MicroServices And Docker

Steve Franson
Java Practice Manager/Consultant
STG Consulting
steve.franson@stgconsulting.com
801.595.1000
https://www.linkedin.com/in/sfranson/

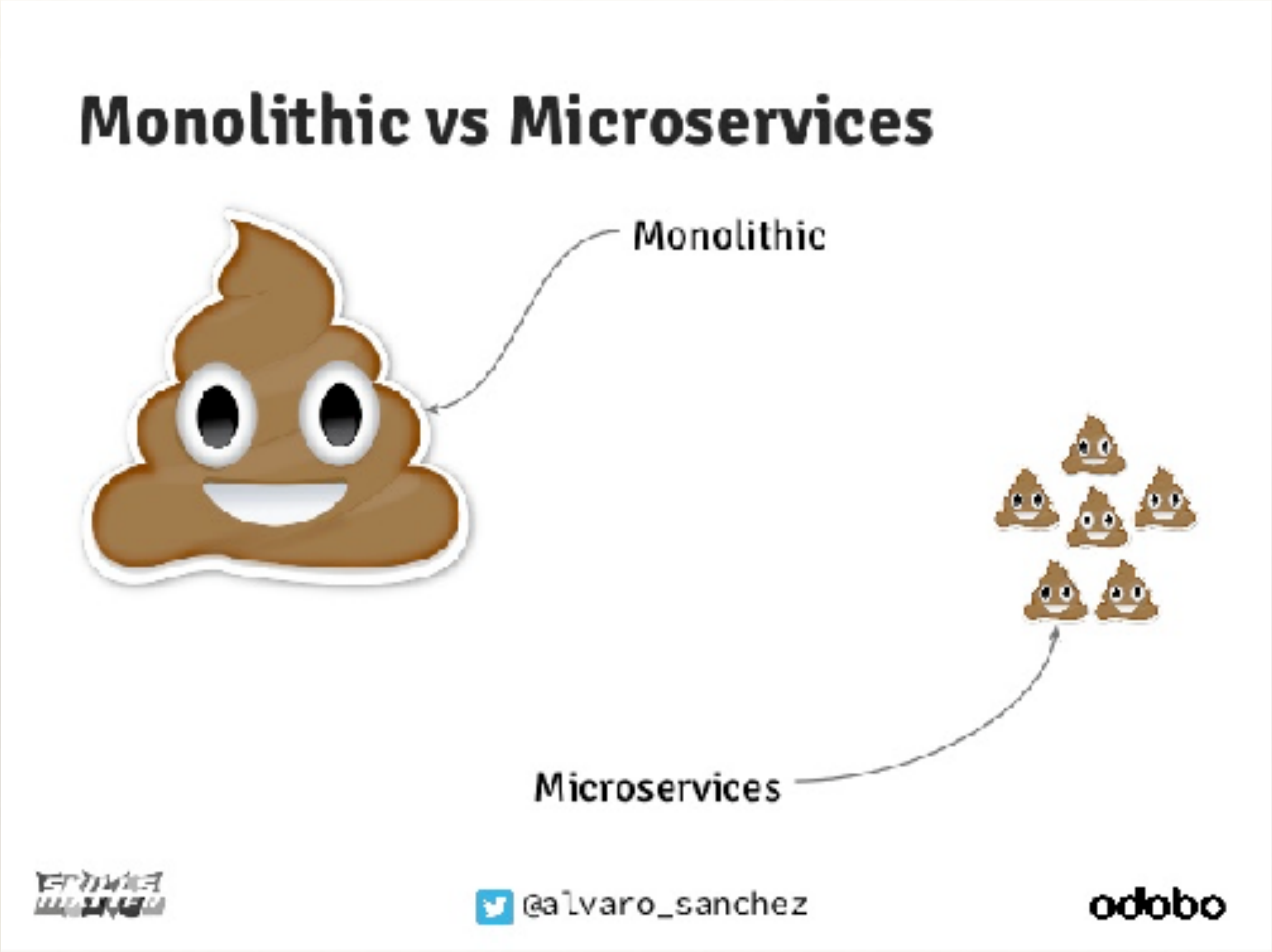
stg SOFTWARE TECHNOLOGY GROUP

Problem

- One system, many clients
  - Desktop
  - Web
  - Mobile (2…3 flavors?). Try dozens…
  - IoT
  - 3rd party API

Problem

- Typical architecture - Monolith
- Large, complex code base
- Large re-deploys
- High Overhead
  - Lots of hardware (VMs)
  - Process bureaucracy
- Tight coupling
- Maintenance
- Inflexible



Monolithic vs Microservices

Microservices

“Anything not a monolith.”

