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

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.

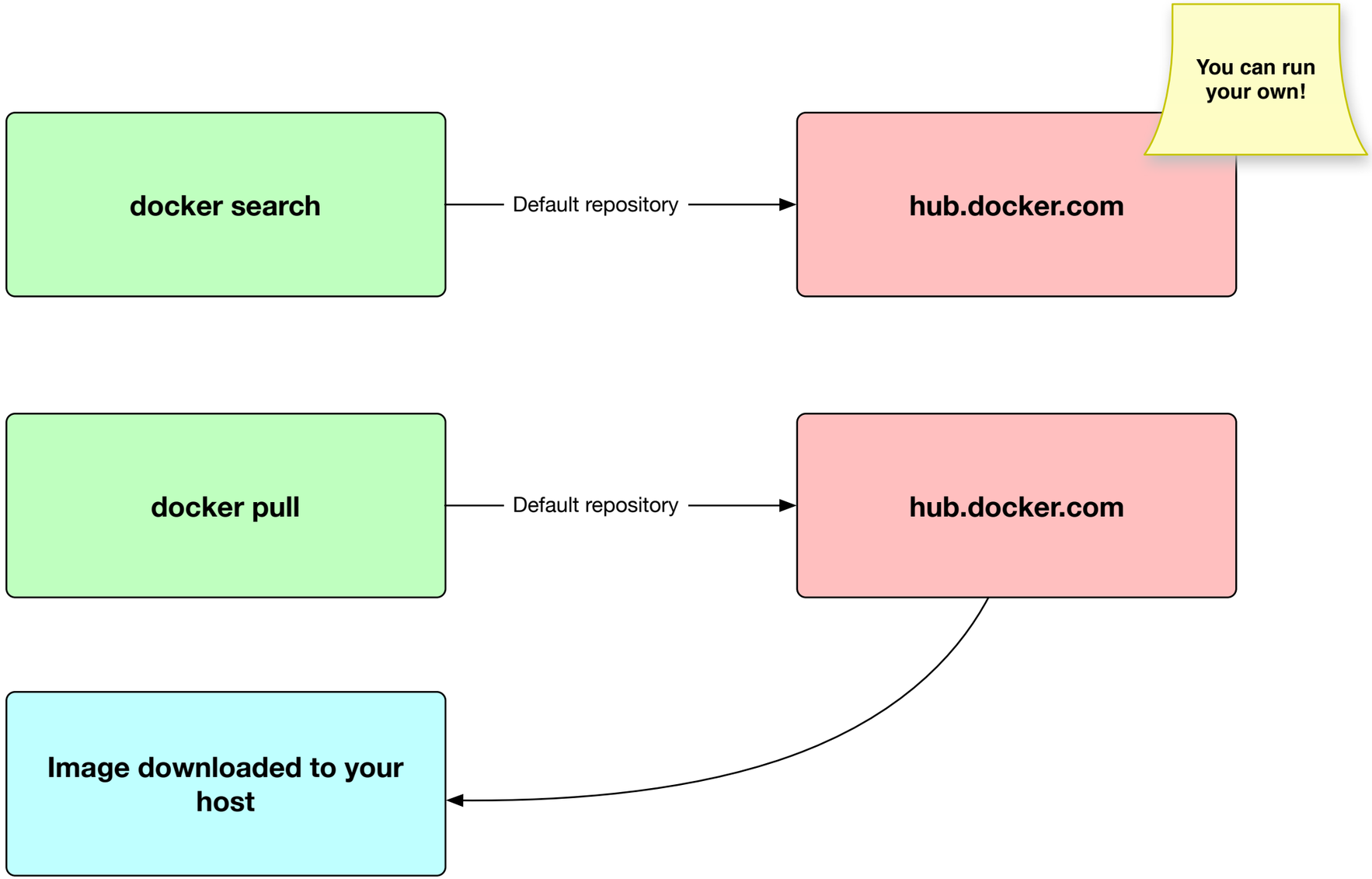
More info on images is available from the Docker docs and glossary.

