최종 보고서

- 클라우드 컴퓨팅 -

프로젝트명	의료계 종사자를 위한 익명 웹서비스		
작성일	2023.06.20 (화)	버전	v1.0
팀장	정민준(32184074)		
팀원	배지현(32192004), 홍찬희(32185010)		

목 차

표 목차

사진 목차

A. 프로젝트의 목표 및 내용

1) 프로젝트의 최종 목표

병원 관계자와 비 관계자 간의 소통 및 정보 공유를 위한 커뮤니티 구축을 목표로 삼아 쿠버네티스와 MSA 아키텍쳐를 사용하여 안정적이고 확장성있는 서비스를 구현한다.

2) 프로젝트의 내용

제작 동기

지인이 근무하는 병원의 원장님께서 그 지역에서 유명한 의사이지만, 실제로는 돈을 벌기 위해 효과 없는 치료도 감행하고 있다고 들었다. 따라서 관계자만이 알 수 있는 정보를 소비자와 나누는 투명성 있는 의료 커뮤니티가 필요하다고 생각했다.

사용대상

- 병원이나 의료 정보를 알고 싶은 비 관계자
- 의료 커뮤니티가 필요하거나 병원의 정보를 알려주고/알고 싶은 관계자

내용

사용자를 관계자와 비 관계자로 나눈다. 관계자는 회원가입 절차를 통해 본 서비스를 이용할 수 있고 비 관계자는 정보 조회에 한해서, 회원가입 없이도 서비스를 이용할 수 있다.

본 서비스는 자유게시판, 병원 정보 및 리뷰 기능을 담고 있다.

- 자유게시판은 관계자와 비 관계자 모두 게시글을 조회할 수 있으며 회원가입을 한 사용자에 한해 질의응답 또는 커뮤니티 등이 가능하다.
- 병원 정보 및 리뷰는 병원에 대한 정보를 제공하며 관계자들이 이용할 수 있는 공간이다.
관계자는 병원과 관련된 리뷰 조회/쓰기가 가능하며 비 관계자는 리뷰 조회만 가능하다.

기대 효과

관계자와 비 관계자들은 정보를 공유하고 소통하며 더 나은 의료 서비스를 제공하는 데 도움이 될 것이다. 또한, 모두가 익명을 사용해 자유롭고 투명한 제보가 가능해 비관계자가 더 나은 병원을 선택할 수 있는 기회를 제공한다.

개발 범위

결과물: 의료계 블라인드 웹서비스

프로젝트 로드맵 및 타임라인:

5월 둘째 주: 개발 환경 세팅
5월 셋째 주: 프론트서버, 인증서버, 게시판 서버, 병원정보/리뷰 서버 개발
5월 넷째 주: 쿠버네티스 설치 및 셋팅
5월 다섯째 주: 평가기준 테스트

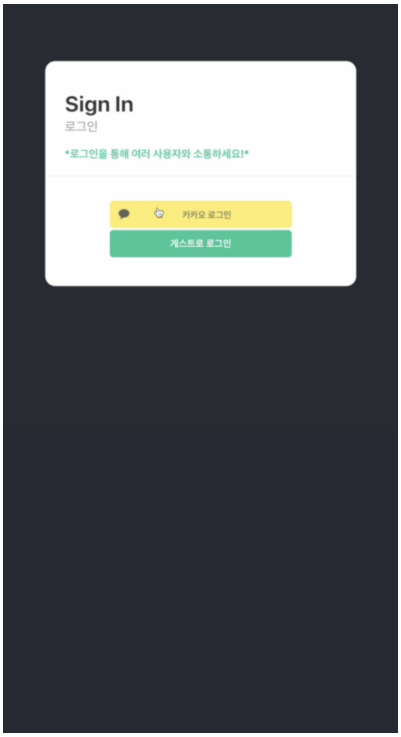
제약 사항

회원가입 절차에 대해 까다롭게 진행하기가 어려웠다. 보유한 병원의 정보가 한계가 있고 그 병원에 직접 근무하는 관계자인지, 병원에 확인하기 어렵다는 제약 사항이 있다.

