# Deploying Microservices

Memi Lavi
www.memilavi.com

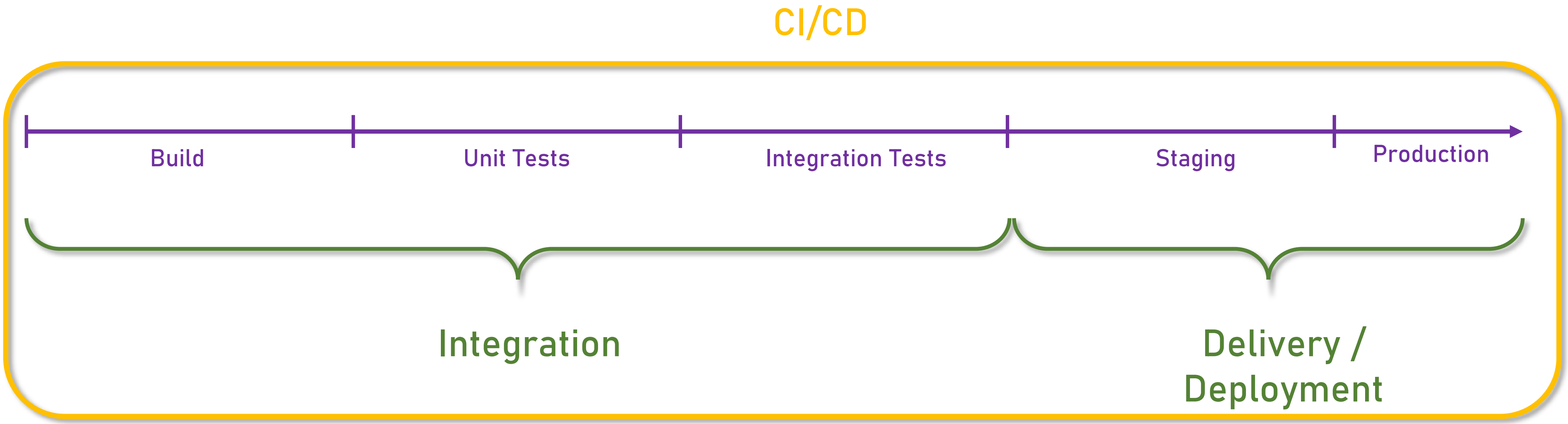
# Deploying Microservices

- Deployment of microservices is extremely important
- Remember: “Infrastructure Automation”
- Slow and complicated deployment will render the whole system ineffective and useless
- Architect should be aware of deployment, not responsible
- We’ll discuss high-level concepts

# CI/CD

- Stands for:
  - Continuous Integration / Continuous Delivery
  - The full automation of the integration and delivery stages

