GlobalLogic®

DevOps Training: Docker, Kuber

Volodymyr Volkov

July-2020

My name is Vova, and I`m a kubernetes admin ...

Practice Requirements

- AWS EC2
  - Frankfurt
  - Ubuntu Server 18.04 LTS (HVM)
  - t2.micro 1 instance
  - 1 Public IP
  - Security:
    - ssh from your public IP
    - 8080 http (tcp) from your public IP
  - Install Docker: snap install docker
  - *JFYI: ssh user: ubuntu, to become rootuse: “sudo su -”*

Lection 1: Container - What Are You?

Prepare: AWS, Frankfurt, Ubuntu 18.04 LTS, Docker: snap install docker; Public: ssh, 8080(tcp)

OS Level User Process Isolation

Became meaningful on multitasking introduction.

Initial low-level understanding:

Process isolation is a set of different hardware and software technologies[1] designed to protect each process from other processes on the operating system.

...

Security is easier to enforce by disallowing inter-process memory access, in contrast with less secure architectures such as DOS in which any process can write to any memory in any other process.

WIKI

Prepare: AWS, Frankfurt, Ubuntu 18.04 LTS, Docker: snap install docker; Public: ssh, 8080(tcp)

Unix/Linux Resource Isolation Tools History

1979 Unix
1982 BSD
(later on Linux - now not yet invented:)

Chroot

Substituting root point in file tree for process

2000 FreeBSD

Jails

clear-cut separation between its services in "jails" own IP for each jail (processes in it)

2001 Linux

vServer

Jail + resource partitioning for file systems, network addresses, memory

2004 Solaris

Containers

resource controls and boundary separation provided by zones snapshots, cloning from ZFS

2005 Linux

VPS buttle with KVM, Xen, VMWare

OpenVZ

Operating system-level virtualization: isolation, resource management (IPCs, devices) and checkpointing

2006 Linux

CGroup →
(Control Group by Google)

limiting, accounting and isolating resource usage (CPU, memory, disk I/O, network) of a collection of processes