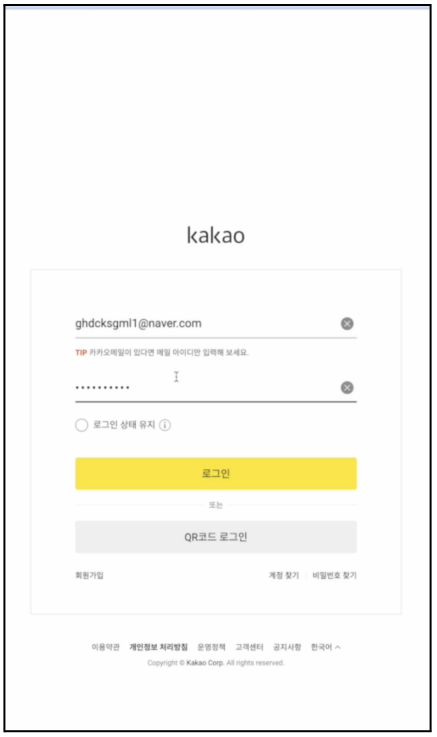
웹사이트 시연

〈병원 관계자 시점〉

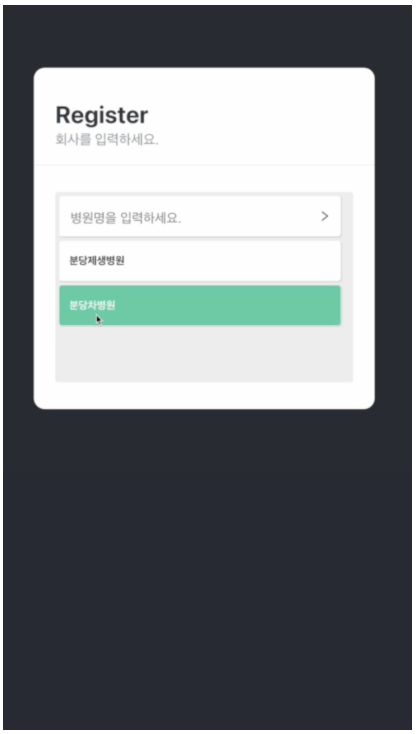
① 카카오톡 로그인



[사진 1] 관계자 시점 로그인



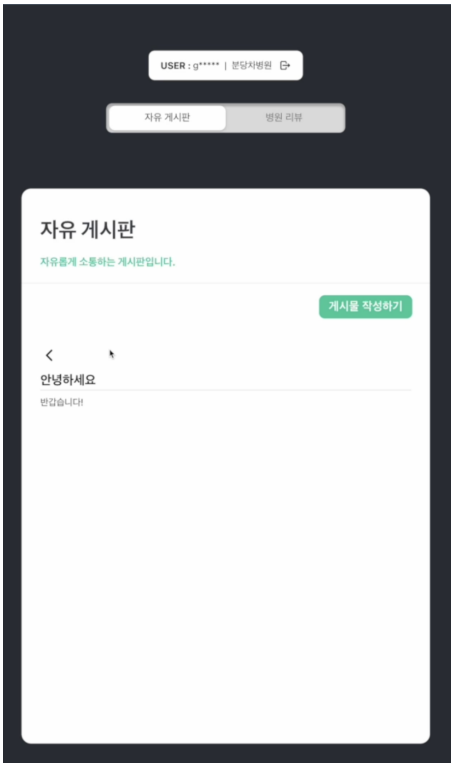
[사진 2] 카카오톡 로그인 진행



[사진 3] 근무 중인 병원 선택

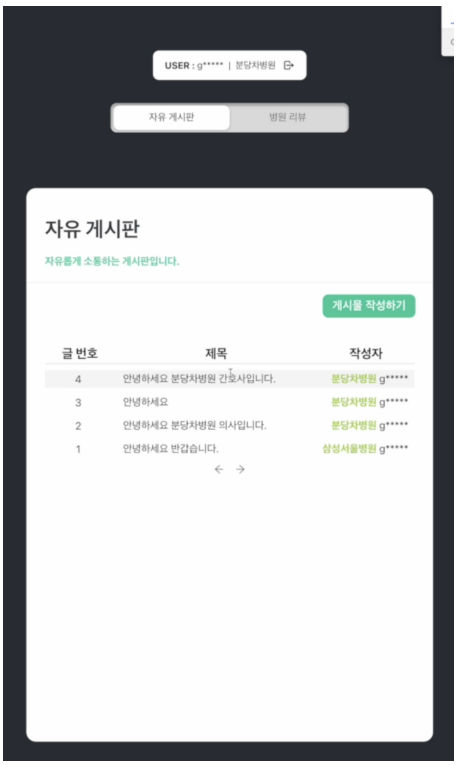
② 자유게시판

[사진 4] 게시물 목록 조회



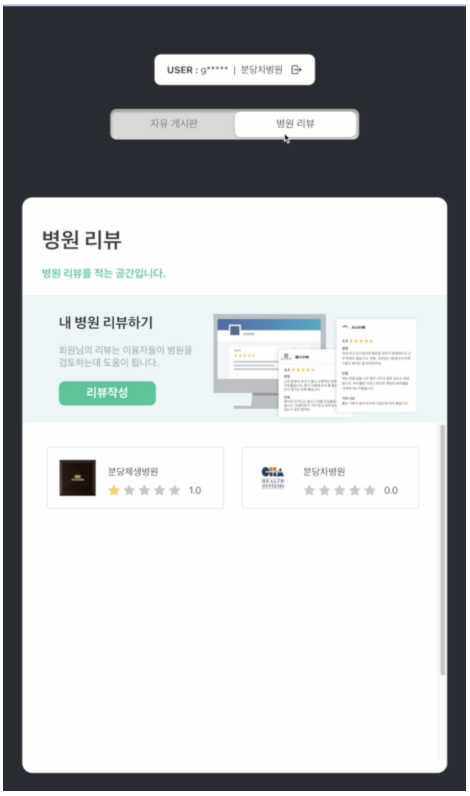
[사진 5] 게시글 조회

[사진 6] 게시글 목록 조회 > 게시글 작성하기 > 글 작성



[사진 7] 글 작성 결과 (4번 게시글 추가)

③ 병원 정보 및 리뷰



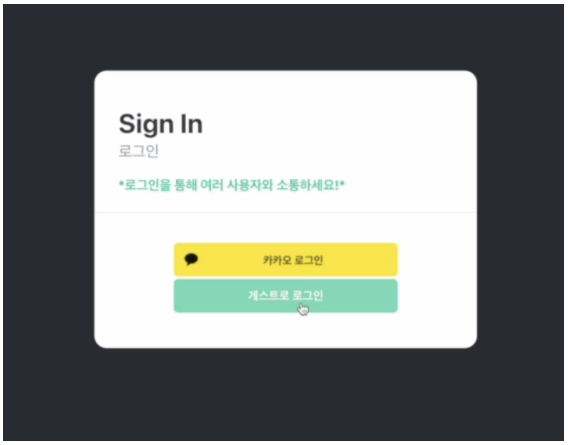
[사진 8] 병원 목록 조회

[사진 9] 병원의 리뷰 조회

[사진 10] 병원 리뷰 작성

[사진 11] 병원 리뷰 작성 결과

〈비 관계자 시점〉



[사진 12] 비 관계자 시점 로그인

[사진 13] 자유 게시판

[사진 14] 병원 리뷰

관계자와 달리, 관계자는 글 또는 병원 리뷰를 작성할 수 없고 조회만 가능하다. 따라서 작성하기 버튼이 보이지 않는다.

