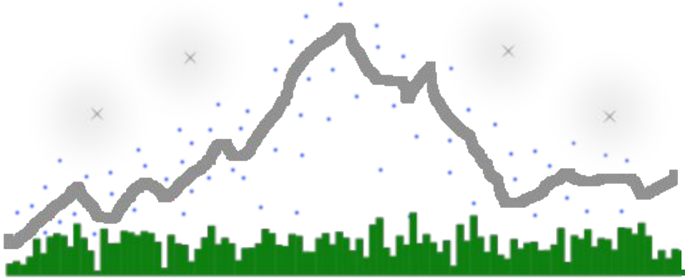
# Applications with R and Docker

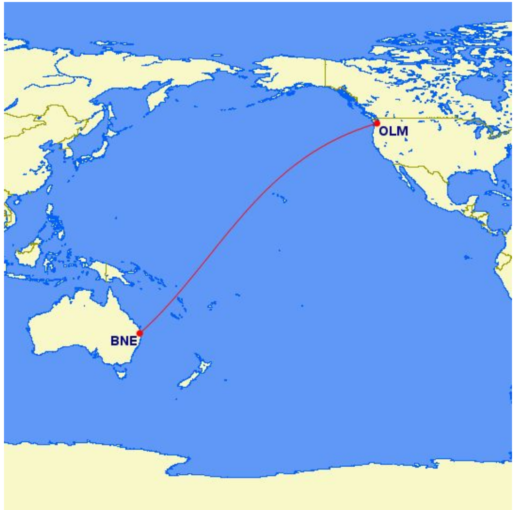
*useR! 2018 - Brisbane, Queensland, Australia - July 10, 2018*



Scott Came
Principal
Cascadia Analytics LLC

# Welcome - About Me

- Cascadia Analytics
- Data Science Interests:  Justice System, Elections, Social Media and Disinformation, Sport
- Using Docker for almost 3 years
- My hometown:



- Olympia, Washington, USA
- 11,753 great circle km from Brisbane
- State Capital
- Population: 280,588 (Thurston County, 2017)
- 1 hr S of Seattle
- 2 hr N of Portland
- 1 hr E of Pacific Coast

# Welcome - About you

Please tell us:

- Who you are
- Where you’re from
- Either:
    - A cool thing you’ve accomplished with Docker, or
    - One thing you’d like to know how to do in three hours that you aren’t comfortable doing now
    - (Or both!)

# Tutorial Plan

- Docker Basics
- R on Docker and the rocker project
- Docker Architecture
- Docker Networking and Storage
- Scaling applications with Docker Swarm
- Designing multi-container applications
- Using R and Docker together for reproducibility
- Open lab

# Takeaways

- Understand Docker components and how they work together
- Understand “the Docker way”
- See how to add backend database and frontend authentication to Shiny with Docker
- Explore useful tools like Play With Docker and AWS
- Explore Docker runtimes to take advantage of hardware like GPUs
- Have basic building blocks you can use on your own applications

# Environment Setup Preliminaries

- Docker for Mac / Windows / Linux
  - Nothing to do!
  - Verify setup: `docker --version`
  - For cli options: `docker --help`
  - Recommend stopping existing containers

- Docker Toolbox
  - VirtualBox (should be installed already)
  - Recommend creating a fresh Docker machine for today

```
docker-machine create --driver virtualbox useR-vm
docker-machine env useR-vm     << (and follow directions)
```

- Also might want to install jq: https://stedolan.github.io/jq/

## Exercise:

1. What version of Docker are you running?
2. How much memory is available to the Docker daemon?
3. How many images are in your local registry?

# Alternative: PWD

Play with Docker

A simple, interactive and fun playground to learn Docker

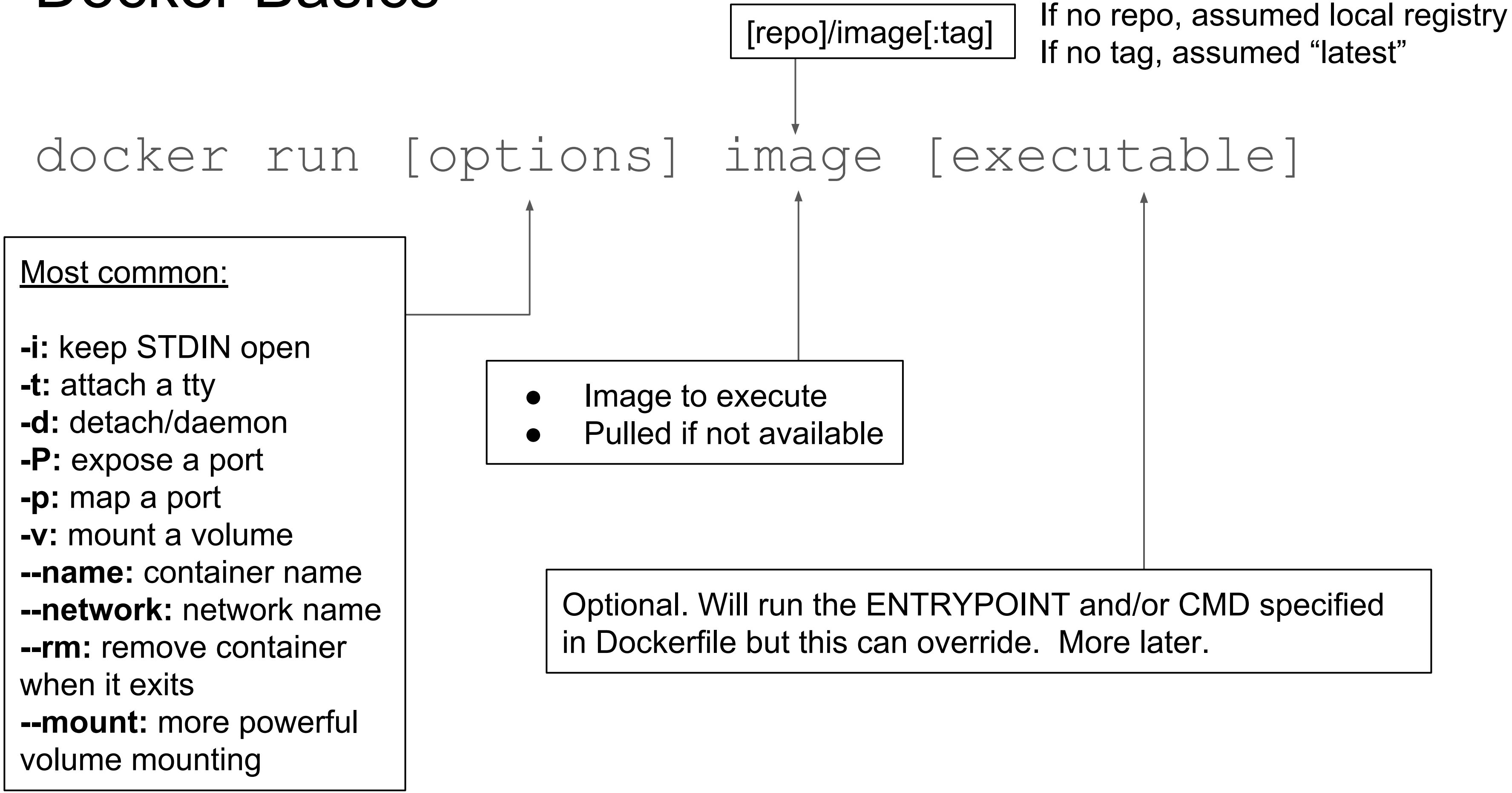
https://labs.play-with-docker.com/

- Magic of “dind” (Docker-in-Docker)
- DockerCon17 demo and discussion: https://dockr.ly/2yLyfpH
- If you go this route:
    - Note ssh link at top of instance page
    - Public IP of the running instance is same as ssh address, just replace ‘@’ with ‘.’
- Goes without saying:
    - Not secure
    - Data are not persisted
    - Sessions only last 4 hours
- Still really cool! 🤯

# Docker Basics

- Docker is a tool for running Linux processes in an isolated or “sandbox” environment
- Process and its context is defined in an **image**
- An instance of an image is a **container**
- Images inherit **FROM** a parent image (ultimately, `scratch`)
- Images reside in registries
    - You can **build** images locally (based on code in a **Dockerfile**) and store them in your local registry
    - You can **push** images to remote registries (including DockerHub)
    - You can **pull** images from remote registries (especially DockerHub)
- You can **run** a container from an image (if the container does not exist in the local environment, Docker will automatically **pull** it for you)

# Docker Basics



[repo]/image[:tag]

If no repo, assumed local registry
If no tag, assumed “latest”

```
docker run [options] image [executable]
```

Most common:

**-i:** keep STDIN open
**-t:** attach a tty
**-d:** detach/daemon
**-P:** expose a port
**-p:** map a port
**-v:** mount a volume
**--name:** container name
**--network:** network name
**--rm:** remove container when it exits
**--mount:** more powerful volume mounting

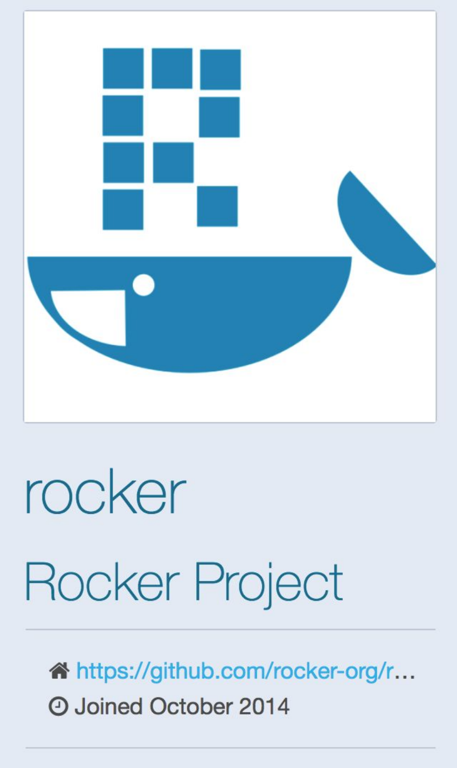
- Image to execute
- Pulled if not available

Optional. Will run the ENTRYPOINT and/or CMD specified in Dockerfile but this can override.  More later.

