Microservices, Containers, and Kubernetes

Bob Cotton
Splunk
rcotton@splunk.com
February 24, 2021

About Me

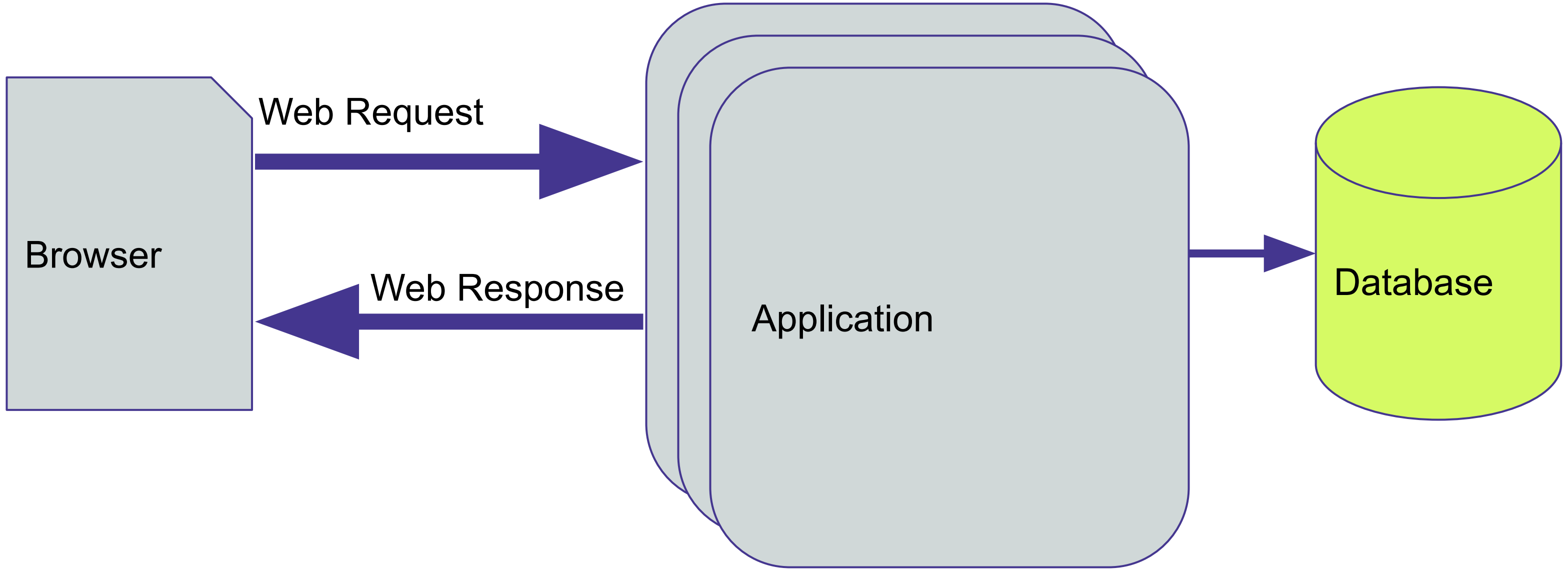
- Senior Principal Engineer at Splunk
  - Cloud Infrastructure and Operations
    - Observability and SRE
- Working on Observability Systems for the last 15 years

