



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 root use: "sudo su -"*

Lecture 1: Container - What Are You?

OS Level User Process Isolation

Became meaningful on multitasking introduction.

Initial low-level understanding:

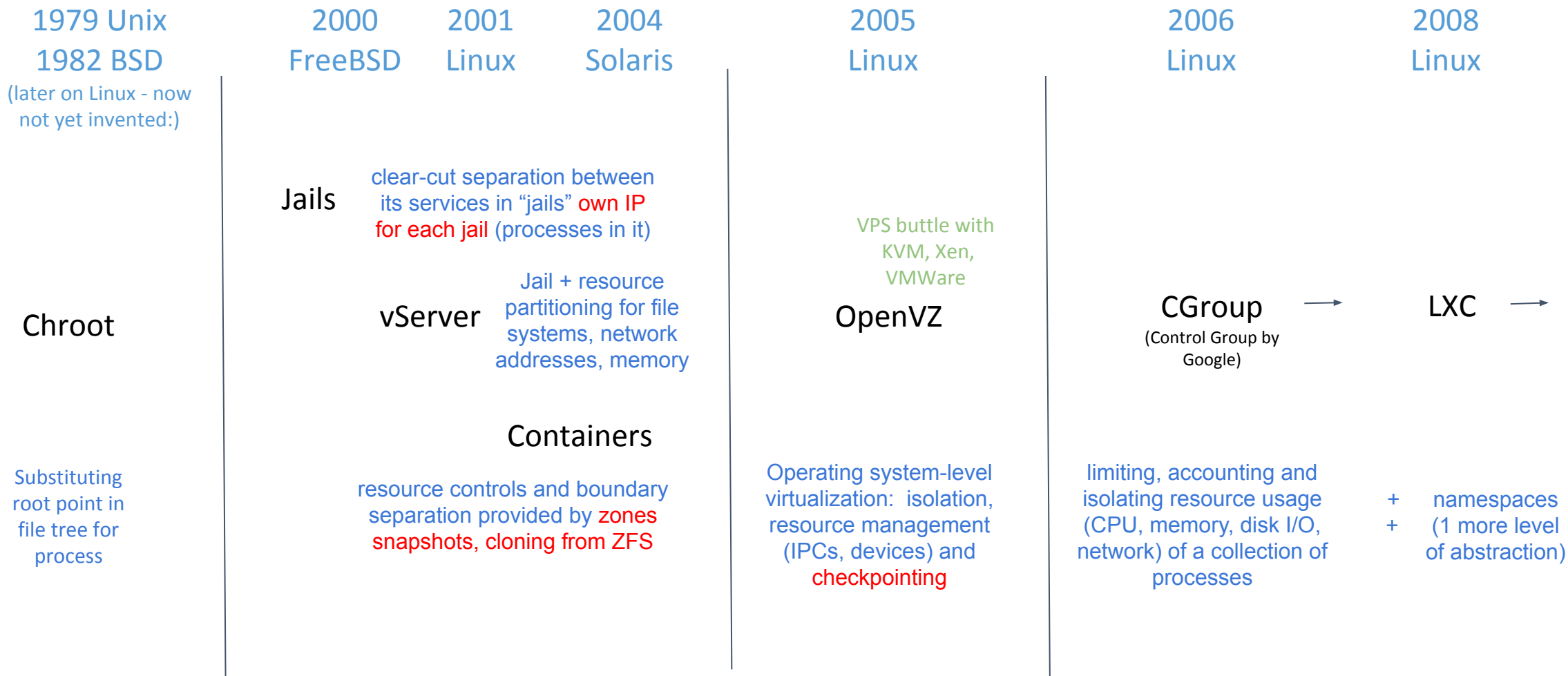
Process isolation is a set of different hardware and software technologies[1] designed to protect each process from other processes on the operating system.

...

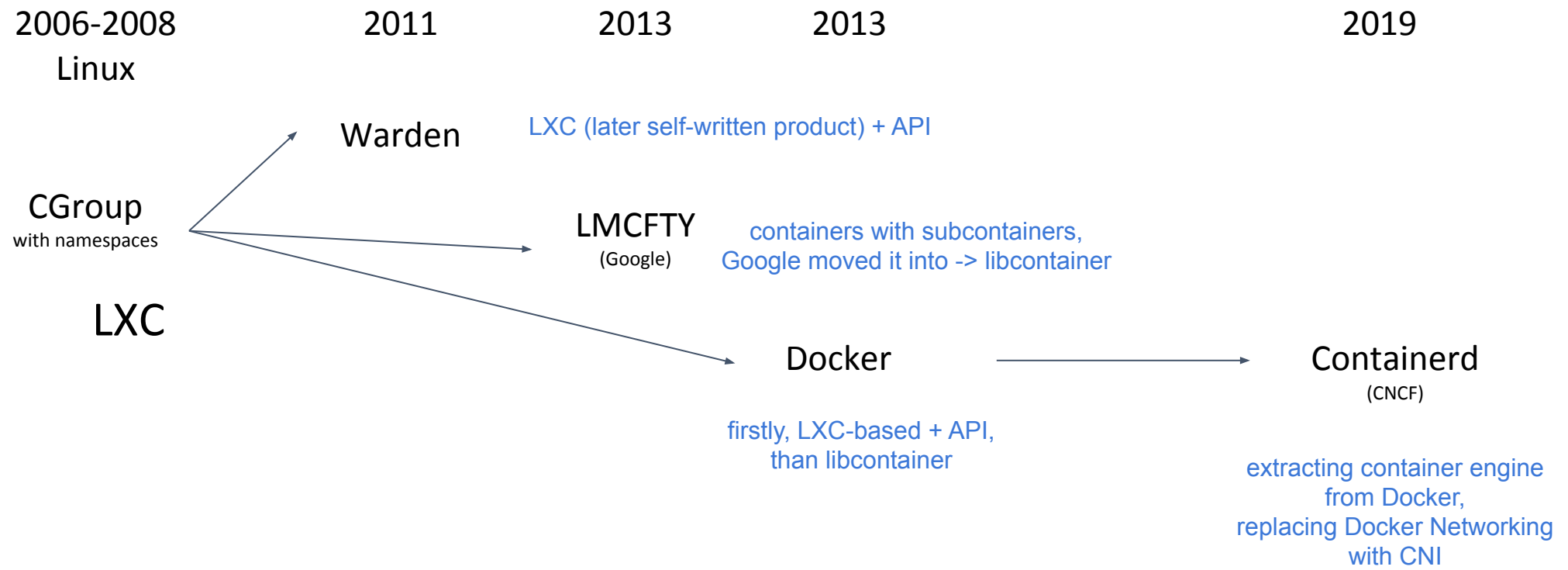
*Security is easier to enforce by disallowing inter-process memory access, in contrast with less secure architectures such as **DOS** in which any process can write to any memory in any other process.*

WIKI

Unix/Linux Resource Isolation Tools History



Unix/Linux Resource Segregation ToolsHistory



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
b504891d0e11	centos	"sleep 1200"	9 minutes ago	Up 9 minutes		nifty_sammet

```
# 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 for resource: `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

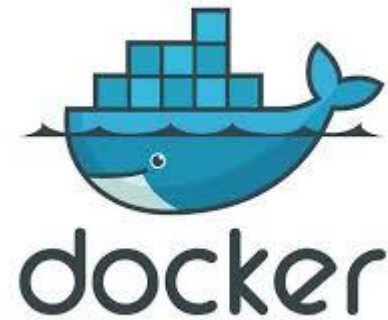
What Mean Microservice

- 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
```

