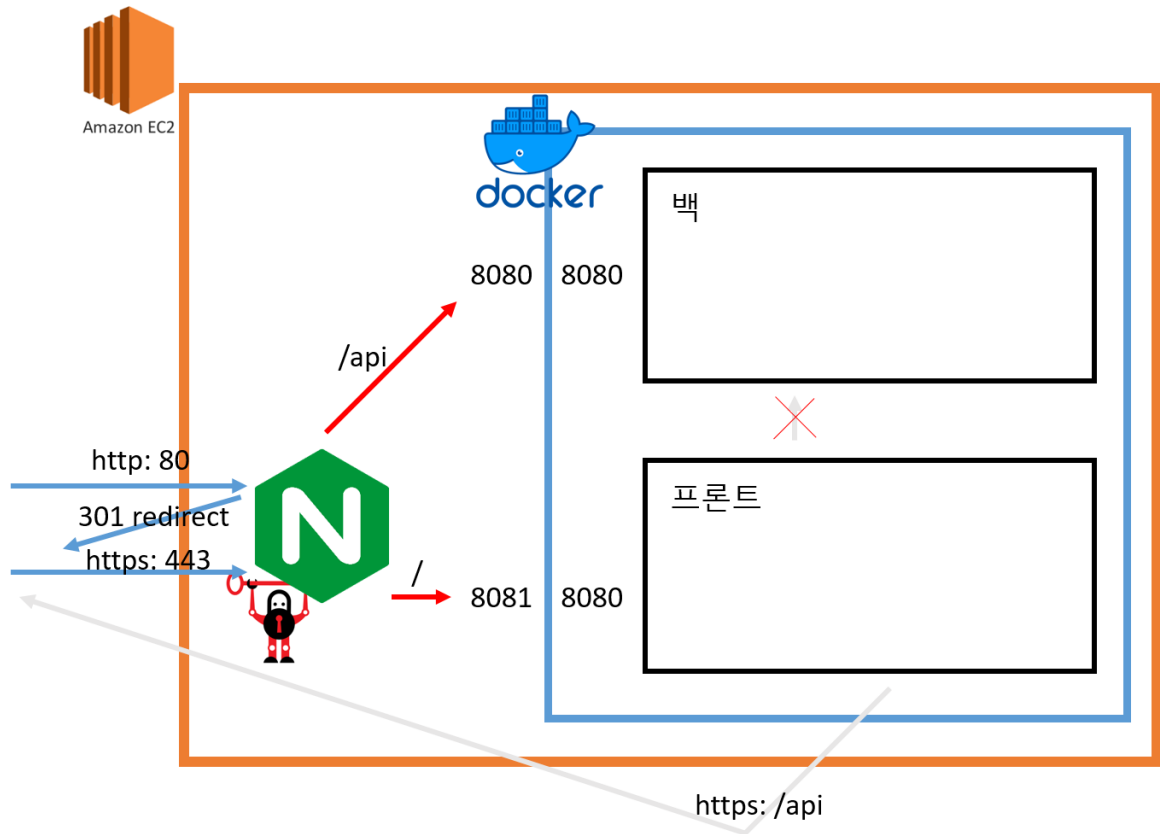


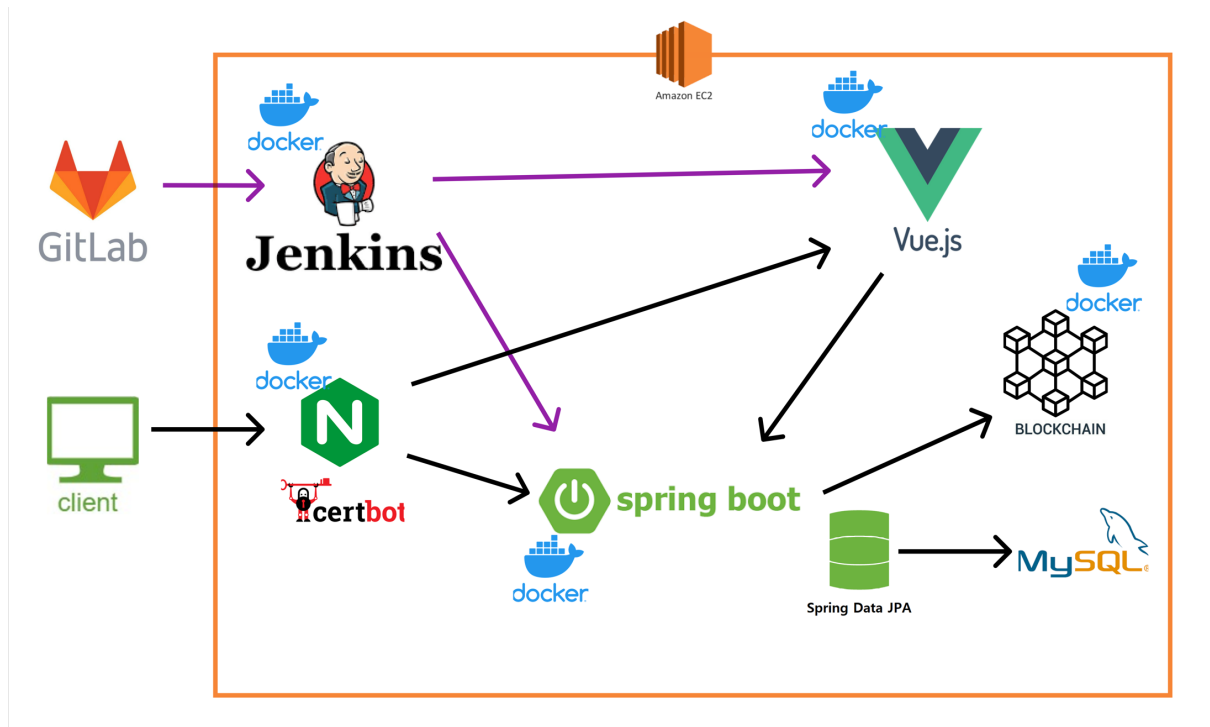
서버 세팅

설계

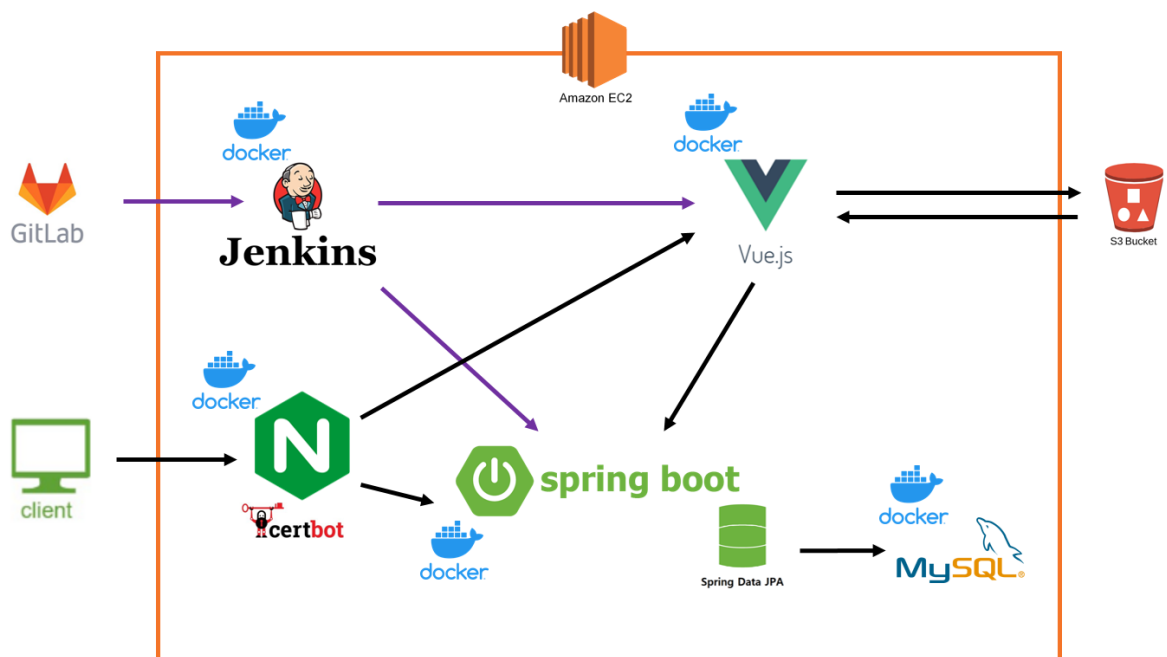
이전 프로젝트



초기 설계



최종 완성



이번 서버작업은 빠른 완성이 중요했기 때문에 이전 프로젝트에서 했던 작업을 많이 활용했습니다.

