

AWS

[주요 개념](#)

[AWS란?](#)

[AWS 기본 구성](#)

[서비스 사용 순서](#)

[AWS 사용 순서](#)

[온프레미스, 클라우드, 하이브리드](#)

[Region과 가용영역\(AZ:Availability Zone\)](#)

[Region](#)

[AZ\(Availability Zone\)](#)

[EC2 \(Amazon Elastic Compute Cloud\)](#)

[EC2 인스턴스](#)

[EC2 주요 구성 요소](#)

[EC2의 주요 기능](#)

[EC2 인스턴스 생성 과정](#)

[EC2의 IAM 역할](#)

[IAM\(Identity and Access Management\)](#)

[IAM의 목적](#)

[EC2와 IAM의 관계](#)

[EC2에서 IAM 기능](#)

[EC2에서 IAM 구성 요소](#)

[EC2에서 IAM 역할 사용하는 방법](#)

[IAM 과 EC2 주요 장점](#)

[NACL\(Network Access Control List\)](#)

[NACL 주요 특징](#)

[NACL과 Security Group 비교](#)

[S3\(Simple Storage Service\)](#)

[S3 특징](#)

[S3 객체 스토리지 서비스\(파일을 다루는 서비스\)](#)

[주요 특징](#)

[Block Storage와 Object Storage의 차이](#)

[S3 API](#)

[S3 객체의 구성](#)

[S3 버저닝](#)

[Static Hosting](#)

[일관성 모델](#)

[S3 Cross Region Replication](#)

[AWS 계정 및 초기 설정](#)

[MFA\(Multi-Factor Authentication\) 할당](#)

[MFA를 사용하는 이유](#)

[MFA 할당 과정](#)

[과금 정보 및 비용 관리](#)

[실습](#)

[1. IAM - ID 및 액세스 관리](#)

[1\) IAM 사용자 및 그룹 실습](#)

[로그인하기](#)

[EC2 - Elastic 컴퓨팅 클라우드](#)

[AWS 예산 설정](#)

[EC2에 도커 설치](#)

[최신 상태 가져오기](#)

[도커 설치](#)

[도커 실행하기](#)

[nginx 설치하기](#)

[nginx 실행하기](#)

▼ 주요 개념

- AWS 주요 개념

☒ Region

- AWS 데이터 센터가 위치한 물리적 지역.
- 예: `us-east-1` (버지니아 북부), `ap-northeast-2` (서울).
- 선택 시 네트워크 지연과 서비스 가용성을 고려해야 함.
-

☒ Availability Zone

- 한 지역(Region) 내에 여러 개의 데이터 센터.
- 같은 리전에 속해 있지만 물리적으로 독립되어 장애에 대비.

☒ IAM(Identity and Access Management)

- 사용자, 그룹, 권한을 관리하는 서비스.
- EC2 인스턴스 접근이나 S3 버킷 접근 권한 관리에 사용.
-

☐ VPC(Virtual Private Cloud)

- AWS 리소스를 격리된 네트워크 환경에서 실행하도록 설정.
- EC2는 반드시 VPC 내부에 생성됨.

• AWS 핵심 서비스

☒ EC2 (Elastic Compute Cloud)

- 가상 서버를 제공하는 AWS의 핵심 서비스.
- 서버의 CPU, 메모리, 스토리지를 선택하여 원하는 환경을 구성 가능.
- 온디맨드, 예약 인스턴스, 스팟 인스턴스 등 과금 방식 다양.
-

☒ S3(Simple Storage Service)

- 파일을 저장하기 위한 객체 스토리지 서비스.
- 정적 웹사이트 호스팅, 백업, 로그 저장 등에 사용.
-

☐ RDS(Relational Database Service)

- MySQL, PostgreSQL, MariaDB, Oracle, SQL Server 등 관리형 데이터베이스 제공.
- 백업 및 복구, 스케일링이 간편

☐ ELB(Elastic Load Balancer)

- 트래픽을 여러 EC2 인스턴스로 분산하는 서비스.
- 무중단 배포 시 필수적.

☐ Route 53

- DNS 서비스. 도메인 이름을 EC2, S3 버킷 등과 연결.

☒ CloudWatch

- AWS 리소스의 모니터링 및 로그 관리.
- EC2의 CPU 사용량, 메모리 사용량 등을 추적 가능.

• EC2 이해를 위한 배경 지식

◦

☒ AMI(Amazon Machine Image)

- EC2 인스턴스의 운영 체제와 기본 소프트웨어가 포함된 템플릿.
- 사용자는 AMI를 선택해 인스턴스를 생성.

☒ Instance Type

- EC2의 하드웨어 스펙 선택.
- 예: `t2.micro` (소규모 테스트용), `m5.large` (일반적인 컴퓨팅).

☒ Key Pair

- EC2 인스턴스에 SSH로 접속하기 위한 보안 키.
- Key 파일은 인스턴스 생성 시 다운로드 후 보관 필수.

☒ Security Group

- EC2 인스턴스의 방화벽 역할.
- 허용된 IP 및 포트만 접근 가능하게 설정.

☒ Elastic IP

- 고정 IP 주소를 EC2 인스턴스에 할당.
- 인스턴스를 재부팅해도 변경되지 않는 IP 필요 시 사용.

- 배포와 관련된 추가 개념

☐ Auto Scaling

- 트래픽에 따라 EC2 인스턴스를 자동으로 증가/감소.
- 비용 최적화와 장애 대응에 유용.

