novuhealth

Using Docker For Development

Aaron Bartell &&

Andrew Rich &&

John Knutson

2019-08-26

So you're telling me that Docker solves all my problems?
No, this is a secret message telling you to go to the doctor across the street who is in-network and will save you $500.
Maybe they'll appreciate my earrings matching my blouse.

Love what you do again.

Code at Novu.

novu.com/join-us

Benefits

Highly competitive salary

100% health/dental premiums covered

401K with no cap on employer match

Yearly funds for training and conferences

A goal of keeping meetings to a minimum

A focus on quality

Fun and collaborative work environment

© 2019 NovuHealth, LLC. All rights reserved.

Agenda

- **Part I**
  - Why Docker
  - Day in the life of a Rails dev
  - What’s different, what’s the same
- **Part II: Demo**
- **Part III**
  - `Dockerfile` vs `docker-compose.yml`
  - Containers and Images
  - Unresolved issues

The Why

docker + RAILS

Consistency. Makes it easy for devs to use same versions of tech (i.e. Ruby, Nginx, Postgres, Node.js, Linux).

Production Proximity. Dev is done on a Mac and the code is run on Linux.

Throwaway-ness. Sometimes starting over is quicker than trying to fix. Now it almost always is.

Remedy half-life of knowledge. Bake in the Node.js requirements of your Rails app so everyone doesn’t have to learn how to maintain every tech.

Fast by default. E.g. Include Postgres fine-tuning in the Docker files so it’s not a *post install activity* for all devs.

More apps, less config. Microservices? Write code that configures dependencies. No app is a snowflake.

Part I

A DAY IN THE LIFE

of a Rails developer using Docker

A day in the life...

Starts all services declared in `docker-compose.yml`.

Nginx, Postgres, and Rails.

Logs are color coordinated by service.

```
$ git clone blog && cd blog
$ docker-compose up
Pulling nginx (99999999.dkr.ecr.us-east-1.amazonaws.com/novu/nginx:0.4.1)...
Starting blog_nginx_1    ... done
Starting blog_postgres_1 ... done
Starting blog_app_1      ... done
Attaching to blog_nginx_1, blog_postgres_1, blog_app_1
nginx_1       | + SSL_KEY=/etc/nginx/ssl/server.key
nginx_1       | + SSL_CERT=/etc/nginx/ssl/server.cer
postgres_1    | listening on IPv4 address "0.0.0.0", port 5432
postgres_1    | listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
nginx_1       | nginx: [warn] the "ssl" directive is deprecated
postgres_1    | database system was shut down at 2019-08-13 19:40:08 UTC
postgres_1    | database system is ready to accept connections
app_1         | [6] Puma starting in cluster mode...
app_1         | [6] * Version 3.12.1 (ruby 2.6.2-p47), codename: Llamas in Pajamas
app_1         | [6] * Environment: development
app_1         | [6] * Process workers: 5
app_1         | [6] * Listening on unix:///opt/blog/socket/blog.sock
```

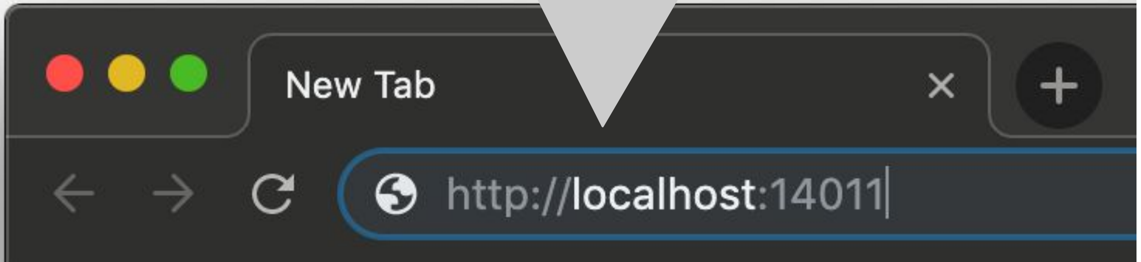
What’s running?

