Navigating the Docker Devstack

OpenCraft_

Kshitij Sobti
kshitij@opencraft.com
@xitij2000 (github)

Cliff Dyer
cliff@opencraft.com
@jcdyer (github/twitter)

Overview

1. Docker Basics
2. Docker vs. Vagrant
3. The Docker Devstack under the hood
4. Some tips and tricks

Major elements

Docker:
  Define and run a container

Docker Compose:
  Orchestrate multiple containers

Makefile:
  Simplify command execution

Docker Basics

Docker is like virtualenv

Like virtualenv but for any Linux/Windows software

Packages installed in Docker don’t affect host system

Packages installed on host system don’t affect Docker

Docker is Like Git

Push/Pull/Commit/Diff Software distributions

Changes build on top of each other

Add tags for significant states

Push to different repositories

Build on top of others’ work

Main Docker nouns

Image

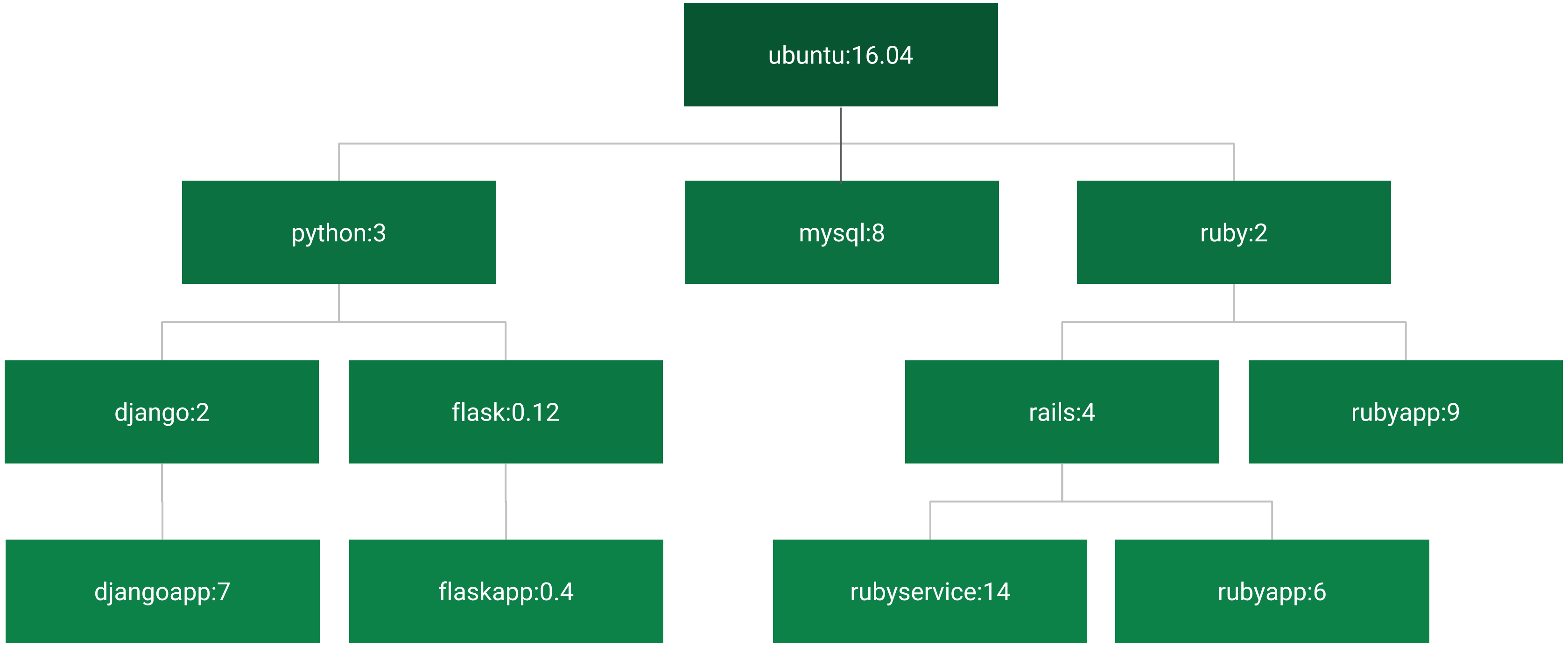
Layer

Container

Volume

Docker to Vagrant mapping

Image	Box
Layer	Snapshots
Container	VM
Volume	Shared folders



Hierarchy of Layers with Docker

Dockerfile

- A recipe for building images
- Roughly each line / step becomes its own layer
- Steps are cached
- Like taking a Vagrant snapshot at each line

```
FROM ubuntu:16.04
RUN apt-get update
RUN apt-get install python3
RUN pip install django
ADD .
CMD [“./manage.py runserver”]
```

