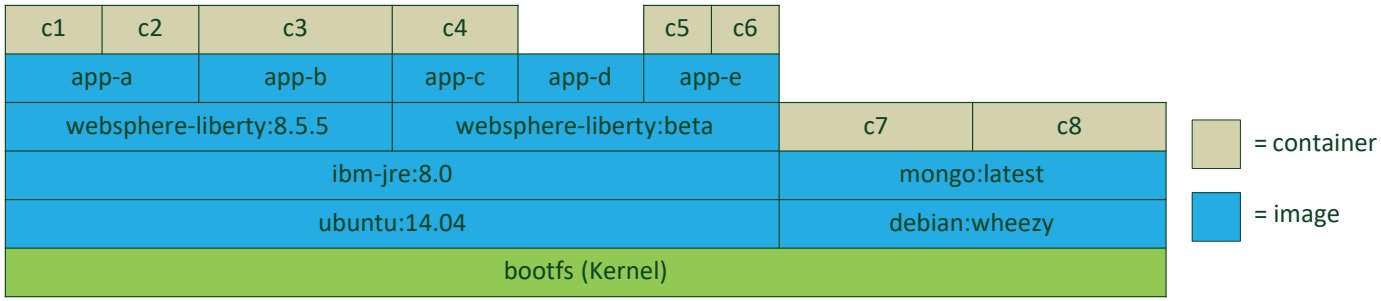
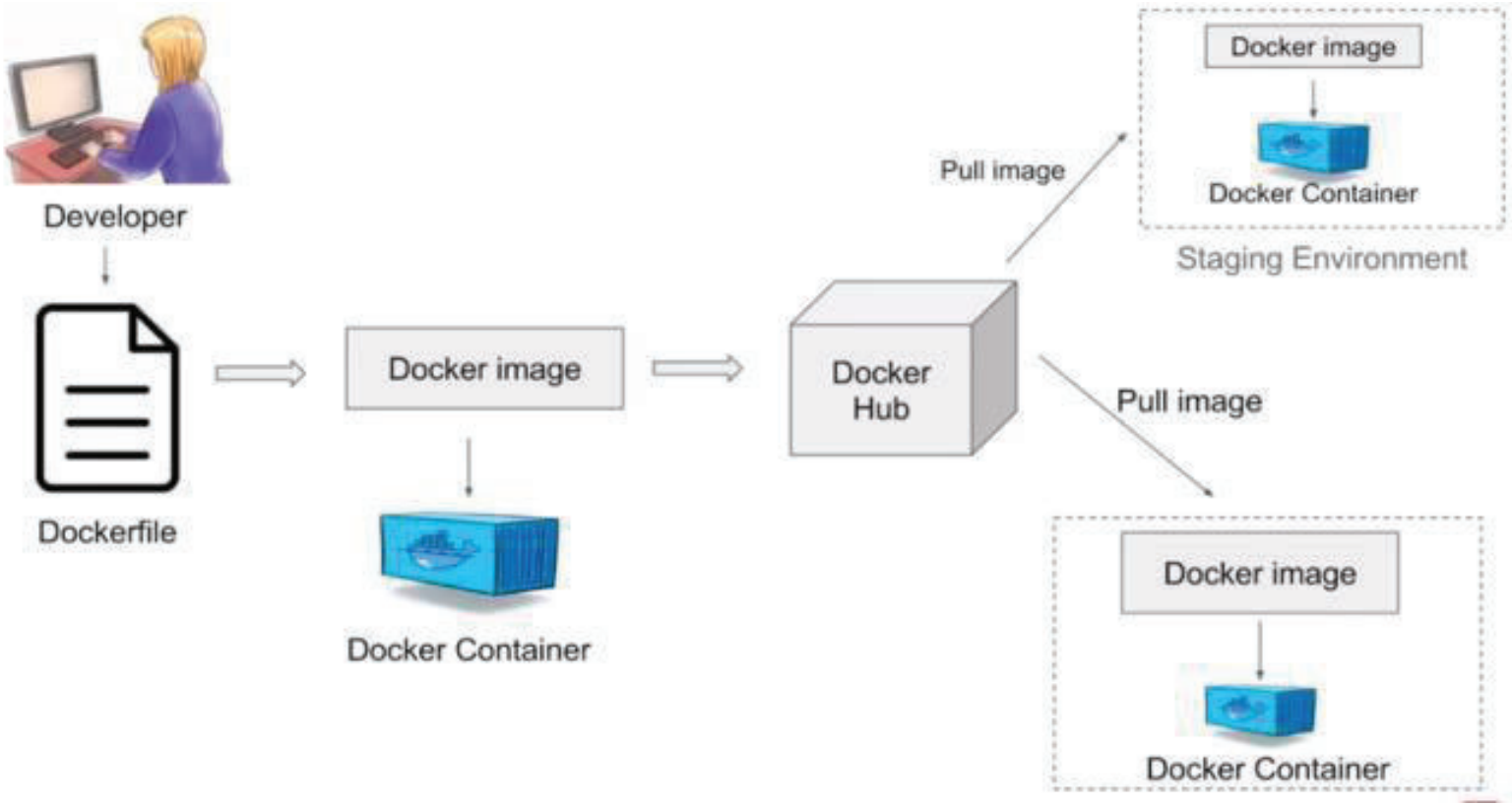
DOCKER IMAGES