이전 프로젝트와의 가장 큰 차이점은 젠킨스를 활용해 CI/CD 환경을 구축했다는 점입니다.

초기 설계와 달리 MySQL을 Spring Docker에서 분리시켜 하나의 독립적인 공간으로 만들어 졌습니다. 또한 이미지 저장을 위한 S3 서버를 추가했습니다.

Docker

인터넷을 조금만 검색해보면 도커에 대해 쉽게 설명한 글이 많습니다. 그렇기 때문에 도커에 대한 자세한 설명은 생략하겠습니다.

여기서는 독립적인 작업환경을 만들기 위한 수단 정도로 이해해 주시면 됩니다. 더 간략히는 1컨테이너 == 1컴퓨터 라고 이해해도 무방할 거 같습니다.

작업을 이해하기 위해 필요한 컨테이너, 이미지, 도커파일 에 대해서도 간략히 설명해 보겠습니다.

- 이미지: 'exe' 느낌의 설치 파일
- 컨테이너: 실제 실행 파일
- 도커파일: 이미지를 생성하기 위한 파일
 - '이런이런거 넣어서 이렇게 이미지를 만들어 주세요' 라는 느낌의 주문서
 - 도커파일의 이름은 다르게 변경하기보다 Dockerfile 이라고 이름짓고 그대로 사용하는 게 편리합니다. (아무런 확장자도 없어야 하며 D는 대문자여야 합니다)

ec2 환경에 도커 설치하기

우분투에서 도커 설치 가이드: <https://roseline124.github.io/kuberdocker/2019/07/17/docker-study02.html>

```
sudo apt-get update

sudo apt install apt-transport-https

sudo apt install ca-certificates

sudo apt install curl

sudo apt install software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"

sudo apt update

apt-cache policy docker-ce

sudo apt install docker-ce

sudo chmod 666 /var/run/docker.sock
```

저는 도커로 프론트엔드(Vue), 백엔드(Spring), Nginx, Jenkins, Mysql 가 동작할 수 있는 환경을 각각 만들어 졌습니다. 각각의 환경은 다음 이미지를 기반으로 만들어 졌습니다.

- Mysql → mysql:8.0
- Jenkins → jenkins/jenkins:lts
- Nginx → ubuntu:20.04
- Frontend → node:16.16 기반의 도커파일
- Backend → basic_setting이라는 내가 만든 이미지 기반의 도커파일
 - basic_setting → ubuntu:20.04에 java, gradle, git 을 설치한 이미지 파일

실습에 필요한 주요 명령어는 다음과 같습니다.

`docker ps` 실행중인 컨테이너 확인

`docker ps -a` 모든 컨테이너 확인

`docker images` 모든 이미지 정보 확인

```
docker run -d -it --name <컨테이너 이름> -p 로컬PC포트:도커_포트 이미지_이름:태그 컨테이너 생성
docker commit -a "작성자" -m "커밋_메시지" 이미지화_할_컨테이너_이름 이미지_이름:태그 이미지 생성
docker stop <컨테이너 이름> 컨테이너 멈추기
docker start <컨테이너 이름> 컨테이너 실행하기
docker exec -it <컨테이너_이름> bash 컨테이너 들어가기
service mysql(nginx) start mysql(nginx) 실행
curl ident.me 내 ip주소 확인
docker logs 컨테이너 이름 해당 컨테이너가 어떤 이유로 실행되지 않는지 확인 가능
```

⚠ 실습 중간중간 `docker images` 명령어로 이미지가 생성됐는지, `docker ps`, `docker ps -a` 명령어로 컨테이너가 생성됐는지, 생성되고 실행 중인지 아니면 실행이 안되고 있는지를 계속 확인하는 걸 **매우** 권장합니다.

이로써 기본적인 준비는 끝났습니다. 자세한 실행방법 및 설명은 아래에서 개별로 진행하겠습니다.

Mysql

백엔드 컨테이너는 젠킨스에 의해 매번 새롭게 Build됩니다. 이러한 백엔드 컨테이너 내부에 Mysql이 존재한다면 Build마다 DB가 초기화 될 겁니다. 그렇기 때문에 DB서버는 백엔드 서버와 분리된 공간에 존재해야 합니다.

`docker pull mysql:8.0` 명령어로 mysql 이미지를 다운받습니다.

`docker run -itd -p 3306:3306 -e MYSQL_ROOT_PASSWORD=<비밀번호> --name mysql mysql:8.0` 명령어로 mysql 컨테이너를 생성합니다.

`docker exec -it mysql bash` 명령어로 mysql 컨테이너 내부에 들어옵니다.

`mysql -u root -p` + 엔터 입력 후 비밀번호를 입력하면 mysql에 접속할 수 있습니다.

이후 `create database dangdang`, `use dangdang` 명령어를 입력하고 나옵니다.

Nginx + Certbot

<https://velog.io/@damiano1027/Nginx-Nginx와-SpringBoot-내장-Tomcat-연동>

<https://velog.io/@jihyunhillpark/2.-spring-boot-기반-웹-배포-Cerbot-인증서-발급과-SSL-적용>

Nginx역시 Mysql과 Jenkins처럼 자체적인 이미지를 제공합니다. 저는 Nginx이미지로 작업하면서 certbot관련 부분에서 어려움을 겪었고, ubuntu이미지에 nginx를 설치하는 방식으로 우회했습니다.

`docker pull ubuntu:20.04` 명령어로 ubuntu 이미지를 다운받습니다.

`docker run -itd --name nginx -p 80:80 -p 443:443 ubuntu:20.04` 명령어로 nginx 컨테이너를 생성합니다.

`docker exec -it nginx bash` 명령어로 nginx 컨테이너 내부에 들어옵니다.

아래 명령어를 실행합니다.

```
apt-get update # apt로 다운받을 수 있는 내용 최신화

apt-get install -y vim # vi 명령어로 파일 편집

apt-get install -y nginx # nginx 설치

apt-get install -y python3-certbot-nginx # certbot 설치

certbot certonly --nginx -d j7a306.p.ssafy.io # certbot 인증서 다운
```

`/etc/letsencrypt/live/j7a306.p.ssafy.io` 위치에 pem키가 4개 추가됐다면 성공적으로 certbot 인증서가 다운받아졌다는 의미입니다.

nginx 설정을 위한 conf파일을 생성합니다.

- `cd /etc/nginx/sites-available`
- `vi test.conf`

```
server{ // 80포트로 들어오는 요청을 https로 던져주기
    listen 80;
    server_name j7a306.p.ssafy.io;
    return 301 https://j7a306.p.ssafy.io$request_uri;
}

server{ // 443요청 처리하기
    listen 443 ssl http2;
    server_name j7a306.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/j7a306.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j7a306.p.ssafy.io/privkey.pem;

    location /{
        proxy_pass http://j7a306.p.ssafy.io:8081;

        # WebSocket connection to failed 에러 대응
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        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_set_header Host $http_host;
    }

    location /api {
        proxy_pass http://j7a306.p.ssafy.io:8080;
    }
}
```

위 내용을 아주 간략히 살펴보면 다음과 같습니다.

- http + 80번 포트로 들어오는 모든 요청은 https로 redirect합니다.
 - http의 기본 포트가 80번이기 때문에 일반적으로 `j7a306.p.ssafy.io` 만 입력했을 때 해당 redirect가 실행됩니다.
- https + /api가 붙은 모든 요청을 백엔드(`http://j7a306.p.ssafy.io:8080`)로 전달합니다.
- 그 외 일반적인 https 요청은 모두 프론트엔드(`http://j7a306.p.ssafy.io:8081`)로 전달합니다.

심볼릭 링크를 설정해 줍니다.

- `ln -s /etc/nginx/sites-available/test.conf /etc/nginx/sites-enabled`
- `site-enabled` 로 이동해 `rm default` 명령어로 default 파일 삭제
- nginx 시작 혹은 재시작 (`service nginx start` or `service nginx restart`)

프론트엔드와 백엔드

frontend컨테이너와 backend컨테이너는 dockerfile을 통해 생성 됩니다.

