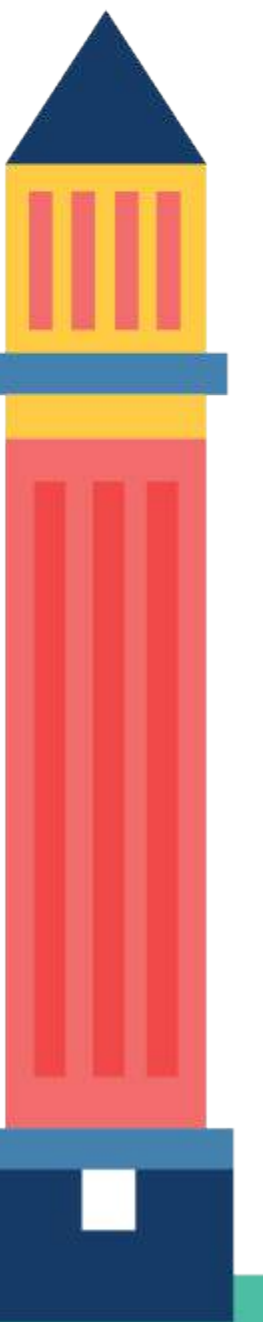
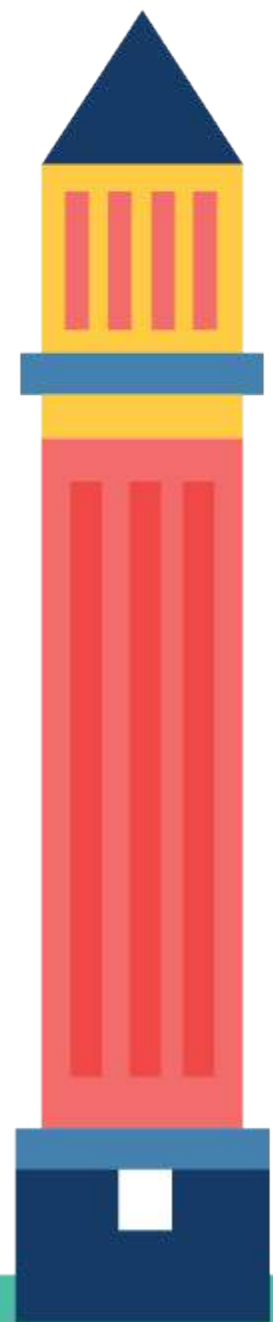




Laura Frank Tacho

@rhein_wein

Director of Engineering
CloudBees



CLOUD NATIVE TRAIL MAP

The Cloud Native Landscape [/cncf.io](https://landscape.cncf.io) has a large number of options. This Cloud Native Trail Map is a recommended process for leveraging open source, cloud native technologies. At each step, you can choose a vendor-supported offering or do it yourself, and everything after step #3 is optional based on your circumstances.

HELP ALONG THE WAY

A. Training and Certification

Consider training offerings from CNCF and then take the exam to become a Certified Kubernetes Administrator or a Certified Kubernetes Application Developer cncf.io/training

B. Consulting Help

If you want assistance with Kubernetes and the surrounding ecosystem, consider leveraging a Kubernetes Certified Service Provider

cncf.io/kcsp

C. Join CNCF's End User Community

For companies that don't offer cloud native services externally

cncf.io/enduser

WHAT IS CLOUD NATIVE?

Cloud-native technologies, such as containers and microservices, empower organizations to develop and deploy scalable,

1. CONTAINERIZATION

- Commonly done with Docker containers
- Any size application and dependencies (even PDP-11 code running on an emulator) can be containerized
- Over time, you should aspire towards splitting suitable applications and writing future functionality as microservices

3. ORCHESTRATION & APPLICATION DEFINITION

- Kubernetes is the market-leading orchestration solution
- You should select a Certified Kubernetes Distribution, Hosted Platform, or Installer: cncf.io/ck
- Helm Charts help you define, install, and upgrade even the most complex Kubernetes application



5. SERVICE MESH AND DISCOVERY

- CoreDNS is a fast and flexible tool that is useful for service discovery
- Envoy and Linkerd each enable service mesh architectures
- They offer health checking, routing, and load balancing



2. CI/CD

- Setup Continuous Integration/Continuous Delivery (CI/CD) so that changes to your source code automatically result in a new container being built, tested, and deployed to staging and eventually, perhaps, to production
- Setup automated rollouts, roll backs and testing

4. OBSERVABILITY & ANALYSIS

- Pick solutions for monitoring, logging and tracing
- Consider CNCF projects Prometheus for monitoring, Fluentd for logging and Jaeger for Tracing
- For tracing, look for an OpenTracing-compatible implementation like Jaeger



6. NETWORKING

To enable more flexible networking, use a CNI-compliant network project like Calico, Flannel, or Weave Net.

