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
```

What about this?

```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
```

$\left.\vphantom{\begin{array}{c}a\\a\\a\\a\\a\\a\\a\\a\end{array}}\right\}$ The new web front-end

```
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

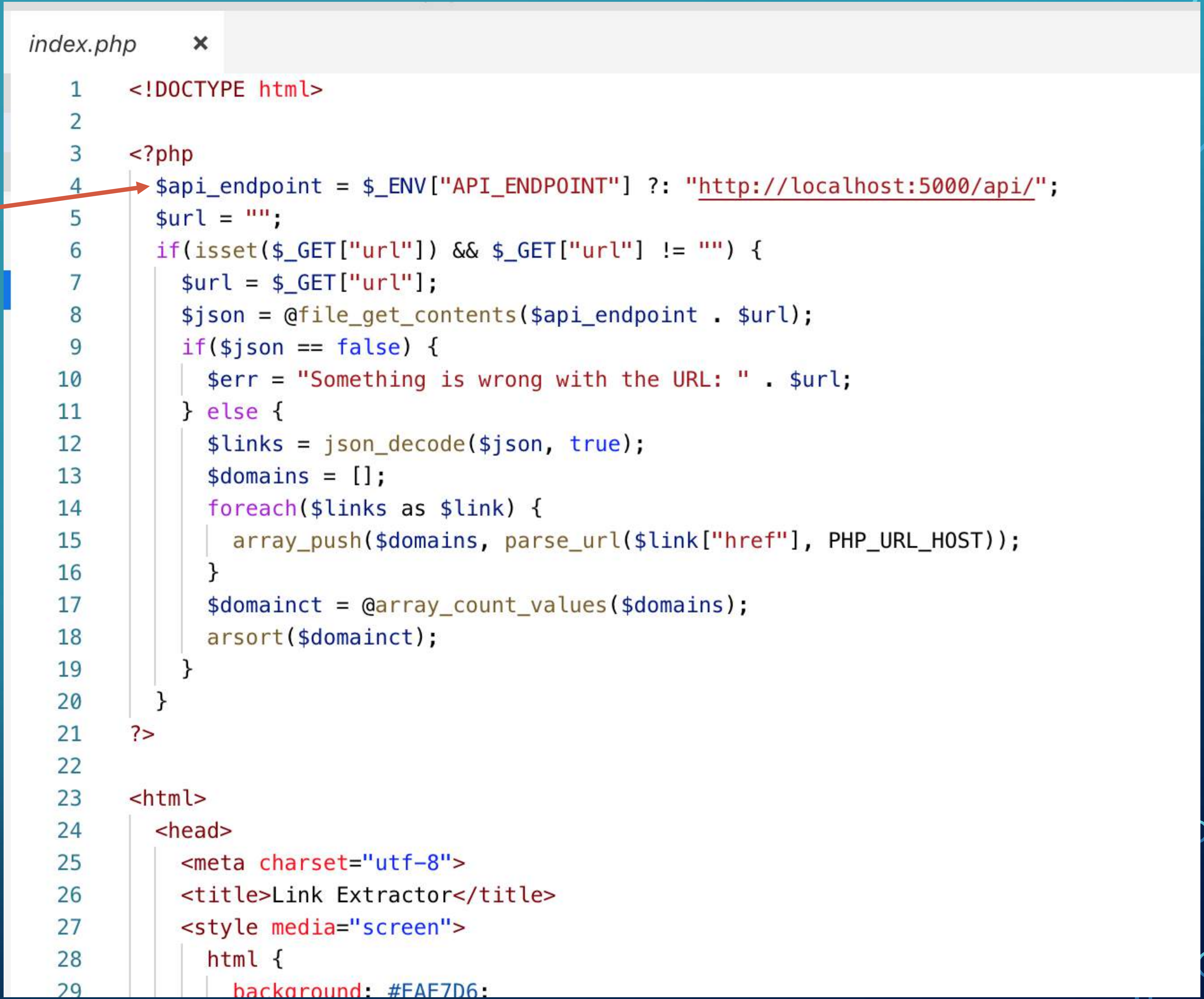
```
version: '3'

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

