

DevOps 구축 BOOTCAMP



3주차 목표

Web server 영역과 동일한 WAS 다중 서버 환경 구축

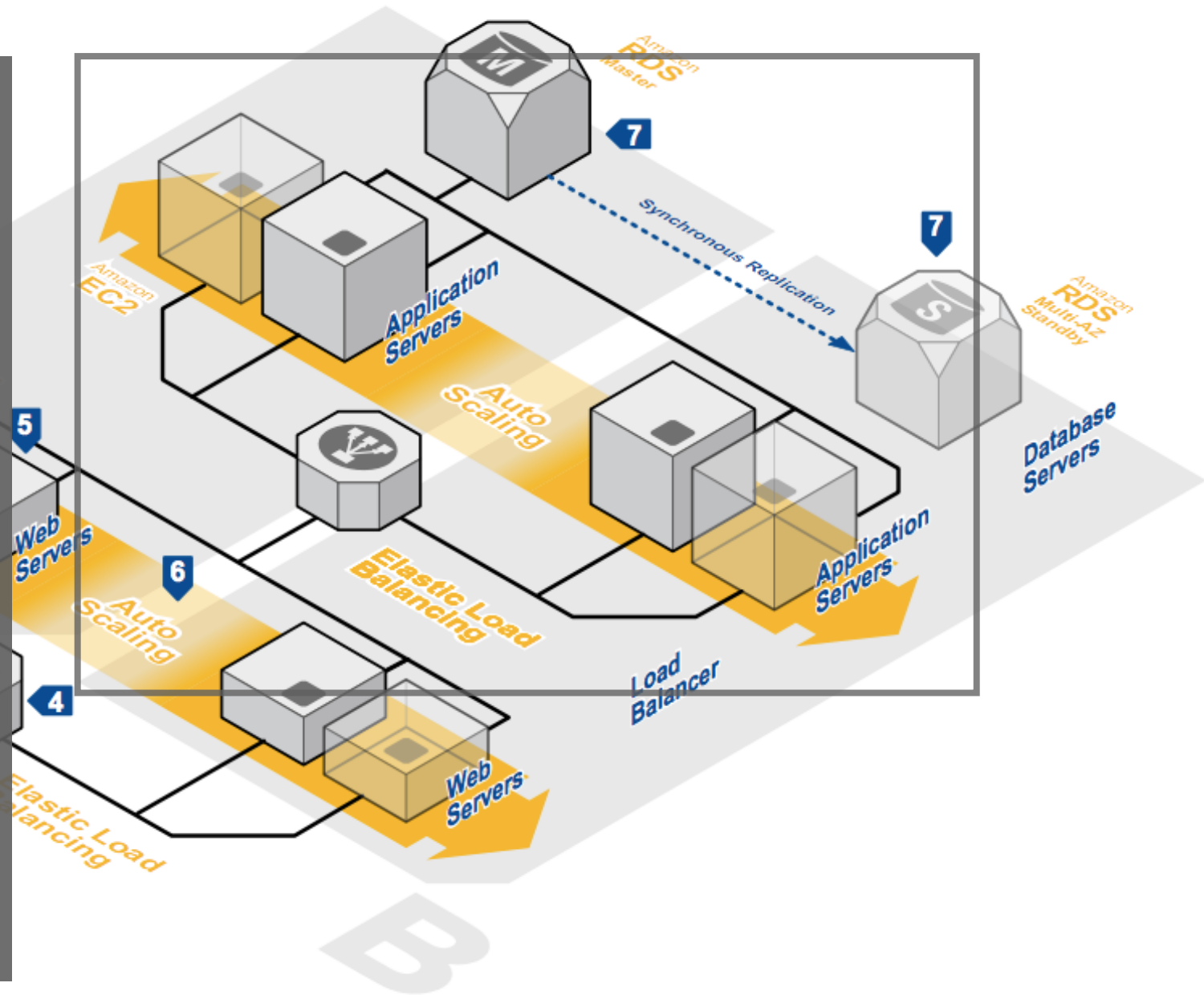
AWS RDS를 활용해 DB 서버 구축 후 Web server + WAS + DB 간 인프라 구축

DevOps로 이전 준비

WAS 영역

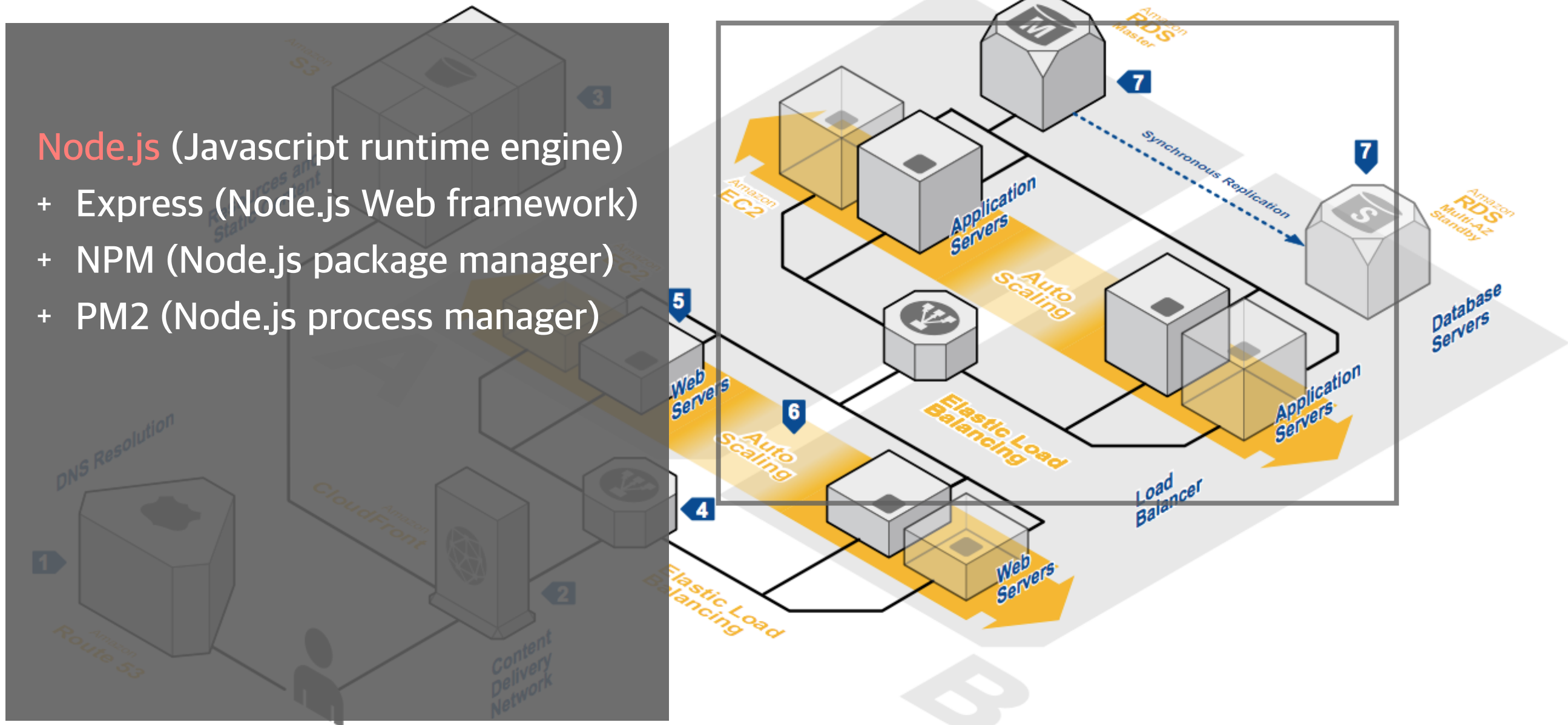
WAS(Web application server)

- 동적 데이터를 제공하기 위한 서버
- Nginx를 proxy 서버로 설정하여 WAS 영역과 연결하는 경우가 많음
- 개발팀이 사용하는 언어, 프레임워크가 제각각이라 **DevOps 영역에서 효율적**으로 설계, 운영해야함



WAS 영역

- Node.js** (Javascript runtime engine)
- + Express (Node.js Web framework)
 - + NPM (Node.js package manager)
 - + PM2 (Node.js process manager)



Node.js

- Chrome V8 엔진 기반 **Javascript 런타임**
 - Node.js는 WAS or 웹서버가 아닌 javascript 코드를 실행하는 런타임
 - HTTP 통신을 위한 서버는 직접 작성해야함
 - ExpressJS, KoaJS 등 웹 프레임워크 사용
- Chrome V8을 기반으로 하여 빠른 코드 실행 환경 제공
- 단일 쓰레드와 이벤트 루프(Nonblocking I/O)를 사용하여 많은 connection을 처리하는 웹 서비스 환경에서 **유리함**
- 서버 리소스의 효율성과 라우팅이 유리함으로 Microservice Architecture에 적합하여 해당 강의의 WAS 구축 기술 스택으로 사용합니다.

Node.js 환경 구축

Node.js 환경이 구동될 EC2 인스턴스를 생성해주세요.

Node.js 환경 구축

1. AMI 선택 2. 인스턴스 유형 선택 3. 인스턴스 구성 4. 스토리지 추가 5. 태그 추가 6. 보안 그룹 구성 7. 검토

단계 6: 보안 그룹 구성

보안 그룹은 인스턴스에 대한 트래픽을 제어하는 방화벽 규칙 세트입니다. 이 페이지에서는 특정 트래픽을 인스턴스에 도달하도록 허용할 규칙을 추가할 수 있습니다. 예를 들면 웹 서버를 설정하여 인터넷 트래픽을 인스턴스에 도달하도록 허용하려는 경우 HTTP 및 HTTPS 트래픽에 대한 무제한 액세스를 허용하는 규칙을 추가합니다. 새 보안 그룹을 생성하거나 아래에 나와 있는 기존 보안 그룹 중에서 선택할 수 있습니다. Amazon EC2 보안 그룹에 대해 [자세히 알아보기](#)

보안 그룹 할당: ☒ 새 보안 그룹 생성

☐ 기존 보안 그룹 선택

보안 그룹 이름: nodejs-was-secure

설명: 최대 255자

유형 ⓘ	프로토콜 ⓘ	포트 범위 ⓘ	소스 ⓘ	설명 ⓘ	
SSH	TCP	22	사용자 지정 0.0.0.0/0	예: 관리자 데스크톱용 SSH	✕
사용자 지정 TCP	TCP	8080	사용자 지정 0.0.0.0/0, ::/0	예: 관리자 데스크톱용 SSH	✕

규칙 추가

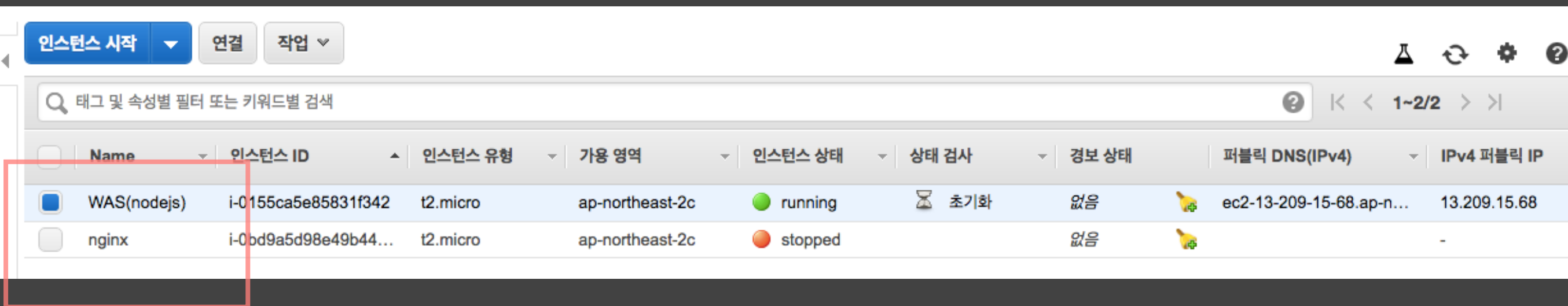


경고

소스가 0.0.0.0/0인 규칙은 모든 IP 주소에서 인스턴스에 액세스하도록 허용합니다. 알려진 IP 주소의 액세스만 허용하도록 보안 그룹을 설정하는 것이 좋습니다.

- WAS(Node.js)가 구동될 인스턴스입니다. WAS는 8080포트로 통신하기에 새로운 보안 그룹을 생성하여 8080, 22(SSH) 포트만 허용합니다.

Node.js 환경 구축



인스턴스 시작 | 연결 | 작업

태그 및 속성별 필터 또는 키워드별 검색

	Name	인스턴스 ID	인스턴스 유형	가용 영역	인스턴스 상태	상태 검사	경보 상태	퍼블릭 DNS(IPv4)	IPv4 퍼블릭 IP
<input checked="" type="checkbox"/>	WAS(nodejs)	i-0155ca5e85831f342	t2.micro	ap-northeast-2c	● running	초기화	없음	ec2-13-209-15-68.ap-n...	13.209.15.68
<input type="checkbox"/>	nginx	i-0bd9a5d98e49b44...	t2.micro	ap-northeast-2c	● stopped		없음		-

- 이번 강의부터 각 EC2 인스턴스들의 역할(웹 서버/WAS)이 분리됩니다. 헛갈리지 않도록 Name 탭에서 각 인스턴스의 이름을 부여합니다.

