


Faster deployments with multi-stage build caching

Here's to incremental deployments.

About me



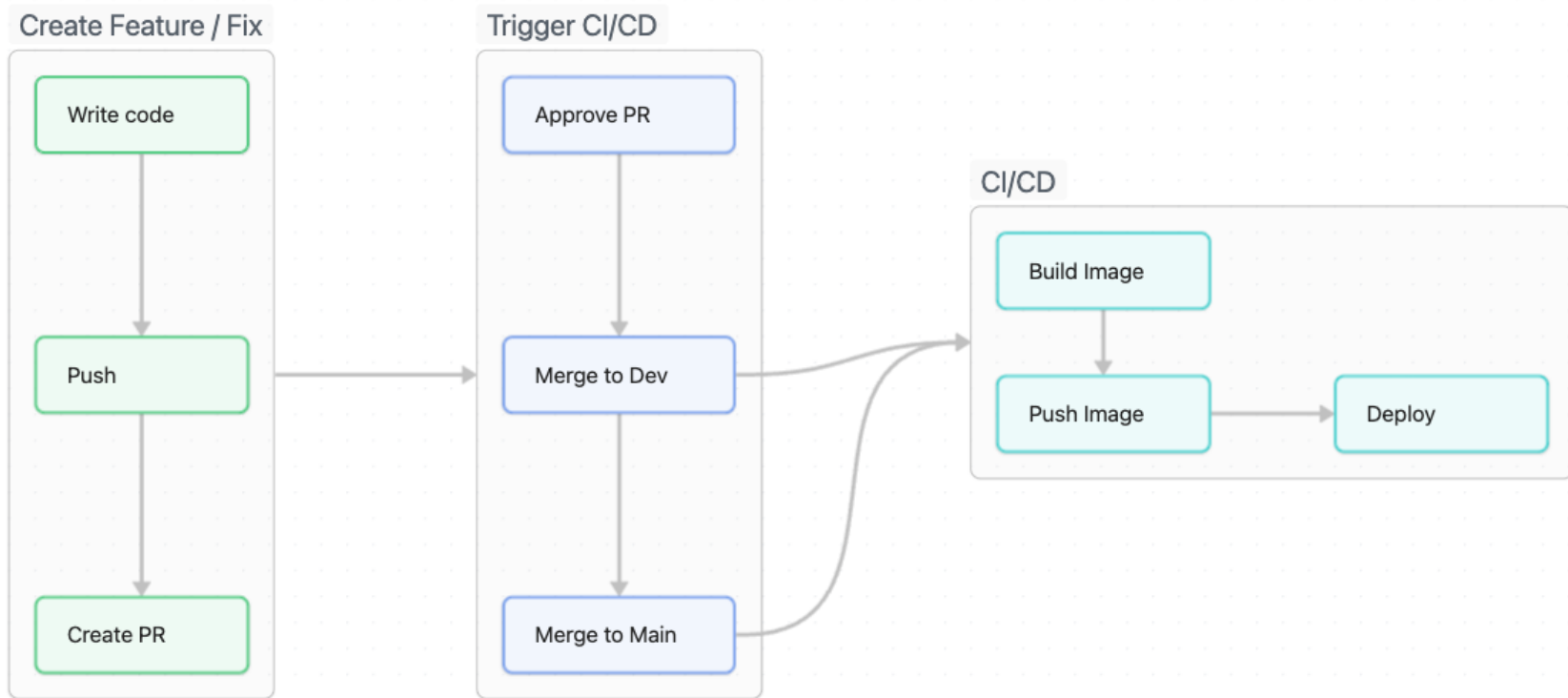
- Head of Platform Engineering @Baania
- Often known as DevSecMLFinDataOps
- Faster deployments -> Faster iterations
- I love automation

 [kahnwong](#)

 [Karnsiree Wong](#)

 [karnwong.me](#)

CI/CD workflow



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

How do we know that we're using cache?

