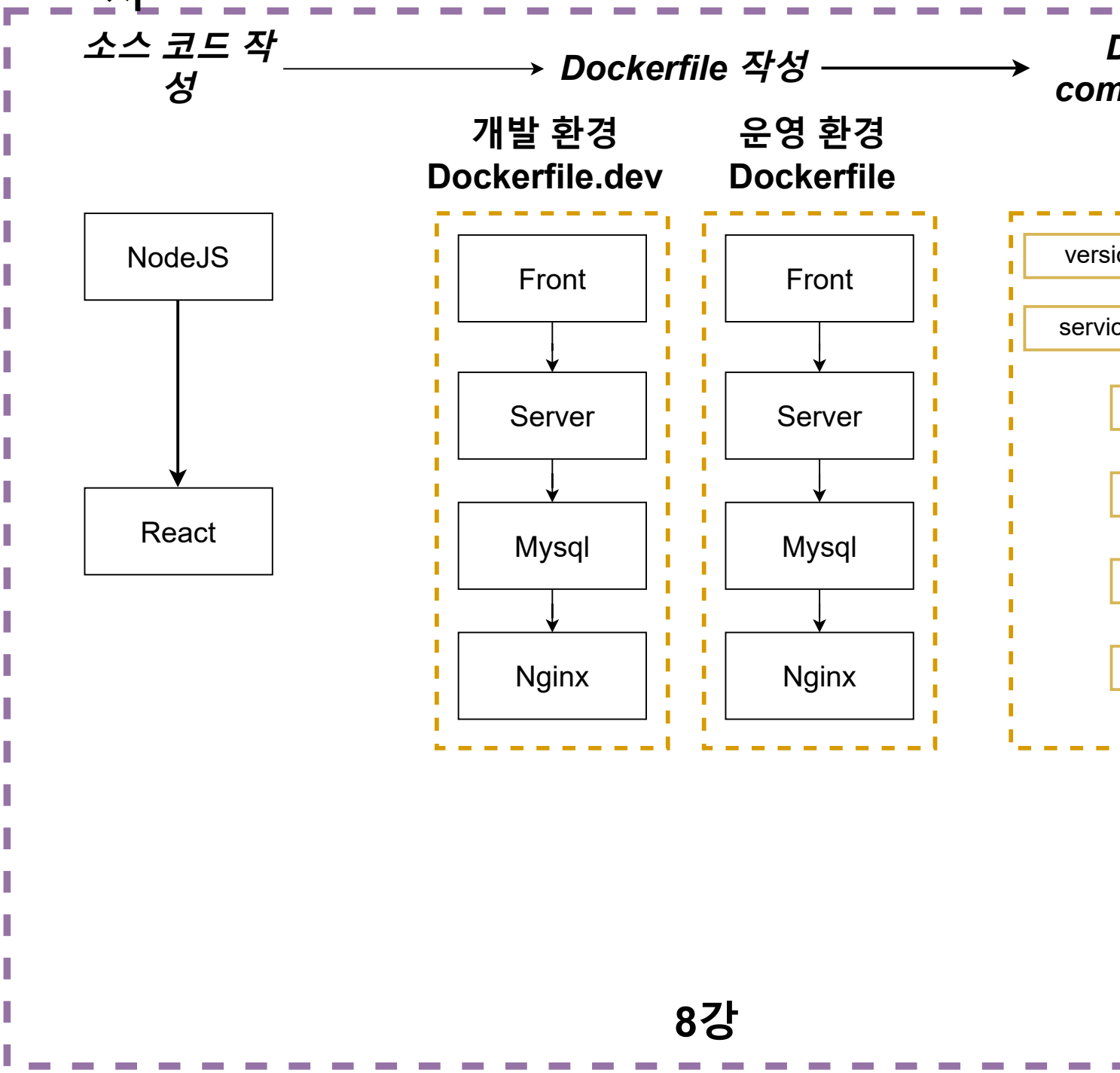
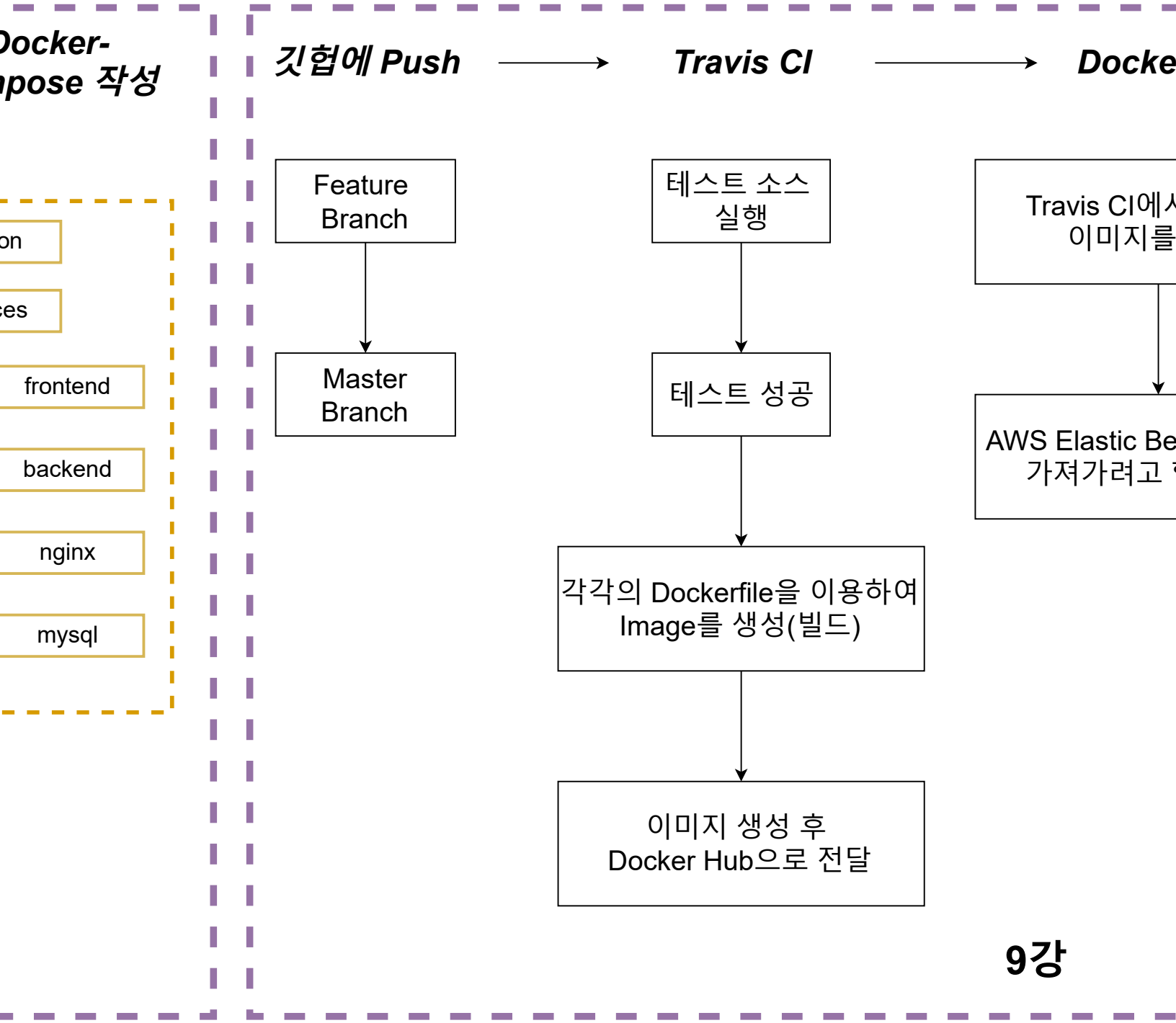
복잡한 애플리케이션을 배포하기

섹션 설명

구현 순서



8강에서 애플리케이션을 위한 전체 소스코드를 작성하고 간간에 맞는


Docker-
mpose 작성
on
ces
frontend
backend
nginx
mysql
깃헙에 Push
Travis CI
Docke
Feature
Branch
Master
Branch
테스트 소스
실행
테스트 성공
각각의 Dockerfile을 이용하여
Image를 생성(빌드)
이미지 생성 후
Docker Hub으로 전달
Travis CI에서
이미지를
AWS Elastic Be
가져가려고
9강


r Hub
AWS
ElasticBeanStalk
서 빌드된
보관
Elastic Beanstalk을 통해서
배포
anstalk에서
할때 전달

8강에서 애플리케이션을 위한 전체 소스코드를 작성하고 각각에 맞는 Dockerfile을 작성했으며 그 각각의 컨테이너를 연결시켜주고 간단하게 한 번에 실행시킬 수 있는 Docker Compose를 작성했습니다.

