Containerizing Local Development... Is It Worth it?

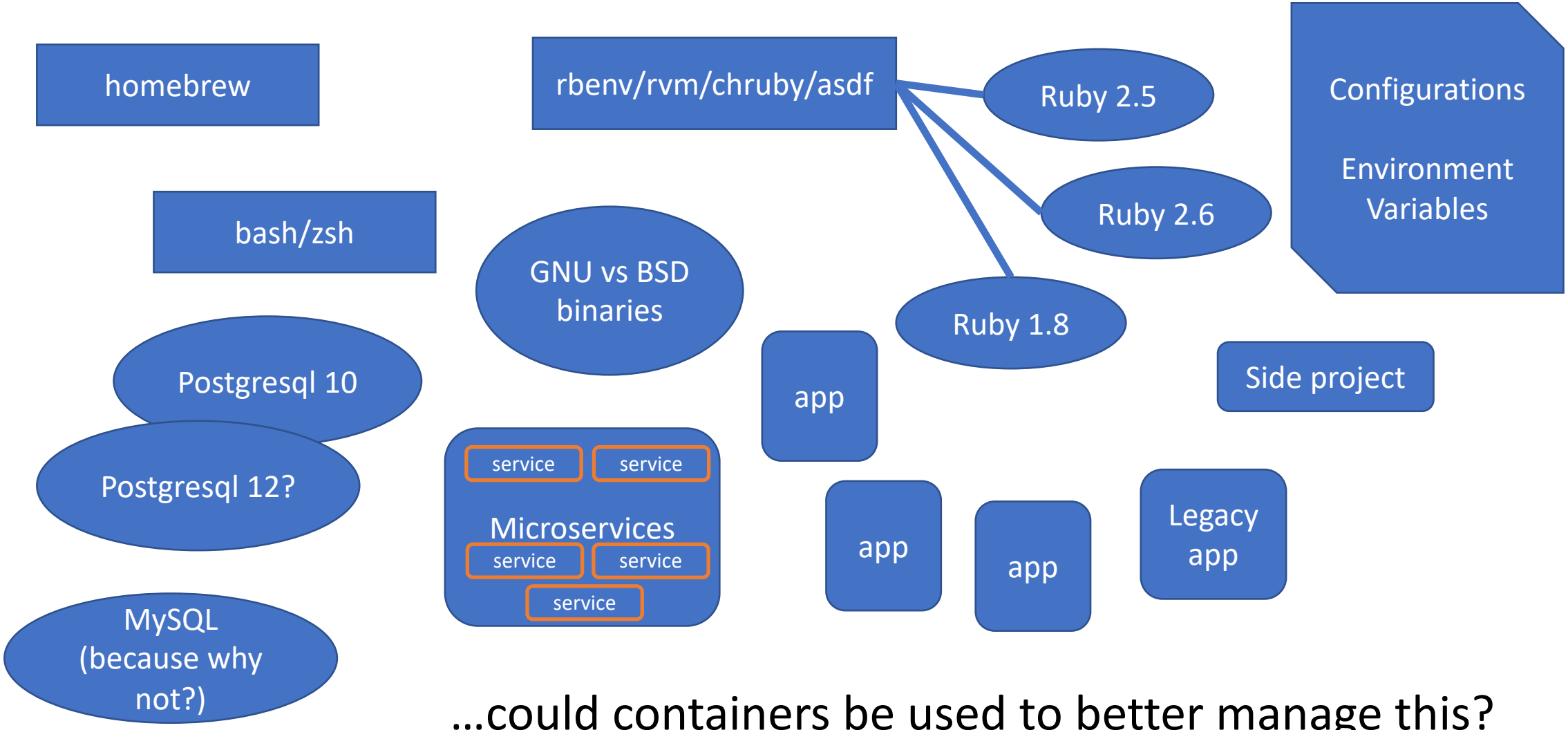
Tony Drake

@t27duck, Indiana

RubyConf 2019

Copy of Slides: github.com/t27duck/showandtell

Currently… in Local Development-Land…



…could containers be used to better manage this?

Ground Rules

- Using Docker Community Edition
- Composition with docker-compose (as in, not Kubernetes)
- I am not an expert
- There is no correct setup
- "Ruby app" == "Ruby code (usually a web app) doing some work and may have external dependencies"
- Your app may not be containerized in production already
- This is ***not*** a tutorial on how containers work or how to use Docker!

