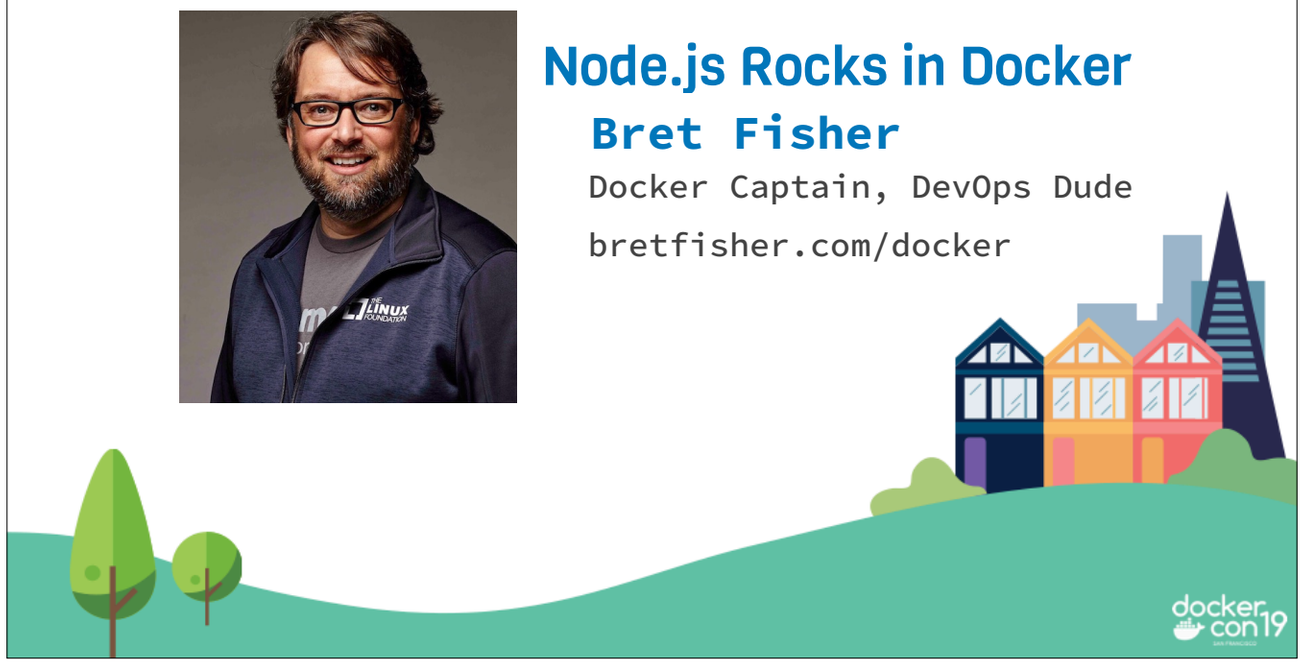

Node.js Rocks in Docker
Bret Fisher
Docker Captain, DevOps Dude
bretfisher.com/docker
dockercon19

Who's This Session For?

- You know some Node
- You know some Docker
- You want more Node+Docker awesomesauce

What We Gonna' Learn Bret?

- Node Dockerfile Best Practices
- Make a real-world multi-stage Dockefile
- Build with auditing and sec scans
- Proper Node shutdown
- Node HTTP connection management

Node Dockerfiles

Every Node Sample Dockerfile

```
1  FROM node:12
2
3  EXPOSE 3000
4
5  WORKDIR /app
6
7  COPY package.json package-lock*.json ./
8
9  RUN npm install && npm cache clean --force
10
11 COPY . .
12
13 CMD ["npm", "start"]
```

Node Base Image Guidelines

- Stick to even numbered major releases
- Don't use **:latest** tag
- Start with Debian if migrating
- Use stretch (default) not jessie
- Try slim first
- Move to Alpine later, maybe

Resources

Node LTS Releases https://github.com/nodejs/Release#release-schedule

why does slim say not recommended: https://github.com/nodejs/docker-node/issues/26

default images often based on buildpacks (big): https://hub.docker.com/_/buildpack-deps

ONBUILD not recommended https://github.com/docker-library/official-images/issues/2076

