

DevOps 구축 BOOTCAMP



4주차 목표

VPC, Subnet 가용 영역 등 AWS 네트워크 환경에 대한 이해

Jenkins를 기반으로 하는 자동화 배포 과정 진행

컨테이너 환경으로 이전 준비

AWS 네트워크 환경

서비스 ▾ 리소스 그룹 ▾

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

인스턴스 세부 정보 구성

이 인스턴스를 구성합니다. 동일한 AMI의 여러 인스턴스를 시작하고 스팟 인스턴스를 요청하여 보다 저렴한 요금을 활용하며 인스턴스에 액세스 관리 역할을 할당하는 등 다양한 기능을

인스턴스 개수 **VPC & Subnet** Auto Scaling 그룹 시작 ⓘ

구매 옵션 ⓘ ☐ 스팟 인스턴스 요청

네트워크 ⓘ vpc-ec151b84 (기본값) ↕ 새 VPC 생성

서브넷 ⓘ subnet-ab9cf7e7 | 기본값 ap-northeast-2c ↕ 새 서브넷 생성
4086개 IP 주소 사용 가능

퍼블릭 IP 자동 할당 ⓘ 서브넷 사용 설정(활성화) ↕

owen ▾ 서울 ▲

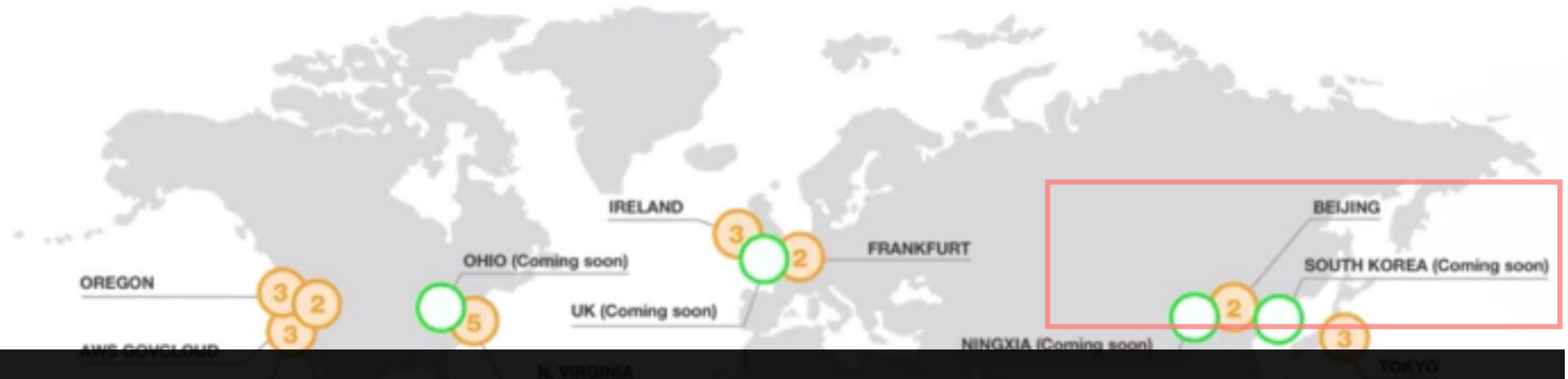
- 미국 동부 (버지니아 북부)
- 미국 동부 (오하이오)
- 미국 서부 (캘리포니아 북부)
- 미국 서부 (오레곤)
- 아시아 태평양 (뭄바이)
- 아시아 태평양 (서울)**
- 아시아 태평양 (싱가포르)
- 아시아 태평양 (시드니)
- 아시아 태평양 (도쿄)
- 캐나다 (중부)
- EU (프랑크푸르트)

Region

AWS 네트워크 환경 - Region



AWS 네트워크 환경 - Region



AWS 리전

- 세계 각지에 위치한 AWS IDC
- AWS의 서비스가 위치한 지리적인 장소이며, 글로벌 기준으로
 - 지역적 위치를 묶어서 관리하는 단위
- ○ 사용자와 가까운 위치에 있는 리전일수록 응답속도가 빠름
- 하나의 리전은 여러 개의 가용 영역(AZ)로 구성된다

CloudPing.info

Amazon Web Services™ are available in several regions. Click the link from your browser to each AWS™ region.

Region	Latency
US-East (Virginia)	234 ms
US East (Ohio)	204 ms
US-West (California)	139 ms
US-West (Oregon)	176 ms
Canada (Central)	247 ms
Europe (Ireland)	293 ms
Europe (London)	339 ms
Europe (Frankfurt)	337 ms
Asia Pacific (Mumbai)	165 ms
Asia Pacific (Seoul)	19 ms
Asia Pacific (Singapore)	125 ms
Asia Pacific (Sydney)	165 ms
Asia Pacific (Tokyo)	58 ms
South America (São Paulo)	334 ms
China (Beijing)	85 ms
AWS GovCloud (US)	148 ms

HTTP Ping

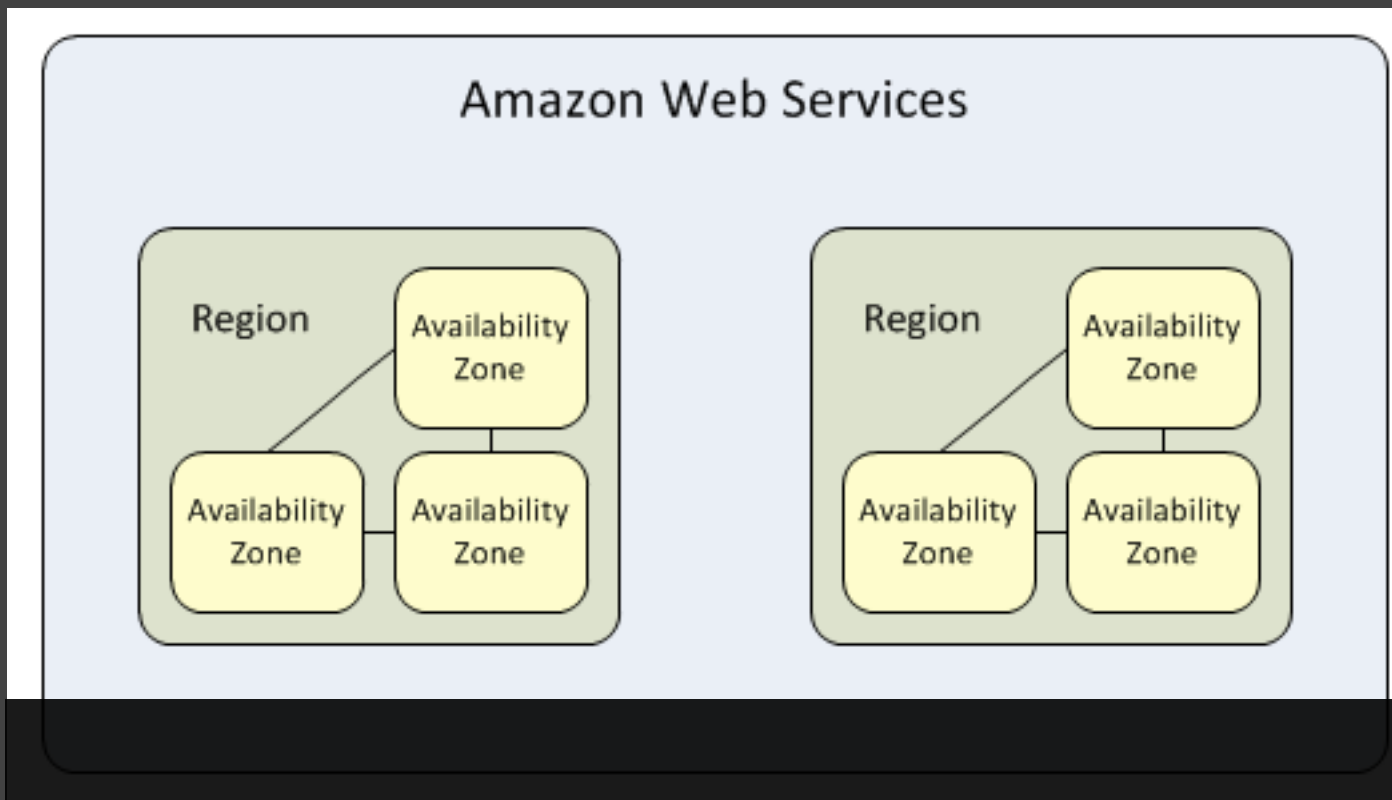
If you like this tool, please check out [RestBackup.com](https://restbackup.com).

Region



수록 응답속도가 빠름
(2)로 구성된다

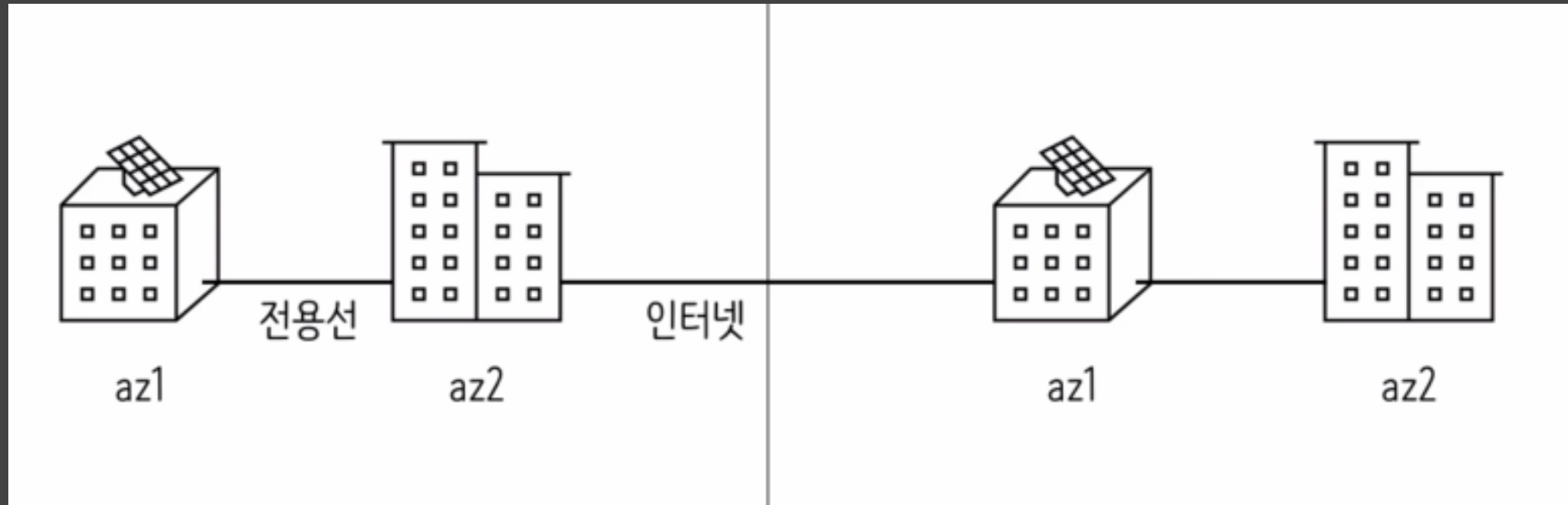
AWS 네트워크 환경 - 가용 영역



AWS 가용 영역 (Availability zone, **AZ**)

- 하나의 리전은 물리적으로 독립된 여러 곳의 IDC(가용 영역)으로 구성
- 가용 영역은 low latency link로 연결되어 물리적으로 떨어져 생기는 네트워크 latency를 보장
- 리전 / AZ를 통해 손쉽게 멀티 IDC를 운영하는 이점을 가질 수 있음

AWS 네트워크 환경 - 가용 영역



AWS 가용 영역 (Availability zone, **AZ**)

- AZ = IDC (물리적으로 독립된 실제 컴퓨팅(베어메탈)이 운영되는 곳
- 서울 리전은 두 곳의 AZ로 구성되어 있음
- RDS의 Multi AZ 구성이 해당 개념을 바탕으로 구성됨 (**고가용성** 보장)

AWS 네트워크 환경 - 가용 영역

로드 밸런서: nginx-lb

설명 | **인스턴스** | 상태 검사 | 리스너 | 모니터링 | 태그 | 마이그레이션

연결 드레이닝: 활성, 300 초 (편집)

두 곳의 AZ(가용 영역)에 AWS 서비스 배포

고가용성 보장

인스턴스 ID	이름	가용 영역	상태	작업
i-0bd9a5d98e49b444e	nginx	ap-northeast-2c	OutOfService ⓘ	로드 밸런서

가용 영역 편집

가용 영역	서브넷 ID	서브넷 CIDR	인스턴스 개수	정상 상태?
ap-northeast-2a	subnet-2d1a1d45	172.31.0.0/20	0	아니요 (가용 영역에 정상 대상이 없음)
ap-northeast-2c	subnet-ab9cf7e7	172.31.16.0/20	1	아니요 (가용 영역에 정상 대상이 없음)

AWS 네트워크 환경

AWS Seoul region

AWS 네트워크 환경

AWS Seoul region

Availability Zone 1
(ap-northeast-2a)

Availability Zone 2
(ap-northeast-2c)

AWS 네트워크 환경

AWS Seoul region

Availability Zone 1
(ap-northeast-2a)

Availability Zone 2
(ap-northeast-2c)

VPC

AWS 네트워크 환경

VPC

Availability Zone 1

Availability Zone 2

AWS 네트워크 환경

VPC

Availability Zone 1

Availability Zone 2

VPC

- 한 리전 내 여러 가용 영역을 걸쳐 논리적으로 분리하여 독립적으로 구성한 AWS 계정 전용 가상 네트워크(**Virtual Private Cloud**)

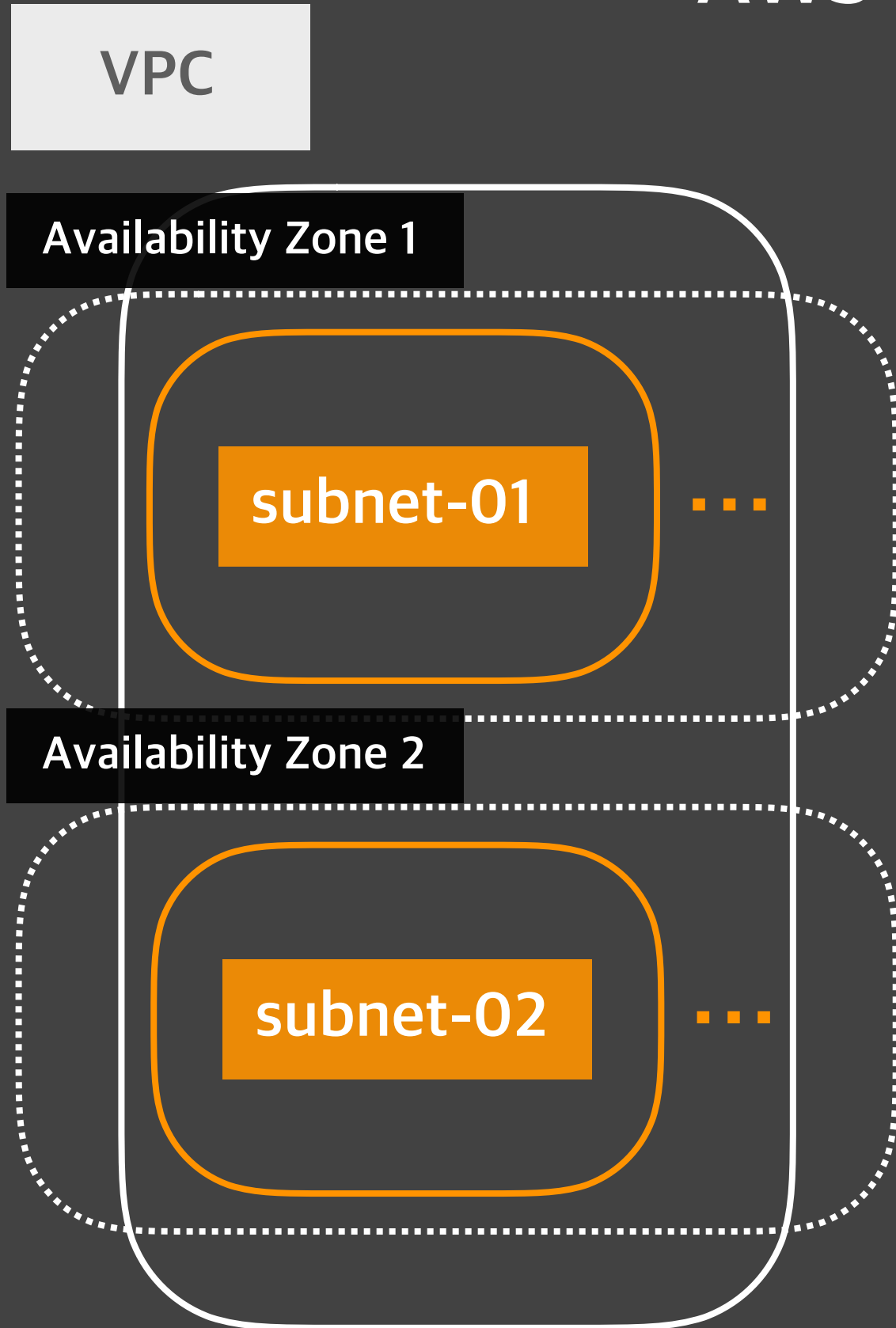
AWS 네트워크 환경 - VPC

- AWS에서 기본으로 제공하는 논리적 가상 네트워크
- 한 리전 내 여러 가용 영역을 걸쳐 운영 가능
- 필요한 경우 네트워크를 만들어 Private subnet, Public subnet을 운용할 수 있다
- Public subnet 내 인스턴스는 외부 인터넷과 통신 가능한 public IP와 VPC 안에서만 사용 가능한 private IP를 할당받는다
- Private subnet 내 인스턴스는 private IP만 할당받는다

AWS 네트워크 환경 - VPC

- 다른 계정에서 AWS 리소스 격리
- 인스턴스와의 네트워크 트래픽 라우팅
- 네트워크 침입으로부터 인스턴스 보호
- 한 리전에선 최대 5개의 VPC 생성 가능

AWS 네트워크 환경



AWS 네트워크 환경

VPC

Availability Zone 1

subnet-01

...

Availability Zone 2

subnet-02

...