☒ EBS(Elastic Block Store)

- EC2 인스턴스에 연결되는 스토리지.
- 디스크처럼 사용하며, 데이터 영구 보관 가능.
-

☐ CloudFormation

- 인프라를 코드로 관리(IaC). EC2, VPC 등 리소스를 템플릿으로 정의 가능.

<https://velog.io/@ghldjfldj/AWS-SG%EB%9E%80-%EB%AC%B4%EC%97%87%EC%9D%B8%EA%B0%80>

AWS란?



아마존에서 온프레미스를 클라우드 서비스로 만든 것

AWS 기본 구성

서비스 사용 순서

- 특정 서비스에 연결하기 위해 도메인 질의
- DNS 서비스가 응답
- 해당 도메인을 IP로 변환하여 서버에 연결
- 서버 디스크의 데이터 읽기
- 요청 시 디비에서 데이터 읽기
- 스토리지에 저장된 이미지나 첨부파일 다운 받아 사용

AWS 사용 순서

- Route53이 도메인에 대한 응답 (DNS 포트가 53)
- 서버 EC2로 접속
- 로컬 디스크인 EBS에서 데이터를 읽는다.
- 요청 시 Amazon DB에서 데이터 읽기
- S3(이미지 저장소)에 저장

온프레미스, 클라우드, 하이브리드

- **온프레미스** : 일반 데이터 센터로 물리 서버로 운영하는 데이터 센터
개인이나 단체가 자체적으로 서버를 구입하여 설치하고 운영하는 방식으로 원격 환경에서 서버를 운영하는 클라우드와는 반대 개념
- **하이브리드** : 클라우드와 온프레미스를 같이 운영
- **클라우드** : AWS, AZURE, GCP, Naver, NHN 등의 유료 서비스를 사용하는 경우
- **멀티 클라우드** : AWS, AZURE, GCP 등 여러개 퍼블릭 클라우드를 사용하는 경우

온프레미스	클라우드
방화벽	보안그룹
ACL	NACL
관리자 권한	IAM
L4, 로드 밸런서	ELB, 탄력적인 로드 밸런서, ALB, NLB, GWLB
네트워크	VPC
서버	EC2
NAS	EFS(탄력적인 파일 시스템)
디스크	EBS
DB	RDS(관계형 데이터 베이스)

Region과 가용영역(AZ:Availability Zone)

클라우드의 리소스는 여러 물리적 위치에서 호스팅되며, 위치는 리전과 가용 영역으로 구성

⇒ 리소스를 특정 리전의 서로 다른 가용영역에 배치하는 경우 인프라, 하드웨어, 소프트웨어 등에서 발생할 수 있는 다양한 장애 상황의 위험을 줄일 수 있다.

또한 더욱 높은 수준의 보안을 달성할 수 있다.

Region

리소스를 호스팅할 수 있는 특정한 지리적 위치 (리전은 여러 개의 AZ로 구성)

- 인프라를 지리적으로 나누어 배포한 것을 의미
- 나라별 단위

AZ(Availability Zone)

리전 내에서 물리적으로 격리된 공간

- 개별 데이터 센터로 구성되어 있기 때문에 리전이 동일하더라도 AZ가 다르다면 복수의 데이터 센터를 사용하는 것이다.
- 동일 리전 내의 복수의 AZ는 물리적으로 격리되어 있지만 고속 프라이빗 네트워크를 통해 논리적으로 연결되어 있다.

- HA(High Availability)를 위해 리전 내 복수의 가용 영역에 데이터와 애플리케이션을 배포하는 것이 좋다
 - 하나의 AZ에 문제가 생겨도 다른 AZ에서 서비스가 중단없이 유지될 수 있도록 함.
- **Elastic IP** : AWS에서 제공하는 Static IP Address service
AWS 계정에 할당되며 릴리즈할 때까지 할당된 상태로 유지
 - 장애가 발생한 AZ로부터 다른 영역으로 인스턴스 주소를 신속히 변경 / 매핑 가능
 - 도메인이 인스턴스를 가리키도록 도메인에 대한 DNS 레코드에 탄력적 IP 주소를 지정가능

EC2 (Amazon Elastic Compute Cloud)

가상 서버를 구축하고 보안 및 네트워크 구성과 스토리지 관리가 가능한 서비스 Amazon Web Services(AWS) 클라우드에서 온디맨드 확장 가능 컴퓨팅 용량을 제공

- 장점
 - 하드웨어 비용 절감 ⇒ 애플리케이션을 더욱 빠르게 개발/배포 가능
 - auto scaling 지원

EC2 인스턴스

AWS 클라우드의 가상 서버

- 시작 시 지정하는 인스턴스 용역에 따라 인스턴스에 사용할 수 있는 하드웨어결정
 - 이때 인스턴스 유형에 따라 서로 다른 컴퓨팅, 메모리, 네트워크 및 스토리지 리소스의 균형을 제공
- 물리적인 서버를 직접 관리하지 않고 가상 서버 생성/관리 가능
- 필요에 따라 확장(스케일링) 가능
- **Elastic** : 필요에 따라 컴퓨팅 리소스 조정 가능

EC2 주요 구성 요소

1. **AMI (Amazon Machine Image)**

EC2 인스턴스를 생성할 때 사용할 운영 체제 및 소프트웨어가 포함된 템플릿

- 종류
 - AWS 제공 AMI : Amazon Linux, Ubuntu, Windows
 - 사용자 정의 AMI : 사용자가 직접 운영체제 설치하고 필요한 소프트웨어 구성한 후 AMI로 저장
 - Marketplace AMI : 외부에서 제공하는 유료/무료 AMI
- 2. **인스턴스 유형**

인스턴스의 하드웨어 사양 선택

- 주요 카테고리
 - 범용
 - 컴퓨팅 최적화
 - 메모리 최적화
 - 스토리지 최적화
 - GPU 최적화
- 3. **Storage**

- **EBS(Elastic Block Store)**
 - EC2에 연결된 디스크 역할
 - 인스턴스 중지/재부팅 시에도 데이터 유지
- **인스턴스 스토어**
 - EC2의 임시 저장소
 - 인스턴스 중지 시 데이터 사라짐.

