Docker

Docker Containers are not VMS

- Easy connection to make
- Fundamentally different architectures
- Fundamentally different benefits

Virtual Machines (VMs)

- Fully self Contained with dedicated resources
- Stand Alone
- Large Size
- Several Minutes to start up

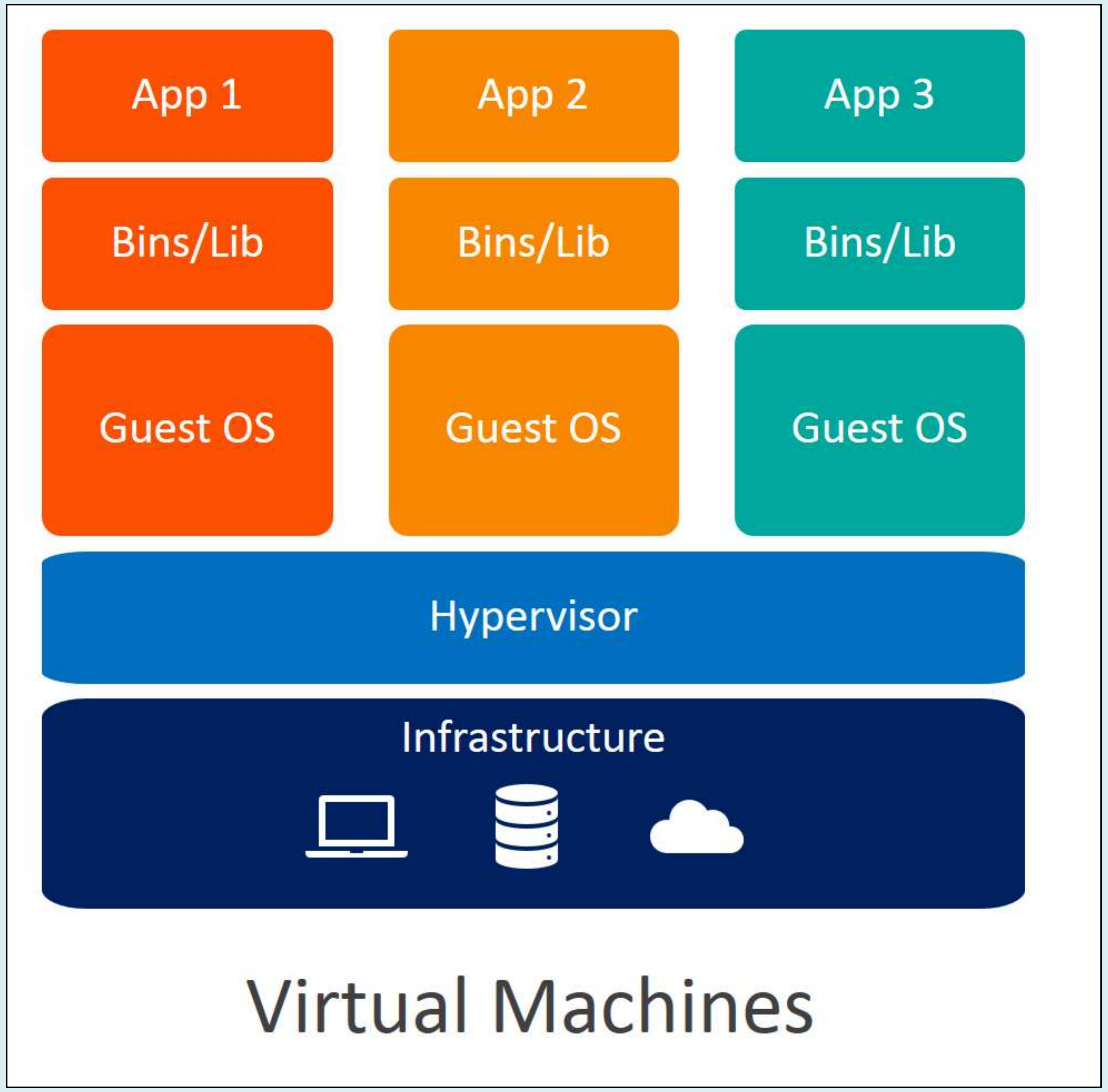
docker

Containers

- Shared resources
- Shared Host – The server running Docker Daemon
- Each Container can be of different sizes
- Contain only what they want
- Smaller size
- Faster Start up time (in seconds)

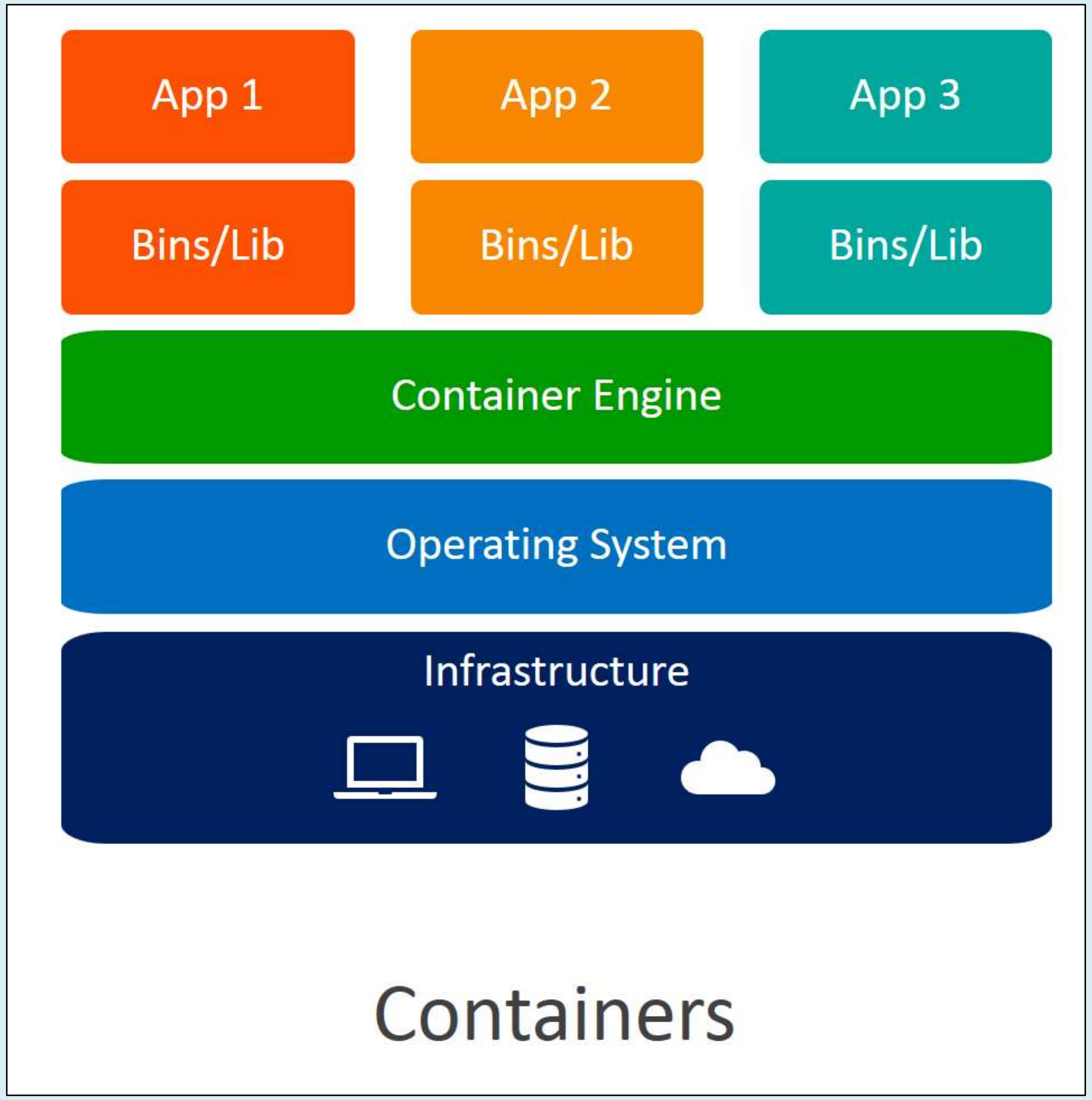
Virtual Machines

- Run on a hypervisor.
- A hypervisor emulates the physical hardware.
- Type 1 and Type 2 Hypervisors
- A VM running on a hypervisor appears as if it has its own CPU, memory and resources.



Containers

- Are not Virtual machines, since there is no hypervisor involved.
- Isolation is provided by a container engine.
- Container Engine uses some specific features of the Linux operating system implement isolation.



Containers

- A logical way to package applications
- Provides isolated execution environments
- Composable
- Portable

```
# Example Dockerfile
FROM ubuntu
MAINTAINER asbilgi@microsoft.com

RUN apt-get update
RUN apt-get install –y python
CMD [/bin/bash]
```

Containers

- Use features of the Linux operating system to implement isolation.
- If you know these Linux Operating System features, you can write your own container runtime, like docker, containerd, rkt, podman, etc.
- Even if you do not plan to write your own container runtime, it is good to know how containers work.

Containers for the curious

- Unix File Systems & buildroot
- OverlayFS
- chroot
- namespaces
  - Process
  - User ID
  - UTS
  - Mount
  - Cgroups
  - Network
  - IPC

Containers

- docker

RUNC

containerd

rkt

Service Providers

rackspace the open cloud company
Google Cloud Platform
amazon web services
SOFTLAYER
openstack CLOUD SOFTWARE
OPENSHIFT
Microsoft Azure
DigitalOcean
CLOUD FOUNDRY
Google Compute Engine
stackato by ActiveState
Baidu 百度
Yandex
Google App Engine
StackDock
Quay.io
linode
Gilliam
DEIS
tutum
ORCHARD
domino
Flynn
Baremetal Industries
VOXOZ
clever cloud

Operating Systems

redhat
boot2docker
CoreOS
ubuntu
debian
fedora
CentOS

Configuration Management

puppet labs
Chef
Jenkins
SALTSTACK
Capistrano
Packer
ANSIBLE

Big Data

OpenCore
hadoop
INTEGRATING BIG DATA... SMARTER syncsort

Service Discovery

NETFLIX Eureka
etcd
SkyDNS

docker

Dev Tools

Shippable
Fig
VAGRANT
heroku
wercker
drone.io
runnable beta
NODECHECKER

Official Repositories

MySQL
mongoDB
PostgreSQL
RAILS
Ruby
Java
WordPress
redis
NGINX
node
ubuntu
debian
CentOS

