Getting Familiar

Commands Reference:
https://gist.github.com/initcron/5dcd6d2fb031ade5096d

Objectives

- Getting familiar with docker shell
- Finding help, basic operations
- Running Ephemeral Container
- Running Interactive Container

$ docker

```
Commands:
    attach    Attach 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 from a container's filesystem to the host
 path
    create    Create a new container
    diff      Inspect changes on a container's filesystem
    events    Get real time events from the server
    exec      Run a command in an existing container
    export    Stream the contents of a container as a tar archive
    history   Show the history of an image
    images    List images
    import    Create a new filesystem image from the contents of a tarball
    info      Display system-wide information
    inspect   Return low-level information on a container
    kill      Kill a running container
    load      Load an image from a tar archive
    login     Register or log in to a Docker registry server
    logout    Log out from a Docker registry server
    logs      Fetch the logs of a container
    port      Lookup the public-facing port that is NAT-ed to PRIVATE_PORT
    pause     Pause all processes within a container
    ps        List containers
    pull      Pull an image or a repository from a Docker registry server
    push      Push an image or a repository to a Docker registry server
    restart   Restart a running container
    rm        Remove one or more containers
    rmi       Remove one or more images
    run       Run a command in a new container
    save      Save an image to a tar archive
    search    Search for an image on the Docker Hub
    start     Start a stopped container
    stop      Stop a running container
    tag       Tag an image into a repository
    top       Lookup the running processes of a container
```

Finding Help

Syntax => docker <command> --help

e.g.

```
$ docker login --help
```

```
bash-3.2$ docker login --help

Usage: docker login [OPTIONS] [SERVER]

Register or log in to a Docker registry server, if no server is specified
"https://index.docker.io/v1/" is the default.

  -e, --email=""       Email
  -p, --password=""    Password
  -u, --username=""    Username
```

Display System Wide Info

```
$ docker info
```

```
bash-3.2$ docker info
Containers: 32
Images: 46
Storage Driver: aufs
 Root Dir: /mnt/sda1/var/lib/docker/aufs
 Dirs: 111
Execution Driver: native-0.2
Kernel Version: 3.16.7-tinycore64
Operating System: Boot2Docker 1.3.2 (TCL
24 20:40:58 UTC 2014
Debug mode (server): true
Debug mode (client): false
Fds: 11
Goroutines: 13
EventsListeners: 0
Init Path: /usr/local/bin/docker
Username: initcron
Registry: [https://index.docker.io/v1/]
```

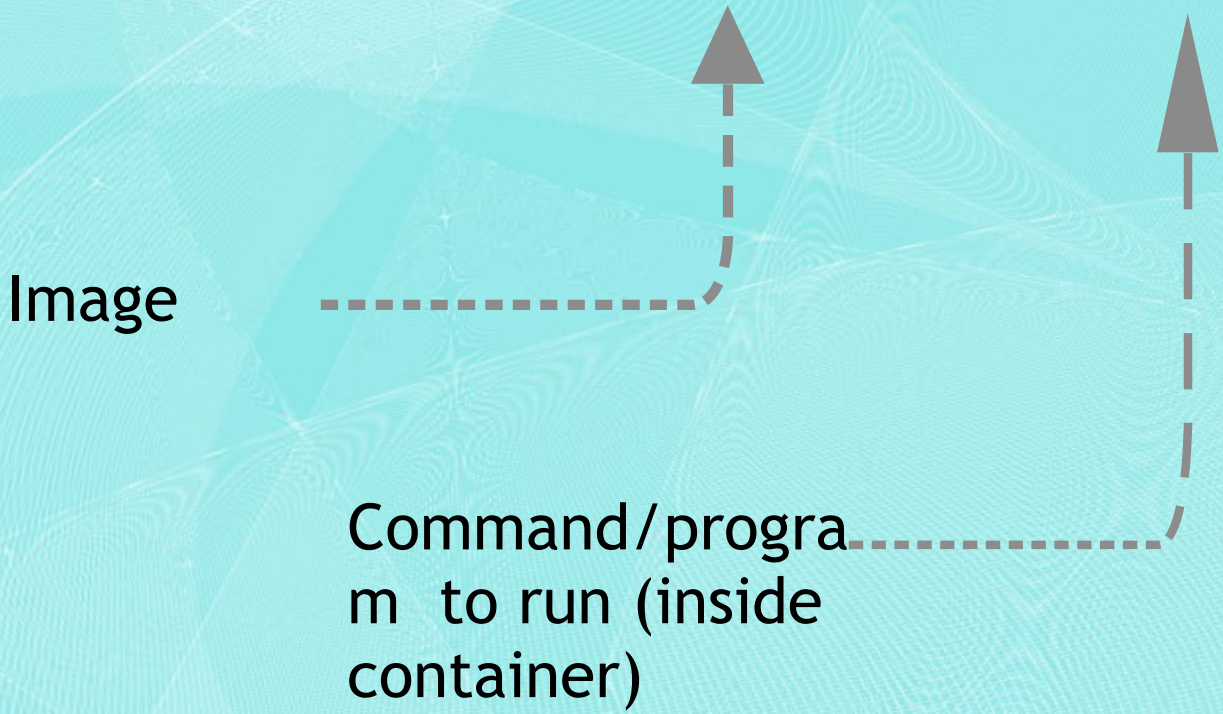
Component Versions

```
$ docker version
```

```
bash-3.2$ docker version
Client version: 1.3.2
Client API version: 1.15
Go version (client): go1.3.3
Git commit (client): 39fa2fa
OS/Arch (client): darwin/amd64
Server version: 1.3.2
Server API version: 1.15
Go version (server): go1.3.3
Git commit (server): 39fa2fa
```

