9주차 Storage

Aws Storage Option

1. Block Storage

Change one block

2. Object Storage

Entire file

Amazon EBS

* Block Storage 방식으로 저장
* Volume이 인스턴스와 독립적
* 모든 volume 유형은 월 단위로 프로비저닝된 용량에 따라 청구
* Availability zone내에서 자동으로 볼륨 복제
* Snapshot을 통해 자동으로 S3에 백업 가능
* Snapshot 추가 비용은 저장된 데이터 GB-월당 비용으로 청구
* 데이터 전송: 인바운드는 무료, 지역 간 아웃바인드는 비용O

IOPS

-general purpose ssd: 스토리지 해제 시까지 월 단위로 프로비저닝한 용량 따라 청구

- magnetic: 볼륨에 대한 요청 수 따라 비용 청구

- Provisioned IOPS ssd: 프로비저닝한 IOPS 양에 따라 청구

\*Snapshot: EBS 볼륨의 데이터를 안정적으로 보관하고, 백업 및 복구, 데이터 복사 등의 용도

Volume type // SSD가 HDD가 더 우수

SSD: general purpose, provisioned IOPS

HDD: throughput-optimized, cold

1. general purpose

Virtual dektop, system boot volume, 낮은 지연 시간

2. provisioned IOPS

Large database workload

3. throughput-optimized

Big data, data warehouse

Feature

1. Snapshot: 특정 시점, 언제든 새볼륨으로 재생성

2. encryption(암호화): 추가 비용X

3. elasticity(탄력성): 용량 증가, 유형 변경

Amazon S3

* 데이터는 버킷에 객체로 저장
* 무제한의 저장 공간이 제공, 단일 객체는 5tb 제한
* 버킷, 객체에 대한 세밀한 접근 가능
* Region에 데이터 중복 저장
* 원활한 확장성을 통해 설계(동적 할당, 유연성)
* 어디서든지 데이터에 접근O(aws, sdk 통해)
* Server-side encryption

Classes

* Standard
* Intelligent-tiering
* Standard-infrequent access(standard-IA)
* One zone-infrequent access(one zone-IA)
* Glacier
* Glacier deep archieve

Bucket URL

Path style: region code 먼저 bucket name 나중에

Virtual hosetd style: bucket name 먼저 region code 나중에

Pricing

사용한 만큼 지불

-월별로 GB 단위로 사용, 다른 지역으로 전송(OUT), PUT, COPY, POST 등의 요청

지불X

-S3로의 전송(IN), 동일 지역내로의 전송(OUT, s3->ec2, cloudfront)

비용 추정 위한 고려사항

1. 저장 클래스 유형

표준 스토리지: 내구성 11 9s, 가용성 4 9s

Standard-IA: 내구성 11 9s, 가용서 3 9s

2. 저장용량: 객체 수, 크기

3. 요청의 수, 유형: 요청마다 다른 요금 적용

4. 데이터 전송

Out에 대한 데이터 전송은 요금 청구

Amazon EFS: aws cloud의 파일 저장소

1. 빅데이터, 미디어 처리 워크플로우, 콘텐츠 관리 등에 적합

2. PB 규모의 저지연 파일 시스템

3. 탄력적 용량, 공유 저장소

4. 4.0, 4.1 버전의 네트워크 파일 시스템(NFS) 지원

5. ec2용의 모든 리눅스 기반 AMI와 호환

Efs와 Aws의 작업 절차

1. ec2 인스턴스 시작, 필요 리소스 프로비저닝

2. efs 파일 시스템 생성

3. 적절한 서브넷에 마운트 대상 생성

4. ec2 인스턴스 마운트 대상과 연결

5. aws 계정의 리소스와 보호 상태 체크

Resouces

파일 시스템

마운트 대상

* 서브넷 ID
* 보호 그룹
* 파일 시스템 당 하나 이상
* Vpc 서브넷에서 생성
* 가용 영역당 하나
* 동일한 vpc 내

태그

* 키-value 쌍

Amazon S3 glacier: 데이터 아카이빙 서비스, 보안, 내구성, 저렴

* 데이터 전송, 정지 시 SSL, TLS 통한 데이터 암호화 지원
* Vault lock 기능 통해 규정 준수 강제
* 아카이브 액세스: expedited, standard, bulk 세 옵션 제공
* Expedited: 1-5분, standard: 3-5시간, bulk: 5-12시간
* 객체에 대해 11 9s의 내구성 제공
* S3의 수명 주기를 glacier로 구성 가능
* S3 standard-(30일 이상)-> S3 standard-IA-(60일 이상)->glacier-(1년 이상)-> delete
* Upload, retrieval 유료 -> retrieval: per request, per gb
* 데이터가 기본적으로 암호화 되어있음

