Dongkuk University x Cloud4C:

Cloud4C Academy

2021 public cloud group project

Group4 Final Report

제출 날자 : 2021. 06. 15

4팀 팀원 별 기여도

이름	학번	주역할	기여도(합 100%)	비고
고승렬	2017112172	*자료 조사 및 보고서 작성*	25%	
박성혁	2015112255	*아키텍쳐 구현 및 aws 설계 위한 이론 취합*	25%	
유방현	2016112200	*시스템 아키텍쳐 설계 및 구현* *자료조사* *보고서 작성*	25%	

최시은	2019112099	*아키텍쳐 구현 및 aws 설계 위한 이론 취합 발표*	25%	

Table

I. 프로젝트 목표 및 방향

1. 프로젝트 목표

AWS 환경을 사용한 퍼블릭 클라우드 시스템을 이해하고 클라우드 산업 실무에 이용되는 아키텍트의 구축 및 운영 환경 관련 실무를 이해한다. 이를 기반으로 클라우드의 기본 동작 구조와 체계를 이해하고 동일하게 작동 가능한 시스템을 구현한다.

2. AWS의 목적 및 필요성

AWS(Amazon Web Service)는 Amazon EC2를 이용해 클라우드에서 확장 컴퓨팅 용량을 제공하고 원하는 만큼 가상서버를 구축하거나 보안, 네트워크 구성 및 스토리지 관리가 가능하게 해준다. 클라우드 서버의 데이터가 점점 많아지고 속도가 빨라짐에 따라 기업이 서버와 장비를 유지 및 관리하기보다는 클라우드 서버를 이용하여 장비의 유지, 보수 없이 서비스에 대한 이용료만 지불하는 것이 훨씬 이상적이고 효율적인 방안이 되었다. 그에 따라 AWS 연구의 필요성과 중요도 또한 상승했다.

3. 관련 사례

AWS는 AI, 머신 러닝, quantam 등의 혁신적인 기술을 사용하여 데이터 파이프라인을

분석해 효율적으로 연구를 진행할 수 있게 해준다. 보험중개사 에이온에서는 보험 상품의 가치 평가, 관리를 위해 AWS의 GPU 최적화 인스턴스를 이용하였고 그 결과, 총 소요 시간은 1400배 이상 단축되었다. 또 대표적인 콘텐츠 서비스 제작기업인 넷플릭스에서는 AWS에서 100,000개 이상의 서버 인스턴스를 두고 데이터베이스, 분석, 추천 엔진 등을 사용해 특정 사용자가 관심있을 만한 콘텐츠를 제공해준다.

II. 구현 요소

1. VPC

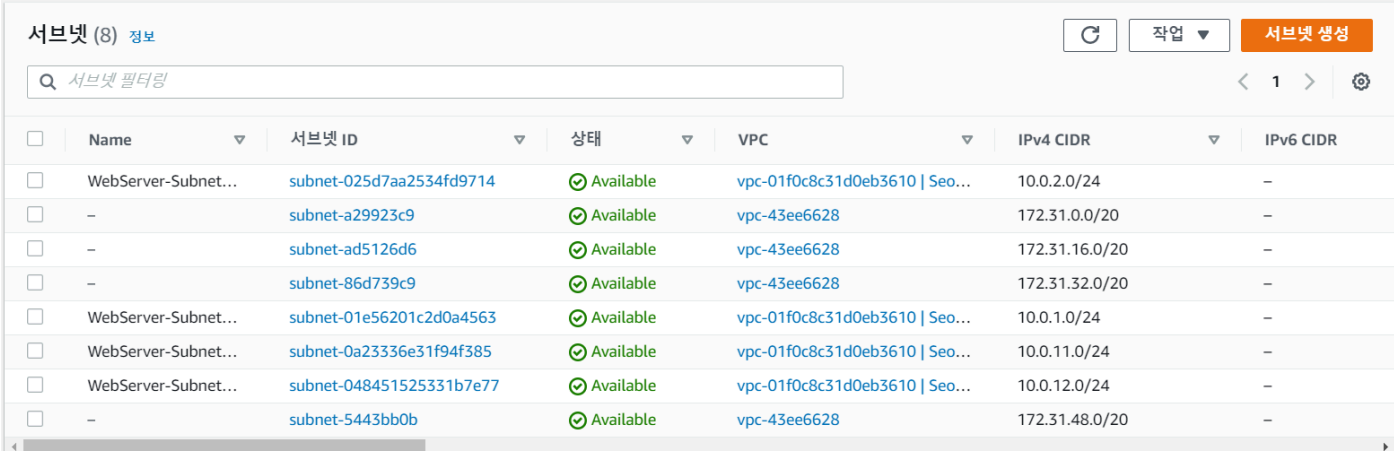
vpc는 클라우드 환경을 생성하는 데 가장 기본이 되는 리소스이며 하나의 리전에 종속된 vpc는 독립적이다.

vpc를 구성하기 위해서 필요한 요소로는 서브넷, 라우팅 테이블, 인터넷 게이트웨이, 엔드포인트, CIDR 블록 등이 있다. VPC를 구성하기 위해 다음의 과정들을 거쳤다.

-Subnet

서브넷은 vpc보다 작은 단위로 물리적인 리소스가 생성되는 가용 영역(available zone)과 연결되며 vpc 내부의 리소스가 생성될 수 있는 네트워크이다.

IPv4 CIDR은 사설 IP 대역 10.0.0.0을 사용하였고 할당 가능한 ip의 개수는 masking에 16을 사용하여 64K개만큼 할당해주었다. webserver-subnetpublic1, 2와 webserver-subnetprivate1, 2 총 4개의 서브넷을 생성하였다.



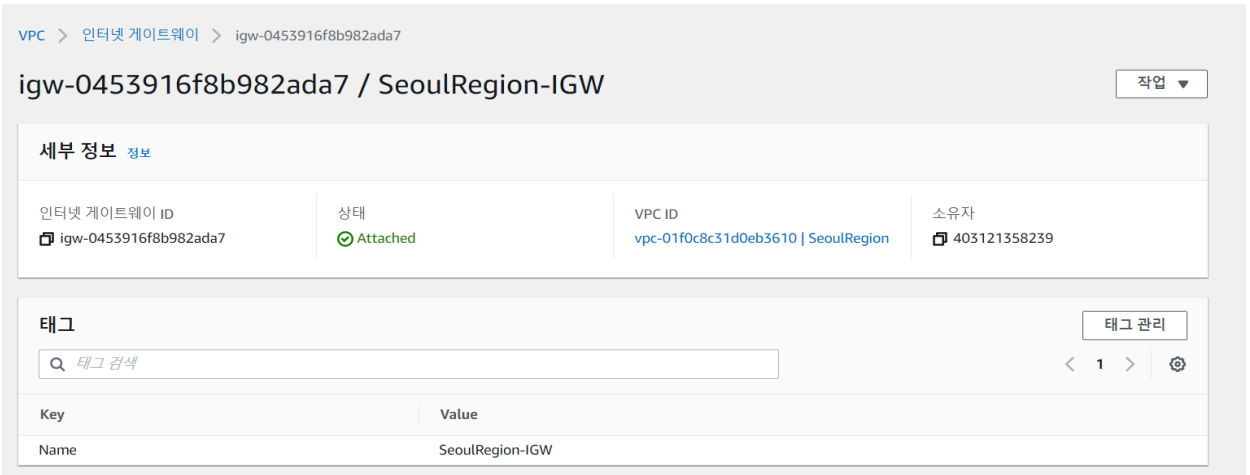
-Routing Table

라우팅 테이블은 네트워크에서 destination address를 네트워크 노선으로 적용시키는데에 이용된다. 라우팅 테이블의 각 노드에는 패킷을 목적지로 보내는 최적의 경로에 대한 정보가 포함되어 있으며, 각 노드에서는 패킷의 데이터를 확인하고 최적의 경로를 설정해준다. 라우팅 테이블 생성 과정에서는 앞에서 생성한 SeoulRegion vpc를 공간으로 선택하여 private용 라우팅 테이블과 public용 라우팅 테이블을 생성하였다. priavte 라우팅 테이블에는 private subnet 1, 2를 연결하고 public 라우팅 테이블에는 public subnet 1, 2를 연결하였다.

-Internet Gateway

IGW는 네트워크 간에 통신을 가능하게 해주고 다른 네트워크에 접근하기 위해서는 반드시 거쳐야 한다. vpc와 subnet을 만들고 subnet 내부에 인스턴스를 생성하여도 인스턴스에 접근하기 위해서는 vpc에 IGW를 연결하고 subnet의 라우팅 테이블이 IGW를 가리키도록 설정해야 한다.

vpc에 IGW를 연결하기 위해 먼저 IGW를 생성하였다. 생성한 IGW를 앞에서 생성한 SeoulRegion vpc와 연결하고 IGW의 이름을 SeoulRegion-IGW로 하였다.



두 번째로, subnet의 라우팅 테이블이 IGW를 가리키게 하기 위해 앞에서 생성했던 라우팅 테이블 WebServer-RouteTable-public의 라우팅 편집을 하였다. 목적지 대상은 0.0.0.0/0으로 설정하고 타겟 대상에는 SeoulRegion-IGW의 id 값을 넣었다. 0.0.0.0/0은 모든 IP를 의미하므로 두 번째 추가한 규칙은 VPC CIDR 접근 범위 이외의 모든 IP를 IGW로 라우트해주는 규칙이다.