이번 9강에서는 소스코드가 에러가 없는지 테스트를 한 후 테스트에서 성공을 하면 AWS를 통해서 배포를 하는 것까지 해보겠습니다.

하지만 데이터 베이스를 운영 환경에서는 도커를 이용하는 게 아닌 AWS RDS를 이용하므로 그에 맞게 8강에서 도커를 이용해서 MYSQL을 이용한 부분을 먼저 정리해 주겠습니다.

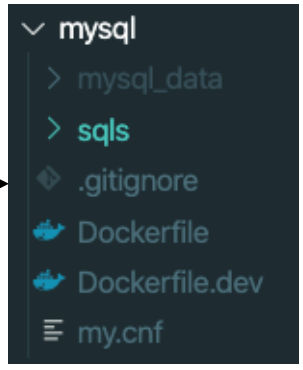
도커 환경의 MYSQL부분 정리하기

Mysql이 이제는 도커 안에서 돌아가는 게 아닌 AWS에서 돌아가고 있는 것을 우리의 애플리케이션에 연결만 해줄 것이니 Mysql을 애플리케이션에 연결해주는 부분

docker-compose.yml 에서 mysql부분 지워주기

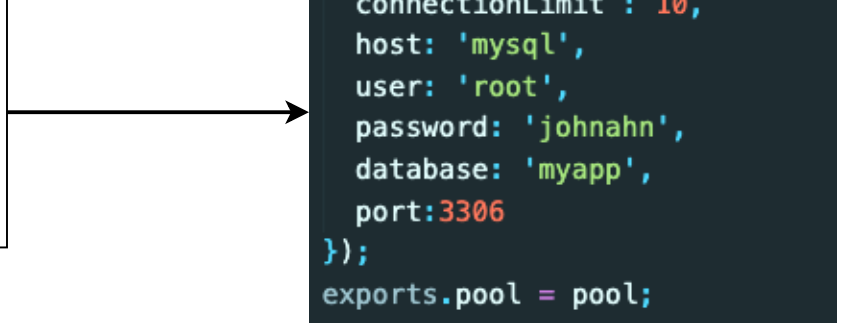
```
# mysql:
#   build: ./mysql
#   restart: unless-stopped
#   container_name: app_mysql
#   ports:
#     - "3306:3306"
#   volumes:
#     - ./mysql/mysql_data:/var/lib/mysql
#     - ./mysql/sqls/:/docker-entrypoint-initdb.d/
#   environment:
#     MYSQL_ROOT_PASSWORD: johnahn
#     MYSQL_DATABASE: myapp
```

mysql 폴더를 지워주기 (하지만 이미 docker-compose.yml에서 mysql 부분을 지워줬기에 안 지워도 작동은 안 된다.)




```
var mysql = require("mysql");
var pool  = mysql.createPool({
```

이 부분에서 host와 같은
정보들은 AWS에서
DB 생성 후
내용을 다시 넣어주기



```
  connectionLimit : 10,
  host: 'mysql',
  user: 'root',
  password: 'johnahn',
  database: 'myapp',
  port:3306
});
exports.pool = pool;
```

이렇게 운영환경을 위한 DB 부분에서

불필요한 부분을 제거해주었습니다.

그럼 우선 이 소스를 Github에 올려서

배포하는 부분을 하나하나 해보겠습니다.

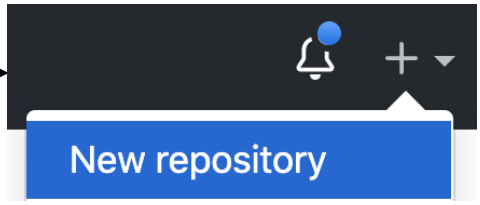
Github에 소스 코드 올리기

먼저는 리액트 애플리케이션을 배포했을 때처럼 먼저는 깃 헙에 소스를 배포를 해보겠습니다.

Steps

GIT HUB 아이디가 없으면 가입! → https://github.com/

Github에서 NEW REPOSITORY 만들기 →



로컬 git 저장소 만들기 → git init

현재까지의 소스를 로컬 git 저장소에 저장 →

```
git add .

git commit -m  "first commit"
```

로컬 git 저장소와 Github Remote을 연결시키기 →

```
git remote add origin
https://github.com/jaewonhimnae/test123.git
```

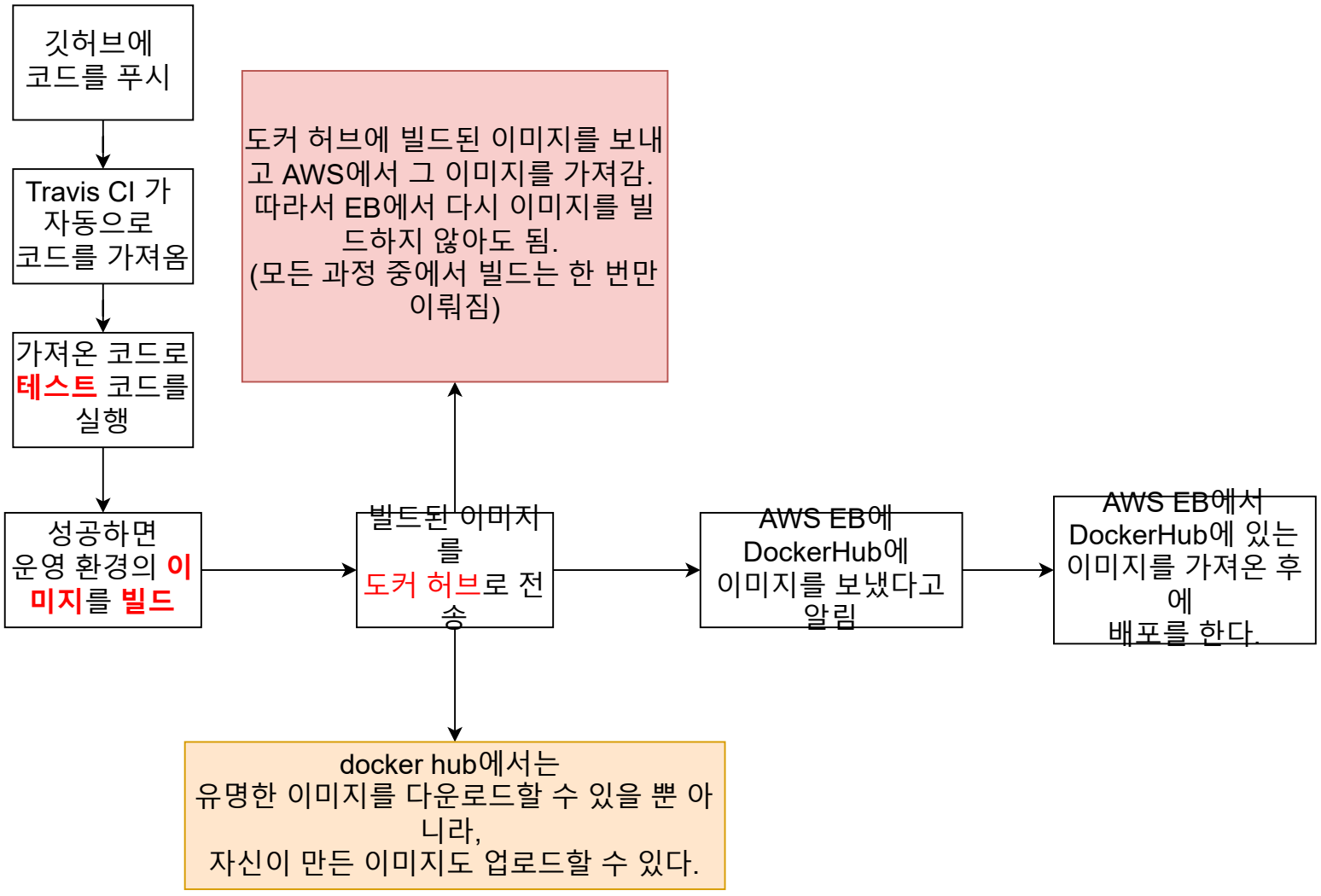
로컬 git 저장소에 있는 소스를 github에 업로드 시키 → git push origin master

가

Travis CI Steps

깃헙에 Master 브랜치에 새로 업데이트된 소스가 올라왔다면
이제는 그 소스를 Travis CI에서 가져와줘야 합니다.

