DOCKER COMPOSE

WHAT IS IT?

FEATURES

- Spin up multiple containers with one simple API
- Network containers together
- Simple YAML-based configuration language
- Easy interface for working with Docker
- Most of the common Docker commands supported in docker-compose

USE CASES

Simplify running Docker commands

- **Turn this:**

```
docker network create my-net
docker volume create my-vol
docker build -t myapp/app1:0.0.1 \
       --build-arg DEBUG=True --build-arg SITE_DIR=/site/ \
       -f Dockerfile.custom .
docker run --rm —name app1 —-port 80 \
       -v `pwd`/media/:/htdocs/media/ \
       -v my-vol:/data/ \
       —-env DB_USER=myuser —-env DB_PASS=1234 —-env DB_HOST \
       --network my-net
       myapp/app1:0.0.1
docker run --name web1 -d -p 8080:80 --network my-net nginx:13-alpine
```

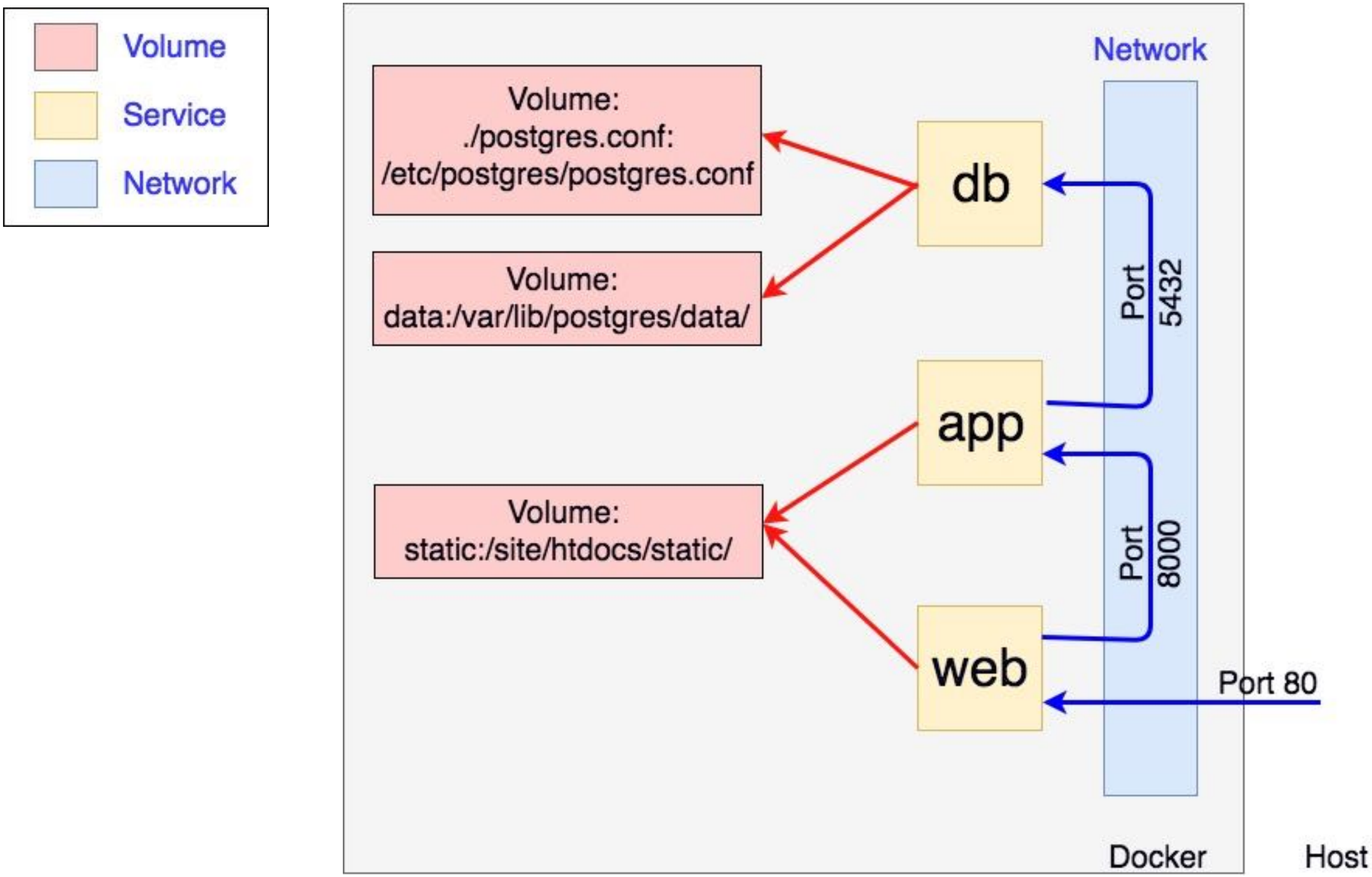
- **Into:**

  docker-compose up --build

USE CASES

- Spin up a local development environment with all required services (db, webserver, etc)
- Run run your unit tests on an actual services similar to production.
- Deploy to Prod with Docker Swarm

SIMPLE COMPOSE CONFIG



INSTALL

- Install / Run Docker
- Mac:
  - Installed with "Docker for Mac"
- Windows:
  - Installed with "Docker for Windows"
- Linux:
  - Download Link:
    - sudo curl -L https://github.com/docker/compose/releases/download/1.19.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
    - Or "sudo pip install docker-compose"

YAML

YAML

```yaml
---
first_name: joe
last_name: jasinski
age: 35
fav_colors:
  - red
  - blue
pets:
  - dog1:
      name: Toby
      age: 12
  - dog2:
      name: Max
      age: 8
```

Python

```python
{'age': 35,
 'fav_colors': ['red', 'blue'],
 'first_name': 'joe',
 'last_name': 'jasinski',
 'pets': [{'dog1': {'age': 12, 'name': 'Toby'}},
          {'dog2': {'age': 8, 'name': 'Max'}}]}
```

