

# Docker 실습 정리 2

Windows환경 NodeJS로 실습

# 목차

- Docker 추가 문법
- 컨테이너 통신
- 다중 컨테이너 어플리케이션
- Docker compose
- MongoDB Atlas 연결
- ECS를 ELB 연결하여 Spring app 배포

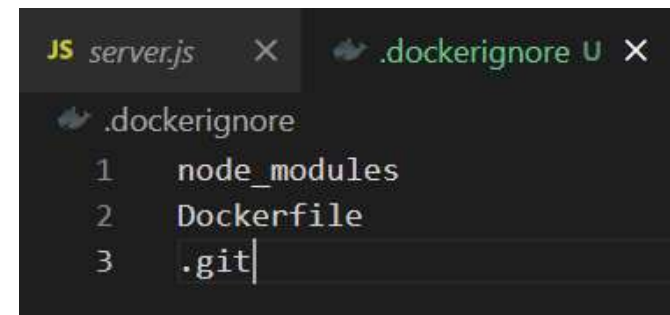
Docker 추가 문법

# "COPY" vs 바인드 마운트

- 바인드 마운트를 사용하면 로컬 호스트에 있는 폴더 또는 파일이 실시간으로 컨테이너에 자동으로 반영이 된다. 그럼에도 불구하고 "COPY"라는 문법을 사용하는 이유는 뭘까?
- 개발만 할 거면 COPY 명령어를 제거할 수 있다. 개발에 바인드 마운트를 사용하면 코드의 변경 사항이 컨테이너에 즉각 반영되므로 매우 편리하다.
- 하지만 서비스를 배포할 때는 바인드 마운트로 실행하지 않는다. 실시간으로 업데이트되는 소스 코드가 없기 때문에 굳이 바인드 마운트를 쓸 이유도 없고, 배포 환경에서는 항상 코드의 스냅샷을 가지고 싶어하기 때문.

# COPY 방지 - dockerignore

- .dockerignore 파일을 만들고 적으면 COPY 명령으로 복사해서는 안 되는 폴더와 파일을 지정할 수 있음
- 예를 들면 node\_modules는 이미지에 담으면 안 된다. Npm install에 의해 이미지에서 실행되기 때문



```
JS server.js  X  .dockerignore U  X
.dockerignore
1  node_modules
2  Dockerfile
3  .git|
```

# 환경 변수, ARG와 ENV

- 도커는 빌드 타임 인수와 런타임 환경 변수를 지원한다.
  1. ARG: 빌드할 때 결정 (docker build --build-arg)
  2. ENV: 런타임에서 결정 (docker run --env)
- 적절히 사용하면 컨테이너와 이미지에 모든 것을 하드 코딩할 필요가 없다.

# ENV 사용 방법, 장점

- 직접 80을 적는게 아니라 process.env.PORT로 대체하였다.
- ENV 를 쓰고 첫번째 인자에 설정 env명, 두번째 인자에 default 값
- 도커 파일내에 있는 ENV를 표시하려면 '\$'를 앞에 붙여줘야 한다.

```
45     });  
46   });  
47  
48   app.listen(process.env.PORT);  
49
```

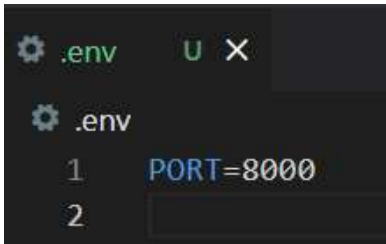
```
COPY . .  
  
ENV PORT 80  
  
EXPOSE $PORT
```

- ENV를 사용하는 장점: docker run을 할 때 --env를 사용해서 포트를 변경할 수 있다. 기존의 Default 값을 무시하고

```
PS D:\docker\docker project> docker run -d --rm -p 3000:8000 --env PORT=8000 --name feedback-app -v feedback:/app/feedback -v "D:\docker\docker project:/app/ro" -v /app/node_modules -v /app/temp feedback-node:env
```

# 또다른 ENV 사용 방법

- .env 파일을 생성하고 변수를 지정해놓고 그것을 도커 명령어를 통해 가져올 수도 있다.
- --env-file 명령어를 사용하여 파일 위치를 가져오면 환경변수 사용할 수 있음



```
PS D:\docker\docker project> docker run -d --rm -p 3000:8000 --env-file ./env --name feedback-app -v feedback:/app/feedback -v "D:\docker\docker project:/app/ro" -v /app/node_modules -v /app/temp feedback-node:env
```



# ARG 사용 방법, 장점

- ARG를 사용하면 하드코딩 된 값을 변수에서 분리시킬 수 있다.
- ARG는 오로지 도커 파일 내에서만 쓸 수 있다.
- 도커 파일 내에서 가져오는 것이므로 '\$'를 앞에 붙여줘야 한다.

```
ARG DEFAULT_PORT=80  
  
ENV PORT $DEFAULT_PORT
```

- 이렇게 ARG를 사용하면 build할 때 '--build-arg'를 통해 포트를 변경시킬 수 있다.

```
PS D:\docker\docker project> docker build -t feedback-node:dev --build-arg DEFAULT_PORT=8000 .  
[+] Building 1.9s (11/11) FINISHED
```

컨테이너 통신

# 컨테이너에서 통신하는 방법

1. HTTP로 WWW에 접근
2. 컨테이너에서 로컬 호스트 머신으로 통신
3. 컨테이너 간 통신

# 1. HTTP로 WWW에 접근

- 그냥 web 사용해서 API 가져오는 방법
- 기본적으로 컨테이너는 월드 와이드 웹에 요청을 보낼 수 있다. 특별한 설정이나 코드 필요 없음. 그냥 쓰던대로 쓰면 됨

```
app.get('/movies', async (req, res) => {  
  try {  
    const response = await axios.get('https://swapi.dev/api/films');  
    res.status(200).json({ movies: response.data });  
  } catch (error) {  
    res.status(500).json({ message: 'Something went wrong.' });  
  }  
});
```

## 2. 컨테이너에서 로컬 호스트로 통신

- 로컬 호스트에 직접 통신한다.
- 로컬 호스트 머신과 통신하기 위해서는 별도의 설정이 필요하다.  
localhost를 도커가 이해할 수 있도록 바꿔야 한다.  
**"host.docker.internal"**
- 이 특별 도메인 주소를 사용하면 도커 이미지에서 로컬 호스트 머신의 IP주소로 변환된다.

```
mongoose.connect(  
  'mongodb://localhost:27017/swfavorites',  
  { useNewUrlParser: true },  
  (err) => {  
    if (err) {  
      console.log(err);  
    } else {  
      app.listen(3000);  
    }  
  }  
);
```



```
mongoose.connect(  
  'mongodb://host.docker.internal:27017/swfavorites',  
  { useNewUrlParser: true },  
  (err) => {  
    if (err) {  
      console.log(err);  
    } else {  
      app.listen(3000);  
    }  
  }  
);
```

# 컨테이너 간 통신 준비 - 추가 컨테이너

- 컨테이너 간 통신을 실습하기 위해서 두 개의 컨테이너가 필요하다. DB 역할을 하는 컨테이너 하나를 더 만든다.
- 로컬 호스트에 MongoDB를 설치하지 않고도 컨테이너 내부에서 MongoDB를 설정할 수 있다.
- 별도의 Dockerfile을 만들지 않아도 된다. 공식 이미지가 도커 허브에 있다. 그냥 pull 하기만 하면 된다
- 혹은 그냥 run 하면 자동으로 pull 되고, 내려 받은 이미지를 토대로 컨테이너를 run 한다.
- 이 컨테이너가 MongoDB를 가동하게 된다.(그래서 설치 필요 없음)



mongo DOCKER OFFICIAL IMAGE · 1B+ · 9.6K

Updated 9 days ago

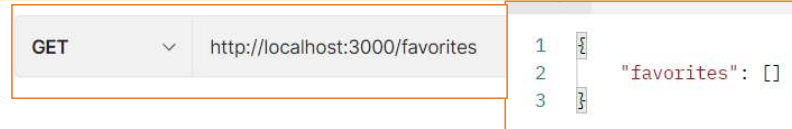
MongoDB document databases provide high availability and easy scalability.

