



CJ OLIVEYOUNG

Amazon ECS Overview

안성진 SA

Amazon Web Services

Agenda

ECS 소개

ECS 오브젝트

AWS Fargate on Amazon ECS

Amazon ECS Networking

Amazon ECS Developer Tools



ECS 소개



AWS 컨테이너 서비스 포트폴리오

MANAGEMENT

배포, 스케줄링, 컨테이너화된
애플리케이션 스케일링 & 관리



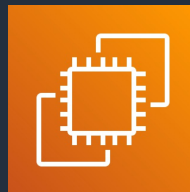
Amazon ECS



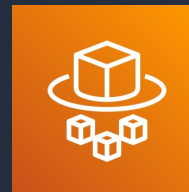
Amazon EKS

HOSTING

컨테이너가 동작하는 환경



Amazon EC2



AWS Fargate

IMAGE REGISTRY

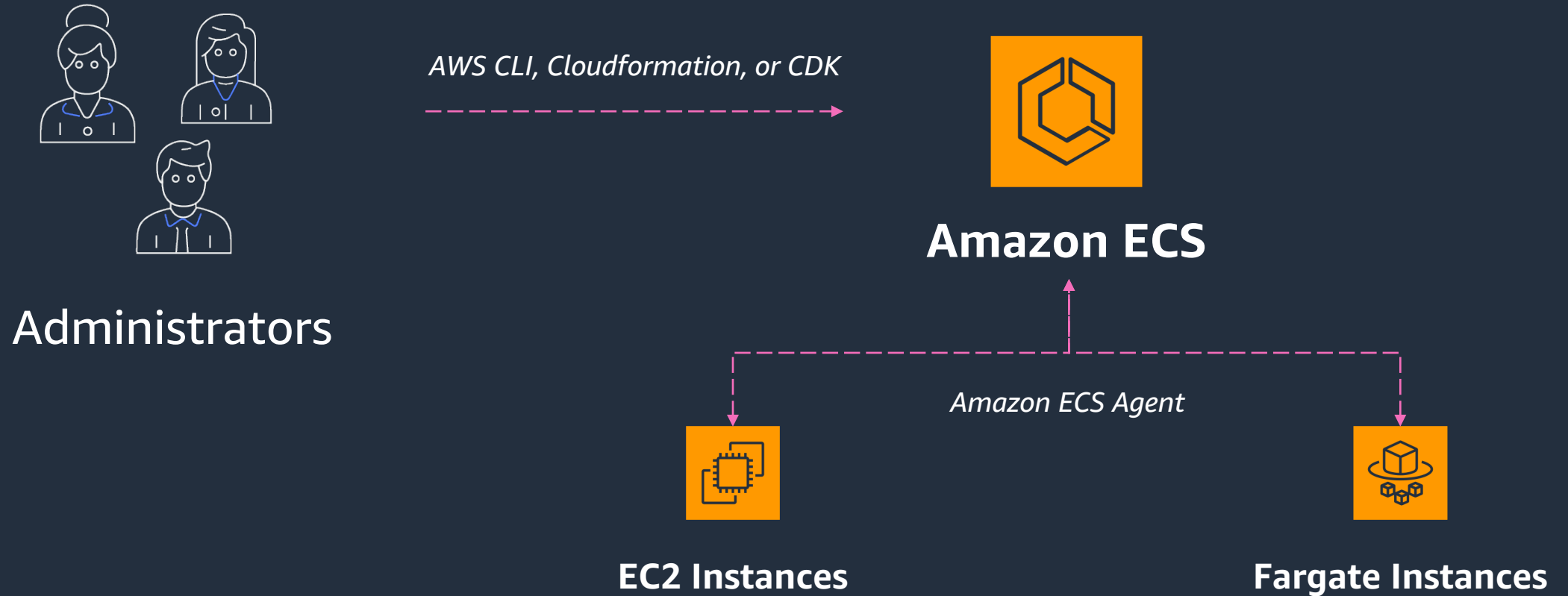
컨테이너 이미지 저장소



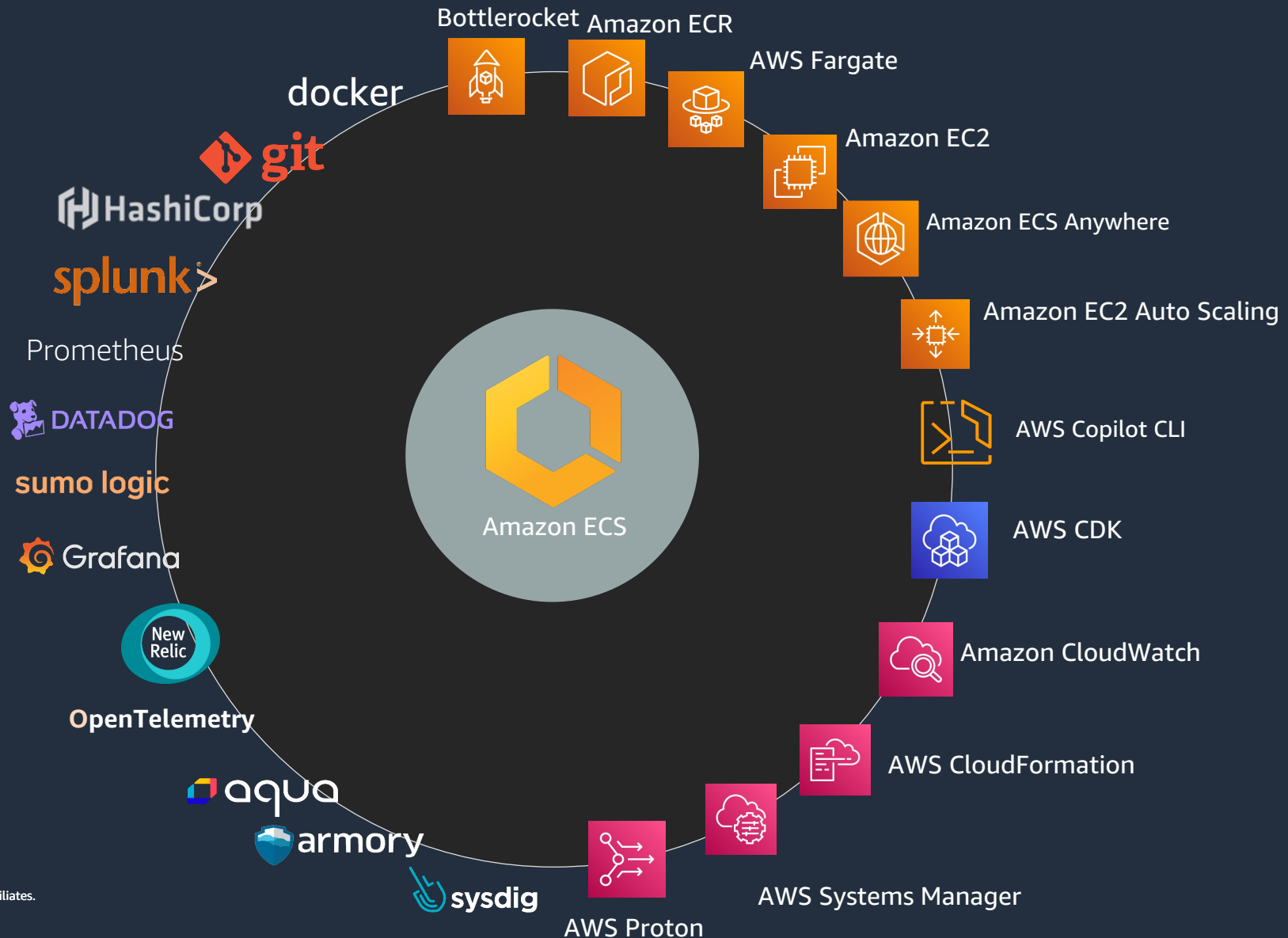
Amazon ECR



Amazon ECS 아키텍처



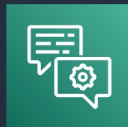
Amazon ECS를 지원하는 통합 에코시스템



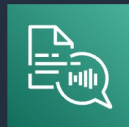
Amazon ECS powers Amazon



Amazon SageMaker



Amazon Lex



Amazon Polly



AWS Batch



Amazon.com recommendation engine

-
- Amazon ECS forms the building blocks for various services at Amazon
 - Tested for security, reliability, availability, and scale

