Intro to Docker Workshop Exercises

Anusha Ragunathan, Harish Jayakumar, Madhuri Yechuri

Agenda

This workshop will help write your first Dockerized application, build it using Dockerfiles, distribute it to a registry and pull an image and run it in seconds. You will also learn to manage storage volumes and networks, observe statistics, inspect container objects.

- Setup check
- Container lifecycle. Run your first container.
- Image lifecycle. Understand docker images.
- Observing events/objects
- Auditing Logs
- Volume subsystem basics
- Network subsystem basics
- Docker Hub
- Building your first docker image
- Writing your first Dockerized app and updating it.

Logistics

TA assignment.

Anusha will assist row 1
Madhuri will assist row 2 and row 3
Harish will assist row 4 and row 5

Pair two students so they can hack and collaborate together.

Introduce yourself to neighbor

Prerequisites:

- Laptop running a 64-bit operating system
    - Windows 7 or higher with virtualization enabled
    - Mac OS X 10.8 or newer
    - Linux: a distro and version on the Docker supported list (Ubuntu, RHEL, CentOS preferred)
- Software requirements:
    - "Docker Toolbox 1.11 on Mac and Windows
    - https://www.docker.com/products/docker-toolbox
    - Docker Engine 1.11 on Linux
    - deb/rpm install: curl -fsSL https://get.docker.com/ | sh
    - Linux 64bit tgz: https://get.docker.com/builds/Linux/x86_64/docker-1.11.0.tgz

In case students have not installed the prerequisites, please raise hand so TAs can come around to assist.

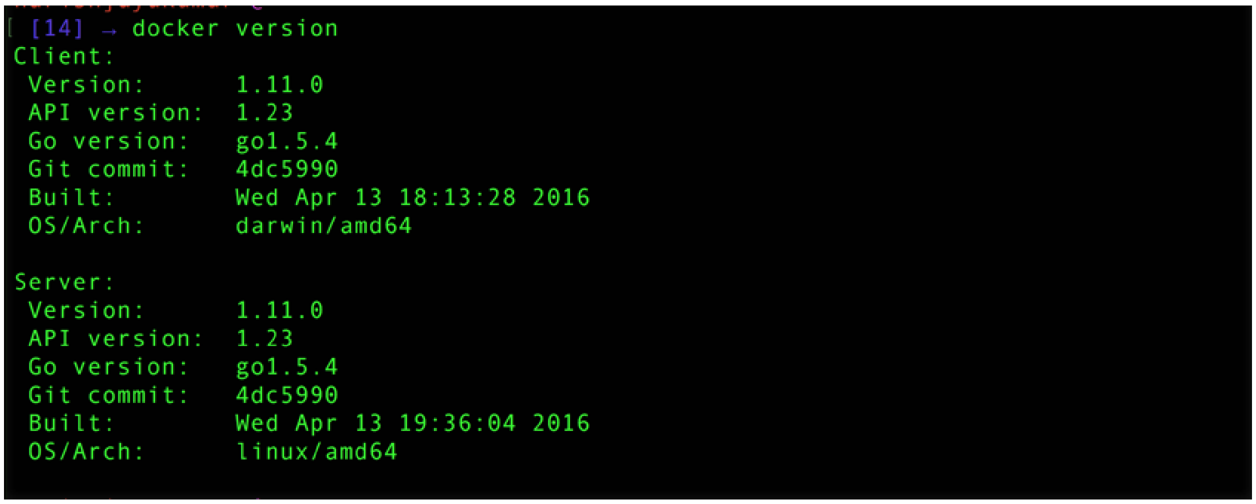
Section 1: Docker Engine & setup check

Docker operates in client and server model.

• The Docker daemon: Receives and processes incoming Docker REST API requests.
• The Docker client: Talks to the Docker daemon via the Docker API.
• Docker Hub Registry:Public image registry. The Docker daemon talks to it via the registry API.

Exercise 1

$→ docker version



```
[ [14] → docker version
Client:
 Version:      1.11.0
 API version:  1.23
 Go version:   go1.5.4
 Git commit:   4dc5990
 Built:        Wed Apr 13 18:13:28 2016
 OS/Arch:      darwin/amd64

Server:
 Version:      1.11.0
 API version:  1.23
 Go version:   go1.5.4
 Git commit:   4dc5990
 Built:        Wed Apr 13 19:36:04 2016
 OS/Arch:      linux/amd64
```

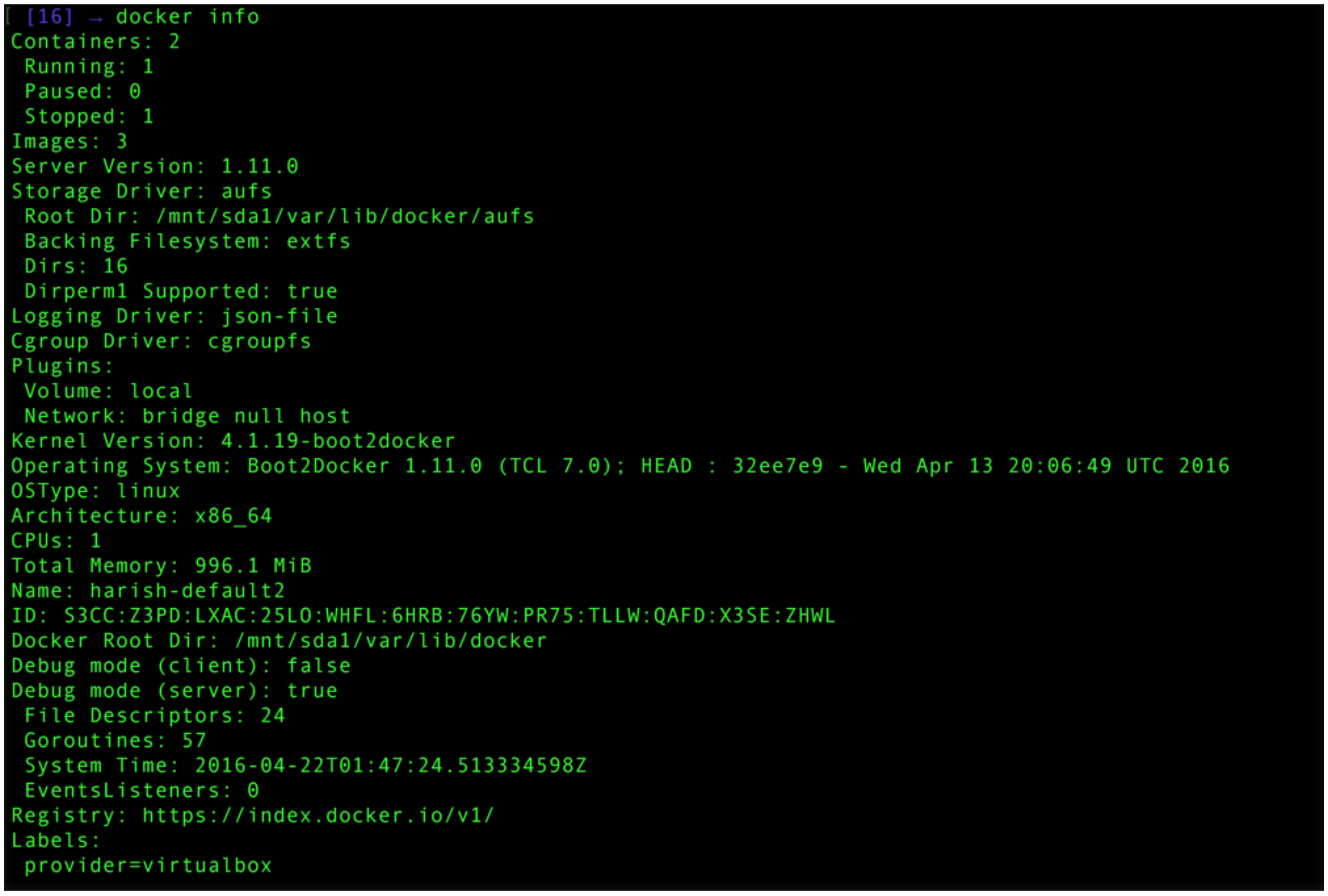
If this doesn't work, raise your hand and one of our TAs will help you.

Troubleshooting notes

- Add non-sudo access to the Docker socket by adding your username to the `docker` group.
  $sudo usermod -a -G docker $USER

  Check **`docker-machine env default`**

$ docker info



```
[ [16] → docker info
Containers: 2
 Running: 1
 Paused: 0
 Stopped: 1
Images: 3
Server Version: 1.11.0
Storage Driver: aufs
 Root Dir: /mnt/sda1/var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 16
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge null host
Kernel Version: 4.1.19-boot2docker
Operating System: Boot2Docker 1.11.0 (TCL 7.0); HEAD : 32ee7e9 - Wed Apr 13 20:06:49 UTC 2016
OSType: linux
Architecture: x86_64
CPUs: 1
Total Memory: 996.1 MiB
Name: harish-default2
ID: S3CC:Z3PD:LXAC:25LO:WHFL:6HRB:76YW:PR75:TLLW:QAFD:X3SE:ZHWL
Docker Root Dir: /mnt/sda1/var/lib/docker
Debug mode (client): false
Debug mode (server): true
 File Descriptors: 24
 Goroutines: 57
 System Time: 2016-04-22T01:47:24.513334598Z
 EventsListeners: 0
Registry: https://index.docker.io/v1/
Labels:
 provider=virtualbox
```

<u>**Section 2: Container lifecycle:**</u>

In this exercise we run our first container as a background container. We understand how to
- shell into it
- query all the containers
- name containers. Fun quiz. What happens if you dont assign a name to the container?
 We'll also see what happens behind the scenes when a container runs for the first time.

What's the PID of the container entry point and what happens to the container what that process exits. (Quiz 1)

Throughout the workshop we will be using and building on top of one image, **"aragunathan/rubyexample".** What does it do?

- It's based on a Ubuntu image
- It contains
    - Ruby
    - Sinatra (web development framework for Ruby)
    - Any required dependencies

Note about exercise format:

- All exercises have clear instructions on what needs to be done.
- All exercises have screen shots of what a sample output should look like.