```
<!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
```

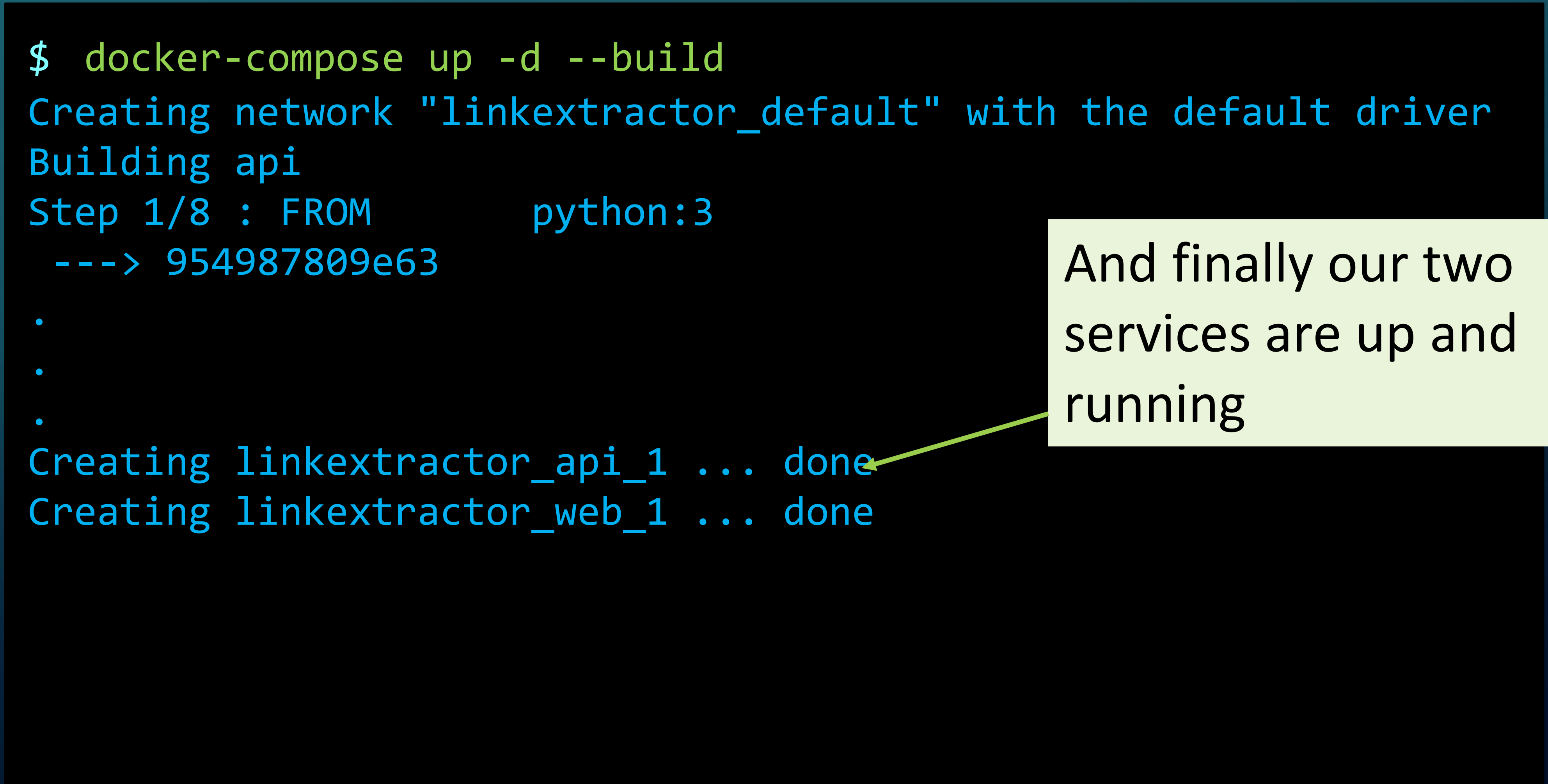
We didn’t specify network settings so we get a default, internal only network, on which our services can talk

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



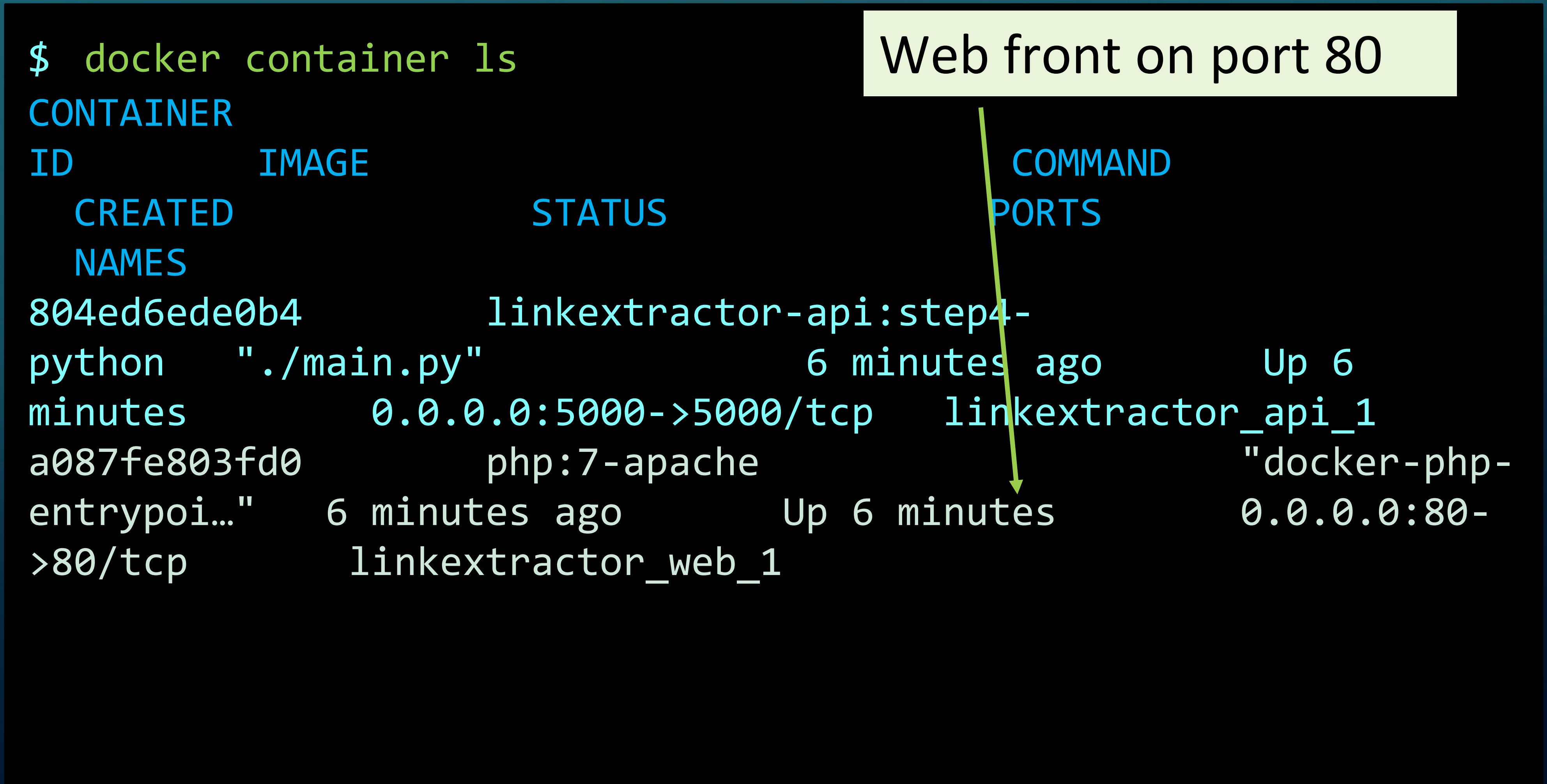
```
$  docker-compose up -d --build
Creating network "linkextractor_default" with the default driver
Building api
Step 1/8 : FROM       python:3
 ---> 954987809e63
.
.
.
Creating linkextractor_api_1 ... done
Creating linkextractor_web_1 ... done
```

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
```

