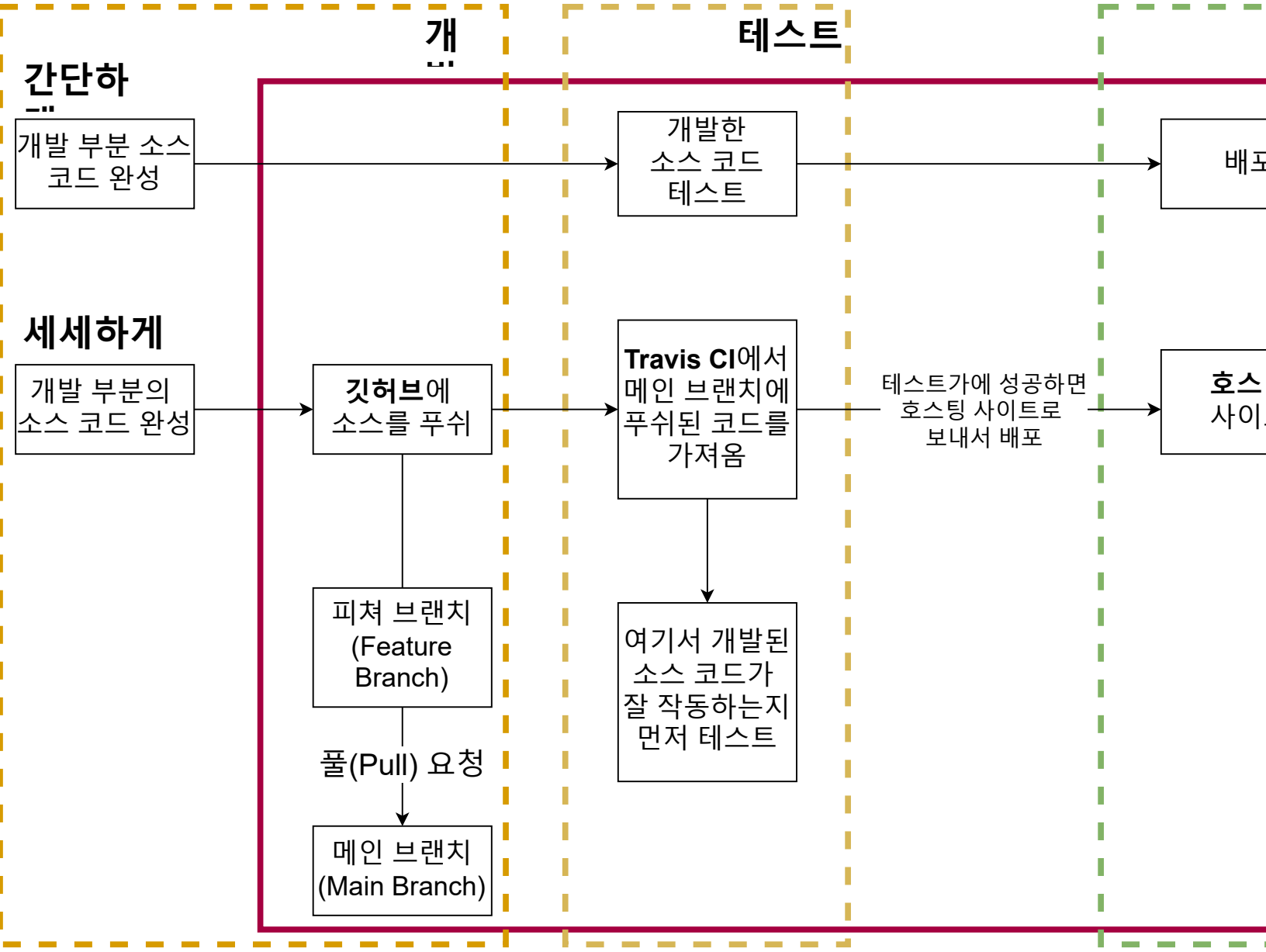
간단한 어플을 실제로 배포해보기(테스트 & 배포 부분)

이번 섹션 설명

이번 7강에서는 6강에서 생성한 소스 코드를
Github에 먼저 배포를 한 후 Travis CI라는 곳에서
소스를 가져 간 후에
그곳에서 소스 코드가 잘 돌아가는지 Test를 한 후
만약 성공을 하면 AWS에 보내서 배포까지 해보겠습니

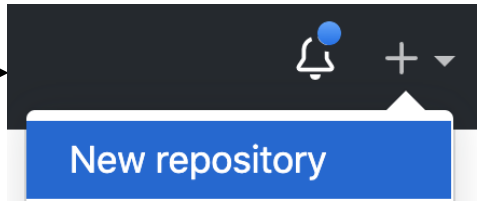


Github에 소스 코드 올리기

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 시키기 → git push origin master

이렇게 소스 코드를 github에 올렸으면
이 소스가 잘 작성된 코드인지 Travis CI라는 곳에서 확

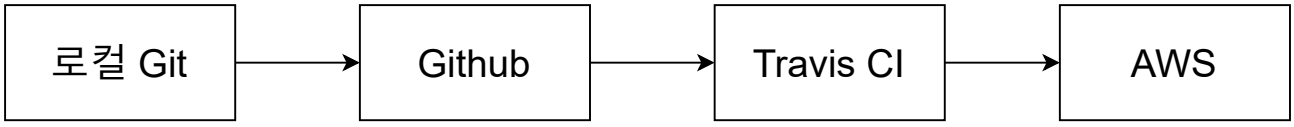
인해 주기 위해서 Travis CI에서 가져가 줘야 합니다.
그럼 다음 강에서 어떻게 그 부분을 구현하는지 알아보겠

Travis CI 설명

Travis CI 란 ?

Travis CI는 Github에서 진행되는 오픈소스 프로젝트를 위한 지속적인 통합(Continuous Integration) 서비스이다. 2011년에 설립되어 2012년에 급성장하였으며 Ruby언어만 지원하였지만 현재 대부분의 개발 언어를 지원하고 있다. Travis CI를 이용하면 Github repository에 있는 프로젝트를 특정 이벤트에 따라 자동으로 테스트, 빌드하거나 배포할 수 있다. Private repository는 유료로 일정 금액을 지불하고 사용할 수 있다.

Travis CI의 흐름



1.로컬 Git에 있는 소스를 Github 저장소에 Push를 합니다.

2.Github master 저장소에 소스가 Push가 되면
Travis CI에게 소스가 Push 되었다고 얘기를 해줍니다.

3.Travis CI는 업데이트된 소스를 Github에서 가지고 옵니다.

4.깃헙에서 가져온 소스의 테스트 코드를 실행해 봅니다.

5. 테스트 코드 실행 후 테스트가 성공하면 AWS 같은
호스팅 사이트로 보내서 배포를 합니다..

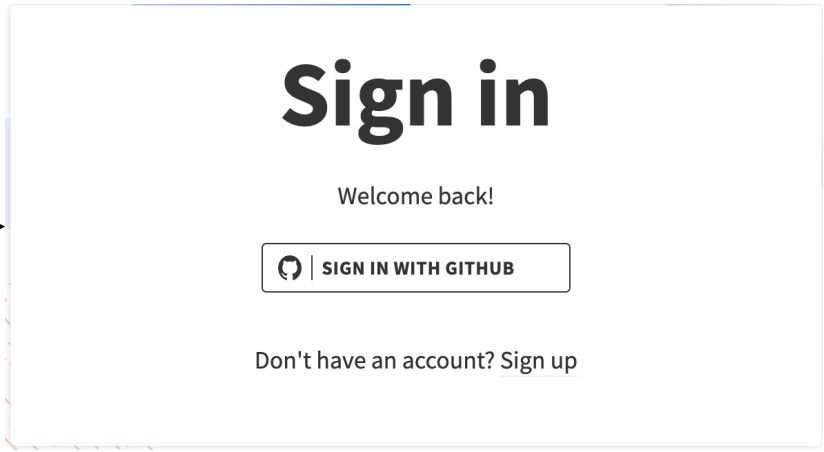
Travis CI 이용 순서

깃헙에 소스를 올렸을 때 Travis CI에서 그 소스를 가져가야 하기에 깃헙과 Travis CI가 연결되어 있어야 합니다.

