클라우드 시스템 구축 중간고사 정리

중간25 기말25 과제30 (태도X)

1. w1\_클라우드 시스템 구축

- 클라우드 컴퓨팅, 이점, 서비스유형 4지선다?

w2\_aws\_아키텍쳐

- Region 5p

- 가용영역 7p

- 엣지로케이션 10

- 글로벌 서비스와 리전 서비스 13

- vpc 16

- iam 20

- mfa(다중인증) 적용가능 22p \*\*\*

w3\_ec2

- 엘라스티컴퓨팅클라우드 \*\*\*

- security grioup 20

- ec2 모니터링 26

w4\_aws cli

- 명령어 자체가나옴 (s3에서 데이터 가져오고 데이터 보내고 하는 명령어 같은거)

- 쓸데없는 명령어 안나옴 --> 이게 먼데

w4\_s3\_v2.0

- 9p 에 나오는 명령어 같은걸 외우지 말아라

- s3 보안 11p

- 정적인 호스팅 관련 문제가 나올건데 접근이 안되게 됐을 때 어떻게 접근을 하게 하는가? \*\*\*\*

ㄴ 이런 오류가 발생했는데 문제 해결을 어떻게 해결해야하는가?

ㄴ s3호스팅 정적인 호스팅 같은 문제가 나옴 --> ppt에 있는 문제가 그대로 나올 예정임

w5\_오토스케일링\_v2.0

- 스케일 업 아웃 4,6p

여태 배운 리소스에서 나옴

ㄴ 템플릿

ㄴ 클라우드 와치

ㄴ 시큐리티 그룹

ㄴ ec2, 오토스켈링ㅇㅇ, 등

---> 접근이 안되는데 어떤 서비스로 수정하면 되는가?

ex) 클라우드 , s3 에 대해 옳지 않은 것은?

ex) 한 회사가 ~~ 서비스를 도입하려는데 aws 서비스는 무엇인가?

\*\*\*\*\*\*\*\*\* 공부X

W5 오토스케일링

- 16P 복붙 코드같은거

W4\_AWS\_CLI

- index.php

---> 차라리 12p 를 외워라

클라우드 컴퓨팅 - 인터넷을 통해 IT 자원 요구에 따라 사용한 만큼 비용을 지불하는 서비스

이점 - Agility(민첩성 / 빠르고 혁신적으로 구축)

        - Elastic(탄력성 / 확장가능, 손쉽게 확장 및 축소가능)

        - Coast saving(비용절감 / 사용한 만큼 지불, Case by Case)

        - 손쉬운 글로벌 서비스

        - 예상치 못한 트래픽 폭주 대응

        - 빅데이터, 인공지능 서비스 확장

클라우드 - 인터넷 구간 어딘가에 눈에 보이지 않는 형태로 구성된 IT 자원 집합

서비스 유형(aws의 대표적인 서비스들)

- IaaS(Infrastructure as a Service / 우리가 현재 배우고 있음)

• 서버, 스토리지, 네트워크, 가상화 제공

• 사용자가 OS 및 애플리케이션 관리

• EC2(가상 서버)

• S3(클라우드 스토리지)

• VPC(가상 네트워크)

• EBS(블록 스토리지)

• IaaS 활용 예시 - 스타트업이 자체 서버 없이 EC2와 S3로 웹사이트 호스팅

- PaaS(Platform as a Service)

• 개발 및 배포를 위한 플랫폼 제공

• 서버 및 OS 관리 불필요

• AWS Elastic Beanstalk(자동 배포 및 관리)

• AWS Lambda(서버리스 컴퓨팅)

• Amazon RDS(관리형 데이터베이스 서비스)

• AWS Fargate(컨테이너 관리)

• PaaS 활용 예시 - 개발자가 AWS Lambda를 사용해 서버 관리 없이 코드 실행

- SaaS(Software as a Service)

• 사용자가 소프트웨어를 설치 없이 이용

• 유지보수 및 업데이트 제공자가 수행

• Amazon WorkDocs(문서 관리 및 협업)

• Amazon Chime(아마존 차임, 화상 회의 서비스)

• AWS Managed Services(완전 관리형 서비스)

• SaaS 활용 예시 - 기업이 Amazon Chime을 통해 원격 회의를 진행

• 타 클라우드 - Google Workspace, GitHub, Dropbox, Microsoft365...

