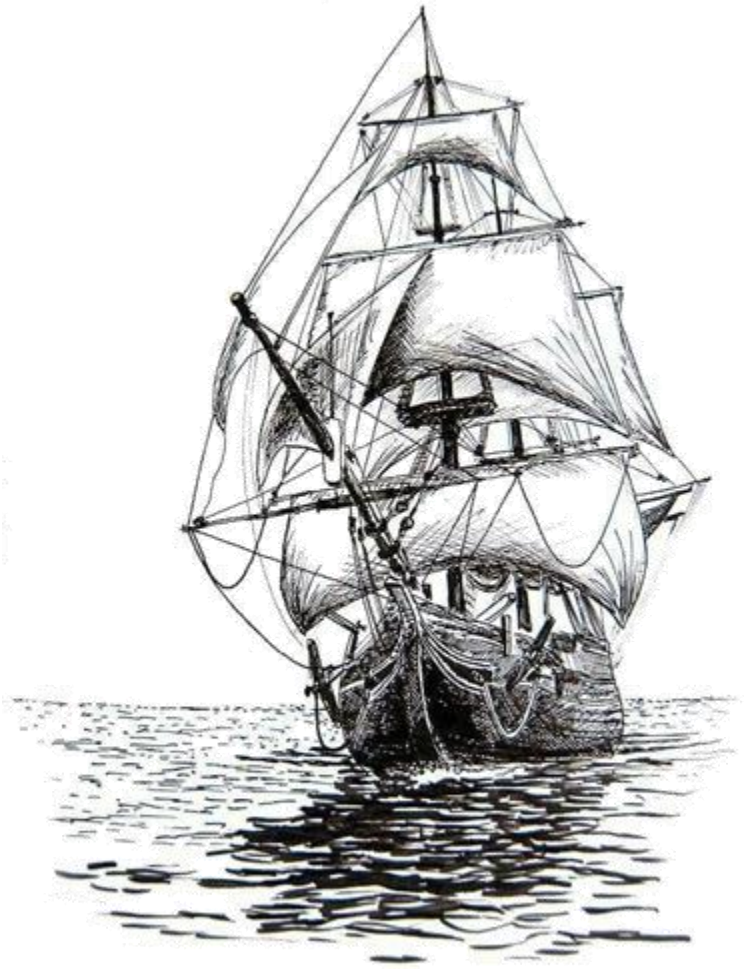
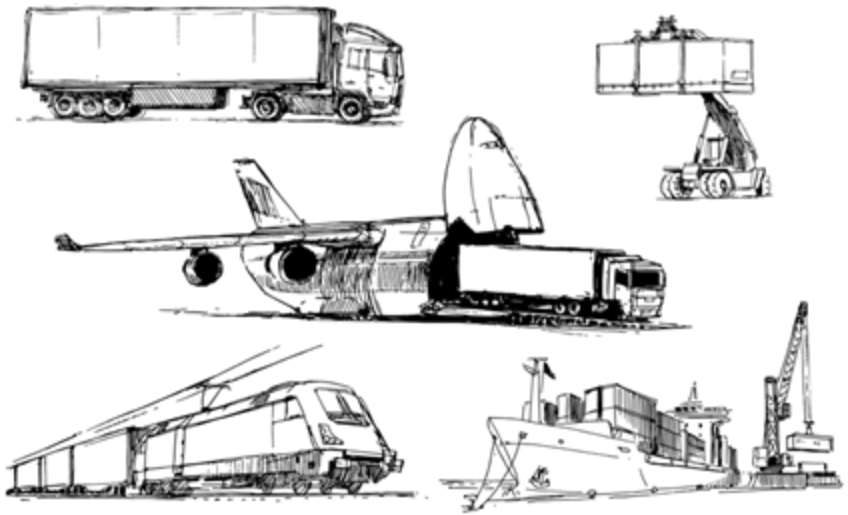

docker
for
DevOps
Engineers
2019
by
Siarhei Beliakou
part 2
LINUX



AGENDA

- Docker Volumes and Data Containers
- Getting Logs From Containers
- Docker Networks
- Docker Compose

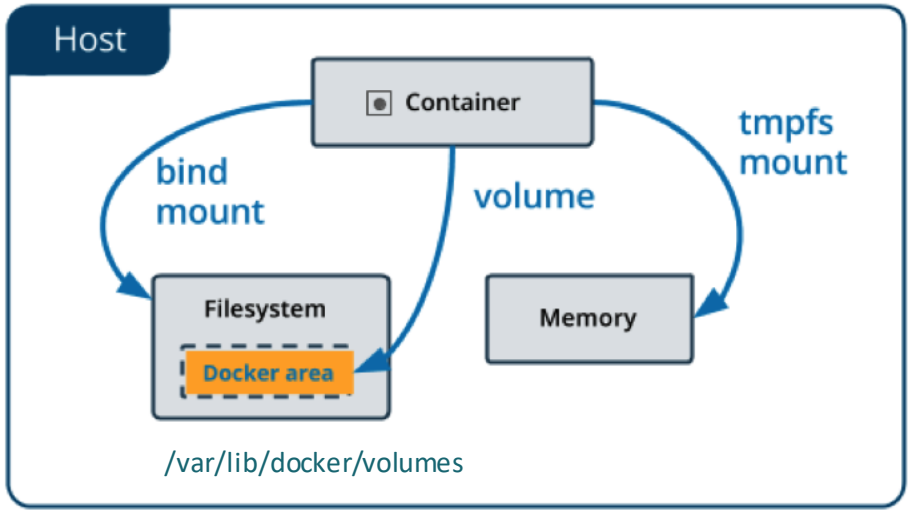


Volumes and Data Containers

- Volumes
- Data Container

Volumes

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. The volume's contents exist outside the lifecycle of a given container. If your container generates non-persistent state data, consider using a tmpfs mount to avoid storing the data anywhere permanently, and to increase the container's performance by avoiding writing into the container's writable layer.