- Rappel :
  - container = plusieurs images/couches en lecture seul + couche Copy-On-Write + RUN
- IMAGES
  - Chaque image est en lecture seule
  - Partage des images entre containers sur le host
  - Cache des couches de base en mémoire



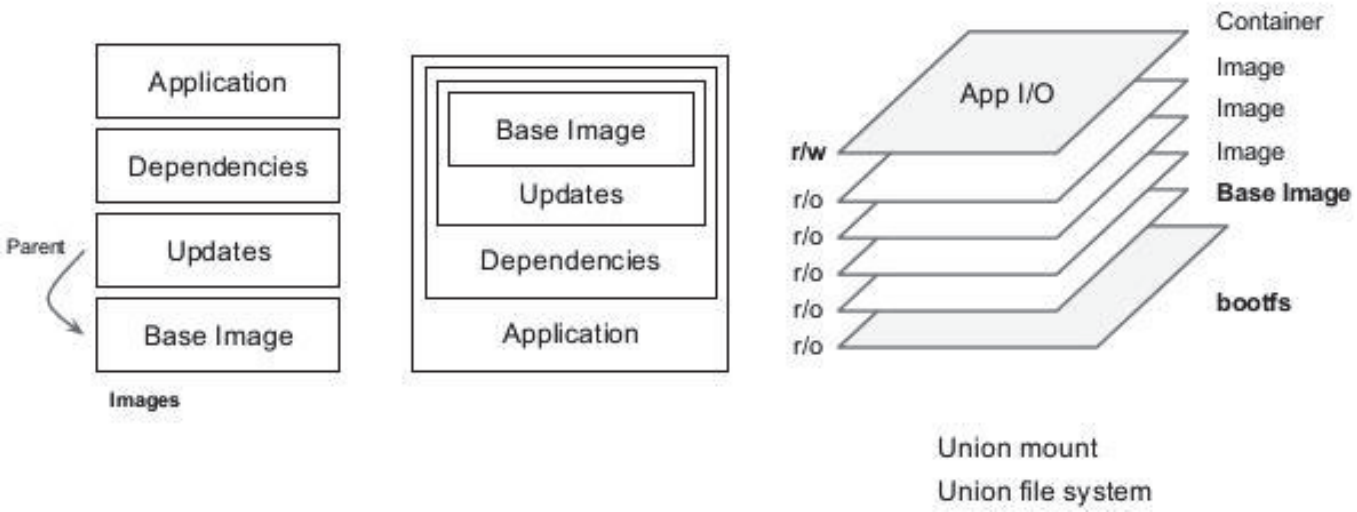
ANDN SERVICES

DOCKER IMAGES - WORKFLOW D'UNE IMAGE



DOCKER IMAGES - WORKFLOW D'UNE IMAGE

- FROM : Specify the base image
- MAINTAINER : Specify the image maintainer
- RUN : Run a command
- ADD : Add a file or directory
- EXPOSE : expose ports to be accessed
- ENV : Create an environment variable
- CMD : What process to run when launching a container from this image.



