Course Overview

- Docker concepts overview
- Docker installation (on a Linux host)
- Searching for and fetching Docker images
- Running single Docker containers
- Docker container management
- Creating a new Docker image from scratch
- Publishing your Docker image to hub docker
- Autobuilds on Docker Hub
- Composing multi-container applications stacks using Docker compose (basic example)
- Demonstration of using Rancher as a Docker orchestration tool

What is not covered?

- Swarms / clusters / remote management
- Advanced permissions
- Secrets
- Remote storage
- Rancher in-depth
- Load balancing

Key Docker Concepts Overview

image	Dockerfile	port	build
container	docker-compose.yml	link	run
repository	rancher-compose.yml	volume	exec

A little Philosopy

Docker is great when you want to densely pack many isolated services onto each server. If you prefer to take the approach of e.g. deploying many microservers (e.g. AWS micro instances) you might be better off looking at using a configuration management tool like ansible. Although even then docker still has some advantages as it is nice to be able to just check out an image onto a micro instance and spin it up knowing everything will 'just work'.

Not virtual machines

If you go with docker really avoid the temptation of treating docker containers like virtual machines. Even though you can run multiple services in each container (e.g. using supervisord) you should really architect your docker images so that they are virtual application appliances, with each appliance doing one thing only (like the old unix mantra eh?). So have a discrete appliance for postgis, another for uwsgi, another for mapserver, another for geoserver etc. rather than mixing things into the same service.

Stateless

You should do everything you can to make your containers stateless. Or put differently, you should be able to confidently destroy and redeploy a container as a fresh instance with no loss of data or configuration. This means that your investment should be in building the images on which your containers are based and not the containers themselves. In practice this means that you should e.g. mount your postgres cluster from a host volume, store your user uploaded files in a host volume and store no generated data inside the container itself.

Configuration management

You should avoid the temptation to build your own container orchestration tools. Rather use a tool like docker-compose that uses a simple yaml file to define your micro services architecture and can be used to reliably spin up a working configuration.

Autobuilds

Lean how to publish your images into hub.docker.com and have them build on push to your repo and build when the upstream container your docker image is based on gets an update (e.g. a security fix). Also invest the time in learning to tag different versions in hub so that you can have a known good working configuration against specific image versions.

Layering

Build up your images in layers. Start with a standard base image e.g. 'ubuntu:trusty' then add python and save that as a new standard image, then add django and save that as a new base image etc. This way each virtual application you create is defined by only the thinnest amount of configuration and software deployment possible, and you can share the underlying logic of the lower layers between as many images as possible.

Images

search

```
docker image pull busybox

docker images
```

What's an Image?

An image is an inert, immutable, file that's essentially a snapshot of a container. Images are created with the build command, and they'll produce a container when started with run. Images are stored in a Docker registry such as registry.hub.docker.com. Because they can become quite large, images are designed to be composed of layers of other images, allowing a miminal amount of data to be sent when transferring images over the network.

Local images can be listed by running `docker images`:

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
ubuntu	13.10	5e019ab7bf6d	2 months ago	180 MB
ubuntu	14.04	99ec81b80c55	2 months ago	266 MB
ubuntu	latest	99ec81b80c55	2 months ago	266 MB
ubuntu	trusty	99ec81b80c55	2 months ago	266 MB
<none>	<none>	4ab0d9120985	3 months ago	486.5 MB

Some things to note:

1. IMAGE ID is the first 12 characters of the true identifier for an image. You can create many tags of a given image, but their IDs will all be the same (as above).
2. VIRTUAL SIZE is *virtual* because its adding up the sizes of all the distinct underlying layers. This means that the sum of all the values in that column is probably much larger than the disk space used by all of those images.
3. The value in the REPOSITORY column comes from the `-t` flag of the `docker build` command, or from `docker tag`-ing an existing image. You're free to tag images using a nomenclature that makes sense to you, but know that docker will use the tag as the registry location in a `docker push` or `docker pull`.
4. The full form of a tag is `[REGISTRYHOST/][USERNAME/]NAME[:TAG]`. For `ubuntu` above, REGISTRYHOST is inferred to be `registry.hub.docker.com`. So if you plan on storing your image called `my-application` in a registry at `docker.example.com`, you should tag that image `docker.example.com/my-application`.
5. The TAG column is just the [:TAG] part of the *full* tag. This is unfortunate terminology.
6. The `latest` tag is not magical, it's simply the default tag when you don't specify a tag.
7. You can have untagged images only identifiable by their IMAGE IDs. These will get the `<none>` TAG and REPOSITORY. It's easy to forget about them.