Travis CI 에서 할 일들

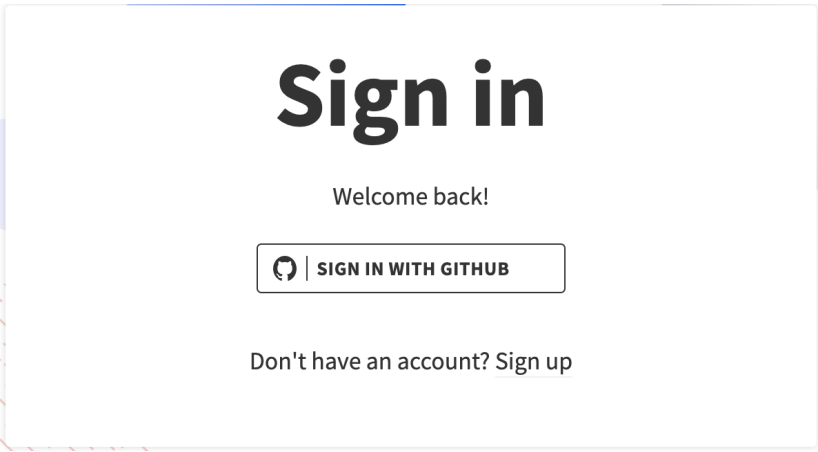


순서

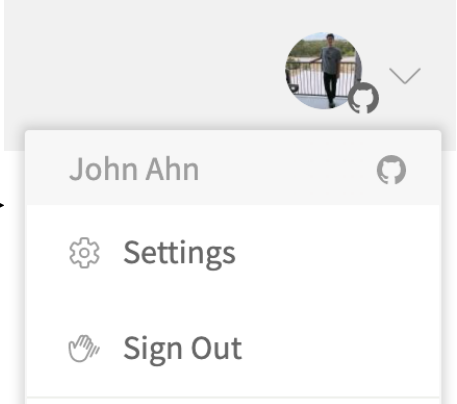
Travis CI 사이트로 이동 → https://travis-ci.org/

Travis CI 사이트로 이동

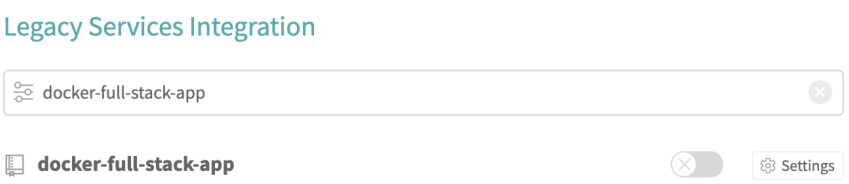
깃헙 아이디로 로그인



Settings 페이지로 이동



깃헙에 방금 올린 저장소를
Travis CI에서 찾기

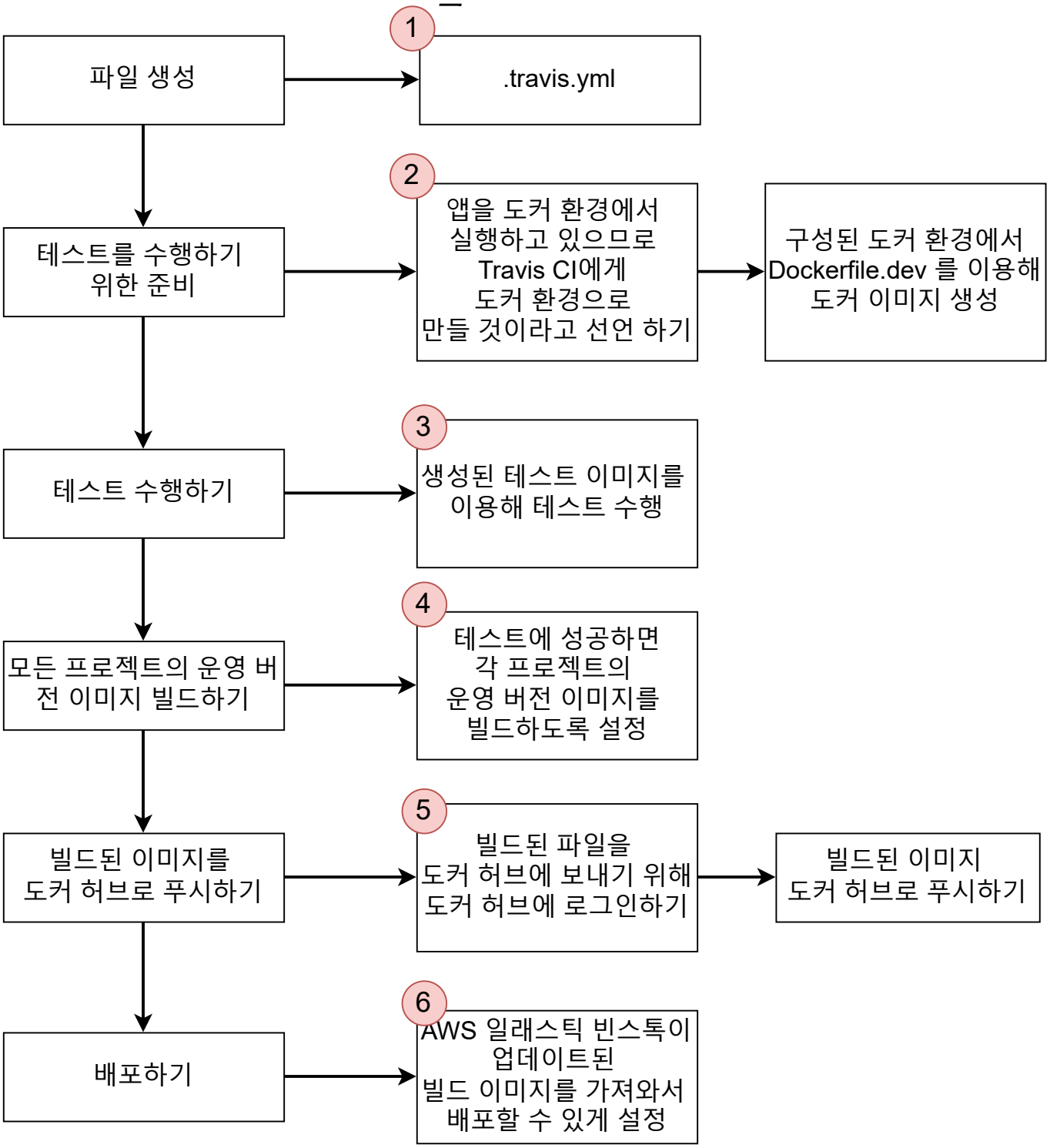


해당 저장소의 Setting 버튼을 눌러서 Travis CI에게
Github 저장소의 소스가 변경될 때마다 소스를 가져와서 테스트하고 배포하라고 알려준다.

travis.yml 파일 작성(이미지 빌드까지)

어떻게 작성하나

간단하게 *구체적으*



```yaml
language: generic

sudo: required

services:
  - docker

before_install:
  - docker build -t smileajw1004/react-test-app -f ./frontend/Dockerfile.dev ./frontend

script:
  - docker run -e CI=true smileajw1004/react-test-app npm test
```

language

sudo

services

before_insta

script

docker | build | -t <도커 아이디> / <어플 이름> | -f <Dockerfile 경로> | 빌드 해야 할 파
있는 경로

```yaml
after_success:
  - docker build -t smileajw1004/docker-frontend ./frontend
  - docker build -t smileajw1004/docker-backend ./backend
  - docker build -t smileajw1004/docker-nginx ./nginx

  - echo "$DOCKER_HUB_PASSWORD" | docker login -u "$DOCKER_HUB_ID" --password-stdin

  - docker push smileajw1004/docker-frontend
  - docker push smileajw1004/docker-backend
  - docker push smileajw1004/docker-nginx
```

