**KSA 혁신성장 프로젝트**

**GBT 금연어플 작업 매뉴얼**

**작성자 장수민**

목차

# **1 . VPC 생성 3**

## **2. EC2 인스턴스 생성5**

**(1) 인스턴스 종료 방지7**

**(2) 탄력적 IP 생성 및 연결8**

**3. 젠킨스 설치10**

**4. 젠킨스와 github 연결16**

**5. Elastic Beanstalk를 이용한 Backend CICD25**

**6. 서로다른 VPC 연결 (피어링) 38**

**7. 역방향 proxy 설정41**

**8. RDS 구축43**

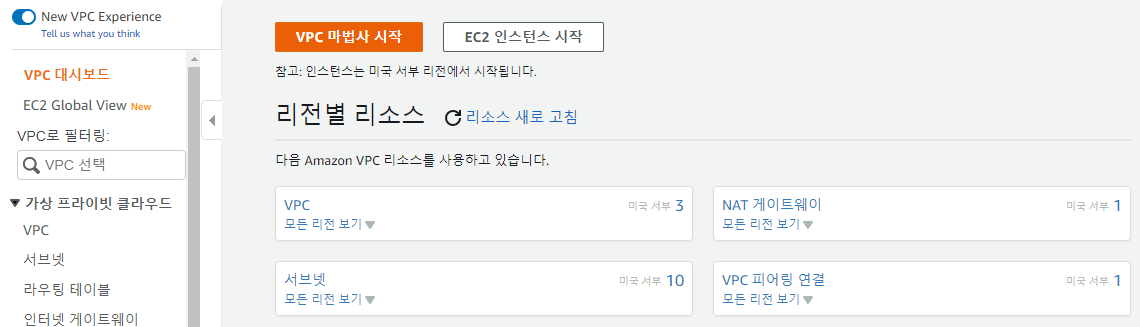
**9. 안드로이드 CICD46**

**10. 통합파이프 라인49**

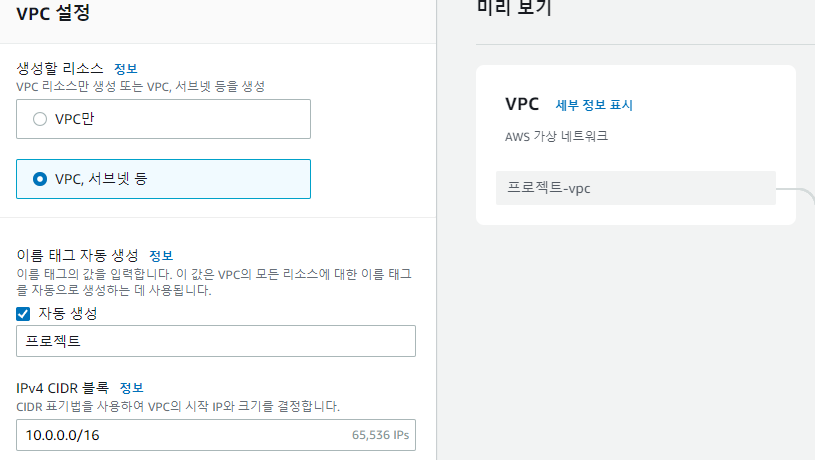
**1 . VPC 생성**

-현재 프로젝트에서 VPC를 2개 만들고 VPC안의 인스턴스가 서로 통신할 수 있게 해주기위해 피어링을 사용할 것입니다

VPC를 생성하기 위해 vpc대시보드에 VPC마법사 시작 버튼을 클릭합니다

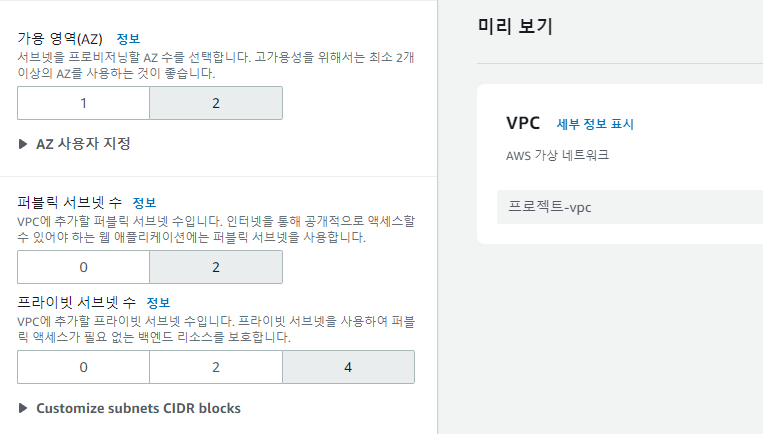


피어링을 하려면 VPC간에 겹치는 CICD 블록이 있으면 안되기 때문에 VPC를 만들 때 하나는 IPv4 CICD블록 설정을 10.0.0.0/16 다른 하나는 1.0.0.0/16 로 설정 했습니다



가용 영역은 고가용성을 위해 2개로 설정합니다

퍼블릭, 프라이빗 서브넷의 수는 프로젝트의 구성에 따라 퍼블릭 서브넷 2개, 프라이빗 서브넷 4개로 선택했습니다



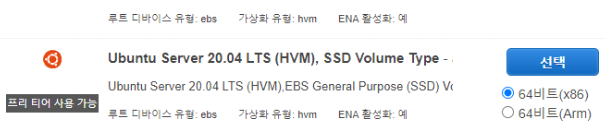
VPC 설정을 한후 VPC를 생성합니다



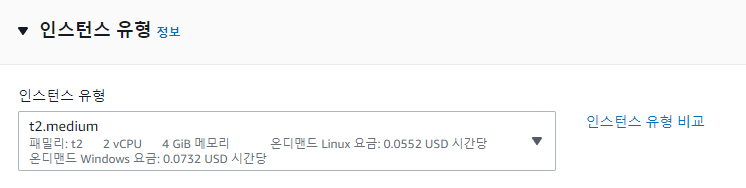
**2. EC2 인스턴스 생성**

매뉴얼에는 인스턴스 유형을 t2.small로 하였지만 실제 프로젝트에서 돌고있는 젠킨스용 ec2는 t2.midium으로 설정했습니다. 후에 있을 안드로이드 CICD를 하기위해서는 t2.small유형의 용량이 부족하기 때문입니다

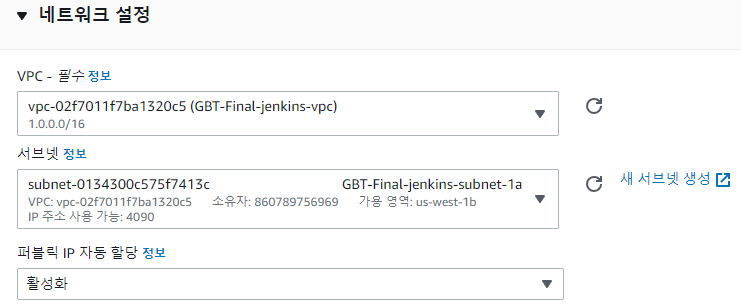
프로젝트에서 사용할 AMI로는 Ubuntu로 설정합니다



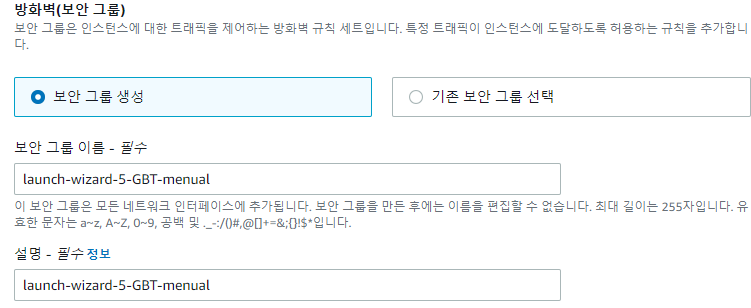
인스턴스 유형은 후에 있을 안드로이드 CICD를 하기위해 용량일 충분한 t2.midium으로 선택했습니다



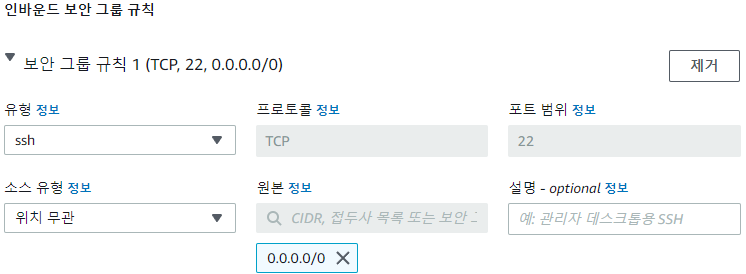
네트워크 설정에는 이전에 만들었던 VPC와 서브넷으로 설정하고 퍼블릭 ip는 만들고자하는 ec2의 용도에 맞게 public이면 활성화 private이면 비활성화를 선택합니다



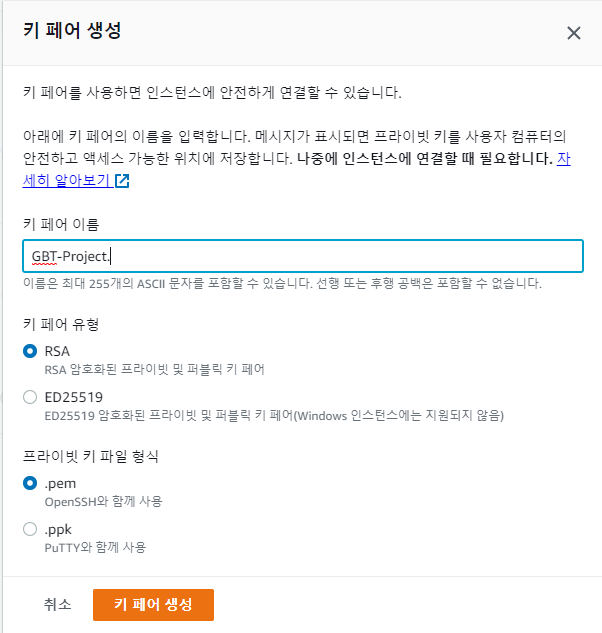
보안 그룹을 생성합니다



public 인스턴스일 경우 ssh, HTTP(웹), 8080(젠킨스)로 접속이 가능하도록 인바운드 보안 그룹에 추가합니다

private 인스턴스인 경우 ssh, 8080만 인바운드 보안 그룹에 추가합니다 

키페어를 생성하여 다운로드하고 인스턴스를 시작합니다

프라이빗 키 파일 형식은 우분투를 사용할 것이기 때문에 우분투가 사용하는 .pem으로 선택합니다

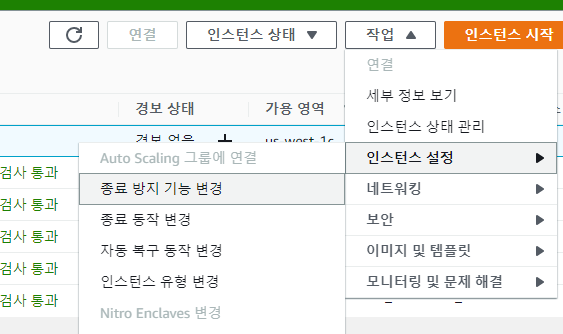
설정을 완료하고 인스턴스 시작버튼을 누르면 인스턴스가 생성이 됩니다



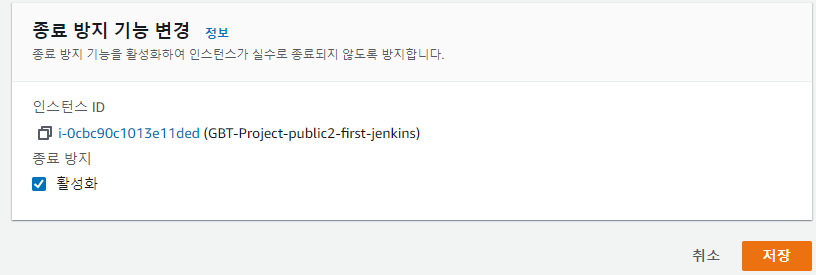
**(1) 인스턴스 종료 방지**

인스턴스를 실수로 종료시키지 않기 위해 종료 방지를 설정합니다

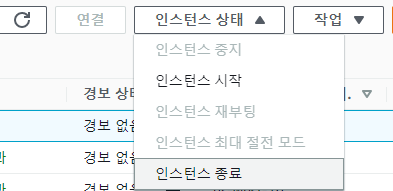
설정할 인스턴스를 선택하고 작업에 인스턴스 설정-종료방지 기능 변경 클릭

