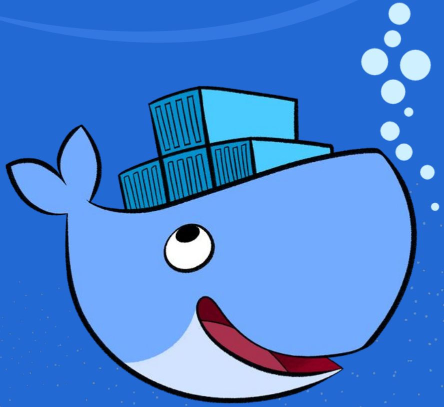
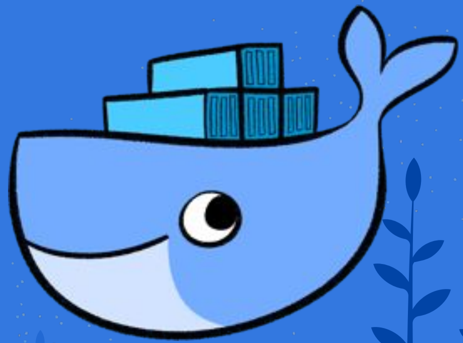


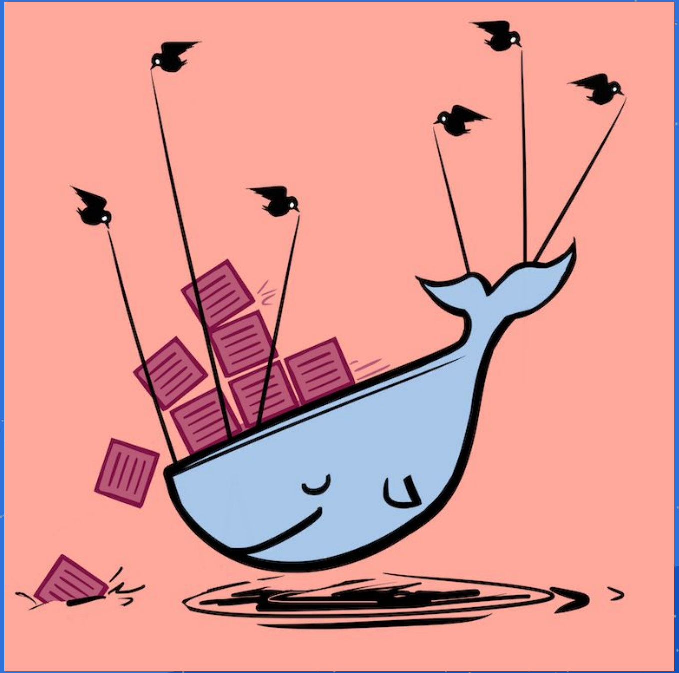
Containers



(aka Docker)

lemurseven, kian

Based on slides by… njha, jvperrin, kian, rjz



Art by Karen Rustad Tölva

Outline

- Containers vs Docker
- What is a container?
- Why use containers?
- How to use Docker
- What does a Dockerfile look like?
- How to orchestrate with Docker Compose
- Where does the OCF use containers?

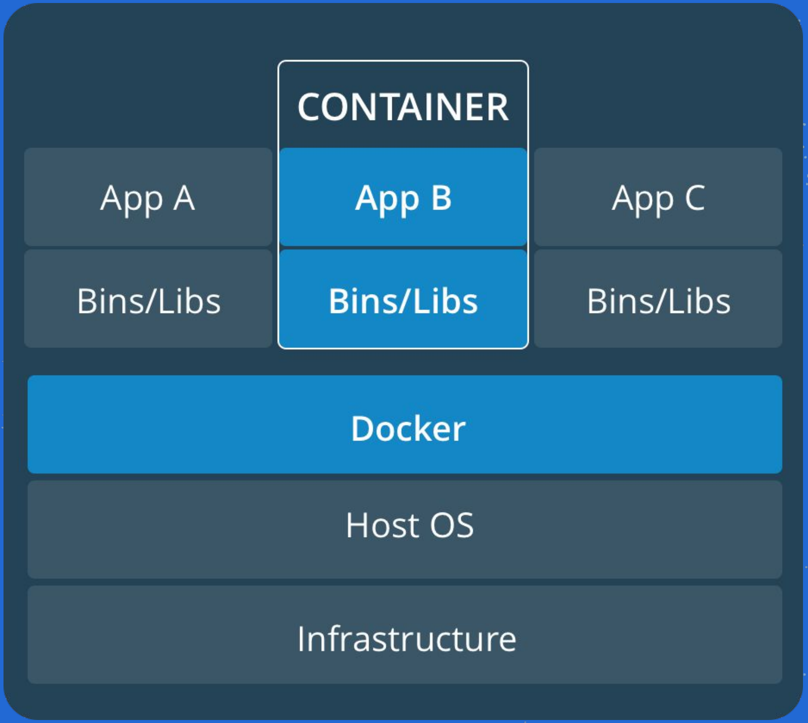
“Container” vs “Docker”

