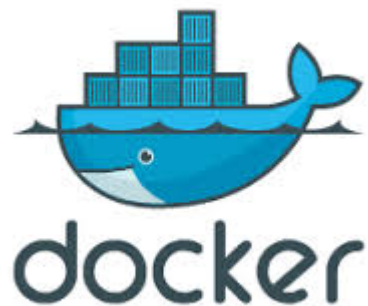


Introduction to Docker Core Concepts

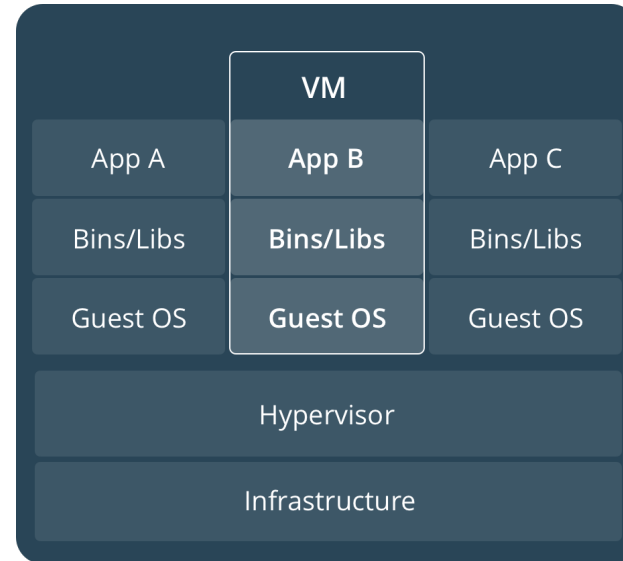
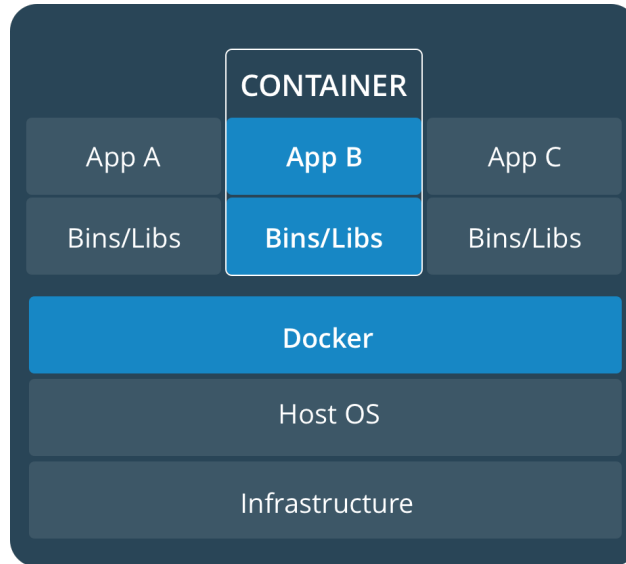


This Document:

<http://arnaud-nauwynck.github.io/docs/Intro-Docker.pdf>

Lightweight Containers ↔ Virtual Machines

Containers



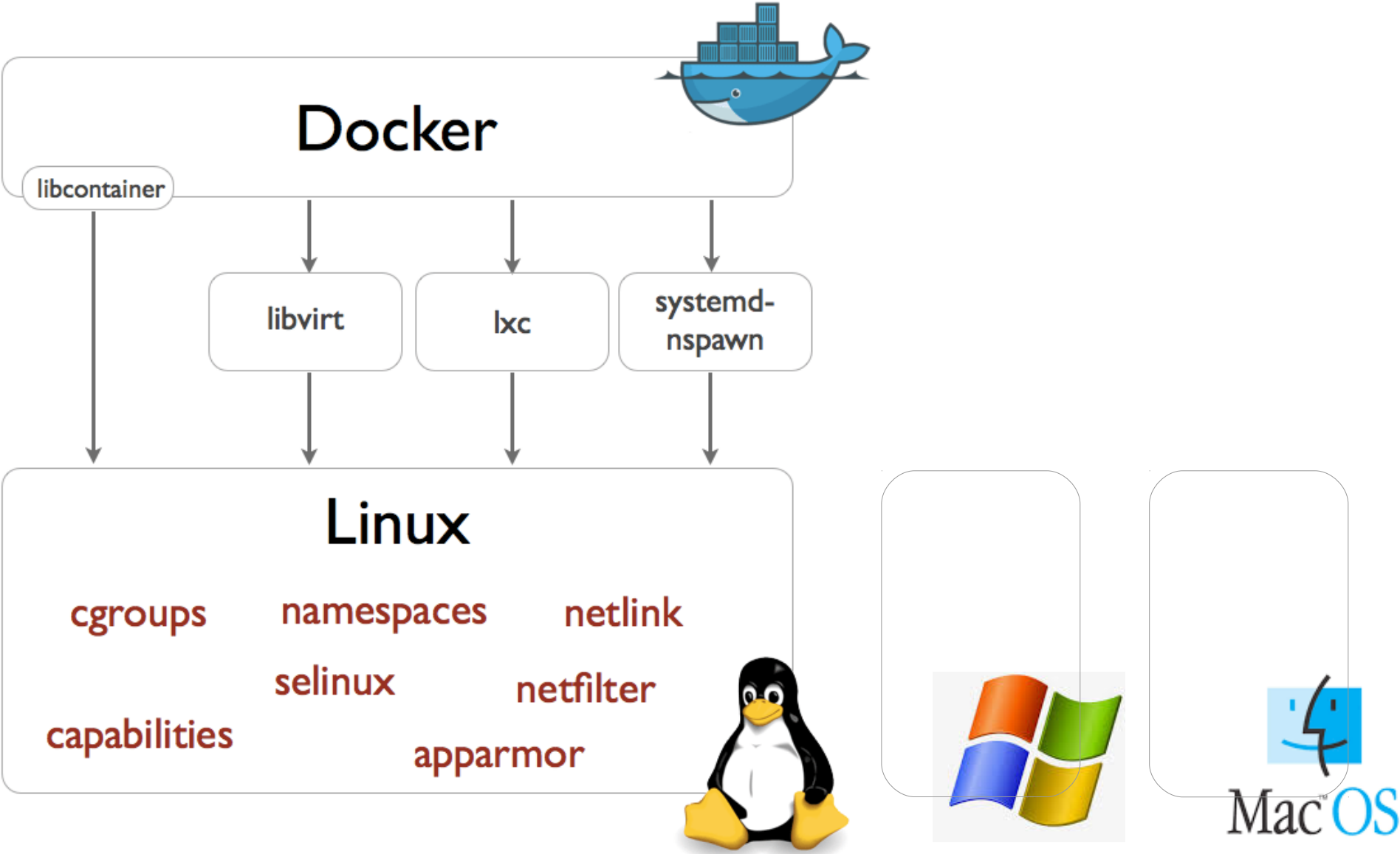
VMs



Docker Linux, Then Windows & Mac



Docker



Docker

https://docs.docker.com/

Search the docs


Guides

Product manuals

Glossary

Reference

Samples

Docker v18.03 (current) 

Get Docker 

Get started 

Get started with Docker 

Docker overview

Develop with Docker 

Configure networking 

Manage application data 

Run your app in production 

Standards and compliance 

Open source at Docker 

Documentation archive 

Docker overview

Estimated reading time: 10 minutes

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

The Docker platform

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines!

Docker provides tooling and a platform to manage the lifecycle of your containers:

 [Edit this page](#)

 [Request docs changes](#)

 [Get support](#)

On this page:

[The Docker platform](#)

[Docker Engine](#)

[What can I use Docker for?](#)

[Docker architecture](#)

[The Docker daemon](#)

[The Docker client](#)

[Docker registries](#)

[Docker objects](#)

[The underlying technology](#)

[Namespaces](#)

[Control groups](#)

[Union file systems](#)

[Container format](#)

Docker CE/EE Installation

Old version of docker was called docker-engine ...

```
# sudo apt-get remove docker docker-engine docker.io
```

CE = Community Edition

EE = Enterprise Edition

```
# add-apt-repository \  
> "deb [arch=amd64] https://download.docker.com/linux/ubuntu \  
> $(lsb_release -cs) \  
> stable" \  
# \  
# \  
# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add - \  
OK \  
# apt-get install docker-ce
```

Docker Post-Install

If Permission Denied..

```
$ docker run hello-world
docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.37/containers/create: dial unix /var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
```

But sudo OK ..

```
$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest: sha256:f5233545e43561214ca4891fd1157e1c3c563316ed8e237750d59bde73361e77
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
```

Then Grant user in group ..

```
$ sudo usermod -aG docker $USER
```

 Then Logout + re-Login ..