The `docker ps` command displays statistics of currently running containers.

```
$ docker ps
CONTAINER ID  IMAGE          COMMAND            STATUS         PORTS                                     NAMES
74b0e5ce7cf6  blog_app       "puma -C con.."    Up 15 minutes                                            blog_app_1
d65876a1628c  nginx:0.4.1    "entrypoint.sh"    Up 15 minutes  8080/tcp, localhost:14011->14011/tcp      blog_nginx_1
3461c9051b3e  postgres:11.1  "entrypoint.sh"    Up 15 minutes  localhost:54311->5432/tcp                 blog_postgres_1
```

Exposed port mappings from the host(Mac) to the Linux container.

Point browser to localhost and port.



Run commands `bundle/rails/rspec`

Obtain an interactive prompt in the `app` service's container.

```
$ docker-compose exec app bash
```

At this point you're inside a Linux container.

```
blog@74b0e5ce7cf6:/opt/blog$ uname -or
4.9.184-linuxkit GNU/Linux

blog@74b0e5ce7cf6:/opt/blog$ rails console
Running via Spring preloader in process 518
Loading development environment (Rails 5.2.3)
irb(main):001:0>
```

Only tools you've installed will exist in the container.

Containers are thin

Containers should only run a single process or group of same processes.

NOTE: ActiveJob processes would run in a separate container.

```
$ docker-compose exec app bash
blog@74b0e5ce7cf6:/opt/blog$ ps -Af
UID          PID   PPID  C STIME TTY          TIME CMD
blog           1      0  0 20:48 pts/0    00:00:00 puma 3.12.1 (un
blog           7      1  1 20:48 pts/0    00:00:00 puma: cluster w
blog           9      7 12 20:48 pts/0    00:00:04 puma: cluster w
blog          11      7 12 20:48 pts/0    00:00:04 puma: cluster w
blog          12      7 11 20:48 pts/0    00:00:04 puma: cluster w
blog          14      7 11 20:48 pts/0    00:00:04 puma: cluster w
blog          29      0  0 20:48 pts/1    00:00:00 bash
blog         125     29  0 20:49 pts/1    00:00:00 ps -Af
```

Notice Nginx and Postgres aren't seen in this container.

Our bash shell. Doesn't count :-)

Logs

docker-compose.yml

```
services:
  nginx:
    volumes:
      - ./log/nginx:/var/log/nginx
  app:
    volumes:
      - ./log/puma:/var/log/blog/puma
      - ./log/blog:/opt/blog/log
```

Map logs generated in the container to the host.

Use tail to watch them all or open individual files in your editor of choice.

```
$ tail -f log/nginx/blog_access.log \
       -f log/nginx/blog_error.log \
       -f log/blog/development.log \
       -f log/puma/stderr.log \
       -f log/puma/stdout.log
```

Pro tip: Save yer fingers! Create a `make` target.

```
$ make tail
```

Stopping containers

```
$ docker-compose up
Starting blog_nginx_1      ... done
Starting blog_postgres_1 ... done
Starting blog_app_1        ... done
Killing blog_app_1         ... done
Killing blog_nginx_1       ... done
Killing blog_postgres_1   ... done
^CGracefully stopping... (press Ctrl+C again to force)
$
```

Ctrl+C ends the containers.

Optionally start in daemon mode to free up your console.

```
$ docker-compose up -d
$ docker-compose stop
```

Use `docker-compose stop` to end all containers defined in `docker-compose.yml`.

How does my code get into a container?

The `docker-compose.yml` file is used to define services (containers).

The `volumes` section of a service maps your project's current directory into the container's `/opt/blog` directory.

