AMRITA VISHWA VIDYAPEETHAM

15CSE337
Cloud Computing and Services

Virtualization basics
Dockers

Dr Ganesh Neelakanta Iyer
Associate Professor, Dept of Computer Science and Engg
Amrita Vishwa Vidyapeetham, Coimbatore

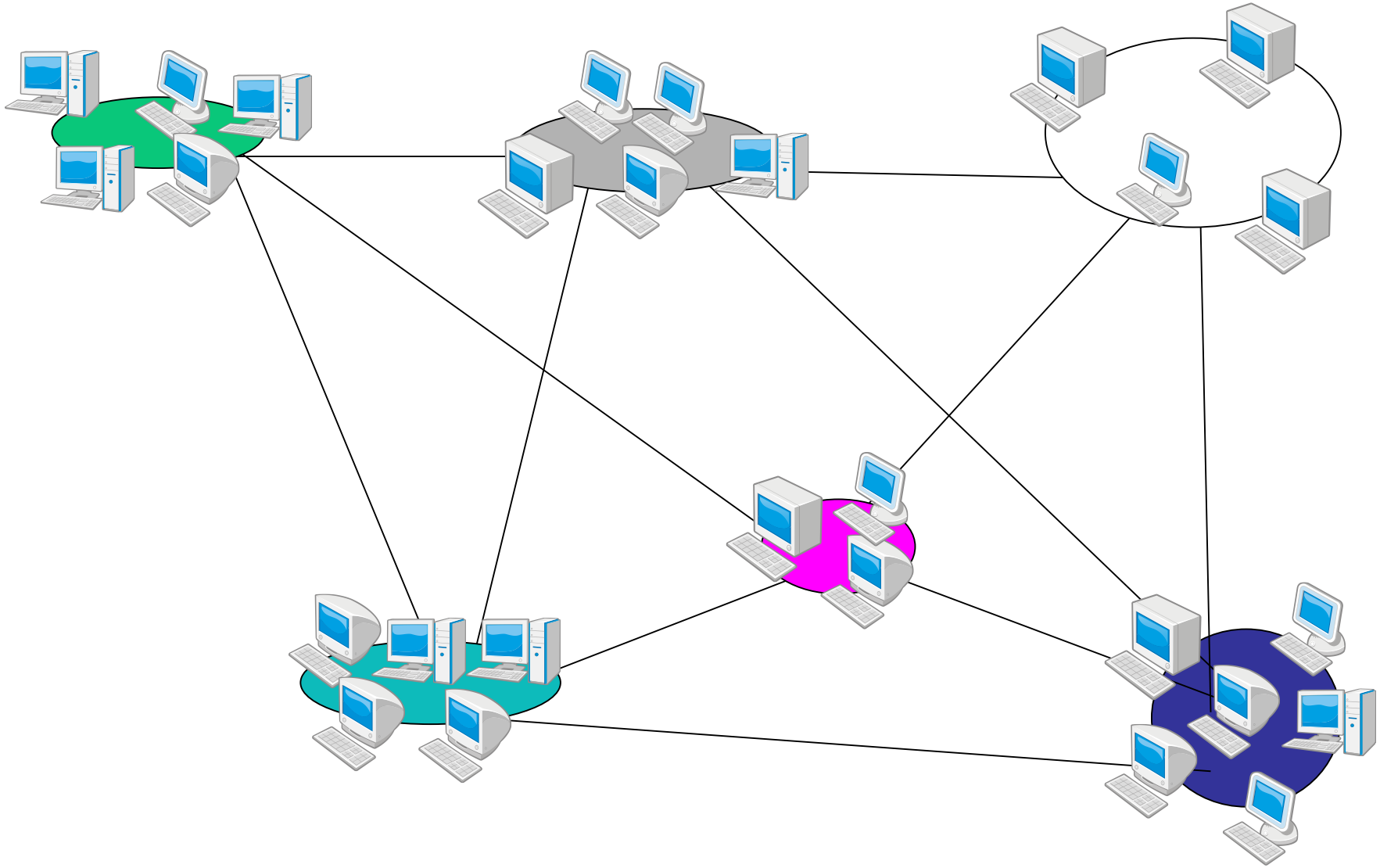
What did we learn so far

- Various distributed computing paradigms
- Overview of cloud computing
  - Deployment models
  - Delivery models
  - Examples
- Web application essentials
  - Web server, app server
  - Web Services

What next?

- Virtualization concepts
- Containerization concepts
  - Dockers

In the computer-age…



A Lot of Servers/Machines...

- Web server
- Mail server
- Database server
- File server
- Proxy server
- Application server
- …and many others

A Lot of Servers/Machines...

- The data-centre is FULL
  - Full of under utilized servers
  - Complicate in management
- Power consumption
  - Greater wattage per unit area than ever
  - Electricity overloaded
  - Cooling at capacity
- Environmental problem
  - Green IT

Recent Advances

- Multi-core: how to fully harness the power of multi-core? Intel has been trying really hard to make us all program for multi-core!!!
- Large scale data: How to manage large scale data? Google, Yahoo, NSF and CRA have been promoting their file systems and MapReduce!!
- Parallel processing of data: How to configure clusters to process data in parallel?
- Answer: Virtualization?

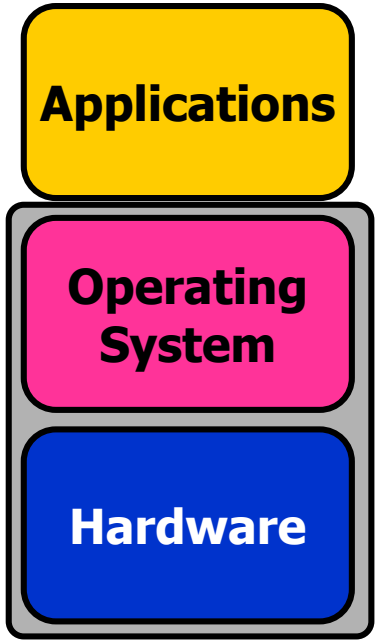
Virtualization

- Virtualization -- the abstraction of computer resources.

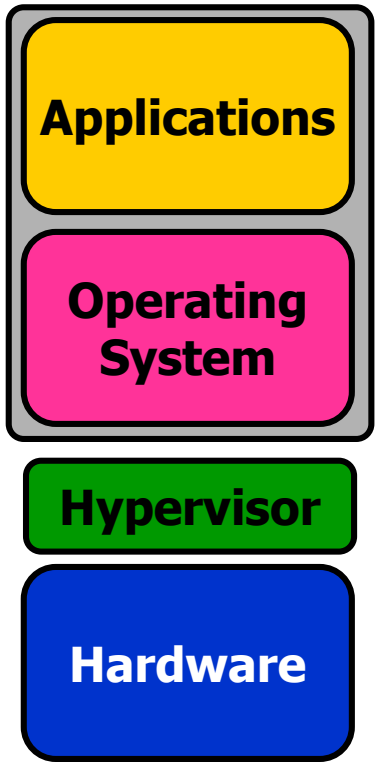
- Virtualization hides the physical characteristics of computing resources from their users, be they applications, or end users.

- This includes making a single physical resource (such as a server, an operating system, an application, or storage device) appear to function as multiple virtual resources; it can also include making multiple physical resources (such as storage devices or servers) appear as a single virtual resource.

The Use of Computers



Virtualization



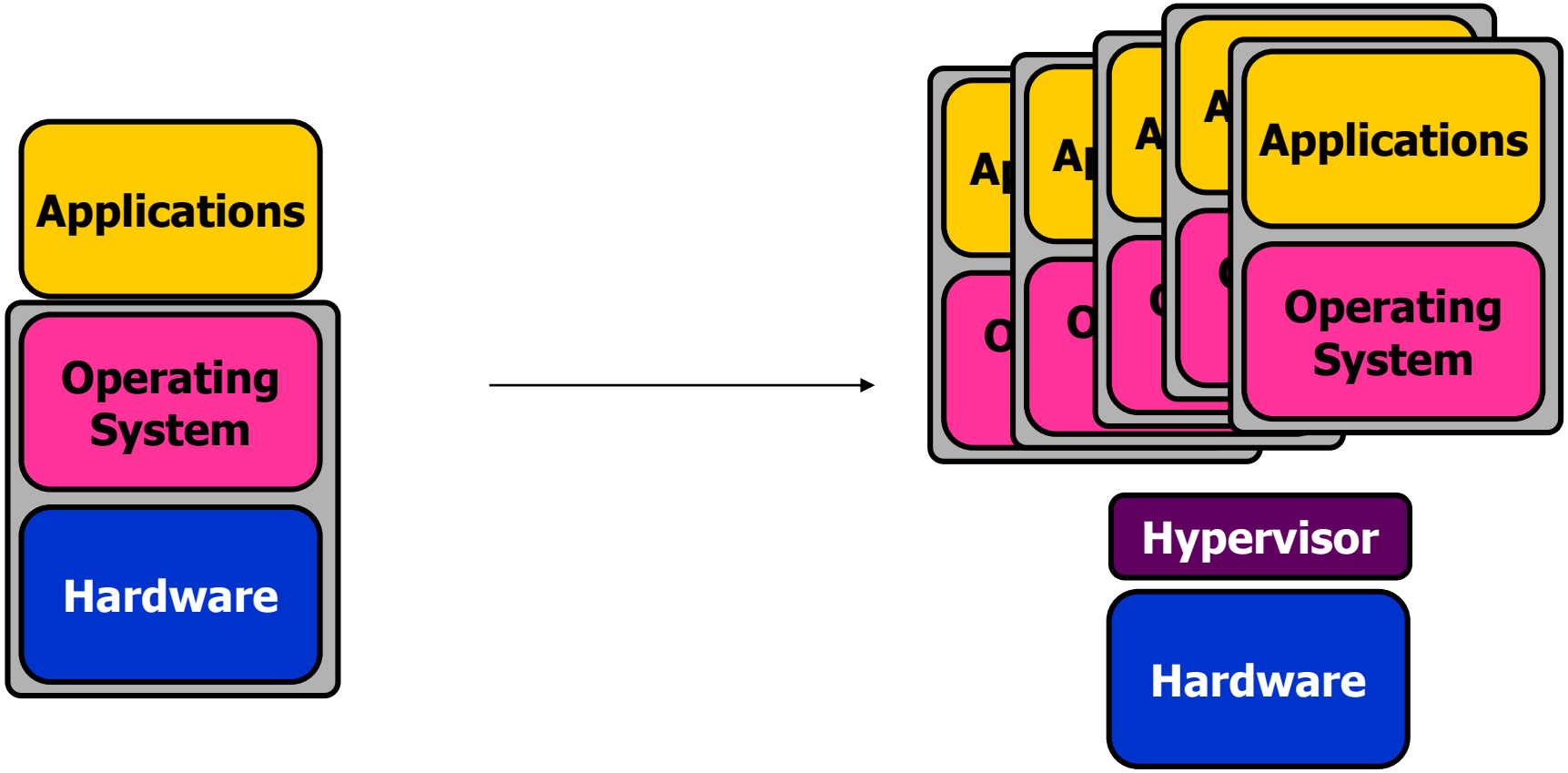
Traditional vs Virtual Architecture

Traditional Architecture

Virtual Architecture

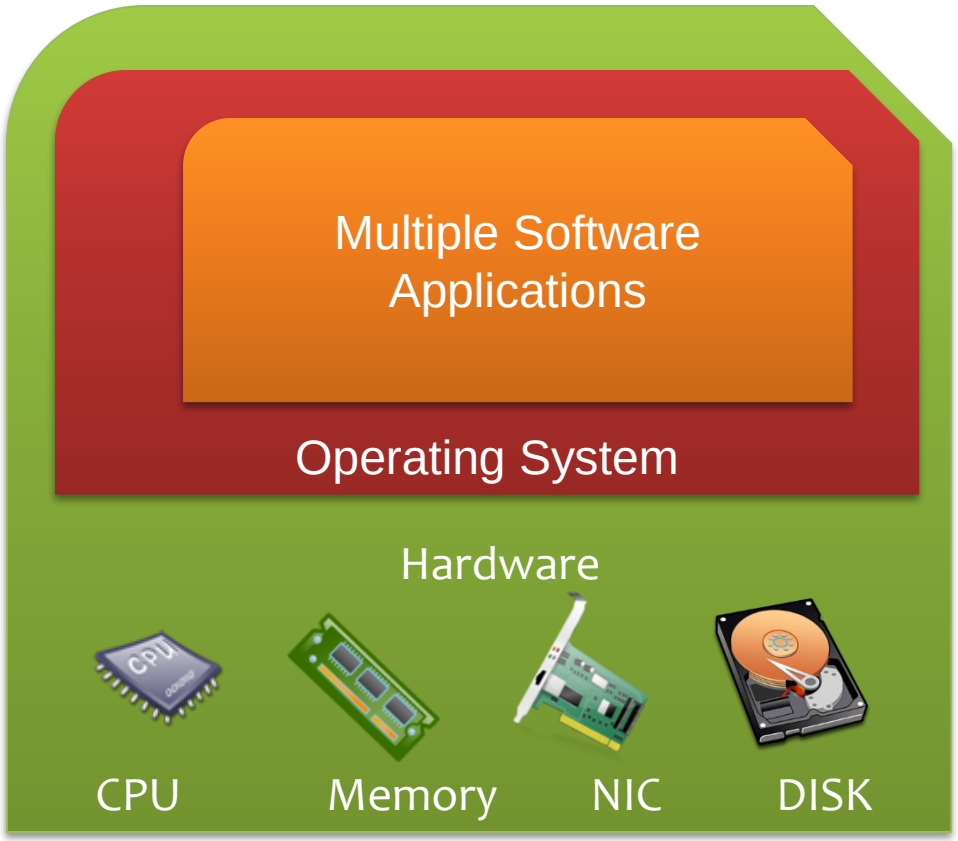
Access to the virtual machine and the host machine or server is facilitated by a software known as Hypervisor. Hypervisor acts as a link between the hardware and the virtual environment and distributes the hardware resources such as CPU usage, memory allotment between the different virtual environments.

Virtualization -- a Server for Multiple Applications/OS



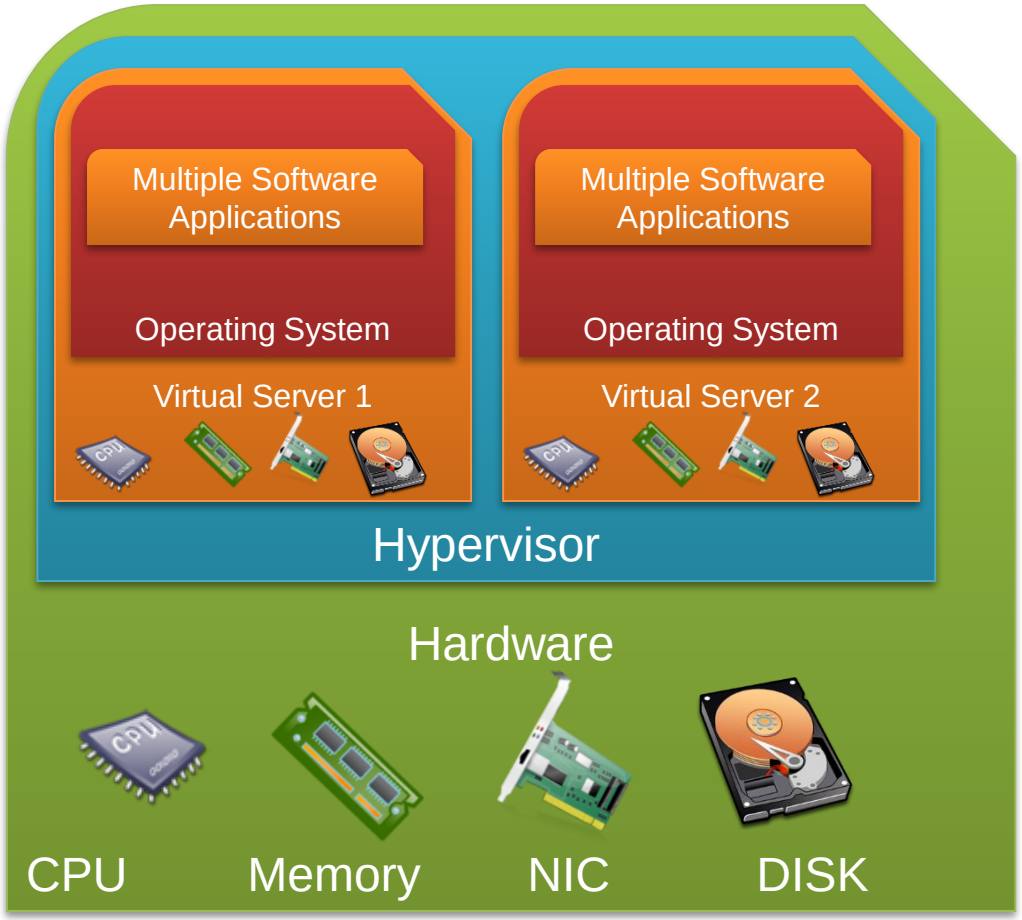
Hypervisor is a software program that manages multiple operating systems (or multiple instances of the same operating system) on a single computer system.
The hypervisor manages the system's processor, memory, and other resources to allocate what each operating system requires.
Hypervisors are designed for a particular processor architecture and may also be called **virtualization managers**.

Server without virtualization



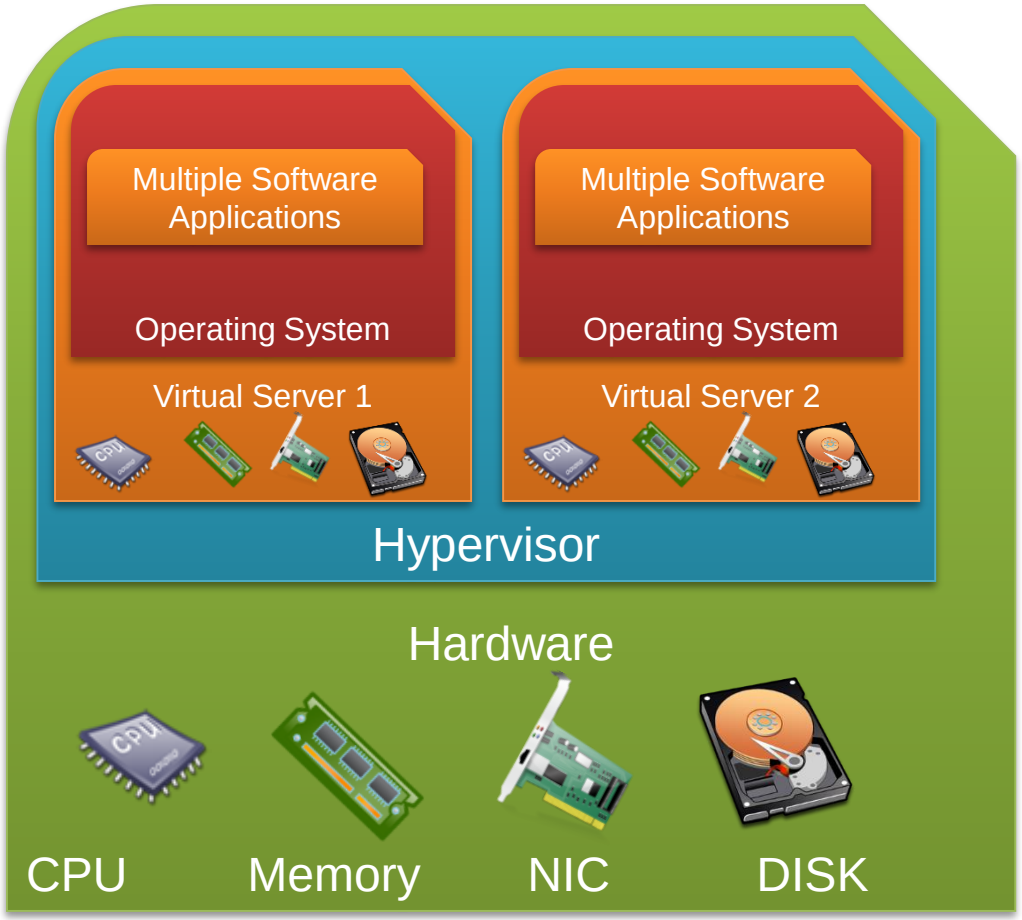
- Only one OS can run at a time within a server.
- Under utilization of resources.
- Inflexible and costly infrastructure.
- Hardware changes require manual effort and access to the physical server

Server with virtualization



- Can run multiple OS simultaneously.
- Each OS can have different hardware configuration.
- Efficient utilization of hardware resources.
- Each virtual machine is independent.
- Save electricity, initial cost to buy servers, space etc.
- Easy to manage and monitor virtual machines centrally.