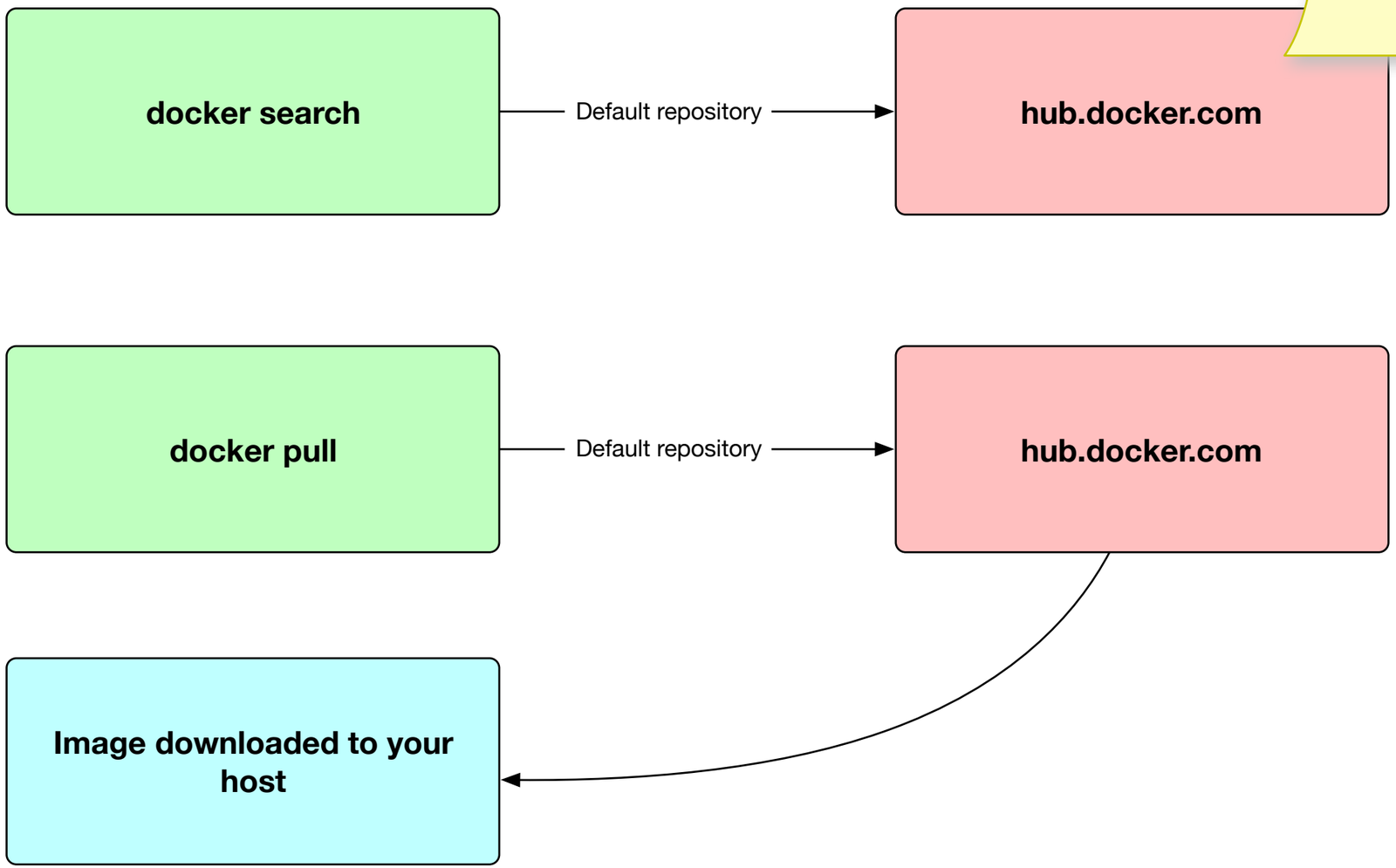
More info on images is available from the Docker docs and glossary.