- 1 DOCKER COMPOSE BASICS**
- 2 OPTIMIZING IMAGES**
- 3 DEBUGGING IN CONTAINERS**

1 DOCKER COMPOSE BASICS

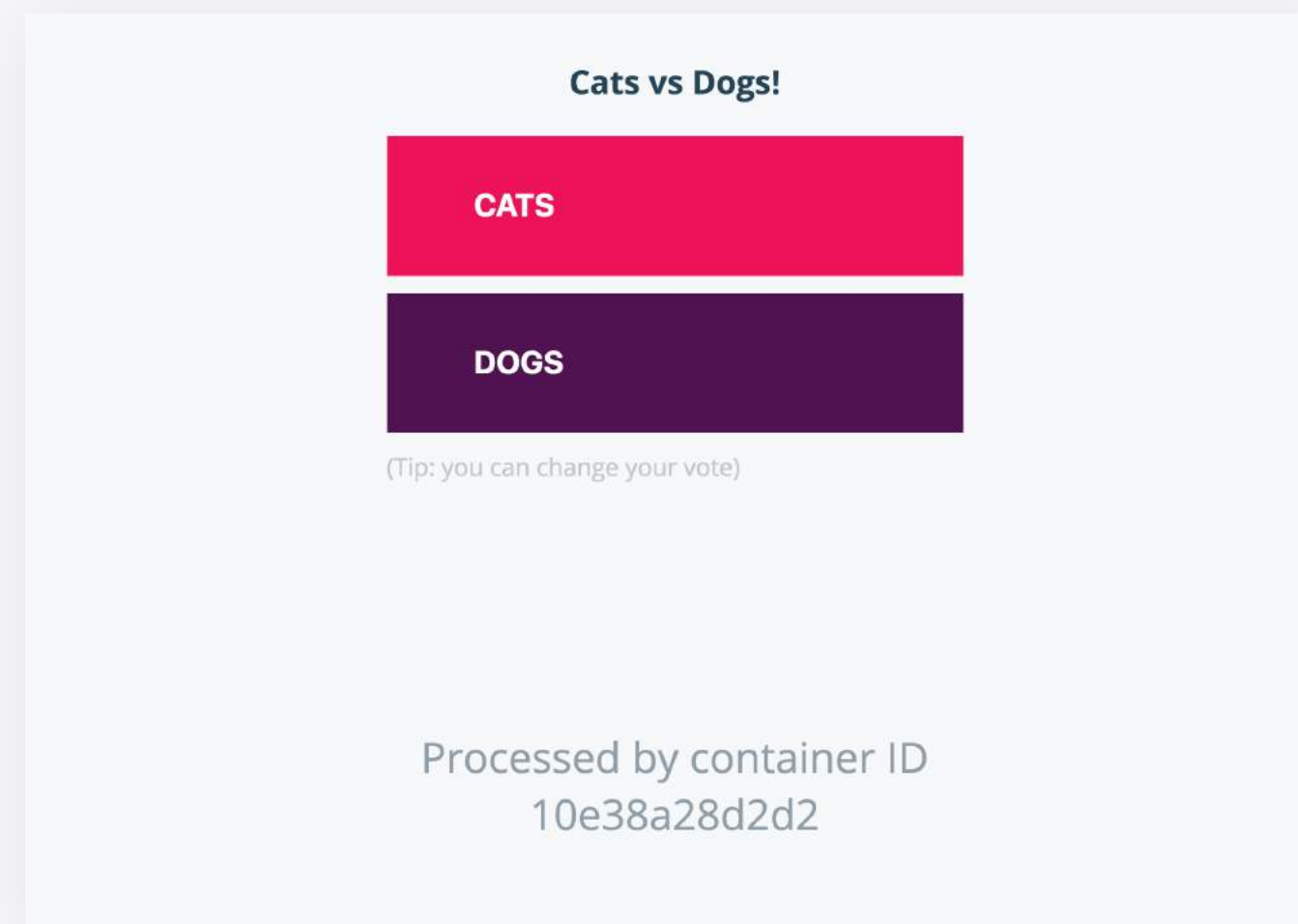
Developer Tools

- Docker Compose
- Docker Desktop
 - Docker for Mac
 - Docker for Windows
 - Easiest way to run Docker and Kubernetes on your local machine

