언택트 비즈니스를 위한 멀티클라우드 아키텍트 양성과정 최종 프로젝트

**클라우드 보안을 고려한 라이브 스트리밍 이커머스 서비스 개발**

2조 : 박종현, 임상혁, 심재혁

**목 차**

[**1. 비즈니스모델** 3](#_Toc75117770)

[**2. 요구사항** 5](#_Toc75117771)

[**2.1 비즈니스 요구사항** 5](#_Toc75117772)

[**2.2 기능적 요구 사항** 5](#_Toc75117773)

[**2.3 아키텍처 요구사항** 6](#_Toc75117774)

[**2.4 ERD** 7](#_Toc75117775)

[**3. 서비스 구현** 8](#_Toc75117776)

[**3.1 개발환경** 8](#_Toc75117777)

[**3.2 서비스 아키텍처** 9](#_Toc75117778)

[**1)** **아키텍처 구성도** 9](#_Toc75117779)

[**2)** **서비스 요약** 10](#_Toc75117780)

[**3)** **라이브 스트리밍 서비스** 10](#_Toc75117781)

[**4)** **CDN아키텍처 - 온디멘드(OnDemand) 콘텐츠 배포** 16](#_Toc75117782)

[**5)** **고가용성 고객 데이터베이스** 29](#_Toc75117783)

[**6)** **보안을 고려한 아키텍처** 32](#_Toc75117784)

[**7)** **고가용성 서비스 배포 Strategy** 38](#_Toc75117785)

[**4. Demo Page** 48](#_Toc75117786)

[**5. 보완 할 점** 56](#_Toc75117787)

[1) 아키텍처 측면 56](#_Toc75117788)

[2) 웹 어플리케이션 측면 56](#_Toc75117789)

[**6. 향후 프로젝트 확대 방향** 58](#_Toc75117790)

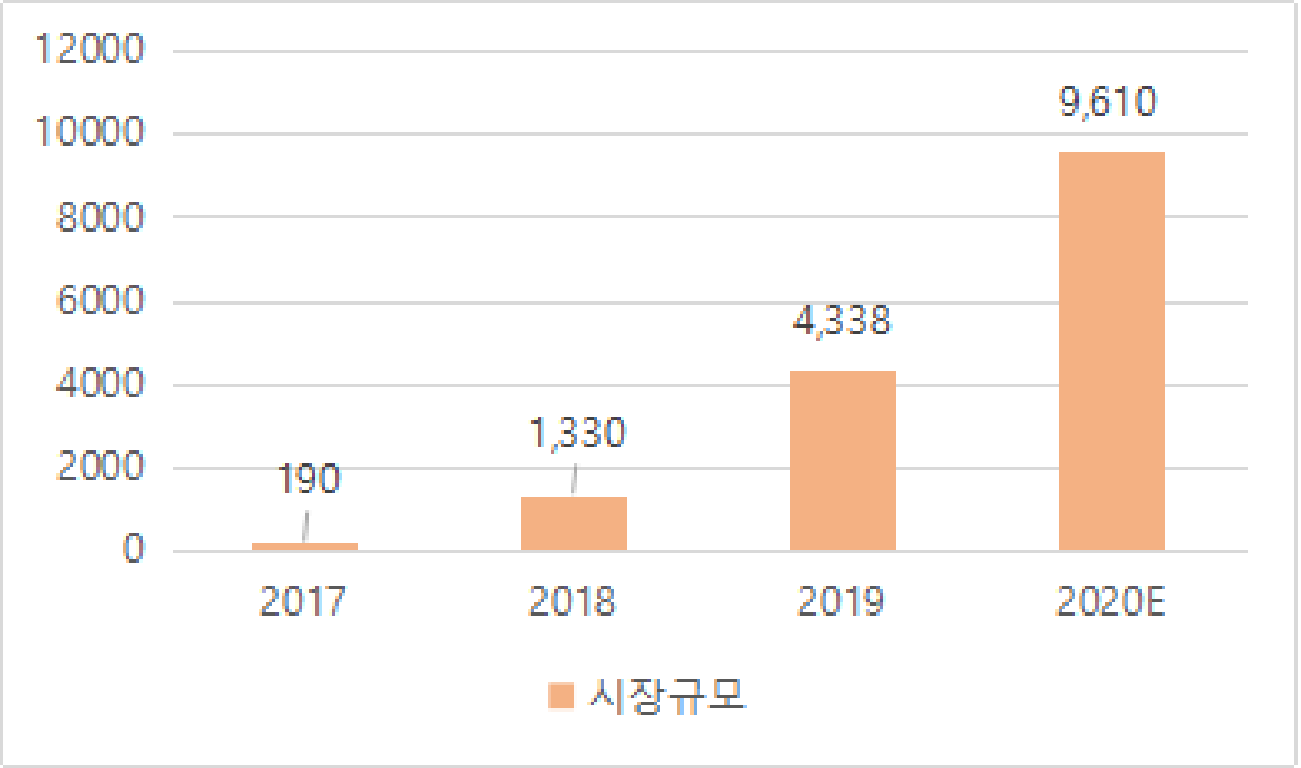
[1) 아키텍처 측면 58](#_Toc75117791)

[2) 서비스 측면 58](#_Toc75117792)

[**7. 마무리** 59](#_Toc75117793)

# **1. 비즈니스모델**

코로나19로 기존 오프라인 업종이 크게 타격을 입은 반면, 온라인 판매는 큰 성장세를 보였다. 이와 더불어 라이브커머스 마케팅 방식은 온라인 소비 촉진 측면에서 효과적인 마케팅 통로로 자리 잡았다. 중국 인터넷정보센터(CNNIC)에 따르면, 2020년 12월 기준 중국의 온라인 라이브방송 이용고객은 약 6억 1700명으로 2020년 3월보다 5703만 명 늘어 전체 네티즌의 62.4%를 차지했다. 이 중 판매와 연계한 ‘라이브커머스’ 생방송 이용자는 3억8000만 명으로 2020년 3월에 비해 1억9100만 명이 증가했다.



<그림1-1 중국내 라이브 커머스 시장규모>

라이브 커머스는 채팅으로 소비자와 소통하면서 상품을 소개하는 형태의 스트리밍 방송이다. 현재 우리나라에서는 네이버와 카카오의 쇼핑라이브, CJ의 올라이브 등을 중심으로 다양한 플랫폼에서 시장을 확대해 나가고 있다.

그렇다면 라이브 커머스의 어떠한 장점이 다양한 기업으로부터 비즈니스 확장 아이템으로 주목받을 수 있었을까?

* 높은 구매 전환율
* 기존 온라인 쇼핑몰의 구매 전환율은 0.37%인 반면, 라이브 커머스의 구매 전환율은 호스트에 따라 6~18%까지 이른다.
* 성장 가능성이 높다
* 라이브 커머스를 이용하는 사람의 대다수가 20-30대의 시장에서 성장 가능성이 높은 구매층이다.
* 데이터 기반
* 소비자의 데이터를 기반으로 매출을 증가시킬 수 있다.
* 누구나 셀러가 될 수 있다.
* 매장 인테리어 비용을 줄일 수 있다.
* 라이브 커머스 시장이 시작하는 단계이기 때문에 ‘블루오션’이다

이러한 장점으로 현재 빠르게 성장하고 있는 라이브커머스 비즈니스 경쟁에서 성공적으로 시장을 점유하기 위해서는 기본적으로 어떠한 형태로 서비스를 제공해야할 지 고민해보게 되었고 몇 가지 요구사항을 바탕으로 프로젝트를 진행하게 되었다.

# **2. 요구사항**

## **2.1 비즈니스 요구사항**

1. 고객 데이터 베이스 보안 요구사항 준수 누구나 거래를 하기 위해 스트리밍 할 수 있는 플랫폼
2. 개인/기업 거래방식 차이를 고려한 서비스 제공

* 회원에 따라 다양한 형태의 거래가 가능하도록 UI/UX(중고거래, 일반판매, 직거래)

1. 채팅 이용을 가능하게 함으로써 시청자가 요청하는 정보를 실시간으로 피드백 함으로써 제품에 대한 신뢰성 확보
2. 방송 다시 보기 기능으로 라이브 방송 이후에도 구매 활성화
3. 광고와 상품 판매로 발생하는 수수료를 기반으로 수익창출
4. 최저 자금으로 서비스 런칭 가능 및 서비스 유지 비용 최적화
5. 핵심 서비스인 스트리밍 서버의 안정성 확보

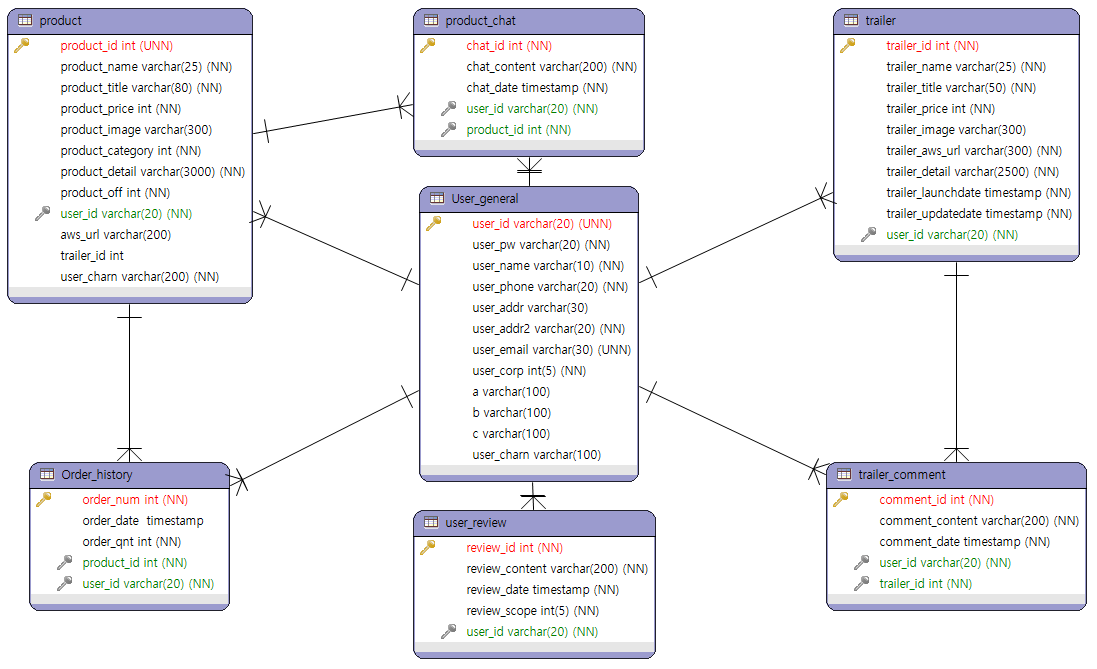
## **2.2 기능적 요구 사항**

1. 비회원
   1. 비회원은 회원가입 후 서비스를 이용할 수 있다.
   2. 회원가입 시 라이브 방송 채널을 생성해 준다.
2. 회원
   1. 회원 정보를 수정할 수 있다.
   2. 회원 탈퇴를 할 수 있다.
   3. 회원 아이디와 비밀번호를 등록된 이메일을 통해 찾을 수 있다.
   4. 주문기록, 판매기록, 방송기록, 방송예고 확인이 가능하다.
   5. 구매한 상품에 대해서 리뷰 작성이 가능하다
3. 라이브 예고 작성
   1. 라이브 예정을 등록할 수 있다.
   2. 라이브 예정용 동영상을 업로드 및 불러오기가 가능하다.
   3. 등록한 예고 정보를 기반으로 방송 시작이 가능하다.
   4. 댓글 작성이 가능하다.
4. 라이브 방송
   1. 소개 및 판매할 상품을 등록할 수 있다.
   2. 라이브 방송이 가능하다
   3. 실시간 채팅이 가능하다.
   4. 상품 주문이 가능하다.
   5. 라이브 방송이 종료되면 별도의 페이지에서 상품에 대한 리뷰를 확인할 수 있다.

## **2.3 아키텍처 요구사항**

1. 서비스 중인 하나의 Available Zone에서 장애 발생시에도 연속적인 서비스 제공이 가능하다
2. 요청이 급격하게 증가하더라도 서비스에 장애가 일어나지 않도록 한다
3. 웹사이트 공격으로부터 보호된다.
4. 클라이언트의 지리적 위치에 관계없이 균일한 품질의 서비스를 제공한다.
5. 하드웨어 프로비저닝, 데이터베이스 설정, 패치 및 백업과 같은 시간 소모적인 관리 작업을 자동화하면서 비용 효율적이고 크기 조정가능한 데이터베이스 서비스를 제공
6. 완전관리형 실시간 스트리밍 서비스를 사용함으로써 자동으로 확장가능하고 안정적인 웹서비스 제공이 가능하다.

## **2.4 ERD**



<그림2-1 ERMaster를 이용한 ERD 작성>

1. User\_general : 회원 테이블

* 일반 서비스 이용자 개인정보 및 라이브 스트리밍 서비스 이용을 위한 데이터

1. Product : 스트리밍 페이지 테이블 (상품 테이블)

* 상품 등록시 라이브 스트리밍 페이지 생성, 상품에 대한 정보 및 product\_off 속성을 통해 생방송 진행 여부를 저장,

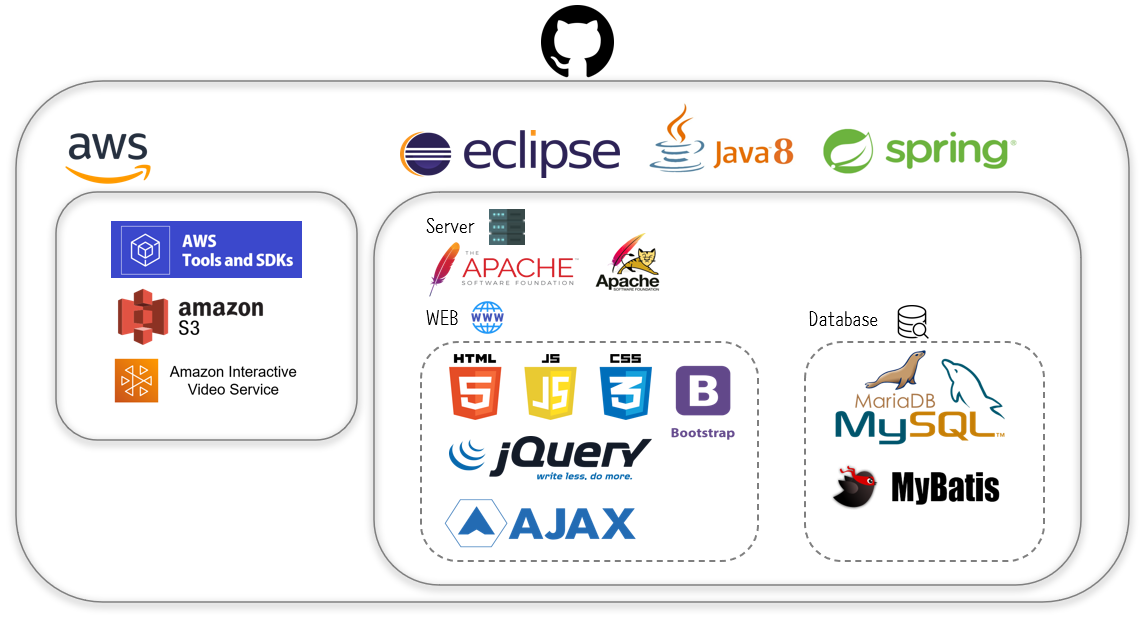
1. User\_review : 리뷰 테이블

* 상품을 구매한 유저에 한해서 리뷰 작성 가능

1. Product\_chat : 스트리밍 페이지 채팅 기능 테이블
2. Order\_history : 주문기록, 판매기록 테이블
3. Trailer : 라이브 예정 테이블
4. Trailer\_comment : 라이브 예정 페이지 댓글 테이블

# **3. 서비스 구현**

## **3.1 개발환경**

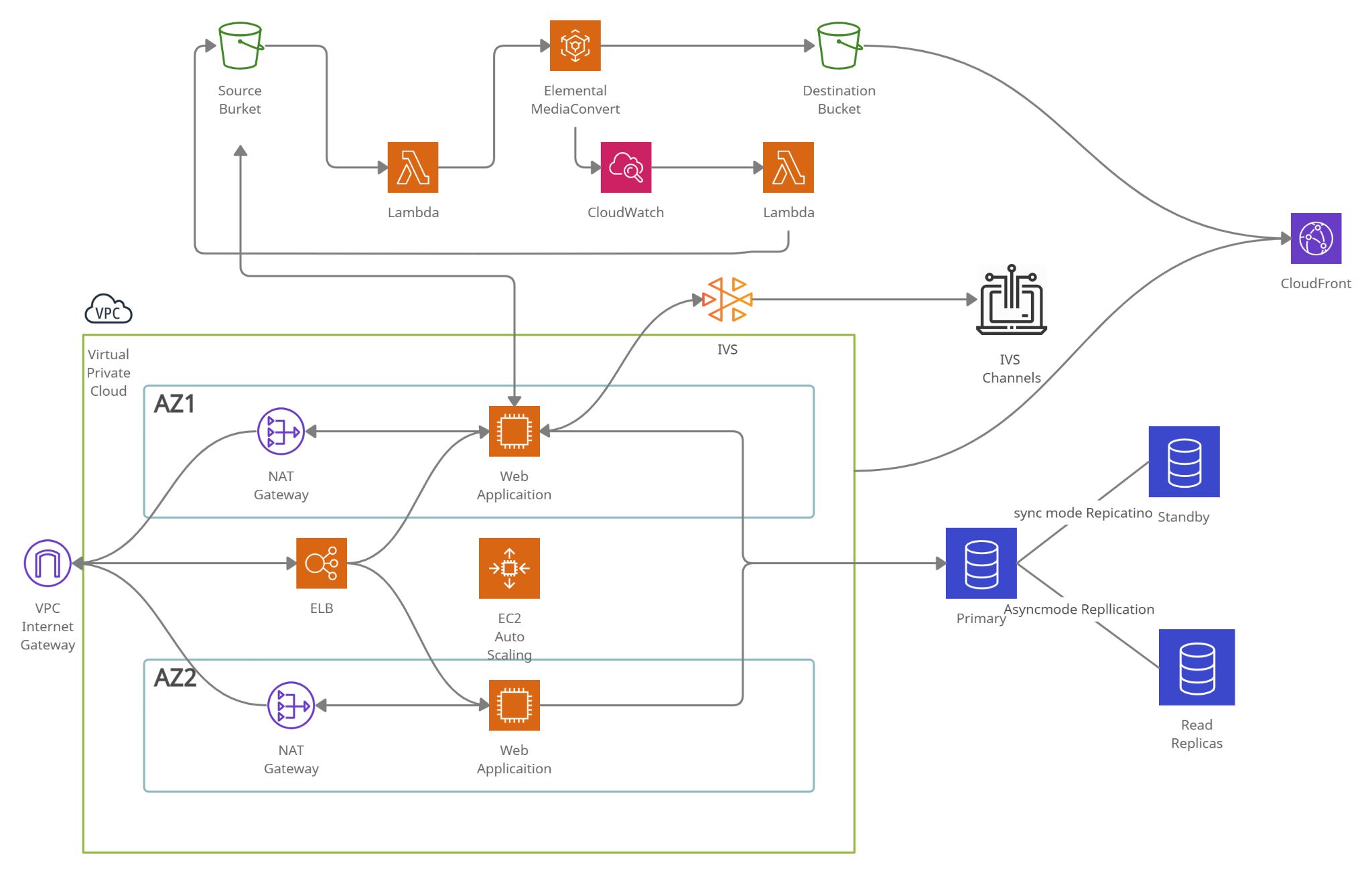


<그림 3-1 팀 BOGOSA 프로젝트 개발환경>

* Eclipse ver Photon
* JAVA 1.8.0\_291
* Jquery 3.6.0
* httpd 2.4.48
* Apache Tomcat 9.0.0.M22
* MySQL 11.0.0.5919
* MyBatis 3.0
* Bootstrap 5.0.1
* HTML 5
* Jquery AJAX 1.7.1
* Javascript ES9
* aws-sdk 2.897.0

## **3.2 서비스 아키텍처**

### **아키텍처 구성도**



<그림 3-2 전체 아키텍처 구성도>

모든 요구사항을 고려하여 설계한 아키텍처는 <그림 3-2>과 같다. 아키텍처의 각 요소를 목적에 따라 분리 해보면 다음과 같이 나눌 수 있다.

* 라이브 스트리밍 서비스 – AWS Interactive Video Service(IVS)
* CDN 아키텍처 – S3, Lambda, AWS Media Converter, Cloudfront, CloudFormation
* 고가용성 고객 데이터 베이스 - RDS(MySQL) Master and Slave
* 보안을 고려한 아키텍처 - VPC, IAM, Cognito, WAF
* 고가용성 서비스 배포 Strategy – EC2, Multi-AZ Auto scaling and Load Balancing

### **서비스 요약**

해당 프로젝트의 메인 기능이라고 할 수 있는 스트리밍 서비스를 안정적으로 제공하기 위해 완전관리형 서비스인 IVS를 활용한다. 그리고 웹서비스 내 영상컨텐츠 업로드 및 재생을 위해 CDN 아키텍처를 구성한다. 이 때 서버의 로컬을 거치지 않고 다이렉트로 S3 객체 스토리지에 업로드 하도록 하며 스트리밍에 적합한 파일로 변환 및 CDN 배포까지의 일련의 과정을 자동화 하기 위해 MeciaConverter, Lambda, CloudFront 서비스를 이용한다. Failover를 대비하여 서로 다른 Available Zone(이하 AZ)에 데이터베이스를 이중화설계 하였고 WAS 또한 같은이유로 2 개의AZ에 각각 최소 1개의 서버를 배치하고 성능저하를 방지하기 위해 Autoscaling 정책과 부하분산을 위한 LoadBalancer를 적용하였다.

최종적으로 보안을 강화하기 위해 퍼블릭 엑세스 가능 여부 및 적합한 계층에 따라 VPC 서브넷 및 보안그룹을 설정해 주었고 웹서비스가 접근하는 AWS 서비스에 대한 자격증명, 외부 공격으로부터 웹서비스를 방어하기 위한 목적으로 각각 Cognito, WAF 서비스를 적용하였다.