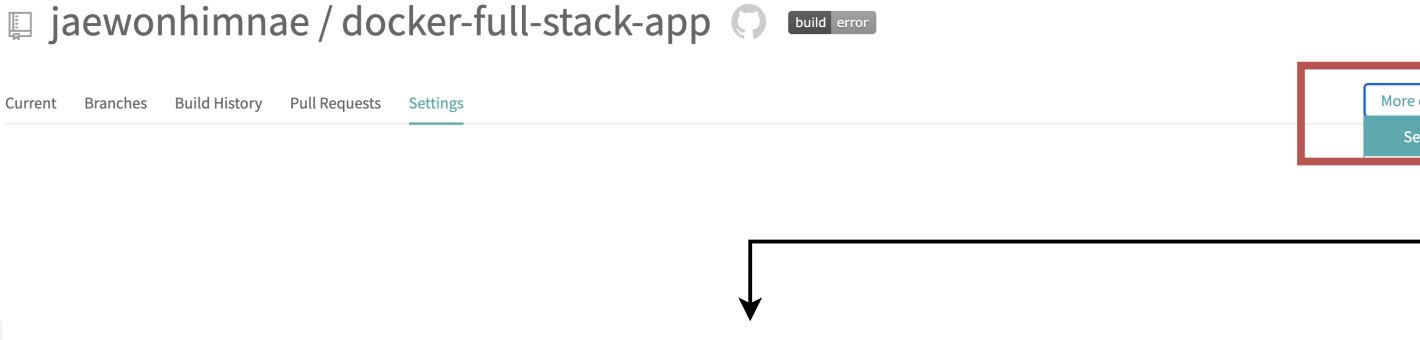
after_suc

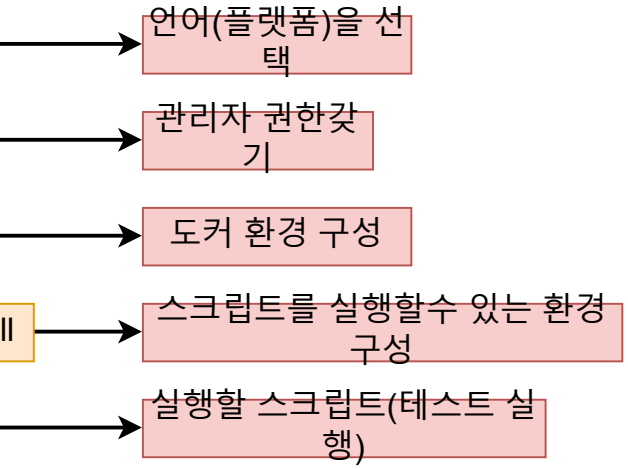
각각의 이

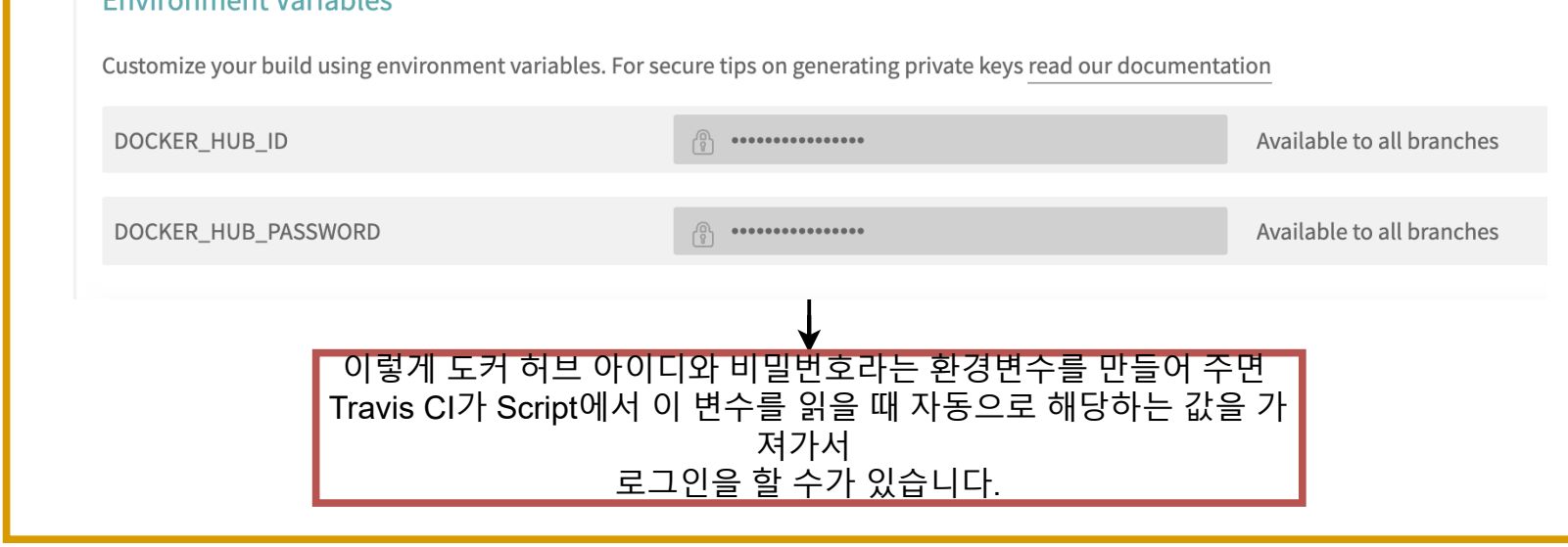
도커 허브

빌드 된

.travis.yml에 파일에 아이디와 비밀번호를 넣으면 노출 위험이 있기에 도커 허브 아이디와 비밀번호를 Travis CI 홈페이지에 미리 넣어줘야 한다.




언어(플랫폼)을 선택
관리자 권한갖기
도커 환경 구성
ll
스크립트를 실행할수 있는 환경 구성
실행할 스크립트(테스트 실행)
일들이
ccess
테스트 성공 후 할일
미지를 빌드하기
브에 로그인
이미지들을 도커 허브에 Push 하기



이렇게 도커 허브 아이디와 비밀번호라는 환경변수를 만들어 주면 Travis CI가 Script에서 이 변수를 읽을 때 자동으로 해당하는 값을 가져가서
로그인을 할 수가 있습니다.

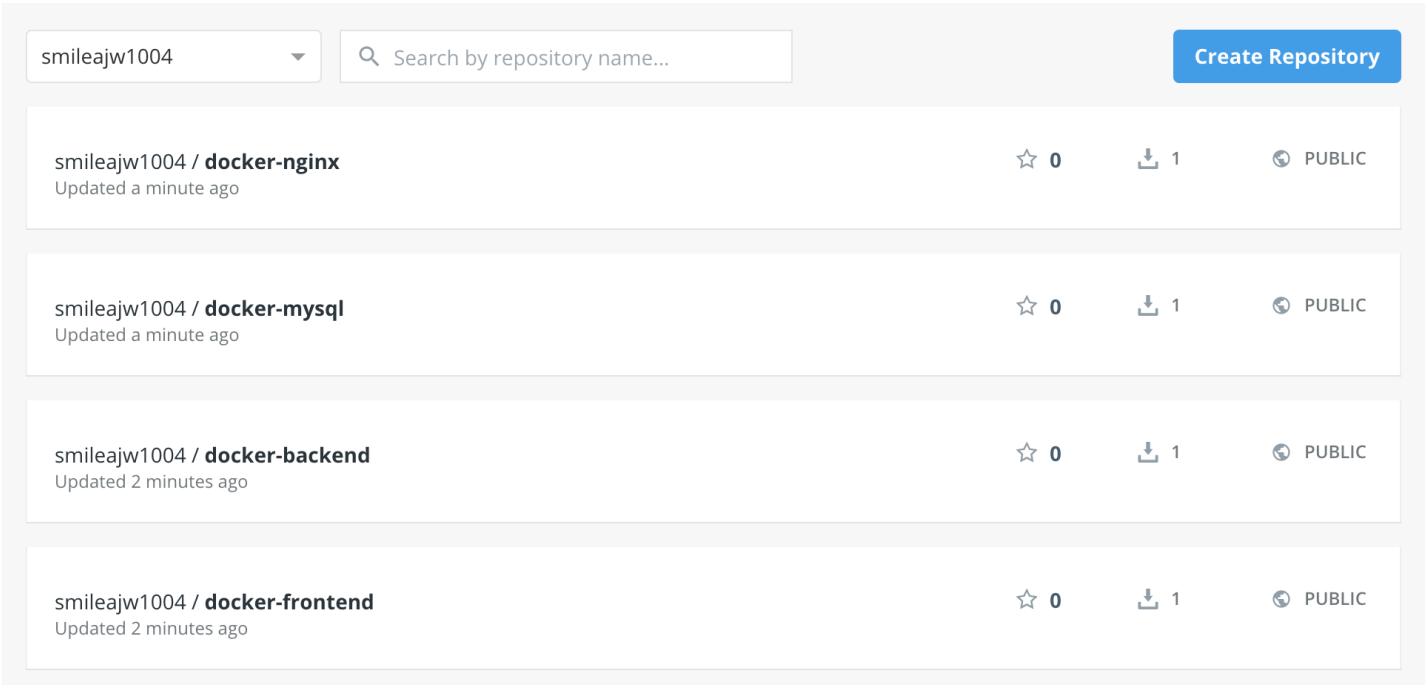
여기까지 한 후 한번 Github에다가 Push 해서 이미지가 잘 빌드돼서 Docker HUB에 올라가는지 확인해보기!