Case #1. VOLUME in the Dockerfile

```
FROM nginx
RUN echo "default page" > /usr/share/nginx/html/index.html
VOLUME /usr/share/nginx/html/
```

```
# docker build -t nginx_v:1 .
...
Successfully built efdfe29e01fd
Successfully tagged nginx_v:1
```

Checking PRE-BUILT data:

```
# docker run -d -p80:80 nginx_v:1
```

```
# curl localhost
default page
```

Changing data:

```
# docker exec $(docker ps -lq) \
    sh -c 'echo changed page > /usr/share/nginx/html/index.html'
```

```
# curl localhost
changed page
```

Case #1. VOLUME in the Dockerfile

```
# docker stop $(docker ps -lq)
```

Where's our data?

```
# docker inspect $(docker ps -lqa) | jq '.[].Mounts'
[
  {
    "Type": "volume",
    "Name": "395c2b4639c0a577ff25b379adc201e77a1cedbc5d50e91150149e1f51191182",
    "Source": "/var/lib/docker/volumes/395c2b4639c0...f51191182/_data",
    "Destination": "/usr/share/nginx/html",
    "Driver": "local",
    "Mode": "",
    "RW": true,
    "Propagation": ""
  }
]
```

```
# cat /var/lib/docker/volumes/395c2b4639c0...f51191182/_data/index.html
changed page
```

Removing Container

```
# docker rm $(docker ps -lqa)
```

```
# cat /var/lib/docker/volumes/395c2b4639c0...f51191182/_data/index.html
changed page
```

Case #2. Creating Volume for the Container

```
# docker run -d -p 80:80 -v /usr/share/nginx/html nginx
```

```
# curl localhost
...
<title>Welcome to nginx!</title>
...
```

Looking for the Volume:

```
# docker inspect $(docker ps -lq) | jq -r '.[].Mounts[].Source'
/var/lib/docker/volumes/b6400a50ad0a0e8f11fa962cb99e7d2e425ac1654c4e4ea1376ae61a05e5dbc8/_data
```

Changing Data:

```
# echo changed > $(docker inspect $(docker ps -lq) | jq -r '.[].Mounts[].Source')/index.html
```

```
# curl localhost
changed
```

Case #3. Creating Named Volume for the Container

```
# docker run -d -p 80:80 -v nginx_data:/usr/share/nginx/html nginx
```

```
# docker volume ls
DRIVER    VOLUME NAME
local     3bfdf3679fb32d5fdfa458c00eae0ce56f5a705515294c744dc9b79b20caa033
local     766152eac9a03cf06d7b9a7da839d0cf60856a95002f50072af2b86d05606fbe
local     93ed64f4a18ab23d34b9426749cd308f92792ff9ecb47f3dd81cae322146ec05
local     9d922d9e608b3b1b1660356fd5e69514f965a80a27a3101b7300d66d679004c5
local     nginx_data
```

```
# docker inspect $(docker ps -ql) | jq '.[].Mounts[].Source'
"/var/lib/docker/volumes/nginx_data/_data"
```

```
# echo changed again > $(docker inspect $(docker ps -lq) | jq -r '.[].Mounts[].Source')/index.html
```

```
# curl localhost
changed again
```

Case #4. Sharing Data between Containers and Data Containers

Container #1

```
# docker run -d -v /usr/share/nginx/html --name c1 nginx
```

```
# echo changed > $(docker inspect c1 | jq -r '.[].Mounts[].Source')/index.html
```

Container #2

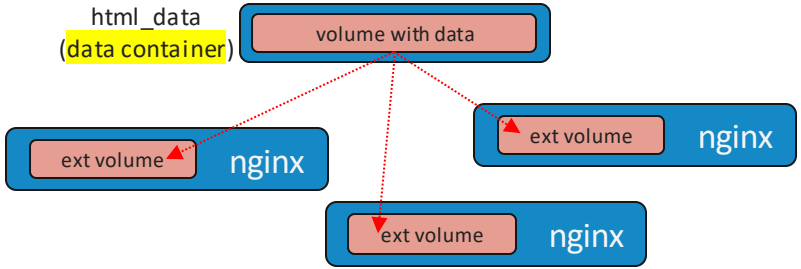
```
# docker run --volumes-from c1 busybox cat /usr/share/nginx/html/index.html
changed
```

```
# docker run --name html_data \
  -v /usr/share/nginx/html \
  busybox sh -c 'echo data page > /usr/share/nginx/html/index.html'
```

```
# docker run -d --volumes-from html_data -p 81:80 nginx
```

```
# docker run -d --volumes-from html_data -p 82:80 nginx
```

```
# docker run -d --volumes-from html_data -p 83:80 nginx
```



Docker Volumes

More details

Command	Description
docker volume create	Create a volume
docker volume inspect	Display detailed information on one or more volumes
docker volume ls	List volumes
docker volume prune	Remove all unused local volumes
docker volume rm	Remove one or more volumes

