What about you?

Credit

- https://robrich.org/slides/welcome-to-docker
- https://github.com/excellalabs/docker-workshop-1

Docker Workshop Rule

- Nice to meet you all
- Keep calm and try your best
- Always participate
- Never give up, Ask for help
- Give me feedback

Scaling your successful application

From monolith to pebbles

Common architectures or deployments

Small autonomous systems

Benefits

Selective scaling

Resilience against failure

Deployment of smaller pieces with lower risk

Alignment with organization

Hetero-geneous technology landscape

Composability

Replaceability

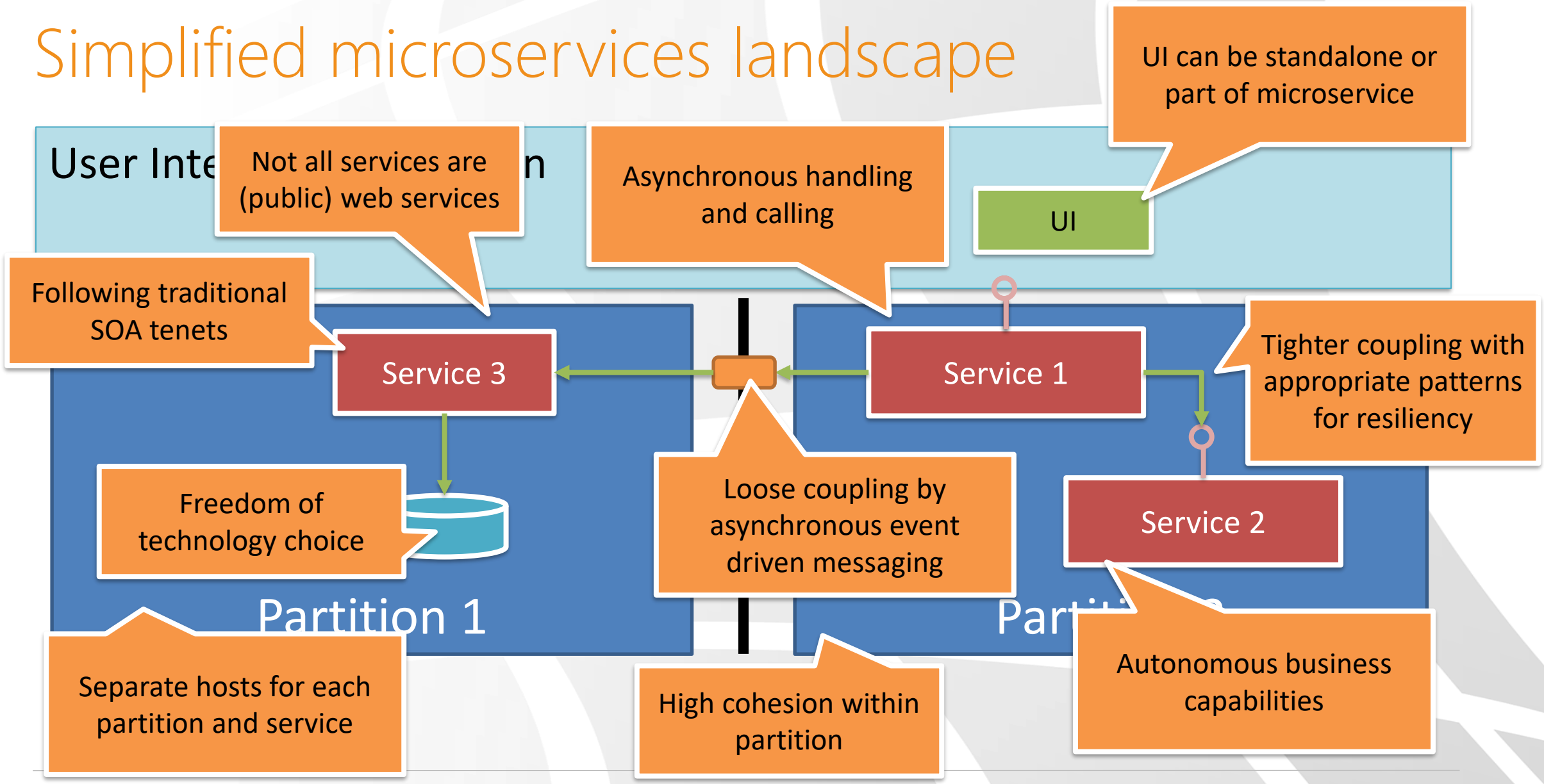
From horizontal to vertical

- Change your approach to vertical partitions

- Organized by logical cohesion
- Teams own layers spanning domains
- Unified technology choice

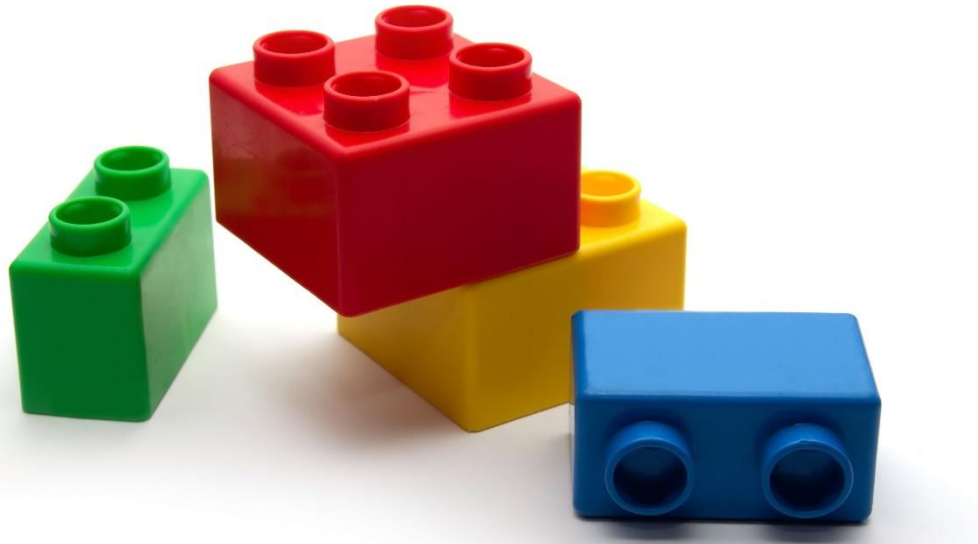
- Modeled after organization’s domains
- Owned by team
- Top to bottom
- Isolated from each other as much as possible

Simplified microservices landscape



Application architecture

- Featuring your favorite patterns
  - Event-driven or message-driven
  - CQRS with or without Event Sourcing
- Microservices or Monolith
  - Multiple, smaller out-of-process components is preferable for container technology



Why containers?

- Improved server density
- Improved recoverability of your infra structure
- Independent scale per container
- Easy to update
- Self-contained deployment packages
- Cluster orchestrators provide 0-downtime deployment capabilities
- Keep your infrastructure simple and clean

What is Docker?

