

클라우드 컴퓨팅 3차 과제

인터넷 서비스 구현 및 시연

IT경영 2014316016

IT경영 2016314015

설동민

박재연

Docker 기반
이미지

서버 환경의
서비스

클라우드 컴퓨팅 3차 과제

인터넷 서비스 구현 및 시연

01

구현한 인터넷
서비스 명칭

02

서비스 제공
아키텍처
다이어그램 및
사용한 AWS
서비스 소개

03

서비스 내용 및
AWS 서비스
연동 시나리오

04

추진 일정 계획과
진행 실적

05

발견된 문제점
및 해결 방안

06

미해결 문제점 및
예상 해결 방안

07

조별 과제 진행
중에 얻은 교훈

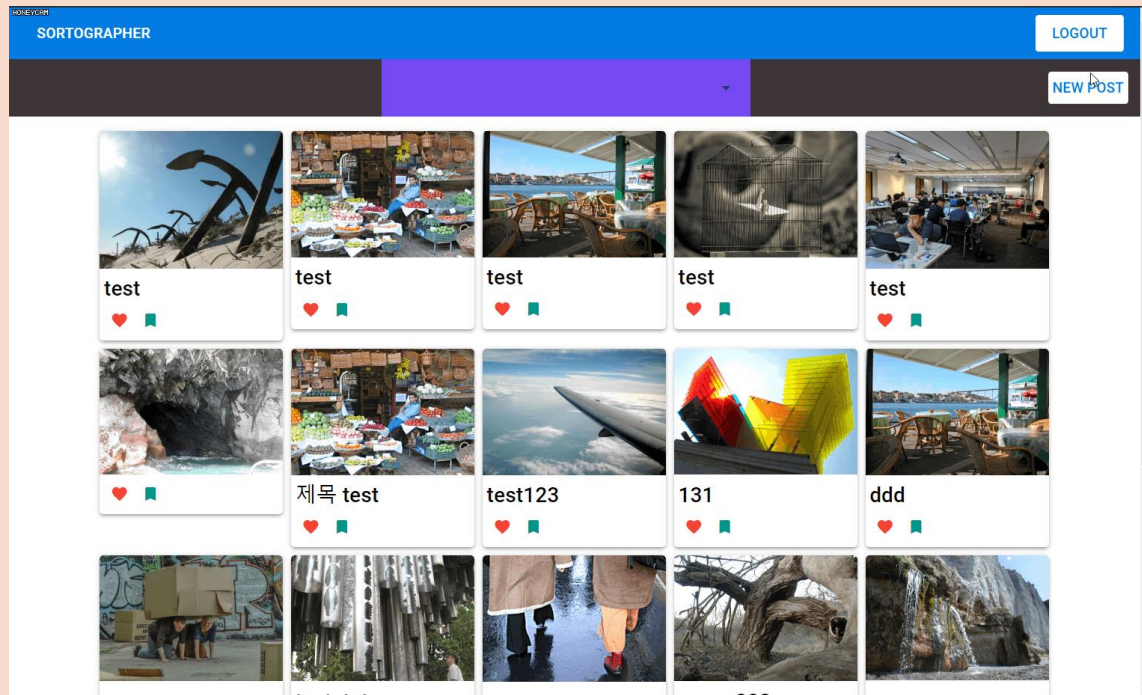
08

기타 과제
관련 내용

01 구현한 인터넷 서비스 명칭

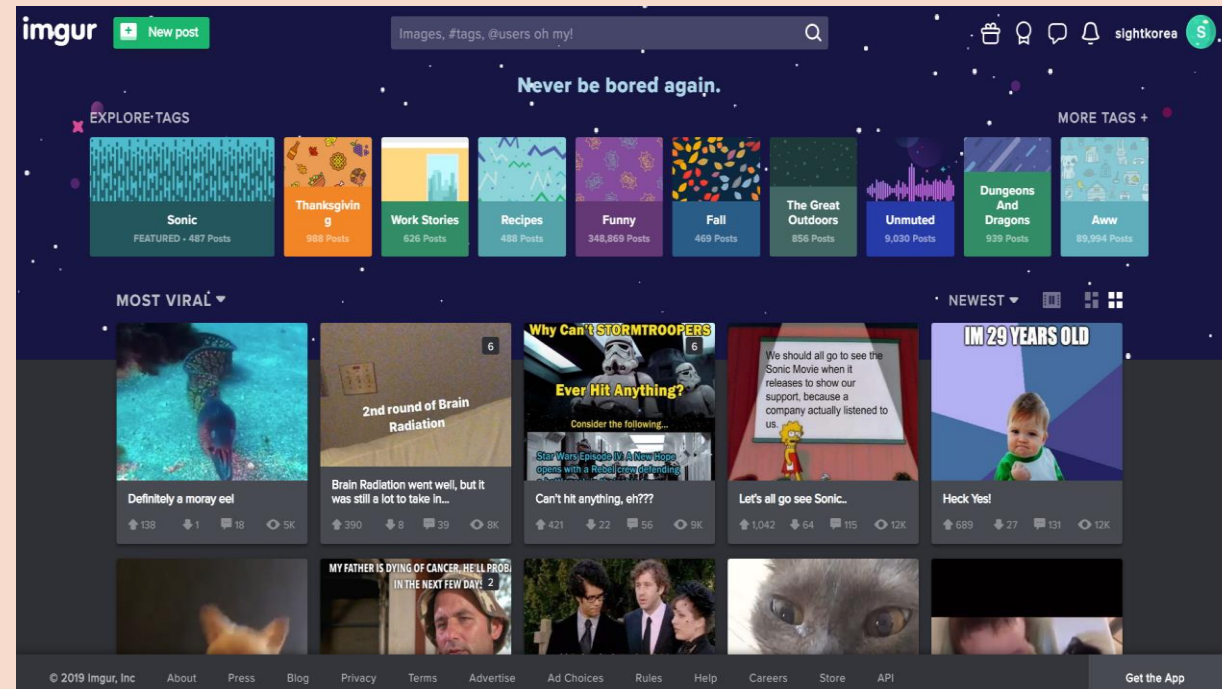
“Sortographer – 이미지 분류 사진첩”

AWS에서 제공하는 딥러닝 기반 모델을 활용하여 사용자가 이미지를 업로드하면,
입력 받은 이미지를 분류하여 라벨링하는 **이미지 호스팅 웹 서비스**