이는 젠킨스에 의해 자동으로 실행되기 때문에 여기서 다루지 않고 젠킨스 부분에서 다루겠습니다.

backend컨테이너는 ubuntu:20.04이미지를 활용해 제가 임의로 만든 basic_setting이라는 이미지를 활용합니다.

basic_setting 이미지는 다음과 같은 명령어로 만들 수 있습니다.

```
docker pull ubuntu:20.04

docker run -itd --name basic_setting -p 8080:8080 ubuntu:20.04

docker exec -it basic_setting bash

apt-get update

apt-get install -y sudo

apt-get install -y vim

apt-get install -y openjdk-8-jdk

apt-get install -y git
```

아래 명령어는 gradle을 설치하기 위한 명령어 입니다. `apt-get install -y gradle` 을 통해 생성된 gradle은 4.4.1버전입니다. 우리 프로젝트에서 사용하는 7.5버전의 gradle을 수동으로 설치하고, 이를 환경변수로 설정하는 과정을 담은 명령어 입니다.

```
apt-get install -y wget

apt-get install -y zip

wget https://services.gradle.org/distributions/gradle-7.5-bin.zip -P /tmp

sudo unzip -d /opt/gradle /tmp/gradle-7.5-bin.zip

sudo ln -s /opt/gradle/gradle-7.5 /opt/gradle/latest

vi ~/.bashrc

export GRADLE_HOME=/opt/gradle/latest

export PATH=${GRADLE_HOME}/bin:${PATH}

source ~/.bashrc

gradle -v
```

Jenkins

https://github.com/hjs101/CICD_manual#docker-설치

<https://crispyblog.kr/development/common/10>

`docker pull jenkins/jenkins:lts` 명령어로 젠킨스 이미지를 다운받습니다.

`docker run -d -it --name jenkins -p 8082:8080 -v /jenkins:/var/jenkins_home -v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock -u root jenkins/jenkins:lts` 명령어로 젠킨스 컨테이너를 생성합니다.

- 명령어가 복잡한 이유는 젠킨스 컨테이너 내부에서 도커 명령어를 사용해야 하기 때문입니다.

가독성을 위해 줄을 나눠봤습니다.

-p 8082:8080 : 서버의 8082번 포트와 서비스 내부의 8080번 포트를 연결합니다.

`-v /usr/bin/docker:/usr/bin/docker` : 젠킨스 컨테이너에서도 호스트 서버의 도커를 사용하기 위한 바인딩입니다.(DooD)

`-v /var/run/docker.sock:/var/run/docker.sock` :젠킨스 컨테이너에서도 호스트 서버의 도커를 사용하기 위한 바인딩입니다.(DooD)

-u root: 젠킨스에 접속할 유저 계정입니다. 저는 root로 설정했습니다.

jenkins/jenkins:its : jenkins/jenkins라는 이미지의 its버전으로 컨테이너를 생성합니다.

1. localhost:8082번으로 접속을 시도하면 아래와 같은 화면이 나옵니다.

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:

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

2. 키 값을 확인하기 위해 cmd 에서 docker logs jenkins 를 입력합니다.

3. 중간에 보이는 9a6643335b54444bacf5ef8d37920d2c 를 복사해 브라우저의 Administrator password 에 입력합니다.

7

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

5. 설치가 완료되면 계정을 생성하고 **Save and Continue** 를 누릅니다.

Getting Started

Create First Admin User

계정명:

암호:

암호 확인:

이름:

이메일 주소:

Jenkins 2.361.1

[Skip and continue as admin](#)

[Save and Continue](#)

6. 다음과 같은 화면이 나올 겁니다. **Save and Finish** 를 누릅니다.

Instance Configuration

Jenkins URL:

http://localhost:8082/

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.361.1

Not now

Save and Finish

환경 설정

1. 플러그인 설치

gitlab, docker, ssh를 검색해 각각 아래 플러그인을 설치합니다.

업데이트된 플러그인 목록

설치 가능

설치된 플러그인 목록

고급

Q gitlab

Install Name ↓

Released

GitLab 1.5.35

Build Triggers

This plugin allows [GitLab](#) to trigger Jenkins builds and display their results in the GitLab UI.



2 mo 9 days ago

This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information.

Generic Webhook Trigger 1.84

notification

github

webhook

Build Parameters

gitlab

Build Triggers

bitbucket

bitbucket-server

jira



4 mo 20 days ago

Can receive any HTTP request, extract any values from JSON or XML and trigger a job with those values available as variables. Works with GitHub, GitLab, Bitbucket, Jira and many more.

Gitlab API 5.0.1-78.v47a_45b_9f78b_7



Library plugins (for use by other plugins)

1 mo 13 days ago

This plugin provides [GitLab API](#) for other plugins.

GitLab Authentication 1.16

Authentication and User Management



This is the an authentication plugin using gitlab OAuth.

4 mo 20 days ago

This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information.

Install without restart

Download now and install after restart

Update information obtained: 21 min ago

지금 확인

업데이트된 플러그인 목록

설치 가능

설치된 플러그인 목록

고급

Q docker

Install	Name ↓	Released
<input checked="" type="checkbox"/>	<u>Docker</u> 1.2.9 Cloud Providers Cluster Management docker This plugin integrates Jenkins with Docker <div>This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.</div>	4 mo 12 days ago
<input checked="" type="checkbox"/>	<u>Docker Commons</u> 1.21 Library plugins (for use by other plugins) docker Provides the common shared functionality for various Docker-related plugins.	11 days ago
<input checked="" type="checkbox"/>	<u>Docker Pipeline</u> 521.v1a_a_dd2073b_2e pipeline DevOps Deployment docker Build and use Docker containers from pipelines.	28 days ago
<input checked="" type="checkbox"/>	<u>Docker API</u> 3.2.13-37.vf3411c9828b9 Library plugins (for use by other plugins) docker This plugin provides docker-java API for other plugins. <div>This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.</div>	4 mo 20 days ago
	docker-build-step 2.8	

Install without restart

Download now and install after restart

Update information obtained: 25 min ago

지금 확인

Plugin Manager

업데이트된 플러그인 목록

설치 가능

설치된 플러그인 목록

고급

Q SSH

Install	Name ↓	Released
<input type="checkbox"/>	SSH 2.6.1 Build Wrappers This plugin executes shell commands remotely using SSH protocol. <div>Warning: This plugin version may not be safe to use. Please review the following security notices:<ul style="list-style-type: none">CSRF vulnerability and missing permission checks allow capturing credentialsMissing permission check allows enumerating credentials IDs</div>	4 yr 4 mo ago
<input checked="" type="checkbox"/>	Publish Over SSH 1.24 Artifact Uploaders Build Tools Send build artifacts over SSH	6 mo 16 days ago
<input type="checkbox"/>	SSH Agent 295.v9ca_a_1c7cc3a_a_ This plugin allows you to provide SSH credentials to builds via a ssh-agent in Jenkins.	3 mo 28 days ago
	SSH Pipeline Steps 2.0.39.v831c5e6468b_c pipeline	

2. credential 설정

젠킨스 관리 > 시스템 설정 > Gitlab

configure system → 해당 칸에 <https://lab.ssafy.com> 만 적어야 함

Gitlab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name

A name for the connection

qwerty1434@naver.com

Gitlab host URL

The complete URL to the Gitlab server (e.g. http://gitlab.mydomain.com)

https://lab.ssafy.com/

Credentials

API Token for accessing Gitlab

gitlab_token

+ Add

고급...

Success

Test Connection

- Credentials 추가하는 법

Kind

Secret text

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Secret

ID ?

Description ?

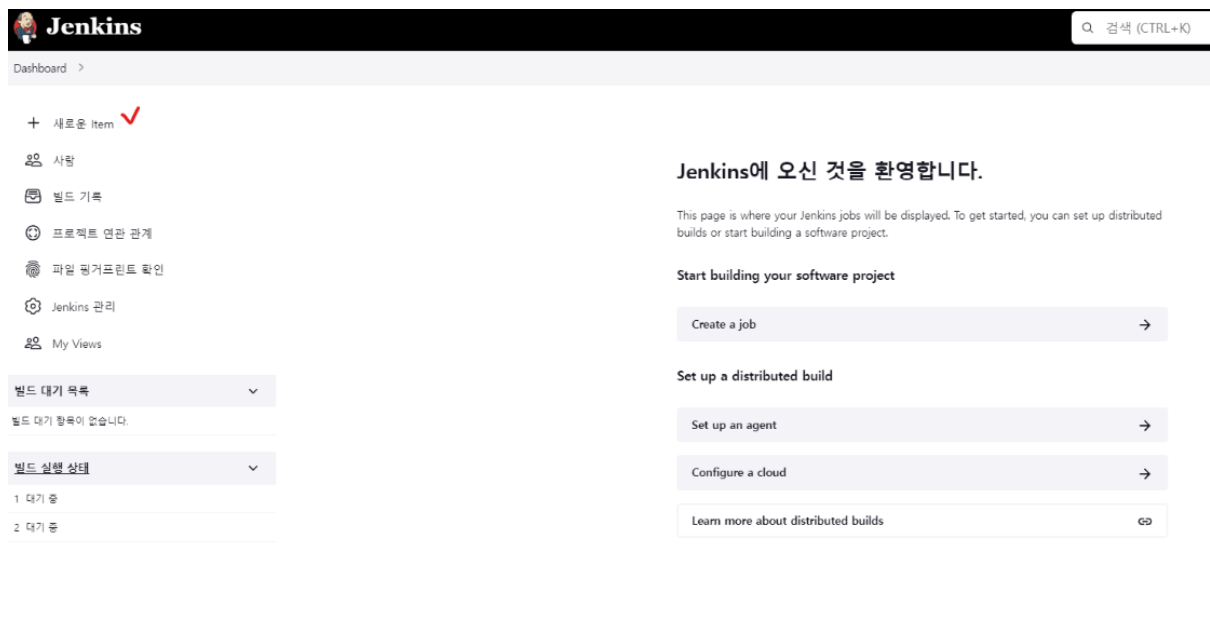
Add

Cancel

- secret text를 선택하고 Secret에 깃랩 계정의 access token 을 넣으면 됨

3. job생성

1. 새로운 Item 클릭



2. 파이프라인 선택

Enter an item name

» Required field

Freestyle project

이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다

Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing

Multi-configuration project

다양한 환경에서의 테스트, 플랫폼 특성 빌드, 기타 동등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.

Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate name long as they are in different folders.

Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.

Organization Folder

Creates a set of multibranch project subfolders by scanning for repositories.

3. 체크, 체크

☒ Do not allow the pipeline to resume if the controller restarts ✓

☐ GitHub project

GitLab Connection

conn1

☐ Use alternative credential

☐ Pipeline speed/durability override ?

☐ Preserve stashes from completed builds ?

☐ Throttle builds ?

☐ 오래된 빌드 삭제 ?

☐ 이 빌드는 매개변수가 있습니다 ?

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: http://j7a306.p.ssafy.io:8082/project/frontend ? ✓

4. 고급에서 Secret token값 확인 → 웹훅에서 사용됨

Secret token ?

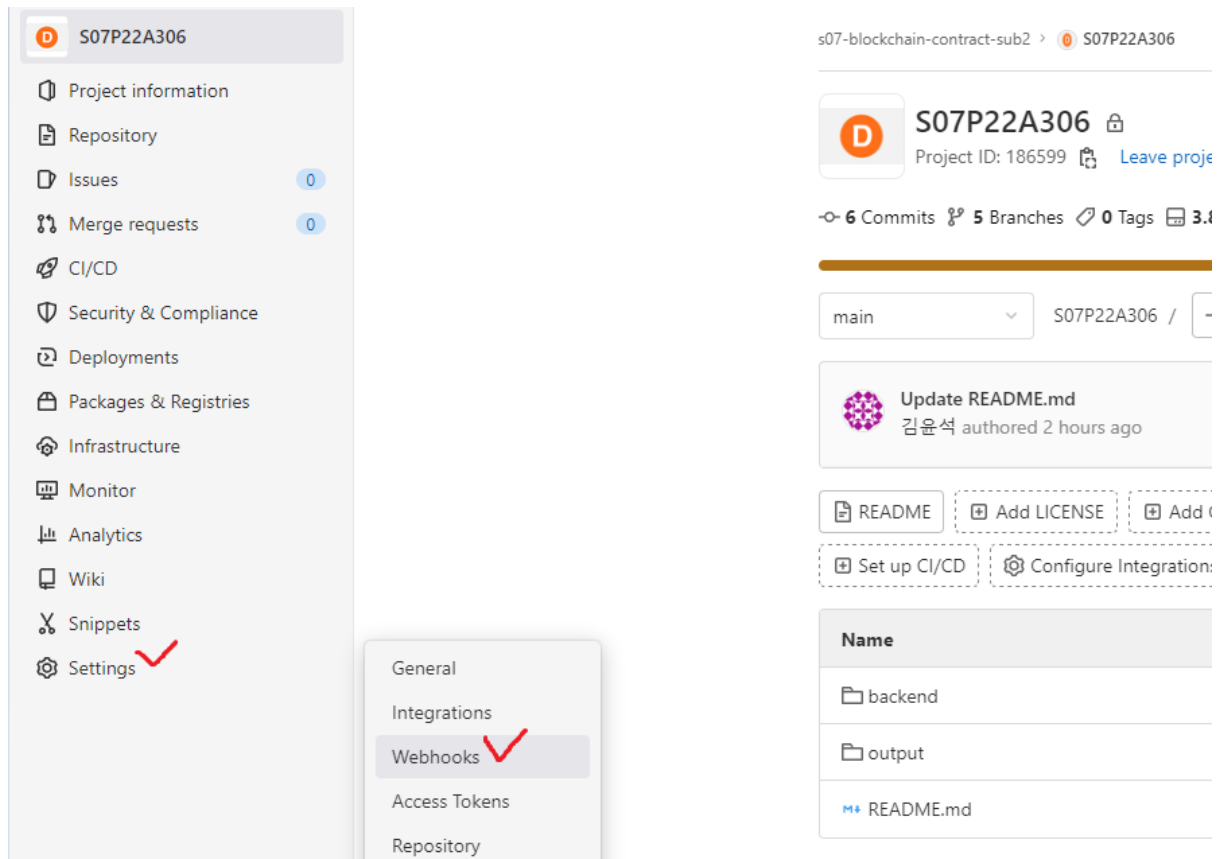
ee02527881a89e3dec886183aa102844

Generate

Clear

4. 깃랩 프로젝트에서 웹훅 설정하기

1. 프로젝트 > settings > Webhooks



2. URL에는 `젠킨스 주소/project/젠킨스 job이름`, Push events에 front, Merge request events 체크

URL

✓

URL must be percent-encoded if it contains one or more special characters.

Secret token

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token HTTP` header.

Trigger

☒ Push events

✓

Push to the repository.

☐ Tag push events
A new tag is pushed to the repository.

☐ Comments
A comment is added to an issue or merge request.

☐ Confidential comments
A comment is added to a confidential issue.

☐ Issues events
An issue is created, updated, closed, or reopened.

☐ Confidential issues events
A confidential issue is created, updated, closed, or reopened.

☒ Merge request events ✓
A merge request is created, updated, or merged.

- Merge request events 는 취소했음 → merge할 때 push만큼의 웹훅도 같이 들어와서

✓ #30	2022. 9. 19. 오전 2:06
Started by GitLab push by 김현지	
✓ #29	2022. 9. 19. 오전 2:06
Triggered by GitLab Merge Request #1:...	
✓ #28	2022. 9. 19. 오전 2:06
Triggered by GitLab Merge Request #1:...	

3. jenkins에서 Secret token 가져와 넣기

→ job생성의 '고급에서 Secret token값 확인' 단계의 토큰을 말합니다.

값 세팅

저는 jenkins컨테이너 안에 dockerfile폴더를 만들었습니다.

dockerfile폴더 안에는 다시 frontend, backend폴더가 있고 그 안에서 각각의 Dockerfile을 보관하고 있습니다.

Frontend의 Dockerfile

```
FROM node:16.16
WORKDIR dockerfile/frontend
```

```

RUN git clone -b front --single-branch https://<깃랩아이디>:<깃랩access토큰>@lab.ssafy.com/s07-blockchain-contract-sub2/S07P22A306.git
WORKDIR S07P22A306/frontend

RUN npm install
RUN npm run lint -- --fix

EXPOSE 8081

CMD ["npm", "run", "serve"]

```

Frontend의 Pipeline

```

pipeline {
    agent any

    stages {
        stage('Deploy') {
            steps{
                script{
                    try{
                        sh 'docker stop frontend'
                        sh 'docker rm frontend'
                    }catch(error){
                    }
                    try{
                        sh 'docker rmi frontend:1.0'
                    }catch(error){
                    }
                }
            }
        }

        stage('Dockerfile'){
            steps {
                script{
                    try{
                        sh 'docker build --no-cache -t frontend:1.0 /dockerfile/frontend'
                    }catch(error){
                    }
                }
            }
        }

        stage('Build'){
            steps{
                script{
                    try{
                        sh 'docker run -d -it -p 8080:8080 --name frontend frontend:1.0'
                    }catch(error){
                    }
                }
            }
        }
    }
}

```

Backend의 Dockerfile

```

FROM basic_setting:0.0
WORKDIR /

RUN git clone -b back --single-branch https://<깃랩아이디>:<깃랩access토큰>@lab.ssafy.com/s07-blockchain-contract-sub2/S07P22A306.git
WORKDIR S07P22A306/backend

# jenkins컨테이너에 있는 파일을 backend 컨테이너로 옮기는 명령어
ADD application-local.properties /S07P22A306/backend/src/main/resources
ADD application-personal.properties /S07P22A306/backend/src/main/resources

RUN chmod +x ./gradlew
RUN ./gradlew build
WORKDIR build/libs

```

```
EXPOSE 8080
CMD ["java", "-jar", "dangdang-0.0.1-SNAPSHOT.jar"]
```

- application.properties는 아이디 비밀번호 등의 중요한 정보를 담고 있어 .gitignore로 gitlab에 올리지 않은 파일들입니다. 이들은 vi 명령어로 직접 생성해 줬으며 ADD 명령어를 편리하게 실행하기 위해 Dockerfile과 같은 위치에서 보관하고 있습니다.