Docker Volumes

Let's create our custom Volume:

```
# docker volume create --name http-custom-data
http-custom-data
```

```
# docker volume ls | grep http-custom-data
local               http-custom-data
```

```
# docker volume inspect http-custom-data
[
    {
        "CreatedAt": "2018-07-29T21:30:35+01:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/http-custom-data/_data",
        "Name": "http-custom-data",
        "Options": {},
        "Scope": "local"
    }
]
```

Docker Volumes

index.html

```
my custom page from Volume
```

Now we need to copy assets:

```
# cp index.html /var/lib/docker/volumes/http-custom-data/_data/

# ls -l /var/lib/docker/volumes/http-custom-data/_data/
total 4
-rw-r--r-- 1 root root 28 Oct 11 20:40 index.html
```

And start our container of nginx:

```
# docker run -d -P -v http-custom-data:/usr/share/nginx/html nginx
b94feb29c143eea7900965706447151e96df86539312bdcea79b42952bae701a

# docker port $(docker ps -lq)
80/tcp -> 0.0.0.0:32769
```

Let's check:

```
# curl 127.0.0.1:32769
my custom page from Volume
```

Docker Volumes

Creating TMPFS Volume (in RAM):

```
# docker volume create --driver local \
  --opt type=tmpfs \
  --opt device=tmpfs \
  --opt o=size=100m,uid=1000 \
  foo
```

Creating BTRFS Volume (on /dev/sda2 partition):

```
# docker volume create --driver local \
  --opt type=btrfs \
  --opt device=/dev/sda2 \
  foo
```

Creating NFS Volume (in Remote NFS Share):

```
# docker volume create --driver local \
  --opt type=nfs \
  --opt o=addr=192.168.1.1,rw \
  --opt device=:/path/to/dir \
  foo
```

Removing Volumes

Getting dangling (not used by existing containers) volumes:

```
# docker volume ls -f dangling=true
DRIVER              VOLUME NAME
local               0d16eec9bbe705afd0144adca0425173b25bc1886b1421c7d15def5fa65a34cf
local               395c2b4639c0a577ff25b379adc201e77a1cedbc5d50e91150149e1f51191182
local               923d3fa1a0d1d395cc86f9c7d316d0723c492ebb6e69555c876cf39b78c9c0ea
```

Remove given volume:

```
# docker volume rm 0d16eec9bbe705afd0144adca04...c7d15def5fa65a34cf
```

Remove all unused volumes:

```
# docker volume prune
WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
395c2b4639c0a577ff25b379adc201e77a1cedbc5d50e91150149e1f51191182
923d3fa1a0d1d395cc86f9c7d316d0723c492ebb6e69555c876cf39b78c9c0ea

Total reclaimed space: 1.019GB
```

Getting Logs from Containers

- In-container Logs
- Container Log Drivers

Container Logs

By default, **docker logs** shows the command's **STDOUT** and **STDERR**

```
# docker run hello-world
# docker logs $(docker ps -aql)
```

Let's check container from *myhttpd:latest* image:

```
# docker run -d -P myhttpd:latest
791c65b5aed05a11a40a953779675b5b94d090e691730c5c0bceaa33143fcbc4

# docker logs $(docker ps -lq)
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
using 172.17.0.17. Set the 'ServerName' directive globally to suppress this message
```

```
# curl localhost:$(docker port $(docker ps -lq) | cut -d: -f2)
my httpd container

# docker logs $(docker ps -lq)
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.17.0.17. Set the 'ServerName' directive globally to suppress this message
```

How Do They Configure This Usually?

```
# docker run nginx ls -l /var/log/nginx
total 0
lrwxrwxrwx 1 root root 11 Oct  2 19:20 access.log -> /dev/stdout
lrwxrwxrwx 1 root root 11 Oct  2 19:20 error.log -> /dev/stderr
```

The official nginx image creates a symbolic link from */var/log/nginx/access.log* to */dev/stdout*, and creates another symbolic link from */var/log/nginx/error.log* to */dev/stderr*, overwriting the log files and causing logs to be sent to the relevant special device instead. See the Dockerfile.

```
# docker run httpd cat conf/httpd.conf | egrep '^ *(Error|Custom)Log'
ErrorLog /proc/self/fd/2
    CustomLog /proc/self/fd/1 common
```

The official httpd driver changes the httpd application's configuration to write its normal output directly to */proc/self/fd/1* (which is STDOUT) and its errors to /proc/self/fd/2 (which is STDERR). See the Dockerfile.

Container Logs

Let's create "Dockerfile" file with following content

```
FROM centos
LABEL maintainer="Siarhei Beliakou"
RUN yum install -y httpd web-assets-httpd && yum clean all
RUN echo "logs are sending to stdout" > /var/www/html/index.html

RUN ln -s /dev/stdout /var/log/httpd/access_log && \
    ln -s /dev/stderr /var/log/httpd/error_log
EXPOSE 80
CMD httpd -DFOREGROUND
```

```
FROM myhttpd:1.0
RUN echo "logs are sending to stdout" > /var/www/html/index.html
RUN ln -s /dev/stdout /var/log/httpd/access_log && \
    ln -s /dev/stderr /var/log/httpd/error_log
```

And build it:

```
# docker build -t myhttpd:2.0 .
...
Successfully built 5ffd0ffc1780
Successfully tagged myhttpd:2.0
```