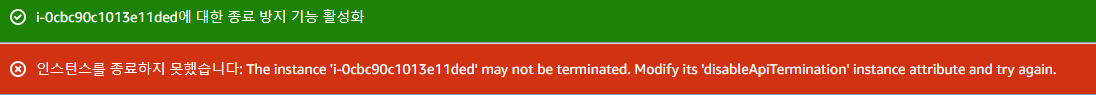


종료 방지를 활성화 시키고 저장합니다



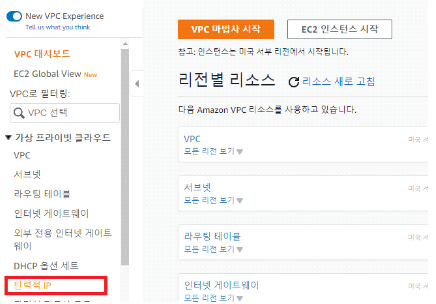
종료방지를 활성화 시키면 종료를 시켜도 종료 시킬 수 없다는 메시지가 뜨며 종료할 수 없게 됩니다





**(2). 탄력적 ip 생성 및 연결**

먼저 탄력적 IP를 생성합니다 (ec2가 중단이 된 후 ip가 자동으로 바뀌게 되는데 이것을 탄력적 ip가 고정시켜줍니다)

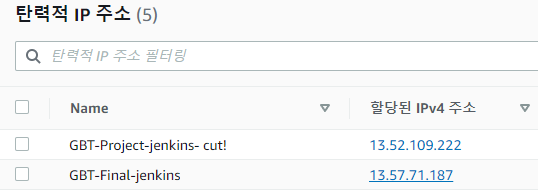


탄력적 IP주소 할당버튼을 눌러 탄력적 IP주소를 할당 받습니다



탄력적 ip 연결

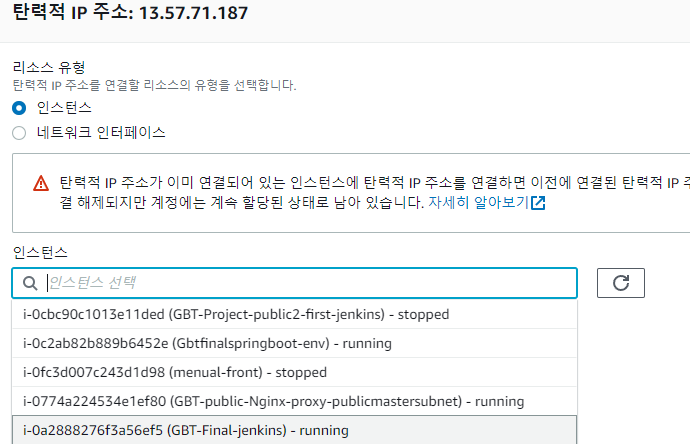
EC2에 할당된 탄력적 ip주소 연결하겠습니다 먼저 할당된 IPv4 주소를 클릭합니다



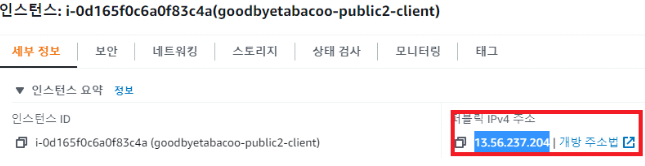
탄력적 IP주소 연결 버튼을 클릭합니다

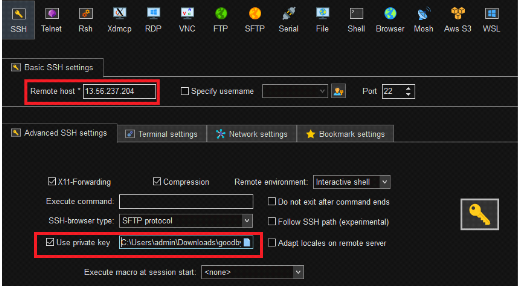


연결할 ec2 인스턴스를 선택 후 연결하면 인스턴스 IP가 고정이 됩니다

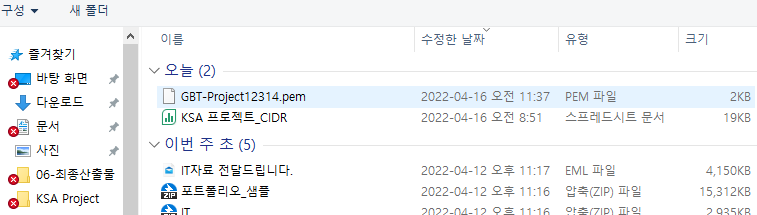


**3. 젠킨스 설치**

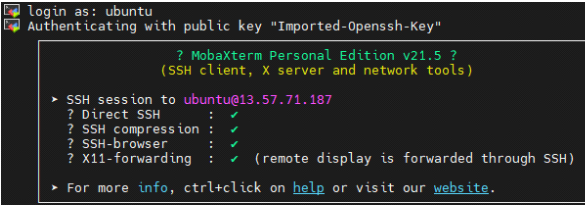
접속할 인스턴스의 퍼블릭 ip주소를 Linux에 입력



인스턴스에 접속하기 위해 발급받은 키 파일(.pem) 선택



Ubuntu로 접속해준다



java버전 11을 쓰기 때문에 jdk버전 11을 다운받는다

먼저 apt update를 한후에



jdk버전 11을 다운받는다



JAVA\_HOME 변수 정의

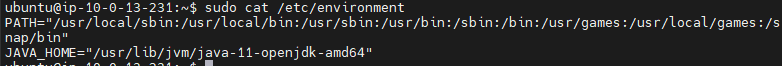
$ sudo nano /etc/environment

JAVA\_HOME="/usr/lib/jvm/java-11-openjdk-amd64"

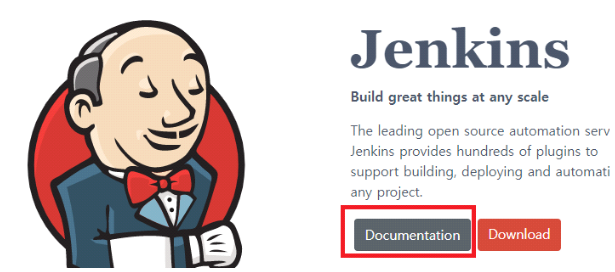


JAVA\_HOME 변수 확인

$ sudo cat /etc/environment



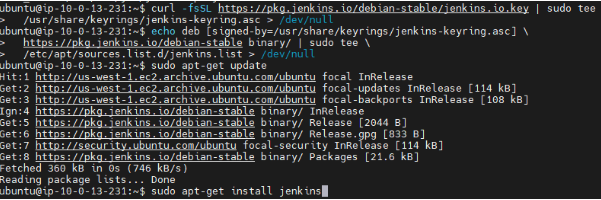
https://www.jenkins.io/ 홈페이지 들어가서 Documentation 클릭



Installing Jenkins 에 Linux 클릭 하면 나오는 코드로 Linux에 jenkins를 설치합니다

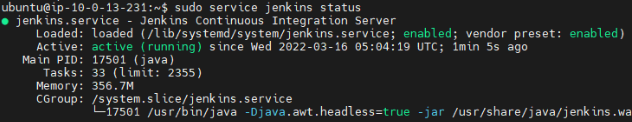






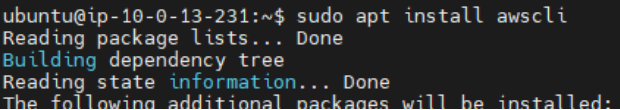
젠킨스 설치확인

sudo service jenkins status



aws명령어 사용을 위한 awscli 설치

Sudo apt install awscli

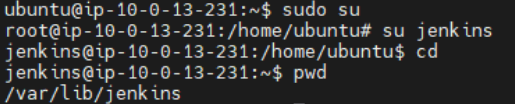


Aws 설치 확인

aws —version



Aws 계정 설정



액세스 키 ID, 시크릿 액세스 키, region name 은 발급 받은 걸 사용합니다

$ aws configure



설치 후 ec2의 {ip주소}:8080을 주소창에 입력하면 아래처럼 뜹니다

아래의 빨간 박스 안에는 최초 실행시 비밀번호 경로 확인하는 코드입니다



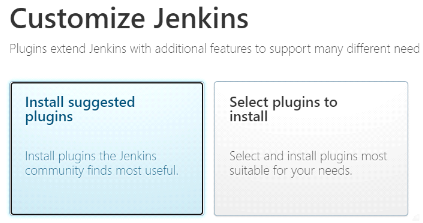
Jenkins 에서 ubuntu로 나온후

최초실행시 비밀번호 경로를 확인하기위해

Sudo cat /var/lib/jenkins/secrets/initialAdminPassword를 입력후 나오는 값을 복사하여 젠킨스 페이지의 Administrator password 칸에 넣어줍니다



Install suggested plugins 선택



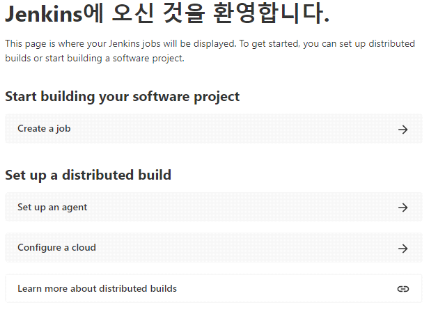
설치후 계정을 생성하는 페이지가 나오면 계정을 생성합니다



생성을 완료하면 젠킨스 주소를 확인 할 수 있습니다



젠킨스 생성 완료



**4. 젠킨스와 Github연동**

EC2에 git이 설치되어있는지 먼저 확인(만약 없다면 sudo apt install git)

