React + API Server 프로젝트 개발과 배포

Intro

기존의 웹개발과 달라진 싱글 페이지 애플리케이션(SPA)의 개발 환경과, 서버에 배포해서 서비스하는 방법을 소개한다. API 문서 자동화를 위해서 Swagger 기술을 포함한다. 로컬 개발 환경과 배포되는 서버 환경 차이를 설명한다. 리액트와 node.js의 문법은 다루지 않고, 개발환경 구축과 배포에 대하여 설명한다. 다루는 내용과 순서는 다음과 같다.

* 준비하기
  * 이 과정에 필요한 개발 플랫폼과 개발 도구, 그리고 관련된 계정 등록을 소개한다.
* SPA 개발 환경 구성
  * 로컬 컴퓨터에서 작업하기 위한 프론트엔드와 백엔드의 기본적인 프로젝트 환경을 구성한다.
* 배포 프로세스
  * 서비스를 위해 서버에 배포하는 절차와 방법을 소개한다.
* 도메인 등록
  * 도메인과 서버를 연결하고, HTTPS 연결을 위한 방법을 소개한다.
* 배포 자동화하기
  * 코드가 변경되면 자동으로 개발 서버에 배포되는 작업을 설명한다.
* CORS 설정
  * 로컬 개발 환경에서 발생하는 CORS 이슈와 빌드 환경 변수를 구성한다.

준비하기

* Node.js: JS 플랫폼 https://nodejs.org
* Git: 버전 관리 도구 https://git-scm.com
* VSCode 개발 도구: 플러그인 생태계가 막강한 개발 도구 https://code.visualstudio.com
* GitHub 계정: Git 원격 저장소 https://github.com
* AWS 계정: 클라우드 서비스 https://aws.amazon.com/ko/

Node.js 플랫폼

자바스크립트 언어의 플랫폼인 Node.js는 2009년 시작되어 현재 가장 큰 에코시스템을 갖추고 있다. 즉 관련된 라이브러리가 가장 많다는 의미이다. 브라우저를 벗어나 명령어, 서버 역할을 수행할 수 있는 플랫폼이다. React, Vue, Svelte 같은 프론트엔드 개발환경도 Node.js 기반으로 구성된다.
https://nodejs.org 에서 LTS(Long Term Support) 버전을 다운받아서 설치한다.
터미널을 열어서 `node -v`, `npm -v`로 설치된 버전을 확인할 수 있다.
참고: https://okdevtv.com/mib/nodejs

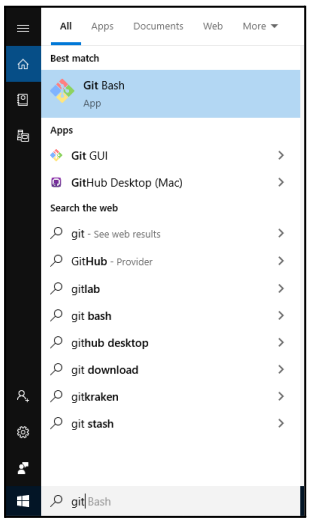
Git 버전관리 도구

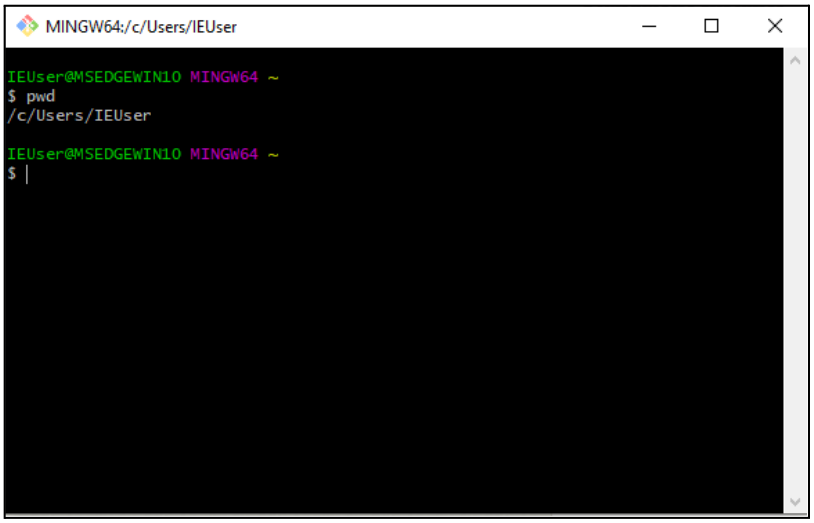
프로젝트 소스의 버전관리를 할 수 있는 도구이다. https://git-scm.com 에서 다운로드 받아서 설치한다. 리눅스를 만든 Linus Torvaldz가 만든 오픈 소스 제품이며, 윈도우의 경우 git bash 프로그램을 실행하면 리눅스 명령어를 대부분 사용할 수 있다.
제일 처음 코드 등록(commit)을 위한 사용자 등록이 필요하며 다음 명령어를 실행한다.
밑줄친 부분은 자신의 영문 정보로 대체해서 실행한다.

```
git config --global user.name kenu
git config --global user.email email_id@gmail.com
```

참고: https://okdevtv.com/mib/git

Windows Git Bash
윈도우 사용자의 경우 Git Bash 창에서 앞으로 나오는 리눅스 명령을 실행할 수 있다.





VSCode 개발 도구

마이크로소프트에서 잘 만들고 있는 오픈 소스 개발도구이며 플러그인이 다양하게 제공되는 생태계를 가지고 있다. https://code.visualstudio.com 에서 다운로드하고 설치한다.
추천하는 플러그인 GitLes, Git Graph, Thunder Client 세 가지이다. 왼쪽 사이드바의 `확장`아이콘을 클릭하면 플러스인을 검색해서 설치할 수 있다.
참고: https://okdevtv.com/mib/vscode

GitHub 계정

GitHub은 원격 저장소 역할과 동시에 프로젝트 관리 기능을 지원한다. https://github.com/ 사이트에 무료로 가입할 수 있으며, 오픈 소스는 무제한으로 프로젝트를 만들 수 있고, 비공개 프로젝트도 무제한으로 무료로 만들 수 있다.

AWS 계정

아마존 웹 서비스(AWS)는 퍼블릭 클라우드 서비스로 가장 많이 이용되고 있다. https://aws.amazon.com/ko/ 에서 계정을 만들 수 있는데, 신용카드를 등록하는 절차가 있다. 대표적인 서비스로는 EC2(Elastic Cloud Compute)로 사용시간에 따라 비용을 내는 서버 임대와 S3(Simple Storage Service) 같은 파일 저장소 기능이 있다.
참고: https://okdevtv.com/mib/aws

SPA 개발 환경 구성

SPA(Single Page Application)은 리액트, 뷰 이후로 각광을 받는 개발방식이다. RESTful 형식의 API로 데이터를 받아와서 화면에 그려주는 자바스크립트 앱이라고 할 수 있다. API를 담당하는 백엔드와 React로 만들어지는 프론트엔드의 로컬 개발환경을 처음부터 구축해 본다.