온디맨드 (on - demand) - IT자원을 사용자가 필요할 때 원하는 만큼 즉각적으로 제공할 수 있는 것

온프레미스(on-premise: 전통적인 방법) - 자체적인 공간과 자원을 이용하여 사용자가 직접 구축 운영하는 방식 / 사용자 입장에서 공간, 자원 등 모은 것을 자체적으로 구축 및 운영하는 방식

[AWS 클라우드 핵심 개념 및 아키텍처]

<AWS의 핵심 서비스 개요>

- 컴퓨팅 서비스

• EC2 (Elastic Compute Cloud) - 가상 서버, 온디맨드 / 스팟인스턴스

• Lambda - 서버리스 컴퓨팅, 이벤트 기반 실행

- 스토리지 서비스

• S3(Simple Storage Service) - 객체 스토리지, 정적 웹 호스팅

• EBS(Elastic Block Store) - EC2 인스턴스를 위한 블록 스토리지

- 네트워크 및 콘텐츠 전송

• VPC(Virtual Private Cloud) - AWS 내 네트워크 구성

•  Route 53 - 도메인 네임 서비스(DNS)

• CloudFront - CDN(콘텐츠 전송 네트워크)

- 데이터베이스 서비스

• RDS(Relational Database Service) - 관리형 데이터베이스(MySQL, PostgreSQL, MariaDB 등)

• DynamoDB - NoSQL, 키-값 저장소

- 보안 및 접근관리

• IAM(Identity and Access Management) - 사용자 및 권한 관리

• Security Group(보안 그룹) - 인바운드 및 아웃바운드 트래픽 제어

<AWS 구조>

리전(Region) - 가장 큰 단위, 리전 안에 다양한 가용영역이 존재함

- AWS의 서비스가 제공되는 서버의 물리적인 위치

- 전 세계의 흩어져 있음. 큰 구분(동남아, 유럽, 북아메리카 등)으로 묶여 있음

- 각 리전에는 고유한 코드가 부여됨

- 예) 서울: ap-northeast-2

- 미국동부(버지니아북부) 리전 -> us-east-1, 뭐든지 최초

- 리전별 가능한 서비스가 다름

- 리전을 선택할때 고려할 점

• 지연속도 ex) 북미서버면 북미리전

• 법률(데이터, 서비스 제광 관련) ex) 개인정보는 그 나라에 서버에 둬야한다...등등

• 사용가능한 AWS 서비스 - 각 리전별로 런칭이 다름

- us-east-1 리전

• 모든 AWS의 서비스가 최초로 서비스되는 리전

• 기타 글로벌 서비스의 서비스 리전 ex) 빌링, 클라우드 프론트(CDN)

<가용영역(Avilability Zone)>

- 리전의 하부 단위

• 하나의 리전은 2개 이상의 가용영역으로 구성

• AZ라고 부름

• 1개 이상의 물리 데이터 센터를 묶은 논리적 데이터 센터

- 가용영역 구성

• 하나이상의 데이터 센터로 구성

• AZ간의 연결은 매우 빠른 전용 네트워크로 구성

• 반드시 물리적으로 일정 거리 (몇 km 이상) 떨어져 있음

• 다만 모든 AZ는 서로 100km이내의 거리에 위치

• 여러 재해에 대한 대비 및 보안

- 가용영역의 위치

• 각 계정별로 AZ의 코드와 실제 위치는 다름

          -> 계정 dev1의 AZ-A와 계정 dev2의 AZ-A와 다른 위치(랜덤)

          -> 보안 이슈, 한 AZ로 몰림 현상 방지

<엣지로케이션(Edge location)>

- AWS의 CloudFront(CDN)등의 여러 서비스들을 가장 빠른 속도로 제공(캐싱)하기 위한 거점

- Global Accelerator와 유저를 연결하는 거점

• 출장소 같은 개념보안 이슈, 한 AZ로 몰림 현상 방지

- 전 세계 여러장소에 흩어져 있음

- CDN(Contents Delivery Network) - 컨텐츠를 보다 빠르게 전송하는 기술

• 미국에 있는 넷플릭스를 한구에서 본다면?