When to use Alpine Images

- Alpine is "small" and "sec focused"
- But Debian/Ubuntu are smaller now too
- ~100MB space savings isn't significant
- Alpine has its own issues
- Alpine CVE scanning fails
- Enterprises may require CentOS or Ubuntu/Debian

https://kubedex.com/follow-up-container-scanning-comparison/
base image security comparison
https://docs.google.com/spreadsheets/d/1GxtCRMyKKvWUvslsGTpqUZ1wPnh1KMGha-Umkdo_xQE/edit#gid=0

Image Sizes for node/slim/alpine

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
node	12.0-slim	8651cebb80e1	23 hours ago	150MB
node	12.0	d97e1f326ca9	23 hours ago	906MB
node	12.0-alpine	80a733d0cd8c	23 hours ago	77.3MB
node	10-slim	914bfdbef6aa	4 weeks ago	143MB
node	10-stretch	64c810caf95a	4 weeks ago	899MB
node	10-jessie	5c6c62fac703	4 weeks ago	680MB

Image Sizes for node/slim/alpine

TAG	IMAGE ID	CREATED	SIZE
12.0-slim	8651cebb80e1	23 hours ago	150MB
12.0	d97e1f326ca9	23 hours ago	906MB
12.0-alpine	80a733d0cd8c	23 hours ago	77.3MB
10-slim	914bfdbef6aa	4 weeks ago	143MB
10-stretch	64c810caf95a	4 weeks ago	899MB
10-jessie	5c6c62fac703	4 weeks ago	680MB

node_modules in Images

- **Problem:** we shouldn't build images with **node_modules** from host
  - Example: **node-gyp**
- **Solution:** add **node_modules** to **.dockerignore**
- copy **.gitignore**?

we have two problems when building local images with our repos that have node_modules as a sub directory

1. by default that node_modules will be copied into image
    1. fix: add to .dockerignore

Least Privilege: Using node User

- Official node images have a node user
- But it's not used until you **USER node**
- Do this after **apt/apk** and **npm i -g**
- Do this before **npm i**
- May cause permissions issues with write access
- May require **chown node:node**

https://docs.docker.com/engine/reference/builder/#user
https://docs.docker.com/engine/reference/builder/#copy

0.Dockerfile

```
1  FROM node:12
2
3  EXPOSE 3000
4
5  WORKDIR /app
6
7  COPY package.json package-lock*.json ./
8
9  RUN npm install && npm cache clean --force
10
11 COPY . .
12
13 CMD ["npm", "start"]
```

1.Dockerfile

```
1  FROM node:12-slim
2
3  EXPOSE 3000
4
5  RUN mkdir /app && chown -R node:node /app
6
7  WORKDIR /app
8
9  USER node
10
11 COPY --chown=node:node package.json package-lock*.json ./
12
13 RUN npm install && npm cache clean --force
14
15 COPY --chown=node:node . .
16
17 CMD ["npm", "start"]
```

Process Management and Shutdown

Node Process Management In Containers

- No need for nodemon, forever, or pm2 on server
  - We'll use nodemon in dev for file watch later
- Docker manages app start, stop, restart, healthcheck
- Node multi-thread: Docker manages multiple “replicas”
- One npm/node problem: They don’t listen for proper shutdown signal by default

The Truth About The PID 1 Problem

- PID 1 (Process Identifier) is the first process in a system (or container) (AKA init)
- Init process in a container has two jobs:
  - reap zombie processes
  - pass signals to sub-processes
- Zombie not a big Node issue
- Focus on proper Node shutdown

Proper CMD for Healthy Shutdown

- Docker uses Linux signals to stop app (SIGINIT/SIGTERM/SIGKILL)
- SIGINIT/SIGTERM allow graceful stop
- npm doesn't respond to SIGINIT/SIGTERM
- node doesn't respond by default, but can with code
- Docker provides a init PID 1 replacement option

Proper Node Shutdown Options

- **Temp:** Use **--init** to fix ctrl-c for now
- **Workaround:** add **tini** to your image
- **Production:** your app captures **SIGINIT** for proper exit

Example init command

- Run any node app with --init to handle signals (temp solution)

```
> docker run --init -d nodeapp
```

Example tini Dockerfile

- Add tini to your Dockerfile, then use it in CMD (permanent workaround)

```
> RUN apk add --no-cache tini
> ENTRYPOINT ["/sbin/tini", "--"]
> CMD ["node", "./bin/www"]
```