B. 프로젝트의 추진전략 및 구조

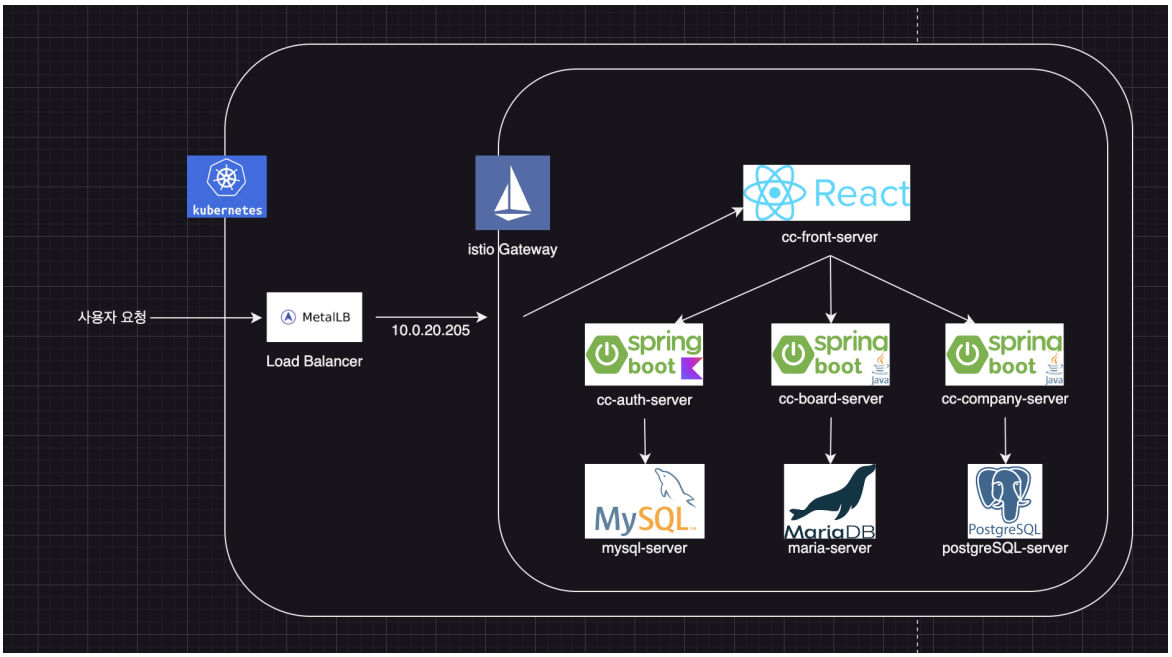
1) 프로젝트의 추진전략

- 로그인, 게시판, 병원 평가, 웹 프론트 서버로 나누어 서비스를 개발한다.
- 깃허브로 의사소통하며 개발을 하며, Github Webhook과 Jenkins를 이용해 CI/CD를 구축한다.
- 쿠버네티스 서버는 마스터노드 서버 1개와 워커노드 서버 2개로 구성한다.

2) 프로젝트의 중점사항

- 마이크로 서비스 아키텍처 설계를 통해 단일 장애 지점 문제를 최소화시켜 서버 안정성을 향상시키면서, 각 서버의 필요 리소스에 맞게 스케일링 한다.
- 각 서버별로 다른 종류의 DB를 사용하여 데이터 가용성을 지킬 수 있다.
- 각 DB별로 독립적인 PV(PersistentVolume), PVC(PersistentVolumeClaim)을 생성해 Pod가 종료되어도 데이터를 영구적으로 저장할 수 있다.
- 쿠버네티스의 Self-healing을 이용해 비정상적인 컨테이너를 자체적으로 교체하여 안정적인 서비스를 유지하도록 한다.
- 쿠버네티스의 Rolling Update 기능을 이용해 새로운 버전으로 배포 시 서비스가 중단되지 않는 무중단 배포를 구현한다.
- 쿠버네티스의 Auto-Scaling 기능을 이용해 HPA(HorizontalPodAutosacler)로 CPU 사용량에 따라 레플리카 셋 개수를 조절한다.
- Jmeter와 chrome의 lighthouse를 이용해 반응 속도와 서비스 성능을 측정 한다.

3) 프로젝트의 구조



[사진 15] 쿠버네티스 서비스 구조

프로젝트의 구조는 React 애플리케이션인 cc-front-server와 Spring Boot 애플리케이션인 cc-auth-server, cc-board-server, cc-company-server로 이루어져있고, mysql, maria, postgreSQL 데이터베이스를 사용하고 있다. MSA 아키텍처를 구성하기 위해 백엔드 애플리케이션으로 하나로 통합하는 것이 아닌, 관심사별로 로그인 서버, 게시판 서버, 회사 정보 서버로 나누어 각각의 서비스를 구현했다. 또한 각 서버마다 다른 데이터베이스를 구축해 서비스간 느슨한 결합이 실현되도록 했다.

이렇게 구현한 서비스들을 쿠버네티스 환경에 띄우고, istio Gateway를 통해 istio가 서비스 트래픽을 관리하면서, 각 서버에 라우팅 되도록 설정했다. 해당 쿠버네티스 환경은 베어메탈 서버에 설치되어 있기 때문에 따로 설치된 로드밸런서가 존재하지 않으므로, MetalLB를 통해 IP를 할당해 외부에서 접근가능하도록 설정했다.