Orchestration

MESOS
Shipper
CONSUL
VNS3
Marathon
fleet
ATOMIC

System Integrators

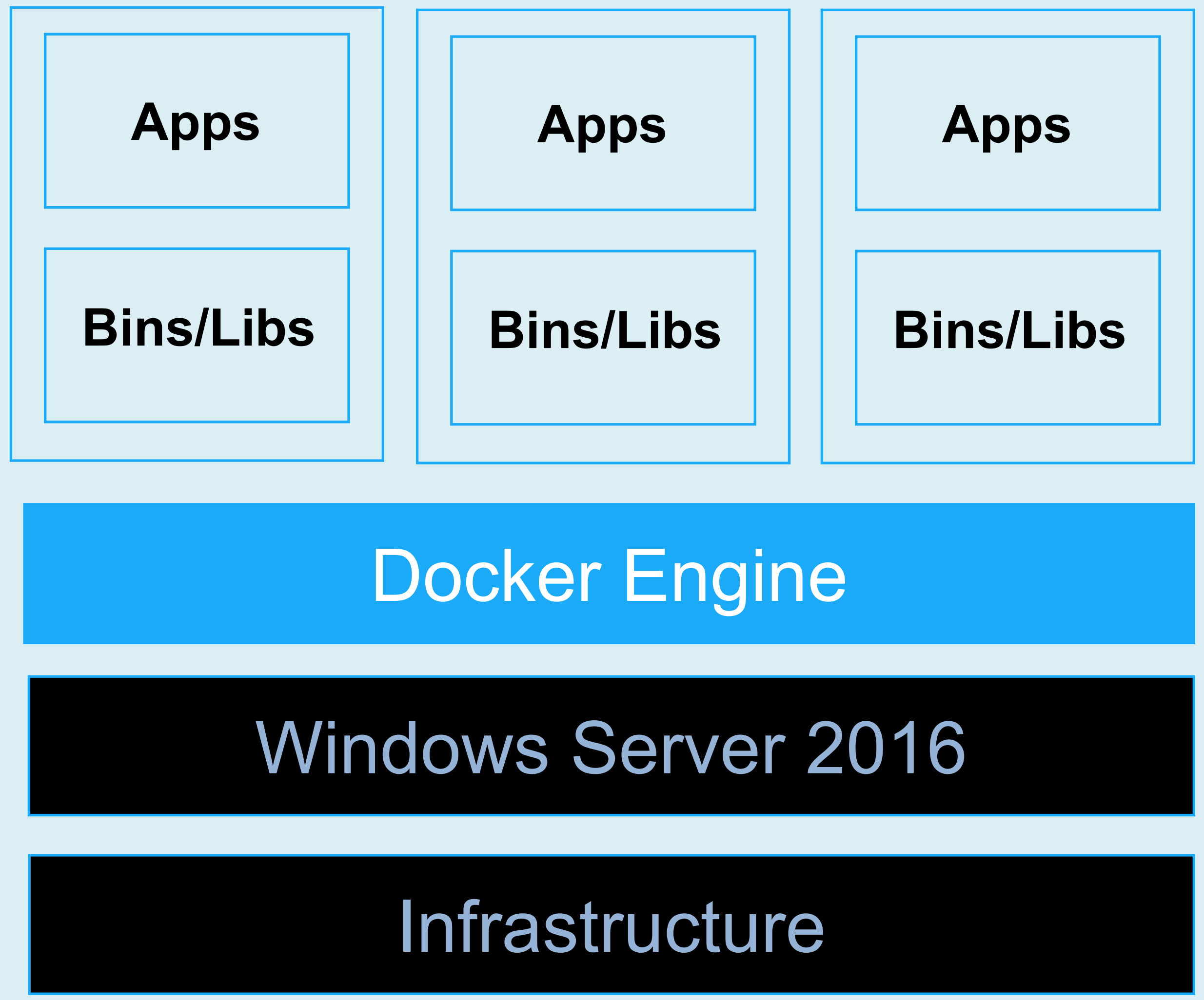
OpDemand
Wiredcraft
relevance lab
VIZURI
SHADOW-SOFT LIVE OPEN. SAVE HARD
Flux7
Dev9
MomentumSI
InfoSiftr

docker

Docker + Windows Server = Windows Containers

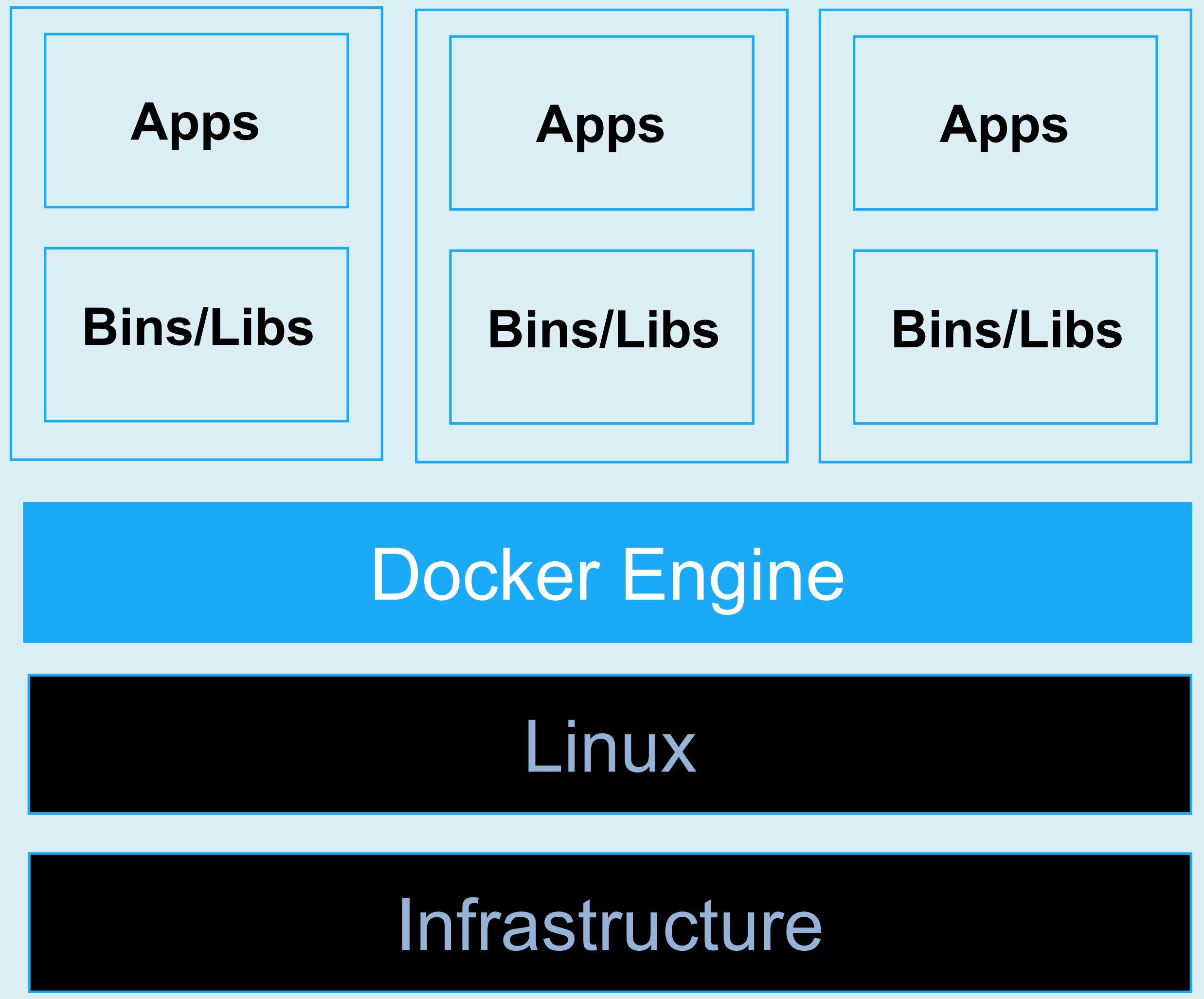
- Native Windows containers powered by Docker Engine

- Windows kernel engineered with new primitives to support containers



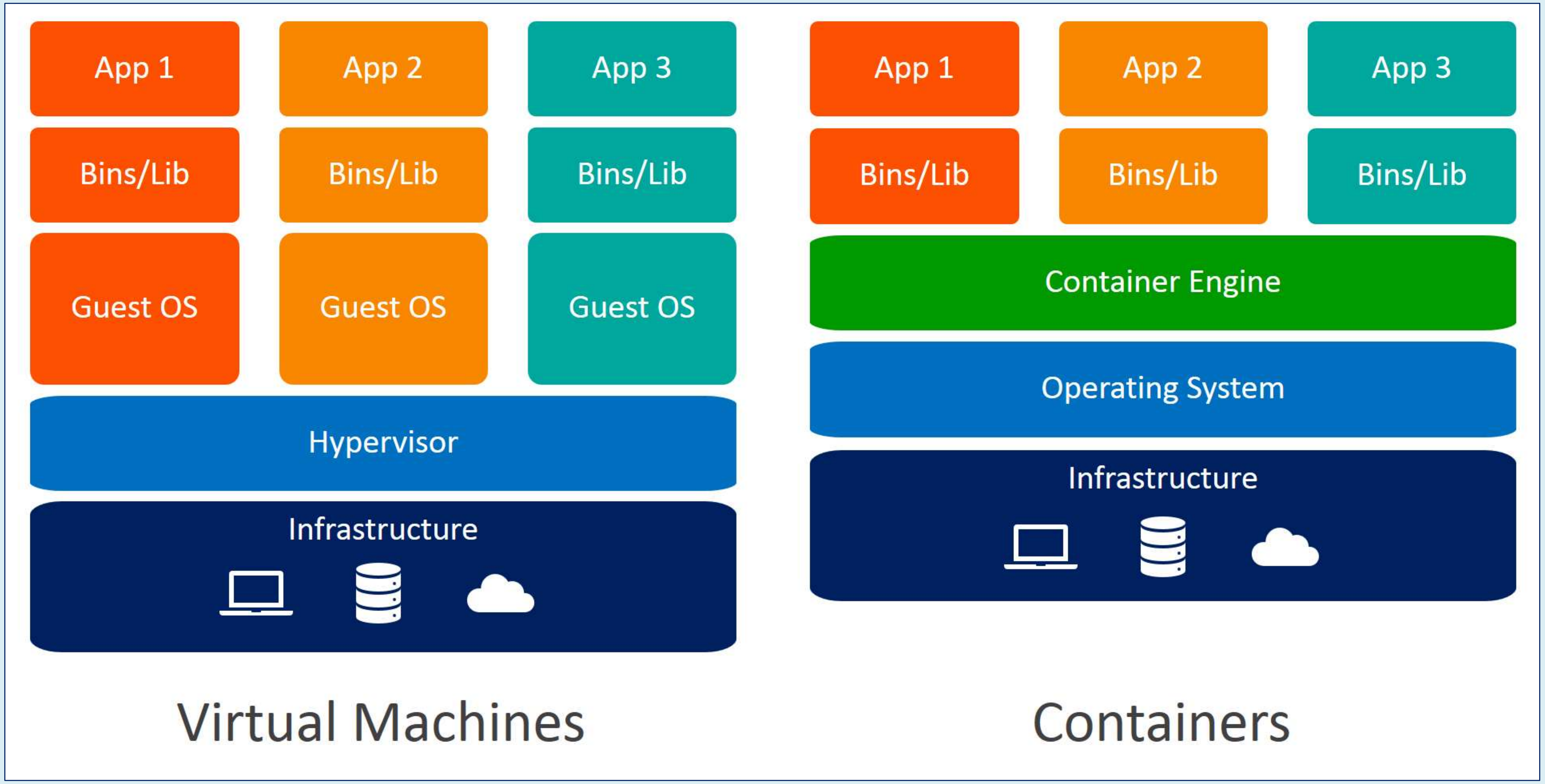
Linux Docker Containers

- Native Linux containers powered by Docker Engine
- Linux kernel already provided support for much of core of what goes into Docker