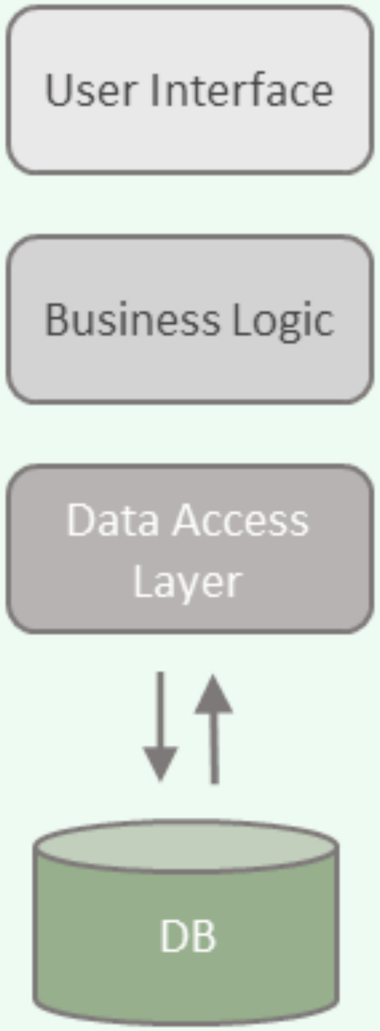
Microservices

- Architectural Style
- Evolution of SOA
- System Composition
    - Many services, one system

Microservices - cont.

- Micro == Discrete
- Service == Application
- Limited scope of responsibility
- Isolated - data, side effects