AWS Rekognition

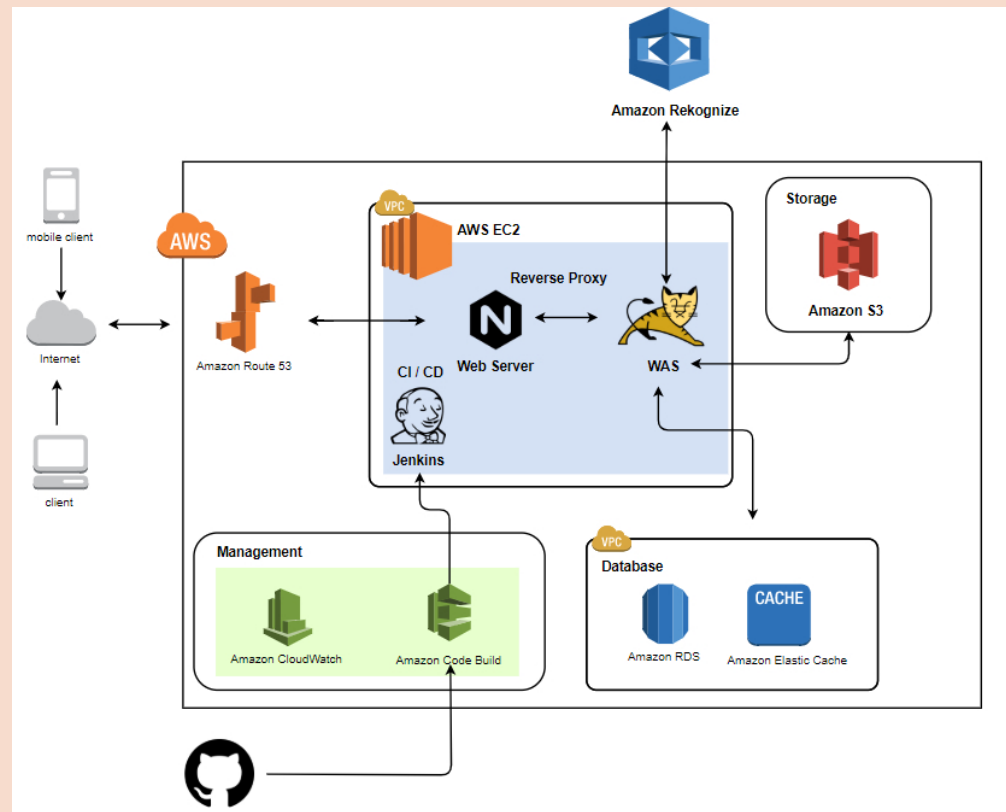
아마존에서 제공하는 영상/이미지 분류 서비스



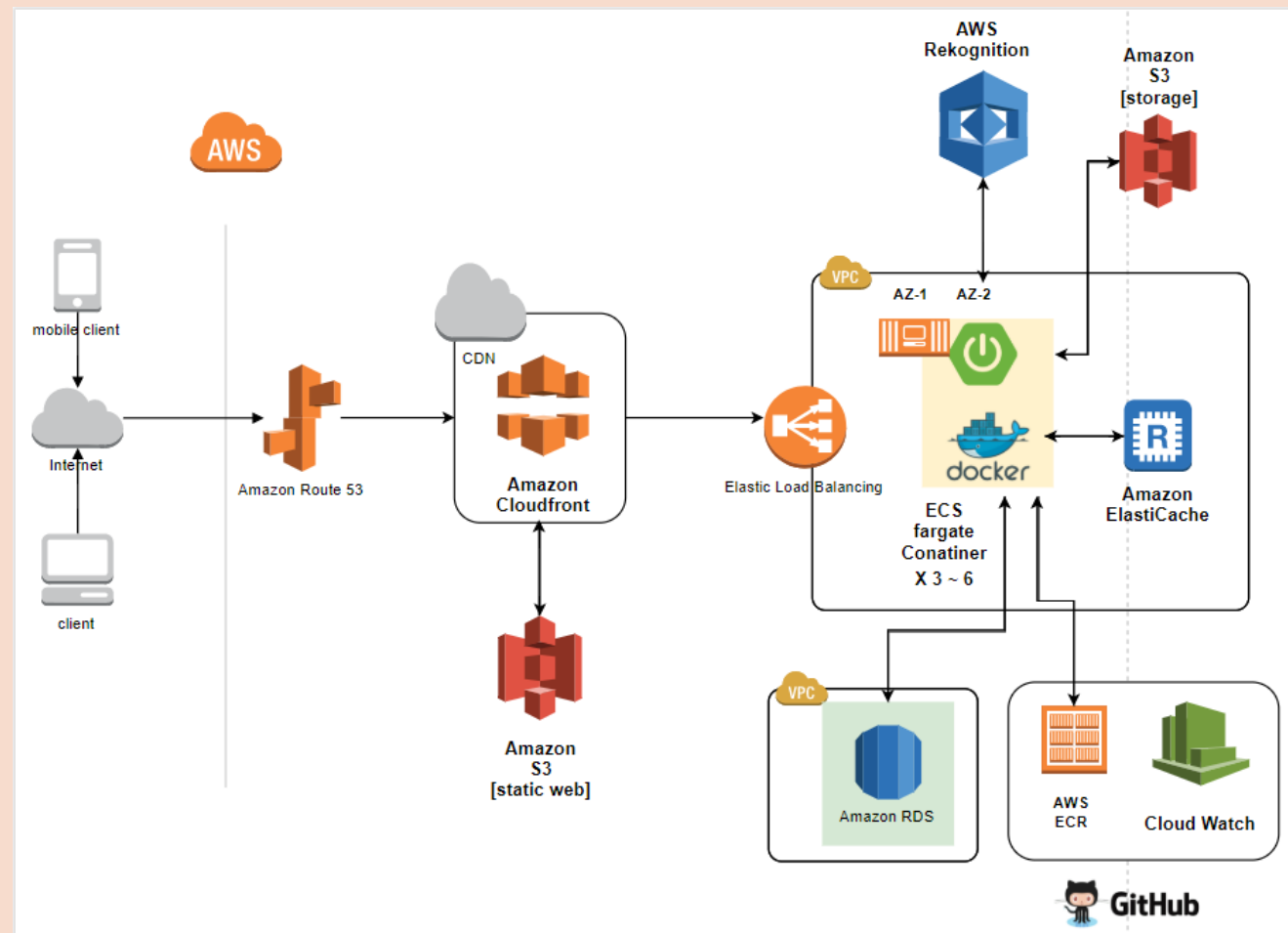
imgur.com

02 서비스 제공 아키텍처 다이어그램

기존 설계



변경 후 설계



02 사용한 AWS 서비스 소개

사용 AWS

- Amazon Rekognition
- Amazon Route 53
- Amazon Certificate Manager
- Amazon CloudFront
- Amazon S3
- Amazon RDS
- Amazon ElastiCache
- Amazon CloudWatch
- Amazon VPC
- Amazon ECS Fargate
- Amazon ECR
- Amazon ELB

설명

- 딥 러닝 기반 시각 분석 서비스
- DNS 서비스
- 도메인 인증
- CDN
- 스토리지 & 정적 웹호스팅
- 관계형 DB (MySQL)
- In-Memory DB (Redis)
- 서비스 모니터링
- 가상 사설 클라우드
- 컨테이너 오케스트레이션
- AWS 도커 이미지 사설 저장소
- 로드밸런서