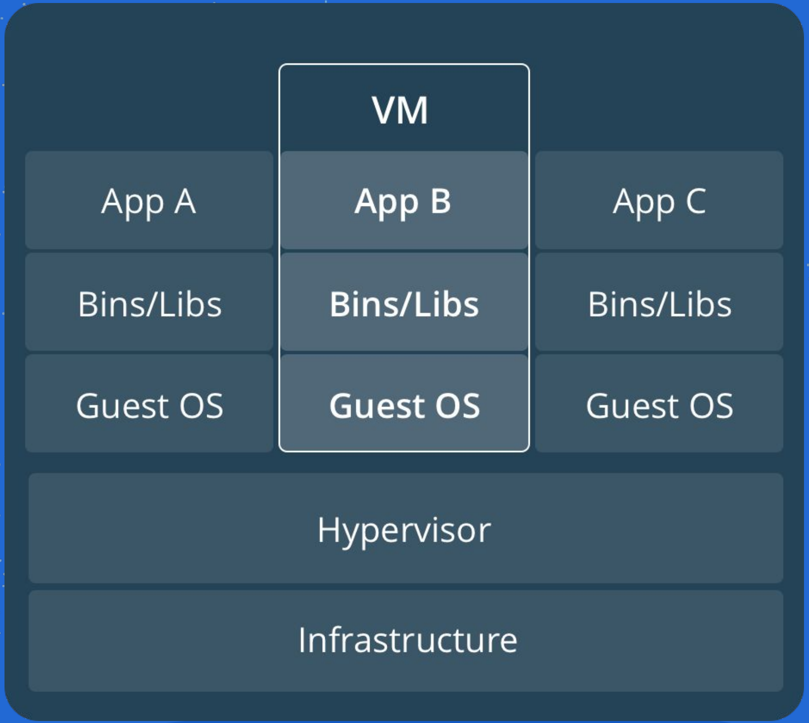
- “Docker” is a **trademark** owned by Docker, Inc.
- It’s the marketing name for their implementation of **containers**.
- Docker, Inc. widely **popularized** the idea of a container.
- An analogy…
    - Tissue Paper : Kleenex
    - Container Platform : Docker

What is a container?

- A packaged unit of software
- Runs the same on any* Linux distribution
- How is it implemented?
  - cgroups (allocate resources like CPU time, system memory to the container)
  - mount namespaces (filesystem isolation)
  - network namespaces
  - pid namespaces
  - … other types of namespaces

What is a container?





Is it like a Virtual Machine (VM)?

- Kind of, but not really.
- In Docker, each container is a set of processes running on the **SAME** kernel.
  - Separated by namespaces
- In a VM, you run many **DIFFERENT** versions of the kernel, and then run different applications on top of those kernels.

Is it like a Virtual Machine (VM)?

- You might start with “Ubuntu” inside a container, but this version of Ubuntu is missing a lot of services you might expect from a normal install of Ubuntu…
    - log rotation, systemd, cron, ntp, ssh, etc.
- Can’t you run Docker on Windows? Where’s the Linux kernel there?
    - It runs a Linux VM, and then runs all your Docker containers on that same VM. <- This is confusing so make sure you ask questions.

Pros and cons: Containers vs VMs

Containers

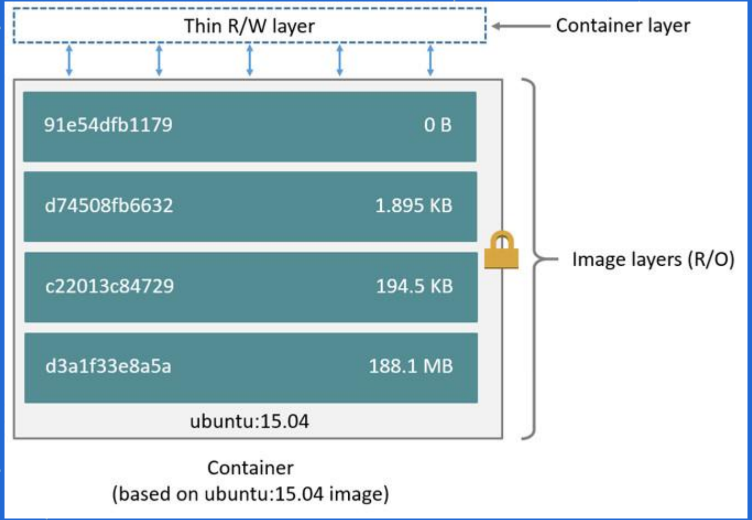
- Lightweight
  - Faster Startup
  - Smaller Images
  - Lower Resource Usage
- Easier to package and ship
- Cattle not Pets
- Local development