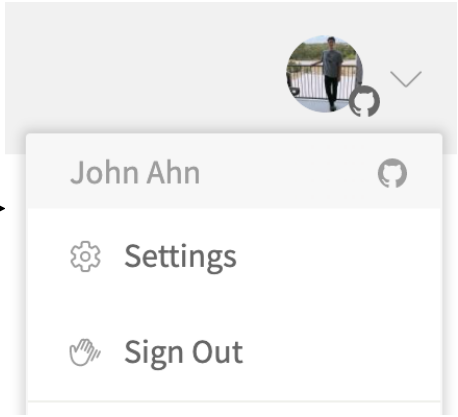
Travis CI 이용 순서 & 깃헙과 Travis CI 연결 순

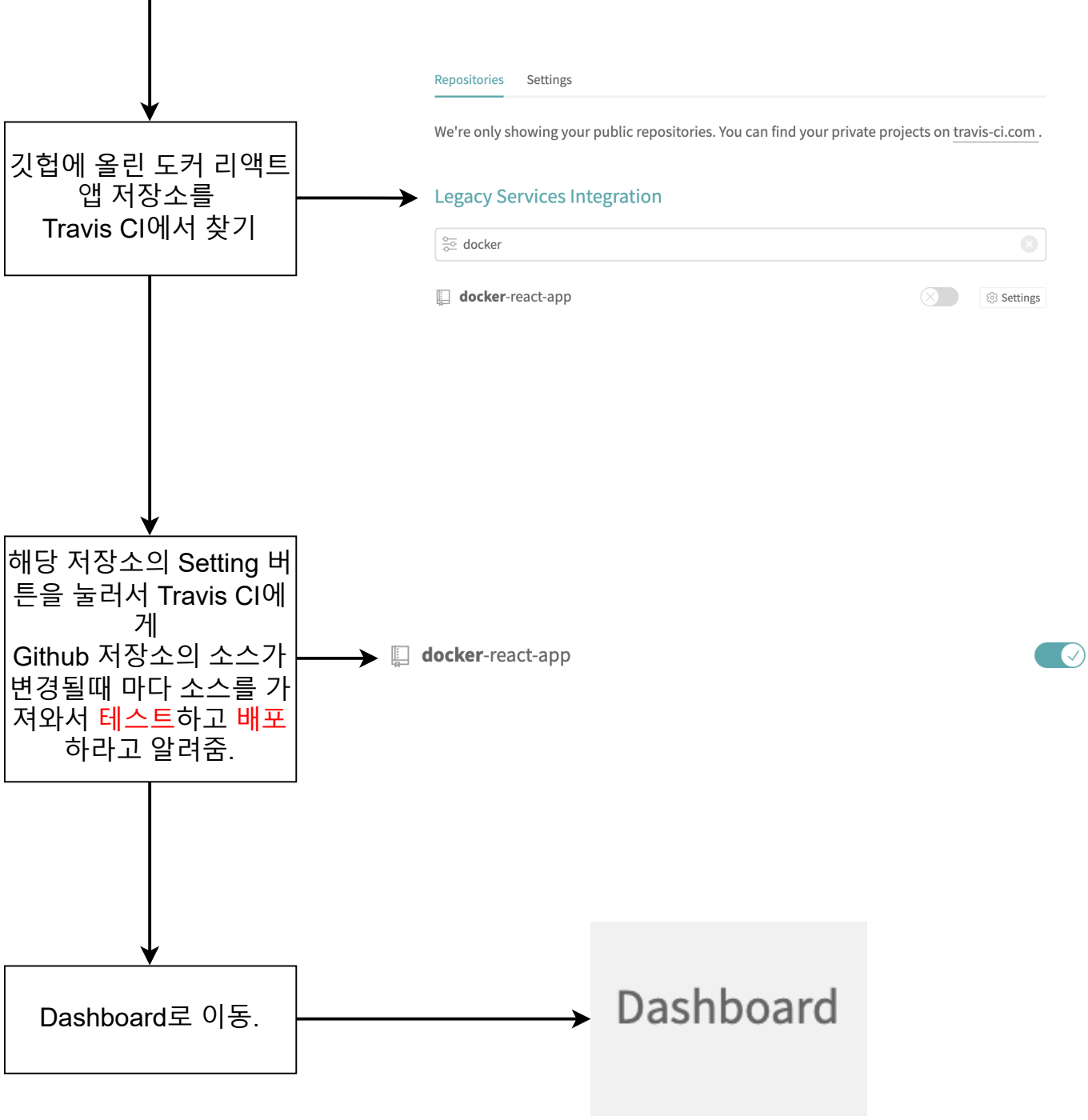
Travis CI 사이트로 이동 → https://travis-ci.org/

Travis CI에 로그인 할때 깃헙 아이디로 로그인 →



Settings 페이지로 이동 →

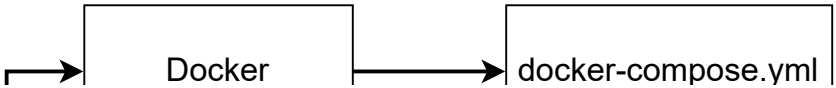


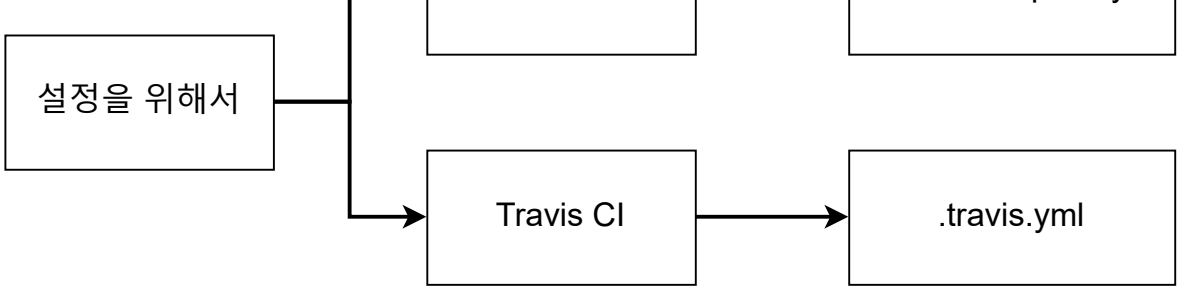


지금 까지는 Travis CI에 가입하고
어떠한 프로젝트를 Travis CI에서 관리할 것인지를 설정해주었습니다.

이제부터는 Github에서 Travis CI로 소스를 어떻게 전달시킬 거며 전달받은 것을 어떻게 Test 하며
그 테스트가 성공했을 때 어떻게 AWS에 전달해서 배포를 할 것인지를 설정해주어야 합니다.

이러한 설정을 위해서는
Docker에서는 docker-compose.yml에 무엇을 할지를 작성해줬다면
Travis CI에서는. travise.yml 파일에서 해준다.