Our API on port 5000

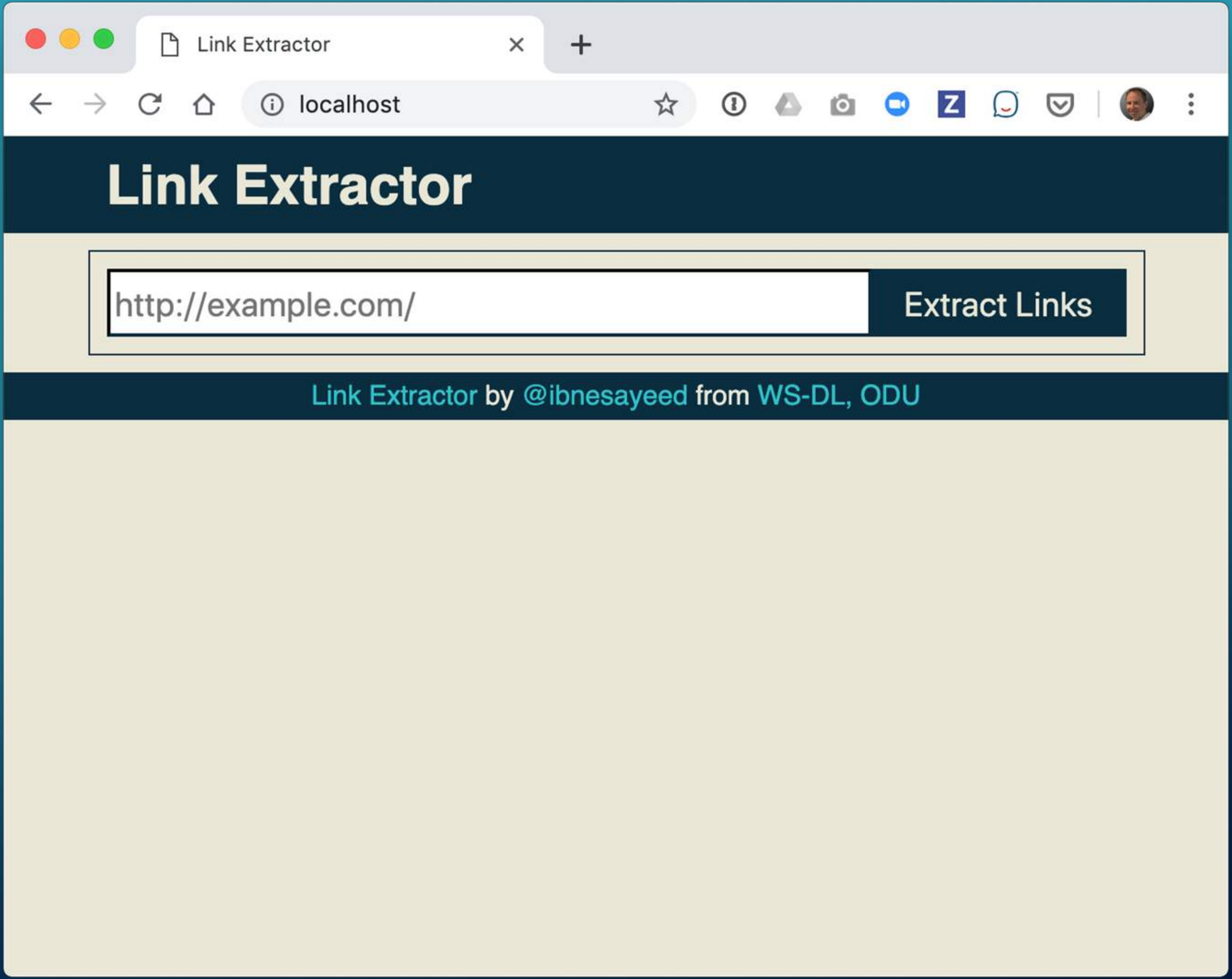
```
$  docker container ls
CONTAINER
ID          IMAGE                                  COMMAND
  CREATED             STATUS                     PORTS
  NAMES
804ed6ede0b4        linkextractor-api:step4-
python   "./main.py"              6 minutes ago       Up 6
minutes         0.0.0.0:5000->5000/tcp   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