Agenda

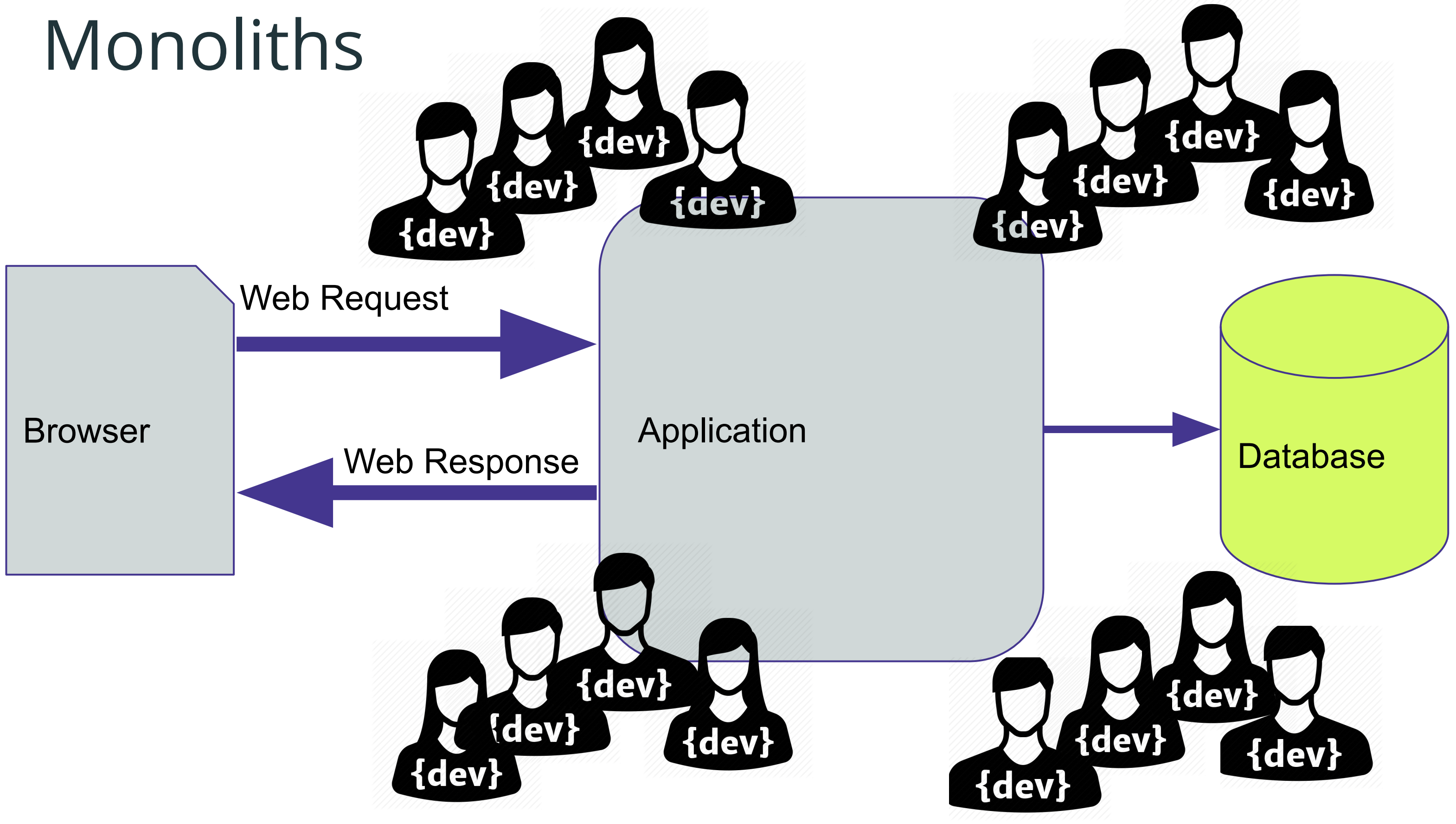
Why Microservices?

