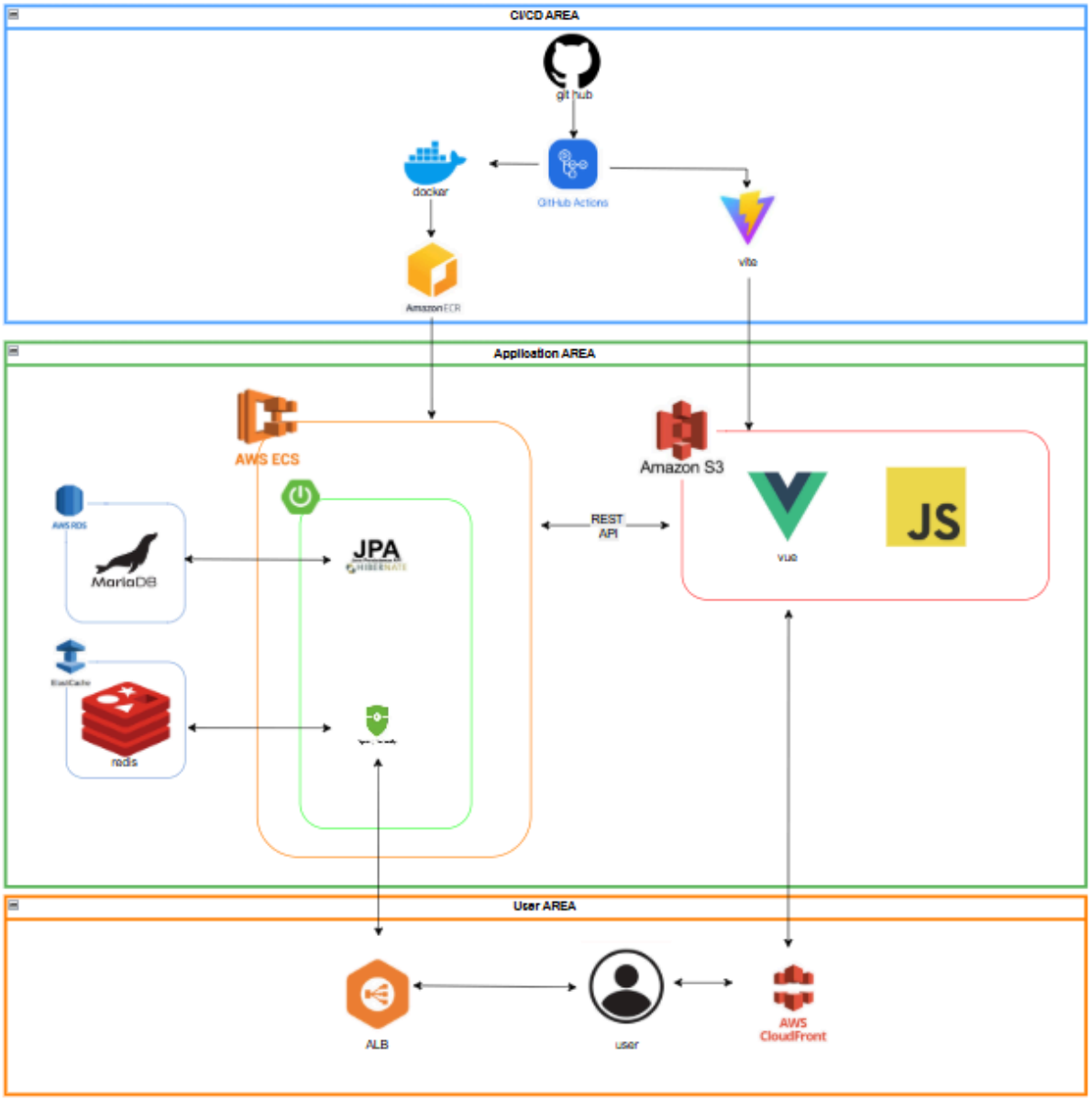
CI/CD 계획서

시스템 아키텍처



Infra

- Front End : S3, CloudFront
- Back End : ECR, ECS Fargate, ALB
- Database / Cache : RDS(MariaDB), ElastiCache(Redis)
- Common : ACM, IAM, CloudWatch