* 프론트엔드 프로젝트 생성
* 백엔드 프로젝트 생성
* 백엔드 포트 4000 설정
* 백엔드 nodemon 설정
* Swagger 추가
* API 샘플 추가
* GitHub 프로젝트 생성
* 원격에 프로젝트 전송

프론트엔드 프로젝트 생성

프론트엔드의 대표적인 기술인 React는 관례적으로 CRA(https://create-react-app.dev/)을 통해서 프로젝트를 생성한다. 하지만 이번에는 CRA가 사용하는 번들러인 Webpack의 중복된 부분을 개선한 Rollup 번들러 기반의 Vite.js(https://vitejs.dev/)를 통해서 프로젝트를 생성하려고 한다.
다음과 같이 frontend 프로젝트를 react 로 생성한다. npm install 명령으로 모듈을 설치하고 npm run dev 명령으로 개발 모드 서버를 실행한다. 브라우저에서 http://localhost:3000 주소를 통해 화면을 확인할 수 있다.

```
# 작업 폴더 생성 mac, linux, windows git bash
mkdir ~/git
cd ~/git

# 프로젝트 폴더 생성
mkdir backfront
cd backfront
git init
git checkout -b main
npm create vite frontend -- --template react
cd frontend
npm install
npm run dev
```

백엔드 프로젝트 생성

로컬 컴퓨터에 작업 폴더를 만들고, node.js 기반의 웹 프로젝트를 만든다. express.js 웹 프레임워크가 가장 많이 사용되며, 주로 API 서버 역할을 담당하게 된다. express 키워드는 express-generator 를 전역으로 설치하면 사용할 수 있다.

```
# express 명령 설치
npm install --global express-generator

# 프로젝트 폴더 생성
cd ~/git/backfront
express backend -e
cd backend
npm install
```

백엔드 포트 4000 설정

다음과 같은 순서로 VSCode를 열어 API 서버의 포트를 변경한다.

```
cd ~/git/backfront
code .
```

backend/bin/www 파일의 3000을 4000으로 수정한다.

```
var port = normalizePort(process.env.PORT || '4000');
```

백엔드 nodemon 설정

개발시 코드가 변하면 자동으로 서버를 재시작해 주는 도구이다. nodemon 도구를 devDependency 로 추가하고, backend/package.json 파일에 아래 코드를 추가한다.

```
cd ~/git/backfront/backend
npm install -D nodemon
```

```
 "scripts": {
   "dev": "nodemon ./bin/www",
   "start": "node ./bin/www"
 },
```

Swagger 추가

프로젝트에서 작업에 대한 문서화는 중요한 일이다. 프로젝트에 참가하려는 사람에게 좋은 가이드가 되기 때문이다. 하지만, API의 경우 수작업으로 문서를 갱신하는 일은 매우 어렵기 때문에 이를 자동화해서 코드와 문서가 일치하게 만드는 것이 필요하다. 이를 위해 swagger 기술을 사용한다. backend에 필요한 swagger 라이브러리를 추가하고 app.js 파일에 다음의 코드를 추가한다.

```
cd ~/git/backfront/backend
npm install swagger-ui-express swagger-jsdoc
```

```
var app = express();

const swaggerUi = require('swagger-ui-express');
const swaggerJsdoc = require('swagger-jsdoc');

const options = {
 definition: {
   openapi: '3.0.0',
   info: {
     title: 'Hello World',
     version: '1.0.0',
   },
 },
 apis: ['./routes/*.js'], // files containing annotations as above
};
const openapiSpec = swaggerJsdoc(options);

app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(openapiSpec));
```

API 샘플 추가

API를 추가해서 실제로 swagger가 동작하는지 테스트해보자. backend의 routes/api.js 파일을 생성한다. 파일의 내용은 다음과 같다. jsdoc 주석을 통해서 api의 정보를 swagger에 전달할 수 있다.

```
const express = require('express');
const router = express.Router();

/**
* @openapi
* /api/hello:
*   get:
*     description: Welcome to swagger-jsdoc!
*     parameters:
*       - name: name
*         in: query
*         required: false
*         schema:
*           type: string
*     responses:
*       200:
```

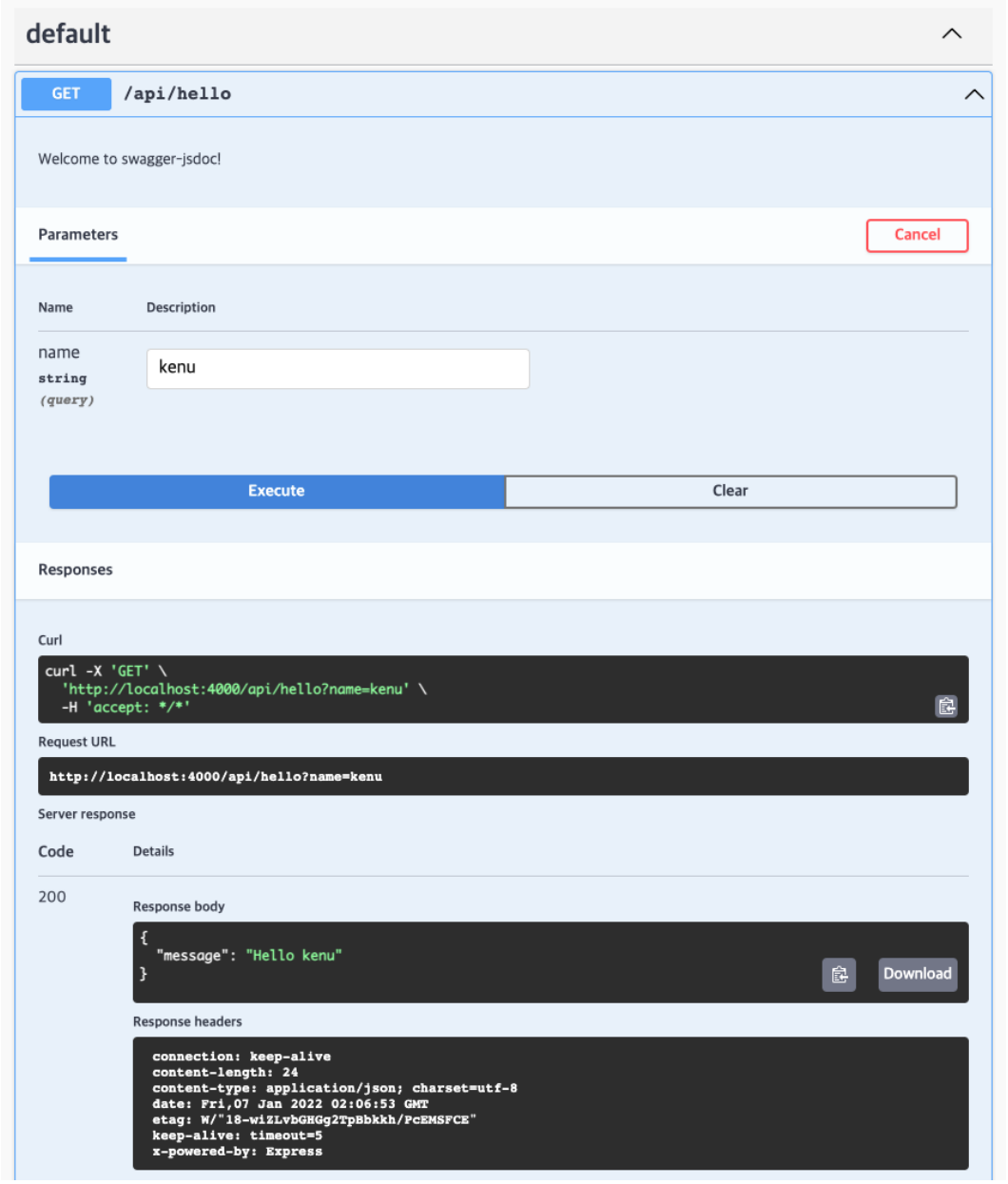
```
*          description: Returns a mysterious string.
*/
router.get('/hello', function (req, res, next) {
 const name = req.query.name || 'World';
 res.json({ message: `Hello ${name}` });
});
module.exports = router;
```

code: https://gist.github.com/kenu/606176842b456fd35875e368b3eebb09

app.js에 api.js 파일을 등록한다.

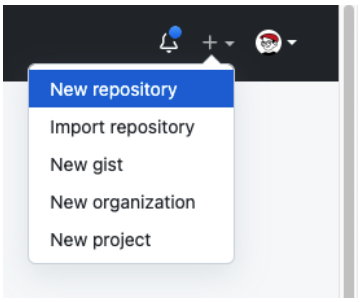
```
app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/api', require('./routes/api'));
```

http://localhost:4000/api-docs/ 페이지를 열어서 추가한 api를 확인한다. API를 클릭해 펼친 다음 우측에 있는 `Try it now` 버튼을 클릭한다. 아래에 있는 `Execute` 버튼을 눌러서 결과를 확인할 수 있다.

