

 **Wiredcraft**

Docker Workshop



Goals

- Get familiar with docker (concepts and commands)
- Give you an idea on what it's good for
- Learn how to use docker in your every day development



Agenda

- Key concepts and usage (Vincent and Juha)
 - Docker setup
 - Key Commands
 - Dockerfile
- Docker for deployment (Oliver)
- Docker for development (Xavier)



Docker is the world's leading software container platform. Developers use Docker to **eliminate “works on my machine” problems** when collaborating on code with co-workers. Operators use Docker to run and manage apps side-by-side in isolated containers to **get better compute density**. Enterprises use Docker to build agile software delivery pipelines to **ship new features faster, more securely** and with confidence for both Linux and Windows Server apps.

- docker.com

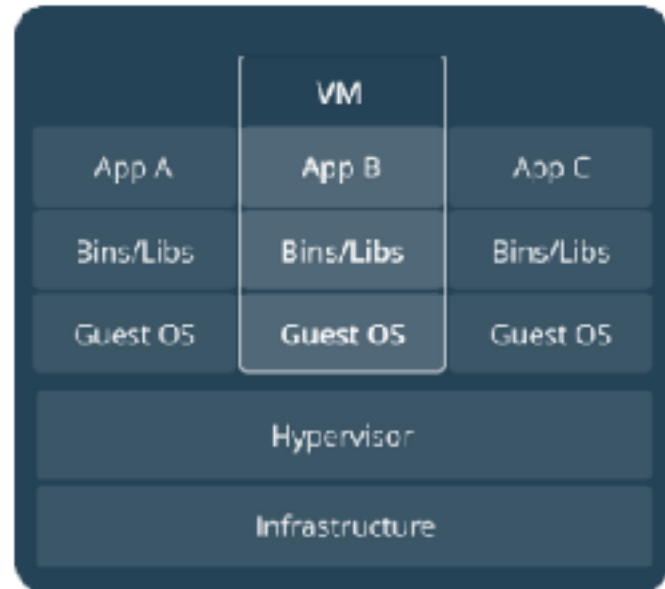
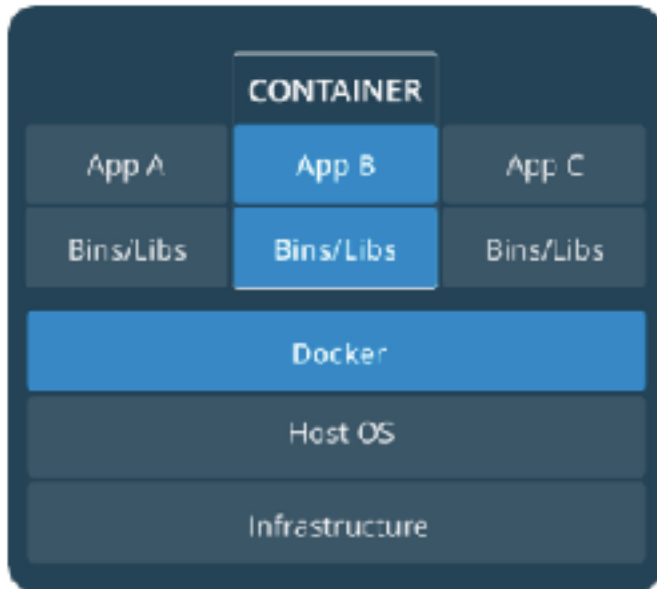


Containers

- Lightweight
- Standard
- Secure



Containers vs. VM



What you should Do

- 1 process / container
- plan ahead (ephemeral vs. persistent, reusable images)
 - use dockerfile caching
 - log



What you should NOT do

- consider as light VM (no init, no ...)
- careful when considering using containers for DB...

More (aging): <http://devo.ps/blog/docker-dos-and-donts/>



Usage

Development

- mock services
 - db
- spawn local architecture

Staging/Prod

- build release as images
 - fast release
 - replicable setup (QA!)
- WARNING; you need your ops team familiar already; many new concepts different from “regular” ops (logs, etc.)