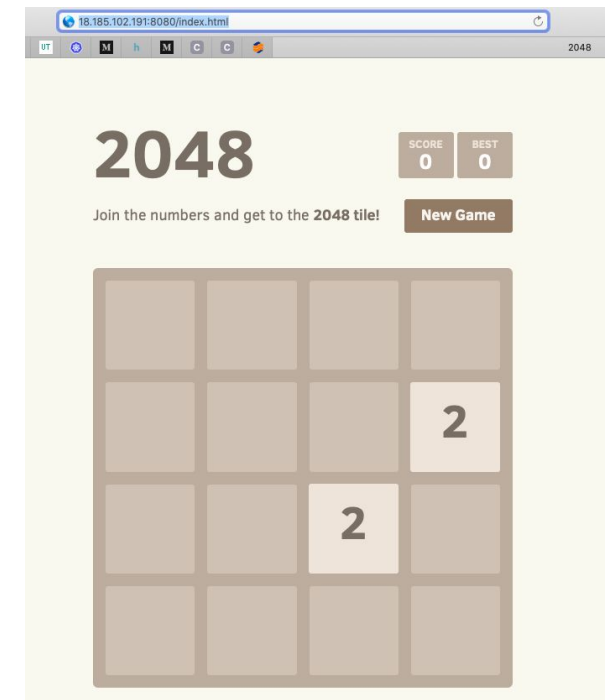
```
#docker commit a45630804dc1
```

```
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
```



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

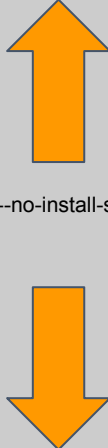
Docker images are layered.

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-psycpg2
RUN pip3 install mysql-connector-pyyaml
RUN pip3 install docker-py
RUN pip3 install psycpg2
COPY ./app/ /app/
CMD /app/cycle.sh
```



Put upper basic non frequently changed parts

Put at the end more frequently changed 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

Q&A



Section 2: Docker: More From Under The Hood.

Practice Requirements

- AWS EC2
 - Frankfurt
 - Ubuntu Server 18.04 LTS (HVM)
 - t2.micro 1 instance
 - 1 Public IP
 - Security:
 - ssh from your public IP
 - 80,8080 http (tcp) from your public IP
 - Install Docker: snap install docker
 - *JFYI: ssh user: ubuntu*

Docker Networking

Type	Docker run Option	How it works	Peculiarities
Hosted	--net=host	Mapping all hypervisor interfaces into container (same network namespace from host referenced into container ns)	If container not privileged - only could occupy free ports on iface. If privileged - all could be done including changing ifaces IPs
None	--net=none	No network interfaces created inside container.	But dedicated namespace still created on start (at least was so)
Default: bridged + private networks	<nothing> + --net=somenetname	(mostly named docker0)	Private Networks organized by internal Docker DNS on 127.0.0.11 address and iptables.
Mapped from another container		All interfaces (namespaces) from one container reused in another - like in Hosted Type	Different containers could communicate with each other through any IP/iface - even through 127.0.0.1. Ports shouldn't override

Hands On: Docker Networking Hosted

in hypervisor:

:\$ ip addr

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel state UP group default qlen 1000
    link/ether 06:2c:a5:cc:29:00 brd ff:ff:ff:ff:ff:ff
    inet 172.31.40.231/20 brd 172.31.47.255 scope global dynamic eth0
        valid_lft 3163sec preferred_lft 3163sec
    inet6 fe80::42c:a5ff:fecc:2900/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:2e:e7:33:cd brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
```

:\$ docker run -it --rm --net=host centos /bin/bash

/\$ ip addr

/\$ ip addr add 10.0.0.1/24 dev eth0

RTNETLINK answers: Operation not permitted

Hands On: Docker Networking None

```
:$ docker run -it --rm --net=none centos /bin/bash
```

```
/ $ ip addr
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever
```

Hands On: Docker Bridged

```
:$ ip addr show docker0
```

```
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default  
    link/ether 02:42:2e:e7:33:cd brd ff:ff:ff:ff:ff:ff  
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0  
        valid_lft forever preferred_lft forever  
    inet6 fe80::42:2eff:fee7:33cd/64 scope link  
        valid_lft forever preferred_lft forever
```

```
:$ docker run -it --rm centos /bin/bash
```

```
[root@60dcba2e9635 /]# ip addr show
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
11: eth0@if12: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default  
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0  
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0  
        valid_lft forever preferred_lft forever
```

Dockerfile - More Options

```
FROM openjdk:8-jdk-stretch
...
RUN apt-get update && apt-get upgrade -y && apt-get install -y ... && rm -rf ...
...
ARG user=jenkins
...
ENV JENKINS_HOME $JENKINS_HOME
...
RUN mkdir -p $JENKINS_HOME \
&& useradd -d "$JENKINS_HOME" -u ${uid} -g ${gid} -m -s /bin/bash ${user}
...
VOLUME $JENKINS_HOME
...
EXPOSE ${http_port}
...
USER ${user}
...
ENTRYPOINT ["/sbin/tini", "--", "/usr/local/bin/jenkins.sh"]
...
COPY install-plugins.sh /usr/local/bin/install-plugins.sh
ADD https://some.git.url
```

RUN - execute command inside container during building the image

ARG - local for dockerfile variable useful and overridable only in “docker build”, refrencable with **`\${ARG name}`**.

ENV - advertises variable injected into Environment Variables in container by container start, overridable in “docker run”

VOLUME - creating directory that could be referenced in docker run command as volume mounting point by name (without full path).

EXPOSE - port container advertised to outside (needed for iptables auto-rules adding), mapping to real port is run option

USER - container will be executed as process of defined user

ENTRYPOINT, CMD - both could set binary/script started on container start, more details on the next slide

COPY - copy, path os source from building directory

ADD - enhanced COPY, supporting wildcards, --chown and URLs

Dockerfile - ENTRYPOINT, CMD

ENTRYPOINT, CMD - why 2 options to set an upstart command?

ENTRYPOINT - purposed to define binary for process #1 in container, if used - CMD is referred as it's parameters.
Could be overridden with --entrypoint option on start.

CMD - purposed to be redfinable on container start, set by the command in run line after image name: `docker run -it centos /bin/bash`. If used without ENTRYPOINT - substitute one.

Docker container goal	Dockerfile	docker run command	Executed on upstart script
Show time -no input options	FROM ubuntu ENTRYPOINT date	docker run date	/bin/sh date
		docker run date +%Z	/bin/sh date
	FROM ubuntu ENTRYPOINT ['date']	docker run date	date
Show time - options eligible and overridable	FROM ubuntu ENTRYPOINT ['date'] CMD ['+%A']	docker run date	date +%A
		docker run date +%Y	date +%Y
Show time - run script overridable	FROM ubuntu CMD date	docker run date	/bin/sh date
		docker run /bin/bash	/bin/bash

Dockerfile - FROM

FROM could use not just some image from repository but also:

- defined “[from scratch](#)” for minimal images,
- use artifacts from intermediate image constructed in the same manifest - named as [multistage](#)

