Docker Foundation

An in depth introduction to Docker and Containers

Docker in Practice

1. Volumes
2. Security
3. Networking
4. Orchestration

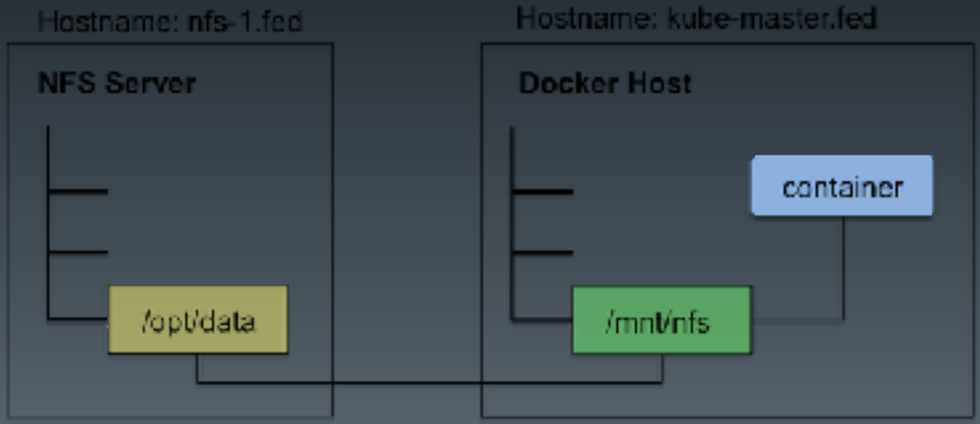
1: Volumes

Objectives

- Define the term “data volume”
- List three ways to create volumes with Docker
- Gain experience with the `docker volume` command
- Learn how to use `--volumes-from`
- Understand the Docker volume architecture

What is a docker volume

- Volumes:
  - Hold data
  - Can be shared/reused across containers
  - Store data in the host file system not the container layer
  - Support 3rd party volume plugins for advanced storage facilities
- A volume mounts a host directory into a container directory
  - Any part of the host file system can be mapped into a container as a volume
  - Any part of a container's file system can be configured as a volume
- Volumes decouple the life of the data from the life of the container(s)
  - You can `docker container rm some_container` without affecting the volumes it uses
  - Running containers can be ephemeral and data can be persistent
  - Docker never automatically deletes volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container
- Volumes bypass the layered file system
  - This makes volumes as fast as the host based filesystem
  - Changes to a volume will not be included when you update an image by committing the container



Volumes

- A volume can be created in three ways:

  -
    ```
    $ docker volume create ...
    ```
    # With the docker volume cmd

  -
    ```
    $ docker container run -v /host/path:/some/cont/dir ...
    ```
    # With the docker run cmd

  -
    ```
    VOLUME /some/dir
    ```
    # Within a Dockerfile

Mounting Host Paths

- A host path volume can be automatically mounted within a docker container using the docker run -v switch
- Volumes default to read/write
  - To mount a volume read only use the `:ro` suffix
    - `$ sudo docker container run -d -P --name web -v /src/wapp:/opt/wapp:ro cisco/wapp python app.py`
- The -v flag can also be used to mount a single file
  - `$ sudo docker container run --rm -it -v ~/.bash_history:/.bash_history ubuntu /bin/bash`

```
docker@ubuntu:~$ mkdir data
docker@ubuntu:~$ cd data
docker@ubuntu:~/data$ vim README.md
docker@ubuntu:~/data$ cd ..
docker@ubuntu:~$ docker container run –i -t -v ~/data:/webdata --name="web01" ubuntu:14.04 /
bin/bash
root@2e5cf445a1b5:/# cat /webdata/README.md
shared Docker data

root@2e5cf445a1b5:/# exit
exit
docker@ubuntu:~$
```

Volume Architecture

- Volumes map container paths to host paths
  - When no host path is supplied Docker creates one (this feature is deprecated)
    - But still works in 1.12.x+
  - MountPoints are found in metadata
- Volumes have a UUID

```
user@ubuntu:~$ docker container run -it -v /some/dir ubuntu
root@0f28f53ab19b:/# echo "hello volumes" > /some/dir/test.md
root@0f28f53ab19b:/# ls -l /some/dir
total 4
-rw-r--r-- 1 root root 14 Nov  4 22:34 test.md
```

```
user@ubuntu:~$ docker container run -it --volumes-from=0f28f53ab19b
ubuntu
root@731342f95ab8:/# cat /some/dir/test.md
hello volumes
```

```
$ docker container inspect -f '{{.Mounts}}' 0f28f53ab19b
[{52b4bd60ebd13131ba36411619aec608929d54bc5b96bd62b69821fabc2b9d52 /var/lib/docker/volumes/
52b4bd60ebd13131ba36411619aec608929d54bc5b96bd62b69821fabc2b9d52/_data /some/dir local  true}]
```

```
$ sudo ls -l /var/lib/docker/volumes/52b4bd60ebd13131ba36411619aec608929d54bc5b96bd62b69821fabc2b9d52/_data
-rw-r--r-- 1 root root 14 Nov  4 14:34 test.md
```

```
user@ubuntu:~$ sudo grep -r 52b4bd60ebd13131ba36411619aec608929d54bc5b96bd62b69821fabc2b9d52 /var/lib/docker
/var/lib/docker/containers/731342f95ab86c33db539ce455a5a01149e4db5991e688077c7e8c6d16ac725e/config.json: {...,
     "MountPoints": {
          "/some/dir": {
               "Name":"52b4bd60ebd13131ba36411619aec608929d54bc5b96bd62b69821fabc2b9d52",
               "Destination":"/some/dir",
               "Driver":"local",
               "RW":true,
               "Source":"",
               "Relabel":""
          }
     },
     "Volumes": {
          "/some/dir":"/var/lib/docker/volumes/52b4bd60ebd13131ba36411619aec608929d54bc5b96bd62b69821fabc2b9d52/
_data"
     },
     "VolumesRW":{
          "/some/dir":true
     }
}
/var/lib/docker/containers/0f28f53ab19b81741f8c854a09a983454025922a395d786d67c9769635afa311/config.json:{...
```

docker volume

```
user@ubuntu:~$ docker volume ls
DRIVER              VOLUME NAME
user@ubuntu:~$ docker container run -v /var/test centos:6
user@ubuntu:~$ docker volume ls
DRIVER              VOLUME NAME
local               b594c8ef1dd7686d4071e6dadd613635ea942bc06a17ab31057cc0bc8879d853
user@ubuntu:~$ docker volume inspect b594c8ef1dd7686d4071e6dadd613635ea942bc06a17ab31057cc0bc8879d853
[
    {
        "Name": "b594c8ef1dd7686d4071e6dadd613635ea942bc06a17ab31057cc0bc8879d853",
        "Driver": "local",
        "Mountpoint": "/var/lib/docker/volumes/b594c8ef1dd7686d4071e6dadd613635ea942bc06a17ab31057cc0bc8879d853/
_data"
    }
]
user@ubuntu:~$ docker volume rm b594c8ef1dd7686d4071e6dadd613635ea942bc06a17ab31057cc0bc8879d853
Error response from daemon: Conflict: volume is in use
user@ubuntu:~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
e27f2f7a1096        centos:6            "/bin/bash"         3 minutes ago       Exited (0) 3 minutes ago
user@ubuntu:~$ docker container rm e27f2f7a1096
e27f2f7a1096
user@ubuntu:~$ docker volume rm b594c8ef1dd7686d4071e6dadd613635ea942bc06a17ab31057cc0bc8879d853
b594c8ef1dd7686d4071e6dadd613635ea942bc06a17ab31057cc0bc8879d853
user@ubuntu:~$ docker volume ls
DRIVER              VOLUME NAME
user@ubuntu:~$ docker volume create --name logdata
logdata
user@ubuntu:~$ docker container run -v logdata:/var/log/data -it centos:6
[root@05c961908425 /]# echo "shared volume" > /var/log/data/output
[root@05c961908425 /]# exit
exit
user@ubuntu:~$ docker container run -v logdata:/var/log/data -it ubuntu
root@6ed0aeaa4c3e:/# cat /var/log/data/output
shared volume
root@6ed0aeaa4c3e:/#
```

The `docker volume` command was introduced in v1.9

- `docker volume create`
- `docker volume inspect`
- `docker volume ls`
- `docker volume rm`
- `docker volume help`

Volumes From

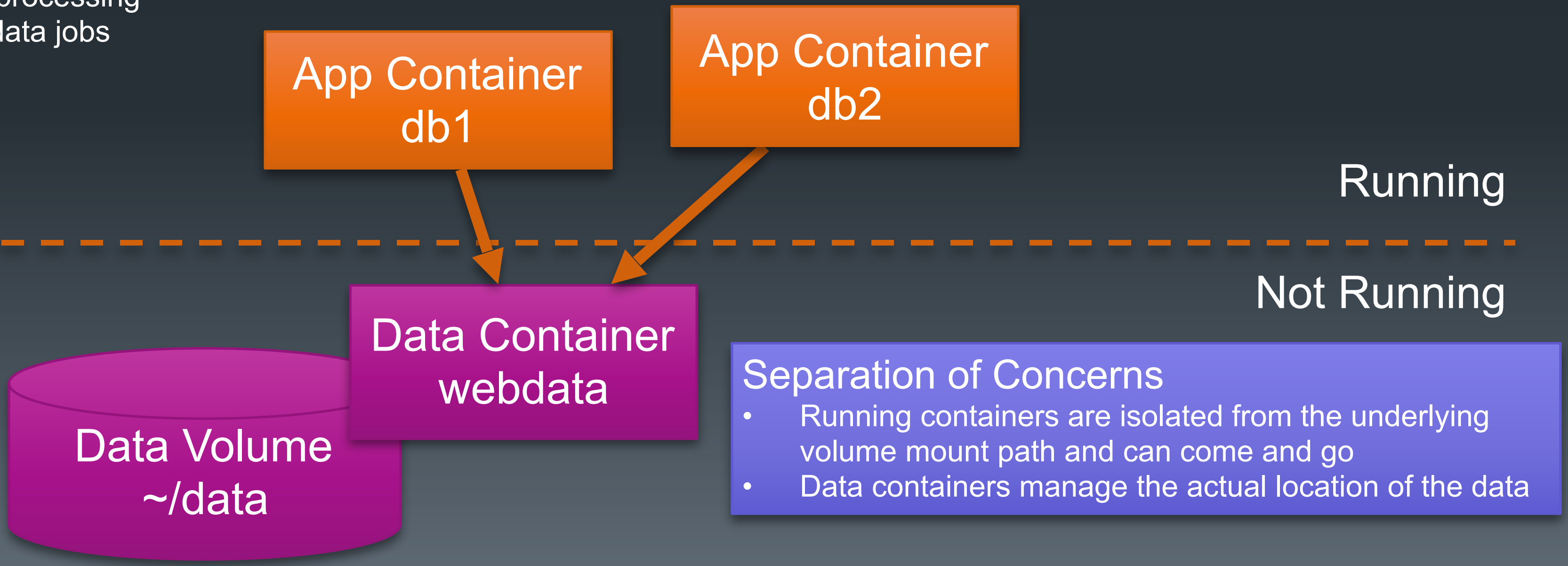
- Containers can expose Docker managed host paths to other containers
  - This allows containers to depend directly on other containers, not the underlying volume
- If you have some persistent data that you want to share between containers, or want to use from non-persistent containers, it's best to:
  - [Docker <= 1.8] Create a named Data Volume Container, and then mount the volumes from that container into other containers
  - [Docker >= 1.9] Create a named volume, and then mount the named volume in other containers
- Data volume containers do not typically run applications
  - `$ docker container create -v /dbdata --name dbdata bobs/postgres`
- `--volumes-from`
  - Used to mount a volume from one container in another container
  - `$ docker container run -d --volumes-from dbdata --name db1 bobs/postgres`
  - `$ docker container run -d --volumes-from dbdata --name db2 bobs/postgres`
  - You can use multiple `--volumes-from` parameters to bring together multiple data volumes from multiple containers
- You can also mount a volume from a container, which in turn was mounted from another container
  - `$ docker container run -d --name db3 --volumes-from db1 bobs/postgres`

Removing Volumes

- If you remove containers that mount volumes the volumes will not be deleted
  - To delete the volume from disk, you must explicitly call `docker container rm -v` against the last container with a reference to the volume
- Alternatively the `docker volume rm` command can be used to remove volumes

Container Volumes (pre docker 1.9)

- Data Containers are a design pattern wherein data volumes are owned by data containers
  - Post Docker v1.9 Data Containers are no longer necessary, volumes can now be managed independently of containers using the docker volume command
  - Older versions of Docker still widely use (and benefit from) the data container pattern
- Data containers run no process
  - `$ docker create -v ~/data:/webdata --name="webdata“ ubuntu:14.04`
  - `$ docker run -d --volumes-from webdata --name db1 bobs/postgres`
- Data Containers exist to own volumes
- Related containers then use `--volumes-from` to mount volumes held by the data container
- A range of operational containers can perform independent tasks in this way
  - User facing applications
  - Backups
  - Log processing
  - Big data jobs
  - Etc.



Separation of Concerns

- Running containers are isolated from the underlying volume mount path and can come and go
- Data containers manage the actual location of the data

Summary

- Volumes are storage mounted from outside of a container bypassing the container union filesystem and with an independent lifespan
- Volumes can be created with
  - `docker volume create`
  - `docker container run -v`
  - The Dockerfile `VOLUME` instruction
- The `docker volume` command allows you to create, list, remove and inspect volumes
- The `--volumes-from` switch allows one container to mount the volumes from another

Lab 1

- Working with Volumes

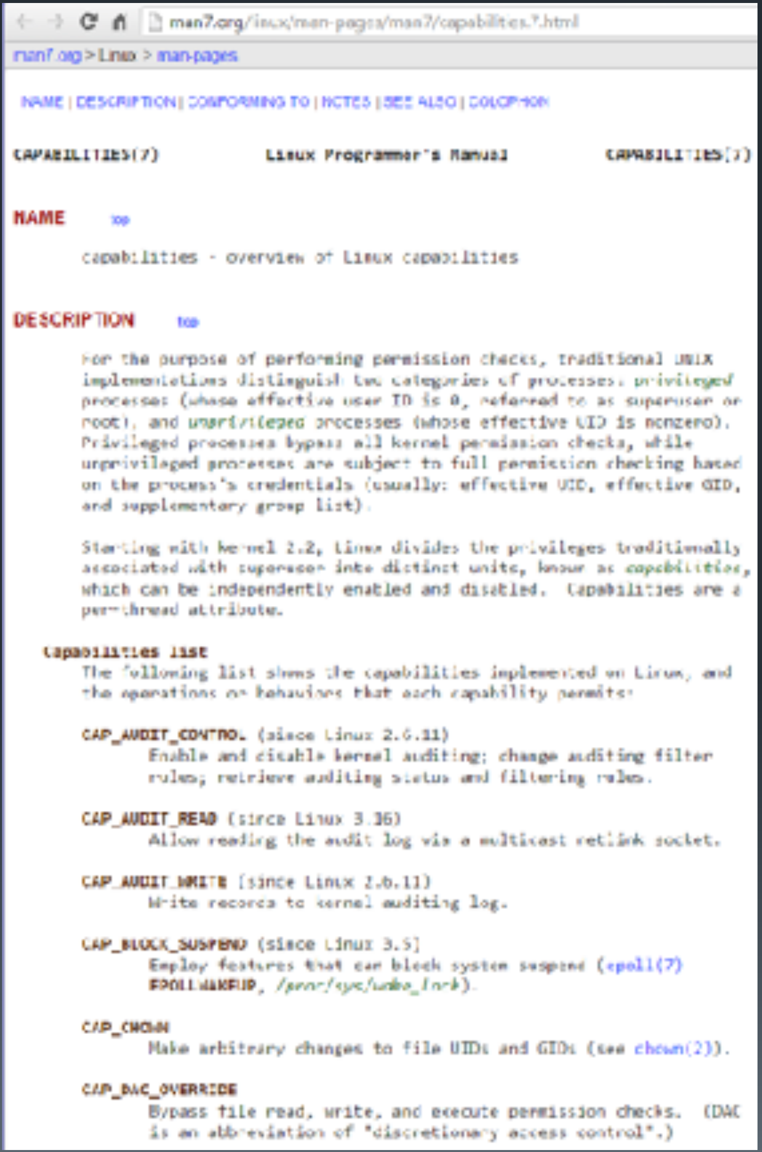
2: Security

Objectives

- Explain the nature and use of Linux Capabilities with Docker
- Describe the benefits and dangers of `--privileged`
- List the container device options
- Explore the Docker `--security-opt` systems supported
- Understand user namespaces
- Discuss security best practices

Capabilities

- The docker container run command offers several security related switches
  - `--privileged`
    - Grants all capabilities inside a container (no whitelist)
    - Not recommended for production (container processes run as if directly on the host)
  - `--cap-add` & `--cap-drop` (v1.2)
    - Linux Capabilities turn the binary "root/non-root" dichotomy into a fine-grained access control system
    - Containers can be given complete capabilities or they can follow a whitelist of allowed capabilities
    - `--cap-add/drop` give you fine grain control over Linux Capabilities granted to a particular container
    - Examples:
      - `docker container run --cap-add=NET_ADMIN \ ubuntu sh -c "ip link eth0 down"`
      - `docker container run --cap-drop=CHOWN ...`
      - `docker container run --cap-add=ALL \ --cap-drop=MKNOD ...`
    - Linux Capabilities documentation: http://man7.org/linux/man-pages/man7/capabilities.7.html
      - There are currently 38 separate capabilities



man7.org/linux/man-pages/man7/capabilities.7.html

man7.org > Linux > man-pages

NAME | DESCRIPTION | CONFORMING TO | NOTES | SEE ALSO | COLOPHON