Docker Setup



Who has Docker running on your laptop already?



Docker setup

Setup

Workstation

MacOS

Servers

Linux



Setup MacOS

Go on; <https://docs.docker.com/docker-for-mac/>

Follow the download link and fetch the

DMG from the store

Local copy: <http://139.198.191.29/Docker.dmg>





```

i@vincent:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b6784b78d: Pull complete
Digest: sha256:f3b3623a45268093bb16542c9531885539438e5e6d1ffc05c72261b0d026f1
Status: Downloaded newer image for hello-world:latest

```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the 'hello-world' image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://cloud.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-docker/>



Docker setup

Setup Linux

No cute UI like Windows/Mac

Go on: <https://docs.docker.com/engine/installation/linux/ubuntu/>

or <https://docs.docker.com/engine/installation/linux/centos/>

or ...

Follow the download links from the store and
follow the indication (Yum repo, APT repo, etc.)

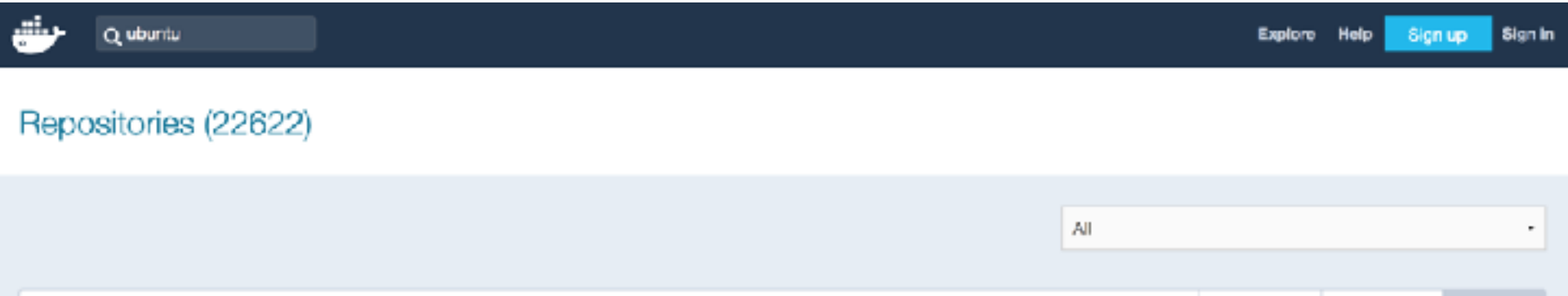


Additional resources

- <https://hub.docker.org> : free docker container registry (for OSS)

Search feature to use / reuse container images.

Auto build of containers



Key Commands



Read The Fine Manual (RFTM!)

Use `--help` on any docker command to get help!

```
docker --help
```

```
docker pull --help
```

etc.

Lots of really good doc at: <https://docs.docker.com/> <https://docs.docker.com/reference/>



Docker concepts

- **image:** used as “template” to spawn containers
- **container:** running code with its own filesystem
- **volume:** external storage that can be mounted on a container at run time
- **port:** listening ports can be exposed on the underlying host
- **link:** containers can be linked to one another allowing direct access to their network ports using the name



Pull images

```
docker pull <image>
```

Fetch container image

```
docker pull alpine
```

<image> can be either of:

- image (alone); assume latest image
- image:tag; e.g. ubuntu:16.04
- repo/image:tag; fetching from another repo than hub.docker.io

```
[vincent:~ vincent$ docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
88286f41530e: Pull complete
Digest: sha256:1072e499f3f655a032e88542330cf75b07
Status: Downloaded newer image for alpine:latest
```

```
[vincent:~ vincent$ docker pull alpine:3.5
3.5: Pulling from library/alpine
019300c8a437: Pull complete
Digest: sha256:af2a5bd2f8de8fc1ecabf1c76611cd
Status: Downloaded newer image for alpine:3.5
vincent:~ vincent$
```