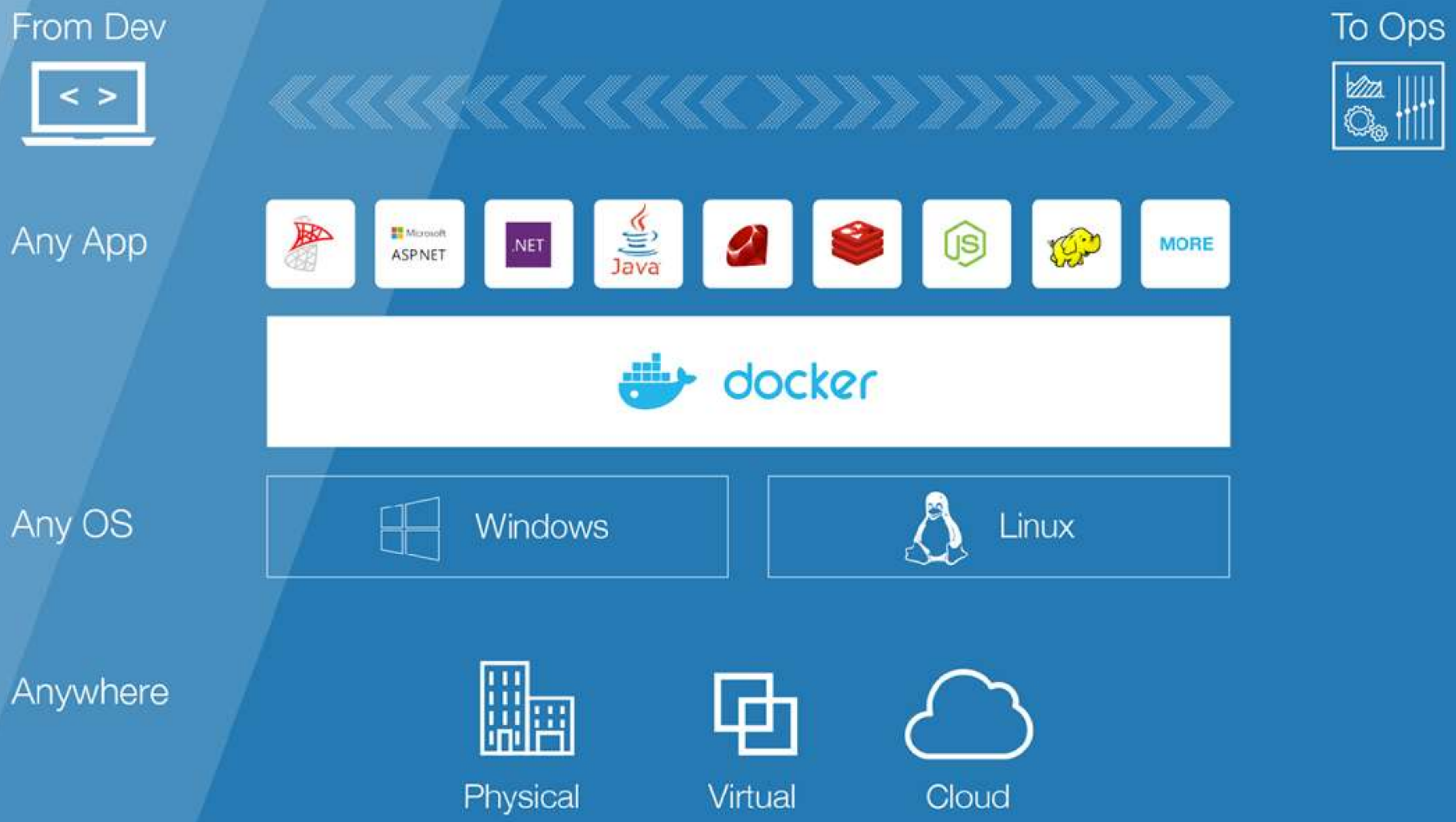


docker

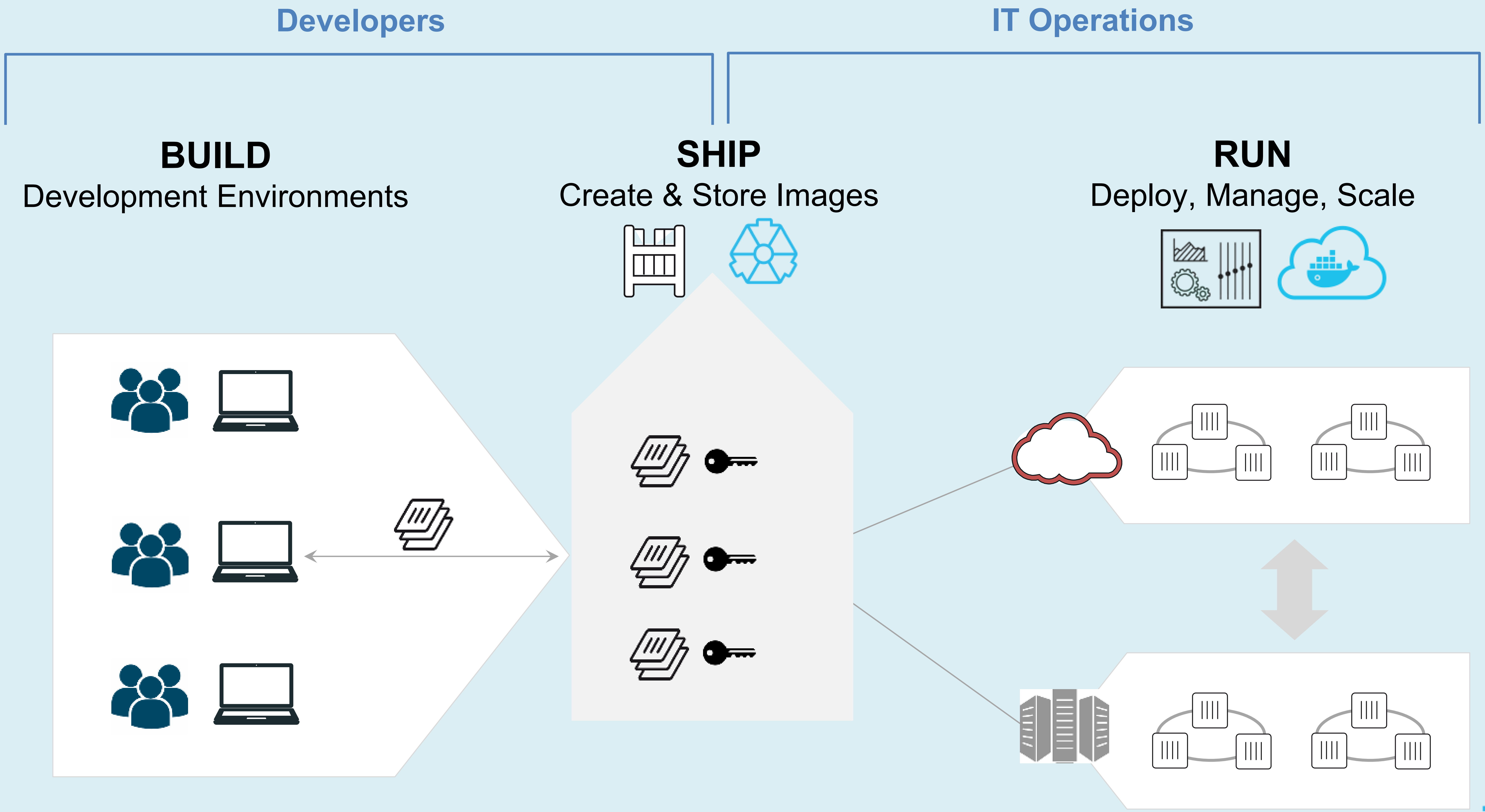
Recap: The Differences



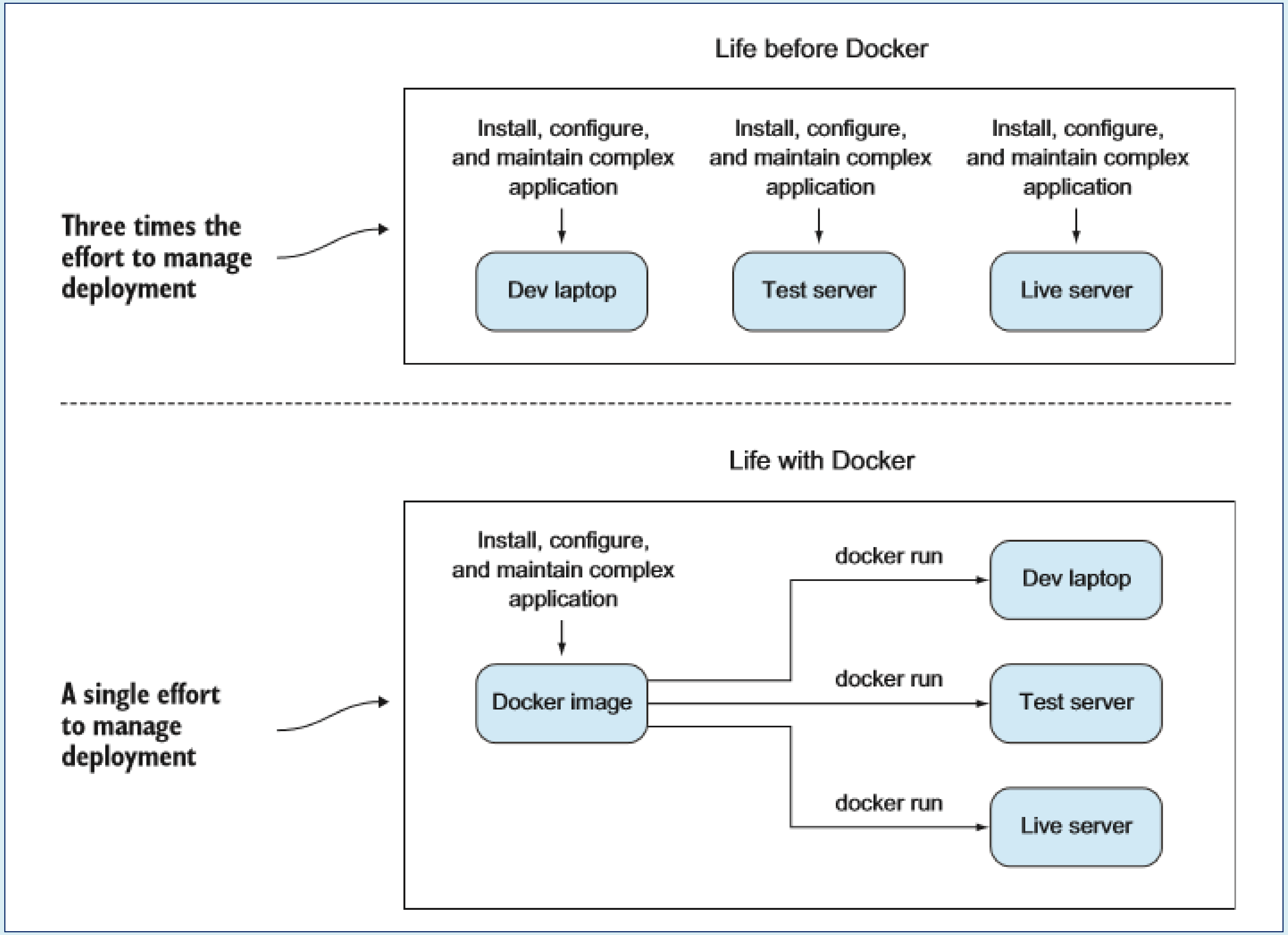
Build, Ship, Run, Any App Anywhere



Put it all together: Build, Ship, Run Workflow

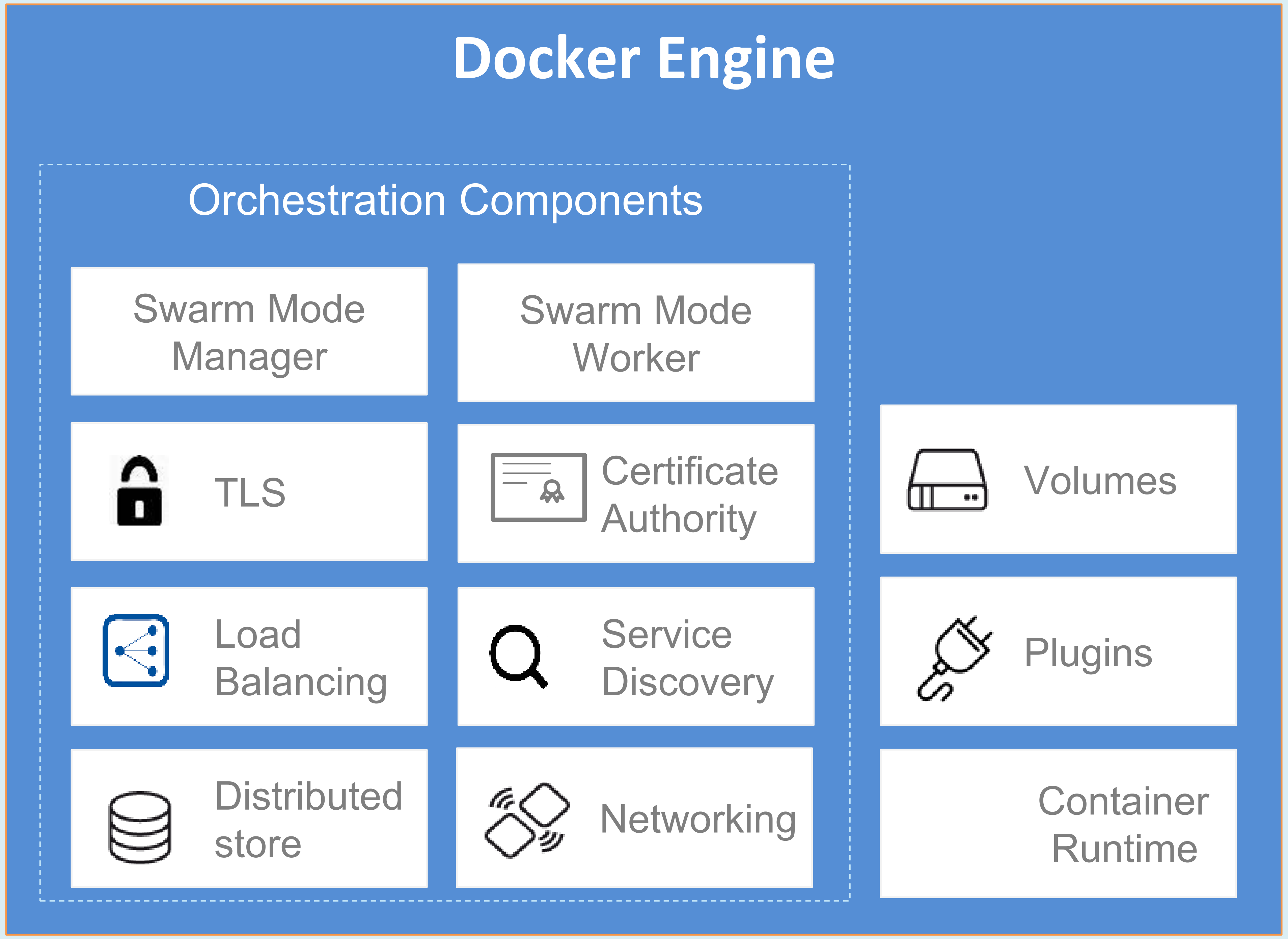


Put it all together: Build, Ship, Run Workflow


Life before Docker
Three times the effort to manage deployment
Install, configure, and maintain complex application
Dev laptop
Install, configure, and maintain complex application
Test server
Install, configure, and maintain complex application
Live server
Life with Docker
A single effort to manage deployment
Install, configure, and maintain complex application
Docker image
docker run
Dev laptop
docker run
Test server
docker run
Live server

The building block: Docker Engine 1.12

Built in orchestration with scheduling, networking and scheduling



- Powerful yet simple, built in orchestration
- Declarative app services
- Built in container centric networking
- Built in default security
- Extensible with plugins, drivers and open APIs

Some Docker vocabulary