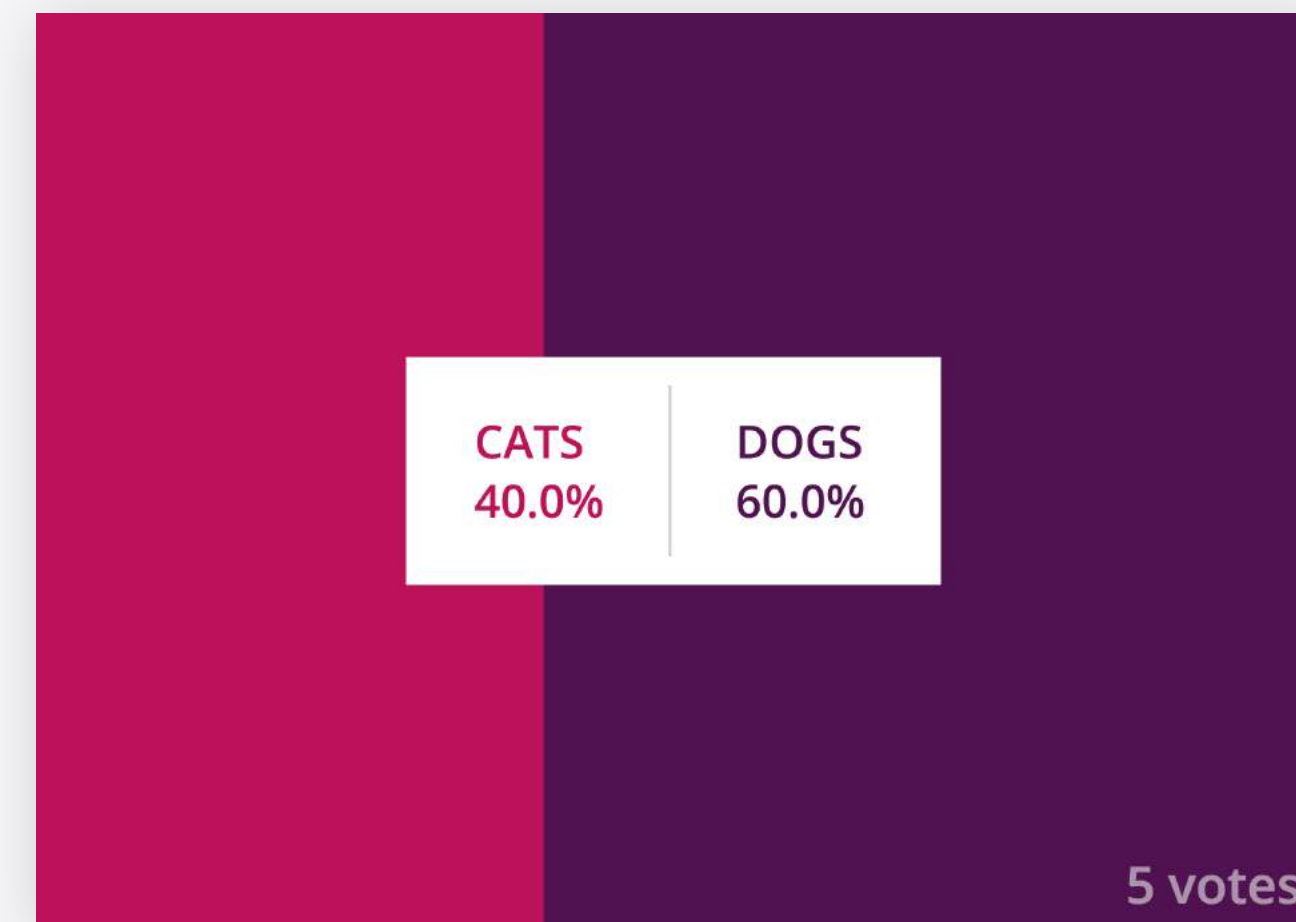
<https://www.docker.com/products/docker-desktop>