### **라이브 스트리밍 서비스**

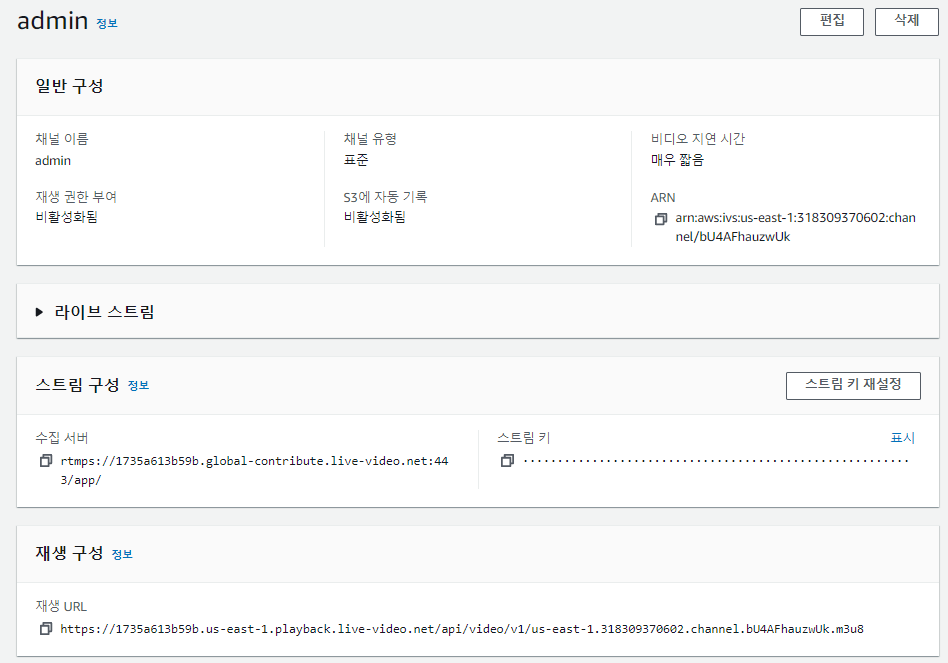
* **웹페이지 내 라이브 스트리밍 구현을 위한 IVS(Interactive Video Service) 서비스**

IVS 서비스는 대화형 동영상 환경을 구현하기에 적합한 관리형 라이브 스트리밍 솔루션이다. 스트리밍 소프트웨어를 사용하여 Amazon IVS로 라이브 스트림을 전송하면 해당 서비스에서 전 세계 모든 최종 사용자에게 짧은 지연 시간으로 라이브 스트림을 제공하는 데 필요한 모든 작업을 처리해주기 때문에 사용자는 라이브 동영상과 함께 대화형 환경을 구축하는 데 집중할 수 있다. 이는 라이브 커머스 환경에서 스트리머와 소비자의 소통을 가능하게 하는 채팅 기능을 사용하기에 적합하며 관리형 서비스이므로 스트리밍 서비스가 다운될 위험을 최소화 할 수 있다.

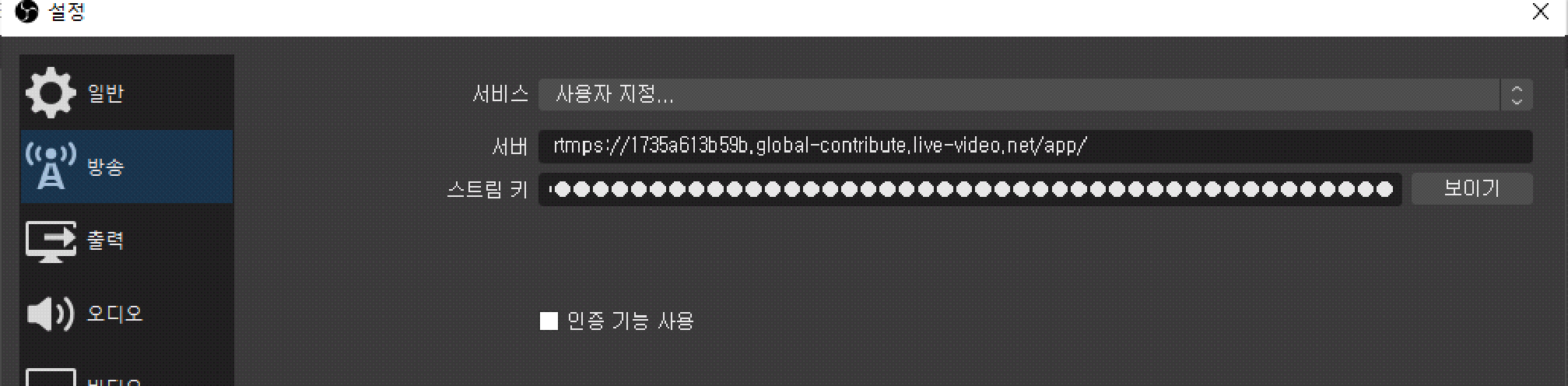


<그림 3-3 Amazon Interactive Video Service의 작동 방식>

1. 스트림 구성 및 재생구성 : IVS에서 채널을 생성하면 스트림키와 스트림 수집채널, 그리고 재생URL을 얻게된다. 웹 어플리케이션에서 가입하는 회원들 각각 채널을 가지게 되며 스트리밍 소프트웨어에서 수집채널, 스트림키를 등록하여 라이브 방송을 진행할 수 있다.



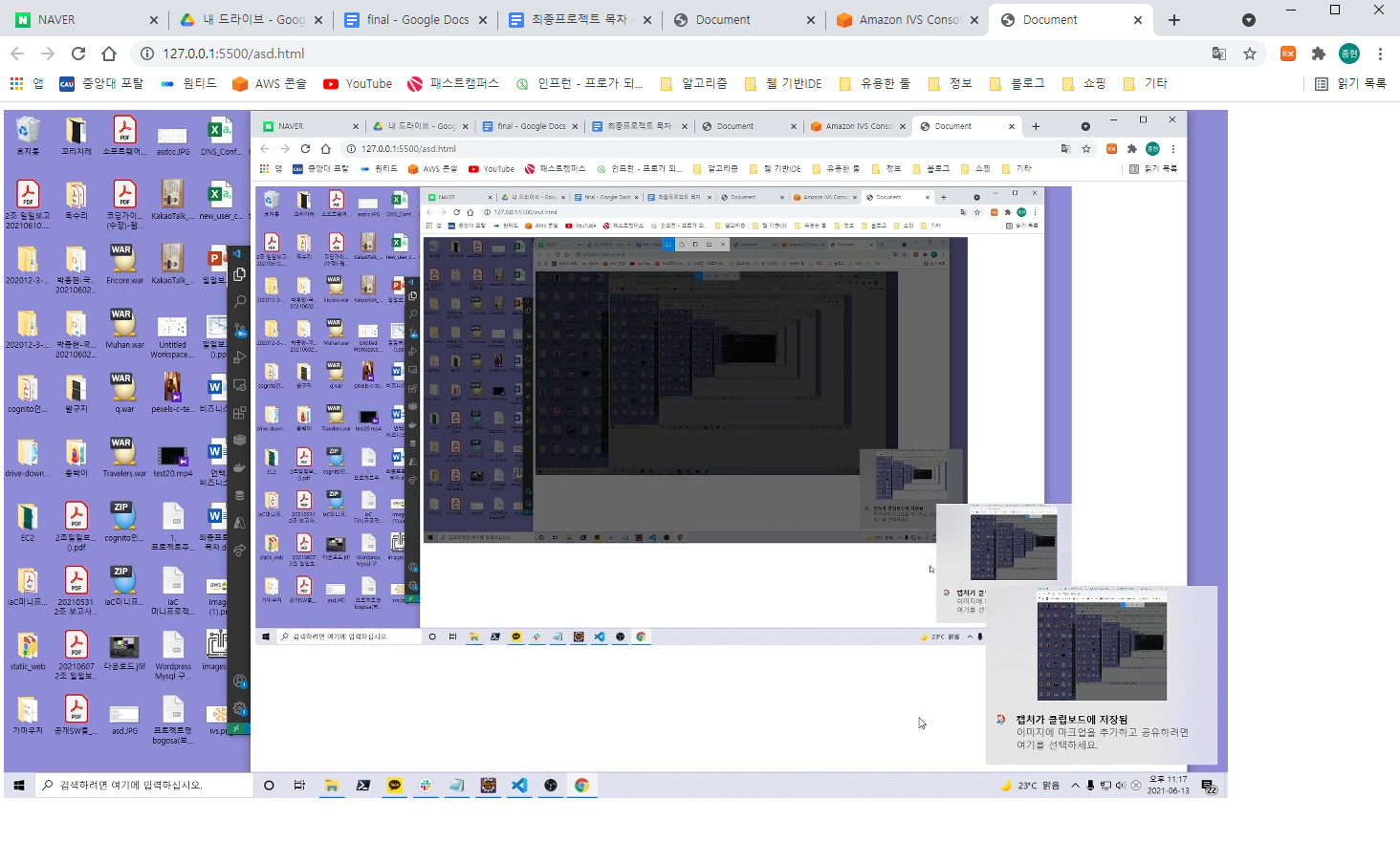
<그림3-4 채널 생성 시 채널 정보>



<그림3-5 OBS 프로그램을 통한 스트리밍 방송>

|  |
| --- |
| <script src="https://player.live-video.net/1.3.1/amazon-ivs-player.min.js"></script> <video id="video-player" playsinline></video> <script>  **if** (IVSPlayer.isPlayerSupported) {  **const** player = IVSPlayer.create();  player.attachHTMLVideoElement(document.getElementById('video-player'));  player.load(PLAYBACK\_URL);  player.play();  } </script> |

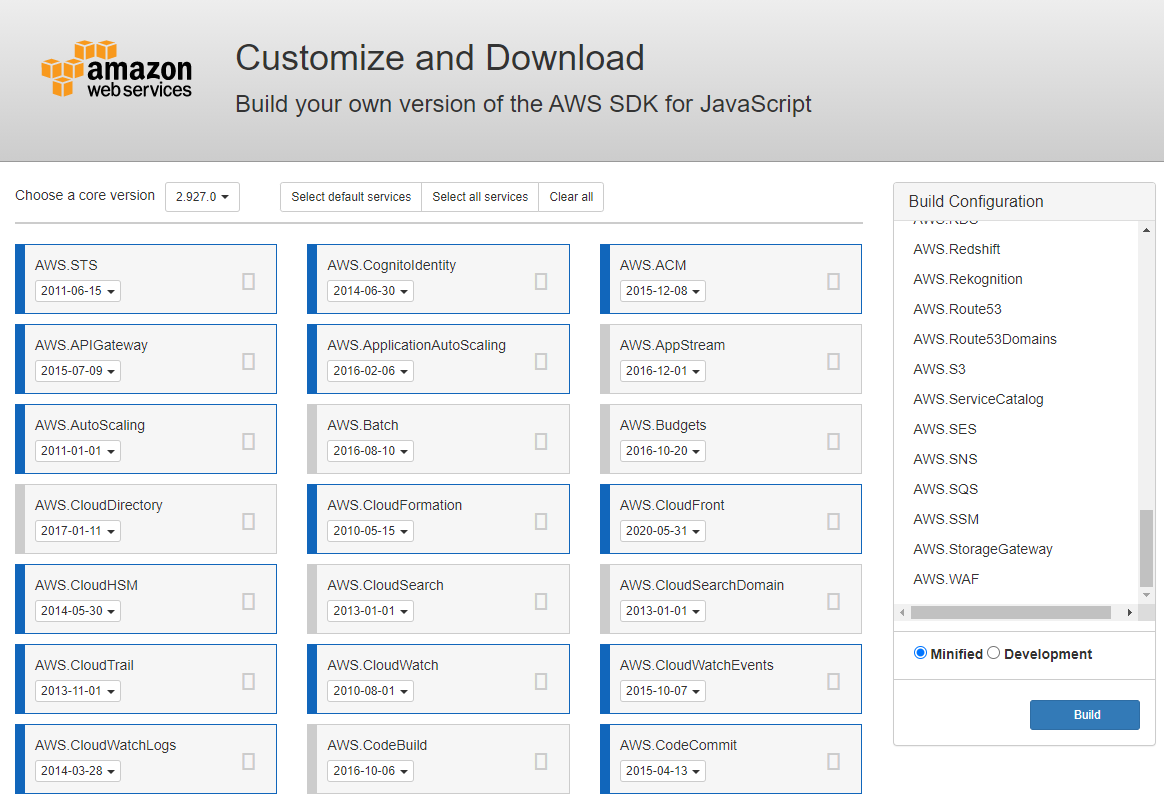
<그림3-6 웹페이지 내 라이브 스트리밍 플레이어 >



<그림3-7 웹페이지 내 플레이어 재생(실시간 화면) >

1. 웹 어플리케이션에서의 활용

웹 어플리케이션의 브라우저에서 AWS 리소스 에 접근하고 데이터베이스로 활용할 수 있도록 하기 위해 AWS 서비스별 객체를 제공하는 AWS-SDK for JavaScript 모듈을 임포트 한다. (<https://sdk.amazonaws.com/builder/js/> 에서 원하는 서비스만 선택해서 aws-sdk를 빌드할 수 있다.)



<그림3-8 브라우저용 SDK 빌드>



<그림3-9 브라우저용 SDK 임포트>

그리고 서비스 리소스에 접근하기 위해서는 권한을 가진 Account 또는 Cognito를 이용한 임시자격 증명과정이 필요하다. (프로젝트에서는 후자의 방식을 선택했는데, 그 이유는 전자의 방식에서는 local의 configure파일을 브라우저에서 인식하지 못하기 때문에 계정의 키값을 하드코딩 해야한다. 때문에 보안성 측면에서 심각하게 취약하다고 판단되었고 Cognito 방식을 이용해 권한을 얻는 방식을 선택하게 되었다. Cognito에 관한내용은 을 참고) IVS서비스에 접근할 수 있게 되면 회원가입 시 채널을 생성하는 함수를 실행하도록 하고 콜백함수로 스트림키, 수집서버, 재생URL을 가져오도록 하여 이를 데이터베이스에 저장하도록 설계한다.

- 채널 생성함수 : createChannel



<그림 3-10 createChannel 함수 형식>



<그림 3-11 createChannel 함수 Request Syntax>



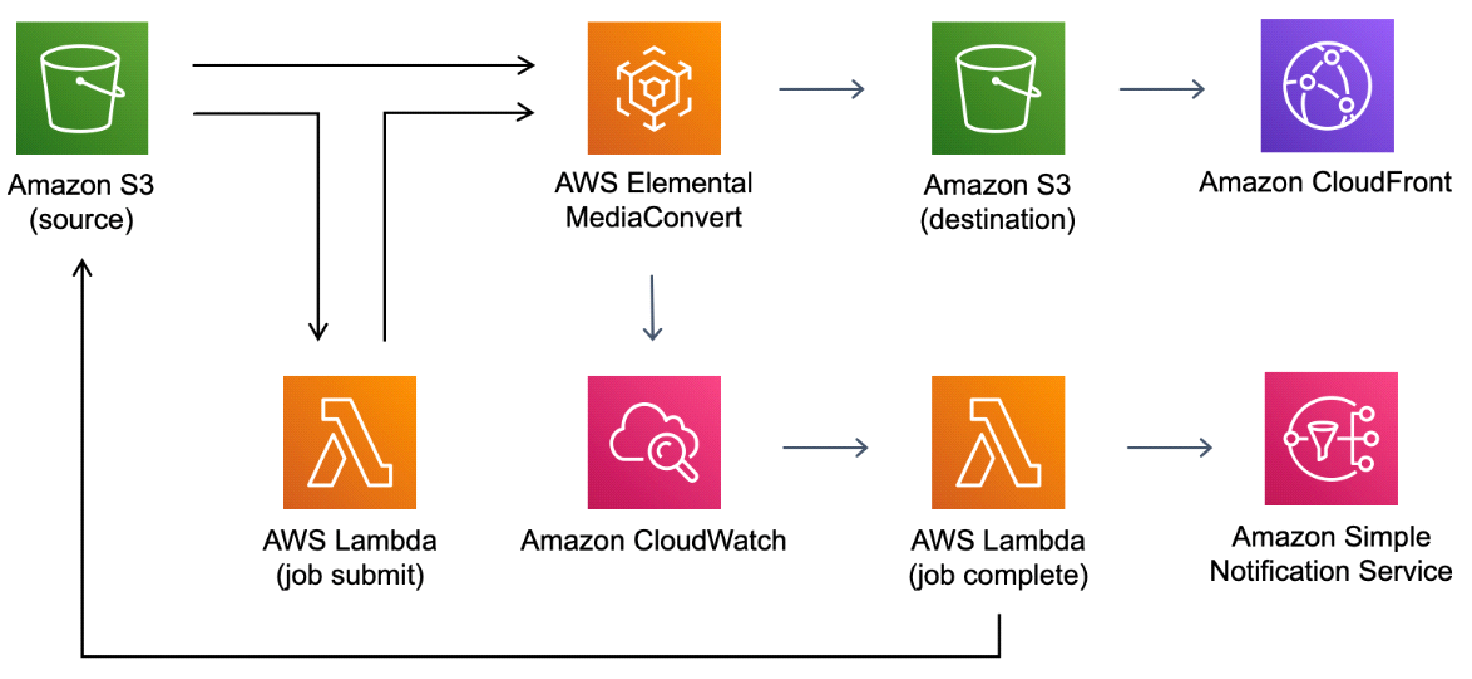
<그림 3-12 createChannel 함수 Response Syntax>

|  |
| --- |
| *// 회원가입 시 채널생성 함수 및 콜백함수* **function** **AWScheck**(){  **var** user\_id = $('#id\_val').val();  **if** (!user\_id){  alert('아이디를 입력해주세요')  **return** false;  }**else**{  **var** ivs = **new** AWS.IVS();   **var** params = {  authorized : false ,  name: user\_id *//{user\_id를 채널이름으로}*  };  ivs.createChannel(params, **function**(err, data) {  **if** (err) console.log(err, err.stack); *// an error occurred*  **else** console.log(data);   **var** a = data.channel.playbackUrl; *// 영상 url*  **var** b = data.streamKey.value; *// 키값*  **var** c = data.channel.ingestEndpoint; *// 키값*  **var** user\_charn = data.channel.arn;  document.getElementById("a").value = a  document.getElementById("b").value = b  document.getElementById("c").value = c  document.getElementById("user\_charn").value = user\_charn  AWS = 1;  alert("채널 생성완료!")  *// return a, b; // successful response*  });  } } |

이렇게 회원데이터에 저장된 스트림 리소스들은 판매자가 라이브 방송을 진행하면 (상품을 등록할 때) 판매자의 재생URL을 통해 라이브 방송을 시청하게 된다. 추가로 채널 시청자수 정보를 가져와서 웹페이지에 노출시켜 주었다.

|  |
| --- |
| ***//view\_count 노출 코드* function ChennelViewCount() {  var ivs = new AWS.IVS();  var user\_charn = $("#user\_charn${menu.product\_id}").val();  var params = {  channelArn: user\_charn,  };  ivs.getStream(params, function (err, data) {  if (err) document.getElementById("aa${menu.product\_id}").value = "방송 준비중";  *// an error occurred*  else var view\_count = data.stream.viewerCount;  if (view\_count === undefined) {  document.getElementById("aa${menu.product\_id}").value = "방송 준비중";  $("#video-player${menu.product\_id}").hide();  $("#ready-image${menu.product\_id}").show();  } else {  document.getElementById("aa${menu.product\_id}").value = view\_count + "명";  $("#video-player${menu.product\_id}").show();  $("#ready-image${menu.product\_id}").hide();  }  }); }** |

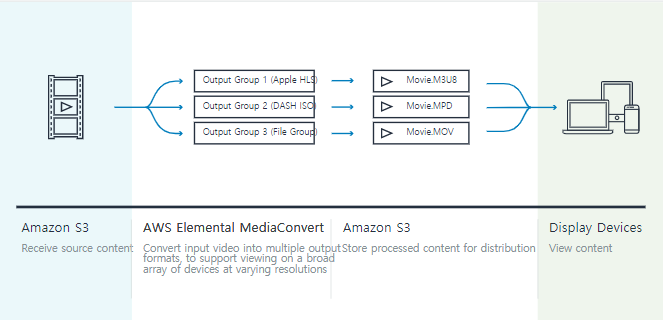
### **CDN아키텍처 - 온디멘드(OnDemand) 콘텐츠 배포**



<그림 4-1 컨텐츠 업로드 ~ CloudFront 배포 자동화 아키텍처>

* **AWS Media Converter**

AWS Elemental MediaConvert는 브로드캐스트 수준의 기능을 제공하는 파일 기반 비디오 트랜스코딩 서비스이다. 이 서비스를 VOD(주문형 비디오) 콘텐츠를 손쉽게 생성할 수 있다. 해당 프로젝트에서는 등록할 상품에 대한 예고영상을 스트리밍으로 제공하기 위해서 사용하고 있으며 추후 IVS 서비스의 영상 저장(S3 Bucket)서비스와 연동하여 다시보기 서비스를 제공하기 위해 사용될 수 있다.



<그림 4-2 AWS Media Converter의 작동 방식>

<그림 3-15>의 아키텍처에서 알 수 있듯이 Source Bucket에 영상이 업로드 되면(Lambda EventInvokeConfig)람다함수가 호출되며 MediaConverter를 통해 Destination Bucket에 VOD가 저장된다. 이 때 작업내용은 Source Bucket 내의 Job-Settings.json를 따른다.

Media Converter의 IAM역할은 다음 정책을 허용한다.

* API 실행
* Source Bucket으로부터의 Read
* Destination 버킷에 Write



<그림 4-3 AWS Media Converter Role>

* **S3 (Simple Storage Service)**

S3는 업계 최고의 확장성, 데이터 가용성 및 보안과 성능을 제공하는 객체 스토리지 서비스이다. 또한 AWS 내에서 다양한 서비스가 S3 리소스에 접근하여 작업할 수 있도록 많은 지원이 제공되는 장점이 있다. 위에서 언급한 바가 있듯이 해당 프로젝트는 웹 어플리케이션에 업로드하는 영상을 VOD 형태로 제공하기 위해서 업로드용 Bucket, VOD 저장용 Bucket 두 가지로 구성되고 이를 각각을 Source Bucket, Destination Bucket이라고 한다.