Linux Windows x86-64 ARM 64 IBM Z

```
PS D:\docker practice\docker project> docker run mongo
Unable to find image 'mongo:latest' locally
latest: Pulling from library/mongo
```

### 3. 컨테이너 간 통신

- 서로 다른 컨테이너끼리 통신하는 방법.
- "docker container inspect '이름'" 명령어를 사용하면 현재 컨테이너의 상태를 볼 수 있다.
- 그 상태 중에서 IPAddress가 있는데 이 주소를 사용하면 해당 컨테이너에 연결할 수 있다.
- 연결 주소로 IPAddress를 사용하여 연결



```
PS D:\docker practice\docker project> docker container inspect mongodb
```

```
"Gateway": "172.17.0.1",
"IPAddress": "172.17.0.2",
"IPPrefixLen": 16,
```

```
mongoose.connect(
  'mongodb://host.docker.internal:27017/swfavorites',
  { useNewUrlParser: true },
  (err) => {
    if (err) {
      console.log(err);
    } else {
      app.listen(3000);
    }
  }
);
```



```
mongoose.connect(
  'mongodb://172.17.0.2:27017/swfavorites',
  { useNewUrlParser: true },
  (err) => {
    if (err) {
      console.log(err);
    } else {
      app.listen(3000);
    }
  }
);
```

	<b>mongodb</b> 7fdcdb371d47
	<b>favorites</b> 3985ab1e3f6d

두 컨테이너  
재생

# Docker networks

```
'mongodb://host.docker.internal:27017/swfavorites',
```

```
'mongodb://172.17.0.2:27017/swfavorites',
```

- 이렇게 주소를 하드코딩하는 것은 좋지 못하다.
- 명령어 "docker run --network '네트워크 이름'"을 사용하면 모든 컨테이너를 하나의 동일한 네트워크에 넣을 수 있다.
- 하지만 볼륨과는 다르게 네트워크는 도커가 자동으로 생성하지 않는다. 미리 네트워크를 생성해야 한다.
- 네트워크 생성 명령어 "docker network create favorite-net"

```
PS D:\docker practice\docker project> docker network create favorite-net
9b8661acc92987dd7af013eababa87abff187ea3dba96177d7f6c135a60e0db0
```

- 네트워크 이용하여 컨테이너 돌리기

```
PS D:\docker practice\docker project> docker run -d --rm --name mongodb --network favorite-net mongo
168cbc9f974e2df852d35aed924ea3379754bf255e7bf524756a028703d1d8ae
```



# Docker network 사용하기

- 동일한 네트워크를 사용하면 컨테이너 이름을 URL주소를 대신하여 통신할 수 있다.

```
'mongodb://172.17.0.2:27017/swfavorites',
```



```
'mongodb://mongodb:27017/swfavorites',
```

- 동일한 네트워크 지정해서 돌린다.



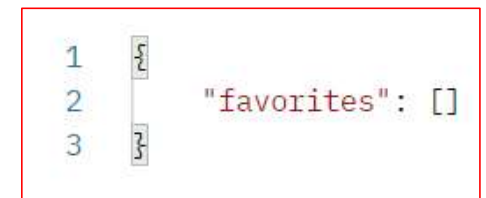
**mongodb**

168cbc9f974e

```
docker run -d --rm --name mongodb --network favorite-net mongo
```

```
docker run -d --rm --name favorites --network favorite-net -p 3000:3000 favorites-node  
5abc4057b0e22ee7570e2d5ee5
```

- Mongodb를 컨테이너로 돌릴 때는 포트지정이 필요하다. 포트 지정은 해당 네트워크 외부에서 그 컨테이너에 연결할 때만 필요하다.



다중 컨테이너 어플리케이션

# 다중 컨테이너 쓰는 이유, 셋팅 환경

- 각각의 컨테이너는 한 가지 역할에만 집중하도록 하는 것이 좋다.
- 예를 들면 웹 API 용 컨테이너 하나와 데이터베이스용 컨테이너 하나로 나눌 수 있다.
- 실습은 세가지 컨테이너를 만드는 것을 목표 (DB, Backend, Frontend)



# 1. DB 이미지, 컨테이너 만들기

- MongoDB는 이미 이미지가 도커 허브에 있다. 그냥 run mongo 하면 된다.

```
PS D:\docker practice\docker project> docker run --name mongodb --rm -d -p 27017:27017 mongo
7b553e28cc9a1033c831ed0e2abad1f64f4283218a447538f396e82c282369e9
```

- MongoDB 컨테이너에 포트를 지정하는 이유: 아직 도커화되지 않은 로컬 실행 백엔드와 통신하게 할 수 있다.
- 백엔드 코드는 localhost의 포트 27017번을 보고 있다.

```
mongoose.connect(
  'mongodb://localhost:27017/course-goals',
  {
```

## 2. 백엔드 이미지 컨테이너 만들기

- 백엔드 컨테이너가 DB 컨테이너와 연결하기 위해서는 localhost로 통신하는 게 아니라 host.docker.internal을 사용해야 한다.

```
mongoose.connect(  
  'mongodb://localhost:27017/course-goals',  
  {  
    // ...  
  }  
)
```

```
mongoose.connect(  
  'mongodb://host.docker.internal:27017/course-goals',  
  {  
    // ...  
  }  
)
```

- DB컨테이너와 마찬가지로 포트를 지정해준다. 이렇게 포트를 지정하면 로컬호스트로 접근할 수 있어서 프론트엔드가 해당 포트의 정보를 읽어올 수 있다.

```
PS D:\docker practice\docker project\backend> docker build -t goals-node .  
[+] Building 10.0s (11/11) FINISHED
```

```
PS D:\docker practice\docker project\backend> docker run --name goals-backend --rm -d -p 80:80 goals-node  
d76da8823c1595cacffc998658b7ced0967acf6d25da796a35c7391e45eff75d
```

```
FROM node  
  
WORKDIR /app  
  
COPY package.json .  
  
RUN npm install  
  
COPY . .  
  
EXPOSE 80  
  
CMD ["node", "app.js"]
```

### 3. 프론트엔드 이미지 컨테이너 만들기

- /app이라는 똑같은 폴더 이름을 사용해도 백엔드와 완전히 다른 컨테이너이기 때문에 충돌이 발생하진 않는다.
- Run을 실행할 때 "-it" 인터랙티브 모드를 활성화해준다. React가 현재 개발모드로 되어있기 때문에 개발모드로 실행해줘야 함

```
PS D:\docker practice\docker project\frontend> docker build -t goals-react .  
[+] Building 136.6s (11/11) FINISHED
```

```
FROM node:14  
WORKDIR /app  
COPY package.json .  
RUN npm install  
COPY . .  
EXPOSE 3000  
CMD ["npm", "start"]
```

```
PS D:\docker practice\docker project\frontend> docker run --name goals-frontend --rm -p 3000:3000 -it goals-react
```

Compiled successfully!