# Interactive vs detached

- Pass options `-it` to docker run for containers that will result in an interactive shell or application (like R console)
- Pass option `-d` to docker to run “detached” or in “daemon” mode; this is for networked services to which clients/browsers will connect (like RStudio server or Shiny server)
- You can also run networked services with the -t option.  Logs will be streamed to stdout, but the service will die when you kill the tty

# R Images



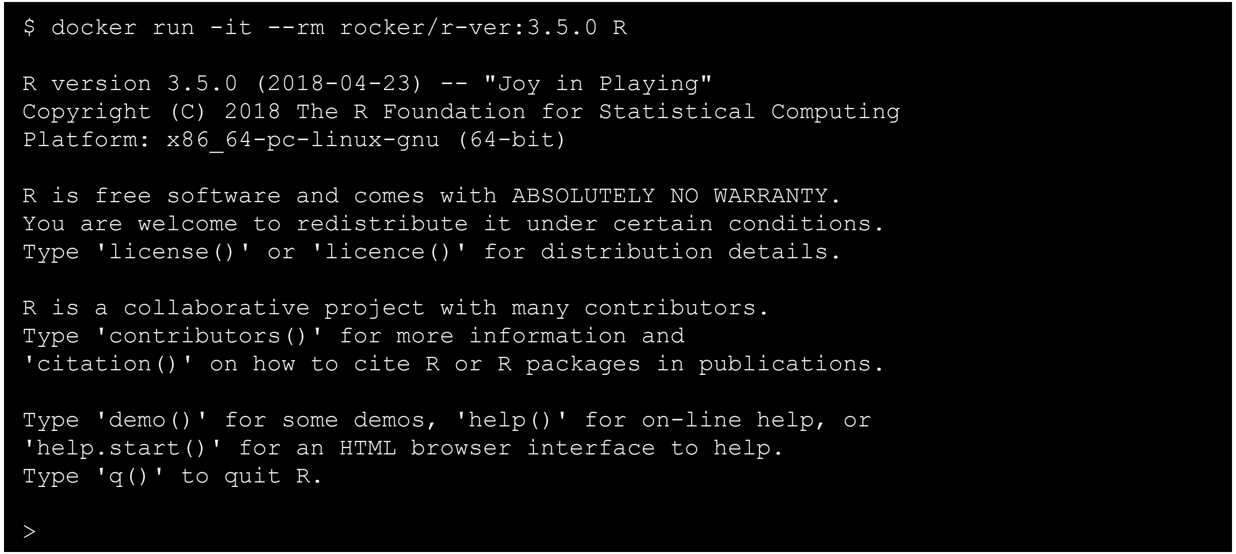
rocker

Rocker Project

https://github.com/rocker-org/r...

Joined October 2014

- For R environments / applications, consider the rocker project images the default
- https://hub.docker.com/u/rocker/
- https://github.com/rocker-org/rocker



```
$ docker run -it --rm rocker/r-ver:3.5.0 R

R version 3.5.0 (2018-04-23) -- "Joy in Playing"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

# Docker Basics - Exercises

## 🤸 Exercises:

1. Bring up an interactive R 3.5.0 prompt using Docker
2. Bonus: bring up an interactive R version 1.1.1 prompt (hint: `docker search scottcame` and look for something **old**).
3. Using the same image from #1, bring up a Linux interactive shell. What Linux distribution are we running?
4. Bring up RStudio Server in Docker and access the application via your browser. Did you choose a port?

Note: for #1, #3, and #4, just use Rocker project images for now.

# Other useful commands

- `ps`: list running containers (pass option -a to see non-running ones too)
- `inspect [container]`: info about a container
- `stats`: real-time info on container resource usage
- `images`: list all the images in the Docker environment
- `diff [container]`: what has changed in a container since it started
- `container prune`: remove all stopped containers

## Caution!

Be careful with `docker container prune` and `docker run … --rm …`

Any state in the container filesystem goes away when it's removed. Better yet: **stateless containers**!

# Modifying running containers

- `exec`: run another command in the container
- `cp`: copy host files to/from container

## Caution!

These commands are great when developing/testing an image.  But: If you do *either* in a production container, you are likely doing it wrong!

Containers should be immutable and their creation should be entirely defined by the Dockerfile.

Remember: containers are not virtual machines!

# So...are containers just little virtual machines?

How Containers are like VMs:

- Isolated Linux distro and filesystem
- Easy to package and replicate
- Portable across hosts
- Scale by deploying on more powerful hardware
- Run networked services on defined ports

Why Containers are **not** VMs:

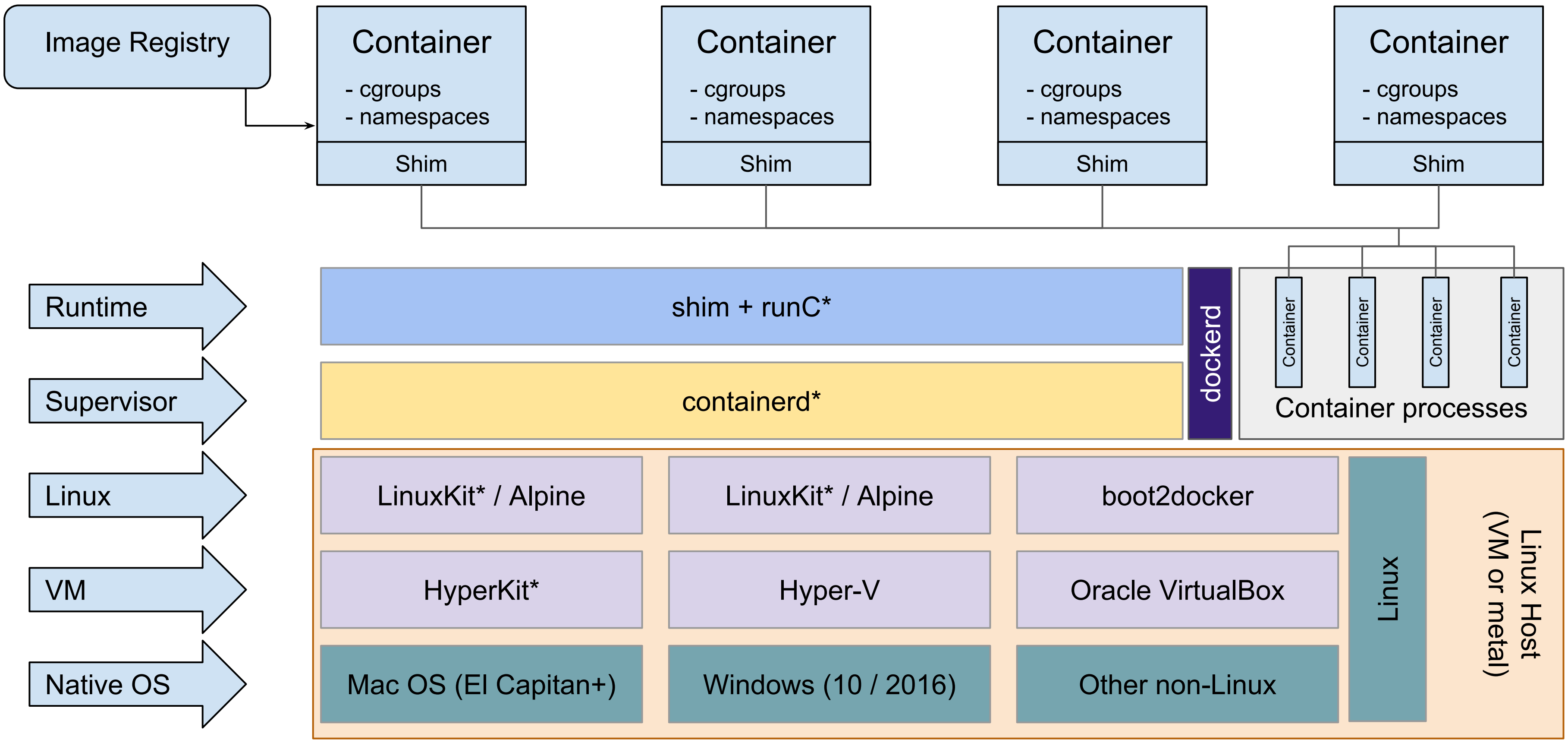
- Container isolation is an artifact of **namespaces**
- Everything is ultimately a process running in same Linux kernel
- No simulation of hardware
- Docker is not running any code...just setting up processes

Good explanation: https://blog.docker.com/2016/03/containers-are-not-vms/

Fun and instructive:

```
docker run -it --privileged --pid=host ubuntu nsenter -t 1 -m -u -n -i sh
```

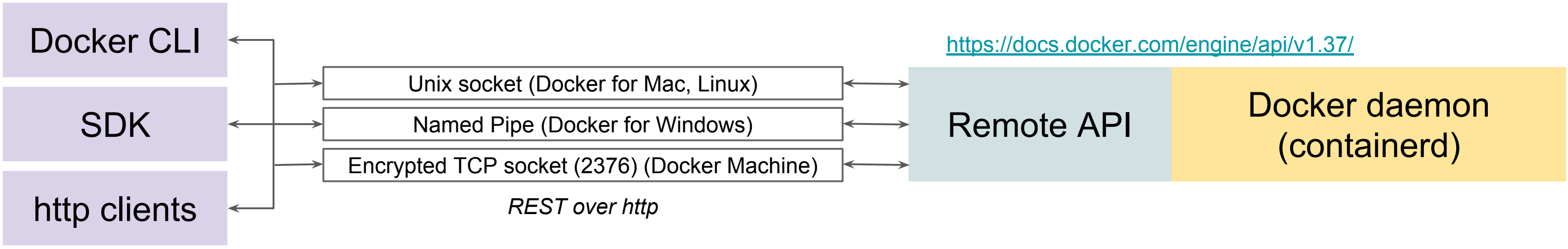
# The Docker Stack



* Moby project components

# Docker Remote API

https://docs.docker.com/engine/api/v1.37/


Docker CLI
SDK
http clients
Unix socket (Docker for Mac, Linux)
Named Pipe (Docker for Windows)
Encrypted TCP socket (2376) (Docker Machine)
REST over http
Remote API
Docker daemon (containerd)

## Docker for Mac:

```
$ curl --unix-socket /var/run/docker.sock http://localhost/info | jq .
```

## Docker Machine:

```
$ curl --insecure --cert ~/.docker/machine/machines/useR-vm/cert.pem --key \
> ~/.docker/machine/machines/useR-vm/key.pem https://$(docker-machine ip useR-vm):2376/info | jq .
```

## In-container:

```
$ docker run --rm -it \
> --mount "type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock" \
> ubuntu bash
$ apt-get update && apt-get install -y curl jq
$ curl --unix-socket /var/run/docker.sock http://localhost/info | jq .
```

# Using the Docker API from R

## Exercise:

In an interactive R container, use the Docker API to create a data frame (or tibble) with one row for each image in your local registry.

How many of your local images have ‘shiny’ in their tagged name?

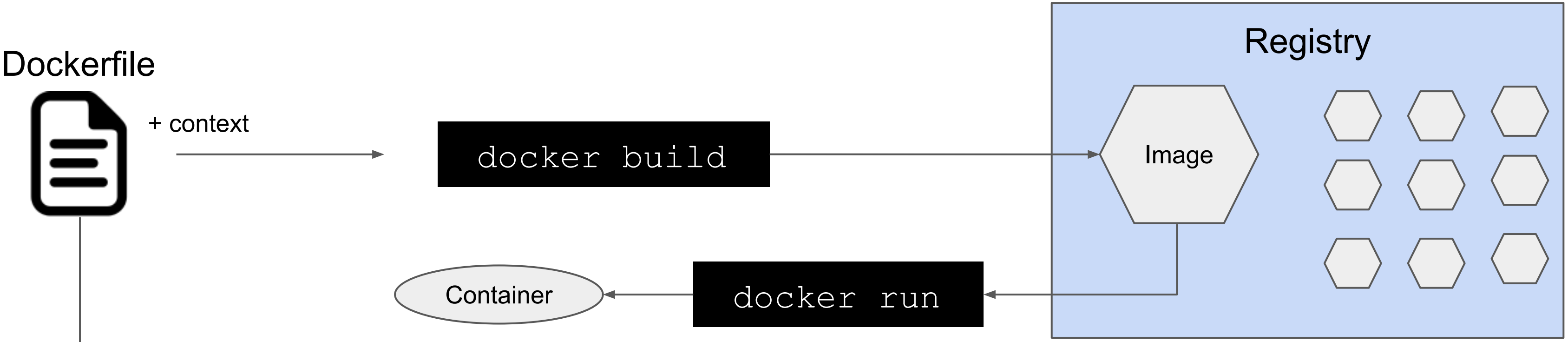
## Hints:

- The `rocker/tidyverse` image conveniently includes `jsonlite` and `httr`
- `httr` config option `unix_socket_path` connects to socket rather than normal http URL
- The `RepoTags` attribute contains all the image’s tag names, and this list column can be unnested with `tidyr::unnest()`

We will learn more about bind mounts shortly.  For now, just know that to mount a file from the Docker host into a container, use the `mount` option to `docker run` :

```
--mount "type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock"
```

# Populating Image Registries: Dockerfiles



- FROM: Specify base image
- LABEL: Metadata
- ADD/COPY: Set up container filesystem from context
- ENV: Set environment variables
- EXPOSE: Document ports
- VOLUME: Specify volume mount points
- ENTRYPOINT/CMD: Specify process command and args to run in container

# Dockerfile Examples

https://github.com/scottcame/shiny-docker-demo/blob/master/docker/tidyverse-mariadb/Dockerfile

https://github.com/rocker-org/rocker-versioned/blob/master/rstudio/3.5.0/Dockerfile

# Specifying what process to run

CMD versus ENTRYPOINT

- ENTRYPOINT defines the executable run within the process
- CMD defines arguments to the ENTRYPOINT executable, or executable if no ENTRYPOINT specified
- CMD is overridden by any args passed to docker run (at the end)
- ENTRYPOINT can be overridden by --entrypoint option to docker run
- Specify both as arrays (best practice)

# One Service?

- CMD / ENTRYPOINT defines the “actual” process of interest
- No child processes spawned by that process

# Or Many?

- CMD / ENTRYPOINT defines a “supervisor” that launches other services
- Options include supervisord, S6 overlay, or even sysvinit or systemd

Advantages:

- Modular, cleaner design
- Generally, better scalability with Docker compose and Swarm
- Greater granularity for leveraging cgroups to optimize cpu, memory, I/O, etc.

Advantages:

- No need for compose or other orchestration specs to deploy multi-process applications
- Easier migration from existing VM-based appliances

*Image source: https://pxhere.com/en/photo/161681 (Creative Commons CC0)*

# Common R Image Task: Installing Packages

## Exercise:

Build a Docker image that provides RStudio with the `lubridate` package pre-installed.

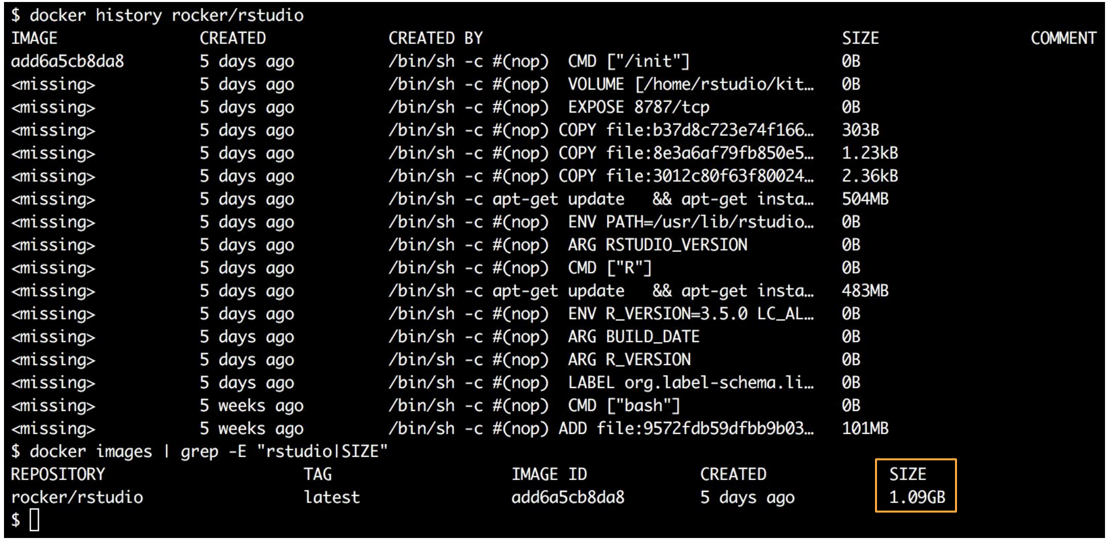
How much did this add to the size of the base RStudio image?

## Tip:

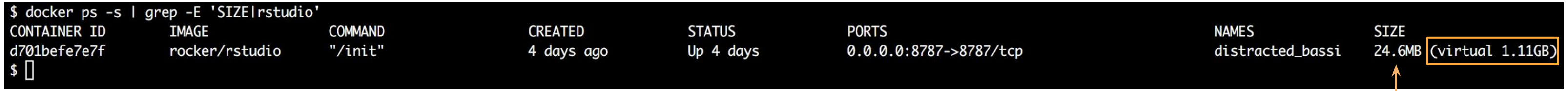
Build image in an interactive bash shell in the base image. Then use the `history` command to retrace your steps (and build your RUN instructions).

# Image Layers and the Cache

- `docker build` caches the results of each instruction
- The cache key is the text of the instruction, not the results
- Only one copy of each layer is stored in a registry
- Good ref: [https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/](https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/)



```
$ docker history rocker/rstudio
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
add6a5cb8da8        5 days ago          /bin/sh -c #(nop)  CMD ["/init"]                0B
<missing>           5 days ago          /bin/sh -c #(nop)  VOLUME [/home/rstudio/kit…   0B
<missing>           5 days ago          /bin/sh -c #(nop)  EXPOSE 8787/tcp              0B
<missing>           5 days ago          /bin/sh -c #(nop) COPY file:b37d8c723e74f166…   303B
<missing>           5 days ago          /bin/sh -c #(nop) COPY file:8e3a6af79fb850e5…   1.23kB
<missing>           5 days ago          /bin/sh -c #(nop) COPY file:3012c80f63f80024…   2.36kB
<missing>           5 days ago          /bin/sh -c apt-get update   && apt-get insta…   504MB
<missing>           5 days ago          /bin/sh -c #(nop)  ENV PATH=/usr/lib/rstudio…   0B
<missing>           5 days ago          /bin/sh -c #(nop)  ARG RSTUDIO_VERSION          0B
<missing>           5 days ago          /bin/sh -c #(nop)  CMD ["R"]                    0B
<missing>           5 days ago          /bin/sh -c apt-get update   && apt-get insta…   483MB
<missing>           5 days ago          /bin/sh -c #(nop)  ENV R_VERSION=3.5.0 LC_AL…   0B
<missing>           5 days ago          /bin/sh -c #(nop)  ARG BUILD_DATE               0B
<missing>           5 days ago          /bin/sh -c #(nop)  ARG R_VERSION                0B
<missing>           5 days ago          /bin/sh -c #(nop)  LABEL org.label-schema.li…   0B
<missing>           5 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           5 weeks ago         /bin/sh -c #(nop) ADD file:9572fdb59dfbb9b03…   101MB
$ docker images | grep -E "rstudio|SIZE"
REPOSITORY                      TAG                 IMAGE ID            CREATED             SIZE
rocker/rstudio                  latest              add6a5cb8da8        5 days ago          1.09GB
$ 
```



```
$ docker ps -s | grep -E 'SIZE|rstudio'
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES               SIZE
d701befe7e7f        rocker/rstudio      "/init"             4 days ago          Up 4 days           0.0.0.0:8787->8787/tcp   distracted_bassi    24.6MB (virtual 1.11GB)
$ 
```

*Each container has a container-specific layer on top of the image layers.  This is its size*

# Documenting your image

- LABEL instruction specifies metadata for an image
- Reported with `docker image inspect`
- Consider using label-schema: http://label-schema.org/rc1/
- Most common labels:
  - maintainer
  - description
  - name
  - vcs-url

```
FROM mariadb

LABEL maintainer="Scott Came (scottcame10@gmail.com)" \
  org.label-schema.description="Image with MariaDB ..." \
  org.label-schema.vcs-url="https://github.com/..."