Run container

```
docker run [options] image [cmd]
```

[options]:

-d; detach (run in background)

--rm; remove the container when exited

-ti; get a tty and interactive mode

--name; give a name to the container

[cmd]:

the command you want to run instead
of the default one (e.g. bash)

```
docker run alpine bash
```

Do it:

```
docker run alpine echo "2017.jsconf.cn"
```

Long running: `docker run -d --name long alpine sleep 10000`



List containers

```
docker ps [-a]
```

List all running containers (list all with -a)

```
[vincent:~ vincent$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS
8f265f4ae9de   redis     "docker-entrypoint..." 5 minutes ago  Up 5 minutes  6379/tcp
[vincent:~ vincent$
[vincent:~ vincent$
[vincent:~ vincent$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS
8f265f4ae9de   redis     "docker-entrypoint..." 5 minutes ago  Up 5 minutes  6379/tcp
cd0c3759c675   redis     "docker-entrypoint..." 21 minutes ago Exited (0) 6 minutes ago
9eb609dfe9af   redis     "docker-entrypoint..." 22 minutes ago Exited (0) 22 minutes ago
1781c1172993   hello-world "/hello"                2 hours ago   Exited (0) 2 hours ago
```



Stop / start / delete container

`docker stop container_id/name;` stop a running container

`docker start container_id/name;` start a previously stopped container

`docker rm container_id/name;` removes the container and its files

```
-----  
[vincent:~ vincent]$ docker run -d --name redis redis  
8f265f4ae9deab2074507b8ef6ea2e0a306d8b58cf61b41f171679f487f29d79  
[vincent:~ vincent]$ docker stop redis  
redis  
[vincent:~ vincent]$ docker start redis  
redis
```



Run a command in a running container

(also used to gain shell access)

```
docker exec -ti container_id/name cmd
```

you can exit via `exit` or `ctrl+d`

```
docker exec -ti long sh
```

```
[vincent:~ vincent$ docker exec -ti redis bash  
[root@8f265f4ae9de:/data# ps aux  
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  
redis      1  0.1  0.1  33316  3896 ?        Ssl   03:12   0:00 redis-server *:6379  
root      14  0.5  0.1  20252  3268 pts/0    Ss    03:14   0:00 bash  
root      19  0.0  0.0   17504  2036 pts/0    R+    03:14   0:00 ps aux  
root@8f265f4ae9de: /data#
```



Details of a container / image

```
docker inspect [options] image/container
```

```
docker inspect --format='{{range .NetworkSettings.Networks}} {{.IPAddress}}  
{{end}}' long
```

```
[vincent:~ vincent$ docker inspect alpine
```

```
[  
  {  
    "Id": "sha256:7328f6f8b41890597575cbaadc884e7386ae0acc53b747401ebce5cf0d624560",  
    "RepoTags": [  
      "alpine:latest"  
    ],  
    "RepoDigests": [  
      "alpine@sha256:1072e499f3f655a032e88542330cf75b02e7bdf673278f701d7ba61629ee3ebe"  
    ],  
    "Parent": "",  
  }  
]
```

```
vincent:~ vincent$ docker inspect --format='{{range .NetworkSettings.Networks}} {{.IPAddress}} {{end}}' redis  
172.17.0.2
```



Build container

```
docker build [-f Dockerfile] .
```

Create a new image based on the Dockerfile (more later)

```
vincent:examples vincent$ docker build .  
Sending build context to Docker daemon  2.048kB  
Step 1/2 : FROM redis  
----> 4e482b286430  
Step 2/2 : RUN echo "toto" > /toto  
----> Running in eac5ca3e175f  
----> 63239e9219eb  
Removing intermediate container eac5ca3e175f  
Successfully built 63239e9219eb
```

```
vincent:examples vincent$ docker build .  
Sending build context to Docker daemon  2.048kB  
Step 1/2 : FROM redis  
----> 4e482b286430  
Step 2/2 : RUN echo "toto" > /toto  
----> Using cache  
----> 63239e9219eb  
Successfully built 63239e9219eb
```