You can now view **frontend** in the browser.

Local: http://localhost:3000  
On Your Network: http://172.17.0.4:3000

Note that the development build is not optimized.  
To create a production build, use `npm run build`.

New Goal

Add Goal

Docker practice go!

## 4. 컨테이너들을 docker network로 연결

- 네트워크 'goals-net'을 만든다.

```
PS D:\docker practice\docker project> docker network create goals-net  
1e652b30f44a7c6747ab9de4991eb4ebd417027ba8e6ef45457099f5e3c8c6af
```

- 더 이상 로컬 호스트 연결이 필요 없으므로 포트 설정은 하지 않고 run
- 로컬 주소를 모두 새로 만든 컨테이너 이름으로 바꾸고 이미지 빌드

```
PS D:\docker practice\docker project> docker run --name mongodb --rm -d --network goals-net mongo  
bccb4c13853bda0875fe850fc9a0e67aba87ceb90a67fee27858789cbb09d548
```

```
mongoose.connect(  
  'mongodb://host.docker.internal:27017/course-goals',  
  {  
    // ...  
  }  
);
```


```
mongoose.connect(  
  'mongodb://mongodb:27017/course-goals',  
  {  
    // ...  
  }  
);
```



```
PS D:\docker practice\docker project\backend> docker run --name goals-backend --rm -d --network goals-net goals-node  
34d777720d3fec8eb707726889de6fdabebad7446b2c7a081035e21b5f1702d4
```

```
try {  
  const response = await fetch('http://localhost/goals');  
  // ...  
}
```

```
try {  
  const response = await fetch('http://goals-backend/goals');  
  // ...  
}
```



```
PS D:\docker practice\docker project\frontend> docker run --name goals-frontend --network goals-net --rm -p 3000:  
3000 -it goals-react
```

# 오류가 나는 이유

- 앞서 설정한대로 컨테이너들을 돌려주면 에러가 발생한다.
- React는 백엔드와는 다르게 동작한다. 개발 서버를 시작하여 애플리케이션을 제공하는데,
- React 코드는 컨테이너 내부에서 실행되지 않고 브라우저에서 실행된다. 그리고 브라우저는 goals-backend가 무엇인지 모른다.

```
try {  
  const response = await fetch('http://goals-backend/goals');
```

Something went wrong!

Failed to fetch

New Goal

Add Goal

No goals found. Start adding some!



# React에서 컨테이너와 통신하는 방법

```
try {  
  const response = await fetch('http://localhost/goals');
```

- 이 코드는 브라우저에서 실행된다. (React의 특징)
- 컨테이너 이름이 아닌 localhost로 다시 바꿔줘야 한다.
- 이미지를 다시 빌드하고 컨테이너를 실행할 때 네트워크 셋팅 코드는 지우고 돌린다. 필요가 없는 옵션이기 때문. 개발서버는 네트워크를 신경 쓰지 않는다.

```
PS D:\docker practice\docker project\frontend> docker run --name goals-frontend --rm -p 3000:3000 -it goals-react
```

# React에서 통신하기 위해 백엔드 포트 설정

- 백엔드 컨테이너를 실행할 때 포트를 살려줘야 한다. 그래야 프론트엔드와 통신할 수 있다.

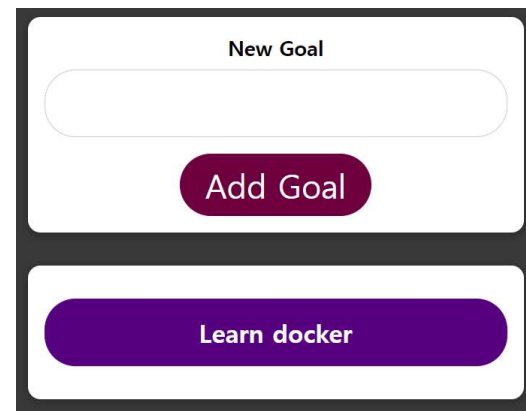
백엔드 돌리기

```
PS D:\docker practice\docker project\frontend> docker run --name goals-backend --rm -d --network goals-net -p 80:80 goals-node 7ca8c398af0ee61230d80816289a4021ee2c3d84118f21320a0ae320f0d3dbfd
```

프론트엔드 돌리기

```
PS D:\docker practice\docker project\frontend> docker run --name goals-frontend --rm -p 3000:3000 -it goals-react
```

정상 작동 확인

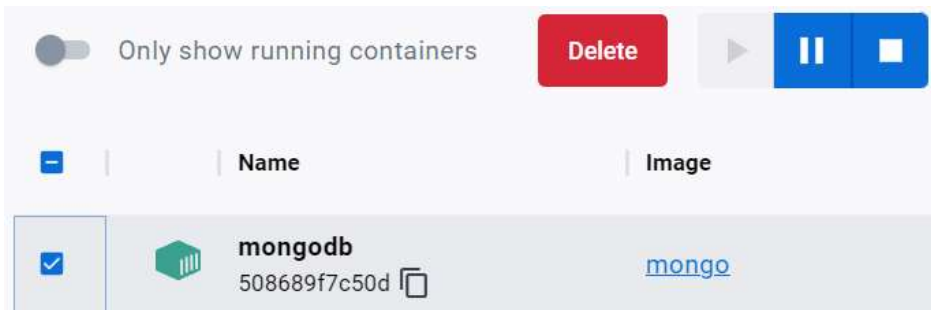


The screenshot shows a web application interface with a dark gray border. It contains two main sections. The top section, titled 'New Goal', has a white background and includes a rounded rectangular input field and a dark red 'Add Goal' button. The bottom section has a white background and features a large purple button labeled 'Learn docker'.

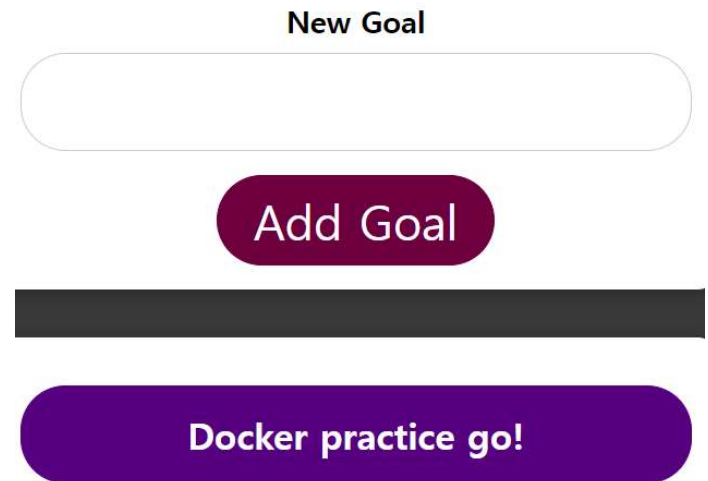
# 명명 볼륨을 사용하여 DB 데이터 지속

- 컨테이너가 종료되어도 DB 데이터는 유지되어야 한다.
- MongoDB 데이터 유지를 위해 명명 볼륨을 사용한다.

```
PS D:\docker practice\docker project> docker run -d --rm --name mongodb -v data:/data/db --network goals-net mongo 508689f7c50de4e5f9a7ff1fbb1d820df9b294cf2b822f549b3197ccabab8b5d
```



- 꺾다가 다시 켜도 데이터가 사라지지 않는다.