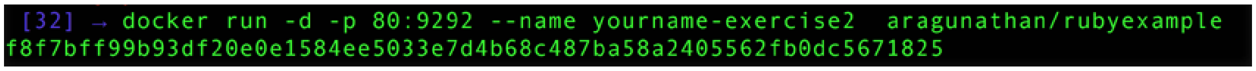
Note for Windows users:

- If there are Windows specific command line changes, please adapt accordingly. Another option is to get a TAs help to use the linux VM on your Windows host to perform your exercises on a native Linux environment (You will be doing an "**docker-machine ssh default"** from Docker Toolbox)

<u>**Exercise 2:**</u>

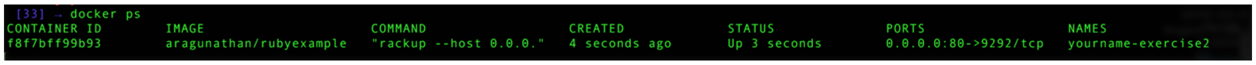
Examples : (Note container id's will be different)

2.1 $ docker run -d -p 80:9292 --name yourname-exercise 2 aragunathan/rubyexample



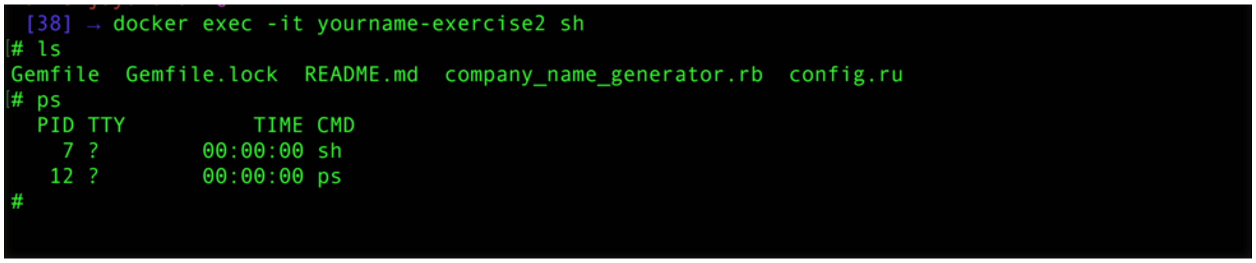
```
 [32] → docker run -d -p 80:9292 --name yourname-exercise2  aragunathan/rubyexample
f8f7bff99b93df20e0e1584ee5033e7d4b68c487ba58a2405562fb0dc5671825
```

2.2 $ docker ps



```
 [33] → docker ps
CONTAINER ID        IMAGE                     COMMAND                  CREATED             STATUS              PORTS                  NAMES
f8f7bff99b93        aragunathan/rubyexample   "rackup --host 0.0.0."   4 seconds ago       Up 3 seconds        0.0.0.0:80->9292/tcp   yourname-exercise2
```

2.3 $ docker exec -it yourname-exercise2 sh

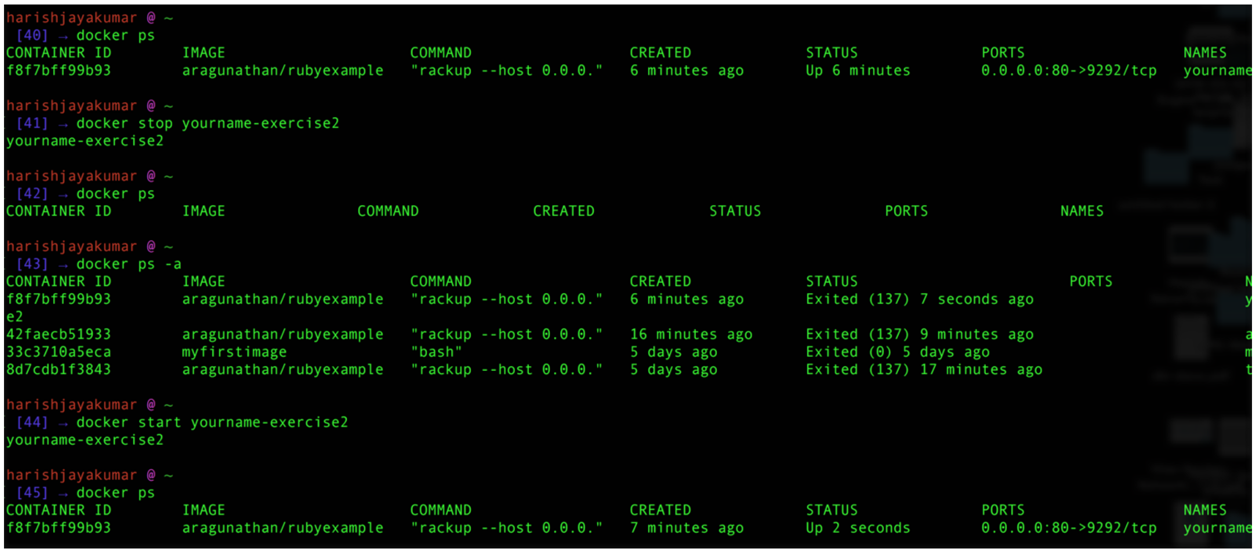


```
 [38] → docker exec -it yourname-exercise2 sh
[# ls
Gemfile  Gemfile.lock  README.md  company_name_generator.rb  config.ru
[# ps
  PID TTY          TIME CMD
    7 ?        00:00:00 sh
   12 ?        00:00:00 ps
#
```

Do an exit and come out of the container

2.4 Stop and Start

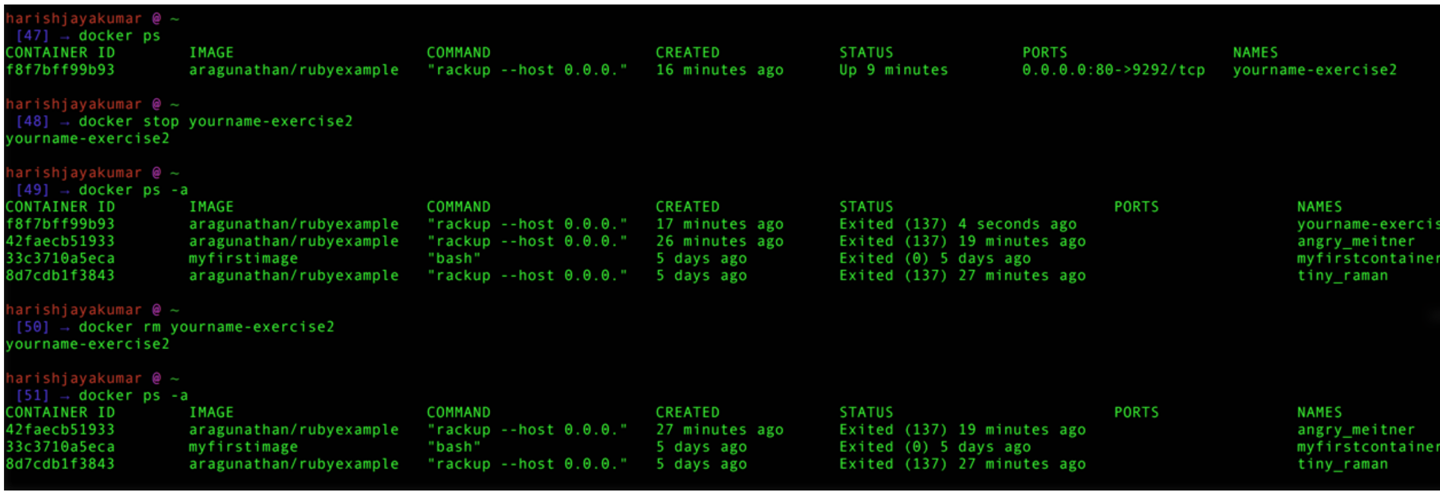
- Check container is running $ docker ps
- Stop the running container $ docker stop yourname-exercise2
- Check container is no longer running $ docker ps
- Check all containers/processes $ docker ps -a
- Start the stopped container$ docker start yourname-exercise2
- Check container is now running $ docker ps



```
harishjayakumar @ ~
 [40] → docker ps
CONTAINER ID        IMAGE                      COMMAND                  CREATED             STATUS              PORTS                  NAMES
f8f7bff99b93        aragunathan/rubyexample    "rackup --host 0.0.0."   6 minutes ago       Up 6 minutes        0.0.0.0:80->9292/tcp   yourname

harishjayakumar @ ~
 [41] → docker stop yourname-exercise2
yourname-exercise2

harishjayakumar @ ~
 [42] → docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES

harishjayakumar @ ~
 [43] → docker ps -a
CONTAINER ID        IMAGE                      COMMAND                  CREATED             STATUS                        PORTS               N
f8f7bff99b93        aragunathan/rubyexample    "rackup --host 0.0.0."   6 minutes ago       Exited (137) 7 seconds ago                        y
e2
42faecb51933        aragunathan/rubyexample    "rackup --host 0.0.0."   16 minutes ago      Exited (137) 9 minutes ago                        a
33c3710a5eca        myfirstimage               "bash"                   5 days ago          Exited (0) 5 days ago                             m
8d7cdb1f3843        aragunathan/rubyexample    "rackup --host 0.0.0."   5 days ago          Exited (137) 17 minutes ago                       t

harishjayakumar @ ~
 [44] → docker start yourname-exercise2
yourname-exercise2

harishjayakumar @ ~
 [45] → docker ps
CONTAINER ID        IMAGE                      COMMAND                  CREATED             STATUS              PORTS                  NAMES
f8f7bff99b93        aragunathan/rubyexample    "rackup --host 0.0.0."   7 minutes ago       Up 2 seconds        0.0.0.0:80->9292/tcp   yourname
```

2.5 Remove:

- Check container is running $ docker ps
- Stop the running container $ docker stop yourname-exercise2
- Check all containers/processes $ docker ps -a
- Remove the stopped container$ docker rm yourname-exercise2
- Check container is now running $ docker ps



```
harishjayakumar @ ~
 [47] → docker ps
CONTAINER ID        IMAGE                    COMMAND                  CREATED             STATUS              PORTS                    NAMES
f8f7bff99b93        aragunathan/rubyexample   "rackup --host 0.0.0."   16 minutes ago      Up 9 minutes        0.0.0.0:80->9292/tcp     yourname-exercise2

harishjayakumar @ ~
 [48] → docker stop yourname-exercise2
yourname-exercise2

harishjayakumar @ ~
 [49] → docker ps -a
CONTAINER ID        IMAGE                    COMMAND                  CREATED             STATUS                        PORTS               NAMES
f8f7bff99b93        aragunathan/rubyexample   "rackup --host 0.0.0."   17 minutes ago      Exited (137) 4 seconds ago                        yourname-exercis
42faecb51933        aragunathan/rubyexample   "rackup --host 0.0.0."   26 minutes ago      Exited (137) 19 minutes ago                       angry_meitner
33c3710a5eca        myfirstimage              "bash"                   5 days ago          Exited (0) 5 days ago                             myfirstcontainer
8d7cdb1f3843        aragunathan/rubyexample   "rackup --host 0.0.0."   5 days ago          Exited (137) 27 minutes ago                       tiny_raman

harishjayakumar @ ~
 [50] → docker rm yourname-exercise2
yourname-exercise2

harishjayakumar @ ~
 [51] → docker ps -a
CONTAINER ID        IMAGE                    COMMAND                  CREATED             STATUS                        PORTS               NAMES
42faecb51933        aragunathan/rubyexample   "rackup --host 0.0.0."   27 minutes ago      Exited (137) 19 minutes ago                       angry_meitner
33c3710a5eca        myfirstimage              "bash"                   5 days ago          Exited (0) 5 days ago                             myfirstcontainer
8d7cdb1f3843        aragunathan/rubyexample   "rackup --host 0.0.0."   5 days ago          Exited (137) 27 minutes ago                       tiny_raman
```

Quiz: What’s the difference between “docker stop containerID” and “docker kill containerID”

<u>Section 3: Image lifecycle</u>

Images form the root filesystem of the container. They are made of layers, conceptually stacked on top of each other.

- Each layer can add, change, and remove files.
- Images can share layers to optimize disk usage, transfer times, and memory use.
- “docker run” starts a container from a given image.

Images have namespaces too!

- root namespace. eg **docker pull ubuntu** is for official images
- user/organization namespace. Eg. **docker pull aragunathan/rubyexample** is for users hosting their images in the docker hub

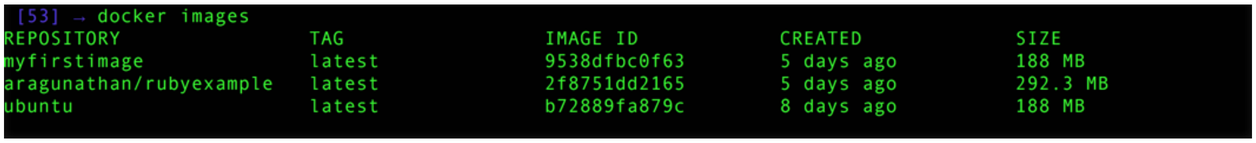
Users typically pull an official image, use it, modify if necessary and push it to their username space. This is how images are distributed.

In this exercise, we will pull images, query images and remove them.