Monoliths vs. Microservices

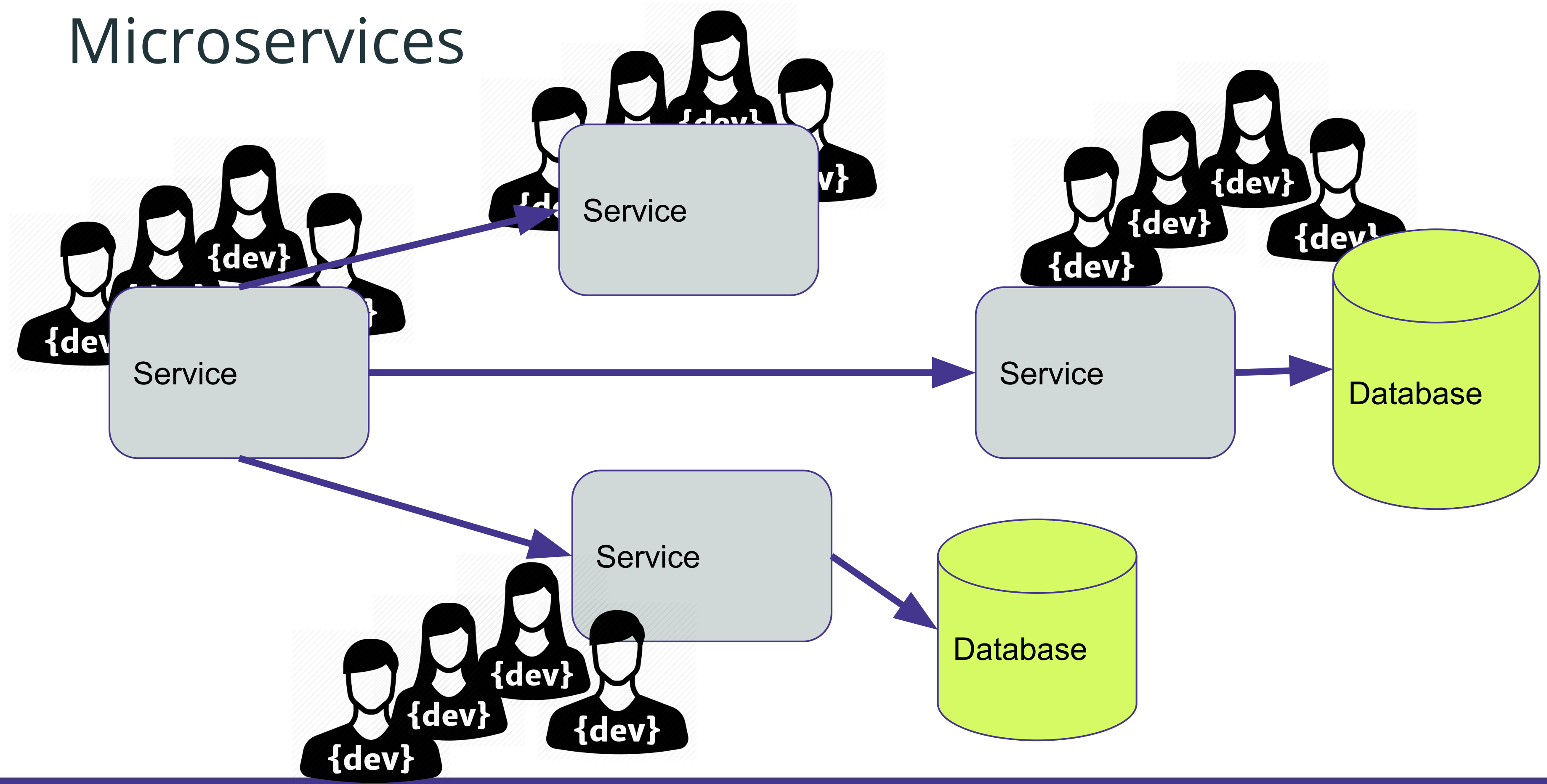
Monoliths



Monoliths


{dev}
{dev}
{dev}
{dev}
{dev}
{dev}
{dev}
{dev}
{dev}
{dev}
Browser
Web Request
Web Response
Application
Database
{dev}
{dev}
{dev}
{dev}
{dev}
{dev}
{dev}
{dev}
{dev}
{dev}