# Integration & Delivery



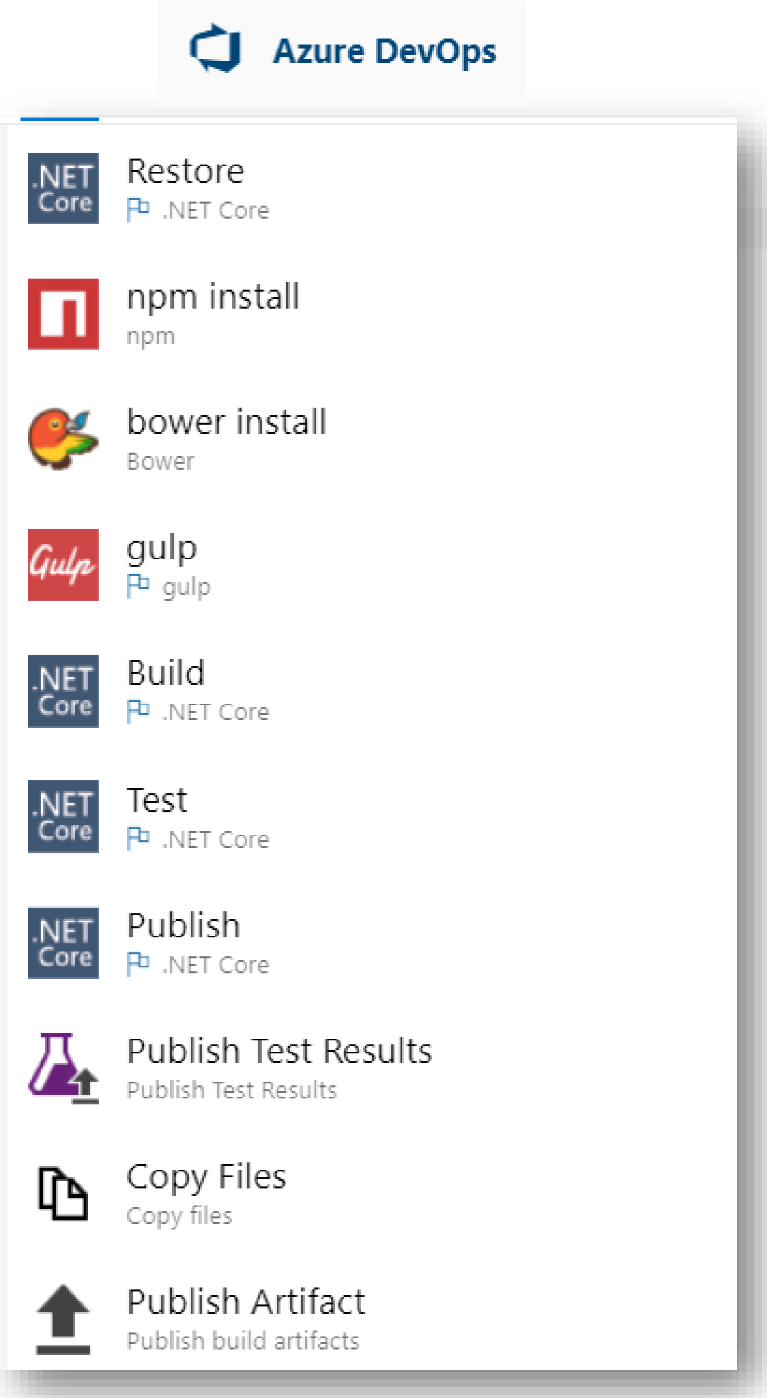
# Why Use CI/CD?

- Faster release cycle
- Reliability
- Reporting

# CI/CD Pipelines

- The heart of the CI/CD process
- Define the set of actions to perform as part of the CI/CD
- Usually defined using YAML, with UI representation

# CI/CD Pipelines



Azure DevOps

- Restore
  .NET Core
- npm install
  npm
- bower install
  Bower
- gulp
  gulp
- Build
  .NET Core
- Test
  .NET Core
- Publish
  .NET Core
- Publish Test Results
  Publish Test Results
- Copy Files
  Copy files
- Publish Artifact
  Publish build artifacts

# CI/CD

- As the Architect:
  - Make sure there is a CI/CD engine in place
  - Help shape the steps in the pipeline

# Containers

- Traditional deployment:
    - Code was copied and built on the production server
    - Problems were found on the servers that weren’t found in the dev machines