Backend의 Pipeline

```
pipeline {
  agent any
  stages {
    stage('Deploy') {
      steps{
        script{
          try{
            sh 'docker stop backend'
            sh 'docker rm backend'
          }catch(error){
            }
          try{
            sh 'docker rmi backend:1.0'
          }catch(error){
            }
        }
      }
    }

    stage('Dockerfile'){
      steps {
        script{
          try{
            sh 'docker build --no-cache -t backend:1.0 /dockerfile/backend'
          }catch(error){
            }
        }
      }
    }

    stage('Build'){
      steps{
        script{
          try{
            sh 'docker run -d -it -p 8080:8080 --name backend backend:1.0'
          }catch(error){
            }
        }
      }
    }
  }
}
```

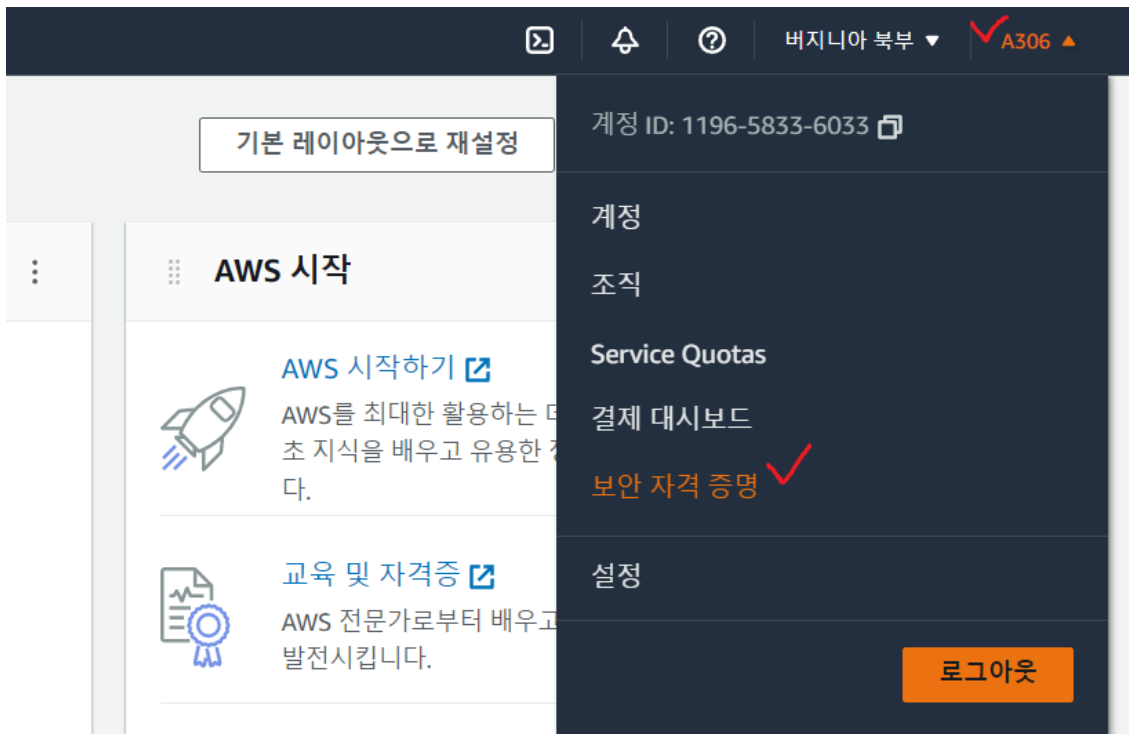
pipeline에 대한 간략한 설명입니다.

- Deploy단계에서는 기존에 존재하는 컨테이너 및 이미지를 삭제합니다.
- Dockerfile단계에서는 도커파일을 실행시켜 이미지를 생성합니다.
- Buil단계에서는 Dockerfile단계에서 생성된 이미지로 컨테이너를 실행합니다.

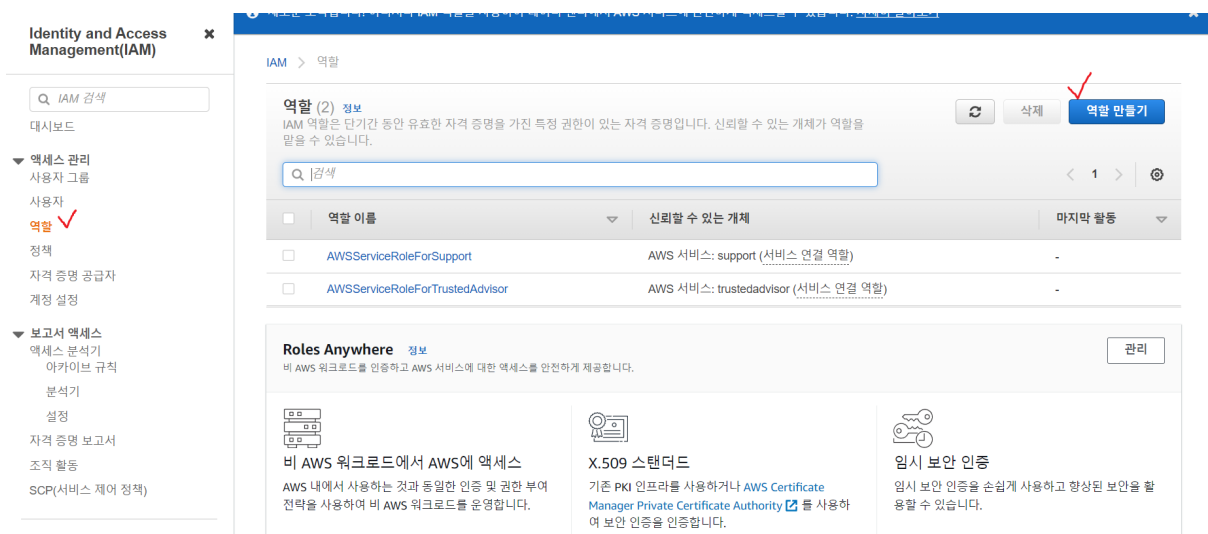
S3

<https://merrily-code.tistory.com/142>

1. aws 루트계정 회원가입 및 로그인
 2. IAM계정 생성
- 루트계정 로그인 후 **보안 자격 증명** 클릭



- 역할 > 역할 만들기 클릭



- AWS 서비스 > S3 검색 후 선택 > 다음

☒ **AWS 서비스**
EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.

☐ **AWS 계정**
사용자 또는 서드 파티에 속한 다른 AWS 계정의 엔터티가 이 계정에서 작업을 수행하도록 허용합니다.

☐ **웹 자격 증명**
지정된 외부 웹 자격 증명 공급자와 연동된 사용자가 이 역할을 맡아 이 계정에서 작업을 수행하도록 허용합니다.

☐ **SAML 2.0 연동**
기업 디렉터리에서 SAML 2.0과 연동된 사용자가 이 계정에서 작업을 수행할 수 있도록 허용합니다.

☐ **사용자 지정 신뢰 정책**
다른 사용자가 이 계정에서 작업을 수행할 수 있도록 사용자 지정 신뢰 정책을 생성합니다.

사용 사례

EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.

일반 사용 사례

☐ **EC2**
Allows EC2 instances to call AWS services on your behalf.

☐ **Lambda**
Allows Lambda functions to call AWS services on your behalf.

다른 AWS 서비스의 사용 사례:

☒ **S3** ☒
Allows S3 to call AWS services on your behalf.

☐ **S3 Batch Operations**
Allows S3 Batch Operations to call AWS services on your behalf.

취소
다음

• AmazonS3FullAccess 선택 후 확인

IAM > 역할 > 역할 생성

1단계
신뢰할 수 있는 엔터티 선택

2단계
권한 추가

3단계
이름 지정, 검토 및 생성

권한 추가

권한 정책 (선택됨 1/767)
새 역할에 연결할 정책을 하나 이상 선택합니다.

