을 테스트하고 상호 작용하지 않습니다.

이 모범 사례 확립의 이점: 관리형 Device Farm을 사용하여 클라우드 지원 애플리케이션을 테스트하면 다음과 같은 여러 이점을 얻을 수 있습니다.

- 여기에는 다양한 디바이스에서 애플리케이션을 테스트할 수 있는 보다 효율적인 기능이 포함되어 있습니다.
- 테스트를 위한 사내 인프라가 필요하지 않습니다.
- 비교적 널리 사용되지 않는 구형 하드웨어를 포함하여 다양한 디바이스 유형을 제공하므로 불필요한 디바이스 업그레이드가 필요하지 않습니다.

이 모범 사례를 따르지 않을 경우 노출되는 위험 수준: 낮음

구현 가이드

관리형 Device Farm을 사용하면 대표적인 하드웨어 집합에서 새 기능에 대한 테스트 프로세스를 간소화할 수 있습니다. 관리형 Device Farm은 사용 빈도가 낮은 오래된 하드웨어를 포함하여 다양한 디바이스 유형을 제공하며, 불필요한 디바이스 업그레이드로 인해 고객의 지속 가능성이 영향을 받지 않도록 합니다.

구현 단계

- 테스트 요구 사항 정의: 테스트 요구 사항 및 계획(예: 테스트 유형, 운영 체제 및 테스트 일정)을 정의합니다.
  - Amazon CloudWatch RUM을 사용하여 클라이언트 측 데이터를 수집 및 분석하고 테스트 플랜을 구체화할 수 있습니다.
- 관리형 디바이스 팜 선택: 테스트 요구 사항을 지원할 수 있는 관리형 디바이스 팜을 선택합니다. 예를 들어, AWS Device Farm을 사용하여 변경 사항이 대표적인 하드웨어 집합에 미치는 영향을 테스트하고 이해할 수 있습니다.
- 자동화 사용: 자동화와 지속적 통합(CI) 및 지속적 배포(CD)를 사용하여 테스트를 예약하고 실행합니다.
  - Integrating AWS Device Farm with your CI/CD pipeline to run cross-browser Selenium tests
  - Building and testing iOS and iPadOS apps with AWS DevOps and mobile services
- 검토 및 조정: 테스트 결과를 지속적으로 검토하고 필요한 개선을 수행합니다.

리소스

관련 문서:

- AWS Device Farm device list(AWS Device Farm 디바이스 목록)
- CloudWatch RUM 대시보드 보기

관련 예시:

- Android용 AWS Device Farm 샘플 앱
- iOS용 AWS Device Farm 샘플 앱
- AWS Device Farm용 Appium Web 테스트

관련 동영상:

- AWS re:Invent 2023 - Improve your mobile and web app quality using AWS Device Farm
- AWS re:Invent 2021 - Amazon CloudWatch RUM을 통해 최종 사용자 인사이트로 애플리케이션 최적화

결론

정부 규제, 경쟁 우위, 고객, 직원 및 투자자 요구의 변화에 대응하여 지속 가능성 목표를 수립하는 조직이 증가하고 있습니다. CTO, 아키텍트, 개발자, 운영 팀의 구성원은 조직의 지속 가능성 목표에 직접 기여할 수 있는 방법을 찾고 있습니다. 이 설계 원칙과 AWS 서비스에서 지원하는 모범 사례를 사용하면 보안, 비용, 성능, 안정성, 운영 우수성과 AWS 클라우드 워크로드의 지속 가능성 결과의 균형을 이루는 정보에 입각한 의사 결정을 내릴 수 있습니다. 리소스 사용을 줄이고 워크로드 전반의 효율성을 높이기 위한 모든 조치가 환경 영향 감소와 조직의 광범위한 지속 가능성 목표에 기여합니다.

기여자

이 문서를 작성하는 데 도움을 주신 분들입니다.

- Sam Mokhtari, Senior Efficiency Lead Solutions Architect, Amazon Web Services
- Brendan Sisson, Principal Sustainability Solutions Architect, Amazon Web Services
- Margaret O'Toole, Sustainability Tech Leader, Amazon Web Services
- Steffen Grunwald, Principal Sustainability Solutions Architect, Amazon Web Services
- Ryan Eccles, Amazon Principal Engineer, Sustainability
- Rodney Lester, Principal Architect, Amazon Web Services
- Adrian Cockcroft, VP Sustainability Architecture, Amazon Web Services
- Ian Meyers, Director of Technology, Solutions Architecture, Amazon Web Services

추가 자료

자세한 내용은 다음을 참조하세요.

- AWS Well-Architected
- AWS 아키텍처 센터
- Sustainability in the Cloud(클라우드의 지속 가능성)
- AWS가 지원하는 지속 가능성 솔루션
- 기후 협약
- UN 지속 가능 개발 목표
- 온실 가스 협약

문서 개정

이 백서의 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하세요.

변경 사항	설명	날짜
백서 업데이트	모범 사례를 새로운 구현 가이드와 함께 업데이트했습니다.	June 27, 2024
위험 수준 업데이트 됨	모범 사례 위험 수준에 마이너 업데이트를 했습니다.	October 3, 2023
모범 사례 지침 업데이트	다음 영역에 대한 새로운 지침으로 모범 사례가 업데이트되었습니다. 수요에 맞춘 조정, 소프트웨어 및 아키텍처, 데이터및 하드웨어 및 서비스.	July 13, 2023
새 프레임워크에 맞게 업데이트되었습니다.	모범 사례를 권장 가이드와 함께 업데이트하고 새로운 모범 사례를 추가했습니다.	April 10, 2023
백서 업데이트	모범 사례를 새로운 구현 가이드와 함께 업데이트했습니다.	December 15, 2022
백서 업데이트	모범 사례를 확대하고 개선 계획을 추가했습니다.	October 20, 2022
최초 게시	지속 가능성 원칙 - AWS Well-Architected Framework를 게시했습니다.	December 2, 2021

고지 사항

고객은 본 문서에 포함된 정보를 독자적으로 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.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the AWS 용어집 Reference.