2008 Linux

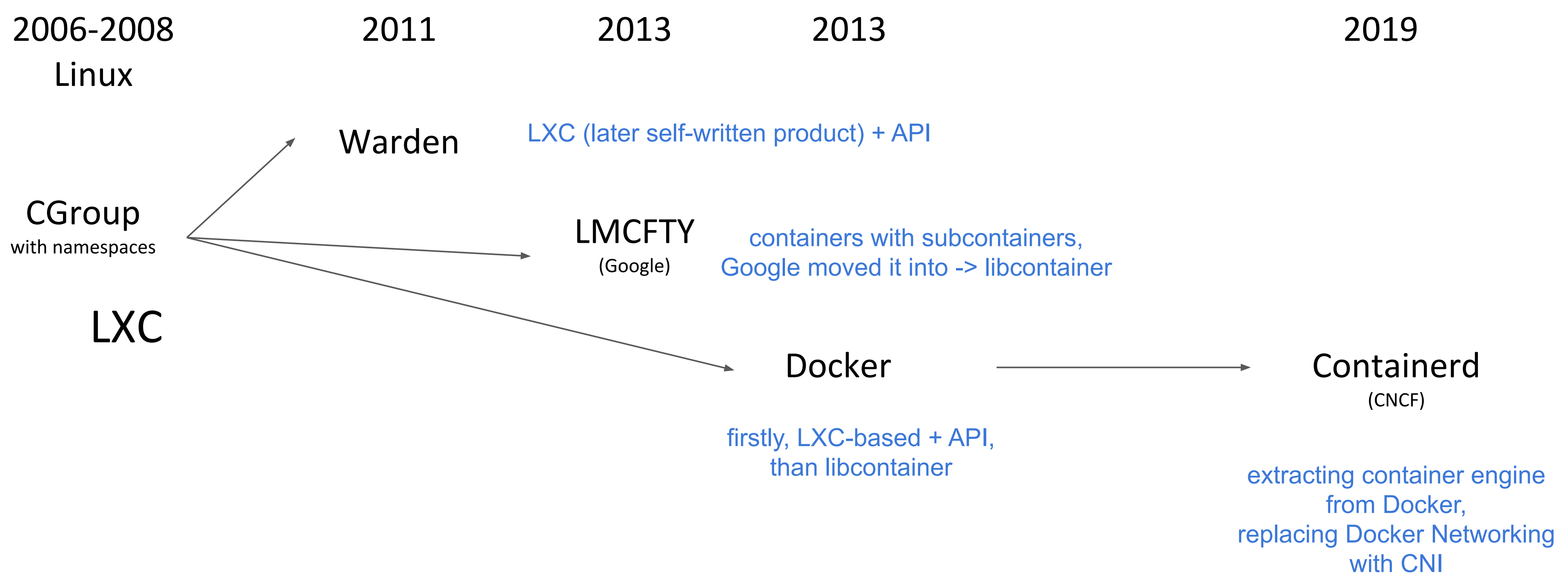
LXC →

+
+

namespaces (1 more level of abstraction)

Prepare: AWS, Frankfurt, Ubuntu 18.04 LTS, Docker: snap install docker; Public: ssh, 8080(tcp)

Unix/Linux Resource Segregation ToolsHistory


2006-2008
Linux
2011
2013
2013
2019
Warden
LXC (later self-written product) + API
CGroup
with namespaces
LMCFTY
(Google)
containers with subcontainers,
Google moved it into -> libcontainer
LXC
Docker
firstly, LXC-based + API,
than libcontainer
Containerd
(CNCF)
extracting container engine
from Docker,
replacing Docker Networking
with CNI

Prepare: AWS, Frankfurt, Ubuntu 18.04 LTS, Docker: snap install docker; Public: ssh, 8080(tcp)

Chroot is The Father of Containers

Prepare: AWS, Frankfurt, Ubuntu 18.04 TLS, Docker: snap install docker; Public: ssh, 8080(tcp)

Docker: Beginnig

Hands On: Docker Run, Docker ps

```
# docker run centos echo "hello world"
hello world

# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS        NAMES
08c65bc171c3        centos              "echo 'hello world'"     4 minutes ago       Exited (0) 4 minutes ago                 boring_wiles

# docker ps -as
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS       NAMES       SIZE
08c65bc171c3        centos              "echo 'hello world'"     4 minutes ago       Exited (0) 4 minutes ago                boring_wiles   0B (virtual
220MB)
```

If have created more than one - remove other by executing “docker rm” following by removing docker IDs:

```
# docker rm 1fcee9605349 08c65bc171c3
1fcee9605349
08c65bc171c3
```

Hands On: Docker start, image

```
# docker start 08c65bc171c3
08c65bc171c3

# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

# docker logs -f 08c65bc171c3
hello world
hello world

# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              0f3e07c0138f        4 weeks ago         220MB

# docker image rm 0f3e07c0138f
Error response from daemon: conflict: unable to delete 0f3e07c0138f (must be forced) - image is being used by stopped container 08c65bc171c3
```

Hands on: Key Points

- If process(es) executed in Docker container are finished - docker container stopped.

- Stopped docker containers are not removed automatically - keeping tying Docker container Resources (image, logs, volumes etc.)

- So docker container could be started again referenced by docker ID or container name!

Hands On: -it, -d, exec

```
# docker run centos /bin/bash
# docker run -it centos /bin/bash
[root@b504891d0e11 /]# yum list rpm
…
[root@b504891d0e11 /]# exit
exit
# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

# docker run -d centos /bin/bash
#
b504891d0e114152980bb3dc300f6110f8860b083f8b7d32ecfaca95859ded91
# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
b504891d0e11        centos              "sleep 1200"        9 minutes ago       Up 9 minutes                            nifty_sammet

# docker exec -it b504891d0e11 /bin/bash
[root@b504891d0e11 /]# ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  23024  1380 ?        Ss   11:28   0:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1200
root        21  6.3  0.3  12028  3224 pts/0    Ss   11:37   0:00 /bin/bash
root        34  0.0  0.3  46340  3248 pts/0    R+   11:37   0:00 ps -aux
```

Hands On Key Points

Containers:

- Containers are made to run application(s) inside them. No app running - container stopping.
- Containers allow to start on same host in different containers code with unexpected or conflicting dependencies
- What has happened in container stays in container.