VMs

- Better Isolation
- Live Migration
- Can run non-Linux OSes

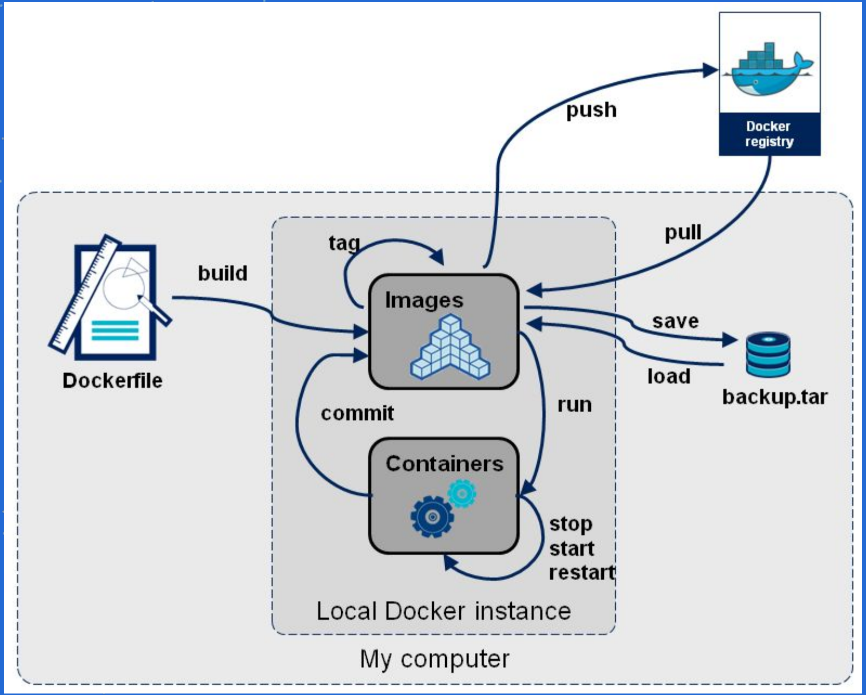
How does Docker even work?

- Layers, like an ogre
- Each step in a Dockerfile adds another image layer
- All layers can be combined into one to save space (docker-squash for example)
- This saves on network bandwidth and storage space when pushing images
- Also means it's easy to take a base image and add on top of it



What is an image vs a container?

- A container is an instance of an image
- Can have many containers running for one single image
- An image is a powered-off snapshot of a container's state



How to use Docker

- `docker images [REPO[:TAG]]`
  - Show images on the current machine
  - `docker rmi IMAGE` to remove images
- `docker ps`
  - Shows running images
- `docker logs CONTAINER`
  - Shows container logs
- `docker start CONTAINER`
- `docker restart CONTAINER`
- `docker stop CONTAINER`
  - These three are pretty self-explanatory, but notice that they all only work on containers, not images. docker run would have to be used for starting a container from an image first.

How to use Docker

- `docker build [-t NAME:TAG] PATH | URL`
    - Build an image, generally invoked as `docker build .` to use a local Dockerfile
- `docker history IMAGE:TAG`
    - Will show all the commands that were used to make up this image and how much size they took up on the resulting image
    - When building, often shows "Removing intermediate container" in the output, but even if the intermediate images are removed, the image layers, commands, and space they took up remain in the final image.

How to use Docker

- `docker run IMAGE [COMMAND]`
  - Run an image
  - Generally when running interactively (debugging, building stuff manually, etc.) you want the `-ti` options
  - `-v` is useful for adding volumes, `-p` is useful for adding port mappings
- `docker exec CONTAINER COMMAND`
  - Run a command in a running container
  - For instance, this is useful for debugging in production containers, or getting another shell on some container
  - Generally, a good command to use is `/bin/bash` (or some other shell that the container includes)
    - You'll want it to be interactive, so `-ti`

How to use Docker

- `docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]`
  - Essentially an alias for images so that other people can refer to it without having to type out `d4c75f89e842` or `496016f6ff53` for instance
  - If no hostname/port given, `registry-1.docker.io` is the default (the public docker image registry)
- `docker login [SERVER]`
  - Log in to a remote repository to allow for pushing images
- `docker push NAME[:TAG]`
  - Push a built image to a remote repository

What does a Dockerfile look like?

```
FROM node:18      # defines an image to base off of
WORKDIR /app      # set current directory (similar to cd)
COPY . .          # copy all files from local directory to image
RUN npm install   # run a command to install dependencies
CMD ["node", "src/index.js"]  # will be run upon startup
EXPOSE 3000       # tell docker that port 3000 accepts traffic
```

Multi-stage build

- Use multiple `FROM` statements to represent different stages of the build.
- Each stage can use its own base image and execute its own set of commands.
- With multi-stage build, you can create smaller images that only include the necessary dependencies and files to run an application, which also improves its security footprint.