4. Security Group (SG)

| EC2 인스턴스 규칙에 접근할 수 있는 가 방화벽 역할

- 서버넷 수준이 아니라 인스턴스에 접근할 수 있는 프로토콜, 포트, IP 범위를 설정.
- 규칙
 - **인바운드 규칙** : 외부에서 EC2로 들어오는 트래픽 허용 (SSH, HTTP, HTTPS)
 - **아웃바운드 규칙** : EC2에서 외부로 나가는 트래픽 허용 설정
- 특징
 - - Network Access Control List(NACL)와 함께 방화벽 역할을 하는 서비스

5. Elastic IP

- 고정된 퍼블릭 IP 주소
- EC2 인스턴스를 재부팅하거나 중단해도 변경되지 않음.

EC2의 주요 기능

1. Auto Scaling

- 트래픽에 따라 EC2 인스턴스를 자동으로 추가/제거
장애 시 자동 복구, 트래픽 급증에 대응
- 비용 최적화와 고가용성 유지에 도움

2. Elastic Load Balancing(ELB)

- 여러 EC2 인스턴스 간 **트래픽을 자동으로 분산**
- 장애 감지 후 트래픽을 정상 인스턴스로 라우팅 (이때 사용되는게 route 53일까?)

3. Monitoring

- **CloudWatch**를 통해 CPU 사용량, 메모리 사용량 등 성능을 모니터링
- 알람 설정을 통해 리소스 문제를 빠르게 탐지
 - **Cloudwatch** : AWS 리소스와 애플리케이션의 모니터링 및 관리 도구
모니터링(EC2 CPU 사용량 모니터링), 로그 관리, 알람, 대시보드, 이벤트, 자동화 및 작업 등으로 데이터를 수집하고 저장/분석하여 대응(비용 최적화 등)한다.

EC2 인스턴스 생성 과정



1. AMI 선택
2. 인스턴스 유형 선택
3. 스토리지 설정(EBS 또는 인스턴스 스토리지)
4. 보안 그룹 설정 (SSH, HTTP, HTTPS 등의 접근 권한 설정)
5. 키 페어 생성
6. 인스턴스 시작

EC2의 IAM 역할

IAM(Identity and Access Management)

AWS 리소스에 대한 접근 관리 서비스

- EC2를 포함한 모든 AWS 서비스에 대한 접근 권한 관리
- 사용자, 그룹, 역할, 정책을 정의하여 AWS 리소스에 누가, 무엇을 할 수 있는지 제어

IAM의 목적

1. 보안 강화
모든 사용자에게 최소 권한만 부여
2. 중앙 집중식 관리
사용자의 AWS 리소스 접근 권한을 일괄적으로 관리
3. 역할 기반 접근 제어(RBAC)
역할을 기반으로 리소스 접근 권한을 정의
4. 감사 및 추적
누가 어떤 작업을 했는지 연계해 추적가능

EC2와 IAM의 관계

EC2에서 IAM이 중요한 이유



EC2 인스턴스는 AWS의 다른 서비스와 자주 상호작용하며 S3에 파일을 저장하거나 DB와 통신한다. 이때 IAM으로 EC2가 필요한 리소스에만 접근할 수 있도록 권한을 제어하여 액세스 키 없이 EC2가 안전하게 리소스에 접근할 수 있도록 한다.

ex.

- 특정 S3 버킷에 접근해야 하는 경우 IAM역할을 사용해 권한 부여
- EC2가 특정 API를 호출할 수 있도록 제한 설정

IAM이 없다면?

- 사용자나 애플리케이션이 AWS 리소스에 접근하기 위해서 액세스 키를 직접 코드에 포함해야하기 때문에 보안이 취약하고 키 관리가 어려움
- IAM을 사용하여 액세스키 없이 EC2가 안전하게 리소스에 접근할 수 있도록 함.

EC2에서 IAM 기능

IAM 역할

- EC2 인스턴스가 AWS 리소스에 접근할 때 사용하는 **임시 권한**
- EC2 인스턴스에 역할을 연결하면, 그 역할에 정의된 권한만 사용할 수 있다.

정책

- JSON 형식으로 누가, 무엇을, 어떤 조건에서 어떤 리소스에 대해 허용/거부할 것인가를 정의

```
{
  "Version": "2012-10-17",
  "Id" : "", 선택사항
  "Statement": [
    {
      "Sid" : "1", 문장의 식별자 선택사항
      "Effect": "Allow", 문장이 특정 API에 접근하는 것을 허용/거부 deny
      "Principal" : { //선택사
        "AWS" : [] 특정 정책이 적용될 사용자, 계정 혹은 역할로 구성
      },
      "Action": "s3:ListBucket", effect에 의해 허용/거부되는 API의 호출 목록
      "Resource": "arn:aws:s3:::example-bucket" //적용될 action의 리소스 목록
    }
  ]
}
```

EC2와 IAM 역할의 연결

- EC2 생성 시 IAM 역할을 연결하거나 기존 인스턴스에 IAM 역할을 추가
- 역할에 설정된 정책에 따라 EC2가 필요한 리소스에 접근 가능

AWS CLI와 SDK

- EC2에서 AWS CLI(Comand Line Interface)나 SDK를 사용해 AWS 리소스에 접근할 때 IAM 역할을 통해서 인증

EC2에서 IAM 구성 요소

IAM 사용자

- AWS 리소스를 사용할 수 있는 사용자 계정
- IAM 사용자는 직접 EC2 인스턴스를 관리하거나 역할을 부여할 수 있음.