Sample App: Cats vs Dogs

github.com/rheinwein/example-voting-app



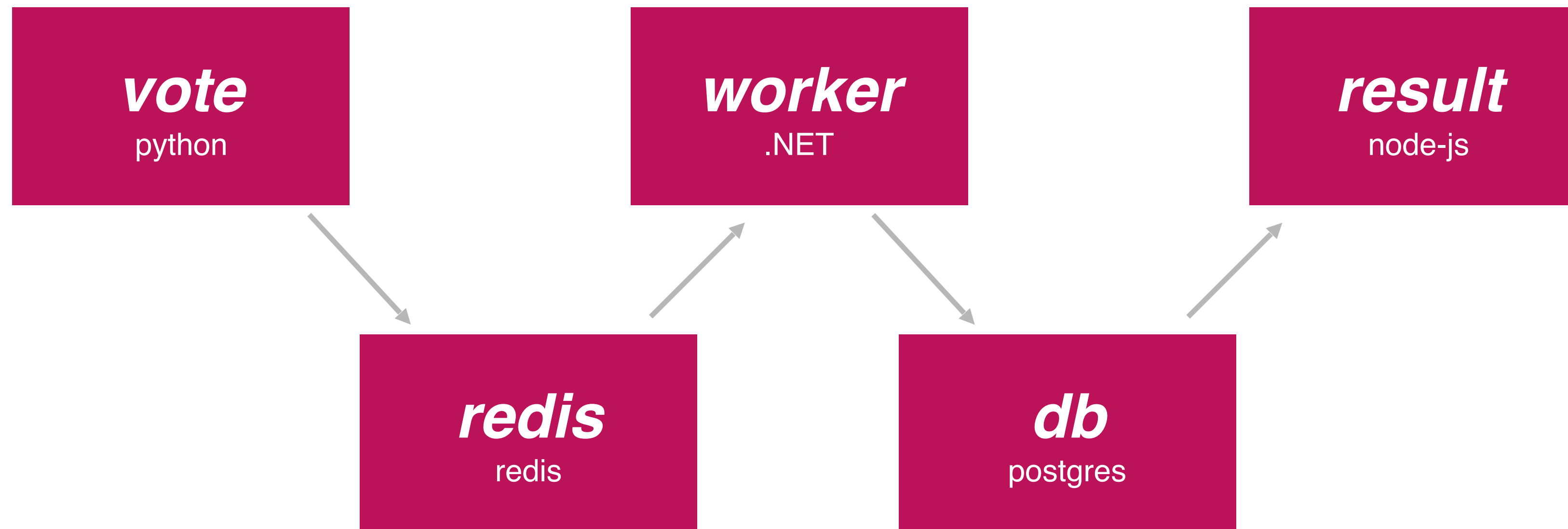
vote



result

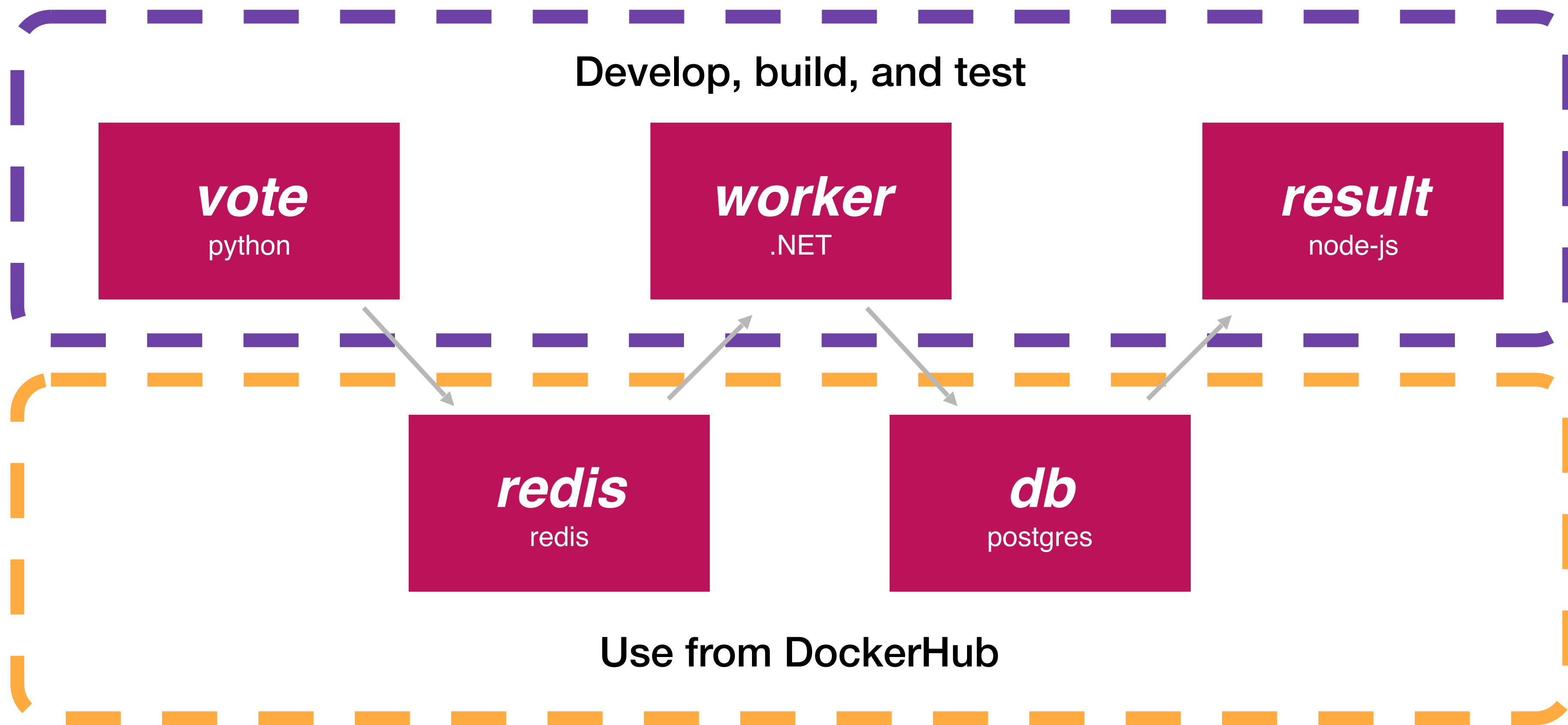
Sample App: Cats vs Dogs

github.com/rheinwein/example-voting-app



Sample App: Cats vs Dogs

github.com/rheinwein/example-voting-app




```
example-voting-app $ docker-compose ps
```