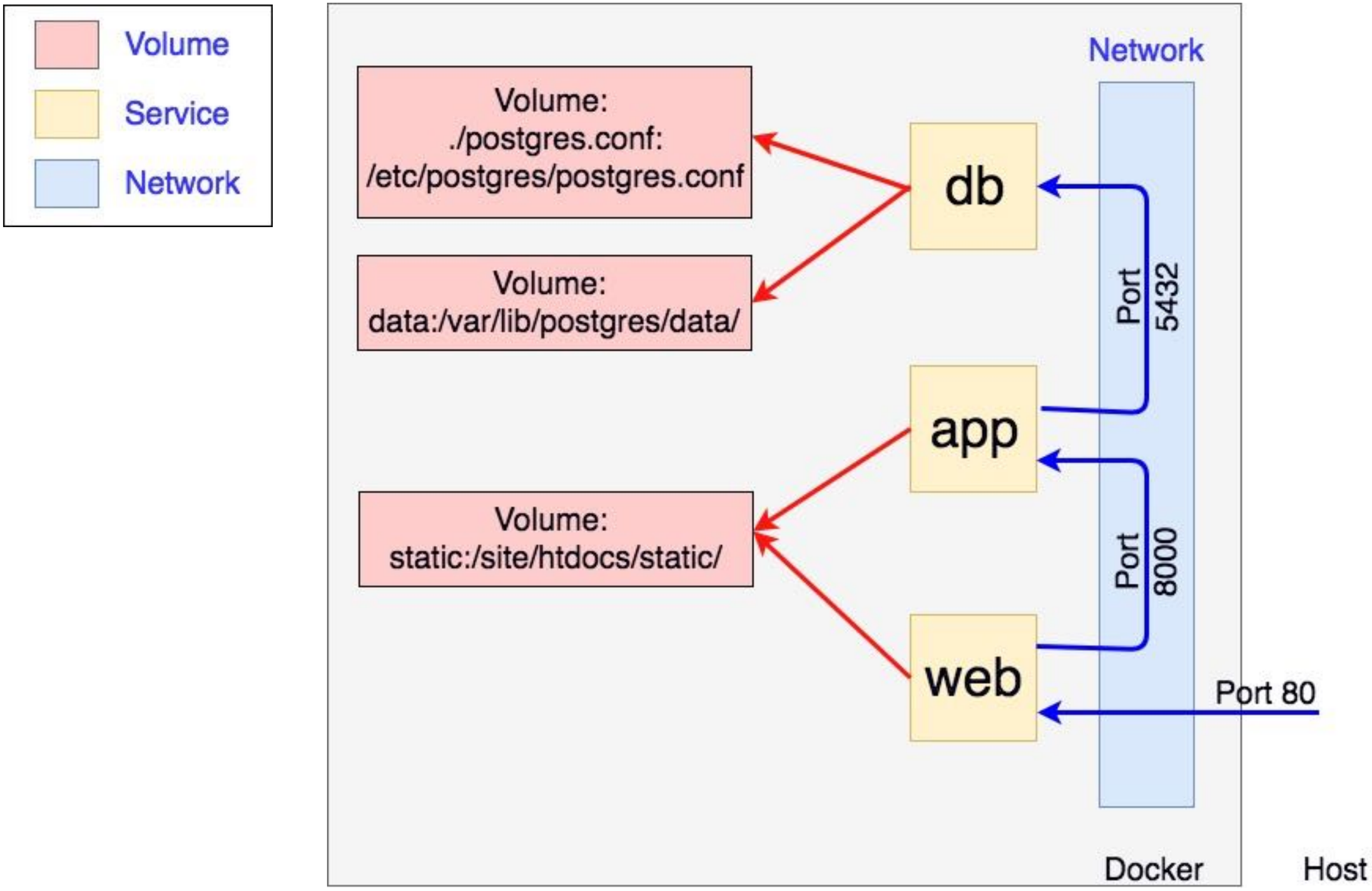
docker-compose.yml

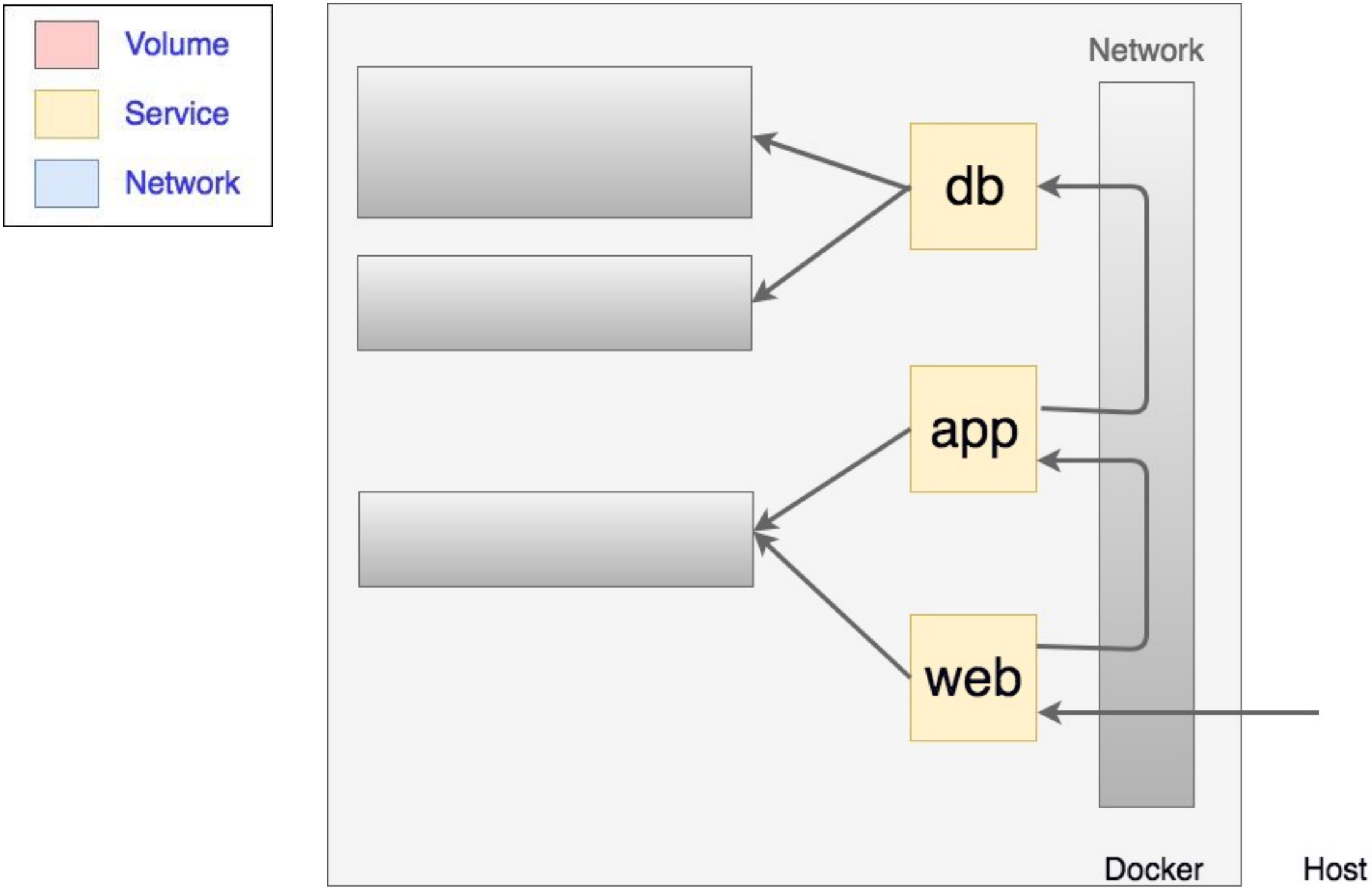
```yaml
version: '3.4'
services:

  app:
    image: docker4data/talkvoter:prod-1.0.1
    build:
      context: ./
      dockerfile: talkvoter/Dockerfile
    command: /app/docker-utils/run-app.sh
    volumes:
      - .:/app/proj/
      - static-volume:/app/htdocs/static/
    environment:
      - DATABASE_URL=postgres://postgres@db/postgres
      - SITE_DIR=/app/
      - SECRET_KEY=super secret key
      - PREDICT_ENDPOINT=http://predict:8000/predict/
    networks:
      - talkvoter_net
    depends_on:
      - db
    stdin_open: true
```

SERVICES

SERVICES

- Core building-block
- Runs a Docker Image
- Ideally should serve one purpose each
- Can be networked together
- Can mount volumes
- Examples: webserver, database, prediction app, etc.


Volume
Service
Network
Network
Volume:
./postgres.conf:
/etc/postgres/postgres.conf
Volume:
data:/var/lib/postgres/data/
Volume:
static:/site/htdocs/static/
db
app
web
Port
5432
Port
8000
Port 80
Docker
Host


Volume
Service
Network
Network
db
app
web
Docker
Host

SERVICE EXAMPLE

```yaml
version: '3.4'
services:

  web:
    image: nginx
    …

  app:
    image: myrepo/myname:mytag-0.0.1
    …

  db:
    image: postgres:10-alpine
    …
```

SIMPLEST COMPOSE FILE

```yaml
version: '3.4'
services:

  app:
    image: python:3.6.5
    command: ["python", "-mhttp.server", "8000"]
```

BUILD

BUILD

```
docker build -f Dockerfile \
   --tag myrepo/myname:mytag-0.0.1 \
   --build-arg SITE_DIR=/site/ .
```

BUILD

```
docker build -f Dockerfile \
    --tag myrepo/myname:mytag-0.0.1 \
    --build-arg SITE_DIR=/site/ .
```

```
version: '3.4'
services:

    app:
      image: myrepo/myname:mytag-0.0.1
      build:
        context: .
        dockerfile: Dockerfile
        args:
          SITE_DIR: /site/
```

DEMO (BASIC USAGE)

example01 — vim -o Dockerfile docker-compose.yml — 57×16

```
FROM python:3.6


CMD ["python", "-mhttp.server", "8000"]
~
~
~
Dockerfile
version: '3.4'
services:

  app:
    build: .

docker-compose.yml
:qa
```

DOCKER COMPOSE COMMANDS

BUILD AND RUN

```
docker-compose up --build
```

START SERVICES

```
docker-compose start
```

STOP SERVICES

```
docker-compose stop
```

RESTSRT SERVICES

```
docker-compose restart
docker-compose restart app
```

DESTROY APPLICATION

```
docker-compose down
```

RUN A CMD IN CONTAINER

```
docker-compose run app \
    /bin/bash
```

ENTER RUNNING CONTAINER

```
docker-compose exec app \
    /bin/bash
```

VIEW CONTAINER LOGS

```
docker-compose logs -f app
```

PORTS

PORTS

```
docker run \
  -p 8000 \
  -p 80:8001 \
  app:0.0.1
```

- Map a host port to a container port

PORTS

```
docker run \
  -p 8000 \
  -p 80:8001 \
  app:0.0.1
```

- Map a host port to a container port

```
version: '3.4'
services:

  app:
    ports:
      - 8000
      - 80:8001
...
```

DEMO (PORTS)

```yaml
version: '3.4'
services:

  app:
    build: .
    ports:
      - 8000:8000
```

docker-compose.yml

```dockerfile
FROM python:3.6
RUN mkdir /htdocs/
WORKDIR /htdocs/
COPY index.html /htdocs/index.html
CMD ["python", "-mhttp.server", "8000"]
```

Dockerfile

```
:qa
```

Problem loading page

localhost:8000

Unable to connect

Firefox can't establish a connection to the server at localhost:8000.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's

COMMANDS

COMMAND

```
version: '3.4'
services:

   app:
      …
       command: ./docker-utils/app-start.sh –DEBUG
       …
```

```
version: '3.4'
services:

   app:
      …
       command: ["./docker-utils/app-start.sh", "--DEBUG"]
       …
```

COMMAND

Docker CLI

```
docker run app:0.0.1 /bin/run.sh
```

Dockerfile

```
CMD ["/app/run.sh"]
```

Compose File

```
version: '3.4'
services:
    app:
      command: /app/run.sh
```

Compose CLI

```
docker-compose run app \
  /app/run.sh
```

ENVIRONMENT

ENVIRONMENT VARS

```
docker run \
  --env SITE_DIR=/site/ \
  -e DEBUG=True \
  app:0.0.1
```

- Pass in environment variables to the container
- Useful for specifying configuration

ENVIRONMENT VARS

```
docker run \
  --env SITE_DIR=/site/ \
  -e DEBUG=True \
  app:0.0.1
```

- Pass in environment variables to the container
- Useful for specifying configuration

```
version: '3.4'
services:

  app:
    environment:
      - DEBUG=True
      - ENV_ROOT=/site/
    ...

  db:
    environment:
      - PG_USER=example
      - PG_PASSWORD=change
      - PG_DB=mydb
    ...
```

ENV
VS
ENV_FILE

```yaml
version: '3.4'
services:

  app:
    env_file:
     - .env
```

```yaml
version: '3.4'
services:

  app:
    environment:
     - DEBUG=True
     - ENV_ROOT=/site/
```

```
# .env
DEBUG=True
ENV_ROOT=/site/
```

DEMO (ENV)

docker-compose.yml

```yaml
version: '3.4'
services:

  app:
    build: .
    ports:
      - 8000:8000
    command: ["flask", "run"]
    environment:
      - FLASK_APP=demo_app.py
      - DEBUG=True
      - PORT=8000
```