Travis CI 결과

```
166 Setting environment variables from repository settings
167 $ export DOCKER_HUB_ID=[secure]
168 $ export DOCKER_HUB_PASSWORD=[secure]
169
170 $ bash -c 'echo $BASH_VERSION'
171 4.3.48(1)-release
172
173 $ docker build -t [secure]/react-test-app -f ./frontend/Dockerfile.dev ./frontend
278 $ docker run -e CI=true [secure]/react-test-app npm test
279
280 > front@0.1.0 test /app
281 > react-scripts test
282
283 PASS src/App.test.js
284   ✓ renders without crashing (69ms)
285
286 Test Suites: 1 passed, 1 total
287 Tests:       1 passed, 1 total
288 Snapshots:   0 total
289 Time:        1.555s
290 Ran all test suites.
291 The command "docker run -e CI=true [secure]/react-test-app npm test" exited with 0.
292
293 $ docker build -t [secure]/docker-frontend ./frontend
376 $ docker build -t [secure]/docker-backend ./backend
417 $ docker build -t [secure]/docker-mysql ./mysql
476 $ docker build -t [secure]/docker-nginx ./nginx
484 $ echo "$DOCKER_HUB_PASSWORD" | docker login -u "$DOCKER_HUB_ID" --password-stdin
490 $ docker push [secure]/docker-frontend
509 $ docker push [secure]/docker-backend
531 $ docker push [secure]/docker-mysql
```

```
565 $ docker push [secure]/docker-nginx
581
582 Done. Your build exited with 0.
```

Docker Hub에 올라온 이미지들



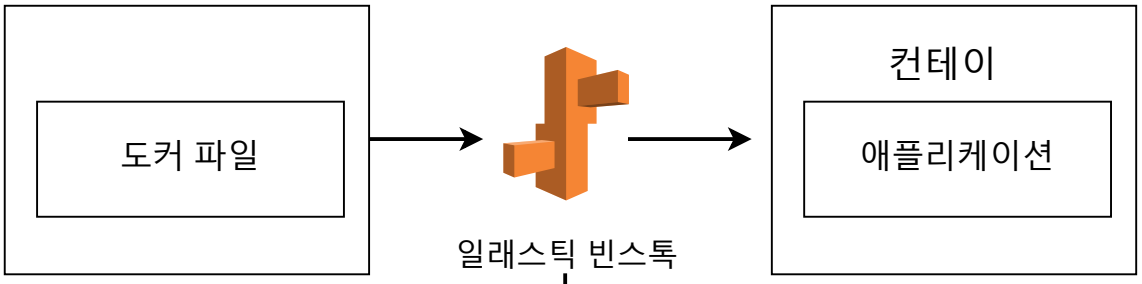
Dockerrun.aws.json에 대해서

**이제 Dockerrun.aws.json이라는 파일을 쓸 때가 왔습니다.
그러기에 우선 그 파일이 무엇인지 알아봐야**

6,7강에서 리액트만을 이용해서 앱을 만들 때는 Dockerrun.aws.json이라는 파일을 사용하지 않았었는데 이번에는 이 파일을 써야지 ElastickBeanstalk에서 애플리케이션을 작동을 시킬 수 있습니다.

왜 그런지 우선 알아보겠습니다.

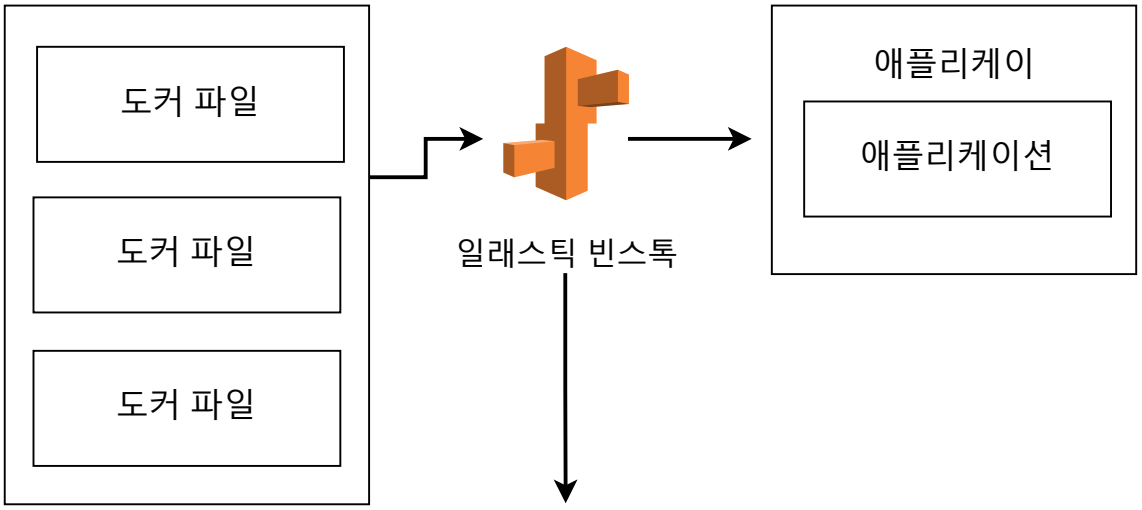
도커 파일이 하나 (리액트 애플리케



저번에 리액트만을
그래서 그 도커 파
그 빌드된 이미지를
아무런 설정을 저희

하나의 도커 파일 ! 내가 알아서 처리할게 !

도커 파일이 여러 개 (풀 스택 애플리케



하지만 이번에
그러기에 EB
자동으로 프로
저희가 임의로
그걸 설정해주

여러 개의 도커 파일, 어떻게 해야 할지 모르겠어 도와줄래?

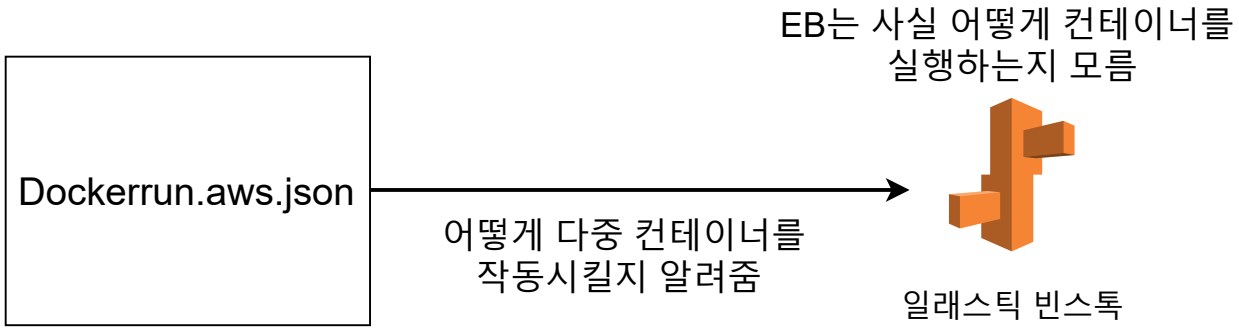
을 이용한 애플리케이션을 만들 때는 Dockerfile이 하나였습니다.
일을 Elastic beanstalk에 전달하면 EB가 알아서 이미지를 빌드하고
를 돌려서 애플리케이션을 실행하였습니다.
희가 해주지 않아도 됐었습니다.