Name	Command	State	Ports
db	docker-entrypoint.sh postgres	Up	5432/tcp
example-voting-app_result_1	nodemon --inspect=[::]:585 ...	Up	0.0.0.0:5858->5858/tcp, 0.0.0.0:5001->80/tcp
example-voting-app_vote_1	python app.py	Up	0.0.0.0:5000->80/tcp
example-voting-app_worker_1	/bin/sh -c dotnet src/Work ...	Up	
redis	docker-entrypoint.sh redis ...	Up	0.0.0.0:32774->6379/tcp

```
example-voting-app $ docker ps # filtering output
```

CONTAINER ID	IMAGE	COMMAND	STATUS	PORTS
7852e34c057f	example-voting-app_worker	"/bin/sh -c 'dotnet ...'"	Up 6 minutes	
3bca5cc2eb09	postgres:9.4	"docker-entrypoint.s..."	Up 6 minutes	5432/tcp
b0d3c9cd007c	example-voting-app_vote	"python app.py"	Up 6 minutes	0.0.0.0:5000->80/tcp
9702bb12b70b	redis:alpine	"docker-entrypoint.s..."	Up 6 minutes	0.0.0.0:32774->6379/tcp
072da9ac67e3	example-voting-app_result	"nodemon --inspect=[..."	Up 6 minutes	0.0.0.0:5858->5858/tcp

```
example-voting-app $ docker-compose ps
```

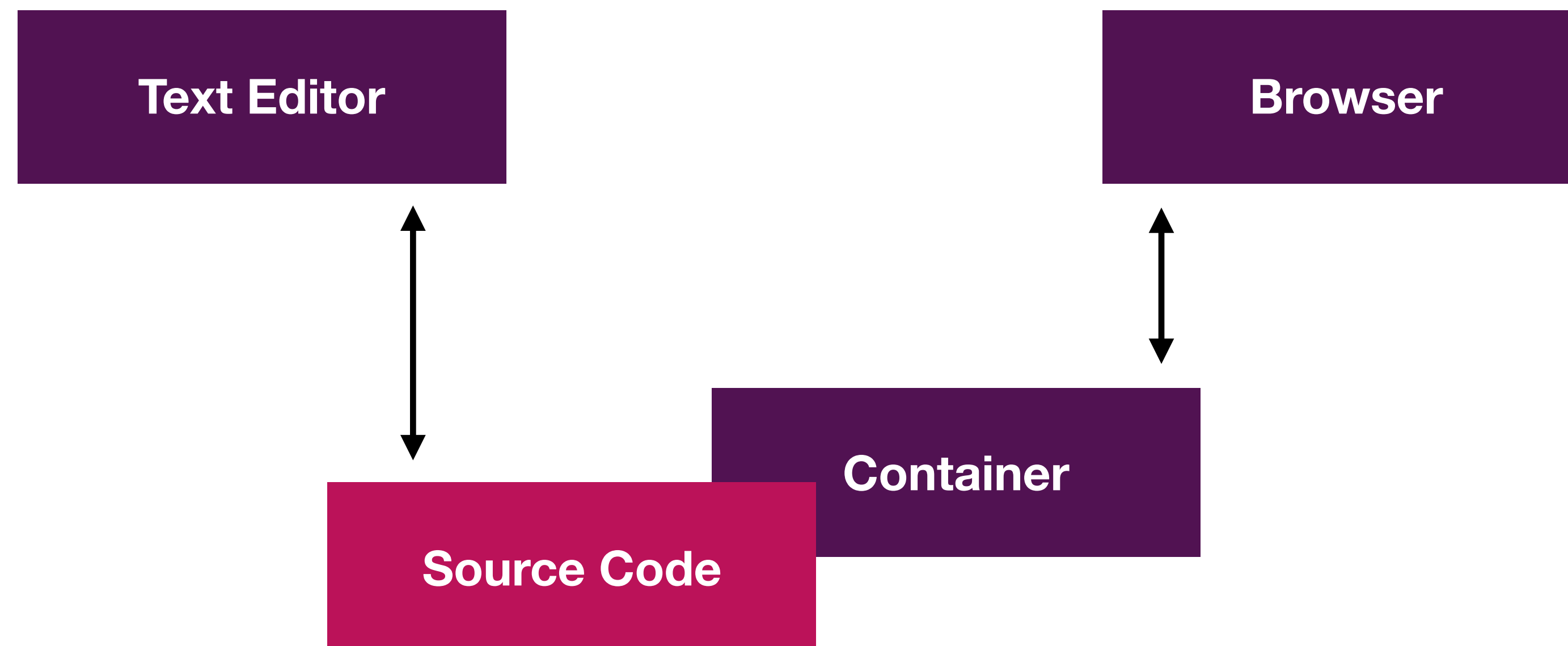
Name	Command	State	Ports
db	docker-entrypoint.sh postgres	Up	5432/tcp
example-voting-app_result_1	nodemon --inspect=[::]:585 ...	Up	0.0.0.0:5858->5858/tcp, 0.0.0.0:5001->80/tcp
example-voting-app_vote_1	python app.py	Up	0.0.0.0:5000->80/tcp
example-voting-app_worker_1	/bin/sh -c dotnet src/Work ...	Up	
redis	docker-entrypoint.sh redis ...	Up	0.0.0.0:32774->6379/tcp

```
example-voting-app $ docker ps # filtering output
```