# 백엔드 컨테이너 볼륨, 바인딩 마운트

- 로그를 남기기 위해 명명 볼륨을 사용하여 로그 기록
- 소스 코드 변경을 실시간으로 반영해주기 위해 바인딩 마운트 사용.
- Node\_module은 바인딩 마운트 예외로 두기 위해 볼륨 지정

```
PS D:\docker practice\docker project> docker run --name goals-backend -v "D:/docker practice/docker project/backend:/app" -v logs:/app/logs -v /app/node_modules --rm -d --network goals-net -p 80:80 goals-node
787e34d463bc7ab9ab294b4c7f330383fba118f23eb2fb02e61ca6a469b5d6fa
```

# 프론트엔드 컨테이너 바인딩 마운트

- 바인딩 마운트로 소스 코드 변경 실시간 반영
- 근데 이건 윈도우 시스템에서는 반영이 안 된다. 이 문제를 해결하려면 리눅스 시스템에서 프로젝트를 만들고 도커를 사용해야함.

```
PS D:\docker practice\docker project> docker run -v "D:\docker practice\docker project\frontend\src:/app/src" --name goals-frontend -rm -p 3000:3000 -it goals-react
```

Docker compose

# Docker compose

- Docker run에는 `-d -rm -p -v -network` 등 여러 설정이 필요하여 명령어가 길어지게 된다.
- Docker compos는 명령어 `docker build`나 `run`을 대신 해주는 역할을 한다. 다중 컨테이너 설정을 쉽게 할 수 있게 해준다.
- 도커 컴포즈는 `dockerfile`을 대체하지 않는다. 함께 사용해야 한다.
- 주의할 점은 도커 컴포즈는 다수의 호스트에서 분배되어 동작하는 다중 컨테이너를 관리하는 데에는 적합하지 않다. 하나의 호스트에서 다중 컨테이너를 가동시킬 때 사용하는 것이 좋다.
- 도커 컴포즈에서 서비스는 컨테이너를 의미한다.

# Docker compose 파일 만들기

- Dockerfile과 별개로 compose 파일을 만들어야 한다. 확장자는 .yaml
- Yaml은 들여쓰기를 사용하여 종속성을 표현한다
- Version은 도커 컴포즈의 사양을 의미한다.
- Services 하위 요소에 컨테이너 이름을 표기한다.

```
version: "3.8"
services:
  mongodb:
  backend:
  frontend:
```



# Docker compose 컨테이너 하위요소

- Image – 컨테이너가 사용할 이미지 이름을 넣는다.
- Volumes – 컨테이너가 사용할 볼륨 정보를 넣는다.
- Network – 이걸 필요하지 않음. 도커 컴포즈를 사용하면 자동으로 기본 네트워크를 만들고 모든 서비스를 자동으로 그 네트워크에 추가한다. 물론 특정 네트워크 이름으로 지정해도 됨.
- 명명 볼륨은 따로 명시해줘야한다. services와 같은 레벨에 작성해야 한다. 익명 볼륨, 바인드 마운트는 지정 필요 없음

```
version: "3.8"
services:
  mongodb:
    image: "mongo"
    volumes:
      - data:/data/db
    network:
volumes:
  data:
```

# Docker compose 컨테이너 하위요소


- Build – 이미지가 없을 때에는 이미지를 빌드하기 위해 이 명령어를 사용한다. Dockerfile이 있는 위치를 작성해주면 된다.
- Port – 컨테이너에 연결할 포트 설정
- Depends on – 해당 컨테이너가 의존하는 컨테이너 이름 넣기
- 인터랙티브 모드를 활성화하기 위해 다음 두 가지 설정
  1. Stdin\_open: true – 개방형 입력 터미널 열기
  2. tty: true – 터미널에 연결

```
backend:
  build: ./backend
  ports:
    - '80:80'
  networks:
  volumes:
    - logs:/app/logs
    - ./backend:/app
    - /app/node_modules
  depends_on:
    - mongodb
frontend:
  build: ./frontend
  ports:
    - '3000:3000'
  volumes:
    - ./frontend/src:/app/src
  stdin_open: true
  tty: true
  depends_on:
    - backend
```

# Docker compose 서비스 시작 및 중지

- 컴포즈 파일이 있는 디렉토리에서 Docker-compose up을 실행
- 네트워크, 볼륨, 컨테이너를 모두 자동으로 생성해준다.

```
docker-compose up -d
```



```
[+] Running 6/6
- Network dockerproject_default      Created
- Volume "dockerproject_data"        Created
- Volume "dockerproject_logs"        Created
- Container dockerproject-mongodb-1  Started
- Container dockerproject-backend-1  Started
- Container dockerproject-frontend-1 Started
```

- Detach로 돌아가는 docker compose를 종료하려면 docker down
- 볼륨도 삭제하려면 down 뒤에 -v도 추가해준다.

```
PS D:\docker practice\docker project> docker-compose down
[+] Running 2/2
- Container dockerproject-mongodb-1 Removed
- Network dockerproject_default      Removed
```

```
PS D:\docker practice\docker project> docker-compose down -v
[+] Running 1/0
- Volume dockerproject_data Removed
```

# Docker Compose build 관련 명령어

- Docker compose up을 했을 때, 기존에 이미지가 있으면 이것을 재 활용한다. 이를 방지하기 위해서는 아래 두 가지를 사용하면 됨
- Docker compose build: 이미지 빌드를 수행하는 문법
- Docker compose up --build: 이미지 재빌드를 강제하는 문법. 기존의 이미지를 다시 사용하지 않고 무조건 다시 빌드한다.

다중 컨테이너 배포

# AWS ECS에서 Docker compose

- Docker Compose는 로컬 호스트에서 다중 컨테이너를 실행하는데 유용하지만 배포를 할 경우 Compose를 사용하는 것은 좋지 못함.
- 각각의 클라우드 서비스에서 필요한 별도의 설정이 필요함.
- AWS ECS의 경우 등록된 이미지를 하나씩 끌어와서 컨테이너를 동작 시키기 때문에 Docker Compose를 사용하지 말고 하나씩 개별적으로 배포한다.

# AWS ECS에서 컨테이너 간 네트워크

- AWS ECS를 사용하면 아래와 같은 IP 설정은 사용하지 못 한다. 컨테이너 인스턴스는 클라우드에 의해 실행되는데, 항상 동일한 머신에서 실행되는 것이 아니다.

```
WORD}@mongodb:27017/course-goal
```

- 그렇지만 예외는 존재한다. AWS ECS에서 동일한 Task에 컨테이너들을 넣으면 동일한 머신에서의 실행이 보장된다.
- 이 경우, AWS ECS는 도커 네트워크를 생성하지 않는다. 로컬 호스트를 컨테이너 어플리케이션 코드 내부 주소로 사용할 수 있게 해준다.

환경 변수로 교체

```
WORD}@${process.env.MONGODB_URL}:27017/
```

```
1 MONGODB_USERNAME=max
2 MONGODB_PASSWORD=secret
3 MONGODB_URL=mongodb|
```

# AWS ECS 클러스터 생성

- 클러스터 생성을 누르고 네트워크 전용으로 설정
- 네트워킹에서 VPC 생성에 체크하면 해당 클러스터를 프라이빗으로 설정한다.
- 클러스터: 컨테이너 주변 네트워크

## 네트워킹

클러스터에 사용할 새 VPC를 생성합니다. VPC는 AWS 클라우드의 격리된 부분으로서, Fargate 작업과 같은 AWS 객체로 이루어집니다.