Commands we used

```
docker pull alpine
```

```
docker run alpine echo "2017.jsconf.cn"
```

```
docker run -d --name long alpine sleep 10000
```

```
docker ps
```

```
docker ps -a
```

```
docker exec -ti long sh
```

```
docker inspect
```

```
docker inspect --format='{{range .NetworkSettings.Networks}} {{.IPAddress}}{{end}}' long
```



[Time for a break]

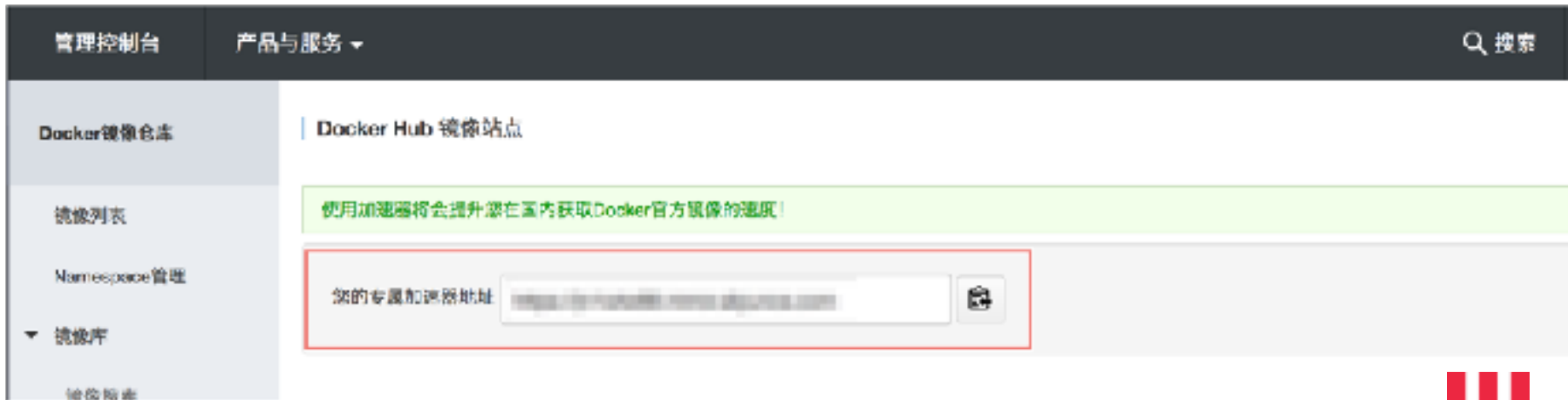


Prepare node.js container



Prepare-I

1. Coffee ☕
2. Docker (<https://www.docker.com/>)
3. Mirror (<http://dev.aliyun.com>)
 - > `docker pull registry.docker-cn.com/library/node:alpine`
 - > `docker pull daocloud.io/library/node:alpine` (daocloud.io/library/node:latest)