```
CAPABILITIES(7)          Linux Programmer's Manual          CAPABILITIES(7)
```

NAME top

capabilities - overview of Linux capabilities

DESCRIPTION top

For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes: *privileged* processes (whose effective user ID is 0, referred to as superuser or root), and *unprivileged* processes (whose effective UID is nonzero). Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking based on the process's credentials (usually: effective UID, effective GID, and supplementary group list).

Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as *capabilities*, which can be independently enabled and disabled. Capabilities are a per-thread attribute.

Capabilities list

The following list shows the capabilities implemented on Linux, and the operations or behaviors that each capability permits:

CAP_AUDIT_CONTROL (since Linux 2.6.11)
Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules.

CAP_AUDIT_READ (since Linux 3.16)
Allow reading the audit log via a multicast netlink socket.

CAP_AUDIT_WRITE (since Linux 2.6.11)
Write records to kernel auditing log.

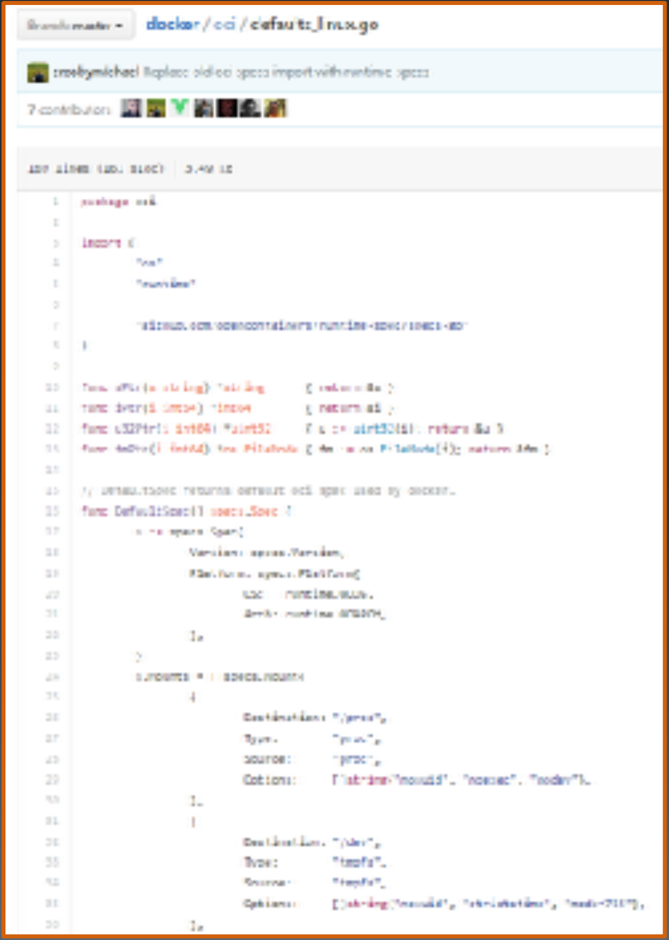
CAP_BLOCK_SUSPEND (since Linux 3.5)
Employ features that can block system suspend (epoll(7) **EPOLLWAKEUP**, */proc/sys/wake_lock*).

CAP_CHOWN
Make arbitrary changes to file UIDs and GIDs (see chown(2)).

CAP_DAC_OVERRIDE
Bypass file read, write, and execute permission checks. (DAC is an abbreviation of "discretionary access control".)

Default Capabilities

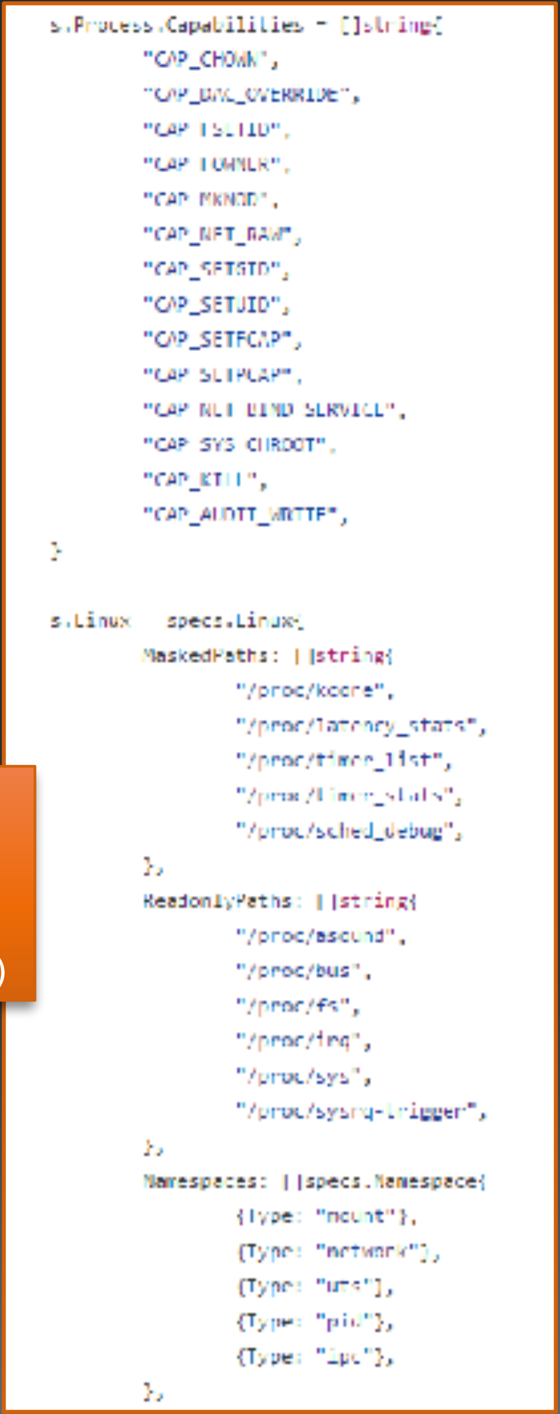
- Default Capabilities
  - https://github.com/moby/moby/blob/master/oci/defaults.go



Branch: master
docker / oci / defaults_linux.go
crosbymichael Replace old oci specs import with runtime-specs
7 contributors
139 lines (125 sloc) 3.49 KB

```go
package oci

import (
        "os"
        "runtime"

        "github.com/opencontainers/runtime-spec/specs-go"
)

func sPtr(s string) *string      { return &s }
func iPtr(i int64) *int64        { return &i }
func u32Ptr(i int64) *uint32     { u := uint32(i); return &u }
func fmPtr(i int64) *os.FileMode { fm := os.FileMode(i); return &fm }

// DefaultSpec returns default oci spec used by docker.
func DefaultSpec() specs.Spec {
        s := specs.Spec{
                Version: specs.Version,
                Platform: specs.Platform{
                        OS:   runtime.GOOS,
                        Arch: runtime.GOARCH,
                },
        }
        s.Mounts = []specs.Mount{
                {
                        Destination: "/proc",
                        Type:        "proc",
                        Source:      "proc",
                        Options:     []string{"nosuid", "noexec", "nodev"},
                },
                {
                        Destination: "/dev",
                        Type:        "tmpfs",
                        Source:      "tmpfs",
                        Options:     []string{"nosuid", "strictatime", "mode=755"},
                },
```

Docker Linux Capability defaults (White List)



```go
s.Process.Capabilities = []string{
        "CAP_CHOWN",
        "CAP_DAC_OVERRIDE",
        "CAP_FSETID",
        "CAP_FOWNER",
        "CAP_MKNOD",
        "CAP_NET_RAW",
        "CAP_SETGID",
        "CAP_SETUID",
        "CAP_SETFCAP",
        "CAP_SETPCAP",
        "CAP_NET_BIND_SERVICE",
        "CAP_SYS_CHROOT",
        "CAP_KILL",
        "CAP_AUDIT_WRITE",
}

s.Linux = &specs.Linux{
        MaskedPaths: []string{
                "/proc/kcore",
                "/proc/latency_stats",
                "/proc/timer_list",
                "/proc/timer_stats",
                "/proc/sched_debug",
        },
        ReadonlyPaths: []string{
                "/proc/asound",
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
                "/proc/sysrq-trigger",
        },
        Namespaces: []specs.Namespace{
                {Type: "mount"},
                {Type: "network"},
                {Type: "uts"},
                {Type: "pid"},
                {Type: "ipc"},
        },
```

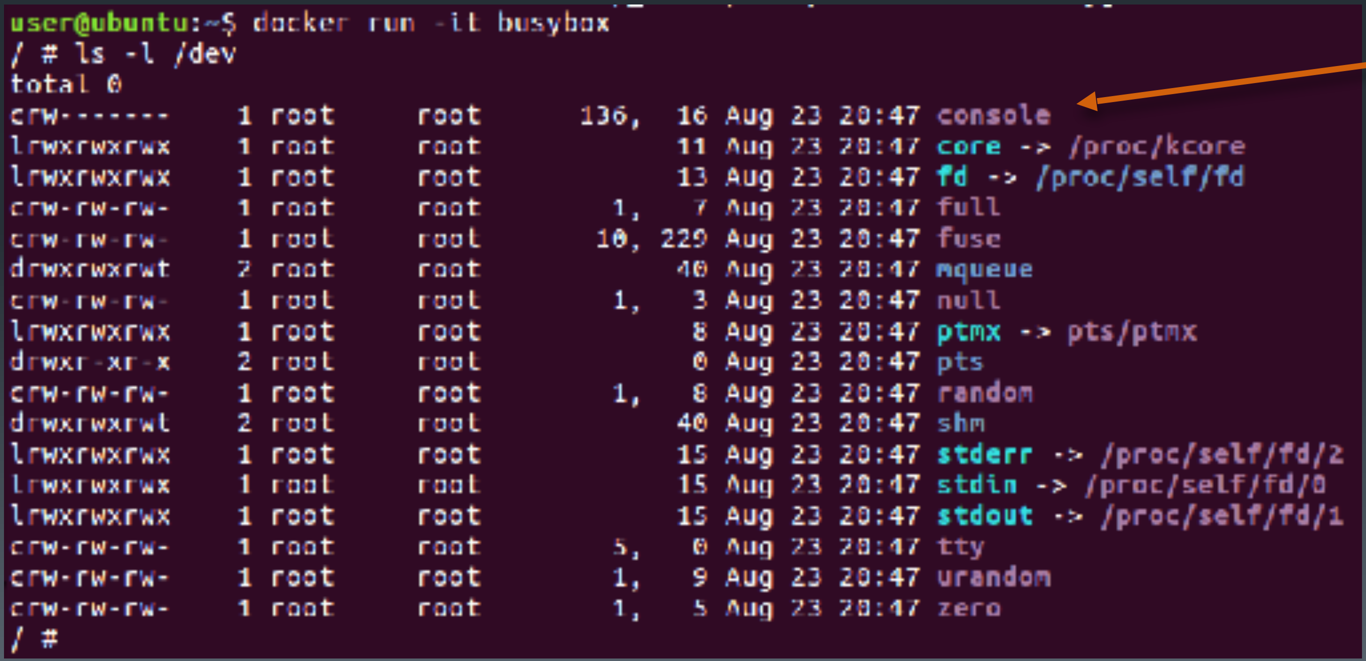
Displaying Capabilities

- *capsh* can be used to display and change capabilities

```
user@ubuntu:~$ capsh --print
Current: =
…

root@ubuntu:/# capsh --print
Current: = cap_chown, cap_dac_override, cap_fowner, cap_fsetid, cap_kill,
cap_setgid, cap_setuid, cap_setpcap, cap_net_bind_service, cap_net_admin,
cap_net_raw, cap_sys_chroot, cap_mknod, cap_audit_write, cap_setfcap+eip
…


user@ubuntu:~$ sudo capsh --print
[sudo] password for user:
Current: = cap_chown, cap_dac_override, cap_dac_read_search, cap_fowner,
cap_fsetid, cap_kill, cap_setgid, cap_setuid, cap_setpcap, cap_linux_immutable,
cap_net_bind_service, cap_net_broadcast, cap_net_admin, cap_net_raw, cap_ipc_lock,
cap_ipc_owner, cap_sys_module, cap_sys_rawio, cap_sys_chroot, cap_sys_ptrace,
cap_sys_pacct, cap_sys_admin, cap_sys_boot, cap_sys_nice, cap_sys_resource,
cap_sys_time, cap_sys_tty_config, cap_mknod,cap_lease, cap_audit_write,
cap_audit_control, cap_setfcap, cap_mac_override, cap_mac_admin, cap_syslog,
cap_wake_alarm, cap_block_suspend, 37+ep
```

Devices

- Containers have a minimal set of devices by default
- Other needed devices from the host can be mapped into a container
  - You can use devices in privileged containers by bind mounting them (with `` `-v` ``)
  - A better option may be `--device` (v1.2)
    - The `--device` flag lets you use a device without `--privileged`
    - Examples:
      - `docker container run --device=/dev/snd:/dev/snd ...`



```
user@ubuntu:~$ docker run -it busybox
/ # ls -l /dev
total 0
crw-------    1 root     root      136,  16 Aug 23 20:47 console
lrwxrwxrwx    1 root     root            11 Aug 23 20:47 core -> /proc/kcore
lrwxrwxrwx    1 root     root            13 Aug 23 20:47 fd -> /proc/self/fd
crw-rw-rw-    1 root     root        1,   7 Aug 23 20:47 full
crw-rw-rw-    1 root     root       10, 229 Aug 23 20:47 fuse
drwxrwxrwt    2 root     root            40 Aug 23 20:47 mqueue
crw-rw-rw-    1 root     root        1,   3 Aug 23 20:47 null
lrwxrwxrwx    1 root     root             8 Aug 23 20:47 ptmx -> pts/ptmx
drwxr-xr-x    2 root     root             0 Aug 23 20:47 pts
crw-rw-rw-    1 root     root        1,   8 Aug 23 20:47 random
drwxrwxrwt    2 root     root            40 Aug 23 20:47 shm
lrwxrwxrwx    1 root     root            15 Aug 23 20:47 stderr -> /proc/self/fd/2
lrwxrwxrwx    1 root     root            15 Aug 23 20:47 stdin -> /proc/self/fd/0
lrwxrwxrwx    1 root     root            15 Aug 23 20:47 stdout -> /proc/self/fd/1
crw-rw-rw-    1 root     root        5,   0 Aug 23 20:47 tty
crw-rw-rw-    1 root     root        1,   9 Aug 23 20:47 urandom
crw-rw-rw-    1 root     root        1,   5 Aug 23 20:47 zero
/ #
```

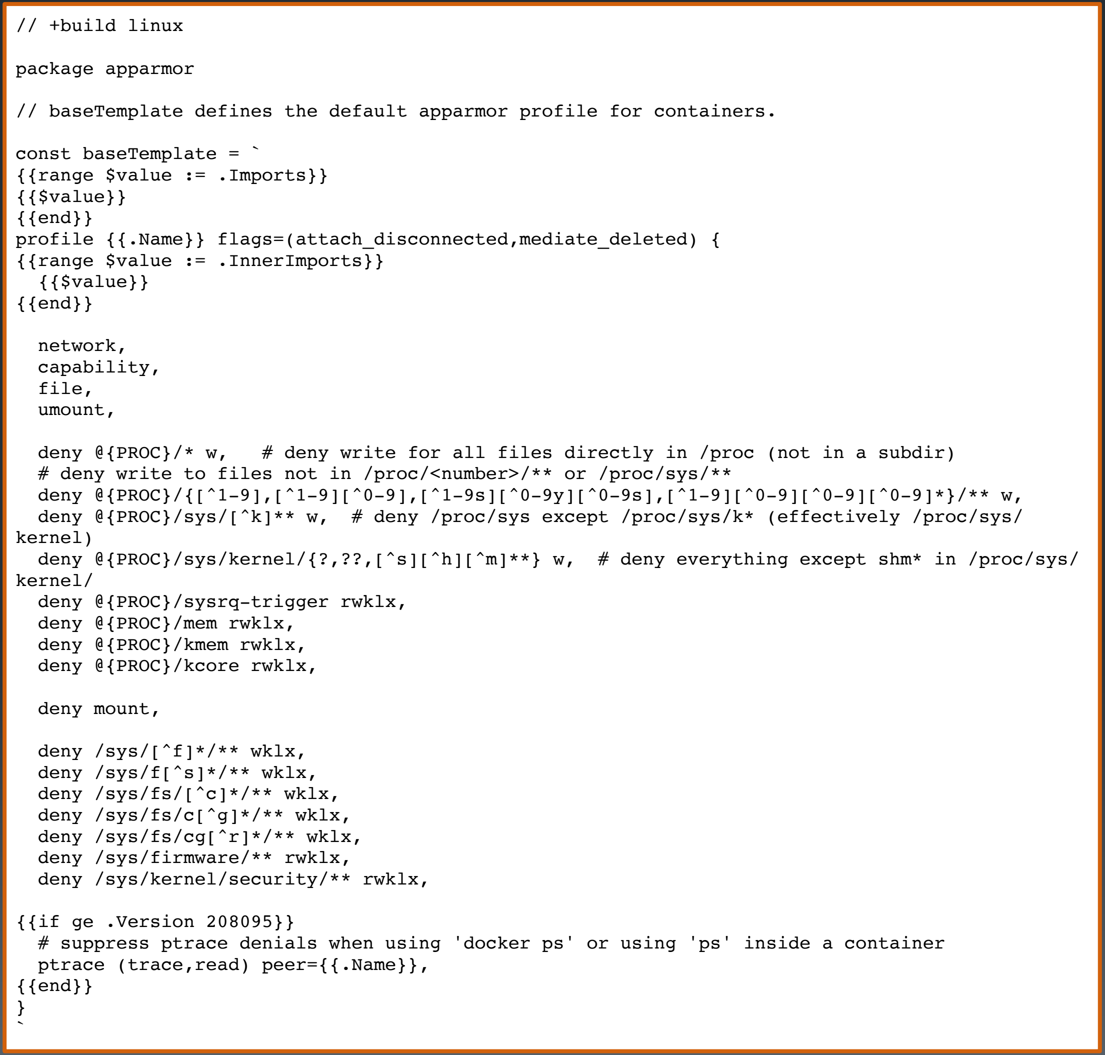
The console device is created by the run `-t` switch

Security Options

- `--security-opt` (v1.3)
  - Sets custom SELinux labels, AppArmor profiles and SecComp files
  - All confine programs (not users) to a limited set of resources
- SELinux
  - Docker offers two forms of SELinux protection:
    - Type enforcement
    - Multi-category security (MCS) separation
  - A policy ("svirt_apache") allowing a container process to listen only on Apache ports can be applied as follows:

    -
      ```
      docker container run \
      --security-opt label:type:svirt_apache \
      -it centos bash
      ```
- AppArmor
  - The Docker binary installs a docker-default AppArmor profile generated from the template at: https://github.com/moby/moby/blob/master/profiles/apparmor/template.go
    - Versions 1.13 and later, generated in tmpfs and loaded into the kernel
    - Versions earlier than 1.13, profile is generated in: /etc/apparmor.d/docker
  - Moderately protective while providing wide application compatibility
  - When you run a container, it uses the docker-default policy unless you override it with the security-opt option
    - Save a custom profile to disk in /etc/apparmor.d/containers/ (as my_profile for example)

    -
      ```
      $ docker container run \
      --security-opt apparmor=my_profile \
      hello-world
      ```
  - A profile for the Docker Engine daemon exists but it is not currently installed with the deb packages
    - located in contrib/apparmor in the Docker Engine source