Front End CI/CD

1. 인프라 개요

- S3 버킷: **`ria-frontend-prod`**
    - 정적 웹 파일(dist) 저장
    - 정적 웹사이트 호스팅 OFF, CloudFront만 Origin으로 사용
- CloudFront
    - Origin: `ria-frontend-prod.s3.amazonaws.com`
    - OAC 기반 S3 Private 접근
    - 도메인: `ria-sales.site`, `www.ria-sales.site`
    - ACM(us-east-1) 인증서 연결

2. 배포 프로세스

1. 개발자가 `main` 브랜치에 Git Commit & Push
2. GitHub Actions Trigger → CI 빌드 수행
3. `npm ci` 후 `npm run build` 실행 → `dist/` 생성
4. GitHub Actions에서 `dist`를 아티팩트로 업로드
5. `deploy-to-s3` Job에서 아티팩트 다운로드
6. `aws-actions/configure-aws-credentials` + OIDC로 AWS Role Assume
7. `dist`를 다음 두 위치로 S3 업로드
    - 버전 보관용: `s3://ria-frontend-prod/releases/{VERSION}`
    - 현재 서비스용: `s3://ria-frontend-prod/`

8. `latest.txt` 파일에 현재 배포 버전 기록 →
   `s3://ria-frontend-prod/deployments/latest.txt` 업로드

9. CloudFront `create-invalidation`로 전체 경로(`/*`) 캐시 무효화

3. 배포 전략

- 정적 웹 자산의 무중단 배포
    - S3 정적 파일 교체 + CloudFront 캐시 무효화로 실시간에 가까운 반영
- 버전 관리
    - 빌드마다 고유 VERSION(YYYYMMDD-HHMMSS-SHA7) 생성
    - `releases/{VERSION}` 경로에 빌드 결과 보관
    - `deployments/latest.txt`로 마지막 배포 버전 추적
- 로그 및 추적
    - GitHub Actions 실행 로그
    - S3 버킷 버전 히스토리
    - CloudFront Invalidation 이력

4. Rollback 전략

1. 장애 시점 기준, 마지막 정상 VERSION 식별
   `deployments/latest.txt` 또는 릴리즈 관리 기록 확인

2. 해당 VERSION을 서비스 루트로 다시 동기화
   `aws s3 sync s3://ria-frontend-prod/releases/{VERSION} s3://ria-frontend-prod/ --delete`

3. CloudFront 캐시 무효화 재실행
   `aws cloudfront create-invalidation --distribution-id <배포ID> --paths "/*"`

5. Front End GitHub Actions 워크플로 (전체)

```
name: CI/CD-FE
```

```yaml
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
      - develop
    types: [opened, synchronize, reopened]

permissions:
  id-token: write
  contents: read

env:
  S3_BUCKET: ria-frontend-prod
  AWS_REGION: ap-northeast-2

jobs:
  # 1) PR / push 공통 빌드
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v5

      - name: Setup Node.js
        uses: actions/setup-node@v5
        with:
          node-version: 20

      - name: Cache node modules
        uses: actions/cache@v4
        with:
          path: ~/.npm
          key: ${{ runner.os }}-node-${{
hashFiles('**/package-lock.json') }}
          restore-keys: |
            ${{ runner.os }}-node-
```

```
      - name: Install dependencies
        run: npm ci

      - name: Build
        run: npm run build

      - name: Upload build artifact
        uses: actions/upload-artifact@v4
        with:
          name: fe-dist
          path: dist

  # 2) main 브랜치 push 시에만 S3 배포 + CloudFront 캐시 무효화
  deploy-to-s3:
    needs: build
    runs-on: ubuntu-latest
    if: github.event_name == 'push' && github.ref ==
'refs/heads/main'

    steps:
      - name: Checkout
        uses: actions/checkout@v5

      - name: Download build artifact
        uses: actions/download-artifact@v4
        with:
          name: fe-dist
          path: dist

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: ${{ secrets.AWS_ROLE_TO_ASSUME }}
          aws-region: ${{ env.AWS_REGION }}
          role-session-name: github-actions-ria-fe

      - name: Set deploy version
        id: version
        run: |
          DATE=$(date +'%Y%m%d-%H%M%S')
          SHA_SHORT=${GITHUB_SHA::7}
```

```
          echo "VERSION=${DATE}-${SHA_SHORT}" >> $GITHUB_OUTPUT

      - name: Upload build to versioned path
        run: |
          VERSION=${{ steps.version.outputs.VERSION }}
          aws s3 sync dist "s3://${S3_BUCKET}/releases/${VERSION}"
--delete

      - name: Upload build to root (current)
        run: |
          aws s3 sync dist "s3://${S3_BUCKET}/" --delete

      - name: Save latest version metadata
        run: |
          VERSION=${{ steps.version.outputs.VERSION }}
          echo "${VERSION}" > latest.txt
          aws s3 cp latest.txt
"s3://${S3_BUCKET}/deployments/latest.txt"

      - name: Invalidate CloudFront cache
        run: |
          aws cloudfront create-invalidation \
            --distribution-id ${{ secrets.CLOUDFRONT_DISTRIBUTION_ID
}} \
            --paths "/*"
```

