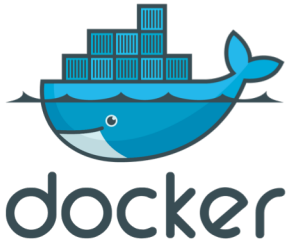
Bonnes Pratiques

Du développement à la production

Image du site de Docker



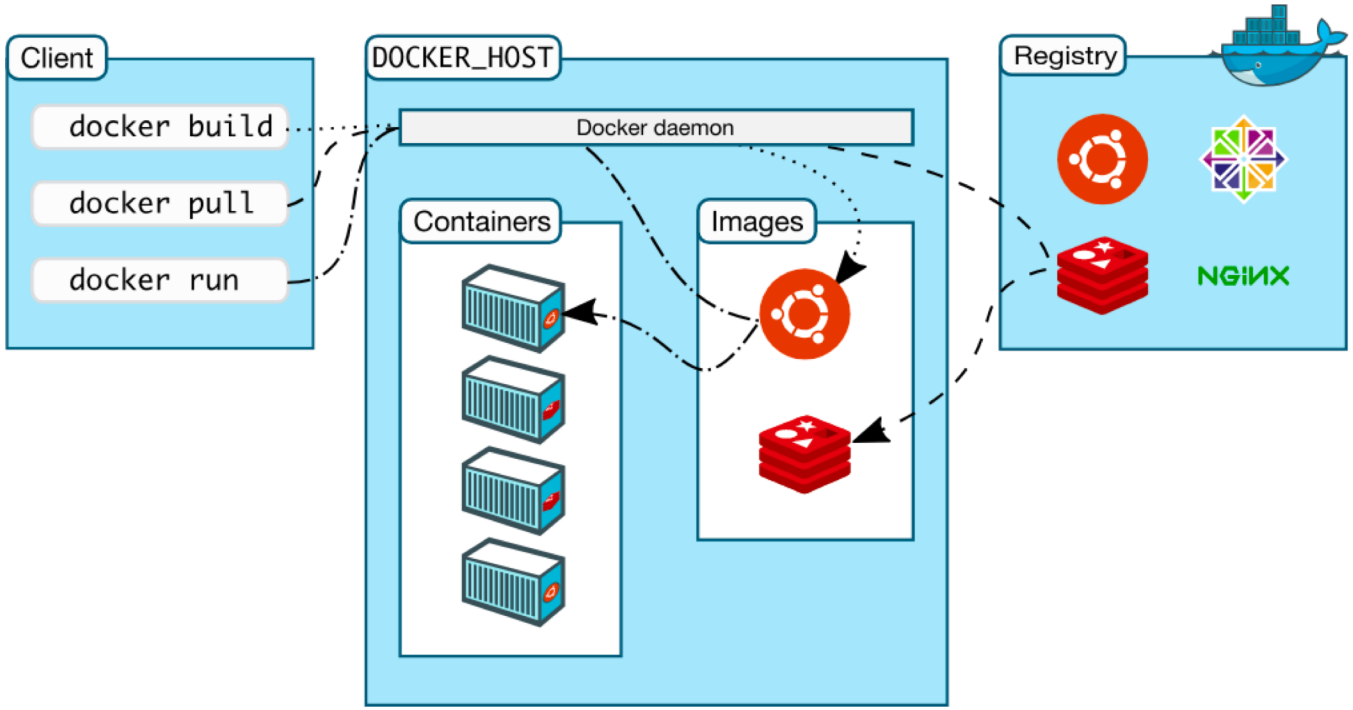
- Reproductibilité
- Isolation
- Sécurité
- Installations rapides
- CI / CD

Ce n’est **pas** une « VM légère »

- `pivot_root()`
- `clone()` + `CLONE_NEWCGROUP / NEWPIC / NEWNET`
- control groups
- capabilities

Image du site de Docker

Vue d'ensemble



Installer Docker

- Sérieusement ?
- Démerdez vous…
- Attention aux permissions du *daemon* !

Cycle de vie

- Récupérer des images depuis des registres
- **ET/OU** construire des images dérivées

- A partir des images, créer des *containers*

- Arrêter, démarrer ou supprimer des *containers*
- Interagir avec les *containers* en cours d'exécution

- Supprimer les images

Session classique

```
$ docker pull nginx
$ docker images
$ docker run --name nginx-container -d -p 127.0.0.1:8080:80 nginx
$ docker ps -a
$ docker exec -it nginx-container bash
$ docker stop nginx-container
$ docker rm nginx-container
$ docker rmi nginx
```

Anatomie d’une commande

```
$ docker run \
      --name nginx-container \
      -d \
      -p 127.0.0.1:8080:80 \
      nginx
```

Dockerfile

- Dériver d'une autre image
- Surement de `_/alpine`
- Récupérer les ressources du *runtime*
- Le piège du `EXPOSE`
- Lancer l'application

Les volumes

```
$ docker run \
      --name squid-running \
      -d \
      --publish 3128:3128 \
      --volume $(pwd)/squid.conf:/etc/squid/squid.conf:ro \
      sameersbn/squid
```

Le problème des réseaux

- Application A est dans son *container*
- Application B est dans son *container*
- Application A écoute sur `0.0.0.0:1111`
- Application B écoute sur `0.0.0.0:2222`
- **B doit interroger A pour son fonctionnement**

Solutions naïves

- Passer par l'hôte
- `172.17.0.10` / `host.docker.internal`
- C'est juste mauvais

- `docker create --network`
- C'est fastidieux

`docker-compose`

- Orchestration
- *Networking* automatique
- Accès aux machines par *hostname*
- Abstraction des *IPs*

Implication dans le développement

- Utilisation de l'environnement
- `docker-compose.yml` != configuration de l'application
- Le piège des microservices
- Tout n'est pas sujet à API

En vrac…

- `docker run --restart=always`
- `docker run --rm`
- `docker inspect | grep`

- `.dockerignore`
- `Les alias !`

Proxy inverse

```
server {
  listen 443 ssl;
  server_name blog.geographer.fr;

  ssl_certificate /etc/letsencrypt/live/geographer.fr-0001/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/geographer.fr-0001/privkey.pem;

  location / {
    proxy_pass              http://localhost:8888/;
    proxy_set_header        Host $host;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_connect_timeout   10;
    proxy_send_timeout      10;
  }
}
```

Image du site de Kubernetes

Pour les vaillants...

Image du site de GitLab

- GitLab ⊂ GitHub
- *Workers* gratuits
- *Registry*
- Déploiement, *monitoring*
- Excellente intégration

Parmi les avantages

- Hygiène de code
- Casse moins souvent
- Développeur confiant
- Manager satisfait
- Commerciaux efficaces

`.gitlab-ci.yml`

- *Pipeline*, par étape
- *Caching*
- Parallélisme
- Conditionner le *merge*
- Il ne faut pas bloquer le *push*

Image du site de Human Coders

```
# master / develop / feature

$ git checkout -b feature
$ vim thing.go
$ git diff
$ git add thing.go
$ git commit
$ git push (--set-upstream-to origin/feature)

# GitLab ! On squash, on delete la branche...

$ git checkout develop
$ git pull
```

Synchronisation

- `develop` a évolué : `merge` (`rebase` ?)
- `git pull origin autre-feature`

- Et on reste propre !
- `git branch --merged`
- `git branch -a | grep …`

Des questions ?

- Au boulot !

- `docker pull tekspice`
- `docker run ...`

- GitLab
- CI (tests) / CD (Docker)

Merci !

A demain...