

DevSecOps 융합 인재 양성 과정

웹 예약 시스템 구축

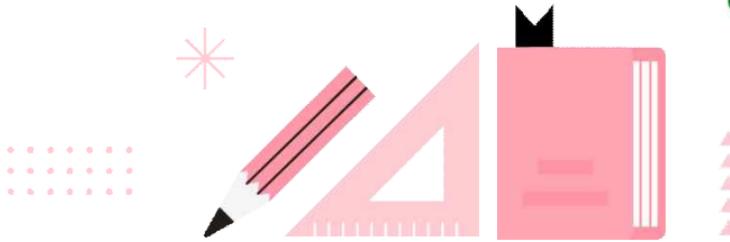
다솜 Study Cafe



김JAVAS

구정빈, 위주현, 안유진, 윤병국, 엄진환, 조현우

Contents



프로젝트 팀 구성 및 역할	01
주요기능 / 개발환경	03
Front-end	05
Docker	07
CI/CD	09
프로젝트 개요 및 목적	02
UI/UX 제작	04
back-end	06
AWS	08
멘토링 : ORM, 타입스크립트	10

01 팀 구성 및 역할



Dasom
StudyCafe



구정빈(팀장) - 백엔드

- 예약페이지 기능 구현
- 백엔드 구성 및 미들웨어 구축
- 백엔드 모듈 통합 및 기능구현
- PM 역할을 및 프로젝트 기획
- 데이터베이스 정규화 설계
- AWS Lambda 코드작성

위주현 - UI/UX, AWS

- AWS Architect (비용 효율적 & CI/CD)
- Docker를 통한 컨테이너 개발환경 구축
- Docker Hub를 통한 이미지 공유환경 구축
- CloudFront - S3 정적페이지 호스팅
- Route53을 통한 사용자 지정 도메인 적용
- Figma를 통한 웹페이지 UI/UX 시각화

안유진 - 프론트, AWS

- 프론트엔드 개발, 사용자 인터페이스 설계
- API 명세 작성 및 관리
- 데이터베이스 설계(ORM 구축 및 설계)
- CI/CD 구축 및 Github Actions를 사용
- 프론트엔드 자동화 파이프라인 구축
- API Gateway 및 Lambda 구축
- Docker 환경 설정 및 워크플로우 파일 작성

01 팀 구성 및 역할



윤병국 - 백엔드

- 데이터베이스 ERD 설계 및 구축
- Mypage 기능구현
- 회원정보관리, 비밀번호 변경
- API Gateway - AWSLambda 통합 및 테스트



엄진환 - 백엔드

- 데이터베이스 ERD 설계 및 구축
- Login / Sign-in 기능 구현
- ID, 비밀번호 찾기 백엔드 구현
- Database Query 모듈화
- ORM을 활용한 백엔드 코드 모듈화
- AWS Lambda 코드 작성
- API Gateway - AWSLambda 통합 및 테스트



조현우 - 프론트

- 프론트엔드 개발
- 갤러리 이미지 애니메이션 구현
- ID, 비밀번호 찾기 프론트 구현
- 회원 탈퇴 사용자 승인 구현
- 외부 API를 활용한 지도 구현(Kakao Map)
- 페이지 정적 구성요소 배치(Text, Image)

프로젝트 개요 및 목적



| 개요

클라이언트가 인증을 통해 예약사이트를 사용하도록 하며
서버관리자는 고객정보, 시스템관리, 데이터 관리 등을 통해 유지보수 할 수 있어야 한다.

| 시스템 운용 목적

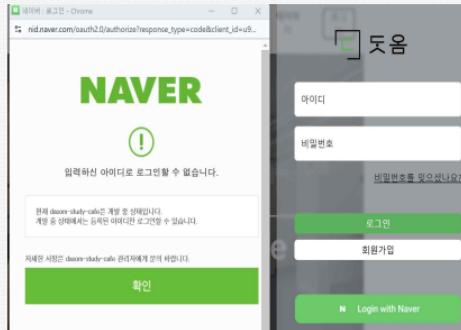
1. 현장에 방문하지 않아도 사전에 잔여 좌석을 확인할 수 있다.
2. 스터디카페의 구성요소(금액, 자리구성, 위치)등 을 확인할 수 있다.
3. 잔여시간을 확인하기 위해 직접방문하지 않아도 웹에서 쉽게 접근이 가능하다.
4. 관리자는 고객의 사전예약 정보를 통해 시설관리, 고객관리를 철저히 할 수 있다.



03주요기능



1



회원가입 - 소셜로그인

- 회원가입 및 로그인 기능 구현
- 회원가입시 중복검사 기능
- 아이디, 비밀번호 찾기 기능
- 로그인 상태 유지
- 소셜로그인(네이버) 및 SNS 인증

2



서비스 예약 및 예약 조회

- 시간, 좌석을 선택하여 예약기능 구현
- 중복된 일정에 좌석은 예약이 불가능하다..

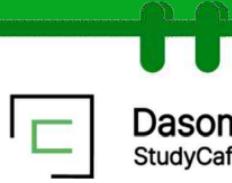
3



사용자 정보 관리

- 회원정보 수정이 가능하다.
- 예약 조회, 취소가 가능하다.
- 회원이 아니면 접근이 불가능하다.

03 주요 기능



Dasom
StudyCafe

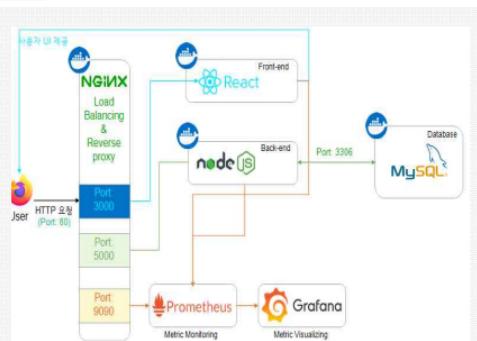
4

회원 정보					
구분	이름	ID	가입일자	전화번호	마지막 로그인 시간
1	관리자	admin	2024-11-05	010-0000-0000	2024-12-05 06:40 오후
2	임진환	kimjwhan	2024-11-05	010-1111-1111	2024-11-28 02:10 오후
3	조현우	johw	2024-11-05	010-2222-2222	2024-11-21 04:18 오후
4	안유진	anyu	2024-11-12	010-5445-9617	2024-11-21 09:53 오후
5	최주현	choiw	2024-11-22	010-9861-3335	2024-11-22 05:52 오후
6	임진환	kimjwhan	2024-11-27	010-3333-3333	-
7	안유진	anyu	2024-12-05	010-5445-9617	2024-12-05 06:38 오후

관리자(admin) 페이지

- 사용자 정보조회 기능
 - 고객의 로그인 이력, 예약정보 조회
 - 예약정보 삭제 권한
 - 회원탈퇴 권한

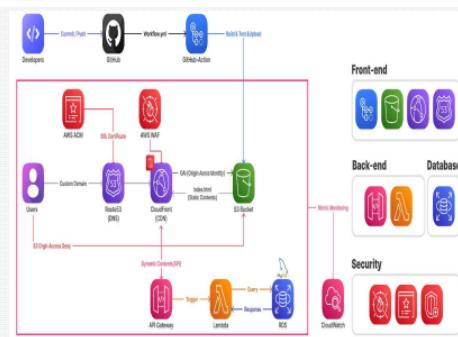
5



Docker 환경구축

- NGINX를 통한 웹 관리, 로드밸런싱
 - 컨테이너화 및 Docker Compose 활용
 - 프로메테우스와 그라파나 활용

6



AWS 및 CI/CD 구축

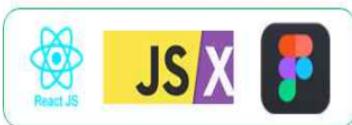
- 무중단 배포 및 관리 아키텍처 구축
 - LAMADA를 활용 기능 구현
 - RDS와 통한 데이터관리
 - GitHub Actions을 통한 CI/CD 구축

