중간보고서

■ 프로젝트명: MatchMinds

■ 팀명: IAM4U

■ EKS 인프라 운영 / 배포 자동화 / 알림 시스템 구성

■ 담당자 : 조건호

1. EKS 클러스터 및 CI/CD 구축

- EKS 클러스터 구성: VPC 및 노드 그룹 구성, match-backend / match-api 서비스 분리 배포

- CI/CD: GitHub Actions → Docker 이미지 빌드 → ECR 푸시 → ArgoCD 자동 배포

2. ArgoCD 기반 GitOps 배포 자동화

- match-api / match-backend 앱 별도 등록

- 정확한 Git repo path 기준 등록: path: terraform/backend/match-api/k8s

- 상태: Health = Healthy, Sync = Synced

3. 실시간 알림 시스템 구성 (CloudWatch → SNS → Slack)

- CloudWatch Alarm: CPU 사용률 80% 초과 감지

- SNS Topic: match-api-alarm-topic

- Slack Webhook 연동: 알람 메시지 실시간 수신

4. 테스트 결과

- EKS 내 stress-ng 사용 CPU 부하 테스트

- 알람 발생 → SNS → Slack 알림 확인 완료

- 시스템 동작 정상 확인

5. 향후 계획

- Grafana + Prometheus 대시보드 구성

- SNS → Lambda 연동 통한 알림 포맷 개선

- CloudWatch Insights 로그 분석 쿼리 구성

■ 실시간 채팅 시스템 및 로그 수집 / 로그 파이프라인 / Azure 연동 담당

■ 담당자: 김종명

1. CloudWatch Logs 기반 실시간 로그 수집 구성

- 로그 그룹: /matchminds/chat 수동 생성

- 로그 이벤트: Lambda 테스트용 메시지 직접 작성 및 수동 업로드 진행

- Subscription Filter: CloudWatch → Lambda 연결 구성

문제점: Terraform으로 자동 생성 시 권한 전파 지연 이슈로 오류 발생

대응: 콘솔 수동 생성 후 terraform import로 상태 관리 일원화

현재 상태: 정상 구독 필터 작동 확인 완료

2. Lambda 함수 구성 및 로그 처리 로직 구현

- 런타임: Node.js 18.x

- 기능: CloudWatch 로그 압축 해제(gzip + base64), 로그를 파싱하여 S3 + DynamoDB에 각각 저장

- 기능 개선:

1) 필드 누락 및 JSON 파싱 실패에 대한 에러 핸들링 추가

2) DynamoDB 저장 중복 방지 로직 추가 (ConditionExpression 사용)

3) 로그 저장 스키마 표준화 (user, message, timestamp)

3. 로그 저장 대상 구성

- S3 버킷: matchminds-log-archive

- 경로 구조: logs/YYYY-MM-DD/{uuid}.json, 저장 여부 및 파일 형식 확인 완료

- DynamoDB 테이블: UserMatches (이용민 팀원이 생성), UserID + Timestamp 기반 고유 항목 저장 및 중복 방지 및 저장 여부 조건 검사 적용

4. Terraform 기반 자동화

- 리소스 생성 자동화: CloudWatch 로그 그룹, Lambda 함수, IAM 역할/정책, S3 버킷

- 구독 필터 자동화 문제 대응: null_resource + depends_on 사용해 배포 지연 최소화 시도

여전히 Terraform apply 시 필터 등록 실패 가능성 존재 → 콘솔 수동 등록 + import로 절충 처리

5. 테스트 결과

1) 테스트 로그 수동 생성 → Lambda 호출 정상 확인

2) S3 객체 저장 정상

3) DynamoDB 항목 추가 정상 (중복 시 저장 거부 확인됨)

4) Lambda 로그 콘솔 및 에러 로깅 정상 동작 확인

6. 향후 계획

1) Athena 기반 S3 로그 쿼리 테이블 구성 (진행 중)

2) Grafana Cloud 연결 및 Athena를 통한 시각화 대시보드 구성

3) Content Moderator, SignalR, 이미지 분석 파이프라인 확장 연결 예정

■ 인증 / DB / 인프라 초기 구성

■ 담당자: 이용민

1. AWS Cognito 사용자 풀 구성 + OAuth 연동

- 사용자 풀 이름: datingapp-user-pool

- 앱 클라이언트 생성: frontend-client-no-secret (ALLOW_AUTH_CODE_GRANT)