Security

* Control access: IAM
* Encrypt: AES-256
* Manages our key

10주차 Database

확장성, 내결함성, 가용성 (리소스 유연성, 데이터 복제&백업, 분산 배치)

Unmanaged(유저), Managed(서비스)

RDS

* Managed service: 자동으로 설치. 운영해줌
* 유저가 서버 통해 rds에 접근
* 유저가 관리할 부분: 응용 프로그램의 최적화
* 예) MySQL, MariaDB, Oracle, Amazon Aurora …
* 유저가 internet gateway 통해 vpc 내의 ec2(public 서브넷), rds(private 서브넷)에 접근
* RDS instance -Synchronous(동기)-> RDS standby instance

\* Synchronous(동기): 작업이 완료될 때까지 다음 작업이 실행되지 않는 동작

DB Instance class(인스턴스의 사양)

* CPU, 메모리, 네트워크 성능

DB Instance storage(인스턴스의 저장 공간 유형)

* Magnetic(HDD, general purpose(SSD), IOPS(SSD의 종류)

Read replicas(DB instance 읽기 부하 분산)

* 비동기 복제 제공
* 필요시 주요한 복제본으로 승격O
* Read-heavy DB 워크로드에 사용O
* 읽기 쿼리 분산처리해 부하 분산O

사용 예시

|  |  |  |
| --- | --- | --- |
| 웹, 모바일 앱 | 전자 상거래 앱 | 모바일, 온라인게임 |
| 높은 처리량 | 저렴한 db | 신속한 용량 확장 |
| 대규모 용량 확장 | 데이터 보안 | 자동 확장 |
| 높은 가용성 | 완전히 관리되는 솔루션 | Db 모니터링 |

RDS를 사용해야 하는 경우

* 복잡한 트랜잭션, 쿼리 필요시
* 중간부터 높은 쿼리, 쓰기 비율
* 단일 워커 노드, 샤드 쓰는 경우
* 높은 내구성 필요시

RDS 사용하면 안되는 경우

* 대규모 읽기, 쓰기 비율 필요시
* 데이터 크기, 처리량 요구로 인한 샤딩
* 간단한 get, put, NoSQL DB가 처리 가능한 쿼리
* RDBMS 사용자 정의 필요시

특징

1. 시간 당 청구: 실행 중일 때 비용 발생

2. DB 특성: 물리적 용량(엔진, 크기, 메모리 클래스)

3. DB 구매 유형

- 온디맨드 인스턴스: 시간 당 컴퓨팅 용량

- 예약 인스턴스: 낮은 일시적 선불금 필요

4. DB 인스턴스 수: 최대 부하 처리 위해 여러 개를 프로비저닝

프로비저닝된 스토리지: 요금x, 활성 DB 100%까지 백업, GB당 월 청구, 종료된 db의 백업

추가 스토리지: GB당 월 청구, 프로비저닝된 스토리지 외의 백업 스토리지

5. 데이터 전송

인바운드: 요금X, 아웃바운드: 계층별 요금O

6. 단일 가용 영역, 다중 가용 영역 따라 요금 다름

DynamoDB(규모와 관계 없이 빠르고, 유연한 NoSQL)

* 무제한의 저장 공간
* 지연 시간이 낮은 쿼리
* 확장 가능한 읽기/쓰기 처리량
* 항목은 다른 속성을 가질 수 있다?

|  |  |  |
| --- | --- | --- |
|  | SQL(Relational) | Non-Relational |
| storage | Row-column | Key-value, document, graph |
| schema | fixed | dynamo |
| query | sql | Collection of document |
| scalability | Vertical(수직 확장) | Horizontal(수평 확장) |

주요 구성 요소

* 테이블, 아이템, 속성
* 두 가지의 기본 키 지원(파티션 키, 정렬 키)