VPC 생성 ☒ 해당 클러스터에 새 VPC 생성

CIDR 블록  ⓘ

서브넷 1  ⓘ

서브넷 2  ⓘ

[+ 서브넷 추가](#)



# AWS ECS Task definition 생성

- 서비스는 태스크를 기반으로 하므로 먼저 태스크를 만들어야 한다.
- 태스크를 만들기 위해서 태스크 정의가 필요하다.

새 작업 정의 생성

시작 유형 호환성 선택

작업을 시작할 위치에 따라 작업 정의를 호환할 시작 유형을 선택하십시오



- Fargate: 컨테이너가 무한대로 확장되는 서버리스 컨테이너 실행 환경

태스크 정의 이름\*

goals



호환성 요구 사항\*

FARGATE

태스크 역할

ecsTaskExecutionRole



인증된 AWS 서비스에 API 요청을 할 때 작업이 사용할 수 있는 IAM 역할 옵션입니다. [IAM 콘솔](#)에서

작업 메모리(GB)

0.5GB



0.25vCPU에 대한 유효한 메모리 범위: 0.5GB - 2GB.

작업 CPU(vCPU)

0.25 vCPU



0.5GB 메모리에 대한 유효한 CPU: 0.25 vCPU

# AWS ECS Task definition – backend 컨테이너

- 태스크에 컨테이너를 추가

컨테이너 추가

컨테이너 이름\*

goals-node

이미지\*

millwheel/goals-node

포트 매핑

컨테이너 포트

80

tcp

프로토콜

- 환경-명령에서 node,app.js를 추가한다. 배포 환경에서는 nodemon이 필요 없음. node app.js 명령어를 실행하여 돌리도록 한다.

명령

node,app.js

```
"scripts": {  
  "test": "echo \"Error: no  
  \"start\": \"nodemon app.js\"  
},
```

```
CMD [\"npm\", \"start\"]
```

# AWS ECS Task definition – backend 컨테이너

- 환경 변수를 수정해야 한다.
- 개발단계에서 URL 이름은 mongodb였다. 도커 네트워크에 의해 컨테이너 이름으로 통신할 수 있었음
- AWS ECS는 이런 기능을 제공하지 않으므로 localhost를 사용해야함
- 동일한 Task 내에서는 모든 컨테이너가 동일한 호스트에서 작동한다고 보면 됨

```
0}@${process.env.MONGODB_URL}:27017/c
```

```
1 MONGODB_USERNAME=max
2 MONGODB_PASSWORD=secret
3 MONGODB_URL=mongodb
```

## 환경 변수

'valueFrom' 필드를 사용하여 AWS Systems Manager 파라미터 저장소 키 또는 ARN을 지정할 수도 있습니다. 간으로 컨테이너에 주입합니다.

### Key

MONGODB_USERNAME	값	max
MONGODB_PASSWORD	값	secret
MONGODB_URL	값	localhost

# AWS ECS Task definition – backend 컨테이너

- 스토리지 로깅은 설정할 필요 없다.
- 배포에서는 바인딩 마운트를 쓸 필요가 없으므로 볼륨을 스토리지 로깅에 넣을 이유가 없음

## 스토리지 및 로깅

읽기 전용 루트 파일 시스템 ☐

탑재 지점

소스 볼륨

<none> ▼

컨테이너

경로

읽기 전용 ☐

# AWS ECS Task definition – mongoDB 컨테이너

- Task에 이번엔 MongoDB 컨테이너를 추가한다.

컨테이너 이름\*

mongodb

이미지\*

mongo

환경 변수

valueFrom\* 필드를 사용하여 AWS Systems Manager 파라미터 저장소 키 또는 ARN; 간으로 컨테이너에 주입합니다.

key

MONGO\_INITDB\_ROOT\_USERNAME

값

max

MONGO\_INITDB\_ROOT\_PASSWORD

값

secret

# AWS ECS 서비스 생성

- Task definition에서 만들었던 것을 가져와서 선택하고 VPC 할당

시작 유형 ☒ FARGATE

AWS Fargate는 현재 Amazon ECS

로드 밸런서 유형\* ☐ 없음  
해당 서비스에서는 로드 밸런서를 사용할 수 없습니다.

☒ Application Load Balancer  
컨테이너가 동적 호스트 포트 매핑을 허용할 때 (여러 서비스가 규칙 기반

작업 정의 패밀리

goals

개정

1 (latest)

클러스터 VPC\* vpc-02a94a46fb5b11175 (10.0.0.0/... ▼

서브넷\*

subnet-00caf7da821d78295 (10.0.1.0/24) | goals-app/Public - ap-northeast-2b  
생성 시 ipv6 할당: 비활성화됨

subnet-07e6303fc2d1886ab (10.0.0.0/24) | goals-app/Public - ap-northeast-2a  
생성 시 ipv6 할당: 비활성화됨

- ALB는 커스텀 도메인을 할당할 수 있다.
- 계속 진행하려면 로드밸런서를 생성해야 한다.

 로드 밸런서를 찾을 수 없음  
EC2 콘솔에서 로드 밸런서를 생성하십시오.

# AWS 로드밸런서 생성

- ALB를 선택하고 이름 넣고, 인터넷 facing으로 선택한다.
- 로드밸런서는 포트 80을 노출하므로 포트 리스너 체크
- 서비스 생성에서 선택한 VPC와 동일한 VPC에 연결해야 한다. 가용영역 모두 체크

## 1단계: Load Balancer 구성

### 기본 구성

Load Balancer를 구성하려면 이름을 입력하고, 체계를 선택하고, 하나 이상의 리스너를 지정하고, 수신하는 리스너입니다.

이름 ⓘ ecs-lb

체계 ⓘ  
☒ 인터넷 경계  
☐ 내부

IP 주소 유형 ⓘ ipv4

### VPC ⓘ

vpc-02a94a46fb5b11175 (10.0.0.0/16)

### 가용 영역

☒ ap-northeast-2a subnet-07e6303fc2d1886ab (goals-app/Public) IPv4 주소 ⓘ AWS에서 할당

☒ ap-northeast-2b subnet-00caf7da821d78295 (goals-app/Public) IPv4 주소 ⓘ AWS에서 할당

# AWS 로드밸런서 생성 설정

- 라우팅 설정에서 target type을 ip로 지정한다. Fargate를 사용하기 위해서 ip 방식을 선택해야 한다.

대상 그룹	 새 대상 그룹
이름	 tg
대상 유형	<input type="radio"/> 인스턴스 <input checked="" type="radio"/> IP <input type="radio"/> Lambda 함수
프로토콜	 HTTP
포트	 80
로토콜 버전	 <input checked="" type="radio"/> HTTP1 HTTP/1.1을 사용하여 대상으로 요청을 전송합니다



# AWS ECS 서비스 생성 - 로드 밸런서 연결

- 로드 밸런서 지정하고 백엔드 컨테이너를 로드밸런서에 추가한다.