Node.js 환경 구축

```
1. ec2-user@ip-172-31-25-131:~ (bash)
a1@1ui-MacBook-Air:~$

sudo ssh -i ~/Desktop/ec2_test.pem ec2-user@{접속할 WAS 인스턴스 DNS 주소} ↵

sudo su ↵

curl --silent --location https://rpm.nodesource.com/setup_8.x | sudo bash -

yum install -y nodejs

node -v

npm -v
```

Node.js 환경 구축

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~\$

- 1) `curl --silent --location https://rpm.nodesource.com/setup_8.x | sudo bash -`
- 2) `yum install -y nodejs`
- 3) `node -v`
- 4) `npm -v`

1. curl 클라이언트를 통해 nodejs(v8.x LTS), NPM 설치 코드 받아오기
2. yum 클라이언트로 nodejs 설치(npm도 같이 설치됨)
3. 설치된 nodejs 버전 확인
4. 설치된 NPM 버전 확인

Node.js 환경 구축

```
1. ec2-user@ip-172-31-25-131:~ (bash)
a1@1ui-MacBook-Air:~$

yum install -y git

git clone -b v1 https://github.com/owen1025/Fastcampus-api-deploy.git

ls -al

cd Fastcampus-api-deploy/
```

Node.js 환경 구축

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~\$

- 1) `yum install -y git`
- 2) `git clone -b v1 https://github.com/owen1025/Fastcampus-api-deploy.git`
- 3) `ls -al`
- 4) `cd Fastcampus-api-deploy/`

1. git 클라이언트 설치
2. git을 통해 Fastcampus-api-deploy 프로젝트 코드를 받아옵니다. 현 프로젝트는 브랜치 별로 코드 버전이 관리되어 지금은 v1 브랜치의 코드를 받아오기 위해 -b 옵션을 사용합니다.
3. API 프로젝트 코드 다운로드가 잘 되었는 지 디렉토리 확인을 합니다.
4. Fastcampus-api-deploy 디렉토리로 이동합니다.

Node.js 환경 구축

```
1. ec2-user@ip-172-31-25-131:~ (bash)
a1@1ui-MacBook-Air:~$
```

`npm install -g pm2`

`npm install`

`pm2 start bin/www --name WAS`

`pm2 list`

`pm2 show WAS`

Node.js 환경 구축

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~\$

- 1) npm install -g pm2
- 2) npm install
- 3) pm2 start bin/www --name WAS
- 4) pm2 list
- 5) pm2 show WAS

1. NPM을 통해 pm2(nodejs process manager)를 -g 옵션으로 글로벌 모드(전체 적용)로 설치합니다.
2. npm install 명령어를 입력했을 때 뒤에 패키지명이 없다면 프로젝트 내에 package.json 파일 안의 내용을 확인하여 관련 모듈들을 해당 프로젝트 내에 설치합니다(-g 옵션과의 차이).
3. pm2를 통해 node.js로 작성된 API 프로젝트를 실행합니다.
4. pm2로 관리되는 프로세스들을 확인합니다.
5. 방금 pm2로 실행한 WAS로 명시한 프로세스의 상세 정보를 확인합니다.

Node.js 환경 구축

아래 링크를 클릭하여 POSTMAN으로 작성된
API 문서를 받아주세요.

[https://www.getpostman.com/collections/
da6d82dc6fabdc2f0816](https://www.getpostman.com/collections/da6d82dc6fabdc2f0816)

Node.js 환경 구축

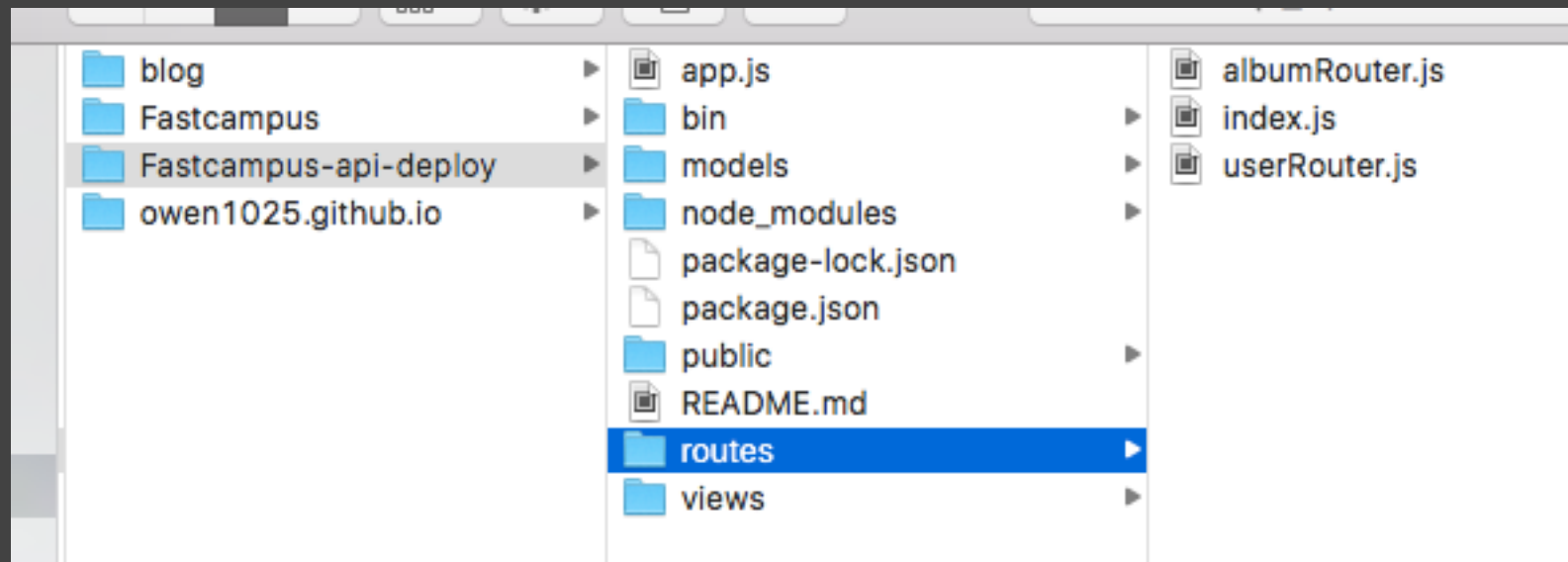
The screenshot shows the Postman application interface. On the left, there's a sidebar with 'History' and 'Collections' tabs. The 'Collections' tab is active, showing a collection named 'Fastcampus-api-deploy-v1' with 2 requests. The first request, 'GET /album', is selected. The main panel shows the details of this request. The URL is 'ec2-13-209-15-68.ap-northeast-2.compute.amazonaws.com:8080/album', which is highlighted with a red box. The 'Send' button is visible next to the URL. Below the URL bar, there are tabs for 'Authorization', 'Headers', 'Body', 'Pre-request Script', and 'Tests'. The 'Type' is set to 'GET'. The 'Response' tab is currently selected, but it's empty. A large text overlay is present in the center-right of the interface, providing instructions in Korean.

해당 주소를 방금 생성한 EC2의 DNS 주소로 바꿔서 Send 버튼을 누르시면 API 호출이 됩니다.

- ex) ec2-13-209-15-68.ap-northeast-2.compute.amazonaws.com:8080

Hit the Send button to get a response.

Node.js 프로젝트 구조



Fastcampus-api-deploy

- app.js
- bin
 - www
- routes
- models
- package.json

API 프로젝트 이름

- node.js 라우팅, 미들웨어 등 설정 파일
- bin/www HTTP 서버 설정
 - 해당 파일을 시작점으로 PM2를 통해 WAS 실행
- API 컨트롤러 영역(URL 라우팅 별 처리)
- API 비즈니스 로직 처리 영역
- 패키지 정보 내역
 - 해당 프로젝트에서 사용하는 모듈들의 의존성 관리
 - 프로젝트의 정보 기록
 - npm install 시 해당 파일을 기반으로 모듈 설치

Node.js 환경 구축

WAS(Node.js) 설치/시작, 로드밸런서 생성/설정.pdf를
참고하시면서 웹 서버(Nginx) 구축 때처럼 2대의 WAS를
하나의 로드밸런서(ELB)에 연결해봅시다.

- Auto scaling group은 따로 안하셔도 됩니다!

WAS 영역 로드밸런서(ELB) 구축



서비스 ▾

리소스 그룹 ▾



owen ▾

서울 ▾

지원 ▾

1. 로드 밸런서 정의
2. 보안 그룹 할당
3. 보안 설정 구성
4. 상태 검사 구성
5. EC2 인스턴스 추가
6. 태그 추가
7. 검토

단계 1: 로드 밸런서 정의

기본 구성

이 마법사는 새 로드 밸런서를 설정하는 방법을 안내합니다. 먼저 새 로드 밸런서를 다른 로드 밸런서와 구별할 수 있도록 고유한 이름을 지정하는 것부터 시작합니다. 또한 로드 밸런서에 포트 및 프로토콜도 구성해야 합니다. 클라이언트의 트래픽은 로드 밸런서 포트부터 EC2 인스턴스의 포트까지 라우팅됩니다. 기본적으로 로드 밸런서는 포트 80에서 표준 웹 서버로 구성되어 있습니다.

로드 밸런서 이름: was-lb

LB 생성할 VPC: 내 기본 VPC (172.31.0.0/16)

해당 로드밸런서로 들어온 HTTP(8080포트) 요청을

고급 VPC 구성 활성화: ☐

리스너 구성:

로드 밸런서 프로토콜

HTTP

로드 밸런서 포트

8080

인스턴스 프로토콜

HTTP

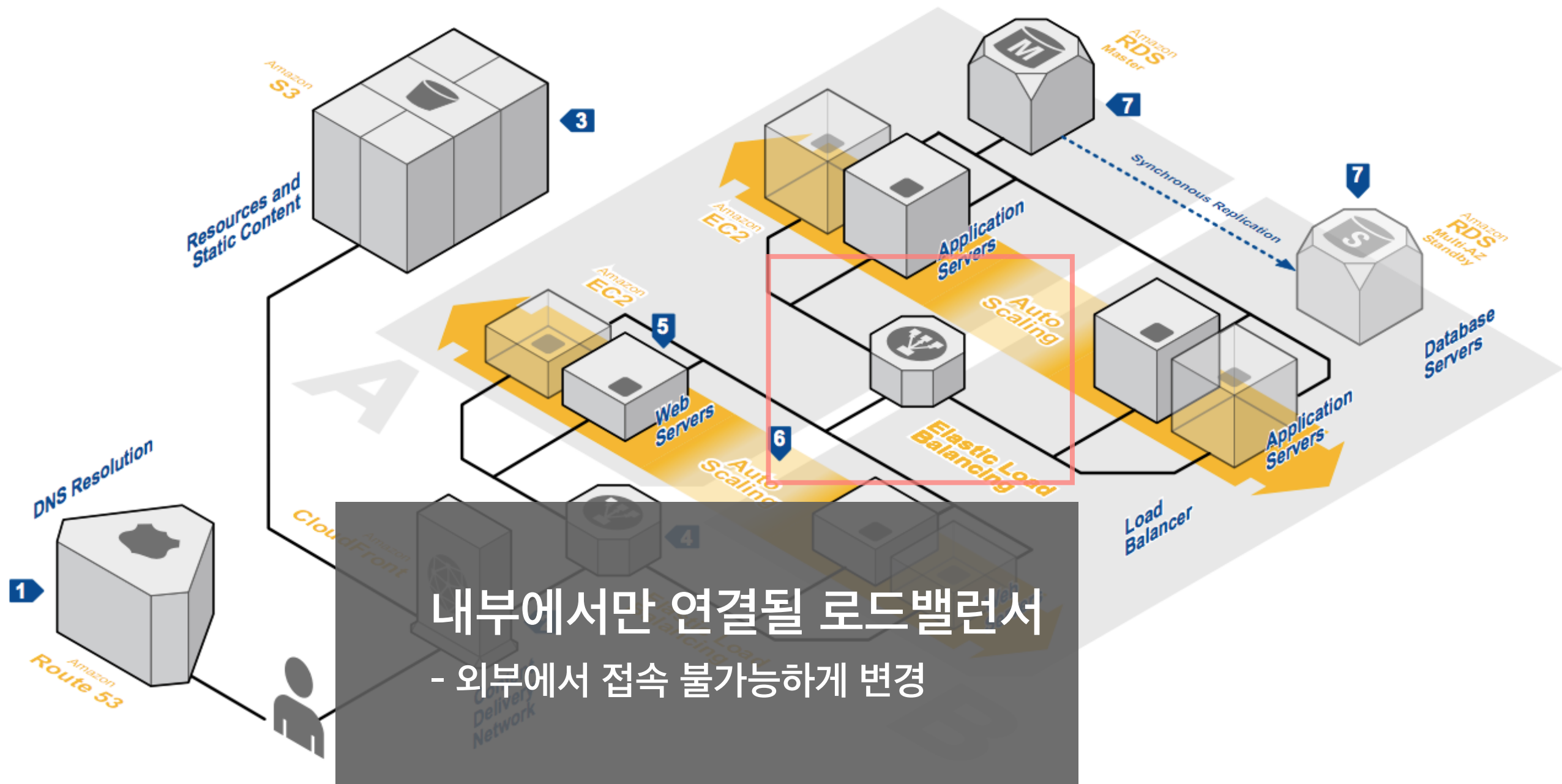
인스턴스 포트

8080

추가

연결된 인스턴스로 전달
(HTTP:8080)

WAS 영역



WAS 영역 로드밸런서(ELB) 구축

단계 1: 로드 밸런서 정의

기본 구성

이 마법사는 새 로드 밸런서를 설정하는 방법을 안내합니다. 먼저 새 로드 밸런서를 다른 로드 밸런서와 구별할 수 있도록 포트부터 EC2 인스턴스의 포트까지 라우팅됩니다. 기본적으로 로드 밸런서는 포트 80에서 표준 웹 서버로 구성되어

로드 밸런서 이름: was-lb

LB 생성할 VPC: 내 기본 VPC (172.31.0.0/16)

내부 로드 밸런서 생성: ☐ (자세히 알아보기)

고급 VPC 구성 활성화: ☐

리스너 구성:

로드 밸런서 프로토콜

HTTP

추가

내부 로드밸런서로 생성
- 외부에서 접속 불가능하게 변경

로드 밸런서 포트

8080

WAS 영역 내부 로드밸런서(ELB) 구축

1. 로드 밸런서 정의 2. 보안 그룹 할당 3. 보안 설정 구성 4. 상태 검사 구성 5. EC2 인스턴스 추가 6. 태그 추가 7. 검토

단계 1: 로드 밸런서 정의

기본 구성

이 마법사는 새 로드 밸런서를 설정하는 방법을 안내합니다. 먼저 새 로드 밸런서를 다른 로드 밸런서와 구별할 수 있도록 고유한 이름을 지정하는 것부터 시작합니다. 또한 로드 밸런서에 포트 및 프로토콜도 구성해야 합니다. 포트부터 EC2 인스턴스의 포트까지 라우팅됩니다. 기본적으로 로드 밸런서는 포트 80에서 표준 웹 서버로 구성되어 있습니다.

