Docker 实践和原理

张晋涛

2019.01.12

Intros

- 张晋涛
- 专注于 Containers/Docker/Kubernetes 等技术
- 《Kubernetes 从上手到实践》掘金小册作者
- https://github.com/tao12345666333
- https://www.zhihu.com/people/TaoBeier

Docker 概览

为何使用容器

- 多种技术栈:
  - Node、Java、Python、Go
  - 框架
  - 各种私有库
- 不同的目标:
  - 生产、测试、灰度
  - 开发、运维、测试
  - 物理机、云服务器、混合

Docker 是什么

- Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries.

Docker 的历史

- 2008, LXC
- March 20, 2013, PyCon, dotCloud released the first version of Docker
- The same year, dotCloud changes name to Docker
- March, 2014, New default driver: libcontainer (Docker 0.9)
- June, 2014, Docker 1.0
- Mesos, Kubernetes etc
- Standardization around the OCI
- Docker CE 17.03 (after 1.13.1 at Feb, 2017))

第一个容器

Hello World

just run the command:

```
➜  ~ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

常用的容器

```
➜  ~ docker run -it --rm node:10.13 bash
root@d6d54e24337d:/# node --version
v10.13.0
root@d6d54e24337d:/# npm --version
6.4.1
```

- `--rm` tells Docker that remove the container when it exits automatically.
- `-i` tells Docker keep stdin open and connect us to the container's stdin.
- `-t` tells Docker than we want a pseudo-TTY.

使用容器

```
root@d6d54e24337d:/# node -p 'process.versions'
{ http_parser: '2.8.0',
  node: '10.13.0',
  v8: '6.8.275.32-node.36',
  uv: '1.23.2',
  zlib: '1.2.11',
  ares: '1.14.0',
  modules: '64',
  nghttp2: '1.34.0',
  napi: '3',
  openssl: '1.1.0i',
  icu: '62.1',
  unicode: '11.0',
  cldr: '33.1',
  tz: '2018e' }
```

容器去了哪里

如果没有添加 `--rm` 参数的话

- 它仍在磁盘，只不过占用 CPU/内存资源被释放掉了.
- 容器当前是 `stopped` 状态.
- 我们可以再次回到容器中去.

Docker 实践

What is an Image?

- Image is files.
- These files form the root filesystem of our container.

Image contents

This is a `debian:9`'s image.

```
(Tao) ➜  debian tree
.
├── 0783bfd2d1fc35d2dd7c457d9c7195ef2512acabe6ffa78fd035cece65292b0e
│   ├── json
│   ├── layer.tar
│   └── VERSION
├── 3bbb526d26083e7a65a7a112ed72e1ec58e81384412f2d3fcdbbd87d49fd588d.json
├── manifest.json
└── repositories

1 directory, 6 files
```

The read-write layer

- Base on `debian:9` docker image.
- Image is read only filesystem.
- Images can share layers to optimize disk usage and more.
- `docker run` start a container from a given image.

Set of commands

Pull

```
(Tao) ➜  ~ docker pull debian:9
9: Pulling from library/debian
Digest: sha256:07fe888a6090482fc6e930c1282d1edf67998a39a09a0b339242fbfa2b602fff
Status: Image is up to date for debian:9
```

Run

```
(Tao) ➜  ~ docker run --rm -it --name debian debian:9
root@17bae8832e6f:/#
```

Build

```
(Tao) ➜  ~ docker commit debian local/debian:9
sha256:86fb2e51de2c8501c51d10f4839154464cba66afe69490da0723a0e0fecb2a35
```

Images namespaces

- Official images

  e.g. `debian`, `centos`

- User images

  e.g. `taobeier/vim`, `taobeier/docker`

- Self-hosted images

  e.g. `xxx.xxx.com/infraop/openjdk`

Dockerfile 实践

Dockerfile overview

- `Dockerfile` 是一个文本文件.
- 包含了镜像结构.
- 可以使用 `docker build` 根据 `Dockerfile` 构建出镜像.

Demo