Amazon ECS Objects



Amazon ECS 구성 요소



ECS 클러스터

작업이 실행되는 논리적 그룹



ECS 작업

실제 컨테이너 작업(최소 실행 단위)
하나 혹은 둘 이상의 컨테이너의 묶음

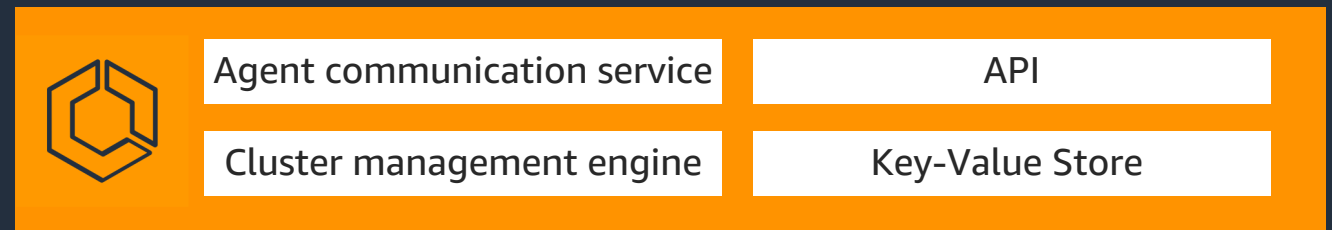
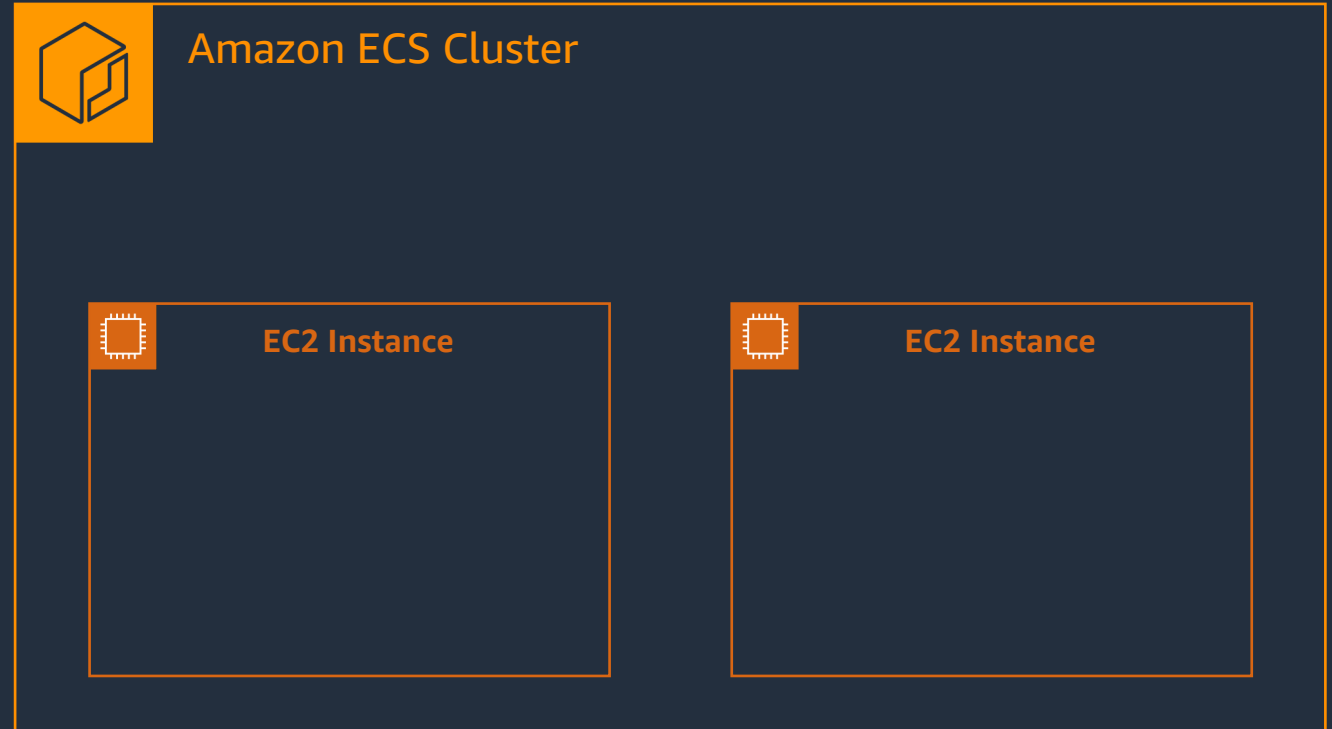


ECS 서비스

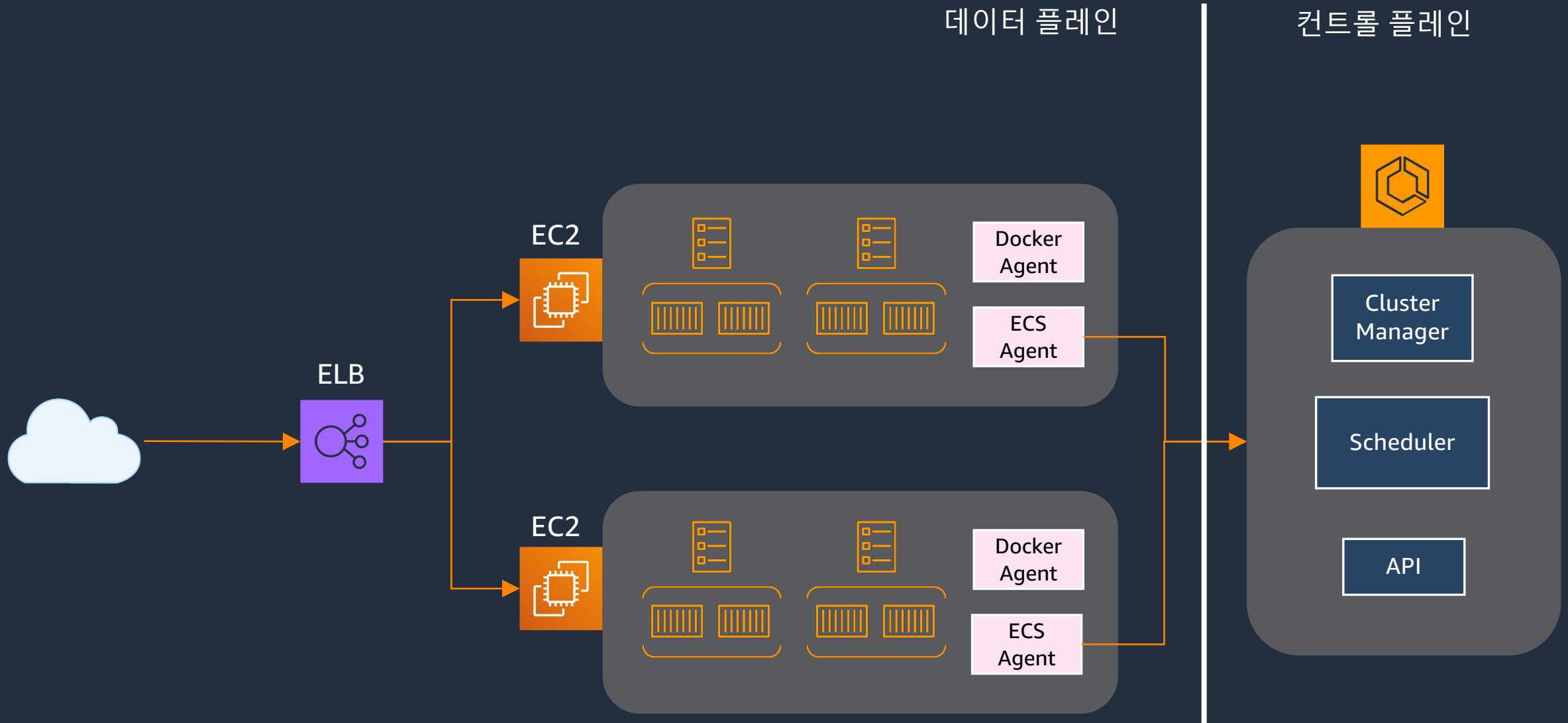
ECS 위에서 작업을 실행하는 방법
정의한 작업 수를 유지하기 위해 지속적으로
관리

Amazon ECS 클러스터

- 작업이 실행되는 논리적인 그룹
- Control Plane / Data Plane으로 나뉘어 구성



ECS 주요 구성 요소



Amazon ECS 작업

- ECS 클러스터에서 최소 실행 단위
 - 네트워킹, 스토리지, 파라미터, IAM 역할, 컴퓨팅 리소스 등의 구성 값 설정 가능
- **작업 정의(Task Definition)** 내용을 기반으로 작업 배포
 - 배포 타입 설정: Fargate, EC2, External
 - 컨테이너 이미지 매핑을 통한 정의 작업
 - 작업 역할을 부여해 API 요청을 받을 때 권한에 따라 동작
 - 작업 크기 설정
 - AppMesh, FireLens 등과의 통합 설정
- 하나의 작업 당 최대 10개 컨테이너 가능



Amazon ECS 작업 정의

JSON

JSON 다운로드

AWS CLI 입력 다운로드

클립보로 복사

task_definition.json

```
1 {
2   "taskDefinitionArn": "arn:aws:ecs:ap-northeast-2:123456789012:task-definition:dogs:1",
3   "containerDefinitions": [
4     {
5       "name": "dogs",
6       "image": "123456789012.dkr.ecr.ap-northeast-2.amazonaws.com/dogs",
7       "cpu": 0,
8       "portMappings": [
9         {
10          "name": "dogs-80-tcp",
11          "containerPort": 80,
12          "hostPort": 80,
13          "protocol": "tcp",
14          "appProtocol": "http"
15        }
16      ],
17       "essential": true,
18       "environment": [],
19       "mountPoints": [],
20       "volumesFrom": [],
21       "logConfiguration": {
22         "logDriver": "awslogs",
23         "options": {
24           "awslogs-create-group": "true",
25           "awslogs-group": "/ecs/dogsdef",
26           "awslogs-region": "ap-northeast-2",
27           "awslogs-stream-prefix": "ecs"
28         }
29       }
30     }
31   ]
32 }
```