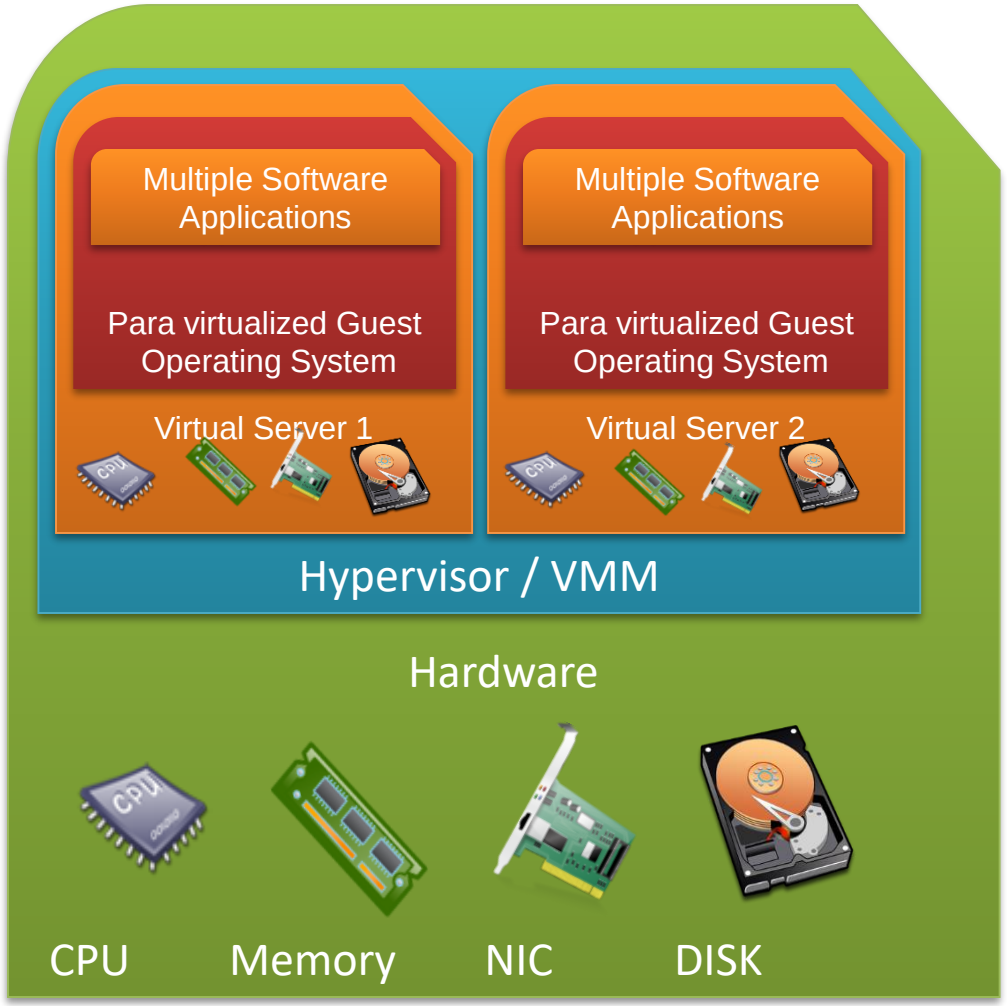
HYPERVISOR TYPE



Full virtualization

- Enables hypervisors to run an unmodified guest operating system (e.g. Windows 2003 or XP).
- Guest OS is not aware that it is being virtualized.
- E.g.: VMware uses a combination of direct execution and binary translation techniques to achieve full virtualization of server systems.

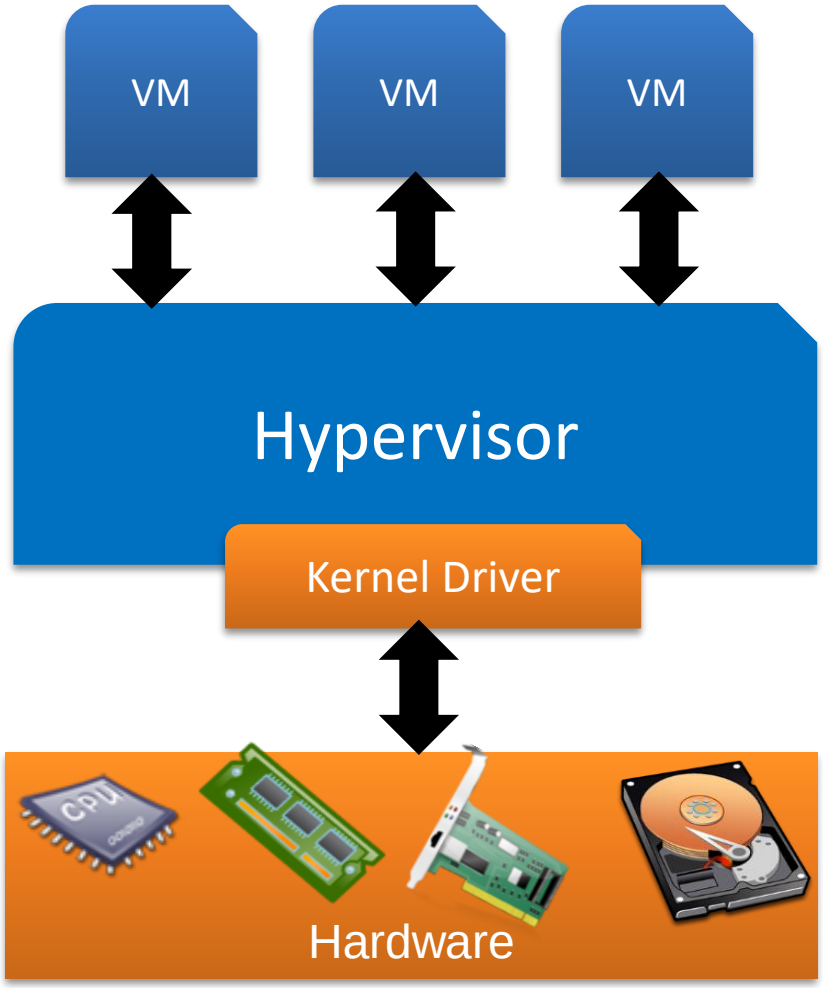
HYPERVISOR TYPE



Para virtualization

- Involves explicitly modifying guest operating system (e.g. SUSE Linux Enterprise Server 11) so that it is aware of being virtualized to allow near native performance.
- Improves performance.
- Lower overhead.
- E.g.: Xen supports both Hardware Assisted Virtualization (HVM) and Para-Virtualization (PV).

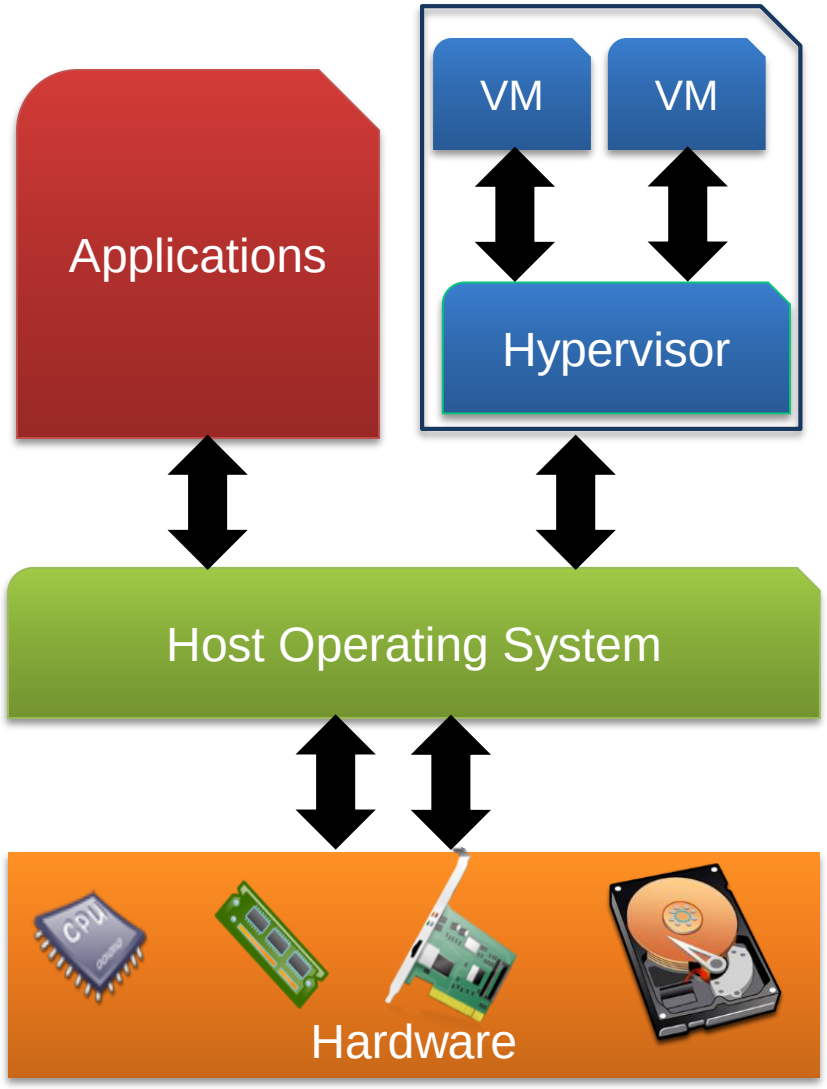
Hypervisor implementation approaches



Bare metal Approach

- Type I Hypervisor.
- Runs directly on the system hardware.
- May require hardware assisted virtualization technology support by the CPU.
- Limited set of hardware drivers provided by the hypervisor vendor.
- E.g.: Xen, VMWare ESXi

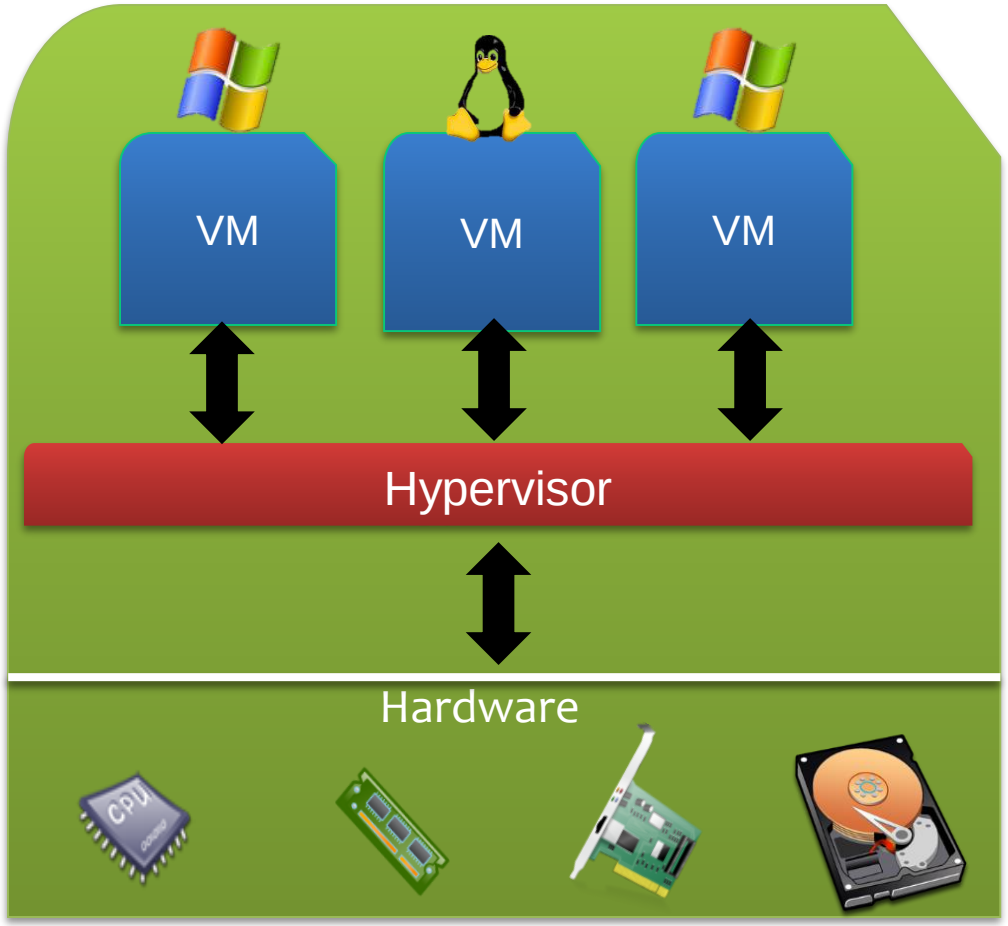
Hypervisor implementation approaches



Hosted Approach

- Type II Hypervisor.
- Runs virtual machines on top of a host OS (windows, Unix etc.)
- Relies on host OS for physical resource management.
- Host operating system provides drivers for communicating with the server hardware.
- E.g.: VirtualBox

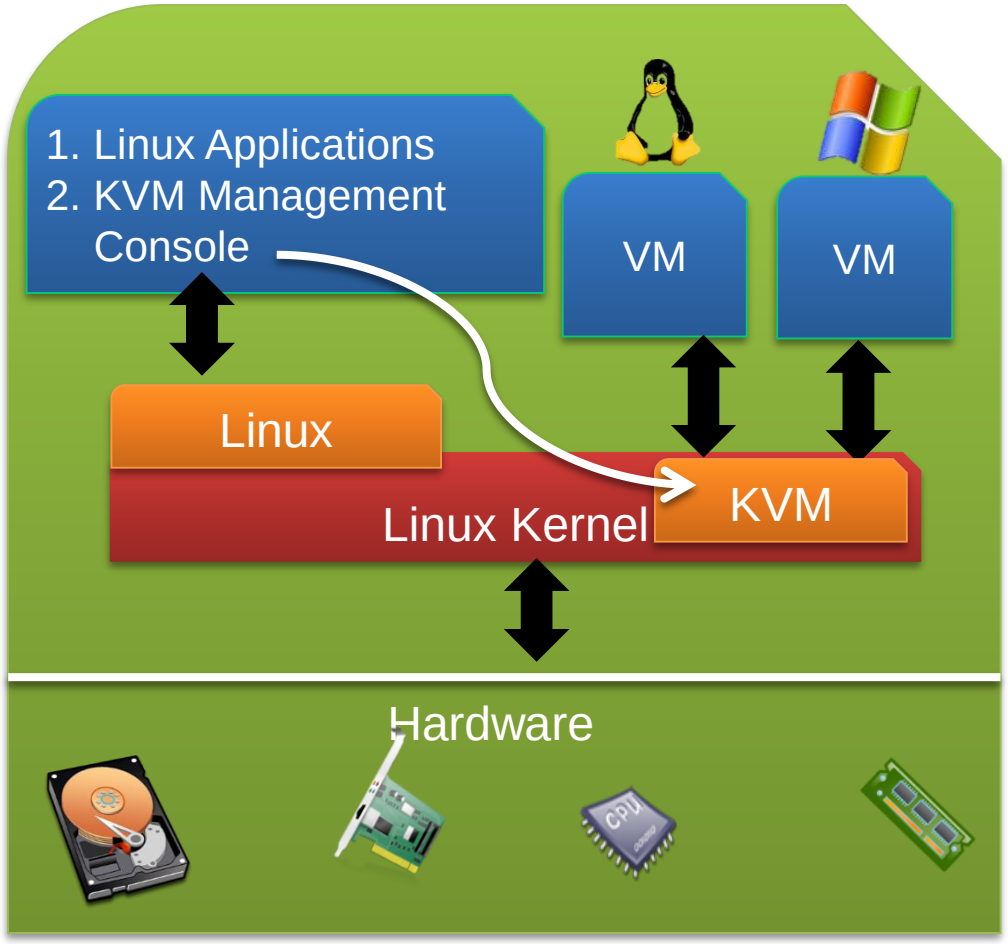
Vmware ESXi



Architecture of VMWare ESXi

- Bare Metal Approach.
- Full virtualization.
- Proven technology.
- Used for secure and robust virtualization solutions for virtual data centers and cloud infrastructures.
- Takes advantage of support for hardware assisted virtualization for 64-bit OS on Intel processors.

Ubuntu KVM



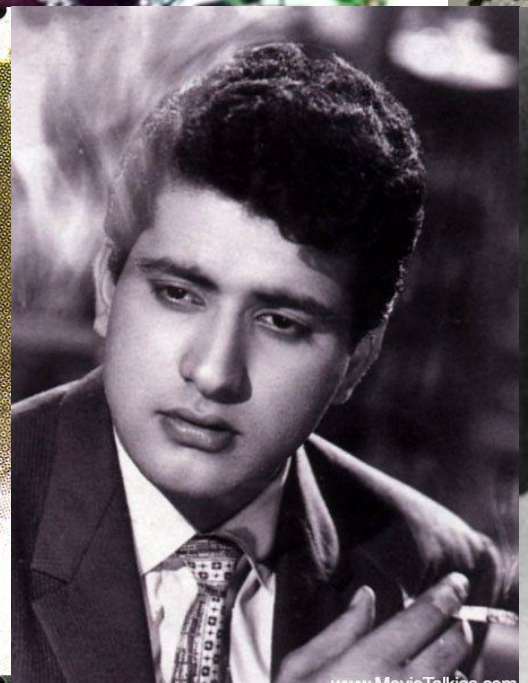
- Kernel based virtual machine (Kernel Based VM)
- Open source.
- Kernel-level extension to Linux.
- Full virtualization.
- Supports full virtualization and hence does not need hardware assisted virtualization support in the CPU.