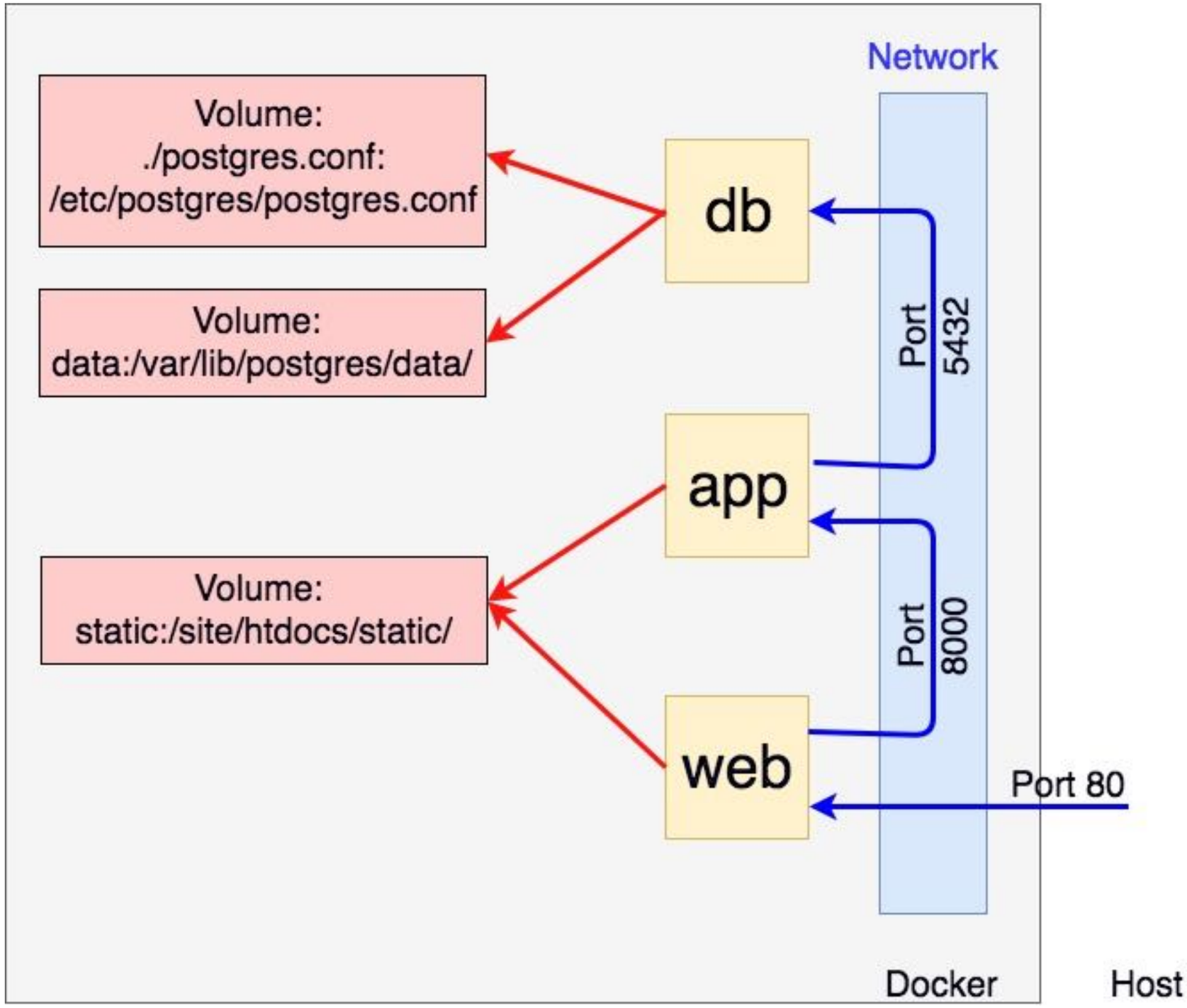
Dockerfile

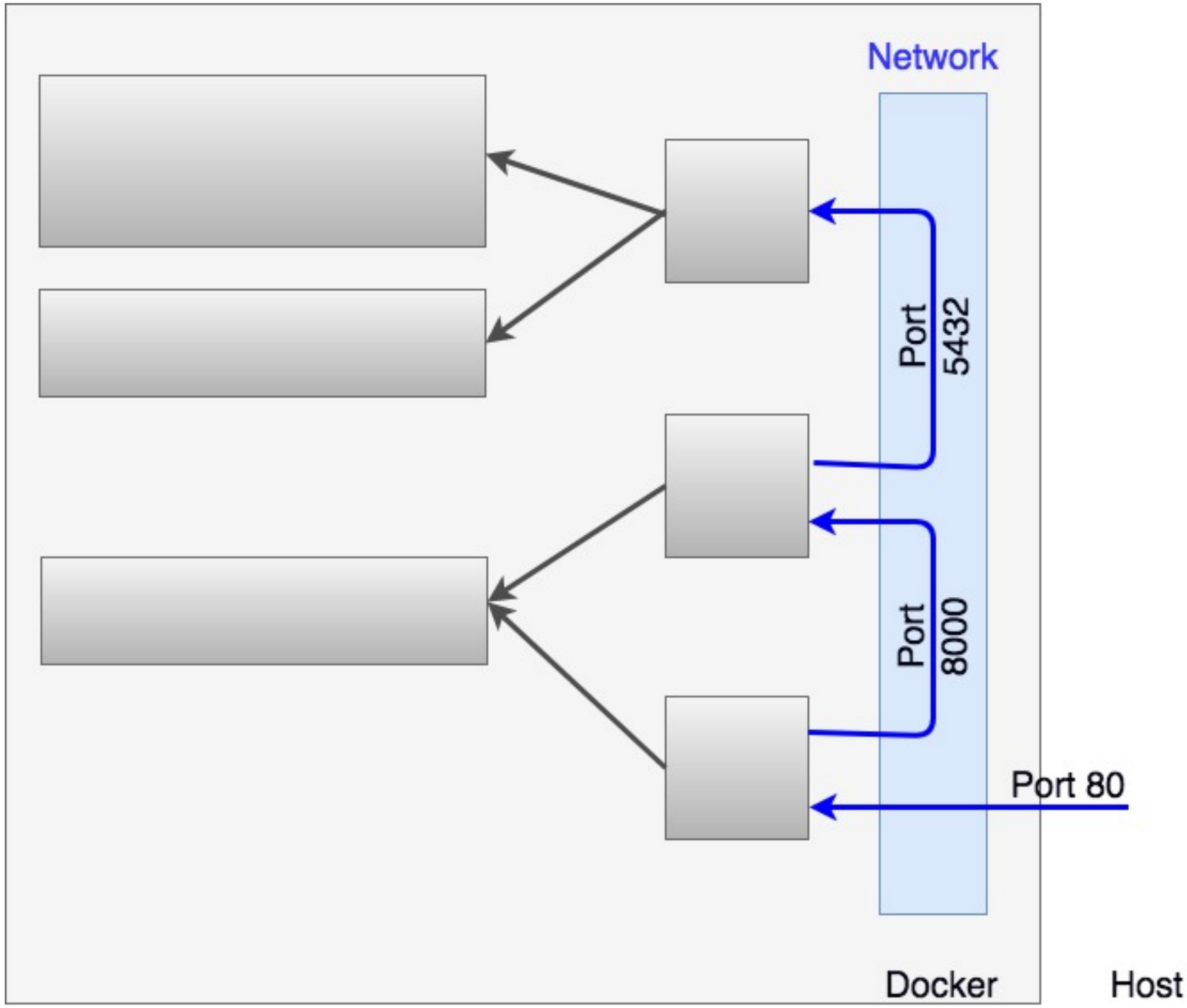
```dockerfile
FROM python:3.6
RUN pip install Flask
RUN mkdir /app/
WORKDIR /app/
COPY demo_app.py /app/demo_app.py
```

demo_app.py

```python
import os
from flask import Flask
app = Flask(__name__)

DEBUG = os.getenv("DEBUG", None)
PORT = int(os.getenv("PORT", "8000"))


@app.route("/")
def hello():
    return (
        "Hello World! "
        "DEBUG {} PORT {}".format(
            DEBUG, PORT))


app.run(
    host='0.0.0.0',
    port=PORT,
    debug=bool(DEBUG == "True"))
```

NETWORKS


Network
Volume:
./postgres.conf:
/etc/postgres/postgres.conf
Volume:
data:/var/lib/postgres/data/
Volume:
static:/site/htdocs/static/
db
app
web
Port
5432
Port
8000
Port 80
Docker
Host


Network
Port
5432
Port
8000
Port 80
Docker
Host

NETWORKS

```
docker network create internal

docker run --network internal \
  myapp:0.0.1 /bin/bash

docker run --network internal \
  postgres
```

NETWORKS

```yaml
version: '3.4'
services:

    app:
      image: myapp
      networks:
        - internal
      ...

    db:
      image: postgres
      networks:
        - internal
      ...

networks:
  internal:
```

- Services can communicate with each other if they are on the same network.
- Services can reference each other using service name as hostname
- Networks should be defined with a networks section
- You can have multiple networks per compose configuration

NETWORKS

```
version: '3.4'
services:

    app:
      image: myapp

     …

    db:
      image: postgres
      …
```

- Specifying the network in compose is optional
- A default network will be created if you do not specify one

NETWORKS

```yaml
version: '3.4'
services:

    app:
      image: myapp
      environment:
        - DB_HOST=db
        - DB_PORT=5432
     ...

    db:
      image: postgres
     ...
```

Reference other hosts using the *service name* in Docker compose

Use that name in:

- Docker Container
- Compose File

NETWORKS

```yaml
version: '3.4'
services:

  app:
    image: myapp
    environment:
      - DB_HOST=db
      - DB_PORT=5432
    ...

  db:
    image: postgres
    ...
```

Reference other hosts using the *service name* in Docker compose

Use that name in:

- Docker Container
- Compose File

Ex: from inside of the "app" container, we can ping the "db" container

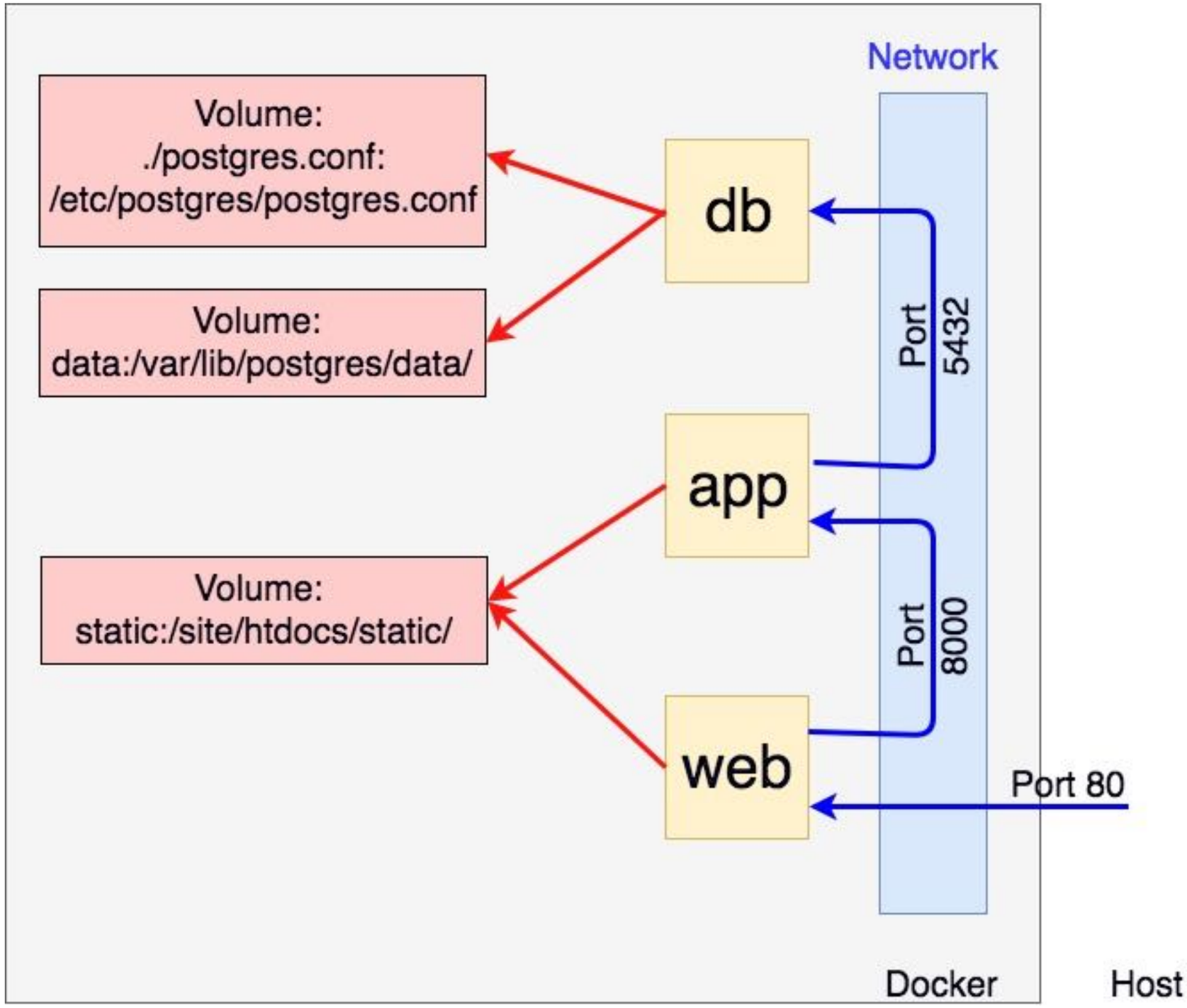
```
$ docker-compose run app /bin/bash
Starting dcdemo_db_1 ... done
root@d17597e732e3:/app# ping db
PING db (172.28.0.2): 56 data bytes
64 bytes from 172.28.0.2: icmp_seq=0 ttl=64 time=0.131 ms
64 bytes from 172.28.0.2: icmp_seq=1 ttl=64 time=0.192 ms
```