```
// +build linux

package apparmor

// baseTemplate defines the default apparmor profile for containers.

const baseTemplate = `
{{range $value := .Imports}}
{{$value}}
{{end}}
profile {{.Name}} flags=(attach_disconnected,mediate_deleted) {
{{range $value := .InnerImports}}
  {{$value}}
{{end}}

  network,
  capability,
  file,
  umount,

  deny @{PROC}/* w,   # deny write for all files directly in /proc (not in a subdir)
  # deny write to files not in /proc/<number>/** or /proc/sys/**
  deny @{PROC}/{[^1-9],[^1-9][^0-9],[^1-9s][^0-9y][^0-9s],[^1-9][^0-9][^0-9][^0-9]*}/** w,
  deny @{PROC}/sys/[^k]** w,  # deny /proc/sys except /proc/sys/k* (effectively /proc/sys/
kernel)
  deny @{PROC}/sys/kernel/{?,??,[^s][^h][^m]**} w,  # deny everything except shm* in /proc/sys/
kernel/
  deny @{PROC}/sysrq-trigger rwklx,
  deny @{PROC}/mem rwklx,
  deny @{PROC}/kmem rwklx,
  deny @{PROC}/kcore rwklx,

  deny mount,

  deny /sys/[^f]*/** wklx,
  deny /sys/f[^s]*/** wklx,
  deny /sys/fs/[^c]*/** wklx,
  deny /sys/fs/c[^g]*/** wklx,
  deny /sys/fs/cg[^r]*/** wklx,
  deny /sys/firmware/** rwklx,
  deny /sys/kernel/security/** rwklx,

{{if ge .Version 208095}}
  # suppress ptrace denials when using 'docker ps' or using 'ps' inside a container
  ptrace (trace,read) peer={{.Name}},
{{end}}
}
`
```

Docker v1.10 Security Improvements

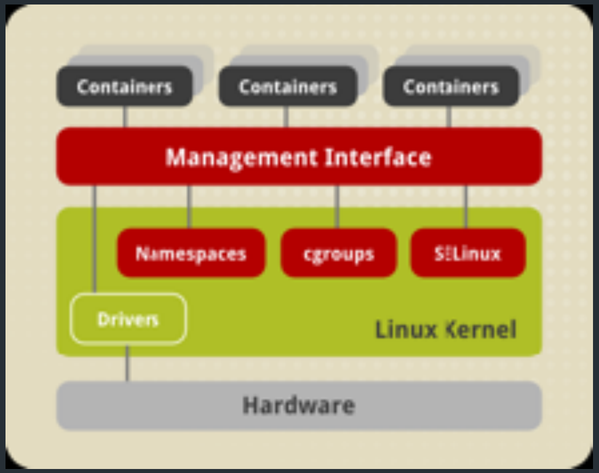
- Many security improvements were added in Docker v1.10
- Seccomp
  - Secure computing mode (Seccomp) is a Linux kernel feature
    - Requires a kernel configured with `CONFIG_SECCOMP`
  - Seccomp was developed by Google to remove system calls from a process (used with Chrome plugins)
    - There are 600 syscalls and a bug in any one could enable privilege escalation
  - libseccomp was created by Red Hat's Paul Moore to simplify the management of the syscall tree, now used in tools like qemu, system, lxc tools and Docker
  - Docker v1.10 `--security-opt` now allows admins to configure seccomp profiles per container

```
docker container run --rm -it --security-opt \
     seccomp:/path/to/seccomp/profile.json hello-world
```

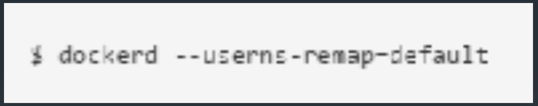
- Seccomp profiles require seccomp 2.2.1
  - Only available with
    - Debian 9 "Stretch" +
    - Ubuntu 15.10 "Wily" +
    - Fedora 22 +
    - RHEL 7 & CentOS 7 +
    - Oracle Linux 7 +
  - Seccomp backports
    - Ubuntu 14.04, Debian Wheezy, or Debian Jessie can download a static Docker Linux binary (not available for other distributions)

Docker v1.10 Security Improvements

- User namespaces
  - A kernel namespace allowing separation between UIDs on the host and the container
  - Added to Linux kernel 2.6 in 2008, enabled in most distros after 2014
  - A range of container UID/GIDs (e.g. 0-1,000) map into the host user namespace as 70,000-71,000
    - e.g. the kernel treats UID 0 (root) inside the container as UID 70,000 outside the container (nobody)
  - Any UID on a file or a process that is not in the mapped range would be treated as UID=-1 and not be accessible in the container
    - Many kernel subsystems are not namespaced and are accessible to containers making this a key security improvement (e.g. SELinux, Kernel Modules and some paths for /sys, /proc, and /dev)
  - Now the best user to use inside a container is ROOT!



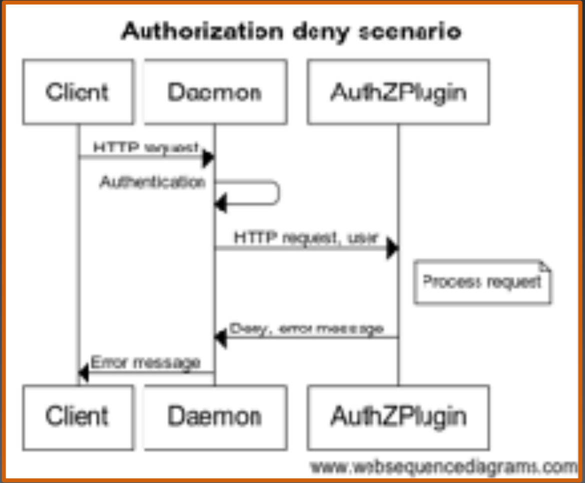
- Options for `--userns-remap`
  - default – maps container users (including root) to a range of high number host users
  - uid – maps root to the specified user
  - uid:gid – maps root to the specified user and group
  - username – maps root to the specified user
  - username:groupname – maps root to the specified user and group



```
$ dockerd --userns-remap=default
```

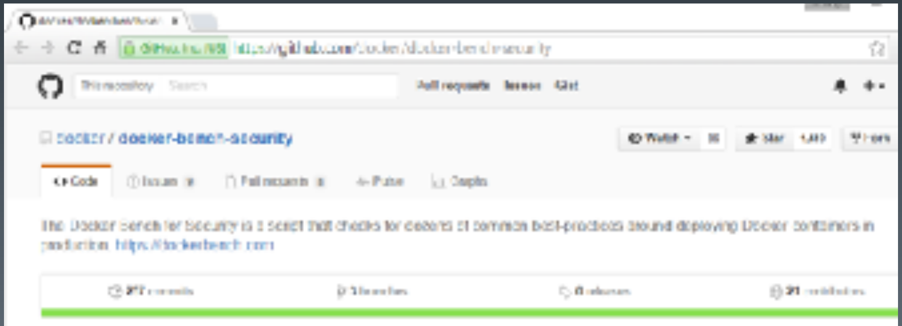
Docker v1.10 Security Improvements

- Auth
  - Prior to Docker v1.10, if you can talk to Docker's socket you can do anything
  - The new Docker daemon `--authorization-plugin` flag allows admins to add RBAC (role based access control) authorization features
  - Auth plugins act as interceptors that can allow/deny any docker API request based on rules created by admins
    - Plugins receive the current authentication context and the command context
  - You can install multiple plugins and chain them together in a defined order
    - All plugins must GRANT access for access to succeed
  - Shortfall
    - Docker does not currently supply any authorization plugins (3rd parties?)
    - Docker only offers certificate-based authentication presently



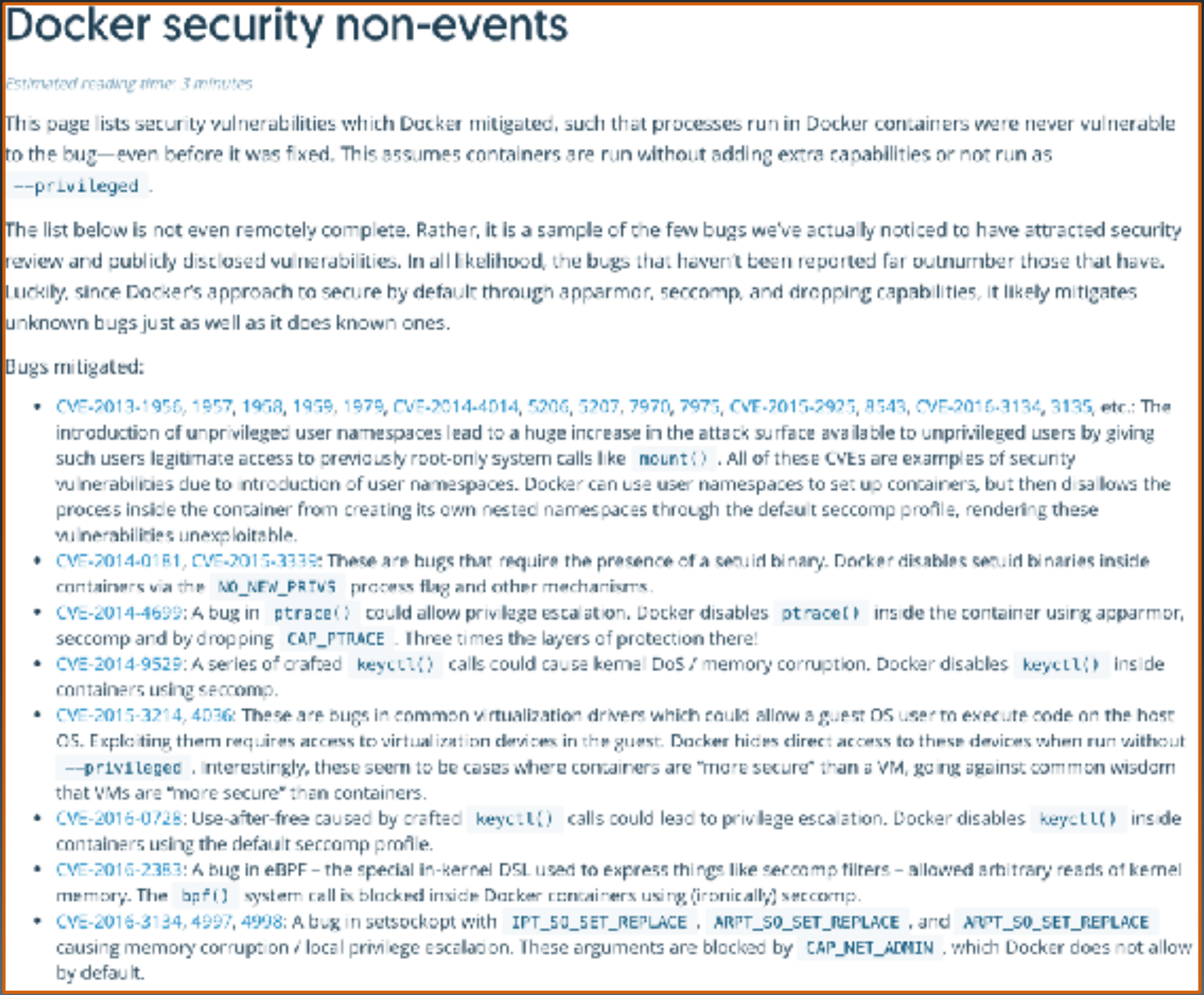
Security Best Practices

- Run Docker Engine with AppArmor, SELinux, or Seccomp to provide better process containment (`--security-opt`)
  - Capabilities is the default security system used by Docker engine
  - AppArmor profile generator for docker containers: https://github.com/jessfraz/bane
- Map groups of mutually-trusted containers to separate machines
- Do not run untrusted applications with root privileges (`--userns-remap`)
- Follow the "Principle of least privilege"
  - In a particular abstraction layer of a computing environment, every module (such as a process, a user or a container) must be able to access only the information and resources that are necessary for its legitimate purpose (`--cap-add/drop`, `seccomp`, or `--device`; never `--privileged`)
- Resources
  - Center for Internet Security Benchmark for Docker 1.1 (7-6-2017):
    - https://www.cisecurity.org/benchmark/docker/
  - Docker Inc's Introduction to Container Security:
    - https://www.docker.com/sites/default/files/WP_IntrotoContainerSecurity_08.19.2016.pdf
  - Docker Security Center
    - http://www.docker.com/docker-security
  - Docker Bench for Security
    - https://github.com/docker/docker-bench-security



The Docker Bench for Security is a script that checks for dozens of common best-practices around deploying Docker containers in production. https://dockerbench.com

Container Value Add



Docker security non-events

Estimated reading time: 3 minutes

This page lists security vulnerabilities which Docker mitigated, such that processes run in Docker containers were never vulnerable to the bug—even before it was fixed. This assumes containers are run without adding extra capabilities or not run as `--privileged` .

The list below is not even remotely complete. Rather, it is a sample of the few bugs we've actually noticed to have attracted security review and publicly disclosed vulnerabilities. In all likelihood, the bugs that haven't been reported far outnumber those that have. Luckily, since Docker's approach to secure by default through apparmor, seccomp, and dropping capabilities, it likely mitigates unknown bugs just as well as it does known ones.

Bugs mitigated:

- CVE-2013-1956, 1957, 1958, 1959, 1979, CVE-2014-4014, 5206, 5207, 7970, 7975, CVE-2015-2925, 8543, CVE-2016-3134, 3135, etc.: The introduction of unprivileged user namespaces lead to a huge increase in the attack surface available to unprivileged users by giving such users legitimate access to previously root-only system calls like `mount()` . All of these CVEs are examples of security vulnerabilities due to introduction of user namespaces. Docker can use user namespaces to set up containers, but then disallows the process inside the container from creating its own nested namespaces through the default seccomp profile, rendering these vulnerabilities unexploitable.
- CVE-2014-0181, CVE-2015-3339: These are bugs that require the presence of a setuid binary. Docker disables setuid binaries inside containers via the `NO_NEW_PRIVS` process flag and other mechanisms.
- CVE-2014-4699: A bug in `ptrace()` could allow privilege escalation. Docker disables `ptrace()` inside the container using apparmor, seccomp and by dropping `CAP_PTRACE` . Three times the layers of protection there!
- CVE-2014-9529: A series of crafted `keyctl()` calls could cause kernel DoS / memory corruption. Docker disables `keyctl()` inside containers using seccomp.
- CVE-2015-3214, 4036: These are bugs in common virtualization drivers which could allow a guest OS user to execute code on the host OS. Exploiting them requires access to virtualization devices in the guest. Docker hides direct access to these devices when run without `--privileged` . Interestingly, these seem to be cases where containers are "more secure" than a VM, going against common wisdom that VMs are "more secure" than containers.
- CVE-2016-0728: Use-after-free caused by crafted `keyctl()` calls could lead to privilege escalation. Docker disables `keyctl()` inside containers using the default seccomp profile.
- CVE-2016-2383: A bug in eBPF – the special in-kernel DSL used to express things like seccomp filters – allowed arbitrary reads of kernel memory. The `bpf()` system call is blocked inside Docker containers using (ironically) seccomp.
- CVE-2016-3134, 4997, 4998: A bug in setsockopt with `IPT_SO_SET_REPLACE` , `ARPT_SO_SET_REPLACE` , and `ARPT_SO_SET_REPLACE` causing memory corruption / local privilege escalation. These arguments are blocked by `CAP_NET_ADMIN` , which Docker does not allow by default.

Summary

- The Linux Kernel API is organized into capabilities which can be added or dropped per container
- The `--privileged` switch extends all capabilities, making it particularly troublesome in security conscious settings
- Devices can be mapped to containers as needed
- Docker `--security-opts` allow container processes to be constrained by AppArmor, SELinux and/or SecComp
  - Default is Capabilities
- User namespaces allow container users to be mapped to unprivileged users on the host

Lab 2

- Docker Security

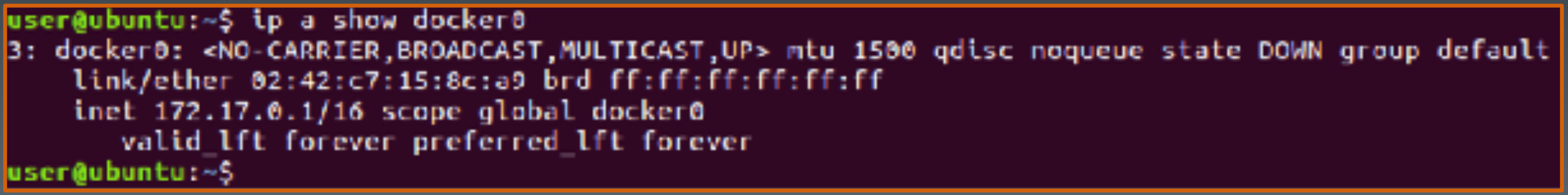
3: Networking

Objectives

- Describe the features of Docker networking
- Use docker and host based commands (ex. iptables) to work with containers and networks
- Describe the Docker container linking feature
- Understand the basics of SDN

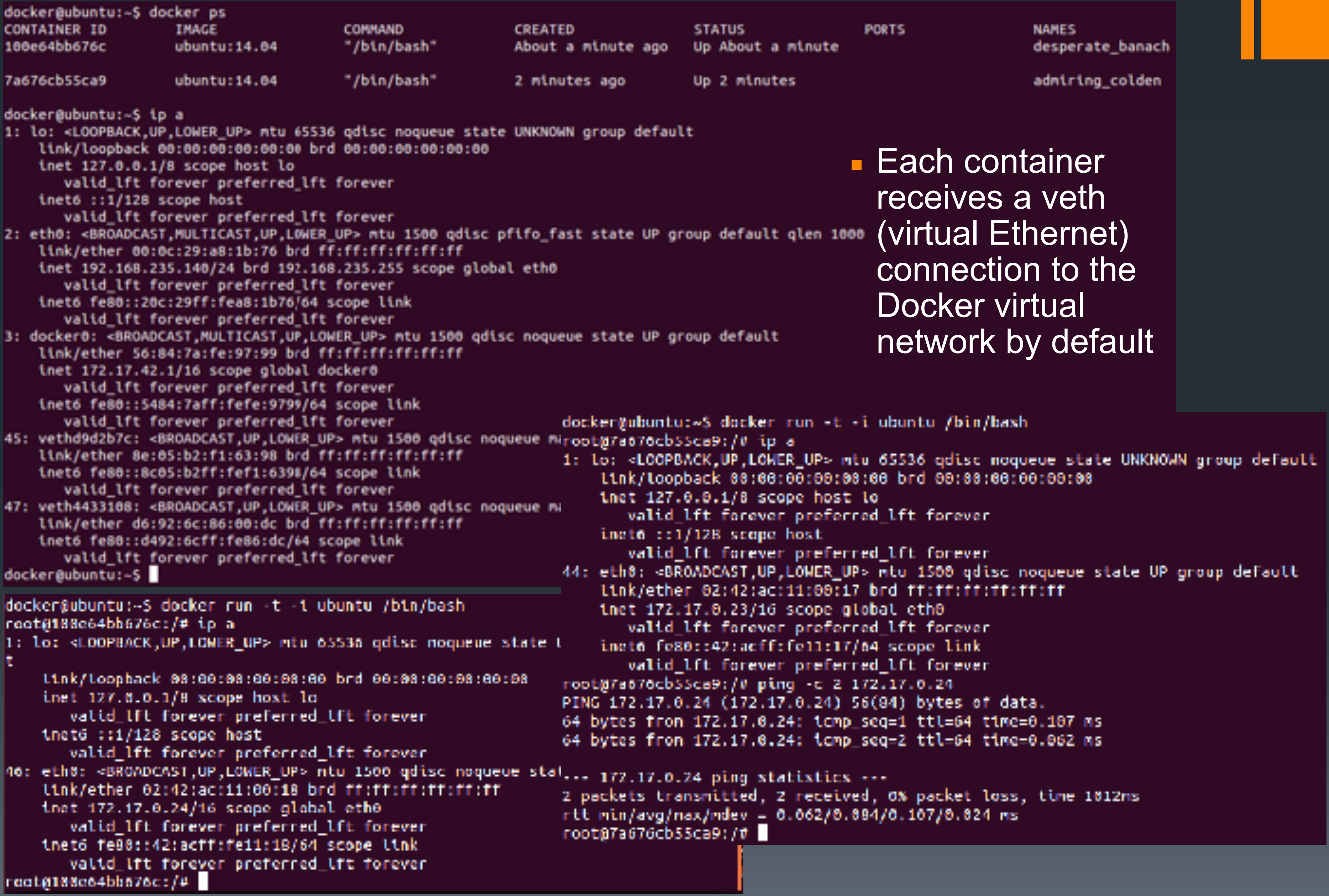
Container Networking

- Docker containers expose ports which can be mapped to host interfaces so that container services are exposed on the host's external network
- Docker also supports internal networking
  - Docker assigns containers an IP address on an isolated host based virtual network (172.17.x.x, though docker will choose an alternate subnet if there is a conflict)
  - Docker defines a host interface as a gateway
    - Docker0, usually 172.17.0.1
- Containers may get new IP addresses each restart
- Docker 1.5+ also supports IPv6 addresses
  - To enable ipv6 run the Docker daemon with the `--ipv6` flag
- Docker 1.9+ introduced key new container networking features
  - Docker can create multiple named networks (previously one net per host, docker0)
  - Networks can span hosts (previously each docker0 network existed on only a single host)
  - Containers can have multiple network interfaces (previously each container had only one network interface)
  - Docker networking is a large topic and covered in detail in the Advanced Docker course, more info:
    - `$ docker network --help`
    - https://docs.docker.com/engine/userguide/networking/