Scenarios

- 1 – N Ruby apps
  - Independent
  - Separated dependencies
- Multiple Ruby apps
  - Some / all talk to each other
  - Separated external stores
  - Some external stores could be shared
- Basic Ruby hacking (gem building)
  - Dependencies possible

Containerizing an App (In Brief): Let's Pretend…

- "Complex" web application
- Needs postgresql
- Needs redis
- Uses imagemagick
- External settings handled by environment variables

Gemfile:

```
source "https://rubygems.org"

gem "pg"
gem "redis"
gem "sinatra"
```

app.rb:

```
require "sinatra"
set :bind, "0.0.0.0"

get "/" do
  ENV.map do |k, v|
    "<strong>#{k}:</strong> #{v}"
  end.sort.join("<br />")
end
```

Dockerfile

```
FROM ruby:2.6.5-stretch

EXPOSE 4567

ENV BUNDLE_PATH=/bundle \
   BUNDLE_BIN=/bundle/bin \
   GEM_HOME=/bundle
ENV PATH="${BUNDLE_BIN}:${PATH}"


RUN \
  echo "deb http://apt.postgresql.org/pub/repos/apt/ stretch-pgdg main" | tee /etc/apt/sources.list.d/pgdg.list && \
  wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add - && \
  apt-get update && \
  apt-get install -y --no-install-recommends \
  postgresql-client-12 \
  imagemagick \
  && rm -rf /var/lib/apt/lists/*

WORKDIR /app

COPY Gemfile* ./
RUN bundle

CMD "bash"
```

Dockerfile

```
FROM ruby:2.6.5-stretch
```

Base image as a starting point
https://hub.docker.com/

```
EXPOSE 4567
```

Port to expose to the docker network

```
ENV BUNDLE_PATH=/bundle \
    BUNDLE_BIN=/bundle/bin \
    GEM_HOME=/bundle
ENV PATH="${BUNDLE_BIN}:${PATH}"
```

Environment variables for the container

(Custom path for installed gems)

Dockerfile

```
RUN \
  echo "deb http://apt.postgresql.org/pub/repos/apt/ stretch-pgdg main" | tee /etc/apt/sources.list.d/pgdg.list && \
  wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add - && \
  apt-get update && \
  apt-get install -y --no-install-recommends \
  postgresql-client-12 \
  imagemagick \
  && rm -rf /var/lib/apt/lists/*
```

Add 3rd party repositories

Install needed packages from OS package manager

Delete unneeded files

Dockerfile

WORKDIR /app	Directory where code will live
COPY Gemfile* ./	Add Gemfile + Gemfile.lock into container (in root of WORKDIR)
RUN bundle	Install gems into the container
CMD "bash" (or ENTRYPOINT "script-file")	A default command to run once built

Dockerfile

```
FROM ruby:2.6.5-stretch

EXPOSE 4567

ENV BUNDLE_PATH=/bundle \
   BUNDLE_BIN=/bundle/bin \
   GEM_HOME=/bundle
ENV PATH="${BUNDLE_BIN}:${PATH}"


RUN \
  echo "deb http://apt.postgresql.org/pub/repos/apt/ stretch-pgdg main" | tee /etc/apt/sources.list.d/pgdg.list && \
  wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add - && \
  apt-get update && \
  apt-get install -y --no-install-recommends \
  postgresql-client-12 \
  imagemagick \
  && rm -rf /var/lib/apt/lists/*

WORKDIR /app

COPY Gemfile* ./
RUN bundle

CMD "bash"
```

docker-compose.yml

Either the location of a Dockerfile to build and run or a premade container

Files (code) to mount from local drive into the container

Override for container's default CMD

Port in container to expose to port on your system

```
version: "3"

services:
  web:
    build: .

    volumes:
      - .:/app:delegated

    command: ["ruby", "app.rb"]

    ports:
      - "4567:4567"
```

docker-compose.yml

```yaml
version: "3"

services:
  web:
    build: .
    volumes:
      - .:/app:delegated
    command: ["ruby", "app.rb"]
    ports:
      - "4567:4567"

  db:
    image: postgres:12-alpine
    logging:
      driver: "none"
  redis:
    image: redis
    logging:
      driver: "none"
```

External services (database and redis)

Uses prebuild images from the docker repository

Logging disabled for now unless needed

docker-compose.yml

Add environment variables to the web service (container)