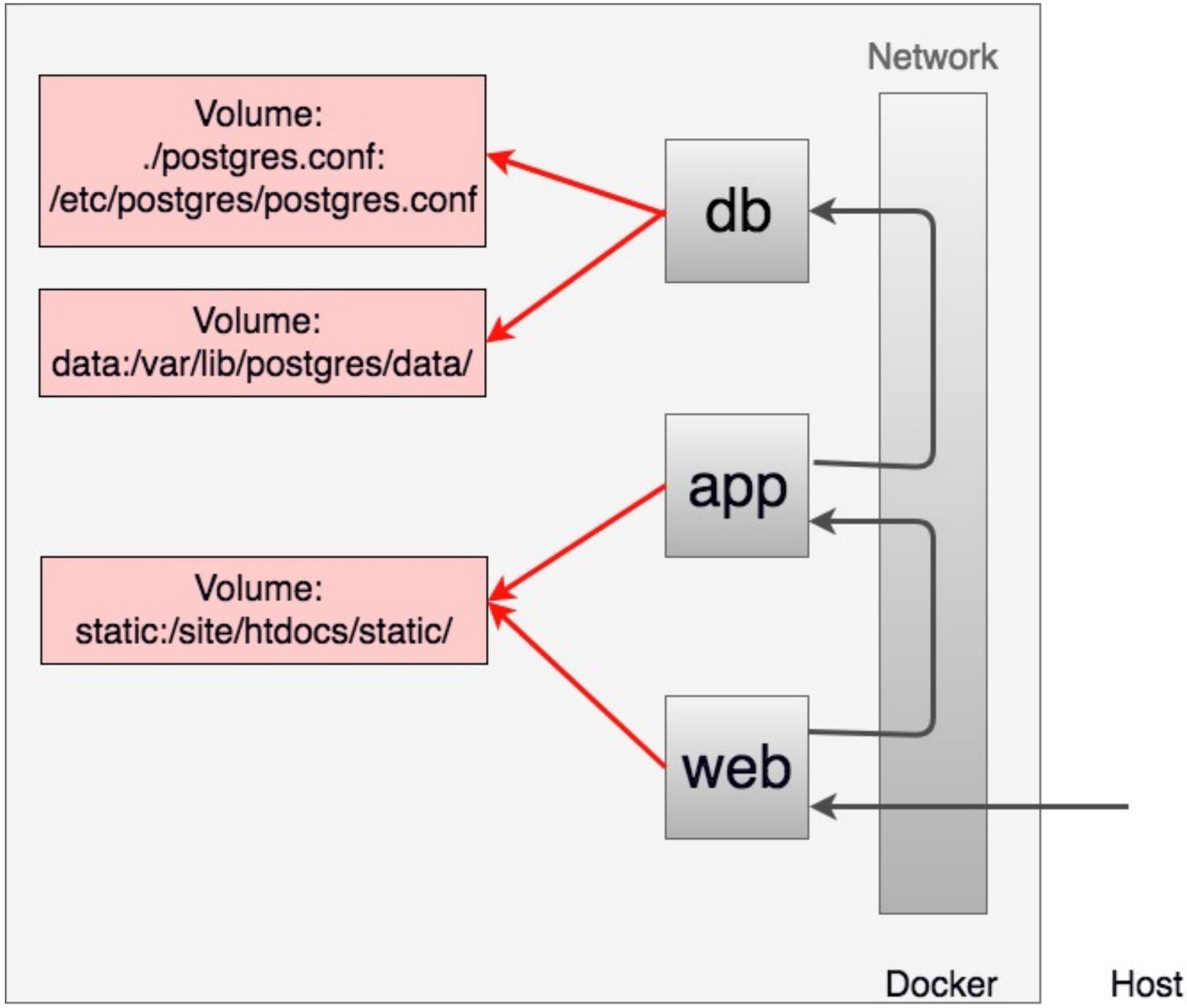
DEPENDS_ON

```
version: '3.4'
services:

    app:
      image: myapp
      environment:
        - DB_HOST=db
        - DB_PORT=5432
      depends_on:
        - db
        …
    db:
      image: postgres
      …
```

- Start app service after db service
- Does NOT wait for app command to boot

VOLUMES


Network
Volume:
./postgres.conf:
/etc/postgres/postgres.conf
Volume:
data:/var/lib/postgres/data/
Volume:
static:/site/htdocs/static/
db
app
web
Port
5432
Port
8000
Port 80
Docker
Host


Network
Volume:
./postgres.conf:
/etc/postgres/postgres.conf
db
Volume:
data:/var/lib/postgres/data/
app
Volume:
static:/site/htdocs/static/
web
Docker
Host

VOLUMES

```
docker volume create my_data

docker run \
  -v my_data:/tmp/ \
  -v .:/media/ \
  web:0.0.1 /bin/bash

docker run \
   -v my_data:/data/ \
   db:0.0.1 /bin/bash
```

VOLUMES

```
version: '3.4'
services:

    app:
      …
       volumes:
        - my_data:/data/
        - .:/source/
       …

    db:
      …
       volumes:
        - my_data:/data/
       …

volumes:
  my_data:
```

- Several Forms:
  - Bind mounts shares data on a host machine to a docker container
  - Volumes create a virtual volume to attach or share share between containers

COMPOSE FILES

MULTIPLE COMPOSE FILES

- Merge multiple compose files can be used together (-f flag)
- **docker-compose.override.yml** is a baked-in override
- One compose file with defaults; another with private settings.
- Split out different environments

MULTIPLE COMPOSE FILES

docker-compose automatically merges **override** file

```yaml
# docker-compose.yml

version: '3.4'
services:
   app:
     image: d4d/talkvoter:prod-1.0
     build:
       context: .
       dockerfile: Dockerfile
     command: /app/utils/run.sh
     volumes:
       - .:/app/proj/
       - static-vol:/app/htdocs/
     environment:
       - DATABASE_URL=
       - SITE_DIR=/app/
       - SECRET_KEY=secret key
```

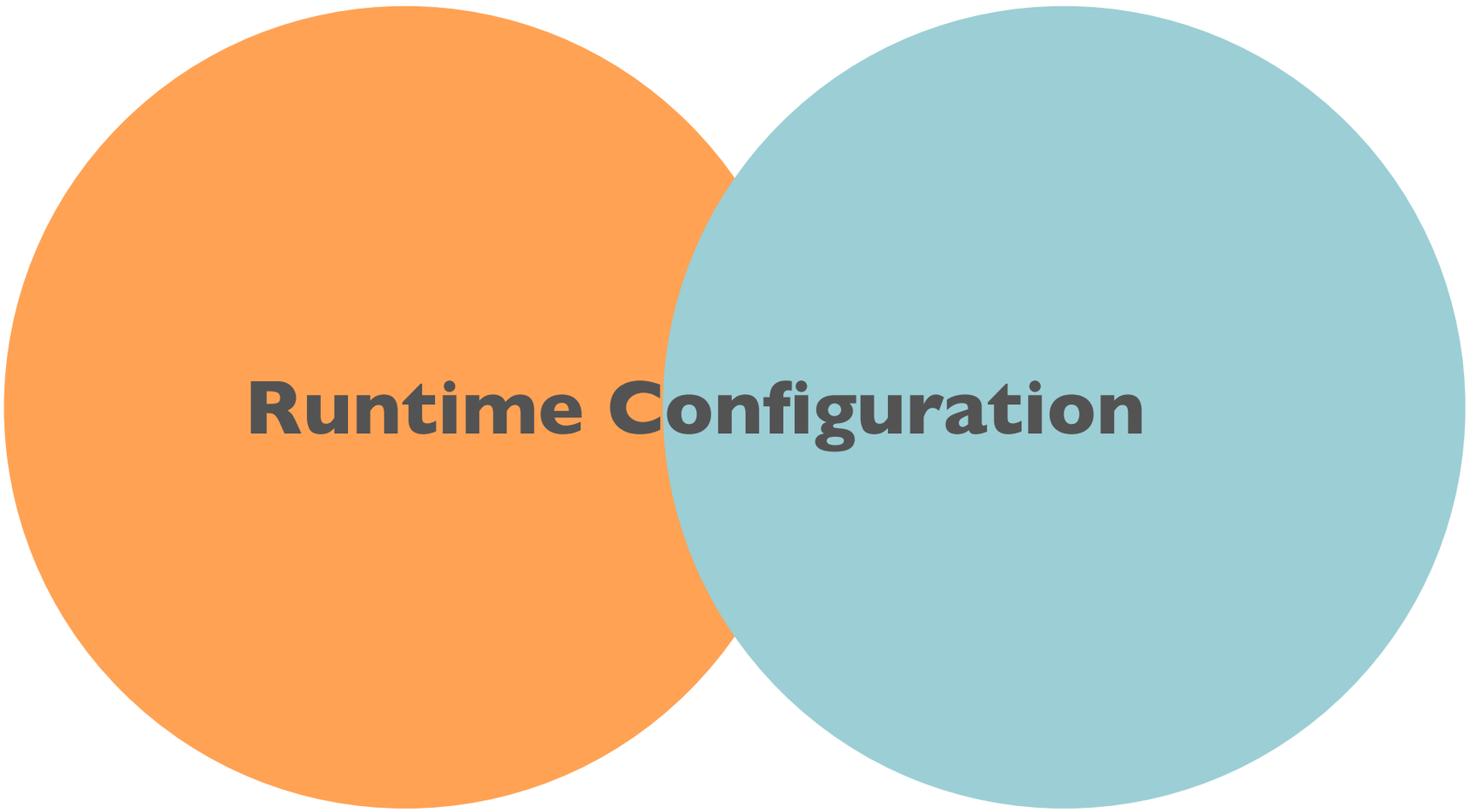
```yaml
# docker-compose.override.yml

version: '3.4'
services:
   app:
     image: experiments:0.0.1
     ports:
       - "8000:8000"
     environment:
       - DATABASE_URL=secret key
```

MULTIPLE COMPOSE FILES

docker-compose automatically merges **override** file

```
# docker-compose.yml
```

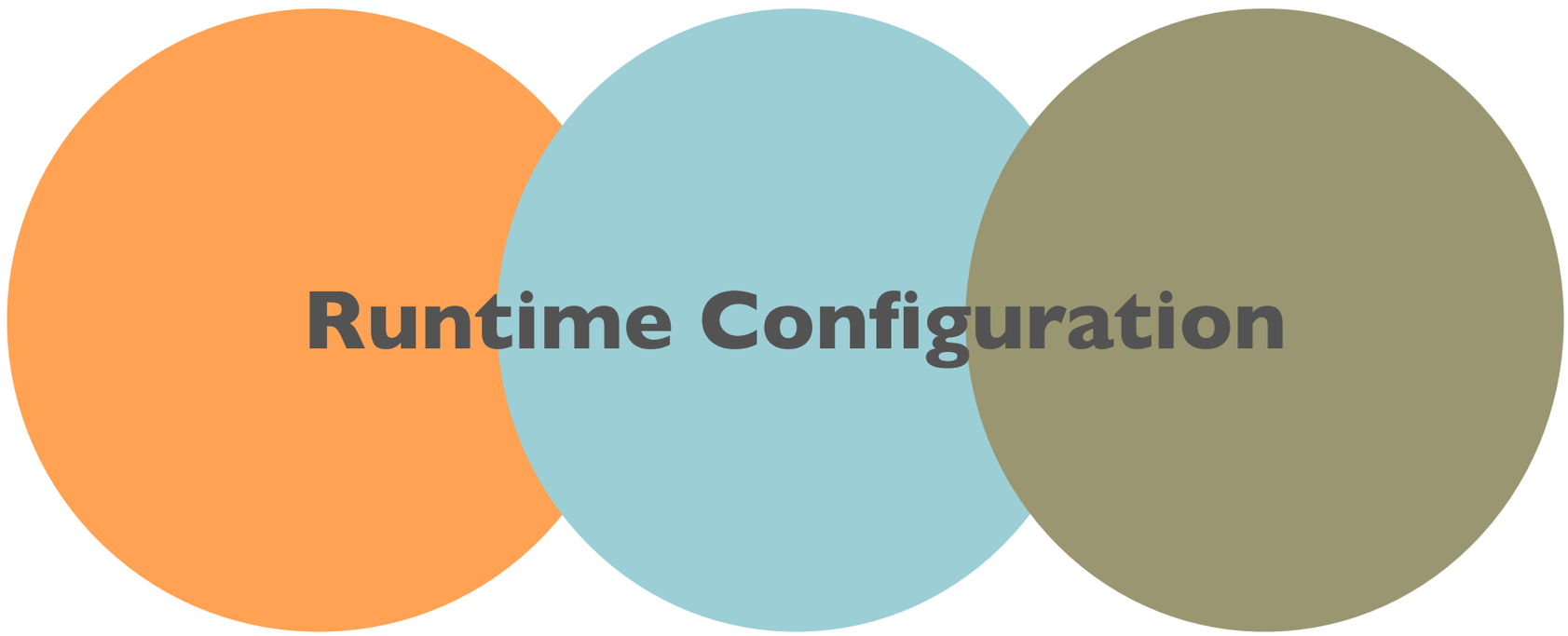
```
# docker-compose.override.yml
```



MULTIPLE COMPOSE FILES

The **-f option** allows arbitrary numbers of compose files

docker-compose.yml **docker-compose.A.yml** **docker-compose.B.yml**




```
docker-compose -f docker-compose.yml \
               -f docker-compose.A.yml \
               -f docker-compose.B.yml ...
```

MULTIPLE COMPOSE FILES

single-value options image, command, entrypoint	Value is replaced
multi-value options ports, expose, dns	Concatenates values
Key/Value options environment, labels, volumes	Concatenates values; replaces existing

MISC

RESOURCE NAMING

- Containers, Volumes, Networks are prefixed with the directory name

- This can be overridden with the project option:

  -p, --project-name NAME

- Useful if spinning up the same compose stack on a single machine.