```
user@ubuntu:~$ ip a show docker0
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:c7:15:8c:a9 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
user@ubuntu:~$
```

Container interfaces



```
docker@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED              STATUS              PORTS               NAMES
180e64bb676c        ubuntu:14.04        "/bin/bash"         About a minute ago   Up About a minute                       desperate_banach

7a676cb55ca9        ubuntu:14.04        "/bin/bash"         2 minutes ago        Up 2 minutes                            admiring_colden

docker@ubuntu:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:a8:1b:76 brd ff:ff:ff:ff:ff:ff
    inet 192.168.235.140/24 brd 192.168.235.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fea8:1b76/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 56:84:7a:fe:97:99 brd ff:ff:ff:ff:ff:ff
    inet 172.17.42.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::5484:7aff:fefe:9799/64 scope link
       valid_lft forever preferred_lft forever
45: vethd9d2b7c: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue ma
    link/ether 8e:05:b2:f1:63:98 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::8c05:b2ff:fef1:6398/64 scope link
       valid_lft forever preferred_lft forever
47: veth4433108: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue ma
    link/ether d6:92:6c:86:00:dc brd ff:ff:ff:ff:ff:ff
    inet6 fe80::d492:6cff:fe86:dc/64 scope link
       valid_lft forever preferred_lft forever
docker@ubuntu:~$
```

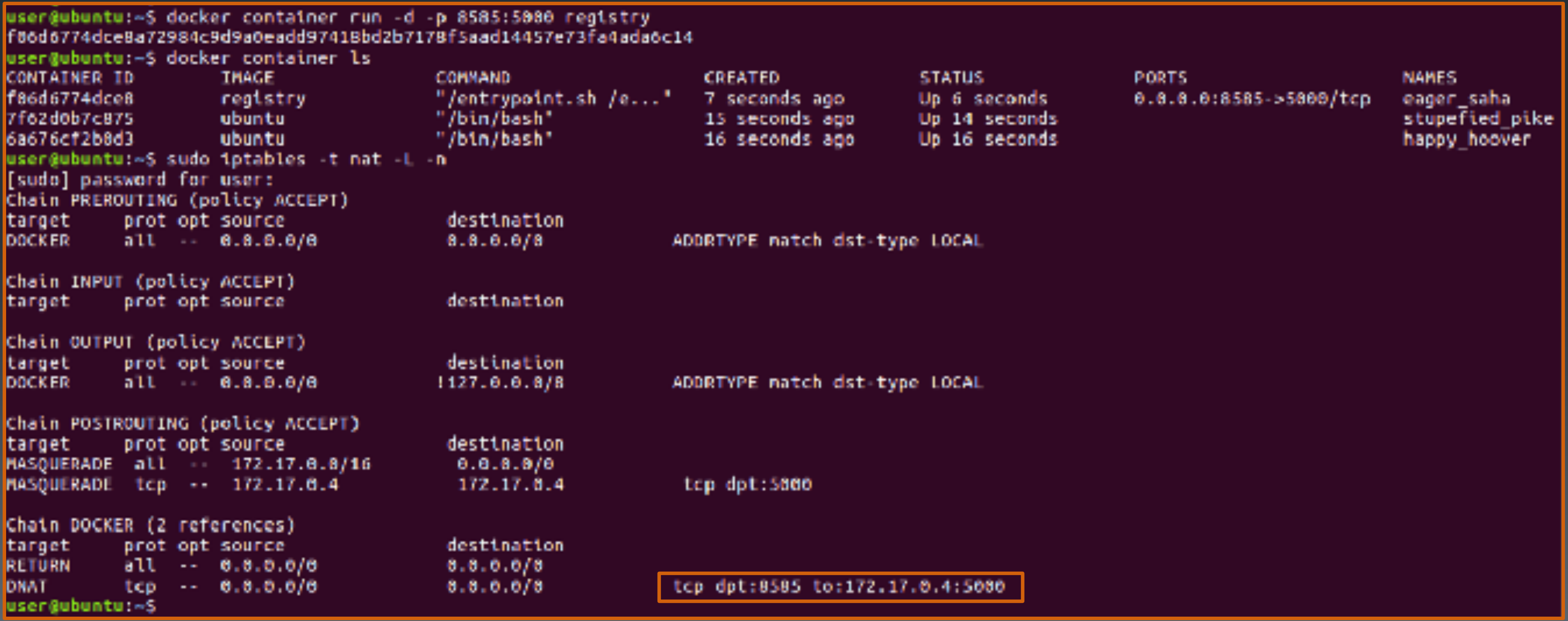
```
docker@ubuntu:~$ docker run -t -i ubuntu /bin/bash
root@180e64bb676c:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state U
t
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
46: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue stat
    link/ether 02:42:ac:11:00:18 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.24/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:18/64 scope link
       valid_lft forever preferred_lft forever
root@180e64bb676c:/#
```

```
docker@ubuntu:~$ docker run -t -i ubuntu /bin/bash
root@7a676cb55ca9:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
44: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:17 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.23/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:17/64 scope link
       valid_lft forever preferred_lft forever
root@7a676cb55ca9:/# ping -c 2 172.17.0.24
PING 172.17.0.24 (172.17.0.24) 56(84) bytes of data.
64 bytes from 172.17.0.24: icmp_seq=1 ttl=64 time=0.107 ms
64 bytes from 172.17.0.24: icmp_seq=2 ttl=64 time=0.062 ms

--- 172.17.0.24 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1012ms
rtt min/avg/max/mdev = 0.062/0.084/0.107/0.024 ms
root@7a676cb55ca9:/#
```

- Each container receives a veth (virtual Ethernet) connection to the Docker virtual network by default

The Docker Host Interface

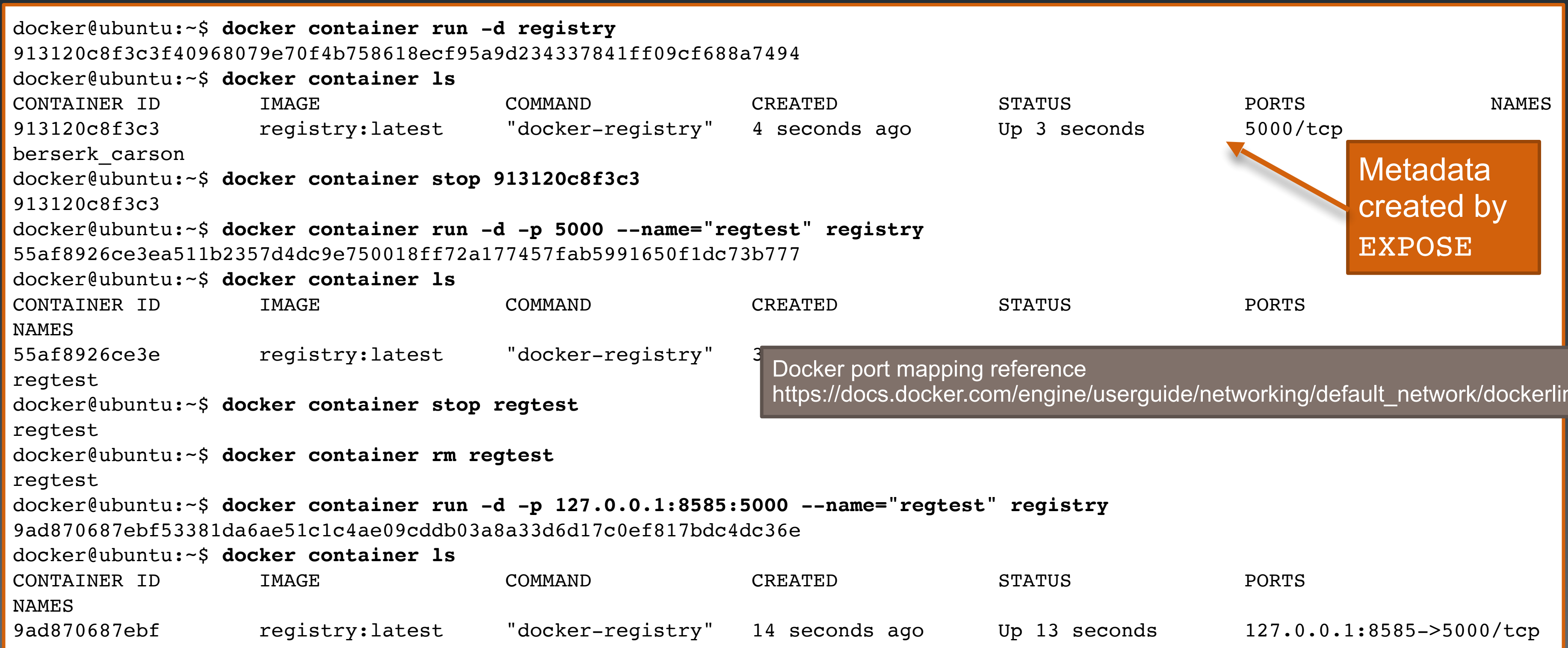
- Docker containers access the outside world through the Host interface
- The 172 network is not externally routable and must be translated using NAT/PAT
  - Firewall rules and NAT configuration allow Docker to route between containers and the host network
  - DNAT (destination based network address translation) is used to map host ports to containers



```
user@ubuntu:~$ docker container run -d -p 8585:5000 registry
f86d6774dce8a72984c9d9a0eadd97418bd2b7178f5aad14457e73fa4ada6c14
user@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
f86d6774dce8        registry            "/entrypoint.sh /e..."   7 seconds ago       Up 6 seconds        0.0.0.0:8585->5000/tcp   eager_saha
7f62d0b7c875        ubuntu              "/bin/bash"              15 seconds ago      Up 14 seconds                                stupefied_pike
6a676cf2b8d3        ubuntu              "/bin/bash"              16 seconds ago      Up 16 seconds                                happy_hoover
user@ubuntu:~$ sudo iptables -t nat -L -n
[sudo] password for user:
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  0.0.0.0/0            0.0.0.0/0            ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  172.17.0.0/16        0.0.0.0/0
MASQUERADE  tcp  --  172.17.0.4           172.17.0.4           tcp dpt:5000

Chain DOCKER (2 references)
target     prot opt source               destination
RETURN     all  --  0.0.0.0/0            0.0.0.0/0
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:8585 to:172.17.0.4:5000
user@ubuntu:~$
```

Port Assignment

- The `-p` flag manages container network port forwarding
  - Containers may be configured in various ways
    - The container ports may be internal to the container (unavailable from the host)
    - Docker can randomly assign a host port (49000 – 49900) to container ports
    - You can specify a host port for each container port
    - Docker can map the same port used by the container on the host interface using `-P` (capital P w/ no params)
    - The host port can be prefixed with an interface
    - UDP ports can be exposed by adding the `/udp` suffix
- The `port` command displays port details (similar to the `container ls` command)
- For example the Docker registry image specifies the `EXPOSE 5000` instruction
  - The session below demonstrates three possible port mapping styles:



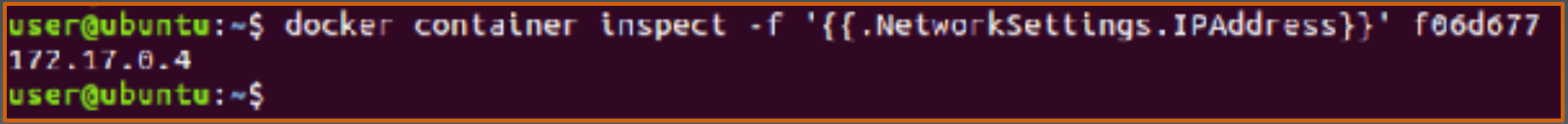
```
docker@ubuntu:~$ docker container run -d registry
913120c8f3c3f40968079e70f4b758618ecf95a9d234337841ff09cf688a7494
docker@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND              CREATED             STATUS              PORTS               NAMES
913120c8f3c3        registry:latest     "docker-registry"    4 seconds ago       Up 3 seconds        5000/tcp
berserk_carson
docker@ubuntu:~$ docker container stop 913120c8f3c3
913120c8f3c3
docker@ubuntu:~$ docker container run -d -p 5000 --name="regtest" registry
55af8926ce3ea511b2357d4dc9e750018ff72a177457fab5991650f1dc73b777
docker@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND              CREATED             STATUS              PORTS
NAMES
55af8926ce3e        registry:latest     "docker-registry"    3
regtest
docker@ubuntu:~$ docker container stop regtest
regtest
docker@ubuntu:~$ docker container rm regtest
regtest
docker@ubuntu:~$ docker container run -d -p 127.0.0.1:8585:5000 --name="regtest" registry
9ad870687ebf53381da6ae51c1c4ae09cddb03a8a33d6d17c0ef817bdc4dc36e
docker@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND              CREATED             STATUS              PORTS
NAMES
9ad870687ebf        registry:latest     "docker-registry"    14 seconds ago      Up 13 seconds       127.0.0.1:8585->5000/tcp
```

Metadata created by `EXPOSE`

Docker port mapping reference
https://docs.docker.com/engine/userguide/networking/default_network/dockerlinks/

Linking Containers

- Inter-Container Communications
  - Docker containers can use external services via MASCARADE rules
  - Docker containers can expose services externally using DNAT rules
  - Docker containers can also be linked directly using Docker virtual networks
- Container naming
  - Named containers on the same virtual network can discover each other by name (via /etc/hosts as of Docker v1.9 and via DNS as of Docker 1.10)
- The docker run `--link` flag is used to create container aliases
  - For example:
    - `$ docker container run -d --name dbsvr train/postgres`
    - `$ docker container run -d -P --name web` **`--link dbsvr:db`** `train/webapp python app.py`
      - “`-P`” (uppercase P) switch publishes the container EXPOSE port(s) directly on the host with random ephemeral ports
  - The `--link` switch takes the form: `--link <name or id>:alias`
    - “name” is the container to link to and “alias” is a resolvable name for the target container within the client container
    - Prior to Docker 1.10 the `--link` switch was only supported on the Docker0 Host network (in Docker 1.10+ link aliases can be created between any two containers on the same network)
- Inter-container connections do not require externally exposed ports
  - Note the db container was started without `-P` or `-p`