Syncing is bi-directional below, though could also be one-way and read-only. Other syncing options (`delegated`, `cached`) improve I/O performance.

docker-compose.yml

```yaml
services:
  app:
    volumes:
      - ./:/opt/blog
```

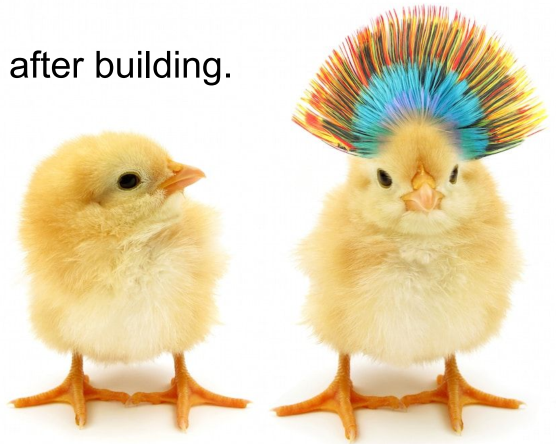
Different and Better

Different

- No longer run code on host, it runs in container.
- Enter into container to do many things (`rails console`, `rspec`, `bundle`)
- Learn to use `docker` commands to start/stop/manage containers.
- Apps use a network to talk to external services (database, cache)
- Apps don't run on localhost, and must use external networks to communicate
- Environment variables become the preferred method of app configuration

Better

- Run CI tests inside the docker image you build. Can store image after building.
- Use linux versions of binary gems (closer to production-like)
- No need for `rvm` or `rbenv` inside a container



What stays the same?

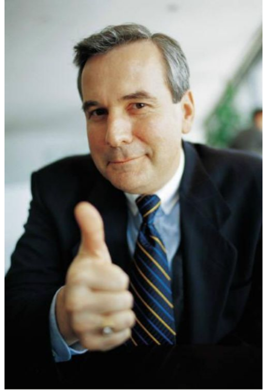
- **You keep your favorite editor and shell!**
- All Git things are still done on the host. Editor Git integrations work as expected.
- Bring app up in browser the same as before.
- You’re still writing ruby!

Running things inside container

`Makefile` targets are handy for frequented commands.

- `make bash app` - Run an interactive bash shell inside of the container. Once inside you are near the same experience as a normal shell in typical Rails development.
- `make binstall` - Run `bundle install` inside of the container.
- `make dbrefresh` - Download obfuscated/trimmed database and restore it inside container. Not run often but saves a lot of time.
- `make tail` - Runs `tail` on all log files (5+).
- `make cleanup` - Removes containers, networks, and volumes for an app.
- `make sane` - Checks environment health things; AWS authenticated, db migrations, etc. Can be unique to each app.
- `make install_vim` - We don't include convenience tools in images to keep our `Dockerfile` production ready. Aaron admission: I change this to be `joe`.
- Another option: `dip` gem by Evil Martians - https://github.com/bibendi/dip

Selling it to management



Portability. Take a docker image built in development or CI and host it anywhere that supports containers. Extend this to push button deploys/environments, and make autoscaling easy.

Cheap QA environments. You can probably use one host for more than one app or service. One of our QA EC2 instances now runs 40 docker tasks.

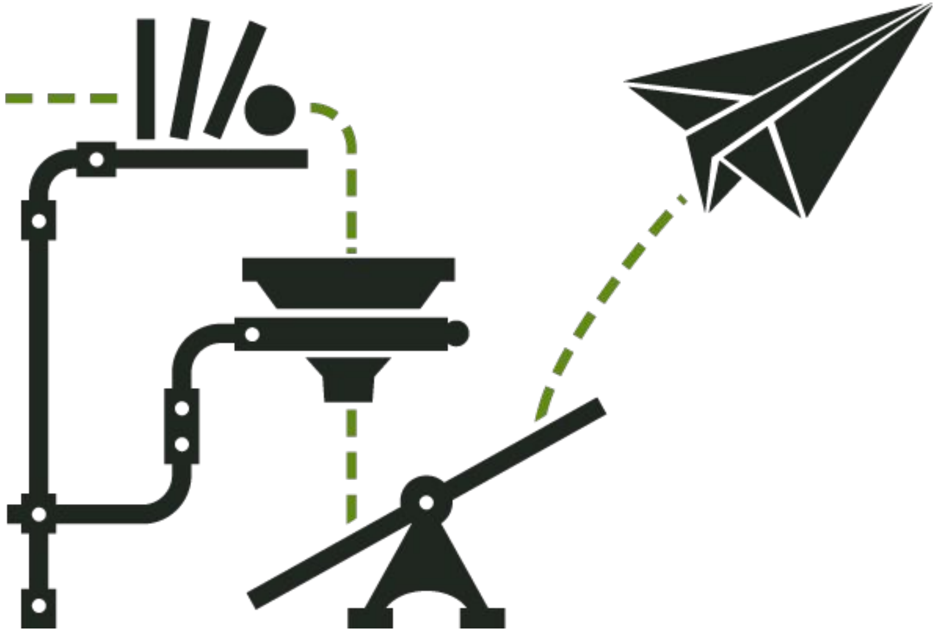
Easy upgrades. Upgrading Ruby, Postgresql, Nginx can be as easy as swapping out a higher versioned image.