MONOLITHIC ARCHITECTURE



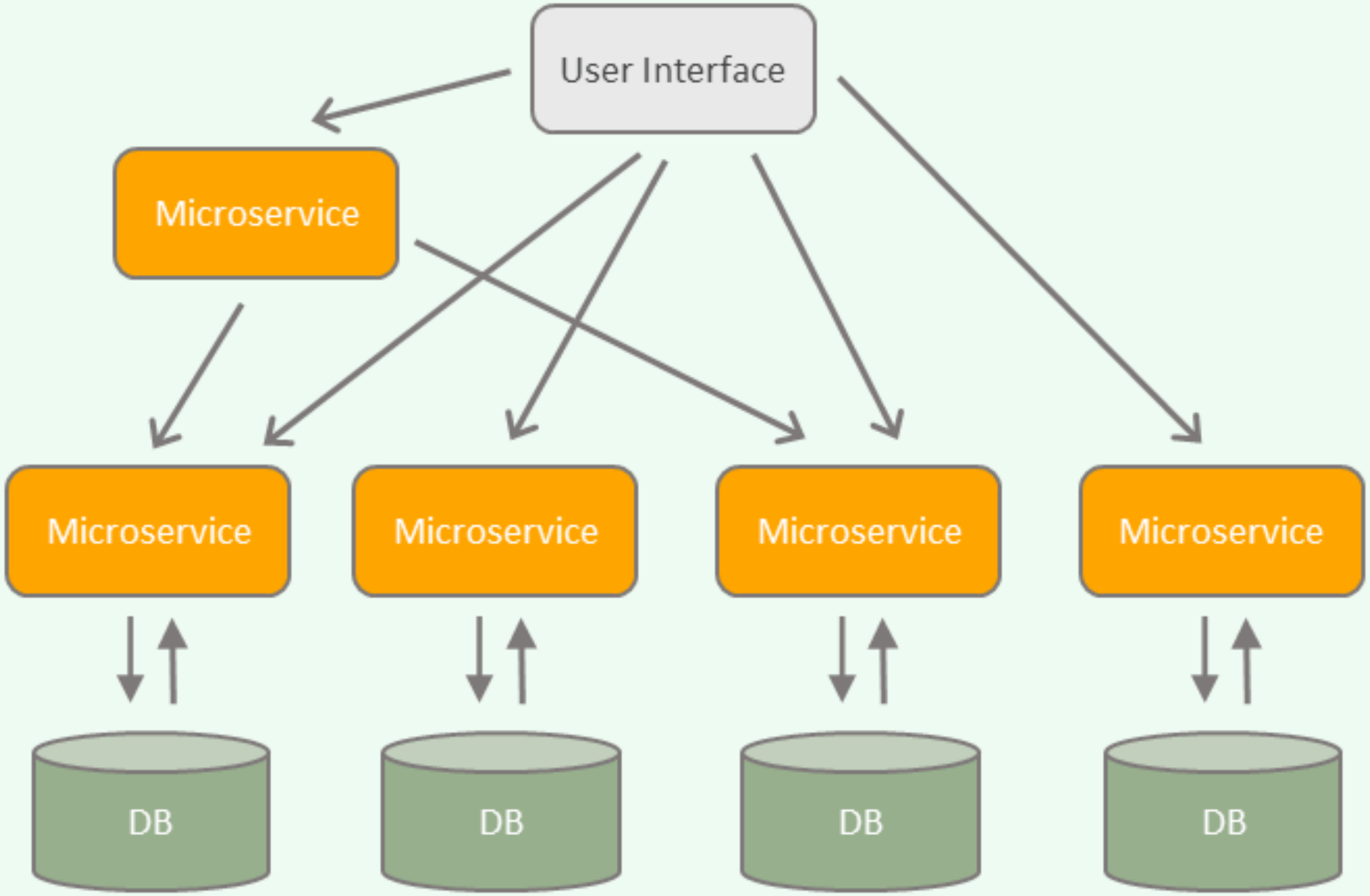
User Interface

Business Logic

Data Access Layer

DB

MICROSERVICES ARCHITECTURE



User Interface

Microservice

Microservice

Microservice

Microservice

Microservice

DB

DB

DB

DB

Goals

- Independent deployability
  - High Cohesion/Low Coupling
  - API Integrity - Don’t upgrade clients unless necessary
- Scalability
- Isolation
  - Own data store
  - Don’t affect other services if something goes wrong
- Fault tolerance

Goals - cont.

- Maintainability
  - Smaller scope, easier to understand
- Lower overhead
  - Deployment Process
- Technology/language agnostic

Flavors

- Point to Point
  - e.g. ReST, SOAP, XML-RPC, etc
  - Service Registry / Discovery
- Event Driven
  - Event Streams
  - Brokers

Adoption

- When (if?) are microservices a good fit?
    - Recurrent deploy issues
    - Fragile configuration
    - Scaling/performance issues
    - Flexibility
    - Team(s) ready

Challenges

- Complexity
- Inter-service communication
- Latency
- Data Consistency
- Distributed Transactions
- Versioning

Challenges

- Service Registration / Discovery
- Configuration
- Cluster management
- Monitoring
- Logging