Docker Image

The basis of a Docker container. Represents a full application and is read only.

Docker Container

The standard unit in which the application service resides and executes

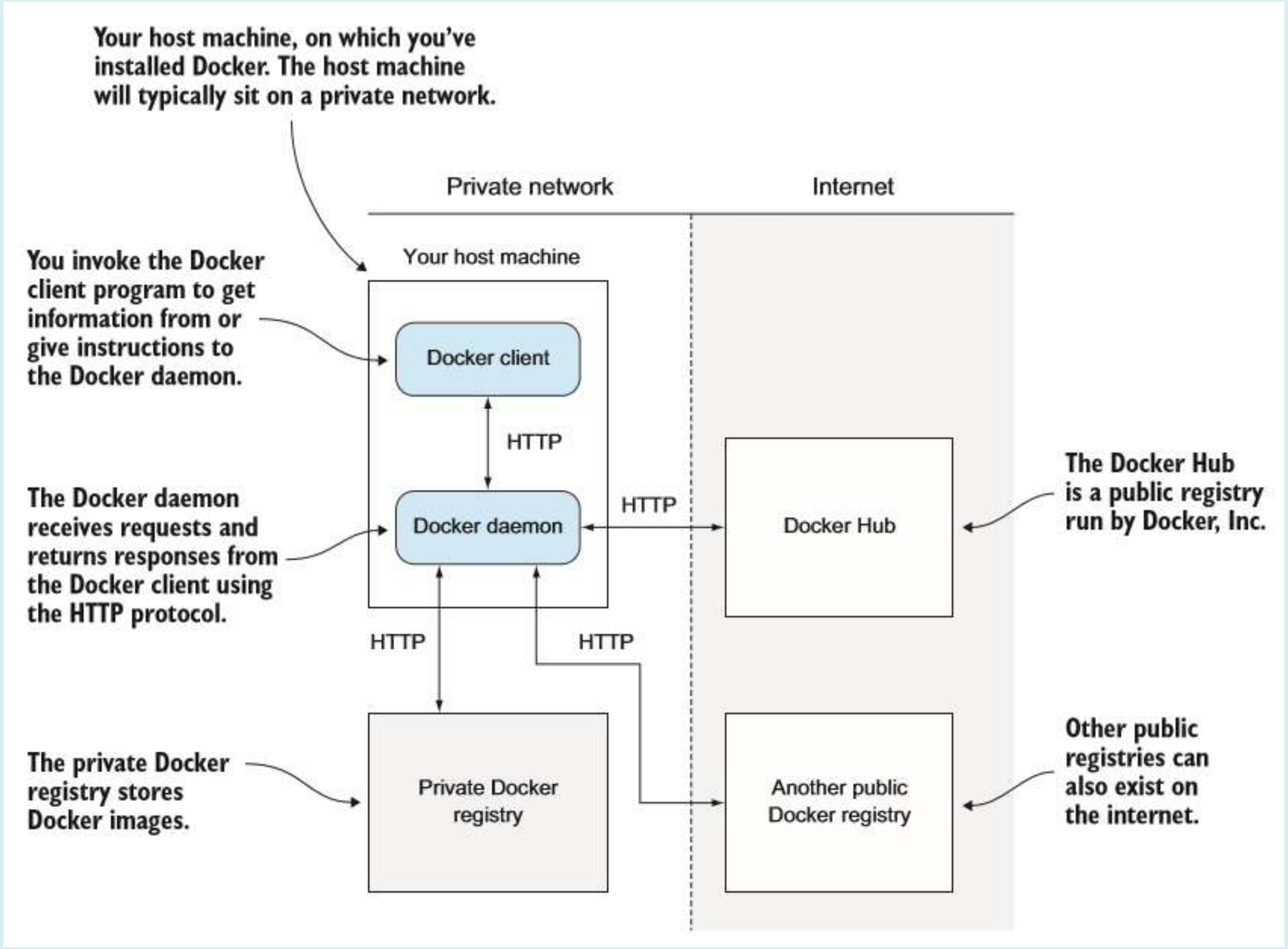
Docker Engine

Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

Registry Service (Docker Hub or Docker Trusted Registry)

Cloud or server-based storage and distribution service for your images

A look at Docker Architecture


Your host machine, on which you've installed Docker. The host machine will typically sit on a private network.
Private network
Internet
Your host machine
You invoke the Docker client program to get information from or give instructions to the Docker daemon.
Docker client
HTTP
The Docker daemon receives requests and returns responses from the Docker client using the HTTP protocol.
Docker daemon
HTTP
Docker Hub
The Docker Hub is a public registry run by Docker, Inc.
HTTP
HTTP
The private Docker registry stores Docker images.
Private Docker registry
Another public Docker registry
Other public registries can also exist on the internet.