RESOURCE NAMING

Containers

```
MacBook-Pro:docker4data jjasinski$ docker container ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS                                  PORTS                     NAMES
2bc1a9018e6f        docker4data/talkvoter:prod-1.0.1   "./docker-utils/entr…"   About a minute ago  Up About a minute (health: starting)    0.0.0.0:8000->8000/tcp    docker4data_app_1
d9026f2c69ea        postgres:10-alpine             "docker-entrypoint.s…"   2 hours ago         Up About an hour (healthy)              0.0.0.0:32779->5432/tcp   docker4data_db_1
```

```
      NAMES
p     docker4data_app_1
cp    docker4data_db_1
```

Volumes

```
$ docker volume ls | grep docker4data
local               docker4data_pgdata
local               docker4data_static-volume
```

Networks

```
$ docker network ls  | grep docker4data_talk
0d5b7379a42b        docker4data_talkvoter_net     bridge              local
```

DEBUGGING (PDB)

- For development only
- Equivalent to the -i and -t Docker command line options
- stdin_open: true
  tty: true
- docker attach to get to interactive pdb terminal
- Process managers (systemd, supervisord) get in the way
- For uwsgi, pass in `--honour-stdin`
- For gunicorn, pass in timeout option: `-t 3600`

https://blog.lucasferreira.org/howto/2017/06/03/running-pdb-with-docker-and-gunicorn.html

AUTORELOAD

- Restart app on code change
- Good for development; **not** for production
- Not something handled natively by Docker or Compose
- Volume mount needed between the host and container codebase
- Uwsgi and Gunicorn (and Django runserver) support auto reload options
  - Uwsgi: `--python-autoreload=1`
  - `Gunicorn: --reload`

DATA

EXAMPLE: POSTGRES CONFIG

```yaml
version: '3.4'
services:

  db:
    image: postgres:10-alpine
    environment:
     - POSTGRES_USER=myuser
     - POSTGRES_PASSWORD=1234
     - POSTGRES_DB=mydb
    volumes:
     - pgdata:/var/lib/postgresql/data
    ports:
     - "5432:5432"

volumes:
  pgdata:
```

POSTGRES: IMPORT/EXPORT

1) RUN DOCKER COMPOSE

```
docker-compose up --build
```

2) GET DB CONTAINER NAME

```
docker-compose ps db
```

3) DUMP DATA

```
docker exec -t -u postgres \
    db_container pg_dump dbname > dbname_db.sql
```

4) LOAD DATA

```
cat dbname_db.sql | docker exec -i -u postgres \
    db_container psql dbname
```

EXAMPLE: MYSQL CONFIG

```yaml
version: '3.4'
services:

  db:
    image: mysql:5.7
    environment:
     - MYSQL_ROOT_PASSWORD=1234
     - MYSQL_USER=myuser
     - MYSQL_PASSWORD=1234
     - MYSQL_DATABASE=mydb
    volumes:
     - mysqldata:/var/lib/mysql
    ports:
     - "3306:3306"

volumes:
  mysqldata:
```

MYSQL: IMPORT/EXPORT

1) RUN DOCKER COMPOSE

```
docker-compose up --build
```

2) GET DB CONTAINER NAME

```
docker-compose ps db
```

3) DUMP DATA

```
docker exec db_container /usr/bin/mysqldump \
    -u root --password=root dbname > dbname_db.sql
```

4) LOAD DATA

```
cat dbname_db.sql | docker exec -i db_container \
    /usr/bin/mysql -u root --password=root dbname
```

LOADING DATA INTO MONGODB

Dump Data to mongo_data/ dir
mongodump --ssl --db mongodb --host mongohost.com --port 26282 --username mongouser --password "password" -o mongo_data/

Create a "helper" container that mounts the sunlight_mongo_storage volume
docker run -v mongo_storage:/data/ --name helper busybox true

Copy Data from outside of the container to that volume (via the container)
docker cp ./mongo_data helper:/data/

Rm the "helper" container:
docker rm helper

Inside of the container, run a mongo restore
run the mongo container
docker run mongo -v mongo_storage:/data/db mongo # or run with docker compose
docker exec my_mongo_container mongorestore -d mydb /data/db/mongo_data/

Source: https://stackoverflow.com/questions/37468788/what-is-the-right-way-to-add-data-to-an-existing-named-volume-in-docker

SUMMARY

- Why it's useful
- Basic Command Usage
- Compose File Format
- Overview of Services, Networks, Volumes, Env Vars
- Helpful hints

OTHER RESOURCES

- Official Docs: https://docs.docker.com/compose/
- Blog Post with Django: http://www.djangocurrent.com/2017/06/django-docker-and-celery.html
- Kompose - Tool for converting compose files to Kubernetes-compatible configuration

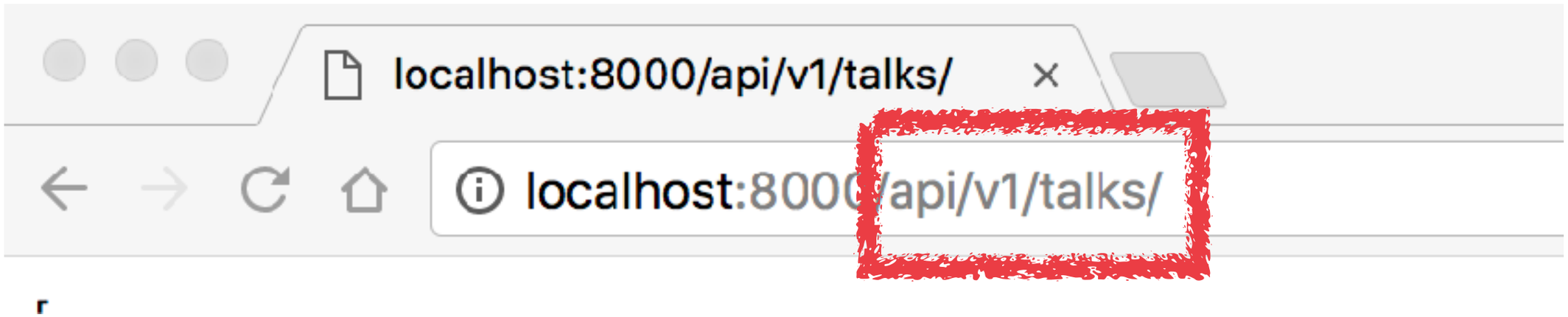
REST

(REPRESENTATIONAL STATE TRANSFER)

REST

- Standardize HTTP
- Stateless
- CRUD
- Client / Server Independence

REQUEST PATH



BAD REQUEST PATH

http://example.com/ ....

/get_people/
/person_id/?person=1
/get_children_of_person/?pid=1
/child_of_person/?pid=1&cid=2

REQUEST PATH

http://example.com/ ....

/people/
/people/<pid>/
/people/<pid>/children/
/people/<pid>/children/<child_id>/

i.e. http://example.com/people/2/children/1/

REQUEST PATH

http://localhost:8000/ ....

/talks/ - refer to all talks

/talks/<id>/ - refer to individual talk

/talks/<id>/vote/ - perform action (vote) on talk

i.e. http://localhost:8000/talks/2/vote/

VERBS

VERBS

- **GET - retrieve**
- **POST - create**
- PUT - update/replace
- **DELETE - delete**
- HEAD - get headers
- PATCH - update/modify

VERBS

GET /people/ - list people

GET /people/<*pid*>/ - get person

POST /people/<*pid*>/ - add person

PUT /person/<*pid*>/ - update person

DELETE /person/<*pid*>/ - delete person

GET /person/<id>/children/<id>/

RESPONSE

- 20x
  - 200 - OK
  - 201 - created
- 30x
  - 301 - moved permanently
  - 305 - not modified
  - 307 - moved temp
- 40x
  - 400 - bad request
  - 401 - unauthorized
  - 403 - forbidden
  - 404 - not found
  - 418 - I'm a teapot
- 50x
  - 500 - internal error
  - 501 - not implemented
  - 504 - timeout

