DAILY DEVELOPMENT WITH DOCKER DESKTOP

https://github.com/JimCodifed/linkextractor

Jim Armstrong
@JDArmstro

WHAT'S THE PLAN?

1. We have a python script!
   - It has issues.
2. Containerize it!
3. Let's make our app more useful.
4. What if it was an API?
5. Add a front-end for users

You don't need to be a Python programmer to complete this workshop!

☆ Extra credit (time remaining): A quick peek at a Java project

THE TOOLS WE'LL USE

- Docker Desktop for macOS or Windows
  - Docker Engine if you're on Linux is fine, too
- Your code editor of choice (I'm using VS Code)
- Git
  - Don't have git?

    `git-scm.com/download`

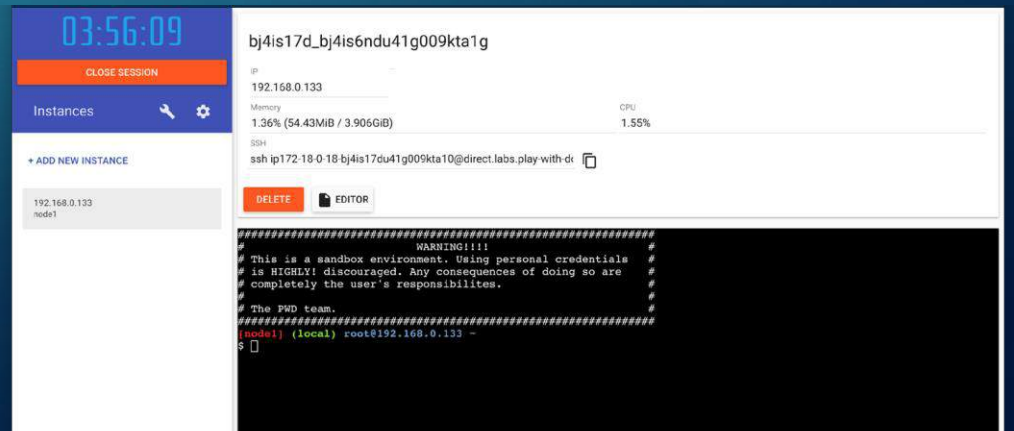
    Or you can download the app as a zip file

If you cannot run Docker Desktop (or don't have time to download and install on the conference wifi):

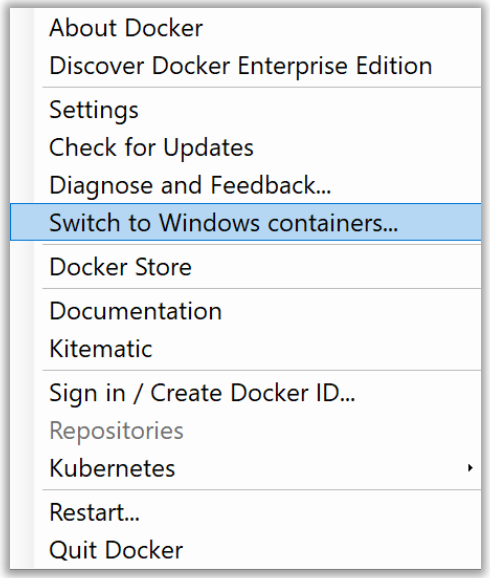
1. Go to

   https://labs.play-with-docker.com

2. Login (docker hub account)
3. "Add new instance"

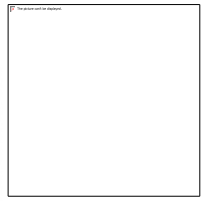


Windows only



This lab has only been tested on Linux

Windows & Mac

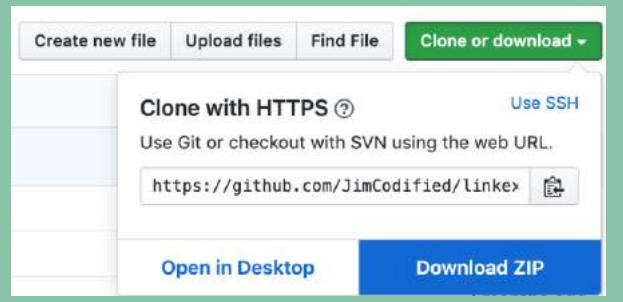


Everything *should* run fine on Kube, but the docker commands shown are for Swarm

DOCKER DESKTOP PREPARATION

LET'S BEGIN

If you don't have git:
Browse to
https://github.com/JimCodified/linkextractor




```
$ git clone https://github.com/JimCodified/linkextractor

$ cd linkextractor

$ git checkout demo

$ git checkout step0

.
├── README.md
└── linkextractor.py
```

OUR SUPER HANDY PYTHON APP

```
$ code .
```

linkextractor.py

```
1   #!/usr/bin/env python
2
3   import sys
4   import requests
5   from bs4 import BeautifulSoup
6
7   res = requests.get(sys.argv[-1])
8   soup = BeautifulSoup(res.text, "html.parser")
9   for link in soup.find_all("a"):
10      print(link.get("href"))
11
```

GIVE IT A SPIN…MAYBE

```
$ linkextractor.py


$ ./linkextractor.py


$ python ./linkextractor.py
```

WHAT HAPPENED?

- It’s not executable
- We could change permissions…
- …or run it via *python*

- But chances are slim any of those will work because dependencies

CONTAINERS TO THE RESCUE!

If you don't want to type out the Dockerfile:
`$ cp cheats/Dockerfile .`

```
FROM        python:3.7-alpine
LABEL       maintainer="<your name>"

RUN         pip install beautifulsoup4
RUN         pip install requests

WORKDIR     /app
COPY        linkextractor.py /app/
RUN         chmod a+x linkextractor.py

ENTRYPOINT [”python”, “./linkextractor.py"]
```

```
$ docker build -t linkextractor:v1 .
```

WHAT???

```
Sending build context to Docker daemon  137.7kB
Step 1/8 : FROM       python:3.7-alpine
 ---> 954987809e63
Step 2/8 : LABEL      maintainer="Jim A. <jim.armstrong@docker.com>"
 ---> Running in b49b516a82e2
Removing intermediate container b49b516a82e2
 ---> 310010a40d22
Step 3/8 : RUN        pip install beautifulsoup4
 ---> Running in 87a356857ea4
Collecting beautifulsoup4
  Downloading
https://files.pythonhosted.org/packages/1d/5d/3260694a59df0ec52f8b4883f5d23b130b
c237602a1411fa670eae12351e/beautifulsoup4-4.7.1-py3-none-any.whl (94kB)
Collecting soupsieve>=1.2 (from beautifulsoup4)
  Downloading
https://files.pythonhosted.org/packages/b9/a5/7ea40d0f8676bde6e464a6435a48bc5db0
9b1a8f4f06d41dd997b8f3c616/soupsieve-1.9.1-py2.py3-none-any.whl
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.7.1 soupsieve-1.9.1
Removing intermediate container 87a356857ea4
 ---> 5b86d63727e1
Step 4/8 : RUN        pip install requests
 ---> Running in 092ea643f938
Collecting requests
  Downloading
```

IT WORKS!

```
$ docker run --rm linkextractor:v1 http://docker.com
/dockercon
/dockercon/register-livestream
/
/why-docker
/resources/what-container
/company
/partners
/products
/products/docker-enterprise
/products/docker-hub
/products/developer-tools
/products/docker-desktop
/products/container-runtime
/products/kubernetes
```

```
$ docker image ls linkextractor

REPOSITORY                                                  TAG
        IMAGE ID            CREATED                 SIZE
linkextractor                                               v1
        2e44d06ccfb0        9 minutes ago           95.9MB
```

CODE-BUILD-TEST

linkextractor.py

```
#!/usr/bin/env python

import sys
import requests
from bs4 import BeautifulSoup

res = requests.get(sys.argv[-1])
soup = BeautifulSoup(res.text, "html.parser")
for link in soup.find_all("a"):
    print("-->", link.get("href"))
```

