AWS Container Service

Jae Hyeon Kim

목차

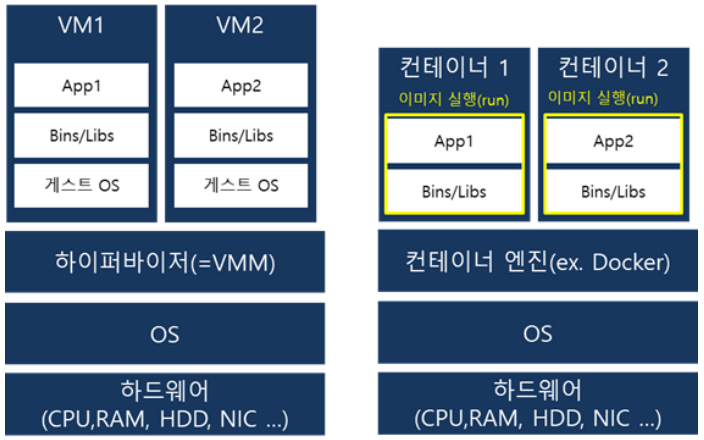
Container

컨테이너(Container)

- 컨테이너(container)는 개별 소프트웨어의 실행에 필요한 실행환경을 독립적으로 운용할 수 있도록 기반환경 또는 다른 실행환경과의 간섭을 막고 실행의 독립성을 확보해주는 운영체제 수준의 격리 기술

컨테이너와 가상머신(VM)의 차이

- 가상머신은 HyperVisor가 운영체제와 가상머신의 리소스를 분리해 VM의 생성과 관리를 지원한다. HyperVisor로 사용되는 물리 하드웨어를 Host라고 하고 리소스를 사용하는 여러 VM을 Guest라고 한다.
- 컨테이너의 경우 하나의 Host OS위에서 마치 각각의 독립적인 프로그램처럼 관리되고 실행된다. 불필요한 OS만드는 작업 및 Infra를 독립적으로 나눌 필요가 없어서 확장성이 좋고 빠르다.



컨테이너 장점

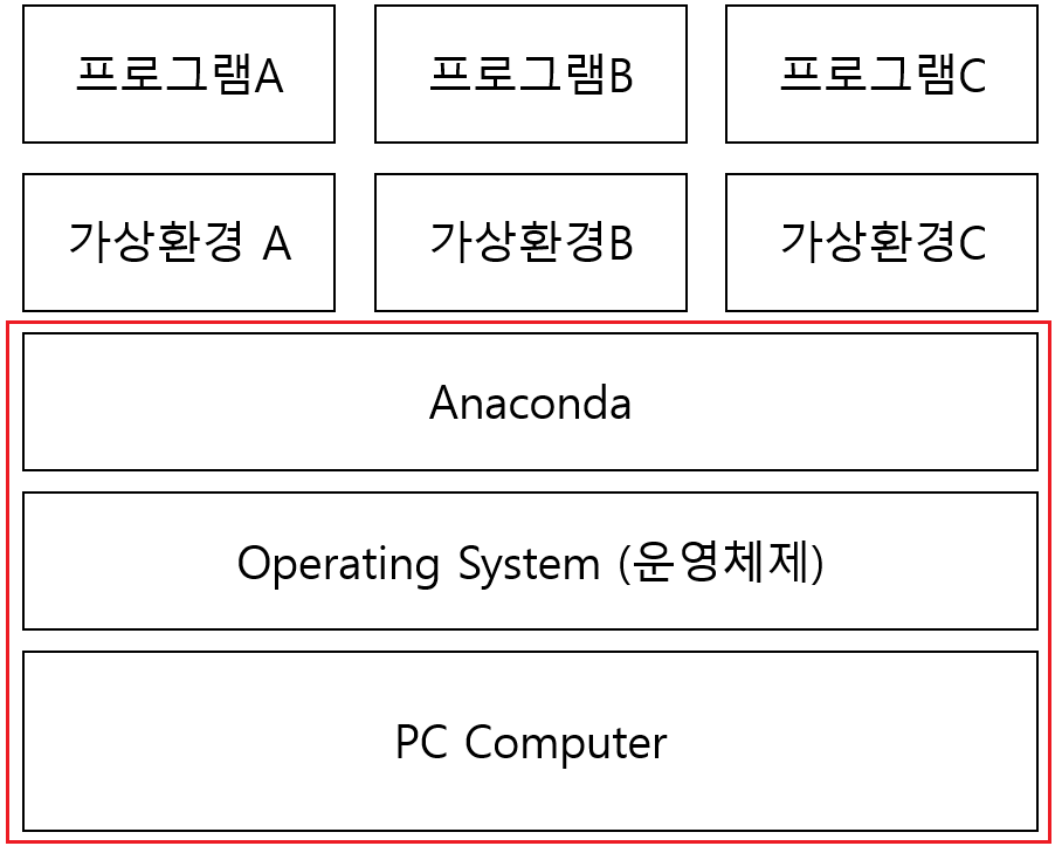
- 애플리케이션 레벨의 고립
- VM보다 빠른 셋업
- VM보다 메모리 덜 소모
- 마이그레이션, 백업, 전송이 쉬움
- 애플리케이션 배치와 유지보수 향상
- 애플리케이션 전달 시간 감소