Docker:

- docker simplifies log handling: just redirect all your app logs to STDOUT (standard output) - dockerd catches this and stored as log for this container

Linux Namespaces

Namespace - it`s context separation of resource management.

Now Linux kernel support 7 such types of separated contexts:

- Cgroups, IPC, Network, Mount, PID, User, UTS

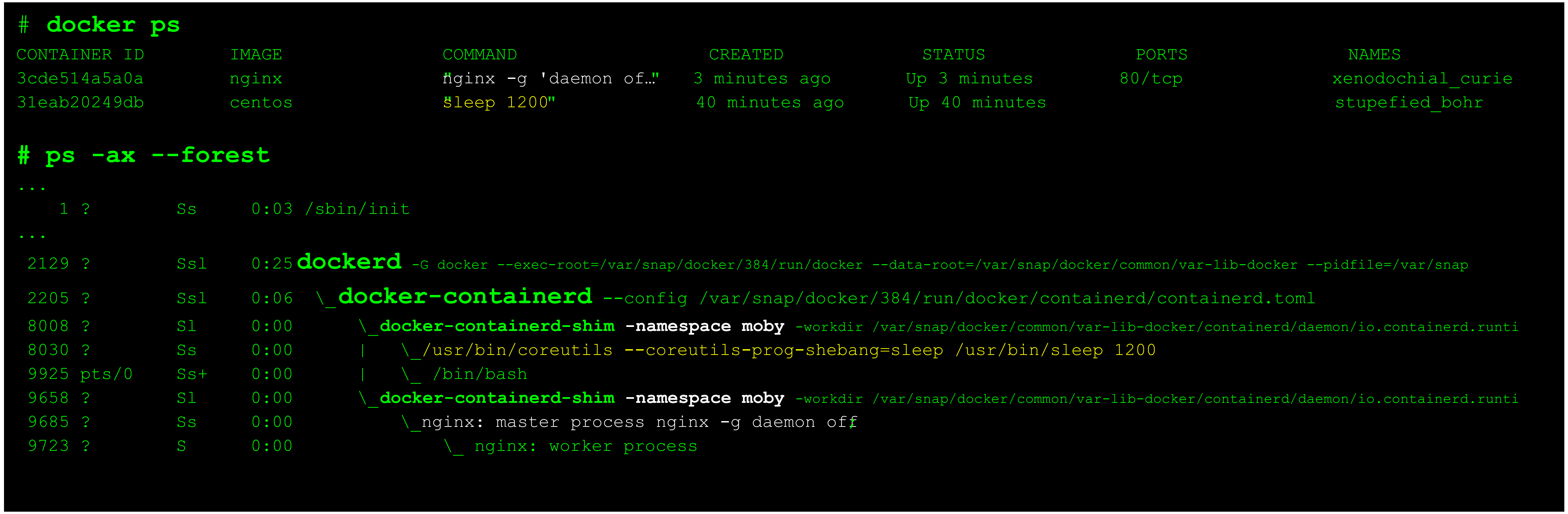
Visualize namespaces for some process:

```
# ls -l /proc/2068/ns
total 0
lrwxrwxrwx 1 root root 0 Nov  2 23:15 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Nov  2 23:15 ipc -> 'ipc:[4026532229]'
lrwxrwxrwx 1 root root 0 Nov  2 23:15 mnt -> 'mnt:[4026532227]'
lrwxrwxrwx 1 root root 0 Nov  2 23:11 net -> 'net:[4026532232]'
lrwxrwxrwx 1 root root 0 Nov  2 23:15 pid -> 'pid:[4026532230]'
lrwxrwxrwx 1 root root 0 Nov  2 23:15 pid_for_children -> 'pid:[4026532230]'
lrwxrwxrwx 1 root root 0 Nov  2 23:15 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov  2 23:15 uts -> 'uts:[4026532228]'
```

Create namespace forresource: unshare -u <binary> (u - UTS)

