Docker 소개

- Docker: 도커는 앱 배포를 위한 소프트웨어 개발 플랫폼, 컨테이너 기술
- 컨테이너에 앱이 패키징되는데 컨테이너는 표준화되어있어서 아무 운영체제에나 실행 가능 -> 호환성 문제 없음
- 반복 작업이 줄어들고, 유지 및 배포가 쉬우며 언어, 운영체제, 기술에 상관없이 실행 가능
- 사용 사례 `마이크로서비스 아키텍처`
- 작동 방식
  - 서버에서 도커 에이전트를 실행하면 도커 컨테이너를 시작할 수 있음
  - 도커 이미지는 **도커 리포지토리에** 저장됨
    - Docker Hub: 퍼블릭 리포지토리, 기본 이미지를 찾을 수 있음
    - Amazon ECR(Amazon Elastic Container Registry): 프라이빗 리포지토리지 이지만 Amazon ECR Public Gallery라 불리는 퍼블릭 리포지토리 옵션도 있음
- 도커와 가상 머신의 차이점
  - 도커는 가상화 기술이긴 하지만 **리소스가 호스트와 공유되어 한 서버에서 다수의 컨테이너를 공유** 가능
  - 가상 머신의 아키텍처를 살펴보면 인프라와 호스트 운영체제가 있으며 그 위에 하이퍼바이저가 있고 앱과 Guest 운영 체제가 있음 -> EC2 원리
  - EC2 머신은 하이퍼바이저에 실행되는 가상 머신
  - 가상 머신의 EC2 인스턴스는 각자 분리되어 있어 리소스를 공유하지 않음
  - 반면 도커 컨테이너의 경우 인프라와 EC2 인스턴스 같은 호스트 OS가 있고, 도커 Daemon 위에 많은 컨테이너가 있음 -> 공존 가능
  - 가상 머신보다 덜 안전하지만 하나의 서버에 많은 컨테이너를 실행할 수 있음
- Dockerfile: 도커 컨테이너를 구성하는 파일
- 베이스 도커 이미지에 몇 가지 파일을 추가해서 구축하면 도커 이미지
- 도커 이미지는 푸시(push)를 해서 도커 리포지토리에 저장 가능
- 퍼블릭 리포지토리인 Docker Hub에 푸시하거나 Amazon 버전의 도커 리포지토리인 Amazon ECR에 푸시
- 나중에 도커 리포지토리에서 이미지를 가져와서 실행
- 도커 이미지를 실행하면 도커 컨테이너가 되고 도커를 구축할 때 사용했던 코드를 실행

AWS 도커

- Amazon ECS 즉 Elastic Container Service 에서 관리
- Amazon EKS 즉 Elastic Kubernetes Service는 Kubernetes의 관리형 버전으로 오픈 소스 프로젝트
- AWS Fargate는 Amazon의 서버리스 컨테이너 플랫폼으로 ECS와 EKS 둘 다 함께 작동
- Amazon ECR은 전에 보여드렸듯이 컨테이너 이미지를 저장하는 데 사용

AWS ECS

- EC2 시작 유형(Launch Type)

    - ECS는 Elastic Container Service의 약어로 AWS에서 컨테이너를 실행하면 ECS 클러스터에 이른바 ECS 태스크를 실행
    - ECS 클러스터에는 들어있는 게 있는데 EC2 시작 유형을 사용하면 EC2 인스턴스가 있음
    - EC2 시작 유형으로 EC2 클러스터를 사용할 때는 인프라를 **직접 프로비저닝**하고 유지
    - Amazon ECS 및 ECS 클러스터가 여러 EC2 인스턴스로 구성됨
    - ECS 인스턴스는 특별하게 각각 ECS 에이전트(Agent)를 실행
    - ECS 에이전트가 각각의 EC2 인스턴스를 Amazon ECS 서비스와 지정된 ECS 클러스터에 등록
    - 이후에 ECS 태스크를 수행하기 시작하면 AWS가 컨테이너를 시작하거나 멈춤
    - 새 도커 컨테이너가 생기면 도식에서 볼 수 있듯 시간에 따라 EC2 인스턴스에 지정
    - ECS 태스크를 시작하거나 멈추면 자동으로 위치가 지정
    - 도커 컨테이너는 미리 프로비저닝한 Amazon EC2 인스턴스에 위치
- Fargate 시작 유형
    - AWS에 도커 컨테이너를 실행하는데 인프라를 프로비저닝하지 않아 관리할 EC2 인스턴스가 없음 -> 서버리스
    - 서버를 관리하지 않아 서버리스라 부르는데 서버가 없는 건 아님
    - Fargate 유형은 ECS 클러스터가 있을 때 ECS 태스크를 정의하는 태스크 정의만 생성하면 필요한 CPU나 RAM에 따라 ECS 태스크를 AWS가 대신 실행
    - 새 도커 컨테이너를 실행하면 어디서 실행되는지 알리지 않고 그냥 실행
    - 백엔드에 EC2 인스턴스가 생성될 필요도 없음
    - 간단하게 태스크 수만 늘리면 됨