```
$  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
```

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

$ curl localhost
```


Link Extractor

localhost

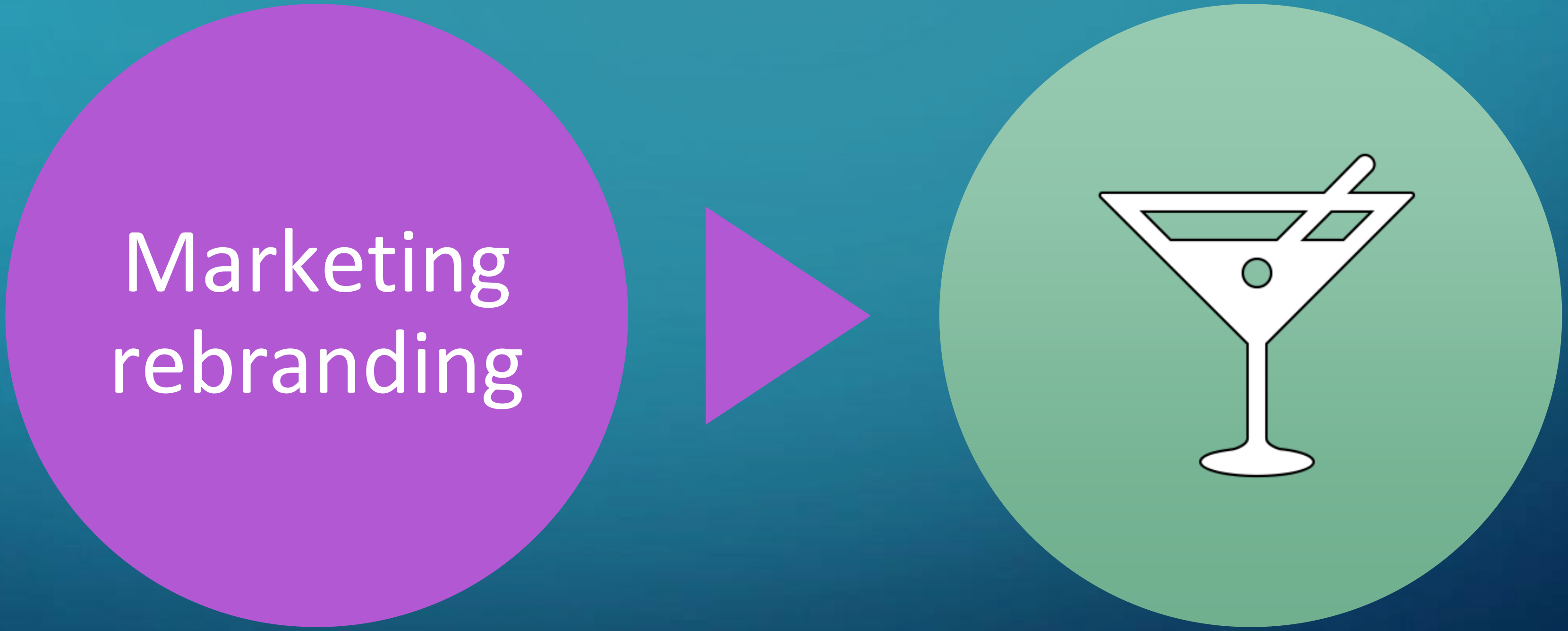
Link Extractor

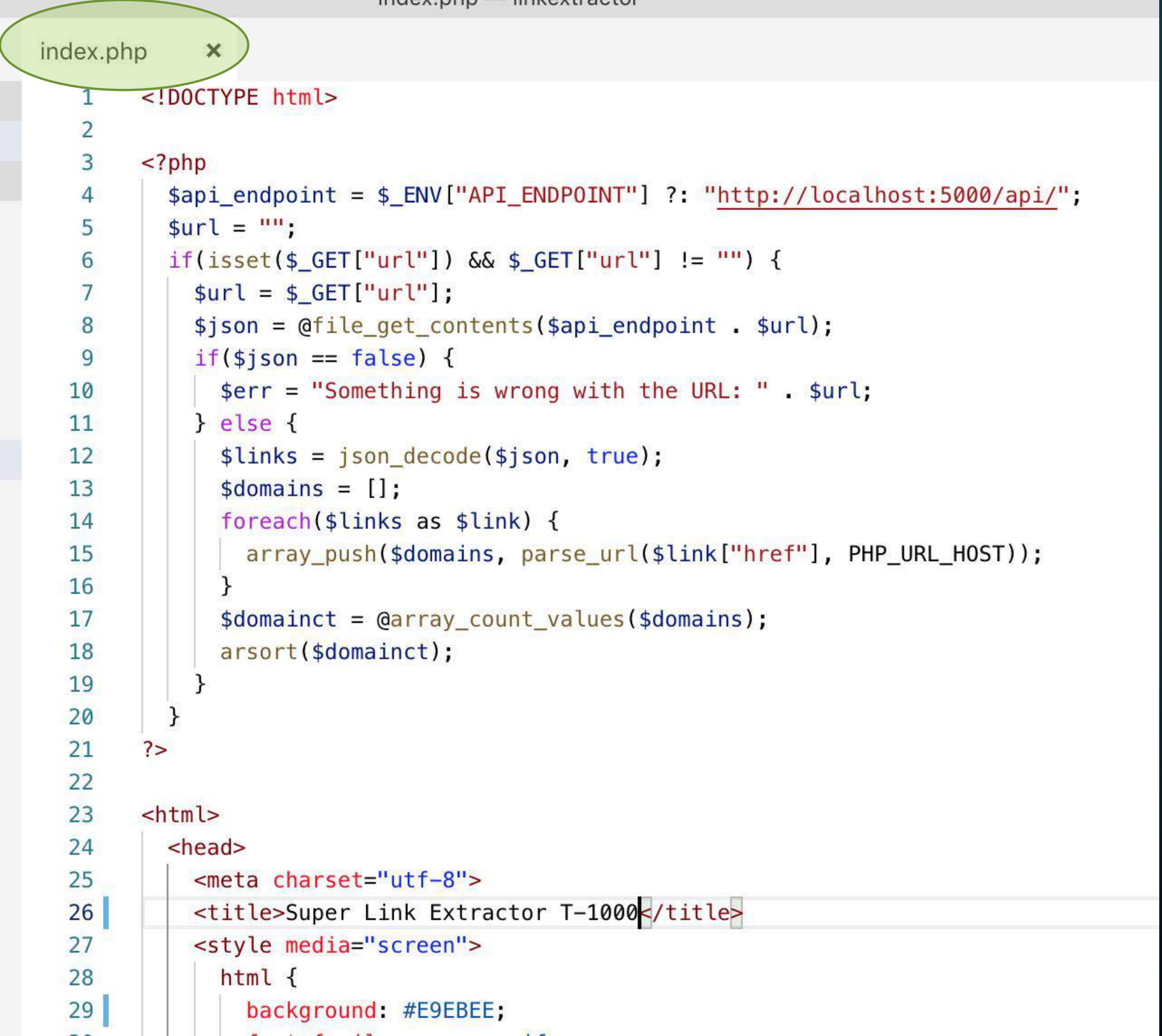
http://example.com/

Extract Links

Link Extractor by @ibnesayeed from WS-DL, ODU

LATE-BREAKING REQUIREMENTS CHANGE!