<u>Exercise 3:</u>

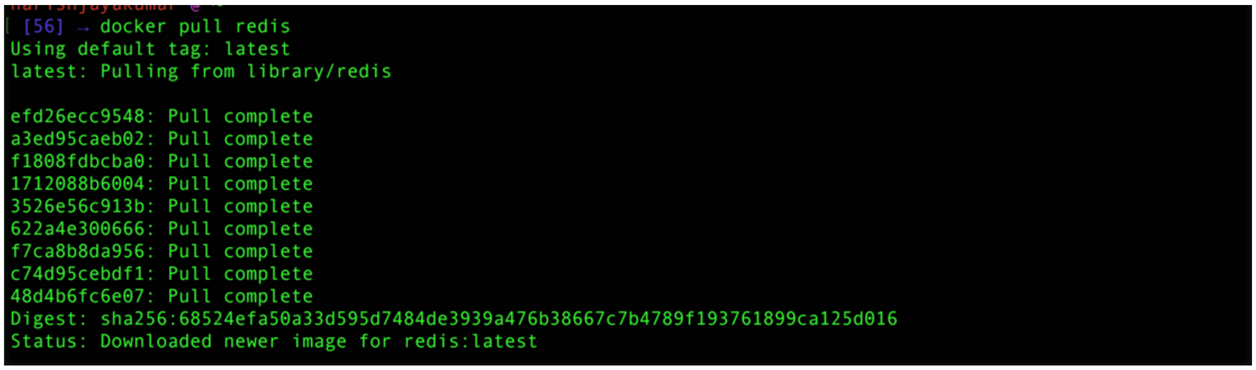
<u>3.1 List Docker images :</u>

<u>$docker images</u>



```
 [53] → docker images
REPOSITORY                 TAG                 IMAGE ID            CREATED             SIZE
myfirstimage               latest              9538dfbc0f63        5 days ago          188 MB
aragunathan/rubyexample    latest              2f8751dd2165        5 days ago          292.3 MB
ubuntu                     latest              b72889fa879c        8 days ago          188 MB
```

<u>3.2 Pull a docker image :</u>
<u>$ docker pull redis</u>



```
[ [56] → docker pull redis
 Using default tag: latest
 latest: Pulling from library/redis

 efd26ecc9548: Pull complete
 a3ed95caeb02: Pull complete
 f1808fdbcba0: Pull complete
 1712088b6004: Pull complete
 3526e56c913b: Pull complete
 622a4e300666: Pull complete
 f7ca8b8da956: Pull complete
 c74d95cebdf1: Pull complete
 48d4b6fc6e07: Pull complete
 Digest: sha256:68524efa50a33d595d7484de3939a476b38667c7b4789f193761899ca125d016
 Status: Downloaded newer image for redis:latest
```

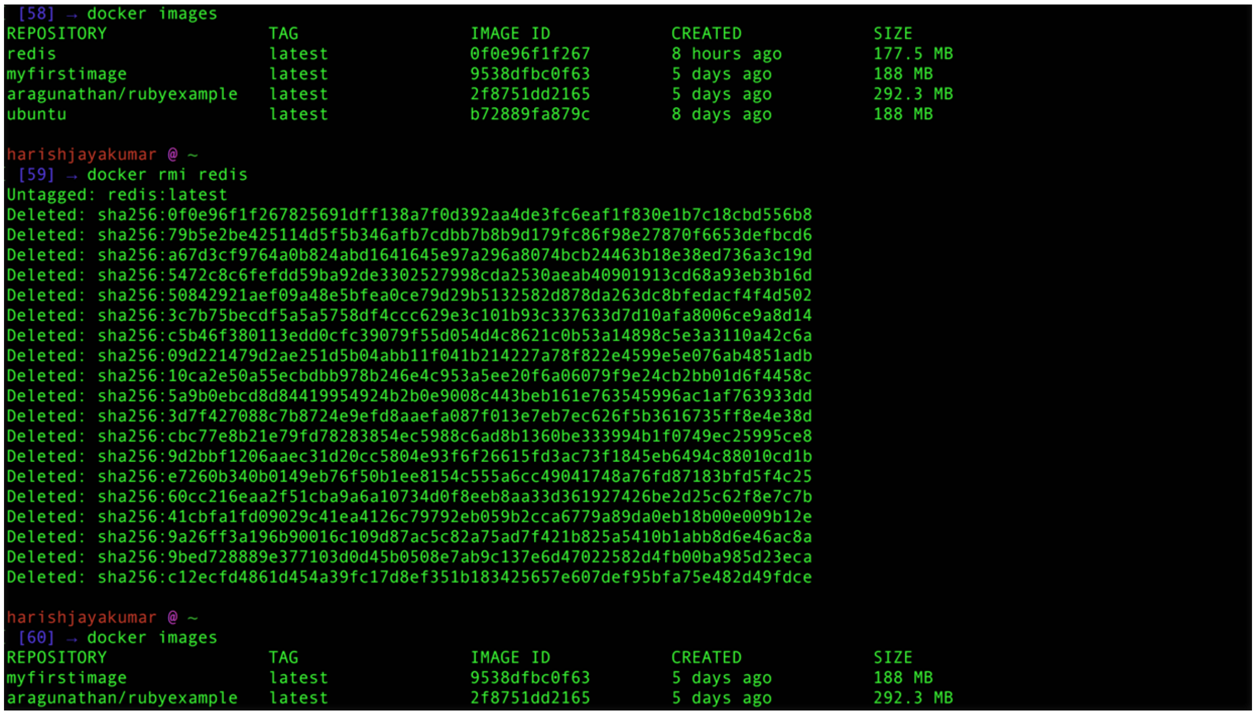
<u>Quiz: What happens when you pull the same image another time?</u>

3.2 List and Remove an image :

List $docker images

Remove $docker rmi <<image id>>

List to check image removed $ docker images



```
 [58] → docker images
REPOSITORY                 TAG                 IMAGE ID            CREATED             SIZE
redis                      latest              0f0e96f1f267        8 hours ago         177.5 MB
myfirstimage               latest              9538dfbc0f63        5 days ago          188 MB
aragunathan/rubyexample    latest              2f8751dd2165        5 days ago          292.3 MB
ubuntu                     latest              b72889fa879c        8 days ago          188 MB

harishjayakumar @ ~
 [59] → docker rmi redis
Untagged: redis:latest
Deleted: sha256:0f0e96f1f267825691dff138a7f0d392aa4de3fc6eaf1f830e1b7c18cbd556b8
Deleted: sha256:79b5e2be425114d5f5b346afb7cdbb7b8b9d179fc86f98e27870f6653defbcd6
Deleted: sha256:a67d3cf9764a0b824abd1641645e97a296a8074bcb24463b18e38ed736a3c19d
Deleted: sha256:5472c8c6fefdd59ba92de3302527998cda2530aeab40901913cd68a93eb3b16d
Deleted: sha256:50842921aef09a48e5bfea0ce79d29b5132582d878da263dc8bfedacf4f4d502
Deleted: sha256:3c7b75becdf5a5a5758df4ccc629e3c101b93c337633d7d10afa8006ce9a8d14
Deleted: sha256:c5b46f380113edd0cfc39079f55d054d4c8621c0b53a14898c5e3a3110a42c6a
Deleted: sha256:09d221479d2ae251d5b04abb11f041b214227a78f822e4599e5e076ab4851adb
Deleted: sha256:10ca2e50a55ecbdbb978b246e4c953a5ee20f6a06079f9e24cb2bb01d6f4458c
Deleted: sha256:5a9b0ebcd8d84419954924b2b0e9008c443beb161e763545996ac1af763933dd
Deleted: sha256:3d7f427088c7b8724e9efd8aaefa087f013e7eb7ec626f5b3616735ff8e4e38d
Deleted: sha256:cbc77e8b21e79fd78283854ec5988c6ad8b1360be333994b1f0749ec25995ce8
Deleted: sha256:9d2bbf1206aaec31d20cc5804e93f6f26615fd3ac73f1845eb6494c88010cd1b
Deleted: sha256:e7260b340b0149eb76f50b1ee8154c555a6cc49041748a76fd87183bfd5f4c25
Deleted: sha256:60cc216eaa2f51cba9a6a10734d0f8eeb8aa33d361927426be2d25c62f8e7c7b
Deleted: sha256:41cbfa1fd09029c41ea4126c79792eb059b2cca6779a89da0eb18b00e009b12e
Deleted: sha256:9a26ff3a196b90016c109d87ac5c82a75ad7f421b825a5410b1abb8d6e46ac8a
Deleted: sha256:9bed728889e377103d0d45b0508e7ab9c137e6d47022582d4fb00ba985d23eca
Deleted: sha256:c12ecfd4861d454a39fc17d8ef351b183425657e607def95bfa75e482d49fdce

harishjayakumar @ ~
 [60] → docker images
REPOSITORY                 TAG                 IMAGE ID            CREATED             SIZE
myfirstimage               latest              9538dfbc0f63        5 days ago          188 MB
aragunathan/rubyexample    latest              2f8751dd2165        5 days ago          292.3 MB
```

Section 4: Observing events/objects

Docker objects can be very deeply inspected. It comes handy while debugging issues or simply to understand the nature of an object (such as container/image) that has so far been a black box.

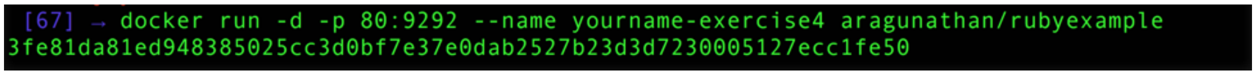
“docker inspect” will produce a detailed JSON output, that can be filtered based on what you are looking for. **“docker stats”** and **“docker events”** are powerful inspection tools as well.

Exercise 4:

We deleted the container in the previous example lets first recreate one before we start this exercise (helps you practice more!)

Create a container:

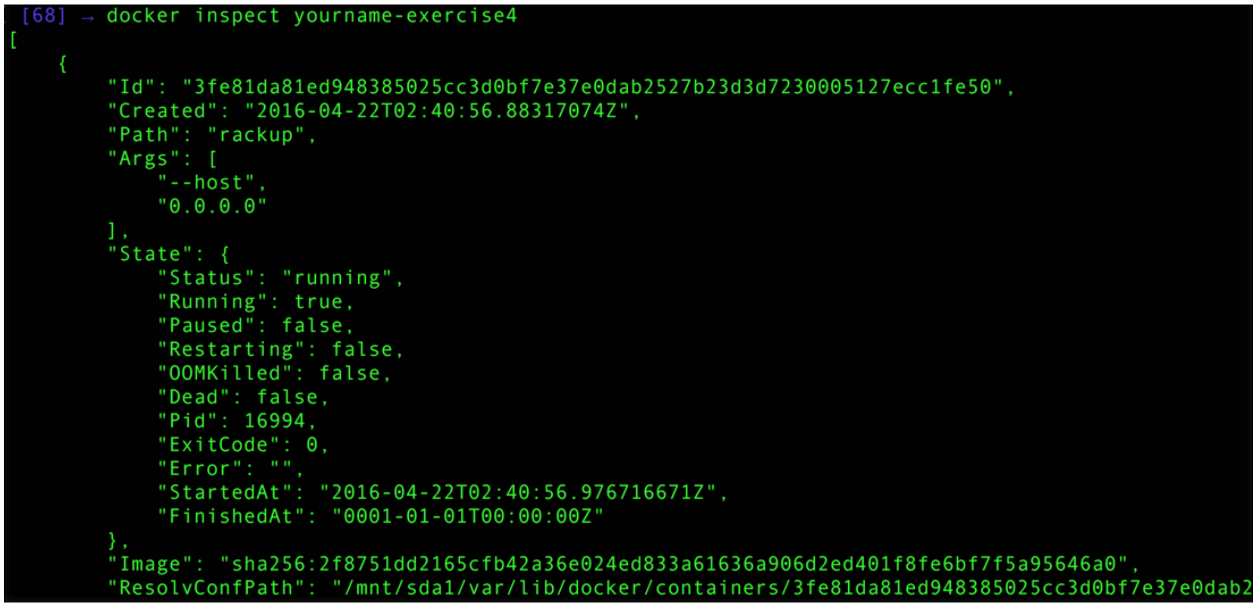
$ docker run -d -p 80:9292 --name yourname-exercise4 aragunathan/rubyexample



```
 [67] → docker run -d -p 80:9292 --name yourname-exercise4 aragunathan/rubyexample
3fe81da81ed948385025cc3d0bf7e37e0dab2527b23d3d7230005127ecc1fe50
```