로드 밸런서 이름: was-internal-lb

LB 생성할 VPC: 내 기본 VPC (172.31.0.0/16)

내부 로드 밸런서 생성: ☒ (자세히 알아보기)

고급 VPC 구성 활성화: ☐

리스너 구성:

로드 밸런서 프로토콜

로드 밸런서 포트

인스턴스 프로토콜

인스턴스 포트

HTTP

8080

HTTP

8080

추가

ELB 생성시 내부 로드 밸런서 생성을 체크합니다.
(나머지 구성은 모두 동일합니다.)

내부 로드밸런서(ELB) 테스트

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~\$

- nginx가 설치된 EC2 인스턴스에 SSH로 접속 이후

curl {앞서 생성한 ELB private DNS:8080}/list

- ex. curl http://internal-was-internal-lb-862573307.ap-northeast-2.elb.amazonaws.com:8080/list

내부 로드밸런서(ELB) 테스트

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~\$

- nginx가 설치된 EC2 인스턴스에 SSH로 접속 이후

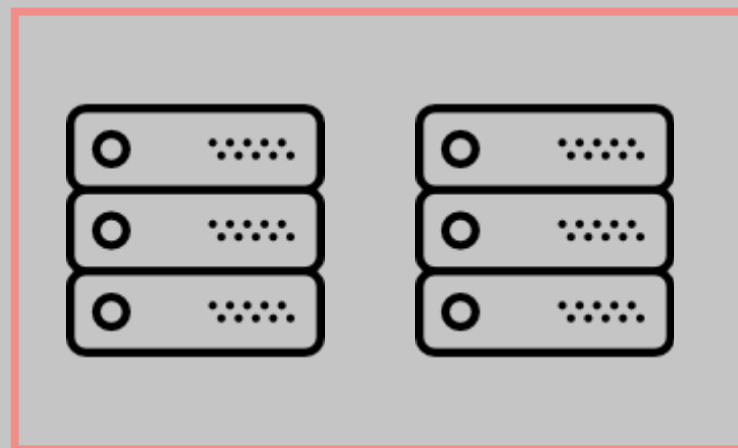
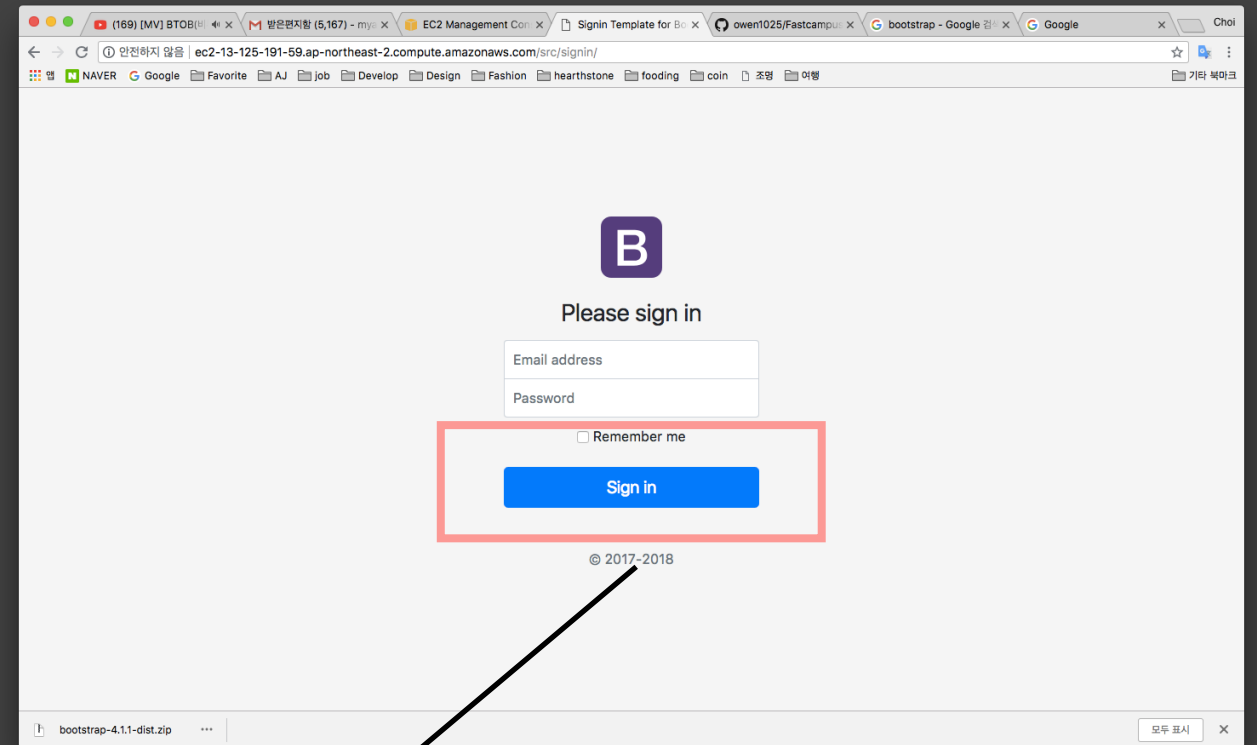
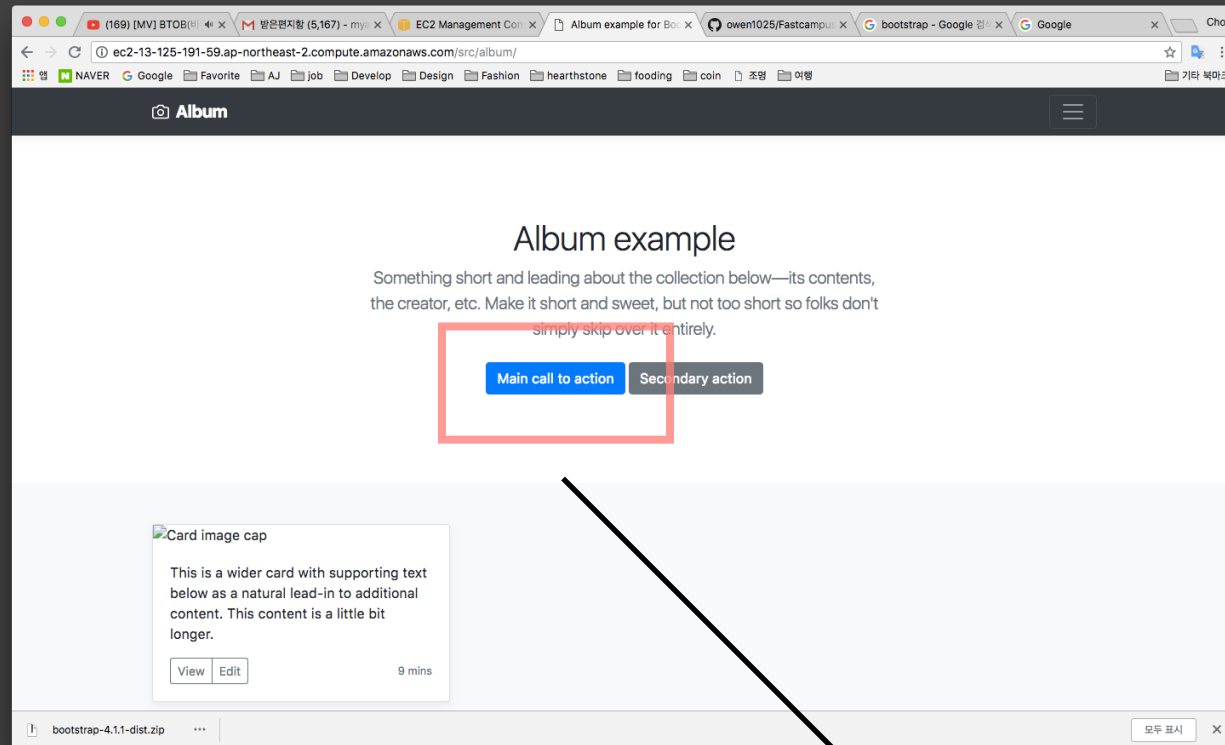
curl {앞서 생성한 ELB private DNS:8080}/album

- ex. curl http://internal-was-internal-lb-862573307.ap-northeast-2.elb.amazonaws.com:8080/list

curl

- command line용 데이터 전송 툴
- HTTP/HTTPS/FTP/LDAP/SCP/TELNET/SMTP/POP3 등 다양한 프로토콜 지원
- Linux, Mac OS에 기본 탑재

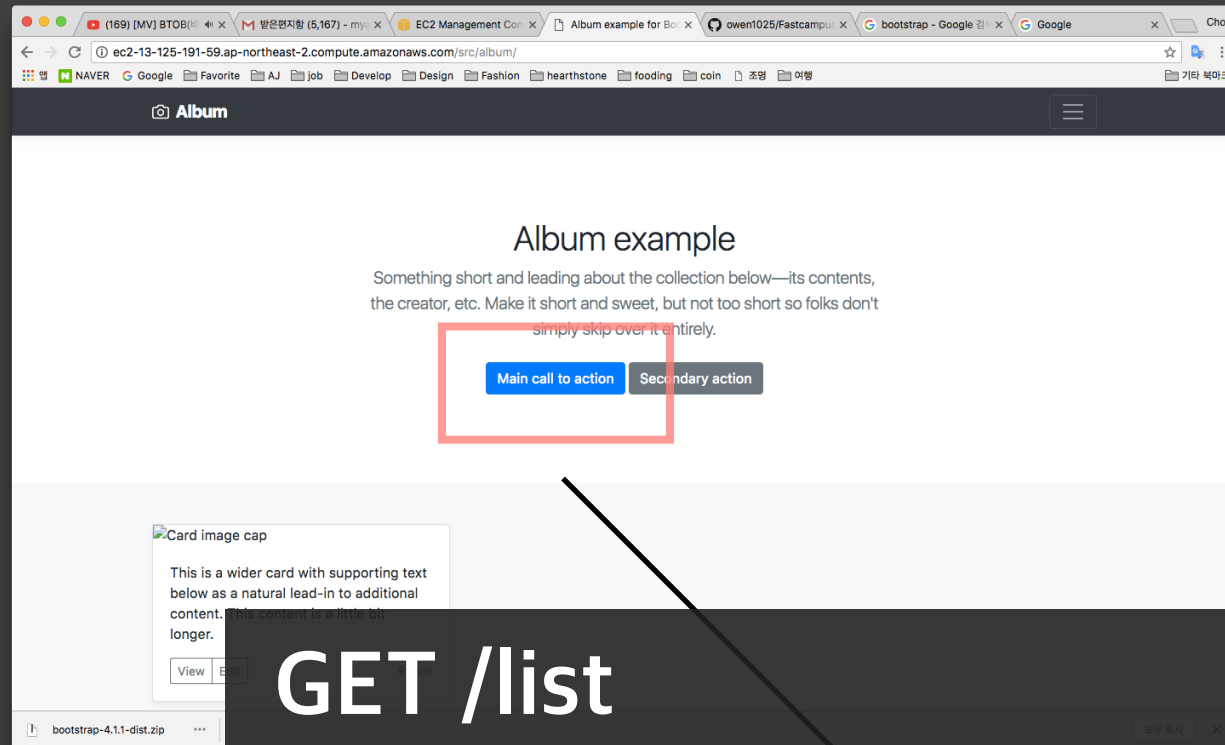
웹 서버 - WAS 연결 테스트



WAS(Node.js)