docker

Basic Docker Commands

```
$ docker pull mikegcoleman/catweb:1.0

$ docker images

$ docker run -d -p 5000:5000 --name catweb mikegcoleman/catweb:latest

$ docker ps

$ docker stop catweb (or <container id>)

$ docker rm catweb (or <container id>)

$ docker rmi mikegcoleman/catweb:latest (or <image id>)

$ docker build -t mikegcoleman/catweb:2.0 .

$ docker push mikegcoleman/catweb:2.0
```

Basic Docker Commands

- Docker CLI structure,
  - Old (Still works as expected) docker <command> options
  - New – docker <command> <sub-command> (options)
- Pulling Docker Image
  - docker pull nginx
- Running a Docker Container
  - docker run –p 80:80 --name web-server nginx
- Stopping the Container
  - docker stop web-server (or container id)

Basic Docker Commands

- **Check what's happening in a containers,**
  - docker container top web-server – Process list in 1 container
  - docker container inspect web-server – Details of one container config
  - docker container stats – Performance stats for all containers
- **Getting a shell inside containers,**
  - docker container run –it – Start a new container interactively
  - docker container exec –it <container_id_or_name> echo "I'm inside the container" – Run additional commands in the container
- **Listing, removing containers and images**
  - docker images
  - docker container ls | docker ps
  - docker <object> rm <id_or_name>

NB: The –it is a short form for –i –t. The options are

-i	Run it interactively (aka – allow for keyboard access)
-t	Run it so that display is on screen
-d	Run it in background. It will run – you just wont see it on screen ☺

What happens when you run a container?

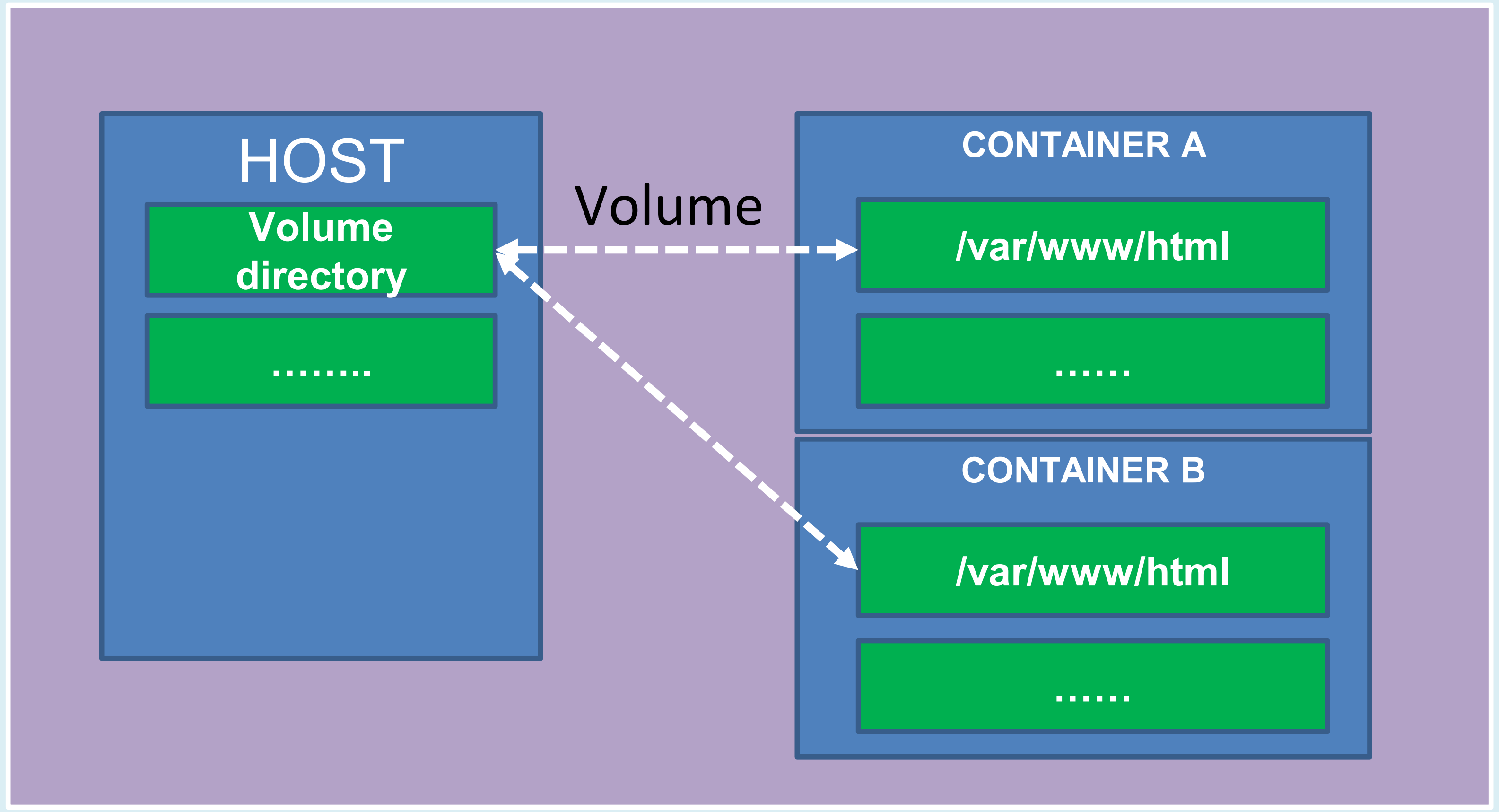
- ***docker run –p 80:80 nginx | docker container run –p 80:80 nginx***

1. Looks for that particular image locally in image cache, if its not found pulls it from the configured registry (image repository). Downloads the latest version by default (nginx:latest)
2. Creates a new container based on that image and prepares to start
3. Docker allocates read write filesystem to the container, as its final layer. This allows running container to modify files and directories in its local filesystem.
4. Gives it a virtual IP on a private network inside docker engine
5. Opens up port 80 on host and forwards to port 80 in container.
6. Starts container by using the CMD in the image Dockerfile.

Docker Volumes

- Data is not persisted when a container is stopped or removed.
- To persist data, use Volumes (see –v option of docker run)
- This maps a directory or file on the HOST to corresponding File or Directory.

Docker Volumes



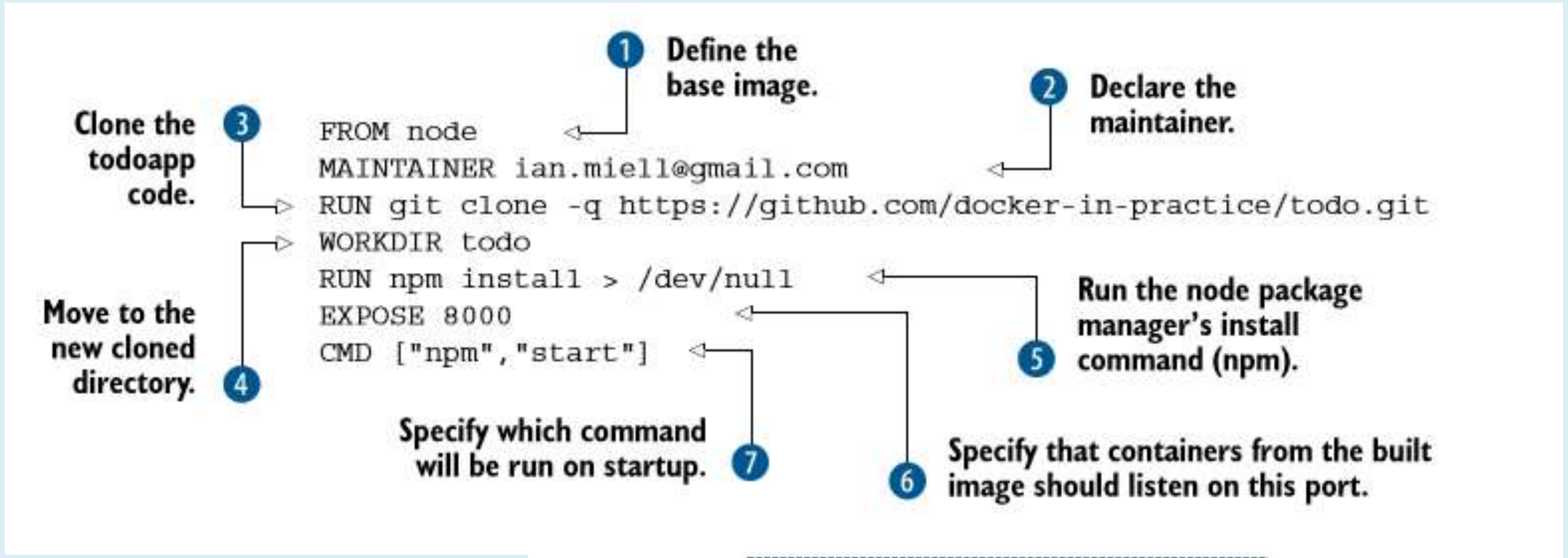
Docker Networks

- Each container is connected to a private virtual network called “bridge”.
  - There are 3 basic networks – bridge, host and none.
- Each virtual network routes through the NAT firewall on the host IP.
- All containers on a virtual network can talk to each other without exposing ports.
- Best practice is to create a new virtual network for each app.

Docker Networks

- Docker enables to:
  - Create new virtual networks.
  - Attach container to more than one virtual network (or none)
  - Skip virtual networks and use host IP (--net=host)
  - Use different Docker network drivers to gain new abilities.
    - Docker Engine provides support for different network drivers – bridge (default), **overlay** and **macvlan** etc.. . You can even write your own network driver plugin to create your own one.
- **Docker Networking – DNS**
  - Docker deamon has a built in DNS, which consider container name as equivalent hostname of the container.