4.1 Docker inspect Container:

$ docker inspect yourname-exercise4



```
 [68] → docker inspect yourname-exercise4
[
    {
        "Id": "3fe81da81ed948385025cc3d0bf7e37e0dab2527b23d3d7230005127ecc1fe50",
        "Created": "2016-04-22T02:40:56.88317074Z",
        "Path": "rackup",
        "Args": [
            "--host",
            "0.0.0.0"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 16994,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2016-04-22T02:40:56.976716671Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:2f8751dd2165cfb42a36e024ed833a61636a906d2ed401f8fe6bf7f5a95646a0",
        "ResolvConfPath": "/mnt/sda1/var/lib/docker/containers/3fe81da81ed948385025cc3d0bf7e37e0dab2
```

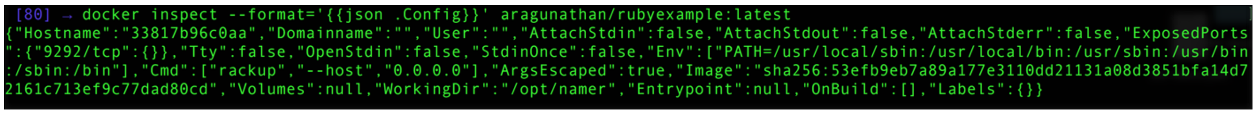
<< Image truncated>>

4.2 Docker inspect of images

Since it lists a lot of information let’s pick a specific variable?

$docker inspect --format='{{json .Config}}' aragunathan/rubyexample:latest

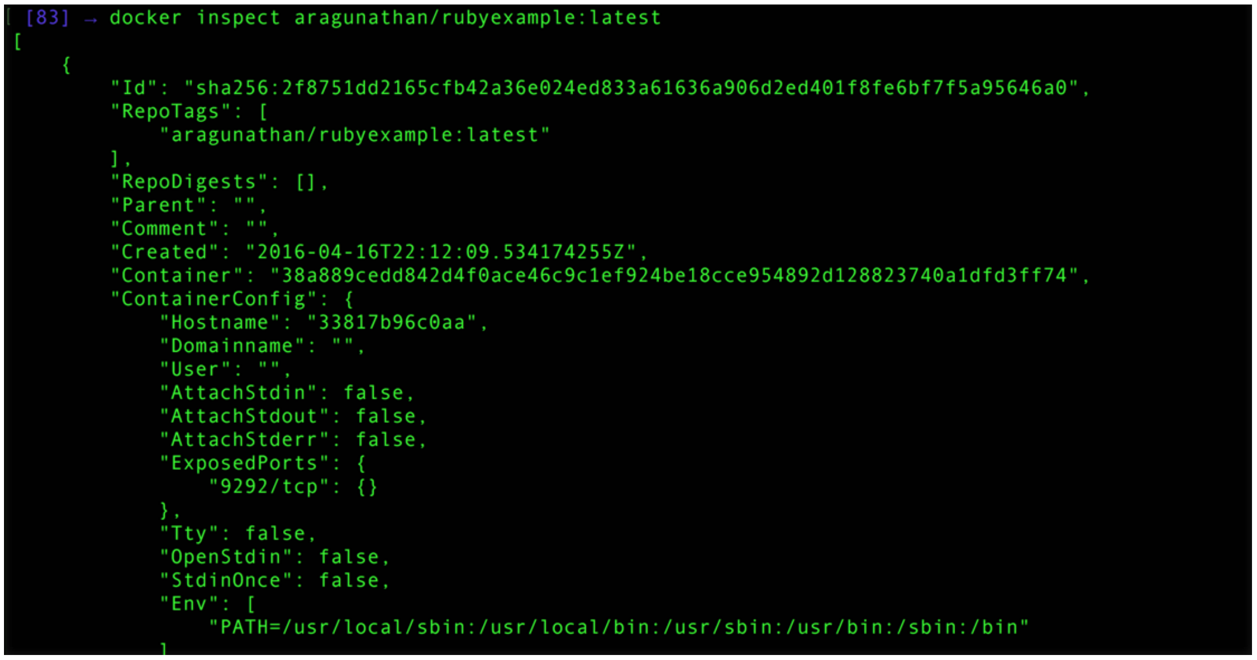
Note that **Config** here is the JSON object of interest.



```
[80] → docker inspect --format='{{json .Config}}' aragunathan/rubyexample:latest
{"Hostname":"33817b96c0aa","Domainname":"","User":"","AttachStdin":false,"AttachStdout":false,"AttachStderr":false,"ExposedPorts
":{"9292/tcp":{}},"Tty":false,"OpenStdin":false,"StdinOnce":false,"Env":["PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
:/sbin:/bin"],"Cmd":["rackup","--host","0.0.0.0"],"ArgsEscaped":true,"Image":"sha256:53efb9eb7a89a177e3110dd21131a08d3851bfa14d7
2161c713ef9c77dad80cd","Volumes":null,"WorkingDir":"/opt/namer","Entrypoint":null,"OnBuild":[],"Labels":{}}
```

You can always do a docker inspect to see the whole information

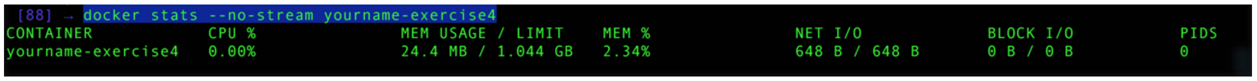
$docker inspect aragunathan/rubyexample:latest



```
[83] → docker inspect aragunathan/rubyexample:latest
[
    {
        "Id": "sha256:2f8751dd2165cfb42a36e024ed833a61636a906d2ed401f8fe6bf7f5a95646a0",
        "RepoTags": [
            "aragunathan/rubyexample:latest"
        ],
        "RepoDigests": [],
        "Parent": "",
        "Comment": "",
        "Created": "2016-04-16T22:12:09.534174255Z",
        "Container": "38a889cedd842d4f0ace46c9c1ef924be18cce954892d128823740a1dfd3ff74",
        "ContainerConfig": {
            "Hostname": "33817b96c0aa",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "9292/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ]
```

Docker stats:

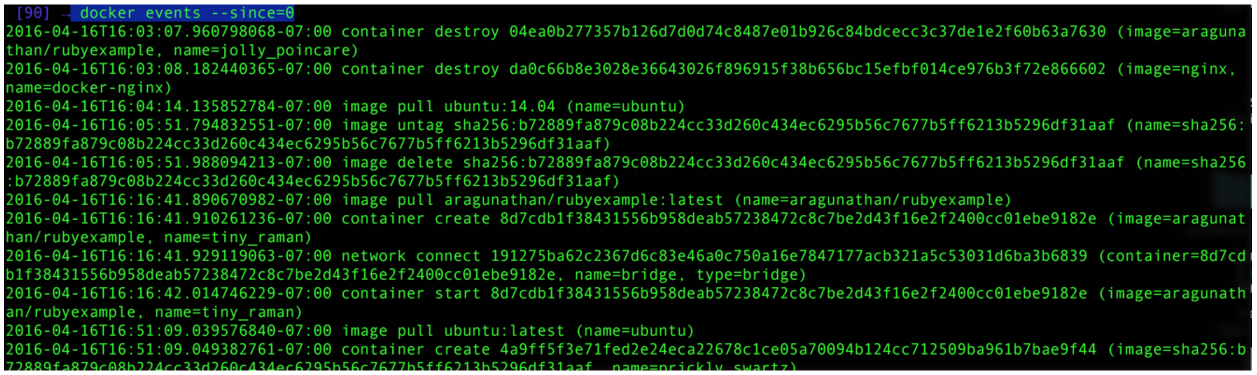
$ docker stats --no-stream yourname-exercise4



```
[88] → docker stats --no-stream yourname-exercise4
CONTAINER            CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
yourname-exercise4   0.00%               24.4 MB / 1.044 GB    2.34%               648 B / 648 B       0 B / 0 B           0
```

Docker events:

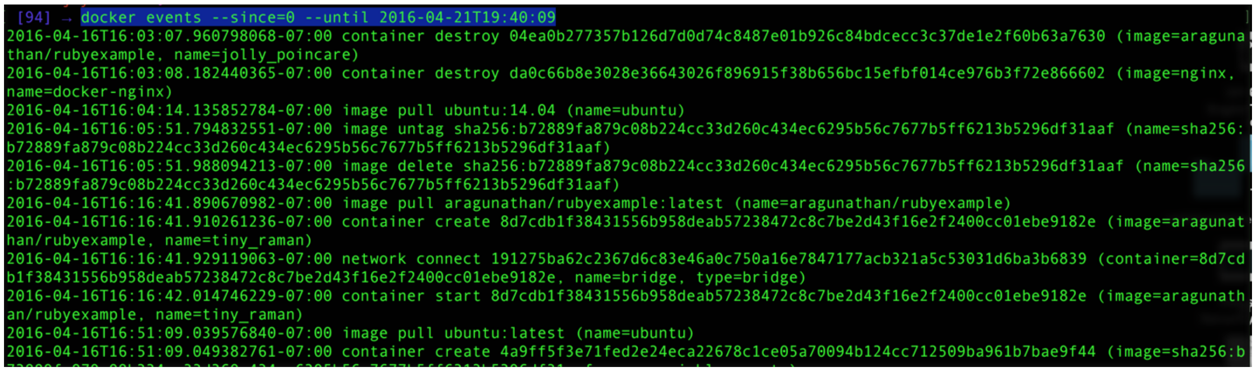
$ docker events --since=0



```
[90] → docker events --since=0
2016-04-16T16:03:07.960798068-07:00 container destroy 04ea0b277357b126d7d0d74c8487e01b926c84bdcecc3c37de1e2f60b63a7630 (image=araguna
than/rubyexample, name=jolly_poincare)
2016-04-16T16:03:08.182440365-07:00 container destroy da0c66b8e3028e36643026f896915f38b656bc15efbf014ce976b3f72e866602 (image=nginx,
name=docker-nginx)
2016-04-16T16:04:14.135852784-07:00 image pull ubuntu:14.04 (name=ubuntu)
2016-04-16T16:05:51.794832551-07:00 image untag sha256:b72889fa879c08b224cc33d260c434ec6295b56c7677b5ff6213b5296df31aaf (name=sha256:
b72889fa879c08b224cc33d260c434ec6295b56c7677b5ff6213b5296df31aaf)
2016-04-16T16:05:51.988094213-07:00 image delete sha256:b72889fa879c08b224cc33d260c434ec6295b56c7677b5ff6213b5296df31aaf (name=sha256
:b72889fa879c08b224cc33d260c434ec6295b56c7677b5ff6213b5296df31aaf)
2016-04-16T16:16:41.890670982-07:00 image pull aragunathan/rubyexample:latest (name=aragunathan/rubyexample)
2016-04-16T16:16:41.910261236-07:00 container create 8d7cdb1f38431556b958deab57238472c8c7be2d43f16e2f2400cc01ebe9182e (image=aragunat
han/rubyexample, name=tiny_raman)
2016-04-16T16:16:41.929119063-07:00 network connect 191275ba62c2367d6c83e46a0c750a16e7847177acb321a5c53031d6ba3b6839 (container=8d7cd
b1f38431556b958deab57238472c8c7be2d43f16e2f2400cc01ebe9182e, name=bridge, type=bridge)
2016-04-16T16:16:42.014746229-07:00 container start 8d7cdb1f38431556b958deab57238472c8c7be2d43f16e2f2400cc01ebe9182e (image=aragunath
an/rubyexample, name=tiny_raman)
2016-04-16T16:51:09.039576840-07:00 image pull ubuntu:latest (name=ubuntu)
2016-04-16T16:51:09.049382761-07:00 container create 4a9ff5f3e71fed2e24eca22678c1ce05a70094b124cc712509ba961b7bae9f44 (image=sha256:b
72889fa879c08b224cc33d260c434ec6295b56c7677b5ff6213b5296df31aaf, name=prickly_swartz)
```