* Source Bucket : Source Bucket은 API를 통해 영상파일이 업로드 되므로 Public Access를 차단하는것 외에 별도의 정책은 설정하지 않는다.
* Destination Bucket : Destination Bucket의 경우 CloudFront의 Origin으로 설정되기 때문에 Origin Access를 허용하는 정책이 필요하다. 또한 CloudFront에 캐싱되는 파일은 Destination Bucket의 자원이므로 이에 접근할 수 있도록 CORS(Cross Origin Resource) 설정을 필요로 한다



<그림 4-4 Destination Bucket의 Bucket Policy>



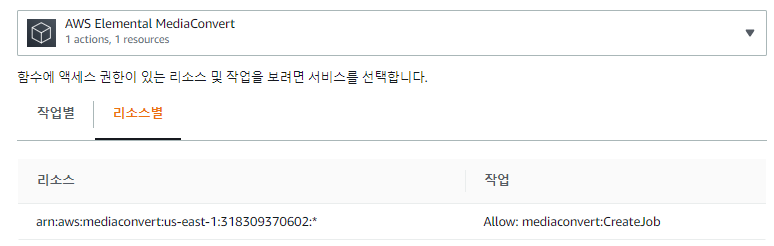
<그림 4-5 Destination Bucket의 CORS>

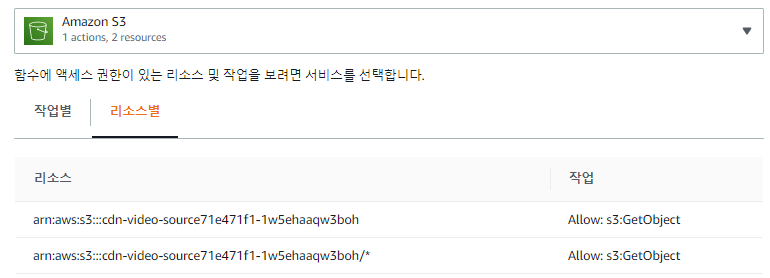
* **Lambda**

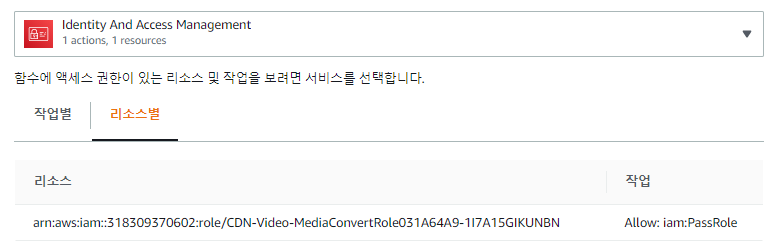
AWS Lambda는 AWS에서 제공하는 서버리스 컴퓨팅 서비스이다. 서버리스 컴퓨팅이란 애플리케이션을 실행하기 위한 별도의 서버 셋업없이 코드를 실행해주는 서비스를 의미하며 거의 모든 프로그래밍 언어를 네이티브하게 지원한다. 해당 프로젝트에서는 Node.js 12.x 런타임을 사용한다. 람다는 특정 이벤트를 기반으로 요청받은 즉시 실행된다. API 게이트웨이나 애플리케이션 로드밸런서가 받은 요청을 기반으로 실행할 수도 있으며, AWS의 다양한 서비스와 연동할 수 있다. 또한 EventBridge를 통해 외부 서비스의 이벤트로 람다 함수를 실행하는 것도 가능하다. 해당 CDN 아키텍처에서는 JobSubmit함수와 JobComplete함수가 작동한다. JobSubmit함수는 Source Bucket에 영상이 업로드 되면 MediaConverter를 동작시켜 업로드 되는 영상을 Destination Bucket에 VOD 형태로 변환하여 저장하도록 한다. JobComplete함수는 MediaConverter의 영상 변환 작업이 마무리 되면 발생하는 이벤트 패턴을 EventBridge 규칙으로 등록하여 해당 패턴이 발생할때 실행되도록 하고 SourceBucket의 manifest파일에 Job Detail Log를 기록한다. 이를 통해 manifest파일에 저장된 로그를 파싱하여 각 영상의 CloudFront Domain을 데이터 베이스에 저장하는데 사용할 수 있다.

* JobSubmit Function

1. IAM Role(요약)

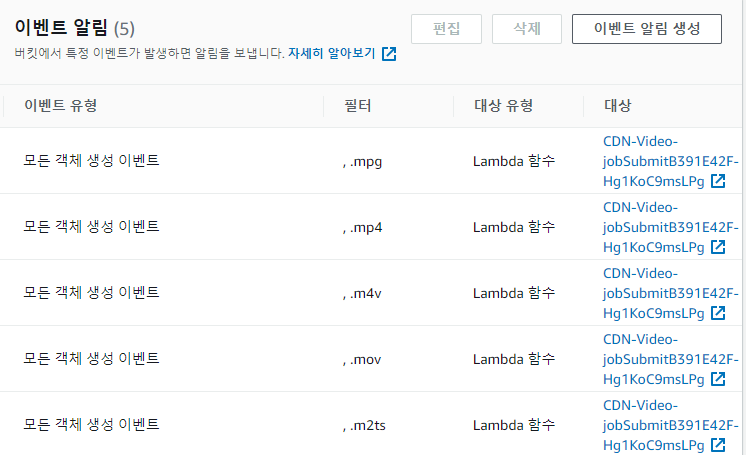






<그림 4-6 JobSubmit Function IAM Role>

1. Trigger



<그림 4-7 JobSubmit Function Trigger>

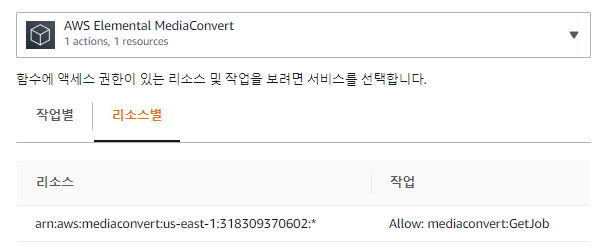
1. 환경변수 및 실행 함수(주요 코드)

|  |  |
| --- | --- |
| DESTINATION\_BUCKET | cdn-video-destination920a3c57-xtxgwh4wzkcc |
| JOB\_SETTINGS | job-settings.json |
| MEDIACONVERT\_ENDPOINT | [https://vasjpylpa.mediaconvert.us-east-1.amazonaws.com](https://vasjpylpa.mediaconvert.us-east-1.amazonaws.com/) |
| MEDIACONVERT\_ROLE | arn:aws:iam::318309370602:role/CDN-Video-MediaConvertRole031A64A9-1I7A15GIKUNBN |
| SNS\_TOPIC\_ARN | arn:aws:sns:us-east-1:318309370602:CDN-Video-NotificationSnsTopicB941FD22-1WOEYSOX4UOET |
| SNS\_TOPIC\_NAME | CDN-Video-NotificationSnsTopicB941FD22-1WOEYSOX4UOET |
| SOLUTION\_ID | SO0146 |
| STACKNAME | CDN-Video |

|  |
| --- |
| **try** {  *// define inputs/ouputs and a unique string for the mediaconver output path in S3.*   console.log(event);  **const** srcVideo = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));  **const** srcBucket = decodeURIComponent(event.Records[0].s3.bucket.name);  **const** settingsFile = `${srcVideo.split("/")[0]}/${JOB\_SETTINGS}`;  **const** guid = uuidv4();  **const** inputPath = `s3://${srcBucket}/${srcVideo}`;  **const** outputPath = `s3://${DESTINATION\_BUCKET}/${guid}`;  **const** metaData = {  Guid:guid,  StackName:STACKNAME,  SolutionId:SOLUTION\_ID  };  *// JobSettings 파일을 다운로드 하고 검증, Video INput, Output Path 지정*   **let** job = **await** utils.getJobSettings(srcBucket,settingsFile);  *// JobSettings 파일을 파싱하여 Source/Destination 대상 업데이트*   job = **await** utils.updateJobSettings(job,inputPath,outputPath,metaData,MEDIACONVERT\_ROLE);  *// MediaConverter 앤드포인트로 작업 제출* |

* JobComplete Function

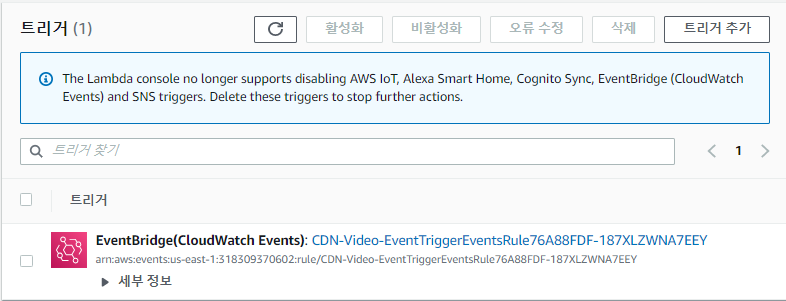
1. IAM Role(요약)

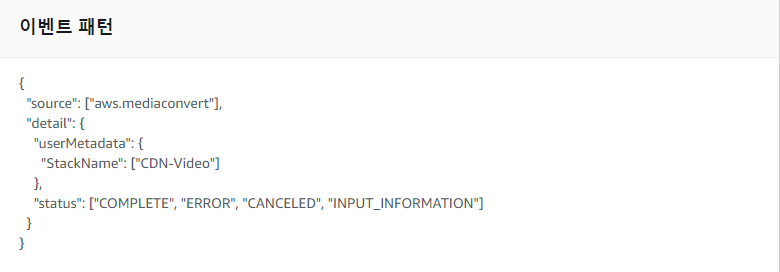




<그림 4-8 JobComplete Function IAM Role>

1. Trigger





<그림 4-9 JobComplete Function Trigger>

1. 환경변수 및 실행 함수(주요 코드)

|  |  |
| --- | --- |
| CLOUDFRONT\_DOMAIN | d25l8kvxym4ist.cloudfront.net |
| JOB\_MANIFEST | jobs-manifest.json |
| MEDIACONVERT\_ENDPOINT | [https://vasjpylpa.mediaconvert.us-east-1.amazonaws.com](https://vasjpylpa.mediaconvert.us-east-1.amazonaws.com/) |
| METRICS | Yes |
| SNS\_TOPIC\_ARN | arn:aws:sns:us-east-1:318309370602:CDN-Video-NotificationSnsTopicB941FD22-1WOEYSOX4UOET |
| SNS\_TOPIC\_NAME | CDN-Video-NotificationSnsTopicB941FD22-1WOEYSOX4UOET |
| SOLUTION\_ID | SO0146 |
| SOURCE\_BUCKET | cdn-video-source71e471f1-1w5ehaaqw3boh |
| STACKNAME | CDN-Video |
| UUID | a09ce11d-aada-499b-bb27-f2a6b310a2b9 |

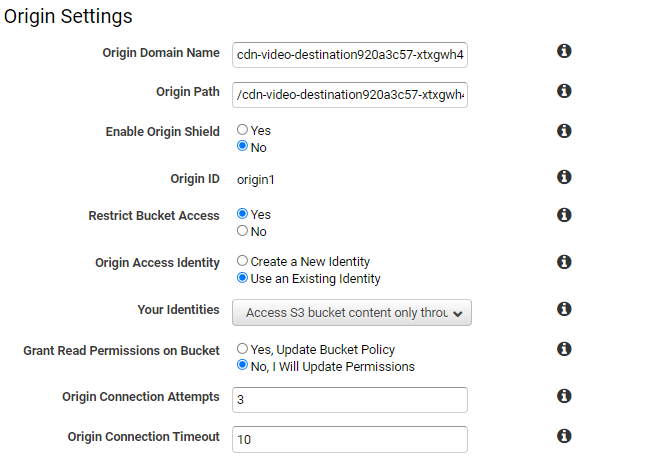
|  |
| --- |
| **try** {  **const** status = event.detail.status;    **switch** (status) {  **case** 'INPUT\_INFORMATION':  */\*\*  \* Source의 정보를 manifest 파일에 write  \*/*  **try** {  **await** utils.writeManifest(SOURCE\_BUCKET,JOB\_MANIFEST,event);  } **catch** (err) {  **throw** err;  }  **break**;  **case** 'COMPLETE':  **try** { *//미디어 변환 작업 세부 정보를 가져오고 이벤트 출력을 파싱한다*  **const** jobDetails = **await**  utils.processJobDetails(MEDIACONVERT\_ENDPOINT,CLOUDFRONT\_DOMAIN,event); *// manifest파일 업데이트*  **const** results = **await** utils.writeManifest(SOURCE\_BUCKET,JOB\_MANIFEST,jobDetails); *// 활성화된 경우 문제성 데이터를 Solution Builder api로 전송*  **if** (METRICS === 'Yes') {  **await** utils.sendMetrics(SOLUTION\_ID,VERSION,UUID,results);  } *// sns로 작업내역 전송, (해당 프로젝트에서는 sns 활용하지 않음)*  **await** utils.sendSns(SNS\_TOPIC\_ARN,STACKNAME,status,results);  } **catch** (err) {  **throw** err;  }  **break**;  **case** 'CANCELED':  **case** 'ERROR': *// sns를 통해 에러메세지 전송 (해당 프로젝트에서는 sns 활용하지 않음)*  **try** {  **await** utils.sendSns(SNS\_TOPIC\_ARN,STACKNAME,status,event);  } **catch** (err) {  **throw** err;  }  **break**;  **default**:  **throw** **new** Error('Unknow job status'); |

* **Edge Location을 통한 컨텐츠 저지연 배포 Strategy - CloudFront**

CloudFront는 AWS에서 제공하는 CDN(컨텐츠 전송 네트워크)이다. CloudFront는 오리진 서버에 위치한 원본파일을 전 세계에 위치한 Edge Location으로 배포하고, Edge Location은 이 데이터를 캐싱하며 사용자는 자신의 위치와 가까운 Edge Location으로부터 데이터를 제공 받으므로 이미지, 오디오, 비디오 및 웹페이지 등을 저지연 서비스를 제공받을 수 있다.

해당 프로젝트에서는 MediaConverter를 통해 스트리밍용 영상파일으로 변환되어 저장되는 Destination Bucket을 Origin으로 설정한다. 서비스 이용자는 엣지로케이션을 통해 캐싱된 스트리밍 컨텐츠(예고영상)을 시청하게 되고 위치에 관계없이 빠르게 컨텐츠를 이용할 수 있다.

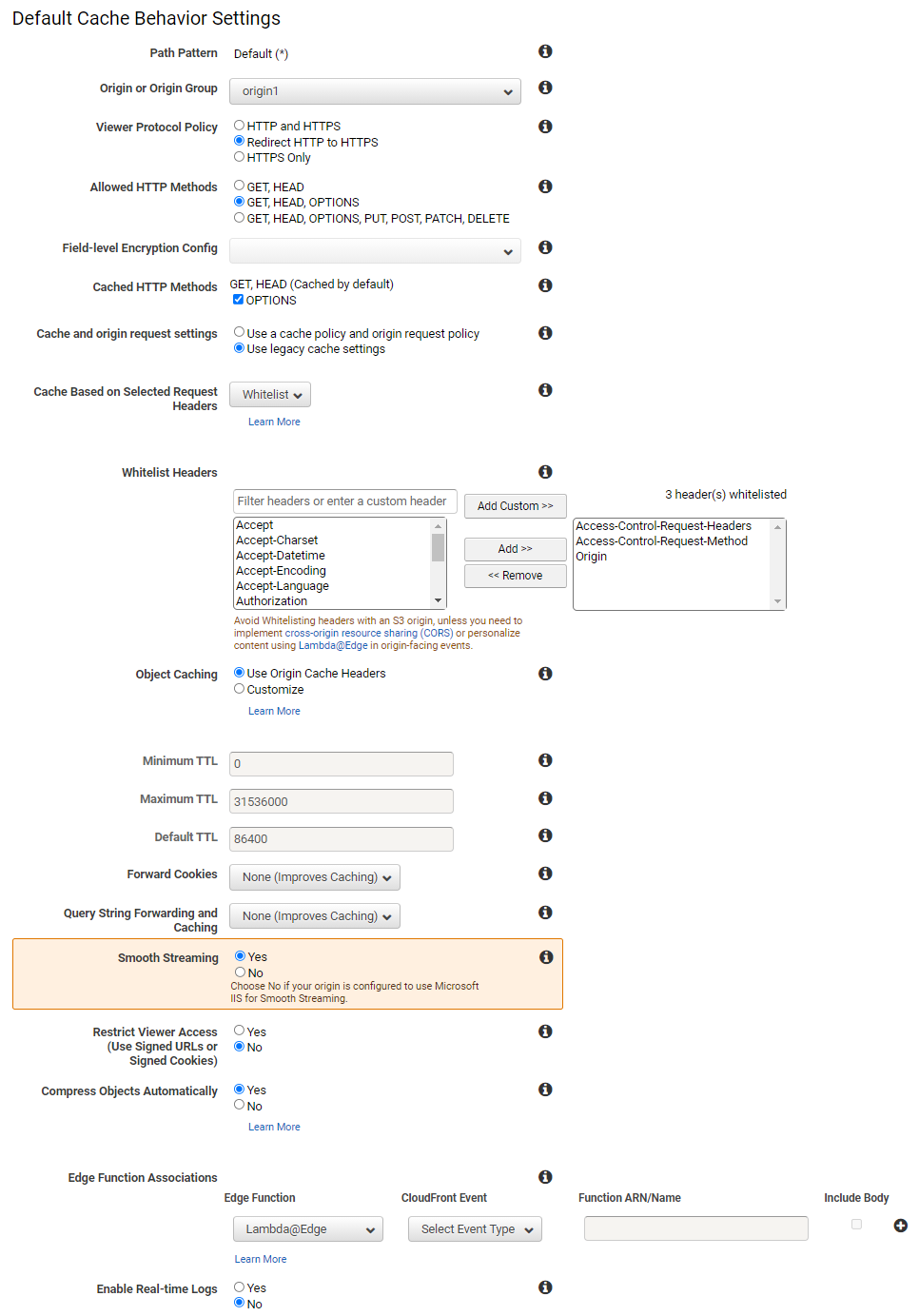
1. Origin Settings



<그림 -4-10 CloudFront Origin Setttings>

CloudFront를 통해서만 Origin에 접근할 수 있도록 설정하였으며 나머지는 기본 설정값으로 두었다.

1. 캐시 정책 설정(Behaviors)



<그림 4-11 CloudFront Behavior Settings>

앞서서 Destination Bucket에 대해 설정할 때 CORS 설정을 해 두었다. Cloud Front에 해당 버킷을 Origin으로 붙여서 사용한다면, 반드시 CORS 관련 Header 값을 Cloud Front에서 S3까지 전달 될 수 있도록 요청 헤더에 대하여 전달을 허용 해주어야 한다. 위의 Whitelist Header 를 다음과 같이 추가해 주고, CloudFront, Viewer 사이 HTTP 프로토콜 및 Request method를 설정한다. MediaConverter를 통해 변환된 스트리밍 파일인 경우 CDN에서 Smooth Streaming을 지원한다. 나머지는 기본 설정으로 둔다.

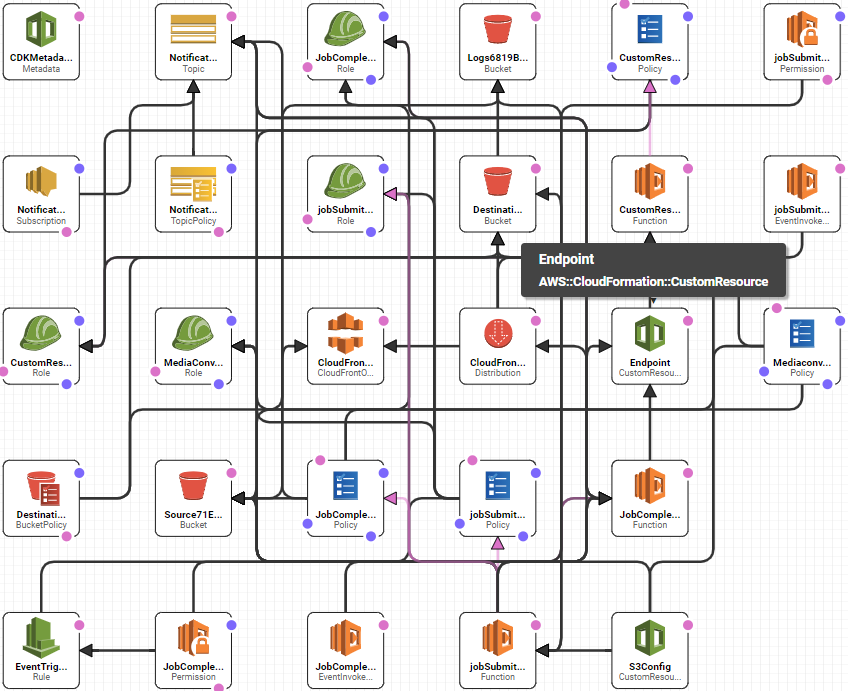
* 웹 어플리케이션에서의 CDN 서비스 활용

사용자가 웹 페이지에서 동영상을 업로드하면 Source Becket에 업로드되고 일련의 변환과정을 거친 스트리밍 영상파일에 대한 CloudFront 배포주소를 데이터베이스에 등록하여 웹페이지에 보여지도록 설계하였다. 여기서 S3 리소스에 접근이 필요하기 때문에 API를 활용한다. 마찬가지로 Cognito 자격증명을 통해 S3서비스와 리소스에 대한 접근권한을 가지게 되고 다음 함수를 실행한다.

|  |
| --- |
| **var fileChooser = document.getElementById('file-chooser'); var button = document.getElementById('upload-button'); var results = document.getElementById('results'); var session = $('#session').val(); var session = $('#session').val(); button.addEventListener("click", function() {  var file = fileChooser.files[0];  if (file) {  results.innerHTML = '';  var objKey = 'assets01/' + session + file.name;  *// 버킷 내에 /assets01 디렉토리에 업로드될 파일명 지정*  var params = {  Key: objKey,  ContentType: file.type,  Body: file,  ACL: 'public-read'  };  bucket.putObject(params, function(err, data) {  if (err) {  results.innerHTML = 'ERROR: ' + err;  document.getElementById('inputform').submit();  } else {  console.log(data);  var s3 = new AWS.S3();  *// S3 객체 생성*  var timer = setInterval(function() {  *//manifest 파일이 업데이트 될때 까지 반복*  s3.getObject({  Bucket: "cdn-video-source71e471f1-1w5ehaaqw3boh", *//Source Bucket 이름*  Key: "jobs-manifest.json"  }, function(err, data) {  if (err) console.log(err, err.stack);  else data = data.Body.toString();  data = JSON.parse(data);  data = data.Jobs.filter(function(element) {  if (element.Job != undefined) {  return element.Job.Settings.Inputs[0].FileInput == 's3://cdn-video-source71e471f1-1w5ehaaqw3boh/assets01/' + session + file.name;  } else {  return element.InputFile == 's3://cdn-video-source71e471f1-1w5ehaaqw3boh/assets01/' + session + file.name;  *// manifest 파일에서 input파일의 이름을 통해 output 파일의 CDN 주소를 가져오도록 업로드한 파일 이름으로 필터링*  }  });  console.log(data.length);  if (data.length != 0) {  if (data[0].Outputs != undefined) {  *// Outputs 가 Undefined가 아니라면, 즉 manifest파일이 최신화 되었음을 의미*  trailer\_aws\_url = data[0].Outputs.HLS\_GROUP[0];  console.log(trailer\_aws\_url);   results.innerHTML = '<input type="hidden" name="trailer\_aws\_url" id="trailer\_aws\_url" value="' + trailer\_aws\_url + '">'  document.getElementById('inputform').submit();  clearInterval(timer);  } else {  alert("파일 업로드 중 문제가 발생했습니다. 다시 시도해 주세요.");  clearInterval(timer);  }  } else {  console.log("data is not detected")  }  })  }, 20000);  timer;  }  });  }** |

* **CloudFormation**

CloudFormation 서비스를 통해 템플릿에서 원하는 리소스와 종속성을 설명하고 이를 모두 하나의 스택으로 구성하고 시작할 수 있다. 이전에 설명한 CDN 아키텍처는 템플릿으로 구성하였고 템플릿의 Desinger View는 다음과 같다.