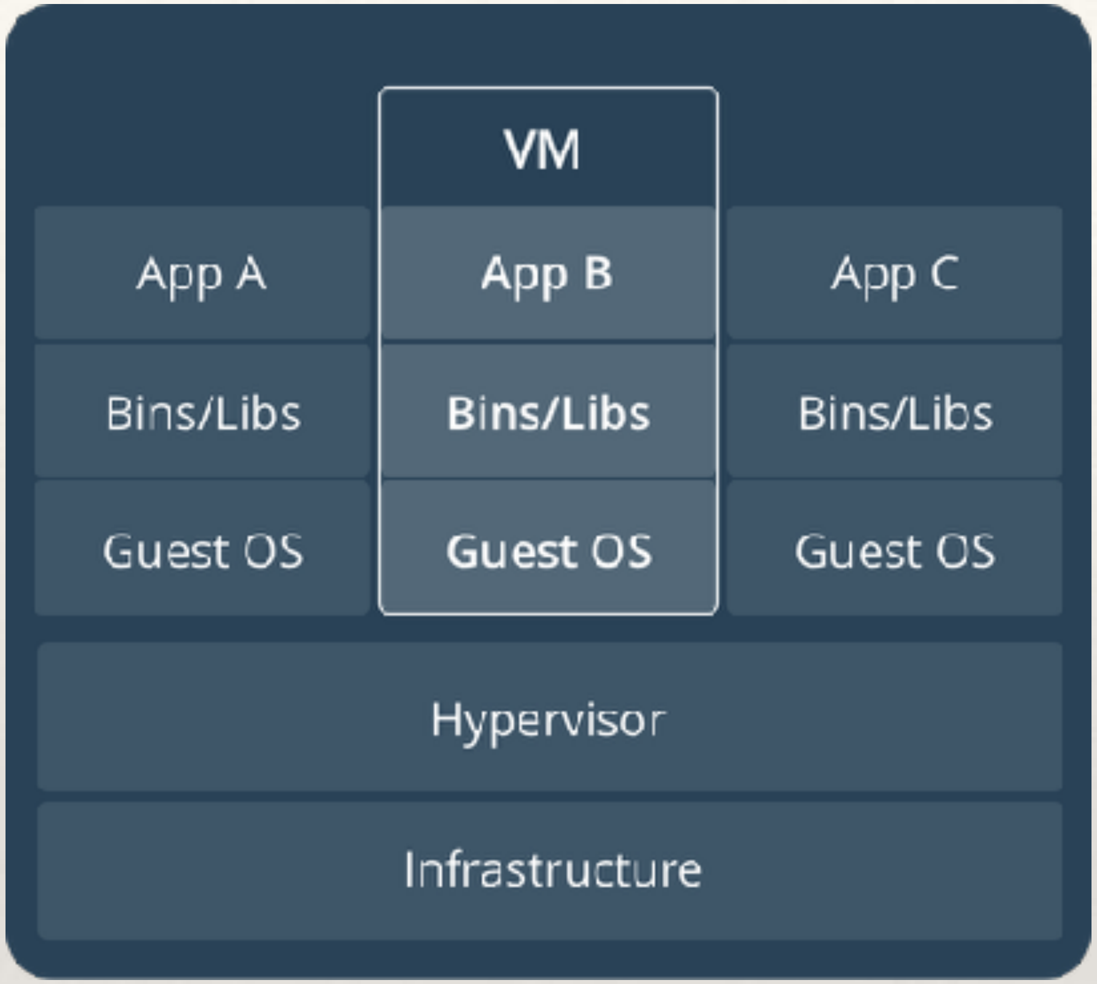
Challenges

- Logistics
  - Build
  - Deployment
  - Updates
  - Rollbacks
- Testing
- Training
- Culture

Docker

Docker Concepts

- Image
- Container
- Host
- Service
- Stack
- Docker Hub / Private Image Repository
- Docker Compose