Source: https://stackoverflow.com/a/26960888

Finding and fetching an image

image

```
docker search 2048

docker pull alexwhen/
docker-2048
```



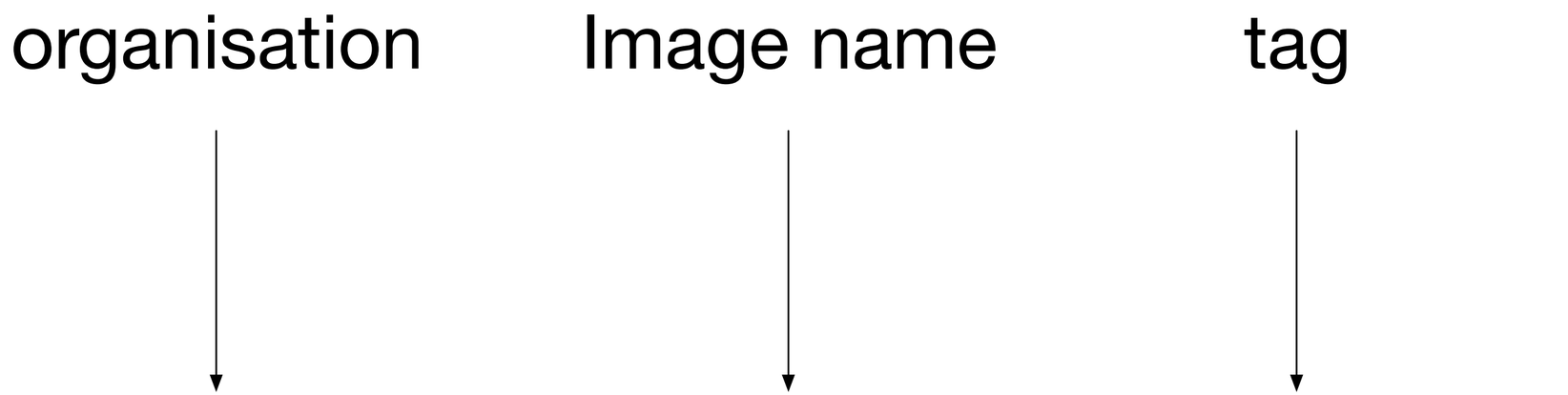
Running an Image

run

```
docker run busybox echo "Hello world"
```


```
Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

Options:

      --cpuset-mems string             MEMs in which to allow execution (0-3, 0,1)
  -d, --detach                         Run container in background and print
container ID
      --dns list                       Set custom DNS servers
      --entrypoint string              Overwrite the default ENTRYPOINT of the
image
  -e, --env list                       Set environment variables
      --env-file list                  Read in a file of environment variables
      --help                           Print usage
  -h, --hostname string                Container host name
  -i, --interactive                    Keep STDIN open even if not attached
      --link list                      Add link to another container
      --mount mount                    Attach a filesystem mount to the container
      --name string                    Assign a name to the container
  -p, --publish list                   Publish a container's port(s) to the host
      --restart string                 Restart policy to apply when a container
exits (default "no")
      --rm                             Automatically remove the container when it
exits
  -t, --tty                            Allocate a pseudo-TTY
  -v, --volume list                    Bind mount a volume
  -w, --workdir string                 Working directory inside the container
```

Image Naming



docker

KARTOZA
OPEN SOURCE GEOSPATIAL SOLUTIONS

Containers

container

```
docker run -ti -d busybox tail
-f /dev/null

docker ps

docker ps -a
```

What's a container?

To use a programming metaphor, if an image is a class, then a container is an instance of a class—a runtime object. Containers are hopefully why you're using Docker; they're lightweight and portable encapsulations of an environment in which to run applications.
View local running containers with `docker ps`:

```
CONTAINER ID        IMAGE                                COMMAND                CREATED
STATUS              PORTS                     NAMES
f2ff1af05450        samalba/docker-registry:latest       /bin/sh -c 'exec doc   4 months ago        Up
12 weeks          0.0.0.0:5000->5000/tcp   docker-registry
```

Here I'm running a dockerized version of the docker registry, so that I have a private place to store my images. Again, some things to note:

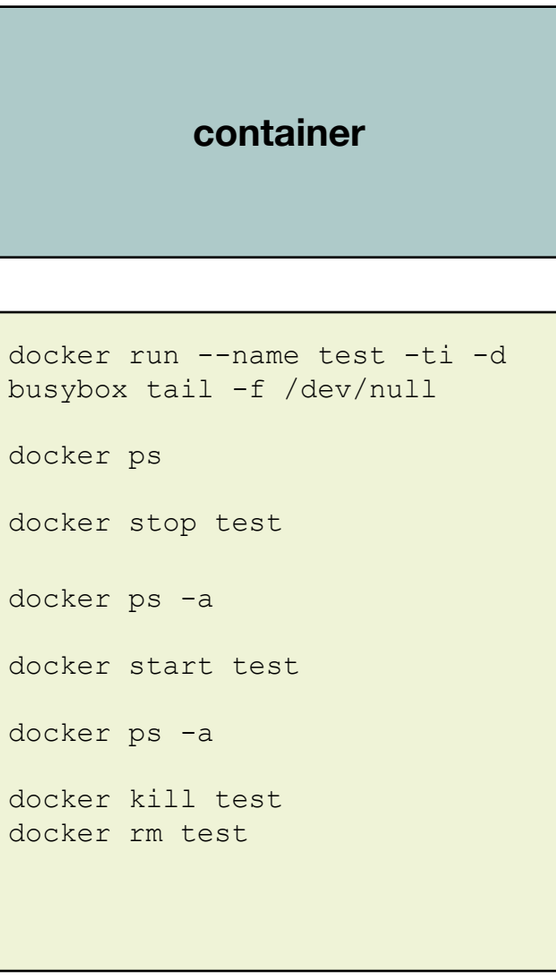
1. Like IMAGE ID, CONTAINER ID is the true identifier for the container. It has the same form, but it identifies a different kind of object.
2. `docker ps` only outputs *running* containers. You can view all containers (*running* or *stopped*) with `docker ps -a`.
3. NAMES can be used to identify a started container via the `--name` flag.