ENV MYSQL_ALLOW_EMPTY_PASSWORD=yes

COPY files/* /docker-entrypoint-initdb.d/
```

```
$ docker image inspect demo-mariadb | jq .[0].Config.Labels
{
  "maintainer": "Scott Came (scottcame10@gmail.com)",
  "org.label-schema.description": "Image with MariaDB ...",
  "org.label-schema.vcs-url": "https://github.com/..."
}
```

# Pushing to DockerHub and Tagging

```
$ docker build -t [DH user]/[image name]:[tag]
```

*If you don't specify, tag = "latest"*

*Becomes relevant if you push*

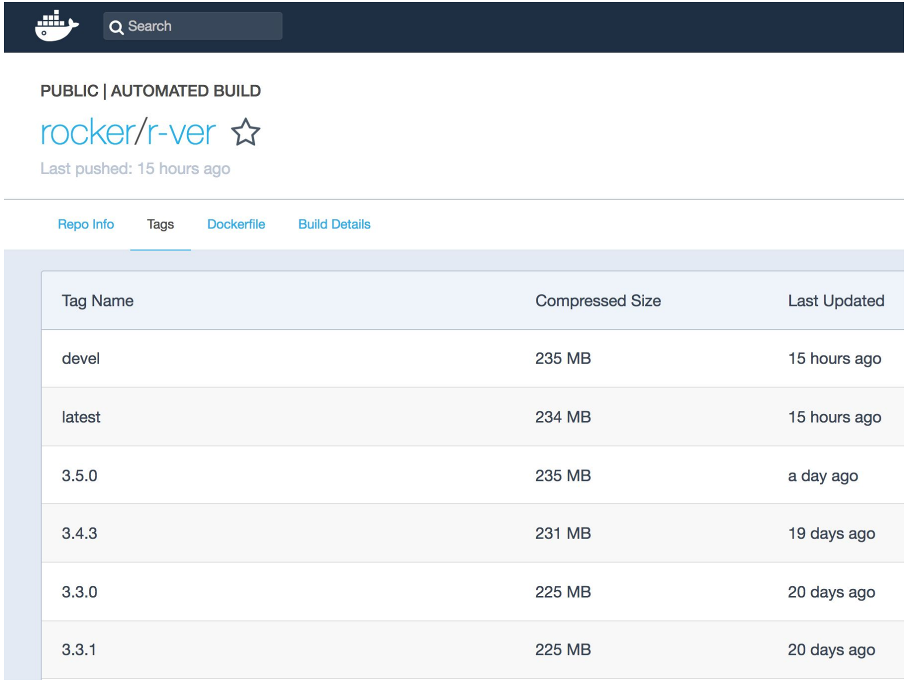
*To tag an existing image:*

```
$ docker tag [image name]:[tag] \
  [DH user]/[image name]:[tag]
```

*To make image available on DockerHub:*

```
$ docker push [DH user]/[image name]:[tag]
```

*Note: need account and prior* `docker login`



PUBLIC | AUTOMATED BUILD

rocker/r-ver

Last pushed: 15 hours ago

| Tag Name | Compressed Size | Last Updated |
|---|---|---|
| devel | 235 MB | 15 hours ago |
| latest | 234 MB | 15 hours ago |
| 3.5.0 | 235 MB | a day ago |
| 3.4.3 | 231 MB | 19 days ago |
| 3.3.0 | 225 MB | 20 days ago |
| 3.3.1 | 225 MB | 20 days ago |

# Docker Volumes

## Caution!

Data stored in a container’s writable layer (i.e., filesystem) is lost when the container is removed!!

Persistent data must be managed in *volumes.*

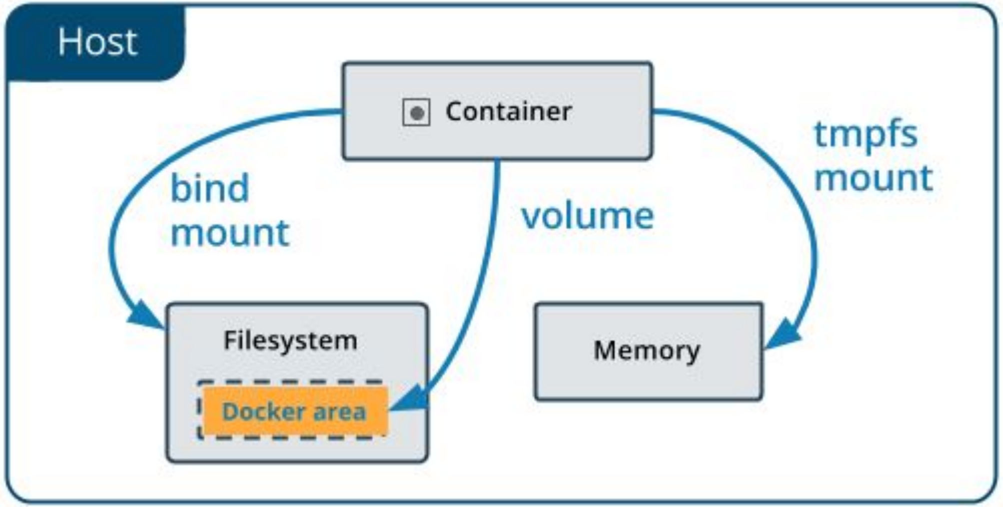
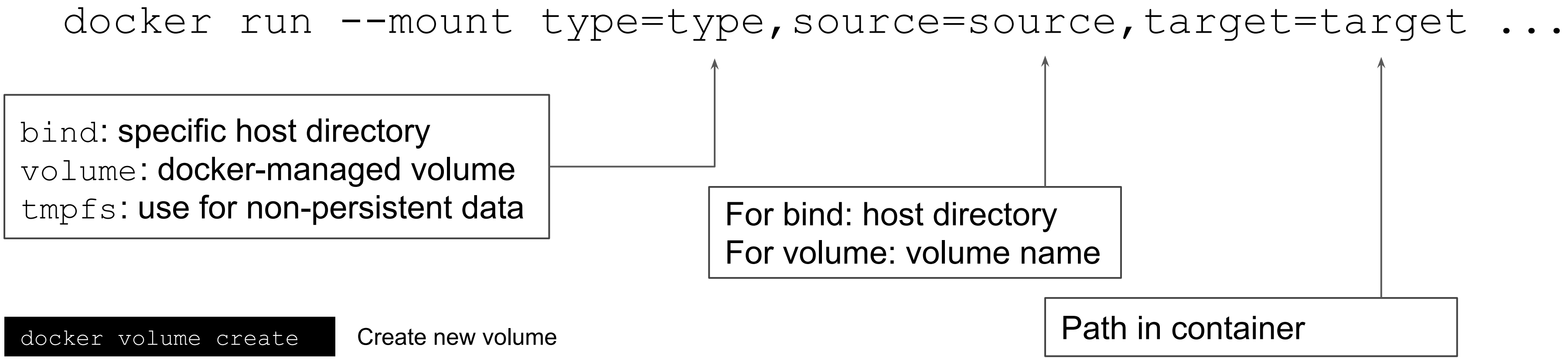


Image Source: docker.com

# Mounting Volumes

- Two options via `docker run` parameters: `-v` and `--mount`
- Consider favoring `--mount` (more powerful/flexible)
- Available host directories for bind mounts is a Docker daemon config