IAM 그룹

- IAM 사용자들을 그룹으로 묶어 동일한 권한 부여

IAM 역할

- EC2 인스턴스에 연결해 AWS 리소스에 대한 권한을 설정

정책

- JSON 형식으로 권한 정의
- 역할, 사용자, 그룹에 연결 가능

EC2에서 IAM 역할 사용하는 방법

1. IAM 역할 생성

- a. AWS 콘솔에서 IAM 서비스로 이동
- b. 역할 생성
 - i. 신뢰할 주체(EC2 서비스) 를 선택
 - ii. 역할에 정책을 연결
- c. 역할 이름 지정 후 생성 완료

2. EC2 인스턴스에 IAM 역할 할당

- a. EC2 생성 시 IAM 역할을 선택
- b. 기존 인스턴스에 IAM 역할 연결
 - i. EC2 콘솔 → 인스턴스 선택 → 작업 → 보안 → IAM 역할 연결/변경

IAM 과 EC2 주요 장점

보안 강화

- 액세스 키 관리 불필요
- 최소 권한 부여로 리소스 접근 제한

자동화

- 역할과 정책 기반으로 EC2의 권한을 자동으로 부여

유연성

- 여러 EC2 인스턴스에서 동일한 IAM 역할을 공유 가능

감사 및 추적

- CloudTrail을 통해 EC2가 어떤 리소스에 접근했는지 추적 가능

NACL(Network Access Control List)

AWS의 VPC(Virtual Private Cloud) 내에서 네트워크 계층에서 트래픽을 제어하는 방화벽 역할
서브넷 수준에서 동작하며, 들어오거나 나가는 네트워크 트래픽에 대한 허용 또는 거부 규칙 설정

NACL 주요 특징

1. 서브넷 수준에서 적용
특정 서브넷에 연결된 모든 인스턴스에 대해 네트워크 트래픽 제어
2. 규칙 기반
숫자로 된 규칙번호를 기반으로 동작하며, 낮은 번호의 규칙이 우선적으로 적용됨.
3. 상태 비저장(State-less)

인바운드 와 아웃바운드 트래픽에서 별도로 허용하지 않으면 반대 방향 트래픽이 차단
4. 서브넷에 기본적으로 할당
모든 서브넷은 기본적으로
NACL 을 사용하며 이 기본 NACL은 모든 트래픽을 허용
5. 여러 규칙 지원
IP범위, 포트, 프로토콜에 따라 다양한 규칙 정의 가능

NACL과 Security Group 비교

특징	NACL (Network ACL)	Security Group
적용 범위	서브넷 단위	개별 인스턴스 단위
상태 관리	상태 비저장 (Stateless)	상태 저장 (Stateful)
허용/거부	허용(Allow) 및 거부(Deny) 규칙 가능	허용(Allow) 규칙만 가능
규칙 순서	규칙 번호 순으로 처리 (낮은 번호가 우선)	규칙 순서 없음 (모두 적용)
트래픽 방향	인바운드/아웃바운드 규칙 각각 정의 필요	인바운드/아웃바운드 트래픽 자동 대응 (Stateful)
적용 리소스	서브넷 내 모든 리소스	개별 인스턴스

S3(Simple Storage Service)

확장성, 데이터 가용성 및 보안과 성능을 제공하는 객체 스토리지 서비스

- 사용하기 쉬운 관리 기능을 제공하여 특정 비즈니스 조직 및 규정 준수 요구 사항에 따라 데이터를 조직화, 세부적인 액세스 제어 구성 가능

S3 특징

- 객체 스토리지 서비스
- AWS에서 제공하는 파일 저장 서비스
- Bucket이라는 단위로 구분 가능
- 버전 관리 가능
- Static Web 호스팅 기능
- 업로드와 업데이트 / 삭제의 데이터 일관성 모델이 다름.
- 암호화 기능
- 보안 설정 가능
- 다른 유저 / 다른 계정과 공유가능
- 다른 Region으로 복제 가능
- 수명주기 설정 가능
- 삭제 방지 기능(MFA)
- 파일 전송 Transfer Acceleration 가능

S3 객체 스토리지 서비스(파일을 다루는 서비스)

데이터를 객체(object) 단위로 저장하는 스토리지 방식

- 각 데이터를 포함한 파일과 함께 메타데이터와 고유 식별자로 구성
- 객체 스토리지 서비스의 경우 Block Storage Service(EBS, EFS)와 달리 파일만 가능하다

▼ Block Storage Service

- 개념
 - 데이터를 블록 단위로 저장
 - 각 블록은 고유한 주소를 가지고 있으며, 데이터의 특정 부분만 읽고/쓰기가 가능하다.
 - ex. 디스크 드라이브(HDD, SSD)와 유사하며 OS와 연결하여 마치 물리적 디스크처럼 사용 가능
- 대표 서비스
 - EBS(Elastic Block Store)
EC2 인스턴스에 연결할 수 있는 블록 스토리지로, 고성능과 낮은 지연 시간의 스토리지가 필요한 애플리케이션에서 사용
- 특징
 - a. 저지연 및 고성능
읽기 / 쓰기 속도가 빠르고, 데이터베이스와 같은 IOPS(Intensive Input/Output) 작업에 최적화
 - b. 인스턴스 연결
 - c. 데이터 지속성
EC2가 중단되더라도 데이터는 EBS 볼륨에 저장되어 유지
 - d. 파일 시스템 지원