Docker Processes From Outside

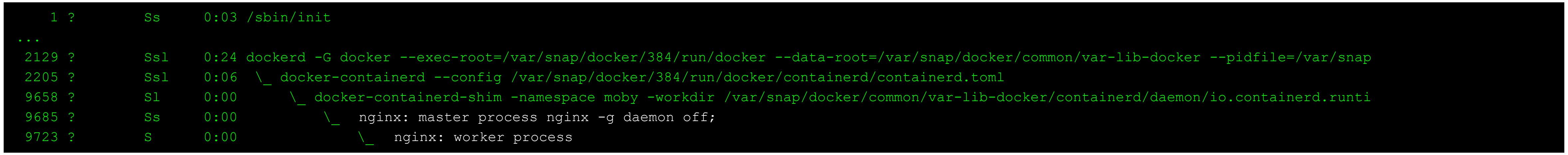
Hipster Docker:



```
# docker ps
CONTAINER ID        IMAGE               COMMAND                   CREATED            STATUS           PORTS               NAMES
3cde514a5a0a        nginx               "nginx -g 'daemon of…"   3 minutes ago     Up 3 minutes     80/tcp              xenodochial_curie
31eab20249db        centos              "sleep 1200"             40 minutes ago    Up 40 minutes                        stupefied_bohr

# ps -ax --forest
...
    1 ?        Ss     0:03 /sbin/init
...
 2129 ?        Ssl    0:25 dockerd -G docker --exec-root=/var/snap/docker/384/run/docker --data-root=/var/snap/docker/common/var-lib-docker --pidfile=/var/snap
 2205 ?        Ssl    0:06  \_ docker-containerd --config /var/snap/docker/384/run/docker/containerd/containerd.toml
 8008 ?        Sl     0:00      \_ docker-containerd-shim -namespace moby -workdir /var/snap/docker/common/var-lib-docker/containerd/daemon/io.containerd.runti
 8030 ?        Ss     0:00      |   \_/usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1200
 9925 pts/0    Ss+    0:00      |   \_ /bin/bash
 9658 ?        Sl     0:00      \_ docker-containerd-shim -namespace moby -workdir /var/snap/docker/common/var-lib-docker/containerd/daemon/io.containerd.runti
 9685 ?        Ss     0:00          \_nginx: master process nginx -g daemon off;
 9723 ?        S      0:00              \_ nginx: worker process
```

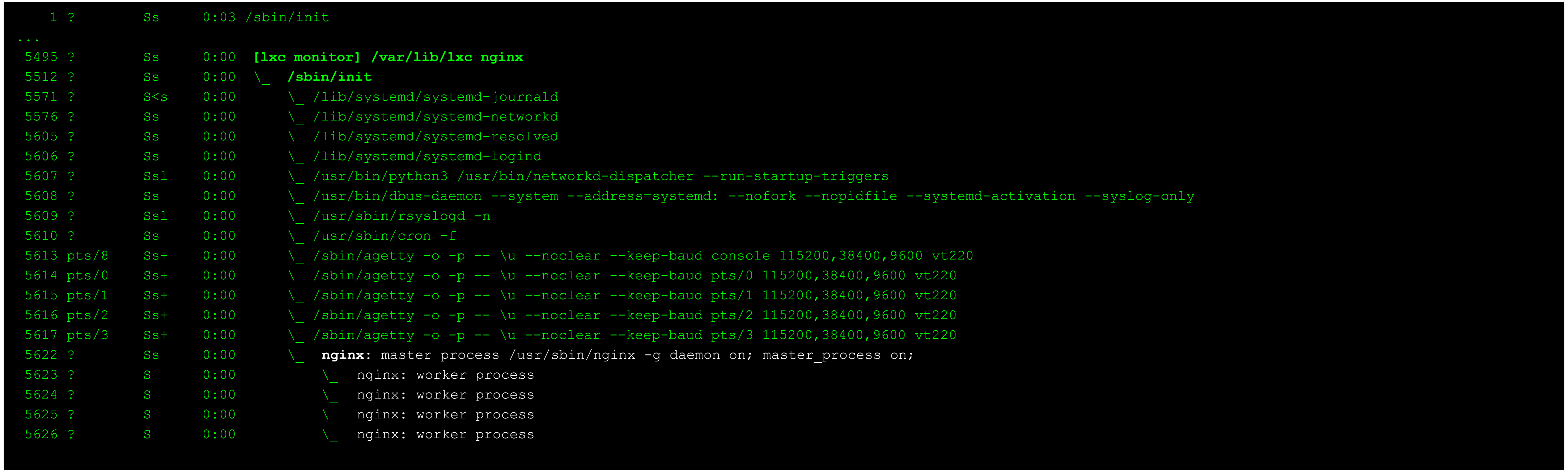
Docker versus LXContainer

Hipster Docker:



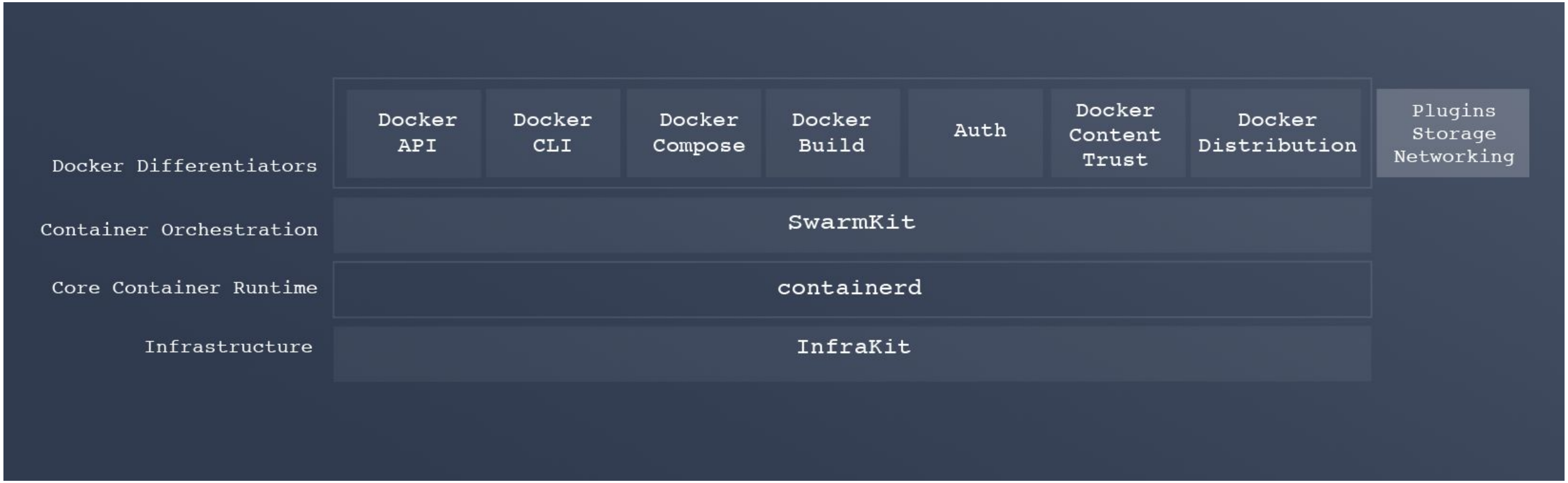
```
    1 ?        Ss     0:03 /sbin/init
...
 2129 ?        Ssl    0:24 dockerd -G docker --exec-root=/var/snap/docker/384/run/docker --data-root=/var/snap/docker/common/var-lib-docker --pidfile=/var/snap
 2205 ?        Ssl    0:06  \_ docker-containerd --config /var/snap/docker/384/run/docker/containerd/containerd.toml
 9658 ?        Sl     0:00      \_ docker-containerd-shim -namespace moby -workdir /var/snap/docker/common/var-lib-docker/containerd/daemon/io.containerd.runti
 9685 ?        Ss     0:00          \_  nginx: master process nginx -g daemon off;
 9723 ?        S      0:00              \_  nginx: worker process
```