Subnet

- VPC의 ip 주소 범위
 - **IP block**을 구분 짓는 네트워크 구성 그룹
- 각 가용 영역에 하나 이상의 서브넷 생성 가능
- 지정된 서브넷으로 AWS 리소스(EC2, RDS 등) 실행

AWS 네트워크 환경

VPC (10.0.0.0/16)

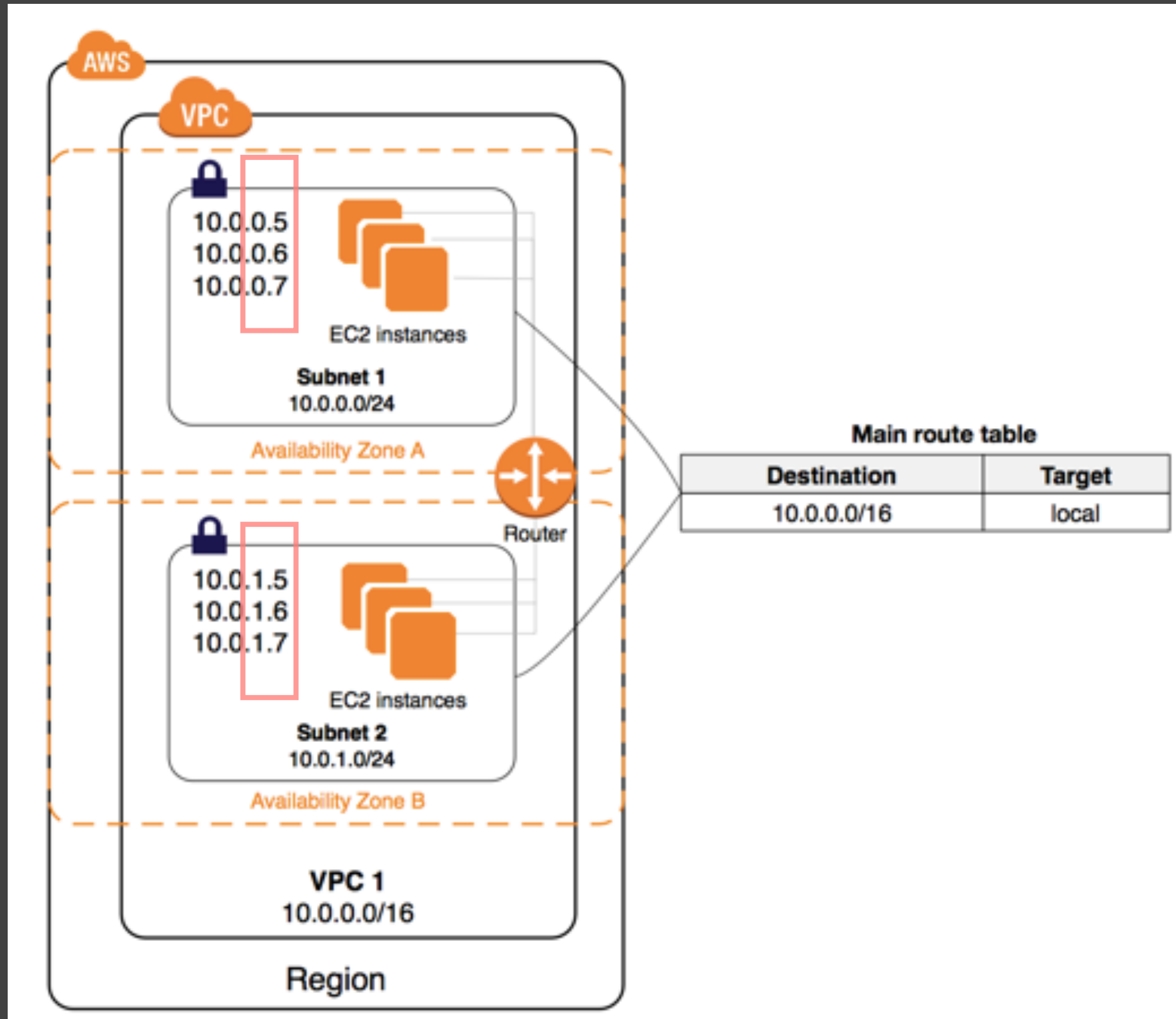
Availability Zone 1

Subnet-01
10.0.0.0/24

Availability Zone 2

Subnet-01
10.0.1.0/24

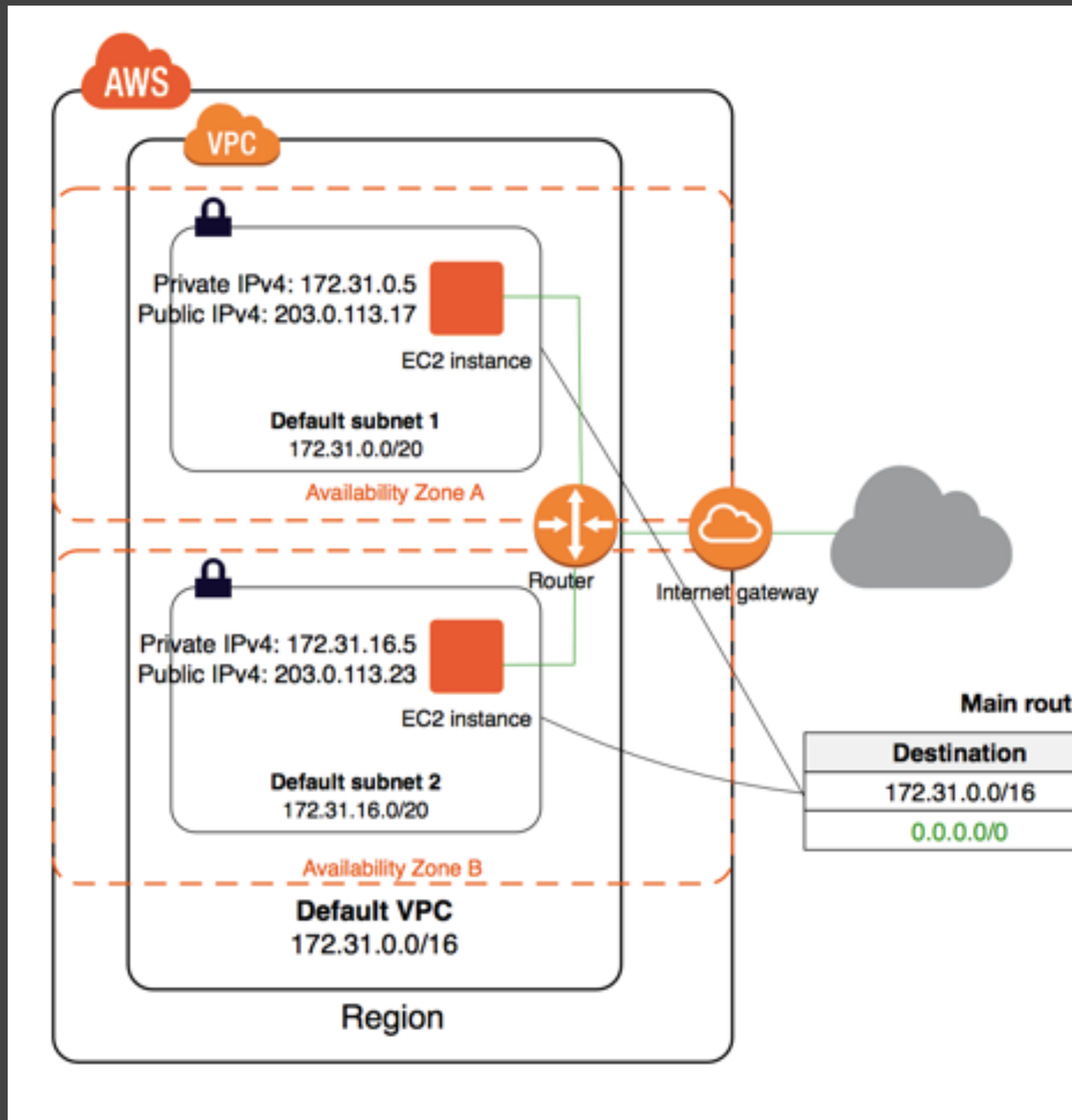
AWS 네트워크 환경



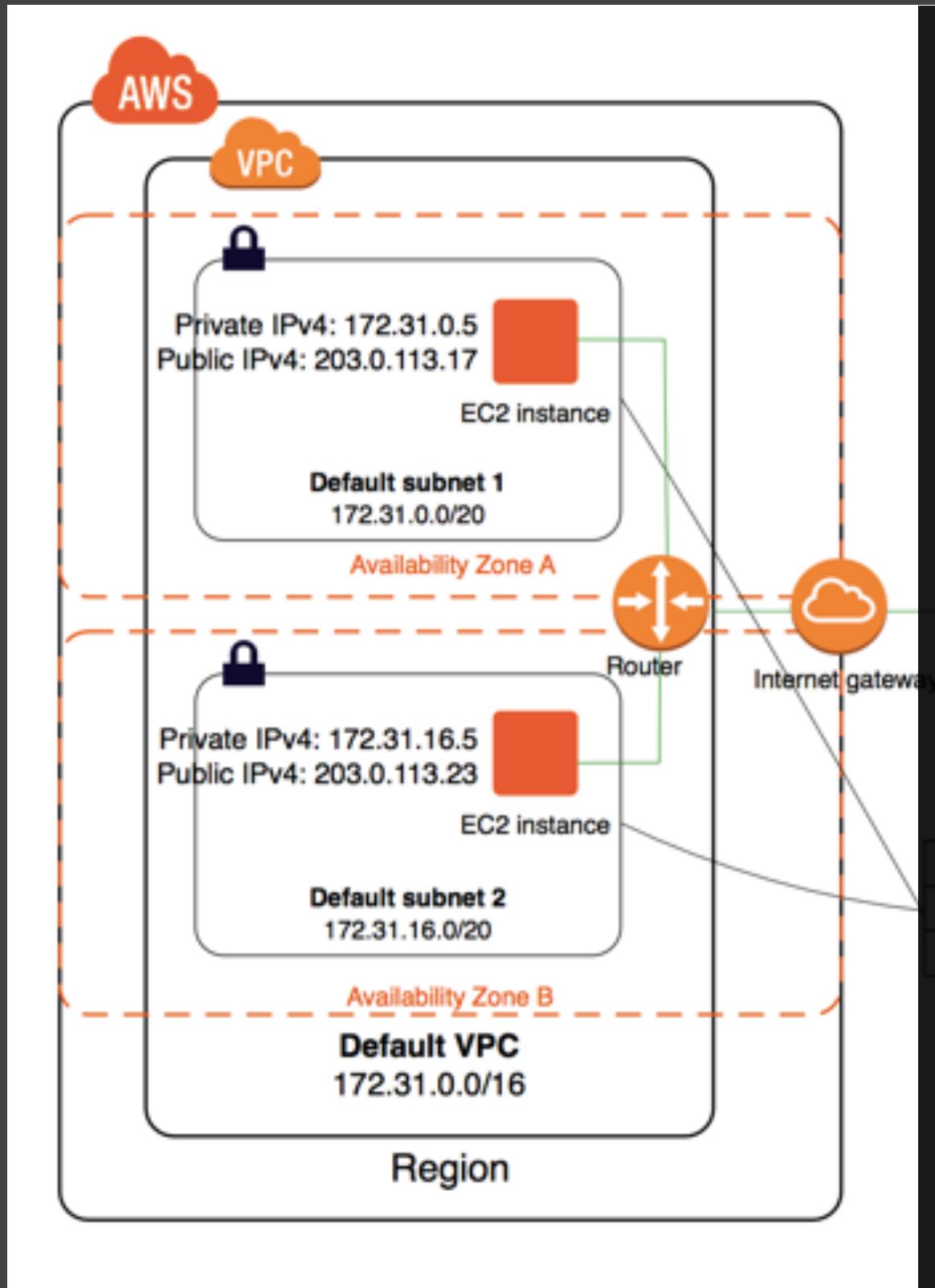
AWS 네트워크 환경



AWS 네트워크 환경

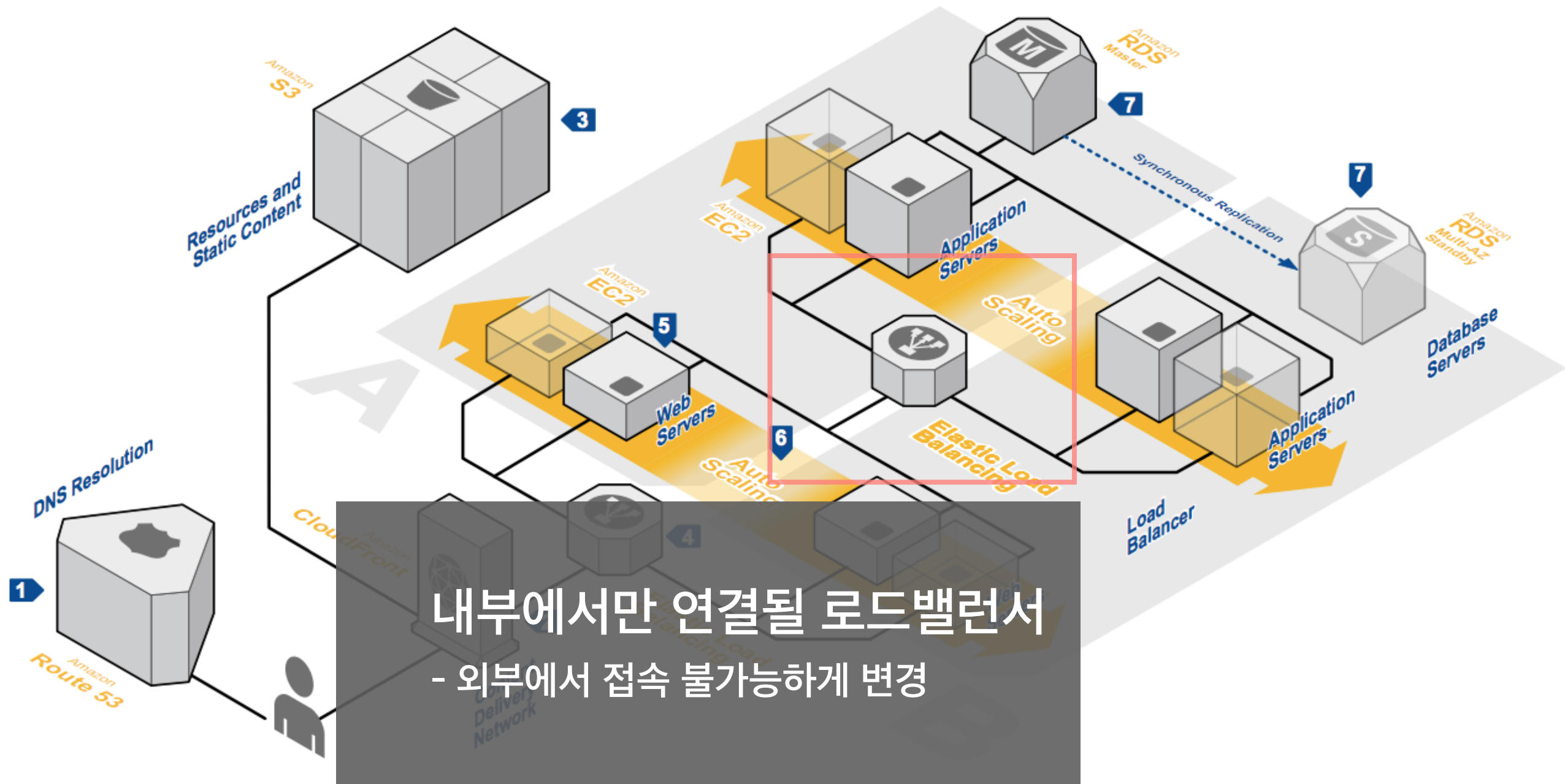


AWS 네트워크 환경

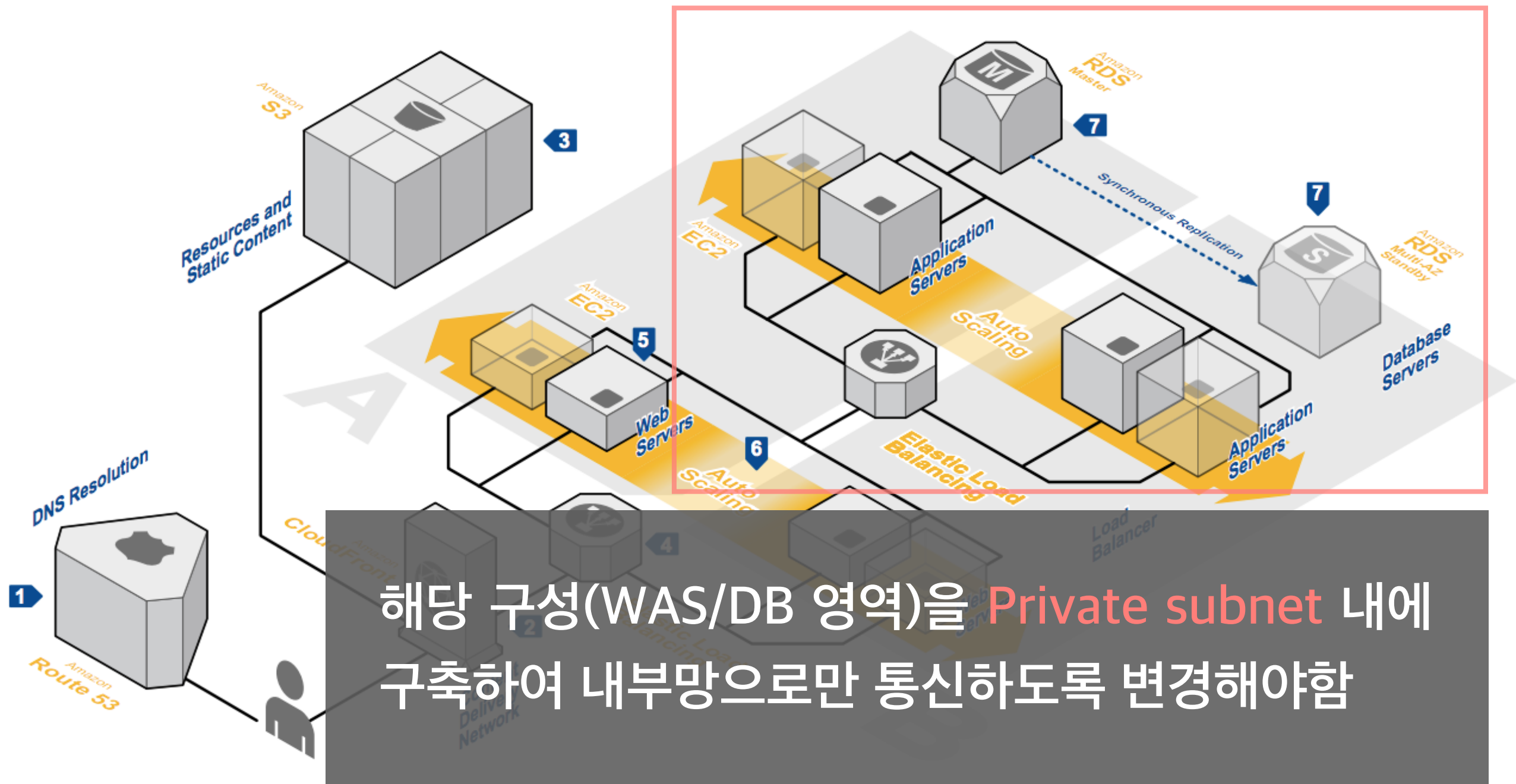


- 기본 VPC로 설정시 각각의 기본 서브넷은 **Public subnet**으로 구성됨
- public subnet에 구축된 AWS 인스턴스엔 public/private ip가 각각 부여됨
- **AWS Internet gateway**를 통해 외부 인터넷에 연결