4) 쿠버네티스 기능 시연

4-1. Auto-healing

```yaml
  replicas: 3
  selector:
    matchLabels:
      app: cc-front-server
  template:
    metadata:
      labels:
        app: cc-front-server
    spec:
      containers:
        - image: ghdcksgml1/cc-front
          name: cc-front-server
          ports:
            - containerPort: 80
          livenessProbe:
            httpGet:
              path: /
              port: 80
            initialDelaySeconds: 15 # 시작하고 몇초 기다릴지
            periodSeconds: 20 # 20초에 한번씩 healthCheck
            timeoutSeconds: 1 # 1초안에 200 응답이 안오면 실패로 간주
            successThreshold: 1 # 1번 성공하면 성공으로 간주
            failureThreshold: 3 # 3번 실패하면 실패로 간주
          resources:
            requests:
              cpu: 200m
              memory: 250Mi
            limits:
              cpu: 1
              memory: 500Mi
```

[사진 16] cc-front Deployment Manifest

쿠버네티스의 Auto-healing 기능을 설정하여 사용하기 위해 먼저 각각의 파드에 livenessProbe 설정을 추가해주었다. livenessProbe 설정은 컨테이너의 상태를 주기적으로 체크해 주는 기능을 한다. 우리의 서비스는 웹 서비스이기 때문에 httpGet에 설정되어있는 경로로 주기적으로 요청을 보내 컨테이너가 동작 가능한 상태 인지를 파악한다.

파드가 생성되고 정상인 기능이 동작하는데에 약간의 시간이 소요되므로, initialDelaySeconds를 통해 파드가 시작되고 생성되고 나서 몇초를 기다릴지 설정을 해주었고, healthCheck 를 너무 자주하는 것을 방지하기 위해 15번씩 요청을 보내도록 설정했다. 또한, QoE를 만족시키기 위해 timeoutSeconds를 설정해 1초안에 200 응답이 오지 않을 경우 실패로 간주했다.

마지막으로, 단 한번의 실패로 파드가 내려갔다 올라가는 것을 방지하기 위해 failureThreshold를 3으로 설정해 3번 연속으로 실패가 발생하면, 파드를 재시동해주도록 설정했다.

4-2. Auto-scaling

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: cc-front-hpa
  labels:
    app: cc-front-hpa
  namespace: default
spec:
  minReplicas: 3
  maxReplicas: 10

  metrics:
  - resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50 # 평균 CPU 사용량이 50 이상일 경우 스케일 아웃
    type: Resource

  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: cc-front-server
```

[사진 17] cc-front HPA Manifest

오토 스케일링 기능을 추가하기 위해 Deployment에 HPA라고 불리는 HorizontalPodAutoscaler를 추가하였다. minReplicas를 3으로 설정해 평소에는 파드 개수를 3개로 설정해두고, maxReplicas를 10으로 설정해 설정한 기준보다 높은 트래픽이 몰리는 경우 10개까지 파드의 개수가 늘어나도록 설정했다.

파드가 늘어나는 기준은 CPU의 사용량이 50% 이상일 경우 스케일 아웃이 되도록 설정했다.

```
NAME           REFERENCE                      TARGETS    MINPODS   MAXPODS   REPLICAS
cc-front-hpa   Deployment/cc-front-server     4%/50%     3         10        3
cc-front-hpa   Deployment/cc-front-server     33%/50%    3         10        3
cc-front-hpa   Deployment/cc-front-server     92%/50%    3         10        3
cc-front-hpa   Deployment/cc-front-server     92%/50%    3         10        6
```

[사진 18] 서버 부하 테스트

다음은 Apache Jmeter를 통해 초당 300개의 요청을 보냈을때의 상황에서 레플리카 개수의 변화를 살펴보았다. 부하가 걸리면서 CPU의 사용률이 증가하여 50%를 넘자 부하를 분산시키기 위해 레플리카가 기존 3개에서 6개까지 늘어난 모습을 확인할 수 있다.

4-3. Persistence Volume

이 프로젝트는 총 3개의 서버로 구성되어 있고, 마스터 노드 1개와 워커 노드 2개로 구성되어 있다. 쿠버네티스에서 파드는 일회성 자원이다. 즉, 파드가 올라갔다 내려가는 동시에 파드의 모든 내용은 초기화된다. 이러한 특성은 데이터베이스를 파드로 띄울 때 파드가 재생성될 때 기존의 데이터가 초기화되는 현상을 초래할 수 있다. 이를 해결하기 위해 볼륨 마운트라는 기술을 통해 데이터 초기화를 방지할 수 있다. 하지만, 일반적인 쿠버네티스의 볼륨 마운트는 파드가 실행되는 워커노드와 연결되기 때문에 위와 같이 워커노드가 2개 이상 존재하는 상황에서 파드가 기존에 연결되어있던 워커노드 서버와 다른 서버에서 다시 파드가 생성될 경우 다시 새로운 워커노드에서 볼륨이 마운트되는 현상이 발생한다. 이를 해결하기 위해 퍼시스턴트 볼륨을 사용하였다.

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
spec:
  capacity:
    storage: 5Gi # 용량 설정
  accessModes:
    - ReadWriteOnce # ReadWriteMany 같은 옵션도 존재함. (Many는 여러명 접근가능)
  persistentVolumeReclaimPolicy: Retain # Pod가 종료되어도 유지하겠다.
  storageClassName: standard # 스토리지 클래스이름
  hostPath: # 어디에 데이터를 저장할지
    path: /home/master/cloud-computing/data
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: standard
  resources:
    requests:
      storage: 5Gi
```