2. RDS 구축 및 연결

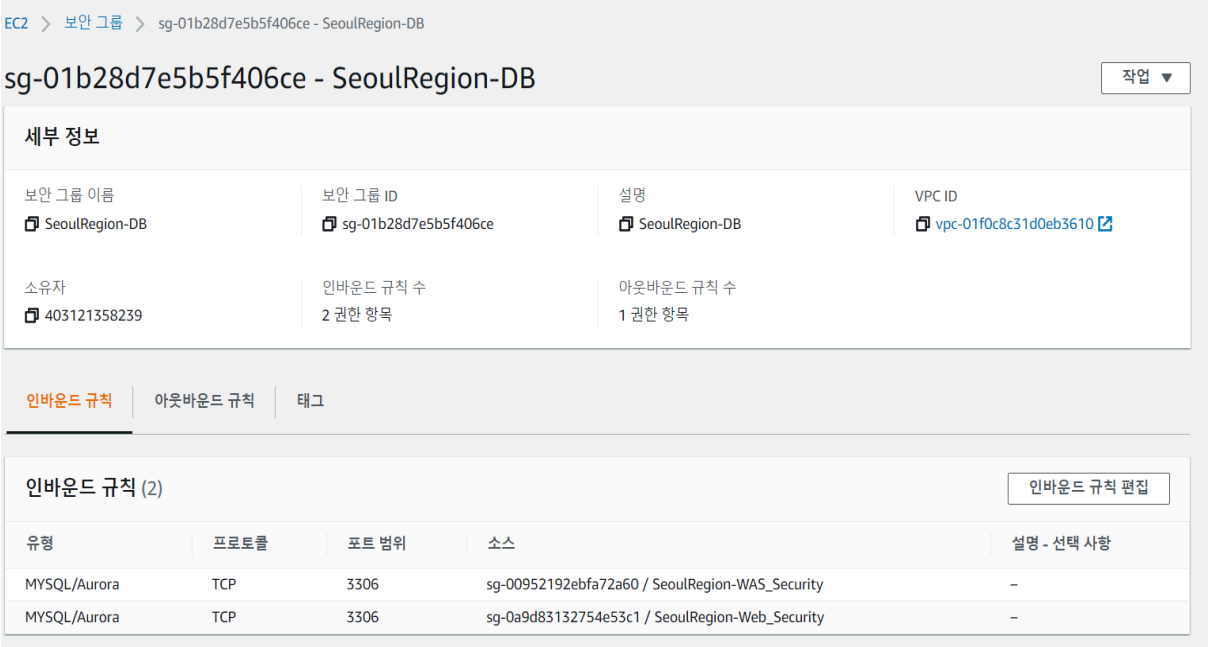
Amazon에서 제공하는 데이터베이스 중 RDS를 선택하여 사용했다. Amazon RDS는 보안성과 확장성이 뛰어나 인스턴스와 스토리지에 대해 유연한 대응이 가능하고 다양한 CPU/Memory 옵션을 제공하여 스토리지를 필요에 따라 확장시킬 수 있다. 또 자동 백업 기능이 있어 손실된 데이터를 쉽게 복구할 수 있으며 백업된 snapshot을 통해 데이터베이스를 생성할 수도 있다. 또한 멀티 AZ 기능을 가지고 있어 다른 AZ에 데이터를 저장하여 안정성을 높여준다. 이렇게 AZ간 동기화 구성이 가능하기 때문에 장애나 재난 대처에도 효과적이며 데이터 이전이 비교적 쉽다. 또한 Amazon RDS 엔진의 저장 및 전송 중 암호화 기능을 활용하면 네트워크에 대한 접근을 쉽게 제어할

수 있어 보안성이 뛰어나다.

RDS를 EC2에 연결하기 위해 먼저 보안그룹을 생성했다.

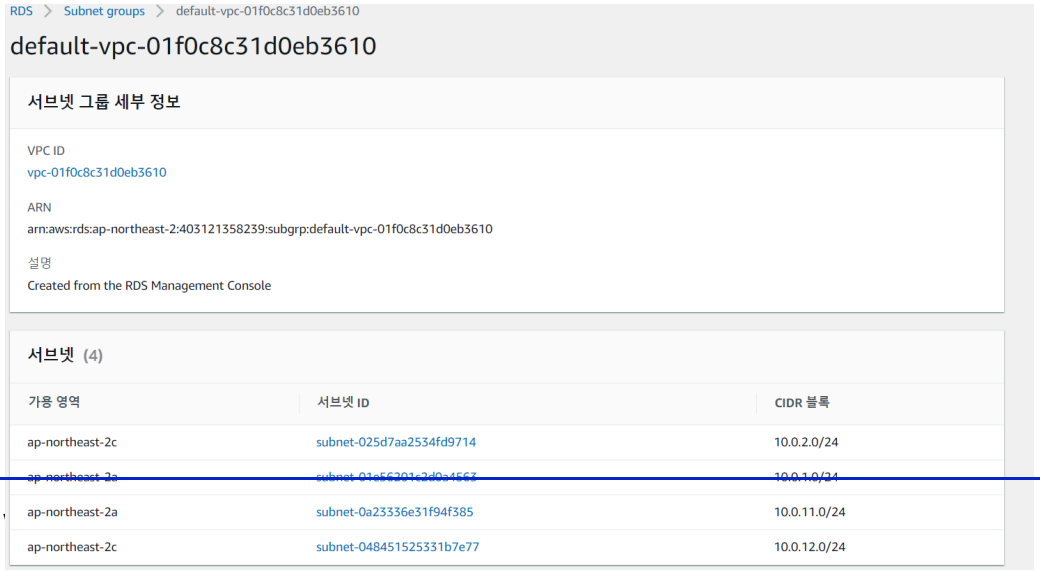
-보안 그룹 생성

보안그룹은 SeoulRegion-DB를 생성했다. 보안 그룹의 인바운드 규칙에는 지정된 EC2의 인스턴스에서만 접근할 수 있도록 규칙을 추가했다.



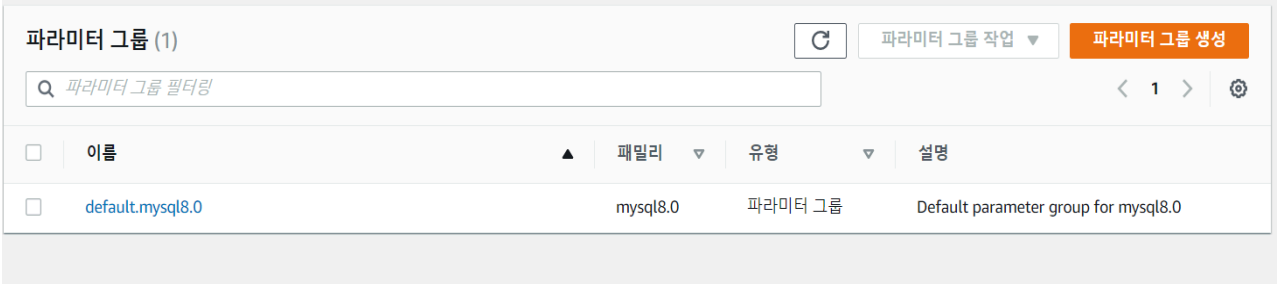
-서브넷 그룹 생성

EC2와 동일한 VPC ID를 선택하고 관련된 모든 서브넷을 추가해 서브넷 그룹을 생성하였다.



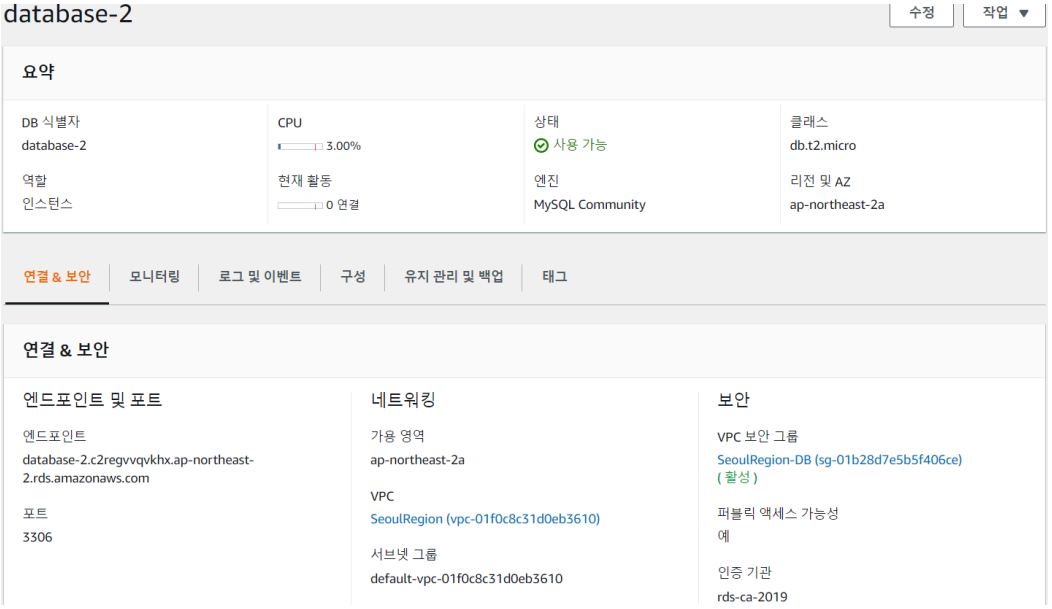
-파라미터 생성 및 설정

MuSQL 8.0버전의 파라미터 그룹을 생성하고 캐릭터 인코딩 값들을 모두 UTF8로 변경했다. 또 collation의 값들을 UTF8_general_ci로 변경했다.



-데이터베이스 생성

데이터베이스 생성 탭에서 MySQL community 엔진을 선택하고 free tier에서 사용 가능한 t2.micro 클래스를 사용하였다. 서브넷 그룹은 SeoulRegion을 선택하고 보안 그룹은 이전에 생성한 SeoulRegion-DB를 선택했다. 파라미터는 앞에서 생성한 파라미터 그룹을 선택하여 데이터베이스를 생성했다.



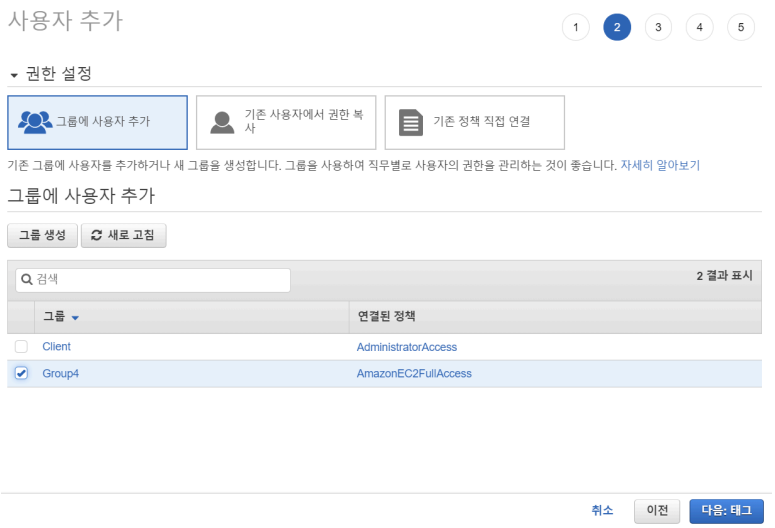
4. IAM 사용자 설정

dgunigrp4(솔류션 아키텍쳐) 계정에게는 Root 계정에 준하는 AWS FULL ACCESS 권한을 부여함으로서 AWS의 모든 리소스에 접근 가능하게끔 하였다.

Company(회사) 계정의 경우 IAM 그룹(Group4) 안에 계정을 생성하게 되었다.