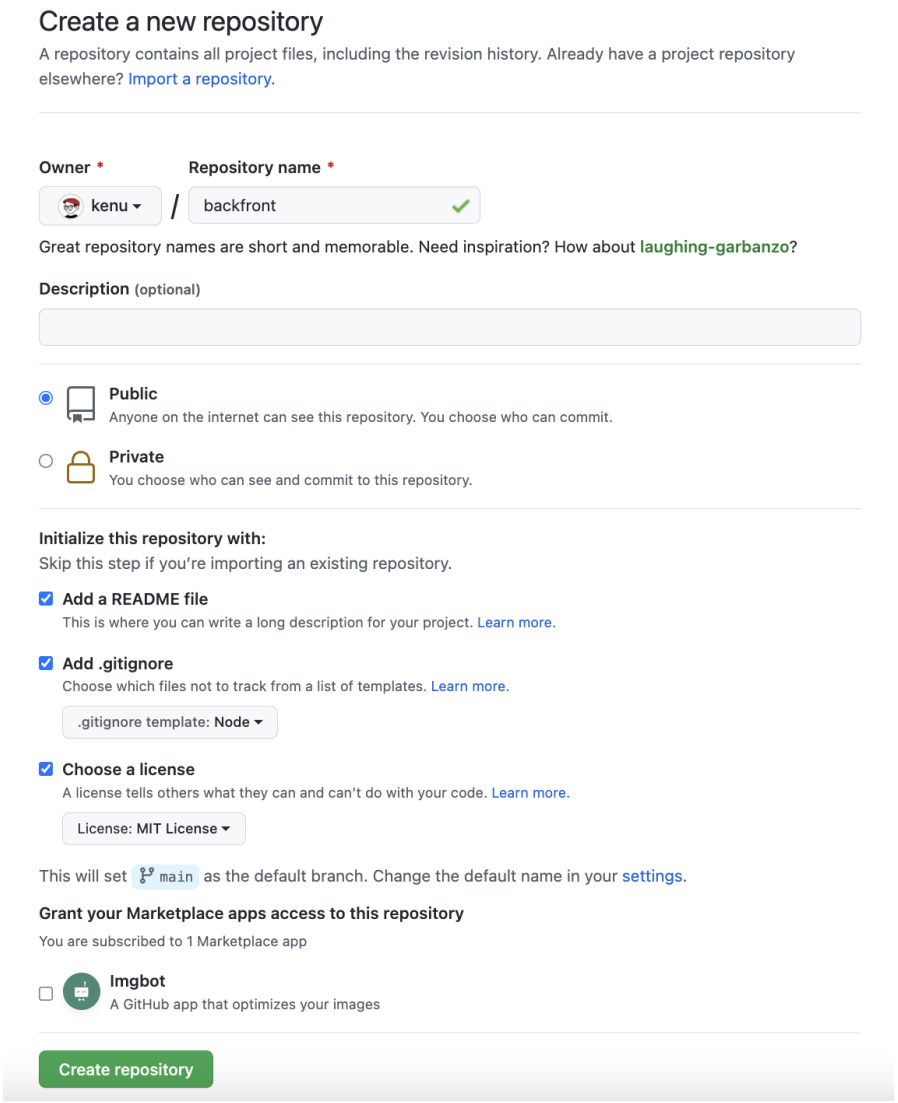


GitHub 프로젝트 생성

로컬 컴퓨터에서 작업한 내용을 커밋했다면 원격 저장소를 만들어 서버 배포에 사용할 수 있다. https://github.com 사이트에 로그인하고, New Repository 메뉴를 선택해서 프로젝트를 생성한다.



`Add .gitignore` 항목을 체크하고, Node를 선택한다. 커밋하지 않아도 되는 node_modules 같은 항목들이 .gitignore 파일에 있다. README.md 파일과 라이선스도 체크한다.



원격에 프로젝트 전송

원격 저장소의 주소를 등록한다. origin이라는 별칭을 사용하는데, 다른 이름을 사용할 수도 있다. 등록된 원격 저장소의 변경사항을 먼저 가져온 다음에 로컬에서 커밋한 것들을 전송한다.

```
cd ~/git/backfront
git remote add origin https://github.com/kenu/backfront
git remote -v
git pull origin main
git status
git add .
git commit -m "init"
git push origin main
```

GitHub의 해당 주소로 가서 커밋이 잘 올라갔는지 확인한다.

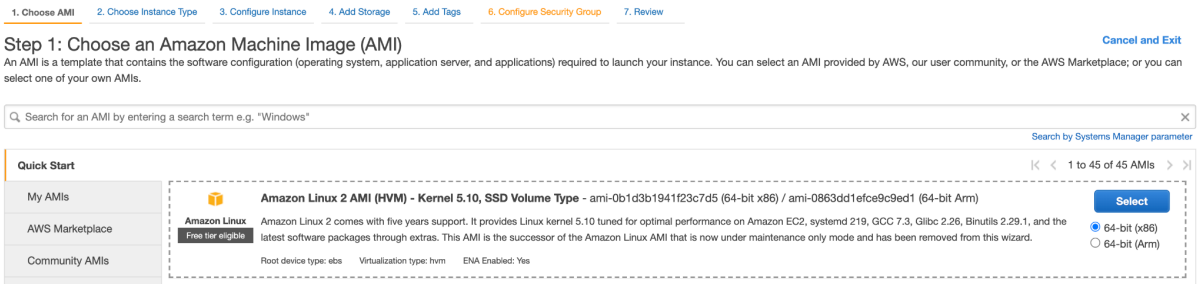
배포 프로세스

* EC2 서버 인스턴스 생성
* git, node.js 설치
* nginx 설치와 node.js 연결
* 프로젝트 빌드와 실행

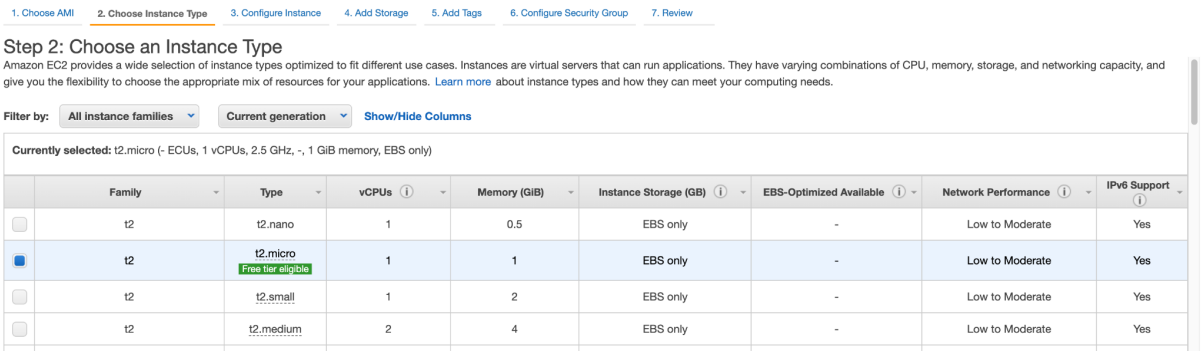
EC2 서버 인스턴스 생성

https://aws.amazon.com/ko 사이트에 로그인하고 EC2 인스턴스를 하나 생성한다. EC2 서비스 페이지의 Launch Instance 버튼을 클릭하면 된다. EC2는 Elastic Compute Cloud의 뜻이다.