Docker images

Are read only

Made up of layers

Which can be reused

Created using Dockerfiles

Can be tagged / named

Can be shared / uploaded

Docker container

Builds on top of a stack of image layers

Runs software in a read-write layer

Changes made are discarded along with container

It’s just an execution environment for an app

Has a single entrypoint

Docker volume

Retained as containers are created / destroyed

Can be linked to a specific folder on the host

Can be shared between containers

Don’t have layers, or tags

Docker vs Vagrant

Devstack comparison

Similarities

- Build environment using a Dockerfile / Vagrantfile
- Isolate the developer environment from the host system
- Restrict the resource usage
- Mount/share folders between host and guest
- Open ports between host and guest
- Can access shell inside container / VM

Some important differences

- Vagrant can work with other things than VirtualBox, including Docker
  - Docker shockingly only supports Docker
- Docker doesn’t need a whole bootable OS image
- Docker shares the host system’s kernel (on Linux/Windows*)
- Docker “shapshots” (tags) don’t keep the state of data on volumes
  - Docker devstack separates each service into a separate container, so each of them can potentially need to be snapshotted
- On non-Linux systems Docker still runs on VirtualBox so some performance benefits are negated.
- Docker is easier to set up on cloud hosting platforms

Performance

- Docker commands run directly on Linux
- Starts in seconds or less
- Potentially lower memory usage
- Potentially lower disk usage
- No virtual machine needs to spin up
  - On OSX / Windows, VM needs to spin up once then is reused
- Can run dozens, even hundreds of containers on Docker
- VirtualBox image needs to be a bootable full hard drive image
- Docker image can be single static binary
- Docker snapshots are often faster and lighter

Docker is (kind of) Linux-only

- Docker relies on core kernel features
- Only supported “natively” on Windows and Linux
- Windows-native Docker can only run Windows app
- Linux-native Docker can only run Linux apps
- “Regular” Docker on OSX and Windows uses VirtualBox

The Docker Devstack under the hood

Docker Compose files

These files describe service requirements such as:

- Shared ports
- Volumes to store data permanently
- Environment variables
- Commands to run in each container

Docker Compose

Coordinate multiple Docker containers

- Share data using volumes
- Link container networks
- Define dependencies between containers
- Manage as single system

```
version: "2.1"
services:
  memcached:
    image: memcached:1.4.24
  mysql:
    command: mysqld
    image: mysql:5.6
    volumes:
      - mysql_data:/var/lib/mysql
  ecommerce:
    command: bash -c 'very long command'
    depends_on:
      - mysql
      - memcached
    image: edxops/ecommerce:latest
    ports:
      - "18130:18130"
```

Docker Compose

Coordinate multiple Docker containers

- Share data using volumes
- Link container networks
- Define dependencies between containers
- Manage as single system

```
version: "2.1"
services:
  memcached:
    image: memcached:1.4.24
  mysql:
    command: mysqld
    image: mysql:5.6
    volumes:
      - mysql_data:/var/lib/mysql
  ecommerce:
    command: bash -c 'very long command'
    depends_on:
      - mysql
      - memcached
    image: edxops/ecommerce:latest
    ports:
      - "18130:18130"
```

Docker Compose

Coordinate multiple Docker containers

- Share data using volumes
- Link container networks
- Define dependencies between containers
- Manage as single system

```
version: "2.1"
services:
  memcached:
    image: memcached:1.4.24
  mysql:
    command: mysqld
    image: mysql:5.6
    volumes:
      - mysql_data:/var/lib/mysql
  ecommerce:
    command: bash -c 'very long command'
    depends_on:
      - mysql
      - memcached
    image: edxops/ecommerce:latest
    ports:
      - "18130:18130"
```

Docker Compose

Coordinate multiple Docker containers

- Share data using volumes
- Link container networks
- Define dependencies between containers
- Manage as single system

```
version: "2.1"
services:
  memcached:
    image: memcached:1.4.24
  mysql:
    command: mysqld
    image: mysql:5.6
    volumes:
      - mysql_data:/var/lib/mysql
  ecommerce:
    command: bash -c 'very long command'
    depends_on:
      - mysql
      - memcached
    image: edxops/ecommerce:latest
    ports:
      - "18130:18130"
```

Docker Compose

Coordinate multiple Docker containers

- Share data using volumes
- Link container networks
- Define dependencies between containers
- Manage as single system

```
version: "2.1"
services:
  memcached:
    image: memcached:1.4.24
  mysql:
    command: mysqld
    image: mysql:5.6
    volumes:
      - mysql_data:/var/lib/mysql
  ecommerce:
    command: bash -c 'very long command'
    depends_on:
      - mysql
      - memcached
    image: edxops/ecommerce:latest
    ports:
      - "18130:18130"
```