- **Lecture seule**
- **Réutilisation**
- **Couche (Layer)**

ANDN SERVICES

DOCKERFILE - CRÉER UNE IMAGE

- Un dockerfile est un fichier de configuration qui à pour objectif de créer une image.
- Nous y retrouverons une séquence d'instruction
  - FROM : Sur quelle image on se base
  - ENV : variables d'environnement
  - EXPOSE : exposer le port du conteneur
  - VOLUME : Définition des volumes
  - COPY : cp entre host et conteneur
  - ENTRYPOINT : processus maitre, car l'idée d'un conteneur est d'avoir un seul process qui tourne

DOCKERFILE - CRÉER UNE IMAGE

- L' intérêt d'un dockefile est de relancer une création d'image à tout moment
- Meilleure visibilité sur ce qui est fait
- Partage facile et possibilité de gitter
- Script d'édition de docker file (variables...)
- Ne pas se poser de question lors du docker run du conteneur
- Création images prod // dev - CI // CD (continuous integration/deployment)

DOCKERFILE - CRÉER UNE IMAGE

- FROM ubuntu:latest
- MAINTAINER vincent
- RUN apt-get update \
- && apt-get install -y vim git \
- && apt-get clean \
- && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

- docker build -t nomimage:version .
  - -t nom de l'image
  - Ne pas oublier le "." qui symbolise le dockerfile

DOCKER FILE - CRÉER SON IMAGE

Dockerfile

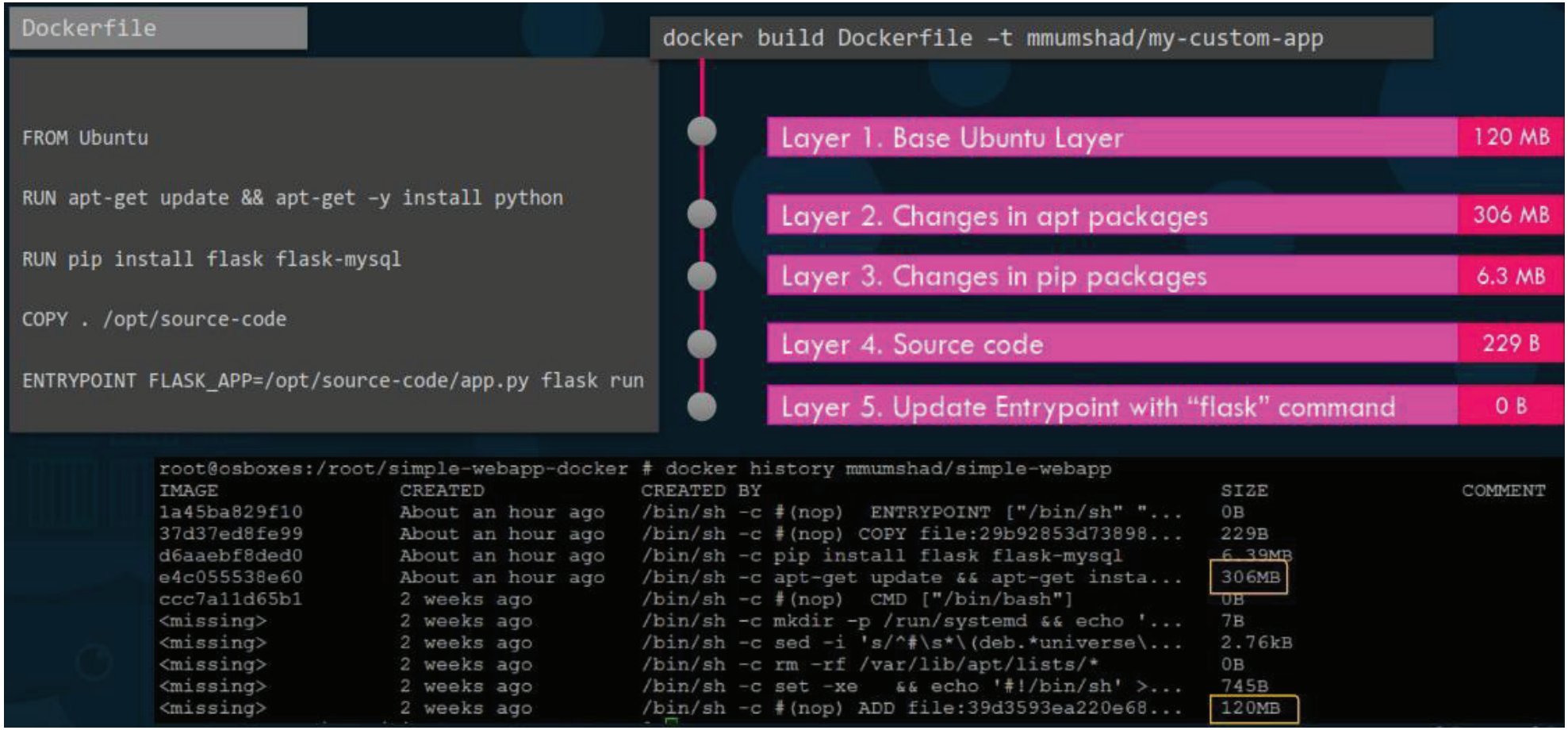
```
FROM Ubuntu

RUN apt-get update
RUN apt-get install python

RUN pip install flask
RUN pip install flask-mysql

COPY . /opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask run
```