Faster dev onboarding. Got a long README.md or document that onboards new devs? This should shorten it considerably.

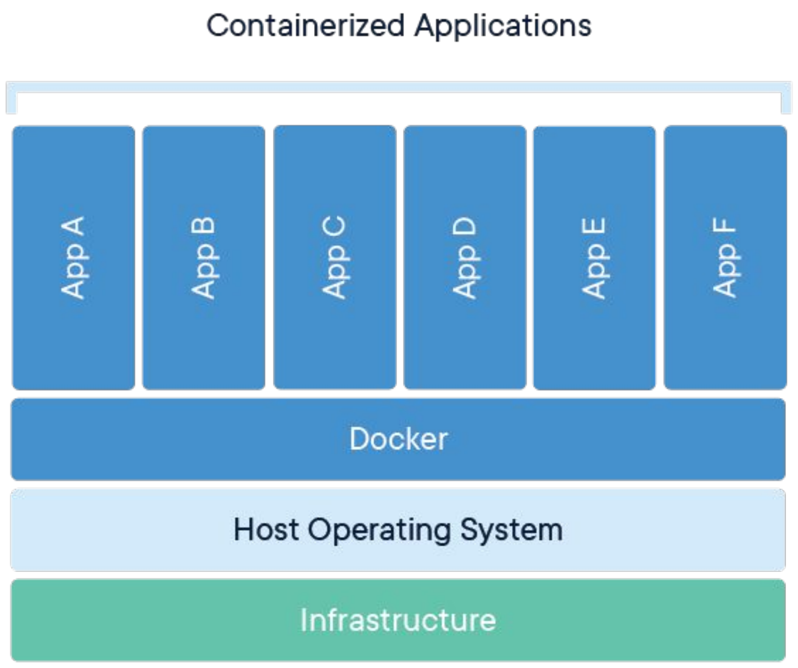
Security Scans. You can point a security tool like Twistlock at an image or a whole repo of docker images and stay ahead of the CVEs.