```
docker run --mount type=type,source=source,target=target ...
```

`bind`: specific host directory
`volume`: docker-managed volume
`tmpfs`: use for non-persistent data

For bind: host directory
For volume: volume name

Path in container

| | |
|---|---|
| `docker volume create` | Create new volume |
| `docker volume rm` | Delete named volume |
| `docker volume inspect` | Get volume info |

*Volumes are stored in host filesystem at* `/var/lib/docker/volumes/[name]`

# Mounting Volumes: Exercise

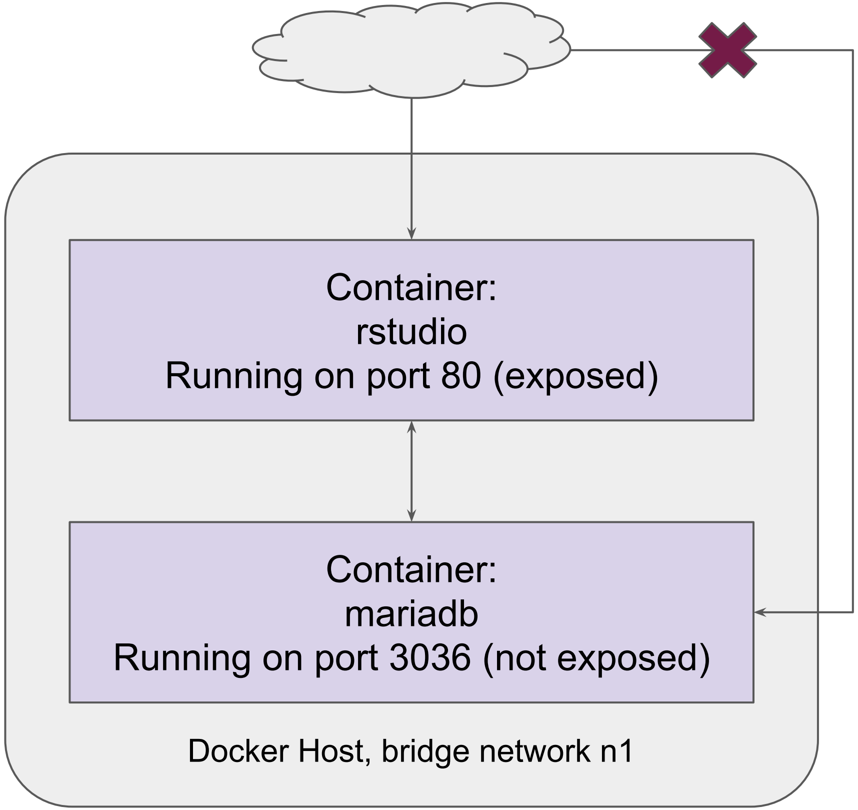
## Exercise:

Mount a bind volume into a base ubuntu image, create a file in the container directory, and exit the container. Do you see your file?

Now do the same with a volume mount. Do you see the file now? (Hint for Docker for Mac users: where is the “host”?)

# Docker Networking

- Outside of swarms, containers live in bridge networks
- Swarms use overlay networks to connect containers across hosts
- Containers can only communicate within their network(s)
- All ports exposed within network, but only explicitly exposed ports are available on the host IP (i.e., to the external network)
- Automatic DNS by container name (but only on user-created networks)



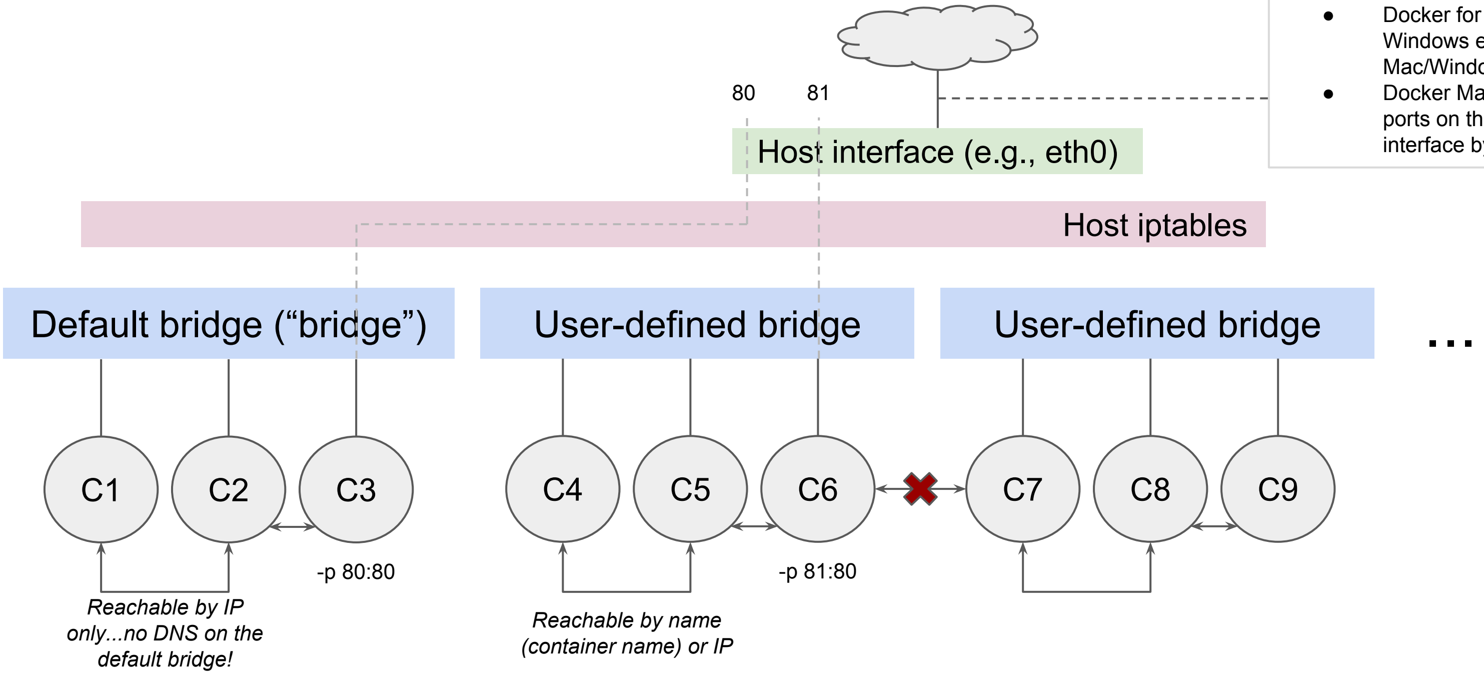
Container:
rstudio
Running on port 80 (exposed)

Container:
mariadb
Running on port 3036 (not exposed)

Docker Host, bridge network n1

```
$ docker network create n1
$ docker run … --network n1 …
```

# Bridge networks



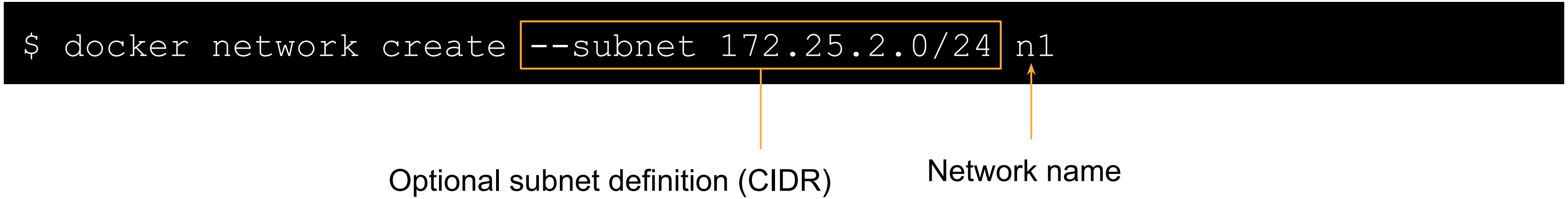
80
81
Host interface (e.g., eth0)
Host iptables
Default bridge ("bridge")
User-defined bridge
User-defined bridge
...
C1
C2
C3
C4
C5
C6
C7
C8
C9
-p 80:80
-p 81:80
*Reachable by IP only...no DNS on the default bridge!*
*Reachable by name (container name) or IP*

- Docker for Mac and Windows expose ports on Mac/Windows interfaces
- Docker Machine exposes ports on the Virtual Box VM interface by default

*This all happens through the magic of network namespaces in the Linux kernel (plus routing configuration in iptables)*

# Specifying a network

- If you run a container without specifying a network, the container will run on the (default) bridge network, named `bridge`
- To specify a network, first create one:



```
$ docker network create --subnet 172.25.2.0/24 n1
```

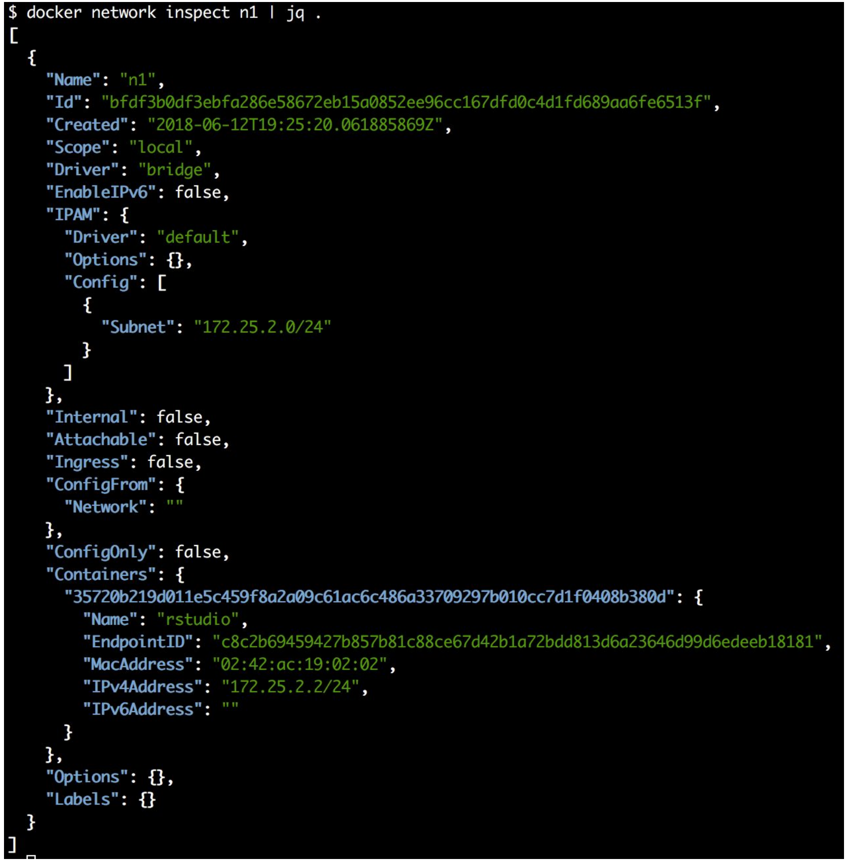
Optional subnet definition (CIDR)

Network name

- Then to run a container on a network, pass network name as --network option:

```
$ docker run -d --network n1 -p 8787:8787 --name rstudio rocker/rstudio
```

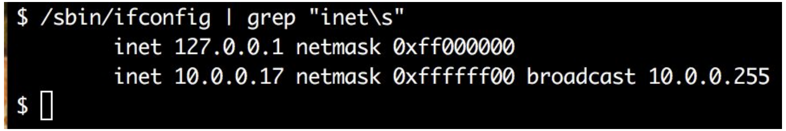
# Inspecting a network