Microservices

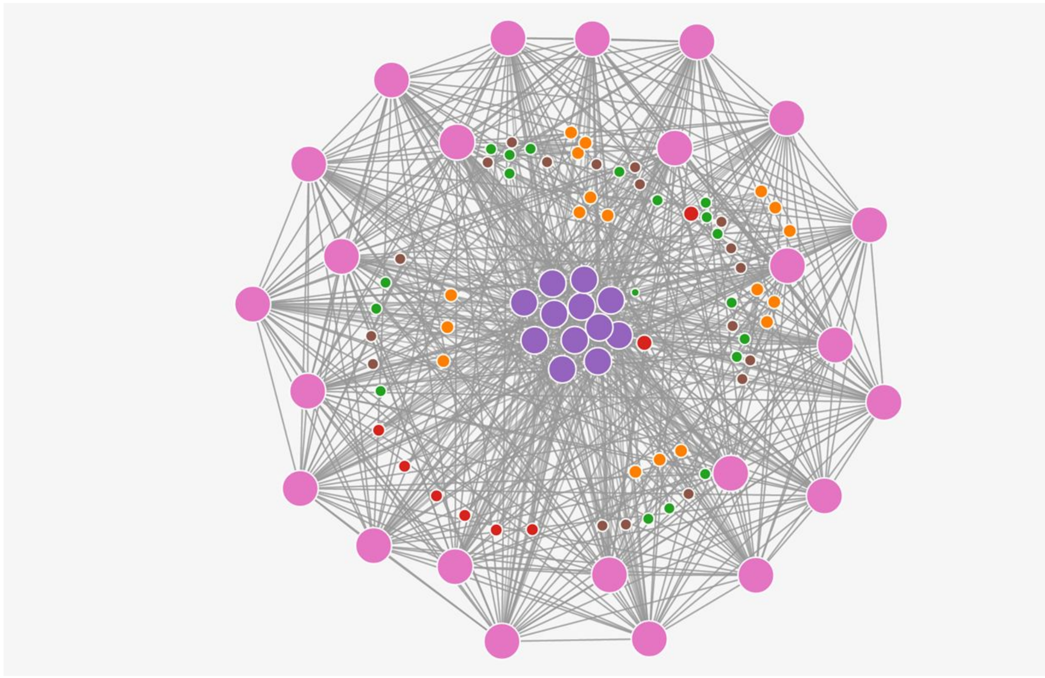


Microservices Benefits

- Developer Productivity
- Separation of Concerns
- Independently Deployable
- Independently Scalable

Microservice Complexity

- More Moving Pieces
- Logging
- Debugging
- Balancing Load
- Multiple/Rolling Deployment
- Monitoring



Why Containers?

Wait, What’s a container?

A Container == Linux Process

- Conceptually a “lightweight” VM
  - Own process space
  - Own network interface
  - Can run stuff as root
  - Can install packages
  - Can run services
- Containers start in ms

- Not Quite a VM
  - Uses the host kernel
  - Can’t boot another OS
  - Can have it’s own kernel modules

A collection of Linux Kernel Primitives

- Set of Control Groups (cgroups) that **meter and limit**
  - CPU
  - Memory
  - Block I/O
  - Network
- Namespaces
  - Process Isolation
- Special File Systems
  - Copy-on-write
  - Layered

Many Different Implementations

- LXC
- systemd-nspawn
- runC
- rkt
- Docker

They ALL use the same primitives!

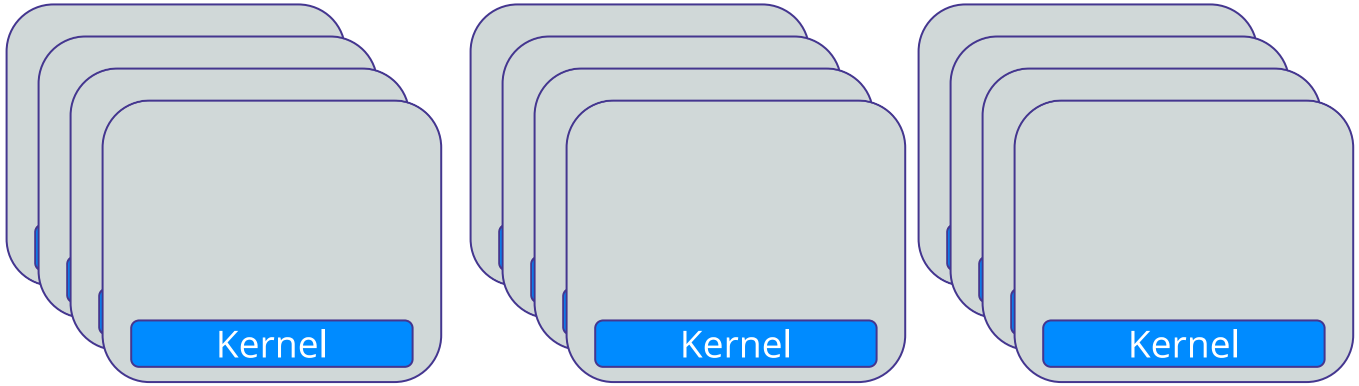
Docker

- Text description of the container -- Dockerfile
- Packaging into an image
  - All filesystem layers and executables
- Registry to share images

Why Containers?