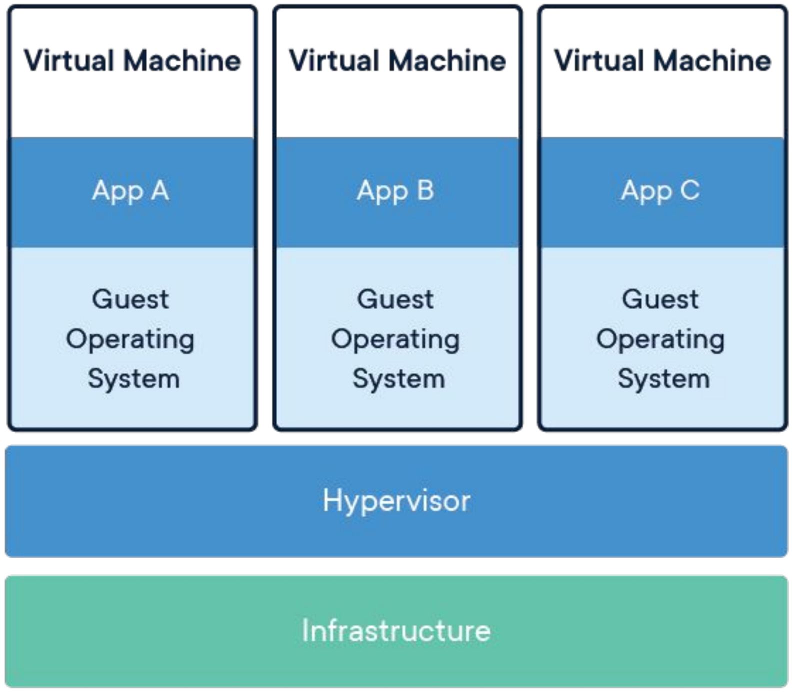
Part II: A Rich Demo

Part III: How Docker Works



Docker vs Virtual Machines





Docker vs Virtual Machines

Pets vs. Cattle

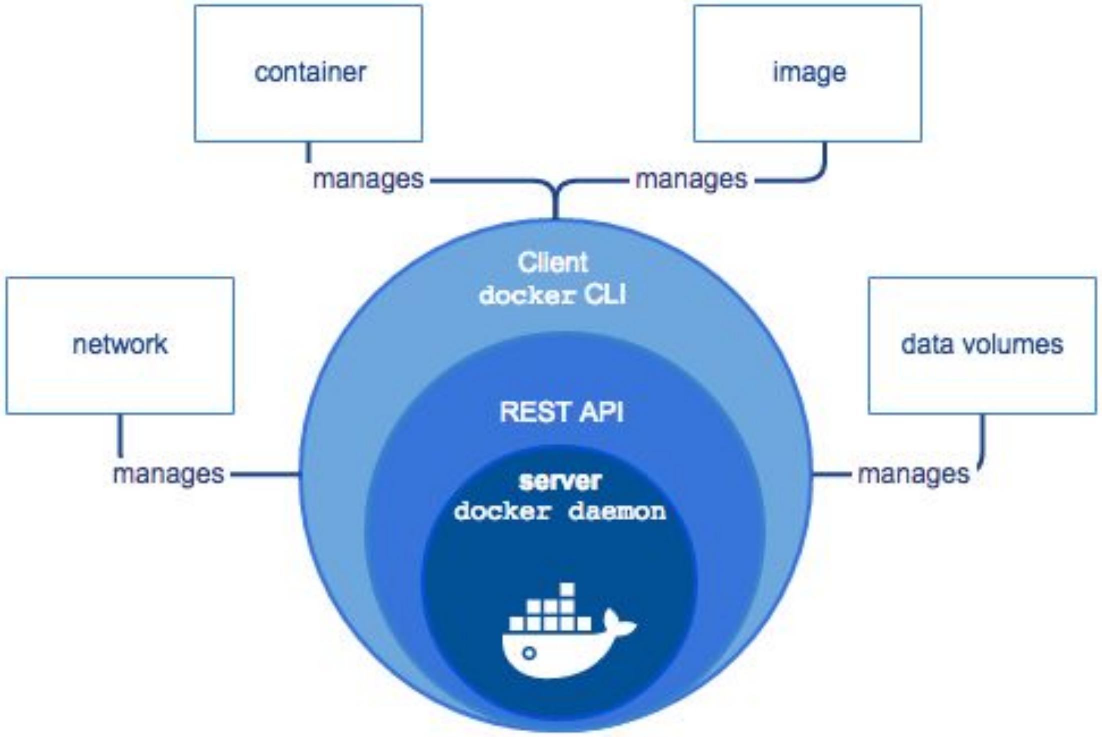
- Short lived, Ephemeral
- Don’t restart it, replace it
- Eliminate “uniqueness”

Security Considerations

- More shared resources
- Don’t run as root!