Docker Compose

Coordinate multiple Docker containers

- Share data using volumes
- Link container networks
- Define dependencies between containers
- Manage as single system

```
version: "2.1"
services:
  memcached:
    image: memcached:1.4.24
  mysql:
    command: mysqld
    image: mysql:5.6
    volumes:
      - mysql_data:/var/lib/mysql
  ecommerce:
    command: bash -c 'very long command'
    depends_on:
      - mysql
      - memcached
    image: edxops/ecommerce:latest
    ports:
      - "18130:18130"
```

Makefile

Wrapper for common docker-compose commands

Simplifies complex command chains

Sometimes it’s better to use docker-compose directly

Scripts

- destroy.sh: delete everything! start from scratch
- dump-db.sh: connect to containers and backup data in volumes
- load-db.sh: load data from included .sql files into the database
- provision-*.sh: provision a service using load-db, run migrations, collect static etc.
- provision.sh: provision almost all services using above scripts
- repo.sh: clones / reset all the needed git repos to master branch
- *.sql: Pre-rendered SQL dumps to speed up provisioning.

Some tips and tricks

Custom make targets

- Create a local.mk file and add custom make targets
- This is an ignored file so it's the best way to add common custom commands
- No need for advanced Makefile knowledge for basic stuff

A simple way to add a new command is:

```
command-name:
[TAB] some-shell-command
[TAB] cp xyz abc
```

Now when you run `make command-name` it will run all of the above

Example make targets

```
commit-all:  # Save all containers with a tag suffix
    @docker ps -f name=edx.devstack --format "{{.ID}} {{.Image}}" | \
                grep edxops | while read command; do \
        docker commit $$command$(if $(TAG),-$(TAG)) ; \
    done
```

Example make targets

```
lms-webpack: # Run "paver webpack" in the lms container
	docker exec -t edx.devstack.lms bash -c    \
		'source /edx/app/edxapp/edxapp_env &&  \
		 cd /edx/app/edxapp/edx-platform/ &&   \
		 paver webpack'
```

Example make targets

```
run-test:  # Run tests in the lms container
    @docker exec -t edx.devstack.lms bash -c     \
     'source /edx/app/edxapp/edxapp_env &&  \
     cd /edx/app/edxapp/edx-platform/ &&   \
     pytest $(TEST) $(PARAMS)'
```

Connecting to containers

- No need to ssh into container
- You can directly run one-off commands on it
- To get a shell just run bash as a one-off command
  - `docker exec --interactive --tty <container> /bin/bash`
  - `docker exec -it <container> /bin/bash`
- Run commands directly without shell
  - `docker exec <container> ls`
- `docker cp` works just like `scp`

Logs

```
docker-compose logs --follow --tail=200 lms
```

- **--follow (-f)** gives you a stream of logs
- **--tail**=**N** shows the last **N** messages instead of starting from the beginning
- **lms** or any other service name is optional, get all logs by omitting it
- set **COMPOSE_HTTP_TIMEOUT** to keep the log stream alive for longer

Take advantage of Devpi

- Devpi is awesome!
- Included recently in devstack
- Run your own PyPI
- Packages are served from local cache
- Falls back to PyPI and caches it
- Store private packages / versions

Use it as an index for your regular Python as well, and benefit.

Add the following to your pip config ($HOME/.pip/pip.conf):

```
[global]
index-url = http://localhost:3141/root/pypi/+simple/
```

Modify docker-compose files

- Comment out the services you don’t need
- Use your own custom image versions
- Change / add environment variables if needed
- Add more shared volumes / files
- Open additional ports
- Change volume driver

Volume snapshots?

Vagrant let you just take a snapshot of whole system, Docker can’t really do that.

Look into:

- Buttervolume
- Convoy

Extend existing backup scripts to make date-wise backups

Crazy idea: Move volumes to btrfs partition and use btrfs snapshots

Devstack in the cloud

- Can run Docker devstack on AWS / DigitalOcean / Google Cloud
  - Docker runs very well
  - Much faster network
  - Pull / push images are often almost instantaneous
  - Useful for people in countries with slow internet (*cough* India)
  - Accessible from multiple computers
- Can’t run VirtualBox
  - Well you can, but probably shouldn’t

Questions?

OpenCraft_

Kshitij Sobti
kshitij@opencraft.com
@xitij2000 (github)

Cliff Dyer
cliff@opencraft.com
@jcdyer (github/twitter)

Move docker data

Docker images, volumes and containers can take up a lot of space

Move /var/lib/docker (on most distros?) to a separate partition

Use btrfs for native snapshot and layers support