Launching 1st Container

```
$ docker    run    alpine:3.4      uptime
```

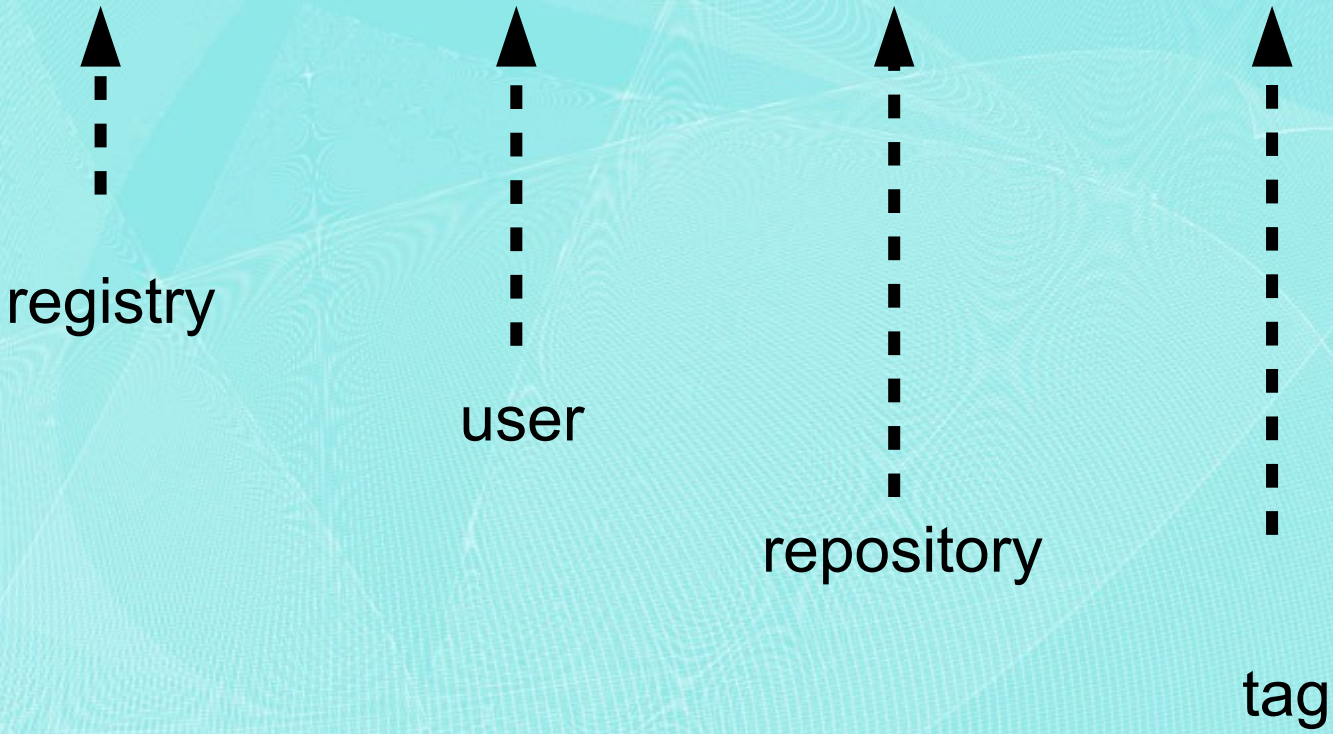


What just happened?