Size

- Containers can be very small
- Lightweight Alpine Linux distro
    - gcc vs. musl



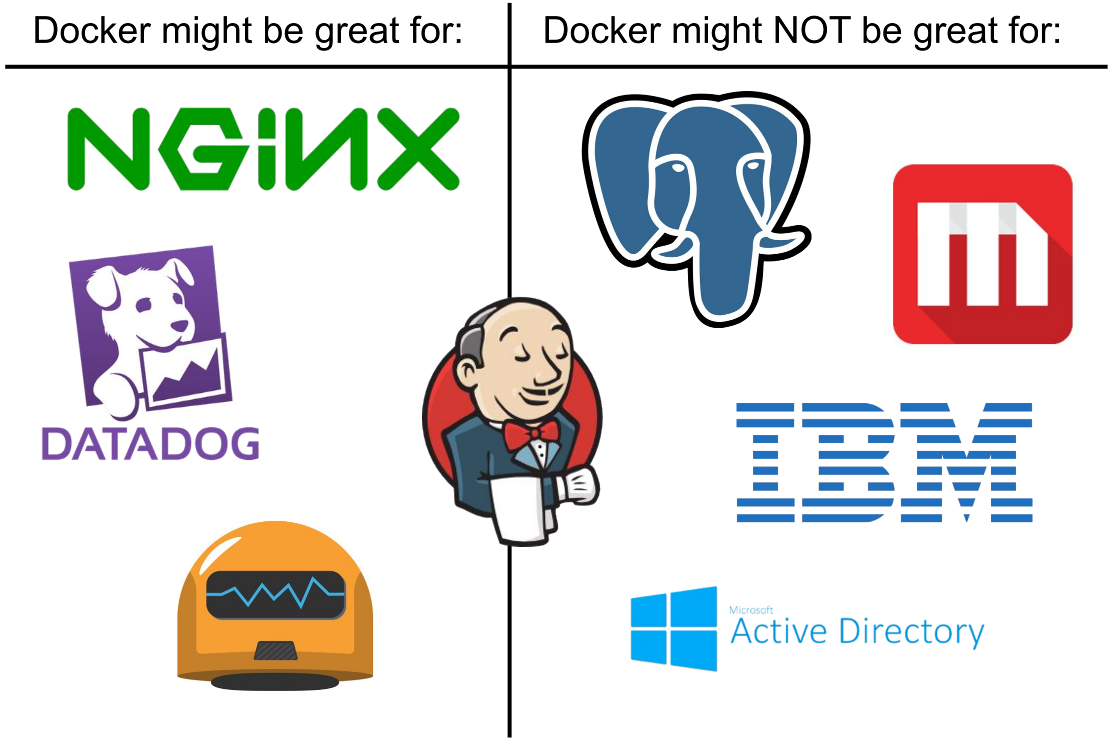
Docker vs Virtual Machines

When to use Containers

- Stateless applications
  - Webserver
- Testing environments
- Local Development

When not to use Containers

- Stateful applications
  - (prod) Database
- “Enterprise Software”
  - Often meant to be run in a specific way
- Applications that uses resources aggressively or unpredictably