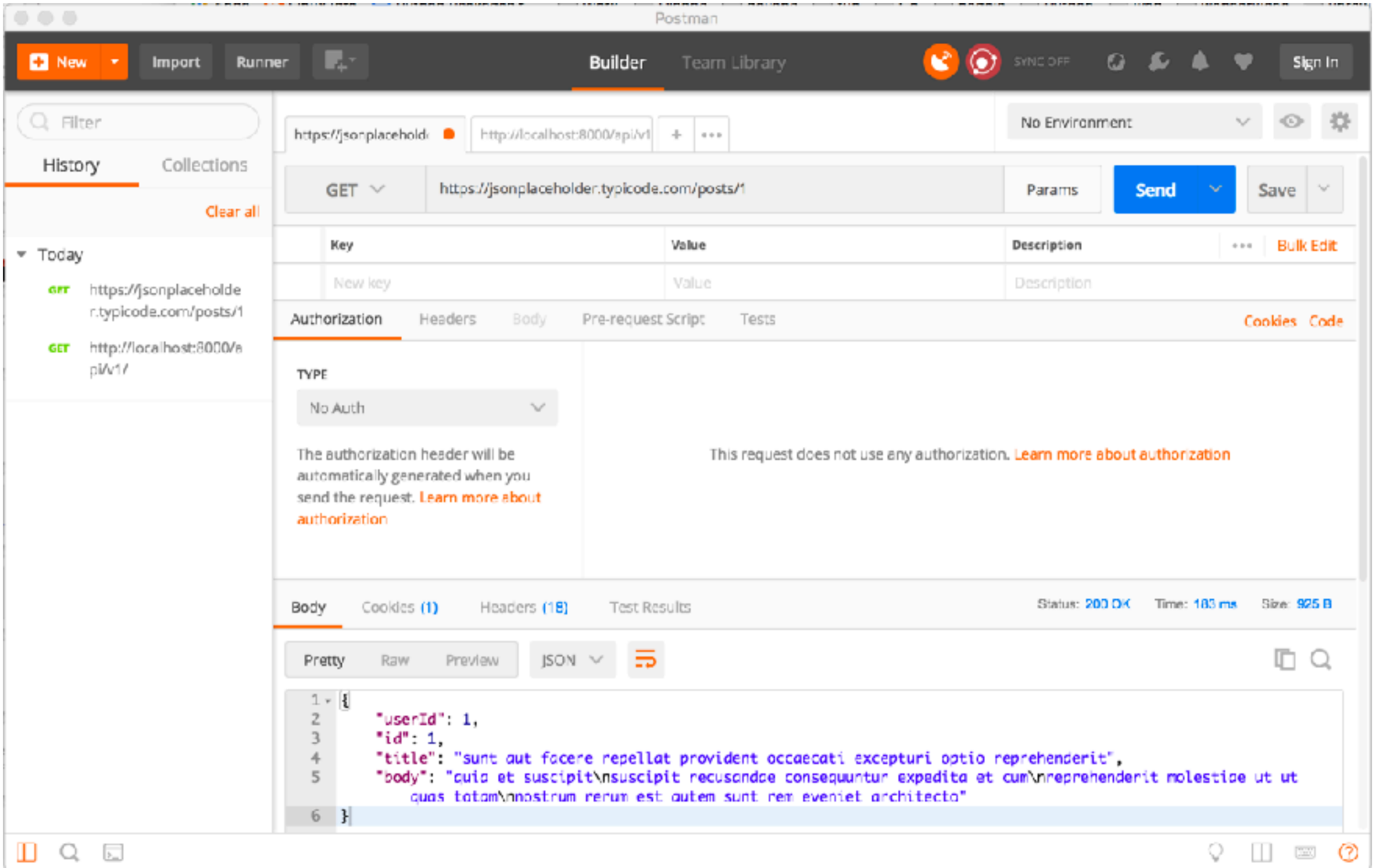
https://httpstatuses.com/

CLIENTS

CLIENTS: CURL

- curl -X GET http://example.com/posts/1/
- curl -X POST http://example.com/posts/2/ \
  -d "param1=value1&param2=value2"

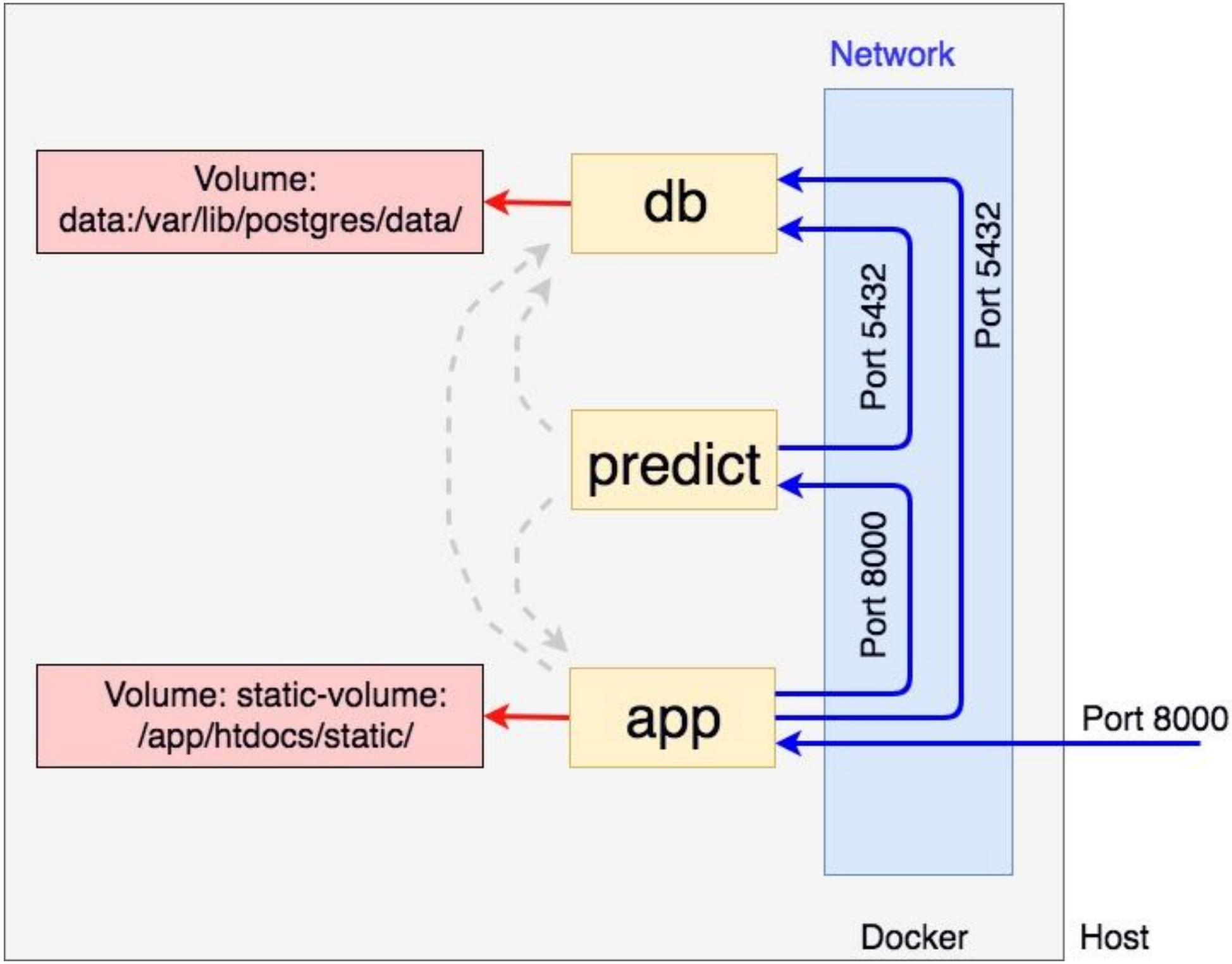
CLIENTS: POSTMAN



CLIENTS: REQUESTS

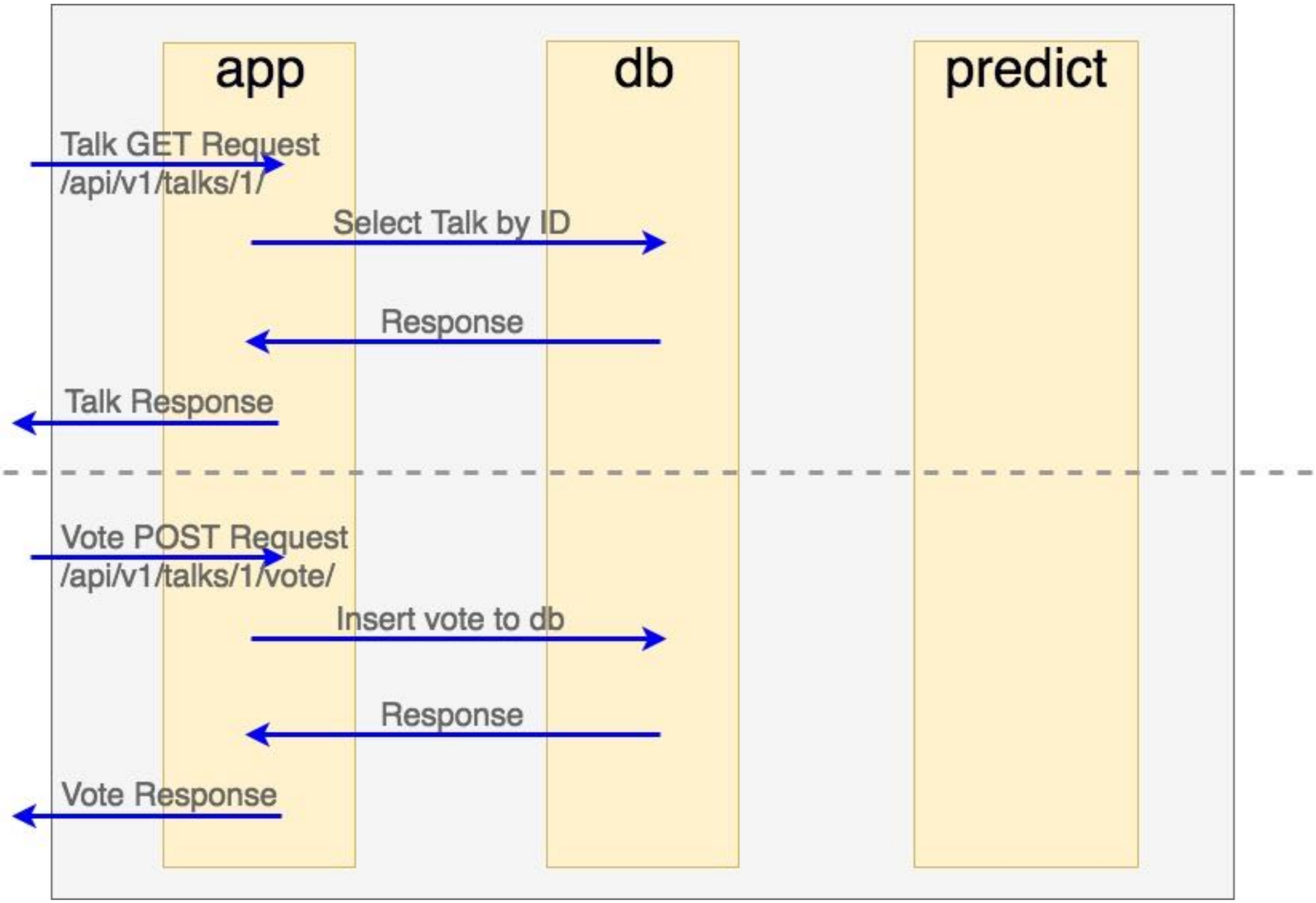
```
import requests
r = request.get("http://localhost:8000/api/talks/2/")
r.status_code
r.text
r.json()
```