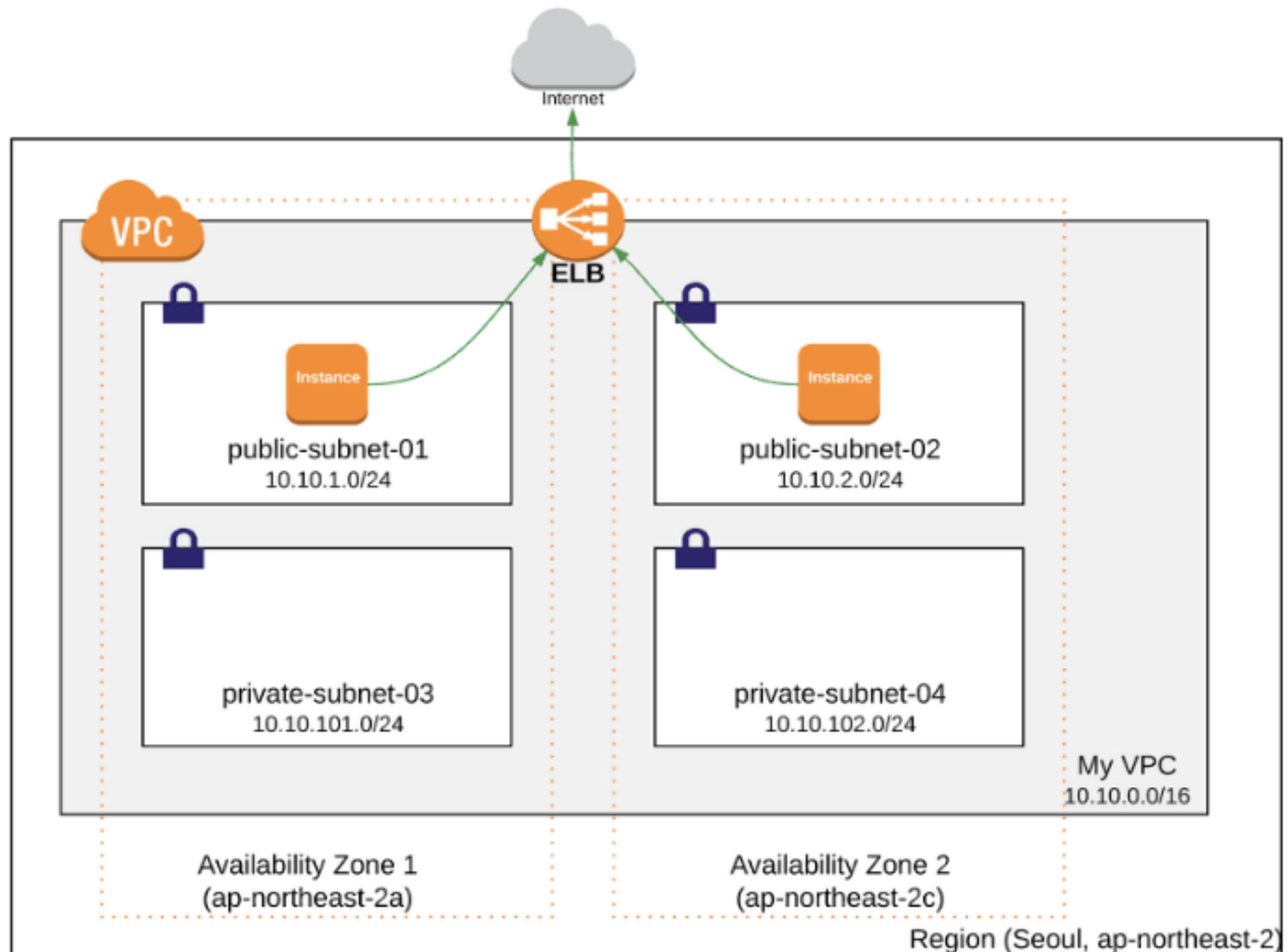
AWS 네트워크 환경



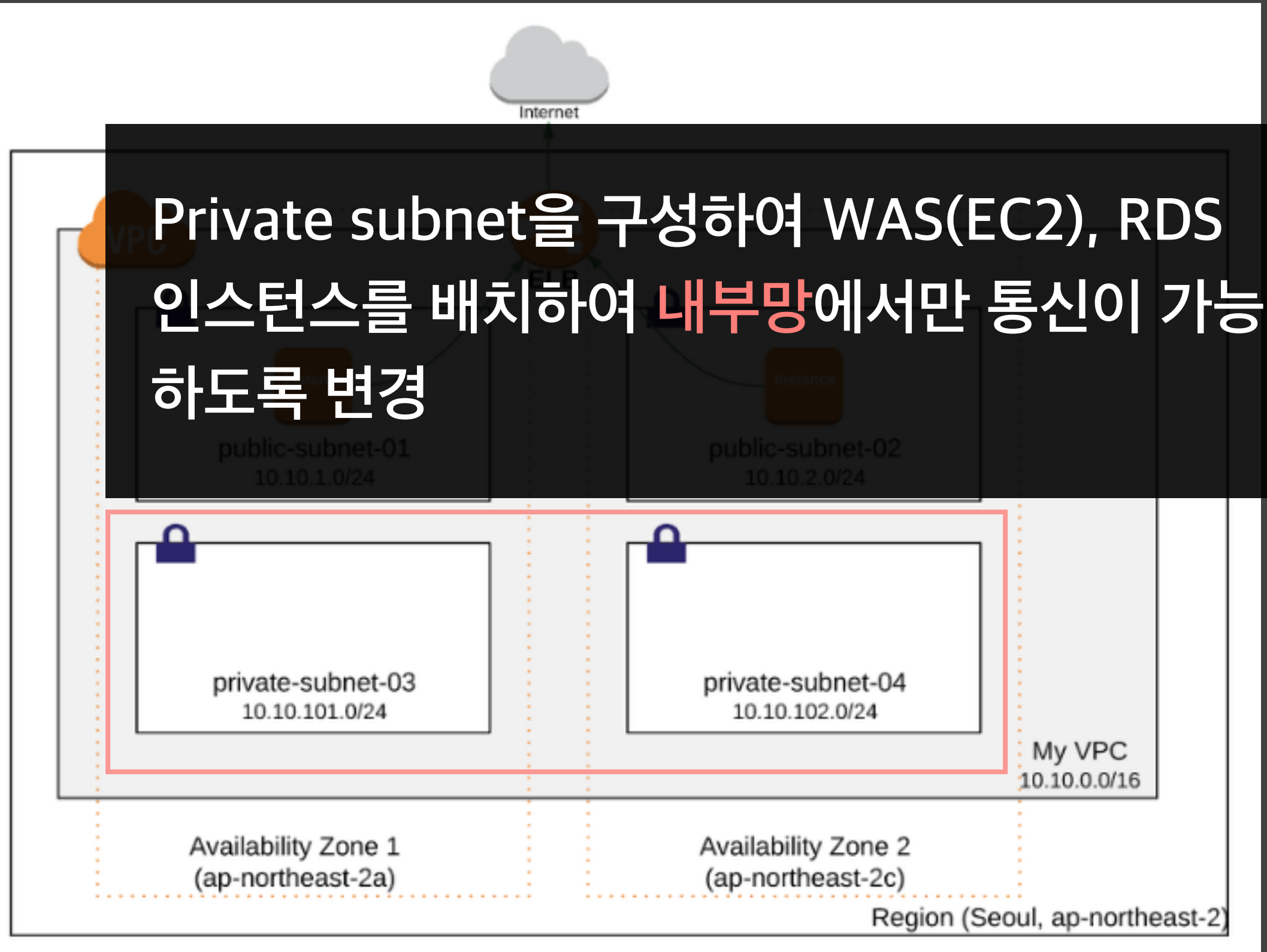
AWS 네트워크 환경



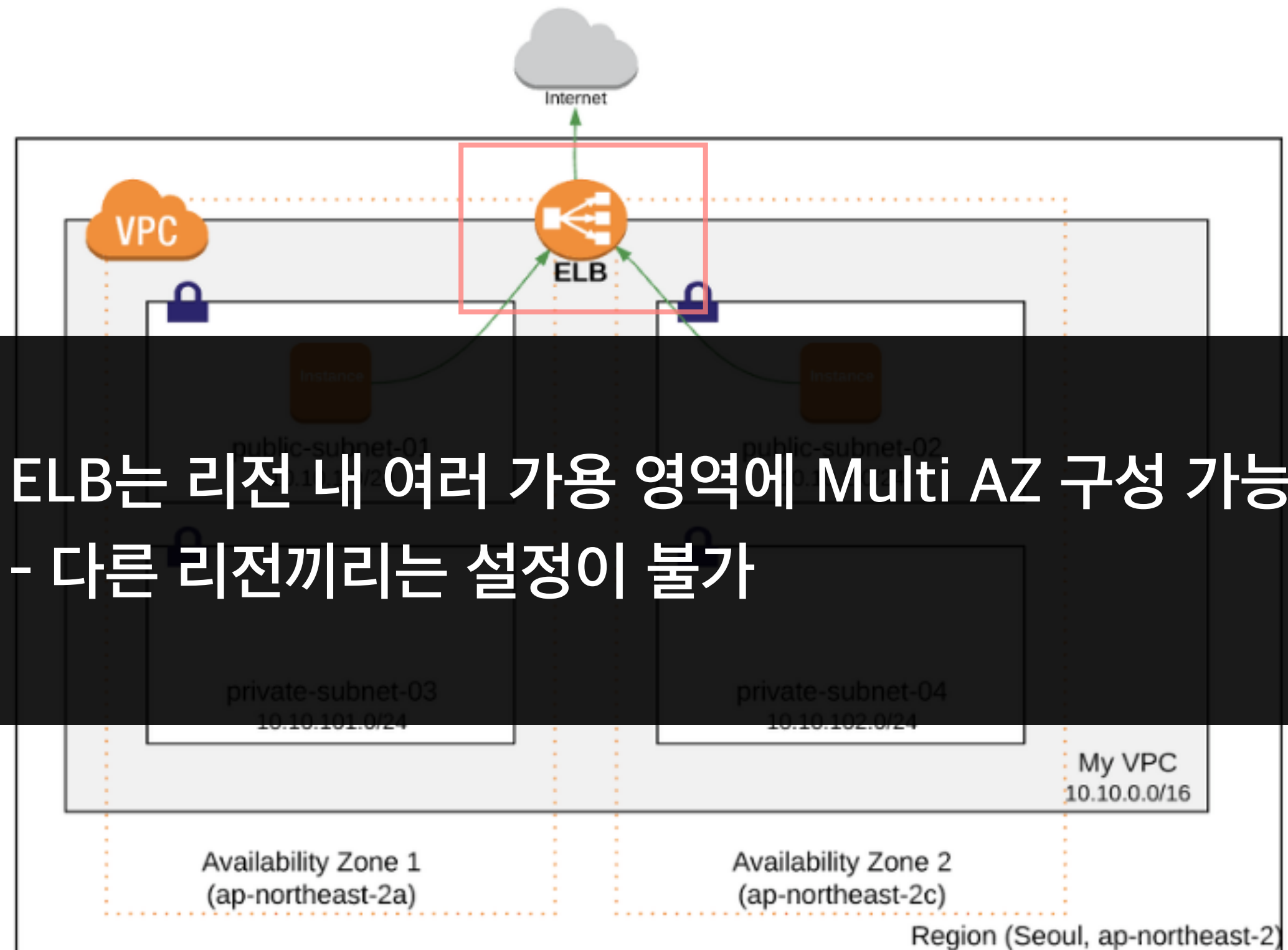
AWS 네트워크 환경



AWS 네트워크 환경

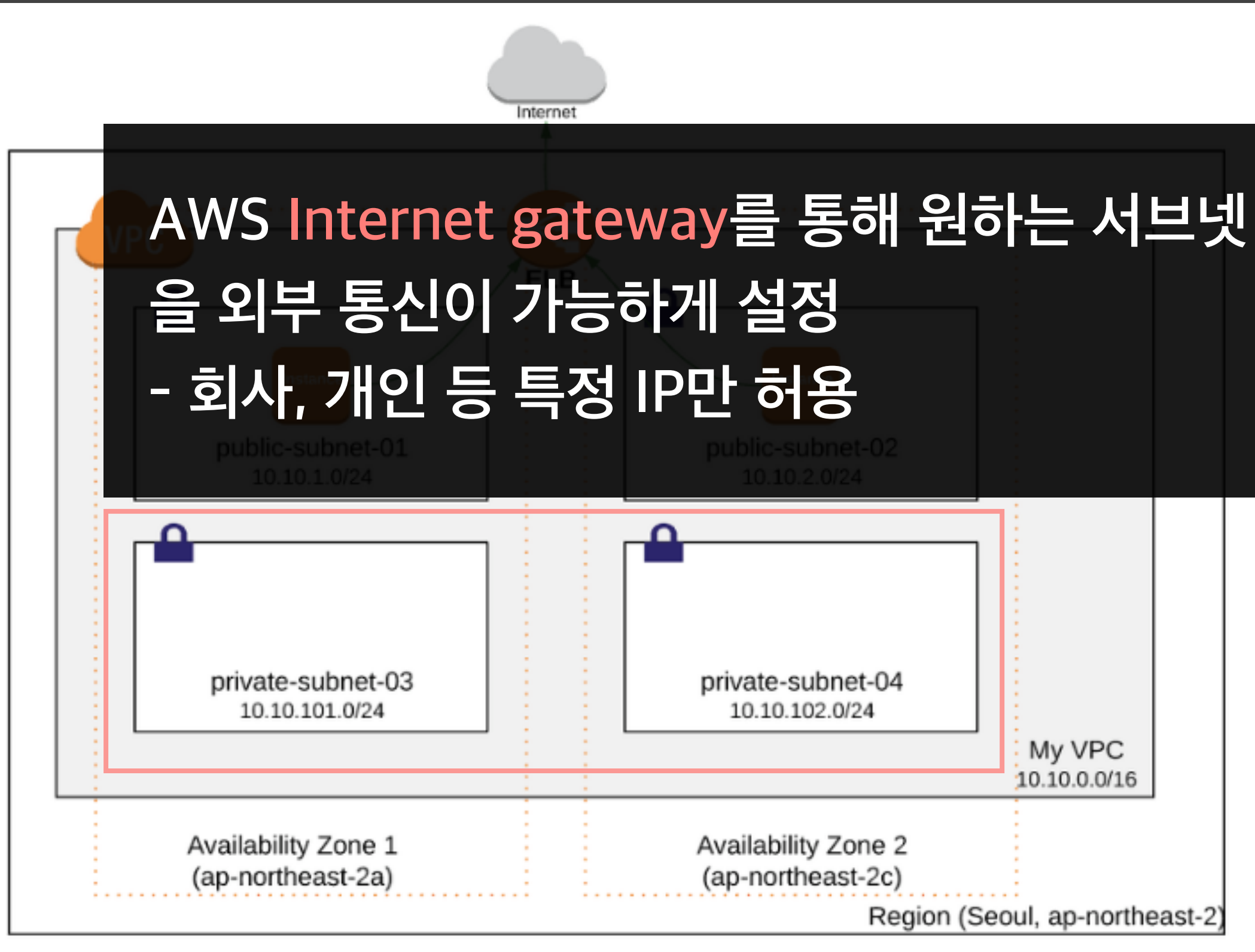


AWS 네트워크 환경

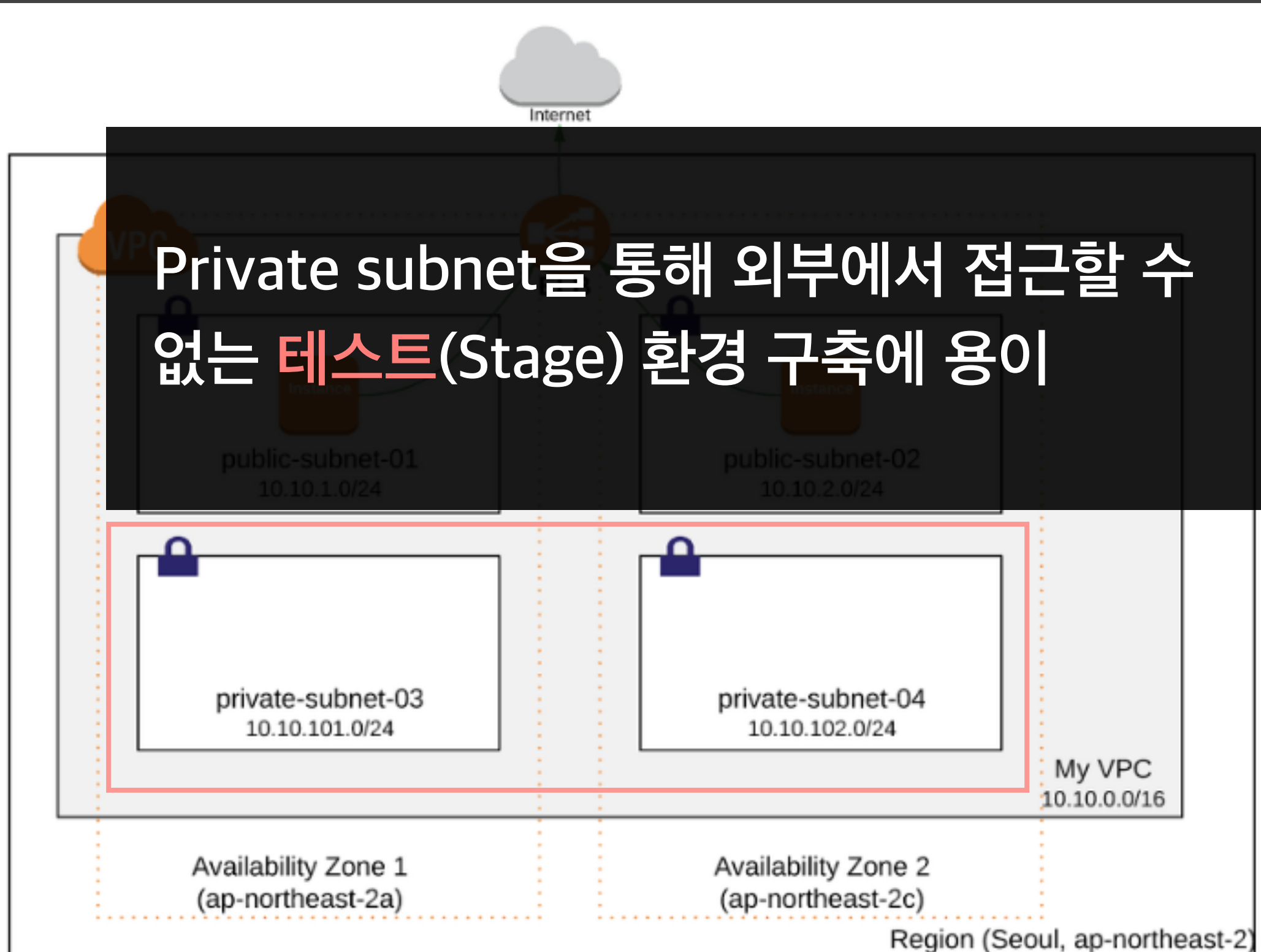


ELB는 리전 내 여러 가용 영역에 Multi AZ 구성 가능
- 다른 리전끼리는 설정이 불가

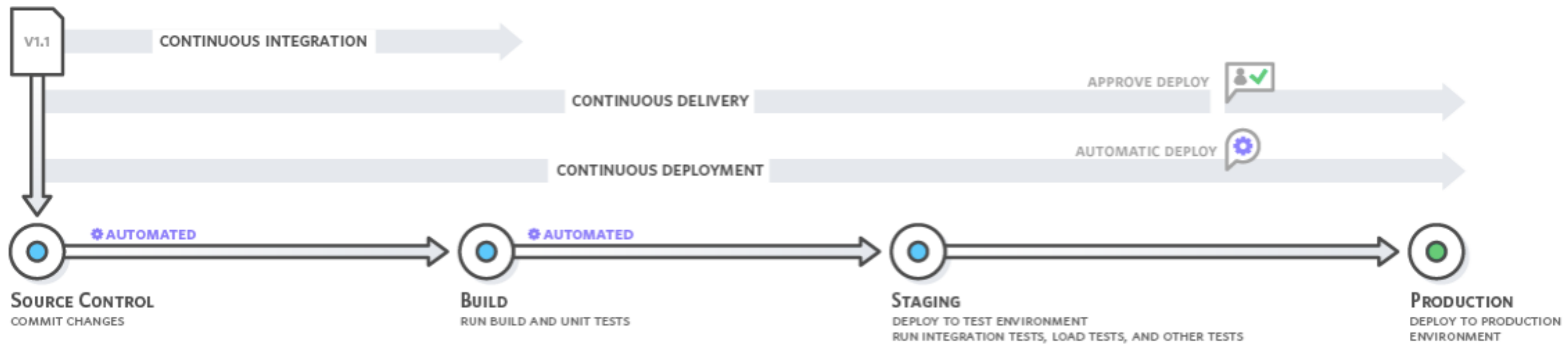
AWS 네트워크 환경



AWS 네트워크 환경



지속적인 통합과 배포(CI/CD)



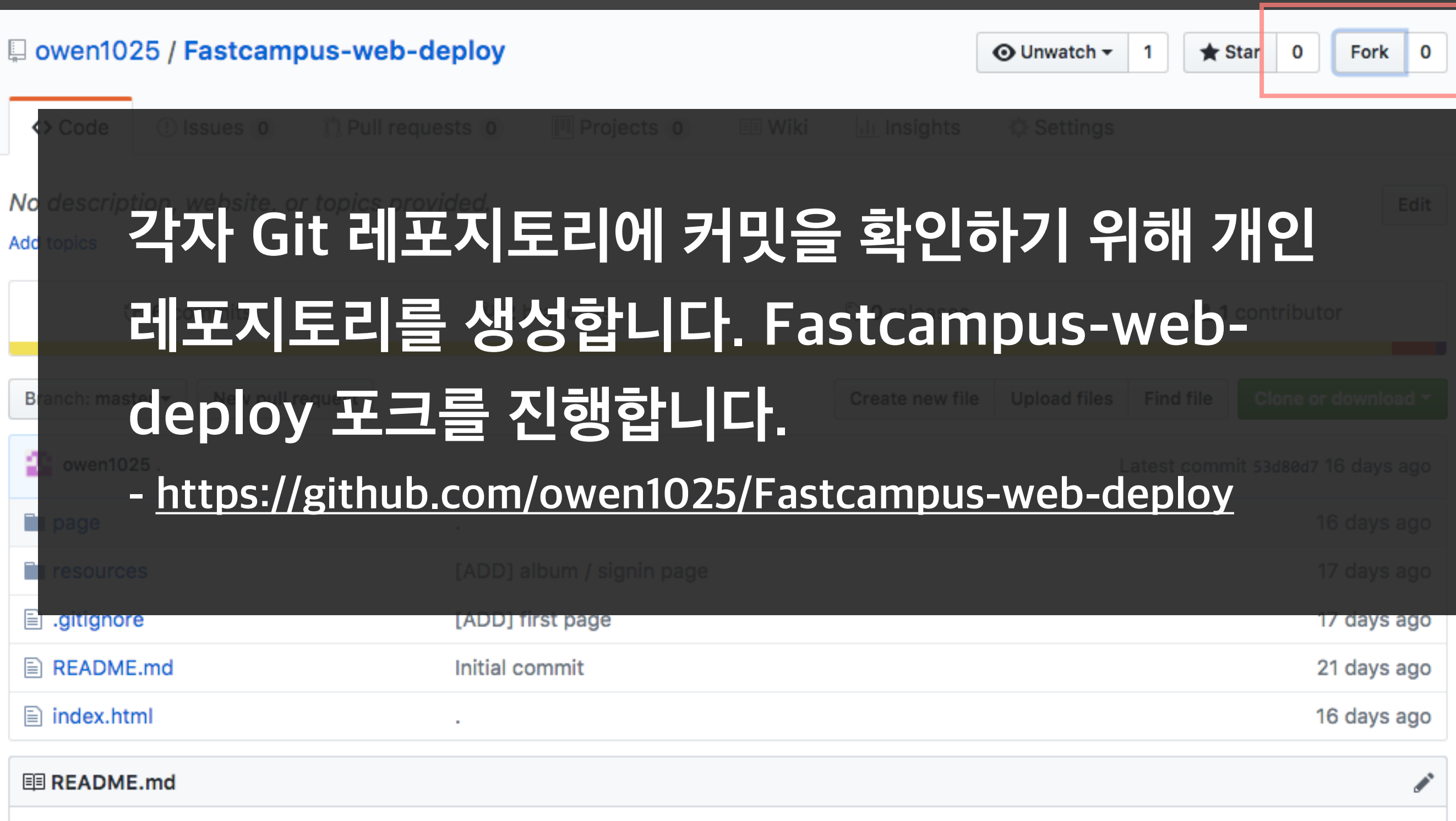
Jenkins



Jenkins

- 파이프라인을 통해 **지속적인 통합(CI), 전달(CD)** 환경 구축을 위한 툴
- 거의 대부분의 언어를 지원 (자바 환경에 가장 적합하다고 생각합니다)
- CI/CD 툴 중 가장 많은 third-party 지원
- 강력한 빌드, 테스트, 배포를 통합적으로 지원

Jenkins 배포 아이템 생성



owen1025 / Fastcampus-web-deploy

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided

각자 Git 레포지토리에 커밋을 확인하기 위해 개인 레포지토리를 생성합니다. Fastcampus-web-deploy 포크를 진행합니다.