CONTAINERIZATION

Lets go back to pre-1960’s


AMBULANCE


500
भारत INDIA
प्रेम नजीर
PREM NAZIR
2013



Cargo Transport Pre-1960

Multiplicity of Goods



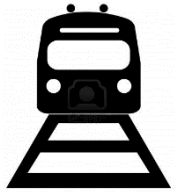


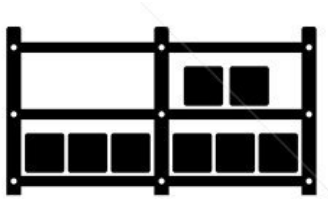


Do I worry about how goods interact (e.g. coffee beans next to spices)

Multiplicity of methods for transporting/storing

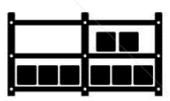





Can I transport quickly and smoothly (e.g. from boat to train to truck)

Also an M x N Matrix

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Intermodal Shipping Container

Multiplicity of Goods





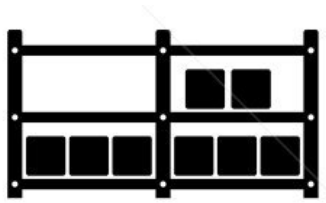


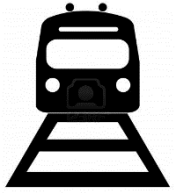
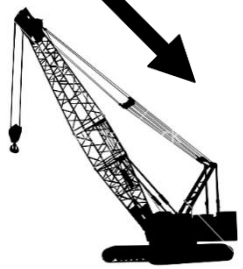
A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

Do I worry about how goods interact (e.g. coffee beans next to spices)

…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

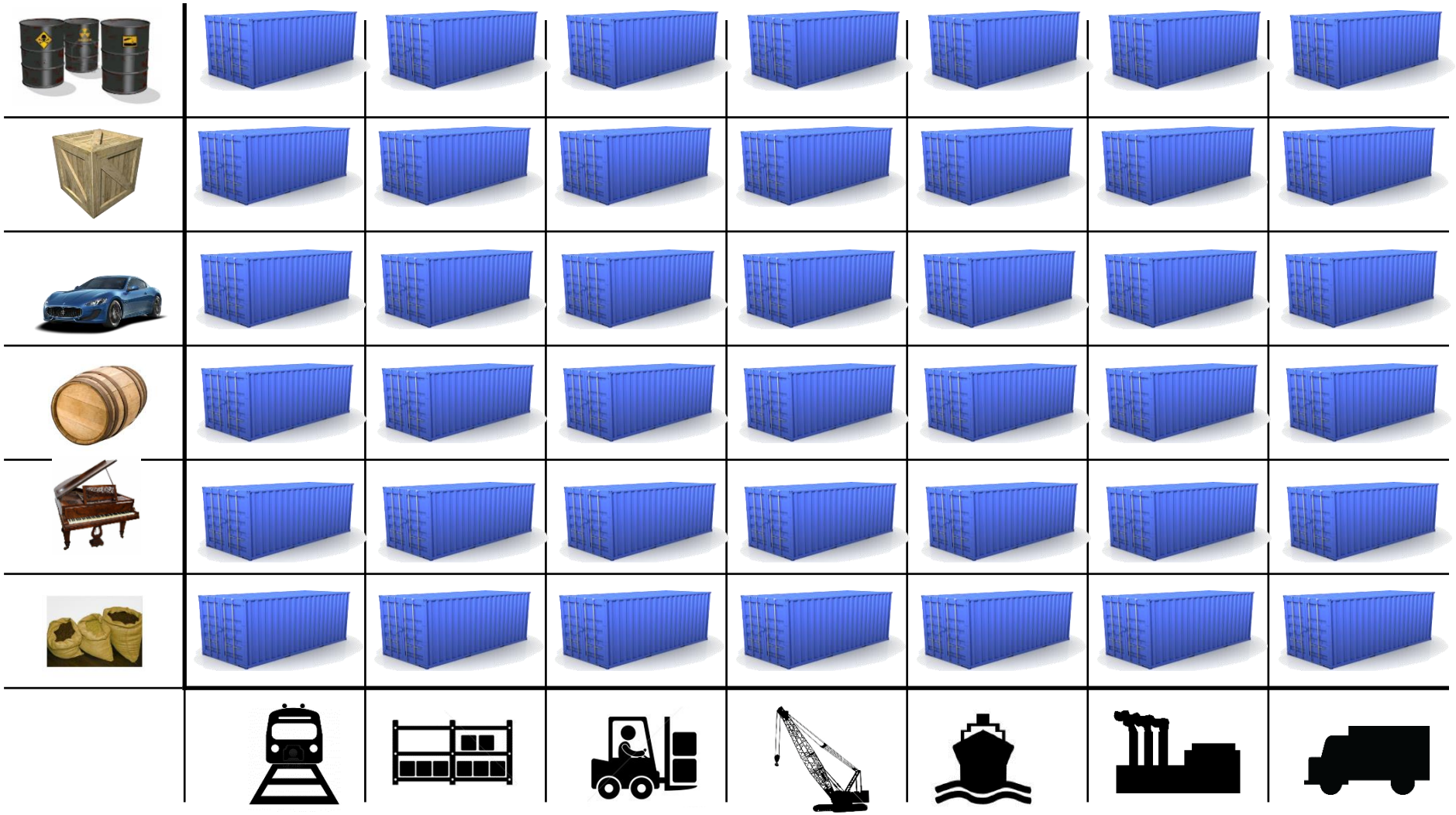
Multiplicity of methods for transporting/storing

Can I transport quickly and smoothly (e.g. from boat to train to truck)

This eliminated the M x N problem…



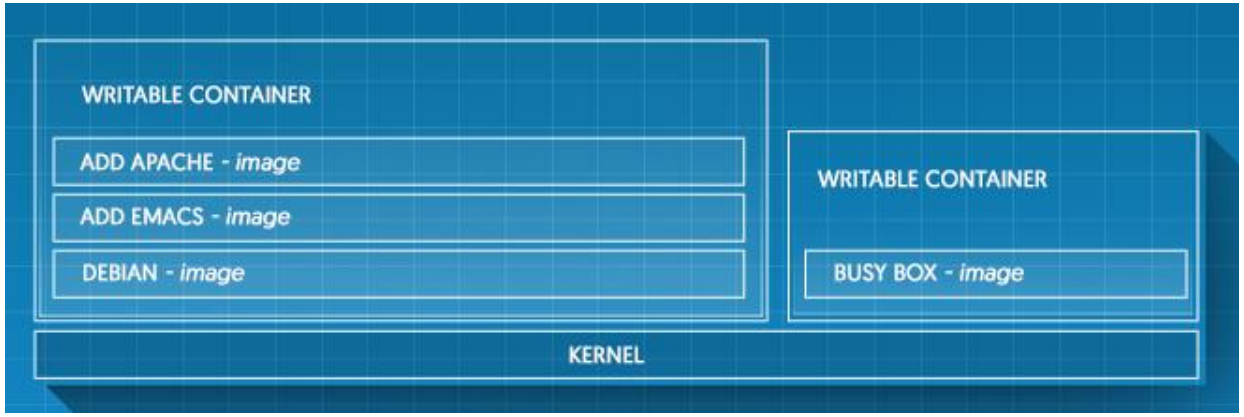
and spawned an Intermodal Shipping Container Ecosystem

- 90% of all cargo now shipped in a standard container
- Order of magnitude reduction in cost and time to load and unload ships
- Massive reduction in losses due to theft or damage
- Huge reduction in freight cost as percent of final goods (from >25% to <3%) massive globalizations
- 5000 ships deliver 200M containers per year

What is Docker?

Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.

[www.docker.com]



http://altinvesthq.com/news/wp-content/uploads/2015/06/container-ship.jpg

श्रद्धावान् लभते ज्ञानम्

What is Docker?

- Developers use Docker to eliminate “works on my machine” problems when collaborating on code with co-workers.
- Operators use Docker to run and manage apps side-by-side in isolated containers to get better compute density.
- Enterprises use Docker to build agile software delivery pipelines to ship new features faster, more securely and with confidence for both Linux and Windows Server apps.

[www.docker.com]

The Challenge

Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

Multiplicity of hardware environments

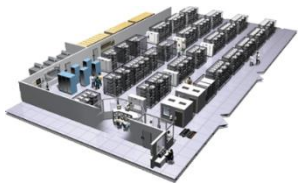
Development VM

QA server

Customer Data Center

Public Cloud

Disaster recovery



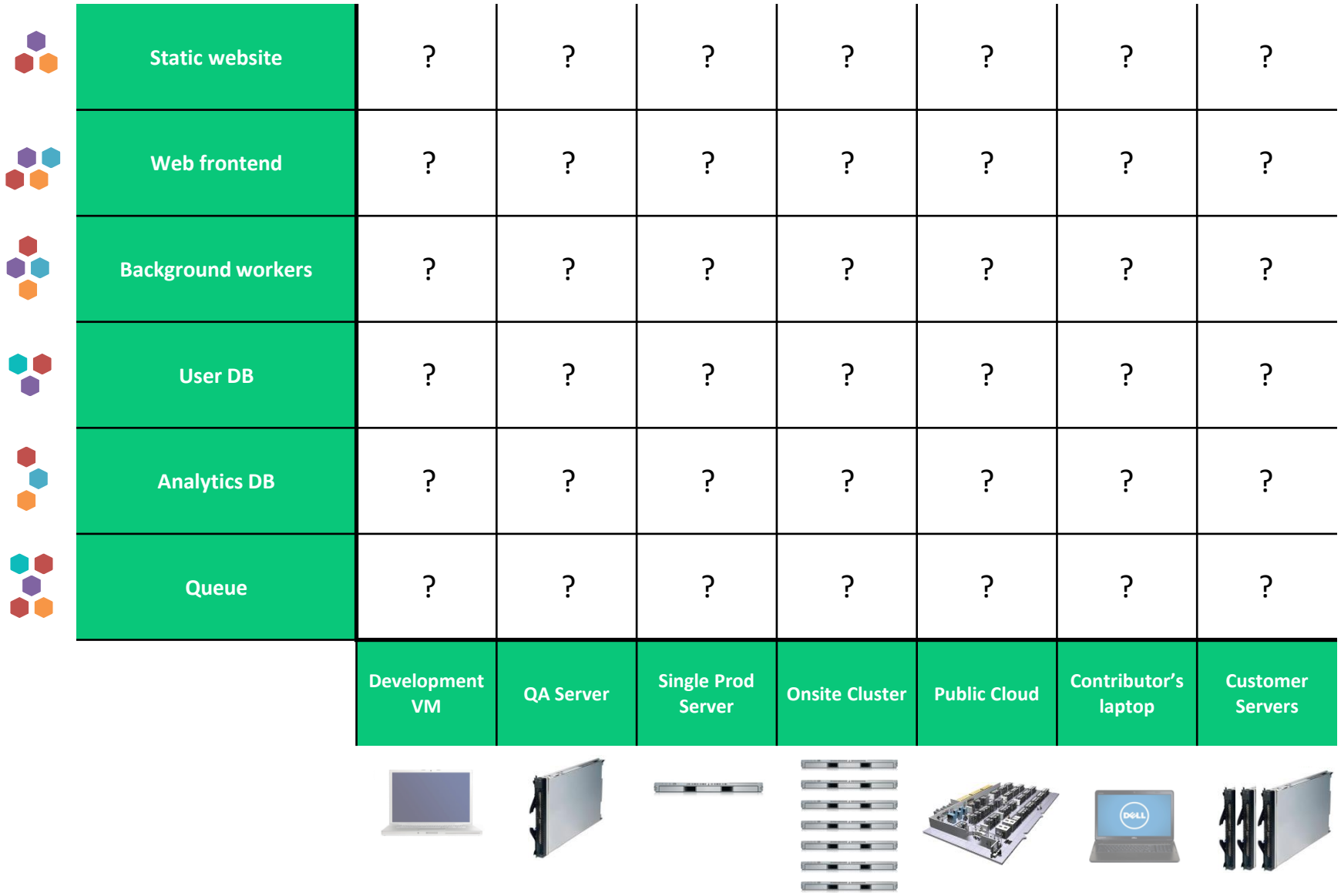
Production Cluster

Production Servers

Contributor's laptop

Can I migrate smoothly and quickly?

Results in M x N compatibility nightmare



	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Docker is a shipping container system for code

Multiplicity of Stacks

Static website

User DB

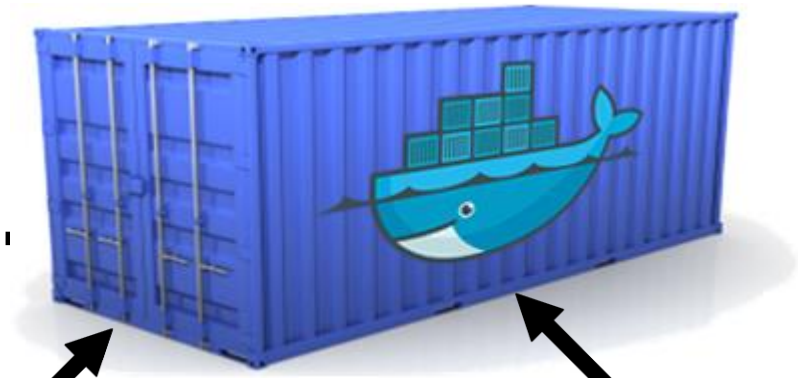
Web frontend

Queue

Analytics DB

Do services and apps interact appropriately?

An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…



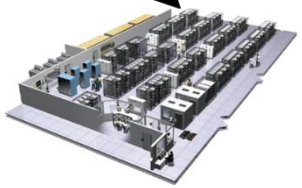
…that can be manipulated using standard operations and run consistently on virtually any hardware platform

Multiplicity of hardware environments

Development VM

QA server

Customer Data Center



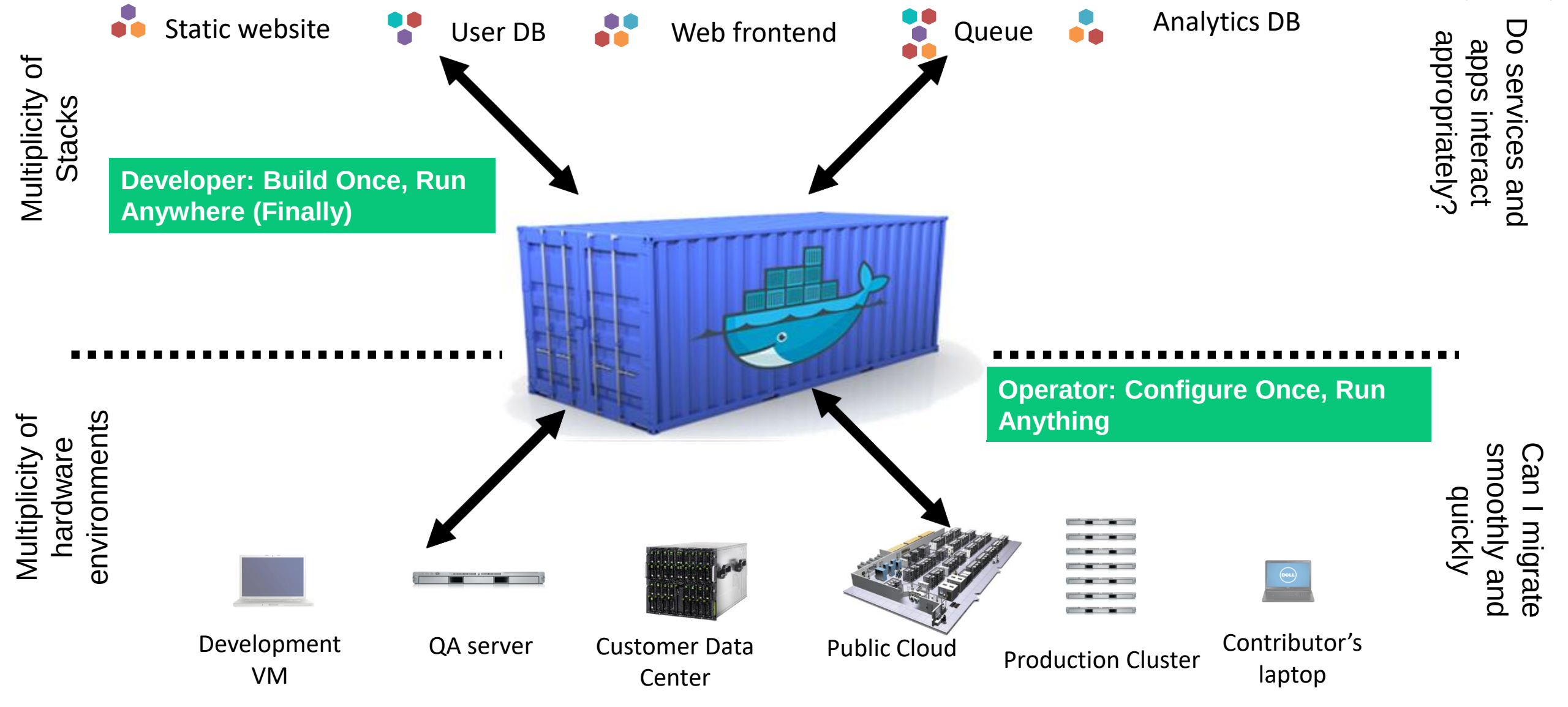
Public Cloud

Production Cluster

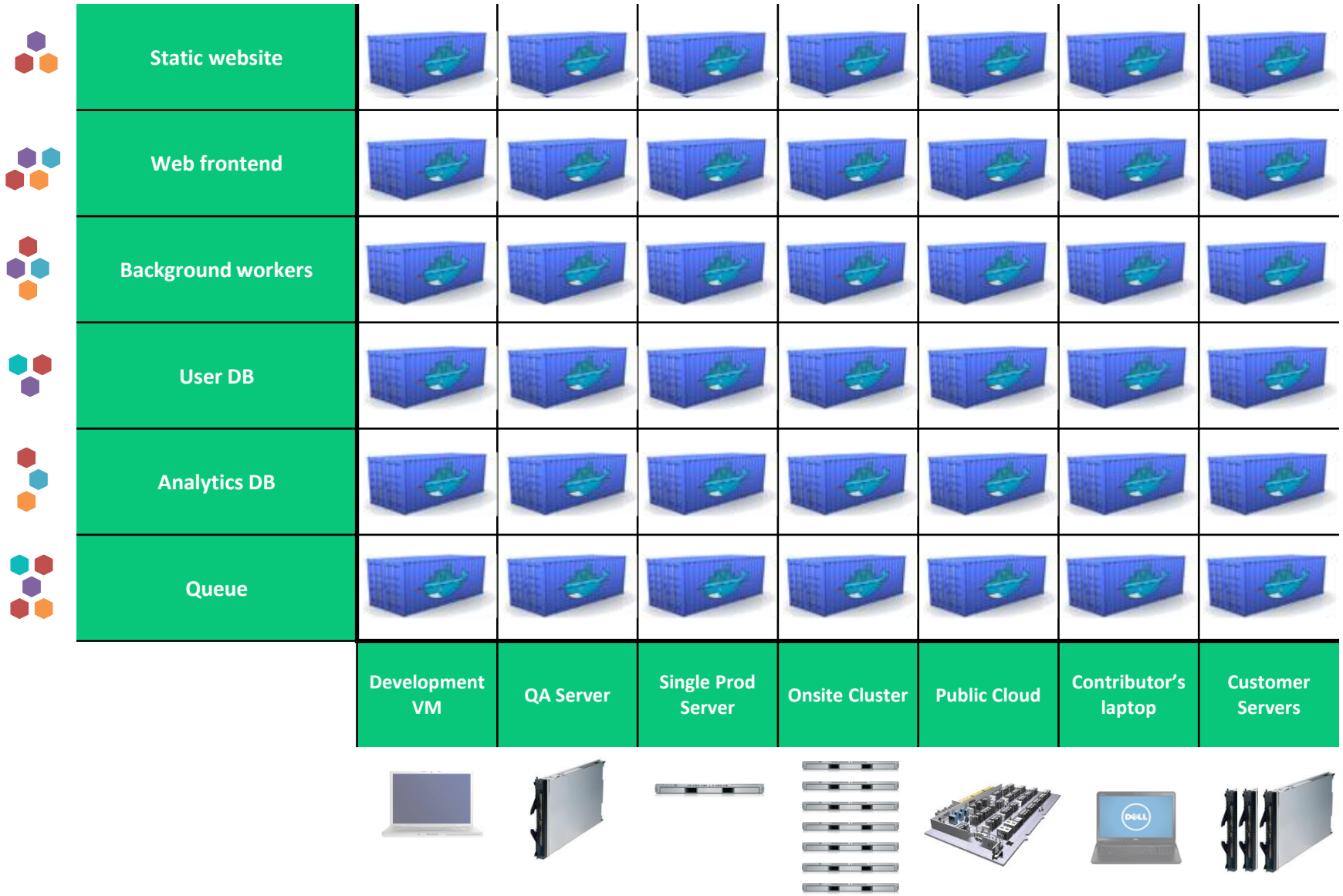
Contributor's laptop

Can I migrate smoothly and quickly

Or…put more simply

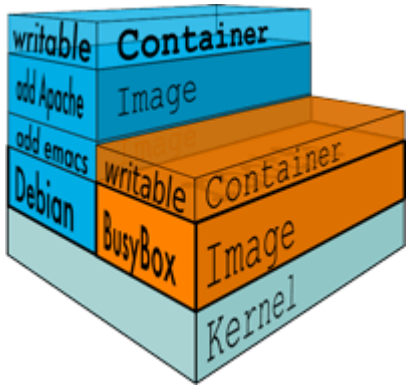


Docker solves the M x N problem



Docker containers

- Wrap up a piece of software in a complete file system that contains everything it needs to run:
  - Code, runtime, system tools, system libraries
  - Anything you can install on a server
- This guarantees that it will always run the same, regardless of the environment it is running in



Why containers matter

	Physical Containers	Docker
Content Agnostic	The same container can hold almost any type of cargo	Can encapsulate any payload and its dependencies
Hardware Agnostic	Standard shape and interface allow same container to move from ship to train to semi-truck to warehouse to crane without being modified or opened	Using operating system primitives (e.g. LXC) can run consistently on virtually any hardware—VMs, bare metal, openstack, public IAAS, etc.—without modification
Content Isolation and Interaction	No worry about anvils crushing bananas. Containers can be stacked and shipped together	Resource, network, and content isolation. Avoids dependency hell
Automation	Standard interfaces make it easy to automate loading, unloading, moving, etc.	Standard operations to run, start, stop, commit, search, etc. Perfect for devops: CI, CD, autoscaling, hybrid clouds
Highly efficient	No opening or modification, quick to move between waypoints	Lightweight, virtually no perf or start-up penalty, quick to move and manipulate
Separation of duties	Shipper worries about inside of box, carrier worries about outside of box	Developer worries about code. Ops worries about infrastructure.