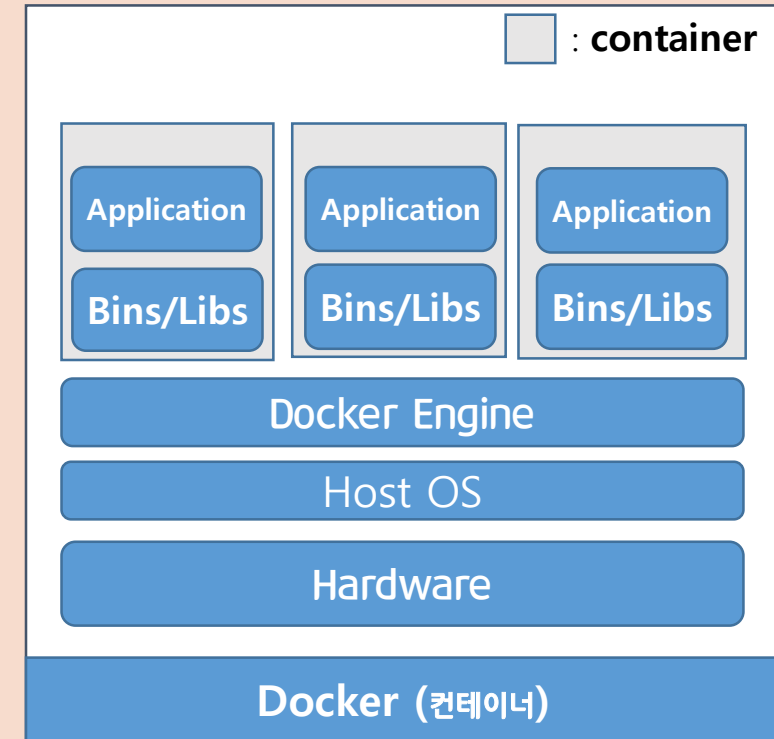
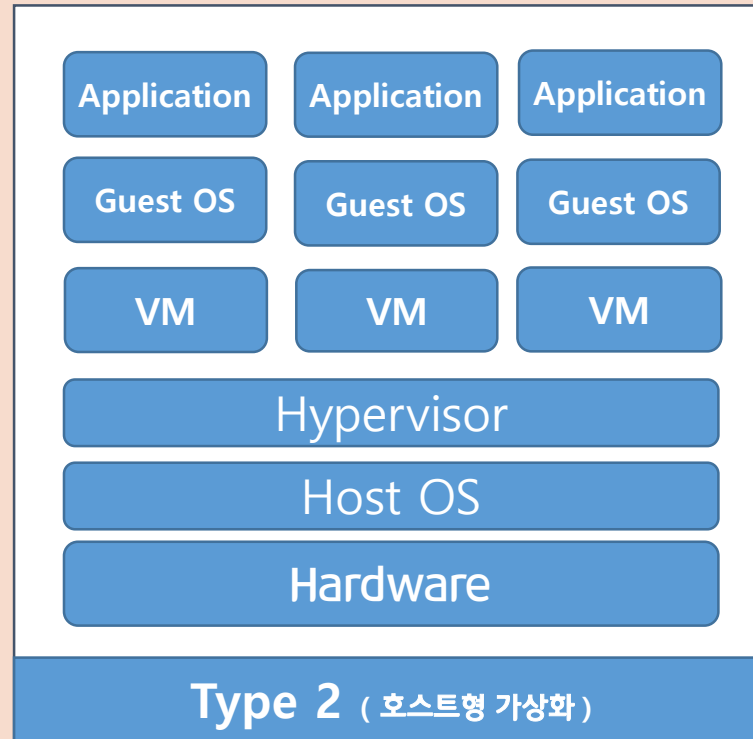
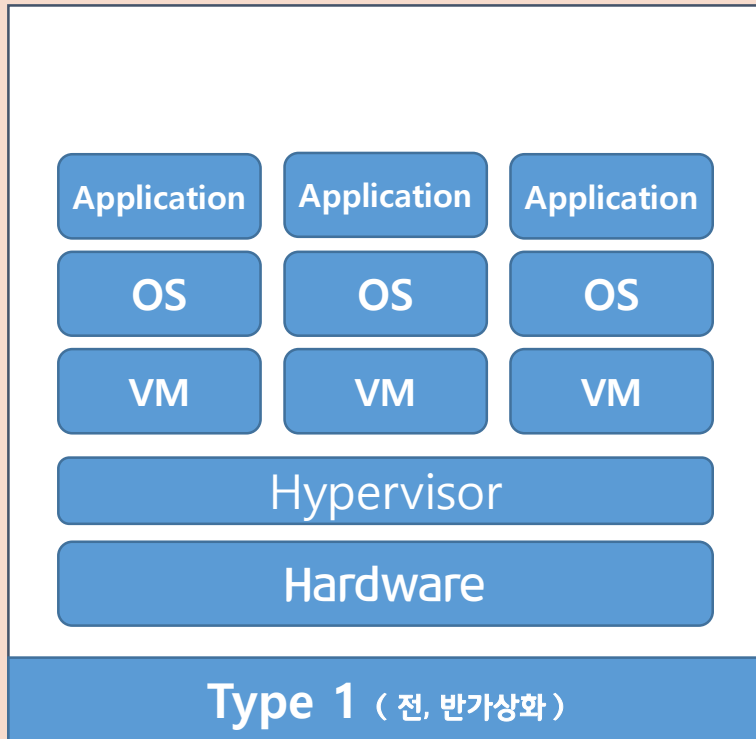
02 사용한 AWS 서비스 소개

“AWS Elastic Container Service [ECS]”

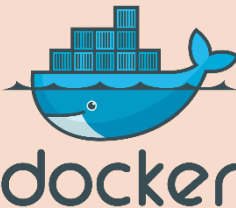
- 선행 지식 : 가상화 기술과와 도커

“같은 서버(application)를 여러 개로 돌리고 싶다.”

“하나의 자원을 쪼개서 효율적으로 쓰고 싶다”



“도커(container) 기술은 적은 비용으로 Host OS 상에서 프로세스 처럼 격리된 공간을 만들 수 있다.”



02 사용한 AWS 서비스 소개

“AWS Elastic Container Service [ECS -fargate] ”

defacto : 사실상 표준

- 컨테이너 오케스트레이션 툴

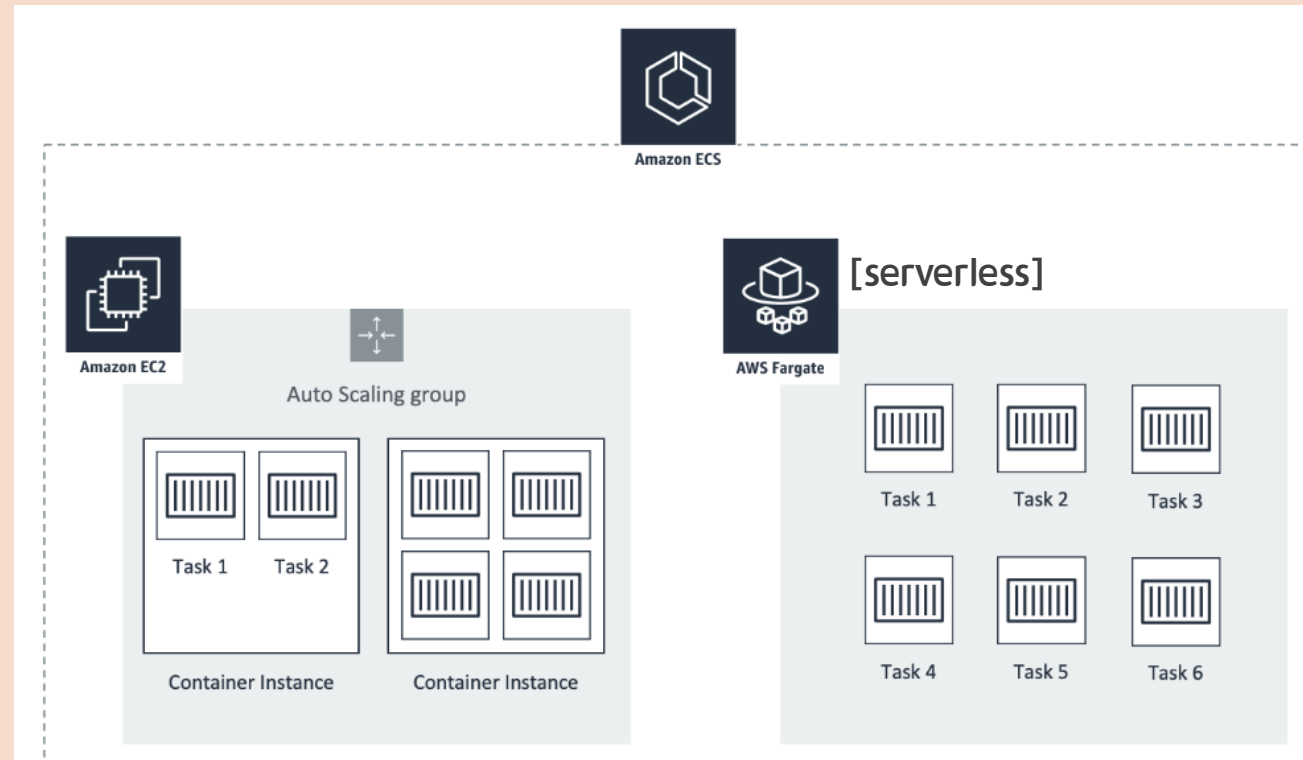
여러 종류의 컨테이너들을 ‘얼마나’, ‘어느 정도의 자원으로’, ‘어떻게’ 운용할지 관리하는 툴



- 현재 대표적인 오케스트레이션 툴(플랫폼)

- AWS ECS & ECS fargate
- Kubernetes [AWS에서 EKS로 서비스] – “defacto”
- Docker Swarm
- Apache Methos Marathon

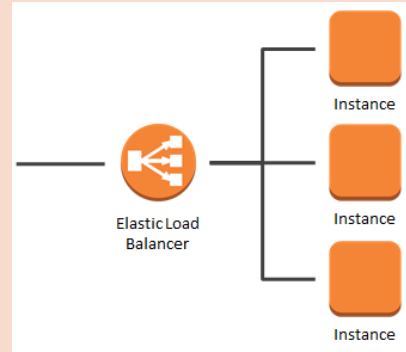
AWS ECS와 ECS fargate의 차이



02 사용한 AWS 서비스 소개

“AWS Elastic Load Balancer”

- 서버가 여러 대 존재할 경우, 부하를 분산



Application Load Balancer



생성

HTTP 및 HTTPS 트래픽을 사용하는 웹 애플리케이션을 위한 유연한 기능이 필요한 경우 Application Load Balancer를 선택합니다. 요청 수준에 따라 작동하는 Application Load Balancer는 마이크로서비스 및 컨테이너를 비롯한 애플리케이션 아키텍처를 대상으로 하는 고급 라우팅 및 표시 기능을 제공합니다.

