인프라 구축 가이드

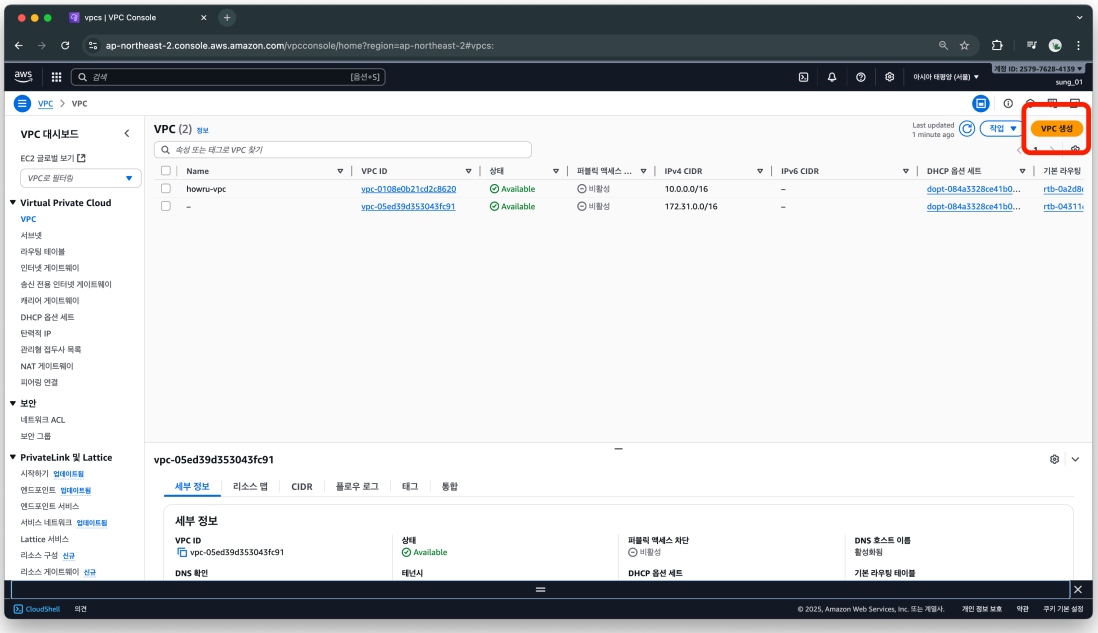
1. 네트워크 구성
    1.1 VPC 생성
    1.2 보안그룹 생성
        1.2.1 Backend용 보안 그룹 생성
        1.2.2 인바운드 규칙 생성
        1.2.3 아웃바운드 규칙 생성
        1.2.3 ALB 보안 그룹 생성
        1.2.4 Backend 보안그룹에 인바운드 규칙 추가
    1.3 ALB 생성
        1.3.1 ALB 기본 설정
        1.3.2 도메인 얻기
        1.3.3 ALB 설정 마무리
    1.4 NAT Gateway 생성
2. Rds & Redis 구축
    2.1 Rds(PostgreSQL) 설정
    2.2 ElastiCache(Redis) 생성
        2.2.1 설정
        2.2.2 고급 설정
        2.2.3 검토 및 생성
3. Github 파이프라인 구축
    3.1 ECR 생성
        3.1.1 Backend Server용 ECR 생성
    3.2 spring server용 Github deploy.yml 및 task-definition 작성
        3.2.1 deploy.yml
        3.2.2 task-definition 생성
        3.2.3 git secrets 생성
        3.2.3+ 변수 확인하는 곳
        AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY
        ECR Registry URL
        REDIS_HOST
        REDIS_PASSWORD
    3.3 Spacy-api 용 Github deploy.yml 및 task-definition 작성
        3.3.1 deploy.yml 작성
        3.2.2 task-definition 생성
        3.3.3 git secrets 생성
    3.4 fast-api 용 Github deploy.yml 및 task-definition 작성
        3.4.1 deploy.yml 작성
        3.4.2 task-definition 생성
        3.4.3 git secrets 생성
4. ECS
    4.1 CloudMap 설정
        4.1.1 namespace 생성
        4.1.2 서비스 생성
    4.2 Task Definition
        4.2.1 fast-api task_defintion.json
        4.2.2 spacy-api task_defintion.json
        4.2.3 libretranslate task_defintion.json

4.2.4 springboot_task_definition.json
4.3 클러스터 생성
4.4 서비스 생성
4.4.1 fast-api service 생성
4.4.2 spacy-api service 생성
4.4.3 LibreTranslate service 생성
4.4.4 SpringBoot service 생성
5. 마무리

1. 네트워크 구성

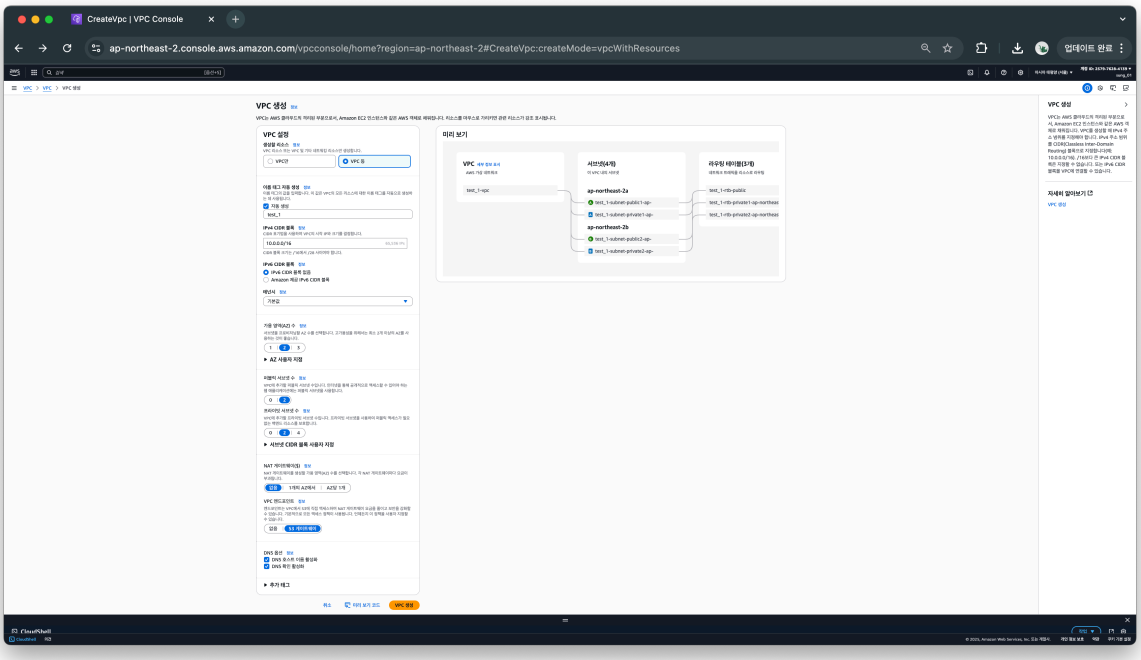
1.1 VPC 생성

- Aws console → VPC → VPC 생성



- ✅ Vpc등 체크하면 다른 UI가 보인다.
- ✅Vpc 이름 태그 자동 완성 작성
  - Vpc 이름을 정해주면 public subnet, private subnet를 수월하게 식별할 수 있도록 prefix로 들어감
- Ipv4 CIDR 블록
  - 10.0.0.0/16 입력 (다르게 입력해도 됨)
- ✅ IPv6 CIDR 블록 없음
- 테넌시 기본값(default)
- 가용영역(AZ) 2개 설정
- Public subnet Private subnet 둘다 2개씩 생성(통상적으로)
- NAT Gateway `없음` 설정 (나중에 직접 추가 예정)

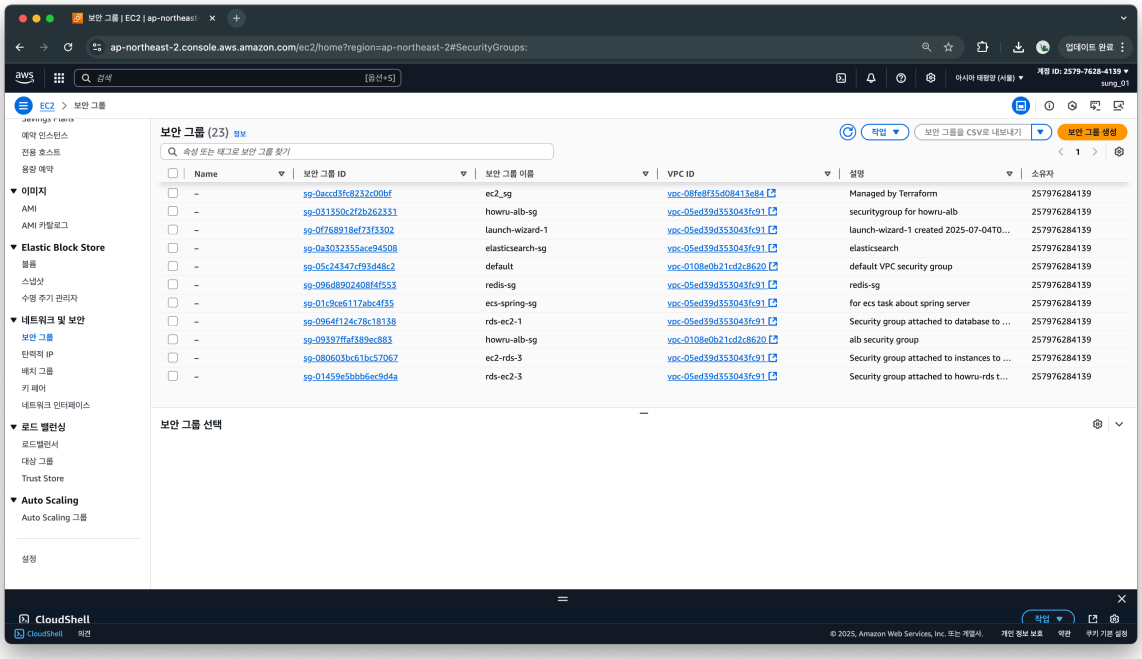
- VPC Endpoint `없음` 설정
- DNS option
  - ✅ DNS 호스트 이름 활성화
  - ✅ DNS 확인 활성화
- VPC생성

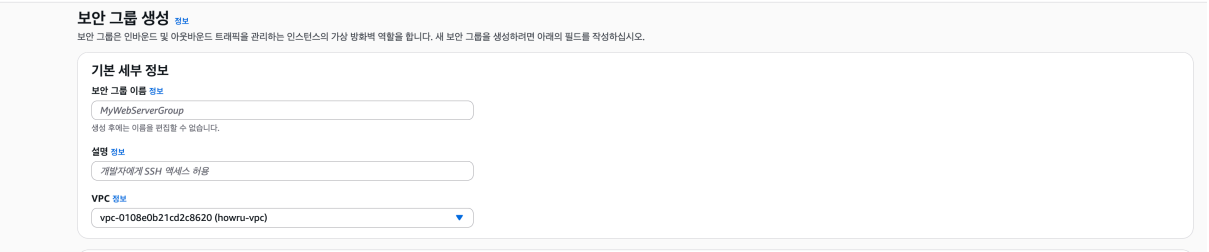


1.2 보안그룹 생성

1.2.1 Backend용 보안 그룹 생성

Console → 네트워크 및 보안 → 보안 그룹 → 보안 그룹 생성 클릭

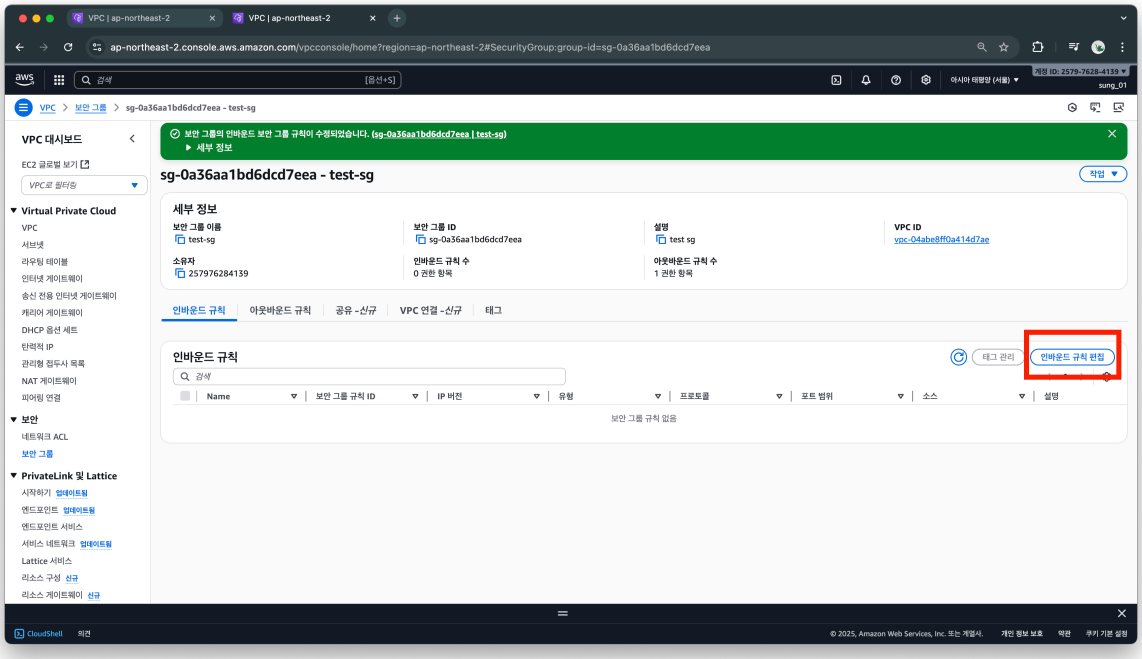




- 보안 그룹 이름 입력
- 설명 입력 (필수이므로 아무거나 입력)
- 자신이 생성한 VPC로 설정 ← 중요
- 보안 그룹 생성

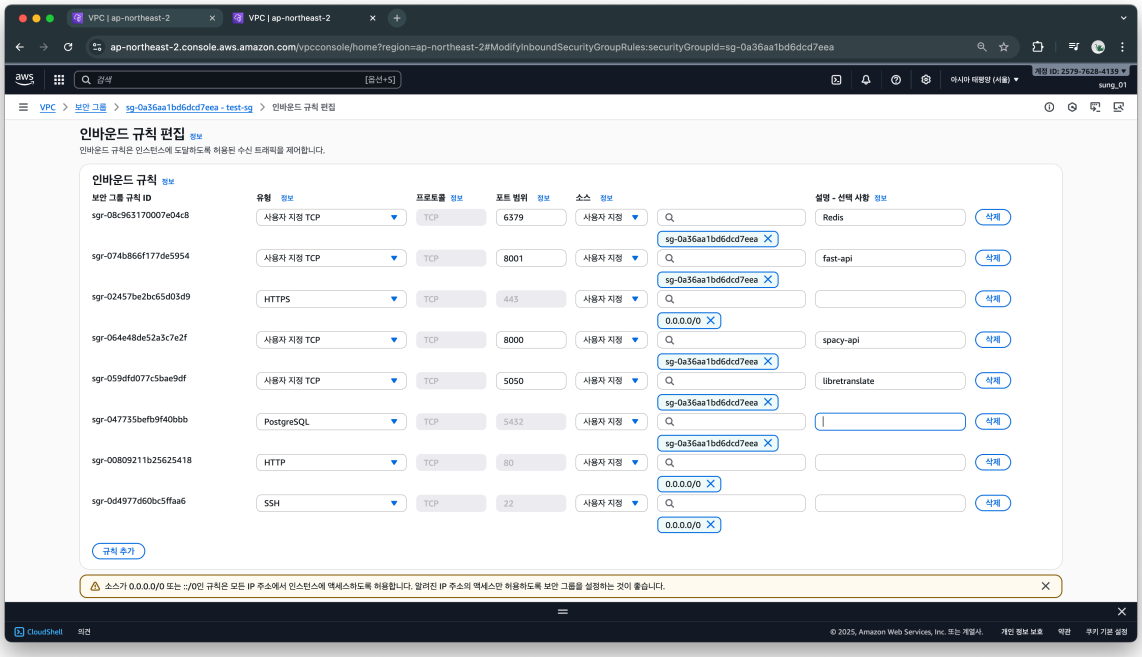
1.2.2 인바운드 규칙 생성

방금 생성한 보안그룹 클릭



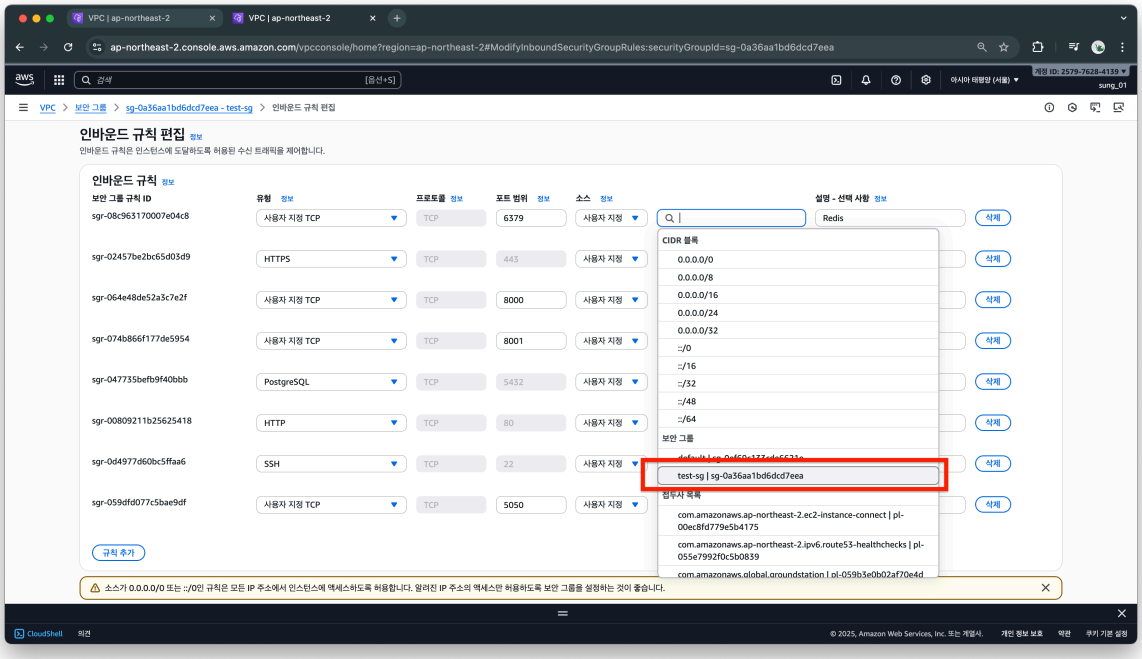
다음과 같은 인바운드 규칙을 입력한다. 하단의 규칙 추가 버튼을 클릭하여 하나씩 입력하면 된다.

설명 은 필수사항이 아니다.



Port 6379, 8000, 8001, 5050,5432 의 소스는 생성했던 보안그룹을 선택해야한다.

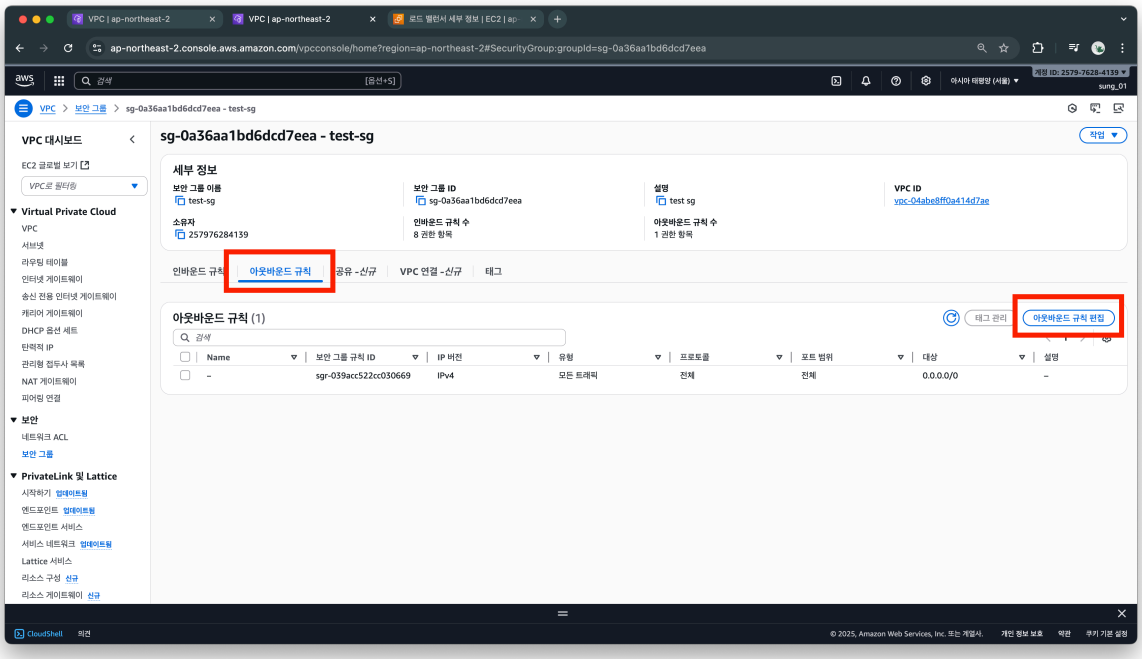
즉 자기 자신을 선택해주면 된다. 다음과 같이 선택하면 된다.

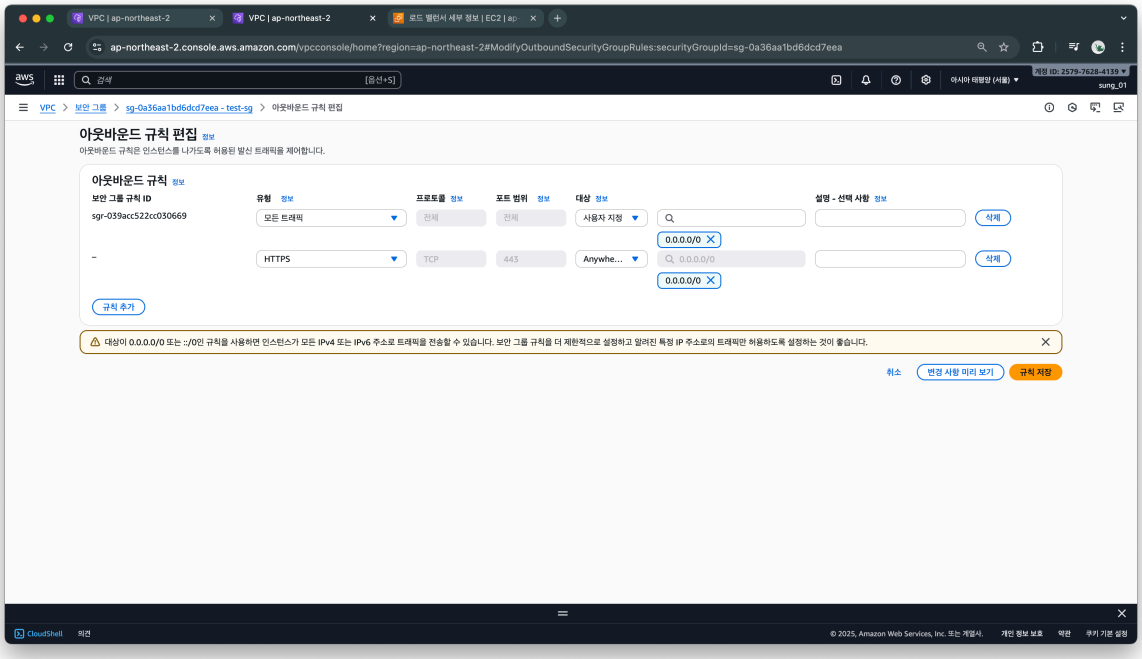


설정을 완료했으면 규칙저장 을 누른다.

1.2.3 아웃바운드 규칙 생성

기존의 보안규칙 → 아웃바운드 규칙→ 아웃바운드 규칙 편집 을 누른다



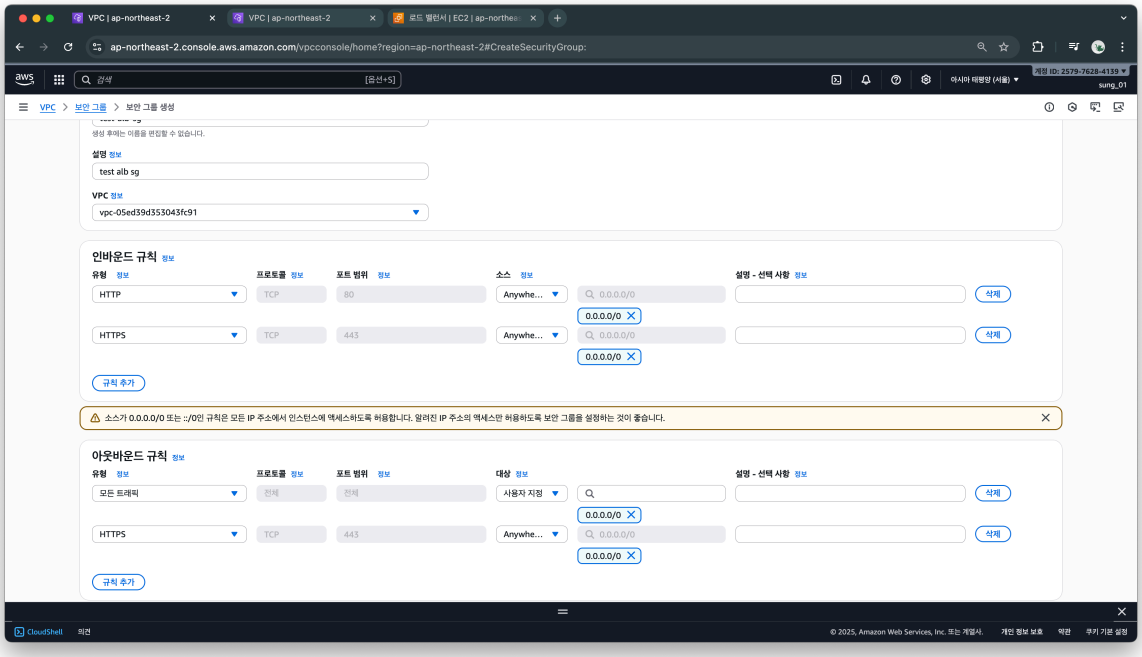


HTTPS 규칙을 생성한다.

1.2.3 ALB 보안 그룹 생성

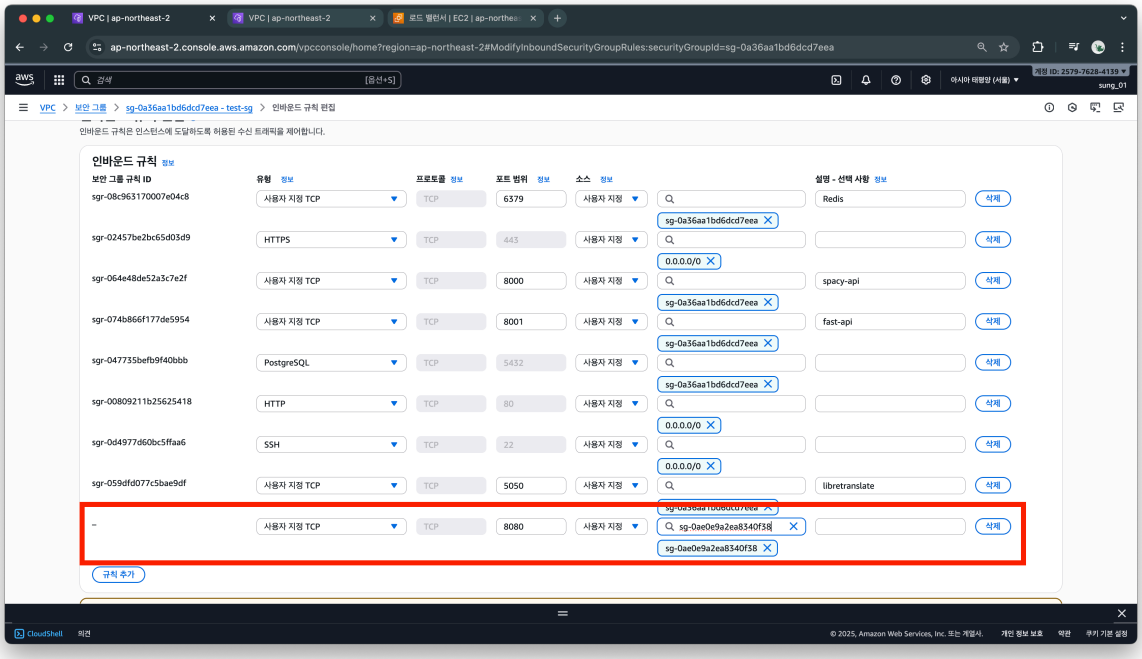
ALB의 보안그룹을 새로 생성한다. 위에서 했던 작업을 똑같이하면 된다.

ALB 보안 그룹은 생성할때 인바운드 규칙과 아웃바운드 규칙을 입력해도 된다.



1.2.4 Backend 보안그룹에 인바운드 규칙 추가

ALB의 보안그룹까지 생성했으면, 1.2.1 에서 생성한 Backend 보안 그룹의 인바운드 규칙에 8080포트를 추가해준다.



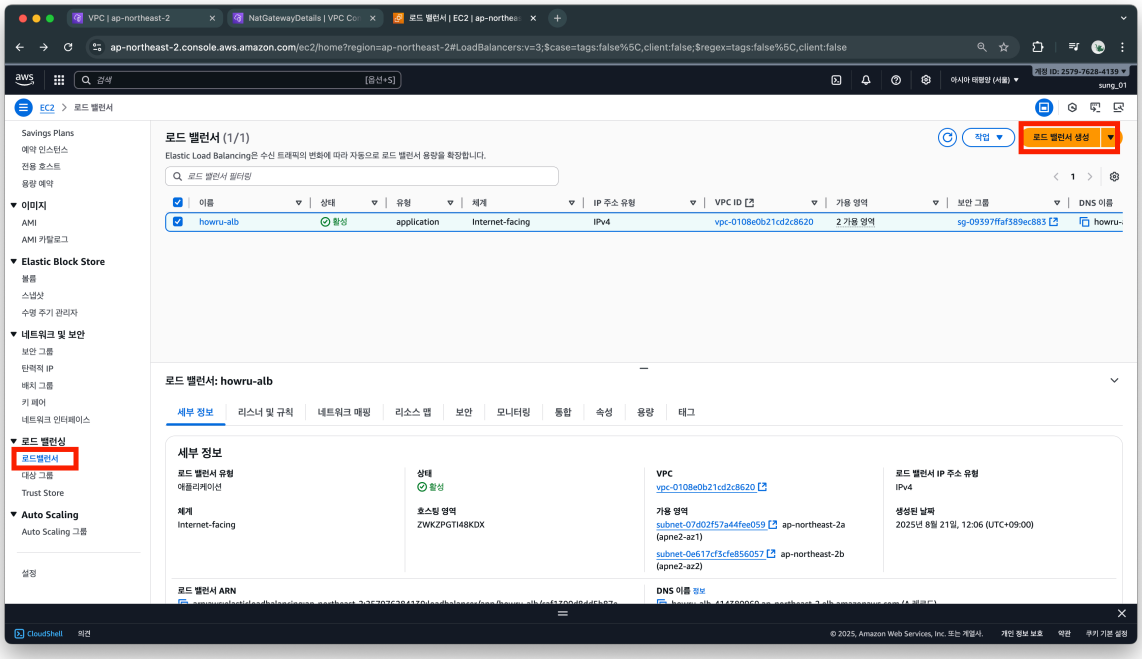
8080 포트에 대한 소스는 자기 자신이 아니라 1.2.3에서 생성한 ALB의 보안그룹이여야한다. 중요하니까 놓치면 안된다.

