Amazon Neptune용 AWS Well-Architected 프레임워크 적용

AWS 권장 가이드

AWS 권장 가이드: Amazon Neptune용 AWS Well-Architected 프레임워크 적용

Table of Contents

Amazon Neptune용 AWS Well-Architected 프레임워크 적용

Amazon Web Services(기여자)

2026년 1월(문서 기록)

Amazon Neptune을 사용하여 Amazon Web Services(AWS)에서 그래프 기반 솔루션을 빌드할 수 있습니다. 이 가이드는 Neptune 배포를 계획할 때 AWS Well-Architected Framework 원칙을 적용하기 위한 권장 가이드를 제공합니다.

AWS Well-Architected Framework를 사용하면 다양한 애플리케이션 및 워크로드를 위한 안전하고 성능이 뛰어나며 복원력이 뛰어나고 효율적인 인프라를 구축할 수 있습니다. 또한 키텍처를 평가하고 확장 가능한 설계를 구현할 수 있는 일관된 접근 방식을 제공합니다.

AWS Well-Architected 프레임워크는 다음 6가지 원칙을 중심으로 구축되었습니다.

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

이 가이드는 AWS Well-Architected Framework 설계 원칙 및 모범 사례의 정보와에 Neptune을 배포할 때 염두에 두어야 할 고려 사항을 제공합니다 AWS.

대상 독자

이 가이드는 AWS에서 그래프를 사용하는 솔루션을 설계하고 구현하는 데이터 엔지니어, 솔루션 아키텍트 및 데이터 분석가를 대상으로 합니다.

목표

이 가이드는 사용자와 조직이 다음을 수행하는 데 도움이 될 수 있습니다.

- 사용 사례 및 쿼리 패턴에 따라 지원되는 배포 옵션 및 쿼리 언어 중에서 선택합니다.

- 복원력과 보안을 개선하는 데 도움이 되는 AWS Well-Architected 설계 패턴을 따릅니다.
- 최적의 성능과 비용 절감을 위해 쿼리를 설계합니다.
- 프로덕션 환경에서 Neptune 클러스터를 관리할 때 운영 효율성을 높이는 방법에 대해 알아봅니다.

운영 우수성 요소

AWS Well-Architected Framework의 운영 우수성 원칙은 시스템을 실행 및 모니터링하고 프로세스와 절차를 지속적으로 개선하는 데 중점을 둡니다. 여기에는 효과적인 개발 및 워크로드 실행을 지원하고, 작업에 대한 인사이트를 얻으며, 지원 프로세스 및 절차를 지속적으로 개선하여 비즈니스 가치를 전달할 수 있는 능력이 포함됩니다. 사람의 개입 없이 대부분의 문제를 감지하고 해결하는 자가 복구 워크로드를 통해 운영 복잡성을 줄일 수 있습니다. 이 섹션에 설명된 모범 사례를 따르면 이 목표를 달성할 수 있습니다. Amazon Neptune 지표, API 및 메커니즘을 사용하여 워크로드가 예상 동작과 다를 때 적절하게 대응합니다.

운영 우수성 원칙에 대한 이 논의는 다음 핵심 영역에 중점을 둡니다.

- 코드형 인프라(IaC)
- 변경 관리
- 복원력 전략
- 인시던트 관리
- 규정 준수를 위한 감사 보고
- 로깅 및 모니터링

IaC 접근 방식을 사용하여 배포 자동화

IaC를 사용하여 Neptune에서 배포를 자동화하는 모범 사례로 다음이 포함됩니다.

- 가능하면 코드형 인프라(IaC)를 적용하여 Neptune 클러스터를 배포합니다. 일관된 환경 구성을 위해 AWS CloudFormation 템플릿, AWS Cloud Development Kit (AWS CDK) 또는 HashiCorp Terraform을 사용하여 클러스터에 필요한 모든 리소스를 생성합니다.
- 인스턴스 크기 조정, 읽기 복제본 추가 또는 제거, 가능하면 글로벌 테이블에서 수동 장애 조치 수행과 같은 Neptune 운영 절차를 자동화합니다.
- 연결 문자열을 클라이언트 외부에 저장합니다. 추출, 전환, 적재(ETL) 프로세스를 사용하여 블루/그린 배포 전략, 재해 복구(DR) 및 가동 중지 시간이 거의 없는 새 클러스터로의 마이그레이션을 용이하게 합니다. 연결 문자열은 AWS Secrets Manager, Amazon DynamoDB 또는 동적으로 변경할 수 있는 모든 위치에 저장할 수 있습니다.
- 태그를 사용하여 Neptune 리소스에 메타데이터를 추가하고 태그를 기반으로 사용량을 추적합니다. 자세한 내용은 Tagging Amazon Neptune Resources를 참조하세요.

되돌릴 수 있는 소규모 변경 자주 적용

다음 권장 사항은 복잡성을 최소화하고 워크로드 중단 가능성을 줄이기 위해 작고 되돌릴 수 있는 변경 사항에 중점을 둡니다.

- GitHub 또는 GitLab과 같은 소스 제어 서비스에 IaC 템플릿 및 스크립트를 저장합니다.

> **Important**
>
> 소스 제어에 AWS 자격 증명을 저장하지 마십시오.

- IaC 배포에서 AWS CodePipeline 또는 AWS CodeBuild와 같은 지속적 통합 및 지속적 전송(CI/CD) 서비스를 사용해야 합니다. 이러한 서비스는 프로덕션 Amazon Neptune 클러스터에 영향을 미치기 전에 임시 Neptune 클러스터가 포함된 비프로덕션 환경에서 코드를 컴파일, 테스트 및 배포합니다.
- 인프라 및 애플리케이션 쿼리를 프로덕션에 배포하기 전에 더 하위 환경에서 테스트합니다. 그러면 중단 가능성을 최소화하고 워크로드 및 규모에 맞게 원활하게 작동할 수 있습니다.

실패 예상

자가 복구 인프라는 장애를 예상하고 개입 없이 문제를 해결하려고 시도하며 운영 우수성을 보여줍니다. 다음 권장 사항은 Neptune을 사용하여 이러한 성숙도를 달성하는 데 도움이 됩니다.

- Amazon CloudWatch 지표를 사용하여 DB 인스턴스의 CPU 및 메모리 사용량을 모니터링하고 사용 패턴을 이해하는 모니터링 계획을 생성합니다. 애플리케이션 로그에 있는 주요 지표 및 Neptune 클라이언트 응답에 대한 CloudWatch 대시보드 및 경보를 생성합니다. CPU 사용률이 높거나 낮은 지표에 대한 자세한 내용은 Neptune 설명서의 Using CloudWatch to monitor DB instance performance in Neptune을 참조하세요.

  쿼리에 out-of-memory 예외가 자주 발생하는 경우 쿼리가 통과하는 총 노드 수를 줄이거나 RAM-to-CPU 비율이 높은 X2 패밀리의 인스턴스를 사용해 보세요.
- 알림을 설정하여 Neptune 클러스터의 상태를 모니터링합니다. 예를 들어 `BufferCacheHitRatio`는 지속적으로 높아야 하지만(99.9% 초과) `MainRequestQueuePendingRequests`는 지속적으로 낮아야 합니다(이상적으로는 0이지만 요구 사항 및 지연 시간 허용치에 따라 다름).

- 읽기 복제본을 사용하여 Neptune 내에서 고가용성을 달성하는 것이 좋습니다. 장애 조치 이벤트 중에 항상 읽기 쿼리를 지원할 수 있도록 인스턴스를 상시 가동하려면 라이터 인스턴스와 다른 가용 영역에 읽기 복제본이 두 개 이상 있어야 합니다.
- 사용률 지표를 기반으로 읽기 복제본의 크기를 자동으로 조정합니다. 자세한 내용은 Auto-scaling the number of replicas in an Amazon Neptune DB cluster를 참조하세요.
- DB 인스턴스의 장애 조치를 테스트하여 사용 사례 절차에 소요되는 시간을 파악하십시오.
- 애플리케이션에 완전한 AWS 리전 중단이 지속되어야 하는 경우 글로벌 데이터베이스를 DR 계획의 일부로 사용하는 것이 좋습니다.

모든 운영 장애로부터 학습

자가 복구 인프라는 드문 문제가 발생하거나 응답이 원하는 만큼 효과적이지 않을 때 반복으로 발전하는 장기적인 노력을 보여줍니다. 다음 사례를 채택하면 다음과 같은 목표에 집중합니다.

- 모든 장애로부터 학습하여 개선을 주도합니다.
- 조직과 여러 팀에서 학습한 내용을 공유합니다. 조직 내 여러 팀이 Neptune을 사용하는 경우 공통 채팅룸 또는 사용자 그룹을 생성하여 학습 내용과 모범 사례를 공유합니다.