IAM 그룹으로 IAM 사용자들을 모아 관리를 하게 될 경우 IAM 그룹에 접근제어 및 권한설정을 할 수 있으며 IAM 그룹에 설정된 내용은 IAM 그룹 안에 포함된 모든 사용자들에게 적용되기에, 관리가 편하기 때문에 설정하게 되었다.

Group4의 권한을 AmazonEC2FullAccess를 부여함으로써, Group4에 속해있는 IAM사용자들은 이 권한을 따르게된다.



Group4에 IAM 사용자(dguunigrp4)를 추가하게 되었다.

사용자(IAM) 이름 설정 및 엑세스 키 ID및 비밀 엑세스 키 활성화를 해야 하는데, 엑세스 키의 경우 분실시 사용자를 다시 만들어야 하므로 잘 보관해둔다.

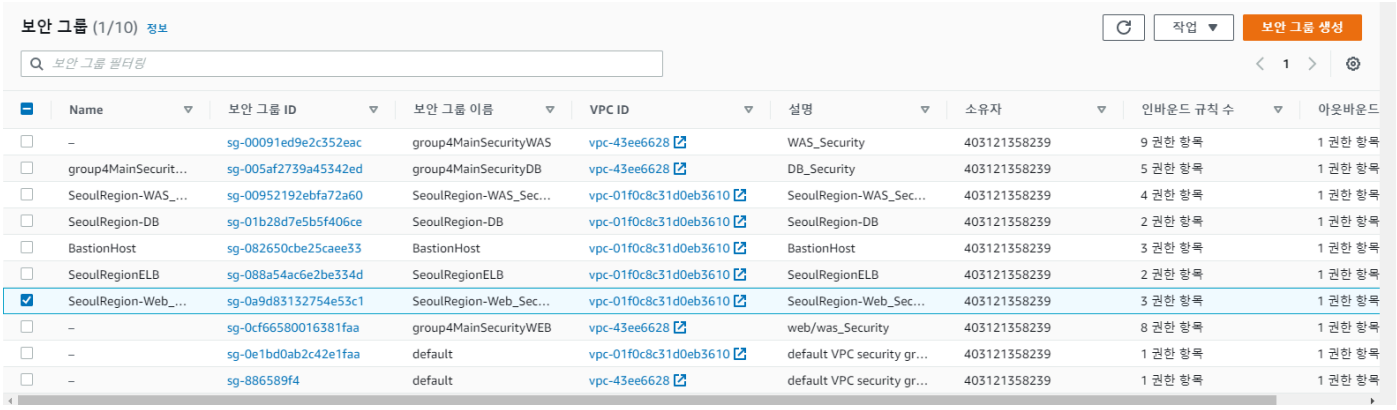
5. WEB Server Tier

추가하게WEB SERVER 의 경우 한 서버가 다운될시를 대비하기 위하여 두개의 서버를 구성하기로 하였다. 그중 가용영역 C에 있는 서버는 가용영역 A에 생성할 서버의 AUTO SCALING 을 통해 생성하기로 하였다.

보통 서버의 경우 크기를 크게해야 여러 트래픽들을 정리하는데 용이하다. 하지만, 우리는 프리티어 내에서 사용해야만 했으므로 프리티어의 최대한도(30GiB)보다 낮은 8Gib 의 서버를 생성하기로 하였다.

이렇게 생성된 서버에 접속하기 위해서는 SSH(Secure Shell)를 사용하게 된다.

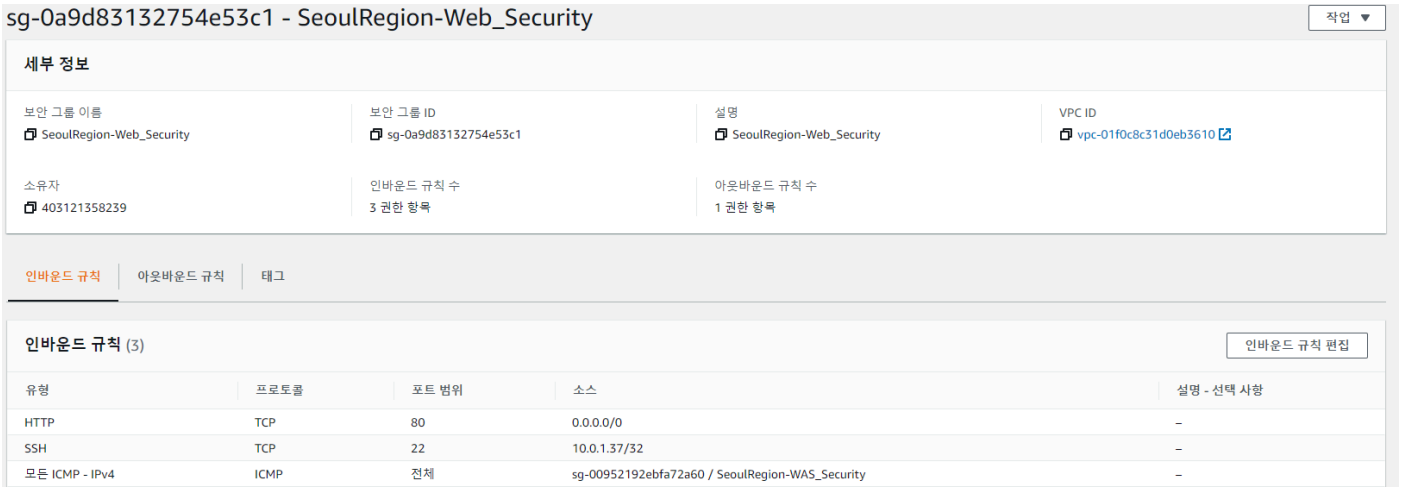
Windows의 경우 putty를 사용하게 된다.



이후 Security Group을 활용해 EC2 인스턴스 즉 웹서버에게 방화벽 설정을 하게 된다. 보안그룹을 생성하고 Security Group Inbound 규칙란에서 HTTP의 경우 위치와 무관하게 모든 IP가 접근할 수 있게끔 하였다. 즉, 2개의 Security Group이 형성 되게 되는데, 이부분은 쇼핑몰 사용자들이 WEB SERVER에 접근 할경우 HTTP 즉 , 홈페이지만 접근할 수 있게끔 구현 한것이다. 반면 SSH , 즉 서버에 직접 접근하여 서버의 속성을 변경 할 수 있는 IP는 솔루션 아키텍쳐의 IP만 부여함으로서 다른 IP는 접근이 불가 하도록 하였다.

또한 ICMP - IPV4 의 주소의 경우 같은 가용영역 내의 WAS Security Group과의 상호통신을 확인하기 위해 같은 가용영역내의 WAS SECURITY GROUP의 주소를 설정하게 되었다.

되었다. 따라서 Security Group 의 경우 Auto Scaling을 활용하여 생성된 가용영역 C에 위치한 WEB SERVER와 본래 생성된 가용영역 A의 WEB SERVER와 한쌍으로 SeoulRegion-WEB_Security, WAS SERVER 한쌍씩 SeoulRegion-WAS_Security 그룹으로 지정하였다.



6. WEB Application Tier

WEB Application Server(WAS)의 경우 WEB SERVER와 마찬가지로 가용영역 C에 속해 있는 PRIVATE SUBNET에 가용영역A에 서버를 먼저 생성한 후 AUTO SCALING을 통해 한개 더 생성하게 되었다.

WAS SERVER도 마찬가지로 SECURITY GROUP을 형성할때 인바운드 규칙의 경우 SSH 설정은 WEB SERVER와 동일하고, ICMP-IPV4의 경우 WEB SECURITY GROUP의 주소를 설정하게 되어 마찬가지로, 통신을 위해 설정하게 되었다.

WAS 서버의 경우 USER의 경우 접근 할 이유가 없다. WAS에서 제공하는 APPLICATION을 WEB SERVER로 옮긴 후 USER에게 접근이 되는 것이기 때문에 HTTP에 대한 접근 권한은 부여할 필요가 없게 된다.

7. WEB, WAS, DB에 대한 연결 테스트

WEB, WAS가 속해있는 Private Subnet과 DB가 속해 있는 Private Subnet이 서로 연결이 되어야만 한다. 이를 확인하기 위해서는 DB Tier에 MySQL 클라이언트를 설치한후 MySQL 명령어로 DB End Point로 접속이 완료되면 각 Subnet과의 연결이 확인되는 것을 볼 수 있다.

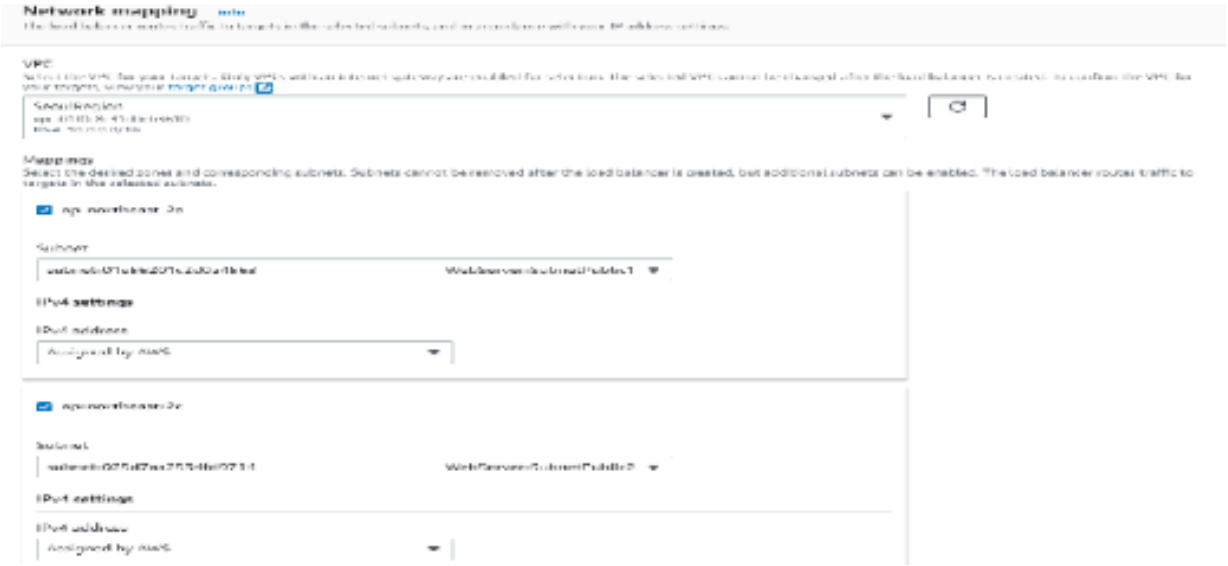
이는 시행착오 부분에서 웹 서버를 이용하여 RDS연결 하는 과정을 보일 것이다.