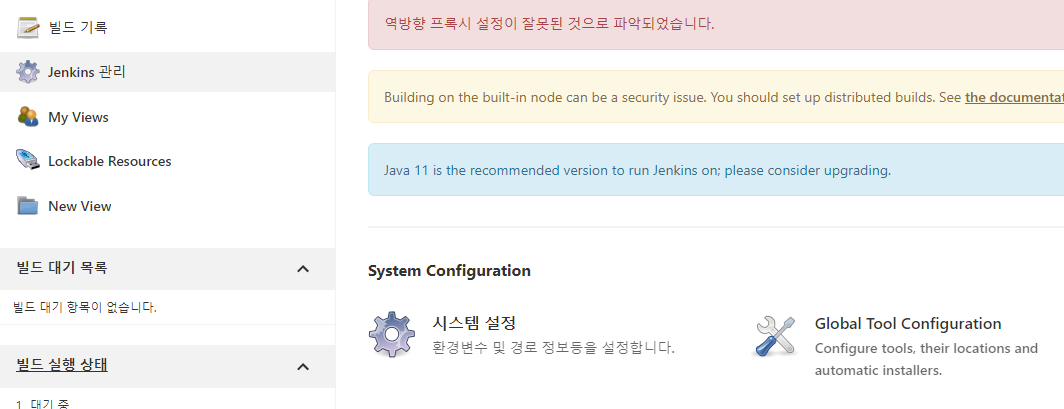
$ sudo git –version



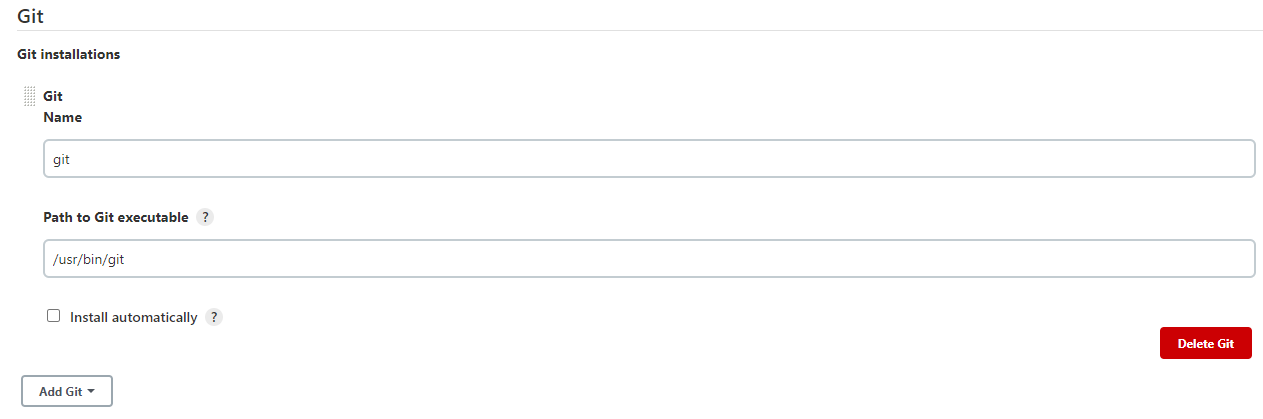
sudo which git으로 git의 설치 경로 확인



젠킨스 관리에서 Global Tool Configuration 클릭



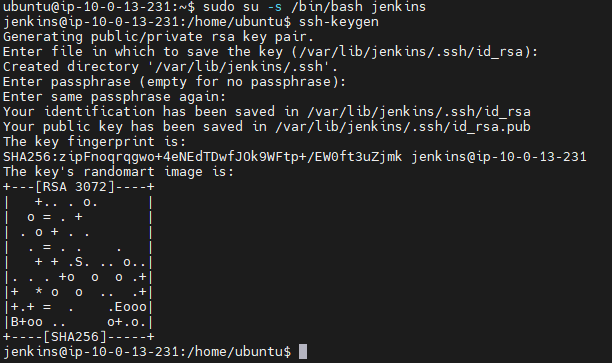
Git의 경로를 확인했던 경로로 수정 후 저장



sudo su 수퍼유저권한으로 젠킨스를 bash형태로 접속하여 keygen 생성

$ sudo su -s /bin/bash Jenkins

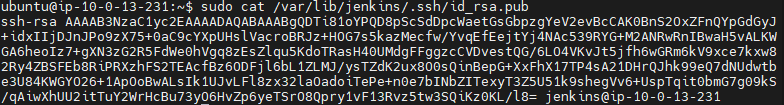
$ ssh-keygen



exit명령어로 jenkins에서 빠져나온 이후 위에서 생성한 퍼블릭 키를 cat 명령어로 조회합니다.

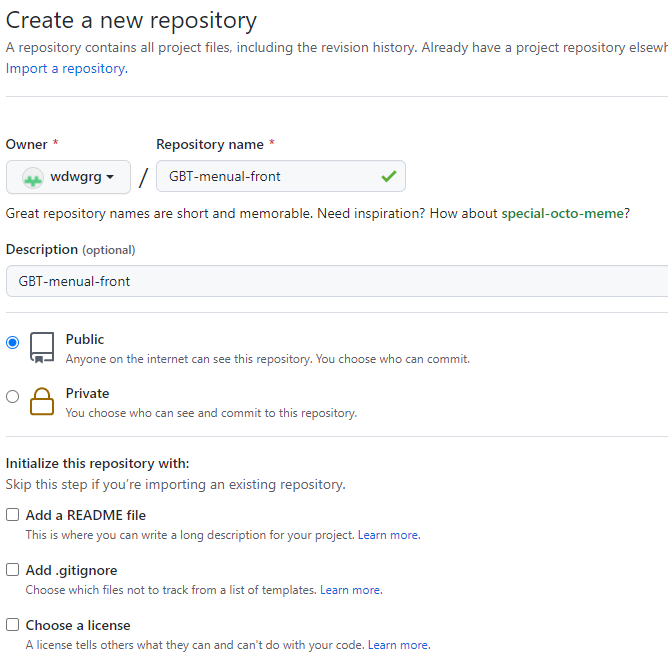
$ sudo cat /var/lib/jenkins/.ssh/id\_rsa.pub



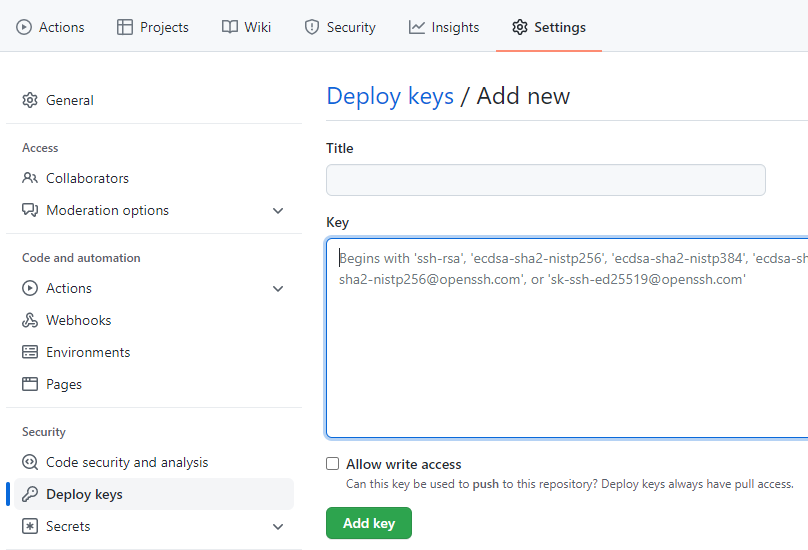


jenkins와 연동할 레포지토리 생성

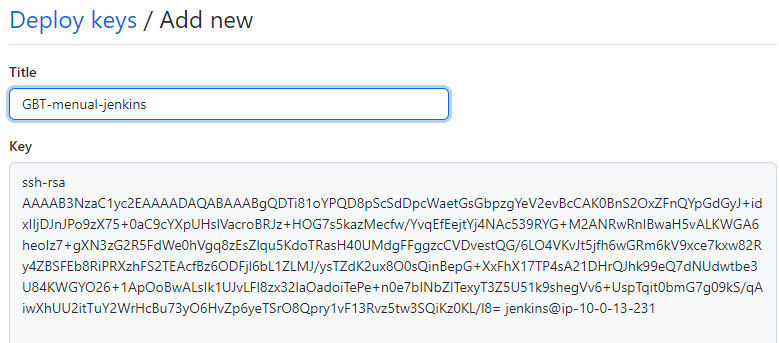
github 에서 젠킨스와 연동할 레포지토리를 생성합니다

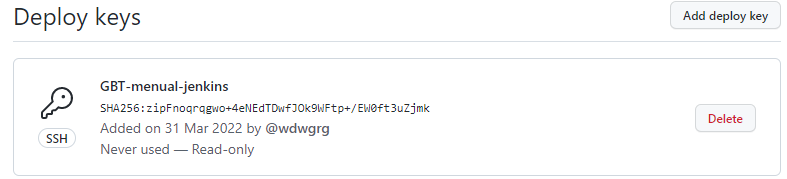


생성한 레포지토리의 설정에서 Deploy keys에서 add deploy key 선택



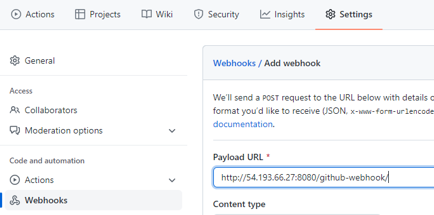
key 부분에 위에서 cat 명령어로 조회했던 jenkins 키 입력후 add key 버튼을 누르면 발급된 키를 확인할 수 있습니다

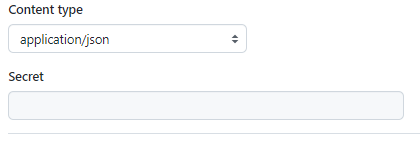


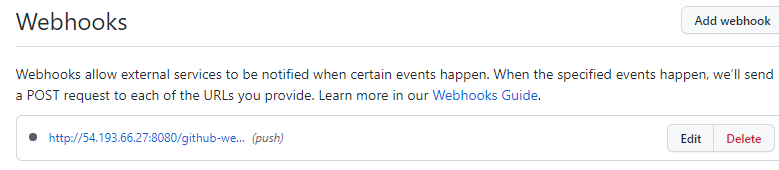


개발자가 push를 했을 때 jenkins에서 인식할 수 있도록 webhook 설정을 합니다

Webhooks 에서 EC2의 주소와 젠킨스 포트를 입력하고 github-webhook/ 을 Payload URL에 입력합니다

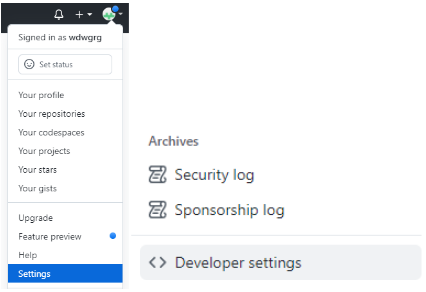


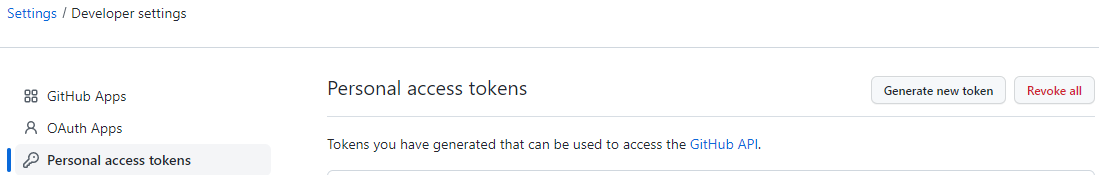
컨텐츠 타입으로는 application/json 타입으로 설정하고 Webhook을 생성합니다  




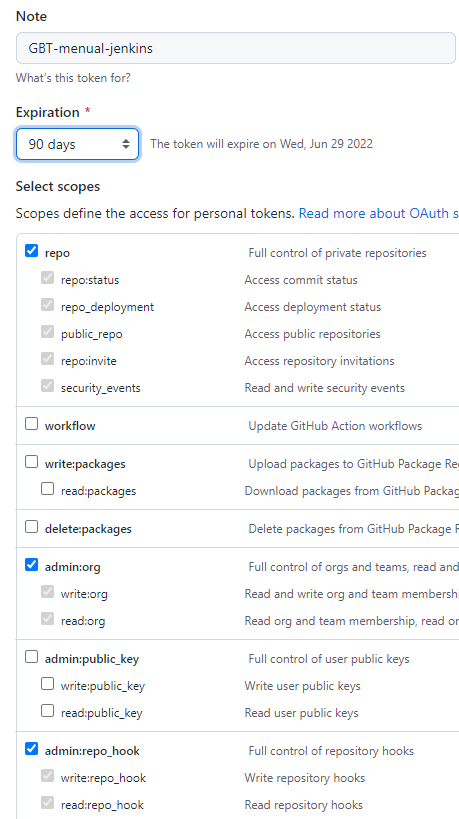
jenkins에서 github로 접근할 수 있도록 토큰을 생성합니다

자신의 계정의 setting에 Develop setting에 personal access token 클릭합니다



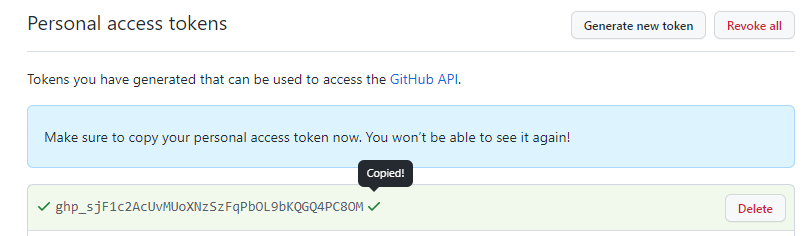


Generate new token 클릭을 한 후 토큰 이름을 설정해주고 만료일과 접속 권한을 설정해줍니다



토큰을 만들면 토큰 키가 나오는데 토큰 키는 지금이 아니면 볼 수 없으므로 다른곳에 저장을 하는 것이 좋습니다

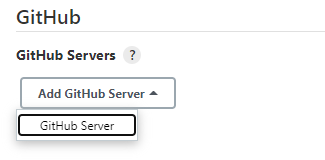
토큰 키: ghp\_QtN8EDjNodRI9ceyKHvxksUOSg01dx4JuEtg