Back End CI/CD

1. 인프라 개요

   컨테이너 오케스트레이션: **ECS Fargate**

   컨테이너 이미지: **ECR Private Repository (`ria-app`)**

   서비스 노출: **ALB → HTTPS(443) → Fargate(8080)**

   로그: CloudWatch Logs (awslogs 드라이버)

2. **ECS /** 네트워크 구조

   ECS Cluster: `ria-fiveguys-cluster`

ECS Service: `ria-backend-task-service-fargate`

Launch Type: `FARGATE`

AssignPublicIp: ENABLED (퍼블릭 서브넷에서 실행)

Target Group: `ria-backend-tg-8080` (Health check: `/actuator/health`)

Rolling Update

새 Task → 헬스체크 통과 → 기존 Task 종료

3. 배포 흐름

1. 개발자 변경 사항 → GitHub에 Push (`develop`, `main`)
2. GitHub Actions Workflow `CI/CD` 실행
3. 공통 Job(`build`)에서 Gradle Test + Build 수행
4. `main` 브랜치 Push 시에만 Docker Image Build & ECR Push
5. ECR에 `latest` + `v<run_number>` 태그로 이미지 업로드
6. `aws ecs update-service --force-new-deployment` 호출
7. ALB 헬스체크 기반 Rolling Update
8. 새 버전이 정상 동작하면 기존 Task 자동 종료

4. 배포 전략

자동화 수준

GitHub Actions → Docker Build → ECR Push → ECS 서비스 재배포까지 전체 자동화

버전 관리

`latest` : 항상 최신 배포 이미지

v<run_number> : 빌드 실행 번호 기반 버전 태그 (롤백용)

- 보안 관리

  GitHub Actions → AWS 접근: OIDC + AssumeRole (AWS_ROLE_TO_ASSUME)

  DB, Redis, JWT, S3, Google API 등은 모두 GitHub Secrets로 관리

- 무중단 배포

  ECS Rolling Update + ALB 헬스체크 조합

  헬스체크 실패 시 기존 Task 유지 → 자동적인 Failover 효과

5. **Rollback** 전략

1. CloudWatch / 서비스 로그에서 문제 탐지
2. 직전 정상 빌드의 IMAGE_TAG (vXXX) 식별
3. Task Definition에서 이미지 태그를 해당 버전으로 지정 후 재배포
   또는 동일 워크플로/수동 명령으로 force-new-deployment 수행
4. 헬스체크 통과 여부 확인 후 안정화