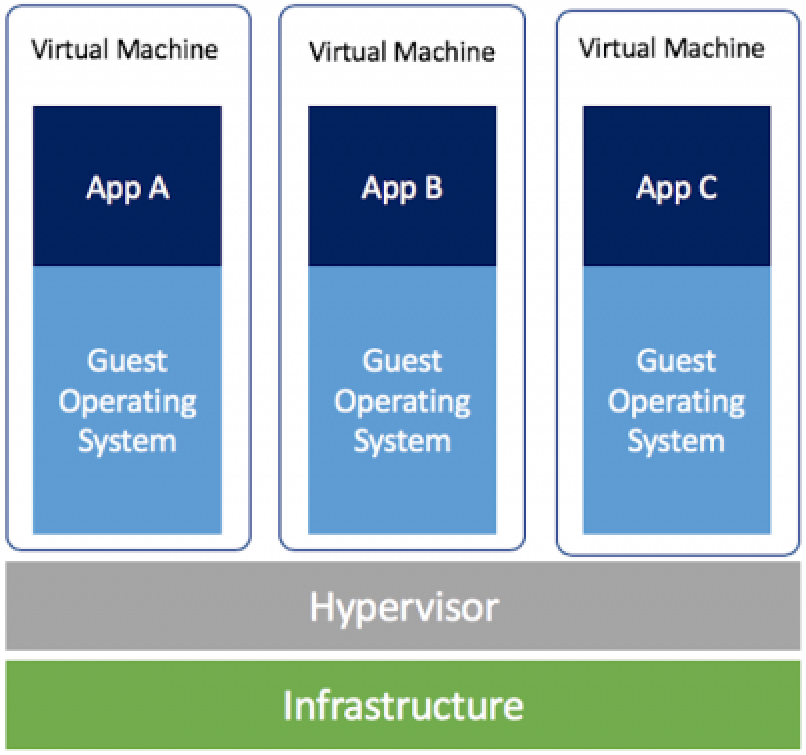
# Containers to the Rescue!

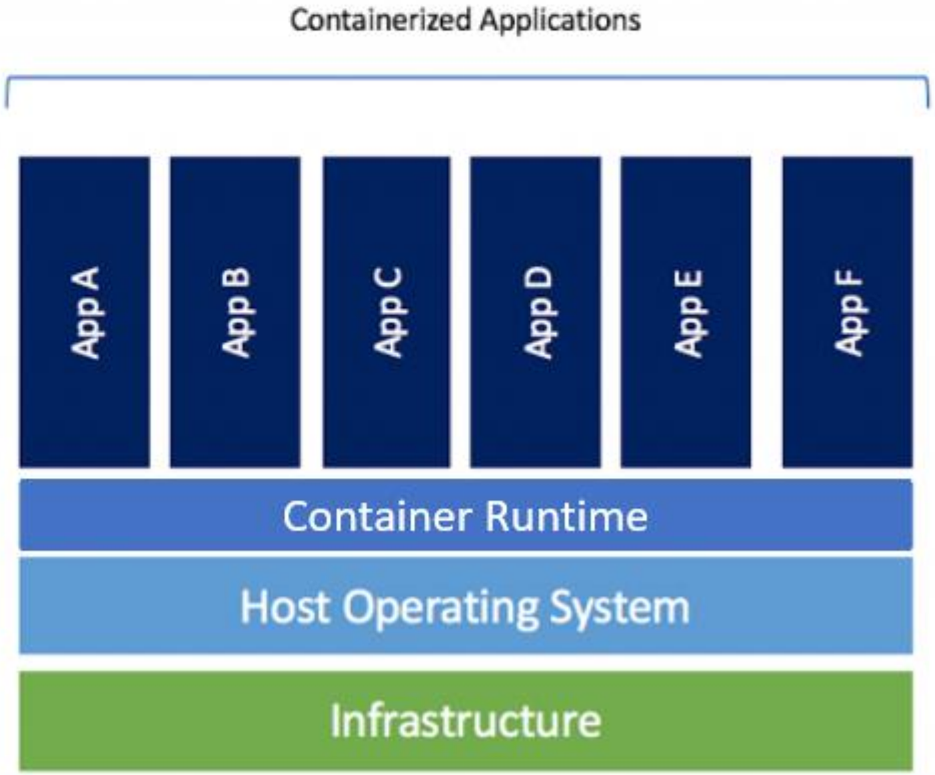
http://www.developermemes.com/2013/12/23/defect-production-works-machine/