에는 노드, Mysql, Nginx 등을 위한 Dockerfile이 여럿 있습니다.
가 어떤 파일을 먼저 실행하고 어떻게 행동을 취해야 하는지
로세스를 해나갈 수 없기 때문에
로 설정을 해줘야 하는데요.
주는 파일이 바로 Dockerrun.aws.json입니다.

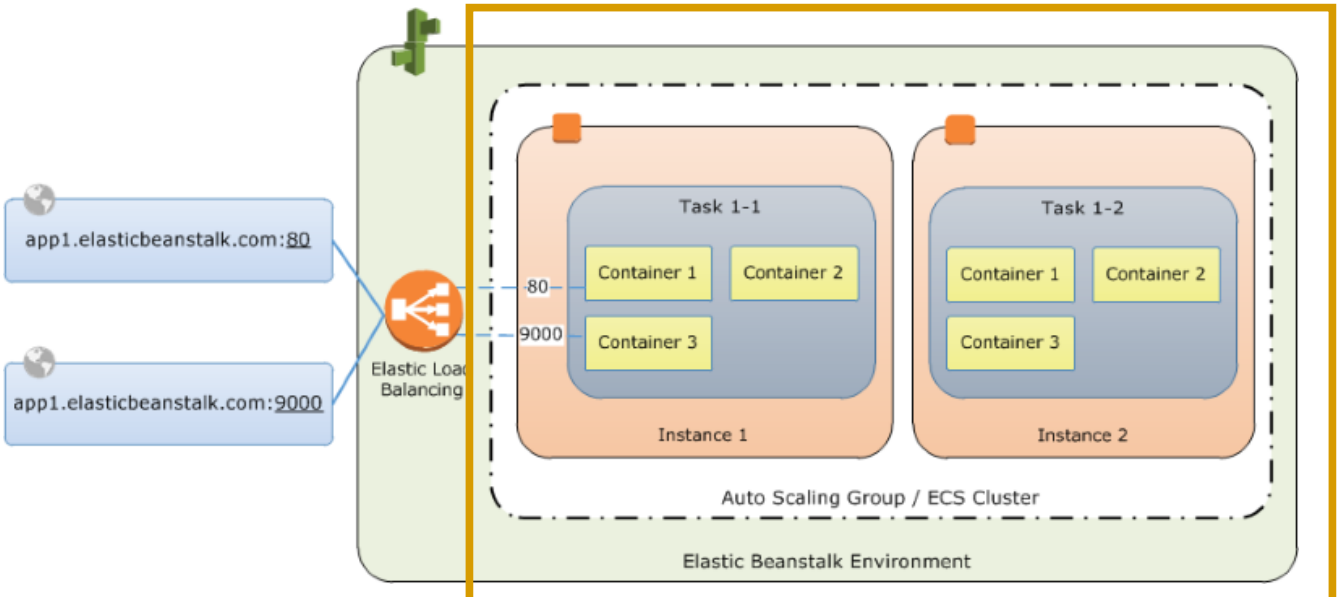
AWS에서 말하는 Dockerrun.aws.json 파일의 정의

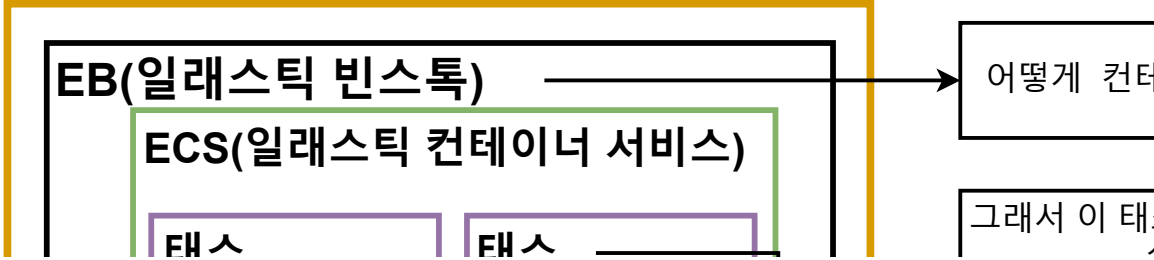
`Dockerrun.aws.json` 파일은 Docker 컨테이너 세트를 Elastic Beanstalk 애플리케이션으로 배포하는 방법을 설명하는 Elastic Beanstalk 고유의 JSON 파일입니다. `Dockerrun.aws.json` 파일을 멀티컨테이너 Docker 환경에 사용할 수 있습니다

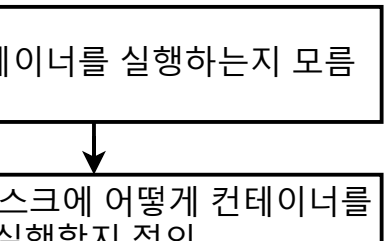
`Dockerrun.aws.json`은 환경에서 각 컨테이너 인스턴스(Docker 컨테이너를 호스트하는 Amazon EC2 인스턴스)에 배포할 컨테이너 및 탑재할 컨테이너의 호스트 인스턴스에서 생성할 데이터 볼륨을 설명합니다.



AWS 제공하는 도표로 좀 더 깊게 이해



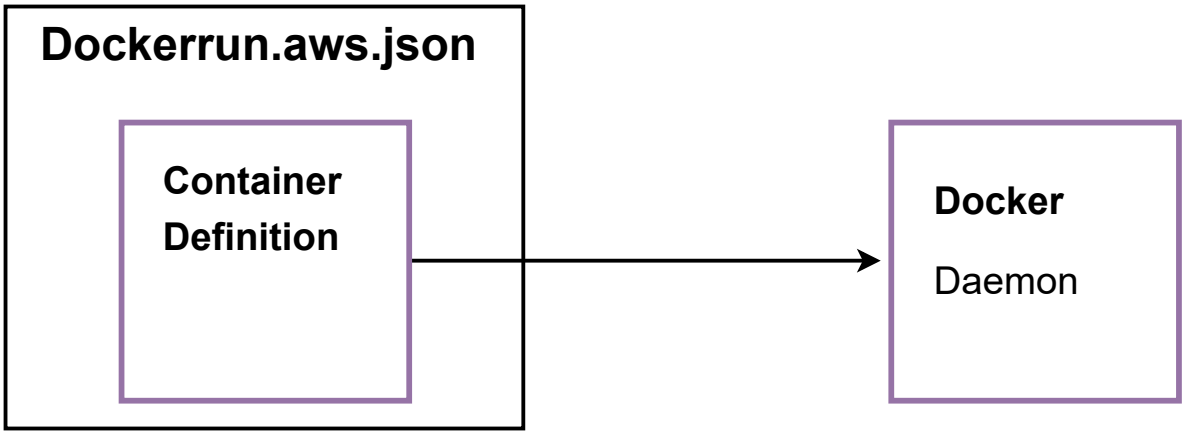



이너를 실행하는지 모름
스크에 어떻게 컨테이너를
실행할지 정의

AWS에서 말하는 Task Definition(작업 정의)에서 지정 할수 있는 것들

- 작업의 각 컨테이너에 사용할 도커 이미지
- 각 작업 또는 작업 내 각 컨테이너에서 사용할 CPU 및 메모리의 양
- 사용할 시작 유형으로서 해당 작업이 호스팅되는 인프라를 결정
- 작업의 컨테이너에 사용할 도커 네트워킹 모드
- 작업에 사용할 로깅 구성
- 컨테이너가 종료 또는 실패하더라도 작업이 계속 실행될지 여부
- 컨테이너 시작 시 컨테이너가 실행할 명령
- 작업의 컨테이너에서 사용할 데이터 볼륨
- 작업에서 사용해야 하는 IAM 역할

이 작업 정의를 등록하려면 Container Definition을 명시해줘야 한다.
그리고 그 Container Definition은 dockerrun.aws.json에 명시해주며
도커 데몬으로 전해진다.