III. 추가 구현 요소

1. Load Balancer

서버의 역할중 하나는 서버를 이용하는 고객들의 요청을 응답해주는 것이다. 고객의 수가 적을 때는 서버가 모든 고객의 요청을 들어줄 수 있지만, 고객의 수가 급격히 많아지면 서버는 과부하가 걸려 동작을 멈추게 된다. 이를 해결하기 위해서는 하드웨어의 성능을 향상시키거나(scale-up) 서버를 여러 개 생성하는 방법(scale-out)이 있다. 하드웨어를 향상시키는 것보다는 서버를 여러 개 생성하는 것이 비용적인 측면에서 더 효과적이고 자연재해에 의한 피해 예방에도 효율적이기 때문에 확장성이 좋은 scale-out 방식을 선호한다. 이때 생성한 여러 개의 서버에 트래픽을 분산시키는 역할을 하는 것이 로드밸런서이다. 즉, 로드밸런싱은 여러 서버에서 트래픽을 분산 처리하여 서버의 부하, 속도 등을 고려해 처리해주는 것이다.

-Load Balance 생성



NLB로드밸런스를 생성하기 위해 이름은 Web-NLB로 설정하고 VPC영역은 생성했던 SeoulRegion VPC를 사용, Subnet은 SubnetPublic1, 2를 선택했다.

가용영역으로는 ap-northeast-2a와 2c를 사용했다. 사용할 VPC, Subnet, 가용 영역을 선택함으로써 로드밸런서에서 트래픽을 라우팅할 VPC와 Subnet, 가용영역을 지정해주었다.

Target Group으로는 NLB인스턴스를 생성하여 사용했다.

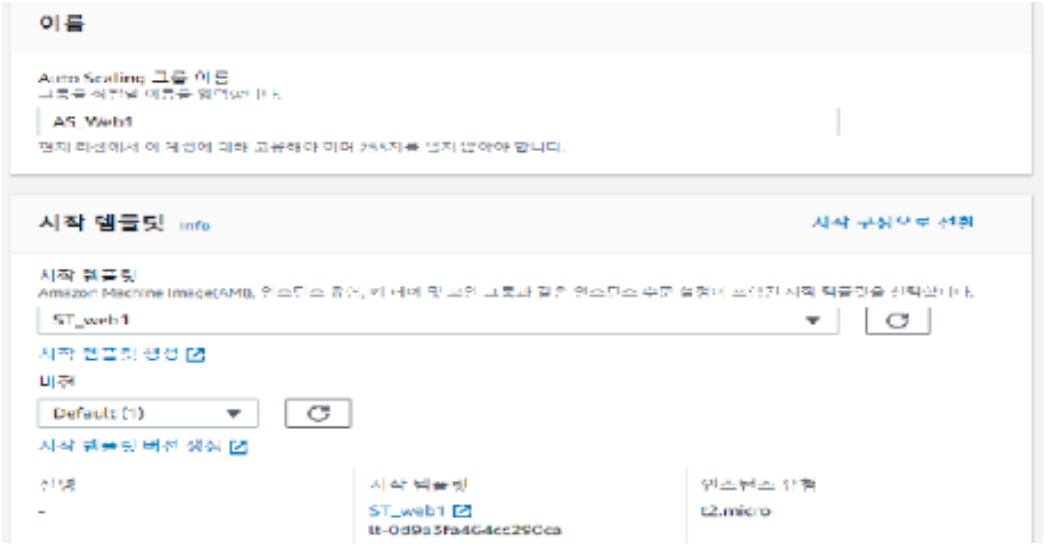
생성한 네트워크 로드밸런서를 확인해보면 target group NLB로 연결 하고 있는 것을 확인할 수 있다.

2. Auto Scaling

Auto Scaling은 사용자 정책에 따라 서버를 자동으로 생성하고 삭제해주는 서비스이다. 사용자가 많을 때는 원활한 서비스를 위해 서버를 늘리고 사용자가 적을 때는 서버를 줄여 불필요한 서버 관리로 발생하는 비용을 줄여준다. 이러한 Auto Scaling의 특성을 제대로 활용하기 위해서는 여러 개의 서버에 트래픽을 자동 분산 시켜주는 Load Balancer에 연결하여 사용해야 한다. 따라서 Auto Scaling을 구현할 때 Load Balancer와 연결하여 구현하였다.

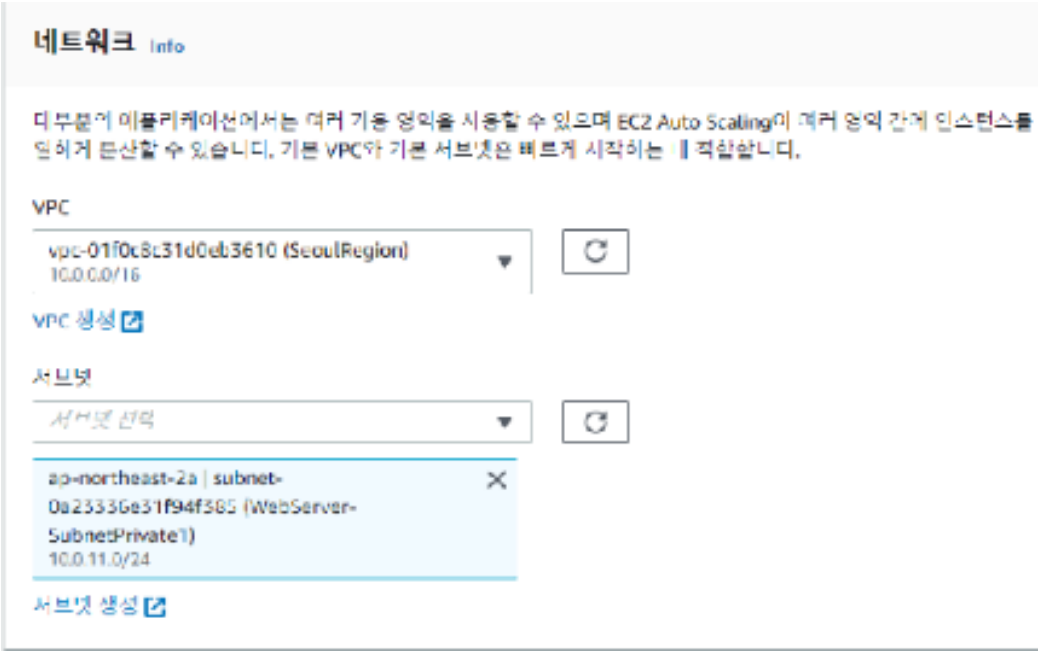
-Auto Scaling 그룹 생성

먼저 Web1 인스턴스를 Auto Scaling 해주었다.



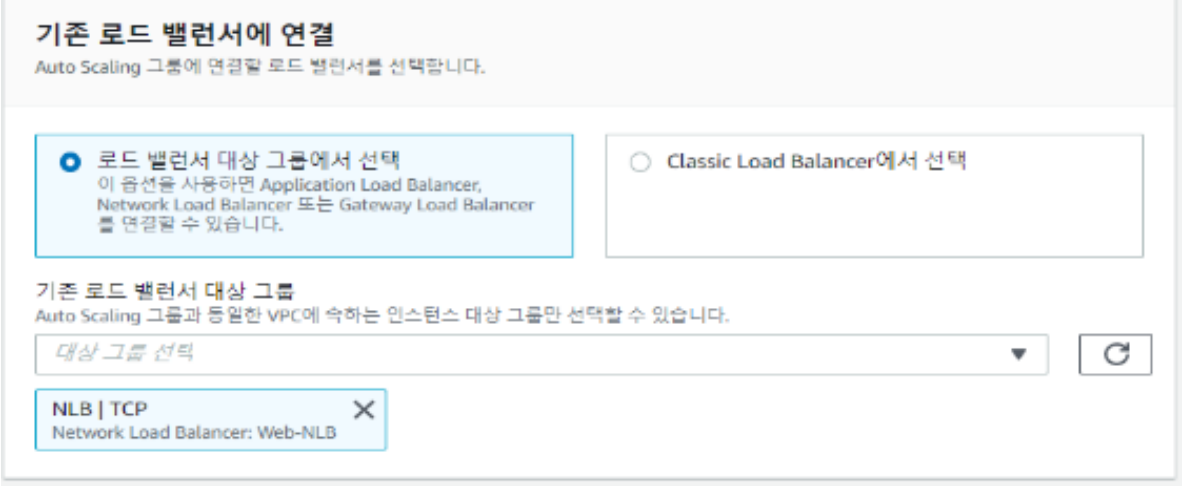
Auto Scaling 그룹 이름을 AS_Web1으로 설정하고 시작 템플릿은 web1의 시작 템플릿인 ST_web1을 선택했다.