로깅 기능을 사용하여 무단 또는 비정상적인 활동 모니터링

비정상적인 성능 및 활동 패턴을 관찰하려면 Amazon CloudWatch Logs에 로그를 저장합니다. 다음 모범 사례를 고려하세요.

- 느린 쿼리 로깅을 비활성화합니다. 로그를 정기적으로 검토하고 특정 쿼리가 느린 이유를 진단합니다. Gremlin, SPARQL 또는 openCypher에 대한 Neptune 설명 및 프로파일 엔드포인트를 사용하여 이러한 쿼리가 느린 이유를 파악할 수 있습니다.
- Neptune 감사 로그를 활성화하고 로그에 무단 액세스 또는 이상이 있는지 정기적으로 검토합니다.
- 느린 쿼리 로깅 또는 감사 로깅을 사용하는 경우 CloudWatch Logs에 대한 게시를 활성화합니다. 이렇게 하면 인스턴스의 디스크 공간 부족을 방지할 수 있습니다. Neptune 인스턴스는 로그 스토리지 용량을 제한하며 로그 공간이 초과되면 이전 로그 파일을 덮어씁니다. CloudWatch Logs는 로그의 장기 보존을 지원합니다. CloudWatch Logs의 향상된 모니터링 기능은 로그를 쿼리하고 문제를 진단하는 기능을 개선합니다.
- 감사 로그에 대한 더 나은 분석 도구를 용이하게 하기 위해 CloudWatch Logs의 로그 그룹에 감사 로그 데이터를 게시하도록 Neptune DB 클러스터를 구성할 수 있습니다. CloudWatch Logs로 로그 데이터에 대한 실시간 분석을 수행하고 CloudWatch를 사용하여 경보를 생성하고 지표를 보고

CloudWatch Logs를 사용하여 로그 레코드를 내구성이 높은 스토리지에 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs에 Neptune Logs 게시를 참조하세요.

- Neptune은 AWS CloudTrail을 사용하여 컨트롤 플레인 작업의 로깅을 지원합니다. 자세한 내용은 를 사용하여 Amazon Neptune API 호출 로깅을 참조하세요 AWS CloudTrail.

보안 요소

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드 보안 - AWS 서비스 에서 실행되는 인프라를 보호할 AWS 책임이 있습니다 AWS 클라우드. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램의 일환으로 AWS 보안의 효과를 정기적으로 테스트하고 확인합니다. Amazon Neptune에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램별 범위 내AWS 서비스를 참조하세요.
- 클라우드의 보안 - 사용자의 책임은 AWS 서비스 사용하는에 따라 결정됩니다. 또한 귀하는 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS Shared Responsibility Model and GDPR 블로그 게시물을 참조하세요.

보안 원칙은 Neptune을 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 Neptune을 구성하는 방법을 보여줍니다. 또한 Neptune 리소스를 모니터링하고 보호하는 데 도움이 AWS 서비스 되는 다른를 사용하는 방법을 알아봅니다.

보안 원칙에는 다음과 같은 주요 핵심 영역이 포함됩니다.

- 데이터 보안
- 네트워크 보안
- 인증 및 권한 부여

데이터 보안 구현

데이터 유출 및 위반은 고객을 위험에 빠뜨리고 회사에 상당한 부정적인 영향을 미칠 수 있습니다. 다음 모범 사례는 우발적이고 악의적인 노출로부터 고객 데이터를 보호하는 데 도움이 됩니다.

- 클러스터 이름, 태그, 파라미터 그룹, AWS Identity and Access Management (IAM) 역할 및 기타 메타데이터에는 기밀 또는 민감한 정보가 포함되어서는 안 됩니다. 해당 데이터가 결제 또는 진단 로그에 표시될 수 있기 때문입니다.
- Neptune에 데이터로 저장된 외부 서버에 대한 URI 또는 링크에는 요청을 검증하기 위한 자격 증명 정보가 포함되어서는 안 됩니다.
- Neptune 암호화 인스턴스는 기본 스토리지에 대한 무단 액세스로부터 데이터의 보안을 유지할 수 있도록 지원함으로써 데이터 보호 추가 계층을 제공합니다. 클라우드에 배포된 애플리케이션의 데이터 보안을 향상하기 위해 Neptune 암호화를 사용할 수 있습니다. 또한 이를 통해 저장 데이터의 암호화에 대한 규정 준수 요구 사항을 충족할 수 있습니다.

  새 Neptune DB 인스턴스에 대한 암호화를 활성화하려면 Neptune 콘솔의 암호화 활성화 섹션에서 예((기본적으로 선택됨)를 선택하거나 CloudFormation에서 AWS::Neptune::DBCluster::StorageEncrypted 속성을 설정합니다. 암호화가 활성화된 경우 Neptune은 기본적으로 Amazon Relational Database Service(Amazon RDS) AWS 관리형 키를 사용하거나 고객 관리형 키를 생성할 수 있습니다. Neptune DB 인스턴스 생성에 대한 자세한 내용은 Creating a new Neptune DB cluster를 참조하세요. 자세한 내용은 Encrypting Neptune Resources at Rest를 참조하세요. 자동화된 스냅샷 및 수동 스냅샷은 Neptune 클러스터에 대해 선택한 것과 동일한 암호화를 사용합니다.
- SPARQL 및 openCypher 언어를 사용하는 경우 적절한 입력 검증 및 파라미터화 기법을 사용하여 SQL 인젝션 및 기타 형태의 공격을 방지합니다. 사용자가 제공한 입력과 문자열 연결을 사용하는 쿼리를 구성하지 마세요. 파라미터화된 쿼리 또는 준비된 명령문을 사용하여 입력 파라미터를 그래프 데이터베이스에 안전하게 전달합니다. 자세한 내용은 Examples of openCypher parameterized queries 및 SPARQL Injection Defence를 참조하세요.
- Gremlin 언어의 경우 잠재적인 인젝션 문제를 방지하려면 문자열 기반 Gremlin 스크립트를 직접 전달하는 대신 Gremlin 언어 변형을 사용합니다.

네트워크 보안

Amazon Neptune DB 클러스터는 AWS의 가상 프라이빗 클라우드(VPC)에서만 생성할 수 있습니다. Neptune 1.4.6.0까지 Neptune DB 클러스터의 엔드포인트는 해당 VPC 내에서만 액세스할 수 있었습니다. Neptune 1.4.6.0 이상부터 Neptune 인스턴스는 인터넷을 통해 공개적으로 액세스할 수 있도록 구성할 수 있습니다. 개발자를 위해 Neptune에 대한 간소화된 액세스를 활성화하려면 비프로덕션 환경에서만이 기능을 사용하는 것이 가장 좋습니다(퍼블릭 액세스 가능성을 활성화하려면 항상에서 IAM 인증이 필요함). 퍼블릭 액세스가 활성화된 경우 데이터베이스 포트에 대한 인바운드 보안 그룹 규칙을 알려진 IP 주소 트래픽으로만 설정하는 것이 좋습니다. 프로덕션 환경 또는 민감한 데이터가 포함된 클러스터에서는 퍼블릭 액세스를 허용하지 않고 Neptune DB 클러스터가 위치한 VPC에 대한 액세스를

제한하여 Neptune 데이터를 보호합니다. 자세한 내용은 Connecting to your Amazon Neptune graph를 참조하세요.

전송 중 데이터를 보호하기 위해 Neptune은 보안 프로토콜 및 암호를 사용하여 HTTPS를 통해 모든 인스턴스 또는 클러스터 엔드포인트에 SSL 연결을 적용합니다. Neptune은 Neptune DB 인스턴스에 대한 SSL 인증서를 제공합니다. Neptune SSL 인증서는 클러스터 엔드포인트, 리더 엔드포인트 및 인스턴스 엔드포인트 호스트 이름만 지원합니다.

로드 밸런서 또는 프록시 서버(예: HAProxy)를 사용하는 경우 SSL 종료를 사용하고 프록시 서버에 자체 SSL 인증서가 있어야 합니다. 제공된 SSL 인증서가 프록시 서버 호스트 이름과 일치하지 않으므로 SSL 패스스루가 작동하지 않습니다. SSL을 사용하여 Neptune 엔드포인트에 연결하는 방법에 대한 자세한 내용은 Using the HTTP REST endpoint to connect to a Neptune DB instance를 참조하세요.

인증 및 권한 부여 구현

Neptune DB 클러스터 및 DB 인스턴스에서 Neptune 관리 작업을 수행할 수 있는 사용자를 제어하려면 IAM 데이터베이스 인증을 활성화하고 IAM 자격 증명을 사용합니다. IAM 자격 증명을 사용하여 AWS 에 연결할 때, IAM 역할은 Neptune 관리 작업을 수행하는 데 필요한 권한을 부여하는 IAM 정책을 보유하고 있어야 합니다. 태스크를 완료하는 데 필요한 권한만 부여하는 최소 권한 원칙을 따라야 합니다. 자세한 내용은 Using different kinds of IAM policies for controlling access to Neptune 및 IAM Authentication Using Temporary Credentials를 참조하세요.