컨테이너 단점

- Host OS에 종속적
- 컨테이너별 커널 구성이 불가능

가상환경(VE)과 컨테이너

- Anaconda로 구축한 가상환경은 PC Computer에 종속된다.



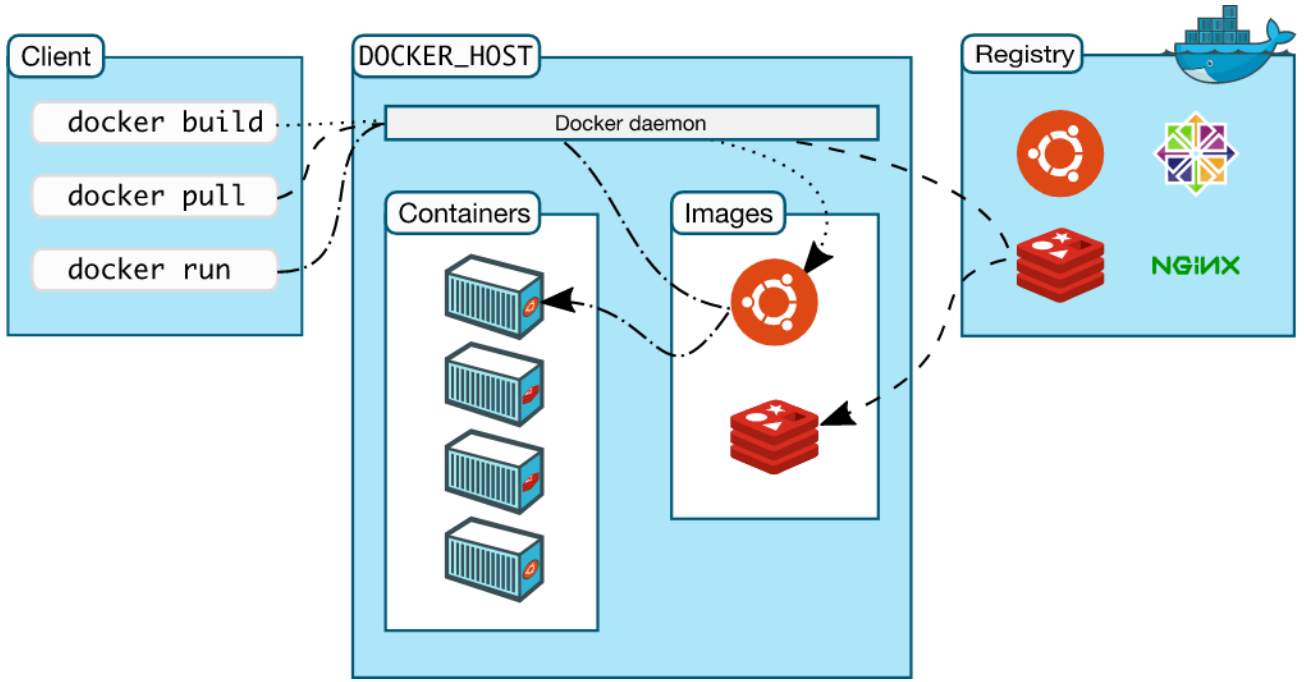
Docker

도커(Docker)

- 도커는 컨테이너 기반의 오픈소스 가상화 플랫폼중 하나.
- 도커를 사용하면 인프라에서 애플리케이션을 분리하여 컨테이너로 추상화시켜 소프트웨어를 빠르게 제공할 수 있다.
- 도커는 컨테이너의 라이프사이클을 관리하고 어플리케이션을 오케스트레이션된 서비스로 배포할 수 있다.

도커 아키텍처

- 도커는 클라이언트-서버 아키텍처.
- 도커 클라이언트와 도커 데몬이 RestAPI를 사용하여 통신.