-네트워크 설정

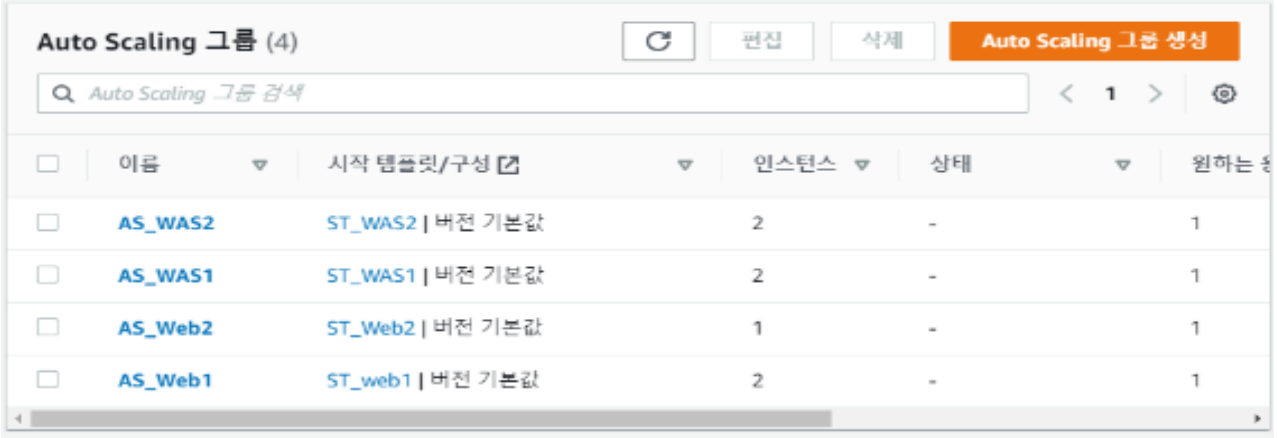


VPC는 SeoulRegion VPC를 선택하고 웹서버 서브넷인 SubnetPrivate1을 사용했다

-로드 밸런서에 연결

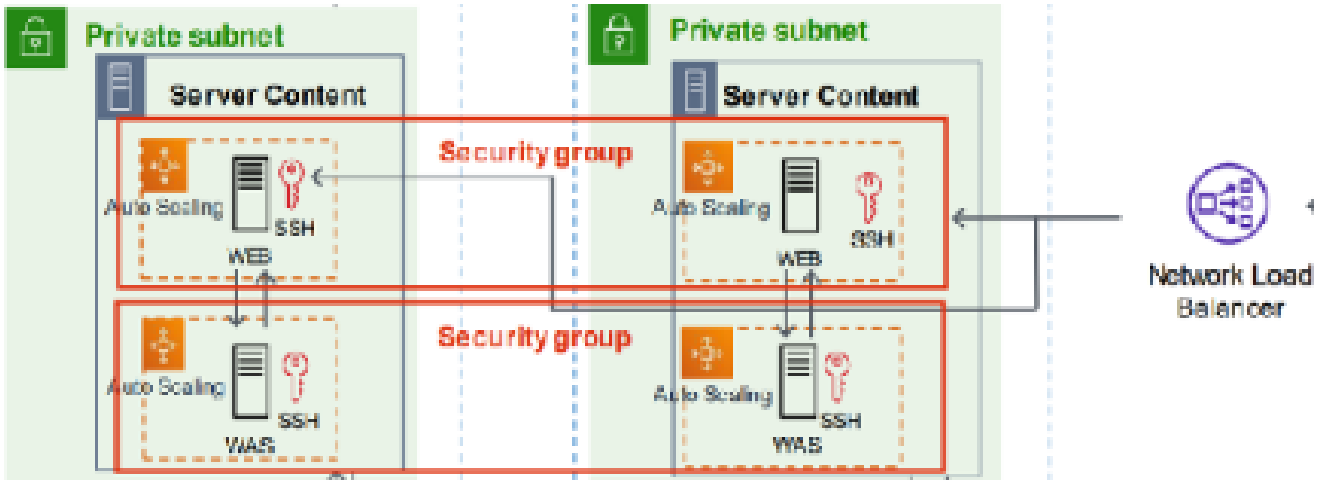


Auto Scaling의 효율을 위해 기존에 생성했던 Load Balancer와 연결해주었다. Network LoadBalancer를 생성하였으므로 로드 밸런서 대상 그룹에서 선택을 확인하고 Web-NLB Load Balancer와 연결해주었다.

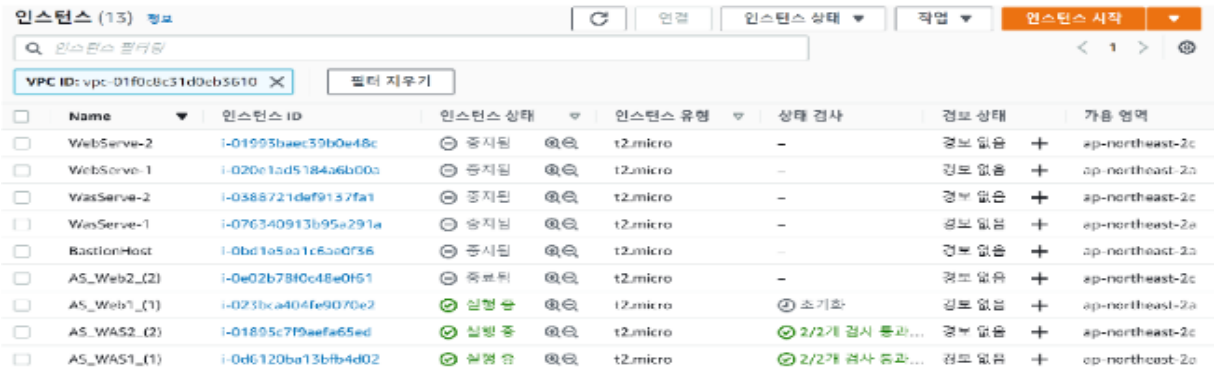


같은 과정을 Web2, WAS1, WAS2에 반복하여 총 4개의 Auto Scaling 그룹을 생성해 서버의 생성 및 삭제를 하면서 트래픽을 다중 분산 처리할 수 있도록 해주었다

.



최종적으로 로드밸런서와 오토스케일링을 사용하여 Web과 WAS의 설계가 위 사진과 같이 되도록 하였으며 AWS의 인스턴스 목록에서 확인 결과 아래 사진과 같이 오토스케일링으로 인해 규모가 자동 확장된 모습을 확인할 수 있었다.



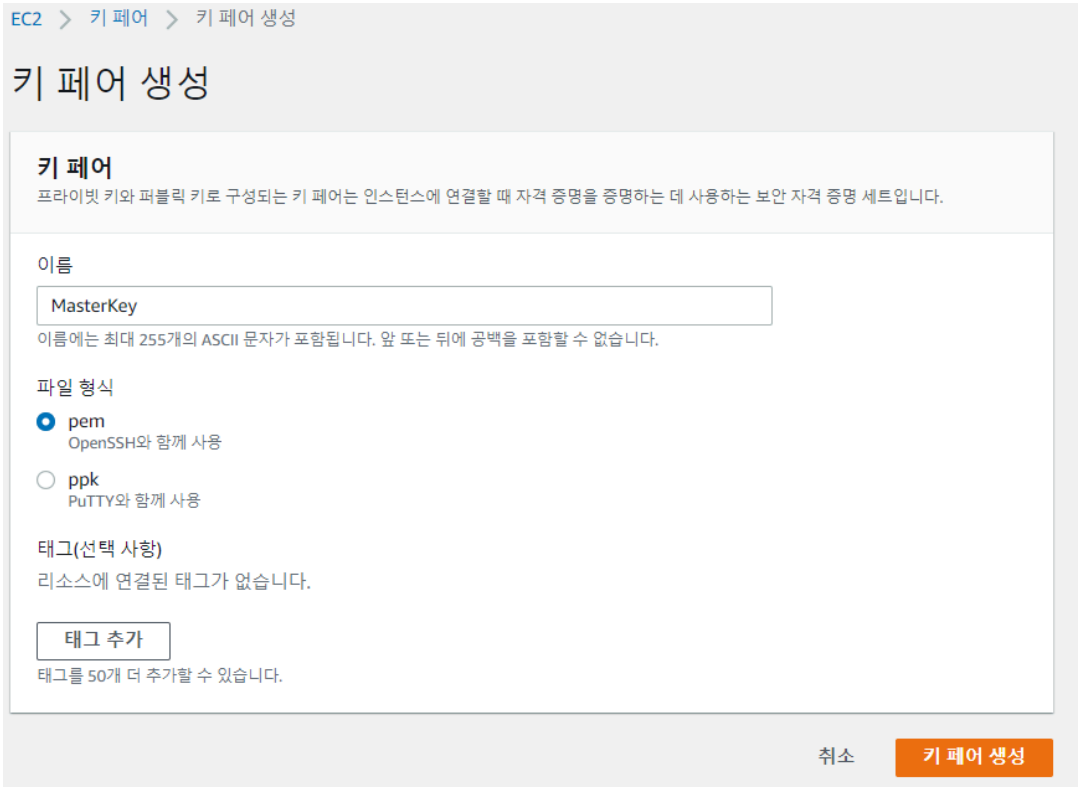
3. PuTTY를 이용한 SSH 접근

SSH는 네트워크 프로토콜의 하나로 public network를 이용해 통신할 때 데이터 전송, 원격 접속 등을 안전하게 사용하기 위해 사용한다. AWS의 클라우드 인스턴스 서비스를 이용하기 위해서 SSH가 사용되는데, SSH를 통해 원격 접속을 하거나 암호화된 통신을 가능하게 해 정보의 유출을 방지해준다.

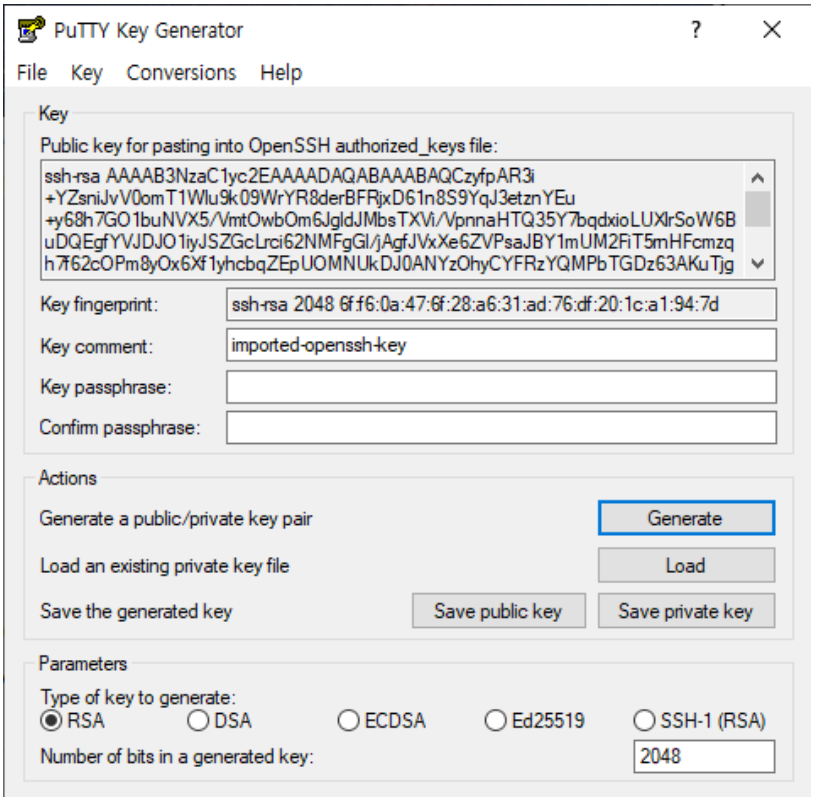
-PuTTY를 사용한 이유

PuTTY는 오픈소스이며 무료로 제공되고 프로그램의 크기가 작아 어느 환경에서나 편하게 이용할 수 있다. 또한 한글을 지원하기 때문에 인코딩도 한글로 가능해 이용하는데 수월할 것이라 생각했다. PuTTY는 SSH의 가장 유명한 프로그램이기도 하고 이러한 여러 편의성들이 도움이 될 것이라 생각해 PuTTY를 사용했다.

