Computing Service(3)

서버리스 활용법 이해

1. AWS Lambda는 다른 서비스와 연계해 사용
    - Lambda 함수는 입력받은 데이터 처리, 다른 AWS 서비스와 연계해 사용자 접속/데이터 연동 처리 자동 수행
    - 사용자 요청에 따라 Lambda 함수 실행
        - **Amazon API Gateway(API Gateway)** 서비스와 Lambda를 결합해 **사용자의 HTTP 요청을 Lambda 함수로 처리 가능** → 이를 이용해 간단한 웹 응용 프로그래밍 제작 가능
        - 사용자로부터 HTTP 요청을 받으면 GET과 같은 HTTP 메서드나 요청 경로 정보가 포함된 JSON 데이터를 Lambda 함수에 전달
        - Lambda 함수에서 응답 내용을 반환하도록 코드 작성 시 API Gateway를 통해 응답 반환
        - 서버를 구축하지 않고도 웹 응용 프로그램을 만들 수 있음 (서버리스 서비스)
    - S3에 데이터를 저장할 때 자동 처리
        - **Amazon S3 버킷**에 데이터 저장 시 데이터를 Lambda로 자동 처리
        - 이 경우에도 S3 버킷이나 데이터 정보가 **JSON 형식으로 Lambda 함수에 전달**되므로 Lambda 함수에서는 이 데이터를 처리하도록 구현
    - 주기적으로 Lambda 함수 실행
        - **Amazon EventBridge**라는 서비스와 Lambda를 연계해 주기적으로 정해진 시간에 어떤 처리를 구현할 수 있음

2. AWS Lambda 추가 설정 및 모니터링
    - 사용자가 변경 가능한 Lambda의 실행 환경 3가지

    - **메모리 용량**
        - 함수 실행 시 사용 가능한 메모리 용량 지정
        - CPU 값은 지정 불가, 지정된 메모리 용량에 비례해 설정됨
    - **타임아웃 시간**
        - 함수가 실행되는 최대 시간
        - 기본값은 3초, 1~900초 사이로 지정 가능
    - **환경 변수**
        - 함수에 사용하는 외부 변수와 같은 개념

- 처음 Lambda 함수 생성 시 다른 서비스에 대한 조작 권한 따로 부여 필요
- Lambda 함수에 **IAM(Identity and Access Management)** 역할 형식으로 권한을 부여
- IAM 역할 지정 안하면 Lambda 기본 IAM 역할이 자동으로 생성, 설정

  → Amazon CloudWatch라는 감시 서비스로 로그를 보낼 수 있는 권한만 부여
- Lambda에 권한을 설정할 때는 **처리 내용에 필요한 최소한의 권한 설정이 중요**

- Lambda 함수 실행 상태 모니터링
    - Lambda 함수를 생성/배포하면 CloudWatch에 다음과 같은 정보가 자동 전송
        - **실행 횟수**
        - **실행 시간(최소/최대/평균)**
        - **오류 수와 실행 성공률**
    - 함수가 출력하는 로그도 자동 전송

3. AWS Lambda 이용요금
    - 요청 수와 처리 시간에 따라 청구, 시간에 대한 과금X
    - GB-초라는 단위: 메모리 1GB의 함수가 1초간 실행된 경우의 요금

컨테이너의 구조와 특징 이해

1. 컨테이너란
    - 1개의 물리 서버에 여러 컨테이너가 동작
    - 서버의 OS와 물리 자원은 각 컨테이너가 공동으로 이용
    - 각 컨테이너는 네트워크가 분리되어 있지만 외부 네트워크와의 통신이나 컨테이너 간 통신을 할 때 컨테이너 런타임을 통해 이루어짐
    - **컨테이너 이미지**
        - 어떤 응용 프로그램을 실행할 지 미리 정의해둔 파일을 바탕으로 실행
        - 컨테이너 실행에 필요한 파일은 이미지에 포함
        - 이미지를 기반으로 여러 컨테이너 실행 가능

2. 가볍고 빠른 컨테이너
    - **가상화에 비해 가볍고 빠름** → 컨테이너 내에 포함된 것이 적기 때문
    - 가상 서버 기동 → OS, 미들웨어, 응용 프로그램 등 필요

      컨테이너 → 응용 프로그램 프로세스만 시작

      컨테이너 이미지 → 응용 프로그램 실행을 위한 의존성 패키지만 포함
    - 가상 머신과 비교해 빠르게 시작
        - 가상 서버 → 운영체제 시작 후 미들웨어, 응용 프로그램 실행
        - **컨테이너 → 직접 응용 프로그램 실행**
    - 처리 속도 역시 OS에 대한 오버헤드가 없는 만큼 가상 서버보다 빠름

3. 배포의 용이성
    - 컨테이너 이미지
        - 한번 생성하면 다른 서버에서 바로 사용 가능

    - 서로 다른 환경에서도 내용은 바뀌지 않으므로 개발할 때 설정한 내용, 사용한 라이브러리 그대로 사용 가능 → **개발이 끝난 이미지는 어느 환경에서나 동일한 동작 보장**
    - 바로 압축 파일 형태로 내보내기/가져오기 수행 가능
    - 이미지 레포지토리를 이용해 업로드/다운로드 가능
- **주의: 컨테이너는 '완성된' 이미지 이므로 종료 시 저장된 내용 사라짐**