True LXC:



```
    1 ?        Ss     0:03 /sbin/init
...
 5495 ?        Ss     0:00  [lxc monitor] /var/lib/lxc nginx
 5512 ?        Ss     0:00  \_  /sbin/init
 5571 ?        S<s    0:00      \_ /lib/systemd/systemd-journald
 5576 ?        Ss     0:00      \_ /lib/systemd/systemd-networkd
 5605 ?        Ss     0:00      \_ /lib/systemd/systemd-resolved
 5606 ?        Ss     0:00      \_ /lib/systemd/systemd-logind
 5607 ?        Ssl    0:00      \_ /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
 5608 ?        Ss     0:00      \_ /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
 5609 ?        Ssl    0:00      \_ /usr/sbin/rsyslogd -n
 5610 ?        Ss     0:00      \_ /usr/sbin/cron -f
 5613 pts/8    Ss+    0:00      \_ /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
 5614 pts/0    Ss+    0:00      \_ /sbin/agetty -o -p -- \u --noclear --keep-baud pts/0 115200,38400,9600 vt220
 5615 pts/1    Ss+    0:00      \_ /sbin/agetty -o -p -- \u --noclear --keep-baud pts/1 115200,38400,9600 vt220
 5616 pts/2    Ss+    0:00      \_ /sbin/agetty -o -p -- \u --noclear --keep-baud pts/2 115200,38400,9600 vt220
 5617 pts/3    Ss+    0:00      \_ /sbin/agetty -o -p -- \u --noclear --keep-baud pts/3 115200,38400,9600 vt220
 5622 ?        Ss     0:00      \_  nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
 5623 ?        S      0:00          \_  nginx: worker process
 5624 ?        S      0:00          \_  nginx: worker process
 5625 ?        S      0:00          \_  nginx: worker process
 5626 ?        S      0:00          \_  nginx: worker process
```

Nowdays Docker Structure



Microservice Architecture Concept

What Mean Microservice

- Microservices are a software development technique —a variant of the service-oriented architecture (SOA) structural style— that arranges an application as a collection of loosely coupled services.[1] In a microservices architecture, services are fine-grained and the protocols are lightweight. [Wiki]

- For instance, Amazon's policy is that the team implementing a microservice should be small enough that they can be fed by two pizzas. [some more Wiki]

Microservice by Microservice.io

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

Application Into Docker

Pushing App Into Containers

Ways how to put your app into container:

1. Take a look around - possibly someone already done this. Docker Hub.
2. Start container, add your code into it, commit. Docker image.
3. Build container with your code from scratch. Dockerfile.
4. If your app code is changed during execution OR/AND logic is not separated from data OR/AND you just don`t want to put it into container but should - use volumes.

1. Docker Hub

1. Official Docker Repo
2. Image could be both pulled and pushed to.
3. Free for some size.

To pull image:

```
# docker pull ubuntu:19.10
```

Running container from not pulled image automatically pulls it:

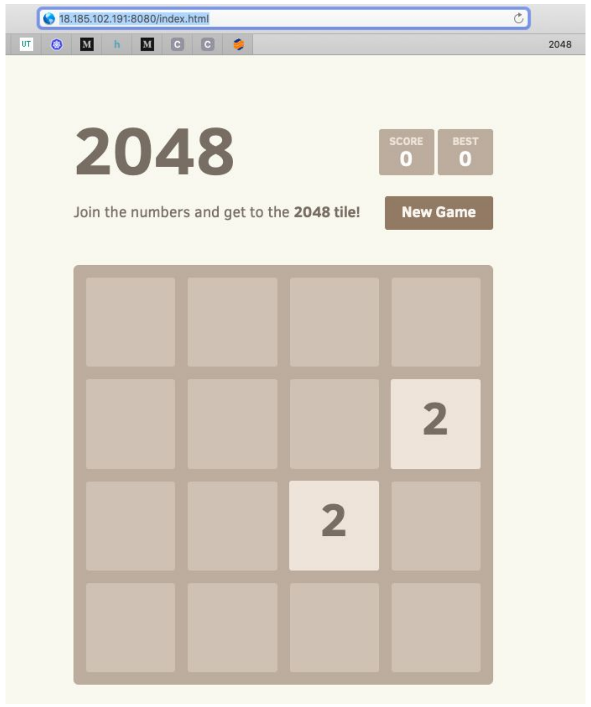
```
# docker run -d --name daydreaming_newton nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
8d691f585fa8: Pull complete
5b07f4e08ad0: Pull complete
abc291867bca: Pull complete
Digest: sha256:922c815aa4df050d4df476e92daed4231f466acc8ee90e0e774951b0fd7195a4
Status: Downloaded newer image for nginx:latest
b28340a80ba178ace4bcd59fa153a7fc149743a340d9cf19db543f8f220274b8
```