And re-test docker..

```
$ docker run hello-world

Hello from Docker!
```


Docker Run Hello-World

```
$ docker run hello-world
```

```
Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.
```

```
To generate this message, Docker took the following steps:
```

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

```
To try something more ambitious, you can run an Ubuntu container with:
```

```
$ docker run -it ubuntu bash
```

```
Share images, automate workflows, and more with a free Docker ID:
```

```
https://hub.docker.com/
```

```
For more examples and ideas, visit:
```

```
https://docs.docker.com/engine/userguide/
```

```
$
```

Docker Daemon

Docker ContainerD

There are 2 unix process

```
$ ps ax | grep docker | grep -v grep
13701 ?        Ssl      0:02 /usr/bin/dockerd -H fd://
13722 ?        Ssl      0:03 docker-containerd --config /var/run/docker/containerd/containerd.toml
```

Dockerd = parent, docker-containerd = child

```
$ dockerd_pid=$(ps ax | grep dockerd | grep -v grep | cut -f1 -d\ )
$ pstree $dockerd_pid
dockerd├─docker-containe──11*[{docker-containe}]
      └─10*[{dockerd}]
```

Dockerd = webserver, listening for http rest (on local unix socket)

```
$ sudo netstat -nlap | grep $dockerd_pid --color=never
unix  2      [ ACC ]     STREAM  LISTENING   76558      13701/dockerd      /var/run/docker/metrics.sock
unix  2      [ ACC ]     STREAM  LISTENING   76594      13701/dockerd      /run/docker/libnetwork/b39827c18c4
unix  3      [  ]       STREAM  CONNECTED   76559      13701/dockerd
unix  2      [  ]       DGRAM   75155      13701/dockerd
unix  3      [  ]       STREAM  CONNECTED   76557      13701/dockerd
unix  3      [  ]       STREAM  CONNECTED   76534      13701/dockerd
unix  3      [  ]       STREAM  CONNECTED   76562      13701/dockerd
```

Docker Daemon

HTTP POST /

.
{ "json": .. }

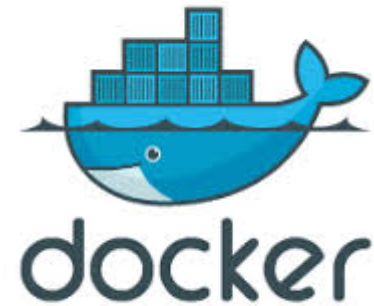


HTTP server
2376

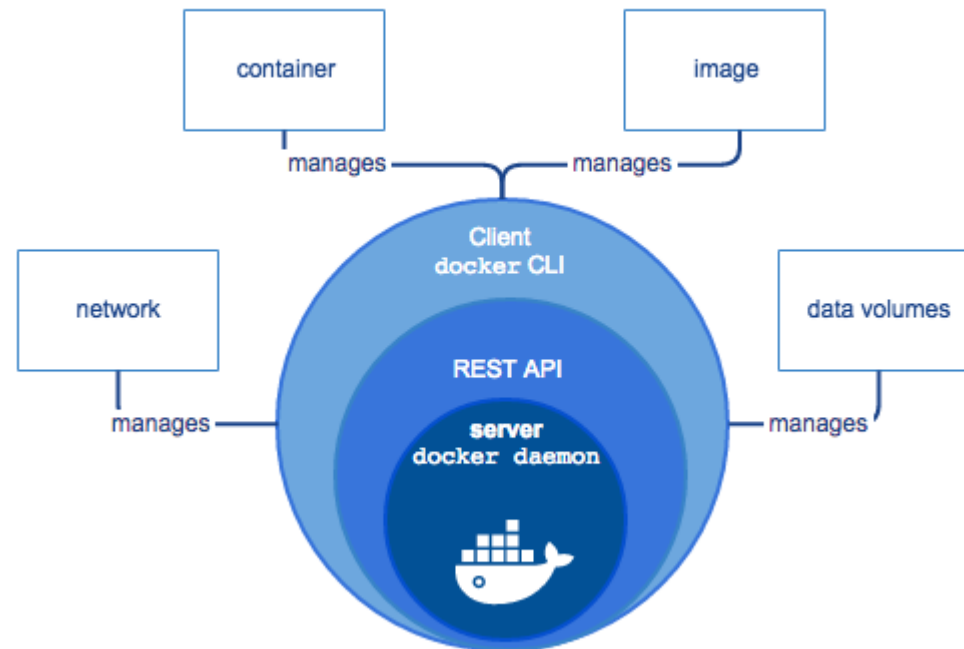


HTTP 200

.
{ "json": .. }



Docker Cli → Rest Api → Daemon



<https://docs.docker.com/engine/docker-overview/#docker-engine>

Docker SDK Language Libraries

Official SDK = GO & Python :

<https://docs.docker.com/develop/sdk/#install-the-sdks>

Go SDK

```
go get github.com/docker/docker/client
```

[Read the full Docker Engine Go SDK reference.](#)

Python SDK

- **Recommended:** Run `pip install docker`.
- **If you can't use `pip` :**
 1. [Download the package directly.](#)
 2. Extract it and change to the extracted directory,
 3. Run `python setup.py install`.

[Read the full Docker Engine Python SDK reference.](#)

View the API reference

You can [view the reference for the latest version of the API](#) or [choose a specific version.](#)

On this page

Install the SDKs

[Go SDK](#)

[Python SDK](#)

[View the API reference](#)

[Versioned API and SDK](#)

[Docker EE and CE API mismatch](#)

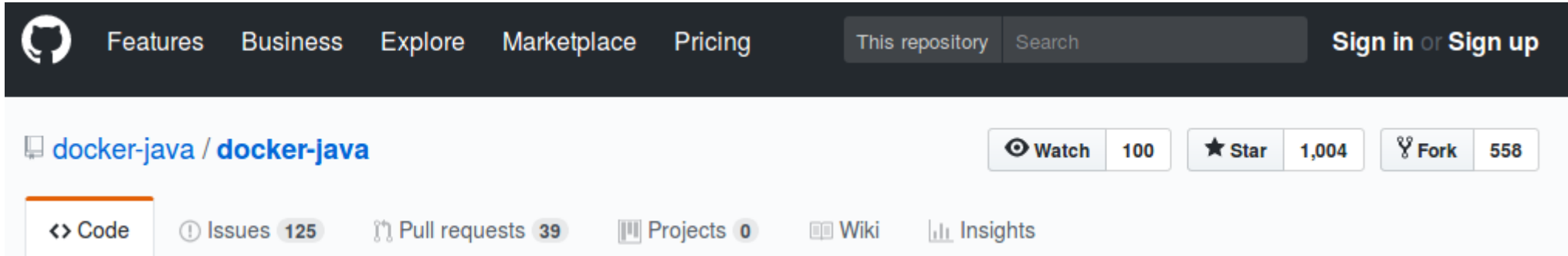
[API version matrix](#)

[Choose the SDK or API version to use](#)

[SDK and API quickstart](#)

[Unofficial libraries](#)