로드 밸런서 이름

로드 밸런싱할 컨테이너

컨테이너 이름: 포트

로드 밸런서에 추가

프로덕션 리스너 포트\*

프로덕션 리스너 프로토콜\* HTTP

대상 그룹 이름

대상 그룹 프로토콜 HTTP

- 서비스 생성 누름

이전

서비스 생성

# [ERROR] AWS ECS 작업 중단

- 작업이 돌아가다가 중단 된다.
- 중단 사유는 "Essential container in task exited"

	이름	컨테이너 런타임 ID	상태	이미지
▶	mongodb	93aab0ac20ca409e83...	STOPPED	mongo
▶	goals-node	93aab0ac20ca409e83...	STOPPED	millwheel/goals-node

마지막 상태 DEPROVISIONING

원하는 상태 STOPPED

중단 사유 Essential container in task exited

- 로그를 살펴보니 MongoDB Auth failed

▶	2023-03-28 10:10:53	MongoError: Authentication failed.
▶	2023-03-28 10:10:53	at MessageStream.messageHandler (/app/node_modules/mongodb/lib/cmap/connection.js:299:20)
▶	2023-03-28 10:10:53	at MessageStream.emit (events.js:400:28)




MongoDB Atlas로 배포

# MongoDB Atlas로 전환


- 앞서 MongoDB를 컨테이너화 해서 구동했지만, 이 모든 DB 관리를 개인이 관리하는 것은 어려운 일이다.
- DB 자체적으로 전문적으로 다루는 지식이 필요함
- DB를 좀 더 효율적으로 관리하기 위해서 DB를 자체적으로 컨테이너로 만들지 말고 클라우드 서비스의 도움을 받는다. MongoDB Atlas가 데이터 관리를 알아서 하도록 해준다.
- MongoDB Atlas는 Scaling과 performance 그리고 Backup과 security를 알아서 처리해준다.

# 클러스터 만들기

**Provider**



**Region** ★ Recommended region i

 **Seoul (ap-northeast-2) ★** ▼

**Name**  
You cannot change the name once the cluster is created.

Cluster0

# 클러스터에 연결

## Database Deployments

Find a database deployment...

Cluster0

Connect

View Monitoring

Browse Collections

...



Connect with the MongoDB Shell

Interact with your cluster using MongoDB's interactive Javascript interface



Connect your application

Connect your application to your cluster using MongoDB's native drivers



Connect using MongoDB Compass

Explore, modify, and visualize your data with MongoDB's GUI



### 2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://poland:<password>@cluster0.xnfdgy.mongodb.net/?
retryWrites=true&w=majority
```

- 제공되는 연결 코드를 살펴보면 이전에 MongoDB에 connect 할 때 형식과 매우 유사함

# Atlas 사용에 주의할 점

Atlas에서의 MongoDB 버전

VERSION	REGION
5.0.15	AWS Seoul (ap-northeast-2)

- 버전을 꼭 체크해야 한다.
- 개발할 때 버전과 배포할 때 버전이 다르면 안 된다.

Docker hub에서 제공되는 MongoDB 버전

## Simple Tags

- 6.0.5-jammy, 6.0-jammy, 6-jammy, jammy
- 6.0.5-windowsservercore-ltsc2022, 6.0-windowsservercore-ltsc2022, 6-windowsservercore-ltsc2022, win
- 6.0.5-windowsservercore-1809, 6.0-windowsservercore-1809, 6-windowsservercore-1809, windowsserverco
- 6.0.5-nanoserver-ltsc2022, 6.0-nanoserver-ltsc2022, 6-nanoserver-ltsc2022, nanoserver-ltsc2022
- 6.0.5-nanoserver-1809, 6.0-nanoserver-1809, 6-nanoserver-1809, nanoserver-1809
- 5.0.15-focal, 5.0-focal, 5-focal
- 5.0.15-windowsservercore-ltsc2022, 5.0-windowsservercore-ltsc2022, 5-windowsservercore-ltsc2022
- 5.0.15-windowsservercore-1809, 5.0-windowsservercore-1809, 5-windowsservercore-1809
- 5.0.15-nanoserver-ltsc2022, 5.0-nanoserver-ltsc2022, 5-nanoserver-ltsc2022
- 5.0.15-nanoserver-1809, 5.0-nanoserver-1809, 5-nanoserver-1809

# 개발과 배포에서 똑같이 atlas 이용

## 2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://poland:<password>@cluster0.xnfdgy.mongodb.net/?  
retryWrites=true&w=majority
```

- 개발과 배포에서 항상 Atlas를 사용하도록 환경변수를 사용해서 채워 넣는다.
- 포트는 제거한다.
- 개발과 배포에서 서로 다른 데이터 베이스를 사용할 수 있게 환경변수를 사용한다.

```
mongoose.connect(  
  `mongodb+srv://${process.env.MONGODB_USERNAME}:${process.env.MONGODB_PASSWORD}@${process.env.MONGODB_URL}/${process.env.  
  MONGODB_NAME}?retryWrites=true&w=majority`,  
  {  
    useNewUrlParser: true,  
    useUnifiedTopology: true,  
  })
```



# 백엔드 환경변수 채워 넣기

- Atlas에서 제공한 형식에 알맞게 환경변수를 매칭시켜줘야한다.

## 2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://poland:<password>@cluster0.xnfdfigy.mongodb.net/?  
retryWrites=true&w=majority
```

```
mongoose.connect(  
  `mongodb+srv://${process.env.MONGODB_USERNAME}:${process.env.MONGODB_PASSWORD}@${process.env.MONGODB_URL}/${process.env.  
  MONGODB_NAME}?retryWrites=true&w=majority`,  
  {  
    useNewUrlParser: true,  
    useUnifiedTopology: true,  
  })
```

```
1 MONGODB_USERNAME=poland  
2 MONGODB_PASSWORD=zpyVV1QlHeDkIm1V  
3 MONGODB_URL=cluster0.xnfdfigy.mongodb.net  
4 MONGODB_NAME=goals-dev
```

# 로컬 호스트에서 테스트

- 그대로 로컬 개발 환경에서 돌려도 작동된다.

```
PS D:\docker practice\docker project> docker-compose up  
[+] Running 1/0
```



# Docker image 재 빌드

- 이미지를 빌드해서 docker hub에 올려준다.

```
PS D:\docker practice\docker project> docker build -t millwheel/goals-node ./backend  
[+] Building 2.3s (11/11) FINISHED
```

```
PS D:\docker practice\docker project> docker push millwheel/goals-node  
Using default tag: latest
```

# AWS ECS 작업 정의 업데이트

- 배포 환경에서 돌리기 위해 작업 정의를 업데이트 해준다.
- DB 컨테이너는 삭제
- Backend 컨테이너에 환경 변수 설정을 바꿔준다.

## 작업 정의의 새 개정 생성

특정 애플리케이션에 맞게 아래의 복사된 작업 정의를 수정하십시오. AWS  
있습니다. [자세히 알아보기](#)

태스크 정의 이름\*

goal

- Name은 그냥 goals로 설정한다. 개발 환경이 아니기 때문

Key

MONGODB\_NAME

Value

goals

MONGODB\_PASSWORD

Value

zpyVV1QIHeDklm1V

MONGODB\_URL

Value

cluster0.xnfdgy.mongodb.net

MONGODB\_USERNAME

Value

poland

# AWS ECS 서비스 업데이트 후 가동

## 로드 밸런싱

서비스 생성 시에 로드 밸런싱 설정을 설정할 수 있습니다.

Load Balancer 이름: ecs-lb

컨테이너 이름: goal-backend

컨테이너 포트: 80

대상 그룹: arn:aws:elasticloadbalancing:ap-northeast-2:453043152051:targetgroup/target/c46b7dd7cc54914c

	이름	컨테이너 런타임 ID	상...	이미지
▶	goal-backe...	a64782e8db064721b...	RU...	millwheel/goals-node

ELB 연결, Spring 앱 배포

# spring app 이미지 빌드, 도커 허브에 push