Add a decoration to the start of each line

THE DOCKER BUILD CACHE

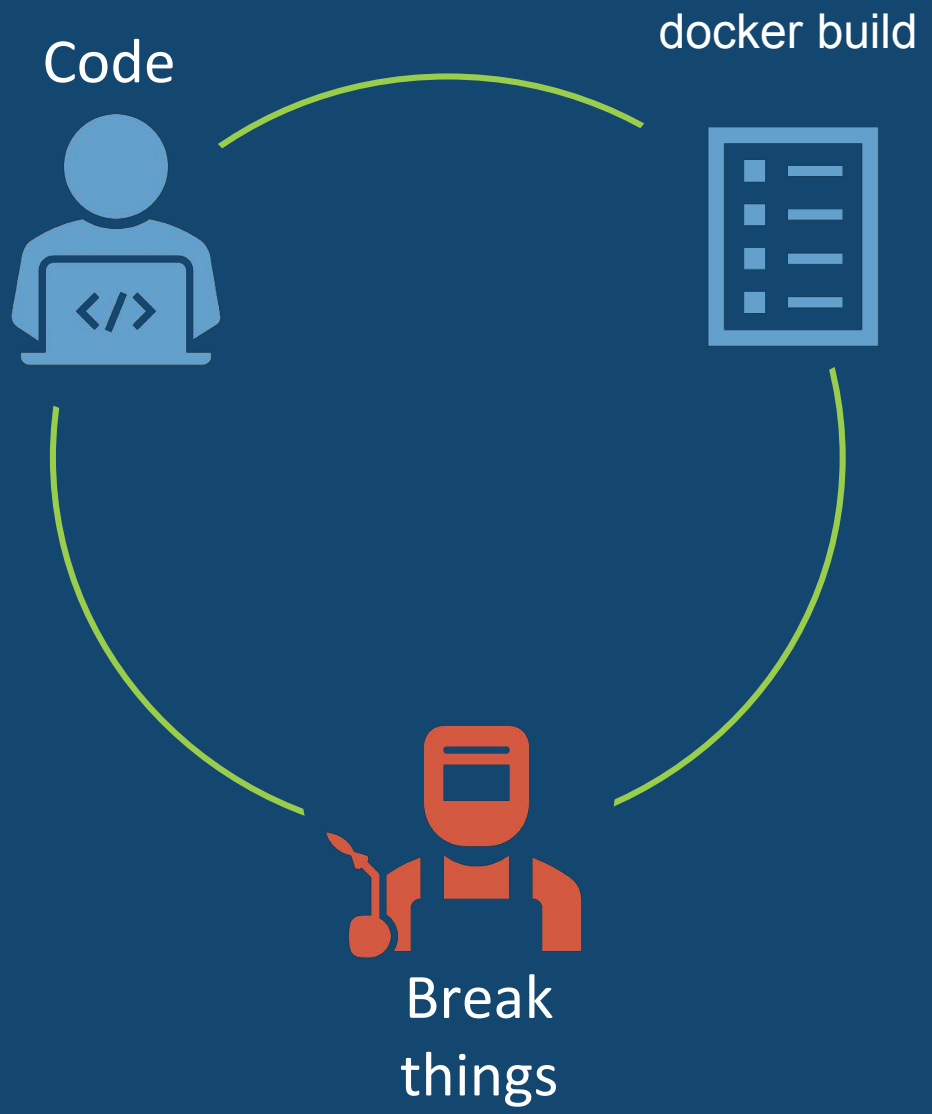
Check out Buildkit for real caching goodness!

```
$ docker build -t linkextractor:v1.1 .
.
.
.
Step 4/8 : RUN         pip install requests
 ---> Using cache
 ---> 8da10403b259
Step 5/8 : WORKDIR     /app
 ---> Using cache
 ---> 6085c3cc3baf
Step 6/8 : COPY        linkextractor.py /app/
 ---> 5ecc75f20edd
```

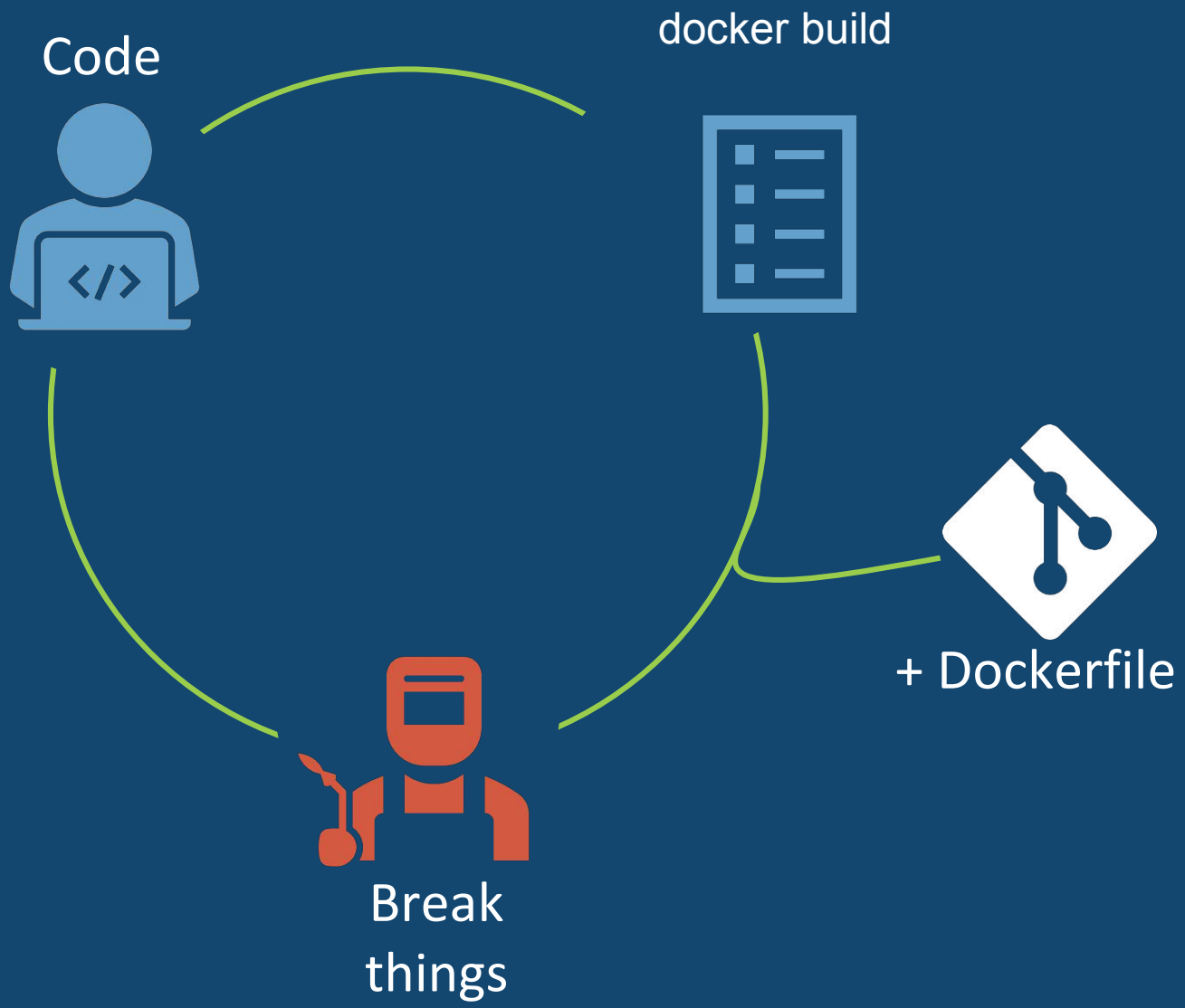
NEW & IMPROVED (?) LINK EXTRACTOR

```
$ docker container run --rm linkextractor:v1.1
  https://docker.com
--> /dockercon
--> /dockercon/register-livestream
--> /
--> /why-docker
--> /resources/what-container
--> /company
--> /partners
--> /products
--> /products/docker-enterprise
--> /products/docker-hub
--> /products/developer-tools
--> /products/docker-desktop
--> /products/container-runtime
```