- 파일 설치 불가
이미지, 동영상, 워드 등 업로드, 삭제, 업데이트만 가능
- 무제한 용량(무한한 확장성 제공)
- 파일을 저장하고 검색하는데 최적화되어 있으며, 인터넷 기반의 대규모 데이터 저장에 적합

주요 특징

- 확장성 및 내구성
데이터를 여러 가용 영역에 복제하여 높은 내구성을 제공하며 무제한 데이터 저장이 가능하다
- HTTP/HTTPS 프로토콜
데이터를 인터넷을 통해 직접 접근 가능
- 메타데이터
각 객체는 키와 함께 메타데이터를 가지고 있어 관리 및 검색이 용이
- 비용 효율성
저장 공간이 증가할수록 GB 당 비용이 저렴
- 데이터가 계층적이 아닌 Flat한 (버킷 - 객체)로 구성

Block Storage와 Object Storage의 차이

특징	Block Storage (EBS)	Object Storage (S3)
데이터 저장 방식	데이터를 블록 단위로 저장	데이터를 객체 단위로 저장
접근 방식	OS를 통해 디스크처럼 마운트 후 사용	HTTP/HTTPS를 통해 접근
유연성	특정 데이터 블록을 직접 수정 가능	전체 객체를 업데이트해야 함
속도 및 성능	높은 IOPS를 제공하며, 읽기/쓰기 속도가 빠름	대규모 데이터를 저장하고 읽는 데 최적화
사용 사례	데이터베이스, 파일 시스템, 애플리케이션 데이터 저장	백업, 로그 파일 저장, 정적 웹사이트 호스팅
확장성	고정된 크기로 프로비저닝 필요	거의 무제한 확장 가능
비용	상대적으로 고비용	대용량 데이터 저장 시 더 경제적
데이터 구조	디스크 파티션과 비슷한 계층적 구조	Flat 구조 (버킷-객체)
연결성	EC2와 같은 특정 인스턴스에 연결	네트워크를 통해 글로벌 접근 가능
내구성	특정 AZ에 저장, 데이터 손실 가능성 있음	여러 AZ에 데이터 복제, 높은 내구성 제공

S3 API

- 파일 관련 API(업로드, 업데이트, 삭제)
 - 큰 용량의 파일의 경우 Multipart Upload 가능(큰 파일을 잘게 쪼갬 뒤 하나씩 업로드한 후 S3에서 합침)
- 업로드가 중간에서 중지되더라도 나중에 이어서 업로드 가능
- 분산에서 업로드하기 때문에 속도가 빠름
- Bucket 관련 API(조회, 생성, 삭제)

S3 객체의 구성

- key, value, Version Id, Metadata, ACL, Torrents

S3 버저닝

- 모든 버전을 관리(삭제 포함)
- 기본적으로 비활성화되어있기 때문에 활성화 필요
 - 대신 한번 활성화 시 비활성화 불가능
- 수명주기 관리와 연동 가능

Static Hosting

- 정적인 데이터를 웹서버처럼 돌릴 수 있음.
- Html/JS 등으로 구성된 Static 사이트만 가능 - react
- 호스팅 비용이 비교적 저렴
- Serverless 구성의 기초
- 동적 데이터는 AJAX 등으로 해결 가능

일관성 모델

- **Put** (새로 생성) : 읽기 후 쓰기
 - 파일을 올리고 성공한 즉시 읽기 가능
 - 먼저 Put한 요청이 우선권을 가진
- **Update / Delete** : 최종 일관성
 - 파일을 삭제하거나 업데이트 후 일정시간 후에 결과 반영
 - 원자성 확보 불가능

S3 Cross Region Replication

- 다른 Region으로 복제 가능
- 버저닝이 활성화되어 있어야 가능
- 동일한 Region으로 복제 불가능
- 복제 기능 활성화 전 데이터는 복제되지 않

AWS 계정 및 초기 설정

- ▼ [gpt] spring boot gradle 사용해 aws sdk로 개발하기 예제

만약 Gradle을 사용할 경우, AWS SDK for Java를 설정하려면 `build.gradle` 파일에 의존성을 추가해야 합니다. Gradle 기반으로 Spring Boot 프로젝트에서 AWS SDK를 사용하는 방법과 예제를 아래에 설명드릴게요.

1. Gradle을 사용한 AWS SDK 설정

1.1. `build.gradle` 에 의존성 추가

AWS SDK의 모듈별로 필요한 서비스를 추가합니다. 예를 들어, S3와 EC2를 사용하려면 다음과 같이 설정합니다.

```
groovy
코드 복사
dependencies {
    implementation 'software.amazon.awssdk:s3:2.20.0' // S3 SDK
    implementation 'software.amazon.awssdk:ec2:2.20.0' // EC2 SDK
}
```

1.2. Gradle 동기화

의존성을 추가한 후, Gradle 동기화를 실행합니다.

```
bash
코드 복사
```

```
./gradlew build
```

2. AWS S3 사용 예제

2.1. S3 클라이언트 설정

Gradle로 설정한 AWS SDK를 사용해 Spring Boot 프로젝트에서 S3 클라이언트를 생성합니다.

```
java
코드 복사
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AwsConfig {

    @Bean
    public S3Client s3Client() {
        return S3Client.builder()
            .region(Region.AP_NORTHEAST_2) // 서울 리전
            .credentialsProvider(DefaultCredentialsProvider.create()) // IAM 인
증
            .build();
    }
}
```