```
PS D:\delivery project\delivery-application\rider> docker build -t millwheel/rider-app .  
[+] Building 1.4s (7/7) FINISHED
```

```
PS D:\delivery project\delivery-application\customer> docker build -t millwheel/customer-app .  
[+] Building 2.5s (8/8) FINISHED
```

```
PS D:\delivery project\delivery-application\restaurant> docker build -t restaurant-app .  
[+] Building 1.8s (7/7) FINISHED
```

```
PS D:\delivery project\delivery-application\rider> docker push millwheel/rider-app  
Using default tag: latest
```

```
PS D:\delivery project\delivery-application\customer> docker push millwheel/customer-app  
Using default tag: latest
```

```
PS D:\delivery project\delivery-application\restaurant> docker push millwheel/restaurant-app  
Using default tag: latest
```

# ECS 클러스터 생성, 작업 정의 생성

- 클러스터를 3개 생성하여 구분해준다. 태스크 정의도 각각 생성

클러스터 ▼	서비스 ▼	태스크 ▼
delivery-customer	1	<div><div></div>0개 대기 중   1개 실행 중</div>
delivery-rider	0	실행 중인 태스크 없음
delivery-restaurant	0	실행 중인 태스크 없음

태스크 정의
<input type="radio"/> delivery-customer
<input type="radio"/> delivery-restaurant
<input type="radio"/> delivery-rider



컨테이너 이름*	rider-container
이미지*	millwheel/rider-app

포트 매핑	컨테이너 포트
	80



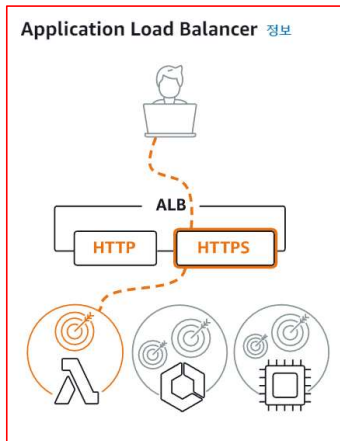
# ECS 서비스 생성 전 로드밸런서 생성

- ECS 서비스에서 사용할 로드밸런서를 먼저 생성해준다.
- 로드밸런서는 EC2 로드밸런싱에서 생성할 수 있다.

로드 밸런서 (1)							로드 밸런서 생성
Elastic Load Balancing은 수신 트래픽의 변화에 따라 자동으로 로드 밸런서 용량을 확장합니다.							
Q 속성 또는 값을 기준으로 필터링							< 1 > ⚙
<input type="checkbox"/>	이름 ▼	DNS 이름 ▼	상태 ▼	VPC ID ▼	가용 영역 ▼	유형	
<input type="checkbox"/>	ecs-customer-lb	ecs-customer-lb-4039289...	✓ Active	vpc-021ee50e9fdd056a0	2 가용 영역	appl	

# ECS 서비스 생성 전 로드밸런서 생성

- 각 클러스터를 생성할 때 만든 VPC에 매칭하여 가용영역을 선택한다. 알맞은 VPC를 선택해야 정상 동작한다.



로드 밸런서 이름  
이름은 AWS 계정 내에서 고유해야 하며 로드 밸런서  
하이픈을 포함하여 최대 32자의 영숫자 문자를 사용

ecs-rider-lb

체계 정보  
로드 밸런서 생성 후에는 스키마를 변경할 수 없습니

☒ 인터넷 경계  
인터넷 경계 로드 밸런서는 인터넷을 통해 클라

☐ 내부  
내부 로드 밸런서는 프라이빗 IP 주소를 사용하여

IP 주소 유형 정보  
서브넷이 사용하는 IP 주소 유형을 선택합니다.

☒ IPv4  
내부 로드 밸런서에 권장합니다.

☐ 듀얼 스택  
IPv4 및 IPv6 주소를 포함합니다.

VPC 정보  
대상 에 대한 Virtual Private Cloud(VPC)를 선택합니다. 인터넷 게이트웨이가 있는 VPC만 선택할 수 있습니다.  
그러면 대상 그룹을 참조하세요.

vpc-022734a5b4fa0b88f  
IPv4: 10.0.0.0/16

매핑 정보  
가용 영역을 2개 이상 선택하고 영역당 하나의 서브넷을 선택합니다. 로드 밸런서는 이러한 가용 영역의 대상으  
택할 수 없습니다.

☒ ap-northeast-2a (apne2-az1)

서브넷  
subnet-035406025d0400aac delivery-rider/Public ▼

IPv4 설정  
AWS에서 할당

☒ ap-northeast-2b (apne2-az2)

# 주의! 보안 그룹

- Default 보안 그룹을 사용하면 트래픽이 접근할 수 없다. (외부에서 이 주소로 접근할 수 없으므로 어플리케이션 이용 못함)

## 보안 그룹 정보

보안 그룹은 로드 밸런서에 대한 트래픽을 제어하는 방화벽 규칙 세트입니다.

## 보안 그룹

최대 5개의 보안 그룹 선택

새 보안 그룹 생성 

default sg-05ad5f2c51ebeefd6 ✕  
VPC: vpc-022734a5b4fa0b88f

# 보안 그룹 생성

- 인바운드 규칙: 외부에서 앱으로 접속할 때 규칙
- 아웃바운드 규칙: 앱에서 외부로 접근할 때 규칙
- 인바운드 규칙에 80번 포트 + 모든 IP 개방으로 설정

보안 그룹 이름 정보

delivery-rider-sg

생성 후에는 이름을 편집할 수 없습니다.

설명 정보

security group for rider application

VPC 정보

Q vpc-022734a5b4fa0b88f X

인바운드 규칙 정보

유형 정보

사용자 지정 TCP ▼

프로토콜  
정보

TCP

포트 범위 정보

80

소스 정보

Anywh... ▼

0.0.0.0/0 X

- 주의사항: 현재 만들고자 하는 로드밸런서에서 매핑한 VPC와 똑같은 VPC를 선택해야한다.

# 보안 그룹 선택

- 방금 생성한 보안 그룹을 지정한다.
- Default 보안그룹은 삭제

## 보안 그룹 [정보](#)

보안 그룹은 로드 밸런서에 대한 트래픽을 제어하는 방화벽 규칙

보안 그룹

최대 5개의 보안 그룹 선택

[새 보안 그룹 생성](#)

delivery-rider-sg sg-067ec99b6749ace98 ✕  
VPC: vpc-022734a5b4fa0b88f

# 대상 그룹 생성

- 리스너가 요청을 분배할 대상그룹을 설정해야 한다.

▼ 리스너 HTTP:80

프로토콜

포트

기본 작업

정보

HTTP ▼

:

80

1-65535

다음으로 전달: 대상 그룹 선택

대상 그룹 생성 ↗

대상 유형 선택

☐ 인스턴스

- 특정 VPC 내의 인스턴스에 대한 로드 밸런싱을 지원합니다.
- Amazon EC2 Auto Scaling ↗을 사용하여 EC2 용량을 관리하고 크기를 조정합니다.

☒ IP 주소

- VPC 및 온프레미스 리소스에 대한 로드 밸런싱을 지원합니다.