```
const Koa = require('koa');
const app = module.exports = new Koa();

app.use(async function(ctx) {
  ctx.body = 'Hello World';
});

if (!module.parent) app.listen(3000);
```

https://github.com/tao12345666333/koa2-docker

The simplest usage

```
FROM node:10.13

ARG NODE_ENV
ENV NODE_ENV ${NODE_ENV:-production}

WORKDIR /app

COPY . /app

RUN npm install

EXPOSE 3000
ENTRYPOINT ["node", "app.js"]
```

Build it

```
(Tao) ➜  docker build --build-arg NODE_ENV=test -t local/koa .
```

- `-t` indicates the tag to apply to the image.
- `.` indicates the build location of the build context.

What happens

```
➜  docker build --build-arg NODE_ENV=test -t local/koa .
Sending build context to Docker daemon  116.2kB
Step 1/8 : FROM node:10.13
 ---> 5432ebbc244b
 ...
Step 6/8 : RUN npm install
 ---> Running in 12217c89f75c
added 81 packages from 464 contributors and audited 112 pack
found 0 vulnerabilities
Removing intermediate container 12217c89f75c
 ---> ba7ca11c26c4
 ...
 ---> 997d579d9e08
Step 8/8 : ENTRYPOINT ["node", "app.js"]
 ---> Running in b869e8b1fb51
Removing intermediate container b869e8b1fb51
 ---> 0c157ae86ca1
Successfully built 0c157ae86ca1
Successfully tagged local/koa:latest
```

- More details: http://dwz.cn/7JzMttGN

缓存构建系统

```
➜  docker build --build-arg NODE_ENV=test -t local/koa .
Sending build context to Docker daemon  116.2kB
Step 1/8 : FROM node:10.13
 ---> 5432ebbc244b
Step 2/8 : ARG NODE_ENV
 ---> Using cache
 ---> 64f91f5ab4dc
Step 3/8 : ENV NODE_ENV ${NODE_ENV:-production}
 ---> Using cache
 ---> af12da7ca49f
Step 4/8 : WORKDIR /app
 ---> Using cache
 ---> fd808405b18e
Step 5/8 : COPY . /app
 ---> Using cache
 ---> 3c1a481de8ce
Step 6/8 : RUN npm install
 ---> Using cache
 ---> ba7ca11c26c4
Step 7/8 : EXPOSE 3000
```

镜像历史

```
(MoeLove) →  docker image history local/koa:latest
IMAGE          CREATED  CREATED BY                                      SIZE
0c157ae86ca1 x  /bin/sh -c #(nop)  ENTRYPOINT ["node" "app.j…   0B
997d579d9e08 x  /bin/sh -c #(nop)  EXPOSE 3000                  0B
ba7ca11c26c4 x  /bin/sh -c npm install                          4.97MB
3c1a481de8ce x  /bin/sh -c #(nop) COPY dir:fe32aa160732b8fc9…   57.1kB
fd808405b18e x  /bin/sh -c #(nop) WORKDIR /app                  0B
af12da7ca49f x  /bin/sh -c #(nop)  ENV NODE_ENV=test            0B
64f91f5ab4dc x  /bin/sh -c #(nop)  ARG NODE_ENV                 0B
5432ebbc244b x  /bin/sh -c #(nop)  CMD ["node"]                 0B
...
<missing>    x  /bin/sh -c #(nop) ADD file:28906382f13932b84…   127MB
```

多阶段构建

```
FROM node:10.13 as builder

WORKDIR /app
COPY . /app

RUN yarn install \
        && yarn build


FROM nginx:1.15

COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=builder /app/dist /usr/share/nginx/html/

EXPOSE 80
```

https://github.com/tao12345666333/saythx/blob/master/fe/Dockerfile

GitLab CI

https://gitlab.com/taobeier/koa2-docker/pipelines/43029655