# Containers

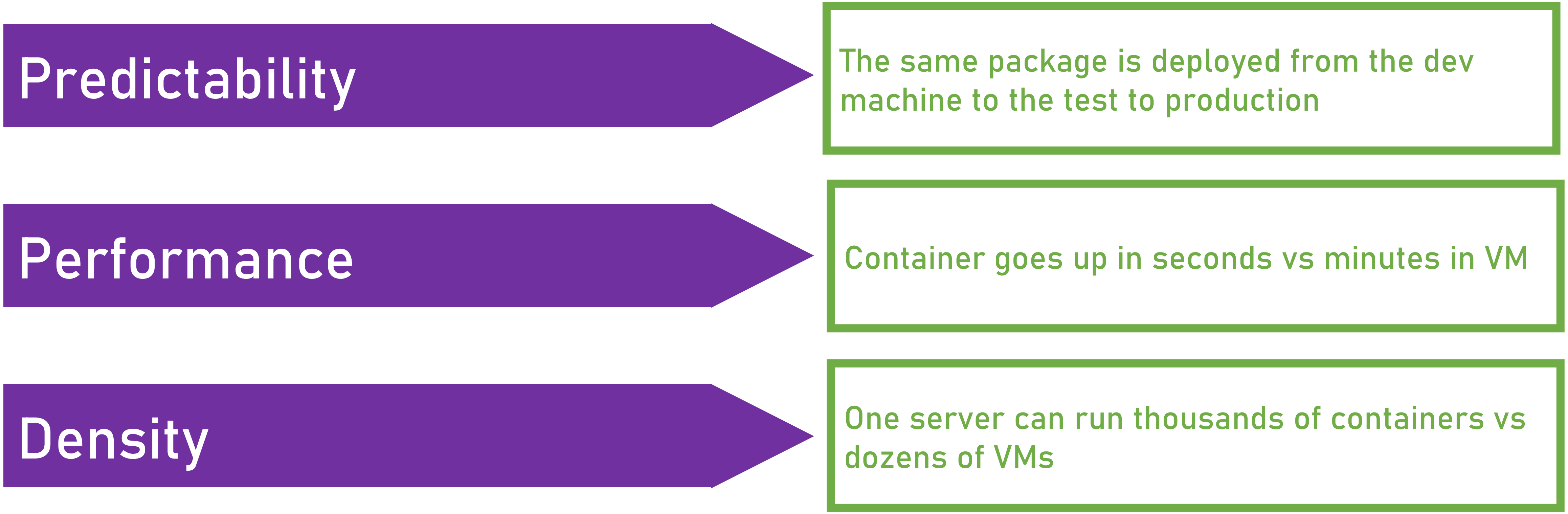
- Thin packaging model
- Packages software, its dependencies, and configuration files
- Can be copied between machines
- Uses the underlying operating system