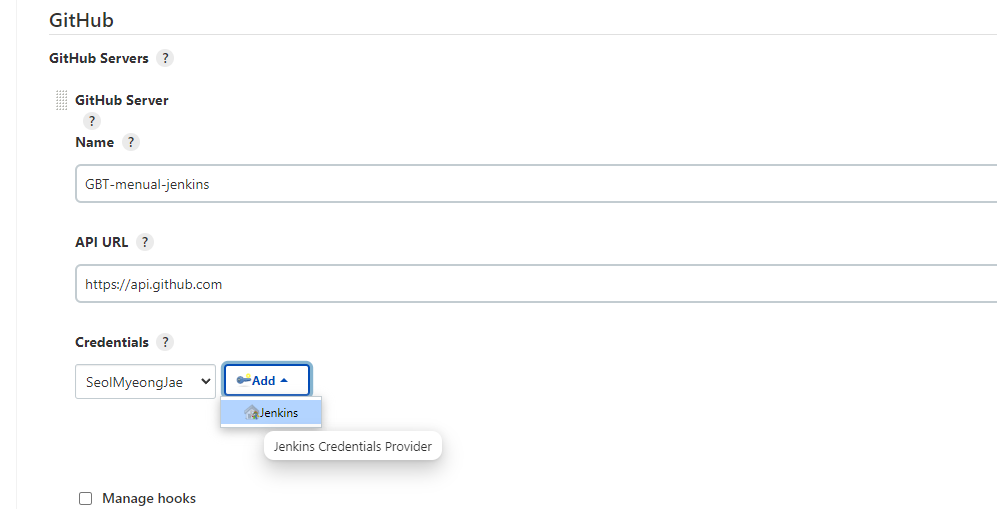
토큰을 발급 받은 후 젠킨스로 돌아와 젠킨스 설정에서 시스템 설정으로 들어갑니다



Github를 찾아서 add github를 해줍니다

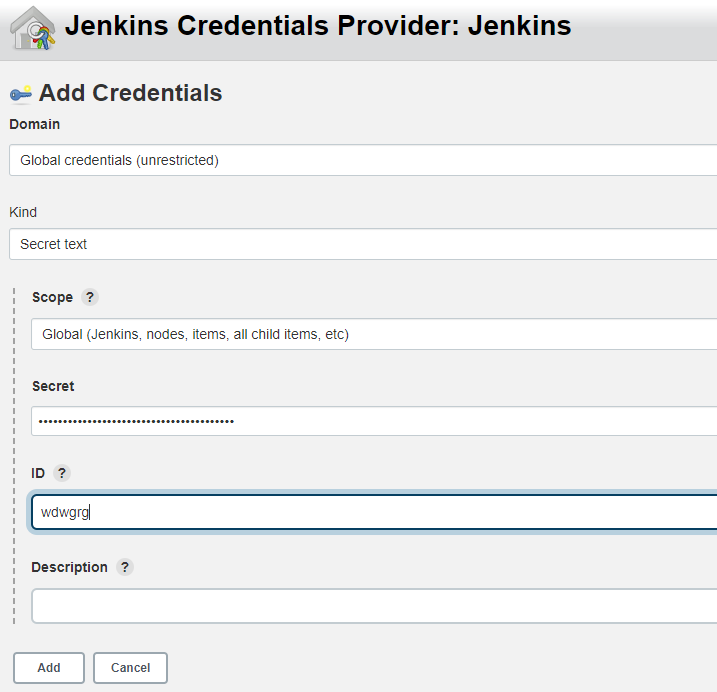


Credential을 추가하는데 Credential 항목에 Add부분에 jenkins클릭합니다



Kind 부분을 secret text로 바꾸고 Secret 부분에는 아까 발급 받은 key값을 넣습니다

ID부분에는 github Id를 적으면됩니다



**Credentials 에서 위에서 만든 Credentials을 선택 후 test connection을 버튼을 눌러 실행합니다**

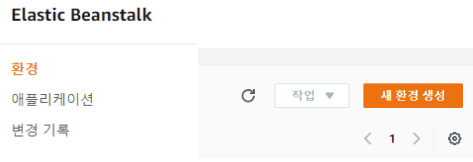
Credentials verified for user YeongSeon1, rate limit: 4999 이렇게 뜨면 성공한것이기 때문에 성공한 것을 확인하고 저장을 해줍니다



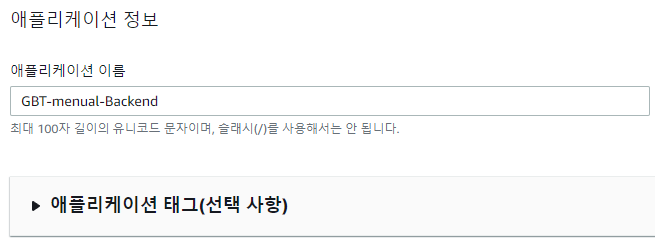
**5. Elastic Beanstalk를 이용한 Backend CICD**

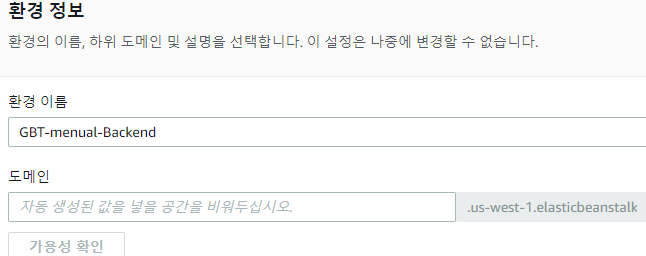
Elastic Beanstalk은 어플리케이션을 신속하게 배포하고 관리할 수 있고 배포를 하면 자체적으로 로드밸런싱, 오토 스케일링을 하며 배포를 할 수 있기 때문에 개발하는데 여러 장점이 있습니다

엘라스틱 빈스톡에서 새 환경을 만들어 줍니다

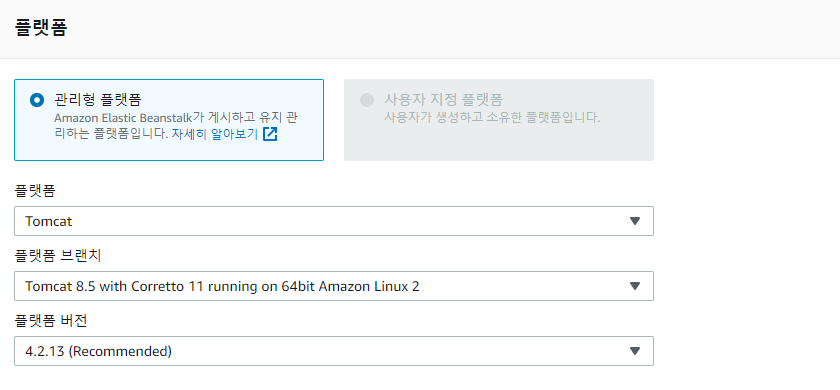
****

애플리케이션 이름과 환경 정보 이름을 설정 합니다

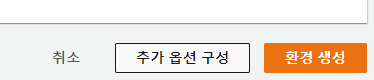


****

**플랫폼에는 빌드를 war로 할 것 이기 때문에 Tomcat을 선택해 줍니다**

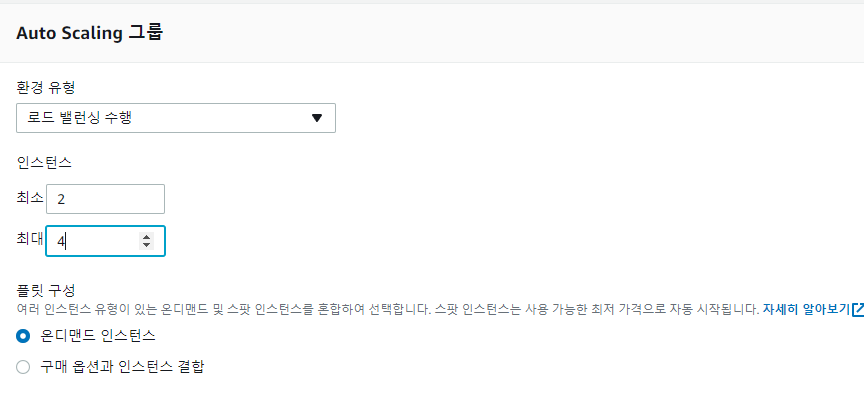
****

**또한 추가적으로 구성할것이 있기 때문에 추가 옵션 구성을 클릭합니다**

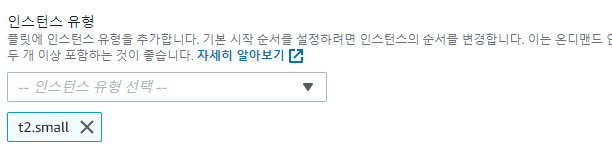
****

**Elastic beanstalk으로 로드 밸런싱을 수행할 것이기 때문에 환경 유형을 로드 밸런싱 수행으로 설정 합니다**

**로드밸런싱을 하려면 인스턴스가 최소 2개가 필요하기 때문에 인스턴스의 수를 최소2, 최대 4 으로 설정하면 오토 스케일링이 자동으로 그 범위 내에서 인스턴스의 수를 조정합니다.**

****

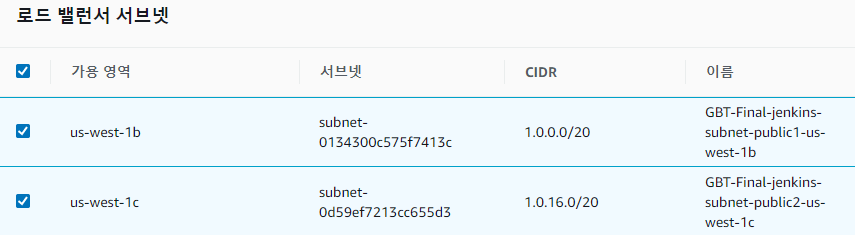
생성하려는 인스턴스의 유형을 설정합니다

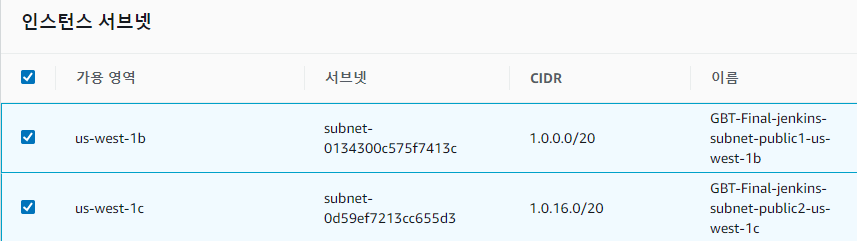
****

**VPC는 앞에서 만들었던 젠킨스 VPC와는 다른 VPC로 설정합니다**

**서브넷은 생성될 EC2에 반영할 서브넷을 설정합니다**

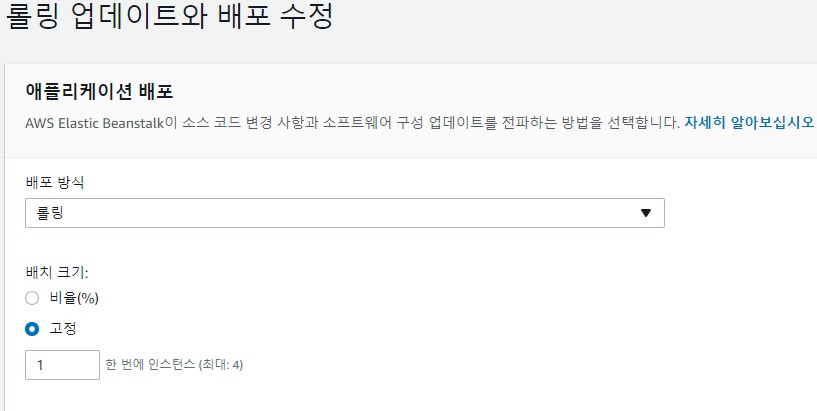
****

****

****

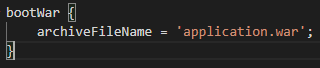
**배포방식은 롤링으로 선택했습니다**

롤링 배포는사용 중인 인스턴스 내에서 새 버전을 점진적으로 교체하는 것으로 무중단 배포의 가장 기본적인 방식으로 관리하기 편하다는 장점이 있습니다.

