


# Reduce deployments runtime with GitHub Actions Cache

Here's to incremental deployments 🥂.


# About Me



- Head of Platform Engineering @Baania
- Often known as DevSecMLFinDataOps
- Faster deployments -> Faster iterations

 [kahnwong](#)

 [Karnsiree Wong](#)

 [karnwong.me](#)

# CI/CD Workflow



# CI/CD Breakdown

- Image build
- Image push
- Image pull (during deployment)

# Dockerfile

```
FROM node:18
```

```
WORKDIR /opt/build
```

```
COPY package.json .
```

```
COPY yarn.lock .
```

```
RUN yarn install
```

```
COPY . .
```

```
RUN yarn build
```

```
EXPOSE 3000
```

```
CMD [ "yarn", "start", "-H", "0.0.0.0" ]
```

# GitHub Actions Buildx

```
- name: Build and tag image
  uses: docker/build-push-action@v5
  with:
    context: .
    builder: ${{ steps.buildx.outputs.name }}
    file: Dockerfile
    push: true
    tags: ${{ steps.meta.outputs.tags }}
    provenance: false
```

**But we can cache docker layers**

# GitHub Actions Buildx with Cache

```
- name: Build and tag image
  uses: docker/build-push-action@v5
  with:
    context: .
    builder: ${{ steps.buildx.outputs.name }}
    file: Dockerfile
    push: true
    cache-from: type=gha # add this
    cache-to: type=gha,mode=max # add this
    tags: ${{ steps.meta.outputs.tags }}
    provenance: false
```



# Buildx

**build.yaml**

on: push



**build**

3m 24s

# Buildx with Cache

**build-with-cache.yaml**

on: push



**build-with-cache**

2m 57s

**What about image size?**

# Dockerfile with multi-stage build

```
# ----- builder ----- #
FROM node:18 AS builder

WORKDIR /opt/build

COPY package.json .
COPY yarn.lock .
RUN yarn install

COPY . .
RUN yarn build

# ----- package ----- #
FROM node:18-alpine AS deploy

WORKDIR /app
COPY --from=builder /opt/build/.next ./next
COPY --from=builder /opt/build/node_modules ./node_modules
COPY --from=builder /opt/build/public ./public
COPY --from=builder /opt/build/next.config.js ./
COPY --from=builder /opt/build/package.json ./

EXPOSE 3000
CMD [ "yarn", "start", "-H", "0.0.0.0" ]
```

**Let's guess the image size!**

# Normal build

```
github-actions-cache-demo on master [!?] via  desktop-linux via @ v18.18.2 on └─ @baania.com
> docker build -f Dockerfile -t nextjs-blog .
[+] Building 37.9s (12/12) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 206B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:18
=> [1/7] FROM docker.io/library/node:18@sha256:7ce8b205d15e30fd395e5fa4000bcd5f95fcff3f434fe75822e54e82a5f5cf82
=> [internal] load build context
=> => transferring context: 1.34MB
=> CACHED [2/7] WORKDIR /opt/build
=> CACHED [3/7] COPY package.json .
=> CACHED [4/7] COPY yarn.lock .
=> CACHED [5/7] RUN yarn install
=> [6/7] COPY . .
=> [7/7] RUN yarn build
=> exporting to image
=> => exporting layers
=> => writing image sha256:9b919bad81fdef523115dfd1553300ab15c054231d36c44e89a821f337a6f354
=> => naming to docker.io/library/nextjs-blog

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview

github-actions-cache-demo on master [!?] via  desktop-linux via @ v18.18.2 on └─  took 38s
> docker images | grep nextjs-blog
nextjs-blog          latest          9b919bad81fd   10 seconds ago   3.87GB
```

# Multi-stage build

```
github-actions-cache-demo on master [!?] via  desktop-linux via @ v18.18.2 on └─ @baania.com
> docker build -f Dockerfile.multi-stage -t nextjs-blog-multi-stage .
[+] Building 47.5s (20/20) FINISHED
=> [internal] load build definition from Dockerfile.multi-stage
=> => transferring dockerfile: 609B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:18-alpine
=> [internal] load metadata for docker.io/library/node:18
=> [builder 1/7] FROM docker.io/library/node:18@sha256:7ce8b205d15e30fd395e5fa4000bcd5f95fcff3f434fe75822e54e82a5f5cf82
=> [internal] load build context
=> => transferring context: 1.44MB
=> [deploy 1/7] FROM docker.io/library/node:18-alpine@sha256:435dcad253bb5b7f347ebc69c8cc52de7c912eb7241098b920f2fc2d7843183d
=> CACHED [builder 2/7] WORKDIR /opt/build
=> CACHED [builder 3/7] COPY package.json .
=> CACHED [builder 4/7] COPY yarn.lock .
=> CACHED [builder 5/7] RUN yarn install
=> [builder 6/7] COPY . .
=> [builder 7/7] RUN yarn build
=> CACHED [deploy 2/7] WORKDIR /app
=> [deploy 3/7] COPY --from=builder /opt/build/.next ./next
=> [deploy 4/7] COPY --from=builder /opt/build/node_modules ./node_modules
=> [deploy 5/7] COPY --from=builder /opt/build/public ./public
=> [deploy 6/7] COPY --from=builder /opt/build/next.config.js ./
=> [deploy 7/7] COPY --from=builder /opt/build/package.json ./
=> exporting to image
=> => exporting layers
=> => writing image sha256:a46142e240c255c29cc98c8dfff8dad5b297a5d6272d7abeb2b92651de250bb5
=> => naming to docker.io/library/nextjs-blog-multi-stage

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview

github-actions-cache-demo on master [!?] via  desktop-linux via @ v18.18.2 on └─ @baania.com took 48s
> docker images | grep nextjs-blog-multi-stage
nextjs-blog-multi-stage  latest          a46142e240c2   About a minute ago   904MB
```

# How much can we save?

## Build time

- 30s per build (from 3m24s to 2m57s)

## Image storage (compressed)

- 300MB per image (from 450MB to 150MB)

# Ready for cost reduction?

If we deploy 150 times / month

# Cost breakdown

Type	Actions Runtime	Image storage	AWS ECR Cost
No cache	$3\text{m}24\text{s} * 150 = 510\text{m}$	$450\text{MB} * 150 = 66\text{GB}$	$66\text{GB} * 0.10 \text{ USD} = 6.6 \text{ USD}$
With cache	$2\text{m}57\text{s} * 150 = 442.5\text{m}$	$150\text{MB} * 150 = 22\text{GB}$	$22\text{GB} * 0.10 \text{ USD} = 2.2 \text{ USD}$

In total, we can save 67.5m and 4.4 USD per month.



Check out the slides and repo!