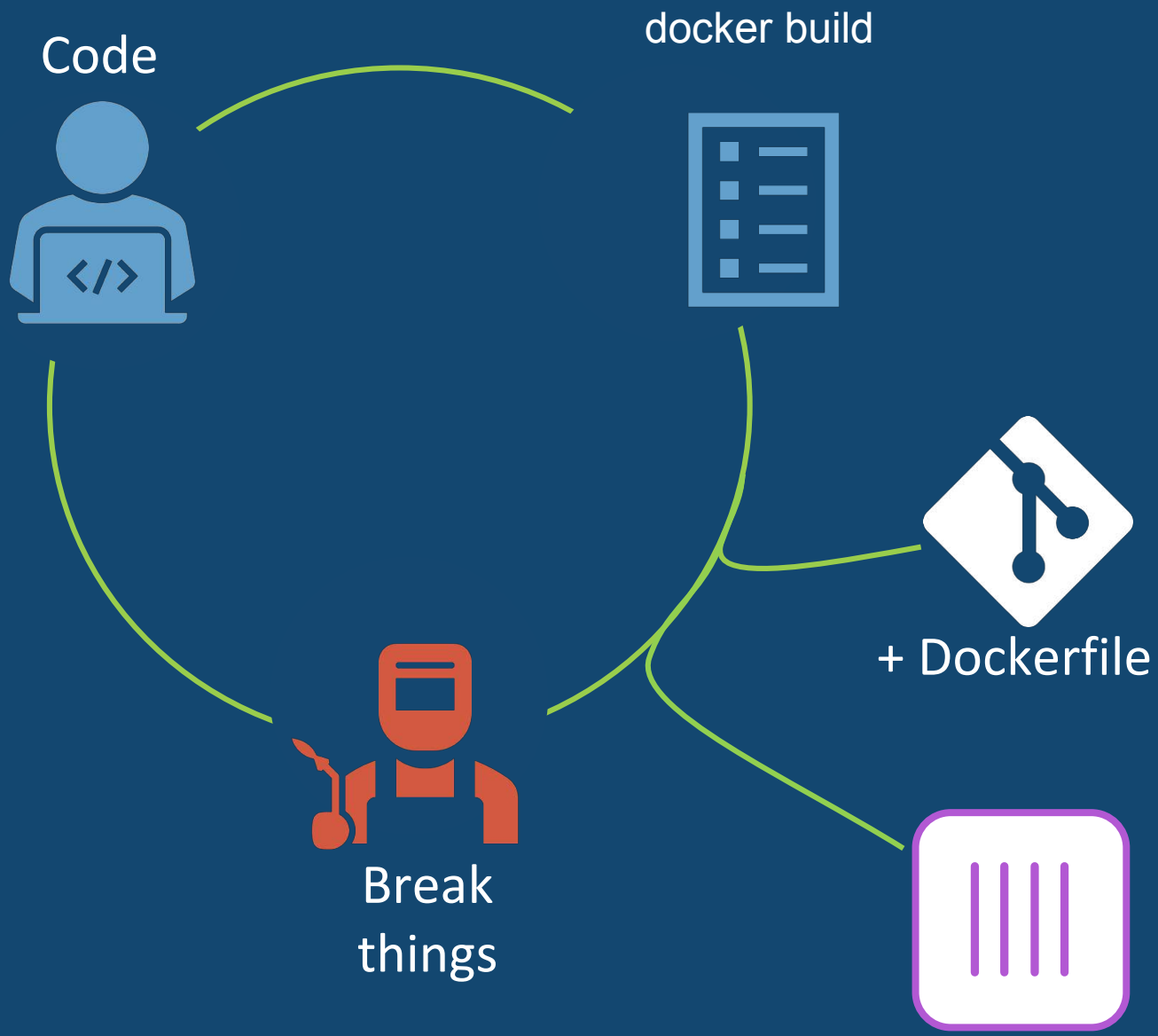
OUR DEV-TEST LOOP SO FAR

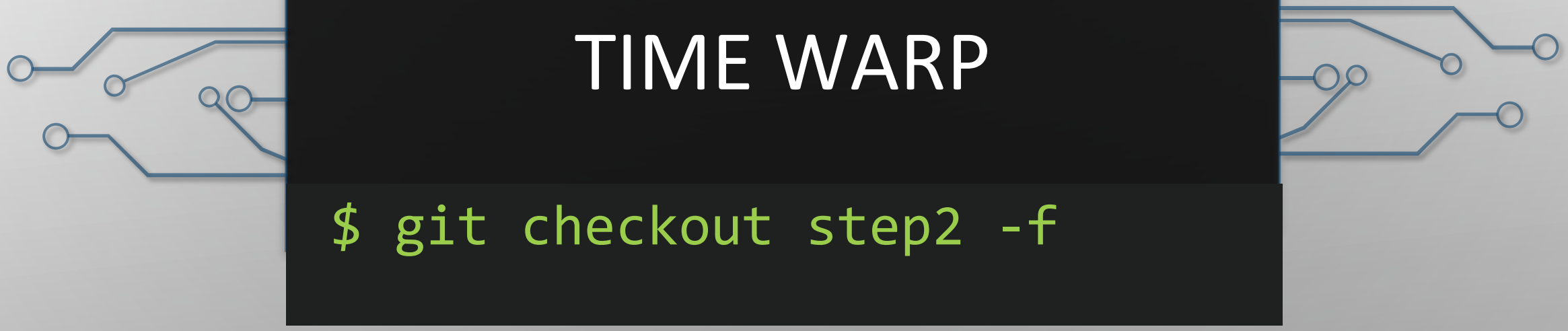


OUR DEV-TEST LOOP SO FAR



OUR DEV-TEST LOOP SO FAR




TIME WARP
$ git checkout step2 -f

linkextractor.py

```python
#!/usr/bin/env python

import sys
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin

def extract_links(url):
    res = requests.get(url)
    soup = BeautifulSoup(res.text, "html.parser")
    base = url
    # TODO: Update base if a <base> element is present with the href attr
    links = []
    for link in soup.find_all("a"):
        links.append({
            "text": " ".join(link.text.split()) or "[IMG]",
            "href": urljoin(base, link.get("href"))
        })
    return links

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("\nUsage:\n\t{} <URL>\n".format(sys.argv[0]))
        sys.exit(1)
    for link in extract_links(sys.argv[-1]):
        print("[{}]({})".format(link["text"], link["href"]))

```

Modular !

Highly informative!

Usage hints!

```
$ docker build -t linkextractor:v2 .

$ docker run --rm linkextractor:v2 https://docker.com
[[IMG]](https://docker.com/dockercon)
[Watch live >](https://docker.com/dockercon/register-livestream)
[[IMG]](https://docker.com/)
[Why Docker?](https://docker.com/why-docker)
[What is a Container?](https://docker.com/resources/what-
container)
[Company](https://docker.com/company)
[Partners](https://docker.com/partners)
[Products](https://docker.com/products)
[Docker Enterprise](https://docker.com/products/docker-
enterprise)
[Docker Hub](https://docker.com/products/docker-hub)
```

```
$ docker image ls linkextractor
linkextractor                       v2
linkextractor                       v1.1
linkextractor                       v1
```

LINKEXTRACTOR ROADMAP

TURN IT INTO AN API SERVICE INSTEAD OF A ONE-OFF COMMAND

PROFIT!

1. ”MAGICALLY” CODE MAIN.PY

```
$ cp cheats/main.py .
```

linkextractor.py | Dockerfile ● | *main.py* ×

```
1   #!/usr/bin/env python
2
3   from flask import Flask
4   from flask import request
5   from flask import jsonify
6   from linkextractor import extract_links
7
8   app = Flask(__name__)
9
10  @app.route("/")
11  def index():
12      return "Usage: http://<hostname>[:<prt>]/api/<url>"
13
14  @app.route("/api/<path:url>")
15  def api(url):
16      qs = request.query_string.decode("utf-8")
17      if qs != "":
18          url += "?" + qs
19      links = extract_links(url)
20      return jsonify(links)
21
22  app.run(host="0.0.0.0")
23
```

1. ”MAGICALLY” CODE MAIN.PY

```
$ cp cheats/main.py .
```

linkextractor.py | Dockerfile ● | *main.py* ×

```python
#!/usr/bin/env python

from flask import Flask
from flask import request
from flask import jsonify
from linkextractor import extract_links

app = Flask(__name__)

@app.route("/")
def index():
    return "Usage: http://<hostname>[:<prt>]/api/<url>"

@app.route("/api/<path:url>")
def api(url):
    qs = request.query_string.decode("utf-8")
    if qs != "":
        url += "?" + qs
    links = extract_links(url)
    return jsonify(links)

app.run(host="0.0.0.0")

```

That’s our baby!

”MAGICALLY” CODE MAIN.PY

```
$ cp cheats/main.py .
```

“Go ahead caller, I’m listening.”

linkextractor.py | Dockerfile ● | *main.py* ×

```python
#!/usr/bin/env python

from flask import Flask
from flask import request
from flask import jsonify
from linkextractor import extract_links

app = Flask(__name__)

@app.route("/")
def index():
    return "Usage: http://<hostname>[:<prt>]/api/<url>"

@app.route("/api/<path:url>")
def api(url):
    qs = request.query_string.decode("utf-8")
    if qs != "":
        url += "?" + qs
    links = extract_links(url)
    return jsonify(links)

app.run(host="0.0.0.0")

```

That’s our baby!

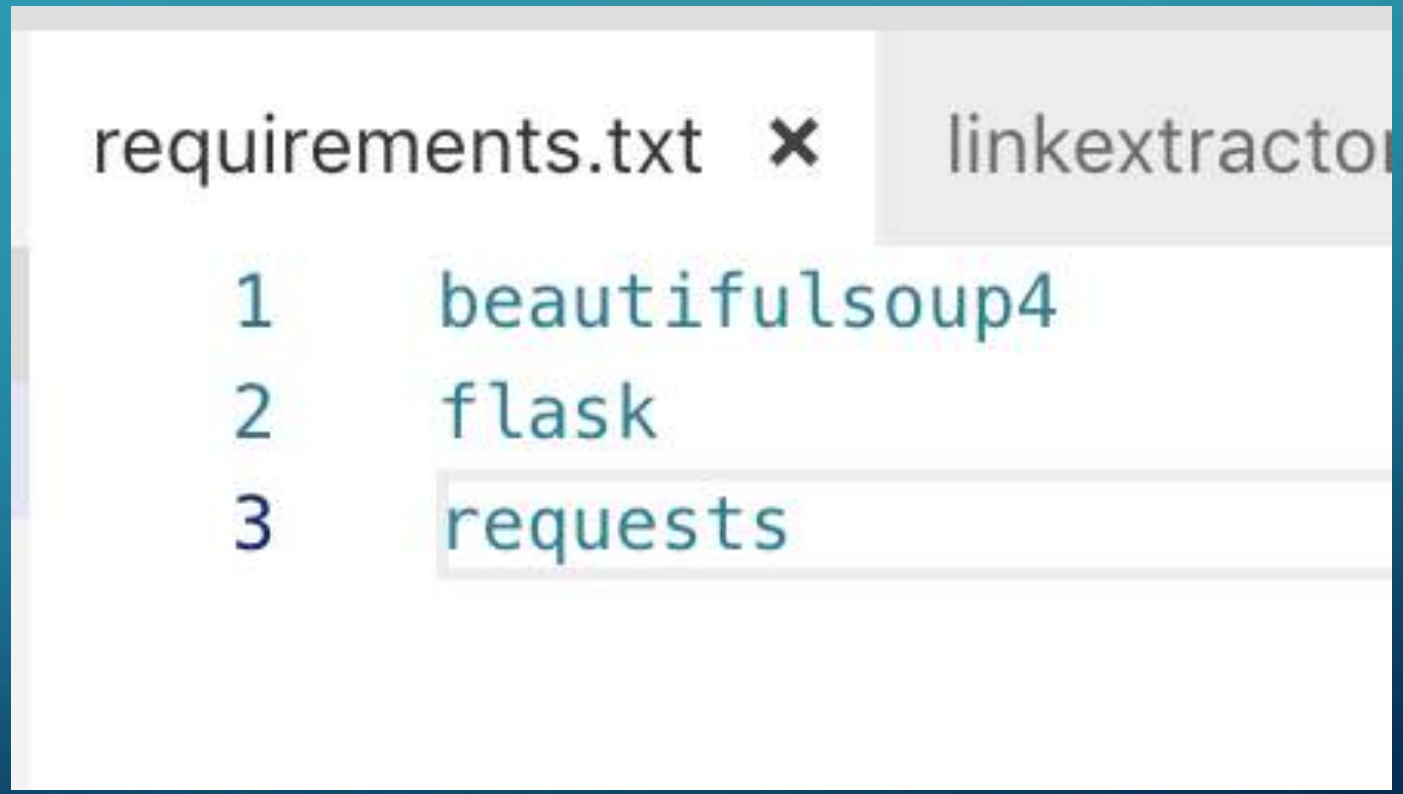
OPTIMIZE OUR DOCKERFILE

```
FROM python:3

LABEL maintainer="<your name>"

RUN pip install beautifulsoup4

RUN pip install requests

WORKDIR /app

COPY linkextractor.py /app/

RUN chmod a+x linkextractor.py

ENTRYPOINT ["./linkextractor.py"]
```