Set db and redis services to start up when web starts up

```yaml
version: "3"

services:
  web:
    build: .
    volumes:
      - .:/app:delegated
    command: ["ruby", "app.rb"]
    ports:
      - "4567:4567"
    environment:
      - DATABASE_URL=postgres://postgres:postgres@db/app_db
      - REDIS_URL=redis://redis:6379
    depends_on:
      - db
      - redis


  db:
    image: postgres:12-alpine
    logging:
      driver: "none"

  redis:
    image: redis
    logging:
      driver: "none"
```

docker-compose.yml

```yaml
version: "3"

services:
  web:
    build: .
    volumes:
      - .:/app:delegated
    command: ["ruby", "app.rb"]
    ports:
      - "4567:4567"
    environment:
      - DATABASE_URL=postgres://postgres:postgres@db/app_db
      - REDIS_URL=redis://redis:6379
    depends_on:
      - db
      - redis


  db:
    image: postgres:12-alpine
    logging:
      driver: "none"

  redis:
    image: redis
    logging:
      driver: "none"
```

docker-compose commands

- $ docker-compose build
  $ docker-compose build [service]

- Pulls and builds containers based on docker-compose.yml
- Only rebuilds if changes in Dockerfile results in a different container
- Use --no-cache to effectively force a rebuild

docker-compose commands

- $ docker-compose up
  $ docker-compose up [service]
- $ docker-compose up web
- $ docker-compose down
  $ docker-compose down [service]

- Brings up all services (or specified services) outlined in docker-compose.yml
- Services in depends_on are automatically brought up
- Services whose container isn't build are built at this time
- Ctrl+C to stop all containers
- Alternatively, "down" to stop

docker-compose commands

- $ docker-compose exec [service] [cmd]
- $ docker-compose exec web rake –T

  $ docker-compose exec web irb

  $ docker-compose exec web bash

- Connects to a running service and runs a command/program

Running our "app"

$ cd myapp/

$ docker-compose build

$ docker-compose up

docker-compose

docker-compose

```
$ docker-compose up
Creating network "dtest_default" with the default driver
Creating dtest_redis_1 ... done
Creating dtest_db_1    ... done
Creating dtest_web_1   ... done
Attaching to dtest_db_1, dtest_redis_1, dtest_web_1
db_1      | WARNING: no logs are available with the 'none' log driver
redis_1   | WARNING: no logs are available with the 'none' log driver
web_1     | [2019-10-24 23:56:47] INFO  WEBrick 1.4.2
web_1     | [2019-10-24 23:56:47] INFO  ruby 2.6.5 (2019-10-01) [x86_
64-linux]
web_1     | == Sinatra (v2.0.7) has taken the stage on 4567 for devel
opment with backup from WEBrick
web_1     | [2019-10-24 23:56:47] INFO  WEBrick::HTTPServer#start: pi
d=1 port=4567
```

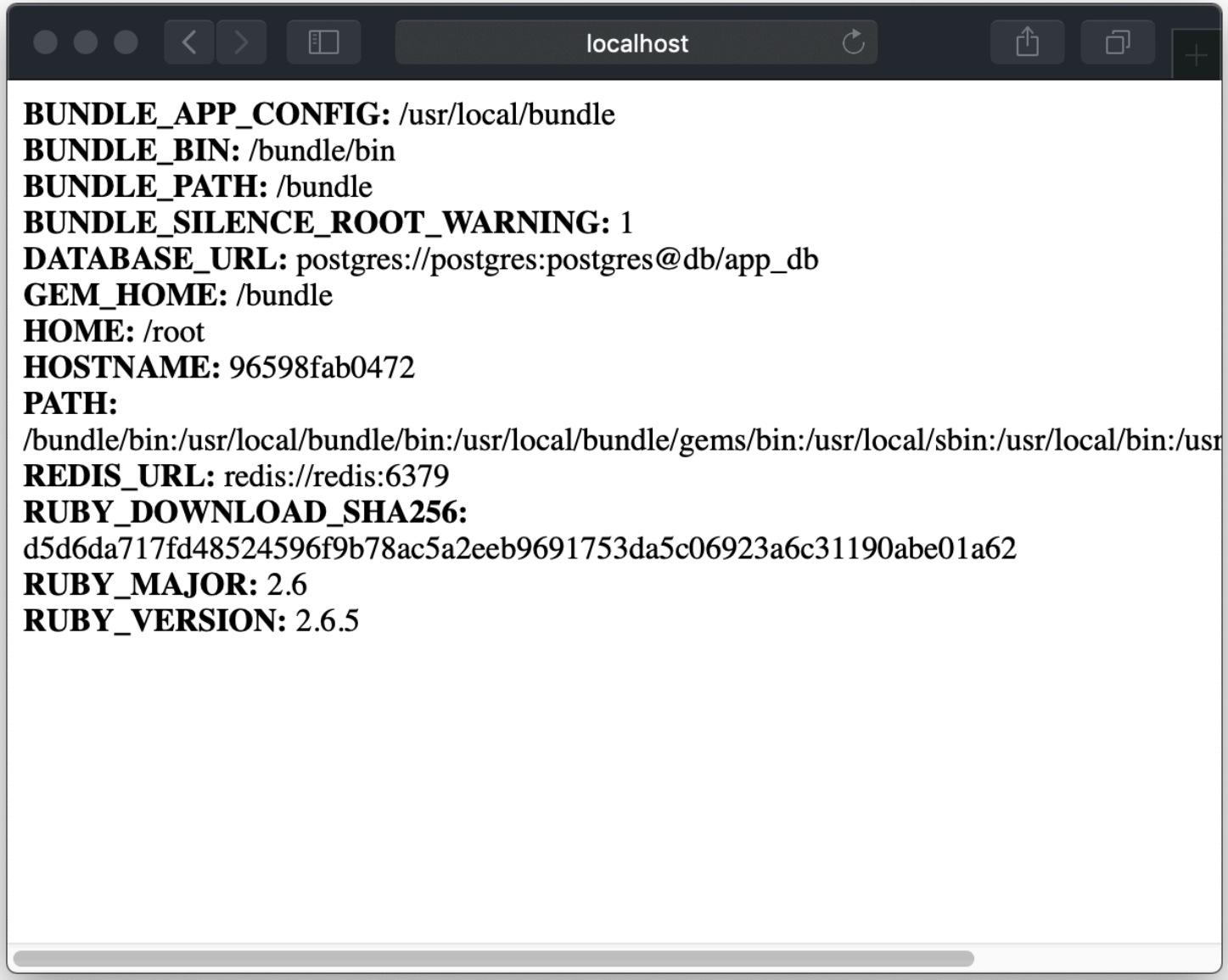
Running our "app"