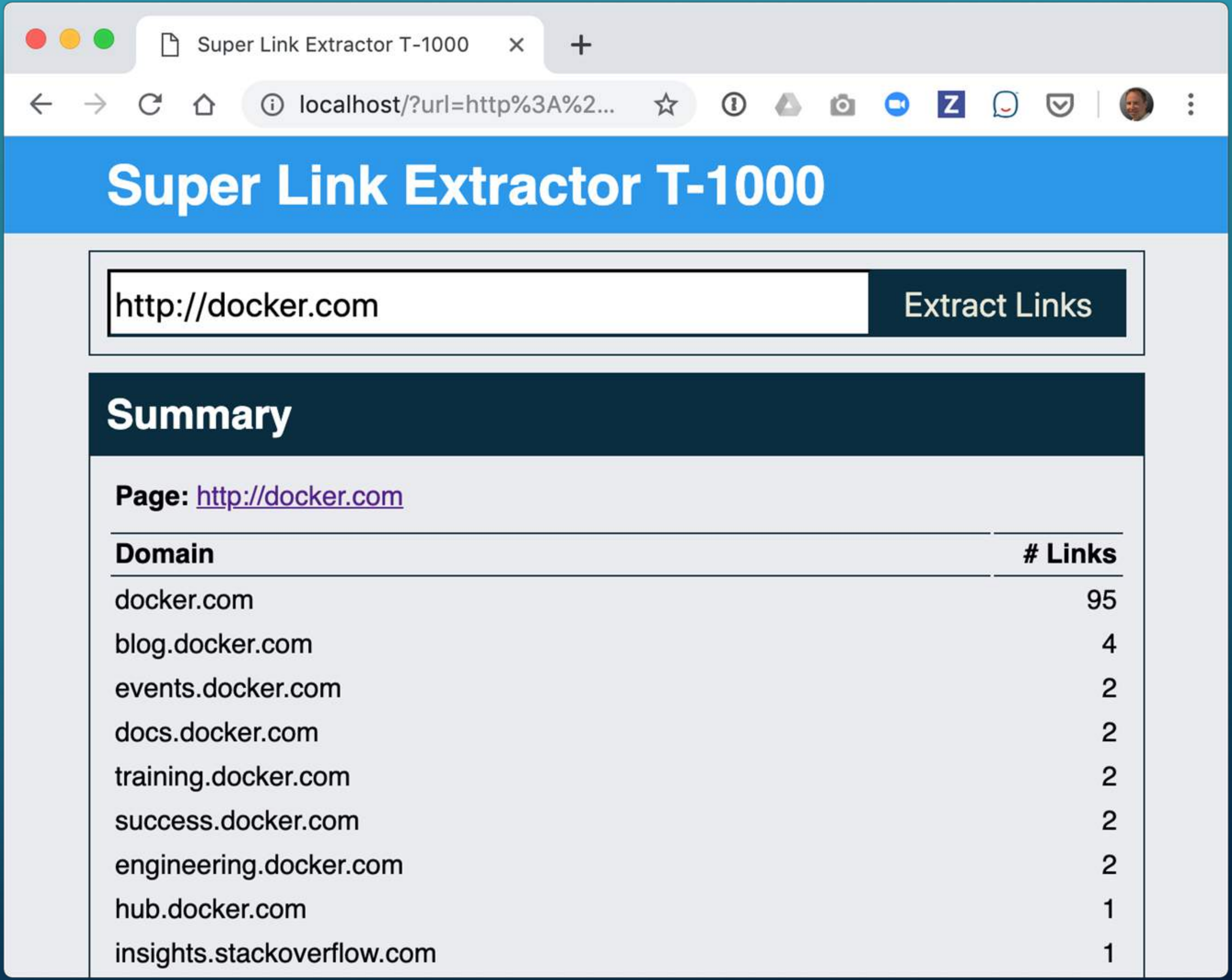




```
<!DOCTYPE html>

<?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

http://docker.com

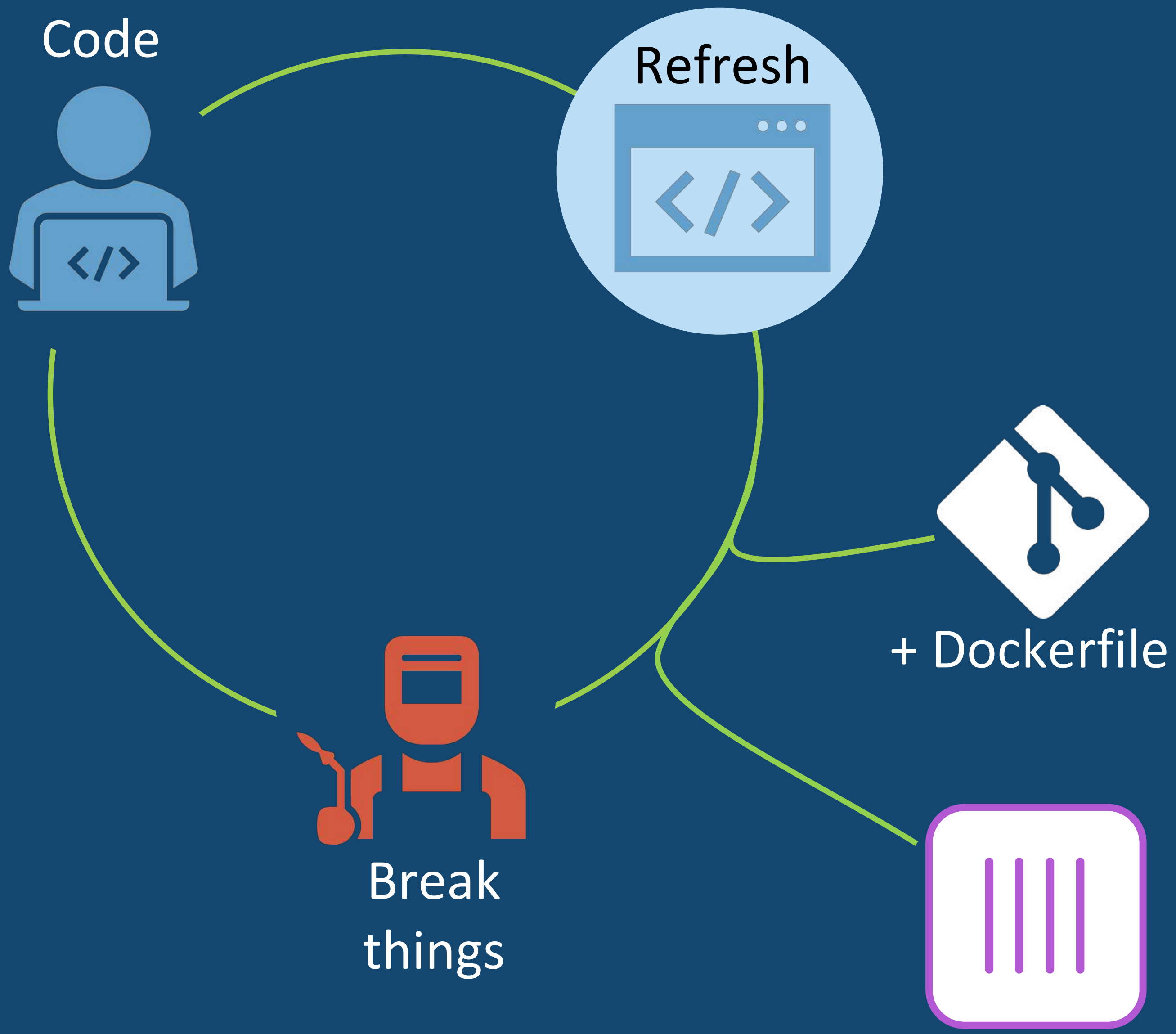
Extract Links