Check GitHub Actions caches from all workflows

<> Code

Issues

Pull requests 6

Actions

Projects

Wiki

Security 1

Insights

Settings

Actions

New workflow

All workflows

Build

Build multi-stage

Build multi-stage cache

Build with cache

Management

Caches

Runners

Beta

Caches

Showing caches from all workflows. [Learn more about managing caches.](#)

Approaching total cache storage limit (8.09 GB of 10 GB Used)
Least recently used caches will be automatically evicted to limit the total cache storage to 10 GB. [Learn more about cache usage.](#)

49 caches

Branch ▾

Sort ▾

index-buildkit-1-49c8f7d5#9 5.8 KB cached 2 days ago	master	Last used 2 days ago	🗑
index-buildkit-1-49c8f7d5#8 5.8 KB cached 2 days ago	master	Last used 2 days ago	🗑
buildkit-blob-1-sha256:d61a850b2d93a5ba8332fe6a1479476de561a339196b210ee67a569ffd5cf69 450 Bytes cached 2 days ago	master	Last used 2 days ago	🗑
buildkit-blob-1-sha256:d0b5d943177825ac5f2ee938d308bdd7a44e444bdabbef822048bdbec99b169c 48 KB cached 2 days ago	master	Last used 2 days ago	🗑
buildkit-blob-1-sha256:c30e0ac6c6d561f6ec4e8d3ccebdeedbe505923a613afe3df4a6ea0a40f2240a 3.3 KB cached 2 days ago	master	Last used 2 days ago	🗑