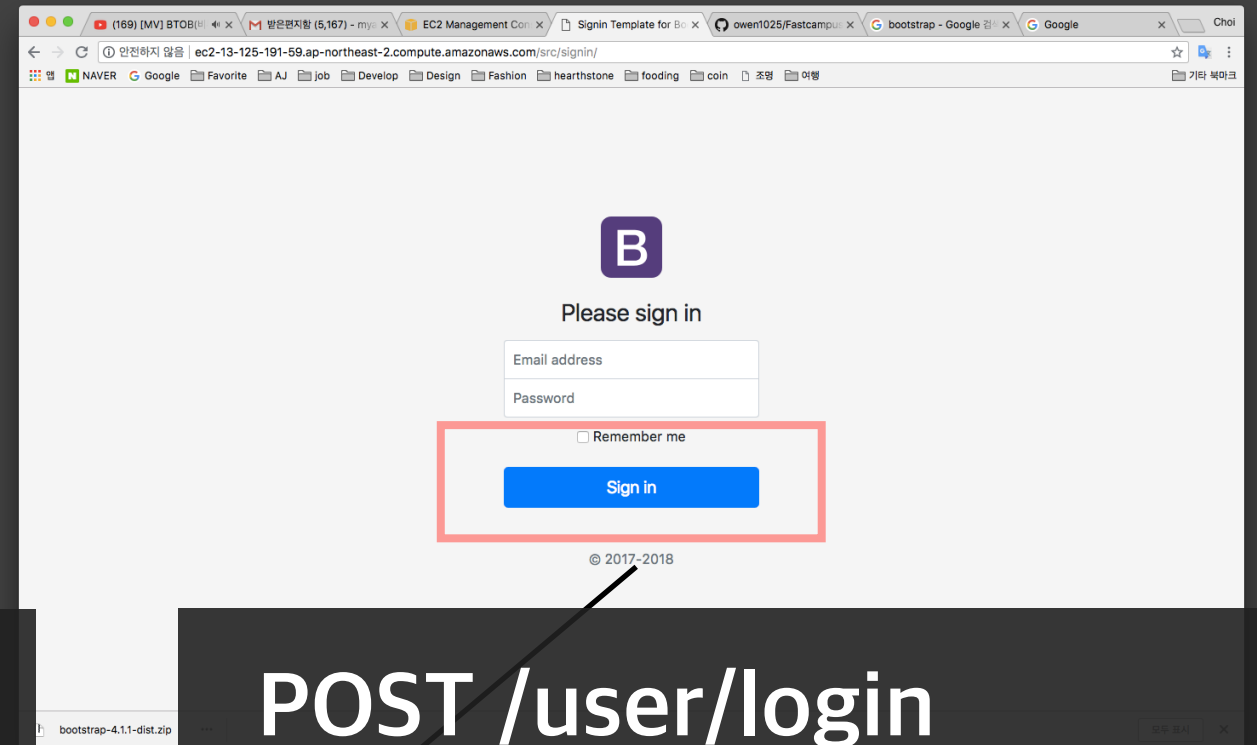
- 동적 데이터 처리

웹 서버 - WAS 연결 테스트



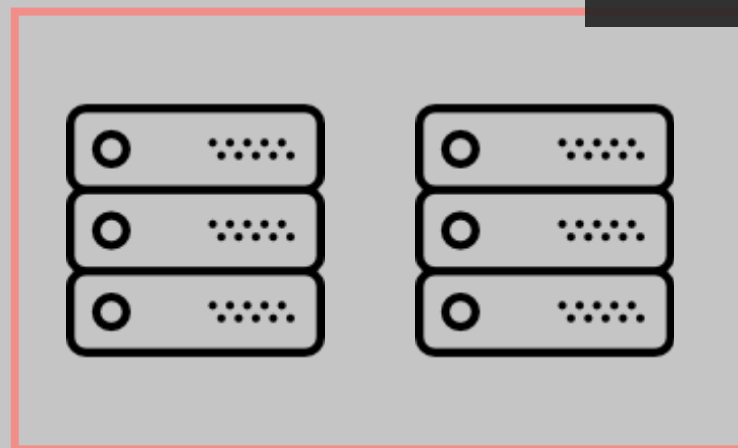
GET /list

- 리스트 만들게 데이터 주세요



POST /user/login

- email하고 password 정보 보내
니 맞는가 확인 좀 해줘



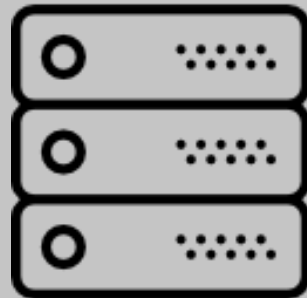
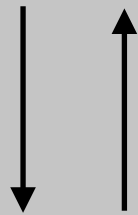
WAS(Node.js)

- 동적 데이터 처리

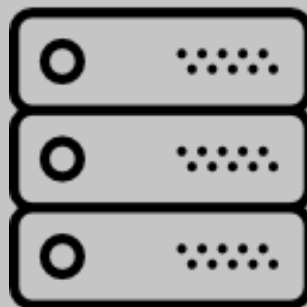
웹 서버 - WAS 연결 테스트



Client(웹 브라우저)



웹 서버 영역(ELB + EC2 N대)



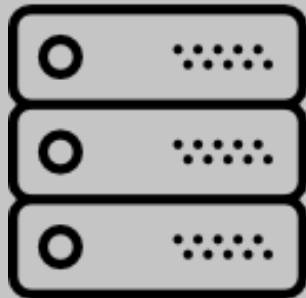
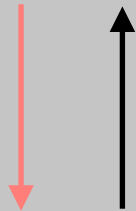
WAS 영역(ELB + EC2 N대)

웹 서버 - WAS 연결 테스트

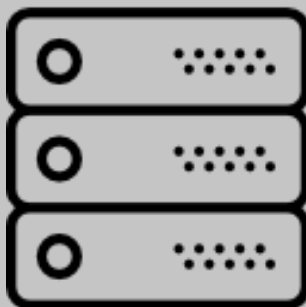


Client(웹 브라우저)

<http://elb-nginx-dns/album>



웹 서버 영역(ELB + EC2 N대)

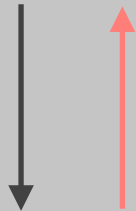


WAS 영역(ELB + EC2 N대)

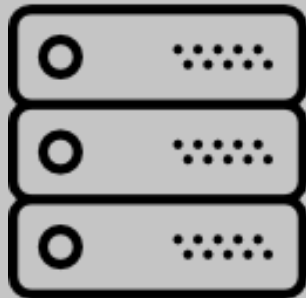
웹 서버 - WAS 연결 테스트



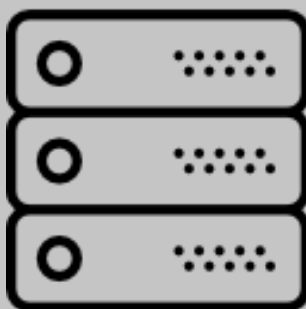
Client(웹 브라우저)



Fastcampus-web-deploy/album/index.html 응답



웹 서버 영역(ELB + EC2 N대)



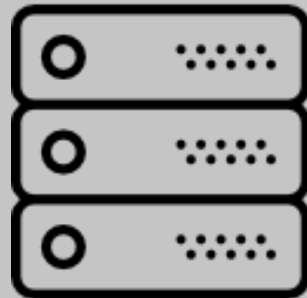
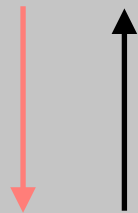
WAS 영역(ELB + EC2 N대)

웹 서버 - WAS 연결 테스트

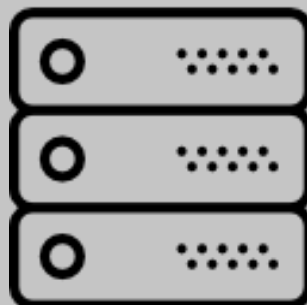


Client(웹 브라우저)

GET http://elb-nginx-dns/api/list



웹 서버 영역(ELB + EC2 N대)



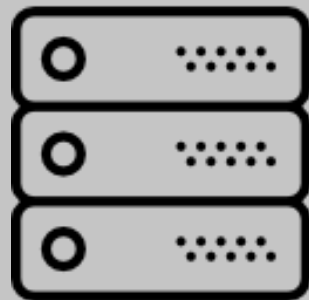
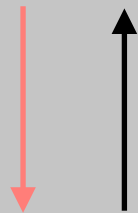
WAS 영역(ELB + EC2 N대)

웹 서버 - WAS 연결 테스트

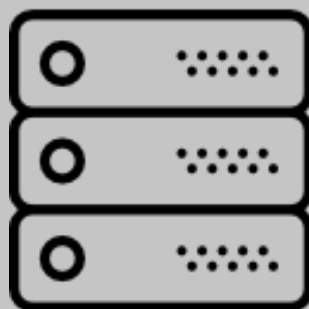


Client(웹 브라우저)

GET http://elb-nginx-dns/**api**/list



NginX는 해당 URL을 **API 요청**으로 인식하여 **proxy_pass**를 통해 내부망으로 WAS 영역에 요청



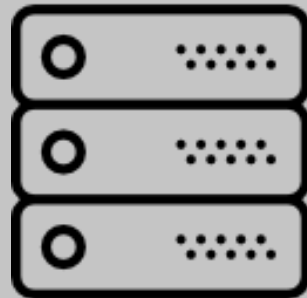
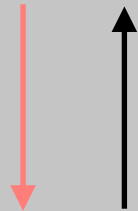
WAS 영역(ELB + EC2 N대)

웹 서버 - WAS 연결 테스트

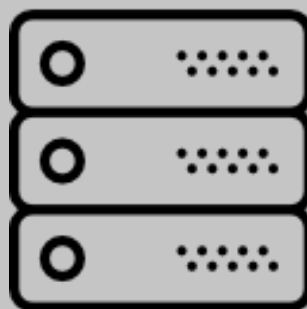


Client(웹 브라우저)

GET http://elb-nginx-dns/api/list



http://elb-nginx-domain:80/api/list
-> http://{was-lb-private-dns}:8080/list



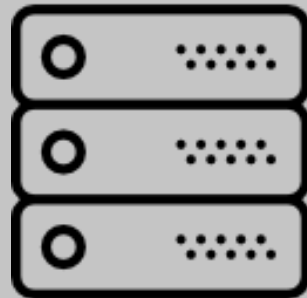
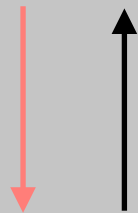
WAS 영역(ELB + EC2 N대)

웹 서버 - WAS 연결 테스트



Client(웹 브라우저)

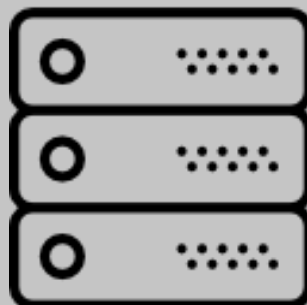
GET http://elb-nginx-dns/api/list



웹 서버 영역(ELB + EC2 N대)



GET /list



WAS 영역(ELB + EC2 N대)

웹 프로젝트 변경

```
1. ec2-user@ip-172-31-25-131:~ (bash)
a1@lui-MacBook-Air:~$
```

- nginx가 설치된 EC2 인스턴스에 SSH로 접속 이후

```
cd /usr/share/nginx/html/

ls -al

rm -rf Fastcampus-web-deploy/

git clone -b v1 https://github.com/owen1025/Fastcampus-
web-deploy.git
```

웹 프로젝트 변경

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~\$

1) `cd /usr/share/nginx/html/`

2) `ls -al`

3) `rm -rf Fastcampus-web-deploy/`

4) `git clone -b v1 https://github.com/owen1025/Fastcampus-web-deploy.git`

1) 기존 Fastcampus-web-deploy가 다운로드 된 디렉토리로 이동

2) 디렉토리 내 파일 리스트 보기

3) 기존 프로젝트 삭제하기

4) 서버와 통신하는 Ajax 코드가 들어있는 v1 브랜치에 있는 웹 프로젝트 다운로드

웹 프로젝트 변경

```
1. ec2-user@ip-172-31-25-131:~ (bash)
a1@1ui-MacBook-Air:~$
```

vi Fastcampus-web-deploy/resources/js/common.js

- BASE_URL = '{nginx-lb-dns}/api/'로 변경
 - ex. BASE_URL = 'http://nginx-lb-2007386994.ap-northeast-2.elb.amazonaws.com/api/'
 - esc 클릭 후 :wq 입력 뒤 vi 저장하고 종료

Nginx proxy_pass 설정

```
1. ec2-user@ip-172-31-25-131:~ (bash)  
a1@1ui-MacBook-Air:~$
```

- 1) `cd /etc/nginx/`
- 2) `cp nginx.conf nginx-copy.conf`
- 3) `vi nginx.conf`

- 1) nginx.conf가 위치한 디렉토리로 이동
- 2) nginx.conf 수정 전 원본 파일 복사하기
- 3) vi 편집기로 nginx.conf 열기

Nginx proxy_pass 설정

```
server {
```

```
...
```

```
location ~* /(album|signin) {
```

```
    root /usr/share/nginx/html/Fastcampus-web-deploy/page;
```

```
}
```

```
location /api {
```

```
    rewrite /api/(.*) /$1 break;
```

```
    proxy_pass http://internal-was-internal-lb-862573307.ap-  
northeast-2.elb.amazonaws.com:8080;
```

```
    proxy_set_header Host $host;
```

```
}
```

```
...
```

```
}
```

Nginx proxy_pass 설정

```
server {
```

```
...
```

해당 내용 추가하기

- proxy_pass 뒤 URL은 WAS(Node.js)가 운영 중인 인스턴스와 연결된 ELB의 private dns
- ex. <http://internal-was-internal-lb-862573307.ap-northeast-2.elb.amazonaws.com:8080>

```
location /api {
```

```
    rewrite /api/(.*) /$1 break;
```

```
    proxy_pass http://internal-was-internal-lb-862573307.ap-  
northeast-2.elb.amazonaws.com:8080;
```

```
    proxy_set_header Host $host;
```

```
}
```

```
...
```

```
}
```

Nginx proxy_pass 설정

```
1. ec2-user@ip-172-31-25-131:~ (bash)  
a1@1ui-MacBook-Air:~$
```