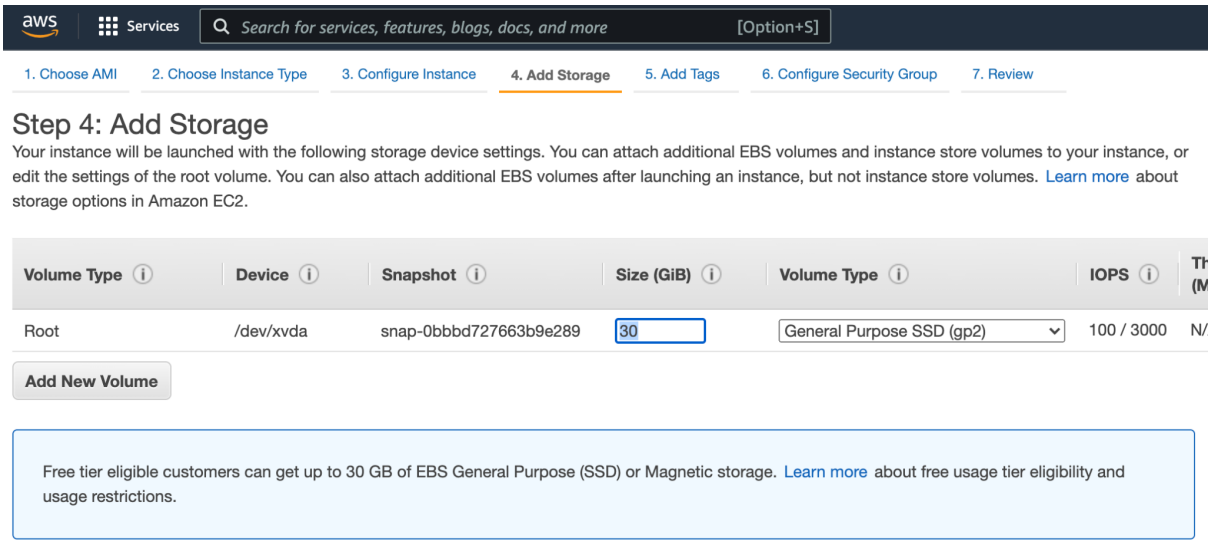
인스턴스 이미지는 Amazon Linux 2 AMI (HVM), 64-bit (x86)을 선택한다.
(2023년 3월 15일 Amazon Linux 2023(AL2023) AMI가 새로 나왔고**,** 기본으로 선택되어 있기 때문에 **AMI** 선택시 주의가 필요합니다**.** 변경된 부분은 **HTTPS** 설정이 가장 크고**, amazon-linux-extras** 명령이 사라지고**, dnf** 라는 패키지 매니저를 사용합니다**.)**



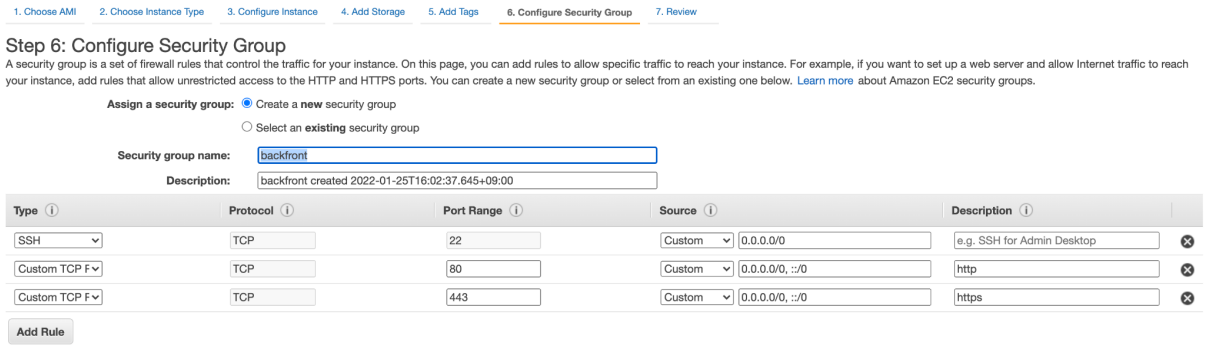
t2.micro 타입을 선택한다. CPU 1개, RAM 1G 사양이다. 최초 1년간 매달 750시간 무료로 사용 가능하다. 대략 월 1만원 정도 나오고, 각 타입마다 시간당 요금의 차이가 있다. 1년 또는 3년의 약정인 예약 인스턴스(Reserved Instance)를 지정할 경우 비용은 절반 정도로 줄일 수 있다.



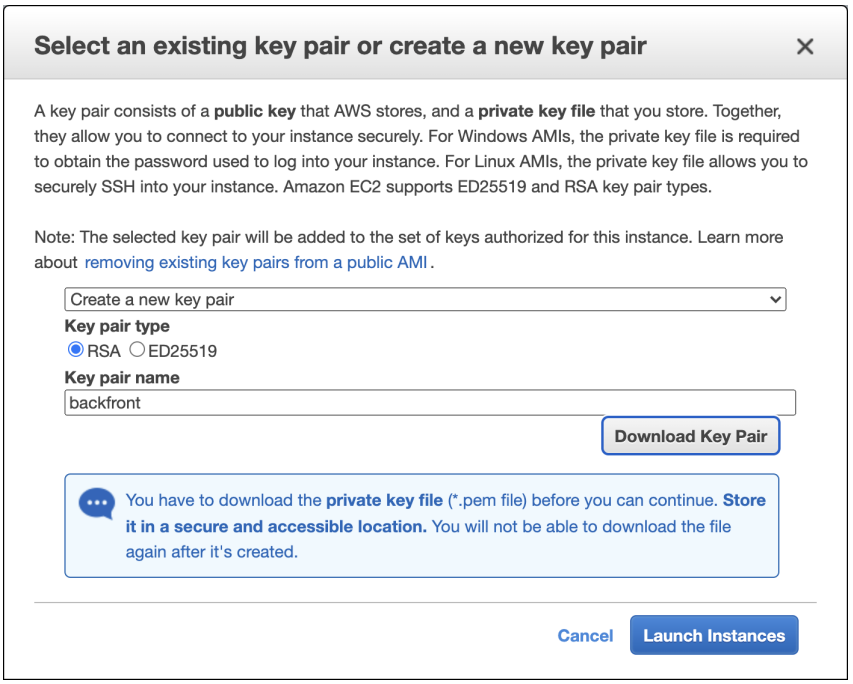
4. Add Storage 탭에서 30으로 수정한다. 30G까지는 무료로 제공된다.



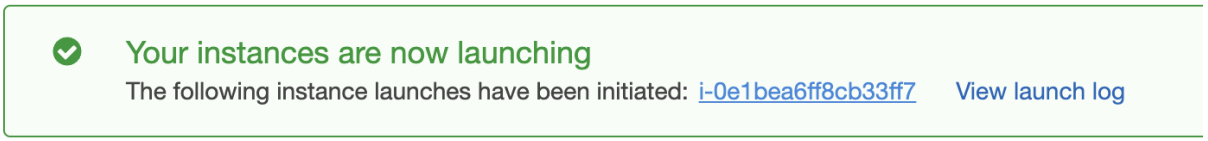
6. Configure Security Group 탭에서 Security group name 항목을 변경한다. Add Rule 버튼을 클릭해서 80, 443 포트를 추가한다. Security Group은 방화벽 설정을 하는 메뉴이고, 이후 설정을 수정할 수 있다.
우측 하단에 있는 `Review and Launch` 버튼을 클릭해서 Review로 이동한다.