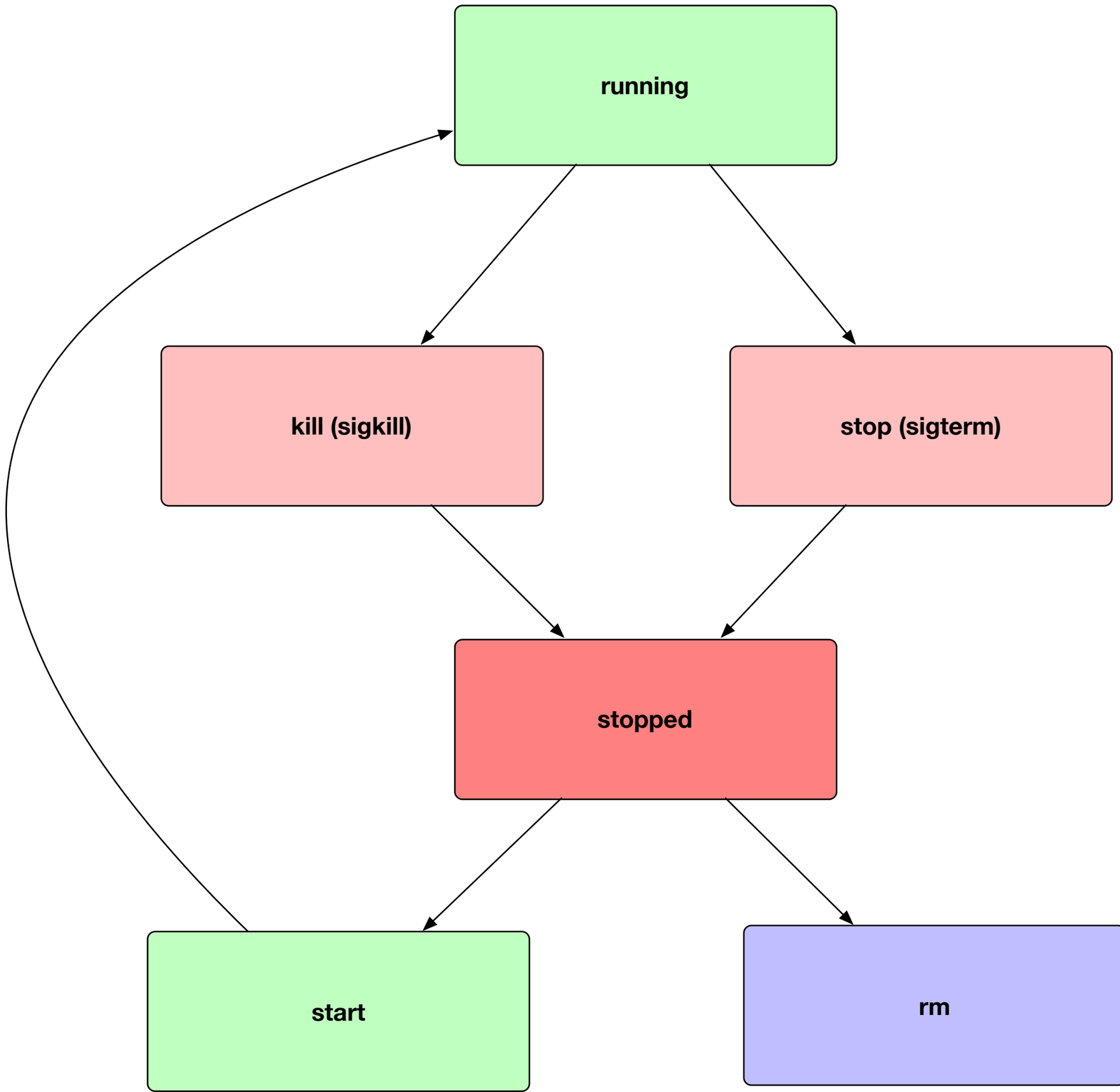
Source: https://stackoverflow.com/a/26960888

Container States

container

```
docker run --name test -ti -d
busybox tail -f /dev/null

docker ps

docker stop test

docker ps -a

docker start test

docker ps -a

docker kill test
docker rm test
```





Container Management

```
Usage:  docker container COMMAND

Manage containers

Options:


Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes
```