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" ]
```

Why multi-stage build?

- ``yarn build`` creates a lot of temporary files
- We don't need those "temporary files" for ``yarn start``

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)

Any questions?

Ready for cost reduction?

If we deploy 150 times / month

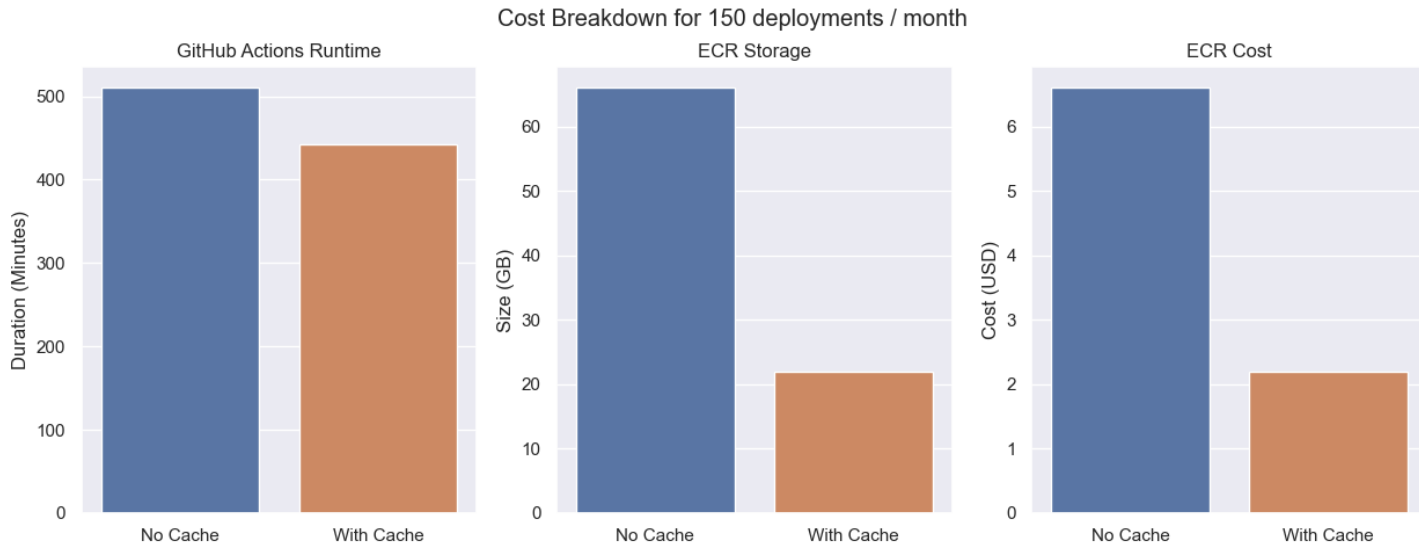
Cost breakdown

Service	No Cache	With Cache
GitHub Actions (Runtime)	$3\text{m}24\text{s} * 150 = 510\text{m}$	$2\text{m}57\text{s} * 150 = 442.5\text{m}$

Service	Normal Build	Multi-Stage Build
ECR (Storage)	$450\text{MB} * 150 = 66\text{GB}$	$150\text{MB} * 150 = 22\text{GB}$
ECR (Cost)	$150\text{MB} * 150 = 22\text{GB}$	$22\text{GB} * 0.10 \text{ USD} = 2.2 \text{ USD}$

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

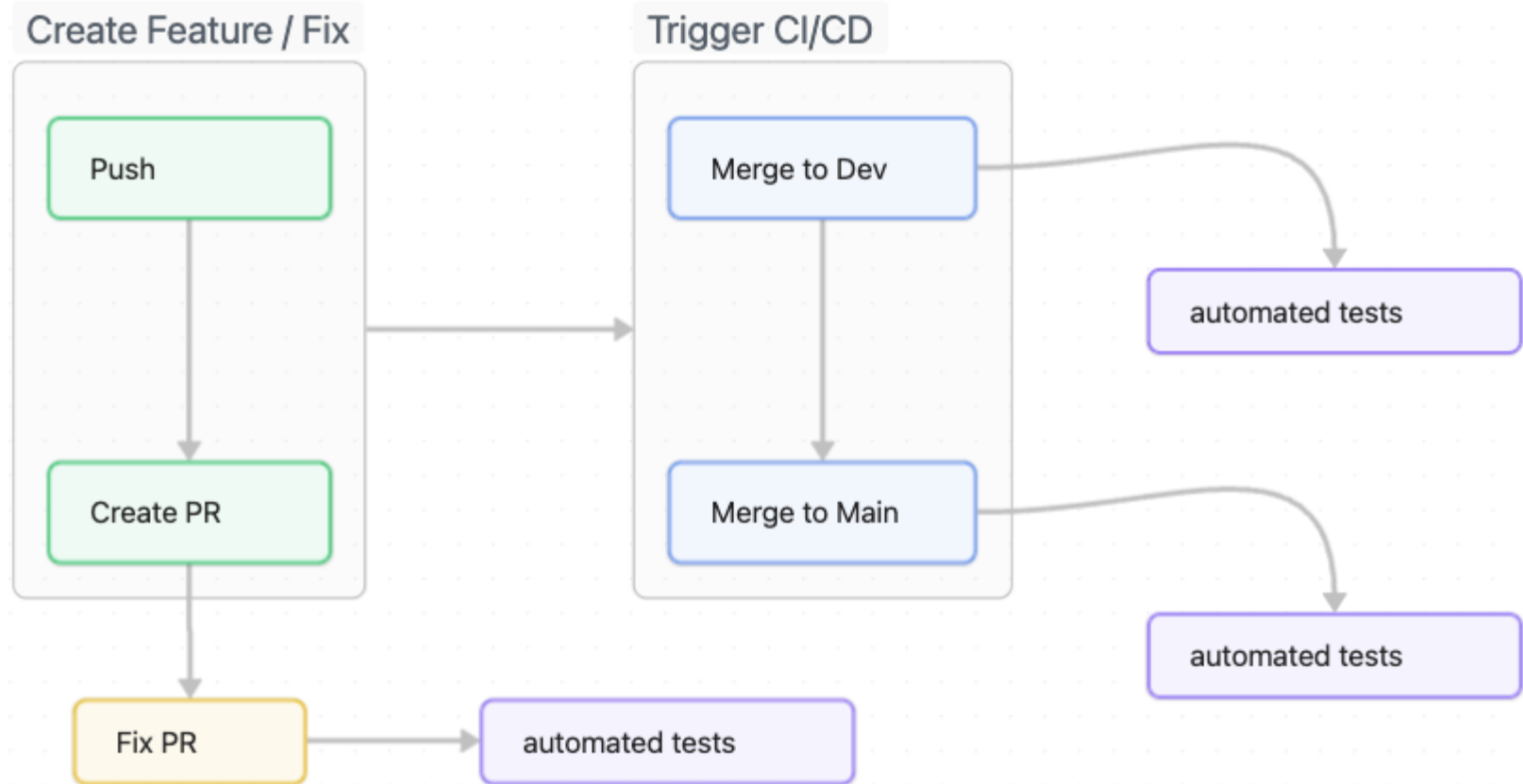
I know you love pretty charts



GitHub Actions Cache can also be used with runtimes

```
steps:  
- uses: actions/checkout@v4  
- uses: actions/setup-node@v4  
  with:  
    node-version: 18  
    cache: 'yarn'  
- run: yarn install  
- run: yarn build
```

Let's revisit CI/CD workflow again



Any questions?

Summary

- GitHub Actions cache can be used with docker image build and setup actions
- Multi-stage build can drastically reduce image size (in turn, reducing image storage cost)
- These lead to faster CI/CD run time and faster deployments
- During PR review, automated checks can be done faster as well

Check out the slides and repo!