03 개발환경



On-premises

Front-end



Back-end



Database



Collaboration

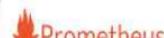
Containerization



Load-Balancing



Metric Monitoring & Visualizing



On-premises

CI/CD & AWS Migration

Front-end



Back-end



DataBase

CI/CD



Monitoring & Logging

진행순서

1차 : 온프레미스 개발



2차 : Docker 이미지 생성 및 Docker hub 사용 숙달



3차 : AWS의 CI/CD 파이프라인 구축 및 배포 숙달

목표

서비스의 안정성과 손쉬운 확장기능 구현

UI/UX 제작

| 브랜드 로고 제작

| Header, Bottom 구성

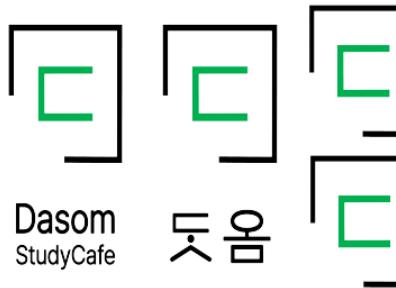
| 페이지 레이아웃 구성 및 컨텐츠 구성



Dasom
StudyCafe

04 UI/UX

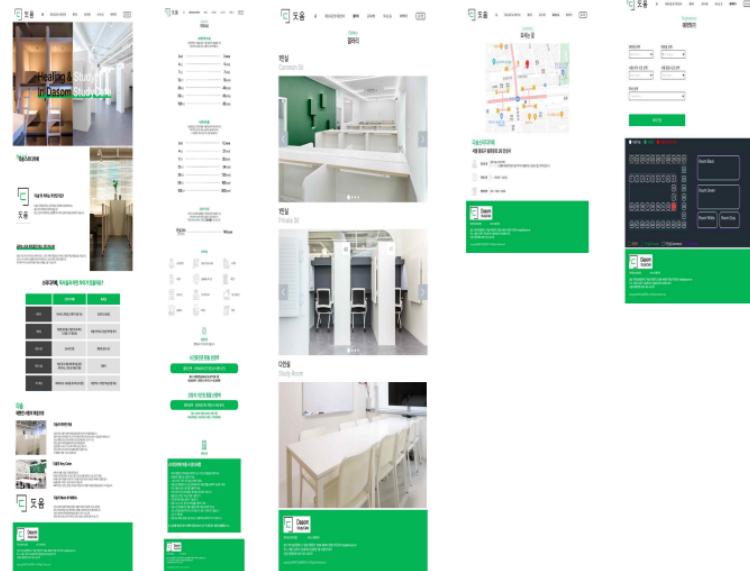
| 브랜드 로고제작



| Header, Bottom 구성



| 페이지 레이아웃



Front-end

| 사이드바 동적운용

| 로그인 절차

| 좌석예약 절차

| OpenAPI 활용
(카카오맵, 소셜로그인, sms로그인)



Dasom
StudyCafe

05 front-end (1/6)



사이드바

```
const [ renderPage, setRenderPage ] = useState(<LoginPage onChangePage={() => {  
    setRenderPage(<SignUpPage onChangePage={() => {  
        setRenderPage(<LoginPage />)  
    }}/>)  
}}>)
```

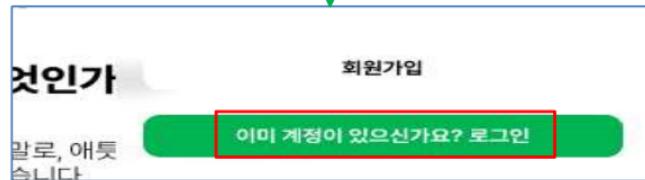
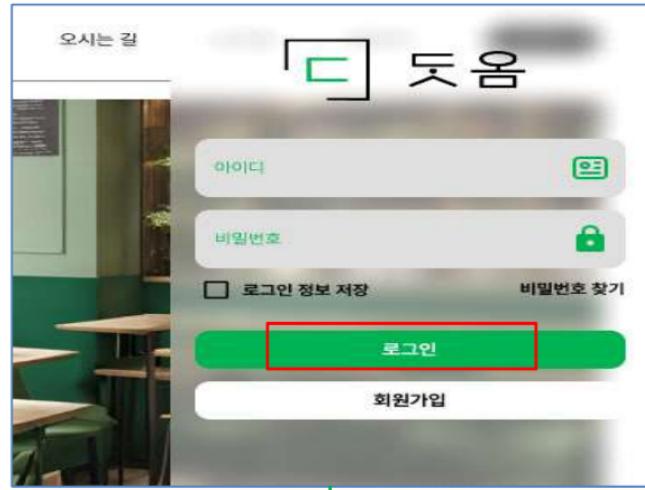
useState로 로그인 사이드바를 동적으로 관리

1. useState이후 LoginPage는 로그인 페이지가 기본으로 표시된다.

2. SignUpPage는 회원가입페이지

3. forgotUserd은 사용자 정보 찾기 페이지

※ 조건부 렌더링 없이 상태값에 따라 페이지 전환이 가능해진다.



05 front-end (2/6)

로그인

```
//서버 전송 함수(axios post)
const conTest = () => api.post('login', data)
.then((res) => {
    //로그인 성공했을 때
    if(res.data.success) {
        localStorage.setItem("id", form.id);
        localStorage.setItem("name", res.data.name);
        alert(localStorage.getItem("name")+'님 로그인 되었습니다.');
        // handleShow();
        props.onLogin();
    }
})
```

로그인 정보는 localStorage에 저장

1. localStorage에 저장하여 로그인 정보를 유지
2. 비밀번호 등 중요 정보는 노출되지 않도록 설정
3. 로컬저장 후 로그인(onLogin)을 사용하기 위해 props 설정
4. 로그아웃이 logoutHandler를 통해 저장 정보를 remove 해준다.

브라우저 스토리지 정보

<https://dasomstudy.site>

Origin https://dasomstudy.site

Key	Value
id	nicewjdlqls
isLoggedIn	1
name	구정빈

로그인 시 프로필 생성



Before Login



After Login

05 front-end (3/6)

좌석예약

```
<p>예약일 선택</p>
<DatePicker
    showIcon
    locale={ko}
    className="reserve-form"
    dateFormat="yyyy년 MM월 dd일" // 날짜 형태
    shouldCloseOnSelect // 날짜를 선택하면 datepicker 닫힐지 여부
    minDate={new Date()} // 오늘 날짜 이후로만
    // maxDate={new Date()} // maxDate 이후 날짜
    selected={form.reserveDate}
    onChange={(date) => {
        setForm({...form, reserveDate: date})
        setVisible(true)
    }}
```

```
const checkForm = () => {
    ticketMenu === '' ? alert('예약권을 선택해주세요')
    : sitMenu === '' ? alert('좌석을 선택해주세요')
    : form.sitNum === '' ? alert('좌석 번호를 선택해주세요')
    : form.reserveDate === null ? alert('예약일을 선택해주세요')
    : form.startTime === null ? alert('시작 시간을 선택해주세요')
    : form.endTime === null ? alert('종료 시간을 선택해주세요')
    : form.endTime < form.startTime ? alert('종료 시간을 다시 선택해주세요')
    : setVisible(true)
```

예약정보는 드롭박스별 데이터를 서버에 전송

1. DatePicker 라이브러리를 사용하여 날짜를 지정하였다
2. 유효성 검사를 넣어서 잘못된 날짜, 시간 정보가 입력되지 않도록 설정
3. 각 드롭박스별로 null값을 예방하여 정확한 예약이 되도록 지정하였다
4. 예약진행을 선택하면 최종 결과 팝업이 나타나서 결제진행을 돋는다..

The screenshot shows a reservation booking interface. At the top right is the Dasom Study Cafe logo. Below it is a 'Reservation' section with a '예약하기' button. The main area has several dropdown menus and input fields:

- '예약권 선택': dropdown menu showing '고정석 기간권'.
- '좌석 선택': dropdown menu showing '고정석' and '29'.
- '예약일 선택': dropdown menu showing '2024년 09월 20일'.
- '사용 시작 시간 선택': dropdown menu showing '05:10 PM'.
- '사용 종료 시간 선택': dropdown menu showing '05:20 PM'.
- A large green button at the bottom right says '결제 진행'.

05 front-end (4/6)

OpenAPI (Naver 소셜 로그인)

```
const handleLoginClick = async () => {
  try {
    const response = await fetch('https://zev4wu0r0a.execute-api.ap-northeast-2.amazonaws.com/api/naverlogin',
      { method: 'GET',
        credentials: 'include',
      });
    const data = await response.json();

    if (data.success) {
      const loginWindow = window.open(data.api_url, 'naverLogin', 'width=500,height=600');

      if (!loginWindow) {
        alert('팝업 차단이 활성화되어 있습니다. 팝업을 허용해 주세요.');
        return;
      }
    }
  } catch (error) {
    console.error(error);
  }
}
```

리다이렉트-인증, 토큰-로그인

1. 네이버 로그인 버튼을 클릭 → 네이버로 리다이렉트

* 예제소스 : <https://nid.naver.com/oauth2.0/authorize>

2. 네이버 인증 및 동의 : Naver Developer에서 설정한 인증방식 → 동의 후 백엔드로 리다이렉트

3. 백엔드에서 토큰발급 요청

* 백엔드(Client_id, Client_secret) → 네이버 (access_token, refresh_token) 을 반환

4. 발급된 토큰으로 사용자 프로필 정보 요청

* 백엔드(Authorization: Bearer access_token) → 사용자 정보 반환(id, email, name 등)

5. 사용자 정보전달(백엔드 → 프론트엔드 → 로그인 완료 처리)

소셜 로그인 버튼

N Login with Naver

인증 리다이렉션

개발자의 품격에서 swko0513님의 개인정보에 접근하는 것에 동의하십니까?