- 동일한 인스턴스에 있는 여러 IP 주소 및 네트워크 인터페이스로의 라우팅을 지원합니다.
- 마이크로서비스 기반 아키텍처를 통한 유연성을 제공하여 애플리케이션 간 통신을 용이하게 합니다.
- IPv6 대상을 지원하여 중단 간 IPv6 통신 및 IPv4에서 IPv6로의 NAT를 활성화합니다.

대상 그룹 이름

rider

A maximum of 32 alphanumeric characters including hyphens are allowed.

프로토콜

포트

HTTP ▼

:

80

IP 주소 유형

표시된 IP 주소 유형의 대상만 이 대상 그룹에 포함될 수 있습니다.

☒ IPv4

☐ IPv6

# 대상 그룹 선택 후 로드밸런서 생성

- 방금 생성한 대상 그룹을 선택

프로토콜	포트	기본 작업	정보
HTTP ▼	: 80 1-65535	다음으로 전달:	rider Target type: IP, IPv4
<a href="#">대상 그룹 생성</a> 			

- 그 외 설정은 손대지 말고 로드밸런서를 생성한다.

취소

로드 밸런서 생성

# 로드밸런서 목록

- 로드밸런서가 Active 상태가 되면 ECS 서비스 생성을 계속 진행한다.

로드 밸런서 (2)			
Elastic Load Balancing은 수신 트래픽의 변화에 따라 자동으로 로드 밸런서 용량을 확장합니다.			
Q 속성 또는 값을 기준으로 필터링			
<input type="checkbox"/>	이름 ▼	DNS 이름 ▼	상태 ▼
<input type="checkbox"/>	ecs-rider-lb	ecs-rider-lb-269118122.a...	⋮ Provisioning
<input type="checkbox"/>	ecs-customer-lb	ecs-customer-lb-4039289...	✔ Active



# ECS 서비스 및 작업 생성

서비스 없음  
표시할 서비스가 없습니다.

생성

## ▼ 컴퓨팅 구성 (고급)

### 컴퓨팅 옵션 [정보](#)

여러 컴퓨팅 유형에 태스크를 분산시키려면 적절한 컴퓨팅 옵션을 사용합니다.

☐ 용량 공급자 전략  
하나 이상의 용량 공급자에 태스크를 분산하는 시작 전략을 지정합니다.

☒ 시작 유형  
용량 공급자 전략을 사용하지 않고 직접 태스크를 시작합니다.

### 시작 유형 [정보](#)

관리형 용량(Fargate) 또는 사용자 지정 용량(EC2 또는 사용자 관리형 External 인스턴스)을 선택합니다. External을 사용하여 클러스터에 등록됩니다.

FARGATE ▼

### 플랫폼 버전 [정보](#)

서비스를 실행할 플랫폼 버전을 지정합니다.

LATEST ▼

- 패밀리를 선택해야 로드밸런서를 사용할 수 있다.

### 애플리케이션 유형 [정보](#)

실행할 애플리케이션 유형을 지정합니다.

☒ 서비스  
중지 및 다시 시작할 수 있는 장기 실행 컴퓨팅 작업(예: 웹 애플리케이션)을 처리하는 태스크 그룹을 시작합니다.

☐ 태스크  
실행하고 종료하는 독립 실행형 태스크(예: 배치 작업)를 시작합니다.

### 패밀리

### 개정

delivery-rider ▼

1 (최신) ▼

### 서비스 이름

이 서비스에 대한 고유한 이름을 지정합니다.

rider-service

# 주의! 서비스 생성에서 네트워킹 체크 필수

- VPC를 알맞게 매칭해야 한다.
- 보안 그룹도 로드밸런서에 적용한 것과 일치하는 것을 선택한다.

## ▼ 네트워킹

### VPC [정보](#)

사용할 가상 사설 클라우드를 선택합니다.

vpc-022734a5b4fa0b88f

### 서브넷

태스크 스케줄러가 배치 시 고려해야 하는 VPC 내 서브넷을 선택합니다.

서브넷 선택

subnet-035406025d0400aac ✕  
퍼블릭 | ap-northeast-2a | delivery-rider/Public

subnet-08407e7b20b27bb72 ✕  
퍼블릭 | ap-northeast-2b | delivery-rider/Public

### 보안 그룹 [정보](#)

기존 보안 그룹을 선택하거나 새 보안 그룹을 생성합니다.

☒ 기존 보안 그룹 선택

☐ 새 보안 그룹 생성

보안 그룹 이름

기존 보안 그룹을 선택합니다.

sg-067ec99b6749ace98 ✕  
delivery-rider-sg

### 퍼블릭 IP [정보](#)

태스크의 탄력적 네트워크 인터페이스(ENI)에 퍼블릭 IP를 자동 지정할지 여부를 선택합니다.

☒ 켜짐

# 로드밸런싱 매칭

- 기존에 만들어놓은 로드밸런서와 대상그룹을 매칭해준다.
- 생성 누름

로드 밸런서 유형 [정보](#)

서비스에서 실행 중인 태스크에서 수신 트래픽을 분산하도록 로드 밸런서를 구성합니다.

Application Load Balancer ▼

Application Load Balancer

새 로드 밸런서를 생성할지, 아니면 기존 로드 밸런서를 선택할지 지정합니다.

- ☐ 새 로드 밸런서 생성  
☒ 기존 로드 밸런서 사용

로드 밸런서

서비스에서 실행 중인 태스크에 수신 트래픽을 분산하는 데 사용할 로드 밸런서를 선택합니다.

ecs-rider-lb ▼

로드 밸런싱할 컨테이너 선택

rider-container 80:80 ▼

취소

생성

리스너 [정보](#)

로드 밸런서가 연결 요청을 수신 대기할 포트 및 프로토콜을 지정합니다.

포트

80

프로토콜

HTTP

대상 그룹 [정보](#)

새 대상 그룹을 생성할지, 아니면 로드 밸런서가 요청을 서비스의 태스크에 라우팅하는 데 사용할 기존 대상 그룹을 선택할지 지정합니다.

- ☐ 새 대상 그룹 생성  
☒ 기존 대상 그룹 사용

대상 그룹 이름

rider ▼

상태 확인 경로

/

상태 확인 프로토콜

HTTP

# 동작 확인

- 로드밸런서를 통해 웹 어플리케이션에 접근 할 수 있다.

ecs-rider-lb

▼ 세부 정보

arn:aws:elasticloadbalancing:ap-northeast-2:453043152051:loadbalancer/app/ecs-rider-lb

로드 밸런서 유형	DNS 이름
Application	ecs-rider-lb-269118122.ap-northeast-2.elb.amazonaws.com (A 레코드)

← → ↻ ⚠ 주의 요함 | ecs-rider-lb-269118122.ap-northeast-2.elb.amazonaws.com

YouTube 프로그래머스 Cambridge Dictionary Papago 정규표현식 (Regex)...

Rider server is activated successfully

- Task의 퍼블릭 IP로도 물론 접근할 수 있다. (퍼블릭 OPEN 상태)

용량 공급자	ENI ID	퍼블릭 IP
-	<a href="#">eni-0da6ed855294f2072</a>	<a href="#">3.36.59.194</a>   주소 열기
시작 유형	네트워크 모드	프라이빗 IP
FARGATE	awsvpc	<a href="#">10.0.0.205</a>

← → ↻ ⚠ 주의 요함 | 3.36.59.194

YouTube 프로그래머스 Cambridge Dictionary

Rider server is activated successfully