$ cd myapp/

$ docker-compose build

$ docker-compose up

Visit http://localhost:4567

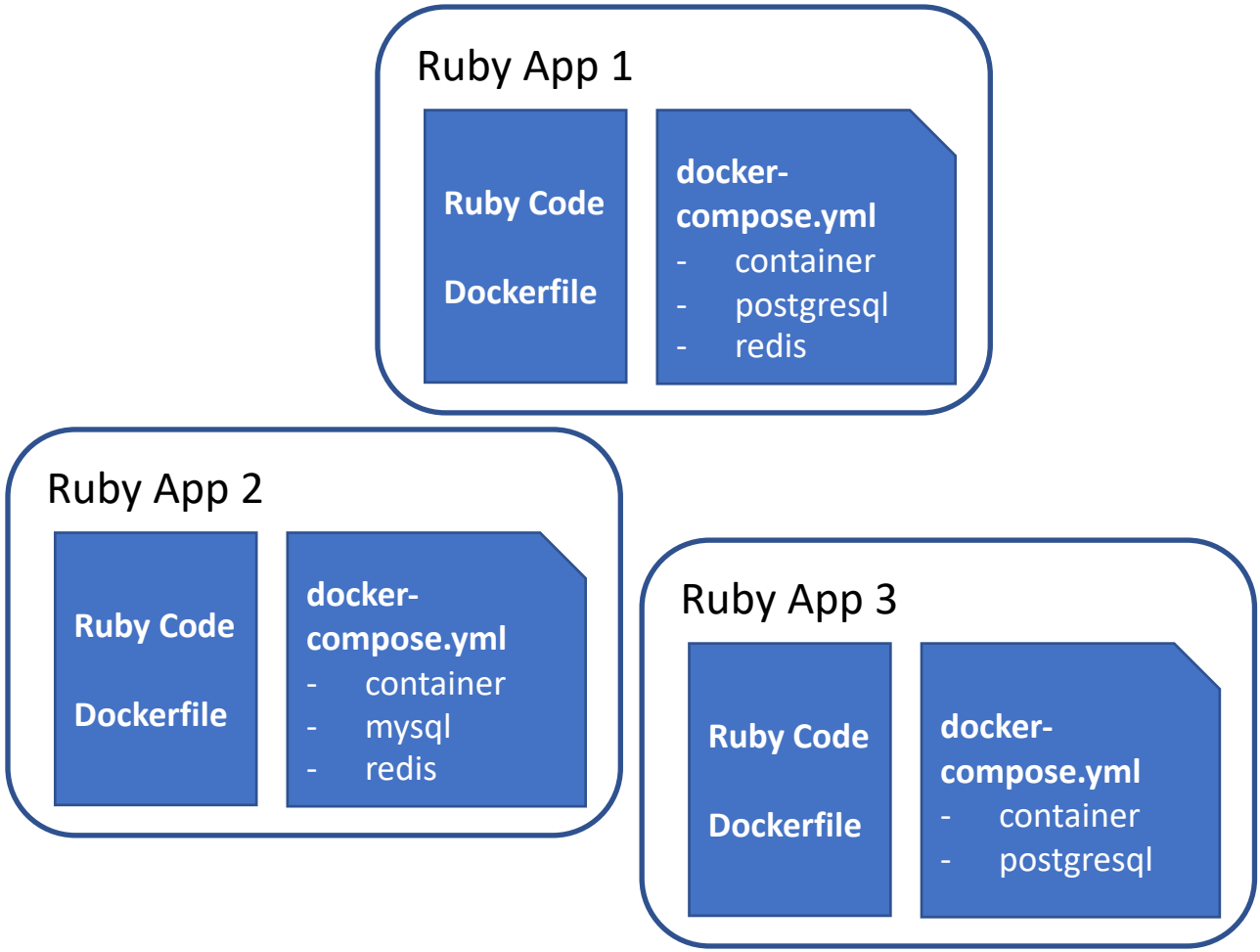


Scenario – One or more independent apps

- All dependencies are siloed
- More direct context switching