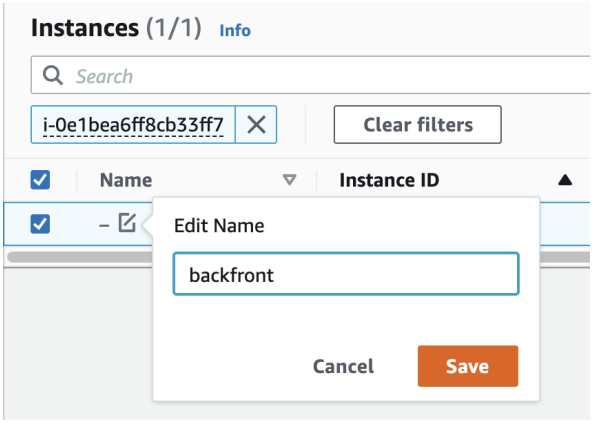
7. Review 탭에서 key pair를 생성할 수 있는데, 이 키는 private key이고, 이 키에 연결된 public key로 서버가 생성된다. key는 한 번만 다운로드 되기 때문에 백업할 필요가 있다. 다운로드를 마치면 Launch Instances 버튼을 클릭한다.



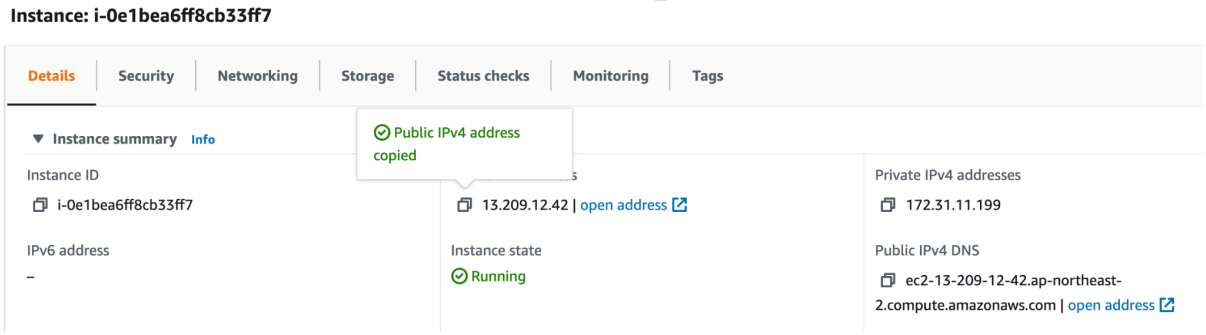
인스턴스 아이디 링크를 클릭한다.



Name 항목을 변경한다. 나중에 인스턴스가 많아질 때 구분하기 쉬워진다.



하단의 Public IP를 복사할 수 있다. 172로 시작하는 IP는 외부에서 접근할 수 없고, 같은 영역의 다른 인스턴스에서 접근할 수 있는 내부 IP이다.



다운로드 받은 key는 사용자 홈 디렉토리에 keys 폴더를 만들어 이동하고 권한을 400으로 수정한다.

```
mkdir ~/keys
mv ~/Downloads/backfront.pem ~/keys
chmod 400 ~/keys/backfront.pem
```

여기까지 EC2 인스턴스를 만들고, 접속할 준비를 마쳤다. 다음 명령으로 인스턴스에 접근한다.

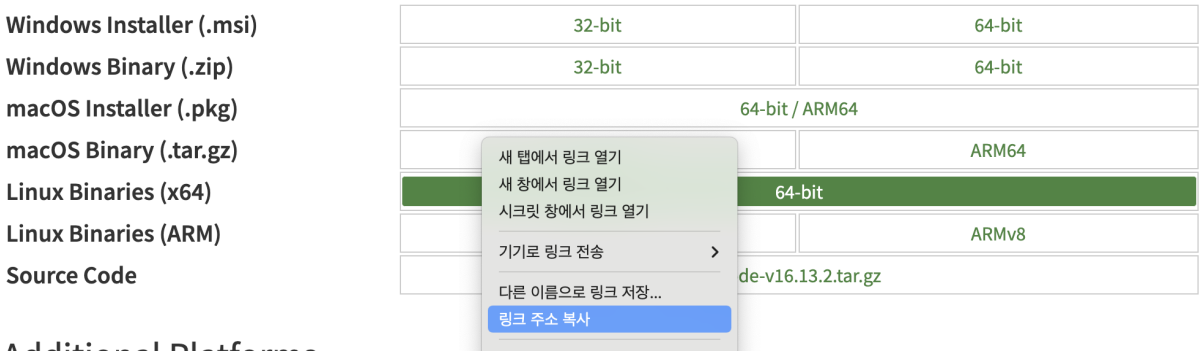
```
ssh -i ~/keys/backfront.pem ec2-user@13.209.12.42
```

서버 git, node.js 설치

서버에 접속해서 시스템을 update하고 git과 htop을 설치한다. htop은 메모리와 CPU 상태를 볼 수 있는 top 명령을 개선한 프로그램이다.

```
sudo yum update -y
sudo yum install git htop -y
```

nodejs는 https://nodejs.org/en/download 사이트에서 Linux Binaries (x64) 링크를 복사하고 다운로드 받아서 압축을 해제한다.




```
mkdir ~/local
cd ~/local
wget https://nodejs.org/dist/v16.14.0/node-v16.14.0-linux-x64.tar.xz
tar xvf node-v16.14.0-linux-x64.tar.xz
cd node-v16.14.0-linux-x64/bin
pwd
# 현재 경로를 복사한다.

vi ~/.bash_profile
```

~/local/node-v16.14.0-linux-x64/bin 폴더에 있는 node파일의 경로를 $PATH 에 추가한다.

```
PATH=$PATH:$HOME/.local/bin:$HOME/bin
PATH=$PATH:/home/ec2-user/local/node-v16.14.0-linux-x64/bin
```

수정한 환경 변수를 적용한다. node 버전을 확인한다.

```
source ~/.bash_profile
node -v
```

nginx 설치와 node.js 연결

nginx 웹서버를 설치하고, reverse proxy 설정을 통해서 node.js 4000번 백엔드 포트로 연결한다.
* AL2023일 경우

```
sudo yum install nginx
sudo vi /etc/nginx/nginx.conf
```

* Amazon Linux 2일 경우

```
sudo amazon-linux-extras install nginx1
sudo vi /etc/nginx/nginx.conf
```

```

아래 location 부분을 추가한다.

```
server {
    listen   80;
    listen   [::]:80;
    server_name _;
    root   /usr/share/nginx/html;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
        sendfile off;
        proxy_pass   http://127.0.0.1:4000;
        proxy_redirect   default;
        proxy_http_version 1.1;
        proxy_set_header   Host $host;
        proxy_set_header   X-Real-IP $remote_addr;
        proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header   X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
        proxy_max_temp_file_size 0;
    }

    error_page 404 /404.html;
    location = /404.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

code: https://gist.github.com/kenu/c203ad343287b44891020b4e6f15822e

수정을 마치고, nginx 설정을 테스트하고, ok 일 경우 서버를 시작한다. 오류가 있을 경우 메시지를 따라서 수정하면 된다. 서버를 재시작할 때 nginx 서비스도 자동으로 시작도록 설정한다.

```
sudo nginx -t
sudo systemctl start nginx
sudo systemctl enable nginx
curl localhost
```
```

최초 프로젝트 빌드와 실행

로컬 컴퓨터와는 달리 서버에서는 API 서버만 프로세스를 띄우고, React는 빌드한 결과물을 backend/public 경로로 복사해서 실행한다. node.js 서버를 띄우기 위해서 pm2 (https://pm2.keymetrics.io/) 명령을 추가한다.