Docker is an ecosystem around Container Virtualization

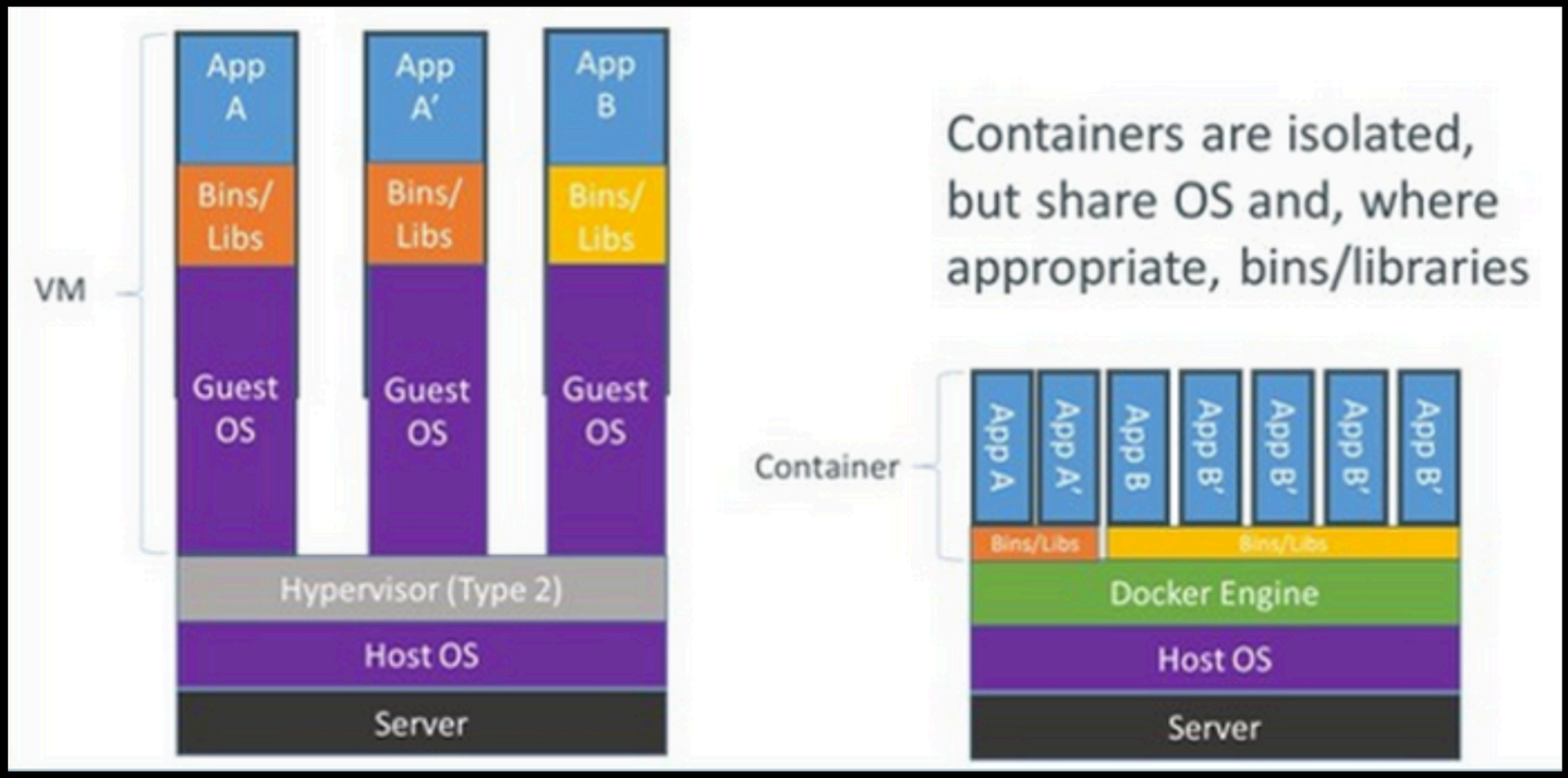
What are Containers?

Light-weight kernel virtualization

What is Docker?

A suite of command-line tools for creating, running, and managing containers

Containers vs VMs



Source: http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

Containers

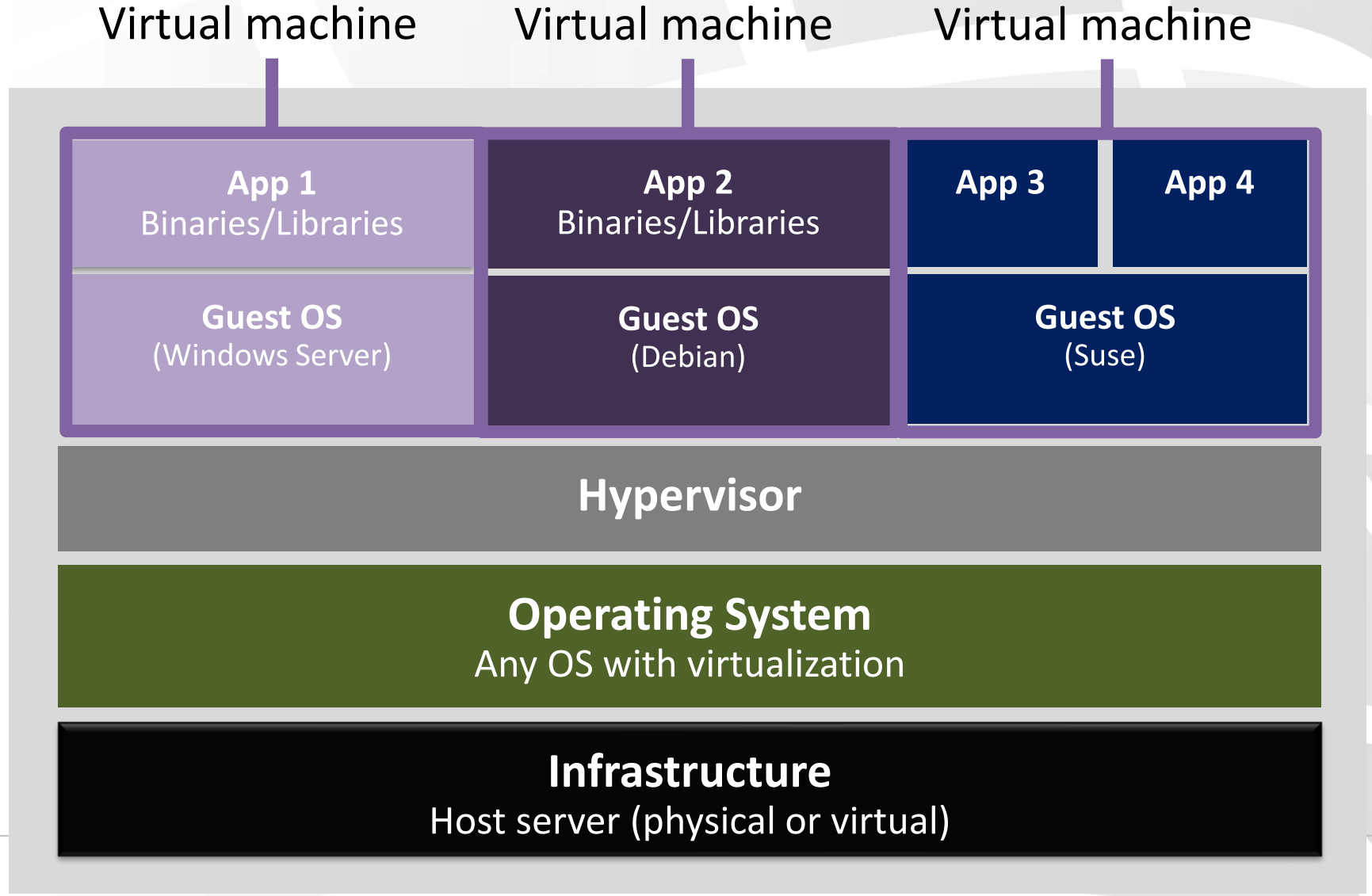
Containers virtualize and share the host kernel

Containers must run on the kernel for which they were built:

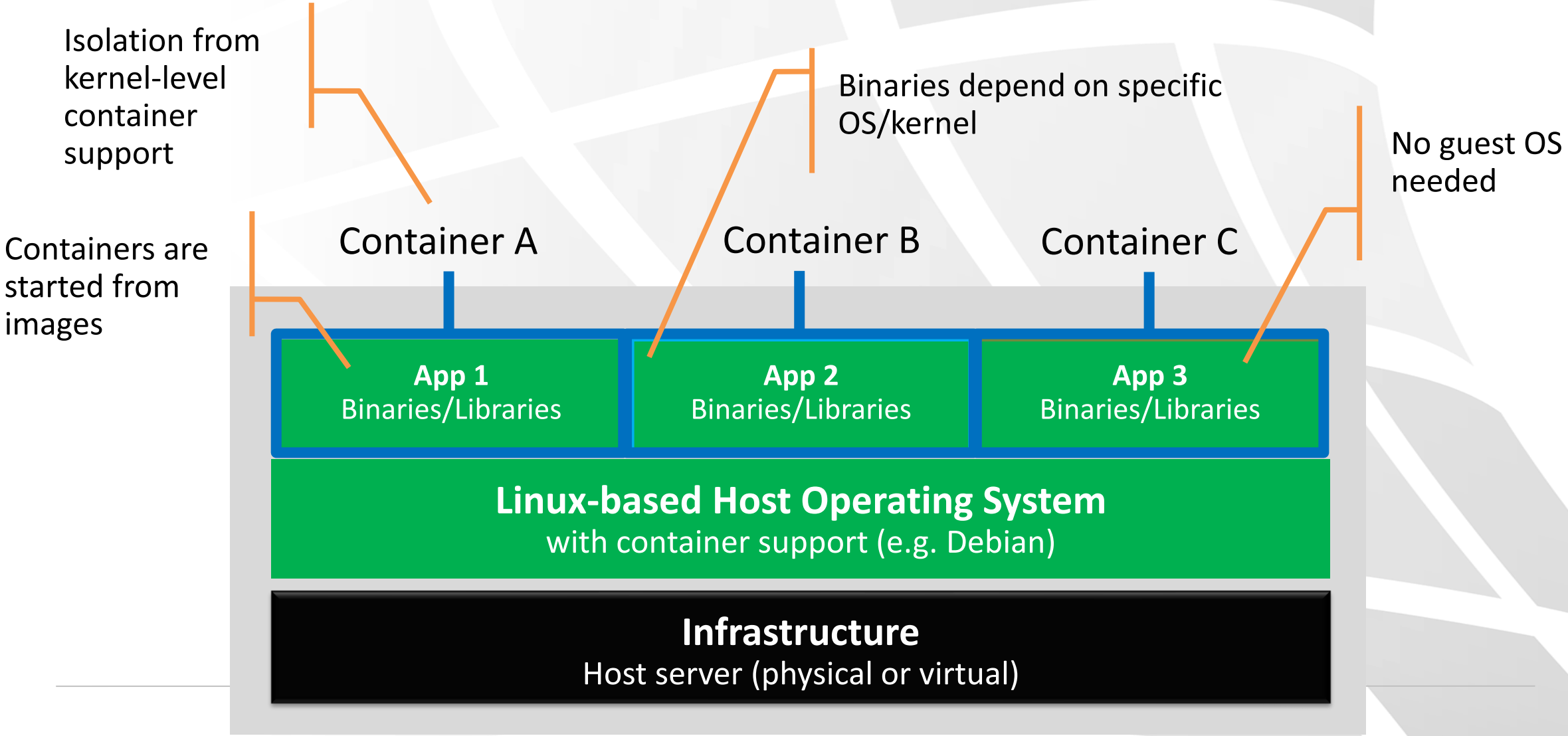
- Linux containers run on a Linux host
- Windows containers run on Windows Server host