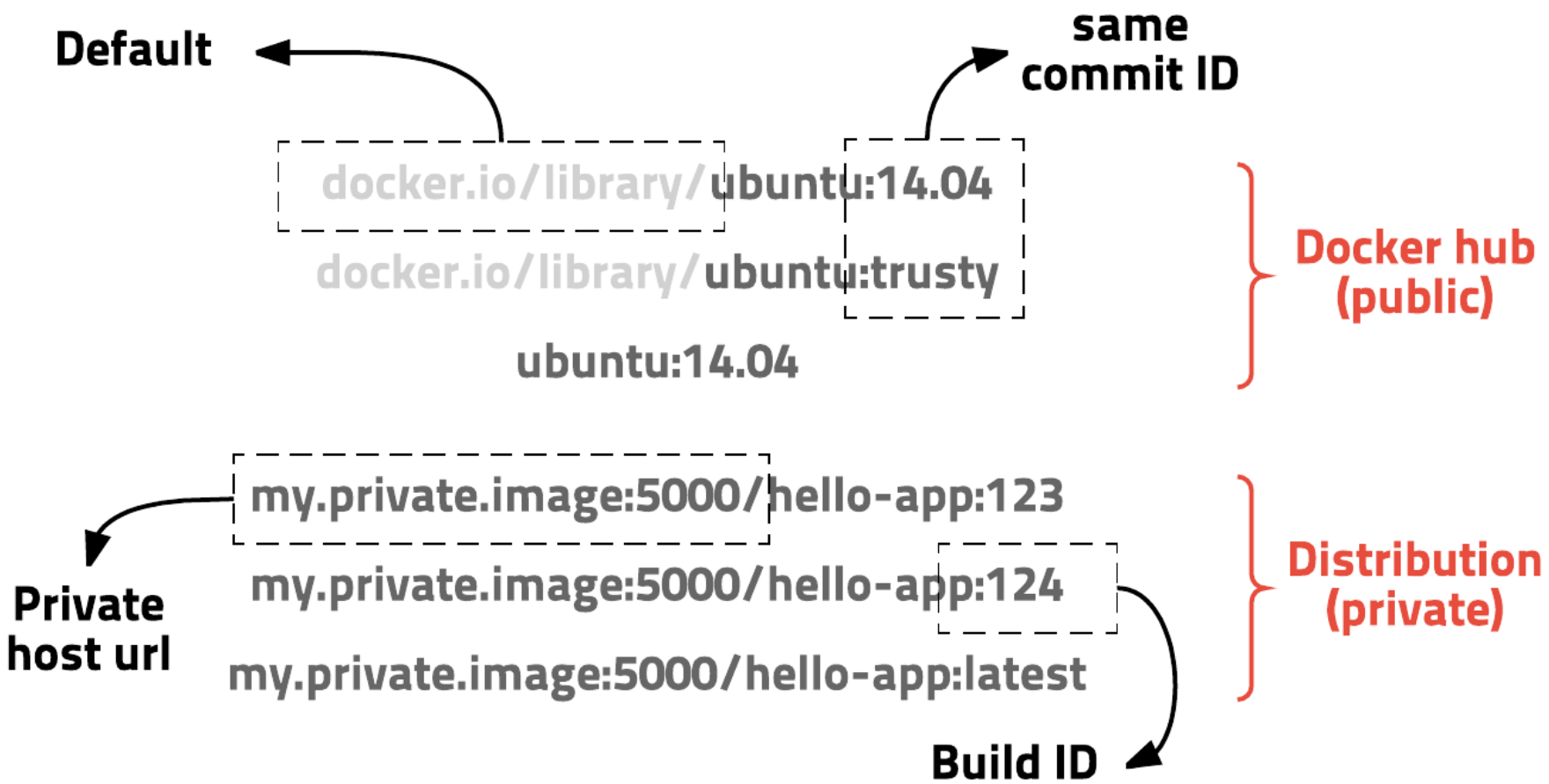
도커 아키텍처

- 도커 데몬
  - 도커 API 요청수신, 이미지, 컨테이너, 네트워크와 같은 도커 객체 및 도커 서비스 관리.
- 도커 클라이언트
  - 도커 사용자가 도커와 상호작용하기 위한 방법. 기본적인 도커 명령어를 통해서 도커 데몬과 통신.
- 도커 레지스트리
  - 도커 이미지를 저장, 도커 허브라는 공용 레지스트리와 개인 private한 레지스트리가 있다.
- 도커 객체
  - 도커 이미지: 도커 이미지는 컨테이너 실행에 필요한 파일과 설정 값 등을 포함하고 있다.
  - 컨테이너: 컨테이너는 도커 이미지의 실행 가능한 인스턴스.

도커 이미지

- 도커 이미지는 컨테이너 실행에 필요한 파일과 설정 값 등을 포함하고 있다.
- 컨테이너에 따른 상태 값이 변하지 않으므로 라이브러리의 버전이 의도치 않게 바뀜에 따른 의존성 문제가 발생하지 않는다.
- 컨테이너는 이미지를 실행한 상태이며, 변하는 값은 컨테이너에 저장된다.
- 같은 이미지에서 여러 개의 컨테이너를 생성할 수 있다.