1. OS - Ubuntu
2. Update apt repo
3. Install dependencies using apt
4. Install Python dependencies using pip
5. Copy source code to /opt folder
6. Run the web server using “flask” command

```
docker build Dockerfile -t mmumshad/my-custom-app
```

```
docker push mmumshad/my-custom-app
```

COMMENT CRÉER SA PROPRE IMAGE DOCKER DOCKERFILE :ARCHITECTURE EN COUCHES



ANDN SERVICES

COMMENT CRÉER SA PROPRE IMAGE DOCKER
DOCKERBUILD : OUTPUT

```
root@osboxes:/root/simple-webapp-docker # docker build .
Sending build context to Docker daemon  3.072kB
Step 1/5 : FROM ubuntu
 ---> ccc7a11d65b1
Step 2/5 : RUN apt-get update && apt-get install -y python python-setuptools python-dev
 ---> Running in a7840dbfad17
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:4 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [46.3 kB]
Get:5 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:6 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [440 kB]
Step 3/5 : RUN pip install flask flask-mysql
 ---> Running in a4a6c9190ba3
Collecting flask
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting flask-mysql
  Downloading Flask_MySQL-1.4.0-py2.py3-none-any.whl
Removing intermediate container a4a6c9190ba3
Step 4/5 : COPY app.py /opt/
 ---> e7cdab17e782
Removing intermediate container faaaaf63c512
Step 5/5 : ENTRYPOINT FLASK_APP=/opt/app.py flask run --host=0.0.0.0
 ---> Running in d452c574a8bb
 ---> 9f27c36920bc
Removing intermediate container d452c574a8bb
Successfully built 9f27c36920bc
```