Docker containers

Lightweight

- Containers running on one machine all share the same OS kernel
- They start instantly and make more efficient use of RAM
- Images are constructed from layered file systems
- They can share common files, making disk usage and image downloads much more efficient

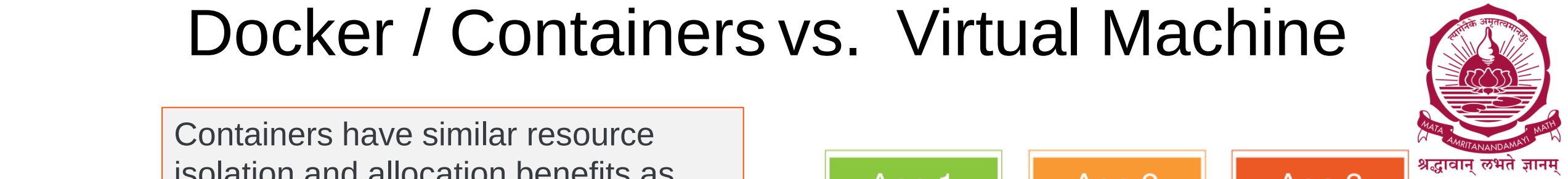
Open

- Based on open standards
- Allowing containers to run on all major Linux distributions and Microsoft OS with support for every infrastructure

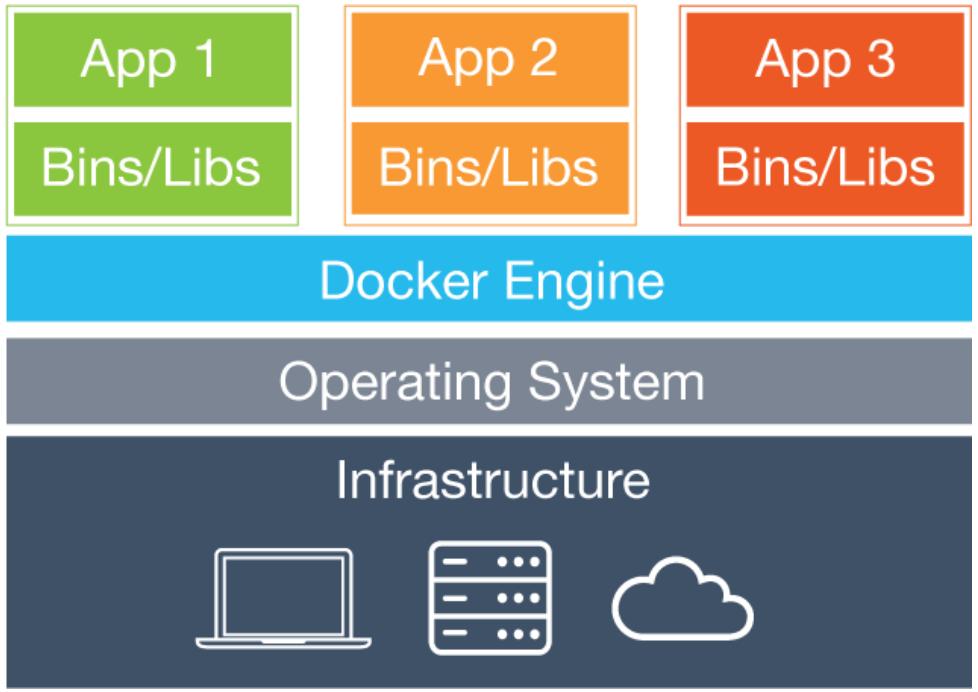
Secure

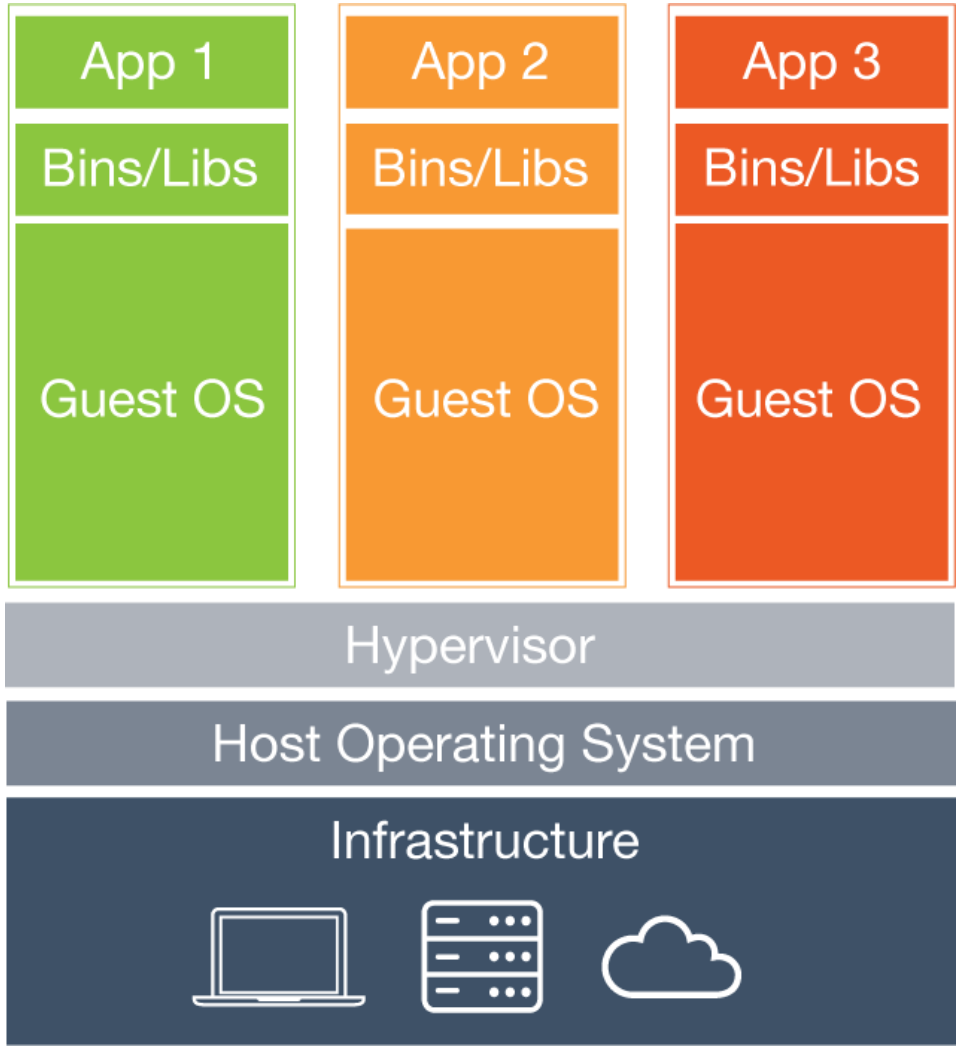
- Containers isolate applications from each other and the underlying infrastructure while providing an added layer of protection for the application

Docker / Containers vs. Virtual Machine



Containers have similar resource isolation and allocation benefits as VMs but a different architectural approach allows them to be much more portable and efficient

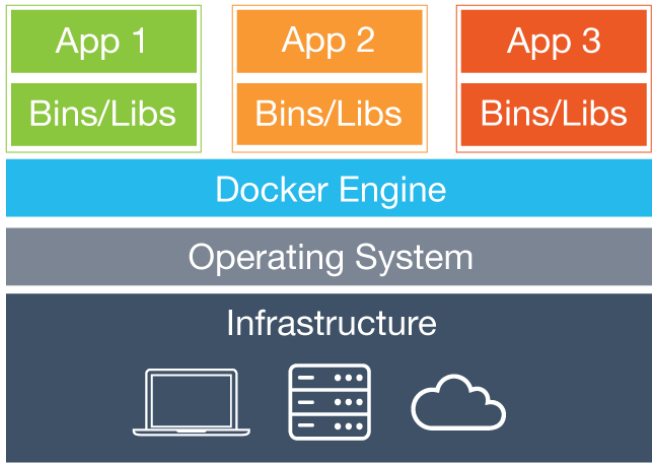




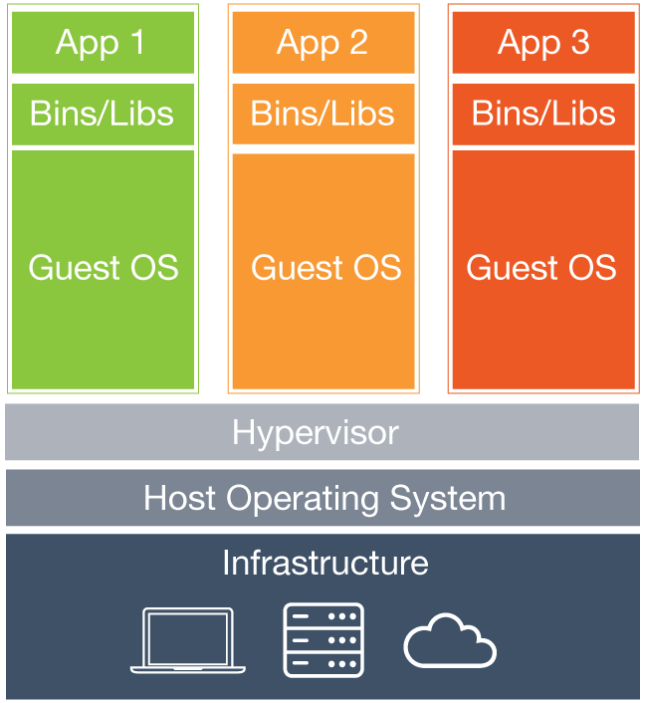
https://www.docker.com/whatisdocker/

Docker / Containers vs. Virtual Machine

- It includes the application and all of its dependencies, but share the kernel with other containers.
- They run as an isolated process in userspace on the host operating system.
- Docker containers run on any computer, on any infrastructure and in any cloud



- Each virtual machine includes the application, the necessary binaries and libraries and an entire guest operating system - all of which may be tens of GBs in size

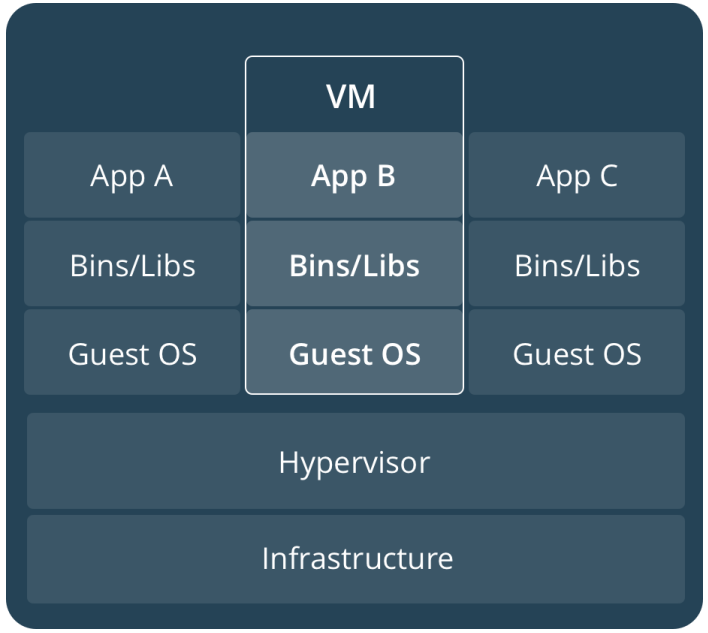


https://www.docker.com/whatisdocker/

Containers vs Virtual Machines

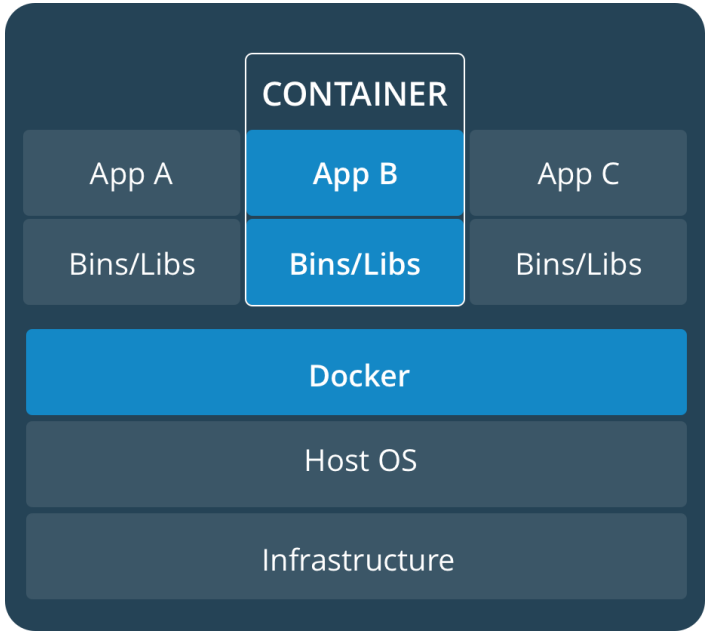
Virtual Machines

Virtual machines run guest operating systems—note the OS layer in each box. This is resource intensive, and the resulting disk image and application state is an entanglement of OS settings, system-installed dependencies, OS security patches, and other easy-to-lose, hard-to-replicate ephemera



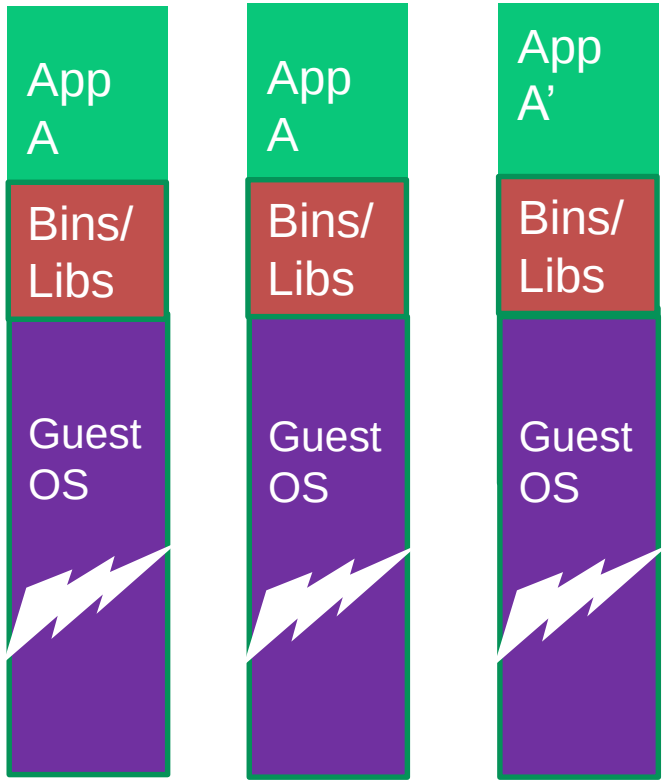
Containers

Containers can share a single kernel, and the only information that needs to be in a container image is the executable and its package dependencies, which never need to be installed on the host system. These processes run like native processes, and you can manage them individually



Why are Docker containers lightweight?