```
$ docker network inspect n1 | jq .
[
  {
    "Name": "n1",
    "Id": "bfdf3b0df3ebfa286e58672eb15a0852ee96cc167dfd0c4d1fd689aa6fe6513f",
    "Created": "2018-06-12T19:25:20.061885869Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.25.2.0/24"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "35720b219d011e5c459f8a2a09c61ac6c486a33709297b010cc7d1f0408b380d": {
        "Name": "rstudio",
        "EndpointID": "c8c2b69459427b857b81c88ce67d42b1a72bdd813d6a23646d99d6edeeb18181",
        "MacAddress": "02:42:ac:19:02:02",
        "IPv4Address": "172.25.2.2/24",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]
```

# Choosing subnets

- Let Docker choose if possible
- Always choose private network
- Based on other networks to which the host belongs
- On Mac/Linux:



```
$ /sbin/ifconfig | grep "inet\s"
        inet 127.0.0.1 netmask 0xff000000
        inet 10.0.0.17 netmask 0xffffff00 broadcast 10.0.0.255
$ 
```

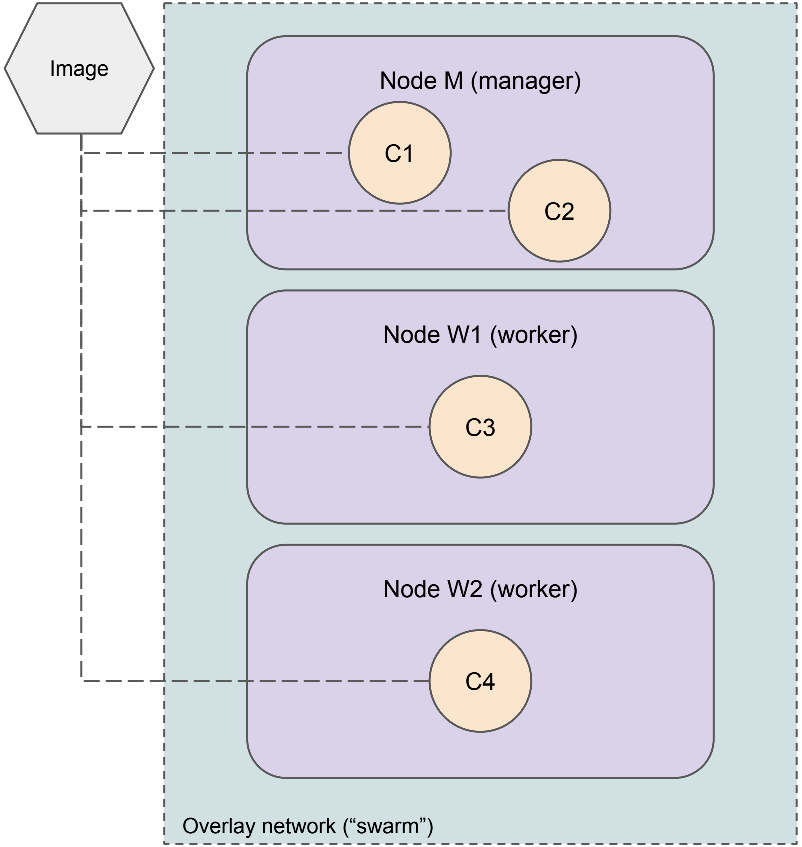
- Private networks:
  - A: 10.0.0.0/8 (16,777,216)
  - B: 172.16.0.0/12 (1,048,576)
  - C: 192.168.0.0/16 (65,536)
- Slash-number indicates number of mask bits (increase to create smaller subnets)

# Networking: Exercise

## Exercise:

1. Create a bridge network (feel free to specify a subnet if you want) named `nw-useR`.
2. Run a (detached) RStudio Server container on this network, exposing the standard RStudio Server port (8787) on host port 80. Use an RStudio Server image that has package RMariaDB pre-installed (hint: scottcame/tidyverse-mariadb). Name the container `rstudio`.
3. Run a (detached) MariaDB container on this network. Use the image scottcame/demo-mariadb. (Note that the root user password is blank in this database.) Name the container `demo-mariadb`.
4. Open RStudio (running in the server) in your browser, and load the contents of table t1A (in database demo1) into a data frame.
5. Bonus: Note that the scottcame/demo-mariadb image also contains all the mysql client tools. Can you demonstrate that a mysql client container, running on the default bridge network, cannot connect to the MariaDB server, but a mysql client container running on the `nw-useR` network can?

# Swarm mode and scaling containers



- Swarm mode creates a cluster of Docker hosts
- A swarm has 1..* manager nodes and 0..* worker nodes
- Nodes can be (and generally are) separate physical or virtual machines (hosts)
- Nodes are connected by a Docker overlay network (secure virtual LAN)
- Any exposed container ports are available on any node, and traffic is load balanced automatically
- Collection of load-balanced containers is called a “service”
- Guaranteed availability of n instances

```
$ docker swarm init
```

Init swarm, current engine becomes manager

```
$ docker swarm join ...
```

Join a node to the swarm

```
$ docker service create ... \
  --replicas 4
```

Create a service:

- Options mostly the same as `docker run`
- Specify number of instances with `--replicas`

# Swarm exercise

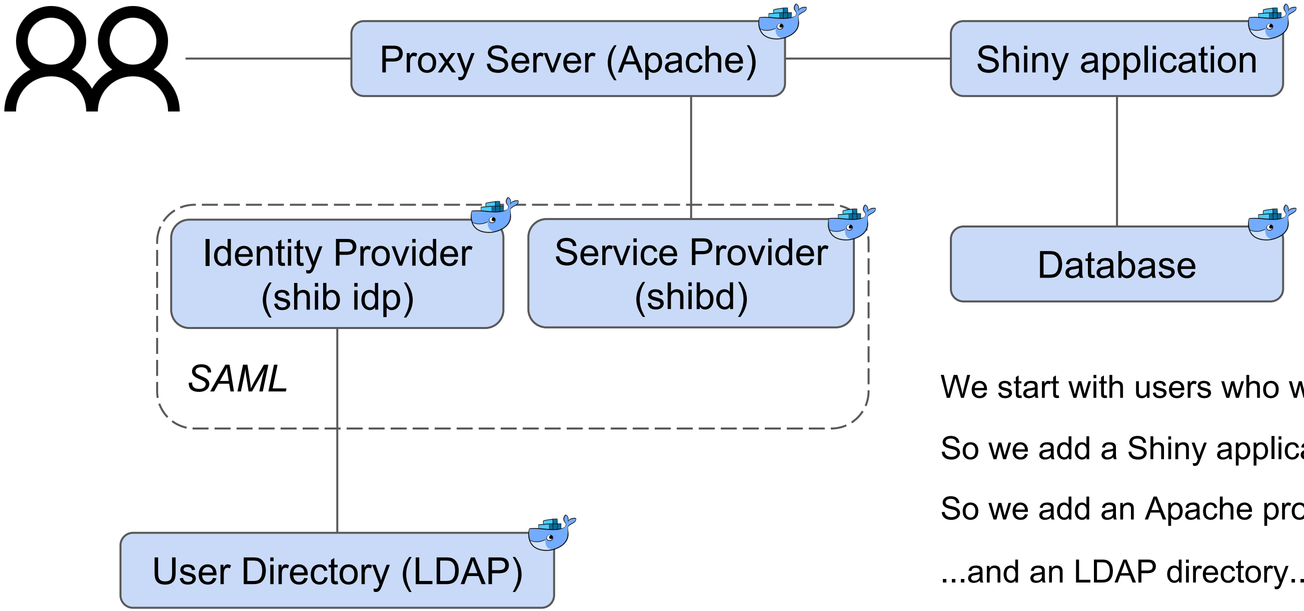
## Exercise:

1. Option 1: clone (or fork+clone) https://github.com/scottcame/shiny-docker-demo and build the docker image in the docker/shiny directory
2. Option 2: pull the docker image scottcame/shiny from DockerHub

Then:

1. Run a container from this image, exposing port 3838 (the standard Shiny port), and bring up the hostinfo Shiny app (should be at http://[localhost or ip]:3838/hostinfo)
2. Init a one-node Swarm in your current Docker engine
3. Now run the container as a service in Swarm mode, with 3 replicas (or scale the one you just created). What do you see when you refresh the page over and over?

# Multi-container applications



We start with users who want to analyze data in a database.

So we add a Shiny application. But what about security?

So we add an Apache proxy server...

...and an LDAP directory...

...and some SAML infrastructure to handle federated login.

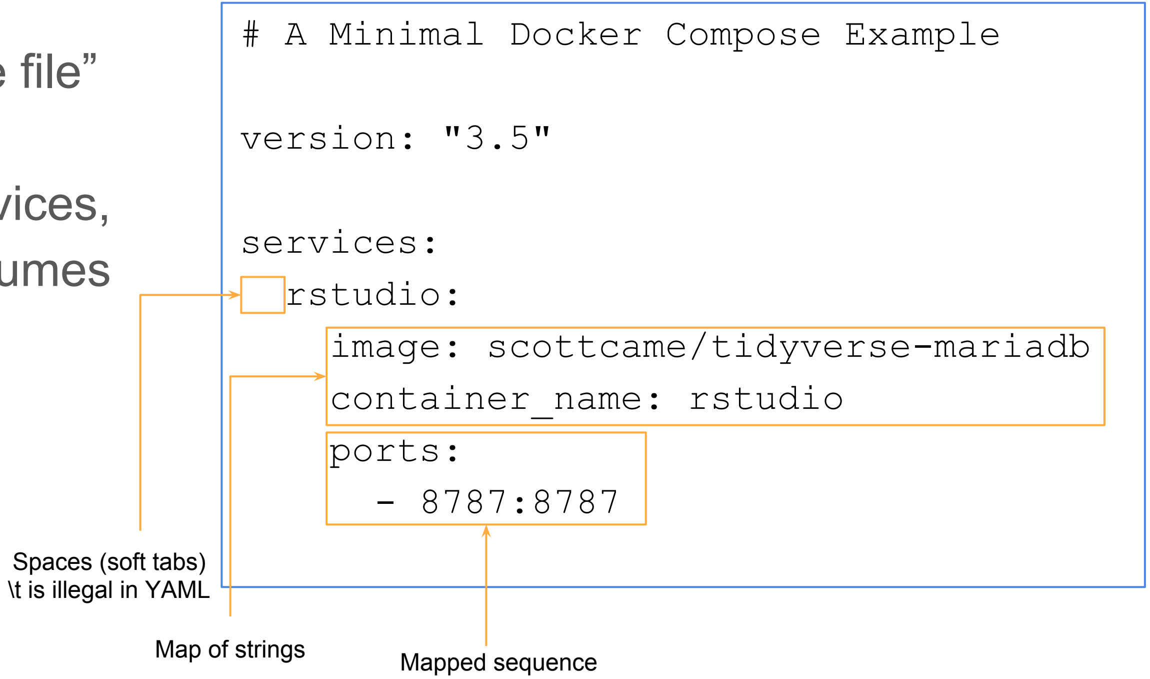
All as Docker containers!  But how do we make sure we have all the pieces, without manually starting all these containers?

# Docker Compose