$

docker events --since=0 --until 2016-04-21T19:40:09



```
 [94] → docker events --since=0 --until 2016-04-21T19:40:09
2016-04-16T16:03:07.960798068-07:00 container destroy 04ea0b277357b126d7d0d74c8487e01b926c84bdcecc3c37de1e2f60b63a7630 (image=araguna
than/rubyexample, name=jolly_poincare)
2016-04-16T16:03:08.182440365-07:00 container destroy da0c66b8e3028e36643026f896915f38b656bc15efbf014ce976b3f72e866602 (image=nginx,
name=docker-nginx)
2016-04-16T16:04:14.135852784-07:00 image pull ubuntu:14.04 (name=ubuntu)
2016-04-16T16:05:51.794832551-07:00 image untag sha256:b72889fa879c08b224cc33d260c434ec6295b56c7677b5ff6213b5296df31aaf (name=sha256:
b72889fa879c08b224cc33d260c434ec6295b56c7677b5ff6213b5296df31aaf)
2016-04-16T16:05:51.988094213-07:00 image delete sha256:b72889fa879c08b224cc33d260c434ec6295b56c7677b5ff6213b5296df31aaf (name=sha256
:b72889fa879c08b224cc33d260c434ec6295b56c7677b5ff6213b5296df31aaf)
2016-04-16T16:16:41.890670982-07:00 image pull aragunathan/rubyexample:latest (name=aragunathan/rubyexample)
2016-04-16T16:16:41.910261236-07:00 container create 8d7cdb1f38431556b958deab57238472c8c7be2d43f16e2f2400cc01ebe9182e (image=aragunat
han/rubyexample, name=tiny_raman)
2016-04-16T16:16:41.929119063-07:00 network connect 191275ba62c2367d6c83e46a0c750a16e7847177acb321a5c53031d6ba3b6839 (container=8d7cd
b1f38431556b958deab57238472c8c7be2d43f16e2f2400cc01ebe9182e, name=bridge, type=bridge)
2016-04-16T16:16:42.014746229-07:00 container start 8d7cdb1f38431556b958deab57238472c8c7be2d43f16e2f2400cc01ebe9182e (image=aragunath
an/rubyexample, name=tiny_raman)
2016-04-16T16:51:09.039576840-07:00 image pull ubuntu:latest (name=ubuntu)
2016-04-16T16:51:09.049382761-07:00 container create 4a9ff5f3e71fed2e24eca22678c1ce05a70094b124cc712509ba961b7bae9f44 (image=sha256:b
```

<u>Section 5: Auditing logs</u>

All container logs can be viewed using the “docker logs” command. Again, very handy for debugging.

You can also try variations to logs such as:

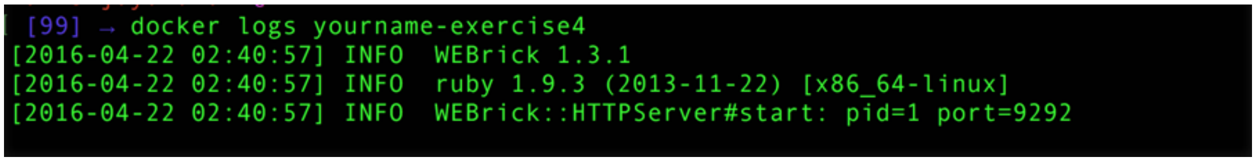
--tail n to observe the last n lines

-- tail -f to follow the logs in real time

<u>Exercise 5:</u>

<u>Print the logs for your container:</u>

```
$ docker logs yourname-exercise4
```



```
 [99] → docker logs yourname-exercise4
[2016-04-22 02:40:57] INFO  WEBrick 1.3.1
[2016-04-22 02:40:57] INFO  ruby 1.9.3 (2013-11-22) [x86_64-linux]
[2016-04-22 02:40:57] INFO  WEBrick::HTTPServer#start: pid=1 port=9292
```

Section 6: Volume subsystem

Volumes allow data to be shared between containers and between host and containers.

- They are special directories mounted in a container
- They are not part of the COW filesystem and hence not part of the image.

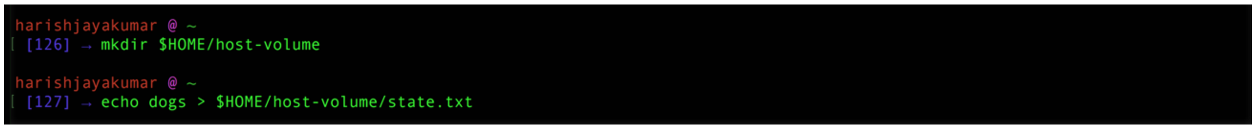
In this exercise, we will see how to export host volumes to containers and use **"docker inspect"**
on volumes. Note that docker volumes are always mounted read-write, but can be overridden to be read-only.

Exercise 6:

6.1 Exporting Host volumes to containers

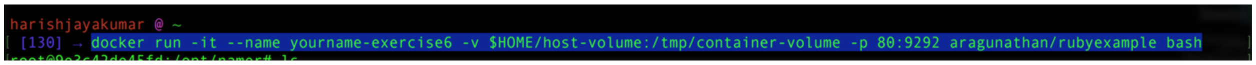
Remove your existing containers and let's start afresh (You should know how to remove them by now :)) After you remove them do a docker ps to see no containers are running. Then
$mkdir $HOME/host-volume

$echo dogs > $HOME/host-volume/state.txt



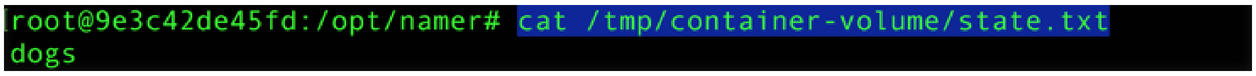
```
harishjayakumar @ ~
[ [126] → mkdir $HOME/host-volume

harishjayakumar @ ~
[ [127] → echo dogs > $HOME/host-volume/state.txt
```

Mount the host volume onto your container
$docker run -it --name yourname-exercise6 -v $HOME/host-volume:/tmp/container-volume -p 80:9292 aragunathan/rubyexample bash



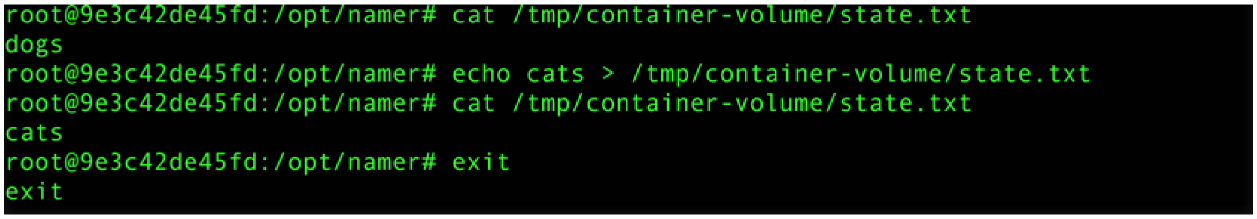
```
harishjayakumar @ ~
[ [130] → docker run -it --name yourname-exercise6 -v $HOME/host-volume:/tmp/container-volume -p 80:9292 aragunathan/rubyexample bash
```

Access the file inside the container
$cat /tmp/container-volume/state.txt



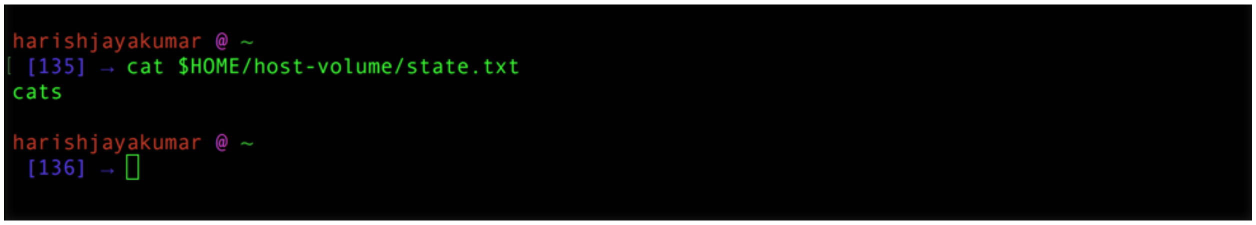
```
root@9e3c42de45fd:/opt/namer# cat /tmp/container-volume/state.txt
dogs
```

$echo cats > /tmp/container-volume/state.txt
$cat /tmp/container-volume/state.txt
$exit



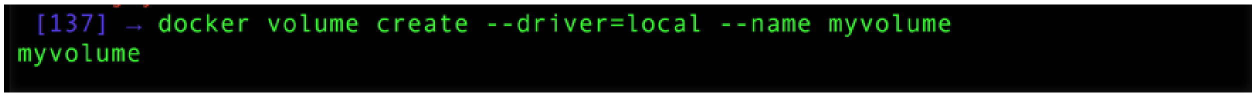
```
root@9e3c42de45fd:/opt/namer# cat /tmp/container-volume/state.txt
dogs
root@9e3c42de45fd:/opt/namer# echo cats > /tmp/container-volume/state.txt
root@9e3c42de45fd:/opt/namer# cat /tmp/container-volume/state.txt
cats
root@9e3c42de45fd:/opt/namer# exit
exit
```

cat $HOME/host-volume/state.txt

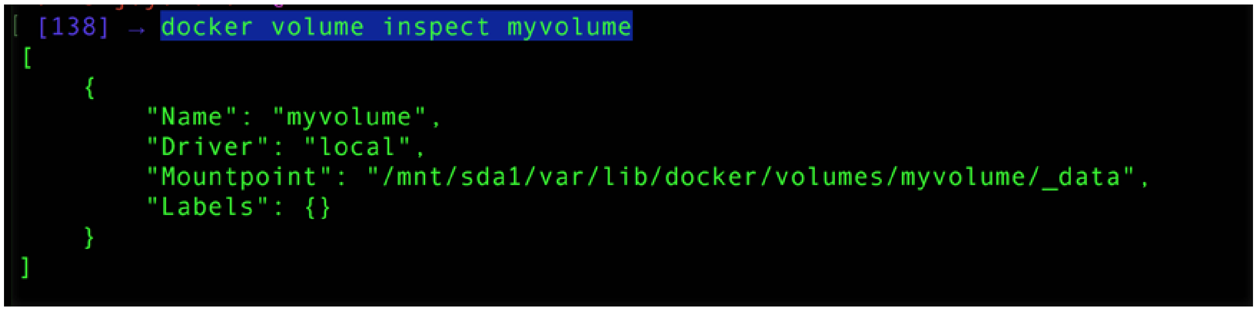

```
harishjayakumar @ ~
[ [135] → cat $HOME/host-volume/state.txt
cats

harishjayakumar @ ~
 [136] → 
```