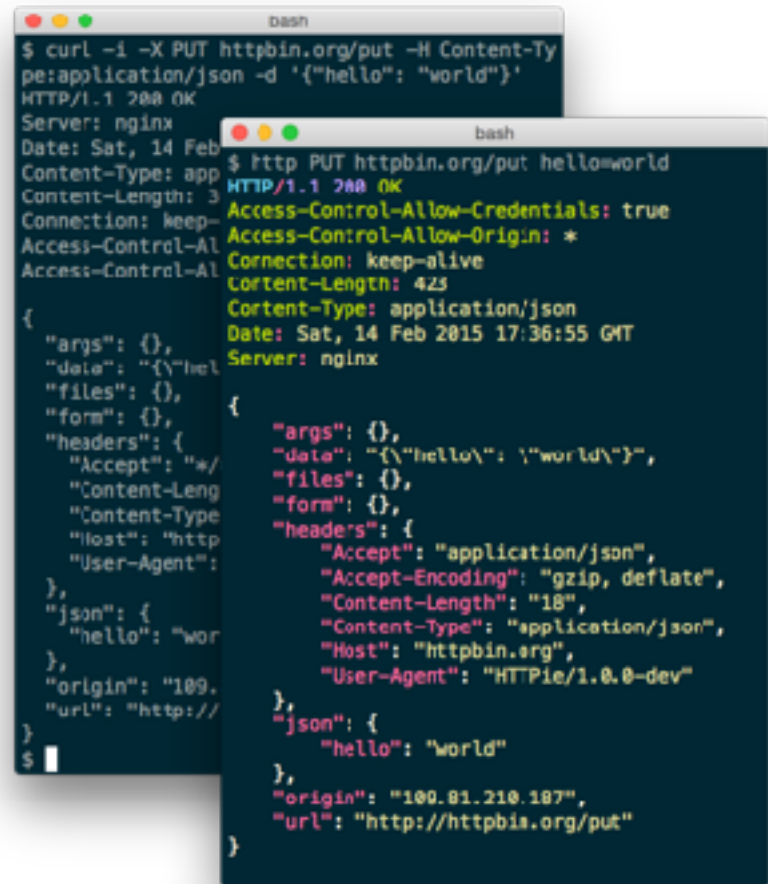
Prepare-II

3. Node Env

4. URL Transfer Tool

- HTTPie <https://httpie.org/>
> brew install httpie
- curl
- Postman <https://www.getpostman.com/>
- Browser(Firefox, Chrome)

5. Favourite Code Editor



The image shows two terminal windows. The top window is titled 'Dash' and shows a curl command being executed: `$ curl -i -X PUT httpbin.org/put -H Content-Type:application/json -d '{"hello": "world"}'`. The output shows the response headers and a JSON body. The bottom window is also titled 'Dash' and shows an httpie command: `$ http PUT httpbin.org/put hello=world`. The output shows the response headers and a JSON body.

```
$ curl -i -X PUT httpbin.org/put -H Content-Type:application/json -d '{"hello": "world"}'
HTTP/1.1 200 OK
Server: nginx
Date: Sat, 14 Feb 2015 17:36:55 GMT
Content-Type: application/json
Content-Length: 42
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Connection: keep-alive

{"args": {}, "data": "{\"hello\": \"world\"}", "files": {}, "form": {}, "headers": {"Accept": "application/json", "Accept-Encoding": "gzip, deflate", "Content-Length": "18", "Content-Type": "application/json", "Host": "httpbin.org", "User-Agent": "HTTPie/1.0.0-dev"}, "json": {"hello": "world"}, "origin": "100.81.210.187", "url": "http://httpbin.org/put"}
```

Clone Project File

```
> git clone https://github.com/Wiredcraft/jsconf-docker-workshop
```

Run Project

```
> npm i && npm start
```



Test Project

- Get people count: GET localhost:3000/people/count
- Add people count: POST localhost:3000/people/count