Container Logs

```
# docker run -d -P myhttpd:2.0
1b0c3a82d0687519ffcd2ee06d345a4d3ad8d475dc0d7710fb279bdd836e5e88

# docker logs $(docker ps -lq)
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
using 172.17.0.17. Set the 'ServerName' directive globally to suppress this message
```

```
# curl localhost:$(docker port $(docker ps -lq) | cut -d: -f2)
logs are sending to stdout

# docker logs $(docker ps -lq)
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.17.0.18. Set the 'ServerName' directive globally to suppress this message
172.17.0.1 - - [29/Jul/2018:22:07:24 +0000] "GET / HTTP/1.1" 200 19 "-" "curl/7.29.0"
```

Configuring Log Driver

Run Container with the name and log driver:

```
# docker run -d -P --name=myhttpd --log-driver=journald myhttpd:2.0
5748f3078b647c43a335bdc1f8bc7c8d9db31a491e635effd58b31b1429c8ee7

# curl localhost:$(docker port $(docker ps -lq) | cut -d: -f2)
logs are sending to stdout
```

Get Container's logs from journald by CONTAINER_NAME:

```
# journalctl -b CONTAINER_NAME=myhttpd
-- Logs begin at Sat 2018-07-28 19:14:07 BST, end at Sun 2018-07-29 23:19:52 BST. --
Jul 29 23:19:29 localhost.localdomain 5748f3078b64[2806]: AH00558: httpd: Could not reliably
determine the server's fully qualified domain name, using 172.17.
Jul 29 23:19:50 localhost.localdomain 5748f3078b64[2806]: 172.17.0.1 - - [29/Jul/2018:22:19:50
+0000] "GET / HTTP/1.1" 200 19 "-" "curl/7.29.0"
```

Configuring Log Driver

Run Container with log driver and log tag:

```
# docker run -d -P --log-driver=journald --log-opt tag=myhttpd myhttpd:latest
0555d7a41ab6af098dfea296e2fa7b4f1c6b73424e2156c387930b13bfcefb24

# curl localhost:$(docker port $(docker ps -lq) | cut -d: -f2)
logs are sending to stdout
```

Get Container's logs from journald by CONTAINER_TAG:

```
# journalctl -b CONTAINER_TAG=myhttpd
-- Logs begin at Sat 2018-07-28 19:14:07 BST, end at Sun 2018-07-29 23:27:27 BST. --
Jul 29 23:26:02 localhost.localdomain myhttpd[2806]: AH00558: httpd: Could not reliably determine
the server's fully qualified domain name, using 172.17.0.21.
Jul 29 23:26:26 localhost.localdomain myhttpd[2806]: 172.17.0.1 - - [29/Jul/2018:22:26:26 +0000]
"GET / HTTP/1.1" 200 19 "-" "curl/7.29.0"
```

Supported Log Drivers

Driver	Description
none	No logs are available for the container and docker logs does not return any output.
json-file	The logs are formatted as JSON. The default logging driver for Docker.
syslog	Writes logging messages to the syslog facility. The syslog daemon must be running on the host machine.
journald	Writes log messages to journald. The journald daemon must be running on the host machine.
gelf	Writes log messages to a Graylog Extended Log Format (GELF) endpoint such as Graylog or Logstash.
fluentd	Writes log messages to fluentd (forward input). The fluentd daemon must be running on the host machine.
awslogs	Writes log messages to Amazon CloudWatch Logs.
splunk	Writes log messages to splunk using the HTTP Event Collector.
etwlogs	Writes log messages as Event Tracing for Windows (ETW) events. Only available on Windows platforms.
gcplogs	Writes log messages to Google Cloud Platform (GCP) Logging.
logentries	Writes log messages to Rapid7 Logentries.

Dockerd: Configuring Log Driver

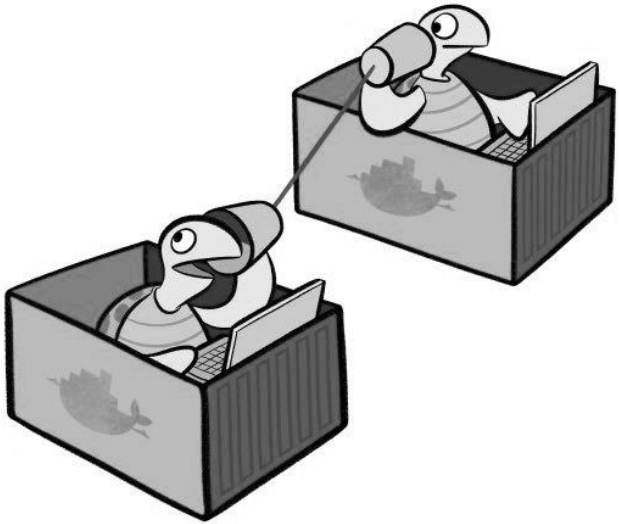
```
# docker info --format '{{.LoggingDriver}}'
json-file

# cat << EOF > /etc/docker/daemon.json
{
  "log-driver": "journald"
}
EOF

# systemctl daemon-reload
# systemctl restart docker.service
# docker info --format '{{.LoggingDriver}}'
journald
```

```
# docker run -d -P --name=myweb --log-opt tag=myweb_tag myhttpd:2.0
```

```
# journalctl -b CONTAINER_NAME=myweb
```

```
# journalctl -b CONTAINER_TAG=myweb_tag
```

https://docs.docker.com/config/containers/logging/log_tags/



Docker Networks

- **Network Drivers**
- **Default Networks:**
  - None
  - Host
  - Bridge
- **User Defined Bridge Networks**
- **Attaching Container to Different Networks**
- **IPAM**

Working with Networks

Command	Description
`docker network connect`	Connect a container to a network
`docker network create`	Create a network
`docker network disconnect`	Disconnect a container from a network
`docker network inspect`	Display detailed information on one or more networks
`docker network ls`	List networks
`docker network prune`	Remove all unused networks
`docker network rm`	Remove one or more networks

Network Drivers

Docker's networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

```
# docker info --format '{{json .Plugins.Network}}' | jq
```

- **bridge**: The default network driver. Bridge networks are usually used when your applications run in standalone containers that need to communicate.
- **host**: For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly.
- **overlay**: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other.
- **macvlan**: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack.
- **null**: For this container, disable all networking. Usually used in conjunction with a custom network driver.