Eliminate extra manual install steps

- Reduce layers
- Maintain dependencies the way we normally do in our language of choice

2. PYTHON REQUIREMENTS FILE



OPTIMIZE OUR DOCKERFILE

```
FROM python:3
LABEL maintainer="<your name>"

RUN pip install beautifulsoup4
RUN pip install requests

WORKDIR /app
COPY linkextractor.py /app/
RUN chmod a+x linkextractor.py

ENTRYPOINT ["./linkextractor.py"]
```


Copy all our code in at once

OPTIMIZE OUR DOCKERFILE

```
FROM python:3
LABEL maintainer="<your name>"

RUN pip install beautifulsoup4
RUN pip install requests

WORKDIR /app
COPY linkextractor.py /app/
RUN chmod a+x linkextractor.py

ENTRYPOINT ["./linkextractor.py"]
```

We no longer want a single, one-off command…we want a service

OPTIMIZE OUR DOCKERFILE

```
FROM python:3
LABEL maintainer="<your name>"

WORKDIR /app
COPY requirements.txt /app/
RUN pip install -r requirements.txt

COPY *.py /app/
RUN chmod a+x *.py

CMD ["./main.py"]
```

Note the switch from ENTRYPOINT to CMD

- ENTRYPOINT - a command to run every time
  - good for executables
- CMD - a *default* command for our container to run but it can be substituted

LINKEXTRACTOR V3 - API-IFIED

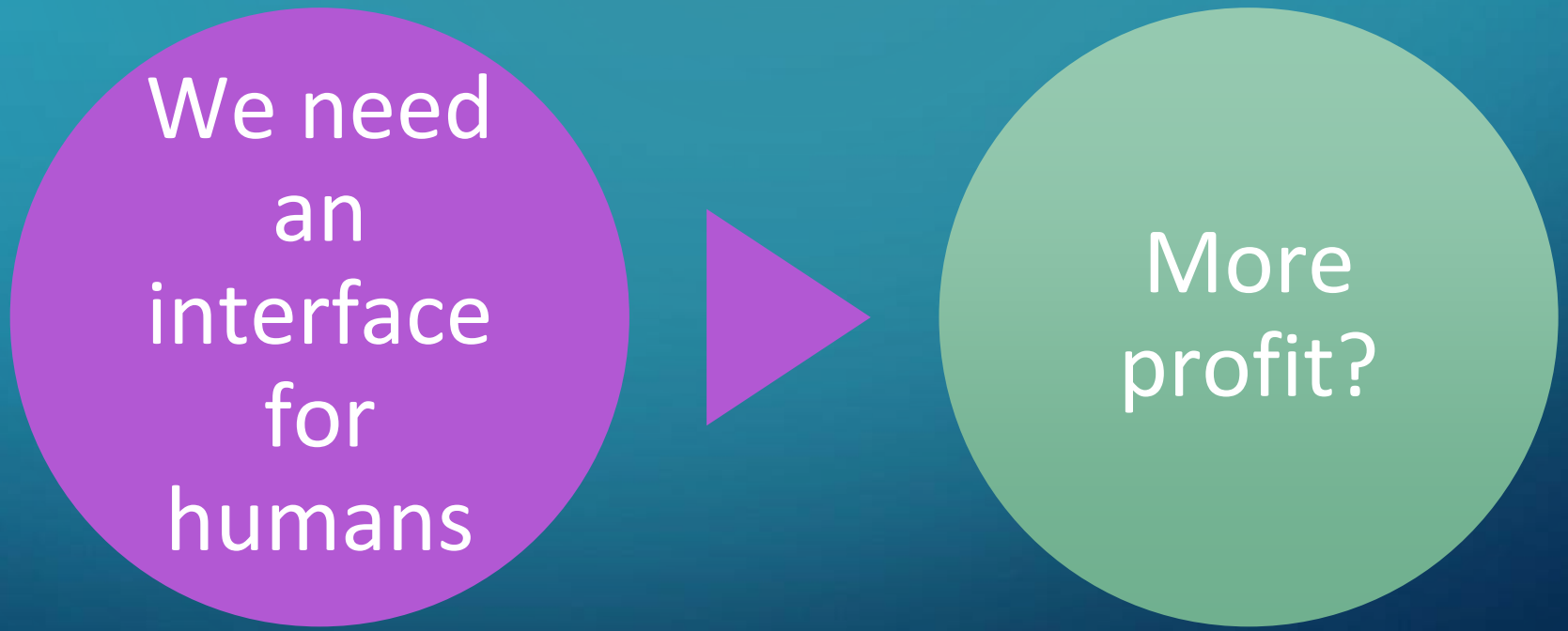
```
$ docker build -t linkextractor:v3 .

$ docker run -d -p 5000:5000 --name=linkextractor
  linkextractor:v3

$ curl localhost:5000
Usage: http://<hostname>[:<prt>]/api/<url>

$ curl localhost:5000/api/http://docker.com
[{"href":"http://docker.com/dockercon","text":"[IMG]"},{"href":"h
ttp://docker.com/dockercon/register-livestream","text":"Watch
live
>"},{"href":"http://docker.com/","text":"[IMG]"},{"href":"http://
docker.com/why-docker","text":"Why
Docker?"},{"href":"http://docker.com/resources/what-
container","text":"What is a
```

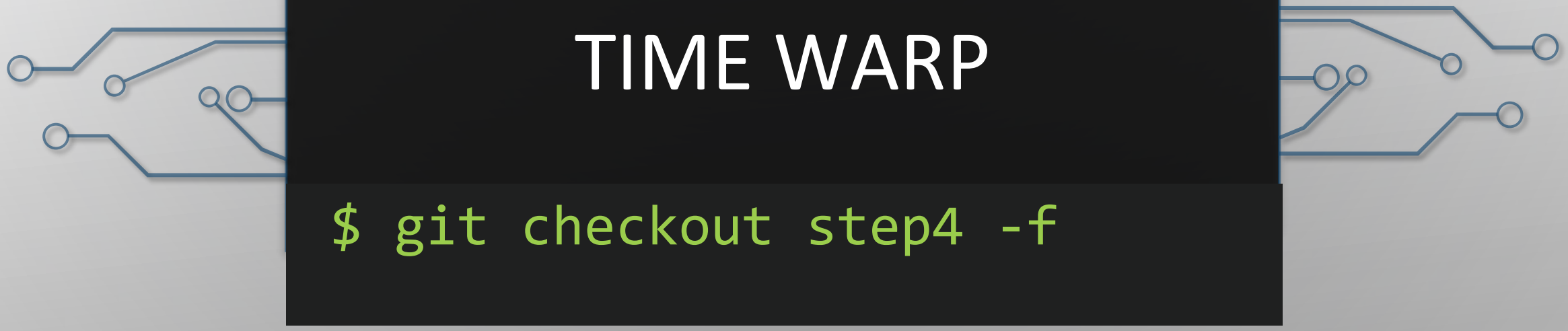
DUDE, WHERE ARE MY LOGS?

```
$ docker container logs linkextractor
 * Serving Flask app "main" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production
environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
172.17.0.1 - - [01/May/2019 09:14:23] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [01/May/2019 09:14:37] "GET
/api/http://example.com HTTP/1.1" 200 -
172.17.0.1 - - [01/May/2019 09:16:33] "GET /api/http://docker.com
HTTP/1.1" 200 -
```

```
$ docker container rm -f linkextractor
```

NEW ROADMAP!




TIME WARP
$ git checkout step4 -f

WHAT'S CHANGED?

```
.
├── README.md
├── api
│   ├── Dockerfile
│   ├── linkextractor.py
│   ├── main.py
│   └── requirements.txt
├── docker-compose.yml
└── www
    └── index.php
```

Here's our v3 stuff

WHAT'S CHANGED?

```
.
├── README.md
├── api
│   ├── Dockerfile
│   ├── linkextractor.py
│   ├── main.py
│   └── requirements.txt
├── docker-compose.yml
└── www
    └── index.php
```

We created a web front-end

WHAT'S CHANGED?

```
.
├── README.md
├── api
│   ├── Dockerfile
│   ├── linkextractor.py
│   ├── main.py
│   └── requirements.txt
├── docker-compose.yml
└── www
    └── index.php
```

```yaml
version: '3'

services:
  api:
    image: linkextractor-api:step4-python
    build: ./api
    ports:
      - "5000:5000"
  web:
    image: php:7-apache
    ports:
      - "80:80"
    environment:
      - API_ENDPOINT=http://api:5000/api/
    volumes:
      - ./www:/var/www/html
```

2 containers / services instead of just one

```yaml
version: '3'

services:
  api:
    image: linkextractor-api:step4-python
    build: ./api
    ports:
      - "5000:5000"
  web:
    image: php:7-apache
    ports:
      - "80:80"
    environment:
      - API_ENDPOINT=http://api:5000/api/
    volumes:
      - ./www:/var/www/html
```

The API we worked do diligently to create

```yaml
version: '3'

services:
  api:
    image: linkextractor-api:step4-python
    build: ./api
    ports:
      - "5000:5000"
  web:
    image: php:7-apache
    ports:
      - "80:80"
    environment:
      - API_ENDPOINT=http://api:5000/api/
    volumes:
      - ./www:/var/www/html
```

The new web front-end

```yaml
version: '3'

services:
  api:
    image: linkextractor-api:step4-python
    build: ./api
    ports:
      - "5000:5000"
  web:
    image: php:7-apache
    ports:
      - "80:80"
    environment:
      - API_ENDPOINT=http://api:5000/api/
    volumes:
      - ./www:/var/www/html
```

New image…Official, unmodified so no Dockerfile required

```yaml
version: '3'

services:
  api:
    image: linkextractor-api:step4-python
    build: ./api
    ports:
      - "5000:5000"
  web:
    image: php:7-apache
    ports:
      - "80:80"
    environment:
      - API_ENDPOINT=http://api:5000/api/
    volumes:
      - ./www:/var/www/html
```

We pass in an environment variable, which our front-end expects

```yaml
version: '3'

services:
  api:
    image: linkextractor-api:step4-python
    build: ./api
    ports:
      - "5000:5000"
  web:
    image: php:7-apache
    ports:
      - "80:80"
    environment:
      - API_ENDPOINT=http://api:5000/api/
    volumes:
      - ./www:/var/www/html
```

Mount our local www directory in the container…LIVE CODE CHANGES!