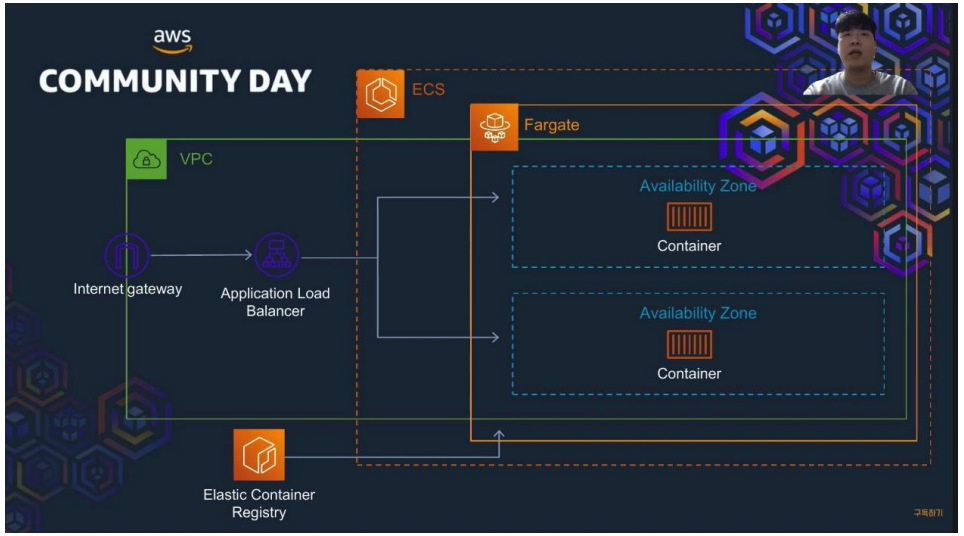
도커 이미지의 경로


Default
same
commit ID
docker.io/library/ubuntu:14.04
docker.io/library/ubuntu:trusty
ubuntu:14.04
Docker hub
(public)
my.private.image:5000/hello-app:123
Private
host url
my.private.image:5000/hello-app:124
my.private.image:5000/hello-app:latest
Build ID
Distribution
(private)

AWS ECS

AWS Elastic Container Service(ECS)

- AWS ECS는 클러스터에서 도커 컨테이너를 손쉽게 실행, 중지 및 관리할 수 있게 해주는 컨테이너 관리 서비스로 확장성과 속도가 뛰어나다.
- 특정 리전 내 여러 가용 영역에 걸쳐 고 가용성 방식으로 컨테이너를 실행하는 과정을 간소화하는 리전 서비스.



AWS ECS 사용시 이점

- 간단한 API 호출을 사용하여 컨테이너 기반 애플리케이션을 시작 및 중지할 수 있다.
- 중앙 집중식 서비스를 사용하여 클러스터 상태를 확인할 수 있다.
- 다수의 친숙한 EC2 기능에 액세스할 수 있다.
- 일관된 배포 및 구축 환경을 생성하고, 배치 및 ETL(Extract-Transform-Load) 워크로드를 관리 및 크기 조정하고, 마이크로 서비스 모델에 정교한 애플리케이션 아키텍처를 구축할 수 있다.

AWS ECS 구성 요소

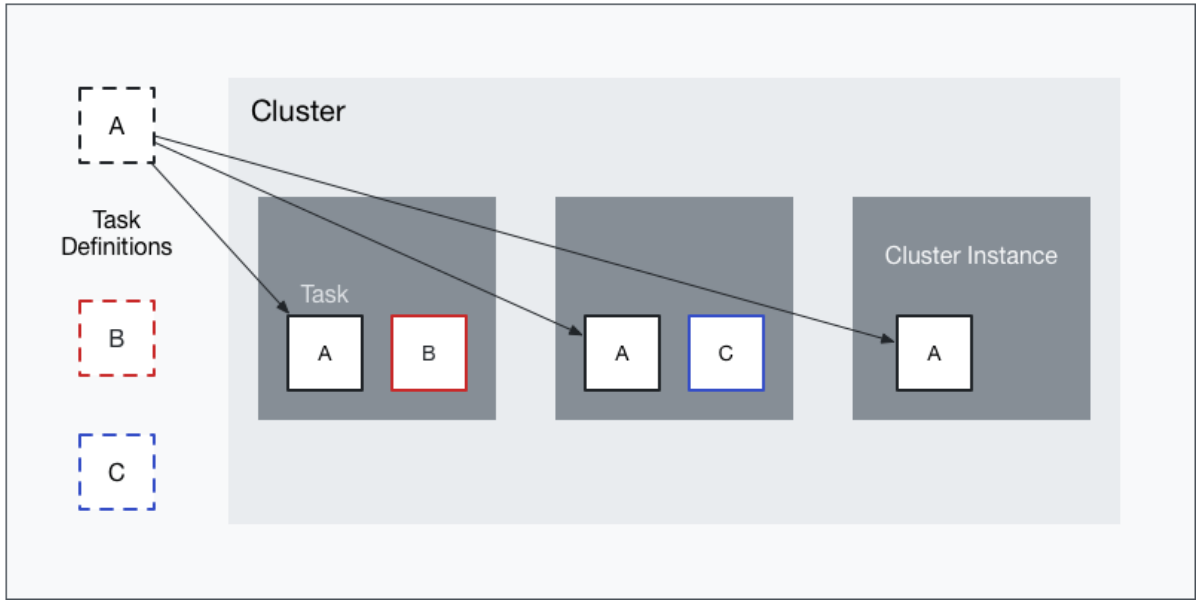
- Task definition
- Task
- Service
- Container Instance
- Cluster