```
$ docker-compose -f cf.yaml up -d
$ docker-compose -f cf.yaml down
```

< Start services in background
< Stop and remove services

- Define and run multi-container applications
- Controlled by a “compose file” written in YAML
- Compose file defines services, and the networks and volumes that they use
- Can build images before instantiating containers
- Compose files can be hierarchical



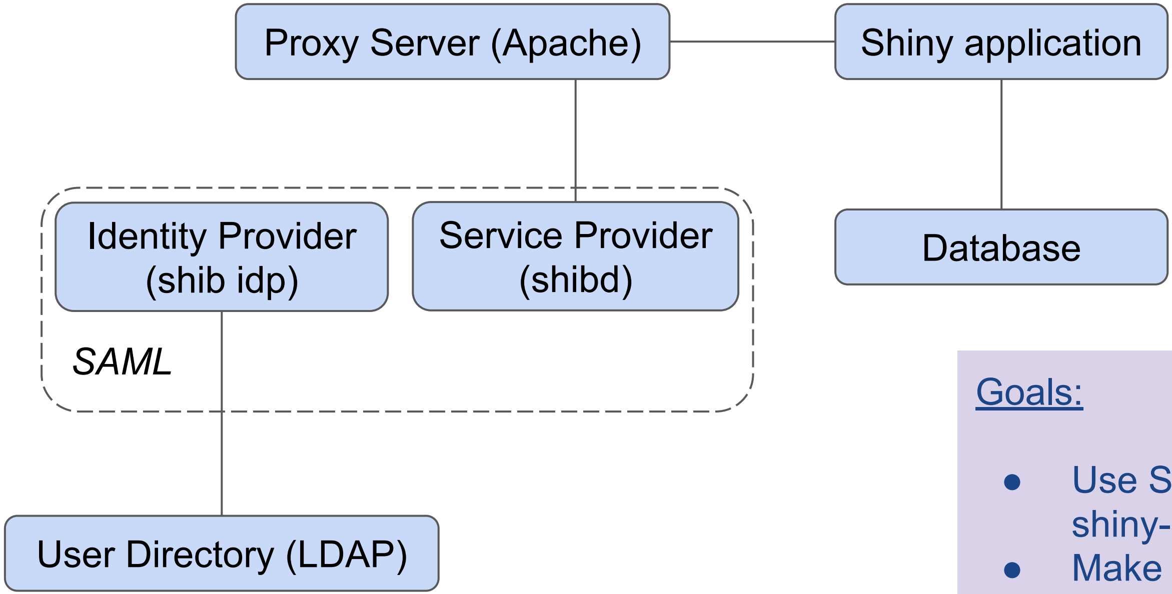
```
# A Minimal Docker Compose Example

version: "3.5"