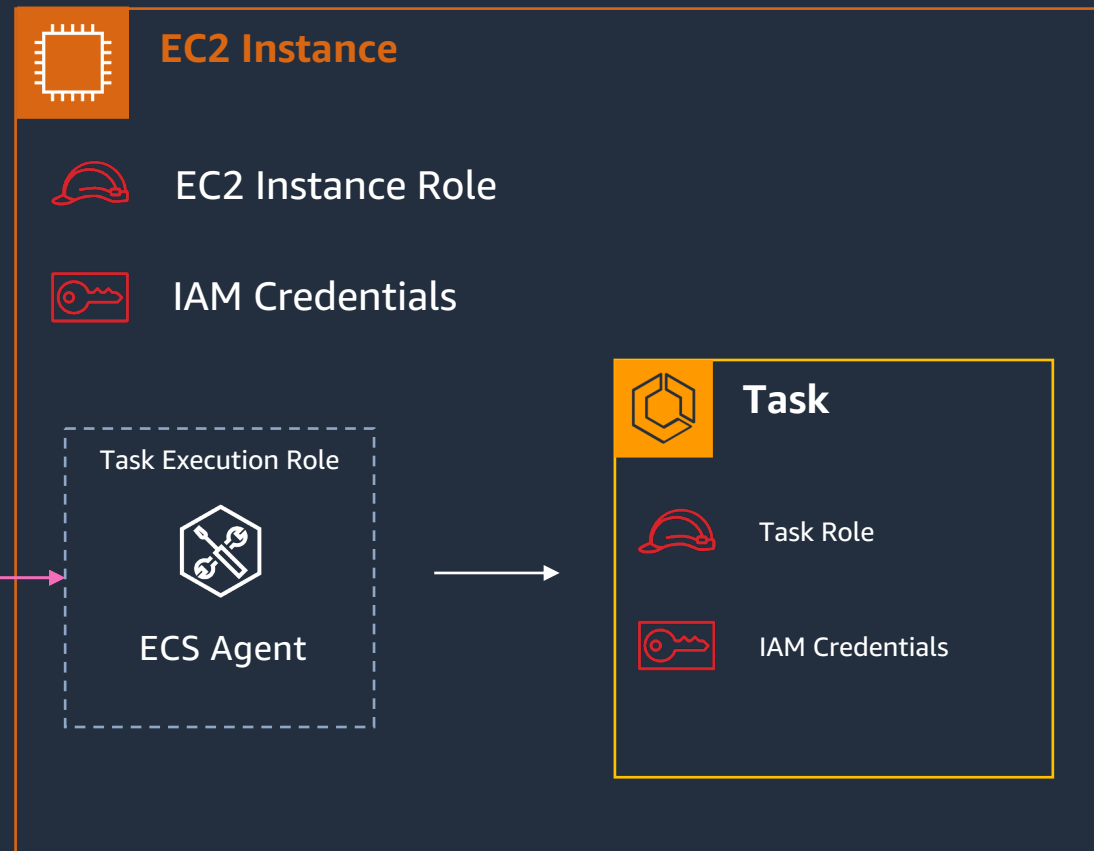
task_definition.json

```
46 {
47   "name": "com.amazonaws.ecs.capability.ecr-auth",
48 },
49 {
50   "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19",
51 },
52 {
53   "name": "ecs.capability.execution-role-ecr-pull",
54 },
55 {
56   "name": "com.amazonaws.ecs.capability.docker-remote-api.1.29",
57 },
58 ],
59 "placementConstraints": [],
60 "compatibilities": [
61   "EC2"
62 ],
63 "requiresCompatibilities": [
64   "EC2"
65 ],
66 "cpu": "512",
67 "memory": "1024",
68 "runtimePlatform": {
69   "cpuArchitecture": "X86_64",
70   "operatingSystemFamily": "LINUX"
71 },
72 }
```



ECS 작업에 부여하는 IAM 역할

호스트에 있는 권한을 상속받지 않고
작업(Task)에 IAM 역할을 할당



ECS 작업에 부여하는 IAM 역할

JSON

JSON 다운로드

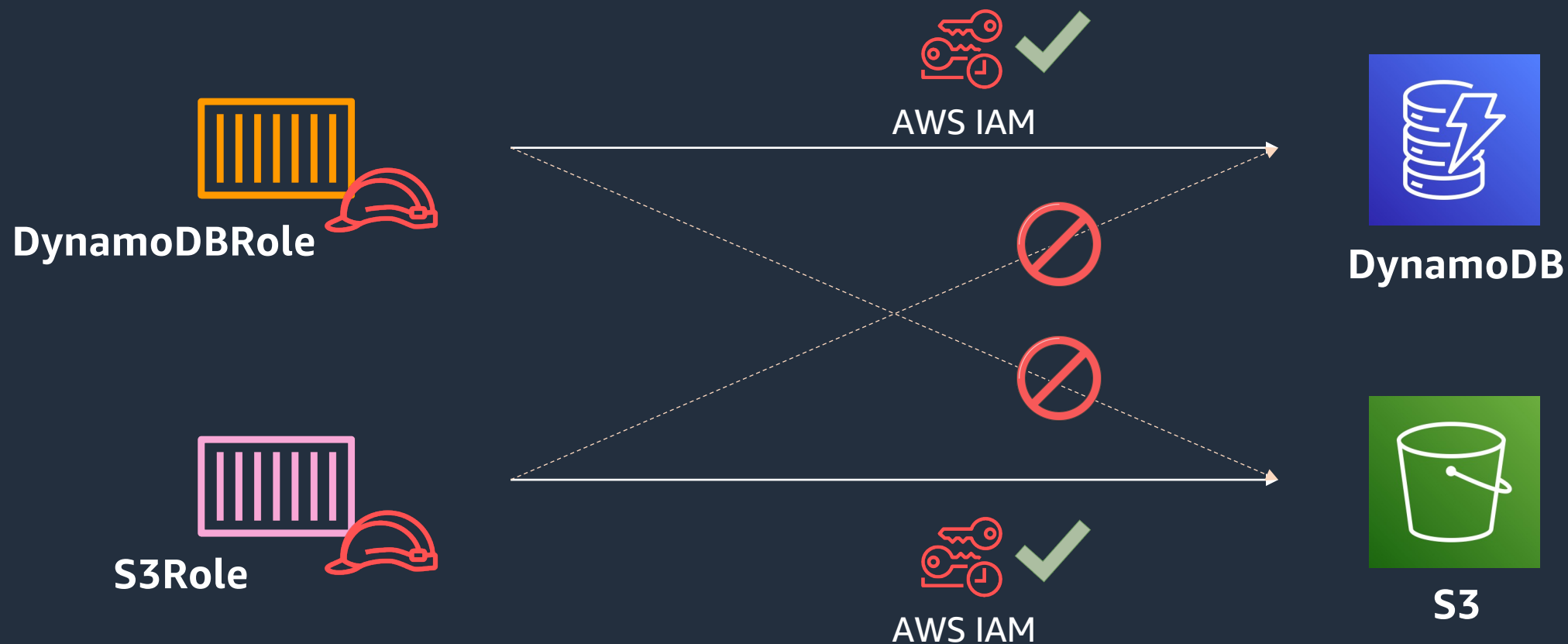
AWS CLI 입력 다운로드

클립보드로 복사

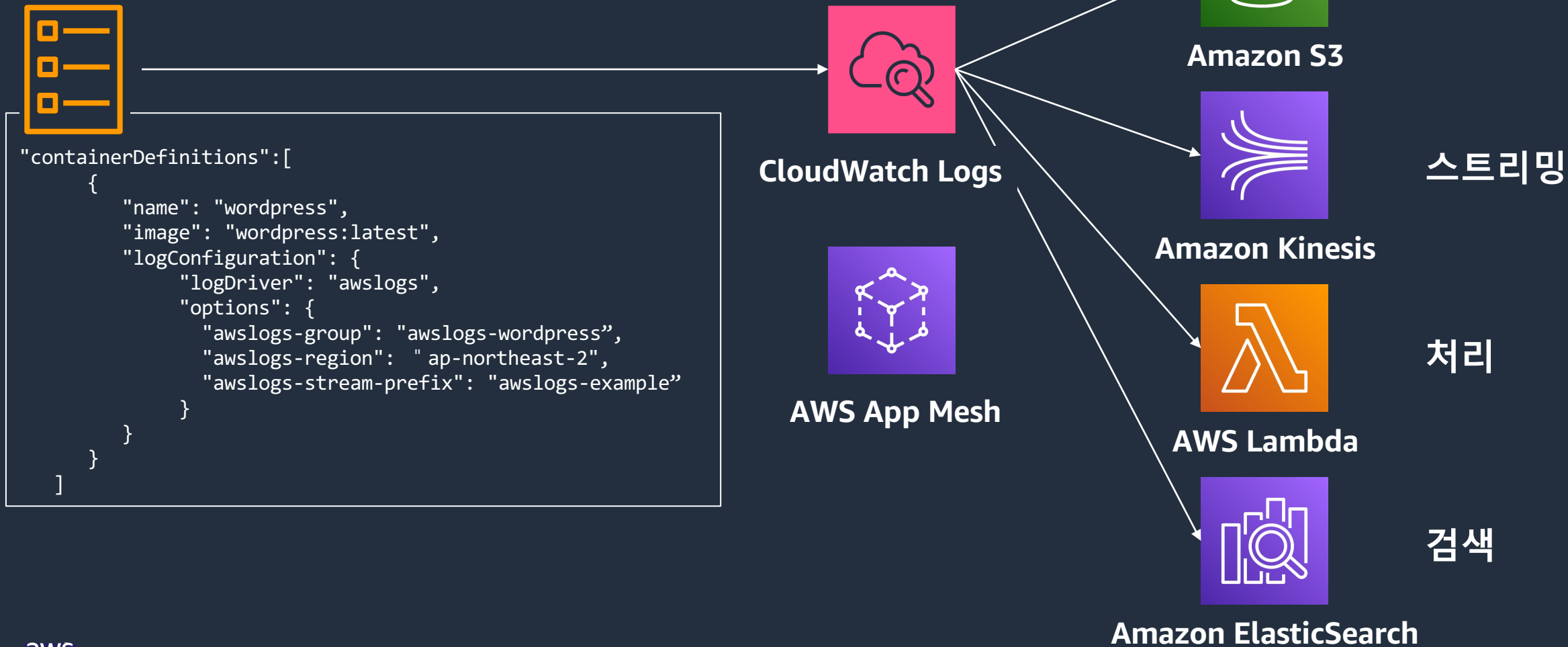
task_definition.json

```
--
12     "hostPort": 80,
13     "protocol": "tcp",
14     "appProtocol": "http"
15   }
16 ],
17   "essential": true,
18   "environment": {},
19   "mountPoints": [],
20   "volumesFrom": [],
21   "logConfiguration": {
22     "logDriver": "awslogs",
23     "options": {
24       "awslogs-create-group": "true",
25       "awslogs-group": "/ecs/dogsdef",
26       "awslogs-region": "ap-northeast-2",
27       "awslogs-stream-prefix": "ecs"
28     }
29   }
30 },
31 ],
32 "family": "dogsdef",
33 "taskRoleArn": "arn:aws:iam::[redacted]:role/ecs-api-monolith-TaskRole-OAV50HL15SGJ",
34 "executionRoleArn": "arn:aws:iam::[redacted]:role/ecsTaskExecutionRole",
35 "networkMode": "awsvpc",
36 "revision": 17,
37 "volumes": [],
38 "status": "ACTIVE",
39 "requiresAttributes": [
40   {
```