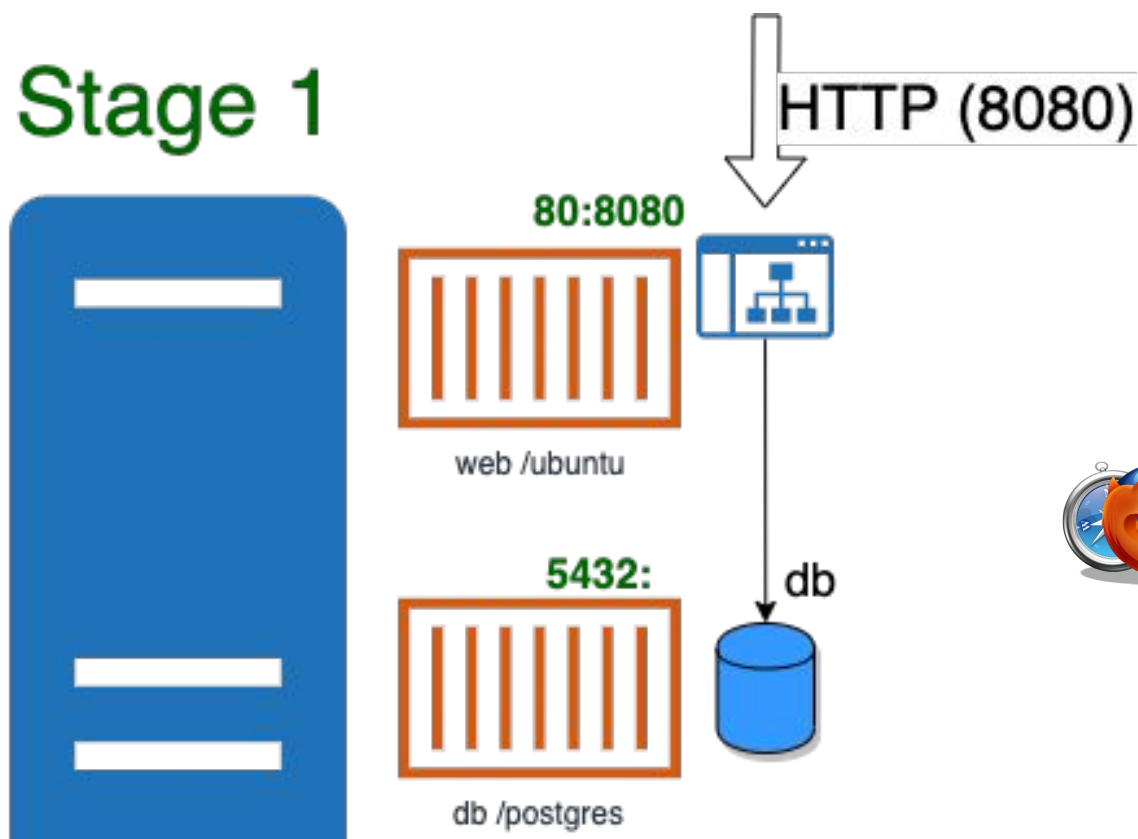
```
FROM scratch
COPY hello /
CMD ["/hello"]
```

```
FROM golang:1.7.3 AS builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href-counter/app .
CMD ["/app"]
```

Hands On: Small Web App With DB, Stage 1

Stage 1



```
:$ git clone https://github.com/ask4ua/DKN  
:$ docker network create mynet
```

```
Stage1/web:$ docker build ./ -t web  
Stage1/web:$ docker run -p 8080:80 -d --name web --net mynet web
```

```
Stage1/db:$ docker build ./ -t db  
Stage1/db:$ docker run -d --name db --net mynet db
```



Serving host ec99fa097683

Time: 2019-11-08 17:41:29

Accessed path: /

Writing to DB status: **Fail**

Hands On: Docker Limitations

Docker Doesn't Containers Upstart Dependendencies

```
:$ docker logs -f web
```

```
Mon Jul 13 15:33:58 2020 webapp: HTTP Server Starts
```

```
Mon Jul 13 15:33:58 2020 webapp: Initiating connection to DB
```

```
DB ERROR: Something is wrong in connecting to DB: could not translate host name "db" to address: Temporary failure in name resolution
```

```
:$ docker stop web
```

```
:$ docker start web
```

Docker Bridged --net

```
:$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ee9d1bfda40f	web	"python3 -u /app/web..."	4 minutes ago	Up 27 seconds	0.0.0.0:8080->80/tcp	web
14833ca9e600	db	"docker-entrypoint.s..."	5 minutes ago	Up 32 seconds	5432/tcp	db

```
:$ docker network ls
```

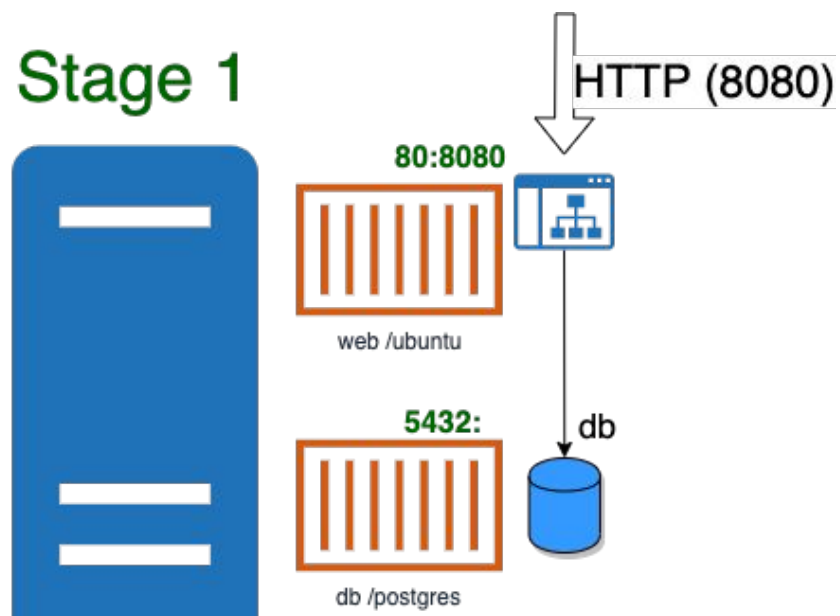
NETWORK ID	NAME	DRIVER	SCOPE
5ee8868a7c3e	bridge	bridge	local
be73f6f78fd0	host	host	local
22100c68614f	mynet	bridge	local
dba2602e072d	none	null	local

```
:$ docker network inspect mynet
```

```
$ brctl show
```

bridge name	bridge id	STP enabled	interfaces
br-22100c68614f	8000.0242823f7f96	no	veth178a6e0 vethda68891

How Docker -p Port Forwarding --net Are Working



Sysctl ip_forward enables forwarding between All interfaces.

Dockerd creates bridges, connects to them containers, dockerd like DHCP assign them IPs

Each Bridged network could have own IP space, Gateway, DNS

Dockerd manages IPtables to simulate containers isolation and to set Port Forwarding

--net mynet shared between containers has dedicated bridge and just disabling of some iptables isolation rules by iptables (no namespaces magic)

dockerd runs DNS on ip 127.0.0.11 to enable resolving IPs between containers in the same --net mynet by container names

Dockerfiles For The App

```
FROM postgres
ENV POSTGRES_USER='DBUSER' POSTGRES_DB='DBNAME' POSTGRES_PASSWORD='DBPASS'
COPY ./upstart.sh /docker-entrypoint-initdb.d
```