```
npm install -g pm2

mkdir ~/git
cd ~/git
git clone https://github.com/kenu/backfront
cd backfront/frontend/
npm i
npm run build
cp -rf dist/* ../backend/public

cd ../backend/
npm i
pm2 start bin/www --name web
pm2 list
```

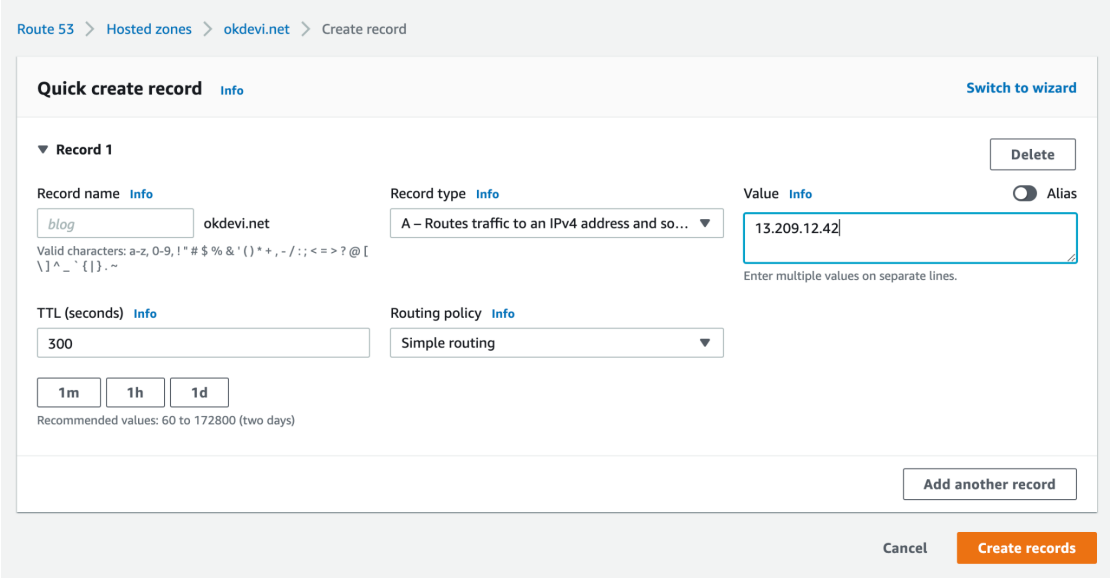
도메인 등록

* Route 53 도메인 추가
* nginx 설정
* HTTPS를 위한 인증서 설정

Route 53 도메인 추가

AWS를 이용하면 도메인 구매와 관리를 쉽게 할 수 있다. Route 53 서비스에 접속해서 필요한 도메인을 검색한 뒤에 장바구니에 넣고 구매를 진행한다. .net, .com은 1년에 $12 정도면 구입할 수 있다. 갱신은 매년 자동으로 진행되며, 설정에서 자동 연장을 그만 둘 수도 있다.

Route 53과 Hosted zones 의 도메인 페이지에 들어가 Create records 버튼을 클릭한다. 등록한 도메인과 EC2 인스턴스의 IP를 매핑한다.



수 분 내에 브라우저에서 해당 도메인을 입력하면, React화면을 확인할 수 있다.
http://okdevi.net
이후로 이 도메인에 인증서를 발급하는 작업을 진행한다.

nginx 설정

해당 도메인을 nginx.conf 파일에서 server_name 라인을 도메인으로 변경한다.

```
sudo vi /etc/nginx/nginx.conf
```

```
        server_name  okdevi.net "";
```

변경된 설정이 맞는지 확인한다.

```
sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

HTTPS를 위한 인증서 설정

도메인이 설정이 되면 HTTPS 설정이 가능하다. LetsEncrypt에서 무료로 인증서를 등록할 수 있다. https://letsencrypt.org 에는 2.6억 개 이상의 사이트가 사용하고 있는 것을 보여준다.

현재까지는 ec2-user 계정으로 작업을 했기 때문에 sudo 를 붙여 주었다. EC2 인스턴스의 root 권한으로 로그인하려면 sudo su - 명령으로 가능하다. 다음 명령으로 인증서 설정 작업에 필요한 라이브러리를 설치한다.

* AL2023일 경우

```
sudo su -
dnf install python3 augeas-libs
dnf remove certbot
python3 -m venv /opt/certbot/
/opt/certbot/bin/pip install --upgrade pip
/opt/certbot/bin/pip install certbot certbot-nginx
ln -s /opt/certbot/bin/certbot /usr/bin/certbot
```

* Amazon Linux 2일 경우

```
sudo su -
yum -y install yum-utils
yum-config-manager --enable rhui-REGION-rhel-server-extras rhui-REGION-rhel-server-optional
yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
yum install certbot python2-certbot-nginx
```

설치가 완료되면 이제 certbot 명령으로 도메인에 해당하는 인증서를 받을 수 있다.

```
certbot --nginx
# ...
Which names would you like to activate HTTPS for?
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1: okdevi.net
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel):
Requesting a certificate for okdevi.net
# ...
Deploying Certificate to VirtualHost /etc/nginx/nginx.conf
Redirecting all traffic on port 80 to ssl in /etc/nginx/nginx.conf

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Congratulations! You have successfully enabled https://okdevi.net
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/okdevi.net/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/okdevi.net/privkey.pem
   Your certificate will expire on 2022-05-10. To obtain a new or
# ...
   renew *all* of your certificates, run "certbot renew"
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

인증서 발급을 정상적으로 마치면, /etc/nginx/nginx.conf 에 인증서를 연결하는 부분이 자동 추가된다. 설정 파일을 확인하고, nginx를 재시작한다. root계정이기 때문에 sudo 는 쓰지 않아도 된다.

```
nginx -t
systemctl restart nginx
```

이제 브라우저에서 도메인을 쳐보면 https로 자동 연결되는 것을 확인할 수 있다.

인증서 유효기간이 90일이기 때문에 자동 업데이트 스케줄을 등록한다.

```
echo "0 0,12 * * * root python -c 'import random; import time;
time.sleep(random.random() * 3600)' && certbot renew" | sudo tee -a /etc/crontab >
/dev/null