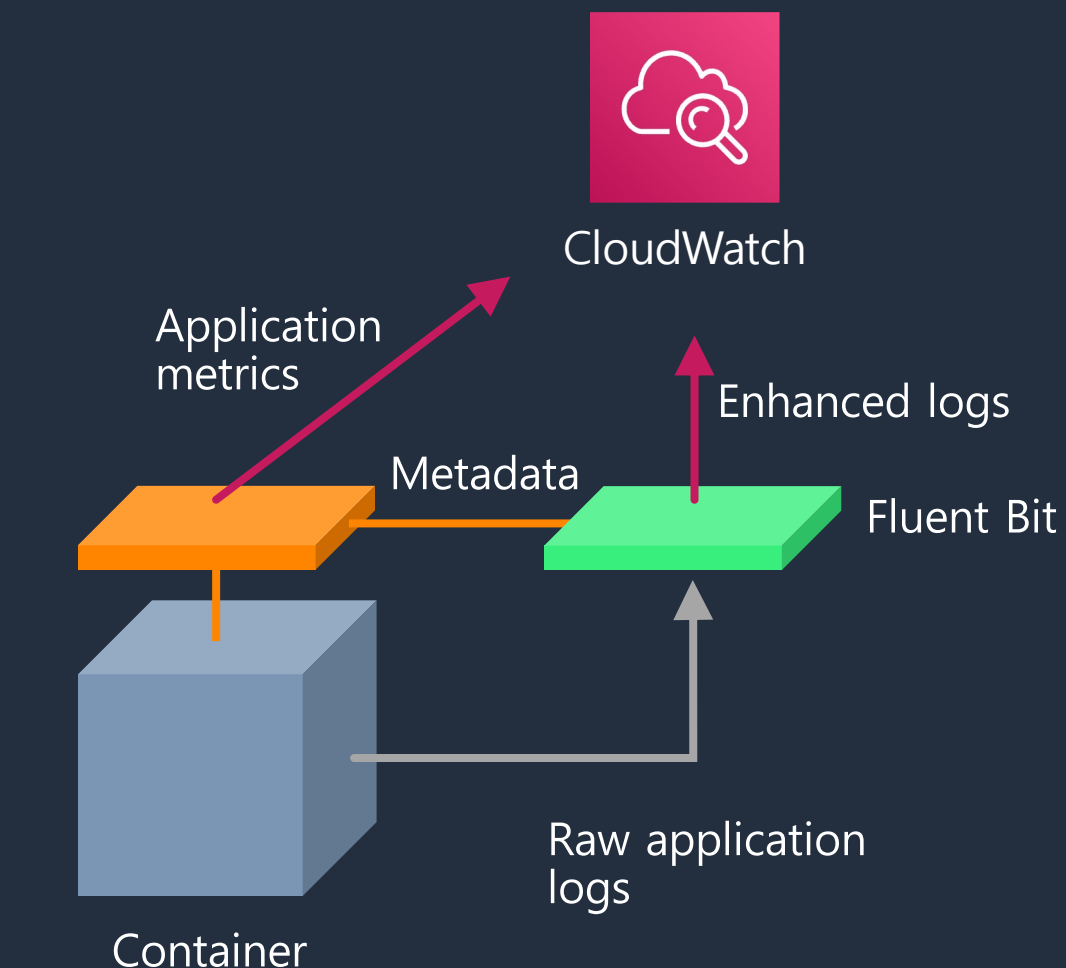
ECS 작업에 부여하는 IAM 역할



다른 서비스들과의 손쉬운 연동



FireLens를 통한, 로그 라우팅



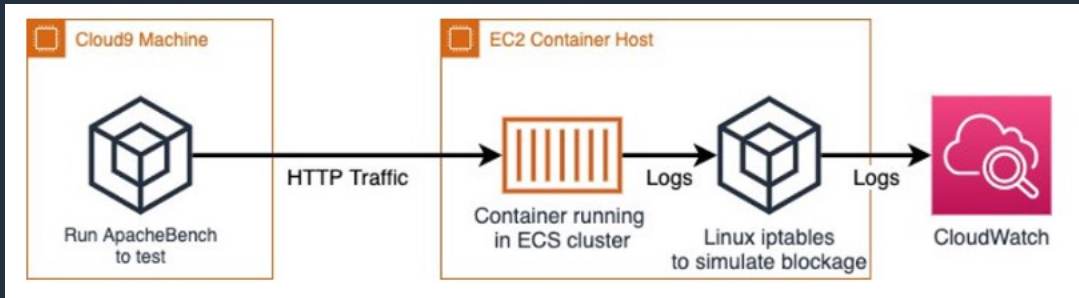
Amazon ECS를 위한 컨테이너 로그 라우터

로그를 필터링 및 라우팅하는 Fluent Bit
사이드카를 자동으로 생성

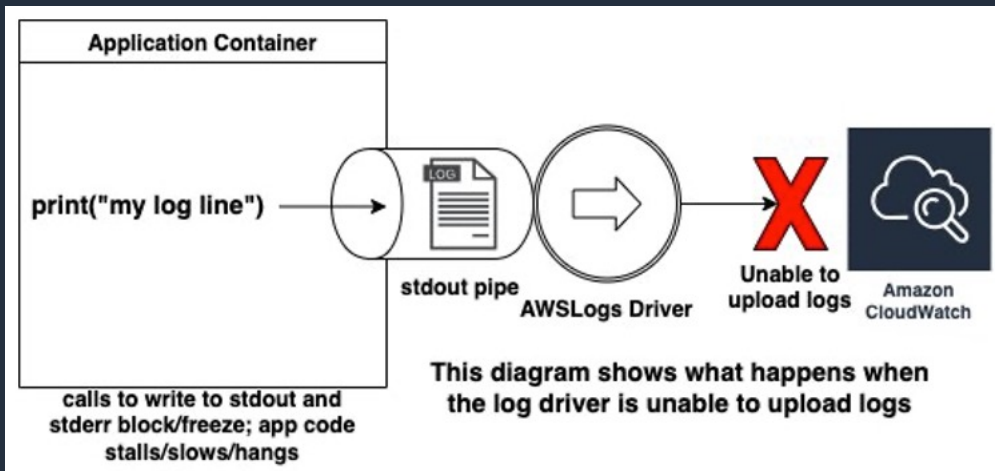
특정 EC2 인스턴스, 작업 ID, 애플리케이션
버전과 같은 추가 메타데이터를 로그에 넣을
수 있음

SaaS Provider로 전달 가능

ECS Log Driver 관련 블로그 (blocking vs non-blocking)



<https://aws.amazon.com/ko/blogs/containers/choosing-container-logging-options-to-avoid-backpressure/>



<https://aws.amazon.com/ko/blogs/containers/preventing-log-loss-with-non-blocking-mode-in-the-awslogs-container-log-driver/>

다양한 볼륨 선택



컨테이너 인스턴스 `/var/lib/docker/volumes` 위치에 도커 관리형 볼륨이 생성됨
도커 볼륨 드라이버(플러그인이라고도 함)들은 Amazon EBS와 같은 외부 스토리지 시스템과 통합하는데 사용됨



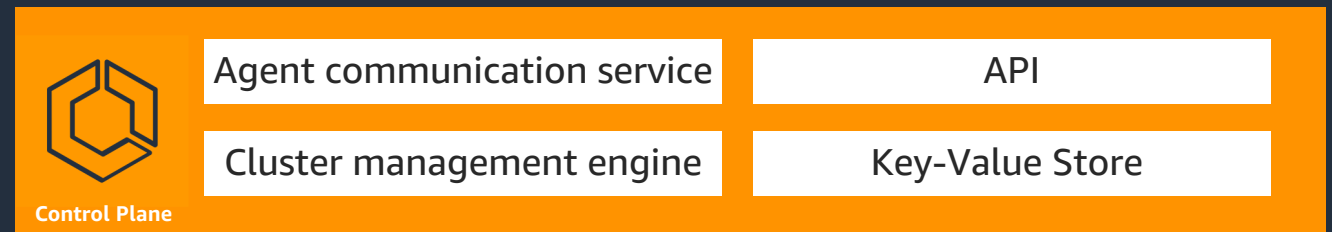
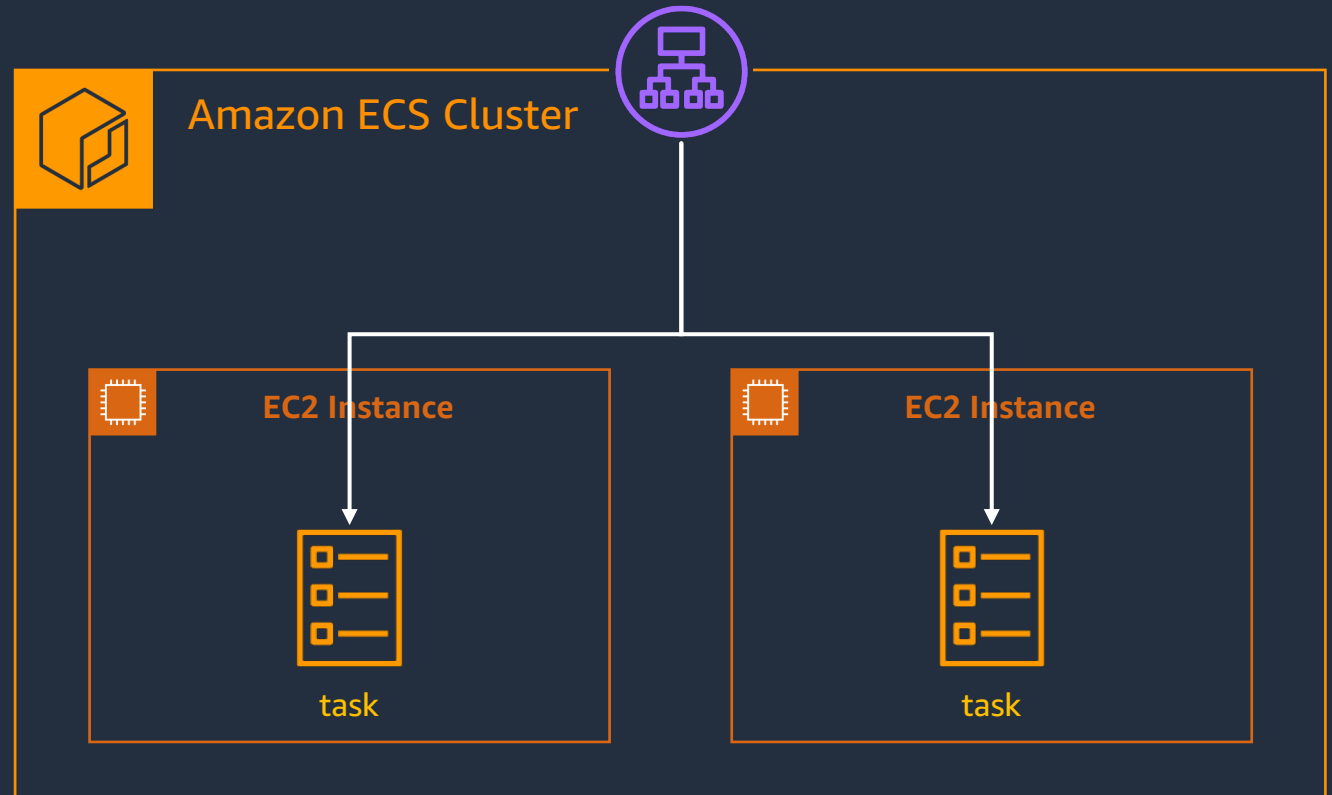
호스트 머신에 있는 파일 혹은 디렉토리를 컨테이너에 마운트
EC2 혹은 Fargate 런치 타입에 상관없이 호스트 볼륨 바인딩 지원