****

****

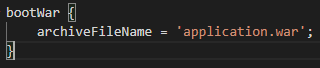
환경을 구성한후 배포할 backend의 build.gradle에서 war파일 name지정합니다

****

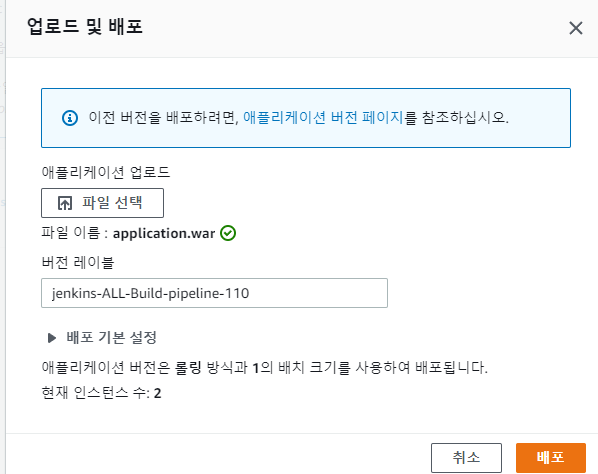
Elastic beanstalk 에서 업로드 및 배포 클릭

****

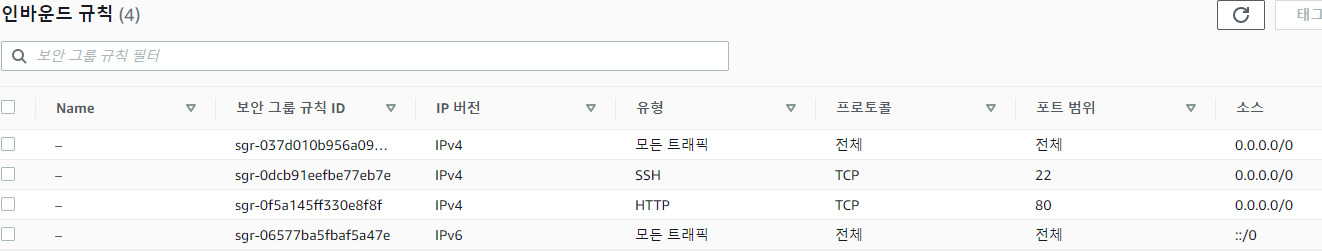
환경을 구성한후 배포할 backend의 build.gradle에서 war파일 name지정을 하고 빌드를 하여 application.war파일을 생성합니다

****

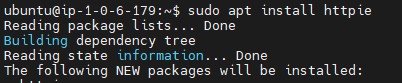
로컬에서 만든 war 파일을 업로드 하여 배포합니다

****

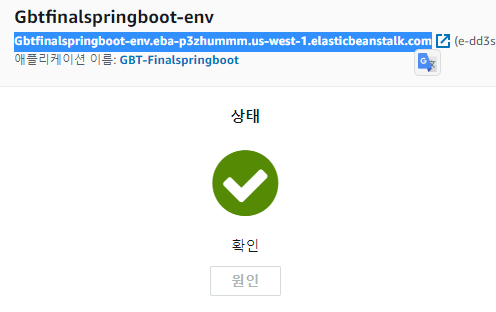
현재 프로젝트에서 배포한 war파일이 private ec2에 올려져있기 때문에 잘 배포가 되었는지 확인하기 위해선 빈스톡 private ec2에 우분투에서 접속이 가능하도록 보안그룹에 인바운드를 설정해준뒤에 우분투에서 rest api로 요청하여 배포를 확인 해야합니다

****

Rest api를 사용하기 위해서 httpie를 설치해줍니다

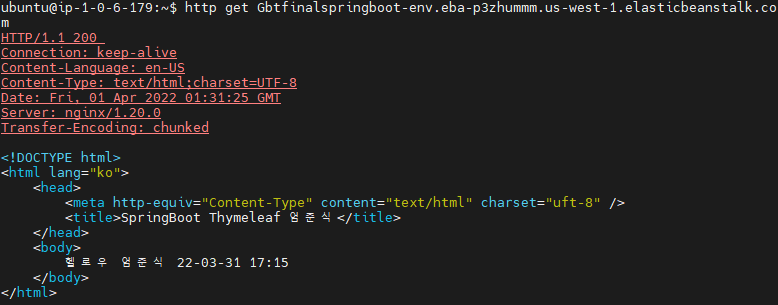
****

엘라스틱 빈스톡의 주소를 복사한후

****

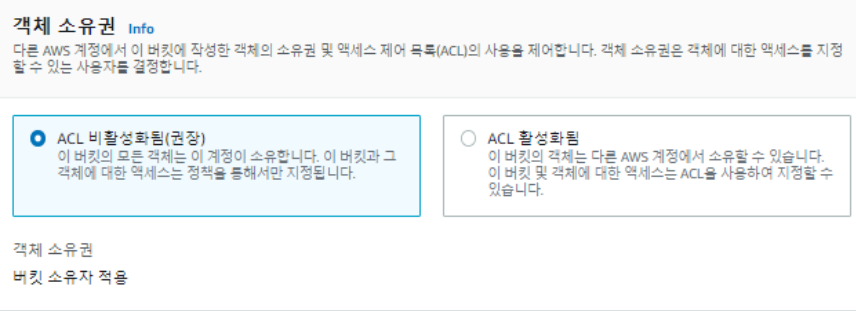
우분투에서 rest api 요청하는 명령문 뒤에 붙여 줍니다

$ sudo http get 표시한주소

****

잘 배포되었는지 확인 할 수 있습니다

Backend가 업로드될 S3 버킷을 생성합니다

****

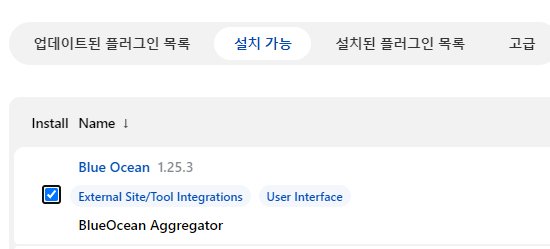
S3에 업로드 된 객체를 어디서든 접근가능하게 하기위해서 퍼블릭 액세스 설정해줍니다

****

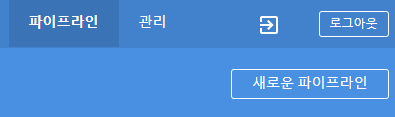
젠킨스에서 블루오션을 설치 합니다

블루 오션은 기존보다 대시보드가 좀더 시각화 되어있고 파이프 라인이 시각화 되어 있어 문제가 발생시 어느 스테이지 또는 스텝이 문제가 있는지 더 쉽게 알 수 있습니다

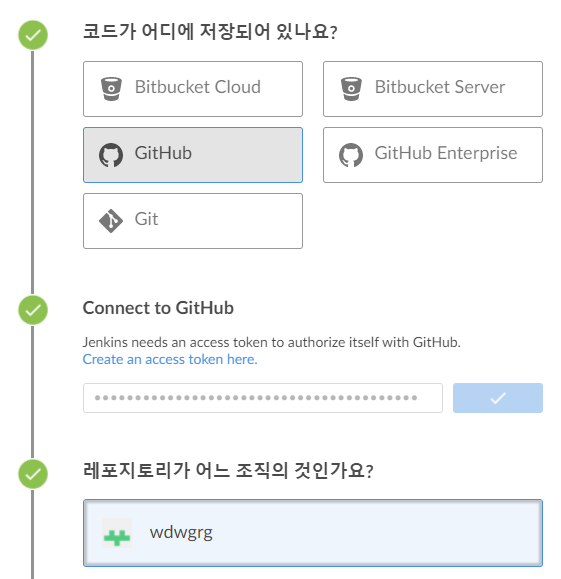
젠킨스 관리의 플러그인 관리에서 blue ocean을 설치합니다

****

설치후 블루오션으로 들어갑니다

****

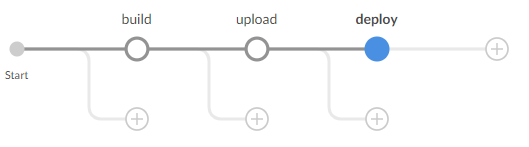
Connect to Github란에는 이전에 만든 access 토큰값을 넣어 줍니다

****

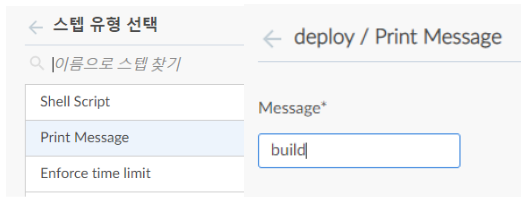
연결하려는 레포지토리 선택후 파이프 라인 생성

****

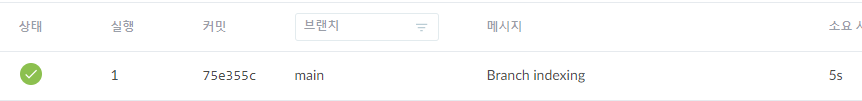
**블루 오션의 파이프 라인입니다 ( + 버튼을 눌러 stage를 추가하는 형태입니다)**

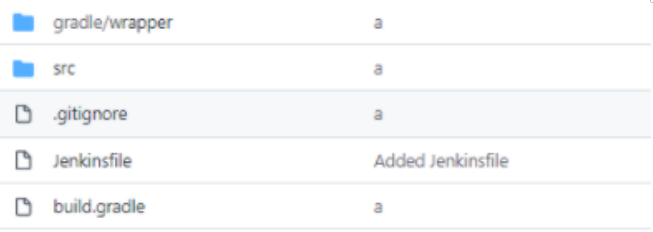
****

**각 스테이지 별로 이름을 넣고 스텝을 추가합니다 (print message를 추가)**

****

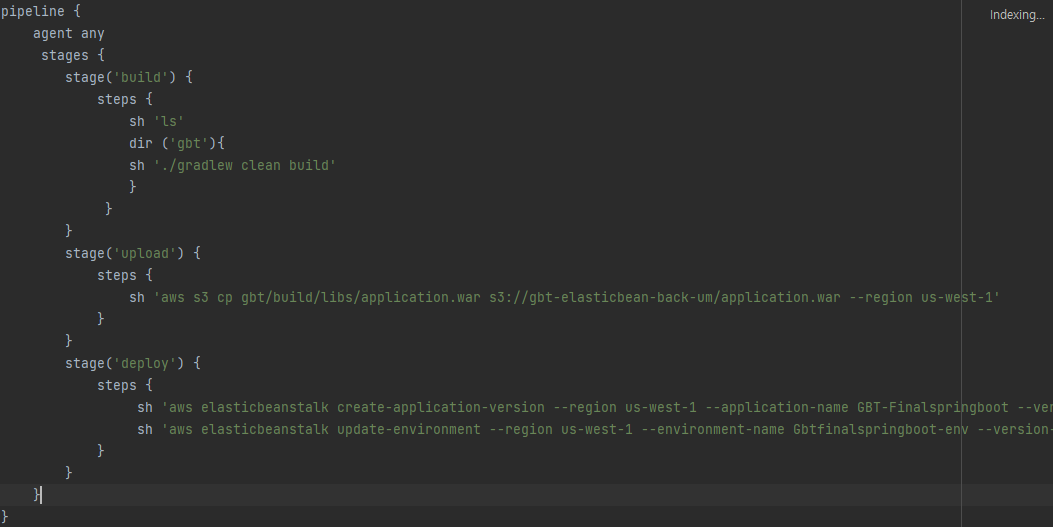
**설정을 완료하고 저장을 하면 레포지토리에 Jenkinsfile 이 생깁니다**