제공된 정보는 이용자 식별, 통계, 계정 연동 및 CS 등을 위해 서비스 이용기간 동안 활용/보관됩니다. 기본정보 및 필수 제공 항목은 개발자의 품격서비스를 이용하기 위해 반드시 제공되어야 할 정보입니다.

기본 정보

이용자 고유 식별자

필수 제공 항목

이메일

별명

동의 후에는, 해당 서비스의 이용약관 및 개인정보처리방침에 따라 정보가 관리됩니다.

취소

동의하기

이용 약관 | 개인정보처리방침 | 책임의 한계와 법적고지 | 회원정보 고객센터

NAVER Copyright NAVER Corp. All Rights Reserved.

05 front-end (5/6)

OpenAPI (Kakao Map)

MapPage.js

```
// React 앱이 시작되는 순간
if (!window.kakao) {
  const script = document.createElement('script');
  script.src = "https://dapi.kakao.com/v2/maps/sdk.js?appkey=YOUR_APP_KEY";
  script.onload = initializeMap;
  document.head.appendChild(script);
} else {
  initializeMap();
}
```

SDK를 통한 카카오 API 사용

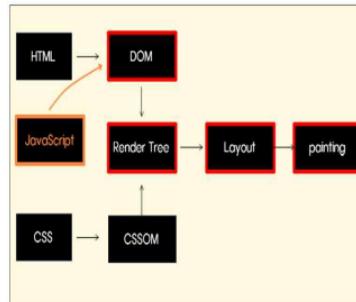
- React 앱이 시작되면 index.html이 브라우저에 실행되고 root로 렌더링된다.(DOM구성)
* 1개의 html을 동적으로 활용하는 SAP이며 SDK(자바스크립트)로 DOM을 업데이트한다.
- index.html의 <script> 태그를 통해 Kakao Map의 SDK가 브라우저에서 실행된다
* SDK는 kakao.maps를 생성한다.
- MapPage.js에서 kaka.maps를 참조하고 카카오 API서버와 통신
- 카카오 개발자모드에서 받은 appkey를 sdk소스에 넣어 인증한다.

index.html

```
<body>
<script type="text/javascript" src="//dapi.kakao.com/v2/maps/sdk.js?appkey=dec9d18948c75dfa5519c67d9d9a171a"></script>
<script type="text/javascript" src="https://static.nid.naver.com/jss/naverLogIn/v1/naverLogIn.js?_v=1.0.5">< charset="utf-8"></script>

<div id="root"></div>
</body>
</html>
```

KakaoMap의 구현방식



오는 길



05 front-end (6/6)

OpenAPI (sms인증)

MapPage.js

```
try{
    const response = await messageService.sendOne({
        to: userPhone,
        from: "<등록 휴대폰 번호>",
        text: `[다솜 스터디 카페] 인증번호\n${verificationCode}입니다`,
    });
    console.log("문자 메시지 전송 성공 :", response);
    res.status(200).json({success:true, message: "문자 메시지가 성공적으로 전송되었습니다"});
} catch (error) {
    res.status(500).json({success:false, message: "문자 메시지 전송 중 오류 발생"});
    console.error("문자 메시지 전송 실패:", error); // 서버 로그에 에러 기록
}
```

coolsms SDK를 통한 sms 전송

1. coolsms.co.kr에서 다른 OpenAPI와 유사하게 API Key와 SDK를 받아서 사용한다.
2. Kakao Map과 다르게 프론트-백엔드를 통해 API를 사용한다
3. 프론트 → POST를 통해 URL ('signUp/sendSMScod')를 통해 백엔드에 sms를 보내는 요청을 보낸다
4. 백엔드 → Math.random() 을 사용해 6자리 인증번호를 생성하고 문자메세지를 전송한다
* coolsms-node-sdk 라이브러리를 사용하여 sms를 전송한다.
5. 프론트 → 인증번호가 일치하면 true를 보낸다.

회원가입시 인증번호 입력

휴대폰 번호 입력 ("-" 제외 11자리 입력)

인증하기

인증번호 발송

[Web발신]
[다솜 스터디 카페] 인증번호
966826입니다
오후 6:47



Dasom
StudyCafe

back-end

다솜
애틋한 사랑의 마음으로



| 비밀번호

| 예약하기

| 마이페이지



다솜의 따뜻한 마음

다솜은 모든 이들의 '따뜻한 애정과 정성'을 지지하고자 만들어졌습니다.
실면서 친정으로 애정을 가지고 무언가에 몰두하는 일이 쉽지 않다는 사실을 알기 때문입니다.
다솜의 역할은 단순히 공부를 위한 공간만을 제공하는 것이 아니라,
따뜻한 분위기와 애정을 통해 여러분이 목표를 향해
흔들림 없이 나아갈 수 있도록 격려하고 돕는 것입니다.
다솜은 여러분이 공부와 자기계발에 대한 사랑을 키우고,
그 열정을 지속할 수 있는 공간입니다.

다솜의 Key Color

초록색은 생명과 성장, 조화를 의미합니다.
푸르른 자연처럼 언제나 싱그럽고 활기찬 에너지를 제공하는 다솜 스터디 카페는
여러분의 꿈과 목표를 지지합니다. 다솜은 따뜻한 애정과 정성으로 여러분이 흔들림 없이
목표를 향해 나아갈 수 있도록 도와드리겠습니다.
초록색의 생기 넘치는 힘처럼, 여러분의 학업과 자기계발의 여정에 활력을 불어넣어 드리겠습니다.

06 back-end (1/3)

비밀번호

비밀번호 암호화

```
const bcrypt = require('bcrypt');
const saltRounds = 10;
const hashedPw = await bcrypt.hash(userPw, saltRounds);
```

bcrypt 라이브러리와 salt 및 hash로 암호화

1. bcrypt로 비밀번호를 해싱하여 원본 비밀번호를 직접저장하지 않도록 함
2. salt는 비밀번호마다 고유한 랜덤 데이터로 saltRound로 10회 실시합니다.
3. hashedPw 변수에 hash된 유저정보를 저장합니다.
4. 로그인시에는 bcrypt.compare()를 통해 입력된 값과 hash에저장된 salt값을 추출하여 비교합니다

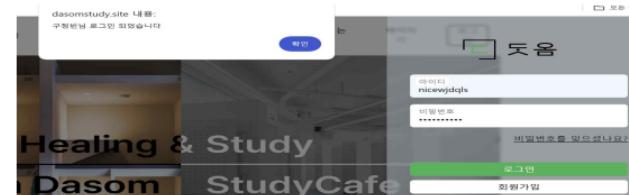
비밀번호 해쉬화

	gabried	nicewjdqsl	\$2a\$10\$gd/vuwlrNFD3zLF2PxKunIMNFJYl5y0c...	이상록
28			\$2a\$10\$WqZPoOlsCU17VT2v1ZfRualEglGrbpa...	정빈

해쉬화 된 유저정보로 로그인

```
const user = results[0];
// user_pw를 userPw로 변경
const match = await bcrypt.compare(userPw, user.userPw);
if (!match) {
  return res.status(400).json({ message: "Invalid user ID or password" });
}
```

일반 비밀번호로 로그인 가능



로그인이 성공하면 세션에 저장

```
req.session.user = { id: userId, name: user.userName };
```



Dasom
StudyCafe

06 back-end (2/3)

예약하기

```
async function reservation(req, res) {
    console.log(req.body);
    try {
        const { userId, sitNum, reserveDate, chargeTime } = req.body;

        // 시작 시간과 종료 시간 계산
        const startDateTime = new Date(reserveDate);
        const endDateTime = new Date(startDateTime);
        endDateTime.setHours(endDateTime.getHours() + chargeTime); // chargeTime이 따라 종료 시간 설정

        // 예약이 겹치는지 확인하는 쿼리
        const checkQuery =
            `SELECT * FROM reservation
             WHERE sitNum = ?
             AND reserveDate = ?
             AND startTime < ? AND endTime > ?`;
        await connection.promise().query(checkQuery, [sitNum, reserveDate, endDateTime, startDateTime]);

        const [rows] = await connection.promise().query(checkQuery, [sitNum, reserveDate, endDateTime, startDateTime]);

        if (rows.length > 0) {
            return res.status(400).json({ success: false, message: "좌석이 이미 예약되어 있습니다." });
        }
    }
```

1. node.js는 스레드를 단일 스레드로 지원하여 동시에 작업이 가능하게 하는 비동기작업을 실행한다.