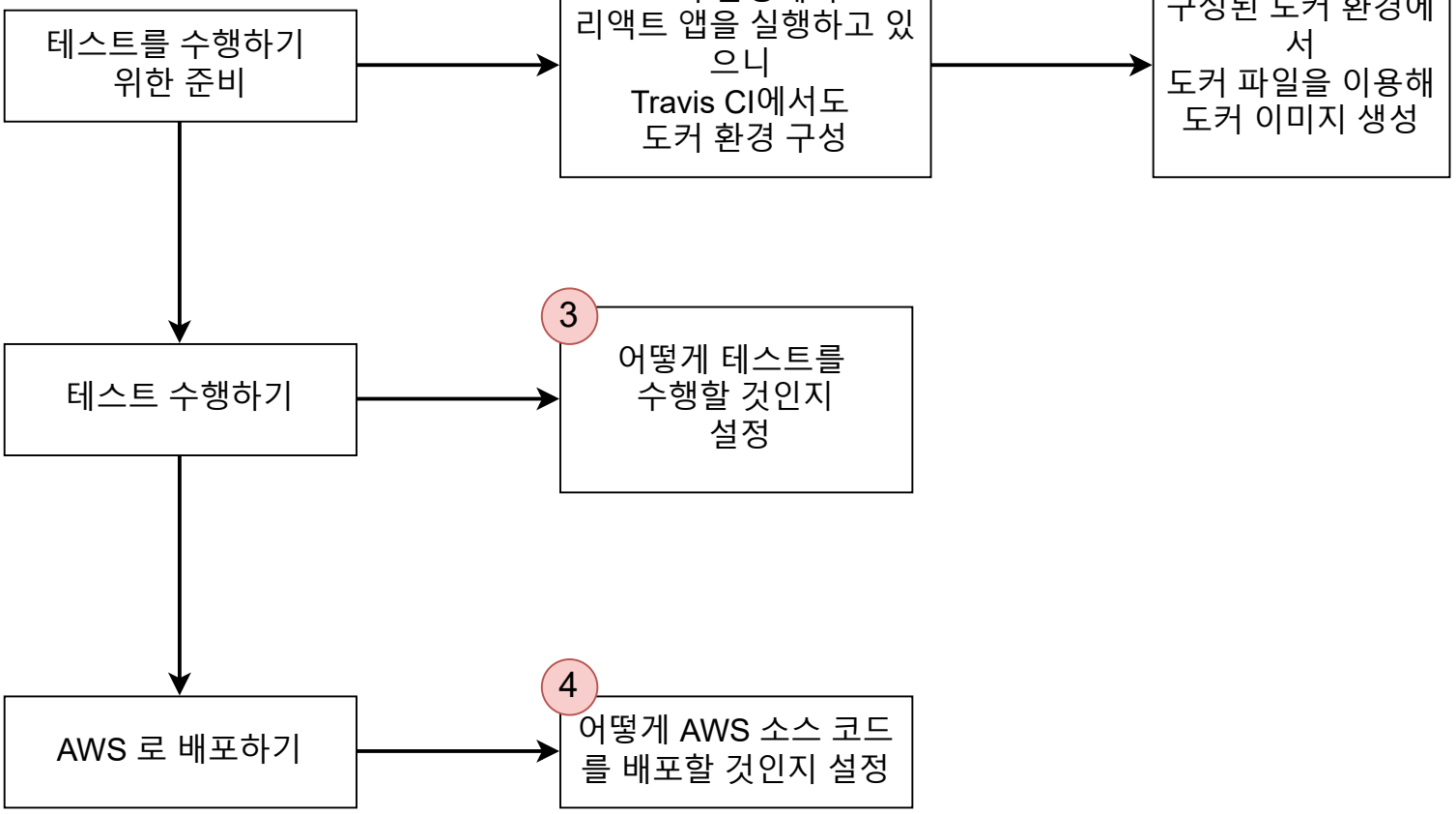

설정을 위해서
Travis CI
.travis.yml

.travis.yml 파일 작성하기 (테스트까지)

Travis CI를 이용해서 테스트 코드를 실행하고 애플리케이션을 배포까지 해주어야 하는데요. 그러기 위해서는 travis.yml파일을 작성해 주어야 합니다.

어떻게 작성하나요?

간단하게 보기



구체적으

소스로 보기

.travis.yml

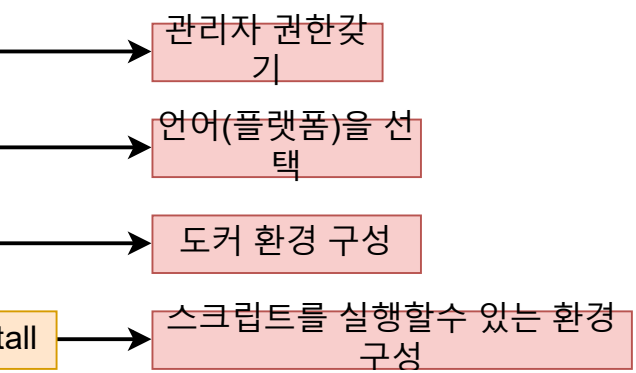
```
sudo: required

language: generic

services:
  - docker

before_install:
  - echo "start Creating an image with dockerfile"
```

sudo

language

services

before_inst


관리자 권한갖기
언어(플랫폼)을 선택
도커 환경 구성
tall
스크립트를 실행할수 있는 환경 구성

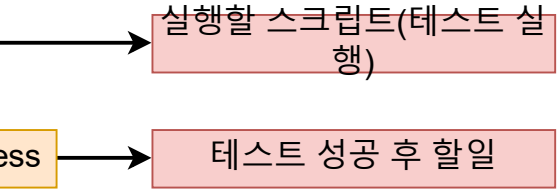
```
  - docker build -t smileajw1004/docker-react-app -f Dockerfile.dev .

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

after_success:
  - echo "Test Success"
```

script

after_succe

이제는 소스 코드를 깃헙에 다시 배포해서 Travis CI가 잘 처리해주는


```
$ sudo systemctl start docker
$ git clone --depth=50 --branch=master https://github.com/jaewonhimnae/docker-react-app.git jaewonhimnae/docker-react-app

$ bash -c 'echo $BASH_VERSION'
4.3.48(1)-release

$ echo "start Creating an image with dockerfile"
travis
$ docker build -t smileajw1004/docker-react-app -f Dockerfile.dev .
$ docker run -e CI=true smileajw1004/docker-react-app npm run test -- --coverage

> docker-lecture@0.1.0 test /usr/src/app
> react-scripts test "--coverage"

PASS src/App.test.js
  ✓ renders learn react link (32ms)
  ✓ renders learn react link (4ms)

------------------|----------|----------|----------|----------|-------------------|
File              |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
------------------|----------|----------|----------|----------|-------------------|
All files         |     2.38 |        0 |     5.88 |     2.38 |                   |
 App.js           |      100 |      100 |      100 |      100 |                   |
 index.js         |        0 |      100 |      100 |        0 |              7,17 |
 serviceWorker.js |        0 |        0 |        0 |        0 |... 32,133,135,138 |
------------------|----------|----------|----------|----------|-------------------|
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.375s
Ran all test suites.
The command "docker run -e CI=true smileajw1004/docker-react-app npm run test -- --coverage" exited with 0.

$ echo "Test Success"

Done. Your build exited with 0.
```