From virtual machines

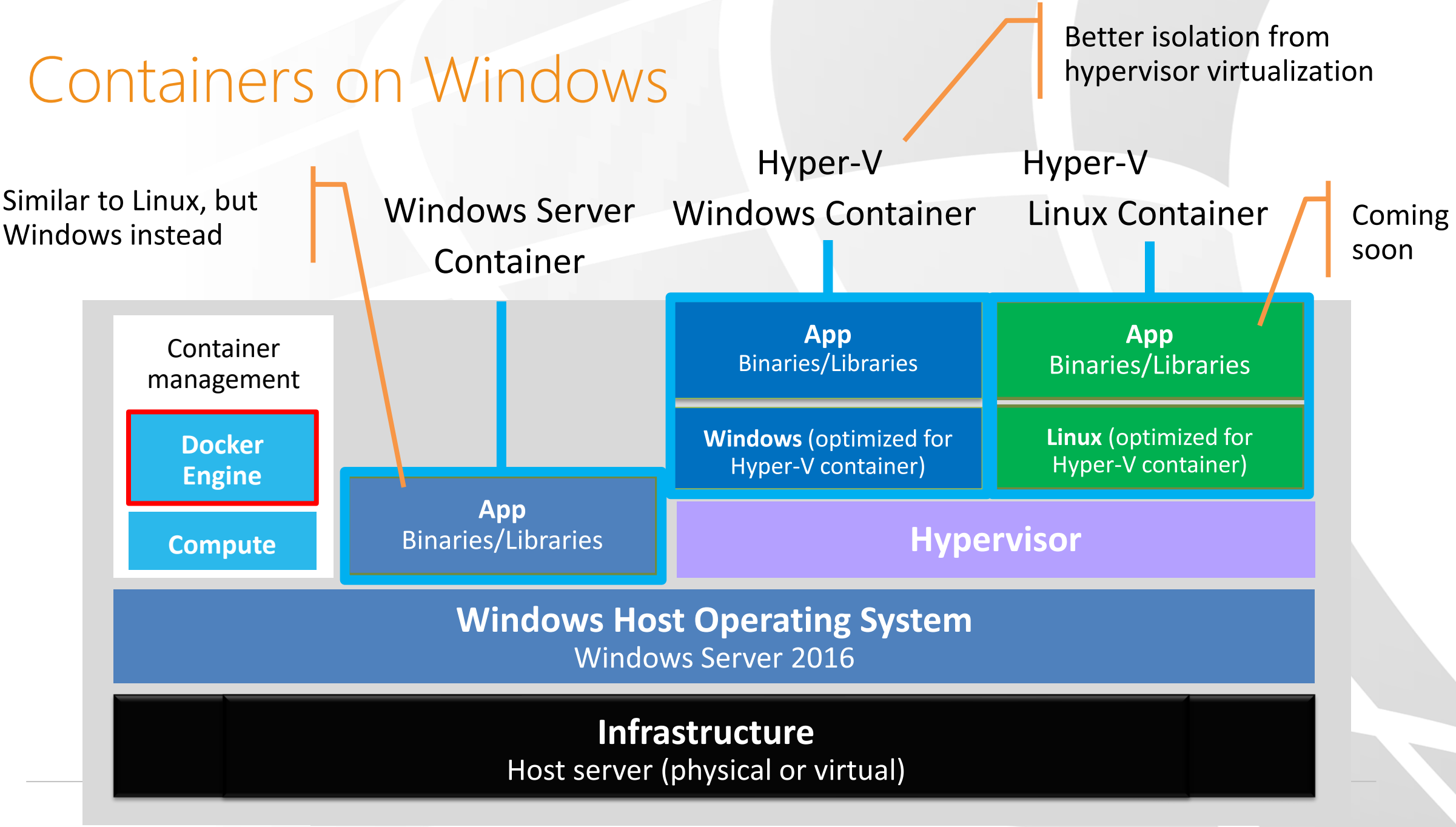
VM Images include binary and guest OS



Running containers on Linux



Containers on Windows

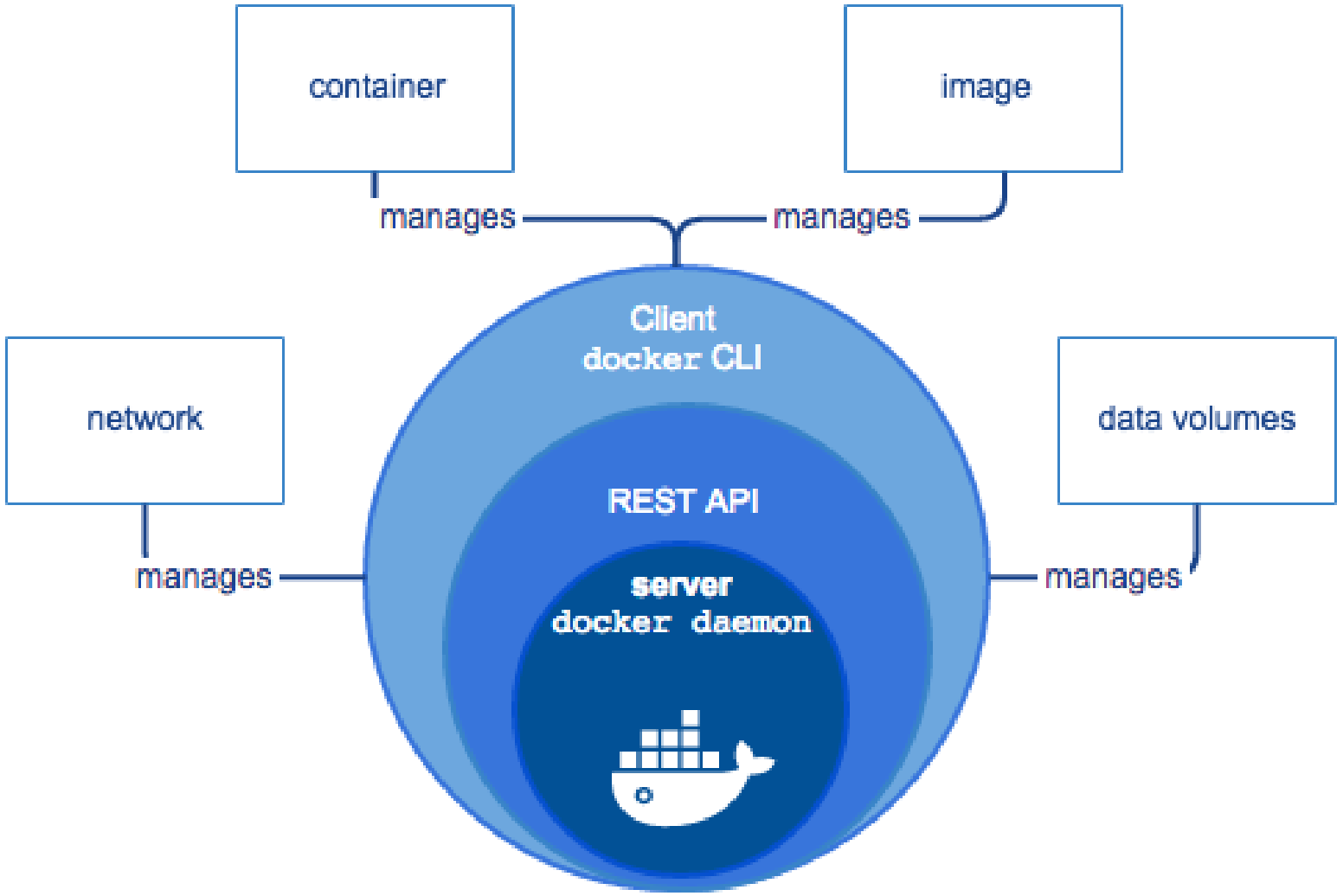


Docker and containers

- Standardized tooling
  - Container management
  - Image format
  - Across Windows and Linux
- Adopted for standalone and cluster scenarios
  - Docker for Windows and Mac
  - Several cluster orchestrators use Docker standards