  따라서 **실행될 호스트 서버의 디렉터리를 마운트해서 작업 내용 저장 등의 처리 필요**

4. 컨테이너 오케스트레이션
    - 컨테이너 오케스트레이션 도구 이용 시 관리를 자동화해 운영 부하 감소

ECS로 누리는 컨테이너의 장점

1. AWS의 컨테이너 오케스트레이션 서비스
    - AWS에서 컨테이너 시스템 구축 시 일반적으로 **ECS**와 **EKS**라는 두 가지 서비스 중 선택 가능

      가장 큰 차이점: 컨테이너 오케스트레이션 기능의 담당이 AWS? 쿠버네티스?
    - ECS
        - AWS의 관리형 컨테이너 오케스트레이션 서비스
        - **오케스트레이션 도구의 구축/관리를 모두 AWS에 맡길 수 있음**
        - ECR이라는 컨테이너 이미지 레지스트리 서비스도 제공

          → 이용 시 사용자 정의된 컨테이너 이미지를 AWS에서 관리 가능, ECS 배포 가능

2. EC2와 Fargate
    - EC2
        - **EC2 인스턴스 내에서 실행되는 컨테이너 런타임에서 컨테이너를 실행하는 타입**

    - 사용자가 컨테이너 런타임이 설치된 서버를 관리해야 함
    - CPU/메모리 확장을 원하면 인스턴스 타입을 적절한 것으로 변경,
      스토리지를 늘리고 싶다면 EBS의 볼륨 크기 확장
- Fargate
    - **AWS에서 관리하는 서버에서 컨테이너 실행**
    - 컨테이너 런타임, OS 버전업 같은 서버 관리를 AWS에서 하므로
      사용자는 컨테이너 이미지만 관리
    - 컨테이너 런타임, OS 같은 버전은 플랫폼 버전(PV)로 관리되므로 실행 시 지정 필요
    - PV는 1년 단위로 갱신, 사용자는 버전 업데이트에 따른 대응
    - CPU/메모리 직접 할당 가능, 지정한 CPU 값에 따라 할당할 수 있는 메모리 크기가 정해져 있음
    - 스토리지는 사전에 Fargate 태스크 스토리지로 태스크 실행 시 할당되며, 할당량은 PV에 따라 다름

3. Fargate 장점
    - Fargate의 주요 특징: **AWS에서 서버 관리 → 유지보수 작업을 AWS에서 담당**

4. 쿠버네티스를 보다 쉽게 사용할 수 있는 서비스
    - 쿠버네티스(Kubernetes)
        - 구글에서 개발한 컨테이너 오케스트레이션 도구, 현재는 오픈소스 소프트웨어
        - **k8s** 로 표기하기도 함
    - k8s는 구축과 운영에 시간과 비용이 든다는 문제 → EKS 사용 시 문제 해결 가능

서버 지식 없이 사용 가능한 대표 서비스 5개

1. AWS Lightsail

- 일반적으로 자주 사용되는 구성의 가상 서버를 쉽고 빠르게 구축하기 위한 서비스
- 아이콘, 설명으로 구축 화면 표시 → 간단히 가상 서버 생성 가능
- 생성한 인스턴스에는 웹 브라우저를 통해 콘솔 접속 가능
- 여러 인스턴스 생성 시 자동으로 로드 밸런싱, 통신 암호화 대응 등 기본 기능
- 다른 AWS 서비스와 연동 가능
- 미들웨어 버전 고려 설치, 유연한 부하 분산 설정 등의 자유도X

2. AWS Elastic Beanstalk
    - 주요 언어나 환경에서 개발된 응용 프로그램 배포를 위한 실행 환경 자동 생성 서비스
    - 플랫폼을 지정해 코드 업로드 시 AWS 자원 자동 구축, 응용 프로그램 배포
    - 사용자가 스스로 구축할 수 있는 시스템을 서버 지식 없이도 구축 가능하다는 장점
    - 원하는 배포 방식으로 쉽게 배포 가능하다는 특징

3. AWS App Runner
    - 컨테이너화된 응용 프로그램을 손쉽게 배포하는 서비스
    - 컨테이너 이미지 지정, 서비스 포트, Auto Scaling 같은 설정만으로 Fargate에 배포 가능

4. AWS Batch
    - 지정된 프로그램을 EC2나 Fargate에서 실행하는 서비스
    - 실행 프로그램의 순서 관리나 다수 프로그램의 병렬 구동 같은 복잡한 제어에 특화된 서비스

5. AWS Outposts
    - 사용자가 온프레미스 환경에서 AWS와 같은 서비스를 실행할 수 있도록 물리 서버를 대여하는 서비스 → AWS를 프라이빗 클라우드화해서 제공하는 서비스

- 데이터 센터에 설치할 수 있는 서버 기기세트 제공

  → AWS Outposts를 AWS의 **새로운 가용영역**으로 사용 가능