VM

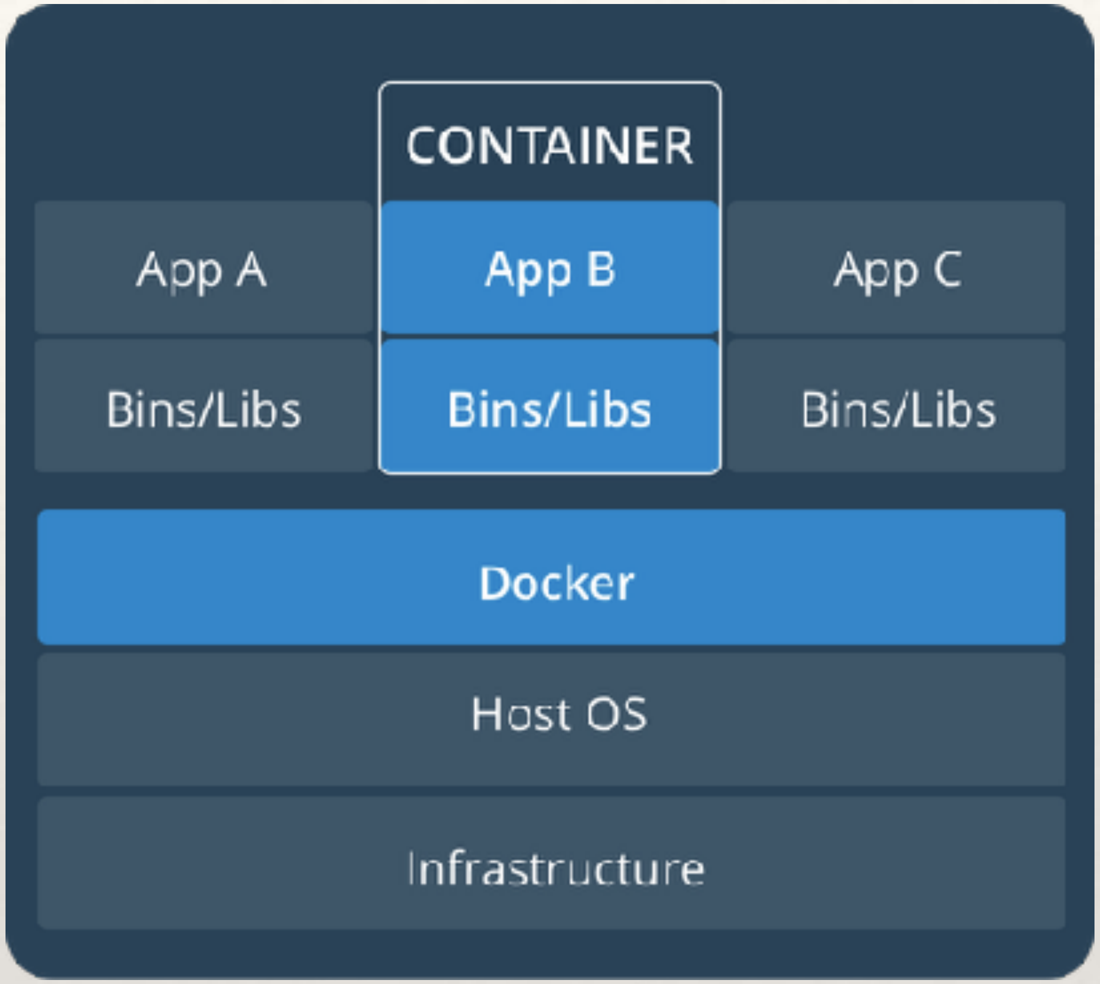
App A | App B | App C

Bins/Libs | Bins/Libs | Bins/Libs

Guest OS | Guest OS | Guest OS

Hypervisor

Infrastructure



CONTAINER

App A | App B | App C

Bins/Libs | Bins/Libs | Bins/Libs

Docker

Host OS

Infrastructure

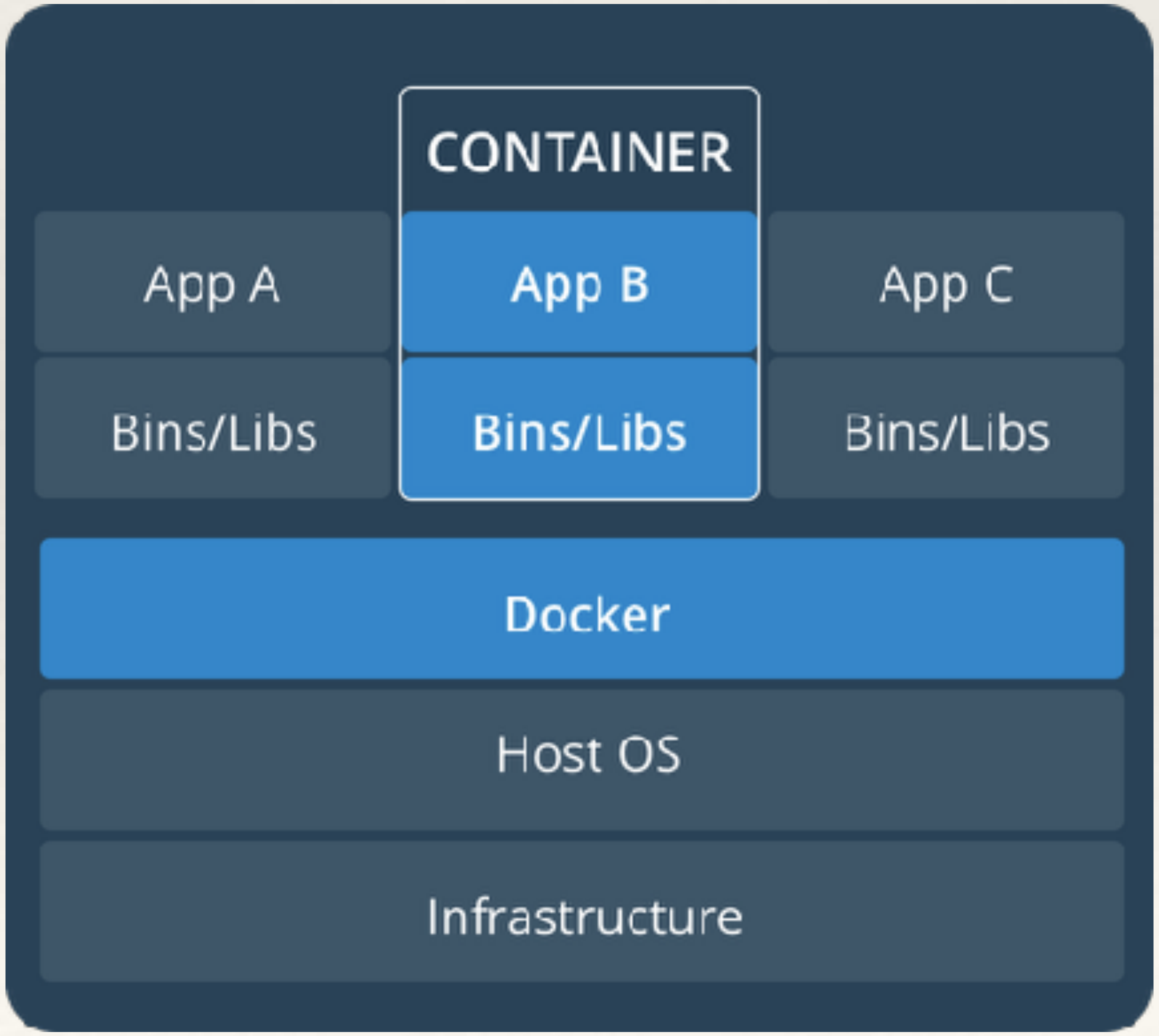
Docker Images

“A container image is a **lightweight**, **stand-alone**, executable package of a piece of software that includes everything needed to run it: **code, runtime, system tools, system libraries, settings**.”

- *Docker Website*

Docker Containers

A container is a running instance of an application defined by an image.



Docker Benefits

- Resource sharing
- Standardization
- Custom Images
- Isolation
- Clustering

Docker Benefits - Resource Sharing

- Single host, multiple containers
  - Shared OS resources
- Small, portable images
- Only necessary space, memory, supporting software to run the application
- Fast startup

Docker Benefits - Standardization

- Application packaging
- Deployment - same deploy commands, all apps
- Environments
  - Dev - provisioning, replicate prod env.
  - Test - consistent, replicated environments
  - Ops - uniform deployment
  - Enterprise - custom images, common tools
- Configuration

Docker Benefits - Custom Images

- Image Layering
  - Building blocks
- Licensing
- Complex configuration
- Portability
  - Reused images already pulled to host, just delta needed