Our environment variable gets consumed

index.php

```php
<!DOCTYPE html>

<?php
  $api_endpoint = $_ENV["API_ENDPOINT"] ?: "http://localhost:5000/api/";
  $url = "";
  if(isset($_GET["url"]) && $_GET["url"] != "") {
    $url = $_GET["url"];
    $json = @file_get_contents($api_endpoint . $url);
    if($json == false) {
      $err = "Something is wrong with the URL: " . $url;
    } else {
      $links = json_decode($json, true);
      $domains = [];
      foreach($links as $link) {
        array_push($domains, parse_url($link["href"], PHP_URL_HOST));
      }
      $domainct = @array_count_values($domains);
      arsort($domainct);
    }
  }
?>

<html>
  <head>
    <meta charset="utf-8">
    <title>Link Extractor</title>
    <style media="screen">
      html {
        background: #FAF7D6;
```

RUNNING A MULTI-SERVICE APP WITH COMPOSE

```
$ docker-compose up -d --build
Creating network "linkextractor_default" with the default driver
Building api
Step 1/8 : FROM       python:3
 ---> 954987809e63
.
.
.
Creating linkextractor_api_1 ... done
Creating linkextractor_web_1 ... done
```

RUNNING A MULTI-SERVICE APP WITH COMPOSE

```
$ docker-compose up -d --build
Creating network "linkextractor_default" with the default driver
Building api
Step 1/8 : FROM       python:3
 ---> 954987809e63
.
.
.
Creating linkextractor_api_1 ... done
Creating linkextractor_web_1 ... done
```

We didn't specify network settings so we get a default, internal only network, on which our services can talk

RUNNING A MULTI-SERVICE APP WITH COMPOSE

```
$ docker-compose up -d --build
Creating network "linkextractor_default" with the default driver
Building api
Step 1/8 : FROM       python:3
 ---> 954987809e63
.
.
.
Creating linkextractor_api_1 ... done
Creating linkextractor_web_1 ... done
```

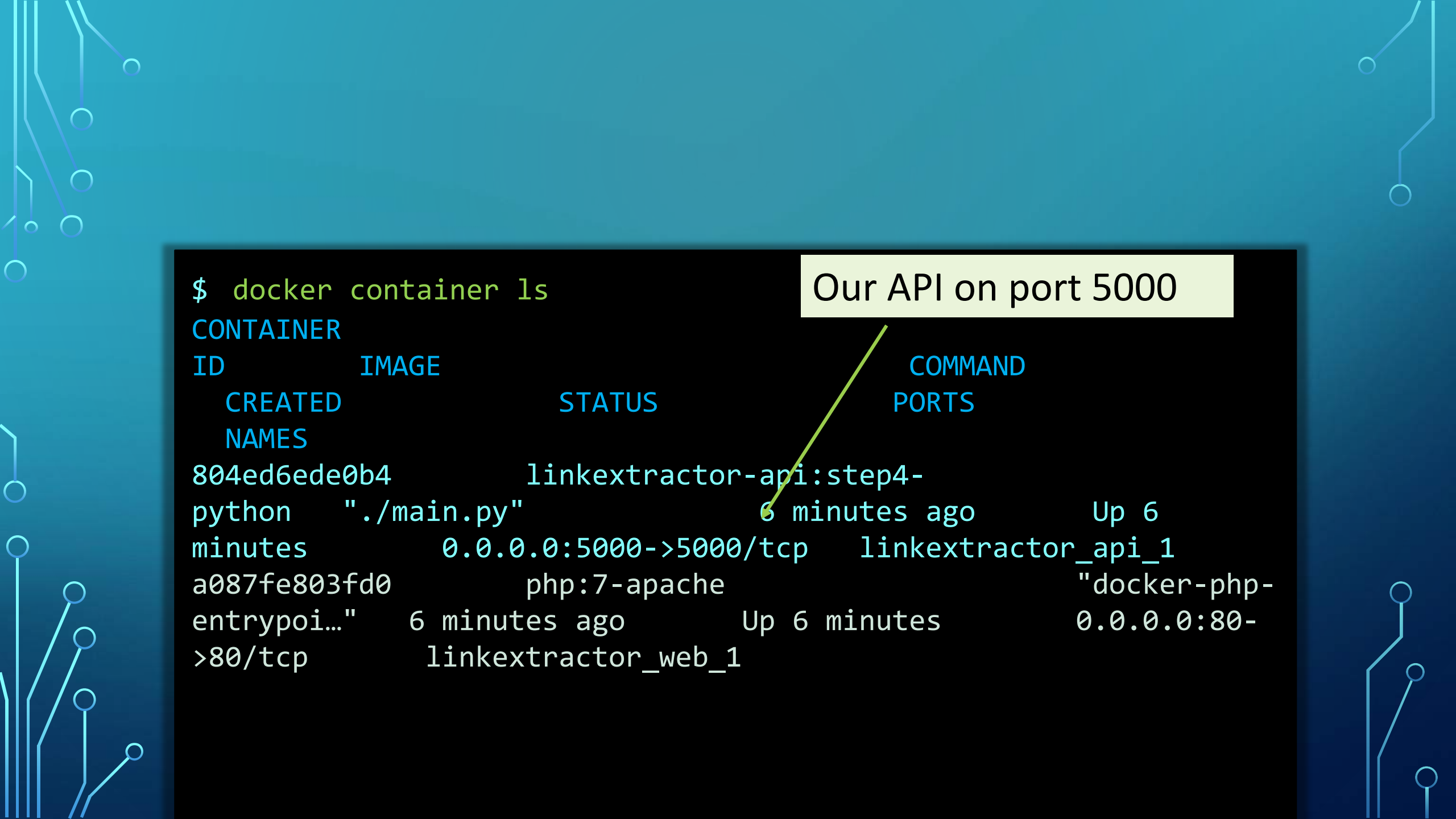
Stuff gets built (*--build* forces a rebuild)…you might see the php image get pulled

RUNNING A MULTI-SERVICE APP WITH COMPOSE

```
$ docker-compose up -d --build
Creating network "linkextractor_default" with the default driver
Building api
Step 1/8 : FROM       python:3
 ---> 954987809e63
.
.
.
Creating linkextractor_api_1 ... done
Creating linkextractor_web_1 ... done
```