1.Dockerfile

```
1  FROM node:12-slim
2
3  EXPOSE 3000
4
5  RUN mkdir /app && chown -R node:node /app
6
7  WORKDIR /app
8
9  USER node
10
11 COPY --chown=node:node package.json package-lock*.json ./
12
13 RUN npm install && npm cache clean --force
14
15 COPY --chown=node:node . .
16
17 CMD ["npm", "start"]
```

2.Dockerfile

```
FROM node:12-slim

ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /tini
RUN chmod +x /tini
ENTRYPOINT ["/tini", "--"]

EXPOSE 3000

RUN mkdir /app && chown -R node:node /app

WORKDIR /app

USER node

COPY --chown=node:node package.json package-lock*.json ./

RUN npm install && npm cache clean --force

COPY --chown=node:node . .

CMD ["node", "./bin/www"]
```

Example SIGINIT Capture

```
1  // quit on ctrl-c when running docker in terminal
2  process.on('SIGINT', function onSigint () {
3      console.info('Got SIGINT (aka ctrl-c in docker). Graceful shutdown ', new Date().toISOString());
4    shutdown();
5  });
6
7  // quit properly on docker stop
8  process.on('SIGTERM', function onSigterm () {
9    console.info('Got SIGTERM (docker container stop). Graceful shutdown ', new Date().toISOString());
10   shutdown();
11 });
12
13 // shut down server
14 function shutdown() {
15   server.close(function onServerClosed (err) {
16     if (err) {
17       console.error(err);
18       process.exitCode = 1;
19     }
20     process.exit();
21   });
22 }
```

```
1  // quit on ctrl-c when running docker in terminal
2  process.on('SIGINT', function onSigint () {
3      console.info('Got SIGINT (aka ctrl-c in docker). Graceful shutdown ', new Date().toIS
4    shutdown();
5  });
6
7  // quit properly on docker stop
8  process.on('SIGTERM', function onSigterm () {
9    console.info('Got SIGTERM (docker container stop). Graceful shutdown ', new Date().toIS
10   shutdown();
11 });
12
13 // shut down server
14 function shutdown() {
15   server.close(function onServerClosed (err) {
16     if (err) {
17       console.error(err);
18       process.exitCode = 1;
19     }
20     process.exit();
21   });
22 }
```

Better: Connection Tracking

- Used to track HTTP connections and send them FIN packets when Node shuts down
  - https://github.com/hunterloftis/stoppable

Multi-stage Builds

- Build multiple images from one file
- Those images can FROM each other
- COPY files between them
- Space + security benefits
- Great for "artifact only"
- Great for dev + test + prod

Avoiding devDependencies In Prod

- Multi-stage can solve this
- prod stages: **npm i --only=production**
- Dev stage: **npm i --only=development**
- Optional: Use **npm ci** to speed up builds
- Ensure **NODE_ENV** is set

2.Dockerfile

```
FROM node:12-slim

ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /tini
RUN chmod +x /tini
ENTRYPOINT ["/tini", "--"]

EXPOSE 3000

RUN mkdir /app && chown -R node:node /app

WORKDIR /app

USER node

COPY --chown=node:node package.json package-lock*.json ./

RUN npm install && npm cache clean --force

COPY --chown=node:node . .

CMD ["node", "./bin/www"]
```

3.Dockerfile

```
FROM node:12-slim as prod
ENV NODE=ENV=production
ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /tini
RUN chmod +x /tini
ENTRYPOINT ["/tini", "--"]
EXPOSE 3000
RUN mkdir /app && chown -R node:node /app
WORKDIR /app
USER node
COPY --chown=node:node package.json package-lock*.json ./
RUN npm ci && npm cache clean --force
COPY --chown=node:node . .
CMD ["node", "./bin/www"]

FROM prod as dev
ENV NODE_ENV=development
ENV PATH=/app/node_modules/.bin:$PATH
RUN npm install --only=development
# NOTE CHANGE ENTRYPOINT?
CMD ["nodemon", "./bin/www", "--inspect=0.0.0.0:9229"]
```

Building A Specific Stage

- To build dev image from dev (last) stage

```
> docker build -t myapp .
```

- To build prod image from prod stage

```
> docker build -t myapp:prod --target prod .
```