- Easier to focus on one app
- Closer representation of production

$ cd app1/

$ docker-compose up



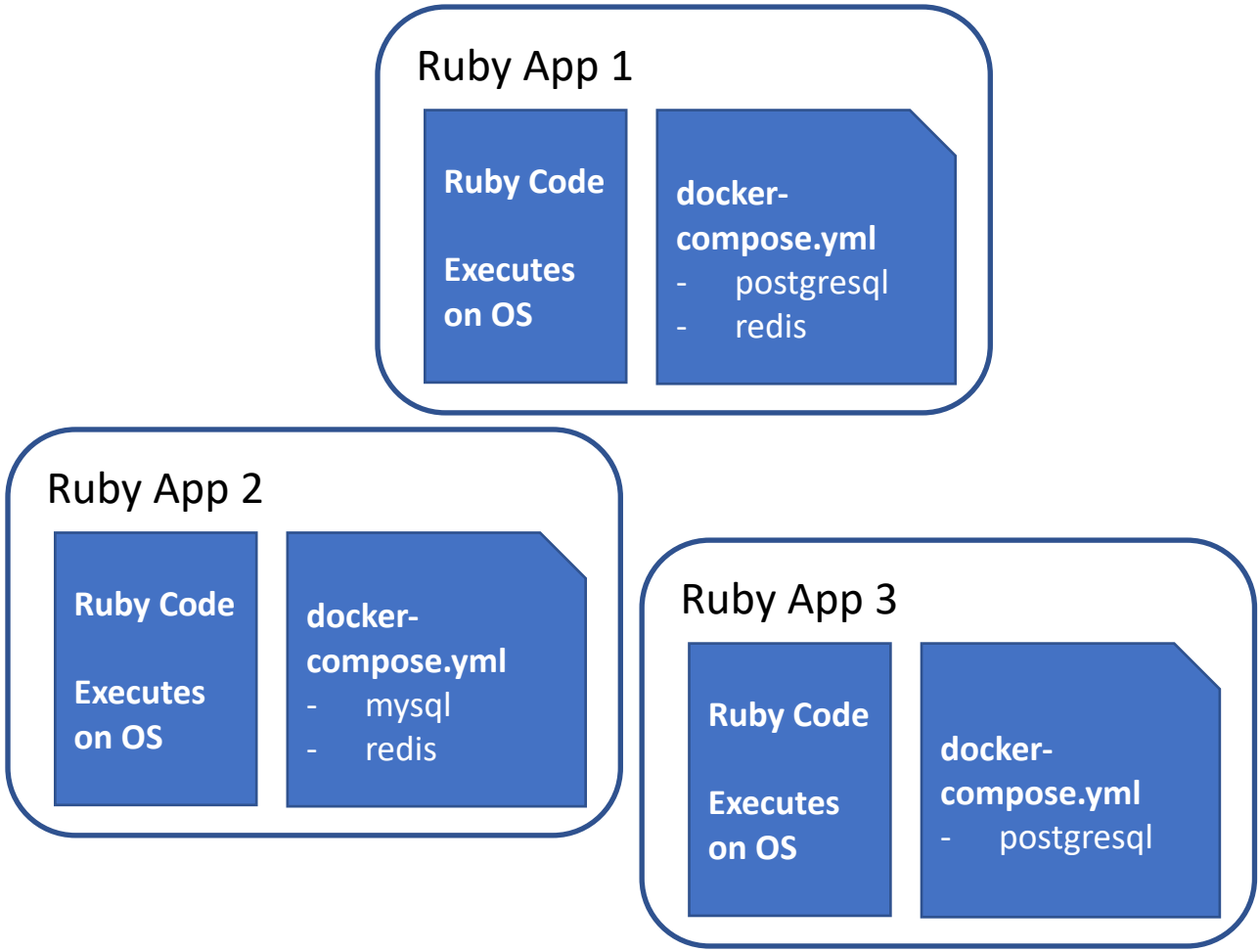
Scenario – One or more independent apps (alternative setup)