Other SDK Langage ... example: java



Features Business Explore Marketplace Pricing This repository Search Sign in or Sign up

docker-java / docker-java Watch 100 Star 1,004 Fork 558

Code Issues 125 Pull requests 39 Projects 0 Wiki Insights

```
<dependency>  
  <groupId>com.github.docker-java</groupId>  
  <artifactId>docker-java</artifactId>  
  <version>3.0.14</version>  
</dependency>
```

```
DockerClientConfig config = DefaultDockerClientConfig.createDefaultConfigBuilder()  
    .withDockerHost("tcp://my-docker-host.tld:2376")  
    .withDockerTlsVerify(true)  
    .withDockerCertPath("/home/user/.docker/certs")  
    .withDockerConfig("/home/user/.docker")  
    .withApiVersion("1.23")  
    .withRegistryUrl("https://index.docker.io/v1/")  
    .withRegistryUsername("dockeruser")  
    .withRegistryPassword("ilovedocker")  
    .withRegistryEmail("dockeruser@github.com")  
    .build();  
DockerClient docker = DockerClientBuilder.getInstance(config).build();
```

Docker-Java example

```

Main.java x test-docker-java/pom.xml
23  * a simple test to run
24  * <PRE>
25  * docker run -it debian bash -c "echo test..; sleep 5; echo test docker!"
26  * </PRE>
27  */
28  public static void main(String[] args) {
29      LOG.info("start test-docker");
30      DockerClientConfig config = DefaultDockerClientConfig.createDefaultConfigBuilder().build();
31      DockerClient dockerClient = DockerClientBuilder.getInstance(config).build();
32
33      CreateContainerResponse container = dockerClient.createContainerCmd("debian")
34          .withCmd("bash", "-c", "echo test..; sleep 5; echo test docker!").exec();
35      String containerId = container.getId();
36      LOG.info("createContainer => " + containerId);
37
38      dockerClient.startContainerCmd(containerId).exec();
39
40      new Thread(() -> listenEvents(dockerClient, containerId)).start();
41
42      dockerClient.logContainerCmd(containerId)
43          .withStdOut(true)
44          .withStdErr(true)
45          .withTailAll()
46          .exec(new LogContainerResultCallback() {
47              @Override
48              public void onNext(Frame item) {
49                  System.out.println("(docker) " + item);
50              }
51          });
52
53      WaitContainerResultCallback waitRes = dockerClient.waitContainerCmd(containerId)
54          .exec(new WaitContainerResultCallback());

```

Console x Tasks Display Jv JUnit Search

<terminated> Main [Java Application] /devtools/jdk/jdk1.8.0_131/bin/java (May 3, 2018, 9:21:35 AM)

```
09:21:36.532 INFO fr.an.tests.testdocker.Main - createContainer => 6e1475ba1862c4c5d6ce10750528f35b7a54d2c04eeb62b14252faa116f
(docker) STDOUT: test..
```

```
(docker) event: Event[status=die,id=6e1475ba1862c4c5d6ce10750528f35b7a54d2c04eeb62b14252faa116fa0451,from=debian,node=<null>,ty
```

```
09:21:41.801 INFO fr.an.tests.testdocker.Main - event container die.. => onComplete listener
```

```
09:21:41.801 INFO fr.an.tests.testdocker.Main - complete listen events
```

```
09:21:41.927 INFO fr.an.tests.testdocker.Main - docker exitCode:0
```

```
09:21:41.927 INFO fr.an.tests.testdocker.Main - finished
```

Docker Cli executable

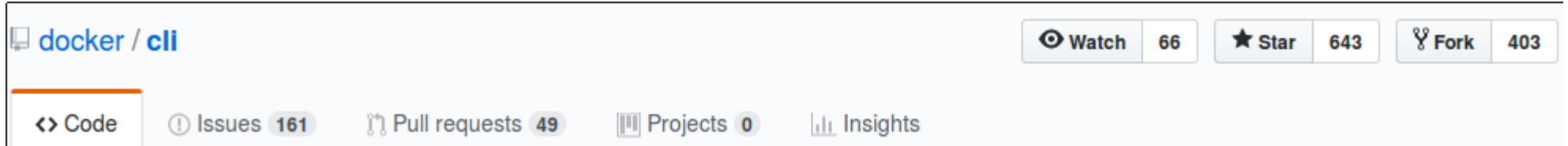
```
$ which docker
/usr/bin/docker
$
$ ldd /usr/bin/docker
        linux-vdso.so.1 => (0x00007fff17396000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f1d88ff2000)
        libltdl.so.7 => /usr/lib/x86_64-linux-gnu/libltdl.so.7 (0x00007f1d88de8000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f1d88a1e000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f1d8920f000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f1d8881a000)
$
$ docker --version
Docker version 18.03.1-ce, build 9ee9f40
$
$ docker --help

Usage:  docker COMMAND

A self-sufficient runtime for containers
```


Docker Cli Source Code

<https://github.com/docker/cli>



README.md

docker/cli is developed using Docker.

Build a linux binary:

```
$ make -f docker.Makefile binary
```

Makefile

<https://github.com/docker/cli/blob/master/scripts/build/binary>

```
12 go build -o "${TARGET}" --ldflags "${LD_FLAGS}" "${SOURCE}"
13
14 ln -sf "$(basename "${TARGET}")" build/docker
```

Written in GO lang



Docker Commands (1/3)

```
Management Commands:  
config      Manage Docker configs  
container   Manage containers  
image       Manage images  
network     Manage networks  
node        Manage Swarm nodes  
plugin      Manage plugins  
secret      Manage Docker secrets  
service     Manage services  
swarm       Manage Swarm  
system      Manage Docker  
trust       Manage trust on Docker images  
volume      Manage volumes
```

Docker Commands (2/3)

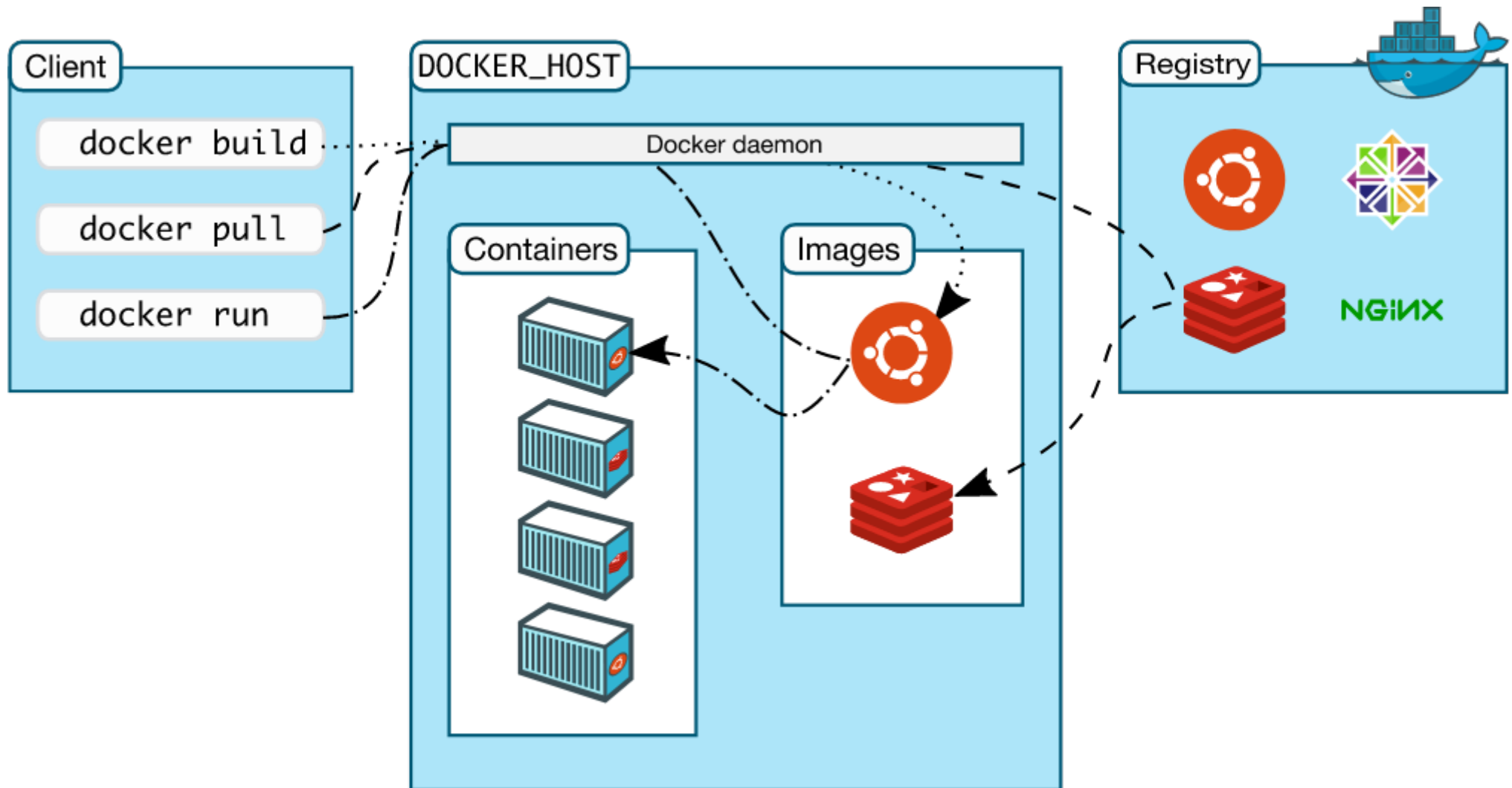
Commands:	
attach	Attach local standard input, output, and error streams to a running container
build	Build an image from a Dockerfile
commit	Create a new image from a container's changes
cp	Copy files/folders between a container and the local filesystem
create	Create a new container
diff	Inspect changes to files or directories on a container's filesystem
events	Get real time events from the server
exec	Run a command in a running container
export	Export a container's filesystem as a tar archive
history	Show the history of an image
images	List images
import	Import the contents from a tarball to create a filesystem image
info	Display system-wide information
inspect	Return low-level information on Docker objects
kill	Kill one or more running containers
load	Load an image from a tar archive or STDIN
login	Log in to a Docker registry
logout	Log out from a Docker registry
logs	Fetch the logs of a container
pause	Pause all processes within one or more containers
port	List port mappings or a specific mapping for the container
ps	List containers