6. **Back End GitHub Actions** 워크플로 **(**전체**)**

```
name: CI/CD

on:
  push:
    branches: [ "develop", "main" ]
  pull_request:
    branches: [ "develop", "main" ]

permissions:
  id-token: write
  contents: read
  checks: write
```

```
  pull-requests: write

jobs:
  # 공통 빌드 (PR + push 모두 실행)
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Set up JDK 21
        uses: actions/setup-java@v4
        with:
          distribution: 'temurin'
          java-version: '21'

      - name: Grant execute permission for gradlew
        run: chmod +x gradlew

      # 1) 테스트 먼저 실행
      - name: Run tests
        run: ./gradlew clean test

      # JUnit 결과 업로드 (테스트 성공/실패와 상관없이 항상)
      - name: Upload JUnit test results
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: junit-results
          path: build/test-results/test

      # 2) 테스트 성공 후 빌드 (테스트 스킵)
      - name: Build (skip tests)
        run: ./gradlew build -x test

  # main 브랜치에 push 될 때만 이미지 빌드 + ECR 푸시 + ECS 재배포
  push-image-main:
    needs: build
    runs-on: ubuntu-latest
```

```
    if: github.event_name == 'push' && github.ref ==
'refs/heads/main'

    env:
      ECR_REPOSITORY: ria-app
      AWS_REGION: ${{ secrets.AWS_REGION }}
      ECS_CLUSTER: ria-fiveguys-cluster
      ECS_SERVICE: ria-backend-task-service-fargate
      IMAGE_TAG: v${{ github.run_number }}

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: ${{ secrets.AWS_ROLE_TO_ASSUME }}
          aws-region: ${{ secrets.AWS_REGION }}
          role-session-name: github-actions-ria

      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v2

      - name: Create .env.prod
        run: |
          cat << EOF > .env.prod
          SPRING_PROFILES_ACTIVE=prod

          GOOGLE_CALENDAR_SERVICE_ACCOUNT=/app/config/google.json
          GOOGLE_CALENDAR_ID=${{ secrets.GOOGLE_CALENDAR_ID }}

          AWS_S3_BUCKET=${{ secrets.AWS_S3_BUCKET }}
          AWS_ACCESS_KEY=${{ secrets.AWS_ACCESS_KEY }}
          AWS_SECRET_KEY=${{ secrets.AWS_SECRET_KEY }}
          AWS_REGION=${{ secrets.AWS_REGION }}

          DB_URL=${{ secrets.DB_URL }}
          DB_USERNAME=${{ secrets.DB_USERNAME }}
          DB_PASSWORD=${{ secrets.DB_PASSWORD }}
```

```
          REDIS_HOST=${{ secrets.REDIS_HOST }}
          REDIS_PORT=${{ secrets.REDIS_PORT }}

          JWT_SECRET=${{ secrets.JWT_SECRET }}
          JWT_ACCESS_EXPIRATION=${{ secrets.JWT_ACCESS_EXPIRATION }}
          JWT_REFRESH_EXPIRATION=${{ secrets.JWT_REFRESH_EXPIRATION
}}
          EOF

      - name: Build and Push Docker image (main)
        env:
          ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
          GOOGLE_JSON_BASE64: ${{ secrets.GOOGLE_JSON_BASE64 }}
        run: |
          docker build \
            --build-arg GOOGLE_JSON_BASE64="$GOOGLE_JSON_BASE64" \
            -t $ECR_REGISTRY/$ECR_REPOSITORY:latest \
            -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG \
            .
          docker push $ECR_REGISTRY/$ECR_REPOSITORY:latest
          docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG

      - name: Force new ECS deployment
        run: |
          aws ecs update-service \
            --cluster $ECS_CLUSTER \
            --service $ECS_SERVICE \
            --force-new-deployment \
            --region $AWS_REGION

  # 테스트 리포트 시각화
  test-report:
    needs: build
    runs-on: ubuntu-latest
    if: always()

    steps:
      # 1) 레포 체크아웃 (추가)
      - name: Checkout
        uses: actions/checkout@v4
```

```
      # 2) JUnit 결과 다운로드
      - name: Download JUnit test results
        uses: actions/download-artifact@v4
        with:
          name: junit-results
          path: build/test-results/test

      # 3) 테스트 리포트 퍼블리시
      - name: Publish Test Report
        uses: dorny/test-reporter@v1
        with:
          name: JUnit Tests
          path: build/test-results/test/TEST-*.xml
          reporter: java-junit
```

CI/CD 계획서 3

Application 설정 (Spring Boot)

1. 기본 설정 파일: `application.yml`

```
server:
  port: 8080
  address: 0.0.0.0

spring:
  profiles:
    active: dev

logging:
  level:
    org.hibernate.SQL: debug
    org.springframework.jdbc.datasource.init: debug

management:
  endpoints:
    web:
      base-path: /actuator
      exposure:
```

```
        include: health
  endpoint:
    health:
      enabled: true
      show-details: never
```

- 로컬/개발 환경 기본 프로필: dev
- 헬스체크 엔드포인트: /actuator/health
- Hibernate SQL 로그: debug 레벨