Docker Default Networks

```
# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
166e19aea71a        bridge              bridge              local
ca70f5e8014e        host                host                local
0db73a8bc0fd        none                null                local
```

none, **host** and **bridge** are the names of default networks that you can find in any Docker installation running a single host. The activation of the *swarm* mode creates another default (or redefined) network called *ingress*.

NONE Network

The network called **none** using the null driver is a predefined network that isolates a container in a way it can neither connect to outside not communicate with other containers in the same host.

```
# docker run --net=none -d --name inNoneContainer busybox
# docker inspect inNoneContainer | jq '.[].NetworkSettings.Networks'
{
  "none": {
    "IPAMConfig": null,
    "Links": null,
    "Aliases": null,
    "NetworkID": "0db73a8bc0fd6de8e25012b144e9a26b47959236c1edfcca45e86535a25d84b1",
    "EndpointID": "4b96675ae873b5d2b477c117d1e8a0681c19e305c674633863f731e955ef3b58",
    "Gateway": "",
    "IPAddress": "",
    "IPPrefixLen": 0,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "MacAddress": "",
    "DriverOpts": null
  }
}

# docker run --network=none busybox ping google.com
ping: google.com: Name or service not known
```

HOST Network

If you want a container to run with a similar networking configuration to the host machine, then you should use the host network.

```
# docker network inspect host

# docker run -d --network=host --name=nginx nginx
491a4625252479a74529210828d432e37a2a97cf07e586e740bfd4e34c60e543

# docker ps -l
CONTAINER ID    IMAGE    COMMAND                  CREATED         STATUS         PORTS      NAMES
491a46252524    nginx    "nginx -g 'daemon of…"   3 seconds ago   Up 2 seconds               nginx

# docker port nginx

# netstat -natpl | grep :80
tcp    0   0 0.0.0.0:80        0.0.0.0:*    LISTEN      25419/nginx: master

# curl -sIL localhost | egrep "(HTTP|Server)"
HTTP/1.1 200 OK
Server: nginx/1.15.2
```

BRIDGE Network

This is the default containers network, any network that runs without the --net flag will be attached automatically to this network. Two Docker containers running in this network could see each other.

```
# docker run -d -it --name=my_container_1 busybox
# docker run -d -it --name=my_container_2 busybox
# docker network inspect bridge | jq '.[].Containers'
{
  "51d7530d62a0fe8c510c82bfafd978a34e9f9af90f2f8b20df718b2b83675f05": {
    "Name": "my_container_1",
    "EndpointID": "8fe1263ee191757e4ba15439b6b0d6a78e871dfddba03f6b952cc01b5aded4c3",
    "MacAddress": "02:42:ac:11:00:02",
    "IPv4Address": "172.17.0.2/16",
    "IPv6Address": ""
  },
  "30810e27de245c7a36f5327b53b615fcde01f0a4082a5238506bbab3dafd1eaa": {
    "Name": "my_container_2",
    "EndpointID": "806d045bd432a02be8e005c8e2b3299bf883274269bb0078b4e5401a17e5cd64",
    "MacAddress": "02:42:ac:11:00:03",
    "IPv4Address": "172.17.0.3/16",
    "IPv6Address": ""
  }
}
```

BRIDGE Network

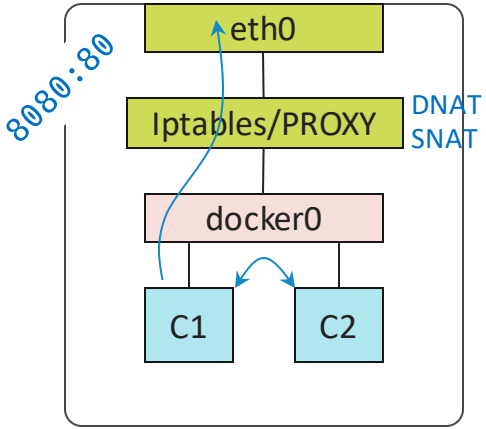
This is the default containers network, any network that runs without the --net flag will be attached automatically to this network. Two Docker containers running in this network could see each other.

```
# docker exec my_container_1 ping -c1 172.17.0.3
PING 172.17.0.3 (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.096 ms

--- 172.17.0.3 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.096/0.096/0.096 ms

# docker exec my_container_1 ping my_container_2
ping: bad address 'my_container_2'
```

As you see, it is not possible for Docker to associate a container name to an IP and this is not possible because we do not run any *discovery service*. Creating a *user-defined* network could solve this problem.