# Container vs VM





# Why Containers?



Predictability

The same package is deployed from the dev machine to the test to production

Performance

Container goes up in seconds vs minutes in VM

Density

One server can run thousands of containers vs dozens of VMs

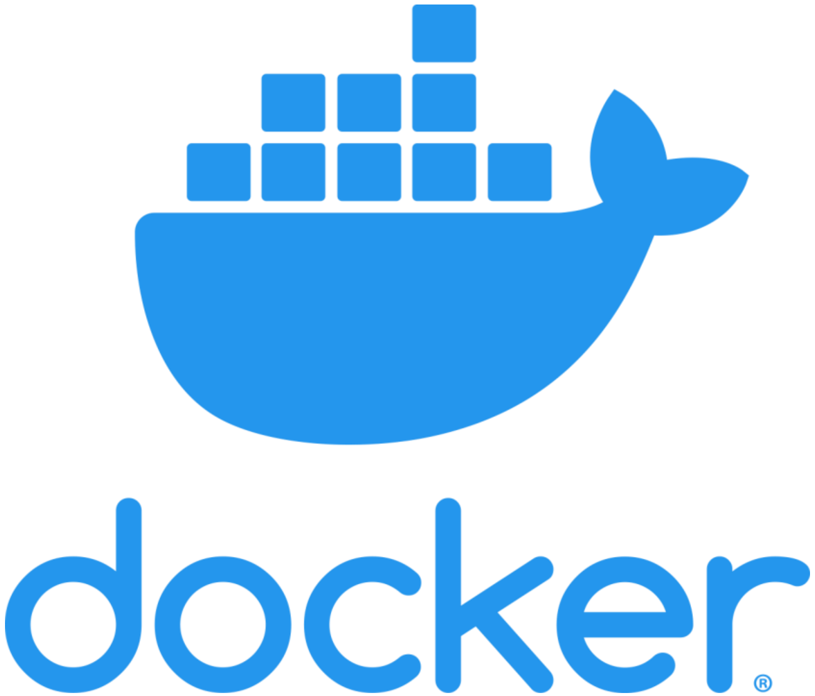
# Why Not Containers?

Isolation

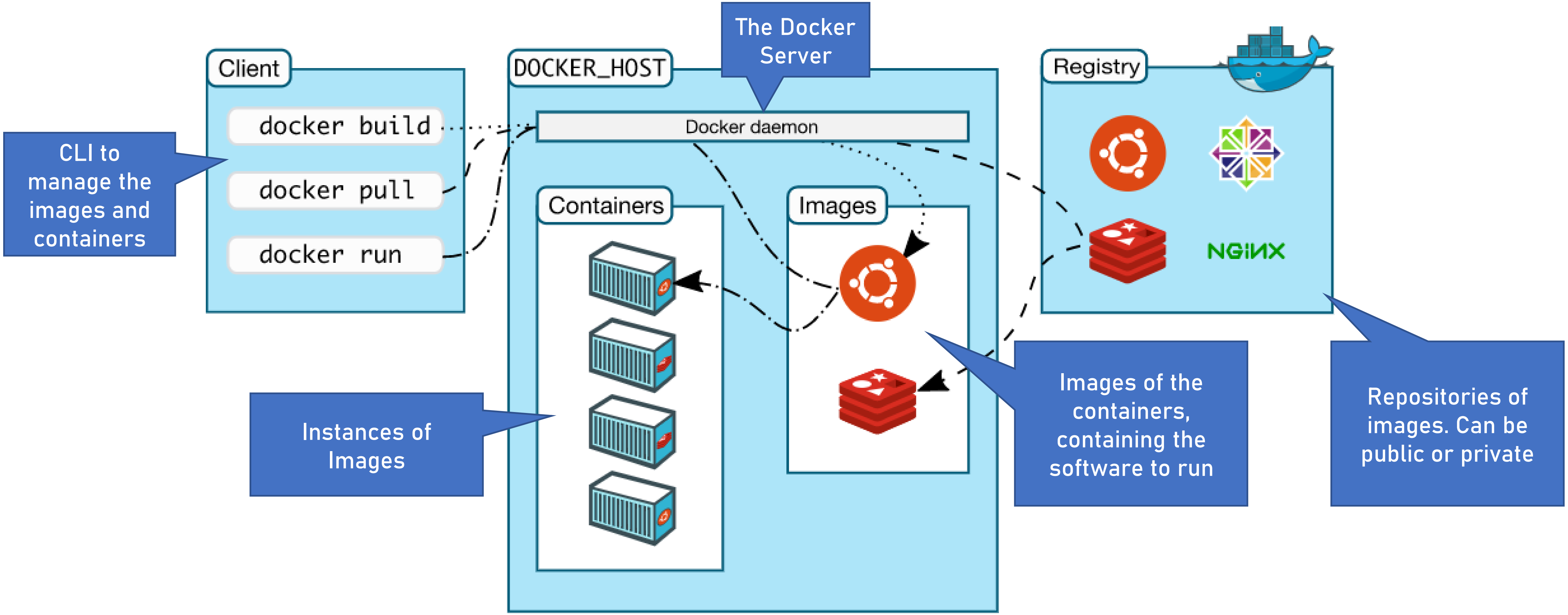
Containers share the same OS, so isolation is lighter than VM