services:
  rstudio:
    image: scottcame/tidyverse-mariadb
    container_name: rstudio
    ports:
      - 8787:8787
```

# Docker Compose: Exercise

## Exercise:

1. Remove the network `nw-useR` and containers `rstudio` and `demo-mariadb`.
2. Using the Compose File Reference (https://docs.docker.com/compose/compose-file/) to help, replicate the prior exercise using Docker Compose.
3. This time, bind-mount a volume into the RStudio Container into which you can save some results of your RStudio session after the container dies.
4. Take down the compose application.
5. Do you see your session output stored in the volume source?

# Securing Shiny Apps with Docker

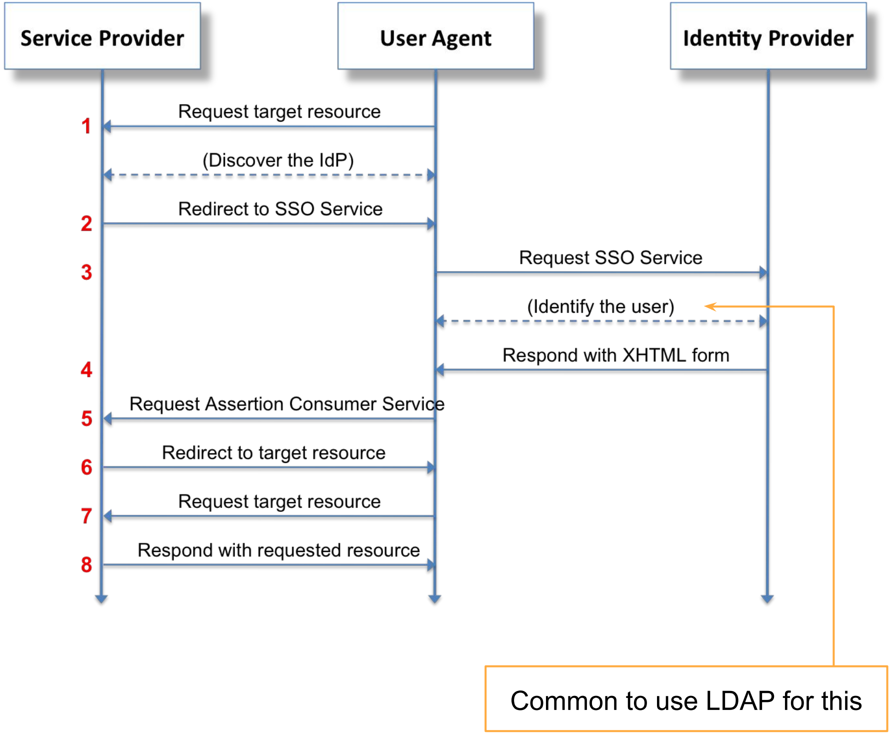


Goals:

- Use SAML to secure access to shiny-server
- Make authenticated user information available to Shiny apps via session parameter to server function

# A Diversion into SAML

- Security Assertion Markup Language
- Used for sharing assertions (claims) about users between relying parties and identity providers.  Claims are mostly about:
  - Authentication (this user demonstrated her identity via this mechanism at this date/time)
  - Attributes (the user who authenticated has this set of known attributes/facts)
- OASIS standard (stable at version 2.0 since 2005)



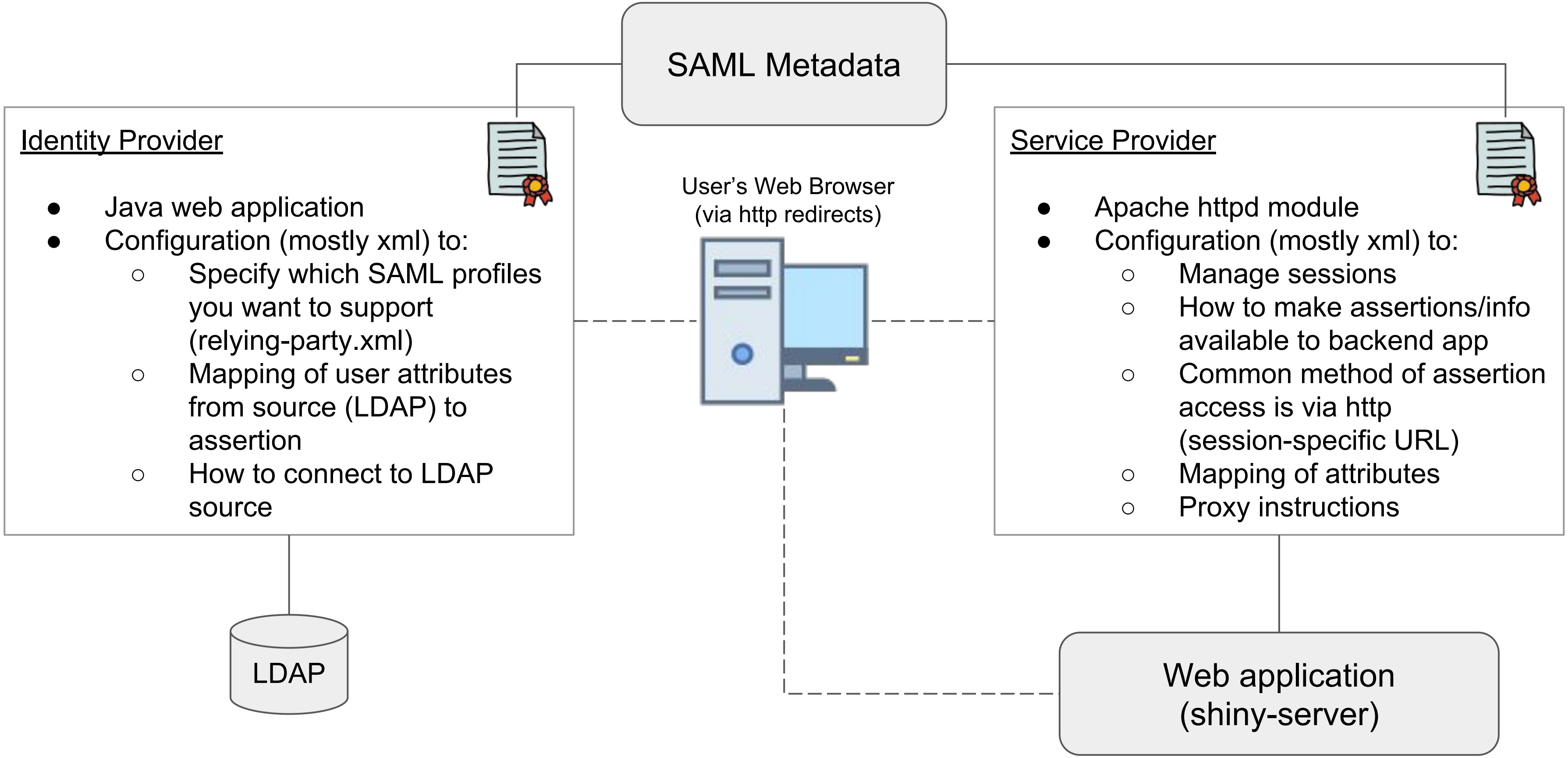
Common to use LDAP for this

*Image source: https://en.wikipedia.org/wiki/File:Saml2-browser-sso-redirect-post.png*

# A SAML Assertion

https://github.com/scottcame/shiny-docker-demo/blob/master/shiny-app/shiny/example-assertion.xml

# Shibboleth:  Open Source SAML Implementation



Identity Provider

- Java web application
- Configuration (mostly xml) to:
    - Specify which SAML profiles you want to support (relying-party.xml)
    - Mapping of user attributes from source (LDAP) to assertion
    - How to connect to LDAP source

Service Provider

- Apache httpd module
- Configuration (mostly xml) to:
    - Manage sessions
    - How to make assertions/info available to backend app
    - Common method of assertion access is via http (session-specific URL)
    - Mapping of attributes
    - Proxy instructions

# Multi-container shiny app in action!

## Exercise:

Use docker-compose to run the complete application. See if you can navigate the image source to determine the login username (all passwords are "password").

Spend a few minutes exploring the compose file and image source, and also explore the running containers.

Note: you can clone my shiny-docker-demo repo from github and build all the images, or just grab the compose file at https://github.com/scottcame/shiny-docker-demo/blob/master/docker/docker-compose.yaml.

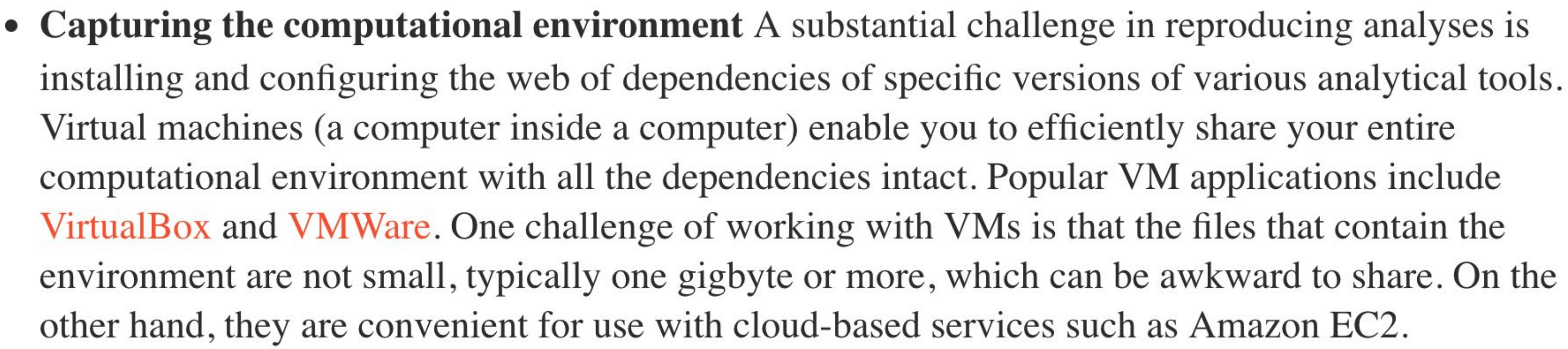
Warning: the shiny-apache-shib-sp image takes **forever** to build.

# Scaling multi-container applications

- Scaling compose applications is straightforward...in theory
  - `docker stack deploy` is mostly equivalent to `docker-compose up -d`
  - In compose file, each service can have a `deploy` section to control Swarm deployment
  - The deploy section takes, among other options, a `replicas` option
- Unfortunately Shiny doesn’t like something about this setup
- Sticky sessions

# Docker and Reproducibility

From http://ropensci.github.io/reproducibility-guide/sections/introduction/ :



- **Capturing the computational environment** A substantial challenge in reproducing analyses is installing and configuring the web of dependencies of specific versions of various analytical tools. Virtual machines (a computer inside a computer) enable you to efficiently share your entire computational environment with all the dependencies intact. Popular VM applications include VirtualBox and VMWare. One challenge of working with VMs is that the files that contain the environment are not small, typically one gigbyte or more, which can be awkward to share. On the other hand, they are convenient for use with cloud-based services such as Amazon EC2.

# Docker Advantages for Reproducibility

- Smaller footprint
- Easier deployment
- Easier sharing and publication
- Open source platform
- Standard scripting of image setup with Dockerfile
- Rocker images as baseline

# Reproducibility components

## Context

- R version
- R packages
- Database software
- Etc.

- Defined in Dockerfile
- Definition managed in source control (e.g., GitHub)
- Stored in registry (e.g., DockerHub)
- Versioned via tags

## Data

- CSV/json/XML
- Serialized data frame
- API endpoint

*Options*

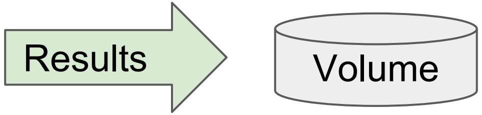
- Access from source**
- Build into image
- Store in version control (e.g., GitHub*)

## Pipeline

- R scripts
- Markdown/notebooks
- Shell scripts, etc.

*Options*

- Manage in version control (e.g., GitHub*) and pull at runtime
- Build into image



* Cite commit hash if possible
** Consider publishing SHA256 hash

# Simple example

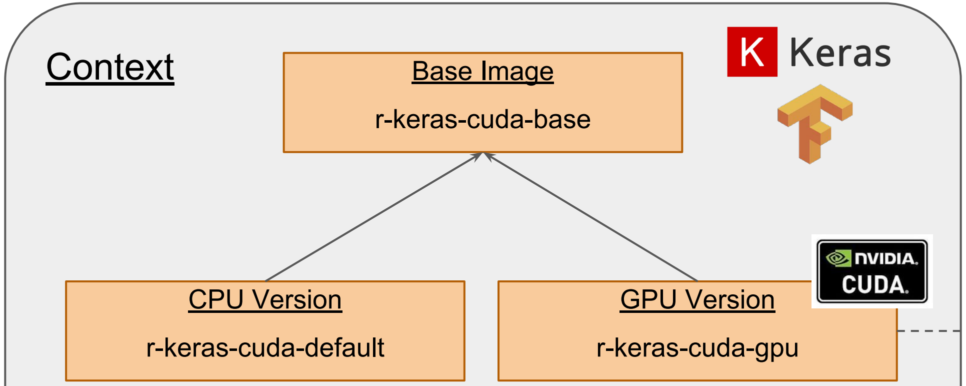
## Context

```
FROM rocker/geospatial:3.5.0

VOLUME /output

RUN apt-get update && apt-get install -y curl
RUN R -e 'install.packages(c("ggthemes"))'
RUN cd /tmp && \
 curl -O https://raw.githubusercontent.com/scottcame/shiny-docker-demo/master/australia-elex-2016/Notebook.Rmd

CMD ["R", "-e", "rmarkdown::render('/tmp/Notebook.Rmd', output_file='/output/Notebook.html')"]
```

Data: https://data.world/scottcame/australian-federal-election-2016

Pipeline: https://github.com/scottcame/shiny-docker-demo/blob/master/australia-elex-2016/Notebook.Rmd

# Machine learning example



Context

Base Image

r-keras-cuda-base

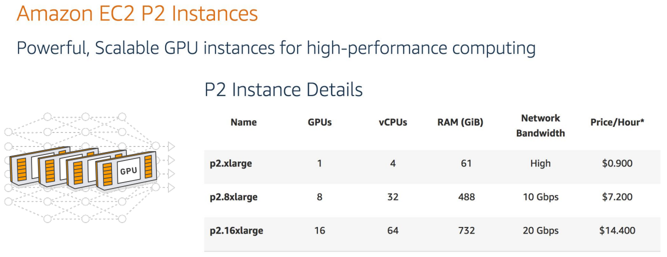
CPU Version

r-keras-cuda-default

GPU Version

r-keras-cuda-gpu

Images in my docker github repo: https://github.com/scottcame/docker
(and on DockerHub, of course)



Amazon EC2 P2 Instances

Powerful, Scalable GPU instances for high-performance computing

P2 Instance Details

| Name | GPUs | vCPUs | RAM (GiB) | Network Bandwidth | Price/Hour* |
|---|---|---|---|---|---|
| p2.xlarge | 1 | 4 | 61 | High | $0.900 |
| p2.8xlarge | 8 | 32 | 488 | 10 Gbps | $7.200 |
| p2.16xlarge | 16 | 64 | 732 | 20 Gbps | $14.400 |

https://github.com/scottcame/docker/wiki/AWS-EC2-ML-instance-setup

Data and Pipeline: https://github.com/rstudio/keras/tree/master/vignettes/examples

# Open Lab

We can:

- Follow up on anything from the tutorial that you'd like to explore in more depth
- Talk about other uses of Docker
- Hear about how you plan to use Docker after the conference
- Anything else on your mind!

My contact information:

scott@cascadia-analytics.com
+1 360-529-2938 (US)
@scottcame on Twitter