설정을 했으면 규칙을 저장한다.

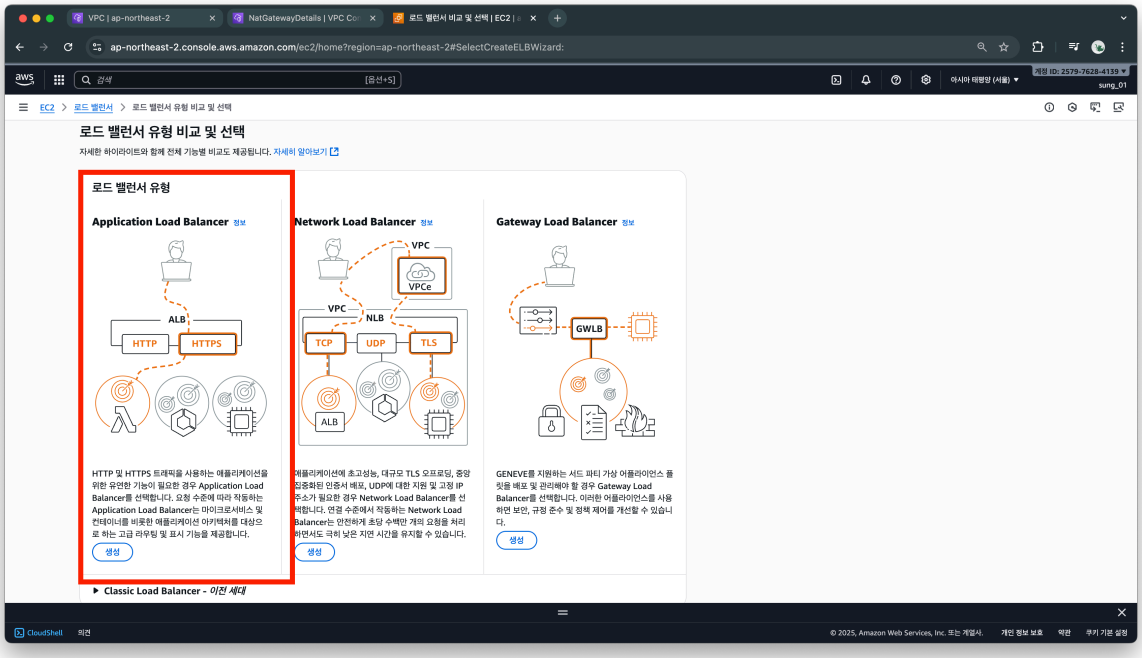
1.3 ALB 생성

1.3.1 ALB 기본 설정

콘솔에서 EC2→로드밸런서→로드 밸런서 생성



로드 밸런서 유형 `Application Load Balancer` 선택

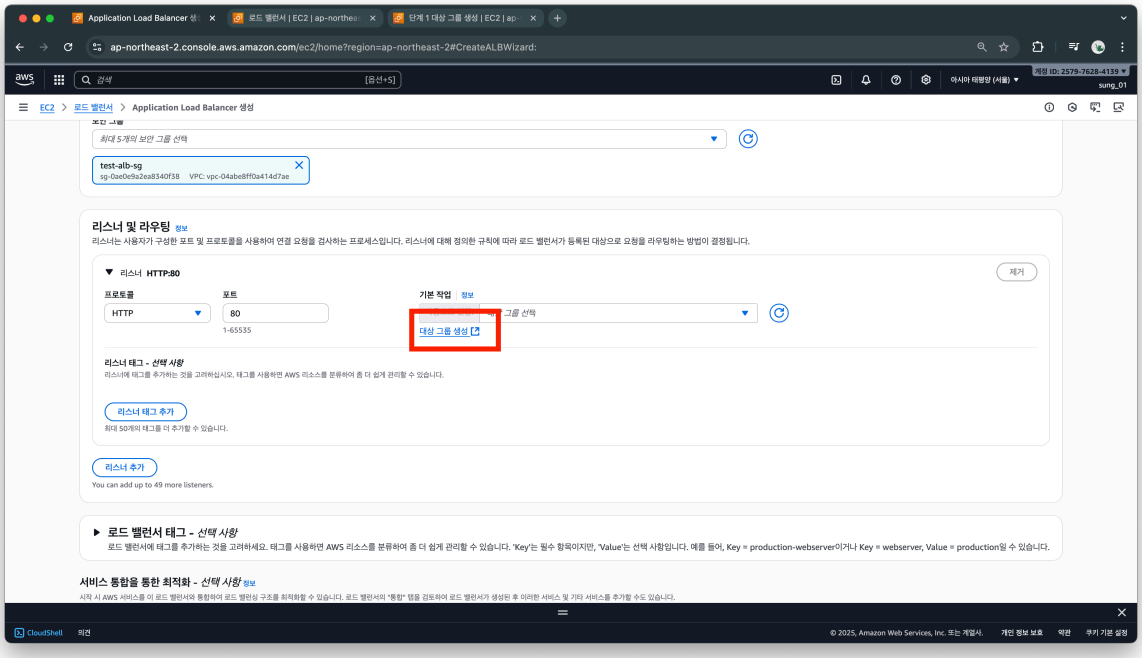


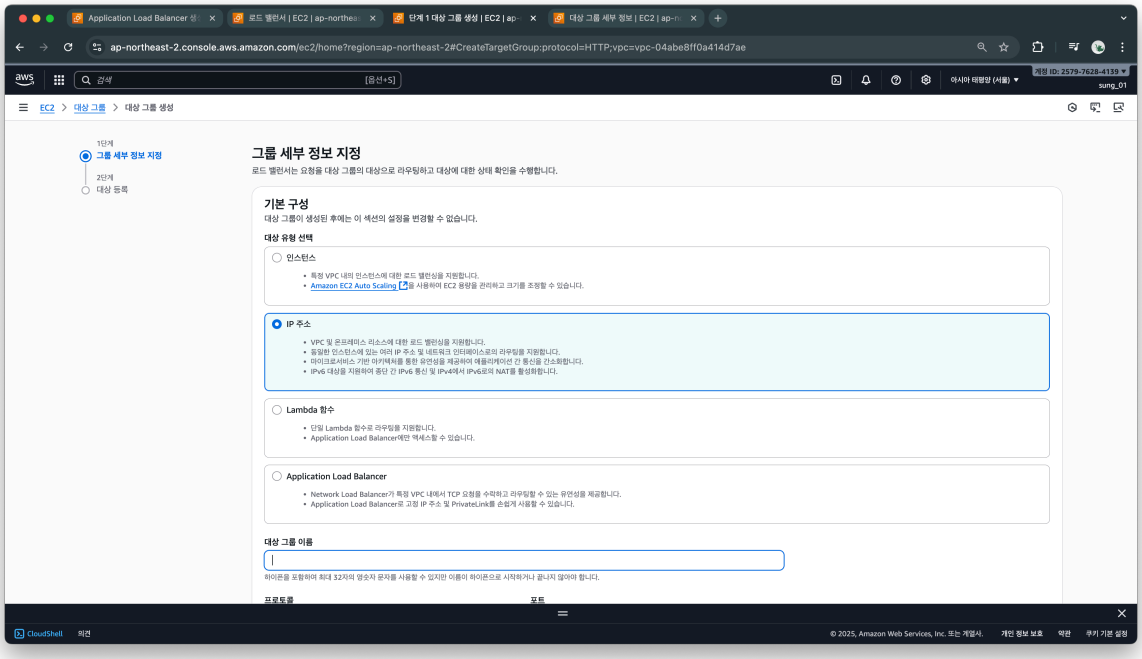
- 체계 : ✅ 인터넷 경계
- 로드밸런서 IP 주소 유형 : ✅ IPv4
- 네트워크 매핑
  - VPC : `1.1` 에서 생성한 VPC 설정
  - IP 풀 : ❌ 체크 해제

- 가용영역 및 서브넷 : VPC 생성할때 만들어진 `Public subnet` 2개를 등록

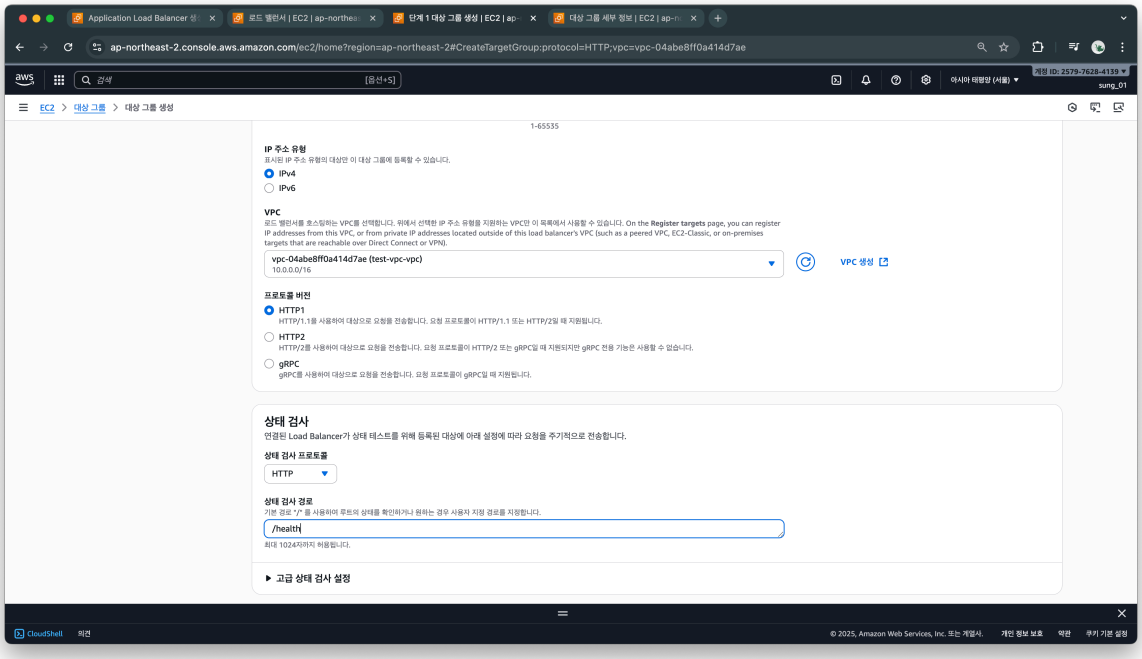
반드시 VPC안에 존재하는 Public subnet으로 설정할것

- 보안 그룹 : `1.2.3` 에서 생성한 ALB 보안 그룹으로 설정
- 리스너 및 라우팅 : 대상 그룹 생성 클릭

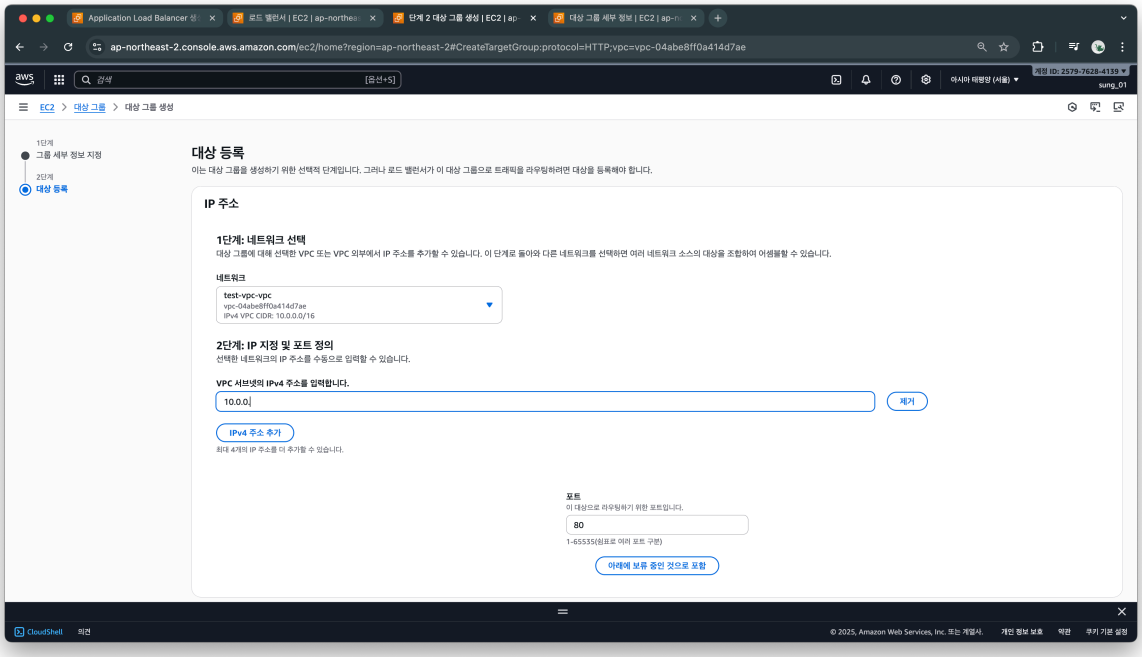




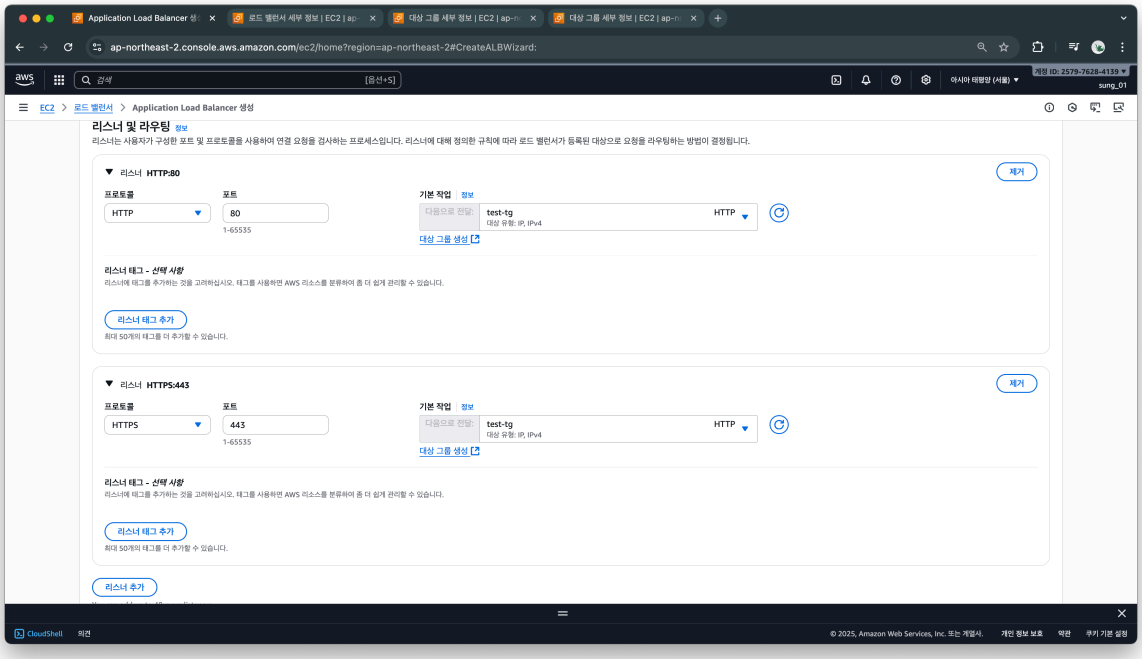
- ✅ IP주소 선택
- 대상 그룹 이름 설정
- 프로토콜 : HTTP로 설정
- VPC : 생성한 VPC로 설정
- 프로토콜 버전 : ✅ HTTP1
- 상태검사 :
  - ✅ HTTP
  - 상태 검사 경로 : `/health` 입력 (Spring 서버에 health controller가 따로 존재)



다음 클릭



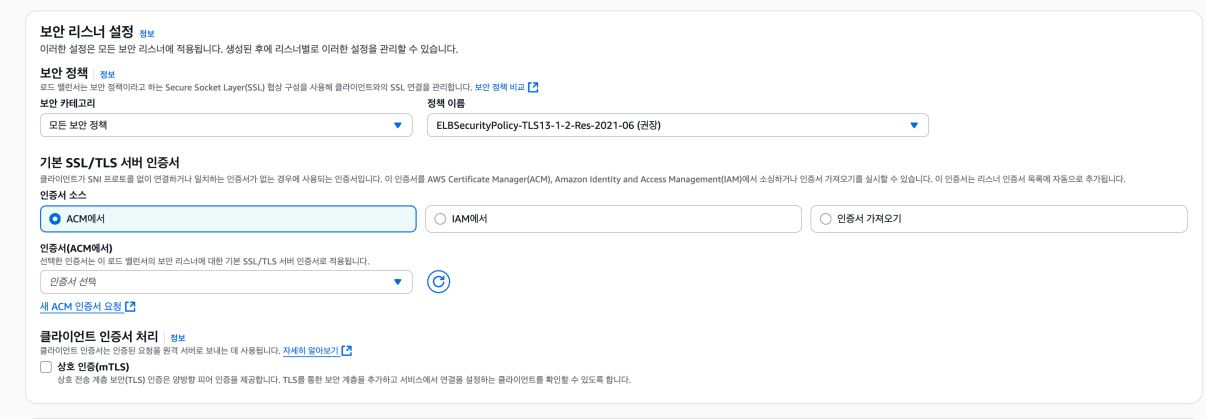
이 페이지는 현재에는 필요없으니 밑으로 내려서 대상 그룹 생성 클릭



다시 ALB 생성 페이지로 돌아와서 리스너를 두개(HTTP,HTTPS) 생성한 다음, 방금 생성한 대상 그룹을 설정

리스너 추가는 하단에 리스너 추가 를 클릭하면 자동으로 생성된다.

- 보안 리스너 설정에서 새 ACM 인증서를 받아야한다. 그 전에, 도메인을 얻기 위한 사전 작업이 필요하다. AWS 는 잠시 멈추자.

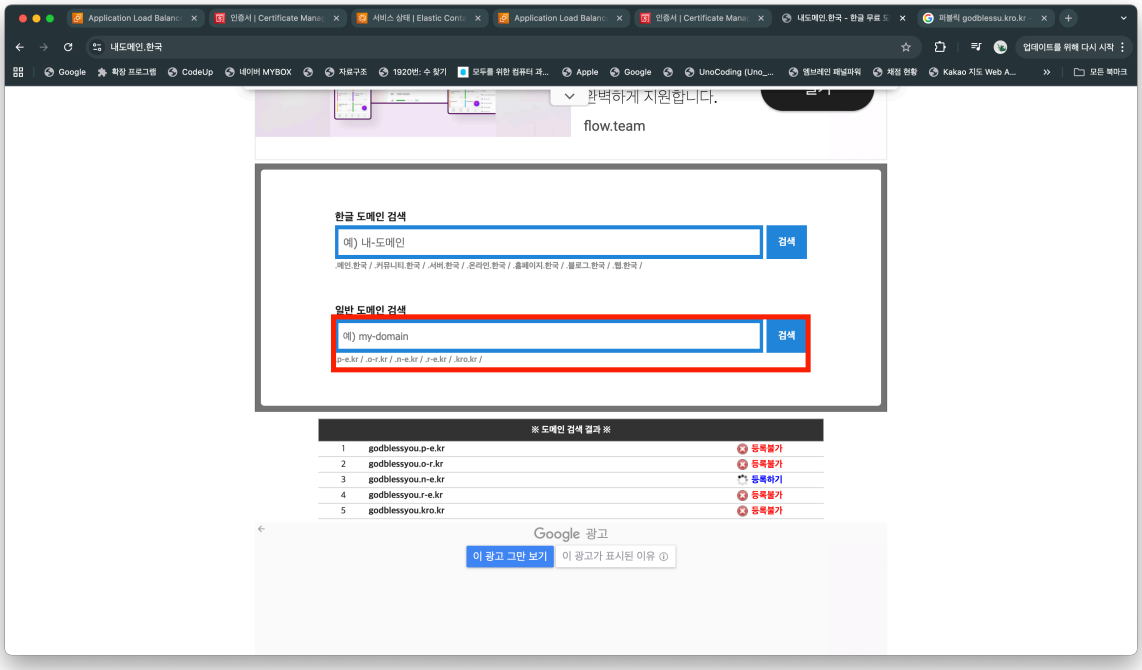


1.3.2 도메인 얻기

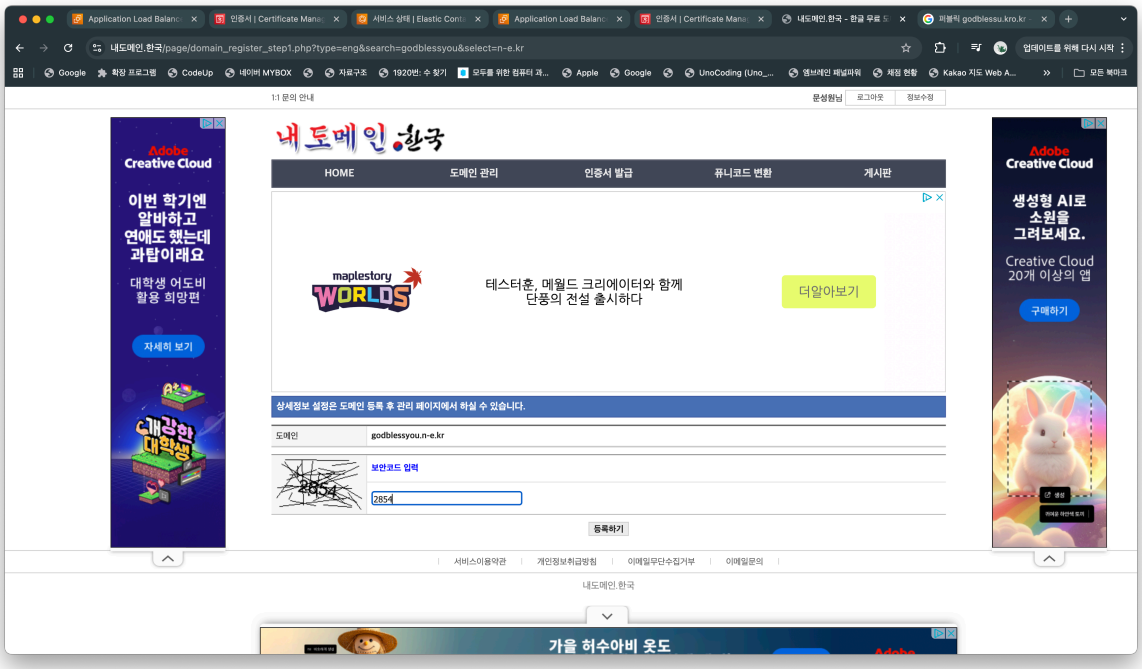
https://내도메인.한국/ 에 들어가 회원가입을 한다.

그 후 HOME 에서 일반 도메인 검색에 원하는 도메인을 검색한다.

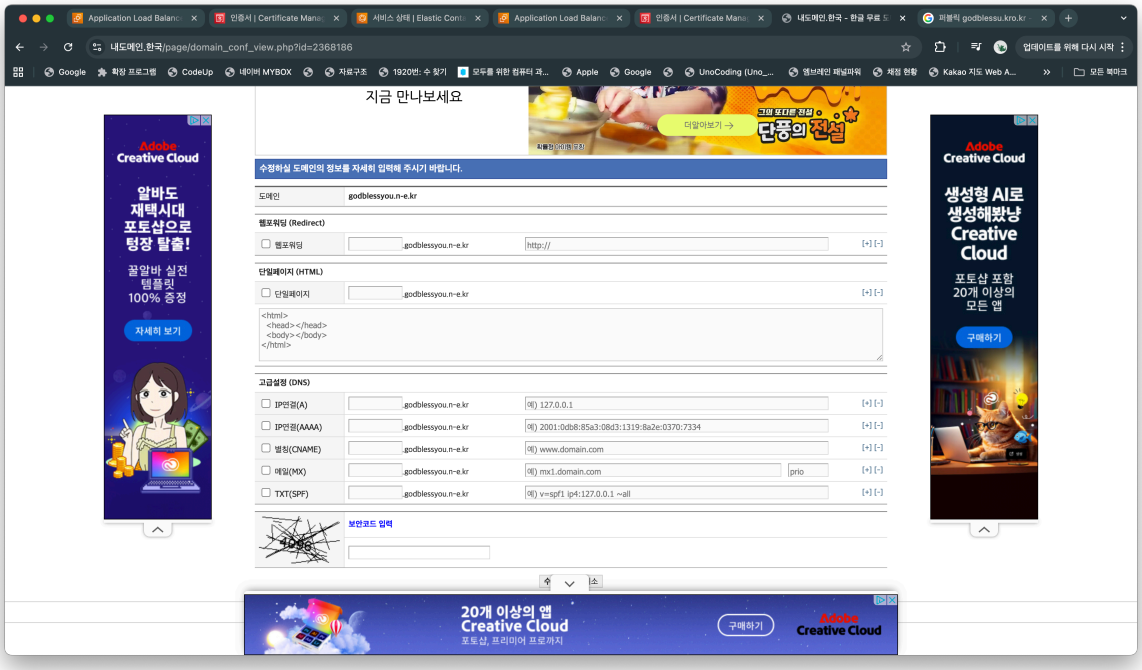
사용가능한 도메인(파란색 글씨로 등록하기 버튼)을 클릭해준다.



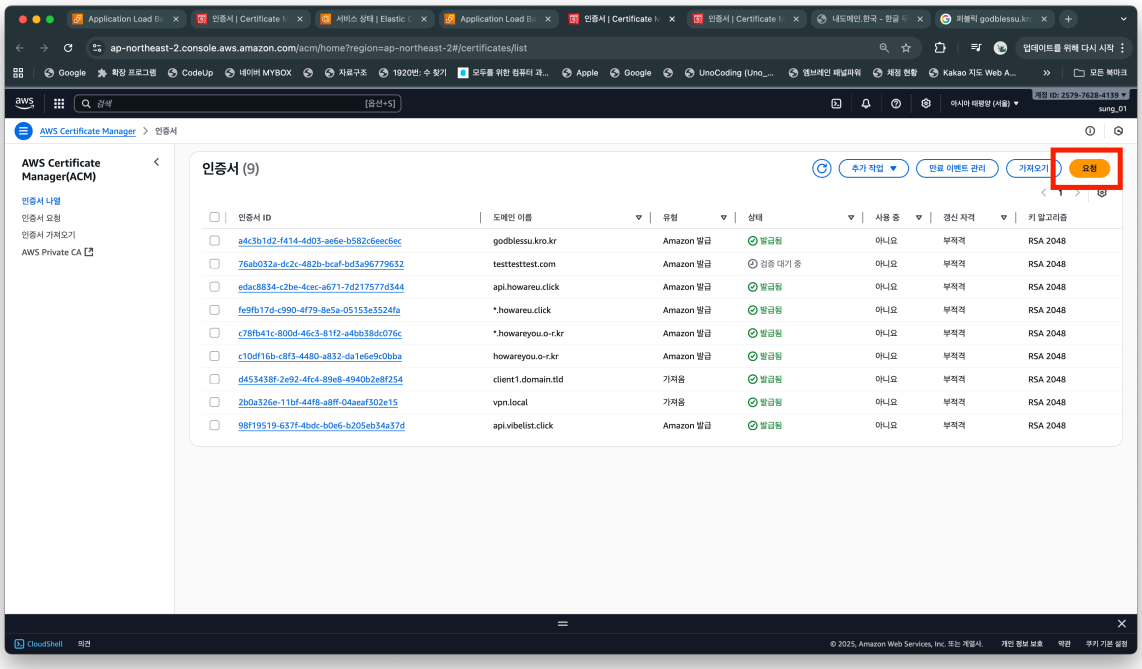
클릭을 누르면 이런 화면을 넘어오는데, 보안코드를 입력하고 `등록하기` 버튼을 눌러주면 도메인이 등록된다.