ECS 태스크의 IAM

- EC2 시작 유형을 사용한다면 EC2 인스턴스 프로파일을 생성
- ECS 에이전트만이 EC2 인스턴스 프로파일을 사용하며 그 EC2 인스턴스 프로파일을 이용해 API 호출
- 인스턴스가 저장된 ECS 서비스가 CloudWatch 로그에 API 호출을 해서 컨테이너 로그를 보내고 ECR로부터 도커 이미지를 가져옴
- Secrets Manager나 SSM Parameter Store에서 민감 데이터를 참고하기도 함
- ECS 태스크는 ECS 태스크 역할을 가짐
- EC2와 Fargate 시작 유형에 모두 해당되며 두 개의 태스크가 있다면 각자에 특정 역할을 만들 수 있음
- 태스크 A는 EC2 태스크 A 역할을 맡고 태스크 B는 EC2 태스크 B 역할
- 역할을 다르게 하는 이유?
    - 역할이 각자 다른 ECS 서비스에 연결할 수 있게 하기 때문
    - EC2 태스크 A 역할은 태스크 A가 Amazon S3에 API 호출을 실행할 수 있도록 한다면 태스크 B 역할은 DynamoDB에 API 호출을 실행할 수 있도록
    - ECS 서비스의 태스크 정의에서 태스크의 역할을 정의
- 로드 밸런서 통합
    - 여러 ECS 태스크들이 실행되며 ECS 클러스터 안에 있음
    - HTTP나 HTTPS 엔드 포인트로 태스크를 활용하기 위해 애플리케이션 로드 밸런서(ALB)를 앞에서 실행하면 모든 사용자가 ALB 및 백엔드의 ECS 태스크에 직접 연결

    - ALB는 이 경우를 포함해, 대부분의 사용 사례를 지원하는 좋은 옵션
    - 네트워크 로드 밸런서(NBL)는 처리량이 매우 많거나 높은 성능이 요구될 때만 권장
    - AWS Private Link와 사용할 때도 권장되는 옵션
    - 구세대 Elastic Load Balancer(ELB)는 권장하지는 않음 ->ELB는 Fargate에 연결할 수 없음
    - 애플리케이션 로드 밸런서(ALB)는 Fargate와도 사용할 수 있음
- Amazon ECS의 데이터 지속성
    - 데이터 볼륨이 필요하며 여러 종류가 있는데 그 중 하나가 EFS에서 자주 사용됨
    - ECS 클러스터에 EC2 인스턴스와 Fargate 시작 유형 둘 다 구현했다고 가정
    - EC2 태스크에 파일 시스템을 마운트해서 데이터를 공유하려고 할 때 Amazon EFS 파일 시스템을 사용하는 게 좋음
    - 네트워크 파일 시스템이라 EC2와 Fargate 시작 유형 모두 호환이 되며 EC2 태스크에 파일 시스템을 직접 마운트 가능
    - 어느 AZ에 실행되는 태스크든 Amazon EFS에 연결되어 있다면 데이터를 공유할 수 있고 원한다면 파일 시스템을 통해 다른 태스크와 연결할 수 있기 때문
    - Fargate로 서버리스 방식으로 ECS 태스크를 실행하고, 파일 시스템 지속성을 위해서는 Amazon EFS를 사용하는 게 가장 좋음
    - EFS 역시 서버리스이기 때문에 서버를 관리할 필요 없고 미리 비용을 지불
    - 미리 프로비저닝하기만 하면 바로 사용할 수 있음
    - 예시
        - EFS와 ECS를 함께 사용해서 다중 AZ가 공유하는 컨테이너의 영구 스토리지가 있음
- Amazon S3는 ECS 태스크에 파일 시스템으로 마운트될 수 없음