• 아프리카에 있는 트위치를 한국에서 본다면? -> 엣지로케이션이 없다면? -> 느림, 스트리밍 불가! -> 엣지로케이션 탄생!

- 빠르고 부하분산

<글로벌 서비스와 리전 서비스>

- 서비스 종류

• AWS에는 서비스가 제공되는 지역의 기반에 따라 글로벌 서비스와 리전 서비스로 분류

- 글로벌 서비스

• 데이터 및 서비스를 전 세계의 모든 인프라가 공유

• ex) IAM, Route53, WAF, Billing

- 지역 서비스

• 특정 리전을 기반으로 데이터 및 서비스를 제공

• 대부분 지역 서비스

<ARN(Amazone Resource Names>

ARN - AWS의 리소스에 부여되는 고유 아이디

<AWS 서비스 구조> (W2사진첨부)

- VPC는 원칙적으로 퍼블릭인터넷으로 접근 불가

- AWS 내부에 있는 서비스라도 원칙적으로 퍼블릭 인터넷을 통해서 접근

- 인터넷게이트웨이를 통해 외부와 접근

<VPC(Virtual Private Cloud)>

- 따라서 VPC의 목적은 외부와 격리된 네트워크를 만드는 것이 목적!

- AWS계정 전용 가상 네트워크

- 가상으로 존재하는 데이터 센터

- 원하는 대로 사설망 구축 가능

- 리전 단위

- 구조(AWS Cloud안에 Amazone VPC가 있음 / W2사진첨부)

<VPC 구성요소>

subnet - VPC를 특정 범위로 나눈 범위

Route Table - 네트워크 트래픽을 전달할 위치가 명시된 규칙 집합 테이블

Internet GW - VPC 리소스에서의 인터넷 통신을 활성화하기 위한 게이트웨이

Nat GW - 네트워크 주소 변환을 통해 private subnet에서 인터넷 통신을 연결하는 게이트웨이

VPC endpoint - Nat, IGW등을 통하지 않고 AWS의 서비스를 비공개로 연결가능하게 하는 서비스

<VPC아키텍쳐>(W2사진첨부)

<IAM(Identity and Access Management)>

- AWS 리소스에 대한 접근을 안전하게 제어하는 서비스

- 누가(AWS 사용자)가 어떤 AWS 서비스에 어떤 권한으로 접근할 수 있는지를 관리하는 역할

- IAM 요소

• 사용자(user) -> AWS 리소스에 접근하는 개별 계정

• 그룹(Group) -> 여러 사용자에게 동일한 권한을 부여할 수 있는 그룹

• 역할(Role) -> 특정 권한을 임시적으로 부여하는 역할(AWS 서비스 간 권한 위임 가능)