```
image: taobeier/docker
services:
  - docker:dind
variables:
  DOCKER_DRIVER: overlay2
  IMAGE_NAME: $CI_REGISTRY/$CI_PROJECT_PATH
stages:
  - test
  - build
test:
  image: node:10.13
  stage: test
  script:
    - npm install
    - npm test
build:
  stage: build
  script:
```

Docker Architecture

Docker Engine

Docker Engine

- Client - docker(CLI)

Docker Engine

- Client - docker(CLI)
- REST API - Over UNIX sockets or a network interface

Docker Engine

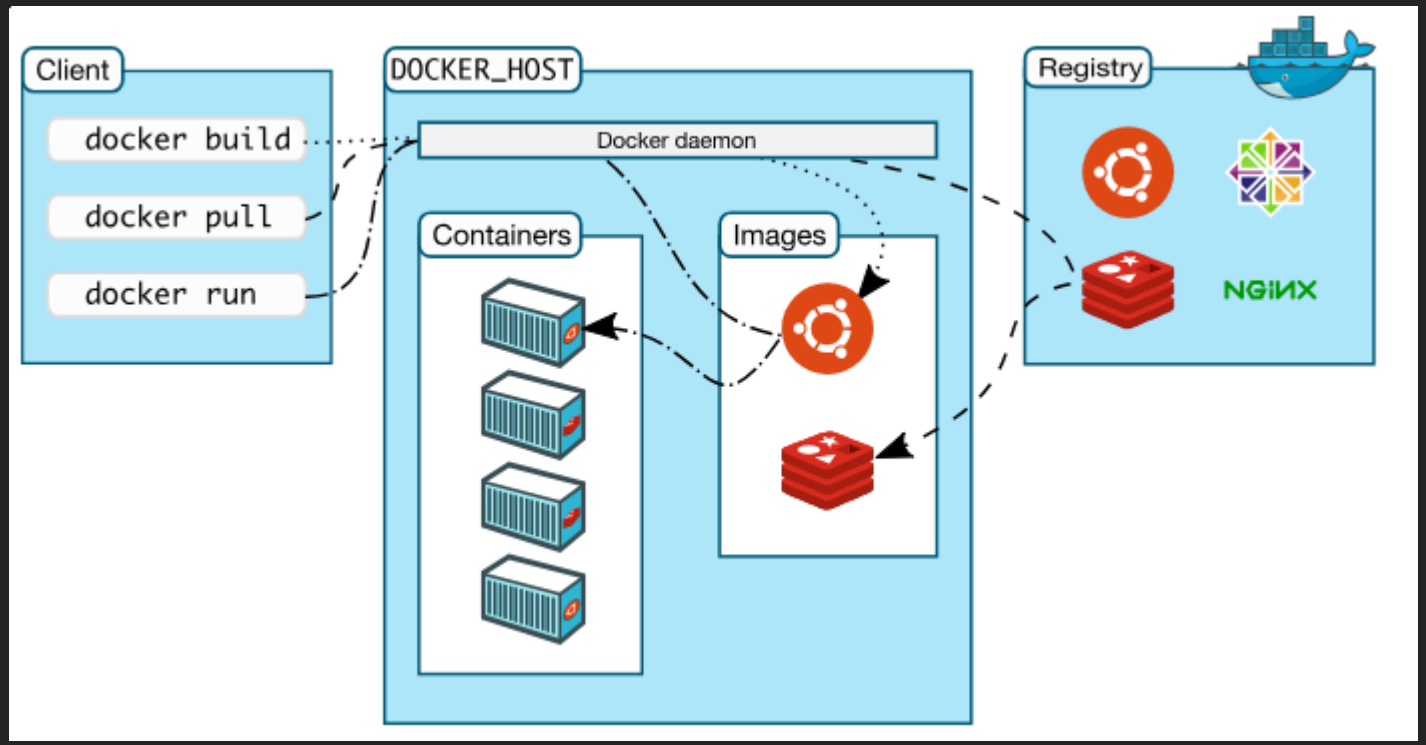
- Client - docker(CLI)
- REST API - Over UNIX sockets or a network interface
- Server - dockerd

docker version