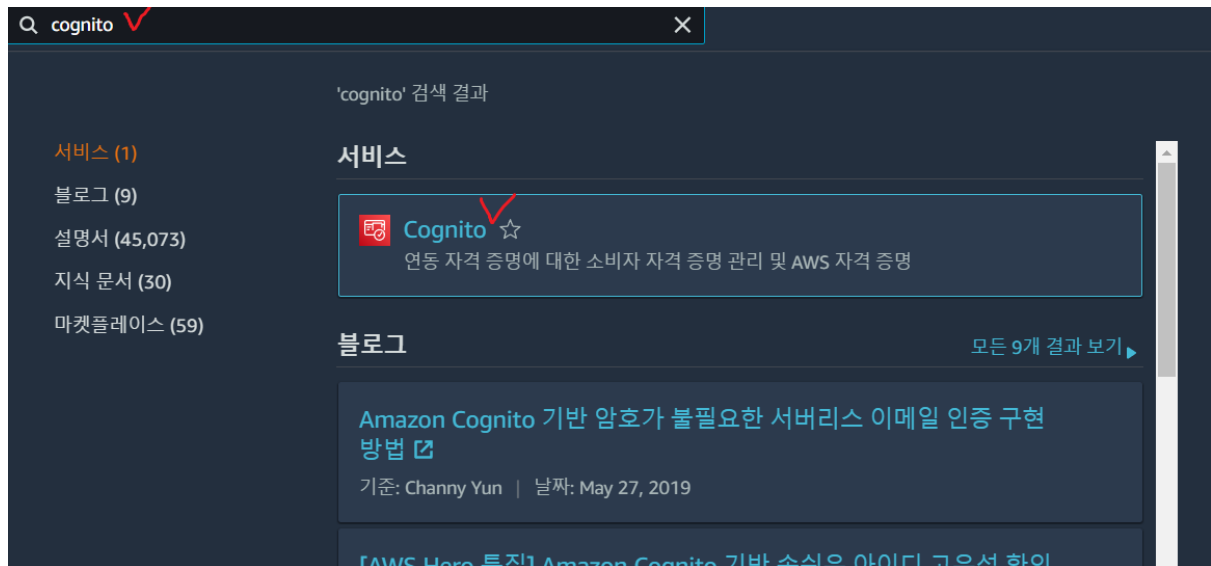
1 개 일치

<input checked="" type="checkbox"/>	정책 이름	유형	설명
<input checked="" type="checkbox"/>	AmazonS3FullAccess	AWS 관...	Provides full access to all buckets via the AWS Management Console.

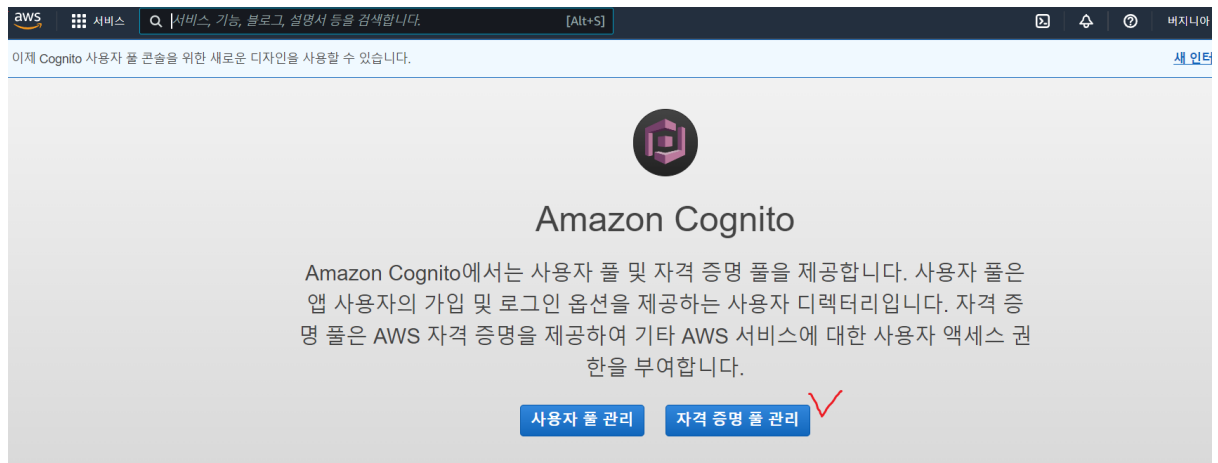
▶ 권한 경계 설정 - 선택 사항
권한 경계를 설정하여 이 역할이 가질 수 있는 최대 권한을 제한합니다. 이는 일반적인 설정은 아니지만 권한 관리를 다른 사용자에게 위임하는 데 사용할 수 있습니다.

취소
이전
다음

- 역할 이름 작성 후 역할 생성 클릭
- 3. cognito로 접근 권한 부여하기
- cognito검색해서 접근하기



- 자격 증명 풀 관리 선택



- 자격 증명 풀 이름 입력, 인증되지 않은 자격 증명에 대한 액세스 활성화 클릭, 풀 생성 클릭
 - 이때 오른쪽 위 지역이 global이 아닌 서울로 설정되어 있어야 함!!!!



- 세부 정보 보기 클릭 > 정책 문서 보기 > 편집

▼ 세부 정보 숨기기 ✓

역할 요약 ?

역할 설명 Your authenticated identities would like access to Cognito.

IAM 역할 새 IAM 역할 생성 ▼

역할 이름 Cognito_dangdangAuth_Role

▼ 정책 문서 숨기기 ✓

편집 ✓

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "mobileanalytics:PutEvents",
        "cognito-sync:*",
        "cognito-identity:*"
      ],
      "Resource": "*"
    }
  ]
}
```

- JSON을 아래 내용으로 교체

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["cognito-sync:*", "cognito-identity:*", "s3:*"],
      "Resource": ["*"]
    }
  ]
}
```

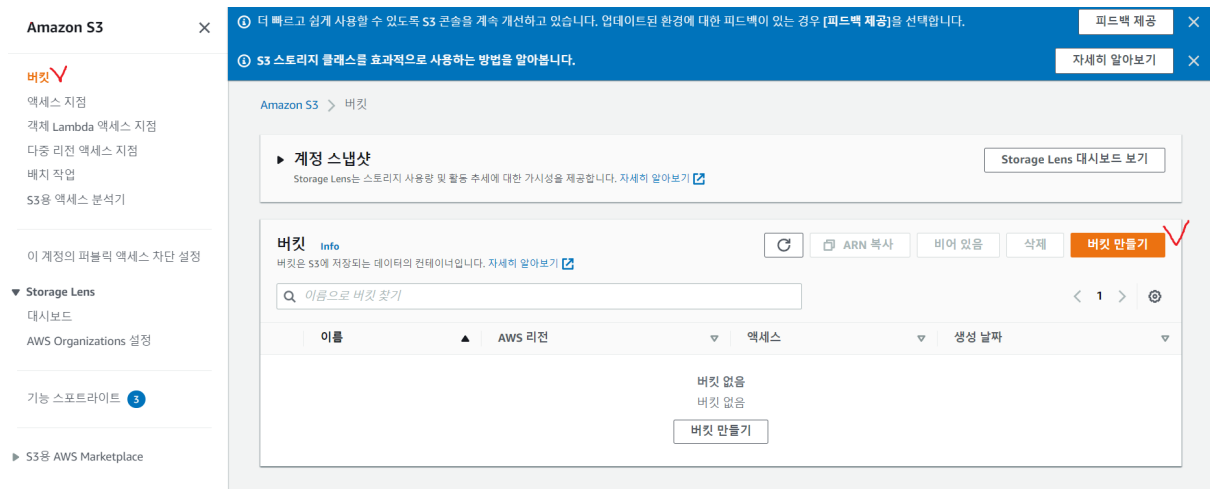
- 부여받은 자격증명 키 저장

▼ AWS 자격 증명 얻기

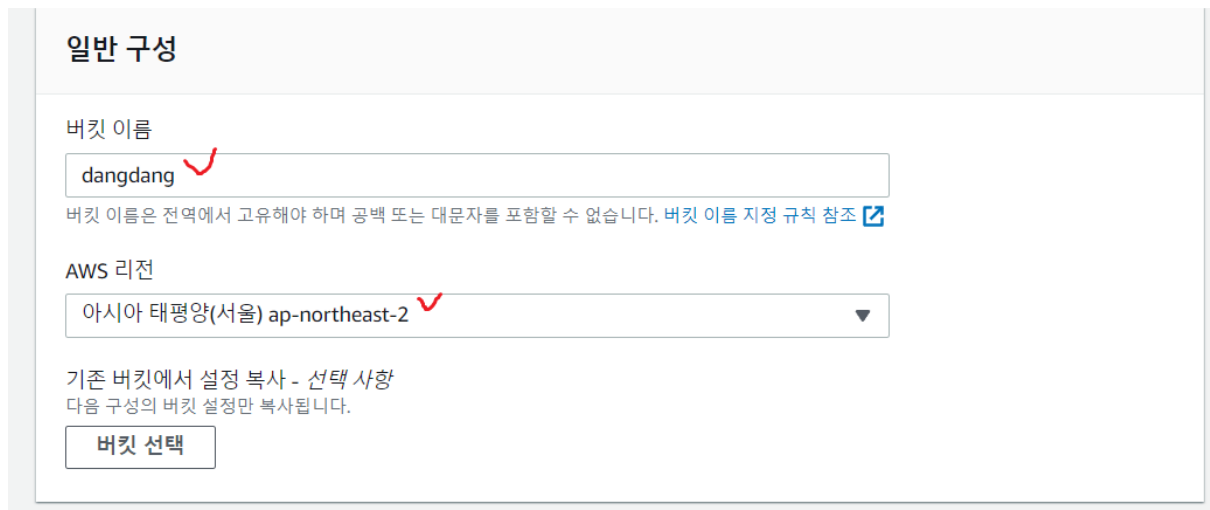
```
// Amazon Cognito 인증 공급자를 초기화합니다
AWS.config.region = 'ap-northeast-2'; // 리전
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: 'ap-northeast-2:81a948c5-f0c2-4e4b-ac0c-6ed0ffbce8b8',
});
```