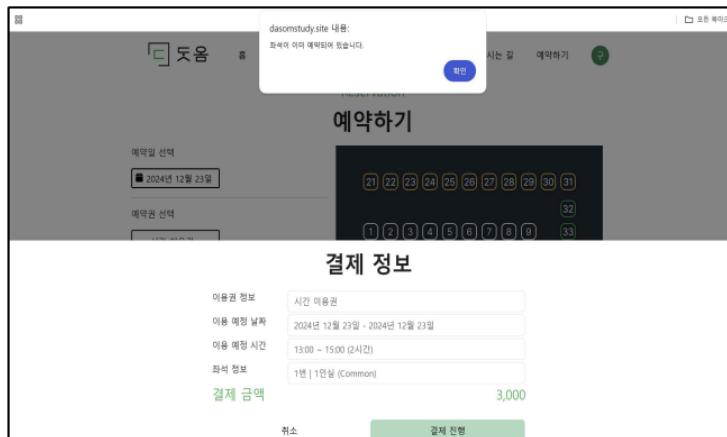
* **async**문을 통해 비동기진행(완료된 순으로 결과값 반환)

2. 예약하기는 조건은 사전에 좌석이 예약이 되어있는지 check하고 쿼리를 진행해야 한다.

* **promise**문으로 **checkQuery**를 진행하고 완료되면 **rows**의 값으로 반환하도록 **await**를 지정한다.

3. **promise**문은 비동기화로 쿼리를 진행하며 pending(대기), Fulfilled(성공), Rejected(실패)로 반환한다.

await문 실행후 좌석중복검사



06 back-end (3/3)

마이페이지

userId와 로그인정보는 localStorage에서 받아온다

```
const isLoggedIn = localStorage.getItem("isLoggedIn");
const userId = localStorage.getItem("id");
```

리액트 → 백엔드 userId값 전송

```
useEffect(() => {
  if (isLoggedIn) {
    api.get(`/mypages/${userId}`)
```

백엔드 함수에서 userId를 받아 쿼리함수 실행

```
async function reserveMyPage(req, res) {
  // URL 파라미터에서 userId 추출
  const { userId } = req.params;
  console.log("Received user ID:", userId); // 디버깅을 위한 콘솔 출력

  try {
    // 비동기로 사용자 정보 가져오기
    const myInfo = await fetchUserInfo(userId);
    console.log(myInfo); // 사용자 정보 출력 (디버깅)
  }
```

```
async function fetchUserInfo(userId) {
  try {
    // SQL 쿼리 정의: 주어진 userId로 사용자의 정보를 조회
    const query = `SELECT userId, userName, userPhone, userBirthDate, created_at
    FROM member
    WHERE userId = ?`;

    const result = await db.query(query, [userId]);
    return result[0];
  }
}
```

백엔드 → 리액트로 JSON으로 응답

```
if (myInfo) {
  // 사용자 정보가 존재하는 경우, 헤더에 상태코드 설정
  return res.status(200).json({
```

→ setUserData(response.data);

특정 userId에 정보를 조회, 삭제, 회원탈퇴가 가능하다



Dasom
StudyCafe



Doker

| 개요

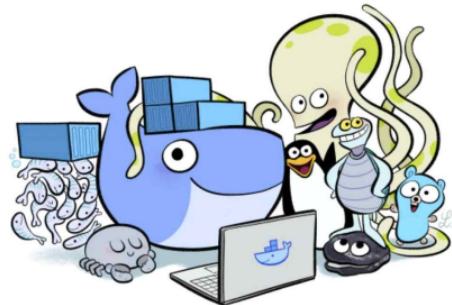
Docker를 활용하여 서비스를 컨테이너화하고, NGINX를 리버스 프록시 및 로드 밸런서로 구성하여 웹 관리와 성능을 최적화한다. 이를 통해 배포를 단순화하고 OS에 독립적인 확장 가능한 환경을 구축한다.

| NGINX

| Docker Compose

| Docker hub

| Prometheus / Grafana



07 Docker(1/7)



Docker 환경구성



NGINX 구축목적

개발환경 (NGINX 없음)



1. NGINX 구성이 없어 사용자의 요청은 직접 웹서버에 전달되어 서버 부담을 가중한다.
2. 단일 웹서버 구성은 장애 발생 시 서비스 가용성에 치명적이다.
3. 응답 시간이 느려진다.
4. 서버 고장나면 전체 시스템에 문제가 발생한다.

Reverse Proxy로서 사용



07 Docker(2/7)

nginx.conf 작성

로드밸런싱 upstream 설정

backend(3):5000, frontend(1):3000

```
events {
    worker_connections 1024;
}

http {
    upstream backend_servers {
        server backend1:5000;
        server backend2:5000;
        server backend3:5000;
    }

    upstream react_servers {
        server frontend:3000;
    }
}
```

독립적인 트래픽 분산



1. 서로 다른 컨테이너에서 독립적으로 실행하여 부하를 분산시킨다.
* 컨테이너의 자체 내부 IP가 다르기 때문 (기존 gateway에서 1개씩 IP 추가!)
2. 서버를 컨테이너화 하여 NGINX로 로드밸런싱을 하기 위한 설정이다.

07 Docker(3/7)



nginx.conf 작성

NGINX의 웹 설정

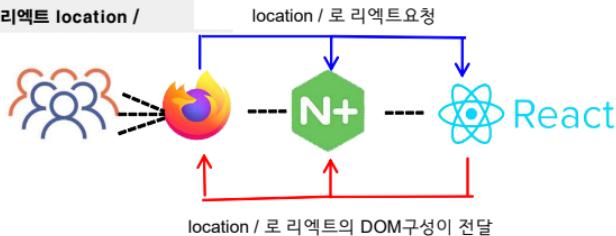
리액트와 백엔드에 요청 값 전달

```
server {  
    listen 80;  
  
    location / {  
        proxy_pass http://react_servers;  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
        proxy_set_header X-Forwarded-Proto $scheme;  
    }  
  
    location /api {  
        proxy_pass http://backend_servers;  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
        proxy_set_header X-Forwarded-Proto $scheme;  
    }  
}
```

react

backend

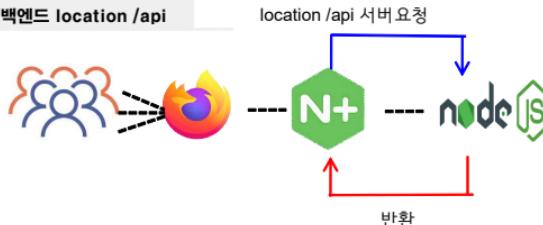
리액트 location /



location / 로 리액트요청

location /로 리액트의 DOM구성이 전달

백엔드 location /api



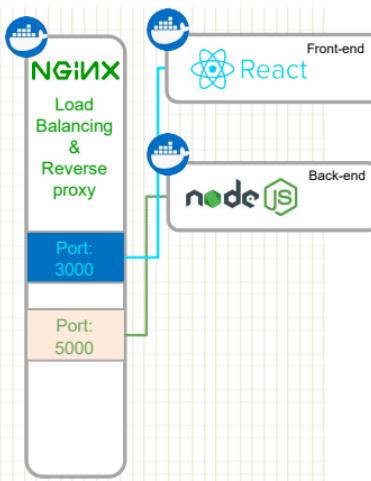
location /api 서버요청

반환

1. 리액트와 백엔드에 location 요청에 따라 NGINX가 데이터를 전달한다.
2. proxy_pass 문을 활용해 nginx 포트 80에 들어온 데이터를 해당 주소에 재할당한다.
3. 데이터는 헤더에 저장되어 전송된다.