2. Hands On: Docker COPY, Commit

```
# docker run -d -p 8080:80 nginx
1fbe97d9c731…….

# git clone https://github.com/gabrielecirulli/2048.git
# cd 2048/; docker cp ./ 1fbe97d9c731:/usr/share/nginx/html
```

http://18.185.102.191:8080/index.html

```
# docker image ls
nginx           latest          2622e6cca7eb       10 days ago         132MB

# docker ps -s
#docker commit a45630804dc1
sha256:a53cd93bc1b89232c6ecf91eb50a22320fca5183e76df5453e8768148cee7e15

# docker image ls
REPOSITORY         TAG              IMAGE ID          CREATED             SIZE
<none>             <none>           a53cd93bc1b8      About a minute ago   133MB
nginx           latest          2622e6cca7eb       10 days ago         132MB
#docker stop 1fbe97; docker run -p 8080:80 -d a53cd93bc1b8
```


18.185.102.191:8080/index.html
2048
SCORE 0
BEST 0
Join the numbers and get to the 2048 tile!
New Game
2
2

3. Hands On: Dockerfile

```
# mkdir docker; git clone https://github.com/gabrielecirulli/2048.git docker/2048; vim Dockerfile
```

```
FROM nginx
COPY 2048/ /usr/share/nginx/html/
```

```
~/docker# docker build ./ -t 2048game
Sending build context to Docker daemon  1.346MB
Step 1/2 : FROM nginx
 ---> 540a289bab6c
Step 2/2 : COPY 2048/ /usr/share/nginx/html/
 ---> 960c02a8cf80
Successfully built 960c02a8cf80
Successfully tagged 2048game:latest

~/docker# docker image ls
REPOSITORY       TAG          IMAGE ID        CREATED          SIZE
2048game         latest       0bc5c1e414d8    13 seconds ago   133MB
<none>           <none>       a53cd93bc1b8    14 minutes ago   133MB
nginx            latest       2622e6cca7eb    11 days ago      132MB

# docker run -p 8080:80 -d 0bc5c1e414d8
```

3. Docker Image Layers

```
~/docker# docker image ls
REPOSITORY        TAG           IMAGE ID        CREATED          SIZE
2048game          latest        cbc77a65d75a    13 seconds ago   133MB
<none>            <none>        05b3d60c717d    14 minutes ago   133MB
nginx             latest        2622e6cca7eb    11 days ago      132MB
```

```
~/docker# docker image inspect cbc77a65d75a
...
    "RootFS": {
        "Type": "layers",
        "Layers": [
            "sha256:13cb14c2acd3...",
            "sha256:d4cf327d8ef50...",
            "sha256:7c7d7f446182...",
            "sha256:9040af41bb66...",
            "sha256:f978b9ed3f26a...",
            "sha256:61fe62a4f2901..."
        ]
    },
...
```

```
~/docker# docker image inspect 05b3d60c717d

...
    "RootFS": {
        "Type": "layers",
        "Layers": [
            "sha256:13cb14c2acd3...",
            "sha256:d4cf327d8ef50...",
            "sha256:7c7d7f446182...",
            "sha256:9040af41bb66...",
            "sha256:f978b9ed3f26a...",
            "sha256:85fc12c04ec79..."
        ]
    },
```

```
# docker ps -as
CONTAINER ID    IMAGE     COMMAND                   CREATED          STATUS                     PORTS     NAMES            SIZE
4c11769c2cf6    nginx     "/docker-entrypoint.…"    4 minutes ago    Exited (0) 8 seconds ago             thirsty_meitner  1.29MB (virtual 133MB)
```

Image: Layers, Dockerfile

Docker images are layered.
Hash of each layer includes files changes made before layer is finished and semi-hash from previous layers.

1 commit = 1 layer
1 line of Dockerfile = 1 Layer

```
FROM ubuntu:18.04
VOLUME /app
VOLUME /data
ENV TZ=Europe/Kiev
RUN apt-get update && apt-get install --no-install-recommends --no-install-suggests -y git python3 python3-pip python3-setuptools python3-dev python3-psycopg2
RUN pip3 install mysql-connector pyyaml
RUN pip3 install docker-py
RUN pip3 install psycopg2
COPY ./app/ /app/
CMD /app/cycle.sh
```