# Docker

- The most popular container environment
- De-facto standard for containers
- Released in 2013



# Docker Architecture



CLI to manage the images and containers

The Docker Server

Instances of Images

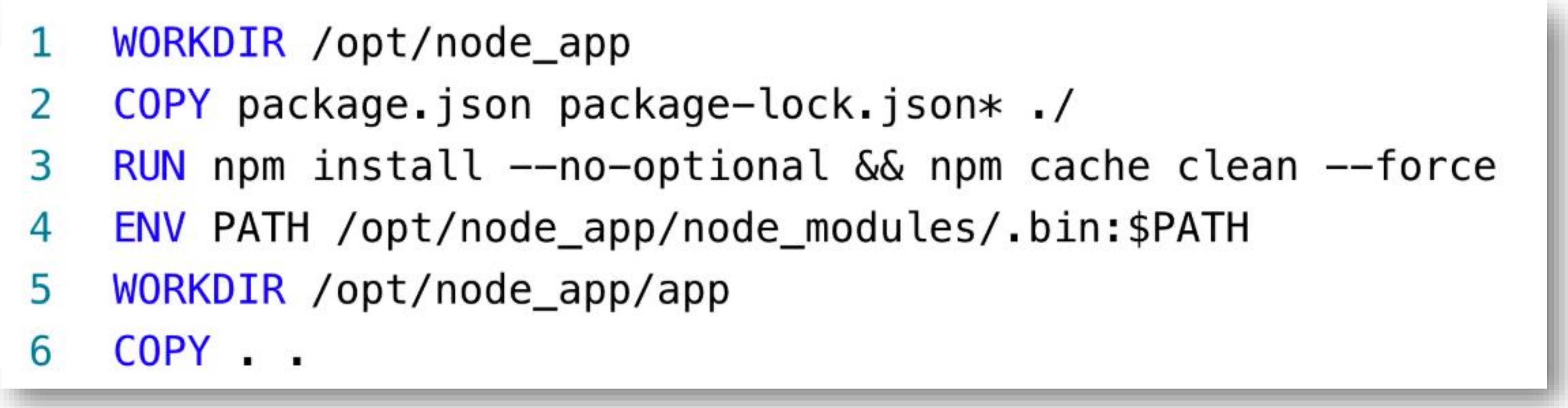
Images of the containers, containing the software to run

Repositories of images. Can be public or private

https://docs.docker.com/get-started/overview/

# dockerfile

- Contains instructions for building custom images



```
WORKDIR /opt/node_app
COPY package.json package-lock.json* ./
RUN npm install --no-optional && npm cache clean --force
ENV PATH /opt/node_app/node_modules/.bin:$PATH
WORKDIR /opt/node_app/app
COPY . .
```

https://www.docker.com/blog/keep-nodejs-rockin-in-docker/

# Support for Docker

- Supported by all major operating systems (Windows, Linux, OSX)
- Supported by major cloud providers