Neptune 클러스터에 연결하고 데이터를 쿼리할 수 있는 사용자를 제어하려면 IAM을 사용하여 Neptune DB 인스턴스 또는 DB 클러스터에 인증할 수 있습니다. Neptune DB 클러스터에서 IAM 인증을 활성화하는 경우 DB 클러스터에 액세스하는 모든 사용자는 먼저 인증을 받아야 합니다. IAM 인증을 활성화하는 단계에 대한 자세한 내용은 Enabling IAM database authentication in Neptune을 참조하세요.

IAM 데이터베이스 인증이 활성화되어 있으면 AWS Signature Version 4를 사용하여 각 요청에 서명해야 합니다. IAM 인증이 활성화된 모든 Neptune 엔드포인트에 서명된 요청을 보내는 방법을 이해하려면 Connecting and Signing with AWS Signature Version 4를 참조하세요. awscurl과 같은 많은 라이브러리와 도구는 이미 AWS 서명 버전 4를 지원합니다.

다른와 상호 작용하기 위해 AWS 서비스 Amazon Neptune은 IAM 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Neptune에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Neptune에서 사전 정의하며 서비스에서 다른 AWS 서비스 자동으로 직접 호출하기 위해 필요한 모든 권한을 포함합니다. 자세한 내용은 Using Service-Linked Roles for Neptune을 참조하세요.

신뢰성 요소

신뢰성 원칙은 워크로드가 의도한 기능을 예상대로 올바르고 일관되게 수행할 수 있는 능력을 포함합니다. 여기에는 전체 수명 주기에 걸쳐 워크로드를 운영 및 테스트할 수 있는 기능이 포함됩니다.

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

신뢰성 원칙은 다음 핵심 영역에 중점을 둡니다.

- 서비스 할당량 및 배포 패턴을 포함한 워크로드 아키텍처
- 변경 관리
- 장애 관리

Neptune 서비스 할당량 이해

중국 및 GovCloud(할당량이 64TiB임)를 제외한 지원되는 모든 AWS 리전 에서 Neptune 클러스터 볼륨은 최대 128테라바이트(TiB) 크기까지 늘어날 수 있습니다.

128TiB 할당량은 그래프에 약 2,000~4,000억 개의 객체를 저장하기에 충분합니다. 레이블이 지정된 속성 그래프(LPG)에서 객체는 노드, 엣지 또는 노드나 엣지의 속성입니다. 리소스 설명 프레임워크(RDF) 그래프에서 객체는 쿼드입니다.

Neptune Serverless 클러스터의 경우 Neptune 용량 단위(NCU)의 최소 및 최대 수를 모두 설정합니다. 각 NCU는 2기가바이트(GiB)의 메모리와 관련 vCPU 및 네트워킹으로 구성됩니다. 최소 및 최대 NCU 값이 클러스터의 모든 서버리스 인스턴스에 적용됩니다. 설정할 수 있는 최댓값은 128.0NCU이고, 가장 낮은 최솟값은 1.0NCU입니다. Amazon CloudWatch 지표 `ServerlessDatabaseCapacity` 및 `NCUUtilization`을 관찰하여 일반적으로 실행되는 범위를 캡처하고 해당 범위 내에서 원치 않는 동작이나 비용을 상호 연관시켜 애플리케이션에 가장 적합한 NCU 범위를 최적화합니다. 많은 워크로드에서 1.0 NCU는 시작점보다 너무 낮으며 일정 기간 동안 활동이 없으면 신뢰할 수 없는 동작이 발생합니다. 워크로드가 충분히 빠르게 조정되지 않는 경우 확장하는 동안 초기 급증에 충분한 처리를 제공하도록 최소 NCU를 늘립니다.

각 AWS 계정 에는 생성할 수 있는 데이터베이스 리소스 수에 대한 각 리전의 할당량이 있습니다. 이러한 리소스에는 DB 인스턴스 및 DB 클러스터가 포함됩니다. 리소스에 대한 제한에 도달하면 해당 리소스 생성을 위한 추가 호출이 예외와 함께 실패합니다. 일부 할당량은 요청에 따라 늘

릴 수 있는 소프트 할당량입니다. Amazon Neptune과 Amazon RDS, Amazon Aurora 및 Amazon DocumentDB(MongoDB 호환) 사이에서 공유되는 할당량 목록과 사용 가능한 경우 할당량 증가를 요청하는 링크는 Amazon RDS의 할당량을 참조하세요.

Neptune 배포 패턴 이해

Neptune DB 클러스터에는 기본 DB 인스턴스 1개와 최대 15개의 Neptune 복제본이 있습니다. 기본 DB 인스턴스는 읽기 및 쓰기 작업을 지원하고, 클러스터 볼륨에 대한 모든 데이터 수정을 수행합니다. Neptune 복제본은 기본 DB 인스턴스와 동일한 스토리지 볼륨에 연결되며 읽기 작업만 지원합니다. Neptune 복제본은 기본 DB 인스턴스에서 읽기 워크로드를 오프로드할 수 있습니다.

고가용성을 달성하려면 읽기 복제본을 사용합니다. 읽기 복제본이 기본 인스턴스의 장애 조치 대상 역할을 하기 때문에 여러 가용 영역에서 하나 이상의 읽기 복제본 인스턴스를 사용 가능하면 가용성이 향상될 수 있습니다. 즉, 라이터 인스턴스에서 장애가 발생하면 Neptune은 읽기 복제본 인스턴스를 기본 인스턴스로 승격합니다. 이 경우 승격된 인스턴스가 재부팅되는 동안 잠시 중단(보통 30초 미만)이 발생하며, 이 동안 기본 인스턴스에 대한 읽기 및 쓰기 요청이 예외적으로 실패합니다. 최고의 신뢰성을 위해 서로 다른 가용 영역에 있는 두 개의 읽기 복제본을 고려합니다. 가용 영역 1의 기본 인스턴스가 오프라인 상태가 되면 가용 영역 2의 인스턴스가 기본 인스턴스로 승격되지만 이 경우 쿼리를 처리할 수 없습니다. 따라서 전환 중에 읽기 쿼리를 처리하려면 가용 영역 3의 인스턴스가 필요합니다.

Neptune Serverless를 사용하는 경우 모든 가용 영역의 리더 및 라이터 인스턴스는 데이터베이스 로드에 따라 서로 독립적으로 스케일 업 및 스케일 다운됩니다. 리더 인스턴스의 승격 계층을 0 또는 1로 설정하여 라이터 인스턴스의 용량과 함께 스케일 업 및 스케일 다운할 수 있습니다. 이렇게 하면 언제든지 현재 워크로드를 인계받을 수 있습니다.

애플리케이션에 전 세계 공간이 있거나 다중 리전 장애 조치가 필요한 경우 Neptune 글로벌 데이터베이스를 사용하는 것이 좋습니다. Amazon Neptune 글로벌 데이터베이스는 여러 개에 걸쳐 AWS 리전 있으므로 지연 시간이 짧은 전역 읽기를 활성화하고 드물게 중단이 전체에 영향을 미치는 경우 빠른 복구를 제공합니다 AWS 리전. Neptune 글로벌 데이터베이스는 한 리전의 기본 DB 클러스터와 다른 리전의 최대 5개의 보조 DB 클러스터로 구성됩니다.

Neptune 클러스터 관리 및 규모 조정

Neptune 오토 스케일링을 사용하여 CPU 사용률 임계치에 따라 연결성 및 워크로드 요구 사항을 충족하도록 DB 클러스터의 Neptune 복제본 수를 자동으로 조정할 수 있습니다. 오토 스케일링을 사용하면 Neptune DB 클러스터가 갑작스러운 워크로드 증가를 처리할 수 있습니다. 워크로드가 감소하면 오토 스케일링은 불필요한 복제본을 제거하여 미사용 용량에 대한 비용을 지불하지 않도록 합니다. 새 인스

턴스 시작에는 최대 15분이 걸릴 수 있으므로 오토 스케일링만으로는 수요의 급격한 변화를 위한 충분한 솔루션이 아닙니다.

오토 스케일링은 이미 기본 라이터 인스턴스와 하나 이상의 읽기 복제본 인스턴스가 있는 Neptune DB 클러스터에서만 사용할 수 있습니다(see Amazon Neptune DB Clusters and Instances 참조). 또한 클러스터의 모든 읽기 복제본 인스턴스는 사용 가능한 상태여야 합니다. 읽기 복제본이 사용 가능하지 않은 상태인 경우 Neptune 오토 스케일링은 클러스터의 모든 읽기 복제본을 사용할 수 있을 때까지 아무 작업도 수행하지 않습니다.