도메인 정보를 수정하는 페이지로 넘어오는데 이 창을 띄어두고, 다시 `aws console`로 돌아가자

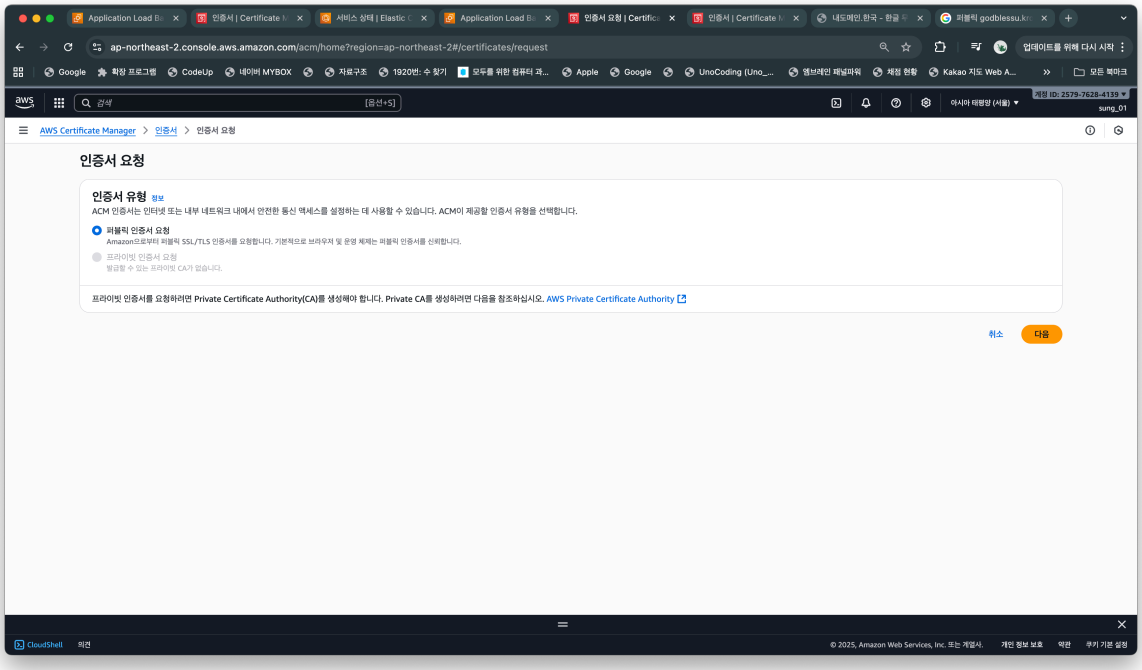


aws console에서새 ACM 인증서 요청 버튼을 누르면 인증서 등록 페이지가 나온다. 그 후 요청 버튼을 누르자

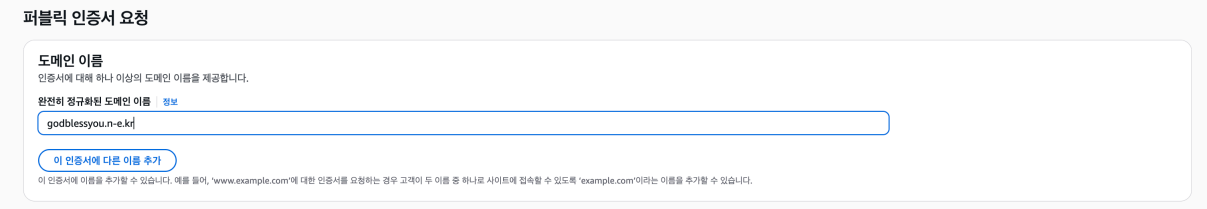


- **인증서 유형 :** ✅ 퍼블릭 인증서 요청

  다음 버튼을 누르자

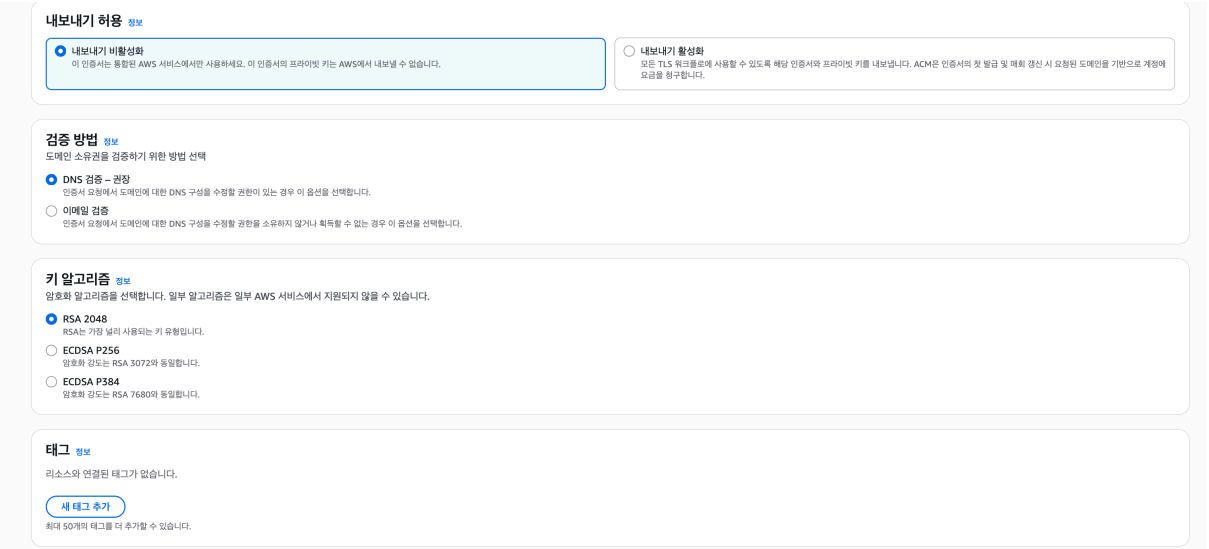


- **도메인 이름**
  - **완전히 정규화된 도메인 이름 :** `내도메인.한국` **에서 등록했던 도메인을 넣어준다.**

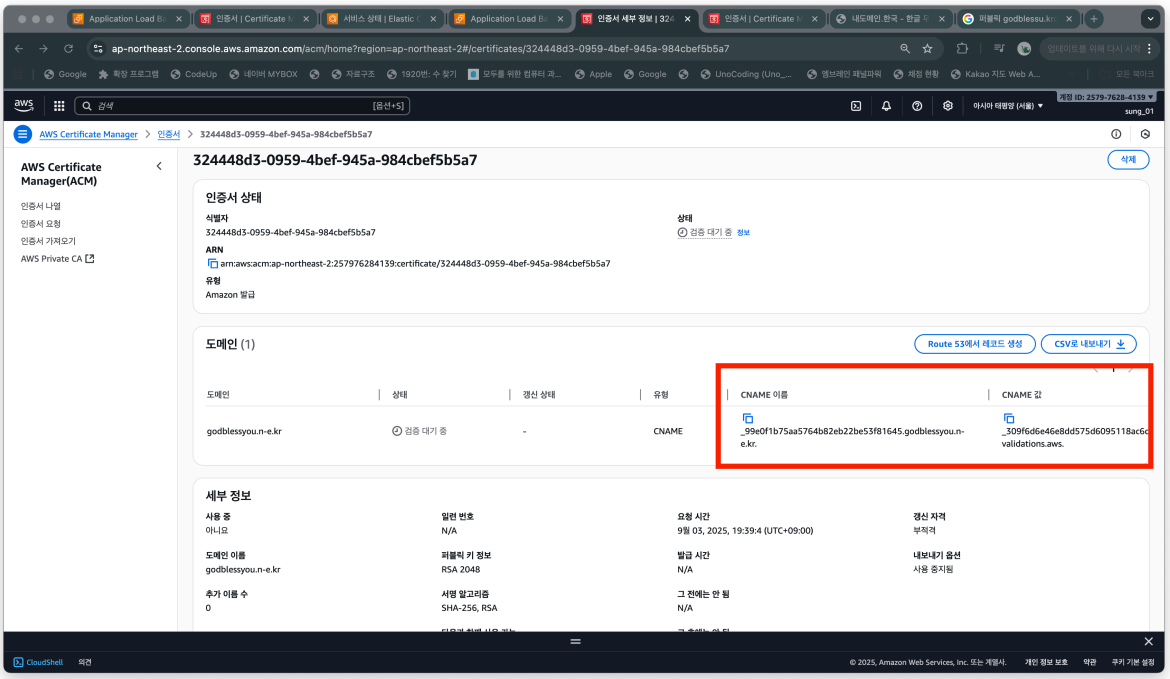


- **내보내기 허용 :**
  - ✅ 내보내기 비활성화
- **검증 방법 :**
  - ✅ DNS 검증 - 권장
- **키 알고리즘 :**
  - ✅ RSA 2048

`요청` 버튼 클릭

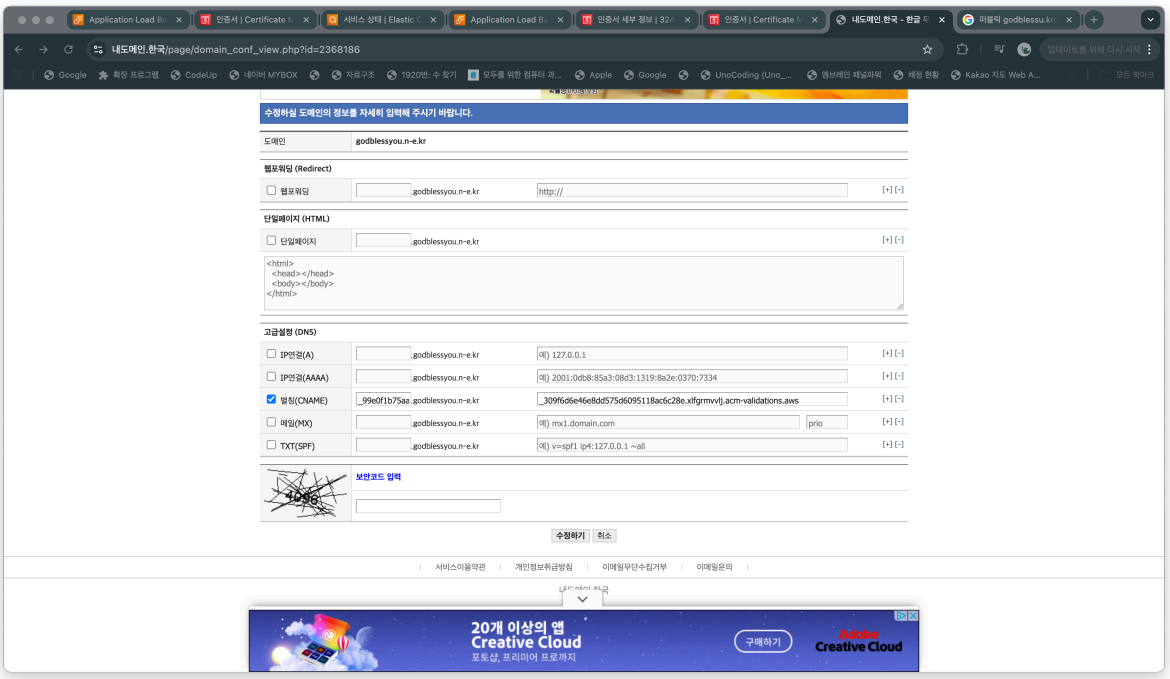


그 후 조금 기다리면 도메인에 `CNAME 이름`, `CNAME 값`이 나온다.

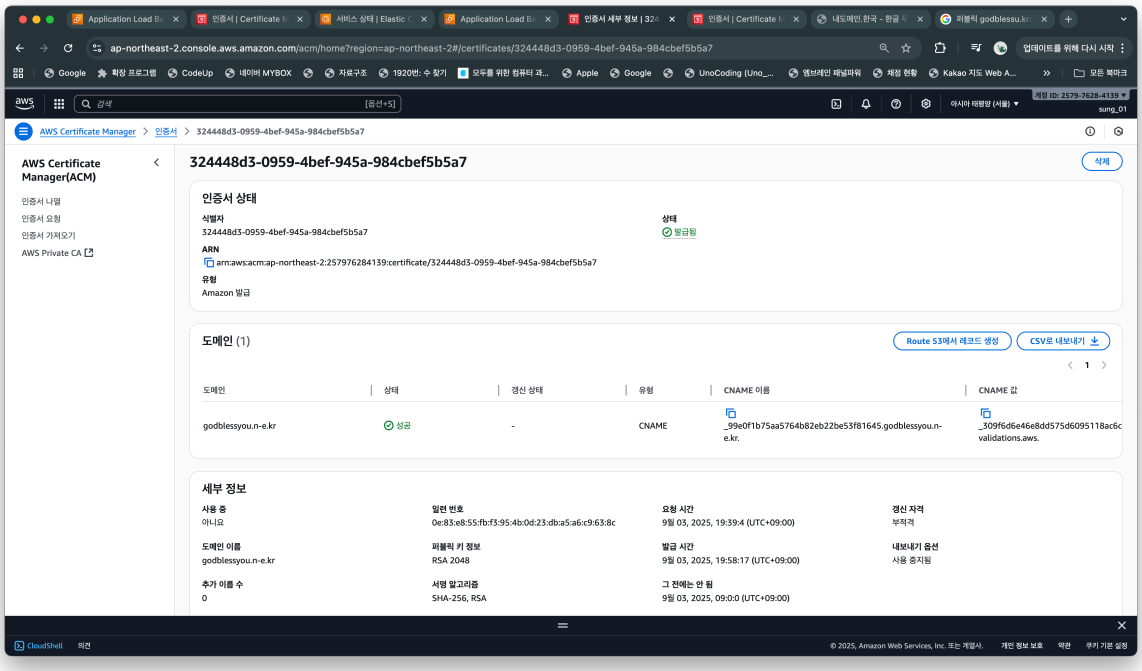


이 값들을 복사해서

`내도메인.한국` 에 별칭(CNAME)에 넣어준다. 주의할점은 그대로 복사해서 값을 넣어서는 안되고 `내도메인.한국` 에서 자동으로 도메인 값을 넣어주기 때문에 `CNAME 이름`의 중복되는 부분을 지워줘야 한다. 그리고 `CNAME 값` 을 복사하면 마지막에 `aws.` 으로 끝나게 되는데 `.` 을 지워져야한다. 다 넣었으면 보안코드 입력 후 `수정하기` 버튼을 눌러준다.

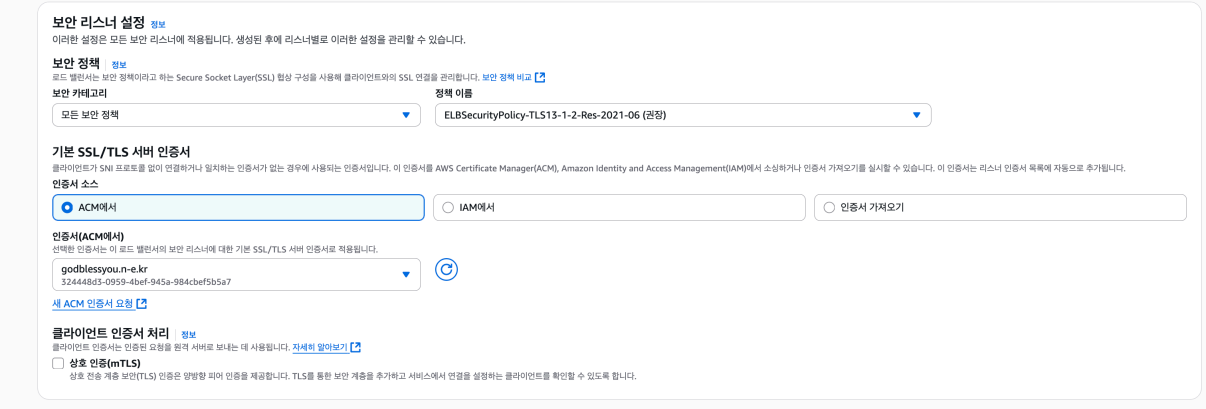


그 후 다시 aws console로 돌아와서 3분~5분 정도 기다리면 상태가 `검토 대기중`에서 `성공`으로 변경된다.



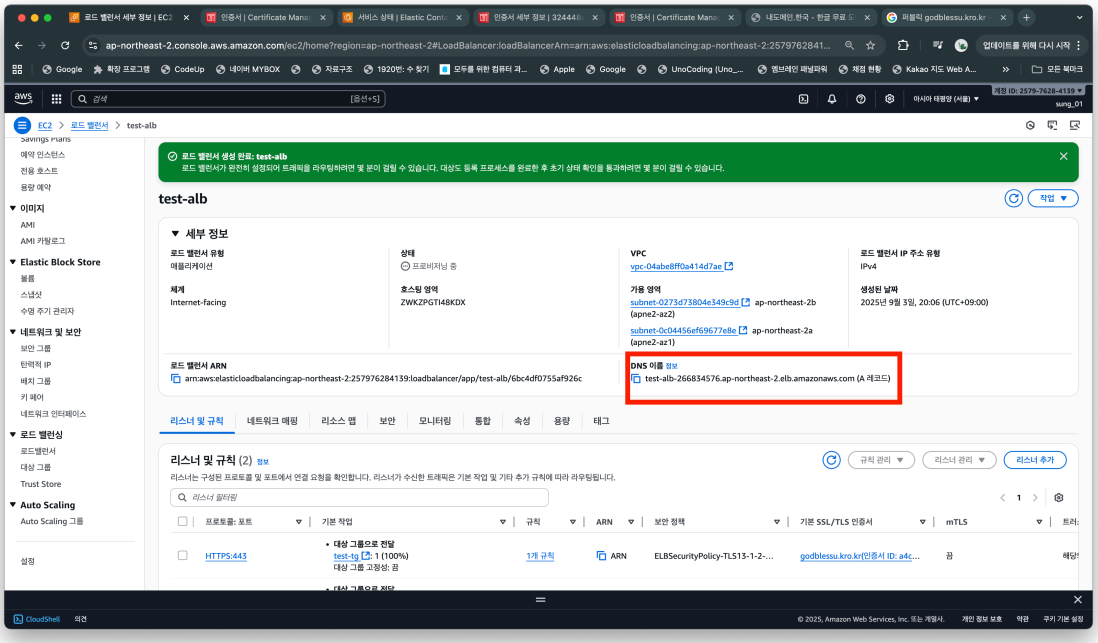
1.3.3 ALB 설정 마무리

`1.3.2` 의 과정을 성공했다면 다시 ALB 설정 페이지로 돌아와 인증서를 선택하자. 오른쪽의 새로고침 버튼을 누르면 방금 생성한 인증서가 보일 것이다.



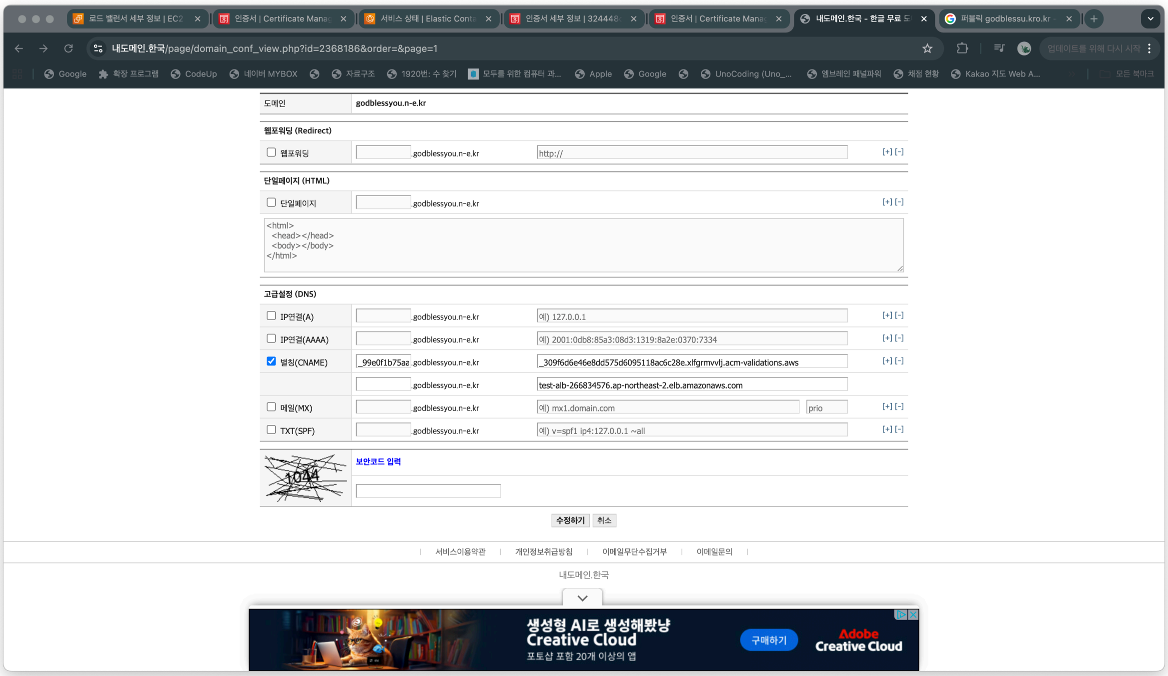
- **보안 리스너 설정**
  - **보안 정책**
    - **보안 카테고리 :** ✅ 모든 보안 정책
    - **정책 이름 :** ✅ ELBSecurityPolicy-TLS13-1-2Res-2021-06(권장)
  - **기본 SSL/TLS 서버 인증서**
    - **인증서 소스 : ✅ ACM에서**
    - **인증서(ACM에서) :** 방금 생성한 인증서
  - **클라이언트 인증서 처리**
    - ❌ 상호 인증

그 후 따로 설정할 것은 없으니 밑으로 스크롤해서 `로드 밸런서 생성 버튼`을 누른다.



로드 밸런서 생성 완료 페이지로 넘어오면 DNS이름을 복사해서 `내도메인.한국` 에 새로운 별칭으로 입력한다.

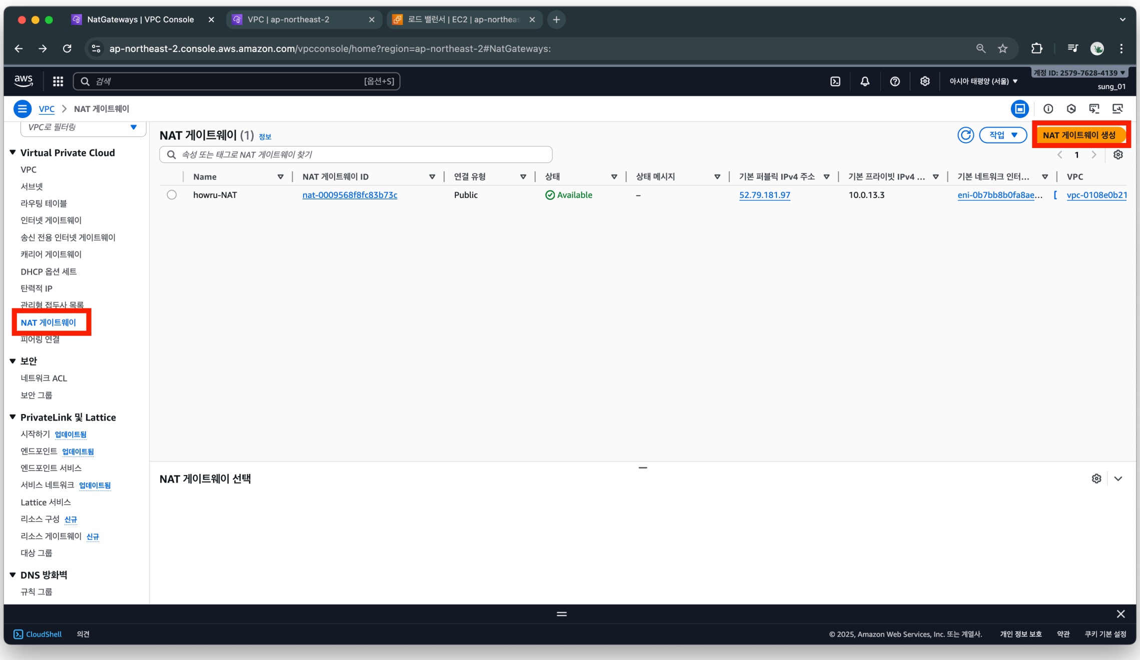
맨 오른쪽 `[+]` 버튼을 누르면 값을 추가할 수 있다.

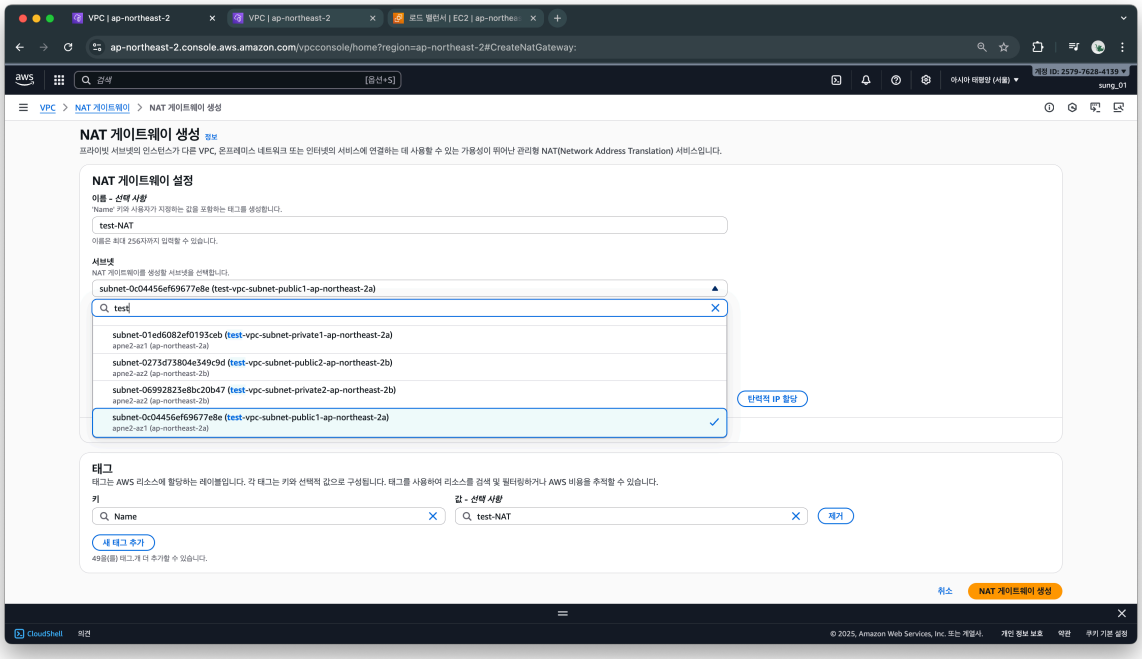


여기까지 완료하면 ALB 설정은 마무리 된다.

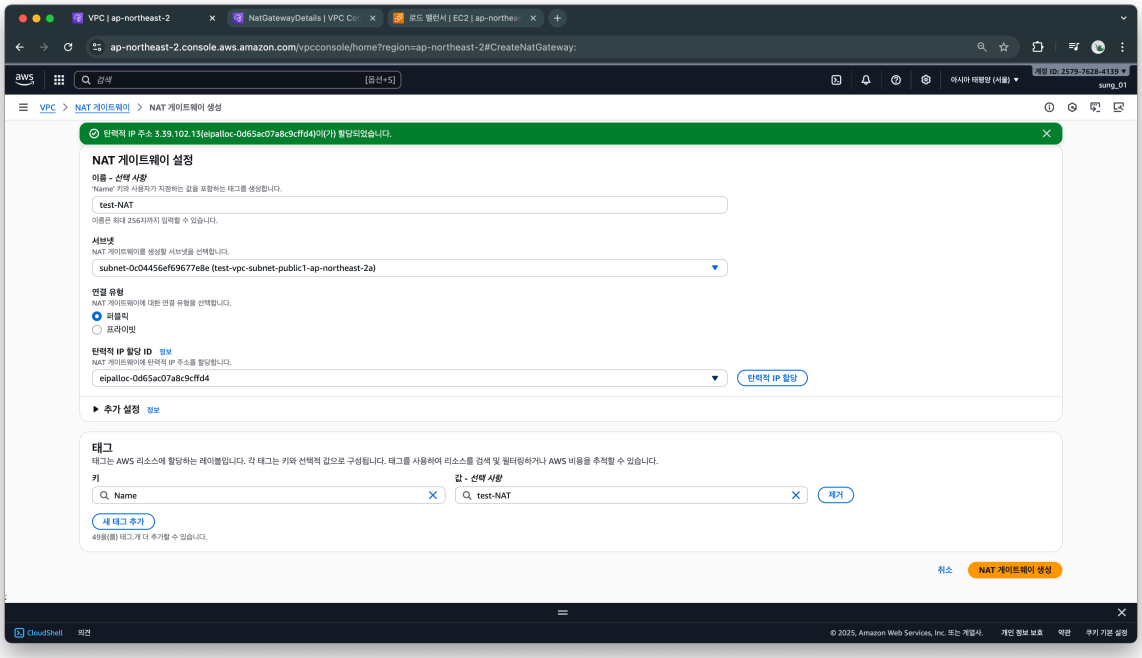
1.4 NAT Gateway 생성

콘솔에서 `VPC→ NAT 게이트웨이` 로 들어가 새로운 NAT Gatewat를 생성한다.





서브넷을 설정할때 1.1 에서 생성한 VPC의 Public subnet을 선택해야하고, 그중에서도 a를 선택하면 된다.



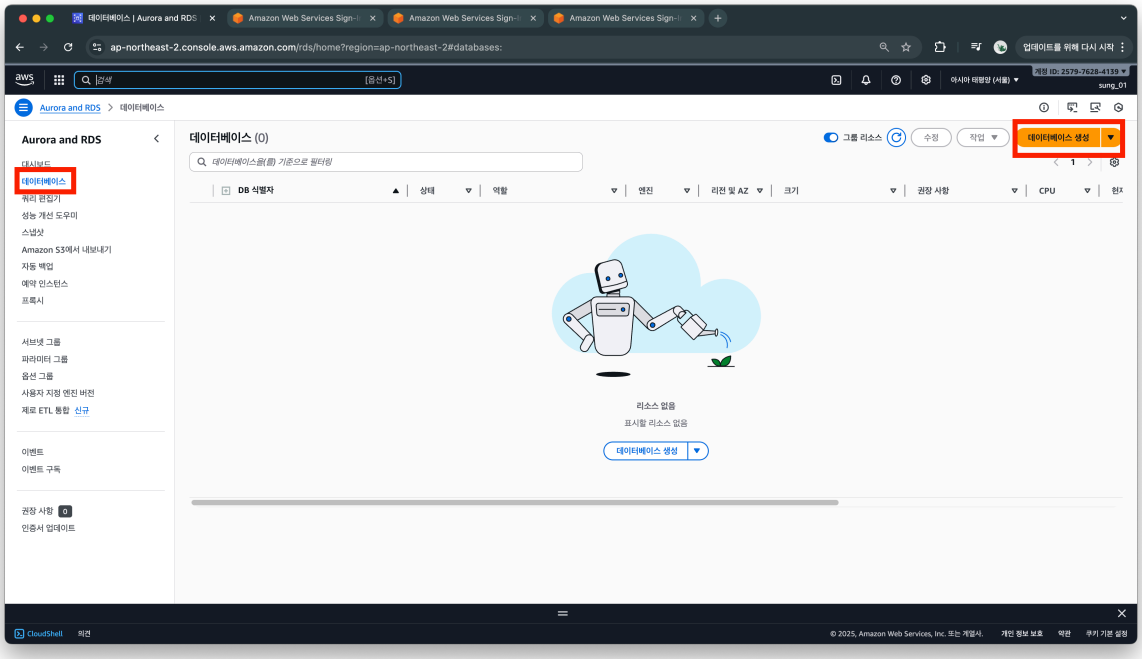
- 연결유형 : ✅ 퍼블릭
- 탄력적 IP 할당 ID는 탄력적 IP 할당 을 클릭하면 자동으로 설정된다.

NAT 게이트웨이 생성을 누른다.