```
user@ubuntu:~$ docker container inspect -f '{{.NetworkSettings.IPAddress}}' f06d677
172.17.0.4
user@ubuntu:~$
```

Link /etc/hosts and ENV updates

- Prior to Docker v1.10
  - Host files
    - When two containers are linked Docker adds a host entry for the source container to the /etc/hosts file

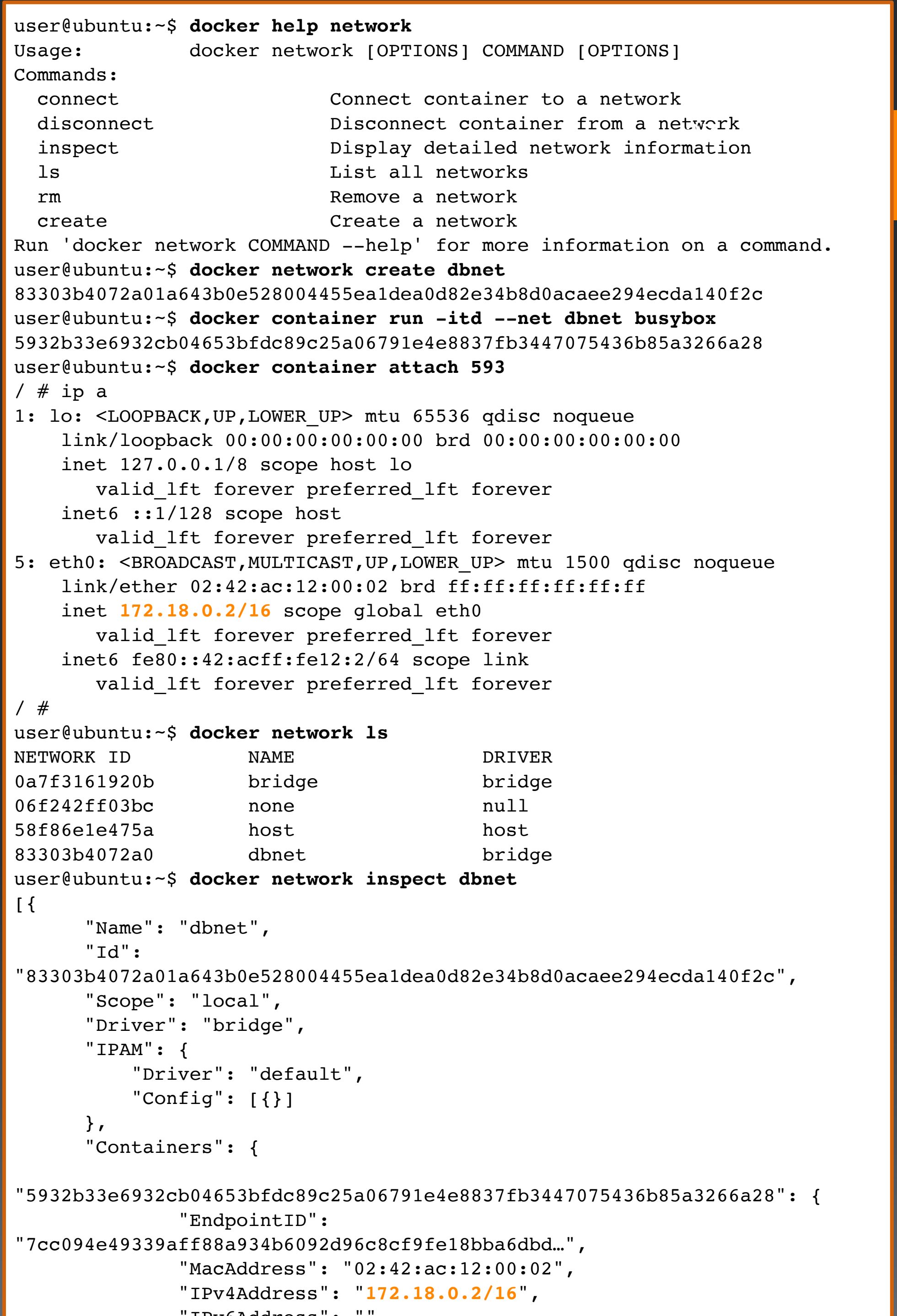
      ```
      $ docker container run -it --rm --link dbsvr:db training/webapp /bin/bash
      root@aed84ee21bde:/opt/webapp# cat /etc/hosts
      172.17.0.7   aed84ee21bde      #the web app itself
      . . .
      172.17.0.5   db                #the linked db container
      ```
  - Environment Variables
    - When two containers are linked Docker adds environment variables in the client container also

      ```
      $ docker container run -it --rm --link dbsvr:db training/webapp /bin/bash
      root@aed84ee21bde:/opt/webapp# env
          . . .
          DB_NAME=/web2/db
          DB_PORT=tcp://172.17.0.5:5432
          DB_PORT_5432_TCP=tcp://172.17.0.5:5432
          DB_PORT_5432_TCP_PROTO=tcp
          DB_PORT_5432_TCP_PORT=5432
          DB_PORT_5432_TCP_ADDR=172.17.0.5
      ```

- Docker v1.10+
  - Docker v1.10 updates /etc/hosts for containers on the Docker0 network for backwards compatibility
  - Containers on user-defined networks now use a Docker daemon supplied DNS server to lookup other containers
  - Links simply add additional aliases for containers

docker network [added in Docker v1.9]

- `docker network`
  - `ls` – list the networks available
  - `create` – creates a new virtual network
    - --internal
  - `rm` – removes an existing virtual network
  - `inspect` – displays network information
  - `connect` – connects an existing container to the specified network
  - `disconnect` – disconnects an existing container from the specified network
- Docker 1.9+ allows containers to connect to multiple networks
  - eth0, eth1, etc.
- Network drivers (`-d <driver>`)
  - `none` – loopback only
  - `host` – no container network namespace
  - `bridge` – Linux bridge (only available on the host it is created on)
  - `overlay` – allows networks to span multiple hosts using VXLAN, requires a key/value store (etcd, Consul or ZooKeeper)



```
user@ubuntu:~$ docker help network
Usage:        docker network [OPTIONS] COMMAND [OPTIONS]
Commands:
  connect                    Connect container to a network
  disconnect                 Disconnect container from a network
  inspect                    Display detailed network information
  ls                         List all networks
  rm                         Remove a network
  create                     Create a network
Run 'docker network COMMAND --help' for more information on a command.
user@ubuntu:~$ docker network create dbnet
83303b4072a01a643b0e528004455ea1dea0d82e34b8d0acaee294ecda140f2c
user@ubuntu:~$ docker container run -itd --net dbnet busybox
5932b33e6932cb04653bfdc89c25a06791e4e8837fb3447075436b85a3266a28
user@ubuntu:~$ docker container attach 593
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
5: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe12:2/64 scope link
       valid_lft forever preferred_lft forever
/ #
user@ubuntu:~$ docker network ls
NETWORK ID          NAME                DRIVER
0a7f3161920b        bridge              bridge
06f242ff03bc        none                null
58f86e1e475a        host                host
83303b4072a0        dbnet               bridge
user@ubuntu:~$ docker network inspect dbnet
[{
      "Name": "dbnet",
      "Id":
"83303b4072a01a643b0e528004455ea1dea0d82e34b8d0acaee294ecda140f2c",
      "Scope": "local",
      "Driver": "bridge",
      "IPAM": {
          "Driver": "default",
          "Config": [{}]
      },
      "Containers": {

"5932b33e6932cb04653bfdc89c25a06791e4e8837fb3447075436b85a3266a28": {
              "EndpointID":
"7cc094e49339aff88a934b6092d96c8cf9fe18bba6dbd…",
              "MacAddress": "02:42:ac:12:00:02",
              "IPv4Address": "172.18.0.2/16",
              "IPv6Address": "",
```

Summary

- Docker uses the built in features Linux to perform basic networking functions
  - IP Tables
  - Linux Bridge
- The `docker container run -p` switch provides a range of port mapping features
- The `docker container run --link` switch creates a caller based alias for a destination container name
- Docker offers a plug in system (CNM – common network model) supported by libnetwork which can be used to enable multihost networking and a range of SDN network systems

Lab 3

- Working with Networks

4: Orchestration

Objectives

- Discuss the basic features provided by container orchestration systems
- Discover various orchestration platforms
- Describe the Docker ecosystem of orchestration tools
- List the features of Swarm Mode

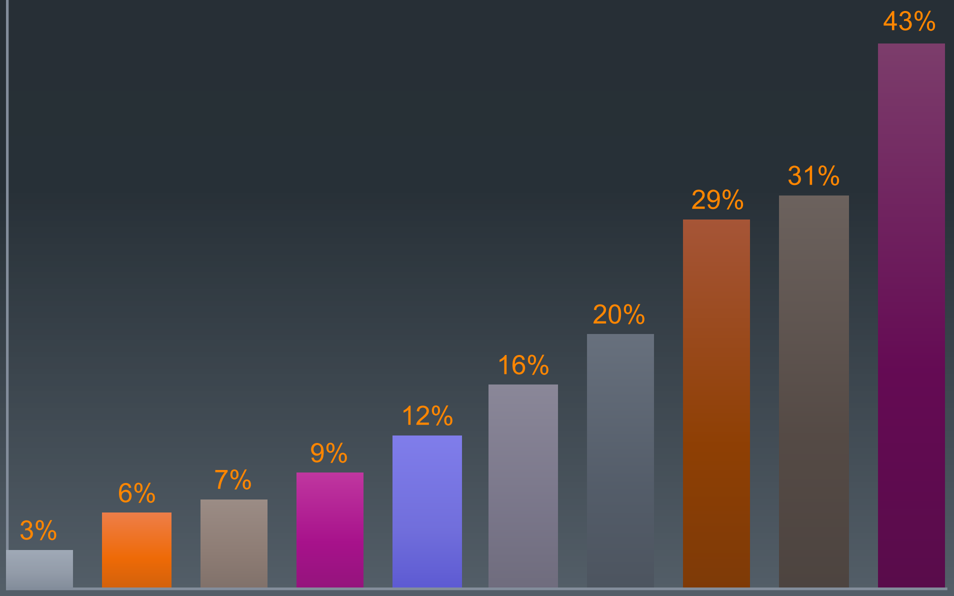
After Containers, what's left?

- Orchestration!
- Real cloud native applications are delivered in 10s of images and require 100s - 1000s of running containers
- Registries manage image distribution
- Orchestration manages:
  - Container Scheduling
    - Distributing containers to appropriate hosts
    - Host resource leveling
    - Availability zone diversity
    - Related container packaging and co-deployment
  - Container Management
    - Monitoring and recovery
    - Image upgrade rollout
    - Scaling
    - Logging
  - Service Endpoints
    - Discovery
    - HA
    - Load balancers
    - Auto scaling
  - External Services
    - Network configuration and management
    - Durable volume management

Container Orchestration

- Distribution and management of containers across a networked cluster of servers
- Orchestration Tools
  - Docker Swarm
    - Docker authored native clustering for Docker containers (integrated with Docker Engine as of Docker 1.12)
    - Also now Docker Data Center, built on Docker Universal Control Plane (Tutum)
  - Kubernetes
    - Google authored open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts
    - EngineYard Deis, RedHat OpenShift PaaS platforms and CoreOS Tectonic are all based on Kubernetes
  - Apache Mesos
    - A distributed system kernel which abstracts CPU, memory, storage, and other compute resources away from machines
    - Mesos 0.20.0 shipped with built in Docker support
  - Cloud Foundry Diego
    - Elastic runtime written in Go supporting the Garden container manager with runC as the backend
    - Well integrated with OpenStack among other platforms
  - Configuration Management Platforms
    - Various CM platforms can be used to deploy/maintain containers
    - Ansible, Chef, Puppet, Capistrano, …
- Other cluster scheduling systems:
  - Mesosphere Marathon for Mesos
  - Apache Aurora for Mesos
  - Facebook Tupperware (fairly proprietary/LXC)
- Cloud Support
  - Google Kubernetes Engine [GKE]
    - Optimized VMs with Kubernetes preinstalled
  - EC2 Container Service [ECS] AWS EC2 Docker containers
  - Elastic Container Service for Kubernetes [EKS] (preview) AWS Kubernetes-as-a-service
  - OpenStack Magnum Container Cluster as a Service
    - Ubuntu support for lxc containers with lxd “lightvisor”
  - Azure Contianer Service [ACS]
    - .Net containers demoed at Build conference 4/2015
    - Windows Server 2016 ships with a native Docker Engine
    - Azure supports Mesos (DC/OS), Docker Swarm and Kubernetes

Which Container orchestration tools does your organization use?

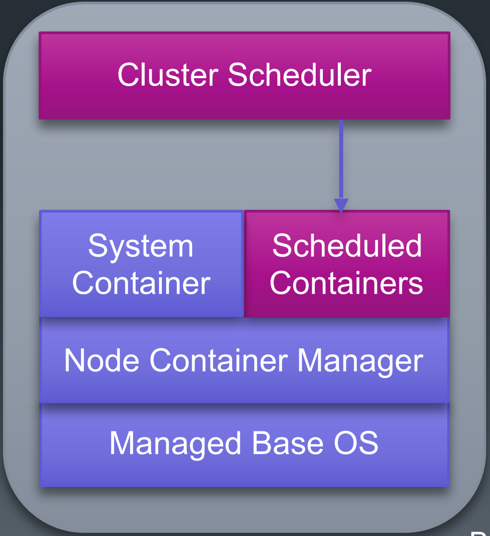


https://portworx.com/2017-container-adoption-survey/

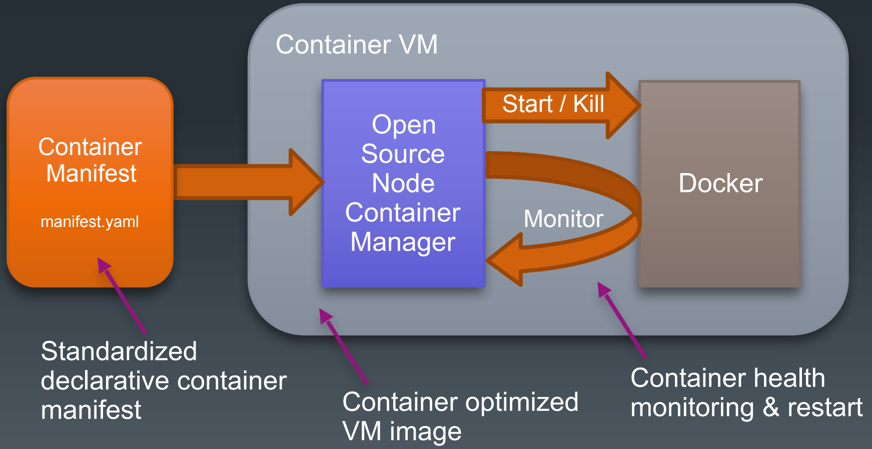
Containers at scale

- Containers - a key enabler of PaaS cloud environments
  - Google App Engine is one of the most visible container based cloud PaaS systems
  - Google's new Google Cloud Platform is a native container as a service based cloud: CaaS
- Everything at Google, from Search to Gmail, is packaged and run in a Linux container
  - Google's Borg system (now replaced by its successor Omega) is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different containerized applications, across a number of clusters each with up to tens of thousands of machines (Borg is the basis for Kubernetes)
  - Over 2 billion containers are started per week at Google (over 3,000 per second on average)
- lmctfy ("Let Me Contain That For You") open source version of Google's container stack
  - Now collaborating with Docker and porting the core lmctfy concepts and abstractions to libcontainer,
  - libcontainer is the go forward OCI solution

Google Container Infrastructure



Google Cloud Platform
Docker containers in Google Compute Engine with Kubernetes

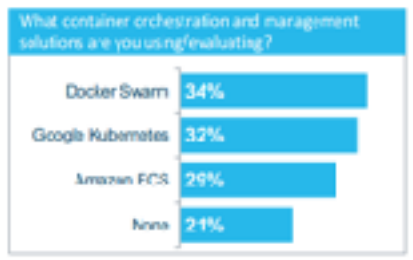

Container VM
Container Manifest
manifest.yaml
Open Source Node Container Manager
Start / Kill
Monitor
Docker
Standardized declarative container manifest
Container optimized VM image
Container health monitoring & restart

Containers At Scale by
Joe Beda
Published May 22, 2014
in Technology

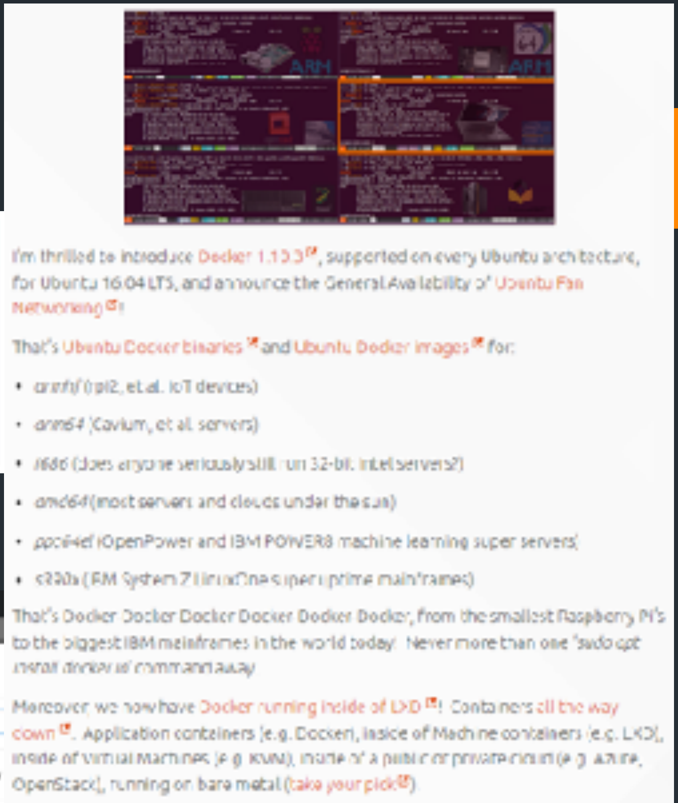
Borg Paper: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.pdf
Omega Paper: http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/41684.pdf

Docker EcoSystem

- Github reports over 100,000+ Docker projects
- Docker, Inc. offers a suite of orchestration services
  - Docker Machine configures Docker on cloud servers
  - Docker Compose multi-container app deployment (uses YAML templates)
  - Docker Swarm container cluster service
  - Docker Trusted Registry on prem Docker Hub
  - Docker Universal Control Plane container deployment
    - Tutum, acquired 2015, renamed Universal Control Plane, offers LDAP/AD integrated container deployment/management
  - Docker Datacenter (UCP and Trusted Reg)
  - Docker Cloud (UCP and Trusted Reg as a Service++)



Docker 3/2016 Survey



I'm thrilled to introduce Docker 1.10.3, supported on every Ubuntu architecture, for Ubuntu 16.04 LTS, and announce the General Availability of Ubuntu Fan Networking!

That's Ubuntu Docker binaries and Ubuntu Docker images for:

- armhf (rpi2, et al. IoT devices)
- arm64 (Cavium, et al. servers)
- i686 (does anyone seriously still run 32-bit Intel servers?)
- amd64 (most servers and clouds under the sun)
- ppc64el (OpenPower and IBM POWER8 machine learning super servers)
- s390x (IBM System Z LinuxOne super uptime mainframes)

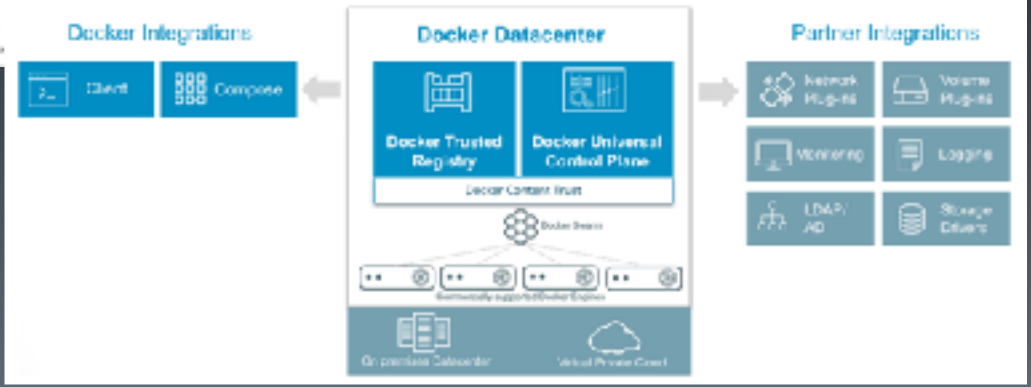
That's Docker Docker Docker Docker Docker Docker, from the smallest Raspberry Pi's to the biggest IBM mainframes in the world today! Never more than one 'sudo apt install docker.io' command away.

Moreover, we now have Docker running inside of LXD! Containers all the way down. Application containers (e.g. Docker), inside of Machine containers (e.g. LXD), inside of Virtual Machines (e.g. KVM), inside of a public or private cloud (e.g. Azure, OpenStack), running on bare metal (take your pick!)

We've found 106,2?? repository results

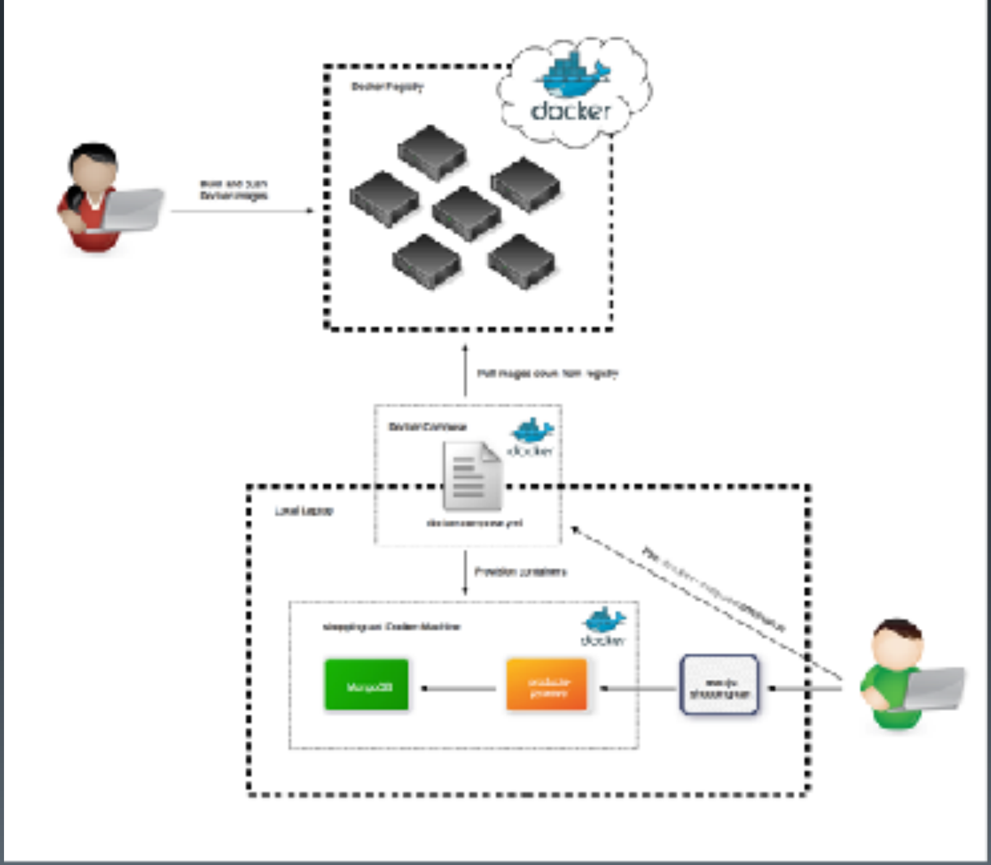
docker/docker
Docker - the open-source application container engine

chef-cookbooks/docker
Chef Cookbook for Docker



Docker Compose Concepts

- Docker Compose is a tool for defining and running multi-container applications with Docker (think Microservices)
  - As of version 1.3, Compose uses Docker labels to keep track of compose application containers
- You define a multi-container application in a single file
- You can then start your application in a single command which does everything that needs to be done to run the app
- Compose is useful in:
  - Development environments
  - Staging servers
  - CI
- Docker does not recommend that you use it in production yet



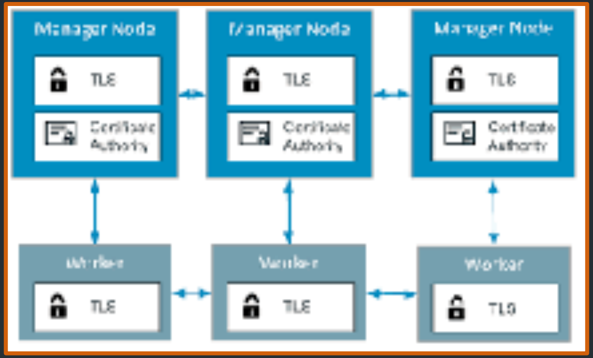
docker-compose.yml

- Each Compose application is defined in a Compose YAML file
- There are four compose file formats
  - v1 – the original and still supported format allowing “services” to be defined
    - Each top level YAML key identifies a named service
  - v2 – the format released with Compose 1.6
    - Top level keys include
      - version
      - services
      - volumes
      - networks
    - v2.1 – minor revision that added some additional functionality
  - v3 – format released with Compose 1.10
    - Between versions 2.x and 3.x, the structure of the Compose file is the same but v3 is designed to be compatible with swarm mode (released w/ Docker 1.12)
    - v3.1 – added the top level key: secrets
    - v3.2 – minor revision that added some additional functionality
- Each service must specify either an image or a build key
  - Images launch containers from an existing image
  - Builds launch containers from a Dockerfile (which is built just prior to execution)
- Other keys are optional, and many are analogous to their `docker run` command line counterparts
- Docker Compose YAML documentation
  - https://docs.docker.com/compose/compose-file/