Docker Commands (3/3)

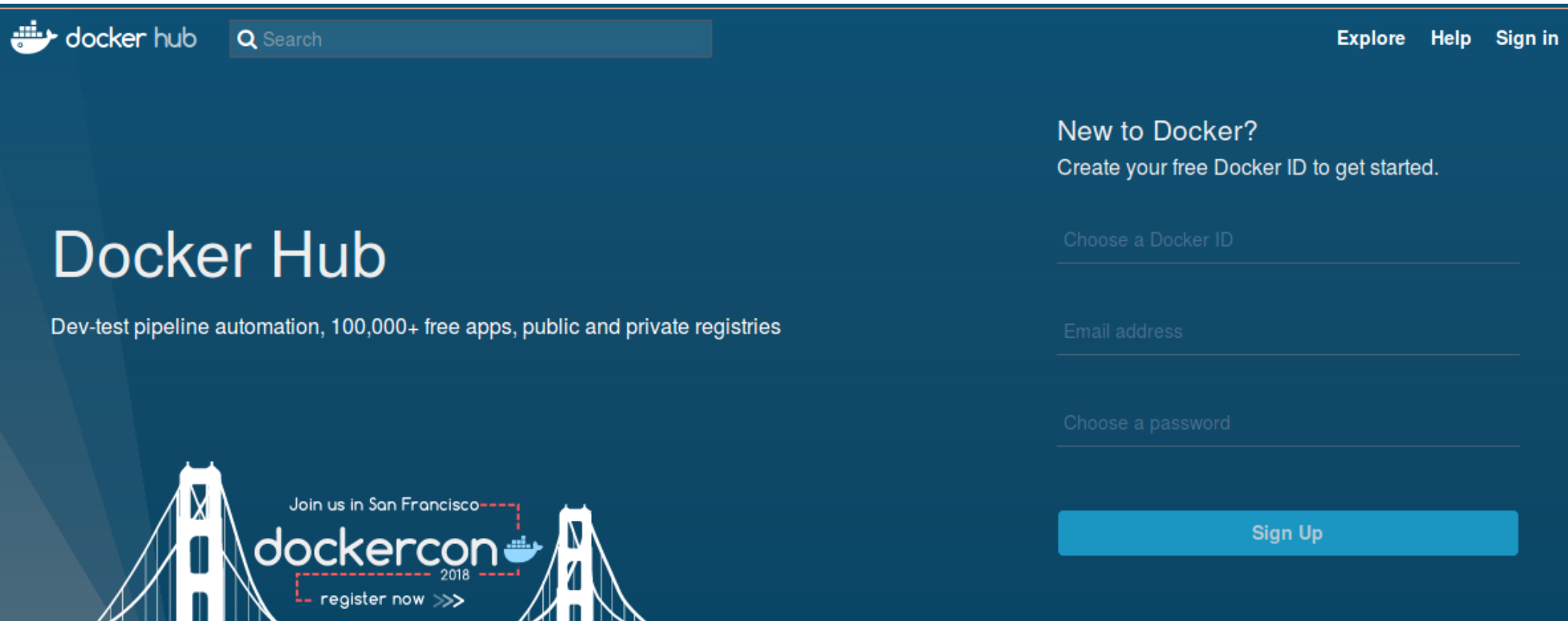
pull	Pull an image or a repository from a registry
push	Push an image or a repository to a registry
rename	Rename a container
restart	Restart one or more containers
rm	Remove one or more containers
rmi	Remove one or more images
run	Run a command in a new container
save	Save one or more images to a tar archive (streamed to STDOUT by default)
search	Search the Docker Hub for images
start	Start one or more stopped containers
stats	Display a live stream of container(s) resource usage statistics
stop	Stop one or more running containers
tag	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
top	Display the running processes of a container
unpause	Unpause all processes within one or more containers
update	Update configuration of one or more containers
version	Show the Docker version information
wait	Block until one or more containers stop, then print their exit codes

Run 'docker COMMAND --help' for more information on a command.

Daemon – Image Registry – Local Cache



https://hub.docker.com : public images repository



The screenshot shows the Docker Hub website interface. At the top left is the Docker Hub logo and a search bar. At the top right are links for 'Explore', 'Help', and 'Sign in'. The main heading is 'Docker Hub' with a subtext 'Dev-test pipeline automation, 100,000+ free apps, public and private registries'. On the right side, there is a 'New to Docker?' section with a 'Sign Up' button. At the bottom, there is a banner for 'dockercon 2018' with a 'register now >>>' link.

docker hub

Explore Help Sign in

Docker Hub

Dev-test pipeline automation, 100,000+ free apps, public and private registries

New to Docker?
Create your free Docker ID to get started.

Choose a Docker ID

Email address

Choose a password

[Sign Up](#)

Join us in San Francisco
dockercon
2018
[register now >>>](#)

Search Images in Docker Hub








Docker Store is the new place to discover public Docker content. [Check it out →](#)