Evolution of Cloud Computing

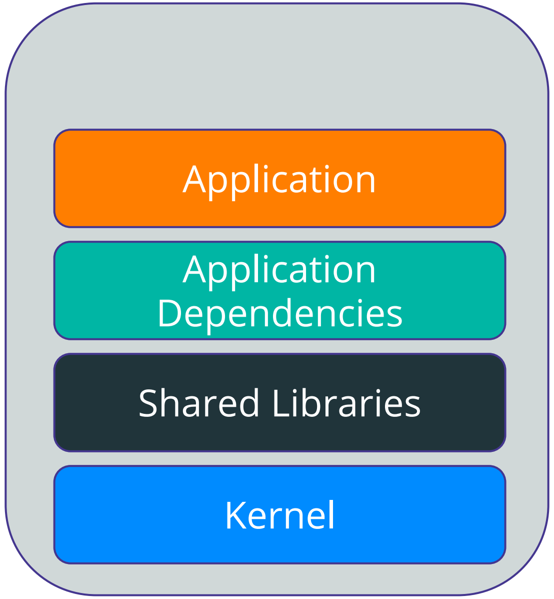
Virtual Machines and Cloud Providers



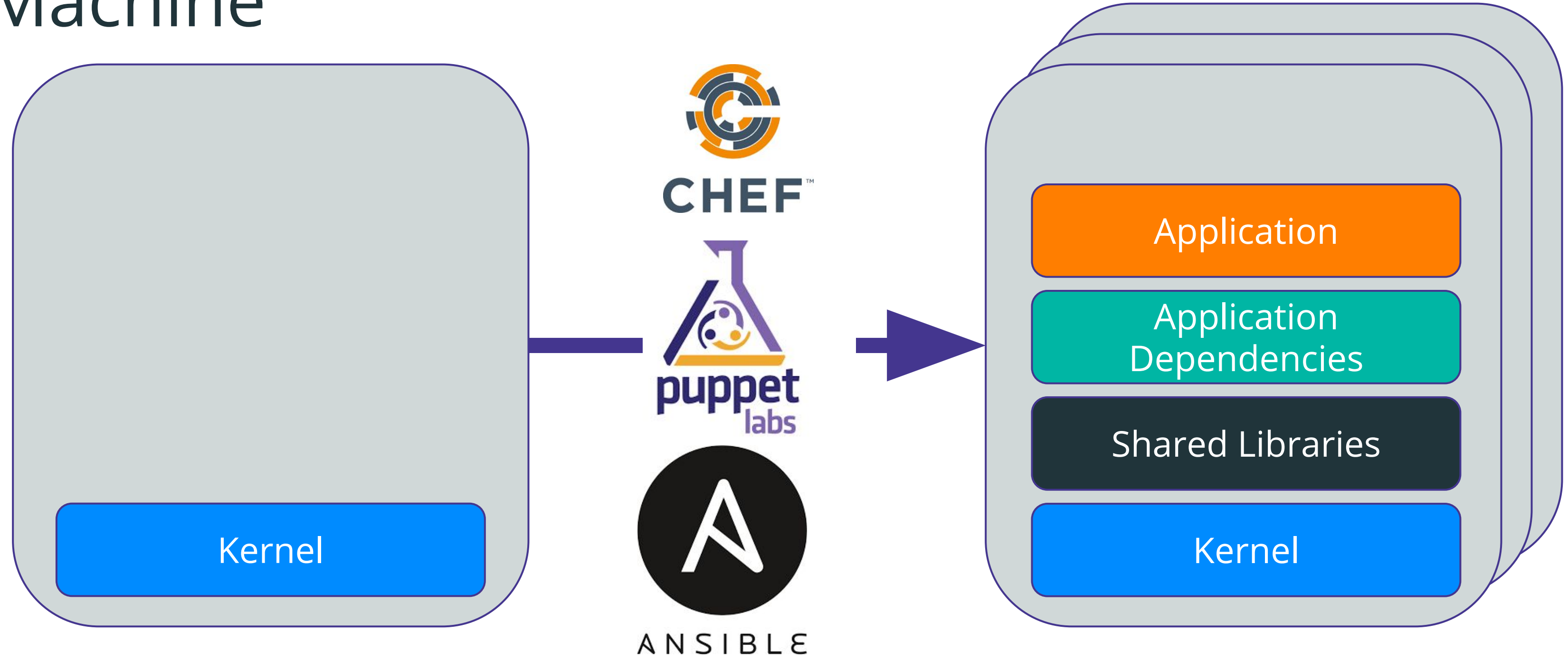
amazon web services™ EC2

Google Compute Engine

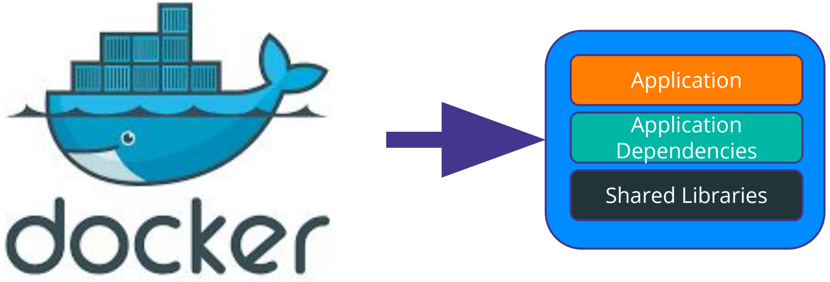
Application Dependencies

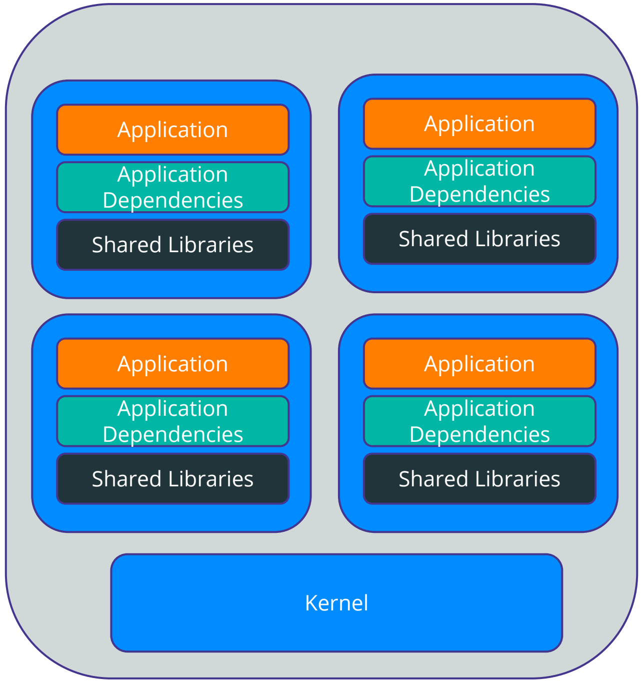


Automate Application Dependencies on the Machine

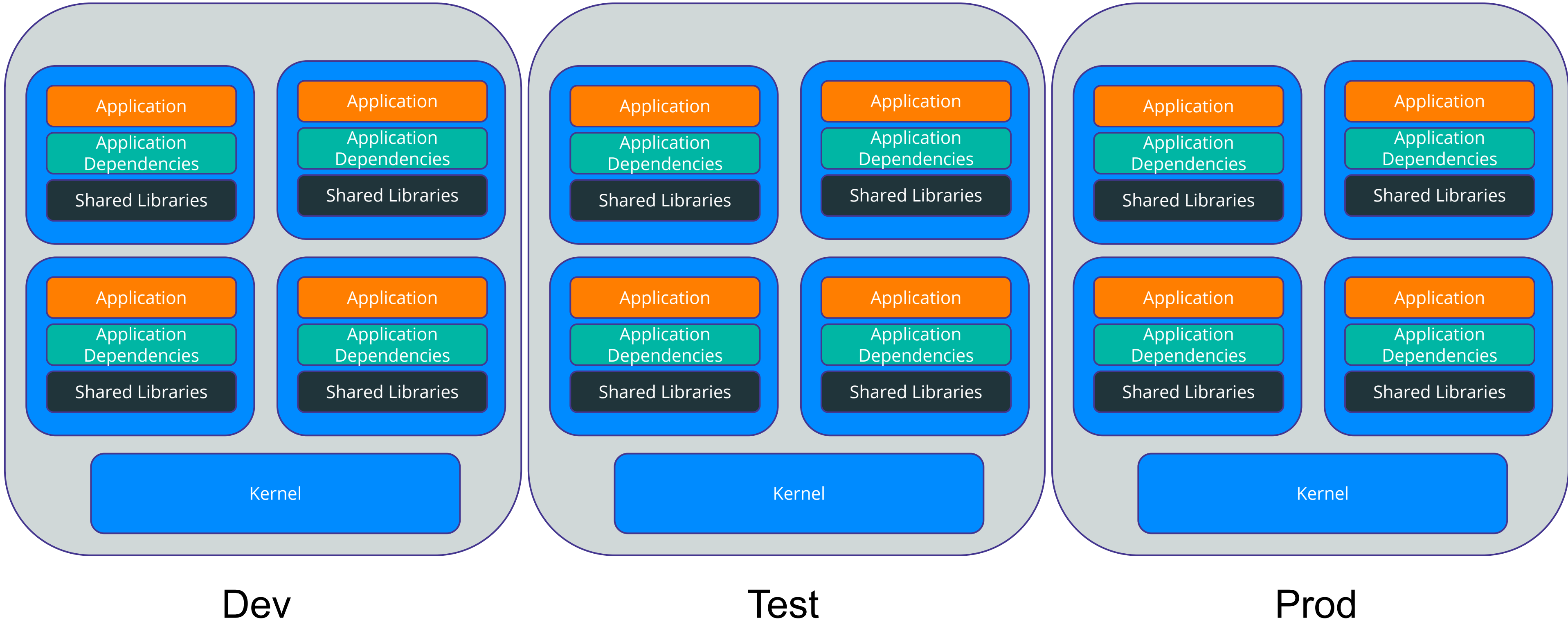


Application Dependencies in the Container





Same Container Runs Everywhere



Container Benefits