수요가 빠르게 변하는 경우 서버리스 인스턴스를 사용하는 방법을 고려합니다. 서버리스 인스턴스는 단기간에 수직적 스케일링을 수행할 수 있는 반면, 오토 스케일링은 장기간에 수평적 스케일링을 수행합니다. 이 구성은 서버리스 인스턴스에서 수직적 스케일링을 수행하는 반면 오토 스케일링은 새 읽기 복제본을 인스턴스화하여 단일 서버리스 인스턴스의 최대 용량을 초과하는 워크로드를 처리하기 때문에 최적의 확장성을 제공합니다. Amazon Neptune Serverless의 용량 조정에 대한 자세한 내용은 Capacity scaling in a Neptune Serverless DB cluster를 참조하세요.

예측 가능한 시간에 조정 요구 사항을 변경해야 하는 경우 최소 인스턴스, 최대 인스턴스 및 임계치에 대한 변경을 예약하여 이러한 전환 요구 사항을 더 잘 처리할 수 있습니다. 필요할 때 해당 인스턴스가 온라인 상태가 될 수 있도록 스케일 아웃 이벤트를 최소 15분 전에 예약해야 합니다.

DB 파라미터 그룹에서 파라미터를 사용하여 Amazon Neptune에서 데이터베이스 구성을 관리합니다. 파라미터 그룹은 하나 이상의 DB 인스턴스에 적용되는 엔진 구성 값의 컨테이너 역할을 합니다. 파라미터 그룹에서 클러스터 파라미터를 수정할 때 정적 파라미터와 동적 파라미터의 차이와 적용 방법 및 시기를 이해합니다. 상태 엔드포인트를 사용하여 현재 적용된 구성을 확인합니다.

백업 및 장애 조치 이벤트 관리

Neptune은 클러스터 볼륨을 자동으로 백업한 후 백업 보존 기간 동안 백업된 데이터를 유지합니다. Neptune 백업은 연속적으로 또는 증분식으로 이루어지기 때문에 백업 보존 기간 내에 어떤 시점으로든 신속하게 복구가 가능합니다. DB 클러스터를 생성 또는 수정할 때 1일에서 35일 사이의 백업 보존 기간을 지정할 수 있습니다.

백업 보존 기간을 넘겨서 백업을 유지하려는 경우 클러스터 볼륨에서 데이터 스냅샷을 생성할 수도 있습니다. 스냅샷을 저장하면 Neptune 표준 스토리지 비용이 발생합니다.

DB 클러스터의 Amazon Neptune 스냅샷을 생성하면 Neptune은 개별 인스턴스가 아닌 모든 데이터를 백업하여 클러스터의 스토리지 볼륨 스냅샷을 생성합니다. 이 DB 클러스터 스냅샷에서 복원하여 추후 새로운 DB 클러스터를 생성할 수 있습니다. DB 클러스터를 복원할 때 복원 원본으로 사용할 DB 클러스터 스냅샷의 이름을 제공한 다음, 이 복원 작업에서 생성되는 새 DB 클러스터의 이름을 제공합니다.

시스템이 장애 조치 이벤트에 어떻게 응답하는지 테스트합니다. Neptune API를 사용하여 장애 조치 이벤트를 강제로 실행합니다. 장애 조치로 재부팅은 DB 인스턴스 결함을 시뮬레이션하여 테스트하거나, 장애 조치 이후 원래 가용 영역으로 작업을 복구할 때 유용한 기능입니다. 다중 AZ 배포에 대한 자세한 내용은 다중 AZ 배포 구성 및 관리를 참조하세요. DB 라이터 인스턴스를 재부팅하면 예비 복제본으로 장애 조치가 발생합니다. Neptune 복제본을 재부팅하면 장애 조치가 시작되지 않습니다.

신뢰성을 위해 클라이언트를 설계합니다. 장애 조치 이벤트 중에 동작을 테스트합니다. 지수 백오프 로직을 사용하여 클라이언트에서 재시도 로직을 구현합니다. 이 로직을 구현하는 코드 예제는 설명서의 AWS Lambda Amazon Neptune 함수 예제에서 확인할 수 있습니다.

여러 데이터베이스 엔진에 적용되는 일반적인 백업 요구 사항이 있는 경우 AWS Backup을 사용하는 방법을 고려합니다.

성능 효율성 요소

AWS Well-Architected Framework의 성능 효율성 원칙은 데이터를 수집하거나 쿼리하는 동안 성능을 최적화하는 방법에 중점을 둡니다. 성능 최적화는 다음과 같은 점진적이고 지속적인 프로세스입니다.

- 비즈니스 요구 사항 확인
- 워크로드 성능 측정
- 성능이 낮은 구성 요소 식별
- 비즈니스 요구 사항에 맞게 구성 요소 조정

성능 효율성 원칙은 사용할 올바른 그래프 데이터 모델 및 쿼리 언어를 식별하는 데 도움이 될 수 있는 사용 사례별 지침을 제공합니다. 또한 Amazon Neptune으로 데이터를 수집하고 여기에서 데이터를 소비할 때 따라야 할 모범 사례도 포함되어 있습니다.

성능 효율성 원칙은 다음 핵심 영역에 중점을 둡니다.

- 그래프 모델링
- 쿼리 최적화
- 클러스터 적정 규모 조정
- 쓰기 테이블 최적화

그래프 모델링 이해

레이블이 지정된 속성 그래프(LPG) 모델과 리소스 설명 프레임워크(RDF) 모델의 차이를 이해합니다. 대부분의 경우 이는 선호도 문제입니다. 그러나 한 모델이 다른 모델보다 더 적합한 몇 가지 사용 사례가 있습니다. 그래프에서 두 노드를 연결하는 경로에 대한 지식이 필요한 경우 LPG를 선택합니다. Neptune 클러스터 또는 기타 그래프 트리플 저장소에서 데이터를 페더레이션하려면 RDF를 선택합니다.

서비스형 소프트웨어(SaaS) 애플리케이션 또는 다중 테넌시가 필요한 애플리케이션을 빌드하는 경우 각 클러스터에 대해 하나의 테넌트를 보유하는 대신 데이터 모델에 테넌트의 논리적 분리를 통합하는 방법을 고려합니다. 이러한 유형의 설계를 달성하기 위해 고객 식별자를 레이블에 추가하거나 테넌트 식별자를 나타내는 속성 키 값 페어를 추가하는 등 SPARQL의 명명된 그래프 및 레이블 지정 전략을 사용할 수 있습니다. 클라이언트 계층이 이러한 값을 주입하여 논리적 분리를 유지하는지 확인합니다. 다중 테넌시 권장 사항에 대한 자세한 내용은 Amazon Neptune 데이터베이스를 실행하는 ISVs.

쿼리의 성능은 쿼리 처리 시 평가해야 하는 그래프 객체(노드, 엣지, 속성) 수에 따라 달라집니다. 따라서 그래프 모델은 애플리케이션의 성능에 상당한 영향을 미칠 수 있습니다. 가능하면 세분화된 레이블을 사용하고 경로 결정 또는 필터링을 달성하는 데 필요한 속성만 저장합니다. 성능을 높이려면 요약 노드 생성 또는 공통 경로를 연결하는 더 직접적인 엣지 생성과 같이 그래프의 일부를 미리 계산하는 방법을 고려합니다.

레이블이 동일한 엣지 수가 비정상적으로 많은 여러 노드에서의 탐색을 피하세요. 이러한 노드에는 종종 수천 개의 엣지가 있습니다(대부분의 노드에는 수십 개의 엣지 수가 있음). 그 결과 컴퓨팅 및 데이터 복잡성이 훨씬 높아집니다. 이러한 노드는 일부 쿼리 패턴에서 문제가 되지 않을 수 있지만, 특히 노드를 중간 단계로 탐색할 경우 데이터를 다르게 모델링하여 사용하지 않는 것이 좋습니다. 느린 쿼리 로그를 사용하여 이러한 노드에서 탐색하는 쿼리를 식별할 수 있습니다. 특히 디버그 모드를 사용하는 경우 평균 쿼리 패턴보다 지연 시간 및 데이터 액세스 지표가 훨씬 더 높을 수 있습니다.

Neptune을 사용하여 ID에 임의의 GUID 값을 할당하는 대신 사용 사례에서 지원하는 경우 노드 및 엣지에 결정적 노드 ID를 사용합니다. ID로 노드에 액세스하는 것이 가장 효율적인 방법입니다.

쿼리 최적화

openCypher 및 Gremlin 언어는 LPG 모델에서 상호 교환적으로 사용할 수 있습니다. 성능이 가장 중요한 문제인 경우 특정 쿼리 패턴에서 한 언어가 다른 언어보다 성능이 좋을 수 있으므로 두 언어를 서로 바꿔서 사용하는 방법을 고려합니다.