docker

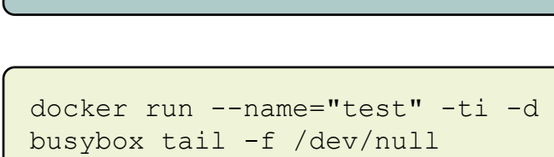
KARTOZA

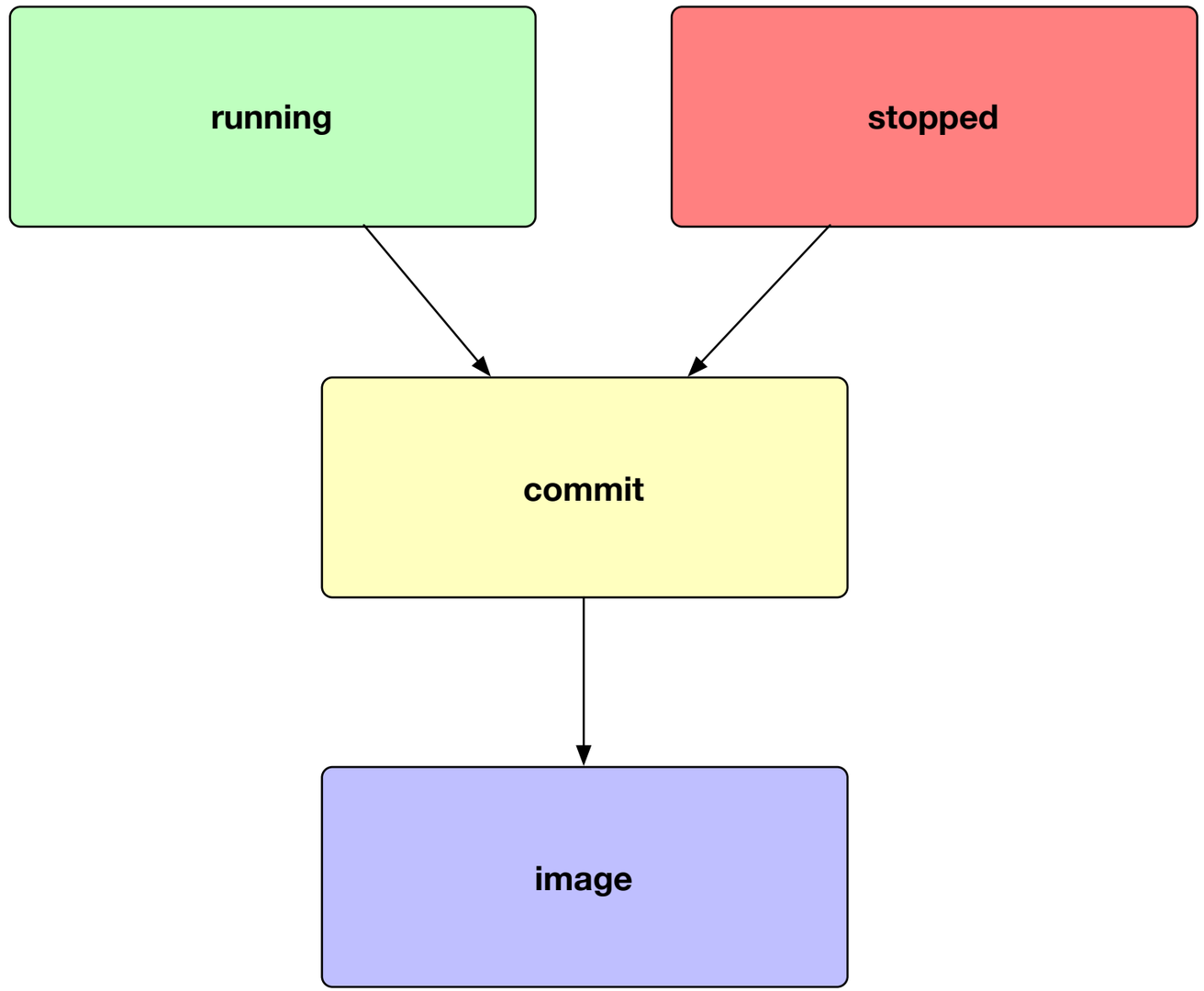
OPEN SOURCE GEOSPATIAL SOLUTIONS

Converting a container to an Image

commit

```
docker run --name="test" -ti -d
busybox tail -f /dev/null

docker ps

docker commit test kartoza/
test:latest

docker images | grep
test
```