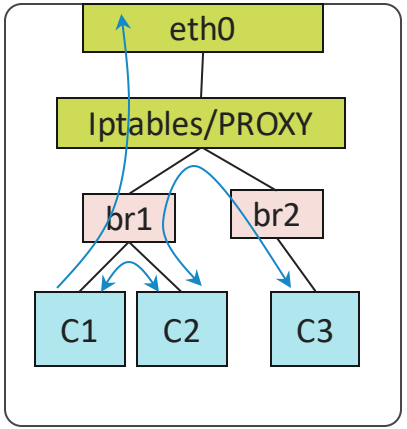
Docker Networking

Default Docker Bridge



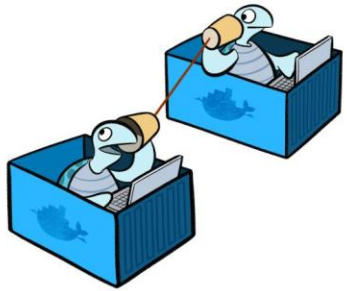
C1 ↔ C2 by IP

User Defined Bridge



C1 ↔ C2 by IP and Names

C2 ↔ C3 by IP



Worth reading: https://blog.docker.com/2016/12/understanding-docker-networking-drivers-use-cases/

User Defined Bridge

This type of networks, in opposite to the default networks, does not come with a fresh Docker installation, but should be created by the user

```
# docker network create < network_name >
# docker network create <options> <network>
# docker network create --help
```

Let's create a network:

```
# docker network create --driver bridge my_bridge_network
b8dcaf29c54adb165c07487b2061e7c246b3d405b496d8a7d9337bb4a766e300

# docker network ls | grep my_bridge
b8dcaf29c54a        my_bridge_network   bridge              local

# ifconfig | sed -n '/b8dcaf29c54a/,/^$/p'
br-b8dcaf29c54a: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.20.255.255
        inet6 fe80::42:e1ff:fe21:e27f  prefixlen 64  scopeid 0x20<link>
        ether 02:42:e1:21:e2:7f  txqueuelen 0  (Ethernet)
        RX packets 6  bytes 616 (616.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 20  bytes 1383 (1.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Intercommunication

Let's try to run containers in just created "my_bridge_network" docker network:

```
# docker run -d --name=inmybridge1 --net=my_bridge_network centos sleep infinity
7e14e1b47fceb1a68b77e77c0899aa13acc21867f01866629203809cb93bd4b8
```

```
# docker run -d --name=inmybridge2 --net=my_bridge_network centos sleep infinity
f53469ddeb739c68ebdc9ec34151a63fcfe2f80d099d78535fcc71db53cb8720
```

```
# docker exec inmybridge1 ping -c1 inmybridge2
PING inmybridge2 (172.18.0.3) 56(84) bytes of data.
64 bytes from inmybridge2.my_bridge_network (172.18.0.3): icmp_seq=1 ttl=64 time=0.069 ms

--- inmybridge2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.069/0.069/0.069/0.000 ms
```

Attaching Container to Custom Bridge Network

my_container_1 and ***my_container_2*** are running in the default bridge network, we want them attached to the new network, we should do:

```
# docker network connect my_bridge_network my_container_1
# docker network connect my_bridge_network my_container_2
# docker network inspect my_bridge_network | jq '.[].Containers'
{
  "30810e27de245c7a36f5327b53b615fcde01f0a4082a5238506bbab3dafd1eaa": {
    "Name": "my_container_2",
    "EndpointID": "1973d9e1b3fb406323e57b7b2f094f5def7b83860fb966d56b9a6040b3b3e33d",
    "MacAddress": "02:42:ac:13:00:03",
    "IPv4Address": "172.18.0.3/16",
    "IPv6Address": ""
  },
  "51d7530d62a0fe8c510c82bfafd978a34e9f9af90f2f8b20df718b2b83675f05": {
    "Name": "my_container_1",
    "EndpointID": "c5fdcc95605fb55272ab876fcab5174c37586eb6e0fe4962dc0bcc507a6c8b02",
    "MacAddress": "02:42:ac:13:00:02",
    "IPv4Address": "172.18.0.2/16",
    "IPv6Address": ""
  }
}
```

Attaching Container to the Custom Bridge Network

Now, both containers know about each other by their names:

```
# docker exec my_container_1 ping -c1 my_container_2
PING my_container_2 (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.054 ms

--- my_container_2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.054/0.054/0.054 ms
```

Docker Networking: IPAM

Let's create a more personalized network with a specified subnet, gateway and IP range and let's also change the behavior of networking in this network by decreasing the size of the largest network layer protocol data unit that can be communicated in a single network transaction, or what we call MTU (Maximum Transmission Unit):

```
# docker network create -d bridge \
    --subnet=172.25.0.0/16 \
    --gateway=172.25.0.1 \
    --ip-range=172.25.1.0/24 \
    --opt "com.docker.network.driver.mtu"="1000" \
    my_custom_network
```

```
# docker inspect my_custom_network | jq '.[].IPAM'
{
  "Driver": "default",
  "Options": {},
  "Config": [
    {
      "Subnet": "172.25.0.0/16",
      "IPRange": "172.25.1.0/24",
      "Gateway": "172.25.0.1"
    }
  ]
}
```

Run Container with Custom IP Address (User Defined Networks Only)

Create a new bridge network with your subnet and gateway for your ip block

```
# docker network create --subnet 203.0.113.0/24 --gateway 203.0.113.254 custom_net
```

Run a nginx container with a specific ip in that block

```
# docker run -d --net custom_net --ip 203.0.113.2 nginx
```

Curl the ip now

```
# curl 203.0.113.2
```

Legacy Container Links

Run named container

```
# docker run -d --name=web nginx
```

Getting direct link to the container:

```
# docker run --link web:web_service tutum/curl curl -sIL web_service
HTTP/1.1 200 OK
Server: nginx/1.15.5
...
```

How does this work?