실행할 스크립트(테스트 실행)
ess
테스트 성공 후 할일

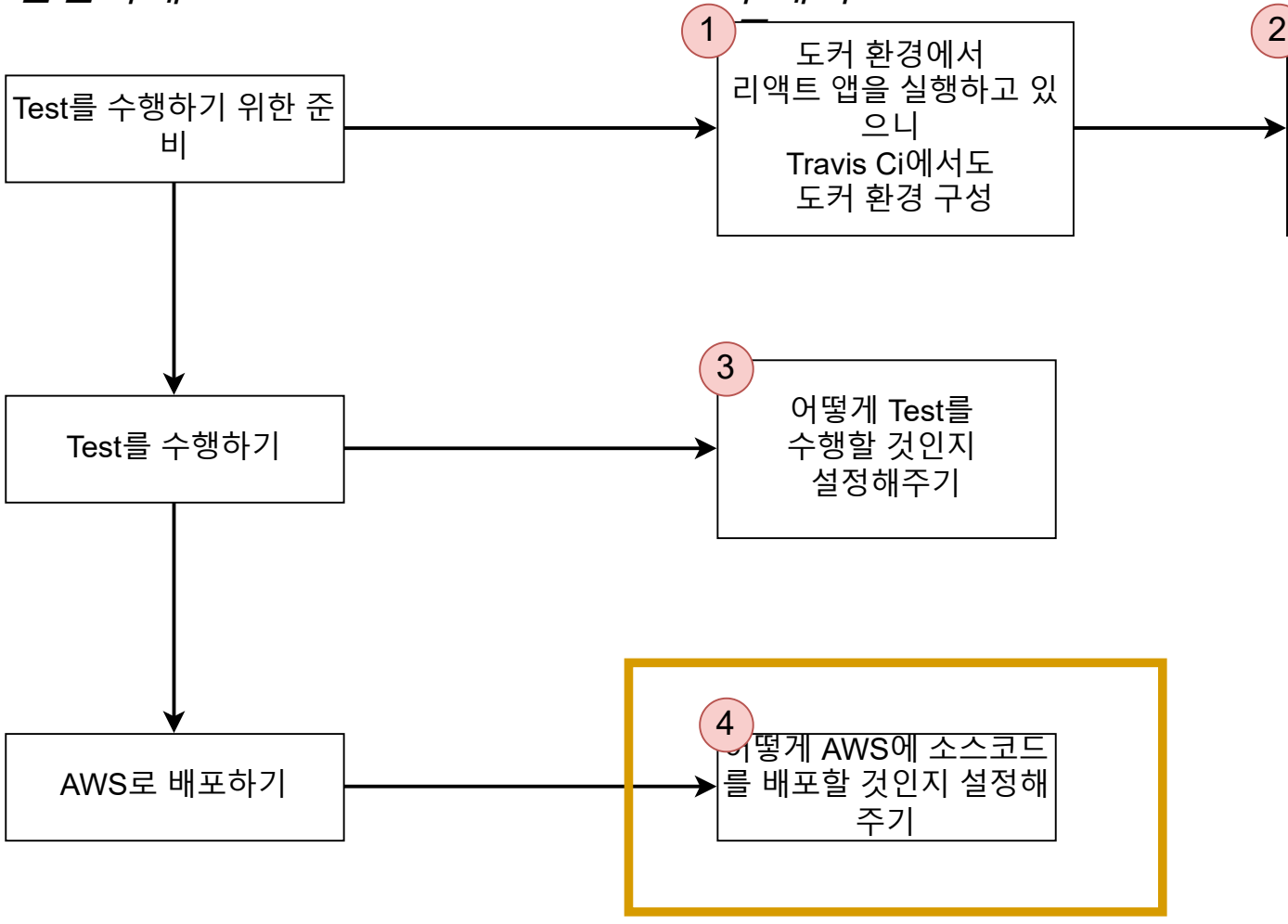
테스트까지 성공 이제는 AWS에 배포할 차례 !!!

AWS 알아보기

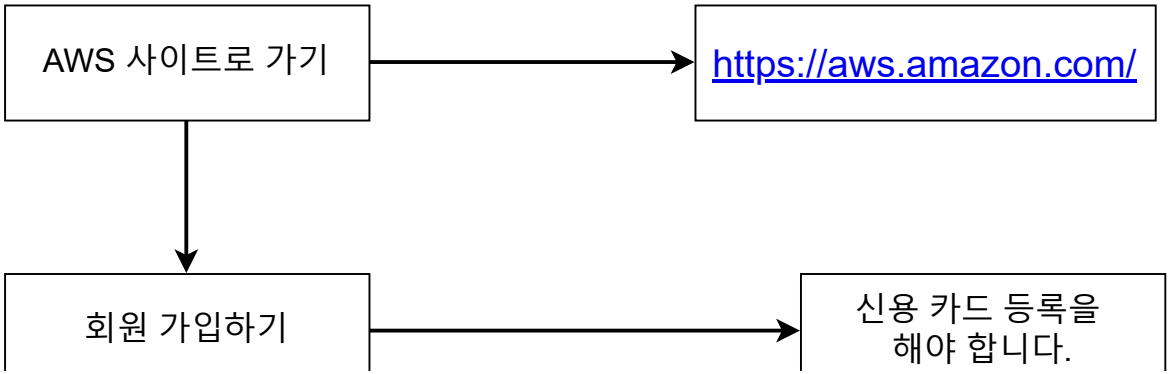
어떻게 작성하나요?