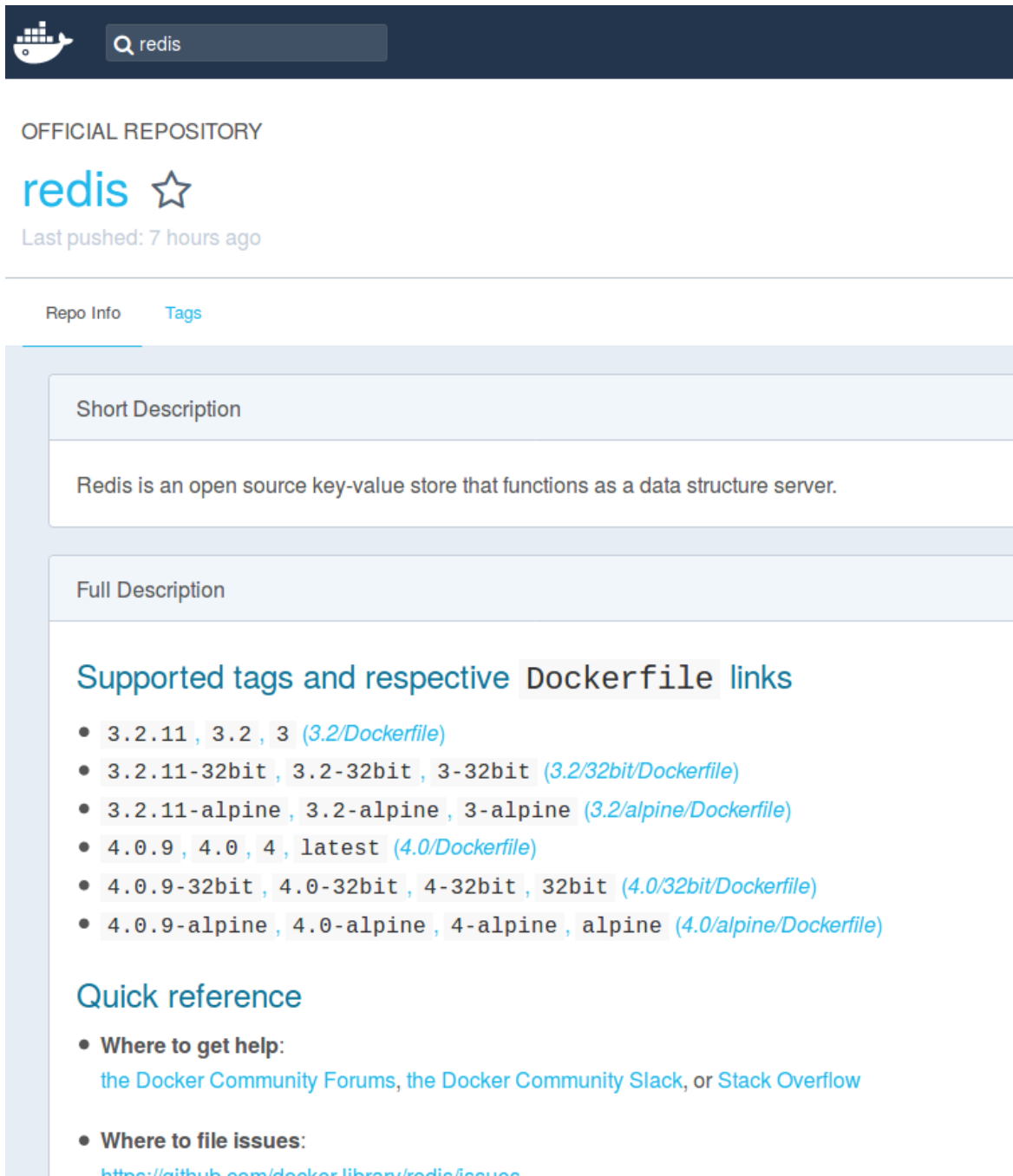
Explore Help [Sign up](#) Sign In

Repositories (8677)

All

 redis official	5.1K STARS	10M+ PULLS	> DETAILS
 bitnami/redis public automated build	75 STARS	1M+ PULLS	> DETAILS
 webhippie/redis public automated build	8 STARS	1M+ PULLS	> DETAILS
 rediscommander/redis-commander public automated build	4 STARS	100K+ PULLS	> DETAILS
 oliver006/redis-exporter			

Docker Hub Image Description 1/2



The screenshot shows the Docker Hub interface for the 'redis' image. At the top, there is a search bar with 'redis' entered. Below the search bar, it says 'OFFICIAL REPOSITORY' and 'redis' with a star icon. The text 'Last pushed: 7 hours ago' is visible. There are two tabs: 'Repo Info' (selected) and 'Tags'. The main content area is divided into sections: 'Short Description' (Redis is an open source key-value store that functions as a data structure server.), 'Full Description' (Supported tags and respective Dockerfile links), and 'Quick reference' (Where to get help: the Docker Community Forums, the Docker Community Slack, or Stack Overflow; Where to file issues: https://github.com/docker-library/redis/issues).

redis

OFFICIAL REPOSITORY

redis ☆

Last pushed: 7 hours ago

Repo Info Tags

Short Description

Redis is an open source key-value store that functions as a data structure server.

Full Description

Supported tags and respective Dockerfile links

- 3.2.11, 3.2, 3 (3.2/Dockerfile)
- 3.2.11-32bit, 3.2-32bit, 3-32bit (3.2/32bit/Dockerfile)
- 3.2.11-alpine, 3.2-alpine, 3-alpine (3.2/alpine/Dockerfile)
- 4.0.9, 4.0, 4, latest (4.0/Dockerfile)
- 4.0.9-32bit, 4.0-32bit, 4-32bit, 32bit (4.0/32bit/Dockerfile)
- 4.0.9-alpine, 4.0-alpine, 4-alpine, alpine (4.0/alpine/Dockerfile)

Quick reference

- Where to get help:
the Docker Community Forums, the Docker Community Slack, or Stack Overflow
- Where to file issues:
<https://github.com/docker-library/redis/issues>

Docker HUB Image Description 2/2

How to use this image

start a redis instance

```
$ docker run --name some-redis -d redis
```

This image includes `EXPOSE 6379` (the redis port), so standard container linking will make it automatically available to the linked containers (as the following examples illustrate).

start with persistent storage

```
$ docker run --name some-redis -d redis redis-server --appendonly yes
```

If persistence is enabled, data is stored in the `VOLUME /data`, which can be used with

```
--volumes-from some-volume-container or -v /docker/host/dir:/data
```

(see docs.docker.com/volumes/).

For more about Redis Persistence, see <http://redis.io/topics/persistence>.

connect to it from an application

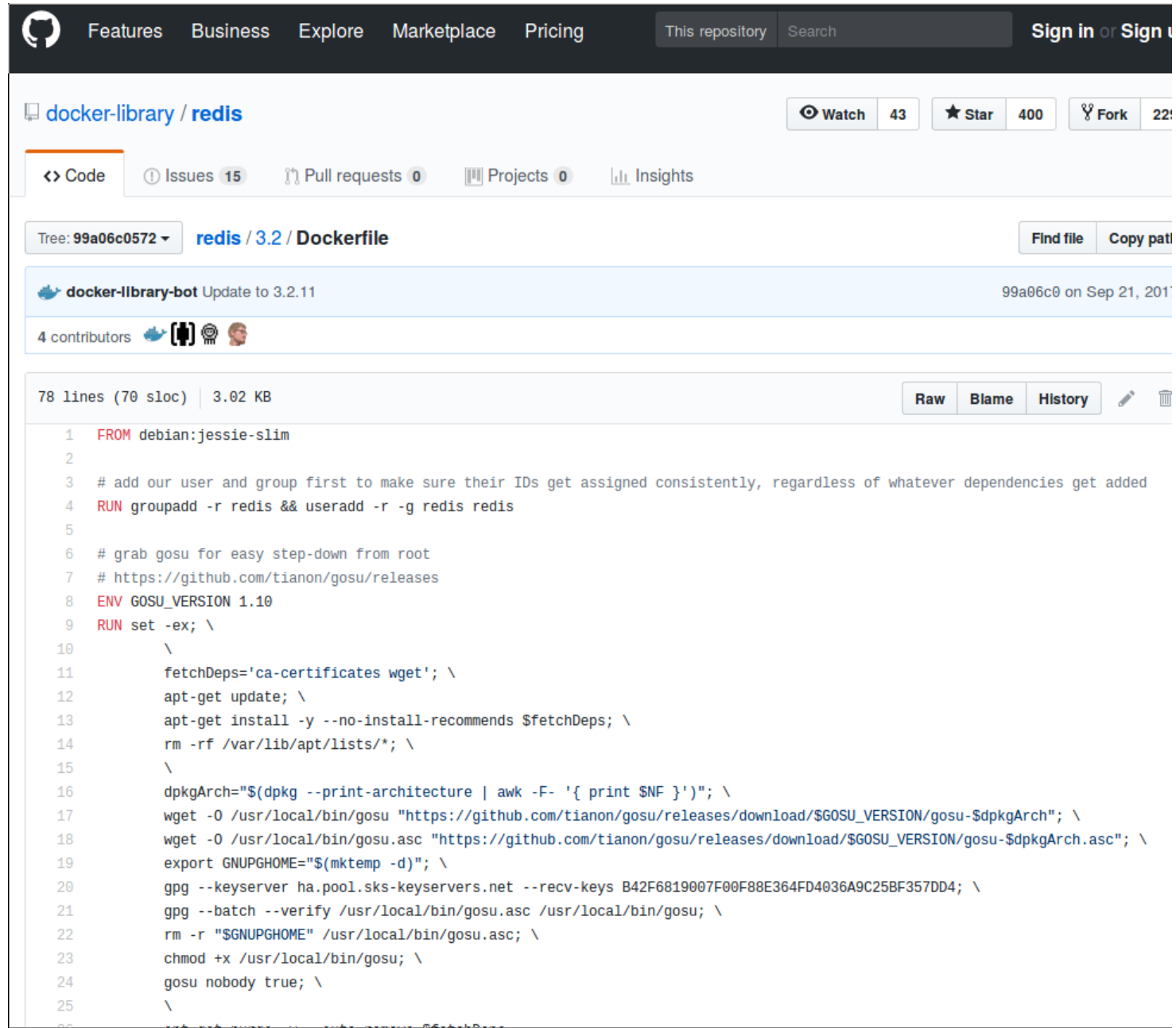
```
$ docker run --name some-app --link some-redis:redis -d application-
```