4. 버킷 생성하기

- S3 검색 후 클릭
- 버킷 > 버킷 만들기



- 버킷 이름, AWS리전 설정 // (버킷 선택 누르는거 아님, dangdang이 있어서 dangdang-bucket으로 이름 변경함)



- 아래로 내리다보면 있는 **모든 퍼블릭 액세스 차단** 클릭해서 체크 해제

이 버킷의 퍼블릭 액세스 차단 설정

퍼블릭 액세스는 ACL(엑세스 제어 목록), 버킷 정책, 액세스 지정 정책 또는 모두를 통해 버킷 및 객체에 부여됩니다. 이 버킷 및 해당 객체에 대한 퍼블릭 액세스가 차단되었는지 확인하려면 모든 퍼블릭 액세스 차단을 활성화합니다. 이 설정은 이 버킷 및 해당 액세스 지정에만 적용됩니다. AWS에서는 모든 퍼블릭 액세스 차단을 활성화하도록 권장하지만, 이 설정을 적용하기 전에 퍼블릭 액세스가 없어도 애플리케이션이 올바르게 작동하는지 확인합니다. 이 버킷 또는 내부 객체에 대한 어느 정도 수준의 퍼블릭 액세스가 필요한 경우 특정 스토리지 사용 사례에 맞게 아래 개별 설정을 사용자 지정할 수 있습니다. [자세히 알아보기](#)

- ☒ **모든 퍼블릭 액세스 차단**
이 설정을 활성화하면 아래 4개의 설정을 모두 활성화한 것과 같습니다. 다음 설정 각각은 서로 독립적입니다.
- ☐ **새 ACL(엑세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단**
S3은 새로 추가된 버킷 또는 객체에 적용되는 퍼블릭 액세스 권한을 차단하며, 기존 버킷 및 객체에 대한 새 퍼블릭 액세스 ACL 생성을 금지합니다. 이 설정은 ACL을 사용하여 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 권한을 변경하지 않습니다.
 - ☐ **임의의 ACL(엑세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단**
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 모든 ACL을 무시합니다.
 - ☐ **새 퍼블릭 버킷 또는 액세스 지정 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단**
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 새 버킷 및 액세스 지정 정책을 차단합니다. 이 설정은 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 정책을 변경하지 않습니다.
 - ☐ **임의의 퍼블릭 버킷 또는 액세스 지정 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 및 교차 계정 액세스 차단**
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 정책을 사용하는 버킷 또는 액세스 지정에 대한 퍼블릭 및 교차 계정 액세스를 무시합니다.



모든 퍼블릭 액세스 차단을 비활성화하면 이 버킷과 그 안에 포함된 객체가 퍼블릭 상태가 될 수 있습니다.
정적 웹 사이트 호스팅과 같은 구체적으로 확인된 사용 사례에서 퍼블릭 액세스가 필요한 경우가 아니면 모든 퍼블릭 액세스 차단을 활성화하는 것이 좋습니다.

5. 버킷 권한 부여

- 버킷 선택 후 권한 클릭

dangdang-bucket Info

객체 | 속성 | 권한 | 지표 | 관리 | 액세스 지정

권한 개요

엑세스
객체를 퍼블릭으로 설정할 수 있음

퍼블릭 액세스 차단(버킷 설정)

퍼블릭 액세스는 ACL(엑세스 제어 목록), 버킷 정책, 액세스 지정 정책 또는 모두를 통해 버킷 및 객체에 부여됩니다. 모든 S3 버킷 및 객체에 대한 퍼블릭 액세스가 차단되었는지 확인하려면 [모든 퍼블릭 액세스 차단]을 활성화합니다. 이 설정은 이 버킷 및 해당 액세스 지정에만 적용됩니다. AWS에서는 [모든 퍼블릭 액세스 차단]을 활성화하도록 권장하지만, 이 설정을 적용하기 전에 퍼블릭 액세스가 없어도 애플리케이션이 올바르게 작동하는지 확인합니다. 버킷 또는 내부 객체에 어느 정도 수준의 퍼블릭 액세스가 필요한 경우 특정 스토리지 사용 사례에 맞게 아래 개별 설정을 사용자 지정할 수 있습니다. [자세히 알아보기](#)

편집

모든 퍼블릭 액세스 차단

비활성

- 버킷 정책에서 편집을 누른 후 아래 JSON 붙여넣기

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:*"
      ]
    }
  ]
}
```

```

    ],
    "Resource": [
      "arn:aws:s3:::dangdang-bucket/*",
      "arn:aws:s3:::dangdang-bucket"
    ]
  }
]
}

```

- 스크롤 내려서 CORS에 아래 코드 붙여넣기

```

[
  {
    "AllowedOrigins": ["*"],
    "AllowedMethods": ["GET", "PUT", "POST", "HEAD"],
    "AllowedHeaders": ["*"],
    "ExposeHeaders": ["x-amz-server-side-encryption", "x-amz-request-id", "x-amz-id-2"],
    "MaxAgeSeconds": 3000
  }
]

```

6. 업로드 테스트

- 버킷 > dangdang-bucket 클릭

Amazon S3 > 버킷

계정 스냅샷
Storage Lens는 스토리지 사용량 및 활동 추세에 대한 가시성을 제공합니다. 자세히 알아보기

버킷 (1)
버킷은 S3에 저장되는 데이터의 컨테이너입니다. 자세히 알아보기

이름으로 버킷 찾기

이름	AWS 리전	액세스	생성 날짜
dangdang-bucket	아시아 태평양(서울) ap-northeast-2	퍼블릭	2022. 9. 22. am 10:25:23 AM KST

- 파일 업로드 하기
- 객체에서 해당 이미지를 클릭하고

Amazon S3 > 버킷 > dangdang-bucket

dangdang-bucket
퍼블릭 액세스 가능

객체 | 속성 | 권한 | 지표 | 관리 | 액세스 지점

객체 (1)
객체는 Amazon S3에 저장되어 있는 기본 엔티티입니다. Amazon S3 인벤토리를 사용하여 버킷에 있는 모든 객체의 목록을 얻을 수 있습니다. 다른 사용자가 객체에 액세스할 수 있게 하려면 명시적으로 권한을 부여해야 합니다. 자세히 알아보기


S3 URI 복사 | URL 복사 | 다운로드 | 열기 | 삭제 | 작업 | 폴더 만들기 | 업로드

집두사로 객체 찾기

이름	유형	마지막 수정	크기	스토리지 클래스
백준 쌤네일.png	png	2022. 9. 22. am 10:30:12 AM KST	44.8KB	Standard

- 객체 url 클릭 시 이미지가 잘 뜨면 성공

객체 개요

소유자 2f7df858d272aadb5ef512274f8e4c51e07ec1297a30f189c89d929d8aabbfe1a AWS 리전 아시아 태평양(서울) ap-northeast-2 마지막 수정 2022. 9. 22. am 10:30:12 AM KST 크기 44.8KB 유형 png 키 백준 썸네일.png	S3 URI s3://dangdang-bucket/백준 썸네일.png Amazon 리소스 이름(ARN) arn:aws:s3:::dangdang-bucket/백준 썸네일.png 엔터티 태그(Etag) 843838bb250cba87fe4907fb9c33be6a 객체 URL  https://dangdang-bucket.s3.ap-northeast-2.amazonaws.com/%EB%B0%B1%EC%A4%80+%EC%8D%B8%EB%84%A4%EC%9D%BC.png
---	---

프론트에서 js에 s3 접속하기

vue로 s3이미지 업로드하기

<https://www.youtube.com/watch?v=CqMoofqWPdQ&t=55s>

1. `vue create 프로젝트이름` 로 vue파일을 생성합니다.
 - 간단한 예제를 만들거기 때문에 default세팅으로 시작하겠습니다.
 - `cd 프로젝트이름` 명령어를 입력한 뒤 `npm run serve` 가 정상작동하는지 확인합니다.
2. aws sdk npm을 설치합니다.
 - `npm install aws-sdk --save`
 - 이후 script부분에 `import AWS from aws-sdk` 를 입력해 aws sdk를 사용할 수 있습니다.
3. helloworld.vue 파일을 수정합니다.

ACLs관련 오류 발생