07 Docker(4/7)

Docker compose 구성



멀티컨테이너 관리

```
networks:  
  frontend:  
  backend:  
  
version: '3.7'  
  
services:  
  nginx:  
    build:  
      context: ./nginx  
    ports:  
      - "80:80"  
    networks:  
      - frontend  
      - backend  
    depends_on:  
      - backend  
  
  react:  
    build: ./react-app  
    volumes:  
      - ./react-app:/app  
      /app/node_modules  
    networks:  
      - frontend  
    ports:  
      - "3000:3000"  
    deploy:  
      restart_policy:  
        condition: on-failure  
  
  backend:  
    build: ./node-app  
    volumes:  
      - ./node-app:/app  
      /app/node_modules  
    networks:  
      - backend  
    depends_on:  
      - db  
    ports:  
      - "5000:5000"  
    deploy:  
      restart_policy:  
        condition: on-failure
```

networks : 네트워크는 브릿지 형식으로 통신

NGINX : 포트 80번으로 React와 Node.js통신

React : 포트 3000번으로 NGINX 통신

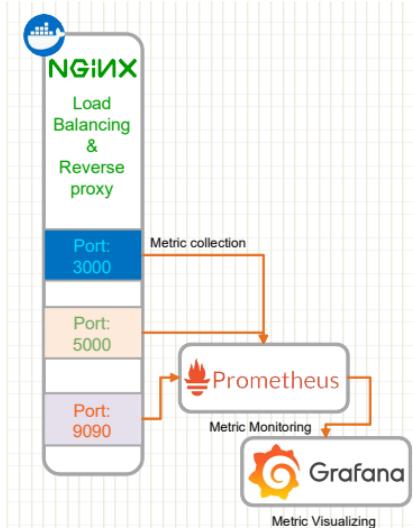
Node.js : 포트 5000번으로 NGINX 통신



07 Docker(5/7)



멀티컨테이너 관리



```
grafana:
  image: grafana/grafana:latest
  ports:
  - "3001:3000"
  environment:
  - GF_SECURITY_ADMIN_PASSWORD=P@ssw0rd
  networks:
  - backend
  depends_on:
  - prometheus

prometheus:
  build:
    context: ./prometheus
  ports:
  - "9090:9090"
  networks:
  - backend
  - frontend
  volumes:
  - prometheus data:/prometheus
  restart: always

nginx-exporter:
  image: nginx/nginx-prometheus-exporter:latest
  ports:
  - "9113:9113"
  networks:
  - backend
  environment:
  NGINX_HOST: nginx
  NGINX_PORT: 80
  depends_on:
  - nginx
```

Grafana: 호스트의 3001포트를 컨테이너 3000과 연결

Prometheus : 호스트의 9090포트를 컨테이너 9090과 연결

Nginx-exporter : 9113포트로 컨테이너 9113과 연결

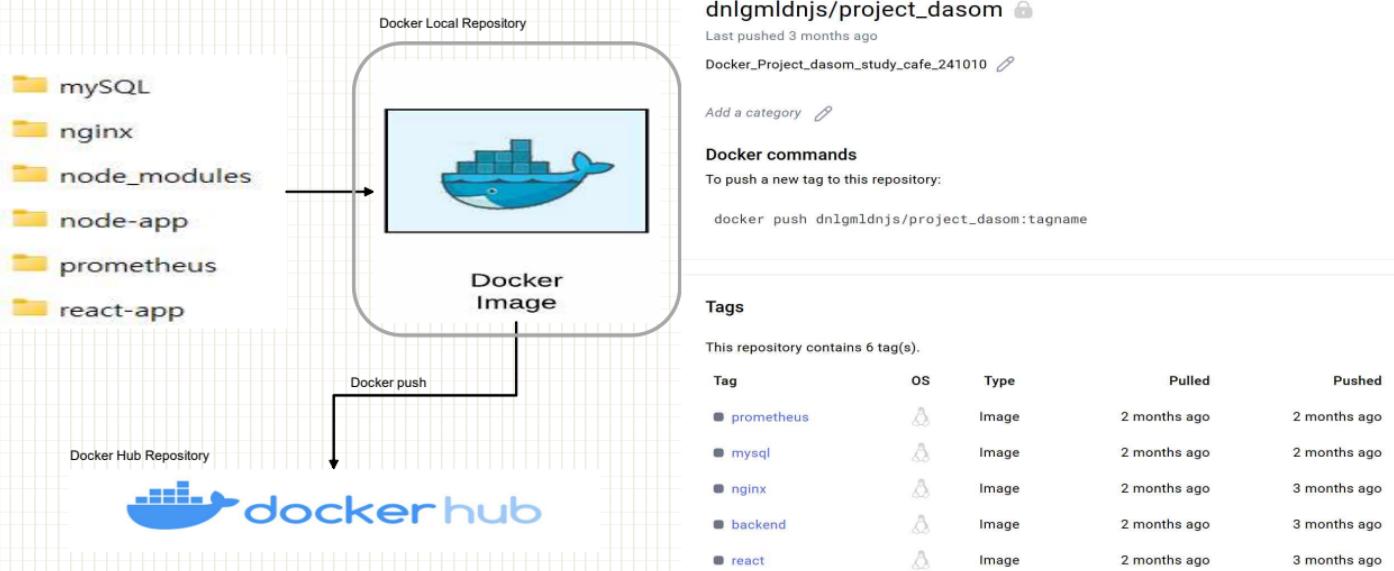


07 Docker(6/7)



Dasom
StudyCafe

Docker hub로 이미지 배포



07 Docker(7/7)

프로메테우스 9090접속

Targets

All scrape pools		All	Unhealthy	Collapse All	Filter by endpoint or labels
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9113/metrics	UP	instance="localhost:9113" job="nginx"	18.796s ago	28.895ms	
prometheus (1/1 up) show less					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	13.218s ago	19.576ms	

그라파나 시각화



1. Prometheus가 Nginx의 lr을 수집하고 CPU 사용량을 수집한다
2. Grafana를 사용해 시간에 따른 트래픽량을 그래프로 시각화 한다.

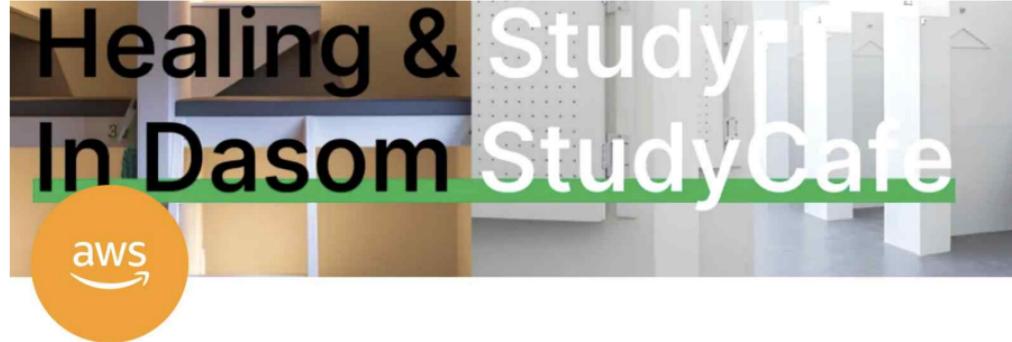


Dasom
StudyCafe



Dasom
StudyCafe

AWS



| 아키텍처

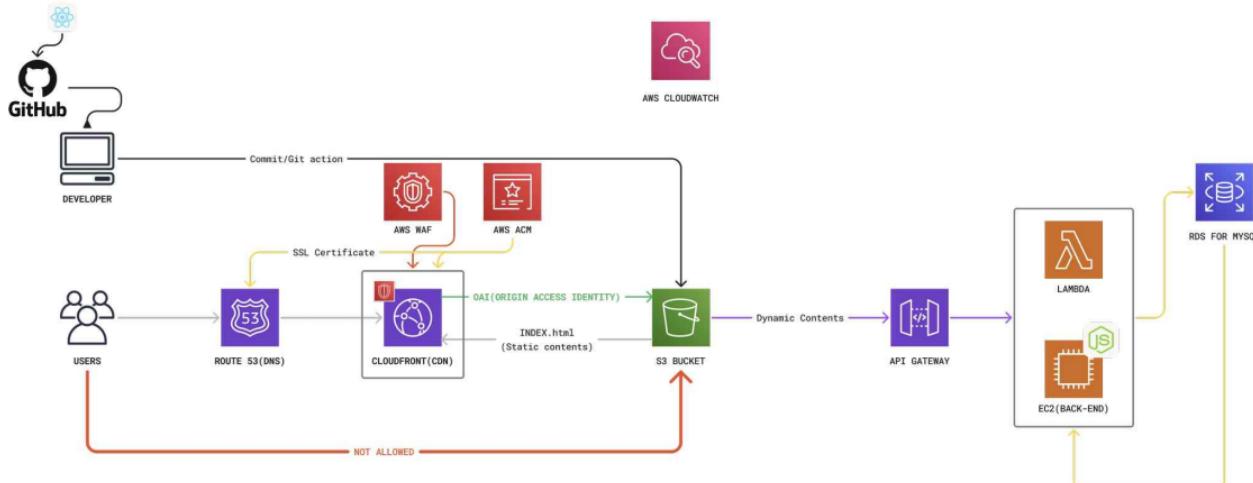
| 구축 방식

08 AWS(1/7)



Dasom
StudyCafe

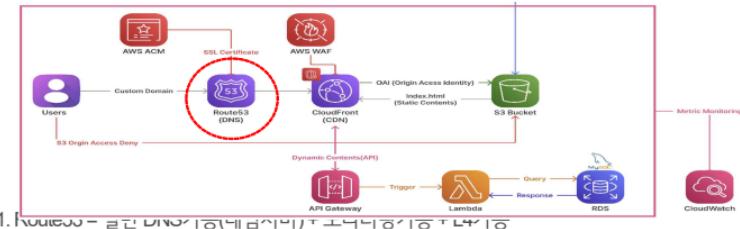
아키텍처



08 AWS(2/7)

Route 53

아키텍처



Route53 기능

항목	내용
Resolver	도메인 이름과 IP 주소를 상호 변환
라운드Robin	고가용성
트래픽 흐름	서버의 가동여부를 체크



Route 53

Public dasomstudy.site Info

Delete zone Test record Configure query logging

Hosted zone details

Records (4) DNSSEC signing Hosted zone tags (0)