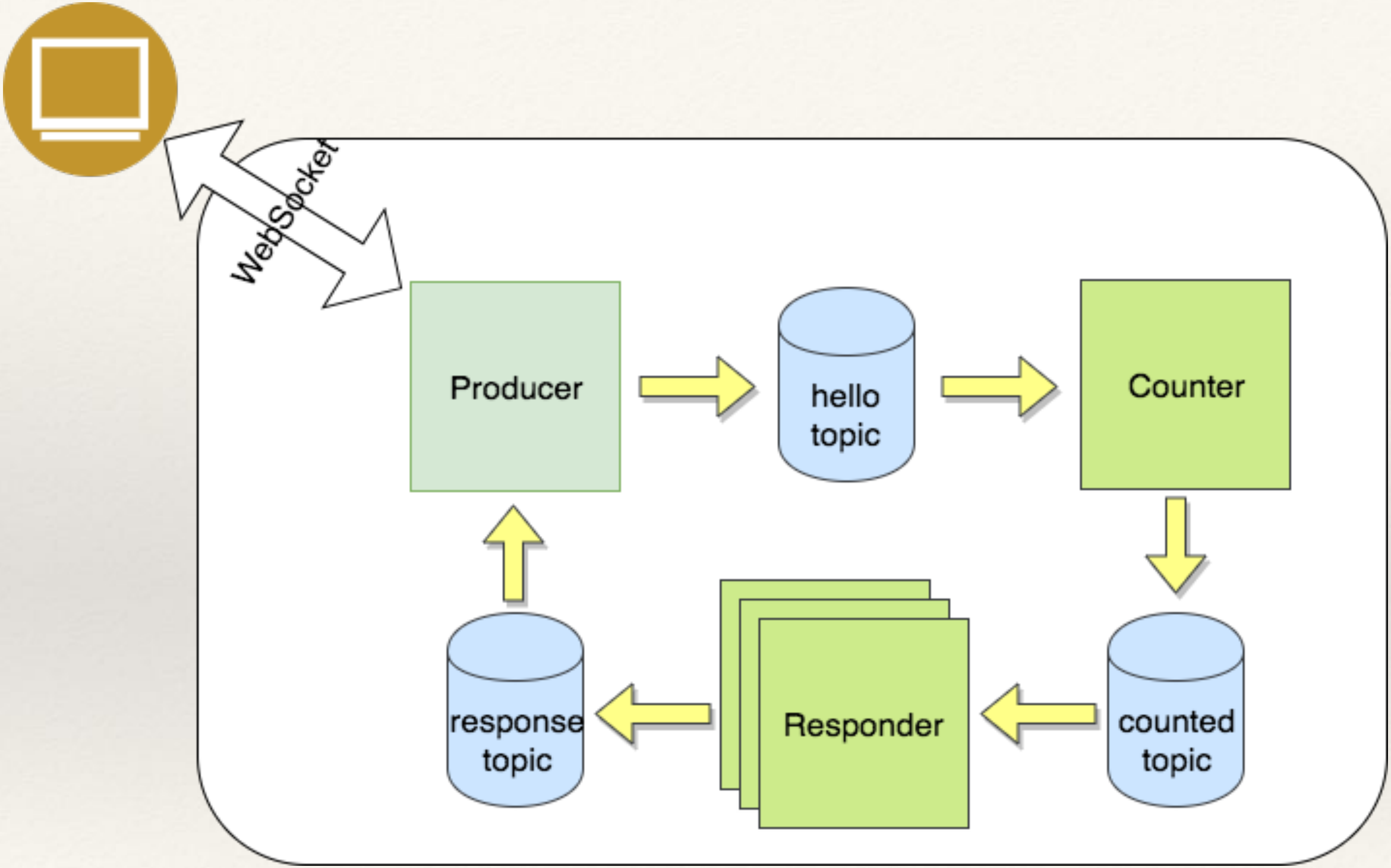
Docker Benefits - Isolation

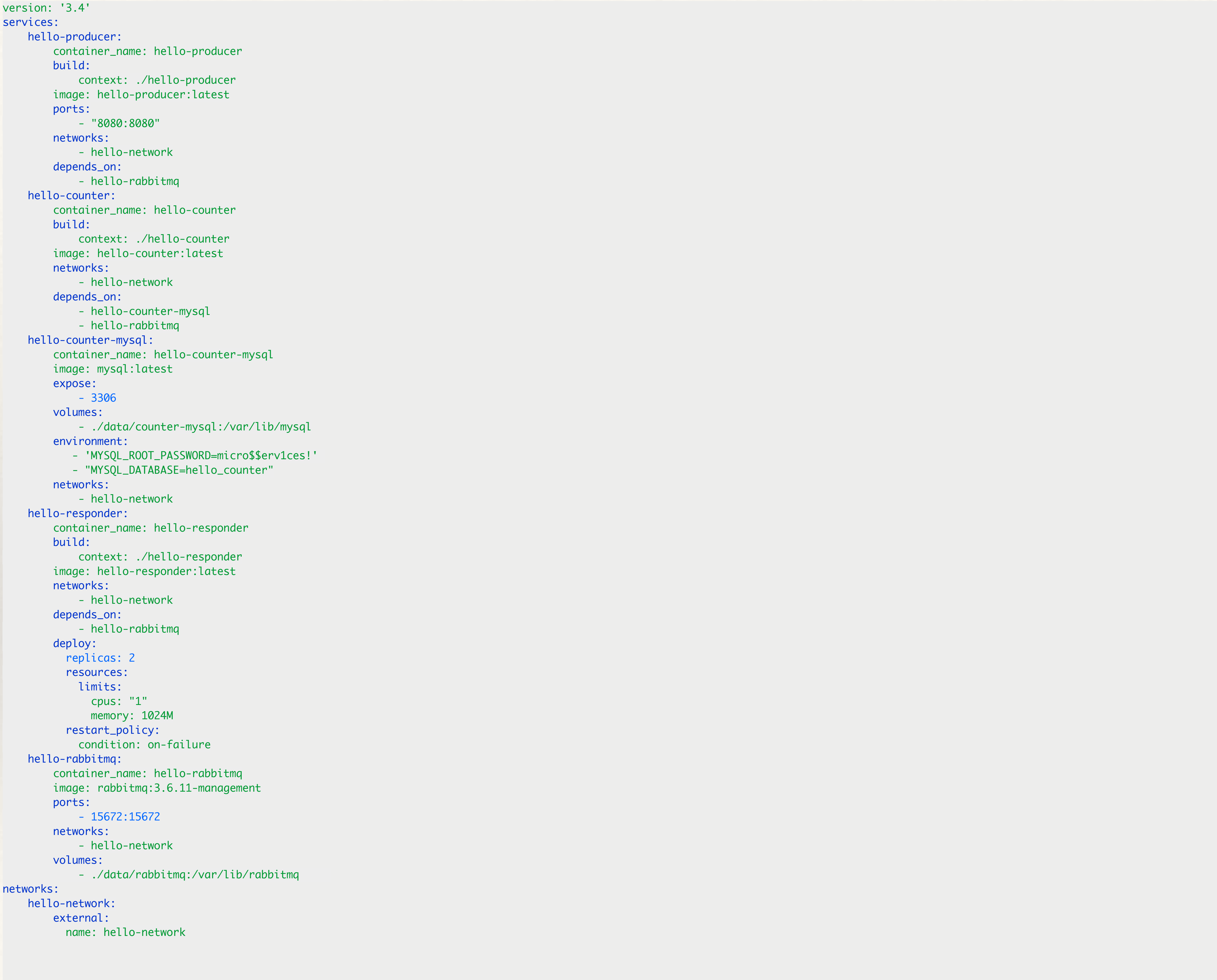
- “Do no harm”
- Closed networks
    - Security
    - Container DNS