<그림 4-12 CDN-video Stack Designer View>

### **고가용성 고객 데이터베이스**

* **RDS**

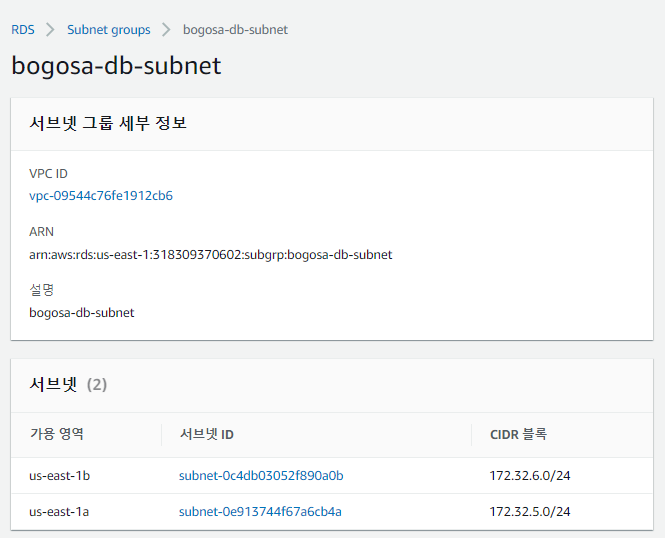
RDS는 Relational Database Services의 약자로, 클라우드에서 관계형 데이터베이스를 더욱 간편하게 설정, 운영 및 확장할 수 있는 서비스다. 하드웨어 프로비저닝, 데이터베이스 설정, 패치 및 백업과 같은 시간 소모적인 관리 작업을 자동화하면서 비용 효율적이고 크기 조정가능한 데이터베이스 서비스를 제공한다.

프로젝트에서는 관계형 데이터베이스를 이용하므로 RDS의 MySQL 엔진을 사용한다.

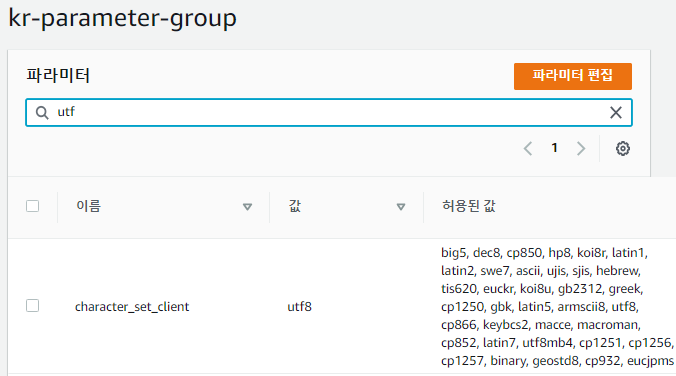
RDS는 뛰어난 확장성, 빠르게 확장 가능한 가용성과 높은 보안성을 제공한다.

이번 프로젝트에 사용한 DBMS는 MySQL이다. 고객 데이터베이스의 가용성을 보장하기위해 원본 데이터베이스와 다른 Available Zone에 대기인스턴스를 복제한다.

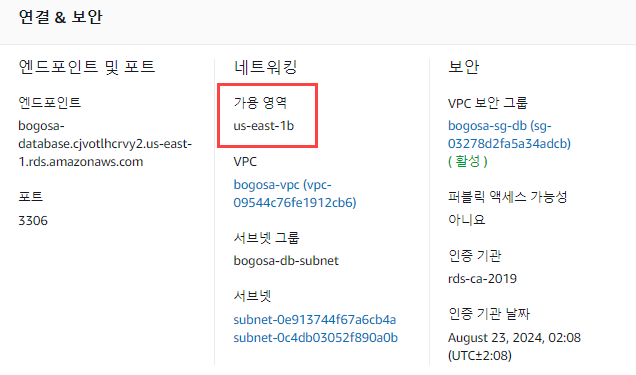
또한 데이터베이스의 한글 입출력을 지원하기 위해 파라미터그룹의 character\_set\_client 의 값을 utf-8로 지정해 준다.



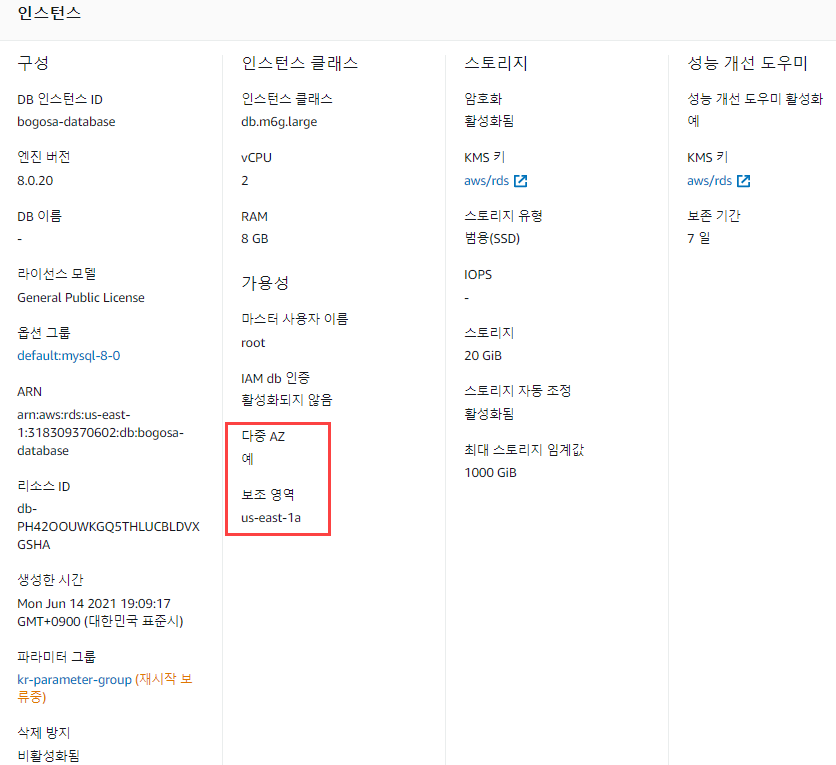
<그림 5-1 데이터 베이스 서브넷 그룹>



<그림 5-2 데이터 베이스 파라미터 그룹>



<그림 5-3 데이터 베이스 연결 및 보안설정>



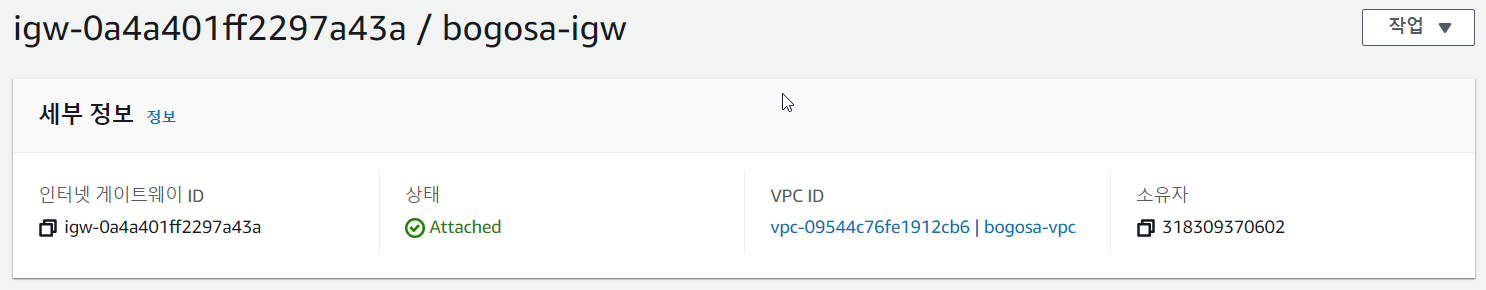
<그림 5-4 데이터 베이스 인스턴스 설정>

### **보안을 고려한 아키텍처**

* **VPC (리전 : us-east-1)**

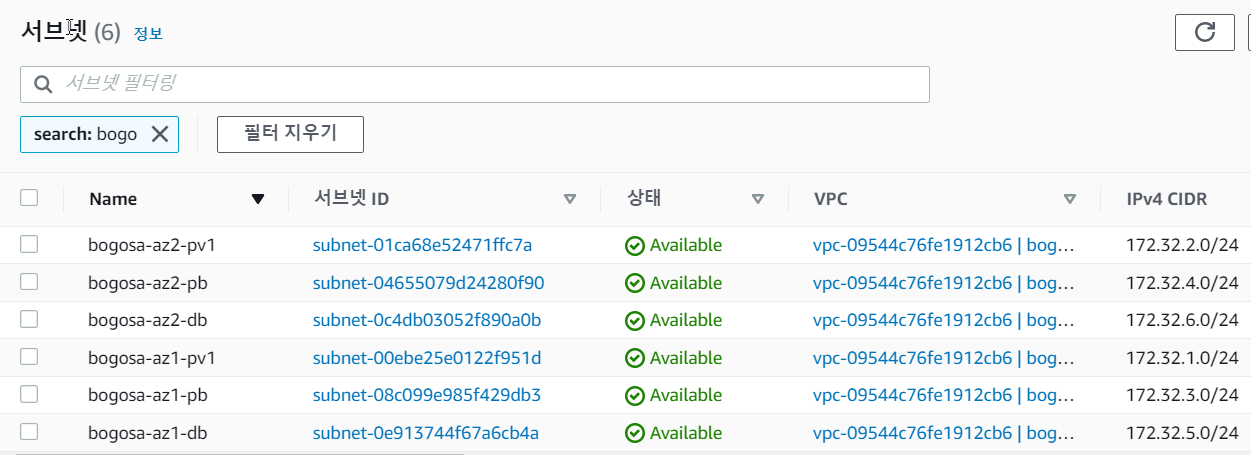
Amazon VPC는 Virtual Private Cloud의 약자로 AWS 클라우드에서 논리적으로 격리된 네트워크 공간을 할당하여 가상 네트워크에서 AWS 리소스를 이용할 수 있는 서비스를 제공한다. Amazon VPC 자체 IP 주소 범위, 서브넷 생성, 라우팅 테이블 및 네트워크 게이트웨이 구성 선택 등 가상 네트워킹 환경을 완벽하게 제어할 수 있으며, VPC에서 IPv4와 IPv6를 모두 사용하여 리소스와 애플리케이션에 안전하고 쉽게 액세스할 수 있다. 이번 프로젝트에서는 고가용성 요구사항을 이행하기 위해서 리전 내 2개의 Available Zone에 각각 Public Subnet, Private Subnet(WAS, Database)을 구성하였고 보안 강화의 측면에서 리소스의 접근을 제한하기 위해 웹 서비스 계층별로 보안그룹을 적용하였다. 세부적인 설정은 아래와 같다.

1. 인터넷 게이트 웨이



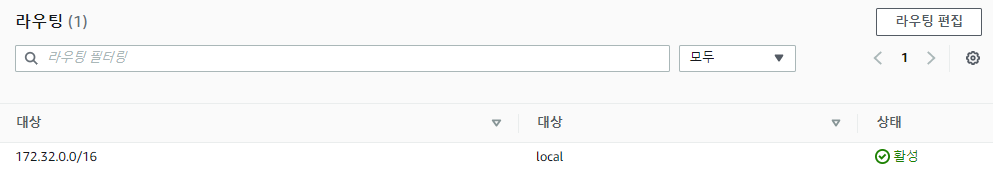
<그림 6-1 인터넷 게이트 웨이>

1. 서브넷 : 서브넷 네트워크 트래픽이 인터넷 게이트웨이(IGW)로 라우팅이 되는 서브넷을 Public Subnet, 인터넷 게이트웨이로 라우팅 되지 않는 서브넷을 Private Subnet 이라 한다. 프로젝트에서 로드밸런서는 Public Subnet에 위치, WAS Autoscaling Group과 Database는 각각 별도의 Private Subnet에 위치하도록 하였다.



<그림 6-2 서브넷>

1. 라우팅 테이블 : 서브넷 간의 통신이나 VPC 간의 원활한 통신을 위해 라우팅 테이블을 구성한다. Private Subnet은 VPC 내부 로컬에서만 라우트 하도록 하고 Public Subnet은 로컬과 인터넷 게이트웨이로 라우팅하도록 설정한다

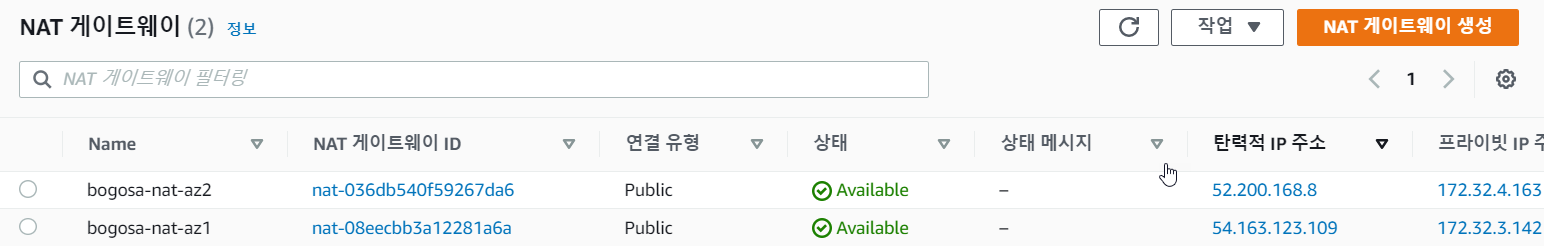


<그림 6-3 Private Subnet과 연결된 라우팅 테이블>



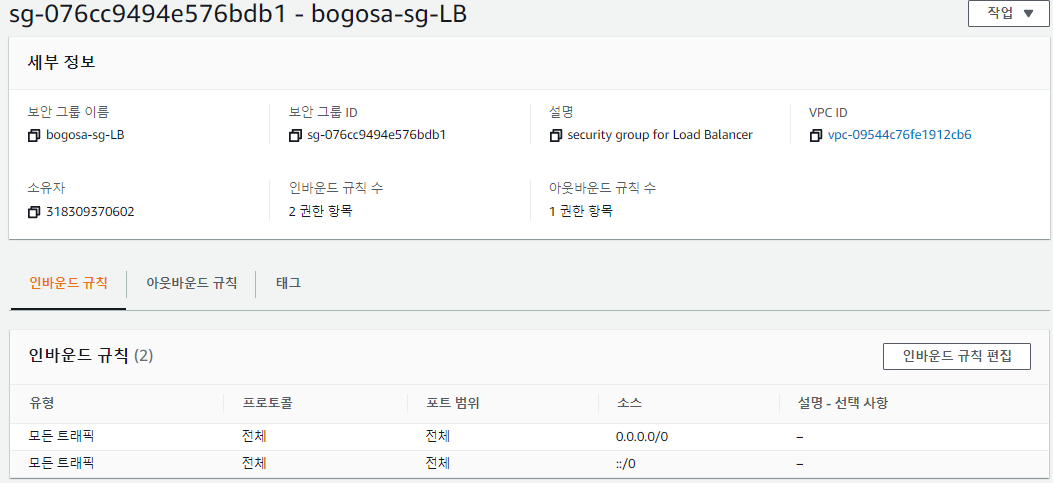
<그림 6-4 Public Subnet과 연결된 라우팅 테이블>

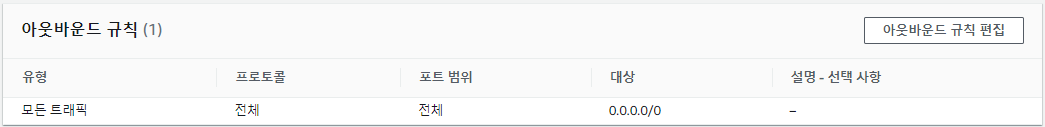
1. NAT 게이트웨이 : Private Subnet에 위치한 WAS는 외부와 통신할 수 없으므로 Public Subnet에 NAT 게이트웨이를 거쳐서 트래픽을 아웃바운딩 하도록 한다



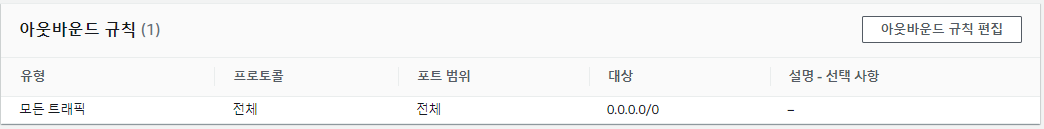
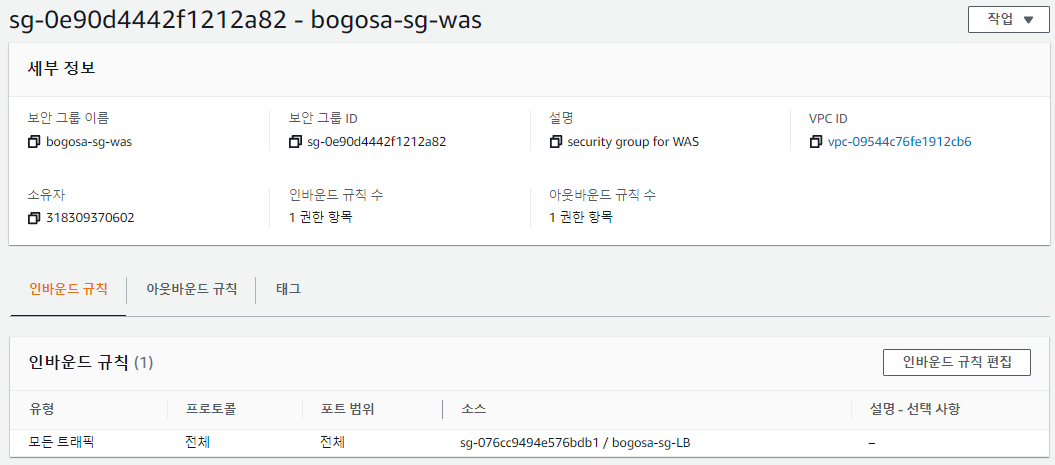
<그림 6-5 NAT게이트웨이>

1. 보안그룹 : 위와 같이 아키텍처의 각 구성요소별로 위치할 서브넷이 결정 되었으면 서브넷 상호간, 혹은 VPC 외부와 통신을 할 때 트래픽에 대한 규칙이 필요하다. 서비스를 제공하기 위해 도메인 네임서버에 노출되는 로드밸런서의 경우에는 인터넷(0.0.0.0/0)으로부터 모든 인바운드 트래픽을 허용하도록 한다. WAS는 로드밸런서로부터 인바운드 트래픽만을 허용, 데이테베이스는 WAS로부터 TCP 프로토콜 3306포트를 개방해준다. 아웃바운드 트래픽은 모두 Default설정으로 하였다.



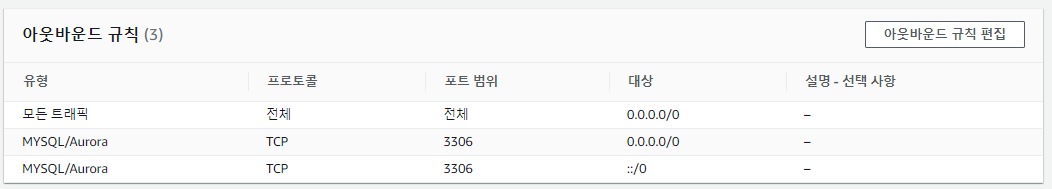


<그림 6-6 LoadBalancer 보안그룹 규칙>



<그림 6-7 WAS 보안그룹 규칙>





<그림 6-8 RDS(MySQL) Database 보안그룹>

* **IAM**

AWS Identity and Access Management(IAM)는 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있는 웹 서비스 이다. 프로젝트에서 이용하는 서비스별로 최소권한의 원칙을 준수하여 리소스간의 접근을 허용하도록 한다.