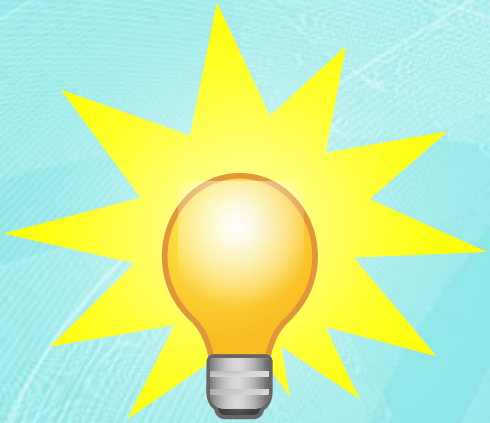
- Docker checks if image “alpine” with tag/version “3.4” is present locally.
- If not, it connects to the docker hub and pulls the image
- Launches container with the image
- Runs the command inside the container.

Image Components

```
reg.123.com/user/ubuntu:12.04.1
```



All that is fine. But where on earth did my container go ? If I run “$ **docker ps**” I don't see a trace of it



Docker **containers** last only till the program you started with it lasts. It will **stop** immediately after the program exits.

Checking Status of last run Container

```
$ docker ps -l
```

```
bash-3.2$ docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
956b80e3716c        ubuntu:14.04        "/bin/echo 'Hello Wo     12 hours ago        Exited (0) 18 minutes ago                       drunk_torvalds
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
956b80e3716c        ubuntu:14.04        "/bin/echo 'Hello Wo     12 hours ago
```

```
STATUS                      PORTS               NAMES
Exited (0) 18 minutes ago                       drunk_torvalds
```

Launch with Interactive Shell

```
$ docker run -i -t alpine:3.4 sh
```


```
root@ee75c3b486db:/# cat /etc/issue
Ubuntu 14.04.1 LTS \n \l
```

Commands

Namespaced

```
$ cat /etc/issue
$ ps aux
$ ifconfig
$ hostname
```

Non Namespaced

```
$ uname -a
$ cat /proc/cpuinfo
$ date
$ free
```

Listing Containers

```
$ docker ps
$ docker ps -l
$ docker ps -n 2
$ docker ps -a
```

Its not fun to have containers which run some ad hoc command and exit. Lets make it persist a little longer.

Running Container which Persists

```
$ docker run -idt schoolofdevops/loop program
```

- d, --detach : detach mode , runs a container and detaches from it Displays container id

Check Status

```
$ docker ps
```

```
bash-3.2$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
fa97bb0bb19e        ubuntu:14.04        "/bin/sh -c 'while t     12 hours ago        Up 4 minutes                            cranky_feynman
```

Check Logs

Syntax: docker logs <container>

e.g.

```
$ docker logs <container_id>
$ docker logs -f <container_id>
```

What is this cranky_feynman ?

Ans: If you do not assign a name to the conainer, it sets some random but entertaining string to it. Thats what it is

When you run commands, replace cranky_feynman with the name on your system

What if I want to connect to the container and run some ad hoc shell commands?

Running one off commands

```
$ docker exec <container_id> ps aux
```

Connect to running container

```
$ docker exec -it <container_id> sh
```

Pausing and Unpausing

```
$ docker pause <container_id>
$ docker unpause <container_id>
```

Create + Start

```
$ docker create --name d01 alpine:3.4 sh
$ docker start d01
$ docker ps

$ docker create -it  --name d02 alpine:3.4 sh
$ docker start d02
$ docker ps
```

Creating Custom Reports

```
$docker ps --format "{{.ID}}: {{.Status}}"
```

Stop Container

```
$ docker stop <container_id>
$ docker ps
```

Finding Previously Run Containers

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
fa97bb0bb19e	ubuntu:14.04	"/bin/sh -c 'while t	12 hours ago	Exited (-1) About a minute ago		cranky_feynman
ee75c3b486db	ubuntu:14.04	"/bin/bash"	12 hours ago	Exited (127) 15 minutes ago		compassionate_engelbar
t						
956b80e3716c	ubuntu:14.04	"/bin/echo 'Hello Wo	13 hours ago	Exited (0) About an hour ago		drunk_torvalds
81f206c4c142	ubuntu:14.04	"/bin/echo 'Hello Wo	13 hours ago	Exited (0) About an hour ago		condescending_blackwel
l						
073f68f26d25	ubuntu:14.04	"/bin/bash"	24 hours ago	Exited (0) About an hour ago		boring_poincare
1b15ac6ae185	ubuntu:14.04	"-it /bin/bash"	24 hours ago			naughty_shockley

Starting Previously Stopped Container

```
$ docker start <container_id>
```

Removing Container

```
$ docker stop <container_id>
```

Its important to stop a container before removing it.

```
$ docker rm <container_id>
```

Summary

- Using Docker Client
- Running Simple Containers
- Pause, Unpause, Start, Stop, Remove Operations