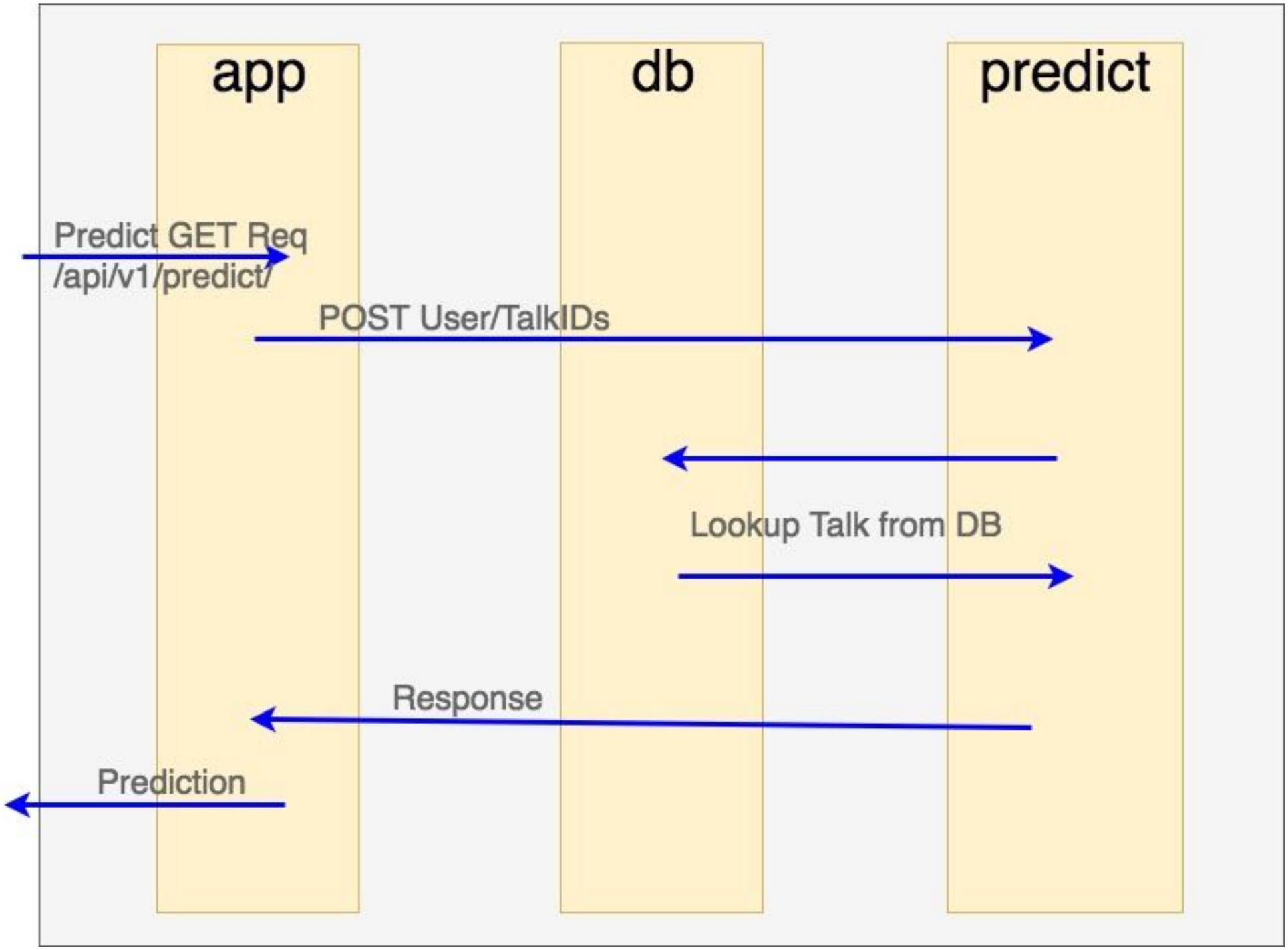
PROJECT COMPOSE DIAG



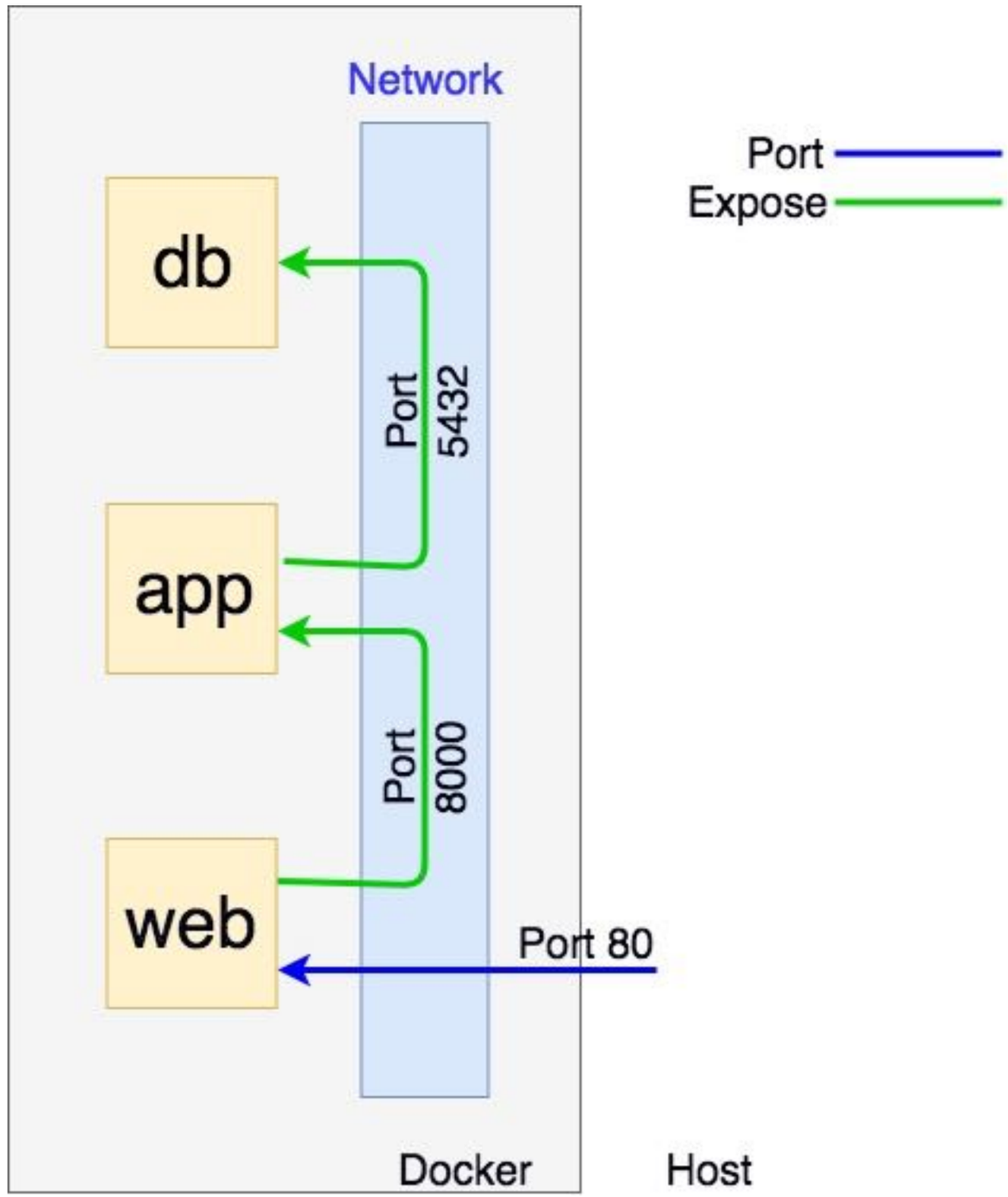
PROJECT ENDPOINTS

- /api/v1/talks/ - list talks
  /api/v1/talks/<id>/ - talk detail
  /api/v1/talks/random/ - random talk
  /api/v1/talks/vote/ - vote on talk
  /api/v1/predict/ - predict


app
db
predict
Talk GET Request
/api/v1/talks/1/
Select Talk by ID
Response
Talk Response
Vote POST Request
/api/v1/talks/1/vote/
Insert vote to db
Response
Vote Response


app
db
predict
Predict GET Req
/api/v1/predict/
POST User/TalkIDs
Lookup Talk from DB
Response
Prediction

EXPOSE VS PORTS



EXPOSE

```
docker run \
  -p 80 \
  nginx

docker run \
  --expose 8000 \
  app
```

- Expose will NOT publish the port to the host machine

```
version: '3.4'
services:

   web:
      image: nginx
      port:
       - "80"
       ...

   app:
      expose:
       - "8000"
       ...
```

VERSIONS

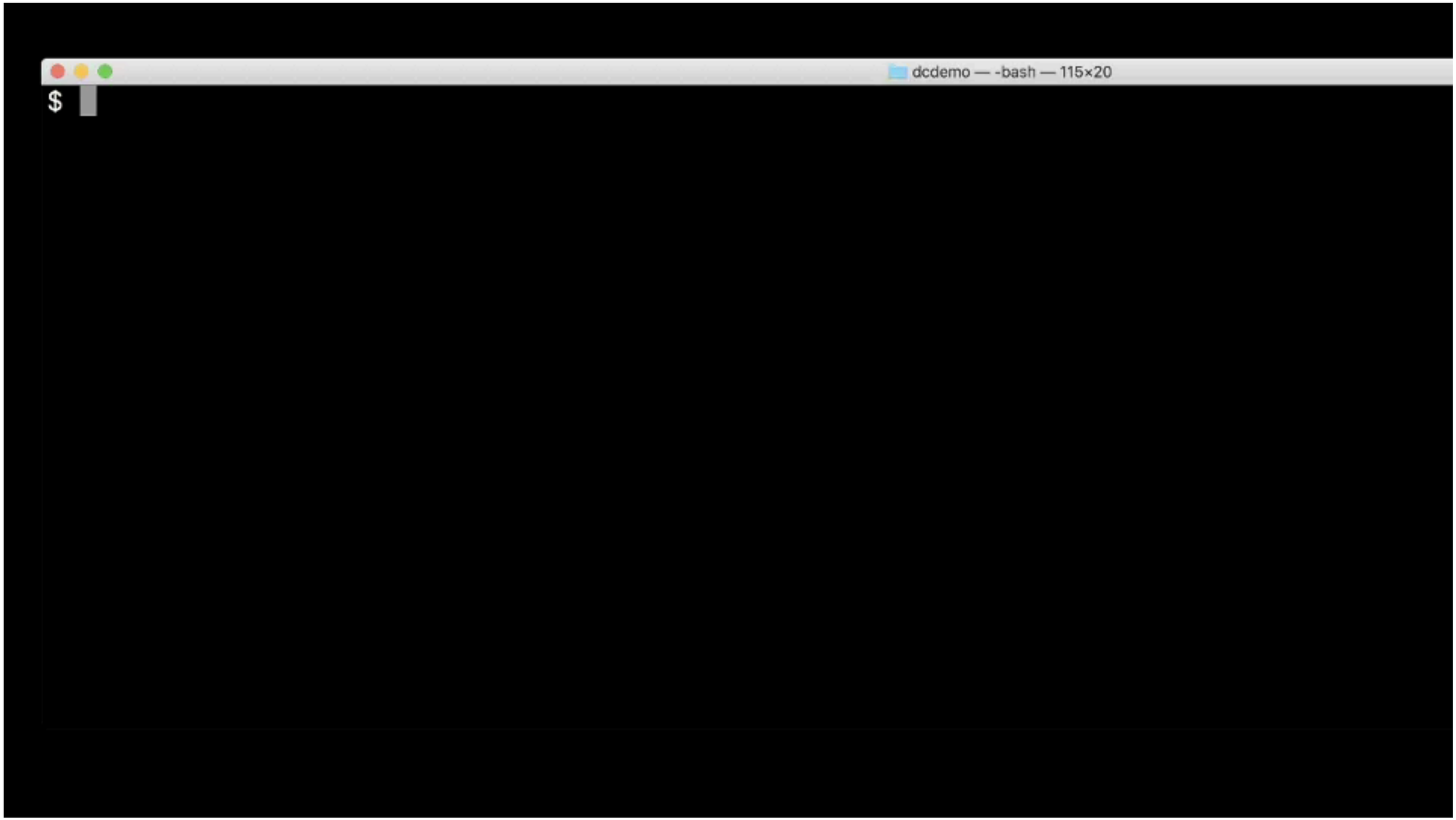
- Version 2.x (latest 2.4)
- Version 3.x (latest 3.6)

DEMO (COMMAND)

docker-compose.yml

```yaml
version: '3.4'
services:

  app:
    build: .
    ports:
      - 8000:8888
    command: ["python", "-mhttp.server", "8888"]

  db:
    image: postgres:10-alpine
```

DEMO (RUN)