```

▶ PUT https://dangdang-bucket.s3.ap-northeast-2.amazonaws.com/docker.png 400 (Bad Request) xhr.js?b00a:81
AccessControlListNotSupported: The bucket does not allow ACLs
    at Request.extractError (webpack-internal:///./node_modules/aws-sdk/lib/services/s3.js:727:35)
    at Request.callListeners (webpack-internal:///./node_modules/aws-sdk/lib/sequential_executor.js:117:20)
    at Request.emit (webpack-internal:///./node_modules/aws-sdk/lib/sequential_executor.js:84:10)
    at Request.emit (webpack-internal:///./node_modules/aws-sdk/lib/request.js:704:14)
    at Request.transition (webpack-internal:///./node_modules/aws-sdk/lib/request.js:32:10)
    at AcceptorStateMachine.runTo (webpack-internal:///./node_modules/aws-sdk/lib/state_machine.js:16:12)
    at eval (webpack-internal:///./node_modules/aws-sdk/lib/state_machine.js:27:10)
    at Request.eval (webpack-internal:///./node_modules/aws-sdk/lib/request.js:48:9)
    at Request.eval (webpack-internal:///./node_modules/aws-sdk/lib/request.js:706:12)
    at Request.callListeners (webpack-internal:///./node_modules/aws-sdk/lib/sequential_executor.js:129:18)

```

s3접속 > 해당 버킷 클릭 > 권한 > ACL > 버킷 소유자 적용 클릭



이 버킷에는 객체 소유권에 대해 버킷 소유자 적용 설정이 적용되어 있습니다.
 버킷 소유자 적용이 적용된 경우 버킷 정책을 사용하여 액세스를 제어합니다. 자세히 알아보기

ACL 활성화됨 > ACL이 복원된다는 것을 확인합니다 > 변경 사항 저장

객체 소유권

다른 AWS 계정에서 이 버킷에 작성한 객체의 소유권 및 액세스 제어 목록(ACL)의 사용을 제어합니다. 객체 소유권은 객체에 대한 액세스를 지정할 수 있는 사용자를 결정합니다.

☐ ACL 비활성화됨(권장)

이 버킷의 모든 객체는 이 계정이 소유합니다. 이 버킷과 그 객체에 대한 액세스는 정책을 통해서만 지정됩니다.



ACL 활성화됨

이 버킷의 객체는 다른 AWS 계정에서 소유할 수 있습니다. 이 버킷 및 객체에 대한 액세스는 ACL을 사용하여 지정할 수 있습니다.



ACL을 활성화하면 버킷 소유자가 객체 소유권에 대해 적용한 설정이 비활성화됩니다.

버킷 소유자 적용 설정이 해제되면 ACL(액세스 제어 목록) 및 연결된 권한이 복원됩니다. 소유하지 않은 객체에 대한 액세스는 버킷 정책이 아닌 ACL을 기반으로 합니다.



☒ ACL이 복원된다는 것을 확인합니다.

객체 소유권

☒ 버킷 소유자 선호

이 버킷에 작성된 새 객체가 bucket-owner-full-control 삽입 ACL을 지정하는 경우 새 객체는 버킷 소유자가 소유합니다. 그렇지 않은 경우 객체 라이터가 소유합니다.

☐ 객체 라이터

객체 라이터는 객체 소유자로 유지됩니다.



새 객체에 대해서만 객체 소유권을 적용하려면 버킷 정책이 객체 업로드에 bucket-owner-full-control 삽입 ACL을 요구하도록 지정해야 합니다. [자세히 알아보기](#)

취소

변경 사항 저장

vue 코드

```
<template>
  <v-container>
    <h1>파일 리스트</h1>
    <!--file.Key를 하면 파일 이름이다 -->
    <div v-for="(file, index) in fileList" :key="file.Key">
      #{{ index + 1 }}: {{ file.Key }}
      <v-btn @click="deleteFile(file.Key)">x</v-btn>
    </div>
    <h1>파일 업로더</h1>
    <input
      id="file-selector"
      ref="file"
      type="file"
      @change="handleFileUpload()"
    />
    <v-button @click="uploadFiles">업로드</v-button>
  </v-container>
</template>
<script>
import AWS from "aws-sdk";
export default {
  data() {
    return {
      file: null,
      // s3설정 -> env파일 등에 숨기는 게 안전
      albumBucketName: "dangdang-bucket",
      bucketRegion: "ap-northeast-2",
      IdentityPoolId: "ap-northeast-2:81a948c5-f0c2-4e4b-ac0c-6ed0ffbc8b8",
      fileList: [],
    };
  },
};
```

```

created() {
  // 페이지 로드 시점에 파일을 불러와라 -> 우리는 일단 콘솔창에만 뜸
  this.GetFiles();
},

methods: {
  handleFileUpload() {
    this.file = this.$refs.file.files[0]; // 지금 선택된 파일이 data의 file로 저장되도록

    console.log(this.file, "파일이 업로드 되었습니다");
  },
  uploadFiles() {
    // 파일 업로드
    AWS.config.update({
      region: this.bucketRegion,
      credentials: new AWS.CognitoIdentityCredentials({
        IdentityPoolId: this.IdentityPoolId,
      }),
    });

    const S3 = new AWS.S3({
      apiVersion: "2012-10-17",
      params: {
        Bucket: this.albumBucketName,
      },
    });

    let photoKey = "folder/1.jpg"; // 업로드될 파일 이름 -> 해당 유저의 이름으로 폴더를 만들고 그 폴더 안에 들어가야 함
    S3.upload(
      {
        Key: photoKey,
        Body: this.file,
        ACL: "public-read",
      },
      (err, data) => {
        if (err) {
          console.log(err);
          return alert("에러");
        } else {
          console.log(data);
          this.GetFiles();
          alert("성공");
        }
      }
    );
  },
  getFiles() {
    // 파일 불러오기
    AWS.config.update({
      region: this.bucketRegion,
      credentials: new AWS.CognitoIdentityCredentials({
        IdentityPoolId: this.IdentityPoolId,
      }),
    });

    const S3 = new AWS.S3({
      apiVersion: "2012-10-17",
      params: {
        Bucket: this.albumBucketName,
      },
    });

    S3.listObjects(
      // 특정 폴더의 파일만 가져오도록
      {
        Delimiter: "/", // 이렇게 설정하면 root에 있는 모든 값을 가져오는 거
      },
      (err, data) => {
        if (err) {
          return alert("에러");
        } else {
          this.fileList = data.Contents;
          console.log(data);
        }
      }
    );
  },
  deleteFile(key) {
    AWS.config.update({
      region: this.bucketRegion,
      credentials: new AWS.CognitoIdentityCredentials({
        IdentityPoolId: this.IdentityPoolId,
      }),
    });
  }
}

```

```

const S3 = new AWS.S3({
  apiVersion: "2012-10-17",
  params: {
    Bucket: this.albumBucketName,
  },
});
S3.deleteObject(
  {
    Key: key,
  },
  (err, data) => {
    if (err) {
      return alert("에러");
    } else {
      console.log("삭제 성공");
      console.log(data);
      this.getFiles(); // 파일 삭제 후 다시 리스팅
    }
  }
);
},
},
};
</script>

```

기타

1차 정리파일

[jenkins - 예제 따라하기\(github버전\).\(2\)](#)

[jenkins - 우리 서버에 적용해보기 \(백엔드\).\(2\)](#)

[jenkins - 우리 서버에 적용해보기 \(프론트엔드\).\(2\)](#)

[다시 nginx + certbot \(2\)](#)


[DB - 백엔드 분리 \(2\)](#)


[basic_setting 만들기 \(2\)](#)


[S3 이미지 서버 설정 \(1\)](#)

이전 프로젝트 때 정리한 내용들

[원격 서버에 접속하기 \(2\)](#)

 [Docker \(2\)](#)

 [Nginx \(2\)](#)

 [Certbot \(2\)](#)

작업하면서 발생했던 에러들

[0919 \(2\)](#)

[0920 \(2\)](#)

[0921 \(2\)](#)