Converting a container to an Image

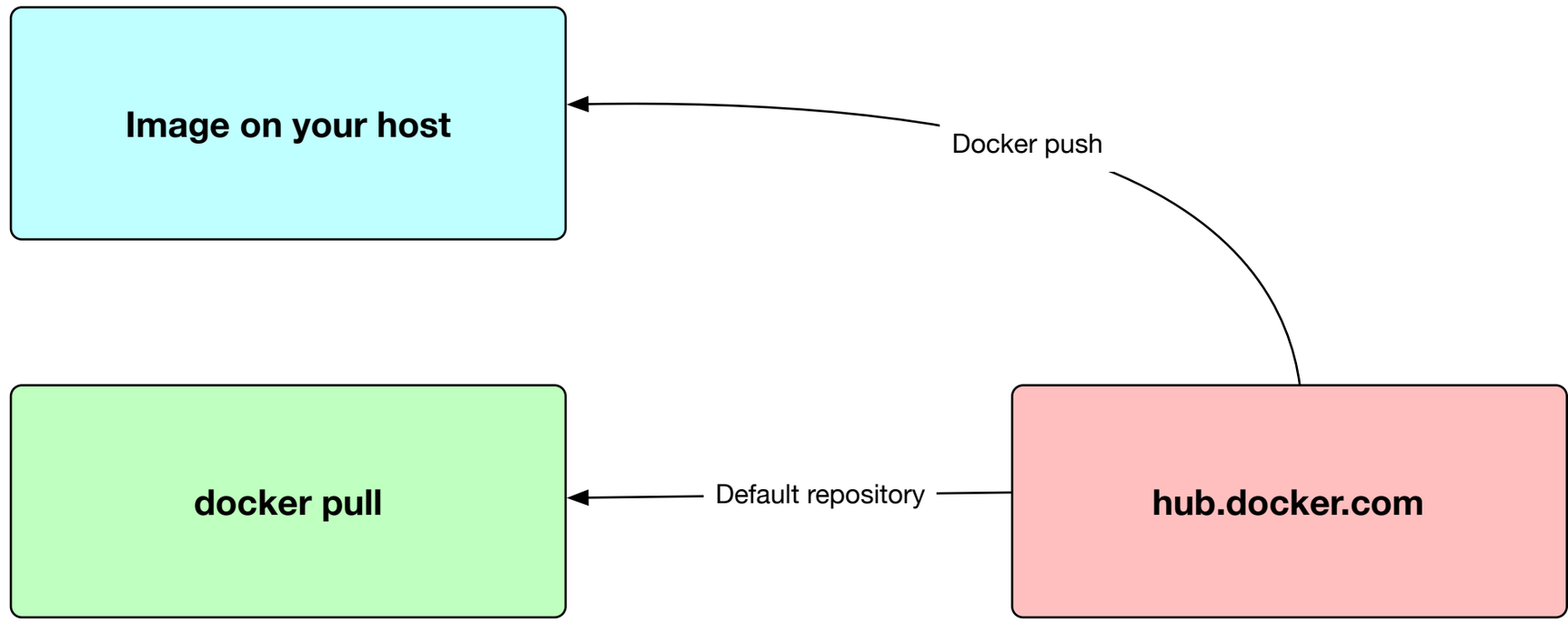
Export

```
docker export test > test.tar
cat test.tar | docker import - test/foo
docker images | grep foo
docker run -ti test/foo /bin/ash


/ # ls
HelloWorld.txt  etc             root            usr
bin             home            sys             var
dev             proc            tmp
```

Publishing an image

Push

```
docker run --name="test" -ti -d
busybox tail -f /dev/null

docker ps

docker commit test timlinux/
test:latest

docker images | grep
test

docker login

docker push timlinux/
test:latest
```



The Dockerfile

https://docs.docker.com/v17.09/engine/userguide/eng-image/dockerfile_best-practices/#expose

FROM	ENTRYPOINT (Application binary)	ENV
RUN	CMD (Application flags)	USER
WORKDIR	VOLUME	ADD / COPY

The Dockerfile

build

https://docs.docker.com/v17.09/engine/userguide/eng-image/dockerfile_best-practices/#expose

```
docker build -t
kartoza/python-server .

docker run -p 8000:8000
kartoza/python-server
```

```
# Let's use a nice small base image, only 1.8mb!
FROM alpine:3.5
# The guy who maintains this
MAINTAINER Tim Sutton <tim@kartoza.com>
# Install some alpine packages ...
# See https://pkgs.alpinelinux.org/packages for a list of available alpine packages
RUN apk add --update \
    python \
    python-dev \
    py-pip \
    build-base \
  && pip install virtualenv \
  && rm -rf /var/cache/apk/*
WORKDIR /
EXPOSE 8000
ENTRYPOINT ["python"]
CMD ["-m", "SimpleHTTPServer"]
```

Read about Copy On Write: https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/#the-copy-on-write-cow-strategy

Host / bindmount volumes

Note: -v will be replaced by —mount in the future (which are not covered in this training)

https://docs.docker.com/storage/volumes/#start-a-service-with-volumes

-v

```
docker build -t
kartoza/python-server .

docker run -v /Users/
timlinux/dev/docker/
docker-training:/home -
p 8000:8000 kartoza/
python-server
```

Directory listing for /

- .dockerenv
- bin/
- dev/
- etc/
- home/
- lib/
- media/
- mnt/
- proc/
- root/
- run/
- sbin/
- srv/
- sys/
- tmp/
- usr/
- var/