Neptune은 대체 쿼리 엔진(DFE)으로 전환하는 중입니다. openCypher는 DFE에서만 실행되지만 쿼리 주석을 사용하여 선택적으로 Gremlin 및 SPARQL 쿼리를 모두 DFE에서 실행하도록 설정할 수 있습니다. DFE가 활성화된 상태에서 쿼리를 테스트하고 DFE를 사용하지 않을 때 쿼리 패턴의 성능을 비교하는 것이 좋습니다.

Neptune은 전체 그래프를 평가하는 분석 쿼리가 아닌 단일 노드 또는 노드 세트에서 시작하고 여기에서 팬아웃하는 트랜잭션 유형 쿼리에 최적화되어 있습니다. 분석 쿼리 워크로드의 경우 Neptune Analytics를 사용합니다. Neptune Analytics는 데이터, 분석 및 알고리즘 처리를 위해 빠른 반복이 필요한 조사, 탐색 또는 데이터 과학 워크로드에 이상적인 선택입니다. 그래프 데이터에 대한 벡터 검색을 수행하고 Neptune 데이터베이스 인스턴스에서 직접 데이터를 로드할 수도 있습니다. Neptune Analytics가 요구 사항을 충족하지 않는 경우 AWS Pandas용 SDK 또는 AWS Glue 또는 또는 Amazon EMR과 결합된 neptune-export를 사용할 수도 있습니다.

모델 및 쿼리에서 비효율성과 병목 현상을 식별하려면 각 쿼리 언어에 대해 `profile` 및 `explain` API를 사용하여 쿼리 계획 및 쿼리 지표에 대한 자세한 설명을 얻습니다. 자세한 내용은 Gremlin profile, openCypher explain, SPARQL explain을 참조하세요.

쿼리 패턴을 이해합니다. 그래프의 명확한 엣지 수가 커질 경우 Neptune의 기본 액세스 전략이 비효율적으로 될 수 있습니다. 다음 쿼리는 매우 비효율적일 수 있습니다.

- 엣지 레이블이 지정되지 않은 경우 엣지를 역방향으로 탐색하는 쿼리.
- Gremlin의 `.both()`와 같이 이 동일한 패턴을 사용하는 절 또는 모든 언어로 노드를 삭제하는 절(레이블에 대한 지식 없이 수신 엣지를 삭제해야 함).
- 속성 레이블을 지정하지 않고 속성 값에 액세스하는 쿼리. 이러한 쿼리는 매우 비효율적일 수 있습니다. 사용 패턴과 일치하는 경우 OSGP 인덱스(객체, 제목, 그래프, 조건자)를 활성화하는 방법을 고려합니다.

느린 쿼리 로깅을 사용하여 느린 쿼리를 식별합니다. 최적화되지 않은 쿼리 계획이나 불필요하게 많은 인덱스 조회로 인해 느린 쿼리가 발생할 수 있으며, 이때 I/O 비용이 증가할 수 있습니다. Gremlin, SPARQL 또는 openCypher에 대한 Neptune 설명 및 프로파일 엔드포인트는 이러한 쿼리가 느린 이유를 이해하는 데 도움이 될 수 있습니다. 원인에 다음이 포함될 수 있습니다.

- 그래프의 평균 노드와 비교했을 때 엣지 수가 비정상적으로 많은 노드(예: 수십 개에 비해 수천 개에 해당)는 계산 복잡성을 추가하여 지연 시간을 늘리고 리소스 소비를 늘릴 수 있습니다. 이러한 노드가 올바르게 모델링되었는지 또는 통과해야 하는 엣지 수를 줄이기 위해 액세스 패턴을 개선할 수 있는지 확인합니다.
- 최적화되지 않은 쿼리에는 특정 단계가 최적화되지 않았다는 경고가 포함됩니다. 최적화된 단계를 사용하도록 이러한 쿼리를 다시 작성하면 성능이 향상될 수 있습니다.
- 중복 필터로 인해 불필요한 인덱스 조회가 발생할 수 있습니다. 마찬가지로 중복 패턴으로 인해 쿼리를 개선하여 최적화할 수 있는 중복 인덱스 조회가 나타날 수 있습니다(프로파일 출력에서 `Index Operations - Duplication ratio` 참조).
- Gremlin과 같은 일부 언어에는 강력한 형식의 숫자 값이 없으며 대신 형식 승격을 사용합니다. 예를 들어 값이 55인 경우 Neptune은 55와 동등한 정수, long, float 및 기타 숫자 유형인 값을 찾습니다. 이로 인해 추가 작업이 발생합니다. 사전에 유형 일치 항목을 알고 있는 경우 쿼리 힌트를 사용하여 이를 방지할 수 있습니다.
- 그래프 모델은 성능에 큰 영향을 미칠 수 있습니다. 보다 세분화된 레이블을 사용하거나 다중 홉 선형 경로에 대한 바로 가기를 미리 계산하여 평가해야 하는 객체 수를 줄이는 방법을 고려합니다.

쿼리 최적화만으로 성능 요구 사항에 도달할 수 없는 경우 Neptune과 함께 다양한 캐싱 기법을 사용하여 이러한 요구 사항을 충족하는 것이 좋습니다.

Neptune 성능은 각 버전마다 지속적으로 개선되고 있습니다. 릴리스 정보를 검토하여 각 릴리스의 개선 사항에 대한 세부 정보를 확인합니다. 최적의 성능을 달성하려면 Neptune DB 클러스터에 대한 정

기적인 업데이트를 계획하는 것이 좋습니다. 최신 버전은 최신 인스턴스도 지원합니다. r8g 인스턴스를 활용하려면 1.4.5.0 이상으로 업그레이드하는 것이 좋습니다. 이렇게 하면 워크로드 성능을 개선할 수 있는 방법에 대한 자세한 내용은 Amazon Neptune v1.4.5를 사용하는 AWS Graviton4 R8g 인스턴스를 사용하여 쿼리 가격 대비 성능을 4.7배 개선했습니다.

올바른 크기의 클러스터

동시성 및 처리량 요구 사항에 맞게 클러스터의 크기를 조정합니다. 클러스터의 각 인스턴스에서 처리할 수 있는 동시 쿼리 수는 해당 인스턴스의 가상 CPU(vCPU) 수의 2배와 같습니다. 모든 작업자 스레드가 점유된 동안 도착하는 추가 쿼리는 서버 측 대기열에 배치됩니다. 이러한 쿼리는 작업자 스레드가 사용 가능해지면 선입선출(FIFO) 방식으로 처리됩니다. `MainRequestQueuePendingRequests` Amazon CloudWatch 지표는 각 인스턴스의 현재 대기열 깊이를 보여줍니다. 이 값이 자주 0보다 크면 vCPU가 더 많은 인스턴스를 선택하는 것이 좋습니다. 대기열 깊이가 8,192를 초과하면 Neptune은 `ThrottlingException` 오류를 반환합니다.

각 인스턴스에 대한 RAM의 약 65%는 버퍼 캐시용으로 예약되어 있습니다. 버퍼 캐시에는 작업 데이터세트(전체 그래프가 아니라 쿼리 중인 데이터만)가 들어 있습니다. 스토리지 대신 버퍼 캐시에서 가져오는 데이터의 비율을 확인하려면 CloudWatch 지표 `BufferCacheHitRatio`를 모니터링합니다. 이 지표가 종종 99.9% 미만으로 떨어지면 메모리가 더 많은 인스턴스를 시도하여 지연 시간 및 I/O 비용을 줄이는지 확인하는 것이 좋습니다.

읽기 전용 복제본의 크기가 라이터 인스턴스와 같을 필요는 없습니다. 그러나 쓰기 워크로드가 많으면 더 작은 복제본이 지연되어 복제를 따라잡을 수 없기 때문에 재부팅될 수 있습니다. 따라서 라이터 인스턴스보다 크거나 같은 복제본을 만드는 것이 좋습니다.

읽기 복제본에 오토 스케일링을 사용하는 경우 새 읽기 복제본을 온라인 상태로 전환하는 데 최대 15분이 걸릴 수 있습니다. 클라이언트 트래픽이 빠르지만 예측 가능하게 증가하면 예약된 조정을 사용하여 해당 초기화 시간을 고려하도록 최소 읽기 복제본 수를 더 높게 설정하는 방법을 고려합니다.

서버리스 인스턴스는 여러 사용 사례와 워크로드를 지원합니다. 다음 시나리오에서는 서버리스 과다 프로비저닝된 인스턴스를 고려합니다.

- 워크로드는 하루 종일 변동되는 경우가 많습니다.
- 새 애플리케이션을 생성했는데 워크로드 크기가 어떻게 될지 잘 모릅니다.
- 개발 및 테스트를 수행하고 있습니다.