[자세히 알아보기 >](#)

ALB - L7 스위치

Network Load Balancer



생성

애플리케이션에 초고성능, 대규모 TLS 오프로딩, 중앙 집중화된 인증서 배포, UDP에 대한 지원 및 고정 IP 주소가 필요한 경우 Network Load Balancer를 선택합니다. 연결 수준에서 작동하는 Network Load Balancer는 안전하게 초당 수백만 개의 요청을 처리하면서도 극히 낮은 지연 시간을 유지할 수 있습니다.

[자세히 알아보기 >](#)

NLB - L4 스위치

Classic Load Balancer

이전 세대
HTTP, HTTPS 및 TCP용

생성

EC2-Classic 네트워크에서 구축된 기존 애플리케이션이 있는 경우 Classic Load Balancer를 선택합니다.

[자세히 알아보기 >](#)

02 사용한 AWS 서비스 소개

“AWS Elastic Load Balancer”

- 부하 분산 작동 원리

- 1. 라운드 로빈
 - 특별한 우선 순위 없이, 주기적으로 다른 서버들에 부하 분산
- 2. 최소 미해결 요청 (least outstanding request)
 - 현재 pending(보류), unfinished(미완료)인 요청 수가 제일 작은 서버에 연결
- 3. 고정 (sticky session)
 - 사용자가 특정 서버에 접속하였을 경우, 해당 사용자는 로드밸런싱과 무관하게 해당 서버로만 접속 시킨다.

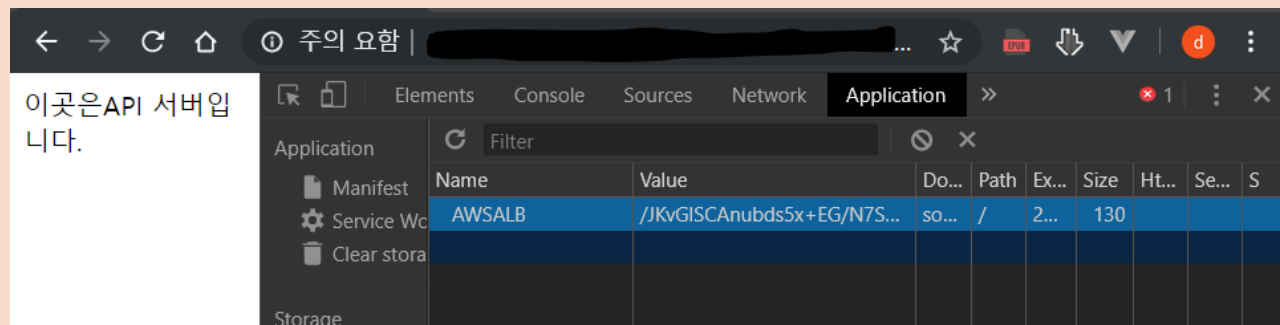
속성 편집

등록 취소 지연 ⓘ 300 초
0~3,600 사이의 값을 지정합니다.

느린 시작 기간 ⓘ 0 초
비활성화하려면 30에서 900 사이의 값을 지정하거나 0을 지정합니다.

로드 밸런싱 알고리즘 ⓘ ☐ 라운드 로빈 ☒ 최소 미해결 요청
고정 ⓘ ☐ 활성화

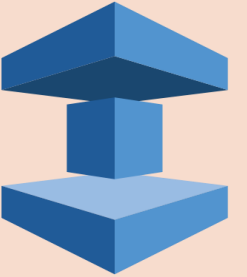
Tip) ALB(L7-switch)는 쿠키를 통해 사용자 접속상태를 판단한다.



02 사용한 AWS 서비스 소개

“AWS ElastiCache” In-Memory DB

접근이 잦은 데이터, 휘발성, 디스크 I/O가 부담될 때



AWS ElastiCache

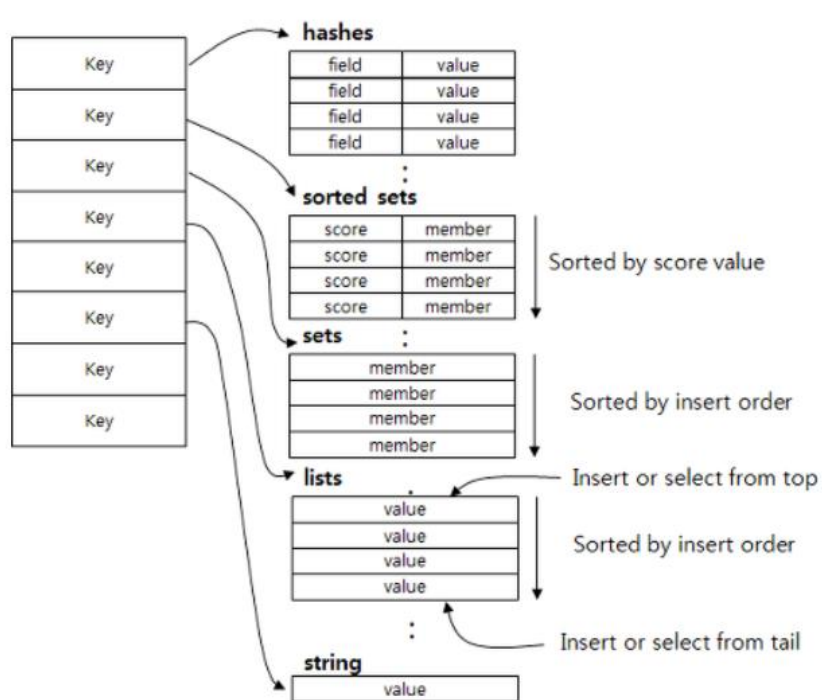


Memcached redis

AWS는 redis와 memcached를 지원 -> redis 사용

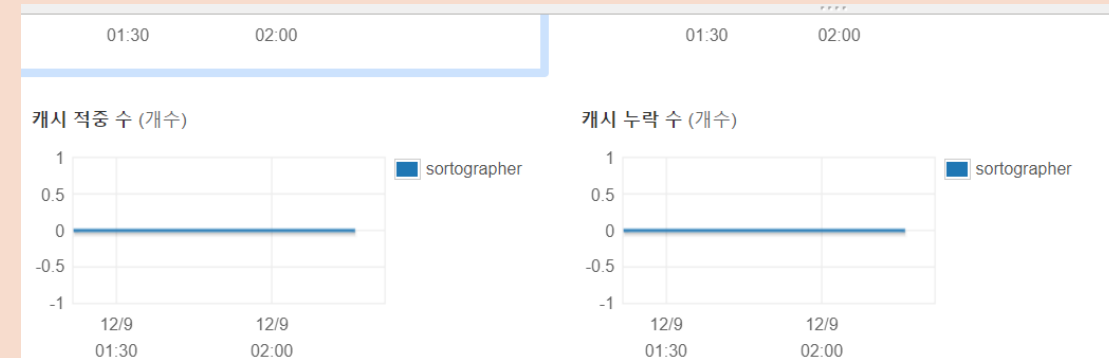
- redis

- < key, value > 형식의 NoSQL
- 데이터는 디스크가 아닌 메모리에 저장 (백업, 덤프는 디스크에 저장)
- 어플리케이션에서는 Spring-data-redis 사용 (구현체 : lettuce)



특징

- 메모리에 공간이 부족하면 LRU 알고리즘으로 제거
- 데이터 타입에 Expire 속성이 있어서 데이터의 수명을 정할 수 있음



02 사용한 AWS 서비스 소개

“ 프론트 엔드 ”

- 1. S3 정적 웹 호스팅 사용
View단은 SPA 형태의 단독 웹페이지
- 2. 웹서버 대용으로 CloudFront 사용 [웹 캐싱 + 프록시]
 - S3에서는 GET을 제외한 나머지 HTTP Request를 할 수 없음
 - S3에서 호스팅한 페이지를 CDN에서 캐싱 -> CDN에서 서버로 리버스 프록시

General

Origins and Origin Groups

Behaviors

Error Pages

Restrictions

Invalidations

Tags

CloudFront compares a request for an object with the path patterns in your cache behaviors based on the order of the cache behaviors in your distribution. Arrange cache behaviors in the order in which you want CloudFront to evaluate them.