- Containers only used for external dependencies
- Code runs directly on OS (performance boost)
- Use exposed Docker network to connect to containers from code

$ cd app1/

$ docker-compose up db redis

$ ruby app.rb



Scenario – One or more independent apps

Code in Docker - Worth it?

Setup used by…

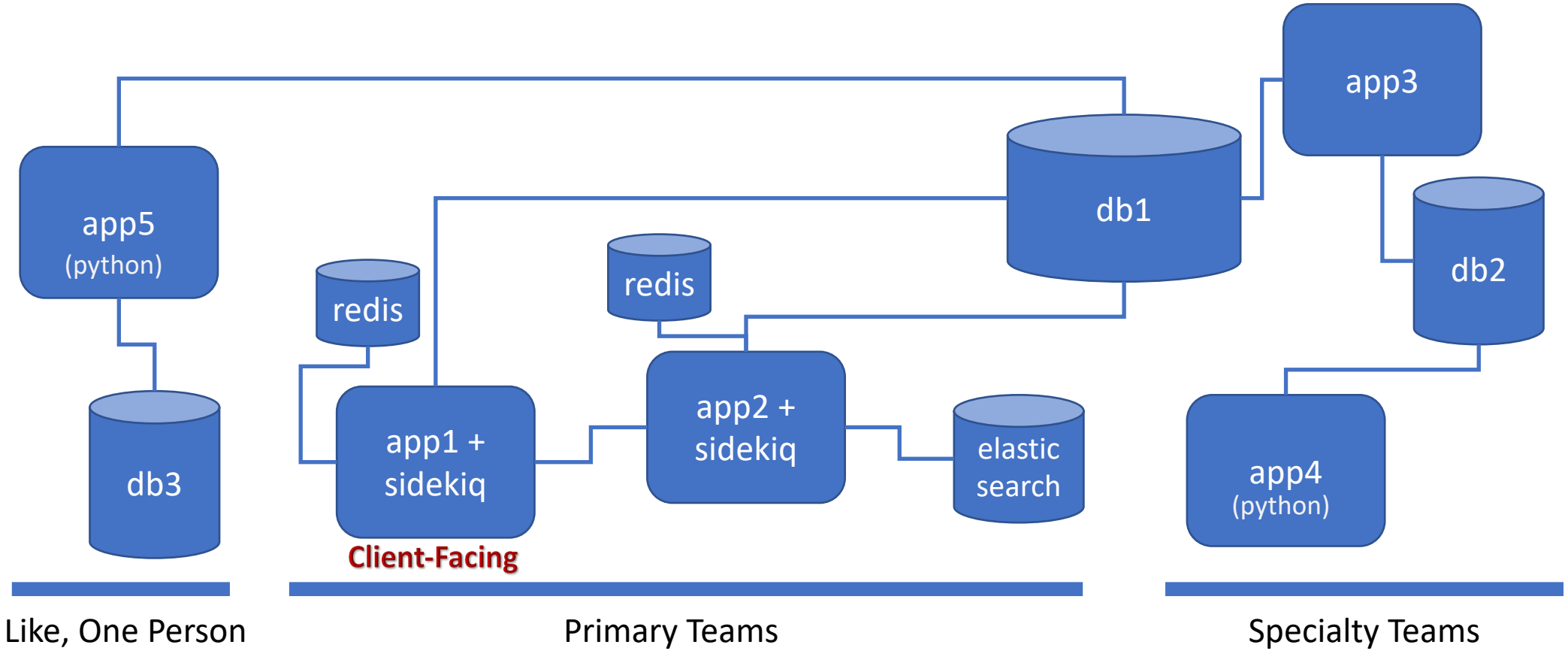
All my side projects

Code outside Docker - Worth it?