But ...

what if I want to run multiple containers?

- You can always docker run everything, but that doesn’t scale well
- This is where container orchestration systems come in!
  - Kubernetes, Docker Compose, etc.
- Instead of running one-off commands, we can use files to define our desired setup
- Many benefits of orchestration:
  - Reproducibility
  - Networking between containers
  - Dynamically spread workload across multiple hosts

Orchestrate with Docker Compose

```
services:
  db:
    image: postgres:14
    volumes:
      - dbdata:/var/lib/postgresql/data
      - ./postgres.conf:/etc/postgresql/postgres.conf
    environment:
      POSTGRES_PASSWORD: password
  web:
    build: .
    command: bash -c "bundle exec rails s -p 3000 -b '0.0.0.0'"
    ports:
      - "3000:3000"
    depends_on:
       - db
volumes:
    dbdata:
```

Orchestrate with Docker Compose

Now that we have our docker-compose.yml defined, we can use it to manage our containers!

- docker compose up to bring up everything we defined in the file
  - We can add -d to "detach" and let it run in the background
- docker compose up <service name> to bring up a specific service
- docker compose stop to stop all containers
- docker compose start to start all containers
- docker compose down to stop and remove all containers
  - Will remove all non-persisted data so be careful!

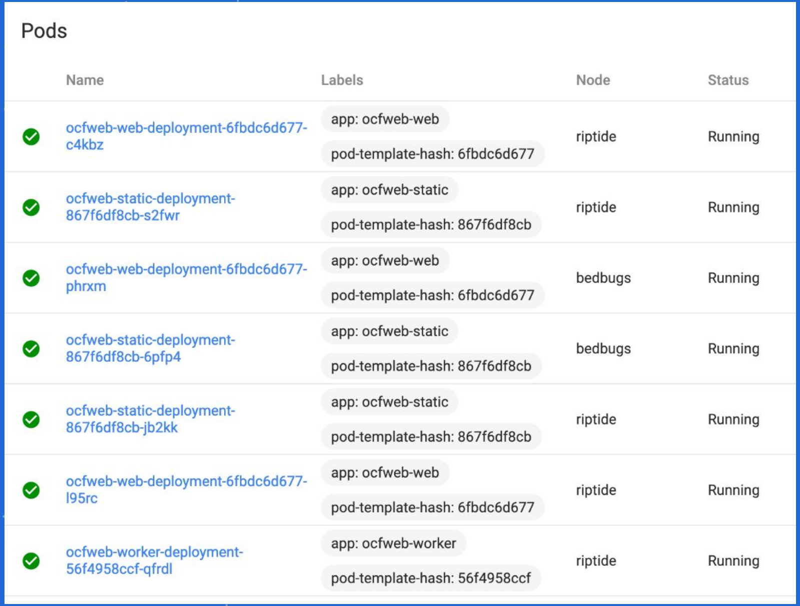
Orchestrate with Docker Compose

Here are some helpful commands when debugging your Docker Compose setup:

- docker compose ps to get the current status of all our containers inside a docker-compose project
- docker compose logs to tail logs of all our containers inside a docker-compose project
- docker compose logs <service name> to tail logs of a specific container
- docker compose -f <file> to specify a file other than docker-compose.yml

Where does the OCF use Containers?

- Primarily in our Kubernetes cluster to run our applications like ocfweb, rt, create, etc.
- Each service has one or more of its own containers, and then requests are proxied to it
- This is easily large enough to be the topic of its own talk



Pods

	Name	Labels	Node	Status
✓	ocfweb-web-deployment-6fbdc6d677-c4kbz	app: ocfweb-web pod-template-hash: 6fbdc6d677	riptide	Running
✓	ocfweb-static-deployment-867f6df8cb-s2fwr	app: ocfweb-static pod-template-hash: 867f6df8cb	riptide	Running
✓	ocfweb-web-deployment-6fbdc6d677-phrxm	app: ocfweb-web pod-template-hash: 6fbdc6d677	bedbugs	Running
✓	ocfweb-static-deployment-867f6df8cb-6pfp4	app: ocfweb-static pod-template-hash: 867f6df8cb	bedbugs	Running
✓	ocfweb-static-deployment-867f6df8cb-jb2kk	app: ocfweb-static pod-template-hash: 867f6df8cb	riptide	Running
✓	ocfweb-web-deployment-6fbdc6d677-l95rc	app: ocfweb-web pod-template-hash: 6fbdc6d677	riptide	Running
✓	ocfweb-worker-deployment-56f4958ccf-qfrdl	app: ocfweb-worker pod-template-hash: 56f4958ccf	riptide	Running

In this instance, each pod is running a single container.

Questions?