2. Rds & Redis 구축

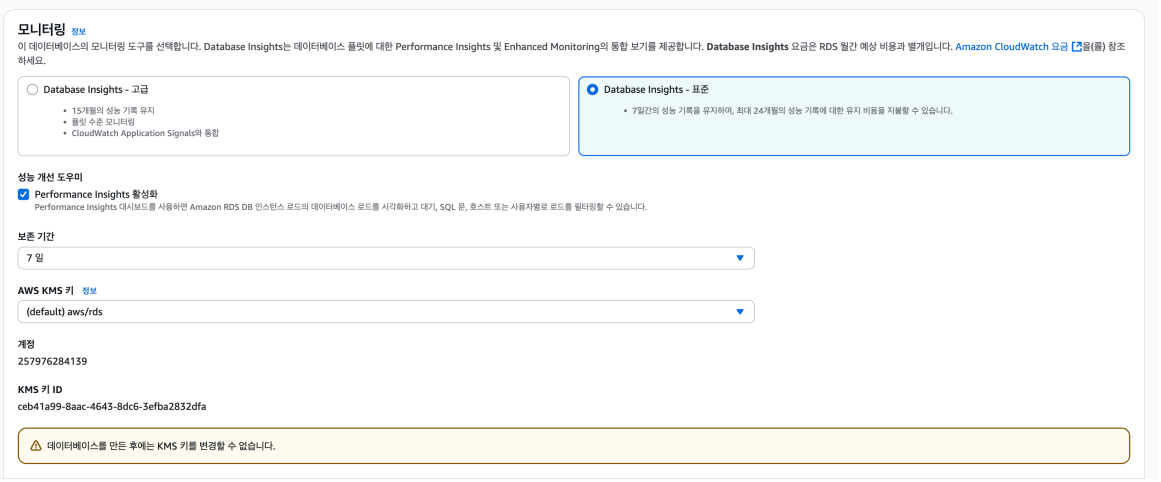
2.1 Rds(PostgreSQL) 설정

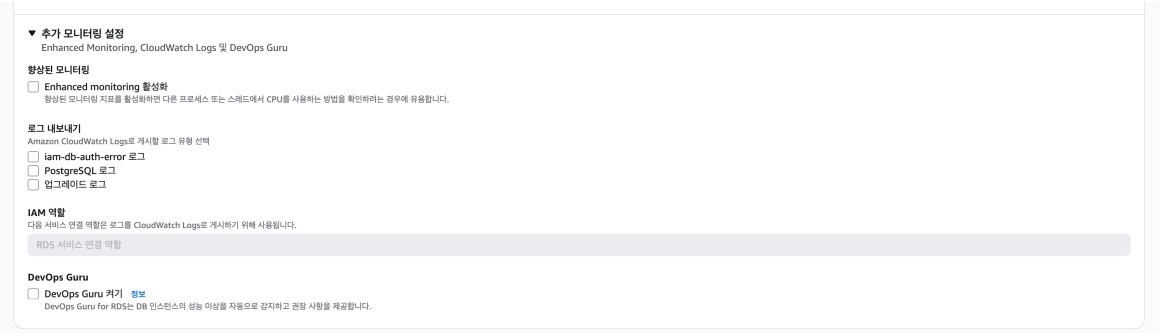
Aurora and RDS → 데이터베이스 → 데이터베이스 생성



- **데이터베이스 생성 방식** : ✅ 표준 생성
- **엔진 옵션** :
  - **엔진 유형** : PostgreSQL
  - **엔진 버전** : PostgreSQL 17.4-R1
  - **RDS 확장 지원 활성화** : ❌
- **템플릿** : ✅ 프리 티어 (반드시)
- **가용성 및 내구성** : 프리 티어로 설정하면 `단일 AZ DB 인스턴스 배포(인스턴스 1개)` 밖에 선택 못한다.
- **설정** :
  - **DB 인스턴스 식별자** : test-rds (원하는 이름으로 하면 됩니다.)
  - **자격 증명 설정** :
    - **마스터 사용자 이름** : root (원하는 이름으로 하면 됩니다.)
    - **자격 증명 관리** : ✅ 자체관리
    - **마스터 암호** : shop1011! (원하는 비밀번호로 설정)
    - **마스터 암호 확인** : shop1011! (마스터 암호에서 입력했던거 그대로 입력)
- **인스턴스 구성** :

    - ✅ 버스터블 클래스(t 클래스 포함
    - ✅ db.t4g.micro (free tier라 선택할 수 있는 인스턴스가 한정적이다.)
- **스토리지 :**
    - **스토리지 유형 :** ✅ 범용 SSD(gp2)
    - **할당된 스토리지 :** 20
    - **추가 스토리지 :** 안 건드려도 됨
- **연결 :**
    - **컴퓨팅 리소스** : ✅ EC2 컴퓨팅 리소스에 연결 안함
    - **네트워크 유형** : ✅ IPv4
    - **Virtual Private Cloud(VPC) ←제일 중요함 :** 1.1 에서 작성한 VPC 선택
    - **DB 서브넷 그룹 :** 새 DB 서브넷 그룹 형성
    - **퍼블릭 엑세스** : ✅ 예
    - **VPC 보안 그룹 :** 1.2.1 **에서 생성한 백엔드 보안 그룹 선택**
    - **가용 영역** : 기본설정 없음
    - **RDS 프록시 :** ❌ RDS 프록시 생성 선택 안함
    - **인증 기관 :** rds-ca-rsa2048-g1(default)
    - **추가 구성** : 데이터베이스 포트 = 5432
- **태그** : 건들거 없음
- 데이트베이스 인증 : ✅ 암호인증
- **모니터링** : 건들거 없음



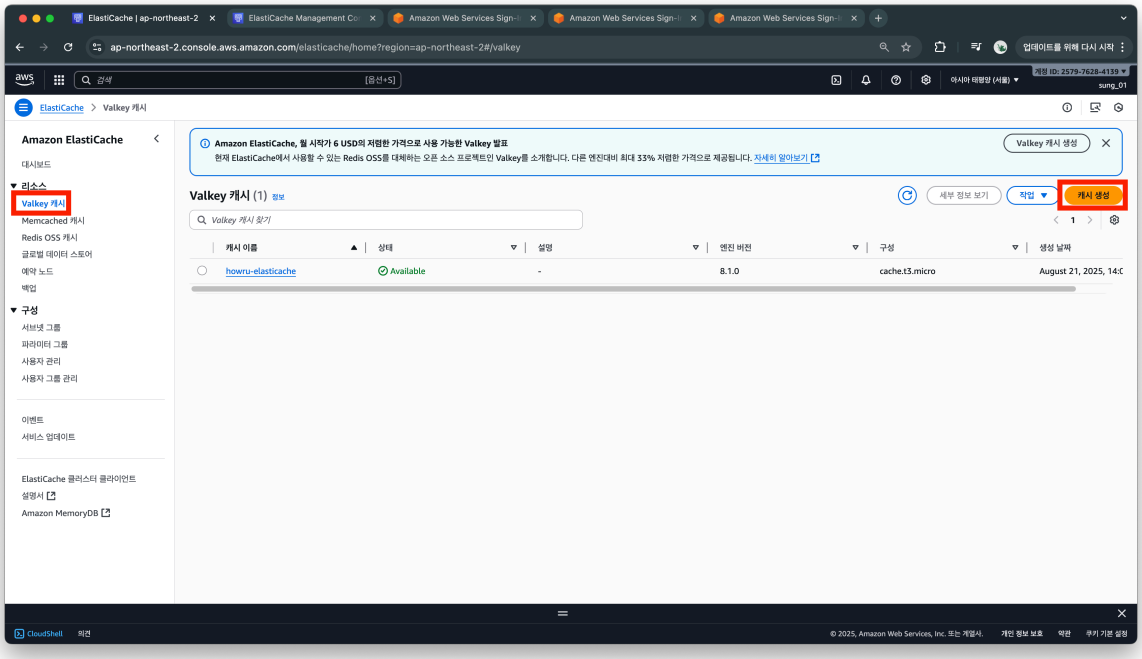


- **추가 구성 :**
  - **데이터베이스 옵션 :**
    - **초기 데이터베이스 이름 :** test (변경 가능)
    - **DB 파라미터 그룹 :** default.postgres17
  - **백업 :** (여기서부터도 설정하기 나름 안 건드려도 됨)
    - ✅ 자동 백업 활성화합니다.
    - **백업 보존 기간 : 1일**
    - **백업 기간 :** 기본 설정 없음
    - ✅ 스냅샷으로 테그 복사
    - **백업 복제** : ✅ 암호화 활성화
  - **AWS KMS 키 :** (default) aws/rds
  - **유지 관리 :**
    - ✅ 마이너 버전 자동 업그레이드 사용
    - **유지 관리 기간** : ✅ 기본 설정 없음
    - ❌ 삭제 방지 활성화

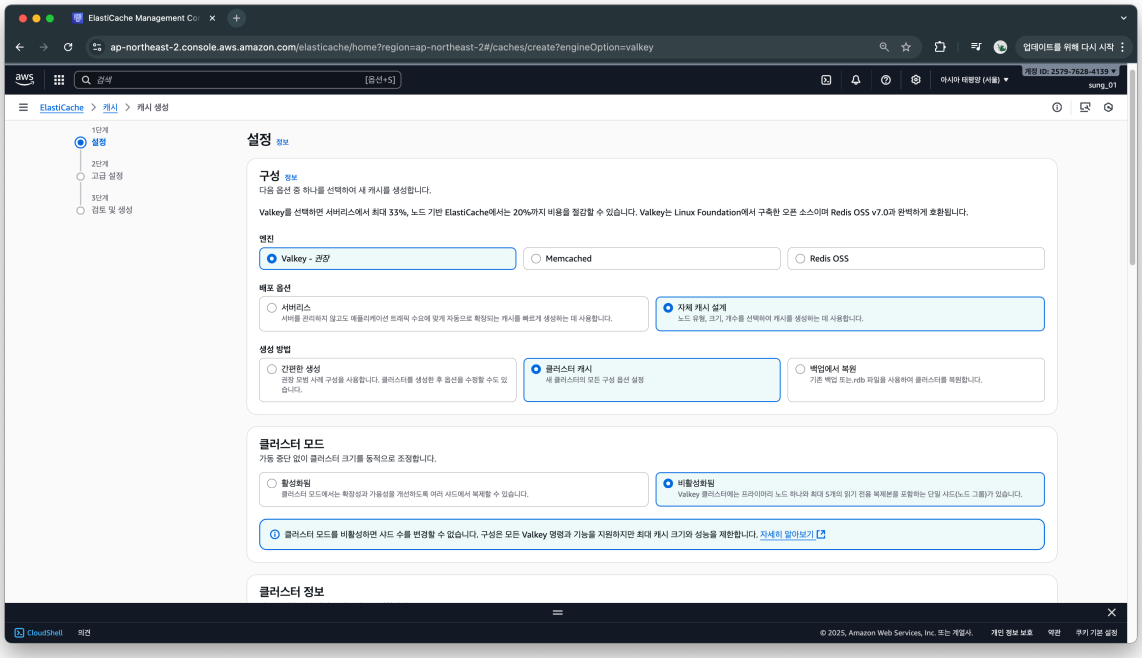
데이터베이스 생성 클릭

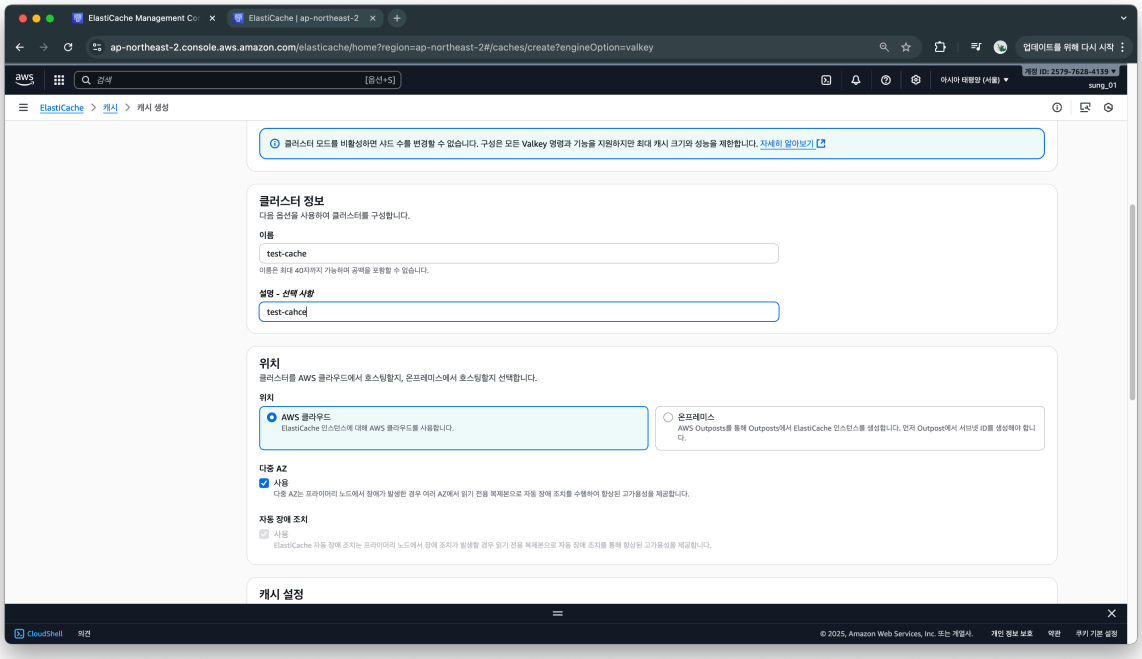
2.2 ElastiCache(Redis) 생성

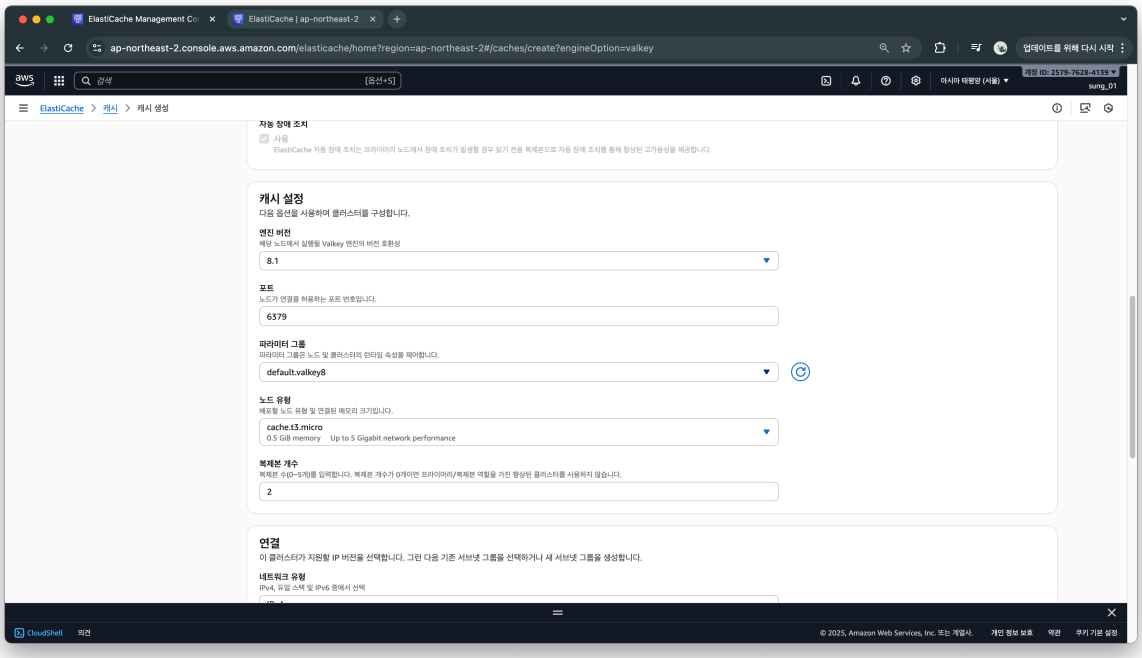
Aws Console에서 `ElastiCache → Valkey 캐시 → 캐시 생성`

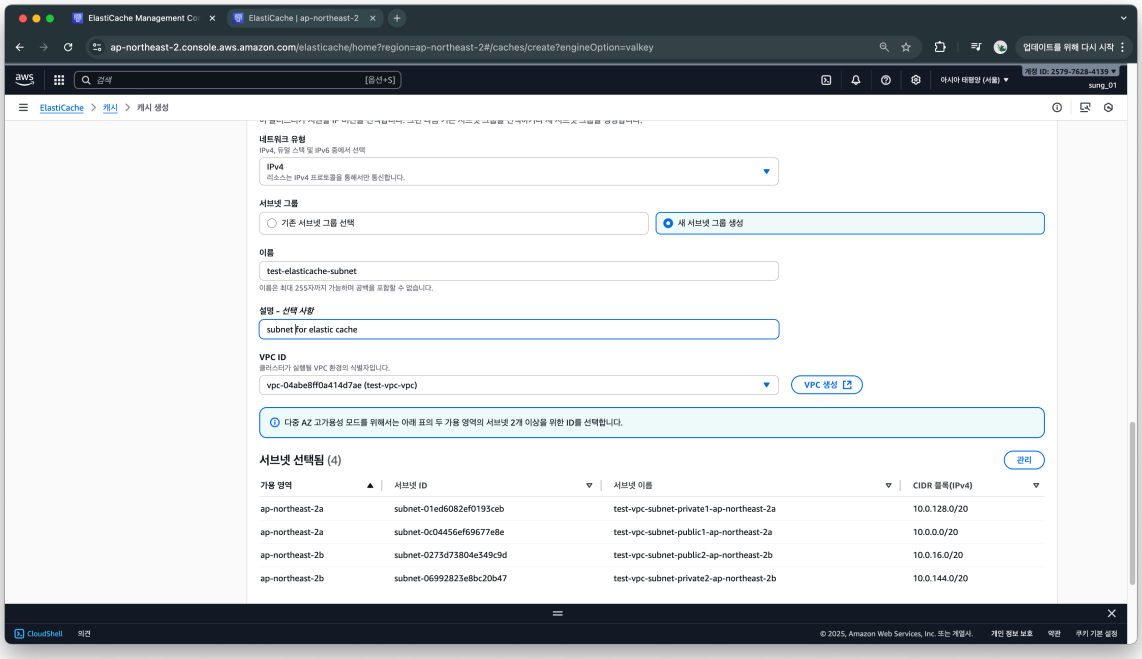

ElastiCache | ap-northeast-2
ap-northeast-2.console.aws.amazon.com/elasticache/home?region=ap-northeast-2#/valkey
Amazon ElastiCache
대시보드
리소스
Valkey 캐시
Memcached 캐시
Redis OSS 캐시
글로벌 데이터 스토어
예약 노드
백업
구성
서브넷 그룹
파라미터 그룹
사용자 관리
사용자 그룹 관리
이벤트
서비스 업데이트
ElastiCache 클러스터 클라이언트
설명서
Amazon MemoryDB
Amazon ElastiCache, 월 시작가 6 USD의 저렴한 가격으로 사용 가능한 Valkey 발표
현재 ElastiCache에서 사용할 수 있는 Redis OSS를 대체하는 오픈 소스 프로젝트인 Valkey를 소개합니다. 다른 엔진대비 최대 33% 저렴한 가격으로 제공됩니다. 자세히 알아보기
Valkey 캐시 생성
Valkey 캐시 (1) 정보
Valkey 캐시 찾기
세부 정보 보기
작업
캐시 생성
캐시 이름
상태
설명
엔진 버전
구성
생성 날짜
howru-elasticache
Available
8.1.0
cache.t3.micro
August 21, 2025, 14:0
CloudShell
의견
© 2025, Amazon Web Services, Inc. 또는 계열사.
개인 정보 보호
약관
쿠키 기본 설정

2.2.1 설정


ElastiCache Management Co
ap-northeast-2.console.aws.amazon.com/elasticache/home?region=ap-northeast-2#/caches/create?engineOption=valkey
ElastiCache > 캐시 > 캐시 생성
1단계
설정
2단계
고급 설정
3단계
검토 및 생성
설정 정보
구성 정보
다음 옵션 중 하나를 선택하여 새 캐시를 생성합니다.
Valkey를 선택하면 서버리스에서 최대 33%, 노드 기반 ElastiCache에서는 20%까지 비용을 절감할 수 있습니다. Valkey는 Linux Foundation에서 구축한 오픈 소스이며 Redis OSS v7.0과 완벽하게 호환됩니다.
엔진
Valkey - 권장
Memcached
Redis OSS
배포 옵션
서버리스
서버를 관리하지 않고도 애플리케이션 트래픽 수요에 맞게 자동으로 확장되는 캐시를 빠르게 생성하는 데 사용합니다.
자체 캐시 설계
노드 유형, 크기, 개수를 선택하여 캐시를 생성하는 데 사용합니다.
생성 방법
간편한 생성
권장 모범 사례 구성을 사용합니다. 클러스터를 생성한 후 옵션을 수정할 수도 있습니다.
클러스터 캐시
새 클러스터의 모든 구성 옵션 설정
백업에서 복원
기존 백업 또는 rdb 파일을 사용하여 클러스터를 복원합니다.
클러스터 모드
가동 중단 없이 클러스터 크기를 동적으로 조정합니다.
활성화됨
비활성화됨
Valkey 클러스터에는 프라이머리 노드 하나와 최대 5개의 읽기 전용 복제본을 포함하는 단일 샤드(노드 그룹)가 있습니다.
클러스터 모드를 비활성화하면 샤드 수를 변경할 수 없습니다. 구성은 모든 Valkey 명령과 기능을 지원하지만 최대 캐시 크기와 성능을 제한합니다. 자세히 알아보기
클러스터 정보
CloudShell
의견
© 2025, Amazon Web Services, Inc. 또는 계열사.
개인 정보 보호
약관
쿠키 기본 설정


ElastiCache Management Co
ElastiCache | ap-northeast-2
ap-northeast-2.console.aws.amazon.com/elasticache/home?region=ap-northeast-2#/caches/create?engineOption=valkey
업데이트를 위해 다시 시작
검색
[옵션+S]
아시아 태평양 (서울)
sung_01
ElastiCache > 캐시 > 캐시 생성
클러스터 모드를 비활성화면 샤드 수를 변경할 수 없습니다. 구성은 모든 Valkey 명령과 기능을 지원하지만 최대 캐시 크기와 성능을 제한합니다. 자세히 알아보기
클러스터 정보
다음 옵션을 사용하여 클러스터를 구성합니다.
이름
test-cache
이름은 최대 40자까지 가능하며 공백을 포함할 수 없습니다.
설명 - 선택 사항
test-cahce
위치
클러스터를 AWS 클라우드에서 호스팅할지, 온프레미스에서 호스팅할지 선택합니다.
위치
AWS 클라우드
ElastiCache 인스턴스에 대해 AWS 클라우드를 사용합니다.
온프레미스
AWS Outposts를 통해 Outposts에서 ElastiCache 인스턴스를 생성합니다. 먼저 Outpost에서 서브넷 ID를 생성해야 합니다.
다중 AZ
사용
다중 AZ는 프라이머리 노드에서 장애가 발생한 경우 이러 AZ에서 읽기 전용 복제본으로 자동 장애 조치를 수행하여 향상된 고가용성을 제공합니다.
자동 장애 조치
사용
ElastiCache 자동 장애 조치는 프라이머리 노드에서 장애 조치가 발생할 경우 읽기 전용 복제본으로 자동 장애 조치를 통해 향상된 고가용성을 제공합니다.
캐시 설정
CloudShell
의견
© 2025, Amazon Web Services, Inc. 또는 계열사.
개인 정보 보호
약관
쿠키 기본 설정

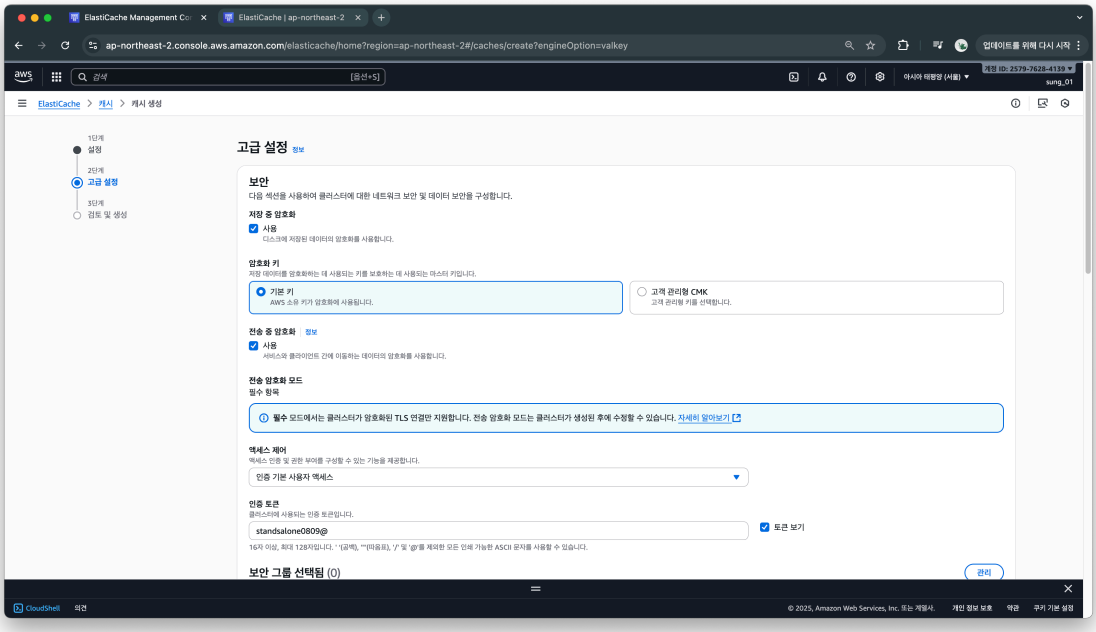

ElastiCache Management Co
ElastiCache | ap-northeast-2
ap-northeast-2.console.aws.amazon.com/elasticache/home?region=ap-northeast-2#/caches/create?engineOption=valkey
업데이트를 위해 다시 시작
검색
[옵션+S]
아시아 태평양 (서울)
sung_01
ElastiCache > 캐시 > 캐시 생성
자동 장애 조치
사용
ElastiCache 자동 장애 조치는 프라이머리 노드에서 장애 조치가 발생할 경우 읽기 전용 복제본으로 자동 장애 조치를 통해 향상된 고가용성을 제공합니다.
캐시 설정
다음 옵션을 사용하여 클러스터를 구성합니다.
엔진 버전
해당 노드에서 실행될 Valkey 엔진의 버전 호환성
8.1
포트
노드가 연결을 허용하는 포트 번호입니다.
6379
파라미터 그룹
파라미터 그룹은 노드 및 클러스터의 런타임 속성을 제어합니다.
default.valkey8
노드 유형
배포할 노드 유형 및 연결된 메모리 크기입니다.
cache.t3.micro
0.5 GiB memory Up to 5 Gigabit network performance
복제본 개수
복제본 수(0-5개)를 입력합니다. 복제본 개수가 0개이면 프라이머리/복제본 역할을 가진 향상된 클러스터를 사용하지 않습니다.
2
연결
이 클러스터가 지원할 IP 버전을 선택합니다. 그런 다음 기존 서브넷 그룹을 선택하거나 새 서브넷 그룹을 생성합니다.
네트워크 유형
IPv4, 듀얼 스택 및 IPv6 중에서 선택
CloudShell
의견
© 2025, Amazon Web Services, Inc. 또는 계열사.
개인 정보 보호
약관
쿠키 기본 설정



- 연결에서 VPC를 설정할때 반드시 전에 생성한 VPC 를 선택해야한다.
- 다음 클릭

2.2.2 고급 설정

- **엑세스 제어** : ✅ 인증 기반 사용자 엑세스 선택
- **인증 토큰 : 작성 후 반드시 다른 곳에 저장해놓기**



- **보안 그룹 : 1.2.1 에서 생성한 백엔드 보안 그룹 선택**

- 밑에 설정은 건드릴 것 없기 때문에 내려서 다음 클릭

2.2.3 검토 및 생성

생성 클릭

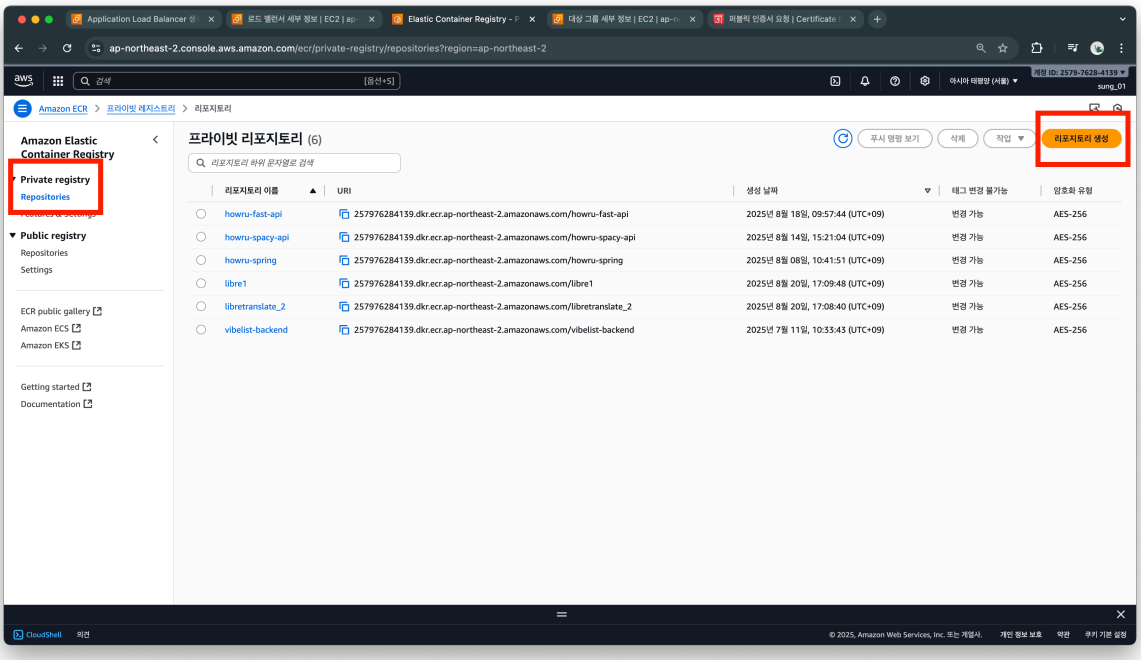
3. Github 파이프라인 구축

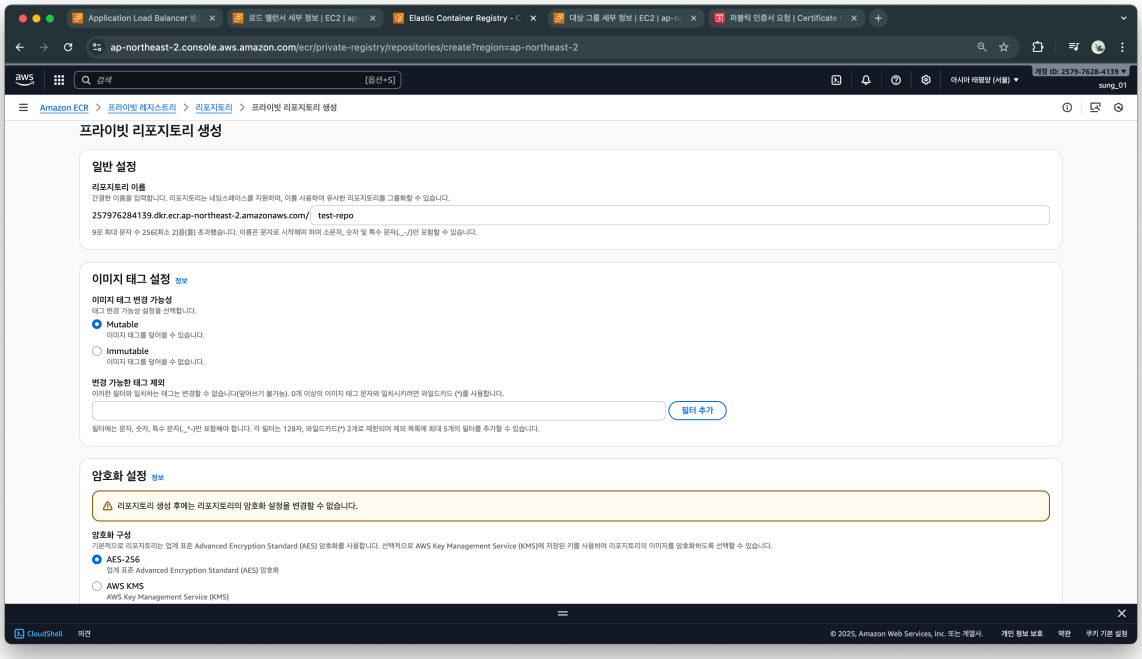
3.1 ECR 생성

3.1.1 Backend Server용 ECR 생성

Github에서 백엔드 이미지 파일을 Push할 `Repository` 를 생성해야한다.

`Amazon ECR → Private` `Registry → 리포지터리 생성`





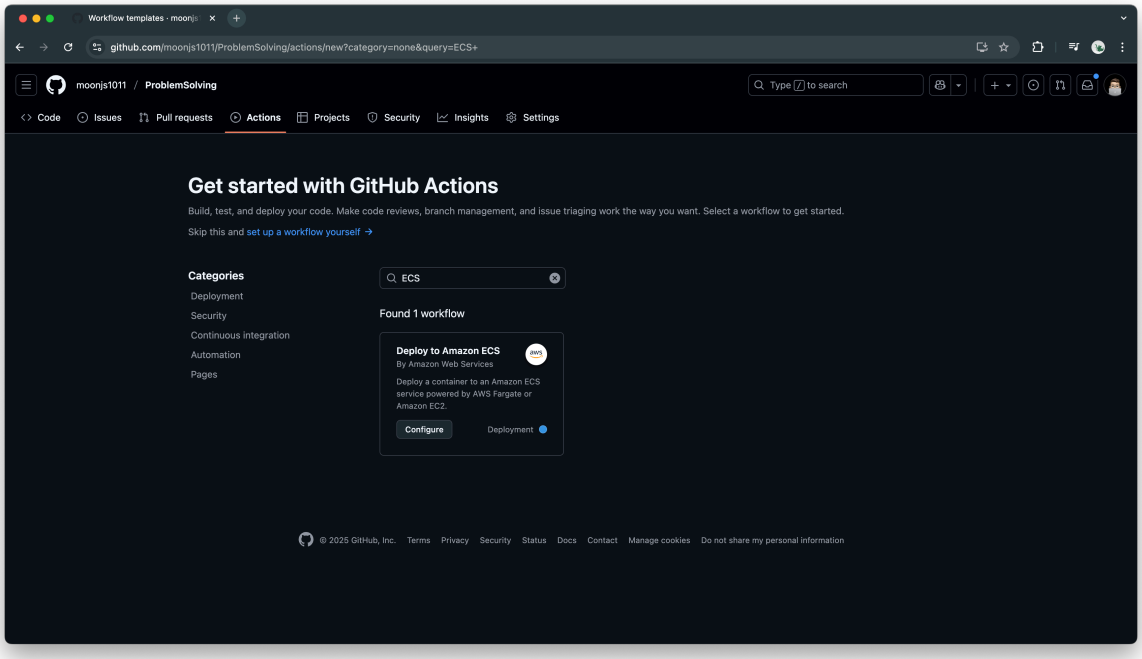
이름 설정하고

- 이미지 태크 설정 : ✅ Mutable
- 암호화 설정 : ✅ AES-256

생성 클릭

3.2 spring server용 Github deploy.yml 및 task-definition 작성

Github Action에 들어가면 ECS에 배포하는 Format이 존재한다. `Configure` 를 누르면 현재 프로젝트에 `.github/workflows/deploy.yml` 파일이 생성된다.