Setup used by…

Lessonly

Scenario – A System Like This…



Scenario – Multiple Apps, Multiple Teams

/work/app1/*git-repo-with-code
/work/app2/*git-repo-with-code
/work/app3/*git-repo-with-code
/work/app4/*git-repo-with-code
/work/app5/*git-repo-with-code

/work/bootstrap

/work/bootstrap

- docker-compose.yml
- Dockerfile-app1
- Dockerfile-app2
- Dockerfile-app3
- Dockerfile-app4
- Dockerfile-app5

Scenario – Multiple Apps, Multiple Teams (docker-compose.yml)

```
services:
  db:
    image: postgres:12
  redis:
    image:redis
  app1:
    build:
      context: ../app1
      dockerfile: ../boostrap/Dockerfile-app1
    environment:
      - DATABASE_URL=postgres://…
      - REDIS_URL=redis://…
    depends_on:
      - redis
      - db
      - app2
```

If all DB versions are the same, share one instance

Apps may use a different redis database per cluster

Directs docker to use app1 directory as its root

App-specific environment variables

Other services to start up when it starts up

Scenario – Multiple Apps, Multiple Teams

```
services:
  db:
  redis:
  elasticsearch:
  app1:
  app2:
  app3:
  app4:
  app5:
```

```
$ cd boostrap
$ docker-compose up app1 app2
$ docker-compose exec app1 rake
```

Scenario – Multiple Apps, Multiple Teams (compose files per-team)

docker-compose.yml

```
services:
  db:
  redis:
  elasticsearch:
  app1:
  app2:
```

docker-compose.st.yml

```
services:
  db:
  redis:
  app3:
  app4:
```

Scenario – Multiple Apps, Multiple Teams (compose files per-team)

$ docker-compose up *(app1 and app2)*

$ docker-compose –f docker-compose.st.yml up *(app3 and app4)*

$ docker-compose –f docker-compose.st.yml –f docker-compose.yml up *(app1, app2, app3, and app4)*

Scenario – Multiple Apps, Multiple Teams

- Allows teams to focus on the app(s) they care about
- Closer represents production
  - Multiple apps and databases communicating
  - Independent systems
- Bootstrapping for new team members a little more straight forward
- Requires communication and a little more organization

Scenario – Multiple Apps, Multiple Teams

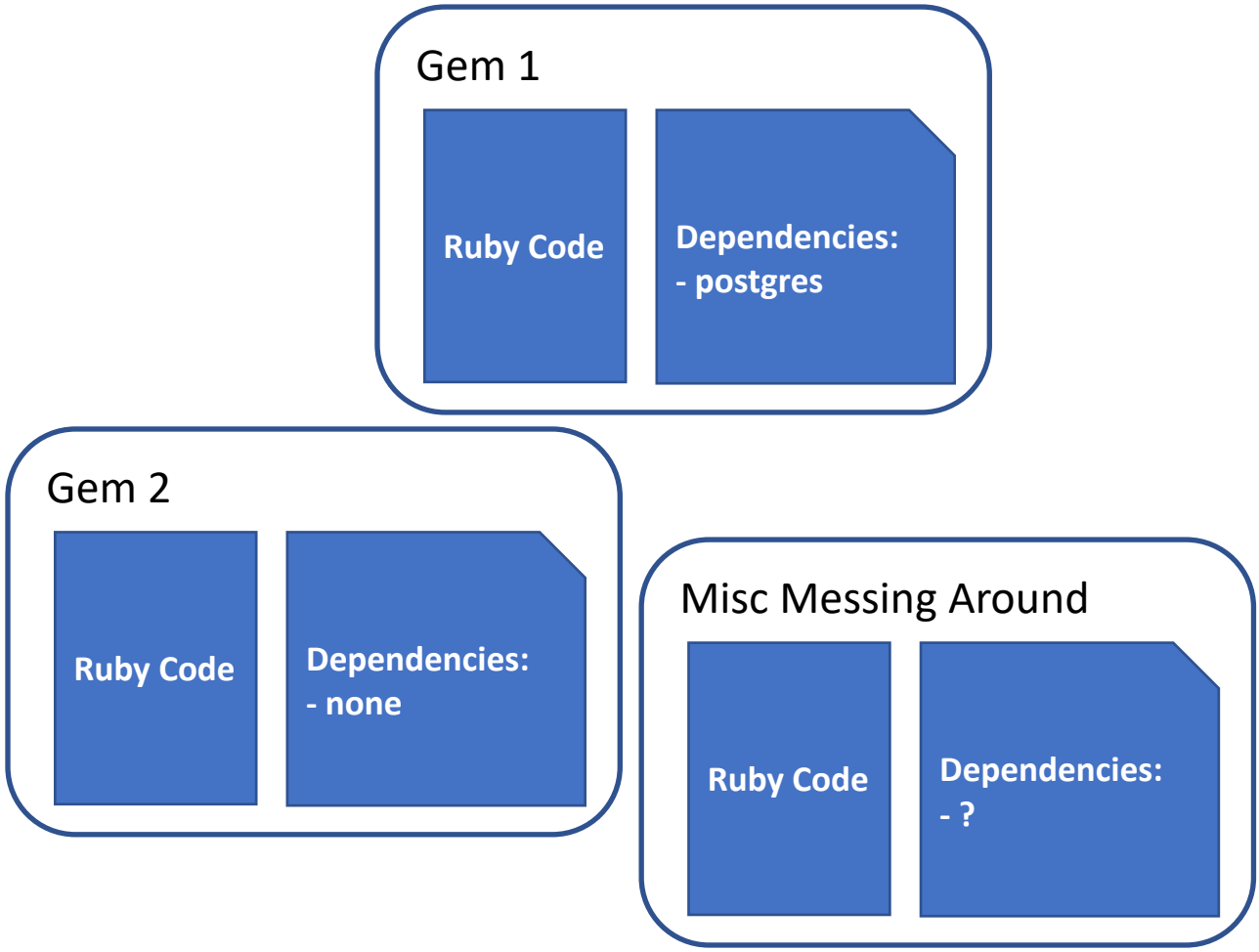
Worth it?