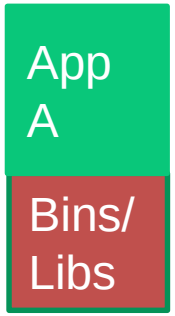
VMs



VMs

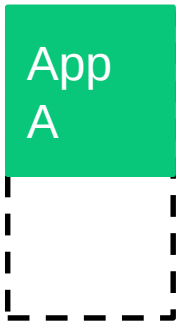
Every app, every copy of an app, and every slight modification of the app requires a new virtual server

Containers



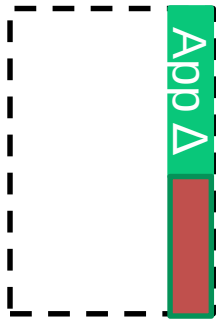
Original App

(No OS to take up space, resources, or require restart)



Copy of App

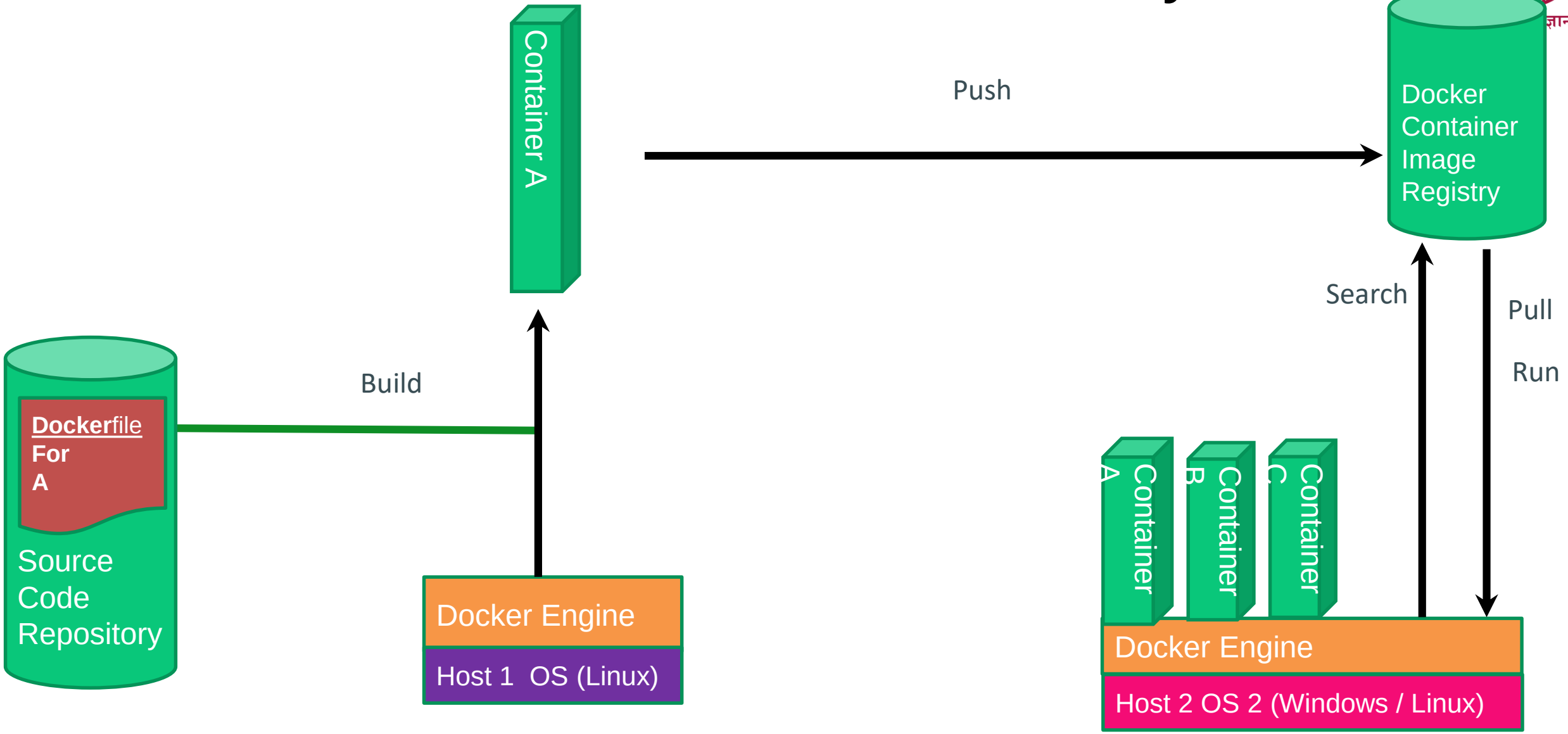
No OS. Can Share bins/libs



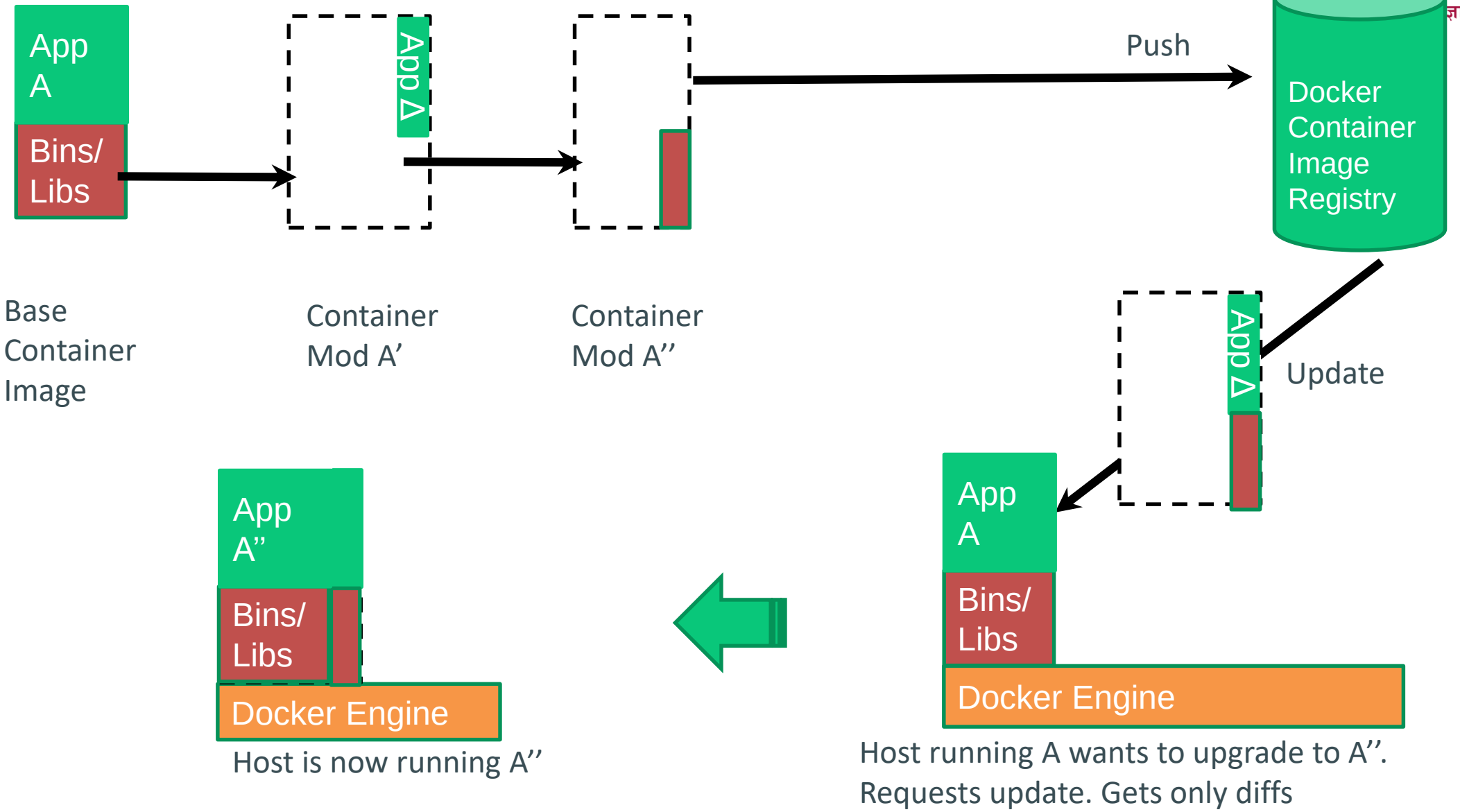
Modified App

Union file system allows us to only save the diffs Between container A and container A'

What are the basics of the Docker system?



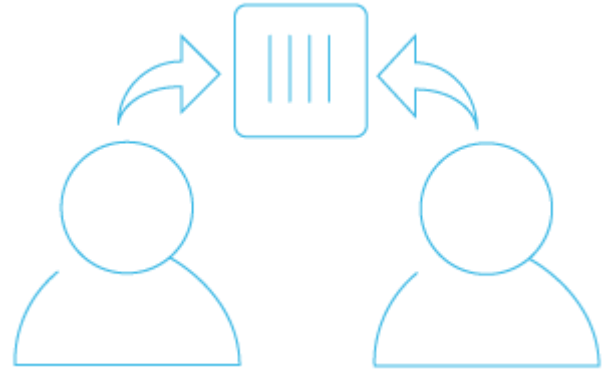
Changes and Updates



Easily Share and Collaborate on Applications

Docker creates a common framework for developers and sysadmins to work together on distributed applications

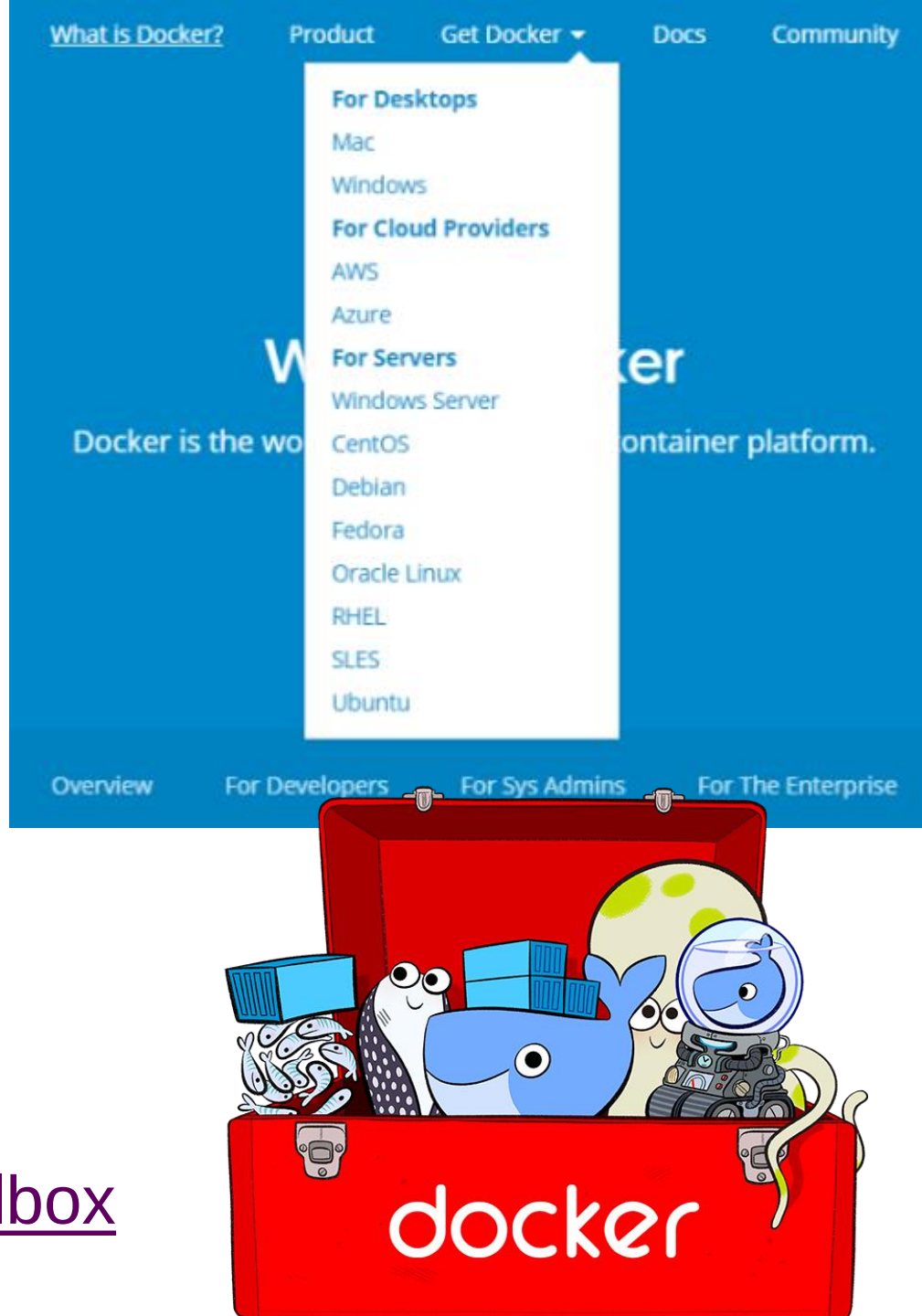
- Distribute and share content
  - Store, distribute and manage your Docker images in your Docker Hub with your team
  - Image updates, changes and history are automatically shared across your organization.
- Simply share your application with others
  - Ship your containers to others without worrying about different environment dependencies creating issues with your application.
  - Other teams can easily link to or test against your app without having to learn or worry about how it works.



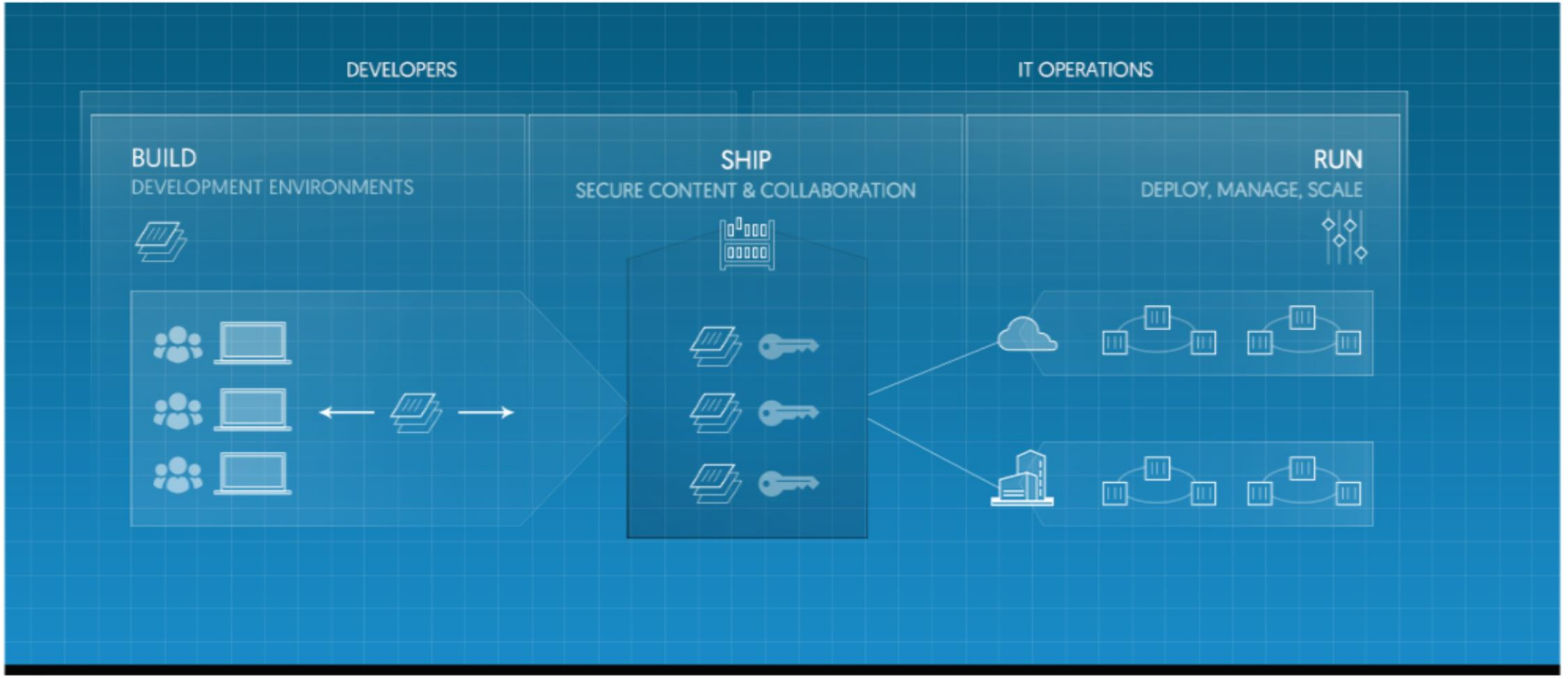
Get Started with Docker

- Install Docker
- Run a software image in a container
- Browse for an image on Docker Hub
- Create your own image and run it in a container
- Create a Docker Hub account and an image repository
- Create an image of your own
- Push your image to Docker Hub for others to use

https://www.docker.com/products/docker
https://www.docker.com/products/docker-toolbox



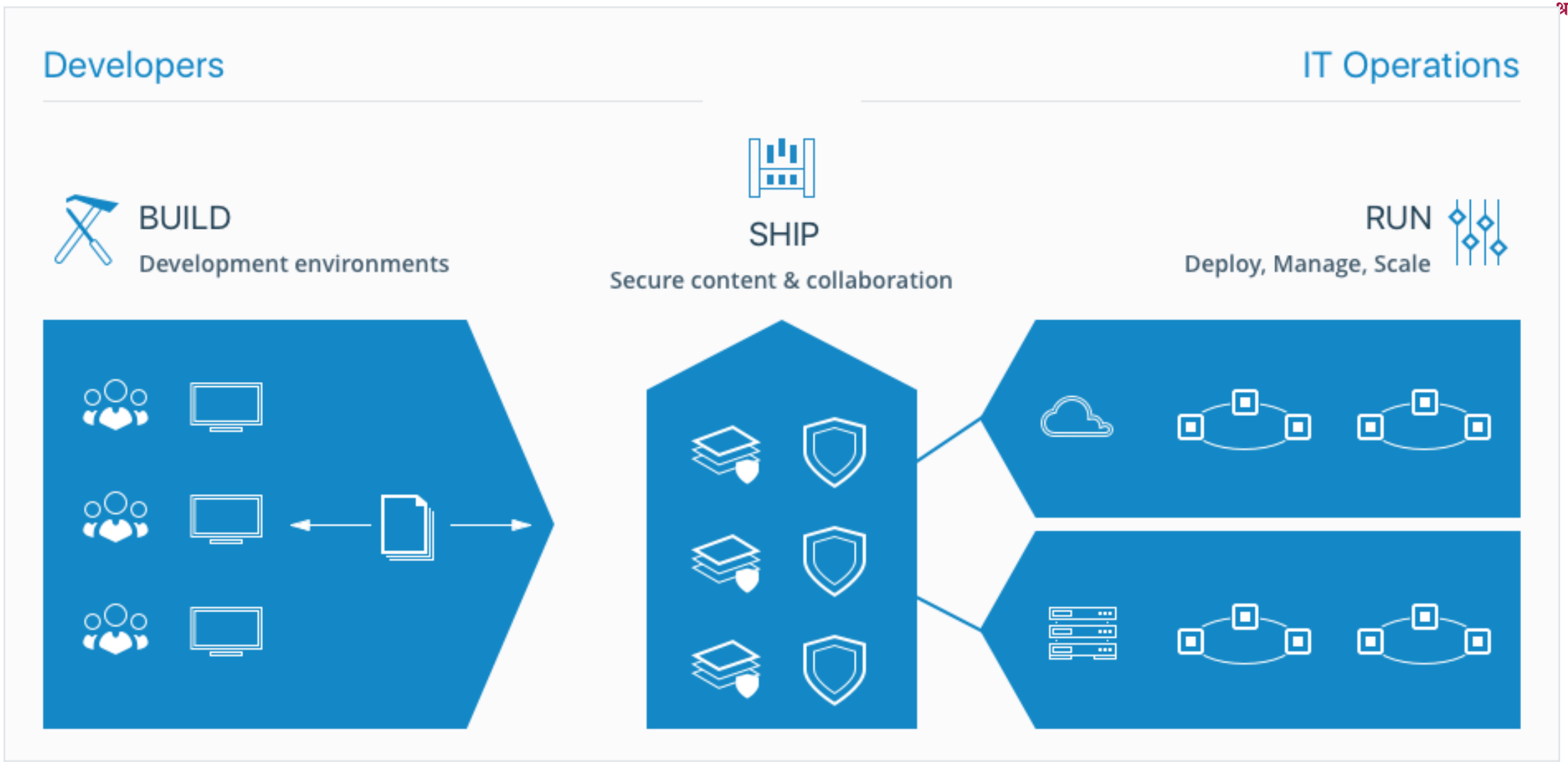
Docker Container as a Service (CaaS)