3.2.1 deploy.yml

그러나 나는 프로젝트 초기에 이미 만들어뒀다. 파일의 내용은 다음과 같다.

```
name: Deploy to Amazon ECS

on:
  push:
    branches: [ "main" ]

env:
  AWS_REGION:          ${{ secrets.AWS_REGION }}
  ECR_REPOSITORY:      ${{ secrets.ECR_REPO }}        # 예: howru-spring
  ECS_SERVICE:         ${{ secrets.ECS_SERVICE }}     # 예: howru-backend-service
  ECS_CLUSTER:         ${{ secrets.ECS_CLUSTER }}     # 예: howru-cluster
  ECS_TASK_DEFINITION:  infra/taskdef.json            # 리포 안의 태스크 정의 파일 경로
  CONTAINER_NAME:       spring-app                    # 태스크 정의의 컨테이너 name

permissions:
  contents: read

jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-latest
    environment: production

    steps:
```

```
      - name: Checkout
        uses: actions/checkout@v4

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id:     ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region:            ${{ env.AWS_REGION }}

      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v2

      - name: Build, tag, and push image to Amazon ECR
        id: build-image
        env:
          ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
          IMAGE_TAG:    ${{ github.sha }}
        run: |
          docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
          docker push      $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
          echo "image=$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG" >> $GITHUB_OUTP
UT

      - name: Fill env & image in task definition
        id: task-def
        uses: aws-actions/amazon-ecs-render-task-definition@v1
        with:
          task-definition: infra/taskdef.json
          container-name: spring-app
          image: ${{ steps.build-image.outputs.image }}
          environment-variables: |
            SPRING_DATASOURCE_URL=${{ secrets.SPRING_DATASOURCE_URL }}
            SPRING_DATASOURCE_USERNAME=${{ secrets.DB_USERNAME }}
            SPRING_DATASOURCE_PASSWORD=${{ secrets.DB_PASSWORD }}
            SPRING_JPA_DATABASE_PLATFORM=org.hibernate.dialect.PostgreSQLDialect
            SPRING_DATA_REDIS_HOST=${{ secrets.REDIS_HOST }}
            SPRING_DATA_REDIS_PORT=${{ secrets.REDIS_PORT }}
            SPRING_DATA_REDIS_PASSWORD=${{secrets.REDIS_PASSWORD}}
            SPRING_DATA_REDIS_SSL_ENABLED=true
            FRONT_URL=${{ secrets.FRONT_URL }}
            GOOGLE_CLIENT_ID=${{ secrets.GOOGLE_CLIENT_ID }}
            GOOGLE_CLIENT_SECRET=${{ secrets.GOOGLE_CLIENT_SECRET }}
            GEMINI_API_KEY=${{ secrets.GEMINI_API_KEY }}
            JWT_SECRET="${{ secrets.JWT_SECRET }}"
            LIBER_HOST=${{secrets.LIBER_HOST}}
```

```
          LIBER_PORT=${{secrets.LIBER_PORT}}
          NLP_BASE-URL=${{secrets.NLP_BASE_URL}}
          TAGGING-NLP_BASE-URL=${{secrets.TAGGING_NLP_BASE_URL}}
      - name: Deploy Amazon ECS task definition
        uses: aws-actions/amazon-ecs-deploy-task-definition@v1
        with:
          task-definition:              ${{ steps.task-def.outputs.task-definition }}
          service:                      howru-backend-service
          cluster:                      howr-final-cluster
          wait-for-service-stability:   true
          force-new-deployment:         true   # ✅ 강제 배포 옵션 추가
```

3.2.2 task-definition 생성

`infra/taskdef.json` 경로에 다음과 같은 파일을 생성한다.

```
{
  "family": "howru-backend", #변경 가능
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["EC2"],
  "cpu": "1024",
  "memory": "4096",
  "executionRoleArn": "arn:aws:iam::257976284139:role/ecsTaskExecutionRole", #사용자에 따라 변경
  "containerDefinitions": [
    {
      "name": "spring-app",
      "image": "257976284139.dkr.ecr.ap-northeast-2.amazonaws.com/howru-spring:latest", #사용자에 따라 변경
      "essential": true,
      "portMappings": [
        { "containerPort": 8080, "hostPort": 8080, "protocol": "tcp" }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/howru-backend",
          "awslogs-region": "ap-northeast-2",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "healthCheck": {
        "command": ["CMD-SHELL", "curl -fsS http://localhost:8080/health || exit 1"],
        "interval": 30,
        "timeout": 5,
        "retries": 3,
```

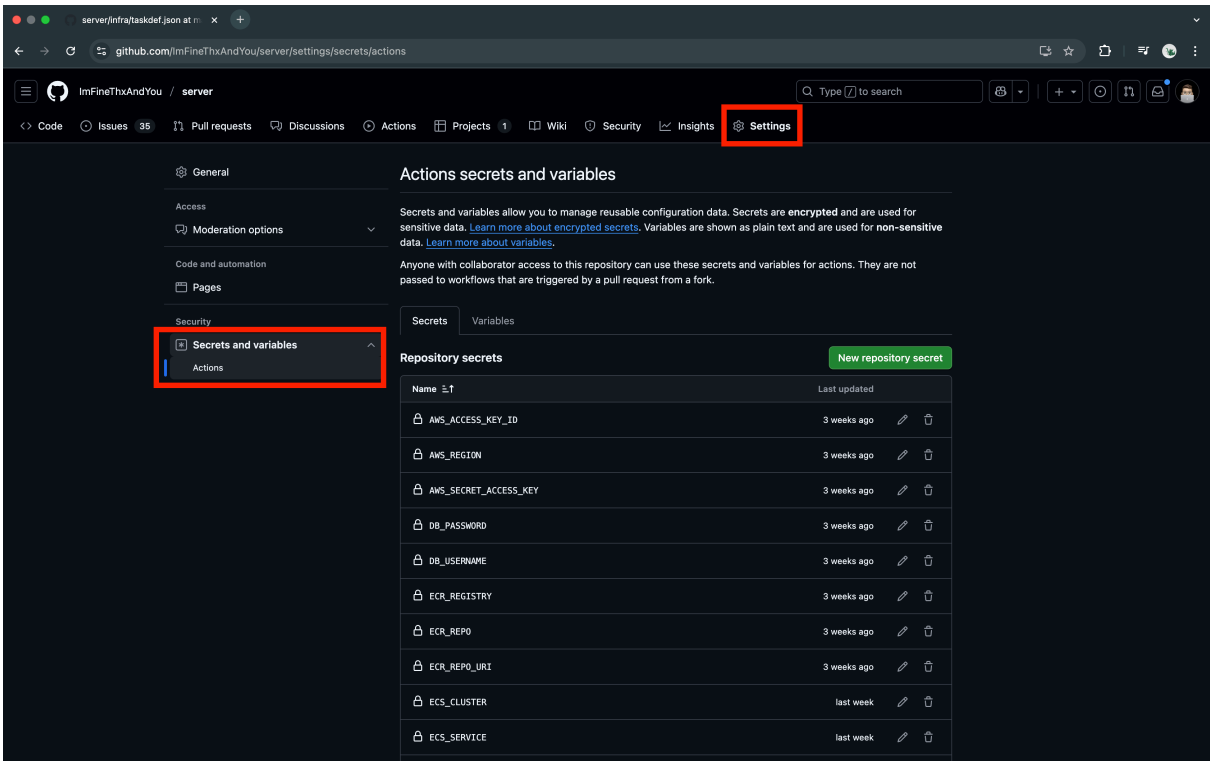
```
      "startPeriod": 60
    },
    "command": [
      "java",
      "-jar",
      "/app/app.jar"
    ]
  }
 ]
}
```

3.2.3 git secrets 생성

Git secrets은 아래의 경로에서 설정할 수 있다.

`Setting → Secrets and variables → Secrets`

New repository secret 누르면 변수가 설정 가능하다. `(한번 설정하면 확인할 수 없고, 재설정 해야하니 주의하자)`



- 설정해야할 변수 목록

GitHub Secret Key	설명 / 용도	예시 값 (형식)
AWS_ACCESS_KEY_ID	AWS IAM User의 Access Key ID	AKIAIOSFODNN7EXAMPLE
AWS_SECRET_ACCESS_KEY	AWS IAM User의 Secret Access Key	wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLE

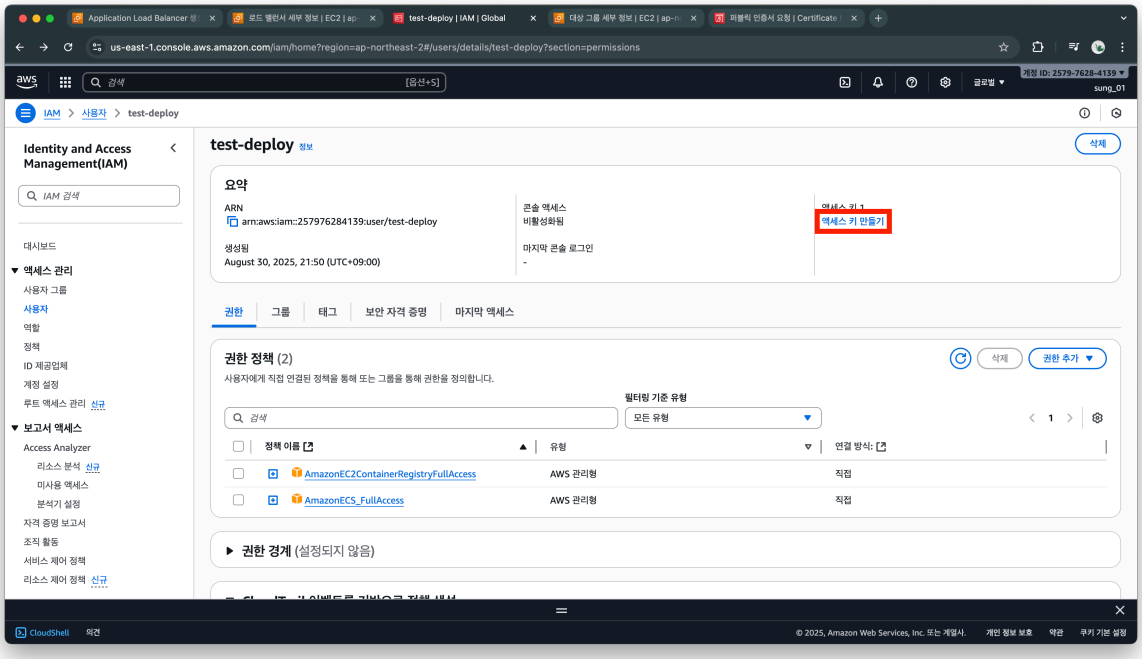
GitHub Secret Key	설명 / 용도	예시 값 (형식)
AWS_REGION	리전 설정	ap-northeast-2
ECR_REPO	ECR 저장소 이름	howru-spring
ECS_SERVICE	ECS 서비스 이름	howru-backend-service
ECS_CLUSTER	ECS 클러스터 이름	howru-cluster
SPRING_DATASOURCE_URL	DB 접속 URL	jdbc:postgresql://xxx.rds.amazonaws.com:5432/vibelist
DB_USERNAME	RDS DB 사용자명	vibelist_user
DB_PASSWORD	RDS DB 비밀번호	********
REDIS_HOST	Redis 호스트	redis.xxxxxx.ap-northeast-2.cache.amazonaws.com
REDIS_PORT	Redis 포트	6379
REDIS_PASSWORD	Redis 비밀번호	********
FRONT_URL	프론트엔드 서비스 URL	https://howareu.click
GOOGLE_CLIENT_ID	구글 OAuth Client ID	xxxxxxxx.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET	구글 OAuth Client Secret	********
GEMINI_API_KEY	Gemini API Key	********
JWT_SECRET	JWT 서명 시크릿 키	********
LIBER_HOST	LibreTranslate 서버 호스트	liber-service
LIBER_PORT	LibreTranslate 서버 포트	5050
NLP_BASE_URL	NLP 분석 서비스 Base URL	http://nlp-service:8000
TAGGING_NLP_BASE_URL	키워드 태깅 서비스 Base URL	http://tagging-nlp-service:8001

3.2.3+ 변수 확인하는 곳

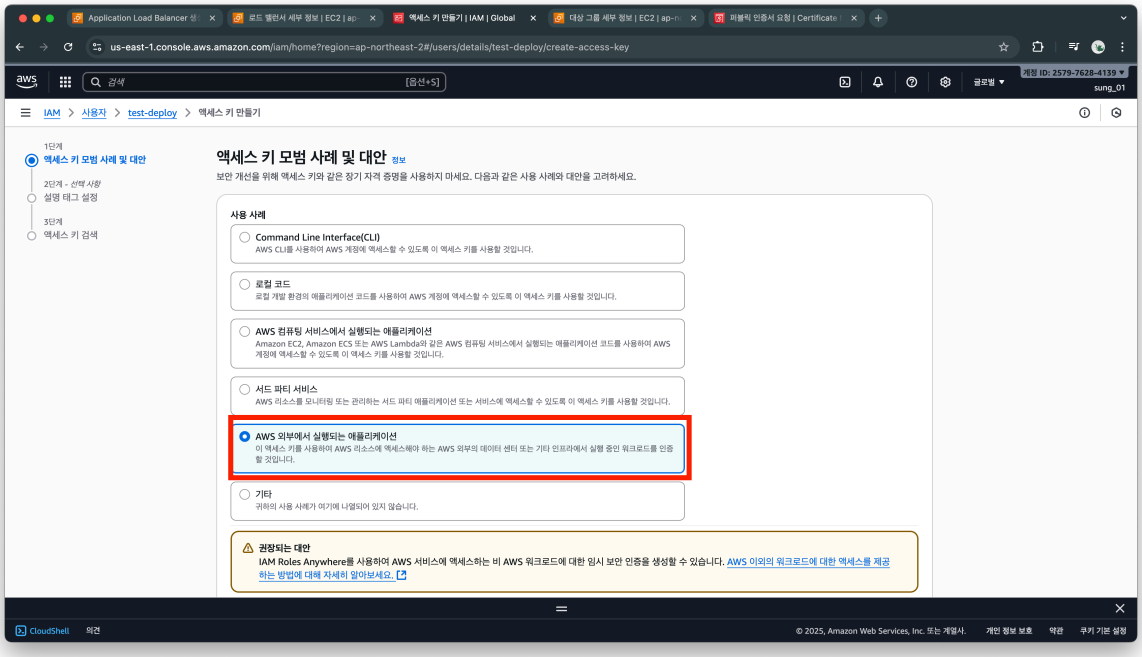
AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY

다음 누르고 사용자 생성을 누른다

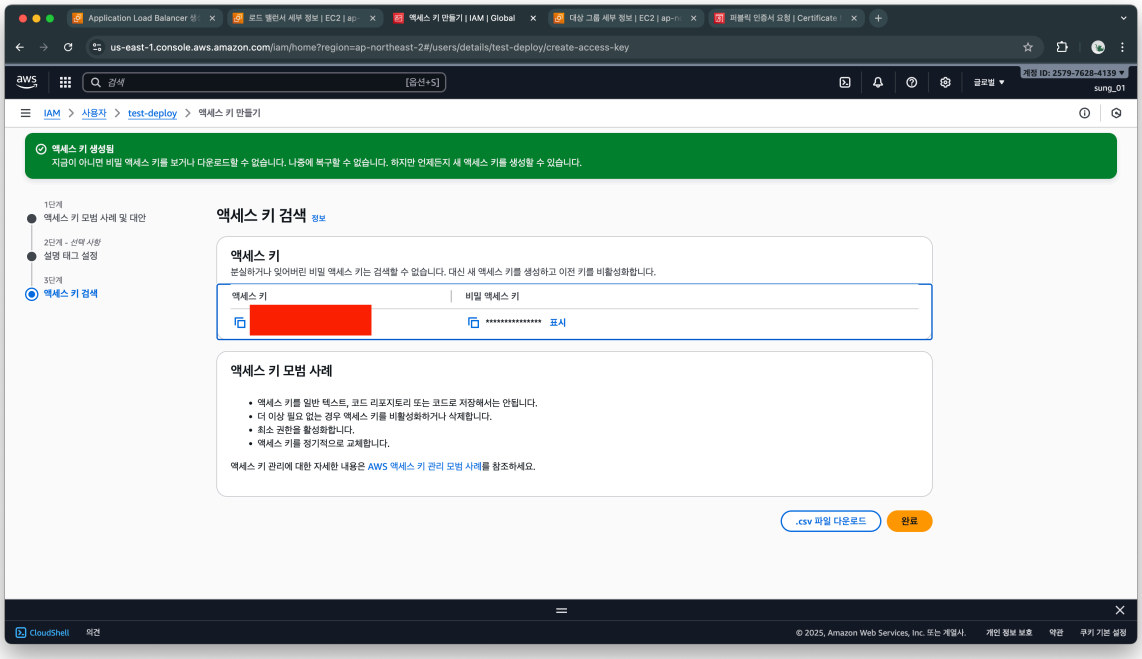
생성된 사용자를 클릭하고 `Aceess Key` 를 생성한다.



사용 사례 : ✅ Aws 외부에서 실행되는 어플리케이션 선택



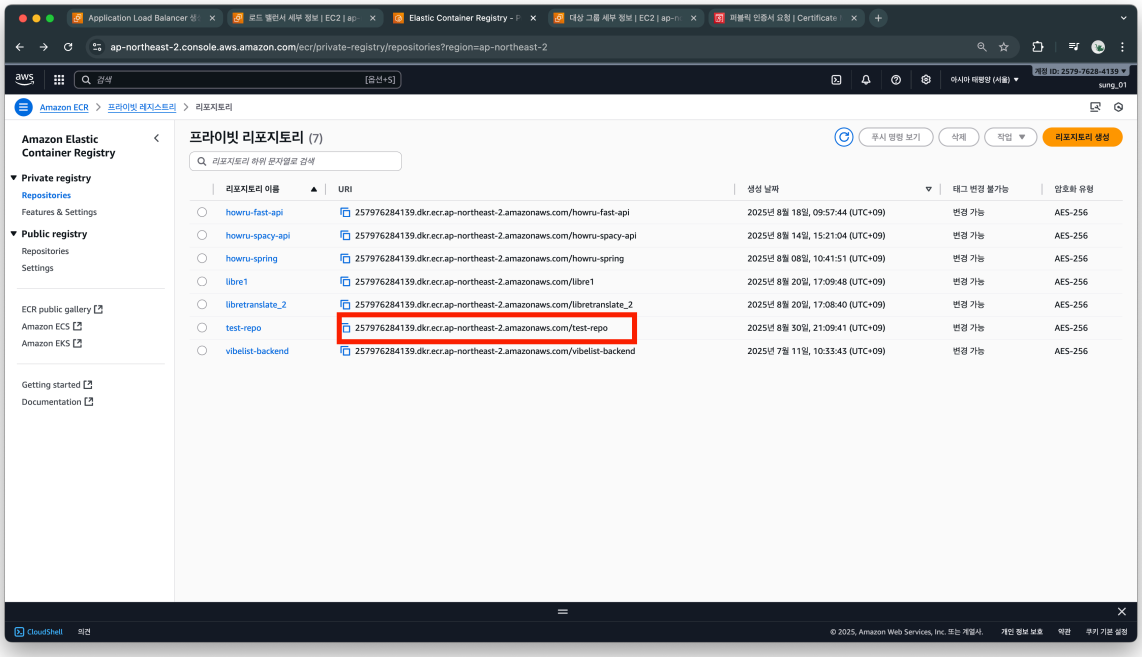
- 설명 태그 설정 는 아무것도 입력 안하고 다음 클릭



`Secret Access Key`는 다시 조회가 안되니까 어딘가에 저장해두는 것을 추천한다. 이 값들을 `git secrets`에 입력하면 된다.

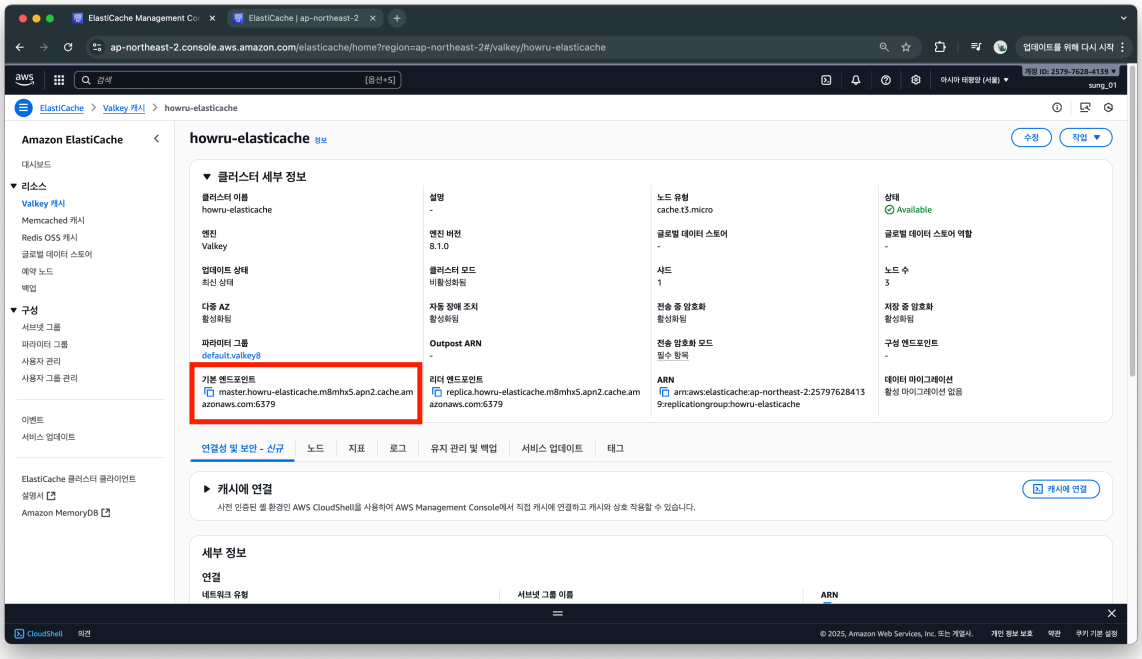
ECR Registry URL

ECR 생성하면 URL이 생성된다.



REDIS_HOST

2.2 에서 생성한 Elasticache의 기본 엔드포인트를 넣어주면 된다.



REDIS_PASSWORD

2.2.2 에서 생성한 사용자 인증 토큰을 넣으면 된다.

3.3 Spacy-api 용 Github deploy.yml 및 task-definition 작성

Github에 spacy-api Repository를 따로 빼놨다.

3.3.1 deploy.yml 작성

```
name: Deploy spacy-api (pin image tag)
on:
  push:
    branches: [ main ]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ secrets.AWS_REGION }}
```

```
    - uses: aws-actions/amazon-ecr-login@v2

    - name: Build & Push with SHA tag
      env:
        REGISTRY: ${{ secrets.ECR_REGISTRY }}
        REPO: ${{ secrets.ECR_REPO }}
        SHA: ${{ github.sha }}
      run: |
        IMAGE=$REGISTRY/$REPO:$SHA
        docker build -t $IMAGE .
        docker push $IMAGE
        echo "IMAGE=$IMAGE" >> $GITHUB_ENV

    # 현재 서비스가 쓰는 태스크 정의 이름/리비전 조회
    - name: Get current task definition ARN
      id: cur
      run: |
        ARN=$(aws ecs describe-services \
          --cluster "${{ secrets.ECS_CLUSTER }}" \
          --services "${{ secrets.ECS_SERVICE }}" \
          --query "services[0].taskDefinition" --output text)
        echo "TASKDEF_ARN=$ARN" >> $GITHUB_OUTPUT

    # 태스크 정의 JSON을 워크스페이스로 덤프
    - name: Export task definition JSON
      run: |
        aws ecs describe-task-definition \
          --task-definition "${{ steps.cur.outputs.TASKDEF_ARN }}" \
          --query "taskDefinition" > taskdef.json

    # 컨테이너 이미지 태그만 교체 (jq 필요)
    - name: Patch image in taskdef
      run: |
        sudo apt-get update && sudo apt-get install -y jq
        cat taskdef.json \
        | jq 'del(.taskDefinitionArn,.revision,.status,.requiresAttributes,.compatibilities,.registere
dAt,.registeredBy)' \
        | jq --arg IMG "${{ env.IMAGE }}" \
            '.containerDefinitions |= (map(if .name=="spacy-api" then .image=$IMG else . en
d))' \
        > taskdef-patched.json
        cat taskdef-patched.json

    # 새 리비전 등록
    - name: Register new task definition
      id: reg
      run: |
```

```
        NEW_ARN=$(aws ecs register-task-definition \
          --cli-input-json file://taskdef-patched.json \
          --query "taskDefinition.taskDefinitionArn" --output text)
        echo "NEW_TASKDEF_ARN=$NEW_ARN" >> $GITHUB_OUTPUT

    # 서비스 업데이트 (롤링 배포)
    - name: Update ECS service
      run: |
        aws ecs update-service \
          --cluster "${{ secrets.ECS_CLUSTER }}" \
          --service "${{ secrets.ECS_SERVICE }}" \
          --task-definition "${{ steps.reg.outputs.NEW_TASKDEF_ARN }}" \
          --region "${{ secrets.AWS_REGION }}"
```

3.2.2 task-definition 생성

`infra/taskdef.json` 경로에 다음과 같은 파일을 생성한다.

```
{
  "family": "howru-spacy-api",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["EC2"],
  "cpu": "256",
  "memory": "512",
  "executionRoleArn": "arn:aws:iam::257976284139:role/ecsTaskExecutionRole",
  "containerDefinitions": [
    {
      "name": "spacy-api",
      "image": "257976284139.dkr.ecr.ap-northeast-2.amazonaws.com/howru-spacy-api:lates
t",
      "essential": true,
      "portMappings": [
        { "containerPort": 8000, "hostPort": 8000, "protocol": "tcp" }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/spacy-api",
          "awslogs-region": "ap-northeast-2",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "healthCheck": {
        "command": ["CMD-SHELL", "curl -fsS http://localhost:8000/health || exit 1"],
        "interval": 30,
        "timeout": 5,
        "retries": 3,
```

```
      "startPeriod": 60
    },
    "command": [
      "python",
      "app.py"
    ]
   }
  ]
}
```

3.3.3 git secrets 생성

- 설정해야할 변수 목록

GitHub Secret Key	설명 / 용도	예시 값 (형식)
AWS_ACCESS_KEY_ID	AWS IAM User의 Access Key ID	AKIAIOSFODNN7EXAMPLE
AWS_SECRET_ACCESS_KEY	AWS IAM User의 Secret Access Key	wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLE
AWS_REGION	AWS 리전 설정	ap-northeast-2
ECR_REGISTRY	AWS ECR 레지스트리 주소	123456789012.dkr.ecr.ap-northeast-2.amazonaws.com
ECR_REPO	AWS ECR 저장소 이름	spacy-api
ECS_CLUSTER	ECS 클러스터 이름	spacy-cluster
ECS_SERVICE	ECS 서비스 이름	spacy-service

3.4 fast-api 용 Github deploy.yml 및 task-definition 작성

Github에 fast-api Repository를 따로 빼놨다.