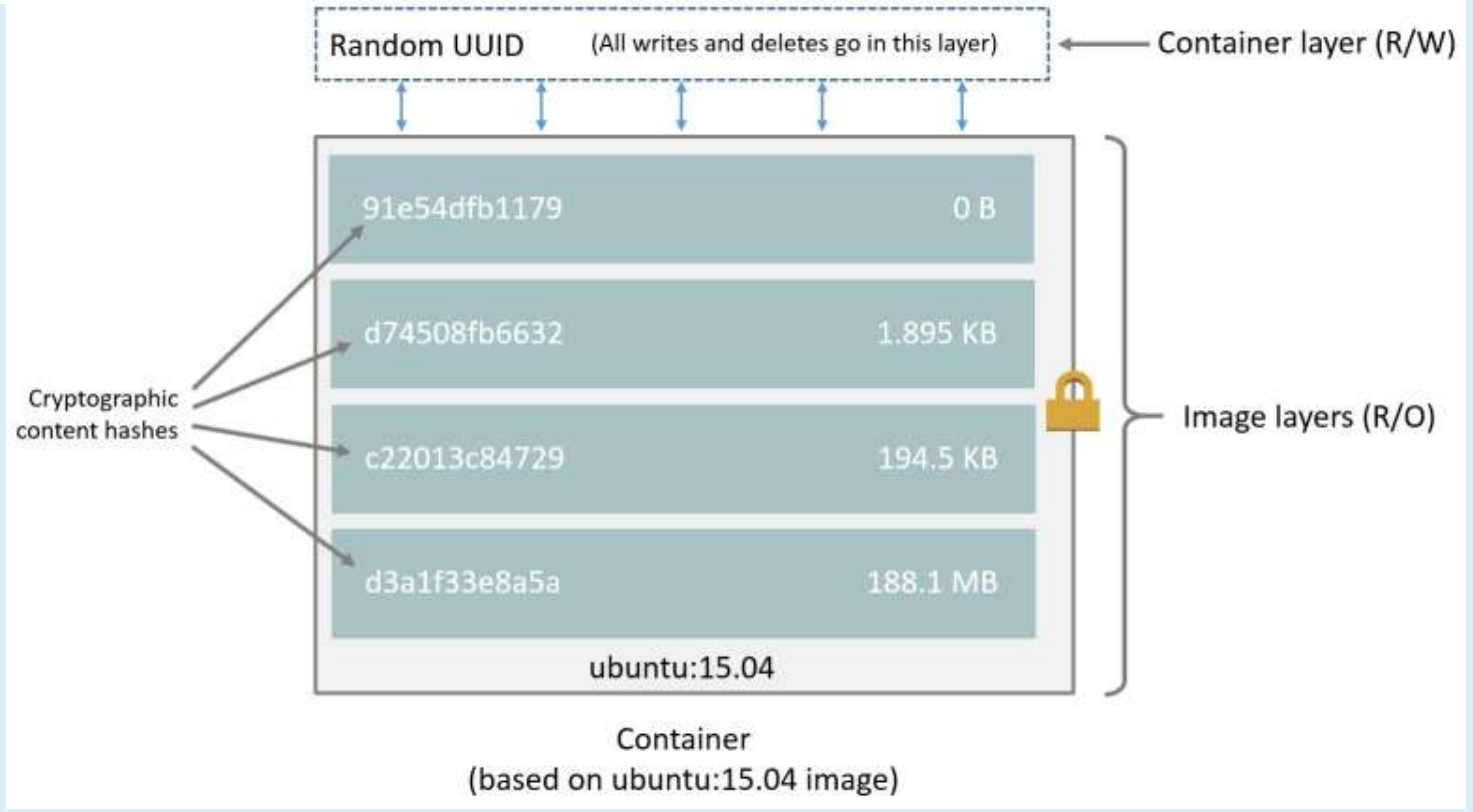
Dockerfile – getting started



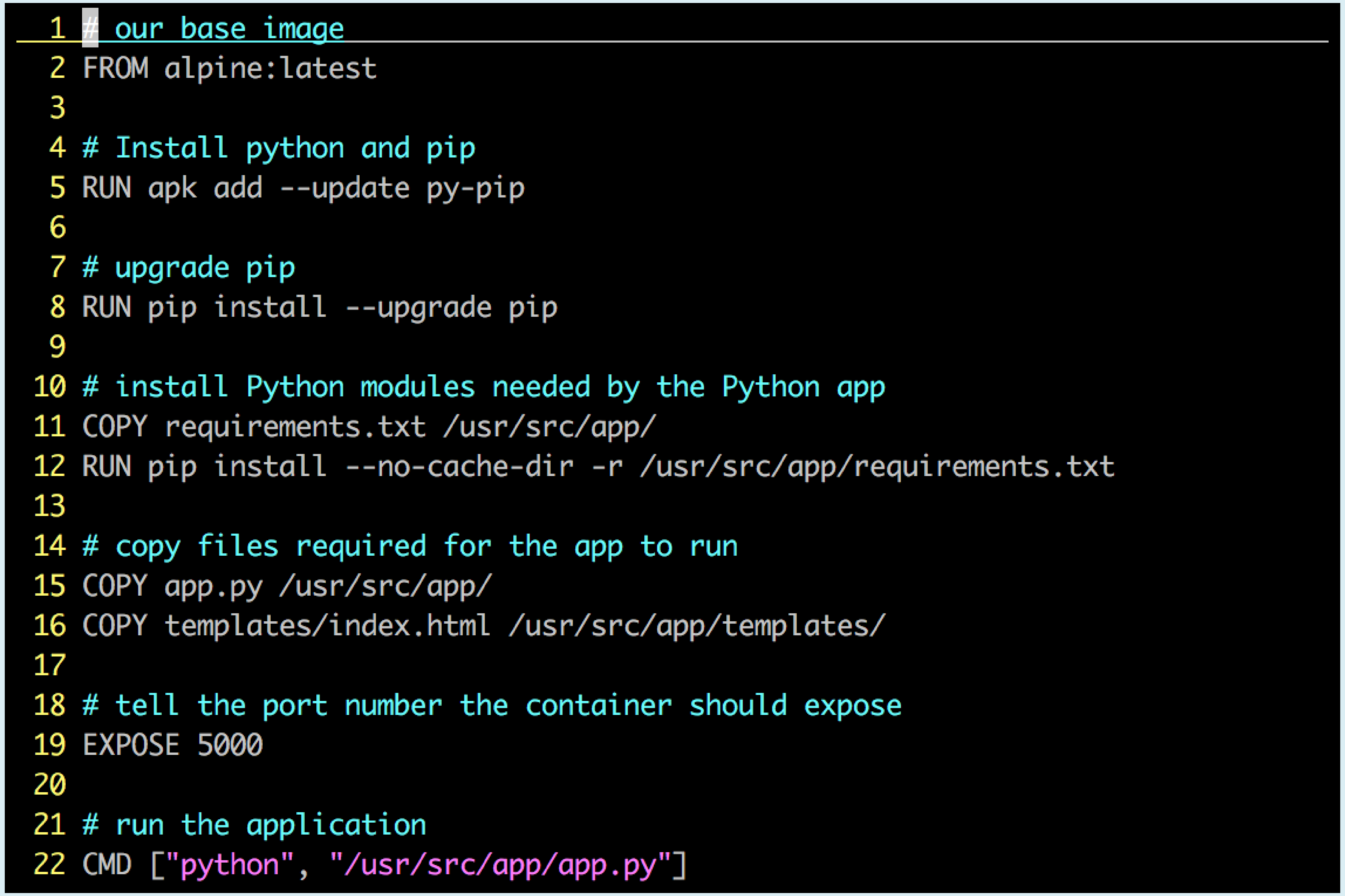
```
FROM node
MAINTAINER ian.miell@gmail.com
RUN git clone -q https://github.com/docker-in-practice/todo.git
WORKDIR todo
RUN npm install > /dev/null
EXPOSE 8000
CMD ["npm","start"]
```

1 Define the base image.
2 Declare the maintainer.
3 Clone the todoapp code.
4 Move to the new cloned directory.
5 Run the node package manager's install command (npm).
6 Specify that containers from the built image should listen on this port.
7 Specify which command will be run on startup.

- Automatically build images and deploy
- Automated build on Docker Hub
- Easy to share and customize config



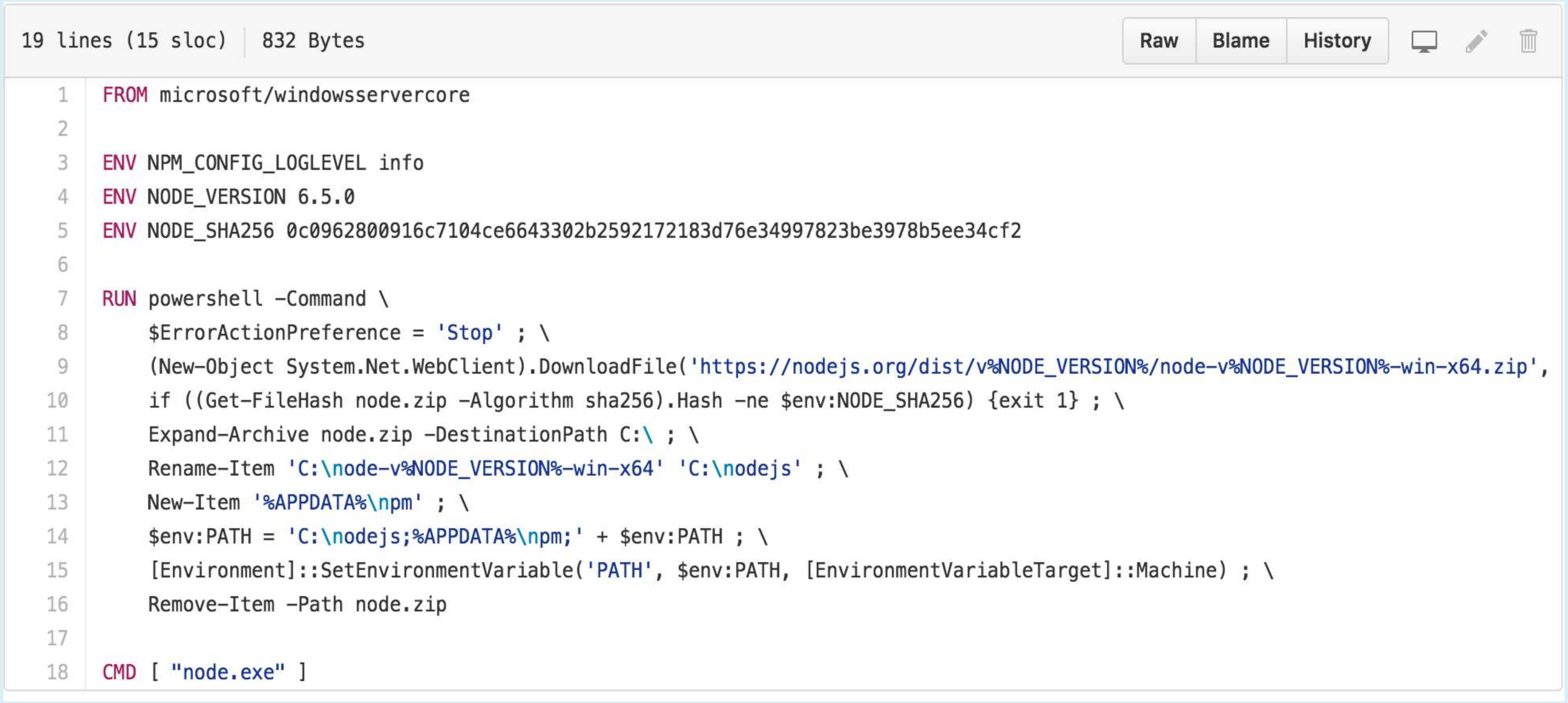
Dockerfile – Linux Example



```
# our base image
FROM alpine:latest

# Install python and pip
RUN apk add --update py-pip

# upgrade pip
RUN pip install --upgrade pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```

- Instructions on how to build a Docker image
- Looks very similar to "native" commands
- Important to optimize your Dockerfile

Dockerfile – Windows Example