2.2. S3 파일 업로드 로직

S3에 파일을 업로드하는 서비스 로직을 작성합니다.

```
java
코드 복사
import org.springframework.stereotype.Service;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;

import java.io.File;
import java.nio.file.Paths;

@Service
public class S3Service {

    private final S3Client s3Client;

    public S3Service(S3Client s3Client) {
        this.s3Client = s3Client;
    }

    public void uploadFile(String bucketName, String keyName, String filePath) {
        File file = new File(filePath);
    }
}
```

```

        s3Client.putObject(
            PutObjectRequest.builder()
                .bucket(bucketName)
                .key(keyName)
                .build(),
            Paths.get(file.getPath())
        );

        System.out.println("File uploaded to S3: " + keyName);
    }
}

```

2.3. API Endpoint 구현

Spring Boot의 Controller를 통해 파일 업로드 API를 제공합니다.

```

java
코드 복사
import org.springframework.web.bind.annotation.*;
import org.springframework.http.ResponseEntity;

@RestController
@RequestMapping("/s3")
public class S3Controller {

    private final S3Service s3Service;

    public S3Controller(S3Service s3Service) {
        this.s3Service = s3Service;
    }

    @PostMapping("/upload")
    public ResponseEntity<String> uploadFile(@RequestParam String bucketName, @RequestParam String keyName, @RequestParam String filePath) {
        s3Service.uploadFile(bucketName, keyName, filePath);
        return ResponseEntity.ok("File uploaded successfully!");
    }
}

```

3. AWS EC2 사용 예제

3.1. EC2 클라이언트 설정

AWS EC2 클라이언트를 생성하는 코드를 작성합니다.

```

java
코드 복사
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

```

```

@Configuration
public class AwsConfig {

    @Bean
    public Ec2Client ec2Client() {
        return Ec2Client.builder()
            .region(Region.AP_NORTHEAST_2) // 서울 리전
            .credentialsProvider(DefaultCredentialsProvider.create()) // IAM 인
증
            .build();
    }
}

```

3.2. EC2 인스턴스 조회 서비스

EC2 인스턴스 목록을 가져오는 기능을 구현합니다.

```

java
코드 복사
import org.springframework.stereotype.Service;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.DescribeInstancesRequest;
import software.amazon.awssdk.services.ec2.model.DescribeInstancesResponse;
import software.amazon.awssdk.services.ec2.model.Reservation;

import java.util.List;

@Service
public class EC2Service {

    private final Ec2Client ec2Client;

    public EC2Service(Ec2Client ec2Client) {
        this.ec2Client = ec2Client;
    }

    public List<Reservation> listInstances() {
        DescribeInstancesRequest request = DescribeInstancesRequest.builder().build
();
        DescribeInstancesResponse response = ec2Client.describeInstances(request);
        return response.reservations();
    }
}

```

3.3. API Endpoint 구현

Spring Boot Controller에서 EC2 인스턴스 정보를 반환하는 API를 구현합니다.

```

java
코드 복사
import org.springframework.web.bind.annotation.*;
import software.amazon.awssdk.services.ec2.model.Reservation;

```

```
import java.util.List;

@RestController
@RequestMapping("/ec2")
public class EC2Controller {

    private final EC2Service ec2Service;

    public EC2Controller(EC2Service ec2Service) {
        this.ec2Service = ec2Service;
    }

    @GetMapping("/instances")
    public ResponseEntity<List<Reservation>> getInstances() {
        List<Reservation> instances = ec2Service.listInstances();
        return ResponseEntity.ok(instances);
    }
}
```

4. AWS 인증 방식

4.1. DefaultCredentialsProvider 사용

- AWS SDK는 기본적으로 IAM 역할이나 환경 변수에서 AWS 자격 증명(Access Key, Secret Key)을 자동으로 가져옵니다.
- Spring Boot 애플리케이션에서 보안을 위해 IAM 역할 또는 AWS CLI로 설정된 자격 증명을 사용하는 것이 좋습니다.

4.2. 자격 증명 설정 파일

- 로컬 개발 시 `~/.aws/credentials` 파일을 설정합니다:

```
java
코드 복사
[default]
aws_access_key_id = YOUR_ACCESS_KEY
aws_secret_access_key = YOUR_SECRET_KEY
```

4.3. 환경 변수

- 환경 변수로 AWS 자격 증명을 설정할 수도 있습니다:

```
bash
코드 복사
export AWS_ACCESS_KEY_ID=YOUR_ACCESS_KEY
export AWS_SECRET_ACCESS_KEY=YOUR_SECRET_KEY
```

5. Gradle 프로젝트에서 AWS SDK 활용의 장점

1. 간소화된 개발: SDK가 복잡한 API 요청을 간단한 메서드 호출로 추상화.
2. 유연한 구성: Spring Boot의 DI(Dependency Injection)를 활용한 AWS 클라이언트 관리.