[사진 19] MySQL PV, PVC Manifest

```yaml
apiVersion: v1
kind: Service
metadata:
  name: cc-auth-server-database
spec:
  selector:
    app: mysql
  ports:
  - protocol: TCP
    port: 3306
    name: mysql
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:8.0.33
          imagePullPolicy: Always
          env:
            - name: MYSQL_HOST
              valueFrom:
                configMapKeyRef:
                  name: mysql-config
                  key: host
            - name: MYSQL_PORT
              valueFrom:
                configMapKeyRef:
                  name: mysql-config
                  key: port
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: password
            - name: MYSQL_DATABASE
              valueFrom:
                configMapKeyRef:
                  name: mysql-config
                  key: database
          ports:
            - containerPort: 3306
              name: mysql
          volumeMounts:
            - name: mysql-pvc
              mountPath: /var/lib/mysql
      volumes:
        - name: mysql-config
          configMap:
            name: mysql-config
            items:
              - key: "database"
                path: "database"
              - key: "host"
                path: "host"
              - key: "port"
                path: "port"
              - key: "user"
                path: "user"
        - name: mysql-pvc
          persistentVolumeClaim:
            claimName: mysql-pvc
```

[사진 20] MySQL Service, Deployment Manifest

퍼시스턴트 볼륨의 데이터는 마스터노드에 생성되어 관리되기 때문에, 파드가 어느 서버에 생성되던지 간에 볼륨 마운트를 할 수 있다. 각각의 데이터베이스마다 퍼시스턴트 볼륨을 생성해 퍼시스턴트 볼륨 클레임과 연결해주고, 연결한 클레임을 볼륨 마운트를 통해 파드와 연결해주었다.

데이터베이스 파드 설정에서는 환경변수를 통해 데이터베이스 컨테이너가 생성될 때 초기 생성할 스키마와 계정 정보, 사용 포트 등을 정의했다. 마지막으로, Cluster IP 서비스 타입을 사용해 해당 데이터베이스를 쿠버네티스 내부에서 사용할 수 있도록 설정했다.

이를 통해 데이터베이스 서버가 중단되거나 재생성 되는 경우에도 데이터를 유지할 수 있었다.

4-4. Load Balancing

쿠버네티스에서 부하 분산을 위한 로드밸런싱 기능을 사용하기 위해 Service와 Deployment를 이용하였다. 먼저 Deployment에 파드를 정의하고 Replicas 개수를 설정해 원하는 숫자의 ReplicaSet을 생성한다. 그 후 Deployment와 Service를 연결시켜 외부로부터 Service에 접근할 때 쿠버네티스에 설정된 알고리즘에 따라 부하가 균등하게 각각의 Replica로 분산시켰다.