```
(Tao) ➜  ~ docker version
Client:
 Version:           18.09.0
 API version:       1.39
 Go version:        go1.10.4
 Git commit:        4d60db4
 Built:             Wed Nov  7 00:48:47 2018
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Version:           dev
 API version:       1.39 (minimum version 1.12)
 Go version:        go1.10.3
 Git commit:        e8cc5a0b3
 Built:             Tue Sep 11 02:09:53 2018
 OS/Arch:           linux/amd64
 Experimental:      false
```

Docker Engine architecture

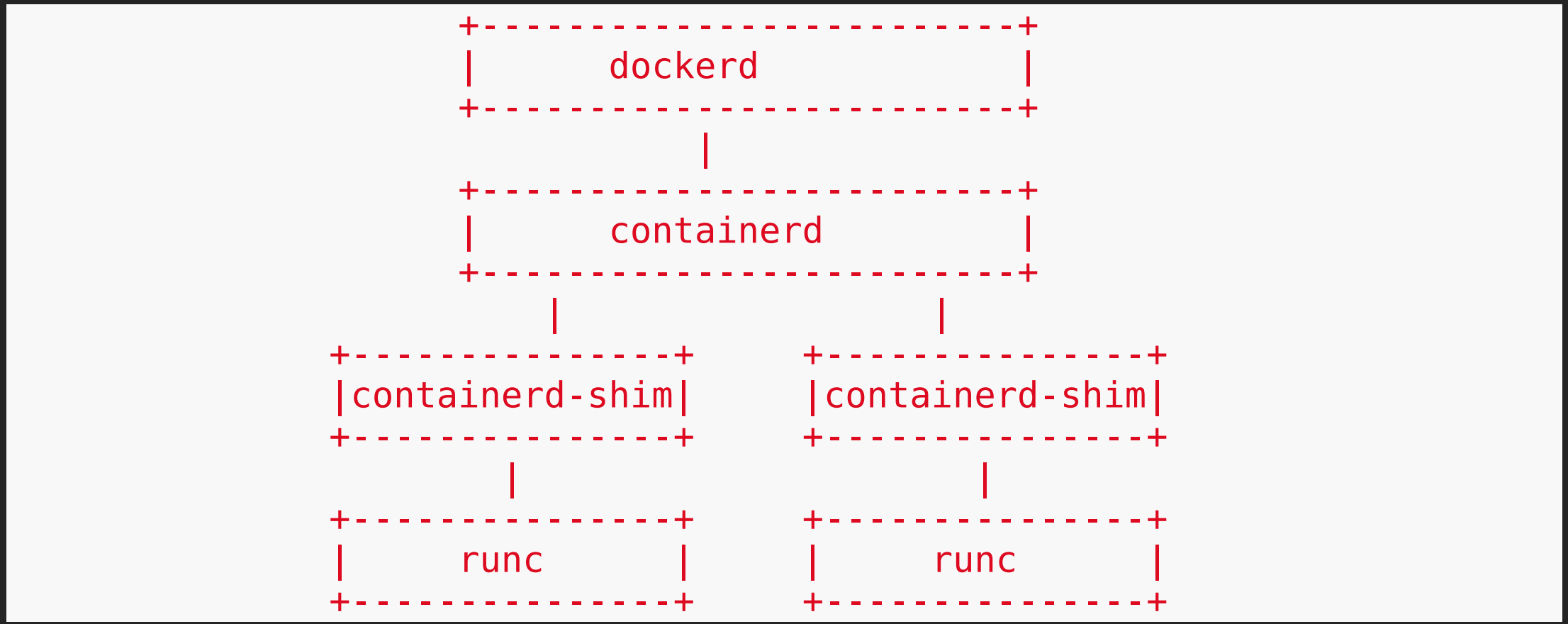


All tools

List all tools about docker.

```
(Tao) ➜  ~ ls /usr/bin |grep docker
docker  # docker cli
docker-containerd
docker-containerd-ctr  # containerd cli
docker-containerd-shim
dockerd
docker-init  # init injects
docker-proxy
docker-runc
```

Docker Engine internal



Q&A

掘金、知乎：TaoBeier

Q&A

Thanks!