Amazon Elastic File System (EFS)를 컨테이너에 마운트

Amazon ECS 서비스

- ECS 위에서 작업을 실행하는 방법
- 주요 설정 내용
 - 서비스 유형(replica, daemon)
 - 작업 배치 전략
 - 배포 유형
- AWS ELB를 통한 효율적 부하 분산
- Service Auto Scaling을 통한 탄력적 워크로드 구성



Service 배포

```
{
  "cluster": "ecs-sarathy-cluster",
  "serviceName": "PythonService",
  "taskDefinition": "PythonTask:1",
  "loadBalancers": [
    {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:937351930975:targetgroup/XXX",
      "containerName": "python-service",
      "containerPort": 5000
    }
  ],
  "serviceRegistries": [
    {
      "registryArn": "arn:aws:servicediscovery:us-east-1:937351930975:service/YYY"
    }
  ],
  "desiredCount": 5,
  "schedulingStrategy": "REPLICA",
  "launchType": "EC2"
}
```

Load Balancer

Service Registry

Strategy

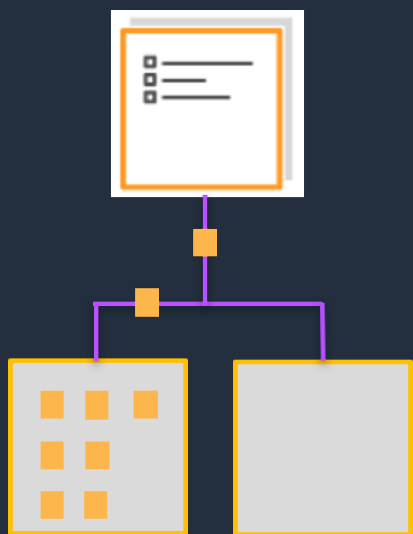
Service 생성

- ECS 콘솔 사용
- JSON 파일과 함께 AWS CLI 사용

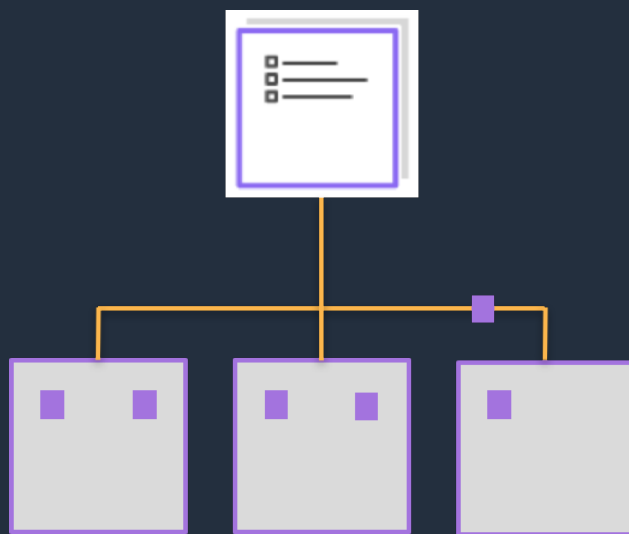
Service 생성을 위한 CLI 명령

aws ecs create-service --cli-input-json
file://pythonService.json

다양한 작업 배치 전략 (ft. Scheduler)



Binpacking



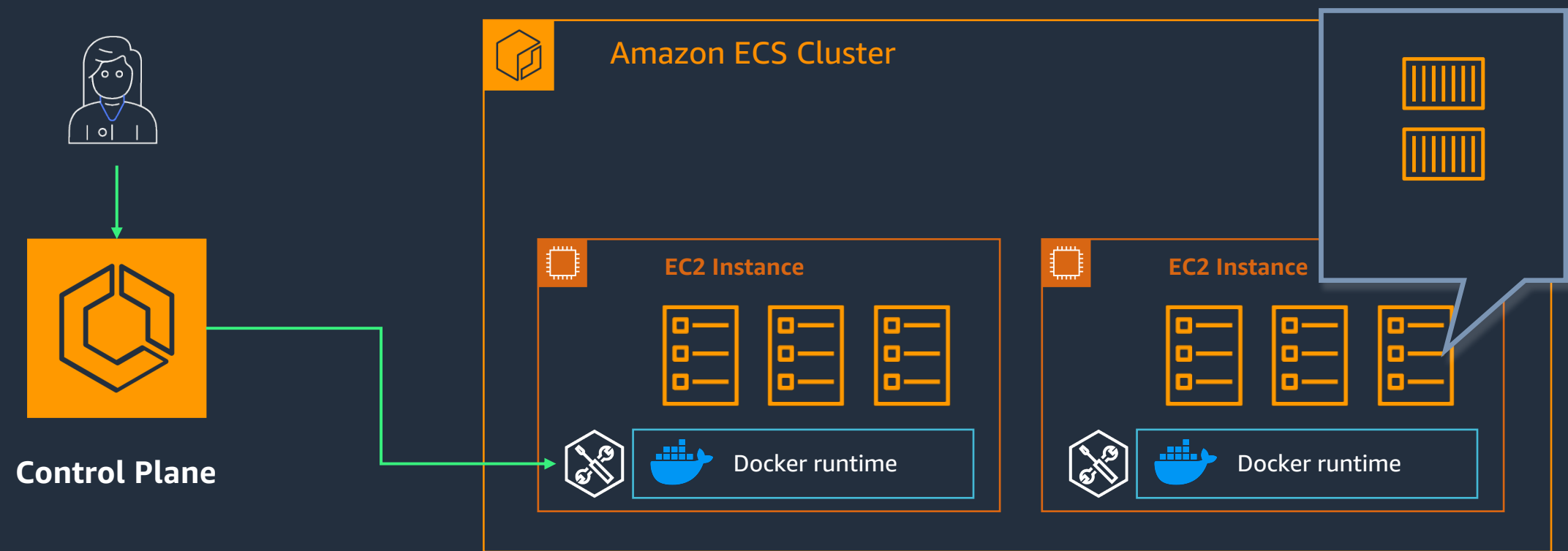
Spread

...

Task 배치 과정

1. Resource 식별 (CPU, Memory, GPU, Port ...)
2. 작업 배치 제약 (m5.large, g4dn.large ...)
3. 작업 배치 전략 (Binpack, AZ Spread ...)
4. Task 배치

Amazon ECS Objects



AWS Fargate on Amazon ECS



AWS Fargate의 장점



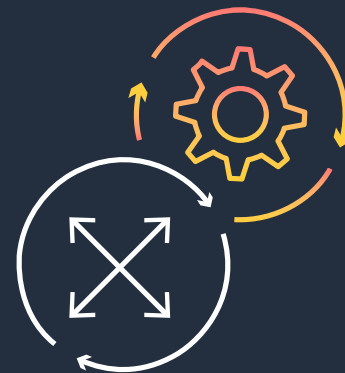
관리 업무로 인한 오버헤드 경감

까다로운 컨테이너 클러스터 관리를
AWS에 위임함으로써 고객은
애플리케이션에만 집중



기존 컨테이너 그대로 배포 가능

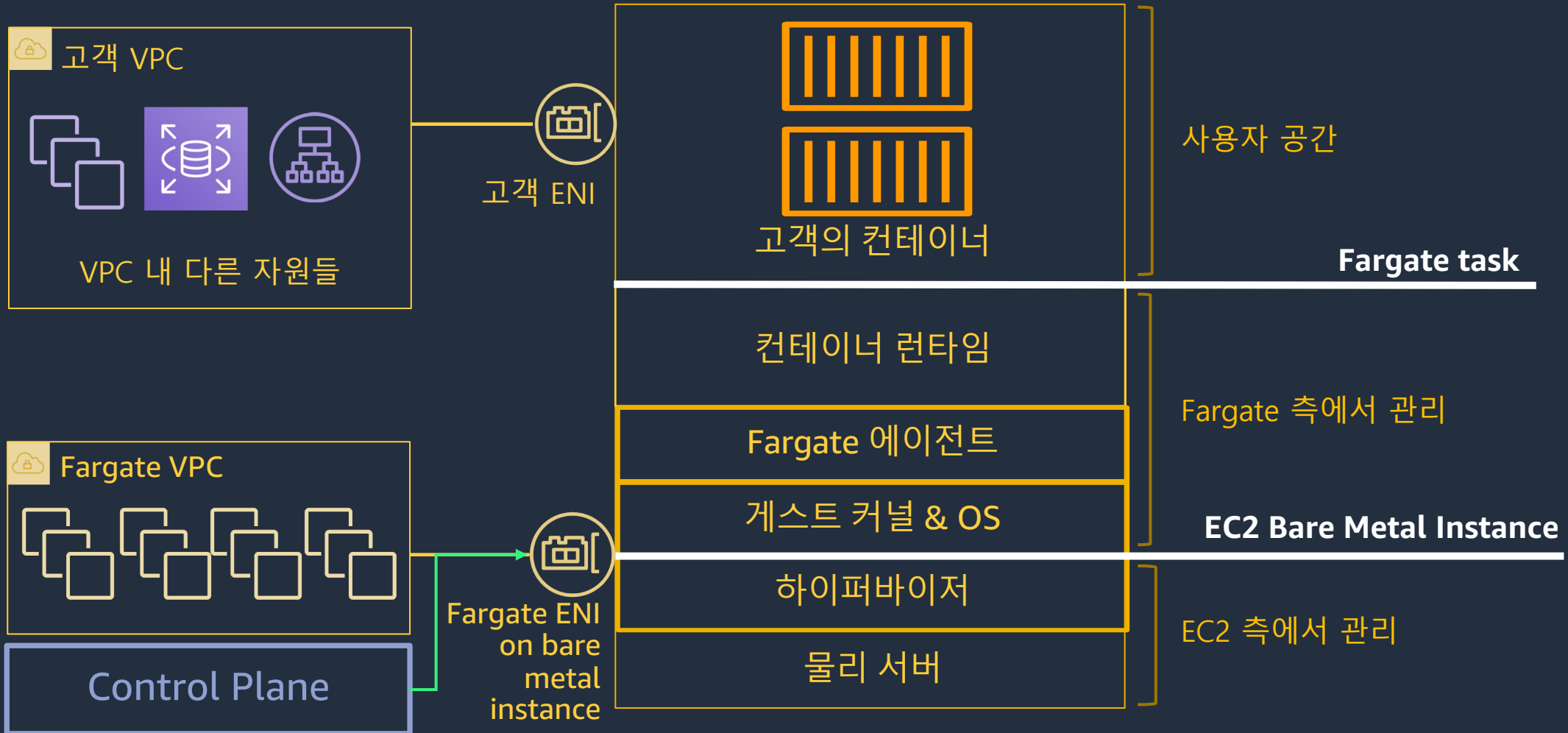
기존의 컨테이너 변경 불필요
현재 쿠버네티스, ECS 클러스터의
서비스, 워크플로우 그대로 이용