****

****

**생성된 jenkins파일을 레포지도리에서 pull해서 가져온 후**

**jenkins파일에서 파이프 라인을 작성하면 됩니다**

****

**--build stage 설명--**

**gradle wapper를 이용하여 build 후 war 파일을 생성합니다.**

**./gradlew clean build**

**--upload stage 설명--**

**AWS CLI를 이용하여 Jenkins 서버에 생성된 war 파일을 S3로 복사합니다.**

**aws s3 cp gbt/build/libs/application.war s3://gbt-elasticbean-back-um/application.war --region us-west-1**

**build/lib/application.war**

**gradle로 build 후 생성되는 war 파일의 위치를 지정합니다.**

**저는 application.war로 생성되도록 설정하였기 때문에 위와 같이 지정하였습니다.**

**s3://gbt-elasticbean-back-um/application.war**

**생성 한 S3의 버킷이름(gbt-elasticbean-back-um)과 복사될 파일명을 입력합니다.**

**--region us-west-1**

**Elastic Beanstalk이 등록된 region을 입력합니다.**

**--deploy stage 설명—**

**deploy은 두 개의 step으로 나눠서 진행합니다.**

**1) create-application-version (Elastic Beanstalk에 새로운 애플리케이션 버전을 등록합니다.)**

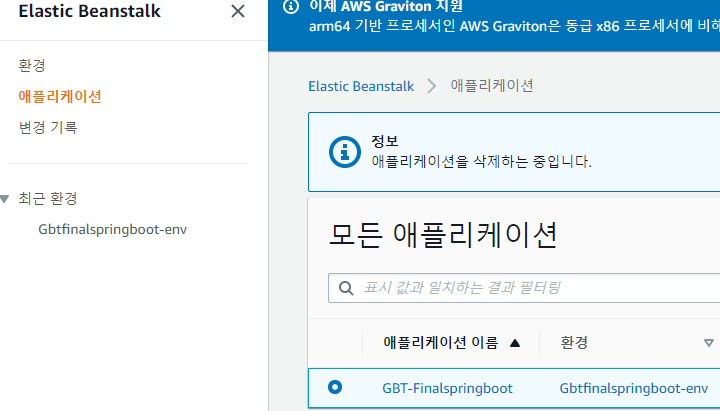
**aws elasticbeanstalk create-application-version --region us-west-1 --application-name GBT-Finalspringboot --version-label ${BUILD\_TAG} --source-bundle S3Bucket="gbt-elasticbean-back-um",S3Key="application.war**

**--region us-west-1**

**Elastic Beanstalk이 등록된 region을 입력합니다.**

**--application-name GBT-Finalspringboot**

**Elastic Beanstalk에 등록 된 애플리케이션명을 입력합니다**

****

**--source-bundle S3Bucket="gbt-elasticbean-back-um",S3Key="application.war”**

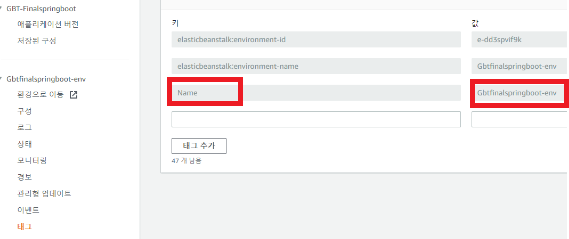
**버전에 등록할 S3 버킷 이름과 파일명을 입력합니다.**

**2) update-environment (새로 등록된 버전으로 배포를 시작합니다.)**

**aws elasticbeanstalk update-environment --region us-west-1 --environment-name Gbtfinalspringboot-env --version-label ${BUILD\_TAG}**

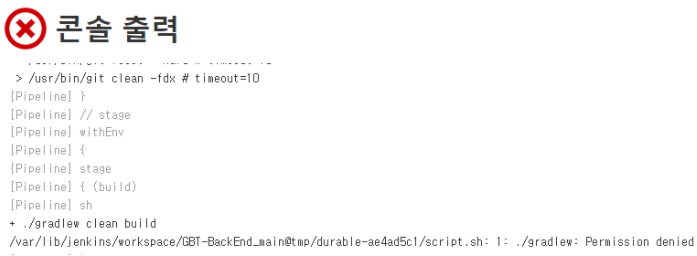
**--environment-name Gbtfinalspringboot-env**

**Elastic Beanstalk 환경 이름을 입력합니다(엘라스틱 빈스톡의 어플리케이션 테그항목에서 확인 가능합니다)**

****

**---gradlew를 사용하여 build 시 발생하는 permission 오류 ---**

**파이프 라인 생성후 gradlew 를 사용하여 build 시 permission deny 오류가 뜰 수 있습니다**

****

**이유는 window 환경에서 작업해서 소스를 push 했기 때문에 window 환경에서 파일 생성시 기본권한이 644로 생성되어 있기 때문입니다**

**해결 방법 : 일단 build 하여 war파일을 생성하면 workspace가 생기는데**

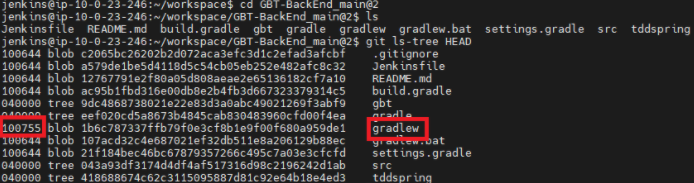
**우분투에서 jenkins의 workspace까지 들어갑니다**

****

**그다음 ls로 해당하는 파이프 라인 이름을 확인한후**

**해당하는 파이프라인에 들어가서 ls로 gradlew 권한 확인(아래의 사진은 권한설정한 후의 사진이라 10755로 나타나 있지만 고치기전에는 10644로 되어있습니다)**

**$ git ls-tree HEAD (권한 확인)**

****

**우선 해당파일을 clean build를 하고 ( 1 - clean, 2 - build)**

****

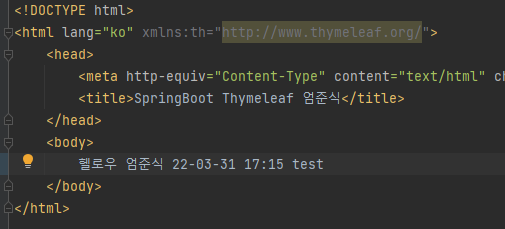
**권한을 변경 후 커밋,푸쉬를 해준뒤에 다시 권한을 확인합니다**

**$ git update-index --add --chmod=+x gradlew (권한 변경 코드)**

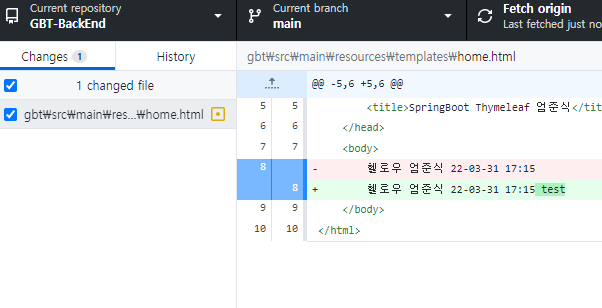
**권한이 644에서 755로 바뀐 것을 확인할 수 있습니다**

****

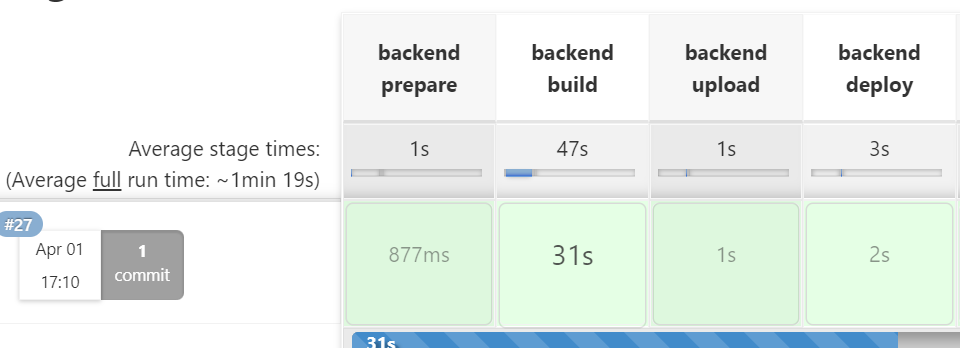
**배포가 잘되었는지 확인하기 위해 html을 수정하고**

****

**깃허브에 커밋 푸쉬를 해주면**

****

**젠킨스에서 자동으로 빌드후 배포되는 것을 확인할 수 있다**

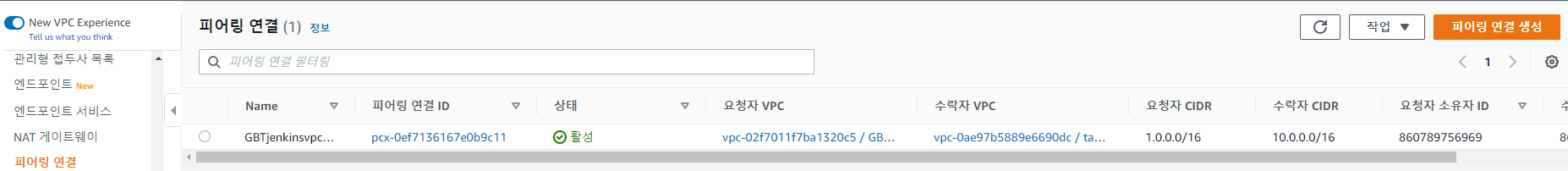
****

**배포확인은 위에서 했던 우분투에서 rest api로 확인해 주면 됩니다**

**6. 서로다른 VPC 연결 (피어링)**

현재 프로젝트에서 젠킨스가 설치된 EC2의 VPC와 elactic beanstalk으로 만든 EC2의 VPC는 서로 다른 vpc라서 통신이 안되기 때문에 통신을 시켜주기 위해 VPC 끼리 피어링으로 연결 해주는 작업이 필요합니다

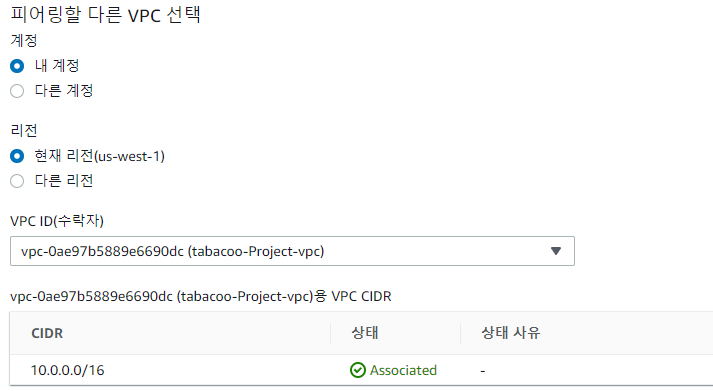
VPC에서 피어링 연결을 클릭한후 피어링 연결 생성

****

피어링의 이름을 입력하고 요청을 할 VPC와 요청을 받고 수락할 VPC를 선택합니다

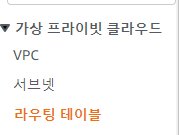
저는 요청할 VPC에 젠킨스에 설정된 VPC를 넣고 요청받을 VPC에는 빈스톡에 설정된 VPC를 선택했습니다

****

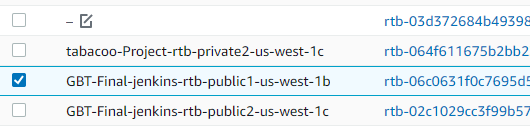
****

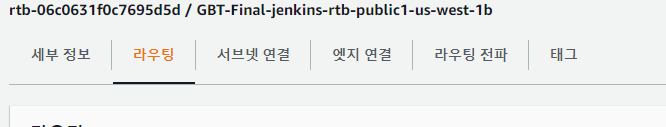
피어링을 만들고 난 뒤 상태창을 보면 대기중이라고 뜰텐데 피어링 요청을 수락하면 피어링이 활성화 됩니다

