AWS , GCP ( Google Cloud Platform ) 에 서비스 배포하는 방법

✓ morgan , express-session 배포환경 설정

gitcodeshare 프로젝트에서 이런 부분을 볼수있었는데

⁃ if ( process.env.NODE\_ENV === 'production' )

process.env.NODE\_ENV는 배포 환경인지 개발 환경인지를 판단할 수 있는 환경 변수이다.

배포 환경일 때는 morgan을 combined모드로 사용하고 개발 환경일 때는 dev모드로 사용한다.

이둘의 차이점이라고 한다면 combined모드는 dev모드보다 더많은 사용자 정보를 로그에 남긴다.

또한, process.env.NODE\_ENV는 .env에 넣을수가 없다. ( .env 파일은 정적 파일이기 때문에)

express-session은 사용자에 따라 설정할 필요가 없을 수도 있다.

https를 적용할 경우에 porxy = true , cookie.secure = true

✓ sequelize

sequelize에서 가장 큰 문제는 비밀번호가 하드 코딩되어 있다는 것이다. JSON 파일이므로 변수를 사용할수가 없다. 하지만 Json 대신 Js 파일을 설정 파일로 쓸수있게 지원해주기 때문에 config.json을 지우고 config.js로 생성시켜준다.

require('dotenv').config(); // Js파일을 사용했기때문에 dotenv모듈을 사용할수있다.

module.exports = {

development : {

username : 'bla' ,

password : process.env.SEQUELIZE\_PASSWORD,

생략 ... ( database , host, dialect , operatorsAliases)

},

production : {

username : 'bla'

password : process.env.SEQUELIZE\_PASSWORD,

생략 ... ( 위와 동일 )

logging : false, // 쿼리 수행시 콘솔에 SQL문 노출을 막기위한것

},

};

이렇게 배포용과 개발용에 따라서 process.env가 개발 속성, 배포 속성으로 적용될수있다.

또한, 쿼리를 수행할 때마다 콘솔에 SQL문이 노출되기 때문에 숨기는 것이 좋다.

\*\* 배포환경에서 데이터베이스에 한글이 저장되지 않는 문제가 발생할 수 있다.

charset : 'utf-8' , collate : 'utf8\_general\_ci' 를 추가해주자

✓ cross-env

cross-env패키지를 사용하면 동적으로 process.env를 변경할 수 있고, 모든 운영체제에서 동일한 방법으로 변경할 수 있게 된다.

package.json 파일에 scripts 부분을 보자

"script" : {

"start" : "NODE\_ENV=production PORT=80 node app",

"dev" : "nodemon app"

}

위에 코드를 보면 npm start는 배포 환경에서 사용하는 스크립트로 만들어주었고 npm run dev는 개발 환경에서 사용하는 스크립트로 만들어 주었다.

npm start를 보면 "start" : "NODE\_ENV=production PORT=80 node app" 이처럼 쓰여있는데 앞에

NODE\_ENV=production PORT=80 처럼 쓰면 스크립트 실행 시 process.env를 동적으로 설정해줄수있다.

• 단!! 해당 방법은 리눅스나 맥에서는 가능하지만 윈도우 환경에서는 저렇게 설정할 수 없다.

• 윈도우 환경에서는 NODE\_ENV앞에 "coress-env"를 추가해주면 윈도우에서 사용할수있다.

✓ pm2

 pm2는 원활한 서버 운영을 위한 패키지이다. 개발시 nodemon을 쓴다면, 배포 시에는 pm2를 쓴다는 말이 있을정도로 유용하다. 가장 큰 기능은 서버가 에러로 인해 꺼졌을 때 서버를 다시 켜주는 역활을 하는것이다.

또 하나 중요한 기능은 바로 멀티 프로세싱입니다. 멀티 스레딩은 아니지만 멀티 프로세싱을 지원하여 노드 프로세스 개수를 1개 이상으로 늘릴 수 있습니다. CPU 코어를 하나만 사용하여 다른 코어들이 노는 일을 방지할 수 있습니다. 노드는 클라이언트로부터 요청이 왔을 때 요청을 여러 노드 프로세스에 고르게 분배합니다. 하나의 프로세스가 받는 부하가 적어지므로 서비스를 더 원활하게 운영할 수 있습니다.

단! 단점 멀티 스레딩이 아니므로 서버의 메모리 같은 자원을 공유하지는 못한다. 지금까지 세션을 메모리에 저장했는데, 메모리를 공유하지 못해서 프로세스 간에 세션이 공유되지 않는 것입니다. 로그인 후 새로고침을 반복할 때 세션 메모리가 있는 프로세스로 요청이 가면 로그인된 상태가 되고, 세션 메모리가 없는 프로세스로 요청이 가면 로그인이 안 된 상태가 되는 것입니다.

이 문제를 해결하기 위해서 memcached나 redis같은 서비스를 사용하는것이다 !!!!

(GitCodeShare에서 사용했지만... 이해못했었는데.... 이해했다)

pm2는 노드의 cluster 모듈처럼 클러스터링을 가능하게 하는 클러스터링 모드가 있다.

> pm2 start app.js -i (생성하길 원하는 프로세스 개수)

현재 프로세스를 모니터링 하고 싶다면. > pm2 monit 를 입력하면 모니터링 할수있다.

\*\*\* 실제 서버 운영 시 , 서비스 규모가 커질수록 비용이 발생할 가능성이 커지므로 놀고 있는 코어까지 클러스터링을 작동하게 하는 것이 비용을 절약하는 길이다. 하지만 프로세스 간에 메모리를 공유하지 못하는 문제도 있으므로 최대한 프로세스 간에 공유하는 것이 없도록 설계해야한다.

✓ winston

실제 서버 운영 시 console.log와 console.error를 대체하기 위한 모듈이다.

log와 error를 사용하면 개발 중에는 편리하게 서버의 상황을 파악할 수 있지만, 실제 배포 시에는 사용하기 어렵다. 언제 호출되었는지와 서버가 종료되는 순간 로그들도 사라져 버리기때문에 !!

로그를 파일이나 다른 데이터베이스에 저장하기위해 winston을 사용한다.

createlogger 메서드로 logger를 만들고 설정을 추가해주면된다 설정은 level(심각도), format(로그형식), transports(로그 저장 방식) 등 이 있으니 입맛대로 골라 쓰자

✓ helmet, hpp

서버의 각종 취약점을 보완해주는 패키지들입니다.

배포 환경일 때만 적용하면 되기때문에

if(process.envNODE\_ENV === 'production') {

app.use(helmet());

app.use(hpp());

}

로 사용해 주면된다.

- 참고

Helmet : https://www.npmjs.com/package/helmet

Winston: https://github.com/winstonjs/winston

✓connect-redis

connect-redis 패키지로부터 RedisStore객체를 require한다.

connect-redis는 express-session에 의존성이 있기때문에 session을 인자로 넣어주는것을 잊지말자.

const RedisStore = require('connect-redis')(session)

express-session 미들웨어에 store옵션을 추가해주고 Host, Port, Pass를 차례대로 넣어주면

세션 정보가 메모리 대신 레디스에 저장된다. 따라서 로그인 후 서버를 껐다 켜도 로그인이 유지된다.

( GitCodeShare에서 redis를 사용한 이유 ?!?!!!! )