- Redirect URI: http://localhost:3000/callback

- 도메인: https://datingapp.auth.ap-northeast-2.amazoncognito.com

- Route53, 인증서, 커스텀 도메인 연동 (예: www.datingapp.store)

- id_token, access_token, refresh_token 수신 확인 완료

2. React 기반 인증 흐름 구현

- React 프로젝트 생성 (create-react-app)

- react-router-dom으로 /callback 경로 구현

- Authorization Code → Token 교환 구현 (fetch + URLSearchParams)

- localStorage 저장 및 jwt-decode로 토큰 디코딩

- 콘솔 로그로 토큰 및 유저 정보 확인

3. Terraform으로 AWS 인프라 구성

- 디렉토리: envs/dev, modules/, global/provider.tf

- 모듈 구성: vpc, rds, dynamodb, iam, route53, cognito

- S3 backend, terraform.tfvars, outputs.tf 설정 완료

4. Aurora Serverless (RDS) + DynamoDB 구성

- RDS: VPC 연동, 서브넷, 보안 그룹 포함

- DynamoDB: 테이블 스키마 설계 및 구성

5. 테스트 결과

- 로그인 페이지 정상 노출

- 이메일 인증 완료 후 Authorization Code 발급 확인

- 토큰 3종 모두 수신 성공

- /callback 경로 진입 시 code 파라미터 수신 확인

- fetch로 토큰 교환 성공

- 디코딩한 id_token에서 유저 정보 확인

- localStorage 저장 값 확인 완료

- `terraform init`, `terraform apply` 모두 정상 작동

- 리소스들 AWS 콘솔에서 확인됨 (VPC, RDS, IAM Role 등)

- Cognito 설정 반영된 것도 확인 완료

- Aurora Serverless RDS 생성 후 콘솔에서 상태 확인됨

- RDS → VPC, SubnetGroup 연결 확인

- DynamoDB 테이블 정상 생성됨 (설정한 파티션 키/정렬 키 확인)

- 도메인 SSL 인증서 발급 성공

6. 향후 계획

- 토큰 서명 검증 (JWT Signature Validation)

- 토큰 유효성 체크 (프론트 / 백 둘 다 가능)

- Access Token 저장 위치 신중히 선택

- Refresh Token 보안 관리

- HTTPS 강제

■ 매칭 API 개발 / EKS 연동 / Redis 및 인증 로직 담당

■ 담당자: 정재근

1. 매칭 API 개발 및 포팅

[1] 기능:

1) /signup: 가입 시 SNS 구독 등록 (이메일 인증용)

2) /login: SNS 구독 상태 확인 (Confirmed 상태만 로그인 허용)

3) /match: Redis 대기열 기반 매칭 처리 → Aurora 저장 → SNS 메일 알림 발송

[2] 진행 내용:

1) 기존 Lambda 매칭 로직을 Express.js 기반 HTTP API로 포팅

2) Docker 컨테이너화 → ECR Push → ArgoCD 배포 자동화 연동 (배포 파이프라인은 팀원 구성)

[3] 현재 상태:

1) HTTP API 정상 작동 확인 (POST /signup, /login, /match 테스트 완료)

2) EKS 기반 외부 Load Balancer 연결, 매칭 API 접근 정상

2. Terraform 기반 인프라 구성

[1] 작성한 모듈:

1) api_gateway: API Gateway 설정

2) lambda_match_logic: 매칭 로직 Lambda (포팅 전 단계)

3) lambda_match_request: 매칭 요청 Lambda

4) signup_api: 가입 처리 API

5) sqs_queue: 매칭 대기열 큐

6) sns_alarm: SNS Topic 및 알림 설정

7) redis: Redis 구성을 위한 모듈

[2] 기능:

1) main.tf, variables.tf, outputs.tf, terraform.tfvars 작성

2) terraform apply로 직접 배포 테스트 진행

[3] 이슈 및 대응:

1) 초기 Redis VPC 분리 문제 → VPC Peering 구성으로 해결₩

2) Routing Table 및 Security Group 수정 → EKS Pod에서 Redis 연결 정상 확인

3. 가입 인증 및 SNS 연동 로직

[1] 구성:

1) Cognito 가입 → Post Confirmation Lambda Trigger 사용

2) 가입된 유저 이메일을 SNS 구독자로 자동 등록 (Double Opt-in)

[2] 구독 확인 후 자동 처리:

1) SNS Confirm 이벤트 수신 Lambda 구성

2) 구독 확인된 이메일을 Cognito User Pool에서 찾아 email_verified = true 자동 업데이트

[3] 진행 상태:

1) SNS 구독 자동 등록 및 확인 로직 정상 작동 확인

2) email_verified 자동 업데이트 로직 구현 완료 및 테스트 중

4. Docker 컨테이너화 및 배포 연동

[1] 작성:

1) Dockerfile 구성 (node:18-alpine 기반)

2) ECR에 컨테이너 이미지 Push

3) CI/CD는 팀원이 구성한 ArgoCD + GitHub Actions 파이프라인 사용 (ECR Push → 배포 자동)

※ 현재 상태: 배포 완료, EKS 환경에서 API 정상 동작 확인

5. 테스트 결과

1) Redis 대기열 매칭 처리 - 정상 작동

2) Aurora DB 매칭 기록 저장 - 정상 저장

3) SNS 구독 등록 및 메일 발송 - 정상 확인

4) Cognito 구독 확인 → email_verified 업데이트 - 정상 작동 확인 중

5) EKS 배포 후 외부 접근 (/match API) - 정상 확인 완료

6. 향후 계획

1) SNS 구독 확인 → Cognito 자동 업데이트 로직 → 운영환경 반영

2) 가입 → 인증 → 매칭 → 알림까지 전체 프로세스 리소스 상태 안정화

3) 테스트 자동화 스크립트 추가 예정 (가입 → 인증 → 매칭 시나리오)

■ React SPA 프론트엔드 구현/ API Gateway ↔ Lambda 연결

■ 담당자: 오세빈

1. React SPA 구현 + S3/CloudFront 배포

- 라우팅: `react-router-dom`으로 로그인/회원가입/메인/프로필/채팅 페이지 구성.

- 상태관리: `Recoil`, `Zustand` 또는 `Redux Toolkit`.

- API 통신: `axios` + interceptor로 Cognito AccessToken 붙여서 요청.

- 파일 업로드: S3 Presigned URL 이용해서 이미지 직접 업로드.

- 프론트엔드 배포: npm run build → /build → S3에 업로드 → CloudFront로 서비스

- S3 버킷 퍼블릭 접근 차단 해제 → 정적 웹 호스팅

- CloudFront: SPA 지원을 위해 오류 문서 → index.html 설정

- HTTPS용 인증서: AWS Certificate Manager (ACM) 사용

2. React SPA와 Cognito 로그인 연동

- 로그인 연동: Cognito UserPool + Federated Identity → amazon-cognito-identity-js or aws-amplify

- OAuth2 기반 인증 플로우 구축, JWT 토큰 관리

- 프론트엔드에서 Amplify/Auth.js 등으로 연동

- Callback/Logout URL: CloudFront 도메인

- 사용자 속성: 이메일, 전화번호 등 등록

- Federated Identities 연결 시 IAM Role 부여

3. React SPA ↔ Route53 도메인 연결 및 HTTPS 적용

- CloudFront 배포된 React SPA를 ReactRoute 53 설정, SSL 인증서 적용

4. API 백엔드: API Gateway + Lambda

    1) API Gateway REST API 또는 HTTP API 생성

    2) `/users`, `/match`, `/chat` 등 엔드포인트 생성

    3) Lambda 함수 연결

4) CORS 설정 (CloudFront 도메인 허용)

5) Cognito Authorizer 연동 가능 → 인증된 요청만 처리

5. 채팅방 UI 및 라우팅 구성

- React Router 적용

- 채팅 컴포넌트 구현

6. 테스트 결과

- React SPA 구현 + S3/CloudFront 정상적으로 배포 완료

- React SPA에 API 백엔드 연결 테스트 완료

- React SPA와 Cognito 로그인 연동 완료

- React SPA에 도메인 연결 및 HTTPS 적용 완료

- 채팅방 UI 및 라우팅 구성 확인 완료

7. 향후 계획

- WebSocket 설계 흐름 관리: socket.io 연결 문제 발생, 실시간 통신 오류 해결중

- Lambda ↔ RDS 연결: 회원가입 시 RDS 테이블과 연동 (해당 담당 팀원과 병행 예정)

- 실시간 채팅 고도화: Amazon API Gateway (WebSocket) + Lambda + DynamoDB Streams + Cognito 형태로 최종 구현하여 프론트 → 백엔드 → 채팅 연동 흐름 마무리

- CI/CD 자동화: React 앱 빌드 → S3 배포 자동화 (해당 담당 팀원과 병행 예정)