```
# docker run --link web:web_service busybox cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0         ip6-localnet
ff00::0         ip6-mcastprefix
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
172.17.0.2      web_service 5b1da5627785 web
172.17.0.3      5bb1bae7b901
```

Legacy Container Links

Syntax:

```
--link << name or id >>:alias
--link << name or id >>
```

```
# docker run -d --name db training/postgres
```

the same

```
# docker run -d -P --name web --link db:db training/webapp python app.py
```

```
# docker run -d -P --name web --link db training/webapp python app.py
```

More Details here

Legacy Container Links

Run Container with env variable(s):

```
# docker run -dit -e MYVAR=VARIABLE1 --name c1 busybox
```

Checking that it's running:

```
# docker ps -l --format "{{.Image}}\t{{.ID}}\t{{.Names}}\t{{.Status}}"
busybox      ee2f01530898   c1     Up 3 minutes
```

Grabbing Env Variable(s) from "c1" Container:

```
# docker run --link c1 busybox env | grep MYVAR
C1_ENV_MYVAR=VARIABLE1
```

Docker-Compose

- Commands
- Docker Compose File
- Managing Containers with Docker Compose
- Customizing Container Settings
- Building Images with Docker Compose

What docker-compose is

Compose is a tool for defining and running multi-container Docker applications. The Compose file is a YAML file defining services, networks and volumes.
The default path for a Compose file is ***./docker-compose.yml***
To learn more about all the features of Compose please take a look at the list of features.

Use Cases:

- Development environments
- Automated testing environments
- Single host deployments

Installing:

```
# pip install docker-compose
# docker-compose --version
docker-compose version 1.22.0, build f46880f
```

- https://docs.docker.com/compose/compose-file/
- https://docs.docker.com/compose/overview/
- https://medium.freecodecamp.org/the-ups-and-downs-of-docker-compose-how-to-run-multi-container-applications-bf7a8e33017e

Docker-compose Commands

```
$ docker-compose << command >> ...
```

Lifecycle:

create	-	Create services
start	-	Start services
stop	-	Stop services
restart	-	Restart services
kill	-	Kill containers
pause	-	Pause services
unpause	-	Unpause services
up [-d]	-	Create and start containers [detached mode]
down	-	Stop and remove containers, networks, images, and volumes

Info, Logs:

ps	-	List containers
port	-	Print the public port for a port binding
top	-	Display the running processes
logs	-	View output from containers

Build and Validate:

build	-	Build or rebuild services
config	-	Validate and view the Compose file

Working with Registry:

pull	-	Pull service images
push	-	Push service images

Docker-compose file

Generally, docker-compose file may consist of following parts:

```
version:
networks:
services:
volumes:
configs:
```

Traditional command:

```
$ docker run -d --restart always -e MYSQL_ROOT_PASSWORD=password mariadb
```

docker-compose.yml

```
version: "2"
services:
  mariadb:
    image: mariadb
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: password
```

Managing Containers with Docker-compose

```
[vagrant@docker-host project_dir]$ docker-compose up -d
Creating network "project_dir_default" with the default driver
Creating project_dir_mariadb_1 ... done
```

```
[vagrant@docker-host project_dir]$ docker-compose ps
         Name                       Command             State    Ports
-------------------------------------------------------------------------
project_dir_mariadb_1   docker-entrypoint.sh mysqld   Up      3306/tcp
```

```
$ docker ps
CONTAINER ID  IMAGE    COMMAND     CREATED         STATUS         PORTS     NAMES
6c71eea413eb  mariadb  "docker-…"  29 seconds ago  Up 28 seconds  3306/tcp  project_dir_mariadb_1
```

```
$ docker-compose exec mariadb mysqladmin -ppassword version
mysqladmin  Ver 9.1 Distrib 10.3.8-MariaDB, for debian-linux-gnu on x86_64
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Server version       10.3.8-MariaDB-1:10.3.8+maria~bionic
Protocol version     10
Connection           Localhost via UNIX socket
UNIX socket          /var/run/mysqld/mysqld.sock
Uptime:              7 min 31 sec

Threads: 7  Questions: 5  Slow queries: 0  Opens: 17  Flush tables: 1  Open tables: 11  Queries per second avg: 0.011
```

Managing Containers with Docker-compose

```
[vagrant@docker-host project_dir]$ docker-compose images
      Container         Repository    Tag       Image Id      Size
-----------------------------------------------------------------------
project_dir_mariadb_1   mariadb      latest   2c73b3262fff   346 MB
```

```
[vagrant@docker-host project_dir]$ docker-compose logs mariadb
Attaching to project_dir_mariadb_1
mariadb_1  | Initializing database
...
mariadb_1  | Database initialized
mariadb_1  | MySQL init process in progress...
mariadb_1  | 2018-08-07 11:36:10 0 [Note] mysqld (mysqld 10.3.8-MariaDB-1:10.3.8+maria~bionic)
starting as process 101 ...
mariadb_1  | 2018-08-07 11:36:10 0 [Note] InnoDB: Using Linux native AIO
```

```
[vagrant@docker-host project_dir]$ docker-compose restart mariadb
Restarting project_dir_mariadb_1 ... done
```

```
[vagrant@docker-host project_dir]$ docker-compose stop mariadb
Stopping project_dir_mariadb_1 ... done
```

```
[vagrant@docker-host project_dir]$ docker-compose down
Removing project_dir_mariadb_1 ... done
Removing network project_dir_default
```

Specifying Container Name