3. 안전한 인증: DefaultCredentialsProvider로 IAM 역할 기반 인증 가능

MFA(Multi-Factor Authentication) 할당

AWS 계정에 로그인하거나 중요한 작업을 수행할 때, 비밀번호만으로 로그인하는 대신 추가적인 인증 단계를 요구하는 방식

MFA를 사용하는 이유

- 보안 강화
- 이중 보호
- AWS 계정 보호

MFA 할당 과정



1. AWS Management console 로그인
2. IAM 서비스로 이동
3. MFA 디바이스 할당
인증 관리자 앱이 가장 범용적
4. MFA 디바이스 할당
5. MFA 디바이스 등록
6. MFA 활성화 완료

과금 정보 및 비용 관리

과금 부과 전 과금 내역을 알 수 있는 아주아주아주 중요한 내용

사용자 설정 - 과금 정보 및 비용 관리 - 좌측 메뉴의 예산 클릭 - 예산 생성 - 템플릿 사용하여 예산 생성 - admin 계정 사용(root 계정이 아닌 IAM 계정 사용)

실습

1. IAM - ID 및 액세스 관리

1) IAM 사용자 및 그룹 실습

- 주의사항
IAM은 글로벌 서비스이며 따라서 선택할 리전이 없음.(리전 비활성화)
⇒ 즉 IAM 에서 사용자 생성 시 어디에서나 사용가능함
- IAM을 만들기 전에 사용하는 user는 바로 root 루저이다. (우측 상단 내 별명을 누르면 사용 가능한 계정임)
- 절대적으로 root 말고 계정을 더 안전하게 만들어 관리자 사용자 같은 user를 생성해야한다.

사용자 세부 정보

사용자 이름

stephane

사용자 이름은 최대 64자까지 가능합니다. 유효한 문자: A~Z, a~z, 0~9 및 +, -, @, _ (하이픈)

☒ **AWS Management Console에 대한 사용자 액세스 권한 제공 - 선택 사항**
 사람에게 콘솔 액세스 권한을 제공하는 경우 IAM Identity Center에서 액세스를 관리하는 것은 **모범 사례**입니다.

① 사람에게 콘솔 액세스 권한을 제공하고 있습니까?

사용자 유형

☐ **Identity Center에서 사용자 지정 - 권장**
 Identity Center를 사용하여 사람에게 콘솔 액세스 권한을 제공하는 것이 좋습니다. Identity Center를 사용하면 AWS 계정 및 클라우드 애플리케이션에 대한 사용자 액세스를 중앙에서 관리할 수 있습니다.

☒ **IAM 사용자를 생성하고 실행**
 액세스 키, AWS CodeCommit이나 Amazon Keyspaces에 대한 서비스별 보안 인증 정보 또는 비동기 액세스를 위한 백업 보안 인증 정보를 통해 프로그래밍 방식 액세스를 활성화해야 하는 경우에만 IAM 사용자를 생성하는 것이 좋습니다.

왼쪽의 부분에서 Identity Center가 권장이지만 iam 사용자가 좀 더 간단

콘솔 암호

☒ **자동 생성된 암호**
 사용자를 생성한 후 암호를 볼 수 있습니다.

☐ **사용자 지정 암호**
 사용자의 사용자 지정 암호를 입력합니다.

☐ **암호 표시**

☒ **사용자는 다음 로그인 시 새 암호를 생성해야 합니다 - 권장**
 사용자는 IAMUserChangePassword 정책을 자동으로 가져와 암호를 변경할 수 있도록 허용합니다.

- 우측의 경우 이 사용자의 계정이 내가 사용할 계정이 아니라면 우측처럼 설정을 해서 다음에 비밀번호를 변경할 수 있도록 하는 것이 맞아

태그 - 선택 사항
 태그는 리소스를 식별, 구성 또는 검색하는 데 도움이 되도록 AWS 리소스에 추가할 수 있는 키 값 페어입니다. 이 사용자와 연결할 태그를 선택합니다.

키

Q Department X

값 - 선택 사항

Q Engineering X

삭제

새 태그 추가
 최대 49개의 태그를 더 추가할 수 있습니다.

우측의 태그는 나중에 메타데이터를 통해 검색을 수월하게 하게 할 수 있어

지금은 stephane이라는 사람의 부서가 기술부라고 작성을 해주자 이렇게 할 수 있

⇒ 이렇게 계정이 생성이 되었다면 root 계정 말고 계정을 변경하여 AWS 를 살펴보는 것을 할 수 있어

로그인하기

대시보드 - 원하는 계정 선택 - AWS 계정 부분을 보고 accout ID랑 로그인 URL 확

EC2 - Elastic 컴퓨팅 클라우드

AWS 예산 설정

인스턴스 서버 한대당 인스턴스 하나

ec2의 경우 750 시간 사용량 초과 시 초과분에 대한 과금이 발생함.

초과 / 과금 되었을 때 aws 고객 지원 센터에 전화해서 환불 요청이 가능함.

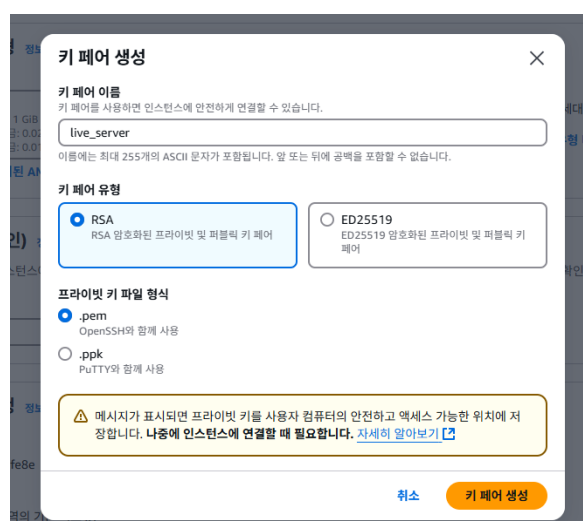
인스턴스 유형 - pc 사양

키페어 -

가상의 pc를 aws 서버에 있는 이 가상의 pc에 접속해서 배포를 하기 위해서는 아무나 막 들어갈 수 있는건 아니고 우리가 이 키 값을 발급을 받아가지고 문을 열고 들어갈 거다.