* **Cognito 역할정책**

이전에 웹 어플리케이션에서 AWS 서비스에 접근하여 기능을 구현하도록 한 부분이 있었다. 다시 언급하자면 해당 기능들은 브라우저에서 동작하도록 설계하였는데 이 때 AWS 서비스의 리소스를 생성하거나 조회할 경우 자격 증명이 필요하고 이러한 경우에 Cognito 역할정책을 설정하여 사용하는 것이 적절하다. 이를 통해 소스코드에 사용자 키를 하드코딩하는 것을 피할 수 있다. 해당 프로젝트에서는 User Pool은 이용하지 않았기 때문에 인증되지 않은 사용자로써의 자격증명만 사용한다. 웹 어플리케이션에서 api를 통해 접근하는 서비스는 IVS, S3 이며 Action과 Resource는 다음과 같다.

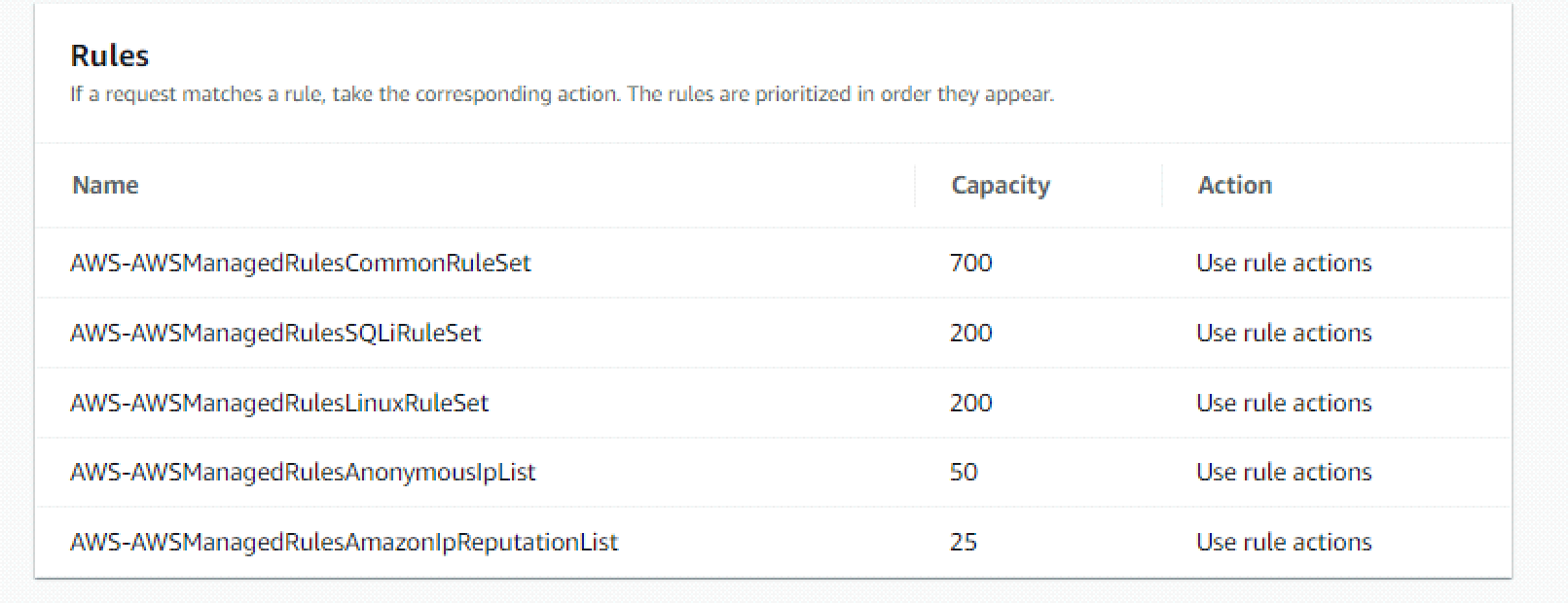
|  |  |  |
| --- | --- | --- |
|  | IVS | S3 |
| Action | createChannel, getChannel | putObject, getObject |
| Resource | Ivs-us-east-1:[계정번호]:\* | Source & Destination Bucket |



<그림 6-9 인증되지 않은 사용자 역할(Cognito\_accessivsUnauth\_Role)>

* **WAF(Web Application Firewall)**

AWS에서 관리형으로 제공하는 방화벽 서비스를 이용하여 가용성에 영향을주거나, 리소스를 과하게 사용하는 웹 공격, SQL 주입 공격으로부터 웹 애플리케이션을 보호한다. 해당 규칙을 적용한 ACL을 서비스 로드밸런서에 연결한다.



<그림 6-10 WAF Rules>

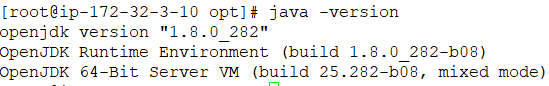
* AWSManagedRulesCommonRuleSet - OWASP 발행물 상위 10개항목에 설명된, 자주 발생하고 위험성 높은 광범위한 취약성을 악용하지 못하도록 보호
* AWSManagedRulesSQLiRuleSet - SQL 명령어 주입 공격과 같은 SQL 데이터베이스 도용과 관련된 요청 패턴을 차단하는 규칙
* AWSManagedRulesLinuxRuleSet - Linux 관련 로컬 파일 공격을 포함하여 Linux에 특정한 취약성 도용과 관련된 요청 패턴을 차단하는 규칙
* AWSManagedRulesAnonymousIpList - 최종 사용자 ID 난독화를 허용하는 서비스의 요청을 차단하는 규칙
* AWSManagedRulesAmazonIpReputationList - 내부 위협 인텔리전스를 기반으로 하는 규칙, 봇이나 다른 위협과 연결된 IP 주소를 차단하려는 경우에 사용

### **고가용성 서비스 배포 Strategy**

* **EC2를 이용한 배포**

이번 프로젝트에 사용된 인스턴스는 Amazon Linux 2 AMI (HVM), SSD Volume Type 이다. t2.micro 를 사용했으며 미리 구성한 VPC 에 생성하였다. 인스턴스에 아파치 웹서버와 아파치 톰캣 WAS를 모두 구성 하였으며 Apache 서버로 요청되는 모든 HTTP Request는 Tomcat 서버로 Redirecting 하도록 TomcatConnector(mod\_jk)를 사용하였다.

1. 인스턴스에 프로젝트에 사용된 웹서버 httpd(Apache) 설치
2. 인스턴스에 프로젝트에 사용된 웹 어플리케이션 서버 apache-tomcat9 을 설치
3. 인스턴스에 jdk 1.8.0\_282 을 설치하고 환경변수 등록



<그림 7-1 자바 환경변수 등록 및 버전확인>

1. Tomcat Connector 빌드
2. Apache 설정파일 추가 및 수정(mod\_jk, workers.porperties)

|  |
| --- |
| # /etc/httpd/conf/httpd.conf - 파일 수정  IncludeOptional conf.d/\*.conf IncludeOptional conf/extra/\*.conf |
| # Apache 최상위 설정 파일(httpd.conf)에서 앞으로 추가할 설정 파일들을 Include 할 수 있도록 IncludeOptional conf/extra/\*.conf 부분을 추가 |
| #/etc/httpd/conf/workers.properties - 신규 파일 작성  worker.list=instance  worker.instance.port=8009  worker.instance.host=127.0.0.1  worker.instance.type=ajp13 |
| # worker(tomcat server) 의 호스트, 포트와 통신프로토콜 입력 |
| # /etc/httpd/conf.modules.d/mod\_jk.conf - 신규 파일 작성  <ifModule jk\_module>  JkWorkersFile conf/workers.properties  JkLogFile logs/mod\_jk.log  JkLogLevel info  JkShmFile /var/log/httpd/jk-runtime-status  JkWatchdogInterval 30  </ifModule> |
| # 여기서는 바라볼 workers.properties 경로를 지정 및 logging 등을 설정 |
| # /etc/httpd/conf/extra/default.conf - 신규 파일 작성  LoadModule jk\_module modules/mod\_jk.so    Include conf.modules.d/mod\_jk.conf    <VirtualHost \*:80>  ServerName localhost    JkMount /\* instance  </VirtualHost> |
| # JkMount /\* instance >> 도메인이 localhost인 모든 요청은 이전에 설정해준 ajp worker의 이름인 instance로 처리하겠다라는 설정 |

|  |
| --- |
| /home/ec2-user/opt/apache-tomcat-9.0.0.M22/conf/server.xml |
| <Connector port="8443" protocol="org.apache.coyote.http11.Http11AprProtocol"  maxThreads="150" SSLEnabled="true" >  <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />  <SSLHostConfig>  <Certificate certificateKeyFile="conf/localhost-rsa-key.pem"  certificateFile="conf/localhost-rsa-cert.pem"  certificateChainFile="conf/localhost-rsa-chain.pem"  type="RSA" />  </SSLHostConfig>  </Connector>  -->  <!-- Define an AJP 1.3 Connector on port 8009 -->  <Connector port="8009" maxKeepAliveRequests="-1" protocol="AJP/1.3" redirectPort="8443" /> |

위와같이 httpd, tomcat connector, tomcat server configuraion의 설정을 통해서 인스턴스에 요청되는 Request는 다음과 같은 동작플로우를 따르게 된다.

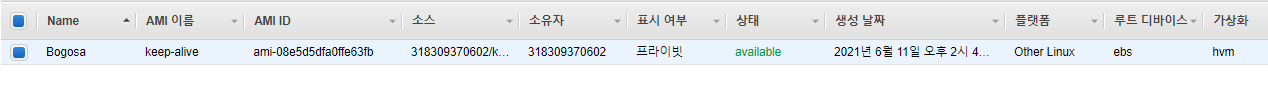
* 사용자의 브라우저는 아파치 웹서버에 접속하여 요청한다. (통상 80 port )
* 아파치 웹서버는 사용자의 요청이 들어왔을때, 이 요청이 톰캣에서 처리되도록 지정된 요청인지 확인한다.
* 톰캣에서 처리해야하 하는 경우 아파치 웹서버는 톰캣의 AJP 포트(통상 8009 port) 에 접속해 요청을 톰캣에게 전달한다.
* 톰캣은 아파치 웹서버로부터 요청을 받아 처리한 후, 처리 결과를 다시 아파치 웹서버에게 돌려준다.
* 아파치 웹 서버는 톰캣으로 전달받은 처리 결과를 사용자에게 전송한다.

1. war 파일을 tomcat 폴더 속 webapps 폴더에 복사
2. /home/ec2-user/opt/apache-tomcat-9.0.0.M22/bin에서 startup.sh 실행

>> War파일 자동으로 빌드되어 웹페이지 접속이 가능해진다.

1. 현재 인스턴스를 AMI로 이미지화

>> AMI는 Amazon Machine Image의 약자로 EC2 인스턴스 생성에 필요한 모든 소프트웨어 정보를 담고 있는 템플릿 이미지다. AMI를 이용하면 이후 동일한 환경을 갖는 인스턴스를 손쉽게 생성할 수 있다.



<그림 7-2 생성된 AMI>

* Multi-AZ AutoScaling

Auto Scaling 이란 서버나 애플리케이션을 모니터링하고 리소스를 자동으로 조정하는것을 의미하며, 최대한 저렴한 비용으로 안정적이고 예측 가능한 성능을 유지할 수 있다. 또한 여러 Available Zone에 최소 인스턴스를 설정해줄 경우 LoadBalancer와 함께 사용하면 하나의 Available Zone에 장애상황이 발생하더라도 HealthCheck를 통해 하나의 Available 서버가 비정상이라고 판단할 경우 장애가 발생한 서버로 트래픽을 전달하지 않기 때문에 Failover 수단으로 사용될 수 있고 고가용성 아키텍처로 설계할 수 있다. 이외에 장점은 다음과 같다.

* + Auto Scaling을 사용하면 애플리케이션 가용성을 간편하게 관리
  + 사용자가 정의한 조건에 따라 EC2 용량이 자동으로 확장/축소
  + 실행 중인 EC2 인스턴스의 수를 원하는 수준으로 유지 가능
  + 수요가 급증할 경우 인스턴스의 수를 자동으로 증가
  + 수요가 적을 경우 자동으로 용량을 감소시켜 비용 낭비를 최소화

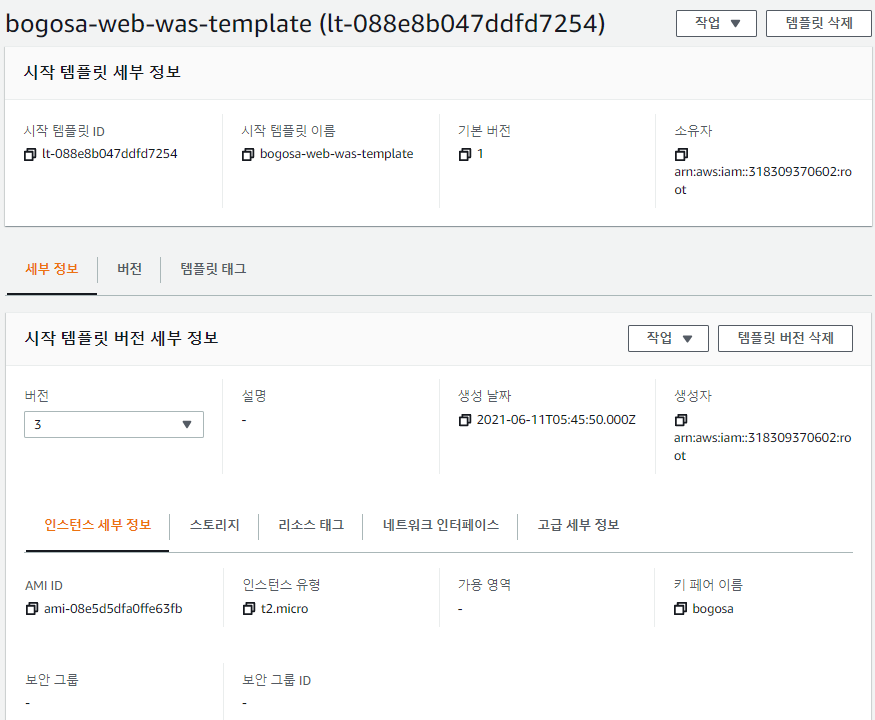
Autoscaling 정책을 적용하여 인스턴스를 유지하려면 다음과 같은 순서로 구성한다.

1. Launch Template 구성

* 미리 만들어 둔 AMI 사용
* 네트워크 구성 - WAS 서브넷 및 보안그룹을 따름
* 사용자 데이터(인스턴스 생성시 실행할 쉘 스크립트)

|  |
| --- |
| **#**! /bin/bash cd /home/ec2-user/opt/apache-tomcat-9.0.0.M22/bin iptables -I INPUT 1 -p tcp --dport 8080 -j ACCEPT ./startup.sh |

* 나머지 설정은 Autoscaling Group 생성 시 설정



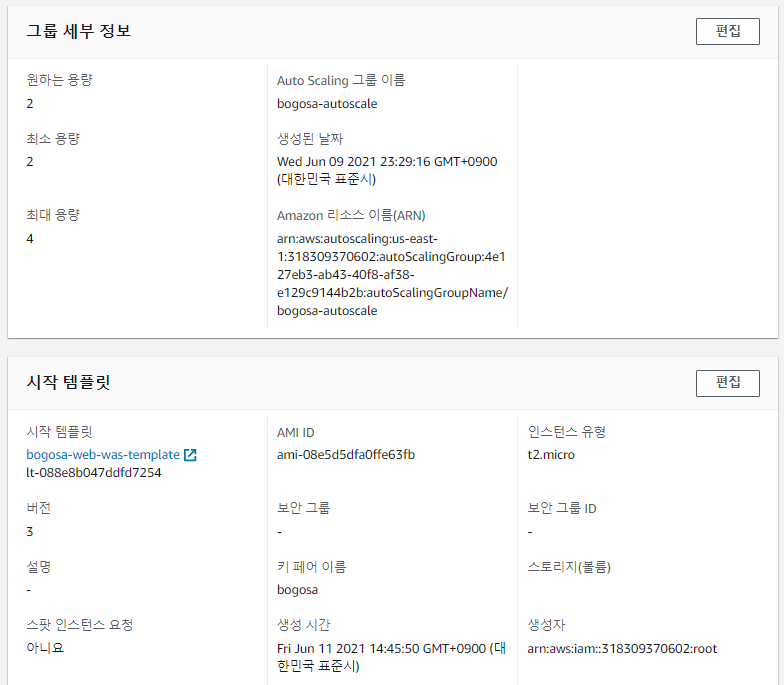
<그림 7-3 Launch Template 세부정보>

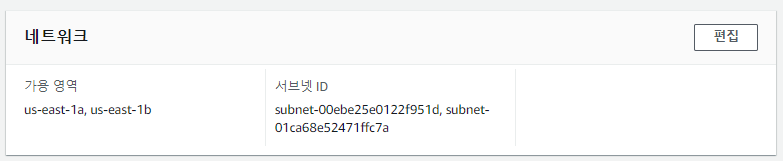
1. AutoScaling Group 구성

* 미리 만들어둔 Launch Template 사용
* 미리 만들어둔 VPC, 서브넷은 각 Available Zone의 Private Subnet
* 원하는 용량 = 최소용량 = 2, 최대용량 4 (용량 = 인스턴스 갯수)

>> 2개의 Available Zone에 적어도 각각 하나이상씩 인스턴스가 배치 됨

* 우선적으로 Load Balancing 없이 구성





<그림 7-4 Auto Scailing group 세부정보>

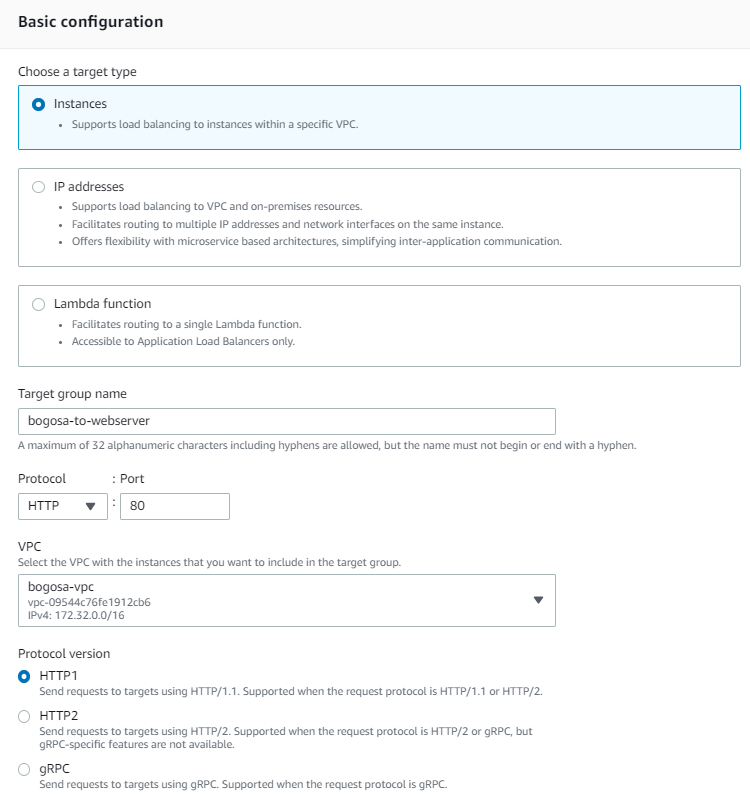
* **Load Balancing**

Load Balancing이란 네트워크 기술의 일종으로 네트워크 트래픽을 하나 이상의 서버나 장비로 분산하기 위해 사용되는 기술로, 로드 밸런싱을 수행하는 소프트웨어나 하드웨어를 로드 밸런서(Load Balancer)라고 한다. 로드 밸런싱 서비스를 통해 외부에서 발생되는 많은 인터넷 트래픽을 여러 웹 서버나 장비로 부하를 분산하여 처리할 수 있다.

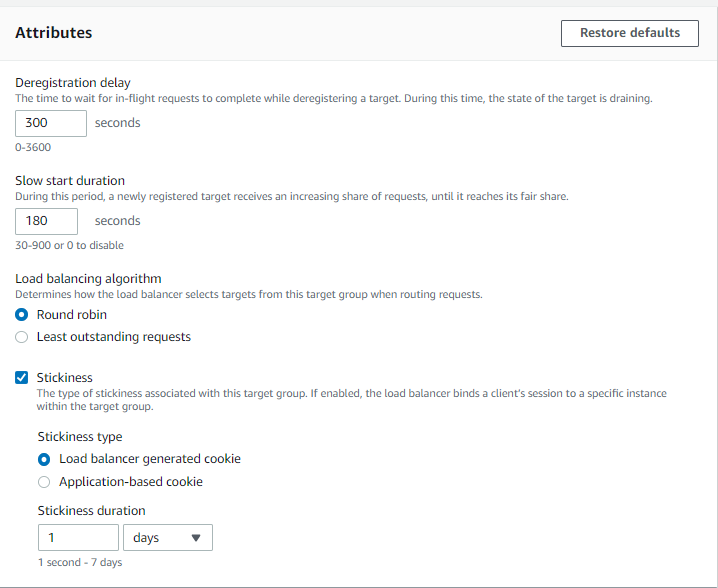
Amazon Elastic Load Balancing은 단일 가용 영역 또는 여러 가용 영역에서 Amazon EC2 인스턴스 및 컨테이너, IP 주소 같은 동일한 서비스를 제공하기 위해 준비된 여러 대상으로 애플리케이션을 자동으로 분산시킨다. 프로젝트에서는 단순 트래픽 분산을 목적으로 Application Load Balancer 서비스를 사용하였으며 Health Check를 통과한 대상으로만 트래픽을 라우팅 함으로써 Failover가 가능하다 즉 로드밸런서에 Autoscaling 그룹을 대상으로한 대상그룹을 등록함으로써 고가용성 아키텍처를 설계할 수 있다.