Docker

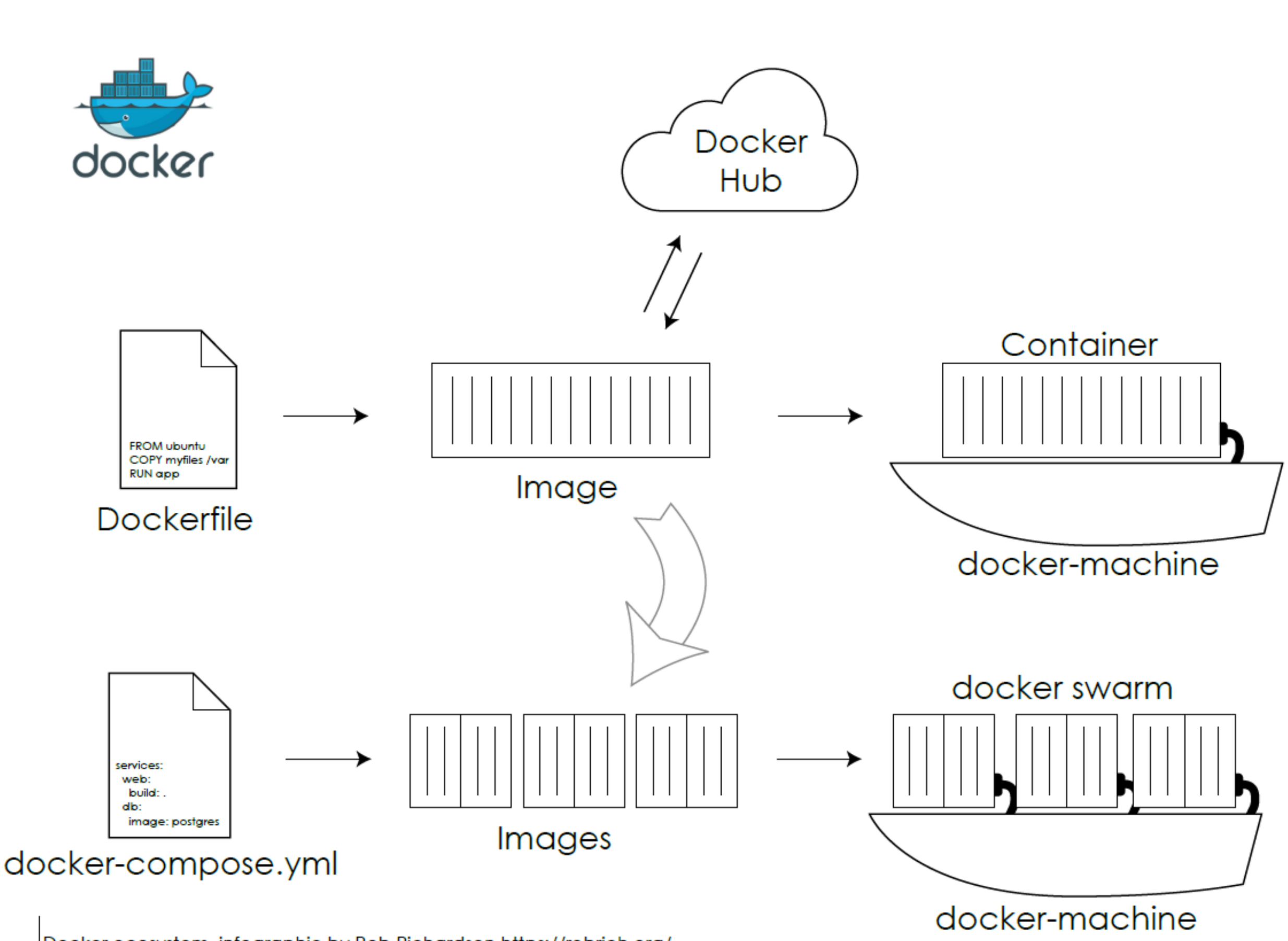
- CLI to control container images and instances
  - `docker pull …`
  - `docker run -it …`
  - `docker commit`
- Builds container images
  - `docker build`
  - `docker push`



Lab 1 – Docker 101

Lab


Docker Ecosystem



Docker ecosystem, infographic by Rob Richardson https://robrich.org/

Docker Machine

Tool for administering the docker hypervisor

- Linux: LXC (built-in)
- Mac: VM in xhyve
- Windows 10: VM in Hyper-V
- Windows Server 2016: Containers in Hyper-V

Docker Machine Drivers

Control a different hypervisor

`docker-machine -d thedriver ...`

- Amazon Web Services
- Microsoft Azure
- Digital Ocean
- Exoscale
- Google Compute Engine
- Generic
- Microsoft Hyper-V
- OpenStack
- Rackspace
- IBM Softlayer
- Oracle VirtualBox
- VMware vCloud Air
- VMware Fusion
- VMware vSphere

Source: docs.docker.com/machine/drivers

Docker Machine

- Create a Docker runtime environment:
  `docker-machine create default`
- Show docker runtimes:
  `docker-machine ls`
- Get environment variables:
  `docker-machine env default`
- Get the runtime's IP:
  `docker-machine ip default`

Docker Community Edition

- Automatically spins up a Linux VM
- Connects docker command-line to the VM
- Proxies localhost traffic to the VM's containers
- ... replaces all of the `docker-machine` commands

Dockerfile - configuration as code

```
FROM ubuntu:14.04

RUN curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
RUN apt-get update
RUN apt-get install -y nodejs

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY package.json /usr/src/app/
RUN npm install
COPY . /usr/src/app

EXPOSE 3000
CMD [ "npm", "start" ]
```

Dockerfile - configuration as code

This is broken, see building node nanoserver image

```
FROM microsoft/windowsservercore

ENV NODE_VERSION 6.2.0
RUN powershell -Command \
    wget -Uri https://nodejs.org/dist/v%NODE_VERSION%/node-v%NODE_V
    Start-Process -FilePath msiexec -ArgumentList /q, /i, node.msi
    Remove-Item -Path node.msi

RUN mkdir -p \app
WORKDIR /app

COPY package.json /app
RUN npm install
COPY . /app

EXPOSE 3000
CMD [ "npm", "start" ]
```

Dockerfile

- `FROM`: what's the base image?
- `ENV`: set environment variable
- `RUN`: run setup content
- `COPY`: copy content into the container
- `EXPOSE`: document inbound ports
- `CMD`: the thing to start

Dockerfile to Image

- build dockerfile into image:
  `docker build .`
- list images:
  `docker image list`
- remove an image:
  `docker image rm myimage`

Dockerfile - configuration as code

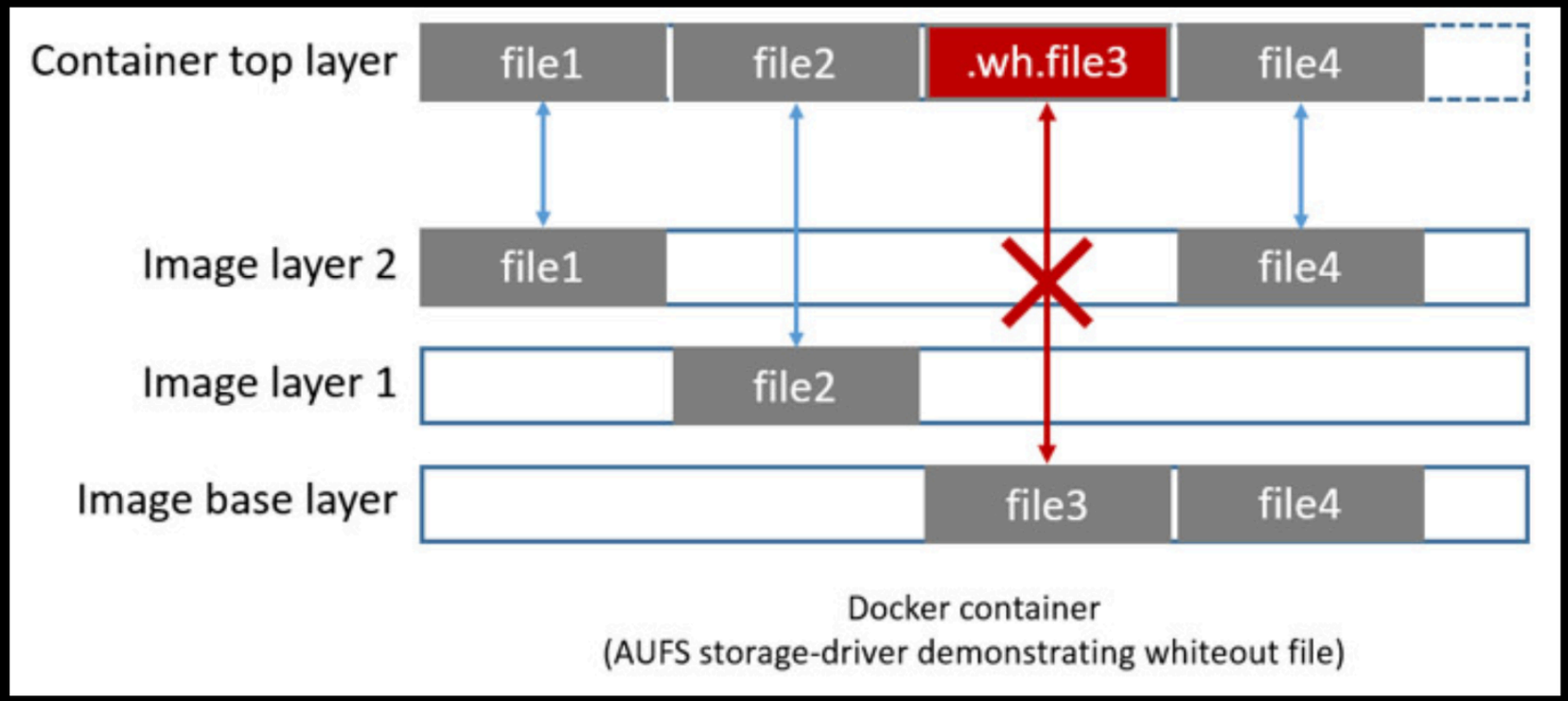
```
FROM node:alpine

WORKDIR /app
COPY package.json .
RUN npm install
COPY . .

EXPOSE 3000
CMD [ "npm", "start" ]
```

