Docker:
Containers for application delivery

UNIVERSITY OF OREGON

NSRC
Network Startup Resource Center

Containers for software delivery

- Wouldn't it be great if:
  - Software environments were exactly identical between the developer's workstation, the test system, and the final production system?
  - Software was self-contained and had no dependencies on the underlying OS version?
  - Multiple applications with different requirements could all run on the same host?
- We can do this with containers!

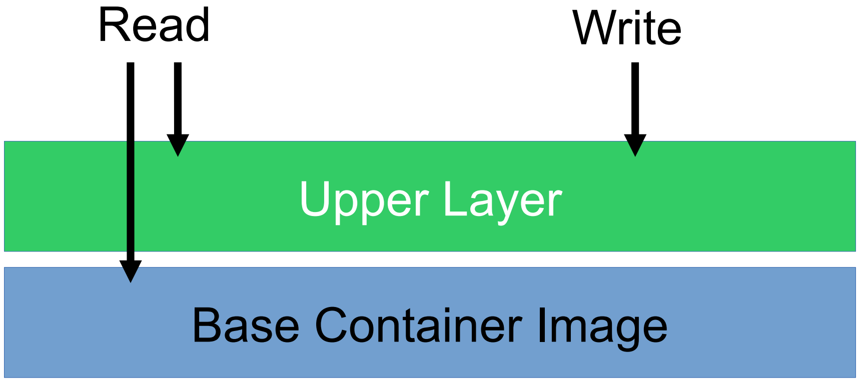
Quick aside on terminology

- We loosely refer to this as "docker containers"
- Docker pioneered the approach
- However, the container format is now standardized
  - OCI: Open Container Initiative
- There are different tools which work with it
- Docker is only one of those tools
  - (and it's a big, monolithic one)

The concept

- One container = one application or service
  - Keeps components isolated from each other
- Containers are always deployed from pre-built images (downloaded from a registry)
  - Container image contains the software and everything it depends on
- Containers are *not* managed like VMs
  - In particular, you never upgrade software within a container
  - You **destroy** the old container, and **create a new one** from a new image!

Filesystem layers



- The container has a *read-only* image as its base layer
- Any files it writes go into an upper layer (stores the differences)
- Multiple containers can share the same base layer

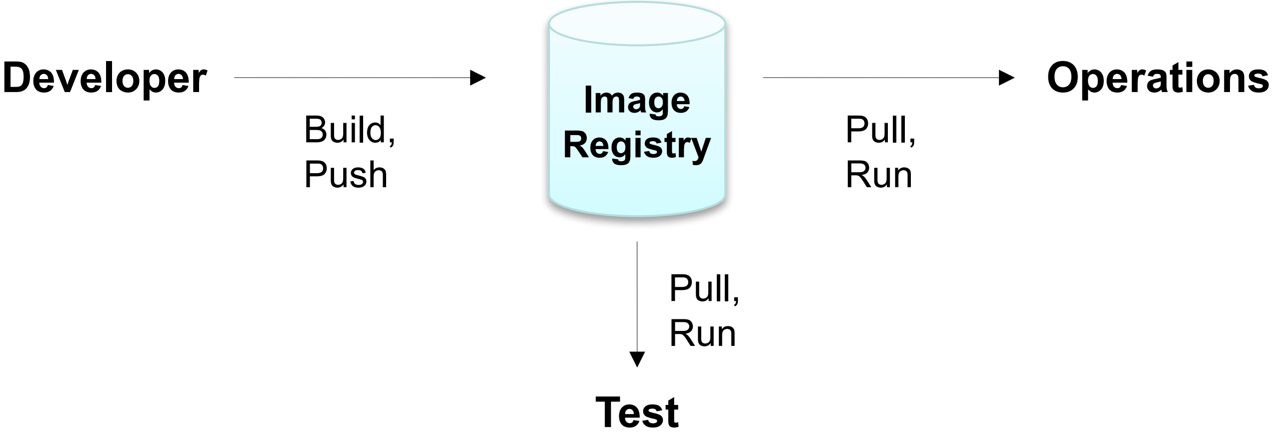
Stateless and Stateful

- Upper layer cannot be moved to a new container
- Ideally, containers are stateless
  - e.g. they access data in a remote database and do not store anything locally
  - Obviously it's safe to blow these away
  - It's also safe to run multiple instances for load sharing and redundancy
- Some containers need to be stateful
  - Store all important files in a specific directory
  - Mount a persistent "volume" at this location
  - When container is recreated, volume preserves existing data

What you need

- A way to **build** container images
  - This is the starting filesystem for a container
  - It contains the application and all its dependencies
- A way to **distribute** container images
- A way to **run** container images

Container workflow



- The registry ensures all users get exactly the same image
- Running as a container isolates it from everything else on the host
- Software runs the same everywhere!

Enabling new workflows: "gitops"

- Developer commits their code to git
- Automatically triggers build of container image
- If build successful, automatically triggers a run of all tests
  - CI: Continuous Integration
- If tests pass, automatically run in production
  - CD: Continuous Deployment
- Rapid, effective application development
  - And easy rollback to previous image if required

Common tools for containers

- *Running containers*: docker, podman (runc, containerd)
- *Building images*: docker, buildah
- *Distributing images*: CNCF Distribution*, docker hub, quay.io, github packages, AWS ECR, Sonatype Nexus, …
- *Orchestrating multiple containers*: docker compose, kubernetes
- *Gitops*: Argo CD, Flux CD, github actions, Jenkins X, …
- Docker still popular with developers. Production deployments are moving away from it

* Formerly known as Docker Registry

Using docker

- You can install docker directly on a Linux laptop or server
  - but beware that it messes with your iptables ruleset
  - anyone in the "docker" group effectively gets root on your machine ⚠️
- Safer to run it in a VM
  - you *might* be able to run it inside a container (with nesting=true) but some things may not work properly
- Docker Desktop*, Podman Desktop and Colima automate the creation of the VM in a convenient way
  - and let you communicate with docker engine as if it were local
  - attractive for developers, especially on macOS and Windows

* Docker Desktop is not free software, but is free to use in *some* cases

Building containers

- "Dockerfile" contains a series of script steps to build/install the application into a container
- This is a software development activity

```
# syntax=docker/dockerfile:1
FROM ubuntu:22.04

# install app dependencies
RUN apt-get update && apt-get install -y python3 python3-pip
RUN pip install flask==3.0.*

# install app
COPY hello.py /

# final configuration
ENV FLASK_APP=hello
EXPOSE 8000
CMD ["flask", "run", "--host", "0.0.0.0", "--port", "8000"]
```

Deploying containers in production

- Requirements:
  - Distribute containers across multiple nodes
  - Self-healing (failed node → redeploy containers)
  - Auto-scaling, redundancy, ...
- There were several competing solutions
- **Kubernetes** won out. The others all lost.
- Very powerful. Very modular. Very complex.
  - There are all-in-one distributions to get started with (k3s, k0s, microk8s, minikube)
  - And a range of frontends like Portainer, Rancher

Complexities

- Things that are likely to trip you up include:
- Persistent Storage
- Networking
  - Container to container communication (CNI)
  - Outside world to container (ingress, load-balancer)
  - TLS certificate deployment
- Applying different settings to different environments
- Resource allocation / limits
- …

Summary: Benefits

- Container images are pre-packaged and ready to run
- Reproducible: runs identically in production, development and test environments
- Relieves you of the need to install application dependencies in the host OS
- Run many different types of application in essentially the same way