필요한 만큼만 & 쉬운 연동

컨테이너 실행에 필요한 자원 만큼만
비용 지불
기존 AWS 네트워크, 보안과
네이티브하게 통합하여 사용

AWS Fargate 내부 파헤쳐 보기



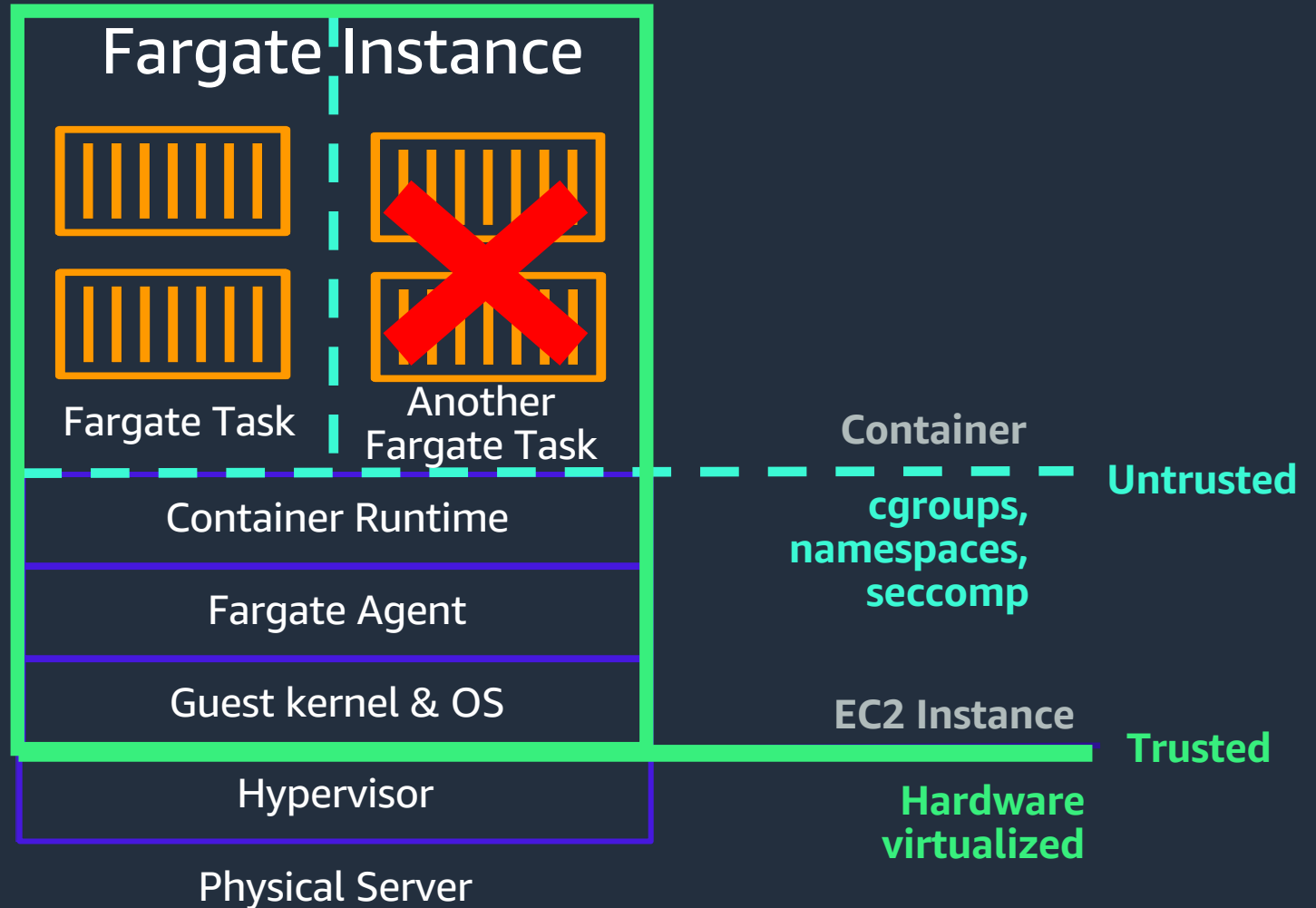
AWS Fargate Task CPU와 메모리

CPU	Memory
256 (.25 vCPU)	512MB, 1GB, 2GB
512 (.5 vCPU)	1GB to 4GB
1024 (1 vCPU)	2GB to 8GB
2048 (2 vCPU)	4GB to 16GB
4096 (4 vCPU)	8GB to 30GB
8192 (8 vCPU)	16 GB to 60 GB
16384 (16 vCPU)	32 GB to 120 GB

- 0.5 vCPU부터는 메모리 할당이 1GB 단위로 가능
- 8 vCPU는 메모리 할당이 4 GB 단위로 가능
- 16 vCPU는 메모리 할당이 8 GB 단위로 가능



```
{
  "family": "sample-fargate",
  "networkMode": "awsvpc",
  "containerDefinitions": [
    {
      "name": "fargate-app",
      "image": "httpd:2.4",
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "entryPoint": [
        "sh",
        "-c"
      ],
      "command": [
        "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top: 40px; background-color: #333;}</style> </head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon ECS.</p> </div></body></html>' > /usr/local/apache2/htdocs/index.html && httpd-foreground\""
      ]
    }
  ],
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "cpu": "256",
  "memory": "512"
}
```



One & only one task per EC2 instance

AWS Fargate를 도입하기 위해 고려해야 될 사항

- 호스트 OS 단에 구축해야하는 compliance 및 소프트웨어 요소가 있다면 사용 불가능
- GPU 지원 X
- 작업 정의 시, 네트워크 모드 중, awsvpc 모드만 지원
- 작업(Task) ENI에 고정 IP 주소 사용 불가능 (아래 해결 방법)

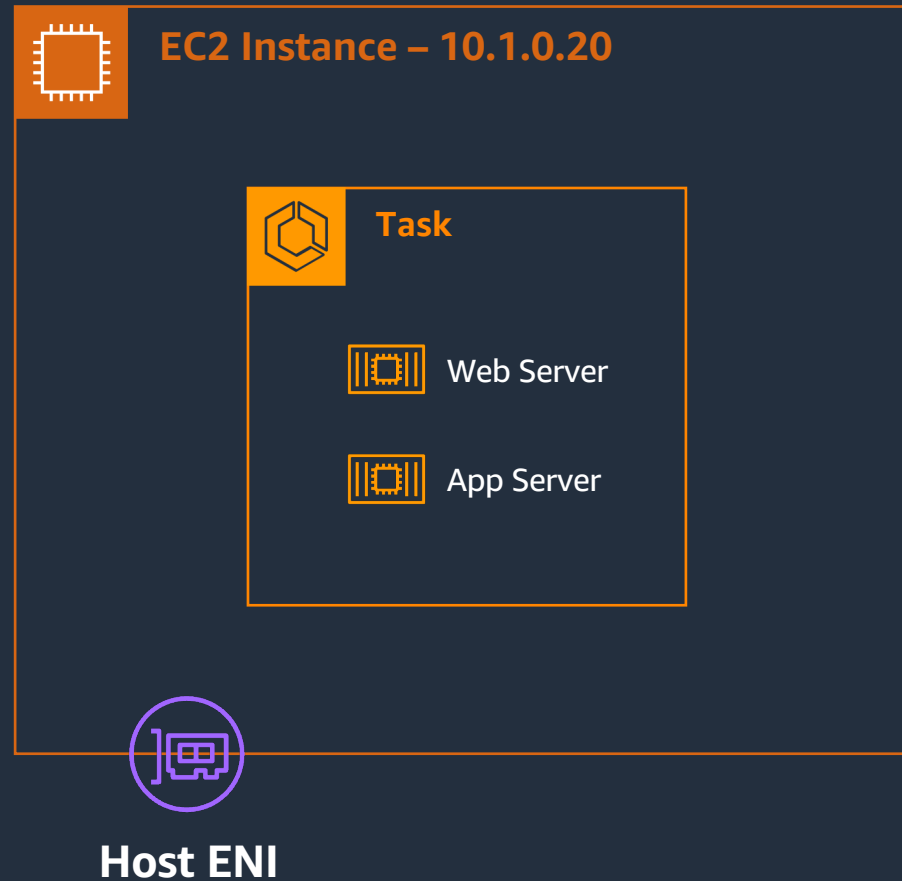
https://aws.amazon.com/ko/premiumsupport/knowledge-center/ecs-fargate-static-elastic-ip-address/?nc1=h_ls