Records (4) Info

Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

Record ...	Type	Routin...	Differ...	Alias	Value/Route traffic to	TTL (s...)	Health ...	Evaluat...
	A	Simple	-	Yes	d2d93bn8vp9f.cloudfront.net.	-	-	No
	NS	Simple	-	No	ns-116.awsdns-14.com. ns-1247.awsdns-27.org. ns-980.awsdns-58.net. ns-1788.awsdns-51.co.uk.	172800	-	-
	SOA	Simple	-	No	ns-116.awsdns-14.com.awsd...	900	-	-
	CNAME	Simple	-	No	_4cf8afdfa1c4a59ac0b4cc9d...	300	-	-

gabia.

08 AWS(3/7)

CloudFront의 ACM,WAF 구축

CloudFront Global Content Delivery Network

Settings

Description: 옛지 로케이션에 캐싱

Alternate domain names: dasomstudy.site

Custom SSL certificate: **Custom SSL certificate** (dasomstudy.site) (highlighted with a red box)

Standard logging: Off

Cookie logging: Off

Security policy: TLSv1.2_2021

Default root object: index.html

Price class: Use all edge locations (best performance)

Supported HTTP versions: HTTP/2.0, HTTP/1.1, HTTP/1.0

아키텍처(CloudFront)

The diagram illustrates the CloudFront architecture. A Client connects via HTTPS to Amazon Route 53, which then routes traffic through Amazon CloudFront. CloudFront integrates with AWS Certificate Manager (ACM) for SSL termination. The traffic then passes through a Web Application Firewall (WAF) before reaching the Origin (Amazon Elastic Compute Cloud - Amazon EC2). The origin returns content via HTTP back to CloudFront, which then forwards it to the client. A screenshot of a browser shows a successful HTTPS connection to the website.

- 클라우드 요청 (<http://dasomstudy.site>) → DNS를 통해 Route53에서 DNS 설정을 통해 CloudFront로 트래픽이 전달
- CloudFront에서 HTTPS 처리 : ACM 인증서
- CloudFront ↔ S3

ACM (AWS ACM)

Certificate status: Issued (2024-01-09T09:53:00Z-dasomstudy)

Domains: dasomstudy.site (Owner), dasomstudy.com (Owner)

Associated resources: CloudFront와 ACM 인증서 연결 : Custom SSL Certificate 옵션에 연결

CloudFront에서 WAF 구축

AWS WAF

CloudFront WAF Rule Groups

CloudFront geographic restrictions

Security trends for the specified time range

Top attack types

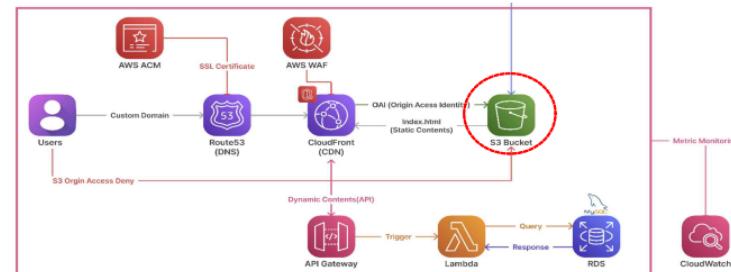
Top countries

This section shows the configuration of CloudFront WAF rules and monitoring of security trends and attacks.

08 AWS(4/7)

S3 버킷

사용목적



SSE-S3 암호화 키 사용

Default encryption Info

Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type Info

Server-side encryption with Amazon S3 managed keys (SSE-S3)

Bucket Key

When KMS encryption is used to encrypt new objects in this bucket, the bucket key reduces encryption costs by lowering calls to AWS KMS. [Learn more](#)

- 데이터 업로드시 S3가 자체적으로 키생성
- S3에서 키를 사용해 데이터를 복호화 후 사용자에게 전송
- OAI를 통해 S3와 CloudFront를 통해 전송

정적 웹 사이트 호스팅

정적 웹 사이트 호스팅

이 버킷을 사용하여 웹 사이트를 호스팅하거나 요청을 리다렉션하십시오. [자세히 알아보기](#)

정적 웹 사이트

- 비활성화
- 활성화

호스팅 유형

- 정적 웹 사이트 호스팅
이미징에 대한 요청은 '정적'으로 사용합니다. [자세히 알아보기](#)
- 캐시에 대한 요청 리다렉션
요청을 다른 버킷 또는 도메인으로 리다렉션합니다. [자세히 알아보기](#)

① 고객이 웹 사이트 앤드포인트의 콘텐츠에 액세스할 수 있게 하려면 모든 콘텐츠를 공개적으로 놓기 가능하도록 설정해야 합니다. 이렇게 하려면, 버킷에 대한 S3 피클리 액세스 차단 설정을 편집하여 됩니다. 자세한 내용은 [Amazon S3 피클리 액세스 차단 사용](#)을 참조하십시오.

S3 버킷 앤드포인트 접근 시 기본 index.html로 설정해 웹사이트의 진입점을 정의

파일과 문서

파일 사이트의 웹 페이지 또는 기본 페이지를 지정합니다.

index.html

모듈 문서 - 선택 사항

모듈이 발생하면 반환됩니다.

error.html

버킷의 End-Point로 접근하였을 때 기본적으로 표시될 인덱스 문서를 지정해야 함(index.html)
모듈 문서의 경우 따로 설정하지 않았을 경우 기본값을 유지한다.

JSON Ln 1, Col 1 오류: 0 경고: 0

주소

변경 사항 저장

URL 정적 웹사이트

정적 웹 사이트 호스팅

이 버킷의 서비스에 대한 서비스를 찾았거나 요청을 처리했습니다. [자세히 알아보기](#)

정적 웹 사이트 호스팅은 AWS Amplify Hosting 사용하는 것이 좋습니다.

② 정적 웹 사이트 호스팅은 AWS Amplify Hosting 사용하여 빠르고 안전하게 만족적인 웹 사이트를 빠르게 배포하세요. [Amplify Hosting](#)에 대해 자세히 알아보기

Amplify 앱 생성

도록 정적 웹 사이트 호스팅

활성화

포스팅 활성화

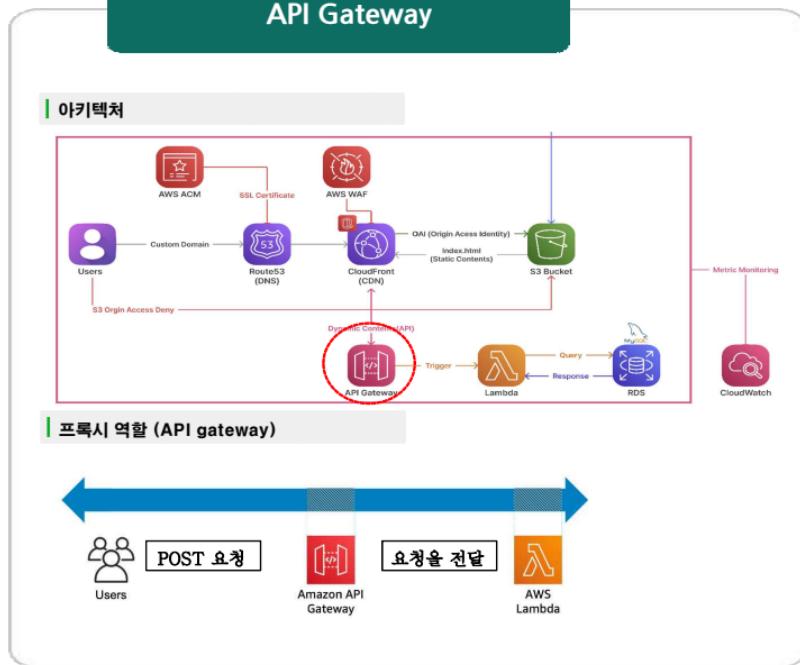
비밀 웹 사이트 앤드포인트

2022-03-22 10:00:00 UTC+00:00 2022-03-22 10:00:00 UTC+00:00 2022-03-22 10:00:00 UTC+00:00 2022-03-22 10:00:00 UTC+00:00

<https://amazonaws.com>

08 AWS(5/7)

API Gateway



정적 웹 사이트 호스팅

Routes

Routes for Dasom_Project_API_GW

- /ChangePassword
- /DeleteReservation
- /FindPassword
- /ReturnUserInfo
- /admin_page
- /adminpages
- /callback
- /DeleteAccount
- /Login

Route details

ANY /ChangePassword [ID: zfemstp]

ARN

arn:aws:apigateway:ap-northeast-2:/apis/zev4w0G0/routes/zfemstp

Authorization

Authorizes protect your API against unauthorized requests. Routes with no authorization attached are open.