```
FROM ubuntu
RUN apt-get update && apt-get install --no-install-recommends --no-install-suggests -y git python3
python3-pip python3-setuptools python3-dev python3-psycpg2 stress
RUN pip3 install psycpg2
ENV DBUSER='DBUSER' DBPASS='DBPASS' DBNAME='DBNAME'
ENV DBHOST='db' DBPORT='5432'
ARG APPDIR='/app'
VOLUME ${APPDIR}
COPY src ${APPDIR}
ENTRYPOINT ["python3", "-u", "/app/webapp.py"]
```

Secrets Forwarding Into Container

	Environment Variables key = value	Volume Files with key=value, csv, archeive etc.
On Build	<p>Via arguments override:</p> <pre>-- --build-arg ARG1=Va --build-arg ARG2=Lue</pre> <p>For compatibility with on run redefine could be used in dockerfile:</p> <pre>ARG SecretArg=password ENV SecretENV=\${SecretArg}</pre>	<p>Dockerfile:</p> <pre>export DOCKER_BUILDKIT=1 RUN --mount=type=secret,id=mysite.key command-to-run</pre> <p>or just VOLUME mount and some RUN commands</p>
On Run	<p>Via Environemnt variable redefine:</p> <pre>docker run -e EnvVariable=SomePass -e AnotherEnvVariable=pass2</pre>	<p>Application should be able to read and parse files from volume</p>

Docker Link

Docker link enables to start few containers on the same hosts with shared both Volumes and Environment variables.

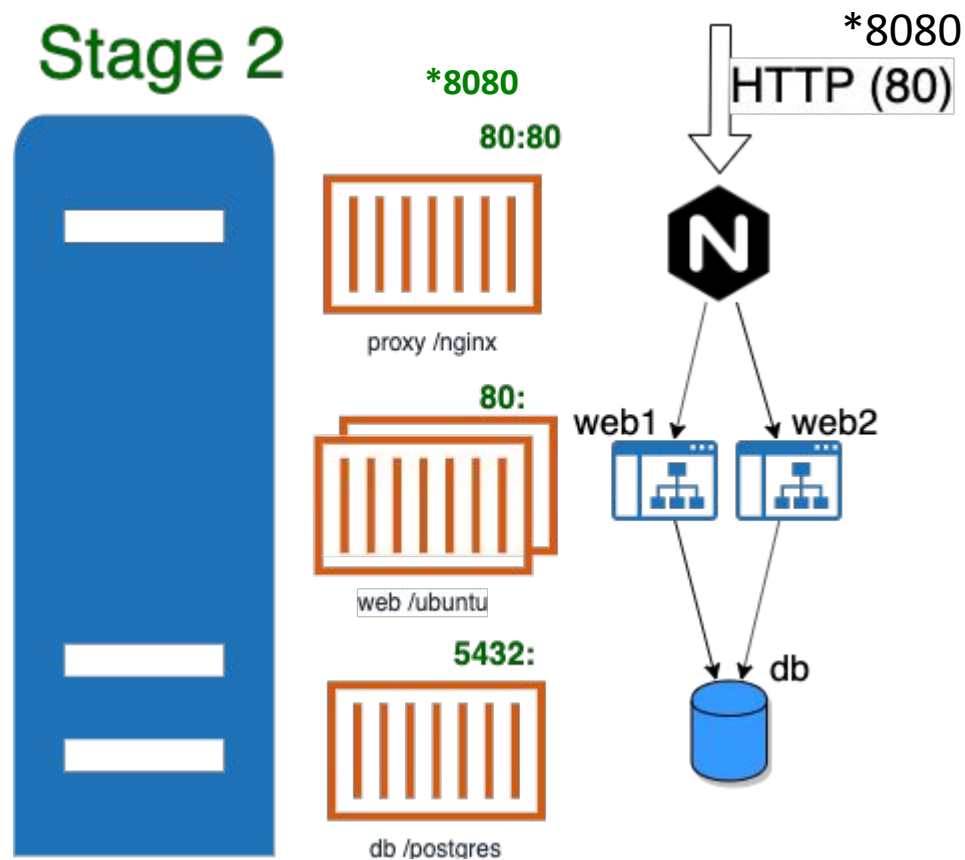
Magic is done by Docker transparently by:

- reinjecting variables on start appending them with container name prefix followed by underscore,
- DNS records are filled into /etc/hosts of container.

```
:$ docker run -d --name database -e MYSQL_ROOT_PASSWORD=root mysql
:$ docker run -d --link database:db --name webapp web

:$ docker exec -ti webapp env | grep MYSQL_ROOT_PASSWORD
DB_ENV_MYSQL_ROOT_PASSWORD=root
```

Hands On: Small Web App With DB, Stage 2



```
:$ docker stop web webapp
```

```
:$ docker run -d --name web1 --net mynet web
```

```
:$ docker run -d --name web2 --net mynet web
```

```
Stage2/proxy:$ docker build ./ -t proxy
```

```
:$ docker run -p 8080:80 -d --name proxy --net mynet proxy
```



Serving host ec99fa097683

Time: 2019-11-08 17:41:29

Accessed path: /

Writing to DB status: **Success**



Serving host 78fe3f2c0be4

Time: 2019-11-08 17:41:29

Accessed path: /

Writing to DB status: **Success**

Hands On: Small Web App With DB, Stage 2

FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf

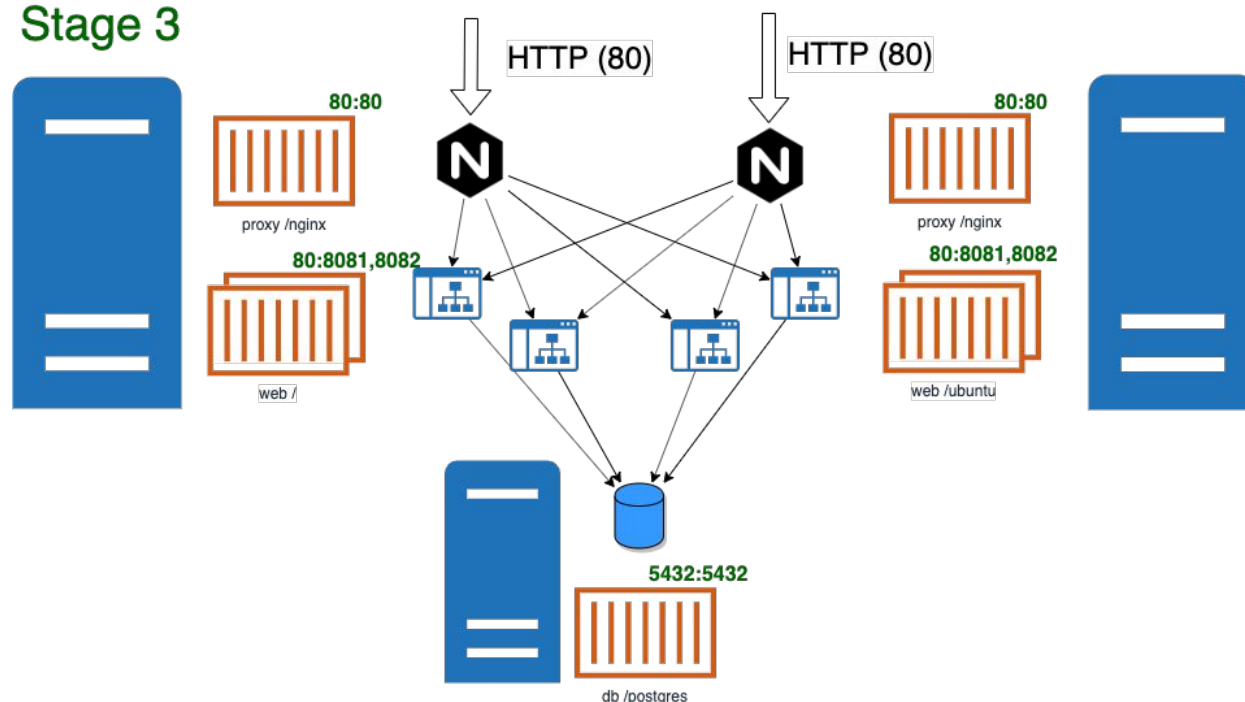
nginx.conf:

```
events { }
http {
    upstream webapp {
        server web1:80;
        server web2:80;
        keepalive 10;
    }

    server {
        resolver 127.0.0.11 valid=10s;
        listen 80;
        location / {
            proxy_pass http://webapp;
        }
    }
}
```

Lecture2 Home Task

Stage 3



Please try to: provide DB IP (docker --net DNS not working outside of host) and Secrets into docker WebApp config by ENV variable on docker run.

P.S.: Local host networking will not work for WebApps from other server!

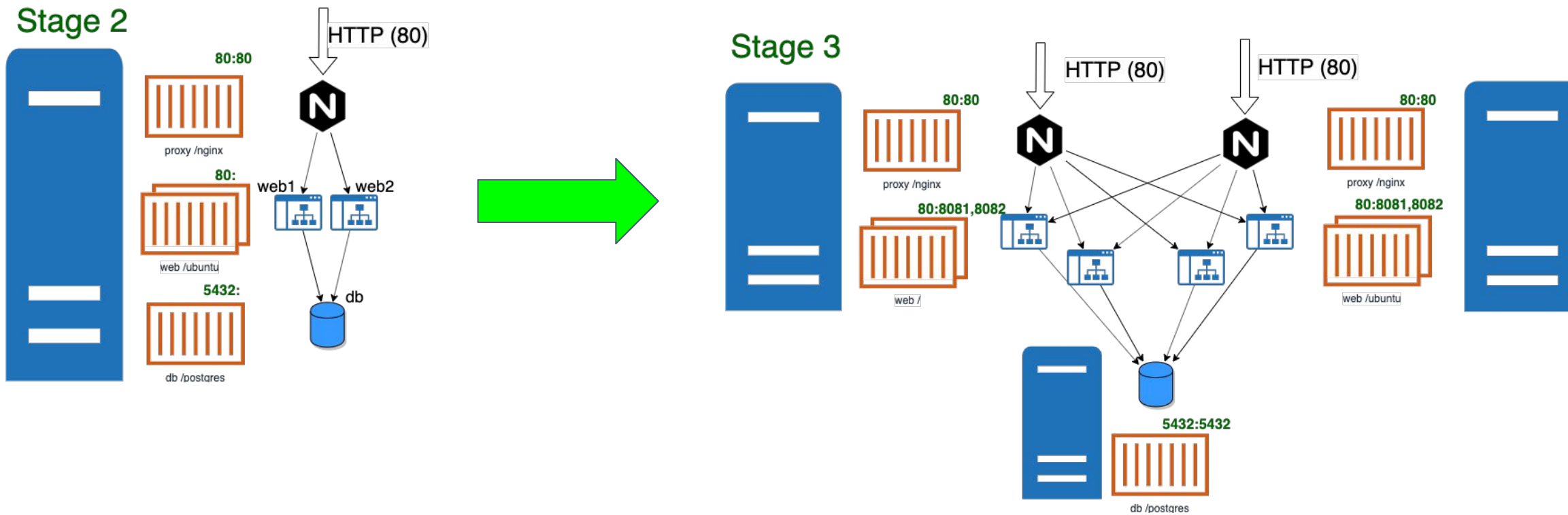
P.P.S.: Nginx Configs ok to hardcode - free NGINX doesn't support keepalives - so WebApp should be running.

Q&A



Section 3: Docker Compose, Docker Swarm, Kubernetes

Growing Service



Service Orchestration on Dockerd Requirements

What is missed in traditional docker

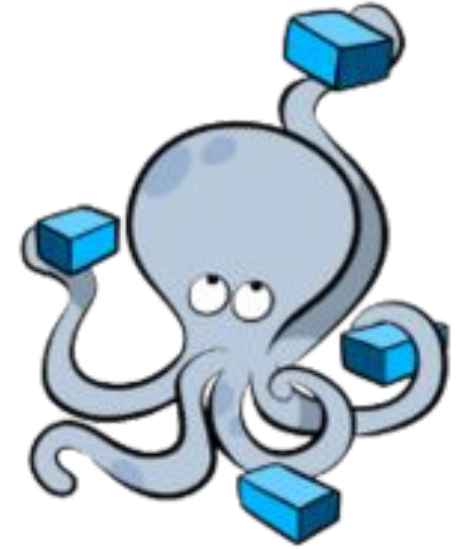
- Containers Upstart Dependencies,
- Network shared between Few Hosts, Network Isolation,
- DNS in shared network, DNS isolation,
- Scaling like a DNS++ = Service discovery
- Secrets Management

Docker Compose

Docker Compose - systemd for docker containers.

- Handles Upstart Dependencies.
- Support Scaling of Containers.
- Tracking containers status.
- Rolling updates.
- Keeping DNS Records.
- Isolating resources by namespaces (not kernel - just DNS)

But all of this only around single node (hypervisor).



Docker Compose Config Example

```
version: "3"

services:
  whir-data:
    image: localhost:5000/whir-data
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
    volumes:
      - "/home/volk/GIT/whir:/app"
      - "/home/volk/txt:/data"
    networks:
      - whirnet
  ...
```

```
...
  whir-parser:
    image: localhost:5000/whir-parser
    depends on:
      - whir-parser
    deploy:
      replicas: 1
    resources:
      limits:
        cpus: "0.5"
        memory: 128M
      restart_policy:
        condition: on-failure
    volumes:
      - "/home/volk/GIT/whir:/app"
      - "/home/volk/txt:/data"
    networks:
      - whirnet
  networks:
    whirnet:
```

Docker Swarm

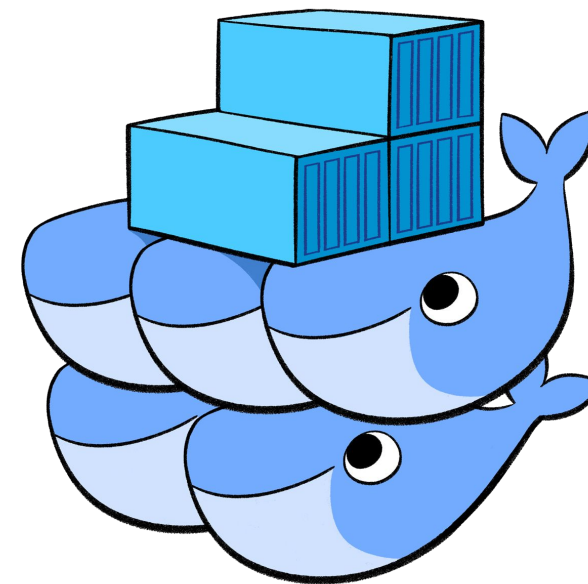
Connects few nodes(hypervisors),
roles: management, worker.

to firstly advertised in compose features added:

- + Multi-host networking,
- + Service Discovery + Load Balancing

But :

- not includes upstart sequence like docker compose.



Docker Swarm, Compose - Secrets Management

```
postgres:
  image: docker.ask4ua.com/whir-db
  ports:
    - 5432:5432
  volumes:
    - postgres_vol:/var/lib/postgresql/data
  secrets:
    - whir_db_password
    - root_db_password
  # mounted into: /run/secrets/secret-name
```

```
#environment:
  # - POSTGRES_USER=whir
  # - POSTGRES_DB=whir
  # - POSTGRES_PASSWORD=password
```

```
networks:
  - whirnet
```

```
deploy:
  replicas: 1
  restart_policy:
```

```
condition: any
```

```
secrets:
  whir_db_password:
    external: true
    #file: db_root_password.txt
  root_db_password:
    external: true
```

```
volumes:
  data_vol:
  postgres_vol:
```

```
networks:
  whirnet:
```

```
echo password | docker secret create whir_db_password -
```

Kubernetes



Kubernetes - What are you looking like?