↑ Put upper basical non frequently changed parts

↓ Put at the end more frequently chaged parts

4. Hands On: Docker Volumes

Volume in Docker is looking like mount -bind directory.

```
~# docker ps -as
...
~# mkdir -p registry-storage;
~# docker run -d -p 5000:5000 -v registry-storage:/var/lib/registry registry:2
dee2ac82f8ff9896987059f64f4a6dc25e5cbe998417f5ba2ff77f6d7f980b9e

~# docker volume ls
DRIVER          VOLUME NAME
local         412b07e4ecf7c735e128458b33c3dd16735c66d0a799dbee5dd1da211740aeb0
local         85cb4930feab7b2663b5846a87e0adcf05f6ca0763c42ce34fb77e5e2f52fafd
local         9e698b47f5a2e24514418514fdec4deb60cac5bf4433689209d87bc5a15ef4ca
local             registry-storage
```

If volume declared in Dockerfile and not mounted on start - Docker automatically creates volume on write access to declared Volume mount point.

```
FROM ubuntu:18.04
VOLUME /app
```

Volumes could be mounted from outside using drivers like NFS. And same volume could be mounted to more than on Docker container!

Hands On: Docker App Distributing, Tag, Registry

Tagging is advertised for images management

Docker Registry - your own Docker Hub.

```
~# docker ps | grep registry
dee2ac82f8ff      registry:2       "/entrypoint.sh /etc…"   2 minutes ago     Up 2 minutes      0.0.0.0:5000->5000/tcp   nervous_kare
```

Docker Tag, Push

```
~# docker tag a53cd93bc1b8 2048game:v01
~# docker tag a53cd93bc1b8 localhost:5000/2048game:v01
~# docker image ls
REPOSITORY          TAG            IMAGE ID          CREATED           SIZE
2048game            latest         0bc5c1e414d8      11 hours ago      126MB
2048game            v01            a53cd93bc1b8      12 hours ago      128MB
localhost/2048game  v01            a53cd93bc1b8      12 hours ago      128MB

~# docker push localhost:5000/2048game:v01
The push refers to repository [localhost:5000/2048game]
c64aa9c614dd: Pushed
a89b8f05da3a: Pushed
6eaad811af02: Pushing [==========================>                  ]  29.77MB/56.98MB
b67d19e65ef6: Pushing [===================>                         ]  26.54MB/69.23MB
```

Hands on: Basic Docker Networking

Exposing a port (making it available - doesn`t mean forwarding is working)

```
FROM ubuntu:18.04
RUN apt-get update; apt-get install nginx
EXPOSE 80
```

Forwarding a port

```
#  docker run -d -p 8080:80 --name nginx nginx
c2fcf6b9017b47ffd45d774697ba350f23cc972065b911e8711a096569c196c1
# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                  NAMES
c2fcf6b9017b        nginx               "nginx -g 'daemon of…"   3 seconds ago       Up 2 seconds        0.0.0.0:8080->80/tcp   nginx
```

Available 3 types of Docker networking:

1) To docker default bridge (default behaviour, worked because Docker running DHCP)
2) Docker to physical interface
3) Docker without network (unmapped)

Docker Networking: iptables, bridging

```
~# brctl show docker0
bridge name  bridge id         STP enabled interfaces
docker0      8000.0242827baa10   no          vetheb31987

~# iptables -vnL -t nat
…

~# iptables -vnL
…
```

What Makes Docker in Containers a Xerox in Copy Machines

Out of the box:

- simple networking (automation of bridging, iptables*)
- Dockerfiles (from code management point of view)
- encapsulating code into images
- dockerd adoption of images on different systems
- cool layering of images
- containers distributing hub (global and local)
- volumes (shared folders)
- simplified logging.

Next Sections

Section 2. Docker: something from under the hood

- Dockerbuild file: more options, more pain.
- More than 1 App Achievements:
    - Environment Variables, Secrets; Volumes sharing;
    - Docker Link.
- Docker Networking;

Section 3. Kuber: beginning

- Microservice App Achievements
    - App Upstart Dependencies;
    - Service Discovery;
    - DNSing.
- Docker Compose.
- Docker Swarm.
- Kuber: Docker ambitions cutter.
- Container.d: Docker dissolver.

Howe Work 1

Home Task: https://github.com/ask4ua/DKN/blob/master/Hometask/Section1/README.md

Email: volodymyr.volkov@globallogic.com

Deadline: 1 week - Next Friday