Task definition

- 원하는 도커 컨테이너를 생성할 때, 어떤 설정으로 몇 개 이상 생성 할 지를 정의한 set
- Task definition에는 한 개 의상의 컨테이너에 대해 정의가 가능하며, Task definition 내부에 정의된 컨테이너 사이는 link 설정으로 연결이 가능하다.
- Task definition에서 정의된 대로 실제 생성된 container set들을 Task라고 부른다.

Task

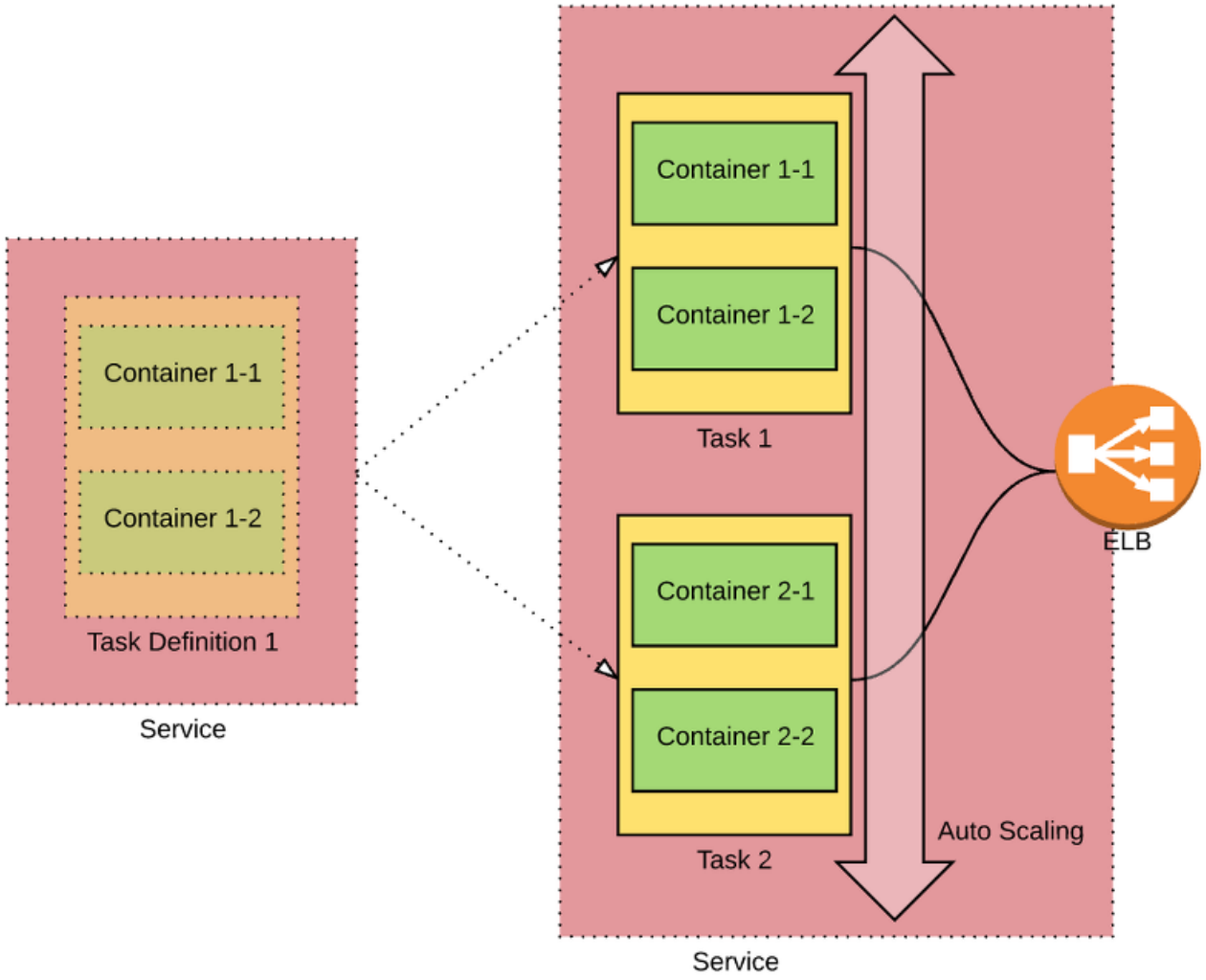
- Task definitio에서 정의된 대로 배포된 container set을 Task라고 부른다.
- Task는 Cluster에 속한 Container instance(EC2 Instance)에 배포된다.
- 또한, Task는 여러 Container instance(EC2 Instance)에 배포 가능하다.