CONTAINER ID	IMAGE	COMMAND	STATUS	PORTS
7852e34c057f	example-voting-app_worker	"/bin/sh -c 'dotnet ...'"	Up 6 minutes	
3bca5cc2eb09	postgres:9.4	"docker-entrypoint.s..."	Up 6 minutes	5432/tcp
b0d3c9cd007c	example-voting-app_vote	"python app.py"	Up 6 minutes	0.0.0.0:5000->80/tcp
9702bb12b70b	redis:alpine	"docker-entrypoint.s..."	Up 6 minutes	0.0.0.0:32774->6379/tcp
072da9ac67e3	example-voting-app_result	"nodemon --inspect=[..."	Up 6 minutes	0.0.0.0:5858->5858/tcp

Access local code

See changes in the
browser immediately

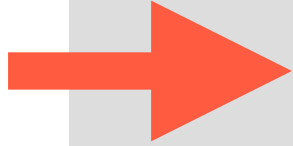


Accessing Local Code

Runtime vs Buildtime

This will overwrite the code copied into the image at buildtime. Now you're able to edit local code and have those changes reflected in the container

docker-compose.yml




```
version: "3"

services:
  result:
    build: ./result
    command: nodemon server.js
    volumes:
      - ./result:/app
    ports:
      - "5001:80"
      - "5858:5858"
    networks:
      - front-tier
      - back-tier
```


Managing External Dependencies

docker-compose.yml



```
version: "3"

services:
  redis:
    image: redis:alpine
    container_name: redis
    ports:
      - "6379"
    networks:
      - back-tier
```

vote/app.py

```
def get_redis():
    if not hasattr(g, 'redis'):
        g.redis = Redis(host="redis", db=0, socket_timeout=5)
    return g.redis
```




You don't need to run your
database in a container.

Data Volumes

- If you store application data in the container, that data is lost when the container is stopped and removed
- Using volumes allows data to persist beyond the lifecycle of the container
 - Drivers are available for cloud services
 - You can also use your local disk

docker-compose.yml

```
services:
  ...
  db:
    image: postgres:9.4
    container_name: db
    volumes:
      - "db-data:/var/lib/postgresql/data"
    networks:
      - back-tier

volumes:
  db-data:
```

2 OPTIMIZING IMAGES

Dockerfiles and container images are the heart of your application.

Everything else in the container ecosystem assumes that you have your source code available as a container time.



result/Dockerfile

A Dockerfile a list of instructions outlining packages, files, dependencies, and everything required for your application to run.

```
FROM node:10.9-alpine

RUN mkdir -p /app
WORKDIR /app

RUN npm install -g nodemon
RUN npm config set registry https://registry.npmjs.org
COPY package.json /app/package.json
RUN npm install \
    && npm ls \
    && npm cache clean --force \
    && mv /app/node_modules /node_modules
COPY . /app

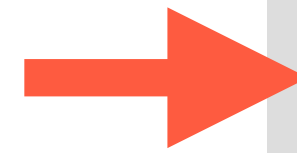
ENV PORT 80
EXPOSE 80

CMD ["node", "server.js"]
```

Depending on your language, select a suitable base image and then install the packages and run setup commands necessary for your application.

In general, aim to have as small of an image as possible, but you can always optimize later.

result/Dockerfile



```
FROM node:10.9-alpine

RUN mkdir -p /app
WORKDIR /app

RUN npm install -g nodemon
RUN npm config set registry https://registry.npmjs.org
COPY package.json /app/package.json
RUN npm install \
  && npm ls \
  && npm cache clean --force \
  && mv /app/node_modules /node_modules
COPY . /app

ENV PORT 80
EXPOSE 80

CMD ["node", "server.js"]
```