C. 프로젝트 결과물의 평가기준

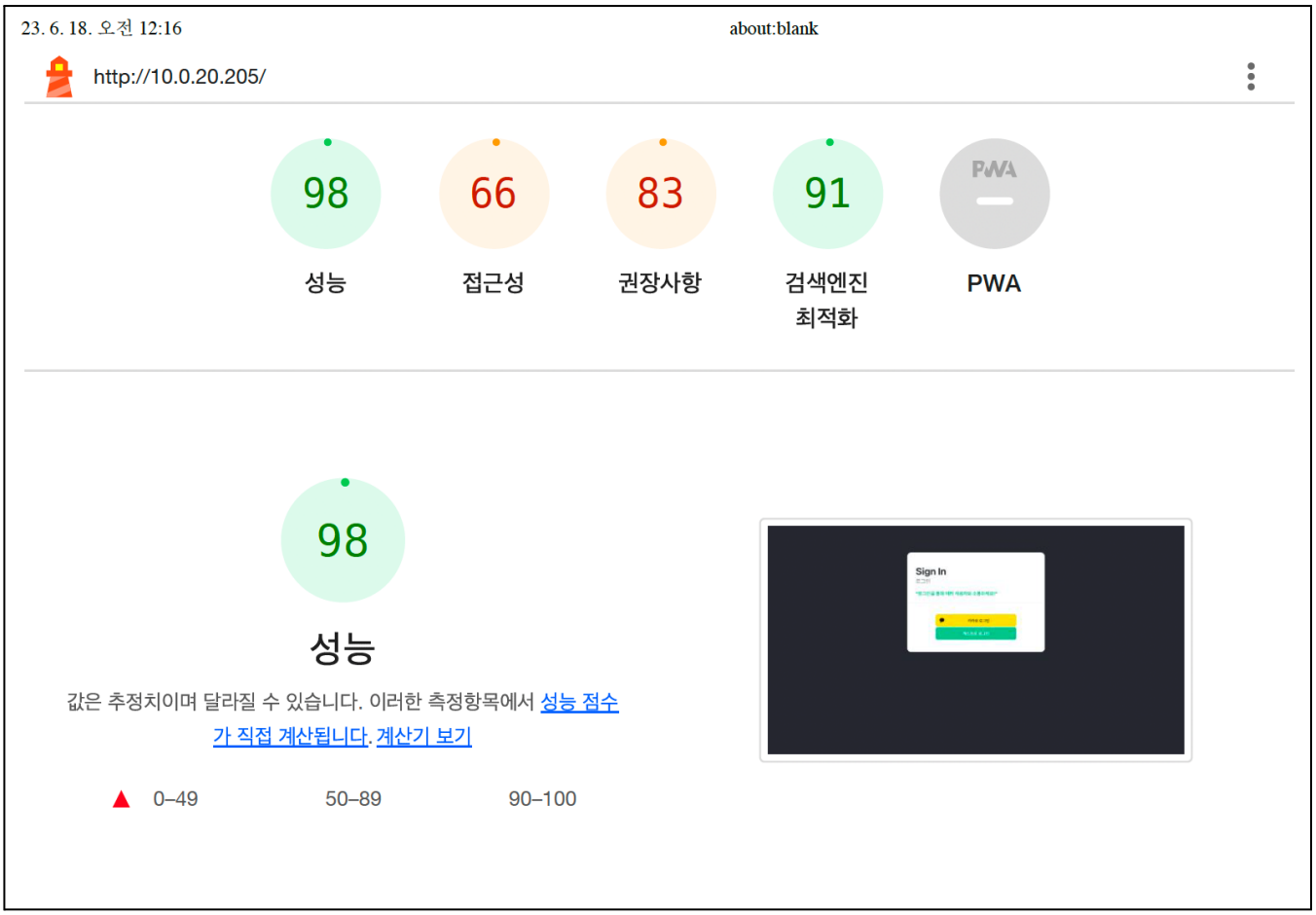
1) 정량적 평가기준

목표치는 아래와 같다.

평가 항목	정량적 목표	평가 방법
초당 접속자 수	총 300명이 자유게시판, 병원 정보 및 리뷰에 동시 접속할 수 있다.	프로메테우스를 이용해서 실시간 CPU 로드율 및 메모리 사용량, HTTP 요청 수, HTTP 응답 시간을 확인한다.
First Contentful Paint	총 300명이 자유게시판, 병원 정보 및 리뷰에 동시 접속하더라도 응답 시간은 FCP 기준 800ms 이내를 보장한다.	Lighthouse를 사용해서 FCP를 측정한다.
Time To Interactive	총 300명이 자유게시판, 병원 정보 및 리뷰에 동시 접속하더라도 응답 시간은 TTI 기준 1000ms 이내를 보장한다.	Lighthouse를 사용해서 TTI를 측정한다.
Max Potential First Input Delay	총 300명이 자유게시판, 병원 정보 및 리뷰에 동시 접속하더라도 응답 시간은 FID는 기준 800ms 이내를 보장한다.	Lighthouse를 사용해서 FID를 측정한다.
Total Blocking Time	총 300명이 자유게시판, 병원 정보 및 리뷰에 동시 접속하더라도 응답 시간은 TBT는 기준 800ms 이내를 보장한다.	Lighthouse를 사용해서 TBT를 측정한다.

[표 1] 정량적 평가기준

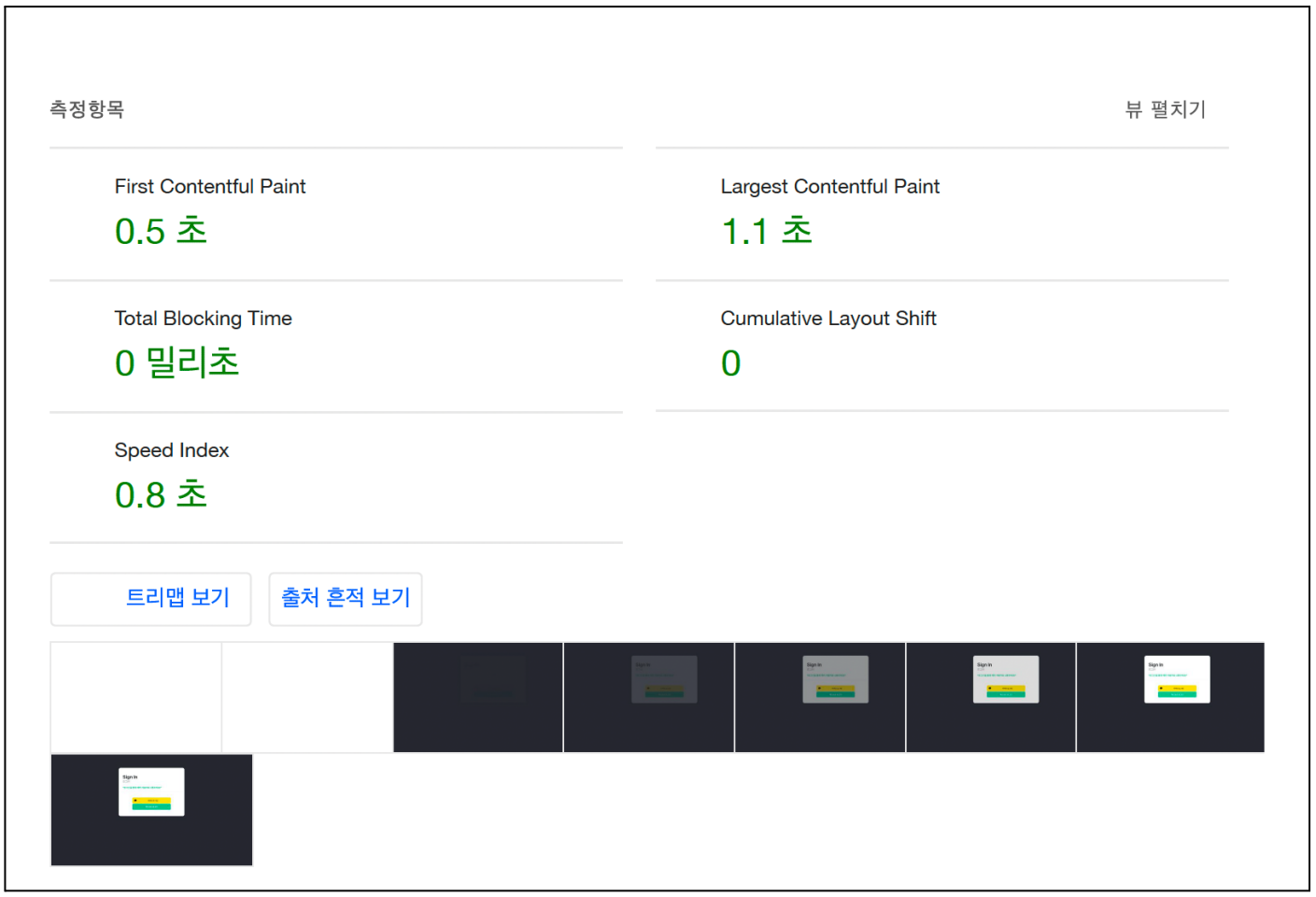
실제 결과는 아래와 같다.