Service

- Task들의 Life cycle을 관리하는 부분을 Service라고 한다.
- 각 Task들은 각자 다른 서비스이다.
- Task를 Cluster에 몇 개나 배포할 것인지 결정하고, 실제 Task들을 외부에 서비스 하기 위해 ELB에 연동되는 부분을 관리하게 된다.
- 만약 실행중인 Task가 어떤 이유로 작동이 중지되면 이것을 자동으로 감지해 새로운 Task를 Cluster에 배포하는 고가용성에 대한 정책도 Service에서 관리한다.
- 즉, 오토스케일링과 로드밸런싱을 관리하는 역할이다.

Task / Service 관계

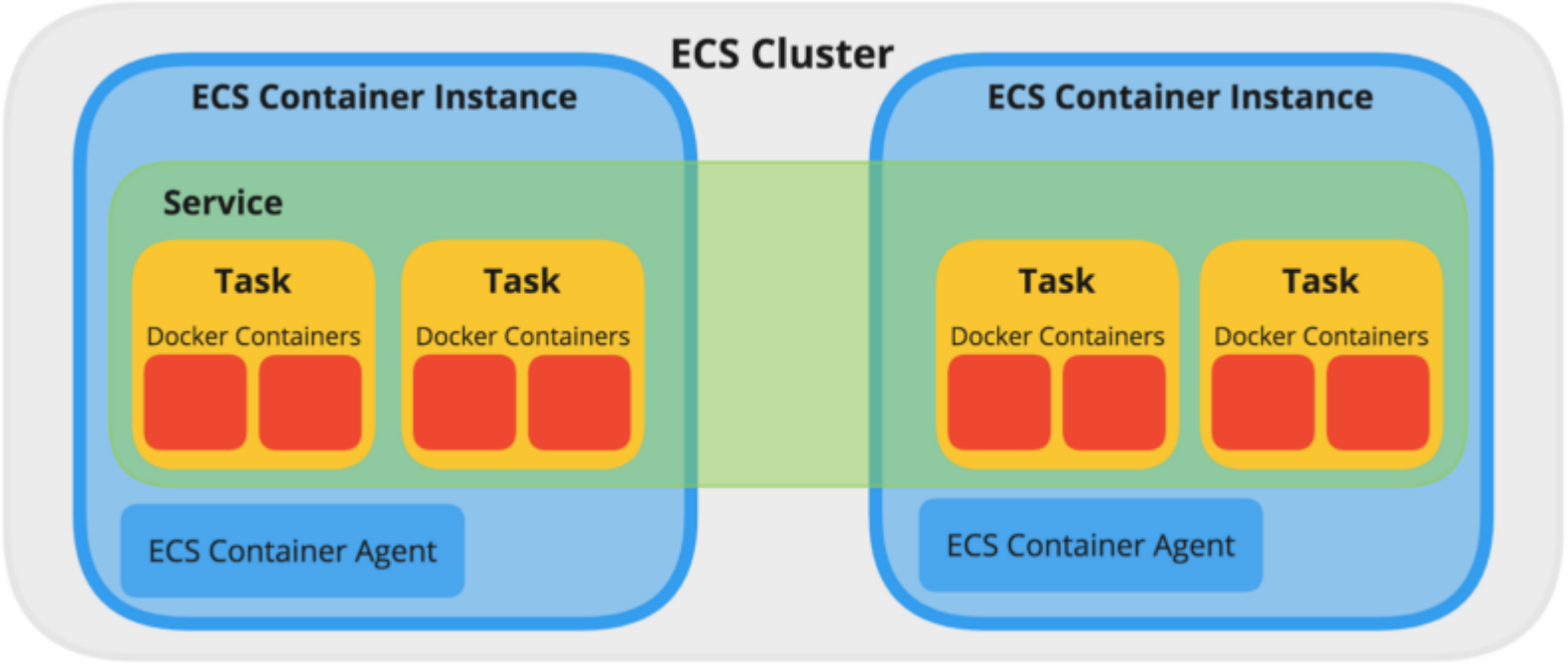


Container Instance

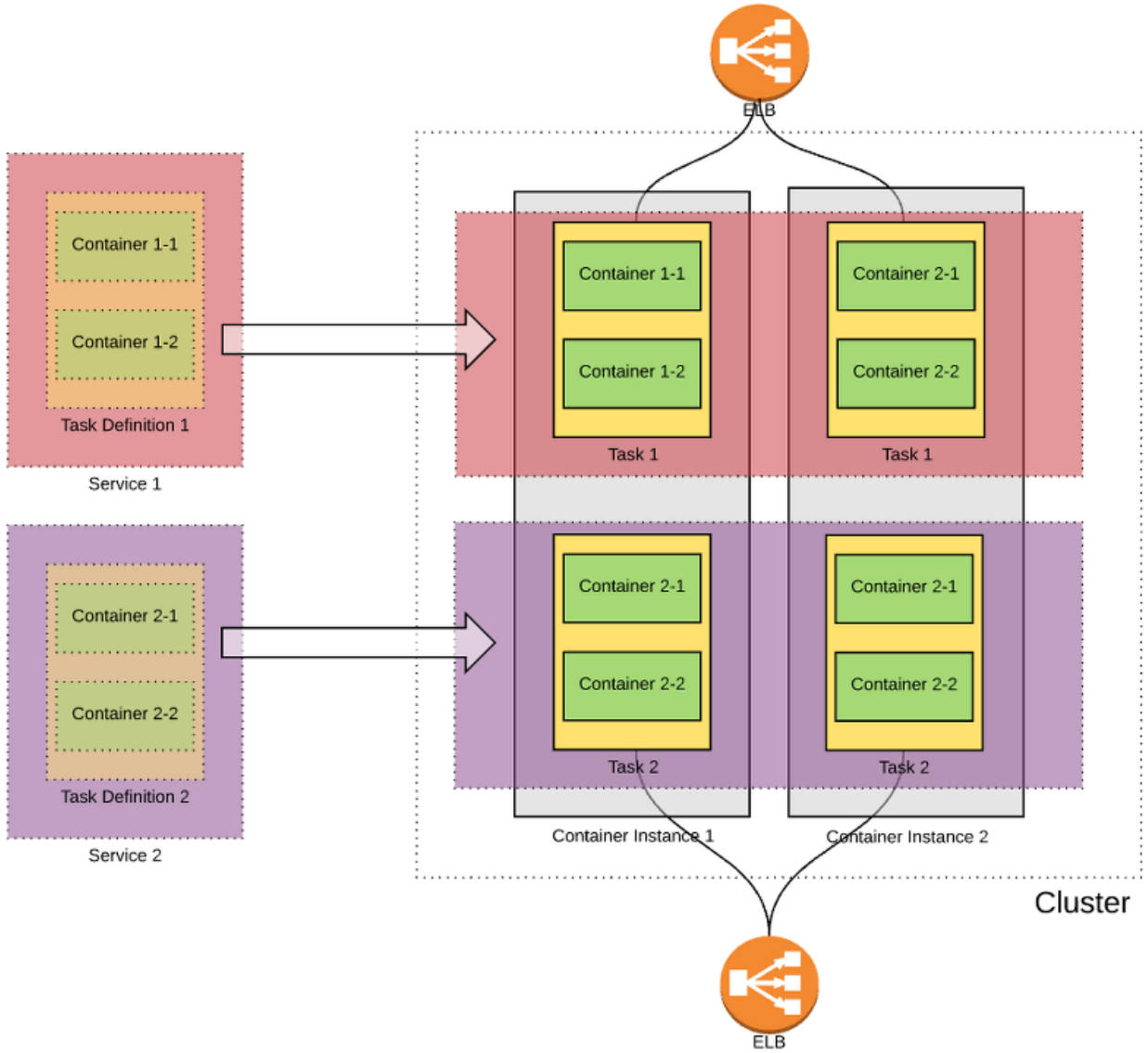
- ECS는 기본적으로 Container 배포를 EC2 Instance 기반에 올리도록 설계됨.
- Task를 올리기 위해 등록된 EC2 Instance를 Container Instance라고 부른다.
- 하나의 Container Instance 내부에는 각각의 다른 Task가 여러 개 있을 수 있다.