• 정책(Policy) -> JSON 형식으로 작성된 접근 권한 규칙

• MFA(Multi - Factor Authentication) -> 다중 인증으로 보안 강화

<IAM의 주요 기능>

- AWS 리소스 접근 제어

• IAM을 사용하여 AWS 서비스와 리소스에 대한 접근을 세밀하게 관리할 수 있음

• 특정 사용자나 그룹에게 EC2, S3, RDS 등 개별 서비스에 대한 권한을 부여 가능

- 정책(Policy) 기반 접근 관리

• JSON 형식의 Policy 문서를 사용하여 세부적인 접근 권한을 설정

- IAM 역할(Role) 및 서비스 연동

• EC2, Lambda, RDS 등의 AWS 서비스에 특정 권한을 부여 가능

• ex) EC2 인스턴스가 S3 버킷 데이터를 읽을 수 있도록 IAM 역할을 부여할 수 있음

- MFA(다중 인증) 적용 가능

• IAM 사용자 로그인 시 OTP 등 추가 인증 단계(MFA)를 설정하여 보안 강화

- AWS 루트 계정 보호

• AWS 계정을 생성하면 root user가 생성됨, 모든 권한을 가짐

• 보안 강화를 위해 루트 계정 대신 IAM 사용자를 생성하여 사용하고, 루트 계정의 MFA 활성화를 권장

<IAM 사용 시 주의할 점>

- 루트 계정(root user)은 최소한으로 사용

- 최소 권한 원칙(Least Privilege Principle)준수 -> 꼭 필요한 권한만 부여

- IAM 액세스 키(access key) 노출 금지 -> GitHub 등 코드 저장소에 올리지 않기

- IAM 정책을 잘못 설정하면 보안 사고 발생 가능 -> 사용자별 최소 권한 유지

[EC2(Elastic Computer Cloud)]

<EC2 인스턴스>

EC2 - 가상서버를 제공하는 AWS 서비스

인스턴스 - EC2에서 생성한 개별 가상 서버

<인스턴스 상태>

• running(실행 중)

• stopped(중지 됨)

• terminated(종료 됨)

• pending(대기 중)

• stopping(중지 중)

• rebooting(재부팅)

• shutting-down(종료 중)

<PEM, PPK>

PEM(.pem) - PEM(Privacy-Enhanced Mail)형식으로 저장된 키파일, Linux, macOS에서 기본적으로 지원되며, ssh 명령어로 쉽게 사용할 수 있음.

PPK(.ppk) - PPK(PuTTY Private Key)형식으로 저장된 키 파일, Windows에서 많이 사용하는 SSH 클라이언트인 PuTTY에서 사용됨.

[AWS CLI 기반 EC2 + S3 자동화 연동 실습]

<AWS CLI>

AWS CLI(Amazon Web Service Command Line Interface)

- AWS 리소스(S3, EC2, IAM 등)를 터미널이나 명령 프롬프트로 제어할 수 있는 도구

- 콘솔을 클릭하지 않고도 명령어만으로 S3 버킷 생성, 파일 업로드, EC2 실행 같은 작업가능

<AWS S3 CLI 주요 명령어> []: 필수 입력 / <>: 필요하면 넣으셈

aws s3 ls - 버킷 목록 확인

aws s3 ls s3 [버킷이름] - 버킷 안의 파일 목록

aws s3 cp [로컬경로] s3://[버킷이름]/경로 - 파일 업로드

aws s3 cp s3://[버킷이름]/<경로 로컬경로>/[넣을 파일명] - 파일 다운로드

aws s3 cp s3://[버킷이름]/<폴더 명> ./<로컬 폴더명> --recursive - 다운로드

aws s3 cp ./<로컬 폴더명> s3://[버킷이름]/<폴더명> --recursive - 업로드

[S3]

<S3>

- AWS에서 EC2 서비스와 더불어 가장 오래되고 기본이 되는 객체 스토리지 서비스

- Amazone S3는 확장성, 내구성, 보안성이 뛰어난 객체 스토리지 서비스

- 99.999999999%(11 9's)의 내구성 제공, 데이터 손실 최소화

- 데이터 저장 공간이 거의 무제한

- 전 세계 수백만 기업의 데이터를 저장하는 핵심 서비스

- 특징

• 객체 스토리지 서비스 - 파일 저장 전용(EBS / EFS 등 블록 스토리지와 구분) -> 어플리케이션 설치 불가

• 글로벌 서비스 단위 - 데이터는 특정 리전에 저장됨

• 무제한 용량 지원 - 단일 객체는 0byte ~ 5TB 저장 가능

• 최소 3개 이상의 물리적으로 분리된 가용 영역에 데이터 복제

• 높은 내구성과 고가용성을 제공

• OS 도움 없이 객체별 접근 가능, 데이터 저장 및 활용이 용이

<버킷과 객체>

- 버킷

• S3에서 저장공간을 구분하는 단위(디렉터리 개념)

• 전 세계 고유 이름을 가져야 함 -> 리전과 무관하게 중복 불가

• 한번 생성 시, 이름 변경 불가

- 객체(S3에 저장되는 기본 매체)

