Dockerize your dev environment

https://github.com/nigel-dunn/dockerize-dev-workshop

Nigel Dunn

ON | CORPS

LAMP app using Slim framework

Simplified view of strategy

- Architecture
- Configuration
- Application

Architecture

Services

Web server	apache
App	PHP
Database	MySQL

Containers

Web server & app options

- Single container for both services
- Separate containers

Images

Use Docker Hub for images (https://hub.docker.com)

- New filters:
  - Docker Certified
  - Verified Publisher
  - Official Images

Decide on appropriate tags for the image (e.g. `php:apache`)

Networking

External:

- Web server (port 80 on the container)

Internal:

- Database (port 3306)

Storage

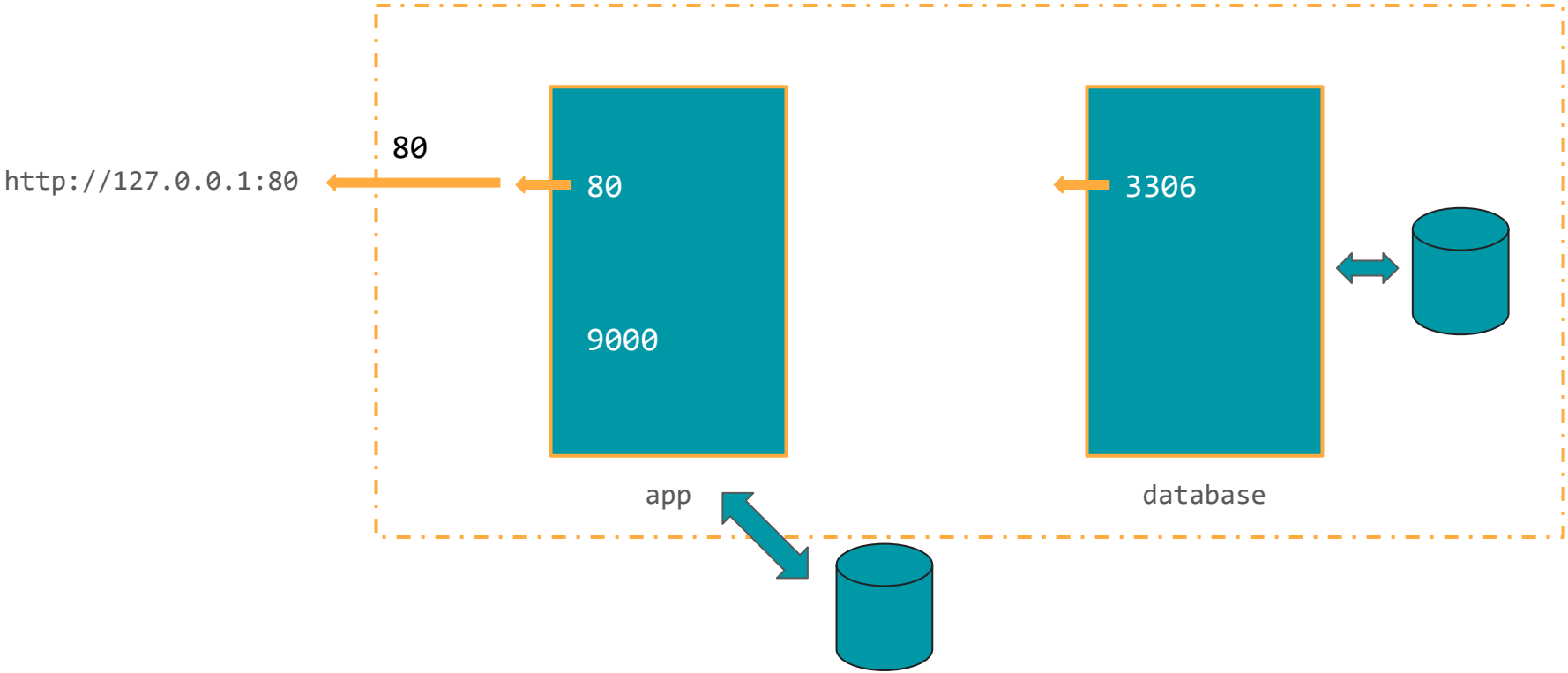
App

Mount local directory into container
Need to know where the apache webroot is inside the container