- <https://github.com/owen1025/Fastcampus-web-deploy>

Branch: master

Create new file Upload files Find file Clone or download

owen1025 Latest commit 53d80d7 16 days ago

page		16 days ago
resources	[ADD] album / signin page	17 days ago
.gitignore	[ADD] first page	17 days ago
README.md	Initial commit	21 days ago
index.html	.	16 days ago

README.md

Jenkins 구축

Jenkins가 구동될 EC2 인스턴스를 생성

- Jenkins는 8080포트로 통신하기에 해당 인스턴스의 보안그룹에 8080포트를 허용합니다.

Jenkins 구축

```
1. ec2-user@ip-172-31-25-131:~ (bash)
a1@1ui-MacBook-Air:~$
sudo ssh -i ~/Desktop/ec2_test.pem ec2-user@{jenkins 인스턴스 DNS 주소} ↵
sudo su
yum -y update
yum install -y java-1.8.0
yum remove java-1.7.0-openjdk
wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
rpm --import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key
yum install -y jenkins
```

Jenkins 구축

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

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

- 1) yum -y update
- 2) yum install -y java-1.8.0
- 3) wget -O /etc/yum.repos.d/jenkins.repo <http://pkg.jenkins-ci.org/redhat/jenkins.repo>
- 4) rpm --import <http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key>
- 5) yum install -y jenkins

1. yum 레포지토리를 최신 상태로 업데이트
2. java8 설치 (jenkins는 자바 환경에서 개발됨. 구동을 위해 설치)
3. yum이 어디서 jenkins를 설치해야 할지 알 수 있도록 Jenkins repository를 추가
4. Jenkins 를 설치할 때, 파일들이 신뢰할 수 있는 source 로 부터 제공됨을 증명하기 위해 로컬 GPG 키링에 Jenkins GPG key 를 추가
5. yum 레포지토리로 jenkins 설치

Jenkins 구축

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

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

yum install -y git

- git 연동을 위해 설치

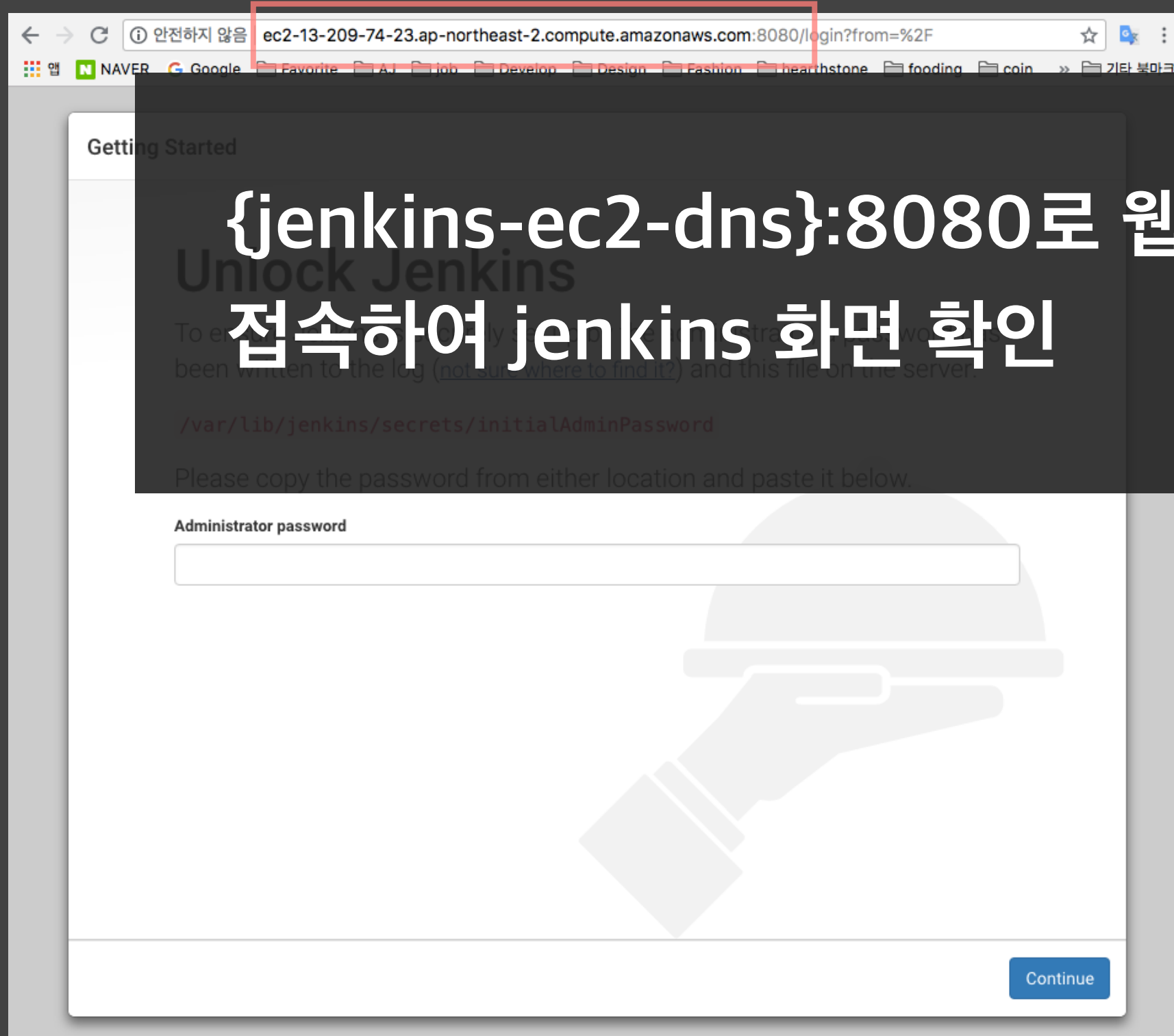
service jenkins start

- 젠킨스 구동

netstat -na | grep 8080

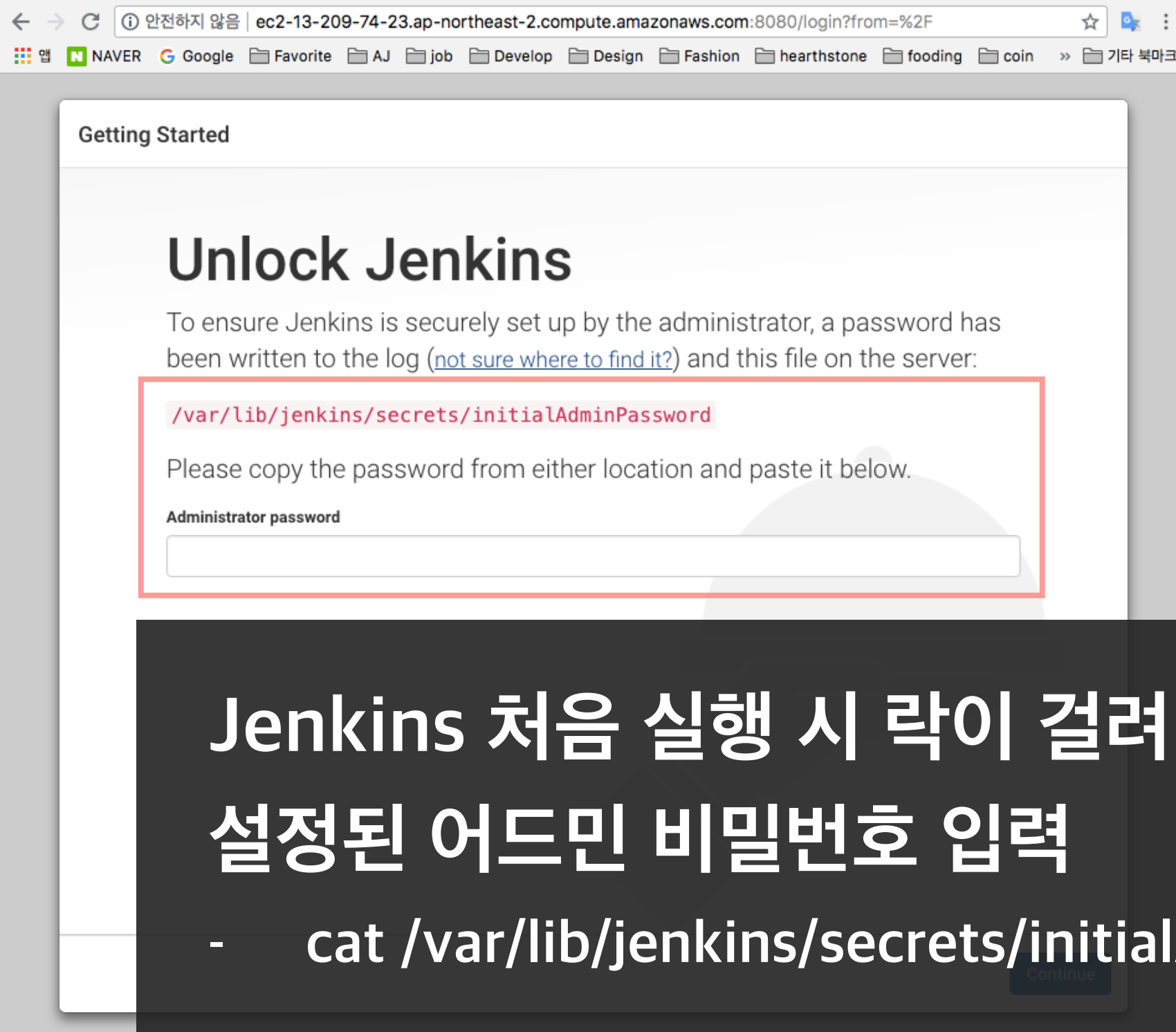
- Jenkins가 구동 됨에 따라 8080포트가 열려있는 지(제대로 실행되고 있는 지) 확인

Jenkins 구축



{jenkins-ec2-dns}:8080로 웹브라우저에서
접속하여 jenkins 화면 확인

Jenkins 구축



Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Jenkins 처음 실행 시 락이 걸려있어 기본 값으로
설정된 어드민 비밀번호 입력

- `cat /var/lib/jenkins/secrets/initialAdminPassword`

Jenkins 배포 아이템 생성

The screenshot shows the Jenkins 'New Item' configuration page. The 'General' tab is selected. The 'Project url' field is highlighted with a red box and contains the URL 'https://github.com/owen1025/Fastcampus-web-deploy.git'. The 'GitHub project' checkbox is checked. The 'Description' field is empty. The 'Build' tab is also visible in the background.

어떤 Github 프로젝트를 기반으로 할 것인지 프로젝트의 URL을 입력

- 전 단계에서 포크(Fastcampus-web-deploy)한 프로젝트의 url 입력

Jenkins 배포 아이템 생성

소스 코드 관리

☐ None
☒ Git

Repositories

Repository URL

Credentials

Branches to build

Branch Specifier (blank for 'any')

Repository browser (자동)

- Repository URL : 앞서 적은 git의 url을 입력합니다.
- Credentials : Github 계정 정보를 생성하여 선택합니다.
- Branch Specifier : 어떤 브랜치에서 Github webhook 메시지가 왔을 때 아이템을 실행할 것인지 결정합니다.

Jenkins 배포 아이템 생성



Jenkins Credentials Provider: Jenkins



Add Credentials

Domain

Kind

Scope

Username

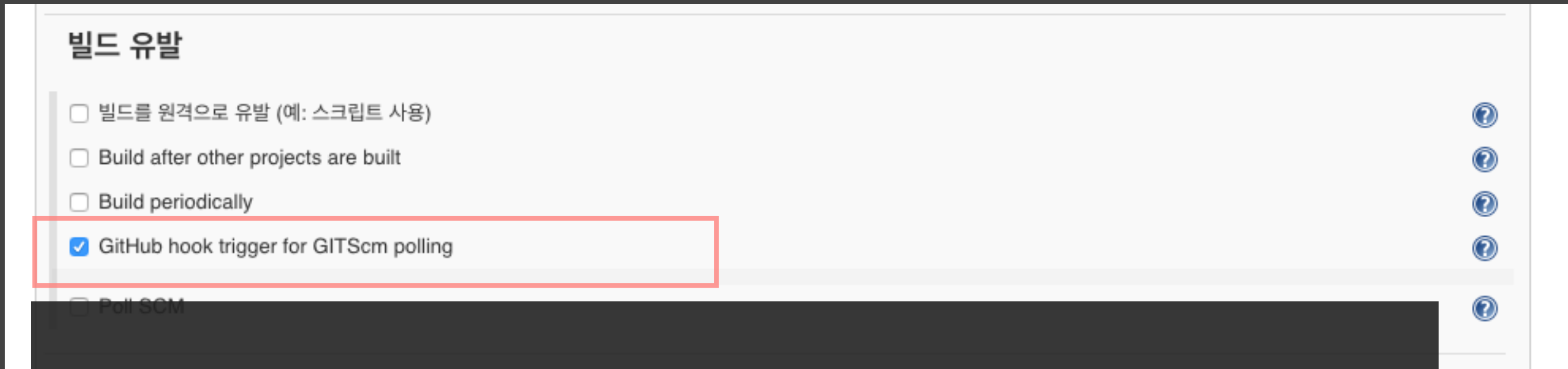
Password

ID

Description

- kind : 인증 방식입니다. Github ID/password, SSH, secret file 등 다양한 방식을 적용할 수 있습니다. Username with password를 선택합니다.
- Username : Github ID를 입력합니다.
- Password : Github password를 입력합니다.
- ID : 해당 계정 정보를 인식하기 위한 Jenkins ID 설정입니다.

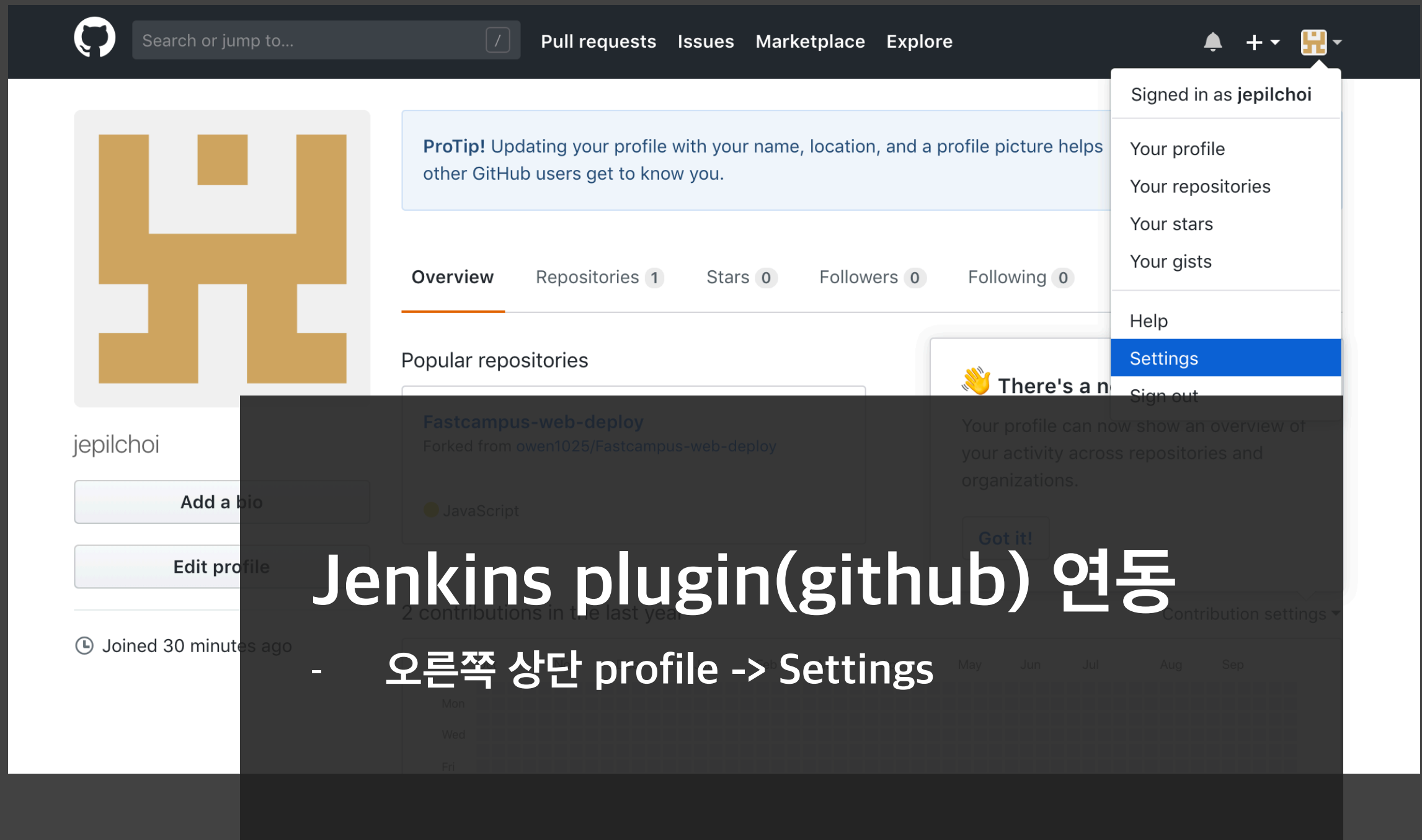
Jenkins 배포 아이템 생성



GitHub hook trigger for GITScm polling

- Github webhook에서 이벤트 발생(push, commit 등)시 해당 아이템을 실행하겠다는 옵션입니다.

Github webhook 설정



The screenshot shows the GitHub profile page for user 'jepilchoi'. The user has a bio placeholder, 1 repository, 0 stars, 0 followers, and 0 following. A 'ProTip!' banner suggests updating the profile. The 'Settings' link in the top right navigation menu is highlighted. A dark overlay with white text is positioned over the lower part of the page.

Jenkins plugin(github) 연동

- 오른쪽 상단 profile -> Settings

Github webhook 설정

Settings / Developer settings

OAuth Apps

GitHub Apps

Personal access tokens

New personal access token

Personal access tokens function like ordinary Git credentials. They can be used to authenticate to the API over Basic Authentication, or can be used to authenticate to the API over Basic Authentication.

Token description

jenkins

What's this token for?

Select scopes

Scopes define the access for personal tokens. [Read more about scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> admin:org	Full control of orgs and teams
<input checked="" type="checkbox"/> write:org	Read and write org and team membership
<input checked="" type="checkbox"/> read:org	Read org and team membership
<input type="checkbox"/> admin:public_key	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys
<input checked="" type="checkbox"/> admin:repo_hook	Full control of repository hooks
<input checked="" type="checkbox"/> write:repo_hook	Write repository hooks
<input checked="" type="checkbox"/> read:repo_hook	Read repository hooks
<input checked="" type="checkbox"/> admin:org_hook	Full control of organization hooks
<input type="checkbox"/> gist	Create gists

Jenkins plugin(github) 연동

- Personal access tokens 탭을 통해 token 발급
- 필요한 권한 부여

Repo

Admin:org

Admin:repo_hook

admin:org_hook

Github webhook 설정

GitHub

GitHub Servers

GitHub Server

Name

my-github-server

API URL

https://api.github.com

Credentials

github-secret

Add

Test connection

Manage hooks ☒

고급...

삭제

고급...

Jenkins plugin(github) 연동

- Jenkins 관리 > 시스템 설정 > Github 탭
- 전 단계에서 발급받은 token을 Credentials 탭에 설정 후 'Test connection'을 통해 연결 확인

Github webhook 설정

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscription. You can choose which data format you'd like to receive (JSON, x-www-form-urlencoded, etc.) in [our developer documentation](#).

Payload URL *

Content type

Secret

Which events would you like to trigger this webhook?

☒ Just the push event.

☐ Send me everything.

☐ Let me select individual events.

☒ **Active**
We will deliver event details when this hook is triggered.

Add webhook

Jenkins plugin(github) 연동

- 어떤 이벤트(push, commit 등)가 발생시 jenkins에 게 알림(github-webhook)을 보낼 것인지를 선택합니다.

Github webhook 설정

Jenkins plugin(github) 연동

- webhook 알림을 보내기 위해 Jenkins가 구동되는 URL을 입력합니다.
- {jenkins-ec2-dns/ip}:8080/github-webhook/

Services / Add Jenkins (GitHub plugin)

Note: GitHub Services are being deprecated. Please contact your integrator for more information on how to migrate or replace a service with webhooks or GitHub Apps.

Jenkins is a popular continuous integration server.

Using the Jenkins GitHub Plugin you can automatically trigger builds when pushes are made to GitHub.

Install Notes

1. "Jenkins Hook Url" is the URL of your Jenkins server's webhook endpoint.
example: `http://ci.jenkins-ci.org/github-webhook/`

For more information see <https://wiki.jenkins-ci.org/display/JENKINS/GitHub+plugin>

Jenkins hook url

`ec2-northeast-2.compute.amazonaws.com:8080/github-webhook/`

☒ **Active**
We will run this service when an event is triggered.