파티션 키

Dynamo DB의 테이블 데이터 분할, 각 항목은 고유한 파티션 키 값 가짐, 항목 식별의 주요 기준

정렬 키: 파티션 키와 함께 쓰임, 파티션 키로 분할된 데이터가 정렬 키 통해 정렬, 단독X

Red Shift(복잡한 SQL 쿼리, 대규모 데이터 관리)

* 병렬 처리(분산) 아키텍처: 데이터 처리 병렬화, 컴퓨팅 리소스 동적 확장
* 컬럼 지향 DB: 데이터를 컬럼 단위로 저장-> 필요한 컬럼만 get
* 비용 효율: 필요한 만큼 리소스 유연하게 조정, 효율적 요금 체계
* 자동화: 데이터 운영과 관련된 많은 작업 자동화, 자동 파티셔닝 기능
* 호환성: 기존 SQL 기반 솔루션과 호환, PostgreSQL 엔진 기반

사용 예시

기업용 데이터 웨어하우스(EDW)

* 이관 속도 good
* 큰 초기 비용, 약정 X
* 비즈니스적 요구에 빠른 대응O

빅데이터

* 소규모 고객 위한 저렴한 가격 정책
* 배포, 유지, 보수 용이한 관리 서비스
* 데이터 관리보다 데이터 자체에 집중O

SaaS

* 수요 따라 용량 확장O, 필요한 만큼!
* 애플리케이션에 분석 기능 추가O
* 하드웨어, 소프트웨어 비용 절감

Amazon Aurora(기업용 관계형DB)

1. 고성능: 인메모리 쿼리 가속화, 병렬 쿼리 처리, 캐시 -> 빠른 응답시간, 높은 처리량

2. 고가용: 여러 가용영역에서 데이터 자동 복제, 장애 발생시 복제본으로 전환해 중단X(S3)

3. 내구성, 신뢰성: 데이터 분산 저장, 자동 백업, 복구

4. 호환성: Mysql, postgreSQL과 호환, 기존 애플리케이션, 도구 이식 쉬움

5. 자동: 프로비저닝, 패치, 복구 등 시간 소모적 작업 자동화, 개발자와 운영자 부담 줄임

6. 확장: 요구에 따라 db용량 확장 쉬움, 자동으로 스케일 업, 아웃

최종 정리

|  |  |
| --- | --- |
| Rds | 기업 단위의 관계형db |
| Dynamo db | 규모 관계없이 빠르고 유연한 NoSQL DB 서비스 |
| Db on ec2 | 운영체제 액세스, 애플리케이션 지원 안할 수도 |
| Aws purpose-built db | 특정 사례에 대한 니즈(머신러닝, 데이터웨어하우스, 그래프) |

11주차 클라우드 아키텍처

아키텍처 디자인 고려사항

* 안전, 고성능, 탄력적, 효율적
* 클라우드 아키텍처 평가, 구현하는 일관된 접근 방식
* 고객 아키텍처 검토해 얻은 교훈 통해 개발된 최상의 사례 제시

좋은 아키텍처 프레임의 기둥

1. 운영 우수성(비즈니스 가치 제공)

* 변경 사항 자동화, 이벤트 대응, 일상 운영 관리위한 표준 정의
* 코드로 작업 수행, 장애 예측, 빈번하고 작은 범위의 변경 수행
* 운영 절차 개선, 모든 운영 이벤트와 실패로부터 학습
* 조직, 준비, 운영, 진화에 대한 질문

2. 보안(시스템 보호, 모니터링)

* 데이터 기밀, 무결성, 보안 이벤트 감지위한 통제 설정, 자격 식별 관리
* 식별 기반 구현, 추적 활성화, 모든 레이어서 보안, 데이터 접근 제어
* 데이터 이동, 정지 상태 보호
* 보안, 탐지, 인프라 보호, 데이터 보호, 사고 대응에 대한 질문

3. 신뢰성(장애 복구, 중단 완화)

- 분산 시스템, 복구

- 수평 확장 통한 워크로드 가용성 증대, 용량 추정 중단 후 관리, 자동화로 변경 관리

- 기초, 워크로드, 관리 변경, 실패 관리에 대한 질문

4. 성능 효율(자원 절약)

- 컴퓨팅 자원 효율적 사용-> 시스템 니즈 충족

- 워크로드 니즈에 맞는 자원 유형, 크기 선택, 성능 모니터링

- 최신 기술, 서버리스 아키텍처, 몇 분내로 글로벌적 확장, 자주 실험

- 선택, 리뷰, 모니터링, 트레이드 오프에 대한 질문

5. 비용 최적화(불필요한 비용 제거)