No authorizer attached to this route. [Attach authorization](#)

Integration

The integration is the backend resource that this route calls when it receives a request.

select [Configure](#)

08 AWS(6/7)

서비스의 Lambda로 코드수정

요청/응답 객체 대신 event와 context 사용



1. Lambda는 서비스로 동작하며 자체적으로 HTTP 요청을 수신하는 서버를 운영하지 않는다.

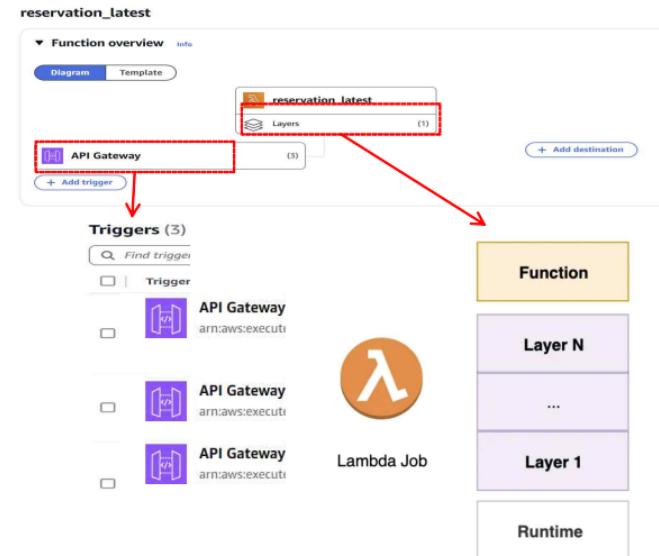
* node.js는 express로 HTTP와 통신하지만 Lambda는 단일 함수로 실행

2. Lambda 함수는 직접 HTTP 응답을 다룰 수 없어 JSON형태로 응답한다.

3. API Gateway가 HTTP 요청을 Lambda 호출로 변환하는 중개 역할을 담당한다.

* API Gateway에서 요청을 순차적으로 트래픽을 완화시킴

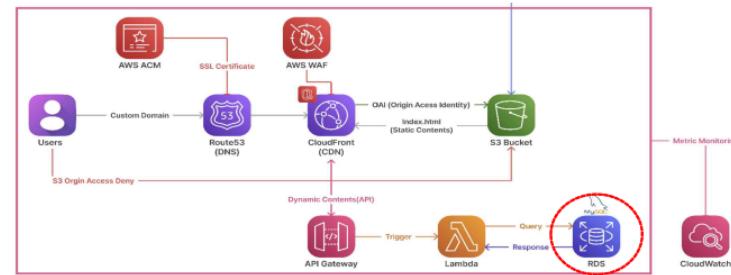
Layer로 구성된 Lambda



08 AWS(7/7)

RDS

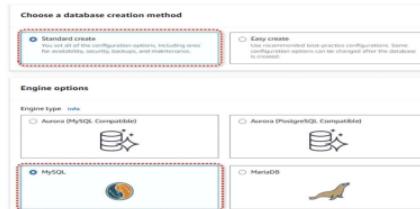
아키텍처



RDS 구축개념

1. RDS에서 DBMS로 MySQL을 사용하였다.
2. EC2에 직접연결하지 않고 Workbench로 연결이 가능하도록 설정
3. Default VPC를 통해 기본 VPC로 통신할 수 있도록 설정
4. Public Access를 통해 RDS 인스턴스가 IP 주소에 할당하여 인터넷 접근이 가능하도록 설정

정적 웹 사이트 호스팅



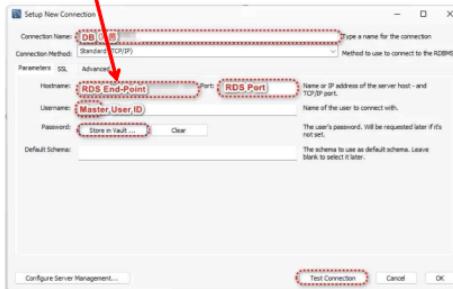
RDS

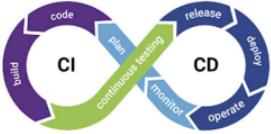
Connectivity & security

Endpoint & port

Endpoint

Port
3306





CI/CD

개요

GitHub Actions와 Amazon S3를 활용하여 코드 변경 사항을 자동으로 빌드하고 테스트하여 소프트웨어 품질을 향상시키고 지속적으로 배포하는 프로세스를 구축한다.

운용 목적

1. 자동화된 배포:

- GitHub Actions를 활용하여 코드 변경 사항이 자동으로 S3에 배포되도록 구현.

2. 버전 관리:

- Git의 커밋과 푸시 작업을 통해 코드 변경 이력을 체계적으로 기록 및 관리.

3. 보안 강화:

- AWS 액세스 키를 GitHub Secrets에 저장하여 민감한 정보를 안전하게 보호.

4. 효율성 및 반복성:

- 워크플로우를 자동화하여 배포 과정에서 수작업을 제거하고 신뢰성 증대.

Git-Action key 밝브

yml 작성

동작확인

09 CI/CD(1/2)

S3와 Git-Action 설정

AWS IAM Access Key 발급

Access keys (1)
One secret key is needed to make programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of three access keys active at one time. Learn more

Description	Status	Actions
aws	Active	... Actions

1. AWS 리소스에 접근하기 위해 인증이 필요하며 Access Key 발급이 필요하다
2. S3 버킷에 대한 쓰기 권한을 부여

리파지토리 저장상태

New repository secret

Name	Last updated	Actions
AWS_ACCESS_KEY_ID	yesterday	edit
AWS_SECRET_ACCESS_KEY	17 hours ago	edit

1. Access Key는 GitHub Repository의 Secrets에 저장한다.
2. 워크플로우 파일(yml)에서 참조할 수 있다.

yml 파일 작성

1. .github/workflows에 yml파일을 구축한다

```
name: React build
on:
  push:
    branches:
      - main
```

- "main" 브랜치에 코드 변경 사항이 발생하면 워크플로우가 실행됨

```
name: Install Dependencies
on:
  push:
    branches:
      - main
    steps:
      - name: Set Legacy-peer-deps-trace
        run: npm install --legacy-peer-deps=true
      - name: Build
        run: |
          rm -rf node_modules
          CI=false npm run build
```

- 애플리케이션 의존성 설치 필요한 경우 "npm install"로 패키지를 설치
- 애플리케이션 빌드, 테스트 모드가 아님을 명시

```
name: Check AWS Credentials
on:
  push:
    branches:
      - main
    steps:
      - if: ${{ secrets.AWS_ACCESS_KEY_ID == '' }}
        run: echo "Error: AWS_ACCESS_KEY_ID is not set. Please set it in your GitHub Secrets"
      - else:
        run: echo "AWS_ACCESS_KEY_ID is set."
      - end
```

- GitHub Secrets에 AWS 자격 증명이 설정되어 있는지 확인
- 자격 증명이 설정되어 있지 않은 경우 에러 메시지를 출력하고 워크플로우를 중단

```
name: Deploy to S3
on:
  push:
    branches:
      - main
    steps:
      - if: ${{ secrets.AWS_ACCESS_KEY_ID == '' || secrets.AWS_SECRET_ACCESS_KEY == '' || secrets.AWS_SECRET_ACCESS_KEY == ''}}
        run: echo "Error: AWS_ACCESS_KEY_ID or AWS_SECRET_ACCESS_KEY or AWS_SECRET_ACCESS_KEY is not set."
      - else:
        run: aws s3 cp --recursive --region ap-northeast-2 build s3://mason-project-1
```

- 빌드 결과물을 S3 버킷에 업로드
- AWS 자격 증명 정보를 환경 변수로 전달하여 AWS CLI에서 접근할 수 있도록 설정
- 디렉토리의 모든 파일을 S3 버킷으로 업로드

- yml 파일 준비 : main 브랜치에 코드 변경이 발생하면 실행된다.