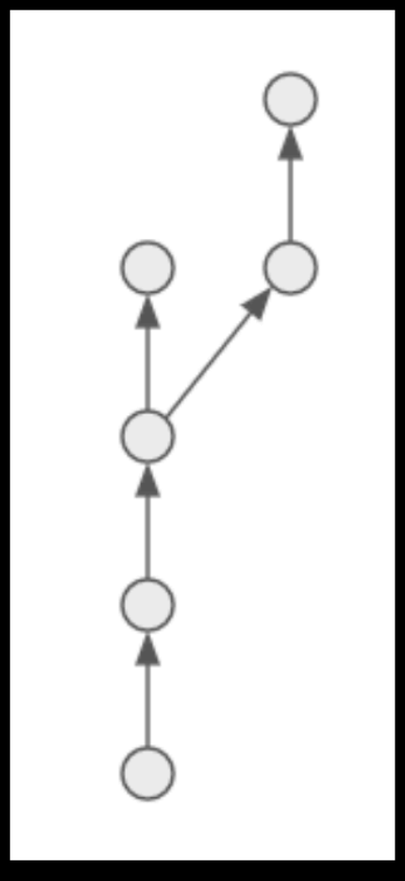
Lab

Layered Filesystem



Source: https://docs.docker.com/engine/userguide/storagedriver/aufs-driver/

Layered Filesystem



Only downloads each layer to disk once

because layers don't change

Volumes - non-temporary storage

Volumes are a pointer in the container
saved to a folder on the host

Volumes

In Dockerfile:

```
VOLUME ["/data"]
```

Starting container

```
docker run -v $(pwd):/data imagename
```

Run Container

`docker run imagename`

run the image as a container
on the current docker machine

Run Container

What did it do?

- download image layers (if needed)
- start process for container
- assign virtual ip to container
- NAT traffic into container

Rule of Thumb

1 container is 1 process

Customize runtime: ports

```
docker run -p 80:5000 imagename:latest
```

Port 80 outside maps to port 5000 inside

Customize runtime: volumes

```
docker run -v $(pwd):/data imagename
```

Map the current directory outside to the `/data` volume inside

Customize runtime: start command

```
docker run imagename /bin/bash
```

Start the normal process then run bash in the terminal

Run Containers

- `docker run -args imagename cmd`
  run the image as a container on the current docker machine
- `docker container list`
  show running containers
- `docker container logs containername`
  get a container's output
- `docker container stop containername`
  power off a container
- `docker container rm containername`
  remove a container

Prod & Dev Strategies

Use containers in creative ways:

- prod container exposes config as environment variables
- dev container has volume mapped to local directory

Many containers running in concert

Features:

- Containers alone in private network
- Only cluster's public ports are available to the outside
- Service discovery so containers can find each other
- Service management to start / scale cluster

Cluster Choices

- Docker Swarm
- Kubernetes: built by Google
- Apache Mesos: most proven at scale
- others ...

Service Discovery Choices

... then get a service discovery store:

- Consul
- Etcd
- ZooKeeper

Source: https://docs.docker.com/swarm/discovery/

docker-compose.yml

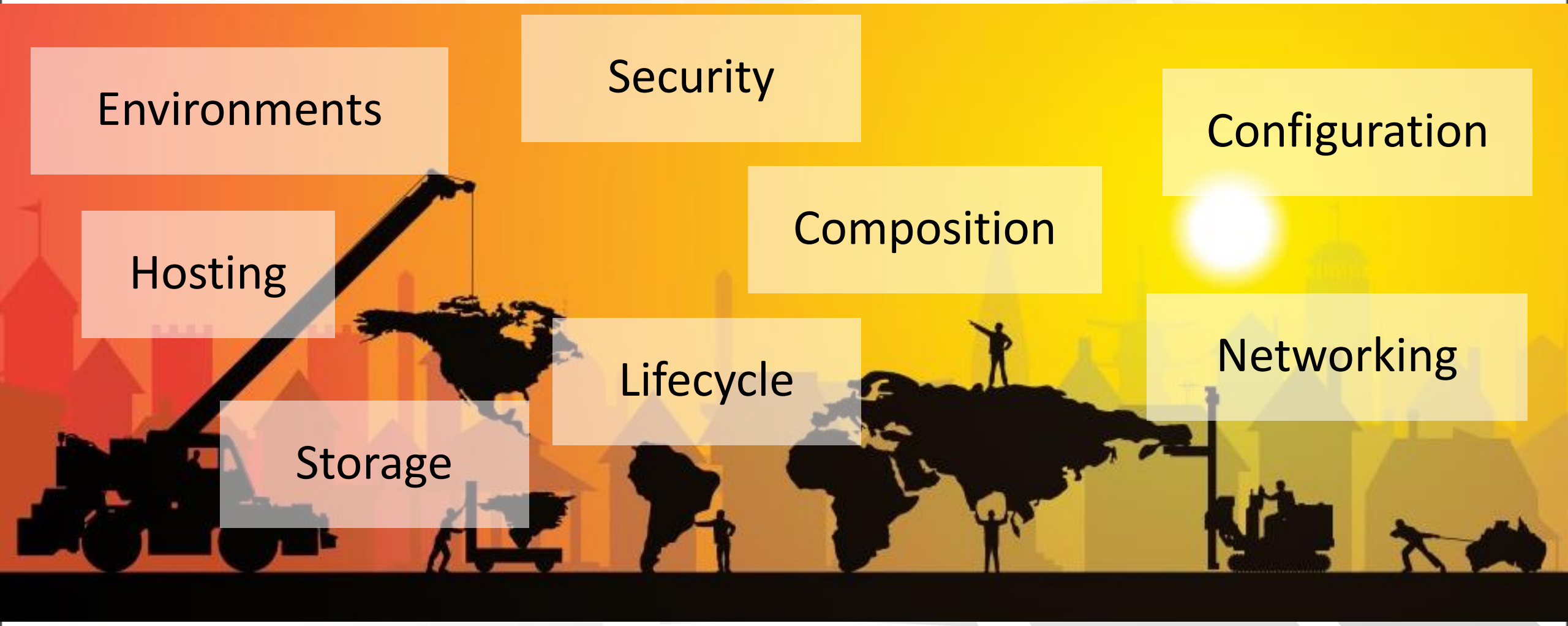
```
version: '3'
services:
  web:
    build: .
    ports:
    - "80:5000"
    volumes:
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Source: https://docs.docker.com/compose/overview/

`docker-compose` commands

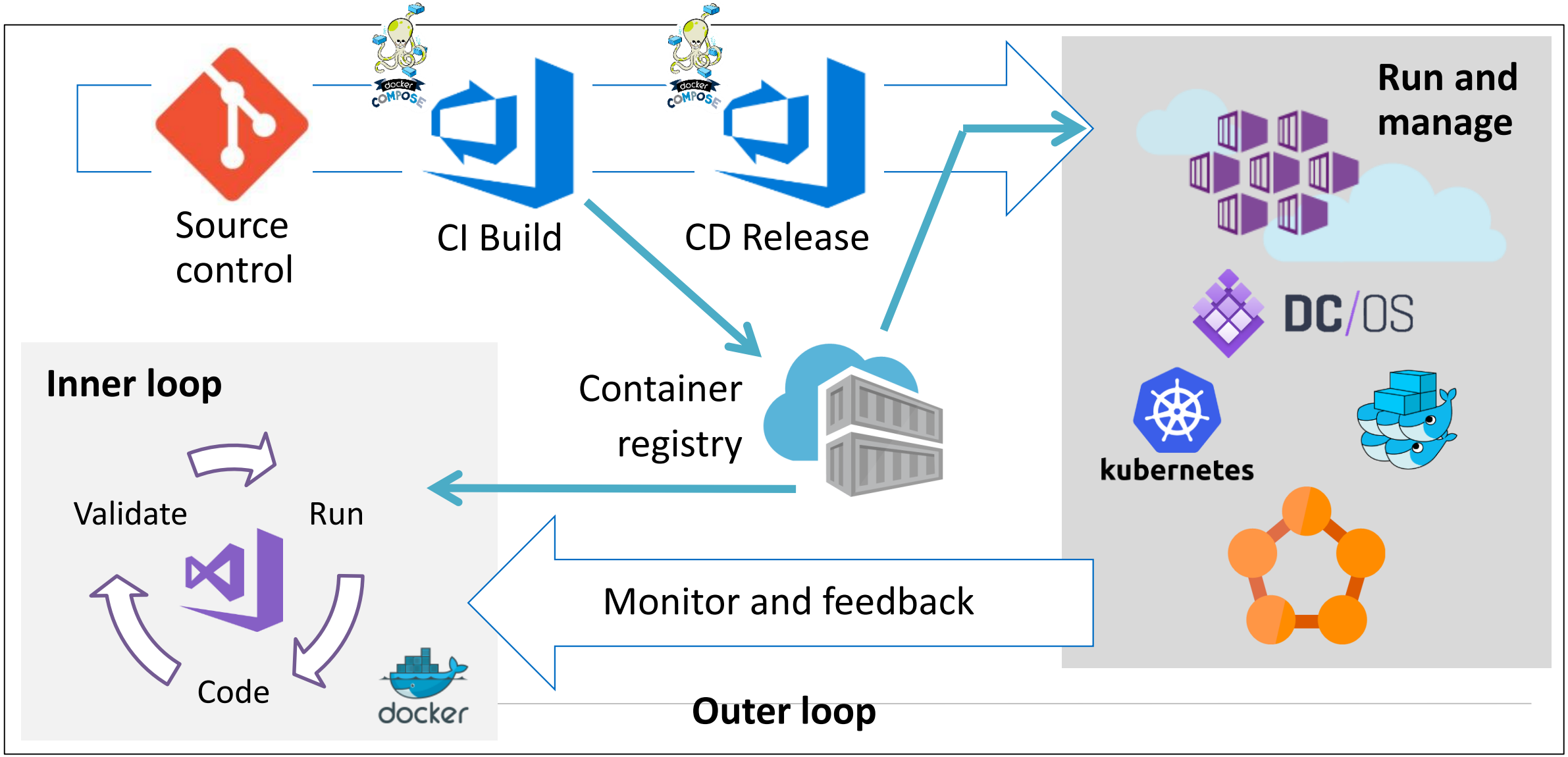
- `docker-compose build` - Build all the images
- `docker-compose run` - Run a swarm
- `docker-compose up -d` - Build and start the swarm
- `docker-compose down` - Stop the swarm
- `docker-compose ps` - List containers in the swarm
- `docker-compose rm` - Remove stopped containers
- `docker-compose logs` - Show containers' logs
- `docker-compose scale web=2 worker=3` - run multiple containers

What changes when switching to containers?



Container application lifecycle

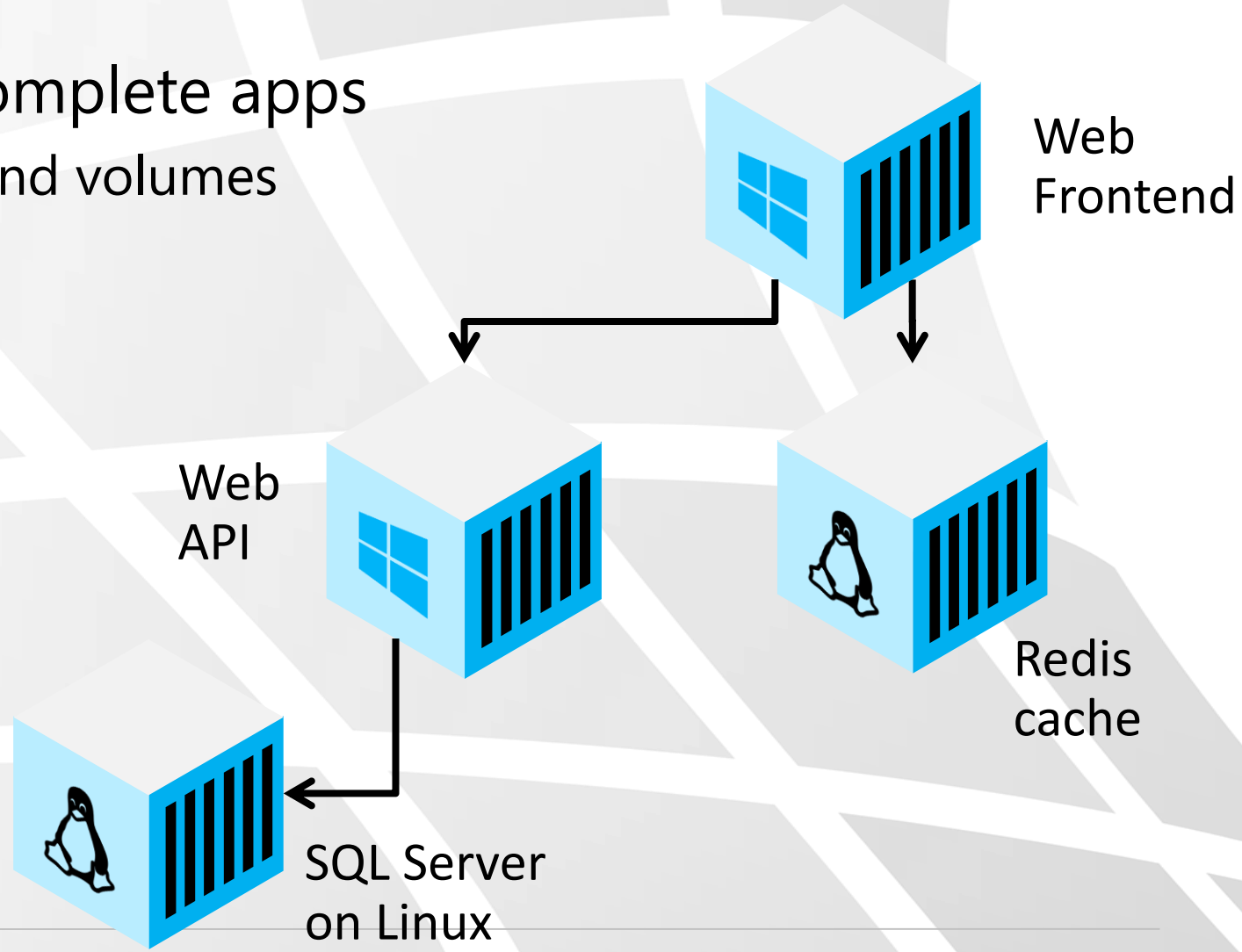
Container workflow and lifecycle



Composing applications

Composing your .NET application

- Combine containers to complete apps
  - Define services, networks and volumes
- Different compositions per environment
  - Standalone laptop
  - Production cluster

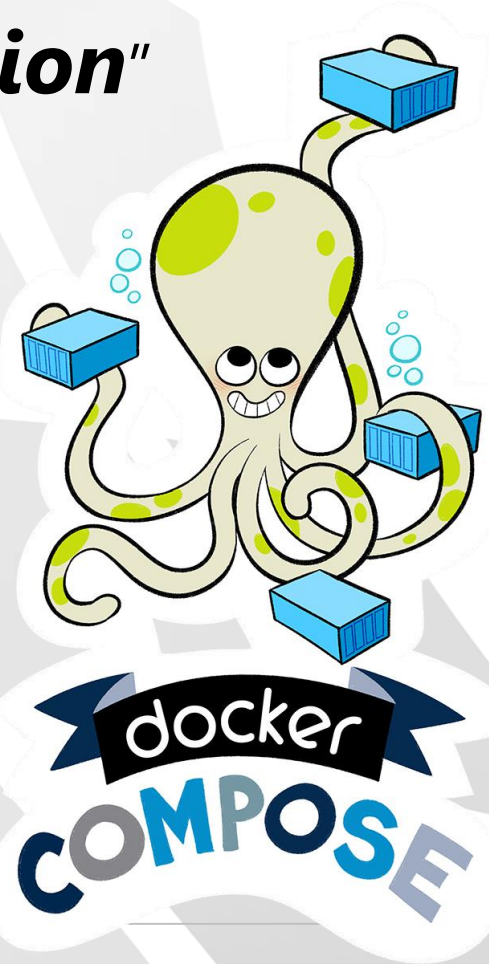


Composing container solutions

Docker-compose:

"***Orchestration tool for container automation***"

- Single command: `docker-compose`
  - Work with multiple containers: build, run, scale, heal
  - Scoped mostly at single-host scenarios
    (Use clusters for multi-host)
- YAML files describe composition
  - Configuration file for images, build,
    services, volumes, networks, environments
  - Same syntax for deploying on clusters (version 3.0+)
  - Allows hierarchies and overriding



Docker-compose structure

