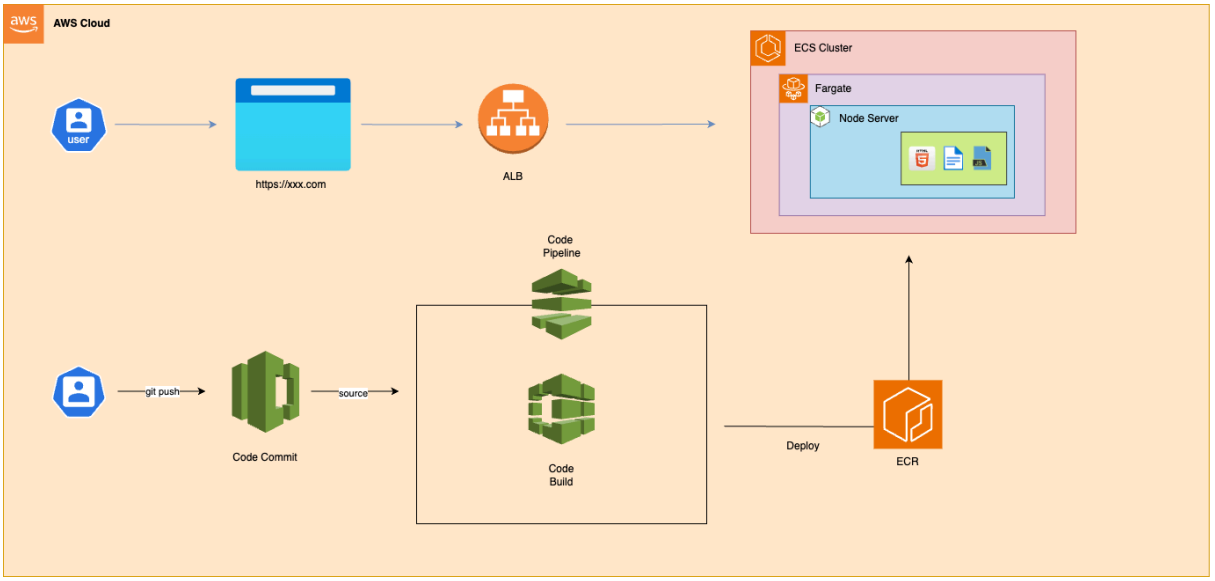
ECS 무중단 배포 및 정적자원 분리

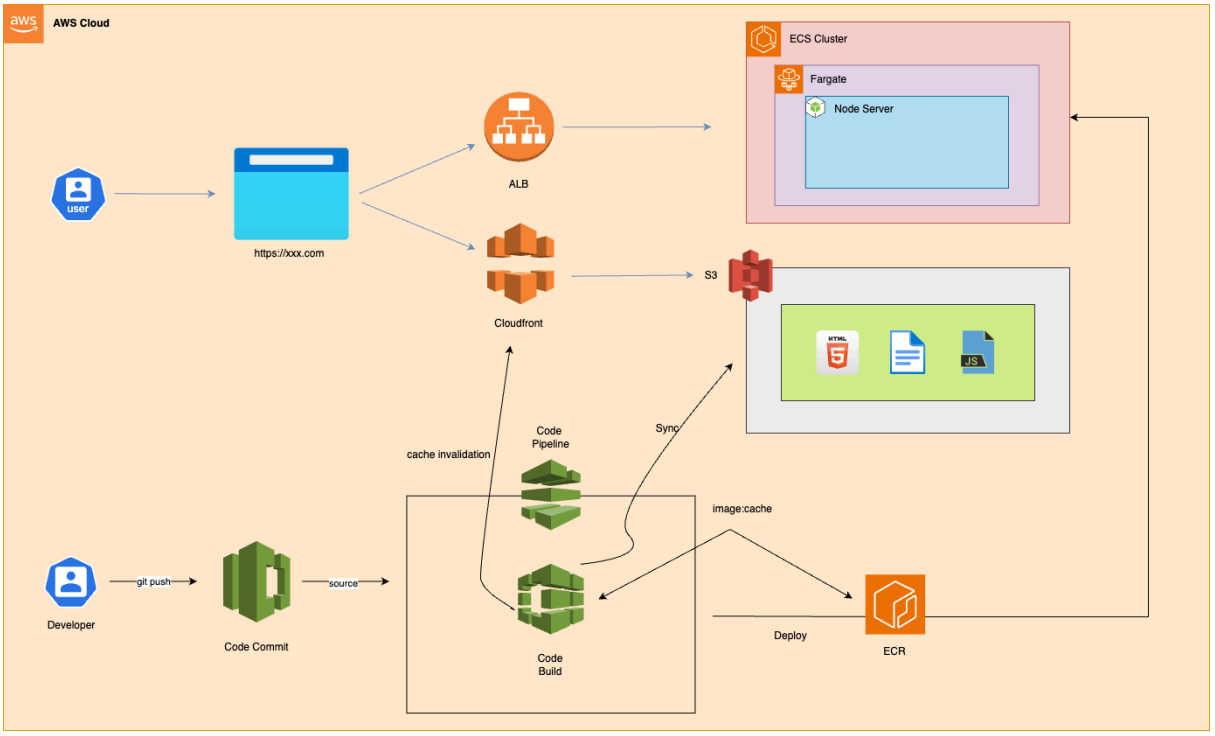
<기존 서비스 aws 설계도>



문제점

1. 배포시 캐시로 인해 기존 버전을 참조하는 이슈 → 정적파일 에러 발생
2. 빌드 속도 저하

<향후 적용할 서비스 aws 설계도>



개선사항

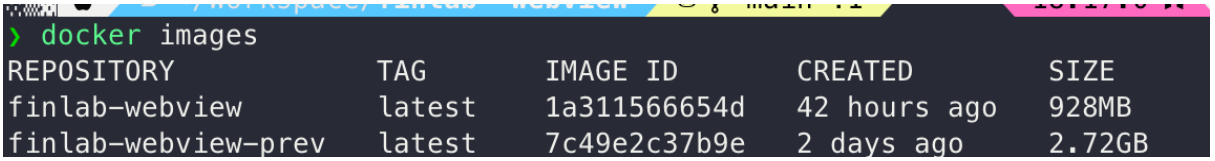
1. 도커 이미지 경량화 (multi-stage build)

예시) finlab-webview - Dockerfile

```
FROM public.ecr.aws/docker/library/node:18.17.0-alpine AS b
uilder

RUN corepack enable

WORKDIR /app

COPY .npmrc package.json yarn.lock ./