And finally our two services are up and running

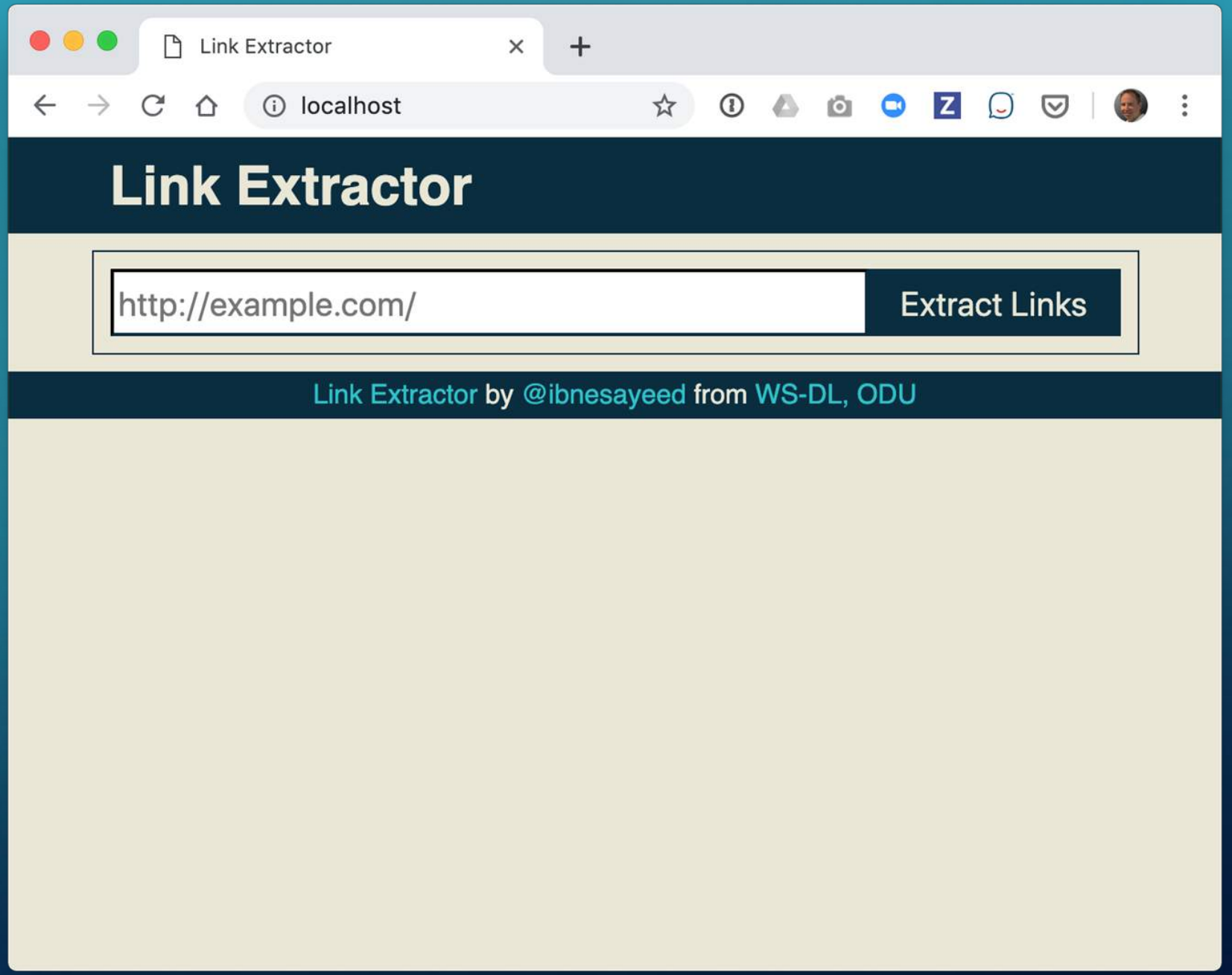
```
$ docker container ls
CONTAINER
ID        IMAGE                              COMMAND
  CREATED              STATUS              PORTS
  NAMES
804ed6ede0b4        linkextractor-api:step4-
python    "./main.py"              6 minutes ago       Up 6
minutes         0.0.0.0:5000->5000/tcp   linkextractor_api_1
a087fe803fd0        php:7-apache                       "docker-php-
entrypoi…"   6 minutes ago       Up 6 minutes        0.0.0.0:80-
>80/tcp       linkextractor_web_1
```


$ docker container ls
CONTAINER
ID IMAGE COMMAND
CREATED STATUS PORTS
NAMES
804ed6ede0b4 linkextractor-api:step4-
python "./main.py" 6 minutes ago Up 6
minutes 0.0.0.0:5000->5000/tcp linkextractor_api_1
a087fe803fd0 php:7-apache "docker-php-
entrypoi…" 6 minutes ago Up 6 minutes 0.0.0.0:80-
>80/tcp linkextractor_web_1
Our API on port 5000

```
$ docker container ls
CONTAINER
ID        IMAGE                                COMMAND
  CREATED              STATUS                 PORTS
  NAMES
804ed6ede0b4        linkextractor-api:step4-
python    "./main.py"                6 minutes ago       Up 6
minutes         0.0.0.0:5000->5000/tcp   linkextractor_api_1
a087fe803fd0        php:7-apache                         "docker-php-
entrypoi…"   6 minutes ago       Up 6 minutes        0.0.0.0:80-
>80/tcp       linkextractor_web_1
```

Web front on port 80

```
$ curl localhost:5000/api/http://docker.com
$ curl localhost
```


Link Extractor
localhost
Link Extractor
http://example.com/
Extract Links
Link Extractor by @ibnesayeed from WS-DL, ODU

LATE-BREAKING REQUIREMENTS CHANGE!