cat /etc/crontab
```

code: https://gist.github.com/kenu/68a90b3ae9823a9857a90c39d33ca39f

다음으로 넘어가기 전에 Ctrl+D를 눌러서 root계정을 로그아웃한다.

배포 자동화하기

* CI/CD 구성하기

CI/CD 구성하기

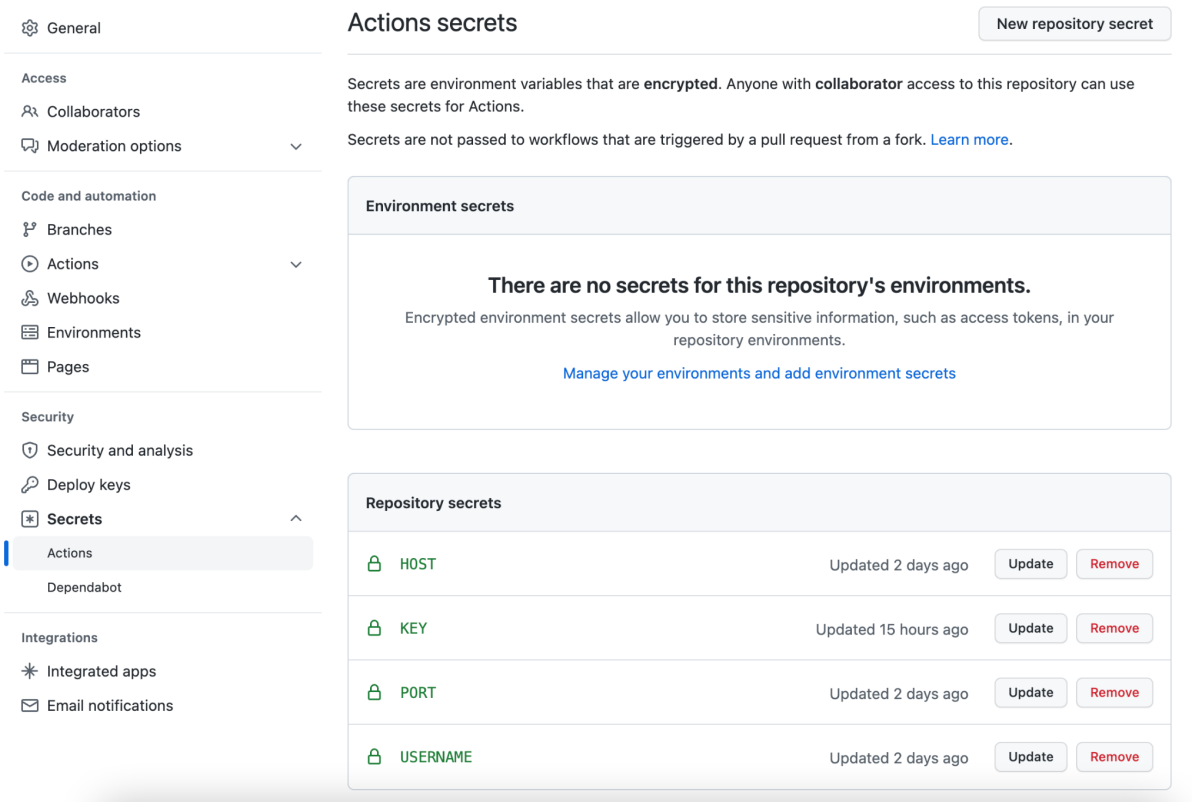
Continuous Integration은 영역별로 나눠서 개발할 때, 인터페이스 등의 충돌을 미리 발견하기 위해 매일 또는 매시간 저장소를 자동으로 빌드하는 개념이다.
Continuous Delivery는 저장소 코드의 변경이 발생하면 개발 서버 또는 운영계에 자동으로 배포하는 작업이다.
CI/CD 도구로 Jenkins가 가장 유명하지만, 이번에는 GitHub Actions를 통해서 구성해 본다.

프로젝트 저장소 root에 .github/workflows 폴더를 만들고, 다음 deploy-main.yml 파일을 추가한다.

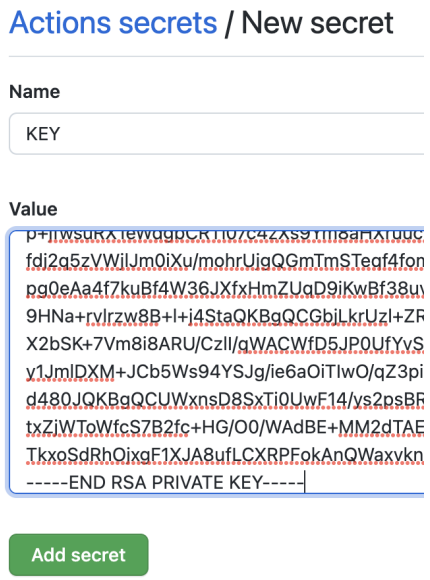
```
name: remote ssh command for deploy
on:
 push:
   branches: [main]
jobs:
 build:
   name: Build
   runs-on: ubuntu-latest
   steps:
     - name: executing remote ssh commands using key
       uses: appleboy/ssh-action@master
       with:
         host: ${{ secrets.HOST }}
         username: ${{ secrets.USERNAME }}
         key: ${{ secrets.KEY }}
         port: ${{ secrets.PORT }}
         script: |
           ./deploy.sh
```

code: https://gist.github.com/kenu/f74bc21c5bf9c68bdf1d84eadcb3b252

https://github.com/kenu/backfront/settings 프로젝트 메뉴 탭에서 Settings 서브 메뉴의 Secrets > Actions 를 선택하고, 우측상단의 New repository secret 버튼을 클릭해서 아래와 같은 항목들을 추가한다.



HOST에는 도메인, KEY는 backfront.pem 파일 내용, PORT는 22, USERNAME은 ec2-user로 설정한다.



EC2 서버에 접속해서 deploy.sh 파일을 만든다. 서버에 접속하지 않고 GitHub Actions 설정으로 저장소의 main 브랜치가 변경될 때마다 서버에 최신 버전을 받아서 자동으로 빌드하고 재시작하는 스크립트이다.

```
ssh -i ~/keys/backfront.pem ec2-user@okdevi.net
vi ~/deploy.sh
```

```

```
#!/bin/bash
source ~/.bash_profile

cd ~/git/backfront/
git pull origin main
cd frontend/
npm i
npm run build
cp -rf dist/* ../backend/public

cd ../backend/
npm i
pm2 stop web
pm2 start bin/www --name web --update-env
sleep 2
pm2 list
```

파일에 실행 권한을 부여한다.

```
chmod +x ~/deploy.sh
```

이상과 같이 설정을 마치면 main 브랜치의 README.md 파일을 수정하고, commit한 후 저장소에 푸시한다.
GitHub의 Actions 메뉴를 보면 deploy.yml 에 정의된 작업의 로그를 볼 수 있다.

이상이 있다면 로그에 나타난 메시지를 확인해서 조치해야 한다.

CORS 설정

도메인을 origin이라고 표현한다. 도메인이 달라질 경우 보안 때문에 정보를 주고 받는 규칙이 생기게 되는데, 이것을 CORS(Cross Origin Resource Sharing) 규칙이라고 한다.
참고: https://okdevtv.com/mib/cors

* 로컬 개발 환경에서 CORS 설정
* VITE 환경변수를 이용한 동적인 빌드

로컬 개발 환경에서 CORS 설정

도메인이 달라질 때 발생하는 이슈가 CORS이다. Cross Origin Resource Sharing을 뜻한다.
프론트엔드는 3000번 포트를 사용하고, 백엔드는 4000번 포트를 사용하기 때문에 API를 호출할 때 http://localhost:4000/api/hello 주소를 사용한다. 만약 /api/hello URI와 같이, 도메인을 사용한다면 발생하지 않는다.
```

우선 프론트엔드에 fetch() 코드를 추가하고, 테스트해 보자. frontend/src/App.jsx 파일을 다음과 같이 수정한다.

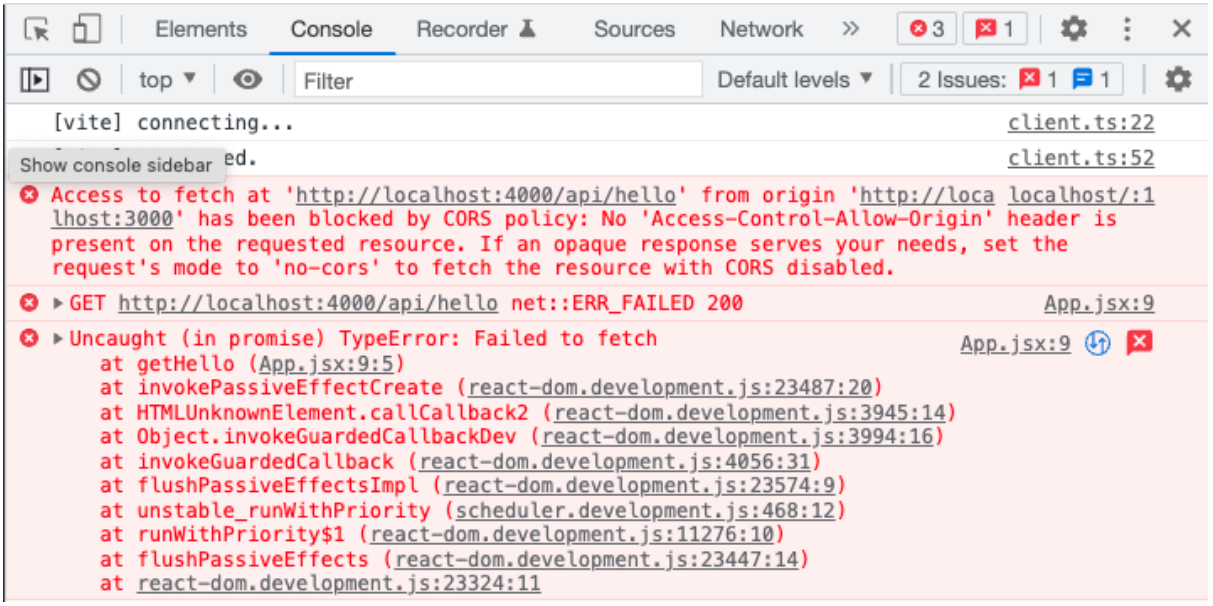
```
import { useState, useEffect } from 'react'
import logo from './logo.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)
  function getHello() {
    const greet = document.getElementById('greet')
    fetch('http://localhost:4000/api/hello')
      .then(response => response.json())
      .then(data => greet.innerHTML = JSON.stringify(data))
  }
  // Similar to componentDidMount and componentDidUpdate:
  useEffect(getHello)
```

code: https://gist.github.com/kenu/1661ec1d40b3753596af5638dba86cba

같은 파일 아래쪽에 있는 렌더링 영역에 태그를 추가한다.