Dockerfile

```
FROM node:alpine

# Set npm registry
RUN npm config set registry https://registry.npm.taobao.org

# Create app directory and setup working directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install dependencies
COPY package.json /usr/src/app
RUN npm install

# Copy source code
COPY . /usr/src/app

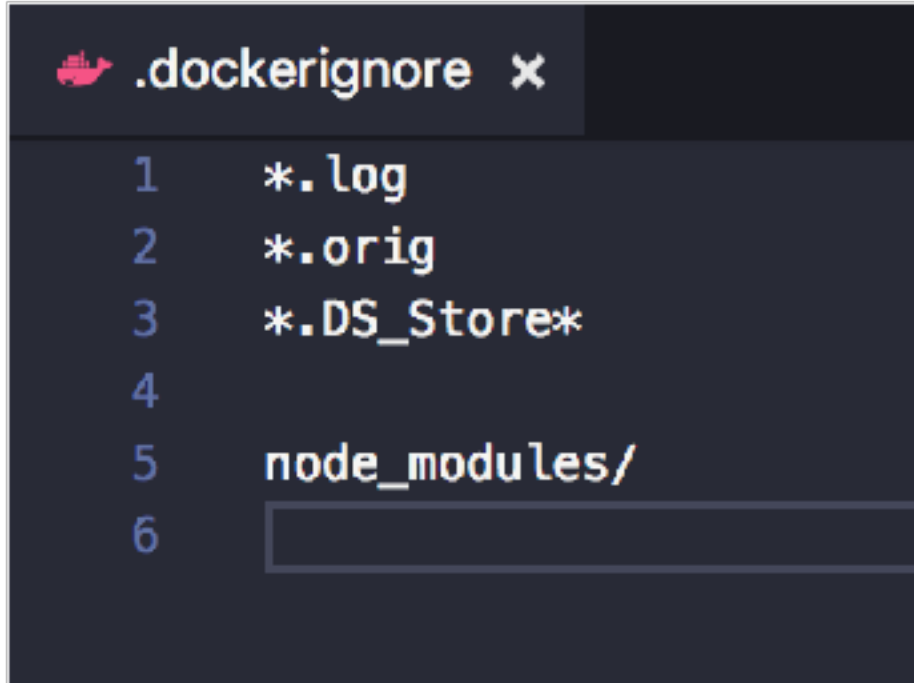
EXPOSE 3000

# Start the server
CMD [ "npm", "start" ]
```

- Image based on node 8.0.0
- Change npm registry
- Create folder for app
- Set working path
- Copy package.json into docker
- Install dependencies(create a layer)
- Copy files into docker
- Expose port
- Start service command



.dockerignore

A screenshot of a code editor window with a dark theme. The title bar at the top shows a red ship icon, the filename ".dockerignore", and a close button (X). The editor contains six lines of text, each preceded by a line number from 1 to 6. The text is as follows:

```
1 *.log
2 *.orig
3 *.DS_Store*
4
5 node_modules/
6
```

The line number 6 is followed by an empty text input field with a light gray border.

- Ignore some file
- Ignore
node_modules
folder

Docker Image

`docker build [OPTIONS] PATH`

`--tag, -t` Name and optionally a tag in the 'name:tag' format

`> docker build -t jsconf-app .`

Check result

`> docker images`



Docker Container

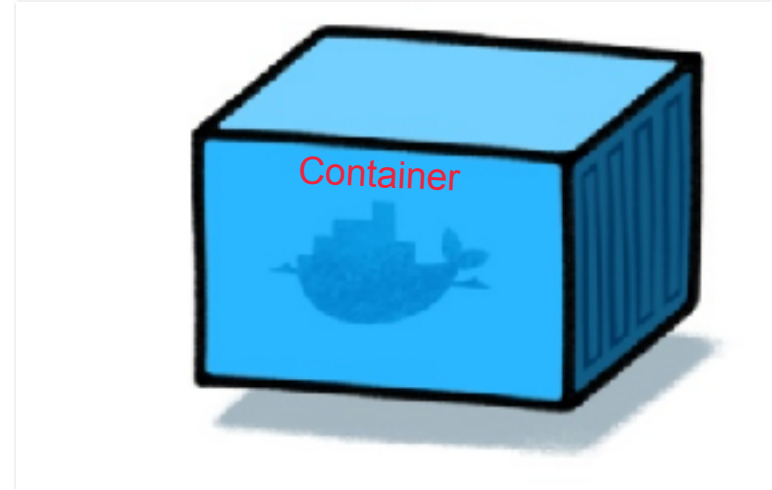
> `docker run --name jsconf-app-container -p 3003:3000 jsconf-app`