amazon
ECR

Azure
ACR

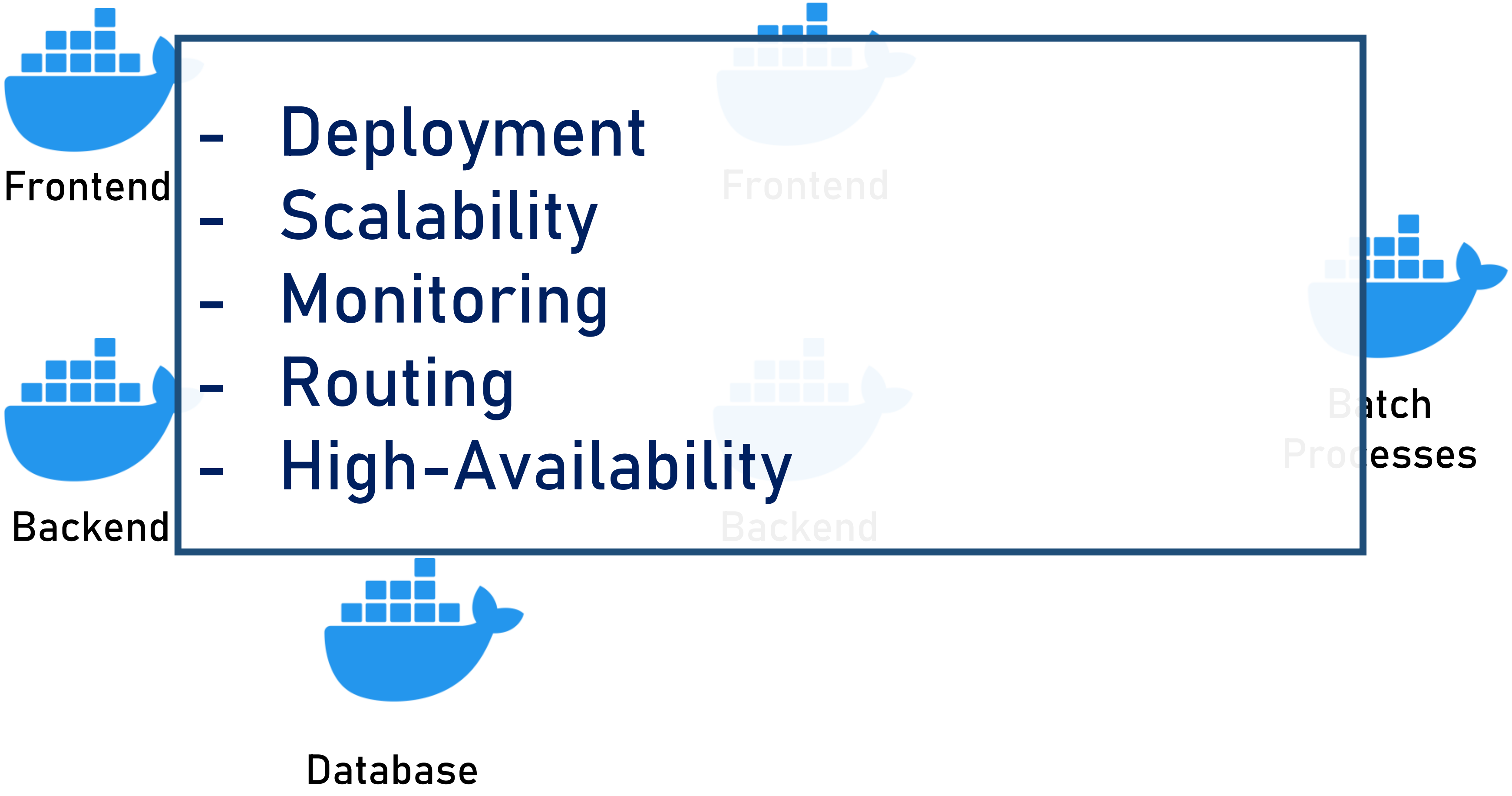
# Containers Management

- Containers are a great deployment mechanism

- Gain popularity

- What happens when there are too many of them?