Create Behavior

Edit

Delete

Change Precedence:

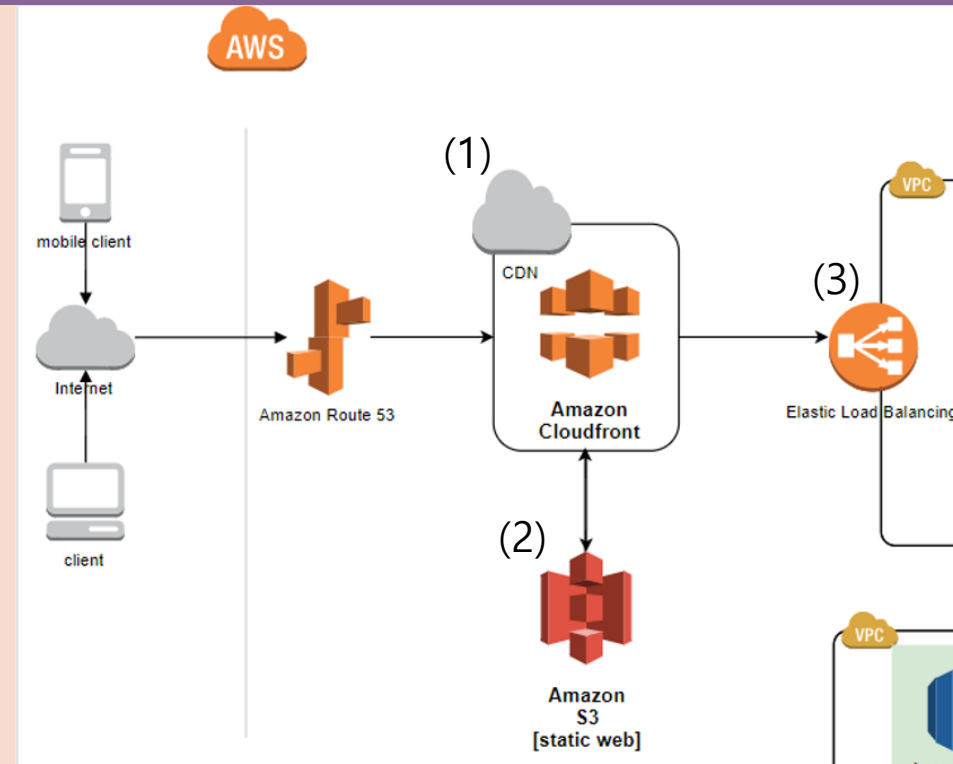
Move Up

Move Down

Save

	Precedence	Path Pattern	Origin or Origin Group	Viewer Protocol Policy
<input type="checkbox"/>	0	api/*	Custom-www.server-sortographer.cf/api	HTTP and HTTPS
<input type="checkbox"/>	1	Default (*)	Custom-www.app-sortographer.cf	HTTP and HTTPS

↳ CDN 상에서 /api/* 이하의 모든 요청은 REST API 서버로 전달



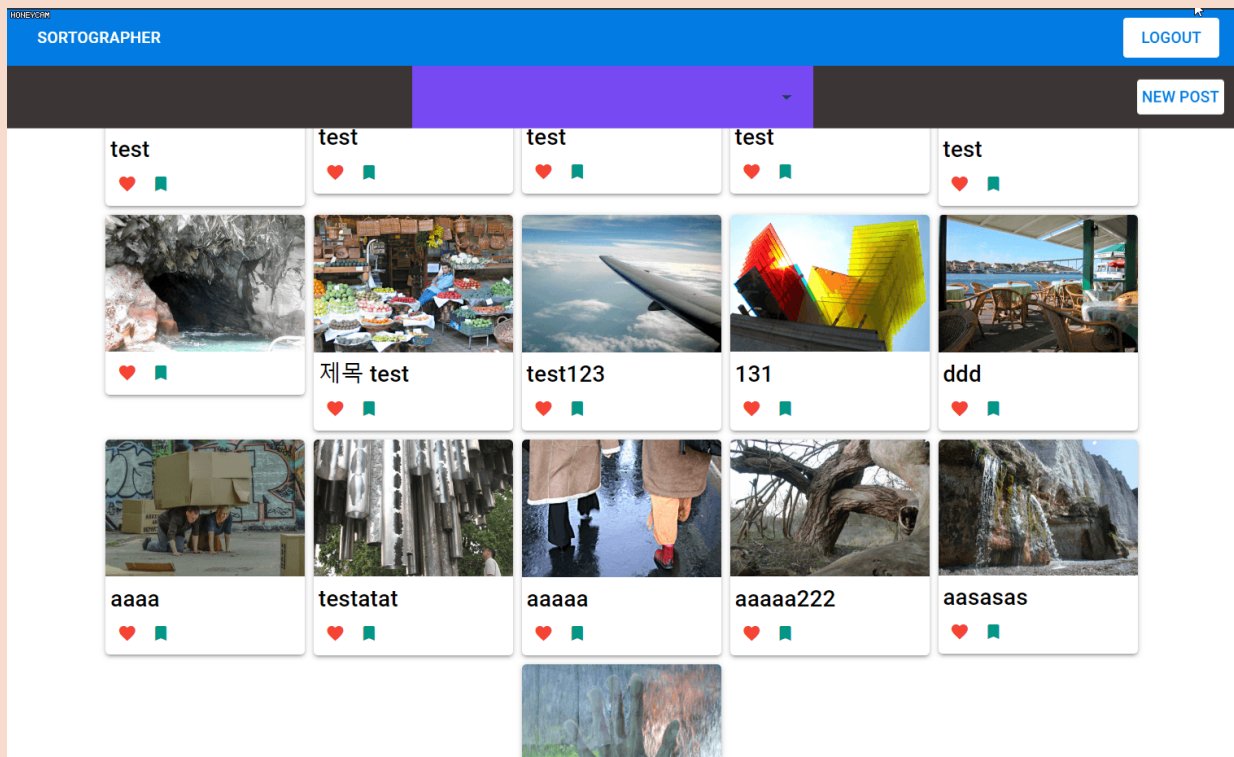
Route 53

- (1) www.sortographer.cf
- (2) www.app-sortographer.cf
- (3) www.server-sortographer.cf