실행할지 정의
의(Task Definition)

업 정의를 등록할 때는
(Container Definition)를
명시해야함

테이너 정의는
run.aws.json에 명시

Dockerrun.aws.json 작성하기

Dockerrun.aws.json
파일 생성

↓

Container Definitions을
작성하기

```
{
  "AWSEBDockerrunVersion": 2,
  "containerDefinitions": [
    {
      "name": "frontend",
      "image": "smileajw1004/docker-frontend",
      "hostname": "frontend",
      "essential": false,
      "memory": 128
    },
    {
      "name": "backend",
      "image": "smileajw1004/docker-frontend",
      "hostname": "backend",
      "essential": false,
      "memory": 128
    },
    {
      "name": "nginx",
      "image": "smileajw1004/docker-nginx",
      "hostname": "nginx",
      "essential": true,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ],
```

AWSEBDockerrunVersion →

containerDefinitions →

이 객체안에서 하나의
다.

name →

image →

hostname →

essential →

Dockerrun 버전 2로 지정

이 안에서 컨테이너들을 정의해줍니다.

컨테이너를 정의합니

컨테이너의 이름 입니다.

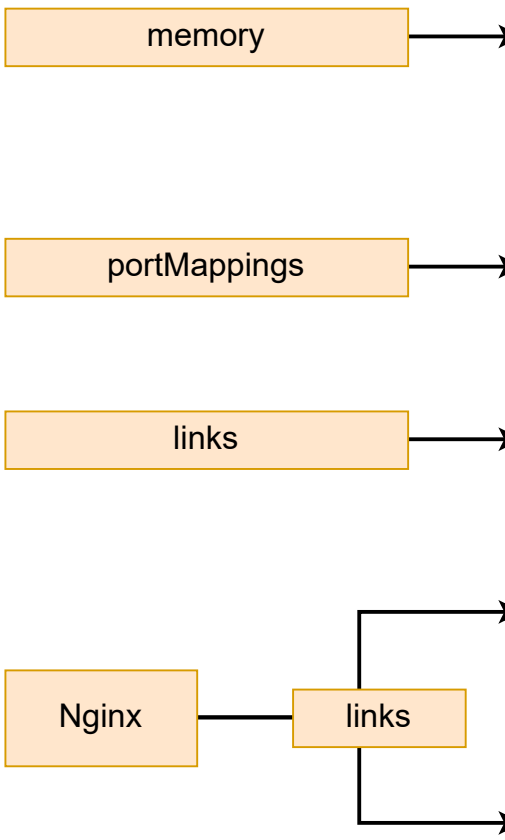
Docker 컨테이너를 구축할 온라인 Docker 리포지토리의 Docker 이미지 이름입니다.

호스트 이름입니다.
이 이름을 이용해서 도커 컴포즈를 이용해 생성된 다른 컨테이너에서 접근이 가능합니다.

컨테이너가 실패할 경우 작업을 중지해야 하면 true입니다. 필수적이지 않은 컨테이너는 인스턴스의 나머지 컨테이너에 영향을 미치지 않고 종료되거나 충돌할 수 있습니다.

컨테이너용으로 예약할 컨테이너 인스턴스에 있는 메모리 양입니다. 컨테이너 정

```
      "links": ["frontend", "backend"],
      "memory": 128
    }
  ]
}
```



의에서 memory 또는 memoryReservation 파라미터 중 하나 또는 모두에 0이 아닌 정수를 지정하면 됩니다.

컨테이너에 있는 네트워크 지점을 호스트에 있는 지점에 매핑합니다.

연결할 컨테이너의 목록입니다. 연결된 컨테이너는 서로를 검색하고 안전하게 통신할 수 있습니다.

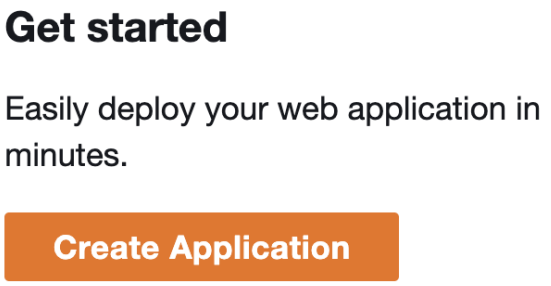
Frontend

Backend

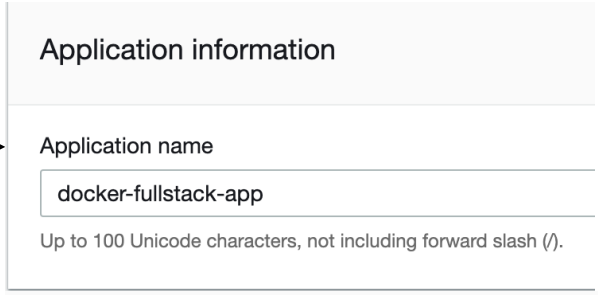
다중 컨테이너 앱을 위한 Elastic Beanstalk 환경 생성

어플리케이션 만드는 순서

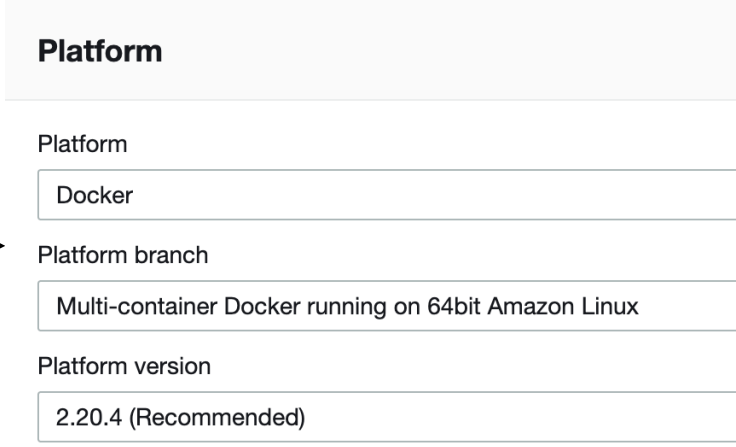
Create Application 버튼 클릭



어플리케이션 이름 정하기

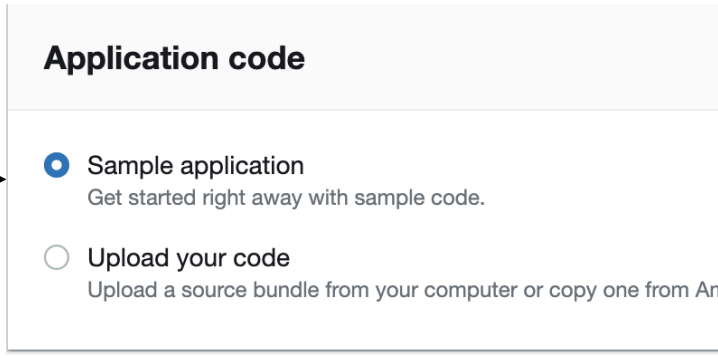


어플리케이션 플랫폼 선택



이것을 하는 과정 중에
과금이 조금 발생할 수 있습니다 ^^
;;;

애플리케이션 코드 선택

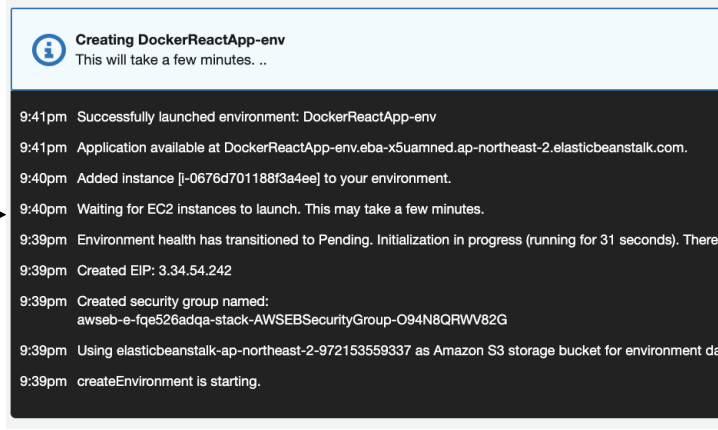


Create Application

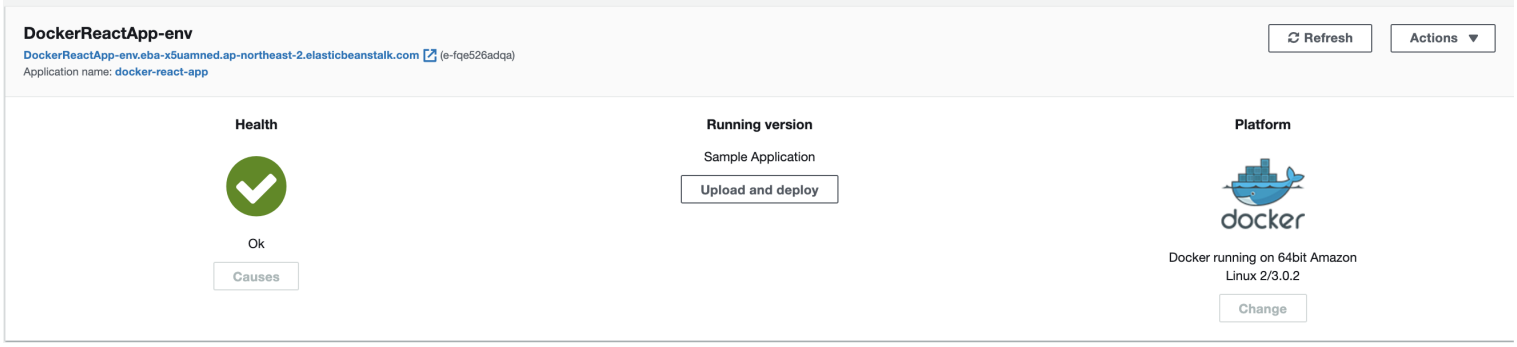
mazon S3.