# Containers Management



- Deployment
- Scalability
- Monitoring
- Routing
- High-Availability

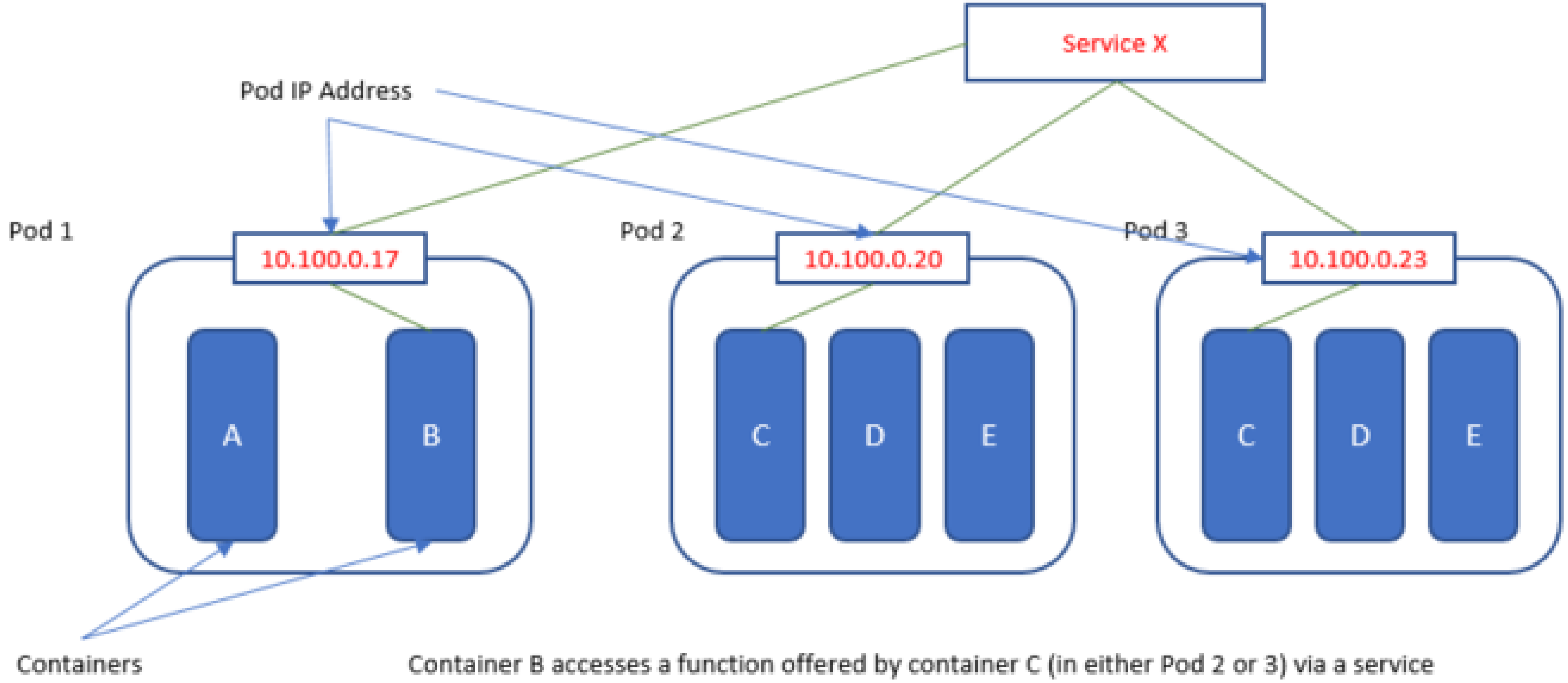
# Kubernetes

- The most popular container management platform
- De-facto standard for container management
- Released by Google in 2014

# Kubernetes

- Provides all aspects of management:
  - Routing
  - Scaling
  - High-Availability
  - Automated Deployment
  - Configuration Management
  - And more…

# Kubernetes Architecture



Service X
Pod IP Address
Pod 1
Pod 2
Pod 3
10.100.0.17
10.100.0.20
10.100.0.23
A
B
C
D
E
C
D
E
Containers
Container B accesses a function offered by container C (in either Pod 2 or 3) via a service

https://en.wikipedia.org/wiki/Kubernetes

# Deployment - Summary

- Automated deployment is a must for effective microservices architecture
- Docker and Kubernetes are the de-facto industry standard
- As an Architect, you’re not responsible for the deployment but should be aware