AWS ECS - 오토 스케일링

- 태스크 수를 자동으로 늘리거나 줄일 수도 있음 -> AWS의 Auto Scaling
- 세 개의 지표에 대해 확장이 가능
- ECS 서비스의 CPU 사용률을 확장할 수 있고, 메모리 사용률 즉 ECS 서비스의 RAM이나ALB 관련 지표인 타겟당 요청 수를 확장할 수 있음
- 특정 타겟을 추적하는 대상 추적(Target Tracking) 스케일링이나 단계(Step) 스케일링, 미리 ECS 서비스 확장을 설정하는 예약(Scheduled) 스케일링 등
- EC2 시작 유형이라면 **태스크 레벨에서의 ECS 서비스 확장이 EC2 인스턴스 클러스터의 확장과 다름**
- EC2 오토 스케일링이 필요하지 않다면, 백엔드에 EC2 인스턴스가 없다면 Fargate를 사용하는 것이 서비스 오토 스케일링에 도움 -> 서버리스라서
- EC2 시작 유형에서는 백엔드의 EC2 인스턴스를 자동으로 확장하는 법
    - 스케일링에서 여러 방법이 있는데 그 중 하나가 오토 스케일링 그룹
    - CPU 사용률에 따라 ASG를 확장한다고 하면 CPU 사용률이 급등할 때 EC2 인스턴스를 추가
    - ECS 클러스터 용량 공급자를 사용할 수 있음
    - 용량 공급자(Capacity Provider)는 새 태스크를 실행할 용량이 부족하면 자동으로 ASG를 확장
    - Capacity Provider는 오토 스케일링 그룹과 함께 사용되며 RAM이나 CPU가 모자랄 때 EC2 인스턴스를 추가

- 오토 스케일링 그룹과 ECS 클러스터 용량 공급자 중에 EC2 시작 유형의 경우 EC2 클러스터 용량 공급자를 사용
- 예시
  - 서비스 A에 태스크 두 개가 실행 중이고 CPU 사용량을 적절히 선택
  - 서비스는 AWS 애플리케이션 오토 스케일링으로 자동 확장
  - 만약 사용자가 훨씬 많아져서 CPU 사용량이 급증하면 ECS 서비스 레벨에서 CPU 사용량을 모니터링하는 CloudWatch 지표가 CloudWatch 알람을 트리거
  - 이 알람이 오토 스케일링을 또 트리거하면 ECS 서비스의 희망 용량이 증가하고 새 태스크가 생성
  - EC2 시작 유형에서 실행되는 서비스라면 ECS 용량 공급자가 EC2 인스턴스를 추가해 ECS 클러스터를 확장