간단하게

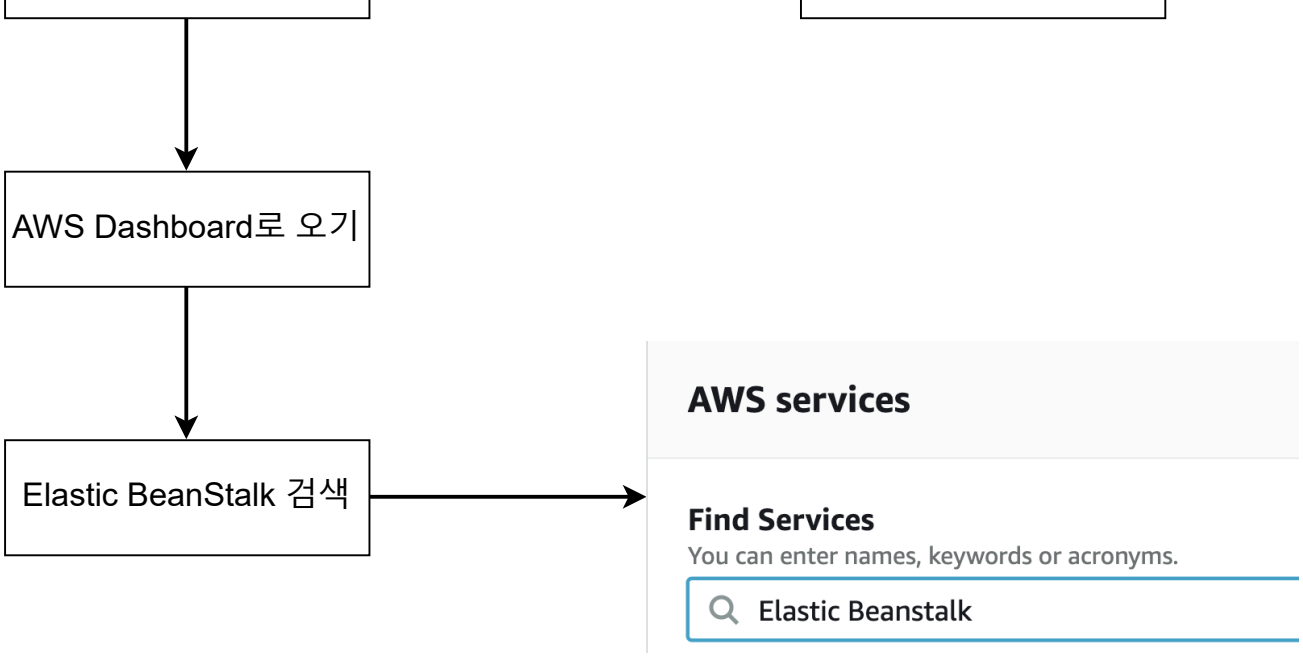
구체적으



AWS로 배포 하는 순서



구성된 도커 환경에서 Dockerfile.dev를 이용해서
도커 이미지 생성



AWS 서비스들에 대한 간략한 설명들

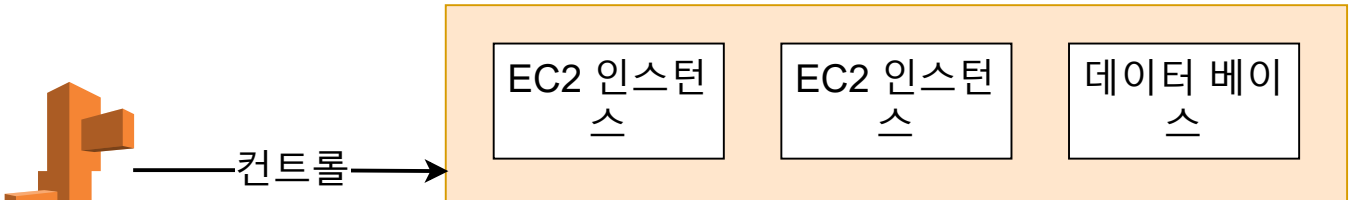
EC2란 무엇인가? (Elastic Compute Cloud)

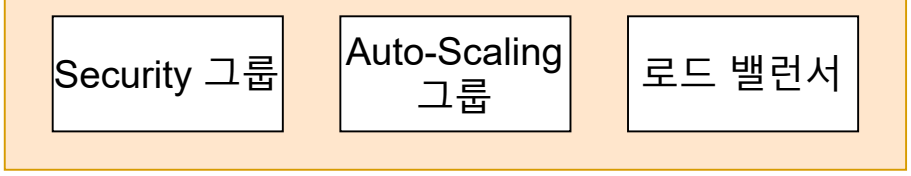
Amazon Elastic Compute Cloud(Amazon EC2)는 Amazon Web Services(AWS) 클라우드에서 확장식 컴퓨팅을 제공합니다. Amazon EC2를 사용하면 하드웨어에 선투자할 필요가 없어 더 빠르게 애플리케이션을 개발하고 배포할 수 있습니다. Amazon EC2를 통해 원하는 만큼 가상 서버를 구축하고 보안 및 네트워크 구성과 스토리지 관리가 가능합니다. 또한 Amazon EC2는 요구 사항이나 갑작스러운 인기 증대 등 변동 사항에 따라 신속하게 규모를 확장하거나 축소할 수 있어 서버 트래픽 예측 필요성이 줄어듭니다.

=> 한대의 컴퓨터를 임대한다고 생각하면 됩니다.
그리고 그 컴퓨터에 OS를 설치하고 웹서비스를 위한 프로그램들(웹서버, DB)을 설치해서 사용 하면 됩니다.
1대의 컴퓨터를 회사의 EC2 인스턴스라고 부릅니다

EB란 무엇인가 ? (Elastic BeanStalk)

AWS Elastic Beanstalk는 Apache, Nginx 같은 친숙한 서버에서 Java, NET, PHP, Node.js, Python, Ruby, Go 및 Docker와 함께 개발된 웹 응용 프로그램 및 서비스를 배포하고 확장하기 쉬운 서비스입니다.
아래서 보는 도표와 같이 Elastic Beanstalk은 EC2 인스턴스나 데이터베이스 같이 많은 것들을 포함한 "환경"을 구성하며 만들고 있는 소프트웨어를 업데이트할 때마다 자동으로 이 환경을 관리해줍니다.



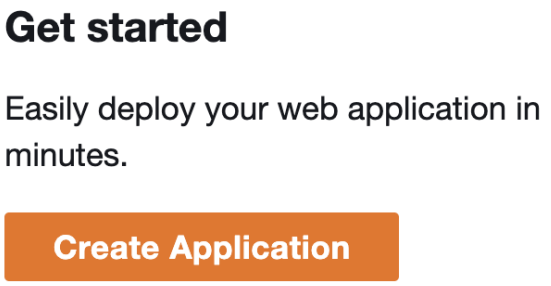


저희의 리액트 앱을 배포할 때 Elastic BeanStalk을 사용하겠습니다.

Elastic Beanstalk에서 애플리케이션 만들기

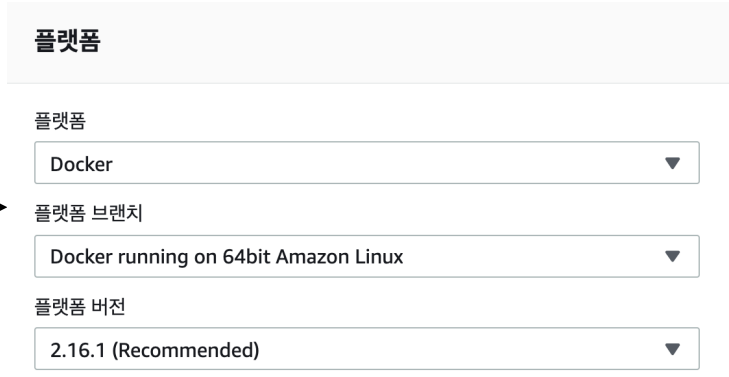
새로운 Elastic Beanstalk 환경 만드는 순서

Create Application 버튼 클릭

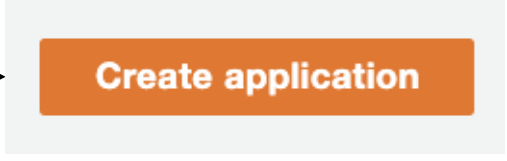


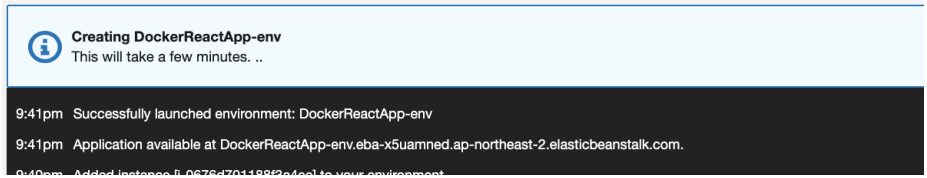
어플리케이션 이름 정하기

어플리케이션 플랫폼 선택



Create Application 버튼 눌러서 생성하기



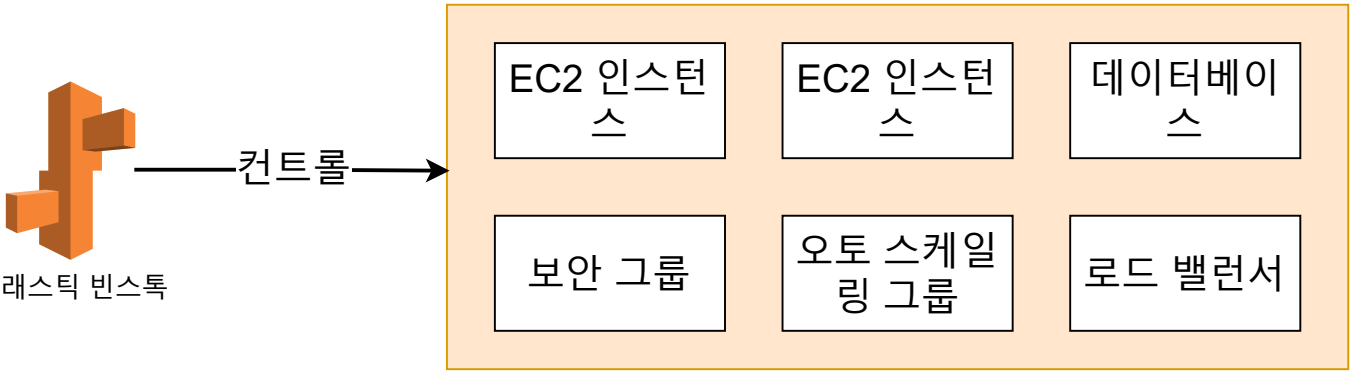


앱 생성중

```
9:40pm  Added instance [i-0678d7011881f3a48e] to your environment.
9:40pm  Waiting for EC2 instances to launch. This may take a few minutes.
9:39pm  Environment health has transitioned to Pending. Initialization in progress (running for 31 seconds). There are no instances.
9:39pm  Created EIP: 3.34.54.242
9:39pm  Created security group named:
        awseb-e-fqe526adqa-stack-AWSEBSecurityGroup-O94N8QRWV82G
9:39pm  Using elasticbeanstalk-ap-northeast-2-972153559337 as Amazon S3 storage bucket for environment data.
9:39pm  createEnvironment is starting.
```