Amazon ECS Networking



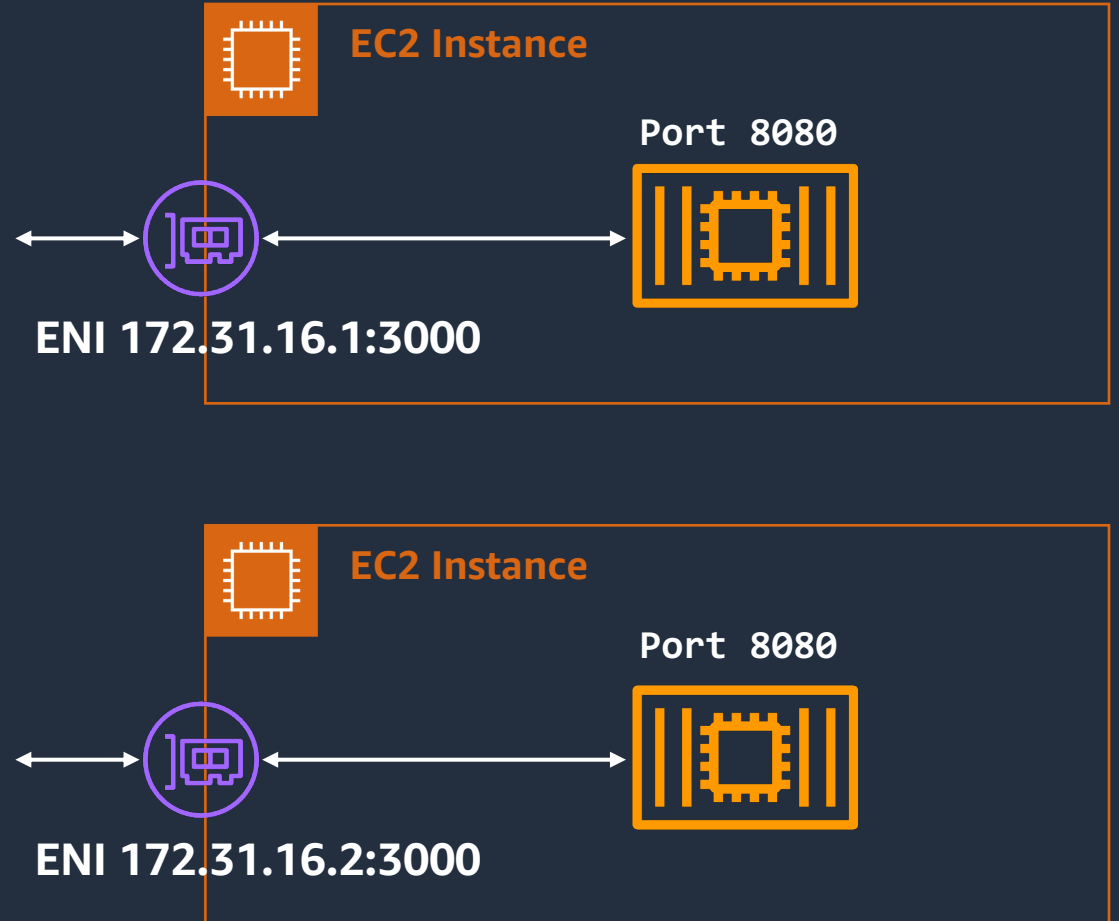
Networking - None

None으로 네트워크 모드가 설정되면
작업의 컨테이너에 외부 연결이 없으며
컨테이너 정의에서 포트 매핑을 지정할 수
없음.



Networking - Host

- TASK가 HOST의 Network Stack을 공유 가능
 - Container Ports는 EC2 Instance's network interface와 직접 매핑됨
- Task는 도커 빌트인 Virtual Network으로 바로 전달
- 1개의 EC2 Instance에 중복으로 같은 Task 실행 불가



Networking - Host

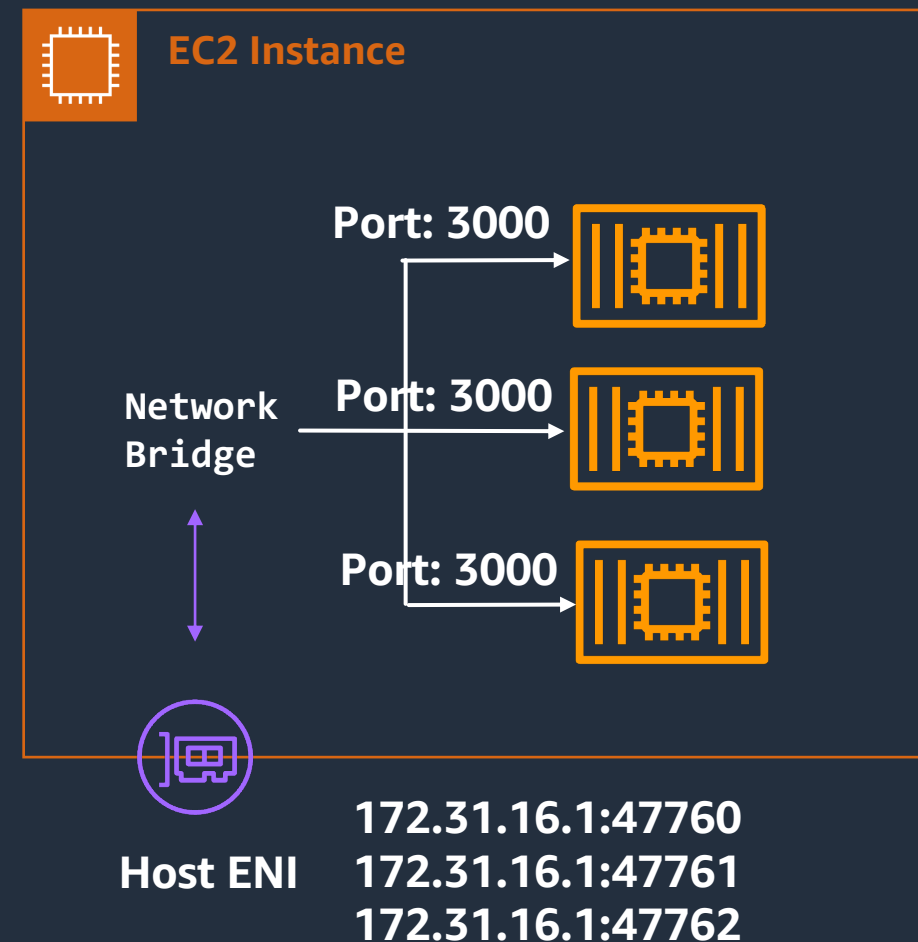
Task definition for Host networkMode

```
{
  "family": "ecs-networking-demo-bridge-dyn",
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecs-networking-exec-role",
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecs-networking-task-role",
  "containerDefinitions": [
    { "name": "nginx",
      "image": "nginx:alpine",
      "linuxParameters": {
        "initProcessEnabled": true
      },
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80
        }
      ]
    }
  ],
  "networkMode": "Host",
  "requiresCompatibilities": [
    "EC2"
  ],
  "cpu": "256",
  "memory": "512"
}
```



Networking - Bridge

- Host와 Container 간에 Virtual Network Bridge 를 이용하여 통신
 - EC2 내, Docker Built-in Virtual network 매커니즘 활용
- Port Mapping 방식: **Static** 또는 **Dynamic**
 - Static: 사용자가 지정, EC2에 같은 Task 실행 안됨
 - Dynamic: 도커가 Host traffic port를 Random으로 할당



Networking - Bridge


Task definition for **Static Mapping Bridge** networkMode

```
{
  "family": "ecs-networking-demo-bridge-dyn",
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecs-networking-exec-role",
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecs-networking-task-role",
  "containerDefinitions": [
    { "name": "nginx",
      "image": "nginx:alpine",
      "linuxParameters": {
        "initProcessEnabled": true
      },
      "portMappings": [
        {
          "hostPort": 8080,
          "protocol": "tcp",
          "containerPort": 80
        }
      ]
    }
  ],
  "networkMode": "bridge",
  "requiresCompatibilities": [
    "EC2"
  ],
  "cpu": "256",
  "memory": "512"
}
```

Networking - Bridge

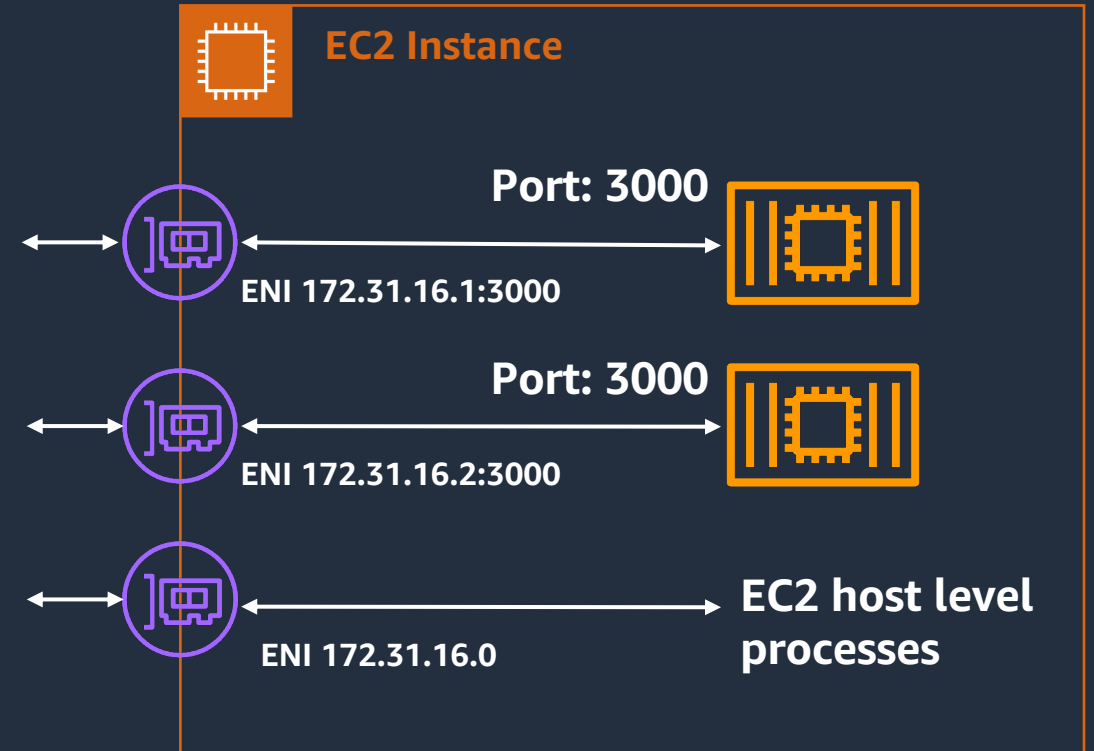
Task definition for **Dynamic** Mapping Bridge networkMode

```
{
  "family": "ecs-networking-demo-bridge-dyn",
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecs-networking-exec-role",
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecs-networking-task-role",
  "containerDefinitions": [
    { "name": "nginx",
      "image": "nginx:alpine",
      "linuxParameters": {
        "initProcessEnabled": true
      },
      "portMappings": [
        {
          "hostPort": 0,
          "protocol": "tcp",
          "containerPort": 80
        }
      ]
    }
  ],
  "networkMode": "bridge",
  "requiresCompatibilities": [
    "EC2"
  ],
  "cpu": "256",
  "memory": "512"
}
```