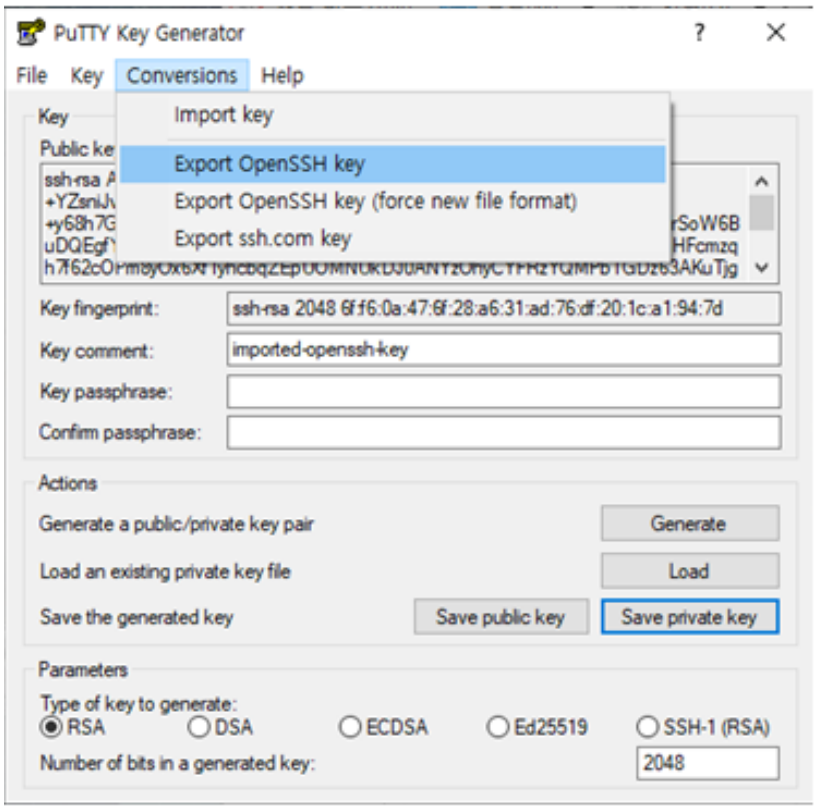
-Key Pair 생성



EC2의 네트워크 보안 탭에서 키 페어 생성 서비스를 이용해 키 페어를 생성하였다. 키 페어 생성을 누른 뒤 자동으로 다운로드 되는 MasterKey.pem을 받아 ppk 형식으로 전환하기 위해 PuTTYgen을 사용하였다.



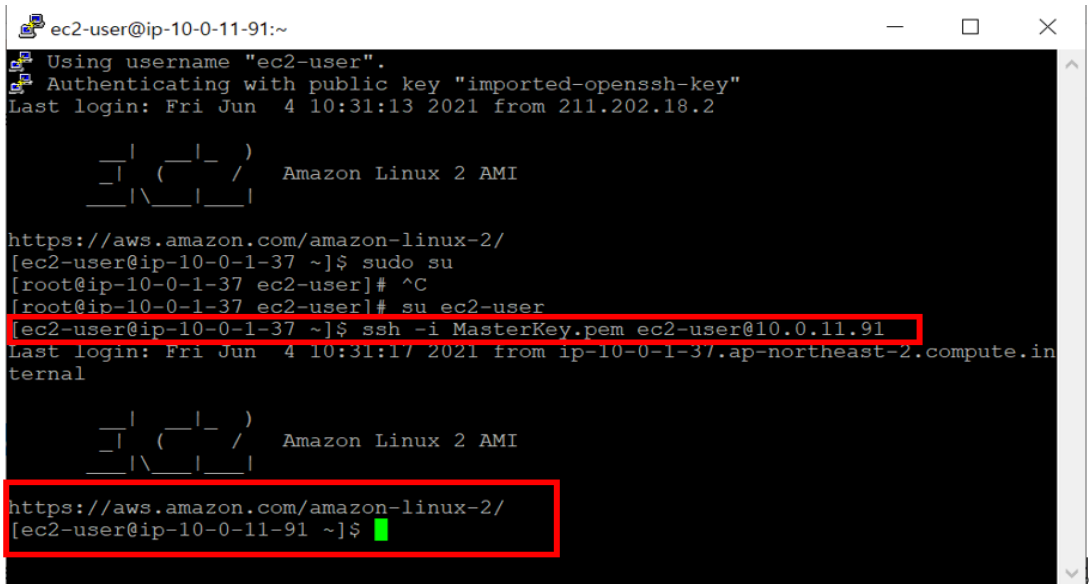
PuTTYgen에서 다운 받은 MasterKey.pem의 경로를 지정하고 Save Private Key를 눌러 ppk형식으로 저장한다. 하지만 MasterKey.ppk를 이용해 Web과 WAS에 PuTTY를 이용해 접속 시도를 했지만 실패했다. 그 이유는 Web과 WAS에 접근하기 위해서는 Bastion host를 거쳐야 하는데 Bastion host는 ppk가 아닌 pem 형식을 지원하기 때문에 ppk에서 다시 pem으로 전환하는 과정이 필요했다.



ppk 형식으로 변환할 때는 conversion 탭의 import key 메뉴를 사용했지만 pem 형식으로 변환할 때는 Export OpenSSH key 메뉴를 사용해 전환하였다.

이렇게 최종적으로 MasterKey.pem을 SSH 접속의 키 페어로써 사용하였다.

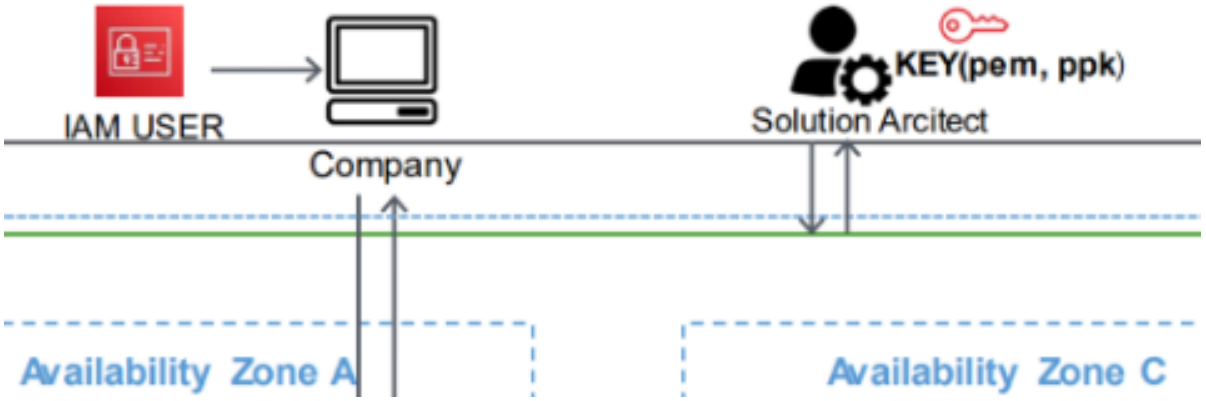
-PuTTY 연결 확인



Bastion host 인스턴스의 인바운드, 아웃바운드 규칙 수정 후 PuTTY를 통해 Private IP주소로 접속해 PrivateSubnet 내의 Webserver에 접속 가능한 것을 확인했다. 이 때 bastion host와의 연결을 위해 master.pem 키를 생성하여 사용하였다.

4. Private Key의 관리

IAM 사용자 접속 및 SSH 접근에 이용되는 Private Key의 관리를 위해 AWS Management 콘솔을 이용해 정기적으로 키를 교체하는 방법을 채택했다.



이는 키가 손상돼도 어플리케이션 작동에 영향이 미미하다는 장점이 있어 키 관리 방법으로 채택하게 되었다.