1. 대상그룹 등록

* Target type : 인스턴스
* Protocol /Protocol Version: HTTP/ HTTP1
* port : 80 (인스턴스의 웹 서버 개방 포트)
* 우선 대상을 지정하지 않고 구성 (Autoscaling Group에서 등록)
* 대상그룹 속성 : 웹 페이지 세션을 유지하기 위해 Sticky Session 사용



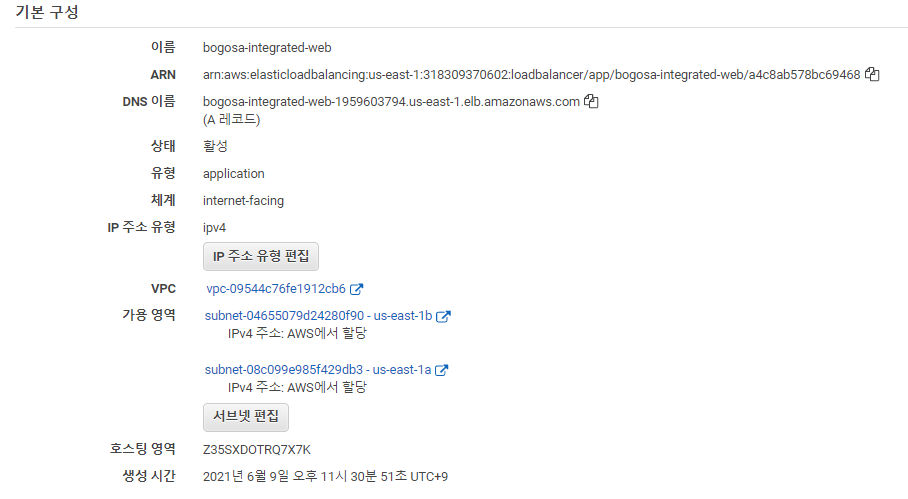
<그림 7-5 대상그룹 Configuration>

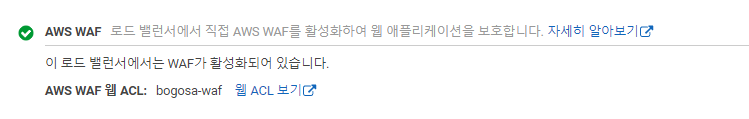
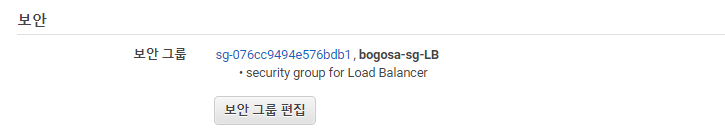


<그림 7-6 대상그룹 Attributes>

1. Load Balancer 구성

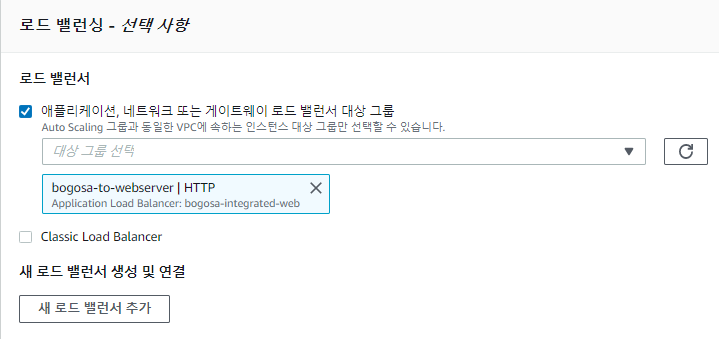
* Application Load Balancer
* Internet-facing
* Autoscaling group과 동일한 Available Zone 선택
* HTTP Listener Port : 80
* 미리 만들어 둔 대상그룹
* WAF 활성화





<그림 7-7 Load Balancer 상세 정보>

1. Autoscaling Group에서 Load Balancer 등록



<그림 7-8 로드밸런서 등록>

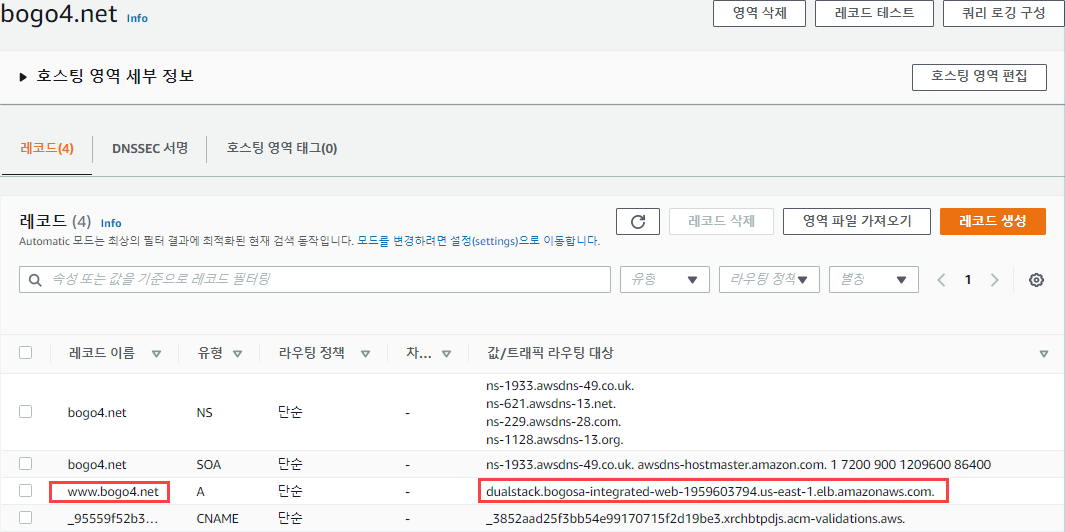
* **Route53**

Amazon Route 53은 가용성과 확장성이 우수한 클라우드 기반의 DNS(Domain Name System) 서비스이다. 이 서비스는 [www.example.com과](about:blank) 같은 이름을 192.0.2.1.과 같이 컴퓨터 간 연결을 위해 사용되는 숫자로 된 IP 주소로 변환 하며, 개발자와 기업은 최종 사용자를 인터넷 애플리케이션에 매우 안정적이며 효율적 비용으로 연결할 수 있다. 또한 사용자의 요청을 Amazon EC2 인스턴스, Elastic Load Balancing, S3 Bucket 등 AWS 에서 실행되는 다양한 인프라에 효과적으로 연결 할 수 있다.

Route53 는 상태 검사(Health Check)와 연결된 장애 조치 레코드를 구성할 수 있다.

상태 검사에서 연결 상태로 정상 상태가 반환되면 응용 프로그램은 계속 정상적으로 작동한다.

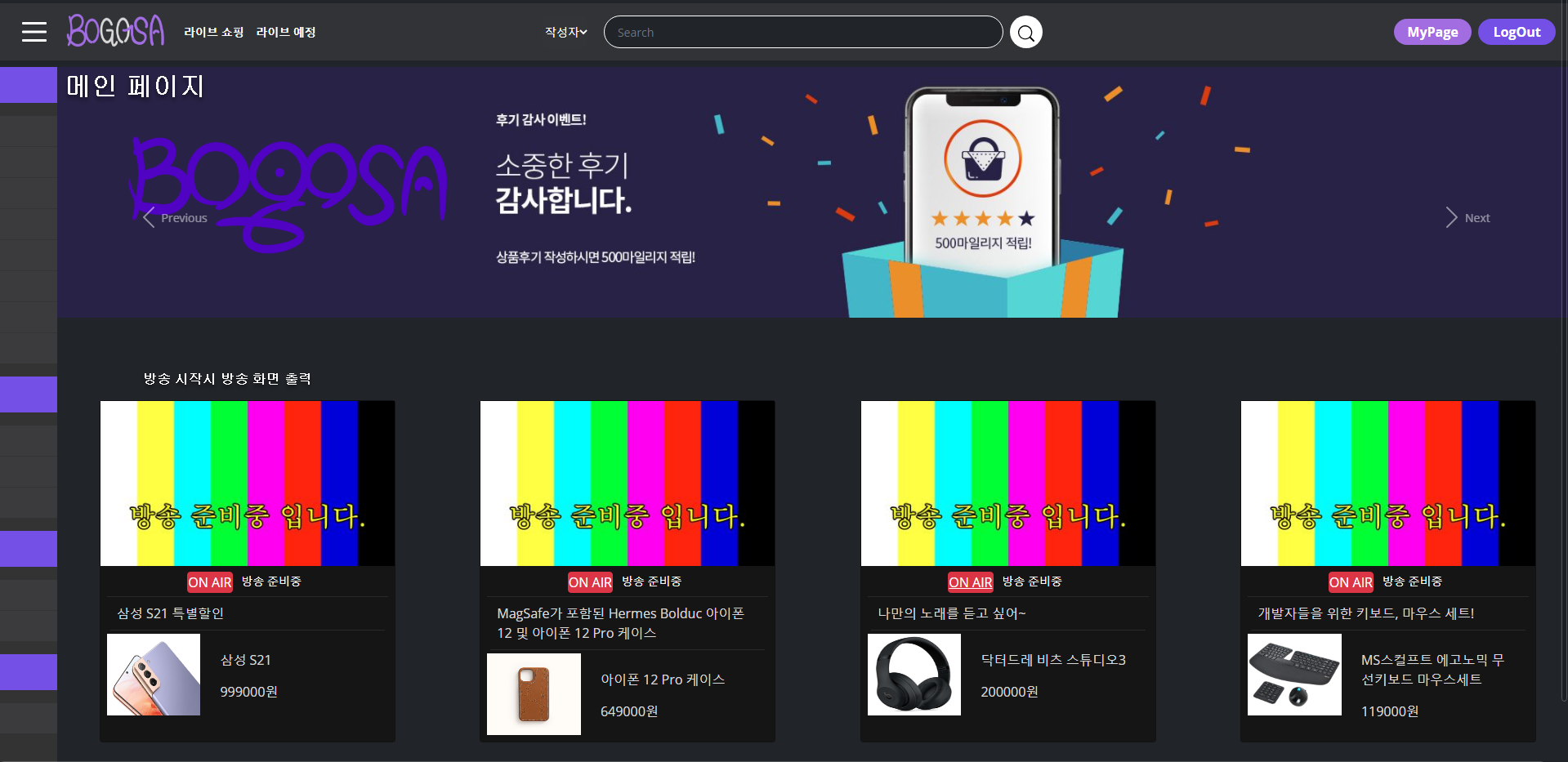
프로젝트에 Route53 에서 도메인을 구매해 적용했다. 웹 기반 서비스로 운영/관리가 쉬우며 다양한 기능과 외부 DDoS 공격을 차단하는 기능을 기본적으로 제공받았다.