More Multi-stage: test

- Add a test stage that runs npm test
- Have CI build **--target test** stage before building prod
- Don't **COPY** code into dev stage
- Keep it DRY (for **COPY** and **RUN**)

3.Dockerfile

```
FROM node:12-slim as prod
ENV NODE=ENV=production
ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /tini
RUN chmod +x /tini
ENTRYPOINT ["/tini", "--"]
EXPOSE 3000
RUN mkdir /app && chown -R node:node /app
WORKDIR /app
USER node
COPY --chown=node:node package.json package-lock*.json ./
RUN npm ci && npm cache clean --force
COPY --chown=node:node . .
CMD ["node", "./bin/www"]

FROM prod as dev
ENV NODE_ENV=development
ENV PATH=/app/node_modules/.bin:$PATH
RUN npm install --only=development
# NOTE CHANGE ENTRYPOINT?
CMD ["nodemon", "./bin/www", "--inspect=0.0.0.0:9229"]
```

4.Dockerfile

```
FROM node:12-slim as base
ENV NODE=ENV=production
ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /tini
RUN chmod +x /tini
EXPOSE 3000
RUN mkdir /app && chown -R node:node /app
WORKDIR /app
USER node
COPY --chown=node:node package.json package-lock*.json ./
RUN npm ci && npm cache clean --force

FROM base as dev
ENV NODE_ENV=development
ENV PATH=/app/node_modules/.bin:$PATH
RUN npm install --only=development
CMD ["nodemon", "./bin/www", "--inspect=0.0.0.0:9229"]

FROM base as source
COPY --chown=node:node . .

FROM source as test
ENV NODE_ENV=development
ENV PATH=/app/node_modules/.bin:$PATH
COPY --from=dev /app/node_modules /app/node_modules
RUN eslint .
RUN npm test
CMD ["npm", "run", "test"]

FROM source as prod
ENTRYPOINT ["/tini", "--"]
CMD ["node", "./bin/www"]
```

4.Dockerfile

```
13  FROM base as dev
14  ENV NODE_ENV=development
15  ENV PATH=/app/node_modules/.bin:$PATH
16  RUN npm install --only=development
17  CMD ["nodemon", "./bin/www", "--inspect=0.0.0.0:9229"]
18
19  FROM base as source
20  COPY --chown=node:node . .
21
22  FROM source as test
23  ENV NODE_ENV=development
24  ENV PATH=/app/node_modules/.bin:$PATH
25  COPY --from=dev /app/node_modules /app/node_modules
26  RUN eslint .
27  RUN npm test
28  CMD ["npm", "run", "test"]
29
30  FROM source as prod
31  ENTRYPOINT ["/tini", "--"]
32  CMD ["node", "./bin/www"]
```

Security Scanning and Audit

- Create audit stage for optional build
- Consider **RUN npm audit**
- Consider CVE scanner
- Only report at first, no failing (most images have at least one CVE vuln)

./multi-stage-scanning/
https://snyk.io/blog/ten-npm-security-best-practices/
TODO recommend a few scanners
https://kubedex.com/container-scanning/
https://kubedex.com/follow-up-container-scanning-comparison/
https://sysdig.com/blog/container-security-docker-image-scanning/
https://sysdig.com/blog/20-docker-security-tools/

TODO check gareth’s slides for other tools

4.Dockerfile

```
13  FROM base as dev
14  ENV NODE_ENV=development
15  ENV PATH=/app/node_modules/.bin:$PATH
16  RUN npm install --only=development
17  CMD ["nodemon", "./bin/www", "--inspect=0.0.0.0:9229"]
18
19  FROM base as source
20  COPY --chown=node:node . .
21
22  FROM source as test
23  ENV NODE_ENV=development
24  ENV PATH=/app/node_modules/.bin:$PATH
25  COPY --from=dev /app/node_modules /app/node_modules
26  RUN eslint .
27  RUN npm test
28  CMD ["npm", "run", "test"]
29
30  FROM source as prod
31  ENTRYPOINT ["/tini", "--"]
32  CMD ["node", "./bin/www"]
```