```
$ docker ps
CONTAINER ID  IMAGE    COMMAND     CREATED          STATUS         PORTS     NAMES
6c71eea413eb  mariadb  "docker-…"  29 seconds ago   Up 28 seconds  3306/tcp  project_dir_mariadb_1
```

```
version: "2"
services:
  mariadb:
    image: mariadb
    container_name: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: password
```

```
version: "2"
services:
  mariadb:
    image: mariadb
    container_name: mariadb
    environment:
      - MYSQL_ROOT_PASSWORD=password
```

```
$ docker-compose up -d
Creating network "project_dir_default" with the default driver
Creating mariadb ... done
```

```
$ $ docker ps
CONTAINER ID  IMAGE    COMMAND     CREATED          STATUS         PORTS     NAMES
f9c0bec8b381  mariadb  "docker-…"  38 seconds ago   Up 37 seconds  3306/tcp  mariadb
```

Env Variables, Volumes, Ports, Logging Options Ulimits

```
version: "3"
services:
  jenkins:
    image: jenkins
    environment:
      - HOME=/opt/jenkins/home
      - JENKINS_HOME=/opt/jenkins/home
    env_file:
      - ./env_file
    ports:
      - "127.0.0.1:8080:8080"
      - "127.0.0.1:50000:50000/tcp"
    volumes:
      - /opt/jenkins/home:/opt/jenkins/home
    logging:
      options:
        tag: jenkins
    ulimits:
      nproc: 65535
      nofile:
        soft: 20000
        hard: 40000
```

Networks and Network Modes in Docker-Compose

Networks to join, referencing entries under the top-level networks key

```
services:
  some-service:
    networks:
      - some-network
      - other-network
  another-service:
    networks:
      - other-network

networks:
  frontend:
    name: some-network
    driver: custom-driver-1
  backend:
    name: other-network
    driver: custom-driver-2
```

Network mode. Use the same values as the docker client ***--network*** parameter, plus the special form ***service:[service name]***

```
services:
  some-service:
    network_mode: bridge

  host-placed-service:
    network_mode: host

  none-placed-service:
    network_mode: none

  service-in-other-service-net:
    network_mode: service:[service-name]

  service-in-other-container-net:
    network_mode: container:[container-name/id]
```

User Defined Networks

```
version: "3.3"

services:
  some-service:
    image: centos
    container_name: some-service
    command: sleep infinity
    networks:
    - backtier

networks:
  backtier:
    driver: bridge
    ipam:
      config:
        - subnet: 172.126.0.0/16
```

```
version: "2.3"

services:
  some-service:
    image: centos
    command: sleep infinity
    networks:
      mynet:
        ipv4_address: 172.21.0.2

networks:
  mynet:
    driver: bridge
    ipam:
      driver: default
      config:
      - subnet: 172.21.0.0/16
        gateway: 172.21.0.1
```

Working with Networks

Pay attention to Compose Version! This doesn't work on the latest one.

```
version: "???"

services:
  infinity:
   image: centos
   command: sleep infinity
   networks:
     mynet:
       ipv4_address: 172.155.0.2

networks:
  mynet:
    driver: bridge
    ipam:
      driver: default
      config:
      - subnet: 172.155.0.0/16
        gateway: 172.155.0.1
```

```
# 2.3
Creating network "mynet" with driver "bridge"
Creating infinity_1 ... done

# 3.3
ERROR: The Compose file is invalid because:
networks.mynet value Additional properties are
not allowed ('enable_ipv6' was unexpected)
networks.mynet.ipam.config value Additional
properties are not allowed ('gateway' was
unexpected)

# unspecified
ERROR: The Compose file is invalid because:
Unsupported config option for services:
'infinity'
Unsupported config option for networks:
'mynet'
```

Using External Networks

```
$ docker network create tools-docker-network
```

```
version: "3"

services:
  webservice:
   image: nginx

networks:
  default:
    external:
      name: tools-docker-network
```

Using Volumes in Docker-Compose

```
version: "3"
services:
  mariadb1:
    image: mariadb
    container_name: mariadb1
    volumes:
      - mariadb_storage:/var/lib/mysql
      - ./:/data/folder/in/container/
      - /full/path:/mountpoint/in/container
    environment:
      MYSQL_ROOT_PASSWORD: password

volumes:
  mariadb_storage:
```

Building Images with Docker Compose

frontend.Dockerfile

```
FROM httpd
LABEL maintainer="Siarhei  Beliakou (sbeliakou@gmail.com)"
```

backend.Dockerfile

```
FROM tomcat
LABEL maintainer="Siarhei  Beliakou (sbeliakou@gmail.com)"
```

docker-compose.yml

```
version: "3"

services:
  frontend:
    build:
      context: .
      dockerfile: frontend.Dockerfile

  backend:
    build:
      context: .
      dockerfile: backend.Dockerfile
```

Building Images with Docker Compose

The following only builds the images, does not start the containers:

```
# docker-compose build
```

The following builds the images if the images **do not exist** and starts the containers

```
# docker-compose up -d
```

If you add the ***--build*** option, it is forced to build the images even when not needed:

```
# docker-compose up -d --build
```

The following skips the image build process:

```
# docker-compose up -d --no-build
```

The ***--no-cache*** option disables the Docker build cache in the image creation process. This is used to cache each layer in the ***Dockerfile*** and to speed up the image creation reusing layers (~ Dockerfile lines) previously built for other images that are identical.

to be continued

Thank you for your attention!

Siarhei Beliakou,

2019