```
docekr ps
```

```
0a28191b1f5c    rancher/hyperkube:v1.14.8-rancher1  "/opt/rke-tools/entr..." ... kube-proxy
edb47836b426    rancher/hyperkube:v1.14.8-rancher1  "/opt/rke-tools/entr..." ... kubelet
5b227fe9ddc1    rancher/hyperkube:v1.14.8-rancher1  "/opt/rke-tools/entr..." .... kube-scheduler
fc1b918ca88f    rancher/hyperkube:v1.14.8-rancher1  "/opt/rke-tools/entr..." ... kube-controller-manager
1b0a25d5baf8    rancher/hyperkube:v1.14.8-rancher1  "/opt/rke-tools/entr..." ... kube-apiserver
a890a2f32f7d    rancher/rke-tools:v0.1.50           "/opt/rke-tools/rke-..." ... etcd-rolling-snapshots
610098479e21    rancher/coreos-etcd:v3.3.10-rancher1 "/usr/local/bin/etcd..." ... etcd
...
5bd4b5b226d5    rancher/hyperkube:v1.14.8-rancher1  "/opt/rke-tools/entr..." ... kubelet
21a38aa88520    rancher/hyperkube:v1.14.8-rancher1  "/opt/rke-tools/entr..." ... kube-proxy
```

Kubernetes - What are you?

Kubernetes project is only:

- Binaries built from source code for key kubernetes components,
- Documented state of APIs and approaches.

Not fixed implementation: this is why exists so huge amount of different ways to build (kubeadm, kops, rancher etc.) and to distribute (binaries +systemd, containers) kubernetes.

kubernetes.io doesn't say how to build your cluster - it's saying only how it should be built to be used.

This approach is following the same State-strategy of describing environemtn as kuber implementing - you are changing manifest (yaml file for some resource) and apply it to kuber - kuber by themself defines what need to be done to get the manifested state.

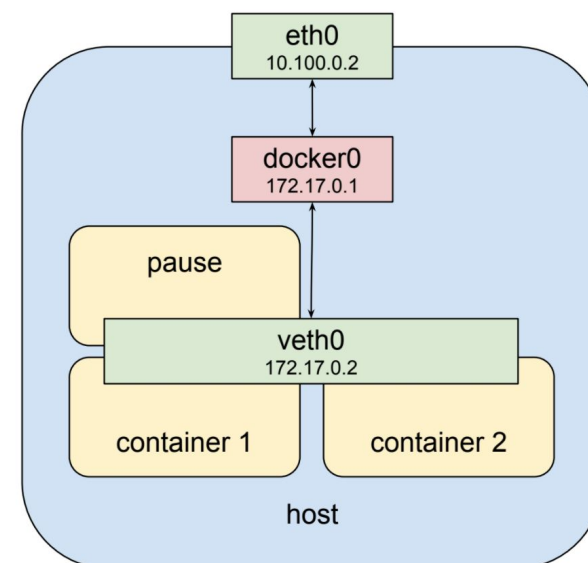
Pod - Minimal Entity Of Orchestration

- Pod - is the same networking namespace shared to 1+ docker containers.

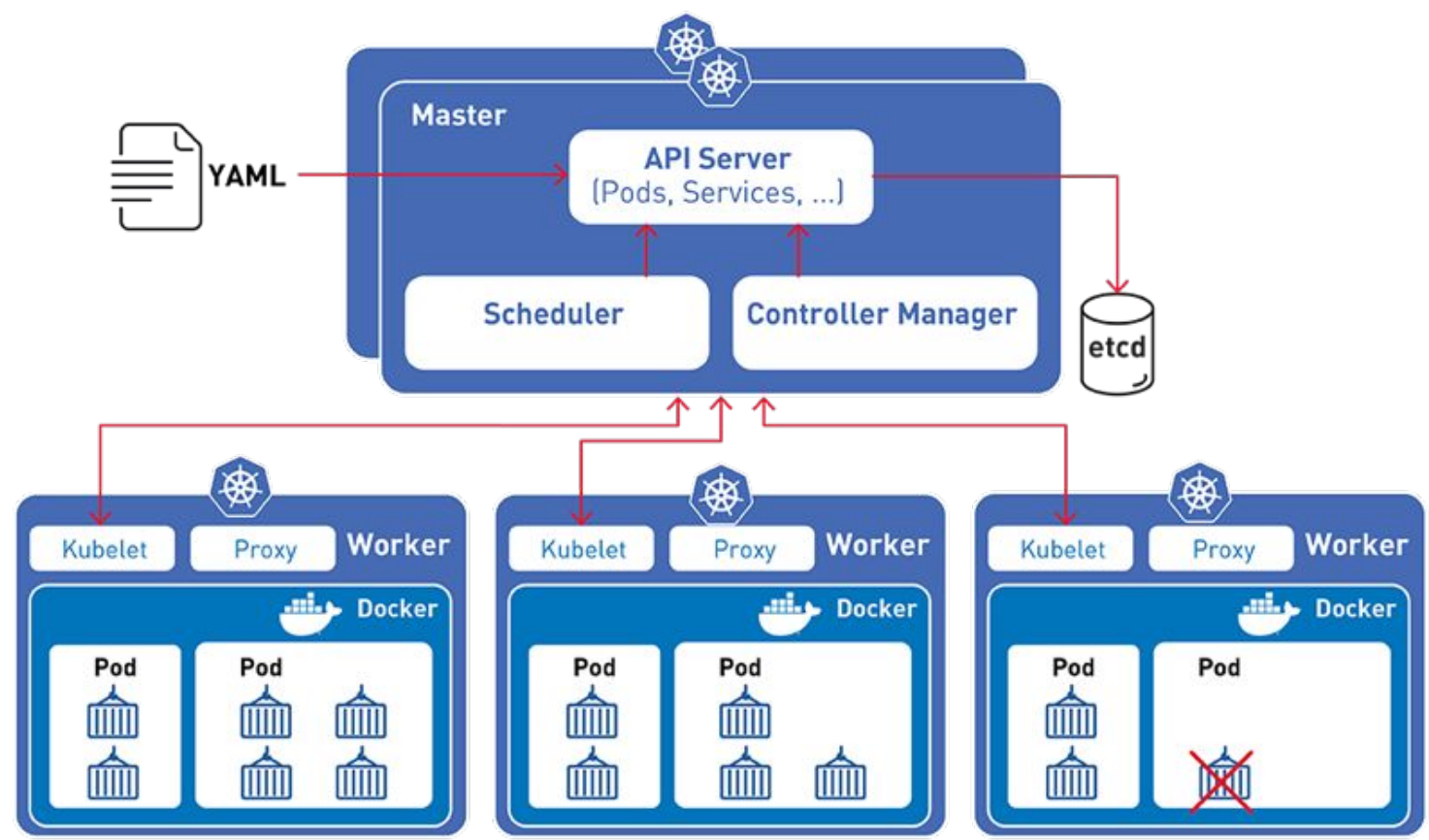
Example of Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: "games"
spec:
  containers:
    - image: docker-2048
      name: "2048"
      ports:
        - containerPort: 80
          hostPort: 8081

    - image: pengbai/docker-supermario
      name: "supermario"
      ports:
        - containerPort: 80
          hostPort: 8082
```



containerd - Docker without Docker Swarm and Compose capabilities



Pods Handling in Kuber

Pod is mortal.