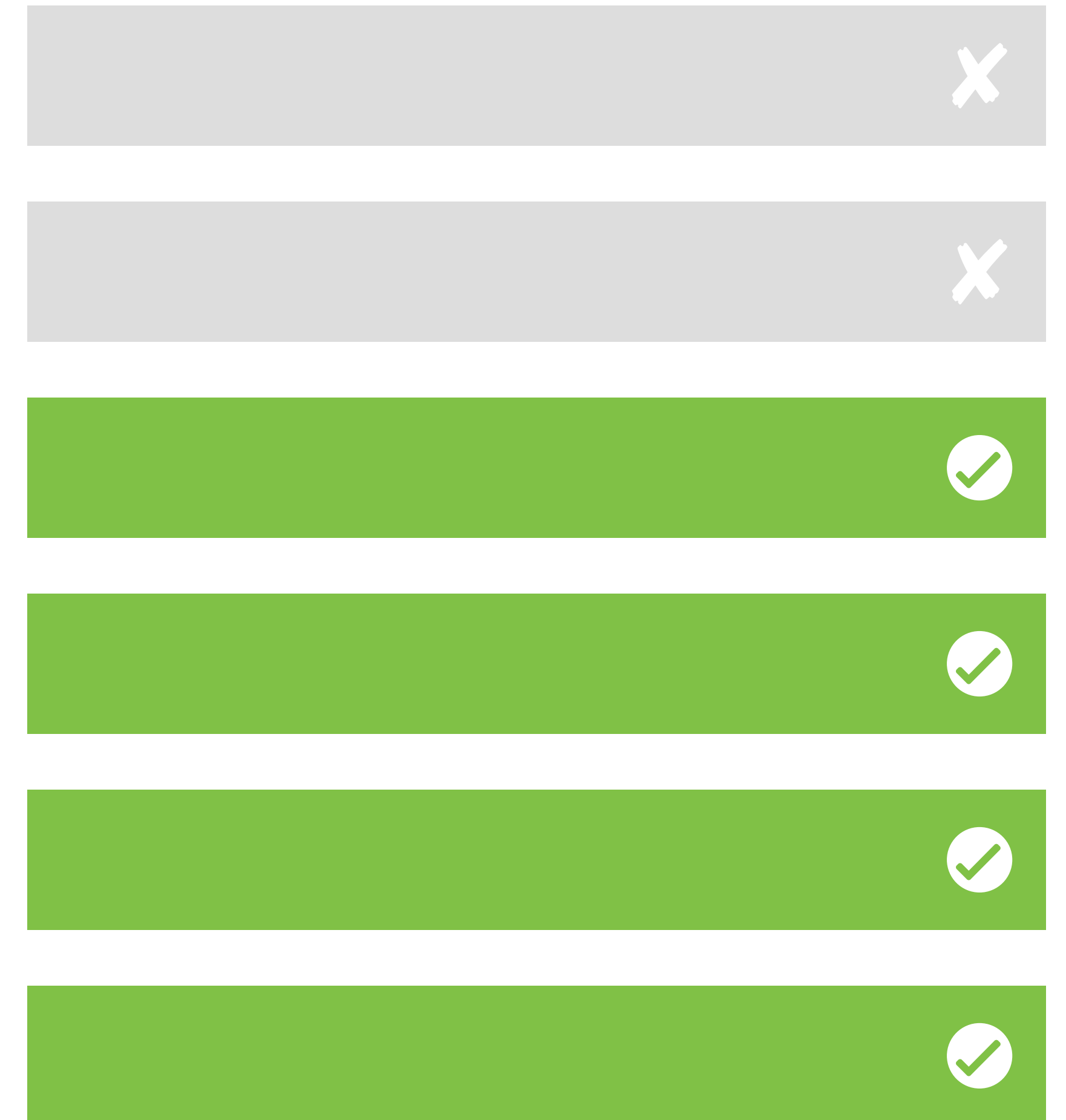


For faster dev cycles, optimize
your images.

Use Caching Effectively

If nothing changes in a layer command, Docker will use an existing version.

At buildtime, these cache hits can drastically reduce the time to build and push your image. Cache misses will add extra time, and you can optimize your image to avoid them during development.



Order Layers for Better Cache Usage

Copy over package or dependency manifests and install those before copying the rest of the code. Lockfiles change infrequently, so it's not necessary to reinstall everything all the time.

```
COPY package.json /app/package.json
RUN npm install \
  && npm ls \
  && npm cache clean --force \
  && mv /app/node_modules /node_modules
COPY . /app
```

Dockerfile Best Practices with Tibor and Sebastiaan

Watch the recording when it's available!

Pick Base Images Wisely!

By now, almost every major language or framework maintains a slim version of their library image, tagged with **slim** or specifically with **alpine**.

Warning: the size of an image on Docker Hub is the *compressed* size.