• Owner - 소유자 정보

• Key - 객체 이름(경로 역할)

• Value - 파일 데이터 자체

• Version ID - 파일 버전 식별자(버전 관리 활성화 시에만 생성됨)

• Metadata - 파일 관련 정보(기본 메타데이터 외에 사용자 정의도 가능함)

<S3의 내구성 & 가용성>

- 내구성 - 파일을 잃어버리지 않을 능력 / 99.999999999% -> 파일 손실 확률 극히 낮음

- 가용성 - 내가 원할 때 그 파일을 쓸 수 있는 능력

- 최소 3개 AZ에 중복 저장(Standard 기준)

- 가용성 SLA - 99.9% 수준

- SLA(Service Level Agreement) - 즉 서비스 수준 협약

• "이 서비스를 얼마나 안정적으로 사용할 수 있는가"

• 99.95% - 1년 중 약 52분 이하의 다운타임

• 99.95% - 1년 중 약 1일 18시간 이하 다운

<S3 스토리지 클래스>

- S3는 다양한 스토리지 클래스 제공

- 클래스별로 저장의 목적, 예산의 방법에 따라 다른 저장 방법 적용

<S3 보안>

- 기본적으로 모든 객체는 Private(비공개) 상태로 생성됨

- 퍼블릭 접근은 명시적으로 허용해야 함(웹 호스팅 등)

- Bucket Policy(버킷 단위 정책)

- IAM 정책

• Identity-based Policy(자격 증명 기반 정책)

         - IAM 사용자, 그룹, 역할에 부여하는 정책

         - "누가 무엇을 할 수 있는가"를 설정함

• Resource-based Policy(리소스 기반 정책)

         - 리소스(S3, SQS등)에 직접 부여하는 정책

         - "이 리소스에 누가 접근 가능한가"를 정의함

<S3 버킷 정책>

- 버킷 단위로 적용되는 리소스 기반 정책

- 언제, 어디서, 누가, 무엇을, 어떻게 할 수 있는지 정의

- 익명 사용자 또는 다른 계정에 대한 권한 부여도 가능

- ex) arn:aws:s3:::[버킷 이름]/[폴더명]/\*(폴더 안에 있는 모든 것)

         -> 해당 경로에 있는 모든 객체 대상으로 권한 설정

<S3 실습>

- 버킷을 활용한 호스팅 실습

- index.html을 생성

- 그 안에 사진도 업로드

- 버킷의 퍼블릭 엑세스 허용

<S3 버킷 정책>

- S3 리소스(버킷 및 그 안의 객체들)에 대한 접근 제어 정책

- JSON 포캣의 문서

- 버킷 단위로 붙이는 정책 -> 버킷 안의 모든 객체에 적용 가능

- IAM 정책과는 다르게 리소스 자체에 붙는 정책

index.html url이 오류가 걸림 -> 이유 - 버킷 정책이 없음!

Allow / Deny

GetObject / OutObject

[Auto Scaling]

<스케일링(Scaling)>

- 애플리케이션 또는 시스템의 성능을 높이기 위해 컴퓨팅 리소스를 확장하거나 축소하는 것

- 요청이 많아질 때 처리할 수 있도록 서버를 키우는 것

<스케일 업(Scale-Up, Vertical Scaling)>

- 서버 한 대의 성능을 높이는 방식 (성능이 오르면 가격도 오른다)

- 장점

• 단순하고 설정이 쉬움

- 단점

• 물리적인 한계 존재(업그레이드 한계가 있음), 100배 가능

• 재시작 필요성 있음

• 하나의 서버에만 의존 -> 장애 발생 시 위험

<스케일 아웃(Scale-Out, Horizontal Scaling)>

- 서버를 여러 대로 늘려서 처리 성능을 확장하는 방식

- 장점

• 무중단 확장 가능!

- 단점

• 복잡한 아키텍쳐, Load Balancer(ELB) 등의 구성 필요

<Auto Scaling>

- 트래픽이 언제 늘고 줄지, 사람이 항상 직접 조절해야 할까?