- 빌드작업 : 의존성 및 빌드 작업을 수행

- S3에 업로드 : S3에 빌드 파일을 업로드

09 CI/CD(2/2)

S3와 Git-Action 설정

push 후 Git-Action 등록 확인

All Workflows

All Workflows

Showing runs from all workflows

25 workflow runs

Event	Status	Branch	Actor
관리자 페이지	succeeded	last month	1m 26s
Update SignUpPage.js	succeeded	last month	1m 8s
Update SignUpPage.js	succeeded	last month	55s
Update SignUpPage.js	succeeded	last month	56s
Update main.yml	succeeded	last month	55s

정상 빌드 확인

Summary

Jobs

build

Annotations

1 warning

Run details

Usage

Workflow file

build

succeeded on Oct 30 in 59s

Search logs

Step	Duration
Set up job	1s
Checkout source code	2s
Cache node modules	3s
Install Dependencies	5s
Build	23s
List build directory	6s
Check AWS Credentials	6s
Install AWS CLI	13s
Deploy to S3	7s
Post Cache node modules	3s
Post Checkout source code	6s
Complete job	6s



Dasom
StudyCafe

멘토링

| 비밀번호

| 예약하기

| 마이페이지



1. 피드백 사항 및 보완 계획 | 피드백 사항

1) Backend - ORM(Object-relational Mapping) 적용



- 피드백: ORM을 구축하여 SQL 쿼리를 오브젝트 단위로 쪼개어 별도 관리하고, 코드로 간단하게 호출할 수 있도록 백엔드 로직 변환
- 객체-DB간의 존성을 배제
- Node.js의 ORM 패키지인 **sequelize**을 사용하여 ORM 적용(완료)
- 람다 레이어로 패키징(완료)

2) Frontend - TypeScript 변환(점진적)



- 피드백: 안정적인 개발과 런타임 오류를 방지하기 위해 기존 React 코드에 **TypeScript** 적용
- 타입을 명시하여 컴파일 단계에서 사전 에러를 발견할 수 있음
- 기존의 js파일로 작성된 리액트 앱을 타입스크립트(TypeScript)로 점진적 변환(진행중)
- 메인(App), 로그인, 로그아웃 기능에 타입스크립트 적용됨

3) Lambda CI/CD With Git-Action



- Front-End 역할을 담당하는 S3 – CloudFront의 Git-Action을 통한 CI/CD는 구현완료
- Back-End 역할을 담당하는 API-Gateway – Lambda의 경우 Console에서 작업하여야 하는데
이에 발생하는 작업상 발생하는 불편함을 해소하기 위함.

2. Backend - ORM(Object-relational Mapping) 적용

1) 모델(model) 생성 및 정의

```
└─ models
    └─ index.js
    └─ login_history.js
    └─ member.js
    └─ reservation.js
    └─ snsmember.js
```

- db설정 정보를 기반으로 db와 연결할 sequelize 객체를 생성하는 파일 작성
- 동일 디렉토리(models) 내의 모든 파일(우리가 작성할 모델파일)을 불러와 db 객체에 맵핑

```
login_history.js
```



```
module.exports = (sequelize, DataTypes) => {
  const LoginHistory = sequelize.define(
    'LoginHistory',
    {
      history_number: {
        type: DataTypes.INTEGER,
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
      },
      user_id: {
        type: DataTypes.STRING(255),
        allowNull: false,
      },
      ip_address: {
        type: DataTypes.STRING(45),
        allowNull: true,
      },
      history: {
        type: DataTypes.DATE,
        allowNull: true,
        defaultValue: sequelize.literal('CURRENT_TIMESTAMP'),
      },
    },
    {
      tableName: 'login_history',
      freezeTableName: true,
      timestamps: false, // "history" 테이블에 타임스탬프 역할을 하므로 "createdAt",
    }
  );
  return LoginHistory;
};
```

- 테이블을 create할 때 스키마를 작성하는 방법과 유사하게 컬럼 이름과 데이터 형식, 기본키OR외래키 등의 여부, NULL 허용 여부 등을 정의
- **freezeTableName: true** 부분의 경우, Sequelize는 기본적으로 모델 이름을 복수형으로 변환하여 테이블 명을 자동 설정하므로(member 모델을 members로) 이를 방지하기 위해 추가

2. Backend - ORM(Object-relational Mapping) 적용

2) 람다 레이어로

패키징



- 작성된 모델들을 기반으로 람다 레이어로 패키징

3) 람다 함수 로직

변환

```
const Sequelize = require('/opt/nodejs/node_modules/sequelize');
const db = require('/opt/nodejs/models'); // 모델 파일 로드
const config = require('/opt/nodejs/config/config.json'); // config 파일 로드

const sequelize = new Sequelize(config.database, config.username, config.password, {
  host: config.host,
  dialect: 'mysql',
});

const member = await db.Member.findOne({ where: { userId: userId } });

// ... (remaining code)
```

- Sequelize를 import한 후 Sequelize 객체를 생성하고, 기존의 쿼리문을 변경

2. Frontend - TypeScript 변환(점진적)

1) tsconfig.json 파일

```
{  
  "compilerOptions": {  
    "target": "es5",  
    "lib": ["dom", "dom.iterable", "esnext"],  
    "allowJs": true,  
    "skipLibCheck": true,  
    "esModuleInterop": true,  
    "allowSyntheticDefaultImports": true,  
    "strict": true,  
    "forceConsistentCasingInFileNames": true,  
    "module": "esnext",  
    "moduleResolution": "node",  
    "resolveJsonModule": true,  
    "isolatedModules": true,  
    "noEmit": true,  
    "jsx": "react-jsx"  
  },  
  "include": ["src"]  
}
```

- allowJs를 true로 설정해 기존의 js파일도 함께 사용할 수 있도록 설정
- 기존의 파일 확장자를 .tsx로 변경

2) 타입 지정(useState 및 props)

```
const [isLoggedIn, setIsLoggedIn] = useState<boolean>(false);  
  
interface LogoutHandlerProps {  
  | onLogout: () => void;  
}  
  
const ProfileBarPage: React.FC<LogoutHandlerProps> = (props) => {
```

- props로 넘겨줄 메소드의 타입과 파라미터 또한 타입을 지정
- 자식 컴포넌트에서 메소드를 상속받을 때 interface를 정의하여 사용

3. 프로젝트 수행 절차 및 방법

구분	기간	활동	비고
사전 기획	2024.07.01 ~ 2024.08.05	- 프로젝트 기획 및 주제 선정 - 기획안 작성 - 아키텍처 설계	아이디어 선정
UI/UX 구성 및 ERD 설계	2024.08.06 ~ 2024.08.19	- Figma를 사용한 UI/UX 구성 및 설계 - MySQL 기반 관계형 데이터베이스 구조 설계 - React 기반 초기 인터페이스 설계 및 구현	UI/UX, ERD, 인터페이스 설계
1차 프로젝트 (온프레미스)	2024.08.20 ~ 2024.09.19	- Node.js 기반 Express 서버 구축 및 미들웨어 구성 - 로그인/회원가입 구현 - 아이디/비밀번호 찾기 구현 - 예약 페이지 구성 및 마이페이지 구현 - Kakao Map 기반 지도 API 및 갤러리 페이지 구현	웹 애플리케이션 기능 구현 및 서비스 개발
2차 프로젝트 (Docker)	2024.09.20 ~ 2024.10.11	- Docker를 통한 개발 환경 및 아키텍처 설계 - Docker Hub를 통한 컨테이너 이미지 공유 환경 구성 - Docker 환경 설정 및 프론트엔드/백엔드 워크플로우 파일 작성 - 빌드 및 테스트 - Docker 이미지 빌드 및 배포	Docker를 활용한 통합 개발 환경 구축

3. 프로젝트 수행 절차 및 방법

구분	기간	활동	비고
3차 프로젝트 (CI/CD&AWS)	2024.10.12 ~ 2024.11.29	<ul style="list-style-type: none">- CI/CD 아키텍처 설계 및 AWS 인프라 구축- Github Actions를 통한 프론트엔드 자동화 파이프라인 구축- S3 정적페이지 호스팅 및 Route53을 통한 사용자 지정 도메인 적용- API Gateway 및 Lambda로의 기능 구현- ORM을 활용한 AWS RDS 기반 백엔드 코드 모듈화- Github Actions를 통한 백엔드 자동화 파이프라인 설계	CI/CD 및 AWS로 마이그레이션
총 개발 기간	2024.07.01 ~ 2024.11.29 (총 151일)		프로젝트 완료, 지속적인 개선 중



다솜

애틋한 사랑의 마음으로