서버리스 인스턴스는 RAM의 GB당 1 USD 기준으로 동등한 프로비저닝된 인스턴스보다 비용이 더 높다는 점에 유의해야 합니다. 각 서버리스 인스턴스는 연결된 vCPU 및 네트워킹과 함께 2GB의 RAM으

로 구성됩니다. 옵션 간에 비용 분석을 수행하여 예상치 못한 비용을 방지합니다. 일반적으로 매일 몇 시간 동안만 워크로드가 매우 많고 하루의 나머지 시간에는 거의 0이거나 하루 종일 워크로드가 크게 변동하는 경우에만 서버리스를 통해 비용을 절감할 수 있습니다.

Amazon Neptune 요금 계산기를 활용하여 queries-per-second(QPS) 요구 사항과 같은 요소를 기반으로 클러스터의 올바른 구성을 평가할 수 있습니다.

쓰기 최적화

쓰기를 최적화하려면 다음을 고려합니다.

- Neptune Bulk Loader는 데이터베이스를 처음 로드하거나 기존 데이터에 추가하는 최적의 방법입니다. Neptune Loader는 트랜잭션에 기반하지 않으며 데이터를 삭제할 수 없으므로 요구 사항인 경우 사용하지 마세요.
- 트랜잭션 기반 업데이트는 지원되는 쿼리 언어를 사용하여 수행할 수 있습니다. 쓰기 I/O 작업을 최적화하려면 커밋당 50~100개의 객체 배치로 데이터를 작성합니다. 객체는 노드, 엣지 또는 LPG의 노드나 엣지에 있는 속성 또는 RDF의 트리플 저장소나 쿼드입니다.
- 모든 트랜잭션 Neptune 쓰기 작업은 각 연결에 대해 단일 스레드입니다. Neptune에 대량의 데이터를 전송할 때는 각각 데이터를 쓰는 여러 병렬 연결을 사용하는 방법을 고려합니다. Neptune에서 프로비저닝된 인스턴스를 선택하면 인스턴스 크기가 여러 vCPU에 연결됩니다. Neptune은 인스턴스의 각 vCPU에 대해 두 개의 데이터베이스 스레드를 생성하므로 최적의 병렬화를 테스트할 때 vCPU 수의 두 배에서 시작합니다. 서버리스 인스턴스는 4개의 NCU마다 약 1개의 비율로 vCPU 수를 조정합니다.

  > **Note**
  >
  > 이는 대량 로드 API에는 적용되지 않고 직접 연결에만 적용됩니다.

- 단일 연결만 언제든지 데이터를 쓰는 경우에도 모든 쓰기 프로세스 중에 ConcurrentModificationExceptions를 계획하고 효율적으로 처리합니다. `ConcurrentModificationExceptions`가 발생할 때 신뢰성을 보장하도록 클라이언트를 설계합니다.
- 모든 데이터를 삭제하려면 동시 삭제 쿼리를 실행하는 대신 빠른 재설정 API를 사용하는 방법을 고려합니다. 후자는 전자에 비해 훨씬 더 오래 걸리고 상당한 I/O 비용이 발생합니다.
- 대부분의 데이터를 삭제하려면 neptune-export를 사용하여 데이터를 새 클러스터로 로드해 유지하려는 데이터를 내보내는 것이 좋습니다. 그리고 원래 클러스터를 삭제합니다.

비용 최적화 요소

AWS Well-Architected Framework의 비용 최적화 원칙은 불필요한 비용을 방지하는 데 중점을 둡니다. 다음 권장 사항은 Amazon Neptune의 비용 최적화 설계 원칙 및 아키텍처 모범 사례를 충족하는 데 도움이 될 수 있습니다.

비용 최적화 원칙은 다음 핵심 영역에 중점을 둡니다.

- 시간 경과에 따른 지출 이해 및 자금 할당 제어
- 올바른 유형 및 수량의 리소스 선택
- 과도한 지출 없이 비즈니스 요구 사항을 충족하도록 규모 조정

필요한 사용 패턴 및 서비스 이해

Neptune은 데이터 모델에 식별 가능한 그래프 구조가 있고 쿼리가 관계를 탐색하고 여러 홉을 통과해야 하는 경우 워크로드에 적합합니다. 그래프 데이터베이스는 다음 패턴에 적합하지 않습니다.

- 주로 단일 홉 쿼리(데이터가 객체의 속성으로 더 잘 표현될 수 있는지 고려)
- 속성으로 저장된 JSON 또는 BLOB 데이터
- 많은 수의 노드에서 숫자 속성의 합계를 계산하는 등 데이터세트 전체에서 집계되는 쿼리

특정 액세스 패턴에 대해 여러 목적별 데이터베이스를 함께 사용하면 모든 요구 사항을 해결할 수 있는지 고려합니다. 예제:

- 단일 노드에 대한 높은 동시 검색 속성과 함께 복잡한 그래프에 대해 필요한 탐색 빈도가 낮은 API는 Neptune, DynamoDB 또는 Amazon DocumentDB 중 하나 이상을 사용하여 제공하는 것이 가장 좋습니다.
- 관계형 데이터베이스는 기존 기능을 유지하기 위해 Neptune과 공존할 수 있지만 관계형 데이터베이스에서 제대로 수행 및 확장되지 않는 다중 홉 순회에만 Neptune을 사용합니다.

다음을 포함하여 Neptune과 상호 작용하고 보완하는 서비스와 관련된 비용을 이해합니다.

- Neptune에 대량 로드되는 데이터 파일의 Amazon Simple Storage Service(Amazon S3) 스토리지 비용
- 삽입 또는 업서트 쿼리, 읽기 쿼리 및 Neptune 스트림 처리에 사용되는 Lambda 함수

- Amazon API Gateway 또는에서 클라이언트 애플리케이션과 상호 작용하기 위해 Neptune에 구축된 API 계층(데이터베이스에 직접 연결하는 대신) AWS AppSync
- AWS Glue Neptune과 데이터를 주고받는 데 사용되는 작업
- Neptune으로 거의 실시간에 가깝게 수집하기 위해 스트리밍 데이터를 수신하는 Amazon Kinesis 또는 Amazon Managed Streaming for Apache Kafka(Amazon MSK) 인스턴스.
- AWS Database Migration Service Neptune으로 관계형 데이터 마이그레이션
- Jupyter Notebook 및 딥 그래프 라이브러리 기계 학습 모델에 대한 Amazon SageMaker Runtime 비용

비용에 주의를 기울여 리소스 선택

Neptune 요금은 시간당 인스턴스 비용(또는 서버리스에 소비되는 Neptune 컴퓨팅 단위), 데이터 I/O 및 스토리지 사용량을 기준으로 합니다. 인스턴스는 평균적으로 전체 비용의 85%를 차지하므로 적정 규모로 조정하면 비용에 상당한 영향을 미칠 수 있습니다. 인스턴스를 적정 규모로 조정하는 가장 좋은 방법은 다양한 인스턴스에서 애플리케이션 성능을 테스트하고 다음 요소를 비교하는 것입니다.

- `MainRequestQueuePendingRequests` CloudWatch 지표가 0에 가까운 일관되게 낮은 수로 유지되나요?
- `BufferCacheHitRatio` CloudWatch 지표가 대부분의 시간 동안 99.9% 이상으로 유지되나요?
- 인스턴스 비용 및 관련 데이터 I/O 비용에 대한 비용 및 성능 곡선은 무엇인가요? 스토리지로 버퍼 캐시를 자주 교체해야 하는 크기가 작은 인스턴스의 경우 데이터 읽기 비용이 크게 증가할 수 있습니다. `BufferCacheHitRatio`는 이러한 시나리오에서 자주 감소합니다.

인스턴스 비용은 동일한 인스턴스 패밀리 내 크기에 따라 선형적으로 조정됩니다. `db.r6i.2xlarge` 인스턴스의 시간당 비용은 `db.r6i.xlarge` 인스턴스의 두 배이며 리소스 할당도 두 배입니다. `db.r6i.24xlarge` 인스턴스는 `db.r6i.xlarge` 인스턴스의 시간당 비용의 24배입니다.

지원해야 하는 동시 쿼리 수를 추정합니다. 읽기 전용 쿼리를 처리하기 위해 0~15개의 읽기 복제본을 보유할 수 있습니다. 요일, 주 또는 월에 따라 요구 사항이 다른 경우 여러 개의 작은 인스턴스를 사용하여 일정에 따라 조정할 수 있습니다. 인스턴스의 각 vCPU는 동시 쿼리를 처리하기 위한 두 개의 스레드를 제공합니다. 각각 4개의 vCPU가 있는 3개의 `db.r6i.xlarge` 읽기 복제본은 24개의 동시 쿼리를 처리할 수 있습니다.