AWS ECS 솔루션 아키텍트

- EventBridge에 의해 호출된 ECS 태스크
  - 예를 들어 Amazon ECS 클러스터가 있고 그걸 AWS Fargate가 지원, S3 버킷이 있음
  - 사용자들은 객체들을 우리 S3 버킷에 업로드, 이 S3 버킷은 예를 들어Amazon EventBridge와 통합되어서 모든 이벤트를 전송할 수 있음
  - Amazon EventBridge는 항상 ECS 태스크를 실행하기 위한 규칙을 갖고 있을 수 있음
  - ECS 태스크가 생성 후 관련된 ECS 태스크 역할이 있는데 태스크 자체로부터 직접 객체를 받고 그걸 처리하고 그 결과를 Amazon DynamoDB로 전송함 -> ECS 태스크 역할을 갖고 있기 때문
  - Docker Container, Amazon EventBridge, Fargate 모드의 ECS, 그리고 Amazon S3 및 Amazon DynamoDB와 대화하기 위한 ECS 태스크 역할을 사용해서 S3 버킷에 업로드 된 객체를 처리할 수 있는 서버리스 아키텍처를 만든 것
  - EventBridge를 사용한 또 다른 아키텍처에는 EventBridge Schedule을 이용한 아키텍처가 있음
  - Fargate와 Amazon EventBridge가 지원하는 Amazon ECS 클러스터가 있음
  - 1시간마다 트리거되는 규칙을 스케줄링하고, 이 규칙은 Fargate에서 ECS 태스크를 실행
  - 1시간마다 우리의 Fargate 클러스터 안에서 새로운 태스크가 생성된다는 의미
  - 예시
    - Amazon S3에 액세스할 수 있는 ECS 태스크 역할을 만들고, Docker Container, 프로그램은 Amazon S3에서 어떤 파일들에 대한 배치 프로세싱을 1시간마다 수행
    - SQS Queue를 사용하는 예는 2개의 ECS 태스크가 있는 서비스가 있을 때 메시지가 SQS Queue로 전송되고, 서비스 자체가 SQS Queue로부터 메시지를 가져와서 그것들을 처리
    - SQS Queue에 더 많은 메시지가 있으면 오토 스케일링 덕분에 ECS 서비스에 더 많은 태스크를 갖게 되고, EventBridge를 사용하여 ECS 클러스터 안에서 이벤트를 가로채는 형태의 통합 있음
    - 태스크가 나가는 것에 대응하려 할 때는 ECS 클러스터에서 나가거나 시작되는 모든 태스크는 EventBridge에서 이벤트로서 트리거될 수 있음
    - STOPPED나 stoppedReason에 대해서 ECS 태스크 상태가 변함
    - SNS 토픽으로 경보하거나 관리자에게 이메일을 전송
    - EventBridge는 최소한 여러분의 ECS 클러스터에 있는 컨테이너들의 라이프사이클을 관리

AWS ECR

- Amazon ECR은 Elastic Container Registry
- AWS에 도커 이미지를 저장하고 관리하는 데 사용
- Docker Hub 등의 온라인 저장소를 활용했는데 이미지를 Amazon ECR에도 저장할 수 있음
- 옵션
  - 계정에 한해 이미지를 비공개로 저장
    - 여러 계정으로 설정 가능
  - 퍼블릭 저장소를 사용해 Amazon ECR Public Gallery에 게시하는 방법
- ECR은 Amazon ECS와 완전히 통합되어 있고, 이미지는 백그라운드에서 Amazon S3에 저장
- ECR 저장소에 여러 도커 이미지가 있는데 ECS 클러스터의 EC2 인스턴스에 이미지를 끌어오기 위해서는 EC2 인스턴스에 IAM 역할을 지정
- IAM 역할이 도커 이미지를 인스턴스에 끌어오면 ECR에 대한 모든 접근은 IAM이 보호
- ECR에 권한 에러가 생긴다면 정책을 보기
- EC2 인스턴스에 이미지를 끌어온 후에는 컨테이너가 시작
- ECS와 ECR이 이런 식으로 함께 작동
- Amazon ECR은 단순히 저장하는 리포지토리에 그치지 않고, 이미지의 취약점 스캐닝, 버저닝 태그 및 수명 주기 확인을 지원
- **도커 이미지를 저장할 때는 ECR**