2. 운영 설정 파일: application-prod.yml

```
server:
  port: 8080
  address: 0.0.0.0

openai:
  api:
    key: ${OPENAI_API_KEY}

sse:
  default-timeout: 3600000  # 1시간 (밀리초)

google:
  calendar:
    service-account: ${GOOGLE_CALENDAR_SERVICE_ACCOUNT}
    application-name: My Calendar App
    id: ${GOOGLE_CALENDAR_ID}

cloud:
  aws:
    s3:
      bucket: ${AWS_S3_BUCKET}
    credentials:
      access-key: ${AWS_ACCESS_KEY}
      secret-key: ${AWS_SECRET_KEY}
    region:
      static: ${AWS_REGION}
```

```yaml
spring:
  datasource:
    url: ${DB_URL}
    username: ${DB_USERNAME}
    password: ${DB_PASSWORD}
    driver-class-name: org.mariadb.jdbc.Driver

  jpa:
    hibernate:
      ddl-auto: update  # 첫 배포에서 테이블 생성용 (생성 후
update/none으로 변경)
    show-sql: false
    defer-datasource-initialization: false
    properties:
      hibernate:
        format_sql: true
        dialect: org.hibernate.dialect.MariaDBDialect

  sql:
    init:
      mode: never

  data:
    redis:
      host: ${REDIS_HOST}
      port: ${REDIS_PORT}
      ssl:
        enabled: true
    # password: ${REDIS_PASSWORD}
    # database: ${REDIS_DATABASE}

  jwt:
    secret: ${JWT_SECRET}
    access-expiration: ${JWT_ACCESS_EXPIRATION}
    refresh-expiration: ${JWT_REFRESH_EXPIRATION}

management:
  endpoints:
    web:
      base-path: /actuator
```

```
      exposure:
        include: health

  endpoint:
    health:
      enabled: true
      show-details: never

  health:
    db:
      enabled: false
    redis:
      enabled: false

logging:
  level:
    org.hibernate.SQL: info
    org.springframework.jdbc.datasource.init: warn
```

- DB, Redis, JWT, S3, Google 등 모든 외부 자원은 환경변수(= GitHub Secrets → 컨테이너 env) 기준으로 주입
- 운영 환경에서는 SQL 로그 최소화, 헬스체크는 ALB/ECS 용도로만 사용

Database / Redis / 운영 전략

1. **RDS(MariaDB)** 구성

- 퍼블릭 접근: YES
- 엔드포인트: `ria-public.XXXX.ap-northeast-2.rds.amazonaws.com:3306`
- 접근 제어:
    - Inbound 3306
        - 개발자 로컬 IP(/32)
        - ECS Service가 사용하는 SG

- 연결 방식:
    - ECS Fargate → Private 네트워크로 RDS 접근
    - 로컬 개발 PC → Public Endpoint로 직접 접속 (SG로 제한)

2. **Redis(ElastiCache)** 구성

- Public: NO
- Private Subnet 내 전용
- Endpoint: `master.ria-redis.XXXX.apn2.cache.amazonaws.com:6379`
- Inbound: 6379, Source = ECS SG
- Spring Boot에서 SSL 활성화로 접근

3. CI/CD 연동

- DB_URL, DB_USERNAME, DB_PASSWORD, REDIS_HOST, REDIS_PORT 등은
    - GitHub Secrets에 저장
    - `.env.prod`를 통해 Docker Build 시 환경 파일로 주입
    - Spring Boot는 `application-prod.yml`의 `${...}` Placeholder로 사용

4. 장애 및 복구 전략(요약)

- RDS
    - 접속 불가 시: SG Inbound IP, Public Accessible 설정, 엔드포인트/포트 확인
    - 데이터 백업: RDS Snapshot, 자동 백업 활용
- Redis
    - 장애 시: 세션/캐시 장애 영향 범위 파악 후, 애플리케이션 재시도 전략으로 대응

운영 체크리스트

1. 배포 직후 확인 항목

- Frontend
    - S3 sync 완료 여부
    - CloudFront Invalidation 성공 여부
    - 주요 페이지(메인/로그인 등) 200 응답
- Backend
    - `/actuator/health` 200 응답
    - ECS Service Events에서 배포 성공(Event: SERVICE_DEPLOYMENT_COMPLETED) 확인
    - CloudWatch Logs에서 ERROR/Exception 여부 확인

2. 문제 발생 시 대응 순서

1. 로그 확인
    - 백엔드: CloudWatch Logs
    - 프론트: 브라우저 콘솔, Network 탭
2. 영향도 파악
    - 전체 장애 / 특정 기능 / 특정 API만 장애인지 확인
3. 롤백
    - 프론트: 이전 S3 Release 버전으로 sync 후 CF Invalidation
    - 백엔드: 직전 정상 Docker Image 태그(`vN`)로 ECS force-new-deployment
3. 향후 확장 고려

- 브랜치/환경 분리
    - `develop` → dev 환경(ECS dev 서비스, S3 dev 버킷, CF dev 배포)
    - `main` → prod 환경
- 도메인 분리
    - dev: `dev.ria-sales.site`

- prod: `ria-sales.site`, `api.ria-sales.site`