Add service

Jenkins + github webhook 연결 확인

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

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

sudo su

git clone {fork한 Fastcampus-web-deploy 프로젝트 git 주소}

cd Fastcampus-web-deploy

vim index.html

git add —all

git commit -m “jenkins test”

git push -u origin master
```

Jenkins + github webhook 연결 확인

All +

S


W


Name ↓

최근 성공

최근 실패

최근 소요 시간






[web-cd-test](#)

23 sec - [#1](#)




—

2 sec



아이콘: [S](#) [M](#) [L](#)

[상세 내용 입력](#)

[Legend](#)  [RSS 모두](#)  [RSS 실패](#)  [RSS 최근 빌드](#)

Jenkins + ssh 자동화 배포

Jenkins Plugin Manager

검색

최제필 | 로그아웃

대시보드로 돌아가기

Jenkins 관리

필터: ssh

업데이트된 플러그인 목록 | **설치 가능** | 설치된 플러그인 목록 | 고급

설치 ↓	이름	버전
<input type="checkbox"/>	SCP_publisher This plugin uploads build artifacts to repository sites using SCP (SSH) protocol. Warning: This plugin version may not be safe to use. Please review the following security notices: <ul style="list-style-type: none">Insecure credential storage and transmission	1.8
<input type="checkbox"/>	SSH This plugin executes shell commands remotely using SSH protocol.	2.6.1
<input type="checkbox"/>	Distributed Fork Turns a Jenkins cluster into a general purpose batch job execution environment through an SSH-like CLI.	1.7
<input type="checkbox"/>	SSH Agent	1.15
<input type="checkbox"/>	SSH Pipeline Steps SSH Pipeline Steps	1.1.0
<input type="checkbox"/>	SSH2 Easy This plugin allows you to ssh2 remote server to execute linux commands , shell , sftp upload, downlaod etc	1.4
<input type="checkbox"/>	Terminate ssh processes This plugin add action delete log to build page. If the build is build of matrix job, the action delete log for all its configurations too.	1.0

재시작 없이 설치하기 | **지금 다운로드하고 재시작 후 설치하기** | Update information obtained: 11 hr ago | **지금 확인**

Jenkins 관리 - 플러그인 관리 - 설치 가능 탭에서
Publish Over SSH 체크 후 '지금 다운로드하고 재시작 후 설치하기' 클릭

Jenkins + ssh 자동화 배포

Publish over SSH

Jenkins SSH Key

Passphrase

Path to key

Key

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEApm6ATXbCyPUP97CBHQ0QFGCbG9+hSfp/T4ep7TJ4IXn2/Z8jWsxOwP8J8P5

```

Disable exec ☐

SSH Servers

SSH Server	
Name	<input type="text" value="nginx-ec2-instance"/>
Hostname	<input type="text" value="52.79.242.89"/>
Username	<input type="text" value="ec2-user"/>
Remote Directory	<input type="text"/>

고급...

Success

Test Configuration

Jenkins 관리 - 시스템 설정 - 화면 중간에 위치한 Publish over SSH
탭으로 이동

Jenkins + ssh 자동화 배포

Publish over SSH

Jenkins SSH Key

Passphrase

Path to key

Key

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEApm6ATXbCyPUP97CBHQ0QFGCbG9+hSfp/T4ep7TJ4lXn2/Z8jWsxOwP8J8P5

Disable exec ☐

SSH Servers

Name	nginx-ec2-instance
Hostname	52.79.242.89
Username	ec2-user
Remote Directory	

Success

Test Configuration

Publish over ssh 플러그인 설정

- Path to key : ssh key가 위치한 경로
- Key : ssh key 본문 입력(Path to key에 해당 경로를 작성하고 ssh key가 있다면 Path to key가 우선 순위가 높음)
- Disable exec : shell script 실행을 막기 위해 사용
- SSH Servers
 - Name : 서버 이름
 - Hostname : 배포할 인스턴스의 Public IP
 - Username : ssh 로그인 시 접속할 계정의 ID(ec2-user)
 - Remote Directory : ssh로 접속시 맨 처음 들어갈 디렉토리의 경로