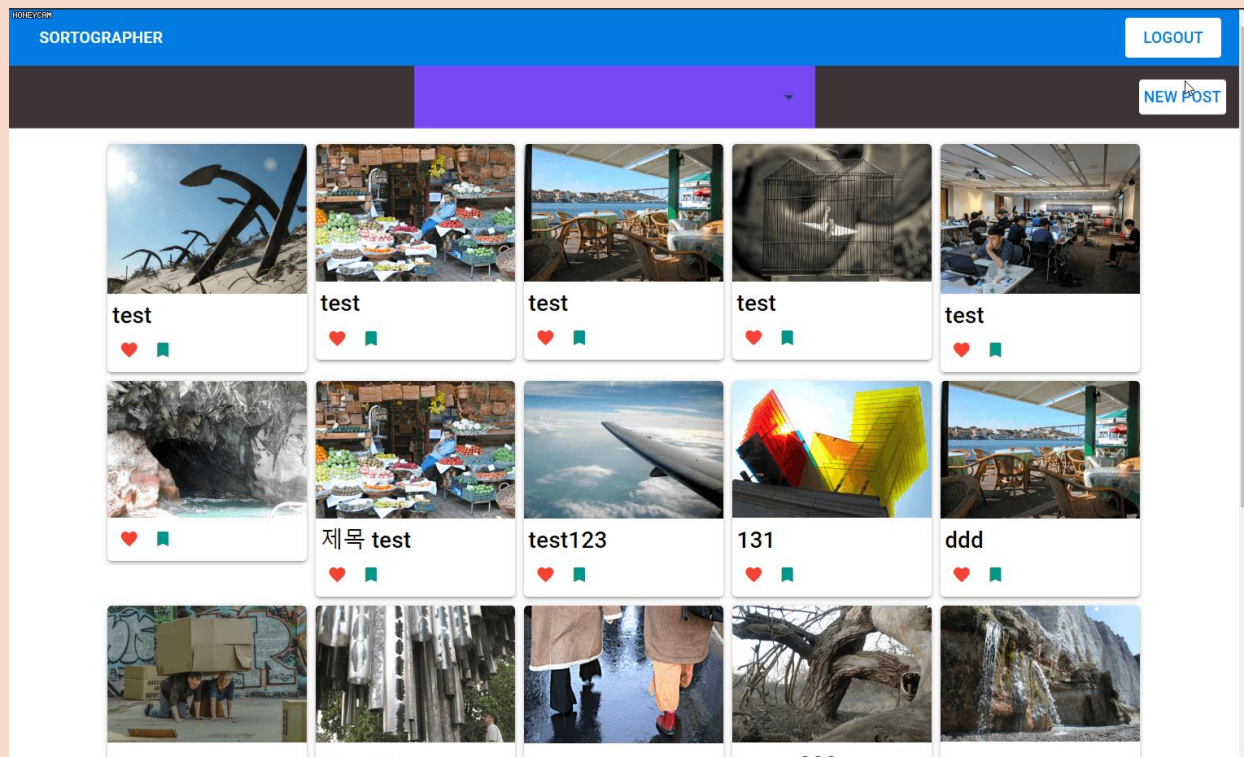
03 서비스 내용

서비스 실제 화면

시연 예정



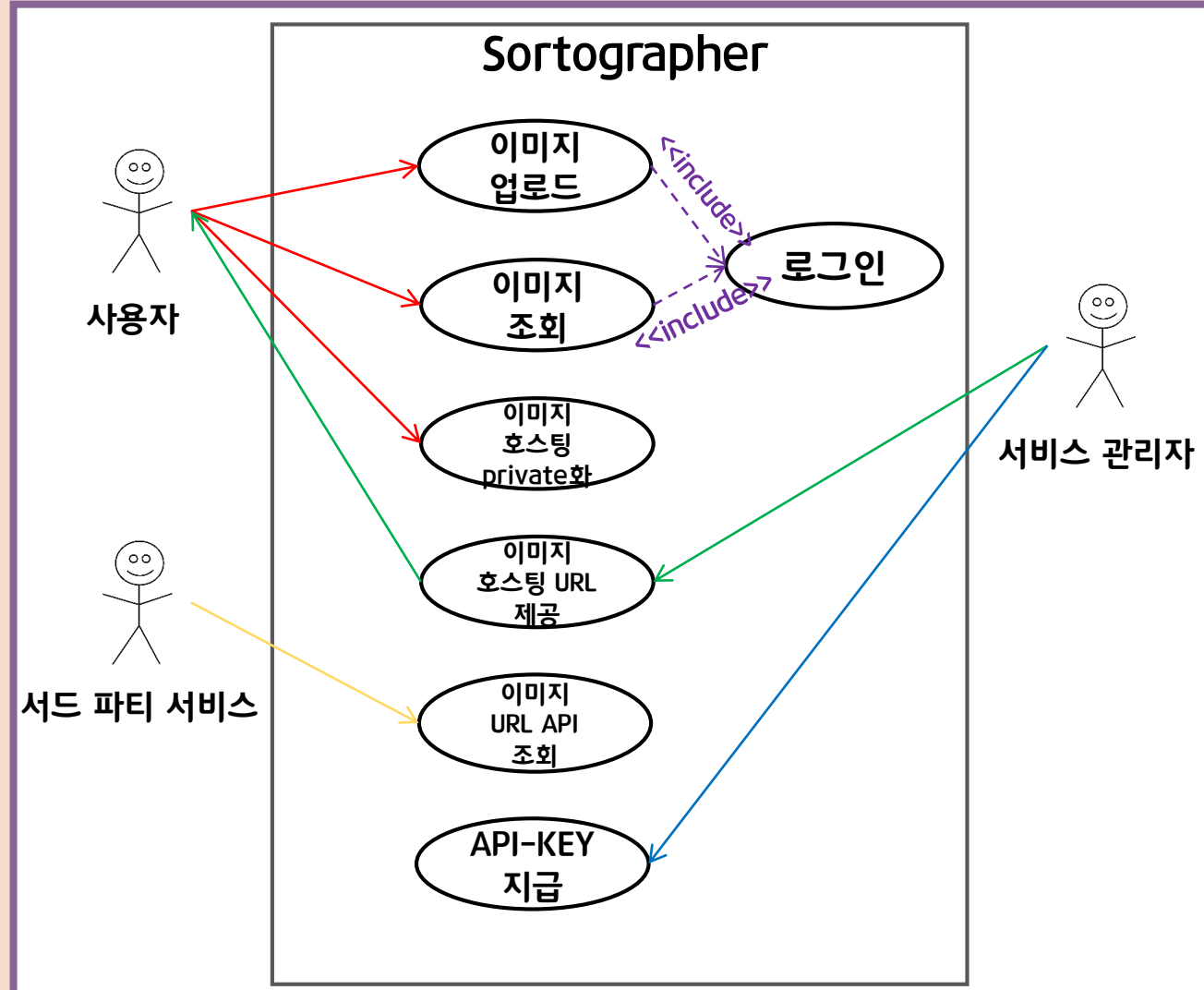
이미지 조회



이미지 분석

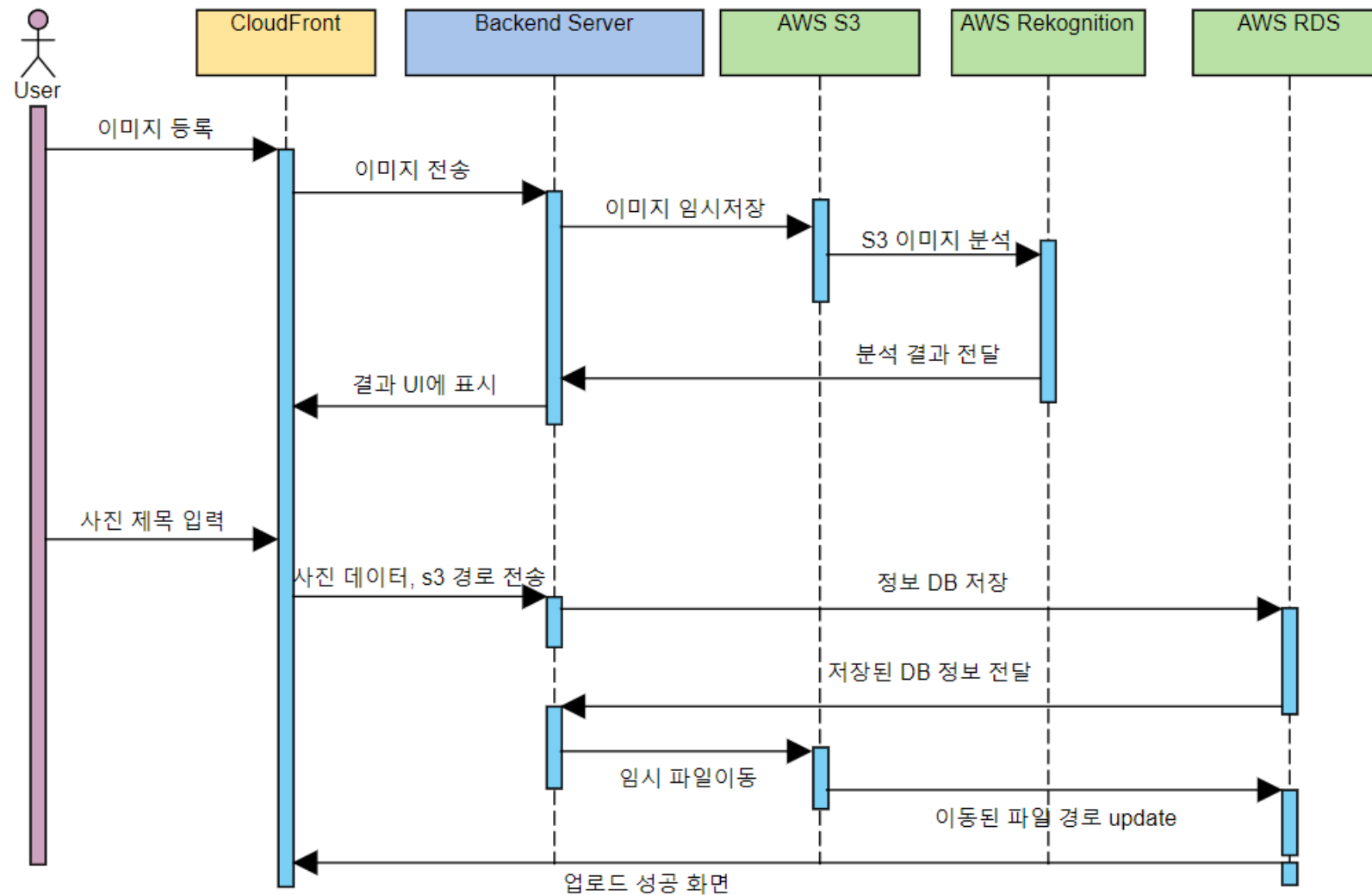
03 서비스 내용

“Sortographer – 이미지 분류 사진첩”



03 서비스 연동 시나리오

- 시퀀스 다이어그램 -

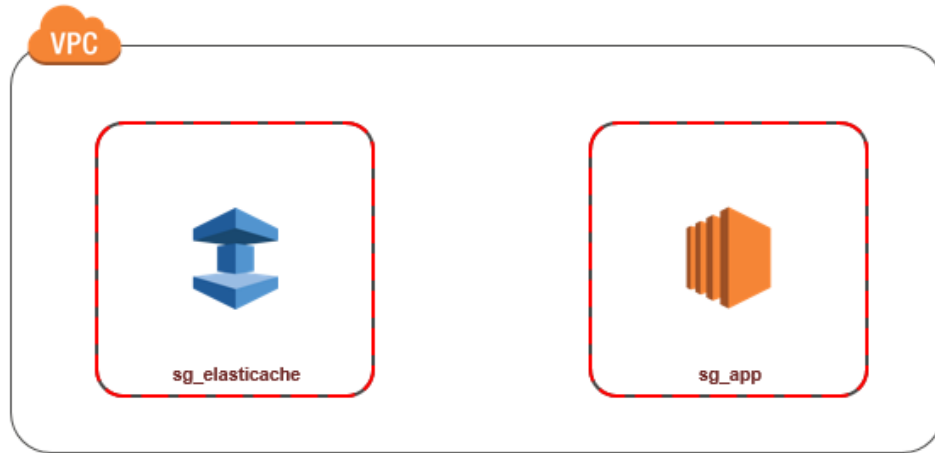


04 추진 일정 계획과 진행 실적

2019년 11월						
월	화	수	목	금	토	일
11	12	13	14	15	16	17
← 아이디어 회의 →			← 인프라 및 운영서버 세팅 →		← 개발 시작 →	
18	19	20	21	22	23	24
				1차 회의		
25	26	27	28	29	30	1
				2차 회의		
2019년 12월						
2	3	4	5	6	7	8
				최종 회의		
9	10	11	12	13	14	15
제출	발표		발표			

05 발견된 문제점 및 해결 방안

“서로 다른 VPC 상에서 elasticache 연결 문제”