```dockerfile
FROM microsoft/windowsservercore

ENV NPM_CONFIG_LOGLEVEL info
ENV NODE_VERSION 6.5.0
ENV NODE_SHA256 0c0962800916c7104ce6643302b2592172183d76e34997823be3978b5ee34cf2

RUN powershell -Command \
    $ErrorActionPreference = 'Stop' ; \
    (New-Object System.Net.WebClient).DownloadFile('https://nodejs.org/dist/v%NODE_VERSION%/node-v%NODE_VERSION%-win-x64.zip',
    if ((Get-FileHash node.zip -Algorithm sha256).Hash -ne $env:NODE_SHA256) {exit 1} ; \
    Expand-Archive node.zip -DestinationPath C:\ ; \
    Rename-Item 'C:\node-v%NODE_VERSION%-win-x64' 'C:\nodejs' ; \
    New-Item '%APPDATA%\npm' ; \
    $env:PATH = 'C:\nodejs;%APPDATA%\npm;' + $env:PATH ; \
    [Environment]::SetEnvironmentVariable('PATH', $env:PATH, [EnvironmentVariableTarget]::Machine) ; \
    Remove-Item -Path node.zip

CMD [ "node.exe" ]
```

Dockerfiles

- Remember – A Dockerfile (Note the **C**apital **D**) is just a set of instructions.
- Directives are always written in CAP CASES. E.g. RUN
- Some Directives are

– FROM	The Base Image on which this image is based
– RUN	A Command that is used to modify the base image by running commands
– COPY	Copy a set of files/directories from HOST to image
– ADD	Same as COPY – Prefer to use COPY for most cases.
– ENV	Sets An Environment Variable
– EXPOSE	Expose a Port
– VOLUME	Specifies that a directory should be stored outside the union file system
– ENTRYPOINT	The starting program of the Image when it runs.
– CMD	Sets Default parameters for the ENTRYPOINT

Dockerfiles

Command	Description	Use Case
CMD	Defines the default executable of a Docker image. It can be overridden by docker run arguments.	Utility images allow users to pass different executables and arguments on the command line.
ENTRYPOINT	Defines the default executable. It can be overridden by the “--entrypoint” docker run arguments.	Images built for a specific purpose where overriding the default executable is not desired.
RUN	Executes commands to build layers.	Building an image

docker

Dockerfiles

- 2 Ways to pass arguments to CMD and ENTRYPOINT.

Form	**Description**	**Example**
Shell Form	Takes the form of <INSTRUCTION> <COMMAND>.	CMD echo TEST or ENTRYPOINT echo TEST
Exec Form	Takes the form of <INSTRUCTION> ["EXECUTABLE", "PARAMETER"].	CMD ["echo", "TEST"] or ENTRYPOINT ["echo", "TEST"]

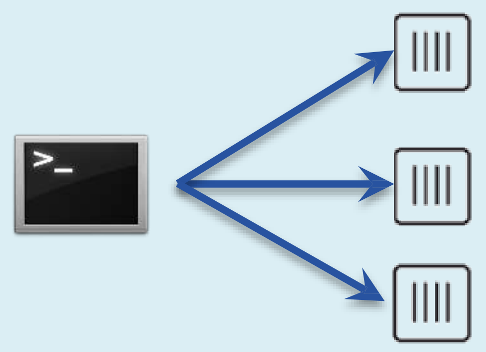
Docker Compose

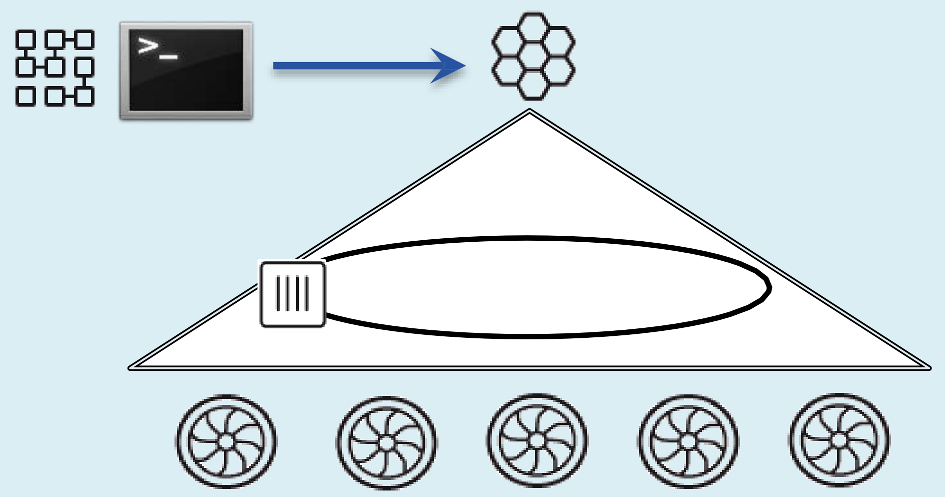
Without using Compose

- ❑ Build and run one container at a time
- ❑ Manually connect containers
- ❑ Manual Dependency Management

With Compose

- ❑ Define multi-container apps in a single file
- ❑ Single command to deploy entire apps
- ❑ Handles Dependencies
- ❑ Works with Networking, Volumes, Swarm





Docker Compose

1

A tool for defining and running multi-container Docker applications

2

With Compose, you use a YAML file to configure your application's services.

3

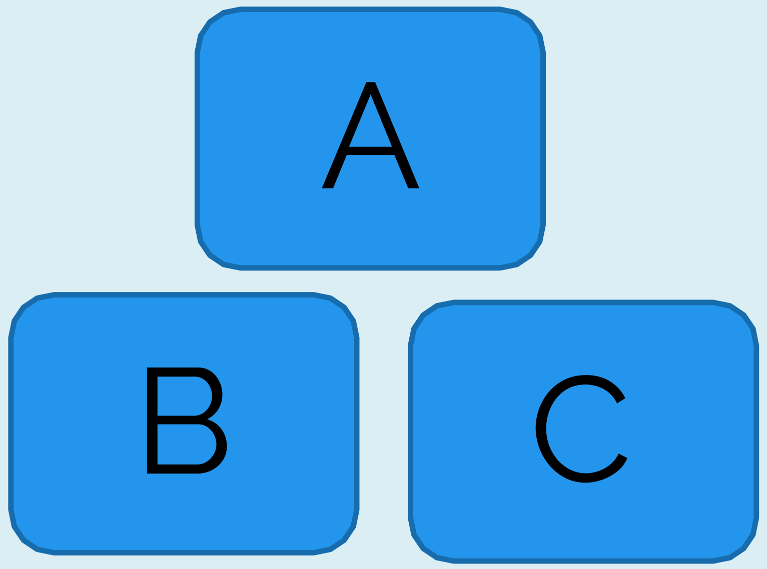
Compose works in all environments: production, staging, development, testing, as well as CI workflows.

4

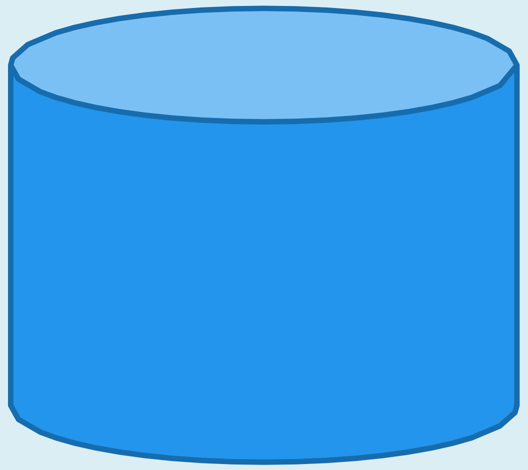
With a single command, you create and start all the services from your configuration

docker

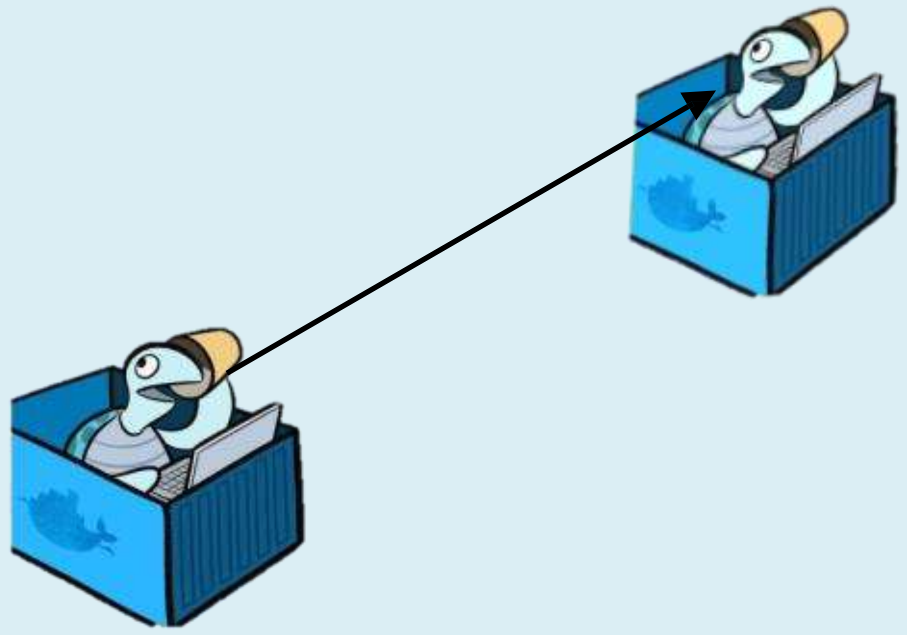
Docker Compose: Multi Container Applications



Services



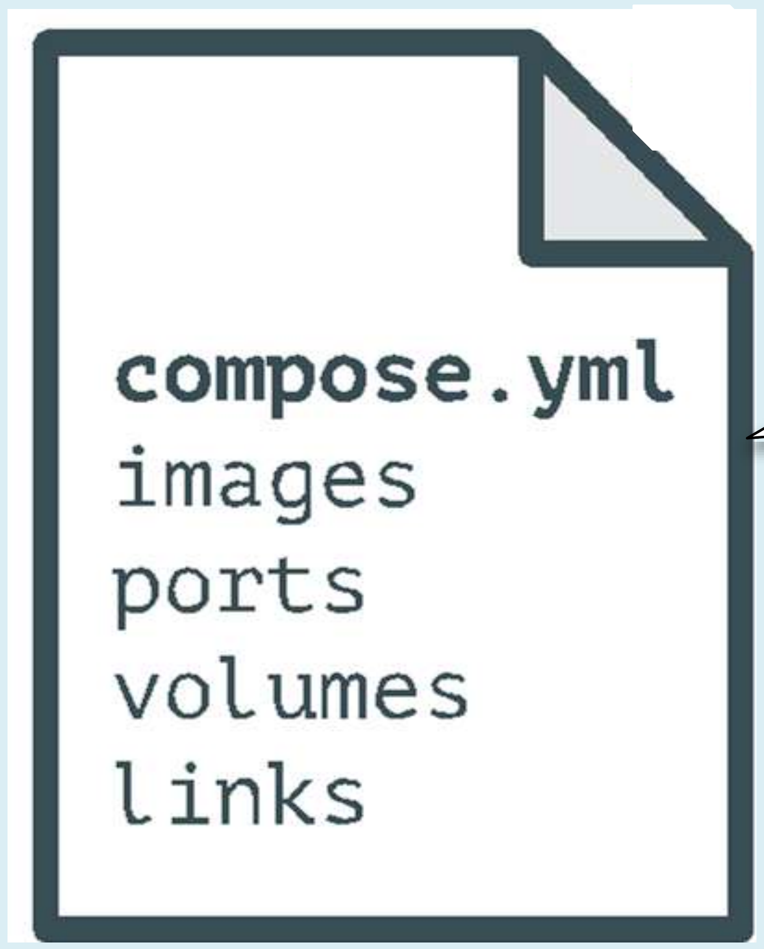
Volumes



Networking

docker

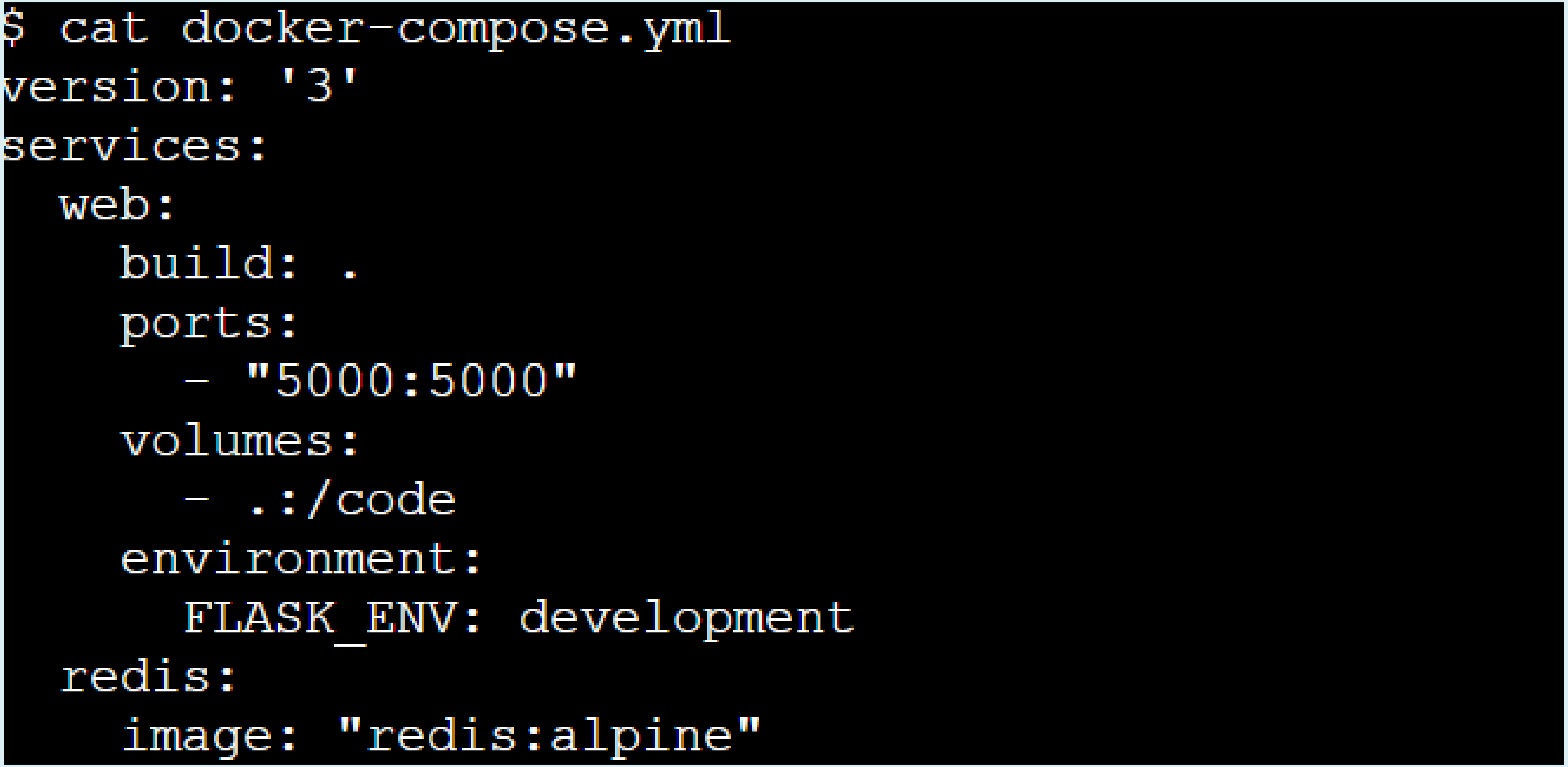
Docker Compose: Multi Container Applications