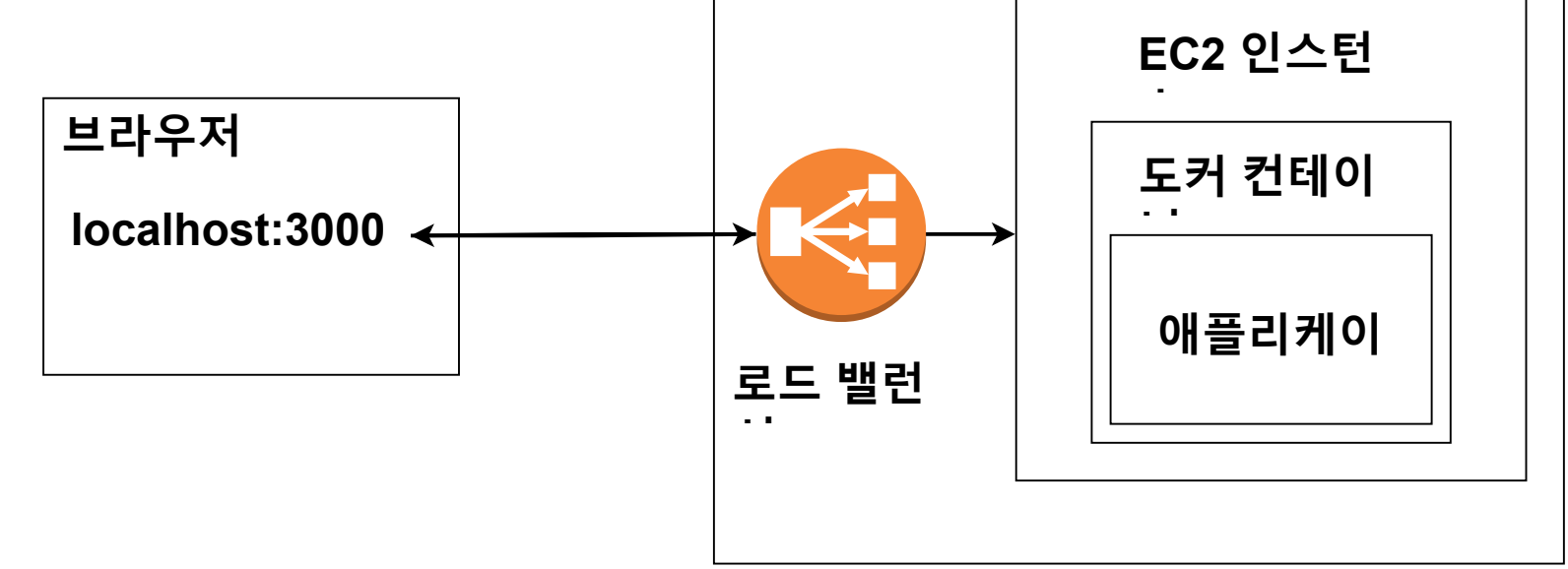
다 생성 후

현재 상황 보고 가

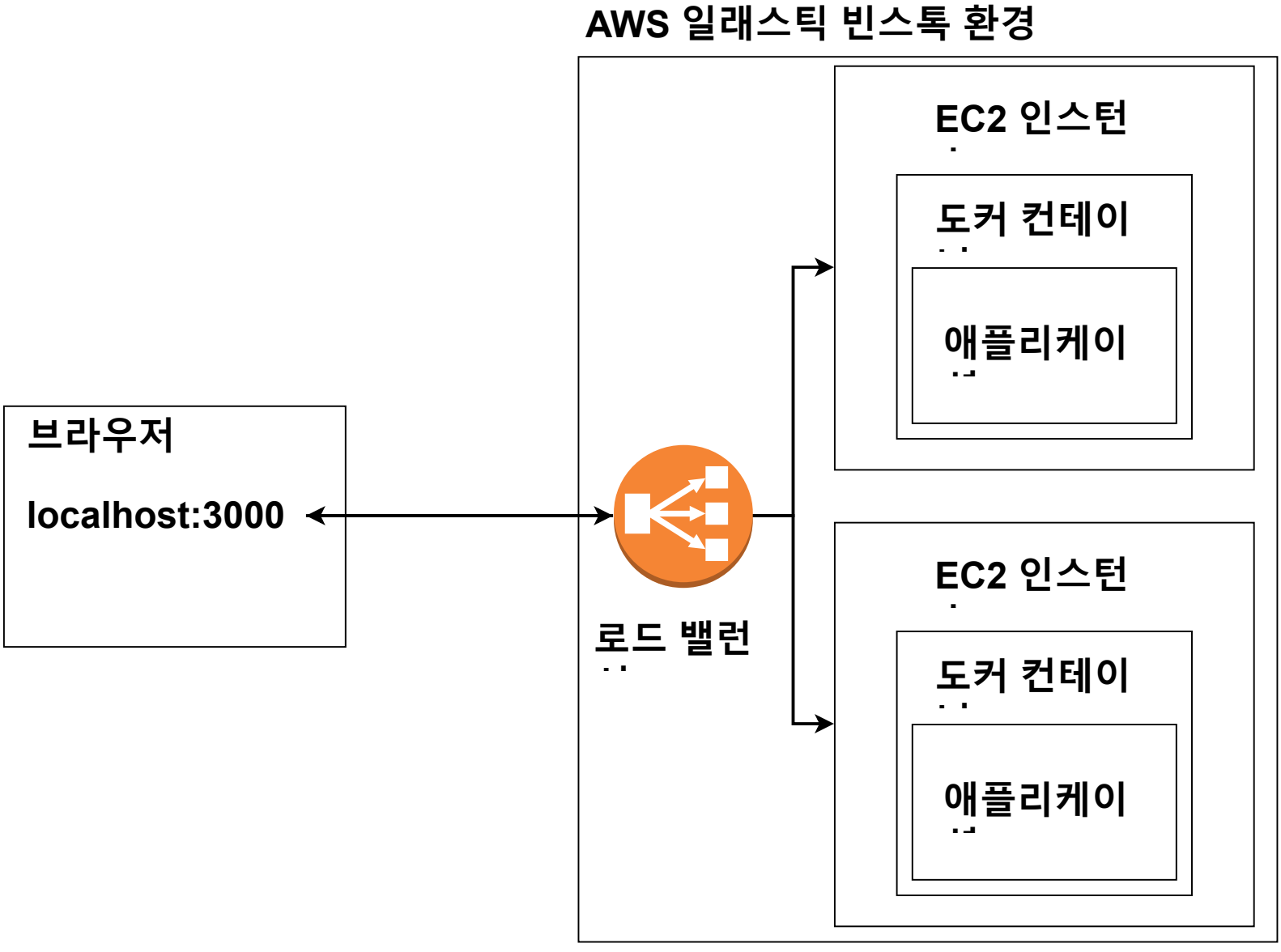


트래픽이 많지 않

AWS 일래스틱 빈스톡 환경


브라우저
localhost:3000
로드 밸런
EC2 인스턴
도커 컨테이
애플리케이

트래픽이 많아 질때


AWS 일래스틱 빈스톡 환경
브라우저
localhost:3000
로드 밸런
EC2 인스턴
도커 컨테이
애플리케이
EC2 인스턴
도커 컨테이
애플리케이

.travis.yml 파일 작성하기 (배포 부분)

현재는 도커 이미지를 생성 후 어플을 실행하여 테스트 하는 부분까지 travis 설정을 하였습니다.