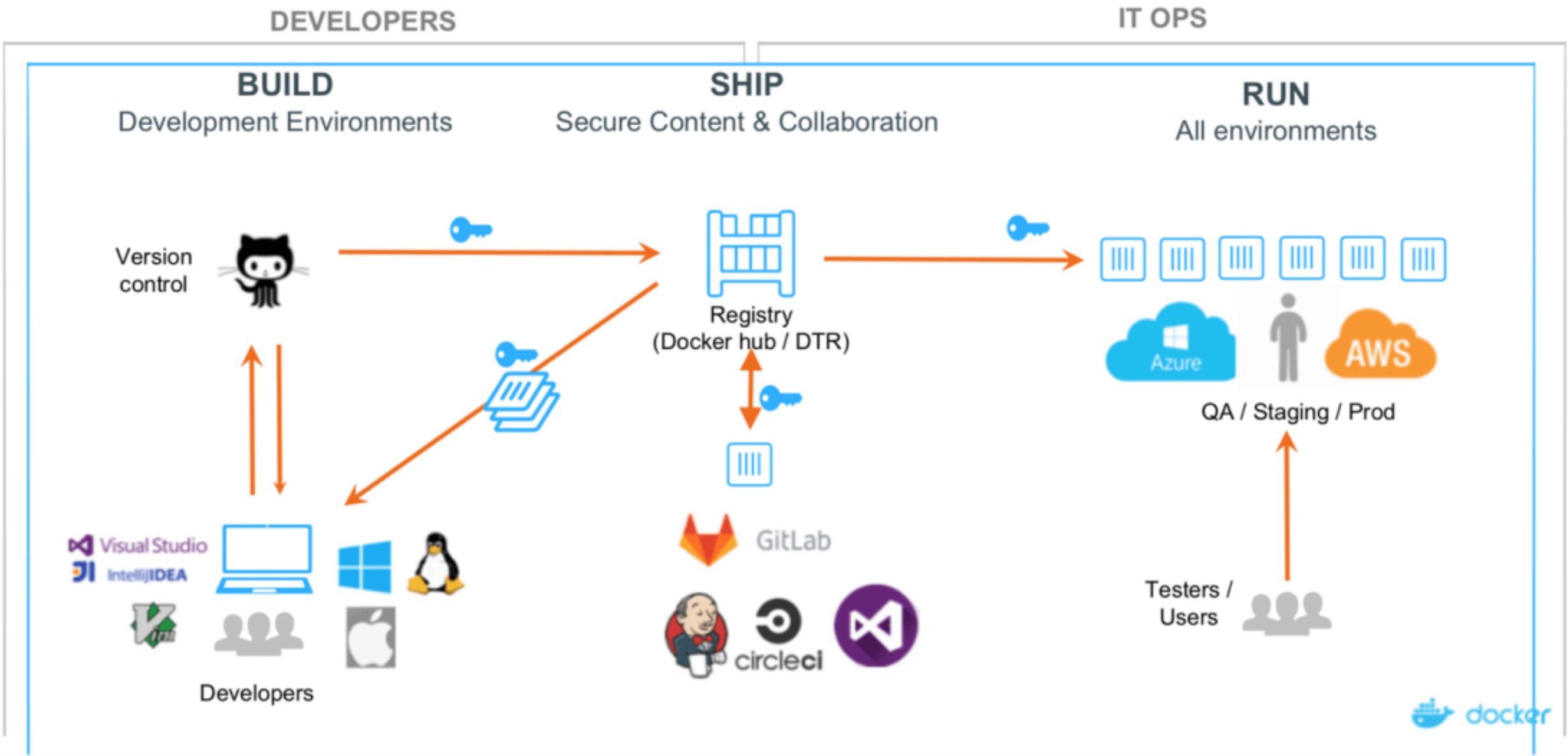
Deliver an IT secured and managed application environment for developers to build and deploy applications in a self service manner

Typical Use cases

App Modernization



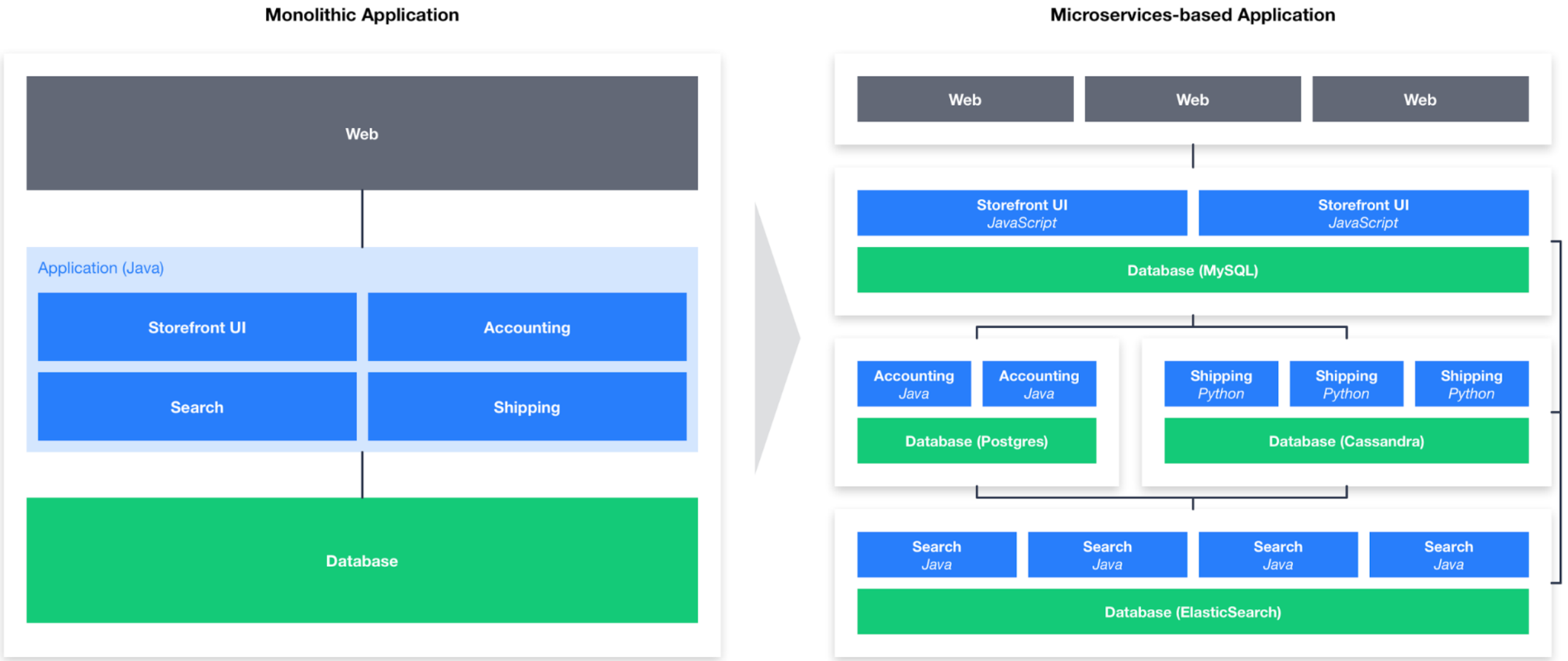
Continuous Integration & Delivery Workflow



Continuous Integration and Deployment (CI / CD)

- The modern development pipeline is fast, continuous and automated with the goal of more reliable software
- CI/CD allows teams to integrate new code as often as every time code is checked in by developers and passes testing
- A cornerstone of devops methodology, CI/CD creates a real time feedback loop with a constant stream of small iterative changes that accelerates change and improves quality
- CI environments are often fully automated to trigger a test at git push and to automatically build a new image if the test is successful and push to a Docker Registry
- Further automation and scripting can deploy a container from the new image to staging for further testing.

Microservices

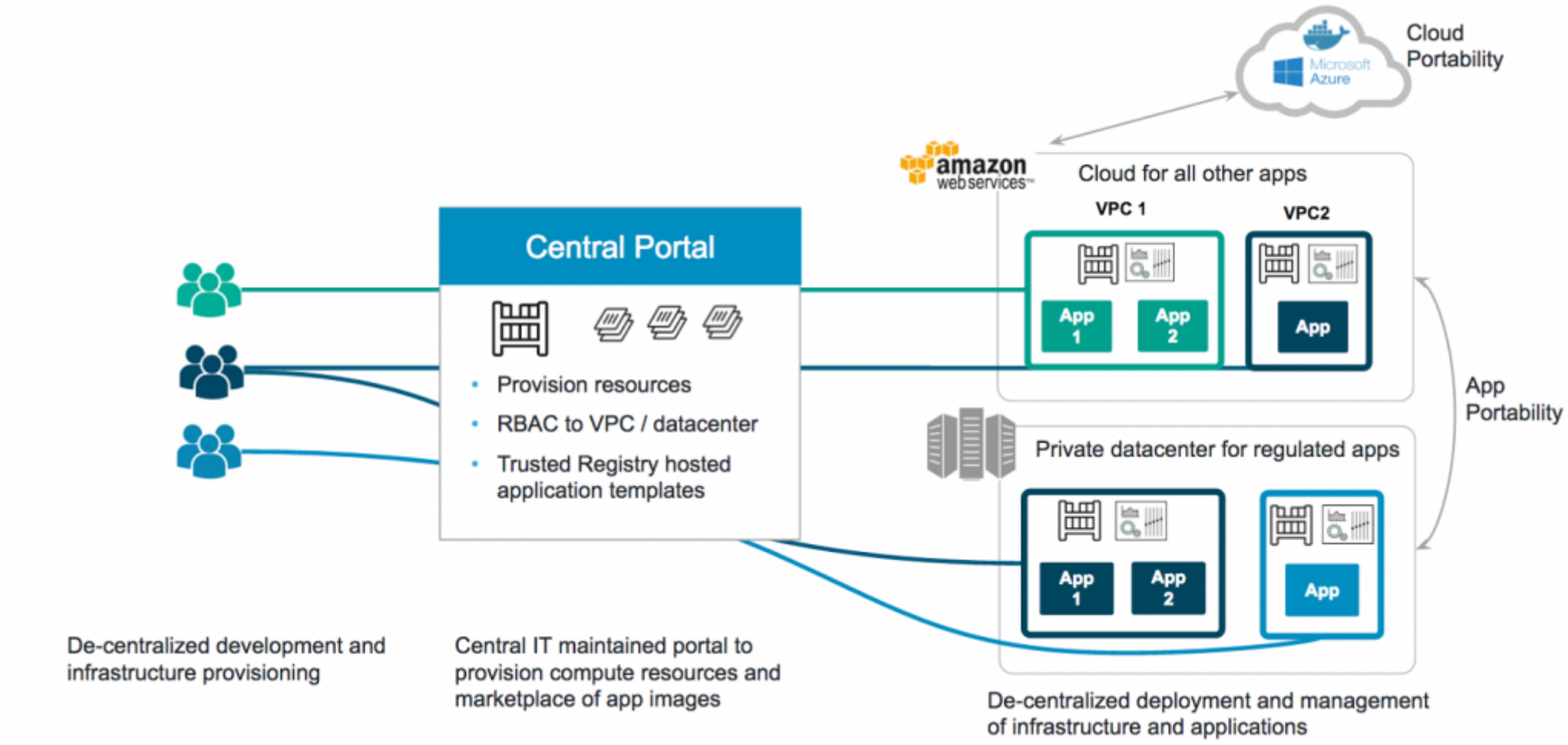


https://mesosphere.com/blog/networking-docker-containers-part-ii-service-discovery-traditional-apps-microservices/

Microservices

- App architecture is changing from monolithic code bases with waterfall development methodologies to loosely coupled services that are developed and deployed independently
- Tens to thousands of these services can be connected to form an app
- Docker allows developers are able to choose the best tool or stack for each service and isolates them to eliminate any potential conflicts and avoids the “matrix from hell.”
- These containers can be easily shared, deployed, updated and scaled instantly and independently of the other services that make up the app
- Docker’s end to end security features allow teams to build and operate a least privilege microservices model where services only get access to the resources (other apps, secrets, compute) they need to run at just the right time to create.

Hybrid Cloud



https://boxboat.com/2016/10/21/maintaining-docker-portability-multi-cloud-world/

Hybrid Cloud

- Docker guarantees apps are cloud enabled - ready to move across private and public clouds with a higher level of control and guarantee apps will operate as designed
- The Docker platform is infrastructure independent and ensures everything the app needs to run is packaged and transported together from one site to another
- Docker uniquely provides flexibility and choice for businesses to adopt a single, multi or hybrid cloud environment without conflict

IT Infrastructure optimization

- Docker and containers help optimize the utilization and cost of your IT infrastructure
- Optimization not just cost reduction, it is ensuring the right amount of resources are available at the right time and used efficiently
- Because containers are lightweight ways of packaging and isolating app workloads, Docker allows multiple workloads to run on the same physical or virtual server without conflict
- Businesses can consolidate datacenters, integrate IT from mergers and acquisitions and enable portability to cloud while reducing the footprint of operating systems and servers to maintain

How does this help you build better software?

Accelerate Developer Onboarding

- Stop wasting hours trying to setup developer environments
- Spin up new instances and make copies of production code to run locally
- With Docker, you can easily take copies of your live environment and run on any new endpoint running Docker.

Empower Developer Creativity

- The isolation capabilities of Docker containers free developers from the worries of using “approved” language stacks and tooling
- Developers can use the best language and tools for their application service without worrying about causing conflict issues

Eliminate Environment Inconsistencies

- By packaging up the application with its configs and dependencies together and shipping as a container, the application will always work as designed locally, on another machine, in test or production
- No more worries about having to install the same configs into a different environment


ADP
amadeus
badoo
BBC NEWS
BUSINESS INSIDER
CORNELL UNIVERSITY
FOUNDED A.D. 1865
doubledutch
ebay
EURECOM
Sophia Antipolis
Expedia
GE Appliances
OXFORD
UNIVERSITY PRESS
PayPal
ING
GROUPON
The Washington Post
UBER
yelp


First Hand Experience

Setting up

- Before we get started, make sure your system has the latest version of Docker installed.
- Docker is available in two editions: **Community Edition (CE)** and **Enterprise Edition (EE)**.
- Docker Community Edition (CE) is ideal for developers and small teams looking to get started with Docker and experimenting with container-based apps. Docker CE has two update channels, **stable** and **edge**:
  - **Stable** gives you reliable updates every quarter
  - **Edge** gives you new features every month
- Docker Enterprise Edition (EE) is designed for enterprise development and IT teams who build, ship, and run business critical applications in production at scale.

Supported Platforms

https://docs.docker.com/install/

Desktop

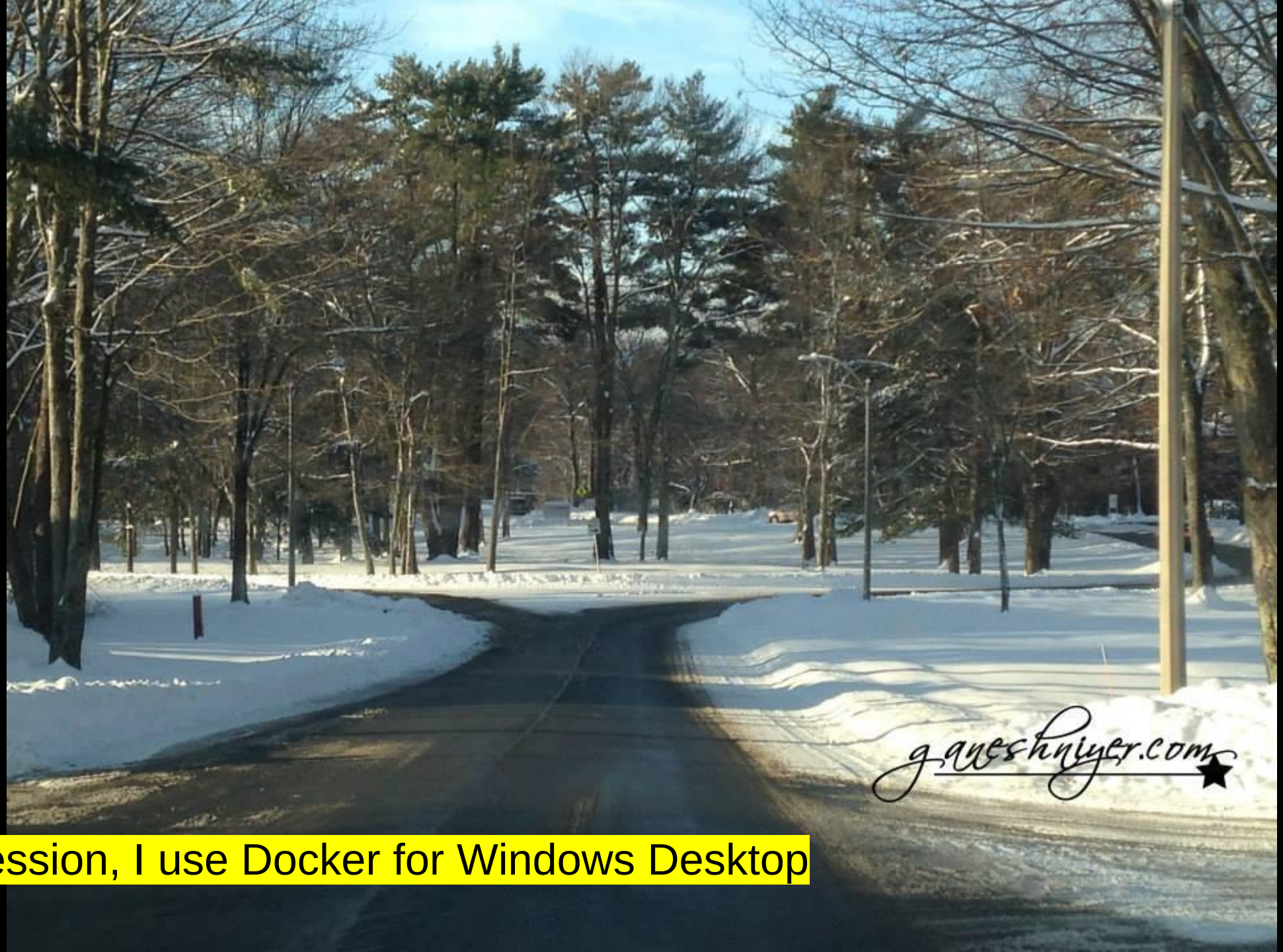
Platform	Docker CE x86_64	Docker CE ARM	Docker EE
Docker for Mac (macOS)	✓		
Docker for Windows (Microsoft Windows 10)	✓		

Cloud

Platform	Docker CE x86_64	Docker CE ARM	Docker EE
Amazon Web Services	✓		✓
Microsoft Azure	✓		✓

Server

Platform	Docker CE x86_64	Docker CE ARM	Docker CE IBM Z (s390x)	Docker EE x86_64	Docker EE IBM Z (s390x)
CentOS	✓			✓	
Debian	✓	✓			
Fedora	✓				
Microsoft Windows Server 2016				✓	
Oracle Linux				✓	
Red Hat Enterprise Linux				✓	✓
SUSE Linux Enterprise Server				✓	✓
Ubuntu	✓	✓	✓	✓	✓



In this session, I use Docker for Windows Desktop

Docker for Windows

Docker for Windows is a Docker Community Edition (CE) app. The Docker for Windows install package includes everything you need to run Docker on a Windows system. This topic describes pre-install considerations, and how to download and install Docker for Windows.

> **Already have Docker for Windows?** If you already have Docker for Windows installed, and are ready to get started, skip to Get started with Docker for Windows for a quick tour of the command line, settings, and tools.

> **Looking for Release Notes?** Get release notes for all versions here.

Download Docker for Windows

If you have not already done so, please install Docker for Windows. You can download installers from the **Stable** or **Edge** channel.

Both Stable and Edge installers come with experimental features in Docker Engine enabled by default. Experimental mode can be toggled on and off in preferences.

We welcome your feedback to help us improve Docker for Windows.