```
version: '3.2'
services:
 wp:
  image: wordpress
  depends_on:
   - db
  deploy:
   replicas: 2
   update_config:
    parallelism: 1
    delay: 30s
   restart_policy:
    condition: on-failure
  ports:
   - target: 30080
    published: 80
    protocol: tcp
    mode: ingress
  volumes:
   - webroot:/var/www/html
  environment:
   WORDPRESS_DB_HOST: db:3306
   WORDPRESS_DB_NAME: wpdb
   WORDPRESS_DB_PASSWORD: ${WP_PASS}
   WORDPRESS_DB_USER: wpuser
  networks:
   webnet:
    ipv4_address: 172.22.
   dbnet:
    aliases:
     - blogsrv
 db:
  image: mysql
  deploy:
   replicas: 2
   update_config:
    parallelism: 1
    delay: 30s
   restart_policy:
    condition: always
  volumes:
   - type: volume
    source: database
    target: /var/lib/mysq
  env_file: dbenv
  secrets:
   - source: dbsecrets
    target: mysecretfile
    mode: 0600
  networks:
   - dbnet
...
```

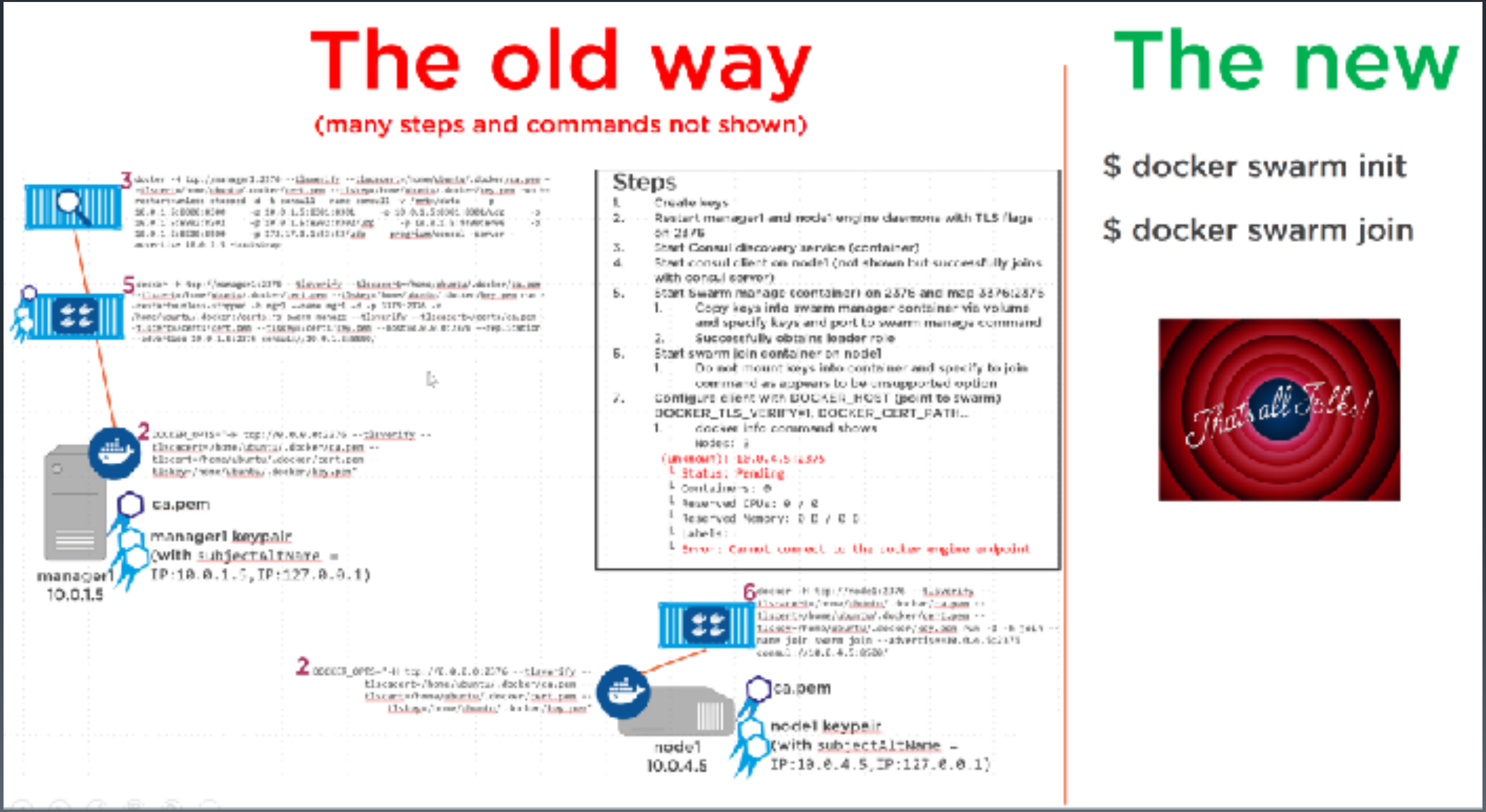
```
...
volumes:
 webroot:
  driver: local
 database:
  driver: netapp
  driver_opts:
   snapshotDir: "false"

secrets:
 dbsecrets:
  file: ./
nothingtoseehere.txt

networks:
 webnet:
  driver: bridge
  driver_opts:
   enable_ipv6: "false"
  ipam:
   driver: default
   config:
   - subnet:
172.22.254.0/24
    gateway: 172.22.254.1
 dbnet:
  driver: overlay
```

Swarm Mode



- Docker Engine v1.12.0 added support for “Swarm Mode”
  - Mode for natively managing a cluster of Docker Engines
  - Allows you to use the Docker CLI to:
    - Create a swarm
    - Deploy application services to a swarm
    - Manage swarm behavior
- Features
  - Integrated with Docker Engine, no additional orchestration software needed to create or manage a swarm
  - Decentralized design, you can deploy managers and workers using the Docker Engine
  - Declarative service model, lets you define the desired state of services in your application stack
  - Scaling, you can declare the instance count for each service
  - Desired state reconciliation, the swarm manager node monitors the cluster state and reconciles any differences between the actual state your expressed desired state
  - Multi-host networking, you can specify an overlay network for your services and your swarm manager will then automatically assigns addresses to the containers on the overlay network when it initializes or updates them
  - Service discovery, the Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers
  - Load balancing, you can expose the ports for services to an external load balancer, internally, the swarm lets you specify how to distribute service containers between nodes
  - Secure by default, each node in the swarm enforces TLS mutual authentication and encryption to secure communications between itself and all other nodes
  - Rolling updates, you can apply service updates to nodes incrementally, the swarm manager lets you control the delay between service deployment to different sets of nodes, if anything goes wrong, you can roll-back a task to a previous version of the service

Docker Swarm
vs
Swarm Mode



The old way

(many steps and commands not shown)

3
```
docker -H tcp://manager1:2376 --tlsverify --tlscacert=/home/ubuntu/.docker/ca.pem -
-tlscert=/home/ubuntu/.docker/cert.pem --tlskey=/home/ubuntu/.docker/key.pem run --
restart=unless-stopped -d -h consul1 --name consul1 -v /mnt/data -p
10.0.1.5:8300:8300    -p 10.0.1.5:8301:8301    -p 10.0.1.5:8301:8301/udp    -p
10.0.1.5:8302:8302    -p 10.0.1.5:8302:8302/udp    -p 10.0.1.5:8400:8400    -p
10.0.1.5:8500:8500    -p 172.17.0.1:53:53/udp    progrium/consul -server -
advertise 10.0.1.5 -bootstrap
```

5
```
docker -H tcp://manager1:2376 --tlsverify --tlscacert=/home/ubuntu/.docker/ca.pem
--tlscert=/home/ubuntu/.docker/cert.pem --tlskey=/home/ubuntu/.docker/key.pem run -
-restart=unless-stopped -h mgr1 --name mgr1 -d -p 3376:2376 -v
/home/ubuntu/.docker:/certs:ro swarm manage --tlsverify --tlscacert=/certs/ca.pem
--tlscert=/certs/cert.pem --tlskey=/certs/key.pem --host=0.0.0.0:2376 --replication
--advertise 10.0.1.5:2376 consul://10.0.1.5:8500/
```

2
```
DOCKER_OPTS="-H tcp://0.0.0.0:2376 --tlsverify --
tlscacert=/home/ubuntu/.docker/ca.pem --
tlscert=/home/ubuntu/.docker/cert.pem
tlskey=/home/ubuntu/.docker/key.pem"
```

ca.pem

manager1 keypair
(with subjectAltName =
IP:10.0.1.5,IP:127.0.0.1)

manager1
10.0.1.5

Steps

1. Create keys
2. Restart manager1 and node1 engine daemons with TLS flags on 2376
3. Start Consul discovery service (container)
4. Start consul client on node1 (not shown but successfully joins with consul server)
5. Start swarm manage container on 2376 and map 3376:2376
   1. Copy keys into swarm manager container via volume and specify keys and port to swarm manage command
   2. Successfully obtains leader role
6. Start swarm join container on node1
   1. Do not mount keys into container and specify to join command as appears to be unsupported option
7. Configure client with DOCKER_HOST (point to swarm) DOCKER_TLS_VERIFY=1, DOCKER_CERT_PATH...
   1. docker info command shows