... or via `redis-cli`

```
$ docker run -it --link some-redis:redis --rm redis redis-cli -h red
```

Dockerfile..

• 3.2.11 , 3.2 , 3 (3.2/Dockerfile)



The screenshot shows the GitHub interface for the `docker-library/redis` repository. The repository is at commit `99a06c0572` and the file being viewed is `redis / 3.2 / Dockerfile`. The file is 78 lines long (70 sloc) and 3.02 KB in size. The content of the Dockerfile is as follows:

```
1 FROM debian:jessie-slim
2
3 # add our user and group first to make sure their IDs get assigned consistently, regardless of whatever dependencies get added
4 RUN groupadd -r redis && useradd -r -g redis redis
5
6 # grab gosu for easy step-down from root
7 # https://github.com/tianon/gosu/releases
8 ENV GOSU_VERSION 1.10
9 RUN set -ex; \
10     \
11     fetchDeps='ca-certificates wget'; \
12     apt-get update; \
13     apt-get install -y --no-install-recommends $fetchDeps; \
14     rm -rf /var/lib/apt/lists/*; \
15     \
16     dpkgArch="$(dpkg --print-architecture | awk -F- '{ print $NF }')"; \
17     wget -O /usr/local/bin/gosu "https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-$dpkgArch"; \
18     wget -O /usr/local/bin/gosu.asc "https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-$dpkgArch.asc"; \
19     export GNUPGHOME="$(mktemp -d)"; \
20     gpg --keyserver ha.pool.sks-keyservers.net --recv-keys B42F6819007F00F88E364FD4036A9C25BF357DD4; \
21     gpg --batch --verify /usr/local/bin/gosu.asc /usr/local/bin/gosu; \
22     rm -r "$GNUPGHOME" /usr/local/bin/gosu.asc; \
23     chmod +x /usr/local/bin/gosu; \
24     gosu nobody true; \
25     \
26     apt-get update; \
27     apt-get install -y --no-install-recommends redis; \
28     rm -rf /var/lib/apt/lists/*;
```

\$ docker image --help

```
$ docker image --help
```

```
Usage:  docker image COMMAND
```

```
Manage images
```

```
Options:
```

```
Commands:
```

```
  build      Build an image from a Dockerfile
  history    Show the history of an image
  import     Import the contents from a tarball to create a filesystem image
  inspect    Display detailed information on one or more images
  load       Load an image from a tar archive or STDIN
  ls         List images
  prune      Remove unused images
  pull       Pull an image or a repository from a registry
  push       Push an image or a repository to a registry
  rm         Remove one or more images
  save       Save one or more images to a tar archive (streamed to STDOUT by default)
  tag        Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
```

\$ docker image ls

```
$ docker image ls
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
my-redis-cli        latest      346a76c28886     About an hour ago 8.36MB
redis               latest      bfc1f6df2db      41 hours ago    107MB
debian              latest      8626492fec3      5 days ago     101MB
hello-world         latest      e38bc07ac18e     3 weeks ago    1.85kB
alpine              latest      3fd9065eaf02     3 months ago    4.15MB
prologic/redis-cli latest      0521d5ffcb39     3 years ago     394MB
$
```

\$ docker build --help

```
$ docker build --help
```

```
Usage: docker build [OPTIONS] PATH | URL | -
```

```
Build an image from a Dockerfile
```

```
Options:
```

--add-host list	Add a custom host-to-IP mapping (host:ip)
--build-arg list	Set build-time variables
--cache-from strings	Images to consider as cache sources
--cgroup-parent string	Optional parent cgroup for the container
--compress	Compress the build context using gzip
--cpu-period int	Limit the CPU CFS (Completely Fair Scheduler) period
--cpu-quota int	Limit the CPU CFS (Completely Fair Scheduler) quota
-c, --cpu-shares int	CPU shares (relative weight)
--cpuset-cpus string	CPUs in which to allow execution (0-3, 0,1)
--cpuset-mems string	MEMs in which to allow execution (0-3, 0,1)
--disable-content-trust	Skip image verification (default true)
-f, --file string	Name of the Dockerfile (Default is 'PATH/Dockerfile')
--force-rm	Always remove intermediate containers
--iidfile string	Write the image ID to the file
--isolation string	Container isolation technology
--label list	Set metadata for an image
-m, --memory bytes	Memory limit
--memory-swap bytes	Swap limit equal to memory plus swap: '-1' to enable unlimited swap
--network string	Set the networking mode for the RUN instructions during build
--no-cache	Do not use cache when building the image
--pull	Always attempt to pull a newer version of the image
-q, --quiet	Suppress the build output and print image ID on success
--rm	Remove intermediate containers after a successful build
--security-opt strings	Security options
--shm-size bytes	Size of /dev/shm
-t, --tag list	Name and optionally a tag in the 'name:tag' format
--target string	Set the target build stage to build.
--ulimit ulimit	Ulimit options (default [])

\$ docker build . -t my-tag

```
$ docker build . -t my-redis-cli
Sending build context to Docker daemon 2.048kB
Step 1/3 : FROM alpine:latest
--> 3fd9065eaf02
Step 2/3 : RUN apk --update add redis
--> Using cache
--> c6e71ba5e36e
Step 3/3 : ENTRYPOINT ["redis-cli"]
--> Using cache
--> 346a76c28886
Successfully built 346a76c28886
Successfully tagged my-redis-cli:latest
```

\$ docker commit ... (internal to build)

```
$ docker commit --help

Usage:  docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

Create a new image from a container's changes

Options:
  -a, --author string      Author (e.g., "John Hannibal Smith <hannibal@a-team.com>")
  -c, --change list        Apply Dockerfile instruction to the created image
  -m, --message string     Commit message
  -p, --pause              Pause container during commit (default true)
```

Create commit image

```
$ docker commit 269dc6f1fd3c my-debian-commit1
sha256:0e510528021cc51c6e855afd45d79325576daf3448ae3adff1b11ab94bd7f4e8
$
```

Check image exist locally

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-debian-commit1	latest	0e510528021c	40 seconds ago	101MB

Run image

```
$ docker run -it --name bash-image1 0e510528021c
root@f0cf5e195254:/#
root@f0cf5e195254:/# hostname
f0cf5e195254
```

Image Compatibilities

Linux – Windows ?

Docker Motto : “build once, run everywhere”

... everywhere on same containerd platform !!
Linux image => on linux ContainerD

Linux Image => any Debian,Ubuntu,RedHat,...
(= Elf - X86 binary libraries files)

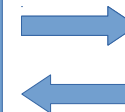
\$ docker run ... docker run -it

```
$ docker run debian:latest /bin/echo "Hello from docker"
Hello from docker
$
```

Run -i(nteractive) -t(erminal)

```
$ docker run -it debian:latest
root@caf4a279f77e:/#
```

Local Terminal
shell



Docker
Container

You are like in a SSH container host
... running debian bash entry point

```
root@caf4a279f77e:/# hostname
caf4a279f77e
root@caf4a279f77e:/#
root@caf4a279f77e:/# exit
```

CTRL+D to exit ...