6.2 Docker volume create/ls/remove/inspect

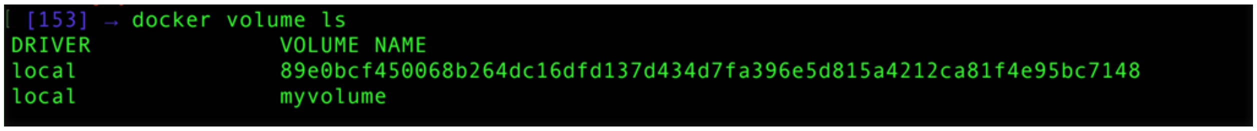
$ docker volume create --driver=local --name myvolume


```
  [137] → docker volume create --driver=local --name myvolume
myvolume
```

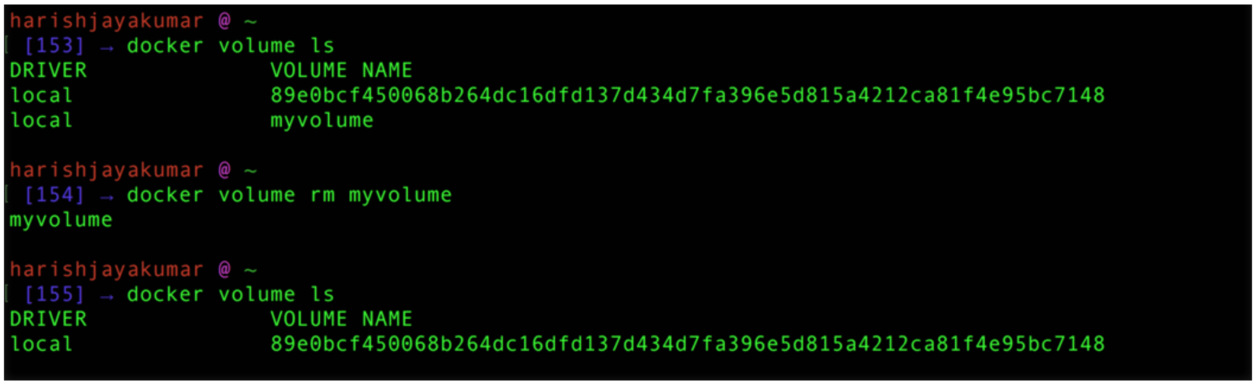
$docker volume inspect myvolume


```
[ [138] → docker volume inspect myvolume
[
    {
        "Name": "myvolume",
        "Driver": "local",
        "Mountpoint": "/mnt/sda1/var/lib/docker/volumes/myvolume/_data",
        "Labels": {}
    }
]
```

$docker volume ls


```
[ [153] → docker volume ls
DRIVER              VOLUME NAME
local               89e0bcf450068b264dc16dfd137d434d7fa396e5d815a4212ca81f4e95bc7148
local               myvolume
```

$docker volume rm myvolume
$docker volume ls


```
harishjayakumar @ ~
[ [153] → docker volume ls
DRIVER              VOLUME NAME
local               89e0bcf450068b264dc16dfd137d434d7fa396e5d815a4212ca81f4e95bc7148
local               myvolume

harishjayakumar @ ~
[ [154] → docker volume rm myvolume
myvolume

harishjayakumar @ ~
[ [155] → docker volume ls
DRIVER              VOLUME NAME
local               89e0bcf450068b264dc16dfd137d434d7fa396e5d815a4212ca81f4e95bc7148
```

Section 7: Docker Networking

Docker allow users to create networks and connect containers to them. Using this feature, we can connect containers on the same host or across hosts. In this exercise, we will focus on same host container connectivity.

Bridged: This is the default mode in which the docker daemon work and uses Linux Ethernet bridge. Do an ifconfig on your system to see your **docker0.**

None: If containers need to be run without a network, use the --net=none option **docker run --net=none ubuntu bash** and observe the output of ifconfig.

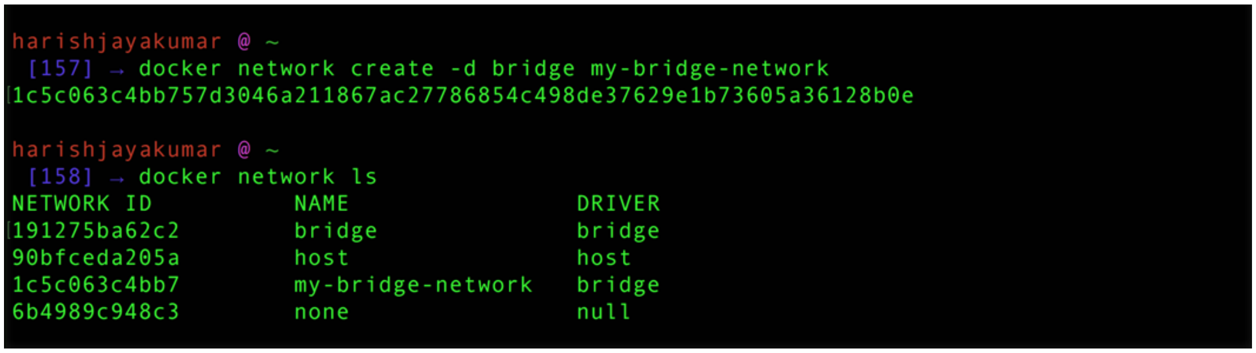
Host: Adds containers to the host networking stack. The container **ifconfig** will be identical to that of the host.

In this exercise, we will create networks and attach containers to it. We will also inspect network objects. You can try to connect more than 1 container on the bridge and test connectivity (using **ping**).

Exercise 7:

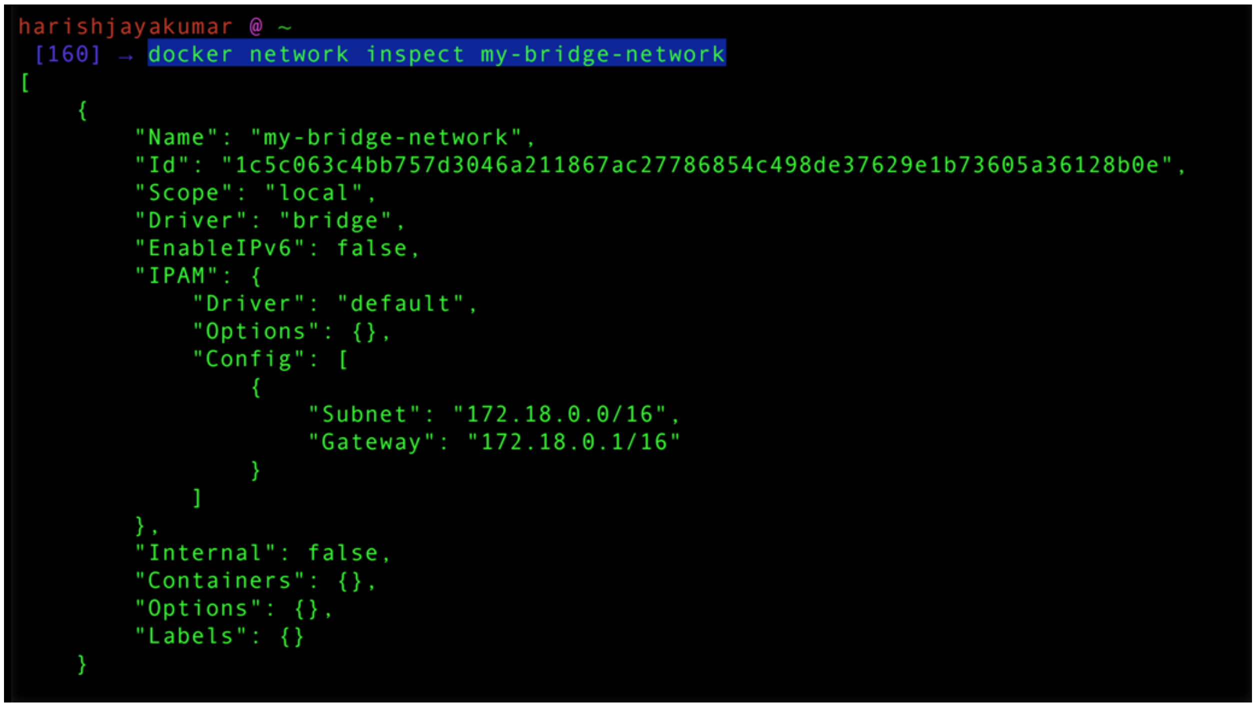
7.1 Create, list and inspect network

Create : $ docker network create -d bridge my-bridge-network
List : $ docker network ls



```
harishjayakumar @ ~
 [157] → docker network create -d bridge my-bridge-network
1c5c063c4bb757d3046a211867ac27786854c498de37629e1b73605a36128b0e

harishjayakumar @ ~
 [158] → docker network ls
NETWORK ID          NAME                DRIVER
191275ba62c2        bridge              bridge
90bfceda205a        host                host
1c5c063c4bb7        my-bridge-network   bridge
6b4989c948c3        none                null
```

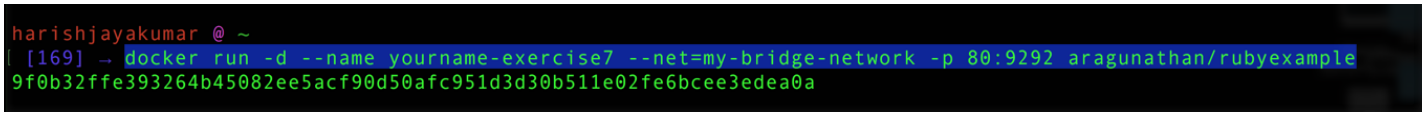
Inspect: $ docker network inspect mybridge-network



```
harishjayakumar @ ~
 [160] → docker network inspect my-bridge-network
[
    {
        "Name": "my-bridge-network",
        "Id": "1c5c063c4bb757d3046a211867ac27786854c498de37629e1b73605a36128b0e",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1/16"
                }
            ]
        },
        "Internal": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
```

7.2 Use Network

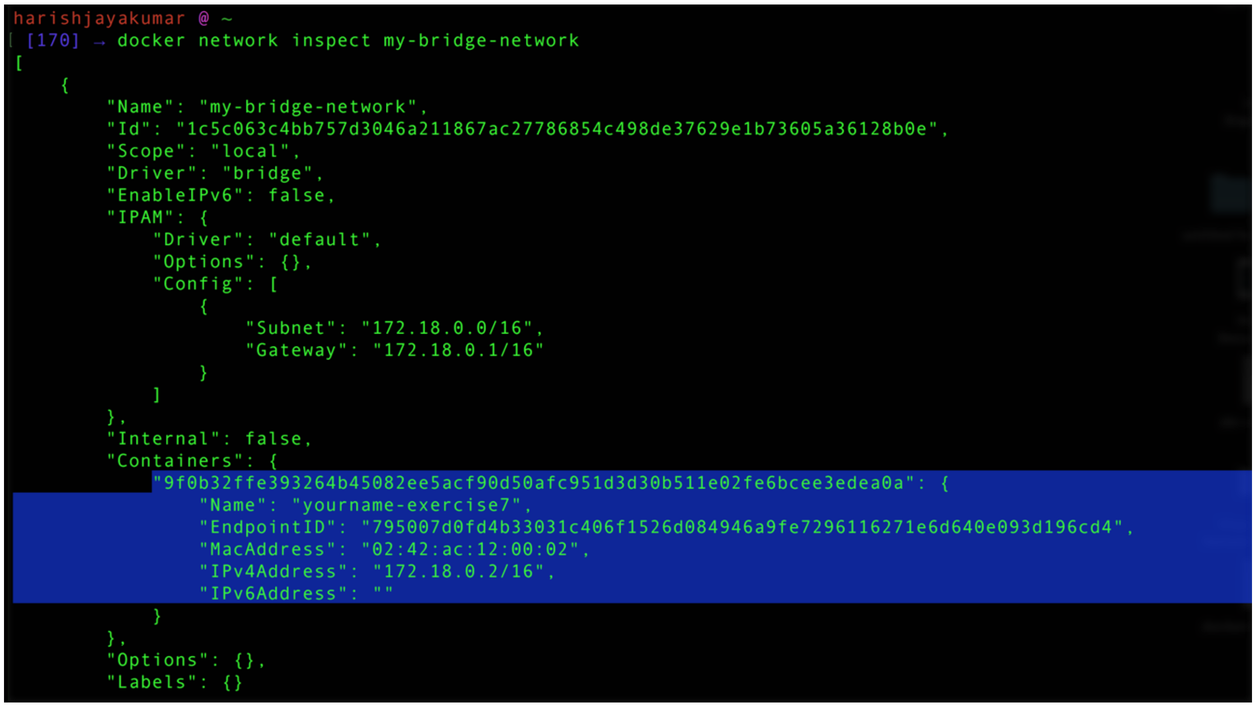
$docker run -d --name yourname-exercise7 --net=my-bridge-network -p 80:9292 aragunathan/rubyexample



```
harishjayakumar @ ~
[ [169] → docker run -d --name yourname-exercise7 --net=my-bridge-network -p 80:9292 aragunathan/rubyexample
9f0b32ffe393264b45082ee5acf90d50afc951d3d30b511e02fe6bcee3edea0a
```