Networking - awsvpc

- Task 별로 elastic network interface (ENI) 할당
- Fargate는 awsvpc mode로만 동작
- Task 에 Amazon EC2 인스턴스와 동일한 네트워킹 속성을 적용할 수 있음.
 - 예: Security Group, VPC Flow logs 등
- EC2 Instance Type 별로 ENI 할당 제한, 제한 이상의 Task를 생성이 필요하면 awsvpcTrunking 모드 활성화



Networking - awsvpc

Task definition for awsvpc networkMode

```
{
  "family": "ecs-networking-demo-bridge-dyn",
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecs-networking-exec-role",
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecs-networking-task-role",
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "nginx:alpine",
      "linuxParameters": {
        "initProcessEnabled": true
      },
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80
        }
      ]
    }
  ],
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
    "EC2"
  ],
  "cpu": "256",
  "memory": "512"
}
```

ECS 개발자 도구



Developer Tools – Copilot for ECS

- 📁 Copilot은 로컬 개발 환경의 ECS에서 생산 준비가 된 컨테이너화된 애플리케이션 구축, 릴리스 및 운영을 단순화
- 📁 최신 애플리케이션 모범 사례를 지원하는 개발자 워크플로와 일치

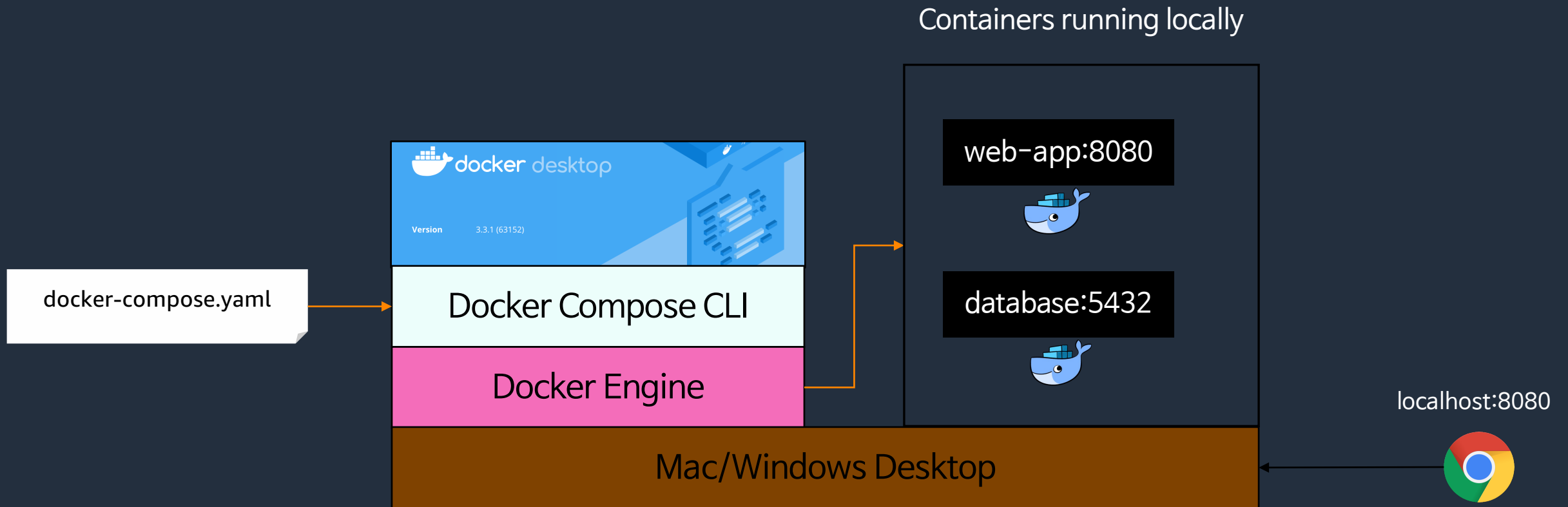
```
copilot init --app python-app \
--name pythin-svc \
--type 'Load Balanced Web Service' \
--dockerfile './Dockerfile' \
--port 80 \
--deploy
```

Developer Tools – Docker Compose for ECS

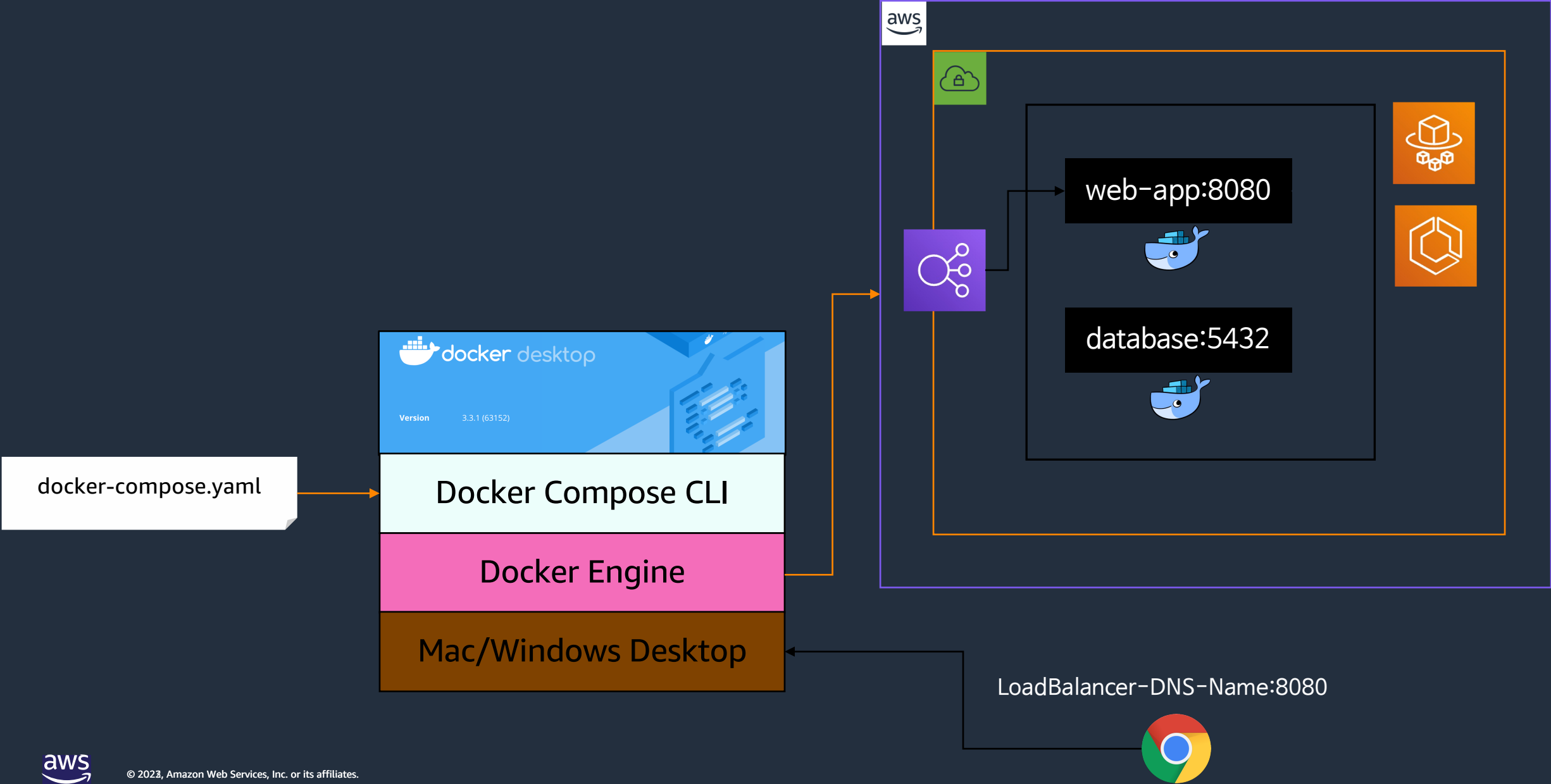
- Docker Compose를 사용하여 ECS Fargate에 애플리케이션 배포
- ECR 또는 Docker Hub에서 컨테이너 이미지 배포
- 로컬 개발 환경과 ECS 환경 간에 빠르게 전환
- ECS에 애플리케이션을 배포하기 위한 독창적인 접근 방식
- `docker compose up` 및 `docker compose down` 만큼 쉽습니다.



Developer Tools – Docker Compose for ECS



ECS Tasks running under Fargate



Take Away



Take Away

ECS 작업 정의에 적정 크기만큼 리소스를 할당

Fargate 사용 시 제약 사항 확인 (Fargate Spot – ARM X, Fargate - GPU X ...)

최소 권한 원칙에 따라 최소한의 권한 부여 (TaskExecutionRole, TaskRole)

필요에 따라 작업 배치 전략/제약 활용

워크로드에 따라 Log Driver 사용 시 awslogs – blocking(default), non-blocking, firelens 고려



Thank you!