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
```

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:

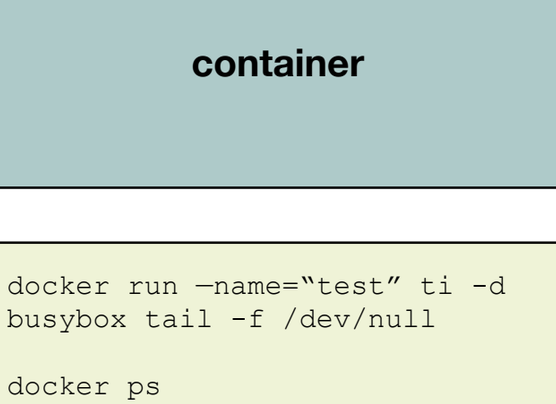
1. Like IMAGE ID, CONTAINER ID is the true identifier for the container. It has the same form, but it identifies a different kind of object.
2. `docker ps` only outputs *running* containers. You can view all containers (*running* or *stopped*) with `docker ps -a`.
3. NAMES can be used to identify a started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

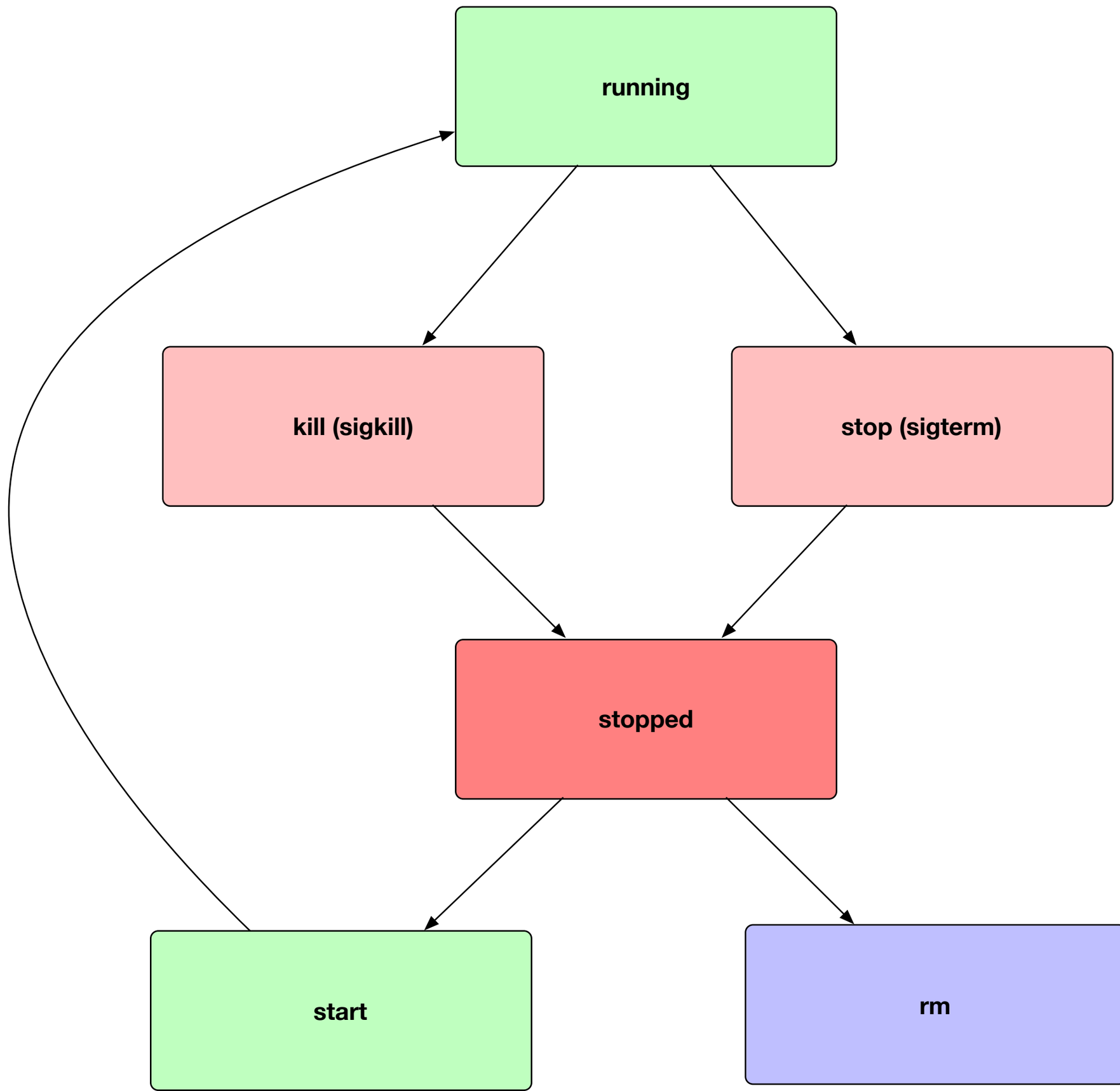
Source: https://stackoverflow.com/a/26960888

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
```