```
version: '3.0'
services:
  service-name:
    image: docker-image
    build: how to build
    depends_on:
      - other services
    environment:
      - key/value pairs
    ports:
      - port mappings
```

```
networks:
  network-name:

volumes:
  volume-name:
```

Lab

Docker Benefits

- denser hardware utilization
- faster spin-up time
  how long does it take to boot Windows vs start Sublime?
- smaller file size
- share image layers
- identical (or nearly) dev / prod configuration
- easily share setup

Docker Drawbacks

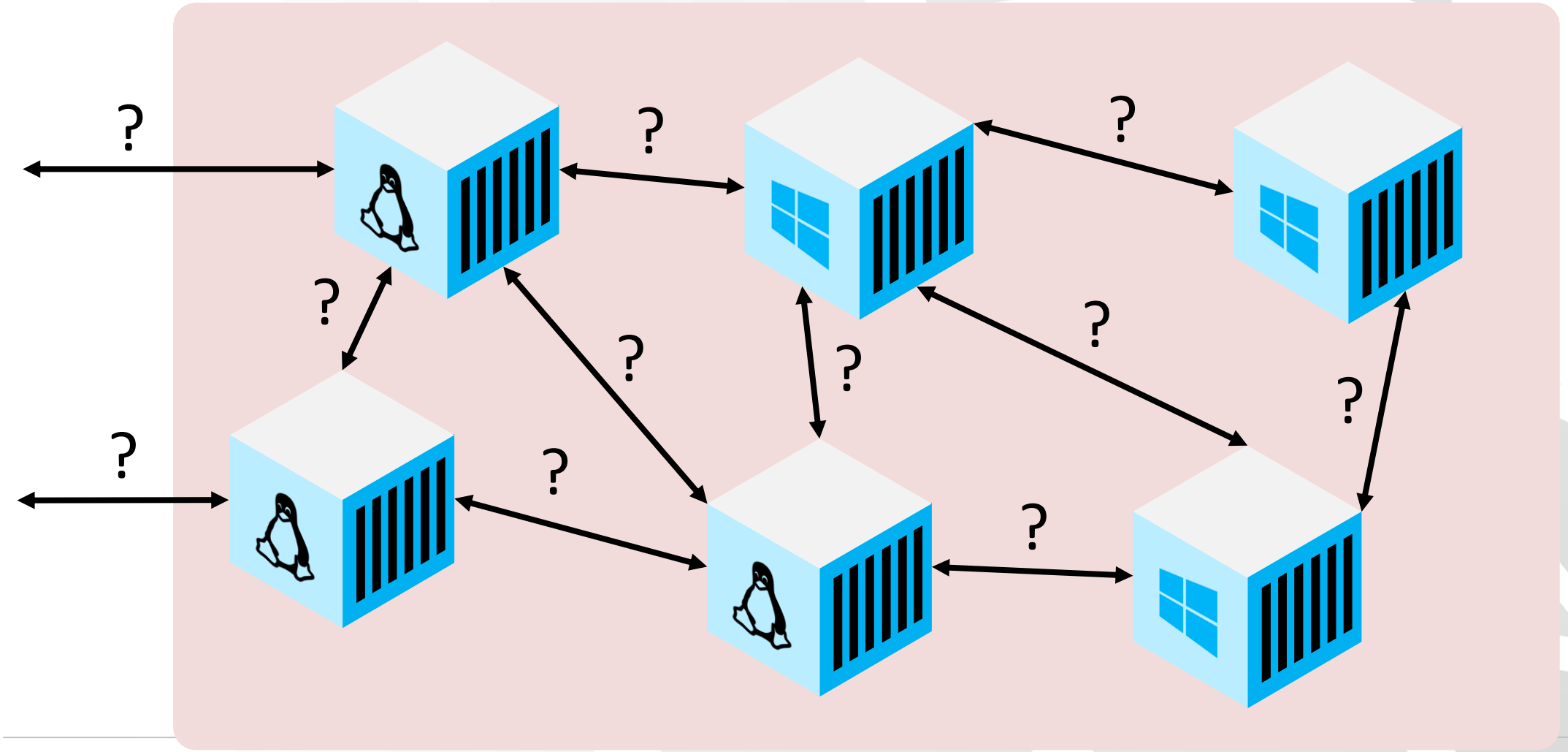
- less isolation
  - still in a sandbox
  - no one has broken out yet
- kernel dependence
- no UI, command-line only
- it's IaaS, PaaS may be better

Best Practices

- combine commands into one line makes less layers
- use SaaS data store - eliminates statefull containers
- 1 process per container
- container with volume for local development
- expose config as environment vars for production
- store secrets carefully
- use docker-compose as simpler command launch
- use docker-compose networks as isolated "virtual lan"
- rebuild images frequently to get latest OS and dependency patches

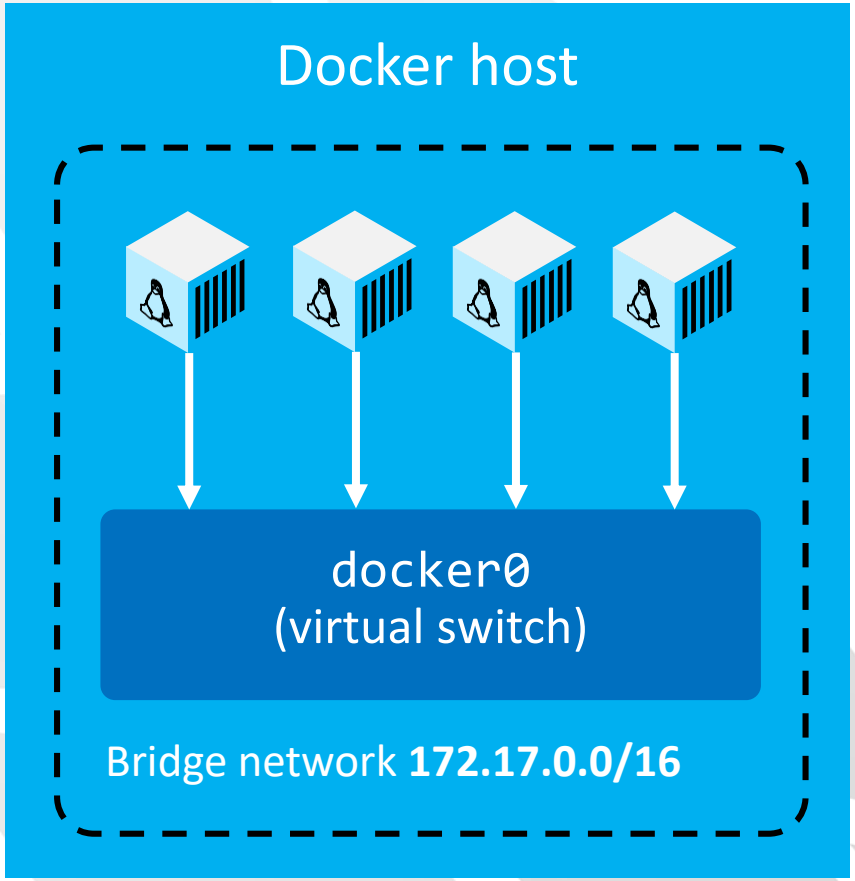
Networking

Connecting containerized services

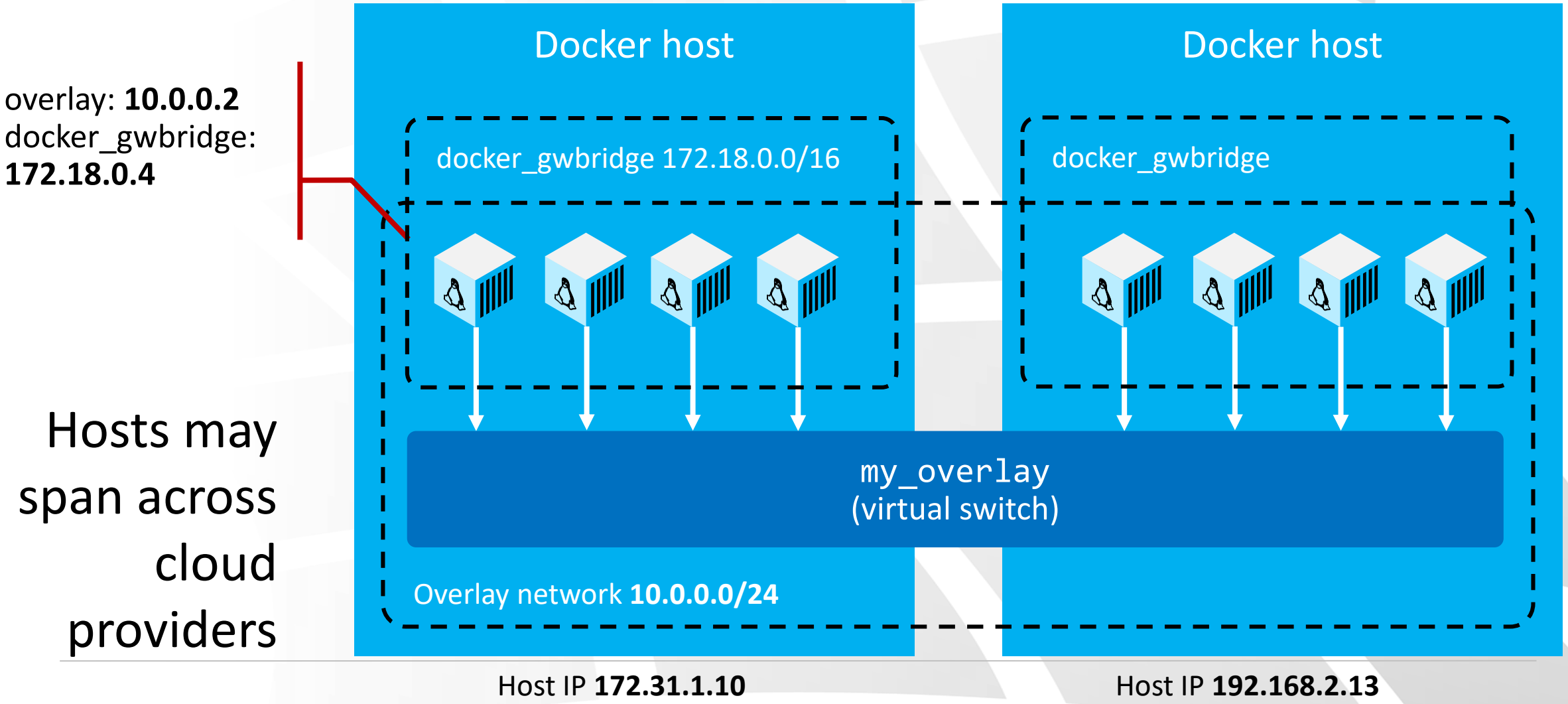


Docker networks

- Docker containers in same network are connected
  - All connected on all ports
  - Applies to single- and multi-host networks
  - To outside network connections are firewalled
- Drivers determine network type