Docker Engine components

docker CLI - How you interact with Docker through REST API

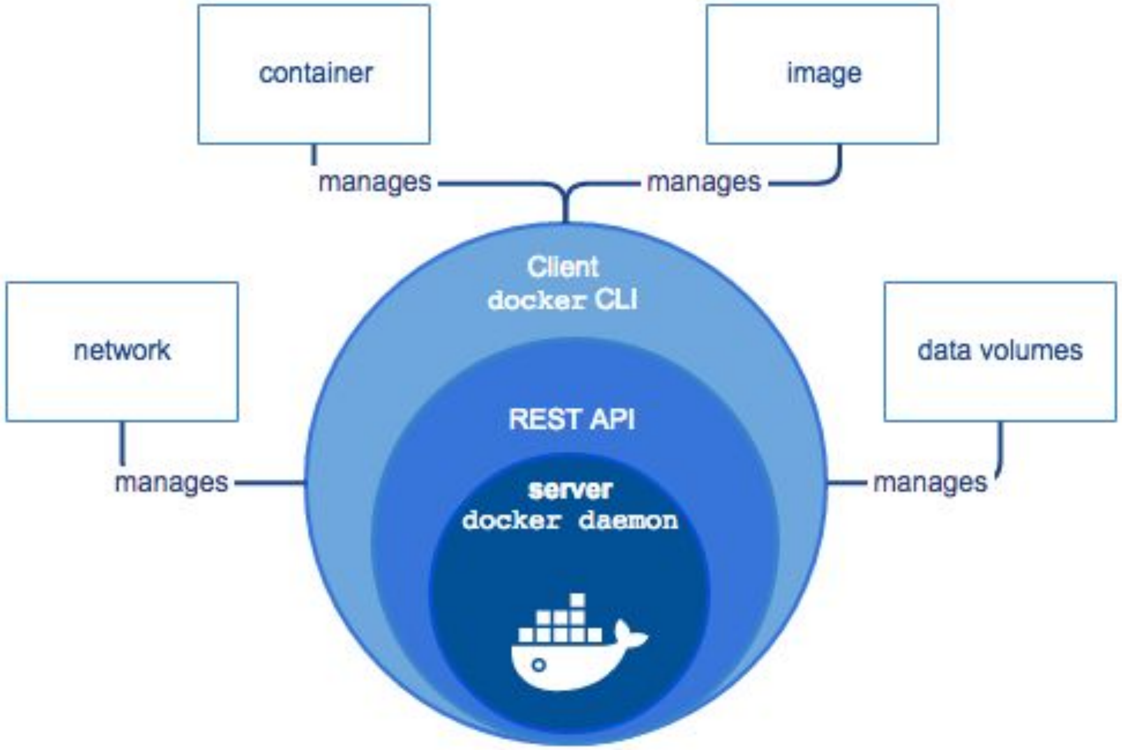
docker daemon - Creates/manages images, containers, networks, and volumes

image - Ordered collection of root filesystem changes and the corresponding execution parameters

container - Runtime instance of an image

network - Allows containers and the host to talk to one another

volume - Mechanism for sharing and/or persisting container data



`Dockerfile` and `docker-compose.yml`

File `Dockerfile` details how to build the container image.

- Define base Linux image
- Create user and set permissions
- Install Gems
- Define startup command

File `docker-compose.yml` details how to run the containers.

- Define volumes for persisted or shared directories (database, Gems, socket for Nginx)
- Define networks
- Define env vars to pass in.
- Define dependencies so when Rails starts it will also start Nginx and Postgres.
- Declare what images to use, a locally built one or a remote one (public or private).
- Like a foreman Procfile

Images

Images are a collection of layers.

Dockerfile

```
FROM debian:stretch-slim
ARG APP=blog
ARG APP_NUMBER=14011

RUN mkdir -m 0777 /app-build /bundle && \
   addgroup --gid $APP_NUMBER --system $APP && \
   adduser --uid $APP_NUMBER --system $APP --ingroup $APP
USER $APP
WORKDIR /app-build

COPY --chown=14011:14011 . /app-build/

ENV BUNDLE_PATH=/bundle BUNDLE_BIN=/bundle/bin GEM_HOME=/bundle GEM_PATH=/bundle
RUN bundle install --jobs 4 --binstubs

CMD ["bundle", "exec", "puma", "-C", "config/puma.rb"]
```

Start from a base image from hub.docker.com or an internal one.

Each line, no matter how trivial, is a new read/write **layer** built on top of the read-only `FROM` layer.

Install Gems into `/bundle` directory which will be persisted in run-time config file `docker-compose.yml`

Declare the command to run when `docker-compose up` is invoked.