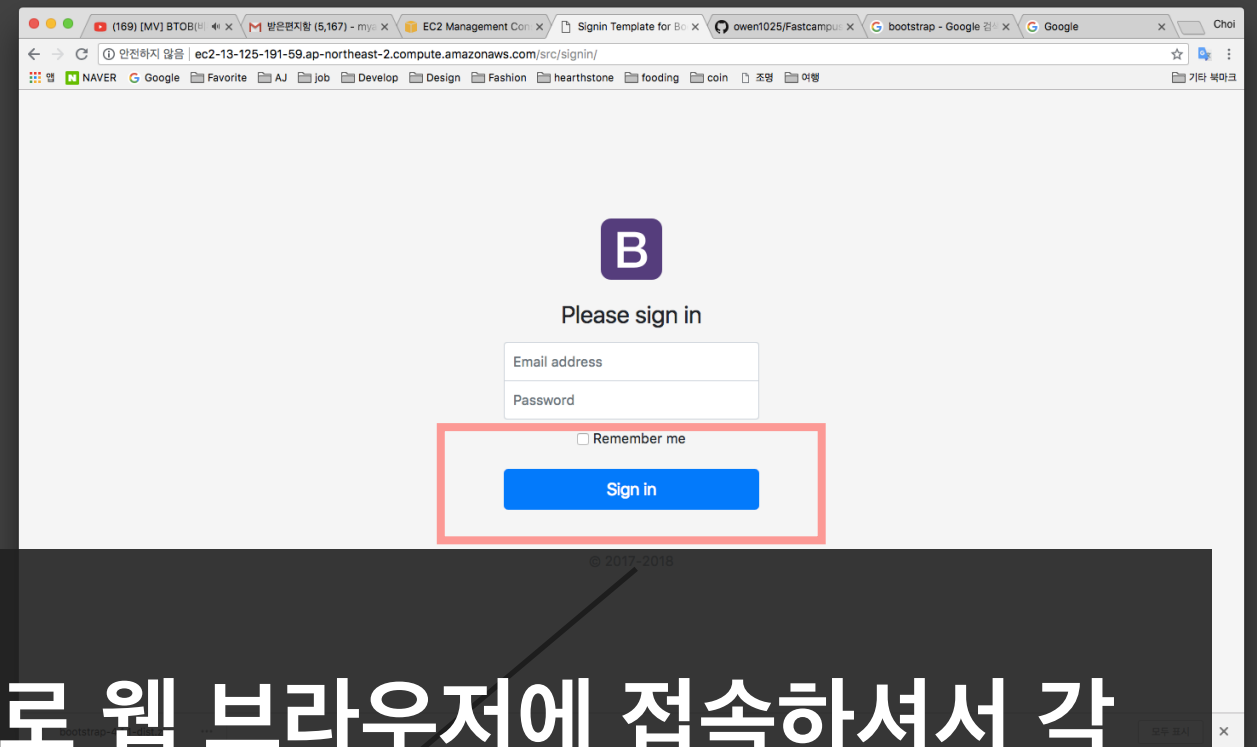
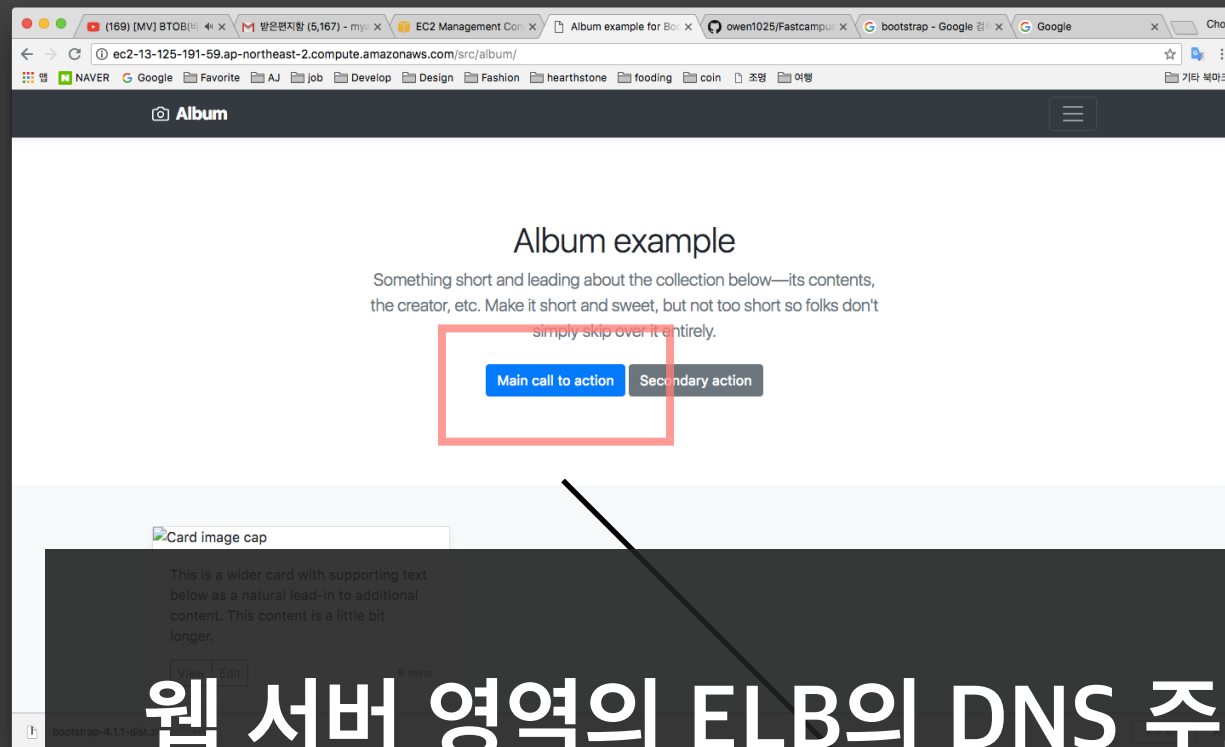
해당 내용을 추가하시고 :wq로 vi를 저장하고 종료합니다.

nginx.conf 수정 내역을 적용하기 위해 아래 명령어

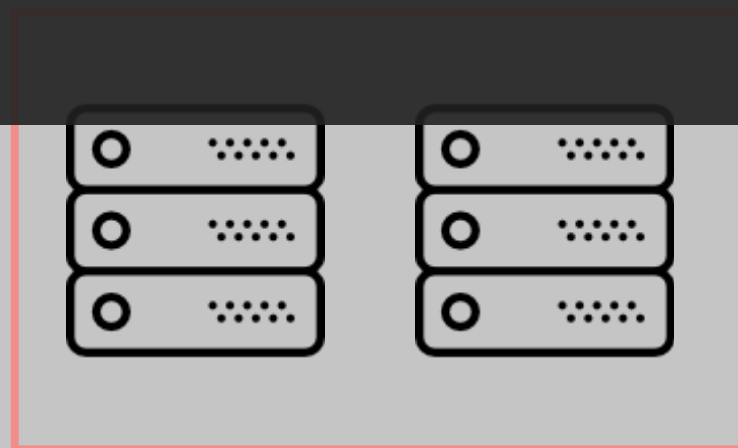
service nginx reload

를 실행합니다.

웹 서버 - WAS 연결 테스트



웹 서버 영역의 ELB의 DNS 주소로 웹 브라우저에 접속하셔서 각 버튼을 클릭하여 API 호출과 클라이언트 적용이 정상적으로 작동하는 지 확인합니다.



WAS(Node.js)

- 동적 데이터 처리

Nginx proxy_pass 설정

```
location /api {  
    rewrite /api/(.*) /$1 break;  
    proxy_pass http://{was-lb-private-dns}:8080;  
    proxy_set_header Host $host;  
}
```

Nginx proxy_pass 설정

```
GET http://{nginx-lb-dns}/api/user/login  
- 해당 요청의 구문을 처리
```

```
location /api {  
    rewrite /api/(.*) /$1 break;  
    proxy_pass http://{was-lb-private-dns}:8080;  
    proxy_set_header Host $host;  
}
```

Nginx proxy_pass 설정

proxy_pass

- 뒤에 오는 dns/ip 주소로 요청 전달

location /api- { Reverse proxy

rewrite /api/(.*) /\$1 break;

proxy_pass http://{was-lb-private-dns}:8080;

proxy_set_header Host \$host;

}

Nginx proxy_pass 설정

rewrite (= 재작성)

- 들어온 요청의 URL의 규칙을 변경

location /api {

rewrite /api/(.*) /\$1 break;

proxy_pass http://{was-lb-private-dns}:8080;

proxy_set_header Host \$host;

}

Nginx proxy_pass 설정

rewrite (= 재작성)

- rewrite {a}를 {b}로 변경
- proxy_pass에서 설정한 URL의 뒤에 붙음
- http://{nginx-lb-dns}/api/user/login => \$1(== user/login)로 치환

=> /user/login만 가져오게 됨

=> 보내는 요청은 http://{was-lb-private-dns}:8080/user/login

```
rewrite /api/(.*) /$1 break;
```

```
proxy_pass http://{was-lb-private-dns}:8080;
```

```
proxy_set_header Host $host;
```

```
}
```

Nginx proxy_pass 설정

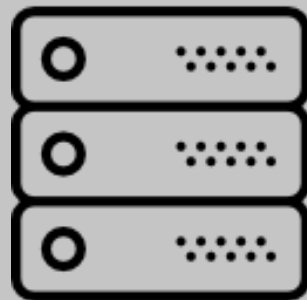
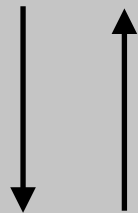
proxy_set_header

```
location /api {  
    - 전달할 요청의 header 값 설정  
    - 웹 서버 영역으로 요청받은 그대로 전달  
    proxy_pass http://{was-lb-private-dns}:8080;  
    proxy_set_header Host $host;  
}
```

웹 서버 - WAS 연결 테스트

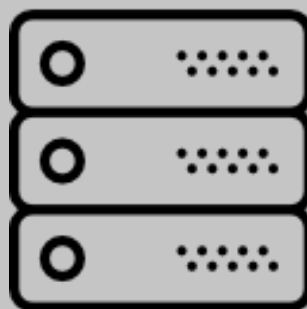


Client(웹 브라우저)



웹 서버 영역(ELB + EC2 N대)

`tail -f /var/log/nginx/access.log`



WAS 영역(ELB + EC2 N대)

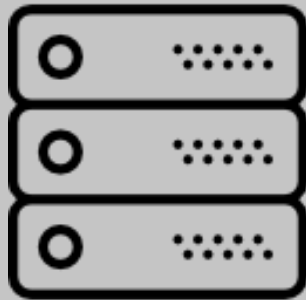
웹 서버 - WAS 연결 테스트



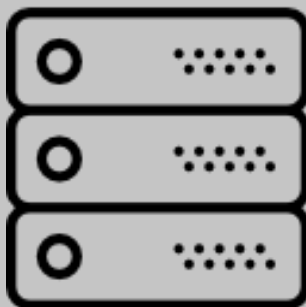
Client(웹 브라우저)



웹 서버 영역(ELB + EC2 N대)



`tail -f /var/log/nginx/access.log`
=> 한 곳으로 **로그 관리**가 가능해진다.

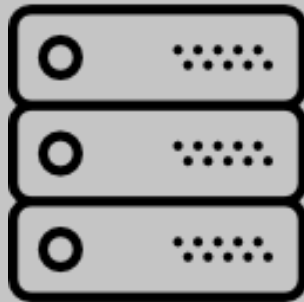
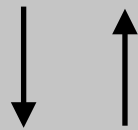


WAS 영역(ELB + EC2 N대)

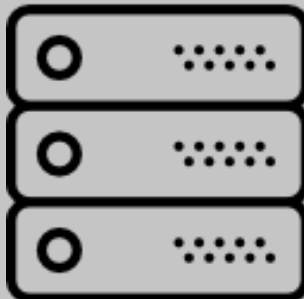
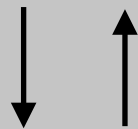
웹 서버 - WAS - DB 연결



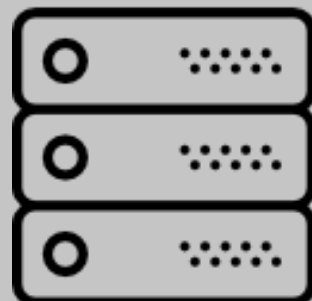
Client(웹 브라우저)



웹 서버 영역(ELB + EC2 N대)

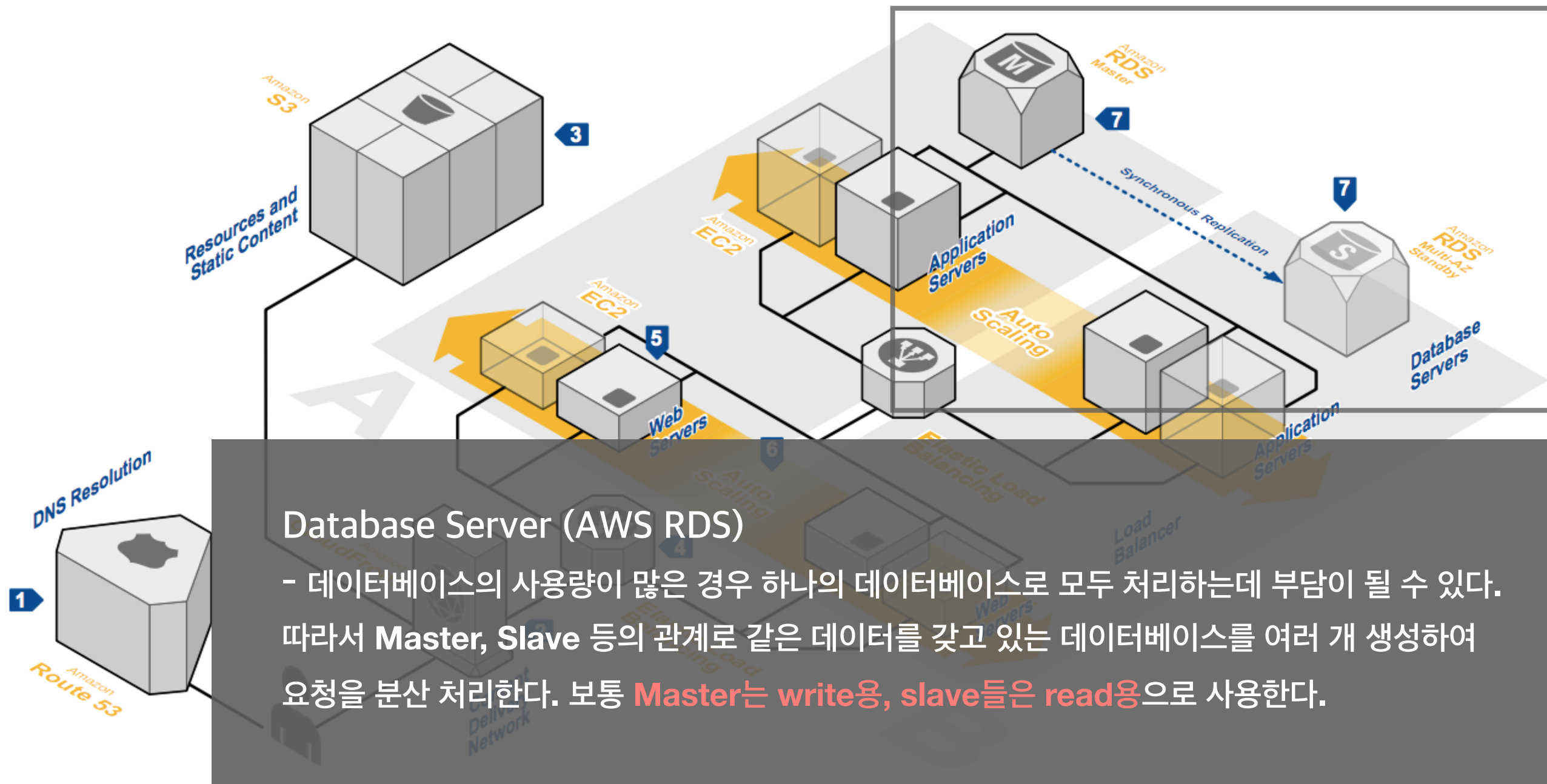


WAS 영역(ELB + EC2 N대)