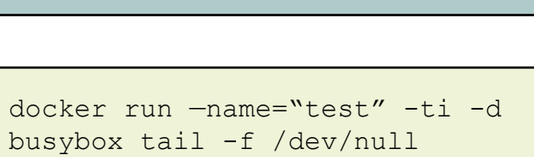
Container Management

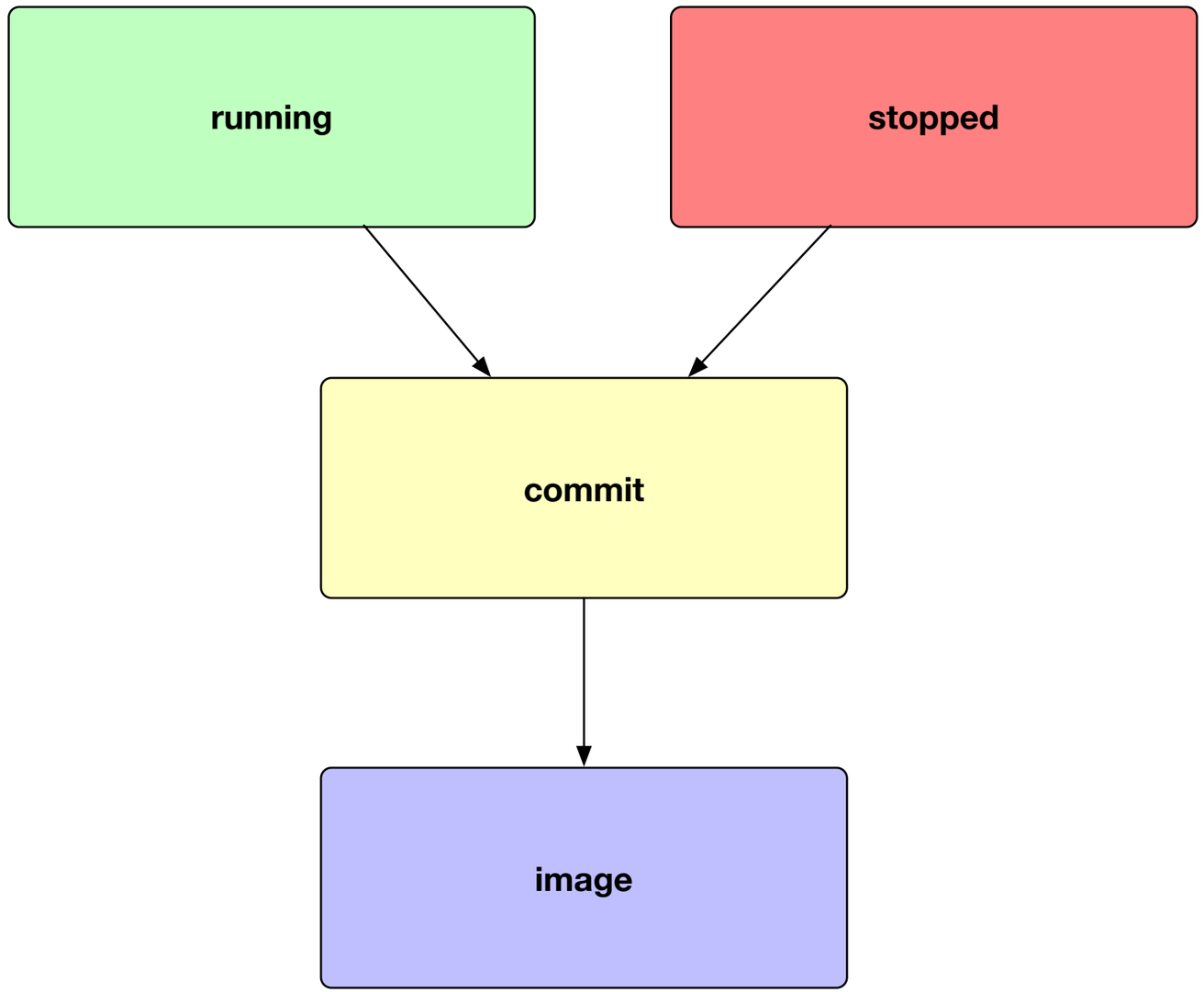
```
Usage:  docker container COMMAND

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
```