3.4.1 deploy.yml 작성

```
name: Deploy fast-api (pin image tag)
on:
  push:
    branches: [ main ]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
```

```
        aws-region: ${{ secrets.AWS_REGION }}

    - uses: aws-actions/amazon-ecr-login@v2

    - name: Build & Push with SHA tag
      env:
        REGISTRY: ${{ secrets.ECR_REGISTRY }}
        REPO: ${{ secrets.ECR_REPO }}
        SHA: ${{ github.sha }}
      run: |
        IMAGE=$REGISTRY/$REPO:$SHA
        docker build -t $IMAGE .
        docker push $IMAGE
        echo "IMAGE=$IMAGE" >> $GITHUB_ENV

    # 현재 서비스가 쓰는 태스크 정의 이름/리비전 조회
    - name: Get current task definition ARN
      id: cur
      run: |
        ARN=$(aws ecs describe-services \
          --cluster "${{ secrets.ECS_CLUSTER }}" \
          --services "${{ secrets.ECS_SERVICE }}" \
          --query "services[0].taskDefinition" --output text)
        echo "TASKDEF_ARN=$ARN" >> $GITHUB_OUTPUT

    # 태스크 정의 JSON을 워크스페이스로 덤프
    - name: Export task definition JSON
      run: |
        aws ecs describe-task-definition \
          --task-definition "${{ steps.cur.outputs.TASKDEF_ARN }}" \
          --query "taskDefinition" > taskdef.json

    # 컨테이너 이미지 태그만 교체 (jq 필요)
    - name: Patch image in taskdef
      run: |
        sudo apt-get update && sudo apt-get install -y jq
        cat taskdef.json \
        | jq 'del(.taskDefinitionArn,.revision,.status,.requiresAttributes,.compatibilities,.registere
dAt,.registeredBy)' \
        | jq --arg IMG "${{ env.IMAGE }}" \
            '.containerDefinitions |= (map(if .name=="spacy-api" then .image=$IMG else . en
d))' \
        > taskdef-patched.json
        cat taskdef-patched.json

    # 새 리비전 등록
    - name: Register new task definition
```

```
      id: reg
      run: |
        NEW_ARN=$(aws ecs register-task-definition \
          --cli-input-json file://taskdef-patched.json \
          --query "taskDefinition.taskDefinitionArn" --output text)
        echo "NEW_TASKDEF_ARN=$NEW_ARN" >> $GITHUB_OUTPUT

    # 서비스 업데이트 (롤링 배포)
    - name: Update ECS service
      run: |
        aws ecs update-service \
          --cluster "${{ secrets.ECS_CLUSTER }}" \
          --service "${{ secrets.ECS_SERVICE }}" \
          --task-definition "${{ steps.reg.outputs.NEW_TASKDEF_ARN }}" \
          --region "${{ secrets.AWS_REGION }}"
```

3.4.2 task-definition 생성

`infra/taskdef.json` 경로에 다음과 같은 파일을 생성한다.

```
{
  "family": "howru-fast-api",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["EC2"],
  "cpu": "256",
  "memory": "512",
  "executionRoleArn": "arn:aws:iam::257976284139:role/ecsTaskExecutionRole",
  "containerDefinitions": [
    {
      "name": "fast-api",
      "image": "257976284139.dkr.ecr.ap-northeast-2.amazonaws.com/howru-fast-api:latest",
      "essential": true,
      "portMappings": [
        { "containerPort": 8001, "hostPort": 8001, "protocol": "tcp" }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/fast-api",
          "awslogs-region": "ap-northeast-2",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "healthCheck": {
        "command": ["CMD-SHELL", "curl -fsS http://localhost:8001/health || exit 1"],
        "interval": 30,
        "timeout": 5,
```

```
      "retries": 3,
      "startPeriod": 60
    },
    "command": [
      "python",
      "analysisTag.py"
    ]
   }
  ]
}
```

3.4.3 git secrets 생성

- 설정해야할 변수 목록

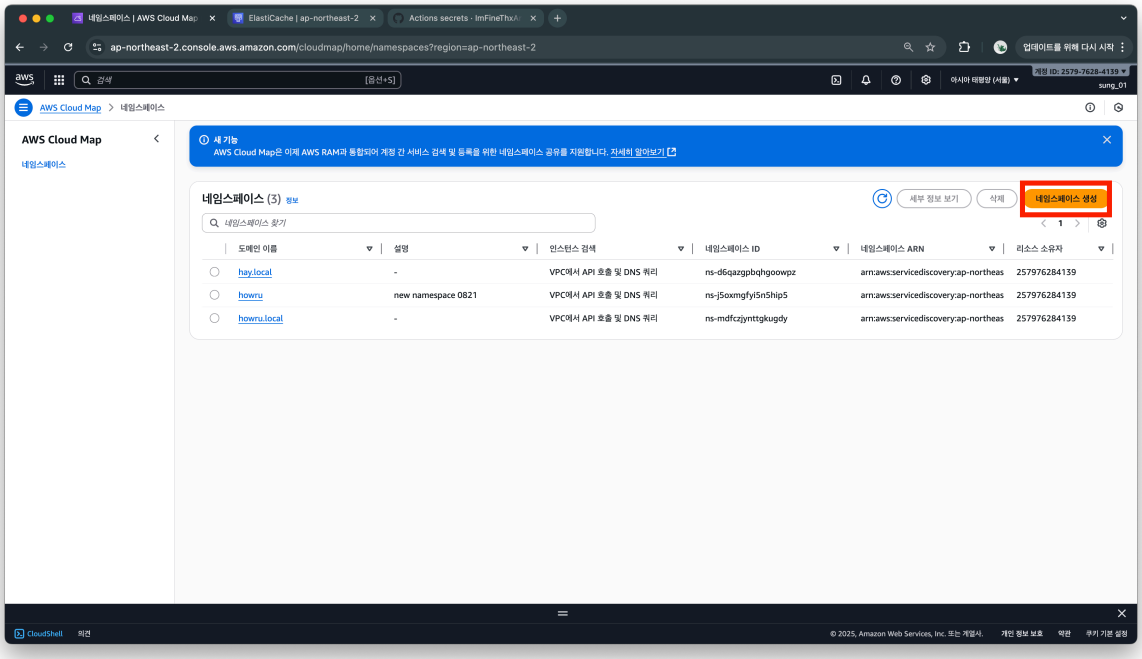
GitHub Secret Key	설명 / 용도	예시 값 (형식)
AWS_ACCESS_KEY_ID	AWS IAM User의 Access Key ID	AKIAIOSFODNN7EXAMPLE
AWS_SECRET_ACCESS_KEY	AWS IAM User의 Secret Access Key	wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLE
AWS_REGION	AWS 리전 설정	ap-northeast-2
ECR_REGISTRY	AWS ECR 레지스트리 주소	123456789012.dkr.ecr.ap-northeast-2.amazonaws.com
ECR_REPO	AWS ECR 저장소 이름	spacy-api
ECS_CLUSTER	ECS 클러스터 이름	spacy-cluster
ECS_SERVICE	ECS 서비스 이름	spacy-service

4. ECS

4.1 CloudMap 설정

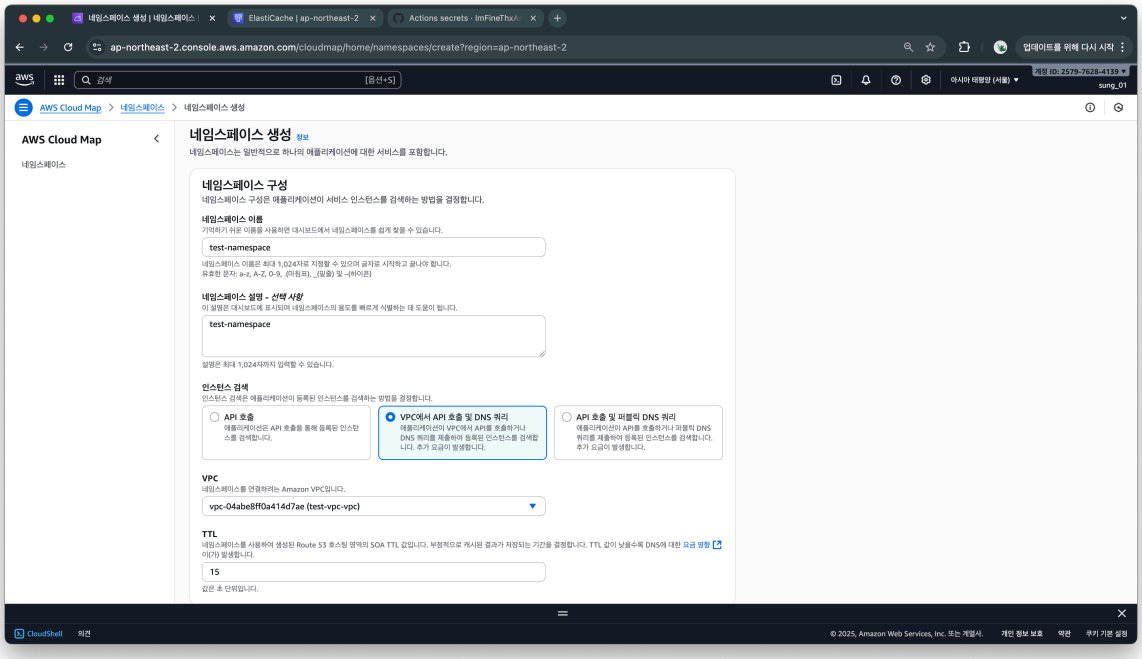
4.1.1 namespace 생성

AWS Cloud Map → 네임스페이스→ 네임스페이스 생성



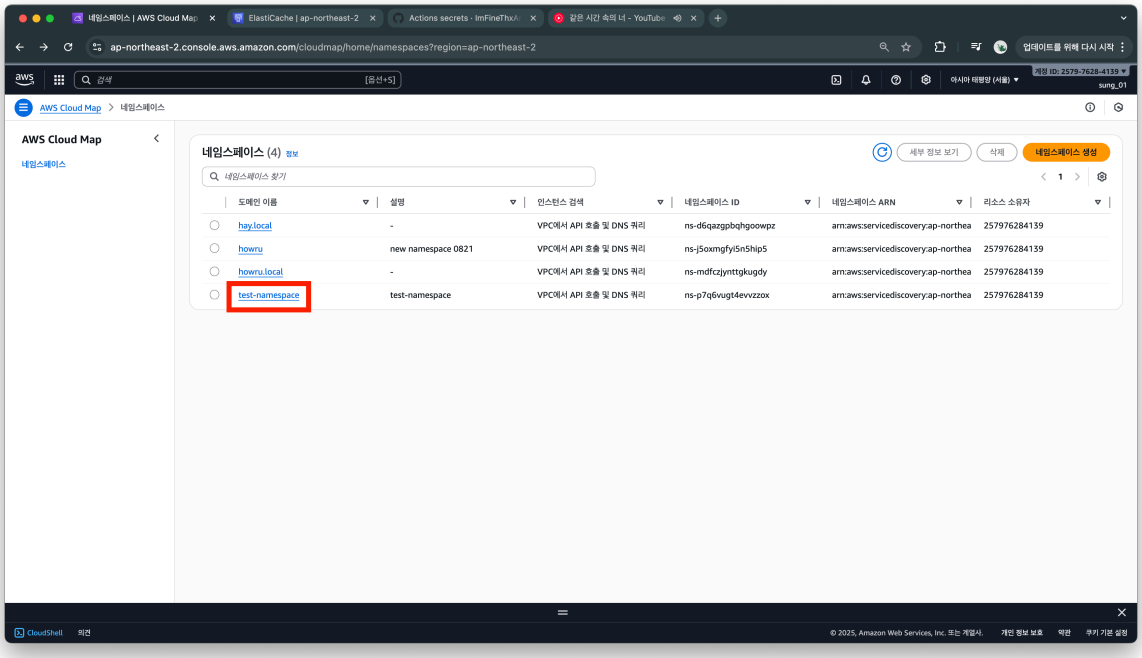
- **네임스페이스 생성 : 임의로 설정 가능**
- **네임스페이스 설명** : 생략 가능
- **인스턴스 검색** : ✅ VPC에서 API 호출 및 DNS 쿼리
- **VPC** : 1.1 에서 생성한 VPC로 설정
- **TTL** : 15
- **태그 :** 설정 안해도 된다.

→ 네임스페이스 생성

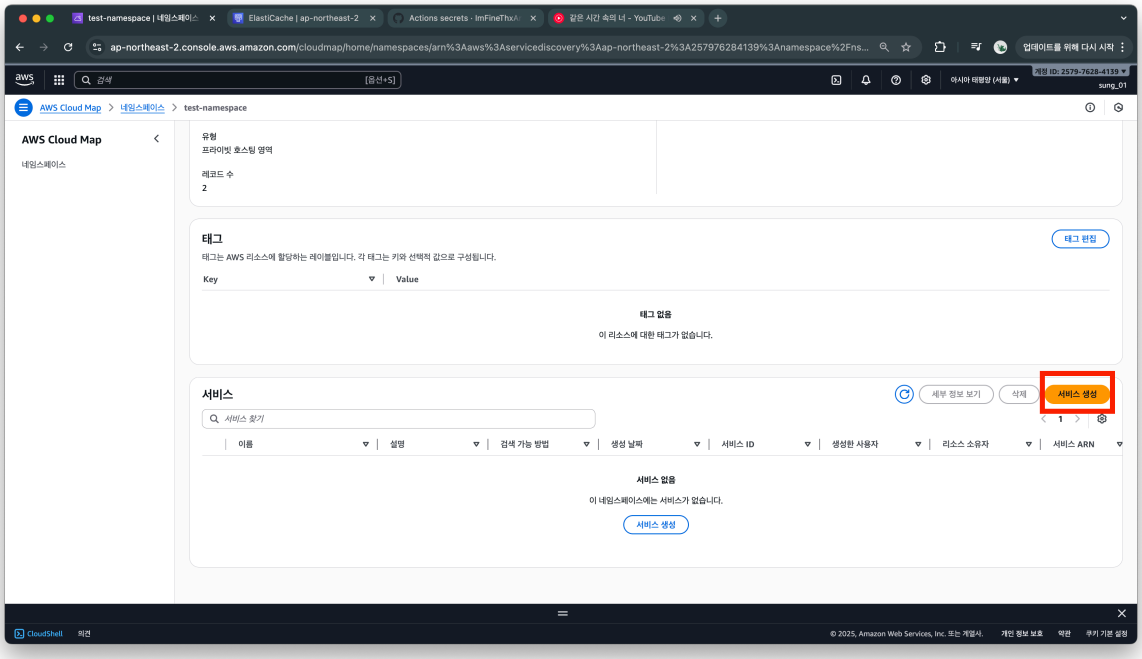


4.1.2 서비스 생성

4.1.1 에서 생성한 네임스페이스 클릭

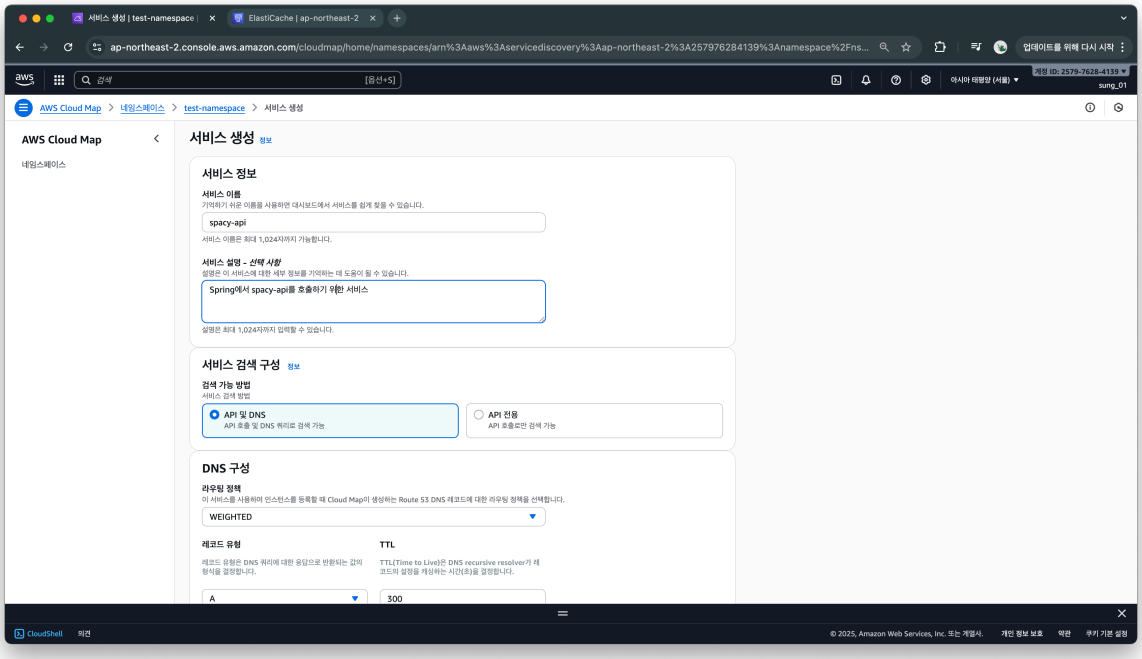


밑으로 스크롤 후 **서비스 생성 클릭**



- **서비스 정보**
  - **서비스 이름** : spacy-api
  - **서비스 설명** : Spring에서 spacy-api를 호출하기 위한 서비스
- **서비스 검색 구성 :** ✅ API 및 DNS
- **DNS 구성**
  - **라우팅 정책 : WEIGHTED**
  - **레코드 유형** : A
  - **TTL** : 300
- **상태 확인 구성**
  - **상태 확인 옵션 :** ✅ **상태 확인 없음**

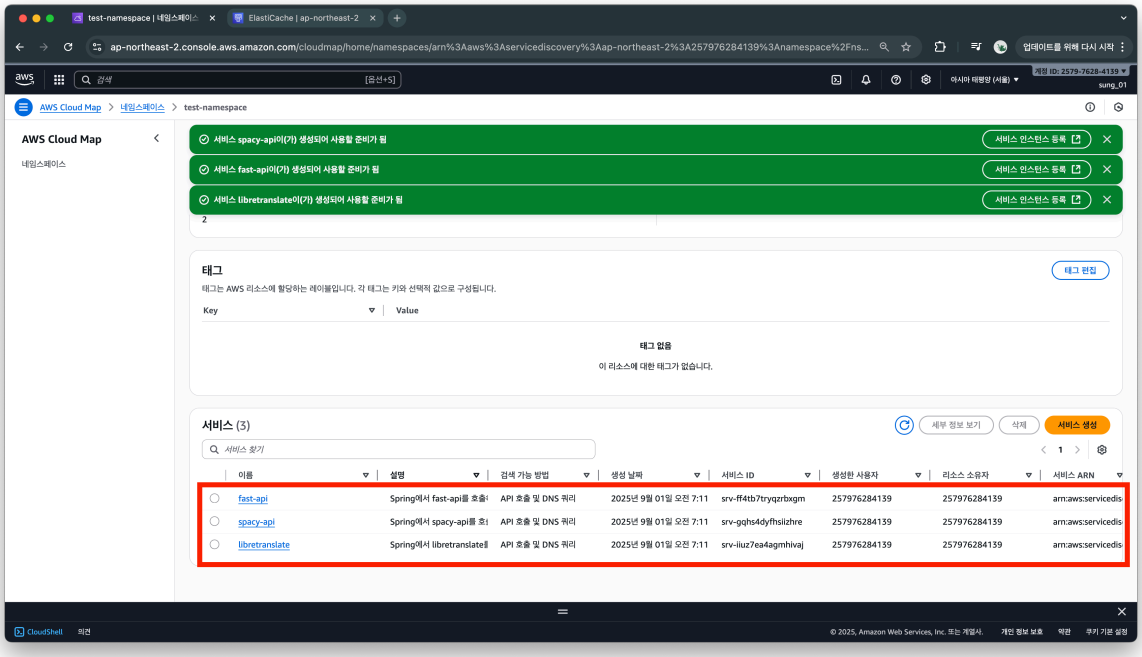
→ 서비스 생성 클릭



이후 fast-api와 libretranslate 서비스 똑같이 생성하면 된다.

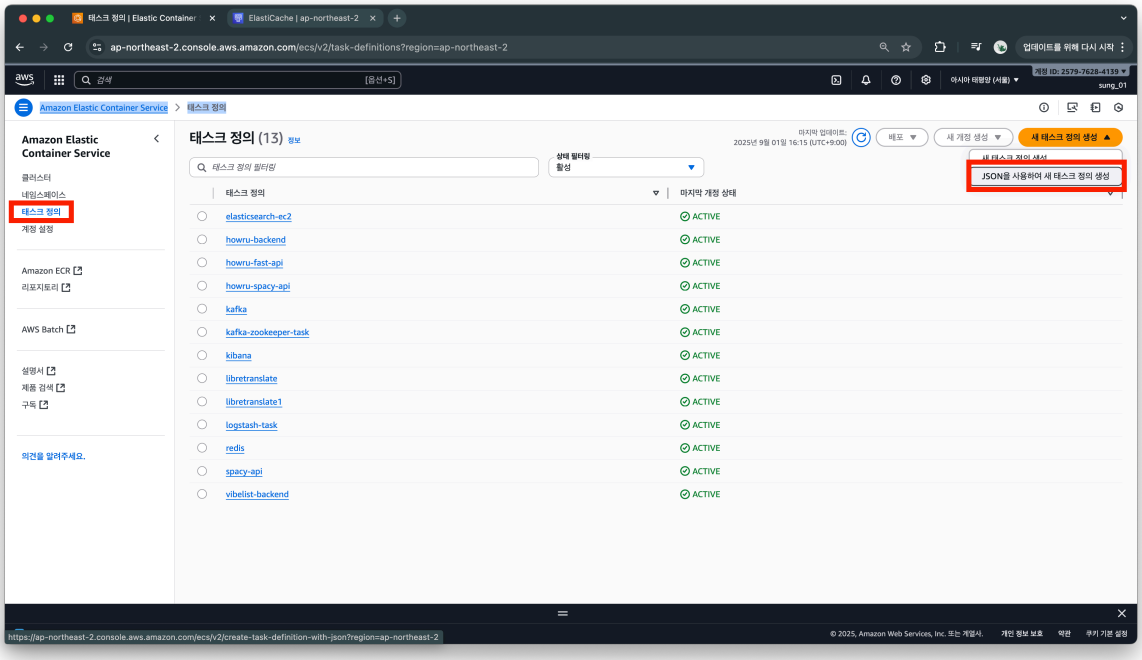
서비스 이름만 변경해서 생성하면 된다.

생성한 네임스페이스에 아래와 같이 3개의 서비스가 등록되있으면 된다.



4.2 Task Definition

Amazon Elastic Container Service → 태스크 정의 → 새 태스크 정의 생성 → JSON을 사용하여 새태스크 정의 생성



그 후, `4.2.1` , `4.2.2` , `4.2.3` 의 json 파일을 3번 복사 붙여넣기 한다.

`family` 는 임의로 정하고, `image` URL은 ECR의 URL로 설정하면 된다.

4.2.1 fast-api task_defintion.json

```
{
   "family": "test-fast-api",
   "containerDefinitions": [
      {
         "command": [
            "python",
            "analysisTag.py"
         ],
         "cpu": 0,
         "environment": [],
         "essential": true,
         "healthCheck": {
            "command": [
               "CMD-SHELL",
               "curl -fsS http://localhost:8001/health || exit 1"
            ],
            "interval": 30,
            "retries": 10,
            "startPeriod": 120,
            "timeout": 5
         },
         "image": "257976284139.dkr.ecr.ap-northeast-2.amazonaws.com/howru-fast-api:718
aa1d96bcf2a849123a0c92b38d6d5b5a6b9cc",
```

```
        "logConfiguration": {
          "logDriver": "awslogs",
          "options": {
            "awslogs-group": "/ecs/fast-api",
            "awslogs-region": "ap-northeast-2",
            "awslogs-stream-prefix": "ecs"
          }
        },
        "mountPoints": [],
        "name": "fast-api",
        "portMappings": [
          {
            "containerPort": 8001,
            "hostPort": 8001,
            "protocol": "tcp"
          }
        ],
        "systemControls": [],
        "volumesFrom": []
      }
    ],
    "executionRoleArn": "arn:aws:iam::257976284139:role/ecsTaskExecutionRole",
    "networkMode": "awsvpc",
    "volumes": [],
    "placementConstraints": [],
    "requiresCompatibilities": [
      "EC2"
    ],
    "cpu": "1024",
    "memory": "4096"
}
```

4.2.2 spacy-api task_defintion.json

```
{
    "family": "test-spacy-api",
    "containerDefinitions": [
      {
        "command": [
          "python",
          "app.py"
        ],
        "cpu": 0,
        "environment": [],
        "essential": true,
        "healthCheck": {
          "command": [
```

```
                "CMD-SHELL",
                "curl -fsS http://localhost:8000/health || exit 1"
            ],
            "interval": 30,
            "retries": 3,
            "startPeriod": 60,
            "timeout": 5
        },
        "image": "257976284139.dkr.ecr.ap-northeast-2.amazonaws.com/howru-spacy-api:3
3346396efeb9c68dfe3f2b339b41c633e2b96b1",
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/spacy-api",
                "awslogs-region": "ap-northeast-2",
                "awslogs-stream-prefix": "ecs"
            }
        },
        "mountPoints": [],
        "name": "spacy-api",
        "portMappings": [
            {
                "containerPort": 8000,
                "hostPort": 8000,
                "protocol": "tcp"
            }
        ],
        "systemControls": [],
        "volumesFrom": []
    }
  ],
  "executionRoleArn": "arn:aws:iam::257976284139:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc",
  "volumes": [],
  "placementConstraints": [],
  "requiresCompatibilities": [
    "EC2"
  ],
  "cpu": "1024",
  "memory": "3072"
}
```

4.2.3 libretranslate task_defintion.json

```
{
  "family": "test-libretranslate",
  "containerDefinitions": [
```

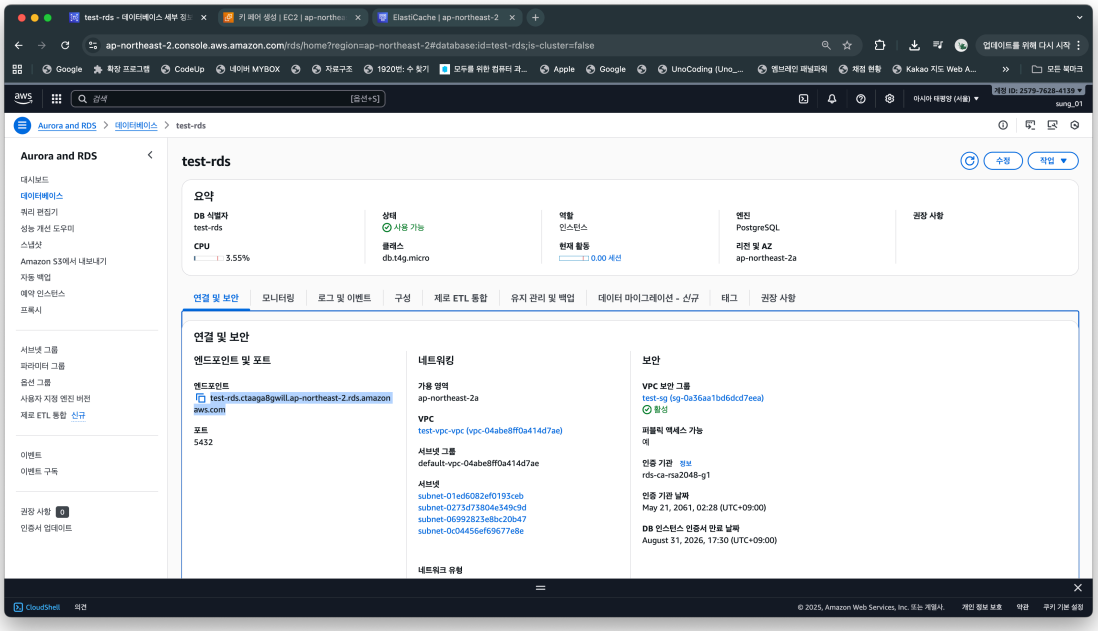
```
    {
      "command": [
        "--host",
        "0.0.0.0",
        "--port",
        "5050",
        "--load-only",
        "en,ko",
        "--debug"
      ],
      "cpu": 1024,
      "environment": [],
      "essential": true,
      "image": "257976284139.dkr.ecr.ap-northeast-2.amazonaws.com/libretranslate_2:late
st",
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/libretranslate",
          "awslogs-region": "ap-northeast-2",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "memory": 2048,
      "mountPoints": [],
      "name": "libretranslate",
      "portMappings": [
        {
          "containerPort": 5050,
          "hostPort": 5050,
          "protocol": "tcp"
        }
      ],
      "systemControls": [],
      "volumesFrom": []
    }
  ],
  "executionRoleArn": "arn:aws:iam::257976284139:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc",
  "volumes": [],
  "placementConstraints": [],
  "requiresCompatibilities": [
    "EC2"
  ],
  "cpu": "1024",
```

```
    "memory": "3072"
}
```

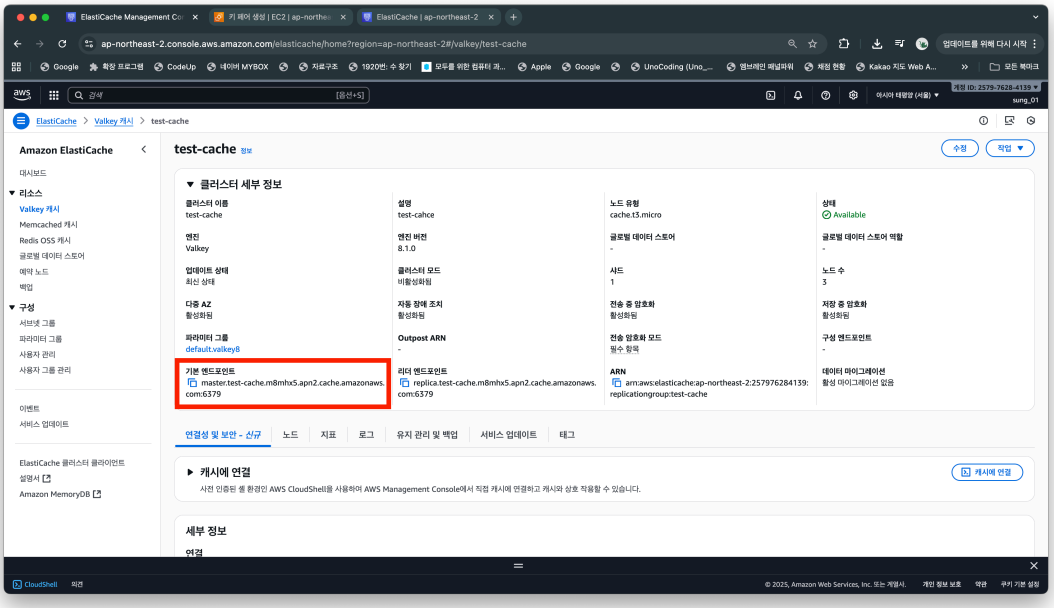
4.2.4 springboot_task_definition.json

springboot의 json파일에는 손봐줄것이 많다.

- environment :
  - **LIBER_HOST :** 4.1.2 에서 CloudMap의 서비스를 생성할 때 네임스페이스와 서비스 이름을 생성했다. 예를 들어 서비스 이름을 libretranslate 으로 생성했고, namespace의 이름을 test-namespace 로 생성했다면, LIBER_HOST는 `libretranslate.test-namespace`가 된다.
  - **NPL_BASE_URL :** `http://space-api.test-namespace:8000` 이 된다.
  - **TAGGING-NLP_BASE-URL:** `http://fast-api.test-namespace:8001` 가 된다.
  -
  - **SPRING_DATASOURCE_URL : 2.1 에서 생성한 RDS의 엔드포인트와 초기 데이터베이스 이름을 넣으면 된다. 예를 들어 엔드포인트가** `test-rds.ctaaga8gwill.ap-northeast-2.rds.amazonaws.com` 라면 **SPRING_DATASOURCE_URL에는** `jdbc:postgresql://test-rds.ctaaga8gwill.ap-northeast-2.rds.amazonaws.com/howru` 가 된다.



  - **SPRING_DATASOURCE_USERNAME : 2.1 에서 RDS를 생성할때 설정한 마스터 사용자 이름을 넣으면 된다.**
  - **SPRING_DATASOURCE_PASSWORD : 2.1 에서 RDS를 생성할때 설정한 비밀번호를 넣으면 된다.**
  - **SPRING_DATA_REDIS_HOST :** 2.2 에서 생성한 ElastiCache에서 기본 엔드포인트에 포트번호를 빼고 넣으면 된다. 아무생각 없이 복사 붙여넣기 했다고 포트번호까지 넣으면 안된다.