RUN yarn install --production --frozen-lockfile

COPY . .

ARG NUXT_APP_CDN_URL

RUN NUXT_APP_CDN_URL=$NUXT_APP_CDN_URL yarn build
```

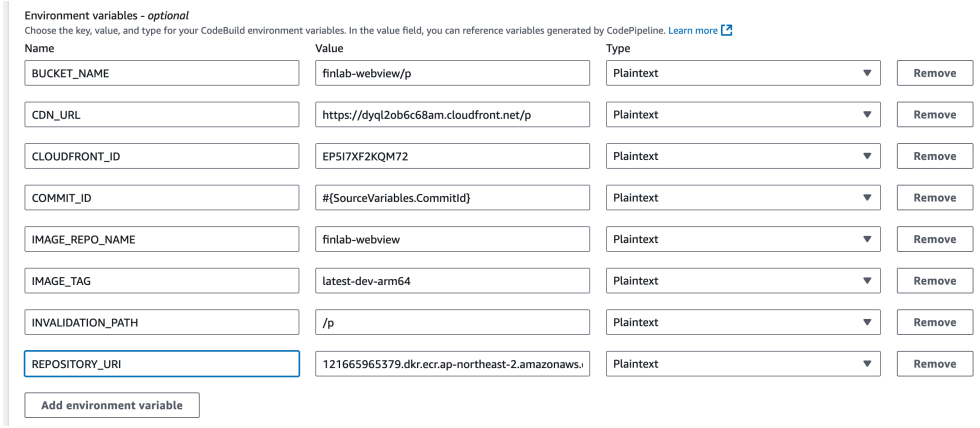
```
FROM public.ecr.aws/docker/library/node:18.17.0-alpine

WORKDIR /app

COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/.output ./.output

EXPOSE 3000

CMD [ "node", ".output/server/index.mjs" ]
```

finlab-webview-prev(도커 이미지 경량화 이전) vs finlab-webview (도커 이미지 경량화 이후)