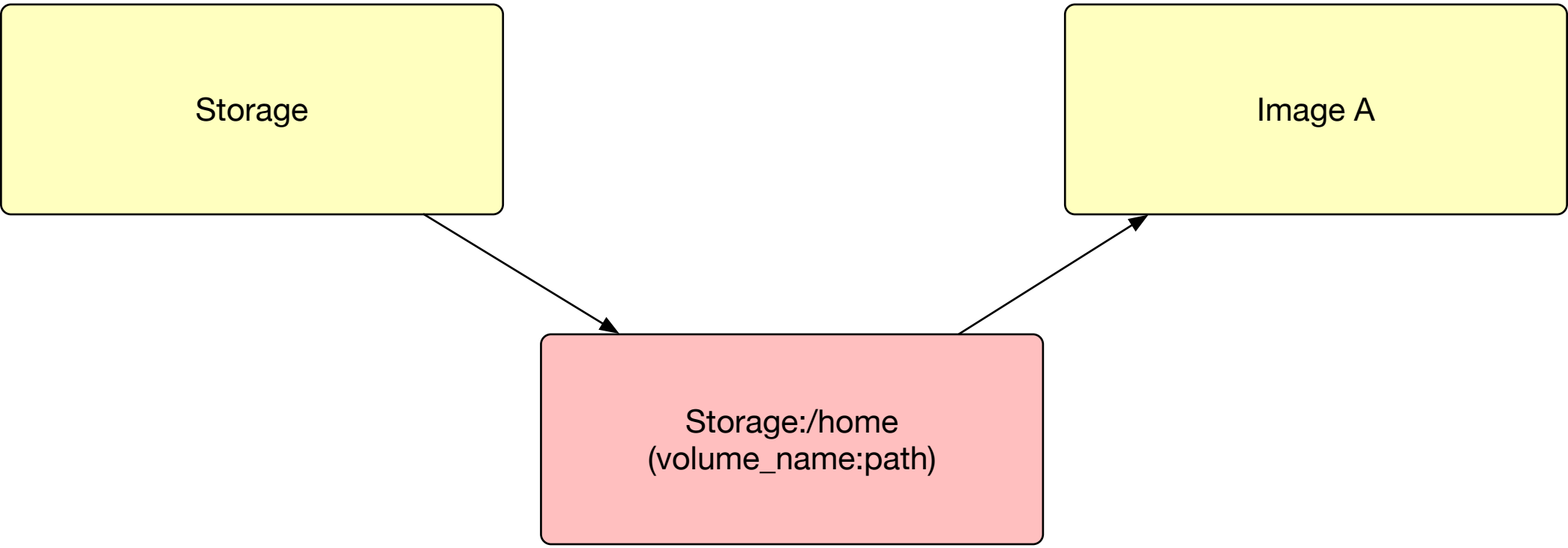
Directory listing for /home/

- build_and_run_simple_server.sh
- build_and_run_simple_server_with_volume.sh
- Dockerfile
- KartozaDockerTrainingResources.pdf

docker

KARTOZA
OPEN SOURCE GEOSPATIAL SOLUTIONS

Storage Containers



docker

KARTOZA
OPEN SOURCE GEOSPATIAL SOLUTIONS

Storage volumes

Note: -v will be replaced by —mount in the future (which are not covered in this training)

https://docs.docker.com/storage/volumes/#start-a-service-with-volumes

In this example we create a storage volume, mount it in an image, copy some data from the host to it, start a second container that shares the volume and then connect to it from a web browser.

HelloWorld.txt

```
Hello World
```

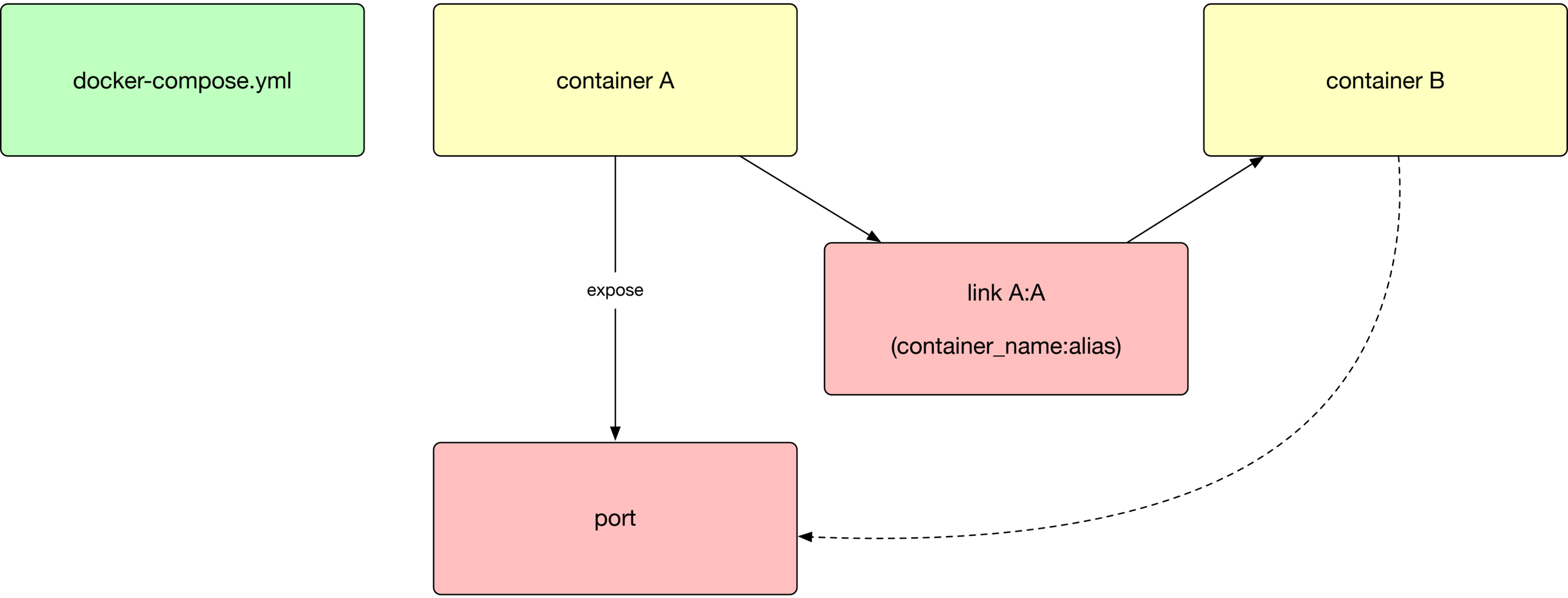
Running …

```
docker build -t kartoza/python-server .

docker volume create Storage

docker run --name="pyserve" -v Storage:/home -p 8000:8000 -dt kartoza/python-server

docker cp HelloWorld.txt pyserve:/home/

docker run --name="pyserve2" -v Storage:/home -p 8001:8000 -dt kartoza/python-server
```