Database

Named volume but doesn’t need to be accessible on host machine
Need to know where MySQL stores its data

Schematic



docker-compose.yml

```
services:
    app:
        image: php:apache
        ports:
            - 80:80
        volumes:
            - .:/var/www/
    database:
        image: mysql:5
        volumes:
            - data:/var/lib/mysql
volumes:
    data: {}
```

Check progress

Configuration

Missing PHP library

Use `phpinfo()` to see configuration

Add `mysqli` library

Options:

- Use `apt-get install` as would be normal on a server
- Use functionality provided in container (`docker-php-ext-install`)

Missing MySQL set up

Environment variable:

- `MYSQL_ROOT_PASSWORD`
- `MYSQL_DATABASE (creates database)`

Change default character set to utf8mb4

docker-compose.yml

```
services:
      app:
            build:
                  context: .
                  dockerfile: app.Dockerfile
            ports:
                  - 80:80
            volumes:
                  - .:/var/www/
      database:
            build:
                  context: .
                  dockerfile: database.Dockerfile
            volumes:
                  - data:/var/lib/mysql
            environment:
                  MYSQL_ROOT_PASSWORD: password
                  MYSQL_DATABASE: bristol_php_training
volumes:
      data: {}
```

app.Dockerfile

```
FROM php:apache

RUN docker-php-ext-install mysqli
```

database.Dockerfile

```
FROM mysql:5

COPY docker/character_set.cnf /etc/mysql/mysql.conf.d/
```

Check progress

Application

Adding `composer`

Add script to container to run during Dockerfile

https://getcomposer.org/doc/faqs/how-to-install-composer-programmatically.md

Move `composer.phar` to directory in `$PATH`

Add missing wget and zip libraries:

```
RUN apt-get update && apt-get install -y wget zip
```

Adding `mod_rewrite`

In Dockerfile `RUN a2enmod rewrite`

Either

Add `.htaccess` to web directory

Or

`COPY` config file to `/etc/apache2/sites-available/000-default.conf`

Check progress

Where's `/vendor/` gone?

In app.Dockerfile

```
COPY . /var/www/

RUN composer install
```

Log files will show packages being installed at build

But `/var/www/vendor` doesn't exist after `docker-compose up`

Timing of package installation

`COPY . /var/www`

`composer.json`

`RUN composer install`

`composer.json`
`composer.lock`
`vendor/`

`docker-compose up`
(mount volume)

`composer.json`

Timing of package installation

`COPY . /var/www`

`composer.json`

`docker-compose up`
(mount volume)

`composer.json`

`ENTRYPOINT`
`composer install`

```
composer.json
composer.lock
vendor/
```

Check progress

Connecting to the database

Environment variable in `docker-compose.yml`:

```
- HOST
- USER
- PASSWORD
- NAME
```

docker-compose.yml

```
services:
    app:
        build:
            dockerfile: app.Dockerfile
        ports:
            - 80:80
        volumes:
            - .:/var/www/
        environment:
            DB_HOST: database
            DB_USER: root
            DB_PASSWORD: password
            DB_NAME: events
    database:
        …
```

Check progress

Moving on

Check through for secrets

Have you committed anything that gives access to services?

(Credentials for the local database service aren't going to be an issue)

Add `docker-compose.override.yml` to `.gitignore`

Create an example (e.g. `docker-compose.dist.yml`) for others to follow

Recreate your repo if necessary

Making port allocation easier

Potential for port clashes on the host system

Put `ports` in docker-compose.override.yml & remove from docker-compose.yml

Indexed arrays are merged so can’t override if still in docker-compose.yml

Experiment without fear

How will your code look in PHP 7.4?

nginx or apache?

Add caching

Try configuration changes

Be careful with databases – your data is migrated as well as the code

Use different volume names for different versions & database servers

Talk to a friendly ops person

Hardening security if you plan to deploy in the wild

Improving config & shell scripting

Reducing image size

Replacing services/distributions with preferred alternatives