[사진 21] LightHouse 성능테스트 결과

정량적 평가 기준을 정할때 5분동안 300명의 사용자가 동시에 접속하는 상황을 가정했다. 따라서 Jmeter를 사용해서 동일한 부하 상황을 갖추고 1초에 한번씩 웹사이트에 요청을 보내는 상황을 구성한다.

테스트 조건에 맞는 부하가 걸린 상황에서 전체적인 성능은 98점으로 실제 서비스에서 사용자가 만족스럽게 사용할 수 있을것으로 예상된다.



[사진 22] LightHouse 성능테스트 결과 수치

실제 수치 결과를 보면 위와 같다.

목표: 총 300명이 자유게시판, 병원 정보 및 리뷰에 동시 접속하더라도 응답 시간은 FCP 기준 800ms 이내를 보장한다.

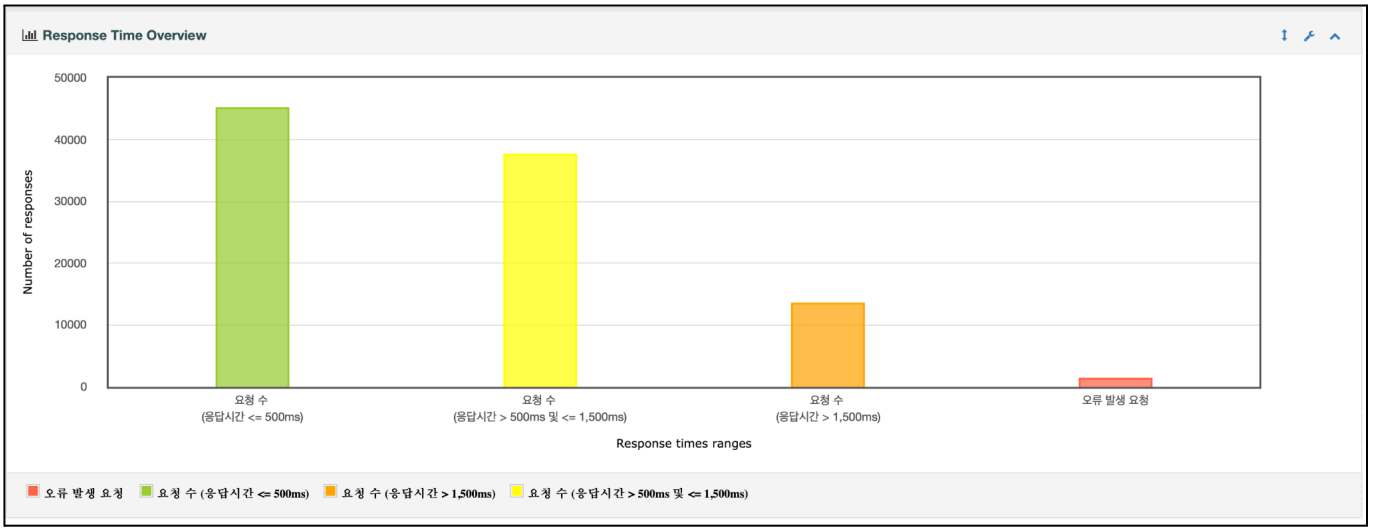
결과: 목표 조건에서 실제 측정된 FCP는 500ms로 FCP 목표를 달성한것을 확인할 수 있다.

목표: 총 300명이 자유게시판, 병원 정보 및 리뷰에 동시 접속하더라도 응답 시간은 FID,TBT 기준 800ms 이내를 보장한다.

결과: FID와 TBT는 같은 계산지표를 공유한다. 목표 조건에서 실제 측정된 TBT는 0ms로 FID와 TBT 목표를 달성한것을 확인할 수 있다.

목표: 총 300명이 자유게시판, 병원 정보 및 리뷰에 동시 접속하더라도 응답 시간은 TTI 기준 1000ms 이내를 보장한다.

결과: 아래 그래프를 보면 대부분의 요청에 대해서 1000ms 안에 요청에 응답하고 있으나 과부하가 걸림에 따라 일부 요청은 목표 응답시간을 초과하는 것을 볼 수 있다. 이는 운영중인 서버의 성능이 부족이 원인으로 생각된다. 그러나 목표 조건에서 요청에 대해서 평균적으로 1000ms 안에 응답한다고 볼 수 있다.