Container Linking

Note: Container linking will be deprecated in future in favour of docker networks (which are not covered in this training)

See https://docs.docker.com/network/network-tutorial-standalone/ for the ‘new’ way…



Container Linking

Note: Container linking will be deprecated in future in favour of docker networks (which are not covered in this training)

See https://docs.docker.com/network/network-tutorial-standalone/ for the 'new' way…

In this example we will create a new Dockerfile and image, then run it, linking it to another running container (and storage volume)

wget-show.sh

```
#!/bin/ash
# Above is not a typo - alpine shell is ash not bash
wget -O hello.txt http://pyserve:8000/home/HelloWorld.txt
cat hello.txt
```

Dockerfile-wget

```
# Let's use a nice small base image, only 1.8mb!
FROM alpine:3.5
# The guy who maintains this
MAINTAINER Tim Sutton <tim@kartoza.com>
# Install some alpine packages ...
# See https://pkgs.alpinelinux.org/packages for a list of available alpine packages
RUN apk add --update wget && rm -rf /var/cache/apk/*
ADD wget-show.sh /wget-show.sh
RUN chmod +x /wget-show.sh
WORKDIR /
EXPOSE 8000
ENTRYPOINT ["/wget-show.sh"]
```

Running …

```
#!/bin/bash

docker build -t kartoza/wget-show:latest -f Dockerfile-wget .
docker build -t kartoza/python-server .
docker volume create Storage
docker run --name="pyserve" -v Storage:/home -p 8000:8000 -dt kartoza/python-server
docker cp HelloWorld.txt pyserve:/home/
docker container ls
docker volume ls | grep Storage
docker run -ti --name="wget-show" --rm --link pyserve:pyserve kartoza/wget-show
```

Autobuilds on hub.docker.com

* Build when your base image changes

* Build when your code / configuration changes

docker-compose

Docker compose

https://docs.docker.com/v17.09/engine/userguide/eng-image/dockerfile_best-practices/#expose

```
pip install docker-
compose
```

```
Commands:
  build              Build or rebuild services
  bundle             Generate a Docker bundle from the Compose file
  config             Validate and view the Compose file
  create             Create services
  down               Stop and remove containers, networks, images, and volumes
  events             Receive real time events from containers
  exec               Execute a command in a running container
  help               Get help on a command
  images             List images
  kill               Kill containers
  logs               View output from containers
  pause              Pause services
  port               Print the public port for a port binding
  ps                 List containers
  pull               Pull service images
  push               Push service images
  restart            Restart services
  rm                 Remove stopped containers
  run                Run a one-off command
  scale              Set number of containers for a service
  start              Start services
  stop               Stop services
  top                Display the running processes
  unpause            Unpause services
  up                 Create and start containers
  version            Show the Docker-Compose version information
```

Docker compose example

https://docs.docker.com/compose/gettingstarted/#step-2-create-a-dockerfile

Rancher Demo

KARTOZA

OPEN SOURCE GEOSPATIAL SOLUTIONS

Docker Helpers

https://github.com/kartoza/docker-helpers

Command quick reference

- **dcall** - Commit all named containers. Provide a prefix for the image name. The committed image will be in the form `<namespace>/<container name>`.
- **dcalldated** - Like the above command but adds a date stamp as the tag. The committed image will be in the form `<namespace>/<container name>:01-Septmber-2014`.
- **deall** - Export all named containers. Exported containers will be named in the form `docker-export-<container name>.tar`.
- **dealldated** - Export all named containers. Exported containers will be named in the form `docker-export:<namespace>:<container name>:<date>.tar`.
- **dipall** - List the name and IP address of all named containers. Output will be in the form `<container name> : IP Address`.
- **dli** - List all images. Shorthand for `docker images`.
- **dnames** - List all the names of named containers.
- **dpasswords** - Print the names and passwords from all named containers. When you set up your container, print any passwords to stdout so they show up in the docker logs command. Be sure to print the word `password` in your stdout message as this is a simple grep operation.
- **dps** - List all running containers. Short hand for docker ps.
- **dpsa** - List all running containers. Short hand for docker ps -a.
- **drmc** - Remove all stopped containers.
- **drmi** - Remove all images that have no tags.
- **dstart** - Start all exited containers. Typically you want to do this after a host reboot.