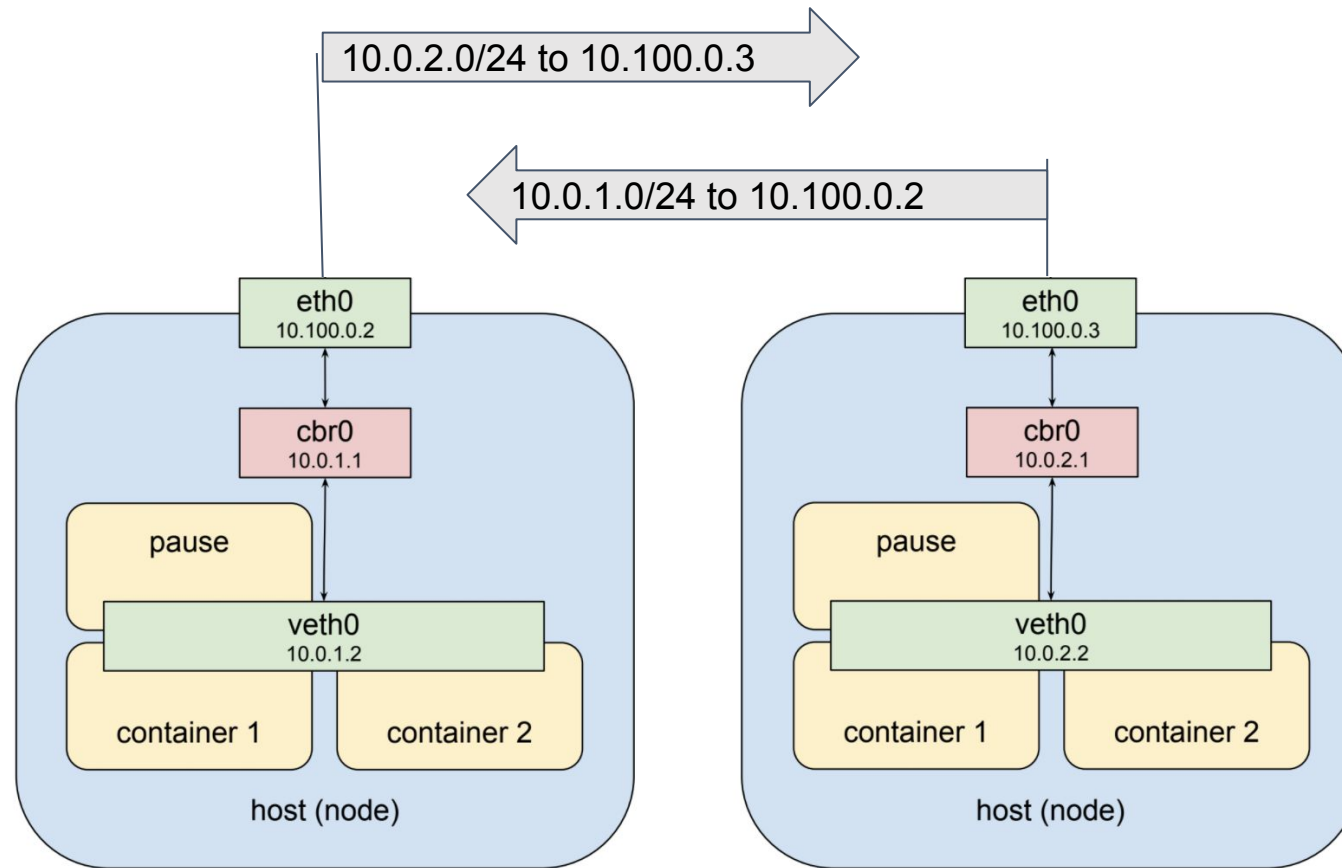
Mostly suddenly mortal.

Kuber removing pod if pod considered unhealthy (not running at least one container in POD or container is dead by predefined healthcheck).

Removed with pod containers totally lose data stored in containers.

Kubernetes - Networking

Networking mechanism between nodes and pods in nodes is not defined by kubernetes!
So choose the plugin to use.



Hands On: Build Your First Kuber Cluster

<https://vitux.com/install-and-deploy-kubernetes-on-ubuntu/>

Build your first Kuber Cluster using kubeadm.

AWS EC2 ubuntu 18.04 LTS t2.medium 2 instances same VPC, enable LAN traffic and forward port 32080 to public on both.

Firstly run `“sudo apt-get -y update”`
Reboot after node rename!

When done:
check all is fine: `ubuntu@master-node:~$ kubectl get pods --all-namespaces -o wide` - all Statuses should be Running

Deploy your first service:

1. `kubectl apply -f https://raw.githubusercontent.com/ask4ua/DKN/master/Practices/Section3/supermario.yml`
2. `kubectl describe pod -l app=supermario`
3. Web: `http://<Any node IP>:32080`

P.S.: how to control supermario: https://microsite.nintendo-europe.com/super-mario-maker-manual/enGB/page_03.html

P.P.A: Don't forget to turn off pricy VMs!

Handson: kubectl cluster - looking around

```
~$ kubectl get nodes -o wide
```

```
~$ kubectl get pods -A
```

```
~$ kubectl describe node prima
```

```
~$ kubectl describe pod <PODNAME>
```

```
~$ kubectl get pods -A -o wide
```


HandsOn: Kubernetes - Pod/Service

Last practice supermario file:

```
apiVersion: v1
kind: Service
metadata:
  name: supermario
spec:
  selector:
    app: supermario
  type: NodePort
  ports:
    - port: 8080
      nodePort: 32080
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: supermario
spec:
  replicas: 1
  selector:
    matchLabels:
      app: supermario
  template:
    metadata:
      labels:
        app: supermario
    spec:
      containers:
        - name: mario-container
          image: pengbai/docker-supermario
          ports:
            - containerPort: 8080
```

Q&A



Section 4: Kuber - dive deeper

Practice Requirements

key: sent in email.

Password: P@ss4Student[eRt

ssh to: **student##@18.195.49.2 -p 2224 -i students.key**

Handson: kubectl cluster - looking around

```
~$ kubectl get nodes -o wide
```

```
~$ kubectl describe node prima
```

```
~$ kubectl get pods --all-namespaces -o wide
```

```
~$ kubectl get namespaces
```

```
~$ kubectl create namespace student100
```

```
~$ kubectl get namespace student100
```

```
~$ kubectl get namespace student100 -o yaml
```

Handson: kubectl cluster - looking around

```
~$ kubectl get namespace student100 -o yaml
```

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    cattle.io/status:
      '{"Conditions":[{"Type":"ResourceQuotaInit","Status":"True","Message":"","LastUpdateTime":"2019-11-16T20:51:56Z"}, {"Type":"InitialRolesPopulated","Status":"True","Message":"","LastUpdateTime":"2019-11-16T20:51:56Z"}]}'
    lifecycle.cattle.io/create.namespace-auth: "true"
  creationTimestamp: "2019-11-16T20:51:54Z"
  finalizers:
  - controller.cattle.io/namespace-auth
  name: student100
  resourceVersion: "45010"
  selfLink: /api/v1/namespaces/student100
  uid: 09daed4c-b9b9-4803-89e0-ecff39b94956
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
```

```
~$ cat student99.yml
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: student99
```

```
~$ kubectl apply -f student99.yml
```

```
namespace/student99 created
```

```
~$ kubectl get namespace student99
```

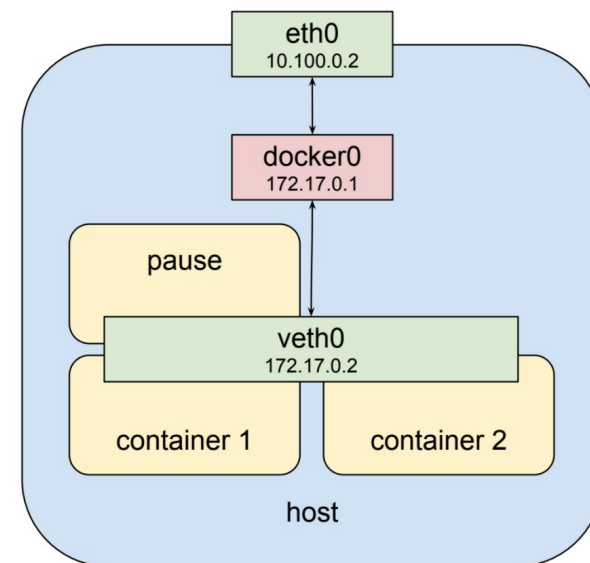
```
NAME      STATUS  AGE
student99 Active  8s
```

Pod - Minimal Entity Of Orchestration

- Pod - is the same networking namespace shared to 1+ docker containers. More looking like localhost with few containers.

Example of Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: "webapp"
spec:
  containers:
    - image: docker.ask4ua.com/webapp:latest
      name: "webapp"
  ports:
    - containerPort: 80
      hostPort: 80
```



Hands On: Starting Pod

```
~$ kubectl -n student99 apply -f db_pod.yaml
```

```
~$ kubectl -n student99 apply -f webapp_pod.yaml
```

```
~$ kubectl get pod webapp -n student99 -o wide
```

```
~$ kubectl describe pod webapp -n student99
```

...