2. ECR 원격 캐싱 적용 (https://aws.amazon.com/ko/blogs/containers/announcing-remote-cache-support-in-amazon-ecr-for-buildkit-clients/)

예시) finlab-webview - buildspec.yml (각각의 환경변수는 codebuild에서 지정)



```
version: 0.2

phases:
  pre_build:
    commands:
      - echo "Logging in to Amazon ECR..."
      - aws ecr get-login-password --region ap-northeast-2
| docker login --username AWS --password-stdin "$REPOSITORY
_URI"
      # rollback tag를 생성시킬 경우 꼭 이미지를 삭제한 뒤에 buil
d 단계로 넘어가야 build 결과물 issue가 안 생김.
      - docker pull $REPOSITORY_URI/$IMAGE_REPO_NAME:$IMAGE
_TAG
      - docker tag $REPOSITORY_URI/$IMAGE_REPO_NAME:$IMAGE_
TAG $REPOSITORY_URI/$IMAGE_REPO_NAME:$ROLLBACK_IMAGE_TAG
      - docker push $REPOSITORY_URI/$IMAGE_REPO_NAME:$ROLLB
ACK_IMAGE_TAG
      - docker rmi -f $(docker images -q) #중요!!! 삭제해야
함.
  build:
    commands:
      - echo "Building the Docker image..."
      - docker buildx create --use
      - |
        docker buildx build --build-arg NUXT_APP_CDN_URL
="$CDN_URL/$COMMIT_ID" -t "$REPOSITORY_URI"/"$IMAGE_REPO_NA
ME":"$IMAGE_TAG" \
          --cache-to mode=max,image-manifest=true,oci-media
types=true,type=registry,ref="$REPOSITORY_URI"/"$IMAGE_REPO
_NAME":"$CACHE_IMAGE_TAG" \
          --cache-from type=registry,ref="$REPOSITORY_UR
I"/"$IMAGE_REPO_NAME":"$CACHE_IMAGE_TAG" \
          --push .
      - docker create --name temp-container "$REPOSITORY_UR
I"/"$IMAGE_REPO_NAME":"$IMAGE_TAG"
      - docker cp temp-container:/app/.output/public ./stat
ic
      - docker rm temp-container
```

```
  post_build:
    commands:
      - echo "Pushing the Docker image..."
      - echo "Generating imagedefinitions.json..."
      - echo "Uploading static files to S3..."
      - aws s3 sync ./static/ s3://$BUCKET_NAME/$COMMIT_ID
      - echo "Invalidating cache..."
      - aws cloudfront create-invalidation --distribution-i
d $CLOUDFRONT_ID --paths "$INVALIDATION_PATH"
```

docker의 고급 기능(캐싱)을 사용하기 위해서 `buildx` 를 사용해야함.

- ECR 원격 캐싱

☐	cache	Other	2025년 3월 26일, 16:50:00 (UTC+09)	438.65	Copy URI	sha256:c839fa10d5e835...	-
☐	latest	Image Index	2025년 3월 26일, 16:49:57 (UTC+09)	78.89	Copy URI	sha256:ecdc13d978e2e0...	2025년 3월 26일, 16:50:00 (UTC+09)
☐	-	Image	2025년 3월 26일, 16:49:57 (UTC+09)	0.00	Copy URI	sha256:19b9628fa83ad0...	-
☐	-	Image	2025년 3월 26일, 16:49:57 (UTC+09)	78.89	Copy URI	sha256:8e6b58ecfa5e748...	2025년 3월 26일, 16:50:00 (UTC+09)
☐	-	Other	2025년 3월 26일, 16:39:40 (UTC+09)	438.65	Copy URI	sha256:59e3999f55661a...	2025년 3월 26일, 16:49:14 (UTC+09)
☐	-	Image Index	2025년 3월 26일, 16:39:37 (UTC+09)	78.89	Copy URI	sha256:615ef6a64f9ed61...	2025년 3월 26일, 16:39:40 (UTC+09)
☐	-	Image	2025년 3월 26일, 16:39:37 (UTC+09)	0.00	Copy URI	sha256:9ada2564eaa9a7...	-
☐	-	Image	2025년 3월 26일, 16:39:37 (UTC+09)	78.89	Copy URI	sha256:917e1e11a9a571...	2025년 3월 26일, 16:39:40 (UTC+09)
☐	-	Other	2025년 3월 26일, 16:35:43 (UTC+09)	438.58	Copy URI	sha256:7f935e9f86725c6...	2025년 3월 26일, 16:38:49 (UTC+09)