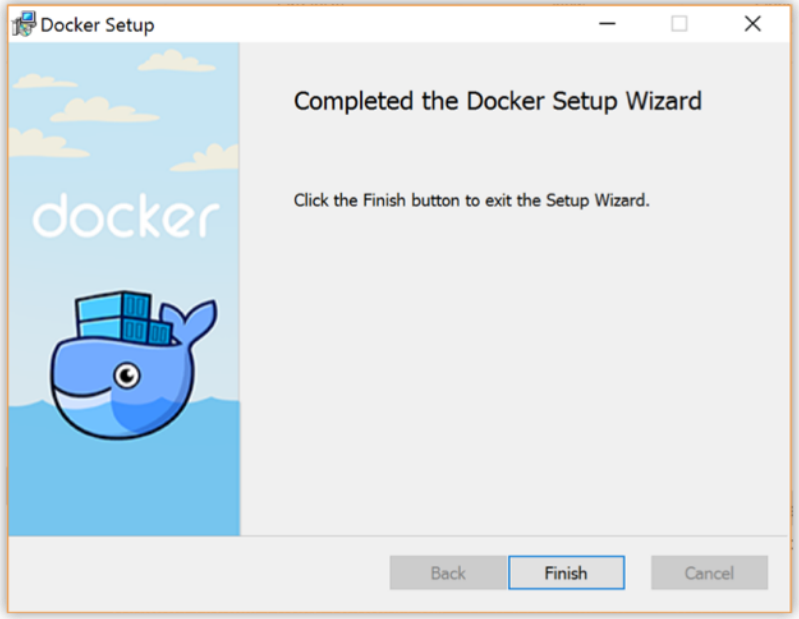
For more about Stable and Edge channels, see the FAQs.

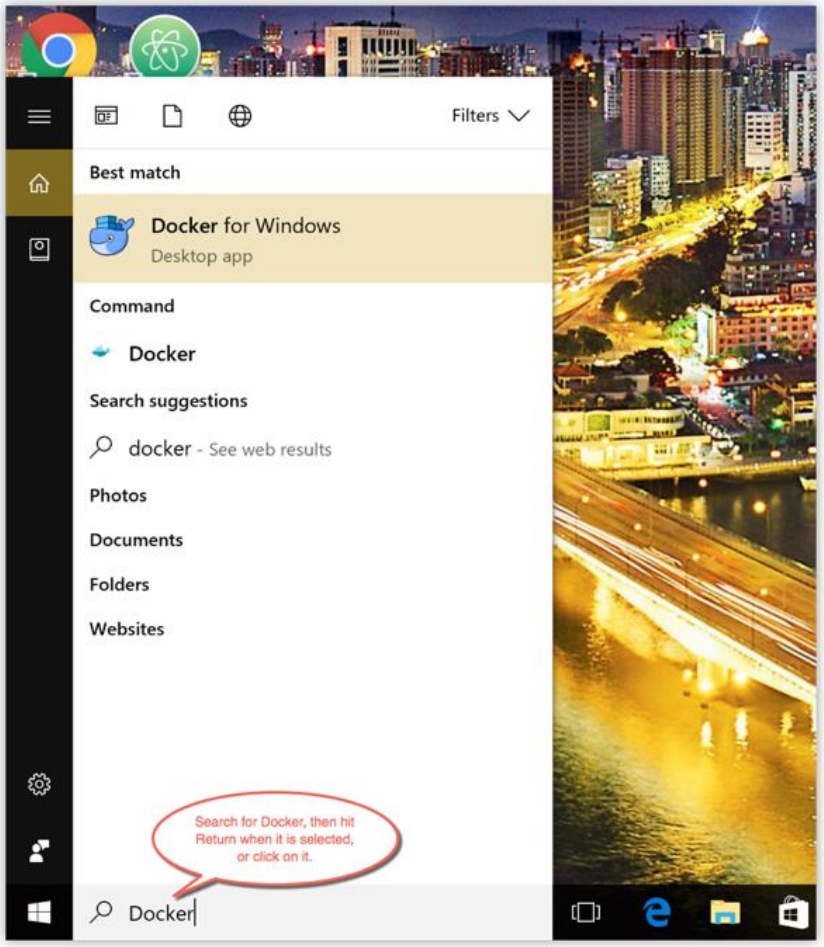
Stable channel	Edge channel
Stable is the best channel to use if you want a reliable platform to work with. Stable releases track the Docker platform stable releases. On this channel, you can select whether to send usage statistics and other data. Stable releases happen once per quarter.	Use the Edge channel if you want to get experimental features faster, and can weather some instability and bugs. We collect usage data on Edge releases. Edge builds are released once per month.
Get Docker for Windows (Stable)	Get Docker for Windows (Edge)

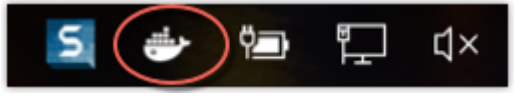
If your windows is not in latest version…

https://docs.docker.com/docker-for-windows/release-notes/#docker-community-edition-17062-ce-win27-2017-09-06-stable

Docker for Windows







When the whale in the status bar stays steady, Docker is up-and-running, and accessible from any terminal window.

Hello-world

- Open command prompt / windows power shell and run

`docker run hello-world`

```
PS C:\Users\gaiyer\dockertest> docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

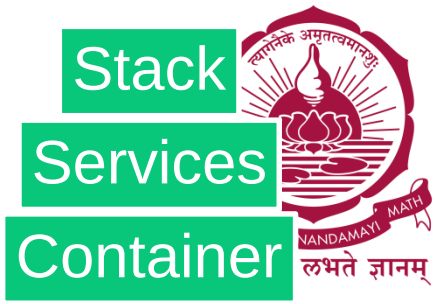
- Now would also be a good time to make sure you are using version 1.13 or higher. Run `docker --version` to check it out.

```
PS C:\Users\gaiyer\dockertest> docker --version
Docker version 17.06.2-ce, build cec0b72
```

Building an app the Docker way

- In the past, if you were to start writing a Python app, your first order of business was to install a Python runtime onto your machine
- But, that creates a situation where the environment on your machine has to be just so in order for your app to run as expected; ditto for the server that runs your app
- With Docker, you can just grab a portable Python runtime as an image, no installation necessary
- Then, your build can include the base Python image right alongside your app code, ensuring that your app, its dependencies, and the runtime, all travel together
- These portable images are defined by something called a Dockerfile

Define a container with a `Dockerfile`



- `Dockerfile` will define what goes on in the environment inside your container
- Access to resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of your system, so you have to map ports to the outside world, and be specific about what files you want to “copy in” to that environment
- However, after doing that, you can expect that the build of your app defined in this `Dockerfile` will behave exactly the same wherever it runs

Dockerfile

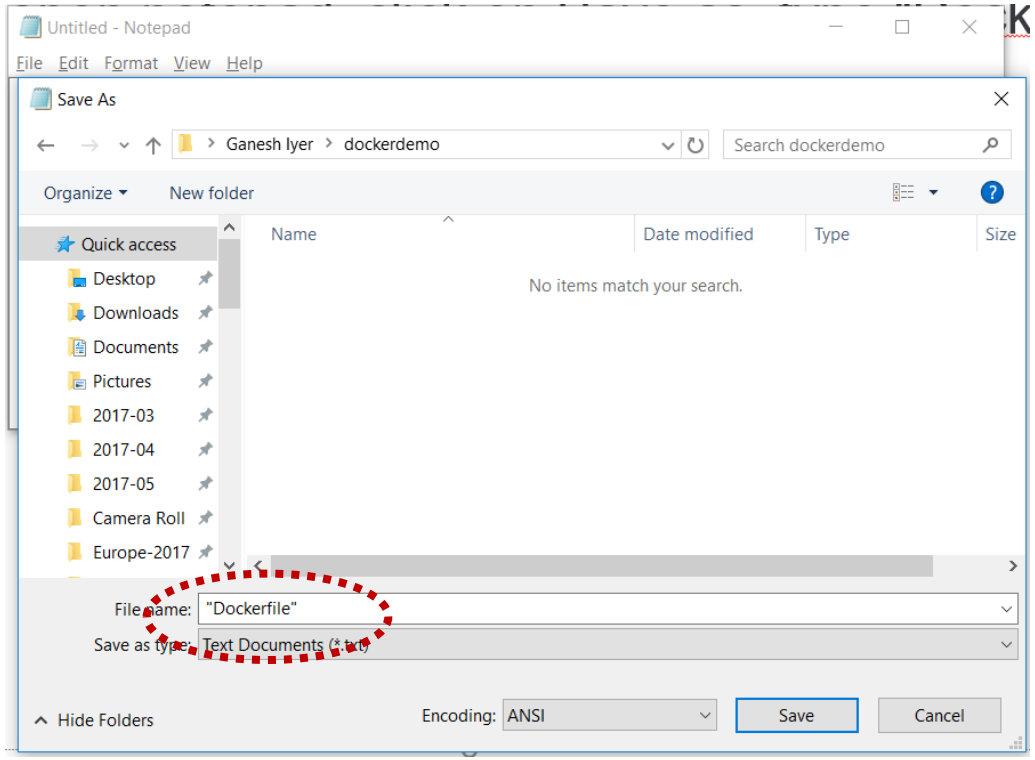
- Create an empty directory
- Change directories (cd) into the new directory, create a

```
PS C:\Users\gaiyer> mkdir dockerdemo


    Directory: C:\Users\gaiyer


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        15-Sep-17   7:16 AM                dockerdemo


PS C:\Users\gaiyer> cd dockerdemo
PS C:\Users\gaiyer\dockerdemo>
```

Dockerfile

- In windows, open notepad, copy the content below, click on Save as, type “Dockerfile”

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

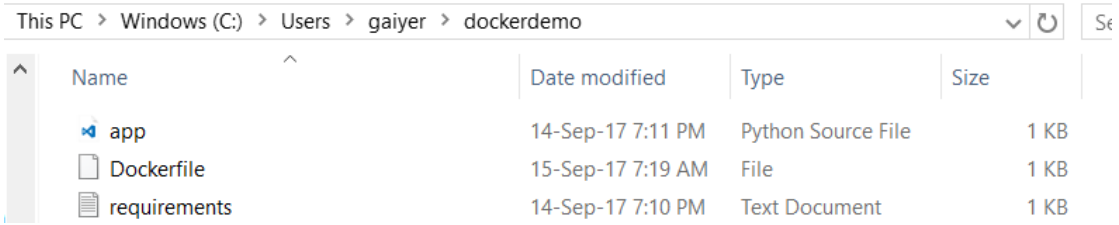


This Dockerfile refers to a couple of files we haven’t created yet, namely `app.py` and `requirements.txt`. Let’s create those next.

The app itself

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

- Create two more files, `requirements.txt` and `app.py`, and put them in the same folder with the `Dockerfile`
- This completes our app, which as you can see is quite simple
- When the above `Dockerfile` is built into an image, `app.py` and `requirements.txt` will be present because of that `Dockerfile's ADD` command, and the output from `app.py` will be accessible over HTTP thanks to the `EXPOSE` command.



The App itself

app.py

Requirements.txt

```
Flask
Redis
```

That's it! You don't need Python or anything in `requirements.txt` on your system, nor will building or running this image install them on your system. It doesn't seem like you've really set up an environment with Python and Flask, but you have.

```python
from flask import Flask
from redis import Redis, RedisError
import os
import socket

# Connect to Redis
redis = Redis(host="redis", db=0, socket_connect_timeout=2, socket_timeout=2)

app = Flask(__name__)

@app.route("/")
def hello():
    try:
        visits = redis.incr("counter")
    except RedisError:
        visits = "<i>cannot connect to Redis, counter disabled</i>"

    html = "<h3>Hello {name}!</h3>" \
           "<b>Hostname:</b> {hostname}<br/>" \
           "<b>Visits:</b> {visits}"
    return html.format(name=os.getenv("NAME", "world"), hostname=socket.gethostname(), visits=visits)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

Building the app

- We are ready to build the app. Make sure you are still at the top level of your new directory. Here's what ls should show

```
PS C:\Users\gaiyer\dockerdemo> ls


    Directory: C:\Users\gaiyer\dockerdemo


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        14-Sep-17   7:11 PM            687 app.py
-a----        15-Sep-17   7:19 AM            499 Dockerfile
-a----        14-Sep-17   7:10 PM             14 requirements.txt
```

- Now run the `build` command. This creates a Docker image, which we're going to tag using `-t` so it has a friendly name.

Building the app


```
PS C:\Users\gaiyer\dockerdemo> docker build -t friendlyhello .
Sending build context to Docker daemon  4.608kB
Step 1/7 : FROM python:2.7-slim
 ---> 44cd565d93d8
Step 2/7 : WORKDIR /app
 ---> Using cache
 ---> ef4143793bcd
Step 3/7 : ADD . /app
 ---> Using cache
 ---> 97db317aaa01
Step 4/7 : RUN pip install -r requirements.txt
 ---> Using cache
 ---> 67bede3446e8
Step 5/7 : EXPOSE 80
 ---> Using cache
 ---> 4afa3b847151
Step 6/7 : ENV NAME World
 ---> Using cache
 ---> c8efa59ca2c5
Step 7/7 : CMD python app.py
 ---> Using cache
 ---> 05bd89474d38
Successfully built 05bd89474d38
Successfully tagged friendlyhello:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories a
ded to build context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitiv
 files and directories.
PS C:\Users\gaiyer\dockerdemo>
```

Where is your built images?

- `docker images`

```
PS C:\Users\gaiyer\dockerdemo> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
friendlyhello       latest              05bd89474d38        12 hours ago        194MB
hello-world         latest              05a3bd381fc2        2 days ago          1.84kB
python              2.7-slim            44cd565d93d8        6 days ago          182MB
PS C:\Users\gaiyer\dockerdemo>
```

Run the app

- Run the app, mapping your machine's port 4000 to the container's published port 80 using –p
- `docker run -p 4000:80 friendlyhello`

```
PS C:\Users\gaiyer\dockerdemo> docker run -p 4000:80 friendlyhello
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
```

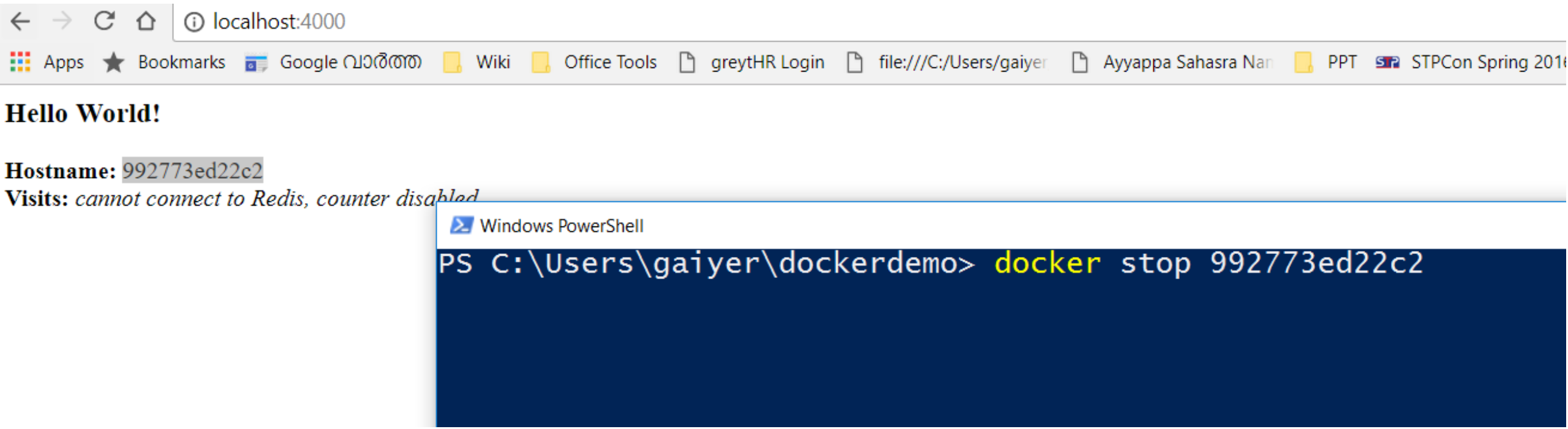
- You should see a notice that Python is serving your app at `http://0.0.0.0:80`. But that message is coming from inside the container, which doesn't know you mapped port 80 of that container to 4000, making the correct URL `http://localhost:4000`
- Go to that URL in a web browser to see the display content served up on a web page, including "Hello World" text, the container ID, and the Redis error message

Hello World!

Hostname: 992773ed22c2
Visits: *cannot connect to Redis, counter disabled*

End the process

- Hit CTRL+C in your terminal to quit
- Now use docker stop to end the process, using the CONTAINER ID, like so



- Now let's run the app in the background, in detached mode:
- `docker run -d -p 4000:80 friendlyhello`

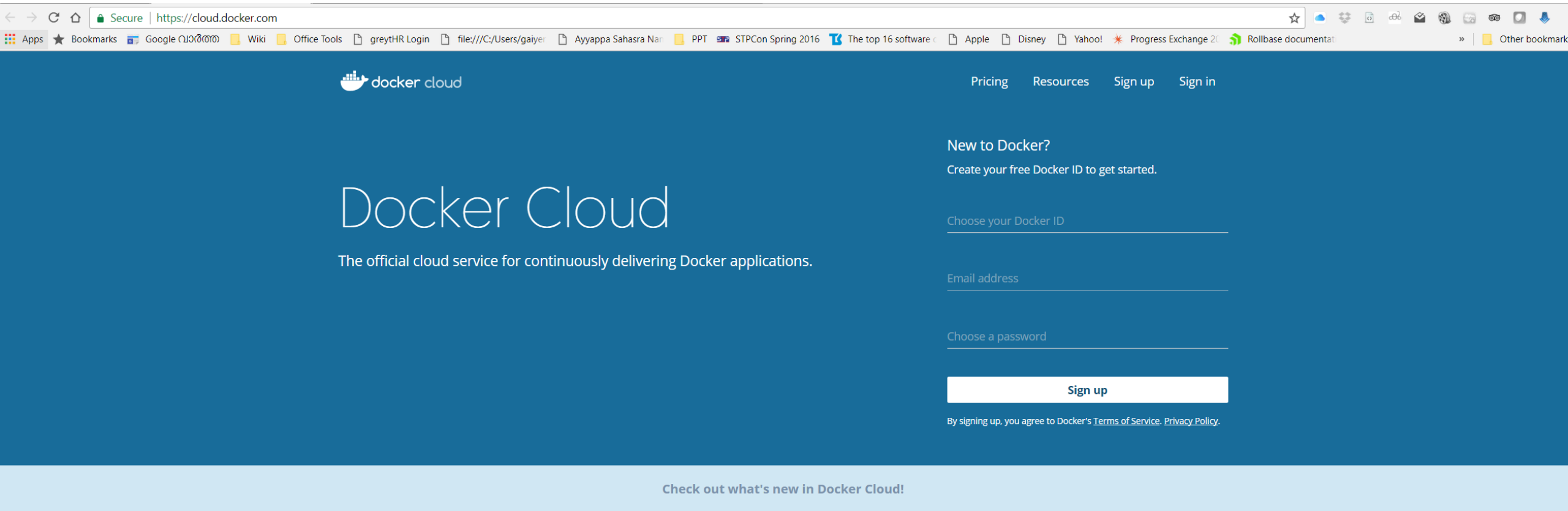
```
PS C:\Users\gaiyer\dockerdemo> docker run -d -p 4000:80 friendlyhello
271c306d3dbb351c9e46b4ad93a82d7a2c41439c89b92c23c96d7b63e5c0e141
PS C:\Users\gaiyer\dockerdemo>
```