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

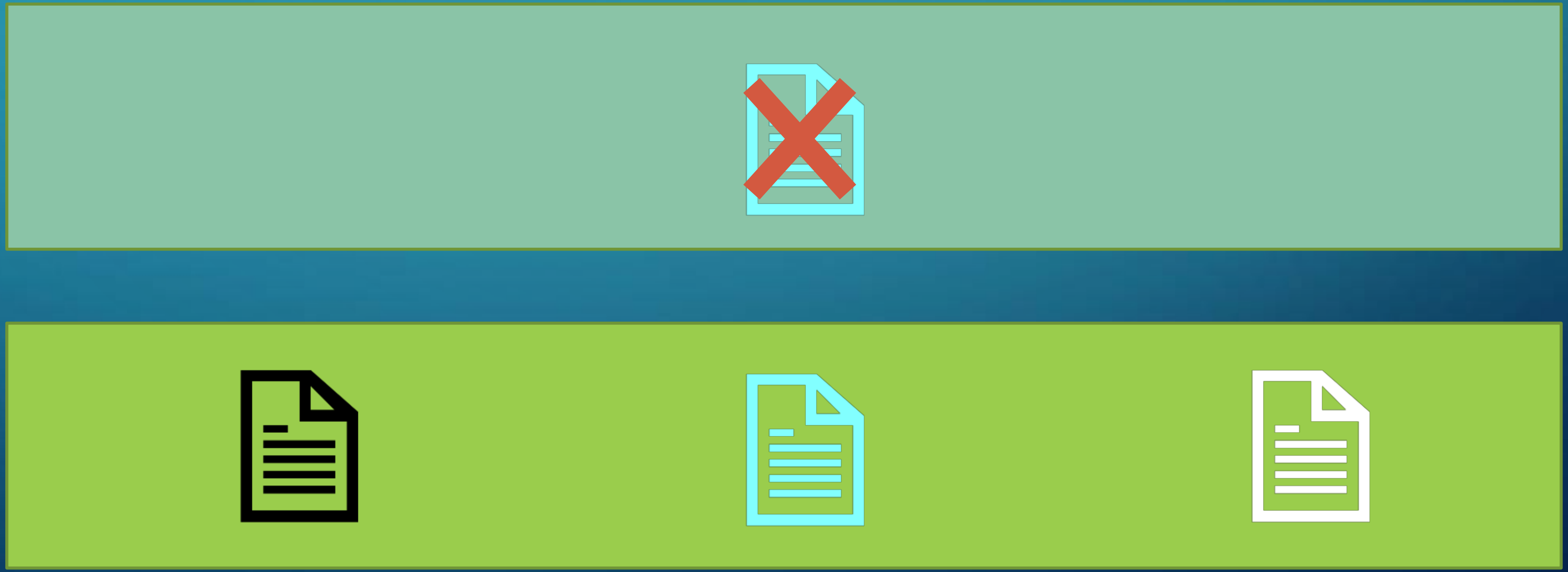
```
FROM node:latest AS storefront
WORKDIR /usr/src/atsea/app/react-app
COPY react-app .
RUN npm install
RUN npm run build