  - **Host**: Shares IP address with host. Must map to unique ports
  - **Bridge (Linux)**: Default single host network
  - **NAT (Windows)**: Default single host network
  - **Null**: No network
  - **Overlay**: Multi-host networking



Docker networks: overlay in swarm



Hosts may span across cloud providers

Exposing containers

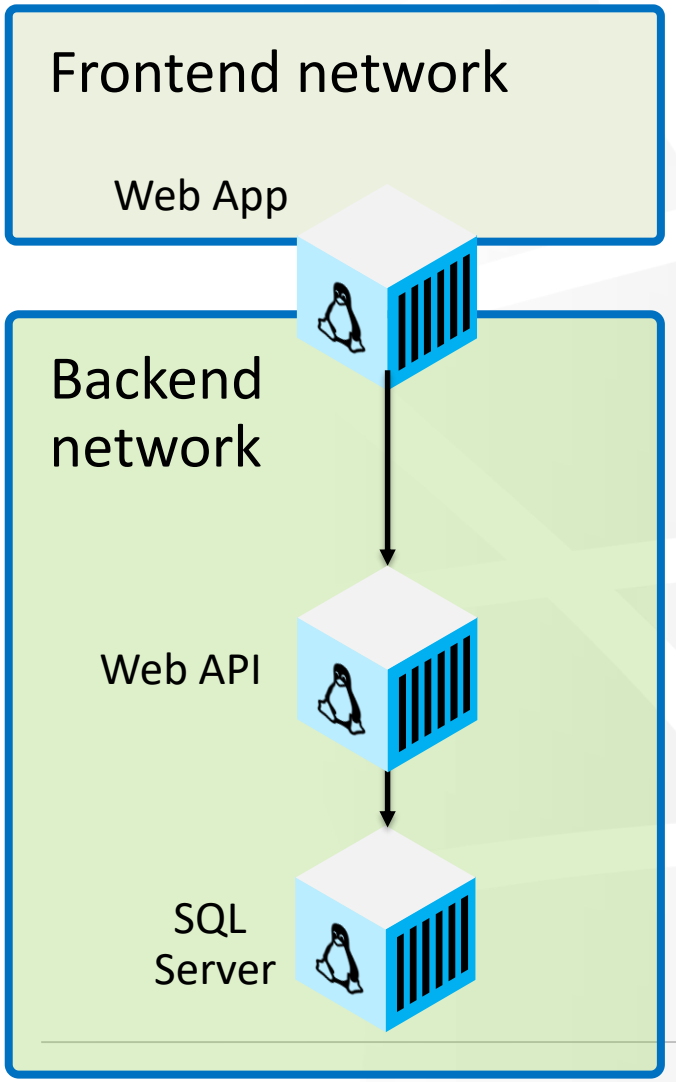
- Port publishing
  - Containers must publish ports for inbound traffic
  - Outbound traffic is masqueraded to ephemeral ports (32768-60999)
  - `docker run -it --publish 5000:80 …`

192.168.0.2:5000
External

10.0.0.2:80
Internal

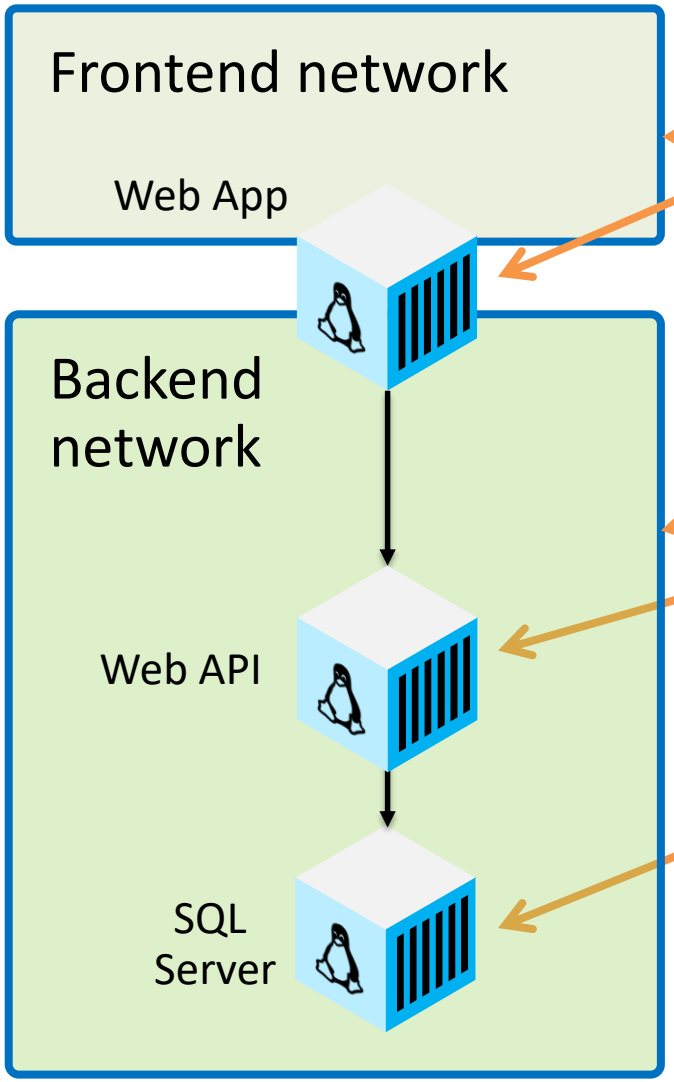
- Built-in DNS
  - Embedded DNS service maintains mapping service and IP address
  - Other services reachable by compose service name
  - Reflected in URI connection endpoints
  - Can forward to external DNS

Composing networks



- By default single network is created
  - When creating a composition or stack
  - *compositionname*_default
- Create user-defined networks for network isolation
  - Bridge: single host
  - Overlay: cluster
- **Note:**
  Overlay does not allow manual adding to network
  Use docker service instead of docker run

Docker compose: networking




```
services:
  gamingwebapp:
    networks:
      - frontend
      - backend

  leaderboardwebapi:
    networks:
      - frontend

  sql.data:
    networks:
      backend:
        aliases:
          - sql1.seladeveloperpractice.local

networks:
  frontend:
    driver: overlay
  backend:
    driver: overlay
```

Lab 3 – Networking

Lab