EKS 개요

- Amazon EKS는 Amazon Elastic Kubernetes Service
- AWS에 관리형 Kubernetes 클러스터를 실행할 수 있는 서비스
- Kubernetes는 오픈 소스 시스템으로 Docker로 컨테이너화한 애플리케이션의 자동 배포, 확장, 관리를 지원
- 컨테이너를 실행한다는 목적은 ECS와 비슷하지만 사용하는 API가 다름
- ECS는 오픈 소스가 아닌 반면 Kubernetes는 오픈 소스 -> 표준화
- EKS에는 두 가지 실행 모드
  - EC2 시작 모드는 EC2 인스턴스에서처럼 작업자 모드를 배포할 때 사용
  - Fargate 모드는 EKS 클러스터에 서버리스 컨테이너를 배포할 때 사용
- EKS의 사용 사례
  - 회사가 온프레미스나 클라우드에서 Kubernetes나 Kubernetes API를 사용 중일 때 Kubernetes 클러스터를 관리하기 위해 Amazon EKS를 사용
  - Kubernetes는 클라우드 애그노스틱으로 Azure, Google Cloud 등 모든 클라우드에서 지원
  - 클라우드 또는 컨테이너 간 마이그레이션을 실행하는 경우 Amazon EKS가 간단한 솔루션이 될 수 있음
  - VPC와 세 AZ가 있고 각 AZ는 퍼블릭 및 프라이빗 서브넷으로 나뉨
  - EKS 작업자 노드를 생성하면 EC2 인스턴스가 구성
  - 각 노드는 EKS 포드를 실행
  - ECS 태스크와 유사하지만 이름에서 포드를 발견하면 Amazon Kubernetes와 관련된 것
  - EKS 포드가 실행되는 EKS 노드는 오토 스케일링 그룹으로 관리
  - ECS와 유사하게 EKS 서비스나 Kubernetes 서비스를 노출할 때는 프라이빗 로드 밸런서나 퍼블릭 로드 밸런서를 설정해 웹에 연결

- Amazon EKS의 여러 노드 유형
  - 먼저 관리형 노드 그룹은 AWS로 노드, 즉 EC2 인스턴스를 생성하고 관리
  - 노드는 EKS 서비스로 관리되는 오토 스케일링 그룹의 일부
  - 온디맨드 인스턴스와 스팟 인스턴스를 지원
  - 자체 관리형 노드를 선택할 경우 -> 사용자 지정 사항이 많고 제어 대상이 많은 경우
  - 직접 노드를 생성하고, EKS 클러스터에 등록한 다음 ASG의 일부로 관리
  - 사전 빌드된 AMI인 Amazon EKS 최적화 AMI를 사용하면 시간을 절약
  - 온디맨드 인스턴스와 스팟 인스턴스를 지원
  - 노드를 원치 않는 경우에는 Amazon EKS가 지원하는 Fargate 모드를 선택
  - 유지 관리가 필요 없고 노드를 관리하지 않아도 되며 Amazon EKS에서 컨테이너만 실행하면 됨
  - Amazon EKS 클러스터에 데이터 볼륨을 연결하려면 EKS 클러스터에 스토리지 클래스 매니페스트를 지정
  - **컨테이너 스토리지 인터페이스(CSI)라는 규격 드라이버를 활용**
  - Amazon EBS와 Fargate 모드가 작동하는 유일한 스토리지 클래스 유형인 Amazon EFS를 지원하고, Amazon FSx for Lustre와 Amazon FSx for NetApp ONTAP을 지원

AWS App Runner - 개요

- 완전 관리형 서비스로 규모에 따라 웹 애플리케이션, API 배포를 관리
- 서비스로는 누구나 AWS에 배포를 할 수 있음
- 인프라나 컨테이너, 소스 코드 등을 알 필요가 없음
- 소스 코드나 Docker 컨테이너 이미지를 가지고 원하는 구성을 설정
- vCPU의 수나 컨테이너 메모리의 크기 오토 스케일링 여부 상태 확인을 설정하면 됨
- 웹 애플리케이션이나 API에 들어갈 기본 설정을 설정
- 자동으로 이뤄짐
- App Runner 서비스가 웹 앱을 빌드하고 배포
- 컨테이너가 생성되고 배포됨
- API나 웹 앱이 배포된 다음엔 URL을 통해 바로 액세스할 수 있음
- 배후에서 어떤 작업이 이뤄지는지 전혀 몰라도 배포할 수 있음
- 배후에서 AWS 서비스가 사용되는 것이겠지만 사용자는 몰라도 빠른 배포를 할 수 있음
- App Runner에는 장점이 많음
- 오토 스케일링이 가능하고 가용성이 높으며 로드 밸런싱 및 암호화 기능을 지원
- 애플리케이션, 즉 컨테이너가 VPC에 액세스할 수도 있어서 데이터베이스와 캐시 메시지 대기열 서비스에 연결할 수 있음
- 사용 사례
  - 빨리 배포해야 하는 웹 앱, API 그리고 마이크로서비스
  - 신속한 프로덕션 배포가 필요할 때