Jenkins + ssh 자동화 배포

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

cat ~/Desktop/ec2_test.pem

- ssh key 본문 열어보기
- 앞서 설정한 Publish over ssh - key 탭에 해당 내용 입력

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEApm6ATXbCyPUP9x34kNpAw==
...
-----END RSA PRIVATE KEY-----%
```

Jenkins + ssh 자동화 배포

빌드 후 조치

Send build artifacts over SSH

SSH Publishers

SSH Server

Name

고급...

Transfers

Transfer Set

Source files

Remove prefix

Remote directory

Exec command

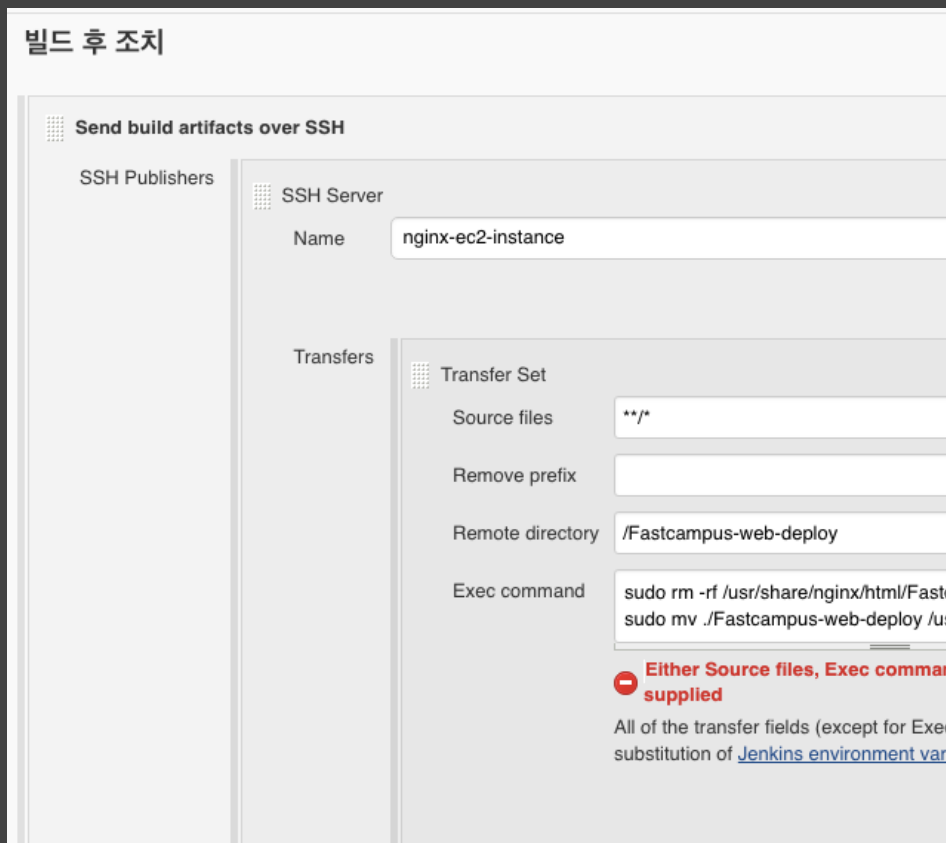
Either Source files, Exec command or both must be supplied

All of the transfer fields (except for Exec timeout) support substitution of [Jenkins environment variables](#)

고급...

Jenkins 관리 - 시스템 설정 - 화면 중간에 위치한 Publish over SSH 탭으로 이동

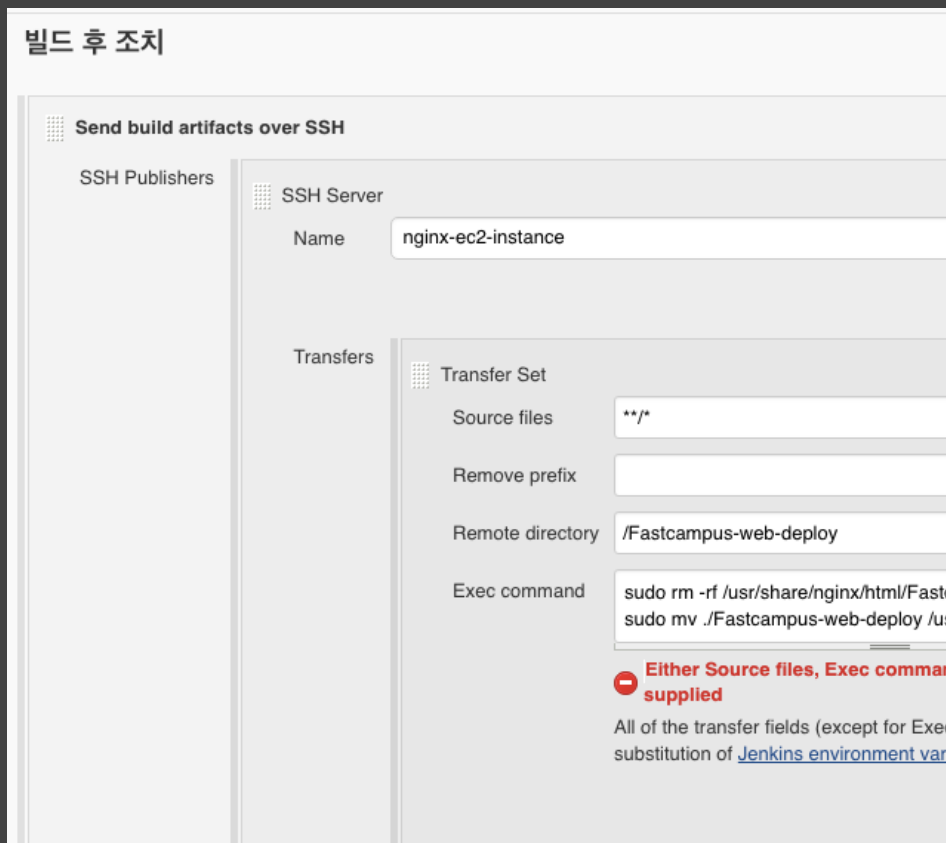
Jenkins + ssh 자동화 배포



Send build artifacts over SSH

- SSH Server : 방금 설정한 SSH Server(배포할 인스턴스, nginx-ec2-instance) 선택
- Transfers
 - Source files : 어떤 파일을 배포할 것인지 선택
 - ****/*** : 모든 파일
 - ****/*.war** : *.war 확장자를 가진 모든 파일
 - Remove prefix : 원격 서버에 배포 후 삭제할 디렉토리
 - Remote directory : 원격 서버에 배포 시 해당 파일이 위치할 디렉토리
 - Exec command : 배포가 끝나고 해당 원격 서버에 실행할 명령어 모음

Jenkins + ssh 자동화 배포






Exec command

- `sudo rm -rf /usr/share/nginx/html/Fastcampus-web-deploy`
기존에 운영되고 있던 프로젝트 삭제
- `sudo mv ./Fastcampus-web-deploy /usr/share/nginx/html/`
배포된 프로젝트 코드를 Nginx root directory로 이동




Jenkins + ssh 자동화 배포

All +

상세 내용 입력

S	W	Name ↓	최근 성공	최근 실패	최근 소요 시간	
		web-cd-test	8 hr 15 min - #38	9 hr 3 min - #15	1.1 sec	

아이콘: [S](#) [M](#) [L](#)

Legend  RSS 모두  RSS 실패  RSS 최근 빌드

해당 버튼으로 Jenkins item 실행

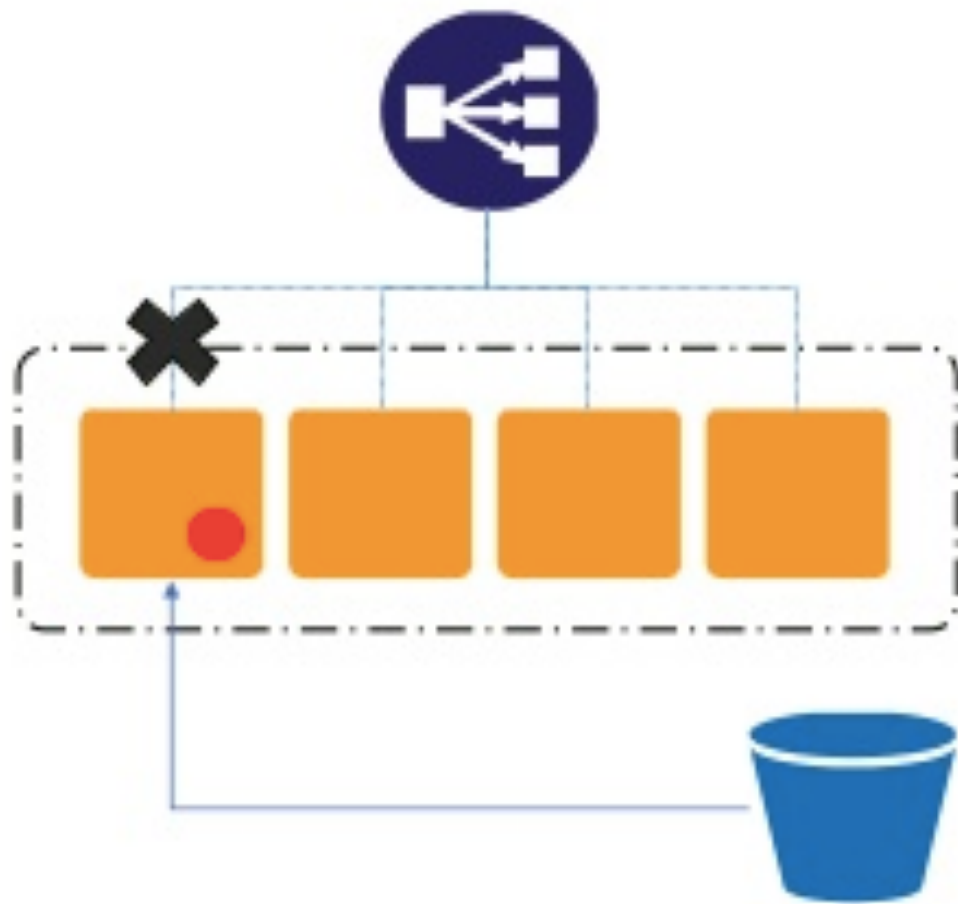
Jenkins + ssh 자동화 배포 - WAS 프로젝트 실습

jenkins - github webhook 설정.pdf를 참고하여
WAS 프로젝트 자동 배포 과정을 진행합니다.

다양한 배포 과정

In-place VS Blue-green deployment

In-place deployment



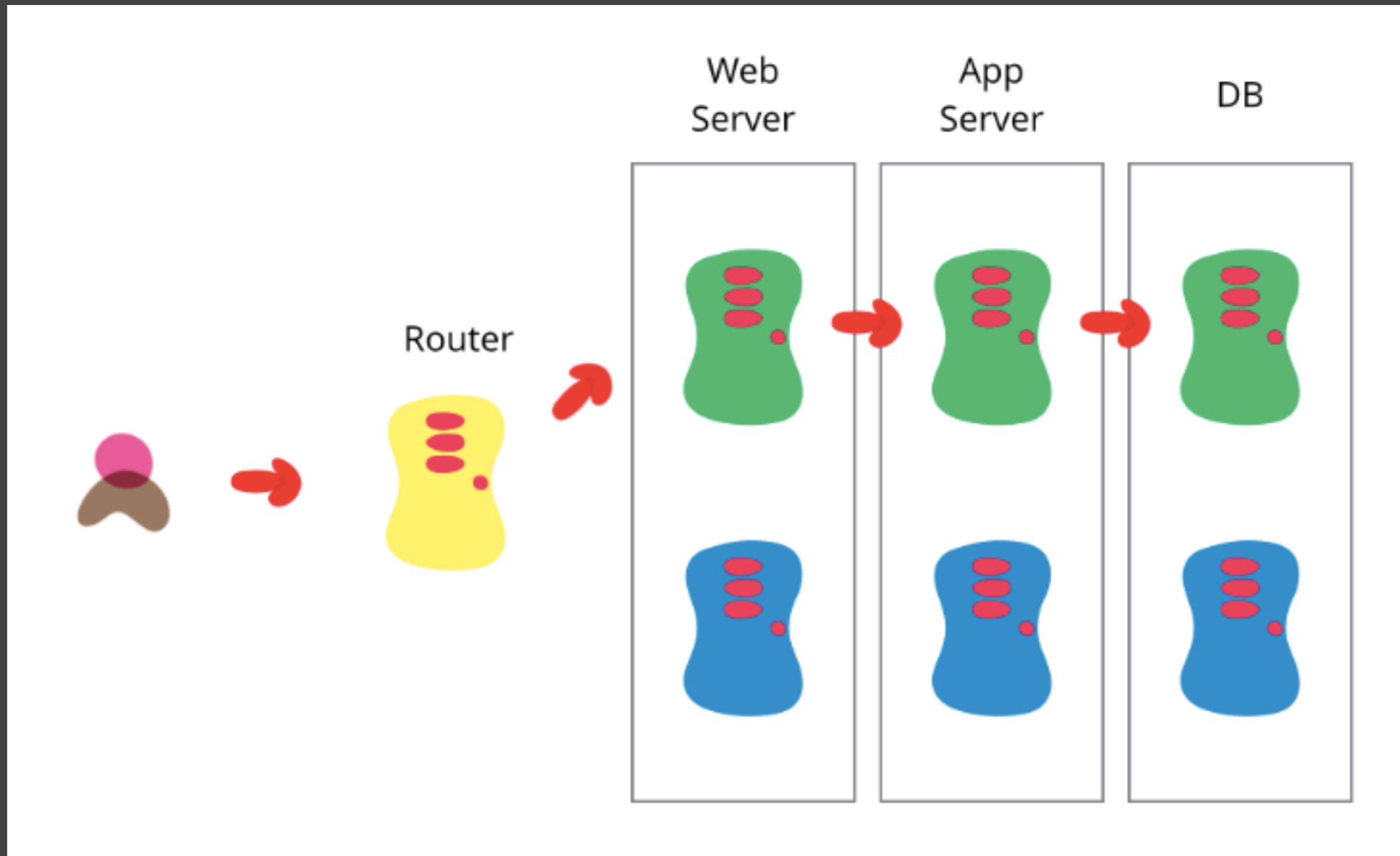
직접 배포

= 중단 배포

= 정기점검

=> AWS CodeDeploy를
활용하여 무중단 배포가 가능

Blue-green deployment



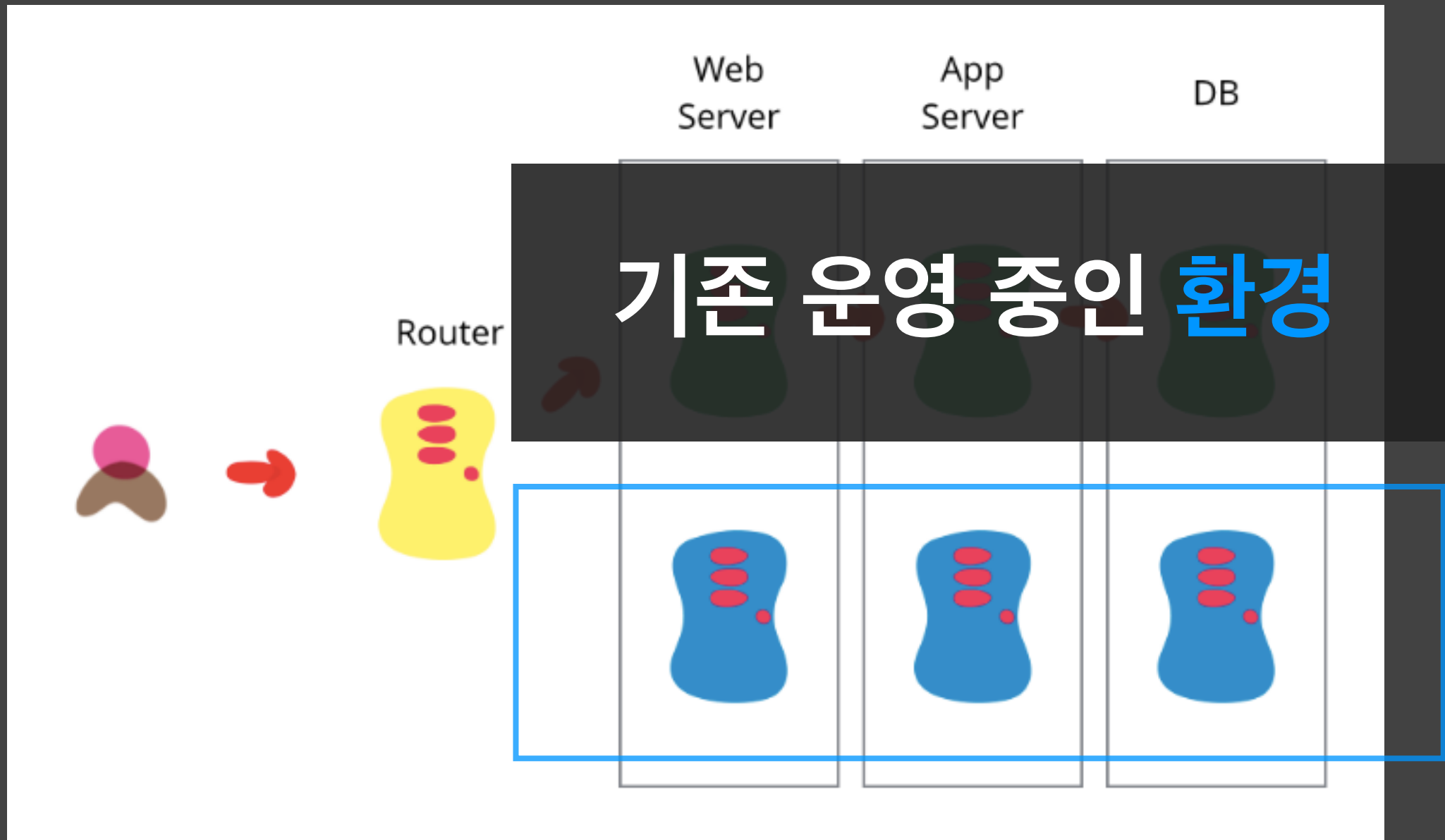
Blue-green deployment

서비스 배포 시 일반적인 자동화 방식은 코드/환경을 배포하는 과정에서 서버의 재시작 등으로 인해 서비스가 중단됨

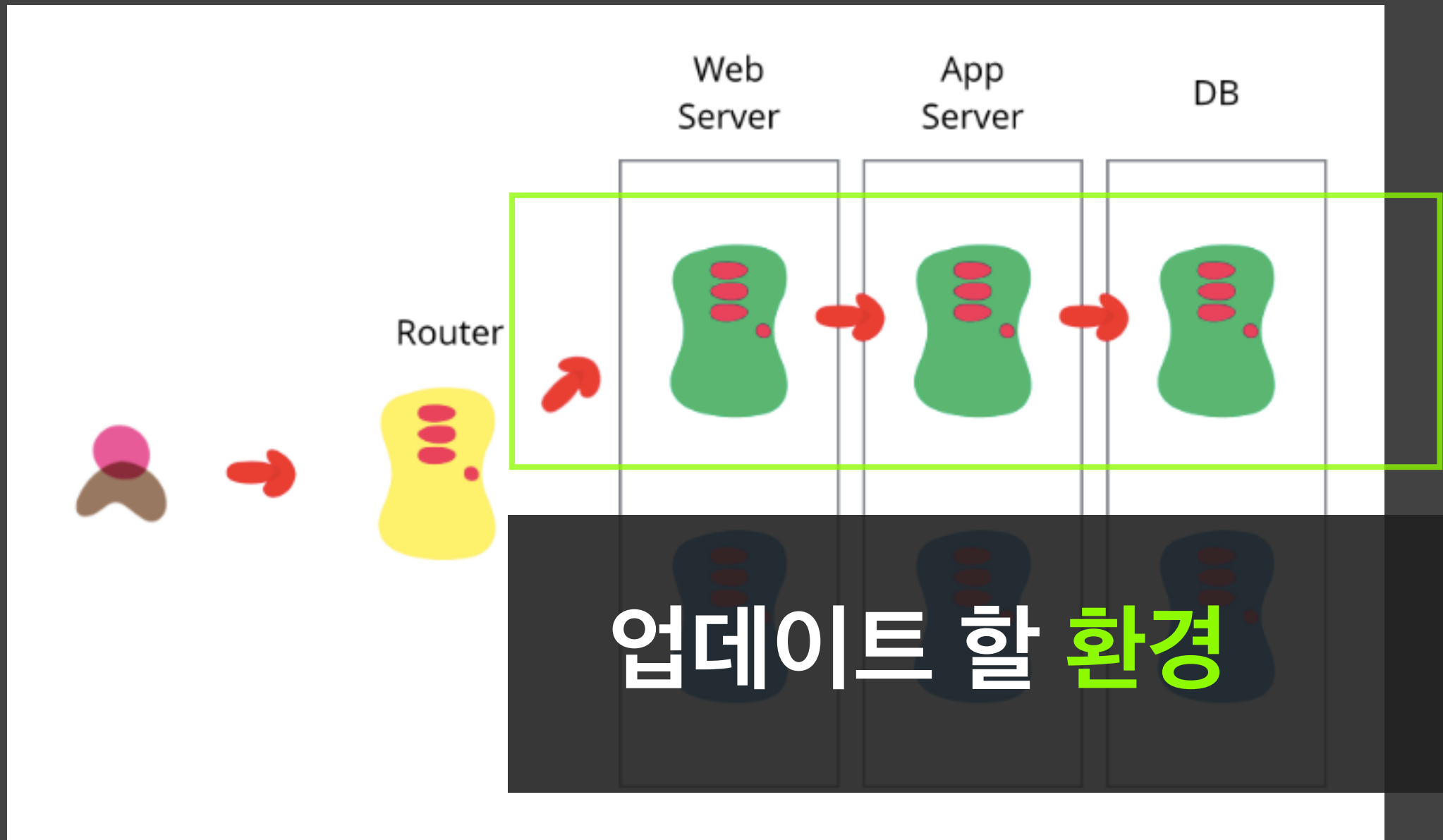
Blue-green deployment

ex) service nginx restart
ex2) pm2 restart WAS
- 모든 연결(Connection)이 **유실**됨

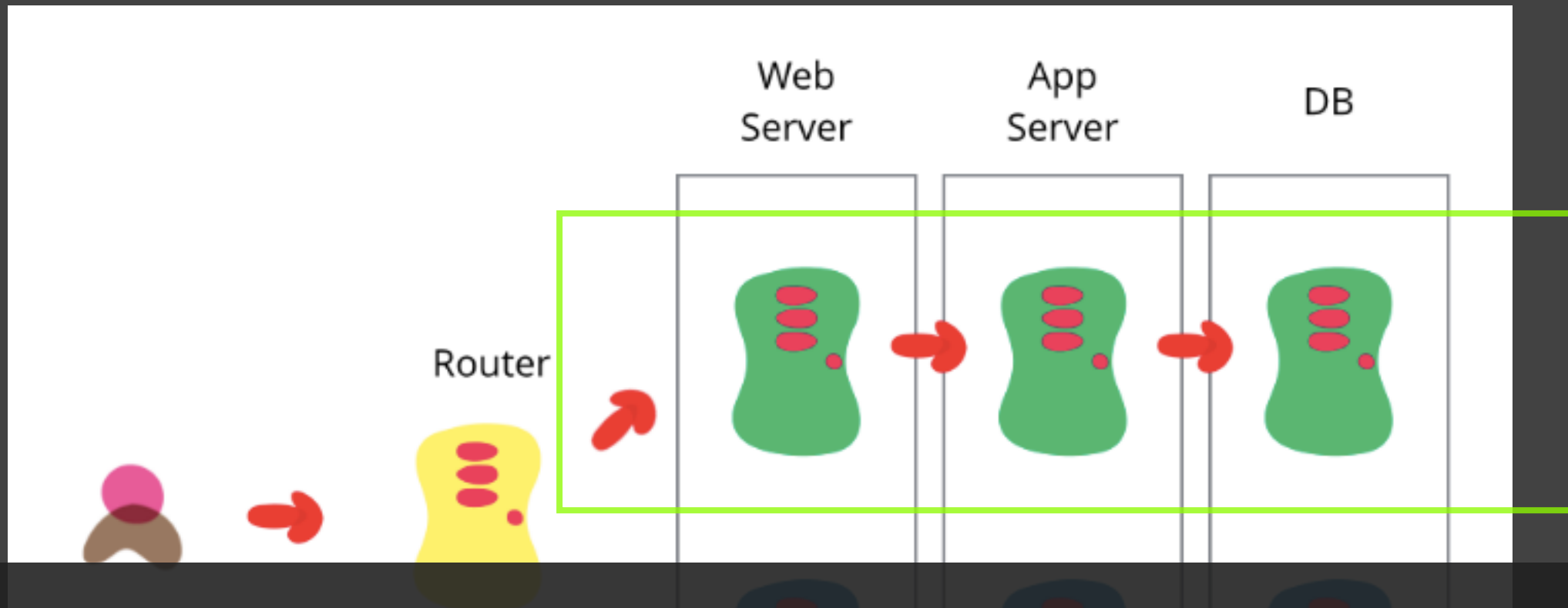
Blue-green deployment



Blue-green deployment

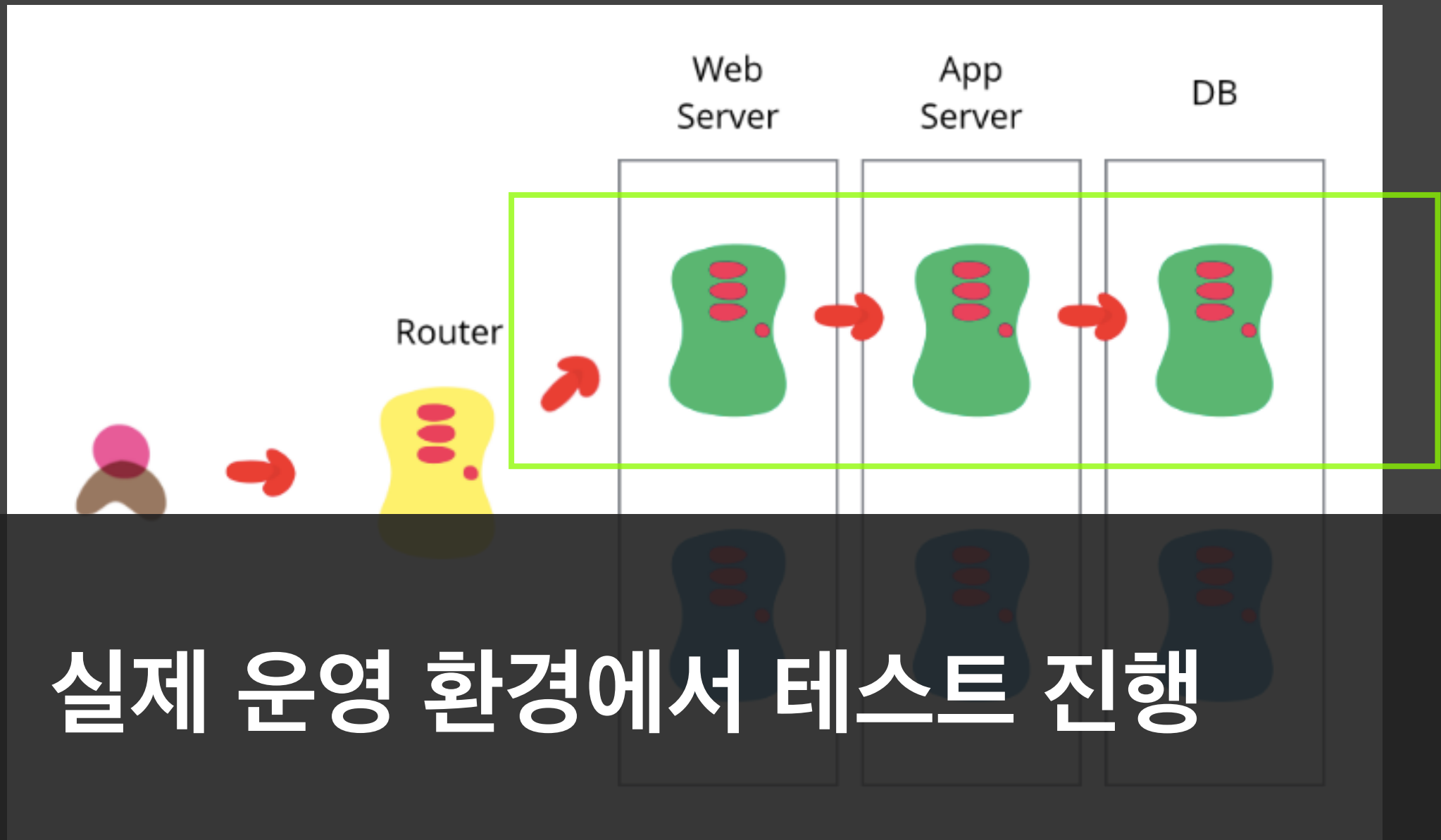


Blue-green deployment



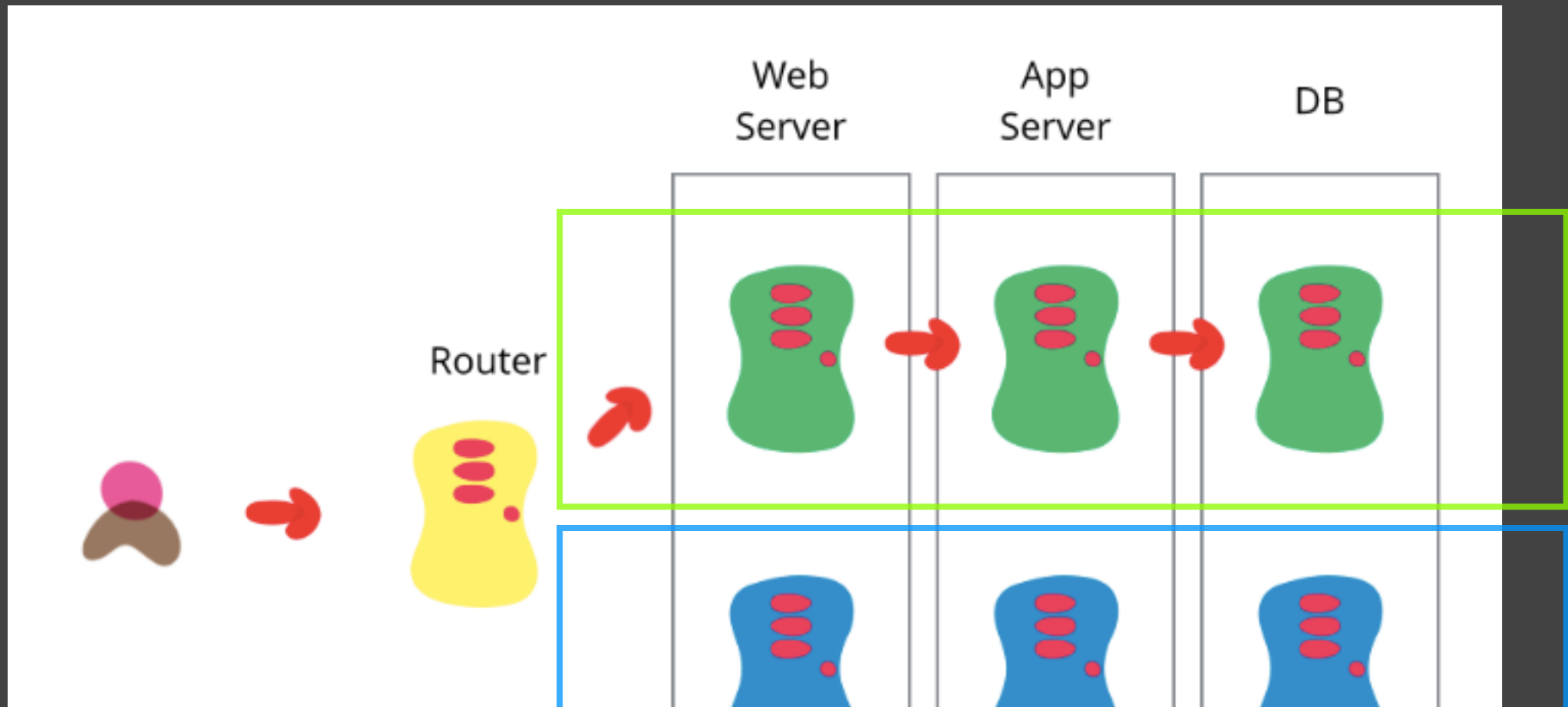
새로운 소프트웨어, 환경 Build ->
Deployment

Blue-green deployment



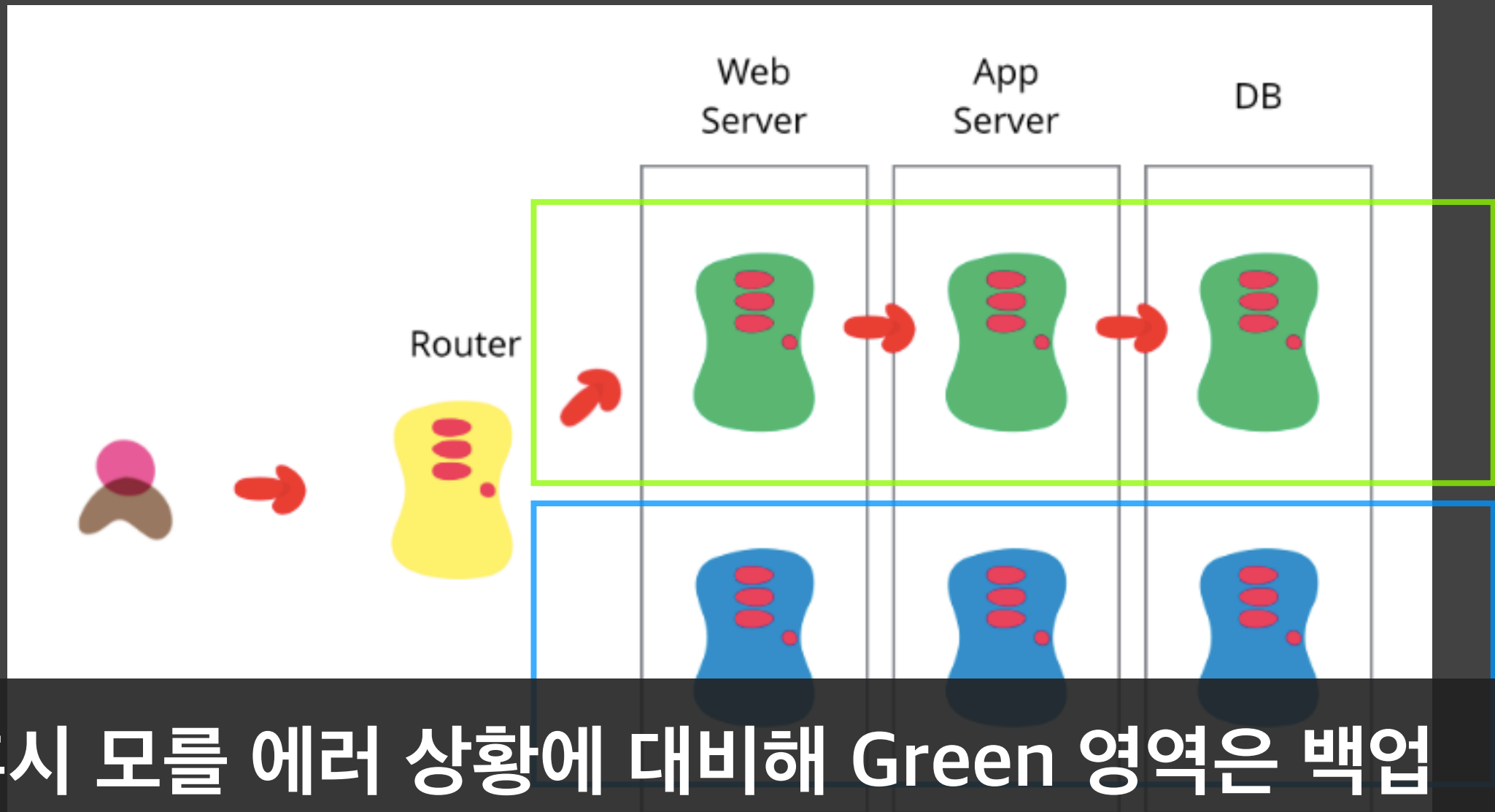
실제 운영 환경에서 테스트 진행

Blue-green deployment



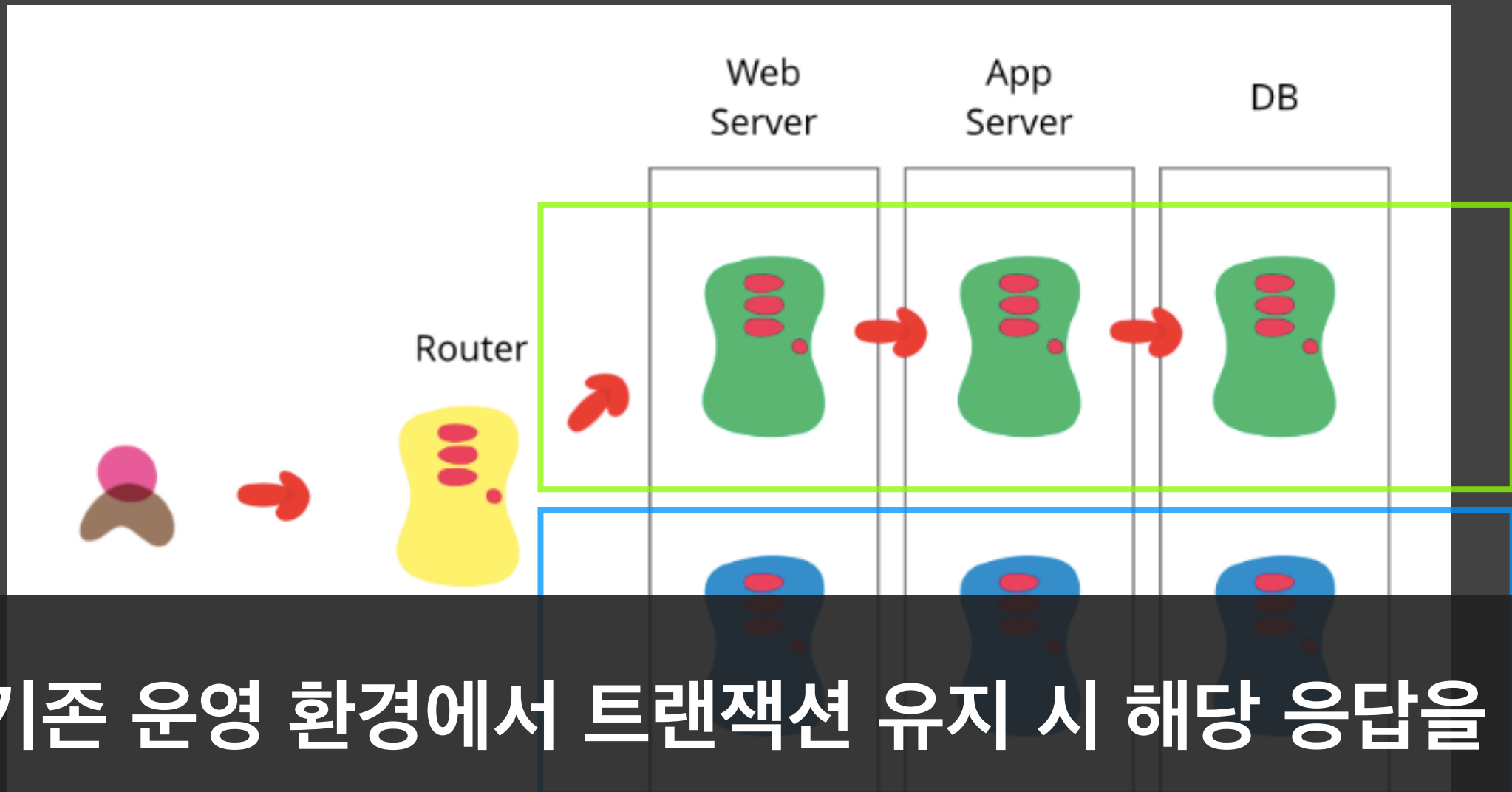
기존 운영 중인 **환경**의 인스턴스를 삭제하고 Green 영역을 운영 환경(Blue 영역)으로 대체

Blue-green deployment



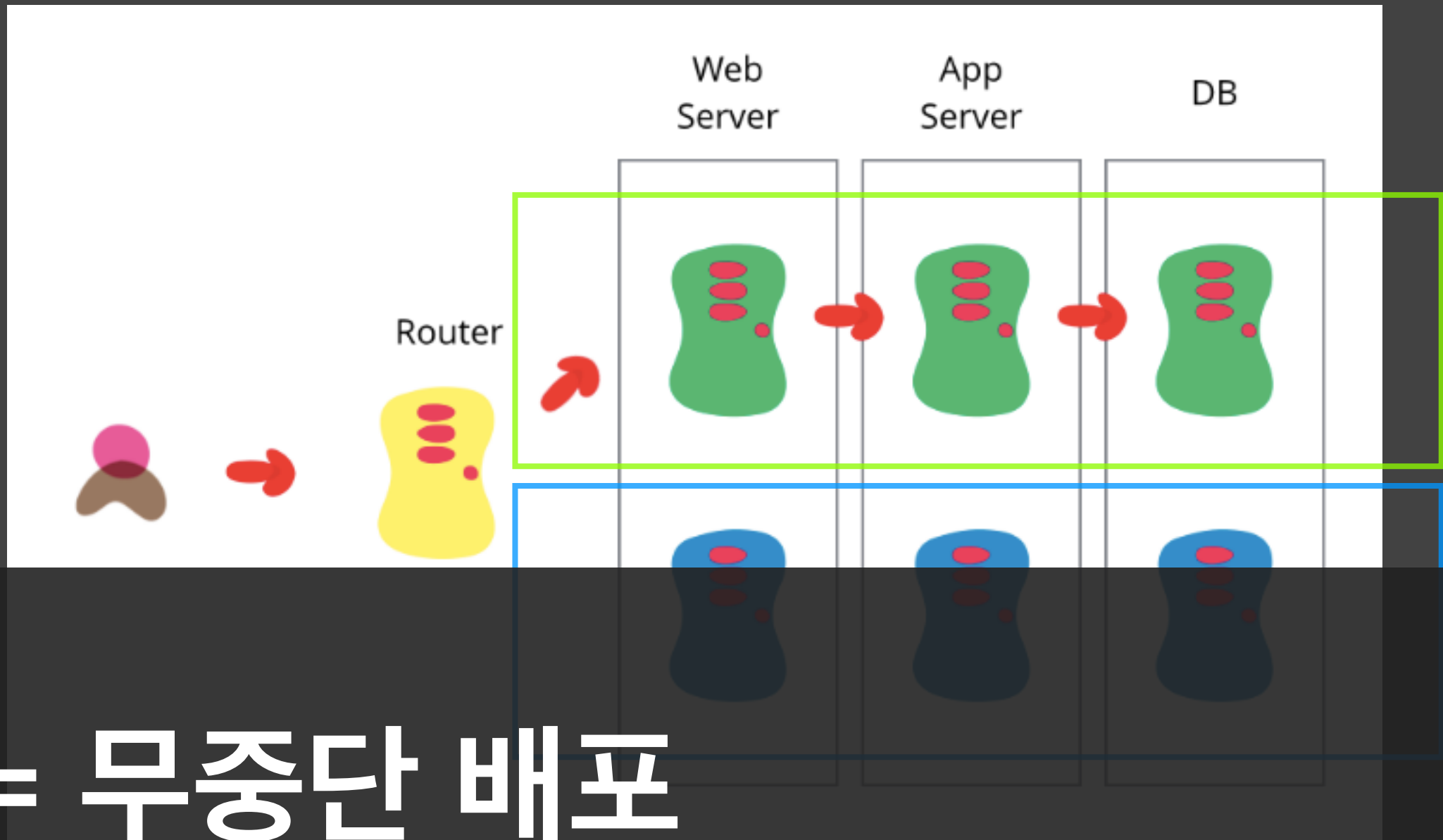
혹시 모를 에러 상황에 대비해 Green 영역은 백업 서버로 대기

Blue-green deployment

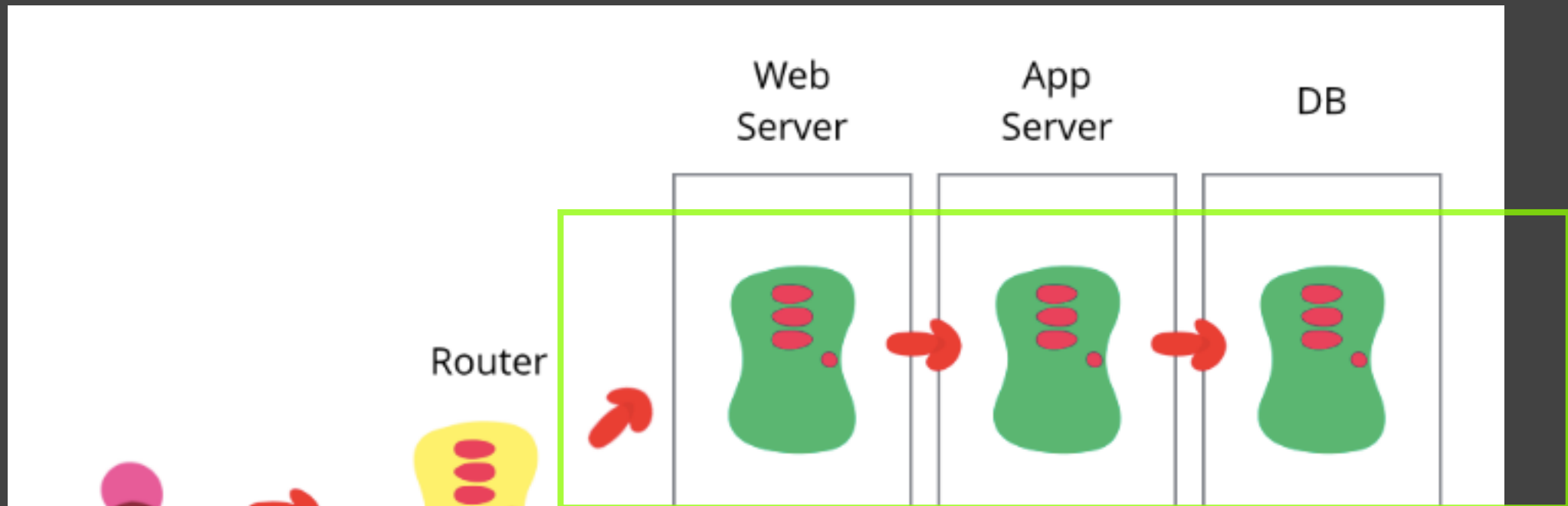


기존 운영 환경에서 트랜잭션 유지 시 해당 응답을
마치고 나서 Blue-green 영역을 switching하여
트랜잭션 보장 가능

Blue-green deployment



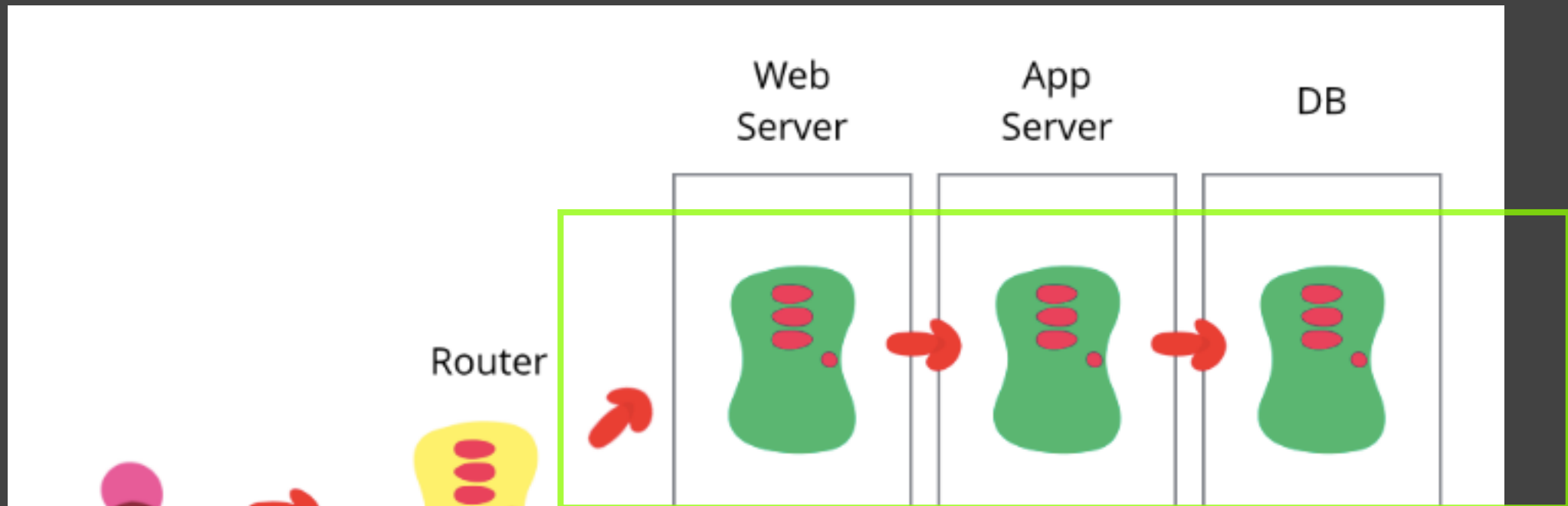
Blue-green deployment



Blue-green 환경은 엄연히 다르지만 동일한 형태를 구성해야 함

- ex 1) 동일한 nginx.conf
- ex 2) 같은 버전의 node.js

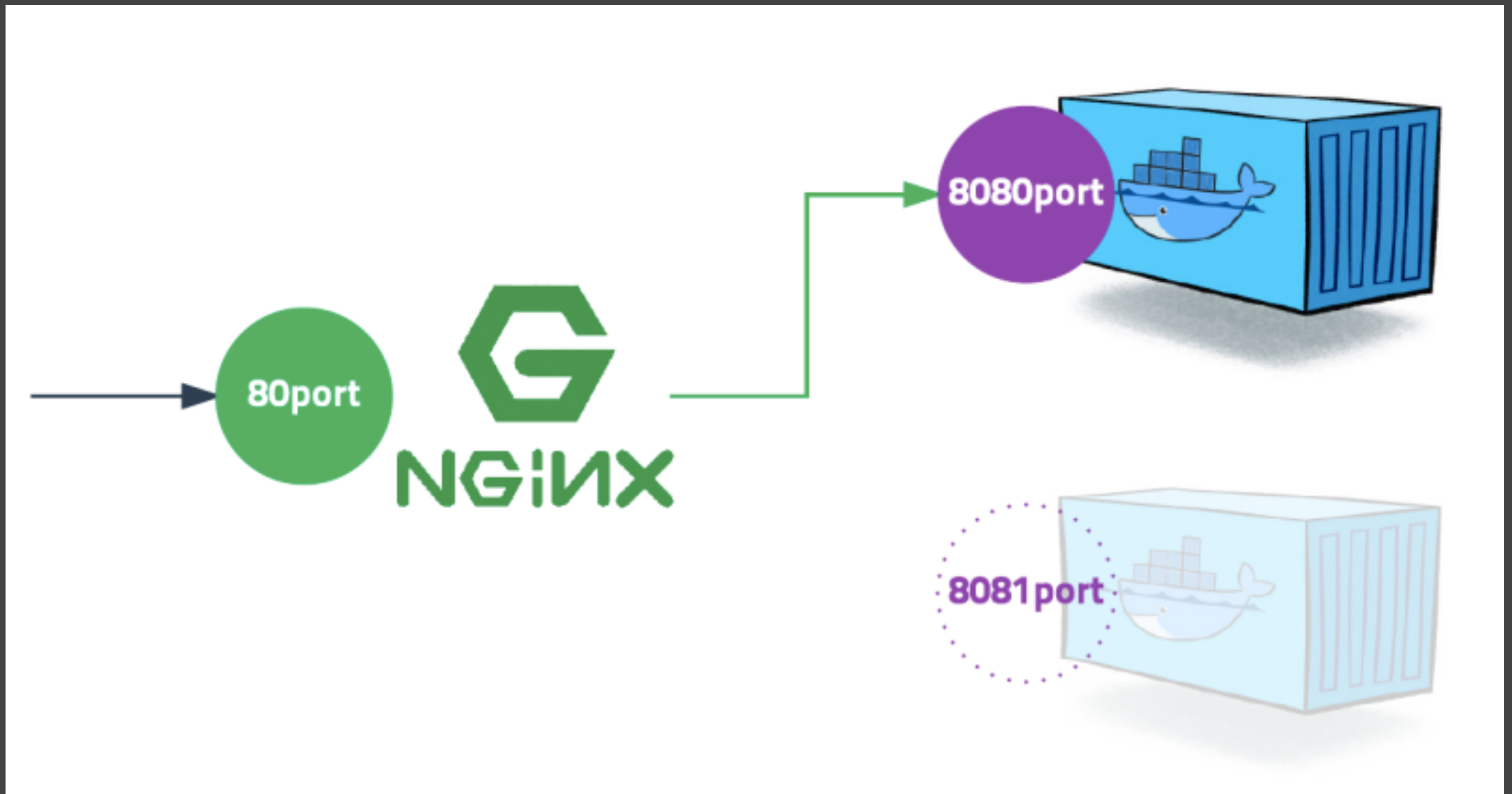
Blue-green deployment



AWS Codedeploy + AMI + Auto scaling
group으로 구축 가능

- nginx 혹은 node.js 등 버전을 올려야 하는(= 같은 환경을 유지해야 하는) 경우가 생긴다면?

컨테이너 환경으로의 이전



CentOS7 + Nginx + Fastcampus-web-deploy 도커 이미지 빌드

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

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

해당 작업은 로컬 환경에서 진행합니다. Fastcampus-web-deploy를 git clone 하기 전 로컬 환경에서 원하는 디렉토리로 이동합니다. (ex. cd ~/Desktop)

git clone {fork한 Fastcampus-web-deploy.git}

CentOS7 + Nginx + Fastcampus-web-deploy 도커 이미지 빌드

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

cd Fastcampus-web-deploy

vim Dockerfile

- 도커 이미지 빌드를 위한 정의 파일인 Dockerfile 생성
- vim 에디터가 아닌 다른 에디터(vs code, sublime 등)를 사용하셔도 무방합니다

CentOS7 + Nginx + Fastcampus-web-deploy 도커 이미지 빌드

Fastcampus-web-deploy/Dockerfile 편집(아래 내용)