동일한 VPC에서 EC2 인스턴스와 DB 인스턴스 간 액세스를 관리하는 가장 간단한 방법은 다음과 같습니다.

1. 클러스터의 VPC 보안 그룹을 만듭니다. 이 보안 그룹을 사용해 클러스터 인스턴스에 대한 액세스를 제한할 수 있습니다. 예를 들어, 클러스터를 만들 때 할당한 포트와 클러스터에 액세스할 때 이용할 IP 주소를 사용해 TCP 액세스를 허용하는 이 보안 그룹의 사용자 지정 규칙을 만들 수 있습니다.

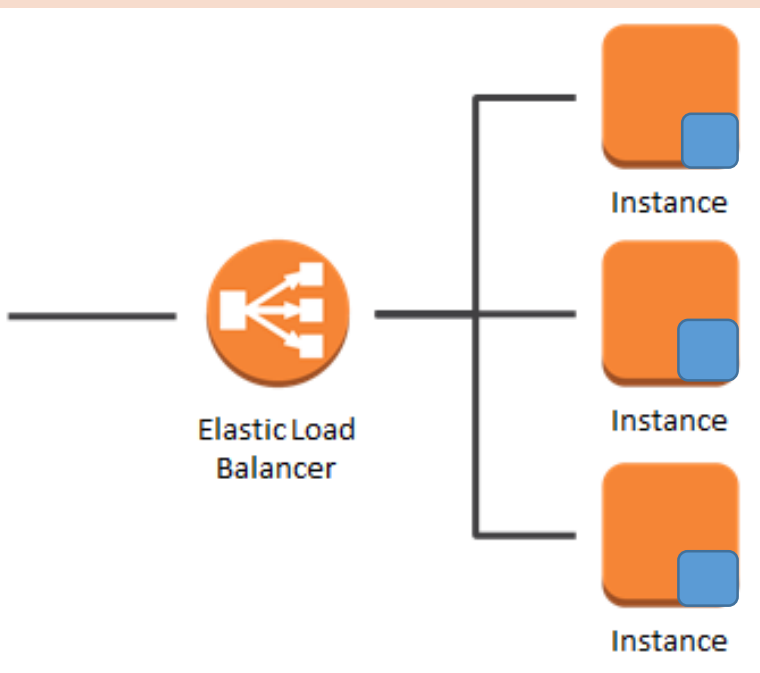
Memcached 클러스터의 기본 포트는 11211입니다.

2. EC2 인스턴스(웹 및 애플리케이션 서버)의 VPC 보안 그룹을 만듭니다. 이 보안 그룹은 필요할 경우 VPC의 라우팅 테이블을 통한 EC2 인스턴스 액세스를 허용할 수 있습니다. 예를 들어, 이 보안 그룹에서 TCP가 포트 22를 통해 EC2 인스턴스에 액세스하도록 허용하는 규칙을 설정할 수 있습니다.
3. 클러스터에 대한 보안 그룹에서 EC2 인스턴스에 대해 생성한 보안 그룹으로부터의 연결을 허용하는 사용자 지정 규칙을 만듭니다. 그러면 보안 그룹의 모든 구성원이 DB 인스턴스에 액세스하도록 허용됩니다.