Inspect network and see your container in it

$docker network inspect my-bridge-network

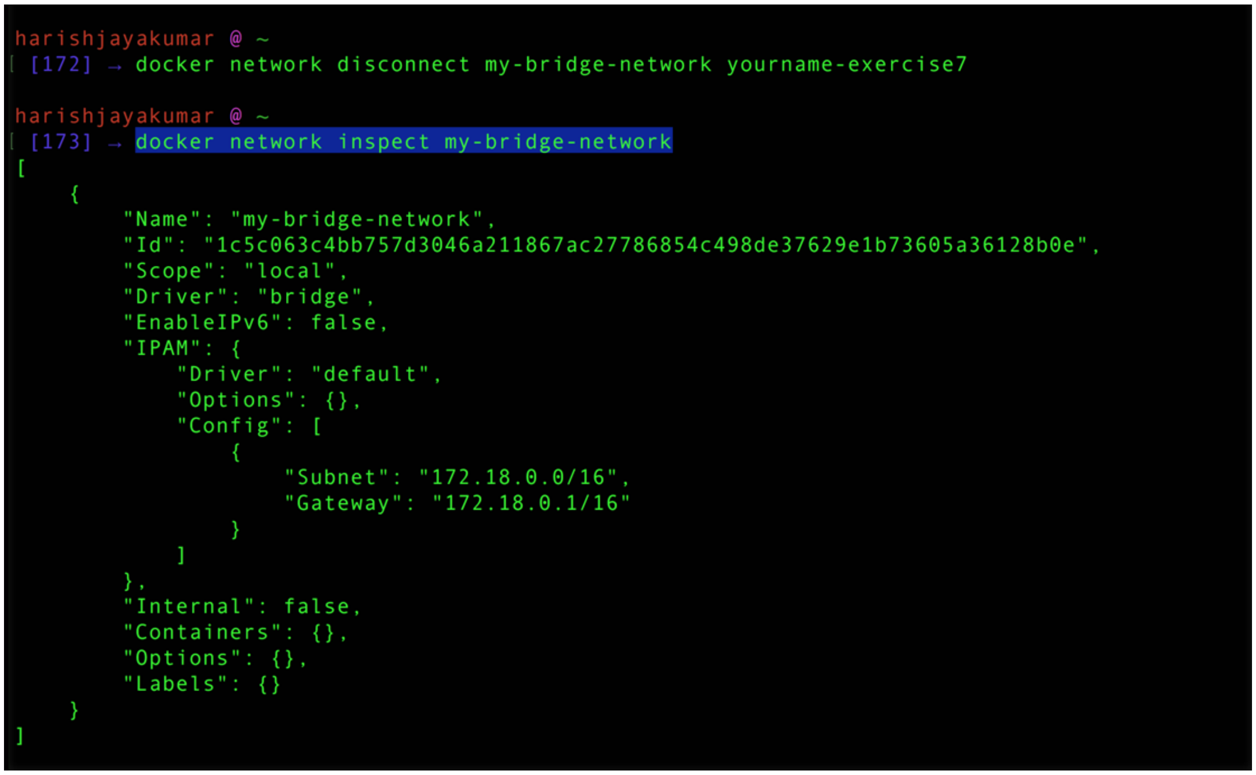


```
harishjayakumar @ ~
[ [170] → docker network inspect my-bridge-network
[
    {
        "Name": "my-bridge-network",
        "Id": "1c5c063c4bb757d3046a211867ac27786854c498de37629e1b73605a36128b0e",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1/16"
                }
            ]
        },
        "Internal": false,
        "Containers": {
            "9f0b32ffe393264b45082ee5acf90d50afc951d3d30b511e02fe6bcee3edea0a": {
                "Name": "yourname-exercise7",
                "EndpointID": "795007d0fd4b33031c406f1526d084946a9fe7296116271e6d640e093d196cd4",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
```

7.3 Disconnect container from network

$docker network disconnect my-bridge-network yourname-exercise7

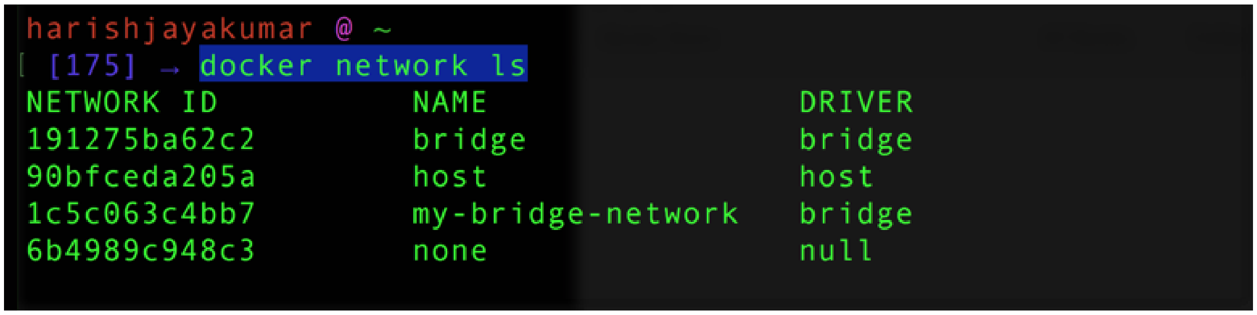
$docker network inspect my-bridge-network



```
harishjayakumar @ ~
[ [172] → docker network disconnect my-bridge-network yourname-exercise7

harishjayakumar @ ~
[ [173] → docker network inspect my-bridge-network
[
    {
        "Name": "my-bridge-network",
        "Id": "1c5c063c4bb757d3046a211867ac27786854c498de37629e1b73605a36128b0e",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1/16"
                }
            ]
        },
        "Internal": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

7.4 List networks

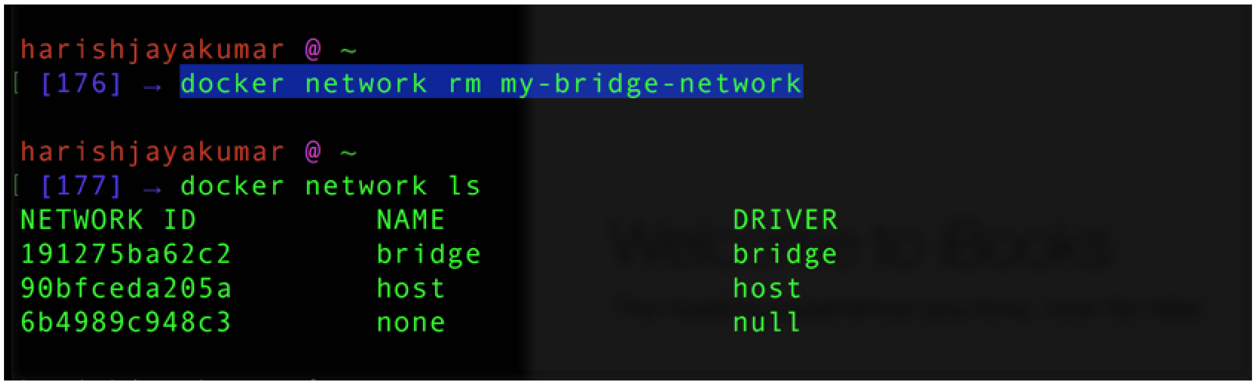
$docker network ls



```
harishjayakumar @ ~
[ [175] → docker network ls
NETWORK ID          NAME                DRIVER
191275ba62c2        bridge              bridge
90bfceda205a        host                host
1c5c063c4bb7        my-bridge-network   bridge
6b4989c948c3        none                null
```

7.5 Destroy network

$docker network rm my-bridge-network
$ docker network ls



```
harishjayakumar @ ~
[ [176] → docker network rm my-bridge-network

harishjayakumar @ ~
[ [177] → docker network ls
NETWORK ID          NAME                DRIVER
191275ba62c2        bridge              bridge
90bfceda205a        host                host
6b4989c948c3        none                null
```

<u>Section 8: Docker Hub</u>

Hub.docker.com is Docker Hub, which stores docker's official images as well as user created images. This section introduces you to the concept of the Hub.

You will be able to access the Hub, once you create an account. We highly encourage you to do so and play with it. Once you start creating images, you can distribute it through the Hub. As discussed in the image section, there are two types of repositories:

- Official repositories: maintained by Docker Inc for official images
- Namespace repos: These are created by users like you to upload their images for distribution. And yes, signing up is free.

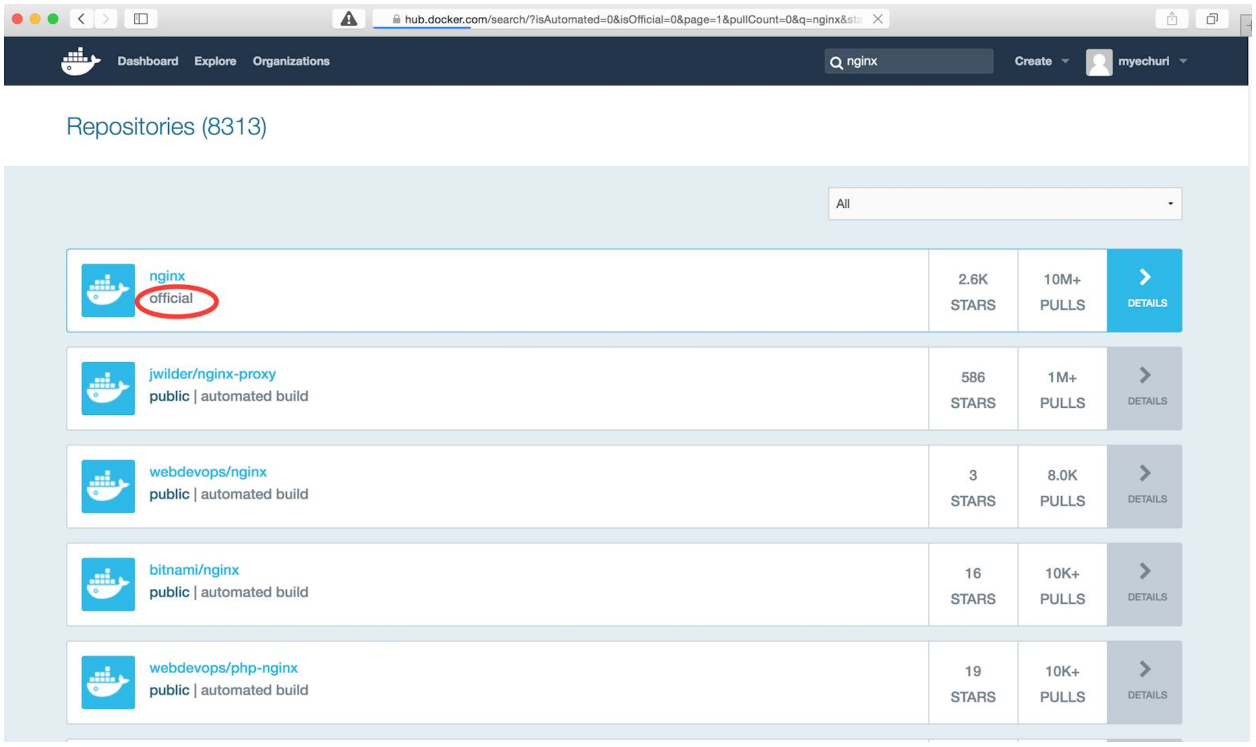
<u>Exercise 8:</u>

8.1 docker search

Docker search is a powerful tool to search the registry, once you **"docker login"** using your Hub account.