- Auto Scaling - 트래픽 상황에 맞춰 서버 수를 자동으로 늘리거나 줄여주는 AWS 서비스

• 트래픽 증가 시 자동으로 인스턴스 증가

• 사용량 감소시 자동으로 인스턴스 축소

• 리소스 비용 절감 + 고가용성 유지

<Auto Scaling 구성요소>

Launch Template

- 새 인스턴스를 생성할 때 필요한 설정을 담고 있는 설계도

- 어떤 AMI(이미지)로 만들지

- 인스턴스 타입(t3.micro 등)

- 키 페어, 보안 그룹

- UserData 스크립트(초기 셋팅 자동화)

Auto Scaling Group(ASG)

- 인스턴스를 묶어서 관리하는 단위

- 최소 / 최대 / 원하는(Desired) 인스턴스 수 설정

- 실제 인스턴스 수를 계속 모니터링하고 자동 조절

- Availability Zone 간 분산 기능

- 설정값

• 원하는 용량(Desired Capacity) - 현재 운영하고 싶은 인스턴스수

• 최소 용량(Min Capacity) - 줄어들 수 있는 하한선

• 최대 용량(Max Capacity) - 늘어날 수 있는 상한선

• 원하는 용량은 최소보다 작을 수 없고, 최대보다 클 수 없다

• 최소 =< 원하는 용량 =< 최대 용량

Scaling Policy(스케일링 정책)

- 언제, 어떻게 서버 수를 조절할지에 대한 규칙

- Target Tracking - 평균 CPU 60% 유지처럼 목표 설정

- Step Scaling - CPU 70~80% -> +1대, 80% 이상 -> +2대

- Scheduled Scaling - 특정 시간에 확장

CloudWatch 알람

- 지표를 감시하다가 스케일링 정책을 실행하는 역할

- CPU 사용률, 네트워크 트래픽, 메모리 등 모니터링

- 조건 충족 시 Scaling Policy 트리거

- 실시간 알람 + 로그 기록 가능

[ELB]

ELB(Elastic Load Balancer - AWS에서 제공하는 트래픽 분산 서비스

- ELB는 들어오는 어플리케이션 트래픽을 Amazon 인스턴스, 컨데이너, IP 주소, 람다 함수와 같은 여러 대상에 자동 분산 시킴.

- ELB는 단일 가용영역, 여러 가용 영역에서 다양한 어플리케이션 부하를 처리 가능.

- ELB에서 제공하는 3가지 로드밸런서는 모두 어플리케이션의 내결합성에 필요한 고가용성, 자동 확장 / 축소, 강력한 보안을 갖추고 있음.

- 사용자가 EC2에 접근하려면 모든 EC2 IP 주소를 알아야한다.

<ELB 특징>

- 다수의 서비스에 트래픽을 분산 시켜주는 서비스

- Health Check - 직접 트래픽을 발생시켜 Instance가 살아있는지 체크

- Autoscaling과 연동 가능

- 여러 가용영역에 분산 가능

- 지속적으로 IP 주소가 바뀌며 IP고정 불가능 - 항상 도메인 기반으로 사용

<ELB 종류>

- Application Load Balancer - 똑똑함

• 트래픽을 모니터링하여 라우팅 가능

• image.test.com -> 이미지 서버로, web.test.com -> 웹 서버로 트래픽 분산

- Network Load Balancer - 빠름

• TCP 기반 빠른 트래픽 분산

• Elastic IP 할당 가능

- Classic Load Balancer - 옛날꺼

• 현재는 잘 사용하지 않음

<ALB - Target Group>

- 타겟 그룹 - ALB가 트래픽을 분산시킬 대상(Target)을 논리적으로 묶어놓은 그룹

- 타겟 그룹 기준으로 요청 분배

- EC2 인스턴스, IP 주소, Lambda 함수, ELB 타겟 그룹 가능

- 역할

• ALB -> 라우팅 규칙에 따라 타겟 그룹으로 요청 전달

• 타겟 그룹은 내부에서 등록된 인스턴스 중 헬스체크 를 통과한 대상에게 트래픽 전달

• 타겟 그룹별로 포트, 프로토콜, 헬스체크 설정 가능