<그림 7-9 Route53 레코드>

# **4. Demo Page**

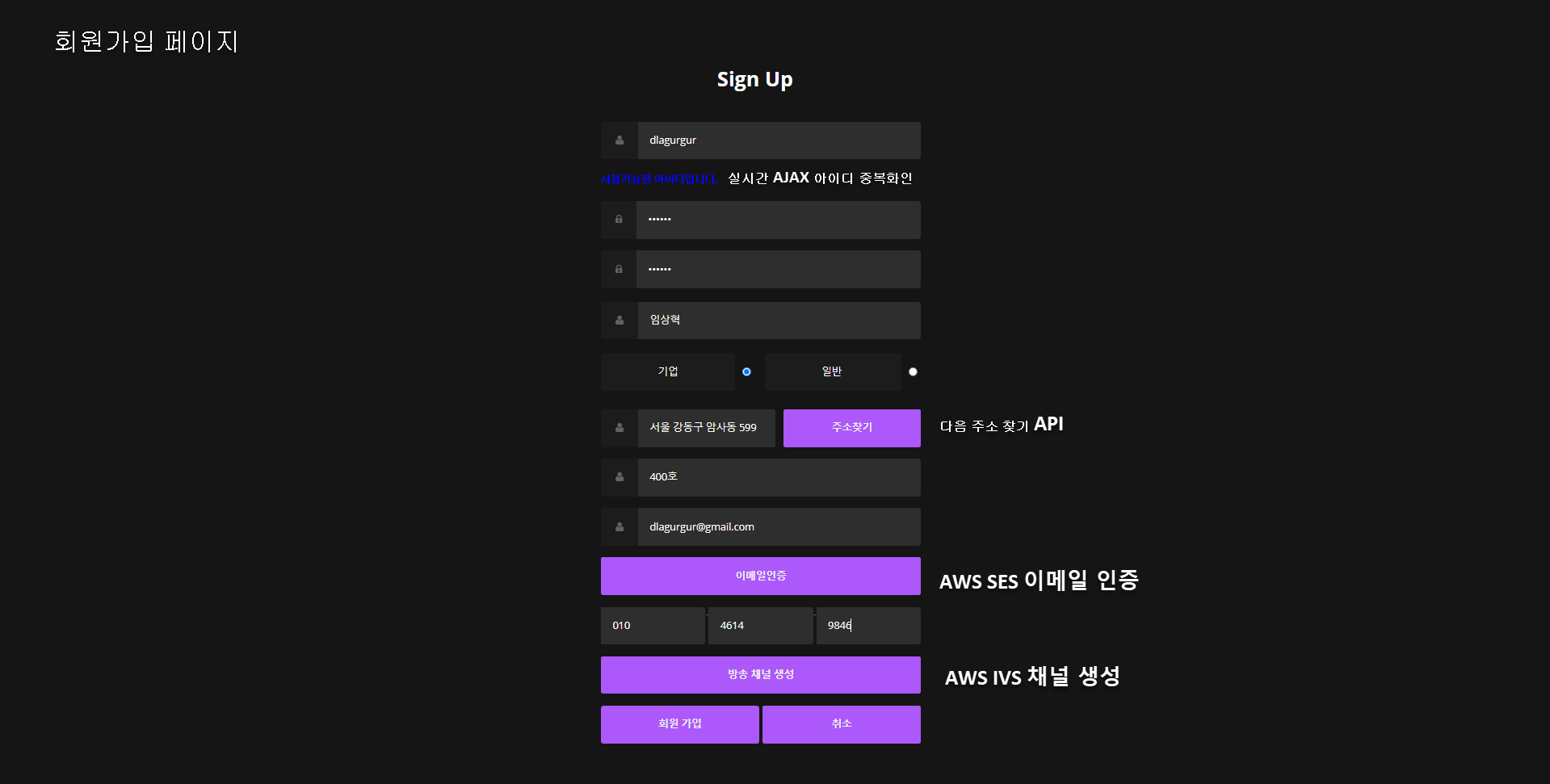
메인 페이지



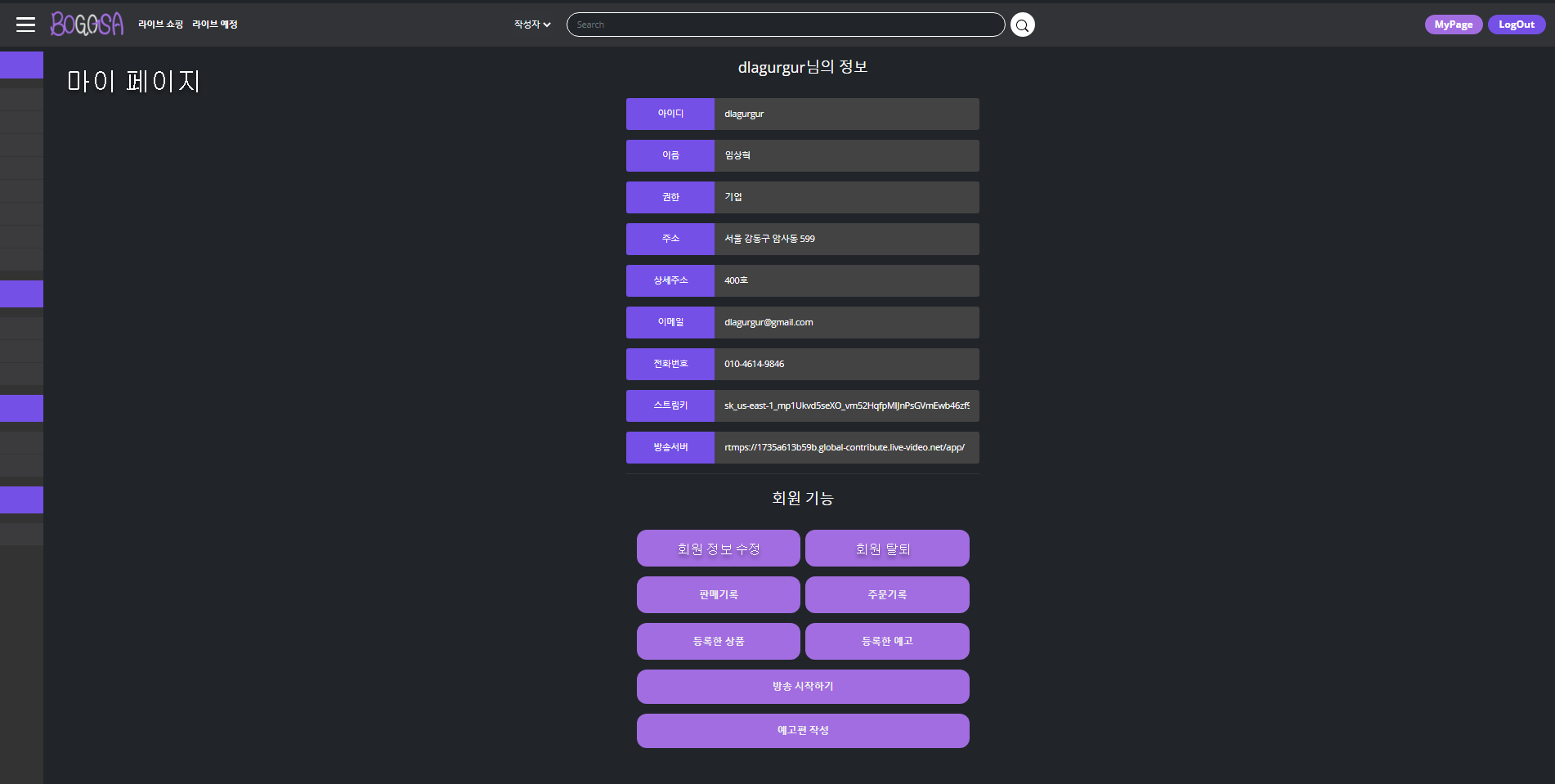
로그인 페이지



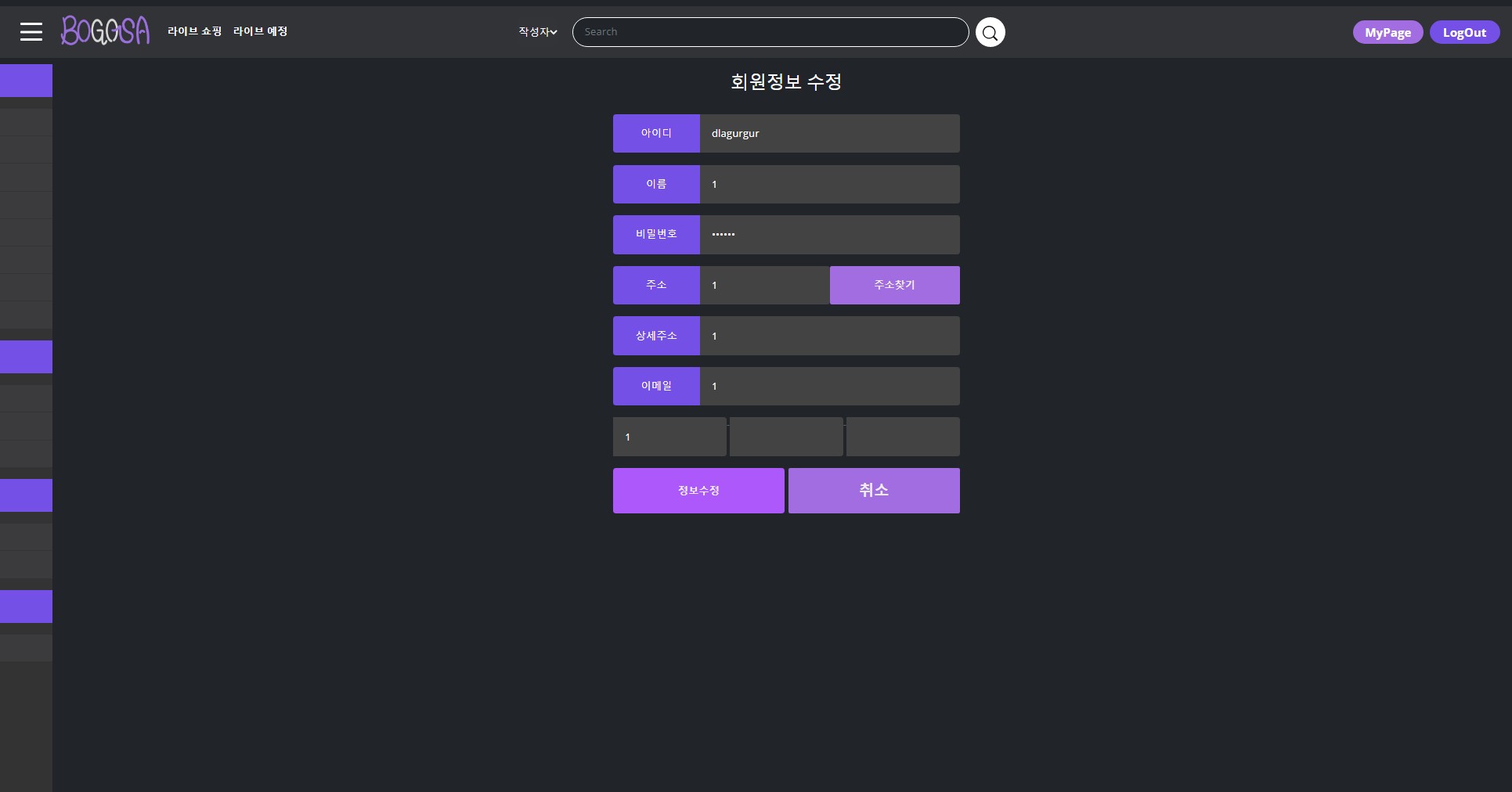
회원가입 페이지



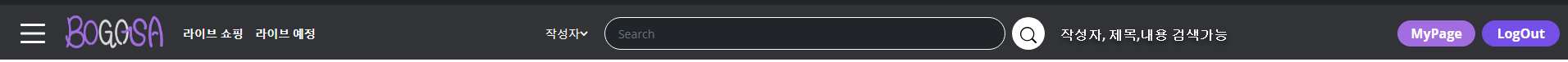
마이 페이지



회원 정보 수정 페이지



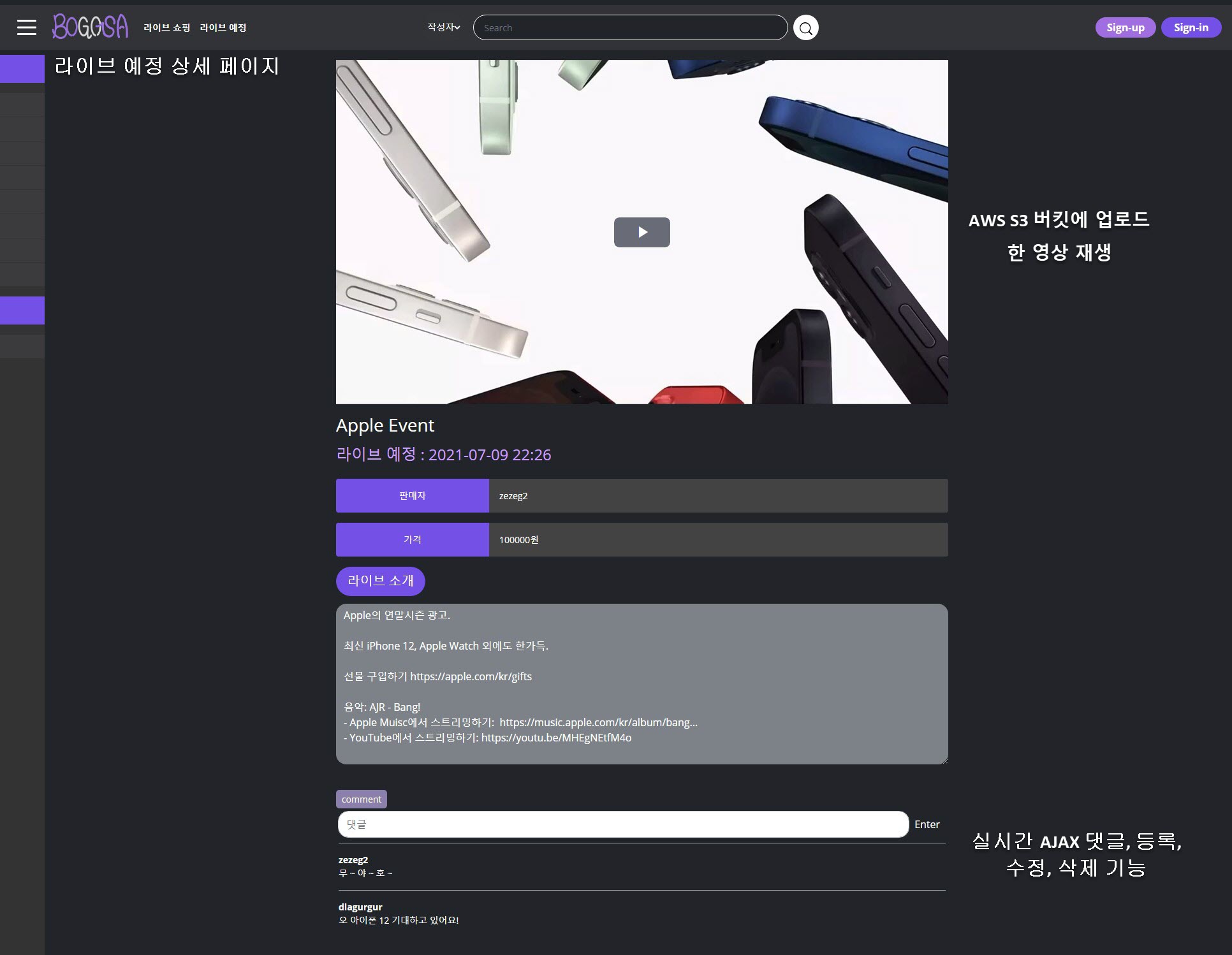
헤더



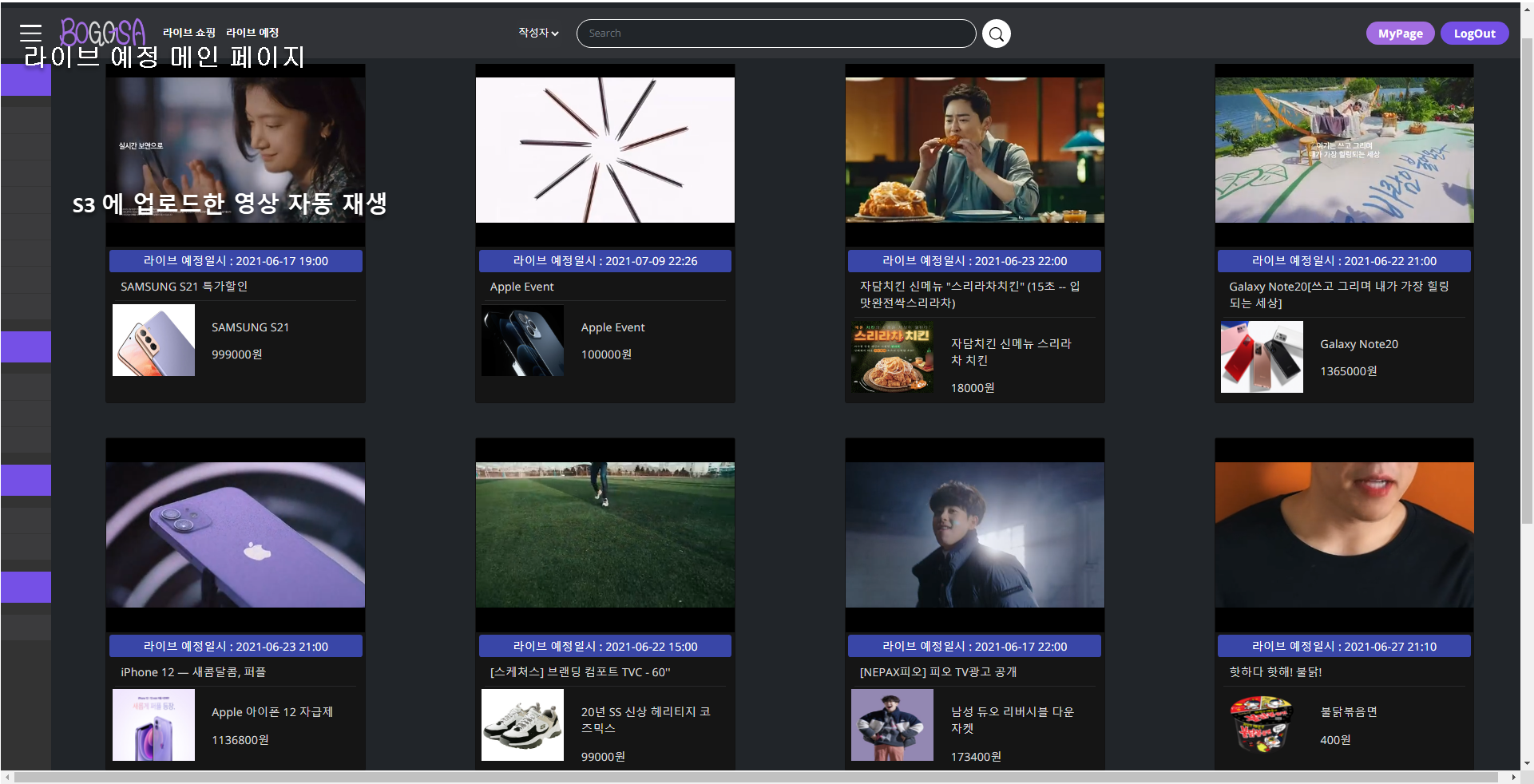
라이브 예정 등록 페이지



라이브 예정 상세 페이지

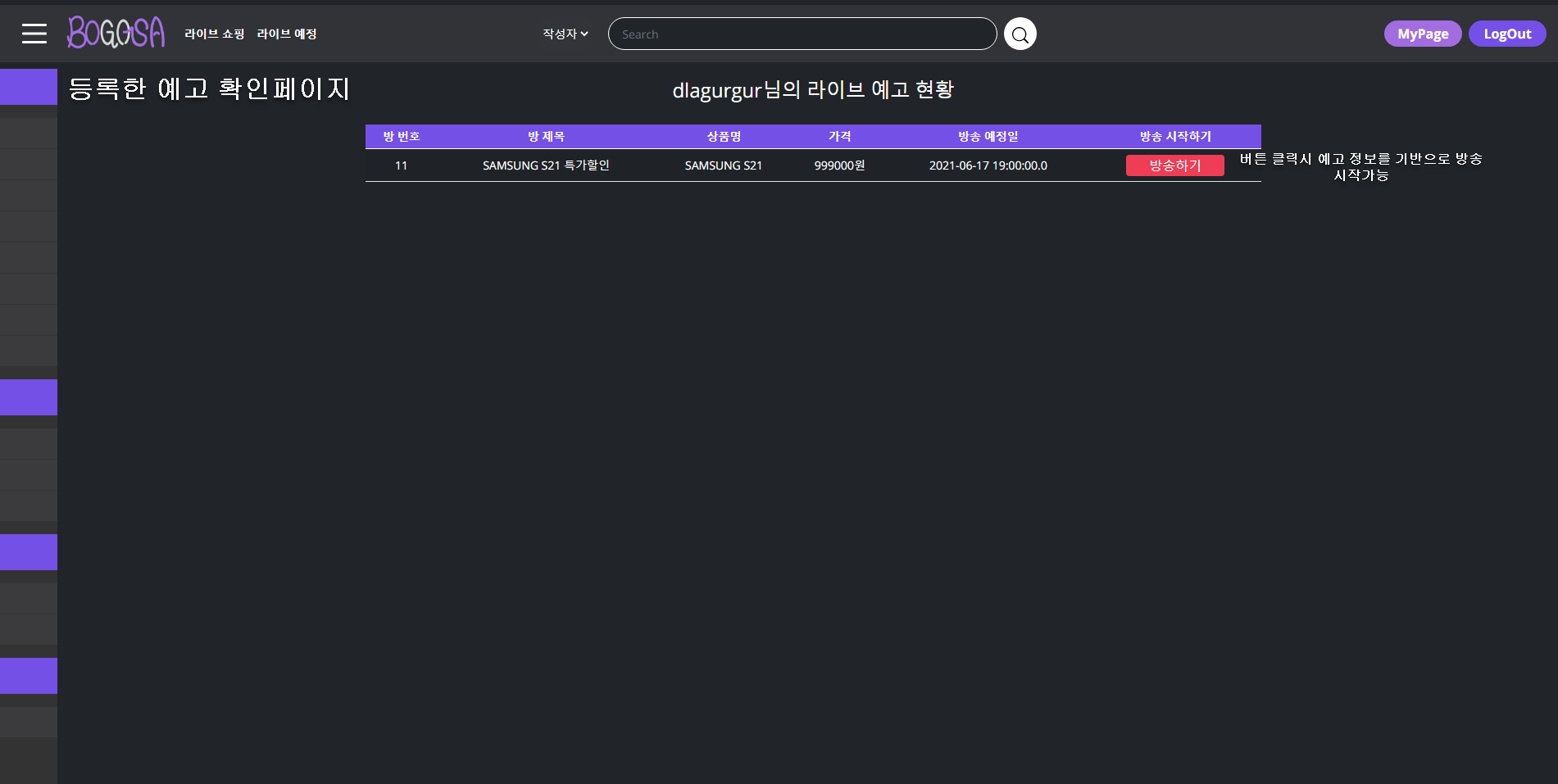


라이브 예정 메인 페이지

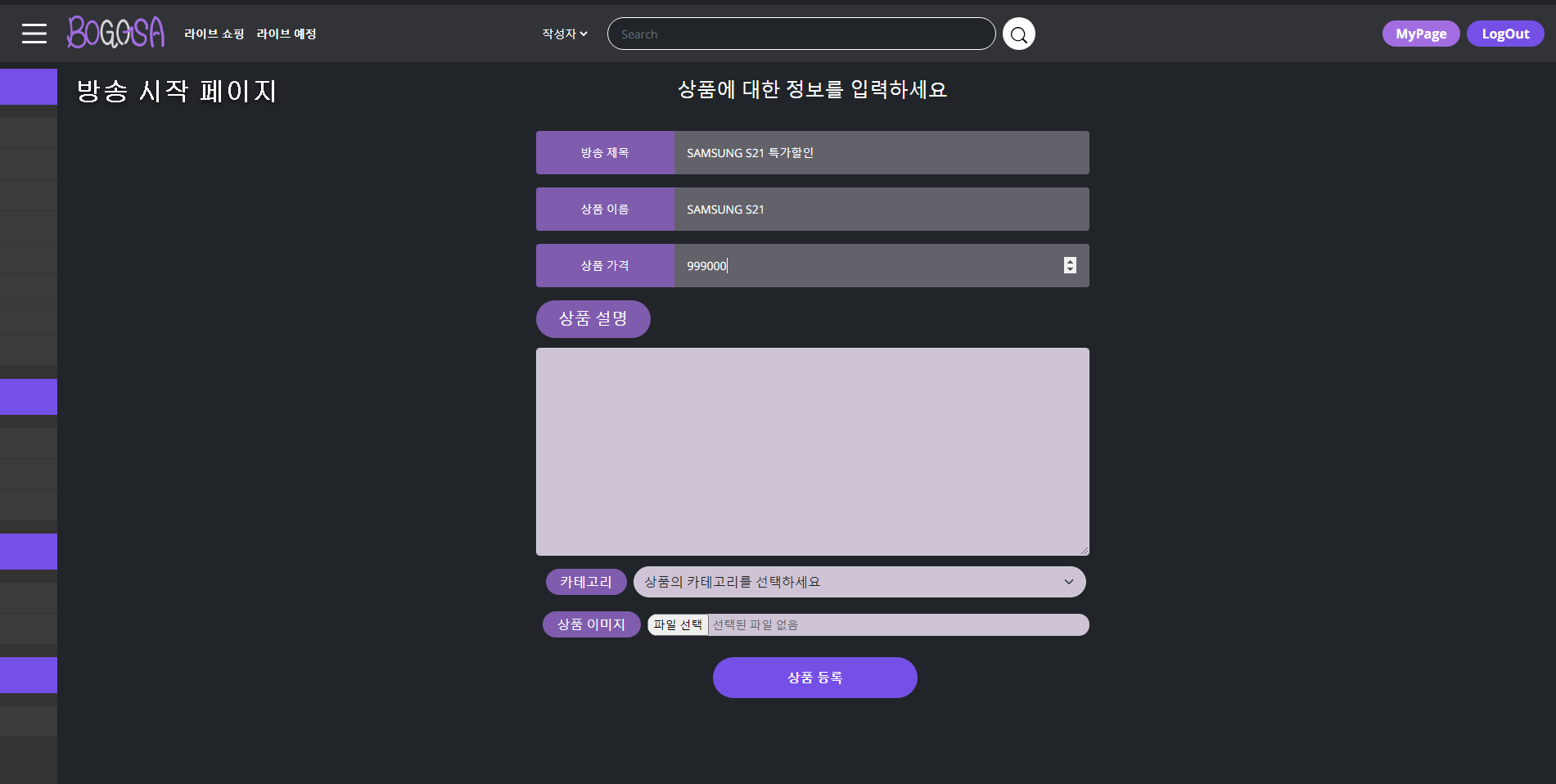


CloudFront에 배포된 영상을 재생

등록한 예고 확인 페이지



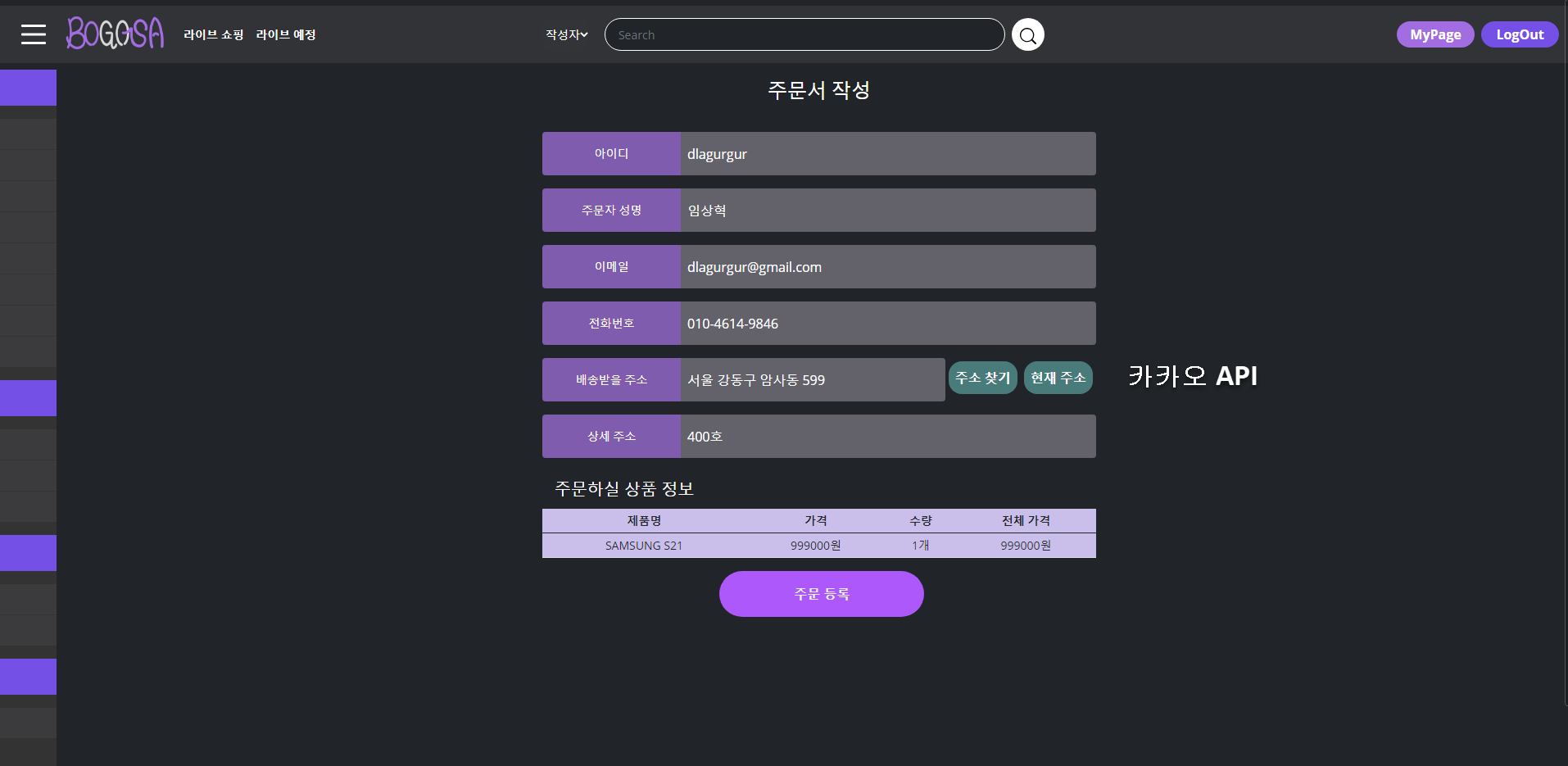
방송 시작 페이지



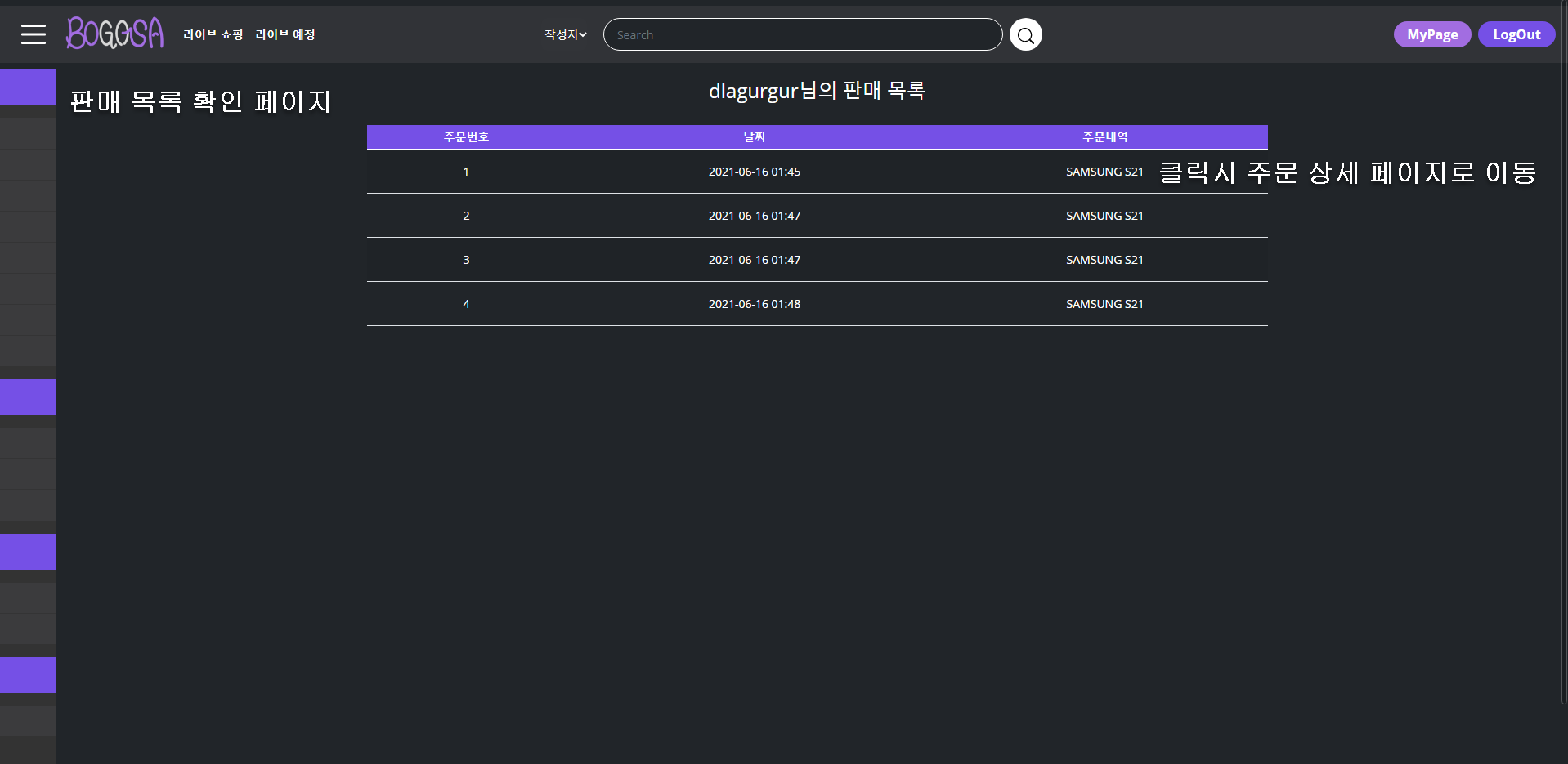
방송 페이지



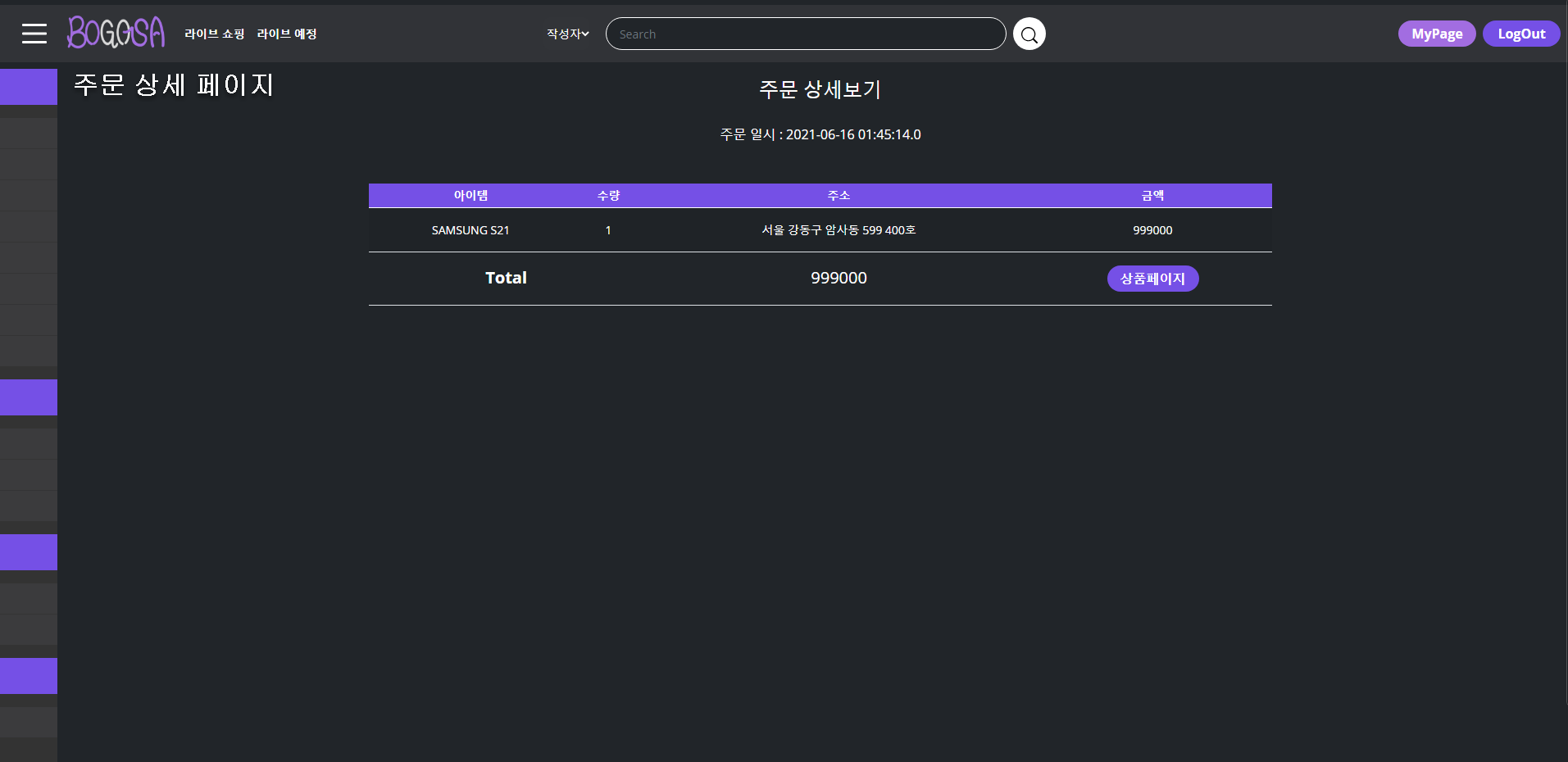
상품 주문 페이지



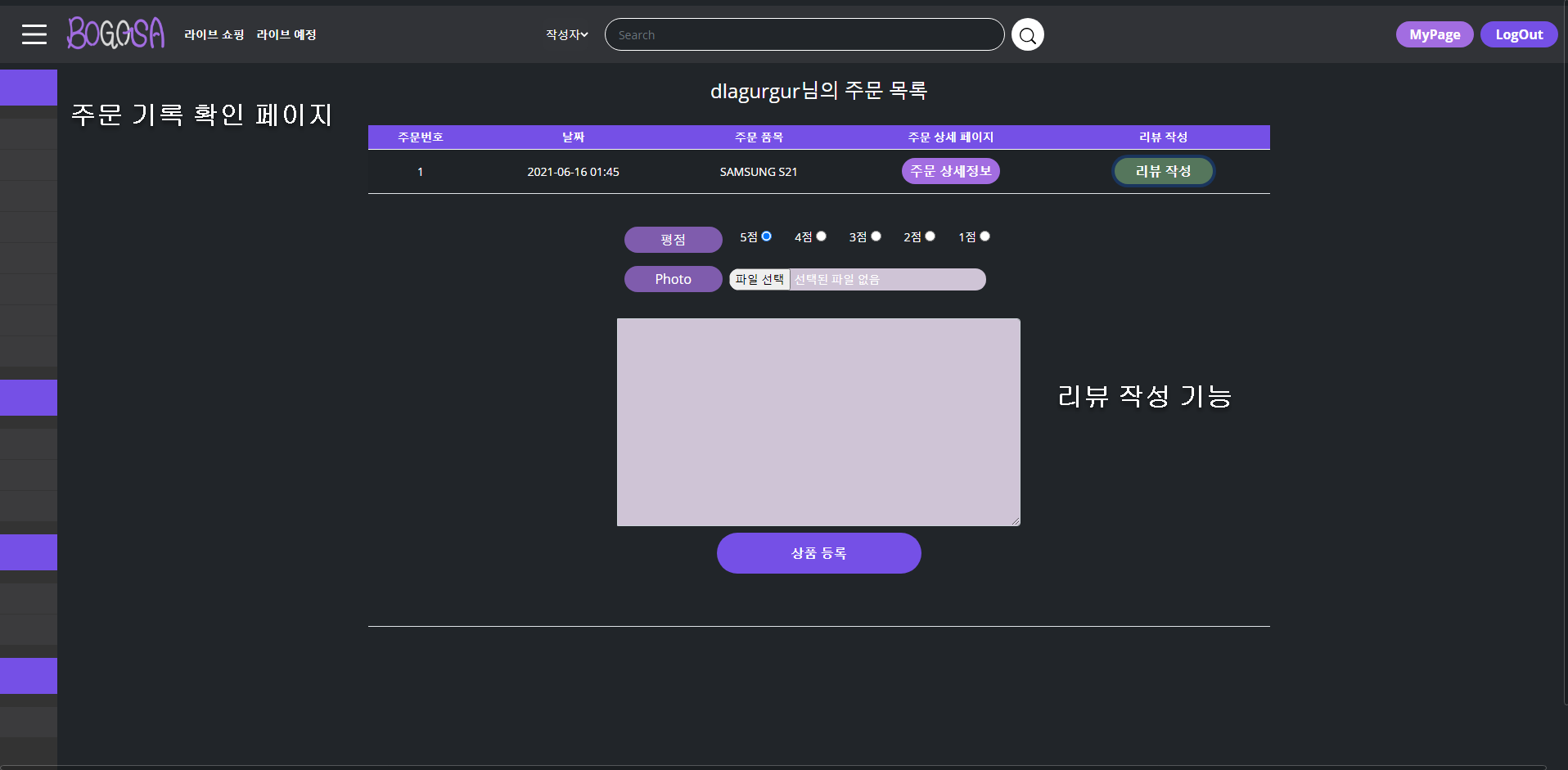
판매기록페이지



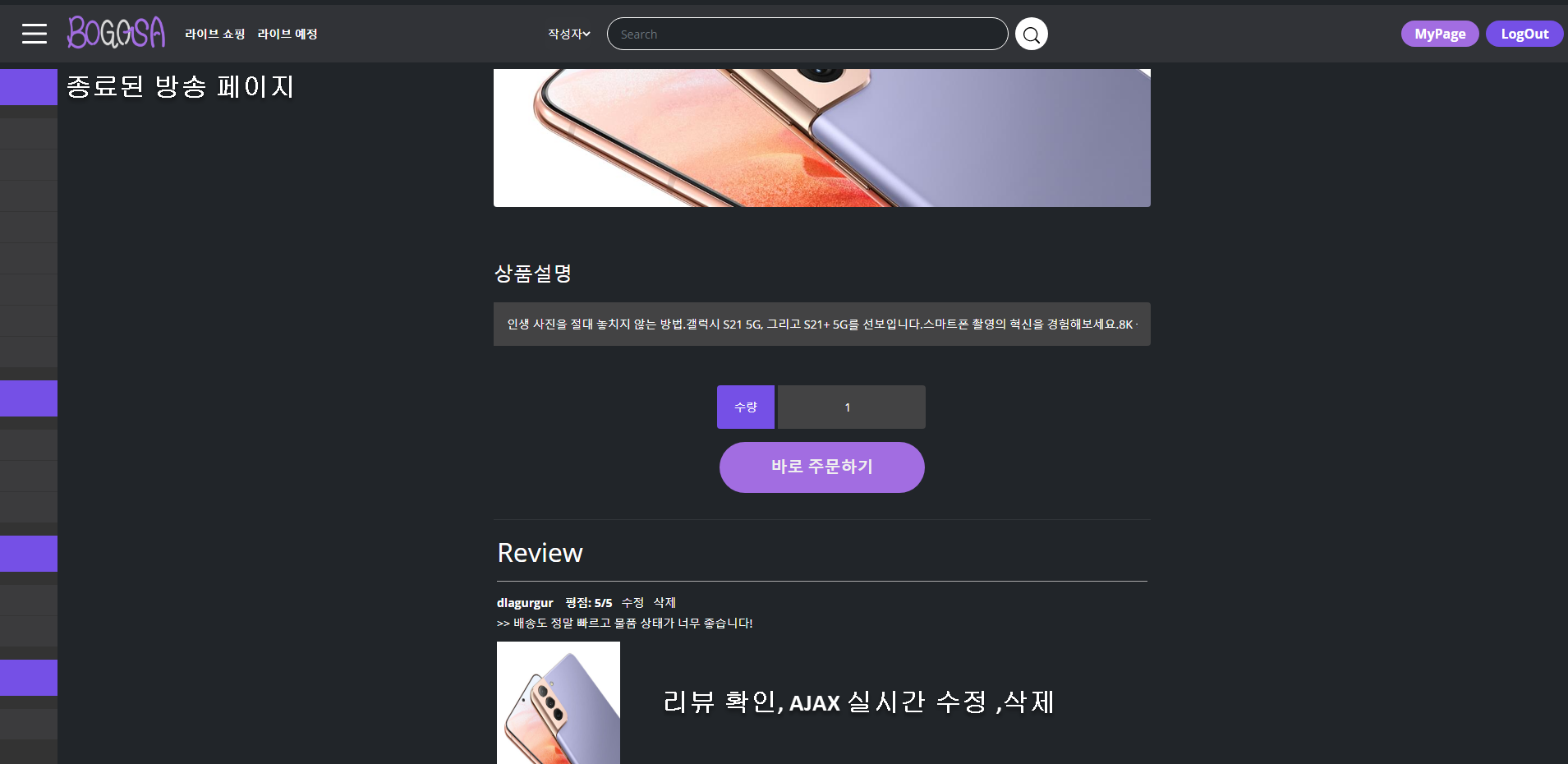
주문 상세 페이지



주문기록 페이지



종료된 방송 페이지



# **5. 보완 할 점**

## 아키텍처 측면

* 설계한 아키텍처대로 웹 애플리케이션을 배포 했을 때 서버 내 미디어파일(이미지)을 불러오지 못하는 현상이 발생하였다. 이는 로드밸런서를 통해 각 서버의 내부 저장소에 미디어 파일이 저장되어 있는 문제인데 이는 인스턴스 공유 저장소를 이용하고 톰캣 서버가 외부 저장소의 파일을 이용하도록 해결할 수 있을것으로 예상된다.
* 현재는 CloudFront에 배포된 주소를 가져오는 방식이 영상파일을 업로드 했을 때(S3\_putObject) 변환작업이 일어나고 그 작업내역이 저장되는 manifest파일을 다운로드 받아(S3-getObject) 파싱하고 필터링 하는 과정을 거치게 된다. 이 방식이 너무 복잡하며 비효율적이라 생각된다. 또한 S3 에 파일이 업데이트 되기 전까지 반복적으로 다운로드 받는 로직으로 인해 자원낭비, 또는 캐시 데이터를 이용함으로써 문제가 발생하였다.(업데이트 되지 않은 파일을 지속적으로 다운로드함으로써 반복문 루프) 이를 해결하기 위해 람다 함수의 기능적, 아키텍처 구조적인 변화가 필요할 것 같다. 웹페이지 내에서 응답을 기다리기 보다는 변환이 완료되었을때 이벤트를 Trigger로 하여 RDS에 직접적으로 쿼리문을 실행하도록 하여 데이터를 저장하는 방식으로 해결할 수 있을 것으로 예상된다.
* 회원가입 하는 유저를 DBMS에 저장하여 관리하는 방식은 각 회원이 웹서비스 내 서비스를 이용할 때 AWS 서비스 및 리소스에 접근하는 권한을 필요 이상으로 가지게 되며 데이터 베이스에 AWS 리소스 값을 저장해야한다. 회원 관리 방식을 관리형으로 전환(Cognito User pool)하여 권한을 부여하고 웹 애플리케이션의 보안적 측면을 향상 해야 할 것 이다.

## 웹 어플리케이션 측면

* 도메인에 ACM(Amazon Certificate Manager)에서 발급 받은 인증서를 등록함으로써 SSL 인증서를 이용한 HTTPS 보안 접속 방식으로 웹 애플리케이션을 배포하려고 했으나, 웹 애플리케이션이 이용하는 API, script등이 http 방식으로 다른 서버와 통신하여 HTTPS로 접속하게 되면 서비스를 정상적으로 운영할 수 없었다. 보안접속을 가능하도록 하기 위해서 다른 서버와 통신하는 방식도 보안접속을 하도록 애플리케이션을 수정 해야 할 것이다.
* 현재 실시간 스트리밍 서비스와 더불어 가장 중요한 요소 중 하나인 실시간 방송에서의 채팅서비스가 현재는 AJAX 를 이용하는 방식으로 설계되어 있다. 사실 이 방식은 지속적으로 페이지가 Refresh 되어야 하기 때문에 실시간 채팅을 구현하기에는 적합하지 않은 방식이다. 이 방식 대신에 웹소켓(Web Socket) 방식과 쓰레드를 활용한 방식으로 라이브 방송방(IVS Play URL + 웹소켓 채팅)을 구현하는것이 적합하다고 판단된다.

# **6. 향후 프로젝트 확대 방향**

이 프로젝트는 처음에 설정했던 모든 요구사항을 만족하지는 못했다. 우선적으로 라이브커머스 라는 서비스에서 핵심적이고 기본적인 기능을 구현하기 위해 라이브 방송을 진행하고 그 페이지에서 상품을 구매할 수 있도록 구현하는것 까지가 프로젝트 기간동안의 과제였다. 우선 앞으로 이 프로젝트는 더욱 일반적인 모습의 라이브커머스 서비스의 형태를 갖추고 보안적인 요소를 강화해야 한다.