4.Dockerfile

```
19  FROM base as source
20  COPY --chown=node:node . .
21
22  FROM source as test
23  ENV NODE_ENV=development
24  ENV PATH=/app/node_modules/.bin:$PATH
25  COPY --from=dev /app/node_modules /app/node_modules
26  RUN eslint .
27  RUN npm test
28  CMD ["npm", "run", "test"]
29
30  FROM source as prod
31  ENTRYPOINT ["/tini", "--"]
32  CMD ["node", "./bin/www"]
```

5.Dockerfile

```
19  FROM base as source
20  COPY --chown=node:node . .
21
22  FROM source as test
23  ENV NODE_ENV=development
24  ENV PATH=/app/node_modules/.bin:$PATH
25  COPY --from=dev /app/node_modules /app/node_modules
26  RUN eslint .
27  RUN npm test
28  CMD ["npm", "run", "test"]
29
30  FROM test as audit
31  USER root
32  RUN npm audit --audit-level critical
33  ARG MICROSCANNER_TOKEN
34  ADD https://get.aquasec.com/microscanner /
35  RUN chmod +x /microscanner
36  RUN /microscanner $MICROSCANNER_TOKEN --continue-on-failure
37
38  FROM source as prod
39  ENTRYPOINT ["/tini", "--"]
40  CMD ["node", "./bin/www"]
```

Got Compose?

docker con 19

Compose YAML v2 vs v3

- Myth busting: **v3 does not replace v2**
- **v2 focus:** single-node dev/test
- **v3 focus:** multi-node orchestration
- **If not using Swarm/Kubernetes, stick to v2**

Resources
https://docs.docker.com/compose/compose-file/compose-versioning/
https://github.com/docker/docker.github.io/pull/7593

Every Node Sample Compose

```yaml
version: '2.4'

services:
  node:
    build: .

  db:
    image: postgres
```

node_modules in Bind-Mounts

- **Problem:** we can't just bind-mount **node_modules** content from host on macOS/Windows (different arch)
- **Two Potential Solutions**

1. when we bind-mount, or host node_modules will overwrite what the image built
    1. fix: move node_modules in image and prevent node_modules in host from showing up using empty volume

node_modules in Bind-Mounts

- **Solution 1, common but less flexible:**
  - Bind-mount /app which includes modules
  - You can't **docker-compose up** until you've used **docker-compose run**
  - **node_modules** on host is now only usable from container
  - Never **npm install** from host

1. when we bind-mount, or host node_modules will overwrite what the image built
    1. fix: move node_modules in image and prevent node_modules in host from showing up using empty volume

node_modules in Bind-Mounts

- **Solution 2, more complex but flexible:**
  - Move **node_modules** up a directory in Dockerfile
  - Use empty volume to hide **node_modules** on bind-mount
  - **node_modules** on host doesn't conflict

Bind-Mounting: Performance

- On Linux, bind-mounts are native
- On macOS add **delegated** write mode
- Slower in Windows, mounting across Samba/SMB
  - Consider file sync if it gets real bad
  - Or WSL + Docker

macOS
https://blog.docker.com/2017/05/user-guided-caching-in-docker-for-mac/
https://docs.docker.com/docker-for-mac/osxfs/
https://docs.docker.com/docker-for-mac/osxfs-caching/
Windows
https://github.com/cweagans/docker-bg-sync
https://www.reddit.com/r/docker/comments/8hp6v7/setting_up_docker_for_windows_and_wsl_to_work/

0.docker-compose.yml

```yaml
version: '2.4'

services:
  node:
    build: .

  db:
    image: postgres
```

1.docker-compose.yml

```yaml
version: '2.4'

services:
  node:
    build:
      dockerfile: 5.Dockerfile
      context: .
      target: dev
    volumes:
      - .:/app:delegated
    ports:
      - "3000:3000"

  db:
    image: postgres
```

File Monitoring and Node Auto Restarts

- Use **nodemon** for compose file monitoring
- webpack-dev-server, etc. work the same
- If Windows, enable polling
- Create a **nodemon.json** for advanced workflows (bower, webpack, parcel)

https://github.com/remy/nodemon#config-files

Startup Order and Dependencies

- **Problem:** Multi-service apps start out of order, node might exit or cycle
- Multi-container dependencies need:
  - Name resolution (DNS)
  - Connection failure handling

Dependency Awareness

- **depends_on:** service A needs service B
- Fixes name resolution issues with "can't resolve <service_name>"
- Only for compose, not Orch
- compose YAML v2: works with healthchecks like a "wait for script"