Containers

Containers are instances of images.

docker-compose.yml

Specify runtime settings in `docker-compose.yml`

```
version: "3.2"
volumes:
 socket_dir:
 bundle_cache:
 pgdata:
networks:
 frontend:
 backend:
services:
 postgres:
   image: postgres:11.1-alpine
   ports:
     - '54311:5432'
   environment:
     POSTGRES_USER: blog
     POSTGRES_PASSWORD: asdf
   volumes:
     - pgdata:/var/lib/postgresql/data
   command: >
     -c ssl=on
   networks:
     backend:
       aliases:
         - blog-postgres.docker
nginx:
   image: 999999999999.ecr.us-east-1.amazonaws.com/novu/nginx:0.4.1
   ports:
     - '14011:14011'
   volumes:
     - ./log/nginx:/var/log/nginx:cached
     - socket_dir:/socket
   environment:
     NGINX_PORT: '14011'
   networks:
     frontend:
       aliases:
         - blog-nginx.docker
 app:
   build:
     context: .
   depends_on:
     - postgres
     - nginx
   networks:
     - backend
   command: bundle exec puma -C config/puma.rb
   volumes:
     - bundle_cache:/bundle
     - ./:/opt/blog:delegated
     - ./log/puma:/var/log/blog/puma:cached
     - ./log/blog:/opt/blog/log:cached
     - socket_dir:/opt/blog/socket
```

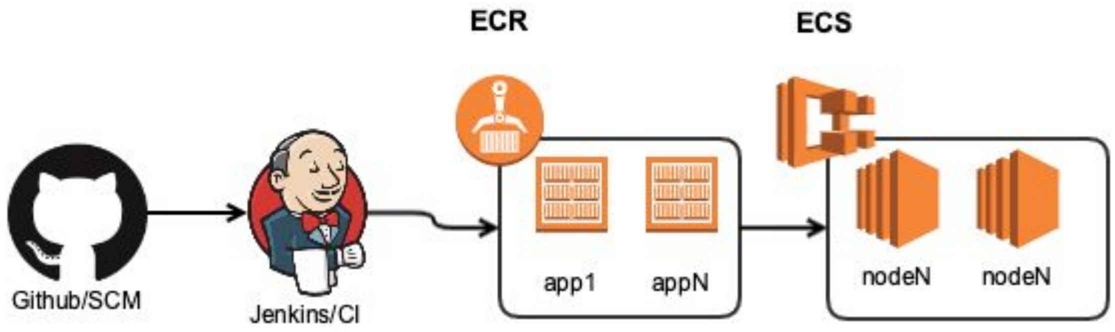
Create custom images

Images are immutable. An image can be made up of one or many upstream “base” images.

Images can be tagged. “latest” is a very common tag represented the most recent version of an image. SHA can also be used. If no tag is specified, “latest” is used.

Custom images can be stored in your private repo, like Amazon ECR (**E**lastic **C**ontainer **R**egistry).

Container platforms, such as Amazon ECS (**E**lastic **C**ontainer **S**ervice), pull images from repos.



Unresolved Issues and Challenges

- **VSCode plugins** look for Rubocop on host.
- **I/O.** Sluggish file system I/O ***(MacOS only. Doesn't affect production Linux)***.Docker/community continues to work on resolutions.
- **Shared Memory/CPU** between host and Docker VM (MacOS only) means some configuration is needed.
- **ENV var support in Rails** Rails config is often YML based. Direct ENV var consumption can be awkward.

It's worth noting the Novu team members on Linux fare better by a large margin (i.e. rspec tests run in half the time).

Getting Started

- github.com/novu/whale-of-a-tale
- docs.docker.com/get-started
- docker.com/get-started

Fun and Surprises

- github.com/cdr/code-server VSCode in the browser on Mac via Docker containers run Linux.
- github.com/localstack/localstack Run AWS commands against a fleet of docker containers locally, for free
- github.com/bibendi/dip dip*(**D**ocker **I**nteraction **P**rocess)* gem by Evil Martians.

Thanks for attending!

Questions?

andrew.rich@novu.com
aaron.bartell@novu.com
john.knutson@novu.com

novu.com/join-us

10 YEARS
StarTribune
TOP
WORK
PLACES
2019

StarTribune
TOP
150
WORKPLACES
2018

Benefits

Highly competitive salary

100% health/dental premiums covered

401K with no cap on employer match

Yearly funds for training and conferences

A goal of keeping meetings to a minimum

A focus on quality

Fun and collaborative work environment

© 2019 NovuHealth, LLC. All rights reserved.