이제는 테스트에 성공한 소스를 AWS Elastic Beanstalk에 자동으로 배포하는 부분을 travis 파일에 넣어줄 차례입니다.

현재까지의 travis 설정

```
sudo: required

language: generic

services:
  - docker

before_install:
  - echo "start Creating an image with dockerfile"
  - docker build -t smileajw1004/docker-react-app -f Dockerfile.dev .

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

after_success:
  - echo "Test Success"
```

배포 부분 추가 된

```
sudo: required

language: generic

services:
  - docker

before_install:
  - echo "start Creating an image with dockerfile"
  - docker build -t smileajw1004/docker-react-app -f Dockerfile.dev .

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

deploy:
```

```
provider: elasticbeanstalk
region: "ap-northeast-2"
app: "docker-react-app"
env: "DockerReactApp-env"
bucket_name: "elasticbeanstalk-ap-northeast-2-972153559337"
bucket_path: "docker-react-app"
on:
  branch: master
```

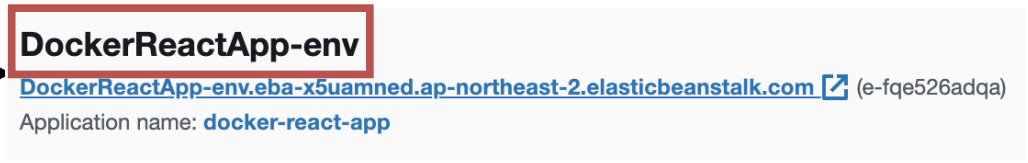
deploy 새롭게 추가된 부분 설

provider → 외부 서비스 표시 (s3, elasticbeanstalk, firebase 등)

region → 현재 사용하고 있는 AWS 의 물리적 장소

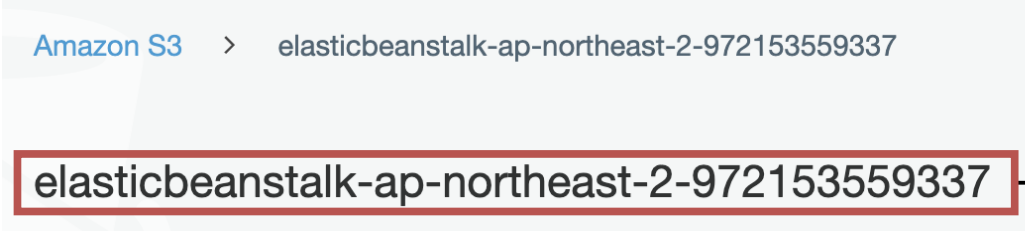
app → 생성한 애플리케이션의 이름

env →



bucket_name → 해당 일래스틱 빈스톡을 위한 s3 버킷 이름

Travis CI → travis 에서 가지고 있는 파일을 압축해서 S3에 보낸다. → S3



bucket_path → 애플리케이션의 이름과 동일

on

branch → 어떤 브랜치에 푸시할 때 AWS에 배포할 것인지 설정

이렇게 대부분의 설정을 끝내었다.

하지만 이렇게 아무런 인증 없이는 Travis CI에서 마음대로 AWS에 파일을 전송할 수는 없다. 그래서 이제는 Travis CI가 AWS에 접근할 수 있게 해 주는 방법을 알

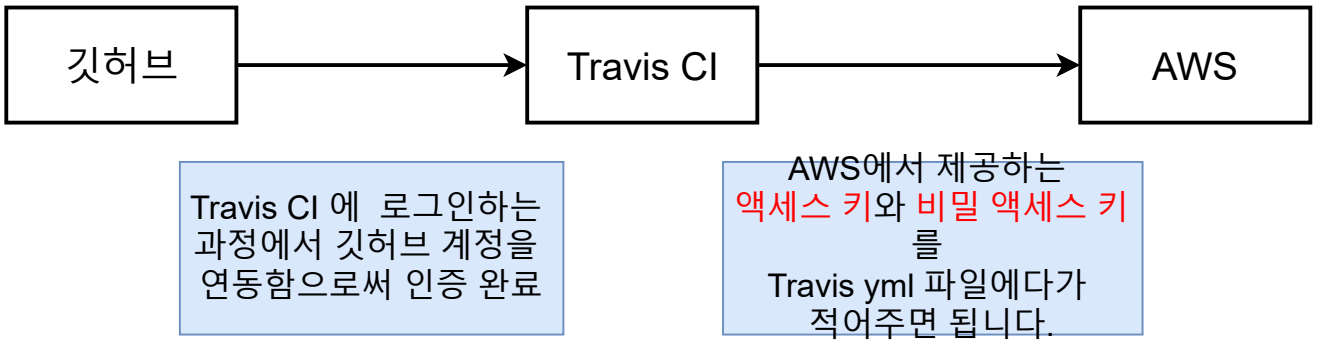
Travis CI의 AWS접근을 위한 API 생성

현재까지 Travis CI에서 AWS에 어떤 파일을 전해줄 거며, AWS에서 어떤 서비스를 이용할 거며, 부수적인 설정들을 적어주었습니다.

하지만 Travis CI와 AWS가 실질적으로 소통을 할 수 있게 인증하는 부분을 설정해주진 않았습니다.

그래서 그 인증하는 부분을 살펴보겠습니다.

소스 파일을 전달하기 위한 접근



인증을 위해서는 API Key가 필요합니다.
그래서 그 API Key를 어떻게 받는지 알아보겠습

Secret, Access API Key 받는 순

1. IAM USER 생성

IAM은 무엇인가 ? (Identity and Access M

AWS 리소스에 대한 액세스를 안전하게 제어할 수 있는 웹 서비
IAM을 사용하여 리소스를 사용하도록 인증(로그인) 및 권한 부
제어합니다.

Management)

스입니다.

여(권한 있음)된 대상을

루트(Root) 사용자

현재 우리가 가입하여
사용하고 있는
AWS 계정

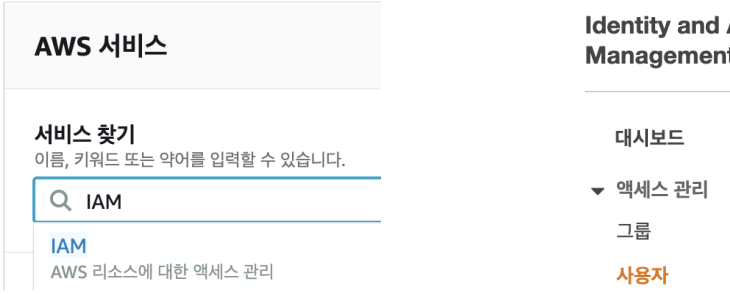
AWS 의
서비스 및 리소스에 대한 모든 접근 권한 이 있음.

일상적인 작업이든 관리 작업이든
루트 사용자를 사용하는 방법은
보안 문제로 바람직하지 않습니다.
그래서 IAM 유저를 사용합니다.