[사진 23] Apache Jmeter Response Time Overview

2) 정성적 평가기준

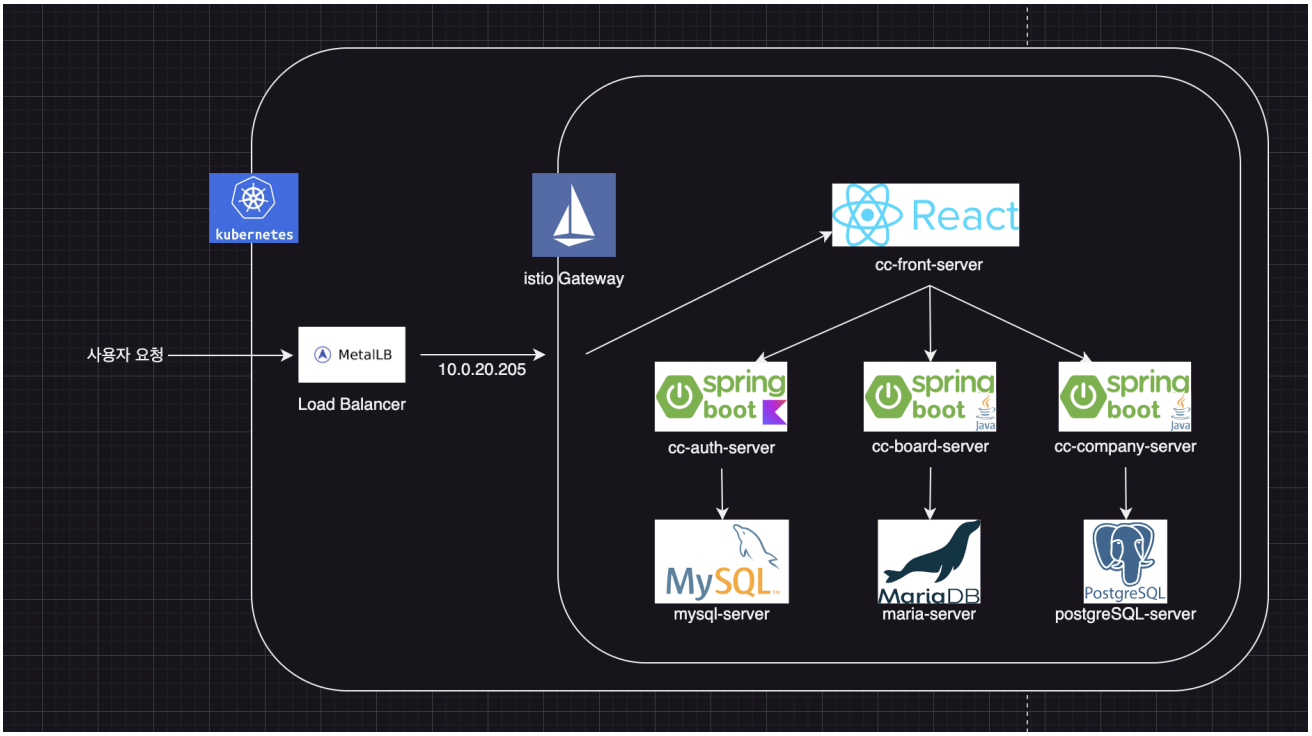
목표치는 아래와 같다.

평가 항목	정성적 목표	평가 방법
서비스 배포	쿠버네티스상에 MSA 구조의 서버 배포	쿠버네티스 상에서 파드가 정상적으로 작동하는지 조회, 외부에서 사용자가 접속할 수 있는지 확인한다.

로그인	로그인 기능의 정상 작동	회원 가입 시 입력한 정보가 DB에 정상적으로 등록되고 로그인할 수 있는지 확인한다.
자유게시판	자유게시판 기능의 정상 작동	자유게시판에 게시글을 작성하고 해당 게시글을 조회할 수 있는지 확인한다.
병원 정보 및 리뷰	병원 정보 및 리뷰 기능의 정상 작동	병원 정보 및 리뷰 게시판에 게시글을 작성하고 해당 게시글을 조회할 수 있는지 확인한다.

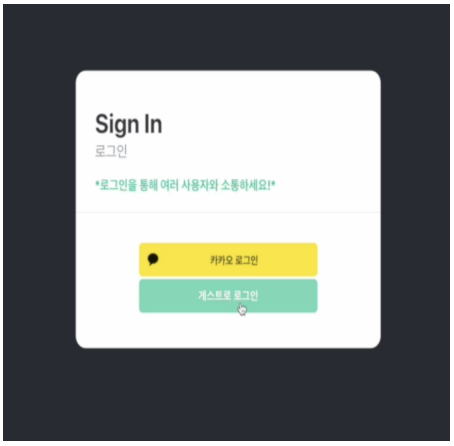
[표 2] 정성적 평가기준

실제 결과는 아래와 같다.



[사진 24] 쿠버네티스 서비스 구조

위와 같은 MSA 구조로 애플리케이션을 설계했다. 각 DB를 포함한 서비스는 모두 독립적으로 실행되며 REST API로 통신하도록 전체적인 서비스를 설계했다. 쿠버네티스 환경에서 배포되기 때문에 AutoHealing, Auto Scaling, PV, Load Balancing, Container Orchestration과 같은 기능들을 이용할 수 있었다.



[사진 25] 로그인 화면

[사진 26] 자유 게시판 화면

[사진 27] 병원 리뷰 화면

위의 웹사이트 시연 부분과 발표 시연 영상을 통해서 정성적 평가 기준에 해당하는 로그인, 자유 게시판, 병원 리뷰 게시판이 정상적으로 작동하는 것을 확인할 수 있다.

- 참고문헌(Reference)

[1] 조대협. (2023). "LivenessProbe에 대한 설명.". Retrieved from https://bcho.tistory.com/1264
[2] Kubernetes Documentation. https://kubernetes.io/ko/docs/concepts/storage/persistent-volumes/