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
```

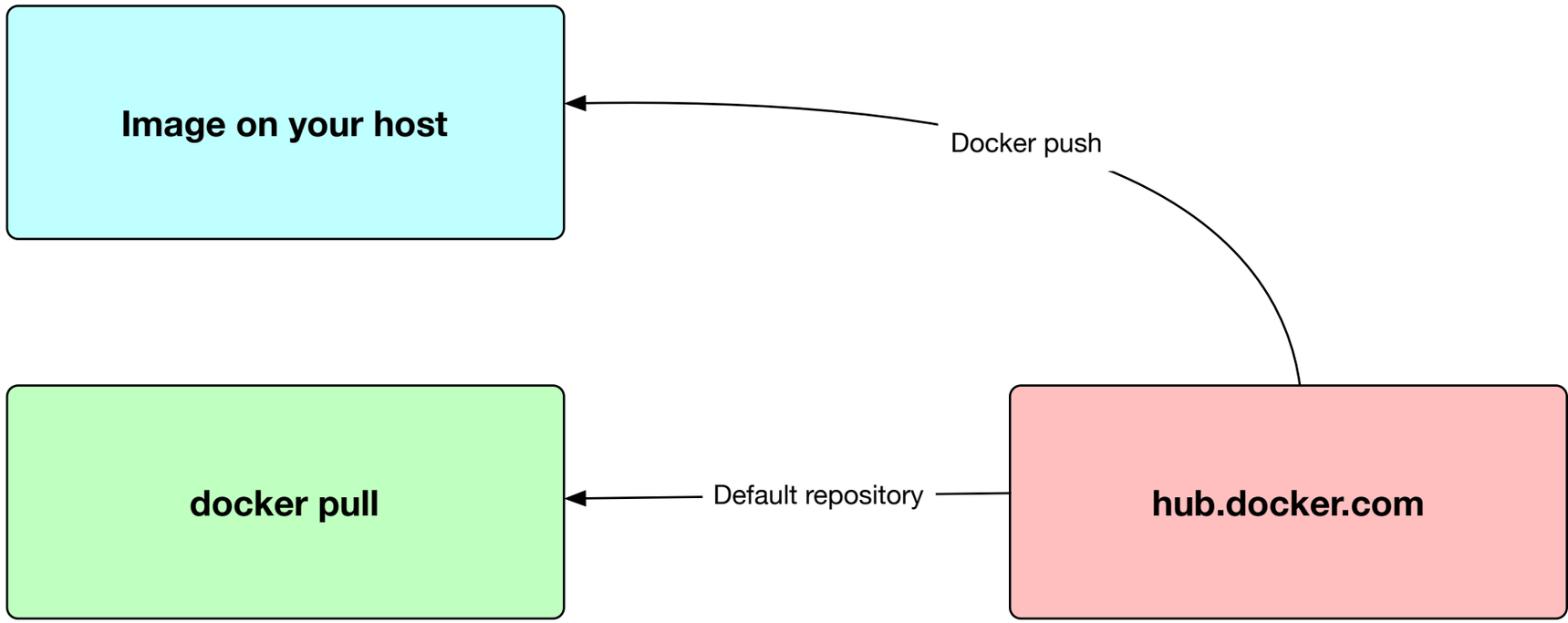




Publishing an image

Push