Events:

Type	Reason	Age	From	Message
------	--------	-----	------	---------

----	-----	----	----	-----
------	-------	------	------	-------

Normal	Scheduled	<unknown>	default-scheduler	Successfully assigned student99/webapp to k8s-worker-node13
--------	-----------	-----------	-------------------	---

Normal	Pulling	2m44s	kubelet, k8s-worker-node13	Pulling image "docker.ask4ua.com/webapp"
--------	---------	-------	----------------------------	--

Normal	Pulled	2m35s	kubelet, k8s-worker-node13	Successfully pulled image "docker.ask4ua.com/webapp"
--------	--------	-------	----------------------------	--

Normal	Created	2m32s	kubelet, k8s-worker-node13	Created container webapp
--------	---------	-------	----------------------------	--------------------------

Normal	Started	2m32s	kubelet, k8s-worker-node13	Started container webapp
--------	---------	-------	----------------------------	--------------------------

```
~$ kubectl -n student99 logs -f pod/webapp
```

```
Sat Nov 16 23:44:38 2019 HTTP Server Starts
```

```
Initial connection to DB
```

```
DB ERROR: Something is wrong in connecting to DB: could not translate host name "db" to address: Name or service not known
```


POD Limitations

1. Pod is not transparently scalable - you only could create more instances of initially same pod with different names (webapp1, webapp2 etc.)
1. If Pod doesn't have more intermediate levels like replicaset/deployment - applying changes like env variables could be brought only by recreating pod (replicaset/deployment truly do the same - but transparently for user).
1. No references by DNS names - DNS names resolution is available only for services!

Hands on: Creating Services

```
apiVersion: v1
kind: Service
metadata:
  name: db
  namespace:
student99
labels:
  app: db
spec:
  selector:
    app: db
  ports:
    - port: 5432
```

```
~$ kubectl apply -f db_service.yaml
```

```
~$ kubectl apply -f webappdb_service.yaml
```

```
~$ kubectl -n student99 get services -o wide
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
db	ClusterIP	10.43.2.129	<none>	5432/TCP	13m	app=db

Entity service as a connectable socket doesn't exist - and living only as record on proxies!

Hands On - Service for Webapp

```
apiVersion: v1
kind: Service
metadata:
  name: webapp
  namespace: student99
labels:
  app: webapp
...
```

```
...
spec:
  selector:
    app: webapp
  type: NodePort
  ports:
    - port: 80
      nodePort: 32099
```

```
~$ kubectl apply -f service/webapp_service.yaml
service/webapp created
```

```
~$ kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
k8s-master-node11	Ready	controlplane,etcd	10h	v1.16.2	10.5.11.111	<none>	Ubuntu 18.04.2 LTS	4.15.0-70-generic	docker://19.3.5
k8s-master-node12	Ready	controlplane,etcd	9h	v1.16.2	10.5.11.112	<none>	Ubuntu 18.04.2 LTS	4.15.0-50-generic	docker://19.3.5
k8s-master-node13	Ready	controlplane,etcd	9h	v1.16.2	10.5.11.113	<none>	Ubuntu 18.04.2 LTS	4.15.0-50-generic	docker://19.3.5
k8s-worker-node11	Ready	worker	9h	v1.16.2	10.5.11.211	<none>	Ubuntu 18.04.2 LTS	4.15.0-70-generic	docker://19.3.5
k8s-worker-node12	Ready	worker	9h	v1.16.2	10.5.11.212	<none>	Ubuntu 18.04.2 LTS	4.15.0-50-generic	docker://19.3.5
k8s-worker-node13	Ready	worker	9h	v1.16.2	10.5.11.213	<none>	Ubuntu 18.04.2 LTS	4.15.0-50-generic	docker://19.3.5

```
student@motel:~$ curl -X GET 10.5.11.211:32099
```

Hadson: Ping service from Host

```
root@webapp:/# ping webappdb -c1
PING webappdb.student99.svc.cluster.local (10.43.169.81) 56(84) bytes of data.
From 10.5.11.1 icmp_seq=1 Time to live exceeded

--- webappdb.student99.svc.cluster.local ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

root@webapp:/# ping db -c1
PING db.student99.svc.cluster.local (10.43.2.129) 56(84) bytes of data.
From 10.5.11.1 icmp_seq=1 Time to live exceeded

--- db.student99.svc.cluster.local ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

root@webapp:/#
```

Labeling

Key mechanism of dynamic teing (choosing) of entities in kubernetes - labeling.

For example service record URL:IP requires pod IPs to load balance to them, but pod could be recreated/moved to another node any time - so service is referencing it by Labels.

```
~$ kubectl -n student99 get services -o wide
NAME      TYPE      CLUSTER-IP  EXTERNAL-IP  PORT(S)  AGE  SELECTOR
db         ClusterIP  10.43.2.129 <none>       5432/TCP  13m  app=db
webappdb   ClusterIP  10.43.169.81 <none>       5432/TCP  24s  app=db
```

Hands On: Label Selecting

```
~$ kubectl get pod -l "app=webapp"
```

NAME	READY	STATUS	RESTARTS	AGE
webapp	1/1	Running	0	5d3h
webapp-6bfbd55-gxvnc	1/1	Running	0	3d23h
webapp-6bfbd55-nqj2h	1/1	Running	0	3d23h

```
~$ kubectl get pod -l "app=db"
```

NAME	READY	STATUS	RESTARTS	AGE
db	1/1	Running	0	5d5h

Liveness and Readiness Checks

3 ways of checking POD status by kubernetes:

- by process #1 status (default)
- HTTP call
- exit code status for custom script executed on container

Hands On: Liveness Checks

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - name: webapp
...
  livenessProbe:
    httpGet:
      path: /
      port: 80
    initialDelaySeconds: 15
    periodSeconds: 10
```

```
~$ kubectl logs -f webapp
```

```
Wed Jul 8 09:12:50 2020 webapp: HTTP Server Starts
Wed Jul 8 09:12:50 2020 webapp: Initiating connection to DB
Opened new DB connection
10.5.11.102 - - [08/Jul/2020 09:13:12] "GET / HTTP/1.1" 200 -
SQL: INSERT INTO logmessages(date, logmessage) VALUES ('Wed Jul 8 09:13:12 2020', 'Accessed path "/" via server
name "webapp"');
Wed Jul 8 09:13:12 2020 webapp: Writing access fact to DB: Success
```


Liveness and Readiness Checks

- Liveness - checking is POD need to be killed?
- Readiness - checking is POD ready to be advertised via Services

Pod could be alive, but not yet ready.

No sense to have both Liveness and Readiness if they are checking same stuff.

Hands on: Creating One More DB Service

```
apiVersion: v1
kind: Service
metadata:
  name: webappdb
  namespace:
student99
labels:
  app: db
spec:
  selector:
    app: db
  ports:
    - port: 5432
```

```
apiVersion: v1
kind: Service
metadata:
  name: db
  namespace:
student99
labels:
  app: db
spec:
  selector:
    app: db
  ports:
    - port: 5432
```

```
~$ kubectl apply -f db_service.yaml
```

```
~$ kubectl apply -f webappdb_service.yaml
```

```
~$ kubectl -n student99 get services -o wide
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
db	ClusterIP	10.43.2.129	<none>	5432/TCP	13m	app=db
webappdb	ClusterIP	10.43.169.81	<none>	5432/TCP	24s	app=db

```
~$ kubectl -n student99 delete service db
```