```
docker@default:~/namer$ docker search aragunathan/rubyexample
NAME                        DESCRIPTION   STARS     OFFICIAL   AUTOMATED
aragunathan/rubyexample                   0
docker@default:~/namer$ █
```

8.2 Hub UI, official images


hub.docker.com/search/?isAutomated=0&isOfficial=0&page=1&pullCount=0&q=nginx&sta
Dashboard
Explore
Organizations
nginx
Create
myechuri
All

Repositories (8313)

nginx official	2.6K STARS	10M+ PULLS	DETAILS
jwilder/nginx-proxy public | automated build	586 STARS	1M+ PULLS	DETAILS
webdevops/nginx public | automated build	3 STARS	8.0K PULLS	DETAILS
bitnami/nginx public | automated build	16 STARS	10K+ PULLS	DETAILS
webdevops/php-nginx public | automated build	19 STARS	10K+ PULLS	DETAILS

Section 9. Building images using Dockerfile

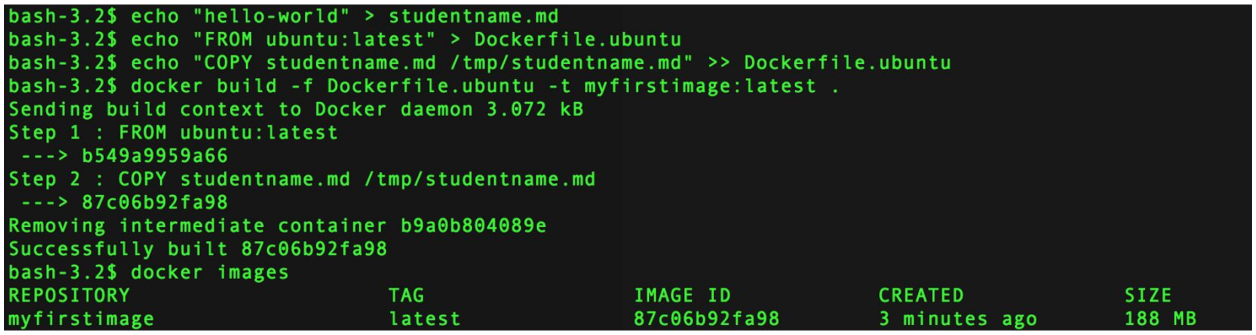
Dockerfiles provide instructions to build images. They are similar to Makefile and have a well-defined format. In this exercise, we will write a Dockerfile and build a container image using it.

Dockerfile commands cannot be interactive. Each line has an instruction to build the image. The first line of the file should be a “FROM” instruction. Each line in the DockerFile will be executed during the build and produces a layer. **“docker build”** command is used to build and tag images.

Exercise 9

Create a Dockerfile with a local file, build custom image:

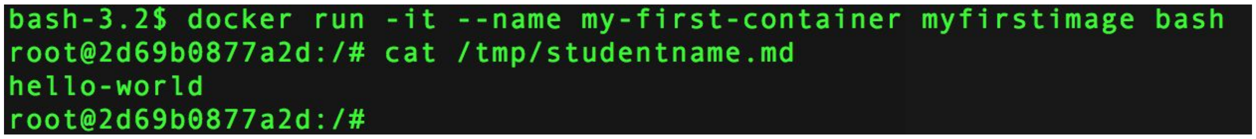
echo “hello-world” > studentname.md
echo “FROM ubuntu:latest” > Dockerfile.ubuntu
echo “COPY studentname.md /tmp/studentname.md” >> Dockerfile.ubuntu
docker build -f Dockerfile.ubuntu -t myfirstimage:latest
docker images



```
bash-3.2$ echo "hello-world" > studentname.md
bash-3.2$ echo "FROM ubuntu:latest" > Dockerfile.ubuntu
bash-3.2$ echo "COPY studentname.md /tmp/studentname.md" >> Dockerfile.ubuntu
bash-3.2$ docker build -f Dockerfile.ubuntu -t myfirstimage:latest .
Sending build context to Docker daemon 3.072 kB
Step 1 : FROM ubuntu:latest
 ---> b549a9959a66
Step 2 : COPY studentname.md /tmp/studentname.md
 ---> 87c06b92fa98
Removing intermediate container b9a0b804089e
Successfully built 87c06b92fa98
bash-3.2$ docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
myfirstimage                  latest              87c06b92fa98        3 minutes ago       188 MB
```

Create a container using custom build image, then validate.

docker run -it --name my-first-container myfirstimage bash
cat /tmp/studentname.md



```
bash-3.2$ docker run -it --name my-first-container myfirstimage bash
root@2d69b0877a2d:/# cat /tmp/studentname.md
hello-world
root@2d69b0877a2d:/#
```

<u>Section 10: Updating rubyexample app</u>

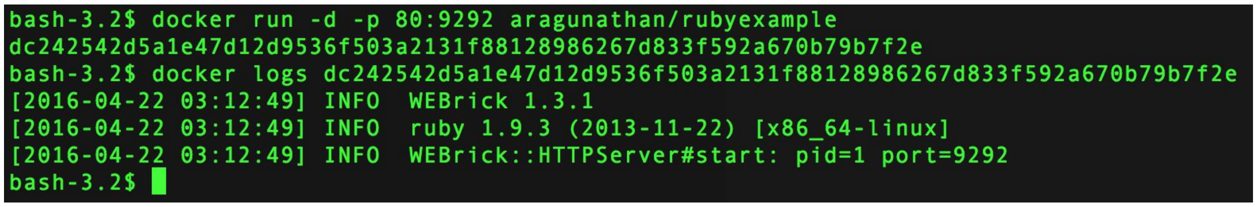
Using a lot of what we've learned in this workshop, we will peek into the rubyapp container, update it and see the updates be deployed realtime.

We will update a small configuration of our app in the container and notice how easy it was to deploy the update to the app.

Exercise 10.

<u>10.1 Start Ruby example application</u>

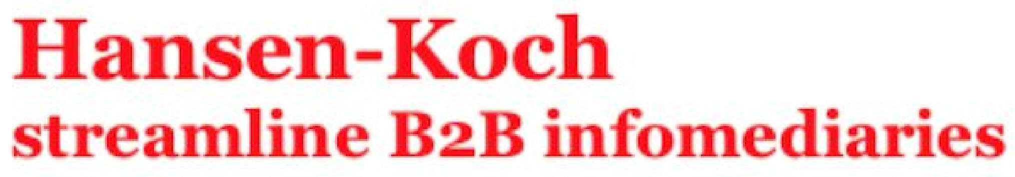
$ docker run -d -p 80:9292 --name aragunathan/rubyexample



```
bash-3.2$ docker run -d -p 80:9292 aragunathan/rubyexample
dc242542d5a1e47d12d9536f503a2131f88128986267d833f592a670b79b7f2e
bash-3.2$ docker logs dc242542d5a1e47d12d9536f503a2131f88128986267d833f592a670b79b7f2e
[2016-04-22 03:12:49] INFO  WEBrick 1.3.1
[2016-04-22 03:12:49] INFO  ruby 1.9.3 (2013-11-22) [x86_64-linux]
[2016-04-22 03:12:49] INFO  WEBrick::HTTPServer#start: pid=1 port=9292
bash-3.2$
```

<u>10.2 Verify that the application is up and running</u>

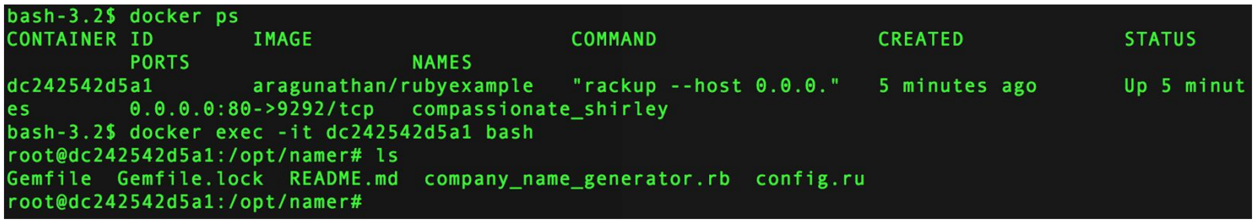
Point your browser to http://<your_host_IP>:80
Look for application text in red.



Hansen-Koch
streamline B2B infomediaries

<u>10.3 Get a bash shell inside the container</u>

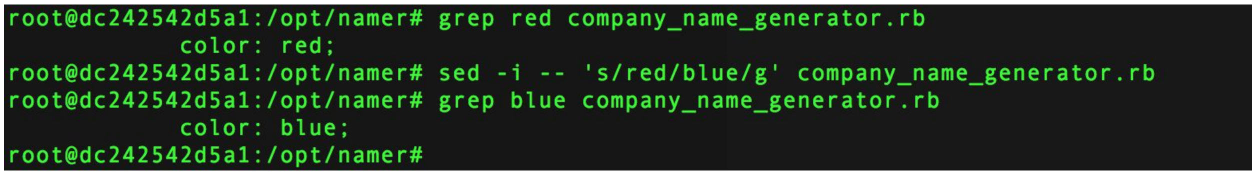
docker exec -it <rubyexample-container-id> bash



```
bash-3.2$ docker ps
CONTAINER ID        IMAGE                        COMMAND                  CREATED             STATUS
            PORTS                  NAMES
dc242542d5a1        aragunathan/rubyexample    "rackup --host 0.0.0."   5 minutes ago       Up 5 minut
es          0.0.0.0:80->9292/tcp   compassionate_shirley
bash-3.2$ docker exec -it dc242542d5a1 bash
root@dc242542d5a1:/opt/namer# ls
Gemfile  Gemfile.lock  README.md  company_name_generator.rb  config.ru
root@dc242542d5a1:/opt/namer#
```

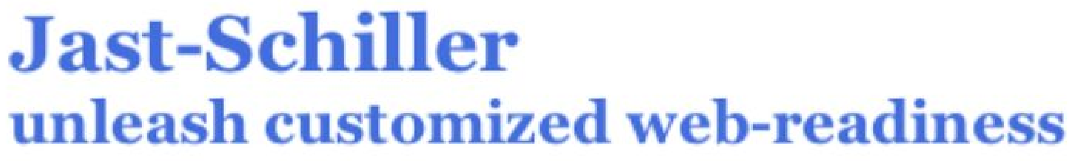
10.4 Change the color of application text from **blue** to **red**

sed -i -- ’s/red/blue/g’ company_name_generator.rb



```
root@dc242542d5a1:/opt/namer# grep red company_name_generator.rb
            color: red;
root@dc242542d5a1:/opt/namer# sed -i -- 's/red/blue/g' company_name_generator.rb
root@dc242542d5a1:/opt/namer# grep blue company_name_generator.rb
            color: blue;
root@dc242542d5a1:/opt/namer#
```

10.5 Verify application changed color on browser



Jast-Schiller
unleash customized web-readiness

“ Congratulations. You have been Dockerized! “