```
          </button>
        </p>
        <p>
          api called: <code id="greet"></code>
        </p>
        <p>
          Edit <code>App.jsx</code> and save to test HMR updates.
        </p>
```

수정을 마치고 frontend와 backend 실행해서 연결하면 다음과 같은 에러가 개발자 도구의 콘솔에 나타난다.



이 에러를 해결하기 위해서 backend에 cors 설정이 필요하다.
express.js에서 cors를 설정하는 것은 비교적 어렵지 않다. 다음과 같이 backend에 라이브러리를 추가한다.

```
cd ~/git/backfront/backend
npm i cors
```

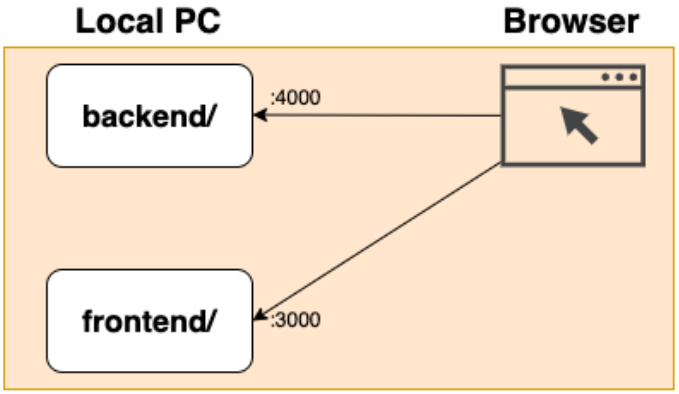
그리고, backend/app.js 파일에 다음 코드를 추가한다.

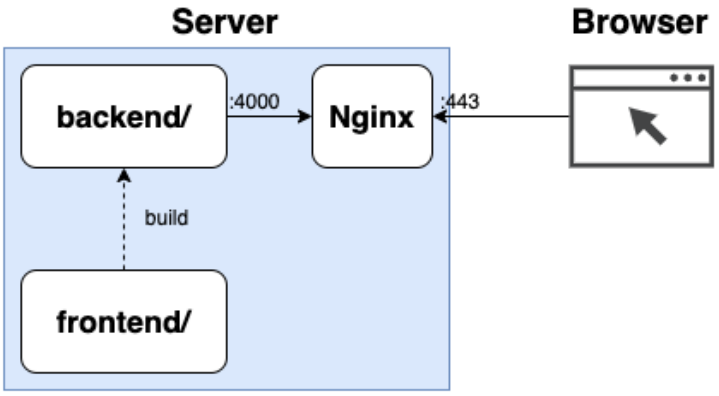
```
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(openapiSpec));


var cors = require('cors');
app.use(cors());
```

다시 서버를 실행하고, 확인하면 정상동작하는 것을 확인할 수 있다. 이를 커밋하고 GitHub에 푸시해서 서버에서 실행되는 것을 확인하면 다른 에러가 발생한다. 이에 대한 해법을 알아보자.

VITE 환경변수를 이용한 동적인 빌드

지금까지 구성한 것을 그림으로 그려보면 다음과 같다. 로컬PC의 개발 환경에서는 3000, 4000 포트 두 개를 사용하는데, 서버의 경우는 다르다. 프론트엔드 React 프로젝트를 빌드한 결과를 백엔드 public 폴더에 복사하고, 같은 도메인으로 Nginx를 통해서 서비스한다. 즉 http://localhost:4000 부분이 없어져야 하는 것이다.





프론트엔드의 빌드도구는 vite.js를 사용하고 있다. 로컬에서 개발할 때와 서버에서 빌드될 때 VITE_로 시작하는 환경변수를 통해서 다르게 만들 수 있다.

frontend/.env.local 파일을 만들고, 아래 코드를 추가한다.

```
VITE_API_SERVER=http://localhost:4000
```

frontend/src/App.jsx 파일의 하드코딩된 부분은 다음과 같이 수정한다.

```
    fetch(import.meta.env.VITE_API_SERVER + '/api/hello')
      .then(response => response.json())
      .then(data => greet.innerHTML = JSON.stringify(data))
```

로컬 PC의 서버를 재시작한다. 환경변수의 변경은 재시작해야 적용되는 경우가 많다. 로컬에서 제대로 동작하는 것을 확인하면 로컬 서버를 모두 종료한다. 변경사항을 커밋하고, 푸시한 뒤에 서버에서 확인한다.

서버에 접속해서 frontend/.env 파일을 직접 수정한다. frontend/.env 파일을 만들고, 아래 코드를 추가한다. 서버에서 읽을 환경 파일이다.

```
VITE_API_SERVER=
```

서버를 재시작하고, 이상이 없는지 확인한다.

정리

지금까지 React + API Server 프로젝트 개발과 배포 과정 그리고, 도메인과 SSL 등을 설정해 보았다.

* 준비하기
  * 이 과정에 필요한 개발 플랫폼은 Node.js, 개발 관련한 도구 git, VS Code, 그리고 관련된 계정으로 GitHub과 AWS를 사용했다.
* 개발 프로세스
  * 로컬 컴퓨터에서 작업하기 위한 기본적인 프론트엔드와 백엔드의 프로젝트 환경을 구성해 보았다.
* 배포 프로세스
  * 서비스를 위해 서버에 배포하는 절차와 방법을 실행해 보았다.
* 도메인 등록
  * 도메인과 서버를 연결하고, HTTPS 연결을 위한 방법으로 설치를 완료해서 EC2 서버에서 도메인으로 정상 동작하는 것을 확인해 보았다.
* 배포 자동화하기
  * GitHub에 코드가 변경될 때 자동으로 개발 서버에 배포되는 작업을 GitHub Actions를 이용해 구성해 보았다.
* CORS 설정
  * 다른 도메인일 경우 발생하는 CORS 이슈의 해결법을 알아보았다. 그리고, 로컬에서 개발할 때와 서버에 배포할 때의 차이점 그리고 환경변수를 구성해 보았다.

React와 API 만드는 것에 대한 자세한 내용은 좋은 문서들과 강좌들이 많이 있기 때문에 여기에서는 깊게 다루지 않았다. **React** 프로젝트의 개발 환경을 처음부터 시작해서**,** 서버에 배포하는 것에 집중하기 위해서였다.
이 강의를 끝까지 확인해 주어서 감사하고, 여기에 소개된 내용은 아주 기본적인 내용들로 구성했기 때문에 각 팀의 상황에 따라 응용해서 사용하길 바란다.

Update

- https://www.inflearn.com/chats/517583 by Alan Choi
    - npm install 할 때 -S 또는 --save 옵션은 npm 5 이후로 필요하지 않습니다.
      참고: https://stackoverflow.com/a/35849725/4010013
    - EC2 instance에 Node 설치는 아래 tutorial을 참고하면 더 간단합니다.
      참고: https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-up-node-on-ec2-instance.html
    - Express application generator를 사용하기 위해 package global install을 하는 대신 npx 명령어를 사용할 수 있습니다.
      참고: http://expressjs.com/en/starter/generator.html

- https://www.inflearn.com/news/903768 by Kenu
    - 2023/03/15 Amazon Linux 2023 AMI 출시에 따른 설정 변경 적용했습니다.
    - amazon-linux-extras 명령이 삭제되었고, 인증서 발급을 위한 certbot 설치도 pip 기반으로 변경되었습니다.