이 키는 아주 중요해

한번 다운로드 받으면 발급받기가 어려움



.pem 키를 사용하자

원격 프로그램에서만 사용이 가능함

pem으로 ppk를 만들 수 있음.

이렇게 만들어진 키 파일은 복사 / 잘라내기 해서 c 드라이브 / 사용자 / user/ .ssh

없으면 .ssh 파일 만들기

보안 그룹이란?

스토리지란?

스토리지 8기가까지로 충분해 일ㄷ

ip와 port 에 대해 완벽 숙지 필요

ip(아파스) port(호수)

pc에 불기 위해 필요한 주소가 ip이다.

문제는 ? ip 주소가 public 이 있고 dns 가 있고 private가 있다.

외부에서 찾아갈 때는 public ip 주소가 있어야 찾아서 들어갈 수 있다.

회사 안에서는 public으로 들어갈 수 있지만 외부에서는 그 사람 이름을 제대로 알려주면 안되서 그때 사용하는게 public 이다.

근데 문제는 aws에서는 기본적으로 public ip 주소가 동적으로 되어 있어

즉, 변할 수 있기 때문에 인스턴스를 꺾다 켜올 때 새로운 public 주소를 발급하게 된다.

이 퍼블릭 ip 주소가 바뀐것을 확인 하고 접속을 해야해

이걸 바로 탄력적 ip 라고 한다.

이 동적 ip 가 계속 바뀌는데 이 바뀌는 동적 ip를 변수라고 생각했을 때

어떤 고정 ip 를 하나 발급 받았을 때 그게 계속 동적 ip를 가리키게 해

그래서 고정 ip 를 가리키면 그게 동적 ip를 가리키게 해 이때 고정 ip가 탄력적 ip 임

그래서 이걸 할당 해야함

이 id를 가지고 인스턴스에 접속할거야

접속 방법

c드라이브 / deploy

```
#!/bin/bash
```

```
ssh -i ~/.ssh/live_server.pem ubuntu@13.124.83.97 //사용자 이름 @ 접속 ip  
//ssh에 접속을 하겠다 이런 뜻임
```

~/ 하면 무조건 user가 된다.

이 폴더에 들어있는 live_server.pem 서버를 사용하겠다. 이런 뜻임.

"C:\deploy\live_server_ssh.sh"

매번 저 작성어를 입력하기귀찮으니까 저걸 저장해두는거지0

이거 실행한게 ec2 실행하고 설정한거

```
$ /c/deploy/live_server_ssh.sh  
The authenticity of host '13.124.83.97 (13.124.83.97)' can't be established.  
ED25519 key fingerprint is SHA256:PMlfYcch8o35Ql/49TmcbGreHKEEWBIlgT42Rdkikk8.  
This key is not known by any other names.  
Are you sure you want to continue connecting (yes/no/[fingerprint])? y  
Please type 'yes', 'no' or the fingerprint: y  
Please type 'yes', 'no' or the fingerprint: yes  
Warning: Permanently added '13.124.83.97' (ED25519) to the list of known hosts.  
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1018-aws x86_64)
```

```
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:        https://ubuntu.com/pro
```

System information as of Wed Jan 15 07:41:27 UTC 2025

System load:	0.0	Processes:	103
Usage of /:	24.7% of 6.71GB	Users logged in:	0
Memory usage:	20%	IPv4 address for enX0:	172.31.42.66
Swap usage:	0%		

Expanded Security Maintenance for Applications is not enabled.

```
0 updates can be applied immediately.
```

Enable ESM Apps to receive additional future security updates.
See <https://ubuntu.com/esm> or run: `sudo pro status`

The list of available updates is more than a week old.
To check for new updates run: `sudo apt update`

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in `/usr/share/doc/*/copyright`.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "`sudo <command>`".
See "`man sudo_root`" for details.

```
ubuntu@ip-172-31-42-66:~$
```

EC2에 도커 설치

최신 상태 가져오기

sudo 명령어

```
ubuntu@ip-172-31-42-66:~$ sudo su
root@ip-172-31-42-66:/home/ubuntu#
```

root로 바뀌는 걸 알 수 있어

`apt-get` : 패키지 관리자임 (maven)

`apt-get update` : 최신 업데이트

`apt-get upgrade` : 업그레이드

도커 설치

아래 페이지 따라 작성

도커 실행하기

docker hub access token 받기

dicker hub에 접속할 비밀번호

현재 비번

`dckr_pat_oqVL9BDv2kITX93hII-QQL7-bgw`

nginx 설치하기

docker hub에 nginx 설정하기

지금 이렇게 설치하는 건 root 계정에서 일때야

```
docker pull nginx
```

nginx 실행하기

```
docker container run --name nginxserver -d -p 80:80 nginx
```

그 결과로

a85a50bec3d89d6211433f4264964f3b45c8bb5518bbbacf061fbf8eeb8842aa

이렇게 나오는데 이미 실행이되고 있다는 뜻이야

```
docker ps
```

상태 확인 가능해

```
root@ip-172-31-42-66:/home/ubuntu# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                    NAMES
a85a50bec3d8   nginx    "/docker-entrypoint..." About a minute ago Up About a minute   0.0.0.0:80->80/tcp, :::80->80/tcp nginxserver
root@ip-172-31-42-66:/home/ubuntu#
```