- 비용이 쓰이는 곳 이해, 통제, 적절한 리소스 선택, 시간당 지출 분석

- 비즈니스 요구 충족하며 과다 지출 피하기 위한 확장

- 클라우드 재무 관리, 소비 모델 채택, 차별화되지 않는 무거운 작업에 지출X

- 수요, 공급 리소스 관리, 시간 따른 최적화에 대한 질문

AWS well-architected 도구

* 워크로드 상태 검토
* Aws 아키텍트의 지식, 모범 사례 액세스
* 액션 플랜 제공
* 일관된 프로세스 제공

신뢰성&가용성

신뢰성

* 유저가 원할 때 시스템이 기능 제공할 수 있는 능력
* 시스템: 하드웨어, 펌웨어, 소프트웨어 등의 모든 구성 요소
* 일정 기간 동안 시스템이 의도한 대로 작동할 확률
* MTBF(고장 사이 평균 시간) = 총 서비스 시간/고장횟수

= 평균 고장 시간(MTTF) + 평균 수리 시간(MTTR)

가용성(정상 작동 시간/총 작동 시간)

* 시간 단위로 측정
* Number of 9s로 표현, 5 9s-> 99.999%
* 고가용성
  + 시스템 다운 타임 최소화, 최소의 인간 개입으로 운영
  + 여러 서버->로드밸런싱, 장애조치, 데이터 복제&백업

|  |  |  |
| --- | --- | --- |
| 가용성 | 최대 장애 시간/ 연당 | 응용 프로그램 종류 |
| 99 | 3일 15시간 | 배치 프로세싱, 데이터 추출 |
| 99.9 | 8시간 45분 | 지식 관리, 프로젝트 추적 |
| 99.95 | 4시간 22분 | 전자 상거래, 결제 시스템 |
| 99.99 | 52분 | 방송, 영상 전달 |
| 99.999 | 5분 | 통신, 현금 인출 |

가용성에 영향 주는 요소

* 장애 허용성, 확장성, 복구력

Aws trust advisor(실시간 가이드 제공하는 온라인 도구)

대시보드: 비용 최적화, 성능, 보안, 고장 허용성, 서비스 제한

\*데이터 전송, 저장 중의 보호를 위해 사용해야하는 서비스: s3, ebs, elb

12주차 자동 확장성, 모니터링

Elastic load balancing

* 여러 대상에 대해 들어오는 애플리케이션, 트래픽을 단일, 다중 영역으로 분산
* 애플리케이션으로의 트래픽이 변화함에 따라 로드밸런서 규모 확장

사용 예시

* 고가용성, 내결함성이 있는 애플리케이션, VPC, 컨테이너화된 애플리케이션
* 탄력성, 확장성, HTTP 통해 lambda 함수 호출, 하이브리드 환경(클라우드+온프레미스)

Application load balancer(네트워크 계층)

* http, https 트래픽 처리, 대상 그룹: ec2 인스턴스, lambda 함수, ip 주소

Network load balancer(전송 계층)

* tcp, udp, tls 트래픽 처리, 대상 그룹: 트래픽 전달할 ip주소, ec2 인스턴스, lambda 함수

Classic load balancer

* http(s), tcp, ssl 트래픽 처리, 대상 인스턴스 직접 등록, 일부 레거시 앱, 아키텍처 호환

|  |  |
| --- | --- |
| 응용 프로그램 로드 밸런서 | 컨테이너화된 응용 프로그램, HTTP |
| 네트워크 로드 밸런서 | 예측 불가한 tcp, 정적, elastic IP, VPC 외부 IP,  낮은 지연시간 +초당 수백만 건의 요청 |
| 클래식 로드 밸런서 | 여러 프로토콜 사용하는 간단한 로드밸런싱 |

구조

로드밸런서가 클라이언트로부터의 트래픽 수신, 분배

리스너는 트래픽이 로드 밸런서로 전달되기 전 연결 요청 확인 후 필요한 처리 수행

로드밸런서는 등록된 대상의 상태를 모니터링하기 위해 건강 상태 검사 진행

\*건강 상태 검사: 응답 확인, 지정된 프로토콜, 경로로 대상 접근해 확인

로드밸런서는 건강한 상태의 대상으로 트래픽 라우팅

Amazon cloudWatch 메트릭: 시스템 작동 확인, 메트릭 허용범위 넘어서면 경고 생성

액세스 로그: 로드 밸런서로 전송된 요청에 대한 자세한 정보 캡처

Aws cloudtrail 로그: aws에서 api 상호작용의 수행 기록

Amazon cloud watch(모니터링, 관찰 서비스)