```
FROM centos:latest
```

```
RUN yum install -y epel-release
```

```
RUN yum install -y nginx
```

```
COPY ./ /usr/share/nginx/html/Fastcampus-web-deploy
```

```
EXPOSE 80
```

```
CMD ["nginx", "-g", "daemon off;"]
```

CentOS7 + Nginx + Fastcampus-web-deploy

도커 이미지 빌드

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

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

`docker build -t web:0.1 .`

- 도커 이미지 빌드 실행 {image_name:tag}

`docker images`

- 도커 이미지 리스트 확인(web:0.1 이미지 생성이 잘 됐는 지 확인)

`docker run -d -p 8000:80 --name web`

`web:0.1`

- web:0.1 이미지를 기반으로 도커 컨테이너 실행

CentOS7 + Nginx + Fastcampus-web-deploy 도커 이미지 빌드

Fastcampus-web-deploy/Dockerfile 편집(아래 내용)

```
FROM centos:latest
```

```
RUN yum install -y epel-release
```

```
RUN yum install -y nginx
```

```
COPY ./ /usr/share/nginx/html/Fastcampus-web-deploy
```

```
COPY ./nginx.conf /etc/nginx # 해당 내용 추가
```

```
EXPOSE 80
```

```
CMD ["nginx", "-g", "daemon off;"]
```


CentOS7 + Nginx + Fastcampus-web-deploy 도커 이미지 빌드

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

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