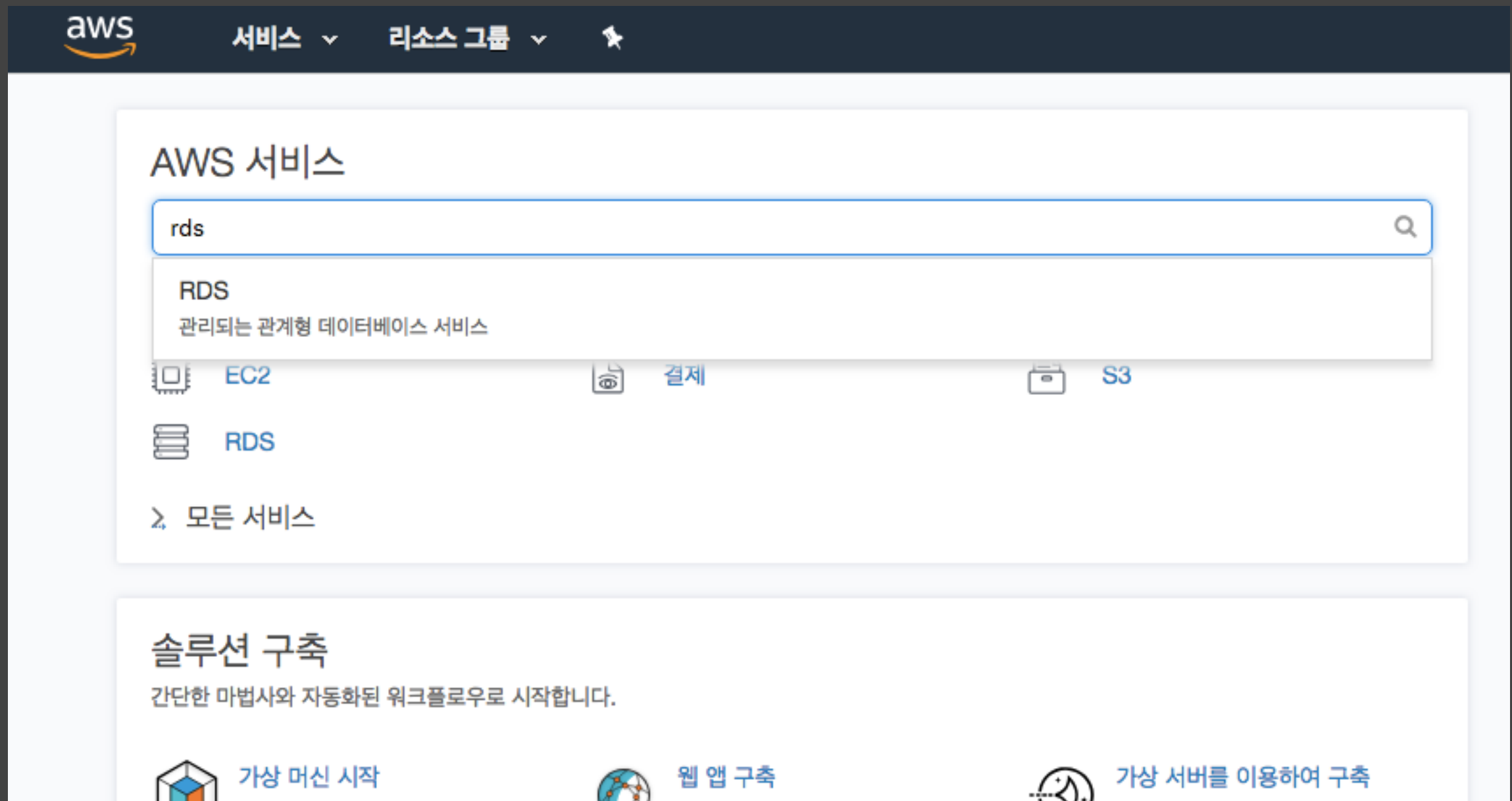


DB 영역(MySQL, AWS RDS)

AWS RDS 설정



AWS RDS 설정



- AWS 콘솔 메인화면에서 AWS 서비스 탭에 rds를 검색하여 RDS 서비스를 클릭합니다.

AWS RDS 설정

The screenshot shows the AWS RDS console interface. On the left is a navigation menu with options like Dashboard, 인스턴스, 클러스터, 성능 개선 도우미, 스냅샷, 예약 인스턴스, 서버넷 그룹, 파라미터 그룹, 옵션 그룹, 이벤트, 이벤트 구독, and 알림. The main content area is titled 'Amazon RDS' and shows resource usage for the Asia Pacific (Seoul) region. It lists metrics for DB instances (1/40), parameter groups (1), option groups (1), snapshots (15), and events (19). A 'Create database' button is highlighted with a red box. Below the button, there is a note about the service and a reference to the Asia Pacific (Seoul) region. At the bottom, there is a '서비스 상태' (Service Status) section with a link to '서비스 상태 대시보드 보기' (View Service Status Dashboard).

Amazon RDS

Dashboard

인스턴스

클러스터

성능 개선 도우미

스냅샷

예약 인스턴스

서버넷 그룹

파라미터 그룹

옵션 그룹

이벤트

이벤트 구독

알림

Asia Pacific (Seoul) 리전에서 사용 중인 Amazon RDS 리소스 정보(사용량/할당량)

DB 인스턴스 (1/40)

할당된 스토리지 (20.00 GB/100.00 TB)

여기를 클릭하여 DB 인스턴스 한도 높이기

예약 인스턴스 (0/40)

스냅샷 (15)

수동 (1/100)

자동화 (1)

최근 이벤트 (19)

이벤트 구독 (0/20)

파라미터 그룹 (1)

기본값 (1)

사용자 지정 (0/100)

옵션 그룹 (1)

기본값 (1)

사용자 지정 (0/20)

서버넷 그룹 (1/50)

지원되는 플랫폼 VPC

기본 네트워크 vpc-ec151b84

Create database

Amazon Relational Database Service(RDS)는 클라우드에 데이터베이스를 쉽게 생성, 운영 및 확장할 수 있는 웹 서비스입니다.

S3에서 복원

Create database

참고: DB 인스턴스가 Asia Pacific (Seoul) 리전에서 시작합니다

서비스 상태

서비스 상태 대시보드 보기

- RDS 화면 중앙에 위치한 Create database 버튼 클릭

AWS RDS 설정

엔진 선택

엔진 옵션


☐ Amazon Aurora
Amazon Aurora

☒ MySQL


☐ MariaDB


☐ PostgreSQL


☐ Oracle
ORACLE

☐ Microsoft SQL Server


MySQL

MySQL은 전 세계에서 가장 많이 사용되는 오픈 소스 데이터베이스입니다. RDS에서 MySQL은 데이터베이스의 컴퓨팅 리소스 또는 스토리지 용량을 쉽게 확장할 수 있는 유연성을 갖춘 MySQL 커뮤니티 에디션의 풍부한 기능을 제공합니다.

- 최대 16TiB 크기의 데이터베이스를 지원
- 인스턴스는 최대 32개의 vCPU 및 244GiB 메모리 제공
- 자동 백업 및 특정 시점으로 복구 지원
- 리전 간 읽기 전용 복제 지원

☒ RDS 프리 티어에 적용되는 옵션만 사용 [정보](#)

취소 다음 단계

· 엔진 선택

- 저희 수업에선 MySQL로 Database 서버를 구축합니다.
- 하단에 위치한 프리 티어에 적용되는 옵션 사용을 체크합니다.

AWS RDS 설정

인스턴스 사양

다음을 사용하여 DB 인스턴스의 월별 청구액을 추산할 수 있습니다. [AWS 월 사용량 계산기](#).

DB 엔진

MySQL Community Edition

라이선스 모델 [정보](#)

general-public-license ▼

DB 엔진 버전 [정보](#)

mysql 5.6.39 ▼



알려진 문제/제한

알려진 문제/제한을(를) 검토하여 특정 데이터베이스 버전과의 잠재적인 호환성 문제를 확인합니다.



프리 티어

Amazon RDS 프리 티어는 단일 db.t2.micro 인스턴스와 최대 20GiB 스토리지를 제공하므로 새로운 AWS 고객은 Amazon RDS를 직접 체험해 볼 수 있습니다. [여기](#)에서 RDS 프리 티어 및 인스턴스 제약 조건에 대해 자세히 알아보십시오.

☒ RDS 프리 티어에 적용되는 옵션만 사용 [정보](#)

DB 인스턴스 클래스 [정보](#)

db.t2.micro — 1 vCPU, 1 GiB RAM ▼

· 인스턴스 사양

- 실습을 위해 프리 티어 사양(기본값) 사용
- 개발팀에서 따로 사용하는 DB버전으로 변경

AWS RDS 설정

다중 AZ 배포 정보

☐ 다른 영역에 복제본 만들기

데이터 중복을 제공하고, I/O 중지를 없애고, 시스템 백업 중에 지연 시간 스파이크 없이 다른 가용 영역(AZ)에 복제본을 생성합니다.

☒ 아니요

스토리지 유형 정보

범용(SSD)

할당된 스토리지

20

GiB

(최소: 20GiB, 최대: 20GiB) 더 많이 할당된 스토리지 개선될 수 있음 IOPS 성능.

· 다중 AZ 배포

- 다른 가용 영역(머신)에 대기 RDS에 동기식 복제
- 고가용성 지원
- 자동 장애 조치 수행
- 프리 티어에선 사용할 수 없는 옵션(과금)

AWS RDS 설정

설정

DB 인스턴스 식별자 [정보](#)

현재 리전에서 AWS 계정이 소유하는 모든 DB 인스턴스에 대해 고유한 이름을 지정합니다.

MySQL-RDS

DB 인스턴스 식별자는 대소문자를 구분하지 않지만 "mydbinstance"와 같이 모두 소문자로 저장됩니다. 1~63자의 영숫자 문자 또는 하이픈(SQL Server의 경우 1~15)으로 구성되어야 합니다. 첫 번째 문자는 글자이어야 합니다. 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.

마스터 사용자 이름 [정보](#)

마스터 사용자의 로그인 ID를 정의하는 영숫자 문자열을 지정합니다.

admin

마스터 사용자 이름은 문자로 시작해야 합니다. 1~16자의 영숫자 문자로 구성되어야 합니다.

마스터 암호 [정보](#)

암호 확인 [정보](#)

마스터 암호는 "mypassword"와 같이 8자 이상이어야 합니다.

• 설정

• DB 인스턴스 식별자

- RDS 인스턴스의 이름, MySQL-RDS를 입력합니다.

• 마스터 사용자 이름

- 통상적인 root 계정이라고 생각하시면 됩니다. 여기선 admin으로 입력합니다.

AWS RDS 설정

네트워크 및 보안

Virtual Private Cloud(VPC) 정보

VPC에서 이 DB 인스턴스의 가상 네트워킹 환경을 정의합니다.

기본 VPC(vpc-ec151b84)



해당 DB 서브넷 그룹이 있는 VPC만 나열됩니다.

서브넷 그룹 정보

선택한 VPC에서 DB 인스턴스가 어떤 서브넷과 IP 범위를 사용할 수 있는지를 정의하는 DB 서브넷 그룹.

default

퍼블릭 액세스 가능성 정보

☒ 예
DB 인스턴스를 호스팅하는 VPC 외부의 EC2 인스턴스 및 디바이스가 DB 인스턴스에 연결됩니다. DB 인스턴스에 연결할 수 있는 EC2 인스턴스와 디바이스를 지정하는 하나 이상의 VPC 보안 그룹을 선택해야 합니다.

☐ 아니요
DB 인스턴스에 퍼블릭 IP 주소가 할당되지 않습니다. VPC 외부의 EC2 인스턴스 또는 디바이스는 연결할 수 있습니다.

가용 영역 정보

기본 설정 없음

VPC 보안 그룹

보안 그룹에는 DB 인스턴스에 액세스해야 하는 모든 EC2 인스턴스와 디바이스의 연결을 인증하는 규칙이 있습니다.