- **SPRING_DATA_REDIS_PASSWORD :** 2.2.2 에서 설정한 인증토큰을 넣으면 된다.

```
{
  "family": "test-backend",
  "containerDefinitions": [
    {
      "command": [
        "java",
        "-jar",
        "/app/app.jar"
      ],
      "cpu": 0,
      "environment": [
        {
          "name": "GOOGLE_CLIENT_SECRET",
          "value": "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"
        },
        {
          "name": "LIBER_HOST",
          "value": "libretranslate.howru"
        },
        {
          "name": "NLP_BASE-URL",
          "value": "http://spacy-api.howru:8000"
        },
        {
          "name": "SPRING_DATASOURCE_URL",
          "value": "jdbc:postgresql://howru-postgresql.ctaaga8gwill.ap-northeast-2.rds.amazonaws.com/howru"
        },
```

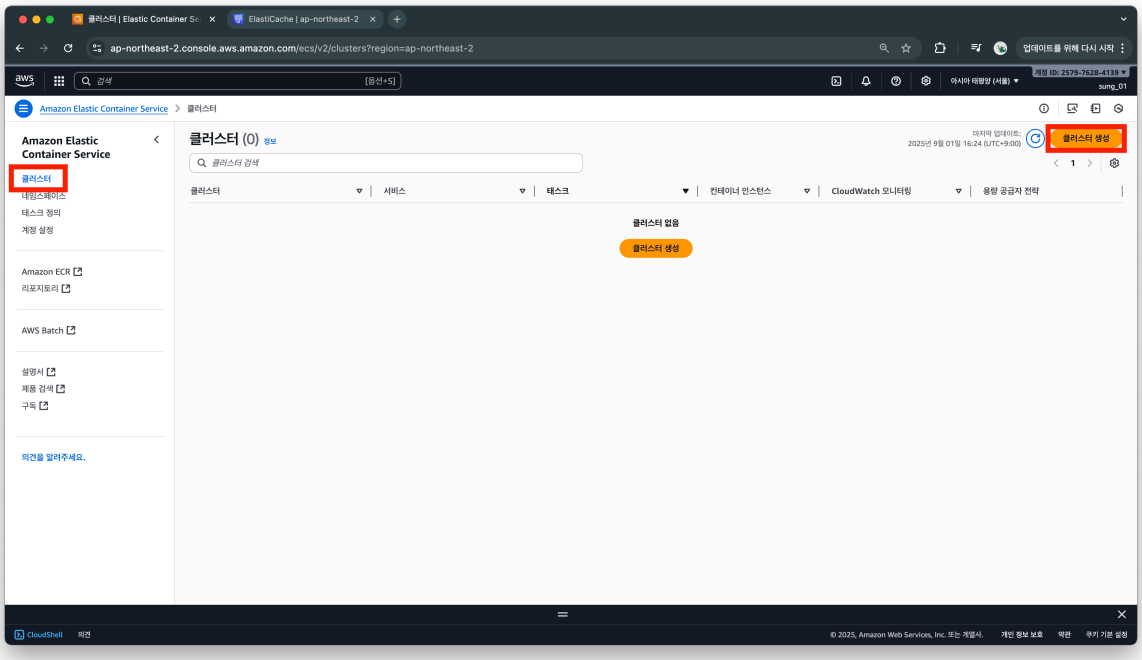
```
            {
              "name": "GOOGLE_CLIENT_ID",
              "value": "281239831867-k3oak565a93gd2ocmdp2jnf8vch62hb4.apps.googleuse
rcontent.com"
            },
            {
              "name": "GEMINI_API_KEY",
              "value": "AIzaSyA4dW7JlKu-tYm42fFjHvGwlELhB_XeEv0"
            },
            {
              "name": "SPRING_DATASOURCE_USERNAME",
              "value": "root"
            },
            {
              "name": "TAGGING-NLP_BASE-URL",
              "value": "http://fast-api.howru:8001"
            },
            {
              "name": "SPRING_DATA_REDIS_SSL_ENABLED",
              "value": "true"
            },
            {
              "name": "SPRING_DATA_REDIS_HOST",
              "value": "master.howru-elasticache.m8mhx5.apn2.cache.amazonaws.com"
            },
            {
              "name": "SPRING_DATA_REDIS_PORT",
              "value": "6379"
            },
            {
              "name": "FRONT_URL",
              "value": "https://howareu.click/"
            },
            {
              "name": "SPRING_DATA_REDIS_PASSWORD",
              "value": "ajtwoddltkwk1234"
            },
            {
              "name": "LIBER_PORT",
              "value": "5050"
            },
            {
              "name": "SPRING_DATASOURCE_PASSWORD",
              "value": "shop1011!"
            },
            {
              "name": "JWT_SECRET",
```

```
                "value": "\"9516e905b77cddafd0b688a98963eaf491549241f3411e2b0afb45e61f
6d54e30b9840fa77e26e71668b6c20fe2883ab75d4544e3bfdfaefc5b43ef3ccabefde\""
            },
            {
                "name": "SPRING_JPA_DATABASE_PLATFORM",
                "value": "org.hibernate.dialect.PostgreSQLDialect"
            }
        ],
        "essential": true,
        "healthCheck": {
            "command": [
                "CMD-SHELL",
                "curl -fsS http://localhost:8080/health || exit 1"
            ],
            "interval": 30,
            "retries": 3,
            "startPeriod": 60,
            "timeout": 5
        },
        "image": "257976284139.dkr.ecr.ap-northeast-2.amazonaws.com/howru-spring:dc9d
343b9ef233e9c7f7f9433f1e88bd05b2f6a3",
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/howru-backend",
                "awslogs-region": "ap-northeast-2",
                "awslogs-stream-prefix": "ecs"
            }
        },
        "mountPoints": [],
        "name": "spring-app",
        "portMappings": [
            {
                "containerPort": 8080,
                "hostPort": 8080,
                "protocol": "tcp"
            }
        ],
        "systemControls": [],
        "volumesFrom": []
    }
  ],
  "executionRoleArn": "arn:aws:iam::257976284139:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc",
  "volumes": [],
  "placementConstraints": [],
  "requiresCompatibilities": [
```

```
      "EC2"
    ],
    "cpu": "1024",
    "memory": "4096"
}
```

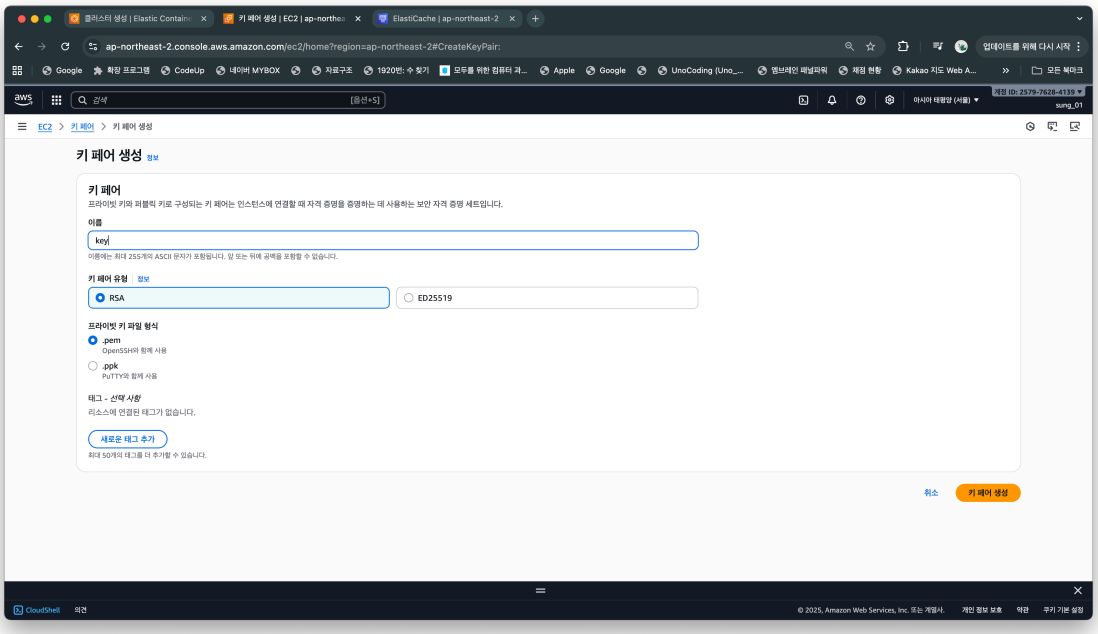
4.3 클러스터 생성

Amazon Elastic Container Service → 클러스터 → 클러스터 생성



- **클러스터 정보**
  - **클러스터 이름** : test-cluster (임의로 설정 가능)
- **인프라**
  - ❌ **AWS Fargate(서버리스)** ← 돈먹는 하마임
  - ✅ **Amazon EC2 인스턴스**
  - **Auto Scaling 그룹(ASG)** : 새 ASG 생성
  - **프로비저닝 모델 :** ✅ **온디멘드**
  - **컨테이너 인스턴스 Amazon Machine Image(AMI):** Amazon Linux 2 ← ECS에 최적화된 AMI임
  - **EC2 인스턴스 유형 : t3.large ←넉넉 잡아서 설정했다, free tier 아님을 주의하자**
  - **EC2 인스턴스 역할** : ecsInstanceRole
  - **원하는 용량**
    - **최소 : 0 / 최대 : 5**

- **SSH 키페어** : 새 키페어 생성 클릭 (키페어 저장해두자)

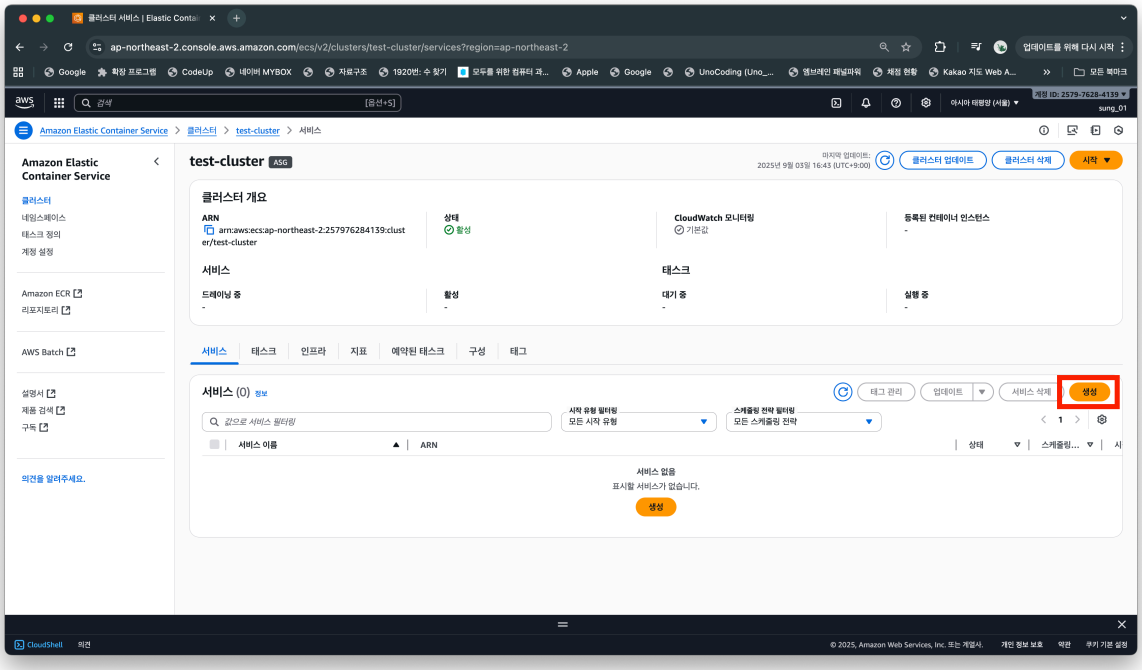


- **Amazon EC2 인스턴스 네트워크 설정**
  - **VPC :** 1.1 에서 생성한 VPC로 설정
  - **서브넷 :** VPC 생성하면서 생성된 subnet 모두 설정, 잘 따라왔으면 4개가 설정되는게 정상
  - **보안 그룹** : ✅ 기존 보안 그룹 선택
    - **보안 그룹 이름 :** 1.2.1 에서 생성한 Backend용 보안 그룹으로 설정
  - 퍼블릭 IP 자동 지정 : ✅켜기
- **모니터링** : 설정 X
- **암호화** : 설정 X
- **태그** : 설정 X

→ 클러스터 생성

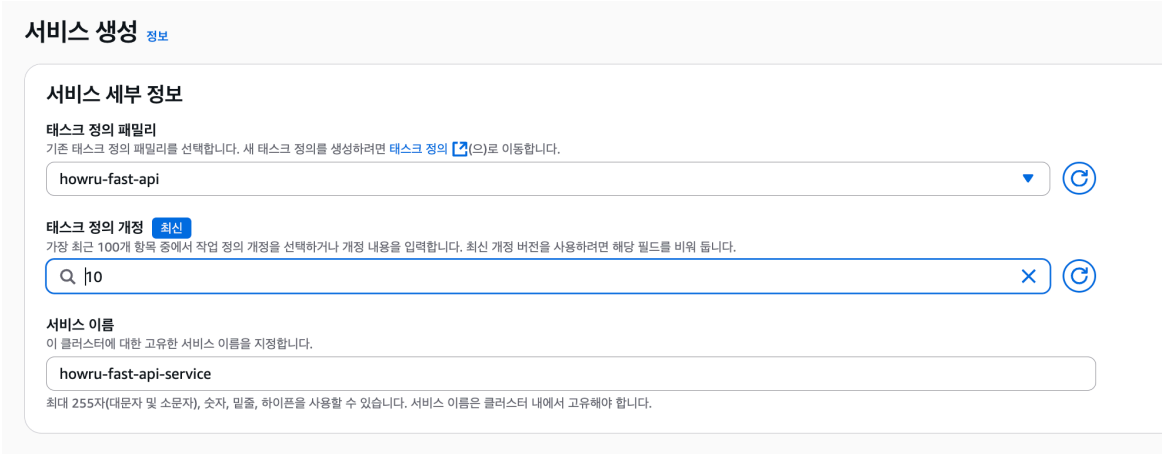
4.4 서비스 생성

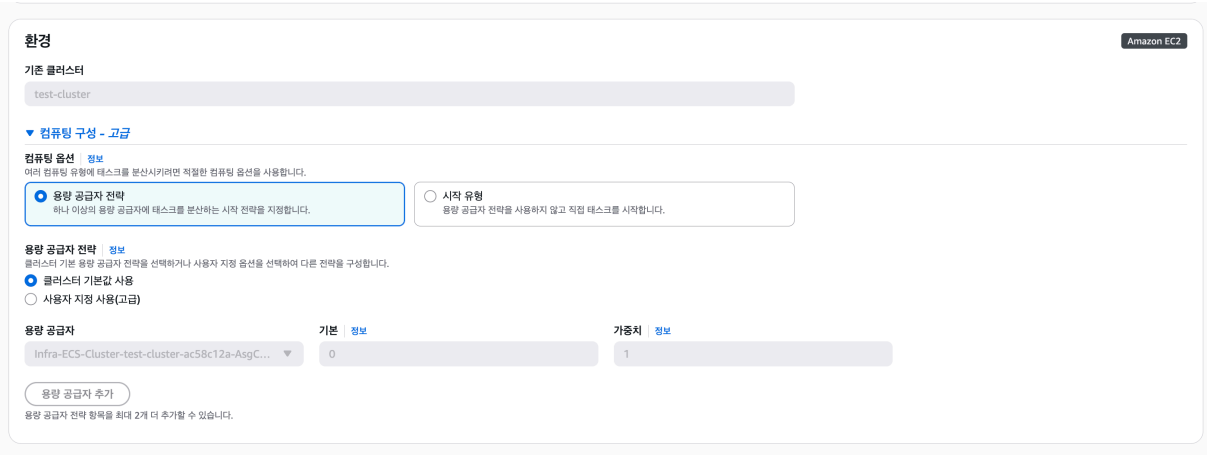
서비스 생성은 4.3 에서 생성한 클러스터에 들어가서 생성 을 누르면 된다.



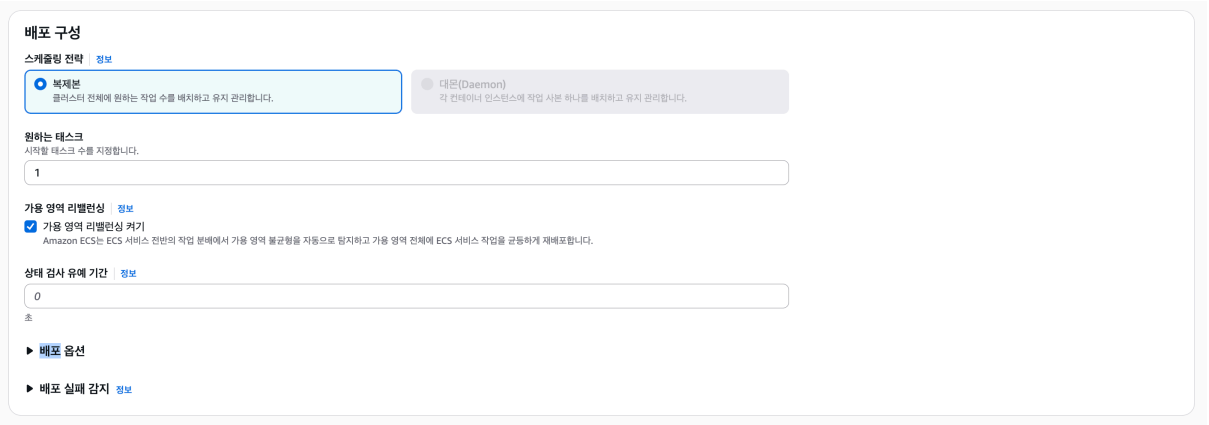
4.4.1 fast-api service 생성

- **테스크 정의 패밀리 :** 4.2.1 에서 생성한 Task를 선택
- **테스크 정의 계정 :** Task의 버전을 의미한다. 처음 생성했다면 1이 되는게 정상
- **서비스 이름** : 임의로 수정 가능하다.

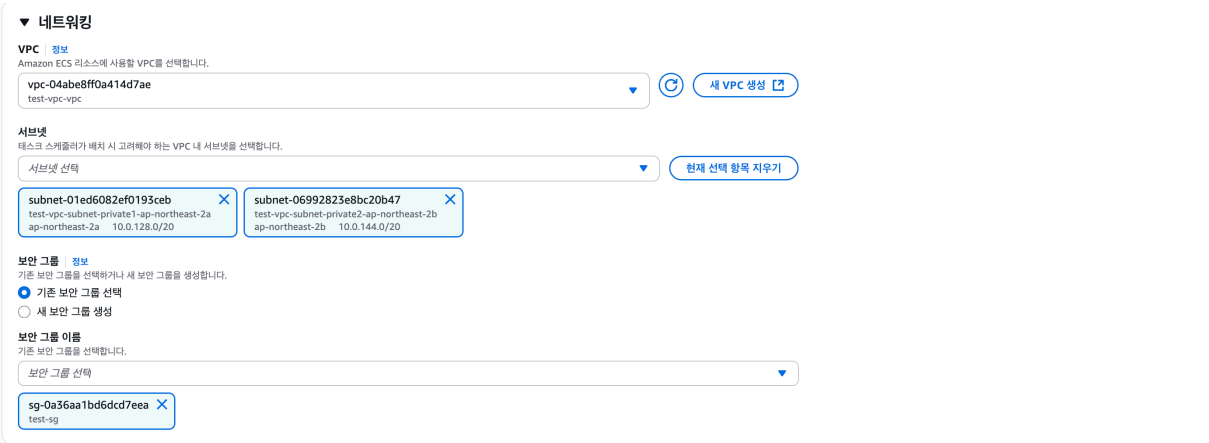




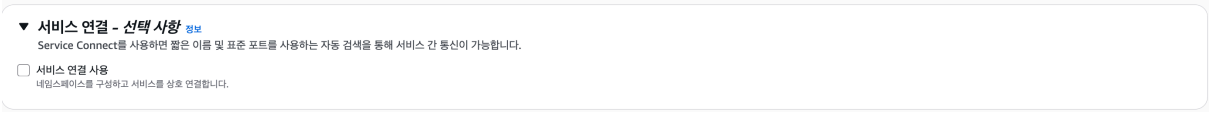
- 배포 구성은 따로 설정할 것이 없다.



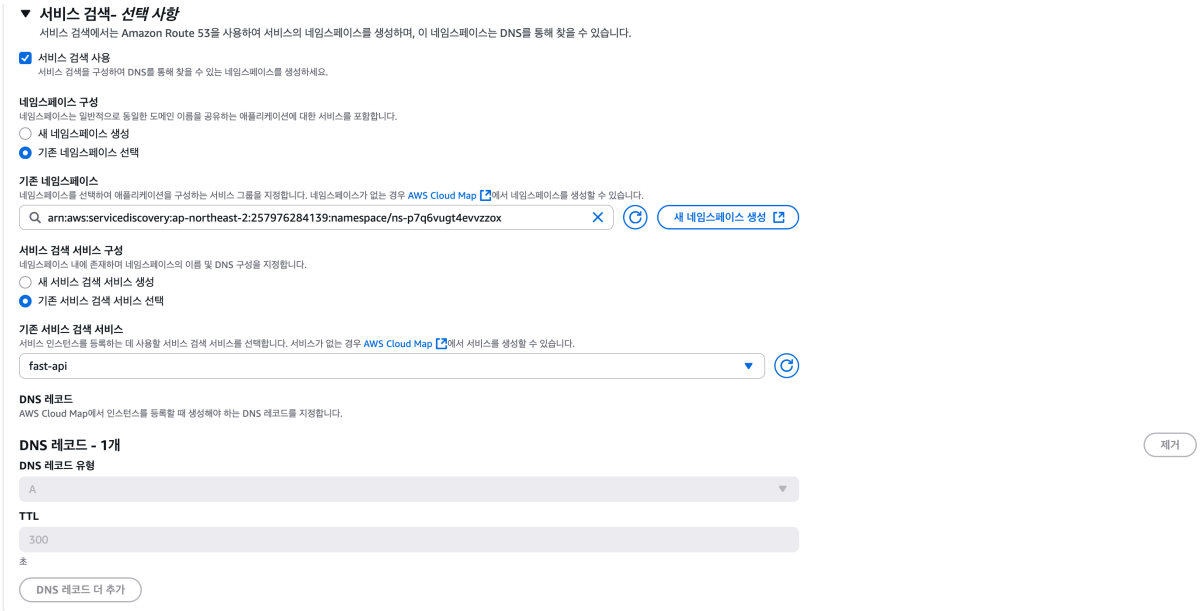
- **네트워킹** :
  - **VPC :** 1.1 에서 생성한 VPC로 설정
  - **서브넷** : 반드시 VPC 생성할 때 같이 생성한 **Private subnet** 2개로 설정
  - **보안 그룹** : ✅ 기존 보안 그룹 선택
  - **보안 그룹 이름** : 1.2.1 에서 생성한 Backend용 보안 그룹으로 설정



- **서비스 연결 :** ❌ 선택 안함



- **서비스 검색 :** ✅ 서비스 검색 사용
- **네임 스페이스 구성 :** ✅ 기존 네임스페이스 선택
- **기존 네임스페이스 :** 4.1.1 에서 생성한 네임스페이스 선택
- **서비스 검색 서비스 구성 :** ✅ 기존 서비스 검색 서비스 선택
- **기존 서비스 검색 서비스 :** 4.1.2 에서 생성한 fast-api 서비스 선택

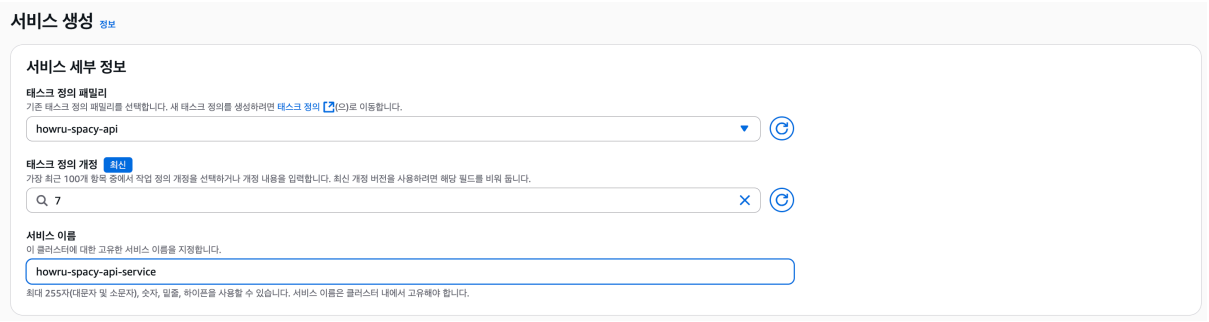


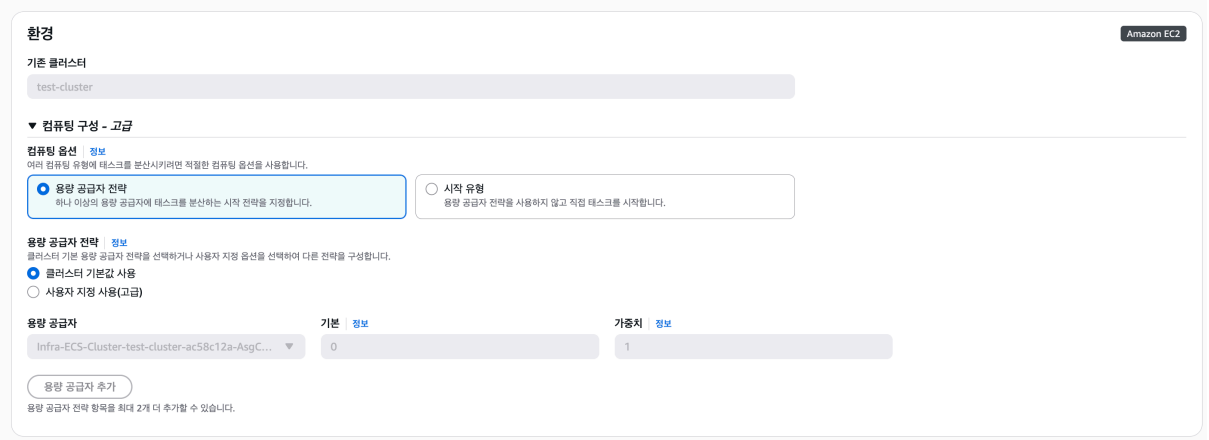
- **로드 밸런싱, VPC Lattice, 서비스 자동 크지 조정, 테스크 배치, 볼륨, 태그는 수정 없음**

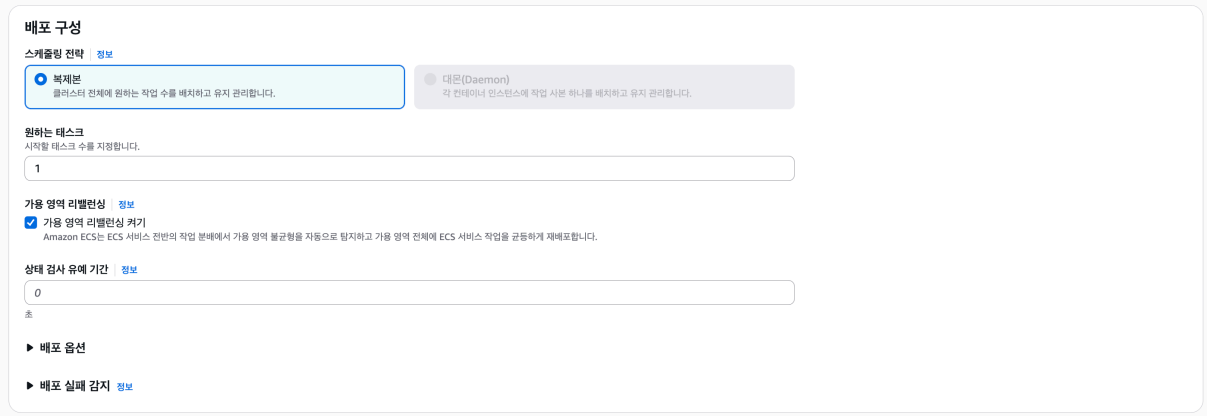
→생성 클릭

4.4.2 spacy-api service 생성

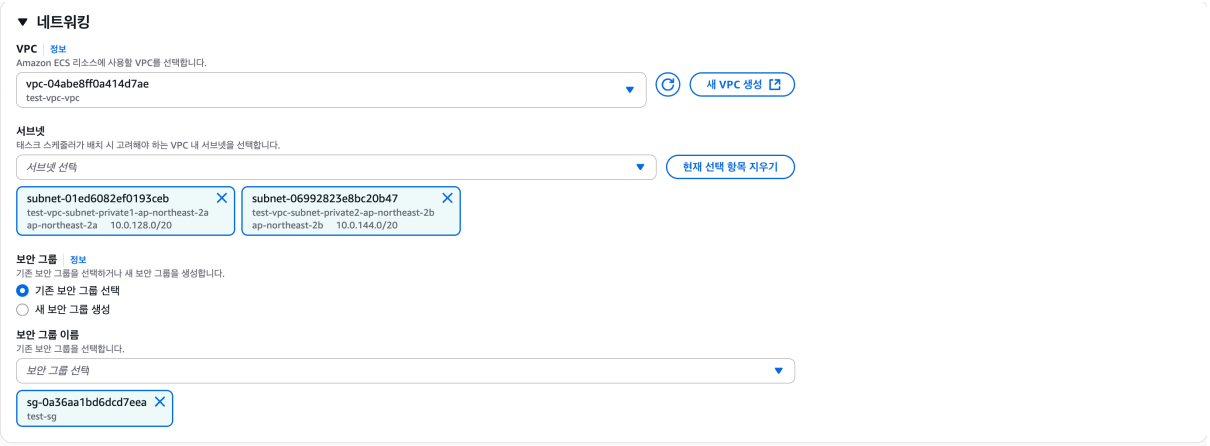
- **테스크 정의 패밀리 :** 4.2.2 에서 생성한 Task를 선택
- **테스크 정의 계정 :** Task의 버전을 의미한다. 처음 생성했다면 1이 되는게 정상
- **서비스 이름** : 임의로 수정 가능하다.