\$ docker ps

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
269dc6f1fd3c      debian:latest      "bash"             2 minutes ago      Up 2 minutes
$
```

\$ docker exec ... like ssh into running container

exec -i(nteractive) -t(erminal)

```
$ docker exec -it 269dc6f1fd3c /bin/bash
root@269dc6f1fd3c:/#
root@269dc6f1fd3c:/# hostname
269dc6f1fd3c
root@269dc6f1fd3c:/#
```

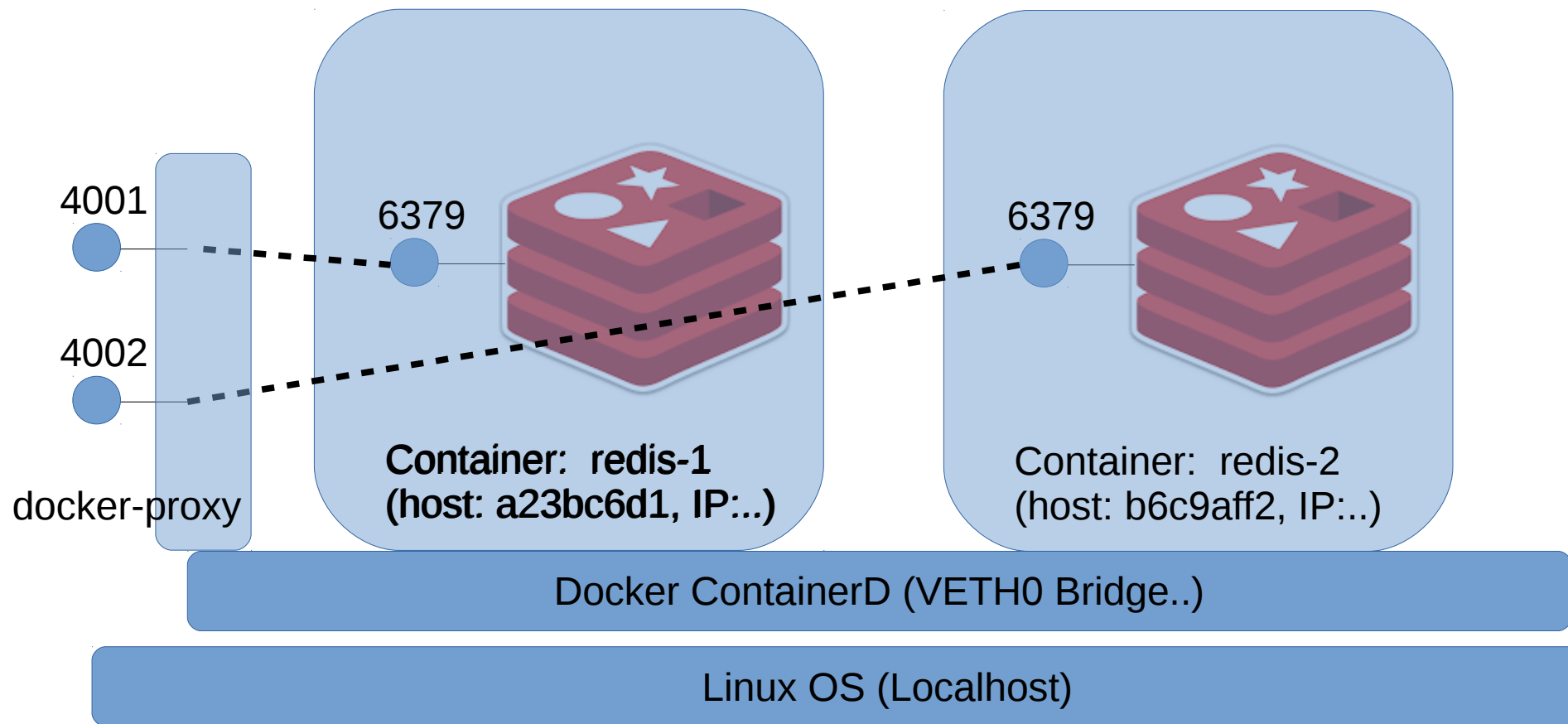
exec non interactive

```
$ docker exec 269dc6f1fd3c /bin/echo 'from docker exec'
from docker exec
$
$
$ docker exec 269dc6f1fd3c /bin/hostname
269dc6f1fd3c
$
```

Docker Port Export

```
$ docker run -p 4001:6379 --name redis-1 redis:latest
```

```
$ docker run -p 4002:6379 --name redis-2 redis:latest
```



.. Docker -p <extport>:<intport>

```
$ docker run -p 4001:6379 --name redis-1 redis:latest
1:C 03 May 21:07:08.779 # o000o000o000o Redis is starting o000o000o000o
1:C 03 May 21:07:08.779 # Redis version=4.0.9, bits=64, commit=00000000, modified=0
1:C 03 May 21:07:08.779 # Warning: no config file specified, using the default conf
redis-server /path/to/redis.conf
1:M 03 May 21:07:08.780 * Running mode=standalone, port=6379.
1:M 03 May 21:07:08.780 # WARNING: The TCP backlog setting of 511 cannot be enforced
```

Check connecting manually (telnet)
to redis on 4001 ... not on 6379 !!

```
$ telnet localhost 4001
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

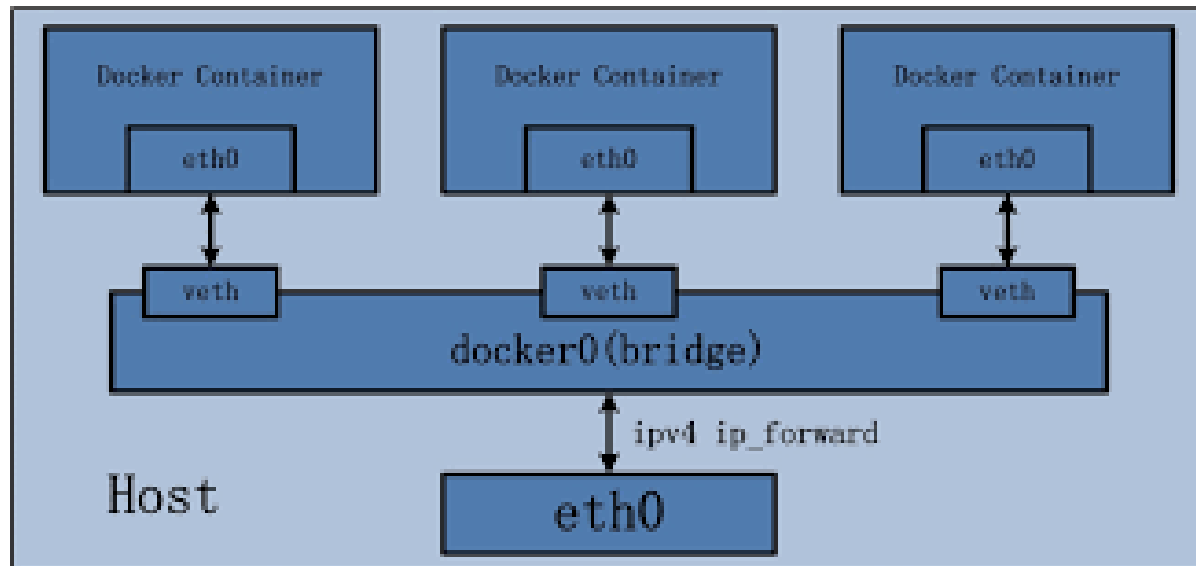
execute redis commands

It works ..

```
Escape character is '^]'.
incr mycounter
:1
incr mycounter
:2
decr mycounter
:1