```
Nodes: 1
 (unknown): 10.0.4.5:2375
  └ Status: Pending
  └ Containers: 0
  └ Reserved CPUs: 0 / 0
  └ Reserved Memory: 0 B / 0 B
  └ Labels:
  └ Error: Cannot connect to the docker engine endpoint
```

6
```
docker -H tcp://node1:2376 --tlsverify
tlscacert=/home/ubuntu/.docker/ca.pem --
tlscert=/home/ubuntu/.docker/cert.pem --
tlskey=/home/ubuntu/.docker/key.pem run -d -h join --
name join swarm join --advertise=10.0.4.5:2375
consul://10.0.4.5:8500/
```

2
```
DOCKER_OPTS="-H tcp://0.0.0.0:2376 --tlsverify --
tlscacert=/home/ubuntu/.docker/ca.pem --
tlscert=/home/ubuntu/.docker/cert.pem --
tlskey=/home/ubuntu/.docker/key.pem"
```

ca.pem

node1 keypair
(with subjectAltName =
IP:10.0.4.5,IP:127.0.0.1)

node1
10.0.4.5

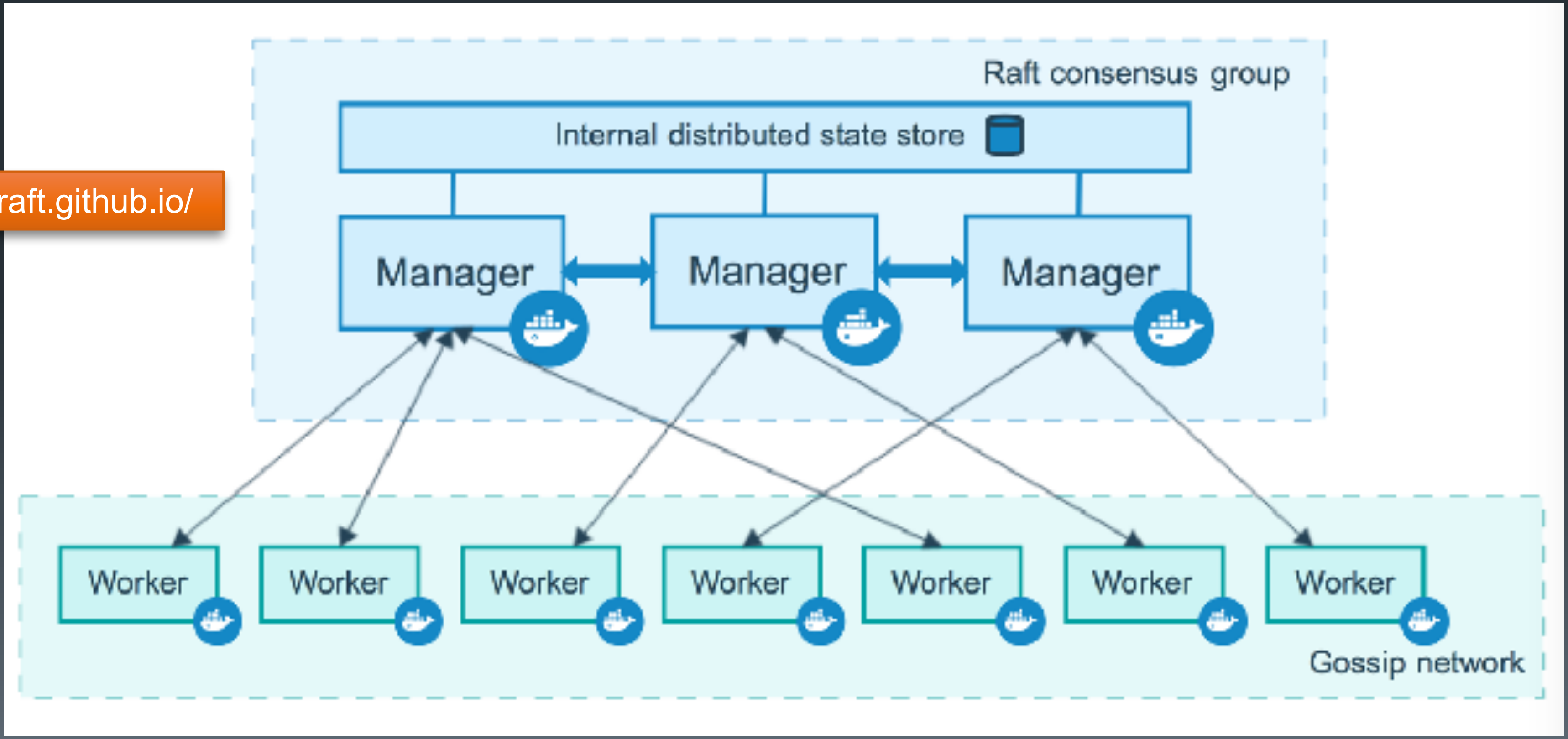
The new

```
$ docker swarm init

$ docker swarm join
```

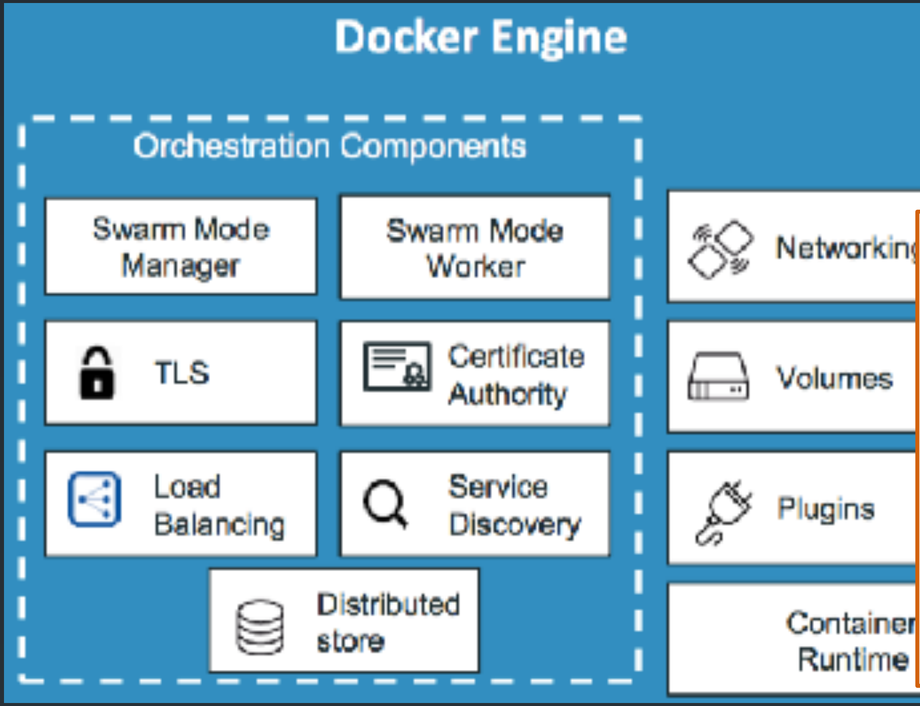
The Swarm k/v store

- Swarm managers form a Raft consensus group
  - Raft is a protocol enabling a group of distributed systems to maintain a consistent view of state
- Swarm workers form a Gossip communications group

https://raft.github.io/



Swarm Components



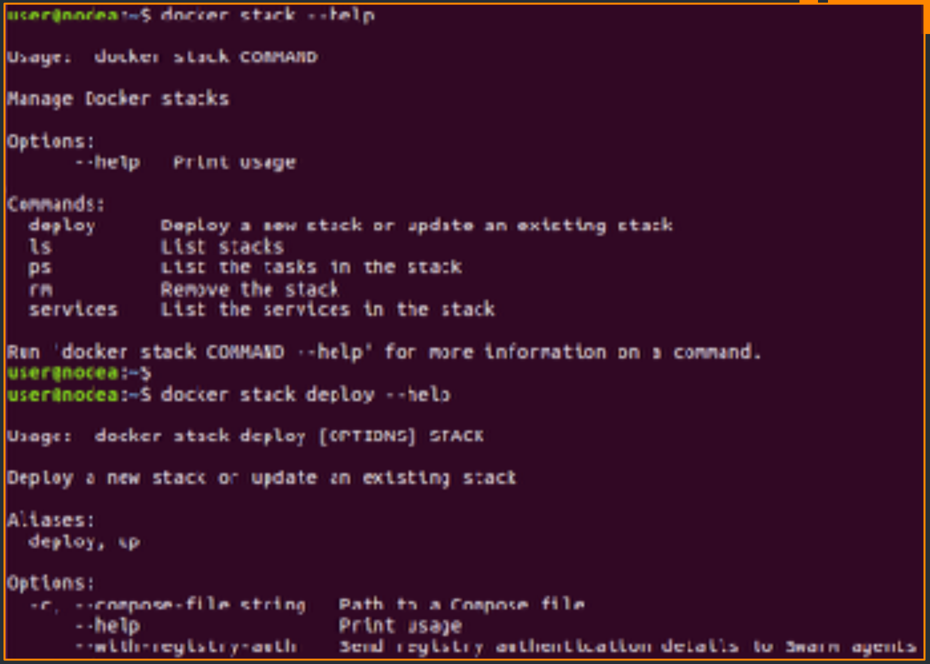
```
user@ubuntu:~$ docker service create nginx
141z2ky40mq8zxv7d7soynfxp
user@ubuntu:~$ docker service ls
ID            NAME             REPLICAS  IMAGE
COMMAND
141z2ky40mq8  awesome_babbage  1/1       nginx
user@ubuntu:~$ docker service scale 141z=2
141z scaled to 2
user@ubuntu:~$ docker service ls
ID            NAME             REPLICAS  IMAGE
COMMAND
141z2ky40mq8  awesome_babbage  2/2       nginx
user@ubuntu:~$
```

```
user@ubuntu:~$ docker swarm init --advertise-addr=10.0.0.220
Swarm initialized: current node (e7n1dpk4axdiyoo2mx4r2tb1r) is now a manager.

To add a worker to this swarm, run the following command:
    docker swarm join \
    --token
SWMTKN-1-0wpinu43yhdm5zkwbtoagqvhgpcekuyl4s2bmzeevksf0tx3fi-9k6q6w4osrj4s6000p50l8q1n \
    10.0.0.220:2377

To add a manager to this swarm, run the following command:
    docker swarm join \
    --token
SWMTKN-1-0wpinu43yhdm5zkwbtoagqvhgpcekuyl4s2bmzeevksf0tx3fi-0nf4ddcnixz0rbv4ebjl25p3k \
    10.0.0.220:2377
```

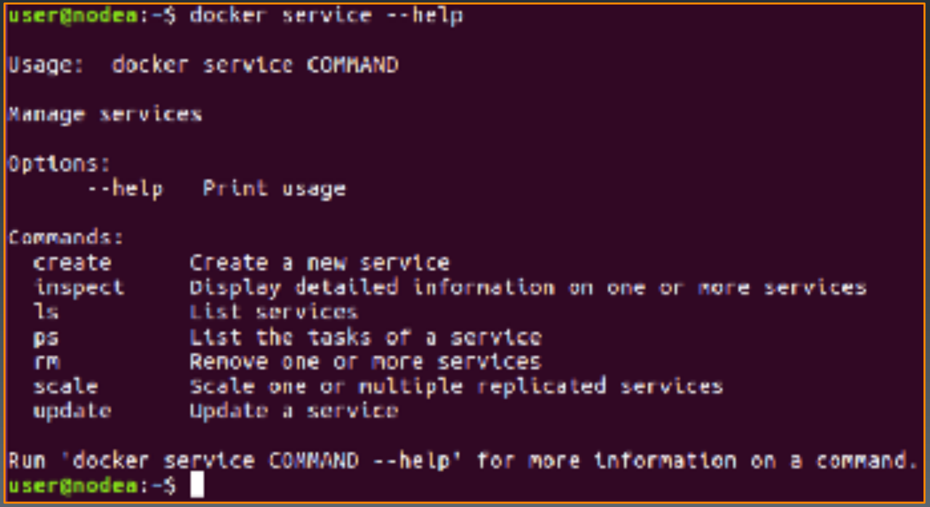
docker deploy & services

- `docker stack` is used to manage Docker stacks
  - Sub commands:
    - `deploy` Deploy or update a stack
    - `ls` List stacks
    - `ps` List tasks in the stack
    - `rm` Remove the stack
    - `services` List services in the stack
  - When used with the `-c` option, deploy will use a Compose v3 file as its source

- `docker service` is used to manage and monitor stack services with swarm
  - Sub commands:
    - `create` Create a new service
    - `inspect` Display detailed information
    - `logs` Fetch the logs of a service
    - `ls` List services
    - `ps` List the tasks of a service
    - `rm` Remove a service
    - `scale` Scale service(s)
    - `update` Update a service



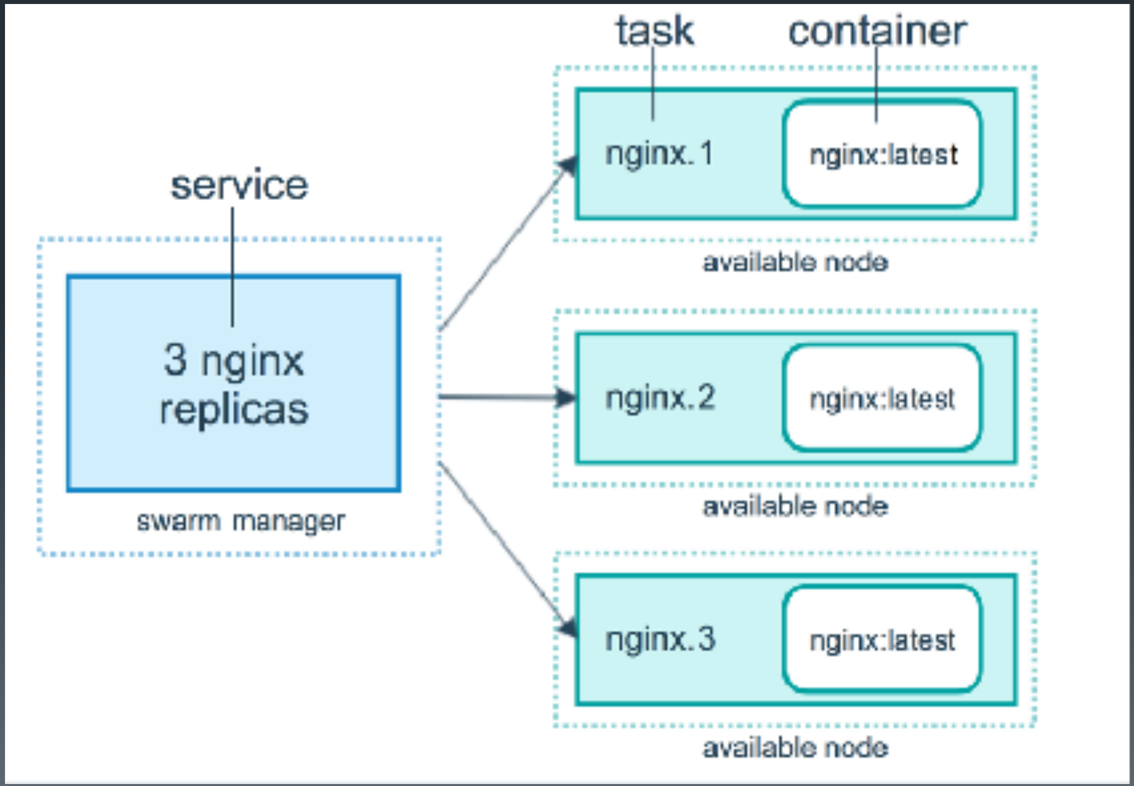
```
user@nodea:~$ docker stack --help

Usage:  docker stack COMMAND

Manage Docker stacks

Options:
      --help   Print usage

Commands:
  deploy      Deploy a new stack or update an existing stack
  ls          List stacks
  ps          List the tasks in the stack
  rm          Remove the stack
  services    List the services in the stack

Run 'docker stack COMMAND --help' for more information on a command.
user@nodea:~$
user@nodea:~$ docker stack deploy --help

Usage:  docker stack deploy [OPTIONS] STACK

Deploy a new stack or update an existing stack

Aliases:
  deploy, up

Options:
  -c, --compose-file string    Path to a Compose file
      --help                   Print usage
      --with-registry-auth     Send registry authentication details to Swarm agents
```



```
user@nodea:~$ docker service --help

Usage:  docker service COMMAND

Manage services

Options:
      --help   Print usage

Commands:
  create      Create a new service
  inspect     Display detailed information on one or more services
  ls          List services
  ps          List the tasks of a service
  rm          Remove one or more services
  scale       Scale one or multiple replicated services
  update      Update a service