> `docker ps`

Check result

> `http localhost:3003/people/count`

> `http post localhost:3003/people/count`



Some Commands

- > `docker exec -it jsconf-app-container bash`
 - Enter bash in container
- > `curl -X GET localhost:3000/people/count`
 - > `curl -X POST localhost:3000/people/count`
- > `docker inspect jsconf-app-container`
 - Restart a container
- > `docker stop jsconf-app-container`
 - Stop a container
- > `docker rm jsconf-app-container`
 - Remove a container
- > `docker restart jsconf-app-container`
 - Inspect a container

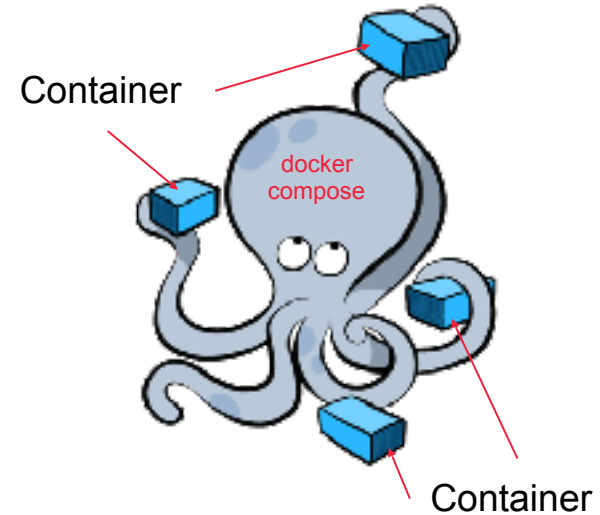


Docker-compose



Docker Compose

- Tool for defining and running multi-container Docker applications
- A file to configure your application's services
- With a single command, create and start all the services



docker-compose.yml

```
version: '3'
services:
  web-app:
    build: .
    container_name: jsconf-app-compose
    ports:
      - "3000:3000"
    depends_on:
      - mongodb
    command: npm run start
    environment:
      - NODE_ENV=development
      - PORT=3000
      - DB_HOST=mongodb
      - DB=test
    networks:
      jsconf-app:
        restart: always
  mongodb:
    image: mongo:latest
    container_name: jsconf-app-mongo
    volumes:
      - ./data/mongodb:/data/db
    ports:
      - 27017:27017
    networks:
      jsconf-app:
        restart: always
networks:
  jsconf-app:
    external:
      name: jsconf-app-network
```

- Give custom name for container
- Mapping ports to local
- Set dependencies for containers
- Overwrite startup command
- Pass in environment value into container
- Specify network between containers
- Restart service after crash
- User docker-compose to set MongoDB work with App
- Persist data outside of container
- Configuration the specific Network for the services to communicate



Create Docker Compose

> `docker network create jsconf-app-network`

> `docker-compose up --build`

> `docker-compose ps`

> `docker-compose stop`

> `docker-compose down`

- Create a customized network for docker compose services
- Build images and starting containers
- List running containers
- Stop running containers
- Stop and remove running containers

Questions?



[Time for a break]



Docker for developpe mocks



Build container for mock service

New requirement!

Now we need to verify the user name before adding the counter.

We will call another service to verify the user from our node app.

The API spec is deadly simple, *GET* <http://{api-host}/profiles/:userName>,

But it's not existing yet (or annoying but not uncommon delay with API provider).

Mock it up with docker!