```

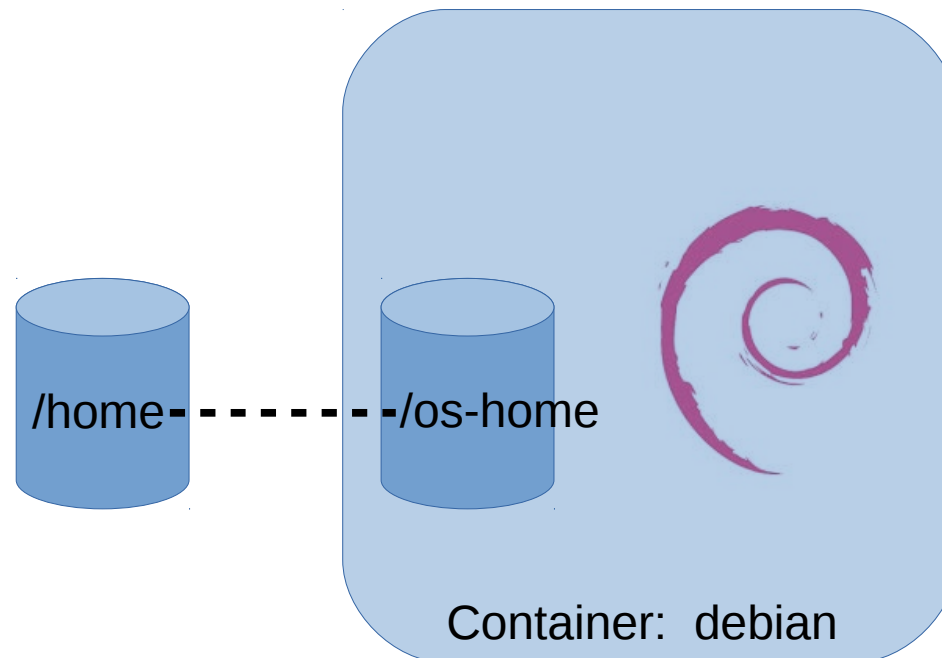
Docker NAT : eth0 → veth → bridge



```
$ ifconfig
docker0  Link encap:Ethernet  HWaddr 02:42:c7:ce:c7:8b
         inet addr:172.17.0.1  Bcast:172.17.255.255  Mask:255.255.0.0
         inet6 addr: fe80::42:c7ff:face:c78b/64  Scope:Link
         UP BROADCAST MULTICAST  MTU:1500  Metric:1
         RX packets:0  errors:0  dropped:0  overruns:0  frame:0
         TX packets:8  errors:0  dropped:0  overruns:0  carrier:0
         collisions:0  txqueuelen:0
         RX bytes:0 (0.0 B)  TX bytes:828 (828.0 B)
```

Docker -v <extdir>:<intdir>

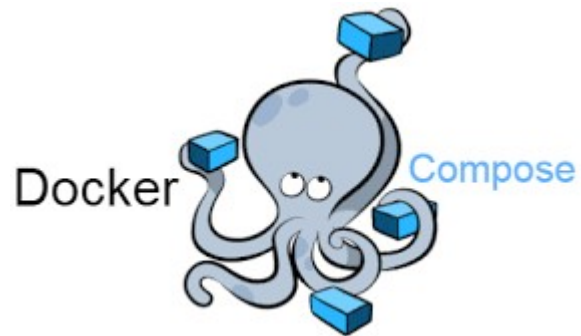
```
$ docker run -it -v /home:/os-home debian:latest
root@5785bed52d75:/#
root@5785bed52d75:/# ls /os-home/
arnaud myriam nedra remy soft
root@5785bed52d75:/#
root@5785bed52d75:/# exit
$
$ ls /home
arnaud myriam nedra remy soft
```



Docker ContainerD (VETH0 Bridge..)

Linux OS (Localhost)

Docker Compose : Local Orchestration



Docker-compose.yml File

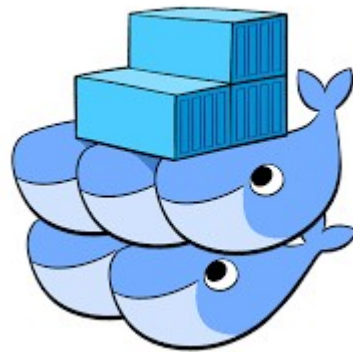
\$ docker-compose up

1. Define your app's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.
3. Run `docker-compose up` and Compose starts and runs your entire app.

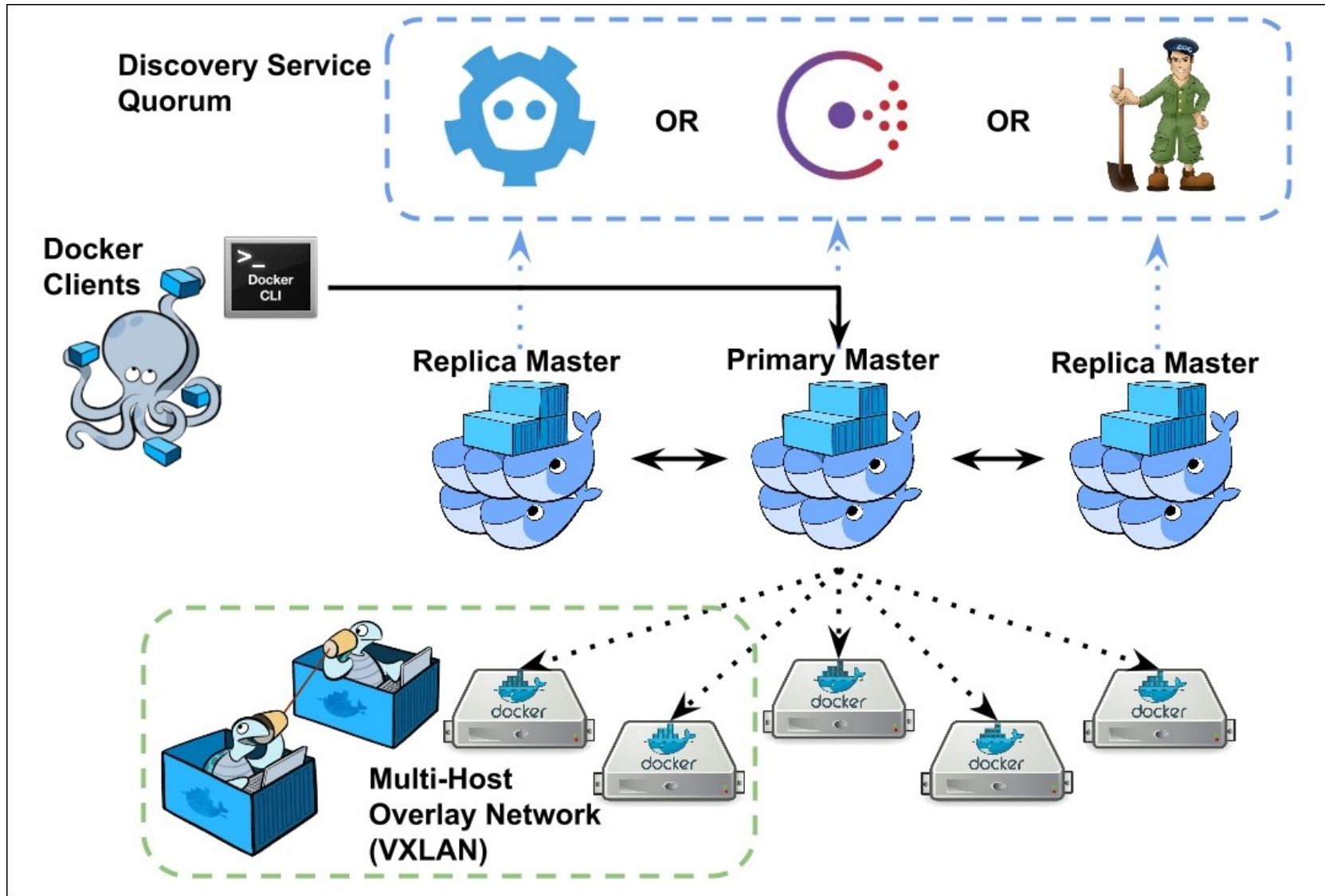
A `docker-compose.yml` looks like this:

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Docker Cluster Orchestration Swarm (=Toy)... Kubernetes



Cluster = Admin(s) + N Nodes



Part 1 : Intro to VM and Containers

Part 2 : Intro To Docker

next steps

Part 3: Intro to Kubernetes
(Cluster Orchestration)