IV. 초기 구현 목표였던 아키텍처

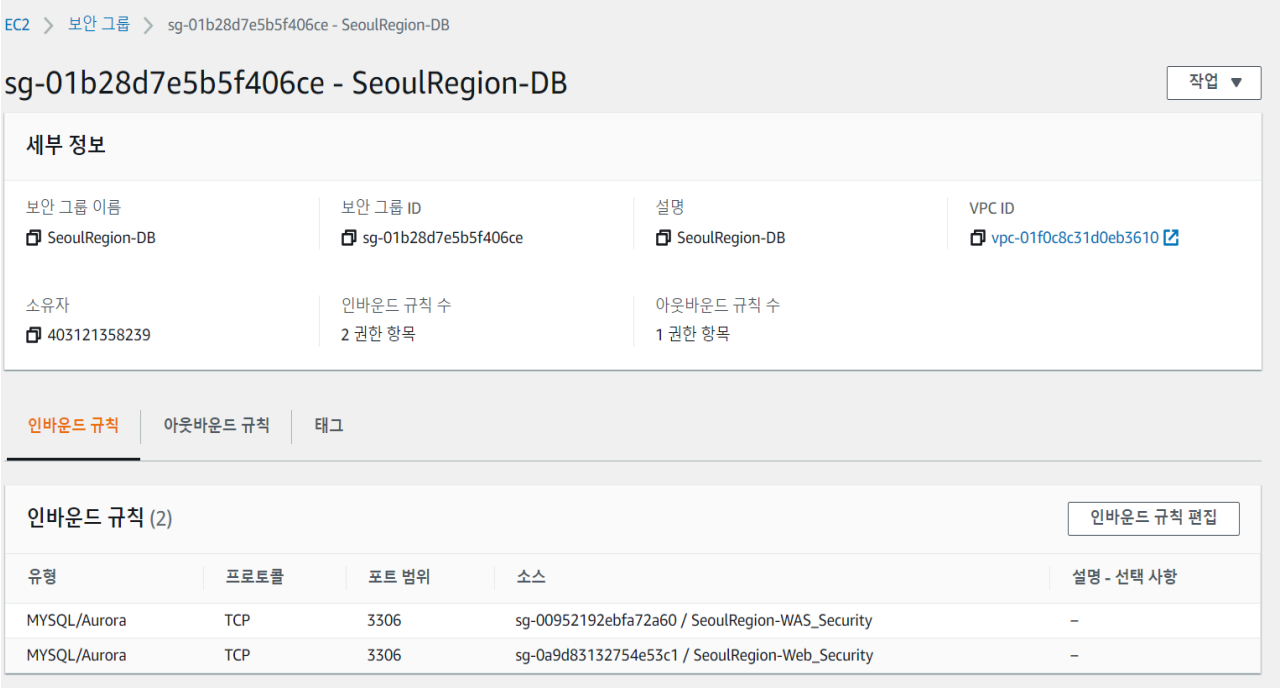
1. 시행 착오

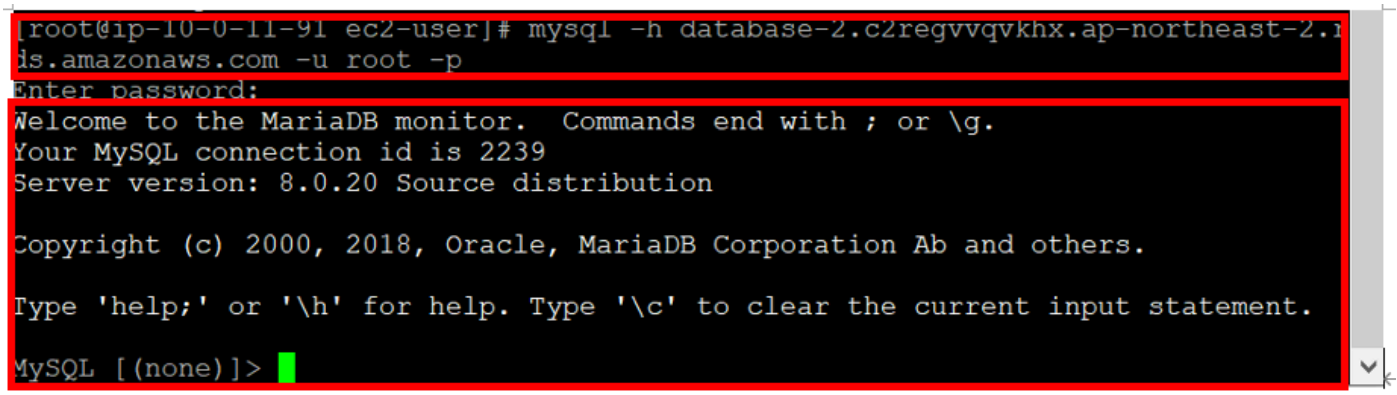
1.Webserver, Web application server, DB에 대한 커넥션 테스트 실패.

Webserver, Web application server에 서버에 접속하여 “yum install mysql”를 입력하여 각 서버에 mysql client를 설치 했다.

“mysql -h <호스트 엔드포인트> -u root -p”을 통해 RDS에 접근을 시도 하였지만, RDS 마스터 계정 비밀 번호가 잘 못 되어 접속 불가. AWS의 RDS 탭에서 database 수정을 클릭하고 마스터 암호 변경 후 Webserver에서 RDS로 재접속을 시도하였지만, 접속이 불가능 하였다.

RDS에 접속이 되지 않아, 보안그룹에 들어가 인바운드 규칙에 “MYSQL/Auroa”에 Webserver, Web application server Private IP를 입력 후에 다시 접속 하였지만, 접속 되지 않았다. 그래서 서버의 IP가 아닌 서버가 현재 사용중인 보안 그룹을 입력 “MYSQL/Auroa”에 입력하고, RDS에 재접속을 해본 결과, 정상적으로 DB와 연동이 되는 걸 확인 할 수 있었다.





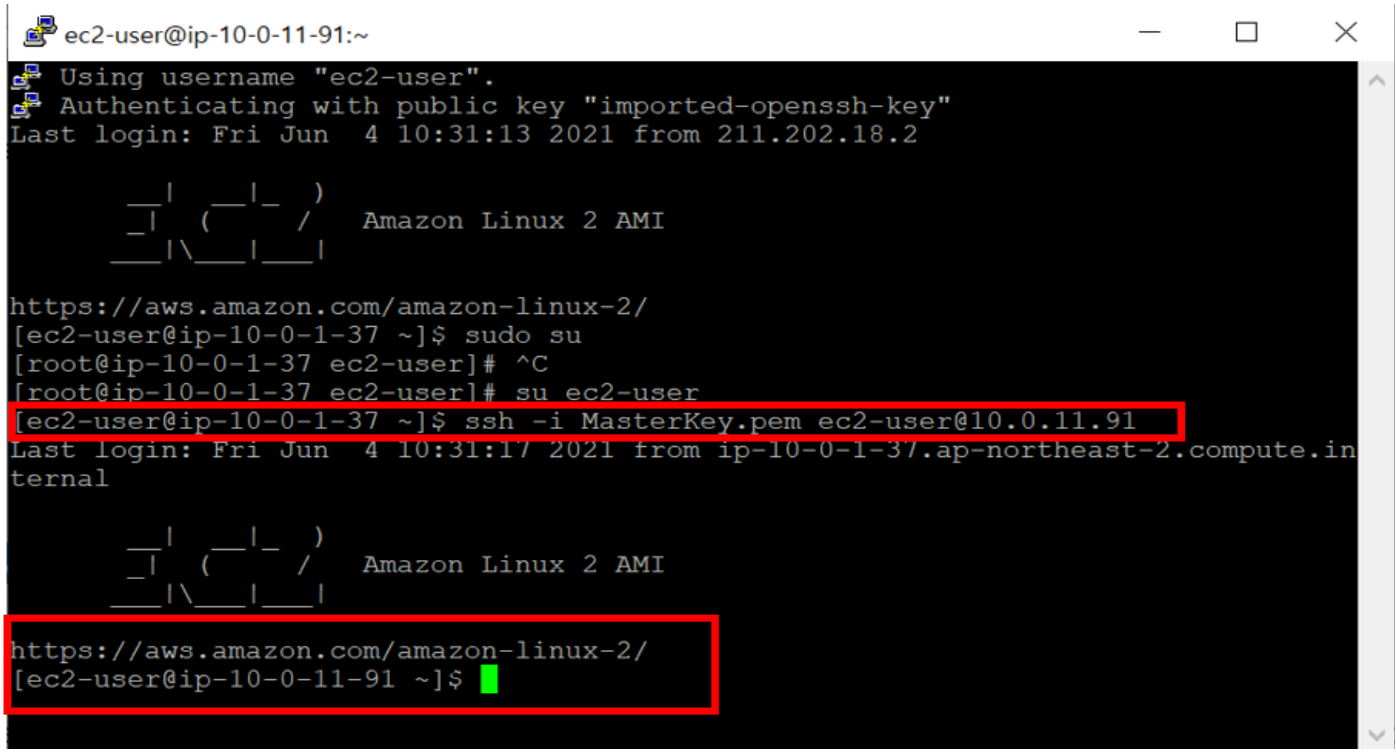
Web Server에서 RDS으로 접속 완료.

2.Bastion Host를 통해 Private Subnet 내의 호스트에 접속 불가.

Bastion Host 인스턴스를 생성하고 , Web Server, Web application server 생성 후에 Bastion Host는 Public Subnet, Web Server, Web application server는 Private Subnet에 만들어 졌기 때문에, 트래픽의 접근과 차단을 설정 하기 위해 VPC 탭에서 보안 그룹과 네트워크 ACL 및 라우팅 테이블 수정하였다. 그리고 FileZilla Client를 통해 Bastion Host에 Web Server, Web application server의 Key Pair인 .ppk 파일을 Bastion Host 업로드 한 후에 ssh -i group4.ppk ec2-user@10.0.11.91로 접근을 시도

하였지만, Web Server로 접근을 하지 못 하였다.. Web Server, Web application server로 접근 할 때 Bastion Host에서 .ppk파일이 아닌 .pem 파일을 이용하여 접속 해야한다는 걸 알게 되어, Web Server, Web application server 서버를 Key Pair로 “master.pem”를 사용하도록 재생성 하였다. Bastion Host에 “master.pem”를 업로드 후 서버에 재접속을 시도 하였지만, 접속이 되지 않았다.

VPC탭에서 네트워크 ACL을 클릭 한 후 인바운드 규칙을 추가 하였지만, 아웃바운드 규칙을 추가 하지 않아 접속이 불가능 했다는 걸 알게 되었다. 그래서 네트워크 ACL의 아웃바운드 규칙에 “모든 트래픽” 0.0.0.0/0을 추가 후 재접속을 시도 하였다.

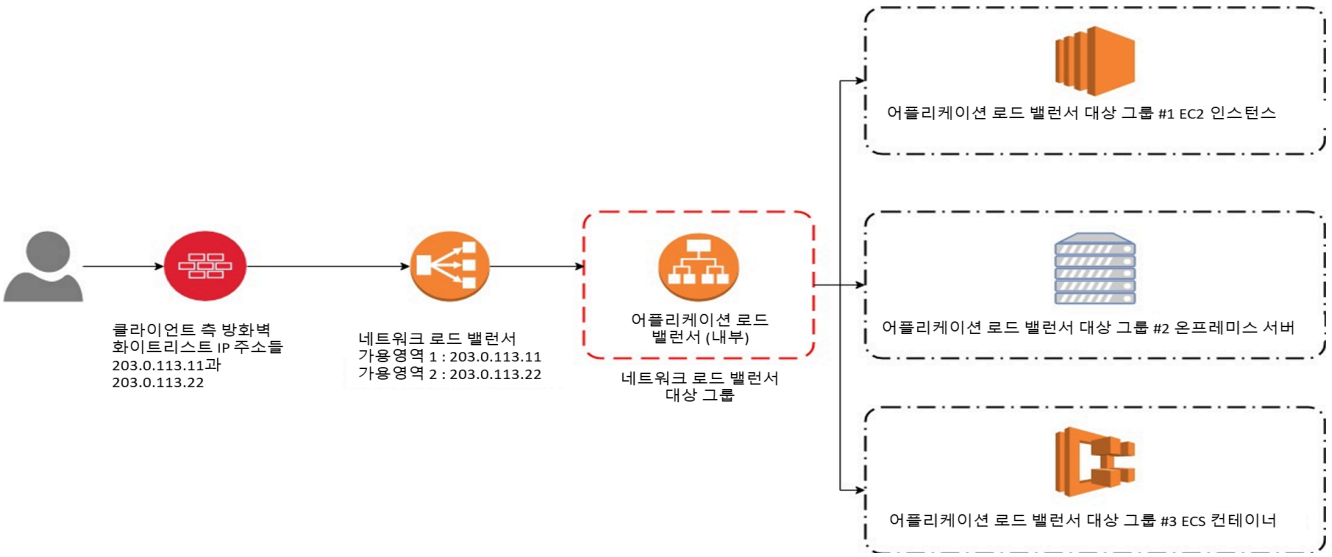


Bastion Host를 통해 Private Subnet 내의 Web Server에 접근 성공.