hub.docker.com/_/node

Full Description

Supported tags and respective `Dockerfile` links

- 8.11.4-jessie, 8.11-jessie, 8-jessie, carbon-jessie, 8.11.4, 8.11, 8, carbon ([8/jessie/Dockerfile](#))
- 8.11.4-alpine, 8.11-alpine, 8-alpine, carbon-alpine ([8/alpine/Dockerfile](#))
- 8.11.4-onbuild, 8.11-onbuild, 8-onbuild, carbon-onbuild ([8/onbuild/Dockerfile](#))
- 8.11.4-slim, 8.11-slim, 8-slim, carbon-slim ([8/slim/Dockerfile](#))
- 8.11.4-stretch, 8.11-stretch, 8-stretch, carbon-stretch ([8/stretch/Dockerfile](#))
- 6.14.4-jessie, 6.14-jessie, 6-jessie, boron-jessie, 6.14.4, 6.14, 6, boron ([6/jessie/Dockerfile](#))
- 6.14.4-alpine, 6.14-alpine, 6-alpine, boron-alpine ([6/alpine/Dockerfile](#))
- 6.14.4-onbuild, 6.14-onbuild, 6-onbuild, boron-onbuild ([6/onbuild/Dockerfile](#))
- 6.14.4-slim, 6.14-slim, 6-slim, boron-slim ([6/slim/Dockerfile](#))
- 6.14.4-stretch, 6.14-stretch, 6-stretch, boron-stretch ([6/stretch/Dockerfile](#))
- 10.9.0-jessie, 10.9-jessie, 10-jessie, jessie, 10.9.0, 10.9, 10, latest ([10/jessie/Dockerfile](#))
- 10.9.0-alpine, 10.9-alpine, 10-alpine, alpine ([10/alpine/Dockerfile](#))
- 10.9.0-slim, 10.9-slim, 10-slim, slim ([10/slim/Dockerfile](#))
- 10.9.0-stretch, 10.9-stretch, 10-stretch, stretch ([10/stretch/Dockerfile](#))
- chakracore-8.11.1, chakracore-8.11, chakracore-8 ([chakracore/8/Dockerfile](#))
- chakracore-10.6.0, chakracore-10.6, chakracore-10, chakracore ([chakracore/10/Dockerfile](#))

Avoid Anti-Patterns

- *Simultaneous Innovation*
- Skipping the fundamentals like giving love to your Dockerfiles
- Forgetting other development best practices like pinning dependencies to versions — this stuff still matters even if you're using containers
 - For example, don't use the **latest** tag

3 DEBUGGING IN CONTAINERS

Viewing Log Output

You can access log output either via Docker Compose or Docker directly.

```
$ docker-compose logs # this returns all logs for all services
db          | LOG: stats_timestamp 2018-09-12 10:35:03.286676+00 is later than collector's time
           | 2018-09-12 10:35:03.262932+00 for database 12141
result_1    | [nodemon] restarting due to changes...
result_1    | [nodemon] starting `node --inspect=[::]:5858 server.js`
result_1    | Debugger listening on ws://[::]:5858/f6dd534f-1976-421c-b4e2-8d3b1baf2622
result_1    | For help, see: https://nodejs.org/en/docs/inspector
result_1    | App running on port 80
result_1    | Connected to db
result_1    | Debugger attached.
redis       | 1:M 12 Sep 08:33:52.991 # WARNING: The TCP backlog setting of 511 cannot be enforced
           | because /proc/sys/net/core/somaxconn is set to the lower value of 128.
redis       | 1:M 12 Sep 08:33:52.991 # Server initialized
```

Viewing Log Output

Logs are visible by default when starting your app with `docker-compose up`. If you prefer a tidy terminal, use `up` with the `-d` flag. You can get logs per service later.

Tip: timestamps are available with a flag.

```
$ docker-compose up -d
...
$ docker-compose logs -t worker # docker logs -t example-voting-app_worker_1
worker_1 | 2018-09-12T08:22:49.367373200Z Waiting for db
worker_1 | 2018-09-12T08:22:50.371369500Z Waiting for db
worker_1 | 2018-09-12T08:22:53.526120800Z Connected to db
worker_1 | 2018-09-12T08:22:53.541336500Z Found redis at 172.20.0.2
worker_1 | 2018-09-12T08:22:53.541400100Z Connecting to redis
worker_1 | 2018-09-12T08:33:54.478754500Z Connected to db
worker_1 | 2018-09-12T08:33:54.495682400Z Found redis at 172.20.0.2
worker_1 | 2018-09-12T08:33:54.496062600Z Connecting to redis
worker_1 | 2018-09-12T08:34:15.528315300Z Processing vote for 'b' by '2d4e42c6c94f4e53'
worker_1 | 2018-09-12T08:37:11.883353600Z Processing vote for 'a' by '2d4e42c6c94f4e53'
```


Starting a Container in Interactive Mode

Want a shell to debug startup commands without the container exiting on failure?

with docker-compose run web

```
version: "3"

services:
  web:
    image: ubuntu:18.04
    tty: true
    stdin_open: true
    entrypoint: "/bin/bash"
```

with Docker only

```
docker run -it ubuntu:18:04 /bin/bash

# volume mount if you need to with `-v`
source:target`
```

Helpful Docker Commands

- **docker top** will give you a list of all running processes in a container
- **docker inspect (name | id)** will give you ALL of the details about a given resource
 - You can use the **--format** flag and pass in a Go template
 - For example, to get the IP(s) of a running container:

```
docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' \
$CONTAINER_ID
```

Running a Debugger

Maintain access to debugging tools by publishing the ports.

You may also have to update the command or entrypoint to access debugging tools.

```
version: "3"

services:
  result:
    build: ./result
    command: nodemon --inspect=0.0.0.0:5858 server.js
    volumes:
      - ./result:/app
    ports:
      - "5001:80"
      - "5858:5858"
    networks:
      - front-tier
      - back-tier
```