트래픽 볼륨이 초당 쿼리(QPS)로 측정되는 경우 쿼리의 평균 지연 시간을 확인하기 위해 실험해야 합니다. Neptune 클러스터가 지원할 수 있는 초당 쿼리 수는 `vCPU × 2 × (1 second/average`

query latency)와 같습니다. 예를 들어 vCPU가 4개이고 쿼리 지연 시간이 100밀리초(0.1초)인 경우 QPS = 4 × 2 × (1s/0.1s) = 80 queries per second입니다.

프로비저닝된 인스턴스는 지속적이고 안정적이며 예측 가능한 워크로드에서 서버리스보다 저렴합니다. 서버리스는 하루에 몇 시간 동안만 사용량이 매우 많고(예: db.r6i.4xlarge) 남은 시간 동안 트래픽이 거의 없는 워크로드(예: Neptune 컴퓨팅 유닛 1개)가 있는 경우 비용을 최적화할 수 있는 기회를 제공합니다. 몇 시간 동안 스케일 업했다가 다시 스케일 다운하는 서버리스 인스턴스는 프로비저닝된 db.r6i.4xlarge 인스턴스를 하루 종일 사용하는 것보다 비용이 저렴합니다.

Neptune 1.4.5.0 이상으로 업그레이드하고 r8g 인스턴스를 활용하여 r7g 또는와 같은 이전 세대 인스턴스보다 저렴한 비용으로 더 나은 읽기 및 쓰기 처리량을 달성하는 것이 좋습니다r6g. 자세한 내용은 Amazon Neptune v1.4.5를 사용하는 AWS Graviton4 R8g 인스턴스를 사용하여 쿼리 가격 대비 성능을 4.7배 향상(블로그 게시물)을 참조하세요.AWS

Neptune 클러스터는 기본적으로 표준 스토리지로 생성됩니다(콘솔을 사용하여 생성하는 경우 기본적으로 I/O 최적화 스토리지 선택). I/O 최적화 스토리지를 사용하면 스토리지 및 인스턴스에 약간 더 높은 비용을 지불하지만 I/O 비용은 없습니다. 이로 인해 반복 비용이 더 예측 가능하게 발생하지만 I/O 사용량이 일반적으로 낮은 경우 표준 스토리지를 활용하는 것이 더 비용 효율적일 수 있습니다. 처음에 많은 데이터를 로드하려는 경우 I/O 최적화 스토리지를 선택하고 초기 데이터 로드를 수행한 다음 표준 스토리지로 전환하여 비용을 최적화할 수 있습니다. 스토리지 유형은 결제 모델에만 영향을 미치며 Neptune DB 클러스터 또는 인스턴스 구성에는 기술적 차이가 없습니다. 30일에 한 번씩 스토리지 유형을 변경할 수 있습니다. 30일 후 세부 Neptune 비용을 확인하고 Neptune 요금 페이지를 사용하여 I/O 최적화 스토리지를 사용하여 비용이 더 높았는지 여부를 계산합니다. 그렇다면 표준 스토리지를 계속 사용하고, 그렇지 않으면 I/O 최적화로 다시 전환하세요.

워크로드에 가장 적합한 Neptune 인스턴스 구성 선택

2025년 7월 15일 AWS 계정 이전에를 생성한 경우 AWS 프리 티어를 사용하여 Neptune을 사용한 엔트리 레벨 실험을 수행할 수 있습니다. 750시간의 무료 db.t3.medium 및 db.t4g.medium 인스턴스 사용만으로도 작은 규모에서 Neptune을 이해하기에 충분합니다. 무료 평가판 기간이 종료된 후에도 클러스터는 유지됩니다. 단, 이후 사용량에 대해서는 요금이 부과됩니다.

db.t3.medium 및 db.t4g.medium 인스턴스는 openCypher, Graph Explorer 또는 다양한 생성형 AI 통합을 사용하지 않는 저비용 개발 환경에 적합합니다. 이러한 인스턴스는 패밀리 인스턴스(8:1) 또는 R 패밀리 인스턴스(1X6:1)보다 RAM-to-vCPU 비율(2:1)이 작습니다. 이렇게 하면 openCypher 성능, GenAI 통합(LLM에 그래프 스키마를 알리기 위해) 및 Graph Explorer를 활성화하는 DFE 엔진 통계를 사용할 수 없습니다. T 패밀리 인스턴스를 사용할 때 특히 앞서 언급한 워크로드의 경우 성능 프로필이 크게 다를 수 있습니다. 이러한 인스턴스는 쿼리가 그래프의 상당 부분을 탐색할

`OutOfMemoryExceptions` 때의 발생을 증가시킬 수도 있습니다. 후자 조건이 영향을 받을 수 있는지 확인하려면 `BufferCacheHitRatio` CloudWatch 지표를 확인합니다.

프로덕션 환경을 나타내지 않는 일관되지 않은 결과가 발생할 수 있으므로 T 패밀리 인스턴스로 성능 또는 로드 테스트를 수행하지 않는 것이 좋습니다.

프로비저닝된 인스턴스는 워크로드가 매우 안정적이고 예측 가능한 경우 최상의 비용과 성능 조합을 제공합니다. 필요한 요청 동시성과 쿼리 복잡성에 따라 인스턴스 크기를 선택합니다. 동시성이 높을수록 vCPU가 더 많이 필요합니다. 쿼리 복잡성이 높을수록 RAM이 더 많이 필요합니다. `MainRequestQueuePendingRequests` CloudWatch 지표를 사용하여 전자의 영향을 확인합니다(0보다 크면 처리할 수 있는 것보다 동시 요청어 더 많은 상태임을 나타냄). `BufferCacheHitRatio` CloudWatch 지표를 사용하여 후자의 영향을 확인합니다. 99.9% 미만으로 자주 떨어지는 비율은 평가 중인 그래프의 작업 부분을 포함할 RAM이 부족하여 캐시 스왑이 더 자주 발생함을 나타냅니다. R 인스턴스 패밀리가 충분한 동시성을 제공하지만 RAM이 충분하지 않은 경우 인스턴스 X 패밀리를 사용해 보는 것이 좋습니다.

서버리스 인스턴스의 이상적인 사용 사례는 Neptune 설명서에 설명되어 있습니다. 프로비저닝 또는 서버리스 중 가장 적합한 옵션이 확실하지 않고 비용이 주요 관심사인 경우 서버리스에서 워크로드를 테스트하여 사용된 NCU 수를 확인하고 프로비저닝(`N hours × hourly provisioned cost`) 비용을 서버리스(`sum of NCUs × hourly cost per NCU`)와 비교합니다. 동일한 크기의 프로비저닝 인스턴스에 대해 잘 모르는 경우 NCU 하나는 약 2GB의 RAM과 관련 vCPU 및 네트워킹에 해당합니다. 프로비저닝된 인스턴스가 `r6i` 패밀리인 경우 비율은 연결된 네트워킹과 함께 RAM 8GB당 vCPU 1개 또는 NCUs 4개입니다. Amazon Neptune 요금 계산기는 최적의 비용 구성을 결정하는 데 도움이 되는 비교도 제공합니다.

기본 및 복제본 인스턴스에 서버리스를 사용하는 경우 승격 티어 0 및 1의 읽기 복제본은 라이터 인스턴스에 따라 NCU를 조정하므로 장애 조치 이벤트가 발생할 경우 적절하게 조정됩니다. 라이터 또는 리더 중 어떤 인스턴스가 가장 많은 트래픽을 수신하는지에 따라 이러한 인스턴스에 대한 NCU 제한을 설정합니다.

클러스터가 연중무휴로 필요하지 않은 환경에서는 사용하지 않을 때 Neptune 인스턴스를 끄고 사용하기 전에 다시 시작하는 스크립트를 작성하는 것이 좋습니다. Neptune 인스턴스는 필요한 유지 관리 업데이트가 적용되도록 7일마다 자동으로 다시 시작됩니다. 인스턴스를 장기간 끄려는 경우 주간 스크립트를 사용하여 인스턴스를 다시 종료합니다.

데이터 스토리지 적정 규모 조정 및 전송

보다 효율적인 쿼리(예: 그래프에서 사용해야 하는 노드, 엣지 및 속성 수가 더 적은 쿼리)는 I/O 전송을 줄이고 필요한 버퍼 캐시가 적기 때문에 더 작은 인스턴스를 사용할 수 있습니다. 쿼리 언어의 프로파

일 또는 설명 엔드포인트를 사용하여 쿼리를 최적화하고 쿼리 성능에 맞게 그래프 모델을 최적화하는 방법을 고려합니다.

Neptune은 큰 문자열에서 사전 인코딩을 사용하며, 해당 사전은 효율성이 아니라 성능에 최적화되어 있습니다. 속성에 대해 큰 BLOB, JSON 또는 빈번하게 변경되는 문자열이 있는 경우 Amazon S3, Amazon DynamoDB 또는 Amazon DocumentDB에서 이를 Neptune 외부에 저장하고 Neptune 노드 내에는 참조만 저장하는 방법을 고려합니다.