☒ 새로운 VPC 보안 그룹 만들기

☐ 기존 VPC 보안 그룹 사용

• 네트워크 및 보안

• VPC 설정

- 해당 인스턴스(RDS)의 가상 네트워크 환경에 대한 옵션입니다. 기본 값(기본 VPC)를 사용합니다.

• 서브넷 그룹

- 위에서 설정한 VPC에서 인스턴스(RDS)가 어떤 서브넷과 IP 범위를 사용할 수 있는 지 정의합니다. 기본 값(default)를 사용합니다.

• 퍼블릭 액세스 가능성

- 다른 VPC, 다른 클라우드 환경, 로컬 환경 등 해당 인스턴스가 위치한 VPC 외부에서 접근할 가능성이 있다면 (예)를 체크합니다. 실 서비스 운영 중이고 DB 서버와 연결할 WAS가 같은 VPC 내에서 통신하는 경우는 (아니요)를 체크하여 private IP로 통신합니다.

AWS RDS 설정

· 백업


· 백업 보존 기간

- 백업 데이터 유지 기간입니다. 최대 35일까지 설정할 수 있습니다. 여기서 지정한 날짜 이전까지 되돌릴 수 있습니다.

· 백업 기간

- 백업 시간입니다. 시간을 선택하면 해당 시간에 백업을 수행합니다. 기간(ex. 30분)에 설정한 시간보다 늦게 끝날 수 있습니다. 기간을 설정하는 이유는 다음 페이지 유지 관리 기간 시간과 겹치지 않게 하기 위해서 입니다.

백업

 자동 백업 및 DB 스냅샷은 현재 InnoDB 엔진에만 적용됩니다. 참조하십시오. [링크](#)

백업 보존 기간 [정보](#)

Amazon RDS가 이 DB 인스턴스의 자동 백업을 유지해야 합니다.

7 일

▼

백업 기간 [정보](#)

☐ 기간 선택

☒ 기본 설정 없음

☒ 스냅샷으로 태그 복사

AWS RDS 설정

유지 관리

자동 마이너 버전 업그레이드 [정보](#)

- ☒ 마이너 버전 자동 업그레이드 사용
새 마이너 버전이 릴리스될 때 자동 업그레이드를 활성화
- ☐ 마이너 버전 자동 업그레이드 사용 안 함

유지 관리 기간 [정보](#)

보류 중인 수정 사항 또는 Amazon RDS가 DB 인스턴스에

- ☐ 기간 선택
- ☒ 기본 설정 없음

· 유지 관리

· 자동 마이너 버전 업그레이드

- 자동으로 마이너 버전을 업데이트하는 옵션입니다. 보안 패치나 버그가 수정된 버전을 자동으로 업데이트합니다. 예를 들면 MySQL의 경우 5.6.13을 사용하고 있는데 5.6.14가 나오면 5.6.14 버전으로 업데이트하게 됩니다. 기본값 그대로 사용합니다.

· 유지 관리 기간

- 점검 시간입니다. 해당 시간을 설정하는 이유는 전 페이지의 백업 기간의 시간과 겹치지 않게 하기 위해서입니다. 점검은 기간에 설정한 시간보다 일찍 끝날 수 있습니다.

AWS RDS 설정

연결

엔드포인트

mysql-rds.cwk8zkcpw6nb.ap-northeast-2.rds.amazonaws.com

포트

3306

퍼블릭 액세스 가능

예

보안 그룹 규칙: (2)

보안 그룹 규칙: 필터링

< 1 > ⚙

보안 그룹

유형

규칙

rds-launch-wizard (sg-01ba35b1335237178)

CIDR/IP - Inbound

218.48.56.126/32

rds-launch-wizard (sg-01ba35b1335237178)

CIDR/IP - Outbound

0.0.0.0/0

- 3~7분 정도 기다리시면 인스턴스 생성이 완료됩니다.
- 중앙에 위치한 연결 탭에서 엔드포인트와 포트 번호로 Mysqlworkbench를 통해 접속합니다.

AWS RDS 설정

Connection Name: RDS

Connection Method: Standard (TCP/IP) Method to use to connect to the RDBMS

Parameters SSL Advanced

Hostname: mysql-rds.cwk8zkcpw6n Port: 3306 Name or IP address of the server host. - and TCP/IP port.

Username: admin Name of the user to connect with.

Password: Store in Keychain ... Clear The user's password. Will be requested later if it's not set.

Default Schema: devops The schema to use as default schema. Leave blank to select it later.

- Hostname : 방금 생성된 RDS의 엔드 포인트를 입력합니다.
- Port : 인스턴스 설정 시 기본값으로 사용하는 포트인 3306을 입력합니다.
- Username : 인스턴스 설정 시 입력한 마스터 ID를 입력합니다(admin).
- Default Schema : 기본적으로 사용할 Schema입니다. devops를 입력합니다.

AWS RDS 설정

devops.sql을 열어서 mysqlworkbench를 통해
실행하면 LIST, USER 테이블이 생성됩니다.

AWS RDS 보안 그룹 설정

연결

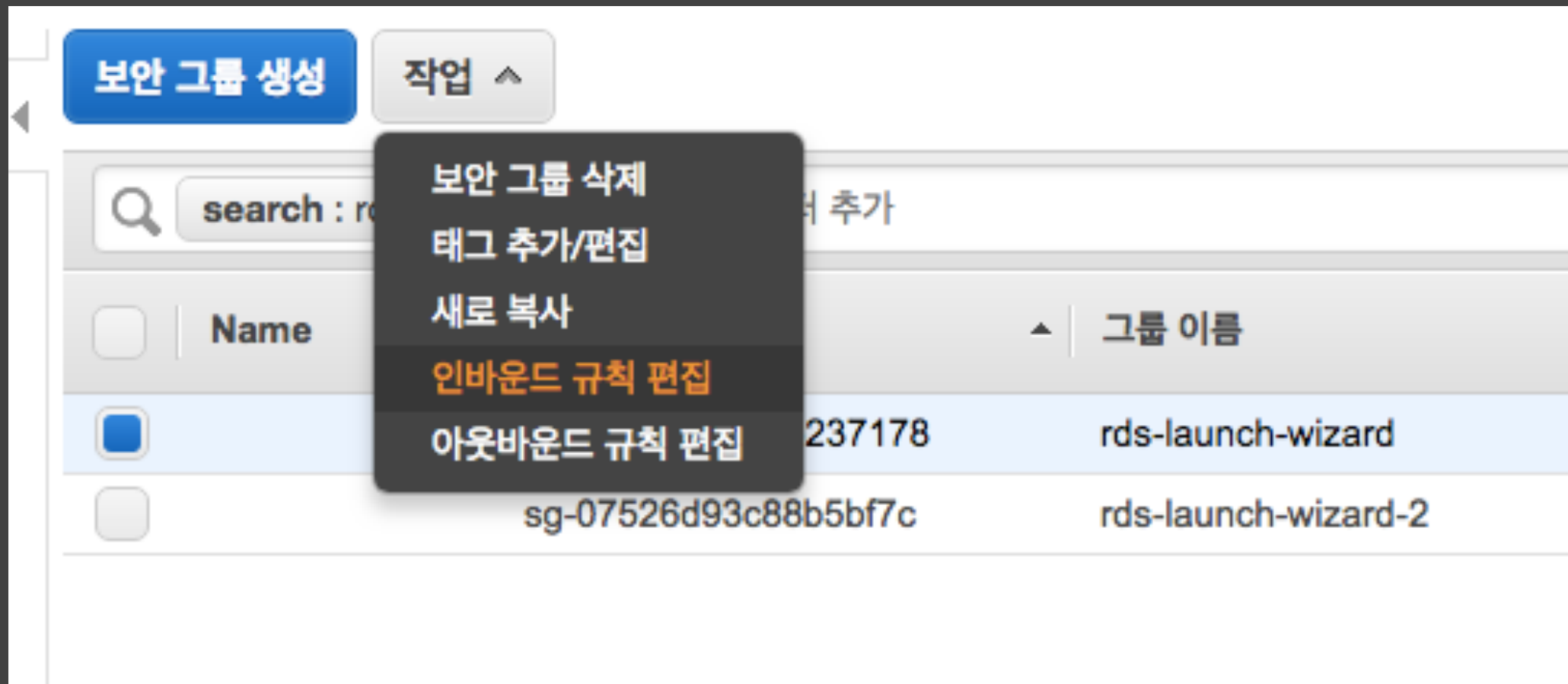
엔드포인트 mysql-rds.cwk8zkcpw6nb.ap-northeast-2.rds.amazonaws.com	포트 3306	퍼블릭 액세스 가능 예
--	------------	-----------------

보안 그룹 규칙: (2)

Q 보안 그룹 규칙: 필터링	< 1 >	⚙
보안 그룹 ▼	유형 ▼	규칙 ▼
rds-launch-wizard (sg-01ba35b1335237178)	CIDR/IP - Inbound	218.48.56.126/32
rds-launch-wizard (sg-01ba35b1335237178)	CIDR/IP - Outbound	0.0.0.0/0

- AWS 콘솔 RDS 화면 중앙에 위치한 보안 그룹 밑에 있는 설정된 보안 그룹을 클릭하여 해당 보안 그룹 편집 화면으로 이동합니다.

AWS RDS 보안 그룹 설정



- 편집할 보안 그룹 클릭 -> 작업 -> 인바운드 규칙 편집을 클릭합니다.

AWS RDS 보안 그룹 설정

인바운드 규칙 편집

유형 ⓘ	프로토콜 ⓘ	포트 범위 ⓘ	소스 ⓘ	설명 ⓘ
MYSQL/Auror ⌵	TCP	3306	사용자 지정 ⌵ 218.48.56.126/32	예: 관리자 데스크톱용 SSH

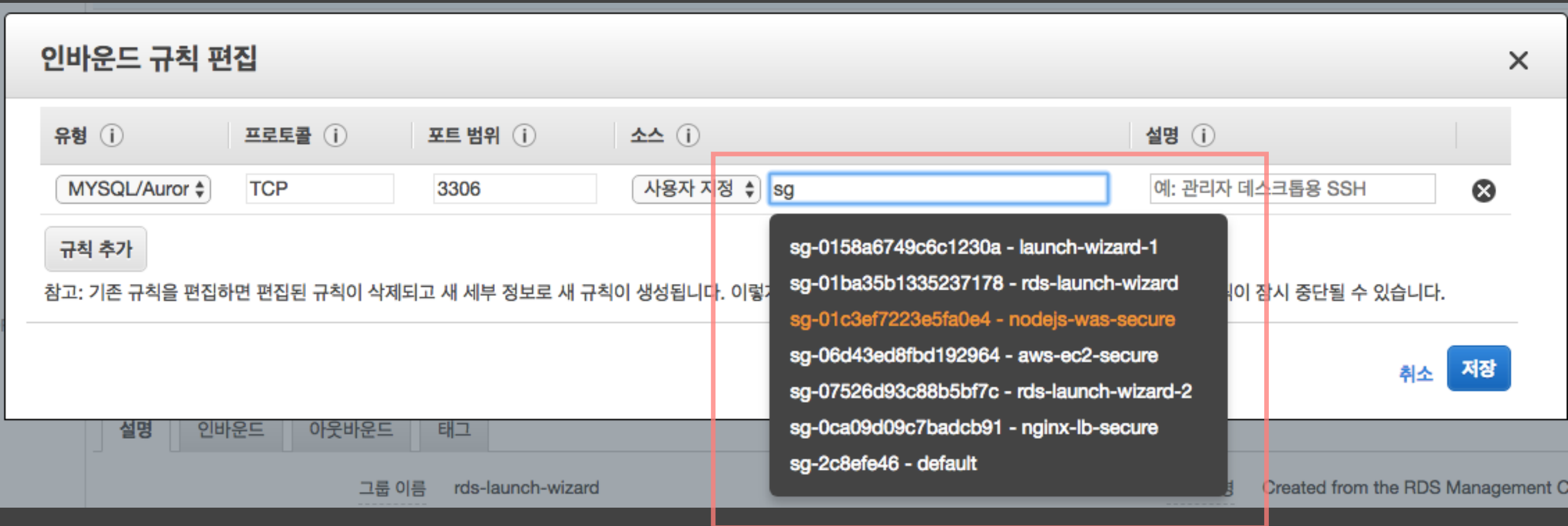
규칙 추가

참고: 기존 규칙을 편집하면 편집된 규칙이 삭제되고 새 세부 정보로 새 규칙이 생성됩니다. 이렇게 하면 새 규칙이 생성될 때까지 해당 규칙에 의존하는 트래픽이 잠시 중단될 수 있습니다.

취소 저장

- RDS를 처음 생성 시 기본 보안 그룹을 사용하면 3306 포트를 통해 내 IP만 접근이 가능하도록 설정되어있습니다.
- 그래서 전 단계에서 로컬 환경에서 MySQLWorkbench로 해당 RDS에 연결이 가능했습니다.

AWS RDS 보안 그룹 설정

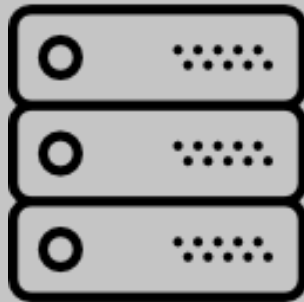
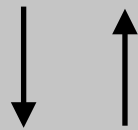


- WAS가 구동하는 EC2 인스턴스에서도 해당 RDS에 연결이 가능하게끔 sg를 입력하시면 sg-*에 해당하는 모든 보안 그룹이 리스트됩니다. 그 중 WAS(node.js)가 구동하는 인스턴스에서 사용하는 보안 그룹을 선택합니다.