피어링이 설정된후 각각의 VPC에 해당하는 라우팅 테이블을 설정 합니다

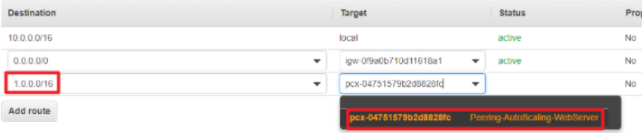
****

VPC 선택후 라우팅 테이블으로 들어가서 라우팅 편집

****

****

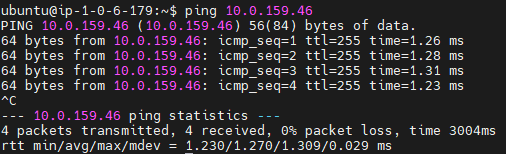
****

****

라우팅을 연결한후 우분투로 연결이 되는지 ping으로 확인 할 것이기 떄문에

보안에 ssh에 대한 인바운드를 허용해 줍니다 (저 같은 경우는 모든 트래픽을 허용했습니다)

이전에 설정 했던 대로 요청을 보내는 젠킨스 서버에서 다른 VPC에 해당하는 EC2 IP주소에 Ping을 보내봅니다

****

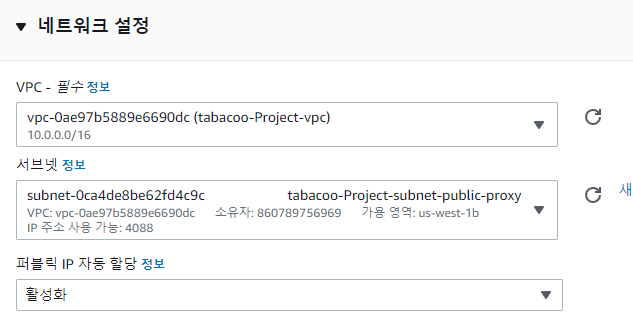
이렇게 요청을 보낸 것을 받으면 피어링이 성공한 것입니다

**7. 역방향 proxy 설정**

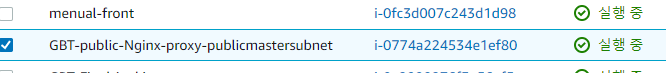
외부에서 들어오는 요청을 받아 프록시로 backend에 보내는 proxy 서버를 만듭니다

Ec2를 하나 생성 합니다

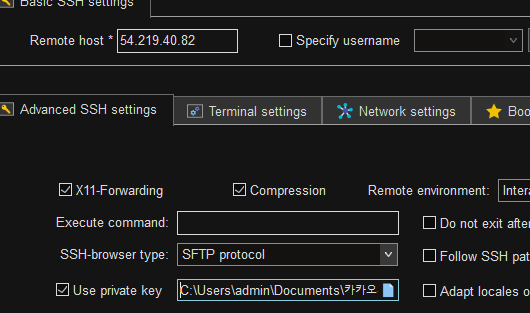
VPC는 backend와 동일한 vpc로 설정합니다

****

외부의 요청을 받아서 backend로 넘길 것이기 때문에 퍼블릭 ip를 활성화 하고 생성합니다

****

Ec2를 만든 후 해당 ec2에 우분투로 접속해 nginx를 설치 합니다

****

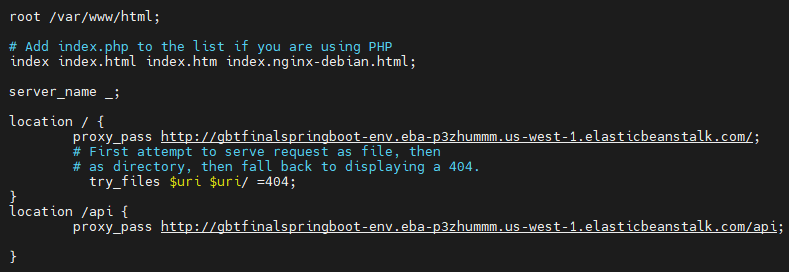
****

Nginx를 설치후 proxy pass설정을 합니다

$ sudo nano /etc/nginx/sites-available/default

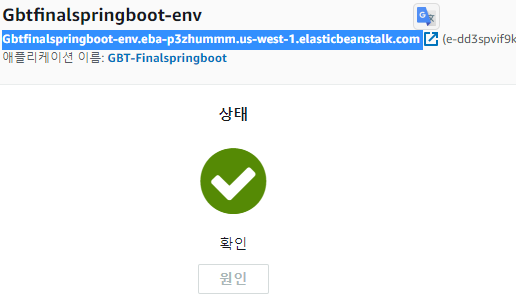
****

Proxy pass를 설정합니다

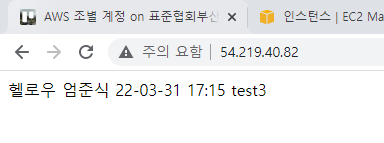
****

Location / { } 는 nginx 서버로 요청일 들어오면 어디와 연결을 시켜줄지 정의합니다

Proxy pass로 넘겨줄 주소로는 엘라스틱 빈스톡의 주소로 설정합니다

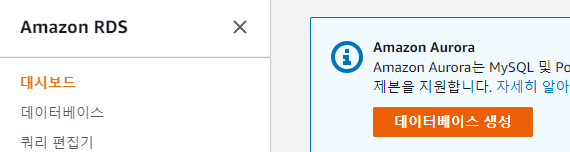
****

프록시 ip주소로 요청을 보내고 backend에 전달이 잘 되었는지 확인해 봅니다

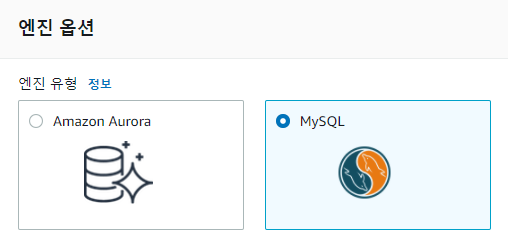
****

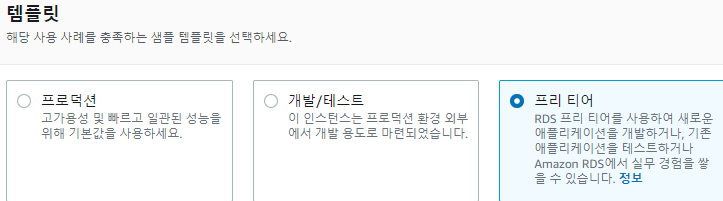
**8. RDS 구축**

AWS RDS에서 데이터 베이스를 생성 합니다

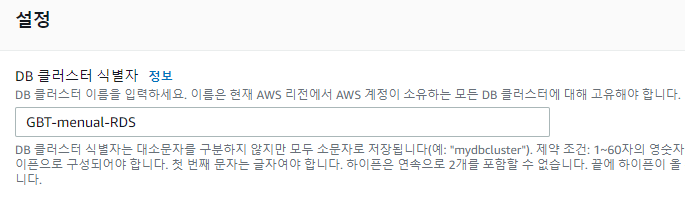
****

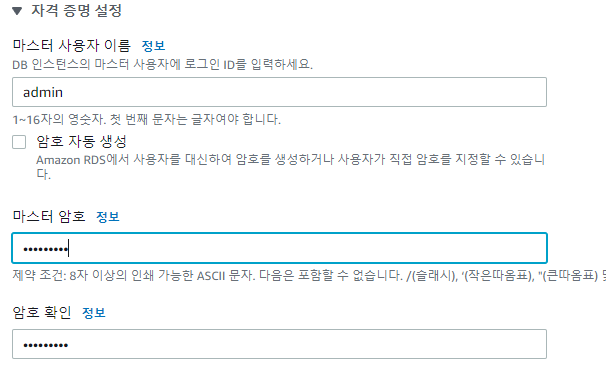
Mysql을 사용하기 위해 Mysql을 선택합니다

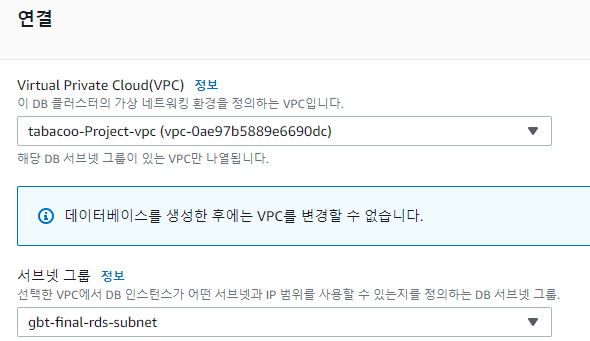
****

****

**데이터 베이스이름과 마스터 사용자 이름, 암호를 설정합니다**

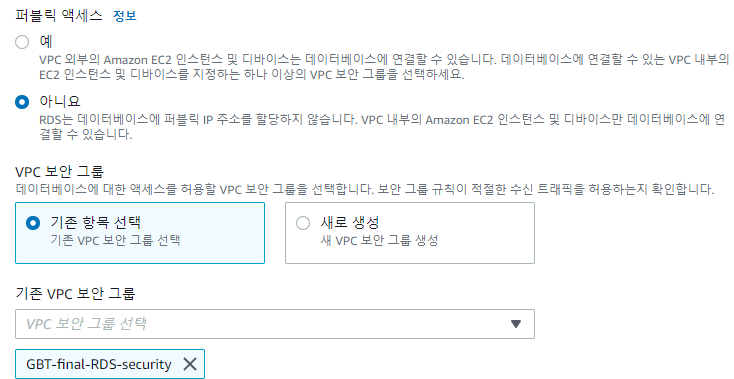
****

****

Backend와 통신하기 위해 backend와 동일한 vpc로 설정합니다 ****

외부의 접근은 차단하기 위해 퍼블릭 액세스는 아니오로 선택하고

보안그룹은 RDS용으로 만들어 두었던 보안그룹으로 선택합니다

****

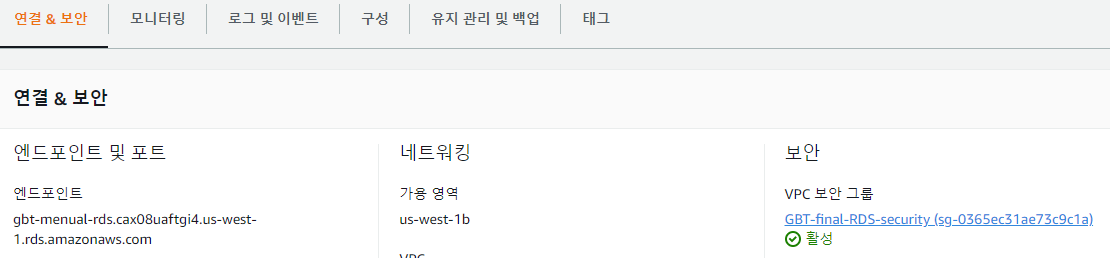
backend와 Mysql을 연결 하기 위해 데이터 베이스인증 방법을 암호인증으로 설정합니다

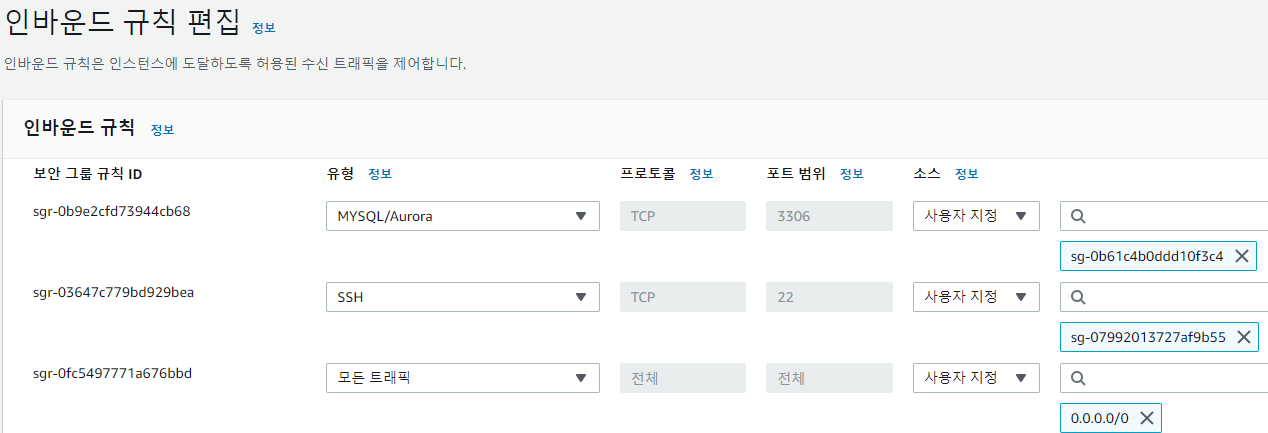
****

데이터 베이스를 생성합니다

****

Backend와 DB의 연결을 우분투에서 해주기 위해 보안그룹의 인바운드 규칙을 편집합니다

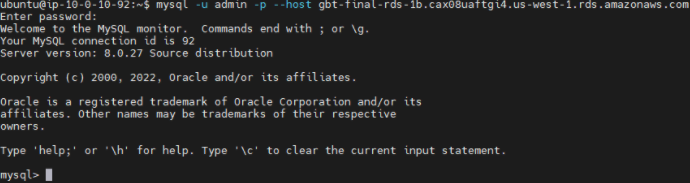
****

****

Ec2에서 RDS엔드포인트로 접속합니다

****

$ mysql -u {마스터사용자 이름} -p -h {RDS 인스턴스 엔드포인트} 를 입력하고 패스워드 입력하면 접속이 된 것을 확인 할 수 있습니다