Setup used by…

springbuk®

Scenario – Simple Ruby Hacking / Gem Dev

- Making gems with multiple Ruby versions possible
- Scratch pad for random Ruby code execution



Scenario – Simple Ruby Hacking / Gem Dev?

/code/Dockerfile

/code/docker-compose.yml

/code/stuff/

/code/stuff/gem1-code/

/code/stuff/gem2-code/

Scenario – Simple Ruby Hacking / Gem Dev?

Dockerfile

```
FROM buster

# Bootstrap rbenv + ruby-build

WORKDIR stuff
CMD "bash"
```

docker-compose.yml

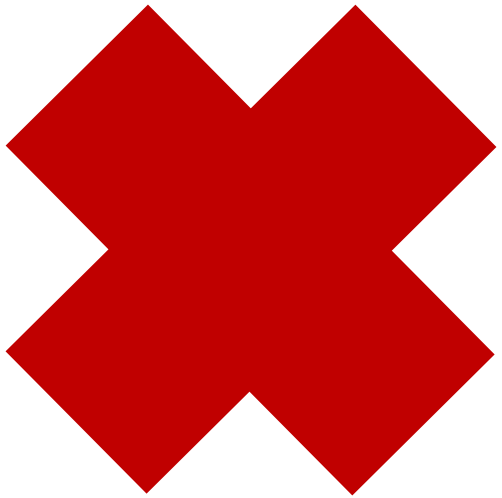
```
services:
  code:
    build: .
    volumes:
      - .:/stuff:delegated
    environment: …
  postgres: …
  redis: …
  mysql: …
```

Scenario – Simple Ruby Hacking / Gem Dev?

- Any and all external dependencies within docker network
- No need to install Ruby locally to execute code
- As dependency needs increase, add more services
- … anything else?

Scenario – Simple Ruby Hacking / Gem Dev?

Worth it?



(Unless you have a lot of external dependencies, maybe)

Shared Amongst All Scenarios (PROs)

- Simplifies bootstrapping
  - Checkout repo(s), build/run containers
- Closer representation of production
- All dependencies contained
  - Multiple versions of the database
- Ruby upgrades easy
  - Update Dockerfile, rebuild
- Broken? Just rebuild!

Shared Amongst All Scenarios (CONs)

- Slower (You're running in a VM)
- Linters have to still be installed locally for fast feedback
- macOS's filesystem isn't great…
- Adds a layer of local complexity
- Docker likes to eat RAM… and CPU… and disk space…

So Local Containers…

Worth it?

???

Your call

(90% of the time, I think it's worth it)

The End!

My Twitter: @t27duck

GitHub: t27duck

Copy of Slides:
github.com/t27duck/showandtell

Couple Neat Indiana Ruby Shops:

Lessonly

lessonly.com

springbuk®

springbuk.com