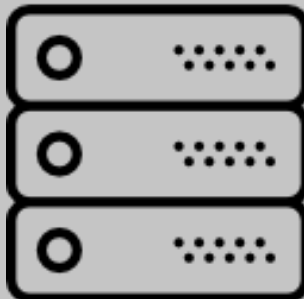
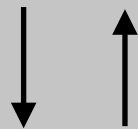
웹 서버 - WAS - DB 연결



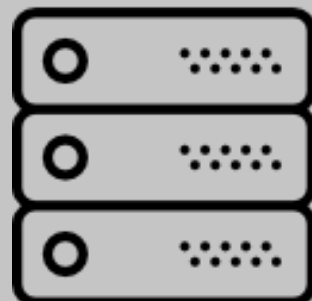
Client(웹 브라우저)



웹 서버 영역(ELB + EC2 N대)



WAS 영역(ELB + EC2 N대)

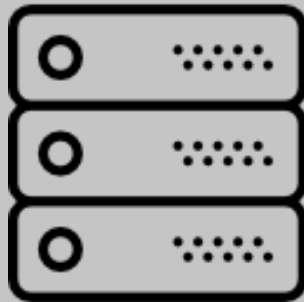
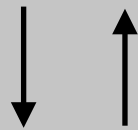


DB 영역(MySQL, AWS RDS)

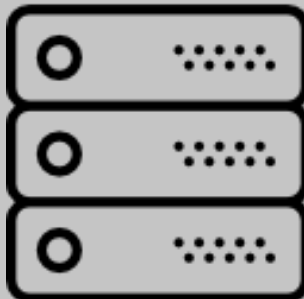
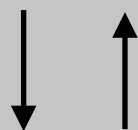
웹 서버 - WAS - DB 연결



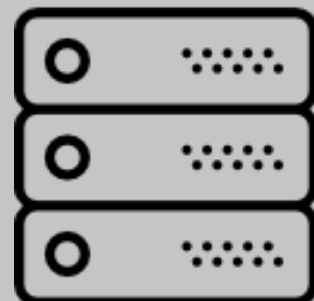
Client(웹 브라우저)



웹 서버 영역(ELB + EC2 N대)



WAS(Node.js) DB 서버와 연결하는 프로젝트
-b **v2**로 배포



DB 영역(MySQL, AWS RDS)

Node.js 프로젝트 재배포

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~\$

SSH 클라이언트를 통해 WAS(Node.js)가 구동하는 EC2 인스턴스에 접속

```
ls -al
```

```
rm -rf Fastcampus-api-deploy/
```

```
git clone -b v2 https://github.com/owen1025/Fastcampus-api-deploy.git
```

```
cd Fastcampus-api-deploy/
```

```
npm install
```

```
vi package.json
```

Node.js 프로젝트 재배포

1. ec2-user@ip-172-31-25-131:~ (bash)
a1@1ui-MacBook-Air:~\$

- 1) `rm -rf Fastcampus-api-deploy/`
- 2) `git clone -b v2 https://github.com/owen1025/Fastcampus-api-deploy.git`
- 3) `cd Fastcampus-api-deploy/`
- 4) `npm install`
- 5) `vi package.json`

1. 기존 API 프로젝트 제거
2. DB 서버와 통신하는 v2 프로젝트 코드 다운로드
3. 새로 받은 v2 프로젝트로 이동
4. package.json에 명시된 모듈 의존성을 확인하여 관련 모듈 다운로드
5. vi 편집기로 package.json 수정

Node.js 프로젝트 재배포

```
"database": {  
  "host": "{RDS-DNS}",  
  "user": "admin",  
  "password": "test1234",
```

```
  "host": "devops"  
}  
password
```

- 앞서 구축한 RDS의 DNS 입력
- ex. "mysql-rds.cwk8zkcpw6nb.ap-northeast-2.rds.amazonaws.com"

- RDS 구축 시 설정한 마스터 계정의 패스워드 입력

Node.js 프로젝트 재배포

```
1. ec2-user@ip-172-31-25-131:~ (bash)
a1@1ui-MacBook-Air:~$
```

vi 편집기에서 :wq로 package.json 저장하고 종료

pm2 restart WAS

- 해당 명령어로 수정 내역을 적용하기 위해 WAS 프로세스 재기동

RDS multi-AZ vs Read replica

다중 AZ 배포

동기식 복제 - 높은 안정성

기본 인스턴스의 데이터베이스 엔진만 활성화

자동 백업은 대기 상태에서 수행

단일 지역 내에 항상 2 개의 가용성 영역을 확장

기본 데이터베이스 엔진 버전 업그레이드가 발생합니다.

문제가 감지되면 대기 모드로 자동 Failover 조치

읽기 전용 복제본

비동기식 복제 - 높은 확장성

모든 읽기 전용 복제본은 접근이 가능하며 읽기 확장도 가능

기본 제공된 백업 구성 없음

가용 영역, 교차 AZ 또는 교차 지역 내에 있을 수 있음

데이터베이스 엔진 버전 업그레이드는 원본 인스턴스와 독립됨

독립형 데이터베이스 인스턴스로 수동 승격 될 수 있음

RDS multi-AZ vs Read replica

기존 RDS 인스턴스 장애시 동기식으로 복제된
예비 RDS 인스턴스로 자동 failover 처리

- DB 서버의 **고가용성** 보장

문제가 감지되면 대기 모드로 자동 Failover 조치

독립형 데이터베이스 인스턴스로 수동 승격 될 수 있음

RDS multi-AZ vs Read replica

다중 AZ 배포

동기식 복제 - 높은 안정성

읽기 전용 복제본

비동기식 복제 - 높은 확장성

읽기 전용 DB 서버(Slave)를 자동 구축, 확장

- DB 서버의 **고성능** 보장
- 쓰기 전용 DB 서버(Master)와 비동기식 복제되어 약간의 시간차 발생
- memcached 서버를 사용하는 이유
- DB 영역의 클러스터링으로 해결 가능
- 읽기 전용은 INSERT, UPDATE, DELETE 등 쓰기 query 사용 불가

해당 인프라의 문제점

Nginx의 설정 파일을 변경해야한다면?

- nginx-lb에 물려있는 모든 인스턴스마다 접속해서 업데이트를 해줘야할까?



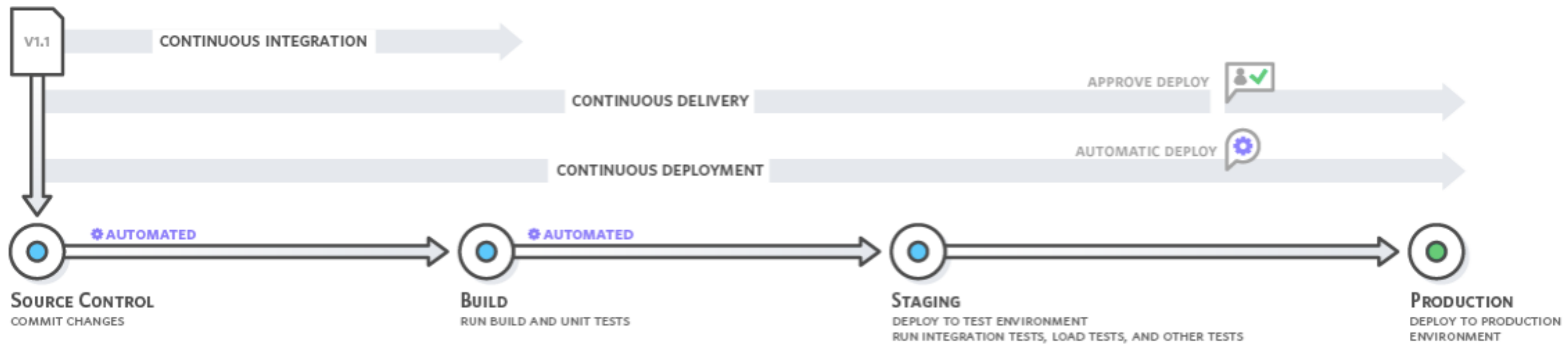
해당 인프라의 문제점

지속적 통합(Continuous Integration)

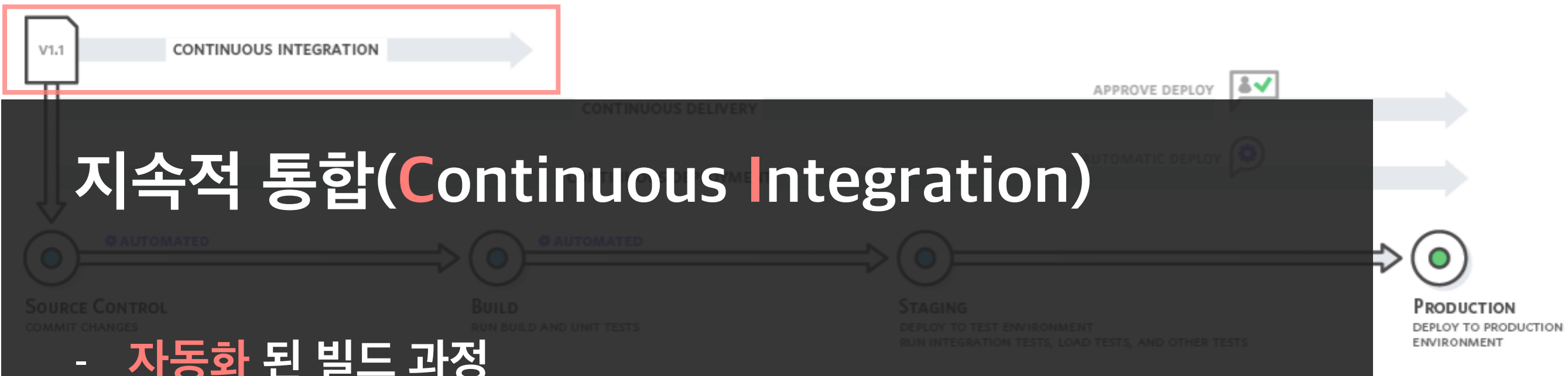
지속적 배포(Continuous Delivery)

- Jenkins CI
- travis CI
- AWS Codepipeline

지속적인 통합과 배포(CI/CD)



지속적인 통합과 배포(CI/CD)



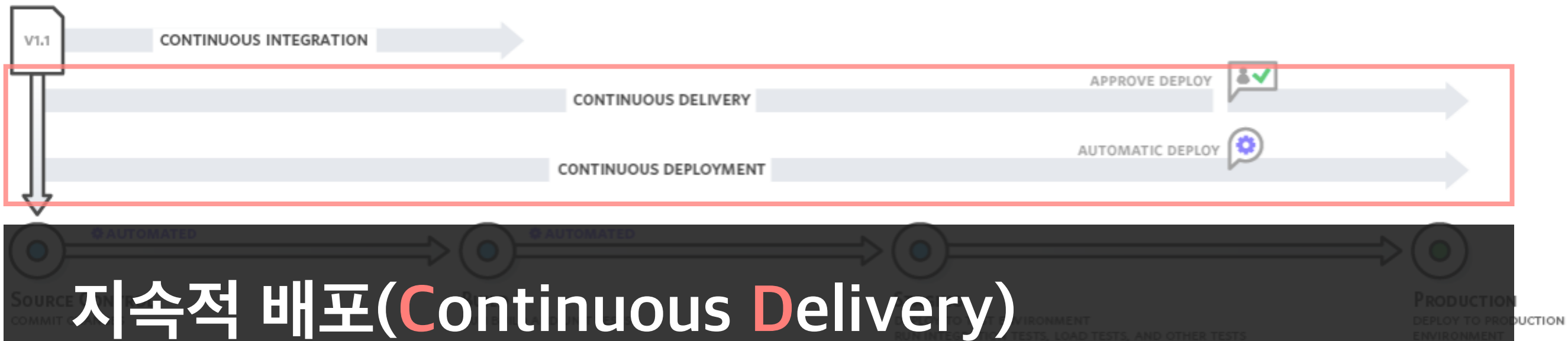
- 자동화 된 빌드 과정

- ex 1) Java(spring) maven 빌드,
- ex 2) dockerfile image 빌드 등

- 자동화 된 테스트 과정

- ex 1) Nginx location 테스트
- ex 2) API 별 시나리오 테스트

지속적인 통합과 배포(CI/CD)



지속적 배포(Continuous Delivery)

- 자동화 된 배포 과정
 - 버전 별로 관리된 Git 레포지토리를 통한 버전 별 통합 배포
 - 잘못된 배포 시 전 단계로 rollback
- 여러 대의 EC2 인스턴스(VM), 베어메탈, 컨테이너 환경에 일일히 배포할 필요 없이 시스템이 자동으로 배포

지속적인 통합과 배포(CI/CD)



개발자 생산성 향상

지속적 통합을 사용하면 개발자가 수동 작업에 대한 부담을 덜고 고객에게 제공되는 오류 및 버그 수를 줄이는데 도움이 되는 기능을 활용함으로써 팀의 생산성을 높일 수 있습니다.



버그를 더 빠르게 발견 및 해결

테스트를 좀 더 빈번하게 수행함으로써, 팀에서는 이후에 더 큰 문제로 발전하기 전에 버그를 조기에 발견하고 해결할 수 있습니다.



업데이트를 더 빠르게 제공

지속적 통합을 사용하면 팀이 좀 더 빠르고 좀 더 빈번하게 고객에게 업데이트를 제공할 수 있습니다.

해당 인프라의 문제점



해당 인프라의 문제점

인프라를 운영하면서 서버들의 모니터링을 할 때

- 외부에 모니터링 시스템을 구축하여 전체 인스턴스 모니터링
- AWS Cloudwatch 통합 모니터링 환경
- ELK 로그 모니터링
- APM(Application performance monitoring)
- Metric(Cpu, ram, etc...) 모니터링

과제

WAS 영역 Auto scaling group 설정

RDS Read replica 환경 구축