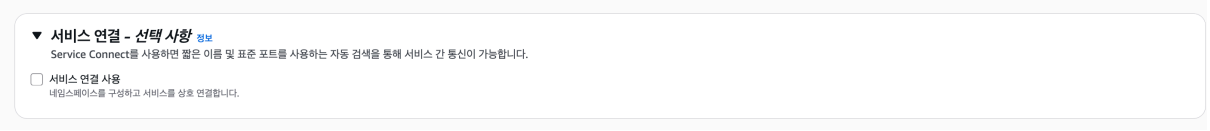




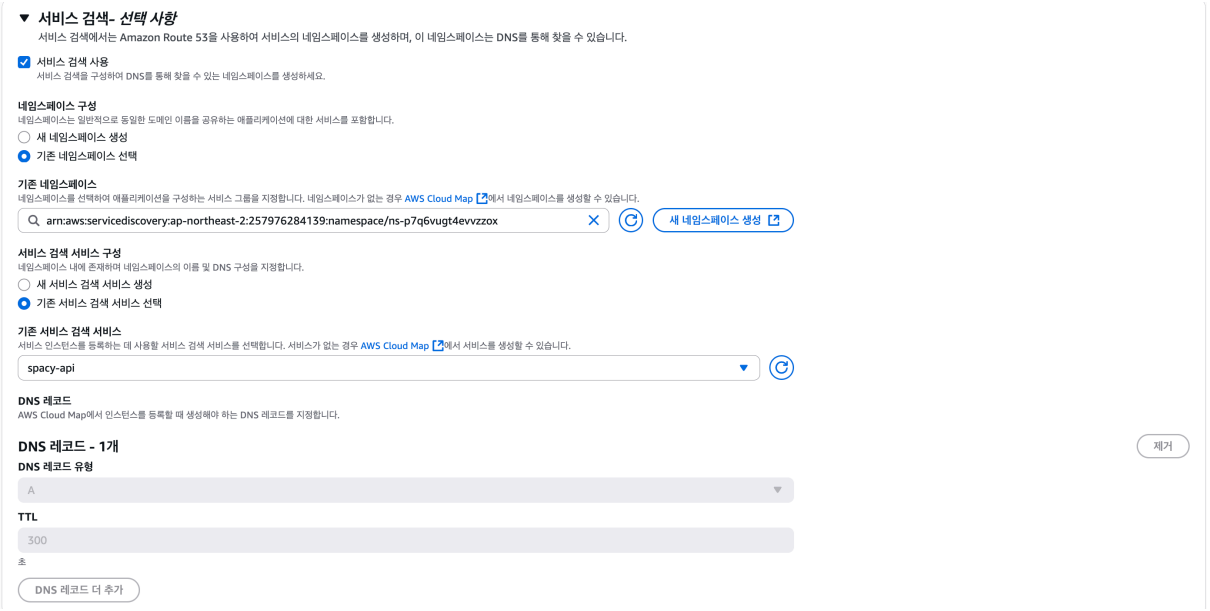
- **네트워킹** :
  - **VPC :** 1.1 에서 생성한 VPC로 설정
  - **서브넷** : 반드시 VPC 생성할 때 같이 생성한 **Private subnet** 2개로 설정
  - **보안 그룹** : ✅ 기존 보안 그룹 선택
  - **보안 그룹 이름** : 1.2.1 에서 생성한 Backend용 보안 그룹으로 설정



- **서비스 연결 :** ❌ 선택 안함



- **서비스 검색 :** ✅ 서비스 검색 사용
- **네임 스페이스 구성 :** ✅ 기존 네임스페이스 선택
- **기존 네임스페이스 :** 4.1.1 에서 생성한 네임스페이스 선택
- **서비스 검색 서비스 구성 :** ✅ 기존 서비스 검색 서비스 선택
- **기존 서비스 검색 서비스 :** 4.1.2 에서 생성한 spacy-api 서비스 선택

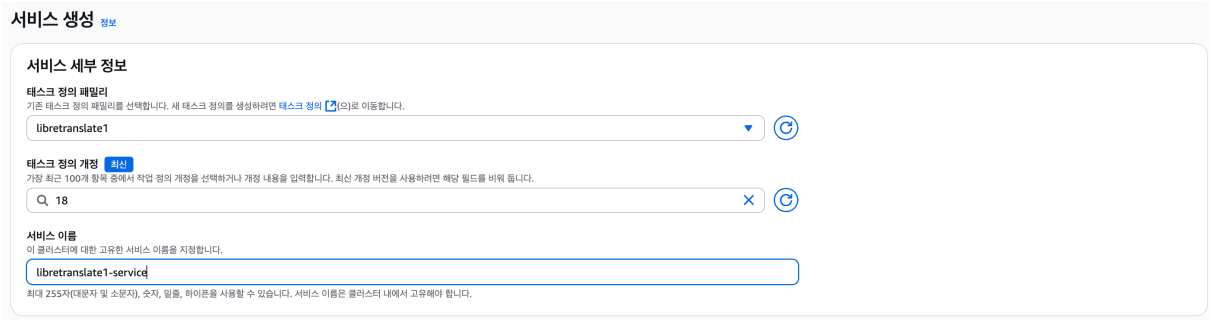


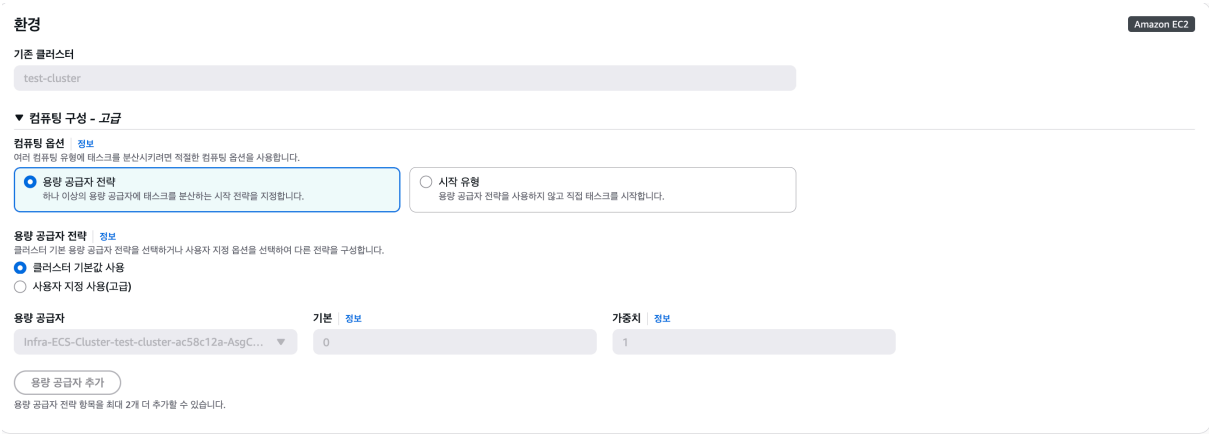
- **로드 밸런싱, VPC Lattice, 서비스 자동 크지 조정, 테스크 배치, 볼륨, 태그는 수정 없음**

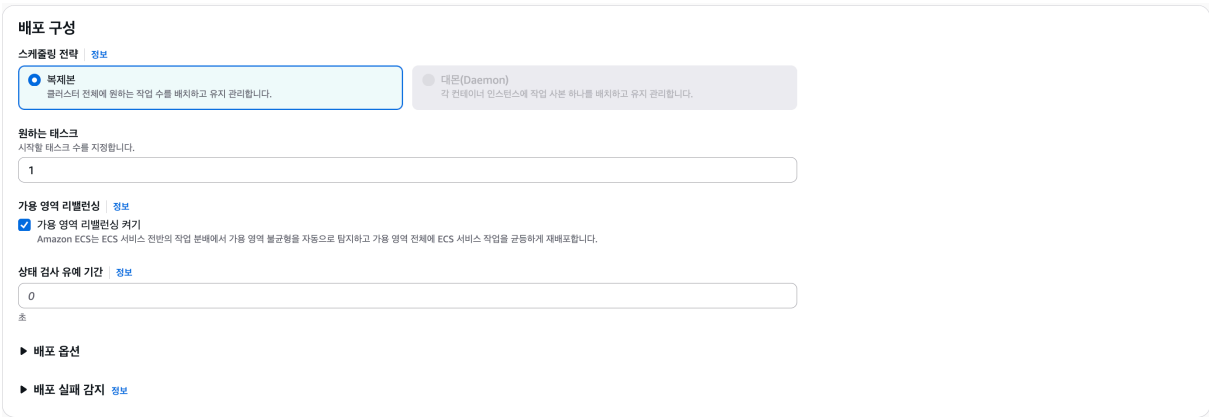
→생성 클릭

4.4.3 LibreTranslate service 생성

- **테스크 정의 패밀리 :** 4.2.3 에서 생성한 Task를 선택
- **테스크 정의 계정 :** Task의 버전을 의미한다. 처음 생성했다면 1이 되는게 정상
- **서비스 이름** : 임의로 수정 가능하다.

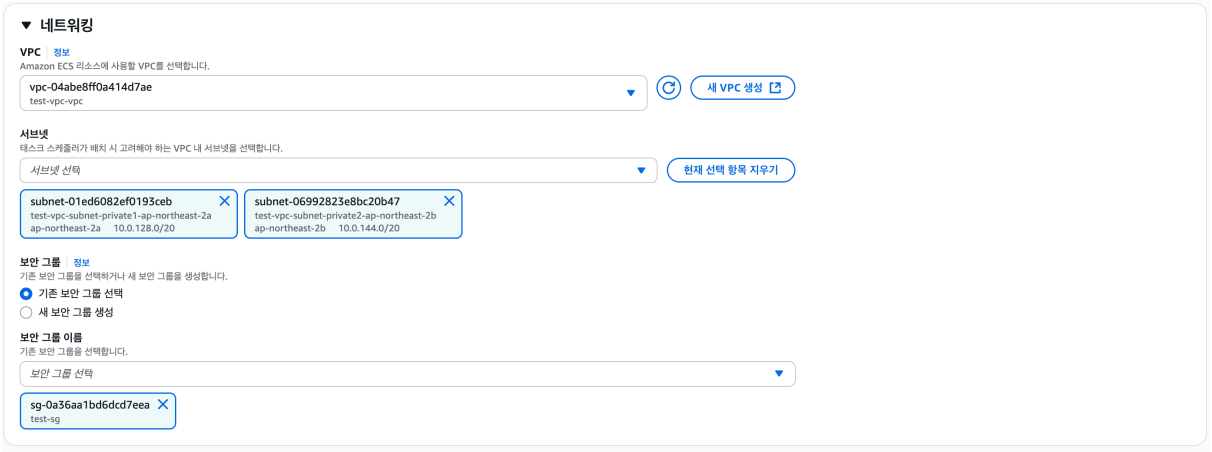




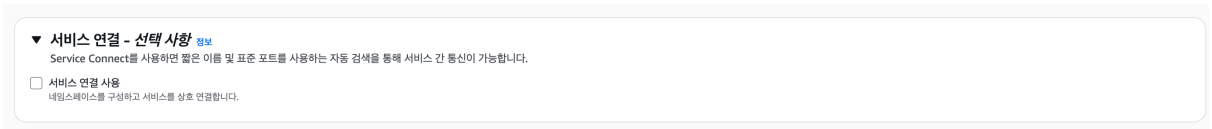


- **네트워킹** :
  - **VPC :** 1.1 에서 생성한 VPC로 설정
  - **서브넷** : 반드시 VPC 생성할 때 같이 생성한 **Private subnet** 2개로 설정
  - **보안 그룹** : ✅ 기존 보안 그룹 선택

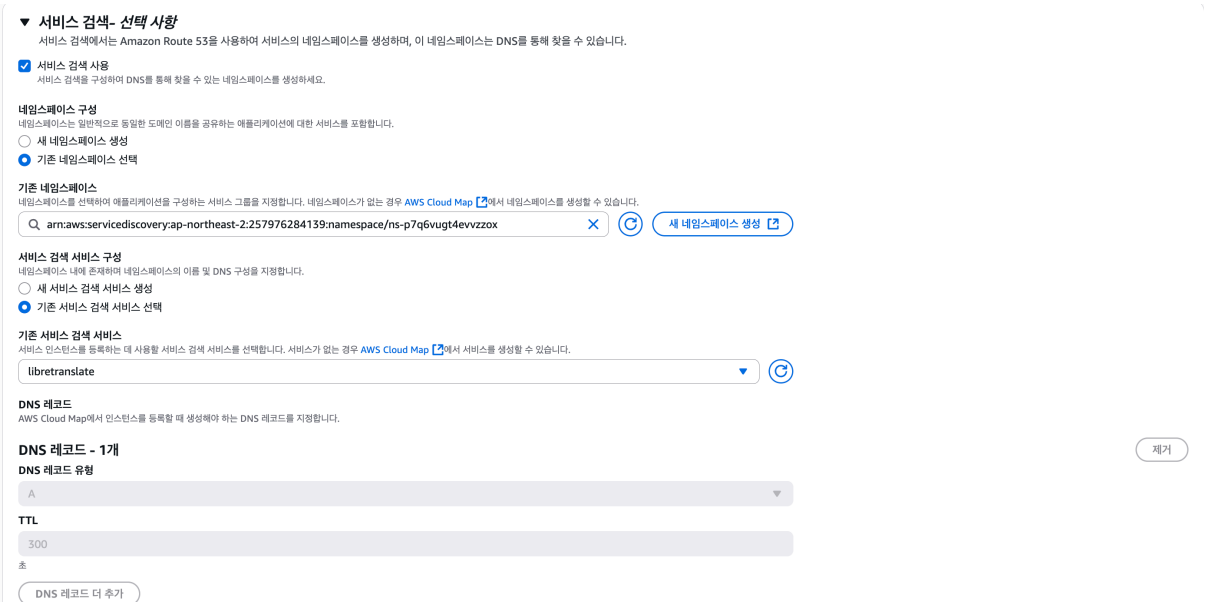
- **보안 그룹 이름** : 1.2.1 에서 생성한 Backend용 보안 그룹으로 설정



- **서비스 연결** : ❌ 선택 안함



- **서비스 검색** : ✅ 서비스 검색 사용
- **네임 스페이스 구성** : ✅ 기존 네임스페이스 선택
- **기존 네임스페이스** : 4.1.1 에서 생성한 네임스페이스 선택
- **서비스 검색 서비스 구성** : ✅ 기존 서비스 검색 서비스 선택
- **기존 서비스 검색 서비스** : 4.1.2 에서 생성한 libretranslate 서비스 선택

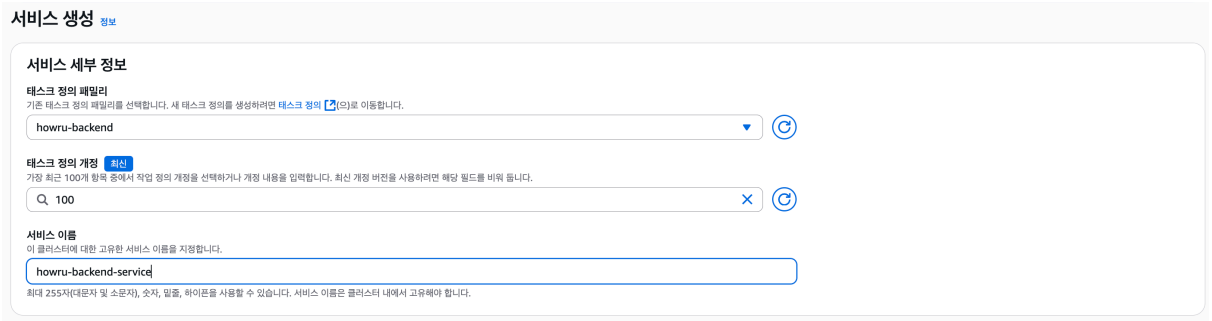


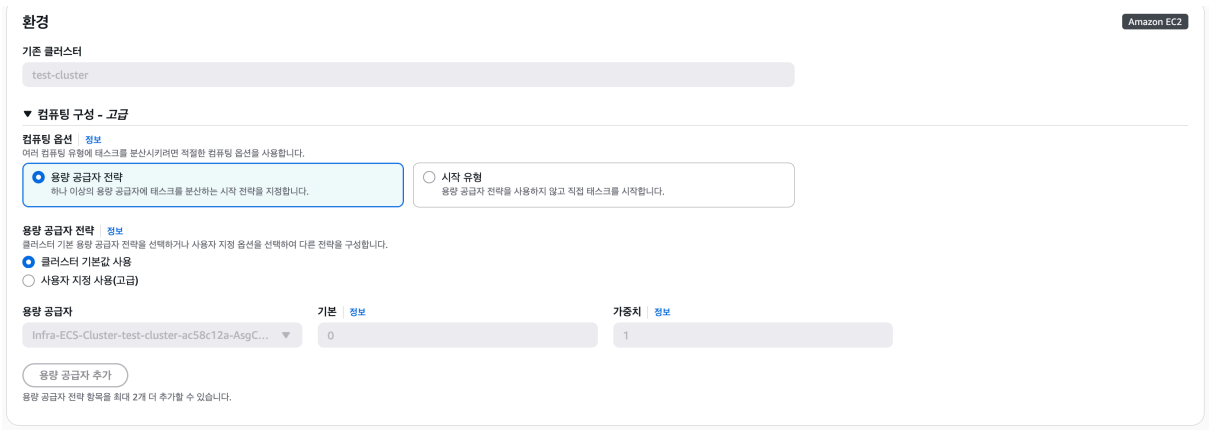
- **로드 밸런싱, VPC Lattice, 서비스 자동 크지 조정, 테스크 배치, 볼륨, 태그는 수정 없음**

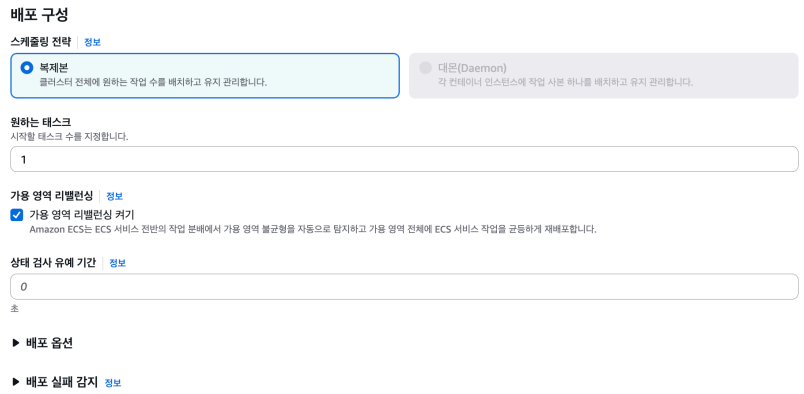
→생성 클릭

4.4.4 SpringBoot service 생성

- **테스크 정의 패밀리 :** 4.2.4 에서 생성한 Task를 선택
- **테스크 정의 계정 :** Task의 버전을 의미한다. 처음 생성했다면 1이 되는게 정상
- **서비스 이름** : 임의로 수정 가능하다.

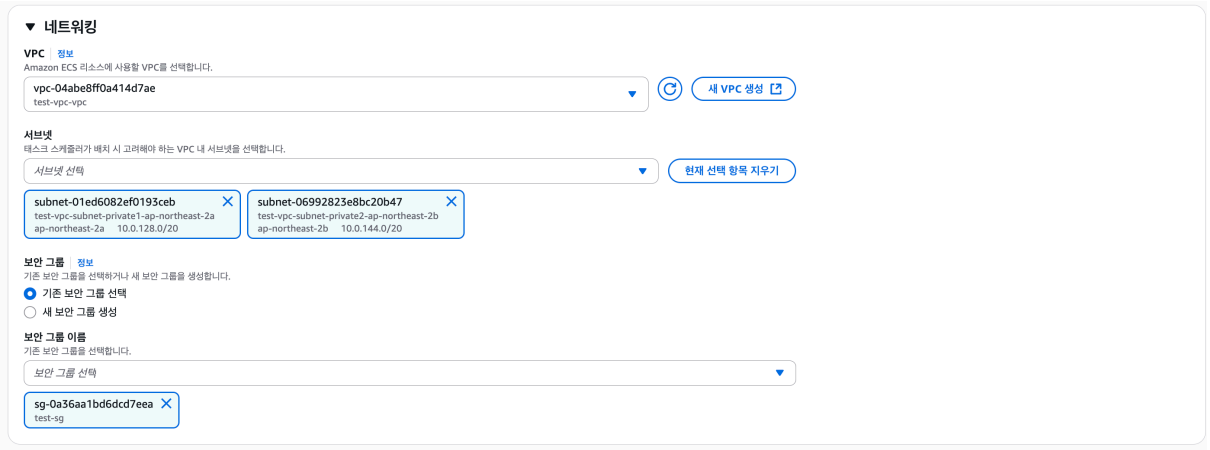




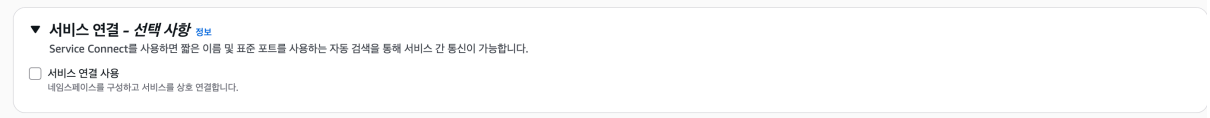


- **네트워킹** :
  - **VPC :** 1.1 에서 생성한 VPC로 설정
  - **서브넷** : 반드시 VPC 생성할 때 같이 생성한 **Private subnet** 2개로 설정

- **보안 그룹 :** ✅ 기존 보안 그룹 선택
- **보안 그룹 이름 :** 1.2.1 에서 생성한 Backend용 보안 그룹으로 설정

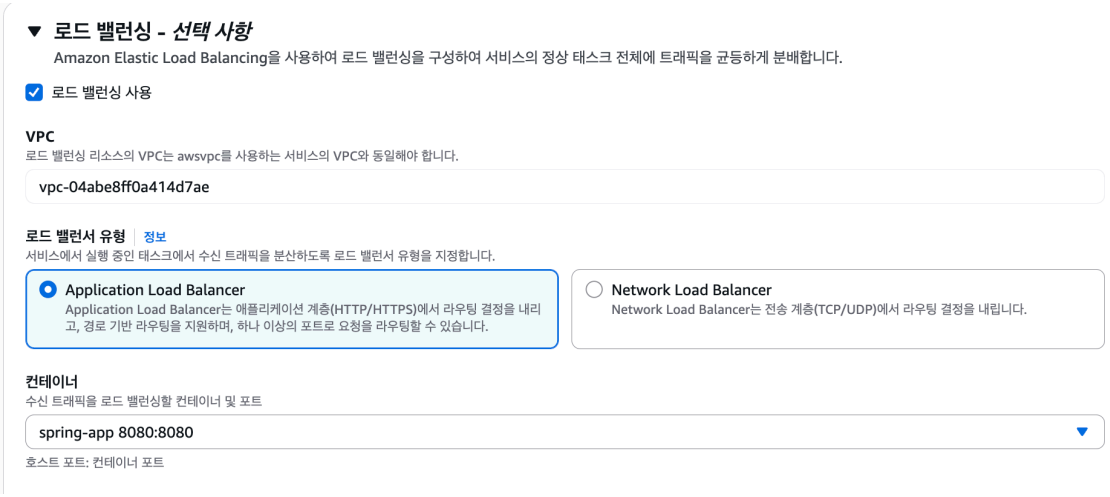


- **서비스 연결 :** ❌ 선택 안함



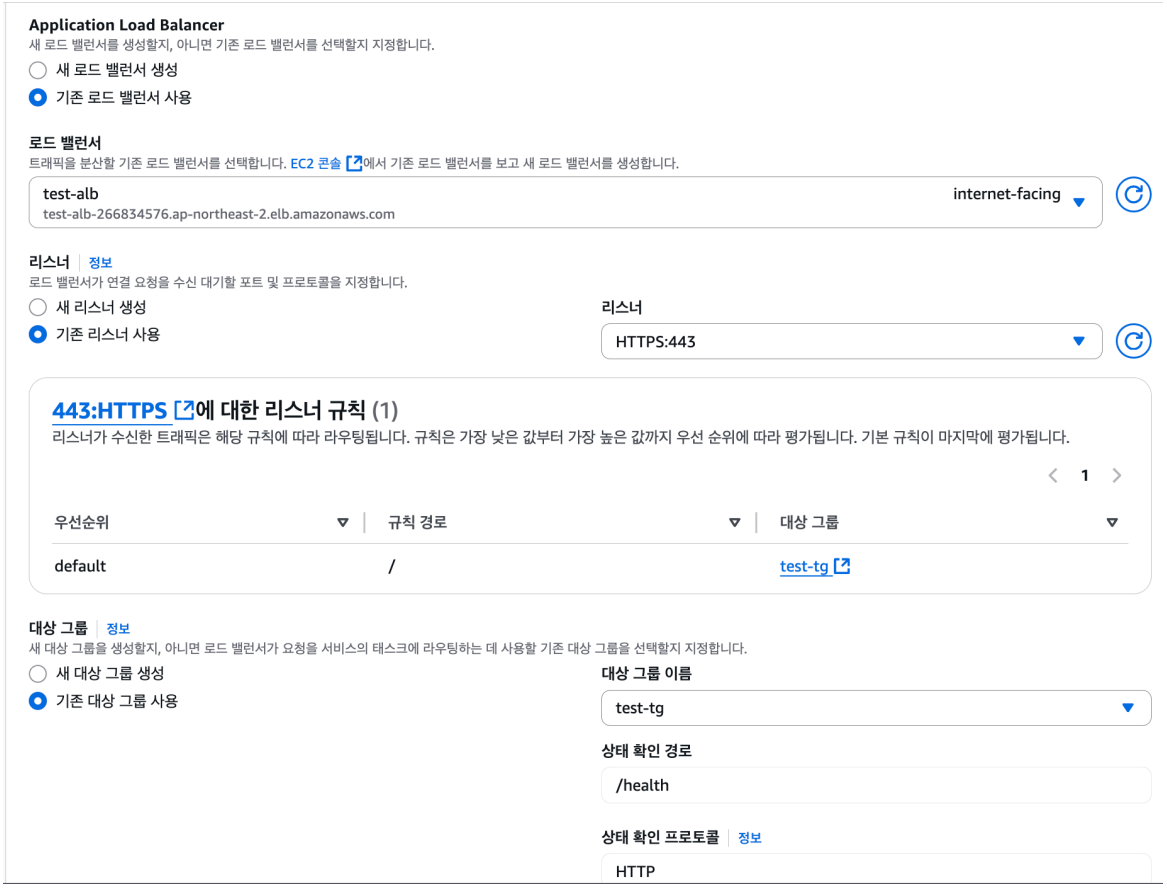
- **서비스 검색 :** ❌ 선택 안함

- **로드 밸런싱 :** ✅사용
- **로드 밸런서 유형** : Application Load Balancer



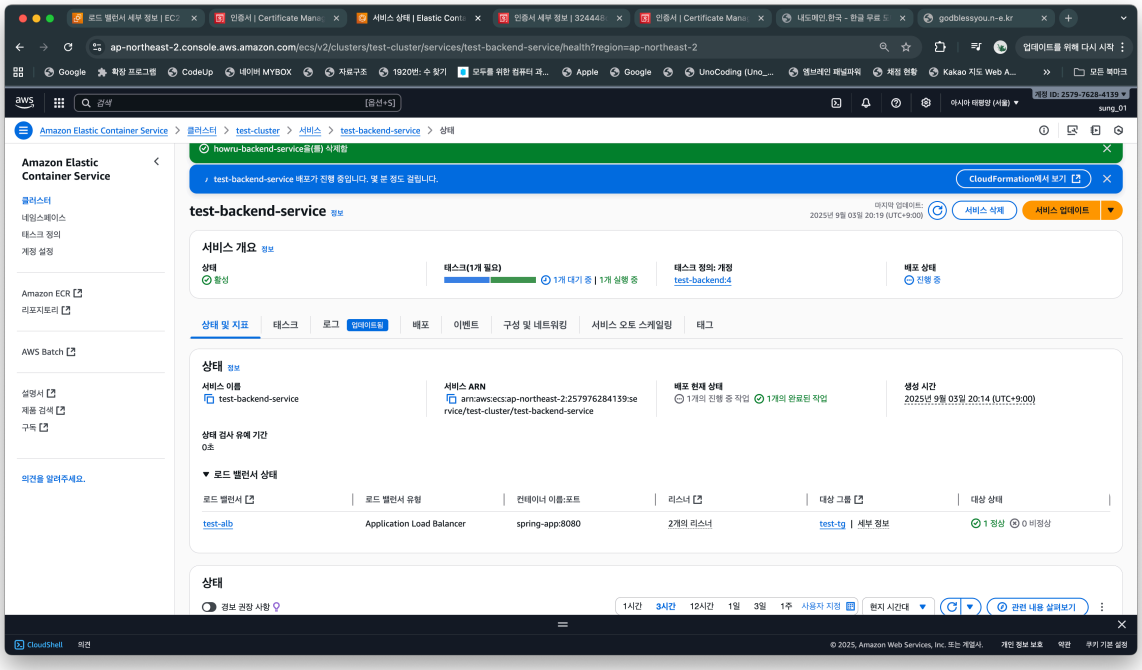
- **Application Load Balancer :** ✅ **기존 로드 밸런서 사용**
- **로드 밸런서 :** 1.3 에서 생성한 로드 밸런서 설정

- ✅ **기존 리스너 사용**
- **리스너** : HTTPS:443
- **대상 그룹** : ✅ 기존 대상 그룹 사용
  - **대상 그룹 이름 :** `1.3` 에서 로드 밸런서 생성할때 만든 대상 그룹 설정



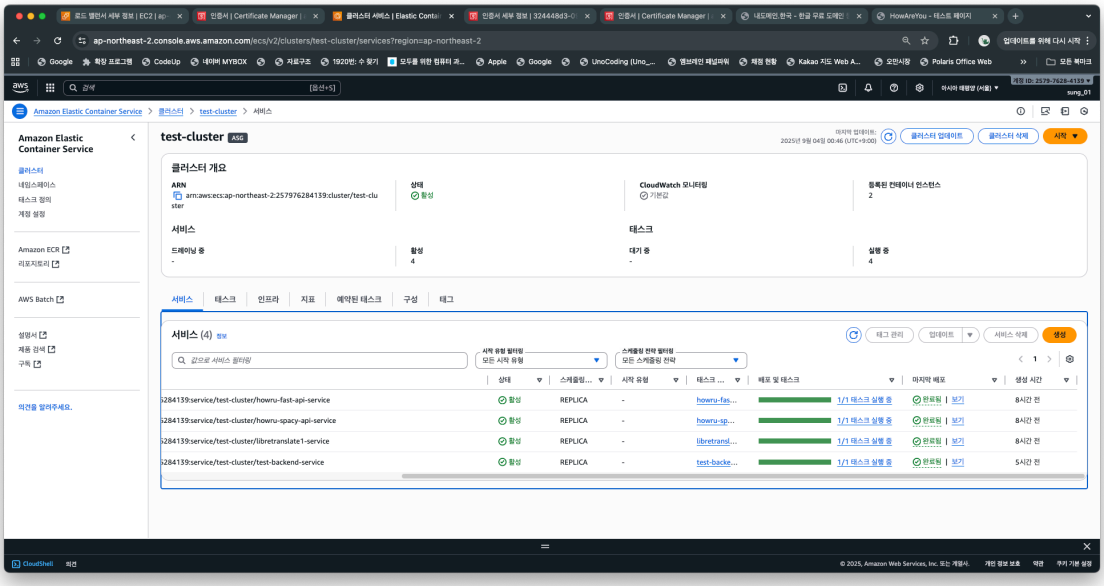
그리고 나머지는 설정할 것 없으니 밑으로 스크롤해서 `생성` 버튼 클릭

그후 기다리다가 `springboot service`의 로드밸러서의 상태를 확인하면 된다.

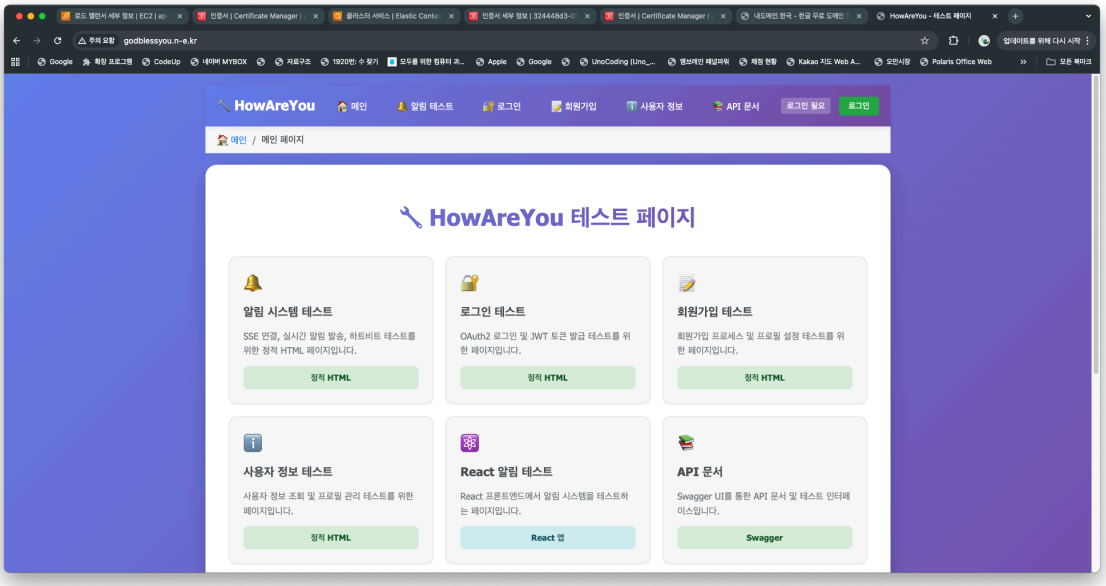


5. 마무리

생성한 클러스터에 모든 서비스들이 배포 완료가 뜨면 성공이다. 모든 서비스들이 그렇지만, 특히 `springboot` 서비스를 배포할때 시간이 오래 걸린다.



`내도메인.한국` 에 등록한 도메인으로 접속하면 잘 작동하는 것을 확인할 수 있다.


godblessyou.n-e.kr
HowAreYou
메인
알림 테스트
로그인
회원가입
사용자 정보
API 문서
로그인 필요
로그인
메인 / 메인 페이지
HowAreYou 테스트 페이지
알림 시스템 테스트
SSE 연결, 실시간 알림 발송, 하트비트 테스트를 위한 정적 HTML 페이지입니다.
정적 HTML
로그인 테스트
OAuth2 로그인 및 JWT 토큰 발급 테스트를 위한 페이지입니다.
정적 HTML
회원가입 테스트
회원가입 프로세스 및 프로필 설정 테스트를 위한 페이지입니다.
정적 HTML
사용자 정보 테스트
사용자 정보 조회 및 프로필 관리 테스트를 위한 페이지입니다.
정적 HTML
React 알림 테스트
React 프론트엔드에서 알림 시스템을 테스트하는 페이지입니다.
React 앱
API 문서
Swagger UI를 통한 API 문서 및 테스트 인터페이스입니다.
Swagger