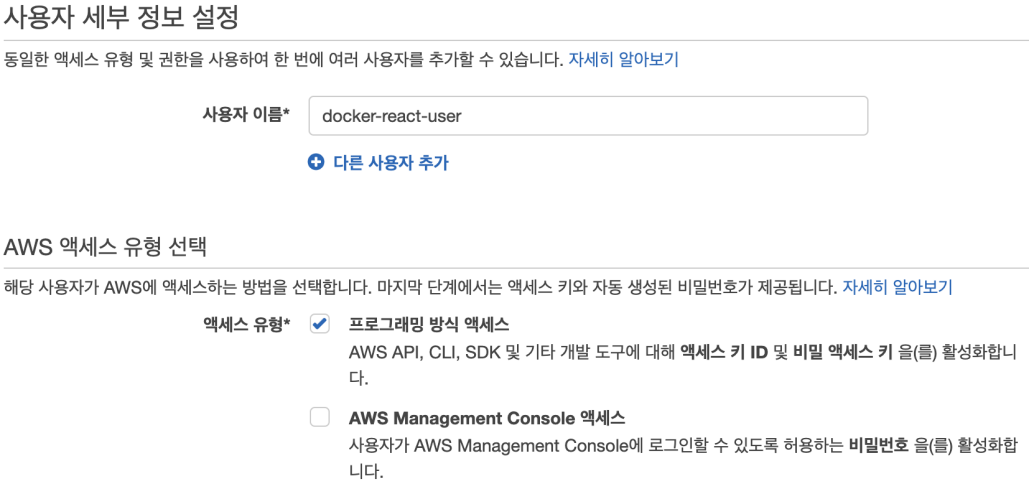
IAM 사용자

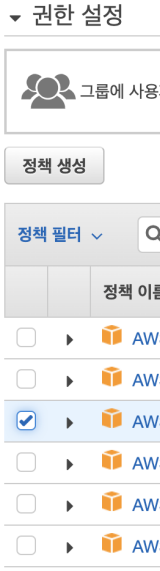
루트 사용자
가
부여한 권한
만
가지고 있음

Dashboard → IAM 검색 → 사용자



사용자 세부 정보 설정 → 사용자





원하시는 사용자 이름과 액세스 유형을 선택해

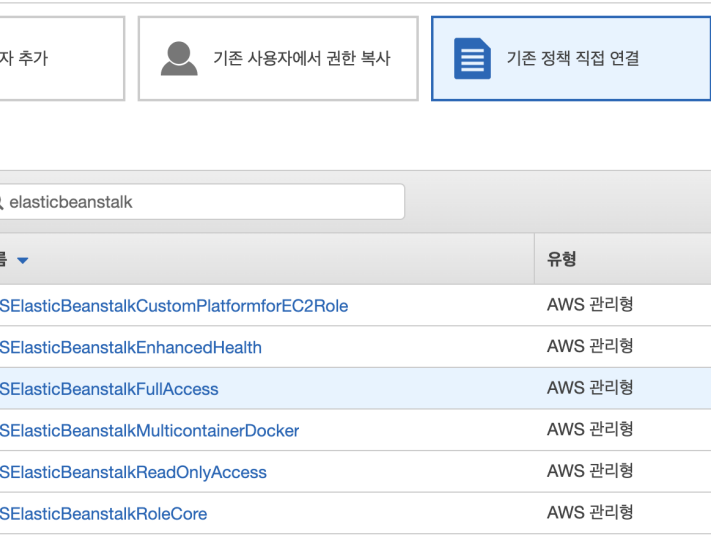
해당 IAM
다.
현재는 T
을 하게 하

자 클릭 → 사용자 추가 클릭

Access
t(IAM)

사용자 추가 | 사용자 삭제

권한 설정 → 계정 생성 성공



M유저에게 필요한 권한을 주어야 합니

Travis CI가 Elastic Beanstalk에 접근
해주어야 하니 Elastic Beanstalk을 검

이렇게 계정 만들기를 성공하면, 액세스
있습니다.

이 API 키들은 이번 한 번만 받을 수 있고

수 있습니다. AWS Management Console 로그인을 위한 사용자 지침을
기회입니다. 하지만 언제든지 새 자격 증명을 생성할 수 있습니다.

59337.signin.aws.amazon.com/console에 로그인할 수 있습니다.

	액세스 키 ID	비밀 액세스 키
	AKIA6EWG6DEUV6SB4VCK	********* 표시

키와 비밀 액세스 키를 받을 수

고 잊어버리면

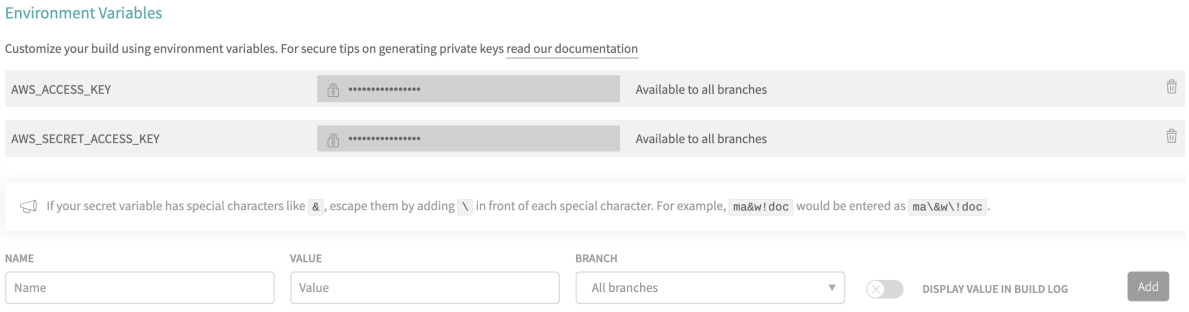
2

API키를
Travis yml 파일에
적어주기

직접 API 키를 Travis yml 파일에 적어 주면 노출이 되기 때문에 다른 곳에 적고 그것을 가져와줘야 한다.

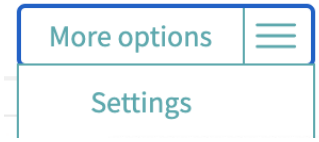
Travis 웹사이트 해당 저장소 대시보드에 오기

사진과 같이 AWS에서 받은 API 키들을 NAME과 VALUE에 적어서 넣어줍니다.
이곳에 넣어 주면 외부에서 접근을 할 수 없어 더욱 안전합니다.



이렇게 Travics CI 가 AWS에 접근할 수 있게 만들어 주었습니다

설정 클릭



Travis CI 웹사이트에서 보관 중인 Key를
로컬 환경에서 가지고 올 수 있게 travis yml 파일에서
설정을 해줍니다.

```
deploy:
  provider: elasticbeanstalk
  region: "ap-northeast-2"
  app: "docker-react-app"
  env: "DockerReactApp-env"
  bucket_name: "elasticbeanstalk-ap-northeast-2-972153559337"
  bucket_path: "docker-react-app"
  on:
    branch: master
access_key_id: $AWS_ACCESS_KEY
secret_access_key: $AWS_SECRET_ACCESS_KEY
```

다

을 합쳐야 합니다

이제는 Github에 Master branch에 소스를 Push 했을 때

자동으로 AWS에 배포할 수 있게 하는 것에 있어서 마지막 포트

```
FROM node:alpine as builder
WORKDIR '/usr/src/app'
COPY package.json .
RUN npm install
COPY ./ ./
RUN npm run build

FROM nginx
EXPOSE 80
COPY --from=builder /usr/src/app/build /usr/share/nginx/html
```

Dockerfile에 EXPOSE 80을 넣어줘서 포트 맵핑 문제를 해결

이제 Github에 마스터 브랜치에 푸시를 해보자.

모든 프로세스가 한 번에 돌아가고 앱이 배포가 될 것입니다.

맵핑이 남

할 수가