ENTRYPOINT

Docker CLI

```
docker run \
  --entrypoint /app/ep.sh \
  app:0.0.1
```

Dockerfile

```
ENTRYPOINT ["/app/ep.sh"]
```

Compose File

```
version: '3.4'
services:
  app:
    entrypoint: /app/ep.sh
```

Compose CLI

```
docker-compose run app \
  --entrypoint /app/ep.sh
```

CONNECT TO DOCKER HOST

As of Docker 18.03

host.docker.internal

gateway.docker.internal

NOTES

brew install highlight
highlight -O rtf myfile.php | pbcopy

https://github.com/pmsipilot/docker-compose-viz

RESPONSE

Headers

Request

Response

Body

```
$ curl -v http://localhost:8000/api/v1/talks/1/
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET /api/v1/talks/1/ HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 1100
<
{"description": "You\u2019ve built a fine Python web application
 and now you\u2019re ready to share it with the world. But what\
u2019s the best way to deploy your app in 2017?  This talk will
demonstrate popular techniques for deploying Python web applicat
ions. We\u2019ll start with a simple Flask application and expos
e it to the world five times over as we learn to use different t
ools and services available to the modern Python developer.  Spe
cific topics covered include:  Exposing your local dev environm
ent with ngrok Using a Platform-as-a-Service (PaaS) like Heroku
Going \u201cserverless\u201d with AWS Lambda Configuring your ow
n VM with Google Compute Engine Thinking inside the box using Do
```

RESPONSE HEADERS

Headers

Request

Response

Body

```
$ curl -v http://localhost:8000/api/v1/talks/1/
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET /api/v1/talks/1/ HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 1100
<
{"description": "You\u2019ve built a fine Python web application
 and now you\u2019re ready to share it with the world. But what\
u2019s the best way to deploy your app in 2017?  This talk will
demonstrate popular techniques for deploying Python web applicat
ions. We\u2019ll start with a simple Flask application and expos
e it to the world five times over as we learn to use different t
ools and services available to the modern Python developer.  Spe
cific topics covered include:  Exposing your local dev environm
ent with ngrok Using a Platform-as-a-Service (PaaS) like Heroku
Going \u201cserverless\u201d with AWS Lambda Configuring your ow
n VM with Google Compute Engine Thinking inside the box using Do
```

RESPONSE

Headers

Request

Response

Body

```
$ curl -v http://localhost:8000/api/v1/talks/1/
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET /api/v1/talks/1/ HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 1100
<
{"description": "You\u2019ve built a fine Python web application
 and now you\u2019re ready to share it with the world. But what\
u2019s the best way to deploy your app in 2017?  This talk will
demonstrate popular techniques for deploying Python web applicat
ions. We\u2019ll start with a simple Flask application and expos
e it to the world five times over as we learn to use different t
ools and services available to the modern Python developer.  Spe
cific topics covered include:  Exposing your local dev environm
ent with ngrok Using a Platform-as-a-Service (PaaS) like Heroku
Going \u201cserverless\u201d with AWS Lambda Configuring your ow
n VM with Google Compute Engine Thinking inside the box using Do
```

RESPONSE CODE

Headers

Request

Response

Body

```
$ curl -v http://localhost:8000/api/v1/talks/1/
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET /api/v1/talks/1/ HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 1100
<
{"description": "You\u2019ve built a fine Python web application
 and now you\u2019re ready to share it with the world. But what\
u2019s the best way to deploy your app in 2017?  This talk will
demonstrate popular techniques for deploying Python web applicat
ions. We\u2019ll start with a simple Flask application and expos
e it to the world five times over as we learn to use different t
ools and services available to the modern Python developer.  Spe
cific topics covered include:  Exposing your local dev environm
ent with ngrok Using a Platform-as-a-Service (PaaS) like Heroku
Going \u201cserverless\u201d with AWS Lambda Configuring your ow
n VM with Google Compute Engine Thinking inside the box using Do
```

REQUEST VERB

Headers

Request

Response

Body

```
$ curl -v http://localhost:8000/api/v1/talks/1/
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET /api/v1/talks/1/ HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 1100
<
{"description": "You\u2019ve built a fine Python web application
 and now you\u2019re ready to share it with the world. But what\
u2019s the best way to deploy your app in 2017?  This talk will
demonstrate popular techniques for deploying Python web applicat
ions. We\u2019ll start with a simple Flask application and expos
e it to the world five times over as we learn to use different t
ools and services available to the modern Python developer.  Spe
cific topics covered include:  Exposing your local dev environm
ent with ngrok Using a Platform-as-a-Service (PaaS) like Heroku
Going \u201cserverless\u201d with AWS Lambda Configuring your ow
n VM with Google Compute Engine Thinking inside the box using Do
```

REQUEST VERB

Headers

Request

Response

Body

```
$ curl -v http://localhost:8000/api/v1/talks/1/
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET /api/v1/talks/1/ HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 1100
<
{"description": "You\u2019ve built a fine Python web application
 and now you\u2019re ready to share it with the world. But what\
u2019s the best way to deploy your app in 2017?  This talk will
demonstrate popular techniques for deploying Python web applicat
ions. We\u2019ll start with a simple Flask application and expos
e it to the world five times over as we learn to use different t
ools and services available to the modern Python developer.  Spe
cific topics covered include:  Exposing your local dev environm
ent with ngrok Using a Platform-as-a-Service (PaaS) like Heroku
Going \u201cserverless\u201d with AWS Lambda Configuring your ow
n VM with Google Compute Engine Thinking inside the box using Do
```

REQUEST PATH

Headers

Request

Response

Body

```
$ curl -v http://localhost:8000/api/v1/talks/1/
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET /api/v1/talks/1/ HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 1100
<
{"description": "You\u2019ve built a fine Python web application
 and now you\u2019re ready to share it with the world. But what\
u2019s the best way to deploy your app in 2017?  This talk will
demonstrate popular techniques for deploying Python web applicat
ions. We\u2019ll start with a simple Flask application and expos
e it to the world five times over as we learn to use different t
ools and services available to the modern Python developer.  Spe
cific topics covered include:  Exposing your local dev environm
ent with ngrok Using a Platform-as-a-Service (PaaS) like Heroku
Going \u201cserverless\u201d with AWS Lambda Configuring your ow
n VM with Google Compute Engine Thinking inside the box using Do
```

REQUEST PATH

Headers

Request

Response

Body

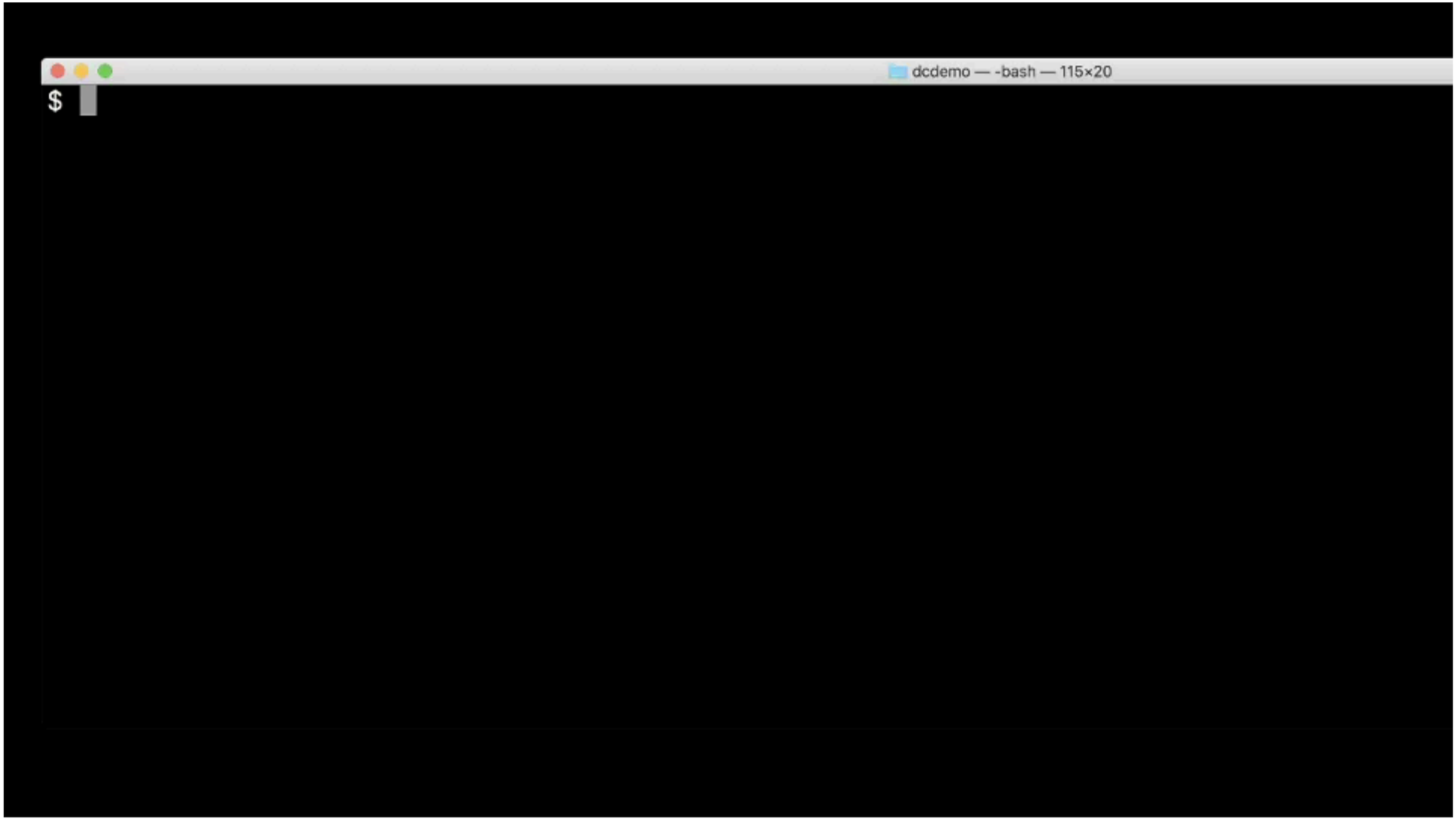
```
$ curl -v http://localhost:8000/api/v1/talks/1/
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET /api/v1/talks/1/ HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 1100
<
{"description": "You\u2019ve built a fine Python web application
 and now you\u2019re ready to share it with the world. But what\
u2019s the best way to deploy your app in 2017?  This talk will
demonstrate popular techniques for deploying Python web applicat
ions. We\u2019ll start with a simple Flask application and expos
e it to the world five times over as we learn to use different t
ools and services available to the modern Python developer.  Spe
cific topics covered include:  Exposing your local dev environm
ent with ngrok Using a Platform-as-a-Service (PaaS) like Heroku
Going \u201cserverless\u201d with AWS Lambda Configuring your ow
n VM with Google Compute Engine Thinking inside the box using Do
```

DEMO (BASIC USAGE)

dcdemo — vim Dockerfile — 126×29

```
ersion: '3.4'
services:

  app:
     build: .

  db:
     image: postgres:10-alpine
~
~
~
~
~
~/Sites/pycon_tut/proj/dcdemo/docker-compose.yml
FROM python:3.6


CMD ["python", "-mhttp.server", "8000"]
~
~
~
~
~
~
~
~
~
Dockerfile
"~/Sites/pycon_tut/proj/dcdemo/docker-compose.yml" 8L, 85C
```

DEMO (BASIC USAGE 2)



DEMO (PORTS)

```
FROM python:3.6

RUN mkdir /htdocs/
WORKDIR /htdocs/
COPY index.html /htdocs/index.html
CMD ["python", "-mhttp.server", "8000"]
~
~
~
~
~
Dockerfile
```

```
version: '3.4'
services:

  app:
     build: .
     ports:
       - 8000:8000

  db:
     image: postgres:10-alpine
~
docker-compose.yml
"docker-compose.yml" 10L, 116C
```

Problem loading page

localhost:8000

Unable to connect

Firefox can't establish a connection to the server at localhost:8000.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox Developer Edition is permitted to access the Web.

Try Again