## 아키텍처 측면

* HTTPS 보안접속 : 보완해야 할 점에서 언급했지만, 실제 서비스가 가능하도록 하기 위해서 인증서 기반의 HTTPS 보안접속이 가능하도록 할 것
* 효율적인 스케일링 전략 : 서비스 이용자 수 증가 대비
* 핀옵스를 고려한 설계 : 기간을 고려하여 완전 관리형 서비스를 사용했는데, 더욱 저렴한 설계가 가능하도록 할 것

## 서비스 측면

* 장바구니 기능 : 특정 라이브 방송방에 접속해 있지 않더라도 장바구니에 상품을 담아두고 다른 라이브 방송을 시청하는 등 웹서비스를 이용하며 자유롭게 구매할 수 있도록 장바구니 기능을 구현 할 것
* 다시보기 기능 : 라이브 방송이 끝나더라도 서비스 이용자의 구매를 활성화 하기 위해서 IVS 라이브 영상 S3 저장 기능을 이용해서 다시보기 기능을 구현하여 서비스 연속성을 보장하도록 할 것
* 타 플랫폼의 인증기반으로 회원가입 구현 : 서비스 접근성을 강화하기 위해 타 플랫폼의 계정으로 간편가입이 가능하도록 할 것

# **7. 마무리**

**박종현**

엔코아 플레이데이터 언택트 비즈니스를 위한 멀티클라우드 아키텍트 양성과정을 최종 프로젝트와 함께 마무리하게 되었다. 프로젝트 주제를 정하는 순간부터 난관의 연속이었다. 클라우드 와 인프라, 자동화에 대해 학습했지만, 이를 보여줄 수 있는 프로젝트는 어떤 모습인지 짐작조차 되지 않았다. 그래서 팀원들과 함께 사업을 한다면 어떤 서비스를 제공하고 싶은지에 대해서 의논했고 결정된 주제에 대해서 배웠던 기술들을 최대한 접목하고자 하였다. 비록 라이브커머스 서비스를 구현하는 프로잭트에서 수업과정에서 다룬 모든 내용을 녹여낼 수는 없었지만 새로운 온라인 쇼핑몰로 떠오르는 마케팅 방식을 구현하기 위해 팀원들과 웹 애플리케이션을 개발하고 아키텍처를 구성하는 과정에서 다양한 기술스택을 접하고 실제로 사용해보게 되었다. 짧게나마 스프링 웹프레임워크를 배워 웹 애플리케이션 제작하는 과정에서 AWS 서비스와 연계하는 역할을 맡았고 웹페이지 디자인과 CDN아키텍처 구성, IVS서비스 활용부분을 책임졌으며. 고가용성 아키텍처 설계 및 배포과정까지 팀원들과 함께 협력했다. 이를 통해 경험을 최대로 활용할 수 있고 모르는 기술에 대해서는 조사하고 적용시킬 수 있는 능동적인 개발자 및 엔지니어로써의 자세를 배운것이 아닐까 생각한다. 주제가 정해지고 나서 프로젝트를 일정대로 진행하기 위해서 각자 책임을 다했던 경험은 앞으로 다른 커뮤니티에서도 믿을수 있고 능력있는 구성원으로 발전하기 위한 초석이 될 수 있을것이라 생각한다.

**심재혁**

라이브 커머스에 관심을 가지고 있던 중에 프로젝트 주제로 팀원들에게 제안을 했다. 팀원들도 흔쾌히 받아들였고 라이브 커머스를 주제로 프로젝트를 시작했다. 라이브 커머스에 대해 조사를 하면 할수록 성장 가능성 있는 시장이었고, 이런한 주제로 프로젝트를 진행해 본다는 경험이 좋은 경험이 될 것 같다는 생각이 들었다. 웹 배포 쪽을 맡아 교육 과정에서 배운 것들을 다시 한번 복습 하며 진행했다. 다른 팀원들이 시간도 많이 들고 힘든 개발을 맡아주어 상대적으로 역할이 적은 나는 맡은 역할이라도 제대로 성공해야겠다는 마음 가짐으로 프로젝트에 임했다. 프로젝트를 진행하며 고민하고 정보를 찾아보는 시간들 겪으며 많은 것을 배웠다고 생각한다. 마지막까지 무사히 마친 팀원들의 노고에 감사한다.

**임상혁**

멀티 클라우드 아키텍트 양성과정 파이널 프로젝트를 시작하고 주제를 정하는 과정에서 라이브 커머스를 만들자고 정해졌다.

수업에서는 Python 과 Django를 배웠지만 해당 기술들로 라이브 커머스 서비스를 만든다는 건 힘들다는 판단하에 JAVA, Spring 을 쓰게 되었다.

하지만 이번 멀티 클라우드 아키텍트 양성과정 특성상 JAVA, Spring을 배우지 않았기 때문에

경험이 있는 내가 혼자서 Full Stack 개발을 하게 되었을 때는 오랜만에 하는 개발 이라 과연 혼자서 할 수 있을까? 라는 생각이 들었다.

초반에는 작업량이 많고 힘들었지만 팀원들이 라이브 커머스의 핵심인 IVS(스트리밍 서버), S3등 웹에 적용시킬 AWS API, CSS, 배포 작업을 해줘서 나는 열심히 개발에만 집중할 수 있게 되어 지금의 결과물이 나온 거라고 생각한다.

이번 프로젝트로 통해 AWS 와 Web을 연동하는 것도 알게 되었고 서버를 직접 만들지 않고 AWS를 통해 서버 환경 구축을 할 수 있다는 것을 배워 좋은 공부가 된거 같다.

마지막으로 오랜만에 하는 프로젝트지만 좋은 팀원들을 만나서 같이 즐겁게 협업하면서 좋은 결과가 나와서 정말 기쁘고 고생해준 팀원들에게 감사하다.