Docker Benefits - Clustering

- Cloud platform support
  - AWS, Azure, Digital Ocean, etc
- Tool support
  - Swarm, Kubernetes, Apache Mesos, etc.

Docker Demo


WebSocket
Producer
hello topic
Counter
counted topic
Responder
response topic



```
version: '3.4'
services:
    hello-producer:
        container_name: hello-producer
        build:
            context: ./hello-producer
        image: hello-producer:latest
        ports:
            - "8080:8080"
        networks:
            - hello-network
        depends_on:
            - hello-rabbitmq
    hello-counter:
        container_name: hello-counter
        build:
            context: ./hello-counter
        image: hello-counter:latest
        networks:
            - hello-network
        depends_on:
            - hello-counter-mysql
            - hello-rabbitmq
    hello-counter-mysql:
        container_name: hello-counter-mysql
        image: mysql:latest
        expose:
            - 3306
        volumes:
            - ./data/counter-mysql:/var/lib/mysql
        environment:
           - 'MYSQL_ROOT_PASSWORD=micro$$erv1ces!'
           - "MYSQL_DATABASE=hello_counter"
        networks:
            - hello-network
    hello-responder:
        container_name: hello-responder
        build:
            context: ./hello-responder
        image: hello-responder:latest
        networks:
            - hello-network
        depends_on:
            - hello-rabbitmq
        deploy:
          replicas: 2
          resources:
            limits:
              cpus: "1"
              memory: 1024M
          restart_policy:
            condition: on-failure
    hello-rabbitmq:
        container_name: hello-rabbitmq
        image: rabbitmq:3.6.11-management
        ports:
            - 15672:15672
        networks:
            - hello-network
        volumes:
            - ./data/rabbitmq:/var/lib/rabbitmq
networks:
    hello-network:
        external:
          name: hello-network
```

Commands

```
git checkout https://github.com/sfransonstg/microservicesdocker.git
```

```
mvn clean package
```

```
docker-compose build
```

```
docker swarm init
```

```
docker network create -d overlay hello-network
```

```
docker stack deploy -c docker-compose.yml microservicesdocker
```

```
docker stack ls
```

```
docker service scale microservicesdocker_hello-responder=3
```

```
docker service ls
```

```
docker service logs -f microservicesdocker_hello-responder
```

```
docker stack rm microservicesdocker
```

Where to start?

- Team Dynamics
    - Maturity
        - Standards/Best practices
        - Code quality
    - Ownership vs "Not my problem"
- Start small
    - Internal app
    - Candidate services/code in monoliths

Where to start?

- Priority #1 - Infrastructure
    - Hosts
    - Cluster management, monitoring, logging, etc.
    - Process - creation, deployment, updating
- Priority #2 - Reduce friction
    - Make development as simple as possible
    - Know and use established patterns
    - Custom images
    - Shared artifacts (docker-compose files)
    - Clear documentation
    - Training

Q&A

Resources

- http://microservices.io/patterns/microservices.html
- https://thenewstack.io/microservices-standardization-moving-monolith-microservices/
- https://www.docker.com/what-docker
- https://www.digitalocean.com/community/tutorials/how-to-remove-docker-images-containers-and-volumes

Steve Franson
Java Practice Manager/Consultant
STG Consulting
steve.franson@stgconsulting.com
801.595.1000
https://www.linkedin.com/in/sfranson/

stg SOFTWARE TECHNOLOGY GROUP