경우에 따라 더 큰 인스턴스 크기를 선택하는 방법이 더 저렴할 수 있습니다. 낮은 `BufferCacheHitRatio`로 인해 I/O 비용이 매우 높은 경우 더 큰 버퍼 캐시를 사용하면 해당 비용을 크게 줄일 수 있습니다. 스토리지에서 자주 전환되고 I/O 전송 비율을 발생시키는 대신 모든 데이터가 캐시에 맞춰지기 때문입니다.

Neptune은 쓸 때 복사 복제를 사용합니다. 그래프를 여러 샤드로 분할하기 위해 복제하는 경우 복제된 클러스터에서 원치 않는 데이터를 삭제하지 않는 방법이 더 효율적일 수 있습니다. 이 경우 새 데이터 페이지가 생성되어 스토리지 비용이 증가하기 때문입니다. 복제 이벤트 이전에 변경되지 않는 데이터는 두 클러스터에서 공유되는 단일 데이터 페이지에 존재하고, 해당 단일 사본에 대해서만 요금이 부과됩니다.

워크로드에 상당한 차이가 있는지 테스트하지 않은 경우 OSGP 인덱스를 활성화하거나 R5d 인스턴스를 사용하지 마세요. 둘 다 거의 발생하지 않는 시나리오를 위해 설계되었으며, 최소한의 이익 또는 전혀 얻는 이익 없이 비용이 증가할 수 있습니다.

지속 가능성 요소

지속 가능성 원칙은 클라우드 워크로드 실행이 환경에 미치는 영향을 최소화하는 데 중점을 둡니다. 주요 주제에는 지속 가능성에 대한 공동 책임 모델, 영향 이해, 사용 극대화로 필요한 리소스를 최소화하고 다운스트림 영향을 줄이는 것이 포함됩니다.

지속 가능성 원칙에는 다음과 같은 주요 핵심 영역이 포함됩니다.

- 사용자의 영향
- 지속 가능성 목표
- 최대 사용량
- 새롭고 보다 효율적인 하드웨어 및 소프트웨어 제품 예측 및 채택
- 관리형 서비스 사용
- 다운스트림 영향 감소

이 가이드는 사용자의 영향에 중점을 둡니다. 기타 지속 가능성 설계 원칙에 대한 자세한 내용은 AWS Well-Architected 프레임워크를 참조하세요.

사용자의 선택 사항과 요구 사항은 환경에 영향을 미칩니다. 탄소 집약도가 낮은 AWS 리전 을 선택할 수 있고 요구 사항이 가동 시간과 내구성을 극대화하는 대신 실제 워크로드 요구 사항을 반영하는 경우 워크로드의 지속 가능성이 증가합니다. 다음 섹션에서는 워크로드 설계 및 지속적 운영에 채택될 경우 환경에 긍정적인 영향을 미칠 수 있는 모범 사례와 신중한 고려 사항에 대해 설명합니다.

AWS 리전 선택

일부는 Amazon 재생 에너지 프로젝트 AWS 리전 근처에 있거나 그리드의 탄소 강도가 다른 프로젝트보다 낮은 곳에 있습니다. 워크로드에 적합할 수 있는 리전의 지속 가능성 영향을 고려하고 목록을 Neptune을 사용할 수 있는 리전과 상호 참조합니다.

사용자 행동 패턴 기반 소비

사용자의 트래픽 및 동작에 맞게 소비를 적정 규모로 조정하면 AWS 에서 환경에 미치는 서비스의 영향을 최소화하는 데 도움이 됩니다. 솔루션을 설계할 때 다음 모범 사례를 고려하세요.

- `CPUUtilization`, `MainRequestQueuePendingRequests` 및 `TotalRequestsPerSec`과 같은 Amazon CloudWatch 지표를 모니터링하여 수요가 가장 높은 시점과 가장 낮은 시점을 결정하고 해당 시간 동안 클러스터 리소스의 크기가 적절한지 확인합니다.
- 사용하지 않는 시간 동안 비프로덕션 환경의 중지를 자동화합니다. 자세한 내용은 블로그 게시물 Automate the stopping and starting of Amazon Neptune environment resources using resource tags를 참조하세요.
- 트래픽 패턴이 자주 예기치 않게 달라지는 경우 피크 트래픽에 대해 프로비저닝된 인스턴스를 사용하는 대신 수요에 따라 스케일 업 및 스케일 다운되는 Neptune Serverless 인스턴스를 사용하는 것이 좋습니다.
- 비즈니스 연속성 목표 외에도 서비스 수준 계약을 지속 가능성 목표에 맞게 조정하는 방법을 고려합니다. 다중 리전 재해 복구, 고가용성 또는 장기 백업 보존, 특히 비프로덕션 환경 또는 미션 크리티컬 이외의 워크로드와 같은 요구 사항을 완화하면 이러한 목표를 달성하는 데 필요한 리소스의 양을 줄일 수 있습니다.

소프트웨어 개발 및 아키텍처 패턴 최적화

낭비를 방지하려면 모델 및 쿼리를 최적화하고 컴퓨팅 리소스를 공유하여 Neptune 인스턴스 및 클러스터에서 사용할 수 있는 모든 리소스를 사용합니다. 구체적인 모범 사례는 다음과 같습니다.

- 개발자가 각각 고유한 인스턴스를 생성하는 대신 Neptune 인스턴스와 Jupyter Notebook 애플리케이션 인스턴스를 공유하도록 합니다. 멀티테넌시 분할 전략을 사용하여 각 개발자에게 단일 Neptune 클러스터의 자체 논리적 파티션을 제공하고 단일 Jupyter 인스턴스에서 각 개발자에 대해 별도의 노트북 폴더를 생성합니다.
- 데이터를 로드하고 레코드를 더 큰 트랜잭션으로 배치 처리하기 위한 병렬 스레드와 같이 리소스 사용을 극대화하고 유휴 시간을 최소화하는 패턴을 구현합니다.
- 결과를 계산하는 데 필요한 리소스를 최소화하도록 쿼리 및 그래프 모델을 최적화합니다.
- Gremlin 쿼리 결과의 경우 결과 캐시 기능을 사용하여 페이지가 지정되거나 자주 반복되는 쿼리를 다시 계산하는 데 소비되는 리소스를 최소화합니다.
- Neptune 환경을 최신 상태로 유지합니다. 최신 버전의 Neptune은 더 효율적인 Graviton과 같은 최신 Amazon EC2 인스턴스를 지원합니다. 또한 쿼리를 계산하는 데 필요한 리소스의 양을 줄이는 쿼리 최적화 개선 및 버그 수정도 있습니다.

리소스

참조

- AWS Well-Architected
- AWS Well-Architected Framework 설명서
- Neptune 최신 업데이트
- 모범 사례: Neptune 최대한 활용
- Amazon Neptune 요금 계산기

블로그 게시물

- Automated testing of Amazon Neptune data access with Apache TinkerPop Gremlin
- Automate the stopping and starting of Amazon Neptune environment resources using resource tags
- Fine Grained Access Control for Amazon Neptune data plane actions
- Amazon Neptune v1.4.5를 사용하여 AWS Graviton4 R8g 인스턴스의 쓰기 쿼리 가격 대비 성능이 4.7배 향상되었습니다.
- Orca Security가 Amazon Neptune 데이터베이스 성능을 최적화한 방법
- Amazon Neptune 퍼블릭 엔드포인트를 사용하여 그래프 애플리케이션을 더 빠르게 구축
- 새로운 Amazon Neptune 엔진 버전은 openCypher 쿼리 성능을 위해 최대 9배 더 빠르고 10배 더 높은 처리량을 제공합니다.

무료 AWS Skill Builder 과정

- Getting Started with Amazon Neptune
- Building Applications on Amazon Neptune
- Data Modeling for Amazon Neptune

기여자

이 가이드의 기여자는 다음과 같습니다.

- Brian O'Keefe, Principal Neptune Solutions Architect, AWS
- Abhishek Mishra, Senior Neptune Solutions Architect, AWS
- Ganesh Sawhney, 팀 리드 - 전략적 파트너 성공 솔루션 아키텍트, AWS
- Michael Havey, Senior Neptune Solutions Architect, AWS
- Kevin Phillips, Neptune 솔루션 아키텍트, AWS
- Melissa Kwok, Neptune 솔루션 아키텍트, AWS
- Sakti Mishra, Principal Solutions Architect AWS
- Javed Ali, Senior Solutions Architect, AWS

문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하십시오.

변경 사항	설명	날짜
Neptune 릴리스 업데이트	Amazon Neptune 1.4.6.0 이상에 대한 정보를 포함하도록 설명서를 업데이트했습니다.	2026년 1월 2일
최초 게시	—	2023년 9월 27일

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