- Build 단계 duration 속도 비교

  캐시 적용전 (116 secs)

Name	Status	Context	Duration
SUBMITTED	Succeeded	-	<1 sec
QUEUED	Succeeded	-	<1 sec
PROVISIONING	Succeeded	-	3 secs
DOWNLOAD_SOURCE	Succeeded	-	1 sec
INSTALL	Succeeded	-	<1 sec
PRE_BUILD	Succeeded	-	12 secs
BUILD	Succeeded	-	116 secs
POST_BUILD	Succeeded	-	<1 sec
UPLOAD_ARTIFACTS	Succeeded	-	<1 sec
FINALIZING	Succeeded	-	<1 sec
COMPLETED	Succeeded	-	-

캐시 적용후 (55 secs)

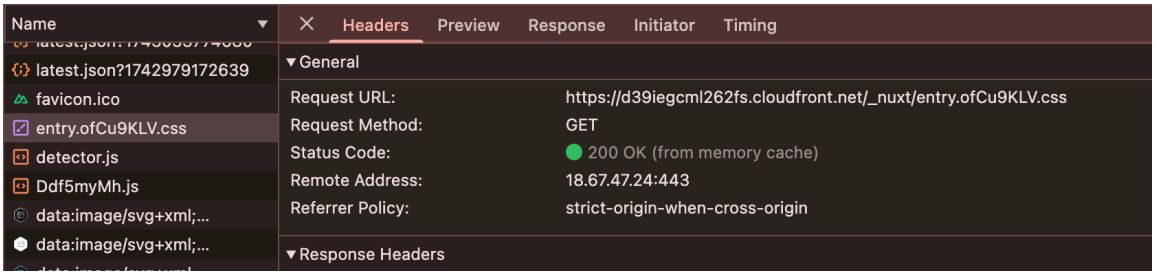
Name	Status	Context	Duration
SUBMITTED	Succeeded	-	<1 sec
QUEUED	Succeeded	-	<1 sec
PROVISIONING	Succeeded	-	4 secs
DOWNLOAD_SOURCE	Succeeded	-	1 sec
INSTALL	Succeeded	-	<1 sec
PRE_BUILD	Succeeded	-	12 secs
BUILD	Succeeded	-	55 secs
POST_BUILD	Succeeded	-	<1 sec
UPLOAD_ARTIFACTS	Succeeded	-	<1 sec
FINALIZING	Succeeded	-	<1 sec
COMPLETED	Succeeded	-	-

도커 layer cache

```
#5 [internal] load build context
#5 transferring context: 222.71kB 0.0s done
#5 DONE 0.0s

#9 [builder 2/7] RUN corepack enable
#9 CACHED

#10 [builder 3/7] WORKDIR /app
#10 CACHED

#11 [builder 4/7] COPY package.json yarn.lock ./
#11 CACHED

#12 [builder 5/7] RUN yarn install --production --frozen-lockfile
#12 CACHED
```

3. 정적자원 분리 S3 배포 및 cloudfront 연결 (Dockerfile에서 NUXT_APP_CDN_URL 사용한다면 이 단계는 건너뜀)

```
// nuxt.config.js
export default defineNuxtConfig({
    // ..중략..
  app: {
    cdnURL: 'https://d39iegcml262fs.cloudfront.net' // e
x) 환경별로 cdn url 지정
  }
})
```



효과

1. 지연 시간 감소

   사용자와 가장 가까운 CloudFront 엣지 로케이션에서 처리. (외국 이용자가 있다면 유리)

2. 캐싱으로 인한 부하 분산

   원본 서버인(S3) 요청하지 않고 바로 제공

3. 배포시마다 캐싱 무효화 진행

   배포 하고 나서 캐싱으로 인한 이전 버전 바라보지 않고 강력 새로고침 할 필요 없이 새로운 버전 적용.