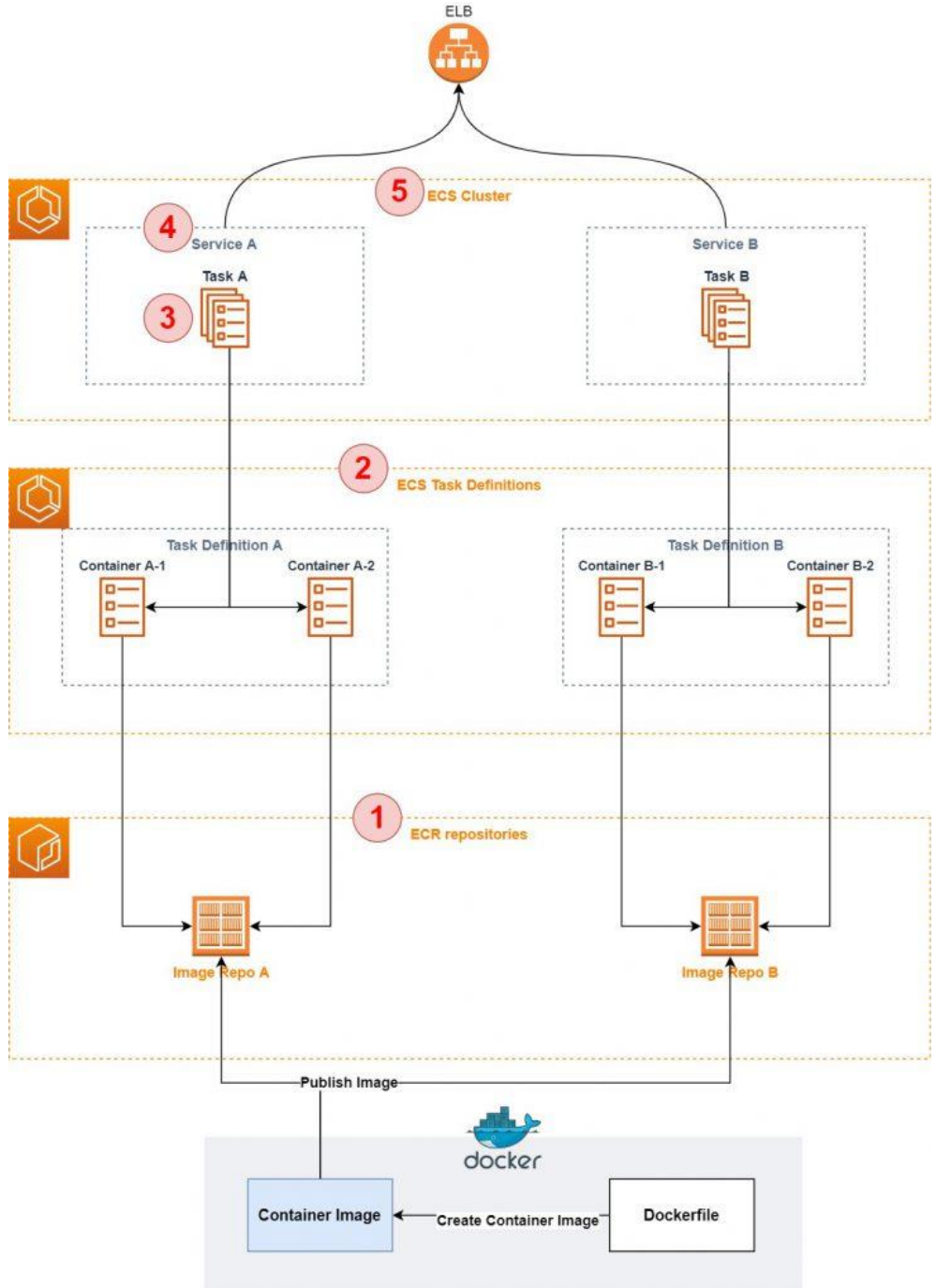
Cluster

- Task가 배포되는 Container Instance들을 논리적인 그룹으로 묶은 단위.
- Task를 배포하기 위한 instance는 반드시 Cluster에 등록되어야 한다.



Cluster 관계 도식화




ELB
5
ECS Cluster
4
Service A
Task A
3
Service B
Task B
2
ECS Task Definitions
Task Definition A
Container A-1
Container A-2
Task Definition B
Container B-1
Container B-2
1
ECR repositories
Image Repo A
Image Repo B
Publish Image
docker
Container Image
Create Container Image
Dockerfile

ECS의 구조적 특징

1. ECS의 Docker host 역할은 EC2 Instance가 담당한다.
2. Overlay network가 아닌 ELB를 이용한 Network 구조.
3. 하나의 Host에는 한 개의 Task만 수용 가능하다.
4. ECS 특성에 맞는 Application 설계가 필요하다.