- Dependencies managed at development time
- Build once, run everywhere
- Lighter than VMs
  - Startup in 10ths of seconds

In the end Deploy and Test Faster

Why Orchestration?

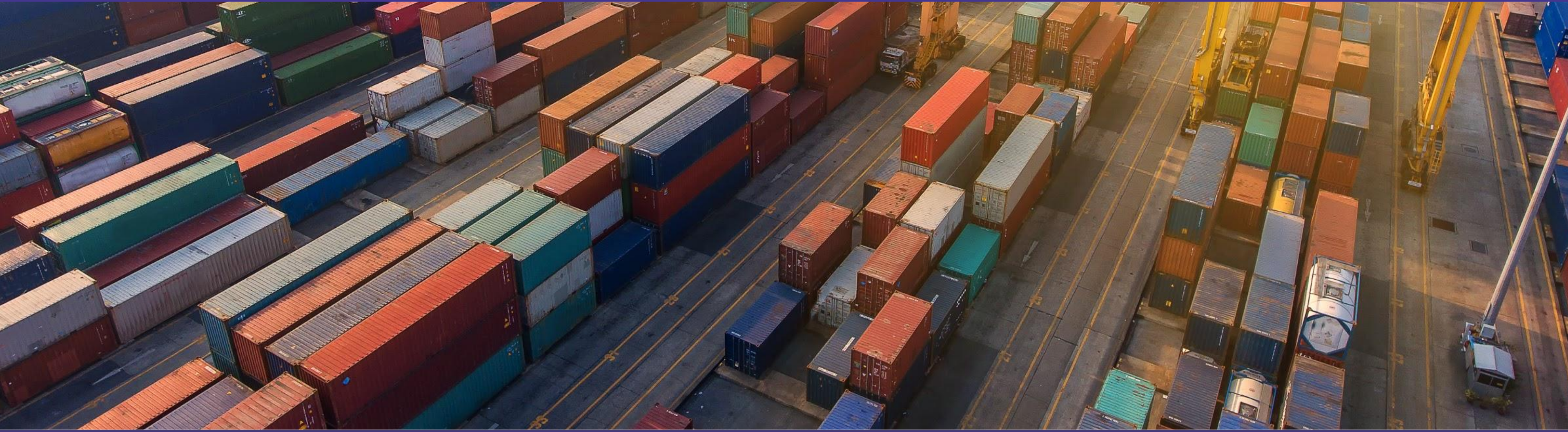
Automated Deployment is **Custom Code**

- DevOps says to “automate all the things”
- However, we usually re-create all things:
  - Asset Distribution
  - Process Supervision
  - Resource Utilization
  - Load-balancing
  - Developer self-service
- All varies by technology (Java, Ruby, Python etc)
  - Polyglot shop? Good luck!

Lifecycle Management is **Custom Code**

- Manage the Number of Instances
- Rolling Deployment
- Graceful Roll-backs
- Auto scaling
- Hybrid Cloud Abstractions

Orchestration is a Solution



Container Orchestration is a Solution

- Standardized solution for:
  - Packaging
    - Containers
  - Deployments
    - Declarative manifests
  - Lifecycle Management
    - Declarative manifests
    - Rolling Deployments
  - Infrastructure Interfacing
    - Cloud Storage
    - GPUs

Questions?