```
compose.yml
images
ports
volumes
links
```

```
containers:
  web:
     build: .
     command: python app.py
     ports:
     - "5000:5000"
     volumes:
     - .:/code
     environment:
     - PYTHONUNBUFFERED=1
  redis:
     image: redis:latest
     command: redis-server --appendonly yes
```

Docker Compose: Multi Container Applications


```
$ cat docker-compose.yml
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

Docker Compose: Multi Container Applications

Using .env file

```
$ cat .env
TAG=v1.5

$ cat docker-compose.yml
version: '3'
services:
  web:
    image: "webapp:${TAG}"
```

Docker Compose: Multi Container Applications

Commands:

- *docker compose up*
- *docker compose down*

- *docker compose run –e DEBUG=1 <services>*

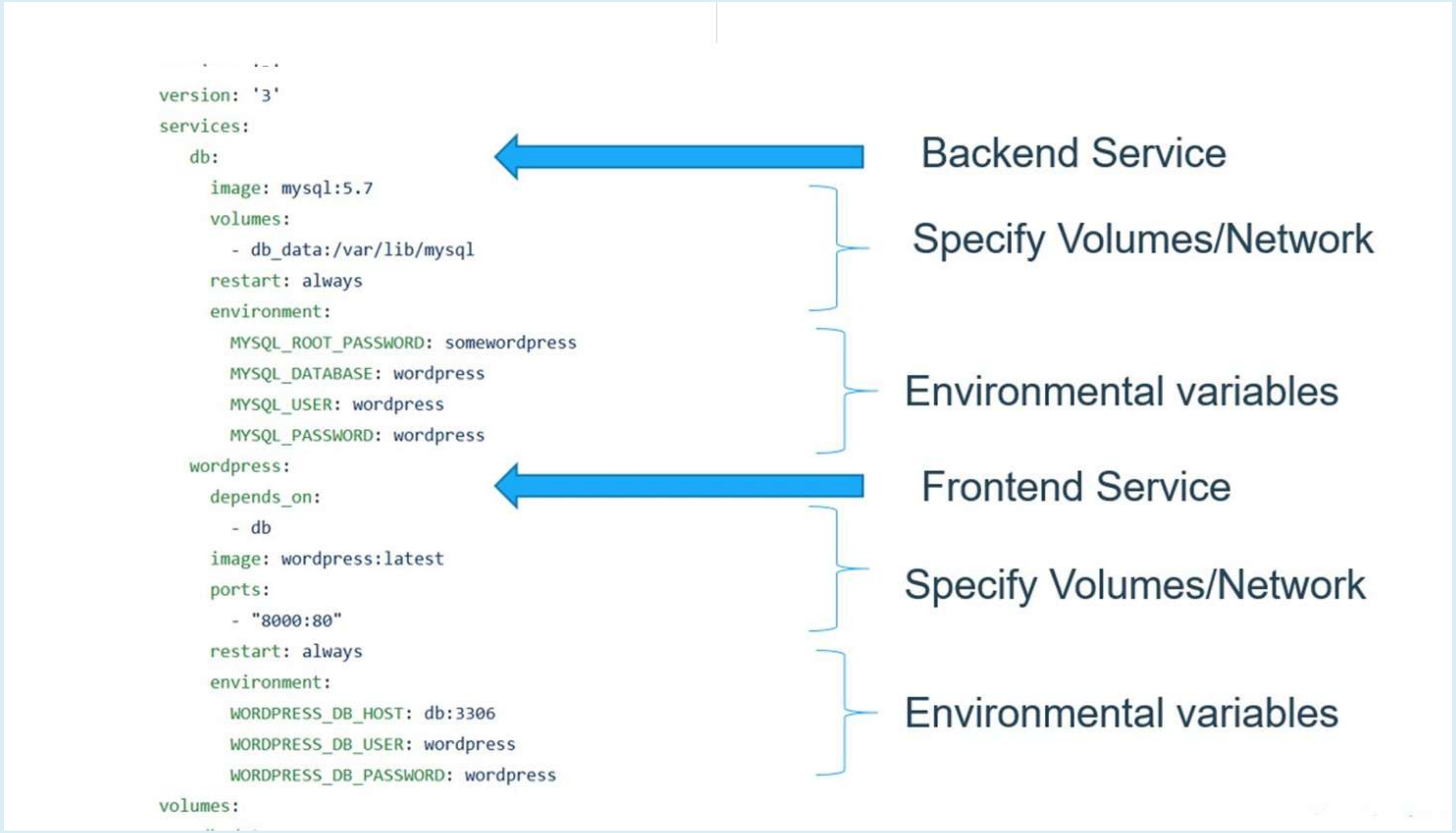
Docker Compose: Multi Container Applications

- A network called myapp_default is created.
- Name is based on directory
- A container is created using web's configuration.
  - It joins the network myapp_default under the name web.
- A container is created using db's configuration.
- Joins the network myapp_default under the name db.

```
version: "3"
services:
  web:
    build: .
    ports:
      - "8000:8000"
  db:
    image: postgres
    ports:
      - "8001:5432"
```

docker

Docker Compose: Multi Container Applications



```
version: '3'
services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress
   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
```

Q & A

docker

Thank you!

docker