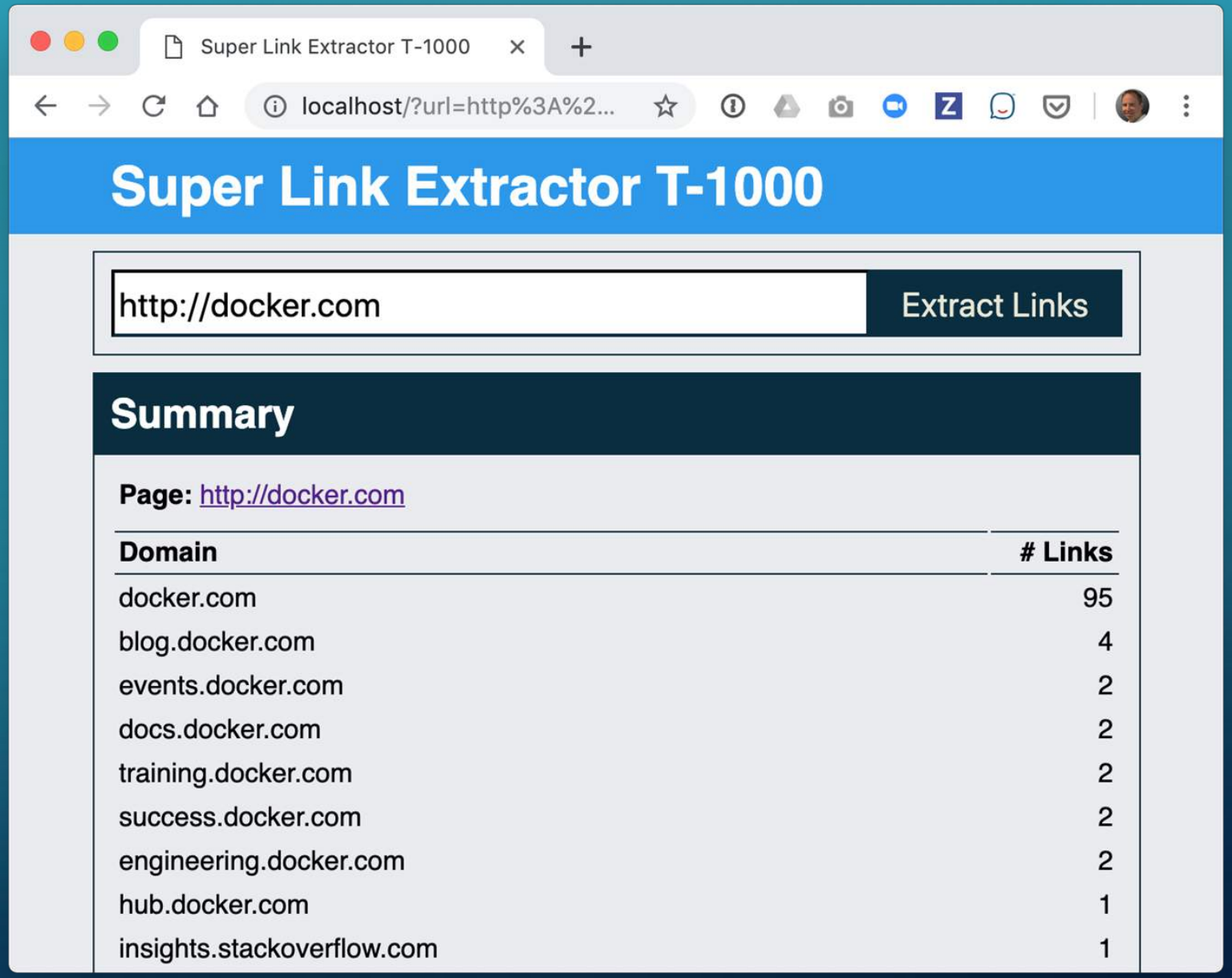
Marketing rebranding

index.php

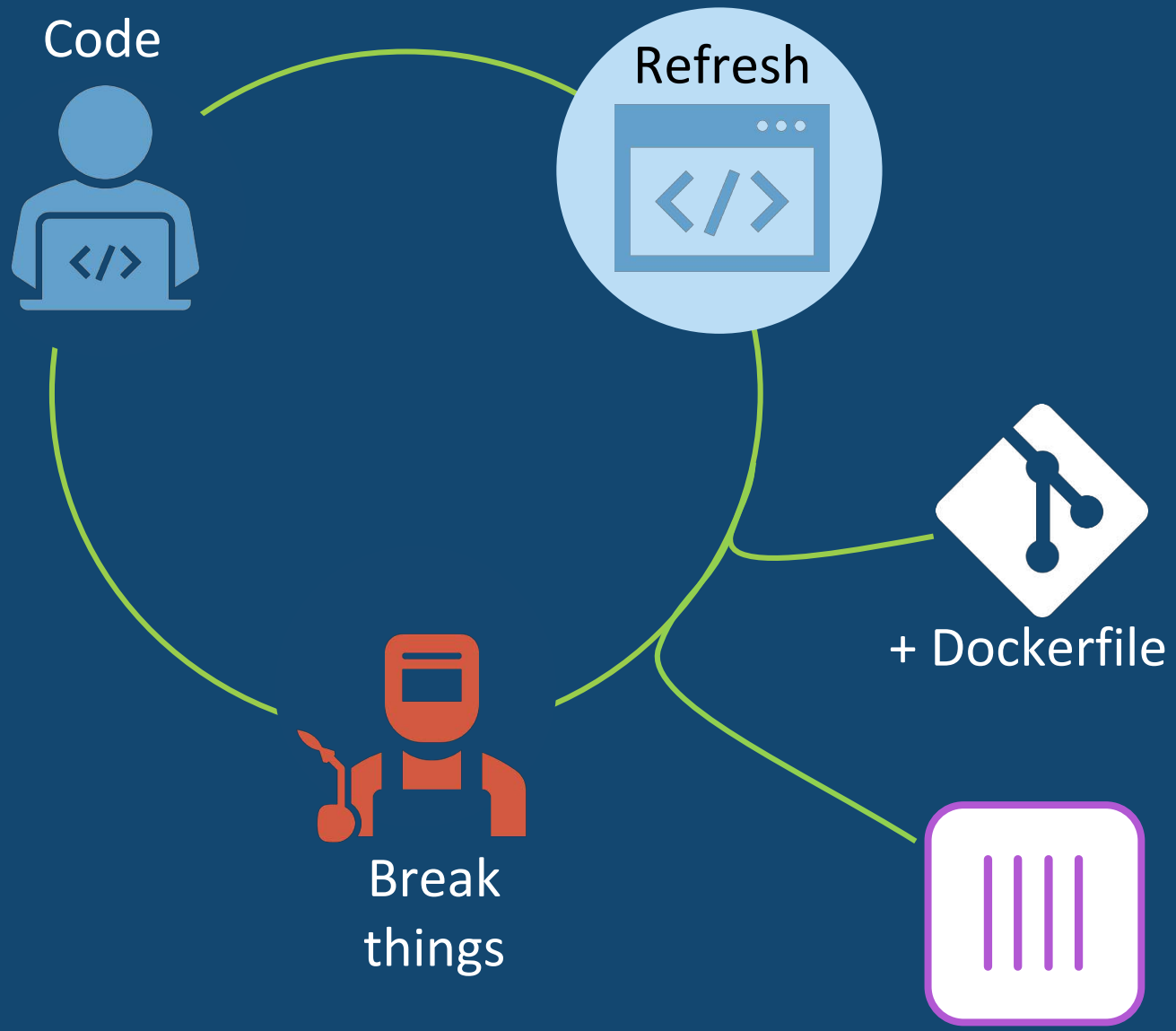
```php
<!DOCTYPE html>

<?php
  $api_endpoint = $_ENV["API_ENDPOINT"] ?: "http://localhost:5000/api/";
  $url = "";
  if(isset($_GET["url"]) && $_GET["url"] != "") {
    $url = $_GET["url"];
    $json = @file_get_contents($api_endpoint . $url);
    if($json == false) {
      $err = "Something is wrong with the URL: " . $url;
    } else {
      $links = json_decode($json, true);
      $domains = [];
      foreach($links as $link) {
        array_push($domains, parse_url($link["href"], PHP_URL_HOST));
      }
      $domainct = @array_count_values($domains);
      arsort($domainct);
    }
  }
?>

<html>
  <head>
    <meta charset="utf-8">
    <title>Super Link Extractor T-1000</title>
    <style media="screen">
      html {
        background: #E9EBEE;
```

- Update the name to “Super Link Extractor T-1000”
  Find & replace “Link Extractor”
- Change the background color to #E9EBEE
  html {
  background:

Save index.php & refresh the web page


Super Link Extractor T-1000
localhost/?url=http%3A%2...
Super Link Extractor T-1000
http://docker.com
Extract Links
Summary
Page: http://docker.com
Domain # Links
docker.com 95
blog.docker.com 4
events.docker.com 2
docs.docker.com 2
training.docker.com 2
success.docker.com 2
engineering.docker.com 2
hub.docker.com 1
insights.stackoverflow.com 1

OUR NEW DEV-TEST LOOP



BEST PRACTICES

- Image tags

```
docker build -t linkextractor:v3 .
```

Note

- Simple semantic version OK for 1 person
- If you have a team you'll inevitably have issues
- And "v3" doesn't really tell you much about the code revision you were on (hard to inspect later)

ONE SUGGESTION FOR BETTER TAGGING & SHARING

```
$ git commit -am "marketing changes"
[step4 a771a28] marketing updates
 1 file changed, 3 insertions(+), 3 deletions(-)

$ git log -1 --pretty=%h
a771a28
$ cd api
$ docker build -t linkextractor_api:$(git log -1 --format=%h) .
$ docker image ls linkextractor
```

OTHER IDEAS

“MAKE” A BUILD AND PUSH

```
NAME   := acmecorp/linkextractor_api
TAG    := $$(git log -1 --pretty=%!H(MISSING))
IMG    := ${NAME}:${TAG}
LATEST := ${NAME}:latest

build:
  @docker build -t ${IMG} .
  @docker tag ${IMG} ${LATEST}

push:
  @docker push ${NAME}

login:
  @docker log -u ${DOCKER_USER} -p ${DOCKER_PASS}
```

https://container-solutions.com/tagging-docker-images-the-right-way/

DOCKERFILE “LABEL”

In your “Dockerfile”:

```
ARG GIT_COMMIT=unspecified
LABEL git_commit=$GIT_COMMIT
```

Your build command:

```
docker build -t flask-local-build --build-arg
GIT_COMMIT=$(git log -1 --format=%h) .
```

https://blog.scottlowe.org/2017/11/08/how-tag-docker-images-git-commit-information/

CLEAN-UP

```
$ docker container logs linkextractor_api_1

$ docker-compose down

$ docker image rm <imagename>:<tag>  # cleans up some space
```

EXTRA CREDIT

- Python, HTML, Ruby and other non-compiled code types will neatly run and let you mount the source code in to a container

- What about compiled stuff like Java?

HTTPS://GITHUB.COM/DOCKERSAMPLES

- Several freely available example apps from other workshops
- Browse to atsea-sample-shop-app
  - Java Spring + React with db, nginx, secrets and more
  - Look at /app/Dockerfile

```
FROM node:latest AS storefront
WORKDIR /usr/src/atsea/app/react-app
COPY react-app .
RUN npm install
RUN npm run build

FROM maven:latest AS appserver
WORKDIR /usr/src/atsea
COPY pom.xml .
RUN mvn -B -f pom.xml -s /usr/share/maven/ref/settings-docker.xml dependency:resolve
COPY . .
RUN mvn -B -s /usr/share/maven/ref/settings-docker.xml package -DskipTests

FROM java:8-jdk-alpine
RUN adduser -Dh /home/gordon gordon
WORKDIR /static
COPY --from=storefront /usr/src/atsea/app/react-app/build/ .
WORKDIR /app
COPY --from=appserver /usr/src/atsea/target/AtSea-0.0.1-SNAPSHOT.jar .
ENTRYPOINT ["java", "-jar", "/app/AtSea-0.0.1-SNAPSHOT.jar"]
CMD ["--spring.profiles.active=postgres"]
```