Run 'docker service COMMAND --help' for more information on a command.
user@nodea:~$ 
```

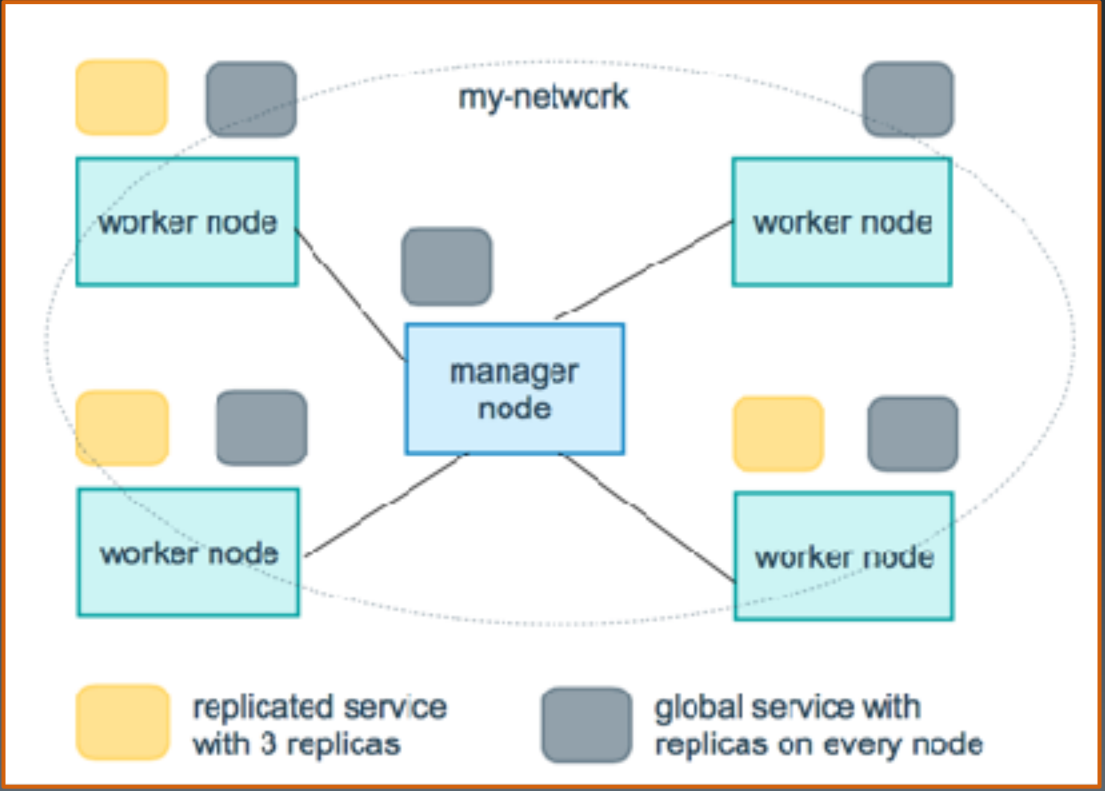
Services

- Creating a service sets the service definition as the desired state for the service
- The manager then schedules the service on nodes in the swarm as one or more replica tasks
- Tasks run independently of each other on nodes
  - Each task invokes exactly one container
  - A task is analogous to a “slot” where the scheduler places a container
  - Once the container is live, the scheduler recognizes that the task is in a running state
  - If the container fails health checks or terminates, the task terminates



Service Types

- There are two types of service deployments:
  - Replicated
  - Global
- Replicated
  - Specify the number of task replicas you want to run
  - Scheduler determines placement
- Global
  - One task replica runs on every node
  - Each time you add a node to the swarm, the orchestrator creates a task and the scheduler assigns the task to the new node
  - Good candidates for global services are monitoring agents, an security scanners, log forwarders, etc.
  - Similar to a Kubernetes DaemonSet


my-network
worker node
worker node
manager node
worker node
worker node
replicated service with 3 replicas
global service with replicas on every node

Summary

- Production container orchestration systems offer a range of critical services for microservice based applications
  - Scaling
  - Rollout
  - Recovery
  - Deployment
  - Load balancing
  - Service Discovery
- Compose enables multi-container deployments on a single host or on a cluster of Swarm nodes
- Swarm Mode is the integrated Swarm functionality introduced in Docker 1.12
- Swarm makes clusters as easy as Docker makes containers

Lab 4

- Docker Swarm Mode

The End

Many thanks for attending!

Appendix

- Additional topics of interest for your reference
  - Links
  - Books
  - Docker GUIs
  - Migrating native systems into containers
  - Changing File System Drivers

Docker Best Practices

- Design containers to be ephemeral
  - You should be able to terminated and restart containers with no concern for their state
  - You should “move” containers between hosts by terminating the container on the source host and running a new container from the image on the target host (no need to move stateless services)
- Design each container to run a single service
  - Container technology works best with microservice oriented, cloud native designs (though it can be greatly beneficial in other application architectures as well)
  - Commands like docker signal and ps interact with the PID 1 process
- Use Dockerfiles to build images
  - Docker commit is great for experimentation and debugging but does not build an “Infrastructure as Code” style repeatable process
  - Dockerfile build operations can be cleanly integrated into existing CI/CD pipelines
- Use Volumes Appropriately
  - Volumes can be shared between containers using --volumes-from
  - Use volumes to share a directory between containers and when writing large amounts of data to a directory
  - Containers should be stateless, application state belongs in volumes, independent of containers and container lifespans
- Use the Data Container Pattern with Volumes
  - Volumes are reference counted and deleted when no longer used by a container when “docker rm -v” is used
    - A dedicated (empty) Data Container associated with such a volume allows the volume to have a life span independent of the other ephemeral containers using the volume
    - The Data Container associated with the disk volume need not ever be run for other containers to use the volume
    - The Data Container allows the underlying volume to be encapsulated, hiding details from other containers (which use --volumes-from to access the Data Container’s volumes)
    - Base Data Containers on the same image the running containers which access the VOLUME use
  - Post Docker v1.9 “docker volume” can be used instead to create shared volumes using volume drivers or host volumes
  - Docker managed host paths are deprecated
- Choose a minimum permissions User [in Docker v1.10 ROOT is a minimum permission user!!]
  - Pre Docker 1.10
    - By default docker containers run as root with full control of the host system prior to Docker v1.10
    - Use the USER instruction to specify a non-root user for containers to run as
    - You can create the users and groups you need in the Dockerfile
    - Images inherit their parent image’s USER, override the parent image USER with a local USER statement when appropriate
  - Docker 1.10 adds user namespaces allowing root to be used safely within containers
    - This is the best of both worlds (root has full control in the container and no control outside)
    - This must be enabled with the daemon switch --userns-remap and a supporting kernel
    - With user namespaces ROOT is a perfectly acceptable inside container user

Much borrowed from Project Atomic: http://www.projectatomic.io/docs/docker-image-author-guidance/

Anti Patterns

1. Stateful containers
   - Use volumes for state and isolate state management in the appropriate microservice tier
2. Shipping containers
   - Containers should be ephemeral, images are the unit of distribution, ship images not containers
3. Large images
   - Production images should be small and easy to push/pull, only including what is needed
4. Too few or too many filesystem layers
   - You should probably have an OS layer (e.g. Centos:7), a runtime layer (e.g. nodejs) and an app layer (e.g. your js code) to simplify and speed changes and updates, to few layers makes builds and patches expensive, too many slows down union filesystem driver operation
5. Build automation with docker commit
   - Commit is best suited for ad hoc experiments, repeatable projects builds should be managed with Dockerfiles
6. Multiple concerns in a single container
   - Containers are best suited for packaging microservices
7. SSH in a container
   - (see above) Containers should be managed from the outside not the inside and treated like services not systems
8. Relying on the latest tag
   - Always explicitly choose a base image: “FROM ubuntu:14.04.3”   -- not --- “FROM ubuntu”
9. Storing credentials/secrets in an image/container
   - Environment variables are the best way to set secrets, keeping sensitive data out of source code and off disk
10. Depending on specific IP addresses
    - Many virtual networking technologies are at plat today and specific IP address assignment is a brittle approach, use service discovery mechanisms to find dependencies at runtime
11. Restarting or moving containers
    - Containers should be ephemeral, never restart a container (on the same node or a different node)
    - To “move” a container from one host to another you should terminate and delete the container on host A and start a new copy of the image on host B

Much borrowed from Rafael Benevides & Erik Dasque

Links

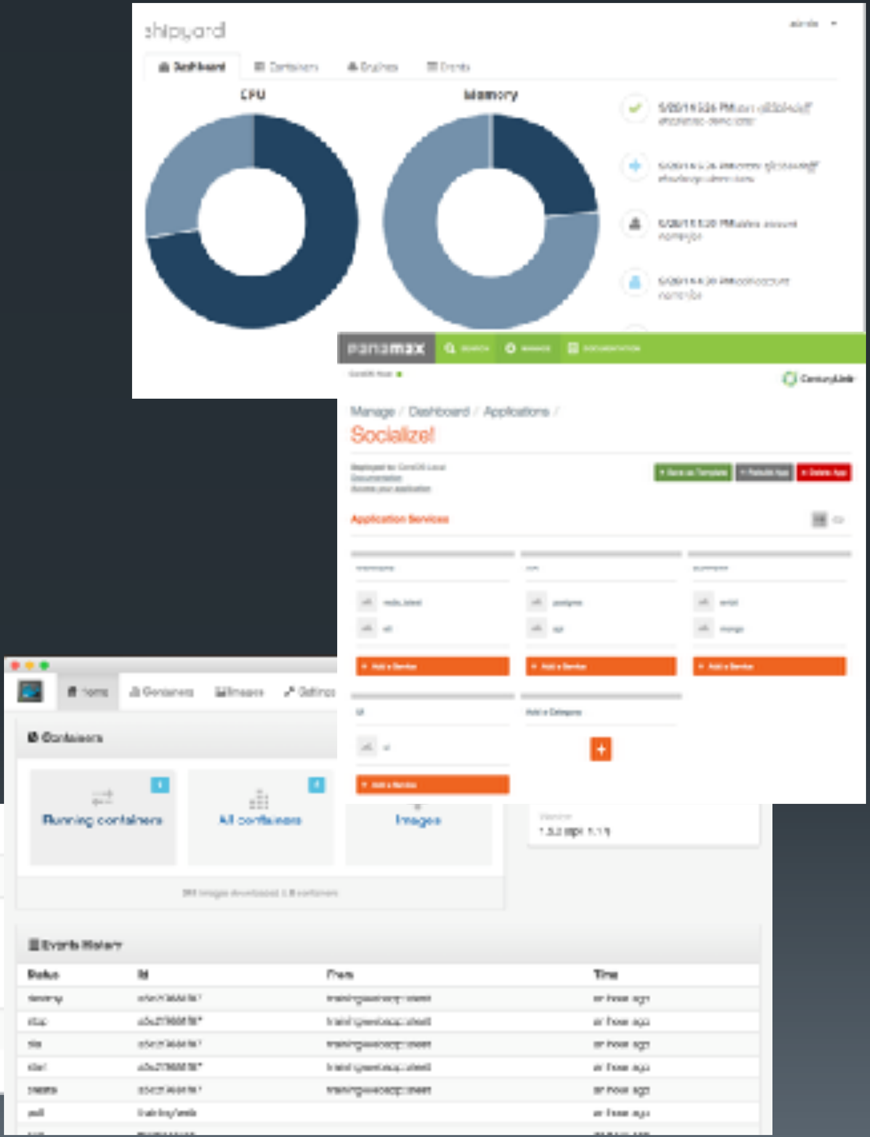
- Docker homepage - https://www.docker.com/
- Docker Hub - https://hub.docker.com
- Docker blog - http://blog.docker.com/
- Docker documentation - https://docs.docker.com/
- Docker code on GitHub - https://github.com/docker/docker
- Docker Forge - https://github.com/dockerforge
  - Collection of Docker tools, utilities, and services
- Docker mailing list - https://groups.google.com/forum/#!forum/docker-dev
- Docker on IRC: irc.freenode.net channel #docker
- Docker on Twitter - #docker
- StackOverflow - http://stackoverflow.com/tags/docker
- Docker Cheat Sheet - https://github.com/wsargent/docker-cheat-sheet
- Docker & Jenkins - https://www.youtube.com/watch?v=7K-8SFmVDIM
- Getting Started with Docker - https://dzone.com/refcardz/getting-started-with-docker-1
- Overlay network options - http://www.slideshare.net/syed1/docker-meetup-52918010

Books

- The Docker Book, Turnbull
  - http://www.dockerbook.com/
- O'Reilly Books:
  - Docker Up and Running
  - Docker Cookbook
  - Using Docker
- Manning Books
  - Docker in Action
  - Docker in Practice

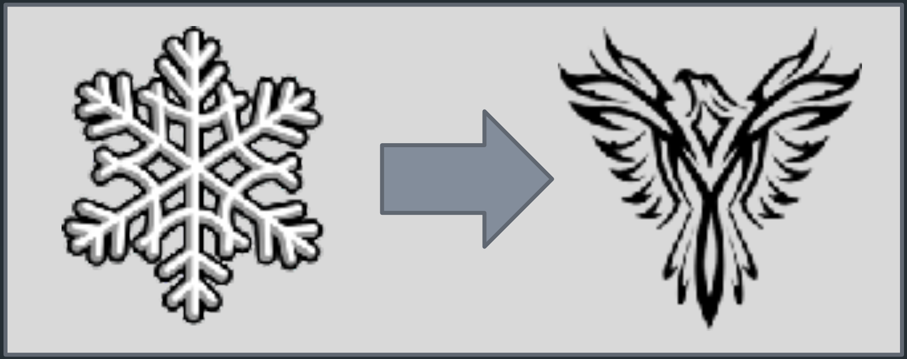
Docker UIs

- There are several web UIs available for Docker
  - Kitematic – Acquired by Docker and provided as part of Docker Toolbox, desktop container management
  - Tutum (now Docker Control Plane) – Enterprise targeted container deployment and management [subscription based Docker product]
  - Shipyard – supports management of Docker hosts and containers (built on the Citadel cluster management toolkit)
    - Provides web and CLI interfaces
    - http://shipyard-project.com
  - Panamax – web based containerized application management based on Fleet
    - http://panamax.io/
  - Simple Docker UI for Chrome
    - https://github.com/felixgborrego/docker-ui-chrome-app
  - DockerUI - a web interface for the Docker Remote API
    - https://github.com/crosbymichael/dockerui



Containers:

Id	Image	Command	Created	Status
9c1a24d...	[illegible]	/bin/sh -c /usr/local/bin/sentry --config=/sentry.conf.py start	[illegible]	Up 4 hours
d60b3f6...	[illegible]	/bin/sh -c /usr/bin/redis-server /etc/redis/redis.conf	[illegible]	Ghost

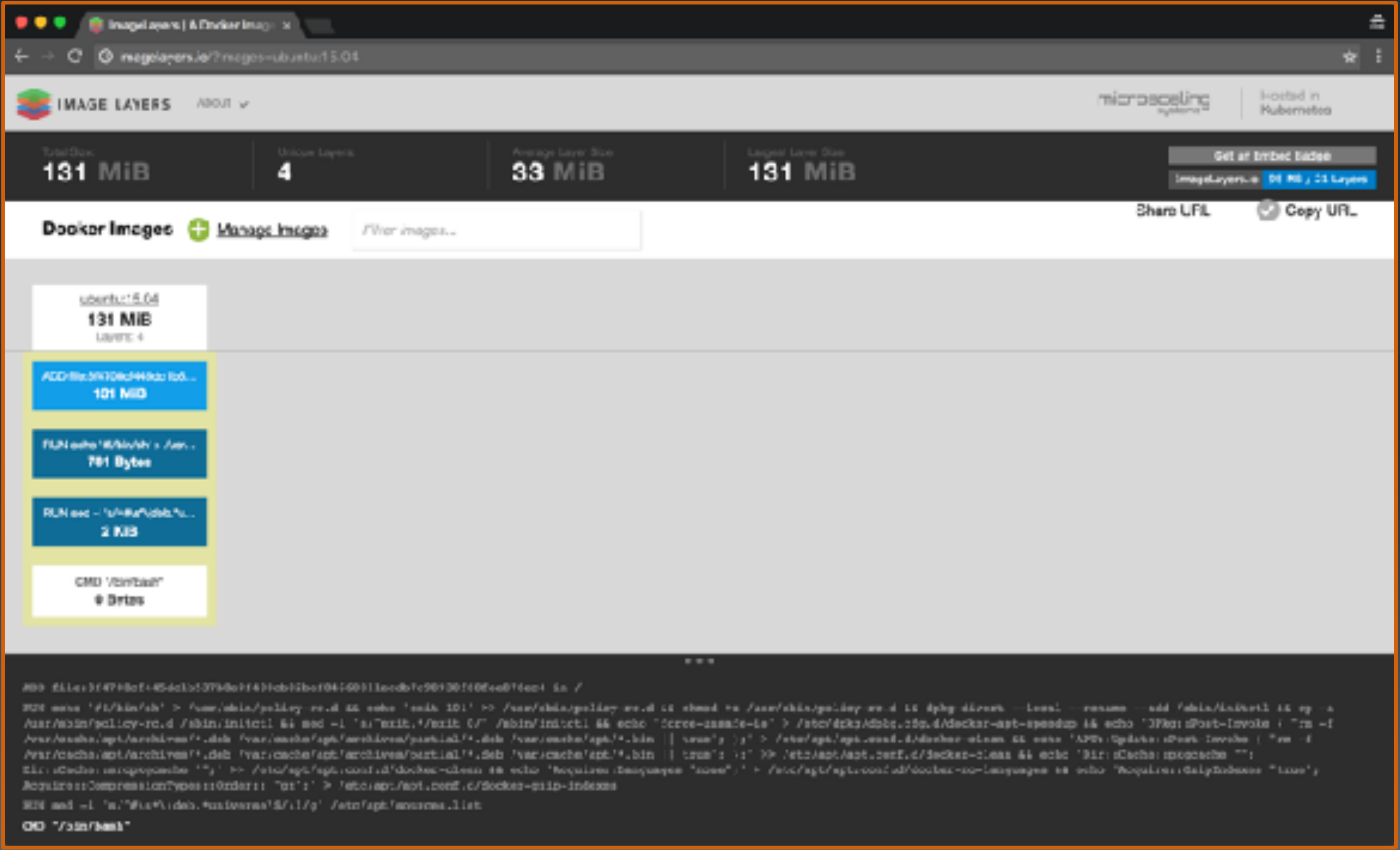


Turning a Snowflake into a Phoenix

- Snowflake servers have one of a kind configurations
  - Modified directly by admins over time
  - The server’s configuration is not easy to reproduce
  - Not a good basis for modern IaC (Infrastructure as Code)
- Phoenix servers can be destroyed and restarted from a base over and over again
  - Like containers launched from images/Dockerfiles or VMs launched from a snapshot or nodes built from a Puppet manifest or a Chef recipe
- There are several ways you can generate a Docker image from a running snowflake system
  - Tar the entire file system and use docker import to create an image from the tarball
  - Use blueprint or other similar tools
    - https://zwischenzugs.wordpress.com/2015/05/24/convert-any-server-to-a-docker-container/

Image Tools

- Image layer explorers
  - imagelayers.io



Total Size: 131 MiB | Unique Layers: 4 | Average Layer Size: 33 MiB | Largest Layer Size: 131 MiB

ubuntu:15.04
131 MiB
Layers: 4

ADD file:[illegible]...
101 MiB

RUN echo '#!/bin/sh' > /usr...
701 Bytes

RUN sed -i 's/^#\s*\(deb.*u...
2 KiB

CMD ["/bin/bash"]
0 Bytes

```
ADD file:[illegible] in /
RUN echo '#!/bin/sh' > /usr/sbin/policy-rc.d && echo 'exit 101' >> /usr/sbin/policy-rc.d && chmod +x /usr/sbin/policy-rc.d && dpkg-divert --local --rename --add /sbin/initctl && cp -a
/usr/sbin/policy-rc.d /sbin/initctl && sed -i 's/^exit.*/exit 0/' /sbin/initctl && echo 'force-unsafe-io' > /etc/dpkg/dpkg.cfg.d/docker-apt-speedup && echo 'DPkg::Post-Invoke { "rm -f
/var/cache/apt/archives/*.deb /var/cache/apt/archives/partial/*.deb /var/cache/apt/*.bin || true"; };' > /etc/apt/apt.conf.d/docker-clean && echo 'APT::Update::Post-Invoke { "rm -f
/var/cache/apt/archives/*.deb /var/cache/apt/archives/partial/*.deb /var/cache/apt/*.bin || true"; };' >> /etc/apt/apt.conf.d/docker-clean && echo 'Dir::Cache::pkgcache "";
Dir::Cache::srcpkgcache "";' >> /etc/apt/apt.conf.d/docker-clean && echo 'Acquire::Languages "none";' > /etc/apt/apt.conf.d/docker-no-languages && echo 'Acquire::GzipIndexes "true";
Acquire::CompressionTypes::Order:: "gz";' > /etc/apt/apt.conf.d/docker-gzip-indexes
RUN sed -i 's/^#\s*\(deb.*universe\)$/\1/g' /etc/apt/sources.list
CMD "/bin/bash"
```