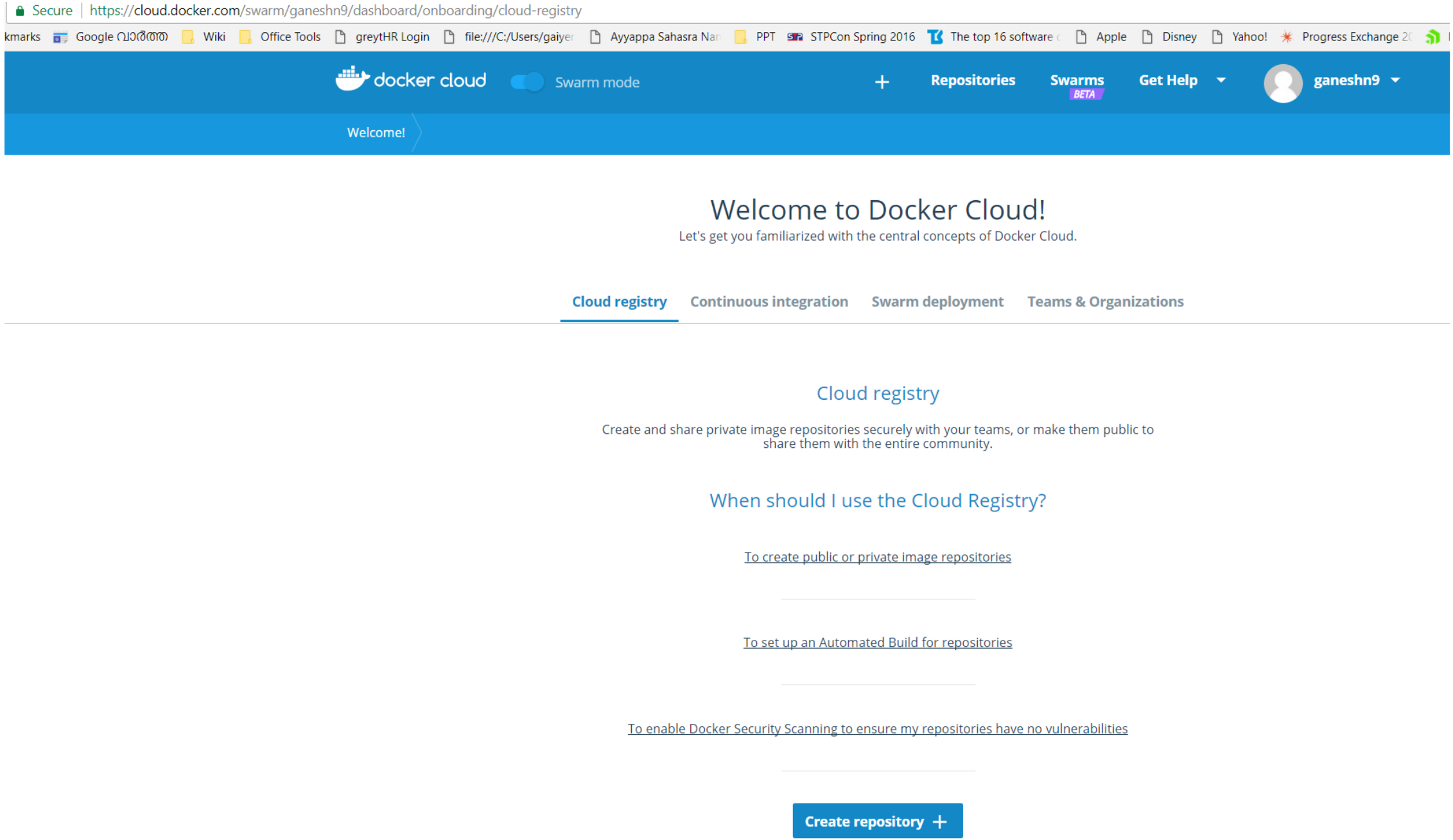
- You get the long container ID for your app and then are kicked back to your terminal. Your container is running in the background. You can also see the abbreviated container ID with docker container ls (and both work interchangeably when running commands):
- `docker container ls`

```
PS C:\Users\gaiyer\dockerdemo> docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                   NAMES
271c306d3dbb        friendlyhello       "python app.py"     About a minute ago  Up About a minute   0.0.0.0:4000->80/tcp    elegant_boyd
PS C:\Users\gaiyer\dockerdemo>
```

Share image

- To demonstrate the portability of what we just created, let's upload our built image and run it somewhere else
- After all, you'll need to learn how to push to registries when you want to deploy containers to production
- A registry is a collection of repositories, and a repository is a collection of images—sort of like a GitHub repository, except the code is already built. An account on a registry can create many repositories. The docker CLI uses Docker's public registry by default
- If you don't have a Docker account, sign up for one at cloud.docker.com. Make note of your username.


Secure | https://cloud.docker.com
Apps Bookmarks Google Wiki Office Tools greytHR Login Ayyappa Sahasra PPT STPCon Spring 2016 The top 16 software Apple Disney Yahoo! Progress Exchange Rollbase documentat Other bookmark
docker cloud
Pricing Resources Sign up Sign in
Docker Cloud
The official cloud service for continuously delivering Docker applications.
New to Docker?
Create your free Docker ID to get started.
Choose your Docker ID
Email address
Choose a password
Sign up
By signing up, you agree to Docker's Terms of Service. Privacy Policy.
Check out what's new in Docker Cloud!


Secure | https://cloud.docker.com/swarm/ganeshn9/dashboard/onboarding/cloud-registry
kmarks Google വാർത്ത Wiki Office Tools greytHR Login file:///C:/Users/gaiyer Ayyappa Sahasra Nar PPT STPCon Spring 2016 The top 16 software Apple Disney Yahoo! Progress Exchange
docker cloud
Swarm mode
Repositories
Swarms
BETA
Get Help
ganeshn9
Welcome!
Welcome to Docker Cloud!
Let's get you familiarized with the central concepts of Docker Cloud.
Cloud registry
Continuous integration
Swarm deployment
Teams & Organizations
Cloud registry
Create and share private image repositories securely with your teams, or make them public to share them with the entire community.
When should I use the Cloud Registry?
To create public or private image repositories
To set up an Automated Build for repositories
To enable Docker Security Scanning to ensure my repositories have no vulnerabilities
Create repository +

Login with your docker id

- Log in to the Docker public registry on your local machine.
- `docker login`

```
PS C:\Users\gaiyer\dockerdemo> docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create
one.
Username: ganeshn9
Password:
Login Succeeded
PS C:\Users\gaiyer\dockerdemo>
```

Tag the image

- The notation for associating a local image with a repository on a registry is `username/repository:tag`. The tag is optional, but recommended, since it is the mechanism that registries use to give Docker images a version. Give the repository and tag meaningful names for the context, such as `get-started:part1`. This will put the image in the `get-started` repository and tag it as `part1`.

- Now, put it all together to tag the image. Run `docker tag image` with your username, repository, and tag names so that the image will upload to your desired destination. The syntax of the command is:

Tag the image

```
PS C:\Users\gaiyer\dockerdemo> docker tag friendlyhello ganeshn9/get-started:part1
PS C:\Users\gaiyer\dockerdemo> docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
friendlyhello               latest              05bd89474d38        13 hours ago        194MB
ganeshn9/get-started        part1               05bd89474d38        13 hours ago        194MB
hello-world                 latest              05a3bd381fc2        2 days ago          1.84kB
python                      2.7-slim            44cd565d93d8        6 days ago          182MB
PS C:\Users\gaiyer\dockerdemo>
```

Publish the image

- Upload your tagged image to the repository
- `docker push username/repository:tag`

```
PS C:\Users\gaiyer\dockerdemo> docker push ganeshn9/get-started:part1
The push refers to a repository [docker.io/ganeshn9/get-started]
11c91e030649: Pushing [==========================>                        ]  6.281MB/11.88MB
b7700d91782f: Pushing [=================================================>]   5.12kB
d4b5e68d6f2a: Pushing  1.536kB
dda0f6767edb: Preparing
cff0b845ede9: Mounted from library/python
1c4694f66f95: Waiting
18f9b4e2e1bc: Waiting
```

- Once complete, the results of this upload are publicly available. If you log in to Docker Hub, you will see the new image there, with its `pull` command

Publish the image

- Upload your tagged image to the repository
- `docker push username/repository:tag`

```
PS C:\Users\gaiyer\dockerdemo> docker push ganeshn9/get-started:part1
The push refers to a repository [docker.io/ganeshn9/get-started]
11c91e030649: Pushed
b7700d91782f: Pushed
d4b5e68d6f2a: Pushed
dda0f6767edb: Mounted from library/python
cff0b845ede9: Mounted from library/python
1c4694f66f95: Mounted from library/python
18f9b4e2e1bc: Mounted from library/python
part1: digest: sha256:d2308105f5abc29b21009707654c7a7f6bc813ee3b9a05f04e8455205551ba49 size: 1787
PS C:\Users\gaiyer\dockerdemo>
```

- Once complete, the results of this upload are publicly available. If you log in to Docker Hub, you will see the new image there, with its `pull` command

Secure | https://hub.docker.com

okmarks Google ഡ്രൈവ് Wiki Office Tools greytHR Login file:///C:/Users/gaiye Ayyappa Sahasra Na PPT STPCon Spring 2016 The top 16 software Apple Disney Yahoo! Progress Exchange 2 Rollbase documentat

Search

Dashboard Explore Organizations Create ganeshn9

ganeshn9 Repositories Stars Contributed

Private Repositories: Using 0 of 1 Get more

Repositories

Create Repository +

Type to filter repositories by name

ganeshn9/get-started public	0 STARS	1 PULLS	DETAILS

Docker Security Scanning

Protect your repositories from vulnerabilities.
Try it free

Pull and run the image from the remote repository

- From now on, you can use docker run and run your app on any machine with this command:
- docker run -p 4000:80 username/repository:tag
- If the image isn't available locally on the machine, Docker will pull it from the repository.
- If you don't specify the :tag portion of these commands, the tag of :latest will be assumed, both when you build and when you run images. Docker will use the last version of the image that ran without a tag specified (not necessarily the most recent image).

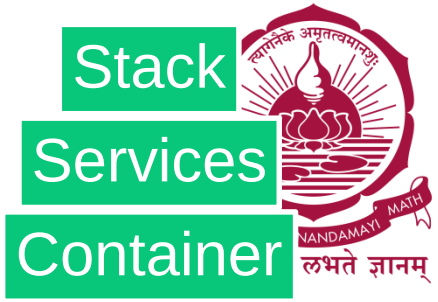
No matter where executes, it pulls your image, along with Python and all the dependencies from , and runs your code. It all travels together in a neat little package, and the host machine doesn't have to install anything but Docker to run it.

What have you seen so far?

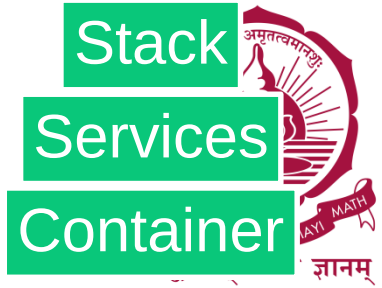
- Basics of Docker
- How to create your first app in the Docker way
- Building the app
- Run the app
- Sharing and Publishing images
- Pull and run images


What next???

Services



- We can scale our application and enable load-balancing
- To do this, we must go one level up in the hierarchy of a distributed application: the **service**.
- In a distributed application, different pieces of the app are called “services.”
- For example, if you imagine a video sharing site, it probably includes a service for storing application data in a database, a service for video transcoding in the background after a user uploads something, a service for the front-end, and so on



Services

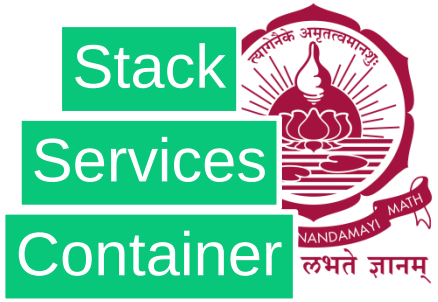
- It’s very easy to define, run, and scale services with the Docker platform
- just write a `docker-compose.yml` file
- This helps you define how your app should run in production by turning it into a service, scaling it up in the process
- You can deploy this application onto a cluster, running it on multiple machines
- Multi-container, multi-machine applications are made possible by joining multiple machines into a “Dockerized” cluster called a **swarm**.

docker-compose.yml

```yaml
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repository:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      - webnet
networks:
  webnet:
```

- Pull the image we uploaded before from the registry.
- Run 5 instances of that image as a service called web, limiting each one to use, at most, 10% of the CPU (across all cores), and 50MB of RAM.
- Immediately restart containers if one fails.
- Map port 80 on the host to web's port 80.
- Instruct web's containers to share port 80 via a load-balanced network called webnet. (Internally, the containers themselves will publish to web's port 80 at an ephemeral port.)
- Define the webnet network with the default settings (which is a load-balanced overlay network).

Stacks



- A stack is a group of interrelated services that share dependencies, and can be orchestrated and scaled together
- A single stack is capable of defining and coordinating the functionality of an entire application
- Though very complex applications may want to use multiple stacks

Summary

- We have seen very basic hands on experience of
  - How to build an app in the docker way
  - Push your app to a registry
  - Pull existing apps from registry
- Overview of scaling apps
- Useful reference: https://docs.docker.com/get-started/

Webapps with Docker

Run a static website in a container

- First, we'll use Docker to run a static website in a container
- The website is based on an existing image
- We'll pull a Docker image from Docker Store, run the container, and see how easy it is to set up a web server

```
docker run -d dockersamples/static-site
```

```
PS C:\Users\gaiyer> docker run -d dockersamples/static-site
Unable to find image 'dockersamples/static-site:latest' locally
latest: Pulling from dockersamples/static-site
fdd5d7827f33: Pull complete
a3ed95caeb02: Pull complete
716f7a5f3082: Pull complete
7b10f03a0309: Pull complete
aff3ab7e9c39: Pull complete
Digest: sha256:daa686c61d7d239b7977e72157997489db49f316b9b9af3909d9f10fd28b2dec
Status: Downloaded newer image for dockersamples/static-site:latest
e666eace16b1a98c75c0aef5882679f59fb13d63a4fa71d705d8b1d346756624
```

Run a static website in a container

- What happens when you run this command?
- Since the image doesn't exist on your Docker host, the Docker daemon first fetches it from the registry and then runs it as a container.
- Now that the server is running, do you see the website? What port is it running on? And more importantly, how do you access the container directly from our host machine?
- Actually, you probably won't be able to answer any of these questions yet! ☺ In this case, the client didn't tell the Docker Engine to publish any of the ports, so you need to re-run the `docker run` command to add this instruction.

- Let's re-run the command with some new flags to publish ports and pass your name to the container to customize the message displayed.
- First, stop the container that you have just launched. In order to do this, we need the container ID
- Run `docker ps` to view the running containers

```
PS C:\Users\gaiyer> docker ps
CONTAINER ID        IMAGE                      COMMAND                  CREATED             STATUS              PORTS               NAMES
e666eace16b1        dockersamples/static-site  "/bin/sh -c 'cd /usr…"   5 minutes ago       Up 5 minutes        80/tcp, 443/tcp     musing_albattani
PS C:\Users\gaiyer>
```

- Check out the `CONTAINER ID` column. You will need to use this `CONTAINER ID` value, a long sequence of characters, to identify the container you want to stop, and then to remove it.

```
docker stop e666eace16b1
docker rm   e666eace16b1
```

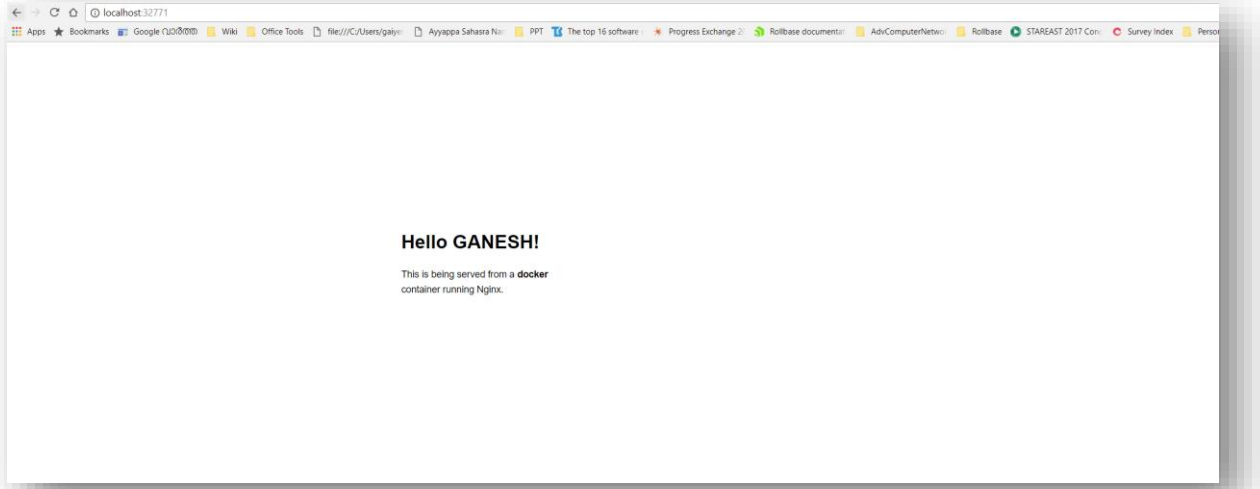
```
PS C:\Users\gaiyer> docker stop e666eace16b1
e666eace16b1
PS C:\Users\gaiyer> docker rm e666eace16b1
e666eace16b1
PS C:\Users\gaiyer>
```

- `docker run --name static-site -e AUTHOR="Your Name" -d -P dockersamples/static-site`

```
PS C:\Users\gaiyer> docker run --name static-site -e AUTHOR="GANESH" -d -P dockersamples/static-site
b338c6e787b1c02042233669fa335c109bbf247ac57c638bd2491e588017e480
PS C:\Users\gaiyer>
```

- `-d` will create a container with the process detached from our terminal
- `-P` will publish all the exposed container ports to random ports on the Docker host
- `-e` is how you pass environment variables to the container
- `--name` allows you to specify a container name
- `AUTHOR` is the environment variable name and Your Name is the value that you can pass

- Now you can see the ports
```
PS C:\Users\gaiyer> docker port static-site
443/tcp -> 0.0.0.0:32770
80/tcp -> 0.0.0.0:32771
PS C:\Users\gaiyer>
```
and
  - `docker port static-site`
- You can now open `http://localhost:[YOUR_PORT_FOR 80/tcp]` to see your site live!



- Let's stop and remove the containers since you won't be

```
PS C:\Users\gaiyer> docker stop static-site
static-site
PS C:\Users\gaiyer> docker rm static-site
static-site
PS C:\Users\gaiyer> docker run --name static-site -e AUTHOR="GANESH" -d -P dockersamples/static-site
42a58a4ef16ba5120884880d80e849fca5599d531df99f21df97a50adb94c307
PS C:\Users\gaiyer> docker rm -f static-site
static-site
PS C:\Users\gaiyer>
```

- `rm -f` is a shortcut to remove the site

Thank you!

Dr Ganesh Neelakanta Iyer

ni_amrita@cb.amrita.edu
ganesh.vigneswara@gmail.com

Office Hours

- Tuesday 4-445 PM @ My office