```
docker run –name=“test” -ti -d
busybox tail -f /dev/null

docker ps

docker commit test kartoza/
test:latest

docker images | grep
test
```



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"]
```

Host volumes

Note: -v will be replaced by —mount in the future see 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
```

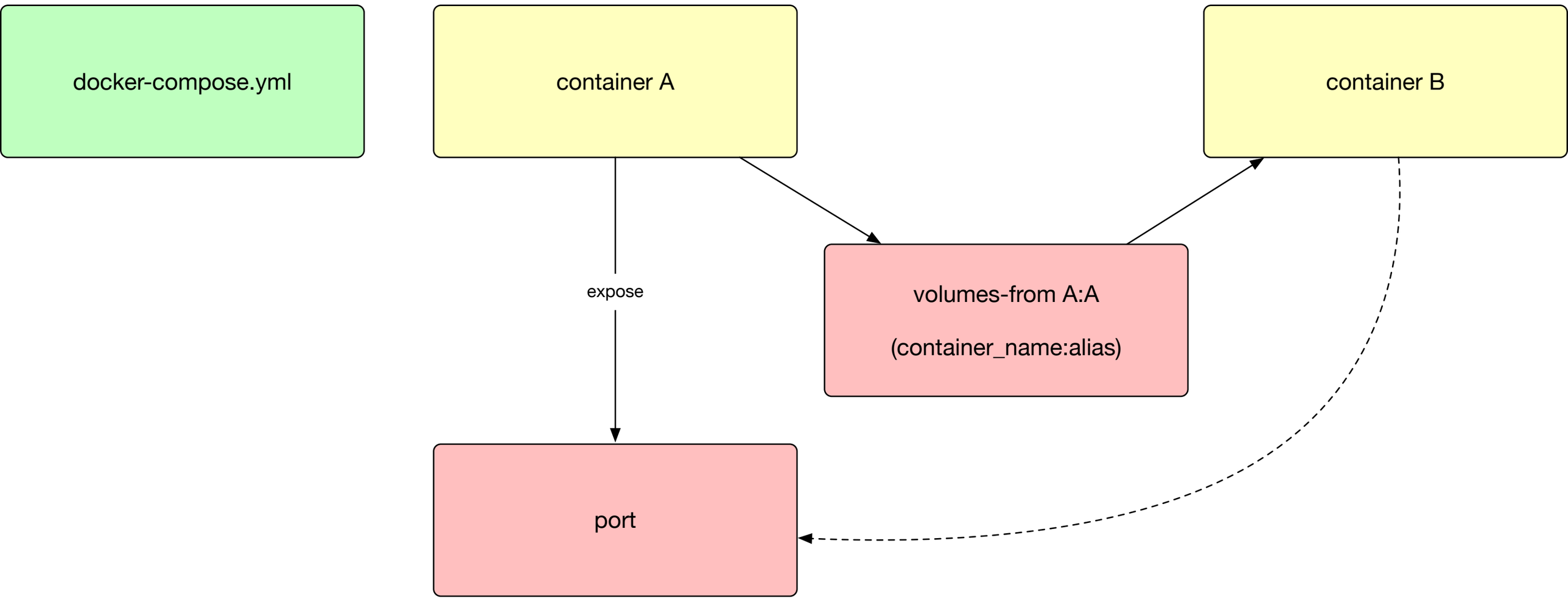
Directory listing for /

- .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

Storage Containers



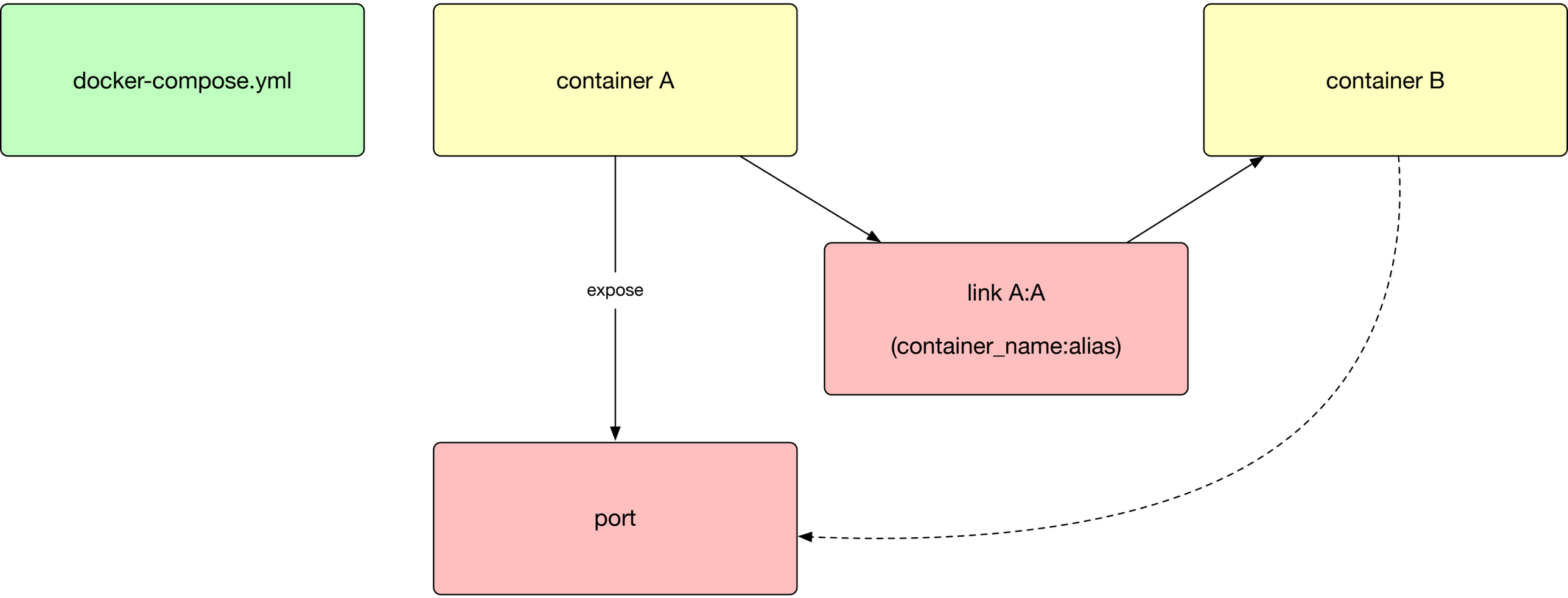
docker

KARTOZA
OPEN SOURCE GEOSPATIAL SOLUTIONS

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…



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.