Create Application
버튼 눌러서 생성하기

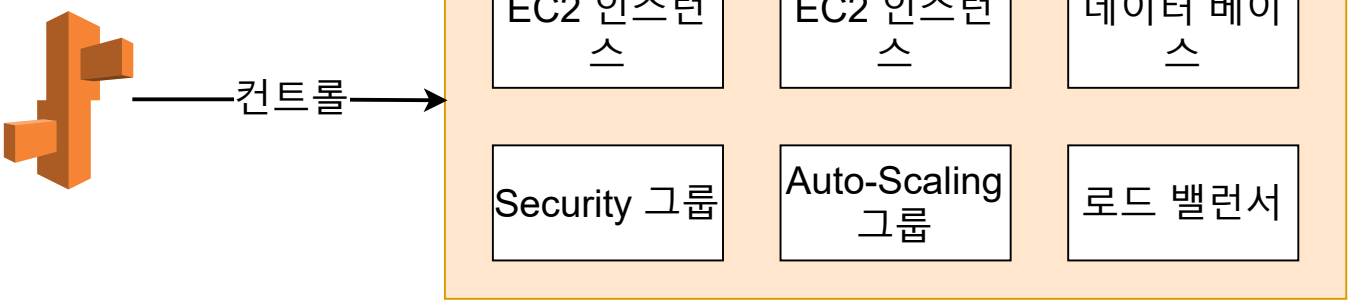
앱 생성중

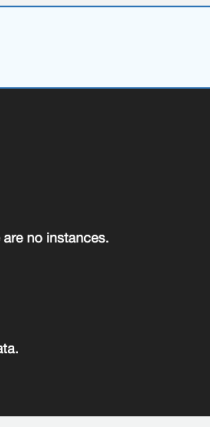


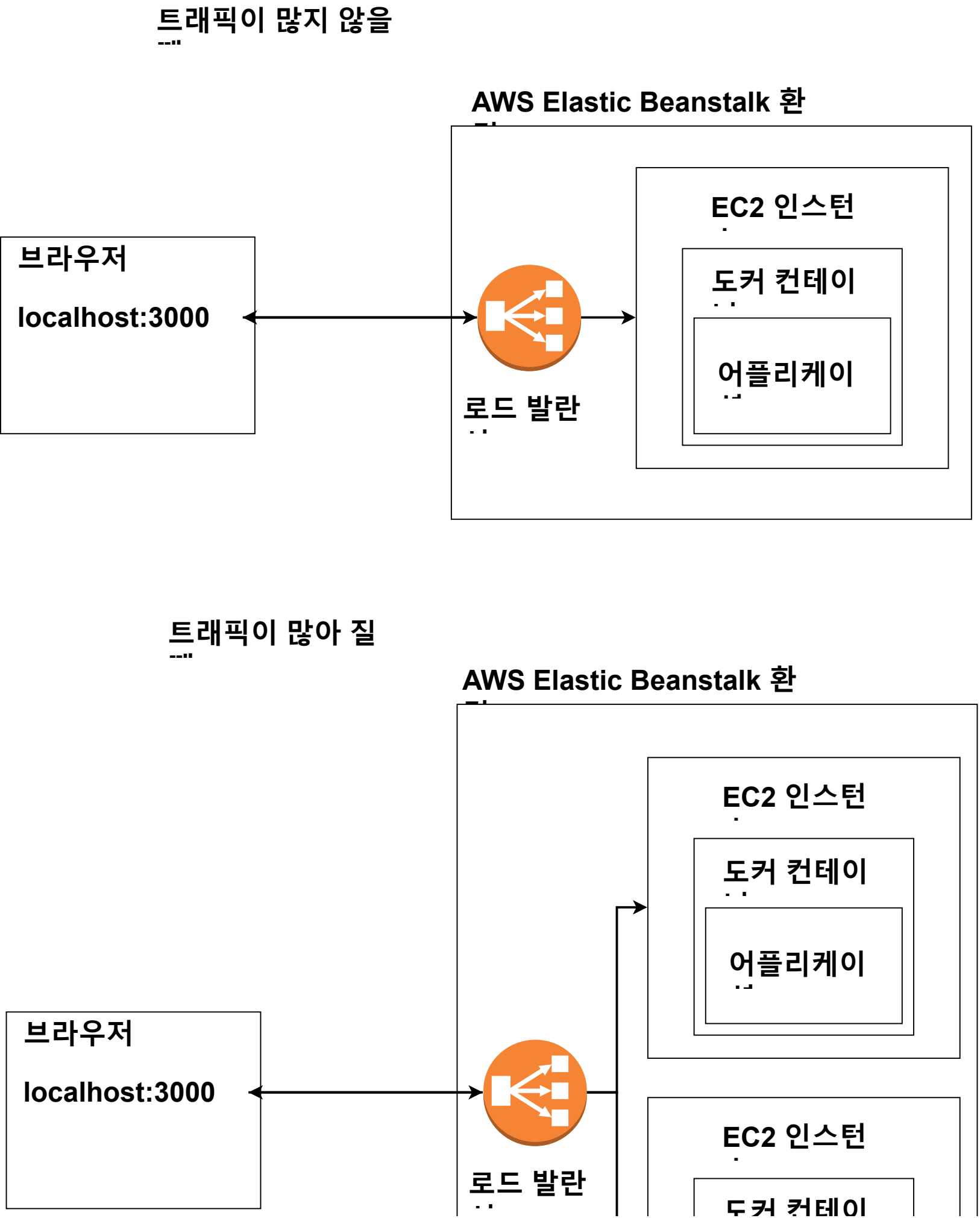
다 생성 후

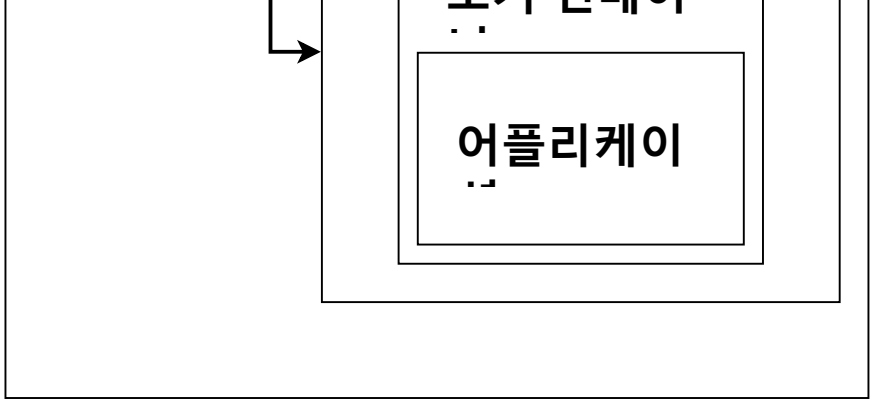


현재 상황 보고 가




are no instances.
ata.


트래픽이 많지 않을
AWS Elastic Beanstalk 환
EC2 인스턴
도커 컨테이
어플리케이
브라우저
localhost:3000
로드 발란
트래픽이 많아 질
AWS Elastic Beanstalk 환
EC2 인스턴
도커 컨테이
어플리케이
브라우저
localhost:3000
로드 발란
EC2 인스턴
도커 컨테이


어플리케이