Check out <https://github.com/Wiredcraft/jsconf-docker-workshop/tree/mock-service>



Mockup profile API with node.js

3 minimum things

app.js, package.json, Dockerfile

```
1 s/app.js > 1 d/docker-compose.yml > 1 d/m/p/Dockerfile > 1 s/com
1 FROM node:6
2
3 # Create app directory
4 RUN mkdir -p /opt/work/profile
5 WORKDIR /opt/work/profile
6
7 # Install dependencies
8 COPY ./src/package.json ./
9 RUN npm install
10 # Volume tricks http://stackoverflow.com/a/32785014/4550665
11 RUN mv ./node_modules ../node_modules
12
13 COPY ./src/ ./
14
15 EXPOSE 3007
16
```

Almost same as the sample mentioned above, but what hell is volume tricks? Let's see the docker-compose file.



Docker-compose.yml for local development

Remove the web-app because now you are running it at local.

Add profile service with:

- Build with ./mocks/profile/Dockefile
- Export port 3007 to localhost
- Mounting src dir to work dir

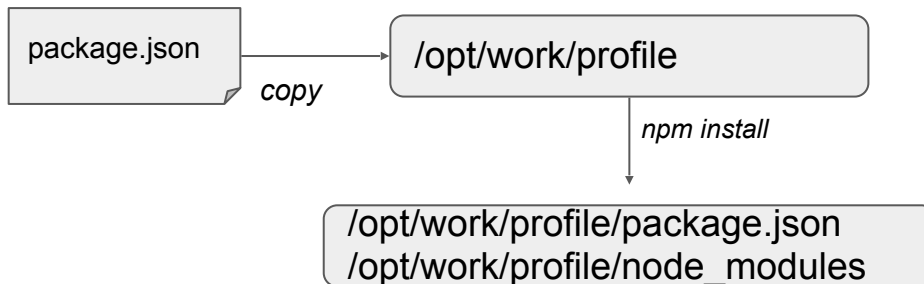
```
" Press ? for help
.. (up a dir)
</jsconf-docker-workshop/
+ docker/
  + mocks/profile/
    + src/
      app.js
      package.json
      Dockerfile
      docker-compose.yml
      start.sh*
+ node_modules/
+ src/
+ test/
  package-lock.json
  package.json
  README.md
  yarn.lock

1 version: '3'
2 services:
3   mongodb:
4     image: 'mongo:latest'
5     container_name: 'jsconf-app-mongo'
6     ports:
7       - '27017:27017'
8   profile:
9     build: ./mocks/profile
10    container_name: 'jsconf-user-profile'
11    ports:
12      - '3007:3007'
13    volumes:
14      - './mocks/profile/src:/opt/work/profile/'
15    working_dir: '/opt/work/profile'
16    command: nodemon app.js
17
```

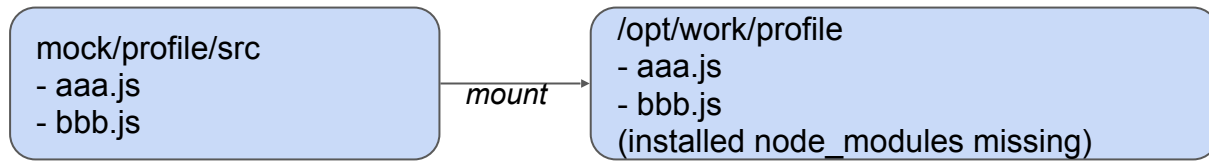


Understand Image build and running

Build phase



Running phase



Another way to build mock API

Still 3 things

Dockerfile, drakov config, api blueprint

Check out <https://github.com/Wiredcraft/jsconf-docker-workshop/tree/mock-service-apib>

Refs:

<https://github.com/Aconex/drakov>

<https://apiblueprint.org/>



Thank you!



Our services

Strategy & Leadership

Need to improve how you manage & grow your team?

UI & UX design

Need to design an app or a device?

Marketing & Growth hacking

Want to understand and grow your audience?

Data science, AI & Big data

Collecting, processing & visualizing data?

Hardware

Want to prototype or produce a hardware product?

Backend & Front-end dev

Building a Web or desktop app?

Mobile dev

Need a mobile, smartwatch or smart-wear app?

Sysadmin & DevOps

Need help with servers?