****

**9. 안드로이드 CI**

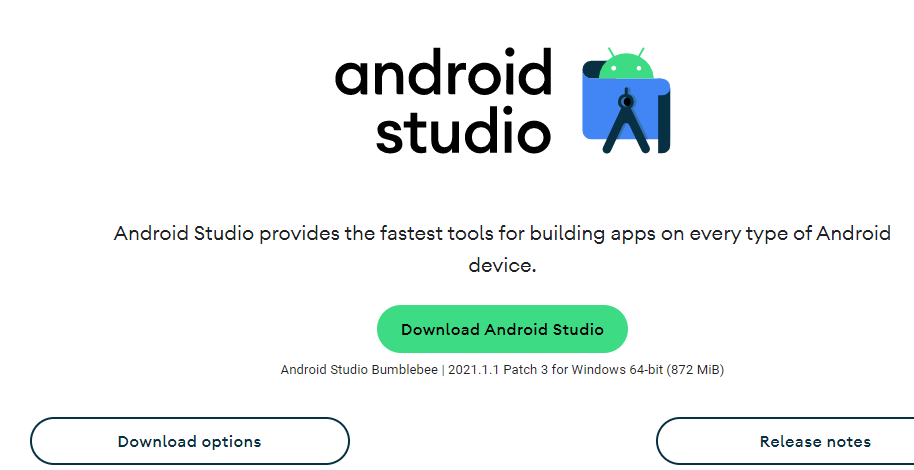
젠킨스 서버로 들어갑니다

****

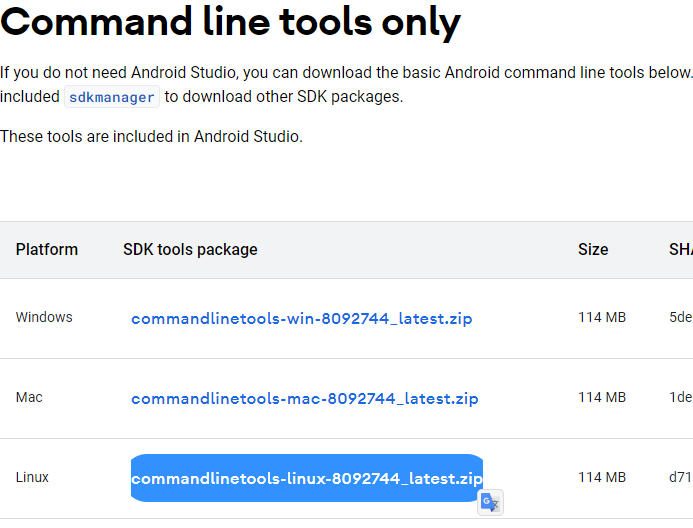
Android-sdk디렉토리 생성후 디렉토리로 들어갑니다

****

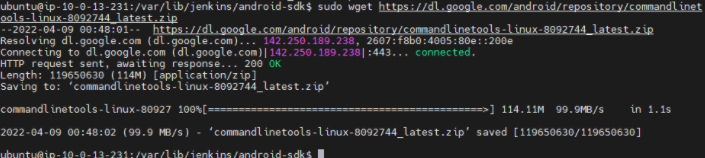
Command line tools를 설치하기 위해 android 스튜디오에 들어갑니다

****

맨 하단에 command line tools에 linux sdk manager를 설치합니다

****

$sudo wget http://dl/google.com/android/repository/위의 zip 패키지

****

그후 zip파일을 unzip 합니다 ( unzip 다운 : sudo apt install unzip)

****

압축 해제 후 생성되는 최상위 폴더가 변경됨에 따라 경로 이슈가 발생할 수 있기 때문에sdkmanager 절대 경로를 설정 합니다

$export ANDROID\_HOME = var/lib/Jenkins/android-sdk/cmdline-tools/latest/bin/sdkmanager

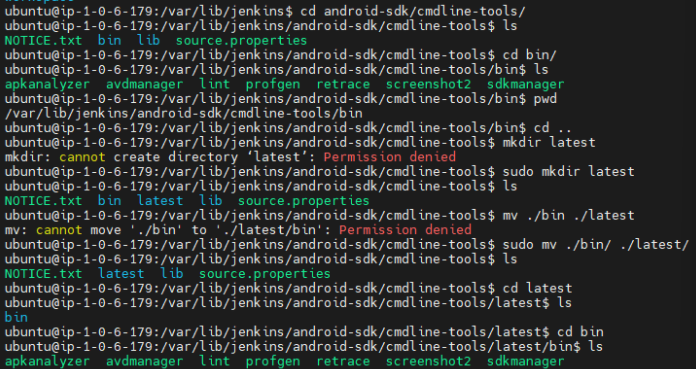
****

Echo 로 경로 확인

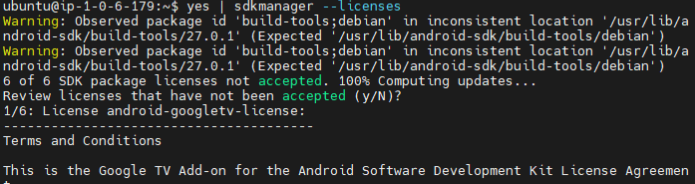
****

지정한 경로 대로 디렉토리를 생성 합니다

Cmdline-tolls에 latest 디렉토리가 없기 때문에 생성해주고 bin파일을 latest로 이동시켜줍니다

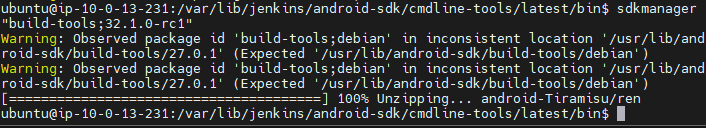
****

그리고 sdkmanager가 있는 디렉토리로 이동하고 라이선스 동의를 합니다

****

Sdkmanager가 있는 디렉토리로 들어가서 빌드 툴즈 설치

****

****

생성한 안드로이드 파일을 올릴 깃허브의 레포지토리와 젠킨스를 연결합니다 이전과 방법이 같으므로 생략하겠습니다

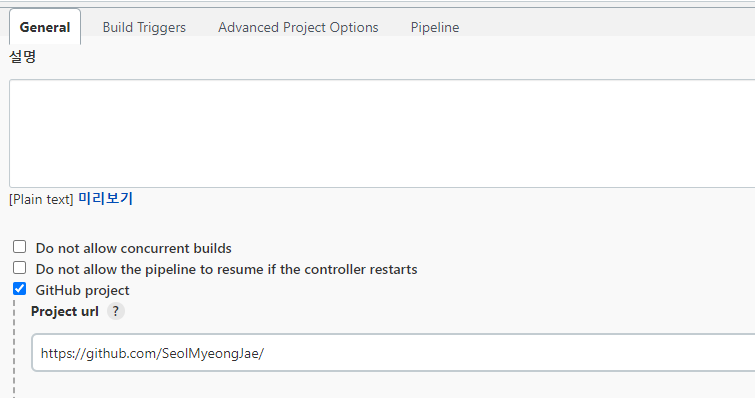
젠킨스에 안드로이드 파이프 라인을 입력합니다

****

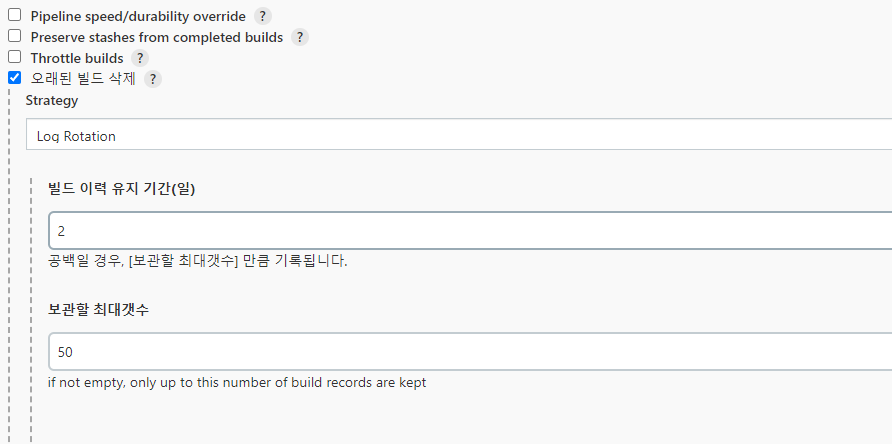
**10. 통합파이프 라인**

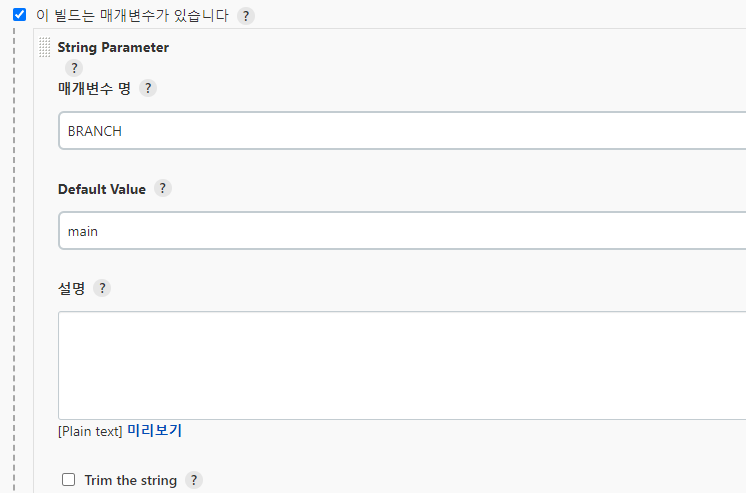
이제 backend와 안드로이드가 github에 push되면 하나의 파이프 라인이 작동하여 backend와 안드로이드가 각각 build가 되도록 통합 파이프 라인을 만듭니다

젠킨스에 파이프라인을 새로 생성하고 설정합니다

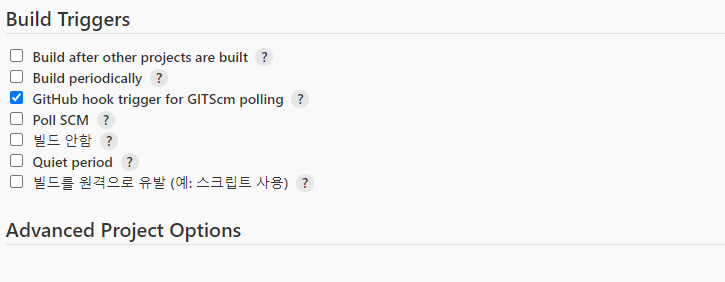


안드로이드를 빌드시 용량으로 인해 피해를 받지 않게 오래된 빌드는 삭제가 되도록 설정해줍니다





Github에 push 되면 반응하여 파이프 라인이 작동하도록 Github hook trigger을 체크합니다



**통합파이프 라인 Script는 backend와 안드로이드의 파이프라인을 하나로 합쳐 stages에 추가 했습니다**

****

****