06 이해결 문제점 및 예상 해결 방안

“Clustered – Server상에서의 회원 인증 문제”

■ WAS session



대안

1. Sticky Session 사용
2. 외장 세션 사용 – redis
3. 세션 정보를 클라이언트가 가지고 있도록 한다. 
 - JWT 토큰 사용
 - 높은 확장성 (앱, 다른 서비스에서 사용가능)

기존에는 유저 로그인 정보를 톰캣 내장 세션에 저장

Username

Password

Connect with:

- Facebook Login with Facebook
- Google Login with Google
- Vkontakte Login with Vkontakte
- Twitter Login with Twitter
- LinkedIn Login with LinkedIn
- Instagram Login with Instagram
- Amazon Login with Amazon
- Salesforce Login with Salesforce
- Microsoft Login with Microsoft

☐ Remember Me

OAuth 2.0
개방형 인증 표준

06 이해결 문제점 및 예상 해결 방안

“Clustered – Server상에서의 회원 인증 문제”

[Debugger](#)[Libraries](#)[Introduction](#)[Ask](#)[Get a T-shirt!](#)

Crafted by Auth0

ALGORITHM HS256Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImV4cCI6IjMwMDAwIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM0MDYyLCJpYXN0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImV4cCI6IjMwMDAwIn0
```

Signature or encryption algorithm

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT",  "exp": "30000"}
```

PAYLOAD: DATA

```
{  "sub": "1234567890",  "name": "John Doe",  "iat": 1516239022}
```

VERIFY SIGNATURE

```
HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  your-256-bit-secret  ) ☐ secret base64 encoded
```

JWT 구조

1. 웹표준 (RFC 7519)
2. 헤더, 정보, 서명으로 구성
3. 암호화, 토큰 유효기간을 통해 사용자 인증할 수 있음
4. 서버에서 토큰을 복호화하여 인증

07 조별 과제 진행 중 얻은 교훈

IT경영 2016314015 박재연

“AWS를 배운다 == AWS를 쓰는 방법을 배운다”

➡ 기존에 선행되어야 할 지식이 너무 많다

07 조별 과제 진행 중 얻은 교훈

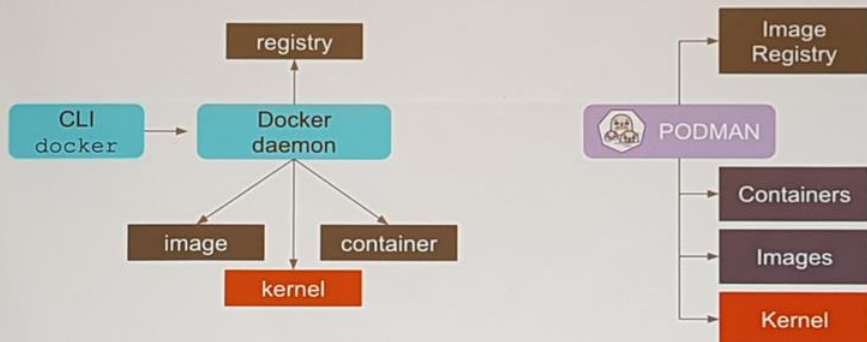
IT경영 2014316016 설동민

“신기술은 계속 나타나고 더 쓰기 편하게 발전한다”

-> 변하지 않는 컴퓨터 과학 기반 지식을 더 많이 공부해야만 한다.

컴퓨터 구조, 운영체제, 자료구조, 알고리즘, 네트워크...

Docker VS. Podman



1. 도커도 Defacto였지만, 점점 컨테이너 기술들이 도커의 의존성을 줄여나가고 있음

- 보안문제 (RHEL에서 표준은 docker가 아닌 PODMAN)
- Loose Coupling [Container Runtime Interface(CRI)]의 구현체로 제공

감사합니다

별첨1. ECR

“ AWS Elastic Container Repository ”

- 1. 도커 이미지를 만드는 Dockerfile과 여러 도커 이미지를 컨트롤하게 해주는 docker-compose

```
Dockerfile
1 FROM openjdk:11-jre-slim
2
3 VOLUME /tmp
4 ARG JAR_FILE=./build/libs/backend-0.0.1-SNAPSHOT.jar
5 COPY ${JAR_FILE} app.jar
6 ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app.jar"]
```

```
docker-compose.yml
1 version: '3'
2 services:
3   application:
4     container_name: spring-boot-app
5     image: sight/sortographer
6     build:
7       context: ./backend
8     ports:
9     - '8080:8080'
```

ECR > 리포지토리 > sight/sortographer

sight/sortographer 푸시 명령 보기

이미지 (3)

🔍 이미지 찾기

<input type="checkbox"/>	이미지 태그	이미지 URI	푸시 위치 ▼	다
<input type="checkbox"/>	latest	556347504352.dkr.ecr.ap-northeast-2.amazonaws.com/sight/sortographer:latest	19. 12. 05. 오전 10:07:44	🔗
<input type="checkbox"/>	<untagged>	556347504352.dkr.ecr.ap-northeast-2.amazonaws.com/sight/sortographer@sha256:32852113c9c2c4ee9a61b8850475f80f4bc757ca521664bd32e79d8df42d1b88	19. 12. 05. 오전 05:18:14	🔗
<input type="checkbox"/>	<untagged>	556347504352.dkr.ecr.ap-northeast-2.amazonaws.com/sight/sortographer@sha256:6785fee4cb5b6ef2b02460fca93a54c2e435c6c7f672be40d8b0bc6014c29a87	19. 12. 05. 오전 02:18:10	🔗

- 2. 만들어진 이미지는 도커 이미지 저장소에 보관
E.g. AWS ECR, Docker Hub

별첨2. ECS 오케스트레이션 방식

“ AWS Elastic Container Service [ECS –fargate] ”

- 예시1. 서버 컨테이너를 0.5vCPU, 1GB 메모리한도로 생성

작업 크기

작업 크기를 통해 작업의 고정된 크기를 지정할 수 있습니다. 작업 크기는 Fargate 시작 유형을 사용하는 작업에 대해 필요하며, EC2 시작 유형에 대해서는 선택 사항입니다. 작업 크기가 설정되면 컨테이너 수준 메모리 설정은 선택 사항입니다. 작업 크기는 Windows 컨테이너에서 지원되지 않습니다.

작업 메모리(GB)

1GB

0.5vCPU에 대한 유효한 메모리 범위: 1GB - 4GB.

작업 CPU(vCPU)

0.5 vCPU

1GB 메모리에 대한 유효한 CPU 범위: 0.25 vCPU - 0.5 vCPU.

컨테이너 메모리 예약에 대한 작업 메모리 최대 할당

0512 공유됨 1024 MiB

컨테이너에 대한 작업 CPU 최대 할당

0512 공유됨 512 CPU 단위

컨테이너 정의

컨테이너 추가

컨테이너 이름	이미지	하드/소프트 메모리 제한(MiB)
spring-boot-server	556347504352.dkr.ecr.ap-northeast-2.amazonaws.com/si...	1024/512

- 예시3. 서버 컨테이너의 메모리가 70% 이상일때 : 컨테이너 추가, 40% 이하일 때 컨테이너 제거

세부 정보	작업	이벤트	Auto Scaling	배포	측정치	Tags	로그
최소 작업: 3			최대 작업: 3				
ScaleOutPolicy: MemoryUtilization >= 70			ScaleInPolicy: MemoryUtilization >= 40				
정책 유형: 단계 조정			정책 유형: 단계 조정				
경보 발생 시: Mem-Higher-then-70			경보 발생 시: Mem-lower-then-40				
다음 작업을 수행:			다음 작업을 수행:				
추가 1 작업 일시 70 <= MemoryUtilization			제거 1 작업 일시 40 <= MemoryUtilization				

- 예시2. 각 컨테이너 상황에 따라 컨테이너를 동적으로 증가/감소

service sortographer registered 1 targets in target-group ecs-sortographer
service sortographer registered 2 targets in target-group ecs-sortographer
service sortographer has started 3 tasks: task 43159cbb-7a60-4578-a235-7aeca0c905cb task 0557780d-1aaa-47bc-abb4-652e58ed15fe task e3ae5ed4-0f78-4c2d-bf23-8f7d44cba6e0 .
service sortographer has stopped 3 running tasks: task 29795b12-3b60-4003-85ab-8fe9f6243737 task bc236858-03fe-4ac7-b238-a686b637b11f task dd657b6e-76c8-4722-bf25-8dffce0d26c0 .
service sortographer deregistered 3 targets in target-group ecs-sortographer
service sortographer (port 8080) is unhealthy in target-group ecs-sortographer due to (reason Health checks failed with these codes: [404]).