1.docker-compose.yml

```yaml
version: '2.4'

services:
  node:
    build:
      dockerfile: 5.Dockerfile
      context: .
      target: dev
    volumes:
      - .:/app:delegated
    ports:
      - "3000:3000"

  db:
    image: postgres
```

2.docker-compose.yml

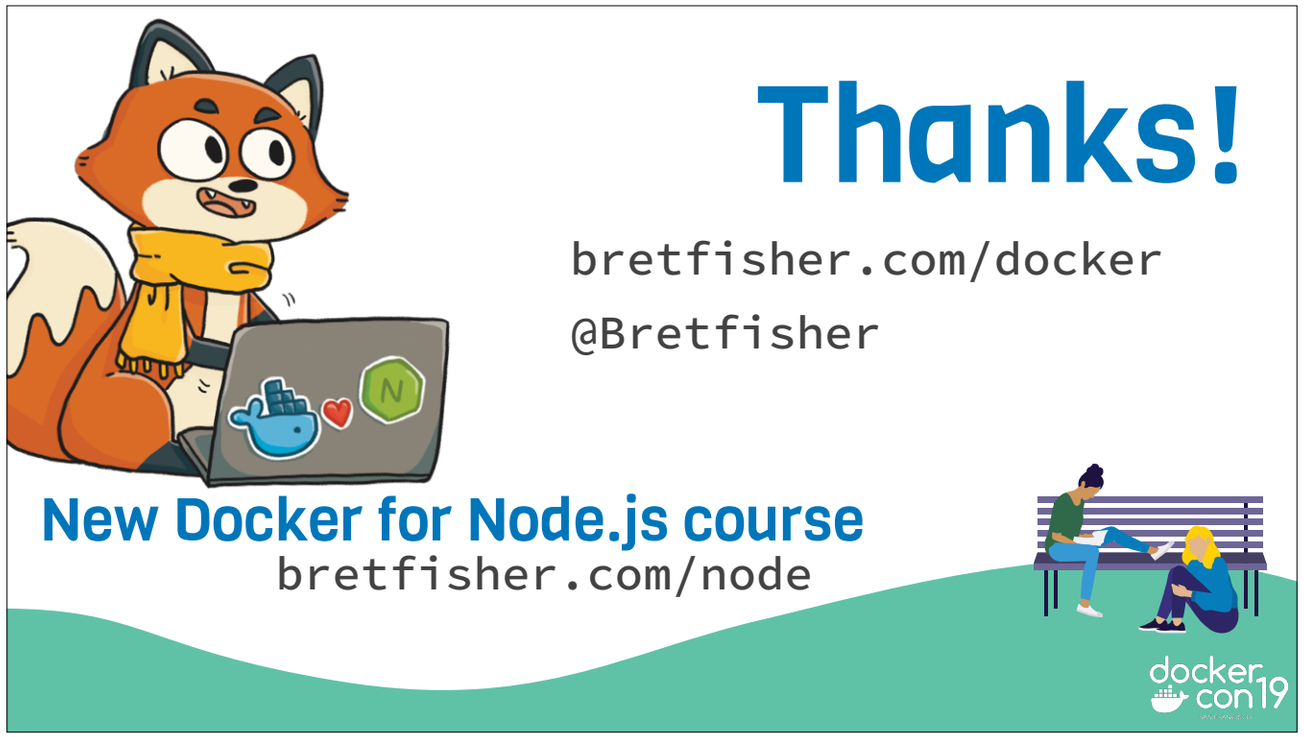
```yaml
version: '2.4'

services:
  node:
    build:
      dockerfile: 5.Dockerfile
      context: .
      target: dev
    volumes:
      - .:/app:delegated
    ports:
      - "3000:3000"
    depends_on:
      db:
        condition: service_healthy

  db:
    image: postgres
    healthcheck:
      test: pg_isready -U postgres -h 127.0.0.1
      interval: 5s
```

Production Checklist

- `CMD` node directly
- Build with `.dockerignore`
- capture `SIGTERM`, properly shutdown
- `npm ci` or `npm i --only=production`
- Scan/audit/test during builds
- Healthchecks (readiness/liveness)



resources: https://www.bretfisher.com/docker
new course coupon: https://www.bretfisher.com/node
https://www.twitter.com/BretFisher