```
docker build -t web:0.2 .
```

- 도커 이미지 빌드 실행 {image_name:tag}

```
docker run -d -p 8001:80 --name  
web2 web:0.2
```

- web:0.1 이미지를 기반으로 도커 컨테이너 실행

컨테이너 환경으로의 이전

Docker container

Running processes

Project code & Configuration

Process

OS

컨테이너 환경으로의 이전

Docker image

Running processes

Project code & Configuration

Process

FROM centos:latest

컨테이너 환경으로의 이전

Docker image

Running processes

Project code & Configuration

RUN `yum install -y nginx`

OS

컨테이너 환경으로의 이전

Docker image

Running processes

COPY ...

Process

OS

컨테이너 환경으로의 이전

Docker image

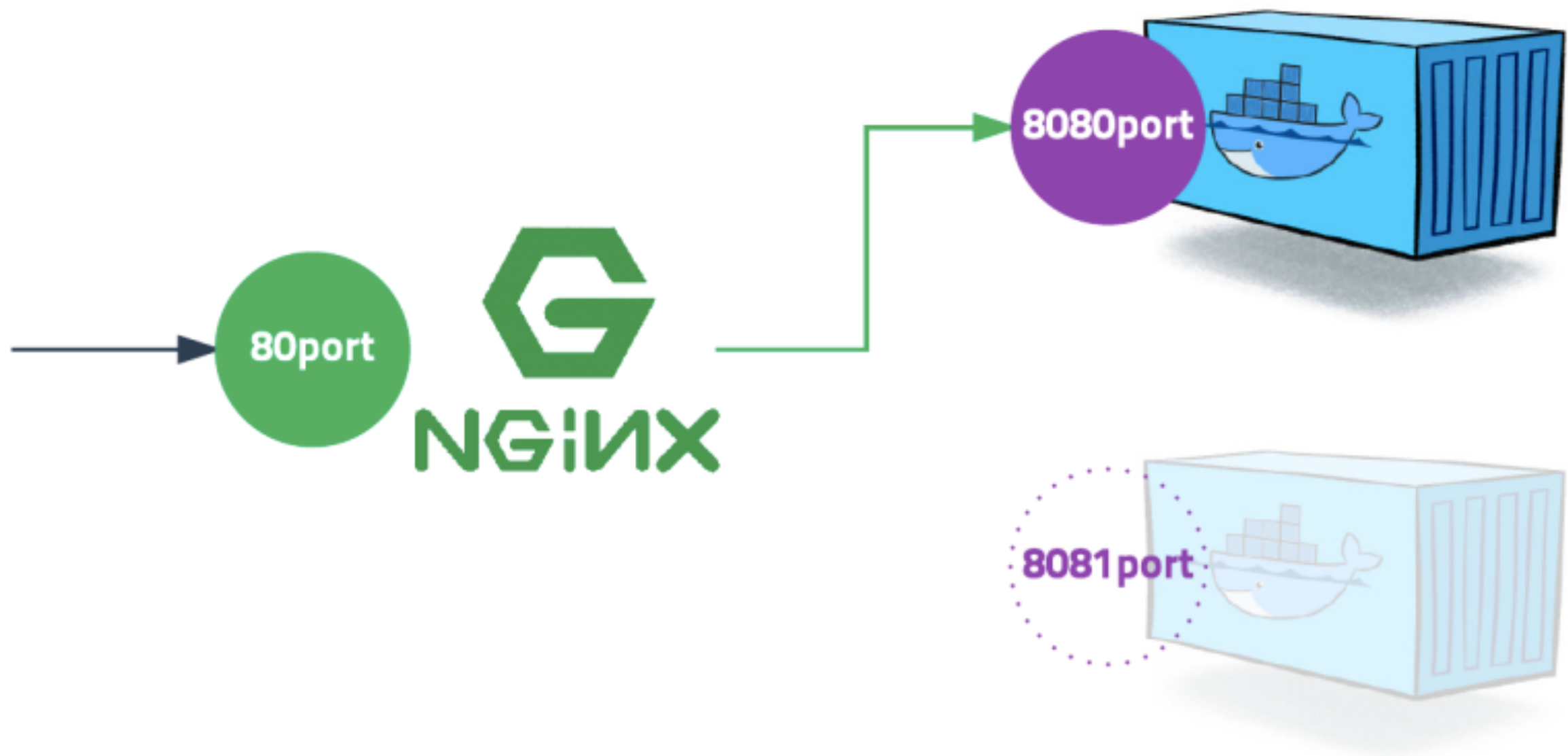
CMD ["nginx","-g","daemon off;"]

Project code & Configuration

Process

OS

컨테이너 환경으로의 이전



컨테이너 환경으로의 이전



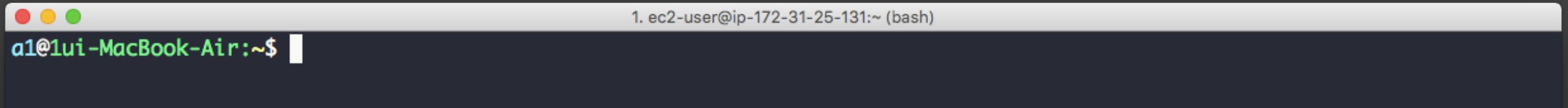
Container(web:0.1)

- 8000 => 80
- <http://.../page/album> ok

Container(web:0.2)

- 8001 => 80
- <http://.../album> ok

컨테이너 환경으로의 이전



1. 운영, 테스트, 개발 환경을 같은 환경에서 제공 가능
2. 환경의 버전 관리 가능(web:0.2)
3. 기존 빌드 환경을 캐싱하여 바뀐 부분만 업데이트
되어 빠르고 간편한 빌드 환경 제공

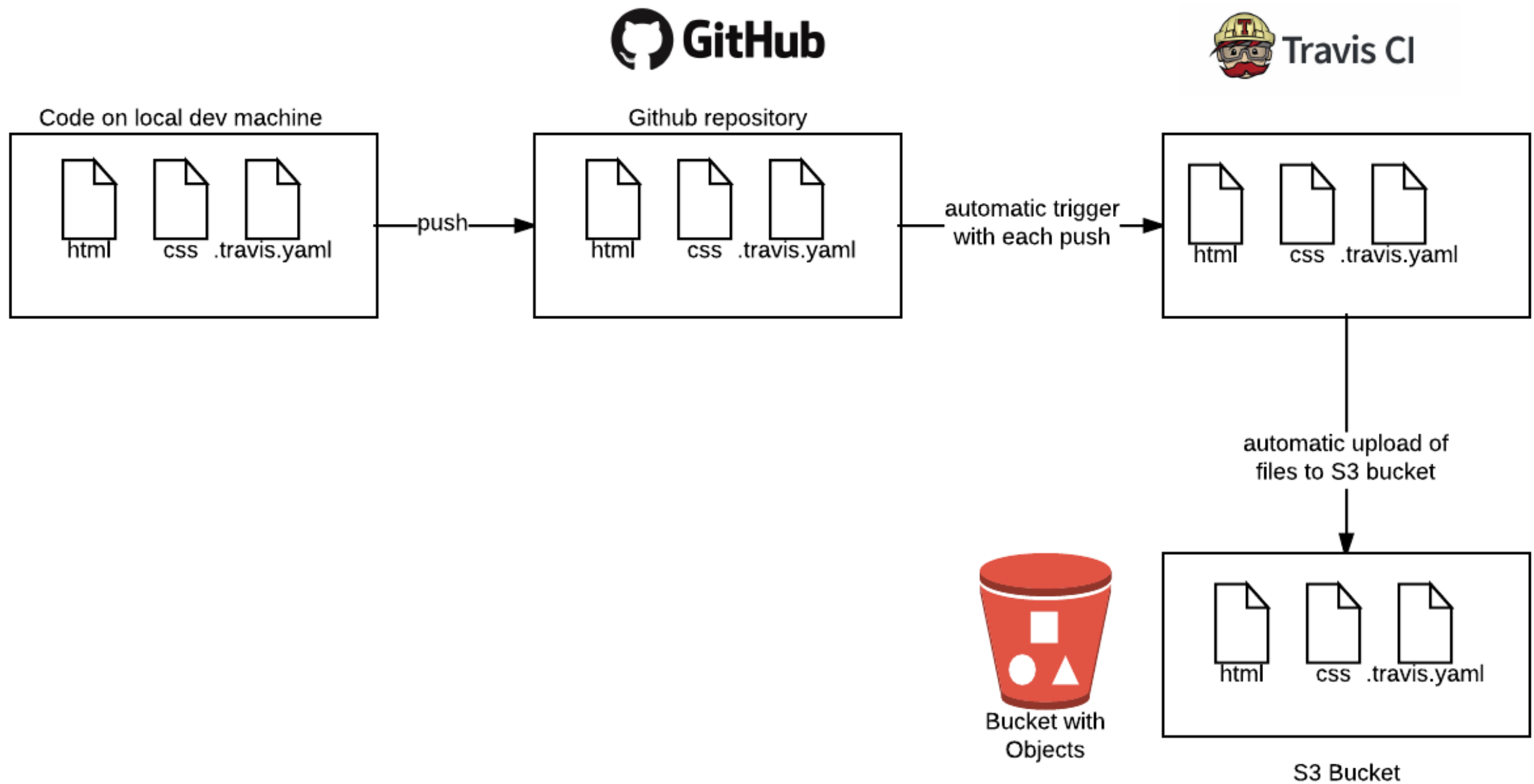
web:0.1 이미지를 기반으로 도커 컨테이너 실행

컨테이너 환경으로의 이전

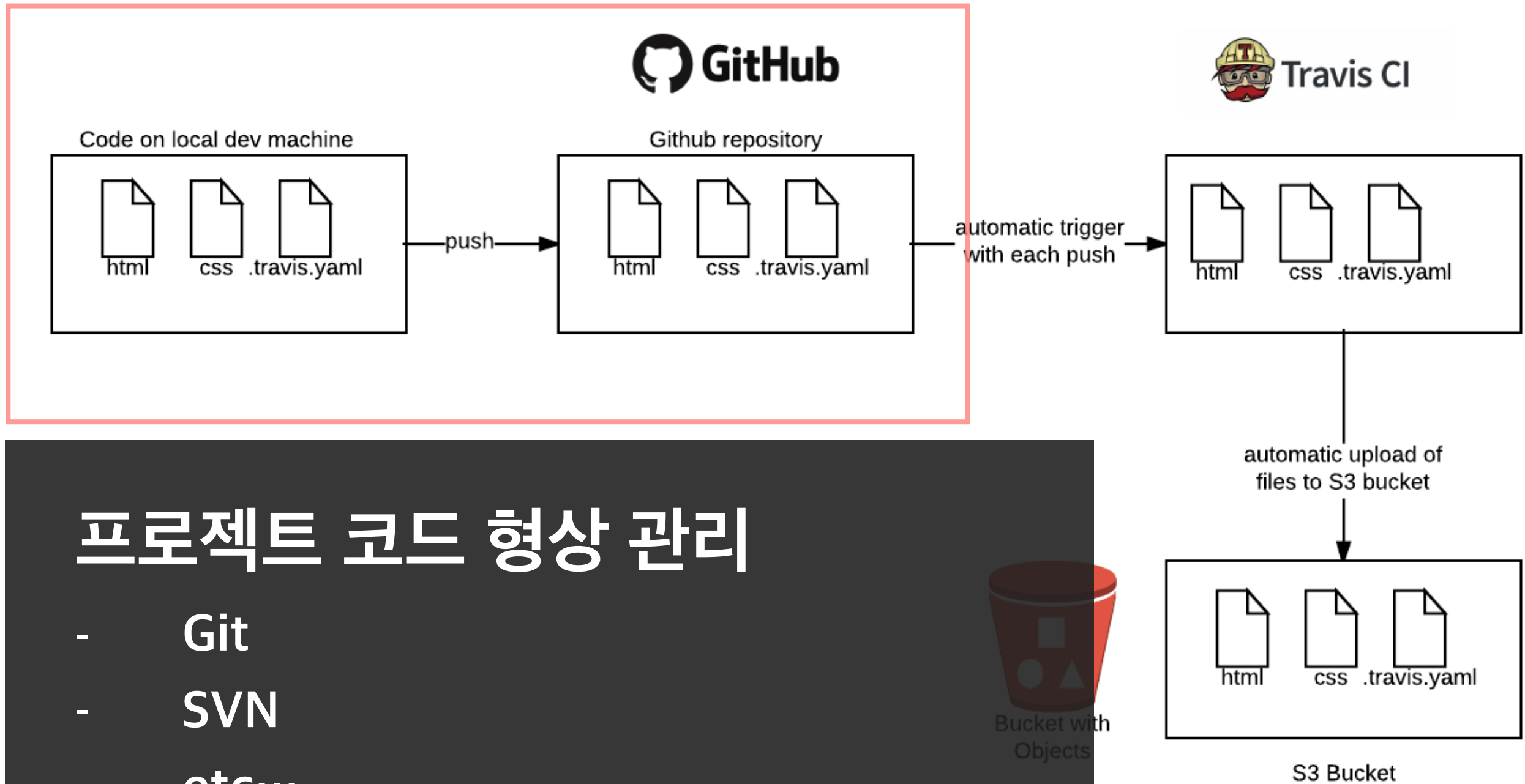
컨테이너 환경을 기반으로 하는
Blue-green(무중단) 배포 환경 구축

=> AWS ECS + ECR + ALB + S3
Codepipeline + Codebuild + Codedeploy

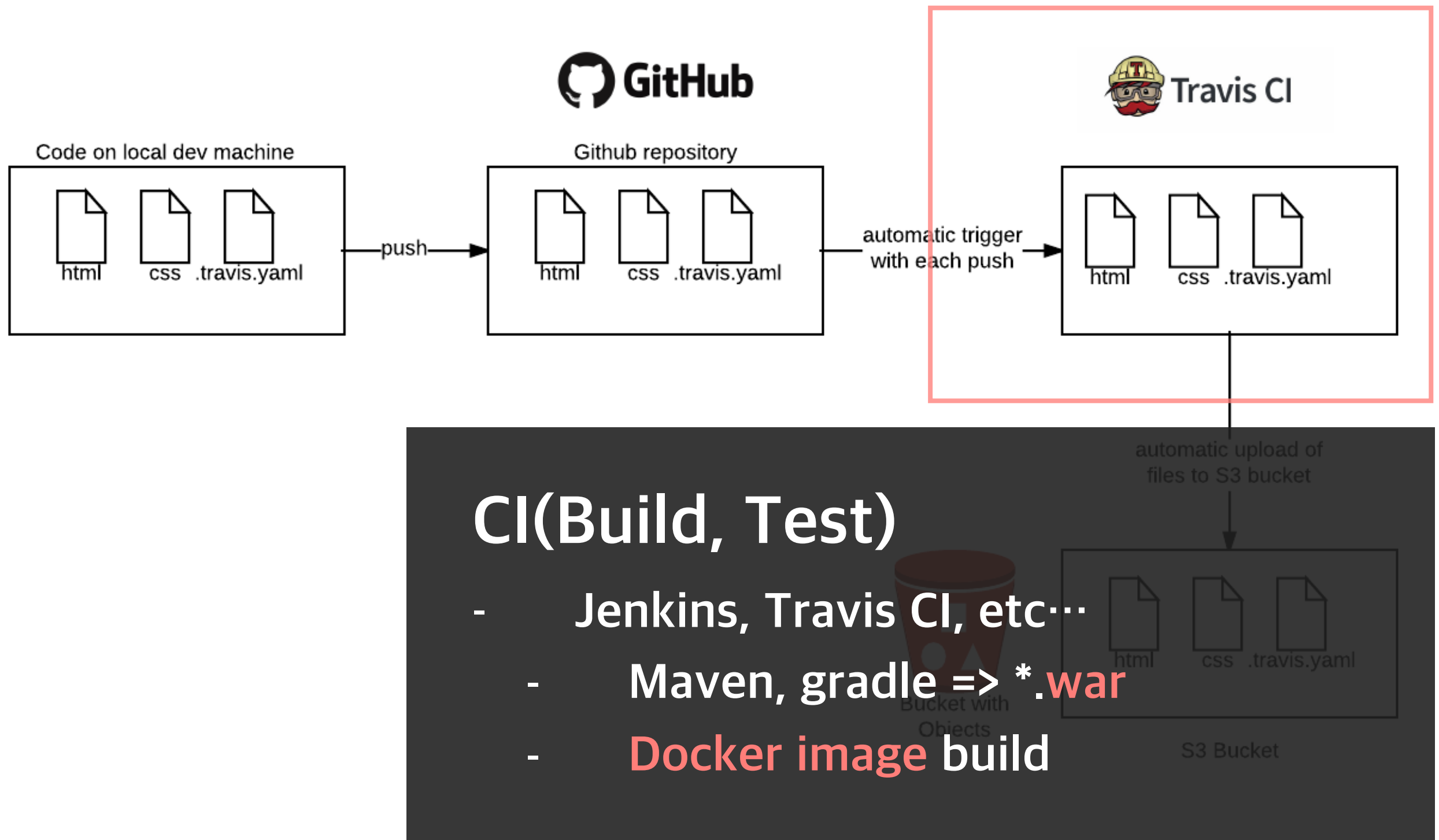
CI(Build, test)에 대해



CI(Build, test)에 대해



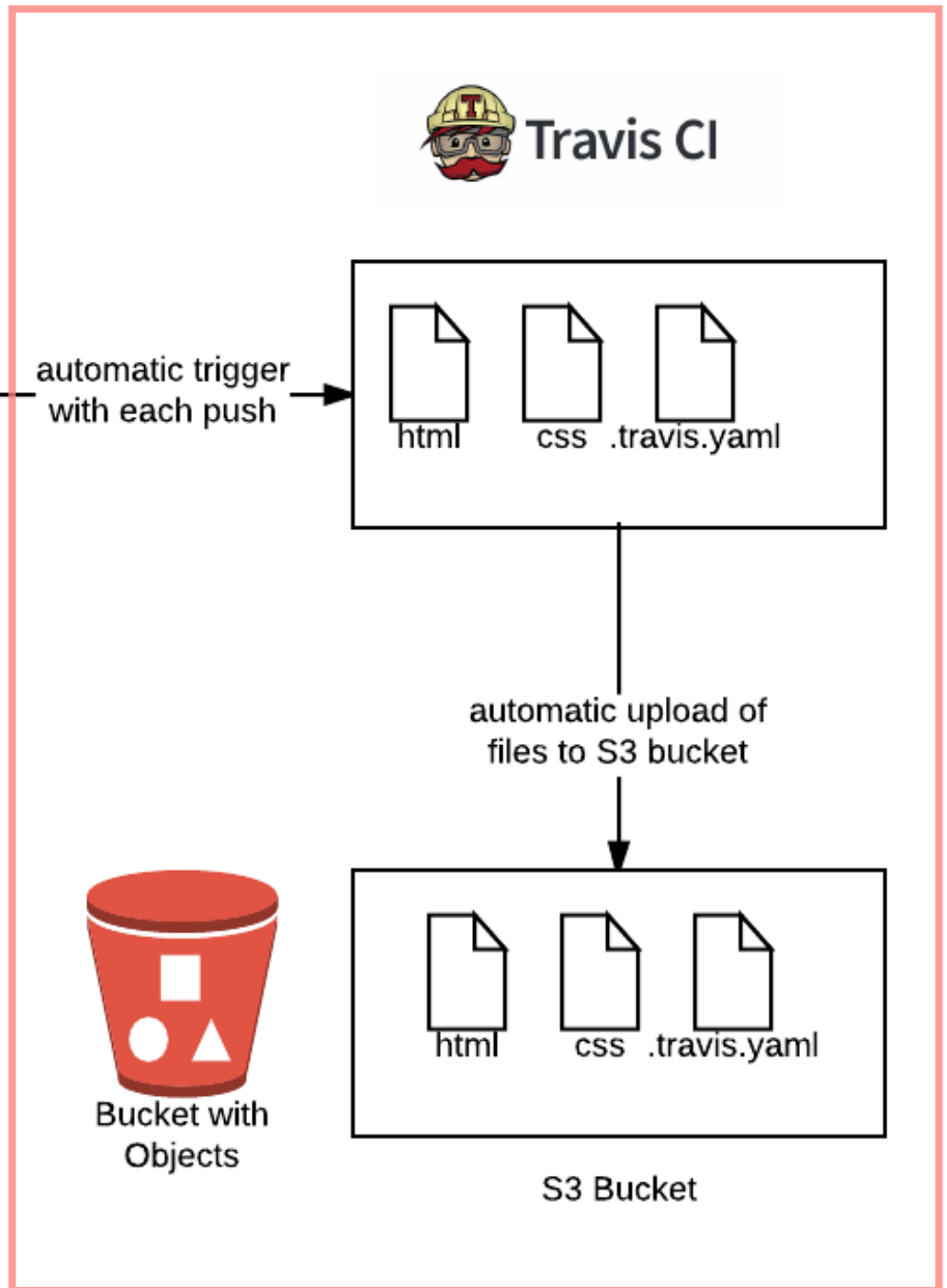
CI(Build, test)에 대해



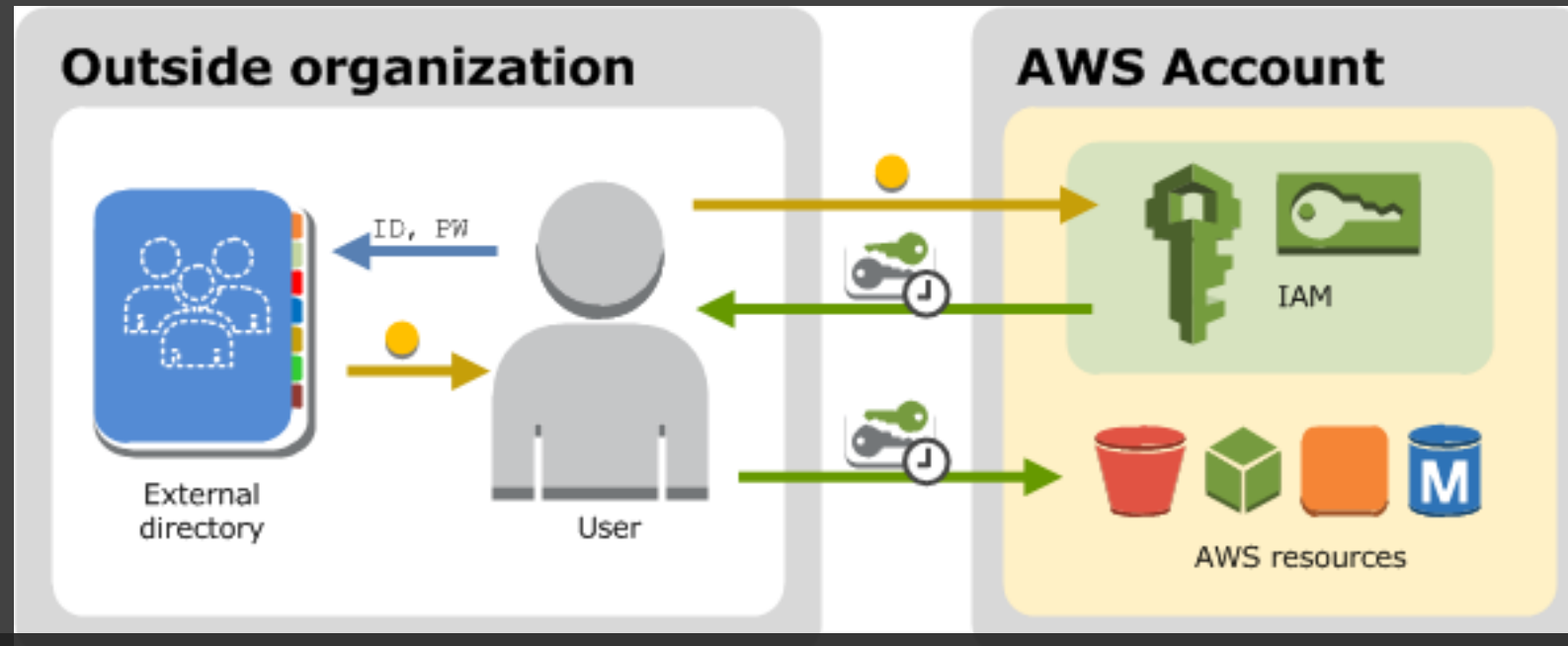
CI(Build, test)에 대해

빌드된 파일(war, etc...)을
S3(storage)에 저장

해당 파일을 읽고 쓰는 데 아무나 허용해도 될까?



AWS IAM



- AWS는 보통 회사 전체가 하나의 root 계정에 대해서 운영되기 때문에 root 계정을 모든 사람들에게 알려줘서 사용할 수 없다
- 각 사람과 server에 대해서 계정을 부여하고 각 계정에서는 필요한 권한만 가지고 있어야 안전하다
- AWS에서는 제공하는 기능, 기능 내 만들어진 리소스 등에 대해서 세분화하여 권한을 지정 할 수 있게 해준다
- 사람 뿐만 아니라 자동화하여 실행되는 서버에도 계정을 만들어서 관리하게 된다
- 서버가 사용하는 서비스에 대해서도 권한을 설정하여 이 서버가 사용하는 서비스는 어떤 리소스까지 조작할 수 있는지 지정할 수 있다