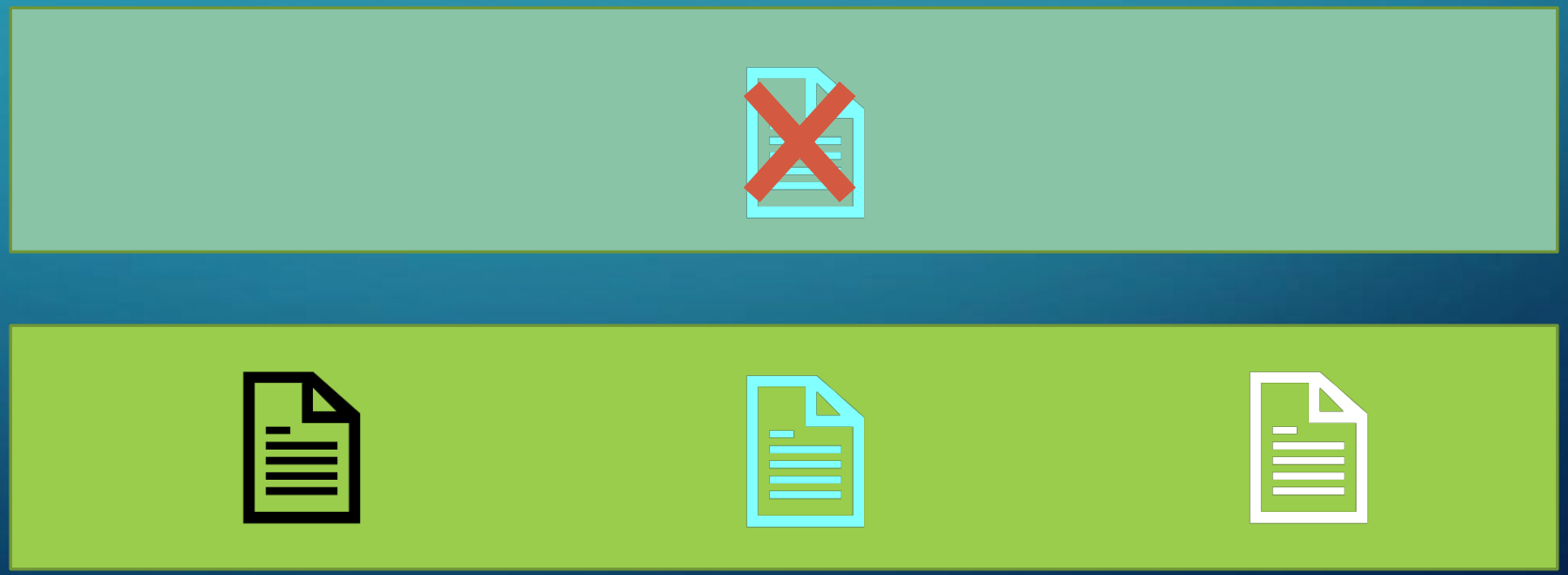
Multi-stage builds

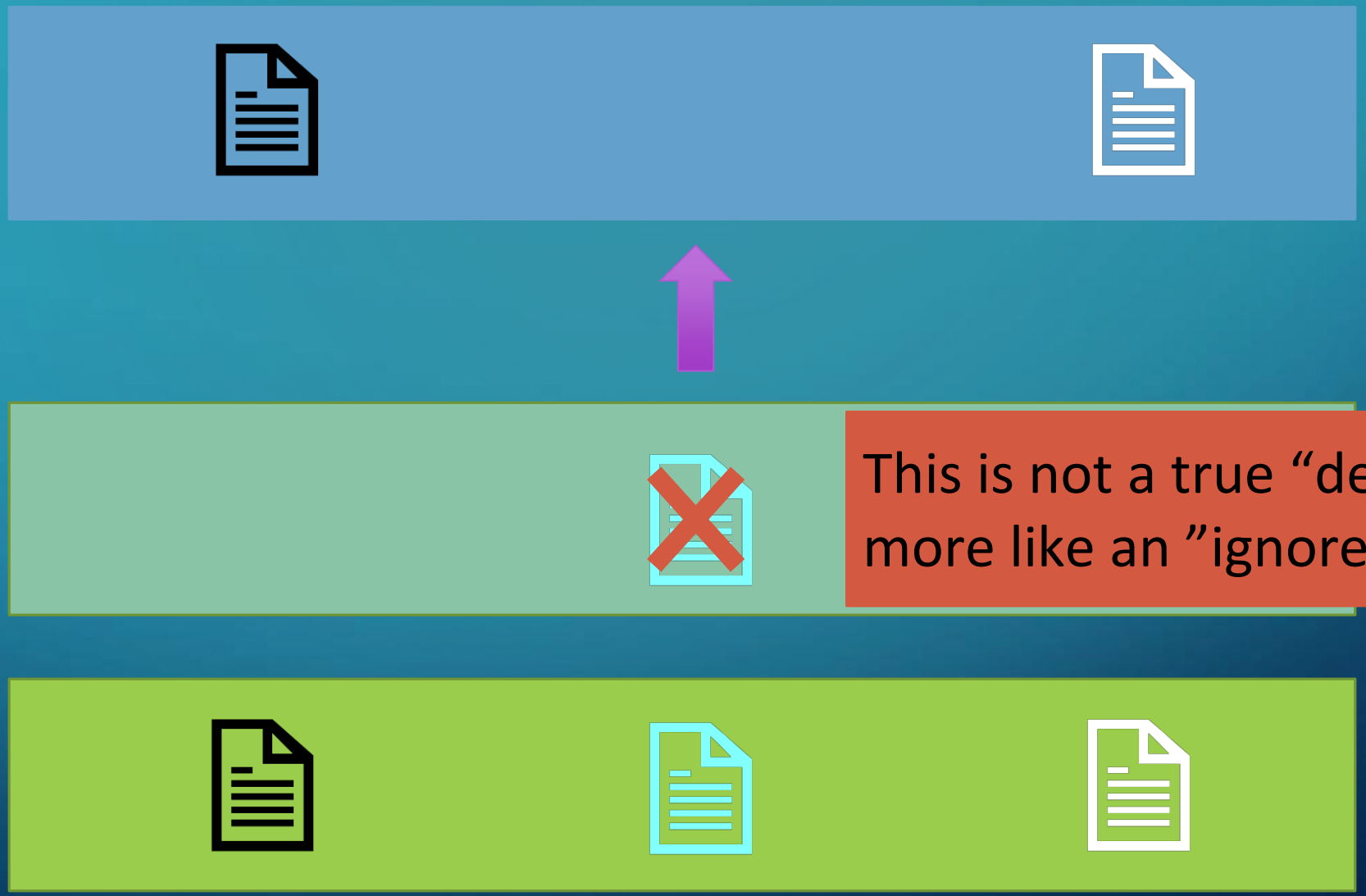
- Keep build artifacts out of final image
- Reduce final image size

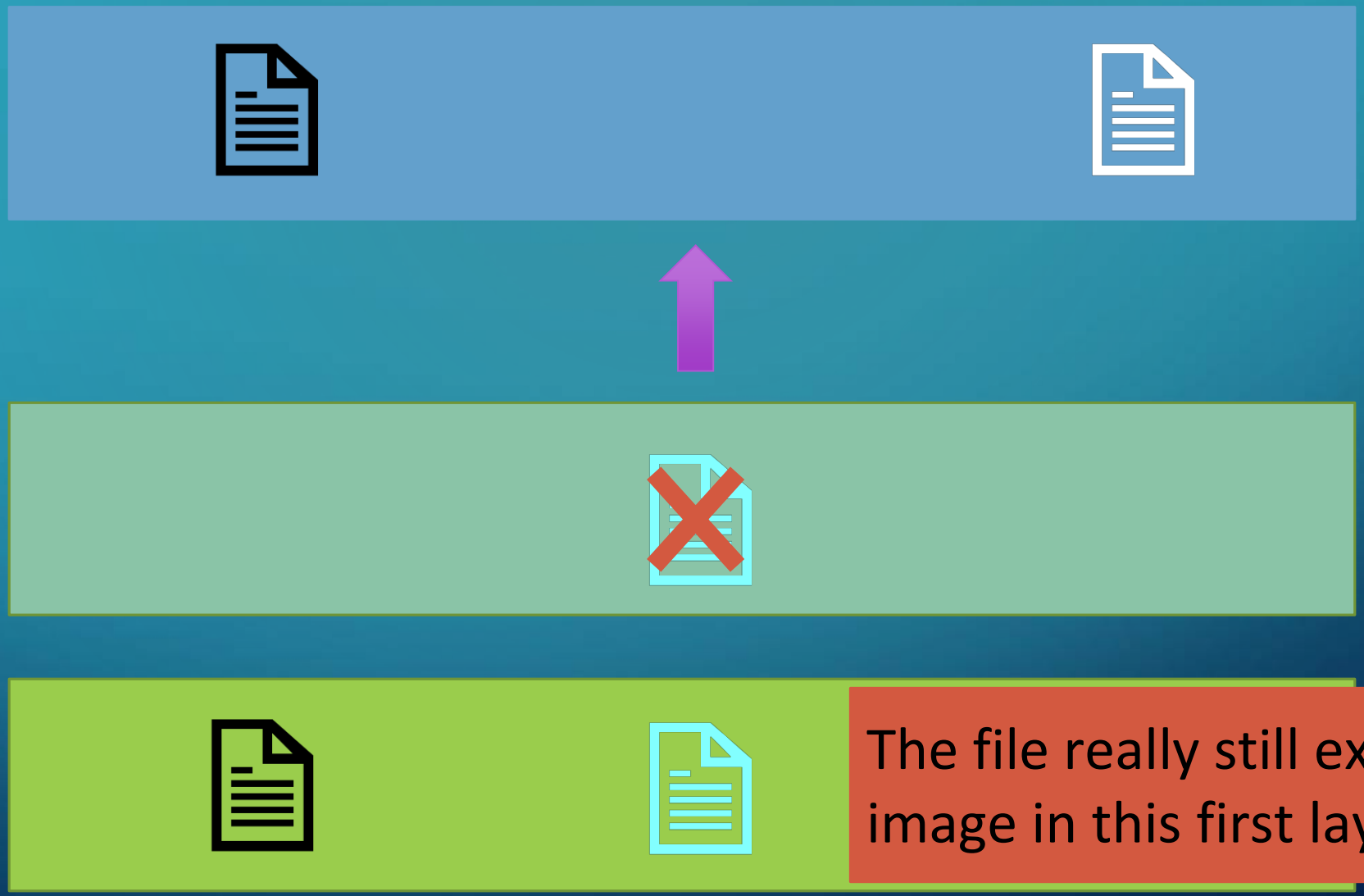
Note

- Deleting files via Dockerfile commands will NOT actually remove files from the image






This is not a true “delete”…it’s more like an ”ignore”


The file really still exists in the image in this first layer.

```
FROM node:latest AS storefront
WORKDIR /usr/src/atsea/app/react-app
COPY react-app .
RUN npm install
RUN npm run build

FROM maven:latest AS appserver
WORKDIR /usr/src/atsea
COPY pom.xml .
RUN mvn -B -f pom.xml -s /usr/share/maven/ref/settings-docker.xml dependency:resolve
COPY . .
RUN mvn -B -s /usr/share/maven/ref/settings-docker.xml package -DskipTests

FROM java:8-jdk-alpine
RUN adduser -Dh /home/gordon gordon
WORKDIR /static
COPY --from=storefront /usr/src/atsea/app/react-app/build/ .
WORKDIR /app
COPY --from=appserver /usr/src/atsea/target/AtSea-0.0.1-SNAPSHOT.jar .
ENTRYPOINT ["java", "-jar", "/app/AtSea-0.0.1-SNAPSHOT.jar"]
CMD ["--spring.profiles.active=postgres"]
```

Multi-stage builds

- The .jar file is built in the *appserver* stage but we don't need all the artifacts that come with a build - we just need the .jar.
- In the final stage the .jar file is COPY'ed in to the final image, resulting in a smaller image.

MORE THINGS TO TRY ON YOUR OWN

- Build Super Link Extractor T-1000 for 
  - https://engineering.docker.com/2019/04/multi-arch-images/
- Or Windows ("Switch to Windows containers…" on Windows OS only)
  - NOTE: There is a Windows Python image, but not PHP (front-end) so you're on your own to figure out the web front-end
- http://training.play-with-docker.com/
- **SECRETS!!!** Don't store sensitive info in your images!
- Networks in Compose: create private segments for internal app comms
- Deploy to Kube - it's built-in to Docker Desktop!
  - Make your life easier by using Compose on Kubernetes
- `docker push` your images to Hub or any other registry and run them somewhere else (`docker push --help`)