모니터링: aws 리소스, aws 실행 애플리케이션

수집, 추적: 표준 메트릭, 사용자 정의 메트릭

알람: ec2 auto scaling, ec2 동작 수행

이벤트: aws 환경 변경 일치 등의 이벤트를 하나 이상의 대상 함수, 스트림으로 라우팅

정적 임계값, 이상 탐지, 메트릭 수식 기반으로 알람 생성

네임 스페이스, 메트릭, 통계량, 주기, 조건, 추가 구성 요소, 동작 특정 가능

|  |  |
| --- | --- |
| Ec2 | 평균 CPU 사용률이 5분 동안 60%보다 크다면, O |
| Rds | 동시 연결 수가 1분 동안 10보다 크다면, O |
| S3 | 1일 동안 최대 버킷 크기가 약 3바이트라면, X, 약은 임계값 옵션X, 부호로 정확히 명시 |
| Elb | 건강한 호스트 수가 10분 동안 5보다 작다면, O |
| Ebs | 읽기 작업의 볼륨이 10초 동안 1,000보다 크다면, X, 통계(평균 볼륨) 지정 |

Amazon ec2 Automatic Scaling(자동 스케일링, 관리)

Maximum size = desired capacity+launch or terminate instance as needed

= 원하는 용량 + 실행, 종료 작업

Scale out(인스턴스+): 트래픽, 워크로드 증가 등의 설정된 조건에 따라 확장

Scale in(인스턴스-): 트래픽, 워크로드 감소 등의 설정된 조건에 따라 종료

Launch configuration(시작 구성) -> Auto scaling group

-> 현재 수 유지, 수동 스케일링, 예약 스케일링, 동적 스케일링, 예측 스케일링

시작 구성

* AMI, 인스턴스 타입, IAM 역할, 보안그룹, EBS 볼륨

Auto scaling group

* VPC, 서브넷, 로드밸런서
* Vpc > private subnet > auto scaling

|  |  |
| --- | --- |
| 현재 수 유지 | 인스턴스 상태 확인 |
| 수동 스케일링 | 최소, 최대, 원하는 용량 |
| 예약 스케일링 | 미리 정의된 일정 따라 스케일링 |
| 동적 스케일링 | 스케일링 자동으로 조정하는 규칙, 액션 정의 |
| 예측 스케일링 | 수요 따라 자동 스케일링 |

구조

Elastic load balancer -> cloud watch -> auto scaling -> Elastic load balancer …

14주차 sagemaker(머신러닝 관련, 완전 관리형)

데이터 준비(데이터 탐색, 전처리), 모델 훈련, 모델 호스팅, 추론에 필요한 모든 단계 지원

**기본 훈련 컨테이너(학습 컨테이너 환경, 학습 모듈 관련 요소 도커 이미지로 패키징해 사용)**

1. 사용하려면 도커 이미지 먼저 생성

\*도커이미지: 학습에 필요한 코드, 데이터, 라이브러리 등

2. 학습 스크립트를 ENTRYPOINT로 지정

3. 컨테이너 초기화 된 이후 자동 실행

4. 컨테이너 시작시 sagemaker가 컨테이너 환경 설정, 입력 데이터 마운트해 학습 수행

5. ENTRYPOINT로 설정된 학습 스크립트는 데이터 활용해 모델 학습 진행

+) 독립적 환경 학습 실행, 필요시에 커스텀 도커 이미지 생성, 사용

**스크립트 모드 컨테이너(Training Toolkit 포함된 도커 이미지)**

사용 이전에 도커 이미지 필요, 여기에 training toolkit 포함

특정 폴더 내의 학습 스크립트 선택해 실행

학습 스크립트는 특정 경로인 /opt/ml/code에 다운로드되고 실행

학습 스크립트는 S3에 업로드

**해당 모드 특정 프레임워크 사용해 모델 학습(텐서플로우, 파이토치)**

**Training toolkit -> 모델 학습 지원, 하이퍼파라미터 튜닝, 분산 학습**