ANDN SERVICES

DOCKERFILE - INSTRUCTIONS

1. FROM
    1. Image parente
2. LABEL
    1. Ajout de métadonnées
3. RUN
    1. Commande(s) utilisée(s) pour construire l'image
4. ENV
    1. Variable d'environnement
5. CMD
    1. Exécuter une commande au démarrage du conteneur
6. EXPOSE
    1. Port(s) écouté(s) par le conteneur
7. ARG
    1. Variables passées comme paramètres à la construction de l'image
8. ADD
    1. Ajoute un fichier dans l'image
9. COPY
    1. Ajoute un fichier dans l'image
10. ENTRYPOINT
    1. Exécuter une commande au démarrage du conteneur
11. WORKDIR
    1. Permet de changer le chemin courant (cd tier)
12. VOLUME
    1. Crée un point de montage
13. USER
    1. Nom d'utilisateur ou UID à utiliser
14. ONBUILD
    1. Instructions exécutées lors de la construction d'images enfants

COPY VS ADD

- COPY et ADD sont les deux instructions qui servent à des fins similaires. **Ils vous permettent de copier des fichiers d'un emplacement spécifique dans une image Docker.**

- **COPY** prend un **src** et une **destination** . Il vous permet uniquement de **copier dans un fichier ou un répertoire local de votre hôte** (la machine créant l'image Docker) dans l'image Docker elle-même.

- **ADD** vous permet de le faire aussi, **mais il prend également en charge 2 autres sources**. Tout d'abord, vous **pouvez utiliser une URL** au lieu d'un fichier/répertoire local. Deuxièmement, vous pouvez **extraire un fichier tar de la source directement dans la destination.**

- Dans la plupart des cas, si vous utilisez une URL, vous **téléchargez un fichier zip et utilisez ensuite la RUN commande pour l'extraire.**

- Cependant, vous pouvez tout aussi bien **utiliser RUN avec curl au lieu d' ADD** afin de tout enchaîner **en une seule RUN** pour créer une image Docker plus petite.

- Un cas d'utilisation valide pour **ADD est lorsque vous souhaitez extraire un fichier tar local dans un répertoire spécifique de votre image** Docker. C'est exactement ce que fait l'image Alpine ADD rootfs.tar.gz / .

- Si vous copiez des fichiers locaux sur votre image Docker, utilisez toujours COPY car c'est plus explicite.

ANDN SERVICES

DOCKERFILE : ENTRYPOINT VS CMD

- Entrypoint est le processus principal sur lequel va tourner le conteneur qui va permettre de lancer l'image.
- CMD est la commande par défaut (arguments/paramètres par défaut)
- CMD seule remplace l'entrypoint

```
FROM alpine
MAINTAINER Vincent
CMD ["ping", "--help"]
```

- Mais ne prend pas les arguments passés en CLI.
- Si docker run --rm demo google.fr
  - Vous aurez un message d'erreur car le CMD ne fera la distinction entre paramètre et process.

```
FROM alpine
MAINTAINER Vincent
ENTRYPOINT ["ping", "--help"]
```

- docker build -t demo -f Dockerfile .
- docker run --rm demo
- La commande passe puis en passant docker run --rm demo google.fr, vous ne chargerez pas le container.

ANDN SERVICES

DOCKERFILE : ENTRYPOINT VS CMD

FROM alpine

MAINTAINER Vincent

CMD ["--help"]

ENTRYPOINT ["ping"]

- docker build -t demo -f Dockerfile .
- docker run --rm demo google.fr
- La commande passe puis en passant docker run --rm demo google.fr.
- On précise le paramètre par défaut dans le CMD et le process dans le entrypoint !

TP : CRÉEZ VOTRE PREMIÈRE IMAGE

- Il s'agit ici de créer une image docker afin de conteneuriser une application web statique,
- Télécharger les fichiers de l'application à l'aide de git clone « https://github.com/sadofrazer/static-website-example.git »
- conteneuriser cette application à l'aide de l'image de base nginx.

- Il s'agit ici de créer une image docker afin de conteneuriser une application web statique,
- conteneuriser cette application sans télécharger les fichiers au préalable en local à l'aide de l'image de base ubuntu.
- Fichiers de l'application se trouvant dans le repo : https://github.com/daviddias/static-webpage-example.git