3.Web Server, Web application server는 Private 영역에 존재 하므로, Load Balancer를 통해 접근 해야한다. 그렇기에 WebServer, Web application server 보안 그룹에 설정한 Load Balancer IP를 넣어주고, HTTP를 통해 로드밸런서에 DNS로 접속 하였지만, Web Server에 접속이 되지 않았다. 그래서 Private Subnet에 네트워크 ACL 설정과 보안 그룹에서 HTTP로 들어오는 모든 IP를 허용하고 다시 재접속을 해본 결과 HTTP를 통해 로드밸런서에 DNS로 접속 할 경우 Web Server에 정상적으로 접속이 되는 걸 확인 하였다.

2. 구현하지 못한 요소

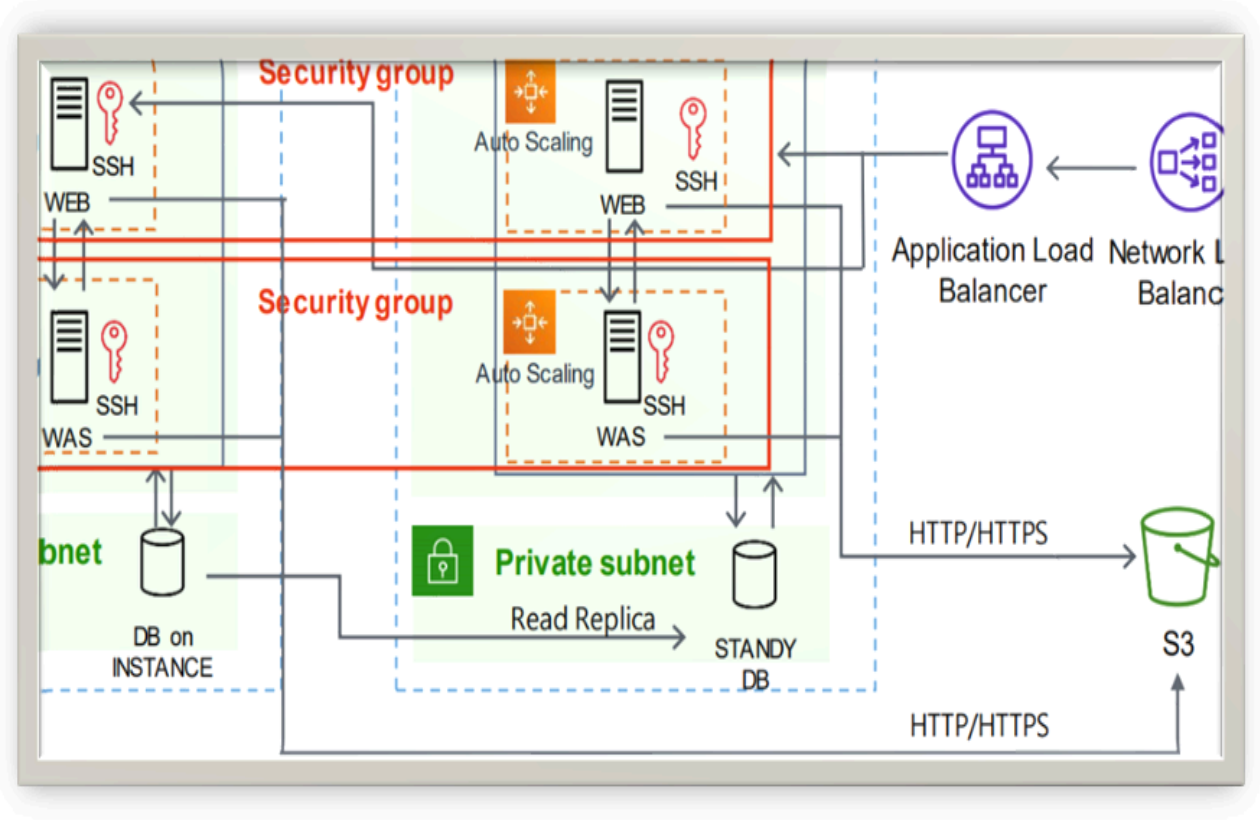
1.애플리케이션 로드 밸런서(ALB)에 고정 IP 주소 설정 및 사용하기



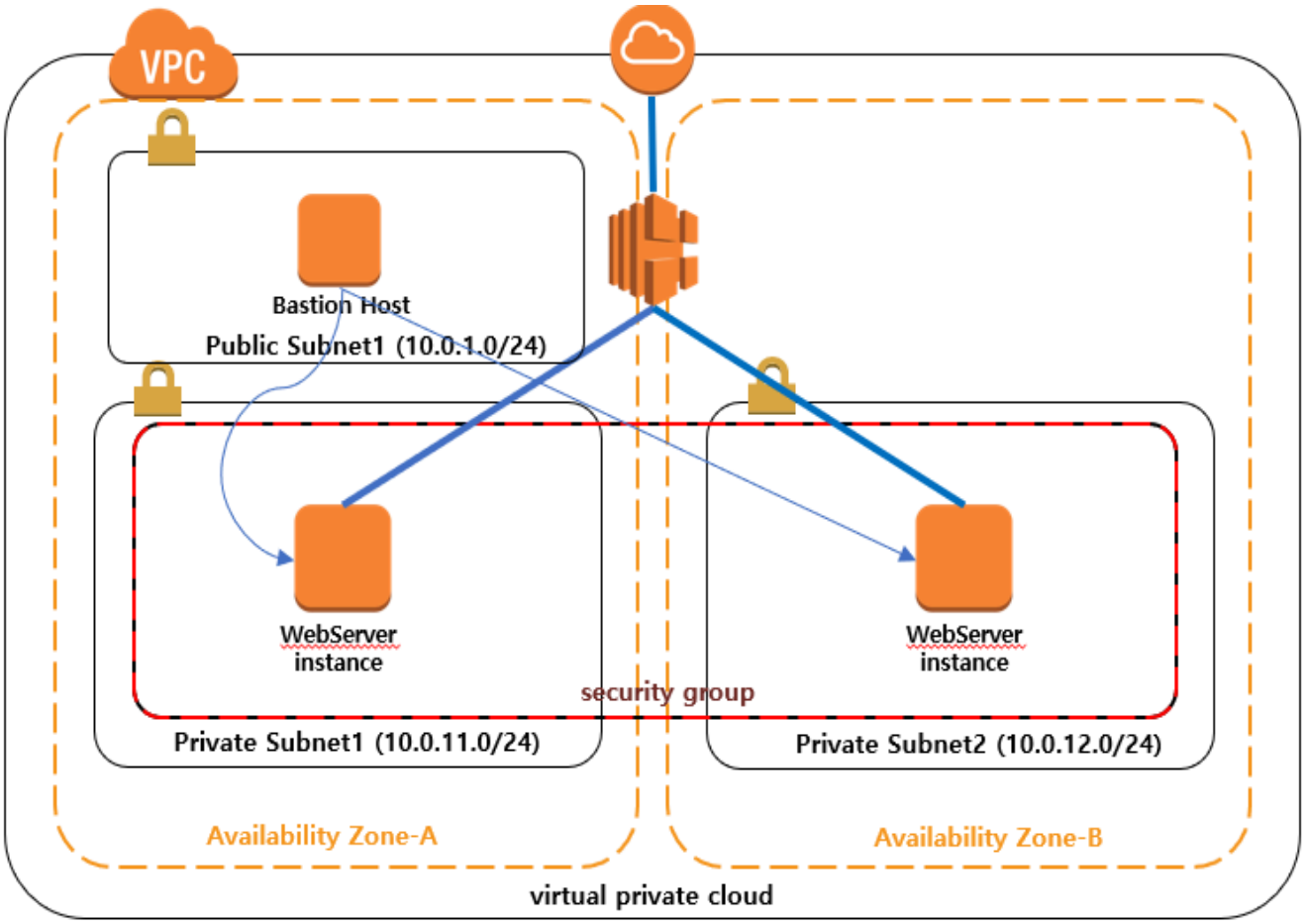
4계층 TCP 로드 밸런서인 네트워크 로드 밸런서(Network Load Balancer, NLB)에 고정 IP를 할당하고 애플리케이션 로드 밸런서(Application Load Balancer, ALB)를 연결하여, OSI 7 계층의 HTTP와 TCP 트래픽을 분산 하려고 하였다.

2.AWS(Amazon) EC2 Instance에 S3 mount 하기

아마존에서 제공하고 있는 S3를 file system처럼 mount하여 스토리지에 EC2에서 출력되는 로그를 자동으로 S3에 저장하여, 서버를 백업 하려고 하였다.



3.Private Subnet에 로드 밸런서(ALB)의 IP만을 허용하며 Web Server의 Public Subnet에 접근하기.



3. 구현하지 못한 이유

1.애플리케이션 로드 밸런서(ALB)에 고정 IP 주소 설정 및 사용하기

외부(인터넷) NLB설정 하고, 탄력적 IP를 할당 받아 내부 ALB와 연결을 하려고 시도 하였지만,

HTTPS를 분산하려고 할 경우, 도메인 생성이 필요하다. 하지만 도메인 생성을 할 경우 비용이 발생하기에, 도메인을 생성하지 않는 HTTP 트래픽만을 분산 하려고 하였지만, Lambda 함수를 설정하는 작업에서 오류가 발생하여 애플리케이션 로드

밸런서(ALB)에 고정 IP 주소를 할당 하지 못 하였다.

2.AWS(Amazon) EC2 Instance에 S3 mount 하기

S3FS를 이용해 AWS S3 mount 해서 사용하려고 시도 하였지만,

```
$ sudo yum install automake fuse fuse-devel gcc-c++ git libcurl-devel libxml2-devel make openssl-devel
```

해당 명령어를 입력 하였음에도 불구하고, 정상적으로 필요한 패키지가 설치 되지 않아 구현하지 못 하였다.

3.Private Subnet에 로드 밸런서(ALB)의 IP만을 허용하며 Web Server의 Public Subnet에 접근하기.

Web Server 보안 그룹에 NLB에 할당된 IP가 HTTP를 이용해 접근 할 수 있도록 허용하고, VPC 탭에 있는 네트워크 ACL 설정에서 TCP 로드 밸런서인 네트워크 로드 밸런서(Network Load Balancer, NLB)의 IP 접속을 허용하고, 로드 밸런서의 DNS 주소를 이용하여 http로 로드밸런서에 접속을 시도 하였지만, 정상적으로 Web Server에 접속이 되지 않아, 어쩔 수 없이 Private Subnet에 로드 밸런서의 IP 뿐만 아니라 http들어오는 모든 트래픽을 허용하였다.

- 프로젝트 수행 보고

1. 프로젝트 팀원 및 참여도

이름	학번	주역할	기여도(합 100%)	비고

2. 프로젝트 추진 단계

3. 프로젝트 전체 추진 일정

4. 프로젝트 팀 전략 및 성과

V. 참고 문헌