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"]
```

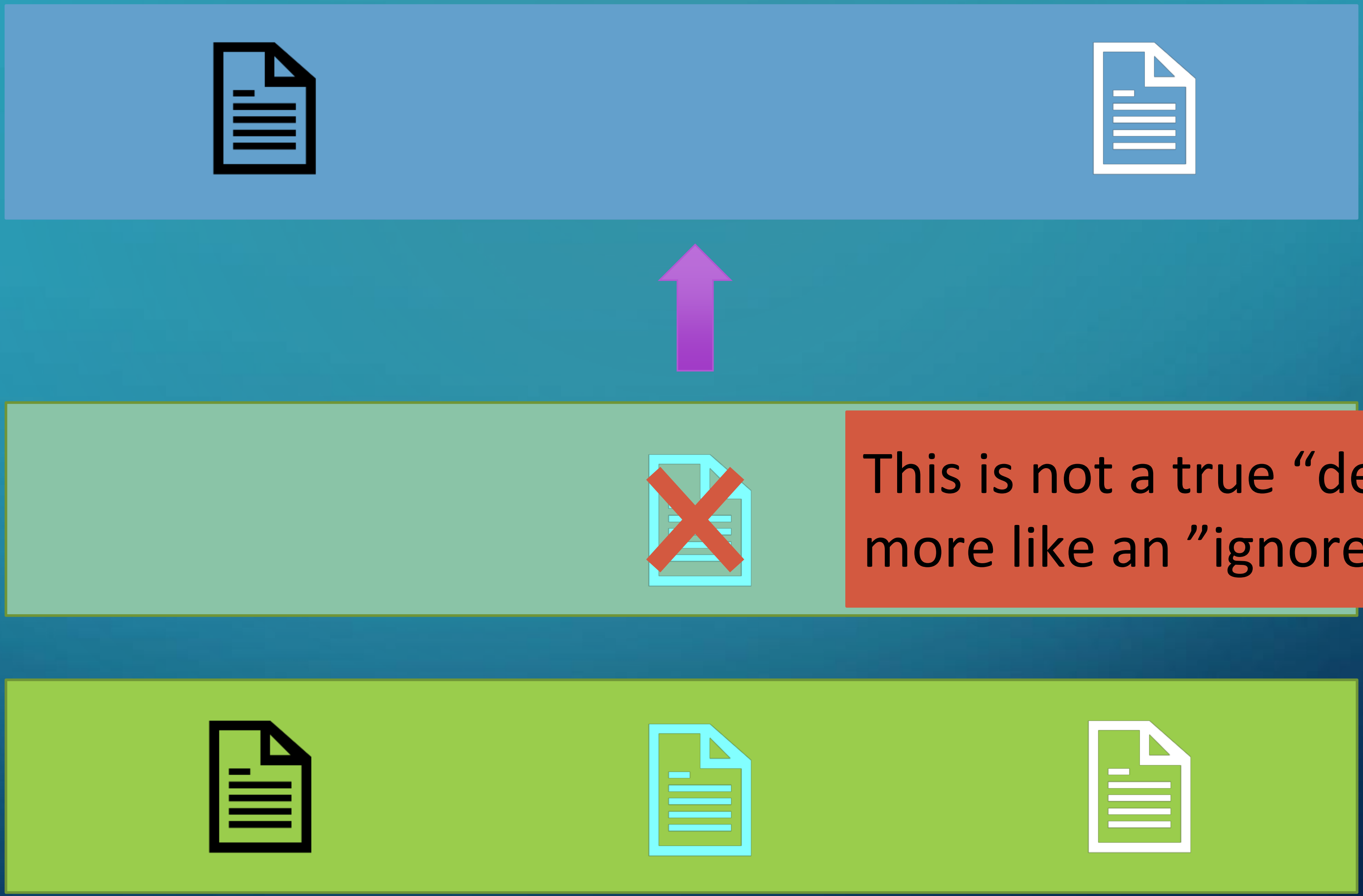
Multi-stage builds

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

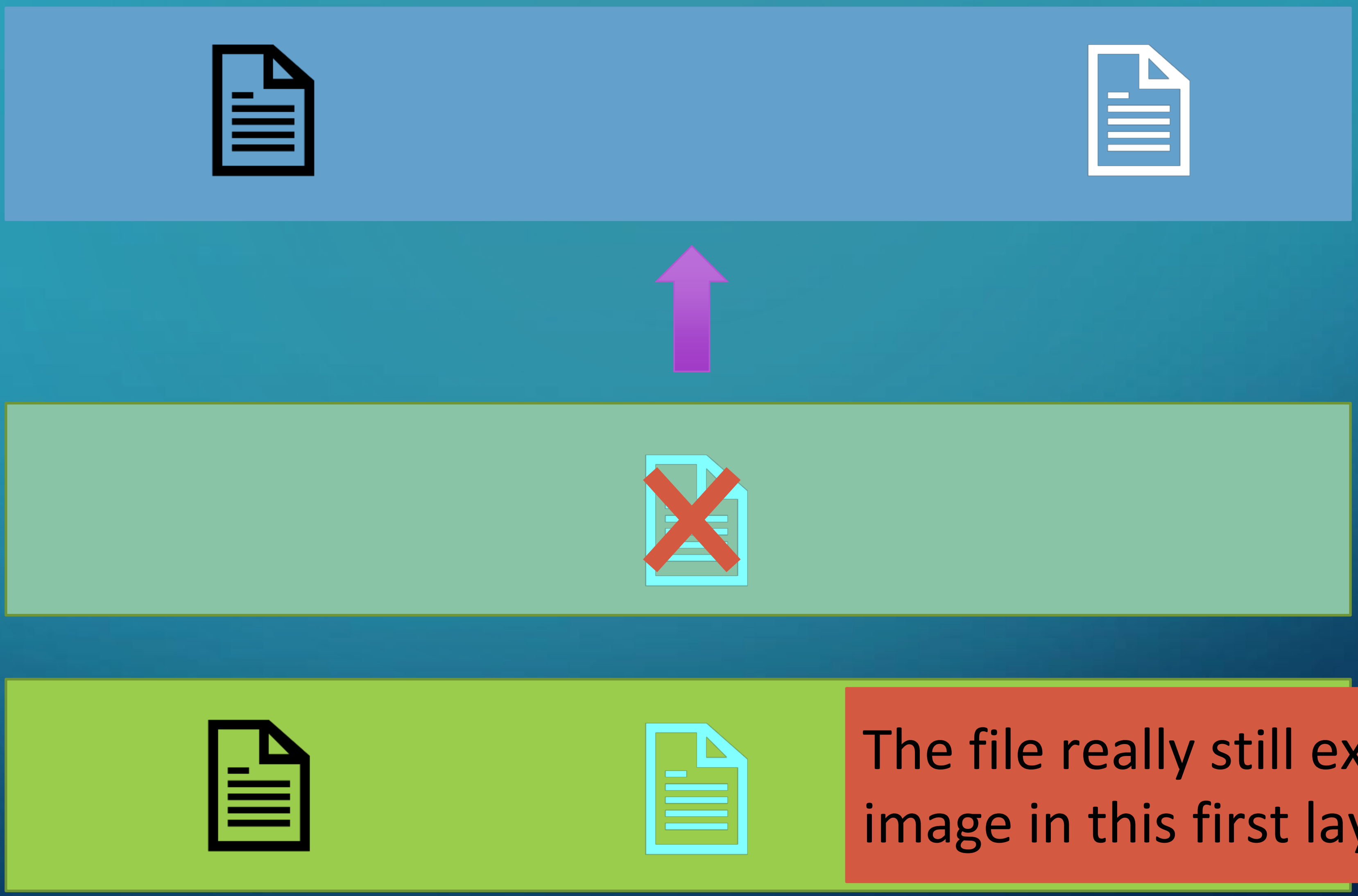
Note

- 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.

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.

```
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"]
```

MORE THINGS TO TRY ON YOUR OWN

- Build Super Link Extractor T-1000 for arm
  - 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`)